Model Checking

René Thiemann

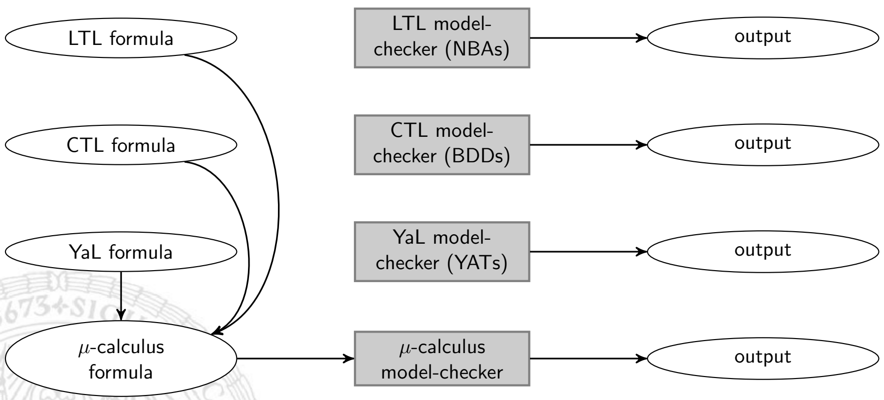
Institute of Computer Science
University of Innsbruck

SS 2008

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

$\Rightarrow$ many logics can be translated into μ-calculus

- Efficient (parallel) model-checking algorithms
- Based upon fixpoints
- Not very human-readable

$\Rightarrow$ use μ-calculus mainly for model-checking of other logics and not for direct specification

Fixpoints

Let $\tau : D \to D$ 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 $\leqslant$

- d is least fixpoint of τ ($\mathit{lfp}(\tau)$) iff
  $\tau(d) = d$ and $d \leqslant e$ for all other fixpoints e of τ
- d is greatest fixpoint of τ ($\mathit{gfp}(\tau)$) iff
  $\tau(d) = d$ and $e \leqslant d$ for all other fixpoints e of τ

Monotone Functions

Let D be a domain with partial order $\leqslant$.

- A function $\tau : D \to D$ is monotone iff $d \leqslant e \Rightarrow \tau(d) \leqslant \tau(e)$

Examples:

- x^2 is monotone over the naturals, but not over the integers
- For $D = 2^S$, $\leqslant = \subseteq$, 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 $S \neq \varnothing$

Existence and Computation of Fixpoints

Theorem (Knaster, Tarski)

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

- $\mathit{lfp}(\tau) = \tau^{|S|}(\varnothing)$
- $\mathit{gfp}(\tau) = \tau^{|S|}(S)$

Proof

Summary of Fixpoints

- X is fixpoint of τ iff $\tau(X) = X$
- Function $\tau : 2^S \to 2^S$ 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:
  - $lfp(\tau) = \tau^{|S|}(\varnothing)$
  - $gfp(\tau) = \tau^{|S|}(S)$

A Small Change in Transition Systems

Transition systems may now have labeled edges:
A *transition system* TS is a tuple

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

where

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

Example

μ-Calculus

Let $TS = (S, Act, \to, I, AP, L)$ be a transition system.
Let $\mathcal{V} = \{x, y, \dots\}$ be a set of variables (ranging over sets of states)

Definition (μ-Calculus Syntax)

A formula of the μ-calculus (L_μ-formula) has one of the following forms:

- p where $p \in AP$
- $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$
- $\langle a \rangle \varphi$ where $a \in Act$ — there is an a-successor satisfying φ
- $[a]\varphi$ where $a \in Act$ — all a-successors satisfy φ
- x where $x \in \mathcal{V}$
- $\mu x.\varphi$ where $x \in \mathcal{V}$ — least fixpoint
- $\nu x.\varphi$ where $x \in \mathcal{V}$ — greatest fixpoint

In last two cases, x may only occur in φ under an even number of negations

Binding priority: $\{\neg, \langle\cdot\rangle, [\cdot]\} \sqsupset \{\wedge, \vee\} \sqsupset \{\mu, \nu\}$

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

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

A Note on Well-Definedness

- Example:
  For $\mu x.\neg x$ obtain $\tau(X) = S \setminus X \quad \Rightarrow \quad$ no $\mathit{lfp} \quad \Rightarrow \quad$ no $[\![\mu x.\neg x]\!]_\alpha$

  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
  - $\mathit{lfp}(\tau)$ and
  - $\mathit{gfp}(\tau)$

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

- Requirement of even number of negations ensures that τ is monotone!

