☉_☉ *Find any errors? Please send them back, I want to keep them!*

1 Preliminaries

Definition

Some things probably important:

Transformational Systems

Transform set of input data into output data: $S_i \rightarrow S_k$. E.g. Compilers, database processing.
Correctness criteria: Termination, Correctness of $S_i \rightarrow S_k$

Reactive Systems

Ongoing interaction with environment, driven by events/stimuli. E.g. Operating Systems, Control Systems.
Correctness: non-termination, correctness of stimuli-response pair.

Embedded Systems

Usually reactive systems, tightly connected to the hardware they control.

Cyber-Physical systems

Integration of computation and physical processes, often networked, e.g. sensor-/actuator systems, automotive control systems.

Real-Time Systems

Correctness depends on time bounds:

soft: violating soft time bounds will decrease quality of system.

hard: violating hard time bounds will make the system fail.

Hybrid systems

Systems characterized by discrete and continuous variables. E.g. thermostat, ...

Fault

Mistake made by a human during software development/production.

Failure

Behaviour of a system deviating from its specified behaviour. This is most often the result of a fault being executed.

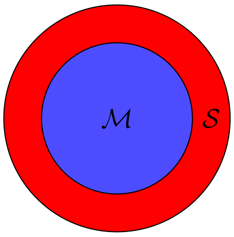
Safety-Critical Systems

When a safety-critical system fails, people, the environment or damage to property or assets may occur.

1.1 System Correctness

When is a system correct?

1. It does what we expect it to do. ⇒functional model checking
2. It does so in a timely manner. ⇒real time or probabilistic model checking.
3. It does so with a certain probability over a certain period of time. ⇒probabilistic model checking.



- Given a model and a specification:
  Does $M \models S$?
- When every behaviour of M is also behaviour of S this is the case. ⇒Model does not reveal properties violating the specification.
- Model of course has to represent the behaviour of the system.

2 CTL and CTL Model Checking

State

Characterizes the salient features of a system at a given point of observation. ⇒A state can be observed as long as the features of interest don't change.

2.1 State-Based Modelling

State Transition in Discrete Systems

Instantaneous change of observed features of systems. Represents computation step.

real-time models

time passes in a state & state must be left when time-bound is reached

stochastic systems

state transitions are labeled with probabilities

hybrid systems

- continuous state variables change in a state
- discrete state variables change during state transition

State transitions:

- In a given state a certain number of events are possible (Leading to several different successor states).
- Represent valid sequence of computations.
- They encode history information (a state can only be reached trough a series of transitions).

Guidelines:

Abstraction: Focus only on important facts, disregard the rest.

Simplicity: Find simplest abstraction that still reveals phenomena of interest.

Characterization of reactive systems:

- State of the system.
- State transitions (caused by events/stimuli).
- Reactions triggered by transitions

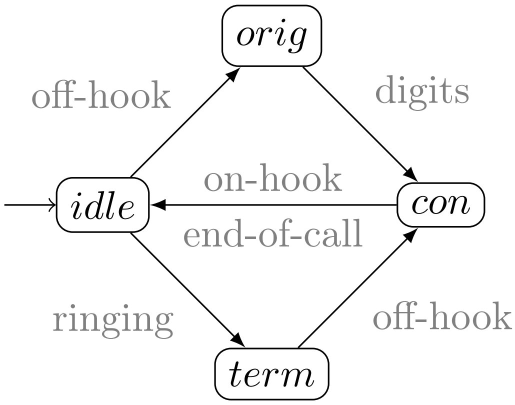
2.2 Transition Systems

Transition System

A **Transition System** TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where:

S	set of states
Act	set of actions
$\rightarrow \subseteq S \times Act \times S$	transition relation
$I \subseteq S$	set of initial states
AP	atomic propositions
$L : S \rightarrow 2^{AP}$	labeling function

- $(s, \alpha . s')$ can be written as $s \rightarrow^{\alpha} s'$ or $s \xrightarrow{\alpha} s'$.



Atomic Propositions:

- logical representation of facts that may hold in a given state.
- AP set of all atomic propositions used in the system model.

Labeling functions:

- which atomic propositions actually hold in a given state.

Predecessors and Successors

$$Post(s,\alpha) = \left\{ s' \in S | s \xrightarrow{\alpha} s' \right\} \qquad Post(s) = \bigcup_{\alpha \in Act} Post(s\alpha)$$

$$Pre(s,\alpha) = \left\{ s' \in S | s' \xrightarrow{\alpha} s \right\} \qquad Pre(s) = \bigcup_{\alpha \in Act} Pre(s\alpha)$$

$$Post(C,\alpha) = \bigcup_{s \in C} Post(s,\alpha) \qquad Post(C) = \bigcup_{s \in C} Post(s) \text{ for } C \subseteq S$$

$$Pre(C,\alpha) = \bigcup_{s \in C} Pre(s,\alpha) \qquad Pre(C) = \bigcup_{s \in C} Pre(s) \text{ for } C \subseteq S$$

Terminal/Final State

a state for which $Post(s) = \varnothing$

Action-Determinism

A TS is **action-deterministic**, iff for all s, α

- $|I| \leq 1$
- $|Post(s, \alpha)| \leq 1$

otherwise it is **action-nondeterministic**. In other words: For every state s and every action α there is at most one outgoing transition labeled with α.

$AP - Determinism$

A TS is **AP-deterministic**, iff for all $s, A \in 2^{AP}$

- $|I| \leq 1$
- $|Post(s) \cap \{s' \in S | L(s') = A\}| \leq 1$

where $|Post(s) \cap \{s' \in S | L(s') = A\}$ denotes the set of all equally labeled successors of s. In other words: For every state s, every successor state has a unique AP labeling.

Nondeterminism can be used to implement abstraction and concurrency.

2.3 System Executions

Finite Execution Fragment

A **finite execution fragment** ϱ of TS is an alternating sequence of states and executions ending with a state:

$$\varrho = s_0 \alpha_1 s_2 \alpha_2 \ldots \alpha_n s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i < n$$

infinite execution fragment

An **infinite execution fragment** ϱ of TS is an alternating sequence of states and executions ending with a state:

$$\varrho = s_0 \alpha_1 s_2 \alpha_2 \ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i$$

maximal execution fragment

An execution fragment, that is

either finite and ending in terminal state

or infinite.

initial execution fragment

An execution fragment is initial, iff $s_0 \in I$.

Execution

A initial, maximal execution fragment.

Reachability

State $s \in S$ is called **reachable** in a TS, if there exists an initial, finite execution fragment $s_0\alpha_1 s_1 \alpha_2 \ldots \alpha_n s_n$ such that $s_n = s$.
$Reach(TS)$ denotes the set of all reachable states in TS.

State Graph

The **State Graph** of TS, $G(TS)$, is the directed Graph (V, E) with vertices $V = S$ and edges $E = \{s, s'\} \in S \times S | s' \in Post(s)\}$.

Transitive Post Hull

$$Post^*(s) \text{ is the set of states reachable from } s \qquad Post^*(C) = \bigcup_{s \in C} Post^*(s), \text{ for } C \subseteq S$$

$$Pre^*(s) \text{ is the set of states from which } s \text{ is reachable} \qquad Pre^*(C) = \bigcup_{s \in C} Pre^*(s), \text{ for } C \subseteq S$$

$$Reach(TS) = Post^*(I)$$

Path fragments

A **path fragment** is an exeuction fragment without actions.

Finite Path fragments

A **Finite path fragment** $\hat{\pi}$ of TS is a state sequence:

$$\hat{\pi} = s_0 s_1 \ldots s_n \text{ such that } s_{i+1} \in Post(s_i) \text{ for all } 0 \leq i \leq n \text{ where } n \geq 0$$

Infinite Path fragments

An **Infinite path fragment** $\hat{\pi}$ of TS is a infite state sequence:

$$\hat{\pi} = s_0 s_1 \ldots \text{ such that } s_{i+1} \in Post(s_i) \text{ for all } i \geq 0$$

Path

A **Path** is a maximal, initial path fragment

Trace

When only registering the atomic propositions along execution, this is called a **trace**.

$$\hat{\pi} = s_0 s_1 \ldots, s_n \qquad trace(\hat{\pi}) = L(s_0)L(s_1)\ldots L(s_n)$$

$$Traces(s) = trace(Paths(s)) \qquad Traces(TS) = \bigcup_{s \in I} Traces(s)$$

$$Traces_{fin} = trace(Paths_{fin}(s)) \qquad Traces_{fin}(s)(TS) = \bigcup_{s \in I} Traces_{fin}(s)$$

2.4 Structural Operational Semsantics

Semantics of a program in terms of computation steps defined by transition system:

$$\frac{premise}{conclusion}$$

If the premise holds, the conclusion holds (and can be used to trigger a new inference rule). Can be recursively applied ⇒structural inductive creation.

Interleaving

$$TS_1 ||| TS_2 = (S_1 \times S_2, Act_1 \uplus Act_2, \longrightarrow, I_1 \times I_2, AP_1 \uplus AP_2, L)$$

where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\longrightarrow$ is defined by:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \text{ and } \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

A Computation tree is obtained from transition system by unfolding operation:

- s_k is a successor node of s_i in the computation tree, iff there is a transition from s_i to s_k in the transition system.

2.5 Property Specification

based on modal logic:

Lp it is **necessary** that p

Mp it is **possible** that p

$\neg Lp$ it is **not necessary** that p

$\neg Mp$ it is **not possible** that p

Kripke-Structure

Let

- $M = W, V, A$ be a Kripke-Structure:
- Π a set of atomic propositions and $p \in \Pi$
- $w, v \in W$
- Φ, ρ formulae

then we define the relation $\models$ (satisfaction relation) for M:

$$\begin{array}{lcl}
(M,w) \models p & \Longleftrightarrow & A(w,p) = true \\
(M,w) \models \neg p & \Longleftrightarrow & A(w,p) = false \\
(M,w) \models \Phi \wedge \rho & \Longleftrightarrow & (M,w) \models \Phi \text{ and } (M,w) \models \rho \\
(M,w) \models L\Phi & \Longleftrightarrow & (\forall v : (w,v) \in V)((M,v) \models \Phi) \\
(M,w) \models M\Phi & \Longleftrightarrow & (\exists v : (w,v) \in V)((M,v) \models \Phi)
\end{array}$$

Further syntactic definitions:

$$\begin{array}{lcll}
\Phi \vee \rho & \cong & \neg(\neg\Phi \wedge \neg\rho) & \\
\Phi \supset \rho & \cong & \neg\Phi \vee \rho & \text{implies} \\
\Phi \equiv \rho & \cong & (\Phi \supset \rho) \wedge (\rho \supset \Phi) & \\
M\Phi & \cong & \neg L \neg\Phi &
\end{array}$$

In temporal logic:

- Mp corresponds to $\Box p$
- Lp corresponds to $\diamond p$

Computation Tree Logic Syntax (CTL Syntax)

$a \in AP$
CTL state formula Φ:

$$\begin{array}{rr}
true & \neg\Phi \\
a & \exists\varphi \\
\Phi_1 \wedge \Phi_2 & \forall\varphi
\end{array}$$

CTL path formula

$$\circ\Phi \qquad \Phi_1 \,\mathcal{U}\, \Phi_2$$

To be syntactically correct, temporal operators and path quantifiers alternate.
Derived Operators:

$$\begin{array}{rl}
\textit{potentially } \Phi : & \exists \diamond \Phi = \exists(true\,\mathcal{U}\,\Phi) \\
\textit{inevitably } \Phi : & \forall \diamond \Phi = \forall(true\,\mathcal{U}\,\Phi) \\
\textit{potentially always}\Phi : & \exists\Box\Phi = \neg\forall \diamond \neg\Phi \\
\textit{invariantly } \Phi : & \forall\Box\Phi = \neg\exists \diamond \neg\Phi \\
\textit{weak until } \Phi : & \exists(\Phi\,\mathcal{W}\,\Psi) = \neg\forall((\Phi \wedge \neg\Psi)\,\mathcal{U}\,(\neg\Phi \wedge \neg\Psi)) \\
 & \forall(\Phi\,\mathcal{W}\,\Psi) = \neg\exists((\Phi \wedge \neg\Psi)\,\mathcal{U}\,(\neg\Phi \wedge \neg\Psi))
\end{array}$$

Computation Tree Logic Semantic (CTL Semantic)

CTL state formulae:

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

Satisfaction Set

The satisfaction set $Sat(\Phi)$ for a CTL formula is defined by:

$$Sat(\Phi) = \{s \in S | s \models \Phi\}$$

A TS satisfies a CTL formula Φ if Φ holds in all initial states:

$$TS \models \Phi \iff \forall s_0 \in I : s_0 \models \Phi$$

CTL Equivalence

CTL formulas Φ and Ψ are **equivalent**, $\Phi \equiv \Psi$ iff $Sat(\Phi) = Sat(\Psi)$.

$$\Phi \equiv \Psi \iff (TS \models \Phi \iff TS \models \Psi)$$

Equivalence-based Rewrite Rules

Duality Laws:

$$\begin{aligned} \forall \circ \Phi &\equiv \neg\exists \circ \neg\Phi \\ \exists \circ \Phi &\equiv \neg\forall \circ \neg\Phi \\ \forall \diamond \Phi &\equiv \neg\exists\Box\neg\Phi \\ \exists \diamond \Phi &\equiv \neg\forall\Box\neg\Phi \\ \forall(\Phi \,\mathcal{U}\, \Psi) &\equiv \neg\exists((\Phi \wedge \neg\Psi) \,\mathcal{W}\, (\neg\Phi \wedge \neg\Psi)) \end{aligned}$$

Expansion Laws:

$$\begin{aligned} \forall(\Phi \,\mathcal{U}\, \Psi) &\equiv \Psi \vee (\Phi \wedge \forall \circ \forall(\Phi \,\mathcal{U}\, \Psi)) \\ \forall \diamond \Phi &\equiv \Phi \vee \forall \circ \forall \diamond \Phi \\ \forall\Box\Phi &\equiv \Phi \wedge \forall \circ \forall\Box\Phi \\ \exists(\Phi \,\mathcal{U}\, \Psi) &\equiv \Psi \vee (\Phi \wedge \exists \circ \exists(\Phi \,\mathcal{U}\, \Psi)) \\ \exists \diamond \Phi &\equiv \Phi \vee \exists \circ \exists \diamond \Phi \\ \exists\Box\Phi &\equiv \Phi \wedge \exists \circ \exists\Box\Phi \end{aligned}$$

Distributive Laws:

$$\begin{aligned} \forall\Box(\Phi \wedge \Psi) &\equiv \forall\Box\Phi \wedge \forall\Box\Psi \\ \exists \diamond (\Phi \wedge \Psi) &\equiv \exists \diamond \Phi \wedge \exists \diamond \Psi \end{aligned}$$

But:

$$\exists\Box(\Phi \wedge \Psi) \not\equiv \exists\Box\Phi \wedge \exists\Box\Psi$$
$$\forall\Diamond(\Phi \wedge \Psi) \not\equiv \forall\Diamond\Phi \wedge \forall\Diamond\Psi$$

2.5.1 LTL

Missing Content

Definition and Explanation of LTL, see Model Checking aggregation.

