Partial order reduction

Model checking with automata

27 October 2005

Model checking "on-the-fly"

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

Specifications given as automata can *guide* verification algorithms:
$\Rightarrow$ only the needed parts of state space are constructed

Approach: build automaton $\mathcal{S}$ from negation of specification

From product state $s = (r, q)$ with $r \in \mathcal{A}$ (system) and $q \in \mathcal{S}$ (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

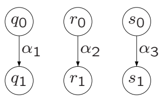
Basic idea: build *reduced* model
– state space and execution paths are subsets of full (original) model
– preserves the same properties as original model

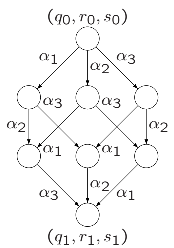
Approach is sound if exluded states/paths bring no extra information
– must determine an *equivalence relation* between paths
– such that specification cannot distinguish between equivalent paths
– reduced model should contain a representative from each equivalence class

Method named initially after *partial ordering* of executed transitions

More generic term: model checking using representatives

An intuitive view





Asynchronous composition $\Rightarrow$ arbitrary ordering of concurrent events
$\Rightarrow$ n transitions generate $n!$ orderings and 2^n states
$\Rightarrow$ combinatorial (exponential) "explosion" of resulting state space

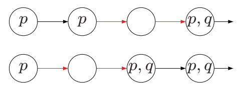
Tranzitions. Dependence and independence

Model: state-transition system (S, T, S_0, L)
A transition $\alpha \in T$ is a *subset* $\alpha \subseteq S \times S$
(viewed as a family of transitions with the same label)
Transition is *enabled* in s: $\alpha \in enabled(s) \Leftrightarrow \exists s' \in S \,.\, \alpha(s, s')$
We consider only *deterministic* transitions: $\forall \alpha, s\ \exists! s' \,.\, \alpha(s, s')$
– the system may still be nondeterministic if $|enabled(s)| > 1$

Independence: two conditions, $\forall s \in S$:
Enabling: $\alpha, \beta \in enabled(s) \Rightarrow \alpha \in enabled(\beta(s)) \land \beta \in enabled(\alpha(s))$
– two independent transitions do not *disable* each other
– but one may lead to the other being enabled
Commutativity: $\alpha, \beta \in enabled(s) \Rightarrow \alpha(\beta(s)) = \beta(\alpha(s))$
– effect of execution same, regardless of ordering

Visible transitions

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



Stuttering invariance

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 $\pi = s_0 s_1 \ldots$ and $\pi' = r_0 r_1 \ldots$ are *stuttering equivalent* $\pi \sim_{st} \pi'$ if they can be split into pairwise corresponding finite blocks of identically labelled states

∃ infinite sequences $0 = i_0 < i_1 < \ldots$ and $0 = j_0 < j_1 < \ldots$, a.î. $\forall k \geq 0$
$L(s_{i_k}) = L(s_{i_k+1}) = \ldots L(s_{i_{k+1}-1}) = L(r_{j_k}) = L(r_{j_k+1}) = \ldots L(r_{j_{k+1}-1})$

An LTL formula $\mathbf{A}f$ is *stuttering invariant* if $\forall \pi, \pi'$ with $\pi \sim_{st} \pi'$, $\pi \models f \Leftrightarrow \pi' \models f$

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

Reduction principle

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'.
⇒ $\forall \mathbf{A}f \in LTL_{-X} \quad M \models \mathbf{A}f \Leftrightarrow M' \models \mathbf{A}f$

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

Selection of transitions: expressed by a set of conditions:

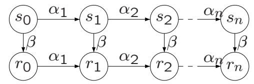
C0: $ample(s) = \emptyset \Leftrightarrow enabled(s) = \emptyset$
successor in original model ⇒ there exists successor in reduced model

Reduction conditions

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 $enabled(s) \setminus ample(s)$

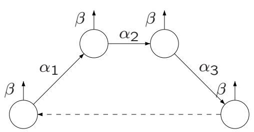
⇒ any transition from a state s has one of the forms:
– a prefix $\alpha_1\alpha_2 \ldots \alpha_n\beta$, where $\beta \in ample(s)$, and α_i independent of β
– an infinite sequence $\alpha_0\alpha_1 \ldots$, with α_i independent of any $\beta \in ample(s)$



C2 (Invisibility) $ample(s) \neq enabled(s) \Rightarrow ample(s) \subseteq invisible(s)$
If s is not explored completely all transitions from $ample(s)$ are invisible.

