Computation Tree Logic for formal verification

Sophie Pinchinat
sophie.pinchinat@irisa.fr

Logica, IRISA/INRIA Rennes

TVA 2016

History

ACM Turing Awards 2007

Recipients in February 2008

- Edmund M. Clarke jr. (CMU, USA)
- Allen E. Emerson (Texas at Austin, USA)
- Joseph Sifakis (IMAG Grenoble, F)

Jury justification

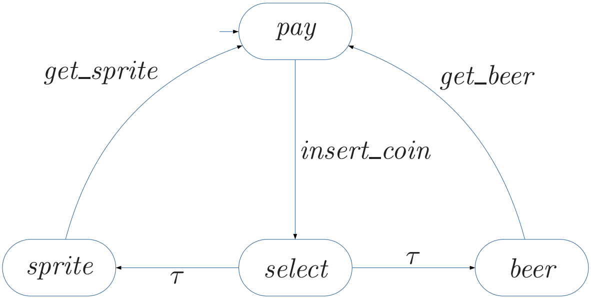
"For their roles in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries."

Outline

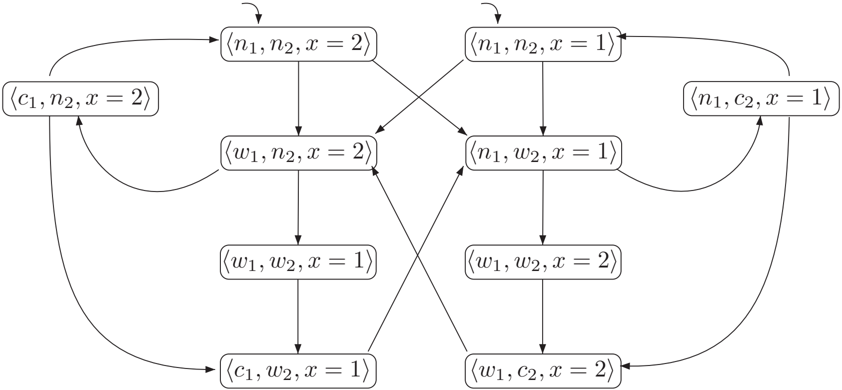
Modelling Systems: Transition Systems

- Model to describe the behaviour of systems
- Directed graph where nodes represent states and edges represent transitions that are “state changes”.

A Beverage Vending Machine



Peterson's Mutual Exclusion

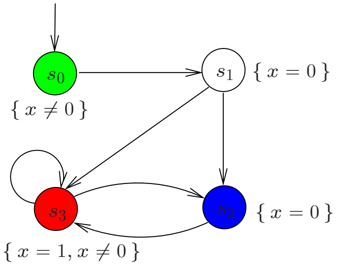


Transition Systems: Formal Definition

$TS = (S, \rightarrow, I, AP, L)$ where

- S is a finite set of states, and $I \subseteq S$ initial states
- $\rightarrow \subseteq S \times S$ is a transition relation
- AP is a finite set of atomic propositions
- $L : S \rightarrow 2^{AP}$ is a labeling function.

The set AP is an abstraction of more refined informations in local states.



Paths in TS and related notions

Let $TS = (S, \rightarrow, I, AP, L)$ be a TS.

- A finite path fragment $\hat{\pi}$ of TS is a state sequence: $\hat{\pi} = s_0 s_1 \dots s_n$ such that $s_i \rightarrow s_{i+1}$ for all $0 \leq i < n$ where $n \geq 0$
- An infinite path fragment π of TS is an infinite state sequence: $\pi = s_0 s_1 s_2 \dots$ such that $s_i \rightarrow s_{i+1}$ for all $0 \leq i < n$
- A path $\pi = s_0 s_1 s_2 \dots$ is an initial (*i.e.* $s_0 \in I$) maximal path fragment

Without loss of generality, we assume that all maximal paths are infinite, and we write $Paths^{TS}(s)$ the set of maximal path fragments π such that $first(\pi) = s$, and simply $Paths^{TS}$ for the set of paths.

For $\pi = s_0 s_1 s_2 \dots \in Paths^{TS}$, we shall write

- $\pi[i] \in S$ for s_i
- $\pi[i..] \in Path(s_i)$ for the path fragment $s_i s_{i+1} \dots$

Example: A Triple Modular Redundant (TMR) System

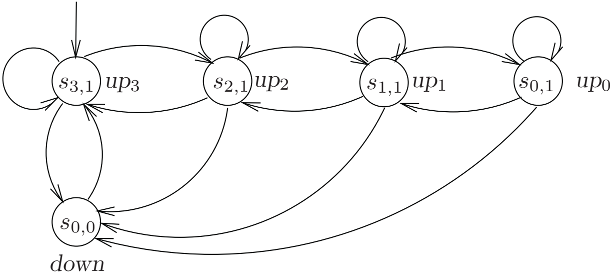
Consider a triple modular redundant (TMR) system with three processors and a single voter. As each component of this system can fail, the reliability is increased by letting all processors execute the same program.

- The voter takes a majority vote of the outputs of the three processors. If a single processor fails, the system can still produce reliable outputs.
- Each component can be repaired. It is assumed that only one component at a time can fail and only one at a time can be repaired.
- On failure of the voter, the entire system fails.
- On repair of the voter, it is assumed that the system starts as being new, i.e., with three processors and a voter.

We consider the TMR system to be operational if at least two processors are functioning properly.

The TS of the Triple Modular Redundant (TMR) System

States are of the form $s_{i,j}$ where i denotes the number of processors that are currently up $(0 < i \leq 3)$ and j the number of operational voters $(j = 0, 1)$.

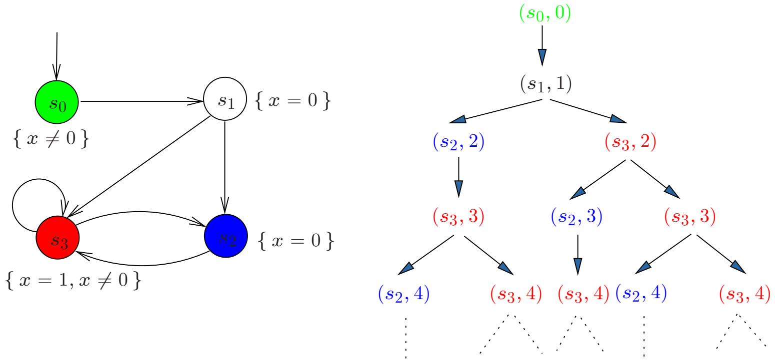


Flipped Classroom

There exist many variant of TS depending on the features one wants to focus on.
Also combinators between TS, such as the parallel composition $||$, enable one to build complex systems from basic ones.
Also more elaborated TS allow one to model, e.g., channels, etc.

Flipped Classroom (PoM, Chapter 2 on "Modelling Concurrent Systems".)

Transition Systems' branching-time behavior: infinite trees



We will use the branching temporal logic CTL whose temporal operators allow the expression of properties of some or all computations that start in a state.

The Whole Picture

M1 MVFA course/Flipped Classroom

	Linear time	Branching time
"behavior" in a state s	path-based: *trace*(s)	state-based: computation tree of s
temporal logic	LTL: path formulas φ $s \models \varphi$ iff $\forall \pi \in \mathit{Paths}(s).\ \pi \models \varphi$	CTL: state formulas existential path quantification $\exists\varphi$ universal path quantification: $\forall\varphi$
complexity of the model checking problems	PSPACE–complete $\mathcal{O}\left(\lvert TS \rvert \cdot 2^{\lvert\varphi\rvert}\right)$	PTIME $\mathcal{O}\left(\lvert TS \rvert \cdot \lvert\Phi\rvert\right)$
implementation-relation	trace inclusion and the like (proof is PSPACE-complete)	simulation and bisimulation (proof in polynomial time)

Outline

Computation Tree Logic

- Clarke and Emerson 1981 "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic", cited more than

3000 times!

Computation Tree Logic Syntax

[CE81, CE82]

- Statements over states
  - $a \in AP$ — atomic proposition
  - $\neg\Phi$ and $\Phi \wedge \Psi$ — negation and conjunction
  - $\exists\varphi$ — there *exists* a path fulfilling φ
  - $\forall\varphi$ — *all* paths fulfill φ
- Statements over paths
  - $\bigcirc\Phi$ — the next state fulfills Φ
  - $\Phi \,\mathsf{U}\, \Psi$ — Φ holds until a Ψ-state is reached

$\Rightarrow$ note that $\bigcirc$ and U *alternate* with $\forall$ and $\exists$

- $\forall\bigcirc\bigcirc\Phi$ and $\forall\exists\bigcirc\Phi \notin$ CTL, but $\forall\bigcirc\forall\bigcirc\Phi$ and $\forall\bigcirc\exists\bigcirc\Phi \in$ CTL

Computation Tree Logic Syntax (as in books)

CTL *state formulae* over the set AP of atomic proposition are formed according to the following grammar:

$$\Phi ::= \text{true} \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi \;\Big|\; \exists\varphi \;\Big|\; \forall\varphi$$

where $a \in AP$ and φ is a path formula. CTL *path formulae* are formed according to the following grammar:

$$\varphi ::= \bigcirc\Phi \;\Big|\; \Phi_1 \mathsf{U} \Phi_2$$

where Φ, Φ_1 and Φ_2 are state formulae. ■

See the [PoM, page 317]

