Model Checking for the μ-calculus

1. Motivation
2. The μ-calculus
3. Evaluation algoritms
4. μ-calculus and OBDD
5. Embedding CTL in the μ-calculus
6. Encoding simulation and bisimulation in the μ-calculus
7. Related results and open problems
8. Conclusions

Motivation

What kind of computer support for verification is there?

- **Specialized tools, i.e. programs developed for fixed specification languages and inference systems.**
  - **This is the most common solution for model-checkers since high performance is important here. E.g. Uppaal, Kronos.**
  - **Some automatic theorem-provers are also of this type and for the same reason. E.g. Gandalf.**
- **Generic tools, i.e. programs which can be adapted for different specification languages and inference systems.**
  - **This is the most common solution for proof-checkers and semi-automatic theorem provers, e.g. HOL, ALF, Lego, Coq.**
  - **Some model-checkers are based on the expressive μ-calculus. The μ-calculus can be used as a specification language for other modal and temporal logics.**

Why is μ-calculus suitable as the basis for a generic model checker ?

- **Many different modal and temporal logics can be embedded in the μ-calculus, i.e. the μ-calculus can be seen as a specification language for other logics.**
- **There are efficient model checking algoritms for the μ-calculus.**

Other interesting properties of the μ-calculus:

- **Some properties which are not expressible in weaker logics can be expressed in the μ-calculus, e.g. simulation preorders, bisimilarity etc.**

The μ-calculus

Syntax

Suppose

AP	=	$\{p, p_1, p_2, ...\}$	*(atomic propositions)*
VAR	=	$\{R, R_1, R_2, ...\}$	*(relational variables)*
ACT	=	$\{a, a_1, a_2, ...\}$	*(actions)*

then the formulas are inductively defined as

$$\Phi \quad = \quad AP \mid VAR \mid \neg\Phi \mid \Phi\wedge\Phi \mid \Phi\vee\Phi \mid \langle ACT\rangle\Phi \mid [ACT]\Phi \mid \mu VAR.(\Phi) \mid \upsilon VAR.(\Phi)$$

where relational variables bound by the fixpoint operators must be in the scope of an even number of negations (Syntactic monotonicity).

Semantics

Formulas are interpreted with respect to a transition system *M = (S,T,L)* **where**

S	*is a nonempty set of states*
$T \in ACT \to 2^{S \times S}$	*is a mapping that takes each action to the corresponding transition relation over S*
$L \in AP \to 2^S$	*is a mapping that takes each atomic proposition to the set of states in which it is true*

and an environment $e \in VAR \to 2^S$**. Each formula** ϕ **is interpreted as a set of states** $\|\phi\|_M\, e$ **defined as follows:**

$$
\begin{aligned}
\|p\|_M\, e &= L(p)\\
\|R\|_M\, e &= e(R)\\
\|\neg\phi\|_M\, e &= S - \|\phi\|_M\, e\\
\|\phi \wedge \psi\|_M\, e &= \|\phi\|_M\, e \cap \|\psi\|_M\, e\\
\|\phi \vee \psi\|_M\, e &= \|\phi\|_M\, e \cup \|\psi\|_M\, e\\
\|\langle a\rangle\phi\|_M\, e &= \left\{s \mid \exists t\left[s \xrightarrow{a} t \wedge t \in \|\phi\|_M\, e\right]\right\}\\
\|[a]\phi\|_M\, e &= \left\{s \mid \forall t\left[s \xrightarrow{a} t \to t \in \|\phi\|_M\, e\right]\right\}\\
\|\mu R\phi\|_M\, e &= lfp(\lambda S \|\phi\|_M\, e[R \leftarrow S])\\
\|\nu R\phi\|_M\, e &= gfp(\lambda S \|\phi\|_M\, e[R \leftarrow S])
\end{aligned}
$$

Observations

- **By the requirement of syntactic monotonicity the operator** $\lambda S \|\phi\|_M\, e[R \leftarrow S]$ **is monotonic and thus have a** *lfp* **and** *gfp* **(Knaster-Tarski) over the lattice** $\langle 2^S, \subseteq, \cap, \cup \rangle$.

