Partial Order Reduction (Recitation)

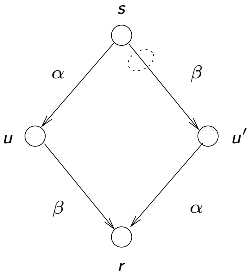
David Henriques

Carnegie Mellon University
Many slides by Deepak D'Souza (thanks!)

December 2nd 2011

What and Why

For specifications that don't worry about the order of "independent" transitions, it should be enough to explore just *one* of the interleaved sequences of independent transitions.



Definitions

- Transitions α and β are said to be *independent* if
  - (Enabledness) Neither α nor β can disable the other. That is if α and β are both enabled in a state s, and $s \stackrel{\alpha}{\to} s'$, then β is still enabled at s' (and vice versa).
  - (Commutativity) The sequence of transitions $\alpha\beta$ and $\beta\alpha$ lead to the same state.
- A transition α is said to be *invisible* wrt a property that uses a set of propositions P if: whenever $s \stackrel{\alpha}{\to} s'$ we have

$$\forall p \in P : \; s \models p \text{ iff } s' \models p.$$

(In)Dependence

In different models of computation, different situations generate dependent transitions:

- Pairs of transitions that share a variable, which is changed by at least one of them.
- Pairs of transitions belonging to the same process.
- Send and recieve transitions that use the same message queue.

Stutter-free LTL

- Two state sequences π and π' are said to be stutter-equivalent, written $\pi \sim_{st} \pi'$, iff the sequence of distinct states is identical in π and π'.
- An LTL formula φ is said to be stuffer-free iff for each pair of stutter-equivalent state sequences π and π', we have

$$\pi \models \varphi \text{ iff } \pi' \models \varphi.$$

- The X-free fragment of LTL given by the syntax

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi U \varphi. \qquad (\text{no } X\varphi)$$

  is stutter-free.
- Conversely, every LTL property that is stutter-free can be expressed in the X-free fragment of LTL.

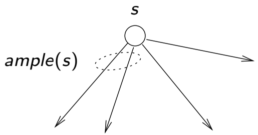
Objective

We want to generate a reduced state transition graph such that each path in the full state transition graph has a stutter-equivalent path in the reduced graph.

We build this graph by DFSing the full state graph except that we *disallow some transitions*. This is called static reduction.

Ample sets

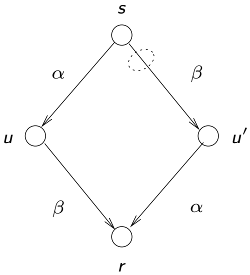
Each state s in the model has several transitions. We call that set *enabled*(s).
We want to identify a subset of transitions enabled at s, called *ample*(s).



In the DFS search we will only consider neighbours of s that arise out of transition in *ample*(s).

Problems with reduction

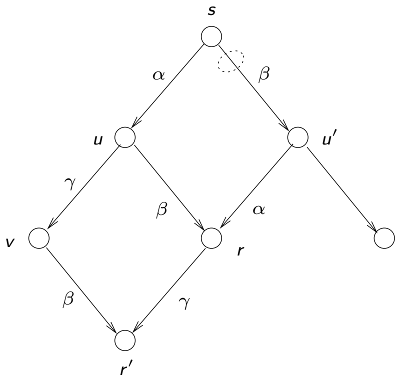
Suppose *ample*(s) is chosen to be $\{\beta\}$.



Problems:

- The state sequence s, u, r is ignored.
- What if we had a transition γ enabled only at u? It (and the paths starting from it) would also not be explored.

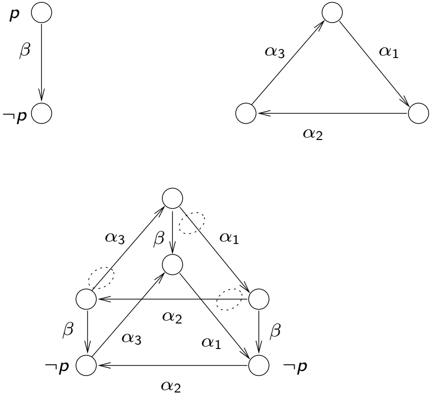
Conditions on ample sets help



Conditions:

- ($\mathbf{C_0}$) Ensure $\mathit{enabled}(s) \neq \emptyset$ implies $\mathit{ample}(s) \neq \emptyset$.
- ($\mathbf{C_1}$) Ensure that no transition dependent on a β in ample set at s can fire without some transition in $\mathit{ample}(s)$ happening first.
- ($\mathbf{C_2}$) Ensure $\mathit{ample}(s)$ contains only transitions invisible wrt the property being checked.