Derived Operators

potentially Φ:	$\exists\Diamond\Phi$	$=$	$\exists(\text{true}\,\mathsf{U}\,\Phi)$
inevitably Φ:	$\forall\Diamond\Phi$	$=$	$\forall(\text{true}\,\mathsf{U}\,\Phi)$
potentially always Φ:	$\exists\Box\Phi$	$:=$	$\neg\forall\Diamond\neg\Phi$
invariantly Φ:	$\forall\Box\Phi$	$=$	$\neg\exists\Diamond\neg\Phi$
weak until:	$\exists(\Phi\,\mathsf{W}\,\Psi)$	$=$	$\neg\forall\big((\Phi\wedge\neg\Psi)\,\mathsf{U}\,(\neg\Phi\wedge\neg\Psi)\big)$
	$\forall(\Phi\,\mathsf{W}\,\Psi)$	$=$	$\neg\exists\big((\Phi\wedge\neg\Psi)\,\mathsf{U}\,(\neg\Phi\wedge\neg\Psi)\big)$

the boolean connectives are derived as usual

Legal CTL Formulae

Let $AP = \{ x = 1, x < 2, x \geqslant 3 \}$ be a set of atomic propositions. Examples of syntactically correct CTL formulae are

$$\exists \bigcirc (x = 1), \forall \bigcirc (x = 1), \text{ and } x < 2 \lor x = 1$$

and $\exists((x < 2) \, \mathsf{U} \, (x \geqslant 3))$ and $\forall(\text{true} \, \mathsf{U} \, (x < 2))$. Some examples of formulae that are syntactically incorrect are

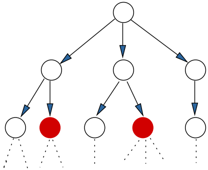
$$\exists(x = 1 \land \forall \bigcirc (x \geqslant 3)) \text{ and } \exists \bigcirc (\text{true} \, \mathsf{U} \, (x = 1)).$$

The first is not a CTL formula since $x = 1 \land \forall \bigcirc (x \geqslant 3)$ is not a path formula and thus must not be preceded by $\exists$ The second formula is not a CTL formula since $\text{true} \, \mathsf{U} \, (x = 1)$ is a path formula rather than a state formula, and thus cannot be preceded by $\bigcirc$. Note that

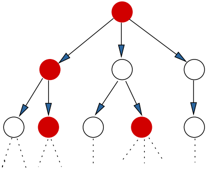
$$\exists \bigcirc (x = 1 \land \forall \bigcirc (x \geqslant 3)) \text{ and } \exists \bigcirc \forall(\text{true} \, \mathsf{U} \, (x = 1))$$

are, however, syntactically correct CTL formulae.

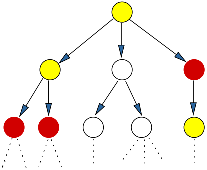
Visualization of semantics



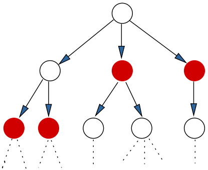
$\exists\Diamond$ *red*



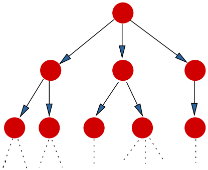
$\exists\Box$ *red*



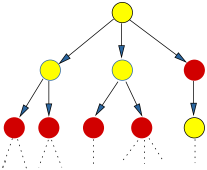
$\exists$(*yellow* U *red*)



$\forall\Diamond$ *red*



$\forall\Box$ *red*



$\forall$(*yellow* U *red*)

Semantics of CTL state-formulas

Defined by a relation $\models$ such that

$s \models \Phi$ if and only if formula Φ holds in state s

$$
\begin{array}{lll}
s \models a & \text{iff} & a \in L(s) \\
s \models \neg\,\Phi & \text{iff} & \neg\,(s \models \Phi) \\
s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \wedge (s \models \Psi) \\
s \models \exists\varphi & \text{iff} & \pi \models \varphi \text{ for } \textit{some} \text{ path } \pi \text{ that starts in } s \\
s \models \forall\varphi & \text{iff} & \pi \models \varphi \text{ for } \textit{all} \text{ paths } \pi \text{ that start in } s
\end{array}
$$

Semantics of CTL path-formulas

Define a relation $\models$ such that

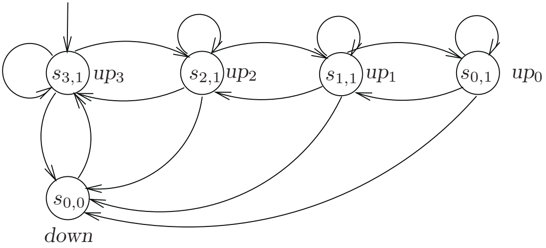
$\pi \models \varphi$ if and only if path π satisfies φ

$$\pi \models \bigcirc\Phi \quad \text{iff } \pi[1] \models \Phi$$

$$\pi \models \Phi \,\mathsf{U}\, \Psi \quad \text{iff } (\exists j \geqslant 0.\, \pi[j] \models \Psi \ \wedge\ (\forall 0 \leqslant k < j.\, \pi[k] \models \Phi))$$

where $\pi[i]$ denotes the state s_i in the path π

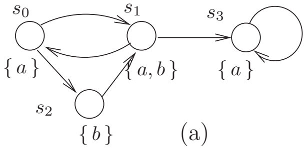
Examples of CTL Properties (1/4) The TMR System



Property	*Formalization in CTL*
Possibly the system never goes down	$\exists\Box\neg down$
Invariantly the system never goes down	$\forall\Box\neg down$
It is always possible to start as new	$\forall\Box\exists\Diamond up_3$
The system always eventually goes down and is operational until going down	$\forall((up_3 \lor up_2)\,\mathsf{U}\,down)$

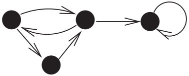
Examples of CTL Properties (2/4)

TS

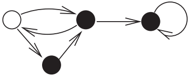


(a)

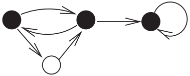
$\exists \bigcirc a$



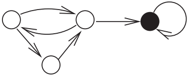
$\forall \bigcirc a$



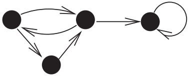
$\exists \Box a$



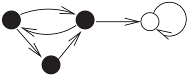
$\forall \Box a$



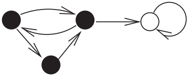
$\exists \Diamond (\exists \Box a)$



$\forall (a \, \mathsf{U} \, b)$



$\exists (a \, \mathsf{U} \, (\neg a \wedge \forall (\neg a \, \mathsf{U} \, b)))$



About Infinitely Often in CTL

Theorem

$s \models \forall \Box \forall \Diamond a$ if and only if for all $\pi \in Path(s), \pi[i] \models a$ for infinitely many i

Proof on the board...

Intuitive examples of CTL Properties (3/4)

- The mutual exclusion property can be described in CTL by the formula

$$\forall\Box(\neg crit_1 \vee \neg crit_2)$$

  The CTL formula

$$(\forall \Box \forall \Diamond crit_1) \wedge (\forall \Box \forall \Diamond crit_2)$$

  requires each process to have access to the critical section infinitely often.

- In case of a traffic light:
  - The safety property "each red light phase is preceded by a yellow light phase" can be formulated in CTL by

$$\forall\Box(yellow \vee \forall \bigcirc \neg red)$$

  - The liveness property "the traffic light is infinitely often green" can be formulated as

$$\forall\Box\forall\Diamond green$$

Intuitive examples of CTL Properties (4/4)

- Progress properties such as "every request will eventually be granted" can be described by
$$\forall\Box(Request \Rightarrow \forall\Diamond response)$$

- The CTL formula
$$\forall\Box\exists\Diamond start$$
expresses that in every reachable system state it is possible to return (via 0 or more transitions) to (one of) the starting state(s).

Semantics of CTL on TS (1/2)

- For CTL-state-formula Φ, the *satisfaction set* *Sat*(Φ) is defined by:

$$\mathit{Sat}(\Phi) \;=\; \{\, s \in S \mid s \models \Phi \,\}$$

- *TS* satisfies CTL-formula Φ iff Φ holds in all its initial states:

$$\mathit{TS} \models \Phi \quad \text{if and only if} \quad \forall s_0 \in I.\, s_0 \models \Phi$$

  - this is equivalent to $I \subseteq \mathit{Sat}(\Phi)$

- Point of attention: *TS* $\not\models \Phi$ and *TS* $\not\models \neg\Phi$ is possible!

Semantics of CTL on TS (2/2)

$TS \not\models \Phi$ and $TS \not\models \neg\Phi$ is possible:

because of several initial states, e.g., $s_0 \models \exists \Box a$ and $s'_0 \not\models \exists \Box a$

Exercise

Any even simpler idea?

CTL Semantics for Transition Systems with Terminal States

Exercise

Adapt the path semantics in case transition systems are considered with terminal states, i.e., when finite paths are possible.

