

Model Checking

René Thiemann

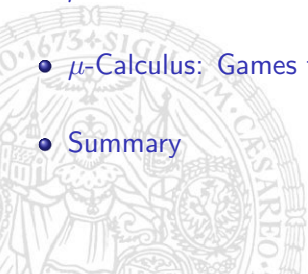
Institute of Computer Science
University of Innsbruck

SS 2009



Outline

- Overview
- Monotone Functions and Fixpoints
- μ -Calculus: Syntax, Semantic, and Naive Model-Checking Algorithm
- μ -Calculus: Alternation Depth and Improved Model-Checking Algorithm
- μ -Calculus: Games for Model-Checking
- Summary

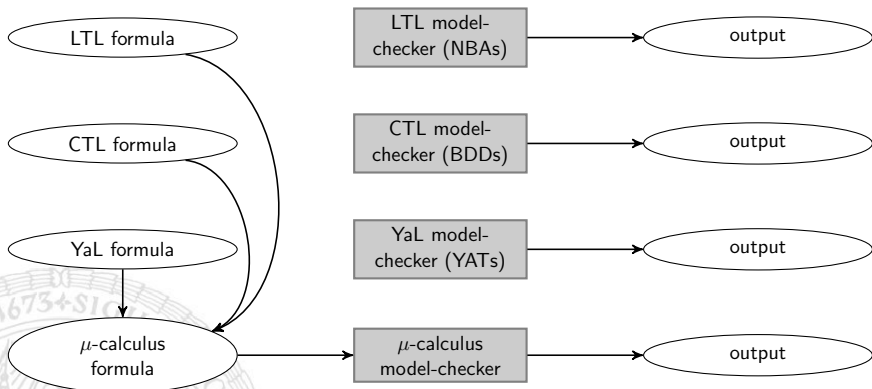


Outline

- Overview
- Monotone Functions and Fixpoints
- μ -Calculus: Syntax, Semantic, and Naive Model-Checking Algorithm
- μ -Calculus: Alternation Depth and Improved Model-Checking Algorithm
- μ -Calculus: Games for Model-Checking
- Summary



Model-Checking for Different Logics



μ -calculus

- Very expressive
- ⇒ many logics can be translated into μ -calculus
- Efficient (parallel) model-checking algorithms
 - Based upon fixpoints
 - Not very human-readable
- ⇒ use μ -calculus mainly for model-checking of other logics and not for direct specification

Outline

- Overview
- **Monotone Functions and Fixpoints**
- μ -Calculus: Syntax, Semantic, and Naive Model-Checking Algorithm
- μ -Calculus: Alternation Depth and Improved Model-Checking Algorithm
- μ -Calculus: Games for Model-Checking
- Summary



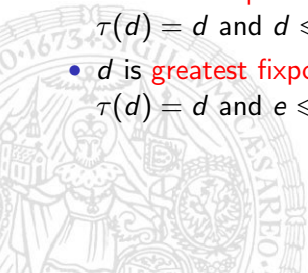
Fixpoints

Let $\tau : D \rightarrow D$ be a function over some domain D

- $d \in D$ is **fixpoint** of τ iff $\tau(d) = d$
- Not every function has a fixpoint
- Some functions have more than one fixpoint

Let D be equipped with a partial order \leq

- d is **least fixpoint** of τ ($lfp(\tau)$) iff
 $\tau(d) = d$ and $d \leq e$ for all other fixpoints e of τ
- d is **greatest fixpoint** of τ ($gfp(\tau)$) iff
 $\tau(d) = d$ and $e \leq d$ for all other fixpoints e of τ



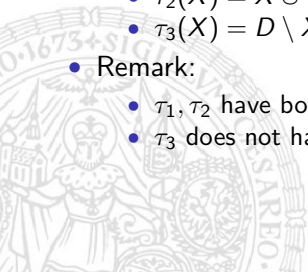
Monotone Functions

Let D be a domain with partial order \leq .

- A function $\tau : D \rightarrow D$ is **monotone** iff $d \leq e \Rightarrow \tau(d) \leq \tau(e)$

Examples:

- x^2 is monotone over the naturals, but not over the integers
- For $D = 2^S$, $\leq = \subseteq$, and arbitrary $Y \in D$, i.e., $Y \subseteq S$:
 - $\tau_1(X) = X \cap Y$ is monotone
 - $\tau_2(X) = X \cup Y$ is monotone
 - $\tau_3(X) = D \setminus X$ is not monotone
- Remark:
 - τ_1, τ_2 have both least and greatest fixpoints
 - τ_3 does not have a single fixpoint if $S \neq \emptyset$

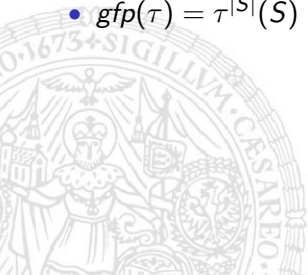


Existence and Computation of Fixpoints

Theorem (Knaster, Tarski)

Let S be a *finite* set, let $D = 2^S$ be ordered by \subseteq , let $\tau : D \rightarrow D$.
If τ is *monotone* then

- $\text{lfp}(\tau) = \tau^{|S|}(\emptyset)$
- $\text{gfp}(\tau) = \tau^{|S|}(S)$

