IA159 Formal Verification Methods

Partial Order Reduction

Jan Strejček

Faculty of Informatics
Masaryk University

Focus and sources

Focus

- stuttering principle
- theory of partial order reduction
- heuristics for efficient implementation

Source

- Chapter 10 of *E. M. Clarke, O. Grumberg, and D. A. Peled: Model Checking, MIT, 1999.*

Basic facts on partial order reduction

- compatible with model checking of finite systems against LTL formulae without X operator
- size of the reduced system is 3–99% of the original size
- model checking process for reduced systems is faster and consumes less memory
- best suited for asynchronous systems
- also known as model checking using representatives

Modified definition of Kripke structure

We consider only deterministic systems.

A Kripke structure is a tuple $M = (S, T, S_0, L)$, where

- S is a finite set of states
- T is a set of transitions, each $\alpha \in T$ is a partial function $\alpha : S \to S$.
- $S_0 \subseteq S$ is a set of initial states
- $L : S \to 2^{AP}$ is a labelling function associating to each state $s \in S$ the set of atomic propositions that are true in s.

- a transition α is enabled in s if $\alpha(s)$ is defined
- α is disabled in s otherwise
- $enabled(s)$ denotes the set of transitions enabled in s

More definitions

Let φ be an LTL formula and $K = (S, T, S_0, L)$ be a Kripke structure.

- $AP(\varphi)$ is the set of atomic propositions occurring in φ
- a path in K starting from a state $s \in S$ is an infinite sequence $\pi = s_0, s_1, \ldots$ of states such that $s_0 = s$ and for each i there is a transition $\alpha_i \in T$ such that $\alpha_i(s_i) = s_{i+1}$
- a path starting in a fixed state can be identified with a sequence of transitions
- a path π satisfies φ, written $\pi \models \varphi$, if $w \models \varphi$, where the word $w = w(0)w(1)\ldots$ is defined as $w(i) = L(s_i) \cap AP(\varphi)$ for all $i \geq 0$
- K satisfies φ, written $K \models \varphi$, if all paths starting from initial states of K satisfy φ

Goal of partial order reduction

$\mathrm{LTL}_{-\mathrm{X}}$ denotes LTL formulae without X operator.

Goal

Given a finite Kripke structure K and an $\mathrm{LTL}_{-\mathrm{X}}$ formula φ, we want to find a smaller Kripke structure K' such that

$$K \models \varphi \iff K' \models \varphi.$$

Reduction method

- K' arises from K by disabling some transitions in some states
- as a result, some states may become unreachable in K'
- for each state s, $ample(s)$ denotes the set of transitions that are enabled in s in K', $ample(s) \subseteq enabled(s)$
- calculation of ample sets needs to satisfy three goals
  1. K' given by ample sets has to satisfy

     $$K \models \varphi \iff K' \models \varphi$$

  2. K' should be substantially smaller than K
  3. the overhead in calculating ample sets must be small

A base of partial order reduction

Stuttering principle

Stuttering on words

- let $w = w(0)w(1)w(2)\dots$ be an infinite word
- a letter $w(i)$ is called redundant iff $w(i) = w(i+1)$ and there is $j > i$ such that $w(i) \neq w(j)$
- canonical form of w is the word obtained by deleting all redundant letters from w
- infinite words w_1, w_2 are stutter equivalent, written $w_1 \sim w_2$, iff they have the same canonical form

Example

- canonical form of $kk\,k\,oooo\,o\,m\,k\,k.n^\omega$ is $komk.n^\omega$
- canonical form of $k\,oo\,o\,mmmmm\,m\,kkk\,k.n^\omega$ is $komk.n^\omega$
- hence $kkkoooomkk.n^\omega \sim kooommmmmmkkkk.n^\omega$

Stuttering principle

Theorem (Lamport 1983)

Let φ be an LTL_{-X} formula and w_1, w_2 be two stutter equivalent words. Then

$$w_1 \models \varphi \iff w_2 \models \varphi.$$

Stuttering on paths

Paths $\pi = s_0 s_1 \ldots$ and $\pi' = s'_0 s'_1 \ldots$ are stutter equivalent with respect to a set $AP' \subseteq AP$, written $\pi \sim_{AP'} \pi'$, iff $w \sim w'$, where w, w' are defined as $w(i) = L(s_i) \cap AP'$ and $w'(i) = L(s'_i) \cap AP'$ for each i.