CTL Equivalence - Duality Laws

Definition

$\Phi \equiv \Psi$ if and only if for all transition system TS,

$$TS \models \Phi \Leftrightarrow TS \models \Psi$$

$$\forall \bigcirc \Phi \equiv \neg \exists \bigcirc \neg \Phi$$

$$\exists \bigcirc \Phi \equiv \neg \forall \bigcirc \neg \Phi$$

$$\forall \Diamond \Phi \equiv \neg \exists \Box \neg \Phi$$

$$\exists \Diamond \Phi \equiv \neg \forall \Box \neg \Phi$$

$$\forall (\Phi \, \mathsf{U} \, \Psi) \equiv \neg \exists ((\Phi \wedge \neg \Psi) \, \mathsf{W} \, (\neg \Phi \wedge \neg \Psi))$$

Exercise

Write proofs

CTL Equivalence - Expansion Laws

$$\forall(\Phi \, \mathsf{U} \, \Psi) \equiv \Psi \vee (\Phi \wedge \forall \bigcirc \forall(\Phi \, \mathsf{U} \, \Psi))$$

$$\forall\Diamond\Phi \equiv \Phi \vee \forall \bigcirc \forall\Diamond\Phi$$

$$\forall\Box\Phi \equiv \Phi \wedge \forall \bigcirc \forall\Box\Phi$$

$$\exists(\Phi \, \mathsf{U} \, \Psi) \equiv \Psi \vee (\Phi \wedge \exists \bigcirc \exists(\Phi \, \mathsf{U} \, \Psi))$$

$$\exists\Diamond\Phi \equiv \Phi \vee \exists \bigcirc \exists\Diamond\Phi$$

$$\exists\Box\Phi \equiv \Phi \wedge \exists \bigcirc \exists\Box\Phi$$

Exercise

Write proofs

Distributive Laws (1/2)

$$\forall\Box(\Phi \wedge \Psi) \equiv \forall\Box\Phi \wedge \forall\Box\Psi$$

$$\exists\Diamond(\Phi \vee \Psi) \equiv \exists\Diamond\Phi \vee \exists\Diamond\Psi$$

note that $\exists\Box(\Phi \wedge \Psi) \not\equiv \exists\Box\Phi \wedge \exists\Box\Psi$ and $\forall\Diamond(\Phi \vee \Psi) \not\equiv \forall\Diamond\Phi \vee \forall\Diamond\Psi$

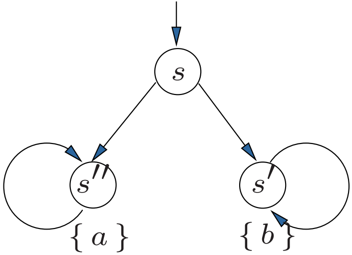
Exercise

Write proofs

Exercise

Argue why $\forall\Diamond(\Phi \vee \Psi) \not\equiv \forall\Diamond\Phi \vee \forall\Diamond\Psi$ entails $\exists\Box(\Phi \wedge \Psi) \not\equiv \exists\Box\Phi \wedge \exists\Box\Psi$, then prove $\forall\Diamond(\Phi \vee \Psi) \not\equiv \forall\Diamond\Phi \vee \forall\Diamond\Psi$.

$\forall \Diamond (\Phi \vee \Psi) \not\equiv \forall \Diamond \Phi \vee \forall \Diamond \Psi$



$s \models \forall\Diamond (a \vee b)$ since for all $\pi \in$ *Paths*(s). $\pi \models \Diamond (a \vee b)$

But: $s\,(s'')^\omega \models \Diamond a$ but $s\,(s'')^\omega \not\models \Diamond b$ Thus: $s \not\models \forall\Diamond b$

A similar reasoning applied to path $s\,(s')^\omega$ yields $s \not\models \forall\Diamond a$

Thus, $s \not\models \forall\Diamond a \vee \forall\Diamond b$

Existential Normal Forms

For $a \in AP$, the set of CTL state formulae in *existential normal form* (ENF, for short) is given by

$$\Phi \quad ::= \quad \text{true} \;\big|\; a \;\big|\; \Phi_1 \wedge \Phi_2 \;\big|\; \neg\Phi \;\big|\; \exists \bigcirc \Phi \;\big|\; \exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\big|\; \exists \Box \Phi.$$

Theorem

For each CTL formula there exists an equivalent CTL formula in ENF, but with an exponential blowup.

Proof.

Use the duality laws for elimination of $\forall$ path quantifier:

$$\begin{aligned} \forall \bigcirc \Phi \quad &\equiv \quad \neg\exists \bigcirc \neg\Phi \\ \forall (\Phi \,\mathsf{U}\, \Psi) \quad &\equiv \quad \neg\exists (\neg\Psi \,\mathsf{U}\, (\neg\Phi \wedge \neg\Psi)) \wedge \neg\exists \Box \neg\Psi \end{aligned}$$

Exercise

Make the proofs, also what would you suggest for $\forall \Diamond \Phi$ and $\forall \Box \Phi$?

□

Notice the exponential blowup of the translation from CTL to CTL in ENF.

Positive Normal Forms

The set of CTL state formulae in *positive normal form* (PNF, for short) is given by

$$\Phi \quad ::= \quad \text{true} \;\Big|\; \text{false} \;\Big|\; a \;\Big|\; \neg a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \Phi_1 \vee \Phi_2 \;\Big|\; \exists\varphi \;\Big|\; \forall\varphi$$

where $a \in AP$ and the path formulae are given by

$$\varphi \quad ::= \quad \bigcirc\Phi \;\Big|\; \Phi_1 \,\mathsf{U}\, \Phi_2 \;\Big|\; \Phi_1 \,\mathsf{W}\, \Phi_2.$$

Theorem

For each CTL formula there exists an equivalent CTL formula in PNF

Proof.

Use the equivalence laws. □

Notice that a law like $\neg\forall\,(\Phi \,\mathsf{U}\, \Psi) \equiv \exists\,((\neg\Psi)\,\mathsf{W}\,(\neg\Phi \wedge \neg\Psi))$ yields an exponential blowup in the translation.

Exercice

What if you allow for the release operator R of the [PoM, page 334]?

Outline

Model Checking

Wikipedia

In computer science, model checking is: Given a model of a system, exhaustively and automatically check whether this model meets a given specification.
Typically, the systems one has in mind are hardware or software systems, and the specification contains safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash. Model checking is a technique for automatically verifying correctness properties of finite-state systems.
To solve such a problem algorithmically, both the model of the system and the specification are formulated in some precise mathematical language: To this end, it is formulated as a task in logic, namely to check whether a given structure satisfies a given logical formula.

Our setting

Given a transition system TS and a formula $\Phi \in CTL$,

$$TS \models \Phi?$$

Preliminaries assumptions

- The transition system TS is finite with no terminal state
- Formula Φ is in Exitential Normal Form (recall)

$$\Phi ::= \texttt{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists \bigcirc \Phi \mid \exists (\Phi_1 \, \mathsf{U} \, \Phi_2) \mid \exists \Box \Phi$$

Basic Algorithm

Consider $TS = (S, \rightarrow, I, AP, L)$ and $\Phi \in CTL$

- The set Sat(Φ) is computed recursively
- It follows that $TS \models \Phi$ if and only if $I \subseteq$ Sat(Φ)

The Model Checking is *global* because we answer a more general problem than "$TS \models \Phi$?", but "$s \models \Phi$?" for all $s \in S$

The Basic Algorithm

We proceed with a bottom-up traversal of the parse tree of the CTL state formula Φ

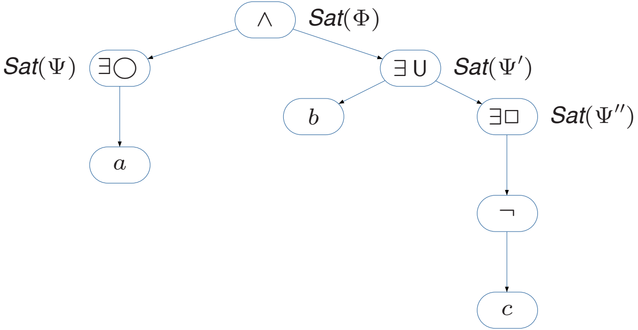
Input: finite transition system *TS* and CTL formula Φ (both over *AP*)
Output: $TS \models \Phi$

(* compute the sets $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$ *)

for all $i \leqslant |\Phi|$ **do**
  for all $\Psi \in Sub(\Phi)$ with $|\Psi| = i$ **do**
    compute $Sat(\Psi)$ from $Sat(\Psi')$ (* for maximal proper $\Psi' \in Sub(\Psi)$ *)
  od
od
return $I \subseteq Sat(\Phi)$

where Sub(Φ) is the set of subformulas of Φ and $|\Phi|$ is the length of Φ, i.e., the number of symbols

Example



$$\Phi = \underbrace{\exists \bigcirc a}_{\Psi} \wedge \underbrace{\exists(b \,\mathsf{U}\, \underbrace{\exists \Box \neg c}_{\Psi''})}_{\Psi'} \quad .$$

Characterization of Sat(.) (1/2)

Let Post$(s) := \{s' \mid s \to s'\}$ be the set of successor states of s

For all CTL formulas Φ, Ψ over *AP* it holds:

$$
\begin{aligned}
\mathit{Sat}(\text{true}) &= S \\
\mathit{Sat}(a) &= \{\, s \in S \mid a \in L(s) \,\}, \text{ for any } a \in AP \\
\mathit{Sat}(\Phi \wedge \Psi) &= \mathit{Sat}(\Phi) \cap \mathit{Sat}(\Psi) \\
\mathit{Sat}(\neg\Phi) &= S \setminus \mathit{Sat}(\Phi) \\
\mathit{Sat}(\exists \bigcirc \Phi) &= \{\, s \in S \mid \mathit{Post}(s) \cap \mathit{Sat}(\Phi) \neq \varnothing \,\}
\end{aligned}
$$

Characterization of Sat(.) (2/2)

- $\mathit{Sat}(\exists(\Phi \mathsf{U} \Psi))$ is the smallest subset T of S, such that:

  $(1)\ \mathit{Sat}(\Psi) \subseteq T$ and $(2)\ (s \in \mathit{Sat}(\Phi)$ and $\mathit{Post}(s) \cap T \neq \varnothing) \ \Rightarrow\ s \in T$

- $\mathit{Sat}(\exists\Box\,\Phi)$ is the largest subset T of S, such that:

  $(3)\ T \subseteq \mathit{Sat}(\Phi)$ and $(4)\ s \in T$ implies $\mathit{Post}(s) \cap T \neq \varnothing$

Characterization of $\mathsf{Sat}(\exists\,(\Phi \,\mathsf{U}\, \Psi))$

Proposition

$\mathsf{Sat}(\exists\,(\Phi \,\mathsf{U}\, \Psi))$ is the smallest set $T \subseteq S$ such that

(1) $\mathsf{Sat}(\Psi) \subseteq T$

(2) $s \in \mathsf{Sat}(\Phi)$ and $\mathsf{Post}(s) \cap T \neq \emptyset$ imply $s \in T$

Proof.

