Lecture 8

Reduction Methods

Optimizing the execution of verification

Bottleneck in Model checking is Memory Consumption

Used memory is product of

|property automaton| * |state vector| * #states

- |property| is usually small (BA can be optimized by a few states
- |state vector| can be reduced by various compression schemes
- #states given, can be reduced by
  - Partial order reduction
  - Atomic sequences
  - Abstraction

Partial order reduction: Basic ideas

Interleaving semantics represent a single concurrent execution by many interleaving sequences.

The interleavings corresponding to the same concurrent execution are related and are not all necessary for the verification of most properties.

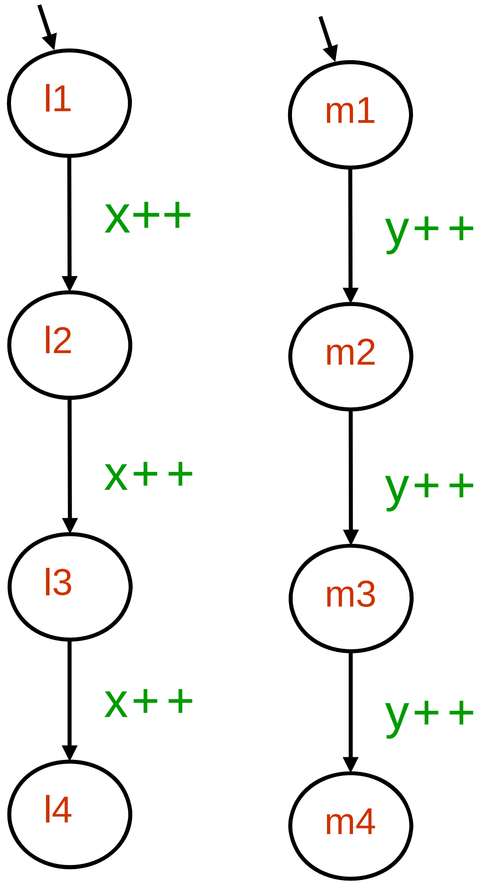
It should thus be possible to verify properties using only some representative interleavings of each concurrent execution.

What interleavings are required could depend on the property to be checked.

Simple Example:

Variables: x, y: integer
Initially x = y = 0

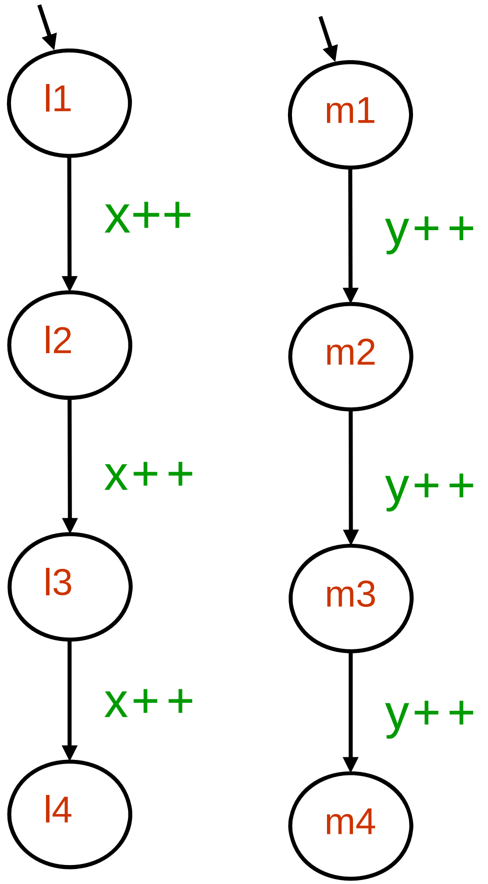
is y == 3 reachable?