$\Rightarrow$ Knaster & Tarski ensures that both $\mathit{lfp}(\tau)$ and $\mathit{gfp}(\tau)$ 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:

$$TS \models \varphi \qquad \text{iff} \qquad I \subseteq [\![\varphi]\!]$$

Naive Model-Checking Algorithm:

- Just compute $[\![\varphi]\!]$ by directly applying the definition of the semantics in a top-down way
- To compute fixpoints use Knaster & Tarski
  - $\mathit{lfp}(\tau) = \tau^{|S|}(\varnothing)$
  - $\mathit{gfp}(\tau) = \tau^{|S|}(S)$
- 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 $I \subseteq \text{sem}(\varphi)$

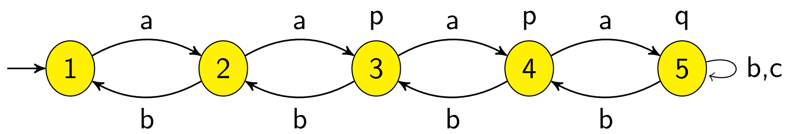
procedure reset(x)
  if x is μ-variable **then** $\alpha(x) := \varnothing$ **else** $\alpha(x) := S$

Naive MC-Algorithm for the μ-Calculus

function sem(φ)
  case φ **of**
    x : **return** $\alpha(x)$
    p : **return** $\{s \mid p \in L(s)\}$
    $\neg\psi$: **return** $S \setminus \mathrm{sem}(\psi)$
    $\psi_1 \wedge \psi_2$: **return** $\mathrm{sem}(\psi_1) \cap \mathrm{sem}(\psi_2)$
    $\psi_1 \vee \psi_2$: **return** $\mathrm{sem}(\psi_1) \cup \mathrm{sem}(\psi_2)$
    $\langle a \rangle \psi$: **return** $\{s \mid \exists s \xrightarrow{a} t, t \in \mathrm{sem}(\psi)\}$
    $[a]\psi$: **return** $\{s \mid \forall s \xrightarrow{a} t : t \in \mathrm{sem}(\psi)\}$
    $Qx.\psi$:
      reset(x)
      while true **do**
        U $:= \alpha(x)$
        V $:=$ sem(ψ)
        if U = V **then return** U **else** $\alpha(x) :=$ V

Example

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



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

Hence, $[\![\varphi]\!] = \{1,2,3,4\}$ and $TS \models \varphi$.
Complexity of naive algorithm:

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

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)

- $\mathrm{A\,X}\,\varphi \rightsquigarrow [a]\varphi$
- $\mathrm{E\,X}\,\varphi \rightsquigarrow \langle a \rangle\varphi$
- $\mathrm{A}\,\varphi\,\mathrm{U}\,\psi \rightsquigarrow \mu x.\psi \vee (\varphi \wedge [a]x)$
- $\mathrm{E}\,\varphi\,\mathrm{U}\,\psi \rightsquigarrow \mu x.\psi \vee (\varphi \wedge \langle a \rangle x)$
- $\mathrm{A\,G}\,\varphi \rightsquigarrow \nu x.\varphi \wedge [a]x$

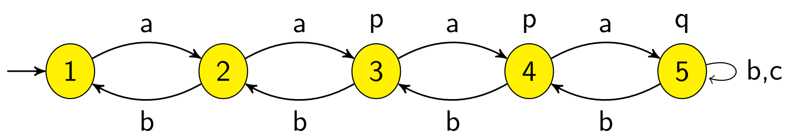
∎

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

Example

Computing $[\![\mu x.\varphi_x]\!]$ for the following TS where

$$\varphi_x = q \vee \langle a \rangle \mu y.\varphi_y$$
$$\varphi_y = p \wedge \langle a \rangle (x \vee y)$$