(i) Show that $\mathsf{Sat}(\exists\,(\Phi \,\mathsf{U}\, \Psi))$ satisfies (1) and (2)

(ii) Show that any T satisfying (1) and (2) is such that $\mathsf{Sat}(\exists\,(\Phi \,\mathsf{U}\, \Psi)) \subseteq T$

See details at [PoM, page 344] □

Fix-point characterization of Sat($\exists\,(\Phi \,\mathsf{U}\, \Psi)$)

We just have seen that:

Proposition

Sat($\exists\,(\Phi \,\mathsf{U}\, \Psi)$) is the smallest set $T \subseteq S$ such that

(1) Sat(Ψ) $\subseteq T$

(2) $s \in$ Sat(Φ) and Post$(s) \cap T \neq \emptyset$ imply $s \in T$

Notice that, because of the expansion laws, $\exists\,(\Phi \,\mathsf{U}\, \Psi)$ is a solution of the equation $Z \equiv \Psi \vee \Phi \wedge \exists \bigcirc Z$ (where Z is a variable), but there are others, e.g., $\exists\,(\Phi \,\mathsf{W}\, \Psi)$ is another one.

Proposition

Sat($\exists\,(\Phi \,\mathsf{U}\, \Psi)$) is the smallest set $T \subseteq S$ satisfying

$$\text{Sat}(\Psi) \cup \{s \in \text{Sat}(\Phi) \mid \text{Post}(s) \cap T \neq \emptyset\} = T$$

Fix-point characterization of Sat($\exists \Box \Phi$)

Proposition

Sat($\exists \Box \Phi$) is the largest set $T \subseteq S$ such that

(3) $T \subseteq$ Sat(Φ)

(4) $s \in T$ implies Post(s) $\cap T \neq \emptyset$

Proof.

(i) Show that Sat($\exists \Box \Phi$) satisfies (3) and (4)

(ii) Show that any T satisfying (3) and (4) is such that $T \subseteq$ Sat($\exists \Box \Phi$)

See details at [PoM, page 345] (+ an ERRATUM in the book) $\Box$

What is the set-theoretic counterpart for Sat($\exists \Box \Phi$)?

Computation of Sat(Φ)

switch(Φ):

a : **return** $\{ s \in S \mid a \in L(s) \}$;

$\ldots$: $\ldots\ldots$

$\exists \bigcirc \Psi$: **return** $\{ s \in S \mid Post(s) \cap Sat(\Psi) \neq \varnothing \}$;

$\exists(\Phi_1 \, \mathsf{U} \, \Phi_2)$:

```
T := Sat(Φ2);     (* compute the smallest fixed point *)
while { s ∈ Sat(Φ1) \ T | Post(s) ∩ T ≠ ∅ } ≠ ∅ do
   let s ∈ { s ∈ Sat(Φ1) \ T | Post(s) ∩ T ≠ ∅ };
   T := T ∪ { s };
od;
return T;
```

$\exists \Box \Phi$:

```
T := Sat(Φ);      (* compute the greatest fixed point *)
while { s ∈ T | Post(s) ∩ T = ∅ } ≠ ∅ do
   let s ∈ { s ∈ T | Post(s) ∩ T = ∅ };
   T := T \ { s };
od;
return T;
```

end switch

We now look at a more detailed version of the backward search for Sat($\exists(\Phi \, \mathsf{U} \, \Psi)$) which exploits its characterization as a least fixed-point.

Compute Sat($\exists(\Phi \,\mathsf{U}\, \Psi)$) (1/3)

- *Sat*($\exists(\Phi \,\mathsf{U}\, \Psi)$) is the smallest set $T \subseteq S$ such that:

$$(1)\ \mathit{Sat}(\Psi) \subseteq T \quad \text{and} \quad (2)\ (s \in \mathit{Sat}(\Phi) \text{ and } \mathit{Post}(s) \cap T \neq \varnothing) \ \Rightarrow\ s \in T$$

- This suggests to compute *Sat*($\exists(\Phi \,\mathsf{U}\, \Psi)$) iteratively:

$$T_0 \ =\ \mathit{Sat}(\Psi) \quad \text{and} \quad T_{i+1} \ =\ T_i \,\cup\, \{\, s \in \mathit{Sat}(\Phi) \mid \mathit{Post}(s) \cap T_i \neq \varnothing \,\}$$

- T_i = states that can reach a Ψ-state in at most i steps via a Φ-path
- By induction on j it follows:

$$T_0 \subseteq T_1 \subseteq \ldots \subseteq T_j \subseteq T_{j+1} \subseteq \ \ldots\ \subseteq\ \mathit{Sat}(\exists(\Phi \,\mathsf{U}\, \Psi))$$

Computing Sat($\exists(\Phi \mathsf{U} \Psi)$) (2/3)

- *TS* is finite, so for some $j \geqslant 0$ we have: $T_j = T_{j+1} = T_{j+2} = \dots$
- Therefore: $T_j = T_j \cup \{ s \in \mathit{Sat}(\Phi) \mid \mathit{Post}(s) \cap T_j \neq \varnothing \}$
- Hence: $\{ s \in \mathit{Sat}(\Phi) \mid \mathit{Post}(s) \cap T_j \neq \varnothing \} \subseteq T_j$
  - hence, T_j satisfies (2), i.e., $(s \in \mathit{Sat}(\Phi)$ and $\mathit{Post}(s) \cap T_j \neq \varnothing) \Rightarrow s \in T_j$
  - further, $\mathit{Sat}(\Psi) = T_0 \subseteq T_j$ so, T_j satisfies (1), i.e. $\mathit{Sat}(\Psi) \subseteq T_j$
- As $\mathit{Sat}(\exists(\Phi \mathsf{U} \Psi))$ is the *smallest* set satisfying (1) and (2):
  - $\mathit{Sat}(\exists(\Phi \mathsf{U} \Psi)) \subseteq T_j$ and thus $\mathit{Sat}(\exists(\Phi \mathsf{U} \Psi)) = T_j$
- Hence: $T_0 \subsetneq T_1 \subsetneq T_2 \subsetneq \dots \subsetneq T_j = T_{j+1} = \dots = \mathit{Sat}(\exists(\Phi \mathsf{U} \Psi))$

Computing Sat($\exists(\Phi \, \mathsf{U} \, \Psi)$) (3/3)

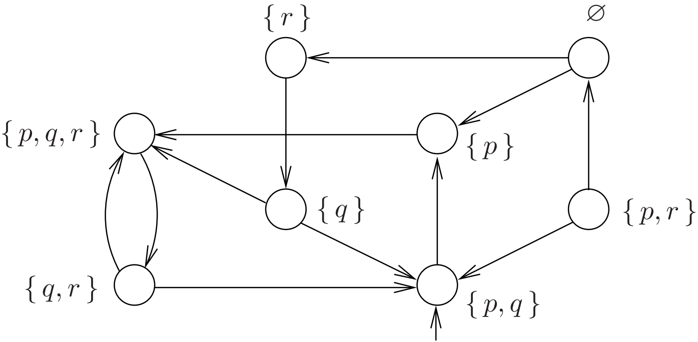
The algorithm assumes a transition system representation by means of "inverse" adjacency lists, based on $\mathsf{Pre}(s') := \{s \in S \mid s' \in \mathsf{Post}(s)\}$

Input: finite transition system *TS* with state-set S and CTL-formula $\exists(\Phi \, \mathsf{U} \, \Psi)$
Output: *Sat*$(\exists(\Phi \, \mathsf{U} \, \Psi)) = \{ s \in S \mid s \models \exists(\Phi \, \mathsf{U} \, \Psi) \}$

E := *Sat*(Ψ); (* E administers the states s with $s \models \exists(\Phi \, \mathsf{U} \, \Psi)$ *)
T := E; (* T contains the already visited states s with $s \models \exists(\Phi \, \mathsf{U} \, \Psi)$ *)
while $E \neq \varnothing$ **do**
  let $s' \in E$;
  $E := E \setminus \{ s' \}$;
  for all $s \in$ *Pre*(s') **do**
    if $s \in$ *Sat*$(\Phi) \setminus T$ **then** $E := E \cup \{ s \}; T := T \cup \{ s \}$; **endif**
  od
od
return T