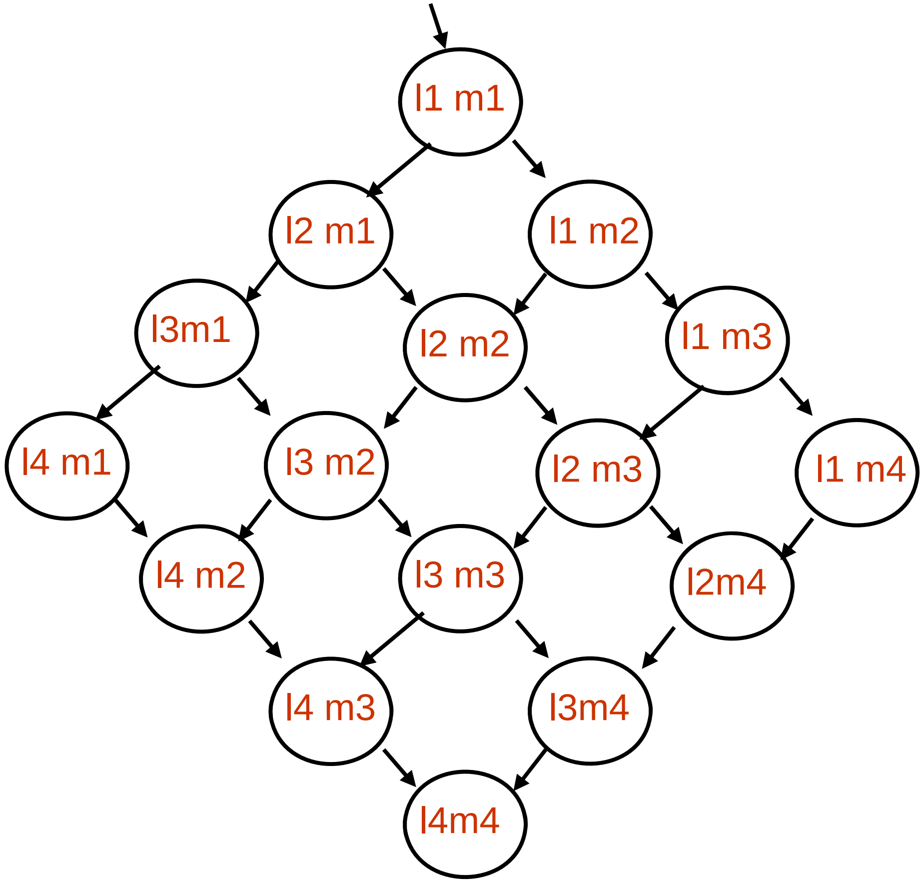


Simple Example:

Variables: x, y: integer
Initially x = y = 0

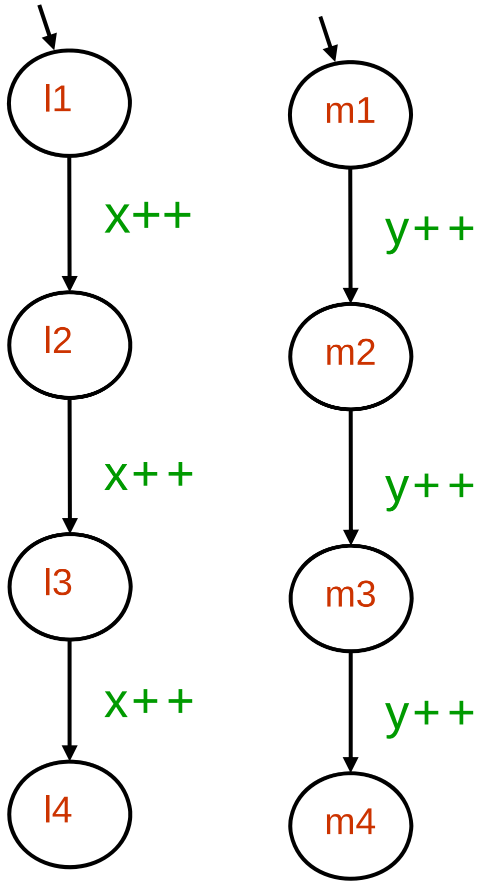
is y == 3 reachable?



