Formal verification. Lecture 7

Marius Minea

Problem setting

Specification formulas can be converted to automata
(LTL tableau construction)
– represent “simplest” system that conforms to the specification
When using an automaton as specification:
– what does it mean to say “system functions like this automaton”
How does one build (abstract) a simpler model from a complex one?
Does verifying a simpler model ensure correctness of the initial one?
Can one deduce correctness of a composite model from proving properties of the components?

Simulation relation

Consider two structures \( M \) and \( M' \), with \( AP \supseteq AP' \). A relation \( \leq \subseteq S \times S' \) is a simulation relation between \( M \) and \( M' \) iff \( \forall s \leq s' \):

- \( L(s) \cap AP' = L(s') \) (s and \( s' \) labeled identically with respect to \( AP' \))
- \( \forall s_1 \text{ with } s \rightarrow s_1 \text{ there exists } s'_1 \text{ with } s' \rightarrow s'_1 \text{ and } s_1 \leq s'_1 \)
- (any successor of \( s \) is simulated by a successor of \( s' \))

The structure \( M' \) simulates \( M \) \( (M \leq M') \) of there exists a simulation relation \( \leq \) such that for the initial states: \( \forall s_0 \in S_0 \exists s'_0 \in S'_0 \cdot s_0 \leq s'_0 \)

Prop.: The simulation relation is a preorder over the set of structures (reflexive and transitive). We choose: \( s \leq s' \iff \exists s'' \leq s' \)

Theorem: If \( M \leq M' \) then \( M' \models f \Rightarrow M \models f \), for any ACTL* formula \( f \) over \( AP' \).

Bisimulation relation

Let \( M \) and \( M' \) be two structures with \( AP' = AP \). A relation \( \equiv \subseteq S \times S' \) is a bisimulation relation between \( M \) and \( M' \) iff \( \forall s, s' \):

- \( L(s) = L(s') \)
- \( \forall s_1 \text{ with } s \rightarrow s_1 \text{ there exists } s'_1 \text{ with } s' \rightarrow s'_1 \text{ and } s_1 \equiv s'_1 \)
- \( \forall s_2 \text{ with } s' \rightarrow s_2 \text{ there exists } s_2 \text{ with } s \rightarrow s_2 \text{ and } s_1 \equiv s'_2 \)
- (or: \( \equiv \) is a symmetric simulation relation between \( M \) and \( M' \) and between \( M' \) and \( M \))

Structures \( M \) and \( M' \) are bisimilar if there exists a bisimulation relation \( \equiv \) such that for initial states: \( \forall s_0 \in S_0 \exists s'_0 \in S'_0 \cdot s_0 \equiv s'_0 \) and \( \forall s_0 \in S_0 \exists s'_0 \in S'_0 \cdot s_0 \equiv s'_0 \)

Prop.: The bisimulation relation is an equivalence relation among structures

Theorem: If \( M \equiv M' \) then \( \forall f \in \text{CTL}^* \text{, } M \models f \Rightarrow M' \models f \); Conversely: Two structures that satisfy the same CTL* (or even CTL) formulas are bisimilar (equivalently: two structures which are not bisimilar can be distinguished by a CTL formula).

Example: language inclusion and simulation

Consider a Kripke structure \( M \) with a set \( AP \) of atomic propositions

Language of \( M \) = set of execution traces seen as sequences of labels
Formally: \( L(M) = \text{set of infinite words (strings)} \ a_0a_1a_2 \ldots \) such that there exists a path \( a_0a_1a_2 \ldots \) of \( M \) with \( L(s_0) = \alpha_0 \).

