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
Emily CLEMENT

Robustness of timed automata : computing the maximally-permissive strategies

Thèse présentée et soutenue à Rennes, le 11/03/2022
Unité de recherche : INRIA, Equipe SUMO & Mitsubishi Electric R&D Centre EUROPE

Rapporteurs avant soutenance :

Catalin DIMA	Professeur, Université Paris-Est Créteil, France
Franck CASSEZ	Chercheur principal et Professeur Associé, ConsensSys Software Inc. & Macquarie University, Australie

Composition du Jury :

Président :	Pierre-Alain REYNIER	Professeur à l'Université d'Aix-Marseille, France
Examinateurs :	Catalin DIMA	Professeur, Université Paris-Est Créteil, France
	Franck CASSEZ	Chercheur principal et Professeur Associé, ConsensSys Software Inc. & Macquarie University, Australie
	Béatrice BÉRARD	Professeure à l'Université P & M Curie, France
	Thomas CHATAIN	Maître de conférence, ENS Paris-Saclay, France
Dir. de thèse :	Thierry JÉRON,	Directeur de recherche, INRIA Rennes Bretagne Atlantique, France
Co-dir. de thèse :	Nicolas MARKEY,	Directeur de recherche, CNRS, IRISA Rennes, France
Encadrant de thèse :	David MENTRÉ,	Research Manager, Mitsubishi Electric R&D Centre Europe, Rennes, France

Remerciements

En préambule des deux cents et quelques pages qui viennent, qui visent en partie à faire apprécier aux courageux lecteurs qui s'apprêtent à les lire les diverses facettes des polyèdres, j'aimerais en prendre quelques-unes pour remercier celles et ceux qui ont été présents durant cette thèse et qui l'ont rendue plus sereine.

Un grand MERCI tout d'abord à Thierry Jéron, David Mentré et Nicolas Markey pour leur encadrement bienveillant et humain ainsi que leurs précieux conseils durant ces trois et quelques années. Merci d'avoir relu aussi consciencieusement ce manuscrit et de m'avoir prodigué tant de conseils pour l'améliorer.

Merci à Pierre-Alain Reynier d'avoir accepté de présider mon jury, à Catalin Dima et Franck Cassez d'avoir accepté de rapporter cette thèse, et enfin, merci à Béatrice Bérard et Thomas Chatain d'avoir fait partie de mon jury. Merci également à Étienne André de s'être joint à Béatrice afin de me conseiller durant cette thèse au sein du comité de suivi doctoral. Leurs conseils m'ont été très précieux.

Merci également aux enseignants, Thomas, Emmanuelle, Pierre-Alain et David P., qui m'ont fait confiance et proposé d'assurer des enseignements pour eux. Certains enseignants vous marquent plus que d'autres et vous donnent envie d'enseigner à votre tour. À ce titre, je remercie Stéphane Génouël pour son incroyable investissement.

Merci aux deux équipes qui m'ont accueillie conjointement pendant cette thèse, l'une à l'Inria Rennes, l'autre au sein de Mitsubishi Electric R&D Centre Europe (MERCE). Je tiens d'abord à remercier l'équipe administrative, Sophie, Magali et Marie, côté MERCE, et Laurence, côté Inria, pour leur aide extrêmement précieuse lors des démarches administratives, pour le temps qu'elles m'ont fait gagner en les simplifiant. Merci aux permanents de l'équipe SUMO de l'Inria : Éric F., Blaise, Thierry, Ocan, Nicolas M., Nathalie, Loïc, Hervé et Éric B., ainsi qu'aux non-permanents de l'équipe : Hugo B., Arthur, Abdul, Léo, Anirban, Antoine, Suman, Adrian, Nicolas W., Aline, Victor, Grégory et Bastien pour les pauses-café, les séminaires au vert et la bonne humeur générale de l'équipe. Merci en particulier à Loïc de m'avoir fait découvrir le comité de centre de l'Inria Rennes, dont je garde un très bon souvenir, notamment lors de nos actions pour les doctorants avec Anne-Marie Lacroix, Peggy Cellier, César Viho et Cécile Bidan. Merci aux chercheurs de

MERCE avec qui j'ai pu partager mes vendredis et des discussions très intéressantes, et en particulier à Benoît B., David M., Denis, François et Florian.

Évidemment, je tiens à remercier des membres d'autres équipes de l'IRISA qui ont accompagné ma route et rendu cette thèse encore plus joyeuse et intéressante. Merci à Joan et Mathias pour leurs fous rires contagieux lors des pauses-café. Merci à Khalil pour toutes les discussions passionnantes que nous avons pu avoir sur l'optimisation linéaire et les polyèdres. Merci à Sophie et François de trouver le temps pour nous faire découvrir des nouveaux outils pour mieux apprendre l'informatique. Merci également à Martin pour m'avoir changé les idées pendant la thèse quand j'en avais fort besoin avec des activités d'informatique débranchée. Merci pour son écoute, et pour ses conseils francs et simples, qui permettent souvent d'y voir plus clair. Merci aux membres de PACAP, notamment Lily, Hugo R., Caroline et Camille, pour leur bonne humeur lors des pauses repas que nous avons pu partager. Merci à Benoît G. de trouver le temps de venir discuter sécurité informatique autour d'un café.

Merci à mes amis rencontrés tout au long de mes études, avec qui j'ai pu partager des moments simples, mais qui rendent la vie bien plus douce. Merci à Axel, Vlad, Aude, Hugo M. et Julia de m'avoir fait découvrir autant de jeux de société. Merci à Rémi et Solène pour leur enthousiasme à l'idée de passer des soirées à expérimenter de nouvelles recettes de cuisine. Merci à Riwan de m'avoir fait découvrir Cachan et la « med » et aux Cachanais plus généralement de m'avoir accueillie aussi chaleureusement lors de mes très nombreux allers-retours sur leur campus. Merci à Clément pour son soutien, son humour décapant et sa détermination à me faire perdre à Seasons (à 78% au dernier compteur)[1] depuis des années. Merci aux amis de la danse, Filam, les trois Guillaume, Marie-Morgane et Logan, de me sortir régulièrement les idées des sciences, le temps d'un week-end ou d'une soirée.

Merci, au risque de répéter des prénoms, à ceux qui ont relu des morceaux de cette thèse à la recherche d'améliorations et/ou m'ont aidé à préparer ce pot de thèse pour que je puisse dormir la veille de ma soutenance : merci à Hugo B., Mathias, Léo, Solène, Clément, Nicolas B. et Nicolas W.[2].

Enfin, merci à mes parents pour leur soutien pendant mes études, à Barbara et Mathilde d'être toujours là pour leur petite sœur et à mon beau-frère, Adrien, pour son humour et sa bonne humeur. Merci à Christine de m'avoir accueillie aussi chaleureuse-

[1] Ironique, pour une doctorante en théorie des jeux ...

[2] Spéciale dédicace, cher cobureau, pour tes mochis bleus

ment dans sa famille. Finalement, un grand merci à Nicolas B. pour être toujours là pour partager une tasse de thé, sa passion pour l'informatique mais aussi, et c'est là le plus important, ma vie.

Table of Contents

Résumé en français

Motivations

Les systèmes temps-réel sont utilisés dans de nombreux domaines et comportent beaucoup de composants. Il peut s'agir par exemple de moteurs de voitures, d'imprimantes, de stimulateurs cardiaques ou de chaînes de montage. Il devient alors nécessaire de vérifier formellement ces systèmes en amont pour plusieurs raisons. Premièrement, il peut s'agir de systèmes critiques (comme les avions, les moteurs de voitures, les fusées ou les stimulateurs cardiaques) : des erreurs peuvent alors coûter beaucoup de temps, mais aussi être fatales. Deuxièmement, ils peuvent être fabriqués en grand nombre, comme les puces électroniques. Vérifier formellement ces modèles nécessite de considérer un modèle formel, qui prend en compte le temps et sur lequel on peut énoncer formellement ce que l'on veut vérifier.

Les automates temporisés sont un modèle mathématique très utile et efficace. Introduits en 1994 par Alur et Dill dans [AD94], ils fournissent une représentation abstraite des aspects temporels et peuvent être généralisés à des modèles plus complexes pour exprimer encore plus de types de systèmes temps-réel. Le principe est de considérer un automate et d'ajouter un nombre fini d'horloges qui contraignent le passage des différentes transitions.

Prenons l'exemple d'un système d'éclairage automatique pour illustrer la modélisation d'un système temps-réel en un automate temporisé. Ce système d'éclairage s'éteint automatiquement après un certain temps afin d'économiser de l'électricité. Cet exemple est modélisé par un automate temporisé, représenté en Figure 1, avec les deux états possibles (appelés localités) de la lampe : allumé (« On ») et éteint (« Off »), et les actions possibles. On considère que la seule interaction avec la lampe est d'appuyer sur le bouton (action « appuyer »), ce qui allume la lampe. Nous pouvons passer de « Off » à « On » en effectuant l'action « appuyer ». Pour s'éteindre automatiquement, la lampe possède un compteur, une horloge, qu'on appellera x, qui est remise à zéro à chaque fois que l'on appuie sur le bouton (« appuyer »). La lampe s'éteint alors automatiquement au bout d'une minute, c'est-à-dire lorsque $x \geq 1$. Pour l'empêcher de s'éteindre, il faut appuyer sur le bouton (« appuyer ») avant $x = 1$, cette action remettant les horloges à zéro.

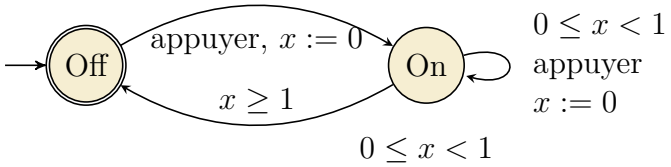


Figure 1: Automate temporisé représentant un éclairage automatique.

Remarquons que les systèmes temps-réel peuvent nécéssiter l'usage de plusieurs horloges. Considèrons un problème d'ordonnancement simple où trois tâches, a, b et c, doivent être effectuées, telles que:

- a et b doivent être effectuées avant c (qu'importe l'ordre).
- L'attente entre les tâches a et b doit être comprise entre 2 et 3 secondes.
- c doit être effectuée au plus tard 5 secondes avant la fin de la tâche a, et au moins 3 secondes après la tâche b.

Ce système se représente à l'aide de l'automate temporisé de la Figure 2.

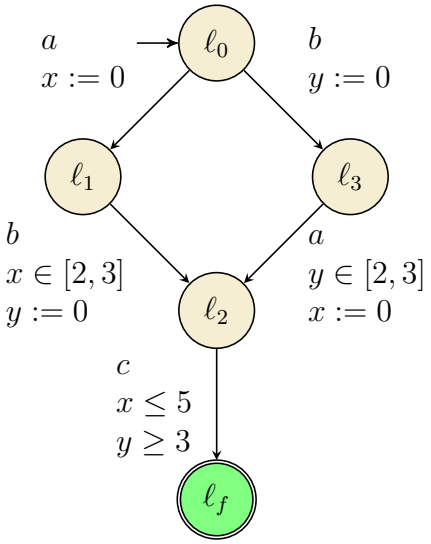


Figure 2: Automate temporisé représentant un ordonnancement de trois tâches.

L'horloge x (*resp.* y) représente le temps écoulé depuis que la tâche a (*resp.* b) a eu lieu.

De nombreuses propriétés peuvent être vérifiées sur des automates temporisés. Dans notre exemple d'éclairage automatique, la propriété que nous souhaitons vérifier est de

savoir si nous pouvons atteindre l'état « Off » à partir de n'importe quelle situation. La réponse est : si on laisse passer le temps, sans appuyer infiniment souvent sur le bouton « appuyer », la seule possibilité est d'atteindre l'état « Off ». Dans notre exemple d'ordonnancement, la propriété que nous souhaitons vérifier est de savoir si nous pouvons atteindre la localisation ℓ_f, à partir de la configuration initiale $(\ell_0, (0, 0))$. Trouver une bonne exécution indiquerait si les tâches a, b et c peuvent être exécutées. On peut aussi se demander si, pour respecter les contraintes de temps, il faut donner un ordre entre les tâches a et b : peut-on trouver une exécution où la tâche a est exécutée avant b, mais aussi une où b est exécutée avant a ?

D'autres exemples d'objets du quotidien peuvent être représentés par des automates temporisés afin d'être vérifiés formellement. Un exemple possible est un ascenseur, pour lequel nous pouvons vouloir vérifier qu'il descendra finalement à notre étage si nous l'appelons. Ces spécifications des localités sont appelées *propriétés d'atteignabilité*. Vérifier l'atteignabilité d'une localité (ou d'un ensemble de localités), étant donné une localité initiale et une valeur initiale des horloges, dans l'automate temporisé, consiste à trouver une exécution depuis la localité initiale et la valeur initiale des horloges, jusqu'à la localité souhaitée. Dans cette thèse, nous étudierons l'atteignabilité.

Cette introduction est organisée comme suit : dans un premier temps, nous décrivons comment les systèmes temps-réel, puis les propriétés temporelles, sont habituellement modélisés. Dans un deuxième temps, nous présentons le principe de la vérification formelle. Dans un troisième temps, nous décrivons comment nous modélisons les imperfections des modèles formels. En quatrième et dernier lieu, nous décrivons l'objectif de cette thèse et ses contributions.

Systèmes temporisés

Afin d'être formellement vérifiés, les systèmes temps-réel sont formellement modélisés. Les principaux modèles utilisés dans la vérification formelle des systèmes temps-réel sont le modèle des automates temporisés et certaines de ses variantes. Nous présentons dans cette section quatre de ces modèles : *les automates temporisés*, *les automates hybrides temporisés*, *les automates temporisés pondérés* et *les jeux temporisés*.

Automates temporisés et automates hybrides temporisés

Un automate temporisé, introduit par Alur et Dill en 1994 dans [AD94], représente un système où le passage d'une transition est contraint par la satisfaction de contraintes temporelles. Selon la sémantique utilisée, les contraintes peuvent prendre plusieurs formes. Chaque horloge peut être bornée indépendamment, comme dans la Figure 1, ou la différence entre deux horloges peut être bornée (comme $0 \leq x - y \leq 1$). Plus généralement, les contraintes sur les horloges peuvent être des inégalités linéaires.

Chaque horloge peut être remise à zéro après avoir franchi une transition. La transition est représentée par une flèche entre deux localités, et la remise à zéro de l'horloge x est représentée par le symbole « $x := 0$ ». Les horloges évoluent à la même vitesse et, pour franchir une transition, on doit proposer un délai et une action. Les horloges sont alors incrémentées de ce délai.

Les automates hybrides sont des systèmes à états infinis. Leur différence avec les automates temporisés est que les valeurs des horloges sont décrites par des équations différentielles ordinaires. Ils peuvent être très utiles pour les applications liées aux systèmes cyber-physiques.

Automates temporisés pondérés

Les automates temporisés pondérés ont été introduits par [ATP01] et [BFH$^+$01]. Ces modèles peuvent être utilisés pour modéliser des embouteillages ou le choix entre différents moyens de transport (avion *vs* train *vs* voiture par exemple). Dans ces modèles, rester à un endroit ou prendre une transition peut avoir un coût. Nous additionnons le coût total pendant une exécution, un run, et l'appelons la *fonction de coût*.

Considérons l'exemple du choix entre l'avion et le train pour se rendre dans une ville. Dans la Figure 3, nous représentons deux manières possibles de rejoindre Londres depuis Rennes. Nous modélisons ces possibilités avec leurs coûts associés (en euros) par un automate temporisé pondéré. Dans cet automate temporisé, les contraintes représentent les horaires (en heures) imposés de chaque mode de transport.

Dans cet exemple, nous pouvons rejoindre Londres en avion, sans escale, ce qui coûte 120 €, ou en prenant deux trains, avec une correspondance à Lille, ce qui coûte un total de 89 €. Les horloges représentent l'heure du jour et sont remises à zéro à minuit. Dans cet exemple, voyager en train est moins cher que de voyager en avion, mais les horaires des trains sont plus stricts dans notre exemple : le train de Lille à Londres doit être pris

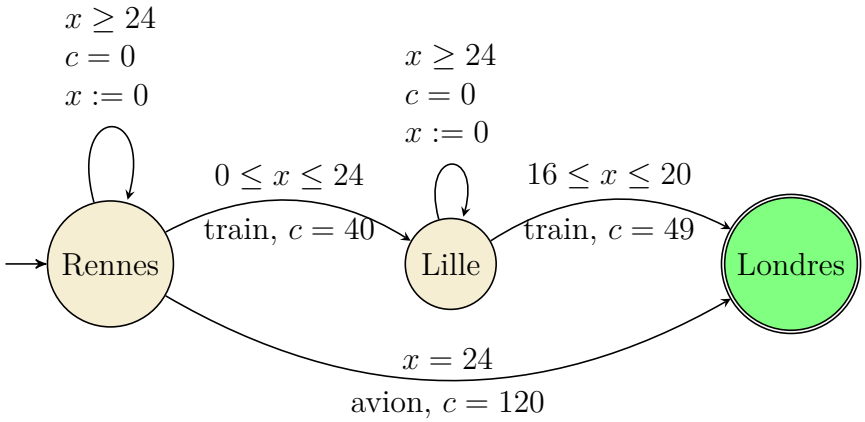


Figure 3: Un automate temporisé pondéré pour représenter deux moyens de transport possibles pour rejoindre Londres depuis Rennes.

entre 16 et 20 h. L'un des objectifs de ces modèles est d'atteindre le but en minimisant (ou en maximisant) la fonction de coût.

Jeux temporisés

Les jeux temporisés généralisent les automates temporisés en ajoutant plusieurs joueurs dans l'automate temporisé, pouvant prendre des décisions différentes (comme le contrôle de certaines localités uniquement), avec des buts pouvant être différents : minimiser ou maximiser une fonction de coût, atteindre ou ne pas atteindre un but, etc. Lorsque leurs objectifs sont opposés, nous pouvons séparer ces joueurs en **joueurs** et **opposants**.

Il peut y avoir plusieurs sémantiques de jeux temporisés. Par exemple, les jeux où les joueurs peuvent contrôler différentes localités sont une forme de *jeux par tours*. Cette sémantique est détaillée dans la sous-section 5.2.3.

Conclusion

D'autres modèles sont possibles. Par exemple, lorsqu'il s'agit de systèmes temporisés distribués, on peut utiliser des automates temporisés distribués. Lorsque certaines décisions (délais, actions...) ne sont pas prises par un joueur mais par une distribution aléatoire sur toutes les possibilités, on peut utiliser des automates temporisés probabilistes.

Propriétés temporelles

Ces modèles nous permettent de formaliser les propriétés que nous voulons vérifier. Il existe plusieurs types de propriétés temporisées, telles que la *vivacité* et la *sûreté*. Elles furent introduites par Alpern et Schneider dans [AS85] et par Alpern, Demers et Schneider dans [ADS86].

Les propriétés de **vivacité** garantissent qu'un certain événement finira par se produire, quelle que soit l'execution. Elles s'opposent aux propriétés de **sûreté** qui garantissent qu'un certain événement ne se produira jamais, quelle que soient les exécutions. Les propriétés de sûreté peuvent être utilisées pour de nombreuses applications, par exemple pour éviter les obstacles pour les trains. Pour qu'une propriété de sûreté soit violée, il faut trouver une exécution finie où cet événement se produit. Alpern et Schneider on démontré dans [AS87] le résultat suivant : nous pouvons exprimer toute propriété temporelle linéaire comme l'intersection de propriétés de vivacité et de sûreté.

La propriété que nous étudierons dans cette thèse est *l'atteignabilité*, qui correspond à la négation d'une propriété de sûreté. Les propriétés d'**atteignabilité** garantissent qu'une certaine localité est atteignable compte tenu d'une (ou plusieurs) configuration(s) initiale(s), *i.e.* qu'il existe une éxecution, depuis une configuration initiale, atteignant cette localité. Considérons par exemple l'exemple de la Figure 3. La propriété d'atteignabilité que nous pourrions vérifier est la suivante : « Peut-on atteindre Londres depuis Rennes si l'horloge initiale est $x = 8$?[3] ».

Vérification de systèmes temporisés

Expliquons à présent comment nous vérifions les systèmes temps-réel. Le but de la vérification formelle est de considérer toutes les configurations possibles des systèmes et de prouver que la propriété temporelle que nous considérons est toujours vérifiée, ou à défaut de construire un contre-exemple où cette propriété n'est pas vérifiée. C'est le principe des algorithmes de model-checking, introduits par Clarke, Emerson, Queille, Sifakis au début des années 80. Leur travail fut récompensé par un prix Turing en 2007. Pour une introduction au model-checking, on pourra se référer au livre [BK08].

Le principe d'un algorithme de model-checking est illustré dans la Figure 4 : étant donné un modèle abstrait et une propriété temporelle, l'algorithme de model-checking

[3] *i.e.*, si l'on quitte Rennes au plus tôt à 8 heures du matin.

apporte soit la garantie que la propriété temporelle (par exemple, « La lumière finira-t-elle par s'éteindre ? ») est vérifiée, soit un exemple d'exécution où elle n'est pas vérifiée. Le principe du model-checking est d'abord de modéliser le système et la propriété que l'on veut satisfaire, puis d'exécuter l'algorithme de model-checking. Si la propriété est vérifiée, le processus s'arrête, sinon le contre-exemple doit être analysé afin de modifier le modèle (ou la propriété), jusqu'à ce que la propriété soit vérifiée. La force du model-checking est

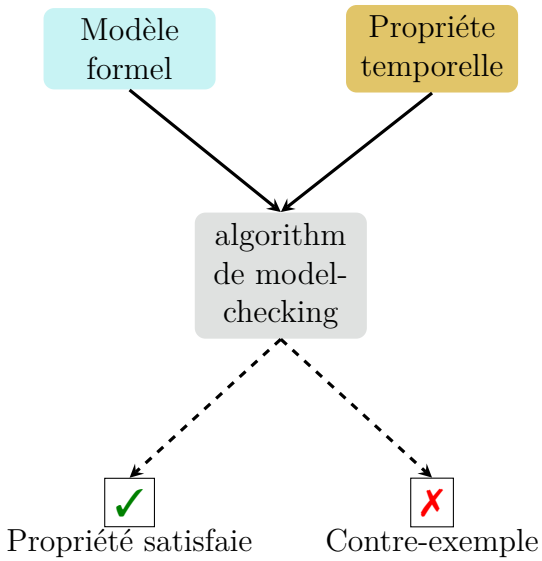


Figure 4: Principe de la vérification formelle.

de vérifier formellement une propriété, pour un modèle général, individuellement (cela signifie que nous pouvons vérifier chaque propriété indépendamment et voir laquelle n'est pas vérifiée) et de fournir des contre-exemples pour expliquer pourquoi la propriété n'a pas été satisfaite. Sa faiblesse est son **coût élevé** pour les systèmes complexes, tels que les automates temporisés, en raison de l'explosion de l'espace d'états.

Plusieurs outils de model-checking, pour différents types de systèmes temporisés, ont été implémentés, parmi lesquels on peut citer : Kronos[4] ([BDM+98], [Yov97]), Uppaal[5] ([BDL+06]), TChecker[6], IMITATOR[7] ([And09], [And10], [AFK+12], [And21]), Roméo[8] ([GLM+05], [LRS+09]) at PAT[9] ([SLD+09]).

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

Robustesse : prendre en compte les possibles imprécisions

La robustesse d'un objet fait référence, en général, à sa capacité à résister à de (petites) perturbations. Lors de la modélisation de systèmes temporisés, on suppose qu'on peut atteindre une précision parfaite sur tous les aspects de la modélisation. Dans ces modèles, nous supposons que :

- ▷ Les horloges croissent exactement à la même vitesse, elles sont synchronisées.
- ▷ Les gardes des transitions seront exactement celles qui seront implémentées.
- ▷ Les délais proposés pour passer une transition seront parfaitement appliqués, avec une précision infinie, sans latence.

Cependant, dans le monde réel, les horloges peuvent se désynchroniser et ainsi ne pas évoluer à la même vitesse. Le délai proposé peut aussi être appliqué avec une légère avance ou latence. Par exemple, dans le système d'éclairage automatique présenté dans la Figure 1, si nous appuyons sur le bouton à $x = 0.8$, le délai peut en fait être appliqué à 0.802, en raison d'une légère latence. Par conséquent, la vérification de la capacité à résister aux perturbations est une question importante. La robustesse peut également être appliquée à l'*ordonnancement*. L'ordonnancement est l'action d'ordonner des tâches pour les réaliser (voir l'exemple illustré dans la Figure 2). Ses objectifs sont nombreux, l'un d'eux est de minimiser le temps d'attente entre deux tâches. Il est facile de concevoir que les latences sont inévitables. Ainsi, dans le monde réel, les actions ne se succèdent pas sans imprécision quant au moment où elles ont été réalisées. La fiabilité du résultat final vis-à-vis de diverses perturbations de l'environnement et d'un matériel perfectible est donc une question cruciale si l'on veut exécuter cet ordonnancement dans le monde réel. Dans l'exemple de la Figure 2, si la latence est supérieure à 1 seconde, cela est problématique car le délai entre les deux tâches a et b doit être compris entre 2 et 3 secondes.

Il existe différentes façons de vérifier la robustesse. Nous en détaillons quatre dans cette section. Considérons un système temps-réel $\mathcal{S}$, modélisé par un modèle $\mathcal{M}$. On veut vérifier une propriété temporelle φ. Ce modèle $\mathcal{M}$ a été construit pour représenter $\mathcal{S}$ à l'aide de **spécifications**.

Analyse du langage Cette première approche, qui ne transforme pas le modèle, consiste à analyser dans quelle mesure la perturbation peut changer ce que le modèle

exprime. Considérons le modèle $\mathcal{M}$ et le langage L que $\mathcal{M}$ exprime. Considérons le modèle $\mathcal{M}$ légèrement perturbé $\widetilde{\mathcal{M}}$. Le but d'une *analyse du langage* sera d'exprimer le langage $\widetilde{L}$ exprimé par le modèle $\widetilde{\mathcal{M}}$.

Analyse de la robustesse Cette autre approche vérifie si le modèle peut résister aux perturbations, sans interférer. Supposons que le modèle $\mathcal{M}$ satisfasse φ , le but de l'analyse de la robustesse est de vérifier si la propriété φ est toujours satisfaite pour un modèle légèrement perturbé $\widetilde{\mathcal{M}}$.

Implémentation robuste Cette approche implémente le modèle robuste, en continuant à respecter les spécification du système temps-réel. Le modèle $\mathcal{M}$ peut ne pas satisfaire la propriété φ après certaines perturbations. L'*implémentation robuste* a pour objectif de transformer le modèle $\mathcal{M}$ en un modèle robuste aux perturbations , $\widetilde{\mathcal{M}}$, qui satisfait toujours les spécifications du système.

Synthèse robuste Enfin, la *synthèse robuste* consiste à construire un *contrôleur* qui forcera le modèle à avoir un bon comportement par rapport à la propriété désirée φ, malgré les perturbations. Cette approche utilise généralement la théorie des jeux pour modéliser et construire la stratégie du contrôleur et de l'environnement.

Les perturbations sur les automates temporisés peuvent se produire à différents niveaux. Pour énumérer les principales d'entre elles, il convient d'énumérer les composants qui peuvent être perturbés sur un automate temporisé : gardes, horloges, délais, transitions, etc. La plupart des travaux ont isolé chaque type de perturbation afin de définir différents modèles de robustesse. Après tout, un automate peut satisfaire de manière robuste une propriété temporelle malgré des imperfections d'horloges, mais ne pas être robuste face à une perturbation sur les gardes. Le premier modèle de robustesse a été proposé par [GHJ97], où ils cherchent à vérifier la robustesse topologique d'un run. Les runs sont représentés comme des trajectoires et le but est de vérifier si les voisins de ces trajectoires sont acceptés par l'automate temporisé. Dans cette thèse, nous nous concentrons sur les perturbations sur les délais pour les automates temporisés et les jeux.

Plan de la thèse

Dans cette thèse, nous présentons des résultats sur la robustesse des automates temporisés en considérant les perturbations sur les délais et la permissivité maximale autorisée. La

permissivité a été étudiée par [BFM15] en 2015, pour des automates temporisés à une horloge. Ils fournissent un algorithme en temps polynomial pour calculer la permissivité maximale autorisée. Les résultats de cette thèse visent à étendre les résultats de [BFM15] pour plusieurs horloges en utilisant une fonction légèrement modifiée pour calculer la perturbation d'un automate temporisé. En effet, dans [BFM15], l'imprécision d'une exécution est la somme de l'inverse de chaque imprécision appliquée tout au long d'une exécution. Dans notre cas, nous représentons l'imprécision d'une exécution comme la plus petite imprécision commise lors d'une exécution, afin de s'assurer que cette imprécision sera autorisée **à chaque transition**. Nous appelons cette imprécision autorisée la *permissivité*. Notre but est de calculer cette permissivité maximale pour les automates multi-horloges pour toute configuration. Cette thèse est organisée comme suit. Les Chapitres 1 à 3 définissent les notions de base et l'état de l'art :

Chapitre 1: Nous présentons les définitions mathématiques que nous utiliserons dans cette thèse. Les deux principaux concepts mathématiques que nous utilisons sont les polyèdres et les fonctions affines par morceaux. Ces fonctions affines par morceaux sont considérées en dimensions supérieures à 1. Nous définissons leurs morceaux comme des polyèdres et définissons les fonctions affines par morceaux avec la notion de partition de polyèdres.

Chapitre 2: Nous donnons une introduction aux automates temporisés et à la vérification d'atteignabilité. Nous définissons également notre modèle de robustesse en définissant la *sémantique permissive*, la permissivité d'une exécution ainsi que la permissivité maximale permise par un automate, étant donné une configuration fixe.

Chapitre 3: Ce chapitre donne un état de l'art des différents types de robustesse proposés dans la littérature.

Cette thèse comporte quatre contributions principales, détaillées dans les Chapitres 4, 5, 6 et 7.

Chapitre 4: Nous définissons une suite de fonctions afin d'avoir une expression itérative de la fonction de permissivité. Nous appelons cette suite de fonctions la *suite de fonctions permissives sous-optimales*. En effet, l'expression de la fonction de permissivité fournie dans le Chapitre 2 n'est pas adaptée à un calcul effectif, car elle considère tous les intervalles proposés en même temps. Afin d'avoir une approche

plus itérative, nous approchons cette fonction par une suite de fonctions, calculant une stratégie sous-optimale. Dans ce chapitre, nous prouvons que cette suite de fonctions et la permissivité sont liées : nous montrons que la fonction de permissivité est la limite de cette suite de fonctions. Nous fournissons de bonnes propriétés de cette suite de fonctions et étendons certaines d'entre elles à la fonction de permissivité. Le but de ces propriétés est d'aider au calcul de ces fonctions.

Chapitre 5: Nous présentons un algorithme pour calculer la suite de fonctions permissives sous-optimales et la fonction de permissivité, en temps au plus non-élémentaire pour les automates temporisés acycliques et les jeux temporisés acycliques.

Chapitre 6: Nous présentons deux implémentations. Dans la Section 6.1, nous proposons une approche numérique, avec un algorithme qui calcule en avant une valeur approximative de la fonction de permissivité pour une configuration fixée numériquement. Nous fournissons une implémentation de cet algorithme. De plus, afin de fournir une preuve de concept et d'étudier les temps de calcul en pratique de l'algorithme présenté au Chapitre 5, nous présentons son implémentation symbolique dans la Section 6.2 et comparons ses résultats avec ceux de l'implémentation de notre approche numérique.

Chapitre 7: Nous étudions des problèmes approximatifs pour calculer la permissivité plus efficacement. L'objectif de ce chapitre est de calculer une *permissivité par niveaux*, c'est-à-dire une fonction qui, étant donné une liste de seuils, décide si la valeur de la fonction de permissivité d'une configuration se situe entre deux seuils. Nous présentons dans ce chapitre un algorithme en temps doublement exponentiel, pour les automates temporisés linéaires, qui, étant donné un seuil positif, détermine si la permissivité est supérieure à ce seuil. Cet algorithme nous permet de présenter un algorithme en temps doublement exponentiel pour calculer la permissivité par niveaux pour les automates temporisés linéaires.

INTRODUCTION

Motivations

Real-time systems are used in many areas and involve a large number of components. That can be car engines, printers, pacemakers or assembly lines for instance. The need of formally checking these systems before implementing them comes from several reasons. First, they can be critical systems (such as planes, car engines, rockets or pacemakers), for which errors can cost life or a lot of time. Secondly, they can be manufactured in large numbers, such as telecommunications chips. Formally verifying these models requires considering a formal model, which takes time into account and on which one can formally state what one intends to verify.

Timed automata is a very useful and efficient mathematical model to provide an abstract model of real-time systems. It was introduced in 1994 by Alur and Dill in [AD94]. They provide a very expressive representation of timing aspects and can be generalised to more complex models to express even more types of real-time systems. The principle is to consider an automaton and to add a finite number of clocks that constrain the passing of the different transitions.

Let us describe how an automatic light system can be modelled as a timed automaton. This light system automatically switches off after a certain period of time in order to save electricity. We illustrate this simple example in Figure 5 with the two possible states (called locations) of the lamp, on and off, and the possible actions. We consider that the only interaction with the lamp is to press the button ('press' action), which turns on the lamp. We can switch from 'Off' to 'On' by performing the 'press' action. In order to switch off automatically, the lamp has a counter, a clock, denoted by x, which is reset to zero each time the button ('press') is pressed. The lamp will then automatically switch off after one minute[10], *i.e.* when $x \geq 1$. To prevent it from turning off, the button ('press') must be pressed before $x = 1$, as this action resets the clocks.

Let us remark that real-time systems may need more than one clock to represent them. Let us take a scheduling example, where we have to execute three tasks a, b and c, such

[10] where the time unit is the minute.

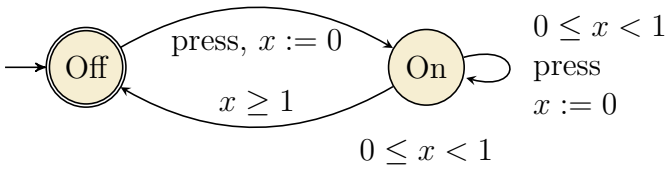


Figure 5: A timed automaton that illustrates an automatic light system.

that:

- a and b need to be executed before c.
- The delay between tasks a and b should be between 2 and 3 seconds.
- Task c must be performed no later than 5 seconds before the end of task a, and at least 3 seconds after task b.

This real-time system is represented by the timed automaton in Figure 2. The clock x (*resp.* y) represents the time elapsed since task a (*resp.* b) took place.

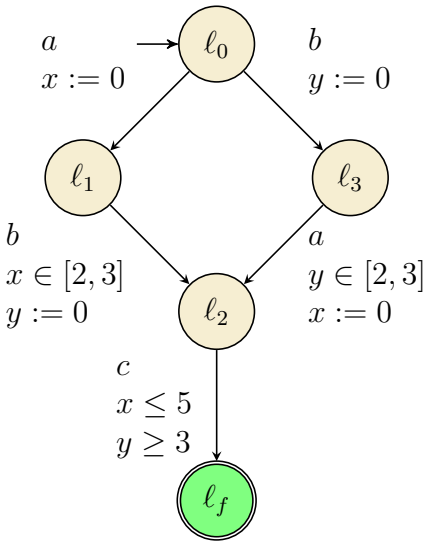


Figure 2: A timed automaton that illustrates a scheduling example.

Many properties can be checked on timed automata. For instance on our lamp example, the property we would like to verify is whether we can reach the 'Off' state from any situation. The answer is: if we let time pass, without pressing the button 'press' infinitely often, the only possibility is to reach the 'Off' state. In our scheduling example, the property we would like to verify is whether we can reach location ℓ_f, from the initial

location ℓ_0. Finding a good execution would indicate whether the tasks a, b and c can be executed. One can also ask whether, in order to respect the time constraints, an order must be fixed between a and b: can one find an execution where a is executed before b, but also one where b is executed before a? Other examples of everyday objects can be represented by timed automata in order to be formally checked. An example is a lift, where we may want to check that it will eventually go down to our floor if we call it. These specification of the locations are called *reachability properties*. Verifying the reachability of a location (or a set of locations), given an initial location and value of clocks in the timed automaton, consists in finding an execution from the initial location and value of clocks, to the desired location. In this thesis, we focus on this particular timed property.

This introduction is organised as follows: first, we describe how real-time systems, and then the properties to verify, are usually modelled. Secondly, we present the principle of formal verification. Thirdly, we describe how we model the imperfections of the formal models. Fourthly and finally, we describe the aim of this thesis and its contributions.

Timed systems

In order to be verified, real-time systems are formally modelled. The main used model in formal verification of real-time systems is timed automata and some variants. We present in this section four of these models: *timed automata*, *weighted timed automata*, *timed games* and *hybrid timed automata*.

Timed and hybrid automata

A timed automaton, introduced by Alur and Dill in 1994 in [AD94], represents a system where to pass a transition is constrained by the satisfaction of timing constraints of the clocks. Depending on the used semantics, the constraints can take several forms. Each clock can be bounded independently, as in Figure 5, or the difference of clocks can be bounded (as $0 \leq x-y \leq 1$). More generally, constraints on clocks can be linear inequalities.

Each clock can be reset after passing a transition. The transition is represented with an edge between two locations, and the reset of the clock x is represented with the symbol '$x := 0$'. The clocks evolve at the same speed and each transition is passed by proposing a delay and an action.

Hybrid automata are an infinite state systems whose difference with timed automata

is that the values of the clocks are described by ordinary differential equations. They can be quite useful for applications related to cyber physical systems.

Weighted timed automata

Weighted timed automata were introduced by [ATP01] and [BFH+01]. These models can be used used for applications such as traffic jam or choosing different types of transport (air-plane *vs* train *vs* car for instance). In these models, staying in a location, or taking a transition may have a cost. We sum the total cost during an execution, a run, and call it the *cost-function*.

Let us consider the example of choosing between air-plane and train to go to a city. In Figure 6 we represent two possible ways to reach London from Rennes. We model these possibilities with their associated costs (in euros) with a weighted timed automaton. In this timed automaton, the constraints represent imposed schedules of each mode of transport. In this example, we can reach London with a direct plane, that costs 120 €, or with two

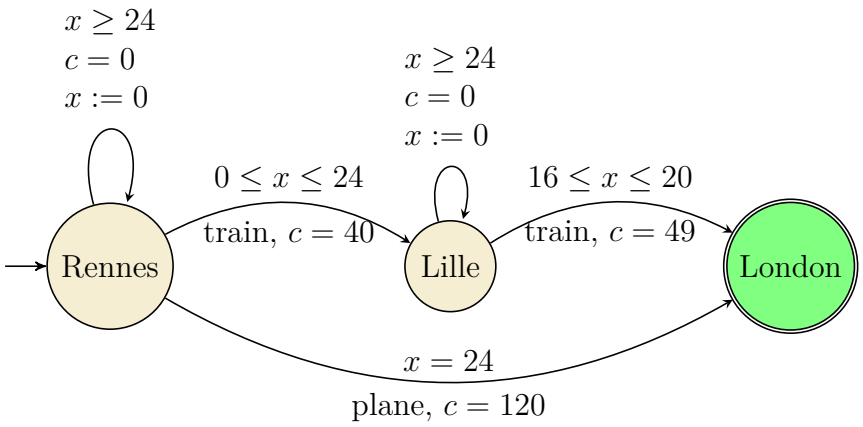


Figure 6: A weighted timed automaton representing the possible ways to reach London from Rennes.

trains, that cost a total of 89 €. The clocks represent the time of the current day (in hours) and are reset at midnight. In this example, the train travel is cheaper than the plane travel, but the schedules of the trains are more constrained in our example: the train from Lille to London should be taken between 4 and 8 pm. One of the aim of these models is to reach the goal while minimising (or maximising) the cost function.

Timed games

Timed games extend timed automata by adding several players that can have different control (proposing delay *or* actions, proposing delay and action on only specific locations) with different purposes: minimising, maximising a cost function, reaching or not reaching a goal etc. When their purposes are opposite, we can separate the players into **players** and **opponents**.

There can be several semantics of timed games. For instance, games where players can control different locations is a form of *turn-based games*. This semantics is detailed in Subsection 5.2.3.

Conclusion

Other models are possible. For instance, when tackling with distributed timed systems, we can use distributed timed automata. When some decisions (delays, action...) are not decided by a player but by a random distribution over all possibilities, we can use probabilistic timed automata.

Timed properties

These models enable us to formalise the properties we want to verify. There are several types of timed properties, such as *liveness* and *safety*, introduced by Alpern and Schneider in [AS85] and by Alpern, Demers and Schneider in [ADS86]. Let us detail them.

Liveness properties ensure that some (good) event will eventually occur. They are opposed to **safety** properties that ensure that some (bad) event will never occur. Safety properties can be used for many application, for example to avoid obstacles for car engines or trains. For a safety property to be violated, a finite execution should be found where this bad event occurs. A useful and strong result was proved by Alpern and Schneider in [AS87]: we can express every linear-time property as the intersection of liveness and safety properties.

The property we will study in this thesis is *reachability*, which corresponds to the negation of a safety property. **Reachability** properties ensure that a target location is reachable given an (or several) initial configurations, *i.e.* that there exists a run from an initial configuration to the target location. Let us consider for instance the example of Figure 6. The reachability property we could verify can be ‘can we reach London from

Rennes if the initial clock is $x = 8$?[11]'.

Principle of formal verification

Now that we have presented how real time systems are modelled, let us explain how to verify them. The aim of formal verification is, considering all possible configurations of our systems, and to either prove that the timed property we consider is verified, or construct a counter-example where this property is not verified. This is the principle of model-checking algorithms, introduced by Clarke, Emerson, Queille, Sifakis in the early 80s. Their work was rewarded by a Turing award in 2007. For an introduction to Model Checking, one can refer to the book [BK08].

The principle of model-checking algorithm is illustrated in Figure 7: given an abstract model and a timed property, the model-checking algorithm gives either the guarantee that the timed property (for instance, 'Will our system eventually finish ?') holds in every configuration, or an example of a execution where it was not verified. The principle a model-checking is first to model the system and the property we want to satisfy, then to run the model-checking algorithm. If the property is verified, the process stops, otherwise the counter-example should be analysed in order to change the model (or the property), until the property is verified.

The strength of model-checking is to formally verify a property, for a general model, individually (meaning that we can check each property independently and see which one is violated) and provide counter-examples to explain why the property was not satisfied. **The weakness of model-checking** is its high cost for complex systems, such as timed automata, because of state-space explosion.

Several model-checking tools, for different types of timed systems, have been designed such as Kronos[12]([BDM+98], [Yov97]), Uppaal[13] ([BDL+06]), TChecker[14], IMITATOR[15] ([And09], [And10], [AFK+12], [And21]), Roméo[16] ([GLM+05], [LRS+09]) and PAT[17] ([SLD+09]).

[11] *i.e.* if we leave Rennes at least at 8 a.m.
[12] https://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/
[13] https://uppaal.org/
[14] https://www.labri.fr/perso/herbrete/tchecker/
[15] https://www.imitator.fr/index.html
[16] http://romeo.rts-software.org/
[17] https://pat.comp.nus.edu.sg/

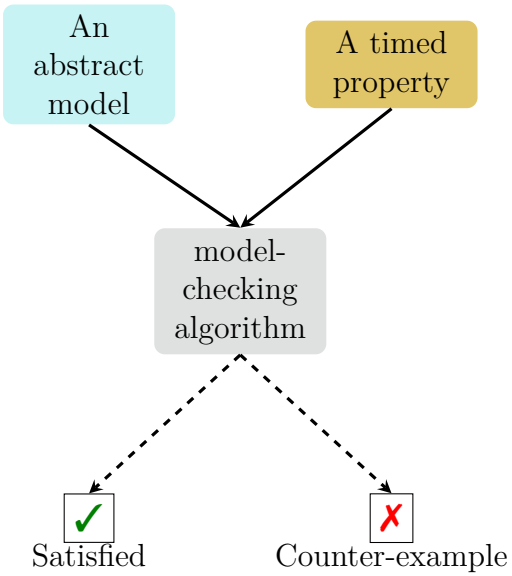


Figure 7: Principle of formal verification.

Robustness: Verifying timed systems despite perturbations

The robustness of an object refers, in general, to its ability to resist to (small) perturbations. When modelling timed systems, perfect precision is supposed in all aspects. In these abstract models, we assume that:

- The clocks grow at exactly the same speed, they are synchronised.
- The guards of the transitions will be exactly those that will be implemented.
- The delays proposed to pass a transition will be exactly applied, with infinite precision.

However, in the real world, clocks can desynchronise, that is, evolve at different speeds. The proposed delay may also be applied with a slight advance or latency. For instance, in the automatic light system presented in Figure 5, if we press the button at $x = 0.8$, the delay actually may be applied at 0.802, because of a slight latency. Therefore, checking the ability to resist to perturbation is an important issue.

Robustness can also be applied on *scheduling*. Scheduling is the action of ordering tasks to perform them (see the example from Figure 2). Its goals are numerous, one of them is to minimise the waiting time between two tasks. It is easy to see that latencies

are unavoidable, and that in the real world, actions will not follow each other without imprecision as to when they were performed. In the example of Figure 2, if the latency is greater than 1 second, it is an issue as the delay between tasks a and b should be between 2 and 3 seconds. The reliability of the final result with respect to various environmental disturbances and perfectible hardware is therefore a crucial issue if one wants to execute this scheduling in the real world.

There are different ways to check robustness. We detail four of them in this section. Let us consider a real-time system $\mathcal{S}$, modelled by an abstract model $\mathcal{M}$, to verify the timed property φ. This model $\mathcal{M}$ was built to represent $\mathcal{S}$ with the help of **specifications**.

Language analysis This first approach, which does not transform the model, consists in analysing how much perturbation can change what the model **expresses**. Consider model $\mathcal{M}$ and the language L that $\mathcal{M}$ expresses. The goal of a *language analysis* will be to express the language $\widetilde{L}$ that the perturbed model $\widetilde{\mathcal{M}}$ expresses.

Robust analysis This other approach checks if the model can resist to perturbations, without interfering. Assuming that model $\mathcal{M}$ satisfies φ, the aim of robust analysis is to check whether property φ is still satisfied if the model is slightly perturbed.

Robust implementation This approach directly implements the robust model. The model may not satisfy the property after certain perturbations. One of the objectives of robustness, called *robust implementation*, is to transform model $\mathcal{M}$ into a robust (to perturbations) model, which always satisfies the system specifications. We implement a robust model of the system, based on the ideal model, so that it is robust to perturbations.

Robust synthesis Finally, *robust synthesis* asks whether one can build a *controller* that will force the model to have a good behaviour with respect to the property φ, despite perturbations. This approach usually uses game theory to model and build the strategy of both the controller and the environment.

Perturbations on timed automata can occur at different levels. To enumerate the main ones, we should list the components that could be perturbed on a timed automaton: guards, clocks, delays, runs. Most works have isolated each type of perturbations to define different robustness models. After all, an automaton may robustly satisfy a temporal property despite clock imperfections, but not for perturbations on guards. The first model

of robustness was proposed by [GHJ97]. Their goal is to verify the topological robustness of a run. Runs are represented as trajectories and the goal is to check if the neighbours of these trajectories are accepted by the timed automaton.

In this thesis, we focus on the perturbations on the delays for timed automata and games.

The goal of this thesis is to construct a *robust synthesis*: we construct a strategy, for any configuration, to satisfy the **reachability** on timed automata, despite the perturbations of delays. On top of that, we want to compute the **maximal perturbation allowed**. In other terms, we want to compute how much permissive the timed automaton is to perturbations on the delays and build a controller that will maximise the permissiveness of a timed automaton.

Content and organisation of this thesis

In this thesis, we present results on the robustness of timed automata considering delay perturbations and the maximal allowed permissiveness. The permissiveness was studied by [BFM15] in 2015, for one-clock timed automata. The authors provide a polynomial time algorithm to compute the maximal allowed permissiveness. The results of this thesis aim at extending the results of [BFM15] for several clocks using a slightly modified function to compute the perturbation of a timed automaton. Indeed, in [BFM15], the imprecision of a run is the sum of inverses of all imprecisions applied throughout a run. Our aim is to represent the smallest imprecision, in order to ensure that this imprecision will be allowed **at each transition**. We call this allowed imprecision the *permissiveness*. Our goal is to compute this maximal permissiveness for multi-clock automata for any configuration.

This thesis is organised as follows. Chapter 1 to 3 define basic notions and state of the art:

Chapter 1: We present the mathematical definitions we will use throughout this thesis. The two principal mathematical concepts we use are polyhedra and piecewise affine functions. These piecewise affine functions are considered in dimensions greater than 1. We define their cells as polyhedra and define piecewise affine functions with the notion of partition of polyhedra.

Chapter 2: We give an introduction to timed automata and reachability verification. We also define our model of robustness by defining the *permissive semantics*, the

permissiveness of a run and the maximal permissiveness allowed by an automaton, given a fixed configuration.

Chapter 3: This chapter reviews the state of the art for the different kinds of robustness proposed in the literature.

There are four main contributions in this thesis, detailed in Chapters 4, 5, 6 and 7.

Chapter 4: We define a sequence of functions to provide an iterative expression of the permissiveness function. We call this sequence the *sequence of suboptimal permissive functions*. Indeed, the expression of the permissiveness function provided in Chapter 2 is not adapted to effective computation as it considers all the intervals proposed at the same time. In order to obtain a more iterative approach, we approximate this function with a sequence of functions. In this chapter, we prove the links between this sequence of functions and the permissiveness: we show that the permissiveness function is the limit of this sequence of functions. We provide good properties of this sequence of functions and extend some of them to the permissiveness function. The aim of these properties is to help computing these functions.

Chapter 5: We give an algorithm to compute the sequence of suboptimal permissive functions and the permissiveness function in non-elementary time for acyclic timed automata and games.

Chapter 6: We present two implementations. In Section 6.1, we provide another numerical approach with a forward algorithm that computes an approximate value of the permissiveness function for a fixed configuration. We provide an implementation of this algorithm. To give a proof-of-concept of our algorithm presented in Chapter 5, and to study its runtime, we present the symbolic implementation in Section 6.2 and compare it with the numeric implementation.

Chapter 7: We study approximate problems to compute permissiveness more efficiently. The aim of this chapter is to compute a *leveled permissiveness*, *i.e.*, a function that, given a list of thresholds, decides whether the value of the permissiveness function of a configuration lies between two thresholds.

In this chapter, we introduce an algorithm for linear timed automata, which, given a positive threshold, computes whether the permissiveness is greater than this threshold. This algorithm allows us to present a double exponential time algorithm for computing permissiveness by levels for linear timed automata.

List of publications

The results of Chapter 5 have been published in 2020 in the following article:

- [**CJM+20**]: Emily Clement, Thierry Jéron, Nicolas Markey and David Mentré. Computing maximally-permissive strategies in acyclic timed automata, *in* Nathalie Bertrand and Nils Jansen, editors, *Formal Modeling and Analysis of Timed Systems - 18th International Conference (FORMATS 2020), Vienna, Austria.* Volume 12288 of *Lecture Notes in Computer Science*, pages 111–126, Springer, September 2020, DOI: 10.1007/978-3-030-57628-8_7, URL: https://doi.org/10.1007/978-3-030-57628-8%5C_7.

The contributions of Chapters 6 and 7 will be submitted later.

NOTATIONS

Mathematical notations

We use standard Landau notations $\mathcal{O}, \Omega, \Theta, o, \omega$.

$\mathbb{R}$	The set of real numbers
$\mathbb{Q}$	The set of rational numbers
$\mathbb{N}$	The set of positive integers
K_+	The set of positive elements of K, for $K = \mathbb{R}$ or $\mathbb{Q}$
$\overline{E}$	The topological closure of the set E
$\mathring{E}$	The topological interior of the set E
$\lvert E \rvert$	The size of the set E
$[x \cdots y]$	The interval of all integers between x and y
$[x, y]$	The interval of all reals between x and y
$]x, y]$ (*resp.* $[x, y]$)	The interval of all reals between x and y, x (*resp.* y) excluded
$]x, y[$	The interval of all reals between x and y, x and y excluded
$\mathcal{M}_{k,n}(\mathbb{R})$	The set of k-by-n real matrices
$\overrightarrow{x}$	vector
x_i	i-th coordinate of a family/vector x
$\lfloor x \rfloor$	floor function of x
$\mathsf{fract}\,(x)$	fractional function of x
$\lVert x \rVert$	The norm of an element
$\lvert I \rvert$	The length of an interval
$\mathbb{1}_E$	The indicator function over the set E
$\bigwedge$	The logical conjunction

Timed automaton notations

$\mathcal{A}$	A timed automaton
ℓ	A location of a timed automaton
ℓ_f	A goal location of a timed automaton
g	A guard
Σ	The set of actions
a	An action
E	The set of transitions
e	A transition
$\mathcal{C}$	The set of clocks
x (or y)	A clock
c	A constraint
v	A valuation
$v \models g$	The valuation v verifies the guard g
$\mathcal{C}_r$	A set of resets
ρ	A run
σ	A strategy
δ	A delay

Chapter 1

Preliminaries

In this chapter, we develop the mathematical notions we need for this thesis. The main mathematical concepts we use are *polyhedra* and *piecewise affine functions*, in dimensions greater than one. *Piecewise affine functions* are a well-known type of functions in one dimension. Our aim in this chapter is to extend this notion to an arbitrary dimension. An example of this kind of functions in dimension 2 is shown in Figure 1.1.

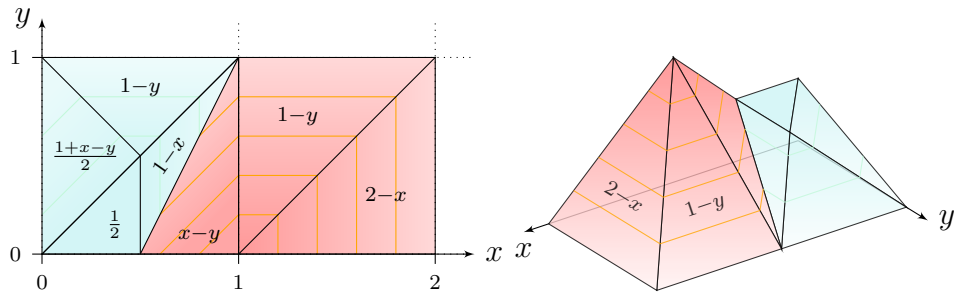


Figure 1.1: An example of a 2-dimension piecewise affine function from $[0, 2] \times [0, 1]$ to $\mathbb{R}$.

We define piecewise affine functions in an arbitrary dimension. We first need to extend the notions of affine function and cells to higher dimensions. First, we extend **affine functions** to a higher dimension in Section 1.1. Next, we need to extend the notion of cells. In dimension 1, they are represented as **intervals**. In dimension 2, they can be, for instance in Figure 1.1, **triangles**. This depends on the number of edges. In higher dimensions, we represent them with **polyhedra**. The partition of these cells is called *partition of polyhedra*. We introduced these two notions in Section 1.2. Finally, we define what a **multidimensional piecewise-affine function** is in Section 1.3.

1.1 Multidimensional affine functions

Let us now formally define affine functions in higher dimension, which we call *multidimensional affine functions*, or *n-dimensional affine functions* when specifying the dimension

used. We define these functions over $\mathbb{R}$ or $\mathbb{Q}$. Therefore we denote $K = \mathbb{R}$ or $\mathbb{Q}$ for the rest of this section. We extend affine functions in higher dimensions and with the possibility to have infinite coefficients. More formally:

Definition 1.1: n-dimensional affine functions

Let n be an integer and E a subset of K^n. An n-dimensional affine function from E to $K \cup \{-\infty, +\infty\}$ is a mapping $f : E \mapsto K \cup \{-\infty, +\infty\}$ such that:

- Either there exists a vector $(a_i)_{0 \leq i \leq n} \in K^n$ such that:

$$\forall x = (x_i)_{1 \leq i \leq n} \in E, f(x) = \sum_{i=1}^{n} a_i \cdot x_i + a_0$$

- Or $f(x) = -\infty$ (*resp.* $+\infty$) for all $x \in K^n$, in that case we can still write $f(x) = \sum_{i=1}^{n} a_i \cdot x_i + a_0$ by setting $a_0 = -\infty$ (*resp.* $+\infty$) and $a_i = 0$ for all $1 \leq i \leq n$

Let us recall the terminology for these functions:

- f is called a multidimensional affine function when n is not specified.
- a_0 is called the *inhomogeneous term* of f.
- $a_1, \cdots, a_n$ are called the *homogeneous terms* of f.
- If $a_0 = 0$, f is called an n-dimensional (or multidimensional) *linear* function. In that case, $f(0) = 0$.

Example 1.1.1 *Let us give a simple example in dimension* 3*:*

$$f : (x, y, z) \mapsto x - y + z + 10 \ , \ g = f - 10.$$

f and g are 3-dimensional affine functions. In particular, g is an 3-dimensional linear function, f is not. The inhomogeneous term of f is $a_0 = 10$, and the homogeneous terms of f are $a_1 = 1, a_2 = -1$ and $a_3 = 1$.

Our goal in the next Sections 1.2 and 1.3 is to define piecewise affine functions over K^n. The Figure 1.1 represents the piecewise affine functions $f : [0, 2] \times [0, 1] \mapsto \mathbb{R}$ such that:

$$f: \ (x,y) \ \mapsto \begin{cases} 1/2 & \text{if } x \in [0,1/2], y \leq x \\ \dfrac{1+x-y}{2} & \text{if } x \in [0,1/2], y \geq x, y \leq 1-x \\ 1-y & \text{if } x \leq 1, y \geq 1-x, y \geq x \\ 1-x & \text{if } y \leq x, x \geq 1/2, y \geq -1+2x \\ x-y & \text{if } y \leq -1+2x, x \leq 1 \\ 1-y & \text{if } y \geq -1+x, x \geq 1, y \leq 1, x \leq 2 \\ 2-x & \text{if } y \leq -1+x, x \leq 2, 1 \leq x \end{cases}$$

In dimensions greater than 1, we represent the pieces of theses functions with polyhedra. To do that, in the next section, we define polyhedra and the partition of polyhedra.

1.2 Polyhedra

General definitions

In this section, we define polyhedra and give some examples. First, let us recall what the convexity property is.

Definition 1.2: Convexity

A set S is convex if and only if, for all x, y in S, the segment $[x, y]$ is included in S, *i.e.*:

$$\forall x, y \in S, \forall t \in [0,1], t \cdot x + (1-t) \cdot y \in S$$

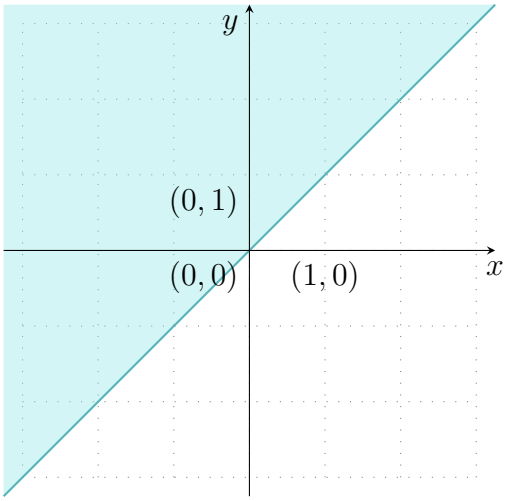
Half-spaces represent sets that cut $\mathbb{R}^n$ in half. They are specified by linear inequalities (as $a_1x_1 + a_2x_2 + \cdots + a_nx_n > b$). In a one-dimension space, half-spaces define rays (ex: $x > 0$). More formally:

Definition 1.3: Half-space

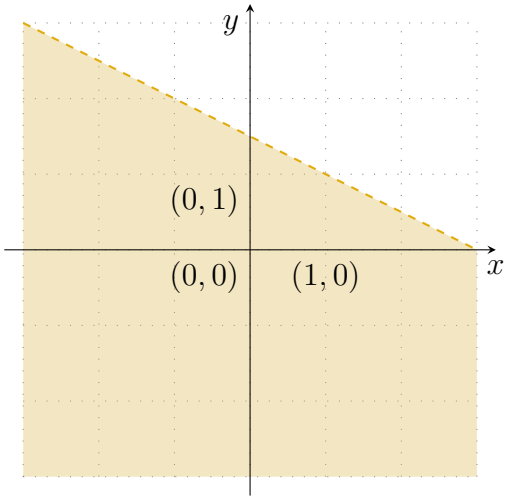
Let $n \geq 0$ be an integer. A **half-space** of $\mathbb{R}^n$ is a set of the form $\{X \in \mathbb{R}^n | f(X) \sim 0\}$ where $\sim \in \{\leq, \geq, <, >\}$ and f is an affine, non-constant, function of $\mathbb{R}^n$.
If $\sim \in \{\leq, \geq\}$ (*resp.* $\{<, >\}$), the half-space is called a closed (*resp.* open) half-space.

We can remark that a half-space is convex. The Figure 1.2 represents examples of half-spaces of $\mathbb{R}$ and $\mathbb{R}^2$.

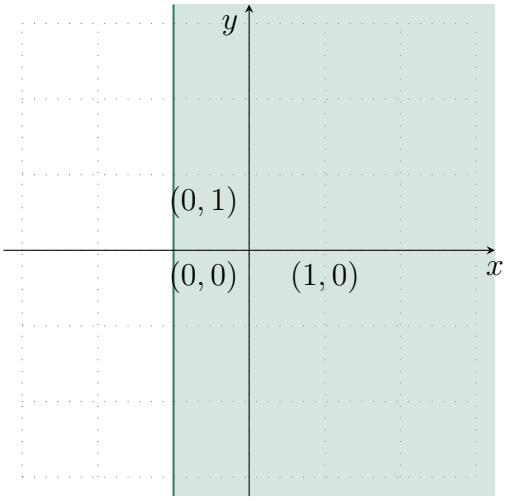
(a) Half-space $\{x \mid x \leq 0\}$ of $\mathbb{R}$.



(b) Half-space $\{(x, y) \mid y - x \geq 0\}$ of $\mathbb{R}^2$.



(c) Half-space $\{(x, y) \mid \dfrac{3 - x}{2} - y > 0\}$ of $\mathbb{R}^2$.



(d) Half-space $\{(x, y) \mid x \geq -1\}$ of $\mathbb{R}^2$.

Figure 1.2: Examples of half-spaces of $\mathbb{R}$ and $\mathbb{R}^2$.

Definition 1.4: Polyhedra and closed polyhedra

A *polyhedron* of $\mathbb{R}^n$ is a finite intersection of half-spaces of $\mathbb{R}^n$. A polyhedron is *closed* if it is a finite intersection of **closed** half-space of $\mathbb{R}^n$.

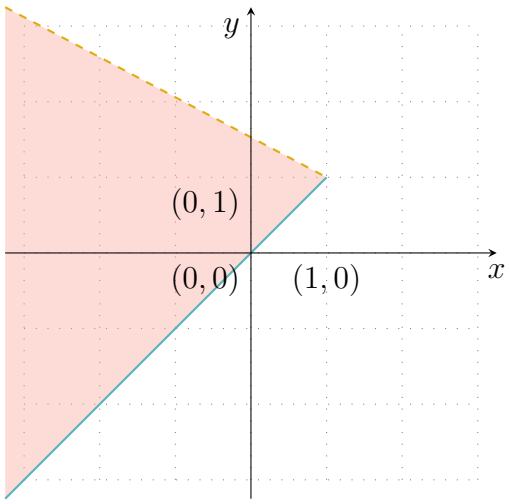
An immediate property, as the intersection of several convex set is a convex set, is the convexity of polyhedron, stated in Proposition 1.5

Proposition 1.5

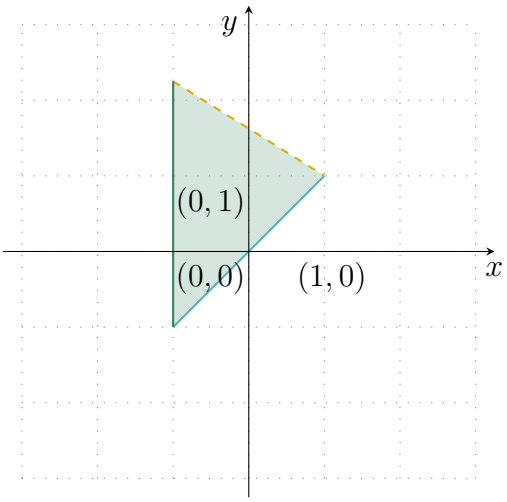
A polyhedron of $\mathbb{R}^n$ is a **convex** set.

Example 1.2.1 *We represent the intersection of the three half-spaces of* $\mathbb{R}^2$ *of Figure 1.2 in Figure 1.3a and the first two half-spaces of* $\mathbb{R}^2$ *of Figure 1.2 in Figure 1.3b.*

In Figure 1.3a, the polyhedron is the set $\{(x, y) \in \mathbb{R}^2 \mid y \geq x, \frac{3-x}{2} > y\}$. *In Figure 1.3b, the polyhedron is the set* $\{(x, y) \in \mathbb{R}^2 \mid y \geq x, \frac{3-x}{2} > y, x \geq -1\}$. *Both are not closed because the half-space of Figure1.2c is open.*



(a) Intersection of half-spaces of Figure 1.2b and 1.2c.



(b) Intersection of half-spaces of Figure 1.2b, 1.2c and 1.2d.

Figure 1.3: Two examples of polyhedra in $\mathbb{R}^2$.

A closed polyhedron can be represented with a so called **H-representation**:

Theorem 1.6: H-representation ([Sch86], p.87)

$P \subseteq \mathbb{R}^n$ is a **closed polyhedron** if and only if there exists some integer k, some matrix $A \in \mathcal{M}_{k,n}(\mathbb{R})$ and some vector $b \in \mathbb{R}^k$ such that P is defined as follows:

$$P = \{x \in \mathbb{R}^n \mid A \cdot x \leq b\}$$

This representation is called a *H-representation*.

Let us finally remark that the intersection of polyhedra preserves the properties of convexity, closeness, openness, while the union does not. More formally:

Proposition 1.7: Intersection of polyhedra

The finite intersection of:

- polyhedra is a polyhedron.
- closed polyhedra is a closed polyhedron.
- open polyhedra is an open polyhedron.

Partition and tiling of polyhedra

The goal of this section is to define what a partition of polyhedra and a tiling of polyhedra are. The intuition is to divide a space, as $\mathbb{R}^n_+$ or $\mathbb{R}^n$, with a collection of disjoint polyhedra. For instance let us take the four following polyhedra:

$$\begin{aligned}
\mathcal{P}_0 &= \left\{(x,y) \in \mathbb{R}^2 | \mid x \geq 1, y \geq 1\right\} \\
\mathcal{P}_1 &= \left\{(x,y) \in \mathbb{R}^2 | \mid x < 1, y \geq 1\right\} \\
\mathcal{P}_2 &= \left\{(x,y) \in \mathbb{R}^2 | \mid x < 1, y < 1\right\} \\
\mathcal{P}_3 &= \left\{(x,y) \in \mathbb{R}^2 | \mid x \geq 1, y < 1\right\}
\end{aligned}$$

These polyhedra are all disjoints two by two and their union forms the space $\mathbb{R}^2$ (see Figure 1.4). Let us define a partition of polyhedra in Definition 1.8

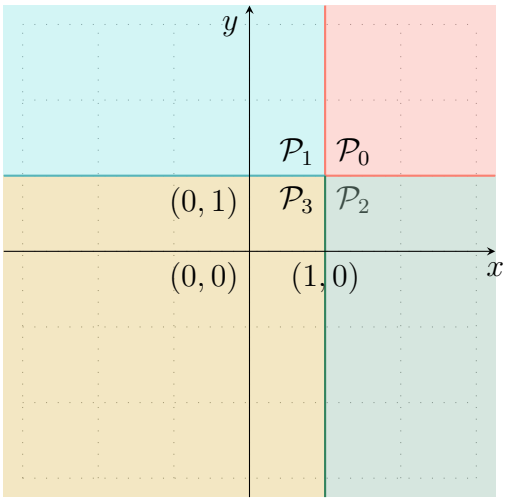


Figure 1.4: Partition of $\mathbb{R}^2$ with the four polyhedra $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$.

Definition 1.8: Partition of polyhedra

Let P be a polyhedron of $\mathbb{R}^n$. A *partition of polyhedra* $\mathcal{T}$ of the polyhedron P is a finite family of $m+1$ polyhedra $(\mathcal{P}_i)_{0\leq i\leq m}$ of $\mathbb{R}^n$ such that:

- For any i in $[0 \cdots m]$, $\mathcal{P}_i \neq \varnothing$.
- For any i, j in $[0 \cdots m]$, $\mathcal{P}_i$ and $\mathcal{P}_j$ are disjoints.
- $\bigcup_{i=0}^{m} \mathcal{P}_i = E$.

The integer $m+1$ is called the **number of cells** of the partition of E. We say that $\mathcal{T}$ is a $m+1$-cells partition of E.
Each $\mathcal{P}_i$ is a *cell* of the partition $(\mathcal{P}_i)_{0\leq i\leq m}$.

A polyhedron can also be represented using a tiling. Tiling of polyhedra is less restrictive because it allows polyhedra to intersect at their frontiers. This definition will be used in the case of a continuous piecewise affine function because the values of the functions at the frontiers of the polyhedra will then be equal. Let us define the tiling formally in Definition 1.9

Definition 1.9: Tiling of polyhedra

Let P be a polyhedron of $\mathbb{R}^n$. A *tiling of polyhedra* of P is a finite family $\mathcal{T}$ of $m+1$ polyhedra $(\mathcal{P}_i)_{0\leq i\leq m}$ of $\mathbb{R}^n$ such that:

- For any i in $[0\cdots m]$, $\mathcal{P}_i \neq \varnothing$.
- For any i, j in $[0\cdots m]$, the interior of $\mathcal{P}_i$ and the interior of $\mathcal{P}_j$ are disjoints.
- $\bigcup_{i=0}^{m} \mathcal{P}_i = E$.

The integer $m+1$ is called the **number of cells** of the tiling of E. We say that $\mathcal{T}$ is a $m+1$-cells tiling of E.
Each $\mathcal{P}_i$ is a *cell* of the tiling $(\mathcal{P}_i)_{0\leq i\leq m}$

To represent $\mathbb{R}^2$ as a tiling of polyhedra, we can for instance consider the following family of polyhedra:

$$\mathcal{P}'_0 = \left\{(x,y) \in \mathbb{R}^2 | \mid x \geq 1, y \geq 1\right\}$$
$$\mathcal{P}'_1 = \left\{(x,y) \in \mathbb{R}^2 | \mid x \leq 1, y \geq 1\right\}$$
$$\mathcal{P}'_2 = \left\{(x,y) \in \mathbb{R}^2 | \mid x \leq 1, y \leq 1\right\}$$
$$\mathcal{P}'_3 = \left\{(x,y) \in \mathbb{R}^2 | \mid x \geq 1, y \leq 1\right\}$$

These polyhedra $\mathcal{P}'_0, \mathcal{P}'_1, \mathcal{P}'_2, \mathcal{P}'_3$ are respectively the closure of $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$.