Conditions still not sufficient



Add Condition:

- ($\mathbf{C_3}$) Ensure that no cycle in reduced graph, and a transition β, which is always enabled in the cycle but never included in the ample sets along the cycle.

Correctness claim

Theorem (Correctness of reduction)

If ample sets are chosen satisfying conditions ($\mathbf{C_0}$) *to* ($\mathbf{C_3}$)*, then the reduced graph contains a stutter-equivalent path for every path in the original graph.*
Hence model checking wrt stutter-free LTL properties can be done soundly on the reduced graph.

Warning

We are actually checking for stronger conditions than the ones needed. However, these procedures are easy to check. There is a tradeoff between how precise we wanto to be and how much computational power we are willing to pay

The fallback plan

If everything else fails, we can always take $\mathit{ample}(s) = \mathit{enabled}(s)$!

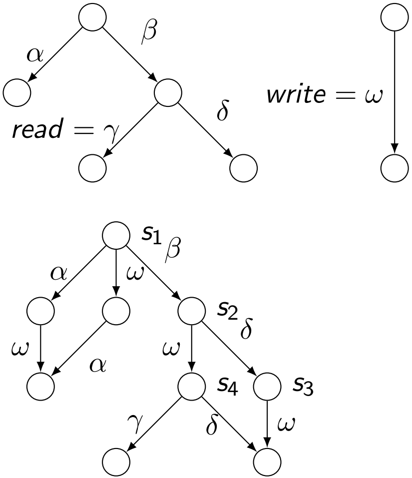
Notation

Assume the existence of several processes P_i

- $pc_i(s)$ denotes the program counter of process P_i;
- $pre(\alpha)$ is the set of transitions that may enable α;
- $dep(\alpha)$ is the set of transitions dependent on α;
- T_i is the set of transitions of process P_i
- $T_i(s)$ is the set of transitions of P_i enabled in s;
- $current_i(s)$ is the set of transitions of P_i enabled in some state s' s.t. $pc_i(s') = pc_i(s)$.

(Note: transitions in $T_i(s)$ are interdependent since they are from the same process)

Notation



- $pc_1(s_3) = 2$
- $pc_2(s_3) = 0$
- $T_1 = \{\alpha, \beta, \gamma, \delta\}$
- $T_2 = \{\omega\}$
- $pre(\delta) = \{\beta, \omega\}$
- $T_1(s_2) = \{\delta\}$
- $current_1(s_2) = T_1(s_2) \cup T_1(s_4) = \{\gamma, \delta\}$

The actual heuristic

Take the set $T_i(s)$ as a candidate for *ample*(s) for some i

- Conditions $\mathbf{C_0}$ and $\mathbf{C_2}$ are easy to check.
- Condition $\mathbf{C_3}$ is harder, but we'll take care of that later.
- Condition $\mathbf{C_1}$ is the complicated one...

Suppose that condition $\mathbf{C_1}$ is violated. Then some transitions independent of $T_i(s)$ are executed, enabling a transition α dependent in $T_i(s)$ in a state s'

First case

If α is in $P_j \neq P_i$...

Necessarily $dep(T_i(s))$ includes α

This is easy to check by examining the dependency relation!

Second case

If α is in P_i...

- Transitions to s' are from other processes because they are independent from $T_i(s)$;
- Then $pc_i(s') = pc_i(s)$, so $\alpha \in current_i(s)$;
- Also $\alpha \notin T_i(s)$ (otherwise does not violate $\mathbf{C_1}$);
- So $\alpha \in current_i(s) \setminus T_i(s)$;
- So a necessary condition to violate $\mathbf{C_1}$ is that $pre(current_i(s) \setminus T_i(s))$ includes transitions from processes other than P_i;

This is also easy to check by examining the dependency relation!

What about C_3?

We check for a stronger condition

A sufficient condition for $\mathbf{C_3}$ is that at least one state along each cycle is such that $ample(s) = enabled(s)$.

This condition is also easy to check.

Back to the heuristic

If any of these conditions fails, we give up on this $T_i(s)$ and try a different one.

If they all fail, we go to our fallback plan and take $ample(s) = enabled(s)$