Kripke structures K, K' are stutter equivalent with respect to AP', written $K \sim_{AP'} K'$, iff

- K and K' have the same set of initial states and
- for each path π of K starting in an initial state s there exists a path π' of K' starting in the same initial state such that $\pi \sim_{AP'} \pi'$ and vice versa.

Stuttering principle for Kripke structures

Corollary

Let φ be an $\mathsf{LTL}_{-\mathsf{X}}$ formula and K, K' be Kripke structures such that $K \sim_{AP(\varphi)} K'$. Then

$$K \models \varphi \iff K' \models \varphi.$$

Stuttering principle for Kripke structures

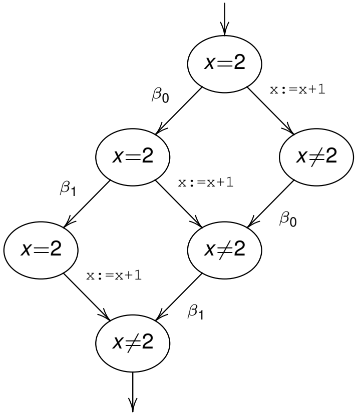
Corollary

Let φ be an $\mathsf{LTL}_{-\mathsf{X}}$ formula and K, K' be Kripke structures such that $K \sim_{AP(\varphi)} K'$. Then

$$K \models \varphi \iff K' \models \varphi.$$

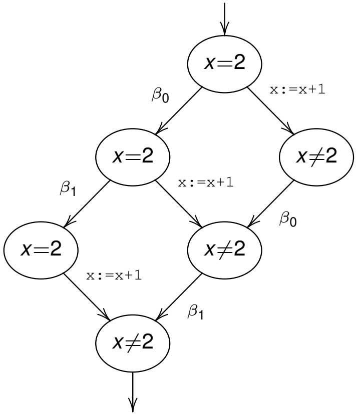
Hence, for every set of stutter equivalent paths (with respect to $AP(\varphi)$) of K it is sufficient to keep at least one representant of these paths in K'.

Example

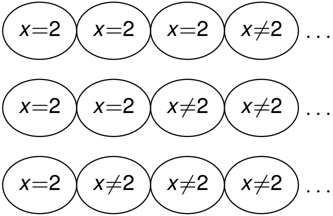


Let $AP(\varphi)$ contain just $x = 2$.

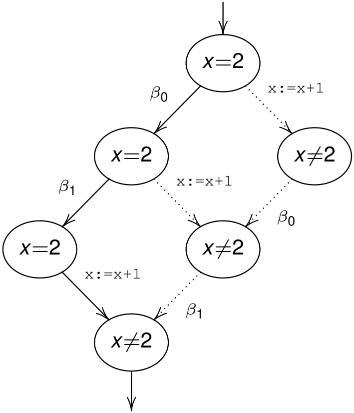
Example



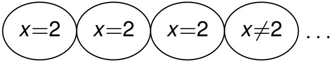
Let $AP(\varphi)$ contain just $x = 2$.



Example



Let $AP(\varphi)$ contain just $x = 2$.



Conditions on ample sets

Terminology: (in)visibility and full expansion

A transition $\alpha \in T$ is invisible if for each pair of states $s, s' \in S$ such that $\alpha(s) = s'$ it holds that

$$L(s) \cap AP(\varphi) = L(s') \cap AP(\varphi).$$

A transition is visible if it is not invisible.

Terminology: (in)visibility and full expansion

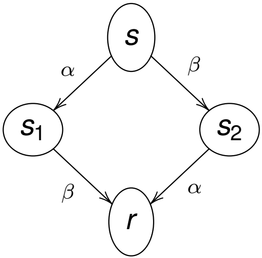
A transition $\alpha \in T$ is invisible if for each pair of states $s, s' \in S$ such that $\alpha(s) = s'$ it holds that

$$L(s) \cap AP(\varphi) = L(s') \cap AP(\varphi).$$

A transition is visible if it is not invisible.

A state s is fully expanded when $\mathit{ample}(s) = \mathit{enabled}(s)$.

Terminology: (in)dependence



An independence relation $I \subseteq T \times T$ is a symmetric and antireflexive relation satisfying the following two conditions for each state $s \in S$ and for each $(\alpha, \beta) \in I$:

1. enabledness: if $\alpha, \beta \in \mathit{enabled}(s)$ then $\alpha \in \mathit{enabled}(\beta(s))$
2. commutativity: if $\alpha, \beta \in \mathit{enabled}(s)$ then $\alpha(\beta(s)) = \beta(\alpha(s))$

The dependency relation D is the complement of I.