1.3 Multidimensional piecewise-affine functions

In this subsection, we define the multidimensional piecewise affine functions in Definition 1.10. We also specify that, when the piecewise-affine function is continuous, we can use a tiling of polyhedra instead of a partition to define it.

Definition 1.10: n-dimensional piecewise affine functions

Let P be a polyhedron of $\mathbb{R}^n$. An n-dimensional affine function is a mapping $f : P \to \mathbb{R} \cup \{-\infty, +\infty\}$ for which there exists a partition into polyhedra $\mathcal{T} = (\mathcal{P}_i)_{0 \leq i \leq m}$ of P and a family $(f_i)_{0 \leq i \leq m}$ of n-dimensional affine functions such that for any $X \in P$, there is a **unique** index i in $[0 \cdots m]$ such that $X \in \mathcal{P}_i$ and $f(X) = f_i(X)$. Let us define some useful terminology:

- f is called a *multidimensional piecewise affine functions* when the dimension n of f is not specified.
- $\mathcal{T}$ is called the partition of polyhedra of f and $(f_i)_{1 \leq i \leq m}$ is called the list of affine functions of f.
- Each cell $\mathcal{P}_i$ is called a *cell* of f and is associated with the affine function f_i.
- The entry domain such that f takes finite values, *i.e.* $f^{-1}(\mathbb{R})$, is denoted $\mathcal{D}_{\text{finite}}(f)$ and called the finite domain of an n-dimensional affine function f.

Example 1.3.1 *Let us construct the example of the 2-dimensional affine function represented in Figure 1.5a.*

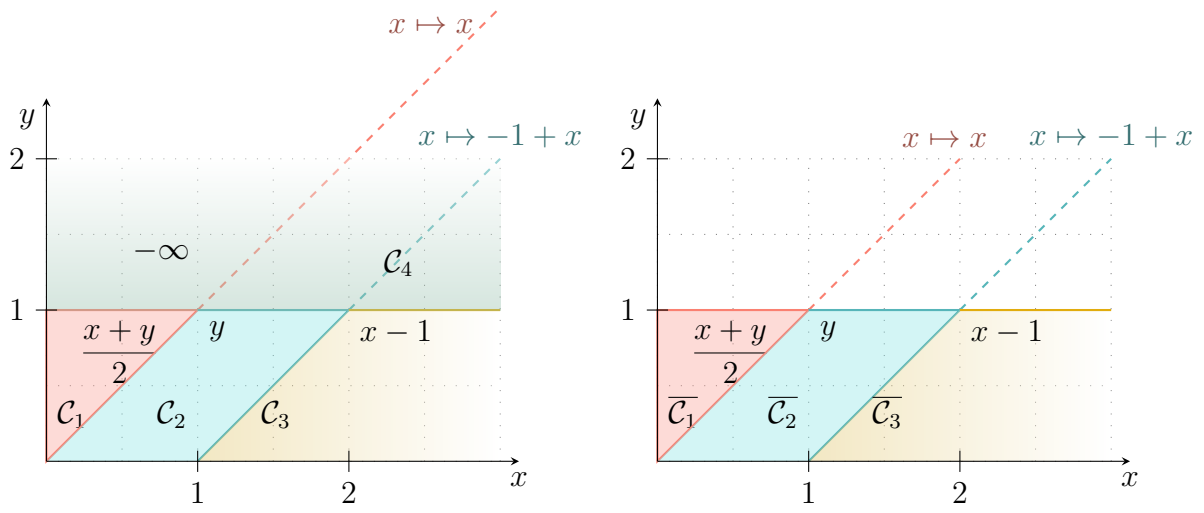
This is a 4-cells partition of $\mathbb{R}^2_+$, that we will denote $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ and $\mathcal{C}_4$, where:

$$\mathcal{C}_1 = \{(x,y) \in \mathbb{R}^2_+ \mid y \leq 1, x - y \leq 0\}, \quad \mathcal{C}_2 = \{(x,y) \in \mathbb{R}^2_+ \mid y \leq 1, 0 < x - y \leq 1\},$$
$$\mathcal{C}_3 = \{(x,y) \in \mathbb{R}^2_+ \mid y \leq 1, x - y > 1\}, \quad \mathcal{C}_4 = \{(x,y) \in \mathbb{R}^2_+ \mid y \geq 1, x \geq 0\}$$

Let us then describe f:

$$f = \begin{cases} f_1 : (x,y) \mapsto (x+y)/2 & \text{over } \mathcal{C}_1 \\ f_2 : (x,y) \mapsto y & \text{over } \mathcal{C}_2 \\ f_3 : (x,y) \mapsto x - 1 & \text{over } \mathcal{C}_3 \\ f_4 : (x,y) \mapsto -\infty & \text{over } \mathcal{C}_4 \end{cases}$$

In the case of **continuous** n-dimensional piecewise affine functions, we do not need to split the polyhedron P into strictly disjoint cells: we can use tiling. The main advantage is to consider only **closed** polyhedra with the H-representation as defined in Theorem



(a) An 2-dimensional affine function f. (b) Its (continuous) restriction over $[0, 1] \times \mathbb{R}_+$.

Figure 1.5: Two 2-dimensional affine functions, represented with partition of polyhedra (Figure 1.5a) or tiling of polyhedra (Figure 1.5b).

1.6. In that case, we can define f as a mapping $P \mapsto \mathbb{R} \cup \{-\infty, +\infty\}$ associated with a tiling of polyhedra $\mathcal{T} := (\mathcal{P}_i)_{0 \leq i \leq m}$ of P and a family $(f_i)_{0 \leq i \leq m}$ of n-dimension affine functions *s.t.* for any $X \in P$, there is an index i in $[1 \cdots m]$ such that $X \in \mathcal{P}_i$ and $f(X) = f_i(X)$. If $X \in \mathring{\mathcal{P}}_i$, the index i is unique. The rest of the definition can be extended trivially from Definition 1.10. Indeed, as the function is continuous, if two cells $\mathcal{P}_i$ and $\mathcal{P}_j$ are not disjoints, the affine function f_i and f_j will be equals on their common frontier. We illustrate the representation with a tiling of polyhedra in Figure 1.5b: we consider the restriction of the function f described in Example 1.3.1. Its restriction is continuous so we can use the tiling of polyhedra $\left\{\overline{\mathcal{C}_1}, \overline{\mathcal{C}_2}, \overline{\mathcal{C}_3}\right\}$. Let us recall that for any set E, $\overline{E}$ represents its topological closure.

As most of our functions will be continuous piecewise affine functions (at least over their finite domain), we will mainly use tilings instead of partitions in this thesis.

Chapter 2

Timed automata and robustness

In this chapter, we present the notions of the *theory of timed automata* that we will need in order to define, quantify and compute robustness of timed automata. In Section 2.1, we present timed automata, describe their syntax and semantics, from the constraints to the runs. Then, we present two fundamental problems in the theory of timed automata: reachability and robustness. The reachability problem asks whether a location of a timed automaton can be visited, knowing the starting configuration. The robustness problem asks whether properties will still be verified despite perturbations. As robustness is a very general problem, we need to specify our perturbation model. In Section 2.2, we present one model where perturbations affect delays. We call this model the *permissiveness*.

2.1 Timed automata

Timed automata are a very convenient way to model the behaviour of real-time systems. Here we give the essential notions and results on timed automata. First, we introduce clock valuations and constraints. Clock valuations represent the values of several clocks and time constraints allow us to check whether the clock values are bounded by some expressions. In a second step, we express the syntax and classical semantics of timed automata. In a third step, we present the reachability problem and how it is solved. A classical construction on timed automata is the representation of valuations with *clock regions*. This abstraction provides a finite-state automaton which is time-abstract bisimilar to the original timed automaton. This allows to prove that the reachability problem is in PSPACE. Finally, we express the limitations of reachability. Indeed, when dealing with reachability properties, the clocks are assumed to be perfectly accurate, which is not the case in the real world. To model the imperfection of the model in the real world, we introduce the *robustness* concept, which studies whether a property holds despite temporal imprecisions.

2.1.1 Clock constraints

Clocks are objects that represent time with real values. Their values progress at the same speed but can be reset independently. We denote the set of all clocks by $\mathcal{C}$. A *clock valuation*, is a function from $\mathcal{C}$ to $\mathbb{R}_+$. It can be represented as a point in $\mathbb{R}_+^{|\mathcal{C}|}$. Given a valuation v and a non-negative real δ, we denote by $v+\delta$ the valuation w such that for any clock variable $x \in \mathcal{C}, w(x) = v(x) + \delta$. Let $I \subseteq \mathbb{R}_+$ be an interval, we write $v + I$ for the set of valuations $\{v + \delta \mid \delta \in I\}$. To denote the reset of clocks, given a valuation v and a subset $\mathcal{C}_r \subseteq \mathcal{C}$, we denote by $v[\mathcal{C}_r \leftarrow 0]$ the valuation w such that $w(x) = 0$ if $x \in \mathcal{C}_r$ and $w(x) = v(x)$ if $x \notin \mathcal{C}_r$.

To verify temporal properties, we first define how a **valuation** satisfies a **timing constraint**. There are several types of clock constraints. In this thesis, we use *classical guards* and *polyhedral guards*. Classical guards are a simple way to constrain clocks: all clocks are bounded with constant values. For example, if we consider the set of clocks $\{x, y\}$, the timing constraint $0 \leq x \leq 5 \wedge 5 \leq y \leq 10$ is a guard. More formally:

Definition 2.1: Classical guards

The set of classical guards over $\mathcal{C}$, denoted $\mathcal{G}(\mathcal{C})$, is defined as $\mathcal{G}(\mathcal{C}) \ni g := x \sim n \mid g \wedge g$ where x ranges over $\mathcal{C}$, n ranges over $\mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. A *constraint* $x \sim n$ is *closed* if $\sim \in \{=, \leq, \geq\}$. A classical guard is *closed* if each of its constraints is closed.
The fact that a valuation v satisfies a classical guard g, denoted $v \models g$ is defined inductively as:

1. $(v \models x \sim n)$ if and only if $(v(x) \sim n)$.
2. $(v \models g_0 \wedge g_1)$ if and only if $(v \models g_0$ and $v \models g_1)$.

If a valuation v does not satisfy a guard g, we denote it $v \not\models g$.

Another type of guards we will use in this thesis is *polyhedral guards*. We define it formally in Definition 2.2. These guards are defined with a polyhedron P of $\mathbb{R}^n$ and a valuation v satisfies this polyhedral guards if its belongs to P. This is a generalisation of **classical guards**. For instance, the timing constraint $x + 6y + 9 \leq 0 \wedge x - y \leq 0$ is a polyhedral guard.

Definition 2.2: Polyhedral guards

A *polyhedral guard* is denoted $g_{\mathbb{P}}$, where $\mathbb{P}$ is a polyhedron (as defined in Definition 1.4) in $\mathbb{R}^{\mathcal{C}}$. A clock valuation v satisfies a polyhedral guard $g_{\mathbb{P}}$, denoted $v \models g_{\mathbb{P}}$ if $v \in \mathbb{P}$. The set of **polyhedral guards** is denoted $\mathcal{G}_p(\mathcal{C})$.
A polyhedral guard is a *closed polyhedral constraint* if its associated polyhedron is closed.

Remark 2.1.1 *Let g be a polyhedral (or classical) guard over $\mathcal{C}$. Let v be a valuation in $\mathbb{R}_+^{|\mathcal{C}|}$ and $I \subseteq \mathbb{R}_+$ be an interal. We say that $v + I \models g$ if for any $\delta \in I$, $v + \delta \models g$.*

Remark 2.1.2 *Classical guards are a particular case of polyhedron constraints, where its associated matrix A of its **H-representation** is of the form $\begin{pmatrix} A_0 \\ A_1 \end{pmatrix}$, such that A_0 and A_1 are **diagonal** matrices of $\mathcal{M}_{n,n}(\mathbb{R})$.*

Proposition 2.3: Guards are convex

Let $g_{\mathbb{P}}$ be a polyhedral guard. Let $v, v' \in \mathbb{R}_+^{|\mathcal{C}|}$ and $\lambda \in [0,1]$ such that $v, v' \models g$. Then $\lambda \cdot v + (1-\lambda) \cdot v' \models g$.

Proof of Proposition 2.3. This is a direct consequence of the convexity of $\mathbb{P}$. Let P be the polyhedron associated with $g_{\mathbb{P}}$. As $\mathbb{P}$ is a convex set, $\lambda \cdot v + (1-\lambda) \cdot v' \in \mathbb{P}$. □

2.1.2 Timed automata

In this subsection, we present the notion of timed automata. Timed automata were first introduced by [AD94]. We first give their syntax in Definition 2.4 and some basic definitions, such as acyclic timed automata, in Definition 2.5.

Definition 2.4: Timed automata ([AD94])

A timed automaton is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, Q_f, \mathcal{C}, E)$ where:

- Σ is a finite *set of actions*,
- Q is a finite *set of locations*,
- $Q_0 \subseteq Q$ is the set of *initial* locations,
- $Q_f \subseteq Q$ is the set of *goals*,
- $\mathcal{C}$ is the finite set of clocks
- $E \subseteq Q \times \mathcal{G}_p(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times Q$ is a finite *set of transitions*.

A transition is a tuple $(\ell, g, a, \mathcal{C}_r, \ell')$ where $\ell \in Q$, $\ell' \in Q$, $g \in \mathcal{G}_p(\mathcal{C})$, $\mathcal{C}_r \subseteq \mathcal{C}$ and $a \in \Sigma$. A *closed* timed automaton is an automaton that uses only closed constraints.

Example 2.1.1 *Let us consider the timed automaton $\mathcal{A}_0$ and $\mathcal{A}_1$ of Figures 2.1 and 2.2:*

- *The set of clocks is $\{x, y\}$ and the set of locations is $\{\ell_0, \ell_1, \ell_2, \ell_f\}$ in $\mathcal{A}_0$ and $\mathcal{A}_1$. Likewise, the set of initial locations is $\{\ell_0\}$ and the set of goal locations is $\{\ell_f\}$. In these examples there is only one initial location and only one goal location.*

  An initial location is represented with an arrow, left to the state: 

. A goal location is double-circled: 

- *Guards are represented as labels above the edges:* $\begin{matrix} 0 \leq x \leq 1 \\ 0 \leq y \leq 2 \end{matrix}$ *for instance for* $g := 0 \leq x \leq 1 \wedge 0 \leq y \leq 2$.
- *Actions are represented as labels below the edges (a_0 for instance).*
- *The set of resets, if it is not empty, is represented by $y := 0$ for $\mathcal{C}_r = \{y\}$. In $\mathcal{A}_0$, the set of resets is empty in all transitions. In $\mathcal{A}_1$, the set of resets is empty for all transitions, except for the transition from ℓ_0 to ℓ_1 where $\mathcal{C}_r = \{y\}$.*

For simplification purposes, we now assume in the rest of this document (unless otherwise stated) that a timed automaton is a **closed** timed automaton and that the set of goal locations is a singleton.

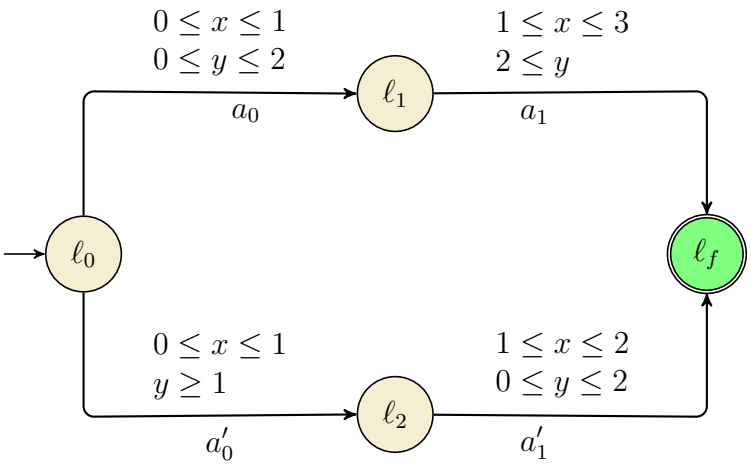


Figure 2.1: A two-clock timed automaton $\mathcal{A}_0$ with four transitions.

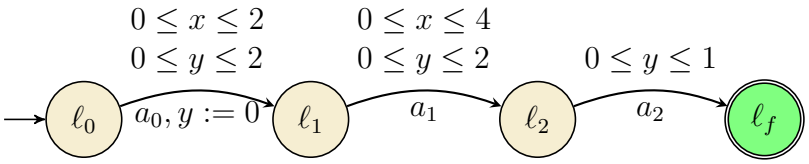


Figure 2.2: A two-clock timed automaton $\mathcal{A}_1$ with three transitions.

Remark 2.1.3 *Let $\mathcal{A} = (\Sigma, Q, Q_0, Q_f, \mathcal{C}, E)$ be a timed automaton. We denote by $G(\mathcal{A})$ its associated directed graph, i.e. a directed graph $G(\mathcal{A}) = (Q, Q_0, Q_f, E')$ where the set of transitions $E' \subseteq Q \times Q$ is defined by projecting away the guards and resets. More formally, $(\ell, g, a, \mathcal{C}_r, \ell') \in E$ if and only if $(\ell, \ell') \in E'$.*

Example 2.1.2 *Figure 2.3 represents the associated graph of the timed automaton $\mathcal{A}_1$ of Figure 2.2.*

Figure 2.3: The associated graph of the timed automaton $\mathcal{A}_1$ of Figure 2.2.

A timed automaton can also contains cycles or a unique transition per location. We define these types of timed automata in the following definition.

Definition 2.5: Single action, acyclic, linear timed automaton

Let $\mathcal{A} = (\Sigma, Q, Q_0, Q_f, \mathcal{C}, E)$ be a timed automaton.

- $\mathcal{A}$ is a *single-action* timed automaton if for any location $\ell \in Q$ and any action $a \in \Sigma$, there exists at most one transition from ℓ labelled with a. In other words, for any $\ell \in Q$ and $a \in \Sigma$, there exists at most one $\ell' \in Q$, $g \in \mathcal{G}_p(\mathcal{C})$ and $\mathcal{C}_r \subseteq \mathcal{C}$ such that the transition $(\ell, g, a, \mathcal{C}_r, \ell') \in E$.
- $\mathcal{A}$ is an *acyclic* timed automaton if its associated directed graph $G(\mathcal{A})$ is acyclic.
- $\mathcal{A}$ is a *linear* timed automaton if it is acyclic and if, from any location $\ell \in Q$, there exists at most **one** transition from ℓ.

  More formally, for any location $\ell \in Q$, there exists a unique 4-tuple $(\ell', a, \mathcal{C}_r, g)$ such that $\ell' \in Q$, $a \in \Sigma$, $\mathcal{C}_r \subseteq \mathcal{C}$, $g \in \mathcal{G}_p(\mathcal{C})$ and

$$(\ell, g, a, \mathcal{C}_r, \ell') \in E.$$

For the rest of this thesis, we will consider only **single-action** timed automata.

Example 2.1.3 *Let us again consider the two timed automata from Figures 2.1 and 2.2. $\mathcal{A}_0$ and $\mathcal{A}_1$ are both acyclic and single-action, but only $\mathcal{A}_1$ is linear. Indeed, in $\mathcal{A}_0$, two transitions are available from ℓ_0.*

Definition 2.6: The largest constant

Let $\mathcal{A} = (\Sigma, Q, Q_0, Q_f, \mathcal{C}, E)$ be a timed automaton. The *largest constant* of the automaton $\mathcal{A}$, denoted $\mathcal{M}(\mathcal{A})$, is the value of the largest constant that appears in the guards of $\mathcal{A}$.

Example 2.1.4 *In the timed automaton from Figures 2.1 and 2.2, the largest constant is 3 for $\mathcal{A}_0$ and 4 for $\mathcal{A}_1$. Indeed the constraints that contain the greatest constant are respectively $1 \leq x \leq 3 \wedge 2 \leq y$ and $0 \leq x \leq 4$.*

2.1.3 Classical semantics

A *configuration* of a timed automaton $\mathcal{A}$ is a pair (ℓ, v) where $\ell \in Q$ is a location of the automaton and v is a clock valuation. The semantics of a timed automaton is an infinite-

state labelled transition system $\mathcal{T} = (\mathcal{S}, \mathcal{S}_0, \mathcal{S}_F, \Delta)$ composed with a set of states $\mathcal{S}$, a set of initial states $\mathcal{S}_0$, a set of final states $\mathcal{S}_f$ and a set of transitions Δ.

The set of states is the set of configurations, the set of *initial states*, $\mathcal{S}_0 = Q_0 \times \mathbb{R}_+^{|\mathcal{C}|}$ is the set of configurations (ℓ, v) such that $\ell \in Q_0$. Such configurations are called *initial configurations*. The set of *final states*, $\mathcal{S}_f := Q_f \times \mathbb{R}_+^{|\mathcal{C}|}$ is the set of configurations (ℓ, v) such that $\ell \in Q_f$. Such configurations are called *goal configurations*.

The *transitions* are of two kinds:

- *delay transitions* model time elapsing: no transition of the timed automaton is taken, but the values of all clocks are increased by the same value. For any configuration (ℓ, v) and any delay $\delta \in \mathbb{R}_+$, there exists a transition $(\ell, v) \xrightarrow{\delta} (\ell, v + \delta)$.

- *action transitions* represent the effect of taking a transition in the timed automaton. For any configuration (ℓ, v) and any transition $e = (\ell, g, a, \mathcal{C}_r, \ell')$, if $v \models g$, then there exists a transition $(\ell, v) \xrightarrow{a} (\ell', v\,[\mathcal{C}_r \leftarrow 0])$.

Even if it means summing up the delays, we suppose that delay transitions and action transitions alternate. Finally, we write $(\ell, v) \xrightarrow{\delta,a} (\ell', v')$ when there exists a configuration (ℓ'', v'') such that $(\ell, v) \xrightarrow{\delta} (\ell'', v'')$ and $(\ell'', v'') \xrightarrow{a} (\ell', v')$.

In a timed automaton, we begin at an initial configuration (ℓ_0, v_0) and propose pairs of delays and actions to reach a successor of ℓ_0, then a successor of this successors, etc. The trace obtained by proposing these pairs and reaching successive configurations is called a *run*. It stores informations about the configurations reached and the sequence of delays-actions proposed. More formally:

Definition 2.7: Run

Let $n \in \mathbb{N} \cup \{+\infty\}$ and $\mathcal{A}$ be a timed automaton. A run of length n can be defined as follows:

- If $n = 0$, the run is empty and is denoted, by convention (s_0) where s_0 is the initial configuration of the run.
- $\rho := (s_i, (\delta_i, a_i), s_{i+1})_{0 \leq i \leq n-1}$ such that for any $i \in [0 \cdots n-1], s_i \xrightarrow{\delta_i, a_i} s_{i+1}$.

Let us introduce some additional definitions:

- The length of ρ is defined as follows: $|\rho| = n$.
- For any run ρ, $\rho_{\leq j}$ is the finite prefix of ρ $(s_i, (\delta_i, a_i), s_{i+1})_{0 \leq i \leq j-1}$.
- ρ is a *finite run* if $n \in \mathbb{N}$, otherwise the run ρ is called *infinite*.
- Let $n \neq 0$, a finite run $\rho = (\rho_i)_{1 \leq i \leq n}$ is *accepting* from a given configuration (ℓ_0, v_0) if it visits a goal $\ell_f \in Q_f$, *i.e.* if, denoting $\rho_i = ((\ell_i, v_i), (\delta_i, a_i), (\ell_{i+1}, v_{i+1}))_{0 \leq i \leq n-1}$, there exists an integer $j \in [1 \cdots n]$ and a goal $\ell_f \in Q_f$ such that $\ell_j = \ell_f$.
- An empty run $((\ell_0, v_0))$ is *accepting* if $\ell_0 \in Q_f$.

For the sake of simplification, the run ρ can also be denoted as follows:

$$s_0 \xrightarrow{\delta_0, a_0} s_1 \xrightarrow{\delta_1, a_1} s_2 \to \cdots \to s_{n-1} \xrightarrow{\delta_{n-1}, a_{n-1}} s_n.$$

Remark 2.1.4 *Even if it means adding a sink state and corresponding transitions, we assume that from any configuration, there always exists a transition* $\xrightarrow{\delta, a}$ *for some* $\delta \in \mathbb{R}_+$ *and some* $a \in \Sigma$. *This way, any* ***finite*** *run can be extended into an* ***infinite*** *run (in terms of its number of transitions).*

Example 2.1.5 *Let us consider again the examples of timed automata from Figures 2.1 and 2.2. We can propose a finite run of length 2 for the timed automaton* $\mathcal{A}_0$ *defined as follows:*

$$(\ell_0, (0, 0)) \xrightarrow{1, a_0} (\ell_1, (1, 1)) \xrightarrow{1, a_2} (\ell_f, (2, 2))$$

We can also propose a finite run of length 4 *for the timed automaton* $\mathcal{A}_1$

$$(\ell_0,(0,0)) \xrightarrow{2,a_0} (\ell_1,(2,0)) \xrightarrow{0.5,a_2} (\ell_2,(2.5,0.5)) \xrightarrow{0.5,a_2} (\ell_f,(3,1))$$

Let us remark that, because of the reset on the second clock in the first transition, the valuation of the second configuration is $(2,0)$.

2.1.4 Verification of timed properties

A fundamental problem in timed automata theory is the *reachability* of a location, *i.e.* given a specific initial configuration, can we reach a goal location?

Definition 2.8: Reachability problem, [AD94]

Given a timed automaton $\mathcal{A}$, an initial configuration $(\ell_0, v_0) \in Q \times \mathbb{R}_+^{|\mathcal{C}|}$, the reachability problem asks whether there exists an accepting run from (ℓ_0, v_0) in the infinite-state transition system defining the semantics of $\mathcal{A}$.

Example 2.1.6 *Let us consider again the examples of timed automata from Figures 2.1 and 2.2.*

In $\mathcal{A}_0$*, the goal can be reached from any initial configuration of the form* $(\ell_0,(x_0,y_0))$ *such that* $0 \leq x_0 \leq 1$ *and* $0 \leq y \leq 2$.

In $\mathcal{A}_1$*, the configuration* $(\ell_0,(2.1,0))$ *is not an initial configuration from which the goal can be reached. A goal location can be reached from the initial configuration* $(\ell_0,(0,0))$ *or* $(\ell_0,(0,2))$*, or more generally from any configuration of the form* $(\ell_0,(0,t))$ *where* $t \in [0,2]$.

The reachability problem was proved decidable and in PSPACE in [AD94], using an abstraction called *clock region*. This abstraction allows us to solve the reachability problem on a finite-state automaton by abstracting the notion of time. A clock region is an equivalence class over the set of configurations. In this equivalent class, if two valuations v and v' are in the same region, then (ℓ, v) and (ℓ, v') have the same successor region. Therefore these two configurations will be able to reach the same future locations. Then, when studying reachability, we only have to consider the classes of configurations that a set of configurations can reach. We formally define *clock region* in Definition 2.9.

Let us first recall the definition of a fractional function. For any $t \in \mathbb{R}$, $\mathsf{fract}\,(t) := t - \lfloor t \rfloor$ where $\lfloor t \rfloor$ is the integral part of t.

Definition 2.9: Clock region, [AD94]

Let $\mathcal{A} = (\Sigma, Q, Q_0, Q_f, \mathcal{C}, E)$ be a timed automaton. The set of valuations of $\mathcal{C}$ has an equivalence relation that we denote $\sim_{\mathcal{R}}$. $v \sim_{\mathcal{R}} v'$ if the following conditions hold, for any clocks $x, x' \in \mathcal{C}$:

1. $(\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor) \vee (\lfloor v(x) \rfloor \geq \mathcal{M}(\mathcal{A}) \wedge \lfloor v'(x) \rfloor \geq \mathcal{M}(\mathcal{A}))$.
2. If $v(x) \leq \mathcal{M}(\mathcal{A})$ and $v(x') \leq \mathcal{M}(\mathcal{A})$, then $\mathsf{fract}(v(x)) \leq \mathsf{fract}(v(x')) \Leftrightarrow \mathsf{fract}(v'(x)) \leq \mathsf{fract}(v'(x'))$.
3. If $v(x) \leq \mathcal{M}(\mathcal{A})$, then $\mathsf{fract}(v(x)) = 0 \Leftrightarrow \mathsf{fract}(v'(x)) = 0$.

The equivalence class of v is called a *clock region* and is denoted r_v. The set of clock regions is denoted $\mathbb{R}_+^{|\mathcal{C}|} / \sim_{\mathcal{R}}$.
Let r and r' be two clock regions. r' is a *time-successor* of r if the following condition holds:

$$\forall v \in r, \exists t \in \mathbb{R}_+, v + t \in r'$$

Example 2.1.7 *Let us consider a constraint* $0 \leq x, y \leq 2$. *The largest constant is* 2 *and there are three types of clock regions:*

- Corner points*: these are punctual regions. The corner point regions here are* $(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2)$.
- Open-line segments*: these are region represented with segments.*

  For instance $\{(x,y) \mid x = 0, 0 < y < 1\}$ *and* $\{(x,y) \mid x = 1, 0 < y < 1\}$.
- Open regions*: these are all the other regions described only with strict inequalities.*

  For instance $\{(x,y) \mid 0 < x < y < 1\}$ *and* $\{(x,y) \mid 0 < y < x < 1\}$.

These clock regions from the first and the second constraints of the timed automaton $\mathcal{A}_3$ *are illustrated in Figure 2.4.*

Region automata are finite-state automata. We use them to solve the problem of reachability of timed automata. The region automaton is a finite-state labelled transition system where configurations are clock regions.[AD94] gives a bound on the number of clock regions for a timed automaton $\mathcal{A}$:

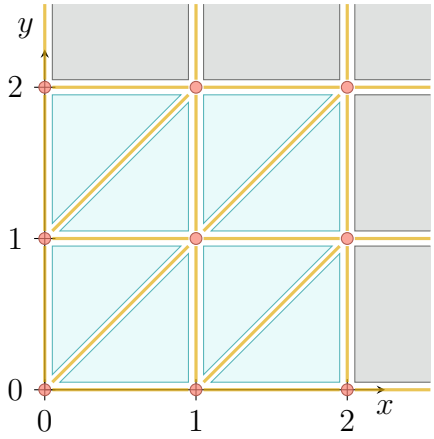


Figure 2.4: Clock regions of Example 2.1.7.

$$\left|\mathbb{R}_+^{|\mathcal{C}|}/\sim_{\mathcal{R}}\right| \leq \left\lfloor |\mathcal{C}|! \cdot 2^{|\mathcal{C}|} \cdot \prod_{x\in\mathcal{C}} \left(2\mathcal{M}\left(\mathcal{A}\right)+1\right)\right\rfloor$$

We can now formally define region automata in Definition 2.10.

Definition 2.10: Region automata, [AD94]

Let $\mathcal{A} = (\Sigma, Q, Q_0, Q_f, \mathcal{C}, E)$ be a timed automaton. The *region automaton* $R(\mathcal{A})$ is a finite automaton where:

- The set of locations is $\left\{(\ell, r) \mid r \in \mathbb{R}_+^{|\mathcal{C}|}/\sim_{\mathcal{R}}\right\}$
- The set of initial locations is $\left\{(\ell_0, r_{v_0}) \mid \ell_0 \in Q_0, v_0 \in \mathbb{R}_+^{|\mathcal{C}|}, \forall x \in \mathcal{C}, v_0(x) = 0\right\}$
- $e = ((\ell, r), (\ell', r'), a)$ is a transition of $R(\mathcal{A})$ if and only if there exists a transition $(\ell, \ell', a, \mathcal{C}_r, g)$ of $\mathcal{A}$ and a clock region r' such that there exists r'' a time-successor of r such that $r'' \models g$ and $r' = r''[\mathcal{C}_r \leftarrow 0]$.

Example 2.1.8 *Let us build the region automaton associated to the timed automaton of Figure 2.5 from the initial configuration $x = 0$. The clocks regions to consider at location ℓ_1 are $\{0\}, \{1\}$ and $]0,1[$. The one to consider at location ℓ_f are $(0), (1),]0,1[, (2)$ and $]1,2[$. We build the region automaton from the initial configuration $x = 0$ only, for the sake of clarity, in Figure 2.6.*

To prove that reachability is in PSPACE, [AD94] proved a correspondance between the runs of a timed automaton $\mathcal{A}$ and the ones of the corresponding region automaton $R(\mathcal{A})$.

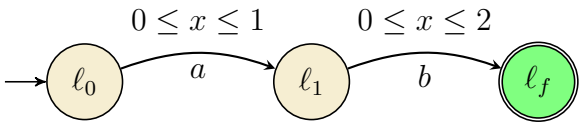


Figure 2.5: A simple timed automaton $\mathcal{A}_3$.

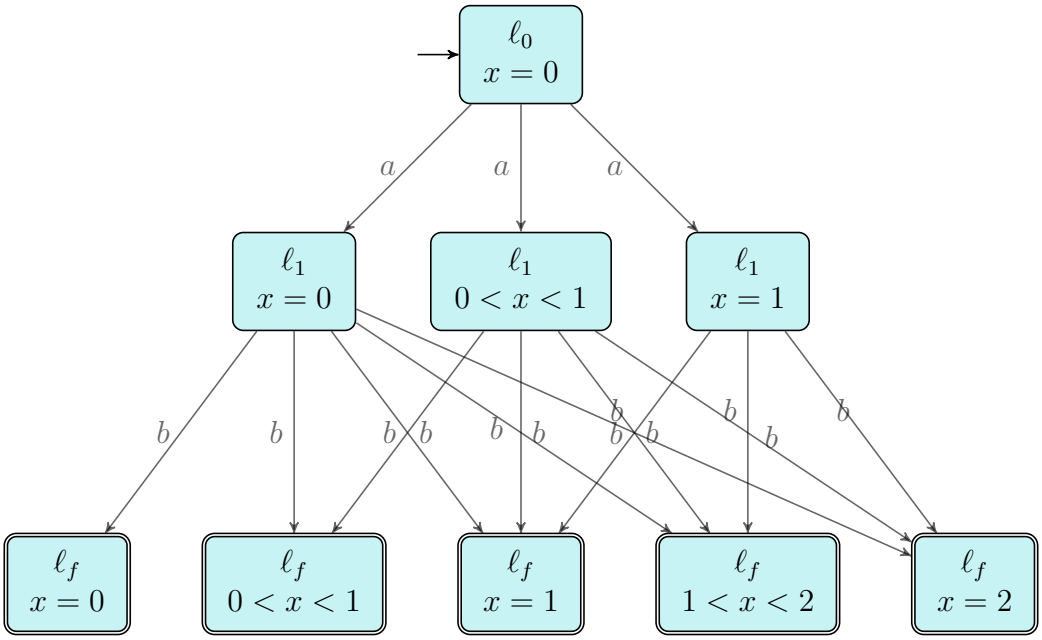


Figure 2.6: The region automaton of the timed automaton of Figure 2.5.

Checking reachability for a timed automaton $\mathcal{A}$ reduces to checking the reachability in the region automaton $R(\mathcal{A})$.

The limits of this abstraction are timing perturbations. Indeed, the semantics of timed automata and region automata assume that clocks are perfectly measured. In practice, clock measures are imperfect. The ability to verify a property despite perturbations is called robustness. This is a very general notion where we can consider several models. In Section 2.2, we present a model of robustness based on verifying reachability with perturbations on delays.

2.2 Robustness and permissiveness semantics

In this section, we present our approach to the robustness of timed automata. Our semantics, called *permissiveness semantics*, models timing perturbations by affecting delays of time-elapsing transitions. These perturbations can change the future delays and actions that will be available. Our goal is to check whether one can preserve the reachability of a goal location despite delay perturbations and to quantify the admissible perturbation. Other approaches, such as guards perturbations, are possible and will be presented in Chapter 3. In our model, we propose a game-based semantics to quantify how permissive we can be. In this semantics, the player proposes a pair of interval of delays $[\alpha, \beta]$ and an action. We call this pair a *p-move*. Then, the opponent chooses the delay in $[\alpha, \beta]$ that will be applied in the time-elapsing transition. We extend the notion of run in this semantics and call it p-run. To quantify how permissive the player has been in a p-run, we use a permissiveness function that depends on the sizes of the proposed intervals.

In Subsection 2.2.1, we present our semantics and the permissiveness function. To do so we explain how to compute the permissiveness of a p-move, of a p-run and finally of a strategy and a configuration. Then we explain the goal of the player and the opponent and present the permissiveness of timed automata. Then in Subsection 2.2.2 we present examples of permissiveness function for some examples of timed automata. Finally, in Subsection 2.2.3, we compare this notion of robustness to the one used in [BFM15] and extend these notions to more general operators.

2.2.1 Permissiveness semantics

Game

Let $\mathcal{A}$ be a timed automaton and Σ the alphabet of actions. Let us first explain how the game semantics works to check reachability with the example of Figure 2.7. In the **classical semantics**, the goal is to reach the location ℓ_f while proposing **delays** and verifying the guards. Let us consider the initial configuration $(\ell_0, (0))$. The player can propose a **delay**[1] to reach the location ℓ_1, for instance $\delta = 0.5$. Then the game has reached the configuration $(\ell_1, (0.5))$. He can again propose the **delay** 0.5 and the clock equals 1 and **the goal is reached**. In the *permissive semantics*, we use p-runs to extend the notion of runs. A **p-run** proceeds as follows:

- At the **initial configuration** $(\ell_0, (0))$, the player must **propose an interval**[2], $[\alpha, \beta]$ such that all delays in the interval satisfy the guard. In this example, the interval must satisfy $0 \leq \alpha \leq \beta \leq 1$. The player can propose $[0, 0.6]$ for instance.
- **After the player's choice**, the **opponent** chooses a **delay in the interval** $[0, 0.6]$ which will be applied, for instance 0.6.
- The game then has reached the **configuration** $(\ell_1, (0.6))$. The player must then propose an interval, *e.g.* $[0, 0.4]$, such that each delay of this interval satisfies the second guard.
- The opponent chooses a delay in $[0, 0.4]$. As each delay in this interval allows to reach the goal, whatever delay he chooses, the run will finish and reach the goal.

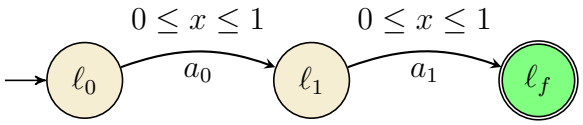


Figure 2.7: A one-clock timed automaton.

Let us present more formally the **permissive semantics**. A p-run consists, for each configuration (ℓ, v), in the following steps:

[1] As only one action is available here for each location, we will only detail the proposed delay and not the action.

[2] Possibly a singleton.

- ▷ The player proposes an **interval** $I = [\alpha, \beta]$ and an **action** a such that there exists a transition $(\ell, g, a, \mathcal{C}_r, \ell')$ such that $v + I \models g$.
- ▷ The opponent proposes a **delay** δ **which belongs to the interval** I.
- ▷ The configuration $(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0])$ is reached.

P-move

Each time the player proposes an **interval** instead of a **delay**, this represents the imprecision he will allow. For instance proposing $[0, 1]$ instead of 0.5 means allowing the delay to be greater or lower than 0.5, with a precision of 0.5. We can quantify this imprecision and say that the player allows a delay of range 1. More formally, we say that proposing an action and an interval of delays is a *p-move* (see Definition 2.11). The **size** of the interval proposed is the *permissiveness* of this p-move (see Definition 2.13).

Definition 2.11: P-move and enabled p-move

Let (ℓ, v) be an arbitrary configuration, **a p-move** of (ℓ, v) is a pair (I, a), where $I \subseteq \mathbb{R}_+$ is a closed interval, possibly right-unbounded, and $a \in \Sigma$.
We say that a p-move (I, a) is an **enabled p-move** if there is a transition $(\ell, g, a, \mathcal{C}_r, \ell')$ such that $v + I \models g$. The set of enabled p-moves of (ℓ, v) is denoted $\mathsf{p\text{-}moves}\,(\ell, v)$.

Remark 2.2.1 *P-moves can be defined with* ***open*** *or* ***half-open*** *intervals too. Nevertheless, we will consider only closed intervals for the purpose of simplification. We denote $\mathcal{I}$ the set of closed intervals of $\mathbb{R}_+$ and detail the extention to non-closed interval in Section 5.2.1.*

Let us first remark that, given a location ℓ and two different valuations v and v', the p-moves of (ℓ, v) can be mapped to those of (ℓ, v'), with a shifted interval. Let us prove this result formally in Lemma 2.12.

Lemma 2.12

Let v and v' be two clock valuations and suppose $\mathsf{p\text{-}moves}\,(\ell, v) \neq \varnothing$. Then for any p-move $([\alpha, \beta], a) \in \mathsf{p\text{-}moves}\,(\ell, v)$, for any sub-interval $I' \subseteq [\alpha + \|v' - v\|_\infty, \beta - \|v' - v\|_\infty]$, if $\beta - \alpha \geq 2\|v' - v\|_\infty$, then $(I', a) \in \mathsf{p\text{-}moves}\,(\ell, v)$.

Proof of Lemma 2.12. As guards are convex, if $([\alpha + \|v' - v\|_\infty, \beta - \|v' - v\|_\infty], a) \in \mathsf{p\text{-}moves}(\ell, v)$, then for any subinterval $I' \subseteq [\alpha + \|v' - v\|_\infty, \beta - \|v' - v\|_\infty]$, $(I', a) \in \mathsf{p\text{-}moves}(\ell, v)$.

Let us prove that $([\alpha + \|v' - v\|_\infty, \beta - \|v' - v\|_\infty], a) \in \mathsf{p\text{-}moves}(\ell, v)$. First, $[\alpha + \|v' - v\|_\infty, \beta - \|v' - v$ is an interval as $\alpha + \|v' - v\|_\infty \leq \beta - \|v' - v\|_\infty$. As $([\alpha, \beta], a) \in \mathsf{p\text{-}moves}(\ell, v)$, there exists a guard g of a transition from ℓ such that $v + [\alpha, \beta] \models g$. On top of that, for any clock x:

$$v(x) + \alpha \leq v'(x) + \|v' - v\|_\infty + \alpha$$
$$v(x) + \beta \geq v'(x) - \|v' - v\|_\infty + \beta$$

This concludes the proof. □

Let us now define the permissiveness of a p-move in Definition 2.13.

Definition 2.13: Permissiveness of a p-move

Let (I, a) be a p-move. The permissiveness of a p-move (I, a), denoted $\mathrm{Perm}(I, a)$, is defined as follows:

$$\mathrm{Perm}(I, a) = |I|.$$

Example 2.2.1 *Let us consider the timed automaton of Figure 2.7. Let $g : 0 \leq x \leq 1$ be the guard of this timed automaton. Let $(\ell_0, (0.5))$ be a configuration. $([0, 0.5], a_0)$ and $([0.3, 0.5], a_0)$ are two enabled p-moves. $([0.5, 1], a_0)$ is not because $(1.5) \not\models g$.*

The permissiveness of the p-move $([0.3, 0.5], a)$ is $0.5 - 0.3 = 0.2$, and the permissiveness of the p-move $([0.5, 1], a)$ is 0.5.

P-run

As for the **classical semantics** in Section 2.1.3, we define the notion of *p-run* in the **permissiveness semantics**. This definition naturally extends to our permissiveness semantics by adding information about the chosen interval. In the classical semantics, we store in a p-run the delay δ_i, the action a_i and the successor s_{i+1}. In the permissive semantics, we also take into account the chosen interval I_i. The other classical definitions such as winning p-runs, length or prefix, naturally extend.

Definition 2.14: p-Run

Let $n \in \mathbb{N} \cup \{+\infty\}$.
A p-run of length n from an initial configuration s_0 is defined as follows:

- If $n = 0$, the run is empty and is denoted, by convention, (s_0) where s_0 is the initial configuration of the run.
- Otherwise, $\rho^{(p)}$ is an infinite sequence of 5−tuples $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq n-1}$ such that for any $i \in [0 \cdots n-1]$, $s_i, s_n \in Q \times \mathbb{R}_+^{|\mathcal{C}|}$, $I_i \in \mathcal{I}, \delta_i \in I_i$, $a_i \in \Sigma$, $s_i \xrightarrow{\delta_i, a_i} s_{i+1}$ and such that (I_i, a_i) is an enabled p-move from s_i.

As for a run in classical semantics, we can use the following notation:

$$s_0 \xrightarrow{I_0, a_0, \delta_0} s_1 \rightarrow \cdots \rightarrow s_{n-1} \xrightarrow{I_{n-1}, a_{n-1}, \delta_{n-1}} s_n$$

During a p-run, the player may have proposed several intervals of different lengths. Our permissiveness quantifies the robustness of a timed automaton. Let us remark that for each p-run $\rho^{(p)}$, there exists a unique run $\rho = s_0 \xrightarrow{\delta_0, a_0} s_1 \cdots s_{n-1} \xrightarrow{\delta_{n-1}, a_{n-1}} s_n$.

Example 2.2.2 *For example, let us consider the timed automaton of Figure 2.7. We can propose two p-runs $\rho_0^{(p)}, \rho_1^{(p)}$ defined as follows:*

$$\rho_0^{(p)} := (\ell_0, (0)) \xrightarrow{[0,1], a_0, 0.5} (\ell_1, (0.5)) \xrightarrow{[0,0.5], a_1, 0.5} (\ell_f, (1))$$

and

$$\rho_1^{(p)} := (\ell_0, (0)) \xrightarrow{[0,1], a_0, 1} (\ell_1, (1)) \xrightarrow{\{0\}, a_1, 0} (\ell_f, (1))$$

In this example, the second p-run $\rho_1^{(p)}$ has a less permissive second p-move than the first p-run $\rho_0^{(p)}$.

To encourage the player to be permissive at **each step** of the p-run, our permissiveness considers the smallest interval that the player has proposed. More formally:

Definition 2.15: Permissiveness of a p-run

Let $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq |\rho|-1}$ be a finite p-run, the permissiveness of the p-run $\rho^{(p)}$ is denoted $\text{Perm}\left(\rho^{(p)}\right)$ and is defined as follows:

$$\text{Perm}\left(\rho^{(p)}\right) = \min_{0 \leq i \leq \left|\rho^{(p)}\right|-1} (\text{Perm}(I_i, a_i)) = \min_{0 \leq i \leq \left|\rho^{(p)}\right|-1} (|I_i|).$$

Example 2.2.3 *Let us consider again the Example 2.2.2 of p-runs* $\rho_0^{(p)}$ *and* $\rho_1^{(p)}$ *of the timed automaton of Figure 2.7. The permissiveness of the first p-run is* $\text{Perm}\left(\rho_0^{(p)}\right) = \min(1, 0.5) = 0.5$ *and the permissiveness of the second p-run is* $\text{Perm}\left(\rho_1^{(p)}\right) = \min(1, 0) = 0$. *The most permissive p-run is the first one.*

Strategy

The permissiveness of the p-run is not sufficient to express the permissiveness of a timed automaton, as the p-run depends on the **choices** of the player and the opponent. Maximising the proposed interval implies maximising it **whatever the choice of the opponent is**. To compute the optimal interval, we consider the **worst-case** where the opponent chooses the delay **minimising** the permissiveness. To model this behaviour, we first define what a permissive strategy is and express the permissiveness of the strategy of the player.

Definition 2.16: Permissive strategy

A permissive strategy is a (partial) function σ that maps finite p-runs $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq |\rho|-1}$ to p-moves in $\mathsf{p\text{-}moves}(s_n)$. We can say that:

- A p-run $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq |\rho|-1}$ is **compatible** with a permissive strategy σ if for all its prefix $\rho_{\leq j}^{(p)}$ such that $j \leq n$, $\sigma\left(\rho_{\leq j}^{(p)}\right) = (I_j, a_j)$. We say that $\rho^{(p)}$ is an outcome from the initial configuration s_0. The set of outcomes from a configuration is denoted $\text{Out}(s_0, \sigma)$.
- A permissive strategy σ is **winning** from a given configuration s_0 if any infinite p-run originating from s_0, that is compatible with σ, is winning.

Example 2.2.4 *If we consider again the timed automaton* $\mathcal{A}_3$ *of Figure 2.7 and the strategy* σ *defined as follows: For any p-run* $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq 1}$, *where* $s_i = (\ell_i, (x_i))$, $\sigma\left(\rho_{\leq i}^{(p)}\right) = ([0, 1 - x], a_i)$ *if* $0 \leq x \leq 1, i \in \{1, 2\}$ *(otherwise no p-move can be*

proposed). The p-runs $\rho_0^{(p)}$ *and* $\rho_1^{(p)}$ *from Example 2.2.2 are compatible with* σ *and* σ *is a winning strategy from any configuration* (ℓ, v) *such that* $v(x) \in [0, 1]$.

The permissiveness of a strategy considers the outcome that minimises the permissiveness function of all p-runs. It corresponds to the optimal choice of the opponent. For a non-winning strategy, the goal is not reachable and the permissiveness is $-\infty$. When the goal ℓ_f is already reached, there are no intervals to propose. By convention, the permissiveness is then $+\infty$. More formally:

Definition 2.17: Permissiveness of a strategy

Let σ be a permissive strategy and (ℓ, v) be a configuration. The permissiveness of σ in (ℓ, v), denoted $\mathrm{Perm}_\sigma(\ell, v)$, is defined as follows:

$$\mathrm{Perm}_\sigma(\ell, v) = \begin{cases} +\infty & \text{if } \ell \in Q_f \\ -\infty & \text{if } \sigma \text{ is not winning from } (\ell, v) \\ \inf\limits_{\rho^{(p)} \in \mathrm{Out}((\ell,v),\sigma)} \left(\mathrm{Perm}\left(\rho^{(p)}\right)\right) & \text{otherwise} \end{cases}$$

Example 2.2.5 *Let us go back to the Example 2.2.4, the permissiveness of* σ *from* (ℓ, v) *is:*

$$\mathrm{Perm}_\sigma(\ell, v) = \begin{cases} +\infty & \textit{if } \ell \in Q_f \\ -\infty & \textit{if } \sigma \textit{ is not winning from } (\ell, v) \\ 1 - x & \textit{otherwise} \end{cases}$$

Permissiveness of a configuration

Our goal is to compute the maximally-permissive strategies among all the possible strategies of the player. To do so, we first define the set of configurations where a winning strategy can be computed, called *winning configurations* in Definition 2.18. We compare it with the set of co-reachable configurations in Definition 2.19

Definition 2.18: Winning configuration

A configuration (ℓ, v) is **winning** if there exists a winning permissive strategy from (ℓ, v) (possibly proposing punctual intervals). Let ℓ be an arbitrary location, the set of winning configurations from ℓ is denoted Win_ℓ and is defined as follows:

$$\mathsf{Win}_\ell := \left\{ v \in \mathbb{R}_+^{|C|} \mid (\ell, v) \text{ is a winning configuration} \right\}$$

We also define the set of all winning configurations, denoted Win as follows:

$$\mathsf{Win} := \bigcup_{\ell \in Q} \mathsf{Win}_\ell$$

Definition 2.19: Co-reachable configurations

A configuration s is **co-reachable** from a configuration s' if there exists a p-run from s to s', *i.e.* if there exists a p-run $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq n-1}$ where $s_0 = s$ and $s_n = s'$.
A configuration s is **co-reachable** from a **location** ℓ if there exists a valuation v such that s is co-reachable from (ℓ, v).
A configuration s is **co-reachable** from a set of configurations $\mathcal{S}' \subseteq \mathcal{S}$ (*resp* locations $\mathcal{L}' \subseteq \mathcal{L}$) if there exists a configuration $s \in \mathcal{S}'$ (*resp.* location $\ell \in \mathcal{L}'$) such that s is co-reachable from s (*resp.* ℓ).
The set of co-reachable configurations from a set of configurations $\mathcal{S}'$ (*resp.* locations $\mathcal{L}'$) is denoted $\mathsf{co\text{-}reach}(\mathcal{S}')$ (*resp.* $\mathsf{co\text{-}reach}(\mathcal{L}')$).

Proposition 2.20

Let $\mathcal{A}$ be a timed automaton with a set of goals Q_f, then the following equality holds:

$$\mathsf{Win} = \mathsf{co\text{-}reach}(Q_f).$$

Proof of Proposition 2.20. As a permissive strategy can propose punctual intervals, we can propose a permissive strategy that is compatible with a unique p-run, from an initial configuration. Then any configuration s is winning *if and only if* there exists a run from s to a final state. As a result $\mathsf{Win} = \mathsf{co\text{-}reach}(Q_f)$ □

Example 2.2.6 *Let us go back to the timed automaton $\mathcal{A}_3$ of Figure 2.7. The set of*

winning configurations is:

$$\mathit{Win} = \bigcup_{i=0,1} \{(\ell_i, x) \mid 0 \leq x \leq 1\} \cup \{(\ell_f, x) \mid x \in \mathbb{R}_+\}$$

We now can compute a maximally-permissive strategy by computing a strategy that maximises the permissiveness of the player's strategy. Such a value is called the *permissiveness of a configuration* and is denoted Perm. More formally:

Definition 2.21: Permissiveness of a configuration

Let (ℓ, v) be a configuration. The permissiveness of (ℓ, v) is denoted $\mathrm{Perm}(\ell, v)$ and is defined as follows:

$$\mathrm{Perm}(\ell, v) = \begin{cases} +\infty & \text{if } \ell \in Q_f \\ -\infty & \text{if } (\ell, v) \text{ is not a winning configuration} \\ \sup_\sigma \mathrm{Perm}_\sigma(\ell, v) & \text{otherwise} \end{cases}$$

An example of computation of the permissiveness function of a configuration will be detailled in Subsection 2.2.2. To sum up, let us consider the configuration (ℓ, v) in Win_ℓ, where $\ell \neq \ell_f$. Let us denote each p-run $\rho^{(p)}$ as $\rho^{(p)} = \left(s_i^{\rho^{(p)}}\left(I_i^{\rho^{(p)}}, a_i^{\rho^{(p)}}\right), \delta_i^{\rho^{(p)}}, s_{i+1}^{\rho^{(p)}}\right)_{0\leq i\leq n-1}$. The permissiveness of (ℓ, v) is defined as follows:

$$\mathrm{Perm}(\ell, v) = \sup_\sigma \inf_{\rho^{(p)} \in \mathrm{Out}((\ell,v),\sigma)} \left[\min_{0\leq i\leq n-1} \left(\left|I_i^{\rho^{(p)}}\right|\right)\right]$$

Our goal in this thesis is to answer the following problem:

Definition 2.22: The maximal-permissiveness problem

Given a timed automaton $\mathcal{A}$, an initial configuration (ℓ_0, v_0) and a goal location ℓ_f, the maximal-permissiveness problem asks to compute $\mathrm{Perm}(\ell_0, v_0)$.

We answer this problem in Chapter 5 first with a **backward** algorithm, where we compute the function $v \mapsto \mathrm{Perm}(\ell, v)$ over Win_ℓ for the successors ℓ of the location ℓ_0. This function is called the *permissiveness function*. We present an algorithm that computes this function in **non-elementary time** for acyclic timed automata and games in Chapter 5. In the next section, we give two examples of computation of the permissiveness function on timed automata.

2.2.2 Examples of computation of permissiveness functions

We first present an intuitive way to compute the permissiveness of timed automata. We compute step-by-step the permissiveness of the initial locations of the two following timed automata (see Figures 2.2,2.10).

A first timed automaton

Our first example is a simple timed automaton with two identical transitions (see Figure 2.8). The two transitions have the same constraints on the guards: the clocks must be between 0 and 1.

Let us look at different possible p-runs starting from an initial configuration $(\ell_0, (0,0))$. For instance, if the player proposes the p-move $([0,1], a_0)$, the opponent may prevent the player from proposing a permissive interval by choosing the delay $\delta = 1$. If the opponent chooses delay 1, the configuration $(\ell_1, (1,1))$ is reached and the player has no choice but to propose the p-move $(\{0\}, a_1)$. The permissiveness of this p-move is 0. As a consequence, the player may have to propose a smaller interval. If he proposes an interval $[0, \beta]$ such that $\beta < 1$ and if the opponent chooses a delay $\delta \in [0, \beta]$, the configuration (ℓ_1, δ, δ) is reached. The largest interval the player can then propose is $[0, 1-\delta]$ and the permissiveness of this p-run is $\min(1-\delta, \beta)$. In this p-run, the player should propose $\left[0, \frac{1}{2}\right]$ to maximise the permissiveness, and the opponent should choose $\delta = \frac{1}{2}$ to minimise it. The intuition of the best strategy for a more general configuration and choice of interval is to split equally the intervals that the player proposes. Let us prove it formally. To compute the permissiveness at ℓ_0, we first compute the future permissiveness for all the possible valuations at ℓ_f and at ℓ_1. We denote $g := (0 \leq x \leq 1 \wedge 0 \leq y \leq 1)$ the guard of the transitions of the timed automaton.

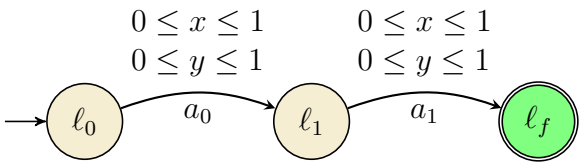


Figure 2.8: A timed automaton with 2 identical transitions.

Permissiveness at ℓ_f: for the goal location ℓ_f, the permissiveness is $+\infty$ for every valuation. As a result, the following equality holds:

$$\forall v \in \mathbb{R}^2_+, \mathrm{Perm}\,(\ell_f, v) = +\infty$$

Permissiveness at ℓ_1: we first compute the set of winning valuations at ℓ_1, denoted Win_{ℓ_1}. As every possible successor in ℓ_f is a winning configuration, the winning valuations at ℓ_1 are the ones that satisfy the guard $0 \leq x, y \leq 1$:

$$\mathsf{Win}_{\ell_1} = \{v_1 = (x_1, y_1) \in \mathbb{R}^2_+ \mid 0 \leq x_1 \leq 1 \wedge 0 \leq y_1 \leq 1\}$$

Let $v_1 = (x_1, y_1)$ be an arbitrary valuation in Win_{ℓ_1} and (I_1, a_1) an enabled p-move proposed by the player at ℓ_1. The permissiveness of the **p-run** is the minimum between $|I_1|$ and the permissiveness of the successor. As the successor is a goal configuration, its permissiveness is $+\infty$. As a result the permissiveness of the p-run is $|I_1|$.

According to the constraints of the guard, the largest interval that can be proposed from (ℓ_1, v_1) is $[0, \min(1 - x_1, 1 - y_1)]$. As a result:

$$\mathrm{Perm}\,(\ell_1, \cdot) : \ (x_1, y_1) \ \mapsto \ \begin{cases} \min(1 - x_1, 1 - y_1) & \text{if } (x_1, y_1) \in \mathsf{Win}_{\ell_1} \\ -\infty & \text{otherwise} \end{cases}$$

The best strategy for the player at $(\ell_1, (x_1, y_1))$, if $(x_1, y_1) \in \mathsf{Win}_{\ell_1}$ is thus to propose the p-move (I_1^*, a_1) where $I_1^* := [0, \min(1 - x_1, 1 - y_1)]$.

Permissiveness at ℓ_0: we compute the set of winning valuations at ℓ_0, Win_{ℓ_0}. The winning valuations are the ones such that there exists an enabled delay that satisfies the transition's guard and reaches a winning configuration at ℓ_1. More formally:

$$\begin{aligned} \mathsf{Win}_{\ell_0} &= \{v_0 = (x_0, y_0) \in \mathbb{R}^2_+ \mid \exists \delta_0 \geq 0, v + \delta_0 \models g \wedge v + \delta_0 \in \mathsf{Win}_{\ell_1}\} \\ &= \{v_0 = (x_0, y_0) \in \mathbb{R}^2_+ \mid \exists \delta_0 \geq 0, 0 \leq x + \delta_0, y + \delta_0 \leq 1\} \end{aligned}$$

We can easily eliminate δ_0:

$$\mathsf{Win}_{\ell_0} = \{v_0 = (x_0, y_0) \in \mathbb{R}^2_+ \mid 0 \leq x_0 \leq 1 \wedge 0 \leq y_0 \leq 1\}$$

Let $v_0 = (x_0, y_0)$ be an arbitrary valuation of Win_{ℓ_0}. The permissiveness at ℓ_0 is the minimum of all the intervals that have been proposed during the p-run where the player

maximises the permissiveness and where the opponent minimises it. As $v_0 \in \mathsf{Win}_{\ell_0}$, we can consider a p-run where the proposed p-moves are (I_0, a_0) and (I_1, a_1) and the proposed delays are respectively δ_0 and δ_1. The permissiveness of such run is $\min(|I_0|, |I_1|)$.

Let us suppose that the interval I_1 and the delay δ_0 and δ_1 are optimally chosen, then:

$$\begin{aligned}\min(|I_0|, |I_1|) &= \min\left(|I_0|, \inf_{\delta_0 \in I_0} \mathrm{Perm}(\ell_1, v_0 + \delta_0)\right) \\ &= \min\left(|I_0|, \inf_{\delta_0 \in I_0} \min(1 - x_0 - \delta_0, 1 - y_0 - \delta_0)\right)\end{aligned}$$

$\delta \mapsto \min(1 - x_0 - \delta, 1 - y_0 - \delta)$ is decreasing with respect to δ, so the opponent's best strategy is to choose the largest possible delay β_0. As a result:

$$\min(|I_0|, |I_1|) = \min(\beta_0 - \alpha_0, 1 - x_0 - \beta_0, 1 - y_0 - \beta_0)$$

The player's best strategy is thus to choose the interval $I_0 = [\alpha_0, \beta_0]$ that maximises $\min(|I_0|, |I_1|)$. As $\min(|I_0|, |I_1|)$ is decreasing with respect to α_0, the α_0 that maximises $\min(|I_0|, |I_1|)$ is 0. On the other side, the β_0 that maximises $\min(|I_0|, |I_1|)$ is the β_0 that is the closest to the β where $\beta_0 - \alpha_0 = 1 - y_0 - \beta_0$, there two function are equals when $\beta = \min\left(\frac{1 - x_0}{2}, \frac{1 - y_0}{2}\right)$. As the player can propose the p-move $\left(\left[0, \min\left(\frac{1 - x_0}{2}, \frac{1 - y_0}{2}\right)\right], a_0\right]$ if $(x_0, y_0) \in \mathsf{Win}_{\ell_0}$, the interval that maximises $\min(|I_0|, |I_1|)$ is $\left[0, \min\left(\frac{1 - x_0}{2}, \frac{1 - y_0}{2}\right)\right]$. As a result, we found an optimal p-move that the player can propose at ℓ_0 To compute the optimal interval, we used the fact that we computed in the previous paragraph its optimal strategy for the location ℓ_1, whatever the valuation was. As a result, we found the optimal intervals I_0 and I_1 and the permissiveness function for an arbitrary configuration (ℓ_0, v_0) values:

$$\mathrm{Perm}(\ell_0, \cdot) : \quad (x_0, y_0) \quad \mapsto \quad \begin{cases} \min(\frac{1 - x_0}{2}, \frac{1 - y_0}{2}) & \text{if } (x_0, y_0) \in \mathsf{Win}_{\ell_0} \\ -\infty & \text{otherwise} \end{cases}$$

Figure 2.9 represents the permissiveness function $\mathrm{Perm}(\ell_0, \cdot)$.

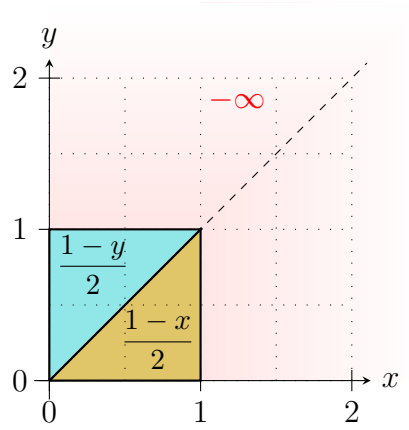


Figure 2.9: The permissiveness function at ℓ_0 for the timed automaton of Figure 2.8.

.

A two-transition timed automaton with a reset

We now consider the timed automaton in Figure 2.10. This timed automaton has a reset and in this example, the opponent's best strategy is not necessarily to choose the largest possible delay. We denote $g_1 := (1 \leq x \leq 2 \wedge 0 \leq y \leq 1)$ and $g_0 := (0 \leq x, y \leq 1)$ the guards of this timed automaton.

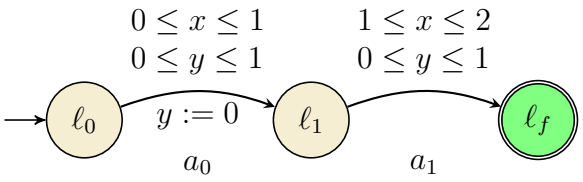


Figure 2.10: A timed automaton with a reset.

Permissiveness at ℓ_f: As in the previous example:

$$\forall v \in \mathbb{R}_+^2, \mathrm{Perm}\,(\ell_f, v) = +\infty$$

Permissiveness function in ℓ_1: the set of winning valuations at ℓ_1 can easily be computed:

$$\mathsf{Win}_{\ell_1} = \{(x_1, y_1) \in \mathbb{R}_+^2 \mid \exists \delta_1 \geq 0, (x_1, y_1) + \delta_1 \models g_1\}$$

Let us eliminate the variable δ_1:

$$\exists \delta_1 \geq 0, \begin{cases} 0 \leq 1 - x_1 \leq \delta_1 \leq 2 - x_1 \\ 0 \leq \delta_1 \leq 1 - y_1 \end{cases} \Leftrightarrow \begin{cases} 0 \leq 1 - x_1 \\ 1 - x_1 \leq 1 - y_1 \\ 1 - x_1 \leq 2 - x_1 \\ 0 \leq 1 - y_1 \end{cases} \Leftrightarrow (x_1 \leq 2, y_1 \leq x_1, y_1 \leq 1)$$

$$\mathsf{Win}_{\ell_1} = \{(x_1, y_1) \in \mathbb{R}_+^2 \mid x_1 \geq y_1 \wedge y_1 \leq 1 \wedge x_1 \leq 2\}$$

Let $v_1 = (x_1, y_1)$ be a valuation of Win_{ℓ_1}. The goal of the player at configuration (ℓ_1, v_1) is to propose an interval $I_1 := [\alpha_1, \beta_1]$ that maximises the permissiveness of the p-run, which is $\min(+\infty, \beta_1 - \alpha_1) = \beta_1 - \alpha_1$, while satisfying the guard g_1. As α_1 and β_1 must satisfy $1 \leq x_1 + \alpha_1 \leq x_1 + \beta_1 \leq 2$ and $0 \leq y_1 + \alpha_1 \leq y_1 + \beta_1 \leq 1$, the best strategy for the player is to propose $\alpha_1 = \max(1 - x_1, 0) = 1 - x_1$ and $\beta_1 = \min(1 - y_1, 2 - x_1)$. As a result:

$$\mathrm{Perm}(\ell_1, \cdot) : \ (x_1, y_1) \ \mapsto \begin{cases} x_1 - y_1 & \text{if } 0 \leq y_1 \leq x_1 \leq 1 \\ 1 - y_1 & \text{if } 0 \leq y_1 \leq 1, y_1 \geq -1 + x_1 \\ 2 - x_1 & \text{if } 1 \leq x_1 \leq 2, y_1 \leq -1 + x_1 \\ -\infty & \text{otherwise} \end{cases}$$

Figure 2.11a represents the permissiveness function $\mathrm{Perm}(\ell_1, \cdot)$.

Permissiveness function in ℓ_0: with the same methods, we can easily compute the set of winning valuations at ℓ_0:

$$\mathsf{Win}_{\ell_0} = \{(x_0, y_0) \in \mathbb{R}_+^2 \mid \exists \delta_0, v_0 + \delta_0 \models g_0, v_0 + \delta_0 [y \leftarrow 0] \in \mathsf{Win}_{\ell_1}\}$$
$$\mathsf{Win}_{\ell_0} = \{(x_0, y_0) \in \mathbb{R}_+^2 \mid x_0, y_0 \in [0, 1]\}$$

Let $v_0 = (x_0, y_0)$ be an arbitrary valuation of Win_{ℓ_0}. With the same arguments as in the previous example, we can compute an optimal p-move. Let us suppose that this p-move exists and denote it (I_0, a_0) where $I_0 = [\alpha_0, \beta_0]$. Then:

$$\mathrm{Perm}(\ell_0, v_0) = \min\left(\beta_0 - \alpha_0, \inf_{\delta_0 \in I_0} \mathrm{Perm}(\ell_1, v_0 + \delta_0 [y \leftarrow 0])\right)$$

After passing the first transition, clock y is reset. The next configuration $(x_0 + \delta_0, 0)$ satifies $0 \leq x_0 + \delta_0 \leq 1$. According to the permissiveness function at location ℓ_1, the permissiveness function of the successor is $x_0 + \delta_0$.

$$\text{Perm}\,(\ell_0, v_0) = \min\left(\beta_0 - \alpha_0, \inf_{\delta_0 \in I_0} x_0 + \delta_0\right)$$

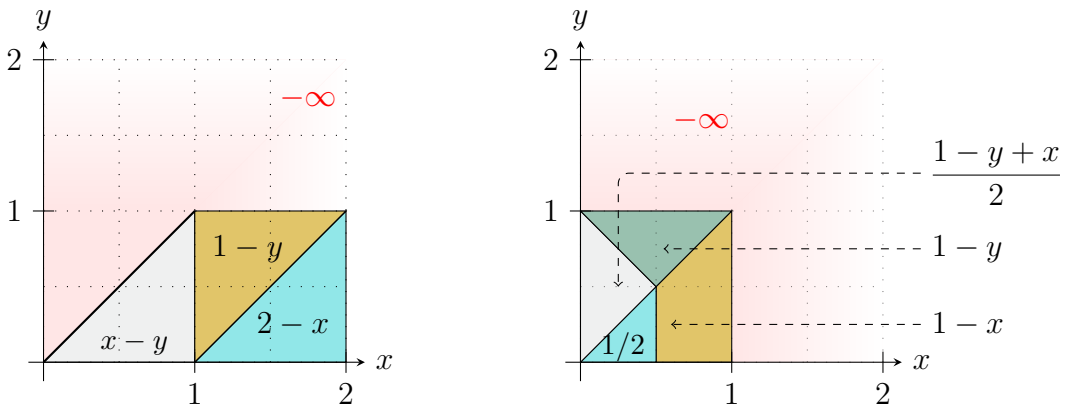
As $\delta_0 \mapsto x_0 + \delta_0$ is an increasing function on $[\alpha_0, \beta_0]$, the best strategy of the opponent is to propose the delay α_0.

$$\text{Perm}\,(\ell_0, v_0) = \min\left(\beta_0 - \alpha_0, x_0 + \alpha_0\right)$$

To maximise $\text{Perm}\,(\ell_0, v_0)$, β_0 has to be as large as possible, *i.e.* $\beta_0 = \min(1 - x_0, 1 - y_0)$. α maximises $\text{Perm}\,(\ell_0, v_0)$ when $\alpha_0 = \dfrac{\beta_0 - x_0}{2}$ if $0 \leq \dfrac{\beta_0 - x_0}{2} \leq \beta_0$ (if not, $\alpha_0 = 0$). As a result:

$$\text{Perm}\,(\ell_0, \cdot) : \quad (x_0, y_0) \quad \mapsto \quad \begin{cases} 1/2 & \text{if } 0 \leq x_0 \leq \dfrac{1}{2}, y_0 \leq x_0 \\ \dfrac{1 - y_0 + x_0}{2} & \text{if } 0 \leq x_0 \leq y_0, y_0 \leq 1 - x_0 \\ 1 - y_0 & \text{if } y_0 \geq 1 - x_0, x_0 \leq y_0 \leq 1 \\ 1 - x_0 & \text{if } \dfrac{1}{2} \leq x_0, 0 \leq y_0 \leq x_0 \leq 1 \\ -\infty & \text{Otherwise} \end{cases}$$

Figure 2.11b represents the permissiveness function $\text{Perm}\,(\ell_0, \cdot)$.



