DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Léo HENRY

There and back again : formal methods and model learning for real-time systems

Acting upon reality and learning from its reactions

Thèse présentée et soutenue à Rennes, le 03/12/2021
Unité de recherche : IRISA, équipe SUMO

Rapporteurs avant soutenance :

Étienne ANDRÉ	Professeur, LORIA Université Nancy
Frits VAANDRAGER	Full Professor, Radboud University Nijmegen

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Prénom NOM	Fonction et établissement d'exercice *(à préciser après la soutenance)*
Examinateurs :	Étienne ANDRÉ	Professeur, LORIA Université de Lorraine
	Frits VAANDRAGER	Full Professor, Radboud University Nijmegen
	Nicolas BASSET	Maître de Conférences, VERIMAG, U. Grenobles-Alpes
	Didier LIME	Professeur, LS2N, École Centrale de Nantes
	Delphine LONGUET	Maîtresse de Conférences, LRI, Univ. Orsay
	Stavros TRIPAKIS	Associate Professor, NorthEastern Univ. Boston (USA)
Dir. de thèse :	Nicolas MARKEY,	Directeur de recherche, CNRS, Université de Rennes
Co-dir. de thèse :	Thierry JÉRON,	Directeur de recherche, INRIA Rennes.

Remerciements

> Victory, after all, I suppose ! Well, it seems a very gloomy business.
>
> — J.R.R. Tolkien "The Hobbit"

J'ai pu lire quelque part qu'une thèse est un aquarium, et les remerciements des poissons rouges. J'aurais plutôt tendance à comparer la thèse à un marais, avec ses zones boueuses où l'on peut s'enfoncer, ses cours d'eau, certains vifs et décidés, d'autres réduits à de simples bras-morts que toute activité a abandonnée. Peut être aussi avec ses mangroves, se nourrissant avidement des courants marins qu'elles peuvent capter.

Logiquement, dans un tel environnement, les remerciements sont des piranhas. Ils se déplacent par bancs, sont typiques de l'image du milieu bien qu'ils n'en soient qu'une toute petite partie, et viennent se coller aux visiteurs pour les accueillir. Il paraît que cela a bon goût aussi. Certes, ce sont des saletés attirées par l'odeur du sang, mais bon dans une métaphore, on ne peut pas tout avoir !

Je tiens à remercier Thierry et Nicolas pour leur encadrement, et pour ne pas avoir sauté en marche lorsque j'ai proposé de défricher un vieux cours d'eau quelque peu effrayant loin des courants habituels. L'équipe SUMO en général pour son accueil et son ambiance, et notamment les permanents, Nathalie, Blaise, Loïc, Hervé, Ocan, Éric et Éric, pour leur aide, leurs points de vue et quelques très bonnes discussions scientifiques. En remerciant l'équipe je ne peux que laisser une place à part à Laurence et à son support si salutaire pour moi qui suit facilement submergé par des difficultés administratives (et d'autant plus émerveillé de les voir disparaître sans que le ciel ne me soit tombé sur la tête).

Merci aussi à Sophie et François, qui m'ont accompagné de bien des manières et sur le temps long, en réussissant souvent à trouver du temps qu'ils ne semblaient pas avoir. Pour m'avoir tous les deux trouvé une place dans d'enseignant dans leurs cours aussi, j'y ai beaucoup appris en patience (un peu) et en pédagogie (beaucoup).

Un grand merci aussi aux membres de mon jury de thèse, qui acceptent de prendre le temps de m'écouter, et particulièrement à Étienne et Frits, pour avoir lu ces pages et avoir proposé un bon nombre d'améliorations.

Je remercie bien sûr tous les non permanents du département que j'ai croisé, ainsi que les anciens de ma promo dont j'ai continué de partager le parcours. Particulièrement Emily, avec qui beaucoup de thé a été versé, et à qui je dois un merci particulier, pour avoir accepté de lire en avant-avant première les piranhas, ce qui mord un peu et gâche la surprise. Arthur, Tristan et Hugo également pour de nombreuses discussions (pas forcément toutes scientifiques). Victor et Choufi, pour la camaraderie. Mathias et Adrian, en particulier pour l'impressionnante énergie qu'ils réussirent à dispenser. J'ai aussi une pensée pour Nicolas, Lily et Camille, avec qui j'ai pu passer du bon temps, et échanger quelques bons mots. Clément enfin, pour beaucoup de bons moments et de nombreuses soirées de jeux permettant de sortir la tête de l'eau.

Ensuite je tiens à remercier ma famille, qui m'a beaucoup soutenu à travers cette thèse et les études précédentes, alors même que j'ai bien moins pu la voir pendant ce temps. En particulier mes parents, qui ont en plus fait l'effort méritoire de s'intéresser aux détails de ma thèse et de traverser quelques fois la France.

Enfin un grand merci à Solène, qui m'a supporté pendant tout ce temps, en première ligne et par tous temps.

Table of Contents

Résumé

Les ordinateurs, et plus généralement les systèmes automatiques, sont de plus en plus répandus à la fois dans notre vie courante (téléphones, logiciels de paie, jusque dans des choses aussi simples qu'une machine à café) et dans nos réalisations les plus complexes et avancées (fusées, médecine de pointe, analyse météorologique...). Les solutions aux problèmes de société, d'ingénierie ou scientifiques sont de plus en plus couramment recherchées avec l'aide de l'informatique, créant d'autant plus de défis pour les systèmes informatiques et au final, les informaticiens.

Conséquemment à cette situation une quantité impressionnante de systèmes liés à l'informatique existe, tant en terme de purs nombres que de taille ou de diversité. De par leurs interactions variées avec le monde, les systèmes informatiques acceptent des entrées de natures totalement différentes et s'attachent à des facteurs dissimilaires dans leurs calculs. Il peut s'agir de données quantitatives ou continues telles que le temps, le poids ou l'énergie, d'un environnement réactif, de probabilités, de changements de comportement brusques...

Les résultats attendus sont eux aussi de natures variables, telle que des décisions opérationnelles, le résultat de fonctions mathématiques ou de l'affichage.

Exemple. *Pensez à une entrée vidéo : une suite d'images se succédant rapidement dans le temps, couplée avec les informations de différents senseurs (vitesse, pression, température) et une voix qui doit être analysée depuis une piste audio et transformée en décisions et changements de comportement accompagnés d'une sortie vocale présentant les décisions.*

Ceci peut parfaitement correspondre à l'interface d'une machine-outil, d'un robot, ou (plus à la mode) d'une voiture autonome.

Afin de traiter ces situations et besoins variés, et à cause de la progression très active de la science informatique, les systèmes informatiques ont des principes et des structures extrêmement dissimilaires.

Remarque. *Cette dissimilarité est reflétée (et peut être même magnifiée) dans la variété des domaines et communautés de recherche en informatique, dont les formations, la méthodologie et même les méthodes de falsifiabilité varient grandement.*

Il faut ajouter à cela que les systèmes informatiques sont interconnectés par nature, créant des réseaux hautement plus complexes que la somme de leurs composants. Prédire le comportement d'un (composant d'un) système une fois interconnecté dans un tel environnement est donc devenu une tâche d'une épouvantable difficulté pour un humain seul. Cela met en lumière l'intérêt de méthodes et d'outils d'*analyse automatique*.

Pour certains de nos systèmes informatiques, une défaillance peut être relativement bénigne, et peut parfois être corrigée. Pensez par exemple à la corruption d'un (fragment d'un) pixel d'une image, ou à une machine à laver qui flanche. Néanmoins, pour un grand nombre de systèmes une erreur peut avoir un coût terrible, qu'il soit monétaire et affecte l'efficacité (un site internet ne répondant plus, la paralysie d'un réseau ferroviaire, des erreurs dans un logiciel de paie, un banque perdant des données...) ou en vies humaines (le le système de contrôle du frein d'un train n'arrivant pas à activer les freins, une machine-outil réalisant des mouvements imprévus en présence de travailleurs, la défaillance d'un pilote automatique sur un avion...). Prévenir des erreurs dans de tels systèmes est donc d'une importance cruciale, alors qu'ils sont souvent parmi les plus élaborés, ou immergés dans des environnements hautement complexes.

Exemple. *Un exemple assez connu d'une telle erreur est l'échec du lancement de la première fusée Ariane 5. Cette fusée a explosé moins d'une minute après son décollage à cause d'une exception matérielle déclenchée par un dépassement provenant de la conversion de l'accélération horizontale d'entiers à virgule flottante 64 bits en entiers signés 16 bits.*

Cette erreur, aussi simple qu'elle puisse paraître une fois découverte, a été causée par l'usage de connexions *légèrement inadaptées entre des composants matériels autrement corrects et la décision de ne pas protéger la valeur contre les conversions pendant le processus de création, laquelle découlait d'hypothèses basées sur la fusée Ariane 4. Elle était donc difficile à repérer et put passer inaperçue à travers de multiples vérifications, simulations et phases de test.*

Cet échec, heureusement non létal, causa la perte des quatre satellites transportés par la fusée, entraînant un déficit de plus de 370 millions de dollars états-uniens. Il devient connu comme l'erreur logicielle la plus coûteuse de l'Histoire (jusqu'alors) et joua un rôle important dans la reconnaissance et le développement des recherches assurant la fiabilité des composants critiques.

La conséquence de ces situations (dépendance toujours croissante envers les systèmes informatiques ; importante dissimilarité de buts et de moyens ; interconnection de systèmes ; existence de systèmes informatiques critiques pour la sécurité) est qu'un nombre toujours

croissant de systèmes informatiques sont créés et connectés entre eux et avec des composants plus anciens. Une défaillance de certains de ces systèmes pourrait avoir de désastreuses conséquences. Dans le même temps, l'analyse et le diagnostic ou la vérification à la (seule) main est devenu irréaliste.

Un besoin vital de modèles

Comme expliqué précédemment, la plupart des systèmes sont trop complexes ou trop grands pour être analysés directement par des humains, particulièrement les systèmes dont la fiabilité est la plus cruciale. Cela est dû à des facteurs variés, notamment le grand nombre de lignes de codes ; les nombreux langages informatiques différents qui peuvent être utilisés ; la variété des applications et des expertises associées ; l'interconnexion avec des composants écrits dans des langages historiques, à peine utilisés mais toujours fonctionnels ; l'interconnexion avec des composants propriétaires qui ne peuvent pas être complètement analysés ; la dépendance au matériel. . . entre autres facteurs. Pour cette raison, l'analyse automatique des systèmes est nécessaire et son domaine d'application ne cesse de s'étendre. Les méthodes formelles, c'est-à-dire les techniques mathématiquement rigoureuses pour la spécification et la vérification, constituent l'un des principaux candidats à une telle analyse. L'utilisation de méthodes mathématiquement correctes et la possibilité d'obtenir des résultats prouvés permettent une grande confiance dans l'analyse effectuée.

À cause de leur précision, les méthodes formelles peuvent être à la fois très complexes à utiliser et calculatoirement coûteuses. De ce fait, il est nécessaire d'utiliser des *abstractions appropriées* afin de concentrer l'expressivité sur les aspects importants d'un système, tout en simplifiant la tâche à accomplir.

Dans le cas des systèmes critiques, plusieurs types d'assurances complémentaires sont recherchées : la vérification formelle donne des preuves de propriétés de haut-niveau, mais dépend de modèles souvent très abstraits ; l'analyse de programmes travaille directement sur le code, qui doit donc être disponible, et dérive des propriétés souvent moins complexes et de type différent ; le test ne peut assurer de propriétés positives (par exemple qu'une erreur ne va pas arriver) mais peut interagir avec le produit final (incluant l'implémentation physique et l'environnement) ; la surveillance au moment de l'exécution peut rectifier ou arrêter les comportements défectueux au fur et à mesure qu'ils se produisent, mais ses options sont plus limitées (par exemple, elle doit parfois arrêter le système, ce qui peut être un problème).

Toutes ces méthodes nécessitent un modèle (pour des raisons d'efficacité et pour obtenir une représentation commune) et sont des exemples de l'action *des modèles sur la réalité* : elles utilisent des modèles pour raisonner sur un objet réel, agir sur eux (surveillance, test) et prendre des décisions à leur sujet (propriétés).

Pour cette raison, elles dépendent de l'existence d'un modèle adapté. Pourtant, les modèles formels de qualité, intégrant les bonnes informations et abstractions, sont rares dans la nature : il est donc utile de pouvoir créer des modèles à partir de systèmes réels.

Apprentissage : un levier pour obtenir des modèles

La plupart des méthodes mentionnées plus haut (notamment le contrôle, le test formel et la vérification) s'appuient sur un modèle mathématique du système et/ou des propriétés à vérifier.

Exprimer de tels systèmes et propriétés avec des modèles peut être difficile. Notamment, formaliser correctement une situation à la main nécessite un expert à la fois de la situation et du modèle, afin de prendre en compte les subtilités des deux. Même pour un tel expert, cela prend du temps et est sujet à de nombreuses erreurs

Pire encore il est parfois totalement impossible ou prohibitivement coûteux d'obtenir un modèle. Cela peut être dû à des composants non-observables, par exemple dans une approche boite-noire où il est seulement possible d'interagir avec le système (ou certains composants) sans observer sa dynamique interne (cela peut arriver par exemple pour les systèmes propriétaires) ; à des réseaux constitués d'un grand nombre de composants différents (cas le modèle d'un réseau est souvent d'une taille bien supérieure à celle des modèles de ses composants) ; à des systèmes ne pouvant être arrêtés pour analyse.

Plus généralement, certains algorithmes peuvent ou doivent utiliser des informations déraisonnables à demander à un expert du domaine (par exemple les probabilités de différents comportements du système lors de son utilisation).

Ces problèmes peuvent être résolus en *apprenant* un modèle pour un système ou une propriété à partir d'exemples de ses comportements (et parfois de comportements défectueux). Apprendre automatiquement de tels modèles à partir d'observations externes est l'objet de l'*apprentissage de modèles*, un sous-domaine de l'*apprentissage automatique*.

L'apprentissage de modèles est un processus allant *de la réalité vers les modèles* : il acquiert des observations de systèmes réels et les synthétise à l'aide d'un formalisme cible.

Entre modèles et réalité

Dans les sections précédentes, nous avons présenté la vérification formelle (c'est-à-dire l'action de modèles sur la réalité) et l'apprentissage de modèles (c'est-à-dire l'action de la réalité sur les modèles). Ces interactions ont d'importants bénéfices : les modèles dirigent les actions sur la réalité, ce qui aide à viser des comportements spécifiques, dont les observations peuvent être utilisées pour raffiner ou construire les modèles.

Au cœur de cette opération se trouve l'abstraction d'évènements réels par des formalismes mathématiques. Cette abstraction sous-tend la définition des modèles et aide à organiser l'apprentissage en visant des comportements d'intérêt. Par exemple, cela permet de détecter quelles observations sont utiles, et lesquelles correspondent à des parties déjà explorées de l'abstraction.

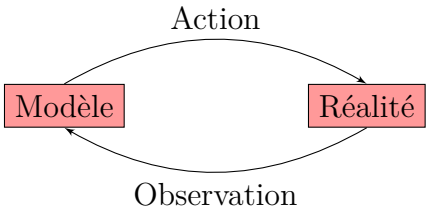


FIGURE 1 : Boucle entre modèles et réalité

Les interactions entre modèles et réalité forment une spirale vertueuse. Par exemple, les tests et l'apprentissage sont essentiellement le même processus observé sous des angles différents : tous deux créent des entrées à proposer à un système et observent ses actions. Lors des tests, le modèle utilisé pour générer ces entrées est supposé correct et le système est « en cours de test ». En apprentissage, le système est la référence, et nous visons à construire un modèle le décrivant. En pratique, les premières phases de test sont souvent utilisées pour détecter les erreurs de modélisation et affiner le modèle, ce qui peut être qualifié d'apprentissage. Pour une discussion plus approfondie sur ce parallèle, voir [Aic+18].

La boucle d'interaction générale entre les modèles et la réalité est esquissée dans la figure 1.

Cette boucle, aussi bénéfique qu'elle soit, vient avec ses propres défis, qui surgissent des différences entre des modèles mathématiquement parfaits et la réalité, où les mesures sont floues et les systèmes durs à contrôler et observer. Cela nécessite des méthodes *robustes* et prenant en compte la *contrôlabilité* et l'*observabilité* limitées d'un système.

La robustesse n'est pas abordée dans cette thèse, bien que la section 2.2.3 donne quelques indications bibliographiques. Elle revient à prendre en compte des erreurs de mesure (par exemple temps d'action d'un capteur qui induit une erreur sur la mesure du temps ; imprécision d'une information GPS sur la localisation) et des imprécisions de contrôle (par exemple, en robotique : différence entre le mouvement planifié et l'effet des moteurs ; pour le temps : un activateur qui peut agir à tout moment dans un intervalle de temps au lieu d'un instant précis où la commande a été donnée).

L'observabilité peut être gérée au niveau des modèles, en transformant un modèle avec des actions non observables (par exemple, les communications internes entre les composants) en un équivalent où toutes les actions sont observables. Un autre élément central de l'observabilité est la *distinguabilité* : un modèle peut-il faire la distinction entre deux observations, ou font-elles partie du même comportement ? Des comportements indiscernables peuvent être utilisés pour détecter des erreurs de modélisation ou pour diminuer le nombre d'observations à générer.

Des problèmes de contrôlabilité surviennent dans les systèmes où un contrôleur ou un testeur ne peut pas toujours appliquer un comportement ciblé. Dans ce cas, le contrôleur doit s'appuyer sur le système lui-même pour atteindre son objectif, et les méthodes correspondantes doivent caractériser ce qui est en dehors du contrôle de l'agent et planifier en conséquence.

Comme déjà mentionné, les abstractions sont au cœur de la boucle entre les modèles et la réalité. Concrètement, différents niveaux d'abstractions sont nécessaires :

- des exécutions des modèles à leurs traces observées par un agent extérieur, des abstractions de différents niveaux d'observations sont nécessaires, ajoutant de plus en plus d'informations sur l'état interne.

- d'exécutions uniques aux ensembles d'exécutions définissant les modèles, en passant par des ensembles minimaux non distinguables, différents niveaux de précisions sont utilisés pour analyser ou construire un modèle.

Dans le contexte de cette thèse, ces abstractions sont les plus importantes lorsque l'on discute des méthodes d'apprentissage ou des méthodes d'approximation d'état, car elles s'intéressent aux abstractions inverses, c'est-à-dire à la déduction d'exécutions de modèles à partir d'observations et à celle d'ensembles (infinis) d'exécutions à partir de quelques exemples. Un soin particulier sera donc apporté à travers tous les chapitres de cette thèse à la définition de ces différentes abstractions et de leurs relations.

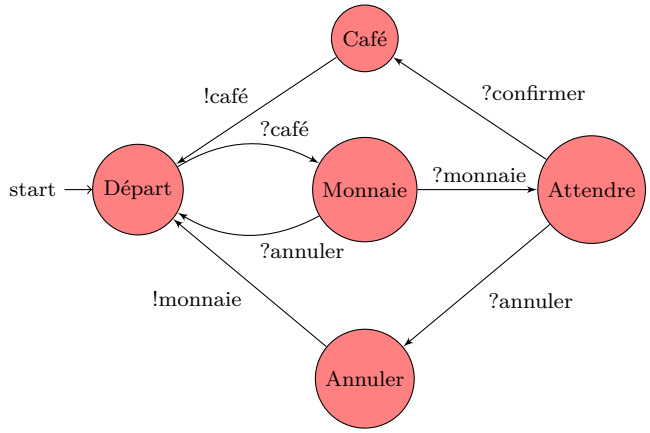


FIGURE 2 : Automate décrivant une simple machine à café.

Ajouter le temps au mélange

Le temps est un aspect important d'un grand nombre d'applications telles que les transports, la fabrication, la sécurité... où les contraintes temporelles sur les actions font partie intégrante du comportement recherché.

Exemple. *Considérons une simple machine à café, un objet banal sans doute critique pour le bon fonctionnement d'un laboratoire de recherche. Sa spécification (de haut niveau) pourrait être donnée par un automate simple, comme le montre la figure 2, avec des entrées données par l'utilisateur préfixées par "?" et les sorties de la machine préfixées par "!". Selon cette spécification, tout utilisateur demandant un café, le payant et confirmant l'achat est sûr de* finalement *recevoir un café. L'absence de toute contrainte temporelle rend ce résultat peu attrayant (personne n'apprécierait attendre des minutes ou des heures pour un café sans aucune garantie de temps). Des contraintes temporelles sont nécessaires pour encoder le comportement désiré.*

La prise en compte du temps apporte un épanouissement de nouveaux défis, notamment lorsque l'on considère que le temps est continu, ce qui est nécessaire pour obtenir une plus grande expressivité sur les contraintes temporelles. En particulier, le temps continu nécessite un nouveau modèle et de nouvelles abstractions, ainsi que des moyens de mettre en œuvre de telles représentations : encoder une valeur continue dans un système numérique est impossible, et l'approximer peut être hasardeux.

Les automates temporisés se sont imposés comme l'un des principaux modèles pour le temps continu, principalement grâce à l'intéressant compromis qu'ils représentent entre

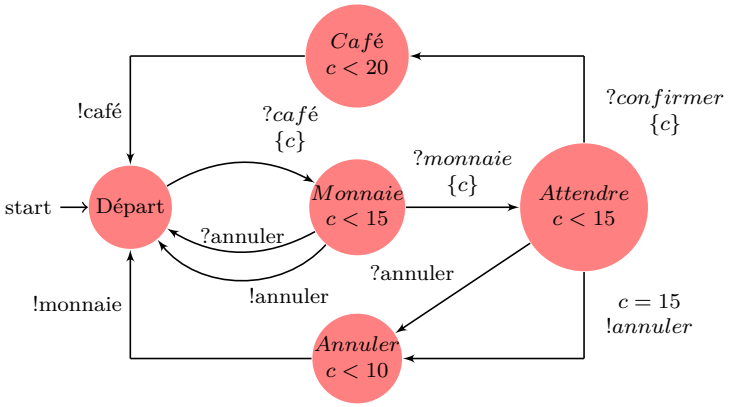


FIGURE 3 : Un automate temporisé spécifiant une machine à café ; temps en secondes.

efficacité et expressivité (ils sont parmi les modèles les plus expressifs dont l'accessibilité est décidable). Par conséquent, les travaux théoriques sur les automates temporisés sont à la fois étendus et applicables, en partie grâce aux multiples sous-classes d'automates temporisés qui ont été identifiées.

Exemple. *La figure 3 représente un automate temporisé spécifiant la machine à café de la figure 2. L'ajout d'une horloge (notée c) mesurant le temps écoulé et remise à* 0 *(ce que l'on note* $\{c\}$*) par certaines transitions permet de spécifier des contraintes de temps (en secondes pour cet exemple). Notez que, par rapport à la version non chronométrée de la spécification, il est possible de spécifier des délais d'attente comme c'est le cas avec les transitions étiquetées par* $!annuler$*. Cette spécification garantit qu'un utilisateur reçoit un café en moins de 20 secondes après avoir confirmé son achat.*

Dans cette thèse, nous nous concentrons sur des algorithmes et des structures de données prenant le temps en compte, avec les automates temporisés comme modèle de référence.

Contributions

Les contributions de cette thèse sont centrées sur les interactions entre modèles et réalité, dans le cas de systèmes temporisés modélisés avec des (variantes d')automates temporisés. Les principaux axes sont la *contrôlabilité*, l'*observabilité* et la *distinguabilité* des comportements. Un certain soin est également apporté à la disposition des différentes abstractions nécessaires à la poursuite des discussions techniques.

Le chapitre 1 présente les définitions centrales et les notions clefs qui sont ré-utilisées dans toutes les contributions. Le chapitre 2 détaille l'état de l'art des différents domaines de contribution. Les chapitres 3 à 5 détaillent les contributions.

Les principales contributions sont résumées ci-dessous :

- Le chapitre 3 présente nos travaux sur la contrôlabilité, appliqués aux automates temporisés de test de conformité (avec entrées et sorties). Il traite des jeux *difficiles* où aucune stratégie gagnante n'existe et discute d'un moyen de s'appuyer sur le système pour progresser – tout en minimisant cette dépendance.

  Plus précisément, lors du test de systèmes complexes, il n'est pas rare de faire face à des systèmes qui ne peuvent pas être complètement contrôlés par l'agent. Pourtant, bien que les cas de test soient modélisés comme des stratégies sur des arènes de jeu, le système n'est pas l'adversaire du testeur : il ne combat pas activement les objectifs du test mais a simplement les siens, qui peuvent ou non correspondre à ceux du testeur. Dans ce cadre, il est logique de discuter de la dépendance vis-à-vis des comportements du système (selon une certaine forme d'équité) tout en quantifiant cette dépendance. Cela se fait via une notion de rangs caractérisant la distance à un objectif d'accessibilité en termes de pertes de contrôle et de *stratégies de baisse de rang*, qui utilisent ces rangs pour progresser vers l'objectif.

  Ce travail a été présenté pour la première fois dans [HJM18] et est présentement soumis sous forme d'article de revue. La section 3.6 étend l'approche du chapitre et discute de l'utilisation des stratégies introduites précédemment à une configuration très générale où le recours à l'équité du système n'est pas suffisant pour assurer la victoire. Il n'est pas encore soumis.

- Le chapitre 4 propose une structure généralisant les automates temporisés et permettant de suivre l'état du système après une observation donnée dans des modèles partiellement observables – un problème que nous montrerons être à la fois difficile et central pour de nombreuses méthodes formelles, notamment de diagnostic, de test formel et de surveillance. Le cœur de la structure est d'obtenir une représentation symbolique pour les ensembles des configurations du modèle, qui est à la fois finiment représentable et close par les opérations habituelles, c'est-à-dire les actions observables et laissant le temps s'écouler avec d'éventuelles transitions inobservables. Nous montrons que cela peut être réalisé pour des automates temporisés avec une seule horloge, et posons les bases théoriques pour obtenir des résultats similaires

pour les automates temporisés généraux, bien que nous n'identifions pas la classe finiment représentable adéquat. Ce travail a été présenté dans [Bou+21] au cours de cette thèse, et d'abord développé dans [BJM18] avant elle.

- Le chapitre 5 généralise une méthode d'apprentissage actif antérieure à une sous-classe beaucoup plus large d'automates temporisés qui présentent une sémantique *inobservable* de haute dimension. Pour cette raison, des suppositions sur les différentes formalisations possibles des observations doivent être formulées, et écartées si nécessaire, grâce à une nouvelle notion que nous appelons *invalidité*.

  Plus précisément, nous étudions une sous-classe nouvellement introduite d'automates temporisés dont les remises à zéro des horloges ne sont pas liées aux observations. Il est donc nécessaire de *deviner* les remises à zéro de l'horloge. La notion d'invalidité caractérise les choix de remise à zéro des horloges qui ne peuvent pas expliquer l'ensemble actuel d'observations et peuvent donc être écartées en toute sécurité. Une version antérieure de ce travail a été présentée dans [HJM20] au cours de cette thèse, tandis que la version actuelle propose des algorithmes mis à jour et une discussion beaucoup plus approfondie sur l'invalidité.

Introduction

> Misery me! I have heard songs of many battles, and I have always understood that defeat may be glorious. It seems very uncomfortable, not to say distressing. I wish I was well out of it.
>
> —J.R.R. Tolkien "The Hobbit"

Computers and automated systems become ever more prevalent, both in our common life (smartphones, payroll softwares, even things as simple as coffee machines) and in our most complex and advanced realizations (rockets, state of the art medicine, weather analysis...). New societal, engineering or scientific obstacles are approached more and more often with the help of computers, creating all the more challenges for computer systems, and ultimately, computer scientists.

This situation leads to the existence of an impressive amount of computer related systems, both in term of sheer size, numbers and diversity. By interacting in disparate ways with the world, computer systems accept as input informations of completely different kinds, and take dissimilar factors into account in their computations. These can be quantitative or continuous data, such as time, weight or energy, a reactive environnement, probabilities, abrupt changes in behaviours...

The expected outputs are themselves of varying nature, such as operative decisions, results of (mathematical) functions or displays (*e.g.* image or video).

Example. *Think of a video input, which has images coming rapidly through time, coupled with different sensor informations (speed, pressure, temperature) and a voice that has to be analysed through an audio feed, and translated into decisions and changes in behaviour as well as an audio output of a voice presenting those decisions.*

This can very well correspond to the interface of a machine-tool, a robot, or, in more trendy fashion, an autonomous car.

To handle these different situations and needs, and because of the active progression in computer science, computer systems are extremely dissimilar in design and structure.

Remark. *This dissimilarity is reflected (and perhaps is easier seen) on the variety of the research domains and communities in computer science, which vary greatly in formalism, methodology, and even falsifiability.*

Furthermore, by nature computer systems are interconnected and the resulting networks are highly more complex than their original components. To predict the behaviour of a (component of a) system once interconnected with such an environment has hence become a horrendous task for a human to approach directly. This emphasizes the interest of *automated analysis* methods and tools.

For some of the computer systems we use, failure is relatively benign, and sometimes can be corrected. Think of a corruption of (part of a) pixel of an image, or a washing machine breaking. Yet for a great number of systems, errors have a terrible cost, either monetary and in efficiency (a website going down, a paralysis of a train system, a payroll software making mistakes, a bank loosing data...) or in human lives (control of a train break failing to activate the brake, machine tool making an unplanned movement around workers, auto-pilot failure on a plane).

Preventing errors in such systems is thus of the outmost importance, while they are often on the most elaborate end of the spectrum, or immersed in highly complex environments.

Example. *A somewhat well known example of such error is the launch failure of the first Ariane 5 rocket. That rocket exploded after less than a minute of flight because of a hardware exception triggered by an overflow originating from the conversion of horizontal acceleration from 64-bit floating points integer to 16-bit signed ones.*

That error, as trivial as it may seem once discovered, was due to the use of somewhat unadapted connections between *otherwise valid hardware components and the decision not to protect that value conversion during the design process, which resulted from assumptions based on the previous Ariane 4 rocket. Hence, it was intricate to discover, and was able to pass unnoticed through multiple verifications, simulations and test phases.*

That failure, fortunately not fatal, caused the waste of four satellites the rocket transported, resulting in a loss of more than US$370 millions. It became known as the most expensive software bug in history (so far) and played a great role in the recognition and development of research ensuring the reliability of safety-critical components.

The result of these situations—ever growing reliance on computer systems; great dissimilarity of purposes and means; interconnection of systems; existence of safety-critical computer systems—is that an ever increasing number of computer systems are created and

connected between themselves and with older components. For some of these systems a failure would have dire consequences. Yet analyzing, diagnosing or checking these systems (only) by hand has become unrealistic.

A dire need for models

As explained previously, most systems are too complex and/or too large to be analysed by humans, especially those whose reliability is the most required. This is due to different factors, notably the sheer number of lines of codes; the numerous different languages that can be used; the variety of applications and associated expertises; the interconnection with components written in historic languages, barely used anymore but still running in practice; the interconnection with proprietary components that cannot be completely analysed; the dependence on hardware... between others. Because of this, automatic analysis of systems is necessary, and its application range ever increasing. One of the main contenders for such analysis is formal methods *i.e.* mathematically rigorous techniques for the specification and verification. The use of sound methods and the possibility to derive proved results allows for a great confidence in the performed analysis.

Due to their precision, formal methods can be both very complex for an expert to design and computationally expensive to run. Hence, it is necessary to rely on *fitting abstractions* to concentrate the expressivity on the relevant aspects of a system, while simplifying the task at hand.

For critical systems, different kinds of complementary insurances are sought: formal verification leads to proofs of high-level properties, but relies on an often very abstract model; program analysis works directly on the code, which must be available and the properties derived are generally less complex and of different kind; testing cannot conclude to positive properties (*e.g.* a bug will not happen) but can interact with the final product (including the physical implementation and the environment); monitoring at run-time can rectify or stop faulty behaviours as they happen but is more limited in its options (*e.g.* sometimes has to stop the system, which can be a problem).

All these methods require a model of some sort (for efficiency and to obtain a common representation), and are examples of actions of the *models on reality*: they use models to reason upon real objects, act on them (monitoring, testing), and take decision about them (properties).

As such, they depend on the existence of a fitting model to be employed. Yet, good formal models integrating the correct informations and abstractions are rare in the wild: it is hence valuable to be able to *create models* from real life systems.

Learning: leveraging model availability

Most of the aforementioned approaches—notably control, formal testing and verification—rely on a mathematical model of the systems and/or properties to be verified.

Expressing such systems and properties with models can be difficult. Notably, correctly formalizing a situation by hand requires an expert in both the situation *and* the model, so as to take into account the subtleties of both. Even for such an expert, it is time consuming, and error prone.

Even worse, sometimes obtaining a model is completely impossible or prohibitively difficult. This can be due to unobservable components *i.e.* in a black-box approach where one can only interact with the system (or some components) without observing its internals (*e.g.* proprietary systems); networks constituted of a great number of different components (as model of the network is often of far greater size than the models of its components); system that cannot be taken offline for analysis.

More generally, some algorithms can or must use information that is unreasonable to ask from a domain expert (*e.g.* probabilities of different behaviours of the system in practice).

These problems can be solved by *learning* a model for a system or a property from examples of its behaviours (and sometimes of faulty behaviours). To automatically learn such models from external observations is the object of *model learning*, a sub-domain of *machine learning*.

Model learning is a process that typically goes *from reality to the models*: it acquires observations from real systems and synthesizes them using a targeted formalism.

Between models and reality

In the previous sections, we presented formal verification *i.e.* actions from the models upon reality and model learning *i.e.* actions from reality on models. These interactions

have important benefits: models direct the action on reality, which in turns helps to target specific behaviours whose observations can be used to refine or build them.

At the core of this interaction is the mathematical abstraction of real events. That abstraction underlies the definition of models, and helps to organize the learning toward behaviours of interest. For example, it allows to detect which observations are useful, and which correspond to abstractions already explored.

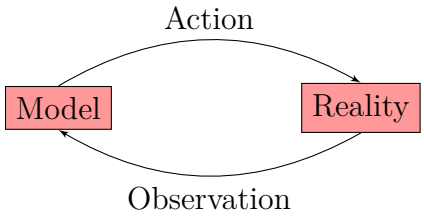


Figure 4: Reality-Model loop

The interactions between models and reality form a virtuous spiral. For example, testing and learning are essentially the same process, except observed from a different perspective: both create inputs to propose to a system and observe its actions. In testing, the model used to generate these inputs is supposed correct, and the system is under test. In learning, the system is the reference, and we aim at constructing a model describing it. In practice, the first testing phases are often used to detect modeling errors and refine the model, which can be argued to be learning. For a more in depth discussion on this parallel, see [Aic+18].

The general interaction loop between models and reality is sketched in Figure 4.

This loop, as beneficial as it is, comes with its own challenges, which arise from the differences between mathematically perfect models and reality, where measures are fuzzy, and systems hard to control and observe. This calls for methods that are *robust*, and take into account the limited *controllability* and *observability* of a system.

Robustness is not addressed in this thesis, although Section 2.2.3 gives some bibliographic pointers. It boils down to taking into account errors in measures (*e.g.* reaction-time of a captor that induces an error in the measure of time; imprecision of a GPS information on localisation) and imprecisions in the control (*e.g.* in robotics: difference between the planned movement and the effect of the motors; for time: an activator which can act at any point in an interval of time instead of a precise instant where the command was given).

Observability can be handled at the level of the models, by transforming a model with unobservable actions (*e.g.* internal communications between components) into an

equivalent one where all actions are observable. Another central element of observability is *distinguishability*: *i.e.* can a model distinguish between two observations, or are they part of the same behaviour. Undistinguishable behaviours can be used to detect modeling errors or to diminish the number of observations to generate.

Controllability issues arise in systems where a controller or a tester cannot always enforce a targeted behaviour. In this case, the controller has to rely on the system itself to achieve its goal, and corresponding methods should characterize what is outside of the control of the agent, and plan around it.

As already mentioned, abstractions are at the core of the loop between models and reality. Specifically, different levels of abstractions are necessary:

- from model executions to their traces observed by an exterior agent, abstractions of different levels of observations are necessary, adding more and more information about the internal state.
- from single executions to sets of executions defining the models, passing by minimal undistinguishable sets, different levels of precisions are used to analyse or construct a model.

In the context of this thesis, these abstractions are most important when discussing learning methods or state approximation methods, as these are interested in *inverse abstractions i.e.* deducing model executions from observations and (infinite) sets of executions from few examples. A special care will thus be given through all chapters of this thesis to the definition of these different abstractions and their relations.

Adding time to the mix

Time is a significant aspect in a number of applications—such as *e.g.* transportation, manufacturing, security—where time constraints on actions are part of the intended behaviour.

Example. *Consider a simple coffee machine, a mundane object arguably critical for the good functioning of a research laboratory. Its (high level) specification could be given through a simple automaton, as depicted in Figure 5, with inputs given by the user prefixed by "?" and outputs of the machine prefixed by "!". Following this specification, any user asking for a coffee, paying it and confirming the purchase is ensured to obtain a coffee* eventually.

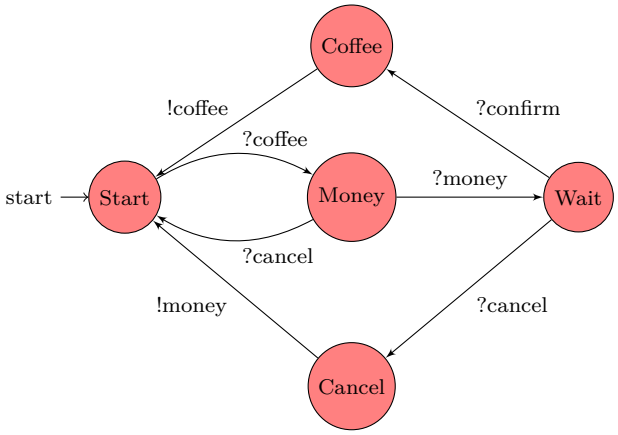


Figure 5: Automaton describing a simple coffee machine.

The lack of any timing constraint makes that result unappealing (no one would appreciate waiting minutes or hours for a coffee without any timing guarantees). Time constraints are necessary to encode the intended behaviour.

Taking time into account brings a flourish of new challenges, notably when considering that time is continuous, which is necessary to obtain a greater expressivity on time constraints. In particular, continuous time requires a new model and abstractions, as well as ways to *implement* such representations: encoding a continuous value in a digital system is impossible, and approximating it can be hazardous.

Timed automata have imposed themselves as one of the main models for continuous time, mainly because of the interesting compromise they represent between tractability and expressivity (they are the among the most expressive models for which reachability is decidable). Hence, theoretical work on timed automata are both far reaching and applicable, thanks in part to the multiple sub-classes of timed automata that have been identified.

Example. *Figure 6 represents a timed automaton specifying the coffee machine from Figure 5. The addition of a clock (noted c) measuring the time elapsing and reset to* 0 *(noted* $\{c\}$*) by some transitions allows to specify time constraints (in seconds for this example). Notice that, compared to the untimed version of the specification, it is possible to specify time-outs as done with the transitions labeled by* !*cancel*.

This specification ensures that a user gets a coffee in less than 20 secondes after confirming its purchase.

In this thesis, we focus on algorithms and data structures that take time into account, with timed automata as a reference model.

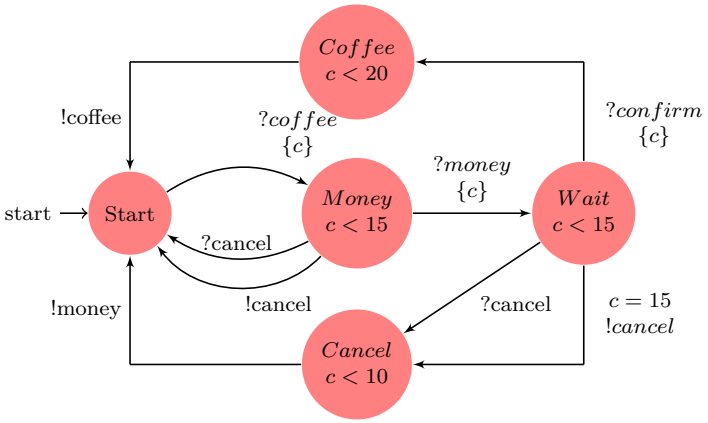


Figure 6: A timed automaton specifying a coffee machine with time in seconds.

Contributions

This thesis contributions are centered on the interactions between models and reality, in the case of timed systems modeled with (variants of) timed automata. The main focuses are on *controllability*, *observability* and *distinguishability* of behaviours. A certain care is also given to the lay out of different abstractions necessary to carry on the technical discussions.

Chapter 1 presents the central definitions and key notions that are re-used in all contributions. Chapter 2 details the state of the art of the different domains of the contributions. Chapter 3 to 5 detail the contributions.

The main contributions can be summarized in the following way:

- Chapter 3 presents our work on controllability, applied to conformance testing timed automata (with inputs and outputs). It deals with *difficult* games where no winning strategy exists and discusses a practical way to rely on the system for progress—while minimizing that reliance.

  More precisely, when testing complex systems, it is not unusual to face systems that cannot be completely controlled by the agent. Yet, although test cases are modeled as strategies on game arenas, the system is not the tester adversary: it does not actively fight the test objectives but simply has its own, that may or may not align with the tester's. In this framework, it makes sense to discuss reliance on the system behaviours (according to some fairness) while quantifying that reliance. This is done via a notion of *ranks* characterizing the distance to a reachability objective in term of control losses and *rank-lowering* strategies, that use these ranks to progress toward

the objective.

This work was first presented in [HJM18], and is now submitted as a journal paper. Section 3.6 extends the approach of the chapter, and discusses the use of the strategies introduced previously to a very general setup where reliance on the fairness of the system is not enough to ensure victory. It is not yet submitted.

- Chapter 4 proposes a structure generalizing timed automata and allowing to track the state of the system after a given observation in partially observable models—a problem that we will show to be both difficult and central for numerous formal methods, notably diagnostic, formal testing and monitoring. The core of the structure is to obtain a symbolic representation for sets of the model configurations, that is both finitely representable and closed by the usual operations *i.e.* observable actions and letting time elapse with possible unobservable transitions. We show that this can be achieved for timed automata with one clock, and lay the theoretic bases to obtain similar results for general timed automata, although we do not identify the fitting finitely representable class. This work was presented in [Bou+21] during this thesis, and first developed in [BJM18] before it.

- Chapter 5 generalizes a previous active learning method to a far broader subclass of timed automata that crucially displays a high-dimensional *unobservable* semantics. Because of this, guesses on different possible formalizations of the observations have to be formulated, and ruled out when necessary, through a new notion that we call *invalidity*.

  More specifically, we study reset-optional event recording automata, a newly introduced subclass of timed automata whose clock resets are not tied to observations. It is hence necessary to *guess* the clock resets. Invalidity characterizes clock resets that cannot explain the current set of observations and can thus be safely discarded. An earlier version of this work was presented in [HJM20] during the course of this thesis, while the current version proposes updated algorithms and a much more thorough discussion on invalidity.

Publications

The publications realized during this thesis are as follows:

- Léo Henry, Thierry Jéron, and Nicolas Markey, "Control strategies for off-line testing of timed systems", *in*: *25th International Symposium on Model-Cherking Software (SPIN'18)*, ed. by María-del-Mar Gallardo and Pedro Merino, vol. 10869, Lecture Notes in Computer Science, Springer, June 2018, pp. 171–189, DOI: 10.1007/978-3-319-94111-0_10
- Léo Henry, Thierry Jéron, and Nicolas Markey, "Active learning of timed automata with unobservable resets", *in*: *18th International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'20)*, ed. by Nathalie Bertrand and Nils Jansen, vol. 12288, Lecture Notes in Computer Science, Springer, Sept. 2020, pp. 144–160, DOI: 10.1007/978-3-030-57628-8_9
- Patricia Bouyer et al., "Diagnosing timed automata using timed markings", *in*: *International Journal on Software Tools for Technology Transfer* 23 (Mar. 2021), DOI: 10.1007/s10009-021-00606-2

Chapter 1

Definitions

All the same, I should like it all plain and clear. Also, I should like to know about risks, out-of-pocket expenses, time required and remuneration, and so forth. What am I going to get out of it? and am I going to come back alive?

— J.R.R. Tolkien "The Hobbit"

1.1 Time representation

In the rest of this thesis, time is modelled using *clocks*, that measure the amount of time elapsed since an arbitrary origin, and are considered *perfect* (no error appears in the stored value, reading them is instantaneous and does not introduce errors) and *infinitely precise*. These clocks will later be reset by timed automata, as described in Section 1.2.

Time is measured in arbitrary *time units* abstracting the level of precision required, that vary from application to application. The value of time units is an important parameter in the construction of models, as we will only compare time to *integer* numbers of those time units.

Example 1.1.1. *If one considers a computer-based system, the time unit can be the computer tick, or any multiple of it. In an interactive system with sensors and activators, the time unit size can be dictated by the precision of those components. In general, determining the time unit is a compromise between precision, that allows more expressivity, and efficiency (most applications having an exponential time dependency on the maximal number of time units considered by a model).*

Formally, given a finite set C of variables named clocks, a *valuation* of C is a function

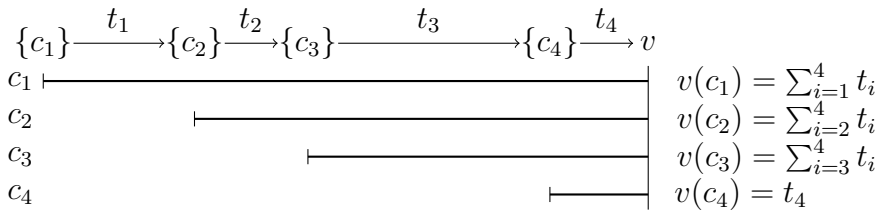


Figure 1.1: Clocks encoding the timed elapsed since their last reset in a valuation v.

$v : C \to \mathbb{R}_{\geq 0}$ associating each clock with its positive value. We write $\mathbf{0}$ for the initial valuation associating 0 to each clock. For any $t \in \mathbb{R}_{\geq 0}$ we write $v + t$ the valuation such that for all $c \in C$, $(v + t)(c) = v(c) + t$. For $C' \subseteq C$ and a valuation v, we write $v_{[C' \leftarrow 0]}$ the valuation mapping $c \in C$ to 0 if $c \in C'$ and $v(c)$ otherwise. Finally, for a valuation v, we define its *future* as $\overrightarrow{v} = \{v + t \mid t \in \mathbb{R}_{\geq 0}\}$.

Remark 1.1.2. *A set of clocks C and its valuations can be used to measure the time elapsed since a maximum of $|C|$ passed events, using resets $v_{[c \leftarrow 0]}$ with $c \in C$ to set the value of c to zero at the time of an event and $v + t$ to measure the passing of time, as depicted in Figure 1.1.*

Remark 1.1.3. *Valuations can seamlessly be considered as vectors in $\mathbb{R}_{\geq 0}^{|C|}$. We will sometimes abuse the notation and write $v \in \mathbb{R}_{\geq 0}^{|C|}$ when it eases the discussion.*

A *clock constraint* is an expression of the form $c_1 - c_2 \sim n$ or $c_1 \sim n$, for $c_1, c_2 \in C$, $\sim \in \{<, \leq, =, \geq, >\}$ and $n \in \mathbb{Z}$.

A valuation v satisfies a constraint $c_1 - c_2 \sim n$ (resp. $c \sim n$) if and only if $v(c_1) - v(c_2) \sim n$ (resp. $v(c) \sim n$).

We call *zone* over C [DT98] any finite conjunction of such constraints, and write $\mathcal{Z}_C$ (or sometimes simply $\mathcal{Z}$) for the set of zones over C. Given a valuation v and a zone z, we write $v \models z$ when v satisfies all the constraints in z.

Remark 1.1.4. *As for valuations, a zone can be considered as a set of $\mathbb{R}_{\geq 0}^{|C|}$ defined by the valuations satisfying its constraints. This justifies notations such as $v \in z$. We will sometime identify a zone and its set of valuations when it eases the discussion, as done below.*

Given a zone z, we define its future $\overrightarrow{z} = \bigcup_{v \in z} \overrightarrow{v}$ and the effect of resetting a subset $C' \subseteq C$ $z_{[C' \leftarrow 0]} = \{v_{[C' \leftarrow 0]} \mid v \in z\}$. Notice that these are also zones.

We call *guard* over C any finite conjunction of clock constraints of the form $c \sim n$, and write $\mathcal{G}_C$ (or simply $\mathcal{G}$) for the set of guards. As guards are zones, we extend every operation and definition made on zones to guards.

Example 1.1.5. *In Figure 1.2 a representation of the valuations for the set of clocks* $C = \{c_1, c_2\}$ *is displayed.*

The valuation $\mathbf{0}$ *is displayed with the result of some operations:* $v_1 = \mathbf{0} + 0.7$, $v_2 = v_{1[\{c_2\} \leftarrow 0]}$ *and* $v_3 = v_2 + 0.5$.

The leftmost highlighted polygon is the zone defined as $1 \leq c_1 \leq 2 \wedge -1 \leq c_1 - c_2 \leq 1$ *while the rightmost is the guard defined as* $c_1 \geq 2$.

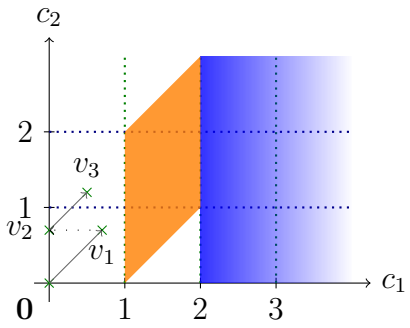


Figure 1.2: Two zones including a guard, and some valuations.

1.2 Timed automata

Now that a representation of time is defined with basic operations, automata handling this representation can be introduced.

Definition 1.2.1. *[AD94] A* timed automaton *(TA) is a tuple* $\mathcal{A} = (\mathcal{L}, \ell_{init}, \Sigma, C, \mathcal{I}, E)$ *such that:*

- $\mathcal{L}$ *is a finite set of* locations*, and* $\ell_{init} \in \mathcal{L}$ *is the* initial location*;*
- Σ *is a finite alphabet of discrete actions;*
- C *is its finite set of clocks;*
- $\mathcal{I} : \mathcal{L} \to \mathcal{G}_C$ *is a function associating an* invariant *to each location;*

- $E \subseteq \mathcal{L} \times \mathcal{G}_C \times \Sigma \times 2^C \times \mathcal{L}$ *is a set of transitions. For a transition* $e = (\ell, g, a, C', \ell')$, *we call* $g = \mathsf{guard}(e)$ *its guard,* $a = \mathsf{act}(e)$ *its action,* $C' = \mathsf{reset}(e)$ *its reset,* $\ell = \mathsf{src}(e)$ *its source and* $\ell' = \mathsf{tgt}(e)$ *its target. We extend these notations to sequences of transitions with* $\mathsf{src}((e_i)_{1\leq i\leq n}) = \mathsf{src}(e_1)$ $\mathsf{tgt}((e_i)_{1\leq i\leq n}) = \mathsf{tgt}(e_n)$ *and* $\mathsf{act}((e_i)_{1\leq i\leq n}) = (\mathsf{act}(e_i))_{1\leq i\leq n}$.

In a timed automaton $\mathcal{A}$ we write $K^{\mathcal{A}}(c)$ to denote the maximal constant of $c \in C$, *i.e.* the maximal constant to which c is compared in $\mathcal{A}$.

Remark 1.2.2. *Notice that a clock which is never tested can safely be removed from a timed automaton. If one wants to define the maximal constant for such a clock, a recurring convention is to use* 0.

We call maximal constant of $\mathcal{A}$, denoted $K^{\mathcal{A}}$, the maximum of the $K^{\mathcal{A}}(c)$ for $c \in C$. A *partial path* $\pi = (e_i)_{1\leq i\leq n}$ is a finite sequence of consecutive transitions (*i.e.* for all $1 \leq i < n$, $\mathsf{src}(e_{i+1}) = \mathsf{tgt}(e_i)$). A *path* is a partial path starting in ℓ_{init}.

Example 1.2.3. *A simple timed automaton is depicted in Figure 1.3. In this automaton,* $\mathcal{L} = \{\ell_{init}, \ell_1, \ell_2\}$, $\Sigma = \{a, b\}$, $C = \{c_1, c_2\}$, $K(c_1) = 2$ *and* $K(c_2) = 3$. *The invariants and transitions are represented in the figure.*

Two paths of this automaton are $\pi_1 = e_1 \cdot e_3$ *and* $\pi_2 = e_1 \cdot e_2 \cdot e_1 \cdot e_3$.

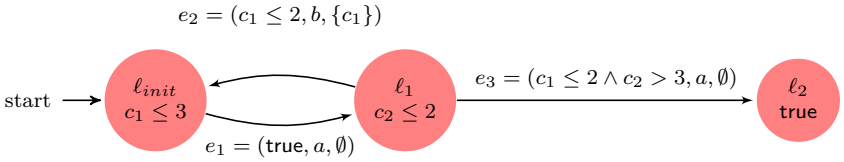


Figure 1.3: A simple timed automaton.

Because clocks take real values, and the availability of a transition depends on valuations (as the clock values have to satisfy the transition guard to enable it), the internal state (that we call *configuration*) of a timed automaton is the pair of its location and the current clock valuation. Thus, timed automata are a finite representation of infinite-state transition systems, that we use to express their semantics.

Definition 1.2.4. *With a TA* $\mathcal{A} = (\mathcal{L}, \ell_{init}, \Sigma, C, \mathcal{I}, E)$, *we associate the infinite-state transition system* $\mathcal{T}^{\mathcal{A}} = (S, s_{init}, \Gamma, \to)$ *where*

- $S = \{(\ell, v) \in \mathcal{L} \times \mathbb{R}^C_{\geq 0} \mid v \models \mathcal{I}(\ell)\}$ *is the set of* configurations *of the timed automaton and* $s_{init} = (\ell_{init}, \mathbf{0})$ *is the* initial configuration*;*

- $\Gamma = \mathbb{R}_{\geq 0} \uplus E$ *is the set of* transition labels*;*
- $\rightarrow \subseteq S \times \Gamma \times S$ *is the transition relation defined as the union of*
  - *the* delay *transitions, which contains all triples*

$$((\ell, v), t, (\ell, v + t)) \in S \times \mathbb{R}_{\geq 0} \times S \text{ [1]};$$

  - *the* action *(or* discrete*) transitions, consisting of all triples*

$$((\ell, v), (\ell, g, a, r, \ell'), (\ell', v_{[r \leftarrow 0]})) \in S \times E \times S$$

such that $v \models g$.

We note $\mathsf{enab}(s) = \{e \in E^{\mathcal{A}} \mid s \xrightarrow{e}\}$ the set of transitions of the timed automaton enabled in a configuration s.

A *partial run* of $\mathcal{A}$ is a (finite or infinite) sequence $\rho = ((s_{i-1}, p_i, s_i))_{1 \leq i < n}$ of transitions in $\mathcal{T}^{\mathcal{A}}$, with $n \in \mathbb{N} \cup \{+\infty\}$. We write $\mathsf{first}(\rho)$ for s_0 and, when $n \in \mathbb{N}$, $\mathsf{last}(\rho)$ for s_n. A *run* is a partial run starting in the initial configuration s_{init}. The duration of ρ is $\mathsf{dur}(\rho) = \sum_{p_i \in \mathbb{R}_{\geq 0}} p_i$. We write $\mathsf{Ex}(\mathcal{A})$ for the set of runs of $\mathcal{A}$ and $\mathsf{pEx}(\mathcal{A})$ the subset of partial runs.

A configuration s is said *reachable* from a configuration s' when there exists a partial run starting in s' and ending in s. We write $\mathsf{Reach}(\mathcal{A}, S')$ for the set of states that are reachable from some configuration in S', and $\mathsf{Reach}(\mathcal{A})$ for $\mathsf{Reach}(\mathcal{A}, \{s_{init}\})$.

A (partial) run ρ is said to *belong to* a (partial) path π, denoted $\rho \in \pi$ when its sequence of discrete transitions is π. A path is said *feasible* if and only if some run belongs to it.

Remark 1.2.5. *Notice that a run* $\rho = ((s_{i-1}, p_i, s_i))_{1 \leq i < n}$ *can also be represented as* $(s_{i-1} \xrightarrow{p_i} s_i)_{1 \leq i < n}$ *using the notation* $\rightarrow$ *defining the transitions of the semantics.*

Example 1.2.6. *Consider again the automaton of Figure1.3. A run of this automaton is*

$$\rho = s_{init} \xrightarrow{1.5} (\ell_{init}, \binom{1.5}{1.5}) \xrightarrow{e_1} (\ell_1, \binom{1.5}{1.5}) \xrightarrow{e_2} (\ell_{init}, \binom{0}{1.5}) \xrightarrow{e_1} (\ell_1, \binom{0}{1.5}) \xrightarrow{1} \xrightarrow{1} (\ell_1, \binom{2}{3.5}) \xrightarrow{e_3} (\ell_2, \binom{2}{3.5}) .$$

As $\rho \in \pi_2 = e_1 \cdot e_2 \cdot e_1 \cdot e_3$ *this shows that* π_2 *is feasible. On the contrary,* $\pi_1 = e_1 \cdot e_3$ *is unfeasible. Indeed, as a run starts from* s_{init} *and* e_1 *does not reset* c*, it is impossible to satisfy the guard of* e_3 *after the first* e_1*.*

1. By definition of S, both v and $v + t$ satisfy the local invariant.

The *(partial) signature* associated with a (partial) run $\rho = ((s_{i-1}, p_i, s_i))_{1 \leq i < n}$ is $\mathsf{sig}(\rho) = (\mathsf{proj}(p_i))_{1 \leq i < n}$, where $\mathsf{proj}(p) = p$ if $p \in \mathbb{R}_{\geq 0}$, and $\mathsf{proj}(p) = (a, C')$ if $p = (\ell, g, a, C', \ell')$. We write $\mathsf{pSig}(\mathcal{A}) = \mathsf{proj}(\mathsf{pEx}(\mathcal{A}))$ and $\mathsf{Sig}(\mathcal{A}) = \mathsf{proj}(\mathsf{Ex}(\mathcal{A}))$ for the sets of (partial) signatures of $\mathcal{A}$. We write $s \xrightarrow{\mu} s'$ when there exists a (partial) finite run ρ such that $\mu = \mathsf{proj}(\rho)$, $\mathsf{first}(\rho) = s$ and $\mathsf{last}(\rho) = s'$, and write $\mathsf{dur}(\mu) = \Sigma_{\gamma_i \in \mathbb{R}_{\geq 0}} \gamma_i$. Note that as expected, when $\mathsf{sig}(\rho) = \mu$ then $\mathsf{dur}(\rho) = \mathsf{dur}(\mu)$. We write $s \xrightarrow{\mu}$ when $s \xrightarrow{\mu} s'$ for some s'.

Notice that in the above definitions, runs and signatures could have several successive delay transitions, and several successive discrete transitions. This formulation is useful when a fine control on the progression of the system is necessary. In the context of this thesis, this formulation is used notably for game theory. It corresponds to a point of view from *inside* the model, when one wants to be able to "stop time" to take a decision before letting it elapse anew.

Yet, from an outside approach, only the total amount of time elapsed between two discrete actions is of interest. In this context, concerned with models for the languages it generates or reads, manipulating words that alternate strictly between delays and discrete events eases the discussion.

We say that a run (resp. signature) is *alternating* when it starts with a time elapse transition and alternates strictly between time elapses and discrete actions. Note that it is always possible to convert a non-alternating run / signature into an alternating one in a canonical way by summing consecutive delays and adding 0 delays between consecutive actions. In the following, we call alternating signatures ending by a discrete action (*i.e.* in $(\mathbb{R}_{\geq 0} \times \Sigma \times 2^C)^*$) *timed words with resets*.

The *trace* of a (partial) signature corresponds to what can be observed by the environment, namely delays and discrete actions. The trace of a signature is the limit of the following inductive definition, for $t, t' \in \mathbb{R}_{\geq 0}$, $a, b \in \Sigma$, $C' \subseteq C$, and partial signatures μ_1, μ_2:

$$\begin{aligned} \mathsf{trace}(\varepsilon) &= 0 \\ \mathsf{trace}(t) &= t \\ \mathsf{trace}((a, C')) &= a \\ \mathsf{trace}(\mu_1 \cdot \mu_2) &= \mathsf{trace}(\mu_1) \cdot \mathsf{trace}(\mu_2) \end{aligned}$$

with the concatenation of traces being defined with two special cases as follow:

$$t \cdot t' = t + t'$$
$$a \cdot b = a \cdot 0 \cdot b \ .$$

Remark 1.2.7. *This definition ensures that traces are always alternating, starting from a delay. The interest is to enforce a structure corresponding to an observation. Notably, the separations between consecutive delays in (non alternating) signatures can correspond to informations about the signature construction that should not be available by an outside observer.*

We say that a (partial) trace ending with a discrete action is a *timed word* (notably the trace of a timed word with resets is a timed word). We denote timed words as elements of $(\mathbb{R}_{\geq 0} \times \Sigma)^*$.

We also define $\mathsf{resets}(\mu)$ as the sequence of clock resets appearing in μ and define the operator $\otimes$ that from a timed word $w_t = ((t_i.a_i))_{1 \leq i < n}$ and a reset sequence of same length $r = (r_i)_{1 \leq i < n}$ builds the timed word with resets $w \otimes r = ((t_i, a_i, r_i))_{1 \leq i < n}$.

Remark 1.2.8. *The $\otimes$ operation could be defined for general signatures and traces, but we only use it for timed words, hence the use of this simpler definition.*

The above definitions are summed up in the Figure 1.4, where the syntax and different levels of observation (semantics, signatures and traces) are highlighted.

Example 1.2.9. *Considering again the run from Example 1.2.6:*

$$\rho = s_{init} \xrightarrow{1.5} (\ell_{init}, \binom{1.5}{1.5}) \xrightarrow{e_1} (\ell_1, \binom{1.5}{1.5}) \xrightarrow{e_2} (\ell_{init}, \binom{0}{1.5}) \xrightarrow{e_1} (\ell_1, \binom{0}{1.5}) \xrightarrow{1} \xrightarrow{1} (\ell_1, \binom{2}{3.5}) \xrightarrow{e_3} (\ell_2, \binom{2}{3.5})$$

one can construct an alternating run

$$\rho' = s_{init} \xrightarrow{1.5} (\ell_{init}, \binom{1.5}{1.5}) \xrightarrow{e_1} \xrightarrow{0} (\ell_1, \binom{1.5}{1.5}) \xrightarrow{e_2} \xrightarrow{0} (\ell_{init}, \binom{0}{1.5}) \xrightarrow{e_1} (\ell_1, \binom{0}{1.5}) \xrightarrow{2} (\ell_1, \binom{2}{3.5}) \xrightarrow{e_3} (\ell_2, \binom{2}{3.5}).$$

The signature of ρ is

$$\mu = \mathsf{sig}(\rho) = 1.5 \cdot (a, \emptyset) \cdot (b, \{c_1\}) \cdot (a, \emptyset) \cdot (b, \{c_1\}) \cdot 1 \cdot 1 \cdot (a, \emptyset).$$

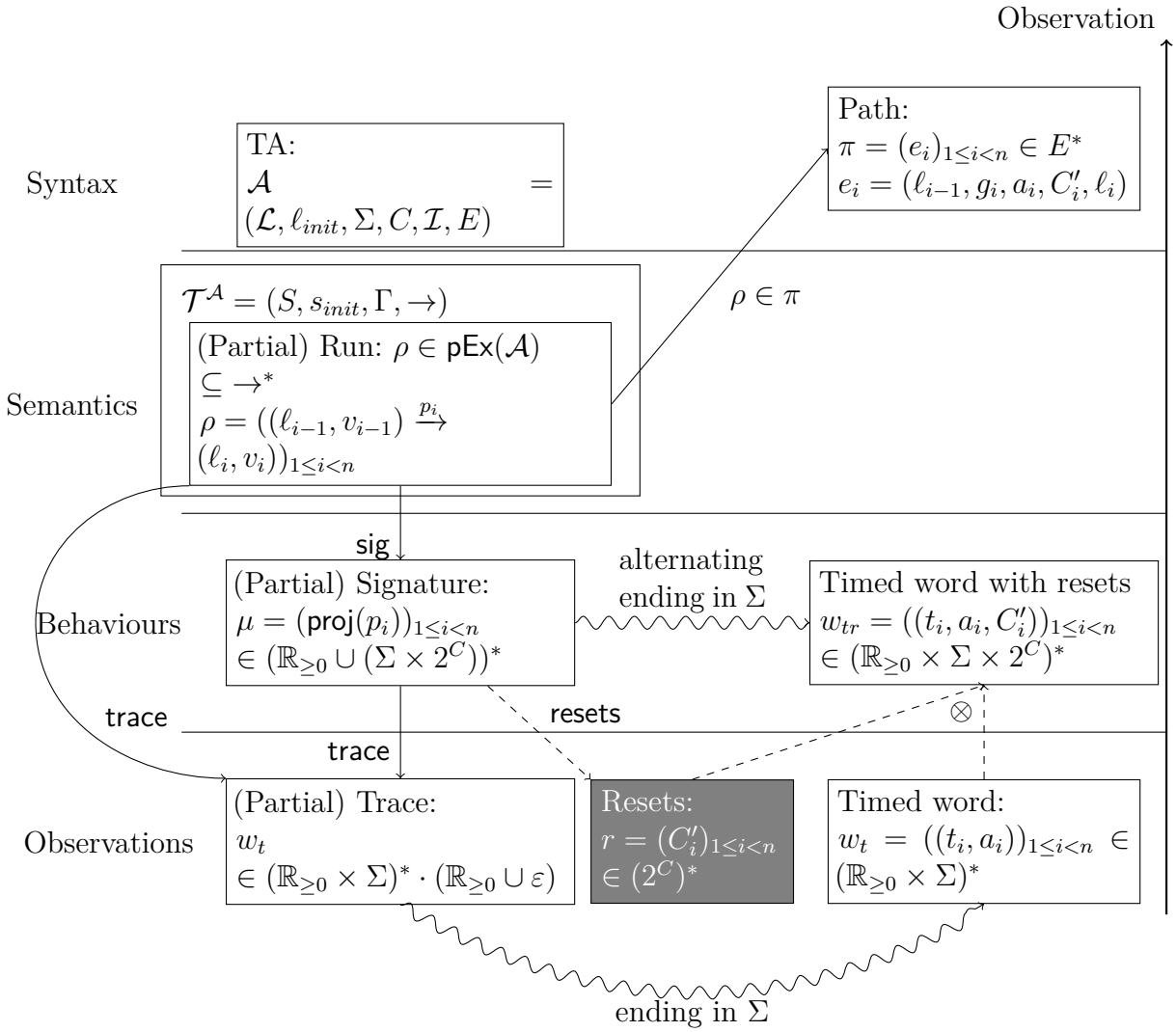


Figure 1.4: The central notions of timed automata executions.

Notice that the timed word with resets μ' constructed from μ by making it alternate is $\mathsf{sig}(\rho')$.

$$\mu' = 1.5 \cdot (a, \emptyset) \cdot 0 \cdot (b, \{c_1\}) \cdot 0 \cdot (a, \emptyset) \cdot 2 \cdot (a, \emptyset).$$

Finally, we can construct the trace $w = \mathsf{trace}(\mu) = \mathsf{trace}(\mu')$*:*

$$w = 1.5 \cdot a \cdot 0 \cdot b \cdot 0 \cdot a \cdot 2 \cdot a.$$

For a run ρ, we write $\mathsf{trace}(\rho) = \mathsf{trace}(\mathsf{sig}(\rho))$. In the same way, for a timed automaton $\mathcal{A}$, we write $\mathsf{Traces}(\mathcal{A}) = \bigcup_{\mu \in \mathsf{Sig}(\mathcal{A})} \mathsf{trace}(\mu)$ the set of traces corresponding to its runs and $\mathsf{pTraces}(\mathcal{A})$ the set of traces corresponding to partial runs. Note that all traces are alternating, as they are meant to encode the words that can be observed. Two timed automata are said *trace equivalent* if they have the same set of traces.

Remark 1.2.10. *From the point of view of the language of timed words generated by a model, timed automata without invariants are as expressives as timed automata with invariants. Indeed, starting from a TA $\mathcal{A}$, one can intersect all the the guards of the outgoing transitions of a given location with its invariant,* i.e. *replace g in (ℓ, g, a, C', ℓ') by $g \wedge \mathcal{I}(\ell)$, and suppress the invariants of the automaton. The only difference between the runs of $\mathcal{A}$ and the resulting automaton $\mathcal{A}'$ is the appearance of greater final delays in $\mathcal{A}'$ which do not appear in timed words (as they end with discrete actions).*

While different from a control point of view as new configurations can be reached in $\mathcal{A}'$ these timed automata have the same language of timed words. For this reason, we will sometimes omit invariants entirely when focusing on the languages of timed words generated by timed automata.

We define the notion of determinism for timed automata.

Definition 1.2.11. *A timed automaton $\mathcal{A}$ is said*

- deterministic *if for all $e_1 = (\ell, g_1, a, C_1, \ell_2')$ and $e_2 = (\ell, g_2, a, C_2, \ell_2')$ either $e_1 = e_2$ or $g_1 \cap g_2 = \emptyset$;*
- determinizable *if there exists a trace-equivalent deterministic TA.*

Remark 1.2.12. *The notion of determinism defined above is* syntactic*. It entails* semantic *determinism: for all $w \in Traces(\mathcal{A})$, there is a unique $\rho \in \mathsf{Ex}(\mathcal{A})$ such that $\mathsf{trace}(\rho) = w$. The converse is not true, as semantic determinism gives no constraints on unreachable valuations.*

We chose the syntactic approach because it makes proofs easier when reasoning at the syntactic level and can be checked at a lesser cost as it only requires to compare transitions of the timed automaton, as opposed to its sets of runs and traces.

An important properties of timed automata is that they are not all determinizable. Even more, the determinizability of a timed automaton is an undecidable problem [Fin06; Tri06]. We write DTA for deterministic timed automata (and will write D- to emphasize that we restrict to deterministic models when manipulating other related classes).

Example 1.2.13. *Figure 1.5 represents a non-deterministic, non-determinizable timed automaton.*

Intuitively, this automaton requires that a pair of actions must be separated by exactly one time unit. A deterministic TA should thus keep the timing information of every action that happened less than one time unit ago, to be able to detect wether those actions happened exactly one time unit ago.

This cannot be done with a finite set of clocks, as unboundedly many actions could occur [AD94].

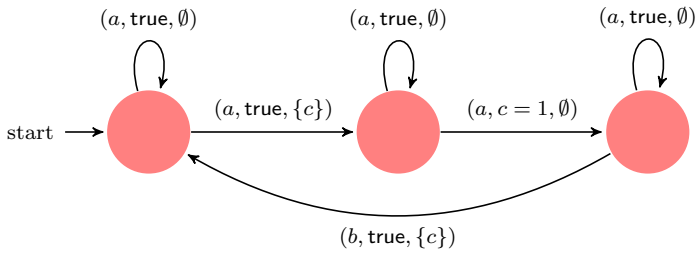


Figure 1.5: A 1-clock non-determinizable timed automaton.

1.3 Equivalence relations

As explained in Section 1.2, the semantics $\mathcal{T}^{\mathcal{A}}$ of a timed automaton $\mathcal{A}$ is an infinite transition system. While it is possible to use such semantics for abstract reasoning, a *finite* representation is necessary, notably for implementation purposes.

Most of these representations are based on zones with states of the form $(\ell, z) \in \mathcal{L} \times \mathcal{Z}$. We will often abuse the notation and write $s \in (\ell, z)$ when $s = (\ell, v)$ and $v \models z$. We first define *minimal* abstractions, based on equivalence relations on valuations: region equivalence and K-equivalence.

Definition 1.3.1. *For a TA $\mathcal{A}$ equipped with the set of clocks C, consider the following relation $\approx_{reg}$ [AD94] between valuations: $v \approx_{reg} v'$ when the following conditions are met:*

1. $\forall c \in C,\ v(c) > K^{\mathcal{A}}(c) \Leftrightarrow v'(c) > K^{\mathcal{A}}(c)$;
2. $\forall c \in C,\ v(c) \leq K^{\mathcal{A}}(c) \Rightarrow (\lfloor v(c) \rfloor = \lfloor v'(c) \rfloor \wedge (\langle v(c) \rangle = 0 \Leftrightarrow \langle v'(c) \rangle = 0))$;
3. $\forall c, c' \in C,\ \Big(v(c) \leq K^{\mathcal{A}}(c) \wedge v(c') \leq K^{\mathcal{A}}(c')\Big) \Rightarrow$
   $(\langle v(c) \rangle \leq \langle v(c') \rangle \Leftrightarrow \langle v'(c) \rangle \leq \langle v'(c') \rangle)$.

where $\lfloor \cdot \rfloor$ is the integer part operator and $\langle \cdot \rangle$ the fractional part operator. $\approx_{reg}$ is an equivalence relation and we call regions *its equivalence classes. We write $\mathcal{R}_{\mathcal{A}}$ for the set of regions of a timed automaton $\mathcal{A}$.*

The $\approx_{reg}$ relation is a time-abstract bisimulation relation *i.e.* if two valuations v, v' are in a same region *reg*, there exists $t \in \mathbb{R}_{\geq 0}$ such that $v + t$ is in a region *reg'* if and only if there exists $t' \in \mathbb{R}_{\geq 0}$ such that $v' + t' \in reg'$. The name of time-abstract bisimulation comes from the fact that from v, one can simulate the behaviour of v' and conversely (bisimulation), but this may require to choose different delays (time-abstract).

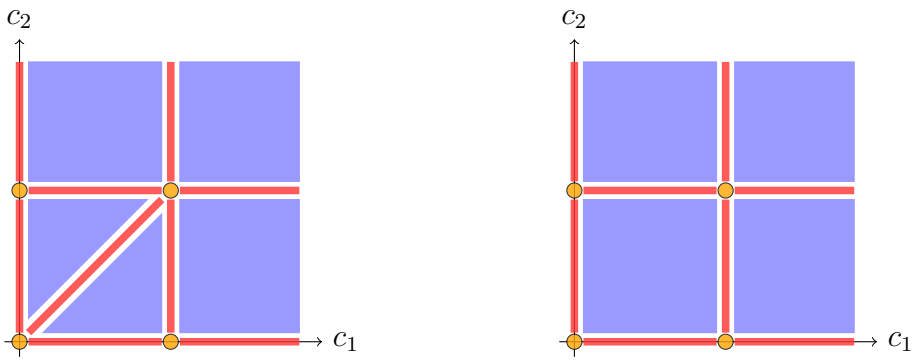
Definition 1.3.2. *For a TA $\mathcal{A}$ equipped with the set of clocks C, consider the relation $\approx_K$ between valuations defined by restricting to points 1. and 2. of Def. 1.3.1.*

*We call the equivalence classes of $\approx_K$ K-*closed zones *and write $Kz(v)$ for the one containing v.*

K-equivalence is coarser than region equivalence. Indeed, it only encodes direct distinguishability by a guard of $\mathcal{A}$ but is not a bisimulation relation. Notice that there is a finite number of regions and K-closed zones in any timed automaton thanks to the limitation by the automaton's maximal constants.

Example 1.3.3. *In Figure 1.6 the equivalence classes for region equivalence and K-equivalence are depicted for two clocks $C = \{c_1, c_2\}$ and maximal constants of* 1.

Note that both regions and K-closed zones are zones. For a region or a K-closed zone z we define $\mathsf{enab}(\ell, z) = \mathsf{enab}(\ell, v)$ for any $v \models z$. Furthermore for a region *reg* and a transition $e \in E^{\mathcal{A}}$, we define $\mathsf{next}(e, reg)$ as the unique region *reg'* such that $\forall s \in (\ell, reg),\ s \xrightarrow{e} s' \in reg'$.



(a) Regions (b) K-closed zones

Figure 1.6: Equivalence classes.

Chapter 2

State of the art

> I have no idea at the moment—if you mean about removing the treasure. That obviously depends entirely on some new turn of luck and the getting rid of Smaug. Getting rid of dragons is not at all in my line, but I will do my best to think about it. Personally I have no hopes at all, and wish I was safe back at home
>
> — J.R.R. Tolkien "The Hobbit"

In this chapter, an overview of the state of the art of some research domains related to this thesis topic is proposed. Most of the literature cited in this chapter is closely related to one or more of the thesis axes, but some subsections will be identified as general culture and are proposed for the reader's interest. They can safely be skipped for the purposes of the technical developments.

2.1 Timed models

Real-time behaviours often play an important role for modelling and specifying correctness of computer systems. Discrete models, such as finite-state automata, are not adequate to model real-time aspects of a behaviour or encode constraints on them. For this reason, different communities have proposed variants of their models taking time into account.

2.1.1 Timed automata and related models

A variety of different models exist to take continuous time into account. The most ancient state-machine including (continuous) time is the *time Markov decision process* that goes

back to the 1960's [How60; Mil68]. In this model, the time elapsed before a selected action happens is sampled from a continuous random variable. There are however no constraints on timing embedded in the model.

Timing constraints have been considered in two concomitant extensions of Petri nets to continuous time: Merlin's *time Petri nets* [Mer74] and Ramchandani's *timed Petri nets* [Ram74]. Both models feature explicit time intervals that have to be respected by executions, but are otherwise not equivalent.

The study of continuous time system has boomed during the 1990's with the development of several models. By far the most expressive, hybrid dynamical systems, or hybrid systems for short [Alu+92; Nic+93; Alu+95; Hen96], combine discrete automata-like jumps between control states and continuous behaviours where quantities evolve over time as dictated by differential equations. As a side effect of hybrid systems expressivity, many related decision problems are undecidable. Notably reachability is in general undecidable for hybrid systems, even as "simple" as *stopwatch automata* where differential equations are of the form $\dot{x} = 0$ and $\dot{x} = 1$, although decidable subclasses of hybrid systems are identified [Hen+95].

Timed automata have been introduced in 1994 by Alur and Dill [AD94][1]. Less expressive than hybrid systems, their reachability problem is decidable (although PSPACE-complete) [AD94]. Yet they remain highly complex models in general. For example, not all timed automata are determinizable and determinizability is undecidable [Fin06; Tri06]; the universality problem (Are all words accepted by a given model?) and the language inclusion problem are decidable only for deterministic timed automata.

For this reason, a number of subclasses have been studied over the years, by reducing the number of clocks, or linking clock resets to discrete actions as in *e.g.* event recording automata [AFH99], a subclass first introduced because it is determinizable. This subclass (and an extension of it proposed during the course of this thesis) will be further discussed and introduced in Chapter 5. The number of subclasses is one of the appeals of timed automata: any general algorithm can be declined for a great number of less complex use cases.

Extensions of timed automata also exist. Some add new considerations to timed automata, such as timed automata with energy that add a constraint in the form of a positive "energy" variable that has to be maintained above 0 through a run [Bou+08; Bac+21], timed game automata [Asa+98; Cas+05] that add controllability aspects in the

1. A first conference version was published in 1990.

form of a (2-player) game or timed automata with inputs and outputs [KT09] (this last will be presented in context in Section 2.2 and further discussed in Chapter 3).

Others are simpler variants on the definition, such as adding invariants (as presented Section 1.2), first introduced in Timed Safety Automata [Hen+94], a model slightly different from modern timed automata (based on [AD94]) that deviated from the first timed automaton [ACD90]. Another classic variant authorizes diagonal constraints *i.e.* constraints of the form $c_1 - c_2 \sim n$ with $c_1, c_2 \in C$, $n \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \geq, >\}$. Diagonal constraints have been first discussed in [AD94]. It is known that a timed automaton with diagonal constraints can be converted into a classic timed automaton (*i.e.* without diagonal constraints), although at the cost of an exponential blow-up in locations, depending on the number of diagonal constraints [Bér+98]. Interestingly, the usual reachability algorithm for TAs, which is zone based, has been found to be faulty in the case of timed automata with diagonal constraints [Bou03; BY03] and a modified algorithm has been proposed [BLR05].

More recently, some models including continuous time have been built upon Mealy machines, such as time delay Mealy machines [CCF15] and their generalization Mealy machines with a single timer [VBE21]. Time delay Mealy machines propose to use a single countdown *timer*, forcing an action to be taken instantly when it reaches 0. The model dictates that the timer is reset at each transition, which limits its expressivity. Mealy machines with a single timer remove that reset constraint.

In the following of this thesis, we will focus our technical developments on timed automata, although related approaches based on other timed models are often part of the relevant literature.

2.1.2 Behaviour abstraction

In the first introduction of timed automata, reachability of a given configuration has been proven to be PSPACE-complete by the mean of the *region graph* associated with a timed automaton, that is a finite (untimed) automaton whose states are pairs of locations and clock regions.

Despite this challenging worst case, a great amount of work has been put into more efficient algorithms to solve reachability and other related problems for timed automata.

Most of these algorithm rely primarily on *zones* (presented in Section 1.1) and their implementations with *Difference-Bound Matrices* (DBMs), defined in [BM83] with an application for Time Petri nets and introduced to timed automata by [Dil90].

Difference-bound matrices for zones upon the set of clocks $C = \{c_1, \ldots, c_n\}$ are

$(n+1) \times (n+1)$ matrices such that the coefficient $m_{i,j}$ of a matrix at the i-th line and j-th column is a pair $(c, \prec)$ with $n \in \mathbb{Z}$ and $\prec \in \{<, \leq\}$. It is interpreted as the constraint $c_{i-1} - c_{j-1} \prec n$ with a special clock c_0 that is always evaluated to 0, serving to encode guard constraints (*i.e.* $c \sim n$). Difference-bound matrices come with operations corresponding to the usual operations on zones and a canonical form uniquely representing zones, see *e.g.* [BY04; San13].

Zones and DBMs, as well as other finite representations (for a discussion and related algorithms, see *e.g.* [Rou20]), are the basis of several scientific and industrial tools for timed automata, which have adopted always more algorithms to gain efficiency in practice. Amongst the best-known tools handling timed automata are:

Kronos [Daw+96; Boz+98] the historic first tool implementing verification of timed automata and a forward reachability algorithm, now outclassed by successors[2];

Uppaal [Beh+06] is an academic and industrial tool building upon timed automata and expending the model in various ways (*e.g.* with game semantics, different type of actions, notions of urgency or parameters). Sadly, the software code is kept by its designers, which has complicated its interaction with the research community[3];

IMITATOR [And21; And+12] a parametric verification tool for real time systems, modeled as networks of timed automata with some extensions[4];

TCHECKER an academic tool providing a test-bed for verification algorithms of timed automata, augmented by libraries implementing classic data-structures and algorithms [5];

Roméo [Lim+09] an open source modeling tool for parametric systems based on Petri-nets, with extensions in real-time such as stopwatches. It can notably translate timed Petri nets into timed automata [6];

PAT [Sun+09] a very general tool dealing in concurrent and real-time systems and providing many model-checking algorithms for them [7];

2. https://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/
3. https://uppaal.org/
4. https://www.imitator.fr/index.html
5. https://www.labri.fr/perso/herbrete/tchecker/
6. http://romeo.rts-software.org/
7. https://pat.comp.nus.edu.sg/

IF TTG [KT05] is a test-generation tool handling non-deterministic timed automata.

Because of the predominance of those representations, most algorithms introduced for timed automata (especially practical algorithms) manipulate a zone-based representation of the automaton behaviours.

2.2 Model verification

Offline verification. Because of the wide range of applications of computer systems—notably their use in numerous safety critical systems—and of their increasing complexity, the use of formal methods for checking their correct behaviours has become essential [HS06; CES09]. It is even one of the main motivations (if not the main) for the introduction of formal models. Numerous approaches have been introduced and extensively studied over the last 40 years, and mature tools now exist and are used in practice, as those mentioned in Section 2.1.2 for timed automata, see *e.g.* [BY04; Beh+06].

Most of these approaches rely on building mathematical models, such as automata and extensions thereof, in order to represent and reason about the behaviours of those systems; various algorithmic techniques are then applied in order to ensure correctness of those behaviours, such as *model checking* [CGP00; Cla+18] or *deductive verification* [Hoa69; Fil11].

These techniques focus on assessing properties of a model in an *offline* manner *i.e.* reasoning on the system model, without executing the system itself. With powerful properties such as the correctness of all of the system behaviours, this can lead to very costly algorithms, in terms of computation and memory. This is sometimes a too demanding requirement, especially with highly expressive timed models. Furthermore, it is sometimes desirable to not only check the correctness of *a model* of the system but to verify properties of its *implementation* in its environment, in which bugs could appear independently of the model correctness *e.g.* due to implementations errors (software) or faulty components (hardware), or interactions between them.

Online verification. To leverage this issue, a set of *online* (or *runtime*) verification approaches have been crafted, that interact with the running system to assess its properties [LS09].

Fault diagnosis is a prominent problem in runtime verification: it consists in monitoring real executions of a partially-observable system, and detect at runtime, as early

as possible, whether some property holds (*e.g.* whether some unobservable fault has occurred)[8] [Sam+96]. This is usually done by synthesizing a diagnoser *i.e.* a formal model used to monitor the executions w.r.t. the properties of interest. For finite-state models, a diagnoser can usually be built by determinizing a model of the system, using the power-set construction; it will keep track of all possible states that can be reached after each (observable) step of the system, thereby computing whether a fault may or must have occurred. The related problem of *prediction*, a.k.a. prognosis, (that e.g. no faults may occur in the near future) [GL09], is also of particular interest in runtime verification, and can be solved using similar techniques [Jér+08].

Conformance testing is another online verification technique whose aim is to check whether the behaviours of a (black-box) running system conform to its specification. To do that a tester runs test cases which apply inputs to the system and observe that outputs conform to the specification. In its model-based declination [Tre96], the specification is formal, *e.g.* an automaton, and since the system and specification are partially observable, the generation of tests (online or offline) may also rely on determinization to know the effect of any given action of the system in order to control it.

The following subsections discuss more in depth the model-based testing of timed systems and focus on the state estimation problem underlying determinization.

2.2.1 Model-based testing

Real-time reactive systems are open systems interacting with their environment and subject to timing constraints. Such systems are encountered in many contexts, in particular in critical applications such as transportation, control of manufacturing systems, etc. Their correctness is then of prime importance, but is also very challenging due to multiple factors: combination of discrete and continuous behaviours, concurrency aspects in distributed systems, partial observability and limited controllability of open systems.

