CSCI 740 - Programming Language Theory

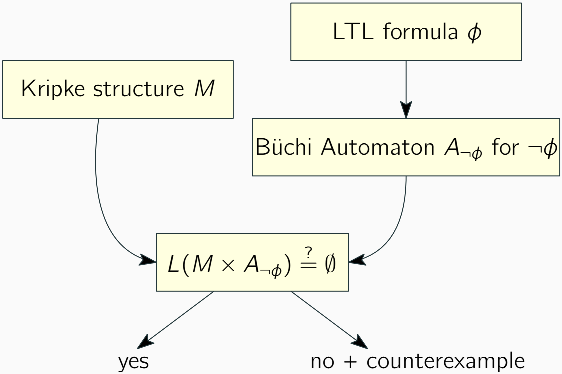
Lecture 34

Partial Order Reduction

Instructor: Hossein Hojjat

November 27, 2017

Recap


LTL formula ϕ
Kripke structure M
Büchi Automaton $A_{\neg\phi}$ for $\neg\phi$
$L(M \times A_{\neg\phi}) \stackrel{?}{=} \emptyset$
yes
no + counterexample

Complexity

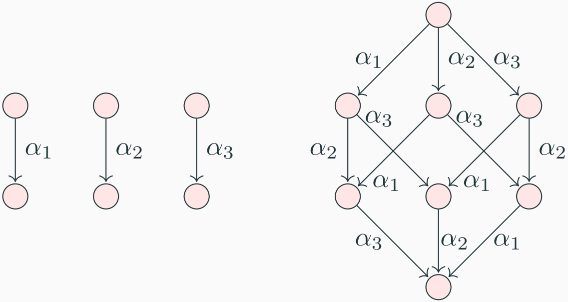
- Time and space complexity of Model checking for Kripke structure M and property ϕ

$$\mathcal{O}\Big(|M| \times 2^{|\phi|}\Big)$$

- Main disadvantage of model checking: state explosion problem
number of states rapidly exceeds computational limits for complex systems
- Approaches to reduce the size of state space:
  - ϕ: not really needed, ϕ is usually small
  - M: partial-order reduction, abstraction

Intuition

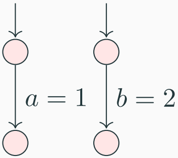
- Concurrent software is usually asynchronous:
  most of the activities by different processes are independent
- Arbitrary ordering of concurrent events: n transitions generate $n!$ orderings and 2^n states
  - Exponential "explosion" of resulting state space

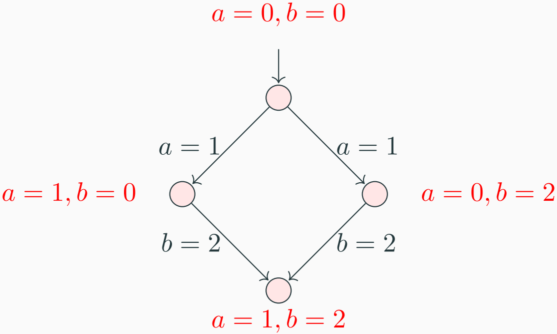


- Partial order reduction exploits the independence of concurrent events: it only explores relevant portions of the state space

Motivating Example 1

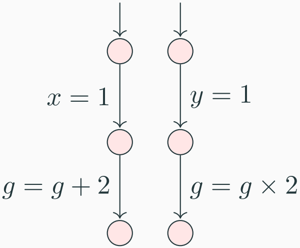
Full asynchronous interleaving of process actions is sometimes redundant





Final result is the same, no matter which path is followed

Motivating Example 2



six runs:

$x = 1; g = g + 2; y = 1; g = g \times 2$

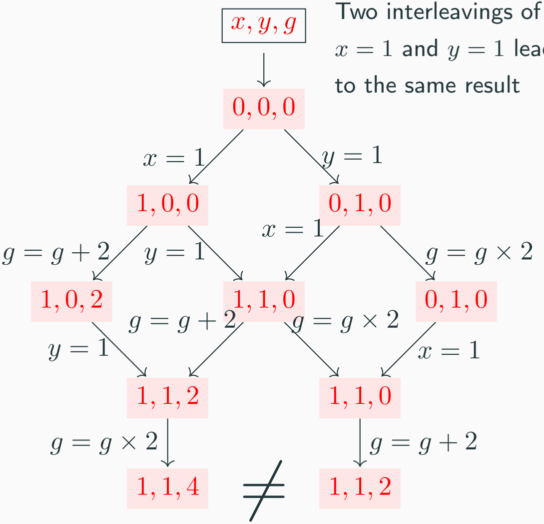
$x = 1; y = 1; g = g + 2; g = g \times 2$

$x = 1; y = 1; g = g \times 2; g = g + 2$

$y = 1; g = g \times 2; x = 1; g = g + 2$

$y = 1; x = 1; g = g \times 2; g = g + 2$

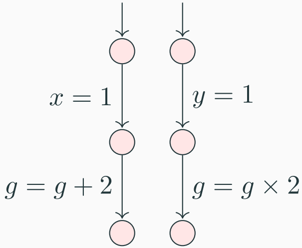
$y = 1; x = 1; g = g + 2; g = g \times 2$



Two interleavings of $x = 1$ and $y = 1$ lead to the same result

Two possible interleavings of $g = g + 2$ and $g = g \times 2$ both lead to different values of g

Dependencies



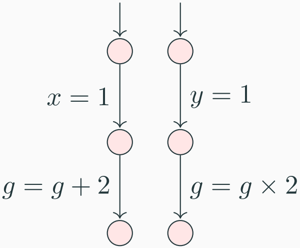
- Assume x and y are local variables, g is a global variable

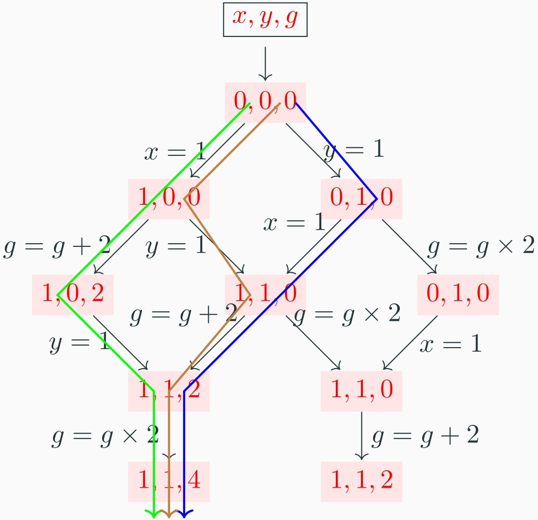
Dependent

- $g = g \times 2$ and $g = g + 2$ because they share same data
- $x = 1$ and $g = g + 2$ because they are both part of first process
- $y = 1$ and $g = g \times 2$ because they are both part of second process

Independent

- $x = 1$ and $y = 1$
- $x = 1$ and $g = g \times 2$
- $y = 1$ and $g = g + 2$



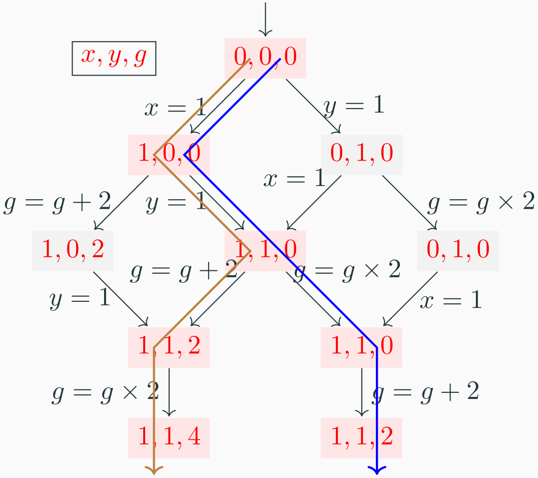


The three runs differ only in the relative order of execution independent operations