(a) The permissiveness function at ℓ_1. (b) The permissiveness function at location ℓ_0.

Figure 2.11: The permissiveness function in ℓ_0 and ℓ_1 for the timed automaton of Fig 2.10.

These two examples gave us a first intuition of the computation of the permissiveness function. To compute the permissiveness function, one should consider the full p-run and optimise each interval in order to maximise the minimum of the sizes of all intervals proposed during the p-run. To compute the optimal interval at each step of the p-run, we can consider the minimum of all the sizes of the future intervals we will have to propose. The future intervals we can propose on a configuration (ℓ, v) depend on the previous interval we had proposed, as v depends on the intervals and delays that have been proposed during the p-run. To avoid this dependence, we compute the permissiveness for **all** the possible valuations. Therefore, we can optimise step-by-step the future optimal intervals in order to finally compute the permissiveness function of the initial configuration. We will define in Chapter 4 a sequence of functions that tends to the permissiveness function, in order to compute it iteratively.

2.2.3 Possible extensions

Other approaches are possible to model and compute the robustness of timed automata. In this subsection, we present the *penalty*, used in [BFM15] to quantify the robustness of timed automata. We finally present a more general model of permissiveness.

Penalty

Penalty semantics was introduced in 2015 by [BFM15], which presents a type of robustness with delay perturbations. In their model, they consider the sum of all inverses of the sizes of all proposed intervals. They call this quantity the penalty. They give an algorithm to compute the strategy of the player that minimises this quantity.

Definition 2.23: Penalty, [BFM15]

We can define the penalty of a:

- **p-move**: let (I, a) be a p-move, then its penalty is $\frac{1}{|I|}$ if $|I| \neq 0$, and $+\infty$ when I is punctual.
- **finite p-run**: let $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq n-1}$ be a finite p-run, the penalty of $\rho^{(p)}$ is defined as the sum of $\frac{1}{|I_i|}$ if all I_i are non punctual, and $+\infty$ otherwise.
- **strategy**: let σ be a permissive strategy and (ℓ, v) a configuration, the penalty of σ in (ℓ, v) is defined as 0 if the location ℓ is the goal, $+\infty$ if σ is not a winning strategy and the supremum of the penalties of all the p-runs of the set $\mathrm{Out}((\ell, v), \sigma)$.
- **configuration**: let (ℓ, v) be a configuration. Then the penalty is the infimum of the penalties, among all strategies, in (ℓ, v).

To sum up, if we consider the configuration (ℓ, v), and if we write every p-run $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0 \leq i \leq n-1}$, the penalty of this configuration is defined as follows:

$$\inf_{\sigma} \left[\sup_{\rho^{(p)} \in \mathrm{Out}((\ell,v),\sigma)} \sum_{0 \leq i \leq n-1} \frac{1}{|I_i|} \right]$$

Example 2.2.7 *Let us consider the timed automaton of Figure 2.2. Let us compute the penalty over the set of winning configurations. As a result, the penalty at (ℓ_f, v) is 0 whatever the valuation v is. At (ℓ_1, v), the penalty is the inverse of the size of the greatest interval the player can propose. As a result it is $\frac{1}{1-x}$ if $1 > x > y > 0$, $\frac{1}{1-y}$ if $1 > y > x > 0$, and $+\infty$ otherwise. Then, at location ℓ_0, to compute the most-permissive strategy, we have to find an interval $[\alpha, \beta]$ that minimises the following quantities:*

$$\frac{1}{\beta - \alpha} + \sup_{\delta \in [\alpha, \beta]} \left(\frac{1}{\min(1 - x - \delta, 1 - y - \delta)} \right)$$

As $\sup_{\delta\in[\alpha,\beta]}\left(\frac{1}{\min(1-x-\delta,1-y-\delta)}\right)=\frac{1}{\min(1-x-\beta,1-y-\beta)}$, *this quantity can be simplified as follows:*

$$\frac{1}{\beta-\alpha}+\left(\frac{1}{\min(1-x-\beta,1-y-\beta)}\right)$$

If $x<y$, *then* α *should be as small as possible (i.e.* 0*) and* β *has to minimise* $\frac{1}{\beta}+\frac{1}{1-y-\beta}$. *It is equivalent to maximising* $(\beta)(1-y-\beta)$. *The optimal* β *is* $\frac{1-y}{2}$.

With the same reasoning, we can prove that, if $x\geq y$, *the optimal* α *is* 0 *and optimal* β *is* $\frac{1-x}{2}$.

To sum up, the penalty at configuration (ℓ_0,x_0,y_0) *is defined as follows:*

$$\mathit{penalty}(\ell_0,(x_0,y_0))=\begin{cases}\frac{4}{1-y_0} & \text{if } 0<x_0<y_0<1\\ \frac{4}{1-x_0} & \text{if } 0<y_0\leq x_0<1\\ +\infty & \text{otherwise}\end{cases}$$

Let us illustrate the difference between penalty and permissiveness in the following Example 2.2.8, where the player should choose the shortest path if he considers the penalty, and the longest, but with larger guard, if he considers the permissiveness.

Example 2.2.8 *Let us consider the timed automaton of Figure 2.12. The path with only one transition has smaller guard than the one with two transitions. By considering the permissiveness and the penalty computed on the timed automaton of Figure 2.2, we can easily compute the penalty and the permissiveness of* $(\ell_0,(0,0))$. *To compute the permissiveness (resp penalty), at location* ℓ, *we have to consider the path that* ***maximises*** *(resp* ***minimises****) the permissiveness (resp the penalty). As a result we compute the maximum between the permissiveness (resp penalty) of the two linear timed automata of Figure 2.13. At* $(\ell_0,(x,y))$, *if* $x\geq y$, *the permissiveness is* $8\cdot\max\left(\frac{1-x}{2},\left(\frac{3}{8}-x\right)\cdot\frac{1}{2}\right)$, *and the penalty is* $\min\left(\frac{1}{3-8\cdot x},\frac{1}{2-2x}\right)$.

At configuration $(\ell_0,(0,0))$, *the permissiveness is* 4 *and the player's optimal strategy is to choose the path with two transitions. On the contrary, the penalty is* $\min\left(\frac{1}{3},\frac{1}{2}\right)=\frac{1}{3}$ *and the player's optimal strategy is to choose the shortest path, even if the guard is smaller.*

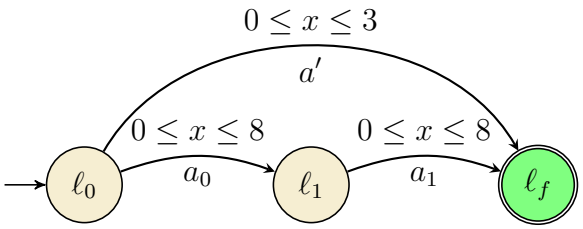


Figure 2.12: An acyclic timed automaton with a short and a long path.

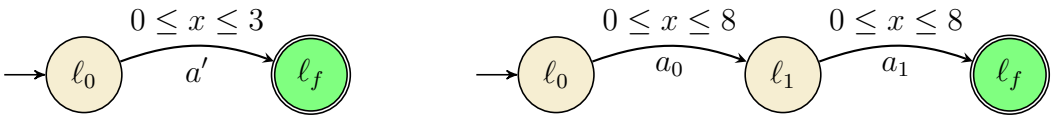


Figure 2.13: Decomposition of the timed automaton of Figure 2.12 into two linear timed automata.

As for the permissiveness, we can formally define the problem of computing the penalty of a configuration in Definition 2.24.

Definition 2.24: Problem of computing the penalty function ([BFM15])

Given a timed automaton $\mathcal{A}$, a configuration (ℓ_0, v_0), a goal location ℓ_f, the problem of computing the penalty function asks to compute the penalty of the configuration (ℓ_0, v_0). The associated decision problem asks, given a threshold p, if the penalty is less than p.

[BFM15] proved that this problem is in **PTIME** for one-clock timed automata.

General operator for permissiveness

To compute the optimal permissive strategy with multiple clocks, we chose the robustness model of Subsection 2.2.1. In the present subsection, we present a more general permissiveness model. Penalty and permissiveness are particular cases of these functions. An interesting future work would be to consider a more general operator than the minimum of the sizes of the intervals. Let us define first a more general permissiveness function. Let Op be a continuous operator from $\mathbb{R}_+^2$ to $\mathbb{R}_+$, f be a monotonic continuous function from $\mathcal{I}$ to $\mathbb{R}_+$ and k_- and k_+ be two constants in $\overline{\mathbb{R}}$.

Definition 2.25: Permissiveness for general operator

Let $n \geq 1$ and let $\rho^{(p)} = (s_i, (I_i, a_i), \delta_i, s_{i+1})_{0\leq i\leq n-1}$ be a finite p-run and $Op : \mathbb{R}_+^n \to \mathbb{R}_+$ and $f : \mathcal{I} \to \mathbb{R}_+$ where $\mathcal{I} = (I_i)_{0\leq i\leq n-1}$. The general permissiveness of a p-run, denoted $\text{Perm}^{Op,f}\left(\rho^{(p)}\right)$ is defined as follows:

$$\text{Perm}^{Op,f}\left(\rho^{(p)}\right) = Op\left(f\left(I_i\right)\right)_{i\in[0\cdots n-1]}$$

Let (ℓ, v) be a configuration of a timed automaton $\mathcal{A}$. The general permissiveness of a configuration (ℓ, v), denoted $\text{Perm}^{Op,f}(\ell, v)$, is defined as follows:

- If (ℓ, v) is not a winning configuration, $\text{Perm}^{Op,f}(\ell, v) = k_-$
- If ℓ is a **goal location**, $\text{Perm}^{Op,f}(\ell, v) = k_+$
- Otherwise, the general definition of the permissiveness function depends on the monotonicity of f:

$$\text{Perm}^{Op,f}(\ell, v) = \begin{cases} \sup\limits_{\sigma}\left[\inf\limits_{\rho^{(p)}\in\text{Out}((\ell,v),\sigma)} \text{Perm}^{Op,f}\left(\rho^{(p)}\right)\right] & \text{if } f \text{ is non-decreasing} \\ \inf\limits_{\sigma}\left[\sup\limits_{\rho^{(p)}\in\text{Out}((\ell,v),\sigma)} \text{Perm}^{Op,f}\left(\rho^{(p)}\right)\right] & \text{if } f \text{ is decreasing} \end{cases}$$

Let us give some specific examples of this general permissiveness function. First, if $(Op = \min, f(I) = |I|, k_+ = +\infty, k_- = -\infty)$, then it corresponds to the permissiveness function, defined in Subsection 2.2.1. Another model we can consider is the *weighted sum*, with non-negative coefficients (b_i). In this model $f(I) = \frac{1}{|I|}$, $Op\left((x_i)_{0\leq i\leq n}\right) = \sum_{i=0}^{n} x_i b_i$, $k_+ = 0$ and $k_- = +\infty$. The case where all b_i are 1 corresponds to the penalty function defined in [BFM15]. Finally, we can consider the **average** size of all intervals proposed with $f(I) = |I|$, $Op\left((x_i)_{0\leq i\leq n}\right) = \sum_{0\leq i\leq n} \frac{x_i}{n}$, $k_- = -\infty$ and $k_+ = +\infty$. These models represent different computational challenges and the optimal strategies can differs.

Conclusion

In this chapter, we presented different semantics to compute a type of robustness based on delay perturbation. Other approaches are possible, such as guard perturbation. The next

Chapter presents a state of the art of the different kinds of robustness and applications of this problem in model-checking and formal methods.

Chapter 3

State of the art: Robustness in real-time systems

In everyday life, being robust means being resistant to perturbations in our environment. For example, a tool is robust to water if it is waterproof, i.e. if it functions despite a significant exposure to water. The notion of robustness depends on the nature of the perturbation (for example we could look at the robustness against water or the robustness against fractures, called toughness). In the context of computer science, we have to specify the nature of the perturbations, and what it means to be functional against perturbations. The systems we consider are real-time systems, which are modelled by timed automata. In these systems, perturbations can occur either on clocks, or on constraints, or on both, and there are many models of them. In this chapter, we present a state of the art of the different definitions of robustness and the obtained results. To go further, two surveys, [Mar11] and [BMS13], on the robustness of timed automata are published. This chapter is organised as follows:

- In Section 3.1, we consider the first model that has been proposed for robustness: a topological viewpoint. It was proposed by [GHJ97], where trajectories of runs are enlarged as tubes.
- In Section 3.2, we present a model where the clocks are allowed to evolved with different speed was proposed. This approach was proposed by [Pur98].
- In Section 3.3, we consider a model of robustness in the context of time sampling. This approach was first proposed by [CHR02] for safety properties.
- In Section 3.4, we present two models where guards are perturbed.
- In Section 3.5, we consider models where delays are perturbed.

3.1 Topological approach

Tubes and metrics. [GHJ97] proposed a topological model of the robustness for timed automata. This article studies the emptiness problem and the language inclusion problem. Intuitively, a timed automaton is robust if a neighbour of an accepted trajectory is accepted too. Let us present more formally their model.

Let us consider an alphabet Σ, a trajectory is a (possibly infinite) sequence $(a_i, t_i)_i$ where $a_i \in \Sigma$ and $t_i \in \mathbb{R}_+$ represent respectively an event and a time gap, *i.e.* the duration of time between two events a_i and a_{i+1}.

The metric between two trajectories can take different forms, but respect the common following rules: consider two trajectories $(a_i, t_i)_i$ and $(a'_i, t'_i)_i$, then:

- These two trajectories are at finite distance if and only if their sequence of events are identical (*i.e.* if for any integer i, $a_i = a'_i$).
- If the previous conditions is verified, the metric between the two trajectories does not depends on the events $(a_i)_i$ and $(a'_i)_i$. It only depends on the time gaps $(t_i)_i$ and $(t'_i)_i$.

The *tube* of a trajectory τ, for a metric d and a real ε, is then the set of trajectories at distance at most ε from τ for the distance d, denoted $T_d(\tau, \varepsilon)$. This tube contains a neighbourhood of the trajectory τ.

Tube acceptance. This model considers timed automata as defined in [AD94]. A trajectory $\tau = (a_i, t_i)_i$ is **accepted** if there exists a run such that the events a_i correspond to the actions taken during the transitions, and the delays t_i to the time between two transitions. The trace of an automaton is then said to be robust if there exists a tube of this trajectory that is accepted.

Results. The paper focuses on checking emptiness and proved its decidability. [GHJ97] reduces the *tube acceptance problem* for an automaton to this problem with specific automata (called *interior automata*). This paper gave hope to prove the decidability of the language inclusion problem, that was unfortunately proven undecidable by [HR00] three years later.

The interest of this model is to isolate the non-robust behaviour by traces, some may not be robust but may also not be relevant. The limitation of this representation is that it does not take into account the structure of the automaton.

3.2 Clock drifting

Clock drifts is a model proposed by [Pur98]. It models the imprecision in the **synchronicity of the clocks**. In an abstract model, each clock evolves at the same speed and can be reset independently. In the real world, imprecision can occur and clocks can get out of sync. The purpose of the model of clock drifting is to verify the properties even when the speeds of the clocks are not equal.

Model of clock drifts Clock drifts is a model where the perturbations affects the clock speed. It consists of allowing the clock speed to be between $1-\Delta$ and $1+\Delta$ for a $\Delta > 0$, instead of all being equal to 1. This model, called Δ-drifting semantics, was introduced by [Pur98] and [Pur00].

The Δ-drifting semantics slightly changes a timed automaton $\mathcal{A}$ into a timed automaton $\mathcal{A}_\Delta$ where each clock x does not necessarily evolve with the same **speed**. For each clock x, its speed, denoted $\dot{x}$, is 1 in the classical semantics. In the Δ-drifting semantics, $\dot{x}$ is between $1-\Delta$ and $1+\Delta$.

Results [Pur98] and [Pur00] provide an algorithm that computes, given an automaton $\mathcal{A}$ and an initial state s_0, the reachable state from s_0 of the drifted automaton $\mathcal{A}_\varepsilon$ when ε tends to 0, using region graph, while [DK06] proposes an algorithm using zone. The argument of [DK06] is that, even if the worst-case complexity of their algorithm is the same, a symbolic algorithm might be more efficient. The limit of these two works is that they have to use the hypothesis for each cycle in the region graph, every clock must be reset at least once. In 2007, [Dim07] extends [Pur98] and [Pur00]'s work by providing a symbolic algorithm for non-necessary closed constraints.

In 2017, [RPV17] provides results for a model that considers both guard enlargement (see Section 3.4) and clock drifting. On top of that, [RPV17] provides results for reachability properties, as in [Pur98] and [Pur00], but removed the hypothesis on the reset of clocks. [DDM+08] studies perturbations model with guard enlargement mixed with clock drifting for safety properties. They enlarged guards by δ and drifted clocks by Δ. [DDM+08] provides an algorithm to decide whether there exists $\Delta > 0$ and $\delta > 0$ such that the safety property is still verified despite clock drifts and guard enlargement.

Other work extended Puri's model and work. For instance, [SFK08] considers Puri's model and the robustness for safety property. They used the UPPAAL tool to test safety robustness for closed guards. The first result of [SFK08] does not bound the clock drifting,

but supposes a finite number of iterations. Its second result does not suppose a finite number of iterations but imposes that $0 < \Delta < 1$.

Finally, other works used the model of clock drifting for different models of timed automata. [JR11] studied the computation of the largest possible Δ, while verifying safety properties. They prove the decidability of this problem for *flat* timed automata (each location belongs to at most one cycle). [ATM05] considered timed systems that can be express as the **product** of single-clock automata, with clock drifting Δ perturbations. Their contribution is to solve the inclusion problem. The space complexity of their algorithm is polynomial in the size of the timed automata. Other types of timed automata have been studied, as in [ABG+08] and [ABG+14] that studied clock drifting for distributed timed automata.

3.3 Time sampling

Time sampling models. A second approach to robustness is to sample time in the context of hybrid systems. In hybrid systems, continuous time and discrete time components coexist. In this context, the environment is represented in continuous time, and the controller in discrete time. This model is more realistic with respect to the implementation if we take into account the fact that controllers are digital systems.

Discrete-time control problem. [CHR02] introduced time sampling models and then raised the following question for the safety properties: can the controller avoid bad states if it only looks at discrete times, at regular intervals, that we call *sampling rate*? They formulate the *Discrete-time control problem with unknown sampling rate*: is there a $\beta \in \mathbb{Q}_+$ such that a property can be assured by a controller, if its sampling rate is β?

Indeed, let us consider an example (with a single clock system) where bad behaviour occurs between $x = 0.3$ s and 0.4 s and is not detectable at another times. If the sampling rate is $\beta > 0.1$ s, the controller may not detect it, but if $\beta \leq 0.1$ s, it may.

Results. [CHR02] showed that this problem is **undecidable** for **safety** properties. [AKY07] and [AKY10] proved the decidability of the existence of a sampling rate that makes sampled and continuous semantics recognise the same untimed languages. [KP05] used a slightly different semantics to prove two results: the **decidability** of this problem for **reachability** of timed automata, but the **undecidability**, for timed automata that

allow clocks to stop at some locations. Finally, [BLM+11] studied the **implementability** of timed automata while being *robust to clock drifting* and *samplable*, which means being able to construct an implementation that preserves the semantics even if time is sampled.

3.4 Perturbations on guards

Robustness can also be modelled as perturbations on guards. Two approaches have been considered. The first approach considers that the implementation might not behave as the idealized model. To model imprecisions, this approach uses *guard enlargement* on the timed automaton, and then checks whether properties are verified even on this enlarged timed automaton. The second approach, called *shrinking*, consists in respecting the initial constraints despite the perturbations of the model we design. To do this, this approach shrinks the guards of the timed automaton to give a more strict timed automaton. Then, the goal is to satisfy the desired property even with timing imprecisions.

Guard enlargement

Guard enlarged timed automata. *Guard enlargement* has been proposed by [BMR06]. It consists in designing and implementing directly the original timed automaton $\mathcal{A}$, which models the idealised real-time system, but to verify the robustness of our desired property φ on an *enlarged timed automaton* $\mathcal{A}_\delta$ where $\delta > 0$ is a fixed parameter. If this is verified, we then say that $\mathcal{A}$ is δ-robust. The verification of this robustness is called *robust model-checking*. Let us describe the enlarged timed automaton $\mathcal{A}_\delta$.

Let $\delta > 0$ be a strictly positive real and $\mathcal{A}$ be a timed automaton with guards of the form $a \leq x \leq b$. $\mathcal{A}_\delta$ is the same timed automaton as $\mathcal{A}$ but each guard $a \leq x \leq b$ of $\mathcal{A}$ is transformed into a guard $a - \delta \leq x \leq b + \delta$.

Robust model-checking problem. The *robust model-checking problem* asks the following question: given a linear-time property φ and a timed automata $\mathcal{A}$, the robust model-checking problem decides whether there exists a $\delta > 0$ such that all runs of $\mathcal{A}_{\delta'}$ satisfy φ, for $\delta' \in [0, \delta]$. If such δ exists, the automaton $\mathcal{A}$ is then δ-robust.

Results. This model was used in [BMR06], [DDM+08], [BMR08] and [BMS11] for timed automata where, for each cycles, each clock is reset at least once. The results were the following ones:

- [BMR06] proved the **PSPACE-completeness** of the robust model-checking problem for **LTL** and **Büchi** properties.
- [DDM+08] proved that the robust model-checking problem for safety properties is **PSPACE-complete**.
- [BMR08] studied this problem for **coFlatMTL** and proved it is **EXPSPACE-complete**.
- [BMS11] studied this problem for ω-regular properties and proved it was **PSPACE-complete**.

Shrinking timed automata

Shrinking timed automata. The *shrinking* approach was first proposed in [SBM11] and is opposed to **guard enlargement**. Instead of verifying if the property φ still holds when the model is enlarged, the goal of the **shrinking** approach is to verify the **idealized** model despite timing perturbations. To do so, every constraints $l_x \leq x \leq u_x$ is shrunk by δ. The resulting constraint is $l_x + \delta \leq x \leq u_x - \delta$. Then, if the timing imprecision is Δ on a valuation v, the goal is the following ones:

Considering a timing perturbation Δ and a valuation v, if any perturbed valuation $v + t$, where $-\Delta \leq t \leq \Delta$, satisfies a shrunk constraint $l_x + \delta \leq x \leq u_x - \delta$, then the non-perturbed valuation v satisfies the constraint $l_x \leq x \leq u_x$. To do so, it is sufficient to verify that $[l_x + \delta + \Delta, u_x - \delta - \Delta] \subseteq [l_x, u_x]$.

More formally, given a timed automaton $\mathcal{A}$, its set of constraints $\mathcal{I}$, a positive integer vector $\boldsymbol{k} = (k_i)_{i \in \mathcal{I}}$ and a parameter $\delta > 0$, the shrunk timed automaton $\mathcal{A}_{-\boldsymbol{k}\delta}$ is the resulting timed automaton where all its constraints correspond to the shrunk constraint $l_x \leq x \leq u_x$, by $k_i\delta$, of the timed automaton $\mathcal{A}$. The goal is that the satisfaction of the shrunk constraints despite perturbation implies the satisfaction of the constraints in the abstract and idealised models. However, the shrinking can suppress some behaviours and even **block** the timed automaton, *i.e.* we find a location in the timed automaton where no transitions are enabled, even after time elapsing.

Shrinkability problems. [SBM11] proposed two types of robustness problems for the shrinking model: *shrinkability* and *non-blocking-shrinkability*. The second problem is a relaxed version of the **shrinkability** problem, where the second condition is not required.

Definition 3.1: Shrinkability

Let $\mathcal{A}$ be a timed automaton. The *shrinkability* problem asks whether there exists a strictly positive $\delta > 0$ and a positive integer vector $\boldsymbol{k}$ such that, for any $\delta' \in [0, \delta]$ the two following properties holds:

1. $\mathcal{A}_{-\boldsymbol{k}\delta'}$ is non-blocking
2. $\mathcal{A}_{-\boldsymbol{k}\delta'}$ time-abstract simulates $\mathcal{A}$.

Results. In [SBM11] and [SBM14], the **non-blocking-shrinkability** problem is proved to be in **PSPACE** and even in **NP** if, from each location, we can bound the number of outgoing locations. The **shrinkability** problem was proven to be in **EXPTIME**. A tool is also available in `http://www.lsv.fr/Software/shrinktech` and presented in [San13].

These two approaches model the robustness by modifying constraints, either by enlarging or shrinking them. Another approach consists in modelling the perturbations on the delays. The following section develops two approaches on delay perturbations.

3.5 Perturbations on delays

In timed automata, delays are proposed to pass a transition. The delay δ proposed must verify the guard g of the transition, such that, given the current valuation v, $v + \delta \models g$. In the classic semantics, the applied delays are supposed to be infinitely precise and the controller is supposed to react instantly. In reality, there might be imprecision on the delays and the controller may have a delayed reaction time. We present in this section several models that perturb the delays:

- The first one, called *almost ASAP semantics* models the time reaction of the controller with a delay $\Delta > 0$. Δ correspond of the time reaction the controller has before proposing a delay to pass the transition.
- The second one models a turn-based game with two players, a **controller** and the **environment**. The delay d proposed by the controller is perturbed by the environment.
- The third one is also based on a turn-based game. In this semantics, called *permissive semantics*, the controller **chooses the degree of perturbations** by proposing an

interval of delays, instead of a single delay.

Almost ASAP semantics

The Almost-ASAP semantics was proposed by [DDR04] and [DDR05] in order to make timed models implementable. In the abstract model, the controller is supposed to react and to change its strategy instantaneously. In reality, it has a reaction time which [DDR04] models by a delay Δ.

They propose a semantics, called the almost-ASAP semantics. Given a parameter Δ, called a reaction delay, the controller reacts with an offset $\delta \in [0, \Delta]$. In [DDR04] and [DDR05], they propose an algorithm which computes the maximum possible delay Δ, such that it is possible to check the correction of the hybrid systems. [DDM$^+$04] extended this work by proving the decidability of the following problem: given an automaton and a property, does such a reaction delay Δ exist, while verifying the automaton's correctness?

Fixed delay perturbations

Conservative game semantics and excess game semantics. The first model, proposed by [CHP11], is a parametrized model. The delay imprecision is modelled by a turn-based game. In this model, the controller chooses a delay that is perturbed by the environment. Therefore the controller must choose a delay that satisfies the guard even if the delay is perturbed. To model it, the maximal perturbation is fixed by a parameter $\delta > 0$. The **controller** chooses an edge and a delay d such that the guard of the selected edge is satisfied by any delay in $[d - \delta, d + \delta]$. The delay that is applied to pass the transition is chosen by the **environment** in the interval $[d - \delta, d + \delta]$. This semantics is called the *conservative game semantics*.

Another semantics was proposed by [BMS12], called *excess game semantics*. In this semantics, the controller does not have to ensure that any delay in $[d - \delta, d + \delta]$ satisfy the guard. The controller only have to ensure that the chosen delay d does satisfy the guard.

Parametrized robust controller synthesis problem. The *Parametrized robust controller synthesis problem* can be defined for a property φ, and both of the semantics defined in the previous paragraph, in Definition 3.2. We apply this definition for timed automata, but also for timed games.

Definition 3.2: Parametrized robust controller synthesis problem

Given a property φ, the parametrized robust controller synthesis problem asks whether there exists $\delta > 0$ such that the controller has a winning strategy such that every outcome satisfies the property φ.

We can also define a robust controller synthesis problem for *weighted timed automata*. Weighted timed automata are an extension of timed automata defined in [ATP01] and [BFH+01]. In this extension, each location is associated to a cost and the global cost function represents the sum of the time spent in each location, weighted by their associated cost function. The interest on these objects is the *cost-optimal reachability* (reaching the goal while minimising the global cost function). In the two semantics defined previously, the global cost function depends on the perturbation δ, [BMS13] defined *optimal limit-cost decision problem* and *optimal limit-cost strong decision problem* as follows:

Definition 3.3: Optimal limit-cost (*resp.* strong) decision problem

Given $p \geq 0$, the optimal limit-cost (*resp.* strong) decision problem asks whether there exists a winning strategy of the controller such that the limits when δ tends to 0 of the global cost function is greater than (*resp.* strictly greater than) p.

Results. The **parametrized robust controller synthesis problem** has been studied for timed games and timed automata for both semantics. For the *conservative game semantics*, this problem has been solved in **exponential time** for timed games and **parity conditions** by [CHP11]. [SBM+13] proved that this problem is in **PSPACE** for timed automata and **Büchi conditions**. [BMR+19] extended [SBM+13]'s work by solving the same problem for a given 'lasso' in polynomial time and showing that computing the largest controllable perturbation is decidable. They provide a C++ implementation.

For the *excess timed semantics*, this problem has proven to be **EXPTIME-complete** for turn-based timed games and **reachability** properties by [BMS12] and [BMS15].

The **optimal limit-cost (*resp.* strong) decision problem** has been studied for both semantics too for weighted timed automata. It is proved to be **PSPACE-complete** for the **conservative game semantics**, but undecidable for weighted timed games by [BMS13]. It is also proved undecidable for weighted timed automata for the **excess game semantics** by [BMS13],

Finally, [ORS14] provided results for a slightly changed semantics. In their model the controller chooses the action a (and not an edge) and a delay d. The delay applied to

pass the transition is chosen by the environment in $[d-\delta, d+\delta]$, but the environment also chooses the a-labelled edge that is taken. Their contribution is to prove that, given a timed automaton and a **Büchi objective**, deciding whether there exists such $\delta > 0$ such that the controller has a winning strategy is an **EXPTIME-complete** problem. They provide a result for probabilistic timed automata, where the environment choices are randomised, that we detail in Subsection 3.5.

Robustness in probabilistic timed automata

Stochastic game and Markov decision process semantics. Probabilistic timed automata were studied in [ORS14] on the same model detailed in Section 3.5. [ORS14] proposed **two** probabilistic models. In the first one, the *stochastic game semantics*, the environment does not choose the applied delay anymore. Each delay is chosen randomly with the uniform distribution. In the second one, the *Markov decision process semantics*, the delays are chosen randomly with the uniform distribution too, but the edges are also chosen randomly with a uniform distribution. As a result, there is only one player, the **controller**.

Results. With both semantics, [ORS14] proved that, for **Büchi objective**, deciding, almost-surely, whether there exists such $\delta > 0$ such that the controller has a winning strategy is an **EXPTIME** problem.

Maximally-permissive strategy

Permissive semantics. Finally, delay perturbations can be modelled as a turn-based game where the controller chooses, for each location, the amount of allowed perturbation. This model was introduced by [BFM15] and is detailed in Section 2.2 and is called the *permissive semantics*. The model is a turn-based game where the first player, the **controller**, chooses an **interval of delays** I and an action a, such that there exists an transition t such that every delay $\delta \in I$ satisfies the guard of t. The **environment** then chooses a delay in the interval I.

An objective function, called the *penalty* function, sums the inverse of the size of all proposed intervals. The **controller**'s objective is to minimise this function. The **environment**'s objective is to maximise it.

Most-permissive strategy problem and results. The *most-permissive strategy problem*, defined in [BFM15], asks, given a timed automaton $\mathcal{A}$ and a configuration s, to compute the penalty of the configuration s. This problem was proven to be in **PTIME** for single-clock timed automata. This approach is extended in this thesis for multiple-clock timed automata.

Conclusion

Several approaches have been studied to model the robustness of timed automata. The first reason is that the objective may be to make the implementation robust to perturbations, or to force the implementation to respect the original idealised model despite perturbations in the environment. The second reason is that perturbations do not occur in only one way. As with the smartphone robustness example, where robustness can refer to resistance to water, shock, deformation or heat, robustness can for instance refers to perturbations on the timing of clocks (clock drifting), guards (enlargement, shrinking), or proposed delays.

The approach we focus on is the perturbation on delays, as proposed in [BFM15]. In this thesis, we strive to propose models and algorithms to verify that the timed automaton is robust to delay perturbations. Our goal is to compute, symbolically, the largest perturbation that can occur while achieving reachability objectives. We slightly changed the model of [BFM15] and presented it in the previous chapter, in Section 2.2.

The problem we study in this thesis can be seen as an optimisation problem on a turn-based game: finding the optimal strategy of two players, where one wants to minimise and the other one wants to maximise the permissive function of all possible runs. The permissiveness function has been defined in Definition 2.21. Nevertheless, it is not defined iteratively, which would be helpful in order to find the optimal strategies. To solve this problem, in the next Chapter 4 we define a sequence that approximates and tends to our permissiveness function, while being defined iteratively.

Chapter 4

The sequence of suboptimal permissive functions

The permissiveness function, as defined in Definition 2.21, considers all intervals proposed during a run at the same time. In the first examples of computation of permissiveness functions in Section 2.2.2, the permissiveness of successors is used to compute the permissiveness of a location. This gave us a first intuition of how to compute the permissiveness function. In order to formalise this intuition, we present in this chapter the sequence of suboptimal permissive functions. This sequence computes the minimum between the permissiveness of the current p-move and the permissiveness of the successor, considering that the opponent has an optimal strategy. It is built as a backward algorithm. This sequence of functions is related to the permissiveness function, and we show in this chapter that it converges to the permissiveness function and has interesting properties. This chapter is organised as follows:

- First, we present the formal definition of the sequence of suboptimal permissive functions in Section 4.1 and its links with the game semantics presented in the permissiveness semantics of Section 2.2.1.
- Secondly, we present the properties of the sequence of suboptimal permissive functions in Section 4.2: its evolution, its link with the permissiveness function, its analytic properties and finally its optimisation properties.

4.1 Definition

The sequence of suboptimal permissive functions, denoted $(\mathcal{P}_i)_{i\in\mathbb{N}}$, depends on the possible successors of the current configuration. To define this sequence formally, let us first introduce a notation for the successor of a configuration. Let (ℓ, v) be an arbitrary configuration such that $\mathsf{p\text{-}moves}\,(\ell, v) \neq \varnothing$. Let $(I, a) \in \mathsf{p\text{-}moves}\,(\ell, v)$ be an arbitrary p-move

and $\delta \in I$. There exists a configuration (ℓ', v') such that $(\ell, v) \xrightarrow{\delta, a} (\ell', v')$. We denote this successor $\text{succ}(\ell, v, \delta, a) := (\ell', v')$. Let us now define the sequence of suboptimal permissive functions in Definition 4.1.

Definition 4.1: The sequence of suboptimal permissive functions

Let $\mathcal{A}$ be an arbitrary timed automaton with n clocks and ℓ be an arbitrary location. The *sequence of suboptimal permissive functions* is the sequence of functions, denoted $(\mathcal{P}_i(\ell, \cdot))_i$, defined as follows:

- Initialisation: for any valuation v:

$$\mathcal{P}_0(\ell, v) = \begin{cases} +\infty & \text{if } \ell \in Q_f \\ -\infty & \text{otherwise} \end{cases}$$

- Step $i+1$: let v be an arbitrary valuation. If $\ell \in Q_f$, $\mathcal{P}_{i+1}(\ell, v) = +\infty$, otherwise:

$$\mathcal{P}_{i+1}(\ell, v) = \begin{cases} \sup\limits_{(I,a) \in \text{p-moves}(\ell, v)} \min\left(|I|, \inf\limits_{\delta \in I} \mathcal{P}_i(\text{succ}(\ell, v, \delta, a))\right) \\ \qquad \text{if } \text{p-moves}(\ell, v) \neq \varnothing \\ -\infty \qquad \text{otherwise} \end{cases}$$

The sequence of suboptimal permissive functions considers the optimal strategy of the player and the opponent, from the game described in the permissive semantics. In this semantics, the player chooses a p-move (I, a) and the opponent chooses a delay $\delta \in I$ to pass the transition and reach a successor (ℓ', v'):

- $\inf\limits_{\delta \in I}(\mathcal{P}_i(\text{succ}(\ell, v, \delta, a)))$ is the permissiveness at $(\ell', v + \delta[\mathcal{C}_r \leftarrow 0])$ if the opponent chooses the delay $\delta \in I$ that **minimises** the permissiveness.

- Considering this optimal delay for each interval the player can propose, the player chooses a p-move that maximises the minimum between the current interval the player proposes and the permissiveness of the successors that will be reached. If an optimal p-move (I^*, a^*) exists, then:

$$\mathcal{P}_{i+1}(\ell, v) = \min\left(|I^*|, \inf_{\delta \in I^*} \mathcal{P}_i(\text{succ}(\ell, v, \delta, a^*))\right)$$

In Figure 4.1, we describe the possible choices of delays and intervals for the guard $0 \leq x \leq 1$ of the timed automaton in Figure 2.2. The player has an infinite number of choices of intervals, for instance $[0, 0.1]$ or $[0.9, 1]$. Each choice will give an interval I such that the opponent can choose a delay $\delta \in I$. For instance if $I = [0, 0.1]$, the opponent can choose any delay between 0 and 0.1 (for instance 0 or 0.1). Each delay corresponds to a future configuration: for instance $(\ell_1, v + 0\,[\mathcal{C}_r \leftarrow 0])$ or $(\ell_1, v + 0.1\,[\mathcal{C}_r \leftarrow 0])$.

Our goal is to reduce the number of possible delays and intervals to a finite set of possibilities, in which an infimum and a supremum can be found (they are in general not unique).

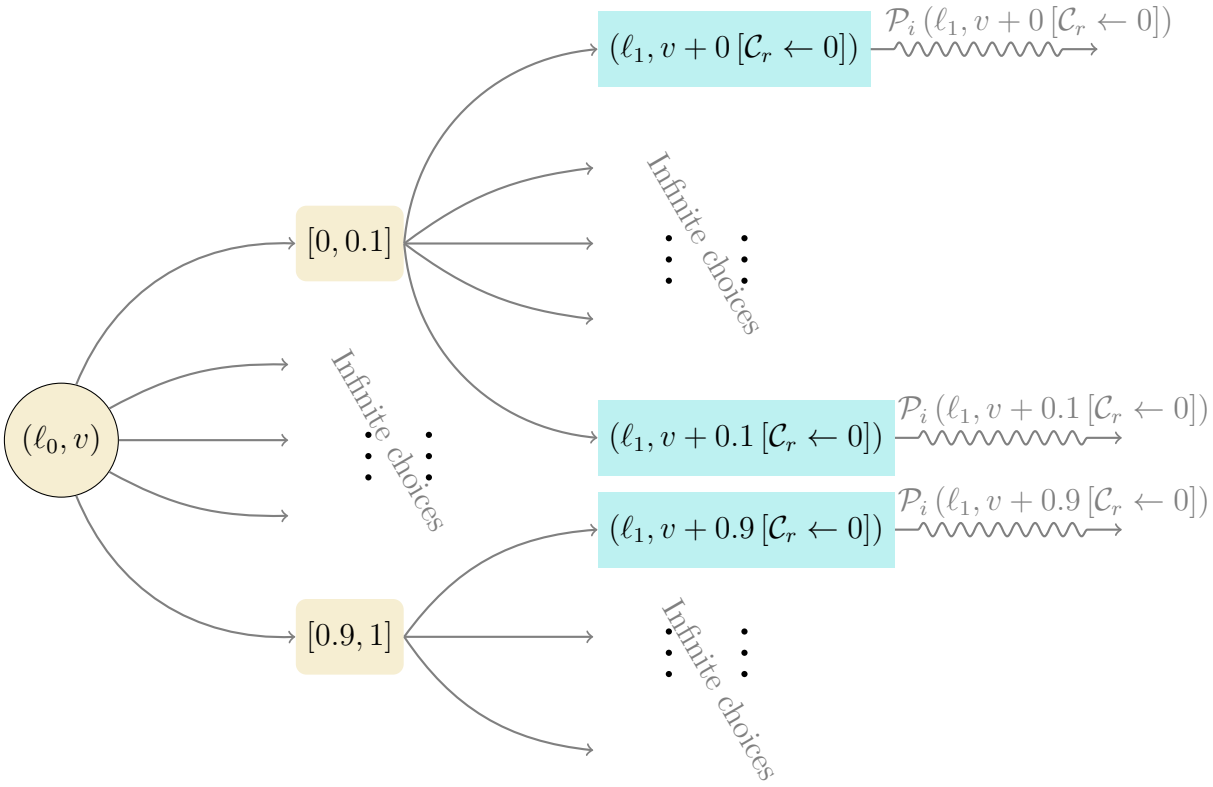


Figure 4.1: The possible choices of intervals and delays with the guard $0 \leq x \leq 1$ when $v = (0, 0)$.

Let us construct the sequence of suboptimal permissive functions on an example studied in Chapter 2.

Example 4.1.1 *Let us consider the first example of timed automaton of Subsection 2.2.2, described in Figure 2.2. Let us recall that the permissiveness function of this timed auto-*

maton is:

$$\text{Perm}(\ell_j, \cdot): \quad (x,y) \quad \mapsto \quad \begin{cases} +\infty & \text{if } \ell_j = \ell_f \\ \min(\frac{1-x}{2-j}, \frac{1-y}{2-j}) & \text{if } (x,y) \in \mathit{Win}_{\ell_j} \wedge j \in \{0,1\} \\ -\infty & \text{otherwise} \end{cases}$$

We describe the sequence of suboptimal permissive functions of this timed automaton for $i = 0, 1$ *and* 2 *in Tables 4.1, 4.2 and 4.3 respectively.*

Location	sequence of suboptimal permissive functions
ℓ_0	$\mathcal{P}_0(\ell_0) = -\infty$
ℓ_1	$\mathcal{P}_0(\ell_1) = -\infty$
ℓ_f	$\mathcal{P}_0(\ell_1) = +\infty$

Table 4.1: Computation of $\mathcal{P}_0$ for the timed automaton of Figure 2.2.

Location	sequence of suboptimal permissive functions
ℓ_0	$\mathcal{P}_1(\ell_0,(x,y)) = -\infty$
ℓ_1	$\mathcal{P}_1(\ell_1,(x,y)) = \begin{cases} \min(1-x, 1-y) & \text{if } x,y \in [0,1] \\ -\infty & \text{otherwise} \end{cases}$
ℓ_f	$\mathcal{P}_1(\ell_f,(x,y)) = +\infty$

Table 4.2: Computation of $\mathcal{P}_1$ for the timed automaton of Figure 2.2.

Location	sequence of suboptimal permissive function
ℓ_0	$\mathcal{P}_2(\ell_0,(x,y)) = \begin{cases} \dfrac{\min(1-x, 1-y)}{2} & \text{if } x,y \in [0,1] \\ -\infty & \text{otherwise} \end{cases}$
ℓ_1	$\mathcal{P}_2(\ell_1,(x,y)) = \begin{cases} \min(1-x, 1-y) & \text{if } x,y \in [0,1] \\ -\infty & \text{otherwise} \end{cases}$
ℓ_f	$\mathcal{P}_2(\ell_f,(x,y)) = +\infty$

Table 4.3: Computation of $\mathcal{P}_2$ for the timed automaton of Figure 2.2.

In Example 4.1.1, the sequence of suboptimal permissive functions is eventually constant from rank 2. Lemma 4.5, in Section 4.2, gives a sufficient rank from which the sequence is eventually constant, for acyclic timed automata. As we also prove that the

sequence of suboptimal permissive functions converges to the permissiveness function, a direct consequence is that the sequence of suboptimal permissive functions converges in a finite number of steps to the permissiveness function, its fixed point, for acyclic timed automata. It allows us to reduce the maximal-permissiveness problem to (for a sufficiently high rank i) the sequence of suboptimal permissive functions problem, that we define in the following definition.

Definition 4.2: Sequence of suboptimal permissive functions problem

Given a timed automaton $\mathcal{A}$, an integer $i \in \mathbb{N}$, an initial configuration (ℓ_0, v_0) and a goal location ℓ_f, the sequence of suboptimal permissive functions problem asks to compute $\mathcal{P}_i(\ell_0, v_0)$.

4.2 Properties of the sequence of suboptimal permissive functions

In this section, we prove some basic properties of this sequence of functions, and in particular its links with the permissiveness function. In Subsection 4.2.1, we study the evolution of the sequence of suboptimal permissive functions. In Subsection 4.2.2, we prove the link between the sequence of suboptimal permissive functions $(\mathcal{P}_i)_{i\in\mathbb{N}}$ and the permissiveness function Perm. For instance, we give sufficient conditions where the limits of the sequence equals the permissiveness function. In Subsection 4.2.3, we prove analytic results such as continuity. Finally, in Subsection 4.2.4, we study optimisation results, such as concavity.

4.2.1 Evolution properties

The goal of this subsection is to study the evolution of the sequence of suboptimal permissive functions, with respect to either the rank i (in Lemma 4.3 and 4.5) or the valuation of the considered configuration in Lemma 4.6. In the first lemma, we state that the sequence is non-decreasing. We prove this property by **induction**.

Lemma 4.3

For any configuration (ℓ, v), the sequence $(\mathcal{P}_i(\ell, v))_{i\geq 0}$ is non-decreasing with respect to i.

Proof of Lemma 4.3. For any configuration (ℓ, v), by Definition 4.1 either $\mathcal{P}_0(\ell, v) = \mathcal{P}_1(\ell, v) = +\infty$ (if $\ell \in Q_f$), or $\mathcal{P}_0(\ell, v) = -\infty$. In both cases, $\mathcal{P}_0(\ell, v) \leq \mathcal{P}_1(\ell, v)$. Then, let $i \in \mathbb{N}$ and let us assume that $\mathcal{P}_i(\ell, v) \leq \mathcal{P}_{i+1}(\ell, v)$ for all (ℓ, v). Let us prove that for all (ℓ, v), $\mathcal{P}_{i+1}(\ell, v) \leq \mathcal{P}_{i+2}(\ell, v)$:

- Suppose that $\mathsf{p\text{-}moves}(\ell, v) = \varnothing$. Then by definition, for any $j \geq 0$ and any configuration (ℓ, v), $\mathcal{P}_j(\ell, v) = -\infty$. As a result, $\mathcal{P}_{i+1}(\ell, v) \leq \mathcal{P}_{i+2}(\ell, v)$.
- On the contrary, suppose that $\mathsf{p\text{-}moves}(\ell, v) \neq \varnothing$, then for **all** p-moves (I, a):

$$\inf_{\delta \in I} \left(\mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right) \leq \inf_{\delta \in I} \left(\mathcal{P}_{i+1}\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right)$$

  Then, the following inequality holds:

$$\min\left(|I|, \inf_{\delta \in I} \left(\mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right)\right) \leq \min\left(|I|, \inf_{\delta \in I} \left(\mathcal{P}_{i+1}\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right)\right)$$

  For all set of p-moves, in particular $\mathsf{p\text{-}moves}(\ell, v)$, the supremum operator preserves the inequality:

$$\begin{aligned}\mathcal{P}_{i+1}(\ell, v) &= \sup_{(I,a) \in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|I|, \inf_{\delta \in I} \mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right) \\ &\leq \sup_{(I,a) \in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|I|, \inf_{\delta \in I} \mathcal{P}_{i+1}\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right)\end{aligned}$$

  The result follows by definition of $\mathcal{P}_i(\ell, v)$ when $\mathsf{p\text{-}moves}(\ell, v) \neq \varnothing$.

□

Example 4.2.1 *Let us consider the timed automaton used in Example 4.1.1 and the configuration (ℓ_0, v). The sequence $(\mathcal{P}_i(\ell_0, v))_{i \geq 0}$ is a non-decreasing sequence for the first three indices, as it is $-\infty$ for $i = 0, 1$.*

If we consider ℓ_1, the sequence $(\mathcal{P}_i(\ell_1, v))_{i \geq 0}$ is $-\infty$ for $i = 0$, and then it is constant for $i = 1, 2$.

We now identify the conditions and ranks from which the sequence of functions $(\mathcal{P}_i(\ell, v))_{i \geq 0}$ is *eventually constant*. Let us first recall the definition of such sequence.

Definition 4.4: eventually constant sequence

A sequence $(u_n)_{n \geq 0}$ is eventually constant if there exists a rank $n_0 \geq 0$ such that for all $n \geq n_0, u_n = u_{n_0}$.

The following Lemma 4.5 states that, for a fixed configuration (ℓ, v), the sequence of suboptimal permissive functions $\mathcal{P}_i(\ell, v)$ is eventually constant at a rank k that represents the length of the longest path to a goal of the timed automaton. More formally, let ℓ and ℓ' be two arbitrary locations. We denote $\Pi(\ell, \ell')$ the set of paths from ℓ to ℓ'. For any path $\pi \in \Pi(\ell, \ell')$, we denote $|\pi|$ its number of transitions. We define the *maximal distance of ℓ to goals*, denoted d_ℓ, as follows:

$$\mathrm{d}_\ell := \max_{\ell_f \in Q_f} \max_{\pi \in \Pi(\ell, \ell_f)} (|\pi|)$$

Lemma 4.5

Let ℓ be an arbitrary location. Then $(\mathcal{P}_i(\ell, v))_{i \geq 0}$ is constant from rank d_ℓ (for any valuation v), *i.e.*, for any valuation v:

$$\forall j \geq 0, \mathcal{P}_{\mathrm{d}_\ell + j}(\ell, v) = \mathcal{P}_{\mathrm{d}_\ell}(\ell, v)$$

Proof of Lemma 4.5. We proceed with a simple induction on d_ℓ.

For $\mathrm{d}_\ell = 0$, only the locations that belong to Q_f satisfy the condition, and the result holds by definition of $\mathcal{P}_j$ for any $\ell_f \in Q_f$.

Now, let us assume that the result holds for some index i, and consider a location ℓ such that $\mathrm{d}_\ell \leq i + 1$ and a valuation v. If $\mathsf{p\text{-}moves}(\ell, v) = \varnothing$, the sequence $(\mathcal{P}_i(\ell, v))_{i \in \mathbb{N}}$ is constant and is $-\infty$, and the result follows immediately. Otherwise, let $(I, a) \in \mathsf{p\text{-}moves}(\ell, v)$ be an arbitrary p-move. For any $\delta \in I$, let us denote $(\ell'_\delta, v'_\delta) := \mathrm{succ}(\ell, v, \delta, a)$. The location of this successor ℓ'_δ verifies $\mathrm{d}_{\ell'_\delta} \leq i$. Hence $\mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a)) = \mathcal{P}_{i+1}(\mathrm{succ}(\ell, v, \delta, a))$. As a result:

$$\begin{aligned}
\mathcal{P}_{i+1}(\ell, v) &:= \sup_{(I,a) \in \mathsf{p\text{-}moves}(\ell, v)} \min\left(|I|, \inf_{\delta \in I} \mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a))\right) \\
&= \sup_{(I,a) \in \mathsf{p\text{-}moves}(\ell, v)} \min\left(|I|, \inf_{\delta \in I} \mathcal{P}_{i+1}(\mathrm{succ}(\ell, v, \delta, a))\right) \\
&= \mathcal{P}_{i+2}(\ell, v)
\end{aligned}$$

It immediately follows that $\mathcal{P}_{i+1}(\ell, v) = \mathcal{P}_{i+2}(\ell, v)$ for any valuation v. □

Example 4.2.2 *Let us go back to the Example 4.1.1. As the sequence of suboptimal permissive functions is always $+\infty$ on ℓ_f, if we consider the configuration (ℓ_1, x, y), let us*

remark that $d_{\ell_1} = 1$ *and that its unique successor location is* ℓ_f*. Let* $i \geq 1$ *be an index:*

$$\begin{aligned}\mathcal{P}_i(\ell_1,(x,y)) &= \sup_{(I,a)\in \textit{p-moves}(\ell_1,(x,y))} \min(|I|,+\infty)\\ &= \mathcal{P}_1(\ell_1,(x,y))\end{aligned}$$

As a result, the sequence $(\mathcal{P}_i(\ell_1,(x,y)))_{i\geq 0}$ *is constant from rank* 1*.*

Proving that the sequence of suboptimal permissive functions is eventually constant gives us an important result to compute its limit. Indeed, the sequence then converges in a finite number of steps. Let us fix a configuration (ℓ, v). In Lemma 4.5, we state that, if a rank k is greater than d_ℓ, the sequence $(\mathcal{P}_i(\ell,v))_{i\geq k}$ is constant. As this number is finite for acyclic timed automata, the sequence $(\mathcal{P}_i(\ell,v))_{i\geq 0}$ converges in a finite number of steps for **acyclic** timed automaton. In Subsection 4.2.2, we prove that the permissiveness function corresponds to the limit of the sequence of suboptimal permissive functions. This result enables us to reduce the maximal-sequence of permissiveness function (Definition 2.22) to the maximal-sequence of suboptimal permissive function problem (Definition 4.2), for acyclic timed automata and for the rank d_ℓ.

Our last result provides a constant above which the exact value of the clocks no longer affects the value of $v \mapsto \mathcal{P}_i(\ell,v)$. The intuition is that if the valuation v is greater than the greatest constant of the automaton $\mathcal{A}$, then the same p-moves can be proposed and the valuation of the successor will be greater than v (except for the clocks that are reset). This lemma is used to prove the correctness of our algorithm that computes $\mathcal{P}_i$ in Section 5.1.

Lemma 4.6

Let $\mathcal{A}$ be a timed automaton, and let $\mathcal{M}(\mathcal{A})$ be its largest constant[a]. Let ℓ be a location, and $i \in \mathbb{N}$. Take two valuations v and v' such that, for any clock x, we have either $v(x) = v'(x)$, or $v(x) > \mathcal{M}(\mathcal{A})$ and $v'(x) > \mathcal{M}(\mathcal{A})$. Then $\mathcal{P}_i(\ell,v) = \mathcal{P}_i(\ell,v')$.

[a] see Definition 2.6

Proof of Lemma 4.6. We proceed by induction on i: for $i = 0$, $v \mapsto \mathcal{P}_i(\ell,v)$ is constant with respect to the valuation, so the result holds.

Let us assume that the result holds for some index $i \in \mathbb{N}$. We now prove that it holds for $i+1$.

Let ℓ be an arbitrary location and v, v' be two valuations such that for any clock,

either $v(x) = v'(x)$, or $v(x) > \mathcal{M}(\mathcal{A})$ and $v'(x) > \mathcal{M}(\mathcal{A})$. With this hypothesis, the same p-moves can be proposed from (ℓ, v) and (ℓ, v'), as the guard constant are all lower than $\mathcal{M}(\mathcal{A})$, so $\mathsf{p\text{-}moves}(\ell, v) = \mathsf{p\text{-}moves}(\ell, v')$, then:

$$\mathcal{P}_{i+1}(\ell, v) = \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v')} \min\left(|I|, \inf_{\delta\in I} \mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a))\right)$$

Let $(I, a) \in \mathsf{p\text{-}moves}(\ell, v)$, $\mathcal{C}_r$ be the reset set of the corresponding transition and $\delta \in I$. Let us prove that $\mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a)) = \mathcal{P}_i(\mathrm{succ}(\ell, v', \delta, a))$. Let ℓ' be the successor of ℓ when choosing the p-move (I, a), then:

$$\mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a)) = \mathcal{P}_i(\ell', v + \delta[\mathcal{C}_r \leftarrow 0]) \text{ and } \mathcal{P}_i(\mathrm{succ}(\ell, v', \delta, a)) = \mathcal{P}_i(\ell', v' + \delta[\mathcal{C}_r \leftarrow 0])$$

Let x be an arbitrary clock. Then either $x \in \mathcal{C}_r$ and $v + \delta[\mathcal{C}_r \leftarrow 0](x) = 0 = v' + \delta[\mathcal{C}_r \leftarrow 0](x)$ or $x \notin \mathcal{C}_r$ and then $(v + \delta[\mathcal{C}_r \leftarrow 0])(x) > v(x) > \mathcal{M}(\mathcal{A})$ and $(v' + \delta[\mathcal{C}_r \leftarrow 0])(x) > v'(x) > \mathcal{M}(\mathcal{A})$. Since these two successors satisfy the induction hypothesis, for any delay $\delta \in I$:

$$\mathcal{P}_i(\ell', v' + \delta[\mathcal{C}_r \leftarrow 0]) = \mathcal{P}_i(\ell', v + \delta[\mathcal{C}_r \leftarrow 0])$$

As a result, $\mathcal{P}_{i+1}(\ell, v) = \mathcal{P}_{i+1}(\ell, v')$.

□

Example 4.2.3 *Let us go back to the same example, with the timed automaton of Figure 2.2 and let us consider an arbitrary location ℓ such that $\ell \neq \ell_f$. The maximal constant of this timed automaton is 1. As computed in Example 4.1.1, if any valuation v is greater than 1 on x or y, then the sequence of suboptimal permissive functions on a configuration (ℓ, v) is always equal to $-\infty$.*

4.2.2 Link between $(\mathcal{P}_i)_{i\geq 0}$ and Perm

In this subsection, we state the links between the sequence of suboptimal permissive functions $(\mathcal{P}_i)_{i\in\mathbb{N}}$ and the permissiveness function, Perm. Intuitively, the sequence $(\mathcal{P}_i)_{i\geq 0}$ is sequence of functions that are always smaller than the permissiveness function, and its limit, if it exists, corresponds to the permissiveness of the timed automaton.

Proposition 4.7

For any $i \in \mathbb{N}$ and for any configuration (ℓ, v), it holds:

1. $\mathcal{P}_i(\ell, v) = -\infty$ if, and only if, there is no run of length at most i from (ℓ, v) to any location $\ell_f \in Q_f$;
2. for any $p \in \mathbb{R}_+$, and any $i \in \mathbb{N}$, it holds that $\mathcal{P}_i(\ell, v) \geq p$ if, and only if, there is a permissive strategy with permissiveness larger than (or equal to) p that is winning from (ℓ, v) within i steps.

Proof of Proposition 4.7. We will prove both claims by induction on i.

1. The result is trivial for $i = 0$: $\mathcal{P}_i(\ell, v) = -\infty$ if and only if $\ell \notin Q_f$. This is equivalent to having no runs of length 0 from ℓ to any goal location $\ell_f \in Q_f$. Indeed, such locations are the ones that already belong to Q_f

   Suppose now that the result holds for some index $i \in \mathbb{N}$. The (equivalence) proof is carried by double implication:

   ($\Rightarrow$) Suppose that $\mathcal{P}_{i+1}(\ell, v) = -\infty$, then either $\mathsf{p\text{-}moves}(\ell, v)$ is empty (and then the result immediately follows). or it is not empty and the following equality holds:

   $$\sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|I|, \inf_{\delta\in I} \mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a))\right) = -\infty$$

   Let $(I, a) \in \mathsf{p\text{-}moves}(\ell, v)$. As, for each $\delta \in I$, $(\{\delta\}, a)$ is also a (punctual) enabled p-move, then:

   $$\{(\{\delta\}, a) \mid (\{\delta\}, a) \text{ is an enabled p-move}\} \subseteq \mathsf{p\text{-}moves}(\ell, v)$$

   As a result:

   $$\begin{aligned}
   \mathcal{P}_{i+1}(\ell, v) &= \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|I|, \inf_{\delta\in I} \mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a))\right) \\
   &\geq \sup_{(\{\delta\},a)\in \mathsf{p\text{-}moves}(\ell,v)} \left(\min\left(|\{\delta\}|, \inf_{\delta\in\{\delta\}} \mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a))\right)\right) \\
   &= \sup_{(\{\delta\},a)\in \mathsf{p\text{-}moves}(\ell,v)} (\min(0, \mathcal{P}_i(\mathrm{succ}(\ell, v, \delta, a))))
   \end{aligned}$$

As $\mathcal{P}_{i+1}(\ell, v) = -\infty$:

$$-\infty \geq \sup_{(\{\delta\},a)\in \mathsf{p\text{-}moves}(\ell,v)} \left(\min\left(0, \mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right)\right)$$

Then:

$$\sup_{(\{\delta\},a)\in \mathsf{p\text{-}moves}(\ell,v)} \mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right) = -\infty$$

Then for each punctual p-move $(\{\delta, \}\, a) \in \mathsf{p\text{-}moves}\,(\ell, v)$, the successor (ℓ', v') such that $\ell, v \xrightarrow{\delta,a} \ell', v'$ is such that $\mathcal{P}_i(\ell', v') = -\infty$. From the induction hypothesis, for any goal location $\ell_f \in Q_f$, there can be no path from those (ℓ', v') to ℓ_f within at most i steps. Hence, there are no paths from (ℓ, v) to ℓ_f within at most $i + 1$ steps.

($\Leftarrow$) Conversely, suppose that for each goal location $\ell_f \in Q_f$, there are no path having at most $i+1$ steps from (ℓ, v) to ℓ_f, then either this is because $\mathsf{p\text{-}moves}\,(\ell, v) = \varnothing$, or this is because whatever enabled p-move (I, a) and $\delta \in I$, there is no path of length at most i from $\mathrm{succ}\,(\ell, v, \delta, a)$ to ℓ_f. By induction hypothesis, $\mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right) = -\infty$, hence:

$$\begin{aligned}
\mathcal{P}_{i+1}(\ell, v) &= \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|I|, \inf_{\delta\in I} \mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right)\right) \\
\mathcal{P}_{i+1}(\ell, v) &= \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|I|, -\infty\right) \\
\mathcal{P}_{i+1}(\ell, v) &= -\infty
\end{aligned}$$

2. Let $p \in \mathbb{R}_+$. The base case is again trivial. $\mathcal{P}_0(\ell, v) \geq p$ if and only if $\mathcal{P}_0(\ell, v) = +\infty$. This is equivalent to $\ell \in Q_f$. As a result, the equivalence is immediate. The equivalence is also trivial if $\ell \in Q_f$. Suppose now that $\ell \notin Q_f$ and that the result holds to some index i. The (equivalence) proof is again carried by double implication:

   ($\Rightarrow$) Suppose that $\mathcal{P}_{i+1}(\ell, v) \geq p$. Then $\mathsf{p\text{-}moves}\,(\ell, v) \neq \varnothing$, there exists a enabled move (I, a) such that $|I| \geq p$ and $\mathcal{P}_i\left(\mathrm{succ}\left(\ell, v, \delta, a\right)\right) \geq p$ for any $\delta \in I$. Applying the induction hypothesis, there is an i-step winning strategy with permissiveness larger than, or equals to, p from each successor configuration $\mathrm{succ}\,(\ell, v, \delta, a)$. By concatenating with the p-move (I, a), there exists an $i + 1$-step winning strategy with permissiveness larger, or equals to, than p from (ℓ, v).

($\Leftarrow$) Let us pick an $i+1$-step winning strategy σ_p from (ℓ, v) with permissiveness larger, or equals to, than p. Write $\sigma_p(\ell, v) = (I_0, a_0)$. Then, for any $\delta \in I_0$ and location (ℓ', v') such that $(\ell, v) \xrightarrow{\delta, a_0} (\ell', v')$, strategy σ_p is an i-step winning strategy with permissiveness larger than, or equals to, p, so that, following the induction hypothesis, the $\mathcal{P}_i(\ell', v') \geq p$. As $|I_0| > p$, it immediately follows that:

$$\mathcal{P}_{i+1}(\ell, v) \geq p$$

□

We can use this result to prove that the pointwise limit of the sequence of suboptimal permissive functions tends to the permissiveness function.

Corollary 4.8

Let $\ell \in Q$ be a location. If $\lim_{i \to +\infty} \Big(v \mapsto \mathcal{P}_i(\ell, v) \Big)$ exists, then $v \mapsto \mathrm{Perm}(\ell, v)$ is the pointwise limit of $(\mathcal{P}_i(\ell, \cdot))_{i \geq 0}$.

Proof of Corollary 4.8. This is a basic result of Lemma 4.3 and Proposition 4.7. Let (ℓ, v) be an arbitrary configuration and suppose that its limit $\lim_{i \to +\infty} \mathcal{P}_i(\ell, v)$ exists. There are three cases:

- The case where $\lim_{i \to +\infty} \mathcal{P}_i(\ell, v) = -\infty$ is trivial by the first claim of Proposition 4.7.
- The case where $\lim_{i \to +\infty} \mathcal{P}_i(\ell, v) = +\infty$ is trivial because by proposition 4.7, for any $p \in \mathbb{R}_+$, there exists a permissive strategy σ such that $\mathrm{Perm}_\sigma(\ell, v) \geq p$. So $\mathrm{Perm}(\ell, v) = +\infty$.
- Otherwise, suppose that there exists $p \in \mathbb{R}_+$ such that $\lim_{i \to +\infty} \mathcal{P}_i(\ell, v) = p$, then by proposition 4.7 there exists a permissive strategy σ such that $\mathrm{Perm}_\sigma(\ell, v) \geq p$. Suppose that there exists another winning strategy σ' such that $\mathrm{Perm}_\sigma(\ell, v) \geq p' > p$ in i steps. Then by Corollary 4.7, $\mathcal{P}_i(\ell, v) > p$ too. By Lemma 4.3, the sequence is non-decreasing so $p = \lim_{i \to +\infty} \mathcal{P}_i(\ell, v) \geq \mathcal{P}_i(\ell, v) > p$, which is absurd. As a result, $\sup_\sigma \mathrm{Perm}_\sigma(\ell, v) = p$ so $\mathrm{Perm}(\ell, v) = p$.

□

It enables us to prove some analytic properties of Perm in the next section.

Corollary 4.9

Let $\mathcal{A}$ be an **acyclic** timed automaton, $\ell \in Q$ a location and $i \geq \mathrm{d}_\ell$. The two functions $v \mapsto \mathrm{Perm}(\ell, v)$ and $v \mapsto \mathcal{P}_i(\ell, v)$ are equal.

Proof of Corollary 4.9. Because of Lemma 4.5. For each location $\ell \in Q$, the sequence $(\mathcal{P}_i(\ell, v))_{i \geq 0}$ is eventually constant from rank d_ℓ. As a result, its limits exists and is reaches from rank d_ℓ. Let $i \geq \mathrm{d}_\ell$, by applying the Corollary 4.8, the functions $v \mapsto \mathrm{Perm}(\ell, v)$ and $v \mapsto \mathcal{P}_i(\ell, v)$ are equal. □

4.2.3 Analytic properties

The goal of this section is to state basic regularity properties of $\mathcal{P}_i$ and Perm. The strongest property we were able to prove for $\mathcal{P}_i$ and Perm is that they are 2-Lipschitz.

We recall some basic definitions:

Definition 4.10: Lipschitz continuous functions

Let $k > 0$ be a strictly positive constant and $E \subseteq \mathbb{R}^n$ be a subset of $(\mathbb{R}^n, \|\cdot\|)$. A function $f : E \mapsto \mathbb{R}$ is said to be k-Lipschitz (or k-Lipschitz continuous) if the following property holds:

$$\forall x, y \in E, |f(x) - f(y)| \leq k \cdot \|x - y\|_E$$

Let us recall that, if $F = \mathbb{R}_+$, **the k-Lipschitz continuity is stable with the operator** max. Indeed let us consider a strictly positive constant $k > 0$, an integer $m \in \mathbb{N}$ and a sequence of functions $(f_i)_{0 \leq i \leq m}$ defined from E to $\mathbb{R}_+$, such that for any $i \in [\![0, m]\!]$, f_i is k-lipschitz. Let us consider $x, y \in E$ and let us denote i and j the index of $[\![0, m]\!]$ such that $\max_{0 \leq l \leq m} f_l(x) = f_i(x)$ and $\max_{0 \leq l \leq m} f_l(y) = f_j(y)$. By definition of the maximum:

$$f_j(x) - f_j(y) \leq f_i(x) - f_j(y) \leq f_i(x) - f_i(y)$$

Then, by the k-lipschitz property:

$$-k\|x - y\|_E \leq f_i(x) - f_j(y) \leq k\|x - y\|_E$$

As a result, $\max_{0 \leq i \leq m}(f_i)$ is k-Lipschitz from E to $\mathbb{R}_+$.

Our main result is the 2-Lipschitz continuity of the sequence of suboptimal permissiveness functions, that we prove in Proposition 4.11.

Proposition 4.11

For any integer $i \in \mathbb{N}$ and any location ℓ, the function $\tau_\ell : \begin{array}{ccc} \mathsf{Win}_\ell & \to & \mathbb{R}_+ \\ v & \mapsto & \mathcal{P}_i(\ell, v) \end{array}$ is 2-Lipschitz.

A direct consequence on Perm is that Perm is 2-Lipschitz when the timed automaton is **acyclic**:

Corollary 4.12

Let $\mathcal{A}$ be an acyclic timed automaton and $\ell \in Q$ a location, $\mathrm{Perm} : \mathsf{Win}_\ell \to \mathbb{R}_+$ is 2-Lipschitz.

Proof of Corollary 4.12. This is a direct consequence of Corollary 4.9. Indeed as $\mathcal{A}$ is an acyclic timed automaton $v \mapsto \mathrm{Perm}(\ell, v)$ is equal to $\mathcal{P}_{\mathrm{d}_\ell}(\ell, v)$. Therefore, because of Proposition 4.11, Perm is 2-Lipschitz on Win_ℓ. □

To prove Proposition 4.11, we use the Lemmas 2.12 and 4.13. The first one, stated in Chapter 2, is an analytic result on the p-moves that enables us, for each p-move, to consider a corresponding one with a shifted interval, in order to formally express the continuity inequalities. The second one helps us when splitting the sequence into the two terms. The first term corresponds to the size of the current interval and the second term to the sequence of suboptimal permissiveness function of the successors. Lemma 4.13 gives the 2-Lipschitz continuity of the first term.

Lemma 4.13

For any location ℓ, the function $\nu_\ell : v \mapsto \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} |I|$ is 2-Lipschitz on the set $\left\{v \in \mathbb{R}_+^{|\mathcal{C}|} \mid \mathsf{p\text{-}moves}(\ell, v) \neq \varnothing\right\}$.

Proof of Lemma 4.13. As the number of possible actions is a finite set, we can write ν_ℓ as follows:

$$\nu_\ell : v \mapsto \max_{a\in\Sigma} \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} |I|$$

Let us denote $\nu_{a,\ell} : v \mapsto \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} |I|$. If each $\nu_{a,\ell}$ is 2-Lipschitz, so is ν_ℓ. Let us prove, for a fixed $a \in \Sigma$, that $\nu_{a,\ell}$ is 2-Lipschitz.

We can suppose that ℓ has a single transition $(\ell, g, a, \mathcal{C}_r, \ell')$. Let $v, v \in \mathsf{Win}_\ell$ be two valuations. We prove that $|\nu_\ell(v') - \nu_\ell(v)| \leq 2\|v' - v\|_\infty$.