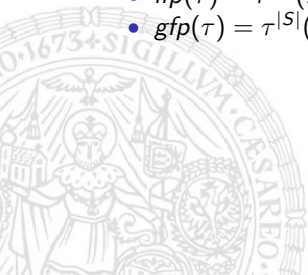


Proof



Summary of Fixpoints

- X is fixpoint of τ iff $\tau(X) = X$
- Function $\tau : 2^S \rightarrow 2^S$ is monotone iff $X \subseteq Y$ implies $\tau(X) \subseteq \tau(Y)$
(union and intersection are monotone, complement is not monotone)
- If S is finite and τ monotone then τ has least and greatest fixpoint:
 - $lfp(\tau) = \tau^{|S|}(\emptyset)$
 - $gfp(\tau) = \tau^{|S|}(S)$



Outline

- Overview
- Monotone Functions and Fixpoints
- μ -Calculus: Syntax, Semantic, and Naive Model-Checking Algorithm
- μ -Calculus: Alternation Depth and Improved Model-Checking Algorithm
- μ -Calculus: Games for Model-Checking
- Summary



A Small Change in Transition Systems

Transition systems may now have **labeled edges**:

A **transition system** TS is a tuple

$$(S, Act, \rightarrow, I, AP, L)$$

where

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

Example



μ -Calculus

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

Let $\mathcal{V} = \{x, y, \dots\}$ be a set of variables (ranging over **sets of states**).

Let $\alpha : \mathcal{V} \rightarrow 2^S$ be a **variable assignment**

Definition (μ -Calculus Semantic)

For each L_μ -formula and variable assignment define the **satisfiability set** as

- $\llbracket p \rrbracket_\alpha = \{s \mid p \in L(s)\}$
- $\llbracket \varphi \wedge \psi \rrbracket_\alpha = \llbracket \varphi \rrbracket_\alpha \cap \llbracket \psi \rrbracket_\alpha$
- $\llbracket \varphi \vee \psi \rrbracket_\alpha = \llbracket \varphi \rrbracket_\alpha \cup \llbracket \psi \rrbracket_\alpha$
- $\llbracket \neg\psi \rrbracket_\alpha = S \setminus \llbracket \psi \rrbracket_\alpha$
- $\llbracket \langle a \rangle \varphi \rrbracket_\alpha = \{s \mid \text{there is } s \xrightarrow{a} t \text{ and } t \in \llbracket \varphi \rrbracket_\alpha\}$
- $\llbracket [a] \varphi \rrbracket_\alpha = \{s \mid \text{whenever } s \xrightarrow{a} t \text{ then } t \in \llbracket \varphi \rrbracket_\alpha\}$
- $\llbracket x \rrbracket_\alpha = \alpha(x)$
- $\llbracket \mu x. \varphi \rrbracket_\alpha = \text{lfp}(\tau)$ where $\tau : 2^S \rightarrow 2^S$, $\tau(X) = \llbracket \varphi \rrbracket_{\alpha[x:=X]}$
- $\llbracket \nu x. \varphi \rrbracket_\alpha = \text{gfp}(\tau)$ where $\tau : 2^S \rightarrow 2^S$, $\tau(X) = \llbracket \varphi \rrbracket_{\alpha[x:=X]}$

A Note on Well-Definedness

- Example:

For $\mu X. \neg X$ obtain $\tau(X) = S \setminus X \Rightarrow$ no *lfp* \Rightarrow no $\llbracket \mu X. \neg X \rrbracket_\alpha$

However, $\mu X. \neg X$ is not a L_μ -formula

(X occurs under an odd number of negations)

- Semantic is well-defined iff both

- *lfp*(τ) and
- *gfp*(τ)

exist where τ is defined as $\tau(X) = \llbracket \varphi \rrbracket_{\alpha[x:=X]}$

- Requirement of **even number of negations** ensures that τ is **monotone!**
- \Rightarrow Knaster & Tarski ensures that both *lfp*(τ) and *gfp*(τ) exist
- \Rightarrow Semantic is well-defined

Model-Checking for the μ -Calculus

- A L_μ -formula is **closed** iff it does not contain free variables
- \Rightarrow For closed formulas α is not required
- \Rightarrow Define model relation for closed formulas:

$$TS \models \varphi \quad \text{iff} \quad I \subseteq \llbracket \varphi \rrbracket$$

Naive Model-Checking Algorithm:

- Just compute $\llbracket \varphi \rrbracket$ by directly applying the definition of the semantics in a top-down way
- To compute fixpoints use Knaster & Tarski
 - $lfp(\tau) = \tau^{|S|}(\emptyset)$
 - $gfp(\tau) = \tau^{|S|}(S)$
- **Model-Checking for μ -calculus boils down to simple set operations**

Naive MC-Algorithm for the μ -Calculus

Input: A closed L_μ -formula φ and
a transition system $TS = (S, Act, \rightarrow, I, AP, L)$