To assess the correctness of such systems, testing remains the most-used validation technique, with variations depending on the design phase. Conformance testing is one of those variations, consisting in checking whether a real system correctly implements its specification, which serves as a reference. Those real systems, also called *implementations under test* are considered as black boxes, thereby offering only partial observability to the tester, for various reasons (*e.g.* because sensors cannot observe all actions, or because the

8. Although the termed "diagnosis" is coined by the community, fault *detection* may be more accurate, as suggested by S. Tripakis [Tri02], as it is not concerned with the root *cause* of a failure.

system is composed of communicating components whose communications cannot all be observed, or because of intellectual property of peer software). Controllability is another well-known issue when the system makes its own choices upon which the environment, and thus the tester, has a limited control. This is the main focus of the ideas developed in Chapter 3 around strategies to control test cases.

One of the most challenging activities of conformance testing is the design of test cases that, when executed on the real system, produce meaningful verdicts about the conformance of the system at hand with respect to its specification. Formal models and methods are good candidates to help test-case synthesis, both in terms of productivity gain and increased confidence in their verdict [Tre96]. Observability and controllability problems are central issues to overcome in this task.

In formal testing, it is adequate to refine the TA model by explicitly distinguishing (controllable) inputs and (uncontrollable) outputs, giving rise to the model of TAIOs (Timed Automata with Inputs and Outputs) [KT09]. Other variations of the TA model have been equipped with inputs and outputs and used in the context of testing, such as TIOA [SVD01; Kay+03], or TIOTS [LMN04].

Test-case synthesis from timed automata. Test-case synthesis from TAs has been extensively studied during the last 20 years (see [CG98; CKL98; SVD01; EDK02; NS03; BB04; LMN04; KT09; Ber+12], to cite a few). When testing offline, the test cases are first computed, stored, and later executed on the implementation. They should thus anticipate all specified outputs after an observed trace. But then, one of the difficulties comes from partial observation. In the untimed framework, this is tackled by determinizing the specification. Unfortunately, this is not always feasible for TAIO specifications since determinization is not possible in general. The solution is then either to perform online testing, where a subset construction is made on the current execution trace [LMN04], or to restrict to determinizable sub-classes (see *e.g.* [NS03] for ERAs). While the latter approach limits the class of models that can be used, the former suffers from the multiplication of possible configurations after a given trace.

Partial observation is addressed in [Dav+10] with a variant of the TA model where observations are described by observation predicates, composed of a set of locations together with clock constraints. Test cases are then synthesized as winning strategies, if they exist, in a game between the specification and its environment that tries to guide the system to satisfy the test purpose. More recently, some advances were obtained in [Ber+12] by the

use of an approximate determinization procedure using a game approach [Ber+15] that preserves tioco conformance and is exact for most of the known determinizable sub-classes of TAs when sufficient resources (*i.e.* number of clocks and maximal constant) are provided.

Controllability of a system under test. The problem of testing is often informally presented as a game between the environment and the system under test (see *e.g.* [Yan04]). But in reality very few papers effectively take into account the controllability of the system in this game.

In the context of timed testing, a game approach has been studied in [Dav+08a], where test cases are synthesized as winning strategies of a reachability game. But this work is restricted to deterministic models, and controllability problems are not really mastered. In fact, like in [Dav+10], when no winning strategies are found, the game is abandoned and it is suggested to modify the test purpose. This is mitigated in [Dav+08b] with the use of cooperative strategies, which rely on the cooperation of the system under test to win the game. The strategy can furthermore distinguish between *cooperating* states, where it needs to rely on the system to win, and *winning* states, where it can win despite the system best effort.

A more in-depth approach to the controllability problem is the one of [Ram98] in the untimed setting, unfortunately a scarcely-known work. Test selection is modelled as a game where the tester tries to satisfy a test purpose while detecting non-conformance, but faces *control losses*, *i.e.*, states where the system proposes uncontrollable but correct outputs that moves it away from its objective. The computed strategies could quantify on the number of *control losses*. Interestingly, this allows to minimize both the reliance on the system decisions and the distance to the next control loss.

Conformance testing of timed automata is discussed in the Chapter 3 of this thesis. Notice that Uppaal has an extension named TIGA (TImed GAmes) that implements timed games algorithms [Cas+05].

2.2.2 State estimation

As mentioned before, efficient offline algorithms for the analysis of timed automata have been proposed and implemented. Despite these successes, the *state estimation problem* remains one of the most challenging limitations that all methods have to encounter. This problem is concerned with the detection of the possible configurations of a system after a given trace. For offline methods, it takes the form of determinization and closure with respect

to silent transitions. Sadly, for timed automata, determinizability is undecidable [Fin06; Tri06] and silent transitions strictly increase expressivity [BGP96].

For online methods, one can try to efficiently keep tracks of the possible configurations [Gre+20]. For example, Baier et al. have proposed a method to construct a deterministic timed tree from a timed automaton, with the limitation that the tree may be infinite [Bai+09]. This method has been declined for diagnosis [Tri02], and implemented in UPPAAL TRON [Hes+08] (one of UPPAAL's libraries) and IF TTG [KT09] for online testing of timed input-output systems.

This approach is computationally very expensive, as one step consists in maintaining the set of all configurations that can be reached by following (arbitrarily long) sequences of unobservable transitions; moreover, the set of possible configurations is updated only when a new event is observed (or after a timeout), which may significantly delay the detection of a fault.

Another possibility is to impose some *restrictions* on the constructed model. Bouyer, Chevalier and D'Souza studied a restricted setting, only looking for diagnosers under the form of deterministic timed automata with limited resources [BCD05]. Similarly, Krichen et al. have proposed to build a deterministic timed automaton from a non-deterministic one, by fixing a set of clocks and possibly making some approximations in the behaviour [KT09]. Bertrand et al. later generalized and improved this approach using games [Ber+15]. Both works have then been used for offline test generation from timed automata models [KT09; Ber+12; HJM18].

Finally, *automata over timed domains* [BJM17], a larger determinizable class of models comprising timed automata has been created to offer a determinization procedure, but with a great expressivity—and hence complexity.

2.2.3 Robustness(es) of models

This subsection is for scientific culture, as no contribution is made on robustness on this thesis. We include it to acknowledge the importance of robustness in model-reality interactions and provide (a few) bibliographic pointers on that matter.

The robustness of a model is generally understood as its ability to maintain good properties under small perturbations. This take is obviously vague, notably depending on what one consider a "good property" and a perturbation—which can both vary. In the case of timed models, an important emphasis is given to *timing errors* that can arise from imprecise measure of time, unexpected runtime behaviours or imprecisions in actuator

controls.

Precise definitions and modeling vary. *Robust timed automata* [GHJ97] first proposed a topological semantics for timed automata and consider that a run is only accepted if an open tube around it is accepted too (*i.e.* it excludes isolated behaviours). Timed automata with clock drifts were also considered [Pur00]. A classic syntactic approach is to consider automata with enlarged or shrinked guards by a small $0 < \epsilon < 1$ [De +04]. A survey of this approach and related works and results for the verification, implementation and monitoring of such system can be found in [BMS13].

Game based approaches also exist, that try to propose optimally robust strategies for reachability [Cle+20].

2.3 Model learning

Machine learning is the automatic improvement of agents through experience and interactions with an environment, which can consist of provided data or be an open word. It is one of the core component of the so-called *artificial intelligence* domain that tries to design machines that can mimic (or approximate) cognitive functions associated with the human or animal mind, typically *learning* and *problem solving*.

Concept learning is a sub-domain of machine learning interested in the automatic inference and refinement of high-level concepts from the environment. In the context of interactions with formal methods, it is interesting that the representation of those concepts takes the form of formal models (state machines, logic formulae...). This specific branch of research is called *model learning*.

In this section we first propose a quick discussion about the various passive model learning methods in existence (Section 2.3.1), then a presentation of *active learning*, the learning method most relevant to this thesis (Section 2.3.2).

2.3.1 Quick dive in passive model learning

This subsection is mostly general culture, as it presents roots of model learning and some research efforts that differ greatly from the technical developments of this thesis which are based on active learning.

One of the theoretical backbones of model learning is the work of Anil Nerode on the so called *Nerode congruence* [Ner58], an equivalence relation between states of a deterministic

finite automaton allowing to define the minimal deterministic automaton recognizing a given language.

The first historical approach to model learning is a *passive learning* approach, where the agent learns from predetermined set of examples, either *positive* (*i.e.* part of the language to be learned) or *negative* (*i.e.* out of the language to be learned).

It is long known that learning from positive examples only is far less expressive. For instance, primitive recursive languages can be identified in the limit—roughly, after an unbounded but finite number of examples—by passive learning, but only finite languages can be identified from positive examples only [Gol67]. Notably, negative examples are necessary to learn the class of regular languages. Gold then investigated the learning from a *finite* set of given data, and found that DFA learning[9] is NP-hard in this framework [Gol78]. Angluin gave a characterization of the learnability from text [Ang80] and investigated some ways to extend the class of learnable languages. This research effort lead to active learning (see Section 2.3.2) and showed that relaxing the learning condition to learning with probability one enhances that expressivity [Ang88].

Learning of cyber-physical systems. Model learning was in part motivated by the need to obtain models to test cyber-physical systems. Obtaining negative examples of a cyber-physical system behaviours is costly, if not outright infeasible (as it corresponds to security issues or require to break the system). Due to the restriction on learning from positive observations only, researchers have sought passive learning methods for probabilistic automata from positive examples only.

One of the best known methods is ALERGIA [CO94]. This algorithm first constructs a tree representing the available examples and then merge its states when their sub-trees have similar languages (*i.e.* the same language with a high probability). This approach has been developed with a global criterion for merging in the algorithm MDI (Minimal Divergence Inference) [TDH00] that allows to obtain a bound on the (Kullback-Leibler) divergence between the samples and the model. A state-merging method based on Kullback-Leibler divergence is also proposed in [CT04]. The article is more theoretical and does not provide an implementation, but gives an in-depth review of their complexity results. It notable proves that probabilistic deterministic automata are Probably Approximately Correct (PAC)-learnable [Val84], PAC-learning being an important theoretical framework for learning, that links together the probability of error, the precision of the model learned

9. and minimal DFA learning

and the number of required data.

When a model integrating time is required, it is necessary to add a splitting operator to distinguish between different behaviours depending on time. The BUTLA (Bottom Up Timing Learning Algorithm) [Mai+11] is a first work on this topic, learning a probabilistic timed automaton. The authors rely on domain specific knowledge to advocate a split performed to separate modes in the probability distribution of a transition rather than a difference in the sub-automata. This idea is re-exploited in [MNE15] to propose a preprocessing of the data to separate the different modes, suppressing the splitting operation. A general scheme for offline learning of deterministic real-time automata is presented in [VWW08; VWW12] with statistical tests to decide merge and splits.

All the aforementioned algorithms are offline, working only on a given set of observations. Yet, online techniques for passive learning have also been investigated, forcing to rethink the general scheme of the algorithm, as a tree of all available observations could not be built. OTALA [Mai14] is an online adaptation of BUTLA, allowing online passive learning from positive examples only in an incremental way. It learns a deterministic real time automaton (only one clock, reset at every transition). It has been extended to a parallel framework [WLN17].

An overview of cyber-physical systems [NL15] identifies the following key-points for the domain:

- A general *learnable* model is needed;
- this model should deal with timed and hybrid constraints;
- more work should be done at the level of the component, instead of always focusing on root causes: early identification of a faulty component can be more valuable and easier to obtain to root causes.

Learning software. Another longstanding application of formal methods is software analysis. In the case of model learning, the specificity of software is that it allows to leverage some black-box hypotheses by making assumptions on the structure of the observed object. One interesting point is that this specificity can be used to learn more complex models (such as automata with multiple clocks).

When learning software models, the two main models of interest are specifications describing the language of a program and normal use models that describe how the software is used in practice, focusing not on its expressivity or design but on its interactions with actual users.

A wide array of applications exist for these models, ranging from test generation, anomaly identification to detection of inefficiencies or debugging assistance.

One of the early methods is the k-Tail algorithm, first introduced theoretically [BF72] as a variation on the work of Nerode and then reformulated and implemented [CW98]. This algorithm takes a local approach and compare nodes based on they "k-future" *i.e.* the next k steps in their executions. This approach is especially efficient at detecting loops, which are a central focus of software learning. It has been extended to models with parameters [LMP08] with the gk-Tail algorithm and to real time models with the Timed k-Tail algorithm [PMM17]. Interestingly, Timed k-Tail is able te learn automata with multiple clocks, which are notoriously difficult for passive learning, by using a learning bias and targeting nested behaviours (*e.g.* nested loops or function calls).

An interesting research direction is implemented in the TAUTOKO tool that combines learning and testing. In its earliest version [Dal+10] this tool generates a first model using the results of a test suite and enrich it using mutations on the test suite. This work has been extended to avoid the need for an initial test suite and generate different types of new tests [Dal+12]. This extension allows to iterate on the learning / testing loop and generate new tests. The comparison of these methods is performed in the latter article. The methodology of this work is of great interest to interleave test and learning even in other frameworks, notably the use of a coverage criterion to direct tests at each learning step.

Let us mention that works exist that learn non automata-based models. One example of this is the TREM tool implementing a set of methods to learn regular expressions as specifications [SNF17].

Other applications These works are more loosely related to ours, as our setting greatly differs from positive inference. Applications of model learning are numerous and are not limited to the somewhat classical formal-method application domains of cyber-physical systems and software. We give some examples of different applications.

A combination of learning, abstraction, testing and modeling has been applied to the learning of large scale software with a focus in keeping an expert user in the center of the loop [Xia+05]. This approach has been demonstrated on commercial games, where the purpose is to maximise a loosely defined enjoyability metric, that can only be evaluated by experts. This makes the learned model central for visualization purposes and calls for an interactive method instead of a monolithic tool.

Model learning is also used in medicine. This is not new, with the important use of Markov's models. In a recent thesis [Sch13] the modeling is pushed further with the addition of an explicit modeling of time. The author use probabilistic real time automata (and subclasses of these) to model the evolution of a disease, taking into account both timing and discrete (the different symptoms) aspects.

2.3.2 Active learning

Active learning [Ang87a] is a type of learning in which a teacher assesses the learner's progress and directs the learning effort toward meaningful decisions. The learner can request information from the teacher via *membership queries*, asking about a specific observation, and *equivalence queries*, proposing to compare the current hypothesis to the correct model; in the latter case, the teacher either accepts the hypothesis or returns a counter-example witnessing mispredictions of the learner's hypothesis.

It was introduced by D. Angluin to leverage the restrictions found on learning from random given samples by E. Gold [Gol67; Gol78]. The author argues that in a human learning process you can assume the learner to be "helpful", and to interact with the learner as a professor would do for a student. They take the example of a human specialist trying to train an expert system. To quantify this helpfulness would not make a lot of sense, but Angluin proposes a "minimally adequate teacher" (MAT) that would be able to answer membership and equivalence queries as a reference. The MAT structure is illustrated in Figure 2.1

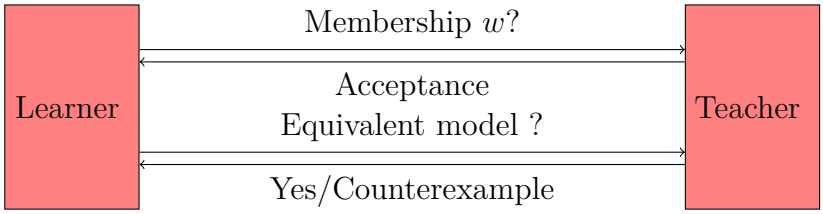


Figure 2.1: The basic active learning framework.

The MAT allows Angluin to define the L^* algorithm, that can learn regular languages represented by a (minimal) deterministic finite automaton. This algorithm identifies nodes by using Nerode's congruence and stands as a proof of concept establishing a well-studied framework [Ang87a; Ang87b; Ang90] which allows for sound proofs of correctness and complexity of learning algorithms. The L^* was afterward refined to deal with input-output

systems [RS93] or to obtain a better data structure based on trees [KV94]. Extensions have then been made to languages of infinite words ([MP95] for a first work on sub-classes and [Far+08] for the general class of ω-regular languages), as well as non-deterministic finite automata [Bol+09]. Other models have also been studied such as Mealy machines [Nie03; SG09], I/O automata [AV10], cover automata [Ipa12] or extended finite state machines (*i.e.* automata with data) [Cas15].

Some work have focused in the addition of continuous time in the active learning framework, notably to some subclasses of timed automata: deterministic TAs with only one clock [An+20] and deterministic event-recording automata (DERA) [GJP06; Gri08; GJL10; Lin+11], which have as many clocks as actions in the alphabet, and where each clock encodes exactly the time elapsed since the last corresponding action was taken. These classes of automata present the advantages of having a low-dimensional continuous behaviour (for 1-clock TAs) and to allow to derive the resets of the clocks directly from the observations (for DERA). Generalization to non-deterministic models has been studied in the case of real-time automata (1-clock automata whose clock is reset at each transition)[An+21].

Timed automata have also been learned via *genetic programming* [Tap+19], a search method different from active or passive learning that generates (guided) mutations on a model to obtain the desired result.

Interestingly, methods combining learning and formal methods have already been used for DERAs in the case of assume-guarantee model-checking [Lin+14]. In modular model-checking, building the model of a multi-component system is avoided by the mean of assumptions on components. Such assumptions have then to be proven correct on the component themselves. The main limitation of this method is that expert-knowledge is required to build the assumptions. In [Lin+14], the authors propose to use active-learning of DERAs [Lin+11] to automatically learn the assumptions.

Chapter 3

Control strategies for offline testing of timed systems

> Program testing can be used to show the presence of bugs, but never to show their absence!
>
> — Edsger W. Dijkstra

This chapter presents the thesis contribution on timed game theory for *difficult games i.e.* games in which a winning strategy cannot exist. This contribution is expressed in the context of black-box model-based testing.

3.1 Introduction

As explained in Section 2.2.1 of the state of the art, real-time systems are prevalent in numerous critical applications, and their correctness is often ensured in practice through testing. The main challenges that an automated test generation must overcome are the *partial observability* and *partial controllability* of implementations under test. Namely, some actions cannot be observed, which makes state estimation after a given trace complex, and as some actions are controlled by the system, notably its outputs, it may not always be possible to reach or avoid a given set of states. In the context of formal testing, inputs and outputs of the system are separated in the models and alphabets, giving rise (notably) to the model of timed automata with inputs and outputs (TAIO). This separation is used to encode the controllability issue as a formal game between tester and implementation.

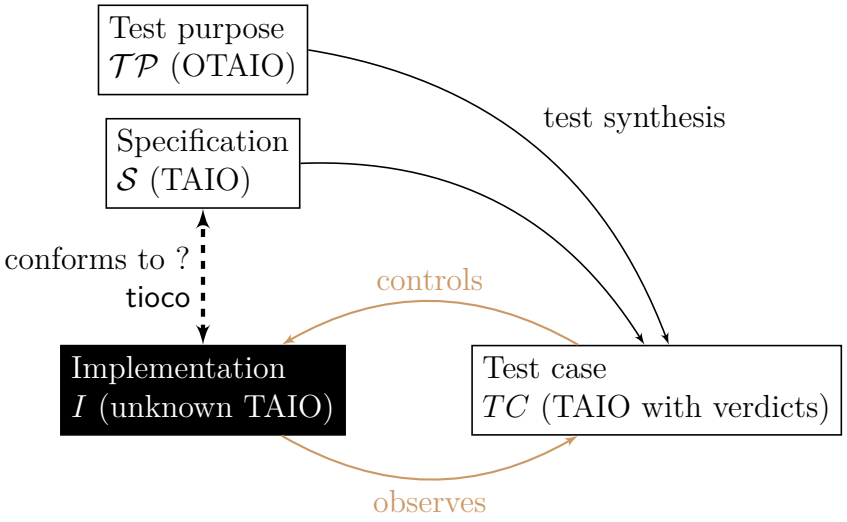


Figure 3.1: The testing framework.

3.1.1 Testing timed systems

In the present chapter, TAIOs will be used for most testing artifacts, namely specifications, implementations, and test cases. Since completeness of testing is hopeless in practice[1], it is helpful and usual to rely on test purposes that focus on some behaviours that need to be tested. We use an extension of TAIOs called Open TAIOs (or OTAIOs) [Ber+12] to formally specify them. OTAIOs play the role of non-intrusive observers of actions and clocks of peer OTAIOs, thus allowing test purposes to identify those behaviours of the specification to be tested. Formal testing also requires to define conformance as a relation between models of specifications and their possible implementations. In the timed setting, the classical tioco relation [KT09] states that, after an observable timed trace of the specification, the outputs and delays of the implementation should be possible in the specification. This testing framework is sketched in Figure 3.1. The general problem we address is to synthesize test cases from a specification, that directs the implementation towards the behaviours targeted by test purposes, with the intention that verdicts issued during the executions of the test cases on the implementation are consistent with the actual conformance between the implementation and the specification. This formal framework will be described more precisely later on. The following example motivates the controllability problem we want to address.

Example 3.1.1. *Consider the simple airport conveyor-belt described in Figure 3.2. Lug-*

1. See this chapter's epigraph.

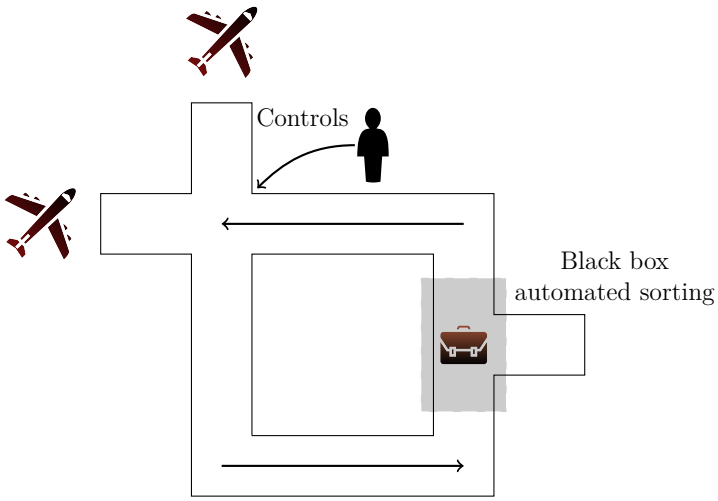


Figure 3.2: An airport conveyor belt[2].

gages arrive on the conveyor belt, and after some time they reach an automated sorting area, where the system may dispose of a luggage. If it is not removed, the luggage reaches an operator, who can choose to route it towards one of two planes. If the operator fails to decide, after some time the luggage loops on the belt and restarts the whole process.

*When testing such system from the operator's point of view, the tester would have no control over the sorting of the luggage by the belt. Possible test cases could verify that the system is able to dispose of a luggage, or that the conveyor speed is as expected (*i.e. *that time constraints are met during the system execution). When testing for such properties, the choice operated by the system may be problematic; for instance, the automatic sorting may choose not to dispose of a given luggage, although it is able to, or on the contrary it could dispose of a luggage used to check time constraints, which would not allow to verify them.*

3.1.2 Contributions and related works

The present chapter extends the work presented in [HJM18]. It adapts the game approach proposed in [Ram98] to the timed context using the framework developed in [Ber+12]. Precisely, we develop *rank lowering strategies* inspired by [Ram98] on top of the testing framework of [Ber+12]. The latter work culminated in the construction of a game between the tester and the system under test, but it did not construct the strategies of the tester

2. Briefcase icon, commercial airplane icon and person icon by Delapouite under CC BY 3.0 from `https://game-icons.net/`.

to solve the game, *i.e.*, the inputs and delays that could control as much as possible the system through behaviours targetted by the test purpose.

Compared to [Ram98], the model of TA is much more complex than finite transition systems, the test purposes are also much more powerful than simple sub-sequences of the specification considered in that work, thus even if the approach is similar, the game has to be completely revised. Furthermore, we present some fairness assumptions that identify a reasonable restriction of the implementation behaviours, under which our strategies are winning. Our model is a bit different to the one of [Dav+10], since we do not rely on observation predicates, but partial observation comes from internal actions and non-determinism. While our approach handles non-deterministic specifications and test purposes (thanks to the determinization game presented in [Ber+15]), at some point in this work we do require exact determinization to ensure some of the test-case properties. We also assume the constant reachability to a subset of so-called restart transitions that send the model back in its initial configuration. We will however explain what happens when relaxing these assumptions. In comparison, [Dav+10] avoids determinizing TAs, relying on the determinization of a finite state model, thanks to a projection on a finite set of observable predicates. Cooperative strategies of [Dav+08b] have similarities with our fairness assumptions, but their models are assumed deterministic, and their strategies cannot count the number of control losses that will be faced, just detecting that some are needed. Our approach takes controllability into account in a more complete and practical way with the reachability game and rank-lowering strategies.

An other related line of work is the thesis [Bos20], that discusses the formal relationships between games and model-based testing in the untimed case. Notably, it coins the *joker-n* strategies, that are winning when the system under tests behaves as the tester needs at n well chosen states along the execution.

The chapter is organized as follows. Section 3.2 introduces basic models: TAIOs, OTAs, OTAIOs, and then timed game automata (TGA). Section 3.3 is dedicated to the testing framework with hypotheses on models of testing artifacts, the conformance relation and the construction of the *objective-centered tester* that denotes both non-conformant traces and the goal to reach according to a test purpose. Section 3.4 constitutes the core of the chapter and the main contribution. The test-synthesis problem is interpreted as a game on the objective-centered tester. Rank-lowering strategies are proposed as candidate test cases, and a fairness assumption is introduced to make such strategies win. Then properties of test cases with respect to conformance are proved. Section 3.5 presents the algorithms

used to compute a machine-compatible symbolic representation of a strategy and some interesting properties of these algorithms.

Section 3.6 defines a notion of resistance of configurations and strategies and explains how rank-lowering strategies can be defined and used in the most general framework, where no restrictions on the models are proposed. Notably, in this setting, rank-lowering strategies are not winning.

Finally, Section 3.7 concludes the chapter with some future works and take-away.

3.2 Timed automata and timed games

In this section, we introduce our models for timed systems and for concurrent games on these objects, along with some useful notions and operations.

3.2.1 Timed automata with inputs and outputs

In order to adapt timed automata to the testing framework, we consider TAs with inputs and outputs (TAIOs), in which the alphabet is split between input, output and internal actions (the latter being used to model partial observability). We present a variation of TAs (and TAIOs) called *open* TAs (and open TAIOs) [Ber+12], in which a distinguished subset of *observed* clocks is only observed and cannot be controlled with resets. This will be useful to specify test purposes describing those behaviours of a TA (or TAIO) that require testing. TAs (and TAIOs) will be viewed as particular cases with no observed clocks.

Definition 3.2.1. *An* open timed automaton *(OTA) is a timed automaton* $\mathcal{A} = (\mathcal{L}^{\mathcal{A}}, \ell^{\mathcal{A}}_{init}, \Sigma^{\mathcal{A}}, C^{\mathcal{A}} = C^{\mathcal{A}}_p \uplus C^{\mathcal{A}}_o, \mathcal{I}^{\mathcal{A}}, E^{\mathcal{A}})$ *where the set of clocks is partitioned into* proper clocks $C^{\mathcal{A}}_p$ *and* observed clocks $C^{\mathcal{A}}_o$ *so that only proper clocks may be reset along transitions.*

Formally $E^{\mathcal{A}} \subseteq \mathcal{L}^{\mathcal{A}} \times \mathcal{G}(C^{\mathcal{A}}) \times \Sigma^{\mathcal{A}} \times 2^{C^{\mathcal{A}}_p} \times \mathcal{L}^{\mathcal{A}}$*. We will often ommit the* $\mathcal{A}$ *superscripts when the referenced timed automaton is clear from context.*

Intuitively, proper clocks C_p are controlled by $\mathcal{A}$ through resets, while observed clocks C_o can only be observed by $\mathcal{A}$ through guards and invariants, but belong to (are proper clocks of) another OTA $\mathcal{B}$ that controls them, and with which $\mathcal{A}$ is synchronized by product (see below).

Definition 3.2.2. *An Open Timed Automaton with Inputs and Outputs (OTAIO) is an OTA in which* $\Sigma^{\mathcal{A}} = \Sigma^{\mathcal{A}}_{?} \uplus \Sigma^{\mathcal{A}}_{!} \uplus \Sigma^{\mathcal{A}}_{\tau}$ *is the disjoint union of* input actions *in* $\Sigma^{\mathcal{A}}_{?}$ *(denoted by* $?a$*,* $?b$*, ...),* output actions *in* $\Sigma^{\mathcal{A}}_{!}$ *(denoted by* $!a$*,* $!b$*, ...), and* internal actions *in* $\Sigma^{\mathcal{A}}_{\tau}$ *(denoted by* τ_1*,* τ_2*, ...). We write* $\Sigma_{\mathsf{obs}} = \Sigma_{?} \uplus \Sigma_{!}$ *for the alphabet of* observable *actions.*

We call TAIO an OTAIO with no observed clocks.

TAIOs will be sufficient to model most objects of the testing framework, but the ability of OTAIOs to observe other clocks will be essential to specify test purposes (see Section 3.3.2), which need to synchronize with the specification to focus on behaviours that need to be tested.

The semantics of TAs need to be expanded for OTAs to take into account the clock resets of observed clocks, that are not specified by the OTA but will be observed along with discrete actions.

Definition 3.2.3. *Let* $\mathcal{A} = (\mathcal{L}^{\mathcal{A}}, \ell^{\mathcal{A}}_{init}, \Sigma^{\mathcal{A}}, C^{\mathcal{A}}_{p} \uplus C^{\mathcal{A}}_{o}, \mathcal{I}^{\mathcal{A}}, E^{\mathcal{A}})$ *be an OTA. Its* semantics *is defined as an infinite-state transition system* $\mathcal{T}^{\mathcal{A}} = (S^{\mathcal{A}}, s^{\mathcal{A}}_{init}, \Gamma^{\mathcal{A}}, \rightarrow^{\mathcal{A}})$ *that differs from the one of a timed automaton in the following ways:*

- $\Gamma^{\mathcal{A}} = \mathbb{R}_{\geq 0} \uplus (E^{\mathcal{A}} \times 2^{C^{\mathcal{A}}_{o}})$ *integrates observed clock resets to the transitions labels;*
- *the transitions corresponding to discrete moves take observed clocks into account* i.e.

$$((\ell, v), (e^{\mathcal{A}}, C'^{\mathcal{A}}_{o}), (\ell', v_{[C'_p \cup C'_o \leftarrow 0]})) \in S^{\mathcal{A}} \times (E^{\mathcal{A}} \times 2^{C^{\mathcal{A}}_{o}}) \times S^{\mathcal{A}} \ .$$

Example 3.2.4. *Figure 3.3 is an example of TAIO specifying a conveyor belt such as the one of Example 3.1.1. It uses one proper clock* c *(no observed one) that is used in invariants to bound the sojourn time in locations. It has as inputs* $?\mathsf{ship}_1$ *and* $?\mathsf{ship}_2$*, and a special restart action* $?\zeta$ *(see later), outputs* $!\mathsf{end}_1$*,* $!\mathsf{end}_2$*,* $!\mathsf{past}$*,* $!\mathsf{waste}$*, and internal action* τ*. After a maximum of 2 time units in location Start (depending for example on their weight), packages reach a sorting point in location Sort, where they are automatically sorted between packages to reject and packages to ship. Packages to reject go to location Waste, while packages to ship are sent to a boarding platform (location Boarding), where an operator can send them to two different destinations* Dest_1 *or* Dest_2*. If the operator takes more than 3 time units to select a destination, the package goes past the boarding platform and restarts the process. The restart action* $?\zeta$ *also allows to go back to Start from several locations.*

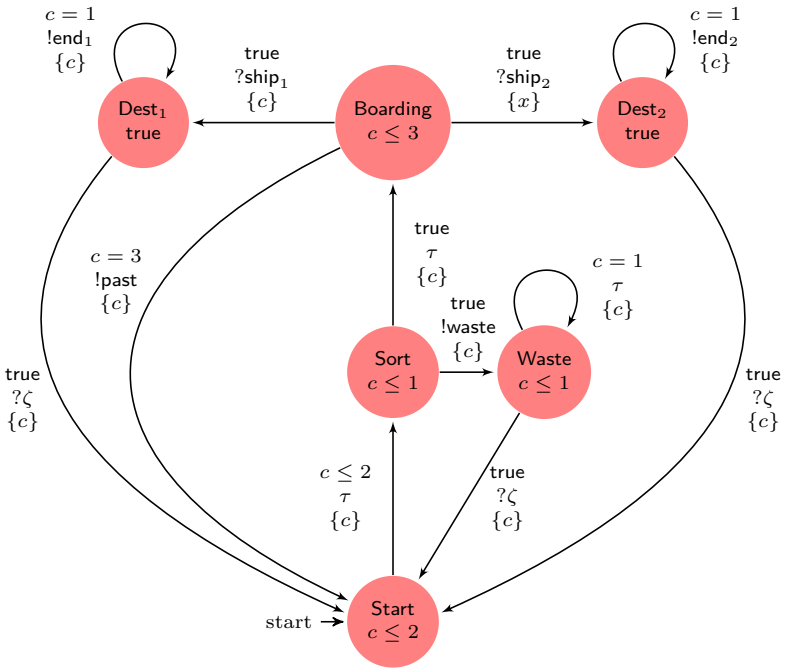


Figure 3.3: A TAIO specifying a conveyor belt.

In the sequel, we only consider infinite runs that are non-Zeno *i.e.* have an infinite duration. This restricts the set of theoretically possible runs of some timed automata, but only suppresses runs that have no interest for test-generation: either a Zeno run expresses a finite delay as an infinite sum of delays converging to that value - which is clearly a mathematical artifact - or realises an infinite number of discrete actions in finite time, which is impossible for most systems under test in practice.

Example 3.2.5. *A simple Zeno run is* $\rho = (s_i, \frac{1}{2^i}, s_{i+1})_{1 \leq i < \infty}$ *that simply corresponds to a delay of* 1 *time unit. As will be explained later in this chapter, separations between consecutive delays in a run will come from so called "decision points" for a strategy. Having an infinite number of them in finite time is thus impossible, as it would require an infinite amount of computations.*

Similarly, the timed automaton in Figure 3.4 excluding Zeno runs, ensure that every run ends in ℓ_2*, while infinite Zeno runs could oscillate between* ℓ_{init} *and* ℓ_1 *by playing a an infinite number of times in less than* 1 *time unit.*

A finite run is *accepted* in a set of locations $F \subseteq \mathcal{L}^{\mathcal{A}}$ if its last configuration belongs to $F \times \mathbb{R}_{\geq 0}$. We denote by $\mathsf{Ex}_F(\mathcal{A})$ the subset of runs accepted in F (note that $\mathsf{Ex}(\mathcal{A}) = \mathsf{Ex}_{L^{\mathcal{A}}}(\mathcal{A})$).

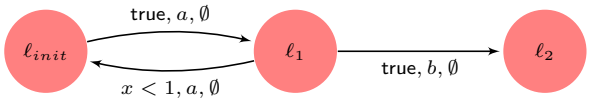


Figure 3.4: A timed automaton displaying Zeno-runs with infinitely many discrete actions.

When discussing signatures for OTAIOs, the resets of observed clocks is preserved: $\mathsf{proj}((\ell, g, a, C'_p, \ell'), C'_o)) = (a, C'_p \uplus C'_o)$. The intuition is that signatures should only abstract informations linked to the *locations* and guards, while preserving everything necessary to identify the behaviour, and observed clocks resets are necessary to deduce the valuations.

Finally, when constructing traces of runs of OTAIOs, and more generally automata with internal actions, these actions are ignored, since they cannot be observed by the environment. For $t_1, \ldots, t_k \in \mathbb{R}_{\geq 0}$, $\tau \in \Sigma_\tau$ $C' \subseteq C$ and a partial signature μ:

$$\mathsf{trace}(t_1, \ldots, t_k \cdot (\tau, C') \cdot \mu) = (\textstyle\sum_{i=1}^{k} t_i) \cdot \mathsf{trace}(\mu)$$

We furthermore define, for an OTAIO $\mathcal{A}$, a trace σ and a configuration s:

- $\mathcal{A}$ after $\sigma = \{s \in S \mid \exists \mu \in \mathsf{Sig}(\mathcal{A}), s_{init} \xrightarrow{\mu} s \wedge \mathsf{trace}(\mu) = \sigma\}$ is the set of all configurations that can be reached when the trace σ has been observed from $s^{\mathcal{A}}_{init}$;
- $\mathsf{elapse}(s) = \{t \in \mathbb{R}_{\geq 0} \mid \exists \mu \in (\mathbb{R}_{\geq 0} \cup (\Sigma_\tau \times 2^C))^*, s \xrightarrow{\mu} \wedge\ \mathsf{dur}(\mu) = t\}$ is the set of delays that can be observed from location s without observing any action;
- $\mathsf{out}(s) = \{a \in \Sigma_! \mid \exists e \in \mathsf{enab}(s), \mathsf{act}(e) = a\} \cup \mathsf{elapse}(s)$ is the set of possible outputs and delays that can be observed from s. For $S' \subseteq S$, we write $\mathsf{out}(S') = \bigcup_{s \in S'} \mathsf{out}(s)$;
- $\mathsf{in}(s) = \{a \in \Sigma_? \mid \exists e \in \mathsf{enab}(s), \mathsf{act}(e) = a\}$ is the set of possible inputs that can be proposed when arriving in s. For $S' \subseteq S$, we write $\mathsf{in}(S') = \bigcup_{s \in S'} \mathsf{in}(s)$.

The notion of enabled transitions and delay transitions are extended to regions as follows: for a region reg, we let $\mathsf{enab}((\ell, reg)) = \mathsf{enab}((\ell, v))$ for any v in reg[3]; for any transition e, we let $\mathsf{next}(e, reg)$ be the unique region reg' such that for all $s \in reg$, $s \xrightarrow{e} s'$ implies $s' \in reg'$, and we write $\mathsf{SucTemp}(reg)$ for all strict time-successor regions reg' of reg. We also extend the notion of execution to regions[4].

3. This definition is valid because the two first conditions in Def. 1.3.1 ensure that guards cannot distinguish between valuations in a given region. Hence the enabled transitions are the same.

4. Observe that writing $reg \xrightarrow{t} reg'$ for a delay t is execution-specific, as that delay may lead to a region $reg'' \neq reg'$ from some configurations in reg.

The notion of determinism and determinizability defined for TAs is generalized to OTAIOs, and we define some more useful sub-classes of OTAIOs. An OTAIO $\mathcal{A}$ is said

- *complete* if any action can be played from any configuration, formally $S = \mathcal{L} \times \mathbb{R}^C_{\geq 0}$ (*i.e.*, all invariants are always true) and for any location $\ell \in \mathcal{L}$ and action $a \in \Sigma$, $\bigvee_{(\ell,g,a,C',\ell') \in E^{\mathcal{A}}} g = \mathsf{true}$;

- *input-complete* if any input can be taken from any configuration, formally for any location $\ell \in \mathcal{L}$ and action $a \in \Sigma_?$, $\bigvee_{(\ell,g,a,C',\ell') \in E^{\mathcal{A}}} g = \mathsf{true}$;

- *non-blocking* if it does not block time waiting for an input, *i.e.* for any $s \in S^{\mathcal{A}}$ and any non-negative real t, there is a partial run ρ from s involving no input actions (*i.e.*, $\mathsf{proj}(\rho)$ is a sequence over $\mathbb{R}_{\geq 0} \cup (\Sigma_! \cup \Sigma_\tau) \times 2^C$) and such that $\mathsf{dur}(\rho) = t$;

- *repeatedly observable* if from any state some action can be observed, formally for any $s \in S$, there exists a partial run ρ from s such that $\mathsf{trace}(\rho) \notin \mathbb{R}_{\geq 0}$.

Remark 3.2.6. *As for determinism in the case of TAs, (input-)completeness is defined in a syntactic way, with the semantic versions being:*

Completeness: *from any reachable state, any delay and action can be taken,* i.e., $S = \mathcal{L} \times \mathbb{R}^C_{\geq 0}$ *(*i.e.*, all invariants are always true) and for any state $s \in \mathsf{Reach}(\mathcal{A})$ and any action $a \in \Sigma$, it holds $s \xrightarrow{(a,C')}$ for some $C' \subseteq C$;*

Input-completeness: *for any $s \in \mathsf{Reach}(\mathcal{A})$, $\mathsf{in}(s) = \Sigma_?$.*

As for determinism, the semantic notions are implied by the syntactic ones, but not equivalent.

The product of two OTAIOs extends the classical product of TAs.

Definition 3.2.7. *Given two OTAIOs $\mathcal{A} = (\mathcal{L}^{\mathcal{A}}, \ell^{\mathcal{A}}_{init}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{A}}_p \uplus C^{\mathcal{A}}_o, \mathcal{I}^{\mathcal{A}}, E^{\mathcal{A}})$ and $\mathcal{B} = (\mathcal{L}^{\mathcal{B}}, \ell^{\mathcal{B}}_{init}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{B}}_p \uplus C^{\mathcal{B}}_o, \mathcal{I}^{\mathcal{B}}, E^{\mathcal{B}})$ over the same alphabets, their* product *is the OTAIO $\mathcal{A} \times \mathcal{B} = (\mathcal{L}^{\mathcal{A}} \times \mathcal{L}^{\mathcal{B}}, (\ell^{\mathcal{A}}_{init}, \ell^{\mathcal{B}}_{init}), \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, (C^{\mathcal{A}}_p \cup C^{\mathcal{B}}_p) \uplus ((C^{\mathcal{A}}_o \cup C^{\mathcal{B}}_o \setminus (C^{\mathcal{A}}_p \cup C^{\mathcal{B}}_p)), \mathcal{I}, E)$ where $E \colon (\ell_1, \ell_2) \mapsto \mathcal{I}^{\mathcal{A}}(\ell_1) \wedge \mathcal{I}^{\mathcal{B}}(\ell_2)$ and E is the (smallest) set such that for each $(\ell^1, g^1, a, C'^1_p, \ell'^1) \in E^{\mathcal{A}}$ and $(\ell^2, g^2, a, C'^2_p, \ell'^2) \in E^{\mathcal{B}}$, E contains $((\ell^1, \ell^2), g^1 \wedge g^2, a, C'^1_p \cup C'^2_p, (\ell'^1, \ell'^2))$.*

Intuitively, proper clocks of the product are proper clocks of one of the operands, while observed clocks are observed clocks of an operand which are not proper of the other. Two transitions carrying a common action synchronize if guards intersect and only proper clocks of the product can be reset. Notice that if a proper clock of $\mathcal{A}$ is an observed clock of $\mathcal{B}$, its evolution is observed by guards of $\mathcal{B}$, and vice versa. Then it is not difficult to see that the product of two OTAIOs corresponds to the intersection of the signatures of the individual OTAIOs, *i.e.* $\mathsf{Sig}(\mathcal{A} \times \mathcal{B}) = \mathsf{Sig}(\mathcal{A}) \cap \mathsf{Sig}(\mathcal{B})$ [Ber+15]. Moreover if TAs are equipped with accepting locations $F^{\mathcal{A}} \subseteq \mathcal{L}^{\mathcal{A}}$ and $F^{\mathcal{B}} \subseteq \mathcal{L}^{\mathcal{B}}$, we also get $\mathsf{Sig}_{F^{\mathcal{A}} \times F^{\mathcal{B}}}(\mathcal{A} \times \mathcal{B}) = \mathsf{Sig}_{F^{\mathcal{A}}}(\mathcal{A}) \cap \mathsf{Sig}_{F^{\mathcal{B}}}(\mathcal{B})$.

3.2.2 Timed games

We introduce timed game automata [Asa+98], which we later use to formalize the interactions between the tester and the system under test.

Definition 3.2.8. *A* timed game automaton *(TGA) is a timed automaton* $\mathcal{A}_g = (\mathcal{L}, \ell_{init}, \Sigma_{co} \uplus \Sigma_{uco}, C, \mathcal{I}, E)$ *where* $\Sigma = \Sigma_{co} \uplus \Sigma_{uco}$ *is partitioned into actions that are controllable by the player* (Σ_{co})*, and those that are not* (Σ_{uco})*.*

Intuitively, the tester is the player for which we will try to find winning strategies against its opponent, the system under test. All the notions of runs and signatures defined previously for TAs are extended to TGAs, with the interpretation of Σ_{co} as inputs controlled by the environment and Σ_{uco} as outputs controlled by the system.

Notice that in this work we consider games that are built observable and deterministic. We can thus define strategies in terms of runs. Indeed, any system output can only correspond to a unique transition from any given configuration. Hence, one can always know in which configuration the tester is (because our strategies are deterministic too) and it is sufficient to define strategies on runs and configurations (instead of set of runs / configurations for non-deterministic games).

Definition 3.2.9. *Let* $\mathcal{A}_g = (\mathcal{L}, \ell_{init}, \Sigma_{co} \uplus \Sigma_{uco}, C, \mathcal{I}, E)$ *be a TGA. A* strategy *for the player is a partial function* $f \colon \mathsf{Ex}(\mathcal{A}_g) \to \mathbb{R}_{\geq 0} \times (\Sigma_{co} \cup \{\bot\}) \setminus \{(0, \bot)\}$ *such that for any finite run* ρ*, letting* $f(\rho) = (t, a)$*,* $t \in \mathsf{elapse}(\mathsf{last}(\rho))$ *is a possible delay from* $\mathsf{last}(\rho)$*, and there is an* a*-transition available from the resulting configuration (unless* $a = \bot$*).*

The special action $\bot$ is used to model situations where the player only wants to spend some delay: when waiting for the adversary to take a move, or when the player has

already won. We do not allow strategies to output $(0, \bot)$, as it would amount to instantly recompute a strategy, and would only loop until time delays, which is not desirable for a decision-making function. Strategies give rise to sets of runs of $\mathcal{A}_g$, defined as follows:

Definition 3.2.10. *Let $\mathcal{A}_g = (\mathcal{L}, \ell_{init}, \Sigma_{co} \uplus \Sigma_{uco}, C, \mathcal{I}, E)$ be a TGA and f be a strategy over $\mathcal{A}_g$. The* set of outcomes *of f from s_{init}, denoted by $\mathsf{Outcome}(f, s_{init})$ (we might omit to mention s_{init} when it is clear from the context), is the smallest subset of partial runs starting from s_{init} containing the empty partial run (whose last configuration is s_{init}), and s.t. for any $\rho \in \mathsf{Outcome}(f)$, letting $f(\rho) = (t, a)$ and $\mathsf{last}(\rho) = (\ell, v)$, we have*

- $\rho \cdot ((\ell, v), t', (\ell, v + t')) \cdot ((\ell, v + t'), e, (\ell', v')) \in \mathsf{Outcome}(f)$ *for any* $0 \leq t' \leq t$ *and* $\mathsf{act}(e) \in \Sigma_{uco}$ *such that* $((\ell, v + t'), e, (\ell', v')) \in \mathsf{pEx}(\mathcal{A}_g)$*;*
- *and*
  - *either $a = \bot$, and $\rho \cdot ((\ell, v), t, (\ell, v + t)) \in \mathsf{Outcome}(f)$;*
  - *or $a \in \Sigma_{co}$, and $\rho \cdot ((\ell, v), t, (\ell, v + t)) \cdot ((\ell, v + t), e, (\ell', v')) \in \mathsf{Outcome}(f)$ with $\mathsf{act}(e) = a$.*

An infinite partial run is in $\mathsf{Outcome}(f)$ if infinitely many of its finite prefixes are.

The outcomes of f characterize the subset of runs of $\mathcal{A}_g$ that can be observed while following the strategy f. The first point corresponds to the system taking an uncontrollable transition while the tester waits. It also includes a race for actions between the system and the tester, when $t' = t$. The second point adds the runs corresponding to the actions selected by the tester, when the system does not interfere.

In this chapter, we will be interested in reachability winning conditions (under particular conditions). In the untimed setting, the set of winning configurations can be computed iteratively, starting from the target locations and computing controllable predecessors in a backward manner. In a timed setting, the computation can be performed on regions, so that it terminates (in exponential time) [Asa+98; Cas+05]. In Section 3.4 we adapt this approach to our test-generation framework, after presenting it in Section 3.3. The main idea of this framework is to build a fully-observable deterministic TAIO synthesizing all the information from the specification and the test purpose, which we call *objective-centered tester*. This tester is then interpreted as a game in order to apply the backward strategy computation.

3.3 Testing framework

We now present the testing framework, defining *(i)* the main testing artifacts, *i.e.*, specifications, implementations, test purposes, and test cases, along with the assumptions we put on them; *(ii)* a conformance relation relating implementations and specifications. The combination of the test purposes and the specification and the construction of an approximate deterministic tester is explained afterwards.

3.3.1 Framework overview

We first give in Figure 3.5 an overview of the framework detailed in the rest of this section. Conformance testing aims at checking whether some black-box implementation conforms to its specification. In formal testing, the specification is given by a formal model and it is assumed that implementations, which are real objects, behave like some unknown model. Conformance is formalized by a relation linking implementations to specifications. This conformance relation is checked by test cases, which interact with the implementation through controls and observations. Test cases are also given by formal models, synthesized from the specification and from test purposes that target some particular behaviours one would like to test. It is then important that some properties can be proved about the accuracy of test cases to detect (non-)conformance. In the following, a particular formalization is presented in the context of timed systems.

3.3.2 Test context

We use TAIOs as models for specifications and implementations, TGAs and strategies for test cases, and OTAIOs for test purposes. This allows us to define expressive test purposes (as we have a timed automaton to describe objectives and conditions) while remaining abstract with respect to some of the system details given in the specification. On a technical side, it gives a unity to the objects we manipulate. In formal testing, the main objective is to detect non-conformances, which correspond to ***Fail*** verdicts. If a test purpose is given, another objective, if no non-conformances are detected, is to exercise behaviours targetted by the test purpose, which are figured out by the ***Pass*** verdict. But due to lack of control of the tester upon the implementation, one may miss those targetted behaviours, giving rise to ***Inconclusive*** verdicts. In order to enforce the occurrence of conclusive verdicts (*i.e.*, ***Fail*** or ***Pass***), we equip specifications with *restart transitions*, corresponding to a

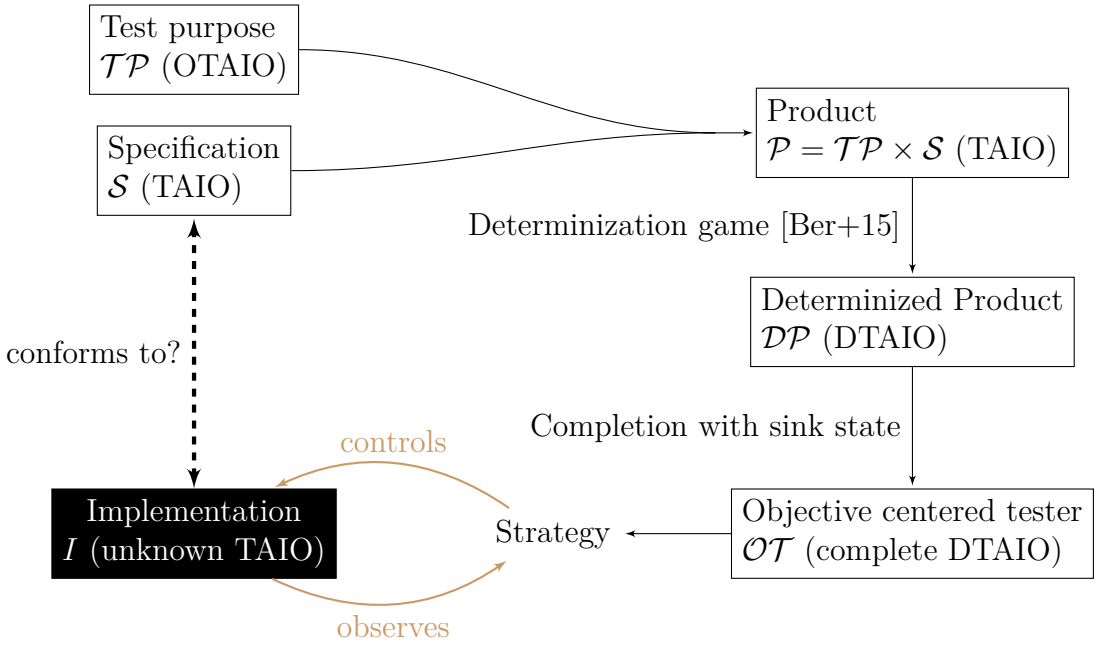


Figure 3.5: The testing framework.

kind of system shutdown and restart, and assume that from any (reachable) configuration, a restart transition is always reachable.

Definition 3.3.1. *A* specification with restarts *(or simply* specification*) on* $(\Sigma_?, \Sigma_!, \Sigma_\tau)$ *is a non-blocking, repeatedly-observable TAIO* $\mathcal{S} = (\mathcal{L}^{\mathcal{S}}, \ell^{\mathcal{S}}_{init}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{S}}_p, \mathcal{I}^{\mathcal{S}}, E^{\mathcal{S}})$*, where* $\zeta \notin \Sigma_?$ *is the restart action. We let* $\mathsf{Restart}^{\mathcal{S}} = E^{\mathcal{S}} \cap (\mathcal{L}^{\mathcal{S}} \times \mathcal{G}_{M^{\mathcal{S}}}(C^{\mathcal{S}}) \times \{\zeta\} \times \{C^{\mathcal{S}}_p\} \times \{\ell^{\mathcal{S}}_{init}\})$ *be the set of* ζ*-transitions, and assume that from any reachable configuration, there exists a finite partial execution containing* ζ*,* i.e. *for any* $s \in \mathsf{Reach}(\mathcal{S})$*, there exists* μ *s.t.* $s \xrightarrow{\mu \cdot \zeta} s^{\mathcal{S}}_{init}$*.*

The non-blocking hypothesis rules out "faulty" specifications having no conformant physically-possible implementation. Indeed, a blocking implementation should be able to stop time to wait for an input that it does not control. As this is impossible, implementations are non-blocking, and so should their specifications be. Repeated-observability will be useful for technical reasons, when analyzing exhaustiveness of test cases. It intuitively ensures that some output will always eventually be visible, allowing to detect the current configuration of the system. Our assumption on ζ-transitions entails:

Proposition 3.3.2. *Let* $\mathcal{S}$ *be a specification with restarts. Then* $Reach(\mathcal{T}_{\mathcal{S}})$ *is strongly-connected.*

Proof. Let s be a configuration of $\mathcal{T}_{\mathcal{S}}$ reachable from $s^{\mathcal{S}}_{init}$. By hypothesis, there exists a finite partial execution starting in s whose trace contains ζ. This trace leads to the configuration $s^{\mathcal{S}}_{init}$ hence any reachable configuration of $\mathcal{T}_{\mathcal{S}}$ is reachable from s, and we conclude that the reachable part of $\mathcal{T}_{\mathcal{S}}$ is strongly-connected. □

Being strongly connected ensures that there is always a possibility to reach any (reachable) configuration, and will help us ensure that a strategy always leads to a conclusive verdict, as it cannot be stuck in a situation where no path exists to a reachability goal. We discuss the consequences of releasing the strong connectivity hypothesis in Remark 3.4.11.

We now introduce the formalization of test purposes. In practice, test purposes are used to describe the intention behind test cases, typically behaviours one wants to test because they describe basic functionalities that must be correct and/or because an error is suspected. In our formal testing framework, we describe them with OTAIOs that observe the specification (its actions and clocks) and use accepting locations to define behaviours to be tested.

Definition 3.3.3. *Given a specification* $\mathcal{S} = (\mathcal{L}^{\mathcal{S}}, \ell^{\mathcal{S}}_{init}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{S}}_p, \mathcal{I}^{\mathcal{S}}, E^{\mathcal{S}} \uplus \mathsf{Restart})$*, a* test purpose for $\mathcal{S}$ *is a pair* $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$ *where* $\mathcal{TP} = (\mathcal{L}^{\mathcal{TP}}, \ell^{\mathcal{TP}}_{init}, \Sigma_? \cup \{\zeta\} \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{TP}}_p \uplus C^{\mathcal{S}}_p, \mathcal{I}^{\mathcal{TP}}, E^{\mathcal{TP}})$ *is a complete OTAIO and* $\mathsf{Accept}^{\mathcal{TP}} \subseteq \mathcal{L}^{\mathcal{TP}}$ *is a subset of accepting locations; it is required that transitions carrying restart actions* ζ *in* $\mathcal{TP}$ *are of the form* $(\ell, g, \zeta, C^{\mathcal{TP}}_p, \ell^{TP}_{init})$*,* i.e.*, they reset all proper clocks and return to the initial state.*

In the following, we may simply write $\mathcal{TP}$ in place of $(\mathcal{TP}, \mathsf{Accept}^{\mathcal{TP}})$. Notice that we force test purposes to be complete because they are non-intrusive observers: they should never constrain the runs of the specification they observe, but should only label those accepted behaviours to be tested. Test purposes observe the clocks of the specification through guards, but cannot reset them. They have their own proper clocks that they can reset; those clocks may serve *e.g.* to count unspecified delays. The condition on restart transitions in test purposes encodes the fact that we do not want to test ζ (no test cases will require to make a certain number of restart to complete), and forces the test purpose to be strongly connected.

Example 3.3.4. *Figure 3.6 is a test purpose for our conveyor-belt example. It aims to test that it is possible to ship a package to destination 2 in less than* 5 *time units, while avoiding to visit* Waste*. The* Accept *set is limited to one location, named* Accept*. The test purpose has a proper clock* c_2*, and no observed clocks. We denote by* oth *(for* otherwise*) the set of transitions that reset no clocks, and are enabled for an action other than* ζ *when*

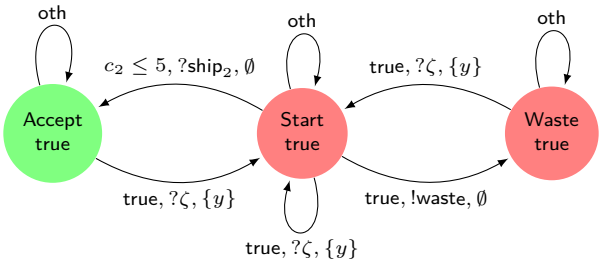


Figure 3.6: A test purpose for the conveyor belt.

no other transition is possible for this action in this location. This set serves to complete the test purpose.

In practice, conformance testing links a mathematical model, the specification, and a black-box implementation, that is a real-life *physical* object observed by its interactions with the environment. In order to formally reason about conformance, one needs to bridge the gap between the mathematical world and the physical world. We then assume that the real implementation has the same behavior as an unknown TAIO that we call *implementation*, and has the same interface as the specification $\mathcal{S}$.

Definition 3.3.5. *Let* $\mathcal{S} = (\mathcal{L}^{\mathcal{S}}, \ell^{\mathcal{S}}_{init}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{S}}_p, \mathcal{I}^{\mathcal{S}}, E^{\mathcal{S}} \cup \mathsf{Restart})$ *be a specification. An* implementation *of* $\mathcal{S}$ *is an input-complete and non-blocking TAIO* $\mathcal{I} = (\mathcal{L}^{\mathcal{I}}, \ell^{\mathcal{I}}_{init}, (\Sigma_? \cup \{\zeta\}) \uplus \Sigma_! \uplus \Sigma^{\mathcal{I}}_\tau, C^{\mathcal{I}}_p, \mathcal{I}^{\mathcal{I}}, E^{\mathcal{I}})$*. We denote by* $\mathcal{I}(\mathcal{S})$ *the set of possible implementations of* $\mathcal{S}$*.*

The hypotheses made on implementations are not restrictions, but model real-world contingencies: the environment might always provide any input and the system cannot alter the course of time.

Having defined the necessary objects, it is now possible to introduce the *timed input-output conformance* (tioco) relation [KT09]. Intuitively, it can be understood as "after any specified trace, outputs and delays of the implementation should be specified".

Definition 3.3.6. *Let* $\mathcal{S}$ *be a specification and* $\mathcal{I} \in \mathcal{I}(\mathcal{S})$*. We say that* $\mathcal{I}$ conforms to $\mathcal{S}$ *for* tioco*, and write* $\mathcal{I}$ tioco $\mathcal{S}$ *when:*

$$\forall \sigma \in \mathsf{Traces}(\mathcal{S}), \mathsf{out}(\mathcal{I} \text{ after } \sigma) \subseteq \mathsf{out}(\mathcal{S} \text{ after } \sigma)$$

3.3.3 Combining specifications and test purposes

Now that the main objects are defined, we explain how the behaviours targetted by the test purpose $\mathcal{TP}$ are characterized on the specification $\mathcal{S}$ by the construction of the product OTAIO $\mathcal{P} = \mathcal{S} \times \mathcal{TP}$. Since $\mathcal{S}$ is a TAIO and the observed clocks of $\mathcal{TP}$ are exactly the clocks of $\mathcal{S}$, the product $\mathcal{P}$ is actually a TAIO. Furthermore, since $\mathcal{TP}$ is complete, $\mathsf{Sig}(\mathcal{P}) = \mathsf{Sig}(\mathcal{S})$. By defining accepting locations in the product by $\mathsf{Accept}^{\mathcal{P}} = \mathcal{L}^{\mathcal{S}} \times \mathsf{Accept}^{\mathcal{TP}}$, we get that signatures accepted in $\mathcal{P}$ are exactly signatures of $\mathcal{S}$ accepted by $\mathcal{TP}$. Formally:

Proposition 3.3.7. *Let $\mathcal{S}$ be a specification and $\mathcal{TP}$ a test purpose on this specification, $\mathcal{P} = \mathcal{S} \times \mathcal{TP}$ their product. Then*

$$\mathit{Sig}(\mathcal{P}) = \mathit{Sig}(\mathcal{S}) \text{ and } \mathit{Sig}_{\mathit{Accept}^{\mathcal{P}}}(\mathcal{P}) = \mathit{Sig}(\mathcal{S}) \cap \mathit{Sig}_{\mathit{Accept}^{\mathcal{TP}}}(\mathcal{TP})$$

Proof. Remember that the set of signatures of the product of two OTAIOs is the intersection of the signatures of the two original OTAIOs [Ber+15], so that $\mathsf{Sig}(\mathcal{S} \times \mathcal{TP}) = \mathsf{Sig}(\mathcal{S}) \cap \mathsf{Sig}(\mathcal{TP})$; since $\mathcal{TP}$ is complete (it cannot prevent any signature of $\mathcal{S}$), $\mathsf{Sig}(\mathcal{TP}) = (\mathbb{R}_{\geq 0} \cup (\Sigma \times 2^{C_p \cup C_o}))^*$, we conclude that $\mathsf{Sig}(\mathcal{S} \times \mathcal{TP}) = \mathsf{Sig}(\mathcal{S})$.

We also have $\mathsf{Sig}_{\mathcal{L}^{\mathcal{S}} \times \mathsf{Accept}(\mathcal{TP})}(\mathcal{S} \times \mathcal{TP}) = \mathsf{Sig}_{\mathcal{L}^{\mathcal{S}}}(\mathcal{S}) \cap \mathsf{Sig}_{\mathit{Accept}^{\mathcal{TP}}}(\mathcal{TP})$ thus $\mathsf{Sig}_{\mathsf{Accept}(\mathcal{P})}(\mathcal{P}) = \mathsf{Sig}(\mathcal{S}) \cap \mathsf{Sig}_{\mathit{Accept}^{\mathcal{TP}}}(\mathcal{TP})$ □

By projection on traces, we immediately get:

Corollary 3.3.8. *Let $\mathcal{S}$ be a specification, $\mathcal{TP}$ a test purpose, and $\mathcal{P}$ their product. $\mathcal{S}$ and $\mathcal{P}$ are trace-equivalent.*

This entails that $\mathcal{I}$ tioco $\mathcal{S}$ if, and only if, $\mathcal{I}$ tioco $\mathcal{P}$. Observe in particular that ζ of $\mathcal{S}$ synchronize with ζ of $\mathcal{TP}$, which are available everywhere. This induces that the product is also strongly-connected, as shown below.

Corollary 3.3.9. *Let $\mathcal{S}$ be a specification, $\mathcal{TP}$ a test purpose, $\mathcal{P}$ their product and $\mathcal{T}_{\mathcal{P}}$ its associated timed transition system. The reachable part of $\mathcal{T}_{\mathcal{P}}$ is strongly-connected.*

Proof. This proof is derived from the proof of Prop. 3.3.2. Although they are really close, we have to do it again as the product does not ensure any relation on the semantics.

Let $((\ell^1, \ell^2), v)$ be a reachable configuration of $\mathcal{T}_{\mathcal{P}}$. There exists a finite partial execution of $\mathcal{S}$ starting in (ℓ^1, v) whose trace contains ζ, and thus by Corollary 3.3.8 there exists a

finite partial execution starting in $((\ell^1, \ell^2), v)$ whose trace contains ζ. Hence this transition leads to the configuration $s_{init} = ((\ell^{\mathcal{S}}_{init}, \ell^{\mathcal{TP}}_{init}), \mathbf{0})$. It comes that there exists a finite partial execution from $((\ell^1, \ell^2), v)$ to s_{init}. Hence any reachable configuration of $\mathcal{T}_{\mathcal{P}}$ is reachable from $((\ell^1, \ell^2), v)$. It can then be concluded that the reachable part of $\mathcal{T}_{\mathcal{P}}$ is strongly-connected. □

Example 3.3.10. *Figure 3.7 represents the product of the conveyor-belt specification of Figure 3.3 and the test purpose of Figure 3.6. All nodes are named by the first letters of the corresponding states of the specification (first) and of the test purpose (second). The only accepting location is* (D_2, A).

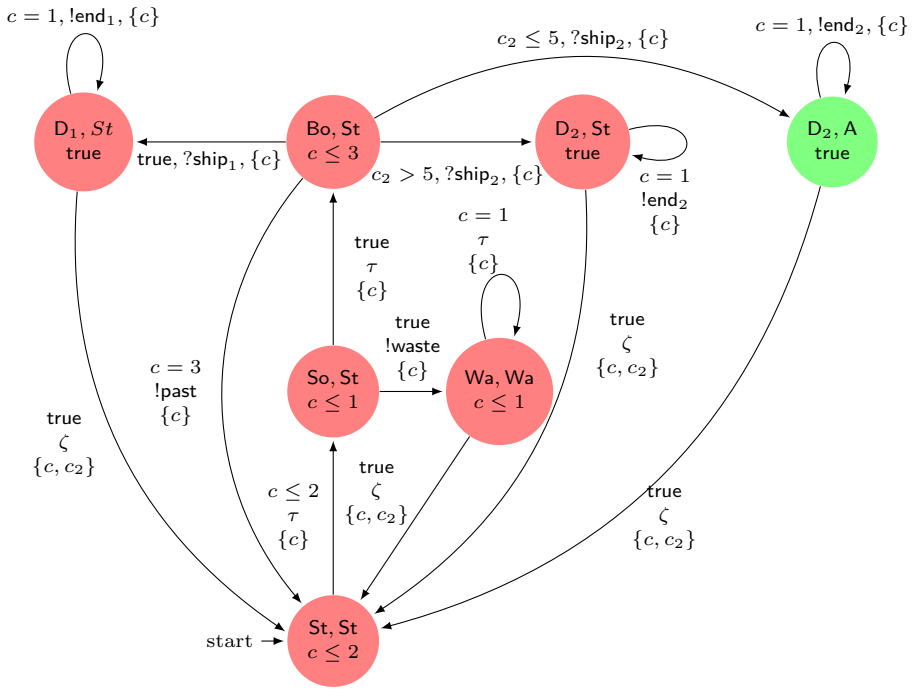


Figure 3.7: Product of the conveyor belt specification and the presented test purpose.

We make one final hypothesis: we consider only pairs of specifications $\mathcal{S}$ and test purposes $\mathcal{TP}$ whose product $\mathcal{P}$ can be exactly determinized. This restriction is necessary for technical reasons: if the determinization is only approximated (*i.e.* the deterministic product $\mathcal{DP}_a$ is not trace-equivalent to the product $\mathcal{P}$), we cannot ensure that restarts are still reachable in general and thus lose strong connectivity. As noted in Remark 3.4.11 our approach can still be of interest in this general case, but can benefit of some refinements. This is later explored in Section 3.6. One possible method to achieve determinization is

to use the determinization game presented in [Ber+15]. This determinization is known to be exact for several classes of timed automata, such as strongly-non-Zeno automata, integer-reset automata, or event-recording automata. Furthermore this game always constructs a deterministic timed automaton, and when the set of traces is approximated, it preserves tioco-conformance in the following sense: if an implementation $\mathcal{I}$ conforms to its specification $\mathcal{S}$ (equivalently it conforms to $\mathcal{P}$), then it also conforms to the approximate determinization $\mathcal{DP}_a$; in other words, non-conformances with respect to the approximation $\mathcal{DP}_a$ are still non-conformances with respect to the specification $\mathcal{S}$.

Given the product $\mathcal{P} = \mathcal{S} \times \mathcal{TP}$, let $\mathcal{DP}$ be its exact determinization. Then $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$, and the reachability of ζ transitions is preserved. We also realize the closure by Σ_τ to obtain an observable model[5]. Moreover the traces leading to $\mathsf{Accept}^{\mathcal{DP}}$ and $\mathsf{Accept}^{\mathcal{P}}$ are the same.

Example 3.3.11. *The automaton in Figure 3.8 is a deterministic approximation of the product presented in Figure 3.7. The internal transitions have collapsed, leading to an augmented* Start *location.*

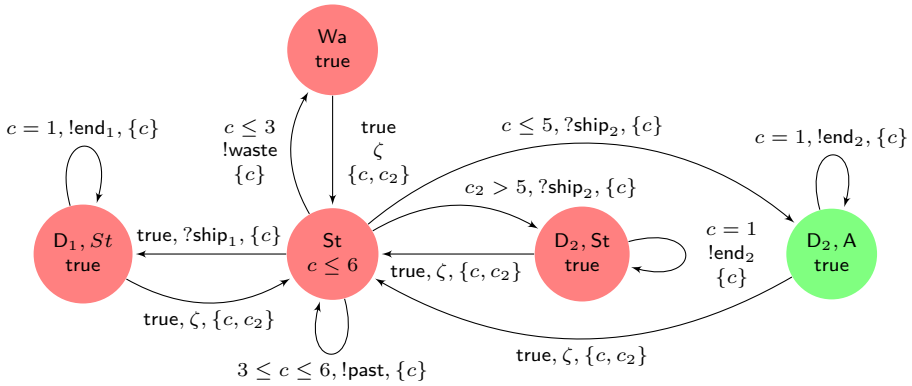


Figure 3.8: A deterministic approximation of the product.

3.3.4 Accounting for failure

At this stage of the process, we dispose of a deterministic and fully-observable TAIO $\mathcal{DP}$ having exactly the same traces as the original specification $\mathcal{S}$, and equipped with a subset of locations labelled as $\mathsf{Accept}^{\mathcal{DP}}$ which identifies traces of runs of $\mathcal{S}$ accepted by the test purpose. From this TAIO, we aim to build a tester, that can monitor the implementation, feeding it with inputs and selecting verdicts from the returned outputs.

5. The determinization procedure in [Ber+15] also realizes the closure.

In order to also explicitly model tioco-faulty behaviours (unspecified outputs and delays after a specified trace), we complete $\mathcal{DP}$ with respect to its output alphabet, by adding an explicit Fail location. We call this completed TAIO the *objective-centered tester*.

Definition 3.3.12. *Given a deterministic TAIO* $\mathcal{DP} = (\mathcal{L}^{\mathcal{DP}}, \ell^{\mathcal{DP}}_{init}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{DP}}_p, \mathcal{I}^{\mathcal{DP}}, E^{\mathcal{DP}})$*, we construct its* objective-centered tester $\mathcal{OT} = (\mathcal{L}^{\mathcal{DP}} \cup \{\mathsf{Fail}\}, \ell^{\mathcal{DP}}_{init}, \Sigma_? \uplus \Sigma_! \uplus \Sigma_\tau, C^{\mathcal{DP}}_p, \mathcal{I}^{\mathcal{OT}}, E^{\mathcal{OT}})$ *where* $\mathcal{I}^{\mathcal{OT}}(l) = \mathsf{true}$*. The set of transitions* $E^{\mathcal{OT}}$ *is defined from* $E^{\mathcal{DP}}$ *by:*

$$E^{\mathcal{OT}} = E^{\mathcal{DP}} \cup \Big(\bigcup_{\substack{\ell \in \mathcal{L}^{\mathcal{DP}} \\ a \in \Sigma_!^{\mathcal{DP}}}} \{(\ell, g, a, \emptyset, \mathsf{Fail}) \mid g \in \overline{G}_{a,l}\} \Big) \cup \Big\{ (\mathsf{Fail}, \mathsf{true}, a, \emptyset, \mathsf{Fail}) \mid a \in \Sigma^{\mathcal{DP}} \Big\}$$

where for each a *and* ℓ*,* $\overline{G}_{a,l}$ *is a set of guards complementing the set of all valuations* v *for which an* a*-transition is available from* (ℓ, v) *(notice that* $\overline{G}_{a,l}$ *generally is non-convex, so that it cannot be represented by a single guard).*

Verdicts are defined on the configurations of $\mathcal{OT}$ *as follows:*

- $\boldsymbol{Pass} = \bigcup_{\ell \in \mathsf{Accept}^{\mathcal{DP}}} (\{\ell\} \times \mathcal{I}^{\mathcal{DP}}(\ell))$*;*
- $\boldsymbol{Fail} = \{\mathsf{Fail}\} \times \mathbb{R}_{\geq 0} \cup \bigcup_{\ell \in \mathcal{L}^{\mathcal{DP}}} \left(\{\ell\} \times \left(\mathbb{R}^{C_p}_{\geq 0} \setminus \mathcal{I}^{\mathcal{DP}}(\ell) \right) \right)$.

Pass corresponds to behaviours accepted by the test purpose, while ***Fail*** corresponds to non-conformant behaviours. Notice that we do not define the usual ***Inconclusive*** verdicts (*i.e.* configurations in which we cannot conclude to non-conformance, nor accept the run with respect to the test purpose) as we will enforce the apparition of ***Pass*** or ***Fail*** thanks to the preservation of strong connectivity.

In the following, we present some of the useful properties of objective centered testers, notably what properties are kept from $\mathcal{DP}$.

First, the ζ-transitions are preserved in $\mathcal{OT}$.

Lemma 3.3.13. *Let* $\mathcal{OT}$ *be an objective-centered tester. For any location* ℓ *in* $\mathcal{L}^{\mathcal{OT}} \setminus \{\mathsf{Fail}\}$*, there exists a finite partial execution* $\rho \in \mathsf{pEx}(\mathcal{OT})$ *starting in* ℓ *and containing a* ζ*-transition.*

Proof. The same result holds from any state in $\mathcal{S}$, and $\mathsf{Traces}(\mathcal{S}) = \mathsf{Traces}(\mathcal{S} \times \mathcal{TP})$. The result follows by exact determinizability of the product $\mathcal{S} \times \mathcal{TP}$. □

This, as for previous objects, is used to ensure the strong connectivity of the semantics.

Corollary 3.3.14. *Let $\mathcal{OT}$ be an objective-centered tester and $\mathcal{T}^{\mathcal{OT}}$ its associated timed transition system. Then $\mathsf{Reach}(\mathcal{T}^{\mathcal{OT}}) \setminus \boldsymbol{Fail}$ is strongly-connected.*

Proof. The proof is the same as the one of Prop. 3.3.2, using Lemma 3.3.13. □

Lemma 3.3.13 is the reason our method assumes exact determinizability: we cannot ensure in general that a restart transition will remain reachable: if determinization is approximated, some traces might be lost, and the lemma would not hold anymore.

Observe that $\mathcal{OT}$ and $\mathcal{DP}$ model the same behaviours. Obviously, their sets of traces differ, but the traces added in $\mathcal{OT}$ precisely correspond to runs reaching ***Fail***. We now define a specific subset of runs, signatures and traces corresponding to traces that are meant to conform to the specification.

Definition 3.3.15. *A run ρ of an objective-centered tester $\mathcal{OT}$ is said* conformant *if it does not reach ***Fail***. We write $\mathsf{Ex}_{conf}(\mathcal{OT})$ for the set of conformant runs of $\mathcal{OT}$, and $\mathsf{Sig}_{conf}(\mathcal{OT})$ (resp. $\mathsf{Traces}_{conf}(\mathcal{OT})$) the corresponding signatures (resp. traces). We write $\mathsf{Ex}_{fail}(\mathcal{OT}) = \mathsf{Ex}(\mathcal{OT}) \setminus \mathsf{Ex}_{conf}(\mathcal{OT})$ and similarly for the signatures and traces.*

The conformant traces are exactly those traces specified by $\mathcal{DP}$, *i.e.* $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ and correspond to executions that are tioco-conformant with the specification; on the other hand, $\mathsf{Ex}_{\mathsf{fail}}$ are runs where a non-conformance is detected. We prove this in the following.

Proposition 3.3.16. *Let $\mathcal{DP}$ be the exact determinization of the product $\mathcal{P}$ between a specification and a test purpose, and $\mathcal{OT}$ be its associated objective-centered tester. Then*

$$\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}_{conf}(\mathcal{OT}).$$

Proof. An execution is in $\mathsf{Ex}_{\mathsf{conf}}(\mathcal{OT})$ if it avoids the ***Fail*** verdict. This amounts to avoiding location Fail and respecting the invariants of $\mathcal{DP}$. By construction of $\mathcal{OT}$, this corresponds exactly to the runs of $\mathcal{DP}$. □

We can furthermore ensure that no approximation is made, by proving that $\mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ and $\mathsf{Traces}(\mathsf{Ex}_{\mathsf{fail}}(\mathcal{OT}))$ are disjoint. As we know by construction that the executions differ, this result highlights that no confusion is made when abstracting from execution to traces.

Lemma 3.3.17. *Given an objective-centered tester $\mathcal{OT}$, we have*

$$\mathsf{Traces}_{conf}(\mathcal{OT}) \cap \mathsf{Traces}(\mathsf{Ex}_{fail}(\mathcal{OT})) = \emptyset.$$

Proof. Let $\rho \in \mathsf{Ex}_{\mathsf{fail}}(\mathcal{OT})$. Consider the longest prefix ρ' of ρ that does not reach ***Fail***, and let γ be the transition taken after ρ' in ρ. Two cases should be considered:

- If γ is a delay transition, then it violates the invariant of the location in $\mathcal{DP}$. By determinism of $\mathcal{DP}$, $\mathcal{DP}$ after $\mathsf{trace}(\rho')$ is a singleton set of states. Hence the same delay is not available after ρ' in $\mathcal{DP}$;
- If $\mathsf{act}(\gamma) \in \Sigma_!$ then this output is not specified in the current location of $\mathcal{DP}$. By determinism of $\mathcal{DP}$, $\mathcal{DP}$ after $\mathsf{trace}(\rho')$ is a singleton. Hence transition γ is not possible after ρ' in $\mathcal{DP}$.

In both cases $\mathsf{trace}(\rho) \notin \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$. As this holds for any run of $\mathsf{Ex}_{\mathsf{fail}}(\mathcal{OT})$ we have the desired property. □

It remains to say that $\mathcal{OT}$ is repeatedly-observable, except for configurations in ***Fail***.

Lemma 3.3.18. *For a repeatedly-observable specification $\mathcal{S}$, $\mathsf{Reach}(\mathcal{OT}) \setminus$ **Fail** is repeatedly-observable.*

Proof. We know that $\mathsf{Traces}(\mathcal{S}) = \mathsf{Traces}(\mathcal{P})$, hence $\mathsf{out}(\mathcal{P}\ \mathsf{after}\ \sigma) = \mathsf{out}(\mathcal{S}\ \mathsf{after}\ \sigma)$ for all $\sigma \in \mathsf{Traces}(\mathcal{P})$. As $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$ by assumption, we also know that for all $\sigma \in \mathsf{Traces}(\mathcal{DP})$, $\mathsf{out}(\mathcal{P}\ \mathsf{after}\ \sigma) \subseteq \mathsf{out}(\mathcal{DP}\ \mathsf{after}\ \sigma)$. It comes

$$\forall \sigma \in \mathsf{Traces}(\mathcal{OT}),\ \mathsf{out}(\mathcal{S}\ \mathsf{after}\ \sigma) \subseteq \mathsf{out}(\mathcal{OT}\ \mathsf{after}\ \sigma)$$

as $\mathcal{OT}$ only adds traces to $\mathcal{DP}$. Hence for all $s \in \mathsf{Reach}(S^{\mathcal{OT}}) \setminus$ ***Fail***, there exists $\mu \in \mathsf{Sig}(\mathcal{OT})$ s.t. $s \xrightarrow{\mu}$ and $\mathsf{trace}(\mu) \notin \mathbb{R}_{\geq 0}$. Indeed, there exists $\sigma \in \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ such that $s = \mathcal{OT}\ \mathsf{after}\ \sigma$ (as $\mathcal{OT}$ is deterministic outside of ***Fail***) and for $s' \in \mathcal{S}\ \mathsf{after}\ \sigma$, there exists μ' such that $s' \xrightarrow{\mu'}$ and $\mathsf{trace}(\mu') \notin \mathbb{R}_{\geq 0}$. It suffices to take $\mu \in \mathsf{pSig}(\mathcal{OT})$ such that $\mathsf{trace}(\mu) = \mathsf{trace}(\mu')$, and by the previous trace-inclusion property, such a trace exists. □

In this section we explained the construction of the objective-centered tester $\mathcal{OT}$ from the specification and a test purpose and some of its properties. It represents the most general behaviours of testers that detect both non-conformance to the specification $\mathcal{S}$ and acceptance by the test purpose $\mathcal{TP}$. In the next section, we go further and explain how to tackle controllability problems by interpreting $\mathcal{OT}$ as a game whose winning strategies will be test cases that try to avoid control losses.

3.4 Translating objective-centered testers into games

In this section, we interpret objective-centered testers as games between the tester and the implementation, and propose strategies that try to avoid control losses. We then introduce a scope in which the tester always has a winning strategy, and discuss the properties of the resulting test cases (*i.e.* game structure and built strategy).

We want to enforce conclusive verdicts when running test cases, *i.e.* either the implementation does not conform to its specification (***Fail*** verdict) or the awaited behaviour appears (***Pass*** verdict). We thus say that an execution ρ is winning for the tester if it reaches a ***Fail*** or ***Pass*** configuration and denote by $\mathsf{Win}(\mathcal{A}_g)$ the set of such executions. Observe however that ***Fail*** is entirely controlled by the implementation, which may or may not reveal non-conformances. We will thus only target ***Pass*** (while monitoring ***Fail***). In the following, we consider the TGA $\mathcal{A}_g^{\mathcal{OT}} = (\mathcal{L}^{\mathcal{OT}}, \ell_{init}^{\mathcal{OT}}, \Sigma_?^{\mathcal{OT}} \uplus \Sigma_!^{\mathcal{OT}}, C_p, \mathcal{I}^{\mathcal{OT}}, E^{\mathcal{OT}})$ where the controllable actions are the inputs $\Sigma_{co} = \Sigma_?^{\mathcal{OT}}$ and the uncontrollable actions are the outputs $\Sigma_{uco} = \Sigma_!^{\mathcal{OT}}$. Indeed, we place ourselves on the tester side, from which inputs of the system are controlled and outputs are observed.

3.4.1 Rank-lowering strategies

In this part, we restrict our discussion to TGAs where ***Pass*** configurations are reachable (when seen as plain TAs). Indeed, when this is not the case (we will discuss the fact that the proposed method can detect this), trying to construct a strategy seeking a ***Pass*** verdict is hopeless. This is a natural restriction, as it only rules out test purposes that are unsatisfiable by the specification.

To discuss partial controllability we introduce *control losses* (that will be formalised later in this section) as configurations where the system under test can propose uncontrollable outputs that move its configuration further away from ***Pass***. We define a hierarchy of configurations, depending on their "distance" to ***Pass***, based on the number of control losses one has to suffer to reach ***Pass***, and the number of transitions toward the next control loss. This uses a backward algorithm, for which we define the predecessors of a set of configuration.

Given a set of configurations $S' \subseteq S$ of $\mathcal{A}_g^{\mathcal{OT}}$, letting $\overline{V}$ denote the complement of V, we define three kinds of predecessors of S':

- discrete predecessors by a sub-alphabet $\Sigma' \subseteq \Sigma$

$$\mathsf{Pred}_{\Sigma'}(S') = \{(\ell, v) \mid \exists a \in \Sigma',\ \exists (\ell, a, g, C', \ell') \in E, v \models g \wedge (\ell', v_{[C' \leftarrow 0]}) \in S'\}$$

- timed predecessors, while avoiding a set V of configurations:

$$\mathsf{tPred}_{\leq}(S', V) = \{(\ell, v) \mid \exists t \in \mathbb{R}_{\geq 0},\ (\ell, v+t) \in S' \wedge \forall\ 0 \leq t' \leq t.\ (\ell, v+t') \notin V\}$$

  with a less constraining variant:

$$\mathsf{tPred}_{<}(S', V) = \{(\ell, v) \mid \exists t \in \mathbb{R}_{\geq 0},\ (\ell, v+t) \in S' \wedge \forall\ 0 \leq t' < t.\ (\ell, v+t') \notin V \setminus S'\}$$

  We furthermore write $\mathsf{tPred}(S') = \mathsf{tPred}_{\leq}(S', \emptyset) = \mathsf{tPred}_{<}(S', \emptyset)$;

- final timed predecessors are defined for convenience (see below):

$$\mathsf{ftPred}(S') = \mathsf{tPred}_{<}(\boldsymbol{Fail}, \mathsf{Pred}_{\Sigma_{uco}}(\overline{S'})) \cup \overline{\mathsf{tPred}(\mathsf{Pred}_{\Sigma}(\overline{S'}))}$$

The first two sets are the easier to understand: $\mathsf{Pred}_{\Sigma'}(S')$ is the set of states from which a transition carrying an action in Σ' leads to S'; $\mathsf{tPred}_{\leq}(S', V)$ and $\mathsf{tPred}_{<}(S', V)$ are the sets of states from which delaying leads to S' without entering V, with a discussion on whether reaching $S' \cap V$ is accepted (it is for $\mathsf{tPred}_{<}(S', V)$).