Language inclusion preserves LTL properties:
\[ L(M) \subseteq L(S) \iff \forall A \in \text{LTL} : S \models A \Rightarrow M \models A \]

Comparing models. Abstraction. Compositional reasoning

28 octombrie 2003

Comparing models. Abstraction. Compositional reasoning

Formal verification. Lecture 7

Marius Minea

Comparing models. Abstraction. Compositional reasoning

Formal verification. Lecture 7

Marius Minea

Comparing models. Abstraction. Compositional reasoning

Formal verification. Lecture 7

Marius Minea

Comparing models. Abstraction. Compositional reasoning

Formal verification. Lecture 7

Marius Minea

Comparing models. Abstraction. Compositional reasoning

Formal verification. Lecture 7

Marius Minea

Comparing models. Abstraction. Compositional reasoning

Formal verification. Lecture 7

Marius Minea

Comparing models. Abstraction. Compositional reasoning

Formal verification. Lecture 7

Marius Minea
Formal verification. Lecture 7  
Marius Minea

Abstraction techniques (e.g. reduced domain for variables)

- semantic
- syntactic

\[
\begin{align*}
\text{Example: simulation and bisimulation} & \\
M_1 & \equiv M_2 \\
M_1 & \models M_2
\end{align*}
\]

Generally: \( M \models M' \Rightarrow M \models M' \land M' \models M \)

In the figure: \( M_1 \models M_2, M_2 \models M_1 \) but \( M_2 \not\equiv M_2 \)

Equivalent definition (as a game): \( M \equiv M' \) if any choice of a model and of a move in it can be matched by an equally labelled move in the other model.

(Choice of model done at each step \( \Rightarrow \) symmetry)

\[
\begin{align*}
\text{Example: bisimulation} & \\
M_1 & \equiv M_2 \\
\text{(duplicating nodes does not change branching properties)} & \\
\end{align*}
\]

Comparing models. Abstraction. Compositional reasoning

Deterministic system = single initial state; any two successors differently labeled \( s \rightarrow s_1 \land s \rightarrow s_2 \land s_2 \not\equiv s_1 \Rightarrow L(s_1) \not\equiv L(s_2) \)

Simulation:

\( M, M' \) deterministic: \( M \models M' \) iff \( L(M) \subseteq L(M') \)

In general, we recursively define:

\[
\begin{align*}
x & \not\in x' \Rightarrow L(x) \cap AP' = L(x') \\
x \in x' \land \forall \exists s_1, s_2 \rightarrow s_1 \Rightarrow 3x_1', x_2' \in x_1 \land x_2 \not\in L(s_2) & \\
\text{We have if } s_{i+1} \subseteq s_i, s_{i+1} \subseteq (s_{i+1} \not\equiv) \text{ (finite models)}
\end{align*}
\]

Bisimulation:

\( M, M' \) deterministic: \( M \equiv M' \) iff \( L(M) = L(M') \)

In general, we recursively define:

\[
\begin{align*}
x & \not\in x' \Rightarrow L(x) = L(x') \\
x \in x' \land \forall \exists s_1, s_2 \rightarrow s_1 \Rightarrow 3x_1', x_2' \in x_1 \land x_2 \not\in L(s_2) & \\
\text{We have if } s_{i+1} \subseteq s_i, s_{i+1} \subseteq (s_{i+1} \not\equiv) \text{ (finite models)}
\end{align*}
\]

Abstraction: Introduction

Abstraction is the key step in verifying systems of realistic size.

- it means constructing an abstract system (with fewer details)
- and establishing a correspondence between the abstract and the original system
  - exact abstractions: preserve truth value
  - conservative abstractions (approximations): correctness of abstract system implies correctness of real system, but not conversely
  (counterexample in the abstract system may not exist in the real one)

The abstract model must be obtained without building the concrete one

(there is often impossible due to size)
  - syntactic abstraction techniques
  - semantic abstraction techniques (e.g. reduced domain for variables)

Comparing models. Abstraction. Compositional reasoning

- Timed abstractions (region automaton, zone graph)
- are finite abstractions of an infinite-state systems
- several states in the concrete system match a state in the abstract system

A specification is usually an abstraction of the implementation

- the tableau for the LTL formula is an abstraction for a system that satisfies it

Refinement relations (language inclusion, simulation, etc.) between two different systems.

Using 1-bit packets in the protocol model of project 1 (data abstraction)
Formal verification. Lecture 7

Marius Minea

⇒ we can not always have a precise abstraction

⇒ the abstract system may introduce new behaviors (e.g., the 3-state traffic light reduced to 2 states)

\[ \text{Cone of influence reduction} \]

Abstraction by removal of variables that do not affect specification.

Let \( M \) be a system with variable set \( V = \{v_1, v_2, \ldots, v_n\} \) described by the equations \( \phi_j = f_j(V) \).