CTL and LTL equivalence

CTL formula Φ and LTL formula ψ are **equivalent**, $\Phi \equiv \psi$, iff for any transition system TS over AP

$$TS \models \Phi \iff TS \models \psi$$

There can only be equivalent Φ and ψ if by omitting all path quantifiers from Φ yields

$$\Phi \equiv \psi$$

otherwise there does not exist an equivalent LTL formula.
⇒LTL and CTL have incomparable expressiveness.

2.5.2 CTL*

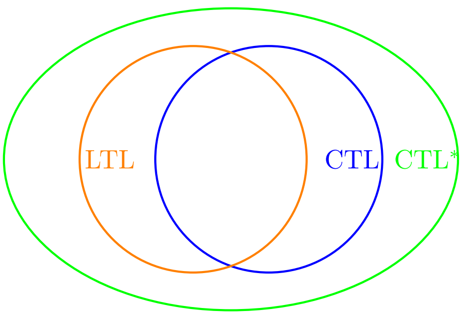
CTL*

State formula: $\Phi \;:=\; true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi$
path formula: $\varphi \;:=\; \Phi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \;\; \circ\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$

CTL* Semantics

$$
\begin{array}{lcl}
s \models a & \iff & a \in L(s) \\
s \models \neg\Phi & \iff & not\ s \models \Phi \\
s \models \Phi \wedge \Psi & \iff & (s \models \Phi) \text{ and } (s \models \Psi) \\
s \models \exists\varphi & \iff & \pi \models \varphi \text{ for some } \pi \in Paths(s) \\
\pi \models \Phi & \iff & \pi[0] \models \Phi \\
\pi \models \varphi_1 \wedge \varphi_2 & \iff & \pi \models \varphi_1 \text{ and } \pi \models \varphi_2 \\
\pi \models \neg\varphi & \iff & \pi \not\models \varphi \\
\pi \models \circ\varphi & \iff & \pi[1..] \models \varphi \\
\pi \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \iff & \exists j \geq 0.(\pi[j..] \models \varphi_2 \wedge (\forall 0 \leq k < j.\pi[k..] \models \varphi_1))
\end{array}
$$

Satisfaction set and TS satisfaction is same as for CTL.



2.5.3 CTL Model Checking Procedure

1. convert CTL formula Φ' into an equivalent CTL formula Φ in **Existential Normal Form (ENF)**
2. recursively compute the set $Sat(\Phi) = \{s \in S | s \models \Phi\}$
3. $TS \models \Phi$ iff **each initial sate** of TS belongs to $Sat(\Phi)$

ENF Conversion

ENF Subset of CTL:

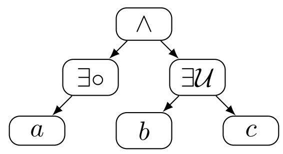
$$\Phi \ := \ true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists \circ \Phi \mid \exists(\Phi_1 \,\mathcal{U}\, \Phi_2 \mid \exists\Box\Phi)$$

Conversion Rules:

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

Computation of $Sat(\Phi)$

1. create **parse tree** from formula
2. compute $Sat(a_i)$ for leaf nodes
3. move up in the parse tree by level, computing $Sat(.)$ from child nodes
4. when all root tree is computed, check if $I \in Sat(\Phi)$



Computation of $Sat(a_i)$

$$\begin{aligned}
Sat(true) &= S \\
Sat(a) &= \{s \in S | a \in L(s)\} \text{ for any } a \in AP \\
Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi) \\
Sat(\exists \circ \Phi) &= \{s \in S | Post(s) \cap Sat(\Phi) \neq \varnothing\} \\
Sat(\exists(\Phi \,\mathcal{U}\, \Psi)) &\text{ is the smallest subset } T \text{ of } S\text{, such that} \\
&Sat(\Psi) \subseteq T \text{ and} \\
&(s \in Sat(\Phi) \text{ and } Post(s) \cap T \neq \varnothing) \Rightarrow s \in T \\
Sat(\exists\Box\Phi) &\text{ is the largest subset } T \text{ of } S\text{, such that} \\
&T \subseteq Sat(\Phi) \text{ and} \\
&s \in T \longrightarrow Post(s) \cap T \neq \varnothing
\end{aligned}$$

smallest Fixpoint calculation: $\exists(\Phi_1 \,\mathcal{U}\, \Phi_2)$

$T := Sat(\Phi_2)$;
while $\{s \in Sat(\Phi_1) \setminus T | Post(s) \cap T \neq \varnothing\} \neq \varnothing$ **do**
  $\{s \in Sat(\Phi_1) \setminus T | Post(s) \cap T \neq \varnothing\} \neq \varnothing$
  $T := T \cup \{s\}$
end

greatest Fixpoint calculation: $\exists\Box\Phi$

```
T := Sat(Φ)
while {s ∈ T | Post(s) ∩ T = ∅} ≠ ∅ do
    let {s ∈ T | Post(s) ∩ T = ∅}
    T := T \ {s};
end
```

Compute $Sat(\exists(\Phi\,\mathcal{U}\,\Psi))$ by **Enumerative Backward Search**

```
T := Sat(Ψ);
E := T
while E ≠ ∅ do
    s' ∈ E;
    E := E \ {s'};
    for  all s ∈ Pre(s') do
        if s ∈ Sat(Φ) \ T then
            E := E ∪ {s};
            T := T ∪ {s};
        end
    end
end
return T
```

Compute $Sat(\exists\Box\Phi)$ by **Enumerative Backwards Search**

```
E := S \ Sat(Φ) ;                        /* E contains any unvisited s' with s' ⊭ ∃□Φ */
T := Sat(Φ) ;               /* T contains any s for which s ⊨ ∃□Φ is not disproven */
for all s ∈ Sat(Φ) do
    c[s] := |Post(s)|;
end
while E ≠ ∅ do
    s' ∈ E;
    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}
            end
        end
    end
end
return T
```

Alternative Algorithm for $Sat(\exists\Box\Phi)$:

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

   $\Rightarrow$ all removed states do not satisfy $\exists\Box\Phi$ and can therefore be removed
2. Dtermine all **non-trivial strongly connected components** in $TS[\Phi]$
   - non-trivial SCC $\Rightarrow$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"
   - simple reachability search (backward manner)

2.5.4 Time Complexity

The CTL Model Checking Problem $TS \models \Phi$ can be determined in $\mathcal{O}(|\Phi| \cdot (N + M)$, where N is the number of states and M the number of transitions, $N + M$ is the size of the transition system, which can be exponentially large.
LTL Model Checking can be done in $\mathcal{O}((N + M) \cdot 2^{|\Phi|}$. But LTL formulae can be exponentially shorter.

2.5.5 Counterexamples

Missing Content

Counterexamples in LTL, see Model Checking aggregation.

Counterexample and Witnesses

- counterexample: path fragment $s \to s'$ such that
  - $s \in I$ and $s' \in Post(s)$ with $s' \not\models \Phi$
- witness: a path fragment $s \to s'$ such that
  - $s \in I$ and $s' \in Post(s)$ with $s' \models \Phi$
- algorithmic computation: Inspection of direct successors of initial states.

Witness for $\Phi \mathcal{U} \Psi$

backwards search starting in $Sat(\Psi)$

Counterexample for $\Phi \mathcal{U} \Psi$

has one of the forms:

- $\underbrace{s_0 \dots s_{n-1} \underbrace{s_n s'_1 \dots s'_r}_{cycle}}_{\text{satisfy } \Phi \wedge \neg\Psi}$ with $s_n = s'_r$ (would work for $\mathcal{W}$, but not $\mathcal{U}$)
- $s_0 \dots s_{n-1} s_n$ where $s_n \models \neg\Phi \wedge \neg\Psi$

Computing Counterexample:

- let $G = (S, E)$ a directed graph, where S is the set of states of the TS and $E = \{(s, s') \in S \times S | s' \in Post(s) \wedge s \models \Phi \wedge \neg\Psi\}$
- Each path in G starting in an $s_0 \in I$ leading to an trivial or non-trivial SCC yields a counterexample.

$\Rightarrow$ counterexample generation requires SCC computation (e.g. Tarjans Algorithm)

3 CTL* Model Checking

CTL* Model Checking

Follow same recursive pattern using parse tree as for CTL model checking.

- replace maximal proper state formula by new proposition a_Ψ
- Ψ is a **maximal proper state subformula** of Φ whenever Ψ is a subformula of Φ that differs from Φ and that is not contained in any other proper state formula of Φ.

- adjust labeling of TS such that $a_\Psi \in L(s)$ iff $s \in Sat(\Psi)$

$\Rightarrow$ LTL formula

$$s \models \exists\varphi \iff s \not\models_{CTL^*} \forall\neg\varphi \iff s \not\models_{LTL} \neg\varphi$$

Algorithm

for all $i \leq |\Phi|$ **do**
  for all $\Psi \in Sub(\Phi)$ **with** $|\Psi| = i$ **do**
    switch Ψ **do**

$$\begin{aligned} true: &\quad Sat(\Psi) := S \\ a: &\quad Sat(\Psi) := \{s \in S | a \in L(s)\}; \\ a_1 \wedge a_2: &\quad Sat(\Psi) := Sat(a_1) \cap Sat(a_2); \\ \neg a: &\quad Sat(\Psi) := S \setminus Sat(a); \\ \exists\varphi: &\quad determine\ Sat_{LTL}(\neg\varphi); \\ \Box\varphi: &\quad Sat(\Psi) := S \setminus Sat_{LTL}(\neg\varphi) \end{aligned}$$

    endsw
    $AP := AP \cup \{a_\Psi\}$; ; `/* introduce fresh atomic proposition */`
    replace Ψ with a_Ψ; **for all** $s \in Sat(\Psi)$ **do**
      $L(s) \cup \{a_\Psi\}$;
    end
  end
end
return $I \subseteq Sat(\Phi)$

3.1 Time Complexity

For transition systems with N states and M transitions the CTL* model checking problem $TS \models \Phi$ can be determined in $\mathcal{O}(N + M) \cdot 2^{|\Phi|}$

3.2 Fairness

Fairness

Fairness Constraints $\Rightarrow$rule out unrealistic executions by putting constraints on actions that occur along infinite executions

$$unconditional \Rightarrow strong \Rightarrow weak$$

weak rules out the least executions
Fairness Assumptions $\Rightarrow$distinct constraints on distinct action sets

Fairness Constraints

unconditional LTL fairness constraint: $u_{fair} = \Box\Diamond\Psi$

strong LTL fairness constraint: $s_{fair} = \Box\Diamond\Phi \longrightarrow \Box\Diamond\Psi$

weak LTL fairness constraint: $w_{fair} = \Diamond\Box\Phi \longrightarrow \Box\Diamond\Psi$

$$fair = u_{fair} \wedge s_{fair} \wedge w_{fair}$$

- strong and unconditional fairness $\Rightarrow$solve contentions
- weak fairness $\Rightarrow$resolve nondeterminism

$$FairPaths_{fair}(s) = \{\pi \in Paths(s) | \pi \models fair\}$$
$$FairTraces_{fair}(s) = \{trace(\pi) | \pi \in FairPaths_{fair}(s)\}$$
$$s \models_{fair} \varphi \text{ iff } \forall \pi \in FairPaths_{fair}(s).\pi \models \varphi$$
$$TS \models_{fair} \varphi \text{ iff } \forall s_0 \in I.s_0 \models_{fair} \varphi$$

For TS and LTL formula φ and LTL fairness assumption $fair$:

$$TS \models_{fair} \varphi \text{ iff } TS \models (fair \rightarrow \varphi)$$

Fairness in CTL

⇒ignore unfair paths

unconditional $u_{fair} = \bigwedge_{0<i\leq k} \Box \Diamond \Psi$

strong: $s_{fair} = \bigwedge_{0<i\leq k} (\Box \Diamond \Phi_i \rightarrow \Box \Diamond \Psi_i)$

weak $w_{fair} = \bigwedge_{0<i\leq k} (\Diamond \Box \Phi_i \rightarrow \Box \Diamond \Psi_i)$

A **CTL fairness** constraint is an **LTL** formula over **CTL formulas**

$$Sat_{fair}(\Phi) = \{s \in S | s \models_{fair} \Phi\}$$

For transition system TS without terminal states, a CTL formula Φ in ENF and CTL fairness assumption $fair$:

1. establish whether $TS \models_{air} \Phi$
2. use bottom-up CTL procedure to determine $Sat_{fair}(\Phi)$
   (a) replace CTL-state formulas in s_{fair} by atomic propositions

$$s_{fair} := \bigwedge_{0<i\leq k} (\Box \Diamond a_i \rightarrow \Box \Diamond b_i)$$

Fair CTL Model Checking

$s \models_{fair} \exists \bigcirc a$ iff $\exists s' \in Post(s)$ with $s' \models a$ and $\underbrace{FairPaths(s') \neq \varnothing}_{s' \models_{fair} \exists \Box true}$

$s \models fair \exists (a \,\mathcal{U}\, a')$ iff there exists a finite path fragment $s_0 s_1 \ldots s_{n-1} s_n \in Paths_{fin}(s)$ with $n \geq 0$ such that $s_i \models a$ for $0 \leq i < n, s_n \models a'$ and $\underbrace{FairPaths(s') \neq \varnothing}_{s' \models_{fair} \exists \Box true}$

Model Checking with fairness can be reduced to:

- Model Checking CTL
- computing $Sat_{fair}(\exists \Box a)$ for $a \in AP$

Algorithm:

compute $Sat_{fair}(\exists\Box true) = \{s \in S | FairPaths(s) \neq \varnothing\}$ **for all** $s \in Sat_{fair}(\exists\Box true)$ **do**
  $L(s) := L(s) \cup \{a_{fair}\}$; `/* compute` $Sat_{fair}(\Phi)$ `*/`
end
for all $0 < i \leq |\Phi|$ **do**
  for all $\Psi \in Sub(\Phi)$ **with** $|\Psi| = i$ **do**
    switch Ψ **do**

$$\begin{array}{rcl} true & : & Sat_{fair}(\Psi) := S; \\ a & : & Sat_{fair}(\Psi) := \{s \in S | a \in L(s)\}; \\ \neg a & : & Sat_{fair}(\Psi) := S \setminus Sat_{fair}(a); \\ a \wedge a' & : & Sat_{fair}(\Psi) := Sat_{fair}(a) \cap Sat_{fair}(a'); \\ \exists \circ a & : & Sat_{fair}(\Psi) := Sat(\exists \circ (a \wedge a_{fair})); \\ \exists(a \,\mathcal{U}\, a') & : & Sat_{fair}(\Psi) := Sat(\exists(a \,\mathcal{U}\, (a' \wedge a_{fair}))); \\ \exists\Box a & : & Sat_{fair}(\Psi) := compute\, Sat_{fair}(\exists\Box a); \end{array}$$

    endsw
    replace all occurrences of Ψ (in Φ) b< the fresh atomic proposition a_Ψ **for all** $s \in Sat_{fair}(\Psi)$ **do**
      $L(s) := L(s) \cup \{a_\Psi\}$
    end
  end
end
return $I \subseteq Sat_{fair}(\Phi)$

Computation of $Sat_{fair}(\exists\Box a)$:

- Consider state s only if $s \models a$, otherwise eliminate s
- $s \models_{fair} \exists\Box a$ iff there is a non-trivial SCC D in $TS[a]$ reachable from s:

$$D \cap Sat(a_i) = \varnothing \text{ or } D \cap Sat(b_i) \neq \varnothing \text{ for } 0 < i \leq k$$

- $Sat_{sfair}(\exists\Box a) = \{s \in S | Reach_{TS[a]}(s) \cap T \neq \varnothing\}$ where T is the union of all non-trivial SSCs C that contain D satisfying above equation.