Remark 3.4.1. *For $\mathsf{tPred}_{<}(S', V)$, the condition $(\ell, v+t') \notin V \setminus S'$ is necessary for open zones: without it, for any open S' and V, $\mathsf{tPred}_{<}$ would be equivalent to $\mathsf{tPred}_{\leq}$.*

The final timed predecessors correspond to situations where the system is *cornered* into reaching S' unless it choses non-conformance or to remain indefinitely idle: on the left side of the union, $\mathsf{tPred}(\boldsymbol{Fail}, \mathsf{Pred}_{\Sigma_{uco}}(\overline{S'}))$ is the set of states from which the system under test only has the choice between taking an uncontrollable transition to S' (as no uncontrollable transition to $\overline{S'}$ will be available) or reach ***Fail***. On the right side are the states from which there are no transitions to anywhere but S' in the future, hence the system will end up playing a transition to S'. Intuitively, this is because the system will not remain infinitely idle when it can act (this will be formally enforced by a fairness hypothesis). Such situations are not considered as control losses, as the system can only take a beneficial transition for the tester (either by going to S' or to ***Fail***). Observe that

$\mathsf{tPred}_{\leq}$ (resp. $\mathsf{tPred}_{<}$) and ftPred need not return convex sets, but are efficiently computable using Pred and simple set constructions [Cas+05].

Now, using these notions of predecessors, a hierarchy of configurations based on the distance to ***Pass*** is defined.

Definition 3.4.2. *The sequence $(W_i^j)_{j,i}$ of sets of configurations is defined as:*

- $W_0^0 = \boldsymbol{Pass}$;
- $W_{i+1}^j = \pi(W_i^j)$ *with* $\pi(S') = \mathsf{tPred}_{<}\left(S', \mathsf{Pred}_{\Sigma_{uco}}(\overline{S'})\right) \cup \mathsf{tPred}_{\leq}\left(\mathsf{Pred}_{\Sigma_{co}}(S'), \mathsf{Pred}_{\Sigma_{uco}}(\overline{S'})\right) \cup \mathsf{ftPred}(S')$;
- $W_0^{j+1} = \pi'(W_\infty^j)$ *with* $\pi'(S') = \mathsf{tPred}(S') \cup \mathsf{Pred}_{\Sigma}(S')$ *and* W_∞^j *the limit*[6] *of the sequence* $(W_i^j)_i$.

In this hierarchy, j corresponds to the minimum number of *control losses* the tester has to go through (in the worst case) in order to reach ***Pass***, and i corresponds to the minimal number of steps before the next control loss (or to ***Pass***). The states in W_0^{j+1} are considered control losses as the implementation might take an output transition leading to an undesirable configuration (higher in the hierarchy). On the other hand, in the construction of W_{i+1}^j, the tester keeps full control, as it is not possible to reach such bad configurations with an uncontrollable transition. The computation of $(W_i^j)_i$ corresponds to finding the controllable zones, the least fix points W_∞^j, and then taking a control-loss step W_0^{j+1}. Finally, the hierarchy iterates this process and terminates on its least fix point, effectively finding all configurations coreachable from ***Pass*** and the minimal number of control losses.

Remark 3.4.3. *The use of $\mathsf{tPred}_{<}$ and $\mathsf{tPred}_{\leq}$ in the definition of W_{i+1}^j corresponds to the following intuition: for $\mathsf{tPred}_{<}(S', \mathsf{Pred}_{\Sigma_{uco}}(\overline{S'}))$, it is undesirable to force conditions on S', which is W_i^j and has been handled by its own definition. On the other hand, $\mathsf{tPred}_{\leq}(\mathsf{Pred}_{\Sigma_{co}}(S'), \mathsf{Pred}_{\Sigma_{uco}}(\overline{S'}))$ enforces that no control loss can happen in $\mathsf{Pred}_{\Sigma_{co}}(S')$.*

Notice that the sequence (W_i^j) is an increasing sequence of pairs of regions and locations, and hence can be computed in time exponential in the size of $C^{\mathcal{OT}}$ and linear in the size of $\mathcal{L}^{\mathcal{OT}}$. We then have the following property saying that the computation of (W_i^j) covers

6. The sequence $(W_i^j)_i$ is non-decreasing, and can be computed in terms of clock regions; hence the limit exists and is reached in a finite number of iterations [Cas+05].

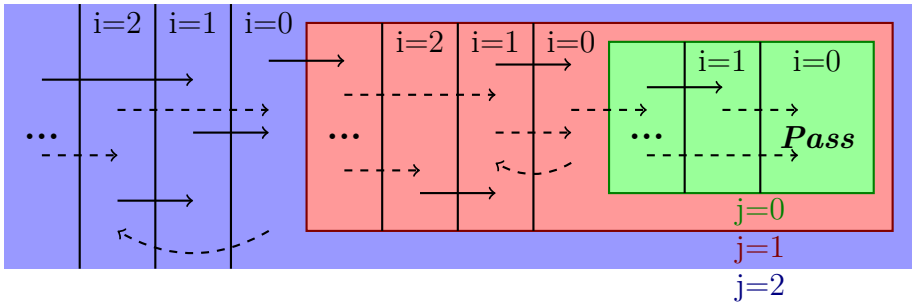


Figure 3.9: A representation of the outcomes of a rank lowering strategy

all reachable states except Fail:

Proposition 3.4.4. *There exists $i, j \in \mathbb{N}$ such that* $(\mathsf{Reach}(\mathcal{A}_g^{\mathcal{OT}}) \setminus \boldsymbol{Fail}) \subseteq W_i^j$.

Proof. Let $s \in \mathsf{Reach}(\mathcal{A}_g^{\mathcal{OT}}) \setminus \boldsymbol{Fail}$ be a reachable configuration. Since i) ***Pass*** is reachable from $s_{init}^{\mathcal{T}}$ (by hypothesis); ii) there is a partial run from s back to the initial configuration (Corollary 3.3.14), then ***Pass*** is reachable from s. Moreover, there is such a partial run with length bounded by the number of regions.

For each $s \in \mathsf{Reach}(\mathcal{A}_g^{\mathcal{OT}}) \setminus \boldsymbol{Fail}$, we fix a finite partial run to ***Pass***, and reason by induction on the length n of this partial run in order to prove that $\mathsf{conf} \in W_0^n$:

- case $n = 0$: in this case $s \in \boldsymbol{Pass} = W_0^0$;
- inductive case: we assume that the result holds for n, and take s with a partial run to ***Pass*** of length $n+1$. Then $s \xrightarrow{\gamma} s'$ for some γ with $\mathsf{act}(\gamma) \in \Gamma$, and there is a partial run from s' to ***Pass*** of length at most n, so that $s' \in W_0^n$. Hence in the worst case $s \in W_0^{n+1}$.

This proves our result. □

As explained above, this property is based on the assumption that the ***Pass*** verdict is reachable. Nevertheless, if this were not the case, it would be detected during the hierarchy construction, which would then converge to a fixpoint not including $s_{init}^{\mathcal{A}_g}$. As all the configurations in which we want to define a strategy are covered by the hierarchy, we can use it to define a preference relation (*i.e.* a total preorder) that will guide the tester to better places in terms of control and distance to ***Pass***.

Definition 3.4.5. *Let $s \in \mathsf{Reach}(\mathcal{A}_g^{\mathcal{OT}}) \setminus \boldsymbol{Fail}$. The* rank $r(s)$ *of s is the pair*

$$(j_s = \arg\min_{j \in \mathbb{N}}(s \in W_\infty^j),\ i_s = \arg\min_{i \in \mathbb{N}}(s \in W_i^{j_s}))$$

When $r(s) = (j_s, i_s)$, it holds $s \in W_{i_s}^{j_s}$ and j_s is the minimal number of control losses before reaching an accepting state, and i_s is the minimal number of steps in the strategy before the next control loss. We write $s \sqsubseteq s'$ when $r(s) \leq_{\mathbb{N}^2} r(s')$, where $\leq_{\mathbb{N}^2}$ is the lexical order on $\mathbb{N}^2$. Intuitively, it is easier to drive trajectories to ***Pass*** from s than from s'.

Proposition 3.4.6. *$\sqsubseteq$ is a total preorder on $\mathsf{Reach}(\mathcal{A}_g^{\mathcal{OT}}) \setminus$ **Fail**.*

Proof. $\sqsubseteq$ inherits transitivity and reflexivity from $\leq_{\mathbb{N}^2}$. It is not antisymmetric because several configurations may have the same rank, for example, all configurations of ***Pass*** have rank $(0, 0)$. As by Prop. 3.4.4 there exists $j, i \in \mathbb{N}^2$ such that W_i^j covers $\mathsf{Reach}(\mathcal{A}_g^{\mathcal{OT}}) \setminus$ ***Fail***, all these configurations have a rank and can be compared, hence $\sqsubseteq$ is total. □

Example 3.4.7. *In Figure 3.10, the (W_i^j) are represented on the game constructed from the $\mathcal{OT}$ corresponding to the deterministic TA presented in Figure 3.8. For clarity, the Fail location and the transitions leading to it are not represented. In this example, $W_0^0 = (D_2, A) \times \mathbb{R}_{\geq 0}$, $W_0^1 = W_0^0 \uplus St \times (c \leq 5)$ and $W_1^1 = W_0^1 \uplus St \times (5 < c \leq 6) \cup \{Wa, (D_1, St), (D_2, St)\} \times \mathbb{R}_{\geq 0}$. The **Fail** verdict corresponds to $Fail \cup (St \times (c \geq 6))$.*

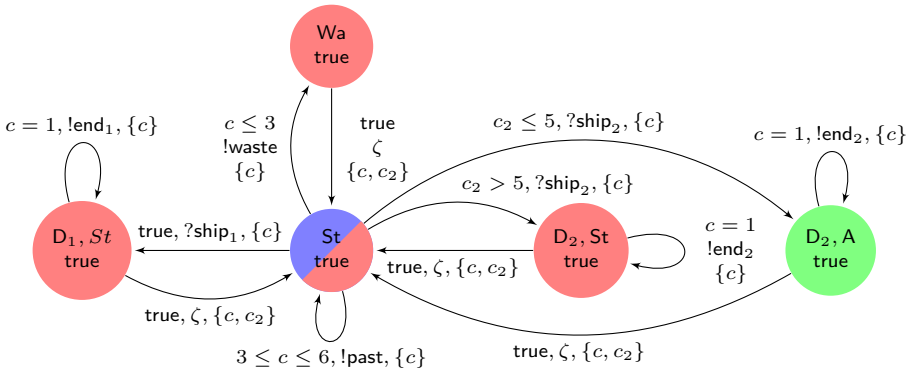


Figure 3.10: The ranks of $\mathcal{A}_g$

Example 3.4.8. *This notion of rank can be applied to many difficult games* i.e. *games where you cannot always win, even out of the testing framework. For example, the game in Figure 3.11 has the following ranks, with all configurations in ℓ_6 considered as accepting: $(0, 0)$ in $\ell_6 \times \mathbb{R}_{\geq 0}$, $(0, 1)$ in $\ell_5 \times \mathbb{R}_{\geq 0}$, $(0, 2)$ in $s_4 \times (c > 4)$, $(1, 0)$ in $\ell_3 \times \mathbb{R}_{\geq 0} \cup \ell_4 \times (c \leq 4)$, $(1, 1)$ in $\ell_2 \times (c < 3)$, $(1, 2)$ in $\ell_2 \times (c \geq 3)$, $(2, 0)$ in $\ell_0 \times (c \leq 3) \cup s_1 \times \mathbb{R}_{\geq 0}$ and $(2, 1)$ in $\ell_0 \times (c > 3)$.*

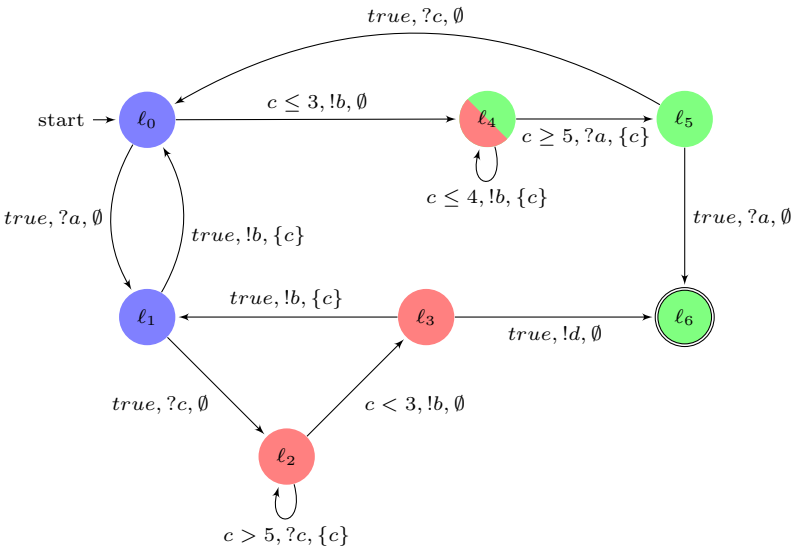


Figure 3.11: The ranks in a complex game

The notion of rank and its order relation $\sqsubseteq$ define a preference relation on configurations, with configurations in ***Pass*** being the minimal elements. We use it to define a strategy trying to decrease the rank during the execution, *i.e.* decrease in each state both the distance and the number of control losses. For any $s \in S$, we write $r^-(s)$ for the largest rank such that $r^-(s) <_{\mathbb{N}^2} r(s)$, and $W^-(s)$ for the associated set in $(W_i^j)_{j,i}$. We (partially) order pairs $(t, a) \in \mathbb{R}_{\geq 0} \times \Sigma$ according to t.

Definition 3.4.9. *A strategy f for the tester is* rank-lowering *if, for any finite run ρ with $\mathsf{last}(\rho) = s = (\ell, v)$, it selects the smallest delay satisfying one of the following constraints:*

- *if $s \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{co}}(W^-(s)))$, then $f(\rho) = (t, a)$ with $a \in \Sigma_{co}$ s.t. there exists $e \in E$ with $\mathsf{act}(e) = a$ and $s \xrightarrow{t} s' \xrightarrow{e} s''$ with $s'' \in W^-(s)$, $s' \notin W^-$ and t is minimal in the following sense: if $s \xrightarrow{t'} \xrightarrow{e'} s'''$ with $s''' \in W^-(s)$, $\mathsf{act}(e') \in \Sigma_{co}$ and $t' \leq t$, then $v + t$ and $v + t'$ belong to the same region;*

- *if $s \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{uco}}(W^-(s)))$, then $f(\rho) = (t, \bot)$ with t such that $s \xrightarrow{t} s' \notin \mathsf{Pred}_{\Sigma_{uco}}(W^-(s))$, $\exists t' < t$ $s \xrightarrow{t'} s'' \in \mathsf{Pred}_{\Sigma_{uco}}(W^-(s))$ and t is minimal if such a t exists. Else t is maximal in the same sense as above (maximal delay successor region);*

- *if $s \in \mathsf{tPred}(W^-(s))$,then $f(\rho) = (t, \bot)$ such that $s \xrightarrow{t} s'$ with $s' \in W^-(s)$, and t is minimal in the same sense as above;*

- *otherwise* $f(\rho) = (t, \bot)$ *where* t *is maximal in the same sense as above (maximal delay-successor region).*

The first three cases follow the construction of the (W_i^j) and propose the shortest behaviour leading to W^-. The fourth case corresponds either to a configuration of ***Pass***, where W^- is undefined, or to a ftPred leading to ***Fail***, cases in which the tester has won. Interestingly, (possibly several) rank-lowering strategies always exist. The constraints of rank-lowering strategies allow the creation of *memoryless* strategies, *i.e.* strategies that only rely on local information without keeping the history in memory. Indeed, they only use local information, aggregated during the incremental construction of sets (W_i^j). Hence, a straightforward implementation only using the sets (W_i^j) is memoryless.

Example 3.4.10. *An example of a rank-lowering strategy on the automaton of Figure 3.10 is: in* (D_2, A)*, play* $\bot$ *(as* W_0^0 *has been reached); in* St*, play* $(0, \mathsf{ship}_2?)$ *from* W_0^1*, otherwise play* $(1, \bot)$*. In any other state, play* $(0, \zeta)$*. The omission of* Fail *in Figure 3.10 does not impact the strategy representation:* ***Fail*** *is a winning set of configurations, but is not targetted by the strategies.*

Remark 3.4.11. *It is worth noting that even in a more general setup where the models are not equiped with* ζ*-transitions, as in [Ber+15], rank-lowering strategies may still be useful: as they are defined on the co-reachable set of* ***Pass****, they can still constitute test cases, and the configurations where they are not defined are exactly the configurations corresponding to a* ***Fail*** *verdict or to an* ***Inconclusive*** *verdict,* i.e. *no conclusions can be made since an accepting configuration cannot be reached.*

Yet, rank-lowering strategies would gain to be refined in this case in order to avoid ***Inconclusive*** *verdicts as much as possible. Indeed, as defined in this chapter, a rank lowering strategy would choose a path that is shorter but more prone to inconclusivity over a longer but safer path. This can be handled by adding a new layer to the ranks encoding the distance to an inconclusive verdict (*i.e. *the minimum number of uncontrollable actions required to reach one) over the path to* ***Pass****. Strategies are defined in a similar fashion, but in this general case we cannot ensure that they win, as they could sometimes reach* ***Inconclusive****.*

We detail this generalization in Section 3.6.

3.4.2 Making rank-lowering strategies win

A rank-lowering strategy is generally not a winning strategy: it relies on the implementation fairly exploring its different possibilities and not repeatedly avoiding an enabled transition. In this section, we introduce a notion of fairness, and prove that the rank-lowering strategies are winning under this fairness assumption.

The following lemma ensures that we cannot end in a situation where no transitions can be taken, forcing the system to delay indefinitely. It will be used with the support of fairness, and will be the key to ensuring victory on fair executions.

Lemma 3.4.12. *If $\mathcal{OT}$ is repeatedly-observable, then for any run $\rho = ((s_i, \gamma_i, s_{i+1}))_{i\in\mathbb{N}} \in \mathit{Ex}(\mathcal{A}_g)$ ending with an infinite sequence of delays, if ρ does no reach **Fail**, there is an infinite number of states in ρ where some transition e is enabled, formally,*[7]

$$\rho \notin \mathit{Ex}_{\mathit{fail}}(\mathcal{A}_g) \Rightarrow \exists e \in E_g^{\mathcal{A}},\ \overset{\infty}{\exists}\, i \in \mathbb{N},\ e \in \mathit{enab}(s_i).$$

Proof. We show this lemma by contradiction. Assume that for some $\rho \in \mathsf{Ex}(\mathcal{A}_g)$, we have

$$\rho \notin \mathsf{Ex}_{\mathsf{fail}}(\mathcal{A}_g) \wedge \forall e \in E_g^{\mathcal{A}},\ \overset{\infty}{\forall}\, i \in \mathbb{N},\ e \notin \mathsf{enab}(s_i).$$

Let $\rho_{\max}$ be the shortest prefix such that no transition is enabled after this prefix along ρ (it exists because E is finite and there is only a finite number of these prefixes per element of E). Consider any prefix ρ' of ρ strictly containing $\rho_{\max}$; there is no partial signature μ such that $\mathsf{last}(\rho') \xrightarrow{\mu}$ and $\mathsf{trace}(\mu) \notin \mathbb{R}_{\geq 0}$, as there is no time successor of $\mathsf{last}(\rho')$ with an enabled transition. This contradicts the repeated-observability of $\mathcal{OT}$ out of ***Fail*** (as $\mathcal{A}_g$ and $\mathcal{OT}$ are the same automaton). □

In order to introduce our notion of fairness, we define three notions of the infinite support of a run. The first one characterizes the set of regions encountered infinitely often, while the other two distinguish the regions from which a discrete action was taken infinitely often from the regions left by elapsing time infinitely often. This distinction will help us precisely define the behaviour expected from the implementation in each case.

Definition 3.4.13. *Let ρ be an infinite run, its* infinite regions support *$\mathit{Inf}(\rho)$ is the set*

7. In this expression, $\overset{\infty}{\exists}\, i \in \mathbb{N},\ \phi(i)$ means that $\phi(i)$ is true for infinitely many integers. In the same way, $\overset{\infty}{\forall}\, i \in \mathbb{N}\ \phi(i)$ means that $\phi(i)$ is true for all but finitely many integers.

of regions appearing infinitely often in ρ:

$$\mathit{Inf}((s_i,\gamma_i,s_{i+1})_{i\in\mathbb{N}}) = \{reg \in \mathcal{R} \mid^{\infty}\!\!\exists\ i \in \mathbb{N},\ s_i \in reg \vee (\gamma_i \in \mathbb{R}_{\geq 0} \wedge \exists s'_i \in reg,\ \exists t_i < \gamma_i,\ s_i \xrightarrow{t_i} s'_i)\}$$

Its infinite transitions support $\mathit{Inf}_E(\rho)$ *is the set of pairs* (reg, e) *of* $\mathcal{R} \times E^{\mathcal{A}_g}$ *such that the transition* e *is taken infinitely often from the region* reg*:*

$$\mathit{Inf}_E((s_i,\gamma_i,s_{i+1})_{i\in\mathbb{N}}) = \{(reg,e) \in \mathit{Inf}(\rho) \times E_g^{\mathcal{A}} \mid^{\infty}\!\!\exists\ i \in \mathbb{N}, s_i \in reg \wedge \gamma_i = e\}$$

and its infinite waiting support *is the set of regions left infinitely often by delaying along* ρ*.*

$$\mathit{Inf}_t((s_i,\gamma_i,s_{i+1})_{i\in\mathbb{N}}) = \{reg \in \mathit{Inf}(\rho) \mid^{\infty}\!\!\exists\ i \in \mathbb{N}, \gamma_i \in \mathbb{R}_{\geq 0} \wedge \exists\ t' \leq \gamma_i,\ s_i \xrightarrow{t'} s'_i \in reg \wedge s_{i+1} \in reg' \in \mathit{SucTemp}(reg)\}$$

Using these notions of infinite support, we will define the fairness of a run. The intuition behind it is as follows: if a behaviour is *implemented*, then it will be ultimately displayed if repeatedly requested. Formally, we reason on the game and not on the implementation, as we do not know its model, making our fairness assumption stronger than what is intuitively expected.

To ease the reading of the definition we define the set of transitions with uncontrollable actions $E_{uco} = \{e \mid \mathsf{act}(e) \in \Sigma_{uco}\}$ and similarly for controllable actions $E_{co} = \{e \mid \mathsf{act}(e) \in \Sigma_{co}\}$. We furthermore define the set of controllable enabled transitions for a region *reg*: $\mathsf{enab}_{co}(reg) = \mathsf{enab}(reg) \cap E_{co}$ and similarly the set of conformant uncontrollable enabled transitions (*i.e.* that do not lead to ***Fail***):

$$\mathsf{enab}_{uco,\mathsf{conf}}(reg) = \mathsf{enab}(reg) \cap E_{uco} \setminus \{e \mid \mathsf{next}(e, reg) \subseteq \boldsymbol{Fail}\}.$$

Definition 3.4.14. *An infinite run ρ in a TGA $\mathcal{A}_g = (\mathcal{L}, \ell_{init}, \Sigma_{uco} \uplus \Sigma_{co}, C, \mathcal{I}, E)$ (with timed transitions system $\mathcal{T} = (S, s_{init}, \Gamma, \rightarrow_{\mathcal{T}})$) is said to be* fair *when:*[8]

8. See definitions of enab, next, SucTemp in subsection 3.2.1.

$$\forall reg \in \mathit{Inf}(\rho), \begin{cases} \{reg\} \times \mathit{enab}_{uco,\mathit{conf}}(reg) \subseteq \mathit{Inf}_E(\rho) \wedge & (3.1) \\ \quad \begin{bmatrix} \vee & \{e \mid (reg, e) \in \mathit{Inf}_E(\rho)\} \cap E_{co} \neq \emptyset \\ \vee & reg \in \mathit{Inf}_t(\rho) \\ & \mathit{enab}_{co}(reg) = \emptyset \wedge \mathit{SucTemp}(reg) \subset \boldsymbol{\mathit{Fail}} \end{bmatrix} & (3.2) \end{cases}$$

We denote by $\mathsf{Fair}(\mathcal{A}_g)$ *the set of fair runs of* $\mathcal{A}_g$*.*

Fair runs model restrictions on the system runs corresponding to strategies of the system. The first part (3.1) of the definition ensures that for each region visited infinitely often, each tioco-conformant enabled action of the implementation will be played infinitely often from that region. Intuitively, it means that the implementation explores all of its options infinitely often. The second part (3.2) ensures that the implementation will infinitely often let the tester play in this region, by saying that either a discrete action (selected by the tester) has been played infinitely often from this region (first disjunctive case) or that it has been possible to leave this region by waiting infinitely often (second disjunctive case). This is limited by the third disjunctive case, when there is no controllable action enabled and the strict timed successors of the region are either empty (maximal delay successor region) or included in ***Fail***. This definition of fairness is related to the "strong fairness" notion used in model checking. Restricting to fair runs is sufficient to ensure a winning execution when the tester uses a rank-lowering strategy. Intuitively, combined with Lemma 3.4.12 and the repeated-observability assumption, it assures that the system will keep providing outputs until a verdict is reached, and allows to show the following property.

Proposition 3.4.15. *Rank-lowering strategies are winning on* $\mathsf{Fair}(\mathcal{A}_g)$ *(i.e., all fair outcomes are winning).*

In terms of testing, this means that if the implementation is fair, a tester following a rank-lowering strategy will reach a conclusive verdict. In order to carry out this proof, we reason on regions. We exploit the fact that a region is included in any W_i^j it intersects.

Proof. Let $\mathcal{T} = (S, s_{init}, \Gamma, \rightarrow_{\mathcal{T}})$ be the timed transition system associated with $\mathcal{A}_g = (\mathcal{L}, \ell_{init}, \Sigma_{uco} \uplus \Sigma_{co}, C, \mathcal{I}, E)$. Let f be a rank-lowering strategy. We want to prove that $\mathsf{Outcome}(f) \cap \mathsf{Fair}(\mathcal{A}_g) \subseteq \mathsf{Win}(\mathcal{A}_g)$. We proceed by contradiction.

Suppose there exists an infinite run $\rho \in \mathsf{Outcome}(f) \cap \mathsf{Fair}(\mathcal{A}_g)$ such that $\rho \notin \mathsf{Win}(\mathcal{A}_g)$. We consider the set of prefixes of ρ ending in a "decision point" *i.e.* where the strategy

proposed a new pair (delay,action). Observe that as there are infinitely many such prefixes, there exist some regions where infinitely many of them end. These regions are a subset of $\mathsf{Inf}(\rho)$. We denote by r_{min} the minimal rank obtained in these regions and consider one of these regions *reg* such that $r(reg) = r_{min}$. By definition of *reg* there are infinitely many prefixes ν of ρ ending in *reg* such that the strategy proposed a new pair. We write $f(\mathsf{last}(\nu)) = (t_\nu^f, a_\nu^f)$. As there are four possible ways to propose an action for a rank lowering strategy, at least one of them appeared infinitely often. We distinguish these four cases in our reasoning:

- $a_\nu^f \in \Sigma_{co}$, *i.e.* $\mathsf{last}(\nu) \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{co}}(W^-(\mathsf{last}(\nu))))$. In this case, there exists $e \in E_g^{\mathcal{A}}$ such that $\mathsf{act}(e) = a_\nu^f \wedge \mathsf{last}(\nu) \xrightarrow{t_\nu^f} s'_\nu \xrightarrow{e} s'' \in W^-(\mathsf{last}(\nu))$. Because of the minimality constraint on t_ν^f there exists a unique region *reg*′ such that for all ν, $s'_\nu \in reg'$. If $\mathsf{last}(\nu) \in reg'$ then $\mathsf{last}(\nu) \in \mathsf{Pred}_{\{a_\nu^f\}}(W^-(last(\nu)))$ and by the minimality constraint on t_ν^f no rank lowering strategy can delay out of *reg*′, hence $reg' \notin \mathsf{Inf}_t(\rho)$. Furthermore $e \in \mathsf{enab}_c(reg)$. It follows from fairness that there exists a controllable transition e such that $(reg, e) \in \mathsf{Inf}_E(\rho)$. As this can only be a transition labeled by a_ν^f by definition of $\mathsf{Outcome}(f)$ and $\mathcal{A}_g$ is deterministic, $\mathsf{next}(e, reg) \in \mathsf{Inf}(\rho) \cap W^-(\mathsf{last}(\nu)) = \mathsf{Inf}(\rho) \cap W^-(reg)$ and f will take a new decision once arriving in this region after each of the infinitely many transitions, contradicting the minimality of r_{min}. Else, when $\mathsf{last}(\nu) \notin reg'$ we have $reg' \in \mathsf{SucTemp}(reg) \setminus \boldsymbol{Fail}$. Hence by construction of a rank lowering strategy and definition of $\mathsf{Outcome}(f)$, there is no controllable transition taken in *reg* along ρ. Hence by fairness $reg \in \mathsf{Inf}_t(\rho)$ and the first delay-successor region of *reg* is in $\mathsf{Inf}(\rho)$. By induction on the number of regions between *reg* and *reg*′, using the case $reg = reg'$ as the base case and the previous remark as induction step, we know that $reg' \in \mathsf{Inf}(\rho)$. We can then conclude using the previous discussion;

- $a_\nu^f = \bot$ and $\mathsf{last}(\nu) \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{uco}}(W^-(\mathsf{last}(\nu))))$. In this case there exists $e \in E_g^{\mathcal{A}}$ such that $\mathsf{act}(e) \in \Sigma_{uco} \wedge \exists t' \leq t_\nu^f,\ \mathsf{last}(\nu) \xrightarrow{t'} s'_\nu \xrightarrow{e} s'' \in W^-(\mathsf{last}(\nu))$. Plus, because of the minimality constraint on t_ν^f there exists a unique region *reg*′ such that for all ν $\mathsf{last}(\nu) \xrightarrow{t_\nu^f} \in reg'$. As in the previous case, if $reg = reg'$ then by fairness $(reg, e) \in \mathsf{Inf}_E(\rho)$ as e is uncontrollable and thus $\mathsf{next}(e, reg) \in \mathsf{Inf}(\rho) \cap W^-(reg)$ and f will take a new decision once arriving in this region after each of the infinitely many transitions, contradicting the minimality of r_{min}. Else $reg' \in \mathsf{SucTemp}(reg) \setminus \boldsymbol{Fail}$ and as a rank lowering strategy will never play a discrete transition in *reg* by minimality constraint on t_ν^f we have the same induction as in the previous case and we can conclude using

the case $reg = reg'$;

- $a^f_\nu = \bot$ and $\mathsf{last}(\nu) \in \mathsf{tPred}(W^-(\mathsf{last}(\nu)))$. In this situation the simple induction on time successors regions is enough to conclude that we reach infinitely often a region in $W^-(\mathsf{last}(\nu))$ in which (by minimality of the delays proposed by a rank lowering strategy) the strategy will take a decision infinitely often. This contradicts once again the minimality of r_{min};

- otherwise, either $W^-(\mathsf{last}(\nu))$ is undefined and hence $r_{min} = (0, 0)$, contradicting the fact that ρ is not winning, or there is no partial execution from $\mathsf{last}(\nu)$ to $W^-(\mathsf{last}(\nu))$ meaning that the system can only delay. By construction of the (W^j_i) this corresponds to a timed predecessor of ***Fail***. Hence the system is cornered and can only delay to ***Fail***, thus ρ should be winning. □

Under the hypothesis of a fair implementation, we thus have identified a test-case generation method, starting from the specification with restarts and the test purpose, and constructing a test case as a winning strategy on the game created from the objective-centered tester. The complexity of this method is exponential in the size of $\mathcal{DP}$. More precisely:

Proposition 3.4.16. *Given a deterministic product $\mathcal{DP}$, the objective-centered tester $\mathcal{OT}$ can be linearly computed from $\mathcal{DP}$, the construction of a strategy relies on the construction of the (W^j_i), and is hence exponential in the size of $C^{\mathcal{DP}}$ and linear in the size of $L^{\mathcal{DP}}$.*

Observe that if $\mathcal{DP}$ is obtained from $\mathcal{P}$ by the game presented in [Ber+15], then $L^{\mathcal{DP}}$ is doubly-exponential in the size of $C^{\mathcal{S}} \uplus C^{\mathcal{TP}} \uplus C^{\mathcal{DP}}$ (in the setting of [Ber+15], $C^{\mathcal{DP}}$ is a parameter of the algorithm).

3.4.3 Properties of the test cases

Having constructed strategies for the tester, and identified a scope of implementation behaviours that allows these strategies to enforce a conclusive verdict, we now study the properties obtained by the test generation method presented above. We call *test case* a pair $(\mathcal{A}_g, f)$ where $\mathcal{A}_g$ is the game corresponding to the objective-centered tester $\mathcal{OT}$, and f is a rank-lowering strategy on $\mathcal{A}_g$. We denote by $\mathcal{TC}(\mathcal{S}, \mathcal{TP})$ the set of possible test cases generated from a specification $\mathcal{S}$ and a test purpose $\mathcal{TP}$, and $\mathcal{TC}(\mathcal{S})$ the set of test

cases for any test purpose. Recall that it is assumed that the test purposes associated with a specification are restricted to those leading to a determinizable product.

We first define *parallel runs* as the possible outcomes of a test case combined with an implementation, which thus model their parallel composition.

Definition 3.4.17. *Given a test case* $(\mathcal{A}_g, f)$ *and an implementation* $\mathcal{I}$*, their* parallel runs *are the sequences* $((s_i, s'_i), (\gamma_i, \gamma'_i), (s_{i+1}, s'_{i+1}))_i$ *such that* $(s_i, \gamma_i, s_{i+1})_i$ *is an outcome of* f *in* $\mathcal{A}_g$*,* $((s'_i, \gamma'_i, s'_{i+1}))_i$ *is a run of* $\mathcal{I}$*, and for all* i*, either* $\gamma_i = \gamma'_i$ *if* $\gamma_i \in \mathbb{R}_{\geq 0}$ *or* $\mathsf{act}(\gamma_i) = \mathsf{act}(\gamma'_i)$ *otherwise. We write* $\mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})$ *for the set of parallel runs of the test case* $(\mathcal{A}_g, f)$ *and of the implementation* $\mathcal{I}$*.*

We say that an implementation $\mathcal{I}$ fails a test case $(\mathcal{A}_g, f)$, and write $\mathcal{I}$ fails $(\mathcal{A}_g, f)$, when there exists a run in $\mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})$ that reaches ***Fail***. Our method is sound, that is, a conformant implementation cannot be detected as faulty.

Proposition 3.4.18. *The test-case generation method is* sound*: for any specification* $\mathcal{S}$*, it holds*

$$\forall \mathcal{I} \in \mathcal{I}(\mathcal{S}),\ \forall (\mathcal{A}_g, f) \in \mathcal{TC}(\mathcal{S}),\ (\mathcal{I}\ \mathit{fails}\ (\mathcal{A}_g, f) \Rightarrow \neg(\mathcal{I}\ \mathit{tioco}\ \mathcal{S})).$$

Proof. Let $\mathcal{S}$ be a specification, $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ and $(\mathcal{A}_g, f) \in \mathcal{TC}(\mathcal{S})$. Suppose that $\mathcal{I}$ fails $(\mathcal{A}_g, f)$, we will prove that $\neg(\mathcal{I}\ \mathsf{tioco}\ \mathcal{S})$.

Since $\mathcal{I}$ fails $(\mathcal{A}_g, f)$, there is a finite run ρ of $\mathsf{ParRun}(\mathcal{A}_g, f, I)$ such that $last(\rho) \in \boldsymbol{Fail} \times S^{\mathcal{I}}$ and it is the first configuration of ρ in this set. Let $\sigma = \mathsf{trace}(\rho)$. By construction of ***Fail***, either $\sigma = \sigma' \cdot t$ (if the configuration of ***Fail*** reached corresponds to a faulty invariant) or $\sigma = \sigma' \cdot a$ with $a \in \Sigma_!$ (and Fail is reached). In both cases $\mathsf{out}(\mathcal{I}\ \mathsf{after}\ \sigma') \not\subseteq \mathsf{out}(\mathcal{DP}\ \mathsf{after}\ \sigma')$, and by definition $\neg(I\ \mathsf{tioco}\ \mathcal{DP})$.

As $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{DP})$ by exact-determinizability hypothesis, $\neg(I\ \mathsf{tioco}\ \mathcal{P})$. Finally, as $\mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{S})$, we have $\neg(I\ \mathsf{tioco}\ \mathcal{S})$, which concludes the proof. □

The proofs of this property and the following one are based on the exact correspondence between ***Fail*** and the faulty signatures of $\mathcal{S}$, and use the trace equivalence of the different models ($\mathcal{DP}$, $\mathcal{P}$ and $\mathcal{S}$) to conclude. As they exploit mainly the game structure, fairness is not used, and they do not rely on the strategy being played. They are in fact true for any strategy and not only for rank-lowering ones.

We first state that if a test case exercises a non-conformant trace, then it produces the ***Fail*** verdict.

Proposition 3.4.19. *The test generation method is* strict*: given a specification* $\mathcal{S}$*,*

$$\forall \mathcal{I} \in \mathcal{I}(\mathcal{S}),\ \forall (\mathcal{A}_g, f) \in \mathcal{TC}(\mathcal{S}),\ \neg(\mathit{ParRun}(\mathcal{A}_g, f, \mathcal{I})\ \mathit{tioco}\ \mathcal{S}) \Rightarrow \mathcal{I}\ \mathit{fails}\ (\mathcal{A}_g, f)$$

Proof. Let $\mathcal{S}$ be a specification, $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ and $(\mathcal{A}_g, f) \in \mathcal{TC}(\mathcal{S})$.
Suppose that $\neg(\mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})\ \mathsf{tioco}\ \mathcal{S})$. We want to show that $\mathcal{I}$ fails $(\mathcal{A}_g, f)$. By definition of $\neg(\mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})\ \mathsf{tioco}\ \mathcal{S})$, there exist $\sigma \in \mathsf{Traces}(\mathcal{S})$ and

$$a \in \mathsf{out}(\mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})\ \mathsf{after}\ \sigma) \setminus \mathsf{out}(\mathcal{S}\ \mathsf{after}\ \sigma)$$

with $\mathsf{out}(\mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})\ \mathsf{after}\ \sigma) = \{a \in \Sigma_! \cup \mathbb{R}_{\geq 0} \mid \exists \rho \in \mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I}),\ \mathsf{trace}(\rho) = \sigma \cdot a\}$ the extension of outputs after a trace to parallel runs. Since $\mathcal{DP}$ is an exact determinization of $\mathcal{P}$ we have the following equalities: $\mathsf{Traces}(\mathcal{S}) = \mathsf{Traces}(\mathcal{P}) = \mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$. Since $a \in \mathbb{R}_{\geq 0} \cup \Sigma_!$, $\sigma \cdot a \in \mathsf{Traces}(\mathcal{OT})$ as invariants have been removed, and the automaton has been completed on $\Sigma_!$ with transitions to Fail). Hence $\sigma \cdot a \in \mathsf{Traces}(\mathsf{Ex}_{\mathsf{fail}}(\mathcal{OT}))$. Thus, for $\rho \in \mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})$ such that $\mathsf{trace}(\rho) = \sigma \cdot a$, $\mathsf{last}(\rho) \in \boldsymbol{\mathit{Fail}}$ and $\mathcal{I}$ fails $(\mathcal{A}_g, f)$. □

Observe that once again, the properties of the strategy are not used.

This method also enjoys a precision property: traces leading the test case to ***Pass*** are exactly traces conforming to the specification and accepted by the test purpose. The proof of this property uses the exact encoding of the Accept states and the definition of ***Pass***. As the previous two, it then propagates the property through the different test artifacts.

Proposition 3.4.20. *The test case generation method is* precise*: for any specification* $\mathcal{S}$ *and test purpose* $\mathcal{TP}$ *it can be stated that*

$$\forall (\mathcal{A}_g, f) \in \mathcal{TC}(\mathcal{S}, \mathcal{TP}), \forall \sigma \in \mathit{Traces}(\mathit{Outcome}(f)),$$
$$\mathcal{A}_g\ \mathit{after}\ \sigma \in \boldsymbol{\mathit{Pass}} \ \Leftrightarrow\ (\sigma \in \mathit{Traces}(\mathcal{S}) \wedge \mathcal{TP}\ \mathit{after}\ \sigma \cap \mathit{Accept}^{\mathcal{TP}} \neq \emptyset)$$

Proof. Let σ be in $\mathsf{Traces}(\mathsf{Outcome}(f))$. Then $\mathcal{A}_g$ after $\sigma \in \boldsymbol{\mathit{Pass}}$ if, and only if, the run ρ such that $\mathsf{trace}(\rho) = \sigma$ (which is unique by determinism of $\mathcal{A}_g$ outside ***Fail***) is such that $\mathsf{last}(\rho) \in \boldsymbol{\mathit{Pass}}$, *i.e.* $\rho \in \mathsf{Ex}(\mathcal{DP})$ and $\mathsf{last}(\rho) \in \mathsf{Accept}^{\mathcal{DP}}$. Hence $\mathcal{DP}$ after $\sigma \in \mathsf{Accept}^{\mathcal{DP}}$ and as the determinization is exact, $\sigma \in \mathsf{Traces}(\mathcal{P})$ and $\mathcal{P}$ after $\sigma \in \mathsf{Accept}^{\mathcal{P}}$, which gives by definition $\sigma \in \mathsf{Traces}(\mathcal{S}) \wedge \mathcal{TP}$ after $\sigma \cap \mathsf{Accept}^{\mathcal{TP}} \neq \emptyset$. □

The proof uses only properties of the game, and once more does not rely on the precise

strategy used.

Lastly, this method is exhaustive in the sense that for any non-conformance, there exists a test case that allows to detect it, under fairness assumption.

Proposition 3.4.21. *The test generation method is* exhaustive*: for any exactly determinizable specification $\mathcal{S}$ and any implementation $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ making fair runs*

$$\neg(\mathcal{I} \textit{ tioco } \mathcal{S}) \Rightarrow \exists(\mathcal{A}_g, f) \in \mathcal{TC}(\mathcal{S}), \mathcal{I} \textit{ fails } (\mathcal{A}_g, f).$$

To demonstrate this property, a test purpose is tailored to detect a given non-conformance, by targeting a related conformant trace.

Proof. Let $\mathcal{S}$ be a specification, and $\mathcal{I} \in \mathcal{I}(\mathcal{S})$ a non-conformant implementation. By definition of $\neg(\mathcal{I} \text{ tioco } \mathcal{S})$, there exists $\sigma \in \mathsf{Traces}(\mathcal{S})$ and $a \in \mathbb{R}_{\geq 0} \cup \Sigma_!$ such that $a \in \mathsf{out}(\mathcal{I} \text{ after } \sigma)$ and $a \notin \mathsf{out}(\mathcal{S} \text{ after } \sigma)$. As $\mathcal{S}$ is repeatedly-observable, there exists $t \in \mathbb{R}_{\geq 0}$ and $b \in \Sigma^{\mathcal{S}}_{\mathsf{obs}}$ such that $\sigma \cdot t \cdot b \in \mathsf{Traces}(\mathcal{S})$. Because $\mathcal{S}$ is also non-blocking, if a is a delay, we can take $b \in \Sigma^{\mathcal{S}}_!$. Indeed, otherwise there would be no trace controlled by the implementation for any finite time (say, for time a).

It is possible to build a test purpose $\mathcal{TP}$ that accepts exactly the trace $\sigma \cdot t \cdot b$. It suffices to direct every transition that is not part of this trace to a sink location. As $\sigma \cdot t \cdot b \in \mathsf{Traces}(\mathcal{S})$, it is also a trace of the product $\mathcal{P} = \mathcal{S} \times \mathcal{TP}$. As $\mathcal{S}$ is exactly determinizable and $\mathcal{TP}$ is deterministic, $\mathcal{P}$ is exactly determinizable by allowing enough resources (number of clocks and maximal constant) to $\mathcal{DP}$. We thus obtain $\mathsf{Traces}(\mathcal{DP}) = \mathsf{Traces}(\mathcal{P})$ and $\sigma \cdot t \cdot b \in \mathsf{Traces}(\mathcal{DP})$. Hence, the minimal elements of $\boldsymbol{Pass}$ are $\mathcal{OT}$ after $\sigma \cdot t \cdot b$.

From $\mathcal{OT}$ a test case $(\mathcal{A}_g, f)$ can be built, with f a rank-lowering strategy. By assumption, the implementation is playing fair runs, hence f is winning. So there exists $\rho \in \mathsf{Outcome}(f)$ such that $\mathsf{trace}(\rho) = \sigma \cdot t \cdot b$, and thus there exists $\rho' \in \mathsf{Outcome}(f)$ such that $\mathsf{trace}(\rho') = \sigma$. By assumption, $\sigma \cdot a \in \mathsf{Traces}(\mathcal{I})$, and depending on the nature of a:

- if $a \in \Sigma_!$ then $\sigma \cdot a \in \mathsf{Outcome}(s^{\mathcal{A}_g}_{init}, f)$ as $\mathcal{A}_g$ is complete on $\Sigma_!$. Hence $\sigma \cdot a \in \mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})$ and as $\sigma \cdot a \notin \mathsf{Traces}(\mathcal{S})$ and the determinization is exact, $\sigma \cdot a \notin \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ and $\mathcal{A}_g$ after $\sigma \cdot a \in \boldsymbol{Fail}$. Hence $\mathcal{I}$ fails $(\mathcal{A}_g, f)$;

- if a is a delay, then $a > t$, and $b \in \Sigma_!$. As b is controlled by the implementation, and there is no invariant in $\mathcal{A}_g$, $\sigma \cdot a \in \mathsf{Outcome}(f)$. Hence $\sigma \cdot a \in \mathsf{ParRun}(\mathcal{A}_g, f, \mathcal{I})$ and as $\sigma \cdot a \notin \mathsf{Traces}(\mathcal{S})$ and the determinization is exact, $\sigma \cdot a \notin \mathsf{Traces}_{\mathsf{conf}}(\mathcal{OT})$ and $\mathcal{A}_g$ after $\sigma \cdot a \in \boldsymbol{Fail}$. Hence $\mathcal{I}$ fails $(\mathcal{A}_g, f)$. □

The properties obtained in this last part correspond exactly to the properties already obtained in [Ber+12] in the determinizable case, our results only adapt the formalism to the game formulation, making it more precise, especially on the interaction between the test strategy and the implementation, as expected from the game formulation. The construction of rank-lowering strategies allows to keep the interesting properties of the test cases, while adding an equally important information about the control of the tests: the test strategies are winning in finite time (*i.e.* an outcome of the strategy cannot loop infinitely without reaching a winning configuration) against all fair implementations.

3.5 Implementing Rank Lowering Strategies

The construction of a practical algorithm to implement a rank lowering strategy mainly boils down to the computation of the (W_i^j) hierarchy on a symbolic representation of the game semantic. Although the construction points toward a backward implementation, we chose a forward one in order to allow to (over) approximate the ranks or to use a partial result while the exact ranks are being computed online. Furthermore, we argue that extending this algorithm to the more general setting where ***Inconclusive*** configurations exist is less complex than for a backward one.

3.5.1 Algorithm

In order to practically use the rank lowering strategies, an algorithm has to be devised to deal with the necessary symbolic representation of locations and zones. For this we develop on the algorithm proposed in [Cas+05], that presents an efficient on-the-fly approach to construct winning strategies for timed games, using a zone-based representation. Our symbolic algorithm presented in Algorithm 1 is inspired from this work, and uses at its core a rank updating function detailed in Algorithm 2. In order to present these functions, we first define $(j,i)^-$ as the maximal rank strictly lower than (j,i) in order to avoid the distinction between $(j,i-1)$ and $(j-1,\infty)$ in the algorithms. We also define, for a zone z, a location ℓ, an action $\alpha \in \Sigma$, the set of states reached from (ℓ,z) after α and some delay:

$$\mathsf{Post}_\alpha((\ell,z)) = \{s' \mid \exists s \in (\ell,z), \exists t \in \mathbb{R}_{\geq 0}\ s \xrightarrow{\alpha.t} s'\}.$$

As $\mathcal{A}_g$ is deterministic, this set of states corresponds to a zone in a unique location.

Observe that the (W_i^j) operators π and π' (defined in Def. 3.4.2) use information about the game structure, and are not restricted to the configurations explored by the algorithm.

In the following, we heavily rely on the properties of π and π', the two functions used to compute the sets (W_i^j). We quickly formalize them bellow.

Proposition 3.5.1. *For p a function in $\{\pi, \pi'\}$ and S' a set of configurations, $p(S') \geq S'$ and p is increasing (*i.e. *$S' \geq S'' \Rightarrow p(S') \geq p(S'')$).*

Proof. For the first property, $p(S') \geq S'$ is implied by the inclusion of null delays in $\mathsf{tPred}_<$. For the second property, notice that Pred_Σ and tPred are increasing, and for π that $\overline{S'}$ decreases when S' increases. □

These properties transfer to the computation of (W_i^j) and are used to ground the algorithms. Although it was not formally stated before, it was already highlighted in Figure 3.10.

To maintain coherence and ease the notations, we present the set of configurations computed by the algorithm as unions of products of a location with a zone of valuations. Of course an actual implementation of the algorithm should suppress the information about locations, that is already referenced in the symbolic states.

The computation of an RLS realized by Algorithm 1 is based on a set Visited of encountered symbolic states and a queue Waiting of transitions to be processed[9]. For each symbolic state, two informations are stored: a list Depend of incoming transitions and a dictionary W of zones associated with different ranks in the state. The algorithm is based on a method updateRank that will be presented later and correctly updates the estimates $W(s)$ according to the available information. After an initialization taking care of the timed successors of the initial configuration, denoted by $(\ell_{init}, \overrightarrow{\mathbf{0}})$ (lines 1 to 9), the main loop starts in line 10 and iterates until the reachable part of the automaton is explored, unless the initial configuration is a final one. In this loop, at each iteration a transition is taken from Waiting and processed. There are two cases to consider, presented in Figure 3.12, where the snake arrow represents the call to an edge, forward or backward. If $s' \notin$ Visited the transition was stored to explore and the first branch of the **if** executes (line 13). It records the new s' and analyses the local information as presented on the left side of Figure 3.12. If new information is acquired, its backpropagation is planed (line 19). The second situation corresponds to a backpropagation triggered by a previous call to

9. The specific order of computation of the transitions does not impact termination or correctness, but can be of great importance for the efficiency.

line 19. In this case we are mostly interested in updating the ranks in the origin of the transition, and backpropagate again if something has been updated. This corresponds to the right side of Figure 3.12. Observe that line 25 is there to account for an exploration that leads to a pre-existing state[10].

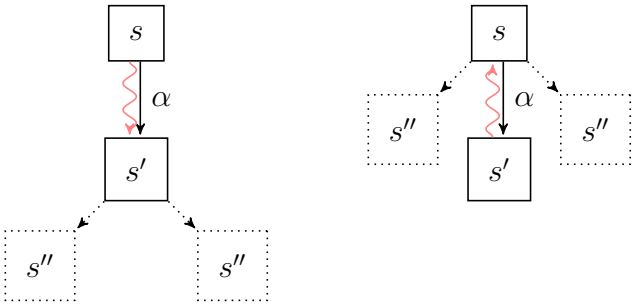


Figure 3.12: The two kind of edges popped from Waiting.

Our algorithm is based on a data structure approximating the (W_i^j). It takes the form of a dictionary $W(s)$ for each symbolic state s. In order to reduce the memory cost of $W(s)$ we chose to keep in memory only the ranks (j, i) that increase strictly the corresponding zone *e.g.* such that $W_i^j \supsetneq W_{i-1}^j$. The ranks that do not have their zone explicitly stored hence refer to the greatest zone of lesser rank. If no such zone exists, then the zone is empty.

Algorithm 2 describes the updateRank function. Its purpose is for a given state s to update $W(s)[(j, i)]$ according to $W(.)$ of its successors. The general structure of the algorithm is a while loop on the ranks. In order to compute more efficiently, and to avoid the complex discussion about the unbounded number of i for a given j (and of j in general), it relies on an identification of *active* ranks through $active(j, i)$, *i.e.* ranks in which $W(s')[(j, i)]$ strictly increases in an s' of interest (either s or a successor). We distinguish between local activity (in act_{loc},) which has to be updated directly, and activity in a successor (in act_{suc}), that is of interest for the next rank. Furthermore, we identify the maximal active rank in $r_{\max}$[11]. This is realized in the initialization (until line 8). In the following we use the notations $r.i$ and $r.j$ to denote the components of a rank. The rest of the algorithm is a loop in the ranks, that uses the jumpNext method to update the current rank according to *active*. The loop terminates when we went far enough to ensure that no more updates can be performed. Lines 10 to 13 compute the current estimation of W_i^j

10. It corresponds to the case where this transition is not triggered by a backpropagation call.
11. The maximal rank may evolve during the algorithm execution as ranks are added to or removed from active.

Algorithm 1: Computation of an RLS

Computation of an RLS
Input: an observable DTA $\mathcal{A}_g$
Output: A symbolic transition system augmented with information about ranks
// Initialization
let $s_0 = (\ell_{init}, \overrightarrow{\mathbf{0}})$ in
Visited $\leftarrow \{s_0\}$
Waiting $\leftarrow \{(s_0, \alpha, s') \mid \alpha \in \Sigma,\ s' = \mathsf{Post}_\alpha(s_0)\}$
Depend$(s_0) = \emptyset$
let $z = s_0 \cap \boldsymbol{Pass}$ in
if $z \neq \emptyset$ **then**
  $W(s_0)[(0,0)] \leftarrow z$
  updateRank(s_0)
// Main
while *Waiting* $\neq \emptyset \wedge ((\ell_{init}, \mathbf{0}) \notin W(s_0)[(0,0)])$ **do**
  $e = (s, \alpha, s') \leftarrow pop$(Waiting)
  if $s' \notin$ *Visited* **then**
    Visited := Visited $\cup \{s'\}$
    Depend$(s') \leftarrow \{(s, \alpha, s')\}$
    let $z = s' \cap \boldsymbol{Pass}$ in
    if $z \neq \emptyset$ **then**
      $W(s')[(0,0)] \leftarrow z$
      updateRank(s')
      Waiting := Waiting $\cup \{e\}$
    Waiting := Waiting $\cup \{(s', \alpha, s'') \mid \alpha \in \Sigma,\ s'' = \mathsf{Post}_\alpha(s')\}$
  else
    $improved =$ updateRank(s)
    if *improved* **then**
      Waiting := Waiting $\cup$ Depend(s)
    Depend(s') := Depend$(s') \cup \{e\}$

according to the $W(s')$. Then, if this zone is strictly greater than the current approximate, it is updated (starting line 19). Else, the data structure is cleared for this rank (lines 14 to 17). In both cases, *local* is updated.

The jumpNext method is used in updateRank to correctly select the next rank of interest. Two main discussions are handled by Algorithm 3. First, local and distant ranks have a different behaviour. Indeed, if something is stored in $W(s)[(j,i)]$, it has to be updated and thus we target (j,i). Else, we go to $(j,i+1)$ as the construction of a zone only depends on the zones of strictly lesser rank. This distinction is performed using the act_{loc} subset of *active*. Second, when the next active rank is after a control loss (*i.e.* j increases), we have to know if that control loss can increase the zone. If the control loss operator (π') has not been applied to the current zone yet, it could lead to an increase of the zone and we go to $(j+1,0)$ to test it. Else we can go directly to the next active rank (with the discussion according to its location).

Example 3.5.2. *In Figure 3.13 an example of execution of* updateRank *is presented. On the first row active*(j,i) *is represented, with the local subset being denoted by (l). On the second row the action of the while loop is represented. The algorithm first looks at rank* $(0,1)$ *which is both local and a successor of an active one. It is updated, and we consider for this example that the region has strictly increased. Hence it becomes active and the next iteration goes to* $(0,2)$*. This one is not increased and hence the next jump targets the next active rank,* $(0,4)$*. As it is a local rank, we stay at* $(0,4)$ *and try to update it. Say that it does not correspond to a greater region than the new* $(0,1)$*. In this case, it is suppressed from the dictionary and the rank is not local anymore. For this example, we consider that it is not active anymore. It leads to a call to* jumpNext *that targets the next control loss,* i.e. $(1,0)$*. The example corresponds to the case where this control loss does not augment the zone, and we jump just after the next active rank as we know that a control loss would not help (it was just tried without success) and the next rank is not local. If* $(2,2)$ *does not yield a strict zone increase, then* updateRank *terminates and the only key remaining in* $W(s)$ *is* $(0,1)$*.*

Using a forward algorithm instead of a purely backward one allows to compute approximate strategies (by requiring only the initial state to be in $W(s_{init})[(j,i)]$), and avoids (most) unreachable states while making it possible to extend the algorithm to prune a (preprocessed) set of losing states (*e.g.* inconclusive states). Observe that if the strategies are approximated, the (W_i^j) are under-approximated *i.e.* the ranks are over approximated.

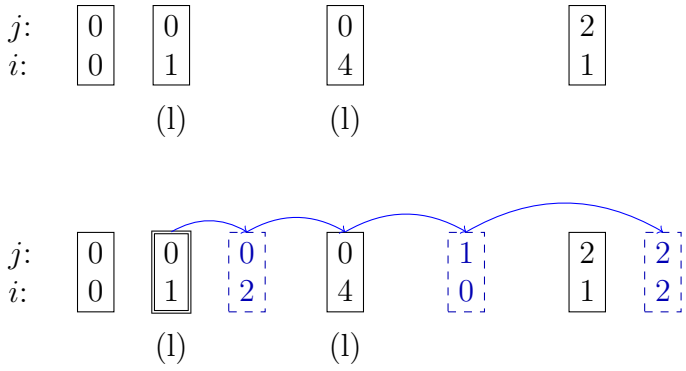
Algorithm 2: updateRank

updateRank
Input: a symbolic state s
Output: a boolean denoting whether an improvement has been performed
let $act_{suc}((j,i))$ be true iff $W(s')[(j,i)] \neq W(s')[(j,i)^-]$ for s' being a successor of s.
and $act_{loc}((j,i))$ be true iff $W(s)[(j,i)] \neq W(s)[(j,i)^-]$
we write $active((j,i)) = act_{suc}((j,i)) \vee act_{loc}((j,i))$
and $r_{\max} = max_{(j,i)} active((j,i))$
$Temp_j = -1$ `// the last j for which a set has been improved`
$Temp_W = W(s)[(0,0)]$ `// the current set`
$j = 0$; $i = 1$
while $(j,i) \leq (r_{\max}.j + 1, 0)$ **do**
  `// We try to improve the set of rank` (j,i)
  if $i = 0$ **then**
    `// then we add a control loss`
    $W \leftarrow \pi'(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)^-]) \cap s$
  else
    $W \leftarrow \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)^-]) \cap s$
  if $W \subseteq Temp_W$ **then**
    `// this set does not need to be explicitly stored`
    erase $W(s)[(j,i)]$ `// handles the data structure`
    $act_{loc}(j,i) \leftarrow false$
    jumpNext
  else
    if $W(s)[(j,i)] \subsetneq W$ **then**
      `// the set was improved`
      $W(s)[(j,i)] \leftarrow W$
      $act_{loc}((j,i)) \leftarrow true$
      $Temp_j = j$
      $Temp_W = W$
    $i := i + 1$
return $Temp_j \geq 0$

Algorithm 3: jumpNext

```
1  jumpNext
   // We jump to the next possible improvement
2  let n_r be the next active rank, including the current one.
3  if j = n_r.j then
4  |  if act_loc(n_r) then
5  |  |  i := n_r.i
6  |  else
7  |  |  i := n_r.i + 1
8  else
9  |  if Temp_j = j then
   |  |  // we might win something more after a control loss
10 |  |
11 |  |  j := j + 1
12 |  |  i := 0
13 |  else
14 |  |  j := n_r.j
15 |  |  if act_loc(n_r) then
16 |  |  |  i := n_r.i
17 |  |  else
18 |  |  |  i := n_r.i + 1
```

Figure 3.13: An execution of updateRank



Furthermore it may be necessary to refine the approximation online if a control loss leads to an unexplored part of the system.

Most importantly, the core of the computation happens inside updateRank, making it easier to adapt the algorithm to different ways to construct the symbolic graph, both forward and backward.

3.5.2 Properties

We prove some interesting properties of the algorithm by constructing them from properties of updateRank and invariants of the main algorithm while loop.

The following lemma ensures that updateRank preserves the soundness of the approximation of the W_i^j. Informally, it states that from an under-approximation of W_i^j updateRank cannot overestimate it (*i.e.* overestimate the rank).

Lemma 3.5.3. *The following property is an invariant of updateRank:*

$$\forall s \in \mathit{Visited},\ \forall\ i, j \in \mathbb{N},\ W(s)[(j,i)] \subseteq W_i^j.$$

Proof. Suppose that the property is satisfied. We show that the property is preserved by induction on (j, i). Let

$$P(j_m, i_m) = \forall j \leq j_m,\ i \leq i_m,\ \forall s \in \mathsf{Visited}, W(s)[(j,i)] \subseteq W_i^j.$$

- for $W(s)[(0,0)]$ nothing is updated so $P(0,0)$ remains true;
- fix $j, i \in \mathbb{N}$ and suppose $P(j,i)$. For a given s, either $W(s)[(j,i+1)]$ is not updated and the property trivially holds, or

  $$W(s)[(j,i+1)] = \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \cap s.$$

  By hypothesis, $\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)] \subseteq W_i^j$. Furthermore, π is an increasing function. Hence $W(s)[(j,i+1)] = \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \cap s \subseteq \pi(W_i^j) \cap s = W_{i+1}^j \cap s$. It comes that $P(j,i+1)$ is satisfied after the call to updateRank;
- fix $j \in \mathbb{N}$. We denote by i_j the minimal i such that $(\forall i' > i,\ W_i^j = W_{i'}^j)$ holds. Suppose $P(j, i_j)$. For a given s, if $W(s)[(j+1,0)]$ is not updated then the property is trivially preserved. Else, with the same arguments as in the previous case, $W(s)[(j+1,0)] =$

$\pi'(\cup_{s' \in \mathsf{Visited}} W(s')[(j, i_j)]) \cap s \subseteq W_0^{j+1}$. It comes that after updateRank $P(j+1, 0)$ is satisfied. □

The following lemma allows to say that $W(s)$ is increasing as a function of the rank, when it corresponds to a coherent approximation.

Lemma 3.5.4. *For $s \in$ Visited, $(j, i) > (j', i')$, when updateRank is not processing a rank between these two, we have that $W(s)[(j, i)] \supseteq W(s)[(j', i')]$.*

Proof. The proof can be done by a direct induction using the fact that either $W(s)[(j, i)]$ is not explicitly stored, which means that $W(s)[(j, i)] = W(s)[(j, i)^-]$ or $W(s)[(j, i)] = \pi(\cup_{s' \in \mathsf{Visited}} W(s)[(j, i)^-]) \cap s \supseteq \cup_{s' \in \mathsf{Visited}} W(s)[(j, i)^-] \cap s = W(s)[(j, i)^-]$ as $\pi(S) \supseteq S$, or the same with π'. □

We can now use the previous result to state that updateRank correctly updates the approximation.

Lemma 3.5.5. *The application of updateRank ensures the two following properties:*

- *if before the call,*

$$\forall s \in \mathit{Visited},\ \forall i, j \in \mathbb{N},\ W(s)[(j, i+1)] \subseteq \pi(\cup_{s' \in \mathit{Visited}} W(s')[(j, i)]) \cap s$$

  then, after the call,

$$\forall s \in \mathit{Visited},\ \forall i, j \in \mathbb{N},\ W(s)[(j, i+1)] = \pi(\cup_{s' \in \mathit{Visited}} W(s')[(j, i)]) \cap s;$$

- *if before the call,*

$$\forall s \in \mathit{Visited},\ \forall i, j \in \mathbb{N},\ W(s)[(j+1, 0)] \subseteq \pi'(\cup_{s' \in \mathit{Visited}} W(s')[(j+1, 0)^-]) \cap s$$

  then, after the call,

$$\forall s \in \mathit{Visited},\ \forall j \in \mathbb{N}, W(s)[(j+1, 0)] = \pi'(\cup_{s' \in \mathit{Visited}} W(s')[(j+1, 0)^-]) \cap s.$$

Proof. We prove the first property. The same proof can be used for the second one with π' instead of π. Fix $j, i \in \mathbb{N}$ and $s \in$ Visited. Suppose that before the call to updateRank, $\forall s \in \mathsf{Visited},\ \forall i, j \in \mathbb{N},\ W(s)[(j, i+1)] \subseteq \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j, i)]) \cap s$.

- if (j,i) is processed, then when the counter of the function are equal to (j,i) we discuss according to the condition $W \subseteq Temp_W$. If it is satisfied, $W(s)[(j,i)]$ is suppressed from the data structure, meaning that $W(s)[(j,i+1)] = Temp_W$. We furthermore have $W \supseteq Temp_W$ as $W \supseteq W(s)[(j,i)]$ as $\pi(S) \supseteq S$ and same for π', and $W(s)[(j,i)] \supseteq Temp_W$ by Lemma 3.5.4. It comes that $W(s)[(j,i+1)] = W$ and the property is satisfied. If the condition $W \subseteq Temp_W$ is not met, either the test $W(s)[(j,i+1)] \subsetneq W$ is satisfied, and in this case the value of $W(s)[(j,i+1)]$ is updated to the correct value, or it is not, and by hypothesis, we already have $W(s)[(j,i+1)] = W = \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \cap s$;

- if (j,i) is not processed by updateRank, then it means that

$$\pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \cap s = \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)^-]) \cap s.$$

  Indeed it means that for all s' that matter for s, there is no increase in the region between these two ranks. In this case, we know that (j,i) is not a key of $W(s)$ as the rank is not local (else it would have been updated). It comes that $W(s)[(j,i)] = W(s)[(j,i)^-]$. By induction $W(s)[(j,i)^-] = \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)^-]) \cap s$ (the induction is correctly initialized as $(0,1)$ is processed). It comes that $W(s)[(j,i)] = \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \cap s$. □

Now that the properties of the auxiliary function are stated, we discuss of the while loop invariants in the main algorithm.

Lemma 3.5.6. *The following properties are true at the beginning and end of every* `while` *loop iteration:*

- $\forall\ s \in \mathit{Visited},\ \forall\ \alpha \in \Sigma,$ *noting* $s' = \mathit{Post}_\alpha(s)$*:*
  $(s,\alpha,s') \in \mathit{Waiting} \vee (s' \in \mathit{Visited} \wedge (s,\alpha,s') \in \mathit{Depend}(s'))$*;*

- $\forall s \in \mathit{Visited},\ \forall\ i,j \in \mathbb{N},\ W(s)[(j,i)] \subseteq W_i^j$*;*

- $\forall s \in \mathit{Visited},\ \forall i,j \in \mathbb{N},$
  $W(s)[(j,i+1)] \subseteq \pi(\cup_{s' \in \mathit{Visited}} W(s')[(j,i)]) \cap s \wedge$
  $(W(s)[(j,i+1)] = \pi(\cup_{s' \in \mathit{Visited}} W(s')[(j,i)]) \cap s \vee$
  $\exists (s,\alpha,s') \in \mathit{Waiting}, s' \in \mathit{Visited})$ *;*

- $\forall s \in \mathit{Visited},\ \forall j \in \mathbb{N}$,
  $W(s)[(j+1,0)] \subseteq \pi'(\cup_{s' \in \mathit{Visited}} W(s')[(j+1,0)^-]) \cap s \wedge$
  $(W(s)[(j+1,0)] = \pi'(\cup_{s' \in \mathit{Visited}} W(s')[(j+1,0)^-]) \cap s \vee$
  $\exists (s, \alpha, s') \in \mathit{Waiting}, s' \in \mathit{Visited})$.

Proof. We prove each point independently.

- we prove the first invariant by induction. Before the loop execution, $\mathsf{Visited} = \{s_{init}\}$ and $\mathsf{Waiting} = \{(s_{init}, \alpha, s') \mid s' = \mathsf{Post}_\alpha(s_{init})\}$. Hence the property is satisfied. During the loop execution, the property is preserved. Indeed, if $e = (s, \alpha, s')$ is the considered edge, if $s' \in \mathsf{Visited}$ the only array taken out of $\mathsf{Waiting}$ is e and it is added to $\mathsf{Depend}(s')$. Furthermore no state is added to $\mathsf{Visited}$, hence the property is preserved. Else, $s' \notin \mathsf{Visited}$ and (1) e is added to $\mathsf{Depend}(s')$ which ensures the property for $\mathsf{Visited} \setminus \{s'\}$; (2) $\{(s', \alpha, s'') \mid s'' = \mathsf{Post}_\alpha(s')\}$ is added to $\mathsf{Waiting}$, ensuring the property of s'. In both cases, the property is preserved;

- we will show the property by induction:

  - before the loop start, the property is satisfied. Indeed, either $W(s_{init})[(0,0)]$ is empty, and in this case all the $W(s_{init})[(j,i)]$ are empty, or $W(s_{init})[(j,i)] = s_{init} \cap \boldsymbol{Pass} \subseteq \boldsymbol{Pass} = W_0^0$. In this case, observe that this satisfies the property, and we know by Lemma 3.5.3 that $\mathsf{updateRank}$ preserves it;

  - with the same argument as in the base case, we know that $W(s)[(0,0)] \subseteq W_0^0$. Hence, by induction hypothesis, the property is always satisfied before the call to $\mathsf{updateRank}$, and this call preserves it by Lemma 3.5.3.

- the property is satisfied when first entering the loop. Indeed, if $s_{init} \cap \boldsymbol{Pass} = \emptyset$, $W(s_{init})$ is empty and the property trivially holds. Else, notice that $W(s_{init})[(j, i+1)] = W(s_{init})[(0,0)] \subseteq \pi(W(s_{init})[(j,i)]) \cap s$ before the call to $\mathsf{updateRank}$ as only $W(s_{init})[(0,0)]$ adds some states (*i.e.*implicitly, all other indices have equal sets). By Lemma 3.5.5 the property is ensured upon entering the loop.
  During the loop execution, when s' is a new state, for any state previously in $\mathsf{Visited}$ the property is ensured by induction hypothesis. For s', if there is no successor of s' in $\mathsf{Visited}$ then the same proof as for s_{init} ensures that the property is satisfied at the end of the loop iteration. Else, such a (s', α, s'') is in $\mathsf{Waiting}$ with $s'' \in \mathsf{Visited}$, and $W(s')$ is empty, and thus $W(s')[j, i+1]$ is included in every set, and in

$\pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \cap s$. This is enough to ensure the property.
When s' is not a new state, the property is ensured for $\mathsf{Visited} \setminus \{s\}$ by induction hypothesis. For s, we know that $W(s)[(j,i+1)] \subseteq \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)])$. Hence by Lemma 3.5.5 the call to updateRank ensures the property;

- this property is proven as the previous one, using the second case of the same lemma.

□

For the following property, we restrict $\mathcal{A}$ to its reachable part. This allows to state the third equation without intersecting W_i^j with $Reach(\mathcal{A})$ or $\cup_{s \in \mathsf{Visited}} s$.

Proposition 3.5.7. *Upon termination of the algorithm, the following properties hold:*

$$\forall s \in \mathit{Visited},\ \forall i,j \in \mathbb{N},\ W(s)[(j,i)] \subseteq W_i^j(\mathcal{A}),$$

$$\mathit{Waiting} = \emptyset \Rightarrow \forall q \in \mathit{Reach}_{\mathcal{A}}((\ell_{init}, \mathbf{0})), \exists s \in \mathit{Visited}, q \in s,$$

$$\mathit{Waiting} = \emptyset \Rightarrow \left(\forall s \in \mathit{Visited}, \forall\, j,i : s \setminus W(s)[(j,i)] \subseteq s \setminus W_i^j(\mathcal{A})\right).$$

Proof. The first property is a direct consequence of Lemma 3.5.6 second point.

For the second property, we reason by induction on an execution.
Consider $q = (\ell, v) \in \mathsf{Reach}((\ell_{init}, \mathbf{0}))$. There exists a path $q_i \xrightarrow{a_i} q_{i+1}$ in $\mathcal{A}$ with $q_0 = (\ell_{init}, \mathbf{0})$ and $q_n = q$. We associate an element of Visited to each q_i by induction as follows: $q_0 \in s_{init}$ by construction. If $a_i \in \mathbb{R}$, $s_{i+1} = s_i$. As the elements of Visited are closed by delays, we have $q_{i+1} \in s_{i+1}$. Else $s_{i+1} = \mathsf{Post}_{a_i}(s_i)$. Observe that s_{i+1} is guaranteed to be in Visited by the first point of Lemma 3.5.6 as $\mathsf{Waiting} = \emptyset$. By induction, we have $s_n \in \mathsf{Visited}$ such that $q \in s_n$.
In order to prove the third property, we suppose that $\mathsf{Waiting} = \emptyset$ and prove the property by induction:

- for $(0,0)$ we have that $W(s)[(0,0)] = s \cap \boldsymbol{Pass} = s \cap W_0^0$. So $s \setminus W(s)[(0,0)] = s \setminus (s \cap W_0^0) = s \setminus W_0^0$;
- for $(j,i+1)$ suppose that the property is satisfied for (j,i). $s \setminus W(s)[(j,i+1)] = s \setminus \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)])$ by the third point of Lemma 3.5.6. Plus, the induction hypothesis implies that $W(s')[(j,i)] \supseteq W_i^j \cap s'$. It comes that $\pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \supseteq \pi(\cup_{s' \in \mathsf{Visited}} (s' \cap W_i^j))$ as π is an increasing function. Hence, $s \setminus \pi(\cup_{s' \in \mathsf{Visited}} W(s')[(j,i)]) \subseteq s \setminus \pi(\cup_{s' \in \mathsf{Visited}} (s' \cap W_i^j))$. Furthermore $s \setminus \pi(\cup_{s' \in \mathsf{Visited}} (s' \cap W_i^j)) = s \setminus \pi(W_i^j \cap$

$(\cup_{s' \in \mathsf{Visited}} s'))$. By using the second equation and restricting $\mathcal{A}$ to its reachable part, we have (since $\mathsf{Waiting} = \emptyset$): $s \setminus \pi(W_i^j \cup_{s' \in \mathsf{Visited}} s') = s \setminus \pi(W_i^j) = s \setminus W_{i+1}^j$. In the end, we have that $s \setminus W(s)[(j, i+1)] \subseteq s \setminus W_{i+1}^j$;

- for $(j+1, i)$ we follow the same ideas using the fourth point of Lemma 3.5.6 instead of the third, and the properties of π' instead of those of π. □

Together, those three properties ensure that either the algorithm terminates early, and then the W_i^j are under approximated (*i.e.*the rank is over approximated) or it computes exactly the W_i^j (on the reachable part).

3.6 Generalizing Rank-Lowering Strategies

This section discusses the use of rank-lowering strategies in the most general case where the reachable configurations of the testing game are not all co-reachable from ***Pass***. In this case, ***Inconclusive*** verdicts reappear and, given that we consider *difficult games* where it is necessary to let the system take (a possibly unbounded number of) uncontrollable transitions to reach our targets, we cannot ensure that our strategies are winning (*i.e.* eventually reach ***Pass*** or ***Fail***). Still it is possible to define *resistant* strategies that try to avoid the ***Inconclusive*** verdict as much as they can - while pursuing a conclusive one.

We define resistent strategies through the notion of *k-resistance*. Intuitively, a configuration is considered k-resistant if there is a path from it to the goal along which the system needs to play more than k well chosen uncontrollable actions to reach ***Inconclusive***. Bluntly speaking "it remains more than k hurtful events away from ***Inconclusive*** configurations".

In the following, we first present the notion of resistance in Section 3.6.1, and then describe a general notion of resistant strategies, before specializing it for resistant rank-lowering strategies 3.6.2.

Remark 3.6.1. *We choose the term of* resistance *over the one of* robustness *as this section treats of pure game theory. In this thesis, robustness is always meant with respect to reality, as opposed to "perfect" formal models. Specifically, we consider that a method or model is robust when it copes well with errors in measures and control.*

3.6.1 k-Resistance

First, we need to define the notion of *inconclusive verdict*. A configuration s is said to be inconclusive with respect to a target set $\boldsymbol{Pass}$ when $s \notin \mathsf{coReach}(\boldsymbol{Pass})$, *i.e.* a configuration is inconclusive as soon as we cannot construct a strategy leading to the accepting verdict. Thus, formally $\boldsymbol{Inconclusive} = \overline{\mathsf{coReach}(\boldsymbol{Pass})}$.

As we now authorize the presence of inconclusive configurations, the conditions on specifications and test purposes can be relaxed: we can consider specifications without restart transitions and products of specifications and test purposes that are not (exactly) determinizable.

The notion of k-resistance is defined similarly to the ranks of rank-lowering strategies, by considering that a "hurtful" uncontrollable transition (or delay) is a control loss. The main difference is that, as this is a safety matter and not an optimization problem, the number of controllable steps separating a configuration from a resistance loss is irrelevant. Based on the prior intuition of resistance, we have that the set of (at least) 0-resistant configurations is $\mathsf{coReach}(\boldsymbol{Pass})$.

Using this, we define the k-resistant sets by induction.

Definition 3.6.2. *For a TGA $\mathcal{A}_g$ with a set of target configurations $\boldsymbol{Pass}$, we define the k-resistant sets R_k for $k \in \mathbb{N}$ inductively as:*

$$R_0 = \mathit{coReach}(\boldsymbol{Pass})$$

$$R_{k+1} = \{s \mid \exists \mu, s \xrightarrow{\mu} \boldsymbol{Pass} \wedge \forall \mu_1 \mu_2 = \mu,\ s \xrightarrow{\mu_1} s' \Rightarrow s' \notin \left(\mathit{Pred}_{\Sigma_{uco}}(\overline{R_k}) \cup \mathit{tPred}_{<}(\overline{R_k}, \mathit{Pred}_{\Sigma_{co}}(R_k))\right) \setminus \boldsymbol{Pass}\}$$

$$R_\infty = lim_{k \to \infty} R_k$$

The construction of R_{k+1} relies on the following intuition: a configuration can resist to $k+1$ control losses if there exists a partial run from it to $\boldsymbol{Pass}$ such that all uncontrollable deviations from this partial run are k resistant. Uncontrollable delays (*i.e.* delays that cannot be avoided with a controllable actions) are considered as control losses with the additions of $\mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))$ in the set of configurations to avoid in the definition of R_{k+1}.

Remark 3.6.3. *It is not explicitly required that $s' \notin \overline{R_k}$ in the definition of R_{k+1} as this condition is already ensured by $s' \notin \mathit{tPred}_{<}(\overline{R_k}, \mathit{Pred}_{\Sigma_{co}}(R_k))$.*

The following lemma gives some insight on the structure of $(R_k)_{k\in\mathbb{N}}$, which is reminiscent of the (W_i^j) structure.

Lemma 3.6.4. *For any $k \in \mathbb{N}$, we have that*

$$R_{k+1} \subseteq \overline{\mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k})} \cap \overline{\mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))} \cup \boldsymbol{Pass}$$

and $R_{k+1} \subseteq R_k$.

Proof. We first prove the first property. Consider $s \in R_{k+1}$. By applying the definition, either $s \in \boldsymbol{Pass}$ and we can conclude, or there exists μ such that $s \xrightarrow{\mu} \boldsymbol{Pass} \wedge \forall \mu_1\mu_2 = \mu,\ s \xrightarrow{\mu_1} s'$ and either $s' \in \boldsymbol{Pass}$ or $s' \notin \mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k}) \cup \mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))$. By taking $\mu_1 = \epsilon$, we can conclude that $s \notin \mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k}) \cup \mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))$, *i.e.* $s \in \overline{\mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k})} \cap \overline{\mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))}$, which implies our first property.

Using this first property we obtain that $R_{k+1} \subseteq R_k$ by noticing that $\forall k \in \mathbb{N}, \boldsymbol{Pass} \subseteq R_k$ and $\overline{\mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))} \subseteq R_k$. This last inclusion comes by applying the complement to $\mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k)) \supseteq \overline{R_k}$. □

The idea of the R_k hierarchy is to propose a lower bound to the number of actions that can be taken by an implementation to reach an ***Inconclusive*** verdict, by ensuring that such an incontrollable action cannot decrease k by more than 1. As long as $k \geq 0$ we know that it is possible to reach a ***Pass*** verdict if we play a correct strategy.

Remark 3.6.5. *We construct R_k with a forward-completion approach. This is useful to avoid "local maxima" of resistance that would trap strategies or require expensive computations to counter, but it also has some flaws, the main one being that if an unsafe set of configurations is unavoidable on the partial run to **Pass**, then all previous configurations cannot be distinguished, as illustrated in Figure 3.14.*

It is possible to leverage this issue by considering "resistance up to a point" to distinguish such configurations but it would either require an enormous computational and memory effort (if every e.g. *region is considered) or (more reasonably) expert knowledge to identify bottlenecks and ask for a refinement. This is not discussed here.*

To ensure that the hierarchy of k-resistant sets can be computed, a first step is to prove that despite being in theory infinite, only a finite number of different k appear for a given game and reachability objective.

Proposition 3.6.6. *For a TGA $\mathcal{A}_g$ and a reachability objective **Pass**, the number of different elements of $(R_k)_{k\in\mathbb{N}\cup\{\infty\}}$ is finite.*

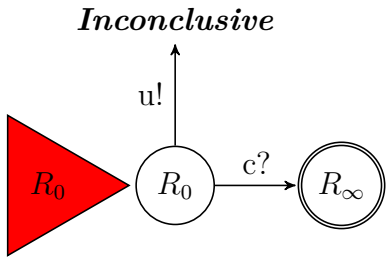


Figure 3.14: A game with undistinguishable states for the R_k hierarchy.

Proof. It is easy to see that a R_k contains every region it intersects, as it depends on possible transitions. Hence, if there is an infinite number of different $(R_k)_{k\in\mathbb{N}}$ (we can omit R_∞ without loss of generality as it is a unique element) we can find an infinite sequence of location-region pair $(\ell_i, reg_i)_{i\in\mathbb{N}}$ and an increasing mapping function $m : i \mapsto k$ associating a k to each element of the sequence such that $(\ell_i, reg_i) \subseteq R_{m(i)+1} \setminus R_{m(i)}$. By Lemma 3.6.4 we know that $(R_k)_{k\in\mathbb{N}}$ is a decreasing sequence. Thus for each infinitely many $k > 0$ such that $R_k \neq R_{k-1}$, $m^{-1}(k)$ points to *new pairs* of locations and regions.

As there are only finitely many such pairs, this is impossible. □

The previous definition of resistance is intuitive (mathematically speaking), but does not hint toward an algorithm to compute the sets. The following characterization will point toward a computation.

Proposition 3.6.7. *Consider a TGA $\mathcal{A}_g$. An equivalent definition of R_k is:*

$$R_{-1} = \mathcal{L}^{\mathcal{A}_g} \times \mathbb{R}_{\geq 0}^{|C^{\mathcal{A}_g}|}$$

$$\begin{aligned} R_{k+1} = lfp\Big[S' \mapsto & \boldsymbol{Pass} \\ & \cup \mathsf{tPred}_<\Big(S', \mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k}) \cup \mathsf{tPred}_<(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))\Big) \\ & \cup \mathsf{tPred}_\leq\Big(\mathsf{Pred}_\Sigma(S'), \mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k}) \cup \mathsf{tPred}_<(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))\Big)\Big] \end{aligned}$$

$$R_\infty = lfp[k \mapsto R_k]$$

where lfp is the least fix point operator.

Remark 3.6.8. *As for the (W_i^j), it is necessary to distinguish $\mathsf{Pred}_\Sigma(S')$, on which we want to implement some constraints (and thus use $\mathsf{tPred}_\leq$) and S' that should not be constrained*

(and hence handled with $\mathsf{tPred}_<$). For example, constraining S' could be problematic when starting from ***Pass*** *if there exists uncontrollable transitions from* ***Pass*** *to undesirable configurations.*

Proof. The proof is made by induction. First, notice that since R_{-1} is the set of all configurations, $\overline{R_{-1}} = \emptyset$. Thus the characterization of R_0 is exactly the definition of co-reachability of ***Pass*** as a fix-point and it matches the definition.

Now consider for $k \in \mathbb{N}$ that the definition and characterization of R_k match. We make the proof for $k+1$ by double inclusion. Let us note $\mathcal{D}_i$ the definition of R_i and $\mathcal{F}_i$ the fixpoint characterization. We will furthermore note $\mathcal{V}_i = \mathsf{Pred}_{\Sigma_{uco}}(\overline{R_i}) \cup \mathsf{tPred}_<(\overline{R_i}, \mathsf{Pred}_{\Sigma_{co}}(R_i))$ when it is known that $\mathcal{D}_i = \mathcal{F}_i$.

$\mathcal{D}_{k+1} \subseteq \mathcal{F}_{k+1}$ Consider $s \in \mathcal{D}_{k+1}$. By definition there exists $s \xrightarrow{\mu} s'$ with $s' \in$ ***Pass***. If $s \in$ ***Pass*** $\subseteq \mathcal{F}_{k+1}$ we have our result. Else, consider a partial run $\rho = s_0\gamma_1 ... \gamma_n s_n$ corresponding to μ. We show by induction that all s_i are in $\mathcal{F}_{k+1}$. Without loss of generality, we consider that there exists $0 \leq j < n$ such that for all $0 \leq i \leq j$, $s_i \notin$ ***Pass*** and conversely for $j < i \leq n$, $s_i \in$ ***Pass***. First, for all such $i > j$, $s_i \in$ ***Pass*** $\subseteq \mathcal{F}_{k+1}$. Next, for $i \leq j$, considering s_{i+1} is in $\mathcal{F}_{k+1}$, we discuss on $\gamma_{i+1} \in E \cup \mathbb{R}_{\geq 0}$.