- **Note! These fixpoints exist regardless of the cardinality of the set** *S***.**

- **Since we are considering transition systems where** *S* **is finite** $\tau = \lambda S \|\phi\|_M\, e[R \leftarrow S]$ **is also** $\cap$**-continuous and** $\cup$**-continuous. The** *lfp* **and the** *gfp* **can thus be computed by iteration:**

$$
\begin{aligned}
\|\mu R\phi\|_M\, e &= \bigcup_{i \in \omega} \tau^i(\varnothing)\\
\|\nu R\phi\|_M\, e &= \bigcap_{i \in \omega} \tau^i(S)
\end{aligned}
$$

where $\tau^0(S) = S$ **and** $\tau^{i+1}(S) = \tau(\tau^i(S))$**.**

Alternation depth

The alternation depth *d*(ϕ) **of a formula** ϕ **is defined as**

$$
\begin{aligned}
d(p) &= 0\\
d(R) &= 0\\
d(\neg\phi) &= d(\phi)\\
d(\phi \wedge \psi) &= max(d(\phi), d(\psi))\\
d(\phi \vee \psi) &= max(d(\phi), d(\psi))\\
d(\langle a\rangle\phi) &= d(\phi)\\
d([a]\phi) &= d(\phi)\\
d(\mu R\phi) &= max(1, d(\phi), 1 + d(\nu R\psi_1), \ldots, 1 + d(\nu R\psi_n))\\
d(\nu R\phi) &= max(1, d(\phi), 1 + d(\mu R\psi_1), \ldots, 1 + d(\mu R\psi_n))
\end{aligned}
$$

where $\nu R\psi_i$ *and* $\mu R\psi_i$ *are top-level* ν*– and* μ*-subformulas of* Φ*, i.e. subformulas of* Φ *not contained in any fixpoint subformula of* Φ*.*

Example: $d(\nu Y(P \wedge \langle a\rangle(\mu X(P \wedge \langle a\rangle X) \vee (h \vee Y)))) = 2$

Evaluation algoritms

A naive algoritm

- **The semantics of** μ**-calculus over finite transition systems suggests a naive evaluation algoritm.**

- **Given a structure** *M=(S,T,L)* **and an environment** *e* **we can find the set of states** *eval*(Φ, *e*) **in which** Φ **is true.**

- **If** Φ **is closed then** *eval*(Φ, *e*) **does not depend on** *e***.**

- **The following notation (and its extensions) will be used below:**

$$\psi(R_j^{i_1 \ldots i_j}) = \|\psi(R_j)\|_M(e[R_j \leftarrow R_j^{i_1 \ldots i_j}])$$

where $R_j^{i_1 \ldots i_j} \subseteq S$ **and** *e* **is given by the context.**

Example:

- Let $\Phi = \mu R_1 \psi_1(R_1, \mu R_2 \psi_2(R_1, R_2))$.
- We will calculate $\|\phi\|_M$ by succsessive approximations:
  - Start by approximating $\|\phi\|_M$ with $R_1^0 = \varnothing$
  - Calculate $R_2^{0\omega} = \mu R_2 \psi_2(R_1^0, R_2)$ by succsessive approximations.
  - Start approximating $\mu R_2 \psi_2(R_1^0, R_2)$ with $R_2^{00} = \varnothing$, then calculate $R_2^{01} = \psi_2(R_1^0, R_2^{00})$, $R_2^{02} = \psi_2(R_1^0, R_2^{01})$ and so on until $R_2^{0i} = \psi_2(R_1^0, R_2^{0i})$ for some finite *i*. Then $R_2^{0i} = R_2^{0\omega}$.
  - Now we can calculate our next approximation to $\|\phi\|_M$, i.e. $R_1^1 = \psi_1(R_1^0, R_2^{0\omega})$.
  - This process can be repeated until $R_1^j = \psi_1(R_1^j, R_2^{j\omega})$ and we have $R_1^j = R_1^\omega = \|\phi\|_M$

Problem with the naive algoritm

- Time complexity is exponential in the length of the formula.
- Consider the loops taking care of the fix-point cases. If $\Phi = \mu R_1(\mu R_2(\mu R_3(\mu R_4(P))))$ is evaluated in a structure with *n* states then $\mu R_2(\mu R_3(\mu R_4(P)))$ will be evaluated maximum *n+1* times, $\mu R_3(\mu R_4(P))$ max $(n+1)^2$ times, $\mu R_4(P)$ max $(n+1)^3$ times and , P max $(n+1)^4$ times.
- Note! Each of the $(n+1)^4$ possible evaluations of P for example, are done with respect to different environments.