Time Complexity

TS with N states and M transitions, CTL formula Φ, CTL fairness constraint $fair$ with k conjuncts: CTL model checking in $\mathcal{O}(|\Phi| \cdot (N + M) \cdot k)$

4 Real-Time Model Checking

soft real-time systems

Violating soft real-time bounds does **not** lead to invalidation of system. ⇒"quality of service" requirements. Usually with probabilities attached (reach state X with probility of Y in Z time).

hard real-time systems

Correctness of system depends on satisfying real-time constraints.

discrete time domain

- time advances in discrete steps

- actions only happen at natural time values $\Rightarrow$time domain $\mathbb{N}$

advantages
- conceptually simple
- no need to change TS
- take LTL or CTL
- use traditional model checking algorithms

disadvantages
- fixed minimal delay granularity, between two points not observable
- not invariant to changes in time scale
- for asynchronous systems determination of mimimal delay hard

- time domain is dense
- infinite branching of computation tree

Clocks

- value increases while in a state
- may only be reset to zero
- can be referenced in constraints
- clocks increase at same pace (with rate 1)

$\Rightarrow$ guards on edges

$\Rightarrow$ invariants on locations

Clock Constraints (CC)

$c \in \mathbb{N}$, $x \in C$, C set of clocks

$$g := x < c | x \leq c | x > c | x \geq c | g \wedge g$$

Clock constraints without any conjunctions are atomic: $ACC(C)$
$\Rightarrow$**rational** valued constraints can be translated into naturals by proper scaling.

Timed Automata

$TA = (Loc, Act, C, \hookrightarrow, Loc_0, Inv, AP, L)$, where:

$$\begin{aligned} Loc &\text{ is a finite set of locations} \\ Loc_0 &\text{ is a set of initial locations} \\ C &\text{ is a finite set of clocks} \\ \hookrightarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc &\text{ is a transition relation} \\ Inv : Loc \rightarrow CC(C) &\text{ is an invariant-assignement function} \\ L : Loc \rightarrow 2^{AP} &\text{ is a labeling function} \end{aligned}$$

Edge $\ell \xhookrightarrow{g:\alpha,C} \ell'$ means intuitively:

- action α is enabled once guard g holds
- when moving from ℓ to ℓ'

- perform action α
- reset any clock in C to zero
- all clocks not in C keep their value

- Nondeterminism if multiple transitions are enabled
- $Inv(\ell)$ constraints amount of time that may be spent in location ℓ
  - once it becomes invalid, ℓ must be left
  - if leaving is not possible, deadlock

Composition

$TA_i = (Loc_i, Act_i, C_i, \hookrightarrow_i, Loc_{0,i}, Inv_i, AP_i, L_i)$ and handshake action set H:

$$TA_1 \|_H TA_2 = (Loc, Act_1 \cup Act_2, C, \hookrightarrow, Loc_0, Inv, AP, L)$$

where

$$\begin{aligned} Loc &= Loc_1 \times Loc_2 \\ Loc_0 &= Loc_{0,1} \times Loc_{0,2} \\ C &= C_1 \cup C_2 \\ Inv(\langle \ell_1, \ell_2 \rangle) &= Inv_1(\ell_1) \wedge Inv_2(\ell_2) \\ L(\langle \ell_1, \ell_2 \rangle\rangle) &= L_1(\ell_1) \cup L_2(\ell_2) \end{aligned}$$

$\hookrightarrow$ is defined by

$\alpha \in H$: $\dfrac{\ell_1 \xhookrightarrow{g_1:\alpha,D_1}{}_1 \ell_1' \wedge \ell_2 \xhookrightarrow{g_2:\alpha,D_2}{}_2 \ell_2'}{\langle \ell_1,\ell_2 \rangle \xhookrightarrow{g_1 \wedge g_2:\alpha,D_1 \cup D_2} \langle \ell_1,\ell_2 \rangle}$

$\alpha \notin H$: $\dfrac{\ell_1 \xhookrightarrow{g_1:\alpha,D_1}{}_1 \ell_1'}{\langle \ell_1,\ell_2 \rangle \xhookrightarrow{g_1:\alpha,D_1} \langle \ell_1,\ell_2 \rangle}$ and $\dfrac{\ell_2 \xhookrightarrow{g_2:\alpha,D_2}{}_2 \ell_2'}{\langle \ell_1,\ell_2 \rangle \xhookrightarrow{g_2:\alpha,D_2} \langle \ell_1,\ell_2 \rangle}$

Clock Valuations

- **clock valuation** η for set C of clocks is a function $\eta : C \to \mathbb{R}_{\geq 0}$ assigning each clock $x \in C$ its current value $\eta(x)$
- $\eta + d$ for $d \in \mathbb{R}_{\geq 0}$ is defined by: $(\eta + d(x) = \eta(x) + d$ for all clocks $x \in C$
- *reset* x in η for clock x is defined by:

$$(reset\, x\, in\, \eta)(x) = \begin{cases} \eta(y) & if\; y \neq x \\ 0 & if\; y = x \end{cases}$$

Satisfaction of Clock Constraints

$\models \subseteq Eval(C) \times CC(C)$ is defined by:

$$
\begin{aligned}
&\eta \models true\\
&\eta \models x < c\, iff\, \eta(x) < c\\
&\eta \models x \leq c\, iff\, \eta(x) \leq c\\
&\eta \models x > c\, iff\, \eta(x) > c\\
&\eta \models x \geq c\, iff\, \eta(x) \geq c\\
&\eta \models g \wedge g'\, iff\, \eta \models g \wedge \eta \models g'
\end{aligned}
$$

Transition System from Timed Automata

For the timed automaton $TA = (Loc, Act, C, \hookrightarrow, Loc_0, Inv, AP, L)$ the transition system is: $TS(TA) = (S, Act', \rightarrow, I, AP', L')$

$$
\begin{aligned}
S &= Loc \times Eval(C), \text{ so states are of the form } s = \langle \ell, \eta \rangle\\
Act' &= Act \cup \mathbb{R}_{\geq 0}, \text{ (discrete) actions and time passage actions}\\
I &= \{\langle \ell_0, \eta_0 \rangle | \ell_0 \in Loc_0 \wedge \eta_0(x) = 0 \text{ for all } x \in C\}\\
AP' &= AP \cup ACC(C)\\
L'(\langle \ell, \eta \rangle) &= L(\ell) \cup \{g \in ACC(C) | \eta g\}\\
&\rightarrow \text{is the transition relation defined below}
\end{aligned}
$$

Discrete Transition: $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \eta' \rangle$ if there is a transition labeled $(g : \alpha, D)$ from location ℓ to ℓ' such that:

- g is satisfied by η, i.e. $\eta \models g$
- $\eta' = \eta$ with all clocks in D resets to 0, i.e. $\eta' = reset\, D\, in\, \eta$
- η' fulfills the invariant of location ℓ', i.e. $\eta' \models Inv(\ell')$

Delay Transition $\langle \ell, \eta \rangle \xrightarrow{d} \langle \ell, \eta + d \rangle$ for $d \in \mathbb{R}_{\geq 0}$ if $\eta + d \models Inv(\ell)$

$\Rightarrow$uncountably many states of the form $\langle \ell, \eta + t \rangle$ possible

Timed Paths through $TS(TA)$

Model possible behaviour of TA. Not every Path is realistic:

time convergence: time converges to a specific value
Time convergence is unrealistic and needs to be ignored (similar to unfair paths) $\Rightarrow$only use time-divergent paths.

timelock: passage of time stops
TA is **timelock-free** if no state in $Reach(TS(TA))$ contains a timelock
timelocks are modelling flaws $\Rightarrow$need mechanisms to check for them

zenoness: infinitely many actions take place in finite time.

Zenoness

- A TA that performs infinitely many actions in finite time is **zeno**.
- Path π in $TS(TA)$ is **zeno**, if it is time-convergent and infinitely many actions $\alpha \in Act$ are executed along π
- TA is **non-zeno** if there does not exist a zeno path in $TS(TA)$
  - any π in $TS(TA)$ is time-divergent
  - any time-convergent path has at least one delay transition.
- Sufficient Condition for **Non-Zenoness** (static analysis):
  Let TA with set C of clocks such that for every (control) cycle:

  $$\ell_0 \xhookrightarrow{g_1:\alpha_1,C_1} \ell_1 \xhookrightarrow{g_2:\alpha_2,C_2} \ldots \xhookrightarrow{g_n:\alpha_n,C_n} \ell_n = \ell_0$$

  there exists a clock $x \in C$ such that:

  1. $x \in C$, for some $0 < i \leq n$, and
  2. for all clock evaluations η there exists $c \in \mathbb{N}_{>0}$ such that

  $$\eta(x) < c \text{ implies } (\exists 0 < j \leq n.n \not\models g_j \text{ or } \eta \not\models Inv(\ell_j))$$

Adequate Modelling

A timed automaton is adequately modeling a time-critical system whenever it is **non-zeno** and **timelock-free**.

4.1 Timed CTL

Syntax of Timed CTL (TCTL)

TCTL formula over AP and set C:

$$\Phi := true|a|g|\Phi \wedge \Phi|\neg\Phi|\exists\varphi|\forall\varphi$$

where $a \in AP, g \in ACC(C)$ and φ is a path formula defined by $\varphi := \diamond^J \Phi$ where $J \subseteq \mathbb{R}_{\geq 0}$ is an interval whose bounds are natural: $J : [n,m], (n,m], [n,m), (n,m)$ for $n, m \in \mathbb{N}$ and $n \leq m$, $m = \infty$ allowed for right-open intervalls.

$$\exists\Box^J\Phi = \neg\forall \diamond^J \neg\Phi$$
$$\forall\Box^J\Phi = \neg\exists \diamond^J \neg\Phi$$
$$\diamond\Phi = \diamond^{[0,\infty)}\Phi$$
$$\Box\Phi = \Box^{[0,\infty)}\Phi$$

Semantics of TCTL

$s \models true$		
$s \models a$	iff	$a \in L(\ell)$
$s \models g$	iff	$\eta \models g$
$s \models \neg\Phi$	iff	$\neg s \models \Phi$
$s \models \Phi \wedge \Psi$	iff	$(s \models \Phi)$ and $(s \models \Psi)$
$s \models \exists\varphi$	iff	$\pi \models \varphi$ for some $\pi \in Pahts_{div}(s)$
$s \models \forall\varphi$	iff	$\pi \models \varphi$ for all $\pi \in Pahts_{div}(s)$

Delay Equivalence Relation $\Longrightarrow$

For infinite path fragments in $TS(TA)$ performing ∞ many actions, let

$$s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} s_2 \overset{d_2}{\Longrightarrow} \dots \text{ with } d_0, d_1, d_2 \dots \geq 0$$

denote the equivalence class such that the same time passes during the path. $ExecTime(\pi) = \sum_{i \geq 0} d_i$

Satisfaction Set

$$Sat(\Phi) = \{s \in Loc \times Eval(C) | s \models_{TCTL} \Phi\}$$

$$TA \models \Phi \text{ iff } \forall \ell_0 \in Loc_0 . \langle \ell_0, \eta_0 \rangle \models \Phi$$

where $\eta_0(x) = 0$ for all $x \in C$

TCTL and CTL

TCTL only uses time-divergent paths, therefore:

$$\underbrace{TS(TA) \models_{TCTL} \forall\varphi}_{\text{TCTL semantics}} \text{ but } \underbrace{TS(TA) \not\models_{CTL} \forall\varphi}_{\text{CTL semantics}}$$

Timelock

A state is **timelock-free** iff $\exists\Box true$. I.e., there is a time-divergent path starting in this state.
TA is timelock-free, iff $\forall s \in Reach(TS(TA)) : s \models \exists\Box true$
$\Rightarrow$Timelock-checking with a timed CTL formula.

TCTL Model Checking

$$\underbrace{TA}_{\text{timed automaton}} \models \Phi \iff \underbrace{TS(TA)}_{\text{infinite transition system}} \models \Phi$$

consider finite quotient of $TS(TA)$ $\Rightarrow$Region Transition System $RTS(TA)$
transform TCTL formula Φ into “equivalent” CTL formula $\hat{\Phi}$

$$TA \models_{TCTL} \Phi \iff \underbrace{RTS(TA)}_{finitetransitionsystem} \models_{CTL} \hat{\Phi}$$

1. elimintation of timing parameters

- eliminate all intervalls $J \neq [0\infty)$ from TCTL formulas
- introduce fresh clock z not formerly in TA
- $s \models \exists \diamond^J \Phi$ iff $reset\, z \in\ s \models z \in J \wedge \Phi$
- process $\exists \Box^J \Phi, \forall \diamond^J \Phi, \forall \Box^J \Phi$ similarly

Formally: for any state s of $TS(TA)$ it holds:

$$s \models \exists \diamond^J \Phi \iff \underbrace{s\{z := 0\}}_{\text{state in } TS(TA \oplus z)} \models \exists \diamond ((z \in J) \wedge \Phi)$$

where $TA \oplus z$ it TA over C extended with $z \notin C$
For any state s of $TS(TA) itholds.that:$

(a) $s \models \exists(\Phi \,\mathcal{U}^J \Psi)$ iff $\underbrace{s\{z := 0\}}_{\text{state in } TS(TA \oplus z)} \models \exists((\Phi \vee \Psi)\,\mathcal{U}\,((z \in J) \wedge \Psi))$

(b) $s \models \forall(\Phi \,\mathcal{U}^J \Psi)$ iff $\underbrace{s\{z := 0\}}_{\text{state in } TS(TA \oplus z)} \models \forall((\Phi \vee \Psi)\,\mathcal{U}\,((z \in J) \wedge \Psi))$

2. Clock Equivalence $\cong$ is an equivalence realtion on clock valuations:
   - Equivalent clock valuations satisfy the same clock constraint g:

   $$\eta \cong \eta' \Rightarrow (\eta \models g \iff \eta' \models g)$$

   - Time-divergent paths of equivalent paths are "equivalent" $\Rightarrow$equivalent paths satisfy the same path formulas.
   - The number of equivalence classes under $\cong$ is finite.