Condition C0

If all ample sets satisfy the following conditions C0, C1, C2, and C3, then $K' \sim_{AP(\varphi)} K$.

Condition C0

If all ample sets satisfy the following conditions C0, C1, C2, and C3, then $K' \sim_{AP(\varphi)} K$.

C0

$ample(s) = \emptyset \iff enabled(s) = \emptyset$.

Condition C1

C1

Along every path in the original structure that starts in s, the following condition holds: a transition outside $ample(s)$ and dependent on a transition in $ample(s)$ cannot be executed without a transition in $ample(s)$ occuring first.

Condition C1

C1

Along every path in the original structure that starts in s, the following condition holds: a transition outside *ample*(s) and dependent on a transition in *ample*(s) cannot be executed without a transition in *ample*(s) occuring first.

Lemma

If C1 holds, then the transitions in enabled(s) $\setminus$ *ample*(s) *are all independent of those in ample*(s).

Condition C1

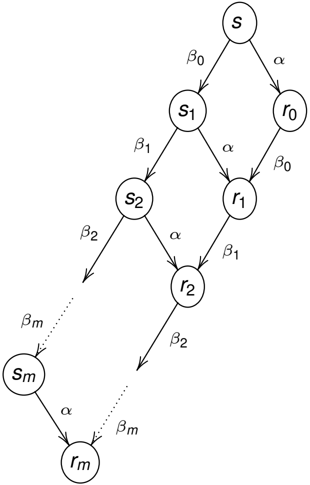
C1

Along every path in the original structure that starts in s, the following condition holds: a transition outside *ample*(s) and dependent on a transition in *ample*(s) cannot be executed without a transition in *ample*(s) occuring first.

Thanks to C1, all paths of K starting in a state s and not included in K' have one of the following two forms:

- the path has a prefix $\beta_0\beta_1 \ldots \beta_m\alpha$, where $\alpha \in$ *ample*(s) and each β_i is independent of all transitions in *ample*(s) including α.
- the path is an infinite sequence of transitions $\beta_0\beta_1 \ldots$ where each β_i is independent of all transitions in *ample*(s).

Condition C1: consequences



Due to C1, after execution of a sequence $\beta_0\beta_1\ldots\beta_m$ of a transitions not in *ample*(s) from s, all the transitions in *ample*(s) remain enabled. Further, the sequence $\beta_0\beta_1\ldots\beta_m\alpha$ executed from s leads to the same state as the sequence $\alpha\beta_0\beta_1\ldots\beta_m$.

As the sequence $\beta_0\beta_1\ldots\beta_m\alpha$ is not included in the reduced system, we want $\beta_0\beta_1\ldots\beta_m\alpha$ and $\alpha\beta_0\beta_1\ldots\beta_m$ to be prefixes of stutter equivalent paths. This is quaranteed if α is invisible.

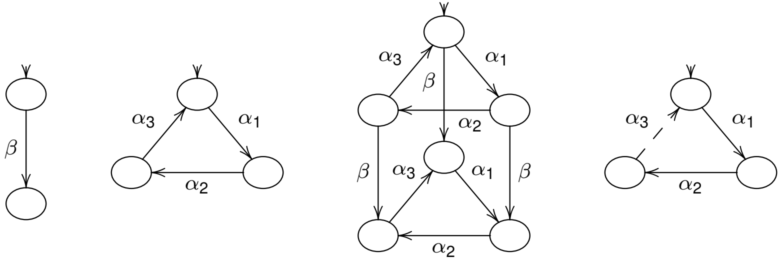
Condition C2

C2 (invisibility)

If s is not fully expanded, then every $\alpha \in \mathit{ample}(s)$ is invisible.

Condition C3: motivation

Conditions C0, C1, and C2 are not yet sufficient to guarantee that K' is stutter equivalent to K. There is a possibility that some transition will be delayed forever because of a cycle.



β is visible, $\alpha_1, \alpha_2, \alpha_3$ are invisible, β is independent of $\alpha_1, \alpha_2, \alpha_3$, and $\alpha_1, \alpha_2, \alpha_3$ are interdependent

Condition C3

C3 (cycle condition)

A cycle in reduced structure is not allowed if it contains a state in which some transition is enabled, but is never included in $ample(s)$ for any state s on the cycle.