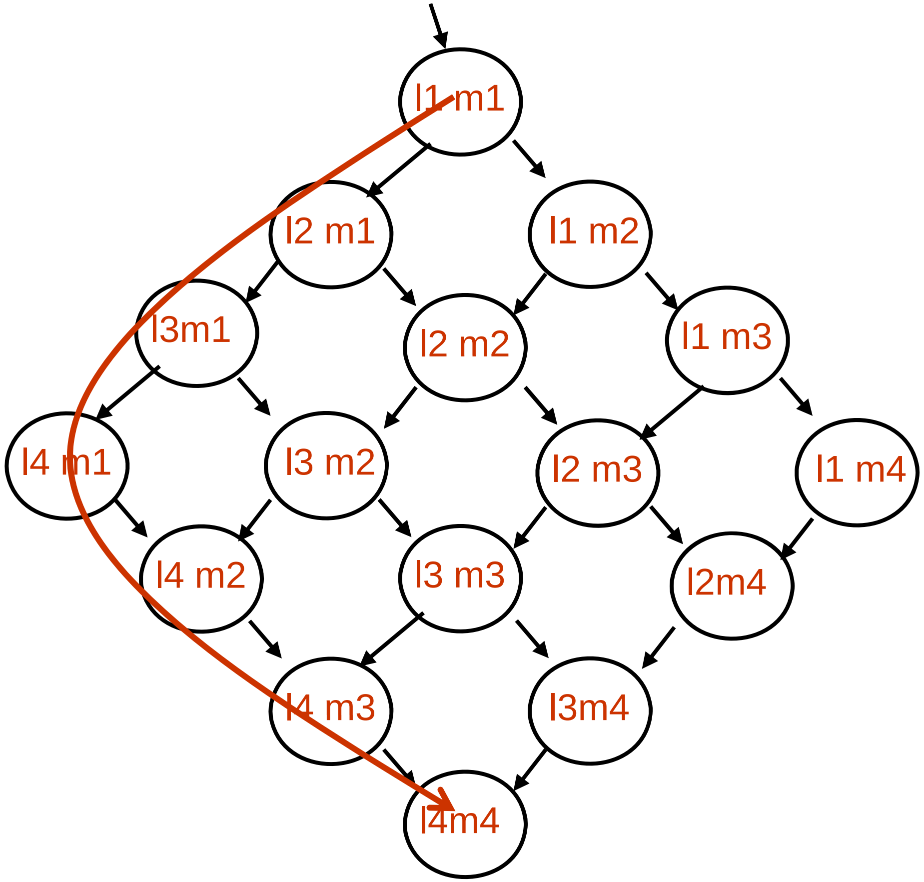


Enough to explore some interleavings:

Variables: x, y: integer
Initially x = y = 0



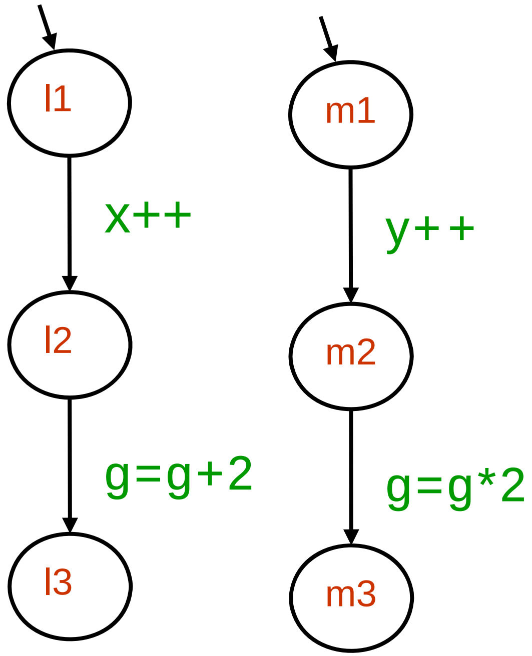
is y == 3 reachable?