Let \( V' \) be the set of variables referenced in the specification.

The cone of influence of \( V' \) is minimal set \( C \subseteq V \) such that:

- \( V' \subseteq C \)
- If \( v_i \in C \), and \( f_i \) depends on \( v_j \), then \( v_j \in C \) (transitive closure)

We build a new system \( M' \) eliminating all the variables that do not appear in \( C \), together with their functional equations.

\[ \text{Data abstraction} \]

- used for reasoning about circuits with large bit width, or about programs with complex data structures
- useful if data processing operations are relatively simple (transfer, small number of arithmetic / logic ops)

Main idea: establishing a correspondence between original domain of data and a smaller-size domain (usually a few values)

Example: sign abstraction

\[ h(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{positive} & \text{positive} \\ \text{negative} & \text{negative} \\ \text{zero} & \text{positive} \end{cases} \]

⇒ we can not always have a precise abstraction
⇒ abstraction domain and function must be carefully chosen

\[ \text{Invariance of CTL* Specifications} \]

We prove that cone of influence reduction preserves the truth values of CTL* specifications (defined over variables from \( C \)).

Let \( V = \{v_1, v_2, \ldots, v_n\} \) be a set of boolean variables, and \( M = (S, S_0, R, L) \), with:

- \( S = \{0, 1\}^n \) is set of assignments to \( V \); \( S_0 \subseteq S \)
- \( R = \{f_1, \ldots, f_n\}(V) \)
- \( L(x) = \{v_i | h(v_i) = 1\} \) (variables equal to 1 in \( x \))

Let \( V \) be numbered such that \( C = \{v_1, \ldots, v_k\} \). We define \( M' = (S', S_0', R', L') \):

- \( S' = \{0, 1\}^k \) is set of assignments to \( C \)
- \( S_0' = \{(d_1, \ldots, d_k) | (d_1 = d_2 = \cdots = d_k = 0) \} \)
- \( R' = \{f_1', \ldots, f_k'(C)\} \)
- \( L'(x) = \{v_i | h(v_i) = 1\} \)

We can show that the concrete model \( M \) and the abstract model \( M' \) are bisimilar.

\[ \text{Generating the abstract system} \]

– for any variable \( x \), we define an abstract variable \( \hat{x} \)
– we label states with atomic propositions indicating the abstract value

\[ \text{Abstraction example} \]

3-state traffic light reduced to 2 states

\[ L_i(R) \equiv \text{stop} \]

\[ L_i(G) \equiv \text{go} \]

\[ L_i(Y) \equiv \text{green} \]

Note: the abstract system may introduce new behaviors (e.g., the system can stay in the "stop" state forever).
Exact and approximate abstractions

Consider a system represented implicitly, by predicates for the transition relation $R$ and the initial states $S_0$. We assume the same abstraction function for all variables, $h: D \rightarrow \hat{A}$ (abstract domain, $\hat{A}$ = abstract domain).

We must define $S_0$ and $R$ for the abstract system:

$S_0 = \{x_1, \ldots, x_n : h(x_1) = \hat{x}_1 \land \cdots \land h(x_n) = \hat{x}_n\}$

We similarly define $\hat{R}(\hat{x}_1, \ldots, \hat{x}_n)$ from $\hat{R}(x_1, \ldots, x_n)$ we obtain $\hat{R}(\hat{x}_1, \ldots, \hat{x}_n)$ expressed in abstract variables

Transforming $\phi \mapsto \hat{\phi}$ may be a complex operation $\Rightarrow$ we apply it (like negation) just to elementary relations between variables ($e.g.$, $=,<,\geq$, etc.).

Define by structural induction an approximate abstraction $\hat{A}$:

$\hat{A}(\hat{P}(x_1, \ldots, x_n)) = \hat{P}(\hat{x}_1, \ldots, \hat{x}_n)$ if $P$ is an elementary relation.