   (a) and (b) are ensured, if equivalent states:

   - agree on the integer part of all clock values
   - agree on the ordering of the fractional parts of all clocks.

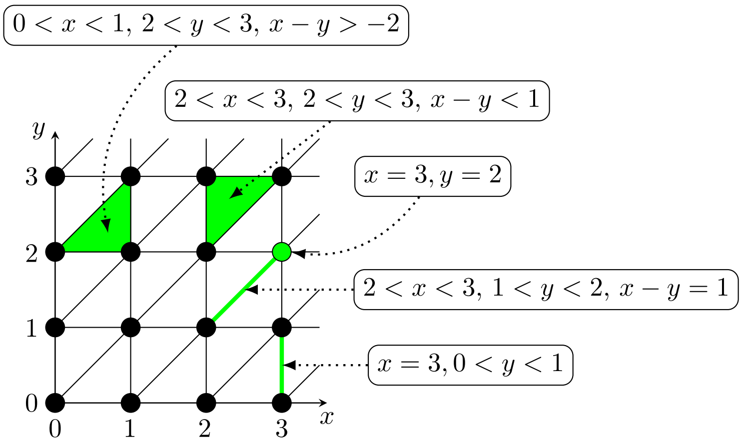
   if clocks exceed the **maximal constant** with which they are compared, their precise value is not of interest.

3. Construct Region Transition System $TS = RTA(TA)$
4. apply CTL model-checking algorithm to check $TS \models \Phi$

Clock Equivalence

η and η' are equivalent, $\eta \cong \eta'$ if:
c_x is the largest constant which x is compared to

- for any $x \in C: \eta(x) > c_x \iff \eta'(x) > c_x$
- for any $x \in C$: if $\eta(x), \eta' \leq c_x$ then:
  $\lfloor \eta(x) \rfloor = \lfloor \eta'(x) \rfloor$ and $frac(\eta(x)) = 0 \iff frac(\eta'(x)) = 0$
- for any $x, y \in C$: if $\eta(x), \eta'(x) \leq c_x$ and $\eta(y), \eta'(y) \leq c_y$, then:
  $frac(\eta(x)) \leq frac(\eta(y)) \iff frac(\eta'(x)) \leq frac(\eta'(y))$



furthermore: $s \cong s'$ iff $\ell = \ell'$ and $\eta \cong \eta'$

Regions

clock region: $[\eta] = \{\eta' \in Eval(C) | \eta \cong \eta'\}$

state region: $[s] = \langle \ell, [\eta] \rangle = \{\langle s, \eta' \rangle | \eta' \in [\eta]\}$ e

Bounds on number of regions

$$|C|! \cdot \prod_{x \in C} c_x \leq |\underbrace{Eval(C) \backslash \cong}_{\text{number of regions}}| \leq |C|! \cdot 2^{|C|-1} \cdot \prod_{x \in C} (2c_x + 2)$$

The number of state regions is $|Loc|$ times larger.
Exponential in number of Clocks.

Preservation of Atomic Properties

1. For $\eta, \eta' \in Eval(C)$ such that $\eta \cong \eta'$:

$$(\eta \models g \text{ if and only if } \eta' \models g) \text{ for any } g \in ACC(TA \cup \Phi)$$

2. For $s, s' \in TS(TA)$ such that $s \cong s'$:

$$s \models a \text{ if and only iff } s' \models a \text{ for any } a \in AP'$$

4.2 Region Automaton

Unbounded Regions

Clock reagion is **unbounded**: $r_\infty = \{\eta \in Eval(C) | \forall x \in C.\eta(x) > c_x\}$

Successor Regions

r' is the **successor** (clock) region of r, $r' = succ(r)$ if either:

1. $r = r_\infty$ and $r = r'$
2. $r \neq r_\infty, r \neq r'$ and $\forall \eta \in r$:

$$\exists d \in \mathbb{R}_{>0}.(\eta + d \in r' \text{ and } \forall 0 \leq d' \leq d.\eta + d' \in r \cup r)$$

The **successor region**: $succ(\langle \ell, r \rangle) = \langle \ell, succ(r) \rangle$

Time Convergence

Time convergent paths that only perform **delay transitions**.
For non-zeno TA and $\pi = s_0 s_1 s_2 \ldots$ a path in $TS(TA)$:

1. π is **time convergent** $\Rightarrow \exists$ state region $\langle \ell, r \rangle$ such that for some j:

$$s_i \in \langle \ell, r \rangle \text{ for all } i \geq j$$

2. If $\exists$ state region with $r \neq r_\infty$ and an index j such that:

$$s_i \in \langle \ell, r \rangle \text{ for all } i \geq j$$

   then π is **time-convergent**.

Region Automaton

For non-zeno TS with $TS(TA) = (S, Act, \rightarrow, I, AP, L)$ let:

$$RTS(TA, \Phi) = (S', Act \cup \{\tau\}, \rightarrow', I', AP', L') \text{ with}$$

$$\begin{aligned}
S' = S \setminus \cong &= \{[s] | s \in S\} && \text{the state regions} \\
I' &= \{[s] | s \in I\} && \text{the initial states} \\
L'(\langle \ell, r \rangle) &= L(\ell) \cup \{g \in AP' \setminus AP | r \models g\} \\
\rightarrow' &: \frac{\ell \xrightarrow{g:\alpha,D} \ell' \quad r \models g \quad reset\ D\ in\ r \models Inv(\ell')}{\langle \ell, r \rangle \xrightarrow{\alpha}' \langle \ell', reset\ D\ in\ r \rangle} && \text{and} \\
& \frac{r \models Inv(\ell) \quad succ(r) \models Inv(\ell)}{\langle \ell, r \rangle \xrightarrow{\tau}' \langle \ell, succ(r) \rangle}
\end{aligned}$$

Correctness

For non-Zeno timed automaton TA and $TCTL_\diamond$ formula Φ:

$$\underbrace{TA \models \Phi}_{\text{TCTL semantics}} \quad \text{iff } \underbrace{RTS(TA, \Phi) \models \Phi}_{\text{CTL semantics}}$$

Timelock Freedom

Non-zeno TA is **timelock-free** iff no reachable state in $RTS(TA)$ is terminal.
⇒timelock freedom checking can be reduced to reachability analysis on $RTS(TA)$

4.3 TCTL Model Checking Algorithm

TCTL Model Checking Algorithm

R:=$RTS(TA \oplus z, \Phi)$; ; `/* with state space` S_{rts} `and labelling` $Lrts$ `*/`
for all $i \leq |\Phi|$ **do**
  for all $\Psi \in Sub(\Phi)$ **with** $|\Psi| = i$ **do**
    switch Ψ **do**

$$
\begin{array}{rcl}
true & : & Sat_R(\Psi) := S_{rts}; \\
a & : & Sat_R(\Psi) := \{s \in S_{rts} | a \in L_{rts}(s)\}; \\
\Psi_1 \wedge \Psi_2 & : & Sat_R(\Psi) := \{s \in S_{rts} | \{a_{\Psi_1}, a_{\Psi_2}\} \subseteq L_{rts}(s)\}; \\
\neg\Psi' & : & Sat_R(\Psi) := \{s \in S_{rts} | a_{\Psi'} \notin L_{rts}(s)\}; \\
\exists(\Psi_1 \mathcal{U}^J \Psi_2) & : & Sat_R(\Psi) := Sat_{CTL}\left(\exists((a_{\Psi_1} \vee a_{\Psi_2})\,\mathcal{U}\,(z \in J) \wedge a_{\Psi_2})\right); \\
\forall(\Psi_1 \mathcal{U}^J \Psi_2) & : & Sat_R(\Psi) := Sat_{CTL}\left(\forall((a_{\Psi_1} \vee a_{\Psi_2})\,\mathcal{U}\,(z \in J) \wedge a_{\Psi_2})\right);
\end{array}
$$

    endsw
    for all $s \in S_{rts}$ **with** $s\{z := 0\} \in Sat_R(\Psi)$ **do**
      $L_{rts}(s) := L_{rts}(s) \cup \{a_\Psi\}$; `/* add` a_Ψ `to labelling of state regions where` Ψ `holds */`
    end
  end
end
return $I_{rts} \subseteq Sat_R(\Phi)$

Time Complexity

timed automaton TA, TCTL Φ, N is number of states, K is number of transitions in $RTS(TA, \Phi)$:

$$TA \models \Phi : \quad \mathcal{O}((N + K) \cdot |\Phi|)$$

4.4 Clock Zones

Number of clock regions too large, need coarser abstraction: **clock zones**, efficient representation in **difference bound matrices**

Forward Analysis

- start from initial configuration
- determine configurations that are reachable within $1, 2, \ldots, n$ steps
- termination: goal configuration reached or no new successors (⇒fixpoint)

Backward Analysis

- start from goal configuration
- determine configurations that can reach the goal within $1, 2, \ldots, n$ steps

- termination: initial configuration reached or no new predecessors ($\Rightarrow$fixpoint)

Clock Zones

Symbolic representation of timed automata configurations.
For set z of clock valuations and edge $e = \ell \xrightarrow{g:\alpha,D} \ell'$ let:

$$Pre_e(z) = \{\eta \in \mathbb{R}^n_{\geq 0} | \exists \eta' \in z, d \in \mathbb{R}_{\geq 0}.(\eta + d \models g) \land \eta' = reset\, D\, in\, (\eta + d)\}$$
$$Post_e(z) = \{\eta' \in \mathbb{R}^n_{\geq 0} | \exists \eta \in z, d \in \mathbb{R}_{\geq 0}.(\eta + d \models g) \land \eta' = reset\, D\, in\, (\eta + d)\}$$

Intuition:

- $\eta \in Pre_e(z)$ if for some $\eta' \in z$ and delay d holds: $(\ell, \eta) \xRightarrow{d} \ldots \xrightarrow{e} (\ell', \eta')$
- $\eta' \in Post_e(z)$ if for some $\eta \in z$ and delay d holds: $(\ell, \eta) \xRightarrow{d} \ldots \xrightarrow{e} (\ell', \eta')$

Zones

Clock Constraints are conjunctions of constraints of the form

- $x \prec c$ and $x - y \prec c$ for $\prec \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{Z}$

A **Zone** is a (maximal) set of clock valuations satisfying a clock constraint.
Clock zone of g : $[\![g]\!] = \{\eta \in Eval(C) | \eta \models g\}$
A **state zone** of $s = \langle \ell, \eta \rangle$ is $\langle \ell, z \rangle$ with $\eta \in z$
For zone z and edge e, $Post_e(z)$ and $Pre_e(z)$ are zones.

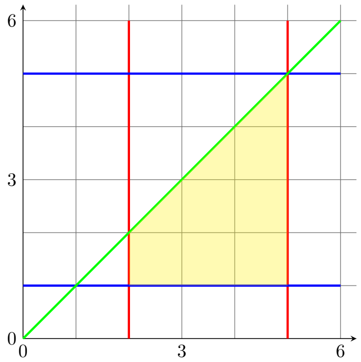
Operations on Zones

Future of z:	$\overrightarrow{z} = \{\eta + d | \eta \in z \land d \in \mathbb{R}_{\geq 0}\}$
Past of z:	$\overleftarrow{z} = \{\eta - d | \eta \in z \land d \in \mathbb{R}_{\geq 0}\}$
Intersection of two zones:	$z \cap z' = \{\eta | \eta \in z \land \eta \in z'\}$
Clock Reset in a zone:	reset D in $z = \{$ reset D in $\eta | \eta \in z\}$
Inverse Clock Reset of a zone:	reset$^-1\, D$ in $z = \{\eta |$ reset D in $\eta \in z\}$

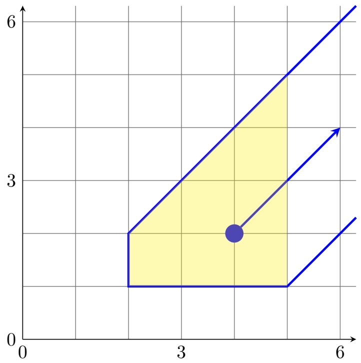
zones are closed under these operations

Clock Zones Examples

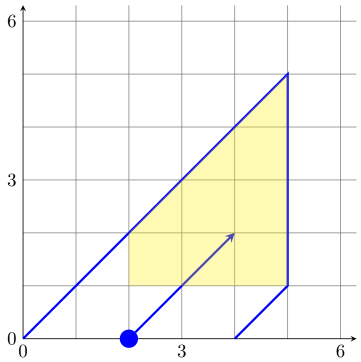
$(2 < x \leq 5) \wedge (1 < y \leq 5) \wedge (y - x \leq 0)$



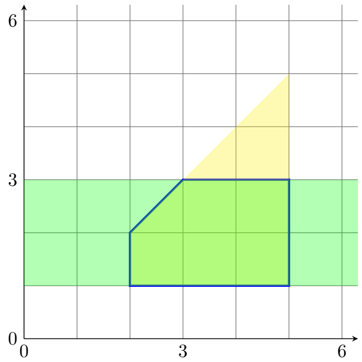
Future of z:



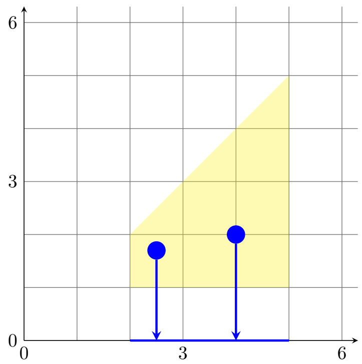
Past of z:



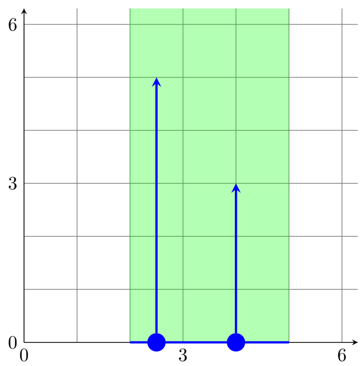
Intersection of two zones: $z' = 1 \leq y \leq 3$



