Partial order reduction

Model checking with automata

27 October 2005

Formal verification. Lecture 4
Marius Minea

Model checking “on-the-fly”

System state space = cartesian product for components: \( S = S_1 \times \ldots \times S_n \)
⇒ exponential if number of components; may be impossible to build

Specifications given as automata can guide verification algorithms:
⇒ only the needed parts of state space are constructed

Approach: build automaton \( S \) from negation of specification

From product state \( s = (r, q) \) with \( r \in A \) (system) and \( q \in S \) (spec):
– consider only those successors of \( r \) labeled the same as transitions from \( q \)
– if counterexample found, terminate without exploring entire state space

Partial order reduction methods

Basic idea: build reduced model
– state space and execution paths are subsets of full (original) model
– preserves the same properties as original model

– must determine an equivalence relation between paths
– such that specification cannot distinguish between equivalent paths
– reduced model should contain a representative from each equivalence class

Method named initially after partial ordering of executed transitions

More generic term: model checking using representatives

An intuitive view

Visible transitions

Visibility (with respect to \( AP' \subseteq AP \))
\( \alpha \in T \) invisible \( \Rightarrow \forall s, s' \in S, s' = \alpha(s) \Rightarrow L(s) \cap AP' = L(s') \cap AP' \)
(\( AP' \) does not change labeling with propositions from \( AP' \))
typically: \( AP' \) = atomic propositions from specification

Asynchronous composition ⇒ arbitrary ordering of concurrent events
⇒ \( n \) transitions generate \( n! \) orderings and \( 2^n \) states
⇒ combinatorial (exponential) “explosion” of resulting state space
In asynchronous composition, the next-time operator X is not relevant:

– Two transitions in different components can occur in any order.
– Two transitions in the same component can be separated by arbitrarily many transitions in other components ⇒ the local state stays the same.

Two infinite paths π₁, π₂, . . . are stuttering equivalent π₁ ≈ π₂ if they can be split into pairwise corresponding finite blocks of identically labelled states.

∃ finite sequences 0 = i₀ < i₁ < . . . and 0 = j₀ < j₁ < . . . s.t. ∀k ≥ 0 L(πₖ) = L(πₖ₊₁) = . . . = L(πₖ₊ₙ) = L(πₖ₊ₙ₊₁) = . . . = L(πₖ₊ₙ₊ₙ). An LTL formula A₁ is stuttering invariant if ∀π₁, π₂ with π₁ ≈ π₂, f ⊨ π₁ ⇒ f ⊨ π₂.

Theorem: Any LTL formula (without the X operator) is a stuttering-invariant property, and conversely.

**Partial order reduction. Model checking with automata**

**Stuttering invariance**

The reduced model is constructed selecting from each state only a subset of the transitions enabled in that state.

Selection is made keeping for every path from the original model M a stuttering-equivalent path in the reduced model M′.

⇒ ∀Af ∈ LTL X M ⇒ M′f ⇒ A′f

Various names and selection criteria: stubborn sets [Valmari], persistent sets [Godefroid]; utilitām ample sets [Peled].

Selection of transitions: expressed by a set of conditions:

**C0**: ample(s) = ∅ ⇔ enabled(s) = ∅

successor in original model ⇒ there exists successor in reduced model

– Two transitions in different components can occur in any order.

**C1** A path from s cannot execute a transition dependent on a transition from ample(s) before executing a transition from ample(s).

Property: Transitions from ample(s) are independent of those in enabled(s) \ ample(s)

⇒ any transition from a state s has one of the forms:
– a prefix α₁, α₂, . . . , αₙβ, where β ∈ ample(s), and α₁ independent of β
– an infinite sequence α₁α₂ . . . with α₁ independent of any β ∈ ample(s)

**C2** (Invisibility) ample(s) ≠ enabled(s) ⇒ ample(s) ⊆ invisible(s)

If s is not explored completely all transitions from ample(s) are invisible.

Formal verification. Lecture 4 Marius Minea

**Reduction conditions**

**C3** A transition activated in all states in a cycle must be included in ample(s) for at least one state s of the cycle.