$\hat{A}(\hat{P}(x_1, \ldots, x_n)) = \hat{P}(\hat{x}_1, \ldots, \hat{x}_n)$

$\hat{A}(\hat{A}( \hat{x}_1) \land \hat{A}( \hat{x}_2) = \hat{A}(\hat{x}_1) \lor \hat{A}(\hat{x}_2)\}$

$\hat{A}(\hat{x}) = h(x)$, then

$\hat{A}(\hat{x} \phi) = \forall \hat{A}(\hat{x})$

Formal verification. Lecture 7 Marius Minea

With the definitions so far, one can prove: $\forall \phi. \phi \Rightarrow \hat{A}(\phi)$

In particular, $S_0 = A(S_0)$ and $\hat{R} = A(R)$.

(approximation may introduce additional initial states and transitions)

Fix the model abstract approximates $M_a = (S_a, A(S_a), A(R), L_a)$. Then $M \leq M_a$ (the abstract approximated model simulates the original).

If the abstraction function preserves the relations which corresponds to primitive operations in a program, the abstraction $A$ is exact.

An abstraction function $h$ defines an equivalence relation between the concrete values for $x$ which correspond to the same abstract values:

$\forall x_1, x_2. x_1 \equiv x_2 \Leftrightarrow h(x_1) = h(x_2)$

If the value of any primitive relation $P$ in the program is the same for any two pair of equivalent concrete values:

$\forall x_1, x_2. x_1 \equiv x_2 \Rightarrow P(x_1) = P(x_2)$

Then $M \leq M_a$ (the abstract model simulates the concrete model)

Formal verification. Lecture 7 Marius Minea

A method for defining the abstract semantics of a program that can be used to analyse the program and produce information about its runtime behavior. [Couust & Cousut ‘77]

Consists in:

- a concrete domain $D$ and an abstract domain $\hat{A}$, linked via a Galois connection:

  - an abstraction function $\alpha: D \rightarrow \hat{A}$

  - a concretization function $\gamma: \hat{A} \rightarrow P(D)$

  (associates to each abstract state a set of concrete states)

  - a $\forall \exists$ in $P(D) : x \subseteq \gamma(h(x)) \Rightarrow \forall x \in \hat{A} \Rightarrow \alpha(\gamma(h(x)))$

(abstraction followed by concretization introduces approximation)

Concretization followed by abstraction is exact

The majority of abstractions can be formulated in this general framework

Formal verification. Lecture 7 Marius Minea

Symbolic abstractions

To verify the datapaths of a system

(main function: computing and preserving values)

Example: correct transmission from $a$ to $b$. Initially, for a fixed value:

$AG(a = 17) \Rightarrow A\hat{X}b = 17$\n
Abstraction function: $h(x) = 1 \text{ if } x = 17$

0 otherwise.

More generally: we introduce the symbolic parameter $c$:

$h(x) = \begin{cases} 1 & \text{ if } x \equiv c \\ 0 & \text{ otherwise} \end{cases}$

$\Rightarrow$ abstract transition relation $\hat{R}(\hat{a}, \hat{b}, \hat{c})$

In a BDD representation, $c$ does not affect the complexity if the system behavior does not depend on $c$

Example: pipelined adder with two stages

$AG(reg1 = a \land reg2 = b \Rightarrow A\hat{X}A\hat{X} a + b)$

Formal verification. Lecture 7 Marius Minea

Compositional reasoning: an application of “divide and conquer” to verification of a system built from components

- verification of local properties of components
- deriving global properties from component properties
- without constructing a model of the entire system (impractical)

Compositional reasoning: generic term for rules of the form:

- $M_1 \models f_1 \land M_2 \models f_2 \Rightarrow Composition(M_1, M_2) \models LogicOp(f_1, f_2)$

E.g. parallel composition, and LogicOp = $\land$,

- $M_1 \land M_2 \models CompOp(M_1) \land CompOp(M_2)$

Ex.: $\Rightarrow$ implementation, refinement; $CompOp(\cdot) = \|M$

- $M_1 \land S_1 \land M_2 \land S_2 \Rightarrow Composition(M_1, M_2) \land Composition(S_1, S_2)$
Comparing models. Abstraction. Compositional reasoning


circular assures-guarantee

Comparing models. Abstraction. Compositional reasoning

Exemplu de dependențe

O modulul de implementare a două numere, $n + d$, în baza lui, cu două componente:

$M_0(n + d) = M_0(n) \land M_0(d)$

Din $M_0$-sau $M_0$, să satisfacă impunerea următorii învariante:

- $S_0: 0 \leq q < b + x \land x \\leq r < (q + 1) + d$
- $S_0: 0 \leq r < b + d$

Totuși, individul nu avem nici $M_0$ sau nici $M_0$, care funcționează corect la fiecare modul depinde de ceilalalt.