Reduction conditions (cont'd)

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 persistenti ignoring of a transition
– implementation: in any cycle, a state is explored completely

Constructing an equivalent path

For the path π from s, we construct an equivalent path π' in the reduced model:
a) if the next transition 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 (∃ transitions $\notin ample(s)$)
b1) if in π there is some transition $\beta \in ample(s)$, 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 $\beta \in ample(s)$ 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

Selecting transitions in practice

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 disjoit 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)

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 $\mathbf{A}f$ $\quad$ (f = path formula
in which the only state subformulas are atomic propositions)
– $\mathbf{A}f = \neg\mathbf{E}\neg f \Rightarrow$ enough to consider $\mathbf{E}f$.
– 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)

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 $L_T : S_T \to 2^{AP_f}$.
Tableau states: sets of *elementary formulas* extracted from f.
- $el(p) = \{p\}$ for $p \in AP_f$
- $el(\neg g) = el(g)$
- $el(g_1 \vee g_2) = el(g_1) \cup el(g_2)$
- $el(\mathbf{X}g) = \{\mathbf{X}g\} \cup el(g)$
- $el(g_1 \mathbf{U} g_2) = \{\mathbf{X}(g_1 \mathbf{U} g_2)\} \cup el(g_1) \cup el(g_2)$

Set of tableau states: $S_T = \mathcal{P}(el(f))$

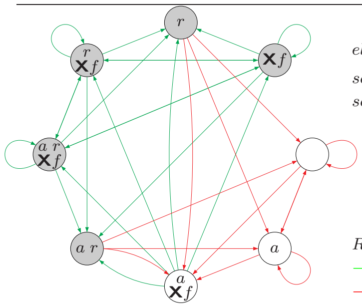
Satisfaction relation in the tableau

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

TTransition relation: must be consistent with semantics of $\mathbf{X}$
– $\mathbf{X}g \in s \to \forall s' . R(s, s') \to g \in s'$
– $\mathbf{X}g \notin s \to \forall s' . R(s, s') \to g \notin s'$

$$R_T(s, s') = \bigwedge_{\mathbf{X}g \in el(f)} s \in sat(\mathbf{X}g) \Leftrightarrow s' \in sat(g)$$

An example: $f = (\neg ack)\mathbf{U}recv$



$el(f) = \{a, r, \mathbf{X}f\}$
$sat(f) =$ $=$
$sat(r) \cup (\neg sat(a) \cap sat(\mathbf{X}f))$

$R_T = \longrightarrow \cup \longrightarrow$
$\longrightarrow = sat(\mathbf{X}f) \times sat(f)$
$\longrightarrow = \neg sat(\mathbf{X}f) \times \neg sat(f)$

Computing the product

Definim $T \times M = (S_T, R_T, L_T) \times (S_M, R_M, L_M) = (S, R, L) = P$
- $S = \{(s_T, s_M) \mid s_T \in S_T, s_M \in S_M, L_T(s_T) = L_M(s_M) \cap AP_f\}$
- $R((s_T, s_M), (s'_T, s'_M)) = R_T(s_T, s'_T) \wedge R_M(s_M, s'_M)$
- $L((s_T, s_M)) = L_T(s_T)$

(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_T ensures $sat(g\mathbf{U}h)$ continually $sat(h)$, but not also $\mathbf{F}sat(h)$
$\Rightarrow$ model checking with *fairness*: $\{sat(g\mathbf{U}h) \to h \mid g\mathbf{U}h$ apare în $f\}$

Theorem: $M, s_M \models \mathbf{E}f \Leftrightarrow \exists s_T \in sat(f) . P, (s_T, s_M) \models_F \mathbf{EG}\,True$
with fairness conditions $\{sat(g\mathbf{U}h) \to h \mid g\mathbf{U}h$ apare în $f\}$