Partial order reduction

Correctness

Statement

Theorem

Let φ be an $\mathsf{LTL}_{-\mathsf{X}}$ formula and K be a Kripke structure. If K' is a reduction of K satisfying C0–C3, then

$$K \sim_{AP(\varphi)} K'.$$

Terminology

- since now a path can be finite or infinite
- $\sigma \circ \eta$ the concatenation of a finite path σ and a (finite or infinite) path η ($\circ$ is applicable if the last state *last*(σ) of σ is the same as the first state of η)
- *tr*(π) denote the sequence of transitions on a path π
- for a (finite or infinite) sequence *v* of transitions, *vis*(*v*) denotes its projection onto the visible transitions

Construction

For every infinite path π of K starting in some initial state we construct an infinite sequence of paths

$$\pi = \pi_0, \quad \pi_1, \quad \pi_2, \quad \pi_3, \quad \dots$$

where, for each i, $\pi_i = \sigma_i \circ \eta_i$ such that $|\sigma_i| = i$.

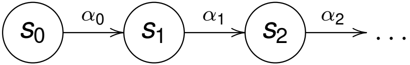
Construction

For every infinite path π of K starting in some initial state we construct an infinite sequence of paths

$$\pi = \pi_0, \quad \pi_1, \quad \pi_2, \quad \pi_3, \quad \ldots$$

where, for each i, $\pi_i = \sigma_i \circ \eta_i$ such that $|\sigma_i| = i$.

$\pi = \pi_0 =$



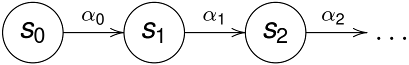
Construction

For every infinite path π of K starting in some initial state we construct an infinite sequence of paths

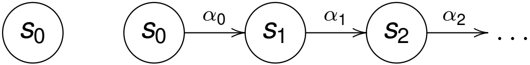
$$\pi = \pi_0, \quad \pi_1, \quad \pi_2, \quad \pi_3, \quad \ldots$$

where, for each i, $\pi_i = \sigma_i \circ \eta_i$ such that $|\sigma_i| = i$.

$\pi = \pi_0 =$

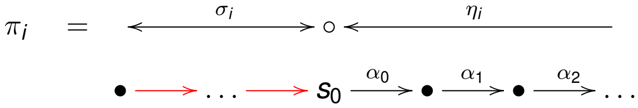


$\pi_0 = \quad \sigma_0 \quad \circ \quad \eta_0$

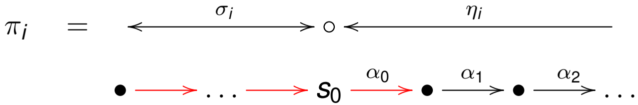


Construction of π_{i+1}

Let s_0 be the last state of σ_i.
The construction of π_{i+1} depends on α_0.

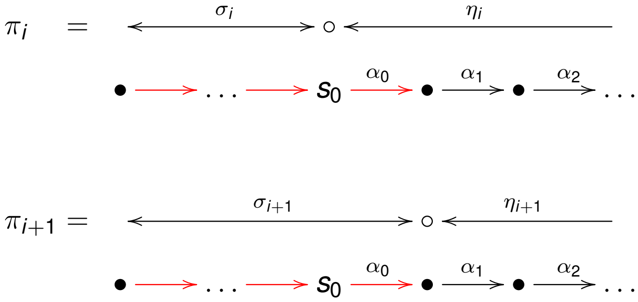


Case A $\alpha_0 \in \mathit{ample}(s_0)$.



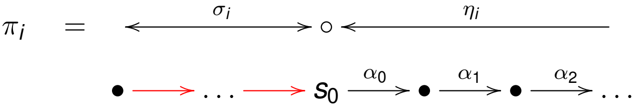
Construction of π_{i+1}

Case A $\alpha_0 \in \mathit{ample}(s_0)$.



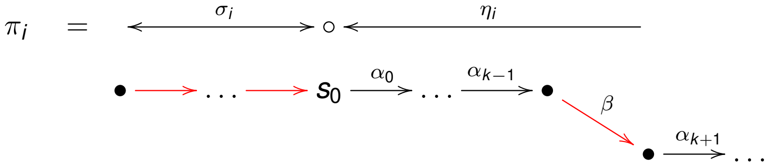
Construction of π_{i+1}

Case B $\alpha_0 \notin \mathit{ample}(s_0)$.
By C2, all transitions in $\mathit{ample}(s_0)$ must be invisible.
Due to C0 and C1, there are two cases.