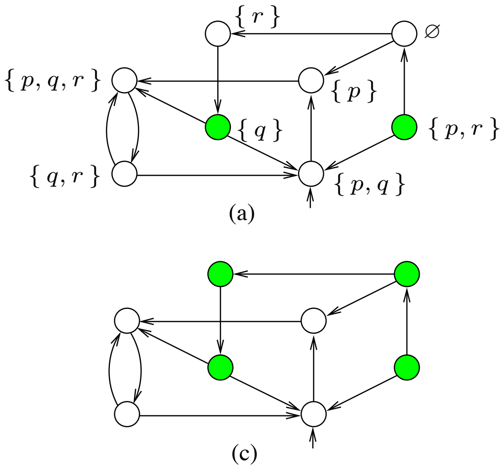
Example

Example



let's check the CTL-formula $\exists\Diamond\left((p = r) \wedge (p \neq q)\right)$

The Computation in Snapshots



Computing Sat($\exists \Box \Phi$) (1/2)

The basic idea is to compute $Sat(\exists\Box\,\Phi)$ by means of the iteration

$$T_0 \;=\; Sat(\Phi) \quad \text{and} \quad T_{i+1} \;=\; T_i \;\cap\; \{\, s \in Sat(\Phi) \mid Post(s) \cap T_i \neq \varnothing \,\}.$$

Then, for all $j \geqslant 0$, it holds that

$$T_0 \supsetneqq T_1 \supsetneqq T_2 \supsetneqq \ldots \supsetneqq T_j = T_{j+1} = \ldots = T = Sat(\exists\Box\,\Phi).$$

The above iteration can be realized by means of a *backward search* starting with

$$T = Sat(\Phi) \quad \text{and} \quad E = S \setminus Sat(\Phi).$$

Here T equals T_0 and E contains all states that refute $\exists\Box\,\Phi$. During the backward search, states are iteratively removed from T, for which it has been established that they refute $\exists\Box\,\Phi$. This applies to any $s \in T$ satisfying

$$Post(s) \cap T \;=\; \varnothing.$$

Although $s \models \Phi$ (as it is in T), all its successors refute $\exists\Box\,\Phi$ (as they are not in T), and therefore s refutes $\exists\Box\,\Phi$. Once such states are encountered, they are inserted in E to enable the possible removal of other states in T.

Computing Sat($\exists \Box \Phi$) (2/2)

In order to support the test whether Post$(s) \cap T = \emptyset$, a counter $c[s]$ is exploited that keeps track of the number of direct successors in $T \cup E$:
$c[s] = |\text{Post}(s) \cap T|$

```
E := S \ Sat(Φ);                           (* E contains any not visited s' with s' ⊭ ∃□Φ *)

T := Sat(Φ);                               (* T contains any s for which s ⊨ ∃□Φ has not yet been disproven *)

for all s ∈ Sat(Φ) do c[s] := | Post(s) |; od        (* initialize array c *)

while E ≠ ∅ do
                                           (* loop invariant: c[s] = | Post(s) ∩ (T ∪ E) | *)
  let s' ∈ E;                              (* s' ⊭ Φ *)
  E := E \ { s' };                         (* s' has been considered *)
  for all s ∈ Pre(s') do
    if s ∈ T then
      c[s] := c[s] − 1;                    (* update counter c[s] for predecessor s of s' *)
      if c[s] = 0 then
        T := T \ { s }; E := E ∪ { s };    (* s does not have any successor in T *)
      fi
    fi
  od
od
return T
```

Let's practice

Exercice

Simulate the execution of the algorithm for Sat($\exists \Box \Phi$) on the structure of Slide 53 for the formula $\exists \Box q$.

Exercice

In the set-theoretic framework, give a characterization of:

- Sat($\forall \bigcirc \Phi$)
- Sat($\forall \Box \Phi$)
- Sat($\forall (\Phi \, \mathsf{U} \, \Psi)$)
- Sat($\exists (\Phi \, \mathsf{W} \, \Psi)$)
- Sat($\forall (\Phi \, \mathsf{W} \, \Psi)$)

Exercice

Adapt Algorithm for formulas $\exists \Box \Phi$ of Slide 55 to formulas $\exists (\Phi \, \mathsf{W} \, \Psi)$

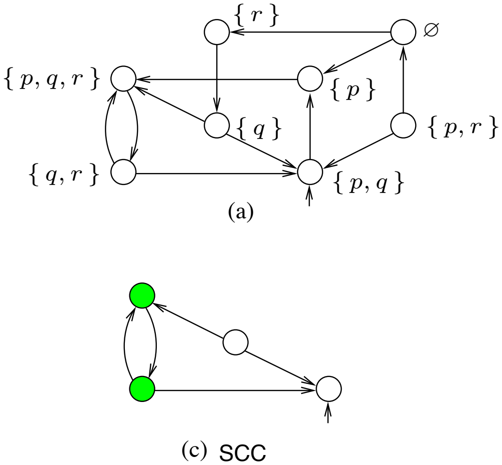
An alternative algorithm for Sat($\exists \Box \Phi$)

1. Consider only state s if $s \models \Phi$, otherwise *eliminate* s
   - change *TS* into *TS*$[\Phi] = (S', Act, \rightarrow', I', AP, L')$ with $S' = Sat(\Phi)$,
   - $\rightarrow' = \rightarrow \cap (S' \times Act \times S')$, $I' = I \cap S'$, and $L'(s) = L(s)$ for $s \in S'$

   $\Rightarrow$ all removed states will not satisfy $\exists \Box \Phi$, and thus can be safely removed
2. Determine all *non-trivial strongly connected components* in *TS*$[\Phi]$
   - non-trivial SCC = maximal, connected subgraph with at least one transition

   $\Rightarrow$ any state in such SCC satisfies $\exists \Box \Phi$
3. $s \models \exists \Box \Phi$ is equivalent to "some *SCC is reachable* from s"
   - this search can be done in a backward manner

Example for $\exists \Box q$



Time Complexity of the CTL Model Checking

Theorem

For transition system TS with N states and K transitions, and CTL formula Φ, the CTL model-checking problem $TS \models \Phi$ can be determined in time

$$O(|\Phi|.(N+K))$$

Proof as a fairly long exercise:

- Consider arbitrary CTL formulas, as ENF yields an exponential blowup
- Treat the modalities $\forall$ U , $\forall \Diamond$, $\forall \Box$, $\exists \Diamond$, etc. analogously to the introduced approaches for $\exists$ U and $\exists \Box$.

Flipped Classroom

Flipped Classroom (PoM, in Chapters 3 and 6 about "Fairness assumptions".)

Counterexample generation for refuted formulas

- Model checking is an effective and efficient "bug hunting" technique
- Counterexamples are of utmost importance:
  - diagnostic feedback, the key to abstraction-refinement, schedule synthesis . . .
- LTL: counterexamples are finite paths
  - $\bigcirc\Phi$: a path on which the next state refutes Φ
  - $\Box\Phi$: a path leading to a $\neg\Phi$-state
  - $\Diamond\Phi$: a $\neg\Phi$-path leading to a $\neg\Phi$ cycle
- Counterexample generation for LTL:
  - use stack contents of nested DFS on encountering an accept cycle
  - use a variant of BFS top find shortest counterexamples

Counterexamples for CTL

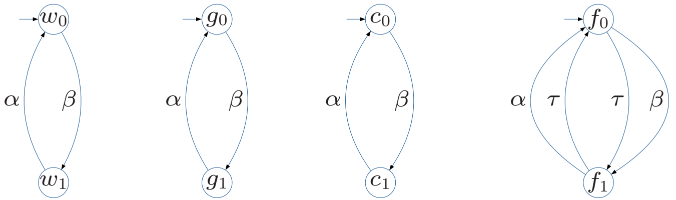
- $TS \not\models \forall\varphi$ where $\forall\varphi$ is also on LTL
  - counterexample = a sufficiently long prefix of a path refuting φ (as in LTL)
  - this is a subset of the so-called universal fragment of CTL
- $TS \not\models \exists\varphi$ where φ is arbitrary CTL formula
  - all paths satisfy φ! $\Rightarrow$ no clear notion of counterexample
  - witness = a sufficiently long prefix of a path satisfying φ
- So:
  - for $\forall\varphi$, a prefix of π with $\pi \not\models \varphi$ acts as counterexample
  - for $\exists\varphi$, a prefix of π with $\pi \models \varphi$ acts as witness

The wolf-goat-cabbage problem (1/5)

- A goat (g), a cabbage (c) and a wolf (w) and two riverbanks (0 and 1)
  - A boat with ferryman (f) that can carry at most two occupants
  - Only the ferryman can steer the boat
  - Goat and cabbage, goat and wolf should neither travel nor left together
- Is there a schedule such that brings c, g, and w to the other side?
- . . . Model this as a CTL model-checking problem
  - transition system *TS* $= \ (wolf \ ||| \ goat \ ||| \ cabbage) \ || \ ferryman$
  - check whether *TS* $\models \exists\varphi$ with

$$\varphi \ = \ \left(\bigwedge_{i=0,1} (w_i \wedge g_i \rightarrow f_i) \ \wedge \ (c_i \wedge g_i \rightarrow f_i) \right) \ \mathsf{U} \ (c_1 \wedge f_1 \wedge g_1 \wedge w_1)$$