	$[\![\mu x.\varphi_x]\!]$	$\alpha(x)$	$[\![\varphi_x]\!]$	$[\![q]\!]$	$[\![\langle a \rangle \mu y.\varphi_y]\!]$	$[\![\mu y.\varphi_y]\!]$	$\alpha(y)$	$[\![\varphi_y]\!]$	$[\![p]\!]$	$[\![\langle a \rangle (x \vee y)]\!]$	$[\![x \vee y]\!]$
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 $TS = (S, I, \ldots, L)$

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

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

Valid $\subseteq \mathcal{V}(\varphi)$ // $x \in$ Valid implies $\alpha(x) = [\![Qx.\varphi_x]\!]_\alpha$

```
function model_check(φ)
    Valid := ∅
    for all x ∈ V(φ) do reset(x)
    return I ⊆ sem(φ)

procedure reset(x)
    if x is μ-variable then α(x) := ∅ else α(x) := S
```


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

```
function sem(φ)
  case φ of
    x : return α(x)
    p : return {s | p ∈ L(s)}
    ¬p : return {s | p ∉ L(s)}
    ψ1 ∧ ψ2 : return sem(ψ1) ∩ sem(ψ2)
    ψ1 ∨ ψ2 : return sem(ψ1) ∪ sem(ψ2)
    ⟨a⟩ψ : return {s | ∃s -a-> t, t ∈ sem(ψ)}
    [a]ψ : return {s | ∀s -a-> t : t ∈ sem(ψ)}
    Qx.ψ : if x ∈ Valid then return α(x) else while true do
      U := α(x); V := sem(ψ)
      if U = V then
        Valid := Valid ∪ {x}; return U
      else
        α(x) := V; touch(Qx.ψ)
```

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

procedure touch($Q'x.\varphi_x$)
  Valid := Valid $\setminus \{y \mid Qy.\varphi_y \in \mathcal{S}ub(\varphi_x), x \in \mathcal{FV}(\varphi_y)\}$
  Reset := $\{y \mid Qy.\varphi_y \in \mathcal{S}ub(\varphi_x), x \in \mathcal{FV}(\varphi_y), Q \neq Q'\}$
  while $z \in \{z \mid \exists y \in$ Reset, $Qz.\varphi_z \in \mathcal{S}ub(\varphi_y), \mathcal{FV}(\varphi_z) \cap$ Reset $\neq \varnothing\}$ **do**
    Reset := Reset $\cup \{z\}$
  for all $y \in$ Reset **do** reset(y)
  Valid := Valid $\setminus$ Reset

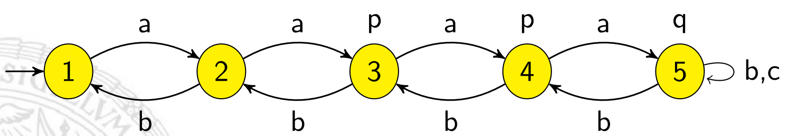
- $\mathcal{FV}(\varphi)$ is the set of *free variables* of φ
- $\mathcal{S}ub(\varphi)$ is the set of sub-formulas of φ
- φ_x is the unique formula which is the argument of "Qx."

Illustration of touch

Example

Computing $[\![\nu z.\varphi_z]\!]$ 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 $x_1 \prec_\varphi \cdots \prec_\varphi x_n$ and
the type of x_i is different to the type of x_{i+1} for every $i < n$.

A formula with $ad(\varphi) \leqslant 1$ 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 $\tau : D \to D$. If τ is monotone then

- $lfp(\tau) = \tau^{|S|}(T)$ *if* $T \subseteq \tau^k(\varnothing)$ *for some* k
- $gfp(\tau) = \tau^{|S|}(T)$ *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.

- ...
- $\mathsf{E}\,\varphi\,\mathsf{U}\,\psi \rightsquigarrow \mu x.\psi \vee (\varphi \wedge \langle a \rangle x)$
- $\mathsf{A}\,\mathsf{G}\,\varphi \rightsquigarrow \nu x.\varphi \wedge [a]x$

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.