Another Example

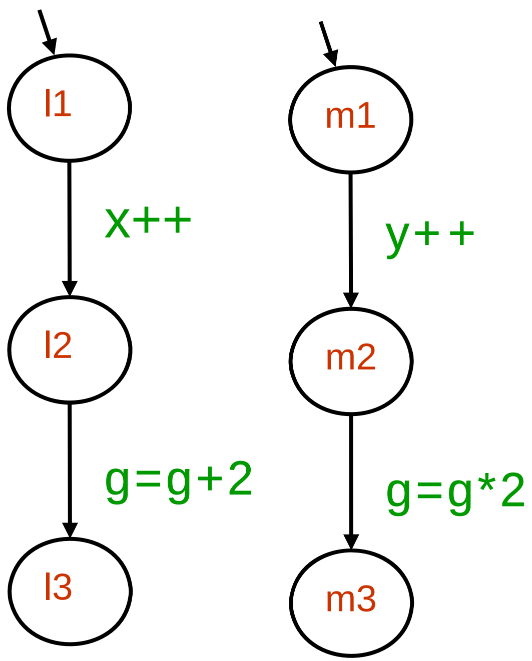
Variables: x, y, g: integer
Initially x = y = 0

is g == 4 reachable?

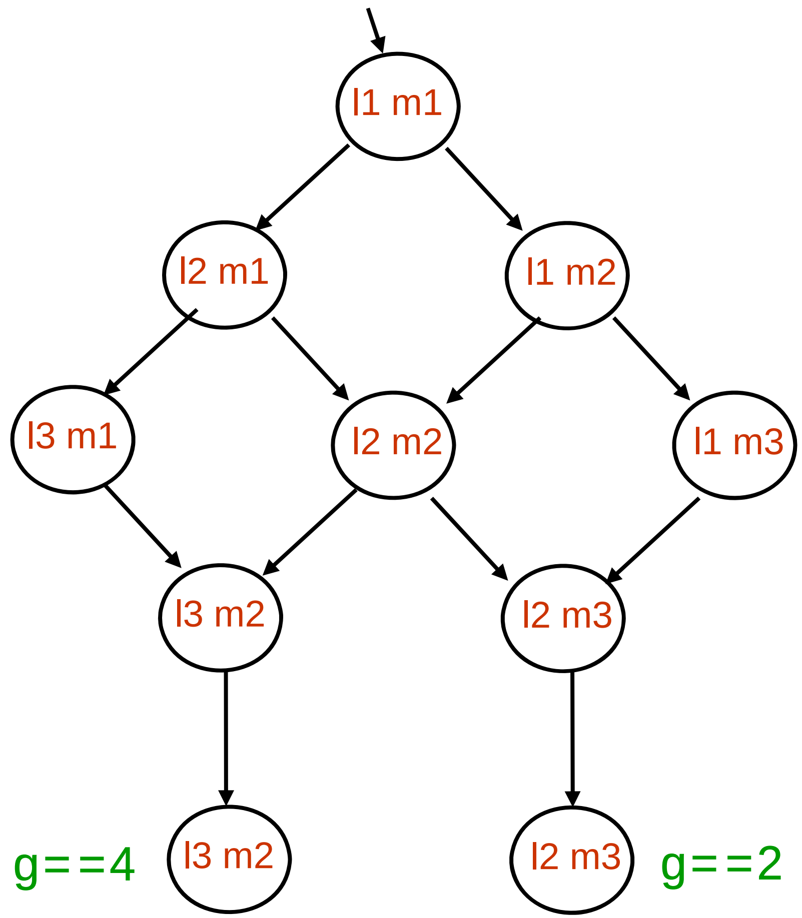


Another Example

Variables: x, y, g: integer
Initially x = y = 0

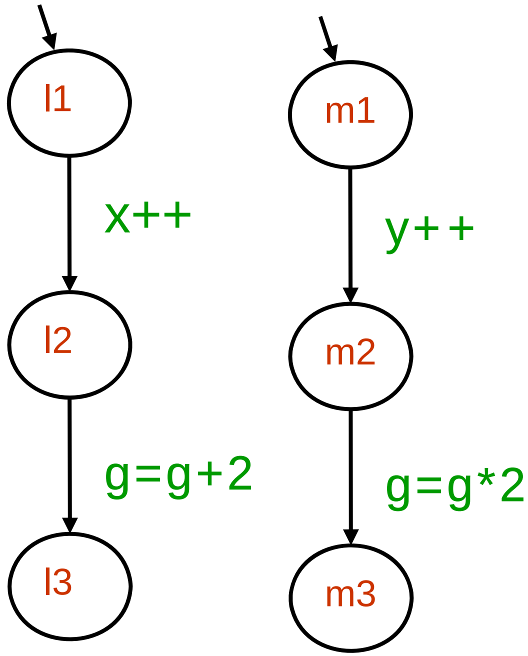


is g == 4 reachable?