Output: The boolean value of $TS \models \varphi$

Global variable: $\alpha : \mathcal{V}(\varphi) \rightarrow 2^S$

function model_check(φ)

return $I \subseteq \text{sem}(\varphi)$

procedure reset(x)

if x is μ -variable **then** $\alpha(x) := \emptyset$ **else** $\alpha(x) := S$

Naive MC-Algorithm for the μ -Calculus

function $\text{sem}(\varphi)$

case φ **of**

x : **return** $\alpha(x)$

p : **return** $\{s \mid p \in L(s)\}$

$\neg\psi$: **return** $S \setminus \text{sem}(\psi)$

$\psi_1 \wedge \psi_2$: **return** $\text{sem}(\psi_1) \cap \text{sem}(\psi_2)$

$\psi_1 \vee \psi_2$: **return** $\text{sem}(\psi_1) \cup \text{sem}(\psi_2)$

$\langle a \rangle \psi$: **return** $\{s \mid \exists s \xrightarrow{a} t, t \in \text{sem}(\psi)\}$

$[a] \psi$: **return** $\{s \mid \forall s \xrightarrow{a} t : t \in \text{sem}(\psi)\}$

$Qx.\psi$:

reset(x)

while true **do**

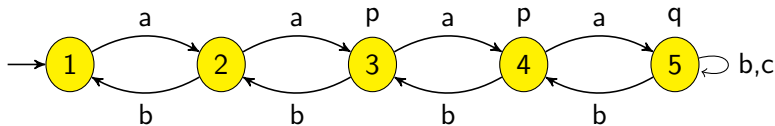
$U := \alpha(x)$

$V := \text{sem}(\psi)$

if $U = V$ **then return** U **else** $\alpha(x) := V$

Example

Computing $\llbracket \varphi \rrbracket$ for $\varphi = \mu x.[b]\nu y.x \vee \langle a \rangle y$ and the following TS.



	$\llbracket \varphi \rrbracket$	$\alpha(x)$	$\llbracket [b]\nu y.x \vee \langle a \rangle y \rrbracket_\alpha$	$\llbracket \nu y.x \vee \langle a \rangle y \rrbracket_\alpha$	$\alpha(y)$	$\llbracket x \vee \langle a \rangle y \rrbracket_\alpha$	$\llbracket \langle a \rangle y \rrbracket_\alpha$
1	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	✓	
5							

Hence, $\llbracket \varphi \rrbracket = \{1, 2, 3, 4\}$ and $TS \models \varphi$.

Complexity of naive algorithm:

$$\mathcal{O}((|TS| \cdot |\varphi|)^{|\nu(\varphi)|})$$

Encoding of Logics into μ -Calculus

Theorem

Every CTL-formula can be translated into a closed L_μ -formula.

Proof.

W.l.o.g. all transitions are labeled by “a” (CTL cannot distinguish these)

- $AX \varphi \rightsquigarrow [a]\varphi$
- $EX \varphi \rightsquigarrow \langle a \rangle \varphi$
- $A \varphi U \psi \rightsquigarrow \mu x. \psi \vee (\varphi \wedge [a]x)$
- $E \varphi U \psi \rightsquigarrow \mu x. \psi \vee (\varphi \wedge \langle a \rangle x)$
- $AG \varphi \rightsquigarrow \nu x. \varphi \wedge [a]x$

Problem: Resulting complexity is **exponential**, although CTL-model checking has **linear** complexity.

Outline

- Overview
- Monotone Functions and Fixpoints
- μ -Calculus: Syntax, Semantic, and Naive Model-Checking Algorithm
- **μ -Calculus: Alternation Depth and Improved Model-Checking Algorithm**
- μ -Calculus: Games for Model-Checking
- Summary

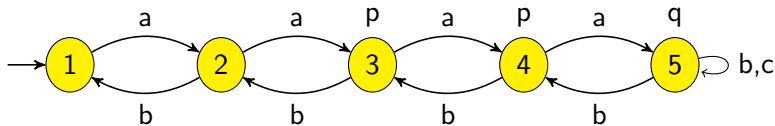


Example

Computing $\llbracket \mu x. \varphi_x \rrbracket$ for the following TS where

$$\varphi_x = q \vee \langle a \rangle \mu y. \varphi_y$$

$$\varphi_y = p \wedge \langle a \rangle (x \vee y)$$



	$\llbracket \mu x. \varphi_x \rrbracket$	$\alpha(x)$	$\llbracket \varphi_x \rrbracket$	$\llbracket q \rrbracket$	$\llbracket \langle a \rangle \mu y. \varphi_y \rrbracket$	$\llbracket \mu y. \varphi_y \rrbracket$	$\alpha(y)$	$\llbracket \varphi_y \rrbracket$	$\llbracket p \rrbracket$	$\llbracket \langle a \rangle (x \vee y) \rrbracket$	$\llbracket x \vee y \rrbracket$
1											
2											
3											
4											
5											

Complexity of improved algorithm:

$$\mathcal{O}((|TS| \cdot |\varphi|)^?)$$

Positive Normal Form

L_μ -formula φ is in **positive normal form** (PNF) iff every variable is bound at most once and “ \neg ” only occurs before propositions p

Theorem

Every closed L_μ -formula can be translated into positive normal form.

Proof.