First, if $\mathsf{p\text{-}moves}(\ell, v)$ contains no p-move (I, a) whose interval size is at least $2\|v' - v\|_\infty$, then obviously:

$$\nu_{\ell,a}(v) - \nu_{\ell,a}(v') \leq 2\|v' - v\|_\infty.$$

Now, assume that there exists an enabled p-move $([\alpha, \beta], a) \in \mathsf{p\text{-}moves}(\ell, v)$ such that $\beta - \alpha \geq 2\|v' - v\|_\infty$. By Lemma 2.12, for any such interval,

$$([\alpha + \|v' - v\|_\infty, \beta - \|v' - v\|_\infty], a) \in \mathsf{p\text{-}moves}(\ell, v').$$

Fix $\varepsilon > 0$, as we can find an interval $I = [\alpha, \beta]$ such that $(I, a) \in \mathsf{p\text{-}moves}(\ell, v)$, $|I| \geq \nu_\ell(v) - \varepsilon$ and $|I| \geq 2\|v' - v\|_\infty$. Since $[\alpha + \|v' - v\|_\infty, \beta - \|v' - v\|_\infty] \in \mathsf{p\text{-}moves}(\ell, v')$, it follows:

$$\begin{aligned}
\nu_{\ell,a}(v') &\geq \beta - \|v' - v\|_\infty - \alpha - \|v' - v\|_\infty \\
\nu_{\ell,a}(v') &\geq \beta - \alpha - 2\|v' - v\|_\infty \\
\nu_{\ell,a}(v') &\geq \nu_{\ell,a}(v) - 2\|v' - v\|_\infty - \varepsilon \\
\nu_{\ell,a}(v') - \nu_{\ell,a}(v) &\geq -2\|v' - v\|_\infty - \varepsilon.
\end{aligned}$$

By symmetry of the role of v and v', we also have:

$$\begin{aligned}
\nu_{\ell,a}(v) - \nu_{\ell,a}(v') &\geq -2\|v - v'\|_\infty - \varepsilon \\
\nu_{\ell,a}(v') - \nu_{\ell,a}(v) &\leq 2\|v' - v\|_\infty + \varepsilon
\end{aligned}$$

As a result:

$$-2\|v' - v\|_\infty - \varepsilon \leq \nu_{\ell,a}(v') - \nu_{\ell,a}(v) \leq 2\|v' - v\|_\infty + \varepsilon$$

Since this holds for any $\varepsilon > 0$, we get the announced inequality and can conclude by the stability of the max operator. □

We can now prove the Proposition 4.11:

Proof of Proposition 4.11. The proof is again by induction on i. The case of $i = 0$ is trivial, as $v \mapsto \mathcal{P}_0(\ell, v)$ is constant from $\mathbb{R}_+^{|\mathcal{C}|}$ to $\mathbb{R}_+$. Let $i \in \mathbb{N}$ be an integer, and suppose that the result holds for some index i. Let ℓ be a location and let us fix an outgoing

transition $(\ell, g, a, \mathcal{C}_r, \ell')$. As in the previous proof, the result for the general case directly follows by stability. Let us pick two valuations $v, v' \in \mathsf{Win}_\ell$. In particular, $\mathsf{p\text{-}moves}\,(\ell, v)$ and $\mathsf{p\text{-}moves}\,(\ell, v')$ are non-empty. We follow the same approach as in the proof of Corollary 4.13, proving that $|\tau_\ell\,(v) - \tau_\ell\,(v')\,| \leq 2\|v' - v\|_\infty$. Again, in case $\mathsf{p\text{-}moves}\,(\ell, v)$ contains no move with an interval of size larger than or equal to $\|v' - v\|_\infty$, the result is immediate. Otherwise, let us fix $\varepsilon > 0$, and take an interval $I = [\alpha, \beta]$ such that:

$$\min\left(|I|\,, \inf_{\delta \in I}\left(\mathcal{P}_{i-1}\left(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0]\right)\right)\right) \geq \tau_\ell\,(v) - \varepsilon.$$

Then $|I| \geq \tau\,(v) - \varepsilon$ and for any $\delta \in I$:

$$\mathcal{P}_{i-1}\left(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0]\right) \geq \tau_\ell\,(v) - \varepsilon.$$

Let $I' = [\alpha + \|v' - v\|_\infty, \beta - \|v' - v\|_\infty]$, then:

$$|I'| \geq |I| - 2\|v' - v\|_\infty \geq \tau_\ell\,(v) - \varepsilon - 2\|v' - v\|_\infty.$$

Moreover, since $I' \subseteq I$, when $\delta \in I'$ we have:

$$\mathcal{P}_{i-1}\left(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0]\right) \geq \tau_\ell\,(v) - \varepsilon.$$

Additionally, for any $\delta \in I'$, $\|v' + \delta\,[\mathcal{C}_r \leftarrow 0] - v + \delta\,[\mathcal{C}_r \leftarrow 0]\,\|_\infty \leq \|v' - v\|_\infty$, so that:

$$\begin{aligned}\mathcal{P}_{i-1}\left(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0]\right) &\leq \mathcal{P}_{i-1}\left(\ell', v' + \delta'\,[\mathcal{C}_r \leftarrow 0]\right) + 2\|v' + \delta\,[\mathcal{C}_r \leftarrow 0] - v + \delta\,[\mathcal{C}_r \leftarrow 0]\,\|_\infty \\ &\leq \mathcal{P}_{i-1}\left(\ell', v' + \delta'\,[\mathcal{C}_r \leftarrow 0]\right) + 2\|v' - v\|_\infty.\end{aligned}$$

Thus for any $\delta \in I'$,

$$\begin{aligned}\mathcal{P}_{i-1}\left(\ell', v' + \delta\,[\mathcal{C}_r \leftarrow 0]\right) &\geq \mathcal{P}_{i-1}\left(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0]\right) - 2\|v' - v\|_\infty \\ &\geq \tau_\ell\,(v) - \varepsilon - 2\|v' - v\|_\infty\end{aligned}$$

Since $|I'| \geq \tau_\ell\,(v) - \varepsilon - 2\|v' - v\|_\infty$, and $\tau_\ell\,(v') \geq \min\left(|I'|\,, \inf_{\delta \in I'}\left(\mathcal{P}_{i-1}\left(\ell', v' + \delta\,[\mathcal{C}_r \leftarrow 0]\right)\right)\right)$ we have:

$$\tau_\ell\,(v') - \tau_\ell\,(v) \geq -\varepsilon - 2\|v' - v\|_\infty.$$

By symmetry of the role of v and v', $\tau_\ell(v) - \tau_\ell(v') \geq -\varepsilon - 2\|v - v'\|_\infty$, so:

$$-\varepsilon - 2\|v' - v\|_\infty \leq \tau_\ell(v') - \tau_\ell(v) \leq \varepsilon + 2\|v' - v\|_\infty.$$

This prove that τ_ℓ is 2-Lipschitz. □

4.2.4 Optimisation properties

In this section, we prove properties about the optimisation of the opponent strategy.

The following lemma proves that $\mathcal{P}_i(\ell, v + \delta) \leq \mathcal{P}_i(\ell, v)$. A direct consequence of this lemma is that for any **non-resetting transition**, the optimal choice of the opponent is the largest delay in the interval proposed by the player. As proved in [BFM15], this choice is also optimal for any one-clock timed automaton, even with resets. Indeed, in that case, the clock is reset and the new valuation does not depend on the delay chosen by the opponent. This corresponds to the intuition that by playing later, the opponent forces a faster reaction from the player at the next step. However this is not the optimal strategy in general for multiple-clock timed automata (see Counter-example 5.1.1 in the next section).

Lemma 4.14

Let (ℓ, v) be a configuration, $\delta \in \mathbb{R}_+$ such that $(\ell, v + \delta)$ is a configuration of the automaton, and $i \in \mathbb{N}$. Then $\mathcal{P}_i(\ell, v) - \delta \leq \mathcal{P}_i(\ell, v + \delta) \leq \mathcal{P}_i(\ell, v)$.

Proof of Lemma 4.14. First, let us prove that $\mathcal{P}_i(\ell, v + \delta) \leq \mathcal{P}_i(\ell, v)$. For any enabled p-move (I, a) from $(\ell, v + \delta)$, the p-move $(I + \delta, a)$ is an enabled p-move from (ℓ, v). Moreover, the set of valuations on which $\mathcal{P}_{i-1}$ is minimised is the same in both cases, namely $\{(v + \delta[\mathcal{C}_r \leftarrow 0]) \mid \delta \in I\}$. It follows that $\mathcal{P}_i(\ell, v + \delta) \leq \mathcal{P}_i(\ell, v)$.

Finally, let us prove that $\mathcal{P}_i(\ell, v) - \delta \leq \mathcal{P}_i(\ell, v + \delta)$. For any enabled p-move (I, a) from (ℓ, v) with $|I| \geq \delta$ (if any), the p-move $((I - \delta) \cap \mathbb{R}_+, a)$ is an enabled p-move from $(\ell, v + \delta)$. The set of valuation on which $\mathcal{P}_{i-1}$ is minimised is $\{v + \delta[\mathcal{C}_r \leftarrow 0] \mid \delta \in I\}$ too. As $|(I - \delta) \cap \mathbb{R}_+| \leq |I|$, the second inequality follows. □

In the following proposition, we state that the function $v \mapsto \mathcal{P}_i(\ell, v)$ is a concave function from Win_ℓ to $\mathbb{R}_+$ for any **linear** timed automaton. The direct corollary of this property is that the opponent's best choice is either the earliest or the latest delay (see Corollary 5.1 in the next section) for linear timed automaton.

Proposition 4.15

Let $i \in \mathbb{N}$. Let ℓ be a location of a linear timed automaton $\mathcal{A}$, the function $\mathcal{P}_i(\ell, \cdot) : v \mapsto \mathcal{P}_i(\ell, v)$ is a concave function over Win_ℓ.

Let us recall briefly what a concave function is. $\begin{array}{ccc} \mathsf{Win}_\ell & \to & \mathbb{R}_+ \\ v & \mapsto & \mathcal{P}_i(\ell, v) \end{array}$ is concave if and only if, for any $v_1, v_2 \in \mathsf{Win}_\ell$, for any $\lambda \in [0;1]$, and $v_\lambda = \lambda \cdot v_1 + (1-\lambda) \cdot v_2$, the following inequality holds:

$$\mathcal{P}_i(\ell, v_\lambda) \geq \lambda \cdot \mathcal{P}_i(\ell, v_1) + (1-\lambda) \cdot \mathcal{P}_i(\ell, v_2).$$

Proof of Proposition 4.15. We prove this proposition with an induction on i. For $i = 0$, it is trivial since $\mathcal{P}_0(\ell, v)$ does not depend on v. Let i be an arbitrary positive integer and let us assume that the result holds for $\mathcal{P}_i$. Let us now prove that it still holds for $\mathcal{P}_{i+1}$. Let ℓ be an arbitrary location of the automaton $\mathcal{A}$, and $(\ell, g, a, \mathcal{C}_r, \ell')$ be its unique outgoing transition (as $\mathcal{A}$ is linear). Let $(I_j, a) \in \mathsf{p\text{-}moves}(\ell, v_j)$ for $j \in \{1, 2\}$. By definition of $\mathsf{p\text{-}moves}$, for $j \in \{1, 2\}$ we then have $v_j + \delta_j \models g$ for any $\delta_j \in I_j$. We can then define the following set:

$$I_\lambda = \{\lambda \cdot \delta_1 + (1-\lambda) \cdot \delta_2 \mid \delta_1 \in I_1,\ I_2 \in I_2\}.$$

I_λ is an interval. Let us pick any $\delta_\lambda \in I_\lambda$: then, by definition of I_λ, there exists $\delta_1 \in I_1$ and $\delta_2 \in I_2$ such that $\delta_\lambda = \lambda \cdot \delta_1 + (1-\lambda) \cdot \delta_2$. Then $v_\lambda + \delta_\lambda$ can be written as $\lambda \cdot (v_1 + \delta_1) + (1-\lambda) \cdot (v_2 + \delta_2)$. Since both $v_1 + \delta_1$ and $v_2 + \delta_2$ satisfy the guard g and since, by Proposition 2.3, g is convex, we have that $v_\lambda + \delta_\lambda \models g$. This proves that $(I_\lambda, a) \in \mathsf{p\text{-}moves}(\ell, v_\lambda)$. Moreover $|I_\lambda| = \lambda \cdot |I_1| + (1-\lambda) \cdot |I_2|$. Let us fix $\varepsilon > 0$, and take two intervals I_1 and I_2 such that:

$$\forall j \in \{1, 2\}, \min\left(|I_j|, \inf\left\{\mathcal{P}_i\left(\ell', v'_j\right) \mid \exists \delta \in I_j.\ (\ell, v_j) \xrightarrow{\delta, a} \left(\ell', v'_j\right)\right\}\right) \geq \mathcal{P}_i(\ell, v_j) - \varepsilon.$$

We can define I_λ as above. Then:

$$\mathcal{P}_{i+1}(\ell, v_\lambda) = \sup_{(I,a) \in \mathsf{p\text{-}moves}(\ell, v_\lambda)} \min\left(|I|, \inf_{\delta \in I} \mathcal{P}_i(\mathrm{succ}(\ell, v_\lambda, \delta, a))\right)$$

As the supremum over all p-moves of this minimum is larger than (or equal) to the minimum for any particular p-move (I_λ, a):

$$\begin{aligned}\mathcal{P}_{i+1}(\ell, v_\lambda) &\geq \min\left(|I_\lambda|, \inf\left\{\mathcal{P}_i(\ell', v'_\lambda) \mid \exists \delta \in I_\lambda.\ (\ell, v_\lambda) \xrightarrow{\delta, a} (\ell', v'_\lambda)\right\}\right) \\ &= \min\left(|I_\lambda|, \inf_{\delta_\lambda \in I_\lambda}\left(\mathcal{P}_i(\ell', v_\lambda + \delta_\lambda\left[\mathcal{C}_r \leftarrow 0\right])\right)\right)\end{aligned}$$

As $I_\lambda = \lambda \cdot I_1 + (1-\lambda) \cdot I_2$:

$$\mathcal{P}_{i+1}(\ell, v_\lambda) \geq \min\left(|I_\lambda|, \inf_{\delta_1 \in I_1, \delta_2 \in I_2}\left(\mathcal{P}_i(\ell', (\lambda \cdot (v_1 + \delta_1) + (1-\lambda) \cdot (v_2 + \delta_2))\left[\mathcal{C}_r \leftarrow 0\right])\right)\right)$$

By linearity of the projection over the valuations:

$$= \min\left(|I_\lambda|, \inf_{\delta_1 \in I_1, \delta_2 \in I_2}\left(\mathcal{P}_i(\ell', \lambda \cdot v_1 + \delta_1\left[\mathcal{C}_r \leftarrow 0\right] + (1-\lambda) \cdot v_2 + \delta_2\left[\mathcal{C}_r \leftarrow 0\right])\right)\right)$$

We apply the induction hypothesis:

$$\begin{aligned}\mathcal{P}_{i+1}(\ell, v_\lambda) &\geq \min\left(|I_\lambda|, \inf_{\delta_1 \in I_1, \delta_2 \in I_2}\left(\mathcal{P}_i(\ell', \lambda \cdot v_1 + \delta_1\left[\mathcal{C}_r \leftarrow 0\right]) + (1-\lambda) \cdot \mathcal{P}_i(\ell', v_2 + \delta_2\left[\mathcal{C}_r \leftarrow 0\right])\right)\right) \\ &= \min\left(\lambda \cdot |I_1| + (1-\lambda)|I_2|, \lambda \cdot \inf_{\delta_1 \in I_1}\left(\cdot \mathcal{P}_i(\ell', v_1 + \delta_1\left[\mathcal{C}_r \leftarrow 0\right])\right.\right. \\ &\quad \left.\left. + (1-\lambda) \cdot \inf_{\delta_2 \in I_2} \mathcal{P}_i(\ell', v_2 + \delta_2\left[\mathcal{C}_r \leftarrow 0\right])\right)\right)\end{aligned}$$

As $\min(a + b, a' + b') \geq \min(a, a') + \min(b, b')$:

$$\begin{aligned}\mathcal{P}_{i+1}(\ell, v_\lambda) &\geq \lambda \cdot \min\left(|I_1|, \inf_{\delta_1 \in I_1} \mathcal{P}_i(\ell', v_1 + \delta_1\left[\mathcal{C}_r \leftarrow 0\right])\right) \\ &\quad + (1-\lambda) \cdot \min\left(|I_2|, \inf_{\delta_2 \in I_2} \mathcal{P}_i(\ell', v_2 + \delta_2\left[\mathcal{C}_r \leftarrow 0\right])\right)\end{aligned}$$

As for all $j \in \{1, 2\}$, $\min\left(|I_j|, \inf\left\{\mathcal{P}_i\left(\ell', v'_j\right) \mid \exists \delta \in I_j.\ (\ell, v_j) \xrightarrow{\delta, a} \left(\ell', v'_j\right)\right\}\right) \geq \mathcal{P}_i(\ell, v_j) - \varepsilon$, the following inequality holds:

$$\mathcal{P}_{i+1}(\ell, v_\lambda) \geq \lambda \cdot \mathcal{P}_{i+1}(\ell, v_1) + (1-\lambda) \cdot \mathcal{P}_{i+1}(\ell, v_2) - 2\varepsilon$$

As ε is arbitrary, we consider its limits to 0, then $\mathcal{P}_{i+1}(\ell, v_\lambda) \geq \lambda \cdot \mathcal{P}_{i+1}(\ell, v_1) + (1-\lambda) \cdot \mathcal{P}_{i+1}(\ell, v_2)$ By induction, for all $i \in \mathbb{N}$,

$$\mathcal{P}_i(\ell, v_\lambda) \geq \lambda \cdot \mathcal{P}_i(\ell, v_1) + (1-\lambda) \cdot \mathcal{P}_i(\ell, v_2)$$

□

Conclusion

In this section, we have exposed several properties of the sequence of suboptimal permissive function $\mathcal{P}_i$. Its useful analytic and evolution properties and its link with the permissiveness function give us a very useful tool to compute the permissiveness function. Indeed, we proved that this sequence is eventually constant from a specific rank, and converges to the permissiveness. We can then reduce the computation of the permissiveness to the computation of $\mathcal{P}_i$ for the appropriate index i. This property holds if the timed automaton is **acyclic**. We did not manage to prove that it still holds when tackling cycles.

In the next chapter, we expose an algorithm that computes with a symbolic approach the sequence of suboptimal permissive functions, in order to solve the maximal-permissive problem (Definition 2.22) and the maximal-sequence of suboptimal permissive functions problem (Definition 4.2).

Chapter 5

Maximal-permissiveness problem: a symbolic backward algorithm

The goal of this chapter is to provide a constructive proof of an upper bound of the complexity of the maximal-permissiveness problem defined in Definition 2.22. More precisely, we show that this problem can be computed in **non-elementary time** for **acyclic** timed automata and games. The algorithm we propose computes the strategy of the player for any location by computing the function $v \mapsto \mathrm{Perm}(\ell, v)$. This is a symbolic approach.

However, the permissiveness function of a configuration is expressed with p-runs. We defined in Chapter 4 a sequence of functions, called **sequence of suboptimal permissive functions** which depends on its possible direct successors[1] and the current proposed interval. We studied its link with the permissiveness function in Section 4.2.

We also stated that the sequence of functions $\mathcal{P}_i(\ell, v)$ is constant for acyclic timed automata from a rank d_ℓ we computed. This sequence $\mathcal{P}_i(\ell, v)$ converges to the permissive function of (ℓ, v). As a result, we only have to compute $\mathcal{P}_{d_\ell}(\ell, v)$ to compute the permissiveness function. Let us recall its form for a winning configuration (ℓ, v) such that $\ell \notin Q_f$:

$$\mathcal{P}_i(\ell, v) = \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|I|, \inf_{\delta\in I} \mathcal{P}_{i-1}(\mathrm{succ}(\ell, v, \delta, a))\right)$$

For the purpose of simplification, we first develop this algorithm for linear timed automata with closed polyhedral guards and then present possible extensions: non-necessary closed p-moves, acyclic timed automata and acyclic timed games. Computing the permissiveness for linear timed automata is simpler, as $\mathcal{P}_{i+1}(\ell, v)$ is concave over Win_ℓ. Therefore, the best choice of the opponent is to choose one of the bounds of the interval I. As a result the sequence of functions will be simplified as follows:

[1] that means the configurations that can be reached when taking only one transition.

$$\sup_{([\alpha,\beta],a)\in \mathsf{p\text{-}moves}(\ell,v)} \min\left(|\beta-\alpha|, \mathcal{P}_{i-1}\left(\mathrm{succ}\left(\ell,v,\alpha,a\right)\right), \mathcal{P}_{i-1}\left(\mathrm{succ}\left(\ell,v,\beta,a\right)\right)\right)$$

This chapter is organised as follows:

- ▷ First, we present the algorithm for linear timed automata in Section 5.1:
  - In Subsection 5.1.1, we compute an optimal strategy for the opponent, which is to choose one of the bounds of the proposed interval.
  - In Subsection 5.1.2, we present an algorithm to compute the sequence $\mathcal{P}_i$ and show that the maximal-permissiveness problem can be solved in non-elementary time.
  - Finally, we present an example of an execution of this algorithm in Subsection 5.1.3.
- ▷ Secondly, we extend this algorithm to more general models in Section 5.2. We extend it for:
  - Non necessarily closed p-moves in Subsection 5.2.1.
  - Acyclic timed automata in Subsection 5.2.2.
  - Acyclic timed games in Subsection 5.2.3.
- ▷ Thirdly and finally, this algorithm uses a powerful optimisation result that we present in the dedicated Section 5.3.

5.1 A symbolic backward algorithm for linear timed automata

The goal of this section is to compute, for any linear timed automaton, for any location ℓ and integer i, the function $v \mapsto \mathcal{P}_i(\ell, v)$.

Remark 5.1.1 *As only one action a will be possible for each location, we will sometimes simplify the notation of a p-move (I, a) using only the interval I.*

Firstly, we prove formally the optimal strategy of the opponent for linear automata. It enables us to simplify a lot the expression of $\mathcal{P}_i(\ell, v)$. Secondly, we design an algorithm

that computes the function $\mathcal{P}_i(\ell, v)$ for any integer i. Our main contribution in this section is to prove that the maximal-permissiveness problem presented in Definition 2.22 can be solved in non-elementary time for linear timed automata. To prove that, we show that, considering the successor location ℓ' of ℓ and g the guard associated to the transition between ℓ and ℓ', assuming that $v \mapsto \mathcal{P}_i(\ell', v)$ is a piecewise-affine function (with m cells, represented with at most c_i linear inequalities), $v \mapsto \mathcal{P}_{i+1}(\ell, v)$ can be computed in time $\mathcal{O}\left((c_i + c_g)^{4 \cdot m^2}\right)$, where c_g is the number of linear inequalities that represent g

5.1.1 Optimal strategy for the opponent

Let us consider a configuration (ℓ, v), a transition $(\ell, g, \mathcal{C}_r, \ell')$, an enabled p-move (I, a) proposed by the player and a step $i > 0$. The goal of the opponent is to propose the delay $\delta \in I$ that minimises $\mathcal{P}_{i-1}((\ell', v + \delta[\mathcal{C}_r \leftarrow 0]))$.

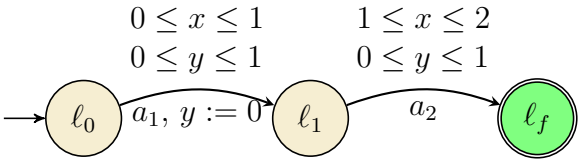
In the model of [BFM15], the best choice of the opponent is to choose the greatest delay, but this model is restricted to one-clock timed automata. We extend this result to our model for **non-resetting transitions** (see Lemma 4.14) where the best option for the opponent is also to choose the greatest delay. Nevertheless, when a timed automaton contains resets, the greatest delay is not always the best opponent's choice. Let us develop a counter-example in Example 5.1.1.

Example 5.1.1 *Let us consider the timed automaton in Figure 5.1a. Let us analyse an example of two different p-runs that both start in the configuration $(\ell_0, (0, 0))$ and are illustrated in Figure 5.1b. In these p-runs the player proposes the interval $[0, 1]$. Then, let us show two responses by the opponent that illustrate that, in this case, choosing the greatest delay is not his best strategy.*

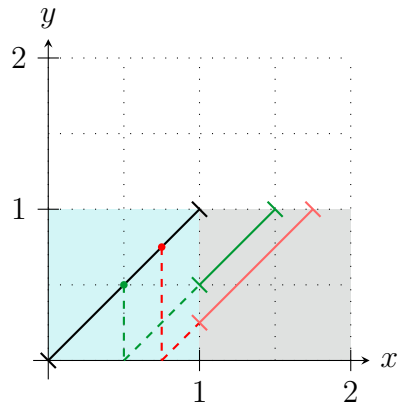
- *In the first p-run, the opponent chooses the delay $\delta = 0.5$. The future configuration after the transition is then $(\ell_1, (0.5, 0))$. The greatest interval that can be proposed to pass the second transition is $[0.5, 1]$. The permissiveness of this p-run is then 0.5 (see the green p-run in Figure 5.1b)*
- *In the second p-run, the opponent chooses the delay $\delta = 0.75$, and reaches the future configuration $(\ell_1, (0.75, 0))$. In this configuration, a greater interval is enabled: $[0.25, 1.75]$. The permissiveness is then 0.75 (see the red p-run in Figure 5.1b).*
- *Finally, if the opponent chooses the delay $\delta = 0$, the future configuration reached is $(\ell_1, (0, 0))$. In this configuration, the player have only one choice: propose a punctual*

interval $\{1\}$*. The permissiveness is then* 0*.*

In this example, the best strategy of the opponent is to propose the smallest possible delay, $\delta = 0$*.*



(a) A timed automaton with a reset.



(b) Two possible p-runs when proposing $([0,1], a_1)$ in configuration $(\ell_0, (0,0))$.

Figure 5.1: An automaton with a reset and two of its possible p-runs.

This counter-example proves that we cannot extend for every timed automaton, linear or not, the result of [BFM15]. Nevertheless, in the case of linear timed automata, we use the concavity result of Subsection 4.2.4 to prove that an optimal strategy of the opponent is to choose between the bounds of the interval of a proposed p-move. We state this result in Corollary 5.1.

Corollary 5.1

Let ℓ be a location of a linear timed automaton. For any valuation v, and bounded interval $[\alpha, \beta]$, and any transition $(\ell, g, a, \mathcal{C}_r, \ell')$:

$$\inf_{\delta \in [\alpha,\beta]} \left(\mathcal{P}_i\left(\text{succ}\left(\ell, v, \delta, a\right)\right)\right) = \min\left(\mathcal{P}_i\left(\ell', v_\alpha\right), \mathcal{P}_i\left(\ell', v_\beta\right)\right)$$

where $v_\alpha := v + \alpha\left[\mathcal{C}_r \leftarrow 0\right]$ and $v_\beta := v + \beta\left[\mathcal{C}_r \leftarrow 0\right]$.

Proof of Corollary 5.1. Let v and v' be two clock valuations, $\lambda \in [0,1]$, and $v_\lambda = \lambda \cdot v + (1-\lambda) \cdot v'$. Then for all i, because of Proposition 4.15:

$$\mathcal{P}_i\left(\ell, v_\lambda\right) \geq \min\left(\mathcal{P}_i\left(\ell, v\right), \mathcal{P}_i\left(\ell, v'\right)\right)$$

Additionally, we have:

$$\inf_{\delta\in[\alpha,\beta]}\left(\mathcal{P}_i\left(\mathrm{succ}\left(\ell,v,\delta,a\right)\right)\right)=\inf\left\{\mathcal{P}_i\left(\ell',v'\right)\mid\exists\delta\in[\alpha,\beta]\,.\;(\ell,v)\xrightarrow{\delta,a}(\ell',v')\right\}$$

As $\left(v+\left(\lambda\alpha+(1-\lambda)\,\beta\right)\right)\left[\mathcal{C}_r\to 0\right]=\lambda\left(\left(v+\alpha\right)\left[\mathcal{C}_r\to 0\right]\right)+(1-\lambda)\left(\left(v+\beta\right)\left[\mathcal{C}_r\to 0\right]\right)$:

$$\begin{aligned}\inf_{\delta\in[\alpha,\beta]}\left(\mathcal{P}_i\left(\mathrm{succ}\left(\ell,v,\delta,a\right)\right)\right)&=\inf\left\{\mathcal{P}_i\left(\ell',v'\right)\mid\exists\lambda\in[0,1]\,.\;v'=\lambda\cdot v'_\alpha+(1-\lambda)\cdot v'_\beta\right\}\\&=\min\left(\mathcal{P}_i\left(\ell',v_\alpha\right),\mathcal{P}_i\left(\ell',v_\beta\right)\right).\end{aligned}$$

□

This corollary only applies for linear timed automata. For acyclic timed automata, we explain the best strategy of the opponent in the Subsection 5.2.2.

5.1.2 Description and proof of the algorithm

Now that we have a better understanding of the optimal strategy of the opponent, the expression of $\mathcal{P}_i\left(\ell,\cdot\right)$ is simplified and we only have to compute the supremum of the following expression for the set of valuations v such that $\mathsf{p\text{-}moves}\left(\ell,v\right)\neq\varnothing$. $\mathcal{P}_i\left(\ell,\cdot\right):$ $v\mapsto\sup_{([\alpha,\beta],a)\in\mathsf{p\text{-}moves}(\ell,v)}\min\left(\beta-\alpha,\mathcal{P}_{i-1}\left(\ell',v+\alpha\left[\mathcal{C}_r\leftarrow 0\right]\right),\mathcal{P}_{i-1}\left(\ell',v+\beta\left[\mathcal{C}_r\leftarrow 0\right]\right)\right)$

Our goal in this subsection is to compute the most-permissive strategy of the player for reaching a goal $\ell_f\in Q_f$. We prove that for any integer i and any location ℓ, the function $v\mapsto\mathcal{P}_i\left(\ell,v\right)$ is a continuous piecewise-affine function over Win_ℓ.

We provide an algorithm to compute this function. First notice that, following Lemma 4.5, for any location ℓ of a linear timed automaton, the sequence of functions $\mathcal{P}_i\left(\ell,\cdot\right)_i$ converges in at most d_ℓ steps (to Perm).

We state this result in Theorem 5.2. To prove that, we compute the permissiveness function step by step with the sequence of suboptimal permissive function $\mathcal{P}_i$ in Lemma 5.3

Theorem 5.2

Let $\mathcal{A}$ be a linear timed automaton with n clocks and with polyhedral guards, represented with at most c_g linear inequalities. For any location ℓ, $v\mapsto\mathrm{Perm}\left(\ell,v\right)$ is a piecewise-affine function that can be computed in **non-elementary** time.

To compute the permissiveness function, we can apply the following Algorithm 1.

Data: A linear timed automaton $\mathcal{A}$, a location ℓ_0
Result: $v \mapsto \mathrm{Perm}(\ell, v)$
Compute $v \mapsto \mathcal{P}_0(\ell_f, v)$;
$\ell \leftarrow \ell_f$;
for $i \leftarrow 1$ **to** d_ℓ **do**
  Compute $v \mapsto \mathcal{P}_i(\ell, v)$ with the algorithm presented in the proof of Lemma 5.3 and $v \mapsto \mathcal{P}_{i-1}(\ell', v)$, where ℓ' is the successor of ℓ;
  $\ell \leftarrow$ predecessor of ℓ;
end
return $v \mapsto \mathcal{P}_{d_{\ell_0}}(\ell_0, v)$

Algorithm 1: Computation of $v \mapsto \mathrm{Perm}(\ell, v)$.

Let us state the algorithm to compute the sequence of suboptimal permissive functions in the proof of Lemma 5.3.

Lemma 5.3

Let $\mathcal{A}$ be a linear timed automaton with polyhedral guards, represented with at most c_g linear inequalities, with n clocks. Let $(\ell, g, a, \mathcal{C}_r, \ell')$ be a transition of $\mathcal{A}$.
Assume that $v \mapsto \mathcal{P}_i(\ell', v)$ is an n-dimensional piecewise-affine function over $\mathbb{R}^n_+$, such that its restriction over $\mathsf{Win}_{\ell'}$ is continuous and can be represented with an m-cells tiling of polyhedra, such that each polyhedron is represented with at most c_i linear inequalities.
Then $v \mapsto \mathcal{P}_{i+1}(\ell, v)$ is an n-dimensional piecewise-affine function, continuous over Win_ℓ, represented with a tiling of polyhedra with at most $\mathcal{O}\left((c_i + c_g)^{4 \cdot m^2}\right)$-cells, such that each polyhedron can be represented with at most $3 \cdot m^2 \left((c_i + c_g) + 5\right)$ linear inequalities.
It can be computed in time $\mathcal{O}\left((c_i + c_g)^{4 \cdot m^2}\right)$.

5.1.2.1 Proof of Lemma 5.3.

There are some basic cases that can be solved trivially.

1. If $v \mapsto \mathcal{P}_i(\ell', v)$ is constantly $-\infty$, then also is $v \mapsto \mathcal{P}_{i+1}(\ell, v)$.
2. Similarly, if $\mathsf{p\text{-}moves}(\ell, v)$ is empty for every valuation v, then $v \mapsto \mathcal{P}_{i+1}(\ell, v)$

is constantly $-\infty$.

Let us now suppose that these previous conditions do not hold. Then, $v \mapsto \mathcal{P}_i(\ell', v)$ is not constantly $-\infty$ and there is some v such that $\mathsf{p\text{-}moves}(\ell, v) \neq \varnothing$. Since $v \mapsto \mathcal{P}_i(\ell', v)$ is a continuous n-dimensional piecewise-affine function over Win_ℓ, we can represent it with a tiling of polyhedra $\mathcal{T} = (h_i)_{1 \leq i \leq c_i}$ associated with a set of affine functions $(f_i)_{1 \leq i \leq c_i}$. Let us recall that for each polyhedron $h_j \in \mathcal{T}$, for any valuation $v \in h_j$, $\mathcal{P}_i(\ell', v) = f_i(v)$.

Let us present the organisation of the proof in the following subsection.

Organisation of the proof Our procedure for computing $\mathcal{P}_{i+1}$ in ℓ first consists in listing all the possible pairs of cells (h_α, h_β) of the tiling of polyhedra defining $\mathcal{P}_i$ in ℓ'. For each pair (h_α, h_β) of such cells, we perform the following three steps (illustrated in Figure 5.2):

Step 1 : We characterise the set $\mathcal{S}_{(h_\alpha, h_\beta)}$ of all valuations from which those cells can be reached, by proposing an enabled p-move $([\alpha, \beta], a]$ in order to take the transition from ℓ to ℓ'. We compute this polyhedron using quantifier elimination with the Fourier-Motzkin algorithm.

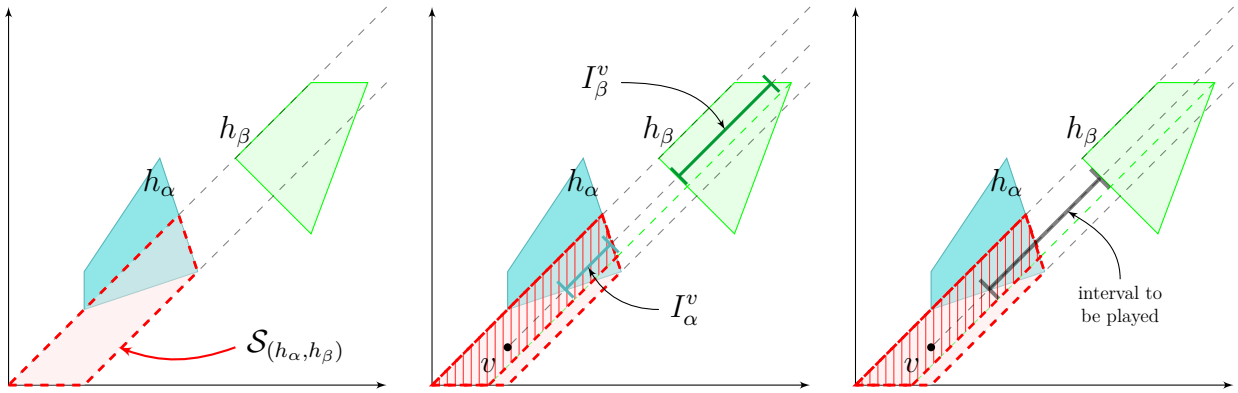
Step 2 : Then, we compute which p-moves can be proposed. For any valuation v of $\mathcal{S}_{(h_\alpha, h_\beta)}$, we compute the smallest and the greatest lower (*resp.* upper) bound that can be proposed as an interval $[\alpha, \beta]$. We denote the set of possible lower (*resp.* upper) bounds I_α^v (*resp.* I_β^v). These are intervals and their bounds are expressed as piecewise-affine functions in the coefficients of v. In order to apply the next step of the procedure, our aim is to express these bounds as affine functions. To do so, we may have to tile the polyhedron $\mathcal{S}_{(h_\alpha, h_\beta)}$ such that in each cell of the tiling of polyhedra, I_α^v and I_β^v will be affine functions of v.

Step 3 : For each refined polyhedron, compute the optimal values for α and β: following Corollary 5.1, this amounts to find values for $\alpha \in I_\alpha^v$ and $\beta \in I_\beta^v$ that maximise the following function:

$$\mu : (\alpha, \beta) \mapsto \min\left(\beta - \alpha, \mathcal{P}_i(\ell', v + \alpha[\mathcal{C}_r \leftarrow 0]), \mathcal{P}_i(\ell', v + \beta[\mathcal{C}_r \leftarrow 0])\right) .$$

This is performed by applying an optimisation result that we will develop in Subsection 5.3, for the sake of clarity. It may again require another refinement of the

sub-polyhedra. Over the resulting sub-polyhedron, the supremum of μ is an affine function with respect to v.



(a) Step 1. (b) Step 2. (c) Step 3.

Figure 5.2: The three steps of our algorithm. Step 1: compute $\mathcal{S}_{(h_\alpha,h_\beta)}$. Step 2: compute expressions for I^v_α and I^v_β for any v. Notice that we consider a sub-polyhedron (hatched zone) of $\mathcal{S}_{(h_\alpha,h_\beta)}$ because we had to refine it. Indeed the expression of I^v_β would be different for the lower part of $\mathcal{S}_{(h_\alpha,h_\beta)}$, since it ends on a different facet of h_β. Step 3: select best values for α and β.

For each pair (h_α, h_β), we end up with a piecewise-affine function, defined on $\mathcal{S}_{(h_\alpha,h_\beta)}$, returning the optimal $\mathcal{P}_i$ that can be obtained when proposing the p-move $([\alpha,\beta], a)$ such that taking the transition to ℓ' after delay α (*resp.* β) leads to h_α (*resp.* h_β).

Our final step to compute $\mathcal{P}_{i+1}$ in ℓ consists in taking the maximum of all these functions; this may introduce one more refinement of our polyhedra.

Let us remark that we perform all these computations symbolically with respect to the valuation v. Indeed our goal is to compute $\mathcal{P}_{i+1}(\ell, v)$ for any valuation v. To do so, we manipulate affine functions of v, on specific sets of valuations. For instance for each pair of successor cells h_α and h_β, we compute the polyhedron $\mathcal{S}_{(h_\alpha,h_\beta)}$ to determine the valuations v that can reach h_α or h_β. That gives us conditions on v, used to define I^v_α and I^v_β with respect to v.

Let us now detail the first three steps. For this we fix two cells h_α and h_β in $\mathcal{T}$.

Step 1: Computing $\mathcal{S}_{(h_\alpha,h_\beta)}$. The set $\mathcal{S}_{(h_\alpha,h_\beta)}$ of valuations that can reach h_α and h_β with delays $\alpha \leq \beta$ (and after taking the transition to ℓ') is defined as follows:

$$\mathcal{S}_{(h_\alpha,h_\beta)} := \left\{ v \in \overline{\mathbb{R}}_+^n \mid \exists 0 \leq \alpha \leq \beta \ s.t. \ \begin{cases} v+\alpha \models g, v+\beta \models g \\ v+\alpha\left[\mathcal{C}_r \leftarrow 0\right] \in h_\alpha, v+\beta\left[\mathcal{C}_r \leftarrow 0\right] \in h_\beta \end{cases} \right\}.$$

Let us first introduce some notations:

Guards: We denote the inequalities of g as follows: $\bigwedge_{j=1}^{c_g} \varphi_j^{(g)}(X) \geq 0$.

Cells h_α, h_β: These are polyhedra, each defined with at most c_i inequalities. Let us denote these inequalities as follows:

$$h_\alpha : \bigwedge_{k=1}^{c_i} \varphi_k^{(\alpha)}(X) \geq 0, h_\beta : \bigwedge_{k=1}^{c_i} \varphi_k^{(\beta)}(X) \geq 0$$

With these notations, we can re-write $\mathcal{S}_{(h_\alpha,h_\beta)}$ as the set of valuations v such that there exists $0 \leq \alpha \leq \beta$ that satisfy the following enumerated inequalities:

$$\alpha \geq 0, \alpha \leq \beta \tag{5.1}$$
$$v+\alpha \models g : \forall 0 \leq j \leq c_g : \varphi_j^{(g)}(v+\alpha) \geq 0 \tag{5.2}$$
$$v+\beta \models g : \forall 0 \leq j \leq c_g : \varphi_j^{(g)}(v+\beta) \geq 0 \tag{5.3}$$
$$v+\alpha\left[\mathcal{C}_r \leftarrow 0\right] \in h_\alpha : \forall 0 \leq k \leq c_i : \varphi_k^{(\alpha)}(v+\alpha\left[\mathcal{C}_r \leftarrow 0\right]) \geq 0 \tag{5.4}$$
$$v+\beta\left[\mathcal{C}_r \leftarrow 0\right] \in h_\beta : \forall 0 \leq k \leq c_i : \varphi_k^{(\beta)}(v+\beta\left[\mathcal{C}_r \leftarrow 0\right]) \geq 0 \tag{5.5}$$

Let us eliminate α and β from these inequalities. Let $j \in [0 \cdots c_g]$ and $k \in [0 \cdots c_i]$. Let us express the n-dimensional affine function $\varphi_j^{(g)}$ with its coefficients:

$$\varphi_j^{(g)}(v+\alpha) := \sum_{i'=0}^{n} \varphi_{j,i'}^{(g)}(v_{i'}+\alpha) \text{ and } \varphi_j^{(g)}(v+\beta) := \sum_{i'=0}^{n} \varphi_{j,i'}^{(g)}(v_{i'}+\beta)$$

Let us denote $S_j^{(g)} = \sum_{i'=0}^{n} \varphi_{j,i'}^{(g)}$, then:

$$\varphi_j^{(g)}(v+\alpha) = S_j^{(g)} \cdot \alpha + \varphi_j^{(g)}(v) \text{ and } \varphi_j^{(g)}(v+\beta) = S_j^{(g)} \cdot \beta + \varphi_j^{(g)}(v)$$

As previously:

$$\varphi_k^{(\alpha)}\left(v+\alpha\left[\mathcal{C}_r\leftarrow 0\right]\right) := \sum_{i'=0, i'\notin\mathcal{C}_r}^{n} \varphi_{k,i'}^{(\alpha)}\left(v_{i'}+\alpha\right)$$

Let us denote $S_k^{(\alpha)} = \sum_{i'=0, i'\notin\mathcal{C}_r}^{n} \varphi_{k,i'}^{(\alpha)}$, then:

$$\varphi_k^{(\alpha)}\left(v+\alpha\left[\mathcal{C}_r\leftarrow 0\right]\right) = S_k^{(\alpha)}\cdot\alpha + \varphi_k^{(\alpha)}\left(v\left[\mathcal{C}_r\leftarrow 0\right]\right)$$

As previously, let us denote $S_k^{(\beta)} = \sum_{i'=0, i'\notin\mathcal{C}_r}^{n} \varphi_{k,i'}^{(\beta)}$, then:

$$\varphi_k^{(\beta)}\left(v+\beta\left[\mathcal{C}_r\leftarrow 0\right]\right) = S_k^{(\beta)}\cdot\beta + \varphi_k^{(\beta)}\left(v\left[\mathcal{C}_r\leftarrow 0\right]\right)$$

In order to isolate α and β in these inequalities, we divide our inequalities by $S_j^{(g)}$ (or $S_k^{(\alpha)}$ or $S_k^{(\beta)}$). Therefore we have to distinguish the cases where $S_j^{(g)}$ (or $S_k^{(\alpha)}$ or $S_k^{(\alpha)}$) is positive, negative, or equals to zero. Let us denote the following sets.

$$\begin{array}{lll}
\mathrm{Pos}_g = \left\{j\in[1\cdots n] \mid S_j^{(g)}>0\right\}, & \mathrm{Neg}_g = \left\{j\in[1\cdots n] \mid S_j^{(g)}<0\right\}, & \mathrm{Zero}_g = \left\{j\in[1\cdots n] \mid S_j^{(g)}=0\right\} \\
\mathrm{Pos}_\alpha = \left\{k\in[1\cdots n] \mid S_k^{(\alpha)}>0\right\}, & \mathrm{Neg}_\alpha = \left\{k\in[1\cdots n] \mid S_k^{(\alpha)}<0\right\}, & \mathrm{Zero}_\alpha = \left\{k\in[1\cdots n] \mid S_k^{(\alpha)}=0\right\} \\
\mathrm{Pos}_\beta = \left\{k\in[1\cdots n] \mid S_k^{(\beta)}>0\right\}, & \mathrm{Neg}_\beta = \left\{k\in[1\cdots n] \mid S_k^{(\beta)}<0\right\}, & \mathrm{Zero}_\beta = \left\{k\in[1\cdots n] \mid S_k^{(\beta)}=0\right\}
\end{array}$$

Let us denote, for $\gamma\in\{\alpha,\beta,g\}$, the size of the previously defined sets as follows:

$$p_\gamma := \left|\mathrm{Pos}_\gamma\right|, n_\gamma := \left|\mathrm{Neg}_\gamma\right|, z_\gamma := \left|\mathrm{Zero}_\gamma\right|.$$

As a result, we will obtain five forms of inequalities: $A\leq\alpha$, $A'\geq\alpha$, $B\leq\beta$, $B'\geq\beta$ and

$C \geq 0$ that characterise $\mathcal{S}_{(h_\alpha,h_\beta)}$. The inequalities 5.2 and 5.3 are equivalent to:

$$\forall j \in \mathrm{Pos}_g, \frac{-\varphi_j^{(g)}(v)}{S_j^{(g)}} \leq \alpha \tag{5.6}$$

$$\forall j \in \mathrm{Pos}_g, \frac{-\varphi_j^{(g)}(v)}{S_j^{(g)}} \leq \beta \tag{5.7}$$

$$\forall j \in \mathrm{Neg}_g, \frac{-\varphi_j^{(g)}(v)}{S_j^{(g)}} \geq \alpha \tag{5.8}$$

$$\forall j \in \mathrm{Neg}_g, \frac{-\varphi_j^{(g)}(v)}{S_j^{(g)}} \geq \beta \tag{5.9}$$

$$\forall j \in \mathrm{Zero}_g, \varphi_j^{(g)}(v) \geq 0 \tag{5.10}$$

The inequalities 5.1, 5.4 and 5.5 are equivalent to:

$$0 \leq \alpha, \alpha \leq \beta \tag{5.11}$$

$$\forall k \in \mathrm{Pos}_\alpha, \frac{-\varphi_k^{(\alpha)}(v\,[\mathcal{C}_r \leftarrow 0])}{S_k^{(\alpha)}} \leq \alpha \tag{5.12}$$

$$\forall k \in \mathrm{Neg}_\alpha, \frac{-\varphi_k^{(\alpha)}(v\,[\mathcal{C}_r \leftarrow 0])}{S_k^{(\alpha)}} \geq \alpha \tag{5.13}$$

$$\forall k \in \mathrm{Zero}_\alpha, \varphi_k^{(\alpha)}(v\,[\mathcal{C}_r \leftarrow 0]) \geq 0 \tag{5.14}$$

$$\forall k \in \mathrm{Pos}_\beta, \frac{-\varphi_k^{(\beta)}(v\,[\mathcal{C}_r \leftarrow 0])}{S_k^{(\beta)}} \leq \beta \tag{5.15}$$

$$\forall k \in \mathrm{Neg}_\beta, \frac{-\varphi_k^{(\beta)}(v\,[\mathcal{C}_r \leftarrow 0])}{S_k^{(\beta)}} \geq \beta \tag{5.16}$$

$$\forall k \in \mathrm{Zero}_\beta, \varphi_k^{(\beta)}(v\,[\mathcal{C}_r \leftarrow 0]) \geq 0 \tag{5.17}$$

We can eliminate α and β by writing, for every type of inequalities previously mentioned, $A \leq A'$, $A \leq B'$, $B \leq B'$ and $C \geq 0$. By doing so, we obtain $2\left(2c_i + c_g\right)^2$ linear inequalities. As a result, we can compute $\mathcal{S}_{(h_\alpha,h_\beta)}$ as a polyhedron with at most $2\left(2c_i + c_g\right)^2$ inequalities.

Step 2: Computing the range for the bounds α and β. In case $\mathcal{S}_{(h_\alpha,h_\beta)}$ is non-empty, we compute, for an arbitrary v of $\mathcal{S}_{(h_\alpha,h_\beta)}$, all the possible values of α and β that

indeed lead to h_α and h_β. The sets of such α and β are respectively denoted I^v_α and I^v_β and are defined as follows:

$$I^v_\alpha := \{\alpha \mid \alpha \geq 0, v + \alpha \models g, v + \alpha\,[\mathcal{C}_r \leftarrow 0] \in h_\alpha\}$$

and

$$I^v_\beta := \{\beta \mid \exists \alpha \in I^v_\alpha, \beta \geq \alpha, v + \beta \models g, v + \beta\,[\mathcal{C}_r \leftarrow 0] \in h_\beta\}$$

I^v_α and I^v_β are intervals. We compute them by computing the maximum and minimum of affine functions. Let us denote $I^v_\alpha\,(\text{low})$ (*resp.* $I^v_\beta\,(\text{low})$) the lower bound of I^v_α (*resp.* I^v_β). Let us also denote $I^v_\alpha\,(\text{up})$ (*resp.* $I^v_\beta\,(\text{up})$) the upper bound of I^v_α (*resp.* I^v_β). We can express them as follows:

$$I^v_\alpha\,(\text{low}) = \max\left(\left\{\frac{-\varphi^{(g)}_j\,(v)}{S^{(g)}_j} \mid j \in \text{Pos}_g\right\} \cup \left\{\frac{-\varphi^{(\alpha)}_k\,(v\,[\mathcal{C}_r \leftarrow 0])}{S^{(\alpha)}_k} \mid k \in \text{Pos}_\alpha\right\} \cup \{0\}\right)$$

$$I^v_\alpha\,(\text{up}) = \min\left(\left\{\frac{-\varphi^{(g)}_j\,(v)}{S^{(g)}_j} \mid j \in \text{Neg}_g\right\} \cup \left\{\frac{-\varphi^{(\alpha)}_k\,(v\,[\mathcal{C}_r \leftarrow 0])}{S^{(\alpha)}_k} \mid k \in \text{Neg}_\alpha\right\}\right)$$

$$I^v_\beta\,(\text{low}) = \max\left(\{I^v_\alpha\,(\text{low})\} \cup \left\{\frac{-\varphi^{(\beta)}_k\,(v\,[\mathcal{C}_r \leftarrow 0])}{S^{(\beta)}_k} \mid k \in \text{Pos}_\beta\right\} \cup \left\{\frac{-\varphi^{(g)}_j\,(v)}{S^{(g)}_j} \mid j \in \text{Pos}_g\right\}\right)$$

$$I^v_\beta\,(\text{up}) = \min\left(\left\{\frac{-\varphi^{(g)}_j\,(v)}{S^{(g)}_j} \mid j \in \text{Neg}_g\right\} \cup \left\{\frac{-\varphi^{(\beta)}_k\,(v\,[\mathcal{C}_r \leftarrow 0])}{S^{(\beta)}_k} \mid k \in \text{Neg}_\beta\right\}\right)$$

Let us explain how we compute $I^v_\alpha\,(\text{low})$: α is lower-bounded by the inequalities of the form $A \leq \alpha$ where A is an affine function, therefore we compute the maximum of all the affine functions A. We can apply the same reasoning for $I^v_\alpha\,(\text{up})\,, I^v_\beta\,(\text{low})$ and $I^v_\beta\,(\text{up})$.

As a result, $I^v_\alpha\,(\text{low})$, $I^v_\alpha\,(\text{up})\,, I^v_\beta\,(\text{low})$, $I^v_\beta\,(\text{up})$ are n-dimensional piecewise-affine functions, represented with a tiling of polyhedra with respectively at most $p_\alpha + p_g + 1$, $n_\alpha + n_g$, $1 + p_\alpha + p_\beta + 2p_g$ and $n_\beta + n_g$ cells. For each of these tilings of polyhedra, we can bound the number of constraints of the polyhedra respectively by $p_\alpha + p_g$, $n_\alpha + n_g - 1$, $p_\alpha + p_\beta + 2p_g$ and $n_\beta + n_g - 1$.

In order to get affine expressions for the bounds of I^v_α and I^v_β, we refine $\mathcal{S}_{(h_\alpha, h_\beta)}$ into cells on which one of the affine functions in the expressions of $I^v_\alpha\,(\text{low})$ (or $I^v_\alpha\,(\text{up})\,, I^v_\beta\,(\text{low})$ or $I^v_\beta\,(\text{up})$) realises the maximum (*resp.* minimum). This refinement is obtained by expressing the fact that the selected affine function is indeed larger than (*resp.* smaller than) all other

functions. This may refine $\mathcal{S}_{(h_\alpha,h_\beta)}$ into a tiling of polyhedra with at most $2 \cdot (c_i + c_g)^4$ cells, defined with the affine functions previously enumerated. Each polyhedron of this tiling of polyhedra is represented by at most $3\,(c_i + c_g)$ linear inequalities.

Step 3: Computing the optimal values for α and β. Let us fix v in one of the resulting refined polyhedra of $\mathcal{S}_{(h_\alpha,h_\beta)}$, that we denote h, such that the bounds of I^v_α and I^v_β are affine functions with respect to v. We let $\mathcal{D} = \Big\{(\alpha,\beta) \mid \alpha \in I^v_\alpha,\ \beta \in I^v_\beta,\ \alpha \leq \beta\Big\}$. It remains to find the optimal choices for α and β, i.e., the values that maximise the following function μ over $\mathcal{D}$:

$$\mu : \ (\alpha,\beta) \ \mapsto \ \min\left(\beta-\alpha, \mathcal{P}_i\left(\ell', v+\alpha\left[\mathcal{C}_r \leftarrow 0\right]\right), \mathcal{P}_i\left(\ell', v+\beta\left[\mathcal{C}_r \leftarrow 0\right]\right)\right)$$

over the set $\mathcal{D}$. Let us express the last two terms of this expression in terms of α and β. We denote these terms f_{h_α} and f_{h_β}:

$$f_{h_\alpha} : \ \alpha \ \mapsto \ \mathcal{P}_i\left(\ell', v+\alpha\left[\mathcal{C}_r \leftarrow 0\right]\right) \ ; f_{h_\beta} : \ \beta \ \mapsto \ \mathcal{P}_i\left(\ell', v+\beta\left[\mathcal{C}_r \leftarrow 0\right]\right)$$

f_{h_α} is constant with respect to β, and f_{h_β} is constant with respect to α. As $v \mapsto \mathcal{P}_i(\ell', v)$ and $v \mapsto \mathcal{P}_i(\ell', v)$ are multi-dimensional affine functions over h. Therefore f_{h_α} and f_{h_β} are affine functions respectively with respect to α and β. We can express them with inhomogeneous and homogeneous terms as follows:

$$\begin{aligned} f_{h_\alpha}\left(v+\alpha\left[\mathcal{C}_r \leftarrow 0\right]\right) &= F^{\mathcal{C}_r}_{h_\alpha} \cdot \alpha + f_{h_\alpha}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right) \\ f_{h_\beta}\left(v+\beta\left[\mathcal{C}_r \leftarrow 0\right]\right) &= F^{\mathcal{C}_r}_{h_\beta} \cdot \beta + f_{h_\beta}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right) \end{aligned}$$

$F^{\mathcal{C}_r}_{h_\alpha}$ is the sum of the coefficients corresponding to clocks that are not reset in $\mathcal{C}_r$:

$$F^{\mathcal{C}_r}_{h_\alpha} = f_{h_\alpha}(\mathbf{1}) - f_{h_\alpha}\left(\mathbf{1}_{\mathcal{C}_r}\right)$$

Similarly for $F^{\mathcal{C}_r}_{h_\beta}$. We then have the following equality.

$$\mu(\alpha,\beta) = \min\Big(\beta-\alpha, F^{\mathcal{C}_r}_{h_\alpha} \cdot \alpha + f_{h_\alpha}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right), F^{\mathcal{C}_r}_{h_\beta} \cdot \beta + f_{h_\beta}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right)\Big)$$

Our goal is to find the couple (α,β) in $\mathcal{D}$ that **maximises** $\mu(\alpha,\beta)$. Let us recall that v is **fixed**.

To find these values, we will use a result that we will state in the Theorem 5.9, presented in page 139, and prove in Subsection 5.3, for pedagogical purposes. The result will not tackle the cases where I_α^v and I_β^v are unbounded. Let us solve this optimisation problem here for this particular case using a constructive proof similar to, but simpler than, the proof of Theorem 5.9. To simplify the notations, let us denote for this special case $a = F_{h_\alpha}^{\mathcal{C}_r}, b = f_{h_\alpha}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right), c = F_{h_\beta}^{\mathcal{C}_r}$ and $d = f_{h_\beta}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right)$. **Let us suppose that I_α^v or I_β^v are infinite.** As their lower bounds are positive, that means the only bound that can be infinite is the upper bound. Let us write the intervals with their lower and upper bounds:

$$I_\alpha^v = [m_\alpha, M_\alpha], I_\beta^v = [m_\beta, M_\beta]$$

We will describe the optimal α and β for the three possibles cases:

- If both M_α and M_β are infinite, due to Lemma 4.6, f_{h_α} and f_{h_β} are constant and $\mu = \beta - \alpha$, the optimal choice is $\alpha = m_\alpha, \beta = M_\beta$.
- Otherwise, if $M_\alpha = +\infty$, then $M_\beta = +\infty$ too as $\alpha \leq \beta$ for any $(\alpha, \beta) \in I_\alpha^v \times I_\beta^v$. Due to Lemma 4.6, we can conclude that f_{h_α} and f_{h_β} are constant. Thus:

  $$\mu(\alpha, \beta) = \min\left(\beta - \alpha, a \cdot m_\alpha + b, c \cdot M_\beta + d\right).$$

  The optimal choices for α and β are respectively $\alpha = m_\alpha$ and $\beta = M_\beta$.
- Otherwise, if $M_\beta = +\infty$, due to Lemma 4.6, we can conclude that f_{h_β} is constant and that we can choose $\beta = M_\beta = +\infty$. Then:

  $$\begin{aligned}\mu(\alpha, M_\beta) &= \min\left(M_\beta - \alpha, a \cdot \alpha + b, c \cdot M_\beta + d\right)\\ &= a \cdot \alpha + b\end{aligned}$$

  The optimal choice of α is the smallest, *i.e.* $\alpha = m_\alpha$.

Let us go back to the general cases where I_α^v and I_β^v have no infinite bounds. We apply Theorem 5.9 with the following parameters:

- $[m_\alpha, M_\alpha] = I_\alpha^v$ and $[m_\beta, M_\beta] = I_\beta^v$,
- $a = F_{h_\alpha}^{\mathcal{C}_r}$ and $b = f_{h_\alpha}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right)$,
- $c = F_{h_\beta}^{\mathcal{C}_r}$ and $d = f_{h_\beta}\left(v\left[\mathcal{C}_r \leftarrow 0\right]\right)$.

This may again require refining the polyhedra being considered into at most 15 sub-polyhedra, since there may be up to 13 different cases for maximising $\mu(\alpha, \beta)$ (see the summary of the technical Theorem 5.9, in Subsection 5.3.5, page 164). Each of these polyhedra are represented with at most $3(c_g + c_i) + 5$ linear inequalities. We then get the optimal values for α and β, as well as the value of μ at that maximal point, depending on the signs of $F_{h_\alpha}^{\mathcal{C}_r}$ and $F_{h_\beta}^{\mathcal{C}_r}$. We can check that both the optimal choices for α and β, as well as the resulting permissiveness function, are affine functions of v. Indeed, in our instance of the problem of Theorem 5.9, a and b are constants, while c and d, and m_α, M_α, m_β and M_β are affine functions of v. The latter may only be multiplied by constants, and/or added with one another.

The coefficients of those affine functions can be computed from those of f_{h_α} and f_{h_β}, and from those of functions $\varphi_i^{(g)}, \varphi_j^{(\alpha)}, \varphi_j^{(\beta)}$ defining the guard g and the tiling of polyhedra of the piecewise-affine function $v \mapsto \mathcal{P}_{i+1}(\ell, v)$. In the worst case, the numerators are multiplied by the sum of all coefficients, and the denominators may be multiplied by the product of two sums of coefficients. In any case, the space needed to store one such function (assuming binary encoding) is at most affine in the space needed to store $v \mapsto \mathcal{P}_i(\ell', v)$. This concludes the third step of our computation.

Conclusion: Finalising the computation of $v \mapsto \mathcal{P}_{i+1}(\ell, v)$. Now, we have a collection of m^2 piecewise affine functions, associated with a couple of polyhedra in which the valuation may end up. Each function is a candidate expression for $v \mapsto \mathcal{P}_{i+1}(\ell, v)$. We thus have to refine one last time the partition we obtained, by considering sub-polyhedra where one of the m^2 candidate functions is the maximal one. Our resulting function is $v \mapsto \mathcal{P}_{i+1}(\ell, v)$. This proves that we can compute it as an n-dimensional piecewise affine function, represented with a tiling of polyhedra of at most $\mathcal{O}\left((c_i + c_g)^{4 \cdot m^2}\right)$ cells, such that each polyhedron can be represented with at most $3 \cdot m^2 ((c_i + c_g) + 5)$ linear inequalities. In the end, this proves that the function $v \mapsto \mathcal{P}_{i+1}(\ell, v)$ can be computed from $v \mapsto \mathcal{P}_i(\ell', v)$ in time $\mathcal{O}\left((c_i + c_g)^{4 \cdot m^2}\right)$. The coefficients of the affine functions defining $\mathcal{P}_{i+1}(\ell, \cdot)$ are polynomial in the coefficients of the affine functions defining $\mathcal{P}_i(\ell', \cdot)$.

For any location ℓ, $\mathcal{P}_0(\ell, \cdot)$ can be computed in constant time. As a result, the function $v \mapsto \mathcal{P}_{i+1}(\ell, v)$ can be computed in non-elementary time.

Conclusion: Computation of the permissiveness function.

As stated in Algorithm 1, it follows that, we obtain the permissiveness function in location ℓ as a piecewise-affine function in non-elementary time which proves Theorem 5.2.

This complexity is quite high, but it is a rough approximation. To illustrate our algorithm and have an example of the number of cells we can have, we develop in the next subsection a complete computation of the sequence of suboptimal permissiveness function $\mathcal{P}_i$ on the linear timed automaton of Figure 5.3 (which only differs from the example Figure 2.10 in the guard of the first transition). In this computation, we have many intermediary cases to handle, but the final function $\mathcal{P}_2$ in ℓ_0, depicted in Figure 5.4, has a tiling of polyhedra with only four cells (in the winning zone).

5.1.3 An example

In this section, we develop step-by-step the algorithm of Subsection 5.1.2 in order to compute $v \mapsto \mathcal{P}_2(\ell_0, v)$ of the timed automaton from Figure 5.3. This function corresponds to the permissiveness function for this automaton on location ℓ_0.

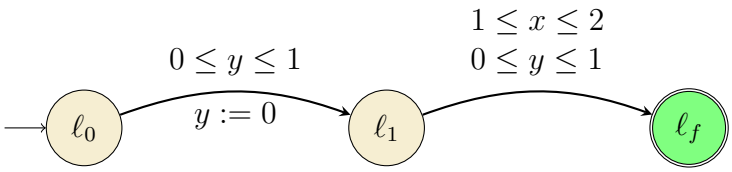


Figure 5.3: Automaton of Figure 2.10 where the guard on the first transition has been slightly extended.

As explained in Subsection 4.2.2, the following functions are equal:

$$\mathrm{Perm}(\ell_f, \cdot) = \mathcal{P}_0(\ell_f, \cdot), \mathrm{Perm}(\ell_1, \cdot) = \mathcal{P}_1(\ell_1, \cdot), \mathrm{Perm}(\ell_0, \cdot) = \mathcal{P}_2(\ell_0, \cdot)$$

Let us remember that $\mathcal{P}_1(\ell_1, \cdot)$ and $\mathcal{P}_0(\ell_0, \cdot)$ have already been computed in the example of Subsection 2.2.2. See Figure 2.11a for the permissiveness function at ℓ_1. Here we detail the computation of $\mathcal{P}_2(\ell_0, \cdot)$. Following the proof of Lemma 5.3, we list the pairs of possible cells where the automaton may enter ℓ_1 after delays α and β when the player proposes the interval $[\alpha, \beta]$: since the transition to ℓ_1 resets y, we have the two following

possible cells[2]:

$$h_0 = \{(x,0) \mid 0 \leq x \leq 1\} \text{ and } h_1 = \{(x,0) \mid 1 \leq x \leq 2\}.$$

Hence, the successors $(\ell_1, v + \alpha\,[y \leftarrow 0])$ and $(\ell_1, v + \beta\,[y \leftarrow 0])$ can be in four possible cases:

1. Both $v + \alpha\,[y \leftarrow 0]$ and $v + \beta\,[y \leftarrow 0]$ in h_0;
2. Both $v + \alpha\,[y \leftarrow 0]$ and $v + \beta\,[y \leftarrow 0]$ in h_1;
3. $v + \alpha\,[y \leftarrow 0] \in h_0$ and $v + \beta\,[y \leftarrow 0] \in h_1$;

Indeed, $v+\alpha\,[y \leftarrow 0] \in h_1$ and $v+\beta\,[y \leftarrow 0] \in h_0$ cannot happen since α should be smaller than β.

Step 1: Computing the entry sets $\mathcal{S}_{(h_\alpha,h_\beta)}$.

For each pair, we begin by computing the set of valuations v for which there are values $0 \leq \alpha \leq \beta$ satisfying the conditions. We call the following set of valuations for $h_\alpha, h_\beta \in \{h_0, h_1\}$:

$$\mathcal{S}_{(h_\alpha,h_\beta)} = \{v \mid \exists \alpha, \beta, 0 \leq \alpha \leq \beta, v+\alpha, v+\beta \models g, v+\alpha\,[y \leftarrow 0] \in h_\alpha, v+\beta\,[y \leftarrow 0] \in h_\beta\}$$

1. Having $v + \alpha\,[y \leftarrow 0]$ and $v + \beta\,[y \leftarrow 0]$ in h_0 can be written as: there exist $\alpha \leq \beta$ such that :
$$\begin{cases} 0 \leq v(y) + \alpha \leq 1, & 0 \leq v(y) + \beta \leq 1 \\ 0 \leq v(x) + \alpha \leq 1, & 0 \leq v(x) + \beta \leq 1 \end{cases}$$
The constraints on y come from the guard of the transition, while those on x correspond to having the target valuations in h_0. In this simple case, the quantifier elimination returns:

$$\mathcal{S}_{(h_0,h_0)} = \{(x,y) \mid 0 \leq x \leq 1 \text{ and } 0 \leq y \leq 1\}.$$

[2]For convenience in this 2-clock example, we may write valuations either as v or as pairs (x, y), depending on the situation.

2. Having $v+\alpha\,[y \leftarrow 0]$ and $v+\beta\,[y \leftarrow 0]$ in h_1 translates to: there exist α and β such that $\alpha \leq \beta$ and:

$$\begin{cases} 0 \leq v\,(y)+\alpha \leq 1, & 0 \leq v\,(y)+\beta \leq 1 \\ 1 \leq v\,(x)+\alpha \leq 2, & 1 \leq v\,(x)+\beta \leq 2 \end{cases}$$

   This results in:

$$\mathcal{S}_{(h_1,h_1)} = \{(x,y) \mid 0 \leq x \leq 2 \text{ and } 0 \leq y \leq 1 \text{ and } y \leq x\}$$

3. The case where $v+\alpha\,[y \leftarrow 0] \in h_0$ and $v+\beta\,[y \leftarrow 0] \in h_1$ translates to: there exist α and β such that $\alpha \leq \beta$ and:

$$\begin{cases} 0 \leq v\,(y)+\alpha \leq 1, & 0 \leq v\,(y)+\beta \leq 1 \\ 0 \leq v\,(x)+\alpha \leq 1, & 1 \leq v\,(x)+\beta \leq 2 \end{cases}$$

   This results in:

$$\mathcal{S}_{(h_0,h_1)} = \{(x,y) \mid 0 \leq x \leq 1 \text{ and } 0 \leq y \leq 1 \text{ and } y \leq x\}$$

The next step of the algorithm consists in computing the set of possible α and β corresponding to the cases where v belongs to either $\mathcal{S}_{(h_0,h_0)}$, $\mathcal{S}_{(h_1,h_1)}$ or $\mathcal{S}_{(h_0,h_1)}$. For any valuation v, this set of possible α (*resp.* β) is denoted I^v_α (*resp.* I^v_β).

Step 2: Computing the range for the bounds α and β.

We now compute the intervals of possible values for α and β: this just amounts to writing the conditions to have $v+\alpha$ satisfying the guard and $v+\alpha\,[y \leftarrow 0]$ belonging to the target cell (the computation is identical for β):

- Having $v+\alpha\,[y \leftarrow 0]$ ending up in h_0 requires $\alpha \in [0, 1-v\,(x)] \cap [0, 1-v\,(y)]$;
- Having $v+\alpha\,[y \leftarrow 0]$ ending up in h_1 requires $\alpha \in [1-v\,(x)\,, 2-v\,(x)] \cap [0, 1-v\,(y)]$.

We end up with the following situations:

- Having both $v+\alpha\,[y \leftarrow 0]$ and $v+\beta\,[y \leftarrow 0]$ in h_0 can be performed from $\mathcal{S}_{(h_0,h_0)}$. From that zone:

1. If $x \leq y$, we have $I^v_\alpha = I^v_\beta = [0, 1-y]$ **(case 1)**,
2. If $x \geq y$, we have $I^v_\alpha = I^v_\beta = [0, 1-x]$ **(case 2)**.

▷ Having both $v + \alpha\,[y \leftarrow 0]$ and $v + \beta\,[y \leftarrow 0]$ in h_1 can be performed from $\mathcal{V}_{1,1}$. From that zone:

1. If $x \leq 1$ and $y \leq x$, we have $I^v_\alpha = I^v_\beta = [1-x, 1-y]$ **(case 3)**,
2. If $1 \leq x \leq 1+y$, we have $I^v_\alpha = I^v_\beta = [0, 1-y]$ **(case 4)**,
3. If $1+y \leq x \leq 2$, we have $I^v_\alpha = I^v_\beta = [0, 2-x]$ **(case 5)**.