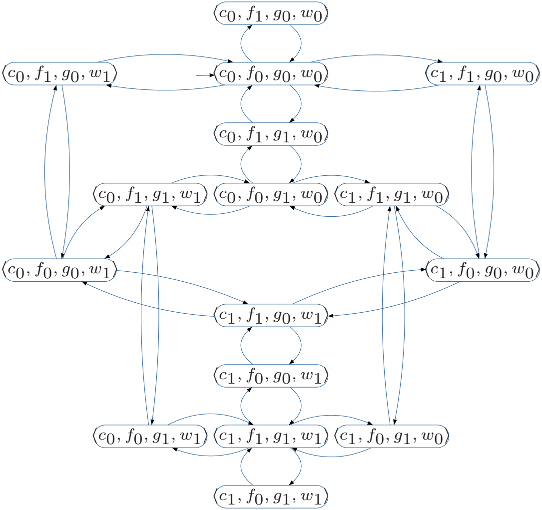
The wolf-goat-cabbage problem (2/5)



$$\textbf{TS} = (wolf \mid\mid\mid goat \mid\mid\mid cabbage) \parallel ferryman$$

$\mid\mid\mid$ is interleaving parallel composition and $\parallel$ is synchronized parallel composition

The wolf-goat-cabbage problem (3/5)



The wolf-goat-cabbage problem (4/5)

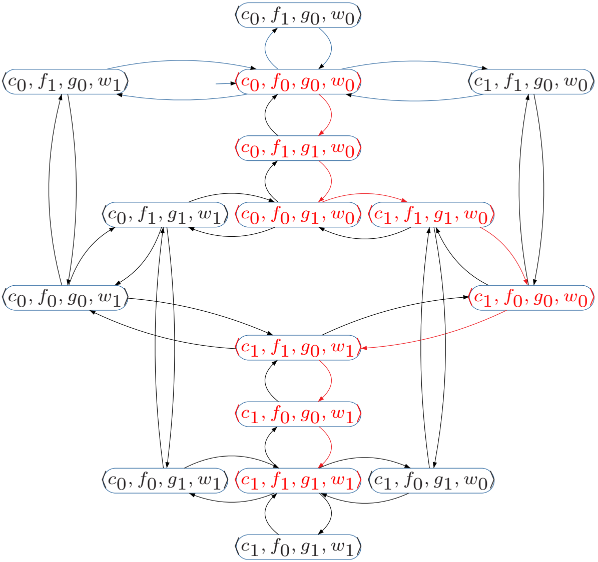
A witness of $\exists\varphi$ with:

$$\varphi = \left(\bigwedge_{i=0,1} (w_i \wedge g_i \rightarrow f_i) \;\wedge\; (c_i \wedge g_i \rightarrow f_i) \right) \mathsf{U} \; (c_1 \wedge f_1 \wedge g_1 \wedge w_1)$$

is a path fragment from initial state $\langle c_0, f_0, g_0, w_0 \rangle$ to target state $\langle c_1, f_1, g_1, w_1 \rangle$ such that g, c and g, w are not left on a single riverbank. Such as:

$\langle c_0, f_0, g_0, w_0 \rangle$	goat to riverbank 1
$\langle c_0, f_1, g_1, w_0 \rangle$	ferryman comes back to riverbank 0
$\langle c_0, f_0, g_1, w_0 \rangle$	cabbage to riverbank 1
$\langle c_1, f_1, g_1, w_0 \rangle$	goat back to riverbank 0
$\langle c_1, f_0, g_0, w_0 \rangle$	wolf to riverbank 1
$\langle c_1, f_1, g_0, w_1 \rangle$	ferryman comes back to riverbank 0
$\langle c_1, f_0, g_0, w_1 \rangle$	goat to riverbank 1
$\langle c_1, f_1, g_1, w_1 \rangle$	

The wolf-goat-cabbage problem (5/5)



Counterexamples for $\bigcirc \Phi$

- A counterexample of $\bigcirc\Phi$ is a path fragment $s\,s'$ with
  - $s \in I$ and $s' \in \mathit{Post}(s)$ with $s' \not\models \Phi$
- A witness of $\bigcirc\Phi$ is a is a path fragment $s\,s'$ with
  - $s \in I$ and $s' \in \mathit{Post}(s)$ with $s' \models \Phi$
- Algorithm: inspection of direct successors of initial states

Counterexamples for $\Phi \,\mathsf{U}\, \Psi$

- A witness is an initial path fragment $s_0\, s_1 \ldots s_n$ with
  - $s_n \models \Psi$ and $s_i \models \Phi$ for $0 \leqslant i < n$
- Algorithm: backward search starting in the set of Ψ-states
- A counterexample is an initial path fragment that indicates a path π:
  - for which either $\pi \models \Box(\Phi \wedge \neg\Psi)$ or $\pi \models (\Phi \wedge \neg\Psi) \,\mathsf{U}\, (\neg\Phi \wedge \neg\Psi)$
- Counterexample is initial path fragment of either form:
  - $\underbrace{s_0 \ldots s_{n-1}\, \underbrace{s_n\, s_1' \ldots s_r'}_{\text{cycle}}}_{\text{satisfy } \Phi \wedge \neg\Psi}$ with $s_n = s_r'$ or $\underbrace{s_0 \ldots s_{n-1}}_{\text{satisfy } \Phi \wedge \neg\Psi}\; s_n$ with $s_n \models \neg\Phi \wedge \neg\Psi$

Counter examples generation for $\Phi \,\mathsf{U}\, \Psi$

Determine the SCCs by of the digraph $G = (S, E)$ where

$$E = \{ (s, s') \in S \times S \mid s' \in \textit{Post}(s) \;\wedge\; s \models \Phi \wedge \neg\Psi \}$$

Each path in G that starts in an initial state $s_0 \in S$ and leads to a non-trivial SCC C in G provides a counterexample of the form:

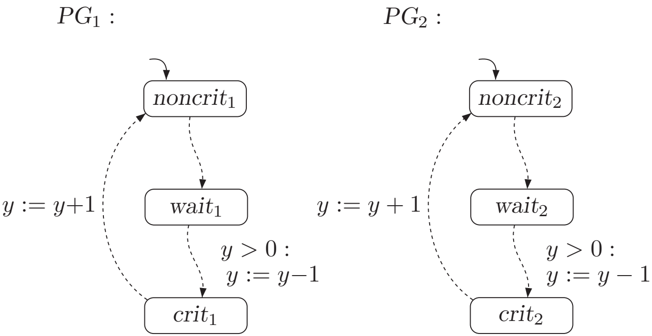
$$s_0\, s_1 \ldots s_n \underbrace{s'_1 \ldots s'_r}_{\in C} \quad \text{with} \quad s_n = s'_r$$

Each path in G that leads from an initial state s_0 to a trivial terminal SCC

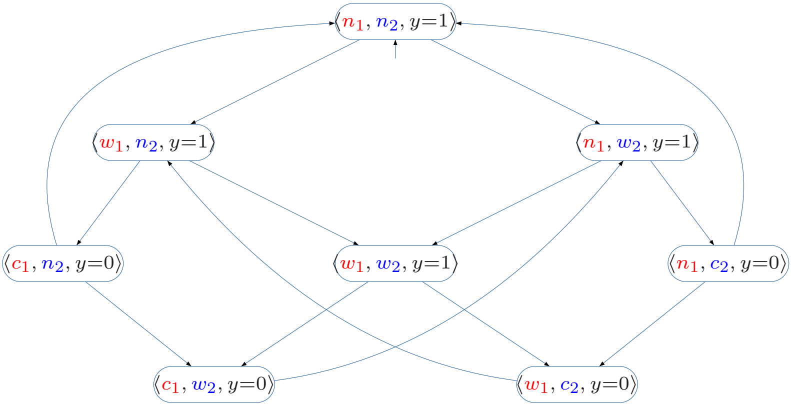
$$C = \{ s' \} \quad \text{with} \quad s' \not\models \Psi$$

provides a counterexample of the form $s_0\, s_1 \ldots s_n$ with $s_n \models \neg\Phi \wedge \neg\Psi$

Example : Semaphore-based mutual exclusion (1/3)



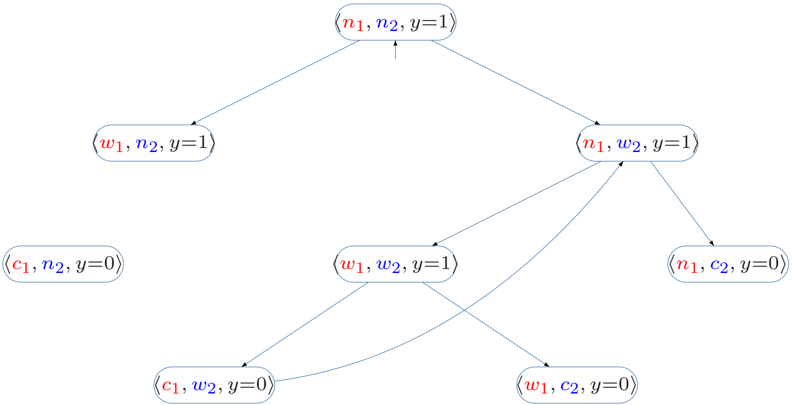
Example : Semaphore-based mutual exclusion (2/3)



$$\forall\Big(\underbrace{((n_1 \wedge n_2) \ \vee \ w_2)}_{\Phi} \ \mathsf{U} \ \underbrace{c_2}_{\Psi}\Big)$$

"Process P_2 gets access to the crit. sec. once it starts waiting to enter it."