Clock reset in a zone:



Inverse Clock reset:



Symbolic Representation of Successor and Predecessors

For edge $e = \ell \xrightarrow{g:\alpha,D} \ell'$ we have:

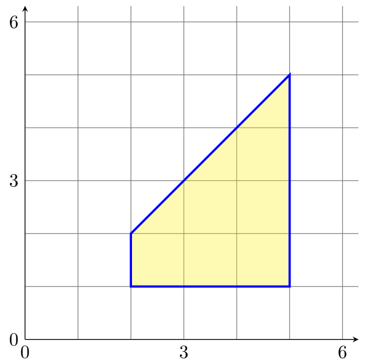
$$Pre_e(z) = \{\eta \in \mathbb{R}^n_{\geq 0} | \exists \eta' \in z, d \in \mathbb{R}_{\geq 0}.(\eta + d \models g) \wedge \eta' = reset\, D\, in\, (\eta + d)\}$$
$$Post_e(z) = \{\eta' \in \mathbb{R}^n_{\geq 0} | \exists \eta \in z, d \in \mathbb{R}_{\geq 0}.(\eta + d \models g) \wedge \eta' = reset\, D\, in\, (\eta + d)\}$$

Express this symbolically with operations on zones:

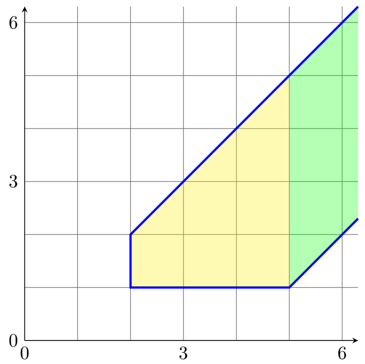
$$Pre_e(z) = \overleftarrow{reset^{-1}\, D\, in\, (z \cap [\![D = 0]\!]) \cap [\![g]\!]}$$
$$Post_e(z) = reset\, D\, in\, (\overrightarrow{z} \cap [\![g]\!])$$

Successor (Forward Analysis)

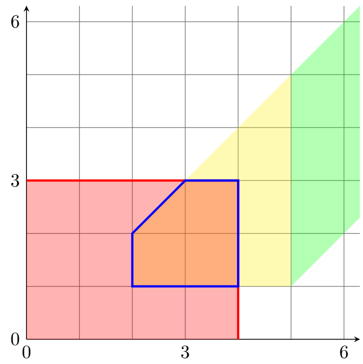
zone: Z



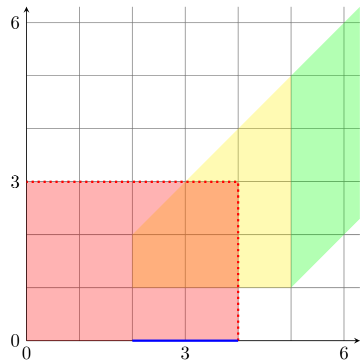
advance time: $\overrightarrow{Z}$



satisfy guard: $\overrightarrow{Z} \cap g$

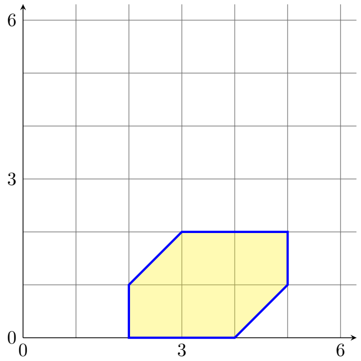


reset clock: $[y \leftarrow 0](\overrightarrow{Z} \cap g)$

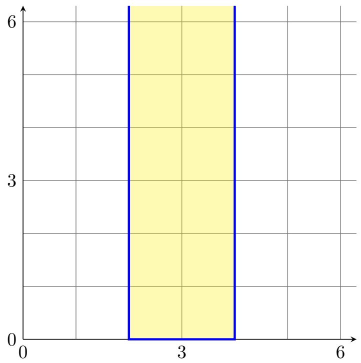


Predecessor (Backward Analysis)

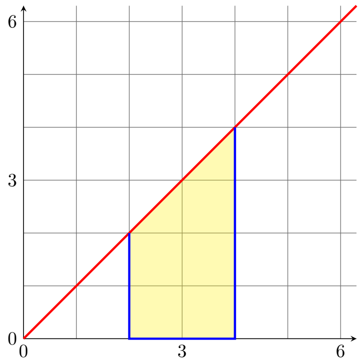
zone: Z



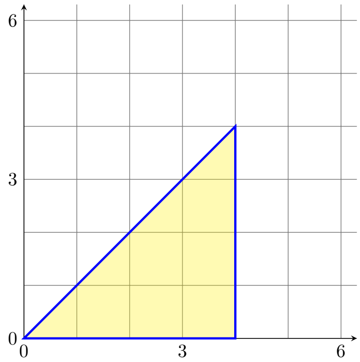
undo clock reset: $[C \leftarrow]^{\overbrace{-1}^{y}} (Z \cap (\overbrace{C = 0}^{x}))$



consider guard: $([C \leftarrow]^{-1}(Z \cap (C = 0))) \cap g$



go back in time: $\overleftarrow{([C \leftarrow]^{-1}(Z \cap (C = 0))) \cap g}$



Backward Symbolic Reachability Analysis

Backward Symbolic Transition System of TA is inductively defined by:

$$\frac{e = \ell \xleftrightarrow{g:\alpha,D} \ell',\ z = Pre(z')}{(\ell', z') \Leftarrow (\ell, z)}$$

Computation Schema:

$$\begin{array}{rcl}
T_0 & = & \{(\ell, \mathbb{R}^n_{\geq 0}) | \ell \text{ is a goal location}\} \\
T_1 & = & T0 \cup \{(\ell, z) | \exists (\ell', z') \in, T_0.(\ell', z') \leftarrow (\ell, z) \text{ and } \ell' = \ell \text{ implies } z \not\subseteq z'\} \\
\ldots & & \ldots \\
T_{k+1} & = & T0 \cup \{(\ell, z) | \exists (\ell', z') \in, T_0.(\ell', z') \leftarrow (\ell, z) \text{ and } \ell' = \ell \text{ implies } z \not\subseteq z'\} \\
\ldots & & \ldots
\end{array}$$

until

- computation stabilizes (fixpoint)
- initial configuration reached (property violated)

Forward Symbolic Reachability Analysis

Forward Symbolic Transition System of TA is inductively defined by:

$$\frac{e = \ell \xrightarrow{g:\alpha,D} \ell',\ z' = Post(z)}{(\ell, z) \Rightarrow (\ell', z')}$$

Computation Schema:

$$\begin{array}{ccc} T_0 & = & \{(\ell_0, z_0) | \forall x \in C.z_o(x) = 0\} \\ T_1 & = & T0 \cup \{(\ell', z') | \exists (\ell, z) \in, T_0.(\ell, z) \to (\ell', z') \text{ and } \ell = \ell' \text{ implies } z \not\subseteq z'\} \\ \dots & & \dots \\ T_{k+1} & = & T0 \cup \{(\ell', z') | \exists (\ell, z) \in, T_k.(\ell, z) \to (\ell', z') \text{ and } \ell = \ell' \text{ implies } z \not\subseteq z'\} \\ \dots & & \dots \end{array}$$

until

- computation stabilizes (fixpoint)
- goal configuration reached (property violated)

Forward Symbolic Reachability Analysis is correct but may not terminate.

Abstract Forward Reachabilty (only proposed)

Let γ associate sets of valuations to sets of valuations.
Forward Symbolic Transition System of TA is inductively defined by:

$$\frac{e = \ell \xrightarrow{g:\alpha,D} \ell',\ z' = \gamma(z)}{(\ell, z) \Rightarrow \gamma(\ell', \gamma(z')}$$

Computation Schema:

$$\begin{array}{ccc} T_0 & = & \{(\ell_0, \gamma(z_0) | \forall x \in C.z_o(x) = 0\} \\ T_1 & = & T0 \cup \{(\ell', z') | \exists (\ell, z) \in, T_0.(\ell, z) \to \gamma(\ell', z')\} \\ \dots & & \dots \\ T_{k+1} & = & T0 \cup \{(\ell', z') | \exists (\ell, z) \in, T_k.(\ell, z) \to \gamma(\ell', z')\} \\ \dots & & \dots \end{array}$$

inclusion check and termination as before.

- Soundness: (anything found in abstract system is also in actual system)

$$\underbrace{\langle \ell_0, \eta_0 \rangle \to^* \langle \ell, \eta \rangle}_{\text{reachability in } TS(TA)} \implies \exists \ \underbrace{\langle \ell_0, \eta_0 \rangle \to^* \langle \ell, \eta \rangle}_{\text{reachability in } TS(TA)} \text{ with } \eta \in z$$

- Completeness: (anything from the actual system can also be found in abstract system)

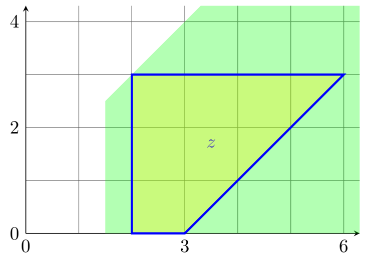
$$\underbrace{\langle_0, \eta_0 \to^* \langle \ell, \eta \rangle}_{\text{reachability in } TS(TA)} \implies \underbrace{\exists \langle \ell_0, \gamma(\{\eta_0\}) \rangle \implies {}^*_\gamma \langle \ell, z \rangle}_{\text{abstract symbolic reachability}} \text{ for some } z \text{ with } \eta \in z$$

for any γ soundness and completeness are desirable

- Finiteness: $\{\gamma(z)|\gamma \text{ defined on } z\}$ is finite
- Correctness: γ is sound wrt. reachability
- Completeness: γ is complete wrt. reachability
- Effectiveness: γ is defined on zones and $\gamma(z)$ is a zone

k-Normalization

A k **bounded zone** is described by a k-bounded clock constraint (consisting of k atomic clock constraints). The $norm_k(Z)$ is the smallest k-bounded zone containing zone z



- Finiteness: $norm_k(\bullet)$ is a finite abstraction operator
- Correctness: $norm_k(\bullet)$ is sound wrt. reachability (provided k is the maximal constant appearing in the constraints of TS)
- Completeness: $norm_k(\bullet)$ is complete wrt. reachability, since $z \subseteq norm_k(\bullet)$ so $norm_k(\bullet)$ is an over-approximation
- Effectiveness $norm_k(\bullet)$ is a zone

4.5 Difference Bound Matrices

Difference Bound Matrices

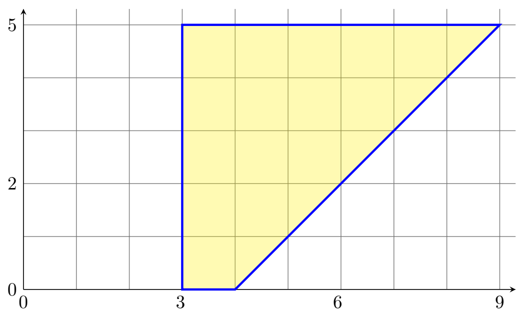
Zone z over C is represented by DBM Z of cardinality $|C+1| \cdot |C+1|$. For $C = \{x_1, \ldots, x_n\}$, let $C_0 = \{x_0\} \cup C$ with $x_0 = 0$ and:

$$Z(i,j) = (c, \prec) \iff x_i - x_j \prec c$$

Further:

$$\begin{aligned}
Z(i,j) &:= (c, \prec) \text{ for each bound } x_i - x_j \prec c \text{ in } z \\
Z(i,j) &:= \infty \text{ (no bound) if clock difference } x_i - x_j \text{is unbounded in } z \\
Z(0,j) &:= (0, \leq)\, 0 - x \leq 0 \text{ {\tiny all clocks are positive}} \\
Z(i,i) &:= (0, \leq) \text{ each clock is at most itself}
\end{aligned}$$

rows are for lower bounds, columns for higher bounds, always: $x_0 = 0$



$$\Rightarrow \begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix}$$

Canonical Form

A zone z is in **canonical form** iff no constraint can be strengthened without reducing $[\![z]\!] = |\{\eta|\eta \in z\}|$ the size of the zone.
For each zone z there eixts a **unique** zone z' such that $[\![z]\!] = [\![z']\!]$ and z' is in canonical form.
Zone z is in its **canonical form** iff DBM Z satisfies:

$$Z(i,j) \leq Z(i,k) + Z(k,j) \text{ for any } x_i, x_j, x_k \in C_0$$

Operations on DBM entries

Let $\preceq \in \{<, \leq\}$

- Comparison of DBM entries: $(c, \preceq) < (c', \preceq')$ if $c < c'$
- Addition of DBM entries:

$$c + \infty = \infty$$
$$(c, \leq) + (c', \leq) = (c + c', \leq)$$
$$(c, <) + (c', \leq) = (c + c', <)$$

Transform DBM into its canonical form

Deriving the **tightest constraint** on a pair of clocks in a zone is equivalent to finding the shortest weighted path between their vertices.
For Example Floyd-Warshall's All-Pairs Shortest-Path Algorithm

for all $k := 1\,to\,n$ **do**
  for $i := 1$ **to** n **do**
    for $j := 1$ **to** n **do**
      $\underbrace{path[i][j]}_{\text{i to j}} = min(\underbrace{path[i][j]}_{curMin}, \underbrace{path[i][k] + path[k][j]}_{candidateMin});$
    end
  end
end

Worst Case time complexity in $\mathcal{O}(|C_0|^3)$

A canonical zone may contain redundant constraints: $x_i \xrightarrow{(n,\preceq)} x_j$ is **redundant** if a path if a path from x_i to x_j has weight of at most $(n, \preceq)$

DBM Operations for Property Checking

- Nonemptiness: $[\![Z]\!] \neq \varnothing$?
  - $Z = \varnothing$ if $x_i - x_j \preceq c$ and $x_j - x_i \preceq' c'$ and $(c, \preceq) < (c', \preceq')$

  $\Rightarrow$ search for negative cycles in the graph representation
  - mark Z when upper bound is set to value $<$ its corresponding lower bound
- Inclusion test: is $[\![Z]\!] \subseteq [\![Z']\!]$?
  for DBMs in canonical form, test whether $Z(i,j) \leq Z'(i,j)$ for all $i, j \in C_0$

might be $|Z(\ldots)|$ if canonicalization does not remove negative values.