Dar avem $S_0 \Rightarrow M_0 \Rightarrow S_0 \Rightarrow S_0 \Rightarrow S_0$.

(1) $M \subseteq f \Rightarrow A$  
(2) $M \equiv_{IP} M^f \Rightarrow A$  
(3) $A \equiv_{IP} M \Rightarrow f$  
(4) $M \equiv_{IP} \Rightarrow T_0$  
(5) $M \equiv_{IP} \Rightarrow T_0$  
(6) $M \equiv_{IP} \Rightarrow f$  
(7) $T_0 \equiv_{IP} \Rightarrow f$  
(8) $M \equiv_{IP} \Rightarrow f$  
(9) $M \equiv_{IP} \Rightarrow f$  
(10) $M \equiv_{IP} \Rightarrow f$  
(11) $M \equiv_{IP} \Rightarrow f$

Justificarea raționamentului

Demonstrăm teoreme pot funcționa de expresii descriitorului prezentate în raționamentele pe componente și asigura validitatea deductiei.

Comparing models. Abstraction. Compositional reasoning

Circular assures-guarantee rules

Studied in various contexts [Chandi & Misra’81, Abadi & Lamport ‘93]

We refer to Reactive Modules [Alur & Henzinger ‘95] – models with input and output variables, and transition relation

- dependence relation $\subseteq V_{out} \setminus V_{seq}$

We define the refinement (implementation) relation $M \subseteq M'$ iff

$\forall V(M') \subseteq V(M)$, $V_{out}(M') \subseteq V_{out}(M)$, $\forall M \subseteq M'$, $\forall L(M') \subseteq L(M)$

(1) $P$ can function in a context, so can $Q$.
Circular assume-guarantee rules (cont'd)

For reactive modules:

\[
M_1 || S_2 \leq S_1 || S_2 \\
M_1 || M_2 \leq S_1 || S_2 \\
M_1 || S_2 \leq S_1 || S_2 \\
M_1 || S_1 \leq S_1 || S_2
\]

(assuming all compositions well defined)

Advantage: although there are two relations to prove, each is simpler than the original one.

- Specification description \( S_i \) is much simpler than the implementation \( M_i \).
- Need not compose two different implementations (often impossible).

Rule with temporal induction [McMillan’97]

Valid for invariants (safety properties)

- If \( P_1 \land Q_1 \) true at 0, 1, \ldots, \( t \) \( \Rightarrow \) \( Q_2 \) true at \( t + 1 \)
- If \( P_2 \land Q_2 \) true at 0, 1, \ldots, \( t \) \( \Rightarrow \) \( Q_1 \) true at \( t + 1 \) or \( \text{at} \)
- Then for any \( t \), \( P_1 \land P_2 \Rightarrow Q_1 \land Q_2 \)

Compositionality and refinement

[Henzinger’01] - Study of the theory of interfaces

For a refinement relation \( \leq \) and a composition relation \( || \), we wish:

If \( M_1 \leq S_1 \) and \( M_2 \leq S_2 \), then \( M_1 || M_2 \leq S_1 || S_2 \)

Generally, insufficient – components may be incompatible.

\( \Rightarrow \) two variants:

- If \( M_1 \leq S_1 \) and \( M_2 \leq S_2 \) and \( M_1 || M_2 \) is defined, then \( S_1 || S_2 \) is defined and \( M_1 || M_2 \leq S_1 || S_2 \)
  - Formalism focused on components
  - Allows independent verification of components (bottom-up)

- If \( M_1 \leq S_1 \) and \( M_2 \leq S_2 \) and \( S_1 || S_2 \) is defined, then \( M_1 || M_2 \) is defined and \( M_1 || M_2 \leq S_1 || S_2 \)
  - Formalism focused on interfaces
  - Allows independent implementation of interfaces (top-down)