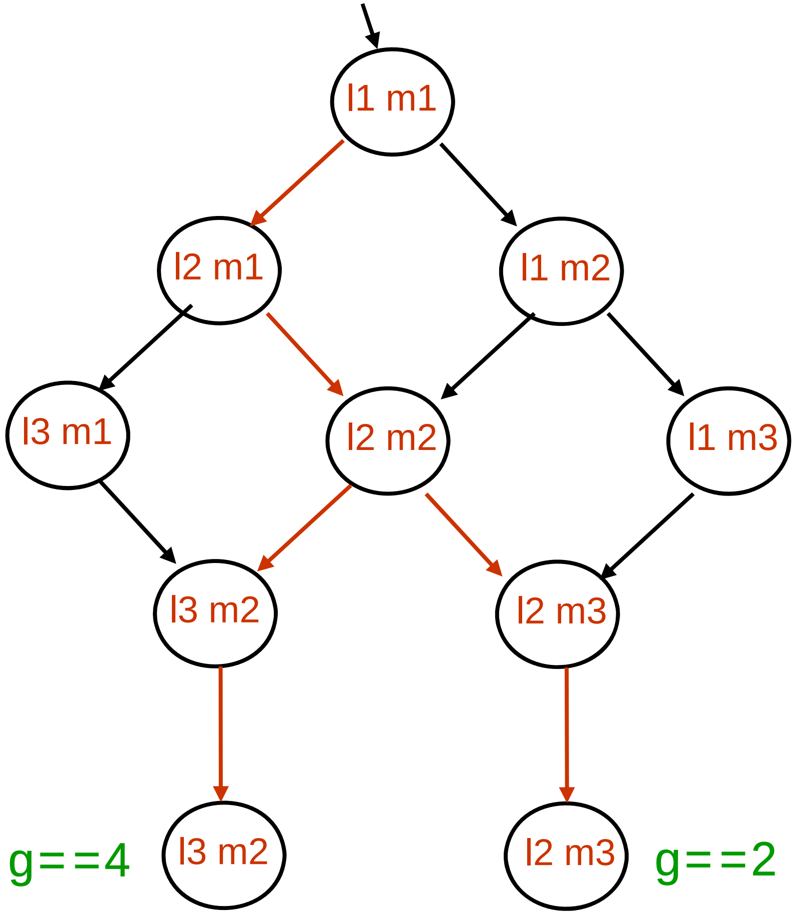


Reducing search

Variables: x, y, g: integer