Overview

Current approach:

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

Upcoming approach:

- Formula $\rightsquigarrow$ L_μ-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 sytem $\rightsquigarrow$ 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 $\mathcal{S}ub(\varphi)$ 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 $Q_1, \ldots, Q_n$ such that

- Each Q_i has only edges to $Q_i \cup Q_{i+1} \cup \cdots \cup Q_n$
- 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

$$c_0 \hookrightarrow c_1 \hookrightarrow c_2 \hookrightarrow \ldots$$

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

Here:

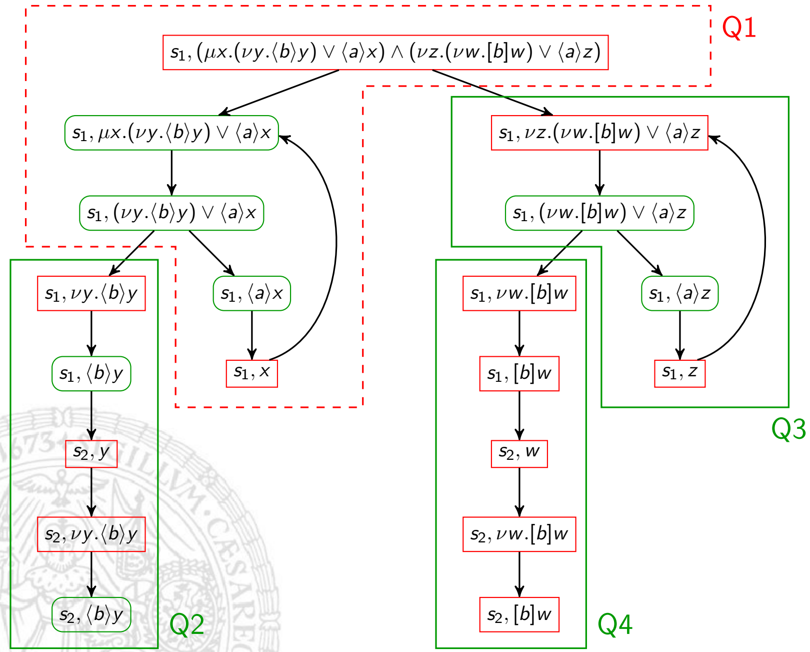
- Game graph for $TS = (S, Act, \rightarrow, I = \{s_0\}, AP, L)$ and φ has configurations $C = S \times \mathcal{S}ub(\varphi)$, initial configuration $c_0 = (s_0, \varphi)$
  (similar to tabular of Emerson Lei algorithm, but here only reachable part has to be computed! $\Rightarrow$ on-the-fly algorithm)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

Game Graph

The edges of the game graph are determined as follows:

1. If $c = (s, \psi_1 \wedge \psi_2)$ then ∀belard can move to (s, ψ_1) or (s, ψ_2)
2. If $c = (s, [a]\psi)$ then ∀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 ∃loise can move to (s, ψ_1) or (s, ψ_2)
6. If $c = (s, \langle a\rangle\psi)$ then ∃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 ∀belard, cases 5-8 belong to ∃loise (in cases 3,4,7,8 this is not important, as there is no choice)



Playing a Game

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

- If play is finite, $c_n = (s, \psi)$ is last configuration then ∀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 ∃loise wins
- ∀belard/∃loise wins an infinite play iff the maximal subformula that is visited infinitely often is a μ/ν-formula

Strategies

A strategy $\mathcal{S}tr$ 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:

$$\mathcal{S}tr : C^* C_{player} \to C \cup \{\bot\} \text{ such that for all } c_0 \ldots c_n \in C^* C_{player}:$$

- If $\mathcal{S}tr(c_0 \ldots c_n) \in C$ then $c_n \hookrightarrow \mathcal{S}tr(c_0 \ldots c_n)$ is allowed move
- If $\mathcal{S}tr(c_0 \ldots c_n) = \bot$ 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 $\mathcal{S}tr$-play