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg(\neg\varphi) \rightsquigarrow \varphi$
- $\neg\langle a \rangle\varphi \rightsquigarrow [a]\neg\varphi$
- $\neg[a]\varphi \rightsquigarrow \langle a \rangle\neg\varphi$
- $\neg\mu x.\varphi \rightsquigarrow \nu x.\neg\varphi[x/\neg x]$
- $\neg\nu x.\varphi \rightsquigarrow \mu x.\neg\varphi[x/\neg x]$
- $\neg x$ does not occur due to “even number of negations”-condition

Example



Improved MC-Algorithm for the μ -Calculus [Emerson,Lei]

Input: A closed L_μ -formula φ in PNF and
a transition system $TS = (S, I, \dots, L)$

Output: The boolean value of $TS \models \varphi$

Global variables: $\alpha : \mathcal{V}(\varphi) \rightarrow 2^S$

Valid $\subseteq \mathcal{V}(\varphi)$ // $x \in \text{Valid}$ implies $\alpha(x) = \llbracket Qx.\varphi_x \rrbracket_\alpha$

function model_check(φ)

Valid := \emptyset

for all $x \in \mathcal{V}(\varphi)$ **do** reset(x)

return $I \subseteq \text{sem}(\varphi)$

procedure reset(x)

if x is μ -variable **then** $\alpha(x) := \emptyset$ **else** $\alpha(x) := S$

Improved MC-Algorithm for the μ -Calculus [Emerson,Lei]

function $\text{sem}(\varphi)$

case φ **of**

x : **return** $\alpha(x)$

p : **return** $\{s \mid p \in L(s)\}$

$\neg p$: **return** $\{s \mid p \notin L(s)\}$

$\psi_1 \wedge \psi_2$: **return** $\text{sem}(\psi_1) \cap \text{sem}(\psi_2)$

$\psi_1 \vee \psi_2$: **return** $\text{sem}(\psi_1) \cup \text{sem}(\psi_2)$

$\langle a \rangle \psi$: **return** $\{s \mid \exists s \xrightarrow{a} t, t \in \text{sem}(\psi)\}$

$[a] \psi$: **return** $\{s \mid \forall s \xrightarrow{a} t : t \in \text{sem}(\psi)\}$

$Qx.\psi$: **if** $x \in \text{Valid}$ **then return** $\alpha(x)$ **else while true do**

$U := \alpha(x); V := \text{sem}(\psi)$

if $U = V$ **then**

$\text{Valid} := \text{Valid} \cup \{x\}$; **return** U

else

$\alpha(x) := V$; **touch**($Qx.\psi$)

Improved MC-Algorithm for the μ -Calculus [Emerson,Lei]

procedure touch($Q'x.\varphi_x$)

Valid := Valid \setminus $\{y \mid Qy.\varphi_y \in \text{Sub}(\varphi_x), x \in \mathcal{FV}(\varphi_y)\}$

Reset := $\{y \mid Qy.\varphi_y \in \text{Sub}(\varphi_x), x \in \mathcal{FV}(\varphi_y), Q \neq Q'\}$

while $z \in \{z \mid \exists y \in \text{Reset}, Qz.\varphi_z \in \text{Sub}(\varphi_y), \mathcal{FV}(\varphi_z) \cap \text{Reset} \neq \emptyset\}$ **do**

 Reset := Reset \cup $\{z\}$

for all $y \in \text{Reset}$ **do** reset(y)

Valid := Valid \setminus Reset

- $\mathcal{FV}(\varphi)$ is the set of *free variables* of φ
- $\text{Sub}(\varphi)$ is the set of sub-formulas of φ
- φ_x is the unique formula which is the argument of “ Qx .”

Illustration of touch



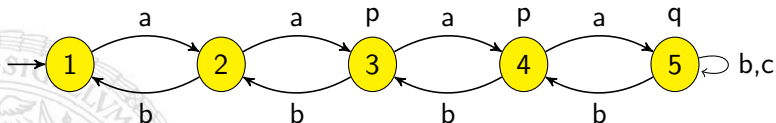
Example

Computing $\llbracket \nu z. \varphi_z \rrbracket$ for the following TS where

$$\varphi_z = z \wedge \langle a \rangle \mu x. \varphi_x$$

$$\varphi_x = q \vee \langle a \rangle \mu y. \varphi_y$$

$$\varphi_y = p \wedge \langle a \rangle (x \vee y)$$



Complexity of the Algorithm

Definition (Alternation Depth)

Variable x **depends on** y in φ ($x \prec_{\varphi} y$) iff φ contains subformula $Q x.\psi$ and y is a **free variable** of ψ .

The **alternation depth** of a formula φ in PNF is defined as $ad(\varphi) = n$ where n is the largest number such that $x_1 \prec_{\varphi} \dots \prec_{\varphi} x_n$ and the type of x_i is different to the type of x_{i+1} for every $i < n$.

A formula with $ad(\varphi) \leq 1$ is called **alternation free**.