- If $\gamma_{i+1} \in E$ then $s_i \in \mathsf{Pred}_\Sigma(s_{i+1}) \backslash \mathcal{V}_k$ by $\mathcal{D}_{k+1}$. Thus $s_i \in \mathsf{tPred}_\leq\Big(\mathsf{Pred}_\Sigma(s_{i+1}), \mathcal{V}_k\Big)$. It comes that $s_i \in \mathcal{F}_{k+1}$.
- If $\gamma_{i+1} \in \mathbb{R}_{\geq 0}$ then $s_i \in \mathsf{tPred}_<(s_{i+1} \cup$ ***Pass***$, \mathcal{V}_k)$. Indeed we get the restriction by applying the constraint of $\mathcal{D}_{k+1}$ for $\mu_1 = \gamma_1 \dots \gamma_i \gamma'_{i+1}$ with $0 \leq \gamma'_{i+1} < \gamma_{i+1}$. For each such signature, either it ends in ***Pass*** or in $\mathcal{V}_k$, hence the result. It comes that $s_i \in \mathcal{F}_{k+1}$.

By induction, it comes that $s = s_0 \in \mathcal{F}$.

$\mathcal{F}_{k+1} \subseteq \mathcal{D}_{k+1}$ It suffices to prove that $\mathcal{D}$ is a fixpoint of

$$S' \mapsto \textbf{\textit{Pass}} \cup \mathsf{tPred}(S' \cup \mathsf{Pred}_\Sigma(S'), \overline{R_k} \cup \mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k}) \cup \mathsf{tPred}_<(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k)))$$

that we will note f here. First note that for any sets S' and V, $\mathsf{tPred}_<(S', V) \supseteq S'$, such that $f(\mathcal{D}_{k+1}) \supseteq \mathcal{D}_{k+1}$. It remains to prove that $f(\mathcal{D}_{k+1}) \subseteq \mathcal{D}_{k+1}$. We discuss according to the cases of f:

- $\boldsymbol{Pass} \subseteq \mathcal{D}_{k+1}$;
- for $s \in \mathsf{tPred}_<(\mathcal{D}_{k+1}, \mathcal{V}_k)$ we know that there exists $t \in \mathbb{R}_{\geq 0}$ such that $s \xrightarrow{t} s' \in \mathcal{D}_{k+1}$. Furthermore, by definition of $\mathcal{D}_{k+1}$, $s' \xrightarrow{\mu} \boldsymbol{Pass}$. By aggregating the constraints on $\mathcal{V}_k$ on t and μ it comes that $s \xrightarrow{t\mu} \boldsymbol{Pass}$ proves that $s' \in \mathcal{D}_{k+1}$;
- for $s \in \mathsf{tPred}_\leq(\mathsf{Pred}_\Sigma(\mathcal{D}_{k+1}), \mathcal{V}_k)$, we can similarly construct $s \xrightarrow{e\mu} \boldsymbol{Pass}$ to prove that $s \in \mathcal{D}_{k+1}$.

Now to conclude on R_∞ it is enough to remark that by Prop. 3.6.6, R_∞ is a R_k for $k \in \mathbb{N}$. □

Remark 3.6.9. *The function iterated is close to the π function defining W_i^j in its construction, but its semantic differs greatly: allowing predecessors by uncontrollable actions and even more importantly avoiding a fix set makes a greatly wider fix point.*

This characterization could equivalently start from $R_0 = \mathsf{coReach}(\boldsymbol{Pass})$. This definition however highlights the inductive nature of R_0 and unites the computation for $k = 0$ and $k > 0$. We can use it to insist on the forward-completion of the k-resistant sets.

Corollary 3.6.10. *For any $k \in \mathbb{N}$ and $s \in R_k$, there exists a partial run ρ such that $s \xrightarrow{\rho} \boldsymbol{Pass}$ and for any s' reached from s after a prefix of ρ, $s' \in R_k$.*

Proof. The characterization function f constructs such a partial run iteratively. □

Example 3.6.11. *Consider the timed game automaton over $C = \{c_1, c_2\}$ depicted in Figure 3.15 where **Pass** is ℓ_5 (true guards and empty resets are omitted for the sake of readability). The inconclusive verdicts and different safety ranks are displayed in the table below in the form of location/zone pairs.*

Inconclusive	$\ell_3 \times \mathit{true}$, $\ell_2 \times c_1 \geq 3$
$R_0 \setminus R_1$	$\ell_2 \times c_1 < 3$, $\ell_8 \times (c_1 \leq 1 \wedge c_2 < c_1)$
$R_1 \setminus R_2$	$\ell_1 \times c_1 < 3$, $\ell_8 \times c_1 > 1$, $\ell_8 \times c_1 \leq c_2$
$R_2 \setminus R_\infty$	$\ell_6 \times \mathit{true}$, $\ell_7 \times \mathit{true}$
R_∞	$\ell_4 \times \mathit{true}$, $\ell_5 \times \mathit{true}$, $\ell_1 \times c_1 \geq 3$

Notice that if most of control losses defining ranks correspond to uncontrollable transitions, we have an example in ℓ_2 or ℓ_8 of a control loss due to an uncontrollable delay: in ℓ_2 for $c_1 < 3$, the condition on $\mathsf{tPred}_<(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))$ is the only thing stopping the zone to be in R_1. The same goes for the zone $(c_1 < 1 \wedge c_2 \leq c_1)$ in ℓ_8.

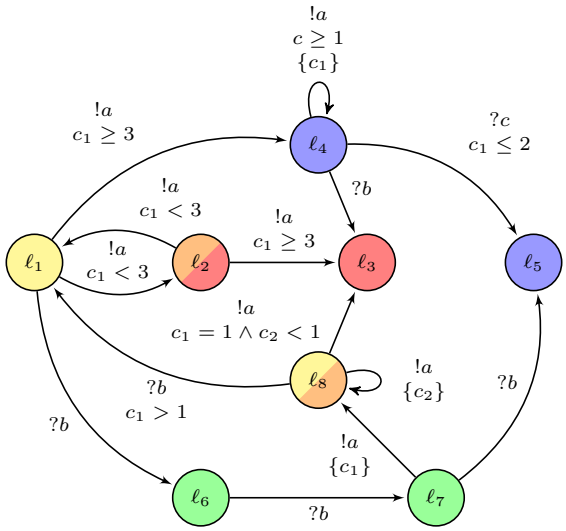


Figure 3.15: A TGA with two clocks, considering $\boldsymbol{Pass} = \ell_5$.

Finally, we point out that choosing to enhance the resistance (*i.e.* go from a k to a $k+1$ resistant configuration) cannot be done without taking risks.

Lemma 3.6.12. *Let $k \in \mathbb{N}$ and $s \in R_k \setminus R_{k+1}$. Then for any partial run ρ, $s \xrightarrow{\rho} s' \in R_{k+1} \Rightarrow \exists \rho' \in \mathit{fPref}(\rho), s \xrightarrow{\rho'} s'' \in \overline{R_k} \cup \mathit{Pred}_{\Sigma_{uco}}(\mathit{tPred}_{<}(\overline{R_k}, \mathit{Pred}_{\Sigma_{co}}(R_k))$ with $\mathit{fPref}(\rho)$ the set of prefixes of ρ.*

Proof. We conduct this proof by contradiction. Consider ρ such that $s \xrightarrow{\rho} s' \in R_{k+1}$ and for all prefixes ρ', $s \xrightarrow{\rho'} s'' \notin \overline{R_k} \cup \mathsf{Pred}_{\Sigma_{uco}}(\overline{R_k}) \cup \mathsf{tPred}_{<}(\overline{R_k}, \mathsf{Pred}_{\Sigma_{co}}(R_k))$. Then, by induction, iterating the function f defined in Prop. 3.6.7 will prove that $s \in R_{k+1}$. □

This impossibility to enhance resistance without taking risks entails that *it should not be a safety decision* to enhance it. Instead, the optimization part of the strategy should weight it with its own criterions *e.g.* is it worth to take that risk, or should another one be preferred?

3.6.2 Combining resistance and optimization

In this section, we explain how to combine safety with respect to the ***Inconclusive*** verdict -which is expressed by the mean of so-called *safety first strategies*- and optimization with respect to a reachability objective.

Safety first strategies The k-resistant sets do not retain enough information to decide what a player *should* do to achieve the reachability objective, but they advise about some actions that *should not* be proposed to avoid as much as possible the ***Inconclusive*** verdict. Using them, we define a set of *safety first strategy* that restrain their choices to actions that are considered "safe" by the resistance hierarchy. For that, we extend the usual notion of strategy with an *aim* token. This will extend strategies in the case where it relies on its adversary to take a move, by specifying a set of possible actions of the adversary that are expected. Although the player cannot control these actions, the aim is used to prove that the strategy aims for a safe configuration.

Definition 3.6.13. *A strategy aim for a strategy f is a partial function a_f from runs to sets of transitions that is coherent with the strategy,* i.e. *when $f(\rho) = (t, a)$ if $a \in \Sigma_{co}$ then $a_f(\rho) = \{e \in E \mid \mathsf{act}(e) = a \wedge \rho \xrightarrow{t \cdot e}\}$ and when $a = \bot$, $a_f(\rho) \subseteq \{e \in E \mid \mathsf{act}(e) \in \Sigma_{uco} \wedge \exists t' \leq t,\ \rho \xrightarrow{t' \cdot e}\}$. We take the convention that an empty aim corresponds to a delay only.*

Aims can be used to complement the strategy definition, to add additional constraints on its behaviour. We use it to specify *safety-first strategies.*

Definition 3.6.14. *A strategy f in a deterministic TGA $\mathcal{A}_g$ with a reachability objective **Pass** is considered to be* safety first *when for all finite ρ such that $\mathsf{last}(\rho) \notin$ **Pass**, with $f(\rho) = (t, a)$,*

- *if $a \in \Sigma_{co}$: $last(\rho) \in R_k \Rightarrow \mathsf{last}(\rho) \xrightarrow{t \cdot e} s \in R_k$ with e the unique transition enabled such that $\mathsf{act}(e) = a$;*
- *if $a = \bot$ and $a_f(\rho) \neq \emptyset$: $last(\rho) \in R_k \Rightarrow \rho \xrightarrow{t' \cdot e} s \in R_k$ for all $e \in a_f(\rho)$ and all $t' \leq t$ such that e is enabled after t';*
- *if $a = \bot$ and $a_f(\rho) = \emptyset$: $last(\rho) \in R_k \Rightarrow \rho \xrightarrow{t} s \in R_k$.*

The set of safety first strategies of a game $\mathcal{A}_g$ is noted $SF_{\mathcal{A}_g}$.

Remark 3.6.15. *This definition of safety-first strategies is tailored for reachability objectives but could easily be adapted for other objectives. We only enforce that the strategy is only safety-first* until the game is won *to relieve technical discussion about actions after the objective is reached.*

These strategies are called *safety first* because if there exists a safe partial run (as safe as the current configuration at least) they will try to follow it *regardless of the efficiency cost*. As emphasized by Lemma 3.6.12 this is the strongest requirement one can make on strategies.

Safety first rank lowering strategies A simple combination of (R_k) and (W_i^j) proposing to simply play "safety first" and optimize in the set of safest moves fails to generate effective strategies. Indeed, although they both recommend somewhat related runs, these ones can differ, and taking a local decision can be difficult. Figure 3.16 (discussed in Example 3.6.16) proposes a timed automaton that traps a memoryless strategy in a loop. Notice that this kind of difficulty does not depend on the timed nature of the automaton, but arises from the conflict between optimization and safety. To highlight it, no clock constraint is put in this example.

Example 3.6.16. *In the automaton displayed in Figure 3.16 a memoryless strategy trying to reach A using only local information about R_k and W_i^j would either be unsafe (and ignore the safest path 1-2-3) or get stuck between locations ℓ_1 and ℓ_4. Indeed, after ℓ_1, all directly reachable locations are in R_∞ and the best W_i^j is $(1,1)$ in ℓ_4. But once in ℓ_4, prioritizing safety means avoiding the risky location ℓ_5 and going back to ℓ_1.*

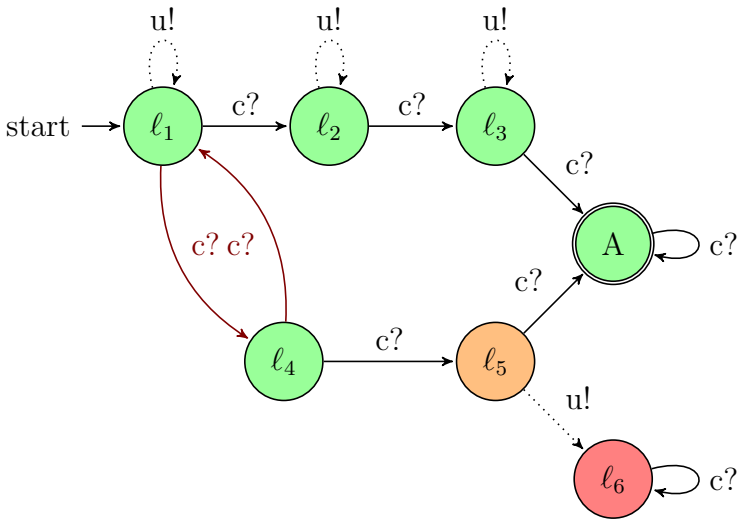
This issue arises from a form of dependency tracking problem: ℓ_4 is safe only because ℓ_1 is, so there is no point in going in ℓ_4 from ℓ_1: this does not progress along a safe path.

In order to prevent such blocking situations, we first compute the (R_k), and use them to define $({}^kW_i^j)$ and a safety first equivalent of rank lowering strategies.

Definition 3.6.17. *Given the sequence $(R_k)_{k\in\mathbb{N}\cup\{\infty\}}$, we define the sequence $({}^kW_i^j)_{j,i\in\mathbb{N}}$ as:*

- ${}^kW_0^0 = \textbf{\textit{Pass}}$ *for all k;*
- ${}^kW_{i+1}^j = \pi_k(\bigcup_{k'\geq k} {}^{k'}W_i^j)$ *where* $\pi_k(S') = \pi(S') \cap \overline{R_{k+1}}$;
- ${}^kW_0^{j+1} = \pi'_k(\bigcup_{k'\geq k} {}^{k'}W_\infty^j)$ *where* $\pi'_k(S') = \pi'(S') \cap \overline{R_{k+1}}$;

Here the intersection of the operators defining W_i^j (π and π') with $\overline{R_{k+1}}$ comes from the backward nature of the computation: by ensuring that k does not increase when computing the predecessors, we ensure that it does not decrease when playing forward. Using this $({}^kW_i^j)$, we can now find back some of the useful results we have on (W_i^j).



L	A	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6
W_i^j	$(0,0)$	$(3,0)$	$(2,0)$	$(1,0)$	$(1,1)$	$(1,0)$	-
R_k	∞	∞	∞	∞	∞	0	-1

Figure 3.16: A TA trapping a memoryless strategy

Proposition 3.6.18. *There exists $i, j \in \mathbb{N}$ such that*

$$\mathit{Reach}(\mathcal{A}_g) \setminus (\boldsymbol{\mathit{Inconclusive}} \cup \boldsymbol{\mathit{Fail}}) \subseteq \bigcup_{k \in \mathbb{N}} {}^kW_i^j$$

Proof. Let $s \in \mathsf{Reach}(\mathcal{A}_g) \setminus (\boldsymbol{\mathit{Inconclusive}} \cup \boldsymbol{\mathit{Fail}})$ be a reachable configuration. We have that $s \in \mathsf{coReach}(\boldsymbol{\mathit{Pass}})$ as $s \notin \boldsymbol{\mathit{Inconclusive}}$, and hence there exists a maximal k such that $s \in R_k$ by finiteness of the number of such k (Prop. 3.6.6). Thus by Corollary 3.6.10 there exists a partial run ρ from s to $\boldsymbol{\mathit{Pass}}$ that stays in R_k. By noting n the length of this partial run, we prove that $s \in {}^kW_0^n$ by induction on ρ. When $n = 0$, $s \in \mathsf{Accept} = {}^\infty W_0^0$ and the property holds. If we assume that the result holds for all partial runs of length n and consider $s \in R_k$ such that its corresponding partial run $\rho = (s, \gamma, s').\rho'$ has length $n+1$. Then by induction hypothesis, as s' is in R_k, $s' \in {}^kW_0^n$ and $s \in \pi'_k({}^kW_0^n)$, *i.e.* $s \in {}^kW_0^{n+1}$. □

This result ensures that we have a rank information available everywhere it can be used. We can thus rank the configurations using this information.

Definition 3.6.19. *Let* $s \in \mathsf{Reach}(\mathcal{A}_g) \setminus (\boldsymbol{Inconclusive} \cup \boldsymbol{Fail})$. *The* rank *of* s *is:*

$$r(s) = (k_s = \arg\max_{k \in \mathbb{N} \cup \{\infty\}} s \in R_k,\ j_s = \arg\min_{j \in \mathbb{N}} s \in {}^{k_s}W^{j}_{\infty},\ i_s = \arg\min_{i \in \mathbb{N}} s \in {}^{k_s}W^{j_s}_{i})$$

We can equip the ranks with an order $\preccurlyeq$ induced by the lexical order on $(\geq, \mathbb{N}) \times (\leq, \mathbb{N})^2$, *i.e.* $(k, j, i) \preccurlyeq (k', j', i')$ if and only if $k > k'$ or $k = k'$ and either $j < j'$ or $j = j'$ and $i \leq i'$. We note $s \sqsubseteq s'$ when $r(s) \preccurlyeq r(s')$.

Proposition 3.6.20. $\sqsubseteq$ *is a total preorder on* $\mathsf{Reach}(\mathcal{A}_g) \setminus (\boldsymbol{Inconclusive} \cup \boldsymbol{Fail})$.

Proof. $\sqsubseteq$ inherit transitivity and reflexivity from $\preccurlyeq$. It is not antisymmetric as different configurations can have the same rank[12]. It is total thanks to Prop.3.6.18 that assures us that $\mathsf{Reach}(\mathcal{A}_g) \setminus (\boldsymbol{Inconclusive} \cup \boldsymbol{Fail})$ is covered by the $({}^{k}W^{j}_{i})$. □

We define $\prec$ as the strict relation corresponding to $\preccurlyeq$, and we write $r^-(s)$ for the greatest rank such that $r^-(s) \prec r(s)$, and $W^-(s)$ for the associated ${}^{k}W^{j}_{i}$. Then using this new notion of $W^-(s)$, a safety first rank lowering strategy can be defined exactly in the same way that it was when using only W^{j}_{i}.

Definition 3.6.21. *We call* safety-first rank-lowering strategy *a rank lowering strategy defined using* $W^-(s)$ *on the ranks induced by* $({}^{k}W^{j}_{i})$. *We define the aim of these rank lowering strategies in the following way (according to the cases of Definition 3.4.9), with* $s = \mathsf{last}(\rho)$*:*

- *if* $s \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{co}}(W^-(s)))$ *then* $f(\rho) = (t, a)$ *with* $a \in \Sigma_{co}$ *and* $a_f(\rho) = \{e\}$ *the unique* e *enabled after* $\rho \cdot t$ *such that* $\mathsf{act}(e) = a$*;*
- *if* $s \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{uco}}(W^-(s)))$, *then* $f(\rho) = (t, \bot)$ *with* t *such that* $s \xrightarrow{t} s' \notin \mathsf{Pred}_{\Sigma_{uco}}(W^-(s))$, $\exists t' < t$, $s \xrightarrow{t'} s'' \in \mathsf{Pred}_{\Sigma_{uco}}(W^-(s))$ *and* $s'' \xrightarrow{e} s''' \in W^-(s)$ *with* $\mathsf{act}(e) \in \Sigma_{uco}$. *We take* $a_f(\rho)$ *to be the set of such* e.
- *in the two last cases* $a_f(\rho) = \emptyset$.

The name of *safety-first* rank lowering strategy is justified by the following proposition.

Proposition 3.6.22. *Let* f *be a safety first rank lowering strategy computed in a deterministic TGA* $\mathcal{A}_g$. *Then* $f \in SF_{\mathcal{A}_g}$.

12. Even configurations coming from different regions or zones can have the same rank.

Proof. We discuss according to the case in the definition of rank-lowering strategy as given in Definition 3.4.9. Noting $s = \mathsf{last}(\rho)$ and k we discuss according to $f(\rho)$.

- In the first case, $s \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{co}}(W^-(s)))$ and then $f(\rho) = (t, a)$ such that there exist $e \in E$ with $\mathsf{act}(e) = a$ satisfying $s \xrightarrow{t \cdot e} s'' \in W^-(s)$. In this case, by definition of W^- $s'' \in R_{k_s}$ with k_s the maximal k such that $s \in R_k$. We thus have that for all k such that $\mathsf{last}(\rho)$ in R_k, $\mathsf{last}(\rho) \xrightarrow{t \cdot a} s'' \in R_k$.

- In the second case, $s \in \mathsf{tPred}(\mathsf{Pred}_{\Sigma_{uco}}(W^-(s)))$ and by the definition $\exists t' < t$ $s \xrightarrow{t'} s'' \xrightarrow{a_f(\rho)} s''' \in W^-(s)$. By definition of W^- we again have our result.

- In the third case $a_f(\rho) = \emptyset$ and by Definition 3.4.9 we know that $f(\rho) = (t, \perp)$ with $\mathsf{last}(\rho) \xrightarrow{t} s' \in W^-(s)$ and by definition of W^- we again have our result.

- In the last case $\mathsf{last}(\rho) \in \boldsymbol{Pass}$ and there is no condition to satisfy.

□

Note that safety-first rank-lowering strategies correspond to rank-lowering strategies when $\boldsymbol{Inconclusive} = \emptyset$. In this setting they are thus winning under the aforementioned fairness assumption. In general however they are not winning, unless the initial configuration happens to be in R_∞.

3.7 Conclusion

This chapter proposes a game approach to the controllability problem for conformance testing from timed automata (TA) specifications. It defines a test synthesis method that produces test cases whose aim is to maximize their control upon the implementation under test, while detecting non-conformance.

Test cases are defined as strategies of a game between the tester and the implementation, based on the distance to the satisfaction of a test purpose, both in terms of number of transitions and potential control losses. Fairness assumptions are used to make those strategies winning and are proved sufficient to obtain the exhaustiveness of the test synthesis method, together with soundness, strictness and precision.

A symbolic algorithm is proposed to effectively compute these strategies, paving the way to an implementation.

Finally, an extension is proposed for the strategies in the case where the ***Inconclusive*** verdict cannot be avoided. Although in this setting the strategies are not winning, they offer a basis to approach the control problem for test cases and related situations.

This chapter opens numerous directions for future work. First, we intend to tackle partial observation in a more complete and practical way. One direction consists in finding weaker conditions under which approximate determinization [Ber+15] preserves strong connectivity, a condition for the existence of winning strategies. One could also consider a mixture of our model and the model of [Dav+10] whose observer predicates are clearly adequate in some contexts. Quantitative aspects could also better meet practical needs. The distance to the goal could also include the time distance or costs of transitions, in particular to avoid restarts when they induce heavy costs but longer and cheaper runs are possible.

The fairness assumption could also be refined. For now it is assumed on both the specification and the implementation. If the implementation does not implement some outputs, a tester could detect it with a bounded fairness assumption [Ram98], adapted to the timed context (after sufficiently many experiments traversing some region all outputs have been observed), thus allowing a stronger conformance relation with equality of output sets. A natural extension could also be to complete the approach in a stochastic view, for example by constructing a probabilistic approximation of the implementation behaviour during the test execution, allowing to use this information to improve efficiency or to gradually adapt our fairness expectation and thus the test strategy. Fundamentally, refining the fairness assumption requires to *learn* the implementation behaviours and adapt the assumption and strategies to them, raising the question of the interleaving of (formal) testing and (formal) learning.

Notably, stochastic information is required in the general setting of Section 3.6 to prove some form of optimality of the strategies in term of resistance to the ***Inconclusive*** verdict.

Chapter 4

Timed markings

> Go back? No good at all! Go sideways? Impossible! Go forward? Only thing to do! On we go!
>
> — J.R.R. Tolkien "The Hobbit"

4.1 Introduction

As discussed in Section 2.2, the state estimation problem (*i.e.* knowing in which state a system is after a given trace) is a major component of formal verification as well as an important limitation for offline verification methods, that circumvent it using approximations or limitations to sub-classes of models.

For timed automata, this problem is especially challenging as both silent transitions and non-determinism *strictly augment* the expressivity of the models. In order to tackle it, it is thus interesting to consider intuitions coming from a larger determinizable class of systems, namely automata over timed domains [BJM17] from which we import the notion of system state as sets of configurations as well as the transitions between such sets.

In this chapter, we develop a technique (first presented in [BJM18]) for efficiently computing, at runtime, the set of all possible configurations in which a partially-observable one-clock timed automaton can be and discuss its extension to general timed automata with multiple clocks - although that extension is not fully developed.

The main ingredient of our approach is the notion of *linear timed sets*: intuitively, a timed set is a set of valuations that evolves over time (formally, for a one-clock timed automaton, it is a mapping $f : t \in \mathbb{R}_{\geq 0} \mapsto f(t) \subseteq \mathbb{R}_{\geq 0}$). Linear timed sets form a restricted class of timed sets, where a valuation v in the set is transformed in $v + t$ by a time elapse of t and then filtered with respect to some minimal constant r.

A linear timed set can be defined as $f : t \in \mathbb{R}_{\geq 0} \mapsto \bigcup_i (X_i + t) \cap [r_i; +\infty)$, with $X_i \subseteq \mathbb{R}$

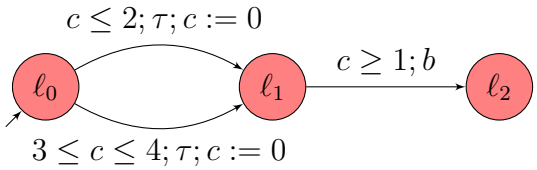


Figure 4.1: A one-clock timed automaton where only the b-transition between ℓ_1 and ℓ_2 is observable.

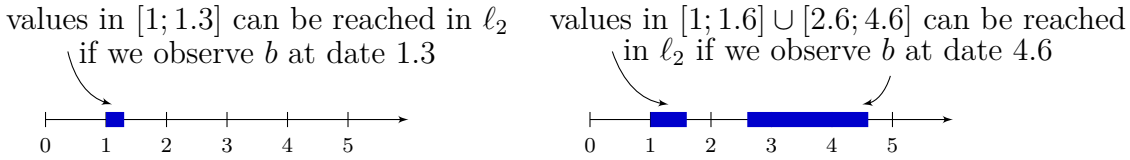


Figure 4.2: Two sets (in location ℓ_2) representing the reachable configurations in the automaton of Figure 4.1 after observing transition b at dates 1.3 (left) and 4.6 (right).

and $r_i \in \mathbb{Q}_{\geq 0}$ for all i (see Figure 4.3 on page 121 for an example of a linear timed set). Linear timed sets are well-suited to represent sets of clock valuations for one-clock timed automata, and compute their evolution along time, which is needed to update state estimations, as we illustrate on the following example.

Example 4.1.1. *Consider the one-clock timed automaton of Figure 4.1: in this automaton, the transition from ℓ_1 to ℓ_2 is observable (labelled with b), while the transitions from ℓ_0 to ℓ_1 are unobservable (labelled with the silent action τ).*

Assume that this automaton starts from the initial configuration $(\ell_0, c = 0)$. As long as no b-action is observed, we have no way of knowing whether the automaton is in ℓ_0 or ℓ_1. Now, assume that a b-action takes place at time 1.3: then we know that one of the transitions from ℓ_0 to ℓ_1 has occurred; moreover, it cannot be the transition guarded with $3 \leq c \leq 4$, since only 1.3 time units have elapsed in total. One easily checks that, if we observe a b-transition at time 1.3, then the automaton is in state ℓ_2 with $c \in [1; 1.3]$. Figure 4.2 (left) represents this set of valuations.

Similarly, if, starting from the initial configuration, we observe a b-transition at time 4.6, then both transitions from ℓ_0 to ℓ_1 may have taken place; in this situation, it can be checked that if the top transition has been taken, then the automaton can be in ℓ_2 with $c \in [2.6; 4.6]$, while if the bottom transition has been taken, the automaton can be in ℓ_2 with $c \in [1; 1.6]$. This set is represented on Figure 4.2 (right).

For such an example, our algorithm would first compute the linear timed set

$$t \in \mathbb{R}_{\geq 0} \mapsto (([-4;-3] \cup [-2;0]) + t) \cap [0;+\infty),$$

corresponding to all clock valuations that can be obtained in ℓ_1 after a delay t, as long as no transition has been observed. Since the b-transition can only take place if $c \geq 1$ (and does not reset clock c), the linear timed set reached when observing b is the intersection of this linear timed set with $[1;+\infty)$, namely:

$$t \in \mathbb{R}_{\geq 0} \mapsto (([-4;-3] \cup [-2;0]) + t) \cap [1;+\infty).$$

This linear timed set is represented at Figure 4.3. It provides a way of representing, as a single object, all possible configurations that can be reached in ℓ_2 right after observing transition b at time t, for any $t \in \mathbb{R}_{\geq 0}$.

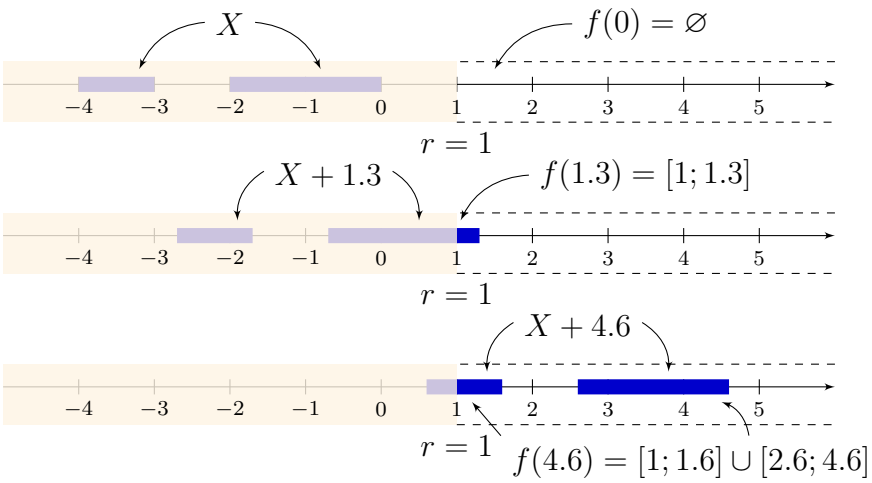


Figure 4.3: The linear timed set $f : t \mapsto (([-4;-3] \cup [-2;0]) + t) \cap [1;+\infty)$ representing all reachable valuations that can be reached in ℓ_2 when observing transition b at time t. The fact that $f(0) = \emptyset$ indicates that transition b cannot be taken at time 0.

In order to deal with all locations of a timed automaton, we use *markings*, which associate a set of valuations with each location of the automaton; similarly, *linear timed markings* associate a linear timed set with each location of the automaton: while sets and linear timed sets represent valuations, markings and linear timed markings represent sets of configurations.

Our algorithm consists in computing linear timed markings representing all configu-

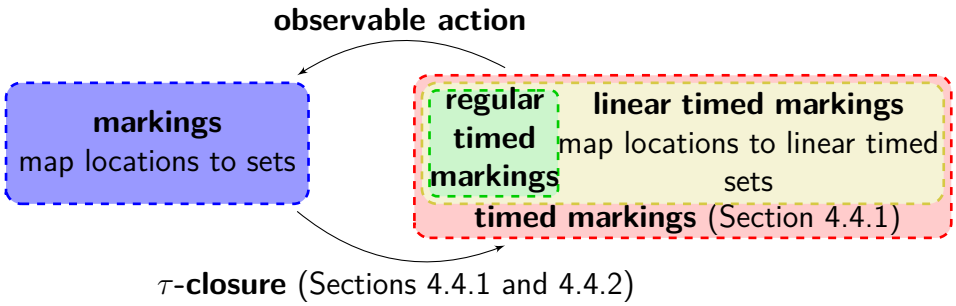


Figure 4.4: Markings, linear timed markings, and related operations

rations that can be reached from a given initial configuration after a given sequence of timed observations (alternations of observable transitions and delay transitions possibly including an arbitrary number of unobservable transitions). Given such a linear timed marking M, when an observation a is received at time t, we can easily compute the marking m containing all possible configurations in which the automaton can end up just after an a-transition is taken. From there, our algorithm computes the set of configurations that can be reached when time elapses (this is a linear timed marking), and the sets of configurations that can be reached by following any possible sequence of silent transitions. We prove that this can be effectively computed and finitely represented as a *regular* timed marking, which we call the τ-closure of m. Figure 4.4 is a graphical representation of those different concepts and operations.

Finally, in Section 4.5 we start extending our technique to timed automata with an arbitrary number of clocks: we prove in particular that linear timed markings are still sufficient to represent the configurations that can be reached after a given sequence of timed observations. But we have not been able to extend our notion of *regular* timed markings to a notion that could be used to properly represent and compute all reachable configurations in the n-clock setting; this is part of our future work.

Our work is most tied with [BJM17] from which automata over timed domains are borrowed, and [Tri02] for the idea of computing sets of states after some finite execution -although we use a precomputation by opposition to Tripakis's fully online method.

4.2 Preliminaries

In this part (focusing on one-clock timed automata), we heavily use sets and intervals of reals with rational bounds, and especially unbounded ones. We first introduce these (and

more generally notations for sets of reals with bounds in any subset of $\mathbb{R}$), and then the model of timed automata and related notions.

4.2.1 Sets and intervals of real

Let X and Y be two subsets of $\mathbb{R}$, we define $X + Y = \{x + y \mid x \in X,\ y \in Y\}$ and $X - Y = \{x - y \mid x \in X,\ y \in Y\}$. For $t \in \mathbb{R}_{\geq 0}$, we write $X + t$ (resp. $X - t$) as a shorthand for $X + \{t\}$ (resp. $X - \{t\}$), used to shift X forward (resp. backward) by t.

For any subset $\mathbb{K}$ of $\mathbb{R}$, we write $\mathcal{I}_{\mathbb{K}}$ for the set of intervals of $\mathbb{R}$ with bounds in $\mathbb{K} \cup \{-\infty, +\infty\}$, and $\mathcal{I}_{\mathbb{K}_{\geq 0}}$ for the set of intervals with bounds in $\mathbb{K}_{\geq 0} \cup \{+\infty\}$.

For $r \in \mathbb{K}$, we define the following sets of $\mathcal{I}_{\mathbb{K}}$:

$$\begin{aligned} \updownarrow r &= [r; +\infty) & \uparrow r &= (r; +\infty) \\ \downarrow r &= (-\infty; r) & \updownarrow r &= (-\infty; r]. \end{aligned}$$

We write $\widehat{\mathbb{K}}_{\geq 0} = \{\updownarrow r, \uparrow r \mid r \in \mathbb{K}_{\geq 0}\}$ for the set of upward-closed intervals in $\mathcal{I}_{\mathbb{K}}$; in the sequel, elements of $\widehat{\mathbb{K}}_{\geq 0}$ are denoted with $\widehat{r}$. Similarly, $\underset{\smile}{\mathbb{K}}_{\geq 0} = \{\updownarrow r, \downarrow r \mid r \in \mathbb{K}_{\geq 0}\} \cup \{\mathbb{R}\}$, and we use notation $\underset{\smile}{r}$ for intervals in $\underset{\smile}{\mathbb{K}}_{\geq 0}$.

The following results are straightforward, and will be useful to ground intuition and in the sequel:

Lemma 4.2.1. *Let $v \in \mathbb{K}_{\geq 0}$, $\widehat{r}$ and $\widehat{s}$ in $\widehat{\mathbb{K}}_{\geq 0}$, and $\underset{\smile}{t}$ in $\underset{\smile}{\mathbb{R}}_{\geq 0}$. Then*

- *if $v \notin \widehat{r}$, then $\widehat{r} \subseteq \uparrow v$;*
- *$\widehat{r} \cap \widehat{s} \in \{\widehat{r}, \widehat{s}\}$;*
- *if $\underset{\smile}{t}$ intersects both $\widehat{r}$ and $\widehat{s}$, then $\widehat{r} \cap \widehat{s} \cap \underset{\smile}{t} \neq \emptyset$.*

4.2.2 One-clock timed automata

We consider in this chapter one (and later n-) clock timed automata without invariants (*i.e.* all invariants are set to true) and equipped with a set of final transitions $\mathcal{F}$ that is used to define their language. We thus note a timed automaton $\mathcal{A} = (\mathcal{L}, \Sigma \uplus \{\tau\}, \{\ell_{init}\}, C, E, \mathcal{F})$ with $\mathcal{F} \subseteq \mathcal{L}$ and omitting the invariant function. Σ is an alphabet of *observable* actions, while τ is an unobservable one. Notice that the automaton is not forced to be deterministic.

As the automata are equipped with a single clock we can partition E by separating E_{id}, the set of *non-resetting* transitions, *i.e.*, having $\emptyset$ as their reset, and E_0 for the complement set of *resetting* transitions.

As for a transition $e = (\ell, g, a, C', \ell')$, the guard g is an interval, it can be written in a unique way as the intersection of an element $\widehat{e} \in \widehat{\mathbb{Q}}_{\geq 0}$ and an element $\underset{\sim}{e} \in \underset{\sim}{\mathbb{Q}}_{\geq 0}$. In the sequel of this chapter, the guard of a transition e will often be written $\widehat{e} \cap \underset{\sim}{e}$.

Remark 4.2.2. *This chapter was originally written with rational guards instead of integer ones. We kept it this way to showcase that the method developed here does not depend on the value of the time unit. For this reason, intervals and (representation of) semantic notions often use $\mathbb{Q}_{\geq 0}$ instead of $\mathbb{N}$. This does not impact most of the discussion, except for Section 4.4.3 (where it is discussed).*

In this chapter we will consider only (partial) runs (resp. signatures) with a strict alternation between delay transitions and action transitions, but not always ending by Σ. As explained in chapter 3, τ does not appear in traces. We write $(\ell, v) \xrightarrow{t}_{\pi} (\ell', v')$ when such a partial run ρ exists with $\mathsf{dur}(\rho) = t$ and π its partial path (*i.e.* in E^*).

For any sequence of observable actions $\lambda \in \Sigma^*$ [1] and any $t \in \mathbb{R}_{\geq 0}$, we write $(\ell, v) \overset{t}{\rightsquigarrow}_{\lambda} (\ell', v')$ whenever there exists a partial run π such that $\lambda = \mathsf{act}(\pi)_{|\{\tau\}}$ and $(\ell, v) \xrightarrow{t}_{\pi} (\ell', v')$.

The untimed language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of words $\lambda \in \Sigma^*$ such that $(\ell_{init}, \mathbf{0}) \overset{t}{\rightsquigarrow}_{\lambda} (\ell', v')$ for some $\ell' \in \mathcal{F}$, some valuation v' and some delay $t \in \mathbb{R}_{\geq 0}$.

In the sequel, we heavily use *markings*, which map locations of $\mathcal{A}$ to sets of clock valuations, thereby representing sets of configurations of $\mathcal{A}$. In our context, a marking of a one-clock timed automaton $\mathcal{A} = (\mathcal{L}, \Sigma \uplus \{\tau\}, \{\ell_{init}\}, C, E, \mathcal{F})$ is a function $m \colon \mathcal{L} \to 2^{\mathbb{R}_{\geq 0}}$ representing the set of configurations $\{(\ell, v) \mid v \in m(\ell)\}$. We note $\mathbb{M} = \{m \mid m \colon \mathcal{L} \to 2^{\mathbb{R}_{\geq 0}}\}$ the set of markings of $\mathcal{A}$ (we omit to reference $\mathcal{A}$ in the $\mathbb{M}$ symbol as it will always be clear from context).

For any $t \in \mathbb{R}_{\geq 0}$, we define the function $\mathbf{O}_t \colon \mathbb{M} \to \mathbb{M}$ by letting, for any $m \in \mathbb{M}$ and any $\ell' \in \mathcal{L}$,

$$\mathbf{O}_t(m) \colon \ell' \in \mathcal{L} \mapsto \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists v \in m(\ell).\ (\ell, v) \overset{t}{\rightsquigarrow}_{\varepsilon} (\ell', v')\},$$

representing the configurations reached from m after observing a delay of t time units.

1. Remember that consider automata over $\Sigma \uplus \{\tau\}$ to distinguish observable and unobservable actions.

Similarly, for any $a \in \Sigma$, we let

$$\mathbf{O}_a(m)\colon \ell' \in \mathcal{L} \mapsto \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists v \in m(\ell).\ \exists e \in \mathsf{enab}((\ell, v)).\ \mathsf{act}(e) = a \wedge (\ell, v) \xrightarrow{0}_e (\ell', v')\}.$$

Then $\mathbf{O}_a(m)$ is the marking representing the set of configurations that can be reached after taking an a-transition from the configurations represented by m.

Remark 4.2.3. $\mathbf{O}_a(m)$ *does not represent the set of configurations that can be reached after* observing *an a-transition, as this would include the possibility of taking unobservable transitions instantly before or after the a-transition. The corresponding marking is $\mathbf{O}_0(\mathbf{O}_a(\mathbf{O}_0(m)))$. We choose this definition because it concentrates the challenges related to silent transitions exclusively in the computation of $\mathbf{O}_t$.*

When all transitions in E are observable (*i.e.* are labelled by observable letters in Σ), the operations $\mathbf{O}_a$ and $\mathbf{O}_t$ can be easily computed. $\mathbf{O}_a$ consists in taking all transitions labelled by a that are enabled in some item of the marking, while $\mathbf{O}_t$ amounts to adding t to each item of the marking (in other terms, for any marking m, any state $\ell \in \mathcal{L}$, and any $v \in \mathbb{R}_{\geq 0}$, we have $v \in m(\ell)$ if, and only if, $v + t \in \mathbf{O}_t(m)(\ell)$).

When timed automata contain a silent (*i.e.* unobservable) action τ, the computation becomes more complex.

Now $(\ell, v) \overset{t}{\rightsquigarrow}_\varepsilon (\ell', v')$ indicates a sequence of zero or more silent transitions in t time units; in that case we may have $\ell' \neq \ell$.

The function $\mathbf{O}_t$ cannot be computed anymore by just shifting valuations by t. In its raw form, the function $\mathbf{O}_t$ can be obtained by the computation of the set of reachable configurations in a delay t by following (arbitrarily long sequences of) silent transitions. This is analogous to the method proposed by Tripakis [Tri02] for keeping track of the set of all possible configurations the automaton can be in after observation $\lambda \in \Sigma^*$ and delay t. In [Tri02], the set of possible configurations is updated each time a new action is observed (or after a time out, if no new observations occur); this approach may increase the delay for detecting the occurrence of faulty actions.

Example 4.2.4. *Consider again the one-clock timed automaton of Figure 4.1, and the (initial) marking m_0 which maps ℓ_0 to the single valuation v_0 such that $v_0(x) = 0$, and locations ℓ_1 and ℓ_2 to the empty set. This marking corresponds to a single valuation.*

Assume that we observe transition b after 1.3 *time units. Obviously, the automaton must have taken one of the transitions from ℓ_1 to ℓ_2, but since they are unobservable,*

we cannot know when this occurred. Actually, the bottom transition requires $3 \leq c \leq 4$*, so it cannot have been used during the first* 1.3 *time units. In the end, it is not hard to check that*

$$\begin{array}{rrcl} \mathbf{O}_{1.3}(m_0)\colon & \ell_0 & \mapsto & \{1.3\} \\ & \ell_1 & \mapsto & [0; 1.3] \\ & \ell_2 & \mapsto & \emptyset \end{array}$$

The b-transition can be taken from the configurations in ℓ_1 *where* $c \geq 1$*. This amounts to applying* $\mathbf{O}_b$ *to the marking above, which results in a marking* $m_{1.3,b}$ *mapping* l_0 *and* ℓ_1 *to the empty set, and* ℓ_2 *to* $[1; 1.3]$*.*

Now, what if, instead, we observe b at time 4.6*? Computing* $\mathbf{O}_{4.6}(m_0)$ *can be done as above, but now taking both transitions between* ℓ_0 *and* ℓ_1 *into account. This results in*

$$\begin{array}{rrcl} \mathbf{O}_{4.6}(m_0)\colon & \ell_0 & \mapsto & \{4.6\} \\ & \ell_1 & \mapsto & [0.6; 1.6] \cup [2.6; 4.6] \\ & \ell_2 & \mapsto & \emptyset \end{array}$$

Then taking transition b is similar to the previous situation: it corresponds to applying $\mathbf{O}_b$*, which results in a marking* $m_{4.6,b}$ *mapping* ℓ_0 *and* ℓ_1 *to the empty set, and* ℓ_2 *to* $[1; 1.6] \cup [2.6, 4.6]$*. We recover the markings represented on Figure 4.2.*

The two following sections will be devoted to the computation of $\mathbf{O}_t$ (and $\mathbf{O}_a$). First Section 4.3 introduces the notion of *linear timed sets*, which we use to represent sets of valuations that evolve over time, and its subclass of *regular timed sets* which allows a finite representation in the case of one-clock timed automata. In Section 4.4 we lift these notions to sets of configurations and their evolution over time, by the notions of *linear timed markings* and *regular timed markings*. We define a τ-closure operator on linear timed markings, and, show that in the case of one-clock timed automata, it can be efficiently computed using regular timed markings, at the expense of some precomputations.

4.3 Regular timed sets

In this section, we define the notion of *linear timed set* to represent sets of clock valuations and their evolution over time. Most important to us is the subclass of *regular timed sets*, that uses regularity (as we will define just below) to ensure a finite representation, and will be key to efficiency in the computation of closure, as will be seen in the next section.

4.3.1 Regular unions of intervals

Before defining linear and regular timed sets, we introduce our notion of regularity for sets.

Definition 4.3.1. *A* regular union of intervals *is a* 4*-tuple* $R = (I, J, p, q)$ *where* I *and* J *are finite unions of intervals in* $\mathcal{I}_{\mathbb{Q}}$ *(*e.g. *intervals of* $\mathbb{R}$ *with bounds in* $\mathbb{Q} \cup \{-\infty, +\infty\}$*),* $p \in \mathbb{Q}_{\geq 0}$ *is the period, and* $q \in \mathbb{N}$ *is the offset. It is required that* $J \subseteq (-p; 0]$ *and* $I \subseteq \overleftarrow{1}(-q \cdot p)$*.*

The regular union of intervals $R = (I, J, p, q)$ *represents the subset of* $\mathcal{I}_{\mathbb{Q}}$ $\mathcal{S}(R) = I \cup \bigcup_{k=q}^{+\infty} (J - k \cdot p)$ *(where* $J - k \cdot p$ *is the interval obtained by shifting* J *by* $-k \cdot p$*).*

A regular union of interval offers an efficient - or at least finite - representation of a structured infinite set of intervals.

Example 4.3.2. *Figure 4.5 shows an example of a regular union of intervals. There, the period is* 1*, the offset is* 3*, and the sets* I *and* J *are as displayed on the figure.*

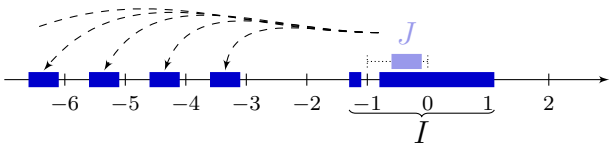


Figure 4.5: Example of a regular union of intervals.

Remark 4.3.3. *The constraints* $J \subseteq (-p; 0]$ *and* $I \subseteq \overleftarrow{1}(-q \cdot p)$ *simply serve to ensure that* I *and the* $J - p \cdot k$ *do not overlap for* $k \geq q$*.*

Regular unions of intervals enjoy the following properties:

Proposition 4.3.4. *Let* R *and* R' *be regular unions of intervals representing the sets of reals* $\mathcal{S}(R)$ *and* $\mathcal{S}(R')$*. Then one can define regular unions of intervals, denoted* $\overline{R}$*,* $R \cup R'$ *and* $R + R'$ *which represent respectively the sets* $\overline{\mathcal{S}(R)}$*,* $\mathcal{S}(R) \cup \mathcal{S}(R')$*, and* $\mathcal{S}(R) + \mathcal{S}(R')$*.*

Proof. We now prove the result for $\overline{\mathcal{S}(R)}$: writing $\mathcal{S}(R) = I \cup \bigcup_{k=q}^{+\infty} J - k \cdot p$, thanks to the constraints imposed on I and J, we have

$$\overline{\mathcal{S}(R)} = (\overleftarrow{1}(-q \cdot p) \setminus I) \cup \bigcup_{k=q}^{+\infty} ((-p; 0] \setminus J) - k \cdot p.$$

For the union we write $R = (I, J, p, q)$ and $R' = (I', J', p', q')$, so that

$$\mathcal{S}(R) = I \cup \bigcup_{k=q}^{+\infty} J - k \cdot p \qquad\qquad \mathcal{S}(R') = I' \cup \bigcup_{k=q'}^{+\infty} J' - k' \cdot p'.$$

We can write $p = \frac{a}{b}$ and $p' = \frac{a'}{b'}$, where a, b, a' and b' are positive integers. Let

$$N = \min\{n \in \mathbb{N} \mid n \cdot b \cdot a' \geq q \text{ and } n \cdot b' \cdot a \geq q'\}.$$

Then we can write

$$\mathcal{S}(R) = \left(I \cup \bigcup_{k=q}^{N \cdot b \cdot a' - 1} J - k \cdot p\right) \cup \bigcup_{k=N}^{+\infty} \left[\left(\bigcup_{r=0}^{b \cdot a' - 1} J - r \cdot p\right) - k \cdot a \cdot a'\right].$$

and

$$\mathcal{S}(R') = \left(I' \cup \bigcup_{k=q'}^{N \cdot b' \cdot a - 1} J' - k \cdot p'\right) \cup \bigcup_{k=N}^{+\infty} \left[\left(\bigcup_{r=0}^{b' \cdot a - 1} J' - r \cdot p'\right) - k \cdot a \cdot a'\right].$$

Since $J \subseteq (-p; 0]$, we have $\bigcup_{r=0}^{b \cdot a' - 1} J - r \cdot p \subseteq (-b \cdot a' \cdot p; 0] = (a \cdot a'; 0]$, and similarly for J'. Taking the union of the equalities above, we get an expression of $\mathcal{S}(R) \cup \mathcal{S}(R')$ under the form $I'' \cup \bigcup_{k=N}^{+\infty}(J'' - k \cdot (a \cdot a'))$, which proves that $\mathcal{S}(R) \cup \mathcal{S}(R')$ can be represented as a regular union of intervals.

The proof for $\mathcal{S}(R) + \mathcal{S}(R')$ follows similar ideas: as for $\mathcal{S}(R) \cup \mathcal{S}(R')$, we first rewrite $\mathcal{S}(R)$ and $\mathcal{S}(R')$ in such a way that they have the same period. Hence we assume w.l.o.g. $R = (I, J, p, q)$ and $R' = (I', J', p, q')$. Since I and J are finite unions of intervals, we can write

$$\mathcal{S}(R) = \bigcup_{k_i} I_{k_i} \cup \bigcup_{k_j} (\bigcup_{k=q}^{+\infty} J_{k_j} - kp)$$

where I_{k_i} and J_{k_j} are intervals. Similarly for $\mathcal{S}(R')$. This way, we can simply consider the following cases, and apply our previous result for union to prove that we end up with a regular union of intervals:

- $I_{k_i} + I'_{k_{i'}}$;
- $I_{k_i} + (\bigcup_{k=q'}^{+\infty} J'_{k_{j'}} - kp)$;
- $(\bigcup_{k=q}^{+\infty} J_{k_j} - kp) + (\bigcup_{k=q}^{+\infty} J'_{k_{j'}} - kp)$.

The first case is trivial. The second and third cases are easy to handle. □

Notice that a regular union of intervals is *finitely representable.*

4.3.2 Linear and regular timed sets

We introduce *linear timed sets* as a way to represent sets of clock valuations (and eventually markings), and how they evolve over time. Informally, a linear timed set represents a mapping from the non-negative reals to the set of subsets of $\mathbb{R}_{\geq 0}$. We begin with the definition of atomic timed sets, which form a special case.

Definition 4.3.5. *An* atomic timed set *is a pair $T = (X; \widehat{r})$ where $X \subseteq \mathbb{R}$ and $\widehat{r} \in \widehat{\mathbb{Q}}_{\geq 0}$. With such a pair $T = (X; \widehat{r})$, we associate a mapping $f_T \colon \mathbb{R}_{\geq 0} \to 2^{\mathbb{R}_{\geq 0}}$ defined as $f_T(t) = (X + t) \cap \widehat{r}$. The set $f_T(t)$ represents the actual valuations after t time units. We call the second component $\widehat{r}$ a* filter*.*

A linear timed set *T is a countable set $\{T_k \mid k \in K\}$ (sometimes also denoted with $\bigsqcup_{k \in K} T_k$) of atomic timed sets. With such a linear timed set, we again associate a mapping $f_T \colon \mathbb{R}_{\geq 0} \to 2^{\mathbb{R}_{\geq 0}}$ defined as $f_T(t) = \bigcup_{k \in K} f_{T_k}(t)$. A linear timed set is* finite *when K is. We write $\mathcal{T}(\mathbb{R})$ for the set of linear timed sets of $\mathbb{R}$.*

For a timed set T after a delay t, we will often call *actual* valuations the elements of $f_T(t) = (X + t) \cap \widehat{r}$ by contrast to *potential* valuations *i.e.* elements of $(X + t) \setminus \widehat{r}$ that are not valuations for this particular delay, but model valuations that will appear for greater delays.

Remark 4.3.6. *Notice that potential and actual valuations heavily depend on the delay t as a potential valuation for a given valuation is an actual valuation for a greater delay.*

Example 4.3.7. *Figure 4.3 displays an example of an atomic timed set $T = (X; \uparrow 1)$, with $X = [-4; -3] \cup [-2; 0]$. The picture displays the sets $f_T(0) = \varnothing$, $f_T(1.3) = [1; 1.3]$, and $f_T(4.6) = [1; 1.6] \cup [2.6; 4.6]$.*

Notice that those sets correspond to the markings reached in the situations of Example 4.1.1 (see Figure 4.2). Actually, T can be used to represent all *clock valuations that may be reached in state ℓ_2 in the automaton depicted on Figure 4.1 (starting from the initial configuration $(\ell_{init}, 0)$), depending on the date at which b is observed.*

Given two linear timed sets T and T', we write $T \sqsubseteq T'$ whenever $f_T(t) \subseteq f_{T'}(t)$ for all $t \in \mathbb{R}_{\geq 0}$. This is a pre-order relation; it is not anti-symmetric as for instance $(\{1\}; \uparrow 0) \sqsubseteq (\{1\}; \uparrow 1)$ and $(\{1\}; \uparrow 1) \sqsubseteq (\{1\}; \uparrow 0)$. We write $T \equiv T'$ whenever $T \sqsubseteq T'$ and $T' \sqsubseteq T$.

Clearly, cycles in timed automata may generate linear timed sets where the set X is infinite. Think of a self-loop silently resetting the clock whenever it reaches 1: the resulting linear timed set would be $(-\mathbb{N}; \uparrow 0)$. We will prove that for keeping track of all the configurations of any one-clock timed automaton, it is always sufficient to use finite unions of atomic timed sets in which the first component has a simple shape, namely that of a regular union of intervals.

Definition 4.3.8. *A* regular timed set *is a finite timed set* $T = \{(X_k; \widehat{r}_k) \mid k \in K\}$ *such that for all* $k \in K$*, the set* $X_k \subseteq \mathbb{R}$ *is* $\mathcal{S}(R_k)$ *the image by* $\mathcal{S}$ *of a regular union of intervals* $R_k = (I_k, J_k, p_k, q_k)$*.*

This defines an adequate structure for representing and manipulating sets of configurations of one-clock timed automata and their evolution over time. In the sequel, we extend linear timed sets into *linear timed markings*, explain how to compute them, and show that *regular* timed markings are (necessary and) sufficient for representing all reachable configurations of partially-observable one-clock timed automata.

4.4 Closure under delay and silent transitions

In this section, we fix a one-clock timed automaton $\mathcal{A} = (\mathcal{L}, \Sigma \uplus \{\tau\}, \{\ell_{init}\}, C, E, \mathcal{F})$, with a unique silent action τ and aim at computing the functions $\mathbf{O}_a$ for any $a \in \Sigma$ and $\mathbf{O}_t$ for any $t \in \mathbb{R}_{\geq 0}$. Computing $\mathbf{O}_a(m)$ for $a \in \Sigma$ is not very involved: for a given location $\ell' \in \mathcal{L}$, for each location $\ell \in \mathcal{L}$ and each transition e labelled with a with source ℓ and target ℓ', it suffices to intersect $m(\ell)$ with the guard $\widehat{e} \cap \underset{\sim}{e}$, and add the resulting interval (or the singleton $\{0\}$ if the intersection with the guard is non-empty and e is a resetting transition) to $\mathbf{O}_a(m)(\ell')$, *i.e.*:

$$\mathbf{O}_a(m)(\ell') = \Big(\bigcup_{(\ell, \widehat{e} \cap \underset{\sim}{e}, a, r, \ell') \in E_{id}} m(\ell) \cap (\widehat{e} \cap \underset{\sim}{e}) \Big) \cup \Big(\bigcup_{(\ell, \widehat{e} \cap \underset{\sim}{e}, a, r, \ell') \in E_0} \big(m(\ell) \cap (\widehat{e} \cap \underset{\sim}{e}) \big)_{[C \leftarrow 0]} \Big).$$

From now on, we only focus on computing $\mathbf{O}_t$, for $t \in \mathbb{R}_{\geq 0}$. For this, it is sufficient to only consider silent transitions of $\mathcal{A}$: we let $U = U_0 \uplus U_{id}$ be the subset of E containing exactly the transitions labelled with τ, partitioned into those transitions that reset the clock (in U_0), and those that do not (in U_{id}). We write $\mathcal{A}_\tau$ for the restriction of $\mathcal{A}$ to silent transitions, and only consider that automaton in the sequel.

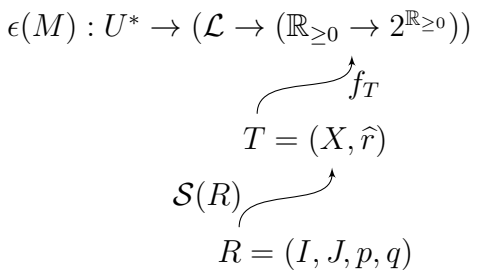


Figure 4.6: Relation between the objects and their representations.

In the following, we first describe the effect of (silent) transitions on markings and use it to define the τ-closure in Section 4.4.1 and then propose an operator ϵ to explicitly compute this closure in Section 4.4.2, using linear timed sets. Finally we prove that this computation and its result can be finitely represented and effectively computed with regular timed sets in Section 4.4.4. This process is summarized in Figure 4.6.

4.4.1 Linear timed markings and their τ-closure

We use markings to represent sets of configurations; in order to compute $\mathbf{O}_t$, we need to represent the evolution of these sets over time. For this, we introduce *timed markings*. A timed marking is a mapping $M \colon \mathcal{L} \to (\mathbb{R}_{\geq 0} \to 2^{\mathbb{R}_{\geq 0}})$. For any delay $t \in \mathbb{R}$, we may (abusively) write $M(t)$ for the marking represented by M after delay t (so that for any $\ell \in \mathcal{L}$ and any $d \in \mathbb{R}_{\geq 0}$, both notations $M(t)(\ell)$ and $M(\ell)(t)$ represent the same subset of valuations in $\mathbb{R}_{\geq 0}$).

For any $\ell \in \mathcal{L}$ and any $t \in \mathbb{R}_{\geq 0}$, $M(\ell)(t)$ is intended to represent all clock valuations that can be obtained in ℓ after a delay of t time units from the marking $M(0)$.

A special case of timed marking are those timed markings M that can be defined using linear timed sets, *i.e.*, for any ℓ, $M(\ell)$ is a mapping f_T for some linear timed set T; timed markings of this kind will be called *linear timed markings* in the sequel. Atomic (resp. finite, regular) timed markings are linear timed markings whose values can be defined using atomic (resp. finite, regular) timed sets (we may omit to mention linearity in these cases to alleviate notations). The structure of these different classes of timed markings can be found in Figure 4.7. As we prove below, regular timed markings are expressive enough to represent how markings evolve over time in one-clock timed automata.

Definition 4.4.1. *We define the union of timed markings M_1 and M_2 as the timed marking such that $M_1 \cup M_2(\ell)(t) = M_1(\ell)(t) \cup M_2(\ell)(t)$. The intersection of two timed markings is*

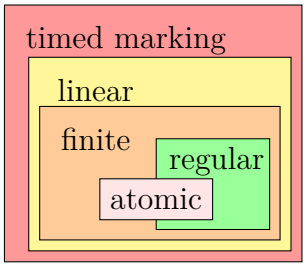


Figure 4.7: The different classes of timed markings.

defined similarly.

Two timed markings are said equivalent *when for all locations ℓ and delays t, $M_1(\ell)(t) = M_2(\ell)(t)$. We write $M_1 \equiv M_2$ when this is the case.*

Proposition 4.4.2. *The union of two linear (resp. finite, regular) timed markings is a linear (resp. finite, regular) timed marking.*

Proof. This can be seen directly by the stability of timed sets and regular unions of intervals by union, which comes respectively by definition and Prop. 4.3.4. □

As we prove below, regular timed markings are expressive enough to represent how markings evolve over time in one-clock timed automata.

We use (linear) timed markings to dispose of a representation in the form of *linear timed sets*, that can store both *actual* valuations, and *potential* valuations that do not exist yet but will exist after some time elapses, as displayed for example in Figure 4.3. This representation has the advantage over markings to be time-independent as a single timed marking represents the set of valuations for all possible delays.

With any marking m, we associate a linear timed marking, which we write $\overrightarrow{m}$, defined as $\overrightarrow{m}(\ell)(t) = \{v + t \mid v \in m(\ell)\}$. This timed marking is linear since it can be defined *e.g.* as $\overrightarrow{m}(\ell) = f_{(m(\ell);\uparrow 0)}$. It can be used to represent all clock valuations that can be reached from marking m after any delay $t \in \mathbb{R}_{\geq 0}$, without taking any transition.

To go further, we define the effect of a partial path $\pi \in U^*$ of silent transitions on a marking m as the following timed marking m^π associating to each delay the marking of reachable configurations:

$$m^\pi : \ell' \mapsto t \mapsto \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}. \exists v \in m(\ell).\ (\ell, v) \xrightarrow{t}_\pi (\ell', v')\}.$$

The set $m^\pi(\ell')(t)$ corresponds to all configurations reachable in ℓ' after reading π from a configuration in m with a delay of exactly t time units. By definition of the transition

relation $\to_\pi$, for $m^\pi(t)$ to be non-empty, π must be a sequence of consecutive transitions. Notice that $m^\varepsilon = \overrightarrow{m}$ as one would expect [2] (and hence is linear). Going further, we can define the τ-closure of m as the timed marking m^τ such that $m^\tau(\ell)(t) = \bigcup_{\pi\in U^*} m^\pi(\ell)(t)$. By definition of $\mathbf{O}_t$, for any delay $t \in \mathbb{R}_{\geq 0}$, we have $\mathbf{O}_t(m) = m^\tau(t)$ for any marking m.

This formalism is easily extended from markings to timed markings by taking for a timed marking M and a partial path of silent transitions $\pi \in U^*$:

$$M^\pi \colon \ell' \mapsto t \mapsto \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}. \exists t_0 \leq t.\ \exists v \in M(\ell)(t_0).\ (\ell, v) \xrightarrow{t-t_0}_\pi (\ell', v')\}$$

and defining $M^\tau(t)(\ell) = \bigcup_{\pi\in U^*} M^\pi(t)(\ell)$.

Remark 4.4.3. *Notice that the definition of M^π differs from the one of m^π only by the addition of an initial delay t_0; this takes into account the fact that some potential configurations may become actual in M. By seeing $M(t_0)$ as a marking, we have*

$$M^\pi \colon \ell' \mapsto t \mapsto \bigcup_{t_0 \leq t} M(t_0)^\pi(\ell')(t - t_0).$$

The following lemmas prove that this extension is self consistent and coherent with what was defined on markings. First, the effect of the empty sequence on a linear timed marking leaves it unchanged:

Lemma 4.4.4. *For any* linear *timed marking M, it holds $M^\varepsilon \equiv M$.*

Proof. Since M is linear, for any ℓ', $M(\ell')$ is a linear timed set, so that $M(\ell')(t) = \bigcup_{k\in\mathbb{N}}(X_k + t) \cap \widehat{r_k}$. On the other hand,

$$\begin{aligned}
M^\varepsilon(\ell')(t) &= \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists t_0 \leq t. \exists v \in M(\ell)(t_0).\ (\ell, v) \xrightarrow{t-t_0}_\varepsilon (\ell', v')\} \\
&= \{v' \in \mathbb{R}_{\geq 0} \mid \exists t_0 \leq t. \exists v \in M(\ell')(t_0).\ (\ell', v) \xrightarrow{t-t_0}_\varepsilon (\ell', v')\} \\
&= \{v' \in \mathbb{R}_{\geq 0} \mid \exists t_0 \leq t. \exists v \in M(\ell')(t_0).\ v' = v + (t - t_0)\} \\
&= \{v' \in \mathbb{R}_{\geq 0} \mid \exists t_0 \leq t.\ \exists k \in \mathbb{N}. \exists v \in (X_k + t_0) \cap \widehat{r_k}.\ v' = v + (t - t_0)\} \\
&= \{v' \in \mathbb{R}_{\geq 0} \mid \exists t_0 \leq t.\ \exists k \in \mathbb{N}. v' \in (X_k + t) \cap \widehat{r_k}\} \\
&= \bigcup_{k\in\mathbb{N}} (X_k + t) \cap \widehat{r_k}.
\end{aligned}$$

2. Remember that in m^ε, ε corresponds to the empty sequence of transitions.

Second, applying a partial path of silent transitions to a marking or to its corresponding linear timed marking yields the same result:

Lemma 4.4.5. *For any marking m and any partial path $\pi \in U^*$, it holds $(\overrightarrow{m})^\pi \equiv m^\pi$.*

Proof. For any $t \in \mathbb{R}_{\geq 0}$ and $\ell' \in \mathcal{L}$, we have

$$\begin{aligned}(\overrightarrow{m})^\pi(t)(\ell') &= \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists t_0 \leq t. \exists v'' \in \overrightarrow{m}(t_0)(\ell).(\ell, v'') \xrightarrow{t-t_0}_\pi (\ell', v')\} \\ &= \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists t_0 \leq t. \exists v \in m(\ell).(\ell, v) \xrightarrow{t_0}_\varepsilon (\ell, v'') \xrightarrow{t-t_0}_\pi (\ell', v')\}\end{aligned}$$

On the other hand,

$$m^\pi(t)(\ell') = \{v' \in \mathbb{R}_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists v \in m(\ell).(\ell, v) \xrightarrow{t}_\pi (\ell', v')\}.$$

It follows that any $v' \in (\overrightarrow{m})^\pi(t)(\ell')$ is in $m^\pi(t)(\ell')$, since $(\ell, v) \xrightarrow{t_0}_\varepsilon (\ell, v'') \xrightarrow{t-t_0}_\pi (\ell', v')$ implies $(\ell, v) \xrightarrow{t}_\pi (\ell', v')$. Conversely, any $v' \in m^\pi(t)(\ell')$ is in $(\overrightarrow{m})^\pi(t)(\ell')$, since if $(\ell, v) \xrightarrow{t}_\pi (\ell', v')$, then taking $t_0 = 0$, we have $(\ell, v) \xrightarrow{t_0}_\varepsilon (\ell, v) \xrightarrow{t-t_0}_\pi (\ell', v')$. □

This result can be generalized in the following way, linking even more the operations m^π (defined on markings) and M^π (defined on linear timed markings):

Corollary 4.4.6. *For any marking m and any two silent partial paths π_1 and π_2 in U^*, it holds $(m^{\pi_1})^{\pi_2} \equiv m^{\pi_1 \cdot \pi_2}$.*

Proof. For any m, π_1 and π_2:

$$\begin{aligned}(m^{\pi_1})^{\pi_2} &= ((\overrightarrow{m})^{\pi_1})^{\pi_2} && \text{(by lemma 4.4.5)} \\ &= (\overrightarrow{m})^{\pi_1 \cdot \pi_2} && \text{(by definition)} \\ &= m^{\pi_1 \cdot \pi_2} && \text{(by lemma 4.4.5)}\end{aligned}$$

Finally, we formally define what we will consider as τ-closures in the sequel:

Definition 4.4.7. *Let M be a timed marking. A timed marking N is a* τ-closure *of M if $N \equiv M^\tau$. The timed marking M is said* τ-closed *if it is a τ-closure of itself.*

Our aim in this section is to compute (a finite representation of) a τ-closure of any given initial marking (defined using regular unions of intervals).

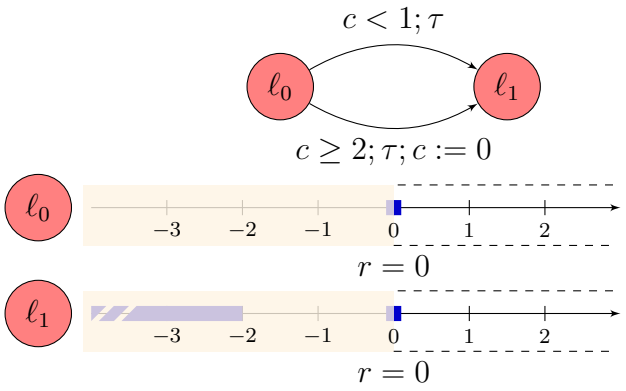


Figure 4.8: A silent timed automaton and its reachable configurations.

Example 4.4.8. *Consider the (silent) timed automaton of Figure 4.8. The initial configuration can be represented by the marking m defined as $m(\ell_0) = \{0\}$ and $m(\ell_1) = \emptyset$, corresponding to the single configuration $\{(\ell_0, c = 0)\}$. It gives rise to a timed marking $\overrightarrow{m}$ defined as $\overrightarrow{m}(\ell_0) = f_{(\{0\};\uparrow 0)}$ and $\overrightarrow{m}(\ell_1) = f_{(\emptyset;\uparrow 0)}$. Write M for this timed marking; M is not closed under silent-transitions, as for instance configuration $(\ell_1, 0)$ is reachable; however, this configuration cannot be reached after any delay: it is only reachable after delay 0, or after a delay larger than or equal to 2 time units. In the end, it can be checked that a τ-*closed *timed marking for this automaton is $M^\tau(\ell_0) = M(\ell_0)$, and $M^\tau(\ell_1) = ((-\infty; -2] \cup [0; 0]; \uparrow 0)$.*

4.4.2 Computing τ-closures

In this subsection, we show how to compute τ-closures. For this, we rely on linear timed sets as a mean to represent $M(\ell)$ for the timed markings M and locations ℓ encountered during the operation.

We will define an operator ϵ computing the effect of a silent transition on a linear timed set, and then extend it to sequences and languages of silent transitions on one hand and to linear timed markings on the other hand. We will also prove that this operator corresponds to the semantic operations on linear timed markings M^π and M^τ used to compute the τ-closure. The approach we have followed so far is summarized in Figure 4.9, with links to the different sections.

Core to the definition of the ϵ operator is the *gauge*, that computes the effect of a resetting transition on a set of (potential) valuations.

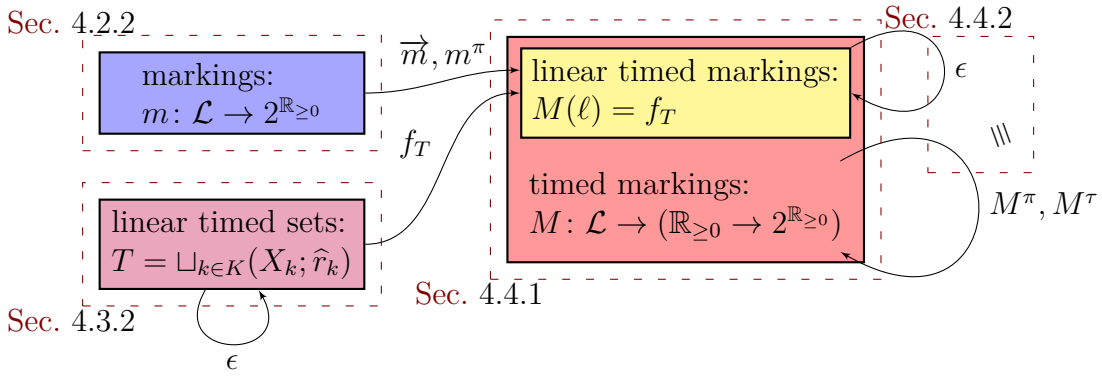


Figure 4.9: Link between the different objects and plan of the 1-clock discussion.

Definition 4.4.9. *Let X and Y be two subsets of $\mathbb{R}$. We define their* gauge *as the set* $X \ltimes Y = (X - Y) \cap \mathbb{R}_{\leq 0}$

This operation can be characterised in the following ways:

Proposition 4.4.10. *Let X and Y be two subsets of $\mathbb{R}$. Then*

$$\begin{aligned} X \ltimes Y &= \{-t \mid t \in \mathbb{R}_{\geq 0} \wedge (Y - t) \cap X \neq \emptyset\} \\ &= \{-t \mid t \in \mathbb{R}_{\geq 0} \wedge (X + t) \cap Y \neq \emptyset\} \end{aligned}$$

Proof. We observe that

$$\begin{aligned} X \ltimes Y &= \{x - y \mid x \in X,\ y \in Y \text{ s.t. } x \leq y\} \\ &= \{-t \mid t \in \mathbb{R}_{\geq 0} \wedge \exists t \in X.\ \exists y \in Y.\ t = y - x\} \\ &= \{-t \mid t \in \mathbb{R}_{\geq 0} \wedge \exists y \in Y.\ y - t \in X\} \\ &= \{-t \mid t \in \mathbb{R}_{\geq 0} \wedge (Y - t) \cap X \neq \emptyset\}. \end{aligned}$$

The other equality is proven similarly. □

These characterizations can be read as "$X \ltimes Y$ is the inverse of the set of delays after which valuations in X satisfy the guard Y". It corresponds to the intuition that after waiting such a delay, the transition can be taken and as it is a resetting one, the clock value becomes 0.

Remark 4.4.11. *These characterizations are specific to the one clock setting. Indeed, $X \ltimes Y$ is by definition a set of* valuations *and only when there is a unique clock do*

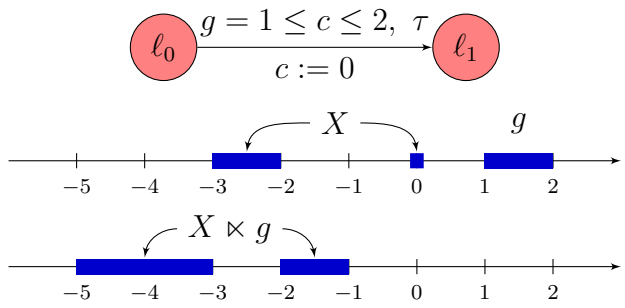


Figure 4.10: Effect of the gauge operator.

valuations and delays correspond.

In general, $X \ltimes Y$ is the set of valuations that will be in $(X + tY)_{[\{c\} \leftarrow 0]}$ (i.e. $\{0\}$) *after t time units. This is the definition used for n-clocks, and it is proven (in Section 4.5) that it corresponds to the current one.*

Example 4.4.12. *Consider the simple transition in Figure 4.10, with guard $g = [1, 2]$. When entering in ℓ_0 with a set of potential configurations corresponding to $X = \{0\} \cup [-3, -2]$ displayed on the bottom of the figure, the result of $X \ltimes g$ is the set of configurations $[-5, -3] \cup [-2, -1]$. To see this, imagine a right shift of X, and consider the delays t during which the two intervals composing X meet g,* i.e. $[1, 2]$ *and* $[3, 5]$*. Each such t is a date at which a configuration $(\ell_1, c = 0)$ can be spawned (because the transition resets the clock); therefore, $-t$ is a potential valuation in ℓ_1. In the end, $(X \ltimes g; \uparrow 0) = ([-5, -3] \cup [-2, -1]; \uparrow 0)$ is the linear timed set of all potential valuations in ℓ_1 in this situation.*

In the following, we use the gauge to define the operator ϵ on linear timed sets and extend it to *linear* timed markings, while proving that it corresponds to the semantic operations M^π and M^τ defined on general timed markings.

Proposition 4.4.13. *The following simple statements will be useful in the sequel:*

- *If $X \leq Y$ (that is, if for any $x \in X$ and any $y \in Y$, it holds $x \leq y$; this is the case in particular if $X \subseteq \mathbb{R}_{\leq 0}$ and $Y \subseteq \mathbb{R}_{\geq 0}$), then $X \ltimes Y = X - Y$;*
- *if $X > Y$* (i.e., *if for any $x \in X$ and any $y \in Y$, it holds $x > y$), then $X \ltimes Y = \emptyset$;*
- *If X and Y are two intervals, then $X \ltimes Y$ is an interval;*
- *If R is a regular union of intervals and Y is an interval, then $\mathcal{S}(R) \ltimes Y$ can be represented as a regular union of intervals noted $R \ltimes Y$;*

- *If* $Y' \subseteq \mathbb{R}_{\geq 0}$, *then* $(X \ltimes Y) \ltimes Y' = (X \ltimes Y) - Y'$.

Proof. The first two claims are trivial from the definition of $X \ltimes Y$. The third claim follows from the fact that $X - Y$ is an interval if X and Y are. The fourth claim follows from Prop. 4.3.4. Finally, the last claim is a consequence of the first one (because $X \ltimes Y \subseteq \mathbb{R}_{\leq 0}$). □

We now define a mapping $\epsilon \colon \mathcal{T}(\mathbb{R}) \times U^* \to \mathcal{T}(\mathbb{R})$, intended to represent the linear timed set that is reached by performing sequences of silent transitions from some given linear timed set. Intuitively, we want to replace semantic operations, such as $M \mapsto M^\pi : \mathbb{M} \times U^* \to \mathbb{M}$ by computations based on their representations (*i.e.* linear timed sets). This will serve to compute a representation of the closure of any linear timed set.

We first consider atomic timed sets, and the application of a single silent transition.

Definition 4.4.14. *For an atomic timed set* $(X; \widehat{r})$ *of* $\mathcal{T}(\mathbb{R})$ *and a transition* $e = (\ell, \widehat{e} \cap \underset{\sim}{e}, \tau, C', \ell')$ *of* U*:*

$$\epsilon((X;\widehat{r}), e) = \begin{cases} (\emptyset; \uparrow 0) & \text{if } \widehat{r} \cap \underset{\sim}{e} = \emptyset \\ (X \cap \underset{\sim}{e}; \widehat{r} \cap \widehat{e}) & \\ \qquad\qquad \text{if } \widehat{r} \cap \underset{\sim}{e} \neq \emptyset \text{ and } C' = \emptyset & \\ (X \ltimes (\widehat{r} \cap \widehat{e} \cap \underset{\sim}{e}); \uparrow 0) & \\ \qquad\qquad \text{if } \widehat{r} \cap \underset{\sim}{e} \neq \emptyset \text{ and } C' = C & \end{cases}$$

The intuition behind these three cases is as follows (see also Figure 4.11 for a graphical illustration, and the proof of Lemma 4.4.17 for the formal arguments):

- if $\widehat{r} \cap \underset{\sim}{e} = \emptyset$, then the transition cannot be taken: it means that the upper bound $\underset{\sim}{e}$ of the guard is smaller than the smallest clock value in $\widehat{r}$ that can be reached;
- if $\widehat{r} \cap \underset{\sim}{e} \neq \emptyset$, then there is some range of delays t such that $f_{(X;\widehat{r})}(t) \cap \widehat{e} \cap \underset{\sim}{e}$ is not empty and the transition can be taken. Furthermore, the transition can be taken from any potential valuation in $X \cap \underset{\sim}{e}$: for valuations in this set the transition will be possible after some delay, while valuations out of $\underset{\sim}{e}$ will never go back in this set. Then after the transition:
  - if the transition does not reset the clock, then the value of the clock is not changed. Then $X \cap \underset{\sim}{e}$ can represent the set of potential valuations. In order for a valuation to be actually reachable, it needs to be actual in $(X; \widehat{r})$ (*i.e.* in $\widehat{r}$) and

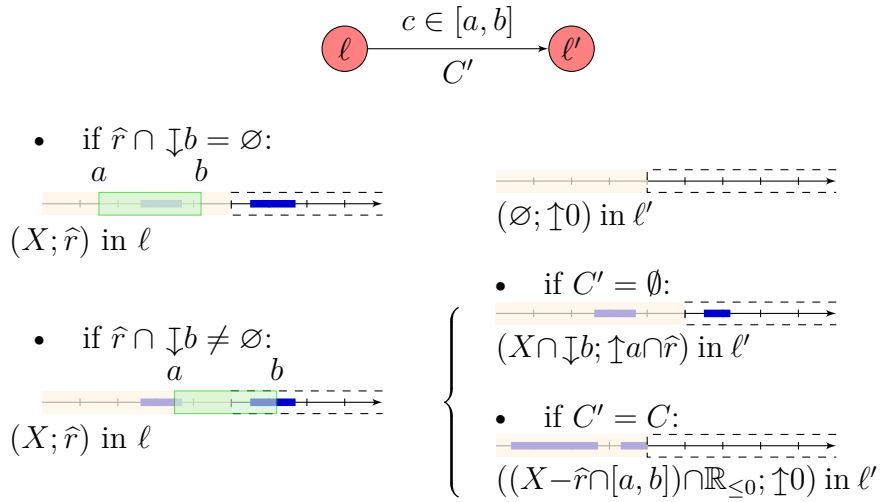


Figure 4.11: Representation of the effect of silent transition $(\ell, [a,b], \tau, C', \ell')$ in three cases.

to have reached $\widehat{e} \cap \underbrace{e}$ (*i.e.* in $\widehat{e}$). Hence the timed set of reachable configurations can be represented by $(X \cap \underbrace{e}; \widehat{r} \cap \widehat{e})$;

– if the transition resets the clock, then the set of reachable values for the clock in ℓ' after delay t_1 corresponds to the set of delays that can be spent in ℓ' (after the clock reset), *i.e.*, the difference between t_1 and the delays t that can be spent in ℓ before taking the transition. Those delays that can be spent in ℓ before taking the transition can be seen to precisely correspond to $X \ltimes (\widehat{r} \cap \widehat{e} \cap \underbrace{e})$. Notice that $\widehat{r} \cap \widehat{e} \cap \underbrace{e}$ corresponds to the set of actual valuations of $f_{(X;\widehat{r})}(t)$ satisfying the guard.

Example 4.4.15. *Consider a linear timed set $T = (X = [-3,-2] \cup \{0\}; \uparrow n)$ with $n \in \mathbb{N}$ representing the configurations reachable in the state ℓ_0 of the automaton in Figure 4.10. If $n = 3$, then for all delays t, $f_T(t) \geq 3$ and it is clear that the transition cannot be taken. Hence, as expected, $\epsilon(T, (\ell_0, \uparrow 1 \cap \downarrow 2, \tau, \{c\}, \ell_1)) = \emptyset$ (this corresponds to the first case of the definition).*

In contrast, if $n = 1$ then all configurations satisfying the guard can be reached after a fitting delay, and $\epsilon(T, (\ell_0, \uparrow 1 \cap \downarrow 2, \tau, \{c\}, \ell_1)) = (X \ltimes g, \uparrow 0)$ with $X \ltimes g = [-5,-3] \cup [-2,-1]$ as displayed on the figure. Notice that the new filter $\uparrow 0$ corresponds to the intuition: all positive clock values can be reached (after a certain delay), as the transition can be taken and it resets the clock.

Definition 4.4.14 (cont.)**.** *We extend ϵ to partial paths inductively as follows:*

$$\epsilon((X;\widehat{r}),\varepsilon) = (X;\widehat{r})$$

and, for $\pi \in U^*$ and $e \in U$,

$$\epsilon((X;\widehat{r}),\pi\cdot e) = \begin{cases} \epsilon(\epsilon((X;\widehat{r}),\pi),e) & \text{if } \mathsf{tgt}(\pi) = \mathsf{src}(e) \\ (\emptyset;\uparrow 0) & \text{otherwise.} \end{cases}$$

Example 4.4.16. *Consider a linear timed set $([-3,-1];\uparrow 2)$, intended to represent the configurations that are reachable in a state ℓ: then after 3 time units, only configuration $(\ell,2)$ can be observed, and after 4.5 time units, the set of reachable configurations in ℓ is $\{(\ell,c) \mid 2 \le c \le 3.5\}$.*

Now, assume that there is a resetting transition from ℓ to ℓ', guarded with $c \in [1;4]$. The set of configurations reachable in ℓ' (originating from the above linear timed set and transition) then is

$$\begin{aligned}([-3;-1] \ltimes (\uparrow 2 \cap \uparrow 1 \cap \downarrow 4);\uparrow 0) &= ([-3;-1] \ltimes [2;4];\uparrow 0) \\ &= ([-7;-3];\uparrow 0).\end{aligned}$$

In particular, assuming we depart from the linear timed set $([-3;-1];\uparrow 2)$ in ℓ, the transition can only take place between 3 and 7 time units.

Definition 4.4.14 (cont.)**.** *We extend this definition to linear timed sets as follows:*

$$\epsilon(\{T_k \mid k \in K\},\pi) = \{\epsilon(T_k,\pi) \mid k \in K\}$$

The ϵ operator is now extended to all linear timed sets and sequences of transitions. We now prove that it indeed corresponds to the effect of taking transitions from a given timed set. For this we do not reason directly on the linear timed sets but on their associated functions f_T. The following lemma states that (informally) $\epsilon(T,\pi)$ corresponds to the effect of following π from T. It is the technical core of the section, linking the semantics and the ϵ operator.

Lemma 4.4.17. *Let T be a linear timed set and $\pi \in U^*$. Then for any $t \in \mathbb{R}_{\ge 0}$ and*

any $v \in \mathbb{R}_{\geq 0}$,

$$v \in f_{\epsilon(T,\pi)}(t) \;\Leftrightarrow\; \exists 0 \leq t_0 \leq t.\ \exists v' \in f_T(t_0).(\mathsf{src}(\pi), v') \xrightarrow{t-t_0}_{\pi} (\mathsf{tgt}(\pi), v).$$

Proof. We carry the proof for the case where T is an atomic timed set. The extension to unions of atomic timed sets is straightforward.