Initially $x = y = 0$



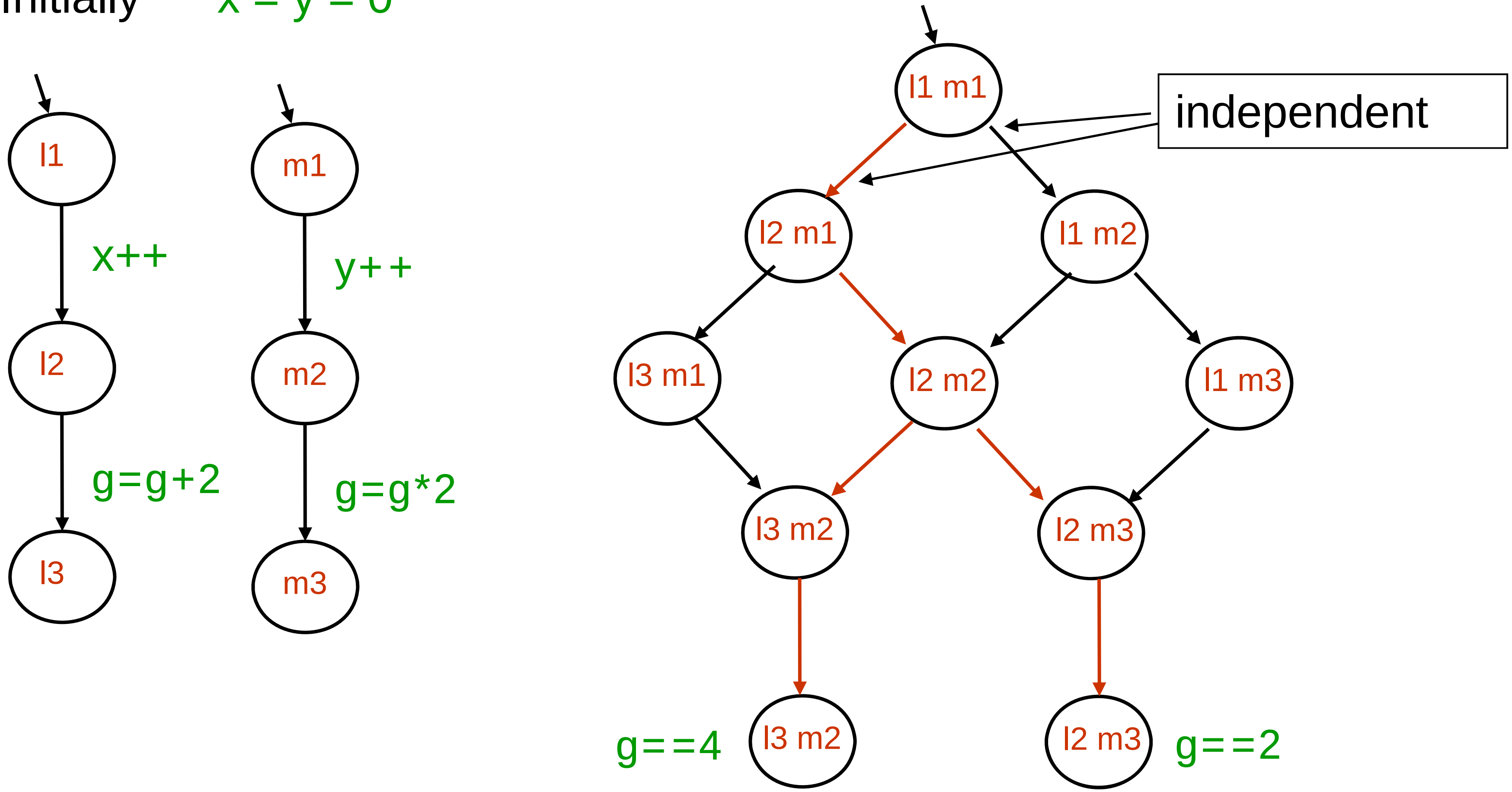
is $g == 4$ reachable?