A strategy $\mathcal{S}tr$ of a player is a winning strategy if for each $\mathcal{S}tr$-play that player is the winner

A strategy $\mathcal{S}tr$ is positional, if $\mathcal{S}tr$ only considers the last configuration, i.e., $\mathcal{S}tr : C_{player} \to C \cup \{\bot\}$

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 $C_1, \ldots, C_n$ such that

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

Thus, every play starting in C_i will either

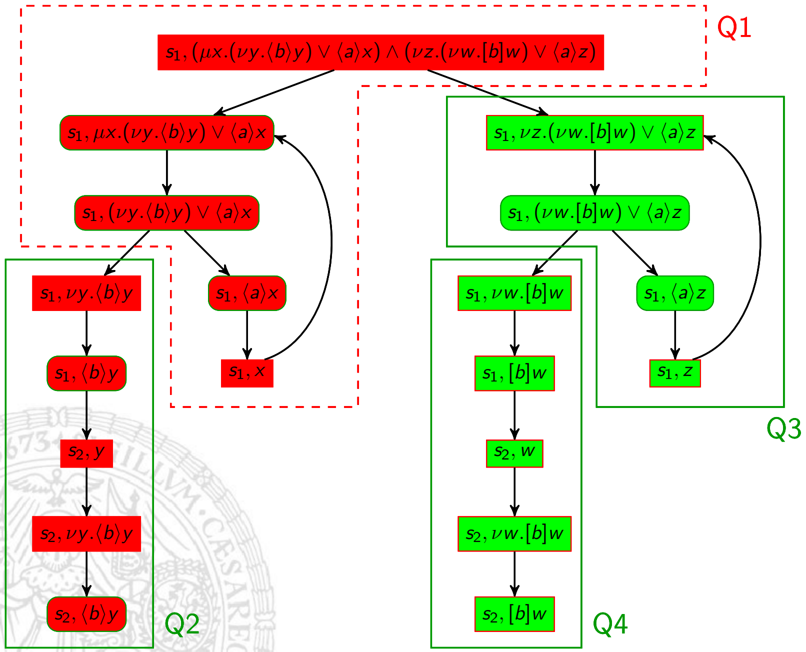
1. leave C_i and continue in some C_{i+k}, $k > 0$
2. reach a terminal configuration in C_i
   (terminal configuration = configuration without successors)
3. stay in C_i forever

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 $c_1, \ldots, c_m$ 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 $C_{i+1}, \ldots, C_n$ 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 ∃loise/∀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

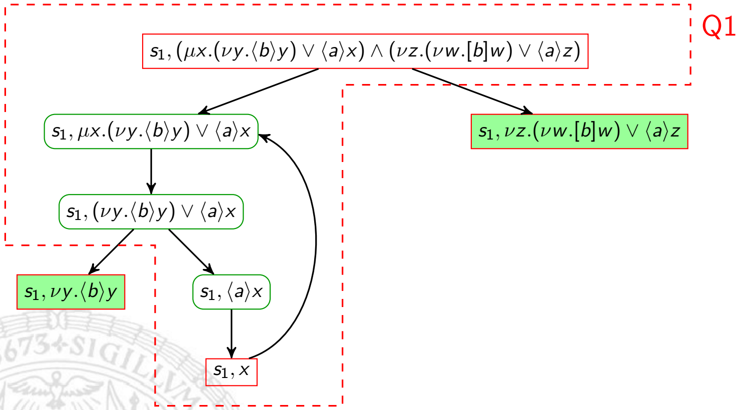
Overview:

- 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 ∃loise looses
  - ⇒ Goal of ∃loise is to leave C_1 (or reach green terminal configuration)
  - Idea: Make successors of C_1 outside C_1 attractive
  - ⇒ 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 $c_1, \ldots, c_m$ 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 $\exists$loise/$\forall$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 $f(c) \mod n = i$
    (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

$\Rightarrow$ Algorithms for parallel termination detection available
(e.g. DFG token termination algorithm of Dijkstra, Feijen, Gasteren)

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