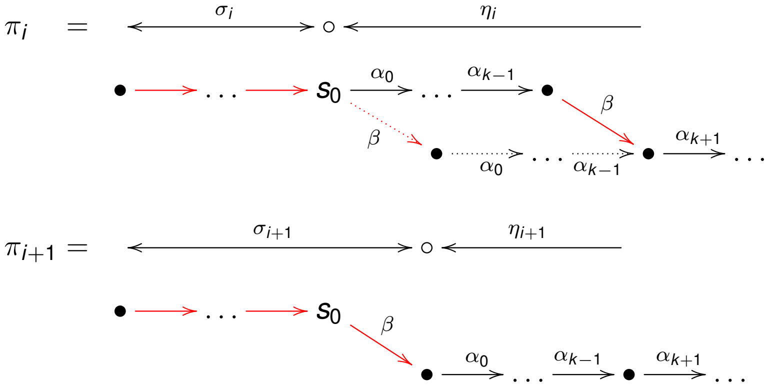
Construction of π_{i+1}

Case B1 $\alpha_0 \notin ample(s_0)$.
Some $\beta \in ample(s_0)$ appears on η_i after a finite sequence of independent transitions $\alpha_0\alpha_1 \ldots \alpha_{k-1}$.



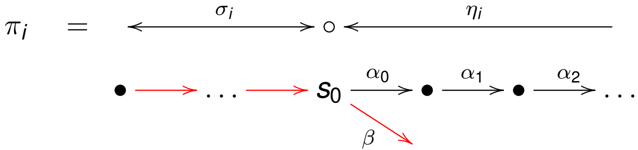
Construction of π_{i+1}

Case B1 $\alpha_0 \notin \mathit{ample}(s_0)$.
Some $\beta \in \mathit{ample}(s_0)$ appears on η_i after a finite sequence of independent transitions $\alpha_0\alpha_1 \ldots \alpha_{k-1}$.



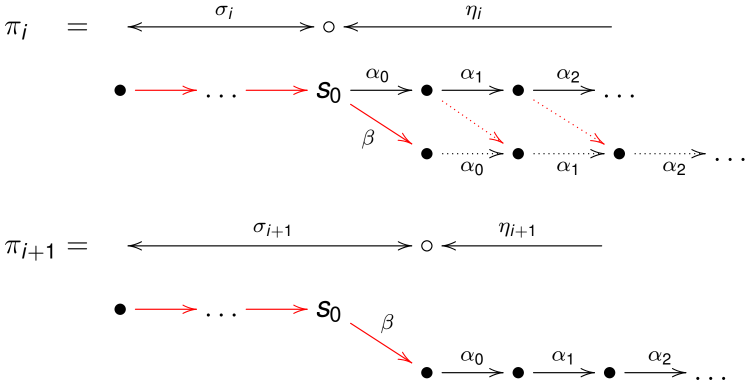
Construction of π_{i+1}

Case B2 $\alpha_0 \notin \mathit{ample}(s_0)$.
Some $\beta \in \mathit{ample}(s_0)$ is independent of all transitions in η_i.



Construction of π_{i+1}

Case B2 $\alpha_0 \notin ample(s_0)$.
Some $\beta \in ample(s_0)$ is independent of all transitions in η_i.



Properties of $\pi_0, \pi_1, \pi_2, \ldots$

Lemma

For all π_i, π_j, it holds:

- $\pi_i \sim_{AP(\varphi)} \pi_j$
- $vis(tr(\pi_i)) = vis(tr(\pi_j))$
- *if ξ_i, ξ_j are prefixes of π_i, π_j satisfying $vis(tr(\xi_i)) = vis(tr(\xi_j))$, then*

$$L(last(\xi_i)) \cap AP(\varphi) = L(last(\xi_j)) \cap AP(\varphi).$$

(It is sufficient to prove it for π_i and π_{i+1}. And this is easy.)

Definition of σ

We define an infinite path σ as the limit of the finite paths σ_i.

To prove correctness of the reduction, we have to show that:

1. σ belongs to the reduced structure K'
2. $\sigma \sim_{AP(\varphi)} \pi$

(The first item follows directly from the construction of σ_i.)

Properties of σ

"Every transition of π eventually appears in σ."

Lemma

Let α be the first transition of η_i. There exists $j > i$ such that α is the last transition of σ_j and, for all $i \le k < j$, α is the first transition of η_k.

(This is a consequence of C3.)

Properties of σ

"Only invisible transitions are added to σ.
Visible transitions of π keep their order."