- Satisfaction: does $Z \models g$?
  check whether $[\![Z \wedge g]\!] = \varnothing$ (if yes, it does not)

DBM Operations Delays

- Future: determine $\overrightarrow{Z}$
  - remove upper bounds on any clock:
    $$\overrightarrow{Z}(i,0) = \infty \text{ and } \overrightarrow{Z}(i.j) = Z(i,j) \text{ for } j \neq 0$$
  - Z is canonical $\implies \overrightarrow{Z}$ is canonical
- Past: determine $\overleftarrow{Z}$
  - set the lower bounds on all individual clocks to $(0, \preceq)$:
    $$\overleftarrow{Z}(i,0) = \infty \text{ and } \overleftarrow{Z}(i,j) = Z(i,j) \text{ for } j \neq 0$$

this formula may be wrong in the slides: slide 3-164

- Conjunction: $[\![Z]\!] \wedge (x_i - x_j \preceq n)$
  - if $(n, \preceq) < Z(i,j)$ then $Z(i,j) := (n, \preceq)$ else do nothing
  - put Z into canonical form (in time $\mathcal{O}(|C_0|^2)$ using that only $Z(i,j)$ changed.
- Clock Reset: $x_j := d$ in Z
  $$Z(i,j) := (d, \leq) + Z(0,j) \text{ and } Z(j,i) := Z(j,0) + (-d, \leq)$$
- k-Normalization: $norm_k(Z)$
  - remove all bounds $x - y \preceq m$ for which $(m, \preceq) > (k, \leq) \rightarrow \infty$
  - set all bounds $x - y \preceq m$ with $(m, \preceq) < (-k, <)$ to $(-k, <)$
  - put the DBM back into canonical form (Floyd-Warshall)

5 Probabilistic Model Checking

- Functional Requirements
  - Functions or Services that the system has to provide.
- Nonfunctional Requirements
  - properties not directly related to functional correctness (often quantitative)
  - response time, reliability, dependability, performance, quality of service

System Correctness

A system is correct if it is capable of doing what it is expected to do over a certain period of time with a certain probability.

Probabilistic Approaches:

- Termination of probabilistic programs
  does a probabilistic program terminate with probability one?
- Markov decision processes
  stochastic and nondeterministic behaviour, does a certain (linear) temporal logic formula hold with probability p?

- Discrete-time Markov Chains
  can we reach a goal state via a given trajectory with probability p?
- Discrete Markov Decision Process
  What is maximal (or minimal) probability of doing something?
- Continuous-time Markov Chains
  Can we do so within a given time interval I?

Probabilistic Model Checking

not probabilistic approaches to model checking like Monte-Carlo Model Checking or Random Walks, which perform incomplete sampling of state space

Checking of Reachability and Probabilistic temporal logic formulae

- time-bounded reachability
- long-run averages, steady state

Measurable Space

- A **sample space** Ω of a chance experiment is a set of values representing the possible outcomes of the experiment.
- A **σ-algebra** is a pair $(\Omega, \mathcal{F})$ with $\Omega \neq \varnothing$ and $\mathcal{F} \subseteq 2^{\Omega}$ a collection of subsets of sample space Ω such that:
  1. $\Omega \in \mathcal{F}$ (äll events are possible")
  2. $A \in \mathcal{F} \implies (\Omega - A) \in \mathcal{F}$ (A is a set of events, so the complement of events is also possible)
  3. $(\forall i \geq 0 . A_i \in \mathcal{F}) \implies (\bigcup_{i \geq 0} A_i) \in \mathcal{F}$ (the countable union of all sets of possible events are also possible)
- elements of $\mathcal{F}$ are called **events**
- the pair $(\Omega, \mathcal{F})$ is called a **measurable space**

Probability Space

A **probability space** $\mathcal{P}$ is a structure $(\Omega, \mathcal{F}, Pr)$ with:

- $(\Omega, \mathcal{F}$ is a σ-algebra
- $Pr : \{ \to [0, 1]$ is a **probability measure**, i.e.
  1. $Pr(\Omega) = 1$, i.e. Ω is the certain event
  2. $Pr\left(\bigcup_{i \in I} A_i\right) = \sum_{i \in I} Pr(A_i)$ for any $A_i \in \mathcal{F}$ with $\underbrace{A_i \cap A_j = \varnothing}_{\text{independent events}}$ for $i \neq j$ where $\{A_i\}_{i \in I}$ is finite or countably infinite

The elements in $\mathcal{F}$ of a probability space $(\Omega, \mathcal{F}, Pr)$ are called **measurable events**.

- No possible outcome of chance experiments is missed when considering Ω.
- Probabilities for different A_is add up if the A_is are pairwise disjoint.

Properties of Probabilities

For measurable events A, B and A_i and probability measure Pr:

$$\begin{aligned}
Pr(A) &= 1 - Pr(\Omega - A) \\
Pr(A \cup B) &= Pr(A) + Pr(B) - Pr(A \cap B) \\
Pr(A \cap B) &= \underbrace{Pr(A|B)}_{A \text{ happens if } B \text{ happens}} \cdot Pr(B) \\
A \subseteq B &\implies Pr(A) \leq Pr(B) \\
Pr(\bigcup_{n \geq 1} A_n) &= \sum_{n \geq 1} Pr(A_n) \qquad \text{provided } A_n \text{ are pairwise disjoint}
\end{aligned}$$

Discrete Probability Space

Pr is a **discrete probability** measure on $(\Omega, \mathcal{F})$ if

- there is a countable set $A \in \Omega$ such that for $a \in A$:

$$\{a\} \in \mathcal{F} \text{ and } \sum_{a \in A} Pr(\{a\}) = 1$$

$(\Omega\mathcal{F}, Pr)$ is then called a **discrete probability space**, otherwise it is a **continuous probability space**.

Measurable Function

Let $(\omega, \mathcal{F}$ and $(\Omega', \mathcal{F}'$ be measurable spaces. Function $f : \Omega \rightarrow \Omega'$ is a **measurable function** if

$$f^{-1}(A) = \{a | f(a) \in A\} \in \mathcal{F} \text{ for all } A \in \mathcal{F}''$$

Random Variable

Measurable function $X : \Omega \rightarrow \mathbb{R}$. $\mathbb{R}$ is a **random variable**.
The **probability distribution** of X is $Pr_X = Pr \circ X^{-1}$ where Pr is a probability measure on $(\Omega, \mathcal{F})$.

Distribution Function

The **Distribution Function** F_X of random variable X is defined by:

$$F_X(d) = \underset{X}{Pr}((-\infty, d]) = Pr(\underbrace{\{a \in \Omega | X(a) \leq d\}}_{\{X \leq d\}}) \text{ for real } d$$

properties:

- F_X is monotonic and right-continuous (increases with greater d)
- $0 \leq F_X(d) \leq 1$

- $lim_{d \to -\infty} F_X(d) = 0$
- $lim_{d \to \infty} F_X(d) = 1$

Distribution Function for Continuous Random Variable

The **distribution function** F_X of random variable X is defined for $d \in \mathbb{R}$ by:

$$F_X(d) = \underset{X}{Pr}(X \in (-\infty, d]) = Pr(\{a \in \Omega | X(a) \leq d\})$$

In the continuous case, F_X is called the **cumulative density function**.

Distribution Function as Sums/Integrals

- For discrete random variable X, F_X can be written as

$$F_X(d) = \sum_{d_i \leq d} \underset{X}{Pr}(X = d_i)$$

- For continuous random variable X, F_X can be written as:

$$F_X(d) = \int_{-\infty}^{d} f_X(i) du \text{ with } f \text{ the density function}$$

Discrete-Time Markov Chains

State-transition systems augmented with probabilities:

States: set of states representing possible configurations of the system being modeled.

Transitions: transitions between states model evolution of systems states; occur in discrete time-steps

Probabilities: probabilities of making transitions between states are given by discrete probability distributions

A **DTMC** $\mathcal{M}$ is a tuple $(S, P, \iota_{init}, AP, L)$ with

- S is a countable nonempty set of states (preferably finite)
- $P : S \times S \to [0, 1]$ transition probability function, s.t. $\sum_{s'} P(s, s') = 1$
  $P(s, s')$ is the probability to jump from s to s' in one step
- $\iota_{init} : S \to [0, 1]$ the initial distribution with $\sum_{s \in S} \iota_{init}(s) = 1$
  - $\iota_{init}(s)$ is the probability that system starts in state s
  - state s for which $\iota_{init}(s) > 0$ is an initial state
- $L : S \to 2^{AP}$, the labeling function

next state is always chosen probabilistically, no non-determinism to model concurrency $\Rightarrow$Markov Decision Processes (MDPs)

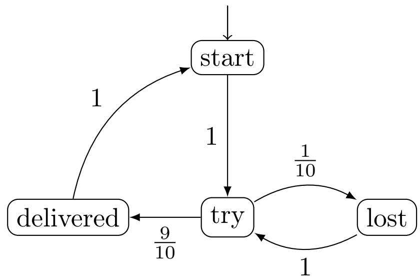
Properties in DTMC

- State graph of DTMC $\mathcal{M}$ is a digraph $G = (V, E)$ with

  - vertices V are states of $\mathcal{M}$ and $(s, s') \in E \iff P(s, s') > 0$

- Paths in $\mathcal{M}$ are maximal (i.e. infinite) paths in its state graph
- Notations: $Paths(\mathcal{M})$ and $Paths_{fin}(\mathcal{M}$ denote the set of finite paths in $\mathcal{M}$
- Direct successors and predecessors
  - $Post(s) = \{s' \in S | P(s, s') > 0\}$ and $Pre(s) = \{s' \in S | P(s', s) > 0\}$
  - $Post^*(s)$ and $Pre^*(s)$ are reflexive and transitive closures
- Absorbing States:
  - state of MC $\mathcal{M}$ is called absorbing iff $Post^*(s) = \{s\}$ (state cannot be left)
  - then $P(s, s) = 1$ and $\forall t \neq s : P(s, t) = 0$

Example for Discrete-Time Markov Chain



$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{10} & \frac{9}{10} \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \quad \iota_{init} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

probability to be in states after one step: $\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$

Paths and Probabilites

need to define probability space over paths

- sample space $Path(s)$ = set of all infinite paths from a state s.
- events: sets of inifinite paths from s
- basic events: **cylinder sets**
- cylinder set $Cyl(\omega)$ for finite path ω = set of infinite paths with the common finite prefix ω

reasoning about quantitative properties of DTMCs (e.g. probabilistic reachability) means reasoning about cylinder sets, i.e., set of paths
sound stochastic basis ⇒relate them to measurable spaces and σ-algebras

σ-Algebra

Let Ω be an arbitrary non-empty set
$(\Omega, \mathcal{F})$ with $\mathcal{F} \subseteq 2^\Omega$ is a σ**-algebra** on Ω if:

- $\varnothing \in \mathcal{F}$
- $E \in \mathcal{F} \implies \Omega \setminus E \in \mathcal{F}$
- $(\forall i \in \mathbb{N}. E_i \in \mathcal{F} \implies \bigcup_{i \in \mathbb{N}} E_i \in \mathcal{F}$

Elements of $\mathcal{F}$ are called **measurable sets** or **events**.
For any family $\mathcal{F}$ of subsets of Ω:

- there exists a unique smallest σ-algebra on Ω containing $\mathcal{F}$

Probability Space

A **probability space** is a structure $(\Omega, \mathcal{F}, Pr)$ with:

- σ-algebra $(\Omega, \mathcal{F})$
- $Pr : \mathcal{F} \to [0, 1]$ is a **probability measure**, i.e.
  1. $Pr(\Omega) = 1$
  2. $Pr(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} Pr(E_i)$ for $E_i \in \mathcal{F}$ and $E_i \cap E_j = \varnothing$ for $i \neq j$
- $Pr(E)$ is the probability of E, i.e., E is **measurable**

Properties of Probability Measures

- An event E with $Pr(E) = 1$ is called **almost sure**
  $Pr(D) = Pr(E \cap D) + \underbrace{Pr(D \setminus E}_{=0} = Pr(E \cap D)$
- $E_1, \ldots E_n$ are almost sure implies $\bigcap_{1 \leq i \leq n} E_i$ is almost sure
- For any Ω and $\mathcal{F} \subseteq 2^{\Omega}$ there exists a smallest σ-algebra containing {
  it is obtained by taking the intersection over all σ-algebras on Ω that contain $\mathcal{F}$

Probability Space on DTMC Paths

- Events are **infinite paths** in the DTMC $\mathcal{M}$, i.e., $\Omega = Paths(\mathcal{M})$
- σ-algebra on $\mathcal{M}$ is generated by **cylinder sets** of finite paths $\hat{\pi}$:

  $$Cyl(\hat{\pi}) = \{\pi \in Paths(\mathcal{M}) | \hat{\pi} \text{ is a prefix of } \pi\}$$

  cylinder sets serve as events of the smallest σ-algebra on $Paths(\mathcal{M})$
- Pr is the **probability measure** on the σ-algebra on $Paths(\mathcal{M}$:

  $$Pr(Cyl(s_0 \ldots s_n)) = \iota_{init}(s_0) \cdot P(s_0 \ldots s_n)$$

  where

  - $P(s_0 s_1 \ldots s_n) = \prod_{0 \leq i \leq n} P(s_i, s_{i+1}$ if $n > 0$
  - $P(s_0) = 1$ for all paths containing a single state

Reachability Probabilities

What is the probability to reach a set of states $B \subseteq S$ in DTMC $\mathcal{M}$?
Which event does $\diamond B$ mean formally?
the union of all cylinders $Cyl(s_0 \dots s_n)$ where $s_0 \dots s_n$ is an initial path fragment $\mathcal{M}$ with $s_0, \dots, s_{n-1} \notin B$ and $s_n \in B$

$$\begin{aligned} Pr(\diamond B) &= \sum_{s_0 \dots s_n \in Paths_{fin}(\mathcal{M}) \cap (S \setminus B)^* B} Pr(Cyl(s_0 \dots s_n)) \\ &= \sum_{s_0 \dots s_n \in Paths_{fin}(\mathcal{M}) \cap (S \setminus B)^* B} \iota_{init} \cdot P(s_0 \dots s_n) \end{aligned}$$

notice: infinite sum

Computing Reachability Properties in finite DTMCs

- Let $Pr(s \models \diamond B) = Pr_s(\diamond B) = Pr_s\{\pi \in Paths(s) | \models \diamond B\}$ where Pr_s is the probability measure in $\mathcal{M}$ with only initial state s
- Let variable $x_s = Pr(s \models \diamond B)$ for any state s
  - if B is not reachable from s then $x_s = 0$
  - if $s \in B$ then $x_s = 1$
- For any state $s \in Pre^*(B) \setminus B$:

$$x_s = \underbrace{\sum_{t \in S \setminus B} P(s,t) \cdot x_t}_{\text{reach B via t}} + \underbrace{\sum_{u \in B} P(s,u)}_{\text{reach B in one step}}$$