Theorem

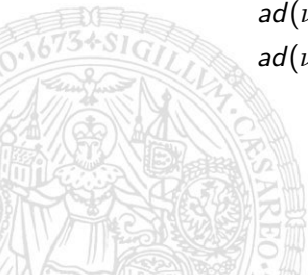
The algorithm of Emerson and Lei is sound and has complexity

$$\mathcal{O}((|TS| \cdot |\varphi|)^{ad(\varphi)}).$$

Efficient implementations available using **binary decision diagrams** (BDDs)

Example

$$\begin{aligned}
 & ad(q \vee \langle a \rangle p) = \\
 & ad(\mu x. q \vee \langle a \rangle (\mu y. p \wedge \langle a \rangle (x \vee y))) = \\
 & ad(\nu z. z \wedge \langle a \rangle (\mu x. q \vee \langle a \rangle \mu y. p \wedge \langle a \rangle (x \vee y))) = \\
 & ad(\mu x. [b] \nu y. x \vee \langle a \rangle y) = \\
 & ad(\nu x. \mu y. y \wedge x \wedge (\nu z. z) \wedge \nu u. (u \wedge x)) = \\
 & ad(\nu x. \mu y. y \wedge x \wedge (\nu z. z) \wedge \nu u. (u \wedge y)) =
 \end{aligned}$$



Proof of Soundness

One crucial point is to use a stronger variant of Knaster-Tarski:

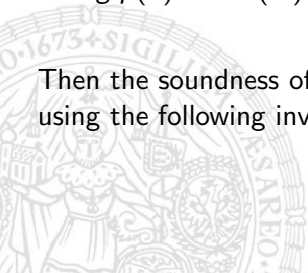
Theorem (Variant of Knaster-Tarski)

Let S be a finite set, let $D = 2^S$ be ordered by \subseteq , let $\tau : D \rightarrow D$.

If τ is *monotone* then

- $\text{lfp}(\tau) = \tau^{|S|}(T)$ if $T \subseteq \tau^k(\emptyset)$ for some k
- $\text{gfp}(\tau) = \tau^{|S|}(T)$ if $T \supseteq \tau^k(S)$ for some k

Then the soundness of the algorithm can be proven by induction on φ using the following invariants:



Encoding of Logics into μ -Calculus

Theorem

Every CTL-formula can be translated into an *alternation free* L_μ -formula.

Proof.

- ...
- $E\varphi U\psi \rightsquigarrow \mu x. \psi \vee (\varphi \wedge \langle a \rangle x)$
- $AG\varphi \rightsquigarrow \nu x. \varphi \wedge [a]x$

Resulting formula has only trivial dependencies $x \prec x$. ■

\Rightarrow CTL-model checking via μ -calculus has linear and hence, optimal complexity

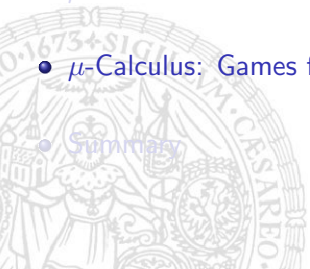
Theorem

Every CTL*-formula can be translated into a L_μ -formula with alternation depth 2.

Outline

- Overview
- Monotone Functions and Fixpoints
- μ -Calculus: Syntax, Semantic, and Naive Model-Checking Algorithm
- μ -Calculus: Alternation Depth and Improved Model-Checking Algorithm
- μ -Calculus: Games for Model-Checking

- Summary



Overview

Current approach:

- Formula $\rightsquigarrow L_\mu$ -formula \rightsquigarrow PNF \rightsquigarrow Emerson Lei MC (BDDs)
- Global approach - whole transition system required and processed

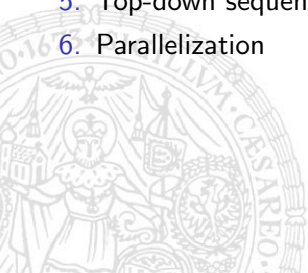
Upcoming approach:

- Formula $\rightsquigarrow L_\mu$ -formula \rightsquigarrow PNF \rightsquigarrow MC based on Games
- Sequential algorithm for alternation free formulas
- Local approach - only parts of transition system required, on-the-fly
- Parallel algorithm for alternation free formulas
- (Not shown: algorithm for formulas with alternation depth 2)

Obtain efficient model-checker for μ -calculus, CTL, CTL*, ...

Overview of Games for Model-Checking

1. PNF \rightsquigarrow graph
2. Graph \times transition system \rightsquigarrow game graph
3. Model-checking = determining winner of game
4. Bottom-up sequential algorithm to determine winner
5. Top-down sequential algorithm to determine winner
6. Parallelization



1. From closed L_μ -formula in PNF to graph

- First write down a given formula φ as a tree where
 - Each formula has as successors its direct subformulas
 - $\neg p$ is seen as an atomic formula
 - Then obtain a graph by adding edges from each x to $Qx.\varphi_x$
- \Rightarrow Nodes of the graph are $Sub(\varphi)$ where **duplicates are allowed**
 (e.g., node $p \wedge p$ has two successors p , each p being a separate node)

φ alternation free: Partition graph into components Q_1, \dots, Q_n such that

- Each Q_i has only edges to $Q_i \cup Q_{i+1} \cup \dots \cup Q_n$
- Each Q_i contains only μ -formulas or only ν -formulas
 (then we call Q_i μ -component or ν -component)

Algorithm: Perform SCC decomposition,
 then merge singleton nodes into adjoint component

Example



2. PNF + Transition System = Game Graph

Two player games:

- Players \forall belard and \exists loise
- **Game graph** is directed graph where nodes are called **configurations**
The set of configurations C is partitioned into $C = C_{\forall\text{belard}} \uplus C_{\exists\text{loise}}$
- A **play** is infinite or maximal finite sequence of configurations

$$c_0 \hookrightarrow c_1 \hookrightarrow c_2 \hookrightarrow \dots$$

If $c_i \in C_{\forall\text{belard}}$ then \forall belard can choose c_{i+1} , same for \exists loise

Here:

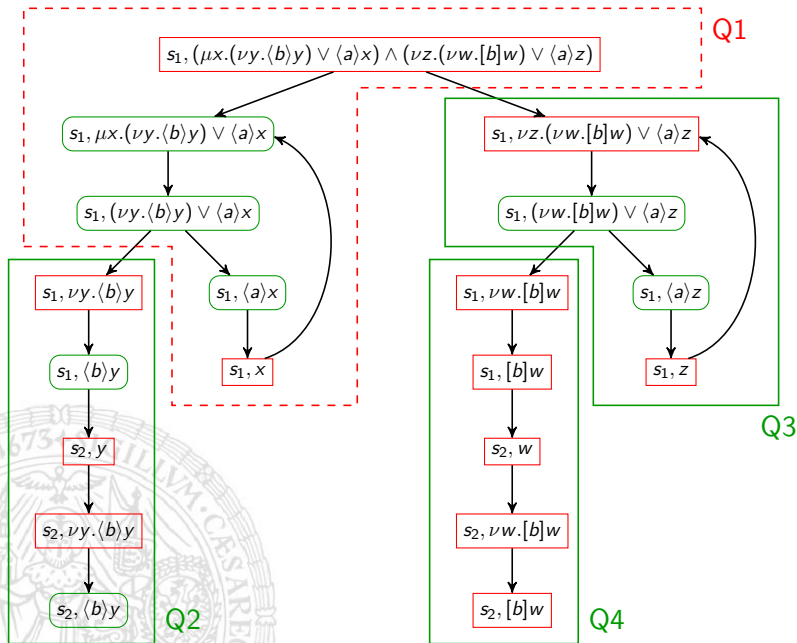
- Game graph for $TS = (S, Act, \rightarrow, I = \{s_0\}, AP, L)$ and φ has configurations $C = S \times Sub(\varphi)$, initial configuration $c_0 = (s_0, \varphi)$ (similar to tabular of Emerson Lei algorithm, but here **only reachable part** has to be computed! \Rightarrow **on-the-fly** algorithm)
- \forall belard wants to show $s \notin \llbracket \psi \rrbracket$, \exists loise wants to show $s \in \llbracket \psi \rrbracket$

Game Graph

The **edges** of the game graph are determined as follows:

1. If $c = (s, \psi_1 \wedge \psi_2)$ then \forall belard can move to (s, ψ_1) or (s, ψ_2)
2. If $c = (s, [a]\psi)$ then \forall belard can move to (t, ψ) for some $s \xrightarrow{a} t$
3. If $c = (s, \nu x.\psi)$ then the successor is (s, ψ)
4. If $c = (s, x)$ then the successor is $(s, Qx.\varphi_x)$
5. If $c = (s, \psi_1 \vee \psi_2)$ then \exists loise can move to (s, ψ_1) or (s, ψ_2)
6. If $c = (s, \langle a \rangle \psi)$ then \exists loise can move to (t, ψ) for some $s \xrightarrow{a} t$
7. If $c = (s, \mu x.\psi)$ then the successor is (s, ψ)
8. If $c = (s, p)$ or $c = (s, \neg p)$ then the play is finished

Configurations in cases **1-4 belong to \forall belard**, cases **5-8 belong to \exists loise**
(in cases 3,4,7,8 this is not important, as there is no choice)



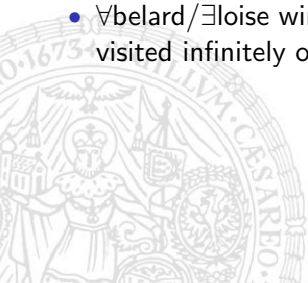
Playing a Game

Given a play $c_0 \hookrightarrow c_1 \hookrightarrow \dots$ there are two possibilities:

- If play is finite, $c_n = (s, \psi)$ is last configuration then \forall belard wins iff
 - $\psi = \langle a \rangle \chi$ (since there is no successor by maximality of play)
 - $\psi = p$ and $p \notin L(s)$ or $\psi = \neg p$ and $p \in L(s)$

In all other finite plays \exists loise wins

- \forall belard/ \exists loise wins an infinite play iff the maximal subformula that is visited infinitely often is a μ/ν -formula



Strategies

A **strategy** Str of a player is a function which takes an initial part of a play which ends in a configuration which belongs to that player and returns the configuration where the player wants to move to. Formally:

$Str : C^* C_{player} \rightarrow C \cup \{\perp\}$ such that for all $c_0 \dots c_n \in C^* C_{player}$:

- If $Str(c_0 \dots c_n) \in C$ then $c_n \hookrightarrow Str(c_0 \dots c_n)$ is allowed move
- If $Str(c_0 \dots c_n) = \perp$ then c_n has no successor

Note that a strategy of player **uniquely determines all moves of that player** for any given play; we then speak of a **Str -play**