The proof is in two parts: we begin with proving the result for $\pi = \varepsilon$, then for a single transition, and finally proceed by induction to prove the full result.

If $\pi = \varepsilon$, then $v \in f_\epsilon(T, \pi)(t)$ is equivalent to $v \in f_T(t)$, which entails the right-hand-side of the equivalence for $t_0 = t$. Conversely, if $v = v' + (t - t_0)$ for some $v' \in f_T(t_0)$, then, writing $T = (X; \widehat{r})$, we have $v' \in (X + t_0) \cap \widehat{r}$, so that $v \in (X + t) \cap \widehat{r}$, *i.e.* $v \in f_T(t)$.

Now, assume that π is a single transition $e = (\ell, \widehat{e} \cap \underset{\sim}{e}, \tau, C', \ell')$, for which we assume (w.l.o.g.) that $\widehat{e} \cap \underset{\sim}{e} \neq \emptyset$. In case T is empty (*i.e.* $X = \emptyset$), then also $\epsilon(T, e)$ is empty, and the result holds. We now assume that $T = (X; \widehat{r})$ is not empty (*i.e.* $X \neq \emptyset$), and consider three cases, corresponding to the three cases of the definition of $\epsilon(T, e)$:

- if $\widehat{r} \cap \underset{\sim}{e} = \emptyset$, then $\epsilon(T, e) = (\emptyset; \updownarrow 0)$. On the other hand, for any t_0 and any $v' \in f_T(t_0)$, it holds $v' \in \widehat{r}$, so that $v' \notin \underset{\sim}{e}$, and the transition cannot be taken from that valuation (even after some delay). Hence both sides of the equivalence evaluate to false, and the equivalence holds.

- now assume that $\widehat{r} \cap \underset{\sim}{e} \neq \emptyset$, and consider the case where e does not reset the clock. Write $\widehat{p} = \widehat{r} \cap \widehat{e}$, and pick $v \in f_{\epsilon(T,\pi)}(t) = f_{(X \cap \underset{\sim}{e}; \widehat{p})}(t)$ (assuming one exists). Then $v \in (X + t) \cap (\underset{\sim}{e} + t) \cap \widehat{p}$; this implies $\widehat{p} \cap \underset{\sim}{e} \cap [v - t; v] \neq \emptyset$: indeed,

  - if $v \in \underset{\sim}{e}$ or $v - t \in \widehat{p}$, then the result is trivial;
  - otherwise, $v \notin \underset{\sim}{e}$ implies $\underset{\sim}{e} \subseteq \downarrow v$, and $v - t \notin \widehat{p}$ implies $\widehat{p} \subseteq \uparrow v - t$, so that $\widehat{p} \cap \underset{\sim}{e} \subseteq (v - t; v)$. Moreover, the fact that both $\widehat{r} \cap \underset{\sim}{e}$ and $\widehat{e} \cap \underset{\sim}{e}$ are non-empty implies that also $\widehat{p} \cap \underset{\sim}{e} = \widehat{r} \cap \widehat{e} \cap \underset{\sim}{e}$ is non-empty.

  Then for any v' in that set $\widehat{p} \cap \underset{\sim}{e} \cap [v-t; v]$, letting $t_0 = v' - (v - t)$, we have $v' \in X + t_0$. In the end, $v' \in f_T(t_0)$, and $v' \in \widehat{e} \cap \underset{\sim}{e}$, so that $(\mathsf{src}(e), v') \xrightarrow{t-t_0}_{e} (\mathsf{tgt}(e), v)$.

  Conversely, if $t_0 \in [0; t]$ and $v' \in f_T(t_0)$ exist such that $(\mathsf{src}(\pi), v') \xrightarrow{t-t_0}_{\pi} (\mathsf{tgt}(\pi), v)$, then $v' \in X + t_0 \cap \widehat{r}$, and for some $t_1 \leq t - t_0$, $v' + t_1 \in \widehat{e} \cap \underset{\sim}{e}$. Then $v = v' + t - t_0$ since t does not reset the clock; also, since $v' \in X + t_0 \cap \widehat{r}$, we have $v \in (X + t) \cap \widehat{r}$; finally,

from $v' + t_1 \in \widehat{e} \cap \underbrace{e}$, we get $v \in \widehat{e} + (t - t_0 - t_1) \subseteq \widehat{e}$ and $v \in \underbrace{e} + (t - t_0 - t_1) \subseteq \underbrace{e} + t$. In the end, $v \in ((X + \underbrace{e}) + t) \cap (\widehat{r} \cap \widehat{e}) = f_{\epsilon(T,e)}(t)$.

- we finally consider the case where $\widehat{r} \cap \underbrace{e} \neq \emptyset$ and e resets the clock. In this case, $v \in f_{\epsilon(T,\pi)}(t)$ means that $v \in \uparrow 0$ and $v - t \in X \ltimes (\widehat{p} \cap \underbrace{e})$, which rewrites as $0 \leq v \leq t$ and $(X + t - v) \cap (\widehat{p} \cap \underbrace{e}) \neq \emptyset$ (by Prop. 4.4.13). Let $t_0 = t - v$. The property above entails that $0 \leq t_0 \leq t$, and that there exists some $v' \in (E + t_0) \cap (\widehat{p} \cap \underbrace{e})$, so that $0 \leq t_0 \leq t$, $v' \in f_T(t_0)$ and $(\mathsf{src}(e), v') \xrightarrow{t-t_0}_e (\mathsf{tgt}(e), v)$. Conversely, if those conditions hold, then for some $0 \leq t_1 \leq t - t_0$, we have $v' + t_1 \in \widehat{e} \cap \underbrace{e}$, and $v = t - (t_0 + t_1) \geq 0$ (remember that e resets the clock). Then $v' + t_1 \in X + (t_0 + t_1) \cap \widehat{r} \cap \widehat{e} \cap \underbrace{e}$, so that $-(t_0 + t_1) \in X \ltimes (\widehat{p} \cap \underbrace{e})$, and finally $v \in f_{\epsilon(T,\pi)(t)}$.

We now consider the case of $\pi \cdot e$, assuming that the result holds for $\pi \in U^+$. In case $\mathsf{tgt}(\pi) \neq \mathsf{src}(e)$, the result is trivial. Otherwise, first assume that $v' \in (\epsilon(T, \pi \cdot e))(t)$, and let $T' = \epsilon(T, \pi)$. Then $v' \in f_{\epsilon(T',e)}(t)$, thus there exist $0 \leq t_0 \leq t$ and $v \in f_{T'}(t_0)$ s.t. $(\mathsf{src}(e), v) \xrightarrow{t-t_0}_e (\mathsf{tgt}(e), v')$. Since $v \in f_{T'}(t_0)$, there must exist $0 \leq t_1 \leq t_0$ and $v'' \in f_T(t_1)$ such that $(\mathsf{src}(\pi), v'') \xrightarrow{t_0-t_1}_\pi (\mathsf{tgt}(\pi), v)$. We thus have found $0 \leq t_1 \leq t$ such that $(\mathsf{src}(\pi), v'') \xrightarrow{t-t_1}_{\pi \cdot e} (\mathsf{tgt}(e), v')$.

Conversely, if $(\mathsf{src}(\pi), v'') \xrightarrow{t-t_1}_{\pi \cdot e} (\mathsf{tgt}(e), v')$ for some $0 \leq t_1 \leq t$ and $v'' \in f_T(t_1)$, then we have $(\mathsf{src}(\pi), v'') \xrightarrow{t_0-t_1}_\pi (\mathsf{tgt}(\pi), v) \xrightarrow{t-t_0}_e (\mathsf{tgt}(e), v')$ for some $t_0 \in [t_1; t]$ and some v. In that case, we prove that $v \in f_{\epsilon(T,\pi)}(t_0)$: indeed, we have $t_1 \in [0; t_0]$, and $v'' \in f_T(t_1)$ such that $(\mathsf{src}(\pi), v'') \xrightarrow{t_0-t_1}_\pi (\mathsf{tgt}(\pi), v)$, which by induction hypothesis entails $v \in f_{\epsilon(T,\pi)}(t_0)$. Thus we have $0 \leq t_0 \leq d$ and $v \in f_{T'}(t_0)$, where $T' = \epsilon(T, \pi)$, such that $(\mathsf{tgt}(\pi), v) \xrightarrow{t-t_0}_e (\mathsf{tgt}(e), v')$, which means $v' \in f_{\epsilon(T',e)}(t)$, and concludes the proof. □

This results is the first link between ϵ and the semantics of timed automata, and the main technical result of the section, the following ones stemming from it. Thanks to this characterization of $\epsilon(T, \pi)$, we get:

Corollary 4.4.18. *For any sequence π of silent transitions, and any two equivalent linear timed sets T and T', the linear timed sets $\epsilon(T, \pi)$ and $\epsilon(T', \pi)$ are equivalent.*

This corollary ensures that we do not depend on the representation of the structure for our results since the mapping ϵ defined on linear timed sets preserves the equivalence of markings that they represent.

Finally, we extend ϵ to linear timed markings in the expected way:

Definition 4.4.14 (cont.)**.** *Given a linear timed marking M, and a sequence π of transitions, we let:*

$$\begin{aligned} \epsilon(M,\pi)\colon \ell &\mapsto f_{\epsilon(T_{M(\mathsf{src}(\pi))},\pi)} && \text{if } \ell = \mathsf{tgt}(\pi), \\ \ell &\mapsto f_{(\emptyset;\mathbb{1}0)} && \text{otherwise.} \end{aligned}$$

with $T_{M(\mathsf{src}(\pi))}$ a timed set such that $f_T = M(\mathsf{src}(\pi))$.

Since $\epsilon(T_1 \sqcup T_2, \pi) = \epsilon(T_1,\pi) \sqcup \epsilon(T_2,\pi)$, we also have $\epsilon(M_1 \sqcup M_2, \pi) \equiv \epsilon(M_1,\pi) \sqcup \epsilon(M_2,\pi)$ when applied to linear timed markings. Then, we can extend the link between ϵ and the semantics to linear timed markings. This is the main result of the section, as it formally establishes the correspondence between the semantic computation of the effect of silent transitions and the operator ϵ that we build to compute it.

Theorem 4.4.19. *$\epsilon(M,\pi) \equiv M^\pi$ for all $\pi \in U^*$ and all linear timed marking M.*

Proof. For $t \in \mathbb{R}_{\geq 0}$ and $\ell \in \mathcal{L}$, we have

$$M^\pi(t)(\ell) = \{v \in \mathbb{R}_{\geq 0} \mid \exists \ell' \in \mathcal{L}.\ \exists t_0 \leq t. \exists v' \in M(t_0)(\ell').\ (\ell', v') \xrightarrow{t-t_0}_\pi (\ell, v)\}.$$

Hence clearly $M^\pi(\ell) \equiv f_{(\emptyset;\mathbb{1}0)}$ if $\ell \neq \mathsf{tgt}(\pi)$. When $\ell = \mathsf{tgt}(\pi)$, we have

$$M^\pi(t)(\ell) = \{v \in \mathbb{R}_{\geq 0} \mid \exists t_0 \leq t. \exists v' \in M(t_0)(\mathsf{src}(\pi)).\ (\mathsf{src}(\pi), v') \xrightarrow{t-t_0}_\pi (\ell, v)\}.$$

By Lemma 4.4.17, this corresponds to $\epsilon(M,\pi)(\ell)(t)$. □

Finally, we extend ϵ from sequences of transitions to languages.

Definition 4.4.20. *For a timed marking M and a language $\mathcal{L} \subseteq U^*$, we let*

$$\epsilon(M,\mathcal{L}) = \bigsqcup_{\pi \in \mathcal{L}} \epsilon(M,\pi)$$

and we note $\epsilon(M) = \epsilon(M, U^*)$.

From this definition and the previous statements, we immediately get:

Corollary 4.4.21. *For any linear timed marking M, it holds $\epsilon(M) \equiv M^\tau$. It comes that ϵ correctly implements se semantics.*

It follows that the τ-closure of any marking (in particular the initial marking) can be represented as a linear timed marking. However, this linear timed marking is currently defined as an infinite union over all sequences of consecutive silent transitions. We make the computation more effective (and representable) in the Section 4.4.4.

4.4.3 Necessity of regular timed sets

The next subsection will prove that regular timed sets are enough to express (and effectively compute) the set of configurations reached by a one-clock timed automaton. To complete this result, we prove that any regular timed marking can appear in the τ-closure of a one-clock timed automaton.

Remark 4.4.22. *In this subsection, we consider timed automata with guards in $\mathbb{Q}_{\geq 0}$. This is necessary to encode the regular timed markings, as their intervals and periods can be rationals.*

Conversely, if it is desired to limit the discussion to automata with integer guards, it suffices to consider regular timed markings with integer bounded intervals and integer periods.

Proposition 4.4.23. *Given a regular timed set T, there exists a one-clock timed automaton $\mathcal{A}$ such that T represents the τ-closure of $\mathcal{A}$.*

Proof. The proof is first made for an atomic regular timed set $T = (X; \widehat{r})$ defined using a regular union of intervals $R = (I, J, p, q)$. The generalization to finite unions of atomic regular timed sets is then proposed at the end. We prove this result by constructing a timed automaton. For this, we first explain how to encode intervals with bounds in $\mathbb{Q}_{\leq 0} \cup \{-\infty\}$, regular repetitions, finite unions, and how to add a filter $\widehat{r}$. Then we construct an automaton for T.

- Consider an interval I with bounds in $\mathbb{Q}_{\leq 0} \cup \{-\infty\}$. In the automaton represented in Figure 4.12, we start from the initial (atomic regular) timed marking M_{init} associating $f_{(\{0\};\uparrow 0)}$ with the initial location ℓ_{init} and $f_{(\emptyset;\uparrow 0)}$ with any other location. The τ-closure $M^{\tau}_{init} = \epsilon(M_{init})$ of M_{init} associates $f_{(\{0\};\uparrow 0)}$ with ℓ_{init}, and $f_{(I;\uparrow 0)}$ with ℓ_1. Indeed, by definition of ϵ, the closure is computed as: $(\{0\} \ltimes -I \cap \uparrow 0; \uparrow 0) = (\{0\} \ltimes -I; \uparrow 0) = (I; \uparrow 0)$.

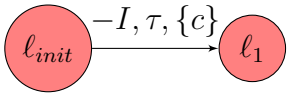


Figure 4.12: An automaton whose closure in ℓ_1 is $(I, \uparrow 0)$.

- Consider a rational $p \in \mathbb{Q}_{\leq 0}$, an integer q, an interval $J \in (-p; 0]$ with rational bounds, and the automaton depicted in Figure 4.13. As argued in the previous case,

starting from M_{init}, the effect of the transition $\ell_{init} \to \ell_1$ through ϵ is represented by $(J - p \cdot q; \uparrow 0)$. Then it is easy to see that the τ-closure is represented by $(\bigcup_{k=q}^{\infty} J - k \cdot p; \uparrow 0)$ in ℓ_1. Indeed, the effect of one application of the self-loop has the form $\cdot \ltimes \{p\}$, effectively repeating the pattern shifted by $-p$. The fixpoint then is $(\bigcup_{k=q}^{\infty} J - k \cdot p; \uparrow 0)$ in ℓ_1.

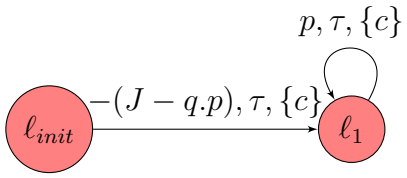


Figure 4.13: An automaton whose closure in ℓ_1 is $(\bigcup_{k=q}^{\infty} J - k.p; \uparrow 0)$.

- The general method to combine a finite number of atomic timed sets representing different behaviours ending in the same configuration is to regroup them in a finite timed set (*i.e.*, a finite collection of atomic timed sets). When all the atomic timed sets share the same constraint $\widehat{r}$, an equivalent method is to take the explicit union of their first part. As argued in Prop. 4.3.4, when the atomic timed sets are regular, this yields an atomic regular timed set. A simple example of such unions is shown in Figure 4.14.

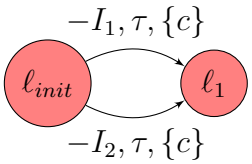


Figure 4.14: An automaton having $(I_1 \cup I_2; \uparrow 0)$ as closure in ℓ_1.

- Adding a stricter filter $\widehat{r}$ to a linear timed set through a transition is straightforward: it is the exact effect of a non-resetting transition of guard $\widehat{r}$.

Using these components, we can build an automaton using $T = (\mathcal{S}(R); \widehat{r})$ to encode the τ-closure of the initial timed marking. A possible example is depicted in Figure 4.15 for $R = (\bigcup_{k=1}^{n_I} I_k, \bigcup_{k=1}^{n_J} J_k, p, q)$, where the dotted line is meant to represent the n_I and n_J transitions. In this automaton, the closure can be represented by $(\mathcal{S}(R); \uparrow 0)$ in ℓ_R and T is ℓ_f.

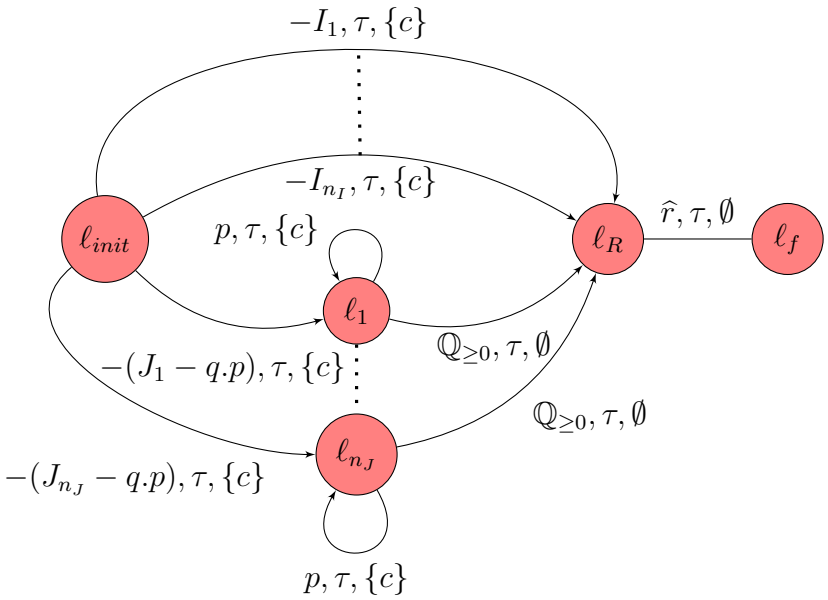


Figure 4.15: An automaton using T to encode the closure in ℓ_f.

It is clear to see that for a regular timed set $T = \{(X_k; \widehat{r}_k) \mid k \in K\}$, as K is by definition finite, building all components necessary to accept the $(X_k; \widehat{r}_k)$ with the same initial and final location yields a finite timed automaton with 1 clock requiring T to represent its τ-closure. □

Remark 4.4.24. *Notice that we rely on the intervals defining a regular union of intervals having bounds in $\mathbb{Q}_{\leq 0}$, in order to define automaton guards with their additive inverse.*

4.4.4 Finite representation of the closure

In Section 4.4.2, we demonstrated how the τ-closure of any linear timed marking (and hence of any marking) can be computed as a linear timed marking using the ϵ operator. To complete this result, we prove in this subsection that when starting from regular timed markings, the τ-closure can be computed with regular timed markings. This presents two main interests:

- First, it corresponds to the semantics of timed automata, as the initial timed marking is clearly regular and we have proved that any regular timed set can appear in a τ-closure (Prop. 4.4.23),

- second, linear timed markings rely on linear timed sets, which in general use infinite unions of sets of $\mathbb{R}$. In order to get an effective algorithm, we need a finite representation of sets, which regular unions of intervals provide.

The following theorem is the general result we prove in the following discussion. We call *language of transitions* of a timed automaton $\mathcal{A}$, noted $\mathcal{L}_E(\mathcal{A})$, the language obtained by reading the names of the transitions instead of the labels.

Theorem 4.4.25. *Consider a timed automaton $\mathcal{A}$ and a regular language $\mathcal{L} \subseteq \mathcal{L}_E(\mathcal{A})$ included in its language of transitions. Given a regular timed marking M, the timed marking $\epsilon(M, \mathcal{L})$ is regular.*

Thanks to this theorem and the fact that the timed marking corresponding to the initial valuation is regular, we can deduce the following facts:

Corollary 4.4.26. *Given a timed automaton $\mathcal{A}$ with τ transitions and its silent fragment A_τ:*

- *the initial (timed) marking of $\mathcal{A}_\tau$ is regular and*
- *the τ-closure of any regular (timed) marking is regular.*

This result entails that regular timed markings are indeed enough to represent and compute the τ-closures.

The remaining of this subsection is a proof of the previous Theorem 4.4.25. We first prove in the Paragraph 4.4.4 that the linear timed marking $\epsilon(M, \mathcal{L})$ is a *finite* timed marking, by discussing the set of possible filters. Then we prove that it is a regular timed marking in Paragraph 4.4.4.

Finiteness

Finiteness is achieved by separating the initial timed marking per location, and then proving that there is only a finite amount of possible filters.

Let M be a regular timed marking: then M can be written as the finite union of atomic regular timed markings $M_{(\ell,X,\widehat{r})}$, defined as $M_{(\ell,X,\widehat{r})}(\ell) = f_{(X;\widehat{r})}$ and $M_{(\ell,X,\widehat{r})}(\ell') = f_{(\emptyset;\uparrow 0)}$ for all $\ell' \neq \ell$. Thus $\epsilon(M, \mathcal{L})$ is the function associating to an input the union of the corresponding outputs of $\epsilon(M_{(\ell,X,\widehat{r})}, \mathcal{L})$ and it suffices to compute the result for atomic regular timed markings $M_{(\ell,X,\widehat{r})}$.

We write $\epsilon_1((X;\widehat{r}),\pi) \subseteq \mathbb{R}$ and $\epsilon_2((X;\widehat{r}),\pi) \in \widehat{\mathbb{R}}_{\geq 0}$ for the first and second components of $\epsilon((X;\widehat{r}),\pi)$. Notice that $\epsilon_2((X;\widehat{r}),\pi)$ does not depend on X (so that we may denote it with $\epsilon_2(\widehat{r},\pi)$ in the sequel). In particular,

- $\epsilon_2((X;\widehat{r}),\pi) = \uparrow\!0$ if $\pi \in U^* \times U_0$ is a sequence of consecutive transitions ending with a resetting transition;
- $\epsilon_2((X;\widehat{r}),\pi) = \widehat{r} \cap \bigcap_{k<n} \widehat{e}_k$ if $\pi = e_1 \dots e_n \in {U_{id}}^*$ is a sequence of consecutive non-resetting transitions.

Letting $\mathbb{J}_{\widehat{r}} = \{\uparrow\!0, \widehat{r}\} \cup \{\widehat{e} \mid e \in U_{id}\}$, it follows that $\epsilon_2((X;\widehat{r}),\pi) \in \mathbb{J}_{\widehat{r}}$ for any $(X;\widehat{r})$ and any π (because by Lemma 4.2.1, $\widehat{r} \cap \widehat{e}$ is either $\widehat{r}$ or $\widehat{e}$, for any $\widehat{r}$ and $\widehat{e}$ in $\widehat{\mathbb{R}}_{\geq 0}$). Hence $\epsilon(M_{(\ell,X,\widehat{r})})$, and thus also $\epsilon(M,\mathcal{L})$, can be written as a finite union of atomic timed markings as $\mathbb{J}_{\widehat{r}}$ is a finite set. By definition, $\epsilon(M,\mathcal{L})$ is thus a finite timed marking.

Regularity

In the following, we reason on each $M_{(\ell,X,\widehat{r})}$ separately. To prove regularity, we first introduce some more notations and state Lemma 4.4.27, that allows to rewrite ϵ using the properties of the gauge. Then we separate $\epsilon(M_{(\ell,X,\widehat{r})},\mathcal{L})$ as we did for M by separating first the ending locations and then the ending filters. For this we separate $\mathcal{L} = \bigcup_{\ell,\ell' \in \mathcal{L},\ \widehat{r},\widehat{r}' \in \mathbb{J}_{\widehat{r}}} [\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r}'}$. The proof that $\epsilon(M_{(\ell,X,\widehat{r})}, [\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r}'})$ is indeed regular is encapsulated in Lemma 4.4.28.

First, some more formalism:

- we let $\widehat{G_{id}} = \{\widehat{e} \mid e \in U_{id}\}$ and $\underbrace{G_{id}} = \{\underbrace{e} \mid e \in U_{id}\}$. We thus have $\mathbb{J}_{\widehat{r}} = \{\uparrow\!0, \widehat{r}\} \cup \widehat{G_{id}}$;
- for $\widehat{r} \in \widehat{\mathbb{R}}_{\geq 0}$ and $e \in U$, we write $\Phi(\widehat{r},e)$ for the interval $\widehat{r} \cap \widehat{e} \cap \underbrace{e}$;
- we define a mapping $\mathcal{J}_{\widehat{r}}\colon U^* \to \mathbb{N}^{\mathbb{J}_{\widehat{r}} \times U_0}$ that counts the number of occurrences of certain timing constraints at resetting transitions along a path: precisely, it is defined inductively as follows (where $\uplus$ represents addition of an element to a multiset):

$$\begin{aligned} \mathcal{J}_{\widehat{r}}(\varepsilon) &= \{0\}^{\mathbb{J}_{\widehat{r}} \times U_0} \\ \mathcal{J}_{\widehat{r}}(\pi \cdot e) &= \mathcal{J}_{\widehat{r}}(\pi) \uplus \{(\epsilon_2(\widehat{r},\pi), e)\} && \text{if } e \in U_0 \\ \mathcal{J}_{\widehat{r}}(\pi \cdot e) &= \mathcal{J}_{\widehat{r}}(\pi) && \text{if } e \in U_{id}. \end{aligned}$$

The idea behind $\mathcal{J}_{\widehat{r}}$ is the following: consider a timed set $(X;\widehat{r})$, and a resetting silent transition guarded with $c \in \underbrace{e} \cap \widehat{e}$. Resetting clock c when it is in $\widehat{r} \cap \underbrace{e} \cap \widehat{e}$ amounts to

substracting to c some value in that interval. The function $\mathcal{J}_{\widehat{r}}$ precisely counts the number of occurrences of each of those intervals along a sequence π of silent transitions. Since there is only one clock, the order of the transitions is not important. Lemma 4.4.27 formalizes this intuition:

Lemma 4.4.27. *Let $(X;\widehat{r})$ be an atomic timed set with $X \subseteq \mathbb{R}_{\leq 0}$, and $\pi \in U^*$. Then either $\epsilon_1((X;\widehat{r}),\pi) = \emptyset$, or*

$$\begin{aligned}\epsilon_1((X;\widehat{r}),\pi) &= X - \sum_{J=(\widehat{y},e)\in \mathbb{J}_{\widehat{r}}\times U_0} \mathcal{J}_{\widehat{r}}(\pi)(J) \times \Phi(\widehat{y},e) \\ &= \left\{ x - \sum_{J\in \mathbb{J}_{\widehat{r}}\times U_0} \mathcal{J}_{\widehat{r}}(\pi)(J)\cdot y_J \;\middle|\; x \in X \text{ and } y_J \in \Phi(J) \text{ for all } J \in \mathbb{J}_{\widehat{r}}\times U_0 \right\}.\end{aligned}$$

Proof. The result is straightforward for $\pi = \epsilon$. Now, assume the result holds for some $\pi \in U^*$, and consider $\pi' = \pi \cdot e$.

Assuming that $\epsilon_1((X;\widehat{r}),\pi) \neq \emptyset$, we first consider the case where $e \in U_{id}$: then

$$\epsilon_1((X;\widehat{r}),\pi\cdot e) = \epsilon_1((X;\widehat{r}),\pi) \cap \underset{\sim}{e}$$

(by Definition 4.4.14 of ϵ). Since $\epsilon_1((X;\widehat{r}),\pi) \subseteq \mathbb{R}_{\leq 0} \subseteq \underset{\sim}{e}$, we have $\epsilon_1((X;\widehat{r}),\pi\cdot e) = \epsilon_1((X;\widehat{r}),\pi)$. Since $\mathcal{J}_{\widehat{r}}(\pi\cdot e) = \mathcal{J}_{\widehat{r}}(\pi)$ when $e \in U_{id}$, our result follows.

Now assume $e \in U_0$: we have

$$\epsilon_1((X;\widehat{r}),\pi\cdot e) = \epsilon_1((X;\widehat{r}),\pi) \ltimes (\epsilon_2(\widehat{r},\pi) \cap \widehat{e} \cap \underset{\sim}{e}).$$

Since $\epsilon_1((X;\widehat{r}),\pi) \subseteq \mathbb{R}_{\leq 0}$ and $\epsilon_2(\widehat{r},\pi) \subseteq \mathbb{R}_{\geq 0}$, Prop. 4.4.13 entails

$$\epsilon_1((X;\widehat{r}),\pi\cdot e) = \epsilon_1((X;\widehat{r}),\pi) - (\epsilon_2(\widehat{r},\pi) \cap \widehat{e} \cap \underset{\sim}{e}).$$

Then, since $\Phi(\epsilon_2(\widehat{r},\pi),e) = \epsilon_2(\widehat{r},\pi)\cap\widehat{e}\cap\underset{\sim}{e}$, substracting $\epsilon_2(\widehat{r},\pi)\cap\widehat{e}\cap\underset{\sim}{e}$ precisely corresponds to the effect of adding $\{(\epsilon_2(\widehat{r},\pi),e)\}$ to the multiset $\mathcal{J}_{\widehat{r}}(\pi)$. □

Let $M_{(\ell,X,\widehat{r})}$ be an atomic regular timed marking. As regular timed markings are defined using linear timed sets, we extend the notations ϵ_1 and ϵ_2 to regular timed markings as the functions returning resp. mappings from locations to sets (that we want to define using regular unions of intervals), and elements of $\widehat{\mathbb{R}}_{\geq 0}$, such that for any regular timed marking M, we have $\epsilon(M,\pi)\colon \ell \mapsto f_{(\epsilon_1(M,\pi)(\ell);\epsilon_2(M,\pi)(\ell))}$. For any state ℓ' of $\mathcal{A}_\tau$, we let

$\mathcal{L}(\ell,\ell')$ be the set of all sequences of consecutive transitions from ℓ to ℓ' in $\mathcal{A}_\tau$. Then

$$(M_{(\ell,X,\widehat{r})})^\tau = \bigsqcup_{\ell'\in\mathcal{L}} \epsilon(M_{(\ell,X,\widehat{r})}, \mathcal{L}(\ell,\ell')).$$

Hence it suffices to prove that $\epsilon(M_{(\ell,X,\widehat{r})}, \mathcal{L}(\ell,\ell'))$ is regular.

For any set $\mathcal{L}$ of sequences of consecutive transitions, and for any $\widehat{r}$ and $\widehat{r}'$ in $\widehat{\mathbb{R}}_{\geq 0}$, we let $\mathcal{L}_{\widehat{r}}^{\widehat{r}'} = \{\pi \in \mathcal{L} \mid \widehat{r}' = \epsilon_2(\widehat{r},\pi)\}$. One easily observes that for any $\widehat{r}\in\widehat{\mathbb{R}}_{\geq 0}$ and any $\mathcal{L}$, it holds $\mathcal{L} = \bigcup_{\widehat{r}'\in\mathbb{J}_{\widehat{r}}} \mathcal{L}_{\widehat{r}}^{\widehat{r}'}$, so that $\epsilon(M_{(\ell,X,\widehat{r})}, \mathcal{L}(\ell,\ell'))$ can be written as

$$\bigsqcup_{\widehat{r}'\in\mathbb{J}_{\widehat{r}}} \ell'' \mapsto f_{\left(\epsilon_1(M_{(\ell,X,\widehat{r})}, [\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r}'})(\ell'');\widehat{r}'\right)}.$$

We can thus focus on $\epsilon_1(M_{(\ell,X,\widehat{r})}, [\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r}'})$. The following key lemma entails that this set is a regular union of intervals:

Lemma 4.4.28. *Let $M_{(\ell,X,\widehat{r})}$ be an atomic regular timed marking. Let $\widehat{r}$ and $\widehat{r}'$ be two elements of $\widehat{\mathbb{R}}_{\geq 0}$, and $\mathcal{L}\subseteq\mathcal{L}(\mathcal{A}_\tau)$ be a* regular *language. Then $\epsilon_1(M_{(\ell,X,\widehat{r})}, \mathcal{L}_{\widehat{r}}^{\widehat{r}'})$ can be defined using a regular union of intervals.*

Proof. The proof of this result is in two parts: we first express $\mathcal{L}_{\widehat{r}}^{\widehat{r}'}$ as the language of a finite automaton, and then—by a tedious proof—express $\epsilon_1(M_{(\ell,X,\widehat{r})}, \mathcal{L}_{\widehat{r}}^{\widehat{r}'})$ using regular unions of intervals.

Finite automaton for $\mathcal{L}_{\widehat{r}}^{\widehat{r}'}$. We assume that $\mathcal{L}$ is accepted by some automaton $\mathcal{B} = (\mathcal{L}, \Sigma, \ell_{init}, C, U, \mathcal{F})$ (in particular, $\mathcal{L}(\ell,\ell')$ would be obtained from $\mathcal{A}_\tau$ by imposing an initial state ℓ and a single accepting state ℓ'). In order to derive a finite automaton accepting $\mathcal{L}_{\widehat{r}}^{\widehat{r}'}$, we have to keep track of the value of ϵ_2 along runs. For this, we take the product of $\mathcal{B}$ with $\mathbb{J}_{\widehat{r}}$: we define the automaton $\mathcal{B}_{\widehat{r}}^{\widehat{r}'} = (\mathcal{L}\times\mathbb{J}_{\widehat{r}}, \widehat{\mathbb{R}}_{\geq 0}\times U\times\widehat{\mathbb{R}}_{\geq 0}, (\ell_{init},\widehat{r}), C, U', \mathcal{F}\times\{\widehat{r}'\})$ over the extended alphabet $\widehat{\mathbb{R}}_{\geq 0}\times U\times\widehat{\mathbb{R}}_{\geq 0}$ (this will be useful for technical reasons), where

$U' =$

$\{([\ell,\widehat{g}], \widehat{e}\cap\underline{e}, (\widehat{g}, e, \uparrow 0), \emptyset, [\ell', \uparrow 0]) \mid e = (\ell, \widehat{e}\cap\underline{e}, \emptyset, \tau, \ell') \in U \text{ and } \widehat{g}\cap\underline{e}\neq\emptyset\} \cup$

$\{([\ell,\widehat{g}], \widehat{e}\cap\underline{e}, (\widehat{g}, e, \widehat{g}'), C, [\ell', \widehat{g}']) \mid e = (\ell, \widehat{e}\cap\underline{e}, C, \tau, \ell') \in U \text{ and } \widehat{g}\cap\underline{e}\neq\emptyset \text{ and } \widehat{g}' = \widehat{g}\cap\widehat{e}\}.$

This automaton accepts words in $(\widehat{\mathbb{R}}_{\geq 0}\times U\times\widehat{\mathbb{R}}_{\geq 0})^*$. For any $\widehat{r}\in\widehat{\mathbb{R}}_{\geq 0}$, we define

$\kappa_{\widehat{r}}\colon U \to (\widehat{\mathbb{R}}_{\geq 0} \times U \times \widehat{\mathbb{R}}_{\geq 0})$ as

$$\kappa_{\widehat{r}}(e) = \begin{cases} (\widehat{r}, e, \widehat{r} \cap \widehat{e}) & \text{if } e \in U_{id} \\ (\widehat{r}, e, \uparrow 0) & \text{if } e \in U_0 \end{cases}$$

and extend this to U^* as $\kappa_{\widehat{r}}(e \cdot \pi) = \kappa_{\widehat{r}}(e) \cdot \kappa_{\widehat{r}'}(\pi)$ where $\widehat{r}'$ is such that $\kappa_{\widehat{r}}(e) = (\widehat{r}, e, \widehat{r}')$. Then:

Lemma 4.4.29. *For any $\widehat{r}$ and $\widehat{r}'$, we have $\kappa_{\widehat{r}}(\mathcal{L}(\mathcal{B})^{\widehat{r}'}_{\widehat{r}}) = \mathcal{L}(\mathcal{B}^{\widehat{r}'}_{\widehat{r}})$.*

Proof. Pick a finite word $\bar{\pi} = (\widehat{r}_k, e_k, \widehat{r}'_k)_{0 \leq k \leq n}$ in $(\widehat{\mathbb{R}}_{\geq 0} \times U \times \widehat{\mathbb{R}}_{\geq 0})^*$ accepted by $\mathcal{B}^{\widehat{r}'}_{\widehat{r}}$. By construction of $\mathcal{B}^{\widehat{r}'}_{\widehat{r}}$, it holds $\widehat{r}_0 = \widehat{r}$, and $\widehat{r}'_k = \kappa_{\widehat{r}_k}(e_k)$ for all k. Hence $\bar{\pi} \in \kappa_{\widehat{r}}(\mathcal{L}(\mathcal{B})^{\widehat{r}'}_{\widehat{r}})$.

Conversely, pick a partial path $\pi = (e_k)_{0 \leq k \leq n}$ in $\mathcal{L}(\mathcal{B})^{\widehat{r}'}_{\widehat{r}}$, and consider the word $\bar{\pi} = \kappa_{\widehat{r}}(\pi) = (\widehat{r}_k, e_k, \widehat{r}'_k)_{0 \leq k \leq n}$. Again by construction of $\mathcal{B}^{\widehat{r}'}_{\widehat{r}}$, any accepting run of $\mathcal{B}$ on π can be transformed into an accepting run of $\mathcal{B}^{\widehat{r}'}_{\widehat{r}}$ on $\bar{\pi}$, which entails our result. □

Writing $\mathcal{P}_U\colon \widehat{\mathbb{R}}_{\geq 0} \times U \times \widehat{\mathbb{R}}_{\geq 0} \to U$ for the projection on the second element of this alphabet (and extending it to sequences in the natural way), we get $\mathcal{P}_U(\mathcal{L}(\mathcal{B}^{\widehat{r}'}_{\widehat{r}})) = \mathcal{L}(\mathcal{B})^{\widehat{r}'}_{\widehat{r}}$.

Defining $\epsilon_1(M_{(\ell,X,\widehat{r})}, [\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}})$ as a regular union of intervals. We now focus on $\epsilon_1(M_{(\ell,X,\widehat{r})}, [\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}})$: this function maps ℓ' to $\epsilon_1((X,\widehat{r}), [\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}})$ (which we write η in the sequel for the sake of readability), and any other state to the empty timed set. To define η as a regular timed set, we progress in three steps: first, we decompose $[\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}}$ according to the presence of a reset in each sequence of transitions, then we quickly conclude for non-resetting sequences. Finally we discuss the resetting ones.

- **Separating resetting and non-resetting transitions.** For any $\widehat{g} \in \mathbb{J}_{\widehat{r}} = \{\uparrow 0, \widehat{r}\} \cup \widehat{G_{id}}$ and any $\underbrace{g'} \in \underbrace{G_{id}}$, we define

$$W^{id}_{\widehat{g},\underbrace{g'}} = \{\bar{\pi} = (\widehat{g}_1, e_1, \widehat{g}'_1) \cdots (\widehat{g}_n, e_n, \widehat{g}'_n) \mid \\ \widehat{g}'_n = \widehat{g} \text{ and } \min_{1 \leq k \leq n} \underbrace{e_k} = \underbrace{g'} \text{ and } e_k \in U_{id} \text{ for all } 1 \leq k \leq n\}.$$

  We decompose $[\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}}$ as the disjoint union of

$$\bigcup_{\underbrace{g'} \in \underbrace{G_{id}}} [\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}} \cap \mathcal{P}_U(W^{id}_{\widehat{r}',\underbrace{g'}})$$

(all runs in $[\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r'}}$ that do not contain any resetting transition) and

$$\bigcup_{\underbrace{g'}\in G_{\underbrace{id}}} \bigcup_{(\widehat{g},e)\in \mathbb{J}_{\widehat{r}}\times U_0} [\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r'}} \cap \mathcal{P}_U\left(W^{id}_{\widehat{g},\underbrace{g'}} \times \{(\widehat{g},e,\updownarrow 0)\} \times (\mathbb{J}_{\widehat{r}} \times U \times \mathbb{J}_{\widehat{r}})^*\right)$$

where the right-hand side of the intersection contains (the projection of) all paths containing at least one resetting transition labelled $(\widehat{g},e,\updownarrow 0)$. We write Z for the set $W^{id}_{\widehat{g},\underbrace{g'}} \times \{(\widehat{g},e,\updownarrow 0)\} \times (\mathbb{J}_{\widehat{r}} \times U \times \mathbb{J}_{\widehat{r}})^*$. Using the decomposition above, we get

$$\eta = \Big(\bigcup_{\underbrace{g'}\in G_{\underbrace{id}}} \bigcup_{\pi\in[\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r'}}\cap\mathcal{P}_U(W^{id}_{\widehat{r'},\underbrace{g'}})} \epsilon_1((X,\widehat{r}),\pi)\Big) \cup$$
$$\Big(\bigcup_{\underbrace{g'}\in G_{\underbrace{id}}} \bigcup_{(\widehat{g},e)\in \mathbb{J}_{\widehat{r}}\times U_0} \bigcup_{\pi\in[\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r'}}\cap\mathcal{P}_U(Z)} \epsilon_1((X,\widehat{r}),\pi)\Big).$$

- **Conclusion for non-resetting transitions.** In the first part, any path $\pi \in [\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r'}} \cap \mathcal{P}_U(W^{id}_{\widehat{r'},\underbrace{g'}})$ contains only non-resetting transitions, so that we have $\epsilon_1((X,\widehat{r}),\pi) = X \cap \underbrace{g'}$. Hence the first part is a finite union of regular intervals (because $M_{(\ell,X,\widehat{r})}$ is a regular timed marking). This can be represented by a regular union of intervals of the form $(I,\emptyset,0,0)$.

- **Discussion on the non-resetting transitions.** The non-resetting transitions are handled in the following way. Lemma 4.4.27 and the regularity of $\mathcal{L}(l,l')_{\widehat{r}}^{\widehat{r'}}$, that we obtained by constructing a finite timed automaton recognising it, are used to reformulate the expression enough to use Parikh's theorem. Then, using the new form arising from the theorem, a final discussion is conducted to separate the case where the closure uses a finite union of sets, and the case where there is an infinite (but regular) union.

  For the second part of η, *i.e.*

$$\Big(\bigcup_{\underbrace{g'}\in G_{\underbrace{id}}} \bigcup_{(\widehat{g},e)\in \mathbb{J}_{\widehat{r}}\times U_0} \bigcup_{\pi\in[\mathcal{L}(\ell,\ell')]_{\widehat{r}}^{\widehat{r'}}\cap\mathcal{P}_U(Z)} \epsilon_1((X,\widehat{r}),\pi)\Big)$$

  decomposing $\pi \in \mathcal{P}_U(Z)$ as $\pi_1 \cdot e \cdot \pi_2$ according to the above unions, we have

$$\epsilon_1((X,\widehat{r}),\pi) = \epsilon_1(\epsilon(\epsilon((X,\widehat{r}),\pi_1),e),\pi_2).$$

Since π_1 only contains non-resetting transitions, we have $\epsilon_1((X,\widehat{r}),\pi_1) = X \cap \underline{g'}$, and since $\pi_1 \in W^{id}_{\widehat{g},\underline{g'}}$ we have $\epsilon_2((X,\widehat{r}),\pi_1) = \widehat{g}$. Then, since e is a resetting transition, we get

$$\epsilon_1((X,\widehat{r}),\pi) = \epsilon_1(((X \cap \underline{g'}) \ltimes \Phi(\widehat{g},e), \updownarrow 0), \pi_2).$$

We then have

$$\bigcup_{\pi \in [\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}} \cap \mathcal{P}_U(Z)} \epsilon_1((X,\widehat{r}),\pi) = \epsilon_1\left(((X \cap \underline{g'}) \ltimes \Phi(\widehat{g},e), \updownarrow 0), \mathcal{Q}\right)$$

where $\mathcal{Q} = \mathcal{P}_U(W^{id}_{\widehat{g},\underline{g'}} \times \{(\widehat{g},e,\updownarrow 0)\}) \backslash [\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}}$ is the left-quotient of $[\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}}$ by $\mathcal{P}_U(W^{id}_{\widehat{g},\underline{g'}} \times \{(\widehat{g},e,\updownarrow 0)\})$, *i.e.* the set of all finite words $\beta \in U^*$ for which there exists a finite word α in $\mathcal{P}_U(W^{id}_{\widehat{g},\underline{g'}} \times \{(\widehat{g},e,\updownarrow 0)\})$ such that $\alpha \cdot \beta$ is in $[\mathcal{L}(\ell,\ell')]^{\widehat{r}'}_{\widehat{r}}$. Hence it remains to prove that for all $\underline{g'} \in \underline{G_{id}}$ and all $(\widehat{g},e) \in \mathbb{J}_{\widehat{r}} \times U_0$, the set

$$\eta' = \epsilon_1\left(((X \cap \underline{g'}) \ltimes \Phi(\widehat{g},e), \updownarrow 0), \mathcal{Q}\right),$$

is a regular union of intervals.

Write $X_{e,\widehat{g},\underline{g'}} = (X \cap \underline{g'}) \ltimes \Phi(\widehat{g},e) \subseteq \mathbb{R}_{\leq 0}$. From Lemma 4.4.27, we derive, for any $\pi_2 \in \mathcal{Q}$:

$$\epsilon_1((X_{e,\widehat{g},\underline{g'}}, \updownarrow 0), \pi_2) = X_{e,\widehat{g},\underline{g'}} - \sum_{J \in \mathbb{J}_{\updownarrow 0} \times U_0} \mathcal{J}_{\updownarrow 0}(\pi_2)(J) \times \Phi(J).$$

Hence

$$\begin{aligned}\eta' &= \bigcup_{\pi_2 \in \mathcal{Q}} X_{e,\widehat{g},\underline{g'}} - \sum_{J \in \mathbb{J}_{\updownarrow 0} \times U_0} \mathcal{J}_{\updownarrow 0}(\pi_2)(J) \times \Phi(J) \\ &= X_{e,\widehat{g},\underline{g'}} - \bigcup_{\pi_2 \in \mathcal{Q}} \sum_{J \in \mathbb{J}_{\updownarrow 0} \times U_0} \mathcal{J}_{\updownarrow 0}(\pi_2)(J) \times \Phi(J)\end{aligned}$$

For any $\pi_2 \in \mathcal{Q}$, we write $\mathbf{p}(\pi_2)$ for the Parikh vector of $\kappa_{\updownarrow 0}(\pi_2)$, *i.e.* the function mapping each letter λ of $\mathbb{J}_{\updownarrow 0} \times U \times \mathbb{J}_{\updownarrow 0}$ to the number of occurrences of λ along $\kappa_{\updownarrow 0}(\pi_2)$. We write $\mathbf{p}(\mathcal{Q})$ for the set of all such vectors. Then for all $(\widehat{n},e) \in \mathbb{J}_{\updownarrow 0} \times U_0$, we have

$\mathcal{J}_{\uparrow 0}(\pi_2)(\widehat{n}, e) = \sum_{\widehat{n}' \in \mathbb{J}_{\uparrow 0}} \mathbf{p}(\pi_2)(\widehat{n}, e, \widehat{n}')$. It follows:

$$\eta' = X_{e,\widehat{g},\underline{g}'} - \bigcup_{P \in \mathbf{p}(\mathcal{Q})} \sum_{\substack{(\widehat{n},e) \in \mathbb{J}_{\uparrow 0} \times U_0 \\ \widehat{n}' \in \mathbb{J}_{\uparrow 0}}} P(\widehat{n}, e, \widehat{n}') \times \Phi(\widehat{n}, e).$$

Now, the set $\mathcal{Q} = \mathcal{P}_U(W^{id}_{\widehat{g},\underline{g}'} \times \{(\widehat{g}, e, \uparrow 0)\}) \backslash [\mathcal{L}(\ell, \ell')]^{\widehat{r}'}_{\widehat{r}}$ is regular, so that by Parikh's theorem, $\mathbf{p}(Q)$ is a semi-linear set; it can be written $\mathbf{p}(Q) = \bigcup_{k=1}^{d} (\mathbf{p}_0^k + \sum_{k'=1}^{d_k} \mathbf{p}_{k'}^k \times \mathbb{N})$, where $\mathbf{p}_{k'}^k \in \mathbb{N}^{\mathbb{J}_{\uparrow 0} \times U_0 \times \mathbb{J}_{\uparrow 0}}$ for all $1 \leq k \leq d$ and $0 \leq k' \leq d_k$. Then:

$$\eta' = X_{e,\widehat{g},\underline{g}'} - \bigcup_{k=1}^{d} \Big(\sum_{\substack{(\widehat{n},e) \in \mathbb{J}_{\uparrow 0} \times U_0 \\ \widehat{n}' \in \mathbb{J}_{\uparrow 0}}} \mathbf{p}_0^k(\widehat{n}, e, \widehat{n}') \times \Phi(\widehat{n}, e) + \bigcup_{(\gamma_{k'})_{k'} \in \mathbb{N}^{d_k}} \sum_{k'=1}^{d_k} \sum_{\substack{(\widehat{n},e) \in \mathbb{J}_{\uparrow 0} \times U_0 \\ \widehat{n}' \in \mathbb{J}_{\uparrow 0}}} \gamma_{k'} \cdot \mathbf{p}_{k'}^k(\widehat{n}, e, \widehat{n}') \times \Phi(\widehat{n}, e) \Big).$$

We write $\Gamma_{e,\widehat{g},\underline{g}'}$ for the second term, so that $\eta' = X_{e,\widehat{g},\underline{g}'} - \Gamma_{e,\widehat{g},\underline{g}'}$. For each $1 \leq k \leq d$ and $0 \leq k' \leq d_k$, we define the interval

$$I_{k'}^k = \sum_{(\widehat{n},e) \in \mathbb{J}_{\uparrow 0} \times U_0} \sum_{\widehat{n}' \in \mathbb{J}_{\uparrow 0}} \mathbf{p}_{k'}^k(\widehat{n}, e, \widehat{n}') \times \Phi(\widehat{n}, e),$$

so that

$$\Gamma_{e,\widehat{g},\underline{g}'} = \bigcup_{k=1}^{d} \Big(I_0^k + \bigcup_{(\gamma_{k'})_{k'} \in \mathbb{N}^{d_k}} \sum_{k'=1}^{d_k} \gamma_{k'} \cdot I_{k'}^k \Big).$$

Notice that $I_{k'}^k \subseteq \mathbb{R}_{\geq 0}$ for all k' and k. We then consider two cases:

- first assume that for some k_0 and $k_0' \geq 1$ and some $(\widehat{n}_0, e_0) \in \mathbb{J}_{\uparrow 0} \times U_0$, it holds $\mathbf{p}_{k_0'}^{k_0}(\widehat{n}_0, e_0, \uparrow 0) > 0$ and $\Phi(\widehat{n}_0, e_0)$ has positive length. Consider the interval $L_\gamma = I_0^{k_0} + \gamma \cdot I_{k_0'}^{k_0}$ for any $\gamma \in \mathbb{N}$. Then clearly $L_\gamma \subseteq \Gamma_{e,\widehat{g},\underline{g}'}$ for all γ. Moreover, by definition of k_0 and k_0', the length of $I_{k_0'}^{k_0}$ is positive, so that the length of $\gamma \cdot I_{k_0'}^{k_0}$ tends to infinity with γ. It follows that for some $\alpha \in \mathbb{Q}_{\geq 0}$, $\uparrow \alpha \subseteq \bigcup_{\gamma \in \mathbb{N}} L_\gamma \subseteq \Gamma_{e,\widehat{g},\underline{g}'}$. Then $\Gamma_{e,\widehat{g},\underline{g}'} \cap \downarrow \alpha$ has finite granularity, so that it is a finite union of intervals of the form $I_{k'}^k \cap \downarrow \alpha$. It follows that $\Gamma_{e,\widehat{g},\underline{g}'}$ is a finite union of intervals, and can

hence be represented by a regular union of intervals.

Now, by hypothesis, X can be represented as a regular union of intervals. By Prop. 4.4.13, so can $X_{e,\widehat{g},\underline{g}'}$. Then by Prop. 4.3.4, $\eta' = X_{e,\widehat{g},\underline{g}'} - \Gamma_{e,\widehat{g},\underline{g}'}$ can be represented as a regular union of intervals.

– We now assume that for all k and $k' \geq 1$ and all $(\widehat{n}, e) \in \mathbb{J}_{\widehat{r}} \times U_0$, either $\mathbf{p}^k_{k'}(\widehat{n}, e, \updownarrow 0) = 0$, or $\Phi(\widehat{n}, e)$ has length 0. Then for all $1 \leq k \leq n$ and $1 \leq k' \leq d_k$, $I^k_{k'}$ contains a single element, which is a rational. Then $-\Gamma_{e,\widehat{g},\underline{g}'} = \{-\gamma \mid \gamma \in \Gamma_{e,\widehat{g},\underline{g}'}\}$ can be represented as a regular union of intervals. By Prop. 4.3.4, we get that η' can also be represented by a regular union of intervals in that case.

In the end, η is the union of two sets that can be represented as regular unions of intervals, thus by Prop. 4.3.4, it can also be represented as a regular union of intervals. □

4.5 Towards efficient diagnosis for n-clocks timed automata

In this section, we extend (part) of our formalism to n-clock timed automata. Precisely, we define a generalized gauge ($\ltimes$) operator, and prove that it can be used to compute an ϵ function. As for one-clock automata, we show that it is enough to consider *linear* timed markings, that ϵ handles.

However we do not propose an extension of regular unions of intervals and thus do not prove that we can furthermore restrict the study to regular timed markings.

We first extend the basic definitions in Section 4.5.1, and then use them to define a generalized $\ltimes$-operator and prove that it can be used to compute τ-closures for any number of clocks (Section 4.5.2).

4.5.1 Valuations for multiple clocks

In this subsection, we present the approach taken on n-clocks timed automata, and most importantly their valuations, and how the generic notations of this chapter are lifted from 1 to $n \in \mathbb{N}$ clocks.

Recall that a clock valuation is usually defined as a function $v \in \mathbb{R}^C_{\geq 0}$. As we use negative values to store *future* clock values, we sometimes consider functions to $\mathbb{R}$ instead.

To simplify furthermore the technical discussions, we fix an (arbitrary) order on the clocks, and hence allow ourselves to consider valuations as vectors in $\mathbb{R}^n$ (more heavily than in other chapters). Using this, we generalize the notions introduced in 4.2.1 for intervals to intersections of constraints on different dimensions. For a subset $\mathbb{K}$ of $\mathbb{R}$, we consider $\widehat{r} \in \widehat{\mathbb{K}}^n_{\geq 0}$ and $\underset{\sim}{r} \in \underset{\sim}{\mathbb{K}}^n_{\geq 0}$[3]. We define the sum of a subset X of $\mathbb{R}^n$ with an integer $t \in \mathbb{R}$ as the translation of the set on all dimensions: $X + t = \{(x_1 + t, .., x_n + t) \mid (x_1, .., x_n) \in X\}$. This corresponds to t time units elapsing. Using these definitions, we can extend atomic, finite and linear *timed sets* to n-clocks simply by considering sets and constraints in n dimensions. We note $\mathcal{T}(\mathbb{R}^n)$ the set of linear timed sets of $\mathbb{R}^n$. We extend the clock resets operations to sets elementwise, *i.e.* for $C' \subseteq C$, $X_{[C' \leftarrow 0]} == \{x_{[C' \leftarrow 0]} \mid x \in X\}$.

To discuss timed predecessors and successors of a set of clock valuations we note respectively $\mathsf{Pre}(X) = \bigcup_{t \in \mathbb{R}_{\geq 0}} (X - t)$ and $\mathsf{Post}(X) = \bigcup_{t \in \mathbb{R}_{\geq 0}} (X + t)$.

The notion of sequences of transitions and its associated notations does not depend on the number of clocks and can be directly reused. We extend markings to elements of $\mathbb{M} = (2^{\mathbb{R}^n_{\geq 0}})^{\mathcal{L}}$ and use similar definitions for $\mathbf{O}_a$ for an observable action $a \in \Sigma$:

$$\mathbf{O}_a(m) \colon \ell' \in \mathcal{L} \mapsto \{v' \in \mathbb{R}^n_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists v \in m(\ell).\ \exists e \in \mathsf{enab}((\ell, v)).\ \mathsf{act}(e) = a \wedge (\ell, v) \xrightarrow{0}_e (\ell', v')\}.$$

and $\mathbf{O}_t$:

$$\mathbf{O}_t(m) \colon \ell' \mapsto \{v' \in \mathbb{R}^n_{\geq 0} \mid \exists \ell \in \mathcal{L}.\ \exists v \in m(\ell).(\ell, v) \overset{t}{\rightsquigarrow}_\varepsilon (\ell', v')\}.$$

4.5.2 τ-closures

Using the previous definitions, we can extend timed markings to n dimensions and get back every definition and result of Section 4.4.1 by just considering valuations in $\mathbb{R}^n_{\geq 0}$ instead of $\mathbb{R}_{\geq 0}$. In this context, $\uparrow 0$ has to be read as the intersection of the corresponding one dimensional constraints on all dimensions. We hence focus here on how to compute the τ-closures using linear timed markings, as done in Section 4.4.2. For this, we first define and formalize the operator transforming a set of (potential) configurations by the effect of a guard and a set of clock resets.

3. Both open and closed intervals may appear in the same element on different dimensions

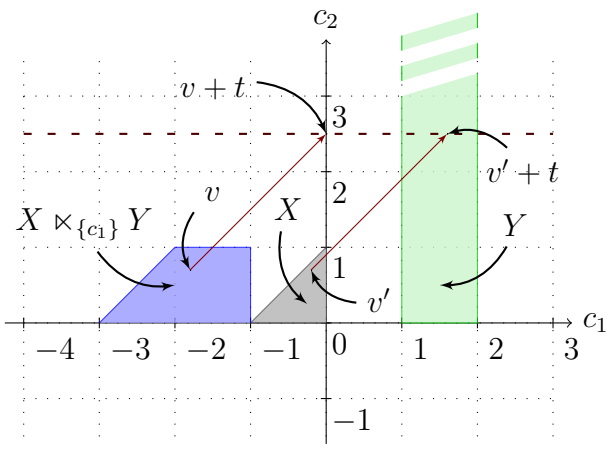


Figure 4.16: Effect of the gauge on c_1 for $X = -1 \leq c_1 \leq 0 \wedge c_2 \geq 0 \wedge c_2 - c_1 \leq 1$ and $Y = 1 \leq c_1 \leq 2$.

Definition 4.5.1. *For two sets X and Y (a guard) and $C' \subseteq C$, we let*

$$X \ltimes_{C'} Y = \{v \in \mathbb{R}^n \mid \exists t \geq 0,\ v \in ((X+t) \cap Y)_{[C' \leftarrow 0]} - t\} \tag{4.1}$$

Remark 4.5.2. *A valuation v in $X \ltimes_{C'} Y$ is an anticipation (*i.e. *a potential valuation corresponding to) an element of $((X+t) \cap Y)_{[C' \leftarrow 0]}$: the intersection of $f_(X, \uparrow 0)(t)$[4] with the guard Y. This can be better seen by rephrasing the definition as*

$$v + t \in ((X+t) \cap Y)_{[C' \leftarrow 0]}\ .$$

Hence $v \in X \ltimes_{C'} Y$ if and only if there exists $v' \in X$ and $t \in \mathbb{R}_{\geq 0}$ such that $v' + t \in Y$ and $v + t = v' + t_{[C' \leftarrow 0]}$ as can be seen in Figure 4.16.

We give some characterizations of $\ltimes_{C'}$, that can both help to represent the resulting set of valuations and obtain a zone-based computation.

Proposition 4.5.3. *For two sets X and Y (a guard) and $C' \subseteq C$,*

$$X \ltimes_{C'} Y = \{v \in \mathbb{R}^n \mid \exists t \geq 0,\ v \in (X \cap (Y-t))_{[C' \leftarrow 0]} - t \times \mathbb{1}_{C'}\} \tag{4.2}$$

with $\mathbb{1}_{C'}$ the vector with ones in the dimensions corresponding to clocks of C' and zeros

4. The filter is taken as $\uparrow 0$ because it is not part of the definition.

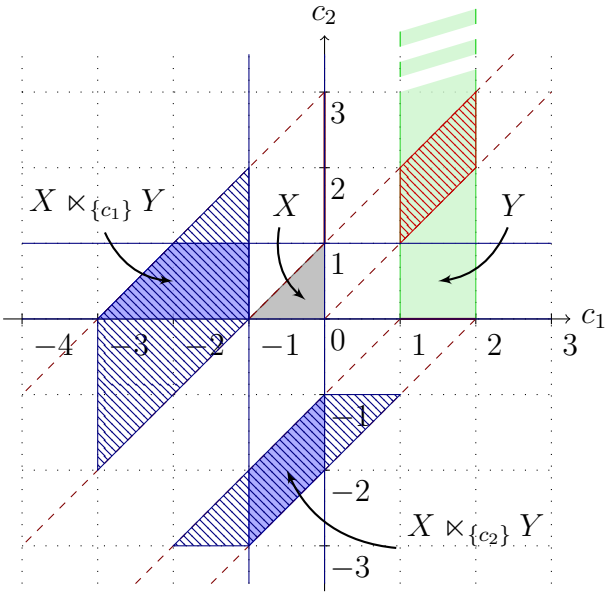


Figure 4.17: Effect of the gauge on individual clocks for $X = -1 \leq c_1 \leq 0 \wedge c_2 \geq 0 \wedge c_2 - c_1 \leq 1$ and $Y = 1 \leq c_1 \leq 2$.

otherwise. Similarly, letting $I = \{t \in \mathbb{R}_{\geq 0} \mid (X + t) \cap Y \neq \emptyset\}$*:*

$$X \ltimes_{C'} Y = \Big(((X + I) \cap Y)_{[C' \leftarrow 0]} - I\Big) \cap X_{[C' \leftarrow 0]^{-1}} \tag{4.3}$$

$$X \ltimes_{C'} Y = \Big((X \cap (Y - I))_{[C' \leftarrow 0]} - I \times \mathbb{1}_{C'}\Big) \cap X_{[C' \leftarrow 0]^{-1}} \tag{4.4}$$

with $X_{[C' \leftarrow 0]^{-1}}$ *the zone obtained from* X *by suppressing all constraints (guard and diagonals) including clocks in* C'*.*

The characterization 4.2 can be used to more easily handle examples of $\ltimes_{C'}$: it behaves as a filter on X where Y is sent back by an increasing delay $-t$ for $t \in \mathbb{R}_{\geq 0}$ and the projection on $C \setminus C'$ is used on the sub-vector space $-t \times \mathbb{1}_{C'}$.

The characterization 4.3 allows to express the gauge using only operations on sets (of delays and valuations). Its variant 4.4 proposes the same for 4.2.

Example 4.5.4. *Consider a transition with guard* $Y = 1 \leq c_1 \leq 2$ *in a timed automaton over the two clocks* $C = \{c_1, c_2\}$*. We take the set of potential valuations* $X = -1 \leq c_1 \leq 0 \wedge c_2 \geq 0 \wedge c_2 - c_1 \leq 1$*. Both* X *and* Y *are depicted in Figure 4.17, together with* $X \ltimes_{\{c_1\}} Y$ *and* $X \ltimes_{\{c_2\}} Y$ *and the constructions necessary to obtain them through characterization 4.3. Notice that in this case,* $I = [1, 3]$*.*

Notice how by adding delays $t \in \mathbb{R}$ to these sets we can see that the gauge corresponds to the resets of clocks for valuations in $X + t \cap Y$.

Proof. The characterization 4.2 is simply obtained from the definition 4.1 by manipulating the $+t$ and considering valuations and zones as vectors in $\mathbb{R}^n$. 4.4 is obtained similarly from 4.3.

Hence, it remains to prove 4.3. The core of the work is to show that the two occurrences of t in 4.1 can be made independent by adding the constraint $v \in X_{[C' \leftarrow 0]^{-1}}$. First, we can see that

$$X \ltimes_{C'} Y \subseteq \Big(((X + I) \cap Y)_{[C' \leftarrow 0]} - I\Big)$$

as by definition $t \in I$. Furthermore, $X \ltimes_{C'} Y \subseteq X_{[C' \leftarrow 0]^{-1}}$ as by definition of the gauge for $v \in X \ltimes_{C'} Y$ and $c \in C \setminus C'$, there exists $v' \in X$ such that $v(c) = v'(c)$ (as argued in Remark 4.5.2). We now show that

$$\Big(((X + I) \cap Y)_{[C' \leftarrow 0]} - I\Big) \cap X_{[C' \leftarrow 0]^{-1}} \subseteq X \ltimes_{C'} Y \ .$$

For this consider $v \in \Big(((X + I) \cap Y)_{[C' \leftarrow 0]} - I\Big) \cap X_{[C' \leftarrow 0]^{-1}}$. By definition, there exists $t_2 \in I$ such that $v + t_2 \in ((X + I) \cap Y)_{[C' \leftarrow 0]}$. We note $\mathcal{K} = ((X + I) \cap Y) \cap (v + t_2)_{[C' \leftarrow 0]^{-1}}$. By definition of t_2, $\mathcal{K}$ is not empty. Now, if we can take a valuation $v' \in (\mathcal{K} - t_2) \cap X$ we will have that $v' + t_2 \in (X + t_2) \cap Y$ and $v = (v' + t_2)_{[C' \leftarrow 0]} - t_2$. Thus we will have proven that $v \in X \ltimes_{C'} Y$.

It remains to prove that $(\mathcal{K} - t_2) \cap X$ is not empty. For this, we translate the problem and prove that $\mathcal{K} \cap (X + t_2)$ is not empty.

The rest of the proof is geometric and relies heavily on the structure of zones and guards. It constructs an element of $\mathcal{K} \cap (X + t_2)$. We note $\mathcal{P} = Y \cap (v + t_2)_{[C' \leftarrow 0]^{-1}}$ and rewrite $\mathcal{K} \cap (X + t_2)$ as the intersection of $\mathcal{P}$, $(X + t_2) \cap (v + t_2)_{[C' \leftarrow 0]^{-1}}$ and $Y \cap (X + t_2)$. By definition of $t_2 \in I$ we know that these three sets are not empty. We will use this to construct an element of $\mathcal{K} \cap (X + t_2)$.

By 4.3 there is $t_1 \in \mathbb{R}_{\geq 0}$ such that $(\mathcal{P} - t_1) \cap X \neq \emptyset$. Consider $p_1 \in (\mathcal{P} - t_1 + t_2) \cap (X + t_2) \neq \emptyset$, $q \in (X + t_2) \cap (v + t_2)_{[C' \leftarrow 0]^{-1}}$ and $s \in Y \cap (X + t_2)$. We can construct $p_2 = p_1 + t_2 - t_2$. We then have that $p_2 \in \mathcal{P}$ and respects all diagonal constraints of $X + t_2$ (as $p_1 \in X + t_2$ and diagonal constraints are invariant by time elapsing).

Finally, we construct p_3 in the following way:

$$\forall c \in C,\ p_3(c) = \begin{cases} p_2(c) \text{ if } c \notin C' \\ s(c) \text{ if } c \in C' \ . \end{cases}$$

We now prove that $p_3 \in (\mathcal{K} - t_2) \cap X$:

- p_3 is in Y. Indeed Y being a guard it suffices to verify that for each clock c, $p_3(c)$ satisfies the guard (*i.e.* non-diagonal) constraints of Y. This is ensured by definition of p_3 as $p_2, s \in Y$;
- p_3 is in $v + t_{2[C' \leftarrow 0]^{-1}}$. Indeed $p_2 \in \mathcal{P} \subseteq v + t_{2[C' \leftarrow 0]^{-1}}$ and the projections on $C \setminus C'$ of p_2 and p_3 are equal;
- p_3 is in $X + t_2$. We prove this by showing that it satisfies the different constraints;
  - p_3 is on the segment between p_2 and s, which both satisfy the diagonal constraints of $X + t_2$. As these constraints are convex, p_3 satisfies them too;
  - for all guard constraints on $c \in C'$, $s \in (X + t_2)$ and $p_3(c) = s(c)$ ensures the satisfaction of the constraint;
  - similarly for all guard constraints on $c \notin C'$, $p_3(c) = v + t_2(c) = q(c)$ and $q \in (X + t_2)$.

  Hence $p_3 \in X + t_2$ as it satisfies all its constraints.

We thus have our result, as p_3 is a witness that $v \in X \ltimes_{C'} Y$. □

Remark 4.5.5. *There is no delay-based characterization in the n-clock case as delays are not of the same type than valuations anymore - by opposition to the* 1 *clock case.*

This operator generalizes the operator we defined for the one-clock case: indeed, for $C = \{c\}$:

$$\begin{aligned} X \ltimes_C Y &= \{v \in \mathbb{R} \mid \exists t \geq 0,\ v \in ((X + t) \cap Y)_{[\{c\} \leftarrow 0]} - t\} \\ &= \{v \in \mathbb{R} \mid \exists t \geq 0,\ ((X + t) \cap Y)_{[\{c\} \leftarrow 0]} \neq \emptyset \ \wedge\ v \in \{0\} - t\} \\ &= \{-t \mid t \in \mathbb{R}_{\geq 0} \ \wedge\ ((X + t) \cap Y) \neq \emptyset\} \end{aligned}$$

which is equal to $X \ltimes Y$ (the one-clock operator) by Prop. 4.4.10.

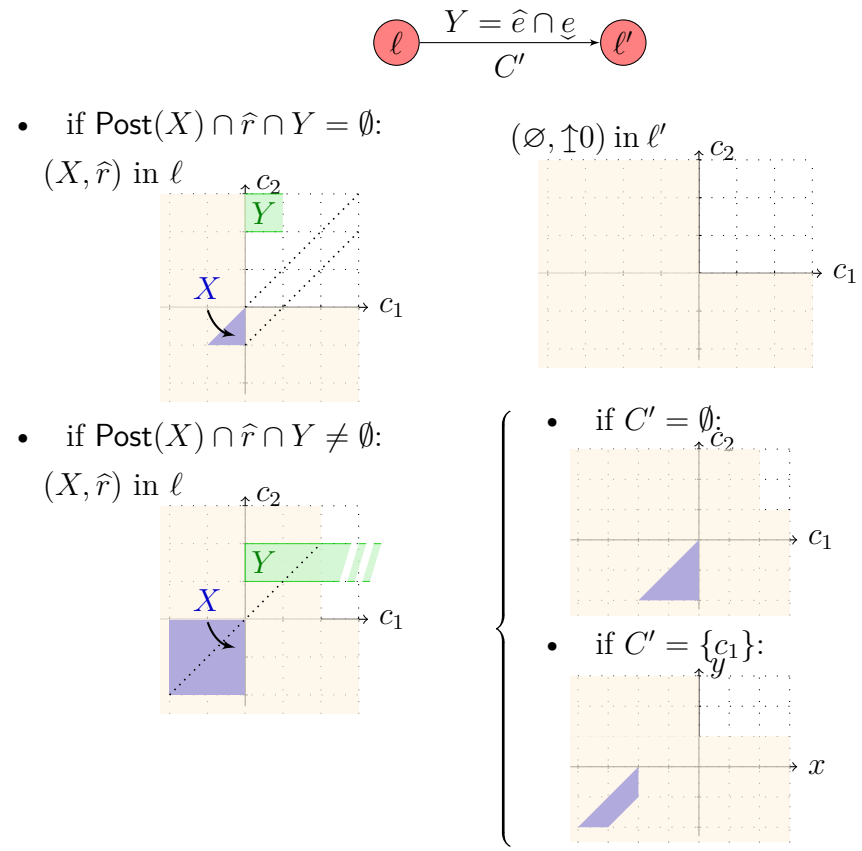


Figure 4.18: Representation of the effect of silent transition $(\ell, Y, \tau, C', \ell')$ in three cases.

Thanks to this operator, we can define the effect of a transition on an atomic timed set:

Definition 4.5.6. *For an atomic timed set $T = (X, \widehat{r})$ of $\mathcal{T}(\mathbb{R}^n)$ and a transition $e = (\ell, g = \widehat{e} \cap \underbrace{e}, \tau, C', \ell')$ of U:*

$$\epsilon((X, \widehat{r}); e) = \begin{cases} (\emptyset; \uparrow 0) & \text{if } \mathsf{Post}(X) \cap \widehat{r} \cap g = \emptyset \\ (X \ltimes_{C'} (\widehat{r} \cap g); (\widehat{r} \cap \widehat{e})_{[C' \leftarrow 0]}) & \\ & \text{if } \mathsf{Post}(X) \cap \widehat{r} \cap g \neq \emptyset. \end{cases}$$

Remark 4.5.7. *We can separate the case $C' = \emptyset$ in the expression of ϵ to obtain $X \cap \mathsf{Pre}(\widehat{r} \cap g) = X \ltimes_\emptyset (\widehat{r} \cap g)$. As the computation is then far less involved it can be interesting to separate this case in an implementation.*

One can notice that for the 1-clock case $C' = \emptyset$ and $C' = C$ were separated and no general expression was given. The reason is that both cases can be expressed with far

simpler expressions than the general case (which corresponds to $)\ltimes_{C'}$*.*

Remark 4.5.8. *Unlike in the* 1*-clock case, we do not separate the guard* g *between upper and lower constraints (*$\widehat{e}$ *and* $\underset{\smile}{e}$*). Indeed, the interactions between the upper and lower ones induce* diagonal constraints *that would not appear with only the upper (resp. lower) ones and are necessary to correctly model the behaviours.*

The definition of ϵ is illustrated in Figure 4.18 for a two-clock automaton ($\widehat{r}$ is represented by the non-grayed-out areas).

Definition 4.5.6 (cont.)**.** *As in the one-clock case, we extend* ϵ *to sequences of transitions inductively by letting* $\epsilon((X;\widehat{r}),\varepsilon) = (X;\widehat{r})$ *and, for* $\pi \in U^+$*,*

$$\epsilon((X;\widehat{r}),\pi\cdot e) = \begin{cases} \epsilon(\epsilon((X;\widehat{r}),\pi),e) & \text{if } \mathsf{tgt}(\pi) = \mathsf{src}(e) \\ (\emptyset;\uparrow 0) & \text{otherwise.} \end{cases}$$

Finally we extend this definition to linear timed sets by letting

$$\epsilon(\{T_k \mid k\in K\},\pi) = \{\epsilon(T_k,\pi) \mid k\in K\}.$$

Using these definitions, we can prove that ϵ corresponds to the effect of a sequence of transitions from a timed set. The following lemma extends Lemma 4.4.17. Notice that the proof has to be adapted, not only to the new gauge definition but also to the diagonal constraints arising in timed automata with multiple clocks.

Lemma 4.5.9. *Let* T *be a linear timed set and* $\pi \in U^*$*. Then for any* $t \in \mathbb{R}_{\geq 0}$ *and any* $v \in \mathbb{R}^n_{\geq 0}$*,*

$$v \in f_{\epsilon(T,\pi)}(t) \;\Leftrightarrow\; \exists t_0 \in [0;t].\ \exists v' \in f_T(t_0).(\mathsf{src}(\pi),v') \xrightarrow{t-t_0}_{\pi} (\mathsf{tgt}(\pi),v).$$

Proof. We carry the proof for the case where $T = (X;\widehat{r})$ is an atomic timed set. The extension to unions of atomic timed sets is straightforward. We begin with the case where π is a single transition $e = (\ell, Y = \widehat{e}\cap\underset{\smile}{e},\tau,C',\ell')$. In case X is empty, then also $f_{\epsilon(T,e)}(t)$ is empty, and $\forall t_0, f_T(t_0) = \emptyset$, so the result holds. We now assume that X is not empty, and consider three cases:

- if $\mathsf{Post}(X)\cap\widehat{r}\cap Y = \emptyset$, then $\epsilon(T,e) = (\emptyset;\uparrow 0)$. On the other hand, for any t_0 and any $v' \in f_T(t_0)$, it holds $v' \in \mathsf{Post}(X)\cap\widehat{r}$, so that $v' \notin Y$, and the transition cannot

be taken from that valuation. Hence both sides of the equivalence evaluate to false, and the equivalence holds.

- now assume that $\mathsf{Post}(X) \cap \widehat{r} \cap Y \neq \emptyset$, and consider the case where C' is empty. We proceed by double inclusion. We first take a valuation $v \in f_{\epsilon(T,\pi)}(t)$ and exhibit a predecessor of this configuration in $f_T(t_0) \cap Y$ for a t_0 to be constructed.

  $v \in f_{\epsilon(T,\pi)}(t)$ means that $v \in (X \cap \mathsf{Pre}(\widehat{r} \cap Y)) + t \cap \widehat{r} \cap \widehat{e}$. Equivalently, $v \in (X+t) \cap \mathsf{Pre}(\widehat{r}+t \cap Y+t) \cap \widehat{r} \cap \widehat{e}$. As $\widehat{e}$ encodes the lower bounds on individual clocks of Y and $\mathsf{Pre}(\widehat{r} + t \cap Y + t)$ keeps all diagonal constraints of $\widehat{r} \cap Y$, we have that $v \in (X+t) \cap \mathsf{Pre}(\widehat{r}+t \cap Y+t) \cap \mathsf{Post}(\widehat{r} \cap Y)$. Hence $\exists t' \in [0,t],\ v-t' \in X+t-t' \cap \widehat{r} \cap Y$. By taking $t_0 = t-t'$ and $v' = v-t+t_0 = v-t'$ we can write $\exists t_0 \in [0,t], \exists v' \in f_T(t_0) \cap \widehat{r} \cap Y$. The first inclusion then holds by definition of π.

  Conversely, if $t_0 \in [0;t]$ and $v' \in f_T(t_0)$ exist such that $(\mathsf{src}(\pi), v') \xrightarrow{t-t_0}_{\pi} (\mathsf{tgt}(\pi), v)$, then $v' \in X + t_0 \cap \widehat{r}$ and as there is no reset on π, $v = v' + t - t_0$, $v \in X + t \cap \widehat{r} + t - t_0 \subseteq X + t \cap \widehat{r}$. Furthermore, $\exists t_1 \leq t - t_0$ such that $v' + t_1 \in Y$. More precisely, $v' + t_1 \in \widehat{r} \cap Y$ as $v' \in \widehat{r}$ and this set has no upper bound. As $v = v' + t - t_0$, we have $v \in \mathsf{Pre}(\widehat{r} \cap Y) + t$ and $v \in \widehat{e}$ as $t_1 \leq t - t_0$. By aggregating the constraints, we obtain $v \in (X + t) \cap \mathsf{Pre}((\widehat{r} \cap Y \cap) + t) \cap \widehat{r} \cap \widehat{e}$ *i.e.* the left-hand side of the equivalence.

- Consider $\mathsf{Post}(X) \cap \widehat{r} \cap Y \neq \emptyset$ and $C' \neq \emptyset$. In this case $v \in f_{\epsilon(T,\pi)}(t)$ means that $\exists t_2 \geq 0,\ v \in (\widehat{r} \cap Y \cap (X + t_2))_{[C' \leftarrow 0]} - t_2 + t$. Notice that as $v \in \mathbb{R}^n_{\geq 0}$, it comes that $t - t_2 \geq 0$. Let $v'_{[C' \leftarrow 0]} = v + t_2 - t \in (f_T(t_2) \cap Y)_{[C' \leftarrow 0]}$ It comes that we can construct $v' \in f_T(t_2) \cap Y$ from $v'_{[C' \leftarrow 0]}$[5]. By taking $t_0 = t_2$ we have that $0 \leq t_0 \leq t$, $v' \in f_T(t_0)$ such that $v' \xrightarrow{t-t_0}_{\pi} v' + t - t_0 = v$.

  Conversely if for a given $0 \leq t_0 \leq t$ and a given $v' \in f_T(t_0)$, we have that $(\mathsf{src}(\pi), v') \xrightarrow{t-t_0}_{\pi} (\mathsf{tgt}(\pi), v)$, then there exists $0 \leq t_1 \leq t - t_0$ such that $v' + t_1 \in Y$ and the transition is taken from v'. By definition of f_T, $v' + t_1 \in f_T(t_0 + t_1) \cap Y$. We have that $v = (v' + t_1)_{[C' \leftarrow 0]} + t - t_0 - t_1 \in (f_T(t_0 + t_1) \cap Y)_{[C' \leftarrow 0]} + t - t_0 - t_1$. Hence by taking $t_2 = t_0 + t_1$ we ensure that $v \in (X \ltimes_{C'} Y) + t$. Furthermore, $t - t_0 - t_1 \geq 0$ as $t_1 \leq t - t_0$ thus $\forall c \in C',\ v(c) \geq 0$, and $v_{|C'} \in (\widehat{r} \cap \widehat{e})_{|C'}$ as $v_{|C'} \in \mathsf{Post}(v' + t_1)_{|C'}$ $(t_1 \leq t - t_0)$ and $v' + t_1 \in f_T(t_0 + t_1) \cap Y \subseteq \widehat{r} \cap Y \subseteq \widehat{r} \cap \widehat{e}$. This proves the left part of the equivalence.

5. Possibly several such v' exists, we consider one of them.

We now extend this result to sequences of transitions. The case where $\pi = \varepsilon$ is straightforward. Now assume that the result holds for some word π, and consider a word $\pi \cdot e$. In case $\mathsf{tgt}(\pi) \neq \mathsf{src}(e)$, the result is trivial.

The case of single transitions has been handled just above. We thus consider the case of $\pi \cdot e$ with $\pi \in U^+$. First assume that $v' \in f_{\epsilon(T,\pi\cdot e)}(t)$, and let $T' = \epsilon(T,\pi)$. Then $v' \in f_{\epsilon(T',e)}(t)$, thus there exist $0 \leq t_0 \leq t$ and $v \in f_{T'}(t_0)$ s.t. $(\mathsf{src}(e), v) \xrightarrow{t-t_0}_e (\mathsf{tgt}(e), v')$. Since $v \in f_{T'}(t_0)$, there must exist $0 \leq t_1 \leq t_0$ and $v'' \in f_T(t_1)$ such that $(\mathsf{src}(\pi), v'') \xrightarrow{t_0-t_1}_\pi (\mathsf{tgt}(\pi), v)$. This way, we have found a value t_1 with $0 \leq t_1 \leq t$ such that $(\mathsf{src}(\pi), v'') \xrightarrow{t-t_1}_{\pi\cdot e} (\mathsf{tgt}(e), v')$.

Conversely, if $(\mathsf{src}(\pi), v'') \xrightarrow{t-t_1}_{\pi\cdot e} (\mathsf{tgt}(e), v')$ for some $0 \leq t_1 \leq t$ and $v'' \in f_T(t_1)$, then we have $(\mathsf{src}(\pi), v'') \xrightarrow{t_0-t_1}_\pi (\mathsf{tgt}(\pi), v) \xrightarrow{t-t_0}_e (\mathsf{tgt}(e), v')$ for some $t_1 \leq t_0 \leq t$ and some v. We prove $v \in f_{\epsilon(T,\pi)}(t_0)$: indeed, we have $0 \leq t_1 \leq t_0]$, and $v'' \in f_T(t_1)$ such that $(\mathsf{src}(\pi), v'') \xrightarrow{t_0-t_1}_\pi (\mathsf{tgt}(\pi), v)$, which by induction hypothesis entails $v \in f_{\epsilon(T,\pi)}(t_0)$. Thus we have $0 \leq t_0 \leq t$ and $v \in f_{T'}(t_0)$, where $T' = \epsilon(T,\pi)$, such that $(\mathsf{tgt}(\pi), v) \xrightarrow{t-t_0}_e (\mathsf{tgt}(e), v')$, which means $v' \in f_{\epsilon(T',e)}(t)$, and concludes the proof. □

As in the one-clock case, this semantic characterization of $\epsilon(T,\pi)$ entails the following, ensuring that the result of a computation by ϵ does not depend on the representation.

Corollary 4.5.10. *For any sequence π of transitions, and any two equivalent linear timed sets T and T', the linear timed sets $\epsilon(T,\pi)$ and $\epsilon(T',\pi)$ are equivalent.*

We can once more extend ϵ to linear timed markings in the same way and generalize Theorem 4.4.19 to n clocks. The proof is the same for $v \in \mathbb{R}^n_{\geq 0}$, using Lemma 4.5.9.

By considering the extensions of ϵ to languages, we immediately get:

Corollary 4.5.11. *For any linear timed marking M, it holds $\epsilon(M) \equiv M^\tau$.*

This proves that our ϵ operator (and the underlying gauge) is a correct representation of the semantic of a (silent) timed automaton, and that linear timed markings are sufficient to represent the semantics of timed automata.

Example 4.5.12. *Consider the timed automaton with two clocks depicted in Figure 4.19 (the τ labels are omitted). The construction with ℓ_0, ℓ_1 and ℓ_2 allows to obtain the atomic timed set $(c_1 \in [-1,0] \wedge c_2 \in [-1,0]; c_1, c_2 \geq 0)$ as input for ℓ_3 [6]. The closure of ℓ_3 is depicted on the side with the two guards generating it. It is composed of several infinite*

6. We slightly abuse the construction to take the union of sets for two timed sets sharing the same filter.

behaviours. Notice that there is some regularity in the structure, but that it can be complex: several nested infinite repetitions appear.