Reducing search

Variables: x, y, g: integer
Initially x = y = 0

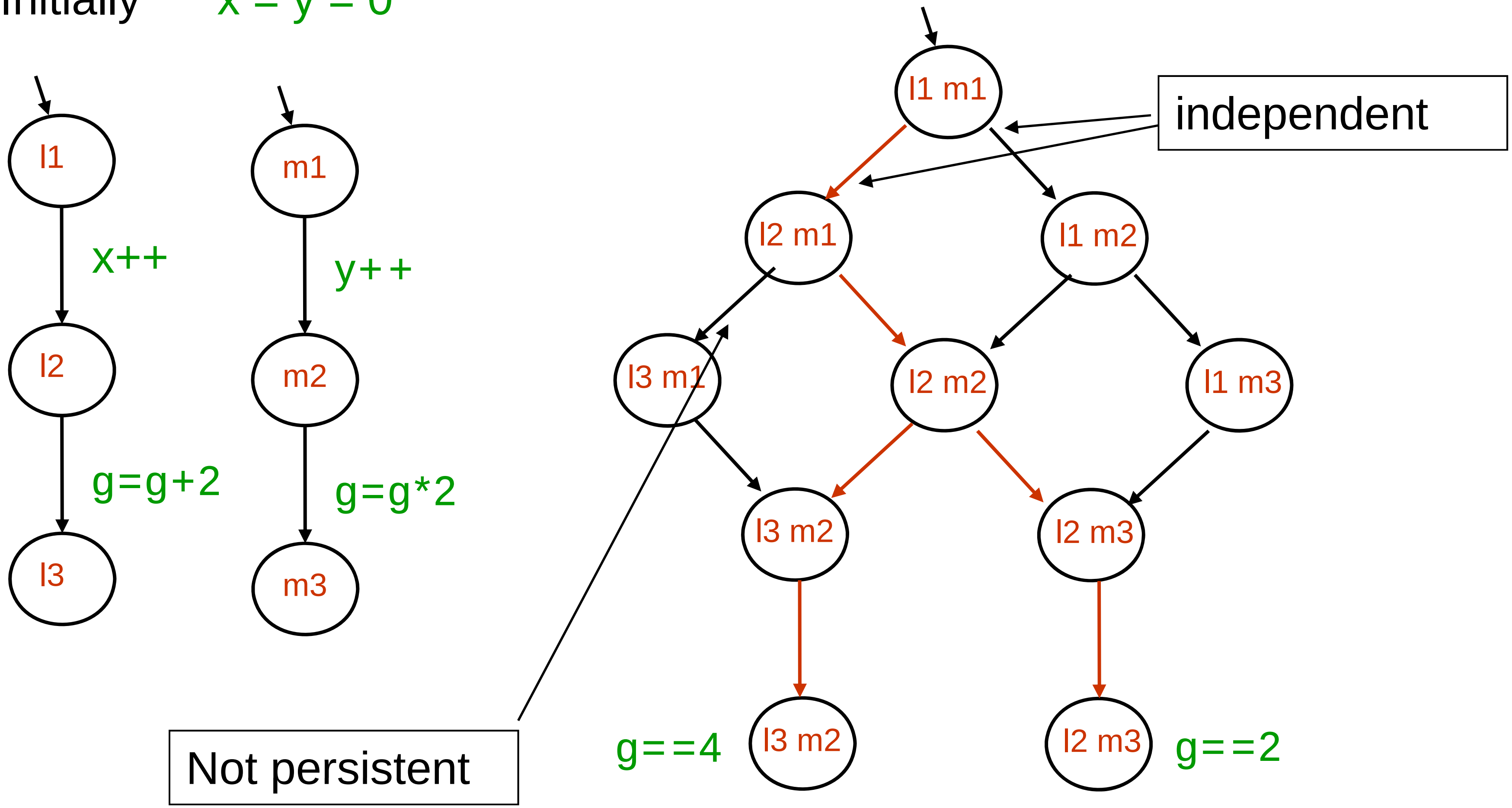
is g == 4 reachable?