▷ Having $v + \alpha\,[y \leftarrow 0] \in h_0$ and $v + \beta\,[y \leftarrow 0] \in h_1$ can be performed from $\mathcal{S}_{(h_0,h_1)}$. We then have $I^v_\alpha = [0, 1-x]$ and $I^v_\beta = [1-x, 1-y]$ **(case 6)**.

Step 3: Computing the optimal values of α and β.

In the last subsection, we have enumerated six different cases. In this section, we will compute the optimal interval to propose for each case. To do so, we will use the result of Section 5.3. In this section, we compute the optimal α and β that maximise the quantity $\min(\beta - \alpha, a \cdot \alpha + b, c \cdot \beta + d)$ when a, b, c, d do not depend on α and β.

1. In **case** 1, we have to maximise $(\alpha, \beta) \mapsto \min(\beta - \alpha, x + \alpha, x + \beta)$ over $\{(\alpha, \beta) \mid \alpha, \beta \in [0, 1-y], \alpha \leq \beta\}$. This corresponds to the case '$a \geq 0$ and $c \geq 0$' of Theorem 5.9. We get:
   - ▷ If $\dfrac{1-y-x}{2} \leq 0$, the optimal interval for the player is $[0, 1-y]$, yielding permissiveness $1-y$.
   - ▷ If $0 \leq \dfrac{1-y-x}{2}$, the optimal interval is $\left[\dfrac{1-y-x}{2}, 1-y\right]$, with permissiveness $\dfrac{1-y+x}{2}$.
2. In **case** 2, we maximise the same function over $\{(\alpha, \beta) \mid \alpha, \beta \in [0, 1-x], \alpha \leq \beta\}$. The situation is the same as above, and we get:
   - ▷ If $\dfrac{1}{2} - x \leq 0$, the optimal interval for the player is $[0, 1-x]$, yielding permissiveness $1-x$.
   - ▷ If $0 \leq \dfrac{1}{2} - x$, the optimal interval is $\left[\dfrac{1}{2} - x, 1-x\right]$, with permissiveness $\dfrac{1}{2}$.

3. In **case** 3, we maximise $(\alpha, \beta) \mapsto \min(\beta - \alpha, 2 - (x + \alpha), 2 - (x + \beta))$ over $\{(\alpha, \beta) \mid \alpha, \beta \in [1 - x, 1 - y], \alpha \leq \beta\}$. We apply Theorem 5.9, with $a \leq 0$ and $c \leq 0$:
   - ▷ The first condition corresponds to $x \leq \frac{1}{2} + y$; then the maximal point is $\min(x - y, 1)$, i.e. $x - y$, and is reached at $(1 - x, 1 - y)$.
   - ▷ The second condition is $x \geq \frac{1}{2} + y$, for which the maximal value is $\frac{1}{2}$ is reached at $\left(1 - x, \frac{3}{2} - x\right)$.
4. In **case** 4, we maximise the same function over $\{(\alpha, \beta) \mid \alpha, \beta \in [0, 1 - y], \alpha \leq \beta\}$ over the zone $(1 \leq x \leq 1 + y \leq 2)$:
   - ▷ The first condition is $y \geq \frac{x}{2}$: in that zone, the maximal point is $1 - y$, reached for $(0, 1 - y)$;
   - ▷ The second condition is $y \leq \frac{x}{2}$, and the maximal value $1 - \frac{x}{2}$ is reached at $\left(0, 1 - \frac{x}{2}\right)$.
5. In **case** 5 we maximise the same function over $\{(\alpha, \beta) \mid \alpha, \beta \in [0, 2 - x], \alpha \leq \beta\}$. Again, the second condition holds, and the maximal value is $1 - \frac{x}{2}$, reached at $\left(0, 1 - \frac{x}{2}\right)$.
6. Finally, in **case** 6, we have to maximise $(\alpha, \beta) \mapsto \min(\beta - \alpha, x + \alpha, 2 - x - \beta)$ over $\{(\alpha, \beta) \mid \alpha \in [0, 1 - x], \beta \in [1 - x, 1 - y], \alpha \leq \beta\}$. Hence we are in the case $a > 0$ and $c < 0$ of Theorem 5.9. This case is divided into 11 cases (see Table 5.5 in page 167). Among them, the only cases whose conditions can hold are cases $1, 4$ and 5. Then:
   - ▷ When $y \geq 1 - x$ and $y \geq \frac{x}{2}$, then the condition of case 1 holds, and the maximal value $1 - y$ is reached at $(0, 1 - y)$.
   - ▷ The conditions of case 4 rewrites as $y \geq x - \frac{1}{3}$ and $y \leq 1 - x$. For those points, the maximal value is $\frac{1 + x - y}{2}$, reached at $\left(\frac{1 - x - y}{2}, 1 - y\right)$.
   - ▷ The conditions of case 5 are $x \geq \frac{2}{3}$ and $y \leq \frac{x}{2}$. Thus the maximal point $1 - \frac{x}{2}$ is reached for $\left(0, 1 - \frac{x}{2}\right)$.

The permissiveness function at ℓ_0

By superimposing those results and taking the maxima on cells where several solutions have been computed, we get the global permissiveness function depicted on Figure 5.4.

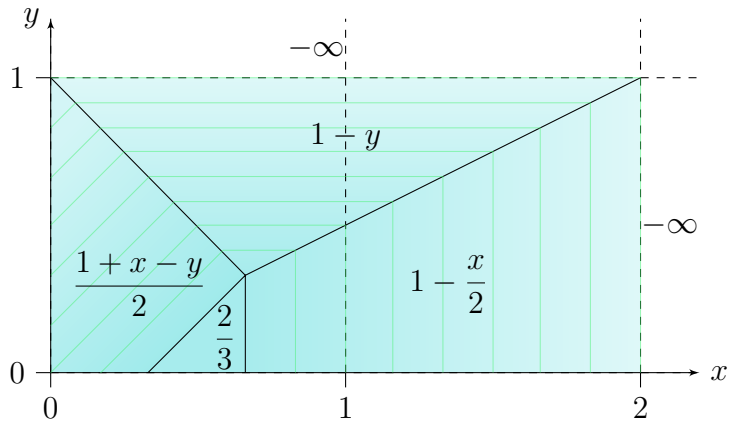


Figure 5.4: A linear timed automaton and its permissiveness at ℓ_0.

5.2 Extensions

In this section, we explain how to extend our algorithm to more general timed automata.

5.2.1 Non-necessarily closed p-moves or guards

Our algorithm currently only applies to closed guards (classical or polyhedral) and closed p-moves, as defined in Definitions 2.11 and 2.1 end 2.2, where non-necessary closed guards and p-moves have also been defined in the definition or in remarks (see Remark 2.2.1). For the purpose of simplification, we have restricted our permissive semantics and algorithm to closed guard and p-moves. Nevertheless the permissiveness can be defined for non-necessary closed guards and p-moves, since the size of the intervals $]\alpha, \beta[$, $[\alpha, \beta[$ and $[\alpha, \beta]$ are equal. We detail here whether non-necessary closed p-moves and guards can be considered in our algorithm and what changes it requires.

Non-necessary closed intervals

As the size of the closed or open intervals is the same, the only step where a non-necessary closed interval changes the algorithm is for the computation of the **strategy of the**

opponent. If the player proposes the interval $]\alpha, \beta[$, the opponent cannot choose the delay α or β, and this might be its optimal choice.

To tackle this issue, let us recall that by Definition 2.21, the permissiveness corresponds to the size of one of the proposed intervals during a run.

- If the permissiveness of a fixed configuration is equal to 0, then no open interval could have been proposed and the optimal choice of the player is to propose a **closed one**.

- Otherwise, if the optimal delay of the opponent is α or β, for a closed interval, and if an open interval $]\alpha, \beta[$ is proposed, we can **approach** the optimal delay with $\alpha + \varepsilon$ or $\beta - \varepsilon$ for a sufficiently small ε, as the permissiveness is continuous. The resulting (approached) permissiveness is equal to $|I| - 2\varepsilon$, if the optimal interval was open. We then can obtain the permissiveness by tending ε to 0.

Let us remark that, if the guards are closed, the player can propose a closed or an open interval without changing the permissiveness. It is then useless to propose a closed interval for the player, unless the guards are not closed guards.

Non-necessary closed guards

For any timed automaton $\mathcal{A}$ that contains open guards, we denote $\mathcal{C}(\mathcal{A})$ its associated timed automaton where all guards become closed (*i.e.* all $<$ are replaced by $\leq$ and all $>$ are replace by $\geq$).

If non-necessary closed guards are allowed, guards can admit constraints of the form $l_x < v(x) \leq u_x$, or $l_x \leq v(x) < u_x$, or $l_x < v(x) < u_x$ for classical guards, or non necessarily closed polyhedra for polyhedral guards. These are still polyhedra (but non-necessary closed ones). This can constrain the player to propose an open interval instead of a closed interval. The steps of the algorithm will not be affected, as explained in the previous subsection.

Nevertheless, for any configuration (ℓ, v) such that the permissiveness of (ℓ, v) is 0, a punctual interval had to be proposed. Therefore there might be issues on the corresponding guard of this punctual interval.

Therefore, accepting non-necessary closed guards might be an issue on these points, that we consider as future work for this algorithm.

5.2.2 Acyclic timed automata

We extend the previous study to the case of acyclic timed automata (with branching). In that case, we can still apply our inductive approach, with a few changes: at each step, we would compute the optimal p-move of the player for each single action, and then select the optimal action by 'superimposing' the resulting permissiveness functions and selecting the action that maximises permissiveness. This however breaks the result of Proposition 4.15: the maximum of two concave functions need not be concave. Example 5.2.1 displays an example where the permissiveness function is not concave.

Example 5.2.1 *Consider the automaton of Figure 5.5. The transition from ℓ_0 to ℓ_f has the same constraints as that from ℓ_1 to ℓ_f. Hence the permissiveness offered by that action is the same as the one from ℓ_1, which we already computed. Hence the global permissiveness from ℓ_0 is the (pointwise) maximal of the two piecewise-linear functions displayed on Figure 5.6. On this diagram, the blue area corresponds to points from where it is better (or only possible) to go via ℓ_1, while the red area corresponds to valuations from where it is better (or only possible) to take the bottom transition.*

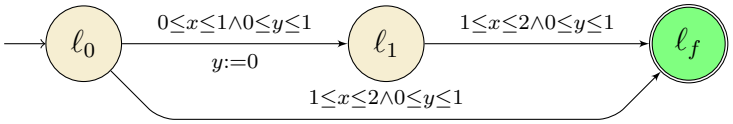


Figure 5.5: An acyclic timed automaton.

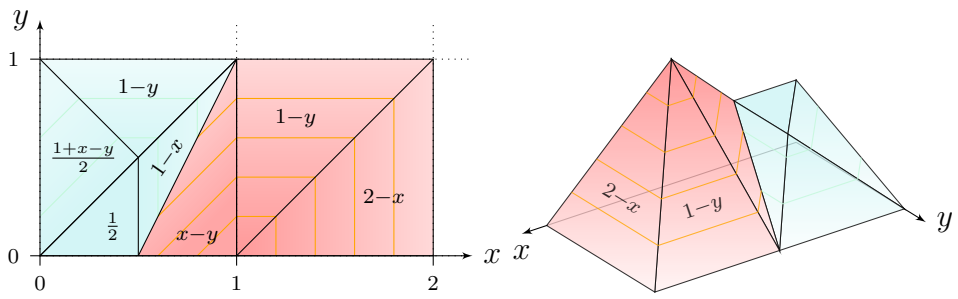


Figure 5.6: Permissiveness function for location ℓ_0 of the timed automaton of Figure 5.5.

We prove by induction that the permissiveness functions still are piecewise-linear in that setting. Hence all four steps of our proof of Lemma 5.3 still apply, with some adaptations:

- For the first step, we again consider two cells h_α and h_β, together with a set H of cells that may be visited between h_α and h_β. Again applying Fourier-Motzkin, we get a polyhedron $S_{(h_\alpha,h_\beta,H)}$ of valuations from which those cells can indeed be visited.

- The computation of the intervals I^v_α and I^v_β is unchanged.

- For each cell $h \in H$, we can compute the values $d_h^{\mathbf{in}}$ and $d_h^{\mathbf{out}}$ for which $v+d_h^{\mathbf{in}}\,[\mathcal{C}_r \leftarrow 0]$ enters h and $v + d_h^{\mathbf{out}}\,[\mathcal{C}_r \leftarrow 0]$ leaves h (notice that this may require further refinement of the polyhedron being considered). Since $\mathcal{P}_i$ is linear on cell h, it reaches its maximum on this cell either at $v + d_h^{\mathbf{in}}\,[\mathcal{C}_r \leftarrow 0]$ or at $v + d_h^{\mathbf{out}}\,[\mathcal{C}_r \leftarrow 0]$. The function we need to maximise now looks like:

$$\mu' \colon (\alpha,\beta) \mapsto \min(\{\beta-\alpha, \mathcal{P}_i\left(\ell', v+\alpha\,[\mathcal{C}_r \leftarrow 0]\right), \mathcal{P}_i\left(\ell', v+\beta\,[\mathcal{C}_r \leftarrow 0]\right) \cup \\ \mathcal{P}_i\left(\ell', v+d_h^{\mathbf{in}}\,[\mathcal{C}_r \leftarrow 0]\right), \mathcal{P}_i\left(\ell', v+d_h^{\mathbf{out}}\,[\mathcal{C}_r \leftarrow 0]\right), \mid h \in H\}).$$

  Now, we notice that all values in the second set are constant, not depending on α and β. We can thus still apply Theorem 5.9 in order to maximise $\mu(\alpha,\beta)$, and then take the above constants into account (which may again refine the polyhedra).

- The above three steps have to be performed for all outgoing transitions from the location ℓ being considered. The last step still consists in selecting the maximum of all the resulting functions. This step may refine again the polyhedra.

The complexity of our procedure is much higher than that of linear automata: because we consider sets of cells already at the first step, we may end up with $\mathcal{P}_{i+1}$ having more than 2^m cells, where m is the number of cells of $\mathcal{P}_i$. Hence our procedure is non-elementary in the worst case. In the end:

Theorem 5.4

The permissiveness function for acyclic timed automata is piecewise-linear. It can be computed in non-elementary time.

5.2.3 Acyclic timed games

5.2.3.1 Timed games

Timed games extend the notion of timed automata with a player, denoted $\mathcal{C}$, and an opponent, denoted $\mathcal{U}$. We present the definition of *two-player turn-based timed games*. In this model, some locations are controlled by a player, denoted $\mathcal{C}$, and the others by an opponent, denoted $\mathcal{U}$. We define timed games and their semantics in this subsection.

Definition 5.5: Timed games

A timed game is a tuple $\mathcal{G}_{\mathcal{C},\mathcal{U}} = (\Sigma, Q, Q_0, Q_f, Q_{\mathcal{C}}, Q_{\mathcal{U}}, \mathcal{C}, E)$ where:

- $\mathcal{A} = (\Sigma, Q, Q_0, Q_f, \mathcal{C}, E)$ is a timed automaton.
- The set of locations Q is partitioned into $Q_{\mathcal{C}}$ and $Q_{\mathcal{U}}$ where:
  - $Q_{\mathcal{C}}$ is the subset of *player-controlled* locations.
  - $Q_{\mathcal{U}}$ is the subset of *opponent-controlled* locations.
- For each location controlled by the opponent, a guard, called *invariant* is added[a].

The definitions of transitions and closed timed games naturally extend from the ones in Definition 2.4.

[a] that prevents the opponent to stay in a location forever.

As in timed automata, a *configuration* of a timed game $\mathcal{G}_{\mathcal{C},\mathcal{U}}$ is also a pair (ℓ, v) where $\ell \in Q$ is a location and v is a clock valuation. The permissive semantics of turn-based timed games is defined as follows, for each configuration (ℓ, v):

- if $\ell \in Q_{\mathcal{C}}$, then the player must propose a pair (I, a) such that there exists a transition $e = (\ell, g, a, \mathcal{C}_r, \ell')$ such that for any delay δ in I, $v + \delta \models g$. Then, the opponent proposes a delay δ in I and we reach the configuration $(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0])$.
- If $\ell \in Q_{\mathcal{U}}$, the player cannot propose any interval, delay or action and it is the opponent's turn to propose a pair (δ, a) under the same conditions. In that case, we reach the configuration $(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0])$ too.

As in timed automata semantics, we can define a transition $(\ell, v) \xrightarrow{\delta,a} (\ell', v')$ with a delay-elapsing transition and an action transition. The definition of p-run does not change, as

we can write p-run as $\rho^{(p)} = ((\ell_i, v_i), (I_i, a_i), \delta_i, (\ell_{i+1}, v_{i+1}))_{0 \leq i \leq n-1}$ by writing $I_i = \{\delta_i\}$ when $\ell_i \in Q_{\mathcal{U}}$. The permissiveness of a run is then only considering p-moves proposed by the player.

The definition of the strategy of the player slightly change. Let us formally define it in Definition 5.6.

Definition 5.6: Strategy, winning strategy

A permissive strategy is a (partial) function σ that maps finite p-runs $\rho^{(p)} = ((\ell_i, v_i), (I_i, a_i), \delta_i, (\ell_{i+1}, v_{i+1}))_{0 \leq i \leq n-1}$ such that $\ell_n \in Q_{\mathcal{C}}$ to p-moves in $\mathsf{p\text{-}moves}(\ell_n, v_n)$. We can say that:

- A p-run $\rho^{(p)} = ((\ell_i, v_i), (I_i, a_i), \delta_i, (\ell_{i+1}, v_{i+1}))_{0 \leq i \leq n-1}$ is **compatible** with a permissive strategy σ if for all its prefixes $\rho^{(p)}_{\leq j}$ such that $j \leq n$ and $\ell_j \in Q_{\mathcal{C}}$, $\sigma\left(\rho^{(p)}_{\leq j}\right) = (I_j, a_j)$. We say that $\rho^{(p)}$ is an outcome from the initial configuration s_0. The set of outcomes from a configuration is denoted $\mathrm{Out}(s_0, \sigma)$.
- A permissive strategy σ is **winning** from a given configuration s_0 if any infinite p-run originating from s_0, that is compatible with σ, is winning.

Example 5.2.2 *Let us consider the timed game of Figure 5.7. The state controlled by the player are represented with a circle state and the states controlled by the opponent with squares. In this example, the locations controlled by the player are* $\{\ell_0, \ell_1, \ell_f\}$ *and the one controlled by the opponent is* $\{\ell_2\}$.

If we consider the next example, in Figure 5.8, the set of locations controlled by the opponent is $\{\ell_1\}$. *In this example, the player can propose, from the configuration* $(\ell_0, (0, 0))$, *the following winning strategy: proposing* $(a_0, [1.5, 2])$. *In that case, whatever the opponent chooses, the player reaches a configuration* $(\ell_1, (\delta, 0))$, *where* $\delta \in [1.5, 2]$. *The opponent has to choose a delay between* 0 *and* 0.5, *because of the constraints on* x. *Then, the reached configuration is* $(\ell_2, (\delta + \delta', \delta'))$ *where* $\delta' \in [0, 0.5]$, *and the player can propose the p-move* $(a_2, [0, 0.5])$.

5.2.3.2 Extension of our algorithm

We finally extend our approach to (acyclic) two-player turn-based timed games as defined in Subsection 5.2.3.1. This setting is easily seen to preserve piecewise-affine character of the permissiveness function. Indeed, in order to compute $\mathcal{P}_{i+1}$ in a location ℓ belonging to

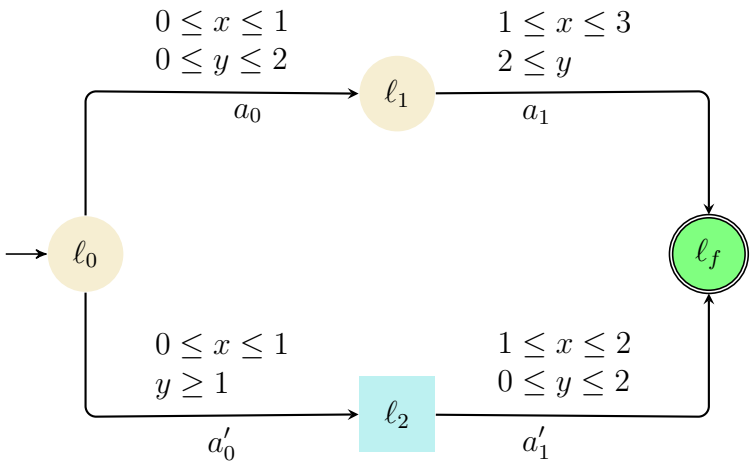


Figure 5.7: A two-clock acyclic timed games.

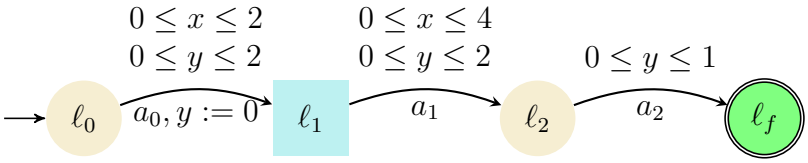


Figure 5.8: A two-clock linear timed games.

the opponent, it suffices to first compute the functions $\mathcal{P}_i^{\ell \to \ell'}$ for all outgoing transitions from ℓ to some ℓ'. This follows the same procedure as above, and results in a piecewise-linear function, assuming (inductively) that $\mathcal{P}_i$ is piecewise-linear. We then compute the (still piecewise-linear) minimum $\mathcal{M}_{i+1}(\ell, v)$ of all those functions, and finally:

$$\mathcal{P}_{i+1}(\ell, v) = \min_{\substack{d \text{ s.t.} \\ v+d \models \mathbf{Inv}(\ell)}} \mathcal{M}_{i+1}(\ell, v+d)$$

which is easily computed and remains piecewise-linear. It follows:

Theorem 5.7

The permissiveness function for acyclic turn-based timed games is piecewise-linear, and can be computed in non-elementary time.

5.3 Optimisation of the minimum of affine functions

In Subsection 5.1.2, we presented an algorithm that computes the permissiveness function and the sequence of suboptimal permissive functions. This algorithm uses in step

3 an optimisation result that we will develop in this section. The optimisation problem asks to maximise the minimum of three 2-dimensional affine functions. Let us recall this problem more formally. Let us fix a valuation v. Let us consider two intervals, I^v_α and I^v_β, with positive lower bounds, a location ℓ' and a set of resets $\mathcal{C}_r$. Let us consider a domain $\mathcal{D} = \left\{(\alpha, \beta) \mid \alpha \in I^v_\alpha, \beta \in I^v_\beta,\ \alpha \leq \beta\right\}$. Let us assume that $v \mapsto \mathcal{P}_i\left(\ell', v\left[\mathcal{C}_r \leftarrow 0\right]\right)$ (*resp.* $v, \mapsto \mathcal{P}_i\left(\ell', v\left[\mathcal{C}_r \leftarrow 0\right]\right)$) is an affine function when $v \in h_\alpha$ (*resp.* $v \in h_\beta$). The optimisation problem asks to find the values of α and β in $\mathcal{D}$ that maximise the following function:

$$(\alpha, \beta) \mapsto \min\left(\beta - \alpha, \mathcal{P}_i\left(\ell', v + \alpha\left[\mathcal{C}_r \leftarrow 0\right]\right), \mathcal{P}_i\left(\ell', v + \beta\left[\mathcal{C}_r \leftarrow 0\right]\right)\right) .$$

We can consider in this section that I^v_α and I^v_β are **bounded closed** intervals. Indeed, the unbounded case was treated in our algorithm in Subsection 5.1.2.

To solve this optimisation problem, we consider here a more **general optimisation problem**. Instead of considering the minimum of an interval and future sequence of suboptimal permissive functions, we consider here the minimum, denoted μ, of $\beta - \alpha$ and two arbitrary affine functions (in one dimension), one with the variable α and one with the variable β. The minimum of these three functions is a 2-dimensional piecewise-affine functions. The domain $\mathcal{D}$ of this function μ will keep as a condition that $\alpha \leq \beta$.

The section is organised as follows:

- In Subsection 5.3.1, we present the general optimisation problem and its resolution. We then split the proof of this resolution on three subsections. There can be three cases: either one of the function has infinite coefficients, or μ is, or is not, monotonic with at least one variables.

- In Subsection 5.3.2, we detail the proof when μ is monotonic with respect to at least one variable.

- In Subsection 5.3.3, we detail the proof when one of the two generic affine functions has infinite coefficients.

- In Subsection 5.3.4, we detail the proof when μ is not monotonic in any variable.

5.3.1 Optimisation problem and results

Let us consider four reals a, c, b, d and three 2-dimensional linear functions:

$$f : (\alpha, \beta) \mapsto a \cdot \alpha + b \; ; g : (\alpha, \beta) \mapsto c \cdot \beta + d \; ; h : (\alpha, \beta) \mapsto \beta - \alpha$$

$\mu = \min(h, f, g)$ is the pointwise minimum between these three functions over $\mathbb{R}^2$. The goal is to compute a couple (α, β) that maximises μ, under linear constraints over α and β.

Definition 5.8: Optimisation problem

Given four reals $m_\alpha, M_\alpha, m_\beta, M_\beta$ from $\mathbb{R}$ such that $m_\alpha \leq m_\beta$, $m_\alpha \leq M_\alpha$, $m_\beta \leq M_\beta$, $M_\alpha \leq M_\beta$, the optimisation problem asks to compute an argsup of μ over the domain $\mathcal{D} = \left\{(\alpha, \beta) \in \mathbb{R}^2 | \alpha \leq \beta, m_\alpha \leq \alpha \leq M_\alpha, m_\beta \leq \beta \leq M_\beta\right\}$ and the value of μ on this point.

Example 5.3.1 *Let* $f : (\alpha, \beta) \mapsto -\alpha + 1$ *and* $g : (\alpha, \beta) \mapsto \beta + \frac{1}{2}$ *and* $\mathcal{D} = [0, 5] \times [4, 6]$. *Then* $\mu(\alpha, \beta) = \min\left(\beta - \alpha, -\alpha + 1, \beta + \frac{1}{2}\right)$ *is a monotonic function.* μ *is non-decreasing with respect to* β *and decreasing with respect to* α. *As a result* μ *is maximised at* $(0, 6)$.

In the following Theorem 5.9, we give the general result for this optimisation problem: the maximum of μ can take four different forms. The proof we provide in the next subsection is a **constructive** proof and the value of the maximum and the coordinate of a maximal point are given in Subsection 5.3.5.

Theorem 5.9

Let $m_\alpha \leq M_\alpha$, $m_\beta \leq M_\beta$, $M_\alpha \leq M_\beta$ and $m_\alpha \leq m_\beta$ be reals from $\mathbb{R}$. Let $\mathcal{D} = \left\{(\alpha, \beta) \in \mathbb{R}^2 | \alpha \leq \beta, m_\alpha \leq \alpha \leq M_\alpha, m_\beta \leq \beta \leq M_\beta\right\}$ be the domain of the variables α and β. The maximum of μ over $\mathcal{D}$ exists and may take one of the four following forms:

$$M_\beta - m_\alpha, \lambda \cdot f(\nu), \lambda \cdot g(\nu) \text{ and } \frac{b \cdot c - d \cdot a}{c - a}$$

where $\lambda \in \left\{1, \frac{1}{1 - c}, \frac{1}{1 + a}\right\}$ and $\nu \in \{m_\alpha, M_\alpha, m_\beta, M_\beta\}$.

Description of the domain $\mathcal{D}$

The domain $\mathcal{D}$ can be drawn as a 2−dimensional polyhedron in Figure 5.9 where $A = (m_\alpha, M_\beta), B = (M_\alpha, M_\beta), C = (m_\alpha, m_\beta), D = (\min(M_\alpha, m_\beta), m_\beta), E = (M_\alpha, \max(M_\alpha, m_\beta))$. Let us remark that B, C, D and E are not necessarily all distinct. For instance if $M_\alpha \leq m_\beta$, then $D = E$.

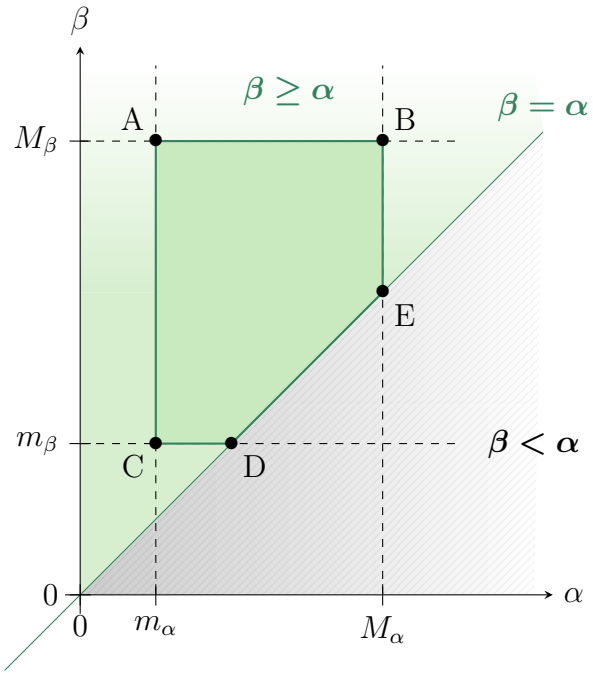


Figure 5.9: The definition set $\mathcal{D}$ when C, D, E and B are all distinct.

Applications for the computation of the permissiveness function

This theorem and the results of its constructive proof in Subsection 5.3.5 are used in the algorithm presented in Subsection 5.1. In this algorithm, the constraints m_α, M_α and m_β, M_β correspond to the lower and upper bounds of the possible values of α and β when the player proposes an interval $[\alpha, \beta]$. f and g correspond respectively to $\mathcal{P}_i(\ell', v + \alpha[\mathcal{C}_r \leftarrow 0])$ and $\mathcal{P}_i(\ell', v + \beta[\mathcal{C}_r \leftarrow 0])$.

Existence of the maximum

First, let us prove that $\mathcal{D}$ is not empty and that the arg sup of μ over $\mathcal{D}$ exists. As $m_\alpha \leq M_\alpha \leq M_\beta$, $\mathcal{D}$ is non-empty and compact. μ is a continuous piecewise affine function

over a compact, so $\arg\sup_{(\alpha,\beta)\in\mathcal{D}} \mu(\alpha,\beta)$ exists.

Let us recall that the notation $\mu_{|\mathcal{D}}$ denotes the restriction of the function μ on $\mathcal{D}$, while μ is defined on $\mathbb{R}^2$.

Organisation of the proof

There are two possible cases for f and g. Either f or g has infinite coefficients and we must then consider that $f = \pm\infty$ (*resp.* $g = \pm\infty$). Or f and g are both finite affine functions and they can then be expressed using real coefficients $b, a, d, c \in \mathbb{R}$. Depending on the sign of their homogeneous terms a and c, μ can be monotonic in at least one variable, in which case its optimisation can be reduced to the optimisation of a one-variable affine function. Therefore, we treat the case where μ is monotonic separately from the case where μ is a non-monotonic function. We thus divide the proof into three cases:

1. In Subsection 5.3.2, we detail the proof when μ **is monotonic with respect to at least one variable** and when f and g are finite affine functions. This is the case when $a \leq 0 \wedge c \geq 0$ or $a \geq 0 \wedge c \geq 0$, or $a \leq 0 \wedge c \leq 0$.

2. In Subsection 5.3.3, we detail the proof when f **or** g **has infinite coefficients**. These cases can be solved using the results of Subsection 5.3.2

3. Finally, in Subsection 5.3.4, we detail the proof when f and g are finite affine function and when μ **is not monotonic in any variable**. This is the case when $a \geq 0 \wedge c \leq 0$.

5.3.2 Proof for the monotonic case with finite multidimensional affine functions

Let us consider that f and g have only finite coefficients. μ is the minimum of three functions, h, f and g. h is non-decreasing with respect to β and decreasing with respect to α. Let us recall that f is decreasing if and only if $a \leq 0$ and g is non-decreasing is and only if $c \geq 0$. If one of these two conditions is satisfied, μ is monotonic with respect to at least one variable. Let us detail each of these cases.

Case $a \leq 0$ and $c \geq 0$

In that case, f and h are decreasing with respect to α and g and h are non-decreasing with respect to β, as h. The variations of f, g and h are illustrated on Figure 5.10. As a

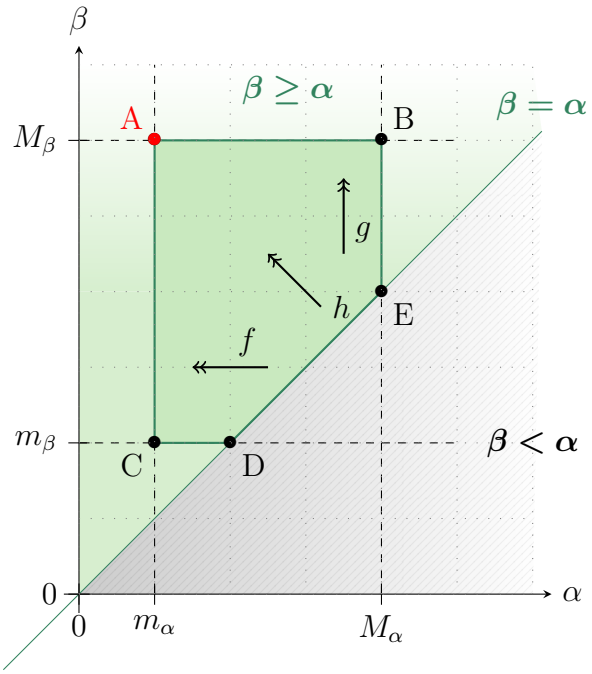


Figure 5.10: Variations of f, g and h in the definition set $\mathcal{D}$. The double arrows of f, g and h indicate the directions of variation of each function. For example the double arrow of f goes from left to right, because f is increasing with respect to α. It is perpendicular to β because it is constant with respect to β. A maximises μ over $\mathcal{D}$.

result, μ is non-decreasing with respect to β and decreasing with respect to α. Then, μ is maximised when:

$$\alpha = m_\alpha \text{ and } \beta = M_\beta.$$

It follows that $\boldsymbol{(\alpha^*, \beta^*) = (m_\alpha, M_\beta)}$ is an argsup of $\mu_{|\mathcal{D}}$.

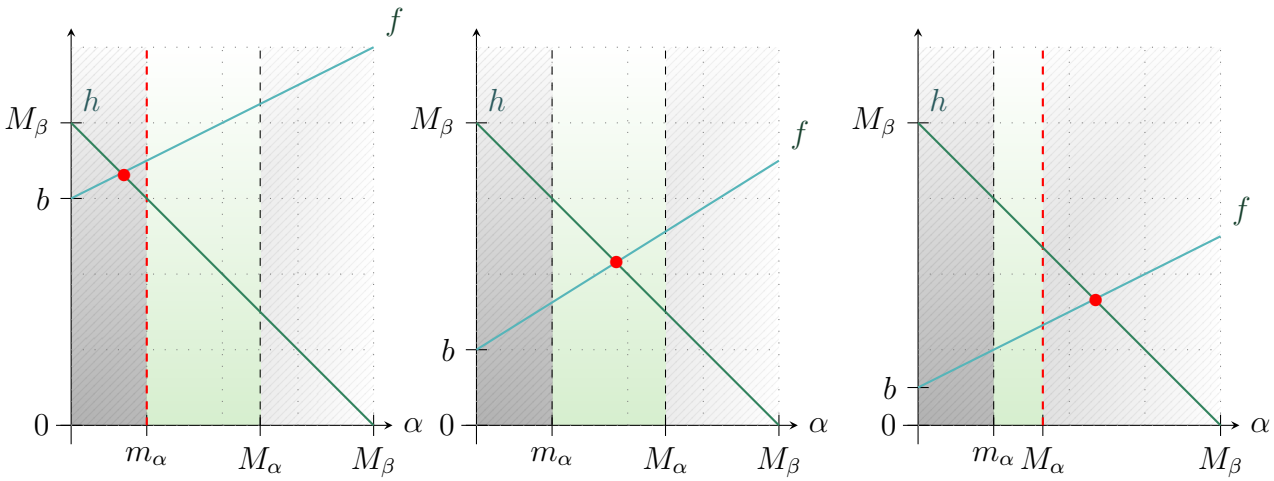
Case $a \geq 0$ and $c \geq 0$

In that case, f is non-decreasing and h is decreasing with respect to α, while g and h are non-decreasing with respect to β. As a result, μ is monotonic, and non decreasing, with respect to β. Hence μ is maximised over $\mathcal{D}$ at a point where $\beta = M_\beta$. It remains to maximise $\alpha \mapsto \mu(\alpha, M_\beta)$ over $\{\alpha \in \mathbb{R} \mid (\alpha, M_\beta) \in \mathcal{D}\}$.

As $\mu(\alpha, M_\beta) = \min(M_\beta - \alpha, a \cdot \alpha + b, c \cdot M_\beta + d)$, this function is maximised over $\mathbb{R}$ when α satisfies $M_\beta - \alpha = a \cdot \alpha + b$, *i.e.* for $\alpha = \dfrac{M_\beta - b}{a+1}$.

If $\left(\dfrac{M_\beta - b}{a+1}, M_\beta\right) \in \mathcal{D}$, this is the maximum of μ over $\mathcal{D}$, otherwise the maximum is reached on the border of $\{\alpha \in \mathbb{R} \mid (\alpha, M_\beta) \in \mathcal{D}\}$, i.e. for $\alpha = m_\alpha$ or $\alpha = \min(M_\beta, M_\alpha) = M_\alpha$. Examples are illustrated in Figure 5.11. **As a result:**

- If $\dfrac{M_\beta - b}{a+1} \leq m_\alpha$, then $\boldsymbol{(\alpha^*, \beta^*) = (m_\alpha, M_\beta)}$ is an argsup of $\mu_{|\mathcal{D}}$.
- If $m_\alpha \leq \dfrac{M_\beta - b}{a+1} \leq M_\alpha$, then $\boldsymbol{(\alpha^*, \beta^*) = \left(\dfrac{M_\beta - b}{a+1}, M_\beta\right)}$ is an argsup of $\mu_{|\mathcal{D}}$.
- If $M_\alpha \leq \dfrac{M_\beta - b}{a+1}$, then $\boldsymbol{(\alpha^*, \beta^*) = (M_\alpha, M_\beta)}$ is an argsup of $\mu_{|\mathcal{D}}$.



(a) Case $\dfrac{M_\beta - b}{a+1} \leq m_\alpha$. (b) Case $m_\alpha \leq \dfrac{M_\beta - b}{a+1} \leq M_\alpha$. (c) Case $M_\alpha \leq \dfrac{M_\beta - b}{a+1}$.

Figure 5.11: Variations of f and $\alpha \mapsto h(\alpha, M_\beta)$ when $a \geq 0$ and $c \geq 0$.

Case $a \leq 0$ and $c \leq 0$

In that case, $a \leq 0$ and $c \leq 0$. Then by letting $\alpha' = -\beta$ and $\beta' = -\alpha$, the problem becomes the one of maximising the following quantity:

$$\mu'(\alpha', \beta') = \min(\beta' - \alpha', -a\beta' + b, -c\alpha' + d)$$

over the set $\mathcal{D}'$ defined as follows:

$$\mathcal{D}' = \{(\alpha', \beta') \mid -M_\alpha \leq \beta' \leq -m_\alpha, -M_\beta \leq \alpha' \leq -m_\beta, \alpha' \leq \beta'\}.$$

Now $-c \geq 0$ and $-a \geq 0$, and we have reduced this case to the case where $a \geq 0$ and $c \geq 0$.

5.3.3 Proof for infinite multidimensional affine functions

In this case, we suppose that f or g have infinite coefficient, *i.e* when $f(\alpha, \beta) = \pm\infty$ or $g(\alpha, \beta) = \pm\infty$ for every $(\alpha, \beta) \in \mathcal{D}$.

Proof when $f = -\infty$ or $g = -\infty$

As $\mu = \min(f, g, h)$ equals $-\infty$ over $\mathcal{D}$, μ is constant over $\mathcal{D}$ and any point of $\mathcal{D}$ reaches the maximum of μ.

Proof when $f = +\infty$ or $g = +\infty$

- If $g = +\infty$ over $\mathcal{D}$, then for any $(\alpha, \beta) \in \mathcal{D}$, $\mu(\alpha, \beta) = \min(a \cdot \alpha + b, \beta - \alpha)$. μ is then non-decreasing with respect to β and is maximised when $\beta = M_\beta$. Maximising μ for the variable α now amounts to maximise $\alpha \mapsto \mu(\alpha, M_\beta)$. We can apply the same methods and results of the case where f and g are non-infinite functions and where $a \geq 0$. These cases are summed up in Table 5.2 in Subsection 5.3.5.

- If $f = +\infty$, then for any $(\alpha, \beta) \in \mathcal{D}$, $\mu(\alpha, \beta) = \min(c \cdot \beta + d, \beta - \alpha)$. μ is maximised when $\alpha = m_\alpha$. It now remains to maximise $\beta \mapsto \mu(m_\alpha, \beta)$. We can apply the results of the cases when $a \leq 0$ and f and g are non-infinite functions. Theses cases are summed up in Table 5.1 in Subsection 5.3.5.

5.3.4 Proof for the non-monotonic case with finite multidimensional affine functions

This case corresponds to the case $a > 0, c < 0$. Let us describe first the variation of μ with respect to α and β in Subsection 5.3.4.1. Then we enumerate all the possible cases in Subsection 5.3.4.2 with a coverage lemma. Finally, we describe an argsup for each of

theses cases in Subsection 5.3.4.3. All theses cases were illustrated in Subsection 5.3.4.4, in Figure 5.14, 5.15 and 5.16.

5.3.4.1 Description of μ

The function μ is no longer monotonic with respect to α or β, but each of the functions f, g and h is monotonic. We then split $\mathbb{R}^2$ into three convex polyhedra, depending on which of h, f, g is minimal, and look at the position of $\mathcal{D}$ with respect to these polyhedra. The domain of μ, $\mathcal{D}$, was described previously in Subsection 5.3.1.

$\mathbb{R}^2$ is split into the three sets of (α, β), depending whether $\min(f, g, h)$ equals f, g or h. Let us consider $\mathcal{P}(f) = \{P \in \mathbb{R}^2 \mid \mu(P) = f(P)\}$, $\mathcal{P}(g) = \{P \in \mathbb{R}^2 \mid \mu(P) = g(P)\}$, $\mathcal{P}(h) = \{P \in \mathbb{R}^2 \mid \mu(P) = h(P)\}$ the sets of points of $\mathbb{R}^2$ such that f (*resp.* g or h) is minimal. Theses sets are polyhedra of $\mathbb{R}^2_+$ and $\mathcal{P}(f) \cup \mathcal{P}(g) \cup \mathcal{P}(h) = \mathbb{R}^2$.

We represent these sets in the Figure 5.12. We can determine the three sets as follows:

$$\begin{aligned}
\mu(f, g, h) = f &\Leftrightarrow a \cdot \alpha + b \leq \beta - \alpha \wedge a \cdot \alpha + b \leq c \cdot \beta + d \\
&\Leftrightarrow \beta \geq (a+1)\alpha + b \wedge \beta \leq \frac{a}{c}\alpha + \frac{b-d}{c} \\
\mu(f, g, h) = g &\Leftrightarrow c \cdot \beta + d \leq \beta - \alpha \wedge c \cdot \beta + d \leq a \cdot \alpha + b \\
&\Leftrightarrow \beta \geq \frac{1}{1-c}(\alpha + d) \wedge \beta \geq \frac{a}{c}\alpha + \frac{b-d}{c} \\
\mu(f, g, h) = h &\Leftrightarrow \beta - \alpha \leq a \cdot \alpha + b \wedge \beta - \alpha \leq c \cdot \beta + d \\
&\Leftrightarrow \beta \leq (a+1)\alpha + b \wedge \beta \leq \frac{1}{1-c}(\alpha + d)
\end{aligned}$$

The point where $f = g = h$ is denoted $T = (T_\alpha, T_\beta)$ and is defined as follows:

$$T = \left(\frac{d - b(1-c)}{ac + c - a}, \frac{-b + d(1+a)}{ac + c - a} \right)$$

T is well defined as $a > 0$ and $c < 0$.

As $a, c \neq 0$, the half-lines d_1, d_2, d_3 are defined by the equations:

$$\begin{aligned}
d_1 &: \beta = (a+1)\alpha + b \wedge \beta \leq T_\beta \\
d_2 &: \beta = \frac{1}{1-c}(\alpha + d) \wedge \beta \geq T_\beta \\
d_3 &: \beta = \frac{a}{c}\alpha + \frac{b-d}{c} \wedge \beta \geq T_\beta
\end{aligned}$$

On d_1, f is equal to h, on d_2, g is equal to h and on d_3, f is equal to g. Therefore, T is the intersection point of d_1, d_2 and d_3.

On top of that, the slopes of d_1 and d_2 are strictly positive (as $a+1>0, 1/(1-c)>0$) and the slope of d_3 is strictly negative as $a/c<0$. The slope of d_1 is higher than the slope of d_2, as $a+1>1>1/(1-c)>0$. The only possibility for the relative positions of the half-lines is then the one described in Figure 5.12.

Moreover, f is an increasing function with respect to α, g is a non-increasing function with respect to β and h is an increasing function with respect to $\beta-\alpha$. μ reaches its maximal value over $\mathbb{R}^2$ at T. We need to compute (one of) the argsup of μ over $\mathcal{D}$. The argsup depends on the position of $\mathcal{D}$, as T is, in general, not necessarily in $\mathcal{D}$ (see the examples in Figure 5.14 and 5.16).

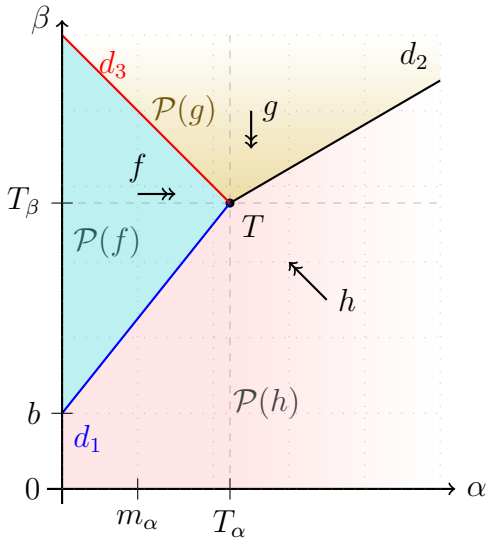


Figure 5.12: Values of μ in $\mathbb{R}^2$. The double arrows of f, g and h indicate the directions of variation of each function.

5.3.4.2 Coverage theorem

In the previous subsection, we show that the argsup of $\mu_{|\mathcal{D}}$ depends on the position of $\mathcal{D}$. There are several cases but the signs of the homogeneous terms of f and g restrict the possibilities. In this subsection, we enumerate 11 cases and in this next paragraph we will provide an argsup for each of these cases.

Intuition. The intuition is the following one: $\mathcal{D}$ can either intersect none, one, two or three of the half-lines d_1, d_2, d_3. A first attempt to find the argsup of $\mu_{|\mathcal{D}}$ was to consider the case where $\mu_{|\mathcal{D}}$ intersects none, one, two or three of these half-lines, and enumerate which one it intersects precisely. The goal, in the algorithm of Section 5.1, is to characterise the set of valuations v such that we can find an argsup of $\mu_{|\mathcal{D}}$. As the coefficients that depend on the valuation are b and d, an ideal solution would be to characterise these sets using linear conditions in b and d. The issue is that the set of valuations such that '$\mathcal{D}$ intersects d_1' cannot be directly characterised by linear conditions. We characterise these conditions with expressions of the form $X \in \mathcal{P}$ where X is a point with constant coordinates and $\mathcal{P}$ is a polyhedron. Indeed, as $\mathcal{P}(f)$, $\mathcal{P}(g)$, $\mathcal{P}(h)$ are polyhedra, the membership of a vertex of $\mathcal{D}$ (or T) to $\mathcal{P}(f), \mathcal{P}(g)$, $\mathcal{P}(h)$ or $\mathcal{D}$ is translated by linear inequalities. We also allow linear inequalities.

Basic notations of geometry. First, let us introduce some geometric notions. To denote the number of edges of $\mathcal{D}$ that a half-line $d \in \{d_1, d_2, d_3\}$ intersects, we use the following notation defined in Definition 5.10.

Definition 5.10: Number of intersections

Let n be an integer and $\mathcal{E}$ be a set edges. Let d be a line, half-line or ray of $\mathbb{R}^n$. We denote $\mathcal{N}(d, \mathcal{E})$ the number of elements of $\mathcal{E}$ that d intersects.

Example 5.3.2 *For instance, if a half-line d intersects $[AB]$ but not $[AC]$ then d intersects only one element in the set $\{[AB], [AC]\}$ and the following equality holds:*

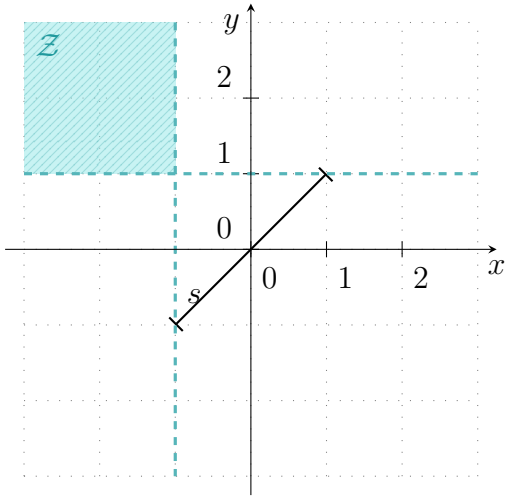
$$\mathcal{N}(d, \{[AB], [AC]\}) = 1$$

We also use in the next proofs vocabulary to describe the relative position of a edges. Let us consider two edges s_1, s_2 in $\mathbb{R}^2$ We say that:

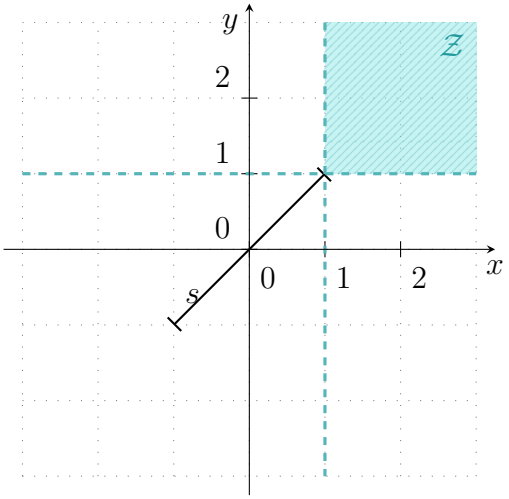
- s_1 is a **top left** edge of s_2 if for any point $p_1 = (x_1, y_1)$ of s_1 and for any point $p_2 = (x_2, y_2)$ of s_2: $x_1 \leq x_2$ and $y_1 \geq y_2$.
- s_1 is a **top right** edge of s_2 if for any point $p_1 = (x_1, y_1)$ of s_1 and for any point $p_2 = (x_2, y_2)$ of s_2: $x_1 \geq x_2$ and $y_1 \geq y_2$
- s_1 is a **bottom left** edge of s_2 if for any point $p_1 = (x_1, y_1)$ of s_1 and for any point $p_2 = (x_2, y_2)$ of s_2: $x_1 \leq x_2$ and $y_1 \leq y_2$

- s_1 is a **bottom right** edge of s_2 if for any point $p_1 = (x_1, y_1)$ of s_1 and for any point $p_2 = (x_2, y_2)$ of s_2: $x_1 \geq x_2$ and $y_1 \leq y_2$

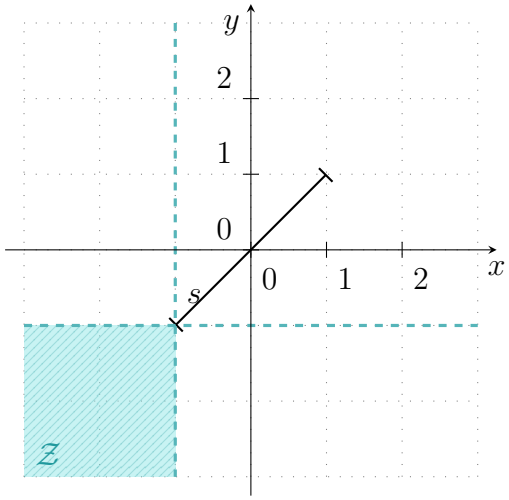
Examples in Figure 5.13 describe these four configurations.



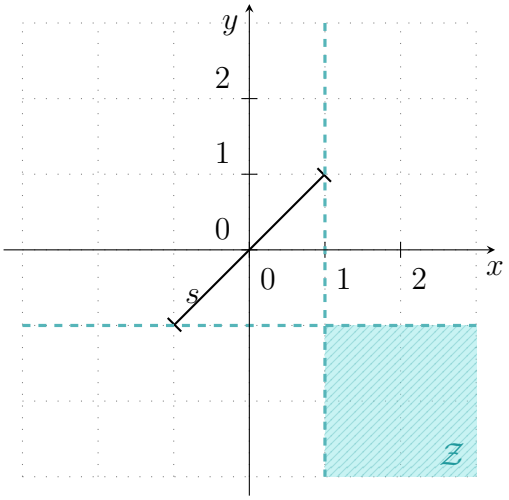
(a) All edges in the dashed zone $\mathcal{Z}$ are **top left** edges of s.



(b) All edges in the dashed zone $\mathcal{Z}$ are **top right** edges of s.



(c) All edges in the dashed zone $\mathcal{Z}$ are **bottom left** edges of s.



(d) All edges in the dashed zone $\mathcal{Z}$ are **bottom right** edges of s.

Figure 5.13: Relative positions of edges.

Coverage theorem. The result is the following Theorem 5.11

Theorem 5.11

If $a > 0$ and $c < 0$, then are 11 different relative positions of $\mathcal{D}$ with respect to the polyhedra $\mathcal{P}(f), \mathcal{P}(g)$ and $\mathcal{P}(h)$. These relative positions can be characterised by the 11 following configurations:

1. $A \in \mathcal{P}(h)$ (see Figure 5.14a).
2. $B \in \mathcal{P}(f)$ (see Figure 5.14b).
3. $C \in \mathcal{P}(g)$ (see Figure 5.14c).
4. $M_\beta \leq T_\beta$, $A \in \mathcal{P}(f)$ and $B \in \mathcal{P}(h)$ (see Figure 5.14d).
5. $m_\alpha \geq T_\alpha$, $A \in \mathcal{P}(g)$ and $C \in \mathcal{P}(h)$ (see Figure 5.15a).
6. $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$ (see Figure 5.15b).
7. $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(f)$ (see Figure 5.15c).
8. $D \in \mathcal{P}(f)$ and $E \in \mathcal{P}(g)$ (see Figure 5.15d).
9. $M_\alpha \leq T_\alpha$, $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(h)$ (see Figure 5.16a).
10. $m_\beta \geq T_\beta$, $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(h)$ (see Figure 5.16b).
11. $T \in \mathcal{D}$ (see Figure 5.16c).

These cases are described in Figures 5.14, 5.15 and 5.16.

Organisation of the proof of Theorem 5.11. The proof of this theorem uses several lemmas (Lemma 5.12, 5.14, 5.18, 5.19 and 5.20) that we will state and prove in this subsection. Lemma 5.12 enumerates which half-lines $\mathcal{D}$ can intersect at the same time. Then, we detail in Lemmas 5.14, 5.18, 5.19, 5.20 the cases when $\mathcal{D}$ intersects respectively no half-lines, one half-lines, d_1 and d_3, d_2 and d_3. Finally, we will detail the case when $\mathcal{D}$ intersects the three half-lines d_1, d_2 and d_3.

The next lemmas are organised as follows:

- ▷ We consider the three half-lines d_1, d_2 and d_3 and we enumerate how many of these half-lines can intersect the domain set $\mathcal{D}$ (see Lemma 5.12). We are then able to study cases with respect to the number of half-lines that can intersect $\mathcal{D}$.

- We first study the case where no half-line intersects $\mathcal{D}$ (see Lemma 5.14).
- Secondly, we consider the set of edges of $\mathcal{D}$, $\{[AB], [AC], [BE], [CD], [DE]\}$. We state for each half-line d_1, d_2 and d_3 which edges of $\mathcal{D}$ is crossed when one of these half-lines intersects $\mathcal{D}$ (see Lemmas 5.15, 5.16 and 5.17).
- Thirdly, we study the case where exactly one of the half-lines intersects $\mathcal{D}$ (see Lemma 5.18).
- Fourthly, we study the cases where exactly two of the half-lines intersect $\mathcal{D}$. It can be either d_1 and d_3 (see Lemma 5.19) or d_2 and d_3 (see Lemma 5.20).
- Fifthly and finally, we study the unique case where the three half-lines intersect $\mathcal{D}$.

Which half-lines can $\mathcal{D}$ intersects? As each half-line has a fixed sign of slope, $\mathcal{D}$ cannot intersect some half-lines at the same time. For instance, it is impossible that $\mathcal{D}$ intersects d_1 and d_2 without intersecting d_3. We detail which half-lines $\mathcal{D}$ can intersect at the same time in Lemma 5.12.

Lemma 5.12

The domain $\mathcal{D}$ can intersect:

1. Either one of the half-lines d_1, d_2, d_3.
2. Or none of them.
3. Or d_1 and d_3.
4. Or d_2 and d_3.
5. Or the three half-lines.

Proof. Let us prove two facts. First let us prove that if the set $\mathcal{D}$ intersects d_1 and d_2, then $\mathcal{D}$ intersects d_3. If there exists a common point of d_1 and d_2, $(\alpha_0, \beta_0) \in d_1 \cap d_2$, then $\beta_0 = T_\beta$. On top of that, as (α_0, β_0) belongs to d_1, its second coordinate satisfies the following equalities:

$$\beta_0 = (a+1)\alpha_0 + b, \beta_0 = \frac{1}{1-c}(\alpha_0 + d).$$

As a result, $\alpha_0 = T_\alpha$ and $T \in \mathcal{D}$. $\mathcal{D}$ then intersects d_3 too. In all cases, it has been shown that $\mathcal{D}$ intersects the three half-lines if it intersects d_1 and d_2. Therefore if it intersects only **two** half-lines, it can only intersect d_1 and d_3, or d_2 and d_3.

□

The following Lemma 5.13 states an important geometric result for the next lemmas of this subsection: it shows that the exact number of edges a half-line intersects when intersecting $\mathcal{D}$. It will be very useful when detailing each half-line, because it will reduce the number of edge combinations to be enumerated.

Lemma 5.13

If a half-line $d \in \{d_1, d_2, d_3\}$ intersects $\mathcal{D}$ **and** if $T \notin \mathcal{D}$, then it intersects exactly two edges of $\mathcal{D}$ (that are $[AB], [BE], [AC], [CD]$ and $[DE]$).

Proof. Let us consider a half-line d such that d intersects $\mathcal{D}$ and T is the initial point of d. d intersects at least one edge of $\mathcal{D}$ and at most two edges. Let us suppose that d intersects only one vertex, then $T \in \mathcal{D}$. As T does **not** belong to $\mathcal{D}$, d intersects exactly two edges of $\mathcal{D}$. □

In the next paragraphs, we will study the intersection of $\mathcal{D}$ for each possible combination and translate the intersection of $\mathcal{D}$ with half-lines into the membership of vertices to $\mathcal{P}(f), \mathcal{P}(g)$ or $\mathcal{P}(h)$.

Cases where $\mathcal{D}$ intersects none of the half-lines. Let us first consider in Lemma 5.14 the case where $\mathcal{D}$ intersects **none** of the half-lines.

Lemma 5.14

If the domain $\mathcal{D}$ intersects no half-lines, then either $B \in \mathcal{P}(f)$, or $C \in \mathcal{P}(g)$, or $A \in \mathcal{P}(h)$.

The cases when $B \in \mathcal{P}(f)$ and $C \in \mathcal{P}(g)$ are illustrated in Figures 5.14b and 5.14c.

Proof. If $\mathcal{D}$ intersects no half-lines, then either $\mu_{|\mathcal{D}} = f$, or $\mu_{|\mathcal{D}} = g$ or $\mu_{|\mathcal{D}} = h$ over $\mathcal{D}$. This directly proves that if $\mu_{|\mathcal{D}} = f$ then $\mathcal{D} \subseteq \mathcal{P}(f)$ and then $B \in \mathcal{P}(f)$. For the other cases we can respectively prove that $C \in \mathcal{P}(g)$ if $\mu_{|\mathcal{D}} = g$ and $A \in \mathcal{P}(h)$ if $\mu_{|\mathcal{D}} = h$. □

Cases where $\mathcal{D}$ intersects at least one half-line. In this paragraph, we detail for each half-line the combination of vertices that a half-line cannot intersect. We can prove the contradiction of some combination because of the informations on the slope of the half-lines, and the relative position of the edges on $\mathcal{D}$. Examples of intersections of half-lines with $\mathcal{D}$ are given in Figure 5.14, 5.15 and 5.16 to illustrate our proof. We study the

case where d_1 intersects $\mathcal{D}$ in Lemma 5.15 (case illustrated in Figures 5.14b 5.14d), where d_2 intersects $\mathcal{D}$ in Lemma 5.16 (case illustrated in Figures 5.14c and 5.15a) and where d_3 intersects $\mathcal{D}$ in Lemma 5.17 (case illustrated in Figures 5.15b, 5.15c and 5.15d).

Lemma 5.15

If d_1 intersects $\mathcal{D}$, if $T \notin \mathcal{D}$ and if $B, C, D \notin d_3$ then:

$$\mathcal{N}(d_1, \{[AB], [BE]\}) = 1$$

and

$$\mathcal{N}(d_1, \{[CD], [AC], [DE]\}) = 1$$

Proof. We exclude the cases here d_1 intersects the edges on B, C or D. For instance, if d_1 intersects $[CD]$ exactly on C, the half-line trivially intersects $[AC]$.

As d_1's slope is strictly greater than 1:

- d_1 cannot intersect both $[CD]$ and $[DE]$ (or $[AC]$).

  Let us suppose that d_1 intersects $[CD]$.

  As $[DE]$ is an edge that starts from $[CD]$ at D, and has a slope of exactly 1, d_1 cannot intersect both edges without crossing D.

  As $[AC]$ is an edge that starts from $[CD]$ at C and has a **infinite** slope, it cannot cross d_1, which has a **finite** slope.

- With the same arguments, d_1 cannot intersect both $[AC]$ and $[DE]$.

  Let us suppose that d_1 intersects $[AC]$ and let us prove that it cannot intersect $[DE]$. Indeed, vertex D is on the right, at the same height as vertex C. As the slope of d_1 is strictly greater than 1 and the slope of $[DE]$ is exactly 1, these half-line and edge cannot cross.

As a result, if d_1 intersects $[CD]$ (*resp.* $[AC]$, or $[DE]$), d_1 intersects either $[AB]$ or $[BE]$

□

Lemma 5.16

If d_2 intersects $\mathcal{D}$, if $T \notin \mathcal{D}$ and if $C, B, E \notin \mathcal{D}$, then:

$$\mathcal{N}\left(d_2, \{[CD], [AC]\}\right) = 1$$

and

$$\mathcal{N}\left(d_2, \{[AB], [BE], [DE]\}\right) = 1$$

Proof. As the slope of d_2 is strictly positive, if it intersects a edge s, it cannot intersect a top left, or a bottom right, edge of s. As a result:

- if d_2 intersects $[CD]$, it cannot intersect $[AC]$.
- if d_2 intersects $[AB]$, it cannot cross $[AE]$, except on A.

Let us now suppose that d_2 intersects $[DE]$, it cannot cross $[AB]$ (or $[AE]$, except on E) because the slope of d_2 is smaller than 1.

□

Lemma 5.17

If d_3 intersects $\mathcal{D}$, if $T \notin \mathcal{D}$ and if $A, D, E \notin \mathcal{D}$, then:

$$\mathcal{N}\left(d_3, \{[AB], [AC]\}\right) = 1$$

and

$$\mathcal{N}\left(d_3, \{[CD], [BE], [DE]\}\right) = 1$$

Proof. The slope of d_3 is strictly negative, therefore:

- if d_3 intersects $[AB]$, it cannot intersect a bottom left edge. As a result, it cannot intersect $[AC]$ (except on A).
- if d_3 intersects $[CD]$, it cannot intersect an top right edge. As a result it cannot intersect $[BE]$ or $[DE]$ (except on D *resp.* E).
- if d_3 intersects $[BE]$, it cannot intersect a bottom left edge. As a result it cannot intersect $[DE]$ (except on E).

□

Cases where $\mathcal{D}$ intersects only one half-line. Now that we study which edge was crossed when each half-line among d_1, d_2 and d_3 intersects $\mathcal{D}$, we can use this result to translate the fact that only **one** half-line intersects $\mathcal{D}$ into a membership of a vertex to a polyhedron. To do so, we will use lemmas 5.15, 5.16 and 5.17 for each half-line. For instance, for half-line d_1, if it intersects $\mathcal{D}$, then it necessarily intersects either $[AB]$ or $[BE]$, but not both. As d_1 is a separation between $\mathcal{P}(f)$ and $\mathcal{P}(h)$, we can deduce the position of A, B or E. Let us state formally these results in Lemma 5.18.

Lemma 5.18

If the domain $\mathcal{D}$ intersects only one half-line, then:

- ▷ If only d_1 intersects $\mathcal{D}$, then either ($B \in \mathcal{P}(f)$ and $E \in \mathcal{P}(h)$) or ($M_\beta \leq T_\beta$, $A \in \mathcal{P}(f)$ and $B \in \mathcal{P}(h)$).

  These cases are illustrated respectively in Figures 5.14b and 5.14d.

- ▷ If only d_2 intersects $\mathcal{D}$, then either $C \in \mathcal{P}(g)$ or ($m_\alpha \geq T_\alpha$, $A \in \mathcal{P}(g)$ and $C \in \mathcal{P}(h)$).

  Theses cases are illustrated respectively in Figures 5.14c and 5.15a.

- ▷ If only d_3 intersects $\mathcal{D}$, then either ($C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$) or ($B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(f)$) or ($D \in \mathcal{P}(f)$ and $E \in \mathcal{P}(g)$).

  Theses cases are illustrated respectively in Figures 5.15b, 5.15c and 5.15d.

Proof. Let us detail the edges that are crossed for each half-lines.

- ▷ If d_1 intersects $\mathcal{D}$ then, thanks to Lemma 5.15:
  - Either d_1 intersects $[AB]$, then $A \in \mathcal{P}(f)$ and $B \in \mathcal{P}(h)$. In that case, as $\mathcal{D}$ does not intersect other half-lines, $M_\beta \leq T_\beta$.
  - Or d_1 intersects $[BE]$, then $B \in \mathcal{P}(f)$ and $E \in \mathcal{P}(h)$.

  As a result ($A \in \mathcal{P}(f)$ and $B \in \mathcal{P}(h)$) or ($B \in \mathcal{P}(f)$ and $E \in \mathcal{P}(h)$).

- ▷ If d_2 intersects $\mathcal{D}$ and then, thanks to Lemma 5.16:
  - Either d_2 intersects $[CD]$, then $C \in \mathcal{P}(g)$.
  - Or d_2 intersects $[AC]$. In that case, as $\mathcal{D}$ does not intersect other half-lines, $m_\alpha \leq T_\alpha$, on top of that $A \in \mathcal{P}(g)$ and $C \in \mathcal{P}(h)$.

As a result, either $C \in \mathcal{P}(g)$ or ($m_\alpha \leq T_\alpha$, $A \in \mathcal{P}(g)$ and $C \in \mathcal{P}(h)$)

- If d_3 intersects $\mathcal{D}$ and then, thanks to Lemma 5.17:
  - Either d_3 intersects $[CD]$, then $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$.
  - Or d_3 intersects $[BE]$, then $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(f)$.
  - Or d_3 intersects $[DE]$, then $D \in \mathcal{P}(f)$ and $E \in \mathcal{P}(g)$.

  As a result, either ($C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$) or ($B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(f)$) or ($D \in \mathcal{P}(f)$ and $E \in \mathcal{P}(g)$).

□

Cases where $\mathcal{D}$ intersects exactly two half-lines. Now, let us study the cases where $\mathcal{D}$ intersects exactly two half-lines. Thanks to Lemma 5.12, we know that if $\mathcal{D}$ intersects exactly **two** half-lines, it is d_1 and d_3, or d_2 and d_3 (otherwise it intersects more than two half-lines). Lemmas 5.19 studies the case where d_1 and d_3 intersect $\mathcal{D}$, and Lemma 5.20 studies the case where d_2 and d_3.

Lemma 5.19

If the domain $\mathcal{D}$ intersects d_1 and d_3, then $T_\alpha \geq M_\alpha$, $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(h)$.

This case is illustrated in Figure 5.16a.

Proof. Considering Lemmas 5.15 and 5.17, we know that:

$$\begin{aligned}
\mathcal{N}(d_1, \{[AB], [BE]\}) &= 1 \\
\mathcal{N}(d_1, \{[CD], [AC], [DE]\}) &= 1 \\
\mathcal{N}(d_3, \{[AB], [AC]\}) &= 1 \\
\mathcal{N}(d_3, \{[CD], [BE], [DE]\}) &= 1
\end{aligned}$$

In addition:

- d_1 and d_3 cannot intersect both $[CD]$ and $[DE]$, otherwise we would have $T \in \mathcal{D}$.
- Suppose that d_1 and d_3 intersect $\mathcal{D}$ and d_1 intersects $[AB]$ then $T_\beta \geq M_\beta$. d_3 does not intersect $\mathcal{D}$ **or** $T \in \mathcal{D}$ (which is a contradiction). As a result, d_1 cannot intersect $[AB]$.

Therefore, d_1 intersects $[BE]$, then d_3 cannot intersect $[CD]$ and $[DE]$ (its initial (and lowest) point is above all points of d_1). d_3 then intersects $[BE]$ too. To conclude, if only d_1 and d_3 intersect $\mathcal{D}$, then they intersect $[BE]$ and $\mathcal{N}(d_3, \{[AC], [DE]\}) = 1$. In conclusion, $B \in \mathcal{P}(g)$, $E \in \mathcal{P}(h)$ and $T_\beta \geq M_\beta$.

□

Lemma 5.20

If the domain $\mathcal{D}$ intersects exactly d_3 and d_2, then $m_\beta \geq T_\beta$, $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(h)$.

This case is illustrated in Figure 5.16b.