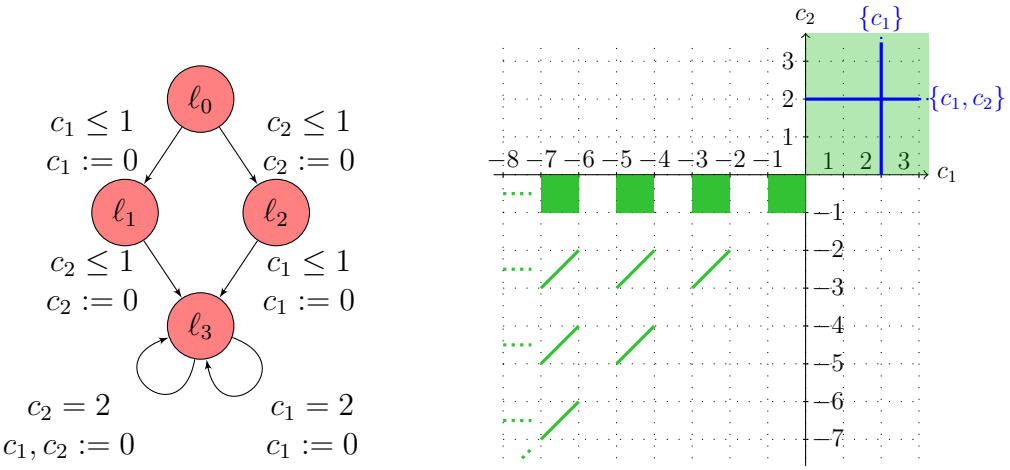


Figure 4.19: A simple timed automaton and its associated timed set.

4.5.3 Stability of a representable class

We do not define regular timed sets and markings for multiple clocks, but in the following, we prove that finite timed markings are enough to represent the closure.

Proposition 4.5.13. *Consider a timed automaton $\mathcal{A}$ and a regular language $\mathcal{L} \subseteq \mathcal{L}_E(\mathcal{A})$ included in the language of transitions. Given a finite timed marking M, $\epsilon(M, \mathcal{L})$ is a finite timed marking.*

Proof. Let M be a finite timed marking: then M can be written as the finite union of atomic timed markings $M_{(\ell,X,\widehat{r})}$, defined as $M_{(\ell,X,\widehat{r})}(\ell) = f_{(X;\widehat{r})}$ and $M_{(\ell,X,\widehat{r})}(\ell') = f_{(\emptyset;\mathbb{1}0)}$ for all $\ell' \neq \ell$.

In the end, any regular timed marking M can be written as the finite union $\bigsqcup_{k\in K} M_k$ of atomic timed markings. Thus we have $\epsilon(M) \equiv \bigsqcup_{k\in K} \epsilon(M_k)$ and it is enough to prove the property for atomic timed markings.

We write $\epsilon_1((X;\widehat{r}),\pi)$ and $\epsilon_2((X;\widehat{r}),\pi)$ for the first and second components of $\epsilon((X;\widehat{r}),\pi)$, and use the same generalization to timed markings than in the one-clock case.

Note that:

- $\epsilon_2((X;\widehat{r}),\pi) = \widehat{r} \cap \bigcap_{k<n} \widehat{e}_k$ if $\pi = e_1 \dots e_n \in {U_{id}}^*$ is a sequence of consecutive non-resetting transitions.

- $\epsilon_2((X;\widehat{r}),\pi\cdot e) = (\epsilon_2((X;\widehat{r}),\pi)\cap \widehat{e})_{[C'\leftarrow 0]}$ if e is a resetting transition for C'.

We define the set of upper bounds of guards appearing in U as

$$\mathcal{G} = \{\widehat{e} \mid \exists\ e = (\ell, Y = \widehat{e} \wedge \underbrace{e}, C', \epsilon, \ell') \in U\}$$

and using this set $\mathbb{J}_{\widehat{r}} = \{\widehat{g} \in \mathbb{R}^n_{\geq 0} \mid \forall i \in [1,n],\ \exists\ \widehat{e} \in \mathcal{G} \cup \{\widehat{r}, \uparrow 0\}, \widehat{e}_i = \widehat{g}_i\}$. It is easy to see that $\mathbb{J}_{\widehat{r}}$ is a finite set and $\epsilon_2((X;\widehat{r}),\pi) \in \mathbb{J}_{\widehat{r}}$. Hence a finite number of atomic timed sets is sufficient to describe the closure of a finite timed marking, *i.e.* the closure is a finite timed set. Indeed, there is a finite number of filters $\widehat{r}$ in the representation of M (one per M_k), no more than $|\mathbb{J}_{\widehat{r}}|$ different atomic timed sets are needed per location of the automaton for each $\widehat{r}$, and there is a finite number of locations. □

4.6 Conclusion and future works

In this chapter, we presented a novel approach to solve the state estimation problem, and in particular to efficiently compute the τ-closure in the case of partially observable one-clock timed automata; it builds on a kind of powerset construction for automata over timed domains, using our new formalism of *linear timed sets* to represent the evolution of the set of reachable configurations of the automaton. We prove that the semantics - and in particular the τ-closure - of a one-clock timed automaton can be computed using *regular* timed sets, a finitely representable subclass of linear timed sets. We furthermore show that the full class of regular timed sets is needed to represent general one-clock timed automata.

We extend the basis of our approach to timed automata with *multiple* clocks, and prove that finite timed sets are again enough to compute the τ-closure.

An implementation demonstrating the approach for 1-clock automata, in the case of diagnosis, and comparing it to the technique proposed in [Tri02] has been realized. It is not reproduced in this thesis as the author did not take part in its creation, but can be found at `http://people.irisa.fr/Nicolas.Markey/download/DOTA.zip`. It is furthermore commented in [Bou+21].

A natural extension of these results is to introduce a (representable) notion of regularity for multiple clocks automata and to prove that it is large enough to represent the semantics and τ-closure.

This is by no means immediate, as first the presence of multiple clocks allows involved behaviours, and second (and most importantly), the proofs proposed in the one-clock case rely on the structure of the $\ltimes$ operator, and the ability to simply "count" resetting transitions (*i.e.* the order of transitions only slightly matters). This is not true for multiple clocks, where the orders of the resets is central to the dynamics.

Another possible direction of research could target priced timed automata, with the aim of monitoring the cost of the execution in the worst case.

Chapter 5

Active learning

Never laugh at live dragons, Bilbo you fool!

— J.R.R. Tolkien "The Hobbit"

In this chapter, we focus on model learning of a new sub-class of timed automata: reset-optional event recording automata. Fundamentally, learning corresponds to the construction of the inverse functions trace^{-1} and sig^{-1} that would allow to deduce a run (of a given model) from an observation.

The work displayed in this chapter focuses on the inference, representation and management of the information necessary to the construction of the inverse functions, *i.e.* to find and represent the semantics of a timed language under a form fitting a timed automaton.

5.1 Introduction

As discussed in Section 2.3.2, active learning is a fitting learning framework for timed automata, as it can overcome passive learning expressivity limitations. For this reason, some contributions have already been made in this domain, although limited to somewhat simple models: automata with one clock displaying low-dimensional behaviours and event-recording automata where clock resets can be directly inferred from observations.

Furthermore, by its interacting nature, active learning is a method of interest to ground a general theoretical framework for a learning-acting loop between (timed) models and reality.

Remark 5.1.1. *It is interesting to note that* reinforcement learning*, a kind of learning stemming from the artificial-intelligence community, would also be a good candidate for such loop. Yet, reinforcement learning theory usually relies on stochastic models and have been less developed (to the author's knowledge) for timed automata, potentially due to the complex nature of probabilistic timed automata [SB18].*

In this chapter, we propose to generalize active learning to a class of timed automata enjoying both several clocks and different possible resets that cannot be inferred directly from observations. This allows us to design and prove algorithms that handle all the main difficulties that arise in deterministic TAs, making this contribution an important first step towards active learning for generic deterministic TAs.

To our knowledge, the closest works to ours are Grinchtein's thesis on active learning of DERA [Gri08], the TL^* algorithm [Lin+11], that proposes the most efficient algorithm to learn DERAs using observation tables, and the paper proposing to learn one clock TAs [An+20]. The work of Grinchtein *et al.* [GJP06] is the most related to ours, as we use some of the data structures they developed and keep the general approach based on timed decision trees. The main difference between our work and this one is that we handle the inference of resets in a class of models in which they cannot be *directly* deduced from observations. The authors of this work have later proposed TL^*_s, a completely different and more efficient algorithm to learn DERAs. This approach relies on the region-graph construction, which uses the knowledge of clock values to minimize the number of queries. As clock values depend on clock resets, we could not use this argument and build upon the method proposed in TL^*_s. The approach reported in [An+20] proposes to deal with reset guessing, but makes it in a somewhat "brute force" manner, by directly applying a branch-and-bound algorithm and jumping from model to model. In order to be able to deal with larger dimensions, *e.g.* to handle TAs with more than one clock, we need to be more efficient by exploiting the theory built around TAs and detecting invalid models as early as possible. Finally, TL^* [Lin+11] is thematically very close to our work (both because it learns DERAs and enhances upon Grinchtein's thesis). Yet it is a different technical stem, that uses specificities of the ERAs to optimize the learning method in a fashion similar to TL^*_s, while our approach goes beyond ERAs. It is furthermore based on a table data-structure, while our work reasons on trees.

5.2 Preliminaries

5.2.1 Timed automata

For the rest of this chapter, we fix a finite alphabet Σ of actions. We are mostly interested in the language generative approach of timed automata. Because of this we consider timed automata without invariants and equipped with a subset of accepting locations that will

be used to define their language.

As a consequence, in this chapter we define timed automata as $\mathcal{A} = (\mathcal{L}, \ell_{init}, C, E, \mathsf{Accept})$ without precising the alphabet Σ, the invariant function $\mathcal{I} : \mathcal{L} \to \{\mathsf{true}\}$ but adding the accepting subset $\mathsf{Accept} \subseteq \mathcal{L}$. We define the set of accepting configurations as $\mathsf{Accept}_{\mathcal{T}} = \mathsf{Accept} \times \mathbb{R}^C_{\geq 0}$.

We say that a path (resp. run) is *accepting* when the target of the last transition is in Accept (resp. in $\mathsf{Accept}_{\mathcal{T}}$).

As this chapter focuses on languages and not controllability, we consider only *finite alternating* runs, signatures and traces *ending in* Σ (as final delays are by nature unobservable in this setting). As mentioned in Section 1.2 (Figure 1.4), we will refer to signatures as *timed words with resets* and traces as *timed words*. Furthermore, for such words one can group delays and discrete transitions. We will thus represent timed words with resets as $w_{tr} = (t_i, a_i, r_i)_{1 \leq i \leq n} \in (\mathbb{R}_{\geq 0} \times \Sigma \times 2^C)^*$ and timed words as sequences of pairs in $(\mathbb{R}_{\geq 0} \times \Sigma)^*$.

Remark 5.2.1. *These denominations accentuate the language recognition point of view of learning. We wish to learn a model from a timed language that we query as needed, as such we are interested in* timed words *and add semantics and behavioural informations, such as* resets.

Event recording automata (ERA) [AFH99] form a subclass of TAs in which there is one clock c_a per action a of the alphabet Σ, and such that for each transition (ℓ, g, a, r, ℓ'), it holds $r = \{c_a\}$. Since clock resets are determined by actions, they can be retrieved from observed timed words, and this class of TAs is easier to learn than general TAs (although more complex than DFAs).

A TA is said *complete* when from any configuration (ℓ, v) and any letter a, there is a transition of the form (ℓ, g, a, r, ℓ') such that $v \models g$. Clearly enough, any timed automaton can be turned into a complete timed automaton accepting the same timed language.

Remark 5.2.2. *This definition is equivalent to the definition of completeness given in Chapter 3 (p.65) but stated in a more semantic manner.*

In a given *deterministic* timed automaton, a timed word (resp. timed word with resets) can be the trace (resp. signature) of *at most* one (alternating) run (ending in Σ). This unique run ρ such that $\mathsf{trace}(\rho) = w_t$ is then denoted $\mathsf{trace}^{-1}_{\mathsf{run}}(w_t)$. As a consequence, a timed word w_t is accepted if and only if the run $\mathsf{trace}^{-1}_{\mathsf{run}}(w_t)$ is (defined and) accepted.

In this chapter, we extend ERAs as *reset-optional ERAs* (RERAs), by allowing transitions to not reset their clock:

Definition 5.2.3. *A* reset-optional event recording automaton *(RERA) over* Σ *is a TA* $\mathcal{A} = (\mathcal{L}, \ell_{init}, C, E, \mathsf{Accept})$ *such that for all transitions* $(\ell, g, a, r, \ell') \in E$*, it holds* $r \subseteq \{c_a\}$.

While not as expressive as generic timed automata, RERAs are more complex than ERAs in a crucial way with respect to learning: the clock resets along a run ρ of an unknown RERA cannot be inferred directly from its observed timed word $w_t = \mathsf{obs}(\rho)$. Furthermore, RERAs are not determinizable in general. Indeed, the classical example of non-determinizable 1-clock TA proposed in Figure 1.5 and adapted to the current setting in Figure 5.1, is in fact a RERA. In the rest of this chapter, we will restrict our study to *deterministic* RERAs.

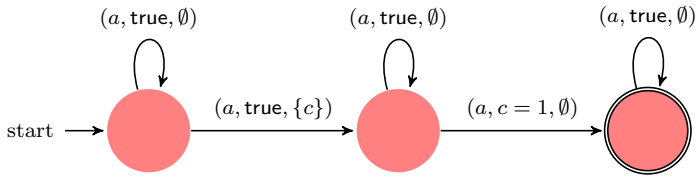


Figure 5.1: A one-clock non-determinizable RERA.

5.2.2 Active learning for timed automata

The general principle of (untimed) active learning is to learn a model from observations acquired by membership queries and equivalence queries made to a teacher, as displayed in Figure 5.2.

In membership queries, a word is provided to a teacher, who in return provides the observation, *i.e.* its membership to the target language. In an equivalence query, a hypothesis (a model) is proposed to the teacher; she either agrees it if it is equivalent to the model we wish to learn, or otherwise provides a counterexample, *i.e.* a word and its observation that separates the language of the target model and that of the hypothesis.

The set of observations, which grows with counter-examples and results from membership queries, is formalized as a partial function Acc mapping words to acceptance status ($+$ or $-$). To build a model, classical active learning algorithms then want to identify a prefix-closed subset U of $\mathsf{Dom}(\mathsf{Acc})$ such that for all letters a in the alphabet and words $u \in U$, $u.a \in \mathsf{Dom}(\mathsf{Acc})$ and either $u.a \in U$, or there is another word $u' \in U$

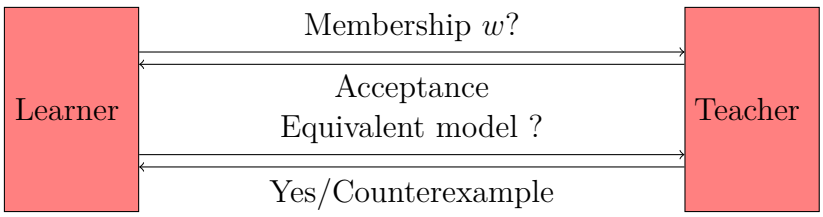


Figure 5.2: The basic active learning framework.

having the same observed behaviour as $u.a$. This is usually done by a data structure that can store observations, makes it possible to check some important properties on the set U and can be transformed into a hypothesis when the current set indeed has these properties. This is the role of the observation table in Angluin's L^* algorithm [Ang87a].

A high level pseudo-code of a generic active learning approach is presented in Algorithm 4. Its principle is the following: starting from the empty hypothesis (*i.e.* a model accepting the empty language), an equivalence query is made, and the counterexamples processed into the data structure, *i.e.* membership queries are made to help the structure get back its set of "good" properties. This is then repeated until an equivalence query returns yes.

Algorithm 4: A high-level pseudo-code of an active learning algorithm.

```
1 Start from the structure representing U = ∅ and the hypothesis H = ∅.
2 while H is not equivalent to the model we want to learn do
3 |   Process counterexample
4 |   Make and process membership queries to obtain good properties on U.
5 |   Fold the data structure into a model H.
6 Return H
```

In the setting of timed automata, the problem can be expressed in the very same way, with the definitions of the previous section: the learner makes membership queries about timed words, and aims at reconstructing a timed automaton matching those observations.

In this setting, the *acceptance function* is a partial function $\mathsf{Acc}\colon (\mathbb{R}_{\geq 0} \times \Sigma)^* \hookrightarrow \{+, -\}$ associating acceptance status to timed words that have been observed. We then define what it means for a timed automaton to *model* a set of observations:

Definition 5.2.4. *Let* Acc *be an acceptance function and* $\mathcal{A}$ *be a deterministic timed automaton. We say that* $\mathcal{A}$ *is a* model *for* Acc *when for all* $w_t \in \mathsf{Dom}(\mathsf{Acc})$*,* w_t *is accepting if, and only if,* $\mathsf{Acc}(w_t) = +$*.*

Remember that in a deterministic TA, there is at most a unique run $\rho = \mathsf{trace}^{-1}_{\mathsf{run}}(w_t)$ of $\mathcal{A}$ such that $\mathsf{trace}(\rho) = w_t$, and the acceptance of w_t by $\mathcal{A}$ is defined by the acceptance of ρ by $\mathcal{A}$. If there is no such run, then by definition w_t is not accepted.

Recall that the objective of an *active* learning agent is not to learn a model corresponding to a fixed set of observations, but rather to acquire enough information via adapted requests and hypotheses, so as to build an automaton equivalent to the teacher's model.

There are two main challenges when considering the timed-automata setting:

- the first one is the uncountable number of possible delays, and more generally the uncountable number of configurations of the automaton. This is a classical issue in the area of timed automata, and finite abstractions have already been studied extensively to cope with this;
- the second challenge lies in the fact that timed words do not contain enough information to recover the full configuration of the timed automaton they are learning: clock resets have to be inferred as well, proposing hypotheses and discarding them on-the-fly when they are contradicted by new observations.

In this chapter, we present a method to perform the active learning of deterministic reset-optional event recording automata. Precisely, we consider that the teacher has a deterministic RERA model in mind. This objective makes us face the two challenges mentioned above, as RERAs have both a high dimensionality and unobservable clock resets. We propose to deal with them by

- introducing relevant abstractions to group the (infinitely) many behaviours in a finite number of sets. These will be used to describe the different levels of precision that we need, namely answering the following two questions: how large can we afford our guards to be? What is the acceptance status of *elementary* (non distinguishable) sets of behaviours?
- using our abstractions to infer from the observations the relevance of different sequences of resets to explain the behaviours. Precisely, we detect reset combinations that do not allow to separate observations despite their disagreement on acceptance, and prune them as soon as they appear.

Section 5.3 introduces more formalism to abstract observations and behaviours of timed automata at different levels, according to our needs for the rest of the chapter.

In Section 5.4, our data-structures and their important properties are described. They separate decision making (akin to the construction of U) and reset guessing, as these two tasks require different levels of abstraction. In Section 5.5, we describe our version of the active learning algorithm and the properties verified by each function. Finally in Section 5.6, the construction of a hypothesis from the data structures is discussed.

5.3 Abstraction

In this section, we will detail several layers of abstractions that are useful to understand timed automata behaviours and that will be used in this chapter.

We extend the notions defined in Chapter 1 to obtain representations of single observations, minimal undistinguishable sets of observations, and large sets of observations that will serve to define the model behaviours (*i.e.* paths). These abstractions are central to our work on model-reality interactions, as they serve to define different languages encoding the information inferred from observations or induced from the models, *e.g.* clock dynamics. One could see a connection with abstract interpretation in the crucial importance of different representations (abstract domains) used to organise the information. The main difference probably being that we are only interested in *exact* representation, whereas abstract interpretation often serves to discuss the balance between precision and efficiency.

Two orthogonal kinds of abstractions arise.

Precision level: A run does not describe only the atomic behaviour that a timed automaton can encode, as if a run is accepted by an automaton, usually an infinite number of related runs are also accepted. Infinite sets of runs can be regrouped in so-called *zone runs*. Using *region equivalence* and K*-equivalence* relations, the atomic behaviour of timed automata can be finitely represented by specific zone runs called *region runs* and K*-closed runs*.

Another interesting level of precision is that of paths, as they define the behaviour of the model by fixing the guards. This gives us three decreasingly-precise levels: runs, region or K-closed runs and paths that are respectively related to single executions, atomic behaviours, and construction of the language.

Observation level: In Section 1.2, runs of timed automata and their corresponding traces, which are obtained by projecting away unobservable aspects, have been introduced. In many applications, such as learning, one intuitively wants to retrieve

runs from traces, *i.e.*, invert the projection. However runs depend on a specific model (notably on the locations and transitions names) that can not be learned. Hence, we will introduce model-agnostic intermediate notions. First, *abstract runs* will be defined, based on the information available in *timed words with resets*, which forget model-specific information. In the same way, but at a less precise level, *zone words with resets* and *guarded words with resets* will be defined as model-agnostic behaviours of zone runs and paths, and at the same time as infinite sets of abstract runs or timed words with resets.

The different notions that will be introduced in this section are summarized in Figure 5.3, along with the two axis of abstraction: precision and observation level.

5.3.1 Zone runs, region runs and K-closed runs

Alternating runs serve as representations of single executions, but we still lack a notion of minimally distinguishable set of runs, which we now define using K-equivalence.

We start by defining (alternating) *zone runs*, *i.e.*, runs in which all valuations are generalized to zones. Formally, a (partial) zone run is an element $\eta = ((\ell_{i-1}, z_{i-1}) \xrightarrow{\delta} (\ell_{i-1}, z'_{i-1}) \xrightarrow{e_i} (\ell_i, z_i))_{1\leq i\leq n}$ of $((\mathcal{L}\times\mathcal{Z})\times\{\delta\}\times(\mathcal{L}\times\mathcal{Z})\times E)^*\times(\mathcal{L}\times\mathcal{Z})$ such that for all $1 \leq i \leq n$, $z'_{i-1} \subseteq \overrightarrow{z}_{i-1}$ and for $e_i = (\ell_{i-1}, g, a, r, \ell_i)$, we have $z'_{i-1} \cap g \neq \emptyset$ and $z_i = (z'_{i-1} \wedge g)_{[r\leftarrow 0]}$. Here, δ is a symbolic representation of time elapse, as any precise delay may lead to different zones from different valuations in the same zone. We say that a partial zone run is a zone run when z_0 is the zone limited to $\mathbf{0}$.

We say that an alternating (partial) run $\rho = ((\ell_{i-1}, v_{i-1}) \xrightarrow{t_i} (\ell_{i-1}, v'_{i-1}) \xrightarrow{e_i} (\ell_i, v_i))_{1\leq i\leq n}$ *belongs to* a zone run $\eta = ((\ell'_{i-1}, z_{i-1}) \xrightarrow{\delta} (\ell'_{i-1}, z'_{i-1}) \xrightarrow{e'_i} (\ell'_i, z_i))_{1\leq i\leq n}$, noted $\rho \in \eta$ when for $0 \leq i \leq n$: $\ell_i = \ell'_i$, $v_i \in z_i$ and for $i > 0$, $e_i = e'_i$.

Similarly, we say that a partial zone run η^1 is included in a partial zone run η^2 of the same length n, noted $\eta^1 \subseteq \eta^2$, when for any $0 \leq i \leq n$, $\ell^1_i = \ell^2_i$, $z^1_i \subseteq z^2_i$, ${z'}^1_i \subseteq {z'}^2_i$ and $e^1_i = e^2_i$ if $i \neq 0$.

Remark 5.3.1. *Notice that once we know that $z^1_0 \subseteq z^2_0$ it is only necessary to check that ${z'}^1_i \subseteq {z'}^2_i$. Notably, for zone runs this is ensured by definition.*

We call *region run* a zone run in which all zones are regions and *K-closed run* a zone run in which all zones are K-closed zones. Notice that for any run ρ, there is a unique K-closed run η (resp. region run η) such that $\rho \in \eta$. Furthermore, any other run ρ' belonging to η is accepting if, and only if, ρ is.

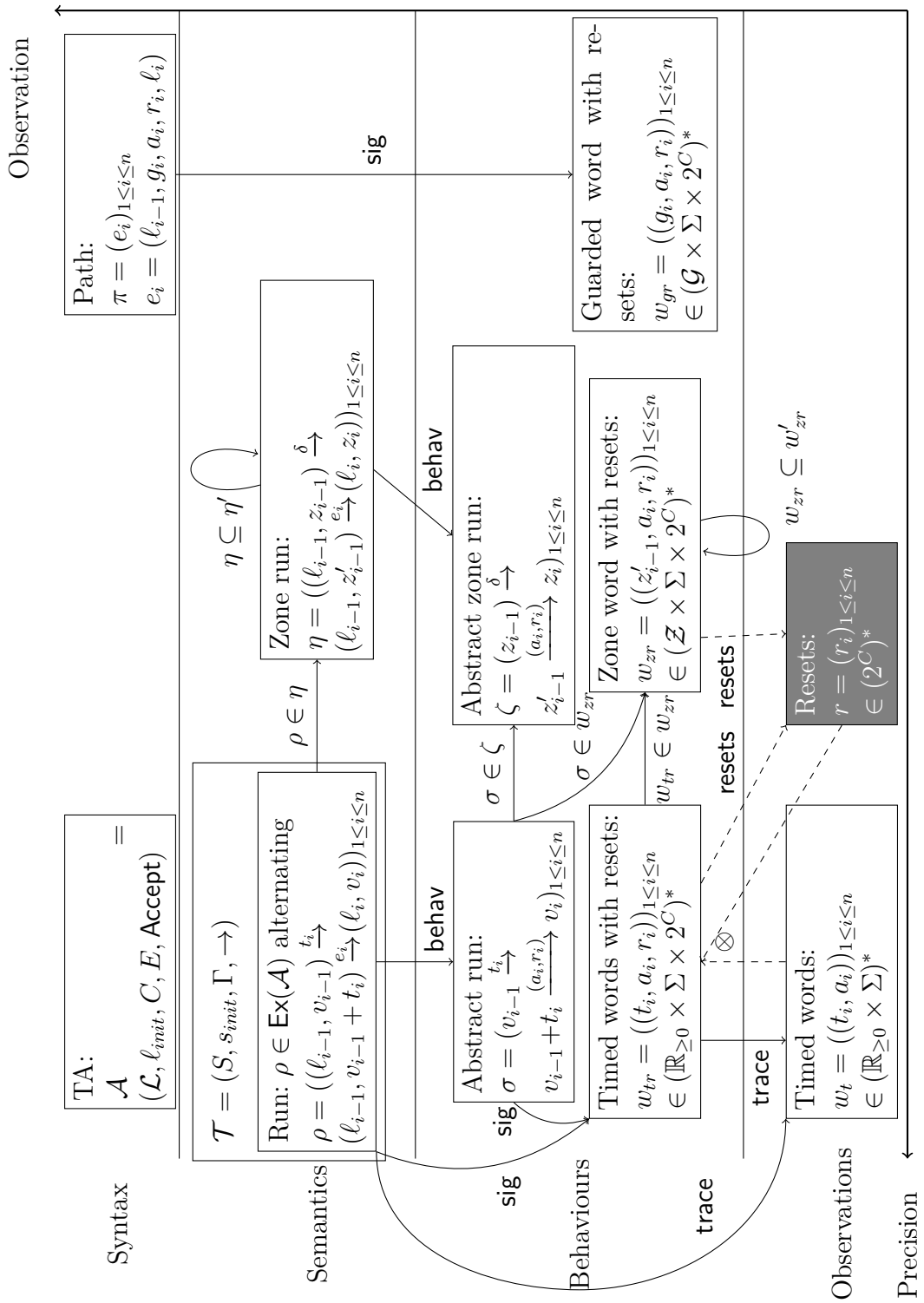


Figure 5.3: Words, runs, models and their relations.

We say that two runs are region (resp. K)-equivalent when they belong to the same region (resp. K-closed) run.

Interestingly, although K-closed runs concentrate on undistinguishability in the present, one can deduce a unique region run from a K-closed run, thanks to the interaction between the constraints and the resets.

Lemma 5.3.2. *Pairs of runs are K-equivalent if and only if they are region-equivalent.*

Proof. We can easily see that pairs of region-equivalent runs are K-equivalent, as region-equivalence adds strictly more constraints.

Consider a K-closed run η and two runs $\rho_1, \rho_2 \in \eta$. We show that ρ_1 and ρ_2 are region-equivalent. We make the proof by induction on the runs, noting $\rho^j = ((\ell_{i-1}, v_{i-1}^j) \xrightarrow{t_i^j} (\ell_{i-1}, {v'_{i-1}}^j) \xrightarrow{e_i} (\ell_i, v_i^j))_{1 \leq i \leq n}$ for $j \in \{1, 2\}$ and $\eta = ((\ell_{i-1}, z_{i-1}) \xrightarrow{\delta} (\ell_{i-1}, z'_{i-1}) \xrightarrow{e_i} (\ell_i, z_i))_{1 \leq i \leq n}$. The central fact underlying the proof is that a region has the same constraints as a K-closed zone plus constraints on the order on the fractional parts of the clock values, *i.e.* to be in the same region, the fractional parts of the clock values must be in the same order (with equalities at the same places).

First, $v_0^1 = v_0^2 = \mathbf{0}$ with all clocks evaluated to 0. These are clearly region-equivalent. After t_1, ${v'_0}^1 \approx_K {v'_0}^2$ and for both valuations, all (fractional parts of the) clock values are equal. Hence they are region-equivalent.

Inductively, starting from two region-equivalent valuations ${v'}_{i-1}^1 \approxeq_{reg} {v'}_{i-1}^2$, for $j \in \{1, 2\}$ $v_i^j = {v'_{i-1}}^j{}_{[r_i \leftarrow 0]}$, hence ${v_i}^1 \approxeq_{reg} {v_i}^2$ as resets preserve region-equivalence. After t_i the valuations $v_i'^j$ are K-equivalent and the order of the fractional parts of the clock valuations in $v_i'^j$ is equal to the one of v_i^j up to a permutation induced by the delay. We show that both runs are affected by the same permutation. Indeed, the permutation depends on the integer constants crossed by the clock values (*i.e.* the K-closed zones with constraints $c = n$ with $n \in \mathbb{N}$ and $c \in C$). Starting from the same region and ending in the same K-closed zone implies that both runs cross the same constants (those in the intersection of the past of the K-closed zone and the future of the region). Hence $v_i'^1 \approxeq_{reg} v_i'^2$.

By induction this gives us our result. □

Remark 5.3.3. *This result obviously does not hold for* partial *K-closed and region runs as it relies on the initial valuation $\mathbf{0}$ to ensure that first valuations are region equivalent.*

Remark 5.3.4. *In this chapter K-closed runs (and related notions) are abundantly used instead of region runs. This is because we are mostly interested in the question "given two*

runs, could a model have separated them?" which is encoded by K-equivalence (for timed automata without diagonal constraints).

5.3.2 Signatures and Behaviours

Now that all semantic notions are defined, we need a way to capture the corresponding behaviours that is model-agnostic, similarly to how timed words with resets capture alternating runs.

Between runs and timed words with resets, we define *abstract runs*, that keep the valuations of a run, and only abstract the locations of the timed automaton. For an alternating run $\rho = ((\ell_{i-1}, v_{i-1}) \xrightarrow{t_i} (\ell_{i-1}, v'_{i-1}) \xrightarrow{e_i} (\ell_i, v_i))_{1\leq i\leq n}$ we define its *behaviour* behav as the abstract run $\sigma = \mathsf{behav}(\rho) = (v_{i-1} \xrightarrow{t_i} v'_{i-1} \xrightarrow{(a_i,r_i)} v_i)_{1\leq i\leq n}$.

We define the signature of an abstract run $\sigma = (v_{i-1} \xrightarrow{t_i} v'_{i-1} \xrightarrow{(a_i,r_i)} v_i)_{1\leq i\leq n}$ as the projection on timed words with resets; *i.e.* $\mathsf{sig}(\sigma) = (t_i, a_i, r_i)_{1\leq i\leq n}$. Notice that this definition is coherent with the previous definition of signatures for runs as $\mathsf{sig}(\mathsf{behav}(\rho)) = \mathsf{sig}(\rho)$ for any run ρ.

From a timed word with resets, we have enough information to infer the clock values after each delay and action/reset pair, starting from the initial valuation. Formally, sig is bijective from abstract runs to timed words with resets. We note $\mathsf{sig}^{-1}_{\mathsf{behav}}$ its inverse. Considering a timed word with reset $w_{tr} = (t_i, a_i, r_i)_{1\leq i\leq n}$ we have $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_{tr}) = (v_{i-1} \xrightarrow{t_i} v_{i-1} + t_i \xrightarrow{(a_i,r_i)} v_i)_{1\leq i\leq n}$ with $v_0 = \mathbf{0}$ the initial valuation and $v_i = (v_{i-1} + t_i)_{[r_i\leftarrow 0]}$.

Remark 5.3.5. *As we have defined several signature functions (from runs and from abstract runs), we differentiate the inverse functions by precising their target: $\mathsf{sig}^{-1}_{behav}$ for abstract runs (*i.e. *behaviours) and sig^{-1}_{run} for runs. Despite similar notations, these two functions are very different in nature. Notably $\mathsf{sig}^{-1}_{behav}$ is always known while sig^{-1}_{run} depends on the model.*

We define similar abstractions and (inverse) projections for zone runs under the name of *abstract zone runs* and *zone words with resets* in $(\mathcal{Z} \times \Sigma \times 2^C)^*$.

For a zone run $\eta = ((\ell_{i-1}, z_{i-1}) \xrightarrow{\delta} (\ell'_{i-1}, z'_{i-1}) \xrightarrow{e_i} (\ell_i, z_i))_{1\leq i\leq n}$, its behaviour is defined as the *abstract zone run* $\zeta = \mathsf{behav}(\eta) = (z_{i-1} \xrightarrow{\delta} z'_{i-1} \xrightarrow{(a_i,r_i)} z_i)_{1\leq i\leq n}$ with $e_i = (\ell_{i-1}, g_i, a_i, r_i, \ell_i)$.

The signature of that abstract zone run is the zone word with resets $\mathsf{sig}(\zeta) = (z'_{i-1}, a_i, r_i)_{1\leq i\leq n}$. Notice that only the zone z'_{i-1} is taken into account, as we want to

know the zone after the observed delay but before the action transition and possible reset is taken.

Again, we define $\mathsf{sig}(\eta) = \mathsf{sig}(\mathsf{behav}(\eta))$ and $\mathsf{sig}^{-1}_{\mathsf{behav}}$ the inverse projection starting from $z_0 = \{\mathbf{0}\}$.

We call K-closed word (resp. region word) (with reset) a zone run with resets where all zones are K-closed zones (resp. regions).

For a path $\pi = (e_i)$, the abstraction has to keep the guards, as they are important to define the behaviours. For this, a specific kind of zone words with resets, named *guarded words with resets* (in $(\mathcal{G} \times \Sigma \times 2^C)^*$) is used. Precisely, for transitions $e_i = (\ell_{i-1}, g_i, a_i, r_i, \ell_i)$, we define the signature of the path as $\mathsf{sig}((e_i)_{1 \leq i \leq n}) = (g_i, a_i, r_i)_{1 \leq i \leq n}$.

Remark 5.3.6. *We do not define abstract paths as we have no use for them, but they could be defined similarly to abstract zones with resets.*

We extend the "belongs-to" relation of abstract runs with respect to abstract zone runs and zone words with resets (resp. the inclusion to pairs of abstract zone runs and zone words with resets) by checking that at all indices, the actions and resets correspond, and that the valuation after delay belongs to the zone, *i.e.* $v_{i-1} + t_i \in z'_{i-1}$ (resp. that at each index, the zone of the first word is included in the corresponding zone of the second one). This "belongs-to" relation can also be extended to timed words with resets with respect to zone runs with resets by $w_{tr} \in w_{zr}$ if $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_{tr}) \in w_{zr}$. We extend the notation Kz (used to define the K-closed zone containing a given valuation) to timed words with resets, so that for a timed word with resets w_{tr}, $Kz(w_{tr})$ is the unique K-closed word for which $w_{tr} \in Kz(w_{tr})$.

We can also extend the resets projection to all these notions (run, abstract run, zone runs with resets, zone words with resets, guarded words with resets) by projecting away everything but resets.

Finally, we say that a timed word w_t is *compatible* with a zone word with resets w_{zr} and write $w_t \dashv w_{zr}$, if there exists a timed word with resets $w_{tr} \in w_{zr}$ with $\mathsf{trace}(w_{tr}) = w_t$. In other words, letting $r = \mathsf{resets}(w_{zr})$, we have $w_t \dashv w_{zr}$ whenever $w_{tr} = w_t \otimes r$ belongs to w_{zr}.

Remark 5.3.7. *The abstractions defined in this section span the semantics (with zone runs) and behaviours (with abstract runs, zone and guarded words with resets) but not the observations. Indeed there is no point in generalizing observations: without information about clocks, one would not know what to deduce out of a single observation.*

5.3.3 Manipulations on words

In this subsection, some manipulations on timed words and K-closed words useful in the rest of the paper are presented.

Linear combinations of timed words

In our learning process, we will manipulate linear combinations of timed words. For two timed words $w_t^1 = ((t_i^1, a_i))_{0\leq i\leq n}$ and $w_t^2 = ((t_i^2, a_i))_{0\leq i\leq n}$ with the same untimed projection, we define their λ-*weighted sum* $w_t^3 = \lambda \cdot w_t^1 + (1-\lambda)\cdot w_t^2$, as the timed word $w_t^3 = ((\lambda \cdot t_i^1 + (1-\lambda)\cdot t_i^2, a_i)_{0\leq i\leq n})$. Such linear combinations have the following property:

Proposition 5.3.8. *For any two timed words $w_t^j = (t_i^j, a_i)_{0\leq i\leq n}$ for $j \in \{1,2\}$ with the same untimed projection, for any $0 \leq \lambda \leq 1$ and for any reset word $r = (r_i)_{0\leq i\leq n}$:*

$$\mathsf{sig}^{-1}_{\mathsf{behav}}((\lambda \cdot w_t^1 + (1-\lambda)\cdot w_t^2) \otimes r) = \lambda \cdot \mathsf{sig}^{-1}_{\mathsf{behav}}(w_t^1 \otimes r) + (1-\lambda)\cdot \mathsf{sig}^{-1}_{\mathsf{behav}}(w_t^2 \otimes r).$$

This means that for any valuation v_i reached in $\mathsf{sig}^{-1}_{\mathsf{behav}}((\lambda \cdot w_t^1 + (1-\lambda)\cdot w_t^2) \otimes r)$ and any clock $c \in C$, $v_i(c) = \lambda \cdot v_i^1(c) + (1-\lambda)\cdot v_i^2(c)$ with v_i^j the corresponding valuation in $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_t^j) \otimes r$.

Proof. The proof is made by induction on $0 \leq i \leq n$ considering the valuations v_i in $\lambda \cdot w_t^1 + (1-\lambda)\cdot w_t^2) \otimes (r_i)_i$. For $i = 0$, the only valuation encountered is $v_0 = \mathbf{0} = \lambda \cdot v_0^1 + (1-\lambda)\cdot v_0^2$. For $i > 0$, assume that we have the property for all valuations reached along the prefixes (*i.e.* for $i' < i$), and especially that $v_{i-1}(c) = \lambda \cdot v_{i-1}^1(c) + (1-\lambda)\cdot v_{i-1}^2(c)$. Then we have that $v_{i-1}(c) + \lambda \cdot t^1 + (1-\lambda)\cdot t^2 = \lambda \cdot (v_{i-1}^1 + t^1) + (1-\lambda)\cdot(v_{i-1}^2 + t^2)$ and by applying the resets commended by r we obtain the result desired for the i-th valuation. □

Offsets on zones and directions

We generalize the notion of offset from valuations (*e.g.* $v + t$ for $t \in \mathbb{R}_{\geq 0}$) to zones, with integer (positive or negative) offsets affecting subsets of clocks in the following way.

Definition 5.3.9. *Consider a zone z defined by a set of constraints of the form $c_i \prec k$ and $c_i - c_j \prec k$ with $c_i, c_j \in C$, $\prec \in \{<, \leq, =, \geq, >\}$ and $k \in \mathbb{N}$. For a vector of integers $N = \begin{pmatrix} n_1 \\ \cdots \\ n_{|C|} \end{pmatrix} \in \mathbb{Z}^{|C|}$ we note $z + N$ the zone defined by the constraints*

$$\forall\ i, j \in \{1, \ldots, |C|\},\ c_i \prec k + n_i \text{ and } c_i - c_j \prec k + n_i - n_j.$$

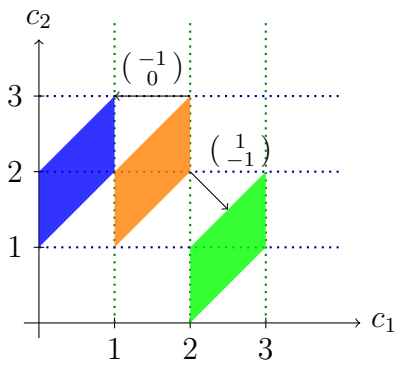


Figure 5.4: The effect of some offsets on a zone.

Example 5.3.10. *In Figure 5.4, an open zone is presented, together with the effect of multiples offsets on it. Offsets are noted* $\binom{n_1}{n_2}$ *with* n_1 *the offset on clock* c_1 *and* n_2 *the offset on clock* c_2*.*

Finally, it will be useful to consider the effect of modifying the delay at a given index in a word on the rest of the word. This is formalized through the notion of *direction*. To ease the notations, we will note $\mathbb{1}_{C'}$ for $C' \subseteq C$ the vector with ones in the dimensions of clocks in C' and zeros everywhere else.

Definition 5.3.11. *Consider a sequence of resets* $r = (r_k)_{1 \leq k \leq n}$*. For two indices* $1 \leq i \leq j \leq n$ *we define the* direction *at rank* i *on rank* j*, noted* $dir_i^j(r)$*, as*

$$dir_i^j(r) = \mathbb{1}_{C'} \text{ with } C' = \{c \in C \mid \forall i \leq k < j,\ c \notin r_k\}.$$

We generalize directions to $j < i$ *by fixing* $dir_i^j(r) = \mathbb{1}_\emptyset$ *and by defining directions on all words with resets (*i.e. *time, zone and guarded words with resets).*

We will use directions to build offsets. To explain the convention for $j < i$, we give a characterization of directions for timed words with resets.

Proposition 5.3.12. *Consider a timed word with resets* $w_{tr} = (t_i, a_i, r_i)_{1 \leq i \leq n}$ *and its corresponding abstract run* $\sigma = \mathsf{sig}^{-1}_{behav}(w_{tr}) = (v_{i-1} \xrightarrow{(t_i,a_i,r_i)} v_i)_{1 \leq i \leq n}$*. Then for any* $1 \leq k, j \leq n$*,* $dir_k^j(w_{tr})$ *corresponds to the effect on* $v_{j-1} + t_j$ *of adding one time unit to* t_k*. Formally, noting* $w'_{tr} = (t_i, a_i, r_i)_{1 \leq i < k} \cdot (t_k + 1, a_k, r_k) \cdot (t_i, a_i, r_i)_{k < i \leq n}$*, and* $\sigma = \mathsf{sig}^{-1}_{behav}(w'_t r) = (v'_{i-1} \xrightarrow{(t'_i,a_i,r_i)} v'_i)_{1 \leq i \leq n}$*, it holds*

$$v'_{j-1} + t'_j = (v_{j-1} + t_j) + dir_k^j(w_{tr}).$$

Proof. For any clock $c \in C$, $(v'_{j-1} + t'_j)(c)$ is the sum of delays since the last index l at which c was reset. If $l \geq k$ then adding time to t_k does not modify the valuation of c. By definition we have $dir_k^j(w_{tr})(c) = 0$, and the equality holds. Conversely, if $l < k$, or if clock c has not been reset, then the modification directly impacts the valuation by the same amount, and we have $dir_k^j(w_{tr})(i) = 1$, matching the addition of 1 time unit. □

Example 5.3.13. *Consider the timed word with resets (for $C = \{c_a, c_b, c_c\}$)*

$$w_{tr} = (1.2, a, c_a)(0.3, b, c_b)(1.6, a, c_a)(2.6, c, c_c)(0.1, c, \emptyset)(0.1, c, \emptyset).$$

Then $dir_6^6(w_{tr}) = dir_5^6(w_{tr}) = 1_C$, $dir_4^6(w_{tr}) = 1_{\{c_a, c_b\}}$, $dir_3^6(w_{tr}) = 1_{\{c_b\}}$ and $dir_2^6(w_{tr}) = dir_1^6(w_{tr}) = \mathbb{1}_\emptyset$.

5.4 Observation structures

In this section, we describe the structures used to represent and process the timed observations acquired during learning (and stored via the acceptance function Acc), *and* the decisions we make based on those observations.

We begin with defining a generic structure, which we call *timed decision graphs*, which are tree-shaped structures encoding (abstractions of) timed words, as well as all possible combinations of resets. We then define two special instances of these graphs:

- *timed language graphs* are timed decision graphs in which we reason more specifically about guards that have to be assigned to transitions. Timed language graphs will be very close to the candidate timed automata we aim at building;
- *timed observation graphs* are timed decision graphs in which we reason more specifically about reset sequences. We use them to detect invalid sequences of resets.

Remark 5.4.1. *From now on, we will rely on the targeted class of automata: RERAs. Notably, we will replace resets denoted by 2^C by $\{\top, \bot\}$ where $\top$ denotes that the clock corresponding to the transition action is reset and $\bot$ that it is not.*

This will notably serve in the construction of the timed decision graphs and in the different languages to simplify the expressions.

We did not apply these restrictions before as the abstractions are fitting for any timed automaton and thus need not be restrained in this way.

5.4.1 Timed decision graphs

Definition 5.4.2. *A* timed decision graph (TDG) *is a labelled graph $\mathcal{D} = (S, E)$ where S is partitioned into two sets S_o and S_d as follows:*

- $S_o \subseteq (\mathcal{Z} \times \Sigma \times \{\top, \bot\})^*$ *is the set of* observation states*: it is a non-empty prefix-closed set of zone words with resets. We write s_o^0 for the empty word, which always belongs to S_o. With any state $s_o \in S_o$, we associate the zone $\zeta(s_o)$ reached after the zone word with reset s_o, and defined inductively as follows:*
  - $\zeta(s_o^0) = \{\mathbf{0}\}$;
  - $\zeta(w_{zr} \cdot (z, a, r)) = (\overrightarrow{\zeta(w_{zr})} \cap z)_{[r \leftarrow 0]}$ *(where we consider that $r = \{c_a\}$ when $r = \top$ and $r = \emptyset$ if $r = \bot$).*
- $S_d \subseteq S_o \times \mathcal{Z} \times \Sigma$ *is the set of* decision states*; a state $s_d = (s_o, z, a)$ represents the situation where we observe the occurrence of action a, and have to decide whether clock c_a will be reset. We require that*
  - *for any $s_d = (s_o, z, a)$, it holds $\zeta(s_o) \cap z \neq \emptyset$;*
  - *for any s_o, for any a, if both (s_o, z, a) and (s_o, z', a) are in S_d, then $z = z'$ or $z \cap z' = \emptyset$.*

Edges are defined as follows: we have $E = E_{od} \uplus E_{do} \subseteq S \times ((\mathcal{Z} \times \Sigma) \cup \{\top, \bot\}) \times S$, such that for any $s_o \in S_o$ and any $s_d \in S_d$:

- *there is an edge $(s_o, f, s_d) \in E_{od}$ if, and only if, f is of the form (z, a) and $s_d = (s_o, z, a)$;*
- *there is an edge $(s_d, f, s_o) \in E_{do}$ if, and only if, $f \in \{\top, \bot\}$ and, writing $s_d = (s'_o, z, a)$, we have $s_o = s'_o \cdot (z, a, f)$.*

Notice that by our definition, for all $s_d = (s_o, z, a)$, the state s_o belongs to S_o. Symmetrically, we require that for all $s_o = s'_o \cdot (z, a, f)$, the state $(s'_o, z, a) \in S_d$. This way, our timed decision structures always are connected, tree-shaped graphs.

Finally, we require that all decision states have at least one successor, so that all leaves are observation states.

We use timed decision graphs to represent and manipulate observations stored in the Acc function (*i.e.* in $\mathsf{Dom}(\mathsf{Acc})$). Fix such a function Acc, and a TDG $\mathcal{D}$. With any

observation state $s_o \in S_o$ of $\mathcal{D}$, corresponding to a zone word with resets w_{zr}, we associate the (possibly empty) set of timed words that are observation-compatible with it, formally:

$$\mathsf{words}(s_o) = \{w_t \in \mathsf{Dom}(\mathsf{Acc}) \mid w_t \dashv w_{zr}\}.$$

We also let

$$\mathsf{label}(s_o) = \{\mathsf{Acc}(w_t) \mid w_t \in \mathsf{words}(s_o)\}.$$

The labelled TDG associated with $\mathcal{D}$ and Acc is then defined as the pair $\mathcal{D}_{\mathsf{Acc}} = (\mathcal{D}, \mathsf{label})$.

Not all labelled TDGs are relevant to handle acceptance functions. A labelled TDG $\mathcal{D}_{\mathsf{Acc}}$ is said

- *complete* (w.r.t. Acc) when $\mathsf{Dom}(\mathsf{Acc}) \subseteq \bigcup_{s_o \in S_o} \mathsf{words}(s_o)$;
- *well-grounded* (w.r.t. Acc) if for any observation state $s_o \in S_o$ it holds $\mathsf{words}(s_o) \neq \emptyset$.

Completeness means that all observations are represented in the TDG, while well-groundedness means that every observation state has a non-empty set of compatible observations. A labelled TDG is said to *implement* an acceptance function Acc when it is both complete and well-grounded w.r.t. Acc.

We will use two instantiations of these structures in our learning process:

- *timed language graphs* (TLG) are timed decision graphs with the following extra requirements:
  - all zones appearing in a TLG are guards (*i.e.*, they do not involve diagonal constraints);
  - for any s_o, writing $Z = \{z \mid (s_o, z, a) \in S_d\}$, if Z is non-empty, then Z partitions the set of clock valuations $\mathbb{R}^C_{\geq 0}$.

  As we explain below, TLGs will be used to refine guards, eventually representing (the unfolding of) a candidate timed automaton modelling the current observation.
- *timed observation graphs* (TOG) are timed decision graphs in which
  - all zones that appear in the definitions of the states are K-closed zones;
  - all decision states have two successors, by both $\top$ and $\bot$.

As we explain below, these will be useful for characterizing (im)possible sequences of clock resets. In a TOG, we note s_ϵ the root state, as it corresponds to the empty word.

Notice that in general, different labelled TDGs can implement the same acceptance function Acc, because different zones can appear in the zone words defining the states of the TDG. This is the case for TLGs. However there is unicity for TOGs, as proven below.

Proposition 5.4.3. *Given an acceptance function Acc, there is a unique TOG $\mathcal{M}$ implementing it.*

Proof. Consider $\mathcal{M}$ and $\mathcal{M}'$ implementing Acc. By definition of TOGs, all zones appearing in these graphs are K-closed, and thus all the zone words are K-closed words.

We will first prove that $S_o^{\mathcal{M}} \subseteq S_o^{\mathcal{M}'}$. As $\mathcal{M}$ is well-grounded w.r.t. Acc, for any $s_o \in S_o^{\mathcal{M}}$. there exist some $w_t \in \mathsf{Dom}(\mathsf{Acc}) \cap \mathsf{words}(s_o)$. And as $\mathcal{M}'$ is complete, there also exists a state $s'_o \in S_o^{\mathcal{M}'}$ such that $w_t \in \mathsf{words}(s'_o)$. Furthermore, since by definition of TOGs all decision states in $\mathcal{M}'$ have two successors, one can choose s'_o with the same sequence of resets as s_o, *i.e.* $r = \mathsf{resets}(s'_o) = \mathsf{resets}(s_o)$. Now, given w_t and r, the K-closed word with resets w_{zr} such that $(w_t \otimes r) \in w_{zr}$ is unique, hence we get $s_o = w_{zr} = s'_o$.

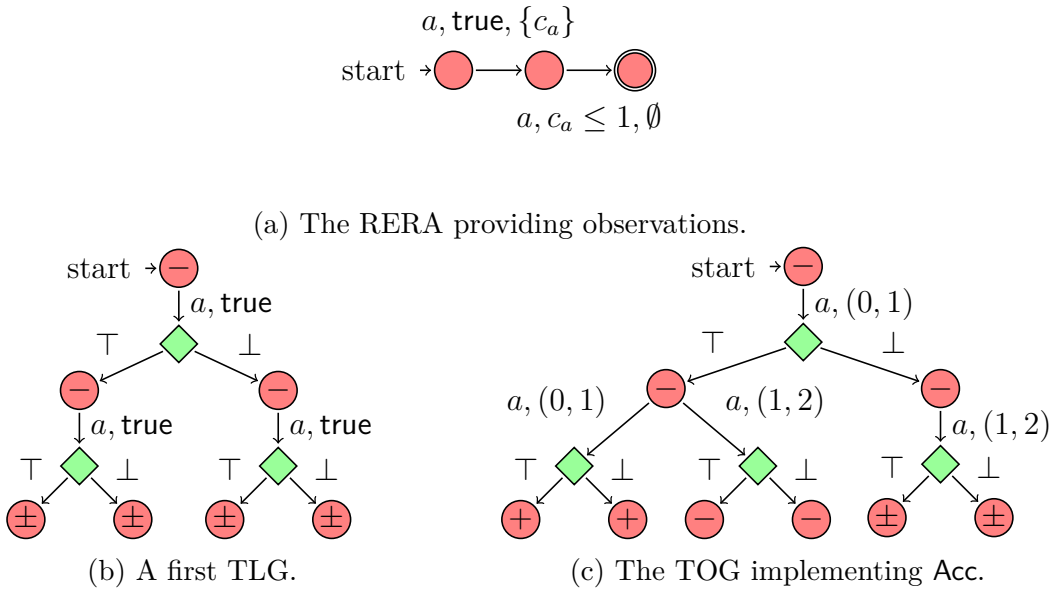
It comes that $S_o^{\mathcal{M}} \subseteq S_o^{\mathcal{M}'}$. We can show the opposite inclusion by symmetry, and thus $S_o^{\mathcal{M}} = S_o^{\mathcal{M}'}$. By the definition of TOGs, this implies that $S_d^{\mathcal{M}} = S_d^{\mathcal{M}'}$ and $E^{\mathcal{M}} = E^{\mathcal{M}'}$. Thus $\mathcal{M} = \mathcal{M}'$. □

Example 5.4.4. *Consider the RERA of Figure 5.5a with the final location being the only accepting one. If the learner issues requests for the timed words ϵ, $(0.7, a)$, $(0.7, a)(0.9, a)$ and $(0.7, a)(1.2, a)$, they end up with the following acceptance function:*

$$\mathsf{Acc}(\epsilon) = - \qquad \mathsf{Acc}((0.7, a)(0.9, a)) = +$$
$$\mathsf{Acc}((0.7, a)) = - \qquad \mathsf{Acc}((0.7, a)(1.2, a)) = -$$

The graph of Figure 5.5b represents a labelled TLG implementing this acceptance function Acc; in that figure (and all similar figures in this chapter), circle states are observation states and diamond states are decision states. In that example, all zones (which we see here as guards) contain all clock valuations.

Notice that some states (here all leaves) are labelled with both $-$ and $+$, indicating that this graph (which can be seen as a timed automaton) is not a good candidate for our learning situation, and has to be refined.



(a) The RERA providing observations.

(b) A first TLG.

(c) The TOG implementing Acc.

Figure 5.5: An active-learning setting.

The TOG implementing Acc *is displayed in Figure 5.5c. Notice that the rightmost leaves are both accepting and non-accepting, as both* $(0.7, a)(0.9, a)$ *and* $(0.7, a), (1.2, a)$ *satisfy their constraints. As the guards of a TOG are minimal, we know that these two words are equivalent and that any model for those observations* should *reset the clock* c_a *after the first transition (at least for* $0 < c_a < 1$*).*

Timed language graphs and timed observation graphs provide two different views of a set of observations, with opposite levels of abstraction. Timed observation graphs are *precise* in the sense that they distinguish the behaviours as finely as possible for a deterministic RERA, using K-equivalence as a base.

Timed language graphs are made to be as *general* as possible, only featuring guards that are necessary to model the observations, *i.e.* a transition labelled by a guard g is only split in g_1 and g_2 when it is necessary to distinguish accepting and non-accepting behaviours.

Remark 5.4.5. *This notion of necessity is local: from a given TLG, if it is necessary to add a guard this does not mean that there exist no fitting TLGs without that guard: perhaps a different one could have been added, or changing another guard would make the addition unnecessary.*

We introduce a function $\alpha\colon S_o^{\mathcal{M}} \to S_o^{\mathcal{N}}$ that links these two models, both in order to display the strong links between the two graphs and for later algorithmic use.

Definition 5.4.6. *Consider a timed language graph $\mathcal{N}$ and a timed observation graph $\mathcal{M}$. For two observation states $s_o^{\mathcal{M}} = w_{zr} \in S_o^{\mathcal{M}}$ and $s_o^{\mathcal{N}} = w_{gr} \in S_o^{\mathcal{N}}$, we define $\alpha(s_o^{\mathcal{M}}) = s_o^{\mathcal{N}}$ when $w_{zr} \subseteq w_{gr}$.*

We prove below that α is well-defined, and some of its basic properties:

Proposition 5.4.7. *Consider a complete TLG $\mathcal{N}$ for Acc and a TOG $\mathcal{M}$ implementing Acc. Then α is well-defined. It is surjective if, and only if, the TLG $\mathcal{N}$ is well-grounded (thus implements Acc). Furthermore, for all $s_o^{\mathcal{M}} \in S_o^{\mathcal{M}}$, $\mathsf{words}(s_o^{\mathcal{M}}) \subseteq \mathsf{words}(\alpha(s_o^{\mathcal{M}}))$, (and consequently $\mathsf{label}(s_o^{\mathcal{M}}) \subseteq \mathsf{label}(\alpha(s_o^{\mathcal{M}}))$).*

Proof. We prove the different properties independently.

- To ensure that α is well-defined, it suffices to check that for every $s_o^{\mathcal{M}} \in S_o^{\mathcal{M}}$ there is a unique covering guarded word $s_o^{\mathcal{N}}$ in $S_o^{\mathcal{N}}$. The unicity comes by induction by the properties of the observation states of a TLG: there is no overlapping between guards corresponding to the same action after a given language state (by definition of a TLG); the existence comes from the *well-groundedness* of $\mathcal{M}$ w.r.t. Acc and the completeness of the observation structure: by well-groundedness, for each observation state $s_o^{\mathcal{N}}$ there is an observation w_t, and by completeness, that observation w_t is covered by the TLG.

- If the TLG is well-grounded, then for any $s_o^{\mathcal{N}} \in S_o^{\mathcal{N}}$, there is a timed word w_t of Acc in $\mathsf{words}(s_o^{\mathcal{N}})$. By completeness of $\mathcal{M}$ w.r.t. Acc, there exists some observation state $s_o^{\mathcal{M}} \in S_o^{\mathcal{M}}$, with $w_t \in \mathsf{words}(s_o^{\mathcal{M}})$, and thus $\alpha(s_o^{\mathcal{M}}) = s_o^{\mathcal{N}}$. Thus α is surjective.

  Conversely, suppose that α is surjective. Then for any $s_o^{\mathcal{N}} = w_{gr} \in S_o^{\mathcal{N}}$, there exists $s_o^{\mathcal{M}} \in S_o^{\mathcal{M}}$ such that $\alpha(s_o^{\mathcal{M}}) = s_o^{\mathcal{N}}$. By well-groundedness of $\mathcal{M}$, there is $w_t \in \mathsf{Acc}$ such that $w_t \in \mathsf{words}(s_o^{\mathcal{M}})$ and by definition of α, we get $w_t \dashv w_{gr}$. Well-groundedness of $\mathcal{N}$ then follows by definition of words.

- The inclusion $\mathsf{words}(s_o^{\mathcal{M}}) \subseteq \mathsf{words}(\alpha(s_o^{\mathcal{M}}))$ comes from the fact that any observation that belongs to $s_o^{\mathcal{M}}$ also belongs to $\alpha(s_o^{\mathcal{M}})$.

□

5.4.2 Consistency and validity

In the learning process we will construct TLGs, *i.e.* basically prefix-closed sets of guarded words with resets associated with their sets of compatible observed timed words. These guarded words with resets are the premises of paths in the targeted TA. However only some of them can form those paths in a deterministic TA modeling Acc, since their image by Acc must be unique for all compatible observed timed words. This is formalized by the notion of consistency:

Definition 5.4.8. *A guarded word with resets w_{gr} is said* consistent *with respect to an acceptance function Acc when for all $w_t, w'_t \in \mathsf{Dom}(\mathsf{Acc})$, if w_t and w'_t are compatible with w_{gr} then $\mathsf{Acc}(w_t) = \mathsf{Acc}(w'_t)$. Otherwise, w_{gr} is said* inconsistent.

Detecting and handling inconsistencies is central to our algorithms, as it characterizes the need to introduce new guards to split observation nodes in the timed language graphs. For this, we use the following characterization of inconsistencies of observation states of a TLG.

Proposition 5.4.9. *Consider a labelled TLG $\mathcal{N}$ implementing an acceptance function Acc. An observation state $s_o \in S_o^{\mathcal{N}}$ is inconsistent if and only if $|\mathsf{label}(s_o)| = 2$.*

Proof. By definition, s_o is inconsistent if and only if there exists two timed words w_t, w'_t in $\mathsf{Dom}(\mathsf{Acc})$ compatible with s_o, but $\mathsf{Acc}(w_t) \neq \mathsf{Acc}(w'_t)$. By definition of labelled TLGs, words and label, this is equivalent to $w_t, w'_t \in \mathsf{words}(s_o)$ such that $\mathsf{Acc}(w_t) \neq \mathsf{Acc}(w'_t)$, which is equivalent to $|\mathsf{label}(s_o)| = 2$. □

Then, we say that a timed language graph $\mathcal{N}_{\mathsf{Acc}}$ is *consistent* when for any $s_o \in S_o$, we have $|\mathsf{label}(s_o)| = 1$; it is said inconsistent otherwise.

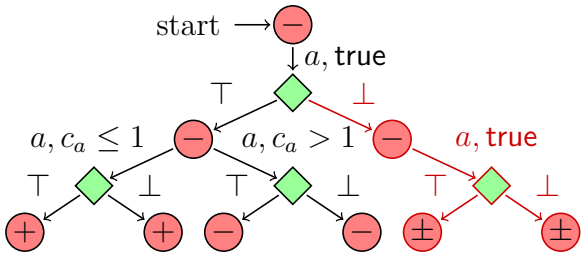


Figure 5.6: The TLG obtained after handling an inconsistency in the TLG of Figure 5.5b.

Example 5.4.10. *The leaves of the TLG in Figure 5.5b are inconsistent: the labels of all leaves have size* 2*. That TLG can be refined into the TLG depicted in Figure 5.6: in the left subtree, guard* true *has been split into two guards,* $c_a \leq 1$ *on one side, and* $c_a > 1$ *on the other side. In this situation, the learner may issue an extra request, namely* $(0.7, a)(1, a)$*, in order to decide how to set the guards; the TLG of Figure 5.6 corresponds to the case where* $\mathsf{Acc}((0.7, a)(1, a)) = +$*.*

In the subtree to the right, however, it can be checked that no guards (only involving clock x_a *and integer constants) can resolve the inconsistencies in this subtree: this indicates that resetting clock* x_a *in the first decision state is the only solution in this example.*

We claim that detecting impossible reset decisions on the TLG, as we do in Example 5.4.10, is not an efficient approach. Indeed, detecting such an impossibility requires discarding every possible guard that could separate the observations. This can be done either by refining guards until no refinements are possible (which may require numerous membership queries, see Algorithm 9 in Section 5.5.2 for more insight), or by checking on any pair of words leading to an inconsistency, whether there is a guard that can separate them, which again may require a large number of queries.

Example 5.4.11. *Consider again the situation of Example 5.4.10; Figure 5.7 displays the TOG implementing the corresponding acceptance function; remember that in a TOG, all zones are* K*-closed zones. Notice that it has labels of size two on the leaves of the right branch. While such incompatibilities can be resolved on TLGs by refining guards, here guards cannot be refined, and as we now argue, this indicates that the decision not to reset the clock cannot be correct.*

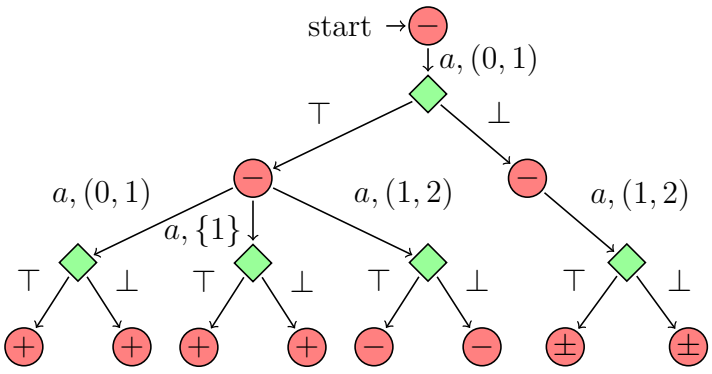


Figure 5.7: The timed observation graph implementing Acc for Example 5.4.10.

As exemplified above, TOGs are composed with K-closed words with resets, that

cannot be refined. However some of them may not be possible in any deterministic TA modeling the acceptance function Acc. This is formalized by the notion of invalidity:

Definition 5.4.12. *Let Acc be an acceptance function. Let w_{zr} be a K-closed word with resets. We say that w_{zr} is* valid *for Acc if there exists a deterministic complete RERA $\mathcal{A}$ that models Acc such that all timed words with resets in w_{zr} are signatures of runs of $\mathcal{A}$. Otherwise, w_{zr} is said* invalid*.*

We generalize (in)validity to TLGs and say that an observation state s_o of a TLG is invalid when there exists an invalid K-closed word with resets included in s_o. We call s_o valid otherwise. We say that the TLG $\mathcal{N}$ is valid when it has no invalid observation state. We call the valid part *of a TLG its restriction to valid observation states and intermediate decision states.*

In the particular case of TOGs, the following proposition characterizes invalid reset combinations in the (unique) TOG implementing Acc; this will allow us to identify and prune them:

Proposition 5.4.13. *Let Acc be an acceptance function, and $\mathcal{M}_{\mathsf{Acc}}$ the TOG implementing Acc. The set of* marked *observation states of $\mathcal{M}_{\mathsf{Acc}}$ is defined by the least fixpoint of the following marking:*

- *any observation state s_o with $|\mathsf{label}(s_o)| = 2$ is marked;*
- *if, for some observation state s_o, there is an action a and a K-closed zone z such that both observation states corresponding to $s_o \cdot (z, a, \top)$ and $s_o \cdot (z, a, \bot)$ exist and are marked in $\mathcal{M}_{\mathsf{Acc}}$, then s_o is marked;*

A K-closed word with resets $w_{zr} \in S_o^{\mathcal{M}_{\mathsf{Acc}}}$ is invalid w.r.t. Acc if, and only if it has a marked predecessor.

Proof. Consider a TOG $\mathcal{M}_{\mathsf{Acc}}$ implementing Acc and an observation state $s_o = w_{zr}$.

Let us first consider that s_o is marked and prove that w_{zr} is invalid by induction on the definition of marked states.

First assume that $|\mathsf{label}(s_o)| = 2$; then by definition of label and words, there exist two timed words w_t and w'_t compatible with w_{zr}, such that $\mathsf{Acc}(w_t) \neq \mathsf{Acc}(w'_t)$. Take the two corresponding timed words with resets $w_{tr} = w_t \otimes \mathsf{resets}(w_{zr})$ and $w'_{tr} = w'_t \otimes \mathsf{resets}(w_{zr})$, implementing the resets of w_{zr} along w_t and w'_t. If a timed automaton $\mathcal{A}$ contains w_{tr} as a

run, it also contains w'_{tr} (because they correspond to the same K-closed word with resets). Since $\mathcal{A}$ must be deterministic, either both w_{tr} and w'_{tr} correspond to accepting runs, or they both correspond to non-accepting runs. By Definition 5.2.4, $\mathcal{A}$ cannot be a model for Acc.

Second, suppose $s_o = w_{zr}$ and there is an action a and a K-closed zone z such that both $s'_o = w_{zr} \cdot (z, a, \top)$ and $s''_o = w_{zr} \cdot (z, a, \bot)$ are marked. By induction hypothesis, they are invalid.

Assume that some complete deterministic RERA $\mathcal{A}$ contains a timed word with resets $w_{tr} \in w_{zr}$ as a run, then by completeness it would also contain $w_{tr} \cdot (v, a, \top)$ or $w_{tr} \cdot (v, a, \bot)$ for some $v \in z$, contradicting our induction hypotheses as these are invalid.

We thus have proven that all marked states are invalid.

Now, assume that $s_o = w_{zr}$ is a successor of a marked observation state $s'_o = w'_{zr}$. Hence w'_{zr} is invalid. By definition, no timed automata containing a timed word with resets $w'_{tr} \in w'_{zr}$ may be a model for Acc. Then, this directly extends to w_{zr} (as all automata containing w_{zr} contain its prefixes), and w_{zr} is invalid.

Conversely, we consider a zone word with resets w_{zr} invalid for the acceptance function Acc, and try to prove that the observation state $s_o = w_{zr}$ has a marked prefix in the TOG $\mathcal{M}_{\mathsf{Acc}}$. The proof follows the following scheme: we first show how TAs can be extracted from the TOG. We then discuss on such TAs containing paths for w_{zr}, and use the fact that they cannot be models to find states that cover accepting and non accepting observations. Finally, the definition of marked states is used to reach the conclusion. The difficulty comes from the fact that the pair of observations that we have may not correspond to w_{zr} in the first place (we have no guarantees except that the TA is not a model). Hence we have to show that other labels of size two can be handled by changing the reset choices (thus changing the TA extracted from the TOG) *without modifying the path containing* w_{zr}. This enforces that, in the end, a pair relevant for w_{zr} will appear.

By definition of invalidity any deterministic RERA $\mathcal{A}$ containing signatures in w_{zr} is not a model of Acc. This is the case in particular for any (tree-shaped) RERA obtained from the TOG $\mathcal{M}_{\mathsf{Acc}}$ by keeping only observation states, freely choosing whether or not to reset the clock on each transition, and making leaves accepting or not depending on label (both choices are possible when $|\mathsf{label}(s'_o)| = 2$).

Take such a TA $\mathcal{A}_{\mathcal{M}}$ containing the path π corresponding to w_{zr} in the TOG. It cannot be a model of Acc, so by Definition 5.2.4 there must be $w_t \in \mathsf{Dom}(\mathsf{Acc})$ such that $\mathsf{Acc}(w_t)$ disagrees with the acceptance of w_t in $\mathcal{A}_{\mathcal{M}}$. Notice that, for $\mathsf{Acc}(w_t)$ not to agree with

the acceptance of the target location, it has to end in a location in $\mathcal{A}_{\mathcal{M}}$ built from an observation state s'_o with $|\mathsf{label}(s'_o)| = 2$. Hence, there is another timed word w'_t in $\mathsf{words}(s'_o)$ such that $\mathsf{Acc}(w_t) \neq \mathsf{Acc}(w'_t)$.

We will now prove that w_{zr} has a marked predecessor. We consider two cases:

- first, assume that for any reset choices, w_t and w'_t cannot be separated by any guard. Formally: for any sequence of reset choices $r \in \{\top, \bot\}^{|w_t|}$, for any index $0 \leq j \leq |w_t|$, and for any guard g, writing v_i and v'_i for the clock valuations reached at step i along the abstract runs $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_t \otimes r)$ and $\mathsf{sig}^{-1}_{\mathsf{behav}}(w'_t \otimes r)$ obtained from w_t and w'_t by resetting clocks according to r, we have $v_i \models g$ if, and only if, $v'_i \models g$.

  In this case, all leaves in the TOG $\mathcal{M}_{\mathsf{Acc}}$ that can be reached by reading w_t or w'_t are labelled with both $+$ and $-$, and are marked by our marking procedure. By our hypothesis, all intermediary valuations visit the same K-closed zones, and all corresponding states in the TOG are marked. In the end, the root of the TOG is marked, and then all states have a marked predecessor. In particular, it is true for $s_o = w_{zr}$;

- now, assume that there exist two timed words with resets $w_{tr} = w_t \otimes r$ and $w'_{tr} = w'_t \otimes r$, obtained from w_t and w'_t by adding a same sequence of resets r, that can be separated at some step i by a guard g. This sequence of resets r might not be the one considered in $\mathcal{A}_{\mathcal{M}}$ but we can modify the reset sequence of $\mathcal{A}_{\mathcal{M}}$ along the branch corresponding to w_t and w'_t (*i.e.* by changing the reset choices and using different branches of $\mathcal{N}$).

  - First assume that we can modify $\mathcal{A}_{\mathcal{M}}$ so that w_{tr} and w'_{tr} correspond to runs of $\mathcal{A}_{\mathcal{M}}$, without modifying the path π corresponding to w_{zr}; in that case, we build a new TA $\mathcal{A}'_{\mathcal{M}}$ from $\mathcal{A}_{\mathcal{M}}$ accordingly, and add the guard g at step i, so as to separate w_{tr} and w'_{tr}. In that case, we can repeat the arguments above with the TA $\mathcal{A}'_{\mathcal{M}}$, as we assumed that no TA containing w_{zr} can be a model for Acc. By furthermore forbidding to re-use a reset combination already discarded (*e.g.* to return to $\mathcal{A}_{\mathcal{M}}$) and since there are only finitely many observations, and they are of finite length, this situation cannot occur indefinitely. Notice that if there is no way to separate two observations without coming back to a previously discarded reset combination, we can conclude to the invalidity of the root as previously.
  - If the modification inevitably leads to excluding w_{zr} from the runs of $\mathcal{A}_{\mathcal{M}}$, then there is a prefix w'_{zr} of w_{zr} (which is also a prefix of w_{tr}) after which no guard can

distinguish between (the suffixes of) w_t and w'_t. Applying the same arguments as above for the root, all leaves in the subtree after w_{tr} corresponding to all reset sequences for suffixes of w_t and w'_t are labelled with $+$ and $-$; the state corresponding to w'_{zr} must also be marked, and so $s_o = w_{zr}$ has a marked predecessor.

□

Example 5.4.14. *In the TOG of Example 5.4.11, both leaves in the right-hand-side subtree are invalid, which indicates that the clock* has to *be reset when reading the first a (at least when it occurs after a delay in* $(0, 1)$*).*

Remark 5.4.15. *We could propagate marks to suffixes to mark exactly all invalid states, but as the TOGs are always explored from the root it is unnecessary (a marked prefix will be reached beforehand) and constitutes a simple optimization.*

When marking a state of the TOG, we are able to aggregate a set of pairs of words that will characterize an invalidity.

Definition 5.4.16. *Let* Acc *be an acceptance function and* $\mathcal{M}$ *the TOG implementing* Acc*. When marking the observation states* s_o *of* $\mathcal{M}$*, construct the following set of pairs of timed words with resets* $\mathcal{W}_{s_o}$*:*

- *if* $|\mathsf{label}(s_o)| = 2$*,* $\mathcal{W}_{s_o} = \{(w_1, w_2)\}$ *for a pair* w_1, w_2 *in* $\mathsf{words}(s_o)$ *such that* $\mathsf{Acc}(w_1) \neq \mathsf{Acc}(w_2)$
- *when marking a state* s_o *with two marked successors* $s_o \cdot (z, a\top)$*,* $s_o \cdot (z, a\bot)$*,* $\mathcal{W}_{s_o} = \mathcal{W}_{s_o \cdot (z,a\top)} \cup \mathcal{W}_{s_o \cdot (z,a\bot)}$*.*

For unmarked successors of marked states, we use their predecessor characteristic sets. For an invalid observation state s_o*, we call* $\mathcal{W}_{s_o}$ *a* characteristic set *of its invalidity.*

One can see that a characteristic set of an invalidity is sufficient to characterize it, as it matches the definition of marking.

Using the marked states introduced in Prop. 5.4.13, the TOG will be used to detect invalid resets, while the purpose of the TLG is to construct possible guards for candidate timed automata. In order to be able to construct as many different hypotheses as possible, all (valid) combinations of resets must be considered, while the invalid ones should be pruned to avoid any overhead. For this we define a notion of *maximal* TLG, where no valid resets are pruned.

Definition 5.4.17. *A TLG $\mathcal{N}$ is said* maximal *if for any decision state $s_d = (s'_o, z, a) \in S_d$ and for $r \in \{\top, \bot\}$, there is $(s_d, r, s'_o \cdot (z, a, r)) \in E$ if, and only if, $s'_o \cdot (z, a, r)$ is valid.*

The algorithms proposed in the following sections will construct and maintain maximal and valid TLGs. Notice that as we suppose that the system we try to learn can be modelled by a RERA, ensuring the maximality of a TLG implementing Acc suffices to ensure that we will find a model (as there is a valid RERA model and we do not discard valid choices).

Example 5.4.18. *Consider the following set of observations over $\Sigma = \{a\}$:*

$$\begin{aligned} \mathsf{Acc}(\epsilon) &= - & \mathsf{Acc}((1.7, a)) &= - & \mathsf{Acc}((3.7, a)) &= - \\ & & \mathsf{Acc}((1.7, a)(1, a)) &= + & \mathsf{Acc}((3.7, a)(1.1, a)) &= + \\ & & \mathsf{Acc}((1.7, a)(1.1, a)) &= - & \mathsf{Acc}((3.9, a)(1.1, a)) &= - \end{aligned}$$

The TOG $\mathcal{M}$ implementing this acceptance function is displayed to the left of Figure 5.8.

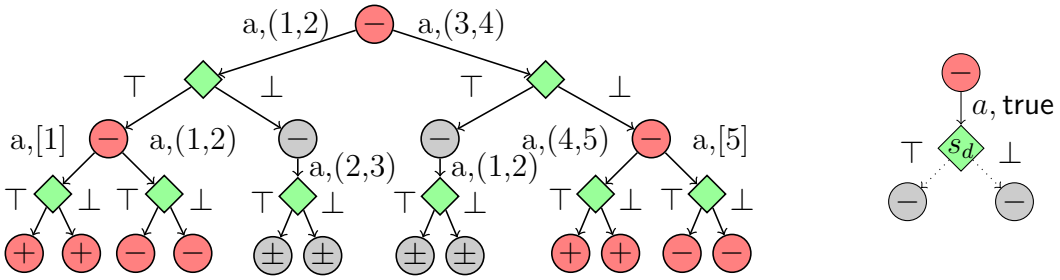


Figure 5.8: On the left: the TOG for Example 5.4.18, with its invalid nodes (in grey); on the right: a corresponding maximal TLG.

It can be noticed that the root of this TOG is valid, but that it is required for a model to distinguish between the cases where the first a occurs before time 2 and the cases where it occurs after time 3. For example, the TLG $\mathcal{N}$ displayed to the right of Figure 5.8 is maximal and implementing Acc*. Indeed, as there is no guard on the first a-transition, the decision state s_d leads only to invalid successors, which are thus pruned.*

Example 5.4.18 is a situation where a guarded word with resets (here ϵ) is not invalid in the TLG, but all its successors for a given action and guard (in this case a and true, corresponding to the decision state s_d in the TLG of Figure 5.8) are. In such situations, two different K-closed words with resets make the successors invalid, and a guard has to be added in the TLG to separate them. This corresponds to a situation where Acc displays

two different behaviours, one that require a clock reset to model, and another one that requires *not to reset* this very clock. Thus, in order to model Acc it is necessary to introduce a guard to separate these two behaviours (identified by two different invalidities).

5.5 Updating observation structures

The algorithms presented in this section are the operational core of our work. They are used to update the previously-defined data structures and ensure their good properties by membership queries and their processing, thus corresponding to line 4 of Algorithm 4 (Section 5.2.2).

We present the details of our algorithms as a set of subfunctions, connected as displayed on Figure 5.9. They work in three phases:

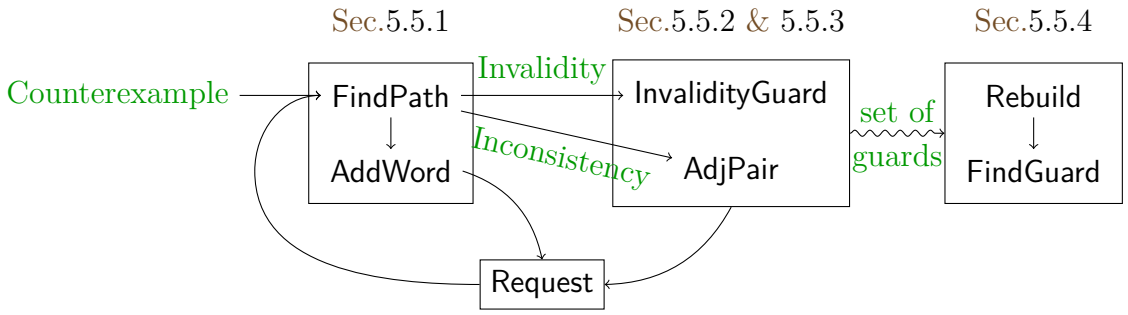


Figure 5.9: The structure of the update algorithms.

- First, starting from a new observation (obtained from a counterexample or via a membership query), FindPath (Algorithm 5) updates the structures (TLG and TOG) while keeping most of their good properties, except possibly consistency. It uses AddWord (Algorithm 6) on prefixes when new states have to be added in either tree. This may require calls to Request (Algorithm 7) on prefixes, which provides new observations (via membership queries) and thus new calls to FindPath on those prefixes when necessary.

  When an invalidity is detected in the TOG during this process, FindPath finds the root of the invalidity and prunes the corresponding subtree of the TLG as it is itself invalid (this approach is further discussed in Remark 5.5.1). These algorithms are presented in Section 5.5.1.

- When an inconsistency is detected by FindPath, a call to AdjPair (Algorithm 9) is made. This algorithm introduces a separating guard for the inconsistency, using a binary search timed words to find a fitting separation. Similarly, when a (valid) decision state is left without successors, InvalidityGuard (Algorithm 12) is called (through a preprocessing via Init-InvalidityGuard (Algorithm 8)). Its process is similar to AdjPair, but it reasons on *K-closed words with resets*, making it more complex.

  These algorithms request new observations through Request, possibly leading to new calls to FindPath. Both these algorithms add guards to symbolic set of guards that can be used, but do not modify directly the structures. AdjPair is described in Section 5.5.2 and InvalidityGuard in Section 5.5.3.

- Finally, after the cycle of new observations and guard storing finishes, the relevant part of the models are rebuilt using these new guards by Rebuild (Algorithm 15) using calls to FindGuard (Algorithm 16) to find relevant guards. These algorithms are presented in Section 5.5.4.

5.5.1 Adding a new observation

Adding a new observation in the TLG $\mathcal{N}$ and in the TOG $\mathcal{M}$ is performed by function FindPath (Algorithm 5): this function recursively descends into $\mathcal{M}$, following the input timed word w_t and exploring both resetting and non-resetting edges. It is called as $\mathsf{FindPath}(\epsilon, w_t, \mathbf{0}, s_\epsilon)$.

- If the descent gets stuck by lack of new successors before exhausting w_t, then new nodes have to be added in the TOG (and possibly corresponding nodes in the TLG). This is performed by AddWord (Algorithm 6), making new membership queries via Request (Algorithm 7) if needed to ensure prefix-closure. Each call to AddWord adds two observation states (corresponding to the two reset choices) in the TOG and possibly two observation states in the TLG (unless they already exist), as well as the intermediary decision states.

- If the descent exhausts w_t, then it reaches a node corresponding to the K-closed timed word containing w_t, whose words (and label, as a side-effect) function can be updated (as well as those of the associated node in the TLG). Invalidities and inconsistencies can be detected and resolved, respectively by scheduling calls to InvalidityGuard (Algorithm 12) and AdjPair (Algorithm 9). These two functions will be

explained in the next sections. Notice that they are not run until FindPath terminates; this is precisely to simplify the termination proof of FindPath.

Remark 5.5.1. *Notice the way FindPath handles invalidities: it finds the root of the invalidity and marks it in the TOG. In the TLG the corresponding observation state is* pruned. *This approach is somewhat lazy as it chooses to prune a node that covers potentially more than the invalidity (and possibly valid observations), without reintroducing ones that cover its valid observations. This is made to avoid both the computational cost and the blow-up of the model (as it would require a great number of new states). This pruning is not problematic as long as any decision states has at least one un-pruned successor. When all successors are pruned, we use algorithm InvalidityGuard (Algorithm 12) to introduce a guard and create new decision states that have valid successors, as discussed in Section 5.5.3.*

Function AddWord recursively creates the nodes in the TOG and TLG as needed. It launches new calls to Request for each intermediary word, which in turn calls FindPath in order to add the new information in all corresponding nodes and checks for inconsistencies and invalidities.

The following statements express soundness of our algorithms. They ensure that the good properties of the structures are preserved by the call to FindPath. Note that we do not consider the consequences of the scheduled calls to InvalidityGuard and AdjPair in the following properties. They will be discussed in the following subsections.

Proposition 5.5.2 ensures that the pruning made during the execution of the FindPath algorithm prunes exactly the invalid states of the TLG (ensuring maximality) and that the corresponding states of the TOG are marked. Proposition 5.5.3 states that FindPath keeps the implementation properties of the TOG and TLG with respect to the updated set of observations. Notice that these algorithms do not handle consistency, but only schedule a call to AdjPair that will handle it later.

We note $\mathsf{NewPref}_{\mathcal{M}}(w_t)$ the set of prefixes of a timed word that is not covered by the TOG $\mathcal{M}$, *i.e.* for $w_t = (t_i, a_i)_{1\leq i\leq n}$, $\mathsf{NewPref}_{\mathcal{M}}(w_t) = \{w'_t = (t_i, a_i)_{1\leq i\leq j} \mid j < n \wedge w'_t \dashv S_o^{\mathcal{M}}\}$ where a timed word is compatible with a set of zone word with resets if and only if it is compatible with one of its elements.