A better algoritm (Emerson and Lei)

- Note that in the example above $R_2^{j0} = \varnothing$ for all *j* and this is the cause for the bad time complexity.
- Lemma:

$$If \qquad S \subseteq \bigcup_{i \in \omega} \tau^i(\bot) \qquad then \qquad \bigcup_{i \in \omega} \tau^i(S) = \bigcup_{i \in \omega} \tau^i(\bot)$$

- Since $R_1^0 \subseteq R_1^1$ also $\mu R_2 \psi_2(R_1^0, R_2) \subseteq \mu R_2 \psi_2(R_1^1, R_2)$ due to monotonicity.
- We have $R_2^{0\omega} = \mu R_2 \psi_2(R_1^0, R_2)$ and $\mu R_2 \psi_2(R_1^1, R_2) = \bigcup_{i \in \omega} \psi_2(R_1^1, \bot)^i$.
- So $\mu R_2 \psi_2(R_1^1, R_2) = \bigcup_{i \in \omega} \psi_2(R_1^1, R_2^{0\omega})^i$, i.e. we can start the iteration at $R_2^{0\omega}$.
- Since $R_1^0 \subseteq R_1^1$, $R_1^1 = \psi_1(R_1^0, R_2^{0\omega})$, $R_2^{0\omega} = \mu R_2 \psi_2(R_1^0, R_2)$ we know $R_1^1 = \psi_1(R_1^0, R_2^{0\omega}) \subseteq \psi_1(R_1^1, R_2^{1\omega}) = R_1^2$ and the same argument can be used again.
- Overall we need only $O(|S|)$ iterations instead of $O(|S|^2)$, i.e. nested fix-point formulas of the same type can be evaluated in linear time.
- This idea leads to an algoritm that is exponential in the alternation dept $d(\phi)$ of a formula ϕ.

μ-calculus and OBDD

Transition systems

A structure *M=(S,T,L)* is encoded as before:

- *S* is encoded as $\{0,1\}^n$.
- *T(a)* is encoded as an ODDB which we denote $ODDB_a(x_1,...,x_n,x'_1,...,x'_n)$. I.e. we get an ODDB for each action $a \in Act$.
- *L(p)* is encoded as an ODDB which we denote $ODDB_p(x_1,...,x_n)$. I.e. we get an ODDB for each action $p \in AP$.

Environment

- An environment over the relational variables $R_1,...,R_k$ is encoded as a function *A* from $\{R_1,...,R_k\}$ to OBDD's of *n* variables.
- Below we assume that *A* is defined for the free relational variables in the treated formula.

The sets denoted by formulas

Given an encoded structure and an encoded environment *A* each formula is associated with an OBDD in the following way:

$$
\begin{array}{lcl}
B(p, A) & = & OBDD_p(\bar{x}) \\
B(R, A) & = & A(R) \\
B(\neg\phi, A) & = & \neg(B(\phi, A)) \\
B(\phi \wedge \psi, A) & = & B(\phi, A) \wedge B(\psi, A) \\
B(\phi \vee \psi, A) & = & B(\phi, A) \vee B(\psi, A) \\
B(\langle a \rangle \phi, A) & = & \exists \bar{x}'(ODDB_a(\bar{x}, \bar{x}') \wedge B(\phi, A)(\bar{x}')) \\
B([a]\phi, A) & = & B(\neg\langle a \rangle \neg\phi, A) \\
B(\mu R\phi, A) & = & FIX(\phi, A, FALSEBDD) \\
B(\nu R\phi, A) & = & FIX(\phi, A, TRUEBDD)
\end{array}
$$

där *FIX* ges av:

Embedding CTL in the μ-calculus

Expressiveness

Different logics can be compared by their expressive power. This requires that we can compare their semantic domains in some way.

Example:

A CTL-formula can be interpreted as true or false in a state of a Kripke structure M_a with a singleton action set *Act={a}*. It is thus possible to compare CTL-formulas and μ-calculus formulas over such restricted Kripke structures.

Equivalence

A CTL-formula ϕ and a closed μ-calculus formula ψ are equivalent, denoted $\phi \equiv \psi$, if the following holds:

$$\forall M_a(\|\phi\|_{M_a} = \|\psi\|_{M_a})$$

where M_a are restricted Kripke structures (*Act={a}*) and $\|\phi\|_{M_a}$ is the set of states of M_a satisfying the CTL-formula ϕ.