Lemma

Let γ be the first visible transition on η_i and $prefix_\gamma(\eta_i)$ be the maximal prefix of $tr(\eta_i)$ that does not contain γ. Then one of the following holds:

- *γ is the first action of η_i and the last transition of σ_{i+1}, or*
- *γ is the first visible transition of η_{i+1}, the last transition of σ_{i+1} is invisible, and $prefix_\gamma(\eta_{i+1}) \sqsubseteq prefix_\gamma(\eta_i)$.*

$v \sqsubseteq w$ denotes that $v = w$ or v can be obtained from w by erasing one or more transitions.

Properties of σ

Lemma

Let v be a prefix of $vis(tr(\pi))$. Then there exists a path σ_i such that $v = vis(tr(\sigma_i))$.

Lemma

$\sigma \sim_{AP(\varphi)} \pi$.

Hence, $K \sim_{AP(\varphi)} K'$.

Calculating ample sets

Complexity of checking conditions C0–C3

Conditions C0 and C2

C0

$ample(s) = \emptyset \iff enabled(s) = \emptyset$.

C2 (invisibility)

If s is not fully expanded, then every $\alpha \in ample(s)$ is invisible.

- conditions C0 and C2 are local: their validity depends just on $enabled(s)$ and $ample(s)$, not on the whole structure
- C0 can be checked in constant time
- C2 can be checked in linear time with respect to $|ample(s)|$

Condition C1

C1

Along every path in the original structure that starts in s, the following condition holds: a transition outside $ample(s)$ and dependent on a transition in $ample(s)$ cannot be executed without a transition in $ample(s)$ occuring first.

- checking C1 for a state s and a set $T \subseteq enabled(s)$ is at least as hard as checking reachability for K (reachability problem can be reduced to checking C1)
- we give a procedure computing a set of transitions that is guaranteed to satisfy C1
- computed sets do not have to be optimal: tradeoff efficiency Vs. amount of reduction

Condition C3

C3 (cycle condition)

A cycle in reduced structure is not allowed if it contains a state in which some transition is enabled, but is never included in $ample(s)$ for any state s on the cycle.

- C3 is also non-local
- in contrast to C1, C3 refers only to the reduced structure
- instead of checking C3, we formulate a stronger condition which is easier to check

Condition C3

Lemma

Assume that C1 holds for all ample sets along a cycle in a reduced structure. If at least one state along the cycle is fully expanded, then C3 hold for this cycle.

- C1 implies that each $\alpha \in \mathit{enabled}(s) \setminus \mathit{ample}(s)$ is independent of transitions in $\mathit{ample}(s)$
- $\alpha \in \mathit{enabled}(s) \setminus \mathit{ample}(s)$ is also enabled in the next state on the cycle in K'
- if the cycle contains a fully expanded state, then it surely satisfies C3

Condition C3'

If K' is generated using depth-first search strategy, then every cycle in K' has to contain a back edge (i.e. an edge going to a state on the search stack)

C3'

If s is not fully expanded, then no transition in $ample(s)$ may reach a state that is on the search stack.

- C3' can be checked efficiently during nestedDFS algorithm

Calculating ample sets

Algorithm

Basic information

Reduced system is constructed on-the-fly: $ample(s)$ is computed only when a model checking algorithm needs to know successors of s.

Algorithm computing ample sets depends on the model of computation. We consider processes with

- shared variables and
- message passing with queues.

Notation

- $pc_i(s)$ denotes the program counter of process P_i in a state s
- $pre(\alpha)$ is a set including all transitions β such that there exists a state s for which $\alpha \notin enabled(s)$ and $\alpha \in enabled(\beta(s))$
- $dep(\alpha)$ is the set of all transitions that are dependent on α
- T_i is the set of transitions of process P_i
- $T_i(s) = T_i \cap enabled(s)$
- $current_i(s)$ is the set of all transitions of P_i that are enabled in some s' such that $pc_i(s) = pc_i(s')$ (note that $T_i(s) \subseteq current_i(s)$)

Tradeoff

We do not compute the sets *pre*(α) and *dep*(α) precisely.
We preffer to efficiently compute over-approximations of these sets.

Computing *pre*(α)

- *pre*(α) includes the transitions of the processes that contain α and that can change a program counter to a value from which α can execute
- if the enabling condition for α involves shared variables, then *pre*(α) includes all other transitions that can change these shared variables
- if α sends or receives messages on some queue *q*, then *pre*(α) includes transitions of other processes that receive or send data through *q*, respectively

Computing *dep*(α)

- pairs of transitions that share a variable, which is changed by at least one of them, are dependent
- pairs of transitions belonging to the same process are dependent
- two receive transitions that use the same message queue are dependent
- two send transitions are also dependent (sending a message may cause the queue to fill)

Note that a pair of send and receive transitions in different processes are independent as they can potentially enable each other, but not disable.