A strategy Str of a player is a **winning strategy** if for each Str -play that player is the winner

A strategy Str is **positional**, if Str only considers the last configuration, i.e., $Str : C_{player} \rightarrow C \cup \{\perp\}$

Example Strategies



3. Model Checking by Games

Theorem (Stirling)

For each formula φ and each transition system TS :

- if $TS \models \varphi$ then \exists loise has a positional winning strategy
- if $TS \not\models \varphi$ then \forall belard has a positional winning strategy

Algorithmic approach for model checking

- Color configuration of game-graph by green/red if \exists loise/ \forall belard has winning strategy when starting from that configuration
- $TS \models \varphi$ iff color of c_0 is green

4. Bottom-Up Coloring

We only consider alternation free formulas

Remember: Then graph for formula (and also game-graph) can be partitioned into components C_1, \dots, C_n such that

- all components have only μ -formulas or only ν -formulas
- all edges of C_i lead to $C_i \cup \dots \cup C_n$

Thus, every play starting in C_i will either

1. leave C_i and continue in some C_{i+k} , $k > 0$
2. reach a terminal configuration in C_i
(terminal configuration = configuration without successors)
3. stay in C_i forever

In case 1, the winner can be determined by the color of the configuration that is visited first in C_{i+k}

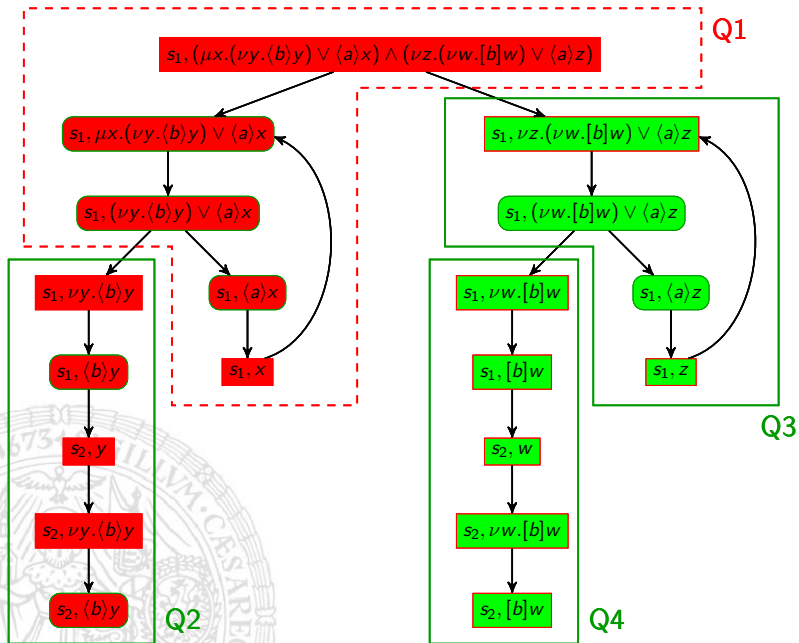
In case 2, the terminal configuration specifies the winner

In case 3, \forall belard/ \exists loise wins iff C_i is μ/ν -component

4. Bottom-Up Coloring

Hence, perform the following coloring process:

- every terminal configuration c is colored by red if the play c is won by \forall belard and by green, otherwise
- colors are propagated bottom-up:
 - let c be configuration with successors c_1, \dots, c_m with $m > 0$
 - $c \in C_{\exists\text{loise}}$, some c_i green \rightsquigarrow color c green
 - $c \in C_{\exists\text{loise}}$, all c_i red \rightsquigarrow color c red
 - $c \in C_{\forall\text{belard}}$, some c_i red \rightsquigarrow color c red
 - $c \in C_{\forall\text{belard}}$, all c_i green \rightsquigarrow color c green
- If all colors of C_{i+1}, \dots, C_n are determined and no propagation is possible for configurations of C_i then
 - color all white nodes of C_i by red if C_i is μ -component
 - color all white nodes of C_i by green if C_i is ν -component



4. Bottom-Up Coloring

Lemma

Once a configuration has a color, it will never be changed.

Theorem (Bollig, Leucker, Weber)

The bottom-up coloring process terminates and c_0 has color green/red iff $\exists \text{loise} \forall \text{belard}$ has a positional winning strategy.

Further properties of the bottom-up coloring algorithm:

- Linear complexity (optimal)
- Every configuration is considered (half on-the-fly)

5. Top-Down Coloring

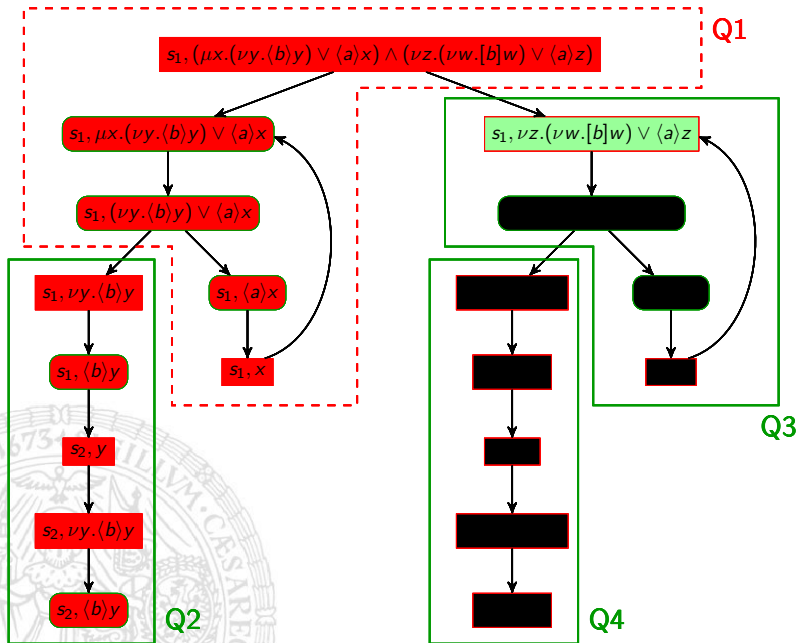
Overview:

- Directly start with top component C_1
- Let C_1 be μ -component (ν -components are treated dually)
 - If play ends in C_1 then winner can be determined
 - If play stays in C_1 then \exists loise loses
 - \Rightarrow Goal of \exists loise is to leave C_1 (or reach green terminal configuration)
 - Idea: Make successors of C_1 outside C_1 attractive
 - \Rightarrow color these nodes with **light-green** (optimistic assumption)
 - Then propagate colors in C_1
- Result after coloring configurations in C_1
 - configurations with full-color have correct color (as in bottom-up)
 - configurations with white color become red (as in bottom-up)
 - if initial configuration has full-color then done
 - otherwise initial configuration has light-green color:
then remove all light-green colors from C_1 , pick some successor component C_k of C_1 with assumed light-green initial configuration and determine the (full) color of C_k 's initial configurations; afterwards color C_1 again, ...

5. Top-Down Coloring

Details on coloring process:

- every terminal configuration obtains full color (as in bottom-up)
- colors are propagated similar to bottom-up:
let c be configuration with successors c_1, \dots, c_m with $m > 0$
 - $c \in C_{\exists \text{loise}}$, some c_i green \rightsquigarrow color c green
 - $c \in C_{\exists \text{loise}}$, some c_i light-green, no c_j green \rightsquigarrow color c light-green
 - $c \in C_{\exists \text{loise}}$, all c_i red \rightsquigarrow color c red
 - $c \in C_{\exists \text{loise}}$, all c_i red or light-red, some c_j light-red \rightsquigarrow color c light-red
 - $c \in C_{\forall \text{belard}}$, some c_i red \rightsquigarrow color c red
 - $c \in C_{\forall \text{belard}}$, some c_i light-red, no c_j red \rightsquigarrow color c light-red
 - $c \in C_{\forall \text{belard}}$, all c_i green \rightsquigarrow color c green
 - $c \in C_{\forall \text{belard}}$, all c_i green or light-green, some c_j light-green \rightsquigarrow color c light-green



5. Top-Down Coloring

Lemma

When coloring a component C_i a configuration can only change from white to colored, and from each light-color to the corresponding full-color.

Theorem (Bollig, Leucker, Weber)

The top-down coloring process terminates and c_0 has color green/red iff $\exists \text{loise} \forall \text{belard}$ has a positional winning strategy.

Further properties of the top-down coloring:

- Full on-the-fly algorithm (optimal)
- Quadratic complexity (sub-optimal)

6. Parallelization

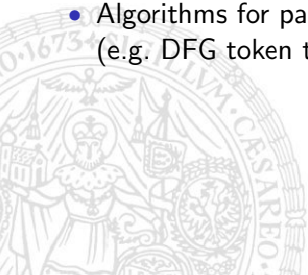
Let us consider n machines (PCs in a cluster, etc.):

- Game graph distribution:
 - Size of game graph unknown when starting algorithm
 - Assume hash function f
 - Machine i stores configuration c iff $f(c) \bmod n = i$
(additionally successors and predecessors of c are stored on machine i)
- Game graph construction:
 - Use breadth-first search (easy to parallelize with above distribution)
- Coloring (both bottom-up and top-down):
 - Process components sequentially, but color each component in parallel
 - as soon as terminal state is detected during game graph construction start backwards coloring process (in parallel)
 - if coloring of component is done, recolor white and light-color configurations (in parallel)

6. Parallelization

Some notes on parallelization:

- **Cycle detection** is inherently sequential
(but required for model checking via NBAs)
- Coloring algorithm does not need cycle detection,
but **parallel termination detection**
- Algorithms for parallel termination detection available
(e.g. DFG token termination algorithm of Dijkstra, Feijen, Gasteren)



Outline

- Overview
- Monotone Functions and Fixpoints
- μ -Calculus: Syntax, Semantic, and Naive Model-Checking Algorithm
- μ -Calculus: Alternation Depth and Improved Model-Checking Algorithm
- μ -Calculus: Games for Model-Checking
- **Summary**



Summary

- μ -calculus is expressive logic (subsumes CTL*, NBAs)
- μ -calculus is based on least- and greatest fixpoint operators
- direct model-checking algorithm based on set-operations, complexity is exponential in alternation depth
- model-checking via games (winning strategy of \exists loise or \forall belard)
- bottom-up and top-down (parallel) on-the-fly coloring algorithms for alternation free formulas