Proof. If d_3 and d_2 intersect $\mathcal{D}$, then by Lemmas 5.16 and 5.17:

$$\begin{aligned}
\mathcal{N}(d_2, \{[CD], [AC]\}) &= 1 \\
\mathcal{N}(d_2, \{[AB], [BE], [DE]\}) &= 1 \\
\mathcal{N}(d_3, \{[AB], [AC]\}) &= 1 \\
\mathcal{N}(d_3, \{[CD], [BE], [DE]\}) &= 1
\end{aligned}$$

The goal of this proof is to prove that necessarily, d_2 and d_3 intersect $[CD]$. First, we prove that d_2 intersects $[CD]$. d_2 must intersect either $[CD]$ or $[AC]$. If d_2 intersects $[AC]$ then $T_\alpha \leq m_\alpha$ and d_3 cannot intersect $\mathcal{D}$, which is absurd. As a result d_2 intersects $[CD]$.

Secondly, we prove that necessarily d_3 intersects $[CD]$:

- d_3 cannot intersect $[DE]$: if d_3 intersects $[DE]$, T is below right of the edge $[DE]$. d_2, which slope is between 0 and 1 cannot intersect $\mathcal{D}$. This leads to a contradiction.
- d_3 cannot intersect $[BE]$: indeed if d_3 intersects $[BE]$, $T_\alpha \geq M_\alpha$ and d_2 then cannot intersect $\mathcal{D}$. This leads to a contradiction.

Necessarily d_3 and d_2 intersect $[CD]$, then $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(h)$. Finally, if $\mathcal{D}$ intersects d_3 and d_2 but not d_1, then $m_\beta \geq T_\beta$. □

Cases where $\mathcal{D}$ intersects the three half-lines. Finally, the domain $\mathcal{D}$ can intersect the three half-lines d_1, d_2 and d_3 at the same time. If these three half-lines intersects the domain $\mathcal{D}$, then their intersection point T belongs to $\mathcal{D}$. As this point is the argsup of μ in $\mathbb{R}^2$, it is a sufficient condition to characterise an optimal point over $\mathcal{D}$. This case is illustrated in Figure 5.16c.

Proof of the final Theorem 5.11. In this paragraph, we prove the Theorem 5.11 that summarises **all** the possible configurations. The intuition is the following one: the domain $\mathcal{D}$ can either intersect none, one, two or three half-lines among the three half-lines that we consider. With the previous lemmas, we stated which half-lines this domain can intersect at the same time, and considering each of the configurations, we translated it in term of membership of T or the vertex of $\mathcal{D}$ to the polyhedra $\mathcal{P}(f), \mathcal{P}(g)$ or $\mathcal{P}(h)$.

Proof of Theorem 5.11. By Lemma 5.12, we know which half-lines $\mathcal{D}$ can intersect among d_1, d_2 and d_3: $\mathcal{D}$ can either intersect one of the half-lines d_1, d_2, d_3, or none of them, or d_1 and d_3, or d_2 and d_3, or the three half-lines. Then:

- Lemma 5.14 provides the possible configuration if $\mathcal{D}$ intersects **none** of the half-lines. These cases are covered by the case 1, 2 and 3 enumerated in the Theorem.
- Lemma 5.14 provides the possible configuration if $\mathcal{D}$ intersects **exactly one** of the half-lines. These cases are covered by the case 2, 3, 4, 5, 6, 7 and 8 enumerated in the Theorem.
- Lemma 5.19 and 5.20 provide the possible configuration if $\mathcal{D}$ intersects **exactly two** of the half-lines. These cases are covered by the case 4 and 10 enumerated in the Theorem.
- By definition, $T \in \mathcal{D}$ if and only if $\mathcal{D}$ intersects **all** the half-lines. These cases are covered by the case 11, enumerated in the Theorem.

□

5.3.4.3 Optimisation for every cases

Now that we have described the different configurations in Theorem 5.11 in the previous Subsection 5.3.4.2, we now find the different possible argsup of $\mu_{|\mathcal{D}}$, depending on the position of $\mathcal{D}$.

Let us recall the different possible configurations when $a > 0$ and $c < 0$:

1. $A \in \mathcal{P}(h)$
2. $B \in \mathcal{P}(f)$
3. $C \in \mathcal{P}(g)$

4. $M_\beta \leq T_\beta$, $A \in \mathcal{P}(f)$ and $B \in \mathcal{P}(h)$
5. $m_\alpha \geq T_\alpha$, $A \in \mathcal{P}(g)$ and $C \in \mathcal{P}(h)$
6. $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$
7. $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(f)$
8. $D \in \mathcal{P}(f)$ and $E \in \mathcal{P}(g)$
9. $M_\alpha \leq T_\alpha$, $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(h)$
10. $m_\beta \geq T_\beta$, $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(h)$.
11. $T \in \mathcal{D}$.

Let us now detail, for each of these cases, the argsup of $\mu_{|\mathcal{D}}$. All these cases are illustrated in the Figures 5.14 and 5.16. The gray zones  represent zones that do not intersect $\mathcal{D}$. The value of the found argsup will be summed up in Tables 5.3, 5.4 and 5.5.

Case 1: $A \in \mathcal{P}(h)$. If $A \in \mathcal{P}(h)$, then $\mathcal{D}$ does not cross any half-line and $\mathcal{D} \subseteq \mathcal{P}(h)$. Therefore, $\mu_{|\mathcal{D}} = h : \ \alpha, \beta \ \mapsto \ \beta - \alpha$. This function reaches its maximal on A$= (m_\alpha, M_\beta)$, α is minimum and β maximal. **As a result, if $A \in \mathcal{P}(h)$, then $(\alpha^*, \beta^*) = A$ is an argsup of $\mu_{|\mathcal{D}}$.**

Case 2: $B \in \mathcal{P}(f)$. If $B \in \mathcal{P}(f)$, then $\mu_{|\mathcal{D}}$ crosses at **most** d_1. As a result, $\mu_{|\mathcal{D}} = \min(f, h)$. As f and h are non-decreasing functions with respect to β, $\mu_{|\mathcal{D}}$ reaches its maximal when β is maximal. Then, if $D \in \mathcal{P}(h)$ $\mu_{|\mathcal{D}}$ reaches its maximal over d_1 and $[BE]$, so in $(M_\alpha, d_1(M_\alpha)) \in \mathcal{D}$. Otherwise, if $E \notin \mathcal{P}(h)$ then $E \in \mathcal{P}(f)$, then $\mu_{|\mathcal{D}}$ reaches its maximal in $B = (M_\alpha, M_\beta)$. As f is constant with respect to β and $d_1 \subseteq \mathcal{P}(f)$, $\mu(B) = \mu(M_\alpha, d_1(M_\alpha))$ when $D \in \mathcal{P}(h)$. As a result, in any case, $\mu_{|\mathcal{D}}$ reaches it maximal value in B when $B \in \mathcal{D}$.

As a result, if $B \in \mathcal{P}(f)$, then $(\alpha^*, \beta^*) = B$ is an argsup of $\mu_{|\mathcal{D}}$.

Case 3: $C \in \mathcal{P}(g)$. For the same reasons, if $C \in \mathcal{P}(g)$, $\mu_{|\mathcal{D}}$ reaches its maximal value at C. Indeed in that case, $\mu_{|\mathcal{D}} = min(g, h)$ because $\mathcal{D}$ crosses at most d_2. In $\mathcal{P}(g)$, μ reaches its maximal value when β is minimum, and does not depend on α. In $\mathcal{P}(h)$,

μ reaches its minimal value over the half-line d_3, but over this half-line, $g \equiv h$, so the maximum is reached on $C = (m_\alpha, m_\beta)$.

As a result, if $C \in \mathcal{P}(g)$, then $(\alpha^*, \beta^*) = C$ is an argsup of $\mu_{|\mathcal{D}}$.

Case 4: $M_\beta \leq T_\beta$, $A \in \mathcal{P}(f)$ and $B \in \mathcal{P}(h)$. As $T_\beta \geq M_\beta$, $\mathcal{D}$ is below T and does not intersect $\mathcal{P}(g)\backslash\{T\}$, $\mu_{|\mathcal{D}} = min(f, h)$. $\mu_{|\mathcal{D}}$ is then increasing with respect to β. As $[AB]$ intersects d_1, $(d_1^{-1}(M_\beta), M_\beta) \in \mathcal{D}$ and $\mu_{|\mathcal{D}}$ reaches its maximal over d_1 on this point. The condition $A \in \mathcal{P}(f)$ and $B \in \mathcal{P}(h)$ can be characterised with $f(A) \leq h(A)$ and $h(B) \leq f(B)$ because $\mu_{|\mathcal{D}} = min(f, h)$.

As a result, if $T_\beta \geq M_\beta$, $f(A) \leq h(A)$ and $h(B) \leq f(B)$, then $(\boldsymbol{\alpha^*, \beta^*}) = \left(\dfrac{\boldsymbol{M_\beta - b}}{\boldsymbol{a+1}}, M_\beta\right)$ is an argsup of $\mu_{|\mathcal{D}}$.

Case 5: $m_\alpha \geq T_\alpha$, $A \in \mathcal{P}(g)$ and $C \in \mathcal{P}(h)$. If $T_\alpha \leq m_\alpha, A \in \mathcal{P}(g), C \in \mathcal{P}(h)$.

As $\mathcal{D}$ is on the right-hand side of the intersection point, it does not intersect $\mathcal{P}(f)\backslash d_1$, so $\mu_{|\mathcal{D}} \equiv \min(g, h)$. As μ is non-increasing with respect to α, μ reaches its maximal value when α is minimum (*i.e* where $\alpha = m_\alpha$). Then if $g(A) \leq h(A) \wedge h(C) \leq g(C)$, μ reaches its maximal value when $\beta = \dfrac{m_\alpha + d}{1 - c}$.

As a result, if $T_\alpha \leq m_\alpha$, $g(A) \leq h(A)$ and $h(C) \leq g(C)$, then $(\boldsymbol{\alpha^*, \beta^*}) = \left(m_\alpha, \dfrac{\boldsymbol{m_\alpha + d}}{\boldsymbol{1 - c}}\right)$ is an argsup of $\mu_{|\mathcal{D}}$.

Case 6: $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$. If $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$, then $\mathcal{D}$ only intersects the half-line d_3 and $\mu_{|\mathcal{D}} = \min(f, g)$. As a result, $\mu_{|\mathcal{D}}$ reaches it maximal point over d_3, when α is minimum, so at the intersection with the edge $[CD]$.

As a result, if $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$, then $(\boldsymbol{\alpha^*, \beta^*}) = (\boldsymbol{((1-c) \cdot m_\beta - d}, m_\beta)$ is an argsup of $\mu_{|\mathcal{D}}$.

Case 7: $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(f)$. With the same argument, the edge $[BD]$ crosses d_3, then either $\mathcal{D}$ is a rectangle and $\mu_{|\mathcal{D}} = min(f, g)$, or $\mathcal{D}$ is a convex pentagon and a triangle and the slope of $[BE]$ is 1, which is strictly smaller than $a+1$. So $\mathcal{D}$ does not intersect $\mathcal{P}(h)\backslash d_1$, and $\mu_{|\mathcal{D}} = min(f, g)$ too. In any case, $\mu_{|\mathcal{D}}$ reaches its maximal value over d_3 and $[BE]$. **As a result, if $g(B) \leq f(B), g(B) \leq h(B), f(E) \leq g(E), f(E) \leq h(E)$, then $(\boldsymbol{\alpha^*, \beta^*}) = (\boldsymbol{M_\alpha}, \max(\boldsymbol{M_\alpha, m_\beta}))$ is an argsup of $\mu_{|\mathcal{D}}$.**

Case 8: $D \in \mathcal{P}(f)$ and $E \in \mathcal{P}(g)$. In this case, $\mu_{|\mathcal{D}} = (f, g)$ and $\mathcal{D}$ intersects only the half-line d_3, in particular with $[DE]$.The maximal value of $\mu_{|\mathcal{D}}$ is then reached over d_3 and $[DE]$, *i.e.* when $\alpha = \beta$ and $\beta = \frac{a}{c}\alpha + \frac{b-d}{c}$.

As a result, if $D \in \mathcal{P}(f)$ and $E \in \mathcal{P}(g)$, then $\left(\frac{b-d}{c-a}, \frac{b-d}{c-a}\right)$ is an argsup of $\mu_{|\mathcal{D}}$.

Case 9: $M_\alpha \leq T_\alpha, B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(h)$. As $M_\alpha \leq T_\alpha$, $\mathcal{D}$ is left to T and does not intersects d_2. As $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(h)$, $\mathcal{D}$ necessarily intersects d_1 and d_3. The maximal value of $\mu_{|\mathcal{D}}$ is reached on its intersection with d_1 or d_3. As on these two half-line, $\mu_{|\mathcal{D}}$ is equals to f, the $[BE] \cap d_1$ and $[BE] \cap d_3$ have the same value with respect to α, $\mu_{|\mathcal{D}}$ has the same value on these two points.

As a result, if $M_\alpha \leq T_\alpha$, $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(h)$, then $(M_\alpha, (a+1) \cdot M_\alpha + b)$ is an argsup of $\mu_{|\mathcal{D}}$.

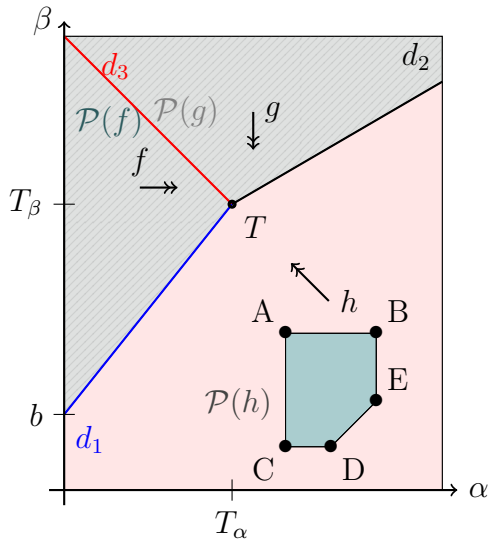
Case 10: $m_\beta \geq T_\beta, C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(h)$. As $m_\beta \geq T_\beta$, $\mathcal{D}$ crosses only d_2 and d_3, and **not** d_1. $\mathcal{D}$ is then split in the three polyhedra $\mathcal{P}(f), \mathcal{P}(g)$ and $\mathcal{P}(h)$. Over $\mathcal{P}(f)$, $\mu_{|\mathcal{D}}$ is maximised on the point where α is the greater, so on the intersection with d_3 and $[CD]$, where $\mu_{|\mathcal{D}}$ is also equal to g. Over $\mathcal{P}(h)$, $\mu_{|\mathcal{D}}$ is maximised on the intersection with d_2, where $\beta - \alpha$ is maximal, *i.e.* on $[CD] \cap d_2$. Both $[CD] \cap d_3$ and $[CD] \cap d_3$ are in $\mathcal{P}(g)$ so $\mu_{|\mathcal{D}}$ have the same value on both of these points, as $\mu_{|\mathcal{D}}$ does not depend on α over $\mathcal{P}(g)$.

As a result, if $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(h)$, $\left(\frac{c}{a}m_\beta + \frac{d-b}{c}, m_\beta\right)$ is an argsup of $\mu_{|\mathcal{D}}$

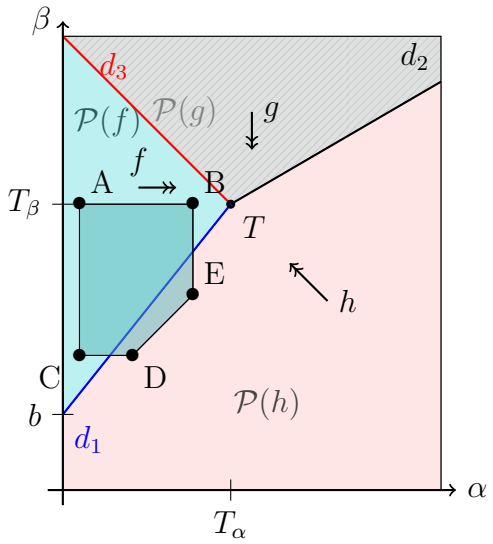
Case 11: $T \in \mathcal{D}$. finally, we can characterise the fact that T belongs to $\mathcal{D}$ by: $T_\beta \geq T_\alpha$, $T_\beta \leq m_\beta$, $T_\alpha \leq M_\alpha$ and $T_\alpha \geq m_\alpha$. If these conditions hold, then μ reaches its maximal at T.

5.3.4.4 Appendix: Figures

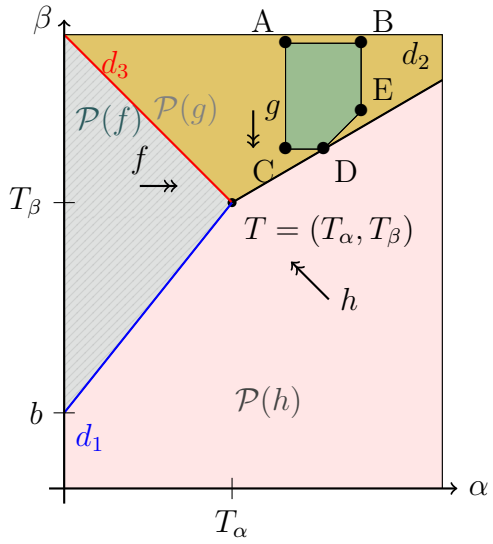
In this subsection, we describe with Figures the different cases that were enumerated in Subsections 5.3.4.2 and 5.3.4.3. Theses cases are represented in Figures 5.14, 5.15 and 5.16. Each cases of Theorem 5.11 is represented in these Figures. Let us recall that the double-arrows represent the variation of μ in $\mathcal{P}(f), \mathcal{P}(g)$ and $\mathcal{P}(h)$. The gray zones represent zones, among $\mathcal{P}(f), \mathcal{P}(g)$ and $\mathcal{P}(h)$, that $\mathcal{D}$ does not intersect.



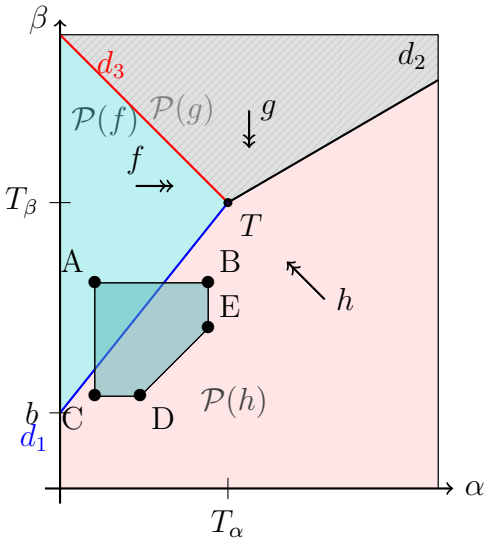
(a) Case 1: $A \in \mathcal{P}(h)$.



(b) Case 2: $B \in \mathcal{P}(f)$.

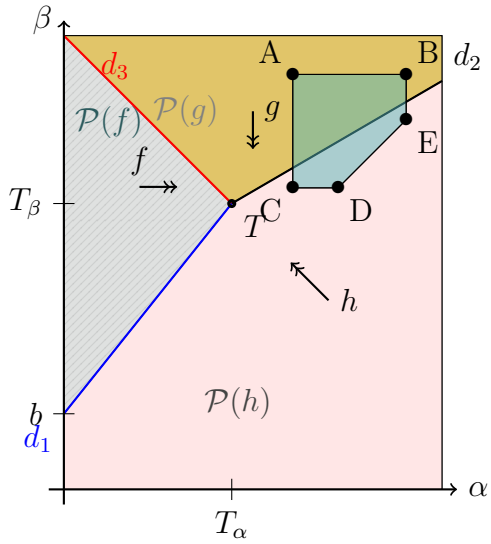


(c) Case 3: $C \in \mathcal{P}(g)$.

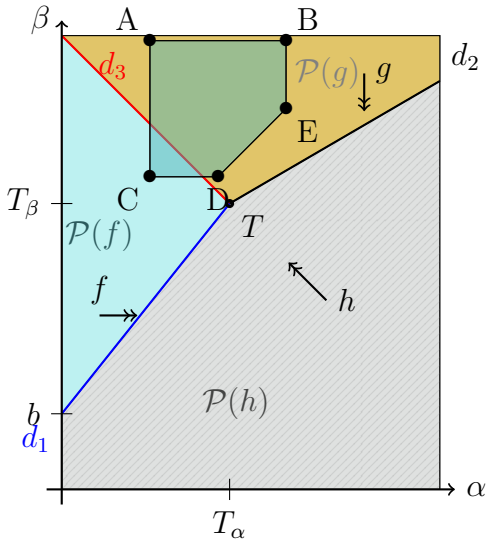


(d) Case 4: $A \in \mathcal{P}(f)$, $B \in \mathcal{P}(h)$ and $T_\beta \geq M_\beta$.

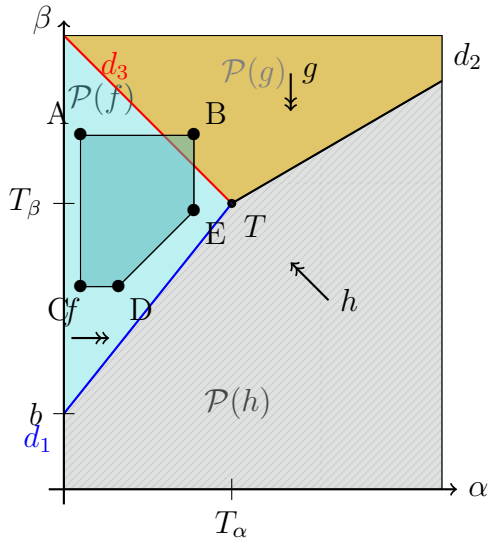
Figure 5.14: Summary of the first fourth cases.



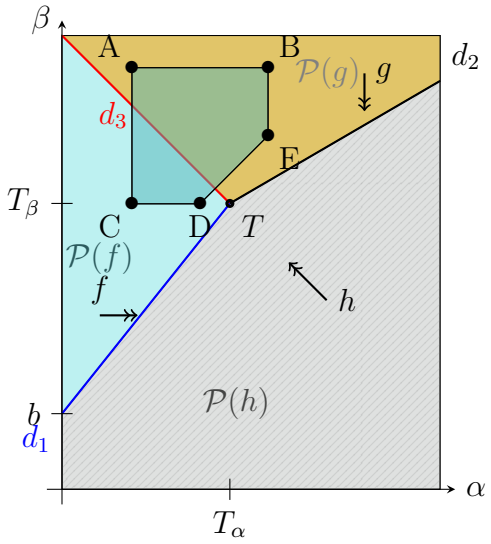
(a) Case 5: $A \in \mathcal{P}(g)$, $C \in \mathcal{P}(h)$ and $T_\alpha \leq m_\alpha$.



(b) Case 6: $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(g)$.

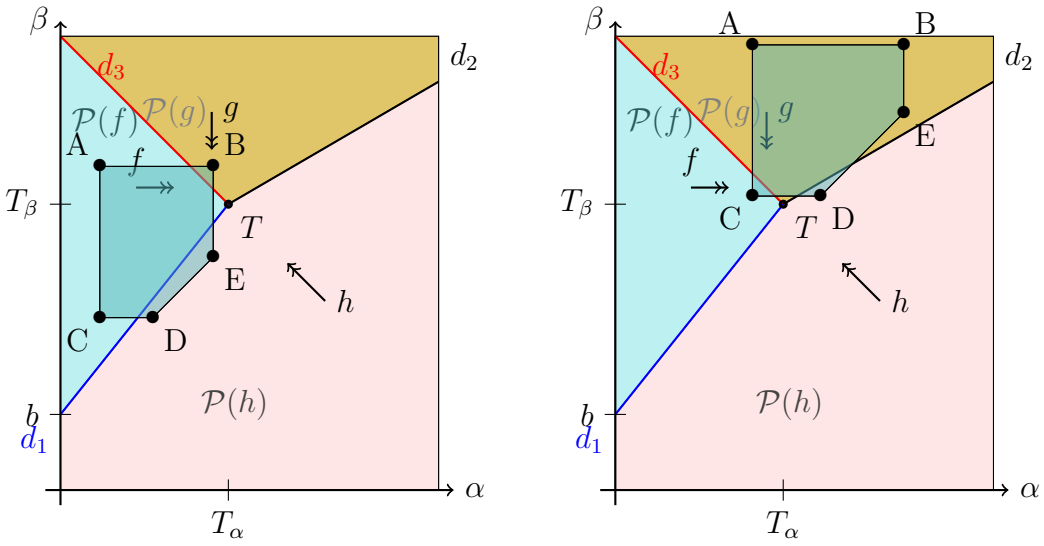


(c) Case 7: $B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(f)$.



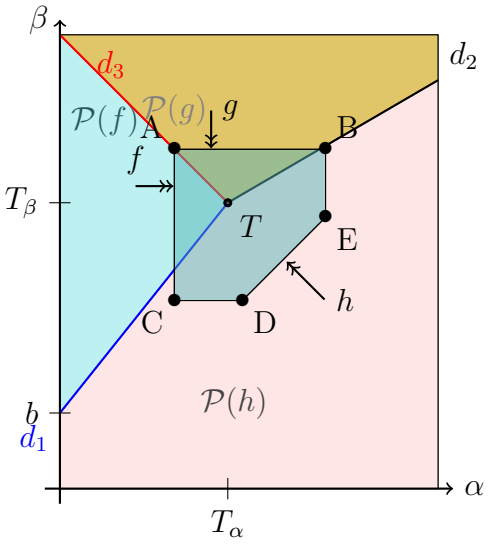
(d) Case 8: $D \in \mathcal{P}(f)$, and $E \in \mathcal{P}(g)$.

Figure 5.15: Summary of cases 5, 6, 7 and 8.



(a) Case 9: $M_\alpha \leq T_\alpha, B \in \mathcal{P}(g)$ and $E \in \mathcal{P}(h)$.

(b) Case 10: $m_\beta \geq T_\beta$, $C \in \mathcal{P}(f)$ and $D \in \mathcal{P}(h)$.



(c) Case 11: $T \in \mathcal{D}$.

Figure 5.16: Summary of cases 9, 10 and 11.

5.3.5 Conclusion and result in general cases

In this section, we sum up every found argsup of $\mu_{|\mathcal{D}}$ depending whether f or g has infinite coefficients, or depending on the signs of a and c. Tables 5.1, 5.2, 5.3, 5.4 and 5.5 are composed of three columns. The first one gives the conditions of the relative position of $\mathcal{D}$, the second one gives the value of a maximal point of $\mu_{|\mathcal{D}}$ and the third one gives the value of $\mu_{|\mathcal{D}}$ on these points.

▷ **When f or g have infinite coefficients**

- **when $f = +\infty$ and $c \geq 0$ (or $g = +\infty$):**

coordinates of a maximal point	value of a maximal point
(m_α, M_β)	$\min\{M_\beta - m_\alpha, c \cdot M_\beta + d\}$

- **when $f = +\infty$ and $c \leq 0$:**

Conditions	coordinates of a maximal point	value of a maximal point
$M_\beta \leq \frac{m_\alpha + d}{1-c}$	(m_α, M_β)	$\min\{M_\beta - m_\alpha, c \cdot M_\beta + d\}$
$m_\beta \leq \frac{m_\alpha + d}{1-c} \leq M_\beta$	$(m_\alpha, \frac{m_\alpha + d}{1-c})$	$\min\{\frac{c \cdot m_\alpha + d}{1-c}, c \cdot \frac{m_\alpha + d}{1-c} + d\}$
$\frac{m_\alpha + d}{1-c} \leq m_\beta$	(m_α, m_β)	$\min\{m_\beta - m_\alpha, c \cdot m_\beta + d\}$

Table 5.1: All solutions where $f = +\infty$.

- **when $g = +\infty$ and $a \leq 0$:**

Coordinates of a maximal point	value of a maximal point
(m_α, M_β)	$\min\{M_\beta - m_\alpha, a \cdot m_\alpha + b\}$

- **when $g = +\infty$ and $a \geq 0$:**

Conditions	coordinates of a maximal point	value of a maximal point
$\frac{M_\beta - b}{a+1} \leq m_\alpha$	(m_α, M_β)	$\min\{M_\beta - m_\alpha, a \cdot m_\alpha + b\}$
$m_\alpha \leq \frac{M_\beta - b}{a+1} \leq M_\alpha$	$(\frac{M_\beta - b}{a+1}, M_\beta)$	$\min\{\frac{a \cdot M_\beta + b}{a+1}, a \cdot \frac{M_\beta - b}{a+1} + b\}$
$M_\alpha \leq \frac{M_\beta - b}{a+1}$	(M_α, M_β)	$\min\{a \cdot M_\alpha + b, M_\beta - M_\alpha\}$

Table 5.2: All other solutions where $g = +\infty$.

▷ **When f and g are non-infinite functions**

- **When $a \geq 0$ and $c \geq 0$:**

Conditions	coordinates of a maximal point	value of a maximal point
$\frac{M_\beta - b}{a+1} \leq m_\alpha$	(m_α, M_β)	$\min\{M_\beta - m_\alpha, cM_\beta + d\}$
$m_\alpha \leq \frac{M_\beta - b}{a+1} \leq M_\alpha$	$(\frac{M_\beta - b}{a+1}, M_\beta)$	$\min\{\frac{a \cdot M_\beta + b}{a+1}, c \cdot M_\beta + d\}$
$M_\alpha \leq \frac{M_\beta - b}{a+1}$	(M_α, M_β)	$\min\{a \cdot M_\alpha + b, c \cdot M_\beta + d\}$

Table 5.3: All solutions when f and g are non-infinite functions and when $a \geq 0$ and $c \geq 0$.

- **when** $a \leq 0$ **and** $c \geq 0$**:**

Coordinates of a maximal point	value of a maximal point
(m_α, M_β)	$\min\{M_\beta - m_\alpha, a \cdot m_\alpha + b, c \cdot M_\beta + d\}$

- **when** $a \leq 0$ **and** $c \leq 0$**:**

Conditions	coordinates of a maximal point	value of a maximal point
$M_\beta \leq \frac{m_\alpha + d}{1 - c}$	(m_α, M_β)	$\min\{M_\beta - m_\alpha, a \cdot m_\alpha + b\}$
$m_\beta \leq \frac{m_\alpha + d}{1 - c} \leq M_\beta$	$(m_\alpha, \frac{m_\alpha + d}{1 - c})$	$\min\{\frac{c \cdot m_\alpha + d}{1 - c}, a \cdot m_\alpha + b\}$
$\frac{m_\alpha + d}{1 - c} \leq m_\beta$	(m_α, m_β)	$\min\{a \cdot m_\alpha + b, c \cdot m_\beta + d\}$

Table 5.4: All solutions when f and g are non-infinite functions and when $a \leq 0$.

- **when $a > 0$ and $c < 0$:**

Case	conditions	coordinates of a maximal point	its evaluation on μ
1	$h \leq f, g$ at A	(m_α, M_β)	$M_\beta - m_\alpha$
2	$f \leq g, h$ at B	(M_α, M_β)	$a \cdot M_\alpha + b$
3	$g \leq f, h$ at C	(m_α, m_β)	$c \cdot m_\beta + d$
4	$T_\beta \geq M_\beta$ $f \leq g, h$ at A $h \leq f, g$ at (B)	$\left(\frac{M_\beta - b}{a+1}, M_\beta\right)$	$\frac{a \cdot M_\beta + b}{a+1}$
5	$m_\alpha \geq T_\alpha$ $g \leq f, h$ at A $h \leq f, g$ at C	$\left(m_\alpha, \frac{m_\alpha + d}{1-c}\right)$	$\frac{c \cdot m_\alpha + d}{1-c}$
6	$f \leq g, h$ at C $g \leq f, h$ at D	$((1-c) \cdot m_\beta - d, m_\beta)$	$c \cdot m_\beta + d$
7	$g \leq h, f$ at B $f \leq h, g$ at E	$(M_\alpha, \max(M_\alpha, m_\beta))$	$a \cdot M_\alpha + b$
8	$f \leq g, h$ at D $g \leq f, h$ at E	$\left(\frac{b-d}{c-a}, \frac{b-d}{c-a}\right)$	$\frac{b \cdot c - d \cdot a}{c-a}$
9	$M_\alpha \leq T_\alpha$ $g \leq f, h$ at B $h \leq f, g$ at E	$(M_\alpha, (a+1) \cdot M_\alpha + b)$	$a \cdot M_\alpha + b$
10	$m_\beta \geq T_\beta$ $f \leq g, h$ at C $h \leq f, g$ at D	$\left(\frac{c \cdot m_\beta + d - b}{a}, m_\beta\right)$	$c \cdot m_\beta + d$
11	$T_\beta \geq T_\alpha, T_\beta \leq m_\alpha$ $T_\alpha \leq M_\alpha, T_\alpha \geq m_\alpha$	T	$T_\beta - T_\alpha$

Table 5.5: All solutions when f and g are non-infinite functions and when $a > 0$ and $c < 0$.

Conclusion

In this chapter, we provided an algorithm that computes the permissiveness function in time at most non-elementary for acyclic timed automata. We are facing a high worst-case complexity that depends on the number of cells used to represent the permissiveness function of the successors when computing the optimal strategy of the player. Despite this, we implement our algorithm, to observe its runtime on small examples. We also want to find other approaches where fewer cells are used to represent the permissiveness function. Therefore in the following chapters, we will present an implementation of the algorithm of this chapter for linear timed automata and two different approaches that compute approximate values of the permissiveness function.

Chapter 6

Robustness tools: forward and backward implementations.

In this chapter, two implementations are presented, whose common goal is to compute permissiveness.

The algorithm of the first implementation consists in exploring forward the timed automaton with a depth-search with no *a priori* knowledge of the strategy of the player and the opponent. Its goal is to compute an **approximate** value of the permissiveness of a **fixed** configuration. The motivation for this implementation is to provide a tool that computes, **even approximately**, a permissiveness of a configuration, for more general timed automata. This tool, implemented in Python, under **GPL Licence**, is available in the following github page: merce-fra/ECL-pyrobustness[1]. This **forward** algorithm explores the possibilities of choices for the player and the opponent, and then compares the different p-runs in order to deduce which one is the optimal one (for the player as well as for the opponent). The strategies of the player and the opponent are not known *a priori* and their choices are explored with brute-force methods.

The second implementation computes the **exact** value of the **permissiveness function** for all locations of the timed automaton with a **backward exploration**. This approach is the symbolic algorithm that we presented in Chapter 5 where we use the knowledge of the permissiveness function of each successor to compute the best current strategy. We have provided prototypes for this approach in Python, under the **GPL Licence**, available in the following github page: merce-fra/ECL-Symbolic-Implementation[2].

These two tools are now maintained on the respective following gitlab pages:

- https://gitlab.inria.fr/emclemen/numpyrobustness for the numerical implementation.

[1] full link: https://github.com/merce-fra/ECL-pyrobustness
[2] full link: https://github.com/merce-fra/ECL-Symbolic-Implementation

- ▷ https://gitlab.inria.fr/emclemen/formats-symbolic-tools for the symbolic implementation.

In this chapter, we present these implementations, their results and then compare their differences. In Section 6.1, we present our forward algorithm and its implementation in Python. In Section 6.2, we present the symbolic implementation of the algorithm proposed in Chapter 5, for linear automata, and compare it with our forward approach.

6.1 Numerical forward approach

In our first implementation, we want to compute an **approximate** value of the permissiveness of a **fixed** configuration. To do so, we want to explore all the p-runs. Nevertheless, there are an infinite number of possible p-runs so we sample all the possibles p-moves and delays that the player and the opponent can propose respectively.

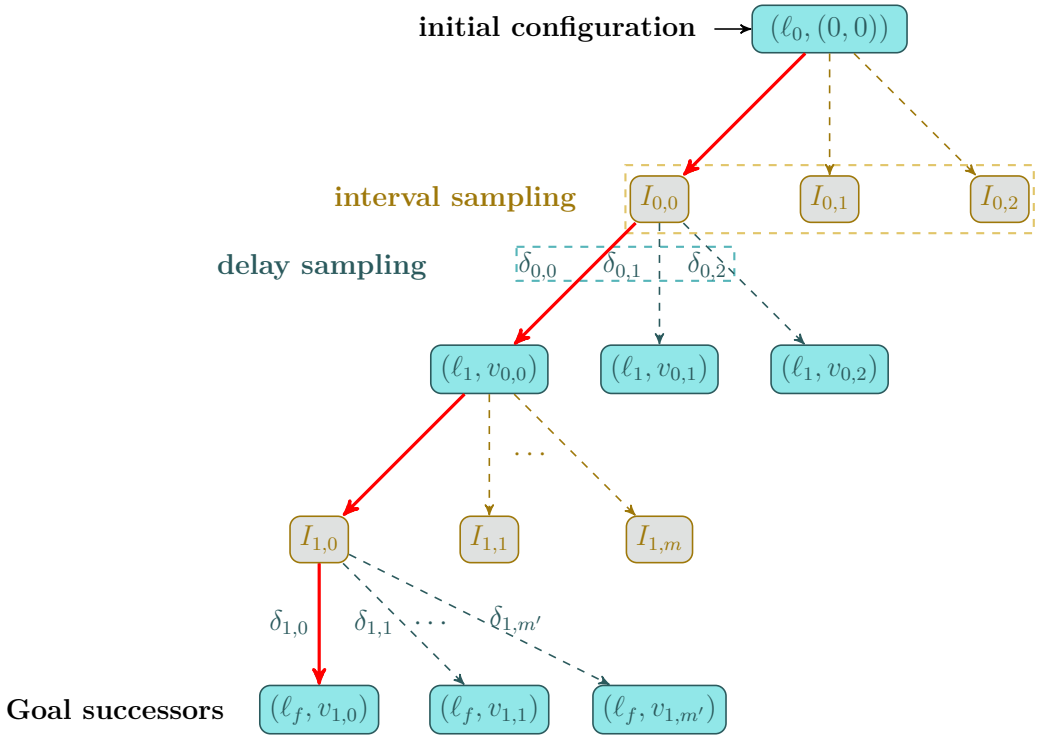
For the sampling, we fix two strictly positive integers p and q and use a sampling step $s_I = 1/p$ for the p-moves sampling and a sampling step $s_\delta = 1/(p \cdot q)$ for the delay sampling. In order to optimise which p-run is the optimal one, we use a backtracking algorithm. Let us present the main steps of this algorithm, illustrated in Figure 6.1:

1. We explore the timed automaton with a simple first trace (see Figure 6.1a), with simple (non-necessary optimal) choices and it will be our 'best current trace' where we choose the greatest intervals and the first possible delays.

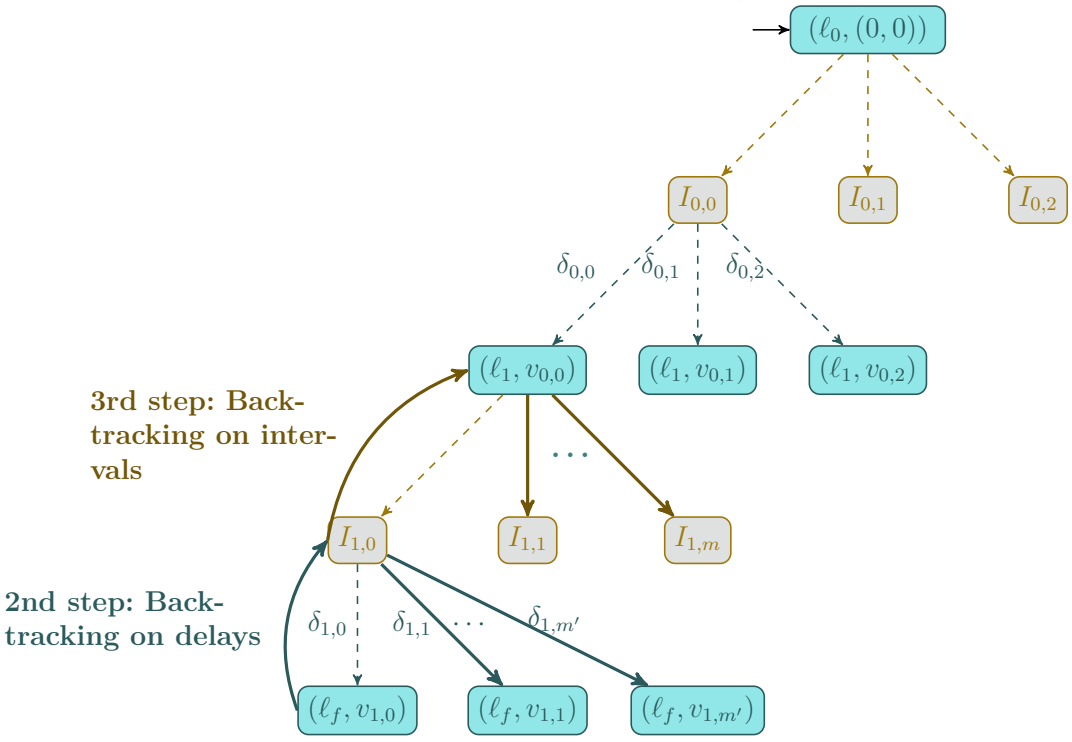
2. We sample the set of possible intervals and possible delays. As we have a forward approach our main issue is that we do not know the future permissiveness of our successors when proposing a p-move. Therefore we do not have the optimal strategy of the player and the opponent. To tackle this problem, we would like to consider all the possible intervals and delays. As we cannot explore infinite sets, we sample them with a fixed precision s.

   For example, if we consider a step $s_I = 1/2$ and we sample all the possible subintervals of $[0, 1]$, we propose the intervals $[0, 1/2]$, $[0, 1]$ and $[1/2, 1]$. If we sample the delays, with a sampling step $s_\delta = 1/2$, all the elements on $[0, 1]$, we consider the delays $0, 1/2$ and 1.

   In general, we proceed to a delay sampling because for general timed automata, the bound of the interval may not be the optimal choices for the opponent. In the case



(a) **First step**: a first trace exploration choosing $I_{0,0}$, $\delta_{0,0}$, $I_{1,0}$ and $\delta_{1,0}$.



(b) **Second step**: backtrack on delays $\delta_{1,1} \cdots, \delta_{1,m'}$. **Third step**: backtrack on intervals: $I_{1,1}, \cdots, I_{1,m}$.

Figure 6.1: Steps of our algorithm.

of linear timed automata, we optimise our algorithm by choosing the bounds of the proposed interval instead of doing a sampling of all possible delays.

3. We backtrack on delays and intervals (see Figure 6.1b), with a different objective in each case. In the case of intervals, we will try to select the one that maximises its permissiveness compared to our 'best current trace'. In the case of delays, we will try to find a trace whose permissiveness is lower.

4. We compute, for each exploration, a trace. If during the backtrack, we find a better trace, *i.e.* a choice of delays that minimise its permissiveness, or a choice of intervals that maximises it, then we will consider that trace as our new 'best current trace'.

In this section, we will present our algorithm more formally and the tool we provide in the github page merce-fra/ECL-pyrobustness in Python 3.8, as an object-oriented program. We proceed as follows:

- ▷ In Subsection 6.1.1 we present more formally our backtracking algorithm. We first explain how to sample the set of possible intervals and delays, then how to backtrack on the intervals and on the delays, and then our main function to compute the optimal trace.

Then, we focus on our concrete implementation in Python. We explain our choices of data structures and the optimisations we could apply so far.

- ▷ In Subsection 6.1.2, we present the classes implementing the following objects: timed automata, guards, and intervals.

- ▷ In Subsection 6.1.3, we present the classes allowing to apply the player and the opponent choices: p-moves and opponent strategies.

- ▷ In Subsection 6.1.4, we will present the core of our implementation: the *Backtracking* module. It contains the *Trace* class, which is used to model the trace in our algorithm, and the *Backtracking* class, which is used for backtracking.

- ▷ In Subsection 6.1.5, we compute an upper bound of the worst-case complexity of this algorithm

- ▷ In Subsection 6.1.6, we present our experimental results.

- ▷ In Subsection 6.1.7 we present a graphical interface designed to visualise the exploration and ease its understanding.

6.1.1 Algorithm

In this subsection, we describe our algorithm. We build a function that **takes as input** a timed automaton $\mathcal{A}$, a fixed start configuration (ℓ, v), an interval sampling step $s_I = 1/p$ and a delay sampling step $s_\delta = 1/(p \cdot q)$ and **returns** a trace of a run, such that its permissiveness is an approximate value of permissiveness function (for $\mathcal{A}$) from (ℓ, v). We decompose our algorithm into five functions:

- Two functions, named *moves_sampling* and *delay_sampling*, generate the set of possibles p-moves and delays.
- One function, named *compare_trace*, compares two traces by their permissiveness.
- Two mutually recursive functions, named *backtrack_interval* and *backtrack_delay*, explore the possible intervals and delays.

Sampling the set of possible intervals

For each available action a, let us denote $\mathcal{I}_a$ the set of all possible intervals. We cannot explore all the intervals of $\mathcal{I}_a$[3]. Indeed, this set contains, in general, an infinite number of possible intervals. In practice, we look at a finite subset $\mathcal{S}_{f,a} \subseteq \mathcal{I}_a$.

To do so, we consider the greatest possible interval the player can propose, defined as follows:

$$I_{max}(a) := [\alpha, \beta].$$

Let us consider a sampling step s_I. The set of sub-intervals we then use is denoted $\mathcal{S}_{f,a}$ and is defined as follows:

$$\mathcal{S}_{f,a} = [\alpha, \beta] \cup \{[\alpha', \beta'] \mid \exists k_\alpha, k_\beta \in \mathbb{N}, \alpha' = \alpha + k_\alpha \cdot s_I, \beta' = \alpha + k_\beta \cdot s_I, \alpha' \leq \beta' \leq \beta\}$$

The pseudo-code of the sampling of such p-moves is described in Algorithm 2.

[3]Except when $\mathcal{I}_a$ contains only a punctual interval.

Data: A timed automaton $\mathcal{A}$, a configuration (ℓ, v), a sampling step s_I
Result: A list of p-moves $(I_i, a)_{i,a}$

```
p-moves = [ ];
for every possible action a do
    Compute the greatest possible interval I_max(a);
    p-moves += [(I', a) | I' ∈ S_{f,a}];
end
return p-moves
```

Algorithm 2: *moves_sampling.*

Therefore, the set of intervals that have been considered in this function is not exactly the set of all possible intervals, but a sampling of intervals. However, we know the size of the proposed intervals. Therefore the imprecision of the size of the proposed intervals is s_I for each sampling.

Sampling the delays

The same problem arises when we want to explore the whole range of possible delays. If we consider a proposed interval I, all the elements $\delta \in I$ are possible. Our sampling method, called *delay_sampling* consists in sampling the interval I with a sampling step s_δ in order to obtain a **finite set of delays to consider**, again with imprecision.

Order relation between traces

The information from the different p-runs explored, which allows the permissiveness to be compared, is stored in traces. A trace is either:

- A sequence of tuples of the form (s, I, δ), where s is a configuration, I an interval, and δ a delay. The permissiveness of such trace is the size of the smallest intervals of this sequence. The permissiveness of a trace where this sequence is empty is infinite.
- An object called Empty trace, which corresponds to an initial trace.

The comparison function, called *compare_trace* applies to a current trace, denoted `current_trace`, and compares it with another trace, denoted `another_trace` and returns:

- A positive value if `another_trace` is the empty trace.

- 1 if (*resp.* -1) if the permissiveness of `current_trace` is greater (*resp.* smaller) than the one of `another_trace`. Therefore it returns a negative value if `another_trace` is the empty list [].

This establishes a simple way to compare traces. We will say that `current_trace` if *greater* (*resp. smaller*) than `another_trace` if this function returns a positive (*resp.* negative) value. Intuitively, the role of the empty trace is a placeholder to be replaced as soon as it is compared.

Backtracking on the intervals

Our main function is the interval backtrack, called *backtrack_interval*, which explores the sampled intervals and then calls the delay backtrack function, called *backtrack_delay*, on each interval. Here we present the pseudo code of the *backtrack_interval* in Algorithm 3.

Data: A fixed configuration (ℓ, v), a timed automaton $\mathcal{A}$, an interval sampling step s_I, a delay sampling step s_δ and a trace T_c
Result: A trace such that its permissiveness is an approximate value of the permissiveness of the initial configuration[a]

```
if ℓ ∈ Q_f then
    return T_c
end
best_trace ← Empty trace;
next_possibilities ← moves_sampling(A, (ℓ, v), s_I);
for p-move (I, a) in next_possibilities do
    minimal_trace ← backtrack_delay((ℓ, v), A, (I, a), s_I, s_δ, T_c);
    if minimal_trace.compare_trace(best_trace ) > 0 then
        best_trace ← minimal_trace
    end
end
return best_trace
```

Algorithm 3: *backtrack_interval.*

[a] See Subsection 6.1.6 for details.

Backtracking on the delays

The role of the function *backtrack_delay* is to consider all the delays in a sampled set of delays. It explores those delays to find one delay that minimises the permissiveness among the selected choices of delays. We present the *backtrack_delay* in Algorithm 4

Data: A configuration (ℓ, v), a timed automaton $\mathcal{A}$, a p-move (I, a), an interval sampling step s_I, a delay sampling step s_δ and a trace T_c
Result: The trace that minimises the permissiveness among the selected choices of delays for the p-move (I, a)

```
minimal_trace ← Empty trace;
for δ in delay_sampling((I, a), ℓ, s_δ) do
    next_config ← succ(ℓ, a, δ, v);
    T'_c ← T_c + (next_config, I, δ) ;
    new_trace ← backtrack_interval (next_config, A, s_I, s_δ, T'_c);
    if new_trace.compare_trace(minimal_trace) < 0 then
        minimal_trace ← new_trace
    end
end
return minimal_trace
```

Algorithm 4: *backtrack_delay.*

Main algorithm

Our final algorithm computes an optimal p-run and its approximate permissiveness. It simply applies the function *backtrack_interval* for the automaton $\mathcal{A}$, the initial configuration (ℓ, v), the interval sampling step s_I, the delay sampling step s_δ and the trace $T_c = [\,]$. It makes our first backtrack over the intervals begin, then it calls the backtrack of the delays ... until it founds the approximate optimal trace.

The choice to initialise the trace to $[\,]$ can be explained easily: if the initial location is in the set of goal locations, the returned trace is an empty list, which permissiveness values $+\infty$ by convention.

Let us now discuss the concrete implementation of our algorithm. Our algorithm has been implemented as an Object-Oriented program in Python 3.8. It is decomposed into **four main modules** that we detail in the following subsections, using simplified UML

class diagrams:

- A TimedAutomaton class for the implementation of timed automata.
- A data structure for p-moves, proposed by the player, and the delays that the opponent chooses.
- A Trace class that stores the trace informations.
- A Backtracking class that implements the backtracking algorithm.

More detailed UML graphs for the methods are presented in the Appendix A.1.

6.1.2 Timed automata, guards and intervals

Timed automata

The class TimedAutomaton, Guard and Intervals are illustrated in the UML graph on Figure 6.2.

Timed automata are modelled by the class **TimedAutomaton**, sub-classed from directed graphs of 'networkx' library. It also uses the **Edge** class. The **TimedAutomaton** class handles the logic of timed automata, such as initial and goal locations, transitions and associated guards and the dimension of the clocks space. This class can be exported and imported as a *JSON* file for easier tests and use. This implementation allows more general timed automata than single action timed automata, but requires them to be deterministic.

Guards

The guards used in this algorithm are **classical guards**. The implementation of timed automata relies on the **Guard** class to deal with the guard constraints of its transitions. In particular, the **Guard** class provides a method that computes, for each action, the maximal enabled interval, required in the backtracking algorithm when sampling the intervals. It also provides a method that verifies, for a given valuation v and delay δ, if $v + \delta$ satisfies the guard.

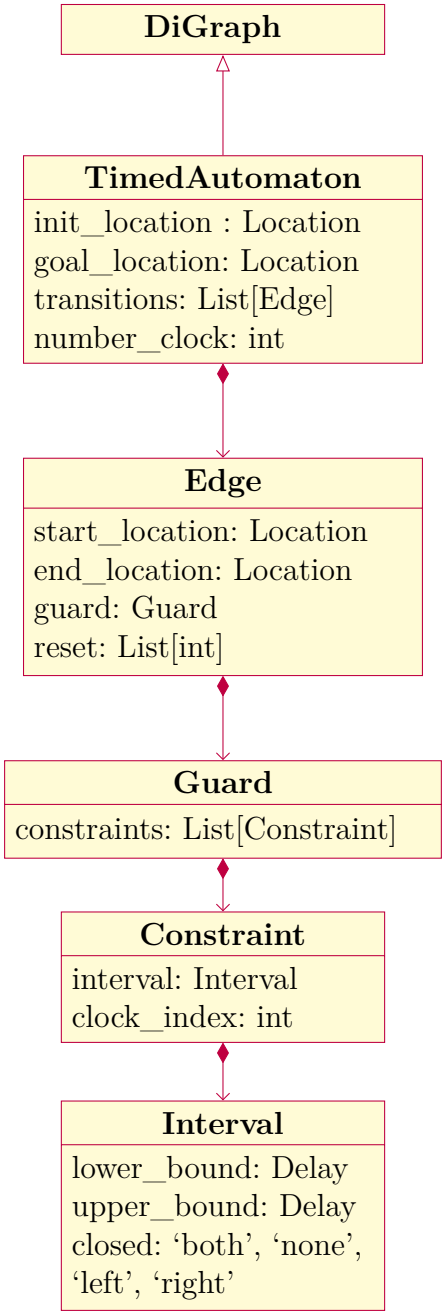


Figure 6.2: UML graph of TimedAutomaton.

Intervals

These intervals are built by giving the lower and upper bounds as rational or infinite numbers, to avoid floating point imprecision. The type of interval is indicating if the interval is open, closed or half-open (at right or left). For the same reason of avoiding imprecision, the values of the clocks can only be integers, rationals or the infinite.

6.1.3 P-moves and opponent strategy

P-move

The **P-Move** class, illustrated in the UML graph in Figure 6.3a, represents the proposed intervals and actions. The **P-Move** class can decompose the interval into sub-intervals to associate each sub-interval to a unique next location in the case of non single action timed automata.

The role of the **P-Move** class is to provide methods to compute the next configuration, given a p-move, an opponent strategy and an initial configuration.

Opponent strategies

We implemented several opponent strategies as functions that given a p-move and possibly a step sampling returns a list of enabled moves[4]. Indeed we provided two main methods. The first one, used for linear timed automata, chooses the bound of the intervals as the best delays and therefore returns two moves. In other cases, the best strategy of the opponent is not known so we provide a sampling of all possible delays and the strategy of the opponent returns a finite list of moves to test. Again, to avoid floating point imprecision, the step sampling should be rational.

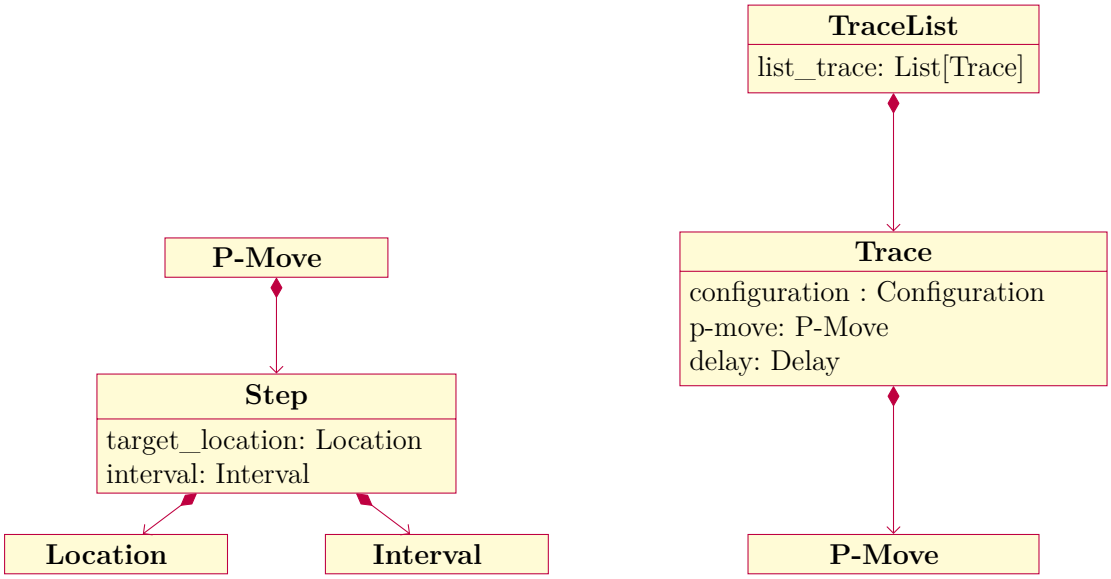
6.1.4 Backtracking and Trace

Trace

The aim of the classes **Trace** and **TraceList** is to represent traces, which contain the informations of a p-run. The trace is modelled as an object of the class **TraceList**, as a list of objects of the class **Trace**. Each object of the class **Trace** is represented by a

[4]these moves are implemented as p-move with punctual intervals

triplet (`configuration, p-move, delay`). As a result, the class **TraceList** depends on the class **Trace**, as illustrated in the UML class diagram in Figure 6.3b.



(a) UML class graph of P-Moves.

(b) UML graph of the trace objects.

Figure 6.3: UML graphs of P-moves and trace objects.

Backtracking

The **Backtracking** class, illustrated in the UML class diagram in Figure 6.4, is used to implement our main algorithm of backtracking. An object of the class **Backtracking** stores the timed automaton, the starting configuration and the samplings steps to apply to the player and the opponent strategies. The attribute `strategy_opponent` is a function that takes as input a `p-move` proposed by the player and a sampling delay step, `step` and returns a list of moves.

The **Backtracking** class provides the methods that implement *backtrack_delay* and *backtrack_interval*, the backtrack on delays and the backtrack on the intervals, and the main method that can apply the main algorithm of backtracking on any object of the class **Backtracking**.

Cycles handling

In the symbolic algorithms we have no proof that the sequence of suboptimal permissive functions tends to the permissiveness function in a finite number of steps. In our numeric

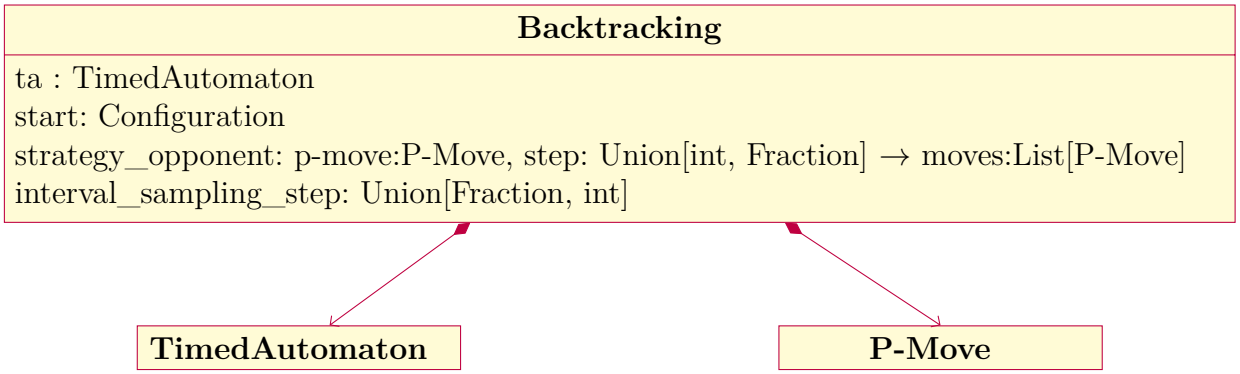


Figure 6.4: UML graph of Backtracking.

approach, we have no guarantee either that our algorithm will finish in a finite number of step if our timed automaton contains cycles. What we do in our implementation, to have **approximates results**, is to bound the number of time loops that can be taken. To do that, we can bound the number of loops taken and the maximal length of the trace in the parameters of the backtracking methods.

Our aim is to propose different increasing bounds and observe if the result seems to stabilise or not when the number of rounds grows. The goal could be to find a property, as a pattern where the number of needed rounds to stabilise seems to be linked on the values of constraints, or a counter-example where the timed automaton **seems** to have a sequence of suboptimal permissive functions that does not reach its limit. Then if a counter-example is found, it could be formally studied.

Optimisations on the sampling of intervals

When sampling all possible p-moves, the result is ordered with respect to the decreasing size of the intervals. Therefore we begin by proposing the largest possible interval. Then, when we choose one of the intervals, we check if its size is smaller than the permissiveness of our current best trace. If it is, it is useless to explore this trace because it will not lead to a better permissiveness. We then stop the exploration. Having sorted the intervals by size allows us to save computation time by avoiding unnecessary explorations.

Optimisations on the sampling of delays

We implement an optimisation when sampling the delays. If we found a delay that leads to a trace that has a negative permissiveness value, we choose this delay. Indeed, that

means that if the opponent chooses this delay, we reach a successor from which a target location cannot be reached. Therefore there is no reason to explore more delays for the opponent.

6.1.5 Worst-case complexity

Let us compute an upper bound of the complexity of our algorithm that computes an trace that, approximately, optimises the permissiveness.

Let us consider a timed automaton $\mathcal{A}$ and the largest interval I_{max} that can be proposed in $\mathcal{A}$. We count the number of intervals and delays explored at each step of the algorithm. Let us suppose that we use the sampling interval step s_I and the sampling delay step s_δ.

Computing the greatest interval

Before sampling the set of all possible intervals, we have to compute the greatest interval that can be proposed. Let us consider a classical guard, computing the greatest interval can be computed in $\mathcal{O}(n)$ where n is the number of clocks.

Exploring the intervals

For each transition, we explore at most $\left(\frac{|I_{max}|}{s_I}\right)^2$ intervals.

Exploring the delays

For each interval I that is proposed, we explore at most $\frac{|I|}{s_\delta}$ delays. We can bound this quantity as follows:

$$\frac{|I|}{s_\delta} \leq \frac{|I_{max}|}{s_\delta}.$$

Let us remark that for linear timed automata, we do not sample the delays with a sampling step but we just consider the bounds of the proposed interval.

Final worst-case complexity of the algorithm

To bound the size of the largest interval with a parameter of the timed automaton $\mathcal{A}$, we can consider the largest constant, defined in Definition 2.6, denoted $\mathcal{M}(\mathcal{A})$.

Let us denote T the maximal number of transitions in the timed automaton, n the number of clocks, and $s := \min(s_\delta, s_I)$ and $B := \max\left[\left(\frac{\mathcal{M}(\mathcal{A})}{s}\right)^2, n\right]$. B represents the complexity of computing the greatest interval and then sampling intervals.

Our algorithm can be executed in $\mathcal{O}\left(\left(B \cdot \frac{\mathcal{M}(\mathcal{A})}{s}\right)^T\right)$ for acyclic timed automata and $\mathcal{O}\left((2 \cdot B)^T\right)$ for linear timed automata.

6.1.6 Experimental results

In this subsection, we provide the runtime results we had by executing our algorithm on examples of timed automata given in this thesis and with a linear timed automaton with n transitions. All the experiments were run in a computer with the following specifications: Intel i7-9700 CPU at 3.00 GHz, 8Go of RAM, under Ubuntu 20.04. For the sake of simplification, we chose the same interval sampling step $1/p$ and delay sampling step $1/p$, that we denote here s.

When possible, we compute the error $\varepsilon = \text{Perm}_\varepsilon - \text{Perm}$ of our algorithm, where Perm is the exact permissiveness and Perm_ε is the approximate permissiveness computed by our numerical algorithm.

We run our algorithm on three examples, two are linear timed automata and one is an acyclic timed automaton. The first one is a simple linear timed automaton where we can vary the number of transition, in order to observe the runtime for a large number of transitions. An example for two transitions is illustrated in Figure 2.8. The second one is a linear timed automaton studied in this thesis, represented in Figure 2.10. We wanted to study the precision of our algorithm and its runtime with a timed automaton with reset. The third and last one is an acyclic timed automaton, represented in Figure 5.5, where we computed the permissiveness in Figure 5.6. We chose to study an acyclic timed automaton to provide some precision results when both strategies (player and opponent) were not necessarily optimal.

A first timed automaton with identical guards

The first timed automaton we use is a generalisation of the following timed automaton, studied in Chapter 2, illustrated in Figure 2.8 that we show again below for convenience.

We generalise this timed automaton with m transitions, by taking a timed automaton with a starting location ℓ_0, a target location ℓ_f, and m transitions, that contain the guard

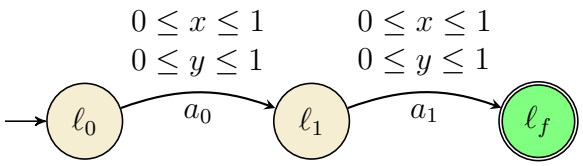
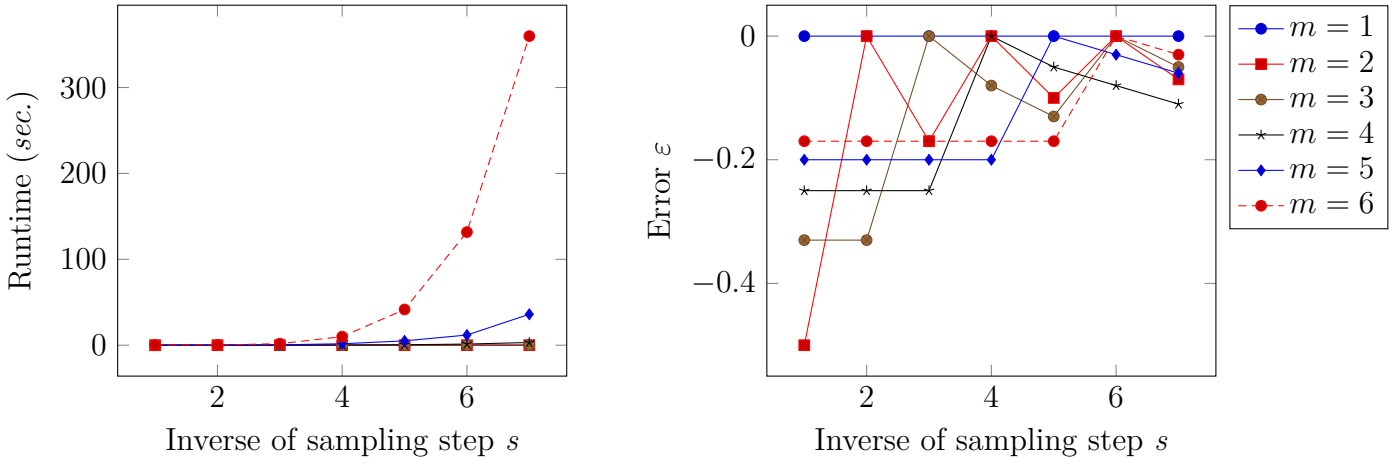


Figure 2.8: A timed automaton with 2 identical transitions presented in Subsection 2.2.2.

$0 \leq x \leq 1, 0 \leq y \leq 1$, and no resets. We run this example with sampling step s varying from $s = \frac{1}{1}$ to $\frac{1}{7}$ and the number of transitions varying from $m = 1$ to 7. We have plotted the runtime and accuracy results in two separate graphs, in Figures 6.5a and 6.5b. We ran the algorithm for the configuration $(\ell_0, (0, 0))$. It can be observed on the graph that



(a) Runtime results.

(b) Precision results.

Configuration used: $(\ell_0, (0, 0))$, exact permissiveness: $\frac{1}{m}$

Figure 6.5: Experimental results (Numeric approach) for the generalisation of the timed automaton of Figure 2.8 with m transitions.

the larger the number of transitions, the faster the algorithm grows exponentially with p, the inverse of the sampling step.

In terms of precision, we observe that on our examples, for all six timed automata tested, the algorithm returns an exact value of the permissiveness when the sampling step is a divisor of the permissiveness value. The error is always negative, so it means the algorithm made an under-approximation of the permissiveness. For example, for the two-transition timed automaton ($m = 2$), the permissiveness in $(\ell_0, (0, 0))$ is 1/2. It can

be seen that the algorithm returns an exact value for 1/2, 1/4 and 1/6 (and only for these values). We can also note that the error decreases on these examples when the sampling step decreases.

We will observe whether we find these same observations for the other following examples: an exponential runtime with respect to the precision, and an exact precision when calculated with a sampling step that divides the permissiveness.

Two timed automata with resets

The timed automaton we use in our next runtime examples was studied in Chapter 2. We represent it in Figure 2.10 that we re-show below for convenience. We run this example for

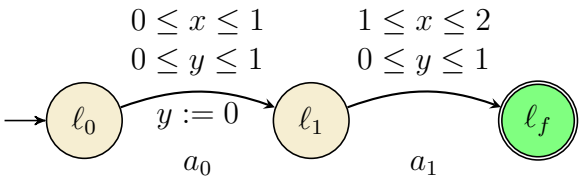
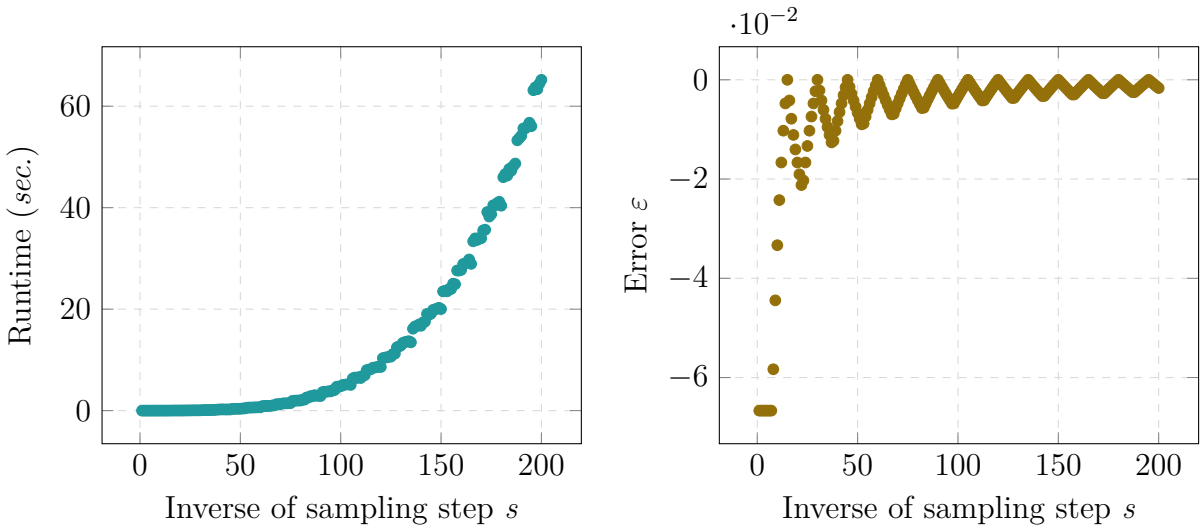


Figure 2.10: A timed automaton with a reset presented in Subsection 2.2.2.

the configuration $(\ell_0, (0, 0))$, on which the permissiveness function values 1/2. We run this examples for the sampling step that ranges from 1 to 1/200 and present the runtime and the error results in graphs in Figures 6.6a and 6.6b. The same results as in the previous example can be observed: the runtime grows when the sampling step decreases and the error, always negative, decreases with respect to the sampling step. The error is cancelled out for the sampling steps $\frac{1}{2 \cdot q}$ where $1 \leq q \leq 100$. The sign of the error can be explained by the fact that the opponent strategy is optimal for linear timed automata. The only non-optimal strategy is the one of the player, whose goal is to maximise permissiveness. It encourages us to study conditions to find a sampling step that give an accurate result, when the sampling step is not accurate. In the next example, we run a non-linear timed automaton. Thus the strategies of the opponent and the player are not optimal.