- Rewrite equations:

$$x = Ax + b$$

  - vector $x = (x_s)_{s \in \tilde{S}}$ with $\tilde{S} = Pre^*(B) \setminus B$, $x_s = 1$ if reachable, $x_s = 0$ if not
  - $A = (P(s,t))_{s,t \in \tilde{S}}$ the transition probabilities in $\tilde{S}$
  - $b = (b_s)_{s \in \tilde{S}}$ contains the probabilities to reach B within one step
- Linear equation system: $(I - A)x = b$
  - More than one solution may exist, if $I - A$ has no inverse (i.e. is singular) $\Rightarrow$characterize the desired probability as least fixed point

Algorithm Scheme

- two phase algorithm
  1. using graph search, determine the set of $\tilde{S}$ of all states that can reach B
  2. generate A and b and solve equation system $(I-A)x = b$
- if $I - A$ singular, i.e., it does not have an inverse, then $(I-A)x = b$ has more than one solution
  - characterize the solution as the least solution in $[0,1]^{\tilde{S}}$
- calculate probability vector with iterative approximation
- consider **constrained reachability** $\underbrace{c}_{\text{constraint}} \; \mathcal{U}^{\leq n} \; \underbrace{B}_{\text{reach set } B}$
  - $C\mathcal{U} \; \leq nB$ is the union of the basic cylinders of fragments

    $$s_0 s_1 \dots s_k \text{ with } k \leq n \text{ and } s_i \in C \text{ for all } 0 \leq i < k \text{ and } s_k \in B$$

  - Let $S_{=0}, S_{=1}, S_?$ be a patition of S such that:
    * $B \subseteq S_{=1} \subseteq \{s \in S | Pr(s \models C\,\mathcal{U}\, B) = 1\}$ certain satisfaction
    * $S \setminus (C\,\mathcal{U}\, B) \subseteq S_{=0} \subseteq \{s \in S | Pr(s \models C\,\mathcal{U}\, B) = 0\}$ will certainly not satisfy
    * all states in $S_?$ belong to $S \setminus B$ don't know yet
  - Let $A = \underbrace{(P(s,t))_{s,t \in S_?}}_{\text{Probability to transit inside } S_?}$ and $(b_s)_{s \in S_?}$ where $\underbrace{b_s = P(s, S_{=1}}_{\text{one step proabbility to reach } S_{=1}}$
- Define: $y \leq y'$ iff $y_s \leq y'_s$ for all $s \in S$
  - y is a **fixed point** of $F : [0,1]^S \to [0,1]^S$ if $F(y) = y$
  - x is a **least fixed point** of F if $x \leq y$ for any other fixed point y of F
- The vector $x = (Pr(s \models C\,\mathcal{U}\, B))_{s \in S_?}$ is the least fixed point of $F : [0,1]^{S_?} \to [0,1]^{S_?}$ given by $F(y) = A \cdot y + b$
  - $x^{(n)} = \left(x_s^{(n)}\right)_{s \in S_?}$ where for any $s : x_s^{(n)} = Pr\left(s \models C\,\mathcal{U}^{\leq n} S_{=1}\right)$
  - $x^{(0)} \leq x^{(1)} \leq x^{(2)} \leq \dots \leq x$ increasing monotonically
  - $x = \lim\limits_{n \to \infty} x^{(n)}$
    $\Rightarrow x$ is the least solution of $Ax + b = x$ in $[0,1]^{S_?}$
    $\Rightarrow$Approximation: $x^{(0)} = 0$ and $x^{(n+1)} = Ax^{(n)} + b$ for $n \geq 0$
  - Power Method: compute vectors $x^{(n)}$ iteratively, abort on:

    $$\max_{s \in S_?} |x_s^(n+1) - x_s^{(n)}| < \epsilon \text{ for some small tolerance } \epsilon$$

    convergence is guaranteed, alternate ways: eg Jacobi, Gauss-Seidel, successive overrelaxation

Unique Solution

For $B, C \subseteq S$ the vector

$$(PR(s \models C \,\mathcal{U}\, B))_{s \in S_?}$$

is the **unique solution** of the linear equation system:

$$x = Ax + b \text{ where } A = (P(s,t))_{s,t \in S_?} \text{ and } b = (P(s, S_{=1}))_{s \in S_?}$$

Example how matrix works ⇒exercises

Transient Probabilities

A **transient probability** is the probability to reside in some state t after exactly n steps.

$$\Theta_n^{\mathcal{M}}(t) = \sum_{s \in S} \iota_{init}(s) \cdot P^n(s,t)$$
$$\Theta_n^{\mathcal{M}} = \underbrace{P \cdot P \cdot \dots \cdot P}_{n \text{ times}} \cdot \iota_{init} = P^n \cdot \iota_{init}$$

⇒Compute $\Theta_n^{\mathcal{M}}$ by successive vector-matrix multiplications (reduces numerical instability)

$$\Theta_0^{\mathcal{M}} = \iota_{init}$$
$$\Theta_n^{\mathcal{M}} = P \cdot \Theta_{n-1}^{\mathcal{M}} \qquad \text{for } n \geq 1$$

Reachability in DTMC

Can be computed via transient probability ⇒adapt $\mathcal{M}$ by making all states in B absorbing, then:

$$\underbrace{Pr^{\mathcal{M}}}_{\text{Reachability in } \mathcal{M}} \left(\Diamond^{\leq n} B\right) = \underbrace{\sum_{s' \in B} \Theta_n^{\mathcal{M}_B}(s')}_{\text{transient probability in } \mathcal{M}_B}$$

Constrained Reachability in DTMC

Can also be computed via transient probability ⇒adapt $\mathcal{M}$ by making all states in B and $S \setminus (C \,\mathcal{U}\, B)$ absorbing, then:

$$\underbrace{Pr^{\mathcal{M}}}_{\text{Reachability in } \mathcal{M}} \left(C \,\mathcal{U}^{\leq n} B\right) = \underbrace{\sum_{s' \in B} \Theta_n^{\mathcal{M}_{C,B}}(s')}_{\text{transient probability in } \mathcal{M}_{C,B}}$$

5.1 Probabilistic CTL (PCTL)

Probabilistic CTL (PCTL)

- Temporal Logic for describing properties of DTMC
- Extension to temporal logic CTL
- probabilistic operator $\mathbb{P}$ replaces universal and existential path quantification: $\mathbb{P}_J(\Phi)$

Syntax of PCTL

For $a \in AP, J \subseteq [0,1]$ an interval with rational bounds and natural n:

$$\Phi := true|a|\Phi \wedge \Phi|\neg\Phi|\mathbb{P}_J(\varphi)$$
$$\varphi := \circ\Phi|\Phi_1 \mathcal{U} \Phi_2|\Phi_1 \mathcal{U}^{\leq n}\Phi_2$$

- $s_0 s_1 s_2 \ldots \models \Phi \mathcal{U}^{\leq n} \Psi$ if Φ holds until Ψ holds within n steps
- $s \models \mathbb{P}_J(\varphi)$ if probability that paths starting in s fulfill φ lies in J

Derived Operators in PCTL

$$\diamond\Phi = true \mathcal{U} \Phi$$
$$\diamond^{\leq n}\Phi = true \mathcal{U}^{\leq n}\Phi$$
$$\mathbb{P}_{\leq p}(\Box\Phi) = \mathbb{P}_{\geq 1-p}(\diamond\neg\Phi)$$
$$\mathbb{P}_{]p,q]}(\Box^{\leq n}\Phi) = \mathbb{P}_{[1-q,1-p[}(\diamond^{\leq n}\neg\Phi)$$

PCTL Semantics

$\mathcal{M}, s \models \Phi$ iff formula Φ holds in s tate s of DTMC $\mathcal{M}$

$s \models a$	iff	$a \in L(s)$
$s \models \neg\Phi$	iff	$not(s \models \Phi)$
$s \models \Phi \wedge \Psi$	iff	$(s \models \Phi)$ and $(s \models \Psi)$
$s \models \mathbb{P}_J(\varphi)$	iff	$Pr(s \models \varphi) \in J$

where $Pr(s \models \varphi) = Pr_s\{\pi \in Paths(s)|\pi \models \varphi\}$
Semantics of path-formulas defined as in CTL

Measurability

For any PCTL path formula φ and state s of DTMC $\mathcal{M}$ the set $\{\pi \in Paths(s)|\pi \models \varphi\}$ is measurable.
Three cases:

- $\circ\Phi$: cylinder sets constructed form paths of length one
- $\Phi \mathcal{U}^{\leq n}\Psi$: (finite number of) cylinder sets from paths of lenght at most n
- $\Phi \mathcal{U} \Psi$: countable union of paths satisfying $\Phi \mathcal{U}^{\leq n}\Psi$ for all $n \geq 0$

PCTL Model Checking

Check whether a state s in a DTMC satisfies a PCTL formula:

- compure recursively the set $Sat(\Phi)$ of states that satisfy Φ
- check whether state s belongs to $Sat(\Phi)$

⇒ bottom-up traversal of the parse tree of Φ (like for CTL)

- for probabilistic operators:
  1. compute $Sat(\Phi)$
  2. compute probabilities

Probability and Next-Operator

- $s \models \mathbb{P}_J(\circ\Phi)$ iff $Prob(s, \circ\Phi) \in J$
- $Prob(s, \circ\Phi) \equiv \sum_{s' \in Sat(\Phi)} P(s, s')$ sum up probabilities to get into $Sat(\Phi)$
- Matrix-Vector Multiplication: $(Probs(s, \circ\Phi))_{s\in S} = P \cdot \iota_\Phi$ one step from init of Φ

Probability and Bounded Until Operator

$s \models \mathbb{P}_J(\Phi\,\mathcal{U}^{\leq h}\Psi)$ iff $Prob(s, \Phi\,\mathcal{U}^{\leq h}) \in J$ is the least solution of:

- 1 if $s \models \Psi$ 0 steps
- for $h > 0$ and $s \models \Phi \vee \neg\Psi$:

$$\sum_{s' \in S} P(s, s') \cdot Prob(s', \Phi\,\mathcal{U}^{\leq h-1}\Psi)$$

  iterate number of steps

- 0 otherwise fail

PCTL Model Checking

- Computation of probabilities $Prob(s, \Phi_1 \mathcal{U} \Phi_2)$ for all $s \in S$
- identify all states where probability is 1 or 0: ("Precomputation")
  - $S^{yes} = Sat(P_{\geq 1}[\Phi_1 \mathcal{U} \Phi_2])$
  - $S^{no} = Sat(P_{\leq 0}[\Phi_1 \mathcal{U} \Phi_2])$
- solve linear equation system for remaining states

$$Prob(s, \Phi_1 \mathcal{U} \Phi_2) = \begin{cases} 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{no} \\ \sum_{s' \in S} P(s, s') \cdot Prob(s', \Phi_1 \mathcal{U} \Phi_2) & \text{otherwise} \end{cases}$$

$\Rightarrow$reduction of linear equation system in $|S^?|$ unknowns instead of $|S|$, where $S^? = S \setminus (S^{yes} \cup S^{no})$

- Make all Ψ and all $\neg(\Phi \wedge \Psi)$-states absorbing in $\mathcal{M}$
  $\Rightarrow$: Check $\diamond^{=h}\Psi$ in obtained DTMC
  $\Rightarrow$Matrix-vector multiplication

Time Complexity

For finite DTMC $\mathcal{M}$ and PCTL formula Φ, $\mathcal{M} \models \Phi$ can be solved in time:

$$\mathcal{O}(poly(size(\mathcal{M})) \cdot n_{\max} \cdot |\Phi|)$$

- $n_{\max} = \max\{n | \Psi_1 \mathcal{U}^{\leq n} \Psi_2 \text{ occurs in } \Phi\}$
- $n_{\max} = 1$ if Φ does not contain the bounded until-operator
- $size(\mathcal{M}$ probably exponential
- Φ can be exponentially larger than LTL

5.2 Outlook

Continuous Time Markov Chain (CTMC)

- transitions are labelled with rates which are parameters of negative exponential distributions
- Continuous Stochastic Logic (CSL)
- Model Checking: reduce to DTMC via uniformization

Discrete Time Markov Decision Process (DTMDP)

- alternating non-deterministic and probabilistic choices
- Model Checking involves computing a scheduler that resolves nondeterminism

Counterexamples

- A set of offending paths with probability equal or greater than p.
- An informative counterexample is one which is small and has a high probability.

6 Binary Decision Diagrams and Symbolic Model Checking

Explicit Representation of TS might be to large, need something smaller.

Boolean Functions

boolean variable $x_1, x_2, \ldots, x, y, z$ ranging over values 0 and 1

Boolean Function

- function $f : \{0,1\}^n \to \{0,1\}$
  - $\overline{0} := 1$ and $\overline{1} := 0$
  - $x \cdot y := 1$ if x and y have value 1, otherwise $x \cdot y := 0$ and
  - $x + y := 0$ if x and y have value 0, otherwise $x + y := 1$ or
  - $x \oplus y := 1$ if exactly one of x and y equals 1

Alternative Representations

- function: $f(x, y) := \overline{x + y}$
- truth tables

x	y	$f(x,y)$
1	1	0
0	1	0
1	0	0
0	0	1

  Seemingly easy comparison (for identical variable ordering), satisfiability, validity, but exponential number of lines (variable combinations)
- boolean formula $\neg(p \vee q)$
  compact, but deciding, e.g., satisfiability is NP-complete

6.1 Binary Decision Tree

Binary Decision Tree

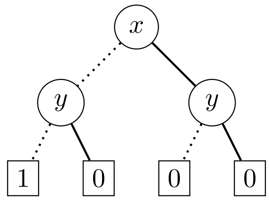
- non-terminal nodes labelled with boolea variables
- terminal nodes labelled with 0 or 1 unique boolean functions on variables in terminal nodes
  - dashed outgoing edge of node: $variable = 0$
  - solid outgoing edge of node $variable = 1$
  - function value: value of terminal node along path

⇒not a compact representation ⇒build something that is no longer a tree

6.2 Binary Decision Diagram (BDD)

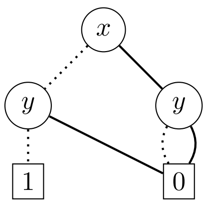
Binary Decision Diagram (BDD)

- a BDD is a finite Directed Acyclic Graph, such that: (all binary decision trees are BDDs)
  - it has a unique initial node
  - all terminal nodes are labelled with 0 or 1
  - all non-terminal nodes are labelled with boolean variables
  - each non-terminal node has exactly two outgoing edges labelled 0 (dashed line) or 1 (solid line)

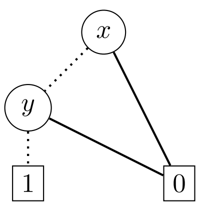


From Binary Decision Tree to BDD

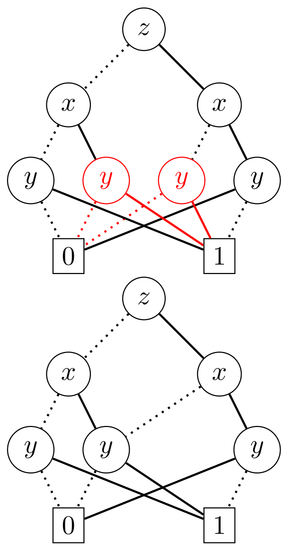
C1: removal of duplicate terminals



C2: removal of redundant tests



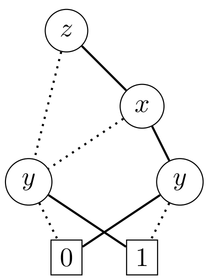
C3: removal of duplicate non-terminals: If two non-terminal nodes n and m are the roots of structurally identical sub-trees, then eliminate one of them and redirect all its incoming edges to the other node.