Proposition 5.5.2. *Consider an acceptance function Acc, a maximal TLG $\mathcal{N}$, a marked TOG $\mathcal{M}$ implementing Acc, such that no invalid states w.r.t. Acc can be reached in $\mathcal{N}$, and a new observation $w_t \notin \mathsf{Dom}(\mathsf{Acc})$. Let Acc' be such that $\mathsf{Dom}(\mathsf{Acc}') = \mathsf{Dom}(\mathsf{Acc}) \cup \{w_t\} \cup \mathsf{NewPref}_{\mathcal{M}}(w_t)$ and for all $w'_t \in \mathsf{Dom}(\mathsf{Acc})$, $\mathsf{Acc}'(w'_t) = \mathsf{Acc}(w'_t)$. Calling $\mathsf{FindPath}(\epsilon, w_t, \mathbf{0}, s_\epsilon)$*

Algorithm 5: Adding a new observed timed word $p_t \cdot w_t$ in $\mathcal{N}$ and $\mathcal{M}$

```
1  def FindPath (p_t, w_t, v, s_o):
       Input: p_t: prefix timed word; w_t: suffix timed word; v: valuation after p_t;
              s_o: state of TOG after p_t
2      if w_t = ε then                                          // end of word reached
3          add p_t to words(s_o) and words(α(s_o))
4          if |label(s_o)| = 2 then                             // invalidity detected in TOG
5              s_o.invalid := true
6              s_d := parent(s_o)
7              while both successors of s_d are invalid do      // seek invalidity root
8                  s_o := parent(s_d)
9                  s_o.invalid := True
10                 s_d := parent(s_o)
11             delete transition entering α(s_o) and subtree rooted at α(s_o) from M
12             schedule a call to Init-InvalidityGuard(s_o)
13         else
14             if |label(α(s_o))| = 2 then                      // inconsistency detected in TLG
15                 pick w_t and w'_t in words(α(s_o)) s.t. Acc(w_t) ≠ Acc(w'_t)
16                 schedule a call to AdjPair(w_t ⊗ (resets(s_o)), w'_t ⊗ (resets(s_o)))
17     else                                                     // w_t not empty
18         (t, a) · w'_t := w_t
19         if there exist z and s_d s.t. (s_o, (z, a), s_d) ∈ E_M and v + t ∈ z then
20             for (s_d, r, s'_o) ∈ E_M do
21                 FindPath(p_t · (t, a), w'_t, (v + t)_[r←0], s'_o)
22         else
23             AddWord(p_t, w_t, v, s_o)
```

Algorithm 6: Extending $\mathcal{M}$ and $\mathcal{N}$ to include a new timed word

def *AddWord* $(p_t,\ w_t,\ v,\ s_o)$**:**
  Input: p_t: prefix timed word; w_t: suffix timed word; v: valuation after p_t; s_o: state of TOG after p_t
  $(t, a) \cdot w'_t := w_t$
  create state $s_d := (s_o, Kz(v+t), a)$ in $S_d^{\mathcal{M}}$ `// Handling` $\mathcal{M}$
  create states $s'_o := s_o.(Kz(v+t), a, \top)$ and $s''_o := s_o.(Kz(v+t), a, \bot)$ in $S_o^{\mathcal{M}}$
  create transitions $(s_o, (Kz(v+t), a), s_d)$, $(s_d, \top, s'_o)$ and $(s_d, \bot, s''_o)$ in $E^{\mathcal{M}}$
  if $\exists s_c = (\alpha(s_o), g, a) \in S_d^{\mathcal{N}}$ *such that* $v + t \in g$ **then** `// Handling` $\mathcal{N}$
    let s'_l, s''_l in $S_o^{\mathcal{N}}$ such that $(s_c, \top, s'_l)$ and $(s_c, \bot, s''_l)$ in $E^{\mathcal{N}}$
  else
    create state $s_c := (\alpha(s_o), \mathsf{true}, a)$ in $S_d^{\mathcal{N}}$
    create states $s'_l := \alpha(s_o).(\mathsf{true}, a, \top)$ and $s''_l := \alpha(s_o).(\mathsf{true}, a, \bot)$ in $S_l^{\mathcal{N}}$
    create transitions $(\alpha(s_o), (\mathsf{true}, a), s_c)$, $(s_c, \top, s'_l)$ and $(s_c, \bot, s''_l)$ in $E^{\mathcal{N}}$
    define $\alpha(s'_o) := s'_l$ and $\alpha(s''_o) := s''_l$
  Request $(p_t \cdot (t, a))$ `// get status of` $p_t \cdot (t, a)$
  add $p_t \cdot (t, a)$ to $\mathsf{words}(s'_o)$, $\mathsf{words}(s''_o)$, $\mathsf{words}(s'_l)$, and $\mathsf{words}(s''_l)$
  if $|\mathit{label}(s'_l)| = 2$ **then** `// inconsistency detected at node` s'_l `(and` s''_l`)`
    pick w_t and w'_t in $\mathsf{words}(s'_l)$ s.t. $\mathsf{Acc}(w_t) \neq \mathsf{Acc}(w'_t)$
    schedule a call to $\mathsf{AdjPair}(w_t \otimes (\mathsf{resets}(s_o)), w'_t \otimes (\mathsf{resets}(s_o)))$
  if $w'_t \neq \epsilon$ **then** `// proceed recursively with the suffix`
    for $(s_d, r, s'_o) \in E^{\mathcal{M}}$ **do**
      $\mathsf{AddWord}(p_t \cdot (t, a), w'_t, (v+t)_{[r \leftarrow 0]}, s'_o)$

Algorithm 7: Requesting an observation.

def *Request* (w_t)**:**
  Input: w_t: timed word
  if $w_t \in \mathit{Dom}(\mathit{Acc})$ **then**
    return $\mathsf{Acc}(w_t)$
  else
    Make a membership query on w_t and set its result o to $\mathsf{Acc}(w_t)$
    $\mathsf{FindPath}(\epsilon, w_t, \mathbf{0}, s_\epsilon)$
    return o

Algorithm 8: Pruning $\mathcal{N}$ after detecting an invalid timed word with resets.

```
1 def Init-InvalidityGuard:
      Input: s_o an observation state corresponding to the root of an invalidity in M
2     Let s_d be the parent of s_o
3     if |{(s_d, _, _) ∈ E_N}| = 0 then
4         Let W_1 be the characteristic set of s_o and W_2 the characteristic set of the
            other pruned child of s_d.
5         InvalidityGuard(W_1, W_2, s_d)
```

modifies $\mathcal{M}$ and $\mathcal{N}$ in such a way that $\mathcal{M}$ becomes marked w.r.t. Acc', $\mathcal{N}$ becomes maximal w.r.t. Acc' and no invalid states can be reached in $\mathcal{N}$.

Proof. Invalidity is detected along the calls to FindPath, and the propagation of the invalid tag follows the definition of marks for all ascendant states. No descendant are tagged, but this does not matter as they cannot be reached without passing by marked states as $\mathcal{M}$ is a tree. A call to SearchPrune is then made, that targets exactly the root of the invalid subtree that has been detected, and prune it in $\mathcal{N}$. As this is made for all detected invalidities and the subtrees are detected, when the procedure terminates, no language state invalid because of an invalidity detected in FindPath can be reached. Furthermore only the invalid subtree is suppressed, hence no valid states are made unreachable (as by definition all descendant of invalid states are invalid).

To conclude, it suffices to notice that every new membership query gives rise to a corresponding call to FindPath, leaving no invalidities undetected. □

Proposition 5.5.3. *Starting from an acceptance function Acc, a TLG $\mathcal{N}$ and a TOG $\mathcal{M}$ implementing the valid part of Acc and a timed word $w_t \notin \mathsf{Dom}(\mathsf{Acc})$, we consider Acc' be such that $\mathsf{Dom}(\mathsf{Acc}') = \mathsf{Dom}(\mathsf{Acc}) \cup \{w_t\} \cup \mathsf{NewPref}_{\mathcal{M}}(w_t)$ and for all $w'_t \in \mathsf{Dom}(\mathsf{Acc})$, $\mathsf{Acc}'(w'_t) = \mathsf{Acc}(w'_t)$.*

A call to $\mathsf{FindPath}(\epsilon, w_t, \mathbf{0}, s_\epsilon)$ with s_ϵ the root of $\mathcal{M}$ modifies $\mathcal{M}$ and $\mathcal{N}$ in such a way that they implement the valid part of Acc'.

Proof. First of all, a call to FindPath terminates, as recursive calls are in finite number and on words of strictly decreasing length, there is at most one call to AddWord or SearchPrune from FindPath along each explored path and recursive calls to AddWord are in finite number and on words of strictly lower length. Thus, there is a finite number of calls to Request, and if these calls make calls to FindPath for new observations, the timed words given as arguments are of strictly lesser length.

We now prove the rest of the property by induction on the calls to both FindPath and AddWord. We use the following induction hypothesis: A call to AddWord/FindPath creates subgraphs of $\mathcal{N}$ and $\mathcal{M}$ that implement the restriction of Acc' to the subset of its domain belonging to s_o the observation state given in argument.

- Basic case for FindPath. Here we have $w_t = \epsilon$. In this case the complete word to add was read before along the path. The past p_t storing that word is added to words of both s_o and $\alpha(s_o)$. Hence the subgraphs are complete with respect to the new observation (by adding delays and actions no other reachable states can correspond to that same word) and every new observation requested during the recursive calls (as they do not reach it). No other states are created, hence the hypothesis on the initial structures suffice to conclude that the subgraphs are complete and well-grounded with respect to the valid part of Acc', as by Proposition 5.5.2 only invalid states w.r.t. Acc' are pruned.

- Basic case for AddWord. We first discuss the properties of $\mathcal{M}$. As AddWord is never called on empty words, we have $w_t = (t, a)$. The call to AddWord adds a new decision state $s_d = (s_o, a, Kz(v + t))$ and two new language nodes s'_o and s''_o corresponding to the effect of resetting or not c_a after the action. As AddWord was called in FindPath, we know that no successor covers (t, a) in s_o. As we are in the base case, words of s'_o and s''_o are augmented with the observation $p_t \cdot (t, a)$, making them complete with respect to Acc' (as no other timed word in $\mathsf{Dom}(\mathsf{Acc}'')$ reaches this state) and well-grounded. The edges constructed by this call agree with the definition, and by the hypothesis on the initial observation graph, the other successors of s_o are complete (except with respect to the new word, but their sequence of letters and K-closed zones do not match) and well grounded, hence in all cases we obtain a subgraph of $\mathcal{M}$ that is complete (as s'_l and s''_l have no successors) and well grounded.

  We now discuss the properties of $\mathcal{N}$. There are two possibilities. Either a decision state covering (t, a) from $\alpha(s_o)$ exists, in which case the behaviour of AddWord mimics the one of FindPath, and we can conclude using the previous point, or there is no such decision state. In this case, by definition of a TLG, we know that no decision state for letter a exist as successors of $\alpha(s_o)$. Hence the new decision and observation states created in $\mathcal{N}$ correspond to the definition of a TLG. Furthermore, the observation states are well-grounded, as $p_t.(t, a)$ is added to their set of words, and this, together with the initial hypothesis on $\mathcal{N}$ implementing Acc, ensures that

the subgraph is complete w.r.t. Acc'.

- Inductive case of FindPath. We consider that $w_t = (t, a).w'_t$. Thus we enter the **else** part in line 2. If the **else** case is called in line 22, we only make a call to AddWord on w_t hence by induction hypothesis, we have the desired properties. Else, as there is only one guard such that w_t can go through that guard (by unicity of the K-closed set containing a timed word with resets), all successors satisfying a prefix of w_t are reached by the recursive calls and by induction hypothesis they lead to implementing subgraphs for both the TOG and TLG. Furthermore, other successors of s_o (resp. $\alpha(s_o)$) constitute, by the hypothesis on the initial structure, a subgraph implementing Acc (except that the guard g of the considered transition is not covered), except for the new word that may not be covered. But by unicity, they cannot correspond to paths satisfying the new word and thus we have our properties.

- Inductive case for AddWord. This case works exactly as the base case, except that calls to Request ensure that the new states have non-empty words, and the completeness with respect to w_t is ensured by the induction hypothesis.

□

5.5.2 Dealing with inconsistency

An inconsistency arises when a language state of the TLG contains both accepting and non-accepting observations. It entails that a guard must be added somewhere in the structure in order to separate these observations.

For this we search for a pair of *adjacent* timed words with resets, which intuitively identify the boundary between accepting and non-accepting behaviours. We then build a finite set of *differences* between adjacent timed words with resets, each of which corresponds to a possible guard. This procedure is described in the AdjPair algorithm (Algorithm 9).

We use K-equivalence to define the notion of adjacency. Intuitively adjacent timed words with resets have the same projection on actions and resets, and their valuations either are K-equivalent, or materialize a boundary between the accepted and non-accepted words.

As these boundaries rely on clock values, we define adjacency on abstract runs and transfer it to timed words with resets using the sig bijection.

Definition 5.5.4. *For two abstract runs sharing the same length, the same actions and resets* $\sigma = (v_{i-1} \xrightarrow{t_i,a_i,r_i} v_i)_{1\leq i\leq n}$ *and* $\sigma' = (v'_{i-1} \xrightarrow{t'_i,a_i,r_i} v'_i)_{1\leq i\leq n}$*, we say that* σ is adjacent to σ' *when for all* $i \in [1, n]$ *and* $c \in C$*:*

- *if* $v_{i-1}(c) + t_i \in \mathbb{N}$ *then* $|(v_{i-1}(c) + t_i) - (v'_{i-1}(c) + t'_i)| < 1$,
- *otherwise,* $v_{i-1}(c) + t_i \approx_K v'_{i-1}(c) + t'_i$.

A timed word with resets w_{tr} *is said adjacent to a timed word with resets* w'_{tr} *when* $\mathsf{sig}^{-1}_{behav}(w_{tr})$ *is adjacent to* $\mathsf{sig}^{-1}_{behav}(w'_{tr})$.

Notice that adjacency is not a symmetric relation. However, we will sometimes say that an ordered pair (w_{tr}, w'_{tr}) is adjacent to mean that w_{tr} is adjacent to w'_{tr}. We will often abuse the term and call (w_{tr}, w'_{tr}) an adjacent pair.

We use adjacency to identify *differences* between the timed words with resets as possible new guards that resolve the inconsistency.

Definition 5.5.5. *Given a pair of adjacent abstract runs sharing the same length, the same actions and resets* $\sigma = (v_{i-1} \xrightarrow{t_i,a_i,r_i} v_i)_{1\leq i\leq n}$ *and* $\sigma' = (v'_{i-1} \xrightarrow{t'_i,a_i,r_i} v'_i)_{1\leq i\leq n}$*, their* difference *noted* $\mathsf{diff}(\sigma, \sigma')$ *is the set of quadruples defined as follows: if for a clock* $c \in C$*,* $v_{i-1}(c) + t_i = k \in \mathbb{N}$*, then if* $v'_{i-1}(c) + t'_i < k$*,* $(i, c, \geq, k) \in \mathsf{diff}(\sigma, \sigma')$*, and if* $v'_{i-1}(c) + t'_i > k$*,* $(i, c, \leq, k) \in \mathsf{diff}(\sigma, \sigma')$.

For a pair of adjacent timed words with resets (w_{tr}, w'_{tr}) *we define* $\mathsf{diff}(w_{tr}, w'_{tr}) = \mathsf{diff}(\mathsf{sig}^{-1}_{behav}(w_{tr}), \mathsf{sig}^{-1}_{behav}(w'_{tr}))$.

Using these definitions, we can derive from two adjacent abstract runs or timed words with resets a set of candidates to make a new guard, that we call *consistency guards*. These guards will be used to reconstruct a language graph that is consistent with respect to the adjacent pair.

Definition 5.5.6. *For an adjacent pair* (w_{tr}, w'_{tr})*, a clock constraint* $c \prec k$ *with* $\prec \in \{<, \leq, \geq, >\}$*,* $(i, c, \prec, k) \in \mathsf{diff}(w_{tr}, w'_{tr})$ *is a* consistency guard *if there is no* $(j, c, \prec', l) \in \mathsf{diff}(w_{tr}, w'_{tr})$ *such that* $j < i$ *or* $j = i$ *and* $l < k$.

The consistency guards are taken on the first difference, so as to ensure that they cannot be overwritten later (as there are no guards that can separate the pair before the consistency guard), and to avoid large constants as much as possible. Notice that we cannot always infer a unique guard from an adjacent pair, as multiple clocks can be different at the same time.

Remark 5.5.7. *We choose to introduce a bias on guards by choosing a consistency guard with a constant as small as possible. In practice, one could replace this bias by another (expert-designed) one more fitted to the application. Our algorithms do not depend on it.*

We define a set $\mathcal{G}_{Cons}$ of consistency guards that will be used to create the guards in the updated TLG afterward.

AdjPair makes membership queries on linear combinations of the two initial observations to perform a binary search until the clock values of the pair have less than 1 time unit of distance. Then it forces every non-K-equivalent pair of clock values to have one of its elements be an integer with more linear combinations. Finally, in order to ensure that only one of the two timed words with resets have such integer distinctions, it compares them with their mean. This gives an adjacent pair, from which consistency guards are extracted and added to $\mathcal{G}_{Cons}$.

Proposition 5.5.8. *The AdjPair algorithm constructs an adjacent pair using at most $\mathcal{O}(m|\Sigma|\log(K))$ membership queries.*

Proof. We refer the reader to the proof of Theorem 5.8 in [GJ08], a long version of [GJP06] appearing in O.Grinchtein's thesis [Gri08]. □

The AdjPair algorithm proposes a guard that separates, as one would expect, the accepting and non-accepting observations given as an argument, as well as the accepting and non-accepting observations made during the algorithm execution.

Proposition 5.5.9. *Let w^1_{tr}, w^2_{tr} be two timed words with resets sharing the same length n, the same actions and resets such that $\mathsf{Acc}(\mathsf{trace}(w^1_{tr})) = +$ and $\mathsf{Acc}(\mathsf{trace}(w^2_{tr})) = -$. Consider $\mathcal{W}_1$ the set of accepting observations requested during the call to $\mathsf{AdjPair}(w^1_{tr}, w^2_{tr})$ and $\mathcal{W}_2$ the set of negative ones. By taking a consistency guard $(i, c, \prec, k)$ for $\prec \in \{<, \leq, =, \geq, >\}$, at depth $1 \leq i \leq n$ in the set of guards added to $\mathcal{G}_{Cons}$ by $\mathsf{AdjPair}(w^1_{tr}, w^2_{tr})$, we have the following:*

There is a guard g being either $x \prec k$, or its complement such that, for any $w^1 \in \mathcal{W}_1 \cup \{\mathsf{trace}(w^1_{tr})\}$ (resp. $w^2 \in \mathcal{W}_2 \cup \{\mathsf{trace}(w^2_{tr})\}$), noting $\mathsf{sig}^{-1}_{\mathsf{behav}}(w^1 \otimes \mathsf{resets}(w^1_{tr})) = (v_{j-1} \xrightarrow{t_j} v_{j-1}+t_j \xrightarrow{(a_j,r_j)} v_j)_{1\leq j\leq n}$ (resp. $\mathsf{sig}^{-1}_{\mathsf{behav}}(w^2 \otimes \mathsf{resets}(w^2_{tr})) = (v'_{j-1} \xrightarrow{t'_j} v'_{j-1}+t'_j \xrightarrow{(a_j,r_j)} v'_j)_{1\leq j\leq n}$), we get $v_{i-1} + t_i \in g$ and $v'_{i-1} + t'_i \notin g$.

Proof. First, notice that every call to Request is made on (the trace of) a linear combination of w^1_{tr} and w^2_{tr}. This can be seen by induction on the **while** and **for** loops: at the beginning

Algorithm 9: Adding to $\mathcal{G}_{Cons}$ consistency guards corresponding to an inconsistency.

AdjPair
Input: w_{tr}, w'_{tr} two timed words with resets of same length n, with same actions and resets, such that $\mathsf{Acc}(\mathsf{trace}(w_{tr})) = +$ and $\mathsf{Acc}(\mathsf{trace}(w'_{tr})) = -$.
Output: $\mathcal{G}_{Cons}$ a modified set of adjacent pairs
Note $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_{tr}) = (v_{i-1} \xrightarrow{(t_i,a_i,r_i)} v_i)_{1\leq i\leq n}$ and $\mathsf{sig}^{-1}_{\mathsf{behav}}(w'_{tr}) = (v'_{i-1} \xrightarrow{(t'_i,a_i,r_i)} v'_i)_{1\leq i\leq n}$.
while *not*$(\forall i \in [0,n], \forall c \in C,\ |v_i(c) + t_i - (v'_i(c) + t'_i)| < 1 \vee ((v_i(c) + t_i > K) \wedge (v'_i(c) + t'_i > K)))$ **do**
  $w''_{tr} := 0.5w_{tr} + 0.5w'_{tr}$
  if *Request*(*obs*(w''_{tr})) **then**
    $w_{tr} := w''_{tr}$
  else
    $w'_{tr} := w''_{tr}$
for $i \in [0,n]$, $c \in C$ **do**
  if $\lfloor v_i(c) + t_i \rfloor \neq \lfloor v'_i(c) + t'_i \rfloor \wedge v_i(c) + t_i \notin \mathbb{N} \wedge v'_i(c) + t'_i \notin \mathbb{N}$ **then**
    if $v_i(c) + t_i < v'_i(c) + t'_i$ **then**
      $\lambda := (\lfloor v'_i(c) + t'_i \rfloor - (v_i(c) + t_i))/(v'_i(c) + t'_i - (v_i(c) + t_i))$
    else
      $\lambda := (\lfloor v_i(c) + t_i \rfloor - (v'_i(c) + t'_i))/(v_i(c) + t_i - (v'_i(c) + t'_i))$
    $w''_{tr} := \lambda.w_{tr} + (1-\lambda).w'_{tr}$ **if** *Request*(*trace*$(w''_{tr})) = +$ **then**
      $w_{tr} := w''_{tr}$
    else
      $w'_{tr} := w''_{tr}$
$w''_{tr} = 0.5w_{tr} + 0.5w'_{tr}$
if *Request*(*trace*$(w''_{tr})) = +$ **then**
  Add to $\mathcal{G}_{Cons}$ the consistency guards from $\mathsf{diff}(w'_{tr}, w''_{tr})$
else
  Add to $\mathcal{G}_{Cons}$ the consistency guards from $\mathsf{diff}(w_{tr}, w''_{tr})$

$w_{tr} = w^1_{tr}$ and $w'_{tr} = w^2_{tr}$, and w''_{tr} is clearly constructed as a linear combination of both. Afterward, either w_{tr} or w'_{tr} becomes w''_{tr}. As linear combinations of linear combinations of w^1_{tr} and w^2_{tr} are linear combinations of w^1_{tr} and w^2_{tr}, we have our result.

By Prop. 5.3.8 we know that the clock values on the abstract run of these timed words with resets are linear combinations of the clock values of $\mathsf{sig}^{-1}_{\mathsf{behav}}(w^1_{tr})$ and $\mathsf{sig}^{-1}_{\mathsf{behav}}(w^2_{tr})$. This tells us that for a given depth $1 \leq j \leq n$, the valuations $v_{j-1} + t_j$ (*i.e.* the valuations before the j-th transition) of the different words are on a segment between the corresponding valuations of w^1_{tr} and w^2_{tr}.

We will now show that the following property is an invariant of the **while** and **for** loops: for any depth $1 \leq j \leq n$, the valuations before the j-th transition of any positive observation's abstract run are between the ones of $\mathsf{sig}^{-1}_{\mathsf{behav}}(w^1_{tr})$ and $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_{tr})$ (w_{tr} being – as defined in the algorithm – the current positive timed word with resets). Similarly, the valuation of any negative observation's abstract run are between the ones of $\mathsf{sig}^{-1}_{\mathsf{behav}}(w^2_{tr})$ and $\mathsf{sig}^{-1}_{\mathsf{behav}}(w'_{tr})$.

At the beginning of the algorithm, this is clear, as $w^1_{tr} = w_{tr}$, $w^2_{tr} = w'_{tr}$ and no other Request has been made yet. Now suppose that it is true before a (**for** or **while**) loop iteration. w''_{tr} is a linear combination of w_{tr} and w'_{tr}, so as previously, by Prop. 5.3.8, the valuation reached in $\mathsf{sig}^{-1}_{\mathsf{behav}}(w''_{tr})$ before the j-th transition is in the segment between the ones of $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_{tr})$ and $\mathsf{sig}^{-1}_{\mathsf{behav}}(w'_{tr})$. As w''_{tr} takes the name of the member of $\{w_{tr}, w'_{tr}\}$ it shares its acceptance with, the property is preserved at the end of the iteration.

Finally, by definition of a consistency guard at depth i, it separates the valuations of the adjacent pair before their i-th transition. As the adjacent pair used is either (w_{tr}, w''_{tr}) or (w'_{tr}, w''_{tr}) according to w''_{tr} acceptance status, as w''_{tr} is a linear combination of w_{tr} and w'_{tr} and by using the previously proved facts, we have our result. □

5.5.3 Dealing with decision states with no successors

As explained earlier, a marked observation state in the TOG indicates an invalidity: it points to a combination of resets that is unable to explain a precise set of observations. Invalidity is first simply dealt with by pruning the invalid parts of the TLG, as done in line 11 of FindPath (Algorithm 5). But sometimes a challenge may arise, as exemplified in Example 5.4.18: it can occur that *all* successors of a decision state of the TLG following a *valid* observation state are pruned, due to invalidities; this is because the guard leading to that decision state is not strong enough, and it has to be reinforced so as to split

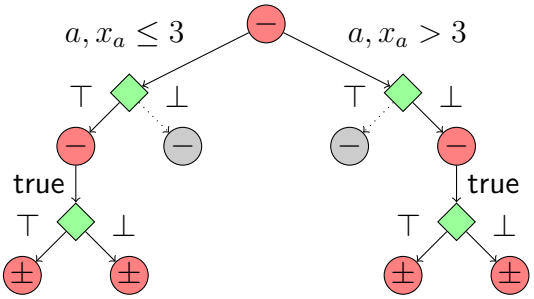


Figure 5.10: TLG of Figure 5.8 after the addition of a guard splitting the two invalidities. Invalid pruned nodes in grey.

the decision state of the TLG into several decision states, as explained by the following example.

Example 5.5.10. *In Example 5.4.18, all the successors of the decision state* $(a, \top)$ *of the TLG were pruned (see right-hand side of Figure 5.8). It is easily seen from the TOG (left of Figure 5.8) that this is due to the TLG merging the behaviours with* $c_a \in (1, 2)$ *and those with* $c_a \in (3, 4)$*.*

Introducing any guard on c_a *separating the behaviours for* $c_a < 2$ *and* $c_a > 3$ *would create two decision states which would each have one valid successor. For example, the TLG after the introduction of the guard* $c_a > 3$ *is presented in Figure 5.10. Note that the valid leaves are inconsistent, as could be expected from the TOG. We would then have to call* AdjPair *as described in Section 5.5.2.*

Remark 5.5.11. *In such situation, the introduction of a guard (called* validity guard*) is the only possible solution. Indeed, pruning a valid node of the TLG would contradict the definition of a (maximal) implementation of* Acc*, and leaving a valid node without successors does not allow to use it to construct an hypothesis.*

Remark 5.5.12. *This situation is a consequence of our somewhat lazy handling of invalidities. Indeed, when detecting an invalidity in the TOG, one could decide to "cut" it out of the TLG by introducing guards modelling it and splitting the corresponding node. However, this would require isolating a* K*-closed zone from its complement, which in turn requires introducing a large number of subtrees, since the complement of a* K*-closed zone can in general only be represented by a disjunction of a large number of guards.*

On the contrary, we chose to directly prune the subtree, and only split nodes (by introducing a validity guard as above) once we have pruned all possible resets. This

approach remains correct, as the TLG remains maximal and we add separations as soon as a decision state is left without successors.

The constant in the guard that is introduced is taken in the set of integer constants separating the two invalid K-closed words with resets in the TOG. This is formalised by the notion of *validity guard*, akin to the consistency guard introduced previously.

Definition 5.5.13. *Consider two K-closed words with resets $w^1_{zr} = (z^1_i, a_i, r_i)_{1\leq i\leq n}$ and $w^2_{zr} = (z^2_i, a_i, r_i)_{1\leq i\leq n}$ sharing the same length, actions and resets. Then $c \prec k$, with $c \in C$, $\prec \in \{<, \leq\}$, and $k \in \mathbb{N}$, is called a* validity guard *at depth $1 \leq i \leq n$ if, and only if, for all $j > i$, it holds $z^1_j \approx_K z^2_j$ and $c \prec k$ separates z^1_i and z^2_i (*i.e.*, either $z^1_i \in c \prec k$ and $z^2_i \notin c \prec k$, or the converse). We then say that $(c \prec k, i, w^1_{zr}, w^2_{zr})$ is a validity guard.*

We extend the definition of validity guard for sets of K-closed words by saying that $(c \prec k, i, \mathcal{W}^1_{zr}, \mathcal{W}^2_{zr})$ is a validity guard if there is $w^1_{zr} \in \mathcal{W}^1_{zr}$ and $w^2_{zr} \in \mathcal{W}^2_{zr}$ such that $(c \prec k, i, w^1_{zr}, w^2_{zr})$ is a validity guard and for all pairs in $\mathcal{W}^1_{zr} \times \mathcal{W}^2_{zr}$, $c \prec k$ separates their zones at depth i.

Remark 5.5.14. *The validity guards are introduced* as deep as possible *as opposed to consistency guards that are introduced high in the tree. The reason for this is that invalidity is not primarily used to introduce guards to the model. In addition, it can be harder to achieve a precise guard separating K-closed words due to their structure and the difficulty to reproduce invalid K-closed words with their characteristic sets, which is discussed in the rest of this subsection.*

We define validity guards for sets of words because we sometimes want to enforce a validity guard deduced from a pair upon other pairs (as we have more insight on its utility) while it is not formally a validity guard for the other pairs (*i.e.* it separates them but does not meet all criteria).

As for inconsistencies, it is desirable to introduce guards that correspond as closely as possible to the behaviours of the teacher's model. To this aim, we try to identify relevant validity guards by making new membership queries that "replicate" an invalidity by shifting its characteristic set (the set of pairs of timed words with resets witnessing the invalidity, see Definition 5.4.16).

Example 5.5.15. *In Examples 5.4.18 and 5.5.10, several guards could have been introduced. In order to identify a good candidate, a solution is to replicate the characteristic set of (one of the) invalidities with $x_a \in (2, 3)$. For this, membership queries are made for*

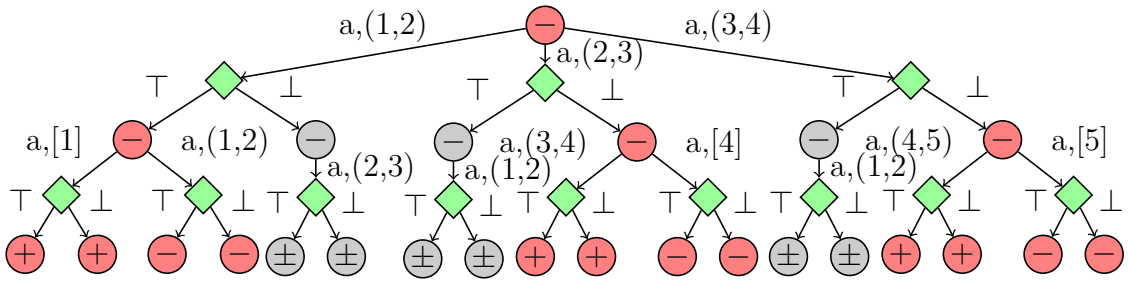


Figure 5.11: TOG after the creation of a new branch, obtained by shifting the observations in the $(3, 4)$ branch in the $(2, 3)$ time interval.

observations stemming from the characteristic set of one of the invalidities. For example, we can shift the observations corresponding to the subtree with $x_a \in (3, 4)$*, which may result in the following new observations:*

$$\mathsf{Acc}((2.7, a)) = - \qquad \mathsf{Acc}((2.7, a)(1.1, a)) = + \qquad \mathsf{Acc}((2.9, a)(1.1, a)) = -$$

We end up with the TOG of Figure 5.11.

Since the subtrees of the TOG for guards $x_a \in (1, 2)$ *and* $x_a \in (2, 3)$ *are different, we know that there is a change of behaviour at* $x_a = 2$*. Here, we cannot determine which behaviour prevails for* $x_a = 2$*, as explained later in Remark 5.5.16.*

Remember that the characteristic set of an invalidity at depth n is made of pairs of timed words with resets (w^1_{tr}, w^2_{tr}), each pair sharing the same length at least equal to n, the same actions and reset sequence $\mathsf{resets}(w^1_{tr}) = \mathsf{resets}(w^2_{tr})$, but diverging on the acceptance of their observations: $\mathsf{Acc}(\mathsf{trace}(w^1_{tr})) \neq \mathsf{Acc}(\mathsf{trace}(w^2_{tr}))$.

Introducing a guard precisely separating two invalidities is more complex than for inconsistencies for three reasons:

(i) first, any attempt to detect the span of an invalidity of depth n should be able to replicate its characteristic set targeting a new K-closed word with resets of length n. Moreover, replication should not modify the K-equivalence structure of the characteristic set even beyond the invalidity at depth n: valuations reached by replicas should be K-equivalent if and only if they were initially.

(ii) second, the reset sequence r cannot explain the observations of the characteristic set, but a yet unknown one r' could. When replicating invalidities we should thus consider all pairs of timed words with resets $(\mathsf{trace}(w^1_{tr}) \otimes r', \mathsf{trace}(w^2_{tr}) \otimes r')$ for any

sequence of resets r' and also try not to modify their K-equivalence structure.

(iii) third, as for inconsistencies, it is important that the guards introduced by the algorithm separate *all* the K-closed words corresponding to each invalidity encountered during the execution. Yet, K-closed zones have a more complex structure than valuations. Notably, K-closed zones can have different forms, being open or closed in different dimensions (*i.e.* with constraints of the form $c = n$ or $n < c < n + 1$), which will make preserving this property more challenging.

Remark 5.5.16. *The second point—having to preserve K-equivalence for other sequences of resets—induces that it is highly difficult to replicate the characteristic sets in K-closed words that have finer constraints than the initial K-closed word (*i.e. *equality constraint replacing a pair of inequalities). In Example 5.5.15 for instance, determining the behaviour for $c_a = 2$ requires to collapse $(2.7, a)(1.1, a)$ and $(2.9, a)(1.1, a)$ which both transform into $(2, a)(1.1, a)$. Thus no invalidity could be detected.*

Remark 5.5.17. *Points (i) and (iii) are absolutely necessary: without always satisfying them, an algorithm becomes meaningless, as it either fails to displace the observations or to introduce meaningful guards separating the sets of K-closed words with resets.*

On the other hand, it is possible that failing to satisfy constraint (ii) still keeps the invalidity: changing the behaviour for other resets may *lose the invalidity, but is not guaranteed to do so, especially if all reset combinations are not affected.*

We found that no simple algorithm was satisfactory to solve the above sketched problem. The rest of this subsection is organized as follows: we first formalize the problem to be solved, then we review some approaches to solve it and point their shortcomings. Finally, we propose an algorithm that produces a validity guard, and explain the choice of our approach.

A formalization of constraints

Consider an invalid K-closed word with reset $w_{zr} = (z_{i-1}, a_i, r_i)_{1 \leq i \leq n}$ and a pair (w^1_{tr}, w^2_{tr}) of timed words with resets in the characteristic set of w_{zr}. We note, for $j \in 1, 2$, $w^j_{tr} = (t^j_i, a_i, r_i)_{0 \leq i < n'}$ [1].

Trying to replicate the invalidity in a K-closed word $w'_{zr} = (z'_{i-1}, a_i, r_i)_{1 \leq i \leq n}$ requires to construct a new characteristic set containing pairs (w'^1_{tr}, w'^2_{tr}) solving the following

1. The length n' of the timed words is greater than the length of w_{zr}.

constraints, by noting $\mathsf{sig}^{-1}_{\mathsf{behav}}(w'^{j}_{tr}) = (v'^{j}_{i-1} \xrightarrow{(t'^{j}_{i}, a_i, r_i)} v'^{j}_{i})_{1 \leq i \leq n'}$.

$$\forall 1 \leq i \leq n. \quad v'^{j}_{i-1} + t'^{j}_{i} \in z'_{i-1} \ (i.e., \ w'^{j}_{tr}[1,n] \in w'_{zr}) \tag{5.1}$$

$$\forall 0 \leq l < l' \leq |w'^{1}_{tr}|. \ \sum_{l<i\leq l'} t'^{1}_{i} \approx_K \sum_{l<i\leq l'} t'^{2}_{i} \text{ iff } \sum_{l<i\leq l'} t^{1}_{i} \approx_K \sum_{l<i\leq l'} t^{2}_{i} \tag{5.2}$$

Here (5.1) corresponds to targeting w'_{zr} (part of constraints (i)), expressing that $w'^{1/2}_{tr} \in w'_{zr}$, and (5.2) encodes (ii) (and the corresponding part of (i)), *i.e.*, the preservation of the equivalence relation between the words for any reset sequence. Indeed, the clock valuations for any reset sequence are sums of consecutive delays. Together, these equations encode (i) and (ii).

Remark 5.5.18. *Notice that the constraints in* (5.2) *can be relieved in the following way: one only has to consider reset combinations for which both words of the pair are valid, as if they reach invalid states we know that the reset combination does not happen in the model we wish to learn. We do not give the details of this optimization here, as it would not help to construct a general algorithm and would make equations* (5.2) *much less readable.*

The constraint (iii) can be stated in the following way: consider two K-closed words with resets $w^1_{zr} = ((z^1_i, a_i, r_i)_{1\leq i\leq n})$ and $w^2_{zr} = ((z^2_i, a_i, r_i)_{1\leq i\leq n})$ corresponding to two different invalidities. When one tries to replicate the invalidity of w^1_{zr} in ${w^1_{zr}}' = ((z^{1\prime}_i, a_i, r_i)_{1\leq i\leq n})$ and w^2_{zr} in ${w^2_{zr}}' = ((z^{2\prime}_i, a_i, r_i)_{1\leq i\leq n})$, the following property should hold: for any guard $c \sim n$ with $c \in C$, $n \in \mathbb{N}$ and $\sim \in \{<, \leq, \geq, >\}$ such that $z^{1\prime}_i \in g$ and $z^{2\prime}_i \notin g$, it holds $z^1_i \in g$ and $z^2_i \notin g$.

Different approaches

Different approaches are now listed, together with counter-examples showing the properties that they would violate.

A first simple approach would be to reason at the level of observations inside the invalidities characteristic sets and construct linear combinations of timed words with resets. The problem of this approach is that it is very well possible that the combinations are not K-equivalent, as shown in Example 5.5.19.

Example 5.5.19. *Consider the words (as depicted on Figure 5.12)* $w^1_1 = (0.1, a, \{c_a\})(0.95, b, \emptyset)$, $w^1_2 = (0.95, a, \{c_a\})(0.1, b, \emptyset)$ *and* $w^2_1 = (3.1, a, \{c_a\})(1.95, b, \emptyset)$, $w^2_2 = (3.95, a, \{c_a\})(1.1, b, \emptyset)$ *that could be prefixes of pairs of elements of* $\mathcal{W}_1$ *and* $\mathcal{W}_2$ *respectively. The final valuations*

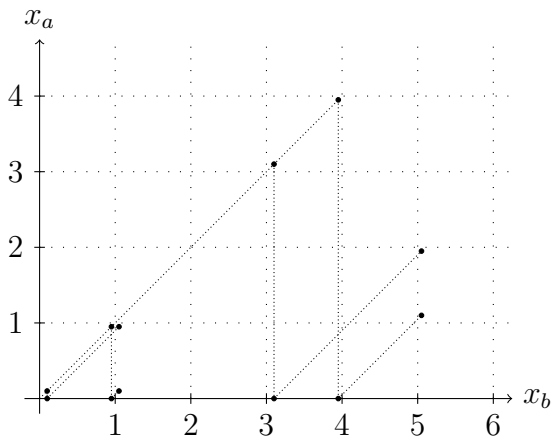


Figure 5.12: Two pairs of timed word with resets that cannot be handled by a direct binary search.

for the first pair are $(c_a = 0.95, c_b = 1.01)$ *and* $(c_a = 0.1, c_b = 1.05)$*, and for the second pair* $(1.95, 5.05)$ *and* $(1.1, 5.05)$*. As it can be seen, the values of* c_a *in both pairs are close to the limits of the constraints of* K*-equivalence (assuming* $K > 1$*). As the values of the pairs in* c_b *are vastly different, most linear combinations of the top elements of those pairs are not equivalent to linear combinations of bottom elements, as* $c_a > 1$ *for the top ones and* $c_a < 1$ *for the bottom ones.*

This makes such direct method impossible to use, as observations that are not K-equivalent can never detect an invalidity.

Another possible approach is to apply offsets to zones in the invalid K-closed words based on their current distances *i.e.* define a "distance" between zones and use it on the zones z_i^1 and z_i^2 to make them closer in the sense of that distance. The problem with that approach is that the distance between two zones at depth i is not affected only by operations at depth i but by all operations on depth $0 < j < i$. Indeed if a zone is modified at depth j, the effect of that modification has to be taken into account on all the future zones to ensure that the resulting word is feasible.

For this reason applying modifications at each depth i simultaneously based on distances between z_i^1 and z_i^2 would be meaningless: the effect of the modifications on the first zones would invalidate the next ones. Notably, the resulting K-closed word would not always be feasible.

To avoid this, one could chose to work only on a given depth i at a time, and loop through the word. Sadly, such modifications would (sometimes) either invalidate condition

(i) or (iii), as shown in Example 5.5.20.

Example 5.5.20. *Consider the two one clock K-closed words represented on Figure 5.13a*

$$w_{zr}^1 = (0 < c_a < 1, a, \emptyset)(1 < c_a < 2, a, \emptyset)$$

$$w_{zr}^2 = (4 < c_a < 5, a, \emptyset)(4 < c_a < 5, a, \emptyset).$$

*An algorithm trying to reduce the distance between them at depth one without breaking constraint (iii) at this depth would likely propose to replace their first zones by $2 < c_a < 3$ (*i.e. *an offset of $2 \times \mathbb{1}_{c_a}$ for w_{zr}^1 and $-2 \times \mathbb{1}_{c_a}$ for w_{zr}^2). If the effect of this modification is not propagated at depth two, then w_{zr}^1 would be transformed in*

$$w_{zr}^3 = (2 < c_a < 3, a, \emptyset)(1 < c_a < 1, a, \emptyset)$$

which cannot be satisfied by any timed word, as it would require a negative delay.

On the other hand, applying the effect of this transformation With the effect of this modification on the zones at depth 2 the same offset would be applied on the zone at depth two, resulting in the words

$$w_{zr}^{1\,\prime} = (2 < c_a < 3, a, \emptyset)(3 < c_a < 4, a, \emptyset)$$

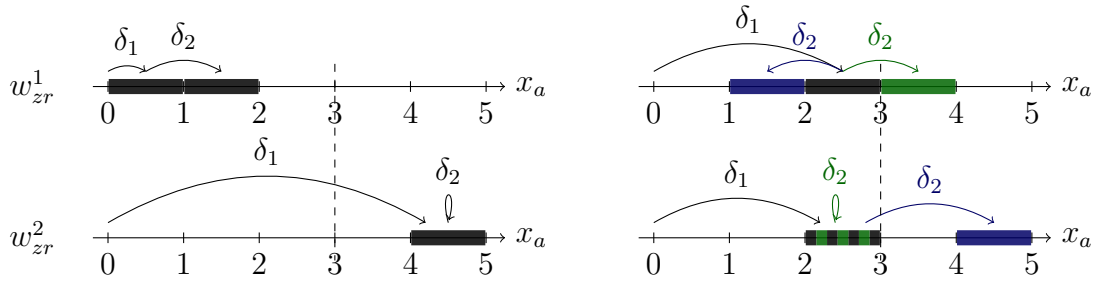
$$w_{zr}^{2\,\prime} = (2 < c_a < 3, a, \emptyset)(2 < c_a < 3, a, \emptyset).$$

*Because of this, the guard $g : c_a > 3$ separates the zones at depth two in a different way for (w_{zr}^1, w_{zr}^2) on one hand (*i.e. *$z_2^1 \notin g$ and $z_2^2 \in g$) and $(w_{zr}^{1\,\prime}, w_{zr}^{2\,\prime})$ (*i.e. *$z_2^{1\prime} \in g$ and $z_2^{2\prime} \notin g$). The zone words after modification for both choices (propagating the offsets or not) are represented in Figure 5.13b.*

A possibility is to mimic the binary search employed for inconsistency: use delays between subsequent zones (or an abstract representation of such delays at least) as the metric we want to converge on. For this we define a pseudo-distance Δ on zones as follows.

Definition 5.5.21. *Consider two K-closed zones z_1 and z_2. We define $\Delta(z_1, z_2)$ as:*

$$\Delta(z_1, z_2) = \min\{|n_2 - n_1| \mid \exists c \in C, \exists n_1, n_2 \in \mathbb{N},\ c \prec_1 n_1 \text{ appears in } z_1 \wedge \\ c \prec_2 n_2 \text{ appears in } z_2\ \wedge\ (\prec_1, \prec_2 \in \{<, \leq, =\}\ \vee\ \prec_1, \prec_2 \in \{>, \geq, =\})\}$$



(a) w^1_{zr} and w^2_{zr}.

(b) Effect of offsets at depth 1 with and without propagation at depth 2.

Figure 5.13: One depth modification.

The intuition behind the distance Δ is the following:

Proposition 5.5.22. *Consider a K-closed zone z and a K-closed time successor z' of z,* i.e. $z' \in \overrightarrow{z}$*, such that no constraint can be suppressed in z or z' without changing the set of satisfying valuations. Then* $\Delta(z, z') = \max\{n \in \mathbb{N} \mid z' \in \overrightarrow{z+n}\}$

This means that Δ is the maximal integer constant you can add to a valuation in z while steel having a successor in z'. Δ will then be used as an abstraction of delays for K-closed words with resets.

Proof. First, notice that indeed $z' \in \overrightarrow{z + \Delta(z, z')}$ as for any constraint $c \prec k$ in $z + \Delta(z, z')$ with $\prec \in \{<, \leq\}$, we have by definition of Δ and offsets that

$$k = n + \Delta(z, z') \leq n'$$

with n the corresponding constraint in z and n' in z'. Furthermore the diagonal constraints are unchanged and the future operator lifts all upper bounds. Hence for all $v \models z$, $v \models \overrightarrow{z + \Delta(z, z')}$.

On the other hand for $N \geq \Delta(z, z') + 1$ by considering an argmin constraint $c \prec n'$ in z' (*i.e.* a constraint that minimizes the $|n' - n|$), as there is no useless constraint in z' there are $v \models z'$ infinitely close to that constraint (else it could be suppressed). For such v we have that $v(c) < n' + 1 = n + \Delta(z, z') + 1 = n + N$ and hence $v \not\models \overrightarrow{z + N}$. Thus we have our result. □

The necessary elements to build a binary search for K-closed words with resets have now been introduced.

Remark 5.5.23. *Notice that this kind of binary search can only work if applied to the whole word. Applying it to just a given depth would in general lead to the exact situation of Example 5.5.20 (when the offset is propagated to future zones).*

Sadly, this approach fails to satisfy constraint (iii) because of the variety of forms of the zones, as shown in Example 5.5.24.

Example 5.5.24. *Consider the K-closed words presented in Figure 5.14a:*

$$w^1_{zr} = (0 < c_a < 1 \wedge 0 < c_b < 1, b, \{c_b\}).((c_a = 1 \wedge 0 < c_b < 1, a, \emptyset))$$

$$w^2_{zr} = (2 < c_a < 3 \wedge 2 < c_b < 3, b, \{c_b\}).((2 < c_a < 3 \wedge c_b = 0, a, \emptyset)) \ .$$

In order to apply a binary search, one has to reason on the K-closed runs associated with these words, so as to deduce the delays Δ taken. We note $\{\mathbf{0}\}$ for the initial zone corresponding to this set.

$$\mathsf{sig}^{-1}_{behav}(w^1_{zr}) = \{\mathbf{0}\} \xrightarrow{\delta} 0 < c_a < 1 \wedge 0 < c_b < 1 \xrightarrow{(b,\{c_b\})} 0 < c_a < 1 \wedge c_b = 0 \xrightarrow{\delta}$$
$$c_a = 1 \wedge 0 < c_b < 1 \xrightarrow{(a,\emptyset)} c_a = 1 \wedge 0 < c_b < 1$$

$$\mathsf{sig}^{-1}_{behav}(w^2_{zr}) = \{\mathbf{0}\} \xrightarrow{\delta} 2 < c_a < 3 \wedge 2 < c_b < 3 \xrightarrow{(b,\{c_b\})} 2 < c_a < 3 \wedge c_b = 0 \xrightarrow{\delta}$$
$$2 < c_a < 3 \wedge c_b = 0 \xrightarrow{(a,\emptyset)} 2 < c_a < 3 \wedge c_b = 0$$

Using this, one can see that at depth 1 the "delays" built from Δ are 0 for w^1_{zr} and 2 for w^2_{zr}. At depth 2 both words have delay 0. Thus the result of a binary search would give the K-closed words presented in Figure 5.14b.

One can see that the zones at depth 2 have exchanged positions with respect to e.g. *the guard $c_a < 2$.*

Remark 5.5.25. *Notice that the issue raised in Example 5.5.24 does not really depend on Δ. It would occur for any pseudo distance used, precisely because we do not have a valid distance on general zones.*

In the example, adding different offsets to the zones at depth 2 would not satisfy constraint (iii).

At depth one. At depth two.

(a) The K-closed words before modification.

At depth one. At depth two.

(b) The K-closed words after modification.

Figure 5.14: Issue with binary search on general zones.

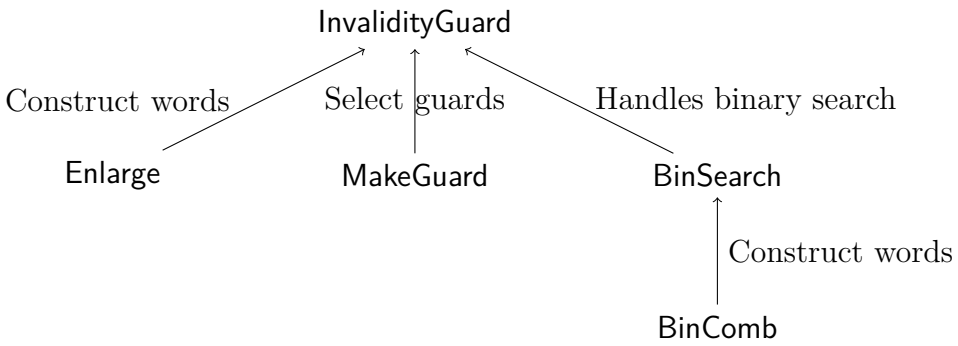


Figure 5.15: Algorithmic structure backing InvalidityGuard.

All the aforementioned approaches cannot be used, as they either fail to construct meaningful K-closed words with resets or to satisfy constraint (iii). In the following, we propose a method that will always satisfy constraints (i) and (iii), but will fail to satisfy constraint (ii) in a specific step. We argue that this is more desirable as loosing the invalidity only impacts *efficiency* (the introduced guard is less precise and may have to be refined) instead of correctness by *e.g.* introducing guards that do not separate all targeted behaviours *i.e.* failing constraint (iii).

Binary search on open zones

We build on the above binary search method, but try to avoid the difficult handling of K-closed zones of different form (open or closed in different dimensions).

The solution we propose is based on a binary search on *open* zones, which satisfies all three constraints (as proven below). That binary search is initialized with a zone *enlargement* phase. This step respects constraints (i) and (iii) but not condition (ii) (a counter example is shown below).

The general algorithm is InvalidityGuard, described in Algorithm 12. It takes as input the decision state of the TLG which successors have been pruned, along with the characteristic sets of the invalidities, and adds a set of validity guards to the set $\mathcal{G}_{Val}$ of validity guards using algorithm MakeGuard (Algorithm 11). The algorithm depends on BinSearch (Algorithm 14) that handles the binary search on open zones and Enlarge (Algorithm 10) that proposes open K-closed words replicas of the initial invalidities. BinSearch itself depends on BinComb to propose new K-closed words replicas (and their characteristic sets). The algorithmic structure is summarized in Figure 5.15.

We first describe the properties of the enlargement phase, then of the binary search on open zones and finally of InvalidityGuard in general.

The Enlarge algorithm is depicted in Algorithm 10. Its idea is to replace equality constraints in the K-closed words by large ones (*i.e.* $k < c < k+1$), based on the following fact.

Lemma 5.5.26. *Any K-closed zone z that is not open has one unique direct time successor K-closed zone, that we note $z\nearrow$. For any constraint $k < c < k+1$ in z, that constraint appears in $z\nearrow$. For any constraint $c = k$ in z, a constraint $k < c < k+1$ appears in $z\nearrow$.*

Proof. We first prove that $z\nearrow$ is well defined. For this consider a zone z that is not open and take a valuation v in that zone. Then there is some clocks c such that $v(c) \in \mathbb{N}$ (as z is not open). Consider $0 < \epsilon << 1$ infinitesimal. Then $v + \epsilon$ belongs to an open zone such that for $c \in C$, if $v(c) = k \in \mathbb{N}$ then $k < v(c) + \epsilon < k + 1$ and if $k < v(c) < k + 1$ then $k < v(c) + \epsilon < k + 1$ (as long as $\epsilon < k + 1 - v(c)$). This being true for any valuation of z, and the constraints on the new zone depending uniquely on the constraints of z, we have that $z\nearrow$ is well-defined and open. □

The Enlarge algorithm takes as parameters an invalid K-closed word with resets w^1_{zr}, its characteristic set $\mathcal{W}$ and a positive real ϵ.

Algorithm 10: Constructing an open K-closed word from an initial K-closed word, and replicating the characteristic set associated.

```
1 Enlarge
  Input: An invalid K-closed word with resets w_zr, its characteristic set W and an
         offset ε > 0.
  Output: An open K-closed word with resets w'_zr and a replica set W'.
2 W' := W and w'_zr := w_zr
3 Noting w'_zr := ((z'_i, a_i, r_i))
4 for for i ∈ [1, |w_zr|] do
5 |   if z'_i is not open then
6 |   |   z'_i := z'_i ↗
7 |   |   for ((t^1_j, a_j, r_j)_j, (t^2_j, a_j, r_j)_j) ∈ W' do
8 |   |   |   t^1_i, t^2_i + := ε
9 Return w'_zr, W'.
```

It simply replaces all non-open zones in w^1_{zr} by their direct time successors, and add ϵ to the delays of the words. We show that for a fitting (small) value of ϵ this algorithm respects equation 5.1.

Proposition 5.5.27. *Consider $w_{zr} = ((z_i, a_i, r_i))_{1 \leq i \leq n}$ an invalid K-closed word and $\mathcal{W}$ its characteristic set. Then for*

$$\epsilon < \frac{\min_{(w^1_{tr}, w^2_{tr}) \in \mathcal{W}} \min_{((t_i,a_i,r_i))_{1 \leq i \leq m} \in \{w^1_{tr}, w^2_{tr}\}} \{1 - \langle \sum_{l=i}^{j} t_l \rangle \mid 1 \leq i \leq j \leq m\}}{n}$$

*$\mathsf{Enlarge}(w_{zr}, W, \epsilon)$ terminates and returns $w'_{zr} = (z'_i, a_i, r_i)$ and W' such that $z'_i = z_i$ if z_i is open and $z'_i = z_i \nearrow$ else. Furthermore all pairs in W' respect the constraints of Equation 5.1 (*i.e. *their elements are in w'_{zr}).*

The value of epsilon proposed is upper bounded so that it cannot make a clock valuation reach a new integer value in an observation, and so for any reset combination.

Proof. First, as Enlarge features only for loops on the (finite) length of w_{zr} and the (finite) number of pairs in $\mathcal{W}'$, it terminates. Next, the constraint on z'_i compared to z_i is clearly satisfied as the z'_i are initialized equal to z_i and then replaced by their direct time successors if and only if they are not open.

It remains to show that the observations of the characteristic set $\mathcal{W}'$ are indeed in w'_{zr}. Consider w'_{tr} an element of a pair of $\mathcal{W}'$. We note $\mathsf{sig}^{-1}_{\mathsf{behav}}(w'_{tr}) = (v'_{i-1} \xrightarrow{t'_i} v'_{i-1} + t'_i \xrightarrow{(a_i,r_i)} v_i)$ and consider the corresponding w_{tr} in $\mathcal{W}$ $\mathsf{sig}^{-1}_{\mathsf{behav}}(w_{tr}) = (v_{i-1} \xrightarrow{t_i} v_{i-1} + t_i \xrightarrow{(a_i,r_i)} v_i)$. We show by induction on $1 \leq i \leq |w'_{zr}|$ that $v'_{i-1} + t'_i \in z'_i$. For $i = 1$, either z_1 was open; in which case $v'_0 + t'_1 = v_0 + t_1 \in z'_1 = z_1$; or z_1 was not and $z'_1 = z_1 \nearrow$. In this case, $v'_0 + t'_1 = v_0 + t_1 + \epsilon$ and for all guards $c = k$ in z_1, $k < (v'_0 + t'_1)(c) < k + 1$ as $\epsilon < 1$. Furthermore for all guards $k < c < k + 1$ in z_1, $k < (v'_0 + t'_1)(c) < k + 1$. Indeed, $\epsilon < 1 - \langle v_0 + t_1 \rangle$ by definition. Hence $v'_0 + t'_1 \in z'_1$.

Suppose that the property is verified up to a depth i (included). We show the result for depth $i + 1$. If z_{i+1} is open then $z'_{i+1} = z_{i+1}$ and $v'_i + t'_{i+1} = v'_i + t_{i+1}$. We want to show that the clock values in $v'_i + t'_{i+1}$ have the same integer part than the ones of $v_i + t_{i+1}$. As both have a non-zero fractional part ($v_i + t_{i+1}$ because it is in an open zone, and $v'_i + t'_{i+1}$ because it only adds positive delays, hence without changing its integer part it cannot reach a zero fractional part), this suffices to prove that $v'_i + t'_{i+1} \in z_{i+1} = z'_{i+1}$.

Notice that for $c \in C$, $(v'_i + t'_{i+1})(c) = (v_i + t_{i+1})(c) + k.\epsilon$ with $k \in \mathbb{N}$, $k \leq i$. By definition of ϵ, $k + \epsilon$ is not enough to change the integer part of a clock value, as clocks values are sum of consecutive delays. Hence we have our result.

If z_{i+1} is not open, we get our result in the same way as $(v'_i + t'_{i+1})(c) = (v_i + t_{i+1})(c) + k.\epsilon$ with $k \in \mathbb{N}$, $1 \leq k \leq i + 1$. First for open constraints in z_{i+1} as before we know that adding $k.\epsilon$ does not change the integer part and thus cannot make the fractional part null. On

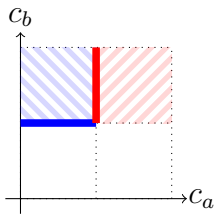


Figure 5.16: Non-open K-closed zones and their direct timed successors.

the other hand, for closed constraints (*i.e.* equality constraints) in z_{i+1}, the addition of ϵ in the last delay ensures that it reach the next open constraint (*i.e.* from $c = c$ to $c < c$) and $k.\epsilon < 1$ thus $(v'_i + t_{i+1})(c) < c + 1$. This corresponds exactly to the constraints of $z_{i+1} \nearrow = z'_{i+1}$, hence we have our result. □

Remark 5.5.28. *The* Enlarge *algorithm is built simple. It could be possible to add ϵ at fewer steps by tracking if the previous ϵ were enough to push the valuations in $z_i \nearrow$. This in turn would allow for slightly more permissive constraints on ϵ. This could be done for example by checking if ϵ has already been added since the last time the clocks which have closed constraints were reset.*

We choose to present the algorithm in a simple way to enhance readability at the price of this optimization.

With respect to constraint (iii), for two K-closed zones z_1, z_2 there are guards that separate $z_1 \nearrow$ and $z_2 \nearrow$ but not z_1 and z_2. An example is given in Example 5.5.29.

Example 5.5.29. *Consider the zones $z_1 : 0 < c_a < 1 \wedge c_b = 2$ and $z_2 : c_a = 1 \wedge 1 < c_b < 2$. They are represented on Figure 5.16 together with their direct time successors z'_1 and z'_2. Notice that the guard $g : c_a \leq 3$ accepts z_1, $z'_1 \nearrow$ and z_2 but not $z'_2 \nearrow$.*

Despite this difficulty, we can use the fact that the zones have been enlarged toward the future to select guards that do not distinguish between non-open zones and their direct time successors, allowing to preserve constraint (iii).

Lemma 5.5.30. *For any K-closed zone z, and any guard of the form $g : c \prec n$ with $n \in \mathbb{N}$, $c \in C$ and $\prec \in \{\leq, >\}$ we have that $z \nearrow \in g$ if and only if $z \in g$. Such guard is called* enlargement closed *guard.*

Proof. This result comes from the fact that these guards cannot distinguish between $v(c) = k \in \mathbb{N}$ and $k < v(c) < k + 1$, which are the only differences between a non-open K-closed zone and its direct time successor, as shown in Lemma 5.5.26. □

We use these guards to separate *open* K-closed zones in BinSearch. Notice that if there exists a guard separating K-closed open zones, then there exists an enlargement closed one that separates them.

Finally, the Enlarge algorithm replicates characteristic sets while guaranteeing that pairs of K-equivalent words *for any sequence of resets* remain equivalent.

Proposition 5.5.31. *For w^1_{tr}, w^2_{tr} elements of pairs of a characteristic set $\mathcal{W}$, considering w'^1_{tr}, w'^2_{tr} the corresponding words in $\mathcal{W}'$ constructed by a call to Enlarge$(w_{zr}, \mathcal{W}, \epsilon)$ with an ϵ satisfying the constraint of Proposition 5.5.27 we have that*

$$\forall\ 1 \leq i \leq j \leq \min(|w^1_{tr}|, |w^2_{tr}|)\ \sum_{k=i}^{j} t^1_k \approx_K \sum_{k=i}^{j} t^2_k \implies \sum_{k=i}^{j} {t'}^1_k \approx_K \sum_{k=i}^{j} {t'}^2_k\ .$$

where t^m_k is the k-th delay in w^m_{tr} and ${t'}^m_k$ is the k-th delay in w'^m_{tr}.

Proof. The proof uses the same arguments as the proof of Proposition 5.5.27 but generalised to all sums of consecutive delays and not only clock values for the sequence of resets considered. □

This property assures that the structure of the characteristic set is not broken (*i.e.* pairs remain K-equivalent). Nevertheless this approach does not respect all the constraints of Equation 5.2. Indeed, adding ϵ sometimes makes equivalent two previously non-equivalent delay sequences, as shown on Example 5.5.32. As explained in that example, this is not a failure of that particular algorithm but a larger problem with any zone enlargement.

Example 5.5.32. *Consider the sequences of delays and clock resets displayed in Table 5.1 for a set of clocks $C = \{c_a, c_b\}$. The three words w_1, w_2 and w_3 are K-equivalent and reach at depth 5 a non-open zone with constraint $c_a = 1$. Hence any algorithm looking to enlarge the zone should add (or remove) an ϵ at the fourth or fifth delays (Enlarge does it at the fifth).*

Now consider the alternative sequence of resets where c_b is not reset by the second transition. For that reset sequence, the three words are not *K-equivalent. Notably, at the end of the fifth delay, the valuation of c_b is 2.1 for w_1, 2 for w_2 and 1.9 for w_3. Hence adding (resp. removing) an ϵ to the total elapsed time makes the valuations of w_1 and w_2 equivalent (resp. w_3 and w_2), which violates the constraints of equation 5.2. Hence to satisfy these constraints, an ϵ should be removed (resp. added) from the delays 1 2 or 3. Yet, the same problem occurs at depth 3 as the valuation of c_b at depth 3 (without the clock reset of c_b) is 1.1 for w_1, 1 for w_2 and 0.9 for w_3.*

	1	2	3	4	5	
w_1	0.5	0.4	0.2	0.3	0.7	$2.1 + \epsilon$
w_2	0.4	0.4	0.2	0.5	0.5	$2.0 + \epsilon$
w_3	0.3	0.4	0.2	0.7	0.3	$1.9 + \epsilon$
	$\{c_a\}$	$\{c_b\}$		$\{c_a\}$		

Table 5.1: An example of enlargement making timed words with resets equivalent.

Hence we have here an example of words that cannot be "enlarged" without making some of their previously non-equivalent valuations equivalent.

The enlargement step is the first performed by InvalidityGuard (Algorithm 12), that uses Enlarge to create open replicas of its initial K-closed zones and performs membership queries on the resulting set. InvalidityGuard makes calls to MakeGuard (Algorithm 11) to introduce all necessary guards when terminating. That algorithm takes an ordered list of pairs of K-closed words, and adds validity guards to separate them when necessary.

To ease the algorithms notations, we define the predicate IsOpen that returns true if and only if its parameter is an open K-closed word.

Algorithm 11: Creates validity guards to separate an ordered list of pairs of K-closed words with resets.

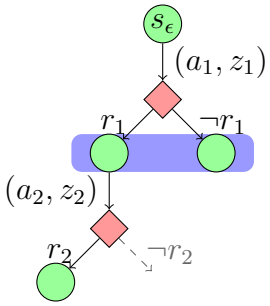
MakeGuard

Input: Two sets $\mathcal{W}_1$ and $\mathcal{W}_2$ of K-closed words with resets that have to be separated by validity guards.

Output: A set of validity guards separating the two sets.

```
1 Let 𝒢 := ∅
2 for w¹_zr ∈ 𝒲₁ do
3     for w²_zr ∈ 𝒲₂ do
4         if a guard (g, i) for (g, i, 𝒲, 𝒲′) ∈ 𝒢 separates (w¹_zr, w²_zr) then
5             Let 𝒲 := 𝒲 ∪ {w¹_zr} and 𝒲′ := 𝒲′ ∪ {w²_zr}
6         else
7             Let 𝒢 := 𝒢 ∪ {(g, i, {w¹_zr}, {w²_zr})} where (g, i, w¹_zr, w²_zr) is a validity
                guard and is enlargement closed if IsOpen(w¹_zr) ∧ IsOpen(w²_zr).
8     Return 𝒢
```

The main purpose of MakeGuard is to avoid to add validity guards to $\mathcal{G}_{Val}$ when another validity guard already separates a pair of incompatible invalidities. In this case, we nevertheless add the pair to the set of the validity guard. This is a consequence of the enlargement phase: as enlargement is not a linear combination, it is possible to separate



Candidates for $Rs_1(w_{zr})$.

Figure 5.17: The K-closed words tested for $Rs_1 w_{zr}$ with $w_{zr} = ((a_i, z_i, r_i))_{1 \leq i \leq 2}$.

enlarged zones from the original ones by a guard (consistency guard for example). In such case, it is important that we know that a validity guard must still be introduced to separate the pair. A more in depth discussion is made in Section 5.5.4 for the proof of Lemma 5.5.51.

We order the pairs from those that are separated by few guards *i.e.* a non-open K-closed word and its enlargement or a pair of large K-closed words on which we were able to apply a binary search to pairs on which we pick from a greater set of guards—and thus have a lower probability of obtaining a meaningful one.

InvalidityGuard first discusses wether or not the invalidities are open, and in the case they are not enlarges them. Then it discusses on the validity of reset combinations for the enlarged K-closed words with resets. To ground the discussion, we define the set of valid resets after a prefix of a word.

Definition 5.5.33. *Consider an acceptance function* Acc*. For a timed word with resets* w_{tr} *and* $0 < i \leq |w_{tr}|$*, we call* reset set *at depth* i*, noted* $Rs_i(w_{tr})$*, the set of valid resets for the* i*-th transition after the prefix of size* $i-1$ *of* w_{tr}*. Formally,* $f \in \{\top, \bot\}$ *is in* $Rs_i(w_{tr})$ *if and only if* $w_{tr}[1, i-1].(t, a, f)$ *is valid, with* t, a *the* i*-th delay and action of* w_{tr}*.*

We generalize this definition to K*-closed words and to a set of timed words with resets which prefix(es) of size* i *belongs to the same* K*-closed word.*

An illustration of the valid reset is given in Figure 5.17. In the case where the two original K-closed words with resets are open, a binary search is performed to introduce a guard separating them in the form of a validity guard for their linear combinations (see BinComb and BinSearch for more details).

Else, an enlargement is performed and the algorithm discusses according to the validity of the resets at depth $|s_o|+1$ of the enlarged words. Notice that in this case, as enlargements are not linear combinations, we may have to introduce several validity guards or to make

Algorithm 12: Finding a guard to separate two invalidities.

InvalidityGuard

Input: $s_d = (s_o, g, a)$ a decision state in $\mathcal{M}$ with two invalid (pruned) successors; $\mathcal{W}_1$ and $\mathcal{W}_2$ the two characteristic sets of the invalidities.

Output: $\mathcal{G}_{Val}$ modified by adding a validity guard

Let w^1_{zr} (resp. w^2_{zr}) be the invalid K-closed word of size $|s_o|+1$ characterized by $\mathcal{W}_1$ (resp. $\mathcal{W}_2$)
Let $\epsilon \in \mathbb{R}_{\geq 0} \setminus \{0\}$ satisfying the constraints of Prop. 5.5.27 w.r.t. $\mathcal{W}_1$ and $\mathcal{W}_2$
switch $(\mathit{IsOpen}(w^1_{zr}), \mathit{IsOpen}(w^2_{zr}))$ **do**
  case $(\top, \top)$ **do**
    Let $(w^1, w^2) :=$ BinSearch(w^1_{zr}, w^2_{zr}) and MakeGuard$(\{w^1\}, \{w^2\})$
  case $(\top, \bot)$ **do**
    ${w^2_{zr}}', \mathcal{W}'_2 :=$ Enlarge$(w^2_{zr}, \mathcal{W}_2, \epsilon)$
    For $(w_{tr}, w'_{tr}) \in \mathcal{W}'_2$ Request(w_{tr}) and Request(w'_{tr})
    switch $Rs_{|s_o|+1}({w^2_{zr}}')$ **do**
      case $Rs_{|s_o|+1}(w^2_{zr})$ **do**
        Let $(w^1, w^2) :=$ BinSearch$(w^1_{zr}, {w^2_{zr}}')$ and MakeGuard$(\{w^1\}, \{w^2, w^2_{zr}\})$
      case $Rs_{|s_o|+1}(w^1_{zr})$ **do**
        MakeGuard$(\{{w^2_{zr}}', w^1_{zr}\}, \{w^2_{zr}\})$
      otherwise do
        MakeGuard$(\{w^1_{zr}\}, \{w^2_{zr}\})$
  case $((\bot, \top))$ **do** Same as above inverting 1 and 2.
  case $((\bot, \bot))$ **do**
    ${w^1_{zr}}', \mathcal{W}'_1 :=$ Enlarge$(w^1_{zr}, \mathcal{W}_1, \epsilon)$; ${w^2_{zr}}', \mathcal{W}'_2 :=$ Enlarge$(w^2_{zr}, \mathcal{W}_2, \epsilon)$
    For $(w_{tr}, w'_{tr}) \in \mathcal{W}'_1 \cup \mathcal{W}'_2$ Request(w_{tr}) and Request(w'_{tr})
    switch $(Rs_{|s_o|+1}({w^1_{zr}}'), Rs_{|s_o|+1}({w^2_{zr}}'))$ **do**
      case $(\emptyset, \emptyset)$ *or* $(\{\top, \bot\}, \{\top, \bot\})$ **do**
        MakeGuard$(\{w^1_{zr}\}, \{w^2_{zr}\})$
      case $(Rs_{|s_o|+1}(w^1_{zr}), \emptyset)$ *or* $(Rs_{|s_o|+1}(w^1_{zr}), \{\top, \bot\})$ **do**
        MakeGuard$(\{w^1_{zr}, {w^1_{zr}}'\}, \{w^2_{zr}\})$
      case $(Rs_{|s_o|+1}(w^2_{zr}), \emptyset)$ *or* $(Rs_{|s_o|+1}(w^2_{zr}), \{\top, \bot\})$ **do**
        MakeGuard$(\{w^1_{zr}\}, \{{w^1_{zr}}', w^2_{zr}\})$
      case $(Rs_{|s_o|+1}(w^1_{zr}), Rs_{|s_o|+1}(w^2_{zr}))$ **do**
        Let $(w^1, w^2) =$ BinSearch$({w^1_{zr}}', {w^2_{zr}}')$
        MakeGuard$(\{w^1, w^1_{zr}\}, \{w^2, w^2_{zr}\})$
      case $(Rs_{|s_o|+1}(w^2_{zr}), Rs_{|s_o|+1}(w^2_{zr}))$ **do**
        MakeGuard$(\{w^1_{zr}\}, \{{w^1_{zr}}', w^2_{zr}, {w^2_{zr}}'\})$
      case $(Rs_{|s_o|+1}(w^2_{zr}), Rs_{|s_o|+1}(w^1_{zr}))$ **do**
        Let $(w^1, w^2), (w^1_{first}, w^2_{first}) =$ BinSearch$({w^1_{zr}}', {w^2_{zr}}', first)$
        MakeGuard$(\{w^1, w^2_{zr}\}, \{w^2, w^1_{zr}\})$
        For $j \in \{1, 2\}$ MakeGuard$(\{{w^j_{zr}}', w^j_{first}\}, \{w^j_{zr}\})$
      otherwise do Symetric to previous cases.

$w_{zr}^{1\,\prime}$ / $w_{zr}^{2\,\prime}$	$\emptyset$ or $\{\top, \bot\}$	$Rs_{\lvert s_o\rvert+1}(w_{zr}^1)$	$Rs_{\lvert s_o\rvert+1}(w_{zr}^2)$
$\emptyset$ or $\{\top, \bot\}$	$g(\{w_{zr}^1\}, \{w_{zr}^2\})$	$g(\{w_{zr}^1, w_{zr}^{1\,\prime}\}, \{w_{zr}^2\})$	$g(\{w_{zr}^1\}, \{w_{zr}^{1\,\prime}, w_{zr}^2\})$
$Rs_{\lvert s_o\rvert+1}(w_{zr}^2)$	$g(\{w_{zr}^1\}, \{w_{zr}^{2\,\prime}, w_{zr}^2\})$	BS: $g(\{w_{zr}^1, w^1\}, \{w_{zr}^2, w^2\})$	$g(\{w_{zr}^1\}, \{w_{zr}^{1\,\prime}, w_{zr}^2, w_{zr}^{2\,\prime}\})$
$Rs_{\lvert s_o\rvert+1}(w_{zr}^1)$	$g(\{w_{zr}^1, w_{zr}^{2\,\prime}\}, \{w_{zr}^2\})$	$g(\{w_{zr}^1, w_{zr}^{2\,\prime}, w_{zr}^{1\,\prime}\}, \{w_{zr}^2\})$	BS: $g(\{w^1, w_{zr}^2\}, \{w^2, w_{zr}^1\})$, $g(\{w_{zr}^1\}, \{w_{zr}^{1\,\prime}, w_{first}^1\})$, $g(\{w_{zr}^2\}, \{w_{zr}^{2\,\prime}, w_{first}^2\})$

Table 5.2: The effect of the enlargement phase of InvalidityGuard depending on $Rs_{\lvert s_o\rvert+1}(w_{zr}^{1/2\prime})$. We note $g(\mathcal{W}, \mathcal{W}')$ a call to MakeGuard with those parameters, BS a call to BinSearch on (w_{zr}^1, w_{zr}^2) and w^1 w_{first}^1 (resp. w^2, w_{first}^2) the result and optional results corresponding to $w_{zr}^{1\,\prime}$ (resp. $w_{zr}^{2\,\prime}$).

validity guards depend on sets of words to ensure that the guards will be available when needed when reconstructing the TLG (this is discussed in the proof of Lemma 5.5.51). The discussion in the case where both initial invalidities have to be enlarged is summarized in Table 5.2.

After the enlargement step, InvalidityGuard either terminates or calls BinSearch that performs a binary search between two *open* K-closed words. The procedure constructing new K-closed words and characteristic sets from given ones, called BinComb, is described in Algorithm 13.

The BinComb algorithm constructs new K-closed sets and characteristic sets that meet constraints (i), (ii) and (iii). The core of its work is to find the middle (in the sense of Δ) between the abstract delays (*i.e.* the Δ between the zone after the $(i-1)$-th reset and the zone before i-th transition) of the two K-closed words with resets given as arguments and construct both the new K-closed word using these middle zones and a characteristic set for it. The main difficulty lies in the nature of distances between (open) zones: they must remain in $\mathbb{N}$. Hence it is sometimes necessary to round the mean up or down.

The fact it is sufficient to reason on abstract delays comes from the following lemma characterizing zones from delays, resets and directions.

Lemma 5.5.34. *Let $((z'_{i-1}, a_i, r_i))_{1\leq i\leq n}$ be an open K-closed word with resets of behaviours*

$$(z_{i-1} \xrightarrow{\delta} z'_{i-1} \xrightarrow{(a_i, r_i)} z_i)_{1\leq i\leq n} \ .$$

Algorithm 13: Replication of open invalidities.

BinComb

Input: $w^1_{zr} = ((z^1_i, a_i, r^1_i))$ an open invalid K-closed word with its characteristic set $\mathcal{W}_1$ and same for $w^2_{zr} = ((z^2_i, a_i, r^2_i))$, $\mathcal{W}_2$ with both invalid words being of same size and having the same actions and resets except fro the last reset.

Output: ${w^1_{zr}}', \mathcal{W}'_1, {w^2_{zr}}', \mathcal{W}'_2$ replicating the inputs such that ${w^1_{zr}}'$ and ${w^2_{zr}}'$ differ only on their last reset.

Let $\mathcal{W}'_1, \mathcal{W}'_2 := \mathcal{W}_1, \mathcal{W}_2$
For $j \in 1, 3$ let $z^j_{\mathsf{temp}} := \bigwedge_{c \in C} c = 0$
Let $r^1_0 := \top$
Pick above $\in \{\top, \bot\}$
for $i \in [1, |w^1_{zr}|]$ **do**
    $\delta := \frac{\Delta(z^1_i, z^1_{\mathsf{temp}}) + \Delta(z^2_i, z^2_{\mathsf{temp}})}{2}$
    if $\delta \notin \mathbb{N}$ **then** `// Rounding δ up or down`
        if *above* **then**
            $\delta := \lceil \delta \rceil$
        else
            $\delta := \lfloor \delta \rfloor$
            if $\delta = 0 \ \wedge \ \neg r_{i-1}$ **then**
                `// Detects risk for fractional values of delays`
                Return $(w^1_{zr}, \mathcal{W}_1, w^2_{zr}, \mathcal{W}_2)$
        above := ¬above
    if z^3_{temp} *is open* **then**
        $z^3_i := z^3_{\mathsf{temp}} + \delta$
    else
        $z^3_i := z^3_{\mathsf{temp}} \nearrow + \delta$
    for $j \in \{1, 2\}$ **do** `// Handles the observations delays`
        for $((t_i, a_i), (t'_i, a_i)) \in \mathcal{W}'_j$ **do**
            $t, t' + := \delta - \Delta(z^j_i, z^j_{\mathsf{temp}})$
    For $j \in 1, 3$ let $z^j_{\mathsf{temp}} := z^j_{\mathsf{temp}[r_i \leftarrow 0]}$
Return $(z^3_i, a_i, r^1_i), \mathcal{W}'_1, (z^3_i, a_i, r^2_i), \mathcal{W}'_2$.

Then, noting $\{0\}\nearrow: \bigwedge_{c\in C} 0 < c < 1$*:*

$$\forall 1 \leq i \leq n,\ z'_{i-1} = \{0\}\nearrow + \sum_{j=1}^{i} \Delta(z_{j-1}, z'_{j-1}) \times dir^i_j((r_k)_k)\ .$$

Proof. We make the proof by induction. For $i = 1$, there is $n \in \mathbb{N}$ such that for any clock $c \in C$, the constraint $n < c < n + 1$ is in z'_0. By definition of Δ, $n = \Delta(z_0, z'_0)$ and we have our result. For $i > 1$, by induction $z'_{i-2} = \{0\}\nearrow + \sum_{j=1}^{i-1} \Delta(z_{j-1}, z'_{j-1}) \times dir^{i-1}_j((r_k)_k)$ and thus $z_{i-1} = \{0\}\nearrow + \sum_{j=1}^{i-1} \Delta(z_{j-1}, z'_{j-1}) \times dir^i_j((r_k)_k)$. By Proposition 5.5.22 we then have our result because $dir^i_i(\cdot) = \mathbb{1}_C$ and as the two zones are K-closed and open. Indeed the characterization of Proposition 5.5.22 maximum is equality for open K-closed zones. □

The fact that BinComb respects (i) simply comes from correct manipulation of delays, while respecting (iii) requires alternating between rounding δ up and down (when an approximation is required) to avoid stacking approximation errors.

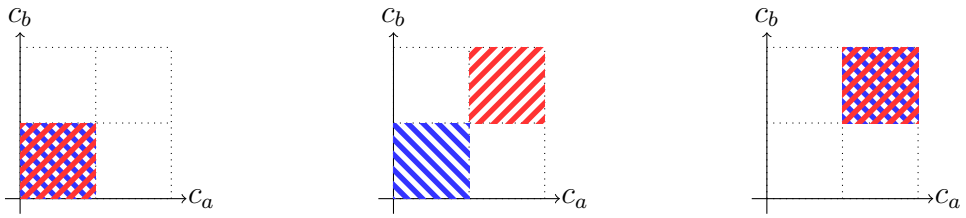
Because of this, we happen to round δ down. This is problematic in the case where it is rounded down to 0 in a delay transition between two open zones (*i.e.* when the previous discrete transition did not reset any clock). In this case one may have to manipulate the *fractional values* of the time elapses in the characteristic sets observations, which would contradict constraints (ii). To avoid this we cut the computation and return the arguments in place of the new K-closed word with resets.

Example 5.5.35. *Consider two (prefixes of) K-closed words with resets depicted in Figure 5.18 (we represent them with $|C| = 2$ but one clock suffice):*

$$w^1_{zr} = (\bigwedge_{c\in C} 0 < c < 1, a, \bot)(\bigwedge_{c\in C} 0 < c < 1, a, \bot)(\bigwedge_{c\in C} 1 < c < 2, a, \bot)\ ,$$

$$w^2_{zr} = (\bigwedge_{c\in C} 0 < c < 1, a, \bot)(\bigwedge_{c\in C} 1 < c < 2, a, \bot)(\bigwedge_{c\in C} 1 < c < 2, a, \bot)\ .$$

There is no problem at depth one as both words have an abstract delay of 0 *(from $\{s_{init}\}$ to its direct successor). At both depth two and three however one word has an abstract delay of* 1 *and one of* 0*. Hence the mean of those delays is* 0.5 *and must be approximated (as abstract delays between zones are always integers). One can see that an upper approximation at both steps would create an un-fitting resulting word, as it would reach zone $\bigwedge_{c\in C} 2 < c < 3$ at depth three, falsifying constraint (iii). Hence using a lower approximation is* necessary*.*



(a) Depth one. (b) Depth two. (c) Depth three.

Figure 5.18: Two problematic close K-closed words with resets.

Yet, it means forcing an abstract delay of 0 *instead of* 1 *on a* K*-closed word with resets. This raises a problem for its characteristic set. Suppose we choose to transform* w^2_{zr} *in* w^1_{zr} *by diminishing its abstract delay at depth two (and augmenting it at depth three) and consider the following (prefix of a) pair of its characteristic set:*

$$(w_t = (0.7, a)(0.7, a)(0.1, a), w'_t = (0.3, a)(1.1, a)(0.1, a)) \ .$$

Diminishing their two first delays to end in $\bigwedge_{c \in C} 0 < c < 1$ *without resets requires to make them equivalent for the reset sequence* $\top\top\top$ *while they are currently not equivalent.*

Remark that this discussion is independent of any algorithm.

Proposition 5.5.36. *Let* w^1_{zr} *and* w^2_{zr} *be two (invalid)* K*-closed words with resets of same size and actions, with equal resets except for the final one and* $\mathcal{W}_1$, $\mathcal{W}_2$ *their characteristic sets. Then* $w^1_{zr}{}', \mathcal{W}'_1, w^2_{zr}{}', \mathcal{W}'_2 = \mathsf{BinComb}(w^1_{zr}, \mathcal{W}_1, w^2_{zr}, \mathcal{W}_2)$ *respect the constraints (i) (ii).*

Proof. We will make the proof using Equations 5.1 and 5.2 instead of constraints (i) and (ii). As already argued this is correct. First notice that all delays given to characteristic set observations are non negative. Indeed, when delays are modified, either the last transition was a resetting one or $\delta \geq 1$. In both cases we show that the new modified delays are non negative. The modification of a delay is of the form

$$t' = t + \delta - \Delta_i$$

with $\Delta_i = \Delta(z_i, z_{\mathsf{temp}})$ the abstract delays between the zones of the word at index i. If the last transition was resetting, $t \geq \Delta_i$ (as the final zone is open). As furthermore $\delta \geq 0$ (it is the mean of two natural integers with a floor or ceil approximation) we have $t' \geq 0$. Else, notice that $t \in [\Delta_i - 1, \Delta_i + 1]$. Thus, as we suppose that $\delta \geq 1$ we have that $t' \geq 0$.

We start by Equation 5.2. Either BinComb returns the arguments, and the result trivially holds (as the delays are not modified), or it adds the same *integer* delay to all elements of $\mathcal{W}_1$ (resp. $\mathcal{W}_2$) to create $\mathcal{W}'_1$ (resp. $\mathcal{W}'_2$). This cannot affect K-equivalence as the fractional parts are not modified and the integer parts are modified in the same way.

We prove that Equation 5.1 is satisfied using the delay based characterization of zones given in Lemma 5.5.34. Note $j \in \{1, 2\}$, δ_i the value of δ during the i-th iteration of the BinComb's main for loop and $\Delta_i^j = \Delta(z_i^j, z_{\mathsf{temp}}^j)$ during this iteration.

$$\begin{aligned} v'_{i-1} + t'_i &= v'_{i-1} + t_i + \delta_i - \Delta_i^j \\ &= v_{i-1} + t_i + \sum_{k=1}^{i} (\delta_k - \Delta_k^j) \times dir_k^i(w_{zr}^1) \end{aligned}$$

by separating the delays added by the algorithm and the pre-existing ones. We hence can say, noting z_i^j the i-th zone in w_{zr}^1:

$$\begin{aligned} v'_{i-1} + t'_i \in z_i^j + \sum_{k=1}^{i} (\delta_k - \Delta_k^j) \times dir_k^i(w_{zr}^1) \\ = \{0\} \nearrow + \sum_{k=1}^{i} (\delta_k) \\ = z_i^3 \end{aligned}$$

by applying Lemma 5.5.34 twice. We thus have our result by the induction principle.