An acyclic example

Our last example is an acyclic timed automaton, represented in Figure 5.5. We computed its permissiveness and represented it in Figure 5.6.



(a) Runtime results.

(b) Precision results.

Configuration used: $(\ell_0, (0,0))$, exact permissiveness: $\frac{1}{2}$

Figure 6.6: Experimental results (Numeric approach) for the timed automaton of Figure 2.10.

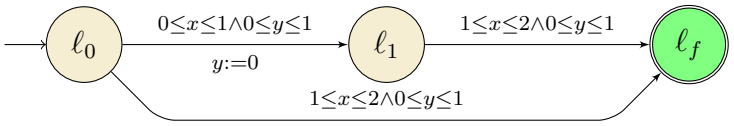


Figure 5.5: An acyclic timed automaton.

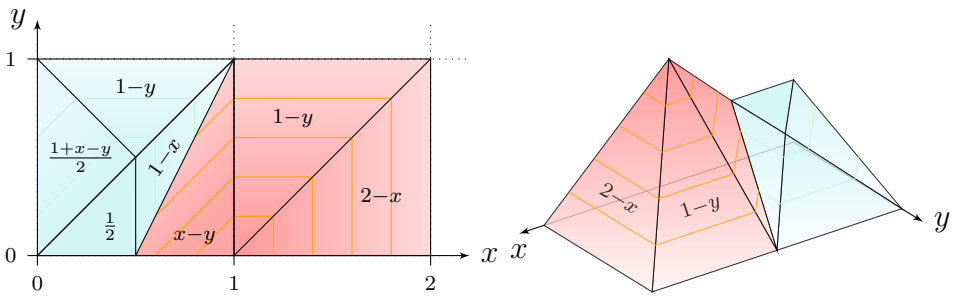
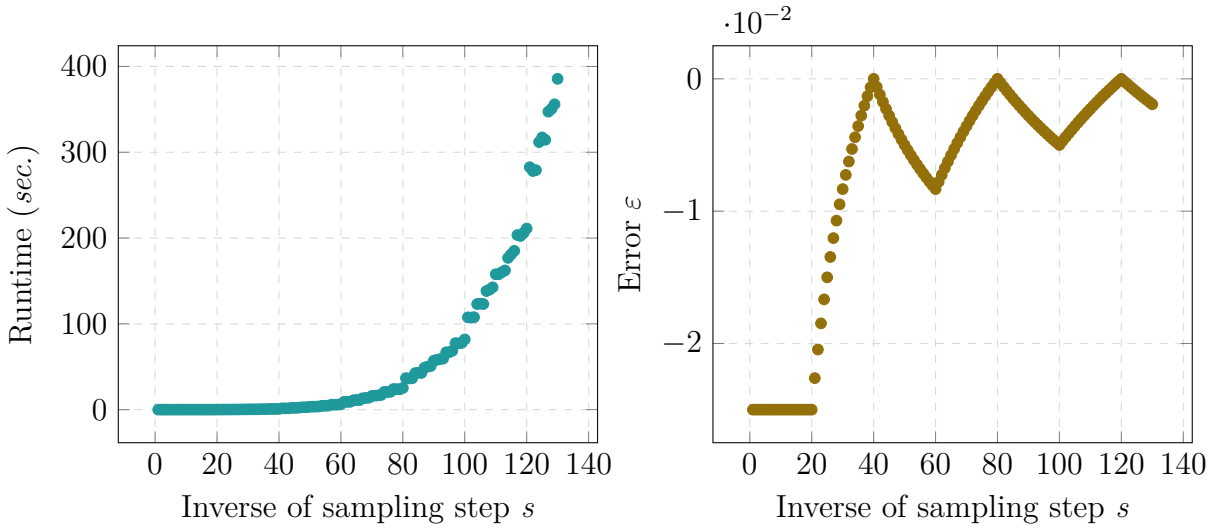


Figure 5.6: Permissiveness function for location ℓ_0 of the timed automaton of Figure 5.5.

We run this example for the configuration $\left(\ell, \left(\frac{1}{4}, \frac{7}{10}\right)\right)$, with a sampling step s varying from 1 to $\frac{1}{130}$. We show the runtime and precision results on Figures 6.7a and 6.7b.



(a) Runtime results.

(b) Precision results.

Configuration used : $\left(\ell, \left(\frac{1}{4}, \frac{7}{10}\right)\right)$, exact permissiveness: 1/40.

Figure 6.7: Experimental results (Numeric approach) for the timed automaton of Figure 5.5.

The delays proposed by the opponent are sampled in acyclic timed automata. The error in our example is still negative here and seems to tend to 0. The runtime is higher than in linear cases. This is expected as there are more possible paths and therefore more intervals and delays to sample, and therefore more p-runs to explore and compare. As in previous example, when the step sampling is a divisor of the exact permissiveness, here 1/40, the computation of the approximate permissiveness is accurate.

To conclude these examples among all the examples we run, we can observe common behaviours:

- The runtime results are represented with scatter plots. The line that could be deduced forms exponential function, which corresponds to the theoretical complexity we computed.
- The precision results are represented with scatter plots too. The resulting curves have the same shape, with oscillations, around a value which seems to converge

towards 0, whose amplitude decreases. In our examples and among the sampling steps tested, the exact permissiveness is reached for the sampling steps that divide the exact value of the permissiveness.

Let us now look at what can be concluded from the accuracy results and compare them with the theoretical aspects of the algorithm.

Precision of our algorithm

The computed permissiveness in our algorithm is approximated because we do not necessarily compute the optimal strategy. Indeed, we do not explore all the possible intervals and all the possible delays, but only a finite number of them.

An important issue is whether we compute an **over-approximation** (an approximation which is higher than the exact permissiveness) or an **under-approximation** (an approximation which is smaller than the exact permissiveness). In the case of **linear timed automata**, we have an optimal strategy for the choice of the delays and this approximation only affects the choice of the intervals. The permissiveness of the p-run we compute is therefore smaller than the permissiveness and our algorithm then provides an **under-approximation** of the permissiveness function.

In the **general case**, the permissiveness of the p-run we compute can be smaller or greater, because the strategy of the player **and** of the opponent may not be optimal. Indeed, giving a non-optimal strategy of the **player** (and an optimal one for the opponent) provide an **under**-approximation, and giving a non-optimal strategy of the **opponent** (and an optimal one for the player) provides an **over**-approximation. If both are non-optimal, we do not know if the computed permissiveness is under or over-approximated. At present, all experimental results give an under-approximation of the permissiveness, but more benchmarks could give us examples where the permissiveness is over-approximated.

We have not yet been able to study the stability of our algorithm in order to control its precision. Nevertheless, the experimental results give us some possibilities to explore.

1. The value where the error curve is cancelled out in our examples suggests that the sampling step to provide an accurate result could be a divisor of the exact value of the permissiveness (with a possible shift). Indeed, we sample the interval with a sampling step s_I. Let us recall the set of the sampled intervals.

$$\mathcal{S}_{f,a} = [\alpha, \beta] \cup \{[\alpha', \beta'] \mid \exists k_\alpha, k_\beta \in \mathbb{N}, \alpha' = \alpha + k_\alpha \cdot s_I, \beta' = \alpha + k_\beta \cdot s_I, \alpha' \leq \beta' \leq \beta\}$$

The size of these intervals are either $\beta - \alpha$, where $[\alpha, \beta]$ is the greatest possible interval, or a multiple of the sampling step s_I. If the interval $[\alpha, \beta]$ is not the optimal interval, in order to provide an interval of the size of the permissiveness, the sampling step has to divide the permissiveness.

2. In the experimental results of the previous paragraph, the error graph oscillates around 0, and the amplitude of these oscillations decreases. For linear timed automata, as it is an under-approximation, a good practice could be to compute the approximate permissiveness for several decreasing sampling step and take the upper-bound, in order to get a more accurate permissiveness. It can be explained for linear timed automata. If the sampling step s' is a divisor of s, we explore more intervals than with s, but we explore all the intervals we explore when choosing s. It is also observed in one of our examples of acyclic timed automata (see 6.7b). Nevertheless we are not able to explain it yet, as the choice of the delay is approximated too.

To conclude, our **experimental** results suggest, in order to decrease the error ε, to fix an strictly positive integer p and run the examples for a finite list of sampling steps of the form $\frac{1}{q \cdot p}$ where q is a strictly positive integer.

In order to provide **formal results**, we could adapt some theoretical results of the permissiveness function to the permissiveness function of **p-runs** and **strategies**. In Chapter 4, we proved that the permissiveness function is 2-Lipschitz on Win_ℓ. We cannot apply them directly as we do not compute the permissiveness, but an approximation of it, but we could try to extend this result to the permissiveness of p-runs. Another idea we could explore is to bound the error knowing that the coefficients of the affine functions of the permissiveness function, a piecewise-affine function, are **constant**.

Studying the precision of our algorithm is an important future work, as it could bound the error made by computing the approximation of the permissiveness.

6.1.7 Logging interface

Finally, in order to provide a graphical view of the trace, a logging interface has been implemented. It aims to visualise exploration and ease the understanding of the choices of

the players and opponent. We present the logger interface when running the backtracking algorithm for the timed automaton presented in Figure 2.10 for the initial configuration $(\ell_0, (0,0))$ and the sampling step $s = \frac{1}{2}$. The aim of the logger is to show the traces of the paths that have been explored by the backtracking algorithms. The logger interface allows to reveal traces that interested us.

The logger is presented as a table, containing sub-tables in each lines. The button 'Hide filtered out' hides or reveals the non-taken paths that were avoided thanks to the optimisations on intervals and delays.

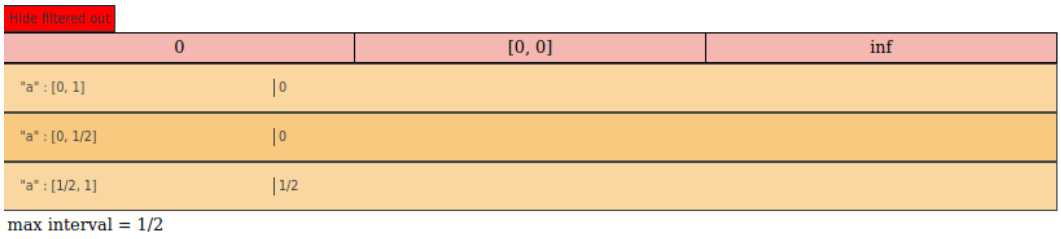
The main table, presented in Figure 6.8a, represents the logger when we do not reveal any sub-tables. It represents the first terms of all traces, with additional informations. The first lines indicates, from left to right, the current location (0)[5], valuations $((0,0))$ and the permissiveness of the current best trace $(+\infty)$. It also indicates all the possibles p-moves $(\text{"}a\text{"}, [0,1])$, $\left(\text{"}a\text{"}, \left[0, \frac{1}{2}\right]\right)$ and $\left(\text{"}a\text{"}, \left[\frac{1}{2}, 1\right]\right)$ and the final permissiveness we will obtain if we choose this interval: here respectively $0, 0$ and $\frac{1}{2}$.

In the Figure 6.8b, we reveal the first sub-tables when choosing the first proposed p-move, $(\text{"}a\text{"}, [0,1])$. The first sub-table revealed is a sub-table that lists the proposed delays in the first row (0 and 1) and the future permissivenesses if we choose each delays in the second rows: respectively 0 and 1. 'min delay = 0' indicates that the trace that minimises the permissiveness for the delays has a permissiveness that values 0.

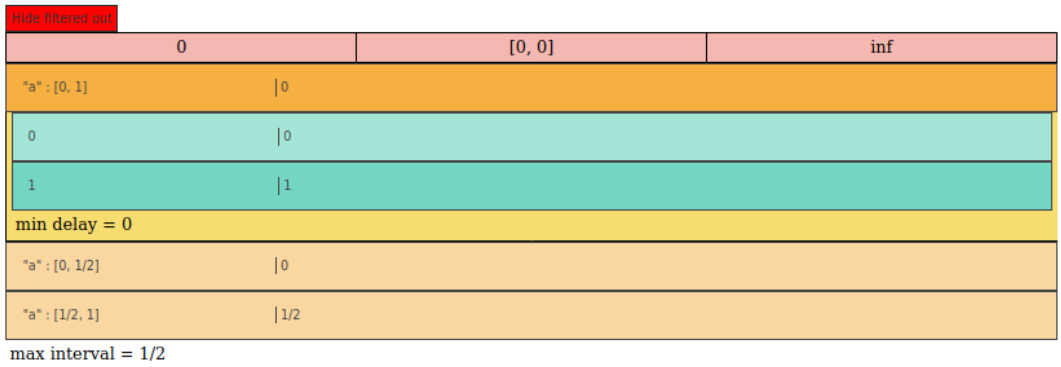
In the Figure 6.8c, we reveal the sub-table that is contained in the first line $(0, \mid 0)$. It reveal a table that gives the informations, from left to right, of the current location (1), valuation $(\left(\frac{0}{1}, 0\right))$, and the permissiveness of our current best trace (here 1, as we have chosen the interval $[0,1]$). The only line of this sub-table contains again the possible p-move, here $(\text{"}b\text{"}, [1,1])$ and the value of the permissiveness when choosing such interval (here 0). The 'max interval = 0' indicates the permissiveness of the trace that maximises the permissiveness among all the possible p-moves.

We can continue in this logger to reveal sub-tables to go further in the trace, until the goal location is reached (which is indicated). Non-taken p-moves are not shown, unless we click on 'Hide filtered out'.

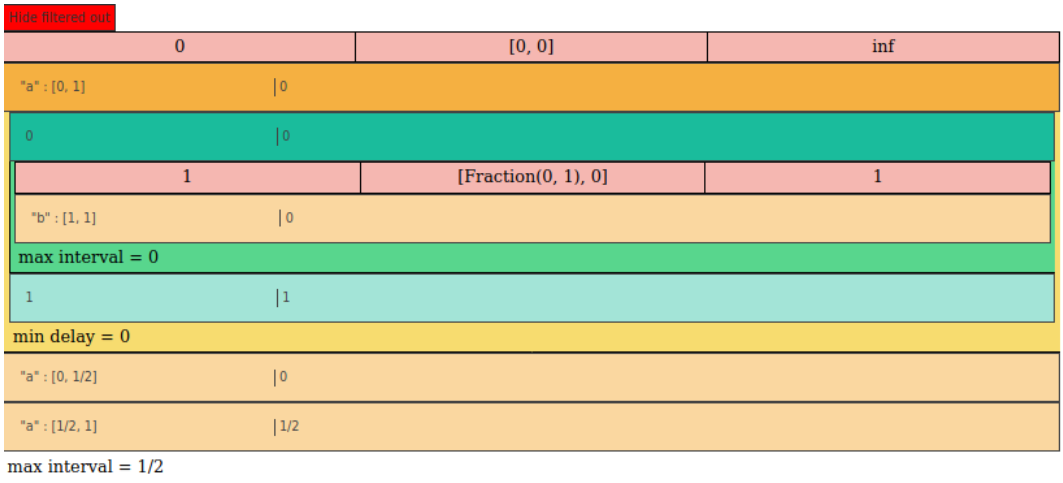
[5]For the sake of simplification, ℓ_i is denoted as i, a_0 is denoted as a and a_1 is denoted as b in the logger.



(a) Step 1: three p-moves are possible.



(b) Step 2: revealing trace when choosing the p-move $(a, [0, 1])$.



(c) Step 3: choosing the delay 1 and then the p-move $(b, [1, 1])$.

Figure 6.8: Screenshots of the logging interface of the exploration of timed automaton of Figure 2.10.

Conclusion and future work

In this subsection, we presented a forward algorithm that, given a fixed configuration, computes a trace whose permissiveness is an approximative value of the permissiveness of the configuration. It optimises the strategies of the player and the opponent, up to a precision. This algorithm runs in time $\mathcal{O}\left(\left(B \cdot \frac{\mathcal{M}(\mathcal{A})}{s}\right)^T\right)$ for acyclic timed automata, where T is the maximal number of transitions in the timed automaton, n the number of clocks, $s := \min(s_\delta, s_I)$ the minimum of the two sampling steps used for respectively intervals and delays and $B := \max\left[\left(\frac{\mathcal{M}(\mathcal{A})}{s}\right)^2, n\right]$.

We then presented its Python implementation, explained our choices and our runtime and precision results for some examples. The main difference with our symbolic algorithm presented in Chapter 5 is that we do not have an *a priori* knowledge of the strategy of the player and the opponent, except for the strategy of the opponent for linear timed automata. We also compute the permissiveness for a fixed configuration, whereas the symbolic algorithm presented in Chapter 5 computes it for an arbitrary configuration. As our forward algorithm can be run with no *a priori* knowledge on the strategy of the players or of the opponent, it can be extended to timed automata where we have no knowledge on the player or the opponent optimal strategies. The disadvantages of this approach is that only an approximate value is obtained. A major future work is studying the precision of our algorithm.

Other future works on this implementation could first focus to extend the models of timed automata supported by our tool. For example, we can extend the guards implementation, currently implemented as classical guards, to polyhedral guards, using a polyhedral library as Parma Polyhedral Library. Second future work could be to study the behaviour of cycles.

In the next section, we will present our implementation of the symbolic approach and compare the runtime results with the one of the forward numerical approach, for linear timed automata.

6.2 Symbolic backward approach

We now present the implementation of the backward approach to compute permissiveness. The algorithm of this approach was presented in Chapter 5 and here we describe its

implementations, provided in the github page merce-fra/ECL-Symbolic-Implementation, in Python 3.8, using the *pplpy* Python Binding to the C++ Parma Polyhedra Library. This implementation currently covers the cases of **linear timed automata** with **polyhedral guards**.

In this section, we explain our choices of implementation, the results we obtain and the future works that could be done to improve our implementation. We organised this section as follows:

1. First, we present Parma Polyhedra Library in Subsection 6.2.1.
2. Secondly, we recall the main steps of the algorithm in Subsection 6.2.2.
3. Thirdly, we discuss in Subsection 6.2.3 the technical aspects and the problems we had to solve to complete this implementation.
4. Fourthly and finally, we present in Subsection 6.2.4 some results and compare them to the forward approach presented in Section 6.1.

6.2.1 Parma Polyhedra Library

Parma Polyhedra Library[6] is a C++ library enabling the manipulation of polyhedra and linear expressions with coefficients in $\mathbb{Z}$. This library has been developped since 2001 by the University of Parma and BugSeng. The main current contributors are Roberto Bagnara, Patricia M. Hill, Enea Zaffanella and Abramo Bagnara. Numerous bindings are available in other languages, as Ocaml or Python. We use the Python binding named *pplpy*[7] (0.8.6 version) provided by Vincent Delecroix.

We use this library to represent the piecewise affine functions and the affine functions in our implementation. Indeed this library enables us to manipulate and represent affine functions and polyhedra with the respective pplpy classes Linear Expression and Polyhedron. Polyhedron class enables to represent a polyhedron with generators or linear constraints, themselves represented with the class Linear Expression. We use the representation with linear constraints. It also provides many operations with polyhedra. The main operations we use are checking membership to a polyhedron, the emptyness of a polyhedron and whether a polyhedron is included in another polyhedron, adding constraints to a polyhedron and computing intersections (with other polyhedra).

[6] https://www.bugseng.com/parma-polyhedra-library
[7] https://gitlab.com/videlec/pplpy

6.2.2 Principle of the algorithm

The goal of the algorithm, presented in Chapter 5, is to compute the **exact value of the permissiveness function** for any location in an acyclic timed automaton. We recall in this subsection the main steps of this algorithm, the full version being presented in Section 5.1. Our implementation only supports linear timed automata with polyhedral guards, but one of the main future works is to extend it to acyclic timed automata, as the algorithm in Chapter 5 can tackle them.

The main steps of the algorithm are summarised in Figure 5.2. The algorithm computes the permissiveness function with a backward explorer: the permissiveness of the goal locations is first computed by computing $\mathcal{P}_0$, then the permissiveness of its successors by computing $\mathcal{P}_1$, ...

When computing $\mathcal{P}_i$ of location ℓ, $v \mapsto \mathcal{P}_{i-1}(\ell', v)$ is a continuous piecewise affine function, where ℓ' is the successor of ℓ. Let us consider the tiling of polyhedra $(h_i)_i$ of $v \mapsto \mathcal{P}_{i-1}(\ell', v)$. Let h_α, h_β be two arbitrary polyhedra of $(h_i)_i$. Then, we compute:

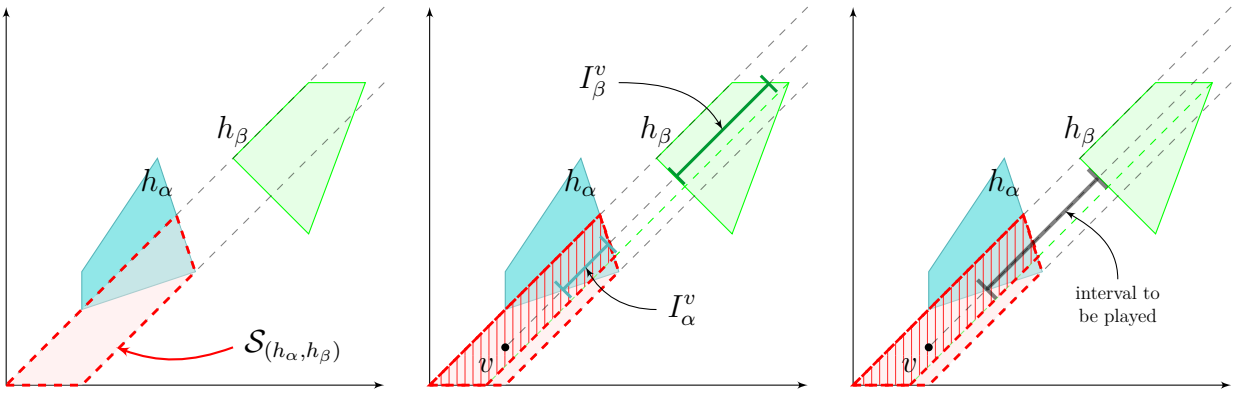
1. The set of valuations v such that there exists an enabled p-move $([\alpha, \beta], a)$ where $v + \alpha\,[\mathcal{C}_r \leftarrow 0] \in h_\alpha$ and $v + \beta\,[\mathcal{C}_r \leftarrow 0] \in h_\beta$ (see Figure 5.2a). We denote it $\mathcal{S}_{(h_\alpha, h_\beta)}$.
2. The set of enabled delays I^v_α and I^v_β as function of v (see Figure 5.2b).
3. Finally, the optimal bounds of the intervals to choose in these sets of enabled delays (see Figure 5.2c).

After having computed the optimal α and β for each couple (h_α, h_β), we compute the resulting permissiveness for each couple (h_α, h_β) and compute the maximum of all these permissiveness among all the possible couples. The resulting function is the permissiveness function.

6.2.3 Issues and implementation choices

In this subsection, we develop the implementation choices we did. In terms of technical aspects, the implementation uses:

- A representation of timed automata with polyhedral guards.
- A representation of piecewise affine functions with coefficients in $\mathbb{Q} \cup \{+\infty\}$.
- The minimisation, maximisation of piecewise affine functions.



(a) Step 1. (b) Step 2. (c) Step 3.

Figure 5.2: The three steps of our algorithm. Step 1: compute $\mathcal{S}_{(h_\alpha,h_\beta)}$. Step 2: compute expressions for I^v_α and I^v_β for any v. Notice that we consider a subpolyhedron (hatched zone) of $\mathcal{S}_{(h_\alpha,h_\beta)}$ because we had to refine it. Indeed the expression of I^v_β would be different for the lower part of $\mathcal{S}_{(h_\alpha,h_\beta)}$, since it ends up in a different facet of h_β. Step 3: select best values for α and β.

- ▷ The comparison of piecewise affine functions.

As for the numerical implementation, all UML graphs are simplified and more detailed UML graphs are presented in the Appendix A.2.

Let us detail these aspects in the following subsections. A final subsection will explain how to merge more than two polyhedra. This method is not yet implemented.

The representation of timed automata

Timed automata objects are used in our implementation to store the information about the timed automaton we will explore when applying our algorithm. The representation of the TimedAutomaton class does not differ from the one of the numeric implementation presented in Subsection 6.1.2, except the guards are **polyhedral guards**. Thanks to the *pplpy* library, the data structure of timed automata is simplified as we can see in the UML graph of TimedAutomaton class in Figure 6.9. Indeed, instead of implementing intervals and constraints, a guard is just defined as a polyhedron. As in the numeric implementation, we provide a *JSON* format to write the timed automaton in a more readable and convenient way.

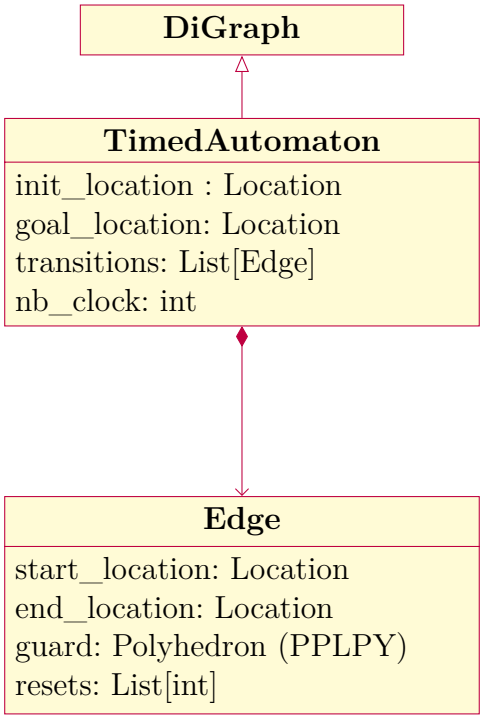


Figure 6.9: UML class graph of TimedAutomaton.

The representation of piecewise affine functions

Piecewise affine functions are used in almost every step of our algorithm: the permissiveness function and every function of the sequence of suboptimal permissive functions are piecewise affine functions over $\mathbb{R}_+^{|\mathcal{C}|}$, the bounds to the intervals of enabled delays I_α^v and I_β^v are piecewise affine functions, etc. We represent piecewise affine functions with the class *Spline*. Piecewise affine functions used here are **continuous** so we can use tiling of polyhedra to represent them. We represent them as a list of pairs of affine functions and cells of the form $(f, \mathcal{P})$. The affine function f is defined over the polyhedron $\mathcal{P}$ and is built with the class *SubSpline*. We represent these classes in the UML graph in Figure 6.10. In our model, affine function can have infinite coefficients, therefore these functions can have two forms:

- Either some coefficients of the affine function are infinite, then the affine function is implemented with the class *InfiniteExpression*. This class represents functions that are either $-\infty$ or $+\infty$. The sign of the function is decided with the boolean `is_positive`.
- Or the coefficients are rational and finite. We implement these functions with the

class *RationalLinearExpression* by associating a *pplpy* linear expression l with a divisor d. Indeed, the *Linear_Expression* class of *pplpy* only covers integer coefficients. Our implementation simplifies the coefficients of l and the divisor used d such that d is the least common multiple of all denominators of the rational coefficients of the affine function. This simplification is done in order to have a unique representation, and therefore compare easily two rational linear expressions.

These two classes are linked through an abstract class that we will not detail in Figure 6.10, for the sake of simplification. We implemented an optimisation to decrease the number

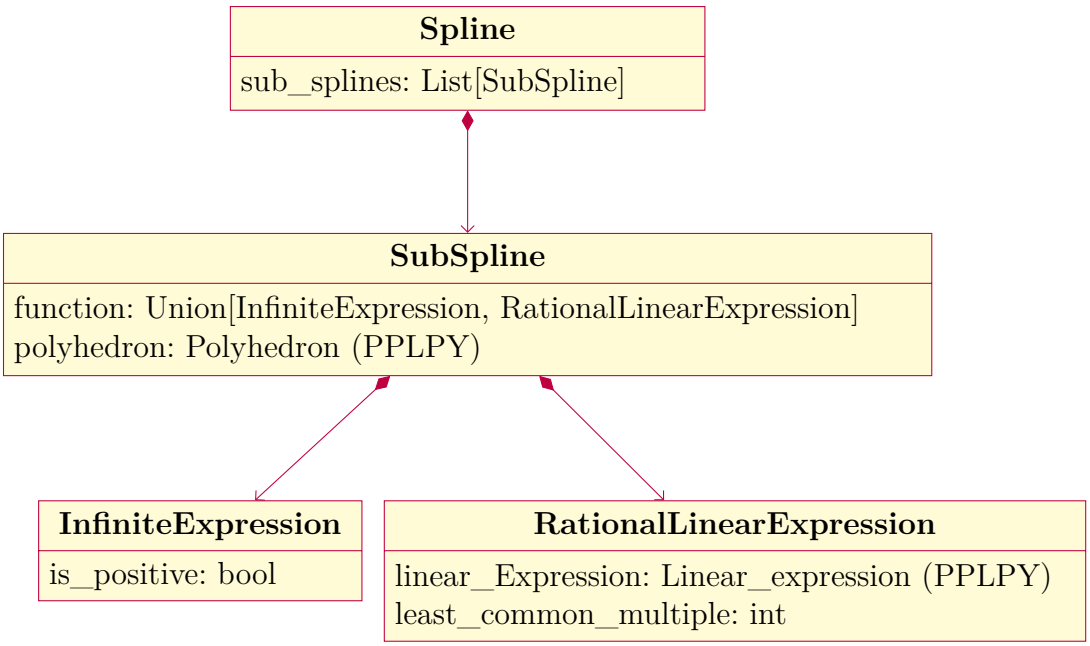


Figure 6.10: UML class graph of Piecewise affine functions.

of polyhedra used to representing the tiling of polyhedra of piecewise affine functions. Indeed, the complexity of our algorithm depends on this number of cells of the tiling of polyhedra of the permissiveness functions. The optimisation consists in choosing a fixed value of affine functions that we will **not** represent. Our choice was to not represent affine functions that are $-\infty$. Indeed, the set of valuations such that the permissiveness is $-\infty$ is not **necessarily** convex (see Figure 6.11). To represent this set as polyhedra, we need as least three polyhedra (see Figure 6.12a). The piecewise affine function of Figure 6.11 is therefore implemented as the one in Figure 6.12b. This optimisation has to be taken into account, for instance, when computing the maximum or the minimum of several piecewise affine functions, as explained in the next subsection.

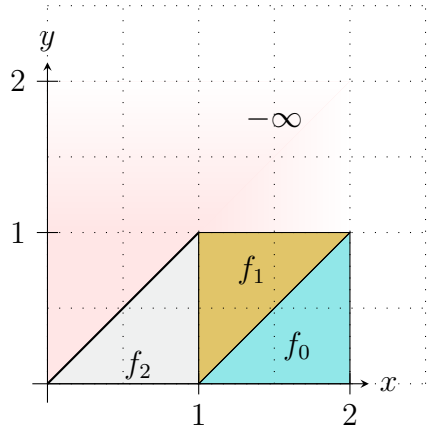
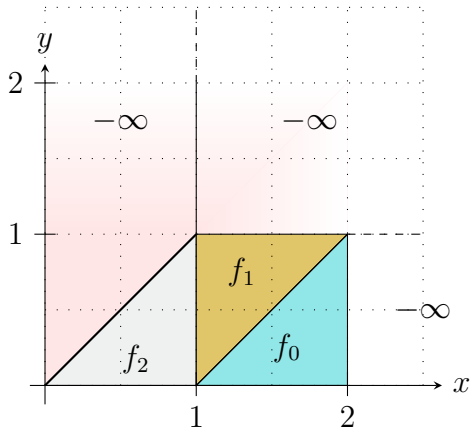
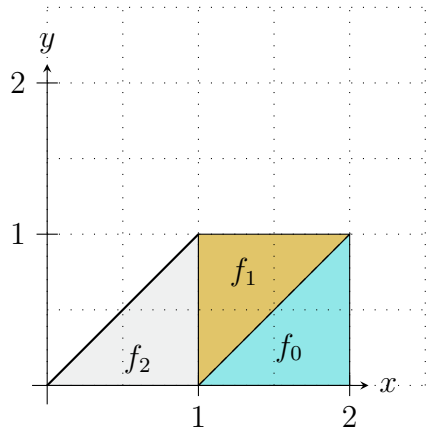


Figure 6.11: An example of piecewise affine function.



(a) Its representation if we represent the affine functions that values $-\infty$.



(b) Its representation without $-\infty$ functions.

Figure 6.12: Two representations of the piecewise affine function of Figure 6.11.

Minimising and maximising several piecewise affine functions

For the sake of simplification, we will explain here how we compute the maximum of **two** piecewise affine functions. Considering more than two functions, or the minimum of them can be extended trivially through pairwise comparison. Let us consider two piecewise affine functions F and G, defined over $\mathbb{R}_+^{|\mathcal{C}|}$. Indeed in our context the piecewise affine functions are defined over that set. Our goal is to compute $\max(F, G)$ over $\mathbb{R}_+^{|\mathcal{C}|}$.

Minimisation and maximisation of piecewise affine functions are used in several steps of our implementation:

1. When computing the bounds of the intervals of enabled delays I_α^v and I_β^v, these bounds are maximum and minimum of piecewise affine functions.

2. When implementing the optimisation results of Tables **??**, 5.1, 5.2, 5.3, 5.4 and 5.5 presented in Section 5.3.

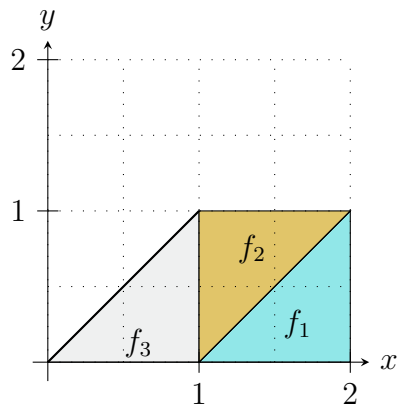
3. After computing an optimal interval to propose for each couple of cells (h_α, h_β), we compute the minimum of three affine functions, that are the size of the interval, and the future permissiveness of the successors, when choosing for the delay the bounds of the interval. The result is then a piecewise affine function. In order to compute the permissiveness function, we have to compute the maximum of these piecewise affine functions for all cells (h_α, h_β).

4. When considering acyclic timed automata, several successors may have to be considered. We should compute the permissiveness functions of the successors and compute the maximum of all these permissiveness functions to choose the transition to choose. This application is an ongoing work.

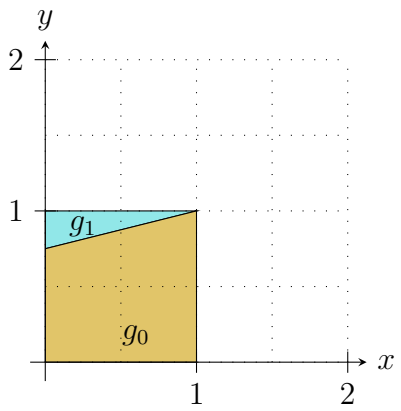
Let us explain the issues of implementing these two methods. First, let us pick an example of two piecewise affine functions f and g described in Figure 6.13a and 6.13b. Let us denote respectively the couple of cell-affine functions of the piecewise affine functions f and g by $((\mathcal{P}_0, f_0), (\mathcal{P}_1, f_1), (\mathcal{P}_2, f_2))$ and $((\mathcal{P}_3, g_0), (\mathcal{P}_4, g_1))$. We recall that the cells where the affine function is $-\infty$ are not represented.

At first sight, one might want to look at the intersection of each cells, and split each cells whether the first affine function is greater then the second one. More formally:

1. For each piece $\mathcal{P}_i$ among $\mathcal{P} := [\mathcal{P}_0, \cdots, \mathcal{P}_5]$, we look if there exists another polyhedron $\mathcal{P}_j \in \mathcal{P} \setminus \{\mathcal{P}_i\}$ such that $\mathcal{P}_i \cap \mathcal{P}_j \neq \varnothing$.



(a) Function f where $f_0(x,y) = 2 - x$, $f_1(x,y) = 1 - y$ and $f_2(x,y) = x - y$.



(b) Function g where $g_0(x,y) = \frac{1-x}{2}$, $g_1(x,y) = 1 - y$.

Figure 6.13: Example of two piecewise affine functions.

2. If the intersection of two polyhedra $\mathcal{P}_i$ and $\mathcal{P}_j$ is not empty, consider the associate affine functions f' and g' and split the intersection of $\mathcal{P}_i \cap \mathcal{P}_j$ with the linear constraint $f' \geq g'$ and $f' \leq g'$.

If we applied this procedure, we would end up with a wrong result. Indeed, if the first function intersects zones where the other function is $-\infty$, their intersection will not be taken into account. For instance, we represented in Figure 6.15a the result if not taking these cells into account.

We have a wrong result for two reasons. First, we did not take into account the cells where the affine function is $-\infty$. That can be solved by considering that if a cell does not intersect another cell, it is because it intersects a cell where the affine function is $-\infty$. Secondly, the tiling of polyhedra is not the same therefore, the cell associated to g_0 will intersect another cell, the one associated with f_3 and we will only compare it over $\mathcal{P}_3 \cap \mathcal{P}_0$.

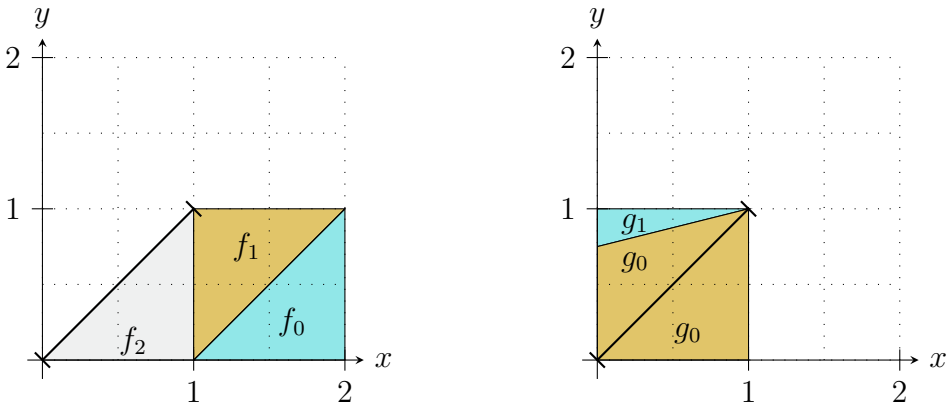
To tackle this issue, we combine the two tiling of polyhedra of each piecewise affine function, in order to have the same tiling of polyhedra in both piecewise affine functions. For instance the functions of Figure 6.13 will be represented by the functions described in Figure 6.14a and 6.14b.

Then, when considering the same tiling of polyhedra, we can proceed as follows:

- If two cells intersect, we consider their associated affine functions, for example f_3 and g_0 and we split the intersections of the cells into two cells by adding a linear constraint. For the first one we add the constraint $f_3 \geq g_0$ and associate this cell

to the affine function f_3. For the second one we add the constraint $g_0 \geq f_3$ and associate this cell to the affine function g_0.

- If a cell does not intersect any cell, we directly add it with its associated affine function. Indeed it means that this cell intersects a cell associated with an affine function that is $-\infty$.



(a) New overtiling of f.

(b) New overtiling of g.

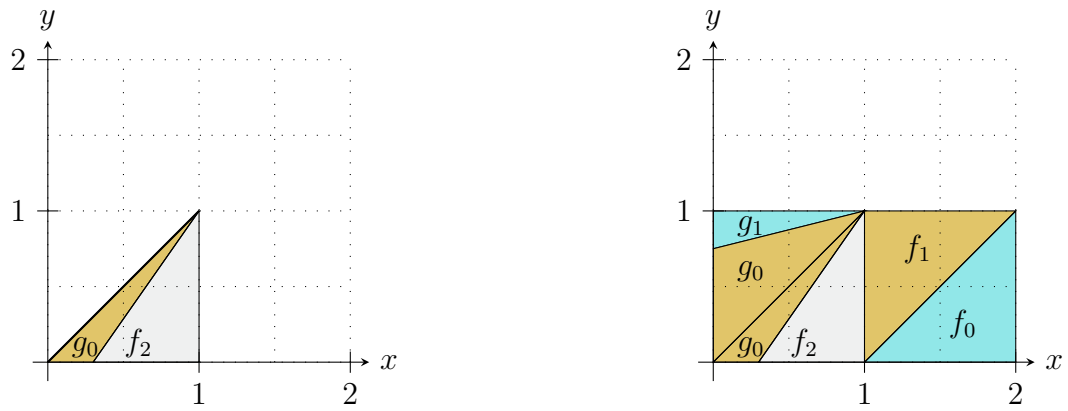
Figure 6.14: Over-tiling of f and g in order to apply our maximisation algorithm.

This solution over-tiles the cells of each piecewise affine function. Thus it may increase the number of cells. Let us consider the previous examples of Figure 6.13. The maximum $\max(f, g)$ is a piecewise-affine function described in Figure 6.15b. We can see that the polyhedron associated if g_0 could be merged. Merging as much polyhedra as possible could be a great optimisation to speed up our implementation.

Comparing several piecewise affine functions

As we have seen in the previous subsection, the representation in terms of list of tuples of polyhedra and affine functions is not unique in general. The piecewise affine functions in Figure 6.13b and 6.14b are the same, but their tiling of polyhedra are not the same.

Having a unique representation would be useful to easily check in unit tests if the obtained piecewise affine function is the one we expected. On top of that, having a unique representation would help extending the implementation for acyclic timed automata by checking is the sequence of suboptimal permissive functions has reached its fix point, by checking equalities between two piecewise affine functions.



(a) Wrong result when applying the naive procedure.

(b) Correct result with over-tiling.

Figure 6.15: Maximisation of the two piecewise affine functions f and g of Figure 6.13: wrong method (Figure 6.15a) and correct method with over-tiling (Figure 6.15b).

Let us remark that the current method that checks equalities between two piecewise affine functions is **sensible to the tiling**: for two piecewise affine functions to be checked as equal, they need to have the same tiling of polyhedra. At first sight, to check the equality between two piecewise affine functions, one would be tempted to look at the pairs whose affine functions are equal, and compute the union of the associated polyhedra together. However, the union of several polyhedra is not necessarily a polyhedron.

[BFT01] provides an algorithm for computing the convex hull of polyhedra and for checking that the union of two polyhedra is indeed convex and therefore corresponds to their convex hull. However, it does not provide an algorithm for more than two polyhedra.

This problem is not trivial and one cannot apply the algorithm of [BFT01] pairwise. Consider a simple example in Figure 6.16, where we divide the space into 4 square polyhedra, and associate them with the same affine functions. Depending on which polyhedra we start with, their union will not correspond to their convex hull. If we begin with the two left polyhedra, when these are merged, if we then check if the union with any right polyhedra is a polyhedron, the answer is no. We did not tackle the problem of checking if the union of $n \geq 3$ polyhedra is a polyhedron.

Checking efficiently the equalities between two piecewise affine functions is an **ongoing future work**. A first possibility given two piecewise affine functions $(\mathcal{P}_i, f_i)$ and $(\mathcal{Q}_i, g_i)$ would be:

1. To compute a representation taking into account the two tilings as in Figure 6.14

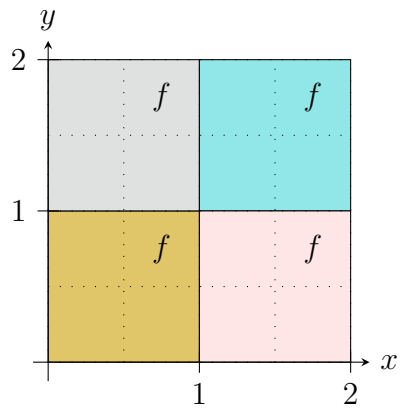


Figure 6.16: A piecewise function where all polyhedra could be merged.

such that the two functions have the same polyhedra partition.

2. Then, to check if the two resulting over-cut piecewise affine functions are equal.

We could also try other methods, such as comparing the values of the permissiveness function on edges of each polyhedra.

Merging (some) polyhedra

As explained earlier, one way to speed up our computation is to reduce the number of cells. Indeed, the complexity of the algorithm grows with the number of polyhedra used to represent our piecewise affine functions.

The first optimisation described above consisted of removing the polyhedra associated with the $-\infty$ function. The second improvement would be to merge polyhedra associated with **the same affine functions** and **whose union is a polyhedron**. As stated in the previous subsection, if there are only two polyhedra, the algorithm is presented in [BFT01] and can be performed in polynomial time. If there are more than two polyhedra, we could not provide an algorithm. However, we can propose a heuristic to solve this issue, but does not guarantee to systematically succeed in merging our polyhedra.

This heuristic would first list all the polyhedra that are assigned with the same affine function and then compute, for each list of polyhedra, the convex hull of these polyhedra. Then, the resulting piecewise affine function is the function that associates the convex hull to the associated affine functions. We compare this function to the original one with the procedure described previously. If the two functions are equals, it means that for each affine function the union of their associated polyhedra formed a polyhedron. If not,

there exists a set of polyhedra whose union was not a polyhedra and we keep the overcut representation of the function.

We can compute the convex hull and compare immediately the result for each associated function. We should evaluate the respective performance of the two procedures because comparing piecewise affine functions can be costly.

In the next section, we present our experimental results. First we present the results of our graphical tool, written in Python, that provides a graphical representation of the results of our algorithm. It encourages us to use the previous procedure to merge polyhedra as for each affine function the union of their associated polyhedra forms a polyhedron. Secondly, we present our runtime results.

6.2.4 Experimental results

In this section, we present our results. All the experiments were run in a computer with the following specifications: Intel i7-9700 CPU at 3.00 GHz, 8Go of RAM, under Ubuntu 20.04.

First we present in Subsection 6.2.4.1 the permissiveness function that our implementation computes and compare the over-tiling to the theoretical and optimized computation. Secondly, we present in Subsection 6.2.4.2 the runtime results and compare some to the one obtained in the numeric implementation.

6.2.4.1 Qualitative results

We implemented a Python tool to provide a graphical view (in *.tex* format) of our obtained permissiveness function for **two-clocks** timed automata with only bounded cells. We restricted to two-clocks timed automata to stay in 2D-representation and we can only managed bounded polyhedra yet because we use the coordinate of the edge of polyhedra to represent them, therefore unbounded polyhedra cannot be tackled yet, as they are represented with point and rays in *pplpy*.

The examples we present in this section are the permissiveness of the timed automata of Figure 2.8, 2.10, 5.3, that we re-show below for convenience.

Our graphical tool provides a picture and a table for each permissiveness function. The picture represents the cells of the piecewise affine function and the bottom table links the cells to an affine function. Let us remark that, as before in the implementation, we do not represent the cells associated with a function $-\infty$. For instance, in Figure 6.17a, the

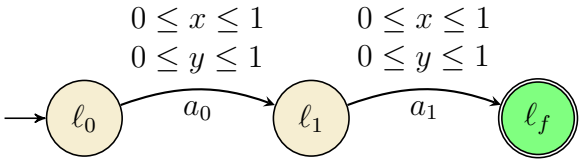


Figure 2.8: A timed automaton with two identical transitions presented in Subsection 2.2.2.

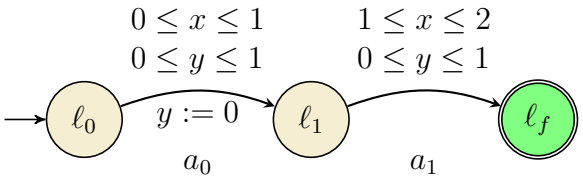


Figure 2.10: A timed automaton with a reset presented in Subsection 2.2.2.

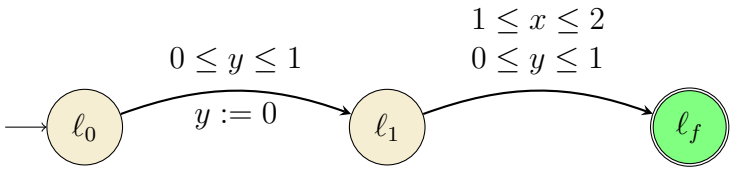


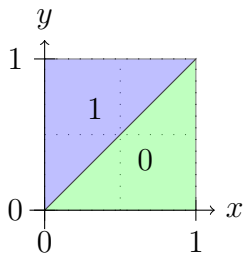
Figure 5.3: A timed automaton similar as the one of Figure 2.10 where the guard on the first transition has been slightly extended.

permissiveness function is defined as follows:

$$\begin{array}{rcl} [0,1]\times[0,1] & \to & \mathbb{R}_+ \\ (x,y) & \mapsto & \min\left(1-x, 1-y\right) \end{array}$$

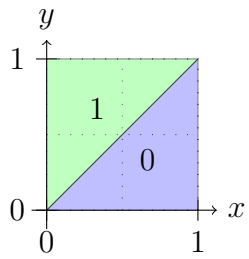
This permissiveness function is represented with 2 cells. We represent in the following Figures 6.17, 6.18 and 6.19 the graphical results obtained when computing the permissiveness of the timed automaton of Figures 2.8, 2.10, 5.3.

We can compare these results in Figure 6.17a, 6.18 and 6.19 with the theoretical permissiveness. The obtained permissiveness functions are accurate and we can observe an over-tiling in Figure 6.18b and 6.19b. For each associated affine function, we can observe that the union of the cells corresponds to the convex hull of these cells. These results encourage us to study ways to reduce the over-tiling of our algorithm and of our implementation.



Permissiveness	Associated cells
$(-y+1)/1$	1
$(-x+1)/1$	0

(a) Permissiveness at location ℓ_1 of the timed automaton of Figure 2.8.



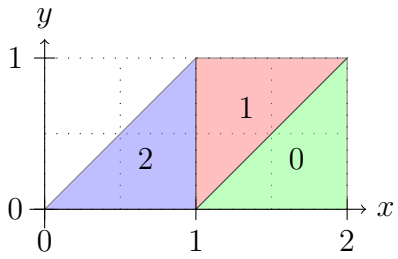
Permissiveness	Associated cells
$(-x+1)/2$	0
$(-y+1)/2$	1

(b) Permissiveness at location ℓ_0 of the timed automaton of Figure 2.8.

Figure 6.17: Permissiveness functions of the timed automaton of Figure 2.8.

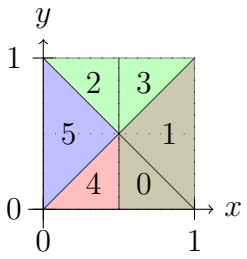
6.2.4.2 Runtime results for timed automaton studied in this thesis

Here we present in Table 6.1 the runtime results of the computation of the permissiveness function for several examples. We restricted in the previous section to simple examples, in order to keep the graphic representation readable. In this section, we will provide the runtime for the examples of the previous section and for additional timed automata (more transitions, three-clocks...).



Permissiveness	Associated cells
$(x-y)/1$	2
$(-x+2)/1$	0
$(-y+1)/1$	1

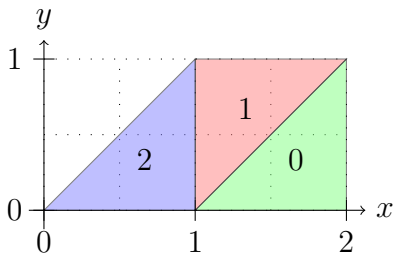
(a) Permissiveness at location ℓ_1 of the timed automaton of Figure 2.10.



Permissiveness	Associated cells
$(x-y+1)/2$	5
$(-y+1)/1$	2, 3
$(1)/2$	4
$(-x+1)/1$	0, 1

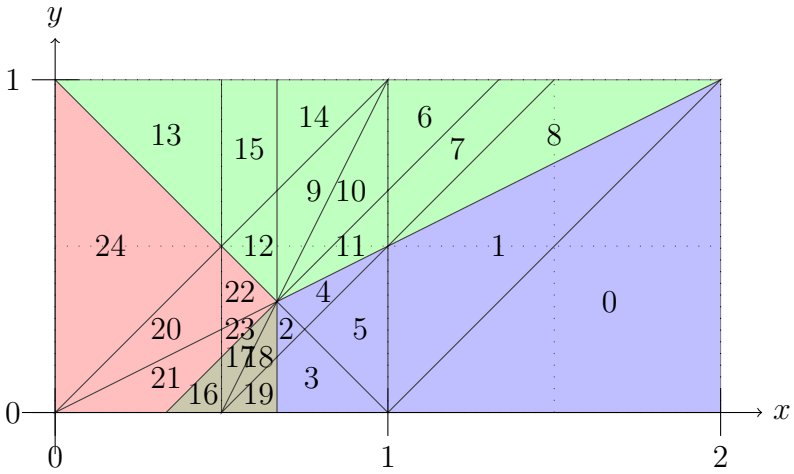
(b) Permissiveness at location ℓ_0 of the timed automaton of Figure 2.10.

Figure 6.18: Permissiveness functions of the timed automaton of Figure 2.10.



Permissiveness	Associated cells
$(x-y)/1$	2
$(-x+2)/1$	0
$(-y+1)/1$	1

(a) Permissiveness at location ℓ_1 of the timed automaton of Figure 5.3.

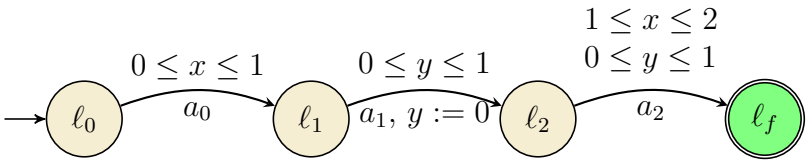


Permissiveness	Associated cells
$(-x+2)/2$	$0, \cdots, 5$
$(-y+1)/1$	$6, \cdots, 15$
$(x-y+1)/2$	$20, \cdots, 24$
$(2)/3$	$16, \cdots, 19$

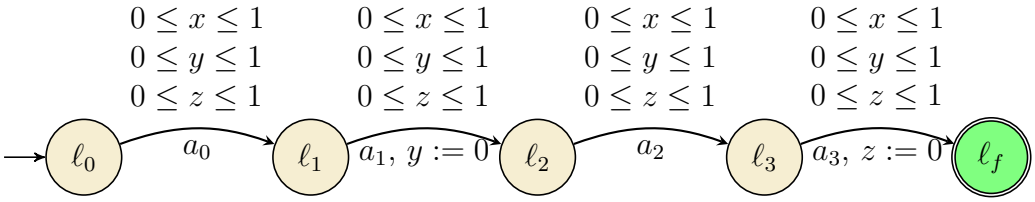
(b) Permissiveness at location ℓ_0 of the timed automaton of Figure 5.3.

Figure 6.19: Permissiveness function of the timed automaton of Figure 5.3.

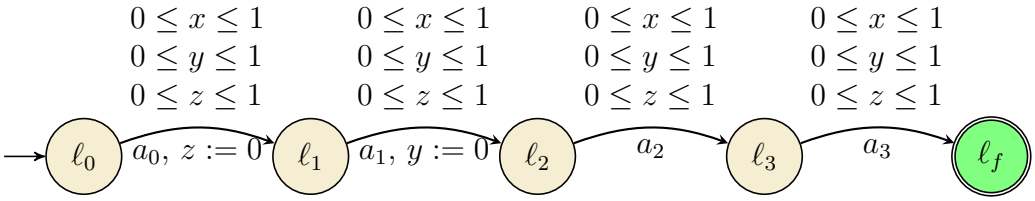
First, let us present the runtime for the examples of the previous section and for the additional timed automata presented in Figure 6.20a, 6.20b and 6.20c. Let us compare



(a) A three transitions timed automaton.



(b) A three clock timed automaton.



(c) Another three clock timed automaton.

Figure 6.20: Three timed automata studied in the symbolic implementation.

the runtime of the symbolic implementation with the one of the numeric implementation, for two interval sampling steps: $\frac{1}{2}$ and $\frac{1}{15}$ in Tables 6.1 and 6.2. We can observe that the runtime grows with respect to the number of cells. The symbolic runtime is higher than the numeric ones, except for our examples with three clocks and four transitions, for a step sampling equal to $\frac{1}{15}$.

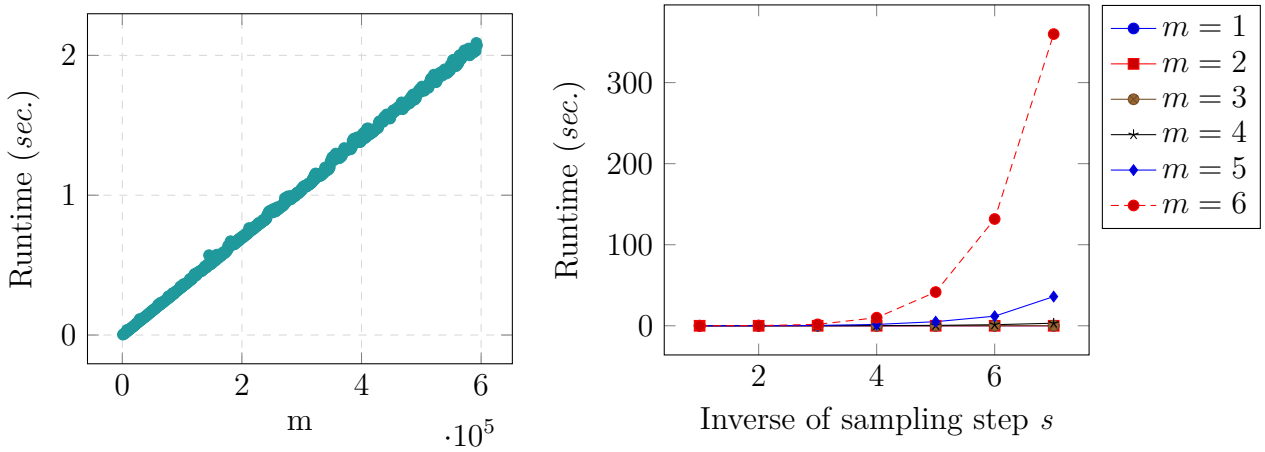
As for the numerical implementation, we tested our implementation on an example which allows us to vary the number of transitions. As a reminder, this is an m transition automaton, linear, with two clocks, whose guards are identical: no reset and the constraints are always $0 \leq x \leq 1 \wedge 0 \leq y \leq 1$. The result is presented in Figure 6.21a. This example has the particularity to be always represented with two cells, as in Figure 6.17. When considering m transitions, the result is a slightly modified result where the tiling of polyhedra stays the same and the associated function becomes $(x, y) \mapsto \frac{1-x}{m}$ and $(x, y) \mapsto \frac{1-y}{m}$. In the obtained results in Figure 6.21a, we represented the runtime in seconds on the y axis and the number of transition on the x axis. The number of transitions

Timed automaton	Runtime for ℓ_0	Runtime for ℓ_1	Runtime for ℓ_2	Runtime for ℓ_3
Figure 2.8	0.82 (2 cells)	0.059 (2 cells)	-	-
Figure 2.10	0.071 (6 cells)	0.062 (3 cells)	-	-
Figure 5.3	0.73 (24 cells)	0.034 (3 cells)	-	-
Figure 6.20a	38.16 (582 cells)	1.01 (25 cells)	0.09 (3 cells)	-
Figure 6.20b	19.95 (234 cells)	0.41 (12 cells)	0.56 (6 cells)	0.14 (6 cells)
Figure 6.20c	143.48 (1825 cells)	0.39 (12 cells)	0.59 (6 cells)	0.14 (6 cells)

Table 6.1: Runtime results (in *sec.*) of our symbolic implementation when computing the permissiveness function on ℓ_0, ℓ_1 and ℓ_2 and ℓ_3.

Timed automaton	Runtime (numeric) for $s = \frac{1}{2}$	Runtime (numeric) for $s = \frac{1}{15}$
Figure 2.8	0.0016	0.15
Figure 2.10	0.021	0.22
Figure 5.3	0.021	0.022
Figure 6.20a	0,0078	7,98
Figure 6.20b	0.026	330.87
Figure 6.20c	0.025	329.69

Table 6.2: Runtime results (in *sec.*) of our numeric implementation for two different interval sampling steps for configuration $(\ell_0, (0, 0))$.



(a) Runtime result for symbolic approach for the location ℓ_0.

(b) Runtime result for numeric approach,for the configuration $(\ell_0, (0, 0))$.

Figure 6.21: Comparison of numeric and symbolic approaches for a two-clocks-automaton with m transitions, with the same guards $0 \leq x \leq 1 \wedge 0 \leq y \leq 1$ on each transitions (and no reset).

should be multiplied by 10^5 as indicated in bottom left. The scatter plot almost forms a straight line. Its slope has an order of magnitude of 10^{-6}, which is an encouraging result compared to the worst case complexity we computed in Section 5.1 and to the numeric implementation runtime result, that we recall in Figure 6.21b. Nevertheless, this is a very specific case where the number of cells and the number of inequalities defining the polyhedra remain constant as the number of transitions increases. This result encourages us to **control** the number of cells when computing the permissiveness function with a symbolic algorithm.

Conclusion and future work

We have presented the symbolic implementation, in the framework of linear timed automata, of the algorithm proposed in Chapter 5. The runtime results are high, compared to the numeric implementation, because of the over-tiling. We saw in an example that when the number of cells is controlled, the runtime of our symbolic implementation are very encouraging, despite the theoretical upper bound of our complexity computed in Section 5.1.

Furthermore, this implementation highlights the direction to take in order to reduce the complexity of our algorithms: reduce the number of polyhedra used to represent a piecewise affine function. Therefore, in the next chapter, Chapter 7, we will discuss a second approach to compute an approximate symbolic version of the permissiveness function, which controls both the number of cells and the precision obtained. However, the complexity will then be on the number of constraints used to represent polyhedra.

Regarding the feasible improvements on our implementation, the results that we provided confirm the presence of an over-tiling of the polyhedron partition. We propose several directions for future work to address this issue. First, we should extend this implementation to more general timed automata, as acyclic timed automata. We could also extend to timed automata with cycles, but we would have no guarantee that our algorithm terminates. Nevertheless it could be a direction to explore some examples to find heuristic about this class of timed automata we could not study formally in Chapter 5. Second, we could improve the efficiency of our implementation. We propose several ways to do it. Among these tracks, we propose an algorithm that allows to check the equality between two piecewise affine functions and an algorithm that allows to compute the union of polyhedra if we know that the result is a polyhedron. We also propose a heuristic, that is not guaranteed to always reduce the number of cells, but which computes the convex

hull of polyhedra for a piecewise affine functions, considering only the polyhedra associated to the same affine functions. The goal of this heuristic is to reduce the number of cells by merging the one that could be merged and still be a polyhedron. To ensure we did compute exactly the union of polyhedra, this algorithm checks for equality with the original piecewise affine function.

Chapter 7

Levelled and binary permissiveness: computing the permissiveness with threshold(s)

The algorithm we presented in Chapter 5 gives an **exact** computation of the permissiveness function. It enables us to compute the strategy of the player for **any** location and for **any** valuation. However, we proved that this algorithm can be executed in at most in non-elementary time for acyclic timed automata. This might be an very pessimistic upper-bound but we implemented this algorithm for linear timed automata and the experimental results presented in Chapter 6 show a fairly long runtime. The complexity of our algorithm depends in the number of cells of the computed permissiveness functions. Therefore, our goal to improve our approach is to reduce the number of cells that have to be considered.

Let us first remark that our implementation, presented in Chapter 6, suffers from over-tiling, which significantly increases the runtime. However, our methods for reducing this over-tiling are costly, as they require the computation of the convex hull of the union of several polyhedra. Our best way to reduce the complexity of our algorithm is to reduce this number of cells.

To reduce the number of cells, our first attempt was to consider a numerical approach, that computes an approximate value of the permissiveness function of a fixed configuration. We presented its algorithm and implementation in Section 6.1. The issue of this algorithm is the lack of proof of its precision. We did not manage to compute a bound of the error we make by approximating the permissiveness function.

Therefore, in this chapter, we propose an approximate approach that controls, by definition, the precision of our permissiveness computation. It will also have the advantage, unlike the numerical approach, of computing the approximate permissiveness value for any configuration, as did the symbolic approach in Chapter 5.

The idea of our approach in this chapter is to compute permissiveness by thresholds: we compute a function that indicates, given a finite ordered number of thresholds, whether the permissiveness function lies between two thresholds, see Figure 7.2 for an example. We call this function the *levelled permissiveness function*. Our method to compute whether permissiveness is between two thresholds is to compute whether it is greater than a threshold. We call the function that computes whether a permissiveness function is greater than a threshold the *binary permissiveness function*. In this chapter, we present an algorithm to compute the binary and levelled permissiveness functions for **linear timed automata**.

This chapter is organised as follows:

- First, we present the binary permissiveness and we propose an algorithm to compute this function in Section 7.1.
- Secondly, we present the levelled permissiveness and formally show how to compute it by computing the binary permissiveness in Section 7.2.

7.1 Binary permissiveness

In this section, we present the binary permissiveness function, which returns whether the permissiveness is greater than a fixed threshold. We first give a formal definition and examples in Subsection 7.1.1, then provide a sequence that computes iteratively this function in Subsection 7.1.2 and a Fourier-Motzkin-based algorithm that computes the binary permissiveness function in Subsection 7.1.3.

7.1.1 Definitions and examples

Let us start by defining the binary permissiveness function in Definition 7.1. Given a timed automaton $\mathcal{A}$, this function returns 1 if the permissiveness of the configuration (ℓ, v) is greater than a fixed threshold and 0 otherwise.

Definition 7.1: Binary permissiveness function

Let us consider a timed automaton $\mathcal{A}$ and a threshold p and $\succ \in \{>, \geq\}$. The binary permissiveness function is denoted $\mathrm{Bin}_{p,\succ}$ and is defined as follows:

$$\mathrm{Bin}_{p,\succ} : (\ell, v) \mapsto \begin{cases} 1 & \text{if } \mathrm{Perm}(\ell, v) \succ p \\ 0 & \text{otherwise} \end{cases}$$

Binary permissiveness function can only take two values, 0 or 1. An advantage is the ability to express this function in terms of sets instead of a function. We can reduce its representation to the set of valuations where the function is equal to 1, that we denote $\mathcal{S}(p, \ell)$ and define as follows:

$$\mathcal{S}(p, \ell) := \mathrm{Bin}^{-1}_{p,\succ}(1).$$

Indeed, the set $\mathrm{Bin}^{-1}_{p,\succ}(0)$ can be deduced using only $\mathrm{Bin}^{-1}_{p,\succ}(1)$. This representation will be preferred in order to give a geometric point of view.

Remark 7.1.1 *In the general case, we suppose that $\succ$ is $\geq$, as our algorithm and proof will not change using $>$. Unless otherwise stated, throughout the rest of the chapter we will always suppose that $\succ$ is $\geq$.* ***The binary permissiveness*** $\mathrm{Bin}_{p,\geq}$ ***will be denoted, by abuse,*** Bin_p***.***

The goal of this section is to provide an algorithm to solve the following problem defined in Definition 7.2.

Definition 7.2: Binary-permissiveness problem

Given a timed automaton $\mathcal{A}$, a threshold $p \geq 0$ and an initial configuration (ℓ_0, v_0), the binary-permissiveness problem asks to compute $\mathrm{Bin}_p(\ell_0, v_0)$.

Let us represent in Figure 7.1 an example of binary permissiveness functions for several thresholds. We consider the timed automaton of Figure 2.10, studied in Chapter 2, that we show again below for the sake of convenience.

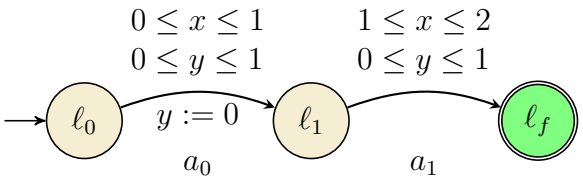
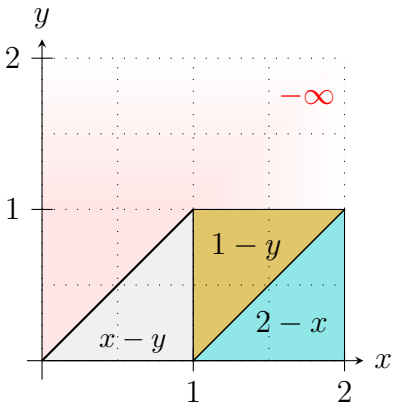


Figure 2.10: An example of timed automaton.

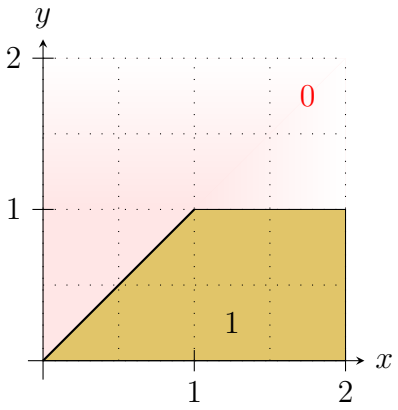
Let us compare the permissiveness function and the binary permissiveness represented in Figure 7.1a. To represent the permissiveness function, we had to use three polyhedra, whereas only one is sufficient for the binary permissiveness function.

7.1.2 Sequence of binary suboptimal permissive functions

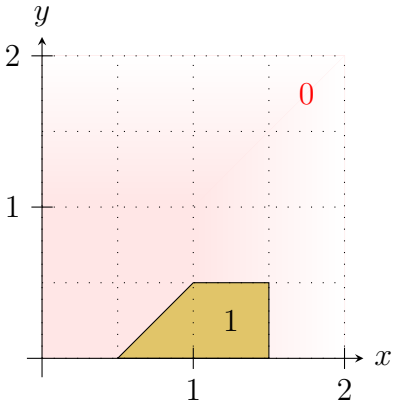
As with the permissiveness function, the binary permissiveness function is not directly iterative. This is why we define a sequence of functions, called a *sequence of binary func-*



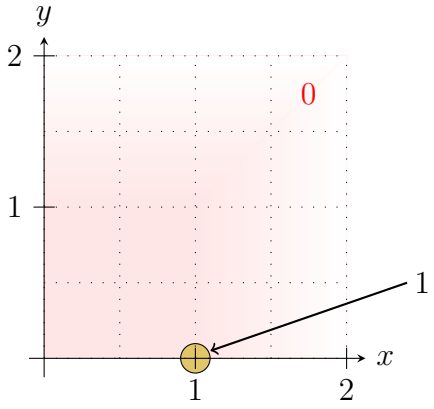
(a) The permissiveness function on ℓ_1 of the timed automaton of Figure 2.10.



(b) Representation of $\mathrm{Bin}_0(\ell_1, \cdot)$: $\mathrm{Bin}_0(\ell_1, v_1)$ is 1 if and only if $\mathrm{Perm}(\ell_1, v_1) \geq 0$.



(c) Representation of $\mathrm{Bin}_{\frac{1}{2}}(\ell_1, \cdot)$: $\mathrm{Bin}_{1/2}(\ell_1, v_1)$ is 1 if and only if $\mathrm{Perm}(\ell_1, v_1) \geq 1/2$.



(d) Representation of $\mathrm{Bin}_1(\ell_1, \cdot)$: $\mathrm{Bin}_1(\ell_1, v_1)$ is 1 if and only if $\mathrm{Perm}(\ell_1, v_1) \geq 1$.