Example : Semaphore-based mutual exclusion (3/3)



Counter examples for $\Box\Phi$

- Counterexample is initial path fragment $s_0\, s_1 \ldots s_n$ such that:
  - $s_0, \ldots, s_{n-1} \models \Phi$ and $s_n \not\models \Phi$
- Algorithm: backward search starting in $\neg\Phi$-states
- A witness of $\varphi = \Box\Phi$ consists of an initial path fragment of the form:
  - $\underbrace{s_0\, s_1 \ldots s_n\, s'_1 \ldots s'_r}_{\text{satisfy } \Phi}$ with $s_n = s'_r$
- Algorithm: cycle search in the digraph $G = (S, E)$ where the set of edges E:
  - $E = \{ (s, s') \mid s' \in \mathit{Post}(s) \wedge s \models \Phi \}$

Time Complexity

Theorem

Let TS be a transition system with N states and K transitions and φ be a CTL-path formula.
If $TS \not\models \forall\varphi$ then a counterexample for φ in TS can be determined in time $O(N + K)$.
The same holds for a witness for φ, provided that $TS \models \exists\varphi$.

Excercise

Justify the claim of the theorem above.

Outline

Bisimulation (Motivations)

- A *binary relation* on transition systems
  - when does a transition systems correctly implements another?
- Important for system *synthesis*
  - stepwise *refinement* of a system specification TS into an "implementation" TS'
- Important for system *analysis*
  - use the implementation relation as a means for *abstraction*
  - replace $TS \models \varphi$ by $TS' \models \varphi$ where $|TS'| \ll |TS|$ such that:

$$TS \models \varphi \text{ iff } TS' \models \varphi \quad \text{or} \quad TS' \models \varphi \Rightarrow TS \models \varphi$$

⇒ Focus on state-based *bisimulation* and *simulation*

- definition: what is bisimulation?
- logical characterization: which logical formulas are preserved by bisimulation?

Bisimulation Equivalence

Let $\mathit{TS}_i = (S_i, \mathit{Act}_i, \rightarrow_i, I_i, \mathit{AP}, L_i)$, $i{=}1, 2$, be transition systems

A *bisimulation* for $(\mathit{TS}_1, \mathit{TS}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that:

1. $\forall s_1 \in I_1 \, \exists s_2 \in I_2 . \, (s_1, s_2) \in \mathcal{R}$ and $\forall s_2 \in I_2 \, \exists s_1 \in I_1 . \, (s_1, s_2) \in \mathcal{R}$
2. for all states $s_1 \in S_1$, $s_2 \in S_2$ with $(s_1, s_2) \in \mathcal{R}$ it holds:
   (a) $L_1(s_1) = L_2(s_2)$
   (b) if $s_1' \in \mathit{Post}(s_1)$ then there exists $s_2' \in \mathit{Post}(s_2)$ with $(s_1', s_2') \in \mathcal{R}$
   (c) if $s_2' \in \mathit{Post}(s_2)$ then there exists $s_1' \in \mathit{Post}(s_1)$ with $(s_1', s_2') \in \mathcal{R}$

TS_1 and TS_2 are bisimilar, denoted $\mathit{TS}_1 \sim \mathit{TS}_2$, if there exists a bisimulation for $(\mathit{TS}_1, \mathit{TS}_2)$

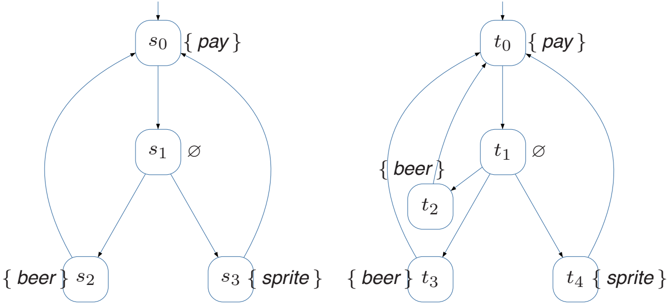
Bisimulation Equivalence

s_1	$\rightarrow$	s_1'		s_1	$\rightarrow$	s_1'
$\mathcal{R}$			can be completed to	$\mathcal{R}$		$\mathcal{R}$
s_2				s_2	$\rightarrow$	s_2'

and

s_1				s_1	$\rightarrow$	s_1'
$\mathcal{R}$			can be completed to	$\mathcal{R}$		$\mathcal{R}$
s_2	$\rightarrow$	s_2'		s_2	$\rightarrow$	s_2'

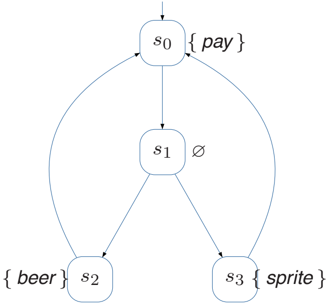
Bisimulation Example (1)

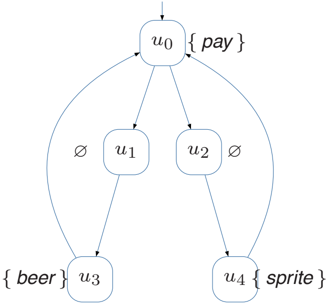


$$\mathcal{R} = \Big\{ (s_0, t_0), (s_1, t_1), (s_2, t_2), (s_2, t_3), (s_3, t_4) \Big\}$$

is a bisimulation for (TS_1, TS_2) where $AP = \{ \textit{pay}, \textit{beer}, \textit{sprite} \}$

Bisimulation Example (2)





$TS_1 \not\sim TS_3$ for $AP = \{ pay, beer, sprite \}$

But: $\{ (s_0, u_0), (s_1, u_1), (s_1, u_2), (s_2, u_3), (s_2, u_4), (s_3, u_3), (s_3, u_4) \}$

is a bisimulation for (TS_1, TS_3) for $AP = \{ pay, drink \}$

$\sim$ is an equivalence

Proposition

For any transition systems TS, TS_1, TS_2 and TS_3 over AP:

$TS \sim TS$ (reflexivity)

$TS_1 \sim TS_2$ implies $TS_2 \sim TS_1$ (symmetry)

$TS_1 \sim TS_2$ and $TS_2 \sim TS_3$ implies $TS_1 \sim TS_3$ (transitivity)

Proof as an exercise

Write here your proof of Slide 83

Bisimulation on Paths

Proposition

Whenever we have:

$$\begin{array}{ccccccccc} s_0 & \rightarrow & s_1 & \rightarrow & s_2 & \rightarrow & s_3 & \rightarrow & s_4 \ldots\ldots \\ \mathcal{R} & & & & & & & & \\ t_0 & & & & & & & & \end{array}$$

this can be completed to

$$\begin{array}{ccccccccc} s_0 & \rightarrow & s_1 & \rightarrow & s_2 & \rightarrow & s_3 & \rightarrow & s_4 \ldots\ldots \\ \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} \\ t_0 & \rightarrow & t_1 & \rightarrow & t_2 & \rightarrow & t_3 & \rightarrow & t_4 \ldots\ldots \end{array}$$

proof: by induction on index i of state s_i

Write here your proof of Slide 85

Bisimulation vs. Trace Equivalence

- Recall: In transition system TS consider paths $\pi = s_0 s_1 s_2 \ldots$.
  Get the trace of π as $trace(\pi) = L(s_0)L(s_1)L(s_2)\ldots \in (2^{AP})^\omega$

  Define Trace(TS) as the set of traces of initial maximal paths

- TS_1 and TS_2 are trace equivalent whenever Trace(TS_1) = Trace(TS_2)

Corollary of Proposition Slide 85

$TS_1 \sim TS_2$ implies Trace(TS_1) = Trace(TS_2)

Make clear your proof of Corollary of Slide 87

Bisimulation On States

$\mathcal{R} \subseteq S \times S$ is a *bisimulation* on *TS* if for any $(s_1, s_2) \in \mathcal{R}$:

- $L(s_1) = L(s_2)$
- if $s'_1 \in \mathit{Post}(s_1)$ then there exists an $s'_2 \in \mathit{Post}(s_2)$ with $(s'_1, s'_2) \in \mathcal{R}$
- if $s'_2 \in \mathit{Post}(s_2)$ then there exists an $s'_1 \in \mathit{Post}(s_1)$ with $(s'_1, s'_2) \in \mathcal{R}$

s_1 and s_2 are *bisimilar*, $s_1 \sim_{TS} s_2$, if $(s_1, s_2) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$ for *TS*

$s_1 \sim_{TS} s_2$ if and only if $TS_{s_1} \sim TS_{s_2}$

Coarsest Bisimulation

Lemma

For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ *it holds that:*

1. $\sim_{TS}$ *is an equivalence relation on* S.
2. $\sim_{TS}$ *is a bisimulation for TS.*
3. $\sim_{TS}$ *is the coarsest bisimulation for TS.*

Write here your proof of Slide 90

Quotient Transition System

For $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ and bisimulation $\sim_{\mathit{TS}} \subseteq S \times S$ on TS let