Partial order Reduction Idea

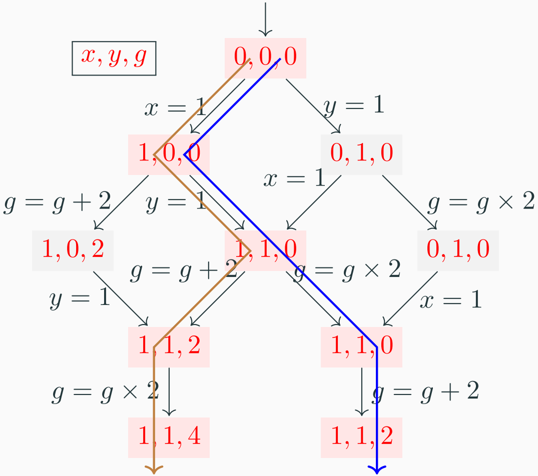
- Partition execution runs into equivalent classes
- Group runs for which the order of independent actions does not matter
  - Hence **partial order** reduction
- Check only one run (i.e. the representative) in each equivalent class
- Model checking using partial order reduction is also/better called "model checking using representatives"

Necessary Runs



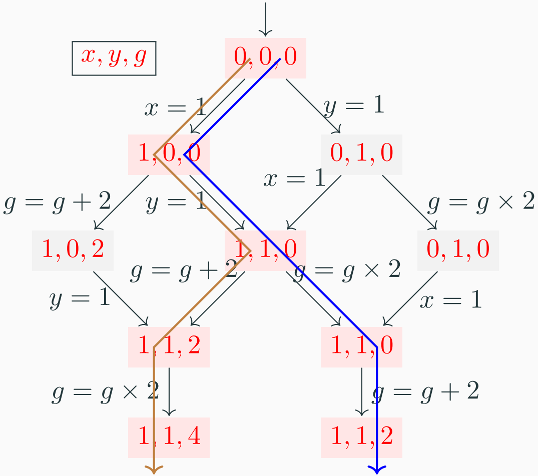
- After eliminating all independences, two runs are left
- Note that three states (of the full state space) are not visited.

Necessary Runs



- LTL properties $G(g = 0 \vee g > x)$, $F(g \geq 2)$, $(g = 0)\ U\ (x = 1)$
- All these properties hold in the full graph and the reduced graph
- (i.e. considering only two necessary runs)

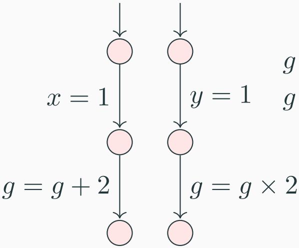
Necessary Runs



- What about $G(x \geq y)$?
- This property holds in the reduced graph, but not in the full graph

Visibility

- Visibility of the variables x and y in $G(x \geq y)$ introduces dependencies that were not assumed to exist
- Dependencies do not only arise from data but also from properties to be checked
- **Solution.** Remove $x = 1$ and $y = 1$ from the independences



	$x = 1$	$y = 1$	$g = g + 2$	$g = g \times 2$
$x = 1$	–	D(prop)	D(control)	indep
$y = 1$	D(prop)	–	indep	D(control)
$g = g + 2$	D(control)	indep	–	D(data)
$g = g \times 2$	indep	D(control)	D(data)	–

Questions

- Given a set of processes how can we automatically identify classes of equivalent runs?
- How to avoid full construction upfront, but deciding on-the-fly which states and transitions are necessary

Implementing Partial Order Reduction

- At each state s, some set of actions is enabled: enabled(s)
- Of this set, a subset are such that any interleaving of them yields the same end state and they do not "influence" other actions: ample(s)
- Pick one order for elements of ample(s) and execute all those actions first in that order
- How to compute ample(s)?

Computing ample(s)

Important characteristics of elements a, b of ample(s): must be independent & invisible

- Action a should not disable b, and vice-versa
- The effect of ample(s) actions should not affect the values of any "relevant" atomic propositions in the LTL property

Conservative heuristics to compute ample(s)

- If the same variable appears in two actions, they are dependent
- If two actions appear in the same process/module, they are dependent
- If an action shares a variable with a relevant atomic proposition, then it is visible