Sketch of the algorithm

- C1 implies that transitions in $\mathit{enabled}(s) \setminus \mathit{ample}(s)$ are independent on those in $\mathit{ample}(s)$
- as transitions in $T_i(s)$ are interdependent, it holds

$$T_i(s) \subseteq \mathit{ample}(s) \;\vee\; T_i(s) \cap \mathit{ample}(s) = \emptyset$$

- hence, $T_i(s)$ is a good candidate for $\mathit{ample}(s)$

Sketch of the algorithm

- C1 implies that transitions in $\mathit{enabled}(s) \setminus \mathit{ample}(s)$ are independent on those in $\mathit{ample}(s)$
- as transitions in $T_i(s)$ are interdependent, it holds

$$T_i(s) \subseteq \mathit{ample}(s) \ \vee \ T_i(s) \cap \mathit{ample}(s) = \emptyset$$

- hence, $T_i(s)$ is a good candidate for $\mathit{ample}(s)$

Idea of the algorithm

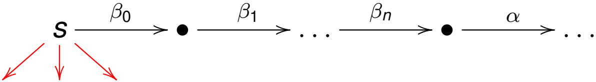
We check whether some $T_i(s) \neq \emptyset$ satisfies the conditions C1, C2, and C3'. If there is no such $T_i(s)$, we set $\mathit{ample}(s) = \mathit{enabled}(s)$.

Checking C1

C1

Along every path in the original structure that starts in s, the following condition holds: a transition outside *ample*(s) and dependent on a transition in *ample*(s) cannot be executed without a transition in *ample*(s) occuring first.

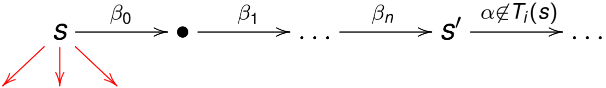
If *ample*(s) $= T_i(s)$ violates C1, then there is a path



where

- $\alpha \notin T_i(s)$ and α is dependent on $T_i(s)$,
- $\beta_0, \ldots, \beta_n$ are independent on $T_i(s)$.

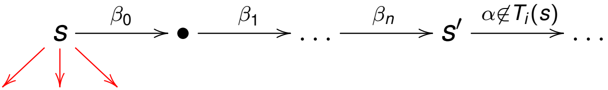
Checking C1



There are two cases.

Case A $\alpha \in T_j$ for some $i \neq j$. Then $dep(T_i(s)) \cap T_j \neq \emptyset$.

Checking C1



There are two cases.

Case A $\alpha \in T_j$ for some $i \neq j$. Then $dep(T_i(s)) \cap T_j \neq \emptyset$.

Case B $\alpha \in T_i$.

- $\beta_0, \ldots, \beta_n$ are independent on $T_i(s)$ and hence $\beta_0, \ldots, \beta_n \notin T_i$ (all transitions of P_i are considered as interdependent).
- Therefore $pc_i(s) = pc_i(s')$ and thus $\alpha \in current_i(s) \setminus T_i(s)$.
- As $\alpha \notin T_i(s)$, some transition of $\beta_0, \ldots, \beta_n$ has to be included in $pre(\alpha)$.
- Hence, $pre(current_i(s) \setminus T_i(s)) \cap T_j \neq \emptyset$ for some $j \neq i$.

Algorithm checking C1

```
function checkC1(s, P_i)
  forall P_i ≠ P_j do
    if dep(T_i(s)) ∩ T_j ≠ ∅ ∨ pre(current_i(s) ∖ T_i(s)) ∩ T_j ≠ ∅ then
      return false
  return true
end function
```

If the function returns true, then C1 holds. It may return false even if $T_i(s)$ satisfies C1.

Algorithm

```
function checkC2(X)
    forall α ∈ X do
        if visible(α) then
            return false
    return true
end function
```

```
function checkC3'(s, X)
    forall α ∈ X do
        if onStack(α(s)) then
            return false
    return true
end function
```

```
function ample(s)
  forall P_i such that T_i(s) ≠ ∅ do
    if checkC1(s, P_i) ∧ checkC2(T_i(s)) ∧ checkC3'(s, T_i(s)) then
      return T_i(s)
  return enabled(s)
end function
```