– guarantees that no portion of the state space is unexplored because of persistent ignoring of a transition
– implementation: in any cycle, a state is explored completely

**Reduction principle**

For the path π from s, we construct an equivalent path π′ in the reduced model:

**a)** if the next transition in π is in ample(s), we add it to π′

**b)** if the next transition in π is not in ample(s)

⇒ cf. **C2** transitions from ample(s) are invisible (3 transitions if ample(s))

**b1)** if in π there is some transition β ∈ ample(s), we add it to π′

– cf. **C1**: β independent of previous transitions

– it’s invisible, thus commuting it doesn’t affect spec

**b2)** there are no transitions from ample(s) in π

⇒ add arbitrary transition β ∈ ample(s) to π′

– cf. **C1**: it does not enable successive transitions

– it’s invisible ⇒ does not affect spec

– cf. **C3** this case appears a finite number of times

**Constructing an equivalent path**

– Conditions cannot be verified directly ⇒ conservative heuristics

– Transitions reading and writing a shared variable are dependent

– Conditional choices in the same process are dependent

– Communication transitions enter dependencies in both processes

– Send operations on the same buffer are dependent.

Likewise, for receives from the same buffer.

Transitions with disjunct process sets are independent

⇒ select a set P of processes which in the current state do not have communication operations with processes outside P

⇒ ample(s) = active transitions from P

Ideally: few transitions in ample(s) (e.g. local transitions in a process)
Formal verification. Lecture 4
Marius Minea

Formal verification. Lecture 4
Marius Minea

Formal verification. Lecture 4
Marius Minea

Relation between implementation and specification
We’ve discussed so far:
implementation (model): finite-state automaton
specification: formula in temporal logic (LTL, CTL)
Another view:
- specification is also an automaton
- with “fewer details” than the implementation
- model checking for LTL: by converting formula to automaton

Model checking for LTL
General idea:
- we check formulas $A f$ ($f$ = path formula
in which the only state subformulas are atomic propositions)
- $A f = A \neg \neg f \Rightarrow$ enough to consider $E f$
- we construct a tableau $T$ for the formula $f$ = an automaton
(Kripke structure) that expresses all paths that satisfy $f$
- we compose the model $M$ with the tableau $T$
- we check if there exists a path in the composition
(with CTL model checking algorithms)

Satisfaction relation in the tableau
We associate to every subformula of $f$ a set of states from $T$
(intuitively: set of states that satisfy the formula)
- sat($g$) = \{ $s$ | $g \in s$\} for $g \in el(f)$
- sat($\neg g$) = \{ $s$ | $s \notin sat(g)$\}
- sat($g_1 \lor g_2$) = sat($g_1$) $\cup$ sat($g_2$)
- sat($g_1 U g_2$) = sat($g_2$) $\cup$ (sat($g_1$) $\cap$ sat($X g_2$))

$T$ Transition relation: must be consistent with semantics of $X$
- $X_p \in s \Rightarrow \forall s'$. $R(s, s')$ $\Rightarrow$ $g \in s'$
- $X_p \notin s \Rightarrow \forall s'$. $R(s, s')$ $\Rightarrow$ $g \notin s'$
- $R_T(s, s') = \bigwedge_{s \in sat(X_p)} s \in sat(g)$

Constructing the tableau. Elementary formulas
Let $AP_f$ be the set of atomic propositions that appear in $f$.
$T = (S_T, R_T, L_T)$, cu $L_T: S_T \rightarrow 2^{AP_f}$
Tableau states: sets of elementary formulas extracted from $f$.
- $el(p) = \{ p \}$ for $p \in AP_f$
- $el(\neg g) = el(g)$
- $el(g_1 \lor g_2) = el(g_1) \cup el(g_2)$
- $el(X_p) = \{ X_p \} \cup el(g)$
- $el(g_1 U g_2) = \{ X(g_1 U g_2) \} \cup el(g_1) \cup el(g_2)$

Set of tableau states: $S_T = T(el(f))$

Computing the product
Define $T \times M = (S_T \times S, R_T, L_T, S_0) = (S \times B, B, L)$
- $S = \{ (s_T, s_M) | s_T \in S_T, s_M \in S_M, L_T(s_T) \subseteq L_M(s_M) \cap AP_f \}$
- $R(s_T, s'_T, s_M, s'_M) = R_T(s_T, s'_T) \land R_M(s_M, s'_M)$
- $L((s_T, s_M)) = L_M(s_M)$

(simultaneous transitions, only for identically labeled states)
Product: restricted to states from which there is at least one transition

Problem: $T$ does not guarantee liveness (eventuality) properties:
$R$ ensures sat($\nu h$) continually sat($h$), but not also $F$ sat($h$)
$\Rightarrow$ model checking with fairness: sat($\nu h \land h \land \nu g U h$) in $f$

Theorem: $M, s_M \models E f \Rightarrow \exists y_T \in sat(f), P(s_T, s_M) \models E G True$
with fairness conditions $sat(\nu g U h) \Rightarrow h \land \nu g U h$ in $f$