Finally we prove that the constraints (iii) are satisfied. □

The following lemma is the technical core ensuring that BinComb (and BinSearch) respect the constraints (iii).

Lemma 5.5.37. *Consider w_{zr}^1 and w_{zr}^2 be two (invalid) K-closed words with resets of same size and actions, with equals resets except for the final one and $\mathcal{W}_1$, $\mathcal{W}_2$ their characteristic sets. Consider the call $w_{zr}^1{}', \mathcal{W}'_1, w_{zr}^2{}', \mathcal{W}'_2 = \mathsf{BinComb}(w_{zr}^1, \mathcal{W}_1, w_{zr}^2, \mathcal{W}_2)$ and suppose that the result of the function is not its arguments. We note $z_{i-1}^j \xrightarrow{\Delta_i^j} z'_{i-1}{}^j \xrightarrow{(a_i, r_i)} z_i^j$ the abstract runs for $j \in \{1, 2, 3\}$, $j \in \{1, 2\}$ corresponding to w_{zr}^j and $j = 3$ corresponding to the behaviours of both $w_{zr}^1{}'$ and $w_{zr}^2{}'$ (we abuse the notation as we will not use the final zone and reset).*

We have that for any $i \leq i'$:

$$\sum_{k=i}^{i'} \Delta_k^3 = \left\lceil \sum_{k=i}^{i'} \frac{\Delta_k^1 + \Delta_k^2}{2} \right\rceil$$

if the first approximation made after i is $\lceil \cdot \rceil$ and

$$\sum_{k=i}^{i'} \Delta_k^3 = \left\lfloor \sum_{k=i}^{i'} \frac{\Delta_k^1 + \Delta_k^2}{2} \right\rfloor$$

otherwise.

Furthermore, in both cases

$$\sum_{k=i}^{i'} \Delta_k^3 = \frac{\sum_{k=i}^{i'} \Delta_k^1 + \Delta_k^2}{2}$$

if and only if the last approximation made is different from the first one (or no approximation has been made).

Proof. We make the proof by induction on $i' \geq i$. We suppose that the first approximation is $\lceil \cdot \rceil$. The proof for $\lfloor \cdot \rfloor$ is similar. For $i' = i$ we have Δ_i^3 equals to δ at the step i of the **for** loop of BinComb. Thus $\Delta_i^3 = \frac{\Delta_i^1 + \Delta_i^2}{2}$ if no approximation is made (*i.e.* iff $\frac{\Delta_i^1 + \Delta_i^2}{2} \in \mathbb{N}$). Else, as the first approximation is $\lceil \cdot \rceil$ we have $\Delta_i^3 = \left\lceil \frac{\Delta_i^1 + \Delta_i^2}{2} \right\rceil$. In both cases, the property holds.

For the inductive case, suppose that the property holds up to $i' \geq i$. We prove that it holds for $i' + 1$. For this we discuss according to the last approximation made and the need for an approximation at this index.

If the last approximation made was $\lceil \cdot \rceil$ then $\sum_{k=i}^{i'} \Delta_k^3 \neq \sum_{k=i}^{i'} \frac{\Delta_k^1 + \Delta_k^2}{2}$. Precisely $\sum_{k=i}^{i'} \Delta_k^3 = \sum_{k=i}^{i'} \frac{\Delta_k^1 + \Delta_k^2}{2} - \frac{1}{2}$.

Suppose that no approximation is necessary at index $i'+1$, *i.e.* $\Delta_{i'+1}^3 = \frac{\Delta_{i'+1}^1 + \Delta_{i'+1}^2}{2} \in \mathbb{N}$. Then we have our result as one can pass integers under the ceil function and no new approximation can correct the sum to make it equal to the approximated one. Now suppose that an approximation is necessary. Then the approximation applied is $\lfloor \cdot \rfloor$ as approximations are alternating. Thus we have that $\sum_{k=i}^{i'+1} \Delta_k^3 = \sum_{k=i}^{i'+1} \frac{\Delta_k^1 + \Delta_k^2}{2}$ as $\Delta_{i'+1}^3 = \frac{\Delta_{i'+1}^1 + \Delta_{i'+1}^2}{2} + \frac{1}{2}$.

The case where the last approximation made is $\lfloor \cdot \rfloor$ follows the same ideas. □

This lemma is used afterward to show that we do not sum approximation mistakes but always keep an integer approximation of the mean of the original delays.

Combining this wih the chracterization by delays of zones given in Lemma 5.5.34, we can show that the result of a call to BinComb satisfies constraints (iii) with respect to the arguments.

Proposition 5.5.38. *Let w^1_{zr} and w^2_{zr} be two (invalid) K-closed words with resets of same size and actions, with equals resets except for the final one and $\mathcal{W}_1$, $\mathcal{W}_2$ their characteristic sets. Consider the call $w^1_{zr}{}', \mathcal{W}'_1, w^2_{zr}{}', \mathcal{W}'_2 = \mathsf{BinComb}(w^1_{zr}, \mathcal{W}_1, w^2_{zr}, \mathcal{W}_2)$ and $j, k \in \{1, 2\}$. Any separating atomic guard g at depth i for w^j_{zr} and $w^k_{zr}{}'$ such that w^j_{zr} passes in g at depth i verifies that w^{3-j}_{zr} does not pass in g at depth i.*

Proof. We first consider the case where the returns of the BinComb call are its arguments. In this case the result trivially holds. Indeed, no guard can separate w^j_{zr} and $w^k_{zr}{}' = w^k_{zr}$ for $k = j$ and for $k = 3 - j$ any guard separating w^j_{zr} and w^{3-j}_{zr} indeed separates them.

We now reason on the case where a modification is made by BinComb. Consider that $g : c \prec n$. Then noting z^j_{i-1} (resp. $z^j_{i-1}{}'$) the K-closed zones appearing before the i-th transition in w^j_{zr} (resp. $w^k_{zr}{}'$) and Δ^j_i (resp. $(\Delta^k_i)'$) the abstract delays leading to it we have by Lemma 5.5.34:

$$z^j_{i-1} = \{\mathbf{0}\} \nearrow + \sum_{l=1}^{i} \Delta^j_l \times dir^i_l(\mathsf{resets}(w^j_{zr}))$$

$$z^j_{i-1}{}' = \{\mathbf{0}\} \nearrow + \sum_{l=1}^{i} \Delta^{k\prime}_l \times dir^i_l(\mathsf{resets}(w^j_{zr})) \ .$$

Furthermore we know that dir^i_l for a given clock c and a fixed i is always 0 until a first l_0 and always 1 afterward.

It comes that for $v^j_{i-1} \in z^j_{i-1}$ and $v^k_{i-1}{}' \in z^k_{i-1}{}'$:

$$v^j_{i-1}(c) \in (\sum_{l=l_0}^{i} \Delta^j_l, \ \sum_{l=l_0}^{i} \Delta^j_l + 1)$$

$$v^k_{i-1}{}'(c) \in (\sum_{l=l_0}^{i} \Delta^{k\prime}_l, \ \sum_{l=l_0}^{i} \Delta^{k\prime}_l + 1) \ .$$

By Lemma 5.5.37 we can conclude, as we know that $\sum_{l=l_0}^{i} \Delta^{k\prime}_l$ is an integer approximation of the mean of $\sum_{l=l_0}^{i} \Delta^j_l$ and $\sum_{l=l_0}^{i} \Delta^{3-j}_l$. □

Remark 5.5.39. *Our algorithm is non-deterministic because of the approximation problem: in practice, a branch and bound approach is to be used, that we do not detail here to avoid a more convoluted discussion. Furthermore, to avoid making the algorithm more complex or longer we interrupt the computation as soon as we detect that modifying fractional parts of delays may be necessary. That necessity could be checked in the characteristic set: if all delays are greater than one it is not necessary.*

Finally, one could choose to still carry on the computation and recreate a characteristic set, even without satisfying (ii). Indeed, we simply need to replicate the invalidity, and it may happen that not completely satisfying (ii) still replicates it.

Remark 5.5.40. *Notice that for RERAs with only one clock, the alternation can be reset after each resetting transition as clock values are put back to* 0*, hence having a nul fractional part.*

The algorithm BinSearch is depicted in Algorithm 14 and uses BinComb to generate new K-closed words and corresponding characteristic sets that it turns into membership queries. It then iterates the process to find a pair of K-closed words that are as close as possible (while keeping different invalidities). For that, initial K-closed words are replaced by the results of BinComb, depending on the valid resets at maximum depth. The decisions depending on the valid resets of the replicas are summarized in Table 5.3.

Remark 5.5.41. *We do not distinguish between the stop case (there is no valid reset) and the return case (both resets are valid) in the implementation to avoid discussing the existence of a return which would make* InvalidityGuard *longer. In practice, if there is no valid reset, then a predecessor has become invalid and will be pruned, hence there is no point in finding a separating guard (the state without successors will be pruned). Hence it is important to stop the call to* InvalidityGuard *in order not to waste the computations.*

Notice that we only reason on $Rs_{|w^1_{zr}|}({w^1_{zr}}')$ because they are the same as $Rs_{|w^1_{zr}|}({w^2_{zr}}')$ as these two K-closed words with resets only differ on the last reset choice. The optional argument of BinSearch is used when one wants to return, on top of the initial pair, the first time w^1_{zr} (resp. w^2_{zr}) is modified during the execution. This in turn is used by InvalidityGuard in the case where the two initial invalidities have to be enlarged and the enlargement have different valid resets at depth $|s_o| + 1$. This is because w^1_{zr} must be separated from ${w^1_{zr}}'$ and other linear combinations, but as the enlargement is not a linear combination itself, a different guard may be necessary for strict linear combinations. A simple example is given in Example 5.5.42.

Algorithm 14: Finding a guard to separate two invalidities.

BinSearch

Input: two invalid K-closed words with resets $w^1_{zr},\ w^2_{zr}$ of same size and $\mathcal{W}_1\ \mathcal{W}_2$ their two characteristic sets. An optional parameter $first$ defaulting to false.

Output: A pair (w^1, w^2) of limit linear combinations of $w^1_{zr},\ w^2_{zr}$ such that for $j \in \{1,2\}$ $Rs_{|w^1_{zr}|}(w^j) = Rs_{|w^1_{zr}|}(w^2_{zr})$. If $first$ was passed as argument a second pair $(w^1_{first}, w^2_{first})$ corresponding to the first strict combinations such that $Rs_{|w^1_{zr}|}(w^j_{first}) = Rs_{|w^1_{zr}|}(w^j_{zr})$.

if $first$ **then** `// Initializes the first combinations`
  Let $w^1_{first} := w^1_{zr}$ and $w^2_{first} = w^2_{zr}$.
  Let $f^1, f^2 := false$
while *true* **do**
  $w^1_{zr}{}', \mathcal{W}'_1, w^2_{zr}{}', \mathcal{W}'_2 := \mathsf{BinComb}(w^1_{zr}, \mathcal{W}_1, w^2_{zr}, \mathcal{W}_2)$
  for $(w_{tr}, w'_{tr}) \in \mathcal{W}'_1 \cup \mathcal{W}'_2$ **do** `// Query the characteristic sets`
    Request(w_{tr})
    Request(w'_{tr})
  for $j \in \{1,2\}$ **do**
    if $Rs_{|w^1_{zr}|}(w^j_{zr}{}') = Rs_{|w^1_{zr}|}(w^j_{zr})$ **then**
      if $w^j_{zr}{}' \neq w^j_{zr}$ **then**
        $w^j_{zr} := w^j_{zr}{}'$
        Let $\mathcal{W}_j$ be the characteristic set for w^j_{zr}.
        if $\neg f^j$ **then**
          $f^j := true$
          $w^j_{first} := w^j_{zr}$.
      else `// BinComb does not manage to progress anymore`
        Break While
  if $|Rs_{|w^1_{zr}|}(w^1_{zr}{}')| \neq 1$ **then**
    `// Invalidity lost or propagated to the parent`
    Break
if $first$ **then**
  Return $(w^1_{zr}, w^2_{zr}), (w^1_{first}, w^2_{first})$
else
  Return (w^1_{zr}, w^2_{zr})

$Rs_{\lvert w^1_{zr}\rvert}(w^1_{zr}{}')$	$\emptyset$	$Rs_{\lvert s_o\rvert+1}(w^1_{zr})$	$Rs_{\lvert s_o\rvert+1}(w^2_{zr})$	$\{\top, \bot\}$
Action	Stop	$w^1_{zr} \leftarrow w^1_{zr}{}'$	$w^2_{zr} \leftarrow w^2_{zr}{}'$	return (w^1_{zr}, w^2_{zr})

Table 5.3: The effect of BinSearch depending on the relative valid resets of the replicas.

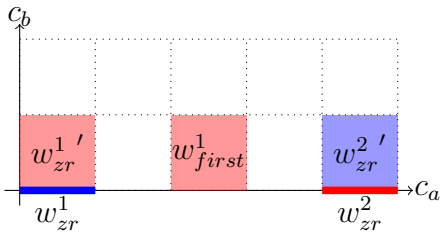


Figure 5.19: Binary search between enlarged zones.

Example 5.5.42. *Consider two K-closed words w^1_{zr} w^2_{zr} (of same size, agreeing on their actions and resets except for the last reset) going through the zones represented in Figure 5.19 at a given depth. Suppose that their enlarged versions $w^{1\,\prime}_{zr}$ $w^{2\,\prime}_{zr}$ are such that $Rs_{|w^1_{zr}|}(w^{1\,\prime}_{zr}) = w^2_{zr}$ and $Rs_{|w^2_{zr}|}(w^{1\,\prime}_{zr}) = w^1_{zr}$. In this case a guard has to be introduced between each initial invalidity and its enlarged version.*

Furthermore, a binary search is made between $w^{1\,\prime}_{zr}$ and $w^{2\,\prime}_{zr}$. Suppose that it leads to w^1_{first} such that $Rs_{|w^1_{zr}|}(w^1_{first}) = w^2_{zr}$. Then w^1_{zr} and w^1_{first} must be separated by a guard, which cannot be the same than the one separating w^1_{zr} and $w^{1\,\prime}_{zr}$ (in the example depicted on Figure 5.19). Thus a new guard is necessary.

We know by Propositions 5.5.27, 5.5.31 and 5.5.36 that InvalidityGuard respects constraints (i) and (ii) except for collapsing K-equivalent classes during the enlargement phase as explained in Example 5.5.32. It remains to show that this algorithm terminates and respects constraint (iii). We first prove termination.

Proposition 5.5.43. *Consider s_d a decision state of a TLG $\mathcal{N}$ succeeding to a* valid *observation state, such that both its successors have been pruned due to invalidities characterised by the set $\mathcal{W}_1$ (resp. $\mathcal{W}_2$) corresponding to the K-closed word w^1_{zr} (resp. w^2_{zr}).*

Then a call to InvalidityGuard$(s_d, \mathcal{W}_1, \mathcal{W}_2)$ *terminates in*

$$\mathcal{O}(|\mathcal{W}_1^{\max} \cup \mathcal{W}_2^{\max}| \times \log(\delta^{\max}) \times n^{\max})$$

where $\mathcal{W}_1^{\max}$, $\mathcal{W}_2^{\max}$ are the largest characteristic sets built during the algorithm, $\delta_{\max}$ is the maximal difference between delays $\Delta(z^1_{i-1}, z^{1\,\prime}_{i-1})$ in behav(w^1_{zr}) *and* behav(w^2_{zr}) *and $n_{\max}$ is the maximal length of observations in $\mathcal{W}_1 \cup \mathcal{W}_2$.*

Such call makes up to

$$\mathcal{O}(|\mathcal{W}_1 \cup \mathcal{W}_2| \times \log(\delta_{\max}))$$

membership queries before termination.

Proof. The initial enlargement phase has a time complexity $\mathcal{O}(|\mathcal{W}_1 \cup \mathcal{W}_2| \times n_{\max})$ dominated by the cost of the copy of the characteristic sets in the calls to Enlarge and makes $|\mathcal{W}_1 \cup \mathcal{W}_2|$ calls to Request, resulting in (up to) the same amount of membership queries. After that step, each loop iteration of the main while loop in BinSearch has the same $\mathcal{O}(|\mathcal{W}_1 \cup \mathcal{W}_2| \times n_{\max})$ complexity (due to the copies in BinComb) and makes up to $|\mathcal{W}_1 \cup \mathcal{W}_2|$ membership queries, with the current values of $\mathcal{W}_1$ and $\mathcal{W}_2$ in the algorithm. For any iteration of that loop, either the algorithm terminates or exactly one of the two replicas replace its original invalid K-closed word (and is not equal to it). Hence as the difference of distances is divided by 2 (by the choice of δ) we have our result. □

Remark 5.5.44. *Notice that the size of $\mathcal{W}_1$, $\mathcal{W}_2$ is bounded during the execution of the algorithm because new characteristic sets are built of replica of ancient ones. In the worst case, a $\mathcal{W}$ can have $2^{n_{max}-|s_o|}$ pairs with $n_{\max}$ being constant through the algorithm and s_o the predecessor of s_d.*

To show that InvalidityGuard respects the constraints (iii) we first show that BinSearch respects them (using the related properties on BinComb). This will then be used to conclude.

Lemma 5.5.45. *At any iteration of the While loop of BinSearch, any (enlargement closed) validity guard $(g, i, {w^j_{zr}}', w^{3-j}_{zr})$ with $j \in \{1, 2\}$ is also a separating guard for (w^j_{zr}, w^{3-j}_{zr}) such that ${z^j_i}' \in g$ if and only if $z^j_i \in g$ with z^j_i (resp. ${z^j_i}'$) the i-th zone of w^j_{zr} (resp. ${w^j_{zr}}'$).*

Proof. This property holds by induction by using Lemma 5.5.38 as ${w^j_{zr}}'$ is taken from the results of a call to BinComb passing $w^{1/2}_{zr}$ as arguments. □

The next property shows that the guards introduced by InvalidityGuard indeed separate the different invalidities, as a corollary of Lemma 5.5.45 and Lemma 5.5.30.

Proposition 5.5.46. *Let Acc be an acceptance function and $\mathcal{N}$ be a TLG implementing the valid part of Acc. Consider $s_d = (s_o, g, a) \in {S_d}^{\mathcal{N}}$, such that s_o is valid and the two possible successors of s_d by $\top$ and $\bot$ are invalid. Consider the characteristic sets $\mathcal{W}_1$, $\mathcal{W}_2$ of the two invalidities.*

We note $\mathcal{W}^\top$ (resp. $\mathcal{W}^\bot$) the set of timed words with resets which prefixes are invalid after the sequence of resets $\mathsf{resets}(s_o).\top$ (resp. $\mathsf{resets}(s_o).\bot$) encountered during the call to $\mathsf{InvalidityGuard}(s_d, \mathcal{W}_1, \mathcal{W}_2)$.

Upon the termination of a call $\mathsf{InvalidityGuard}(s_d, \mathcal{W}_1, \mathcal{W}_2)$, a set of validity guards is added to $\mathcal{G}_{Val}$ that separate $\mathcal{W}^\top$ and $\mathcal{W}^\bot$.

Proof. During this proof, we use the fact that MakeGuard introduces guards that indeed separate its arguments, and are enlargement closed if the pair to be separated is made of open K-closed words with resets.

We divide the proof in cases depending on wether w^1_{zr} and w^2_{zr} are open and on the different cases of the algorithm.

- If both w^1_{zr} and w^2_{zr} are open, InvalidityGuard defaults to a call to BinSearch. By Lemma 5.5.45 and using the fact that we replace invalidities by invalidities of the same type, we know that this call separates correctly the invalidities encountered.

- If exactly one of the two words is open, consider without loss of generality that it is w^1_{zr}. Then ${w^2_{zr}}'$ is constructed by Enlarge from w^2_{zr}. If it has the same valid resets than w^2_{zr} at depth $|w^2_{zr}|$ then we conclude using the case where both words are open, with the addition that the guard returned between two open K-closed sets with resets is enlargement closed. Thus, by Lemma 5.5.30 w^2_{zr} is also separated by the guard and a unique guard is returned by MakeGuard.

  Else, if $Rs_{|w^2_{zr}|}({w^2_{zr}}') = Rs_{|w^2_{zr}|}(w^1_{zr})$ then $\mathcal{W}^{Rs_{|w^2_{zr}|}(w^2_{zr})} = \{w^2_{zr}\}$ and $\mathcal{W}^{\neg Rs_{|w^2_{zr}|}(w^2_{zr})} = \{w^1_{zr}, {w^2_{zr}}'\}$ and this corresponds to the call to MakeGuard made.

  Finally if $|Rs_{|w^2_{zr}|}({w^2_{zr}}')| \neq 1$ (either the invalidity has propagated to the predecessor or the words are not invalid) it suffices to separate w^1_{zr} and w^2_{zr}, which is done by MakeGuard.

- If both initial invalidities are not open, then both are enlarged through Enlarge, generating ${w^1_{zr}}'$ and ${w^2_{zr}}'$. The cases in which one of the two (for $j \in \{1, 2\}$) has $|Rs_{|w^2_{zr}|}({w^j_{zr}}')| \neq 1$ are treated by directly introducing guards that separate $\mathcal{W}^\top$ and $\mathcal{W}^\bot$ has done for the previous case (when only one word is enlarged). Similarly, the case where both enlargements behave like the original one uses the result on binary search and enlargement closed guards to conclude, as done in the previous case. In the last case, where $Rs_{|w^2_{zr}|}({w^2_{zr}}') = Rs_{|w^2_{zr}|}(w^1_{zr})$ and $Rs_{|w^2_{zr}|}({w^1_{zr}}') = Rs_{|w^2_{zr}|}(w^2_{zr})$, more guards are required to separate the different parts of $\mathcal{W}^\top$ and $\mathcal{W}^\bot$. We suppose without loss of generality that $w^1_{zr} \in \mathcal{W}^\top$. Using this we can partition the sets of observations as

$$\mathcal{W}^\top = \{w^1_{zr}, {w^2_{zr}}'\} \cup \mathcal{W}^\top_{\mathsf{strict}}$$

$$\mathcal{W}^\bot = \{w^2_{zr}, {w^1_{zr}}'\} \cup \mathcal{W}^\bot_{\mathsf{strict}}$$

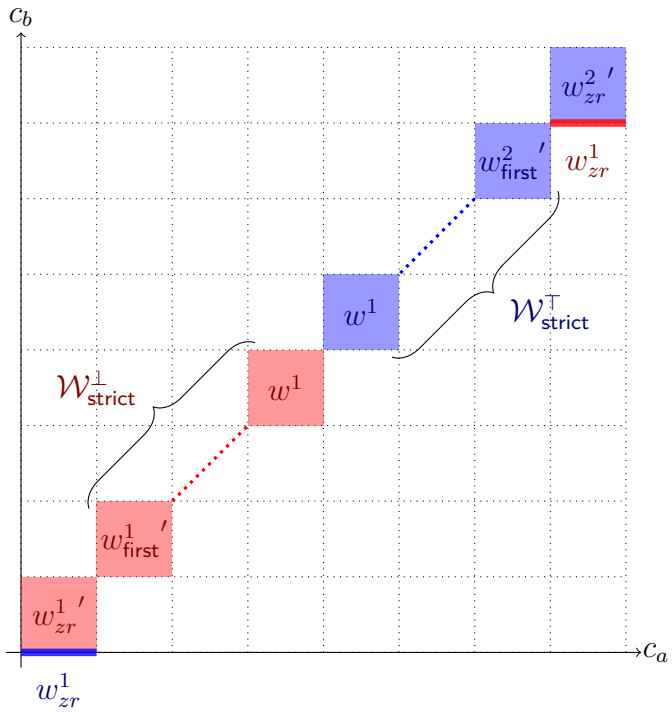


Figure 5.20: An example of $\mathcal{W}^{\top}$ and $\mathcal{W}^{\perp}$ in the case where the enlarged timed words with resets have the complementary valid resets of their initial ones.

with $\mathcal{W}^{\top}_{\mathsf{strict}}$ the set of strict linear combinations (with w^2_{first} the first one and w^2 the last) and respectively for $\mathcal{W}^{\perp}_{\mathsf{strict}}$.

An example of the structure of $\mathcal{W}^{\top}$ and $\mathcal{W}^{\perp}$ is proposed Figure 5.20 (the figure shoes $\mathcal{W}^{\top}_{\mathsf{strict}}$ and $\mathcal{W}^{\top}_{\perp}$ as simple linear combinations instead of means for conciseness.)

We show that every subsets of $\mathcal{W}^{\top}$ and $\mathcal{W}^{\perp}$ are separated by the guards introduced. For any $w^{\top} \in \mathcal{W}^{\top}_{\mathsf{strict}}$ and $w^{\perp} \in \mathcal{W}^{\perp}_{\mathsf{strict}}$, by Lemma 5.5.45 the guard separating w^1 and w^2 separates them (as w^1 and w^2 are the last linear combinations constructed through BinSearch). The same goes for w^1_{zr} and w^2_{zr} as well as for the pair $(w^{1\,\prime}_{zr}, w^{2\,\prime}_{zr})$. In the same way, $w^{1\,\prime}_{zr}$ is separated from $\mathcal{W}^{\perp}_{\mathsf{strict}}$ by the guard separating w^1 and w^2. $w^{2\,\prime}_{zr}$ is separated from $\mathcal{W}^{\top}_{\mathsf{strict}}$ in the same way. w^1_{zr} is explicitly separated from $w^{1\,\prime}_{zr}$ and w^1_{first} by MakeGuard, and in the same way w^2_{zr} is explicitly separated from $w^{2\,\prime}_{zr}$ and w^2_{first}. Hence $\mathcal{W}^{\top}$ and $\mathcal{W}^{\perp}$ are indeed separated by the guards introduced by InvalidityGuard.

□

The proposed method relying on a binary search on open zones is thus satisfying, as

it indeed solves the problem by introducing fitting separating guards and handling the intricate structures of K-closed zones. It yet has some short-comings, namely the risk of collapsing K-equivalence classes for different resets during the enlargement phase and the possibility to stop the binary search when a specific manipulation on small distances between K-closed zones happens. As argued in the previous discussion and shown in examples, these are consequences of the structure of equivalence classes of valuations and would require a completely different approach to be handled (or rather avoided).

Now that the search of all different necessary guards has been presented (through AdjPair and InvalidityGuard) we discuss the introduction of those guards in the TLG and the rebuilding of subtrees that ensues.

5.5.4 Rebuilding the graph

To rebuild a subtree of a TLG is to introduce new guards using consistency and validity guards only when necessary, and re-propagate the informations in the new guarded words with resets they satisfy. We use Rebuild (Algorithm 15) for this.

Intuitively, Rebuild only introduces guards "when needed", which is formalized by the following *well-guardedness* property.

Definition 5.5.47. *A timed language graph $\mathcal{N}$ is said* well guarded *if, for all transitions $(s_l, (g, a), s_d) \in E_{\mathcal{N}}$ and all constraints $c \prec k$ in g, either there is a consistency guard $(c \prec k, |s_l| + 1, w_{tr}, w'_{tr})$ with w_{tr} adjacent to w'_{tr} such that both pass by $s_l.(g, a)$* or *$(c \prec k, |s_l| + 1, \mathcal{W}, \mathcal{W}')$ a validity guard with $w_{zr} \in \mathcal{W}$ and $w'_{zr} \in \mathcal{W}'$ passing through $s_l.(g, a)$.*

Rebuild is described in Algorithm 15. It is called on a valid and consistent observation state of the TLG and erases the subtree rooted in it, before reconstructing a valid and consistent subtree with respect to the current observation function. It uses calls to FindGuard (Algorithm 16) that handles the guard creation.

Remark 5.5.48. *In the rebuild function, we use Request on guarded words with resets instead of timed word. The extension is quite simple thanks to the resets, as searching in $\mathcal{M}$ if an observation modelling the argument exists is only a dive in the tree, and if none is found, making an membership query from the last guess is the same.*

Remark 5.5.49. *As written, Rebuild completely erases the subtree and then reconstructs it. An obvious optimization is to only suppress transitions and nodes when necessary to avoid invalidity or inconsistency. We do not develop this here to keep the algorithm short and simple.*

Algorithm 15: Rebuilds a subtree of $\mathcal{N}$ to handle consistency.

```
1  Rebuild
   Input: A (valid) language state s_l
2  Suppress recursively all successors of s_l
3  for a ∈ Σ such that there is at least an observation passing s_l.(a, ⊤) do
4  |  for each guard g ∈ FindGuard(s_l, a, true) do
5  |  |  create s_d := (s_l, a, g) and (s_l, a, g, s_d) ∈ E_Σ
6  |  |  if s_l.(a, g, ⊤) is not invalid then
7  |  |  |  create s'_l := s_l.(a, g, ⊤)
8  |  |  |  label(s''_l) := Request(s''_l)
9  |  |  |  Rebuild (s'_l)
10 |  |  if s_l.(a, g, ⊥) is not invalid then
11 |  |  |  create s''_l := s_l.(a, g, ⊥)
12 |  |  |  label(s''_l) := Request(s''_l)
13 |  |  |  Rebuild (s''_l)
```

Algorithm 16: Find a partition in guards to be applied to an action in a language node.

```
1  FindGuard
   Input: A (valid and consistent) language state s_l, an action a and a guard g
   Output: a partition of g
2  if there is a consistency guard (g', |s_l| + 1, w_tr, w'_tr) with (w_tr, w'_tr) passing s_l.(a, g)
     then
3  |  return FindGuard(s_l, a, g ∧ g') ∪ FindGuard(s_l, a, g ∧ ¬g')
4  else
5  |  if there is a validity guard (g', |s_l| + 1, W_zr, W'_zr) with (w_zr, w'_zr) ∈ W_zr × W'_zr
   |    passing s_l.(a, g) then
6  |  |  return FindGuard(s_l, a, g ∧ g') ∪ FindGuard(s_l, a, g ∧ ¬g')
7  |  else
8  |  |  return {g}
```

Remark 5.5.50. *Notice that, on top of validity guards being chosen* as deep as possible *in the pairs of words and the consistency guard* as close to the root as possible, *FindGuard introduces all consistency guards before the validity guards at a given depth. This entails that validity guards are used only when absolutely necessary and matches the intuition that invalidity should be used to inform reset guesses and not primarily to construct the model.*

Rebuild constructs a complete, consistent and well-guarded subtree if it is called high enough in the tree. This is proven in Proposition 5.5.52 that assumes all sources of

inconsistencies and decisions states without successors in the subtree have been handled. To verify this would require a prior check.

In practice it may be more efficient to rebuild without that verification, thus handling all detected issues but rebuilding a potentially inconsistent subtree, or with some decisions states having no successors. This would function as a detection method, allowing to treat those (newly discovered) issues and rebuilding again.

We prove in Lemma 5.5.51 that any validity or consistency guard added in $\mathcal{G}_{Cons}$, $\mathcal{G}_{Val}$ indeed separates all the observations and K-closed words that must be separated, independently of other guards introduced by the algorithm.

Lemma 5.5.51. *Consider a call to* AdjPair *(resp.* InvalidityGuard*) in which two sets of timed words with resets $\mathcal{W}_+$ and $\mathcal{W}_-$ respectively accepting and non-accepting (resp. $\mathcal{W}_\top$ and $\mathcal{W}_\bot$ sets of K-closed words with resets respectively invalid for a sequence of resets terminated by $\top$ and $\bot$) have been requested.*

Then independently of the other guards in $\mathcal{G}_{Cons}$ no pairs in $\mathcal{W}_+ \times \mathcal{W}_-$ can end in the same language state in the subtree rebuilt by Rebuild*. Respectively independently of the other guards in $\mathcal{G}_{Val}$ no pairs in $\mathcal{W}_\top \times \mathcal{W}_\bot$ can end in language states child to the same decision state in the subtree rebuilt by* Rebuild*.*

Proof. We separate the proofs for consistency and validity. In the case of AdjPair we know that all valuations in words $\mathcal{W}_+ \cup \mathcal{W}_-$ are linear combinations of valuations in the arguments of AdjPair (by Proposition 5.3.8). Thus any guard separating $\mathcal{W}_+ \cup \mathcal{W}_-$ either separates exactly $\mathcal{W}_+$ and $\mathcal{W}_-$ or separates only one of those two sets in two and the half that stays with the other set contains the observation that carries the consistency guard (as it is the limit linear combination). By Proposition 5.5.9 we have that the sets are separated.

In the case of InvalidityGuard, we first assume that the call did not end with s_o the predecessor of the argument s_d being invalid. We then know by Lemma 5.5.34 that we can characterize the zones in the K-closed words reached during the binary search as linear combinations (based on Δ) of the initial enlarged zones. We thus have the same result as for AdjPair *for the open-zones only.* But notice that either a specific validity guard is introduced between the two initial (non-open)K-closed words or they are added to the set for which the validity guard must be raised. Thus in any case, as the validity guard indeed separates pairs of $\mathcal{W}_\top \times \mathcal{W}_\bot$ by Proposition 5.5.46 we have our result.

Now consider the case where s_o became invalid. In that case, it means that an open K-closed word w_{zr} of size $|s_o|$ was found invalid for a given continuation (z, a) and both

resets $\top, \bot$. In that case if some of the open zones in $\mathcal{W}_\top \cup \mathcal{W}_\bot$ appear in the subtree it means that a guard has been introduced separating it from w_{zr}. As w_{zr} was the last linear combination constructed when the algorithm stopped, it means that it is separated from the open zones that do not agree with its $Rs_{|s_o|+1}$. Hence again we have our result *for the open-zones only*. Notice that even when s_o's invalidity is detected, guards are added to $\mathcal{G}_{Val}$ to separate non-open K-closed words with resets between them and with open K-closed words. We hence have our result. □

Notice that the condition for invalidities not to be covered by the children of a given decision state corresponds to the condition of not being in the same language state for inconsistencies. Indeed the resets of these words are different at the last depth only.

Proposition 5.5.52. *Consider a valid and consistent language state $s_l^{\mathcal{N}}$ of the TLG. After adding consistency (resp. validity) guards to $\mathcal{G}_{Cons}$ (resp. $\mathcal{G}_{Val}$) to separate every pair causing an inconsistency (resp. forcing the pruning of all successors of a decision state), if every guard added is at depth strictly greater than $|s_l^{\mathcal{N}}|$ running* Rebuild$(s_l^{\mathcal{N}})$ *constructs a subtree rooted in $s_l^{\mathcal{N}}$ argument that is* maximal, complete, well-grounded, consistent *with respect to the current acceptance function* Acc *and* well-guarded. *Notably, no decision states are left without successors.*

Proof. First notice that the subtree built is indeed a TLG subtree as it respects the alternation of decision and observation states, the type of edges, the constraints of guards (*i.e.* a call to FindGuard returns a partition of its original guard and it is always called on true) and all leaves are observation states. Indeed, by Lemma 5.5.51 we know that no pair encountered during an call to InvalidityGuard can end in both children of a decision state. By hypothesis all such pairs have been treated hence we have our result.

We show each property independently.

Maximality Maximality comes directly from the structure of Rebuild where all successors of a constructed decision state are considered for building and only abandoned if invalid.

Completeness Consider an observation $w_t \in$ Acc that passes the root $s_l^{\mathcal{N}}$ of the subtree. As the Rebuild function continuously calls itself as long as an observation passes the current word and no decision state is left without successors, there is by induction a language state s'_l such that $w_t \in$ words(s'_l).

Well-groundedness The construction of any new language state stops as soon as no observation continues from the current language state. As $\mathsf{Dom}(\mathsf{Acc})$ is prefix-closed in the sense of K-closed words (this is ensured by the TOG) we know that all previously constructed language states have an observation in their words (as the last one has one).

Consistency Consistency is ensured by Lemma 5.5.51 and the hypothesis that all pairs of words that can create an inconsistency have led to a call to AdjPair that separated them.

Well-guardedness This property is ensured by construction, as all guards introduced by FindGuard are consistency or validity guards.

□

This proposition tells us that we can keep the timed language graph up-to-date with respect to observations (*i.e.*, complete and consistent) while preserving the good properties that were ensured by the previous algorithms.

This concludes the presentation of the algorithms updating the data-structure (TLG and TOG). It remains to show how a candidate timed automaton can be constructed from this structure.

5.6 Building a candidate timed automaton

Following the active learning approach, our purpose is to identify a subset of nodes in the language graph that will correspond to locations of the automaton, and then fold transitions according to an order on the remaining nodes. [GJP06] discusses such orders when resets are fixed. To handle RERA we first have to fix a *reset strategy* before applying the original method. This gives as many hypotheses as we have strategies.

Reset selection. We present the general framework but do not discuss good strategies in the following. Such strategies would rely on heuristics.

Definition 5.6.1. *A* reset strategy *over a timed language graph $\mathcal{N}$ is a mapping $\pi\colon S_d \to \{\top, \bot\}$, assigning a decision to each decision states.*

A reset strategy π is said admissible *if for any state s_d, there is a language state s_l such that $(s_d, \pi(s_d), s_l) \in E$.*

An admissible reset strategy is used to prune the language graph in such a way that only one reset combination is considered for each transition. The effect of an admissible reset strategy π on its timed language graph $\mathcal{N}$ is the TLG $\pi(\mathcal{N})$ defined from $\mathcal{N}$ by keeping only outgoing transitions from decision states that agree with π. We call this TLG the *resulting graph* of π.

It can be seen quite directly that a resulting graph always has exactly one successor to each decision state. Using this, we can notice that those resulting graphs are very close to timed decision trees of [GJP06], in which no decision states exist and the transitions from language states to language states directly hold the (only possible) reset.

Having an admissible strategy thus is the basis of the construction of a hypothesis. The way our data structure is handled suffices to ensure that once all observations are integrated and the TLG is rebuilt, an admissible strategy always exists.

Proposition 5.6.2. *In a timed language graph constructed using the* FindPath *and* Rebuild *algorithms and where every scheduled call to* Rebuild *has been done, there always exists at least one admissible reset strategy.*

Proof. To ensure that an admissible reset strategy exists, one only needs to check that every decision state has at least one successor. We only prune the graph in the SearchPrune algorithm, and this algorithm schedules a call to Rebuild when no successors exist for a decision state. As Rebuild constructs a subtree where all decision states have at least a successor (thanks to the FindGuard function that explicitly checks for this), we have our property. □

Orders and folding. Once an admissible reset strategy is fixed, it is possible to fold the resulting graph into a RERA. This is made through the use of a preorder on states: we want to find a maximal subset for this order.

We define the *height* of a language state s_l, noted $\mathsf{height}(s_l)$, as the height of the subtree it is the root of. A preorder $\sqsubseteq$ on language states is said *height-monotone* when $s_l \sqsubseteq s'_l$ implies $\mathsf{height}(s_l) \leq \mathsf{height}(s'_l)$.

Definition 5.6.3. *Let $\mathcal{N}$ be a timed language graph and $\sqsubseteq$ a preorder on its language states. A prefix-closed subset U of $\mathcal{N}$ is called* $\sqsubseteq$-closed *if $s_l \sqsubseteq U$ for all successors of U and* $\sqsubseteq$-unique *if for all $s_l, s'_l \in U$, $s_l \neq s'_l \Rightarrow \neg(s_l \sqsubseteq s'_l)$.*

$\sqsubseteq$-closedness is used to construct a RERA by folding the successors of U into comparable states of U. $\sqsubseteq$-uniqueness is useful to bound the number of states in U and thus the size of the resulting automaton.

The following lemma (Lemma 6.2 in [GJ08], a long version of [GJP06] appearing in O.Grinchtein's thesis [Gri08]) ensures that there always exists a satisfying set of states U. For its constructive proof, we refer the reader to the original paper.

Lemma 5.6.4. *Let $\sqsubseteq$ be a height-monotone preorder on states in a resulting graph $\pi(\mathcal{N})$. Then there exists a $\sqsubseteq$-closed and $\sqsubseteq$-unique prefix-closed subset of the language states of $\pi(\mathcal{N})$.*

Using such a subset, we can fold the resulting graph into a RERA as follows:

Definition 5.6.5. *Let $\mathcal{N}$ be a consistent TLG for Acc, π an admissible reset strategy and $\sqsubseteq$ a preorder on language states of $\pi(\mathcal{N})$. Consider a $\sqsubseteq$-unique, $\sqsubseteq$-closed and prefix-closed subset U of $\pi(\mathcal{N})$. Then a $U_{\sqsubseteq}$-*merging *of $\mathcal{N}$ according to π is a RERA $(U, \epsilon, C, E, \mathsf{Accept})$ such that $\mathsf{Accept} = \{u \in U \mid \mathsf{label}(u) = \{+\}\}$ and for any observation state $u.(g, a, r)$ of $\pi(\mathcal{N})$ with $u \in U$, there is exactly one edge of the form $(u, (g, a, r), u') \in E$ with $u.(g, a, r) \sqsubseteq u'$. Notice that, by the second condition, a $U_{\sqsubseteq}$-merging RERA is deterministic.*

Furthermore, if the observation structure is complete, a $U_{\sqsubseteq}$-merging generalizes the observations obtained so far.

Constructing a candidate RERA. Using the results of the previous subsections, we can now construct a candidate RERA from our observation structure. All admissible reset strategies can be constructed by branch and bound. Then a merging is constructed for each resulting graph, and equivalence queries are launched.

For each of the RERA constructed by merging, either a counter-example will be returned by the equivalence query, or the candidate is deemed correct. In the latter case, we return this RERA; in the former case, we include the counter-example in our observation structure and repeat the process.

5.7 Conclusion

In this chapter, we propose an active learning method for deterministic reset-optional event recording automata. We add a key feature to the state of the art: invalidity, that allows to

detect incorrect guesses of resets when they are not tied to observations. This required to rework all the data structures and algorithms involved to handle invalidity on-the-fly. Most importantly, this brings the lacking notion to scale up to the class of deterministic timed automata (DTAs). Interestingly, the algorithmic handling of invalidity highlights the complex dynamics of undistinguishable classes of valuations and observations, formalized as K-closed zones and K-closed words with resets.

A clear future work is to generalize this method to actually handle DTAs. This mostly requires to handle resets of sets of clocks instead of single ones. As the complexity would be greatly increased, this calls for some optimization. A promising addition would be to use an implicit structure. Instead of storing all possible reset configurations, only storing a small set of them at the same time would decrease the memory cost. As the models are built directly from observations, and not from previous states, the computational overhead may be limited. Another interesting trail for future development is to find a way to build a timed automaton from the observation structure that exploits the different admissible reset strategies without building all of them. Works on approximate determinization of timed automata through games [Ber+15] deal with similar problems and offer interesting leads. Finally, in [GJP06], the authors propose to refine the adjacent pairs into *critical pairs*, that have a minimal set of differences. This allows to better identify the guards to be added, and thus can have a positive effect on both the size of the constructed models and the computational cost. Sadly, no precise procedure is given to construct the pairs, so creating one would be beneficial to the approach. More generally, studying the efficiency of this algorithm and of the variants proposed as future work could help better understand the applicability and bottlenecks of the approach.

An interesting branch would be to try to integrate invalidity and its characterization to TL^*. This could allow to benefit from the algorithm efficiency. One of the main difficulties in this line of research would probably be to integrate the abstractions we propose into TL^* that uses the ERA structure to remain at the level of guarded words with resets even for membership queries. In a similar fashion, the TL^* algorithm relies on prior learning of the untimed language of the timed automaton. Yet, the untimed language construction is existential (*i.e.* an untimed word is accepted if it is the projection of an accepted timed word), and the implementation of untimed equivalence of membership queries could be difficult (or require a far greater complexity) in a timed setting, which is not discussed in the original paper.

Conclusion

> So comes snow after fire, and even dragons have their ending.
>
> — J.R.R. Tolkien "The Hobbit"

This thesis has tackled different problems related to the interactions between formal models of timed systems (as timed automata and some variants) and sheer reality. Chronologically, the research effort started from a discussion of the action of model-based agents on reality, through formal game theory. This lead to the realization that some informations lack in any reasonable model of a system. It thus makes sense to try to recover those information from the real system we are already interacting with, which can take the form of various (model) learning techniques. The hazard (and the discovery of an exciting body of research compiled during Olga Grinchtein's thesis [Gri08]) led the author to primarily consider active learning.

Contributions The contributions of this thesis focus on a very formal approach of reality-models interactions, with their upsides, such as a fine control and understanding of the behaviours, and their downsides, mainly the complexity that can arise and the formal (or perhaps mathematical) limitations that sprout from the model choices. The main contributions are three-fold:

- A model-based conformance testing method for timed automata with inputs and outputs that relies on game theory, and specifically so-called *rank lowering strategies* for *hard games*, allowing to plan even in situations where victory cannot be achieved without the opponent's cooperation. These strategies allow a quantitative take on such games, minimizing the reliance on cooperation and planing around it. We discuss their interest both in a restricted "fair" setting where they can be proved to be winning and in very general situations where the game difficulty forbids to hope for an assured victory. In this case, these strategies can nevertheless offer an interesting way toward the objective while trying to resist to failure.
- A discussion on a timed markings, a structure allowing to pre-compute a finite

representation of a system set of configurations that can be reached after a given observation, at least for 1-clock automata. We extend the theoretical basis of this structure to general (n-clock) timed automata, but do not propose a sufficient finitely representable class for them. This contribution has a double interest: it allows to avoid the undecidable determinisation problem (in the 1-clock case) by defaulting to a sufficient representation and gives some insight on the structure of time dynamics in timed automata.

- An active learning method for deterministic reset-optional event recoding automata (DRERAs), a new quite general subclass of deterministic timed automata that displays both a high dimensionality and unobservable clock resets (and thus time dynamic). The key of this contribution is the notion of *(in)validity* that allows to characterize reset choices that can or cannot be part of a model of a set of observations. On top of that notion, we propose data structures to handle the learning of DRERAs, focusing on the representation of their behaviours, and the algorithms necessary to construct, update and exploit them. We believe that the notion of invalidity is the last previously missing piece to handle the learning of deterministic timed automata.

Modeling as abstract interpretation A guiding thread of the different axes of this thesis is the correct formalisation of the various abstractions underpinning timed automata in term of level of observation—syntax with paths; semantics with runs; behaviours with abstract runs and timed words with resets; observations with timed words—and precision—from simple runs (and other observation levels) to minimally distinguishable sets (region or K-equivalent) to general sets defined by paths. These different representations, and the abstraction functions that link them, are presented from the beginning (see Figure 1.4) and further developed in the case of learning (see Figure 5.3). The abstraction functions play an important role in the first chapters of the thesis, while the invert-abstractions are at the core of the discussion on active learning.

Even more generally, and perhaps more notably, laying out these abstractions and their relations is akin to *abstract interpretation* as the correct representation (*i.e.* abstract domain) and their properties are central to state estimation (which builds on a new abstract domain) and learning (that sees paths and equivalence classes as an abstract framework that constrains the observations interpretation).

One could see the interactions between models and reality as an interpretation of the "real semantic" (*i.e.* what is happening in the real world) into specific sound approximations (*i.e.* formal objects) that both abstract parts of the reality that are deemed irrelevant for the task at hands and organize the relevant information.

Perspectives The different contributions of this thesis open multiple perspectives for future developments. Firstly, our active learning method is limited to DRERAs, but as discussed earlier, reunites all the key elements to learn deterministic timed automata, thus that extension should be made, in regard for the interest of having a formal learning technique general enough for all learning models. Developing other learning methods for (subclasses of) timed automata could also be of interest, notably reinforcement learning, that is simultaneously historically model-based, linked to formal communities and extremely in vogue among machine learning specialists.

Completing the generalization of the state estimation method of Chapter 4 to general timed automata could also have great practical interests, although it would require new ideas that are not introduced here, as discussed at the end of the chapter.

Finally, developing formal game-based test generation for timed systems would both be interesting for its theoretical implications and practical consequences, due to the far reaching use of tests in the industry. The author believes that further developing such methods greatly depends on the association of testing and learning in mixed methods that would allow to start testing from an initial (partial) model and carry on with increasingly complex and accurate models synthesized through learning. This would allow both to verify properties of the implementation and to construct a model of the system (hopefully of great quality) that helps understanding and analyzing it.

Related to this last point, but outside the scope of timed models, the abstraction based approach supported by this thesis could be used in other domains to intertwine formal learning and acting (*i.e.* testing, control, diagnostic...), especially for systems that have rich behaviours such as *e.g.* models with data, a great number of agents (potentially parametric and/or dynamic) or monads.

BIBLIOGRAPHY

[ACD90] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking for real-time systems", *in*: *Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990, pp. 414–425, DOI: 10.1109/LICS.1990.113766.

[AD94] Rajeev Alur and David L. Dill, "A Theory of Timed Automata", *in*: *Theoretical Computer Science* 126.*2* (Apr. 1994), pp. 183–235.

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger, "Event-Clock Automata: A Determinizable Class of Timed Automata", *in*: *Theoretical Computer Science* 211.*1-2* (Jan. 1999), pp. 253–273, DOI: 10.1016/S0304-3975(97)00173-4.

[Aic+18] Bernhard K. Aichernig, Wojciech Mostowski, Mohammad Reza Mousavi, Martin Tappler, and Masoumeh Taromirad, "Model Learning and Model-Based Testing", English, *in*: *Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172*, ed. by Amel Bennaceur, Reiner Hähnle, and Karl Meinke, Lecture Notes in Computer Science, Springer Nature, July 2018, pp. 74–100, ISBN: 978-3-319-96561-1, DOI: 10.1007/978-3-319-96562-8_3.

[Alu+92] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho, "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems", *in*: *Hybrid Systems I*, Springer, 1992, pp. 209–229.

[Alu+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems", *in*: *Theoretical Computer Science* 138 (1995), pp. 3–34.

[An+20] Jie An, Mingshuai Chen, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang, "Learning One-Clock Timed Automata", *in*: *Proceedings of the 26th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'20) – Part I*, ed. by Armin Biere and David Parker,

vol. 12078, Lecture Notes in Computer Science, Springer, Apr. 2020, pp. 444–462, DOI: 10.1007/978-3-030-45190-5_25.

[An+21] Jie An, Lingtai Wang, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang, "Learning real-time automata", *in*: *Science China Information Sciences*, vol. 64, Science China Press, Sept. 2021, DOI: 10.1007/s11432-019-2767-4.

[And+12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat, "IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems", *in*: *FM 2012: Formal Methods*, ed. by Dimitra Giannakopoulou and Dominique Méry, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 33–36, ISBN: 978-3-642-32759-9.

[And21] Étienne André, "IMITATOR 3: Synthesis of timing parameters beyond decidability", *in*: *CAV'21*, ed. by Rustan Leino and Alexandra Silva, vol. 12759, Lecture Notes in Computer Science, Springer, 2021, pp. 1–14.

[Ang80] Dana Angluin, "Inductive Inference of Formal Languages from Positive Data", *in*: *Information and Control* 45.*2* (1980), pp. 117–135, DOI: 10.1016/S0019-9958(80)90285-5, URL: https://doi.org/10.1016/S0019-9958(80)90285-5.

[Ang87a] Dana Angluin, "Learning Regular Sets from Queries and Counterexamples", *in*: *Information and Computation* 75.*2* (1987), pp. 87–106, DOI: 10.1016/0890-5401(87)90052-6, URL: https://doi.org/10.1016/0890-5401(87)90052-6.

[Ang87b] Dana Angluin, "Queries and Concept Learning", *in*: *Machine Language* 2.*4* (1987), pp. 319–342, DOI: 10.1007/BF00116828, URL: https://doi.org/10.1007/BF00116828.

[Ang88] Dana Angluin, *Identifying languages from stochastic examples*, Research Report, 1988.

[Ang90] Dana Angluin, "Negative Results for Equivalence Queries", *in*: *Machine Language* 5 (1990), pp. 121–150, DOI: 10.1007/BF00116034, URL: https://doi.org/10.1007/BF00116034.

[Asa+98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis, "Controller Synthesis for Timed Automata", *in*: *Proceedings of the 5th IFAC Conference on System Structure and Control (SSC'98)*, vol. 31, Elsevier, July 1998, pp. 469–474.

[AV10] Fides Aarts and Frits Vaandrager, "Learning I/O Automata", *in*: *Proceedibgs of the 21st int. conf on Concurrency Theory - CONCUR 2010*, ed. by Paul Gastin and François Laroussinie, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 71–85.

[Bac+21] Giovanni Bacci, Patricia Bouyer, Uli Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Pierre-Alain Reynier, "Optimal and Robust Controller Synthesis Using Energy Timed Automata with Uncertainty", *in*: *Formal Aspects of Computing* 33.*1* (Jan. 2021), pp. 3–25, DOI: 10.1007/s00165-020-00521-4.

[Bai+09] Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye, "When Are Timed Automata Determinizable?", *in*: *Automata, Languages and Programming, 36th International Colloquium ICALP 2009*, ed. by Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, vol. 5556, Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, July 2009, pp. 43–54, ISBN: 978-3-642-02930-1.

[BB04] Laura Brandán Briones and Ed Brinksma, "A Test Generation Framework for Quiescent Real-Time Systems", *in*: *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, ed. by Jens Grabowski and Brian Nielsen, vol. 3395, Lecture Notes in Computer Science, Springer, Sept. 2004, pp. 64–78, DOI: 10.1007/978-3-540-31848-4_5.

[BCD05] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza, "Fault Diagnosis Using Timed Automata", *in*: *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'05)*, ed. by Vladimiro Sassone, vol. 3441, Lecture Notes in Computer Science, Springer, Apr. 2005, pp. 219–233, DOI: 10.1007/978-3-540-31982-5_14.

[Beh+06] Gerd Behrmann, Alexandre David, Kim G. Larsen, Paul Pettersson, Wang Yi, and Martijn Hendriks, "Uppaal 4.0", *in*: *Quantitative Evaluation of Systems -*

QEST'06: Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems, IEEE Computer Society, 2006, pp. 125–126.

[Ber+12] Nathalie Bertrand, Thierry Jéron, Amélie Stainer, and Moez Krichen, "Off-line test selection with test purposes for non-deterministic timed automata", *in*: *Logical Methods in Computer Science* 8.*4* (2012), DOI: 10.2168/LMCS-8(4:8)2012.

[Ber+15] Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen, "A game approach to determinize timed automata", *in*: *Formal Methods in System Design* 46.*1* (Feb. 2015), pp. 42–80, DOI: 10.1007/s10703-014-0220-1.

[Bér+98] B. Bérard, A. Petit, V. Diekert, and P. Gastin, "Characterization of the Expressive Power of Silent Transitions in Timed Automata", *in*: *Fundam. Informaticae* 36 (1998), pp. 145–182.

[BF72] Alan W. Biermann and Jerome A. Feldman, "On the Synthesis of Finite-State Machines from Samples of Their Behavior", *in*: *IEEE Trans. Computers* 21.*6* (1972), pp. 592–597, DOI: 10.1109/TC.1972.5009015, URL: https://doi.org/10.1109/TC.1972.5009015.

[BGP96] Béatrice Bérard, Paul Gastin, and Antoine Petit, "On the power of non-observable actions in timed automata", *in*: *STACS 96*, ed. by Claude Puech and Rüdiger Reischuk, vol. 1046, Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 255–268, ISBN: 978-3-540-49723-3.

[BJM17] Patricia Bouyer, Samy Jaziri, and Nicolas Markey, "On the determinization of timed systems", *in*: *Proceedings of the 15th International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'17)*, ed. by Alessandro Abate and Gilles Geeraerts, vol. 10419, Lecture Notes in Computer Science, Springer, Sept. 2017, pp. 25–41, DOI: 10.1007/978-3-319-65765-32.

[BJM18] Patricia Bouyer, Samy Jaziri, and Nicolas Markey, "Efficient timed diagnosis using automata with timed domains", *in*: *Proceedings of the 18th International Workshop on Runtime Verification (RV'18)*, ed. by Christian Colombo and Martin Leucker, vol. 11237, Lecture Notes in Computer Science, Springer, Nov. 2018, pp. 205–221, DOI: 10.1007/978-3-030-03769-7_12.

[BLR05] Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier, "Diagonal Constraints in Timed Automata: Forward Analysis of Timed Systems", *in*: *FORMATS 2005*, vol. 3829, Lecture Notes in Computer Science, Springer Berlin Heidelberg, Sept. 2005, pp. 112–126, ISBN: 978-3-540-30946-8, DOI: 10.1007/11603009_10.

[BM83] Bernard Berthomieu and Miguel Menasche, "An Enumerative Approach For Analyzing Time Petri Nets", *in*: *Proceedings IFIP*, Elsevier Science Publishers, 1983, pp. 41–46.

[BMS13] Patricia Bouyer, Nicolas Markey, and Ocan Sankur, "Robustness in timed automata", *in*: *Proceedings of the 7th Workshop on Reachability Problems in Computational Models (RP'13)*, ed. by Parosh Aziz Abdulla and Igor Potapov, vol. 8169, Lecture Notes in Computer Science, Springer, Sept. 2013, pp. 1–18, DOI: 10.1007/978-3-642-41036-9_1.

[Bol+09] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker, "Angluin-style learning of NFA", *in*: *IJCAI'09: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, July 2009, pp. 1004–1009.

[Bos20] Petra van den Boss, "Coverage and Games in Model-Based Testing", PhD thesis, Radboud University Nijmegen, 2020, URL: https://petravdbos.nl/publications/ThesisPetravandenBosDigital.pdf.

[Bou+08] Patricia Bouyer, Uli Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jiří Srba, "Infinite Runs in Weighted Timed Automata with Energy Constraints", *in*: *Proceedings of the 6th International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'08)*, ed. by Franck Cassez and Claude Jard, vol. 5215, Lecture Notes in Computer Science, Springer, Sept. 2008, pp. 33–47, DOI: 10.1007/978-3-540-85778-5_4.

[Bou+21] Patricia Bouyer, Léo Henry, Samy Jaziri, Thierry Jéron, and Nicolas Markey, "Diagnosing timed automata using timed markings", *in*: *International Journal on Software Tools for Technology Transfer* 23 (Mar. 2021), DOI: 10.1007/s10009-021-00606-2.

[Bou03] Patricia Bouyer, "Untameable Timed Automata!", *in*: *20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, ed. by Habib M. Alt H., vol. 2607, Lecture Notes in Computer Science, Springer Berlin Heidelberg,

Feb. 2003, pp. 620–631, ISBN: 978-3-540-00623-7, DOI: 10.1007/3-540-36494-3_54.

[Boz+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine, "Kronos: a model-checking tool for real-time systems", *in*: *Computer Aided Verification 10th International Conference, CAV'98*, ed. by Hu, Alan J.; Vardi, and Moshe Y., vol. 1427, Lecture Notes in Computer Science, Vancouver, BC, Canada: Springer, June 1998, pp. 546–549, DOI: 10.1007/BFb0028779, URL: https://hal.archives-ouvertes.fr/hal-00374784.

[BY03] Johan Bengtsson and Wang Yi, "On Clock Difference Constraints and Termination in Reachability Analysis of Timed Automata", *in*: *Formal Methods and Software Engineering*, ed. by Jin Song Dong and Jim Woodcock, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 491–503.

[BY04] Johan Bengtsson and Wang Yi, "Timed Automata: Semantics, Algorithms and Tools", *in*: *Lectures on Concurrency and Petri Nets*, ed. by Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, vol. 2098, Lecture Notes in Computer Science, Springer, 2004, pp. 87–124, DOI: 10.1007/b98282.

[Cas+05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime, "Efficient On-the-fly Algorithms for the Analysis of Timed Games", *in*: *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, ed. by Martín Abadi and Luca de Alfaro, vol. 3653, Lecture Notes in Computer Science, Springer, Aug. 2005, pp. 66–80, DOI: 10.1007/11539452_9.

[Cas15] Sofia Cassel, "Learning Component Behavior from Tests: Theory and Algorithms for Automata with Data", PhD thesis, Uppsala University, Sweden, 2015, URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-265369.

[CCF15] Ben Caldwell, Rachel Cardell-Oliver, and Tim French, "Learning Time Delay Mealy Machines From Programmable Logic Controllers", *in*: *IEEE Transactions on Automation Science and Engineering* 13 (Dec. 2015), pp. 1–10, DOI: 10.1109/TASE.2015.2496242.

[CES09] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis, "Model checking: algorithmic verification and debugging", *in*: *Communications of the ACM* 52.*11* (Nov. 2009), pp. 74–84, DOI: 10.1145/1592761.1592781.

[CG98] Rachel Cardell-Oliver and Tim Glover, "A Practical and Complete Algorithm for Testing Real-Time Systems", *in*: *Proceedings of the 5th Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, vol. 1486, Lecture Notes in Computer Science, Springer, Sept. 1998, pp. 251–261, DOI: 10.1007/BFb0055352.

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, *Model checking*, MIT Press, 2000, ISBN: 978-0-262-03270-4.

[CKL98] Richard Castanet, Ousmane Koné, and Patrice Laurençot, "On-the-fly Test Generation for Real Time Protocols", *in*: *Proceedings of the International Conference On Computer Communications and Networks (ICCCN'98)*, IEEE Comp. Soc. Press, Oct. 1998, pp. 378–387, DOI: 10.1109/ICCCN.1998.998798.

[Cla+18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, *Handbook of Model Checking*, Springer, Apr. 2018, DOI: 10.1007/978-3-319-10575-8.

[Cle+20] Emily Clement, Thierry Jéron, Nicolas Markey, and David Mentré, "Computing Maximally-Permissive Strategies in Acyclic Timed Automata", *in*: *Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings*, ed. by Nathalie Bertrand and Nils Jansen, vol. 12288, Lecture Notes in Computer Science, Springer, 2020, pp. 111–126.

[CO94] Rafael C. Carrasco and Jose Oncina, "Learning stochastic regular grammars by means of a state merging method", *in*: *Grammatical Inference and Applications*, ed. by Rafael C. Carrasco and Jose Oncina, Springer Berlin Heidelberg, 1994, pp. 139–152, ISBN: 978-3-540-48985-6.

[CT04] Alexander Clark and Franck Thollard, "PAC-learnability of Probabilistic Deterministic Finite State Automata", *in*: *Journal of Machine Learning Research* 5 (2004), pp. 473–497, URL: http://www.ai.mit.edu/projects/jmlr/papers/volume5/clark04a/clark04a.pdf.

[CW98] Jonathan E. Cook and Alexander L. Wolf, "Discovering Models of Software Processes from Event-Based Data", *in*: *ACM Trans. Softw. Eng. Methodol.* 7.*3* (1998), pp. 215–249, DOI: 10.1145/287000.287001, URL: http://doi.acm.org/10.1145/287000.287001.

[Dal+10] Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Sebastian Hack, and Andreas Zeller, "Generating test cases for specification mining", *in*: *Proceedings of the Nineteenth International Symposium on Software Testing and Analysis, ISSTA 2010, Trento, Italy, July 12-16, 2010*, 2010, pp. 85–96, DOI: 10.1145/1831708.1831719, URL: http://doi.acm.org/10.1145/1831708.1831719.

[Dal+12] Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Gordon Fraser, Sebastian Hack, and Andreas Zeller, "Automatically Generating Test Cases for Specification Mining", *in*: *IEEE Trans. Software Eng.* 38.*2* (2012), pp. 243–257, DOI: 10.1109/TSE.2011.105, URL: https://doi.org/10.1109/TSE.2011.105.

[Dav+08a] Alexandre David, Kim G. Larsen, Shuhao Li, and Brian Nielsen, "A Game-Theoretic Approach to Real-Time System Testing", *in*: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'08)*, Mar. 2008, pp. 486–491, DOI: 10.1109/DATE.2008.4484728.

[Dav+08b] Alexandre David, Kim G. Larsen, Shuhao Li, and Brian Nielsen, "Cooperative Testing of Timed Systems", *in*: *Proceedings of the 4th Workshop on Model Based Testing (MBT'08)*, vol. 220, Electronic Notes in Theoretical Computer Science, 2008, pp. 79–92, DOI: https://doi.org/10.1016/j.entcs.2008.11.007.

[Dav+10] Alexandre David, Kim Guldstrand Larsen, Shuhao Li, Marius Mikučionis, and Brian Nielsen, "Testing Real-Time Systems under Uncertainty", *in*: *Revised Papers of the 13th International Conference on Formal Methods for Components and Objects (FMCO'10)*, vol. 6957, Lecture Notes in Computer Science, Springer, Dec. 2010, pp. 352–371, DOI: 10.1007/978-3-642-25271-6_19.

[Daw+96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The Tool KRONOS", *in*: *In Proc. of Hybrid Systems III*, vol. 1066, Lecture Notes in Computer Science, Springer Verlag, 1996, pp. 208–219.

[De +04] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin, "Robustness and Implementability of Timed Automata", *in*: *Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, ed. by Yassine Lakhnech and Sergio Yovine, vol. 3253, Lecture Notes in Computer Science, Springer, Sept. 2004, pp. 118–133.

[Dil90] David L. Dill, "Timing assumptions and verification of finite-state concurrent systems", *in*: *Automatic Verification Methods for Finite State Systems*, ed. by Joseph Sifakis, Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 197–212.

[DT98] Conrado Daws and Stavros Tripakis, "Model checking of real-time reachability properties using abstractions", *in*: *Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 313–329, ISBN: 978-3-540-69753-4.

[EDK02] Abdeslam En-Nouaary, Radhida Dssouli, and Ferhat Khendek, "Timed WP-Method: Testing Real-Time Systems", *in*: *IEEE Transactions on Software Engineering* 28.*11* (Nov. 2002), pp. 1023–1038, DOI: 10.1109/TSE.2002.1049402.

[Far+08] Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang, "Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages", *in*: *Tools and Algorithms for the Construction and Analysis of Systems*, ed. by C. R. Ramakrishnan and Jakob Rehof, Springer Berlin Heidelberg, 2008.

[Fil11] Jean-Christophe Filliâtre, "Deductive software verification", *in*: *International Journal on Software Tools for Technology Transfer* 13.*5* (Oct. 2011), pp. 397–403, DOI: 10.1007/s10009-011-0211-0.

[Fin06] Olivier Finkel, "Undecidable Problems About Timed Automata", *in*: *Proceedings of the 4th International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, ed. by Eugene Asarin and Patricia Bouyer, vol. 4202, Lecture Notes in Computer Science, Springer, Sept. 2006, pp. 187–199, DOI: 10.1007/11867340_14.

[GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan, "Robust Timed Automata", *in*: *Proceedings of the 1997 International Workshop on Hybrid and Real-Time Systems (HART'97)*, ed. by Oded Maler, vol. 1201, Lecture Notes in Computer Science, Springer, Mar. 1997, pp. 331–345.

[GJ08] Olga Grinchtein and Bengt Jonsson, *Inference of Event-Recording Automata using Timed Decision Trees*, tech. rep., Uppsala Universitet, 2008, URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.473&rep=rep1&type=pdf.

[GJL10] Olga Grinchtein, Bengt Jonsson, and Martin Leucker, "Learning of event-recording automata", *in*: *Theoretical Computer Science* 411.*47* (2010), pp. 4029–4054, ISSN: 0304-3975, DOI: https://doi.org/10.1016/j.tcs.2010.07.008, URL: https://www.sciencedirect.com/science/article/pii/S0304397510003944.

[GJP06] Olga Grinchtein, Bengt Jonsson, and Paul Pettersson, "Inference of Event-Recording Automata Using Timed Decision Trees", *in*: *International Conference on Concurrency Theory (CONCUR'06)*, Lecture Notes in Computer Science, Springer, 2006.

[GL09] Sahika Genc and Stéphane Lafortune, "Predictability of event occurrences in partially-observed discrete-event systems", *in*: *Automatica* 45.*2* (Feb. 2009), pp. 301–311, DOI: 10.1016/j.automatica.2008.06.022.

[Gol67] E. Mark Gold, "Language Identification in the Limit", *in*: *Information and Control* 10.*5* (1967), pp. 447–474, DOI: 10.1016/S0019-9958(67)91165-5, URL: https://doi.org/10.1016/S0019-9958(67)91165-5.

[Gol78] E. Mark Gold, "Complexity of Automaton Identification from Given Data", *in*: *Information and Control* 37.*3* (1978), pp. 302–320, DOI: 10.1016/S0019-9958(78)90562-4, URL: https://doi.org/10.1016/S0019-9958(78)90562-4.

[Gre+20] Alejandro Grez, Filip Mazowiecki, Michał Pilipczuk, Gabriele Puppis, and Cristian Riveros, *The monitoring problem for timed automata*, 2020, arXiv: 2002.07049 [cs.FL].

[Gri08] Olga Grinchtein, “Learning of Timed Systems”, PhD thesis, Uppsala University, Sweden, 2008, URL: http://nbn-resolving.de/urn:nbn:se:uu:diva-8763.

[Hen+94] T.A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine, “Symbolic model checking for real-time systems”, *in*: vol. 111(2), 1994, pp. 193–244.

[Hen+95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya, “What’s Decidable about Hybrid Automata?”, *in*: *Journal of Computer and System Sciences*, ACM Press, 1995, pp. 373–382.

[Hen96] Thomas A. Henzinger, “The Theory of Hybrid Automata”, *in*: IEEE Computer Society Press, 1996, pp. 278–292.

[Hes+08] Anders Hessel, Kim Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou, “Testing Real-Time Systems Using UPPAAL”, *in*: *Formal Methods and Testing: An outcome of the FORTEST network*, ed. by Robert M. Hierons, Jonathan P. Bowen, and Mark Harman, vol. 4949, Lecture Notes in Computer Science, Springer, 2008, pp. 77–117, URL: http://www.cs.aau.dk/~marius/tron/FMT2008.pdf.

[HJM18] Léo Henry, Thierry Jéron, and Nicolas Markey, “Control strategies for off-line testing of timed systems”, *in*: *25th International Symposium on Model-Cherking Software (SPIN’18)*, ed. by María-del-Mar Gallardo and Pedro Merino, vol. 10869, Lecture Notes in Computer Science, Springer, June 2018, pp. 171–189, DOI: 10.1007/978-3-319-94111-0_10.

[HJM20] Léo Henry, Thierry Jéron, and Nicolas Markey, “Active learning of timed automata with unobservable resets”, *in*: *18th International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS’20)*, ed. by Nathalie Bertrand and Nils Jansen, vol. 12288, Lecture Notes in Computer Science, Springer, Sept. 2020, pp. 144–160, DOI: 10.1007/978-3-030-57628-8_9.

[Hoa69] Charles Antony Richard Hoare, “An axiomatic basis for computer programming”, *in*: *Communications of the ACM* 12.*10* (Oct. 1969), pp. 576–580, DOI: 10.1145/363235.363259.

[How60] Ronald A Howard, “Dynamic programming and markov processes.”, *in*: (1960).

[HS06] Thomas A. Henzinger and Joseph Sifakis, "The Embedded Systems Design Challenge", *in*: *Proceedings of the 14th International Symposium on Formal Methods (FM'06)*, ed. by Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, vol. 4085, Lecture Notes in Computer Science, Springer, Aug. 2006, pp. 1–15, DOI: 10.1007/11813040_1.

[Ipa12] Florentin Ipate, "Learning finite cover automata from queries", *in*: *Journal of Computer and System Sciences* 78.*1* (2012), JCSS Knowledge Representation and Reasoning, pp. 221–244, ISSN: 0022-0000, DOI: https://doi.org/10.1016/j.jcss.2011.04.002, URL: https://www.sciencedirect.com/science/article/pii/S002200001100047X.

[Jér+08] Thierry Jéron, Hervé Marchand, Sahika Genc, and Stéphane Lafortune, "Predictability of Sequence Patterns in Discrete Event Systems", *in*: *IFAC Proceedings Volumes* 41.*2* (2008), 17th IFAC World Congress, pp. 537–543, ISSN: 1474-6670, DOI: https://doi.org/10.3182/20080706-5-KR-1001.00091, URL: https://www.sciencedirect.com/science/article/pii/S147466701639005X.

[Kay+03] Dilsun Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager, "Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems.", *in*: *Real-Time Systems Symposium*, Jan. 2003, pp. 166–177, DOI: 10.1109/REAL.2003.1253264.

[KT05] Moez Krichen and Stavros Tripakis, "An Expressive and Implementable Formal Framework for Testing Real-Time Systems", *in*: *Testing of Communicating Systems*, vol. 3502, Springer Berlin Heidelberg, May 2005, pp. 209–225, ISBN: 978-3-540-26054-7, DOI: 10.1007/11430230_15.

[KT09] Moez Krichen and Stavros Tripakis, "Conformance testing for real-time systems", *in*: *Formal Methods in System Design* 34.*3* (June 2009), pp. 238–304, DOI: 10.1007/s10703-009-0065-1.

[KV94] Michael J. Kearns and Umesh Vazirani, *An Introduction to Computational Learning Theory*, The MIT Press, Aug. 1994, ISBN: 0262111934, URL: http://www.worldcat.org/isbn/0262111934.

[Lim+09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez, "Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches", *in*: *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, ed. by Stefan Kowalewski and Anna Philippou, vol. 5505, Lecture Notes in Computer Science, York, United Kingdom: Springer, Mar. 2009, pp. 54–57.

[Lin+11] Shang-Wei Lin, Étienne André, Jin Song Dong, Jun Sun, and Yang Liu, "An Efficient Algorithm for Learning Event-Recording Automata", *in*: *Automated Technology for Verification and Analysis*, ed. by Tevfik Bultan and Pao-Ann Hsiung, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 463–472, ISBN: 978-3-642-24372-1.

[Lin+14] Shang-Wei Lin, Étienne André, Yang Liu, Jun Sun, and Jin Song Dong, "Learning Assumptions for CompositionalVerification of Timed Systems", *in*: *IEEE Transactions on Software Engineering* 40.*2* (2014), pp. 137–153, DOI: 10.1109/TSE.2013.57.

[LMN04] Kim Guldstrand Larsen, Marius Mikučionis, and Brian Nielsen, "Online Testing of Real-time Systems Using Uppaal", *in*: *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing (FATES'04)*, ed. by Jens Grabowski and Brian Nielsen, vol. 3395, Lecture Notes in Computer Science, Springer, Sept. 2004, pp. 79–94, DOI: 10.1007/978-3-540-31848-4_6.

[LMP08] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè, "Automatic generation of software behavioral models", *in*: *2008 ACM/IEEE 30th International Conference on Software Engineering* (2008), pp. 501–510.

[LS09] Martin Leucker and Christian Schallart, "A brief account of runtime verification", *in*: *Journal of Logic and Algebraic Programming* 78.*5* (May 2009), pp. 293–303, DOI: 10.1016/j.jlap.2008.08.004.

[Mai+11] Alexander Maier, Oliver Niggemann, Roman Just, Michael Jäger, and Asmir Vodencarevic, "Anomaly Detection in Production Plants using Timed Automata - Automated Learning of Models from Observations", *in*: *ICINCO 2011 - Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics, Volume 1, Noordwijkerhout, The Netherlands, 28 - 31 July, 2011*, 2011, pp. 363–369.

[Mai14] Alexander Maier, “Online passive learning of timed automata for cyber-physical production systems”, *in*: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)* (2014), pp. 60–66.

[Mer74] Philip Meir Merlin, “A Study of the Recoverability of Computing Systems”, AAI7511026, PhD thesis, 1974.

[Mil68] Bruce L Miller, “Finite state continuous time Markov decision processes with a finite planning horizon”, *in*: *SIAM Journal on Control* 6.*2* (1968), pp. 266–280.

[MNE15] Alexander Maier, Oliver Niggemann, and Jens Eickmeyer, “On the Learning of Timing Behavior for Anomaly Detection in Cyber-Physical Production Systems”, *in*: *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015) co-located with 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (Safeprocess 2015), Paris, France, August 31 - September 3, 2015.* 2015, pp. 217–224, URL: http://ceur-ws.org/Vol-1507/dx15paper28.pdf.

[MP95] Oded Maler and Amir Pnueli, “On the Learnability of Infinitary Regular Sets”, *in*: *Information and Computation* 118.*2* (1995), pp. 316–326, ISSN: 0890-5401, DOI: https://doi.org/10.1006/inco.1995.1070, URL: https://www.sciencedirect.com/science/article/pii/S089054018571070X.

[Ner58] Anil Nerode, “Linear automaton transformations”, *in*: *Proceedings of the American Mathematical Society* 9.*4* (1958), pp. 541–544.

[Nic+93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, “An Approach to the Description and Analysis of Hybrid Systems”, *in*: *Hybrid Systems I*, Springer, 1993, pp. 149–178.

[Nie03] Oliver Niese, “An integrated approach to testing complex systems”, PhD thesis, Fachbereich Informatik, Universitat Dortmund, Dec. 2003, DOI: 10.17877/DE290R-14871.

[NL15] Oliver Niggemann and Volker Lohweg, “On the Diagnosis of Cyber-physical Production Systems: State-of-the-art and Research Agenda”, *in*: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI’15, 2015, URL: http://dl.acm.org/citation.cfm?id=2888116.2888294.

[NS03] Brian Nielsen and Arne Skou, "Automated test generation from timed automata", *in*: *International Journal on Software Tools for Technology Transfer* 5.*1* (Nov. 2003), pp. 59–77, DOI: 10.1007/s10009-002-0094-1.

[PMM17] Fabrizio Pastore, Daniela Micucci, and Leonardo Mariani, "Timed k-Tail: Automatic Inference of Timed Automata", *in*: *CoRR* abs/1705.08399 (2017), URL: http://arxiv.org/abs/1705.08399.

[Pur00] Anuj Puri, "Dynamical Properties of Timed Automata", *in*: *Discrete Event Dynamic Systems* 10.*1-2* (Jan. 2000), pp. 87–113, DOI: 10.1023/A:1008387132377.

[Ram74] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets", PhD thesis, 1974.

[Ram98] Solofo Ramangalahy, *Strategies for comformance testing*, Research Report 98-010, Max-Planck Institut für Informatik, May 1998.

[Rou20] Victor Roussanaly, "Efficient Verification of real-time systems", Ph.D. thesis, IRISA, Univ. Rennes 1, France, 2020.

[RS93] Ronald L. Rivest and Robert E. Schapire, "Inference of Finite Automata Using Homing Sequences", *in*: *Information and Computation* 103.*2* (1993), pp. 299–347, ISSN: 0890-5401, DOI: https://doi.org/10.1006/inco.1993.1021, URL: https://www.sciencedirect.com/science/article/pii/S0890540183710217.

[Sam+96] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis, "Failure diagnosis using discrete-event models", *in*: *IEEE Transactions on Computers* 35.*1* (Jan. 1996), pp. 105–124, DOI: 10.1109/87.486338.

[San13] Ocan Sankur, "Robustness in Timed Automata: Analysis, Synthesis, Implementation", Ph.D. thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2013, URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/sankur-phd13.pdf.

[SB18] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction, second edition*, Cambridge, MA, USA: A Bradford Book, 2018, ISBN: 0262039249.

[Sch13] Jana Schmidt, “Machine learning of timed automata”, PhD thesis, Technical University Munich, 2013, URL: http://nbn-resolving.de/urn:nbn:de:bvb:91-diss-20131216-1145664-0-4.

[SG09] Muzammil Shahbaz and Roland Groz, “Inferring Mealy Machines”, *in*: *FM 2009: Formal Methods*, ed. by Ana Cavalcanti and Dennis R. Dams, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 207–222.

[SNF17] Lukas Schmidt, Apurva Narayan, and Sebastian Fischmeister, “TREM: a tool for mining timed regular specifications from system traces”, *in*: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, 2017, pp. 901–906, DOI: 10.1109/ASE.2017.8115702, URL: https://doi.org/10.1109/ASE.2017.8115702.

[Sun+09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang, “PAT: Towards Flexible Verification under Fairness”, *in*: *Proceedings of the 21th International Conference on Computer Aided Verification (CAV'09)*, vol. 5643, Lecture Notes in Computer Science, Springer, 2009, pp. 709–714.

[SVD01] Jan Springintveld, Frits Vaandrager, and Pedro R. D'Argenio, “Testing timed automata”, *in*: *Theoretical Computer Science* 254.*1-2* (Mar. 2001), pp. 225–257, DOI: 10.1016/S0304-3975(99)00134-6.

[Tap+19] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber, “Time to Learn - Learning Timed Automata from Tests”, *in*: *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings*, ed. by Étienne André and Mariëlle Stoelinga, vol. 11750, Lecture Notes in Computer Science, Springer, 2019, pp. 216–235, DOI: 10.1007/978-3-030-29662-9_13, URL: https://doi.org/10.1007/978-3-030-29662-9%5C_13.

[TDH00] Franck Thollard, Pierre Dupont, and Colin de la Higuera, “Probabilistic DFA Inference using Kullback-Leibler Divergence and Minimality”, *in*: *17th International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, 2000, pp. 975–982.

[Tre96] Jan Tretmans, “Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation”, *in*: *Computer Networks and ISDN Systems* 29.*1* (1996), pp. 49–79, DOI: 10.1016/S0169-7552(96)00017-7.

[Tri02] Stavros Tripakis, “Fault Diagnosis for Timed Automata”, *in*: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, ed. by Werner Damm and Ernst -Rüdiger Olderog, Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 205–221, ISBN: 978-3-540-45739-8.

[Tri06] Stavros Tripakis, “Folk theorems on the determinization and minimization of timed automata”, *in*: *Information Processing Letters* 99.*6* (2006), pp. 222–226, ISSN: 0020-0190, DOI: https://doi.org/10.1016/j.ipl.2006.04.015.

[Val84] Leslie G. Valiant, “A Theory of the Learnable”, *in*: *Commun. ACM* 27.*11* (1984), pp. 1134–1142, DOI: 10.1145/1968.1972, URL: http://doi.acm.org/10.1145/1968.1972.

[VBE21] Frits W. Vaandrager, Roderick Bloem, and Masoud Ebrahimi, “Learning Mealy Machines with One Timer”, *in*: *Language and Automata Theory and Applications - 15th International Conference, LATA 2021, Milan, Italy, March 1-5, 2021, Proceedings*, ed. by Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, vol. 12638, Lecture Notes in Computer Science, Springer, 2021, pp. 157–170, DOI: 10.1007/978-3-030-68195-1_13, URL: https://doi.org/10.1007/978-3-030-68195-1%5C_13.

[VWW08] Sicco E Verwer, Mathijs M de Weerdt, and Cees Witteveen, “Efficiently learning simple timed automata”, *in*: *Induction of Process Models, Workshop at ECML PKDD* (2008), ed. by W. Bridewell, T. Calders, A. K. de Medeiros, S. Kramer, M. Pechenizkiy, and L. Todorovski, pp. 61–68.

[VWW12] Sicco E Verwer, Mathijs M de Weerdt, and Cees Witteveen, “Efficiently identifying deterministic real-time automata from labeled data”, *in*: *Machine Learning* 86 (Mar. 2012), pp. 295–333, DOI: 10.1007/s10994-011-5265-4.

[WLN17] Stefan Windmann, Dorota Lang, and Oliver Niggemann, “Learning parallel automata of PLCs”, *in*: *22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017, Limassol, Cyprus, September 12-15, 2017*, 2017, pp. 1–7, DOI: 10.1109/ETFA.2017.8247693, URL: https://doi.org/10.1109/ETFA.2017.8247693.

[Xia+05] Gang Xiao, Finnegan Southey, Robert C. Holte, and Dana F. Wilkinson, "Software Testing by Active Learning for Commercial Games", *in*: *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, 2005, pp. 898–903, URL: http://www.aaai.org/Library/AAAI/2005/aaai05-142.php.

[Yan04] Mihalis Yannakakis, "Testing, Optimization, and Games", *in*: *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, Lecture Notes in Computer Science, Springer, 2004, pp. 28–45, DOI: 10.1007/978-3-540-27836-8_6.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Histoire d'un aller et retour : méthodes formelles et apprentissage de modèles pour les systèmes en temps réel

Mot clés : Automates temporisés, Théorie des jeux, Apprentissage actif, Méthodes formelles, Estimation d'état

Résumé : Cette thèse traite des méthodes formelles utilisant les automates temporisés, de leurs actions sur la réalité, et des informations que l'on peut apprendre grâce à ces observations. Elle propose différentes contributions dans trois domaines distincts : la théorie des jeux et la génération de tests, vue comme un moyen de contrôler un système à l'aide de méthodes formelles ; l'estimation d'état, qui déduit les configurations possibles d'un système à partir d'observations au moyen d'une construction formelle ; l'apprentissage actif de modèles, qui propose de construire un modèle d'un système en lui demandant des observations, en les orientant selon les besoins de la tâche.

Aussi diverses qu'elles puissent paraître, ces contributions sont liées par le formalisme sous-jacent, les abstractions utilisées et les préoccupations qui caractérisent les interactions entre les modèles formels et la réalité. De plus, elles bénéficient les unes des autres dans la pratique, formant un cercle vertueux : la capacité d'apprendre de la réalité permet de meilleurs modèles, qui à leur tour permettent un contrôle plus fin des systèmes, ce qui favorise les processus d'apprentissage.

Nous relions ces différentes contributions entre elles sur la base des motifs ci-dessus, et nous plaidons pour un rapprochement entre les méthodes et les communautés d'apprentissage de modèles et de méthodes formelles.

Title: There and back again : formal methods and model learning for real-time systems

Keywords: Timed automata, Game Theory, Active learning, Formal methods, State estimation

Abstract: This thesis deals with formal methods based on timed automata, their actions upon reality, and the informations that can be learned from it. It proposes different contributions in three separate domains: game theory and formal test generation, seen as a way to control a system using formal methods; state estimation, that deduce the possible configurations of a system from observations by the mean of a formal construction; active model learning, that propose to construct a formal model of a system by requesting observations out of it, directing them as needed for the task.

As diverse as they may seem, these contributions are linked by the underlying formalism, abstractions and preoccupations that characterizes interactions between formal models and reality. Furthermore, they benefit from one another in practice, forming a virtuous loop: the capacity to learn from reality allows for better models, that in turn permit a finer control of the real systems, which helps the learning processes.

We link these different contributions together based on the grounds above, and advocate for a greater rapprochement between learning and formal methods and communities.