Partial order reduction

Example

Example: code

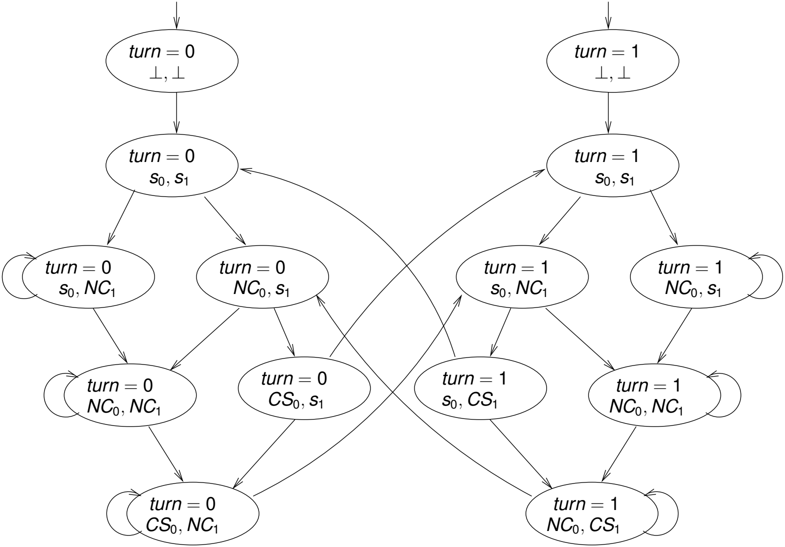
$P ::$ $m :$ *cobegin* $P_0 \| P_1$ *coend*

$P_0 ::$ $s_0 :$ *while* *true* *do*
$NC_0 :$ *wait*$(turn = 0)$;
$CS_0 :$ $turn := 1$;
endwhile;

$P_1 ::$ $s_1 :$ *while* *true* *do*
$NC_1 :$ *wait*$(turn = 1)$;
$CS_1 :$ $turn := 0$;
endwhile;

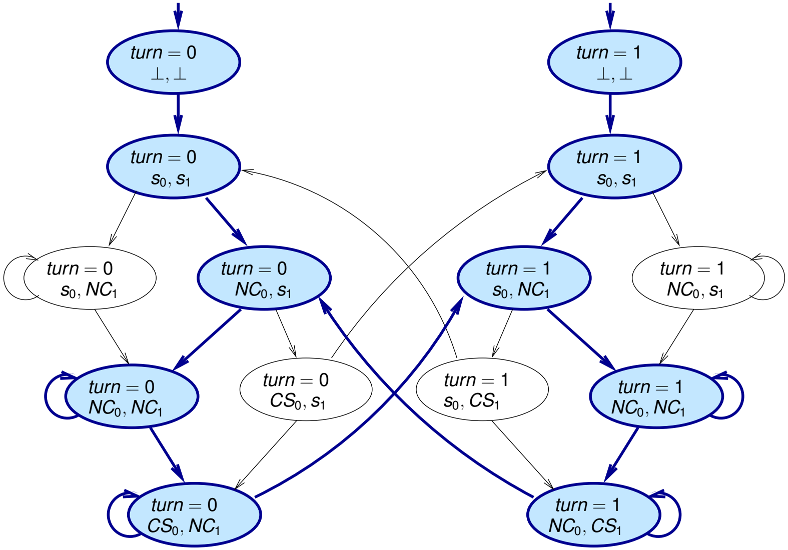
Specification formula $\varphi = \mathsf{G}\neg((pc_0 = CS_0) \wedge (pc_1 = CS_1))$

Example



turn = 0
⊥, ⊥
turn = 1
⊥, ⊥
turn = 0
s0, s1
turn = 1
s0, s1
turn = 0
s0, NC1
turn = 0
NC0, s1
turn = 1
s0, NC1
turn = 1
NC0, s1
turn = 0
NC0, NC1
turn = 0
CS0, s1
turn = 1
s0, CS1
turn = 1
NC0, NC1
turn = 0
CS0, NC1
turn = 1
NC0, CS1

Example



turn = 0
⊥, ⊥
turn = 1
⊥, ⊥
turn = 0
s_0, s_1
turn = 1
s_0, s_1
turn = 0
s_0, NC_1
turn = 0
NC_0, s_1
turn = 1
s_0, NC_1
turn = 1
NC_0, s_1
turn = 0
NC_0, NC_1
turn = 0
CS_0, s_1
turn = 1
s_0, CS_1
turn = 1
NC_0, NC_1
turn = 0
CS_0, NC_1
turn = 1
NC_0, CS_1

Coming next week

Abstraction

- How to verify large systems?
- How to find a good abstraction?
- When is an abstraction considered to be good?