$\mathit{TS}/\sim_{\mathit{TS}} = (S', \{\tau\}, \rightarrow', I', \mathit{AP}, L')$, the *quotient* of TS under $\sim_{\mathit{TS}}$

where

- $S' = S/\sim_{\mathit{TS}} = \{ [s]_\sim \mid s \in S \}$ with $[s]_\sim = \{ s' \in S \mid s \sim_{\mathit{TS}} s' \}$
- $\rightarrow'$ is defined by: $$\frac{s \xrightarrow{\alpha} s'}{[s]_\sim \xrightarrow{\tau}' [s']_\sim}$$
- $I' = \{ [s]_\sim \mid s \in I \}$
- $L'([s]_\sim) = L(s)$

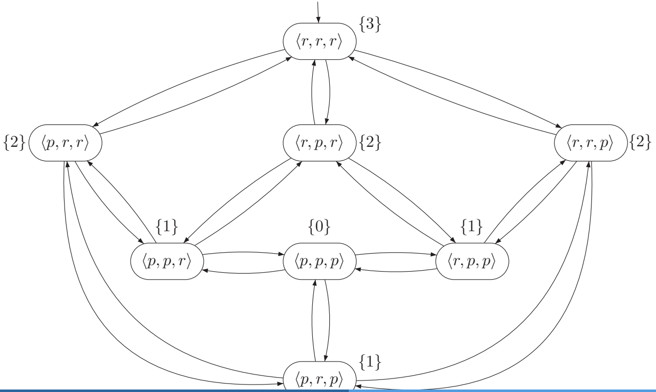
note that $\mathit{TS} \sim \mathit{TS}/\sim_{\mathit{TS}}$ Why?

Example: n printers (1/2)

Consider a system of n printers, each represented as extremely simplified by two states, $ready$ (initial) and $print$, and when started alternate between the states. The entire system is

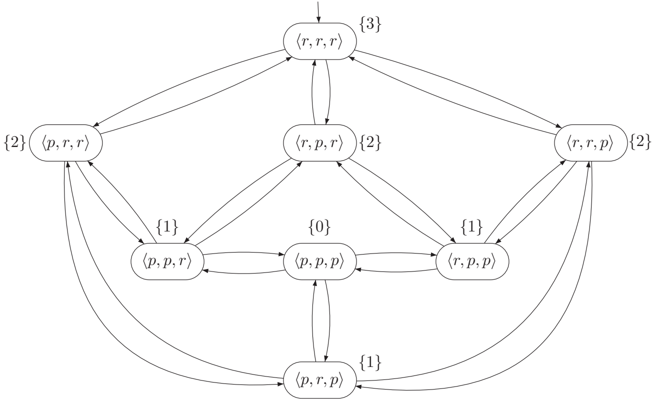
$$TS_n = Printer \mid\mid \ldots \mid\mid Printer \ (n \text{ times})$$

labeled over $AP = \{0, 1, \ldots, n\}$. $L(s) = k$ whenever k printers are ready. Here for $n = 3$:

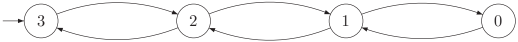


Example: n printers (2/2)

TS_n has 2^n states



but $TS_n/\sim$ has only $n + 1$ states.



Equivalence induced by the logic CTL

Definition

States s_1 and s_2 in TS (over AP) are CTL-equivalent, written $s_1 \equiv_{\mathsf{CTL}} s_2$ if, and only if, $(s_1 \models \Phi \text{ iff } s_2 \models \Phi)$, for all CTL state formulas over AP.

Let $TS_1 \equiv_{\mathsf{CTL}} TS_2$ if and only if $(TS_1 \models \Phi \text{ iff } TS_2 \models \Phi)$

Bisimulation vs. CTL-equivalence

Theorem

Let TS be a finite transition system and s_1, s_2 be two states.

$$s_1 \sim_{TS} s_2 \text{ if, and only if, } s_1 \equiv_{\mathsf{CTL}} s_2$$

Important remark

Theorem above also holds for any sublogic of CTL containing $\neg$, $\wedge$, and $\bigcirc$

Proof of $\equiv_{\mathsf{CTL}} \subseteq \sim_{TS}$ for Theorem on Slide 96 (1/2)

It suffices to show that $\mathcal{R} := \{(s_1, s_2) \in S \times S \mid s_1 \equiv_{\mathsf{CTL}} s_2\}$ is a bisimulation.

Proof of $\equiv_{\mathsf{CTL}} \subseteq \sim_{TS}$ for Theorem on Slide 96 (2/2)

Important Remark

In the proof of Slide 98, only operators $\neg$, $\wedge$, and $\bigcirc$ have been used. Thus, we do not need the full power of CTL to distinguish non-bisimilar states.
In fact, finiteness of TS is not necesseary, we can prove that:

Theorem

Hennessy and Milner 1985 [HM85]
$\equiv_{\mathsf{ML}} \subseteq \sim_{TS}$, for any finitely branching transition system TS, where ML is "Modal Logic", *i.e.*

$$\Phi \ni \mathsf{ML} ::= a|\neg\Phi|\Phi_1 \wedge \Phi_2|\exists \bigcirc \Phi$$

Write here the proof of Slide 99

$\equiv_{\mathsf{ML}} \subseteq \sim_{TS}$ does not hold for infinite transition systems (1/2)

We first consider the "cheating" case with an infinite set AP.
Define the transition system TS with:

- states: $\{s_1, s_2\} \cup \{t_A \mid A \subseteq AP\}$
- transitions:
  - Post$(s_1) = \{t_A \mid A \subset AP\}$
  - Post$(s_2) = \{t_A \mid A \subseteq AP\}$
  - Post$(t_A) = \{s_1\}$
- labelling: $L(s_1) = L(s_2) = \emptyset$, and $L(t_A) = A$ for all $A \subseteq AP$.

Exercice

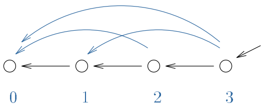
Draw this TS here:

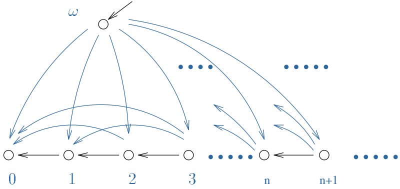
Clearly $s_1 \not\sim_{TS} s_2$, because of the transition $s_2 \rightarrow t_{AP}$.
Now, whichever formula is taken (even a CTL one), there are only finitely propositions of AP used in this formula, which prevents it from distinguishing s_1 and s_2.

$\equiv_{\mathsf{ML}} \subseteq \sim_{TS}$ does not hold for infinite transition systems (2/2)

Still, if AP has to be finite, we shall use ordinal processes from Klop.

- For each ordinal λ (see [Ros82] for linear orderings), define the transtion system $TS_\lambda = (\lambda + 1, <, \lambda)$: for all $\alpha, \beta \leq \lambda$, we have $\alpha \to \beta$ whenever $\alpha < \beta$.
- TS_3 TS_ω





- By [Klo88] $TS_\alpha \sim TS_\beta$ implies $\alpha = \beta$,
- But $\alpha \equiv_{CTL} \beta$ whenever $\alpha, \beta \geq \omega$ [Pin91]

Proof of $\sim_{TS} \subseteq \equiv_{\text{CTL}}$

Theorem

Let TS be a transition system (over AP), s_1 and s_2 be states of TS.

If $s_1 \sim_{TS} s_2$ then for every CTL formula $\Phi : s_1 \models \Phi$ iff $s_2 \models \Phi$

Proof sketch: Establish (a) and (b) by induction on the structure of the formulas of CTL. See the [PoM] with a simultaneaous induction on state and path formulas of the logic CTL* ($\supset$ CTL and that we shall see later in this course).

Consequences of the theorem:

- Bisimilar transition systems preserve the same CTL formulas: $TS_1 \models \Phi$ and $TS_2 \not\models \Phi$ implies $TS_1 \not\sim TS_2$
- Non-bisimilarity can be shown by a single CTL formula: $TS_1 \not\sim TS_2$ implies there exists $\Phi \in$ CTL s.t. $TS_1 \models \Phi$ and $TS_2 \not\models \Phi$
- Actually, you even do not need to use an until-operator!
- To check $TS \models \Phi$, it suffices to check on the quotient: $TS/\sim \models \Phi$

C. Baier and J.P. Katoen.
Principles of model checking, volume 26202649.
MIT Press, 2008.

Edmund M Clarke and E Allen Emerson.
Synthesis of synchronization skeletons for branching time temporal logic.
In *Logic of programs: Workshop*, volume 131, pages 52–71, 1981.

Edmund M Clarke and E Allen Emerson.
Design and synthesis of synchronization skeletons using branching time temporal logic.
Springer, 1982.

D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi.
On the temporal analysis of fairness.
In *Proc. 7th ACM Symp. Principles of Programming Languages, Las Vegas, Nevada*, pages 163–173, January 1980.

M. Hennessy and R. Milner.
Algebraic laws for nondeterminism and concurrency.
Journal of the ACM, 32(1):137–161, January 1985.

J. W. Klop.
Bisimulation Semantics.

Lectures given at the REX School/Workshop, Noordwijkerhout, NL, May 1988.

S. Pinchinat.
Ordinal processes in comparative concurrency semantics.
In *Proc. 5th Workshop on Computer Science Logic, Bern, LNCS 626*, pages 293–305. Springer-Verlag, October 1991.

J. G. Rosenstein.
Linear Orderings.
Academic Press, 1982.

T.A. Sudkamp and A. Cotterman.
Languages and machines: an introduction to the theory of computer science, volume 2.
Addison-Wesley, 2006.