Translation

The CTL-formulas can be translated into closed μ-calculus formula in the following way:

$$
\begin{array}{lcl}
Tr(p) & = & p \\
Tr(\neg f) & = & \neg Tr(f) \\
Tr(f \wedge g) & = & Tr(f) \wedge Tr(g) \\
Tr(EXf) & = & \langle a \rangle Tr(f) \\
Tr(E[fUg]) & = & \mu Y(Tr(g) \vee (Tr(f) \wedge \langle a \rangle Y)) \\
Tr(EGf) & = & \nu Y(Tr(f) \wedge \langle a \rangle Y)
\end{array}
$$

Theorem

For all CTL-formulas *f*, $f \equiv Tr(f)$ holds.

Note

This shows that μ-calculus is more expressive than CTL, in fact μ-calculus is strictly more expressive.

Simulation and bisimulation in μ-calculus

Simulation relation

A relation $E \subseteq S \times S$ is called a simulation relation if the following holds:

$$(s, s') \in E \rightarrow \forall a \in Act \forall q \in S\left[\left(s \xrightarrow{a} q\right) \rightarrow \exists q' \in S\left[\left(s' \xrightarrow{a} q'\right) \wedge (q, q') \in E\right]\right]$$

Bisimulation relation

A relation $E \subseteq S \times S$ is called a bisimulation relation if the following holds:

$$(s, s') \in E \rightarrow \left(\begin{array}{c} \forall a \in Act \forall q \in S\left[\left(s \xrightarrow{a} q\right) \rightarrow \exists q' \in S\left[\left(s' \xrightarrow{a} q'\right) \wedge (q, q') \in E\right]\right] \\ \wedge \\ \forall a \in Act \forall q' \in S\left[\left(s \xrightarrow{a} q'\right) \rightarrow \exists q \in S\left[\left(s \xrightarrow{a} q\right) \wedge (q, q') \in E\right]\right] \end{array}\right)$$

Simulation preorder

$s \le s'$ iff there exists a simulation relation *E* such that $(s, s') \in E$.

Bisimulation equivalence

$s \sim s'$ iff there exists a bisimulation relation *E* such that $(s, s') \in E$.

Checking simulation

Given two transition systems *M=(S,T,L)*, *M'=(S,T',L')* **over** *Act* **and two states** $s,s' \in S$. **Then it is possible to use the** μ**-calculus to check whether** $s' \in S$ **in** *M'* **simulates** $s \in S$ **in** *M.*

Construct the transition system $\overline{M}=(\overline{S},\overline{T},\overline{L})$ **over** $\overline{Act}$ **where**

$$\overline{Act} \subseteq Act \times Act = \{(a,\tau),(\tau,a) \mid a \in Act\}$$

$$\bar{S} = S \times S$$

$$\bar{T}((a,b)) = \{((s,s'),(q,q')) \mid (s,q) \in T(a) \wedge (s',q') \in T'(b)\}$$

$\bar{L}(p)$ = *may be arbitrary*

Note! We assume that the idle action $\tau \in Act$.

Now, $s' \in S$ **in** *M'* **simulates** $s \in S$ **in** *M* **iff the following formula holds in** *(s,s')* **of** $\overline{M}$**.**

$$\nu X[[(a_1,\tau)]\langle(\tau,a_1)\rangle X \wedge \ldots \wedge [(a_n,\tau)]\langle(\tau,a_n)\rangle X]$$

where $\{a_1, \ldots, a_n\} = Act$ **.**

Checking bisimulation

Given two transition systems *M=(S,T,L)*, *M'=(S,T',L')* **and two states** $s,s' \in S$. **Then it is possible to use the** μ**-calculus to check whether** $s' \in S$ **in** *M'* **is bisimulation equivalent to** $s \in S$ **in** *M.*

The state $s' \in S$ **in** *M'* **is bisimilar to** $s \in S$ **in** *M* **iff the following formula holds in** *(s,s')* **of** $\overline{M}$**.**

$$\nu X[[(a_1,\tau)]\langle(\tau,a_1)\rangle X \wedge [(\tau,a_1)]\langle(a_1,\tau)\rangle X \wedge \ldots]$$

where $\{a_1, \ldots, a_n\} = Act$ **.**

Related results and open problems

Related results

- **Satisfiability is decideable for** μ**-calculus. I.e. it is possible to automatically check whether a specification is consistent.**
- **Kozen has constructed an axiomatization (1983) for the** μ**-calculus which was proven complete in (1993 ?) by Walukiewicz.**

Open problems

- **Even though it is possible to exploit monotonicity properties when evaluating formulas the algoritms remain exponential in the alternation depth. Are there polynomial evaluation algoritms?**