Figure 7.1: An example of binary permissiveness functions $\mathrm{Bin}_1(\ell_1, \cdot)$ of the timed automaton of Figure 2.10 for different thresholds $0, \frac{1}{2}, 1$.

tions, in definition 7.3, which will approximate this binary permissiveness function and tends to it.

Definition 7.3: Sequence of binary suboptimal permissive functions

Let $\mathcal{A}$ be an arbitrary timed automaton, p be a positive threshold, and ℓ an arbitrary location. The *sequence of p-binary suboptimal permissive functions*, denoted $(\mathrm{Bin}_{i,p}(\ell,\cdot))_{i\geq 0}$, is a sequence of functions that only take values 0 or 1 and is defined as follows:

- For $i=0$,
$$\mathrm{Bin}_{i,p}(\ell,v)=\begin{cases}1 & \text{if } \ell\in Q_f\\ 0 & \text{otherwise}\end{cases}.$$
- For $i>0$, if $\mathsf{p\text{-}moves}(\ell,v)\neq\varnothing$,
$$\mathrm{Bin}_{i,p}(\ell,v)=\sup_{(I,a)\in\mathsf{p\text{-}moves}(\ell,v)}\min\left[\mathbb{1}_{\{|I|\geq p\}}(I),\inf_{\delta\in I}\mathrm{Bin}_{i-1,p}(\mathrm{succ}(\ell,v,\delta,a))\right]$$
- For $i>0$, if $\mathsf{p\text{-}moves}(\ell,v)=\varnothing$, $\mathrm{Bin}_{i,p}(\ell,v)=0$.

As for the binary permissiveness function, we denote $\mathcal{S}_i(p,\ell):=\mathrm{Bin}_{i,p}(\ell,\cdot)^{-1}(1)$.

The Proposition 7.4 gives the link between the sequence of suboptimal permissive functions and the sequence of suboptimal binary permissive functions. It will be a very useful tool to extend some properties already proven in Section 4.2.

Proposition 7.4

Let $\mathcal{A}$ be an acyclic timed automaton, p be a threshold, (ℓ,v) an arbitrary configuration and $i\in\mathbb{N}$ an integer.
$\mathrm{Bin}_{i,p}(\ell,v)=1$ if and only if $\mathcal{P}_i(\ell,v)\geq p$.

Proof of Proposition 7.4. Let us prove this proposition by a simple induction.

If $i=0$, this case is trivial by definition. Let us now consider an integer $i\in\mathbb{N}$ and let us suppose that $\mathrm{Bin}_{i,p}(\ell,v)=1$ if and only if $\mathcal{P}_i(\ell,v)\geq p$.

$\Rightarrow$ Let us suppose that $\mathcal{P}_{i+1}(\ell,v)\geq p$, then there exits a p-move $(I_0,a_0)\in\mathsf{p\text{-}moves}(\ell,v)$ such that $|I_0|\geq p$ and for any delay $\delta\in I_0$:

$$\mathcal{P}_i(\mathrm{succ}(\ell,v,\delta,a_0))\geq p.$$

As a result, by the induction hypothesis, $\mathrm{Bin}_{i,p}\,\mathrm{succ}\,(\ell, v, \delta, a_0) = 1$ and then:

$$\min\left(\mathbb{1}_{\{|I_0|\geq p\}}\,(I_0)\,,\,\inf_{\delta\in I_0}\mathrm{Bin}_{i,p}\,(\mathrm{succ}\,(\ell, v, \delta, a_0))\right) = 1 \leq \mathrm{Bin}_{i+1,p}\,(\ell, v)\,.$$

As $\mathrm{Bin}_{i+1,p}\,(\ell, \cdot)$ can only be equal to 0 or 1:

$$\mathrm{Bin}_{i+1,p}\,(\ell, v) = 1.$$

$\Leftarrow$ Suppose that $\mathrm{Bin}_{i,p}\,(\ell, v) = 1$, then there exists a p-move $(I_0, a_0) \in \mathsf{p\text{-}moves}\,(\ell, v)$ such that $|I_0| \geq p$ and for any delay $\delta \in I_0$, $\inf_{\delta\in I_0}\mathrm{Bin}_{i,p}\,\mathrm{succ}\,(\ell, v, \delta, a_0) = 1$. Then, with the same arguments as previously, $\mathcal{P}_{i+1}\,(\ell, v) \geq p$.

□

Thanks to Proposition 7.4, several properties of $\mathcal{P}_i$, presented in Section 4.2, can be extended or adapted to $\mathrm{Bin}_{i,p}$, by simple inductions:

- ▷ Lemma 4.3: for a fixed configuration (ℓ, v), the sequence $(\mathrm{Bin}_i\,p\,(\ell, v))_{i\geq 0}$ is non-decreasing.
- ▷ Lemma 4.5: the sequence $(\mathrm{Bin}_{i,p}\,(\ell, v))_{i\geq 0}$ is eventually constant from rank d_ℓ for acyclic timed automata.
- ▷ Proposition 4.7 and its Corollaries 4.8 and 4.9:

  $\mathrm{Bin}_{i,p}\,(\ell, v) = 1$ if and only if there exists a permissive strategy with permissiveness larger than (or equal to) p that is winning from (ℓ, v) within i steps.

  The main consequences that can be deduced are:

  1. $\lim_{i\to+\infty}\mathrm{Bin}_{i,p}\,(\ell, v)$, if it exists, has for limits $\mathrm{Bin}_p\,(\ell, v)$.
  2. This limits exists and is reached within d_ℓ steps for acyclic timed automata.

The properties that cannot be adapted or extended are, of course, the continuity properties. Nevertheless, the Proposition 4.15 will help us proving a major property for linear timed automata. Indeed, as the permissiveness function and the sequence of suboptimal permissive functions are continuous and concave over Win_ℓ for any location ℓ, we can easily prove in Proposition 7.5 that $\mathcal{S}_i\,(p, \ell)$ forms a unique polyhedron when $\mathcal{A}$ is a linear timed automaton.

Proposition 7.5

Let $\mathcal{A}$ be a timed automaton, ℓ be an arbitrary location and p be a positive threshold. $\mathcal{S}_i(p,\ell)$ and $\mathcal{S}(p,\ell)$ can be represented as finite union of closed polyhedra of $\mathbb{R}_+^{|\mathcal{C}|}$. In particular, if $\mathcal{A}$ is a linear timed automaton, $\mathcal{S}_i(p,\ell)$ and $\mathcal{S}(p,\ell)$ are closed polyhedra of $\mathbb{R}_+^{|\mathcal{C}|}$.

Proof of Proposition 7.5. By definition, $\mathcal{S}_i(p,\ell) = \mathcal{P}_i(\ell,\cdot)^{-1}([p,+\infty])$. As $\mathcal{P}_i(\ell,\cdot)$ is a piecewise affine function over Win_ℓ, $\mathcal{S}_i(p,\ell)$ is a finite set of polyhedra.

Let us now suppose that $\mathcal{A}$ is linear and let us prove that $\mathcal{S}_i(p,\ell)$ is a convex set.

Let $v_1, v_2 \in \mathcal{S}_i(p,\ell)$ and $\lambda \in [0,1]$. Let us prove that $v_\lambda := \lambda \cdot v_1 + (1-\lambda)\cdot v_2 \in \mathcal{S}_i(p,\ell)$.

As $\mathcal{P}_i(\ell,\cdot)$ is a concave function, thanks to Proposition 4.15, if $\mathcal{P}_i(\ell,v_1) \geq p$ and $\mathcal{P}_i(\ell,v_2) \geq p$, then:

$$\mathcal{P}_i(\ell, \lambda v_1 + (1-\lambda) v_2) \geq p.$$

As a result $v_\lambda \in \mathcal{S}_i(p,\ell)$. Therefore, $\mathcal{S}_i(p,\ell)$ is a convex polyhedron of $\mathbb{R}_+^{|\mathcal{C}|}$.

As this sequence tends in a finite number of steps to the binary permissiveness, this result also holds for $\mathcal{S}(p,\ell)$. □

7.1.3 Binary permissiveness algorithm

Let us now present an algorithm that computes the binary function sequence for linear automata. We will use the **geometric** point of view in order to compute this sequence of function. Indeed it will be sufficient to compute $\mathcal{S}_i(p,\ell)$ for $i = \mathrm{d}_\ell$.

Our algorithm, given a linear timed automaton $\mathcal{A}$ with n clocks, a location ℓ and its successor ℓ', a threshold p and $\mathcal{S}_{i-1}(p,\ell')$ (if $i > 0$), **computes** the set $\mathcal{S}_i(p,\ell)$.

7.1.3.1 Computation for $i = 0$

The computation of $\mathcal{S}_i(p,\ell)$ is trivial for $i = 0$ as it is sufficient to check whether ℓ belongs to the set of target locations.

7.1.3.2 Computation for $i > 0$

For $i > 0$, we will prove the following lemma that simplifies the expression of $\mathcal{S}_i(p,\ell)$:

Lemma 7.6

Let $\mathcal{A}$ be a linear timed automaton with n clocks, $i > 0$, $\ell \notin Q_f$ be a location, ℓ' its successors and p be a threshold. Then:

$$\mathcal{S}_i(p,\ell) = \left\{ v \in \mathbb{R}_+^n \mid \exists \alpha \geq 0 \ s.t. \ \begin{cases} ([\alpha, \alpha+p], a) \text{ is an enabled p-move} \\ v + \alpha\,[\mathcal{C}_r \leftarrow 0], v + \alpha + p\,[\mathcal{C}_r \leftarrow 0] \in \mathcal{S}_{i-1}(p,\ell') \end{cases} \right\}.$$

Proof of Lemma 7.6. Suppose now that $i > 0$. $\mathsf{p\text{-}moves}(\ell, v) \neq \varnothing$ if there exists an enabled move (δ, a) to reach the configuration $(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0])$. If such move exists, then:

$$\mathrm{Bin}_{i,p}(\ell, v) = \sup_{(I,a)\in \mathsf{p\text{-}moves}(\ell,v)} \min\left[\mathbb{1}_{\{|I|\geq p\}}(I), \inf_{\delta\in I} \mathrm{Bin}_{i-1,p}(\mathrm{succ}(\ell, v, \delta, a))\right]$$

Therefore, to compute $\mathcal{S}_i(p,\ell) := \left\{v \in \mathbb{R}_+^n \mid \mathrm{Bin}_{i,p}(\ell, v) = 1\right\}$, we have to compute the set of configurations (ℓ, v) such that there exists a p-move $([\alpha, \beta], a)$ such that:

1. $|\beta - \alpha| \geq p$
2. For any delay $\delta \in [\alpha, \beta]$, $\mathrm{Bin}_{i-1,p}(\ell', v + \delta\,[\mathcal{C}_r \leftarrow 0]) = 1$

The second proposition is equivalent to: for any $\delta \in [\alpha, \beta]$, $\delta \in \mathcal{S}_{i-1}(p, \ell')$. As a result we can simplify the expression of $\mathcal{S}_i(p,\ell)$ as follows:

$$\mathcal{S}_i(p,\ell) = \left\{ v \in \mathbb{R}_+^n \mid \exists 0 \leq \alpha \leq \beta \ s.t. \ \begin{cases} ([\alpha, \beta], a) \text{ is an enabled p-move} \\ \beta - \alpha \geq p \\ \forall \delta \in [\alpha, \beta], v + \delta\,[\mathcal{C}_r \leftarrow 0] \in \mathcal{S}_{i-1}(p,\ell') \end{cases} \right\}$$

As $\mathcal{S}_{i-1}(p, \ell')$ is a polyhedron, it is a convex set so we only have to check the membership of $v + \alpha\,[\mathcal{C}_r \leftarrow 0]$ and $v + \beta\,[\mathcal{C}_r \leftarrow 0]$:

$$\mathcal{S}_i(p,\ell) = \left\{ v \in \mathbb{R}_+^n \mid \exists 0 \leq \alpha \leq \beta \ s.t. \ \begin{cases} ([\alpha, \beta], a) \text{ is an enabled p-move} \\ \beta - \alpha \geq p \\ v + \alpha\,[\mathcal{C}_r \leftarrow 0], v + \beta\,[\mathcal{C}_r \leftarrow 0] \in \mathcal{S}_{i-1}(p,\ell') \end{cases} \right\}$$

Let g be the polyhedral guard of the transition between ℓ and ℓ'. A valuation satisfies g if it belong to its associated polyhedron, which is a convex set. Similarly $\mathcal{S}_{i-1}(p, \ell')$ is a

convex polyhedron. Therefore, the smallest β that satisfies such conditions is $\alpha + p$. We can then simplify $\mathcal{S}_i(p, \ell)$ as follows:

$$\mathcal{S}_i(p,\ell) = \left\{ v \in \mathbb{R}_+^n \mid \exists \alpha \geq 0 \ s.t. \ \begin{cases} ([\alpha, \alpha + p], a) \text{ is an enabled p-move} \\ v + \alpha\,[\mathcal{C}_r \leftarrow 0], v + \alpha + p\,[\mathcal{C}_r \leftarrow 0] \in \mathcal{S}_{i-1}(p, \ell') \end{cases} \right\}.$$

□

Thanks to Lemma 7.6, if $\ell \notin Q_f$ (if so, computing $\mathrm{Bin}_{i,p}\,\ell$ is trivial), it is sufficient to find an enabled p-move $(\alpha, \alpha + p)$ such that $v + \alpha\,[\mathcal{C}_r \leftarrow 0]$ and $v + \alpha + p\,[\mathcal{C}_r \leftarrow 0]$ belong to $\mathcal{S}_{i-1}(p, \ell')$. To compute the valuations that satisfies these conditions, we use the **Fourier-Motzkin algorithm** in order to eliminate the variable α.

Let us first denote some notations:

Guards: Let g be the polyhedral guard between the location ℓ and ℓ' defined by the following inequalities: $\bigwedge_{j=1}^{c_g} \varphi_j^{(g)}(X) \geq 0$.

$\mathcal{S}_{i-1}(p, \ell')$: This set is a polyhedron, let us define its inequalities: $\bigwedge_{k=1}^{c_i} \varphi_k(X) \geq 0$.

With these notations, we can re-write $\mathcal{S}_i(p, \ell)$ as the set of valuations v such that there exists $\alpha \geq 0$ that satisfies the following inequalities, enumerated in 1., 2., 3. and 4.:

1. $v + \alpha \models g$: $\forall 0 \leq j \leq c_g$: $\varphi_j^{(g)}(v + \alpha) \geq 0$
2. $v + \alpha + p \models g$: $\forall 0 \leq j \leq c_g$: $\varphi_j^{(g)}(v + \alpha + p) \geq 0$
3. $v + \alpha\,[\mathcal{C}_r \leftarrow 0] \in \mathcal{S}_{i-1}(p, \ell')$: $\forall 0 \leq k \leq c_i$: $\varphi_k(v + \alpha\,[\mathcal{C}_r \leftarrow 0]) \geq 0$
4. $v + \alpha + p\,[\mathcal{C}_r \leftarrow 0] \in \mathcal{S}_{i-1}(p, \ell')$: $\forall 0 \leq k \leq c_i$: $\varphi_k(v + \alpha + p\,[\mathcal{C}_r \leftarrow 0]) \geq 0$

Let us eliminate α in these inequalities:

$$\varphi_j^{(g)}(v + \alpha) := \sum_{i'=0}^{n} \varphi_{j,i'}^{(g)}(v_{i'} + \alpha)$$

Let us denote $S_j^{(g)} = \sum_{i'=0}^{n} \varphi_{j,i'}^{(g)}$, then:

$$\varphi_j^{(g)}(v+\alpha) = S_j^{(g)} \cdot \alpha + \varphi_j^{(g)}(v)$$
$$\varphi_j^{(g)}(v+\alpha+p) = S_j^{(g)} \cdot \alpha + \varphi_j^{(g)}(v+p)$$

As previously:

$$\varphi_k(v+\alpha[\mathcal{C}_r \leftarrow 0]) := \sum_{i'=0, i' \notin \mathcal{C}_r}^{n} \varphi_{k,i'}(v_{i'}+\alpha)$$

Let us denote $S_k = \sum_{i'=0, i' \notin \mathcal{C}_r}^{n} \varphi_{k,i'}$, then:

$$\varphi_k(v+\alpha[\mathcal{C}_r \leftarrow 0]) = S_k \cdot \alpha + \varphi_k(v[\mathcal{C}_r \leftarrow 0])$$
$$\varphi_k(v+\alpha+p[\mathcal{C}_r \leftarrow 0]) = S_k \cdot \alpha + \varphi_k(v+p[\mathcal{C}_r \leftarrow 0])$$

In order to isolate α in these inequalities, we divide our inequalities by $S_j^{(g)}$ or S_k. Therefore we have to distinguish the cases where $S_j^{(g)}$ and S_k are positive, negative, or equal to zero. Let us denote the following sets:

$$\mathrm{Pos}_g = \left\{j \in [\![1,n]\!] \mid S_j^{(g)} > 0\right\}, \mathrm{Neg}_g = \left\{j \in [\![1,n]\!] \mid S_j^{(g)} < 0\right\}, \mathrm{Zero}_g = \left\{j \in [\![1,n]\!] \mid S_j^{(g)} = 0\right\}$$

and

$$\mathrm{Pos}_i = \{k \in [\![1,n]\!] \mid S_k > 0\}, \mathrm{Neg}_i = \{k \in [\![1,n]\!] \mid S_k < 0\}, \mathrm{Zero}_i = \{k \in [\![1,n]\!] \mid S_k = 0\}$$

Let us denote, for $\gamma \in \{i, g\}$, the size of the previously defined sets as follows:

$$p_\gamma := |\mathrm{Pos}_\gamma|, n_\gamma := \left|\mathrm{Neg}_\gamma\right|, z_\gamma := |\mathrm{Zero}_\gamma|.$$

As a result, we will obtain three forms of inequalities: $A_{k'} \leq \alpha$, $B_{j'} \geq \alpha$ and $C \geq 0$ that characterise the valuations of $\mathcal{S}_i(p, v)$:

- $\forall j \in \mathrm{Pos}_g$, $\dfrac{-\varphi_j^{(g)}(v)}{S_j^{(g)}} \leq \alpha$ and $\dfrac{-\varphi_j^{(g)}(v+p)}{S_j^{(g)}} \leq \alpha$

- $\forall j \in \mathrm{Neg}_g$, $\dfrac{-\varphi_j^{(g)}(v)}{S_j^{(g)}} \geq \alpha$ and $\dfrac{-\varphi_j^{(g)}(v+p)}{S_j^{(g)}} \geq \alpha$
- $\forall j \in \mathrm{Zero}_g$, $\varphi_j^{(g)}(v) \geq 0$ and $\varphi_j^{(g)}(v+p) \geq 0$
- $\forall k \in \mathrm{Pos}_i$, $\dfrac{-\varphi_k(v[\mathcal{C}_r \leftarrow 0])}{S_k} \leq \alpha$ and $\dfrac{-\varphi_k(v+p[\mathcal{C}_r \leftarrow 0])}{S_k} \leq \alpha$
- $\forall k \in \mathrm{Neg}_i$, $\dfrac{-\varphi_k(v[\mathcal{C}_r \leftarrow 0])}{S_k} \geq \alpha$ and $\dfrac{-\varphi_k(v+p[\mathcal{C}_r \leftarrow 0])}{S_k} \geq \alpha$
- $\forall k \in \mathrm{Zero}_i$, $\varphi_k(v[\mathcal{C}_r \leftarrow 0]) \geq 0$ and $\varphi_k(v+p[\mathcal{C}_r \leftarrow 0]) \geq 0$

We can eliminate α by writing for every j', k': $A_{k'} \leq B_{j'}$ and $C \geq 0$. We obtain $2(z_g + z_i) + 4(p_g + p_i)(n_g + n_i)$ inequalities. As a result, we can compute $\mathcal{S}_i(p, \ell)$ with at most $2(z_g + z_i) + 4(p_g + p_i)(n_g + n_i)$ inequalities.

Then, the binary permissiveness function can be computed by the Algorithm 5.

Data: A timed automaton $\mathcal{A}$, a location ℓ_0, a target location ℓ_f and a threshold p
Result: $\mathcal{S}(p, \ell_0)$
Compute $\mathcal{S}_0(p, \ell_f)$;
$\ell \leftarrow$ predecessor of ℓ_f;
for $i \leftarrow 1$ **to** d_{ℓ_0} **do**
    Compute $\mathcal{S}_i(p, \ell)$ with the algorithm presented in subsection 7.1.3.2 ;
    $\ell \leftarrow$ predecessor of ℓ;
end
return $\mathcal{S}_{d_{\ell_0}}(p, \ell)$

Algorithm 5: Computation of $\mathcal{S}(p, \ell_0)$.

As a result, the binary permissiveness can be computed in double-exponential time. We state the exact complexity in Theorem 7.7

Theorem 7.7

Let $\mathcal{A}$ be a linear timed automaton with n clocks, ℓ be a location and p be a threshold. Let us denote c_g the maximal number of inequalities defining any guard of $\mathcal{A}$. The polyhedron $\mathcal{S}_{\mathrm{d}_{\ell_0}}(p, \ell)$ can be computed in time $\mathcal{O}\left((4c_g)^{2^{\mathrm{d}_\ell}}\right)$ time and be represented with at most $\mathcal{O}\left((4c_g)^{2^{\mathrm{d}_\ell}}\right)$ linear inequalities.

7.2 Levelled permissiveness

Now, let us present the levelled permissiveness function. The aim of this function is to state to which interval the permissiveness function of a given configuration belongs among these intervals: $]-\infty, p_0[$, $[p_0, p_1[, \cdots, [p_m, +\infty[$. Therefore, the levelled permissiveness function provides an approximate value of the permissiveness function **but** its precisions is known and controlled by definition. Let us first present formally the levelled permissiveness function in Definition 7.8.

Definition 7.8: Levelled permissiveness function

Let us consider a timed automaton $\mathcal{A}$ and a finite set of $m+1$ positive thresholds $\mathcal{P} = (p_i)_{0 \leq i \leq m}$ where $m \in \mathbb{N}$ and for any $i \in [\![0, m-1]\!]$, $p_i < p_{i+1}$. The levelled permissiveness function of a configuration (ℓ, v) is denoted $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}$ and is defined as follows:

- $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}(\ell, v) = -\infty$ if $\mathrm{Perm}(\ell, v) < p_0$
- For any integer $i \in [\![0, m-1]\!]$, $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}(\ell, v) = p_i$ if $\mathrm{Perm}(\ell, v) \in [p_i, p_{i+1}[$.
- $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}(\ell, v) = p_m$ if $\mathrm{Perm}(\ell, v) \geq p_m$.

Let us give an example of a levelled permissiveness function of the timed automaton presented in Figure 2.10 in Figure 7.2 for the set of thresholds $\left(0, \frac{1}{2}, 1\right)$. This example can be computed easily with the binary permissiveness function presented in Figure 7.1.

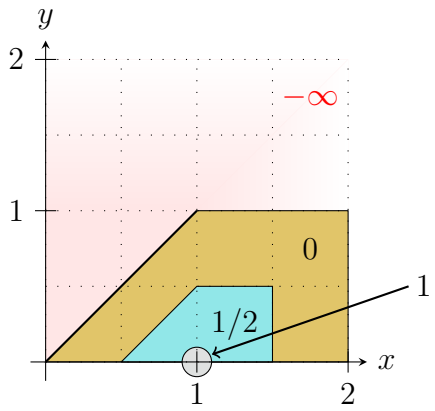


Figure 7.2: The levelled permissiveness of the timed automaton of Figure 2.10, with the thresholds $0, \frac{1}{2}$ and 1.

In order to have a exact control of the precision, and to compute the set of winning

configurations, we usually set p_0 to 0. An obvious intuition is that we can compute the levelled permissiveness by checking if the binary permissiveness of all the thresholds are equal to 0 or 1. We can directly link the binary and permissiveness function.

As with the binary permissiveness function, let us formally state the problem of computing the levelled permissiveness function in Definition 7.9.

Definition 7.9: Maximal-levelled-permissiveness problem

Given a timed automaton $\mathcal{A}$, a set of thresholds $\mathcal{P}$ and an initial configuration (ℓ_0, v_0), the maximal-levelled-permissiveness problem asks to compute $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}(\ell_0, v_0)$.

Thanks to the Algorithm 5 presented in Section 7.1.3, we can conclude on the complexity of the problem stated in Definition 7.9 for **linear timed automata**:

Theorem 7.10

Let $\mathcal{A}$ be a linear timed automaton with n clocks, ℓ be a location and $\mathcal{P} = (p_i)_{0 \leq i \leq m}$ be a set of $m+1$ thresholds. The maximal-levelled-permissiveness problem of Definition 7.9 can be solved in $\mathcal{O}\left((m+1) \cdot (4c_g)^{2^{\mathrm{d}_\ell}}\right)$ time where c_g is the maximal number of inequalities defining any guards of $\mathcal{A}$

Proof of Theorem 7.10. By definition, the three following statements hold:

1. $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}(\ell, v) = -\infty$ if and only if $\mathrm{Bin}_{p_0}(\ell, v) = 0$.
2. For $i \in [\![0, m-1]\!]$, $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}(\ell, v) = p_i$ if and only if $\mathrm{Bin}_{p_i}(\ell, v) = 1$ and $\mathrm{Bin}_{p_{i+1}}(\ell, v) = 0$.
3. $\mathrm{Perm}_{\mathrm{lvl},\mathcal{P}}(\ell, v) = p_m$ if and only if $\mathrm{Bin}_{p_m}(\ell, v) = 1$.

That gives us an immediate algorithm to compute the levelled permissiveness function thanks to the Algorithm 5 that computes the binary permissiveness functions. Indeed, for every threshold p, in a increasing order, we compute $\mathcal{S}(p, \ell)$ in order to compute the levelled permissiveness function. □

Conclusion

In this chapter, we presented a new approach where we compute an approximate value of the permissiveness function with a more efficient algorithm than the symbolic one proposed in Chapter 5. We presented two algorithms, both symbolic (*i.e.* they compute the

function for any arbitrary valuation). The first one computes the **binary permissiveness function**, that values 1 with the permissiveness function is greater than a threshold. The algorithm we proposed can be executed in double exponential time (in $\mathcal{O}\left((4c_g)^{2^{\mathrm{d}_\ell}}\right)$ time). The second one computes the **levelled permissiveness function**: this function is a step function with $m+1$ thresholds and it computes between which couple of thresholds the permissiveness function is. The main advantage of this approach is its control of the precision and the complexity of its algorithm. Indeed, when stating that the permissiveness is between two threshold p_i and p_{i+1}, the approximation we make is at most $p_{i+1} - p_i$. In addition, our algorithm to compute the levelled function is **linear** with respect to the number of thresholds so adding precision is not the most costly part of the algorithm. The algorithm to compute the levelled function can be executed in double exponential time too (in $\mathcal{O}\left((m+1)\cdot(4c_g)^{2^{\mathrm{d}_\ell}}\right)$ time).

Nevertheless, we could only propose an algorithm for **linear** timed automata. Interesting future works would be to extend our approach to acyclic and more general timed automata, and to implement our work in Python with **pplpy**.

Conclusion

In this thesis, we studied the robustness of reachability of timed automata under perturbations of delays. The timed automata we studied take two forms: one with classical guards and one with polyhedral guards, a more general case. The aim of this thesis is to compute the permissiveness of a timed automaton for arbitrary or fixed configuration. The permissiveness computes the maximal imprecision on delays that can be admitted while reaching a goal location of the timed automaton. We define our semantics with a turn-based game with a player that aims to maximise the enabled imprecision and an opponent that aims to minimise it.

We had several approaches in this thesis, that we will detail further in the following paragraphs. We first symbolically compute the exact permissiveness of any location and valuation of a timed automaton. Faced with fairly high time complexity results, we looked at ways to simplify our problem and the algorithms used. We proposed two approaches. The first one was to compute an approximate value of the permissiveness function, for a fixed configuration. The second one was to simplify the computed function, in order to minimise the size of the parameters which make our complexity grow. This conclusion chapter is organised as follows: we first summarise the contributions we have made and then expose the current and future works that can improve our current work.

Contributions

A first exact algorithm: permissiveness algorithm

The first algorithm we propose is a backward symbolic algorithm that computes the permissiveness function. The principle is to compute optimal strategies for the player and the opponent. The algorithm we proposed covers the cases of linear and acyclic timed automata and games. Nevertheless, the worst-case complexity we provided was non-elementary time for acyclic timed automata. The complexity depends on the maximal number of inequalities of guards and grows with the number of cells used to represent our permissiveness as piecewise affine functions. In order to give a proof-of-concept and

to study its runtime on several examples, we implemented this algorithm in Python for linear timed automata. The results we obtain were that our implementation suffers from over-tilling and the runtime can grow exponentially when dealing with more than two clocks.

To tackle these issues, we proposed two other approaches.

A forward approximate approach: numerical algorithm

Our second approach is a numeric approach, that numerically computes an approximate value of the permissiveness by sampling the intervals and delays. Our algorithm computes an approximate value of the permissiveness function of a timed automaton $\mathcal{A}$ in at most $\mathcal{O}\left(\left(B \cdot \frac{\mathcal{M}(\mathcal{A})}{s}\right)^{\mathrm{d}_\ell}\right)$ for acyclic timed automata and $\mathcal{O}\left((2 \cdot B)^{\mathrm{d}_\ell}\right)$ for linear timed automata, where $B := \max\left[\left(\frac{\mathcal{M}(\mathcal{A})}{s}\right)^2, n\right]$ and n is the number of clocks, $\mathcal{M}(\mathcal{A})$ is the maximal constraint of the timed automaton, s is the minimum between the sampling step of the delays and the sampling step of the intervals and d_ℓ is the maximal distance to ℓ.

This approach was implemented in Python and provided quite accurate results, where the error seems to decrease as the sampling step decreases. Nevertheless, its stability is not proven and we did not manage to bound the error made by computing the approximate value of the permissiveness.

Despite the error made by the algorithm, the strength of this forward numerical approach is that no *a priori* knowledge of the strategy of the player and the opponent is needed to run this algorithm. Therefore it can be an interesting tool to study robustness for other models of timed automata.

A symbolic approximate approach: levelled algorithm

Our third approach is a symbolic one. Our goal was to reduce the number of cells used to represent our functions, as the complexity of the symbolic computation depends on the number of cells that represent the permissiveness functions.

We presented two functions that give an approximate value of the permissiveness function. The first one, the binary permissiveness function, checks if the permissiveness function is greater than a fixed threshold. The levelled permissiveness is a step function that partitions $\mathbb{R}_+$ into a partition of $m+1$ left-closed right-open intervals $([s_i, s_{i+1}[)_{0 \leq i \leq m}$

with $s_{m+1} = +\infty$, and takes the value s_i when the permissiveness function takes values in the interval $[s_i, s_{i+1}[$. The main advantage of levelled permissiveness is to provide an approximate symbolic value of the permissiveness with a controlled precision.

We give an algorithm to compute the binary and the levelled permissiveness functions for linear timed automata and show that it can be computed in time respectively at most $\mathcal{O}\left((4c_g)^{2^{\mathrm{d}_\ell}}\right)$ and $\mathcal{O}\left((m+1)\cdot(4c_g)^{2^{\mathrm{d}_\ell}}\right)$ where c_g is the maximal number of inequalities used to describe any guard of the timed automaton and d_ℓ is the maximal distance of ℓ.

The complexity of these problems depends on many parameters. The main advantage of this algorithm is that, for linear timed automata, the binary permissiveness function can be represented by a unique polyhedron and that the computation of the levelled permissiveness function is linear with respect to the number of thresholds used. Nevertheless, theses algorithms currently only cover the case of linear timed automata. Indeed, the ability to represent the binary permissiveness with only one polyhedron does not holds in general for acyclic timed automata and our algorithm relies on this property.

Conclusion and comparison

These three approaches cover different types of timed automata. The implementations provided only cover linear timed automata. They are summed up in the Table 7.1. The second column specifies if our algorithm computes the permissiveness for a fixed configuration ('Numeric') or for all possible configurations ('Symbolic'). The third column specifies if the result is an exact result or an approximation. The fourth and sixth columns specify respectively if the algorithm covers the case of linear or acyclic timed automaton, and the fifth and seventh columns indicate if these cases were implemented.

Algorithm	Type	Exact	Linear	Impl.	Acyclic	Impl.
Permissiveness algorithm	Symbolic	Yes	Yes	Yes	Yes	No
Numerical algorithm	Numeric	No	Yes	Yes	Yes	Yes
Levelled algorithm	Symbolic	No	Yes	No	No	No

Table 7.1: Comparison of our contributions.

Ongoing and future work

Our current work can be extended in several ways. We will explain in this section what are our ongoing and future works for each contribution.

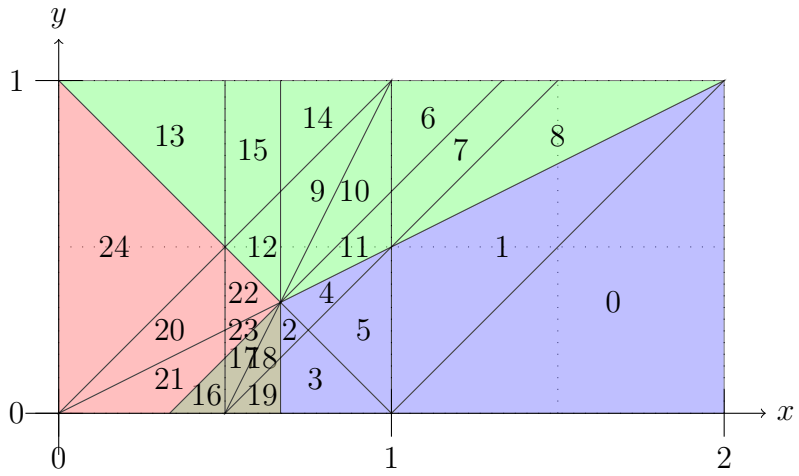
Permissiveness algorithm

Regarding our algorithm presented in Chapter 5, it covers the cases of acyclic timed automata and games, but our implementation only covers the case of linear timed automata. Our first ongoing work is to extend its implementation to acyclic timed automata. To do that, we need to implement a way to check the equality between two piecewise affine functions, independently of the tiling of polyhedra of each piecewise affine functions. Our second ongoing work is to optimise the representation of piecewise affine functions by merging the cells that are associated with the same affine functions. Indeed, our algorithm provides a representation with an over-tiling of the cells of the permissiveness function. To tackle this issue, our goal is to gather all the cells associated with the same affine function and to state whether the union of their polyhedra is a polyhedron. This is a non-trivial question as the union of several polyhedra is not convex in general. We plan to compute, for each group of cells, the convex hull of these polyhedra and associate this set to the affine function. Then, we want to compare the obtained piecewise affine function to the original computed permissiveness function, to check whether they are equal. If they do, the number of cells can be reduced. These steps are illustrated in Figure 7.3. If not, we keep the over-tiling representation. For instance in Figure 7.4, the computation of the convex hull for the affine function $(x, y) \mapsto \frac{-x+1}{2}$ is not the union of the cell $0, 1, 2, 3, 4, 5$ and 13.

An important future work is also to refine the complexity of our algorithm. We proved that it runs in non-elementary time but we may have not exploited all the properties of our constraints for instance. These following possibilities should be considered for future works.

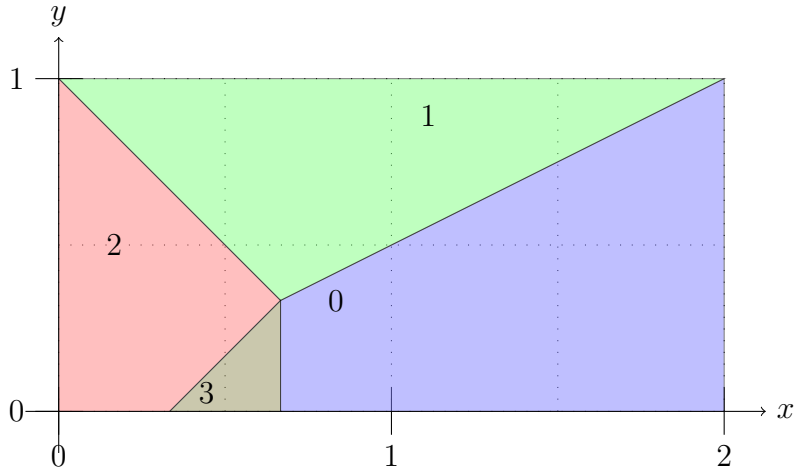
- ▷ Look for an example where the computation time is non-elementary.
- ▷ Improve our algorithm when computing the maximum of piecewise affine functions, when iterating of the couple of cells or refining the tiling of polyhedra.

Other future works for this algorithm would be to tackle the case of cycles. We were not able to prove that the sequence of suboptimal functions converges in a finite number of step to the permissiveness function when the timed automaton contains cycles. Our goal would be to find a counter-example timed automaton where this sequence does not converge and to find conditions where it converges in a finite number of steps. To study these conditions, we want to run our implementation by tackling cycles as in the numerical implementation,



Permissiveness	Associated cells
$(-x+2)/2$	$0, \cdots, 5$
$(-y+1)/1$	$6, \cdots, 15$
$(x-y+1)/2$	$20, \cdots, 24$
$(2)/3$	$16, \cdots, 19$

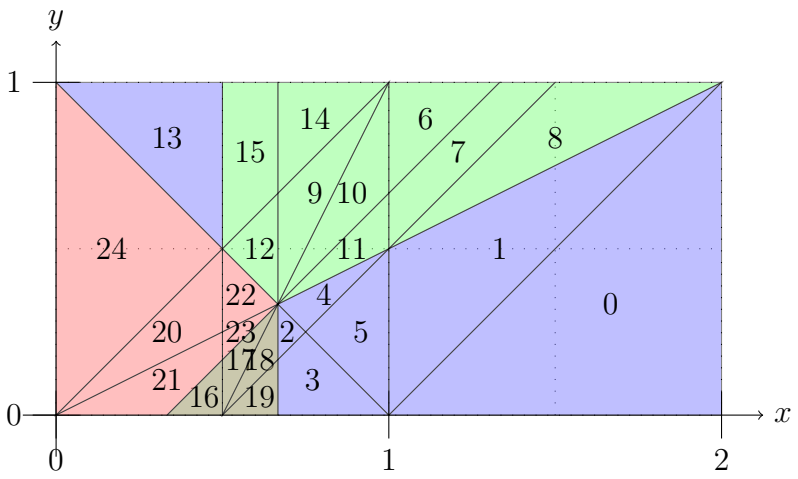
(a) Permissiveness function computed by our algorithm in Section 6.2.4.1.



Permissiveness	Associated cells
$(-x+2)/2$	0
$(-y+1)/1$	1
$(x-y+1)/2$	2
$(2)/3$	3

(b) Obtained piecewise affine function if computing the convex hull for each affine functions.

Figure 7.3: Convex hull computation results.



Permissiveness	Associated cells
$(-x+2)/2$	$0, \cdots, 5$ and 13
$(-y+1)/1$	$6, \cdots, 12, 14$ and 15
$(x-y+1)/2$	$20, \cdots, 24$
$(2)/3$	$16, \cdots, 19$

Figure 7.4: Example where the computation of convex hull will not work.

to observe the behaviour of the sequence of suboptimal permissive functions when we increase the number of allowed cycles.

Numerical algorithm

Regarding our numerical approach, our major future work is to bound the obtained error when computing our permissiveness function, since it is an approximation. Our experimental result on the precision were encouraging and for each example (for both linear and acyclic timed automata) the value of the error curve was cancelled periodically. In all our examples, the error curves have the same shape, with decreasing oscillations. We would study the shape of the oscillations in order to find how to bound the value of the obtained error when computing our permissiveness function.

Another future work would be to extend this implementation to more general models. For instance the guards used were classical guards. Extending it to polyhedral guards would enable us to provide the same JSON formats to run all our implements on the same timed automaton, for the sake of uniformity.

Levelled algorithm

Finally, regarding the binary and levelled permissiveness, our main future work are the following ones. First, we would like to extend our algorithm to acyclic timed automata, and possibly timed automata containing cycles. Secondly, we aim to implement our algorithm with the **pplpy** library. Implementing our current binary and levelled algorithms will be easy for linear timed automata, as these algorithms use the same procedures as the implementation of the permissiveness algorithm, presented in Section 6.2. The main issue of these future works is to extend it to acyclic timed automata. Indeed, a very useful property does not hold in the general case and we were unable to tackle this issue yet.

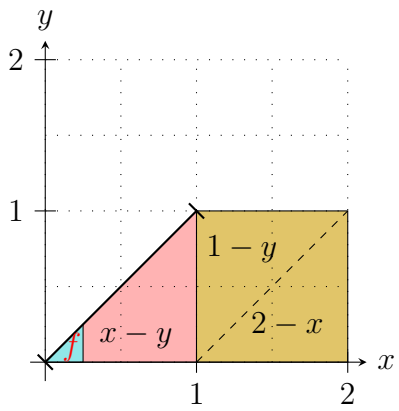
Other approaches

Our future work also consists in proposing new approaches to compute the permissiveness of timed automata.

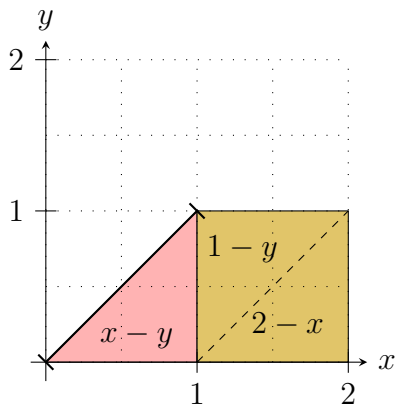
First, we could study other ways to approximate and reduce the number of cells that represent the permissiveness function. An interesting approach would be to avoid computing cells when they could be neglected: we aim to quantify the size of each cells, fix a step ε and neglect the cells whose size is smaller than ε (by merging them with another cell). Indeed, given a location ℓ, as the permissiveness function is 2-Lipschitz, if the distance between two valuations v and v' is smaller than ε, the difference between the permissiveness of (ℓ, v) and (ℓ, v') is smaller than 2ε. This approach would enable us to bound the number of used cells and maybe control the approximation made when computing the permissiveness. A main issue would be to decide how to quantify the size of a cell, how to neglect a cell, and how to bound the approximation made at each step.

We illustrate in Figure 7.5 how we would neglect a cell in a simple example. Let us consider the function in Figure 7.5a is 2-Lipschitz and suppose that the size of the cell associated to f is smaller than ε, in our example merging the cell associated to $x - y$ and to f gives us an approximation of the piecewise affine function where the error made is smaller than 2ε. Nevertheless the general case might be complicated in order to keep polyhedra sets to represent cells.

Indeed, if we bound the error made by computing the 'ε-approximation' of a piecewise-affine function, an important issue is to compute the error made by considering an approximation of the permissiveness function for the possible successors of a configuration when computing the strategies of the player and of the opponent.



(a) An example of a piecewise-affine function.



(b) Supressing the cell associated to f.

Figure 7.5: An intuition of the 'ε-approximation'.

Secondly, we could study different ways to compute the permissiveness functions by considering the functions presented in Section 2.2.3: associate a weight to each transition for instance could give more importance to specific transitions.

Bibliography

[ABG+08] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund and K. Narayan Kumar. Distributed timed automata with independently evolving clocks, *in* Franck van Breugel and Marsha Chechik, editors, *Concurrency Theory, 19th International Conference, (CONCUR 2008) Toronto, Canada.* Volume 5201 of *Lecture Notes in Computer Science*, pages 82–97, Springer, August 2008, DOI: 10.1007/978-3-540-85361-9_10, URL: https://doi.org/10.1007/978-3-540-85361-9%5C_10.

[ABG+14] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund and K. Narayan Kumar. Distributed timed automata with independently evolving clocks, *Fundamenta Informaticae*, 130*(4)*:377–407, 2014, DOI: 10.3233/FI-2014-996, URL: https://doi.org/10.3233/FI-2014-996.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata, *Theoretical Computer Science*, 126*(2)*:183–235, April 1994, DOI: 10.1016/0304-3975(94)90010-8, URL: https://doi.org/10.1016/0304-3975(94)90010-8.

[ADS86] Bowen Alpern, Alan J. Demers and Fred B. Schneider. Safety without stuttering, *Information Processing Letters*, 23*(4)*:177–180, November 1986, DOI: 10.1016/0020-0190(86)90132-8, URL: https://doi.org/10.1016/0020-0190(86)90132-8.

[AFK+12] Étienne André, Laurent Fribourg, Ulrich Kühne and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems, *in* Dimitra Giannakopoulou and Dominique Méry, editors, *Formal Methods - 18th International Symposium, (FM 2012), Paris, France.* Volume 7436 of *Lecture Notes in Computer Science*, pages 33–36, Springer, August 2012, DOI: 10.1007/978-3-642-32759-9_6, URL: https://doi.org/10.1007/978-3-642-32759-9%5C_6.

[AKY07] Parosh Aziz Abdulla, Pavel Krcál and Wang Yi. Sampled universality of timed automata, *in* Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International Conference, (FOSSACS*

2007), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2007), Braga, Portugal. Volume 4423 of *Lecture Notes in Computer Science*, pages 2–16, Springer, March 2007, DOI: 10.1007/978-3-540-71389-0_2, URL: https://doi.org/10.1007/978-3-540-71389-0%5C_2.

[AKY10] Parosh Aziz Abdulla, Pavel Krcál and Wang Yi. Sampled semantics of timed automata, *Logical Methods in Computer Science*, 6*(3)*, September 2010, URL: http://arxiv.org/abs/1007.2783.

[And09] Étienne André. IMITATOR: A tool for synthesizing constraints on timing bounds of timed automata, *in* Martin Leucker and Carroll Morgan, editors, *Theoretical Aspects of Computing - 6th International Colloquium (ICTAC 2009), Kuala Lumpur, Malaysia.* Volume 5684 of *Lecture Notes in Computer Science*, pages 336–342, Springer, August 2009, DOI: 10.1007/978-3-642-03466-4_22, URL: https://doi.org/10.1007/978-3-642-03466-4%5C_22.

[And10] Étienne André. IMITATOR II: A tool for solving the good parameters problem in timed automata, *in* Yu-Fang Chen and Ahmed Rezine, editors, *12th International Workshop on Verification of Infinite-State Systems (INFINITY 2010), Singapore, Singapore.* Volume 39 of *EPTCS*, pages 91–99, September 2010, DOI: 10.4204/EPTCS.39.7, URL: https://doi.org/10.4204/EPTCS.39.7.

[And21] Étienne André. IMITATOR 3: synthesis of timing parameters beyond decidability, *in* Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference (CAV 2021), Virtual Event, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 552–565, Springer, July 2021, DOI: 10.1007/978-3-030-81685-8_26, URL: https://doi.org/10.1007/978-3-030-81685-8%5C_26.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness, *Information Processing Letters*, 21*(4)*:181–185, October 1985, DOI: 10.1016/0020-0190(85)90056-0, URL: https://doi.org/10.1016/0020-0190(85)90056-0.

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness, *Distributed Computing*, 2*(3)*:117–126, September 1987, DOI: 10.1007/BF01782772, URL: https://doi.org/10.1007/BF01782772.

[ATM05] Rajeev Alur, Salvatore La Torre and P. Madhusudan. Perturbed timed automata, *in* Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop (HSCC 2005), Zurich, Switzerland.* Volume 3414 of *Lecture Notes in Computer Science*, pages 70–85, Springer, March 2005, DOI: 10.1007/978-3-540-31954-2_5, URL: https://doi.org/10.1007/978-3-540-31954-2%5C_5.

[ATP01] Rajeev Alur, Salvatore La Torre and George J. Pappas. Optimal paths in weighted timed automata, *in* Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control, 4th International Workshop (HSCC 2001), Rome, Italy.* Volume 2034 of *Lecture Notes in Computer Science*, pages 49–62, Springer, March 2001, DOI: 10.1007/3-540-45351-2_8, URL: https://doi.org/10.1007/3-540-45351-2%5C_8.

[BDL+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi and Martijn Hendriks. UPPAAL 4.0, *in Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), Riverside, California, USA*, pages 125–126, IEEE Computer Society, September 2006, DOI: 10.1109/QEST.2006.59, URL: https://doi.org/10.1109/QEST.2006.59.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis and Sergio Yovine. Kronos: A model-checking tool for real-time systems, *in* Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, (CAV '98), Vancouver, BC, Canada.* Volume 1427 of *Lecture Notes in Computer Science*, pages 546–550, Springer, June 1998, DOI: 10.1007/BFb0028779, URL: https://doi.org/10.1007/BFb0028779.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata, *in* Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control, 4th International Workshop, (HSCC 2001), Rome, Italy.* Volume 2034 of *Lecture Notes in Computer Science*, pages 147–161, Springer, March 2001, DOI: 10.1007/3-540-45351-2_15, URL: https://doi.org/10.1007/3-540-45351-2%5C_15.

[BFM15] Patricia Bouyer, Erwin Fang and Nicolas Markey. Permissive strategies in timed automata and games, *in* Gudmund Grov and Andrew Ireland, editors, *Proceedings of the 15th International Workshop on Automated Verification of Critical Systems (AVOCS'15), Edimburgh, UK.* Volume 72 of *Electronic Communications of the EASST*, European Association of Software Science and Technology, September 2015, DOI: 10.14279/tuj.eceasst.72.1015.

[BFT01] Alberto Bemporad, Komei Fukuda and Fabio Danilo Torrisi. Convexity recognition of the union of polyhedra, *Computational Geometry*, 18*(3)*:141–154, April 2001, DOI: 10.1016/S0925-7721(01)00004-9, URL: https://doi.org/10.1016/S0925-7721(01)00004-9.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*, MIT Press, 2008, ISBN: 978-0-262-02649-9.

[BLM+11] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, Ocan Sankur and Claus R. Thrane. Timed automata can always be made implementable, *in* Joost-Pieter Katoen and Barbara König, editors, *Concurrency Theory - 22nd International Conference (CONCUR 2011), Aachen, Germany.* Volume 6901 of *Lecture Notes in Computer Science*, pages 76–91, Springer, September 2011, DOI: 10.1007/978-3-642-23217-6_6, URL: https://doi.org/10.1007/978-3-642-23217-6%5C_6.

[BMR+19] Damien Busatto-Gaston, Benjamin Monmege, Pierre-Alain Reynier and Ocan Sankur. Robust controller synthesis in timed büchi automata: a symbolic approach, *in* Işil Dillig and Serdar Taşiran, editors, *31st International Conference on Computer Aided Verification (CAV'19), New York City, NY, USA.* Volume 11561 of *Lecture Notes in Computer Science*, pages 572–590, Springer-Verlag, July 2019, DOI: 10.1007/978-3-030-25540-4_33.

[BMR06] Patricia Bouyer, Nicolas Markey and Pierre-Alain Reynier. Robust model-checking of linear-time properties in timed automata, *in* José R. Correa, Alejandro Hevia and Marcos A. Kiwi, editors, *Theoretical Informatics, 7th Latin American Symposium (LATIN 2006), Valdivia, Chile.* Volume 3887 of *Lecture Notes in Computer Science*, pages 238–249, Springer, March 2006, DOI: 10.1007/11682462_25, URL: https://doi.org/10.1007/11682462%5C_25.

[BMR08] Patricia Bouyer, Nicolas Markey and Pierre-Alain Reynier. Robust analysis of timed automata via channel machines, *in* Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference (FOSSACS 2008), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2008), Budapest, Hungary.* Volume 4962 of *Lecture Notes in Computer Science*, pages 157–171, Springer, March 2008, DOI: 10.1007/978-3-540-78499-9_12, URL: https://doi.org/10.1007/978-3-540-78499-9%5C_12.

[BMS11] Patricia Bouyer, Nicolas Markey and Ocan Sankur. Robust model-checking of timed automata via pumping in channel machines, *in* Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems - 9th International Conference (FORMATS 2011), Aalborg, Denmark.* Volume 6919 of *Lecture Notes in Computer Science*, pages 97–112, Springer, September 2011, DOI: 10.1007/978-3-642-24310-3_8, URL: https://doi.org/10.1007/978-3-642-24310-3%5C_8.

[BMS12] Patricia Bouyer, Nicolas Markey and Ocan Sankur. Robust reachability in timed automata: A game-based approach, *in* Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium (ICALP 2012), Warwick, UK, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 128–140, Springer, July 2012, DOI: 10.1007/978-3-642-31585-5_15, URL: https://doi.org/10.1007/978-3-642-31585-5%5C_15.

[BMS13] Patricia Bouyer, Nicolas Markey and Ocan Sankur. Robustness in timed automata, *in* Parosh Aziz Abdulla and Igor Potapov, editors, *Proceedings of the 7th Workshop on Reachability Problems in Computational Models (RP'13), Uppsala, Sweden.* Volume 8169 of *Lecture Notes in Computer Science*, pages 1–18, Springer-Verlag, September 2013, DOI: 10.1007/978-3-642-41036-9_1.

[BMS15] Patricia Bouyer, Nicolas Markey and Ocan Sankur. Robust reachability in timed automata and games: A game-based approach, *Theoretical Computer Science*, 563:43–74, January 2015, DOI: 10.1016/j.tcs.2014.08.014, URL: https://doi.org/10.1016/j.tcs.2014.08.014.

[CHP11] Krishnendu Chatterjee, Thomas A. Henzinger and Vinayak S. Prabhu. Timed parity games: complexity and robustness, *Logical Methods in Computer Science*, 7*(4)*, December 2011, DOI: 10.2168/LMCS-7(4:8)2011, URL: https://doi.org/10.2168/LMCS-7(4:8)2011.

[CHR02] Franck Cassez, Thomas A. Henzinger and Jean-François Raskin. A comparison of control problems for timed and hybrid systems, *in* Claire J. Tomlin and Mark R. Greenstreet, editors, *Hybrid Systems: Computation and Control, 5th International Workshop (HSCC 2002), Stanford, CA, USA*. Volume 2289 of *Lecture Notes in Computer Science*, pages 134–148, Springer, March 2002, DOI: 10.1007/3-540-45873-5_13, URL: https://doi.org/10.1007/3-540-45873-5%5C_13.

[CJM+20] Emily Clement, Thierry Jéron, Nicolas Markey and David Mentré. Computing maximally-permissive strategies in acyclic timed automata, *in* Nathalie Bertrand and Nils Jansen, editors, *Formal Modeling and Analysis of Timed Systems - 18th International Conference (FORMATS 2020), Vienna, Austria*. Volume 12288 of *Lecture Notes in Computer Science*, pages 111–126, Springer, September 2020, DOI: 10.1007/978-3-030-57628-8_7, URL: https://doi.org/10.1007/978-3-030-57628-8%5C_7.

[DDM+04] Martin De Wulf, Laurent Doyen, Nicolas Markey and Jean-François Raskin. Robustness and implementability of timed automata, *in* Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS 2004) and Formal Techniques in Real-Time and Fault-Tolerant Systems, (FTRTFT 2004), Grenoble, France*. Volume 3253 of *Lecture Notes in Computer Science*, pages 118–133, Springer, September 2004, DOI: 10.1007/978-3-540-30206-3_10, URL: https://doi.org/10.1007/978-3-540-30206-3%5C_10.

[DDM+08] Martin De Wulf, Laurent Doyen, Nicolas Markey and Jean-François Raskin. Robust safety of timed automata, *Formal Methods in System Design*, 33*(1-3)*:45–84, September 2008, DOI: 10.1007/s10703-008-0056-7, URL: https://doi.org/10.1007/s10703-008-0056-7.

[DDR04] Martin De Wulf, Laurent Doyen and Jean-François Raskin. Almost ASAP semantics: from timed models to timed implementations, *in* Rajeev Alur and

George J. Pappas, editors, *Hybrid Systems: Computation and Control, 7th International Workshop (HSCC 2004), Philadelphia, PA, USA*. Volume 2993 of *Lecture Notes in Computer Science*, pages 296–310, Springer, March 2004, DOI: 10.1007/978-3-540-24743-2_20, URL: https://doi.org/10.1007/978-3-540-24743-2%5C_20.

[DDR05] Martin De Wulf, Laurent Doyen and Jean-François Raskin. Almost ASAP semantics: from timed models to timed implementations, *Formal Aspects of Computating*, 17*(3)*:319–341, August 2005, DOI: 10.1007/s00165-005-0067-8, URL: https://doi.org/10.1007/s00165-005-0067-8.

[Dim07] Catalin Dima. Dynamical properties of timed automata revisited, *in* Jean-François Raskin and P. S. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems, 5th International Conference (FORMATS 2007), Salzburg, Austria*. Volume 4763 of *Lecture Notes in Computer Science*, pages 130–146, Springer, October 2007, DOI: 10.1007/978-3-540-75454-1_11, URL: https://doi.org/10.1007/978-3-540-75454-1%5C_11.

[DK06] Conrado Daws and Piotr Kordy. Symbolic robustness analysis of timed automata, *in* Eugene Asarin and Patricia Bouyer, editors, *Formal Modeling and Analysis of Timed Systems, 4th International Conference (FORMATS 2006), Paris, France*. Volume 4202 of *Lecture Notes in Computer Science*, pages 143–155, Springer, September 2006, DOI: 10.1007/11867340_11, URL: https://doi.org/10.1007/11867340%5C_11.

[GHJ97] Vineet Gupta, Thomas A. Henzinger and Radha Jagadeesan. Robust timed automata, *in* Oded Maler, editor, *The 1997 International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345, Springer-Verlag, March 1997.

[GLM+05] Guillaume Gardey, Didier Lime, Morgan Magnin and Olivier H. Roux. Romeo: A tool for analyzing time petri nets, *in* Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference (CAV 2005), Edinburgh, Scotland, UK*. Volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Springer, July 2005, DOI: 10.1007/11513988_41, URL: https://doi.org/10.1007/11513988%5C_41.

[HR00] Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems, *in* Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control, Third International Workshop (HSCC 2000), Pittsburgh, PA, USA.* Volume 1790 of *Lecture Notes in Computer Science*, pages 145–159, Springer, March 2000, DOI: 10.1007/3-540-46430-1_15, URL: https://doi.org/10.1007/3-540-46430-1%5C_15.

[JR11] Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata, *in* Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference (FOSSACS 2011), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2011), Saarbrücken, Germany.* Volume 6604 of *Lecture Notes in Computer Science*, pages 229–244, Springer, March 2011, DOI: 10.1007/978-3-642-19805-2_16, URL: https://doi.org/10.1007/978-3-642-19805-2%5C_16.

[KP05] Pavel Krcál and Radek Pelánek. On sampled semantics of timed systems, *in* Ramaswamy Ramanujam and Sandeep Sen, editors, *Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, (FSTTCS 2005), Hyderabad, India.* Volume 3821 of *Lecture Notes in Computer Science*, pages 310–321, Springer, December 2005, DOI: 10.1007/11590156_25, URL: https://doi.org/10.1007/11590156%5C_25.

[LRS+09] Didier Lime, Olivier H. Roux, Charlotte Seidner and Louis-Marie Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches, *in* Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference (TACAS 2009), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2009), York, UK.* Volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, Springer, March 2009, DOI: 10.1007/978-3-642-00768-2_6, URL: https://doi.org/10.1007/978-3-642-00768-2%5C_6.

[Mar11] Nicolas Markey. Robustness in real-time systems, *in 6th IEEE International Symposium on Industrial and Embedded Systems (SIES 2011). Vasteras, Sweden.* Pages 28–34, IEEE, June 2011, DOI: 10.1109/SIES.2011.5953652, URL: https://doi.org/10.1109/SIES.2011.5953652.

[ORS14] Youssouf Oualhadj, Pierre-Alain Reynier and Ocan Sankur. Probabilistic robust timed games, *in* Paolo Baldan and Daniele Gorla, editors, *Concurrency Theory - 25th International Conference (CONCUR 2014), Rome, Italy, September 2-5, 2014*. Volume 8704 of *Lecture Notes in Computer Science*, pages 203–217, Springer, September 2014, DOI: 10.1007/978-3-662-44584-6_15, URL: https://doi.org/10.1007/978-3-662-44584-6%5C_15.

[Pur00] Anuj Puri. Dynamical properties of timed automata, *Discrete Event Dynamic Systems*, 10*(1-2)*:87–113, January 2000, DOI: 10.1023/A:1008387132377, URL: https://doi.org/10.1023/A:1008387132377.

[Pur98] Anuj Puri. Dynamical properties of timed automata, *in* Anders P. Ravn and Hans Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 5th International Symposium, (FTRTFT'98), Lyngby, Denmark*. Volume 1486 of *Lecture Notes in Computer Science*, pages 210–227, Springer, September 1998, DOI: 10.1007/BFb0055349, URL: https://doi.org/10.1007/BFb0055349.

[RPV17] Nima Roohi, Pavithra Prabhakar and Mahesh Viswanathan. Robust model checking of timed automata under clock drifts, *in* Goran Frehse and Sayan Mitra, editors, *20th International Conference on Hybrid Systems: Computation and Control (HSCC 2017), Pittsburgh, PA, USA*. Pages 153–162, ACM, April 2017, DOI: 10.1145/3049797.3049821, URL: https://doi.org/10.1145/3049797.3049821.

[San13] Ocan Sankur. Shrinktech: A tool for the robustness analysis of timed automata, *in* Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference (CAV 2013), Saint Petersburg, Russia*. Volume 8044 of *Lecture Notes in Computer Science*, pages 1006–1012, Springer, July 2013, DOI: 10.1007/978-3-642-39799-8_72, URL: https://doi.org/10.1007/978-3-642-39799-8%5C_72.

[SBM+13] Ocan Sankur, Patricia Bouyer, Nicolas Markey and Pierre-Alain Reynier. Robust controller synthesis in timed automata, *in* Pedro R. D'Argenio and Hernán C. Melgratti, editors, *Concurrency Theory - 24th International Conference, (CONCUR 2013), Buenos Aires, Argentina*. Volume 8052 of *Lecture Notes in Computer Science*, pages 546–560, Springer, August 2013, DOI:

10.1007/978-3-642-40184-8_38, URL: https://doi.org/10.1007/978-3-642-40184-8%5C_38.

[SBM11] Ocan Sankur, Patricia Bouyer and Nicolas Markey. Shrinking timed automata, *in* Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011), Mumbai, India*, volume 13 of *LIPIcs*, pages 90–102, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, December 2011, DOI: 10.4230/LIPIcs.FSTTCS.2011.90, URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2011.90.

[SBM14] Ocan Sankur, Patricia Bouyer and Nicolas Markey. Shrinking timed automata, *Information and Computation*, 234:107–132, February 2014, DOI: 10.1016/j.ic.2014.01.002, URL: https://doi.org/10.1016/j.ic.2014.01.002.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming*, Wiley, New York, NY, USA, June 1986.

[SFK08] Mani Swaminathan, Martin Fränzle and Joost-Pieter Katoen. The surprising robustness of (closed) timed automata against clock-drift, *in* Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri and C.-H. Luke Ong, editors, *Fifth IFIP International Conference On Theoretical Computer Science - (TCS 2008), IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science, Milano, Italy*, volume 273 of *IFIP*, pages 537–553, Springer, September 2008, DOI: 10.1007/978-0-387-09680-3_36, URL: https://doi.org/10.1007/978-0-387-09680-3%5C_36.

[SLD+09] Jun Sun, Yang Liu, Jin Song Dong and Jun Pang. PAT: towards flexible verification under fairness, *in* Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference (CAV 2009), Grenoble, France.* Volume 5643 of *Lecture Notes in Computer Science*, pages 709–714, Springer, June 2009, DOI: 10.1007/978-3-642-02658-4_59, URL: https://doi.org/10.1007/978-3-642-02658-4%5C_59.

[Yov97] Sergio Yovine. KRONOS: A verification tool for real-time systems, *International Journal on Software Tools for Technology Transfer*, 1*(1-2)*:123–133, January 1997, DOI: 10.1007/s100090050009, URL: https://doi.org/10.1007/s100090050009.

List of Tables

List of Figures

Appendix A

UML GRAPHS

In this appendix, we represent the UML graphs of the methods of each classes used in the implementations presented in Chapter 6. In section A.1 and A.2 we respectively present the methods of the classes used in the numerical implementation, presented in Section 6.1, and the symbolic implementation, presented in Section 6.2.

A.1 UML graphs of the numerical implementation

TimedAutomaton
init_location : Location goal_location: Location transitions: List[Edge] number_clock: int
maximum_upper_bound() minimum_lower_bound() existence_infinite_weighted_path(location: Location)

Figure A.1: UML graph of the class TimedAutomaton.

Guard
constraints : List[Constraint]
guard_check(valuation: Valuation, delay: Delay) guard_check_interval(valuation: Valuation, interval: Interval) valuation_after_passing_guard(valuation: Valuation, delay: Delay) enabled_delay_set(valuation: Valuation) disjoint(other_guard: Guard)

Figure A.2: UML graph of the class Guard.

Constraint
interval: Interval clock_index: int
constraint_check(valuation: Valuation, delay: Delay) constraint_check_interval(valuation: Valuation, interval: Interval) enabled_delay_set(valuation: Valuation)

Figure A.3: UML graph of the class Constraint.

Interval
lower_bound: Delay upper_bound: Delay closed: 'both', 'none', 'left', 'right'
is_empty() overlaps(other_interval: Interval) is_disjoint_and_mergeable() include(other_interval: Interval) merge(other_interval: Interval) sub_interval(left: Delay, right: Delay) semi_sorted_sampling(step: Union[int, Fraction], bound: Optional[Union[int, Fraction]])

Figure A.4: UML graph of the class Interval.

P-Move
action: Str step: List[Step] global_interval: Interval
get_as_interval() restricted_interval(restricted_interval: Interval)

Figure A.5: UML graph of the class P-move.

Trace
configuration : Configuration p-move: P-Move delay: Delay
compute_trace_permissiveness() compare_trace(other_trace: Trace)

Figure A.6: UML graph of the class Trace.

Backtracking
ta : TimedAutomaton start: Configuration strategy_opponent: p-move:P-Move, step: Union[int, Fraction] → moves:List[P-Move] interval_sampling_step: Union[Fraction, int]
backtracking()

Figure A.7: UML graph of the class Backtracking.

A.2 UML graphs of the symbolic implementation

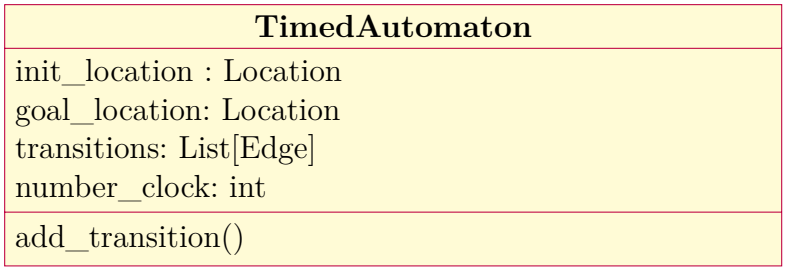


Figure A.8: UML graph of the class TimedAutomaton.

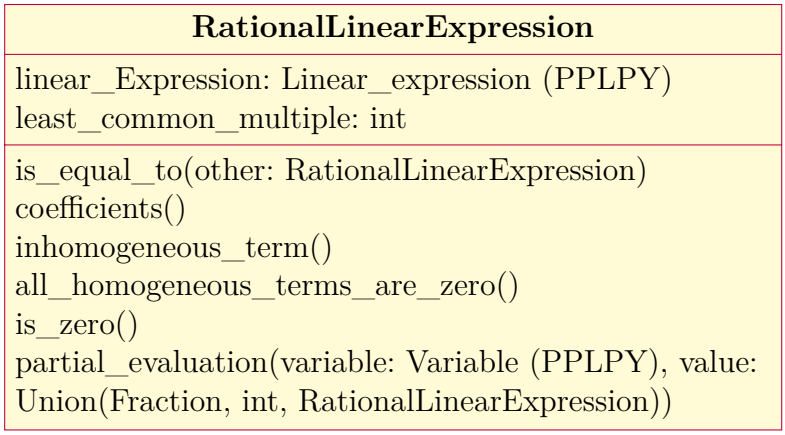


Figure A.9: UML graph of the class RationalLinearExpression.

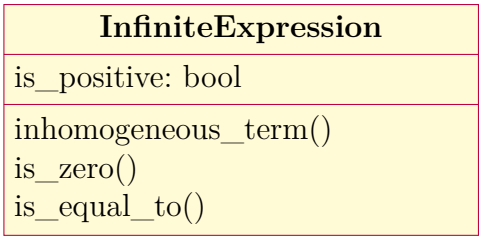


Figure A.10: UML graph of the class InfiniteExpression.

SubSpline
function: Union[InfiniteExpression, RationalLinearExpression] polyhedron: Polyhedron (PPLPY)
minimum(other_spline: SubSpline) maximum(other_spline: SubSpline) is_equal_to(other_spline: SubSpline)

Figure A.11: UML graph of the class SubSpline.

Spline
sub_splines: List[SubSpline]
add_sub_spline(sub_spline: SubSpline) pop_sub_spline(index: int) is_equal_to(other_spline: Spline) fusion(other_spline: Spline) partition(other_spline: Spline) maxmimum(other_spline: Spline) maximum_list(other: Iterable[Spline]) minimum(other_spline: Spline) minimum_list(other: Iterable[Spline])

Figure A.12: UML graph of the class Spline.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Robustesse des automates temporisés : calculer les stratégies les plus permissives

Mot clés : Vérification de modèle, Optimisation, robustesse, automates temporisés

Résumé : Les systèmes temps-réels nécessitent parfois d'être prouvés formellement, en particulier les systèmes temps-réels contenant des parties critiques, comme les avions, les voitures... Les automates temporisés constituent un modèle mathématique commode pour cela. Cependant, même s'ils fournissent une représentation des aspects temporels de ces systèmes, les automates temporisés supposent une précision arbitraire et des actions immédiates. C'est pourquoi même si un état est déclaré atteignable dans un automate temporisé, il est parfois impossible de l'atteindre dans le système physique qu'il modélise.

Le but de cette thèse est de modéliser un type de perturbations, sur des délais, pour les automates temporisés et de calculer les stratégies les plus permissives afin de régler ce problème d'imprécision. Ces stratégies élargiront les délais uniques habituellement proposés en des intervalles de délais et chercheront à atteindre un des états finaux de l'automate quel que soit le délai dans l'intervalle proposé qui a eu lieu.

Title: Robustness of timed automata: computing the maximally-permissive strategies

Keywords: Model verification, optimisation, robustness, timed automaton

Abstract: Real-time systems sometimes need to be formally proven, especially real-time systems containing critical component, as planes, cars etc. Timed automata provide a convenient mathematical model for this. However, although they provide a representation of the temporal aspects of these systems, timed automata assume arbitrary precision and zero-delays actions. Therefore, even if a state is declared reachable in a timed automaton, it is sometimes impossible to reach it in the physical system it models. The aim of this thesis is to model a type of perturbation, on delays, for timed automata and to compute the most permissive strategies to solve this imprecision problem. These strategies propose intervals of delays instead of the usually proposed single delays. These strategies seek to reach one of the final states of the automaton regardless of the delay, in the proposed interval, that has occurred.