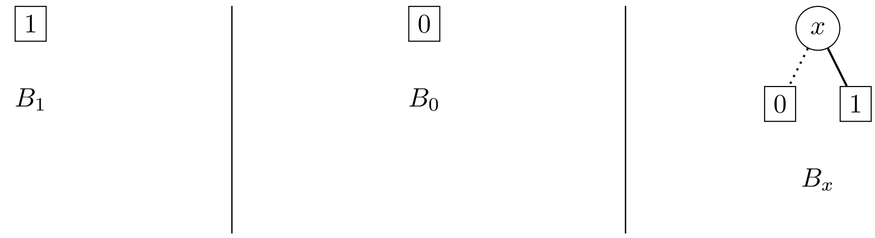
The boolean function is not very recognizable after reduction.

Reduced BDD

A BDD is reduced, when no further reduction $C1 - C3$ is possible:



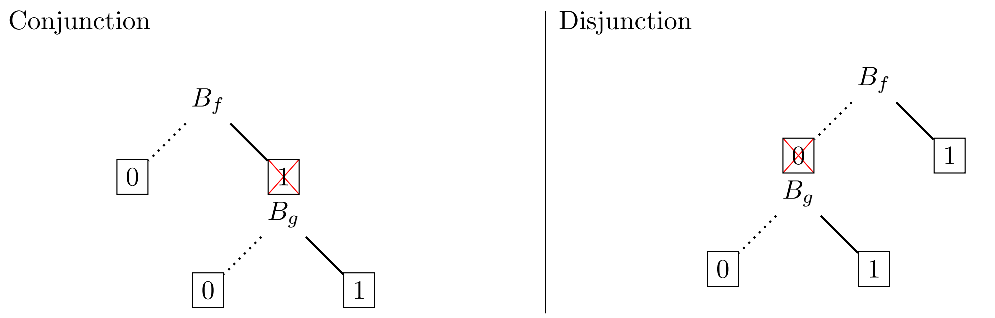
Special BDDs



Consistent Path

A Path through a BDD is **consistent** if every value for a variable is decided no more than once. When a variable is decided upon one value, it cannot be changed.
⇒prevent multiple variable occurrences ⇒impose and order on variables along every path ⇒Ordered BDDs

Conjunction and Disjunction



This works only for consistent paths!

Ordered BDD (OBDD)

- let $[x_1, \ldots, x_n]$ a list of variable names **without duplicates**
- let B BDD so that all its non-terminal nodes are members of this list
- B has the ordering $[x_1, \ldots, x_n]$ if for any path, the occurrence of x_i preceding the occurrence of x_j implies $i < j$
- B is then called an **ordered BDD** (OBBD)

Compatible Variable Ordering

- For OBDDs B_1 and B_2, if it does not happen that there is a variable x occurring before y in B_1 and after y in B_2, then we say that the variable orderings of B_1 and B_2 are **compatible**.
- If reduced OBDDs B_1 and B_2 describe the same boolean function, then they have identical structure.
  - equivalence checking: check for identical structure

- applying $C1 - C3$ to an OBDD until no further reduction ⇒leads to the same reduced OBDD, irrespective of the order they are applied
  ⇒**canonical form**

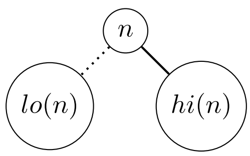
- OBDDs offer **canonical representation** for boolean functions
- OBDDs have worst case exponential size, computing optimal order is expensive, good heuristics usable

Benefits of Canonical Representation

- absence of redundant variables: if the value does not depend on some x it will not appear in reduced OBDD
- test for semantic equivalence of functions f and g
  1. determine compatible ordering of variables (with heuristics to make them good)
  2. reduce B_f and B_g
  3. check B_f and B_g for identical structure
- test for validity: reduced OBDD is B_1
- test for implication $f \implies g$:compute reduced OBDD for $f.g^{-1}$ and check whether it is B_0
- test for satisfiability: reduced OBDD is not B_0

reduce

- idea:
  - implements $C1 - C3$ in efficient fashion $C1$ is just a special case of $C3$
  - traverse BDD bottom-up, start with terminal nodes
  - for B with $[x_1, \ldots, x_l]$, B has at most $l + 1$ layers
  - during traversal: assign integer labels $id(n)$ to each node n
- algorithm
  - node n and m have the same label, is sub-BDD computes same boolean function
  - keep only one node per id



- if $id(lo(n)) = id(hi(n))$, then $id(n) := id(lo(n))$
  boolean test represented by n is redundant
- if another node m labeled with same variable x_i and if $id(lo(n)) = id(lo(m))$ and $id(hi(n)) = id(hi(m))$, then $id(n) := id(m)$ ⇒they compute the same boolean function
- if nothing of above applies, assign next unused integer

apply

- idea: implement the application of operations to boolean functions
  - examples: $+, ., \oplus$, complement $(f \oplus 1)$
  - $\mathit{apply}(op, B_f, B_g)$ computes reduced OBDD of $f \, op \, g$
  - algorithm operates recursively on structure of two OBDDs
    - let v be the variable highest in the variable order that occurs in B_f or B_g
    - solve problem separately for $v = 0$ and $v = 1$
    - at the leaves apply *op* directly
    - reduce result
- Restriction:
  - $f[0/x]$: boolean formula obtained by replacing all occurrences of x in f by 0
  - $f[1/x]$: boolean formula obtained by replacing all occurrences of x in f by 1
- perform recursion on boolean formulas by decomposing them into simpler ones: f on x is equivalent to $\overline{x} \cdot f[0/x] + x \cdot f[1/x]$ (≡ Shannon Expansion)
- use in *apply* $f \, op \, g = \overline{x_i} \cdot (f[0/x_i] \, op \, g[0/x_i]) + x_i \cdot (f[1/x_i] \, op \, g[1/x_i])$

Algorithm of apply:

- proceed from roots of B_f and B_g to construct the nodes of OBDD $B_{f \, op \, g}$
- let r_f and r_g the root nodes of B_f and B_g, respectively
- case:
  - both nodes are terminal nodes: compute $l_f \, op \, l_g$, if $0 \Rightarrow B_0$, if $1 \Rightarrow B_1$
  - both root nodes are x_i nodes: create x_i node n with
    - dashed line to $\mathit{apply}(op, lo(r_f), lo(r_g))$
    - solid line to $\mathit{apply}(op, hi(r_f), hi(r_g))$
  - r_f is an x_i node, but r_g is a terminal node or an x_j node with $j > i$
    - we know there is no x_i node in B_g because the two OBDDs have a compatible ordering
      $\Rightarrow g$ is indepenednt of x_i since $g \equiv g[0/x_i] \equiv g[1/x_i]$
      $\Rightarrow$create x_i node n with
      - dashed line to $\mathit{apply}(op, lo(r_f), r_g$
      - solid line to $\mathit{apply}(op, hi(r_f), r_g$
  - r_g is a non-terminal node, but r_f is a terminal node or an x_j node with $j > i$
    $\Rightarrow$symmetrically to above case
- call reduce on the result

Memoisation

- remember results of apply for future calls with identical arguments
  - more efficient
  - less reduction needed
- without memoization: apply is exponential in size of arguments
- with memoization: number of calls bounded by $2 \cdot |B_f| \cdot |B_g|$
- in praxis often even better

restrict

Purpose

- compute $f[0/x]$ and $f[1/x]$
- calls: $restrict(0, x, B_f)$ and $restrict(1, x, B_f)$

$\Rightarrow$ yields same variable ordering in result as in B_f

Procedure

- $restrict(0, x, B_f)$: for each node n labeled x
  - redirect incoming edges to $lo(n)$
  - remove n
  - call *reduce* on the result (iteratively)
- $restrict(1, x, B_f)$: same, but redirect to $hi(n)$

exists

useful to express relaxations on constraints for subset of variables:

- $\exists x.f := f[0/x] + f[1/x]$ ($\exists x.f$ can be true by x being 1 or 0)
  - $exists: \quad apply(+, restrict(0, x, B_f), restrict(1, x, B_f))$

Implementation Improvements:

- restricted nodes have same structure until x-nodes, compute application of $+$ to these sub-BDDs

OBDD Operations

Boolean formula f	OBDD B_f
0	B_0
1	B_1
x	B_x
$\overline{f}$	swap 0 and 1 nodes in B_f
$f + g$	$\texttt{apply}(+, B_f, B_g)$
$f \cdot g$	$\texttt{apply}(\cdot, B_f, B_g)$
$f \oplus g$	$\texttt{apply}(\oplus, B_f, B_g)$
$f[1/x]$	$\texttt{restrict}(1, x, b_f)$
$f[0/x]$	$\texttt{restrict}(0, x, b_f)$
$\exists x.f$	$\texttt{apply}(+, B_{f_{f[0/x]}}, B_{f_{g[1/x]}})$
$\forall x.f$	$\texttt{apply}(\cdot, B_{f_{f[0/x]}}, B_{f_{g[1/x]}})$

Algorithm	Input OBDD(s)	Output OBDD	Time Complexity
`reduce`	B	reduced B	$\mathcal{O}(\|B\| \cdot \log \|B\|)$
`apply`	B_f, B_g (reduced)	$B_{f\ op\ g}$ (reduced)	$\mathcal{O}(\|B_f\| \cdot \|B_g\|)$
`restrict`	B_f (reduced)	$B_{f[0/x]}$ or $B_{f[1/x]}$ (reduced)	$\mathcal{O}(\|B_f\| \cdot \log \|B_f\|)$
$\exists$	B_f (reduced)	$B_{\exists x_1.\exists x_2.\ldots.\exists x_n.f}$ (reduced)	NP-complete

Domain specific OBDDs exists, which may improve some operations, but they mostly use the canonicity property.

Symbolic Model Checking Algorithm

ϕ is $\top$	: **return** S
ϕ is $\bot$	: **return** $\varnothing$
ϕ is atomic	: **return** $\{s \in S \mid \phi \in L(s)\}$
ϕ is $\neg\phi_1$	: **return** $S - SAT(\phi_1)$
ϕ is $\phi_1 \wedge \phi_2$	: **return** $SAT(\phi_1) \cap SAT(\phi_2)$
ϕ is $\phi_1 \vee \phi_2$	: **return** $SAT(\phi_1) \cup SAT(\phi_2)$
ϕ is $\phi_1 \rightarrow \phi_2$	: **return** $SAT(\neg\phi_1 \vee \phi_2)$
ϕ is $\forall \circ \phi_1$	: **return** $SAT(\neg\exists \circ \neg\phi_1)$
ϕ is $\exists \circ \phi_1$	: **return** $SAT_{\exists\circ}(\phi_1)$
ϕ is $\forall(\phi_1 \mathcal{U} \phi_2)$	: **return** $SAT(\neg(\exists[\neg\phi_2 \mathcal{U} (\neg\phi_1 \wedge \neg\phi_2)] \vee \exists\Box\neg\phi_2))$
ϕ is $\exists(\phi_1 \mathcal{U} \phi_2)$	: **return** $SAT(\exists\mathcal{U}(\phi_1, \phi_2))$
ϕ is $\exists \diamond \phi_1$	: **return** $SAT(\exists(\top \mathcal{U} \phi_1))$
ϕ is $\exists\Box\phi_1$	: **return** $SAT(\neg\forall \diamond \neg\phi_1)$
ϕ is $\forall \diamond \phi_1$	: **return** $SAT_{\forall\diamond(\phi_1)}$
ϕ is $\forall\Box\phi_1$	: **return** $SAT(\neg\exists \diamond \neg\phi_1)$

Representing OBDDs

Transition System: $(S, Act, \rightarrow, I, AP, L)$

- characteristic function f_s for $L : S \rightarrow 2^{AP}$, ordering of OBDD is characteristic vector
- transition relation: two copies of characteristic vector: $s \rightarrow s' \Longrightarrow ((v_1, \ldots, v_n), (v'_1, \ldots, v'_n))$

Operations on OBDDs used in Model Checking Algorithm

- intersection: .
- union: +
- complementation: $\neg$
- $pre_\exists(Y) = \{s \in S \mid \exists s', (s \rightarrow s' \text{ and } s' \in Y)$
- $pre_\forall(Y) = \{s \in S \mid \forall s', (s \rightarrow s' \text{ implies } s' \in Y)$
  $pre_\forall(Y) = S - pre_\exists(S - Y)$

Algorithm

switch Φ **do**

a	:	$return\{s \in S \mid a \in L(s)\}$
...		...
$\exists \circ \Psi$	:	$return\{s \in S \mid Post(s) \cap Sat(\Phi) \neq \varnothing\}$
$\exists(\Phi_1 \,\mathcal{U}\, \Phi_2)$	:	compute smallest fixpoint
$\exists(\Phi)$	:	compute greatest fixpoint

endsw

Sat_{EX}

local var X, Y
$X := SAT(\Phi);$
$Y := pre_{\exists}(X);$
return Y

$SAT_{AF}(\Phi)$

local var X, Y
$X := S;$
$Y := SAT(\Phi);$
while $X \neq Y$ **do**
  $X := Y;$
  $Y := Y \cup pre_{\forall}(Y);$
end
return Y

$SAT_{EU}(\Phi, \Psi)$

$W := SAT(\Phi);$
$X := S;$
$Y := SAT(\Psi)$ **while** $X \neq Y$ **do**
  $X := Y;$
  $Y := Y \cup (W \cap pre_{\exists}(Y)$
end
return Y

OBDD synthesis

- so far: $(model) \rightarrow TS \rightarrow truth\,table \rightarrow OBDD \rightarrow$ `reduce`
- better: $(model) \rightarrow OBDD$ (reduced)