Reducing search

Variables: x, y, g: integer
Initially x = y = 0

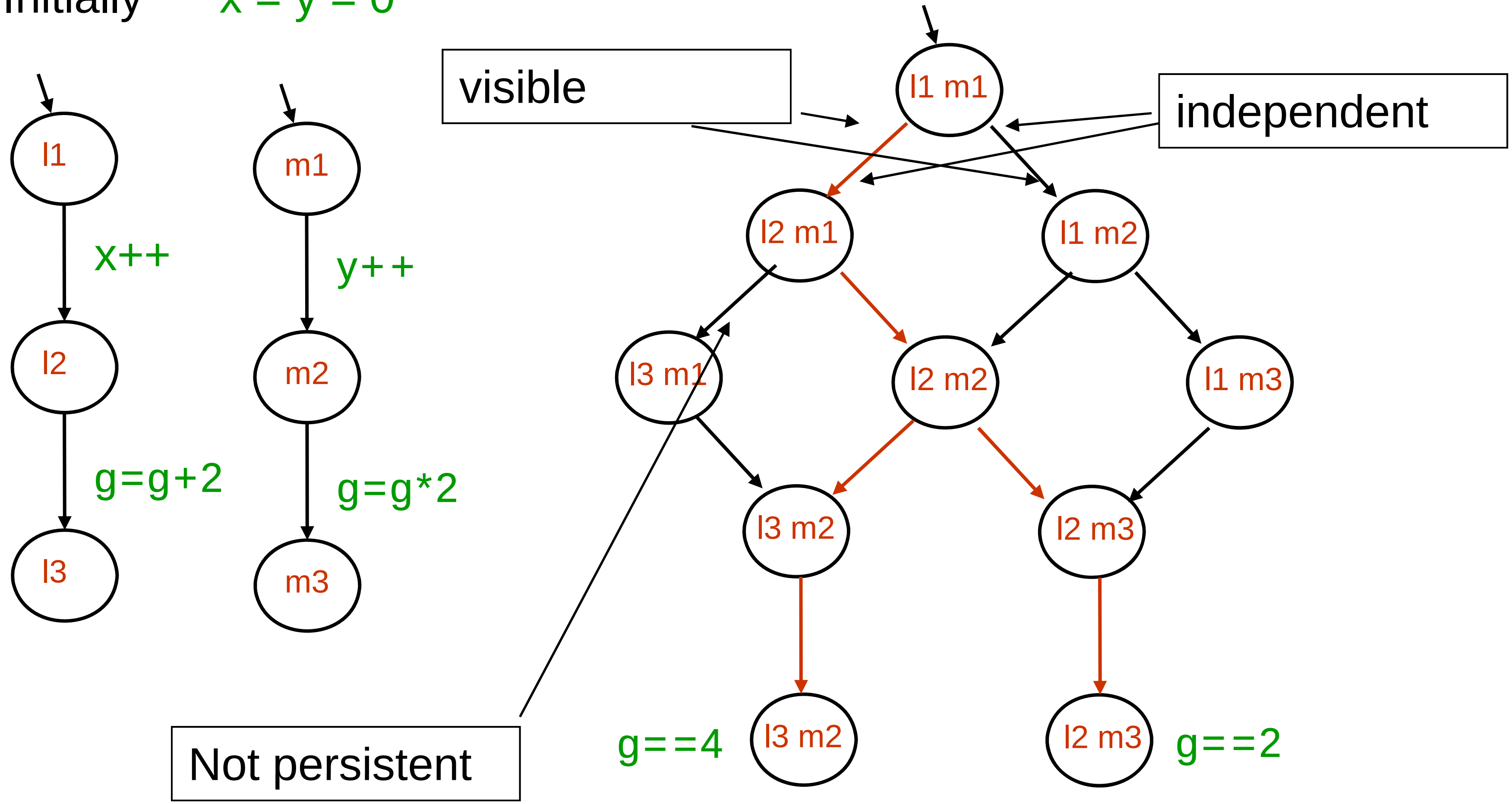
is g == 4 reachable?



Reducing search

Variables: x, y, g: integer
Initially x = y = 0

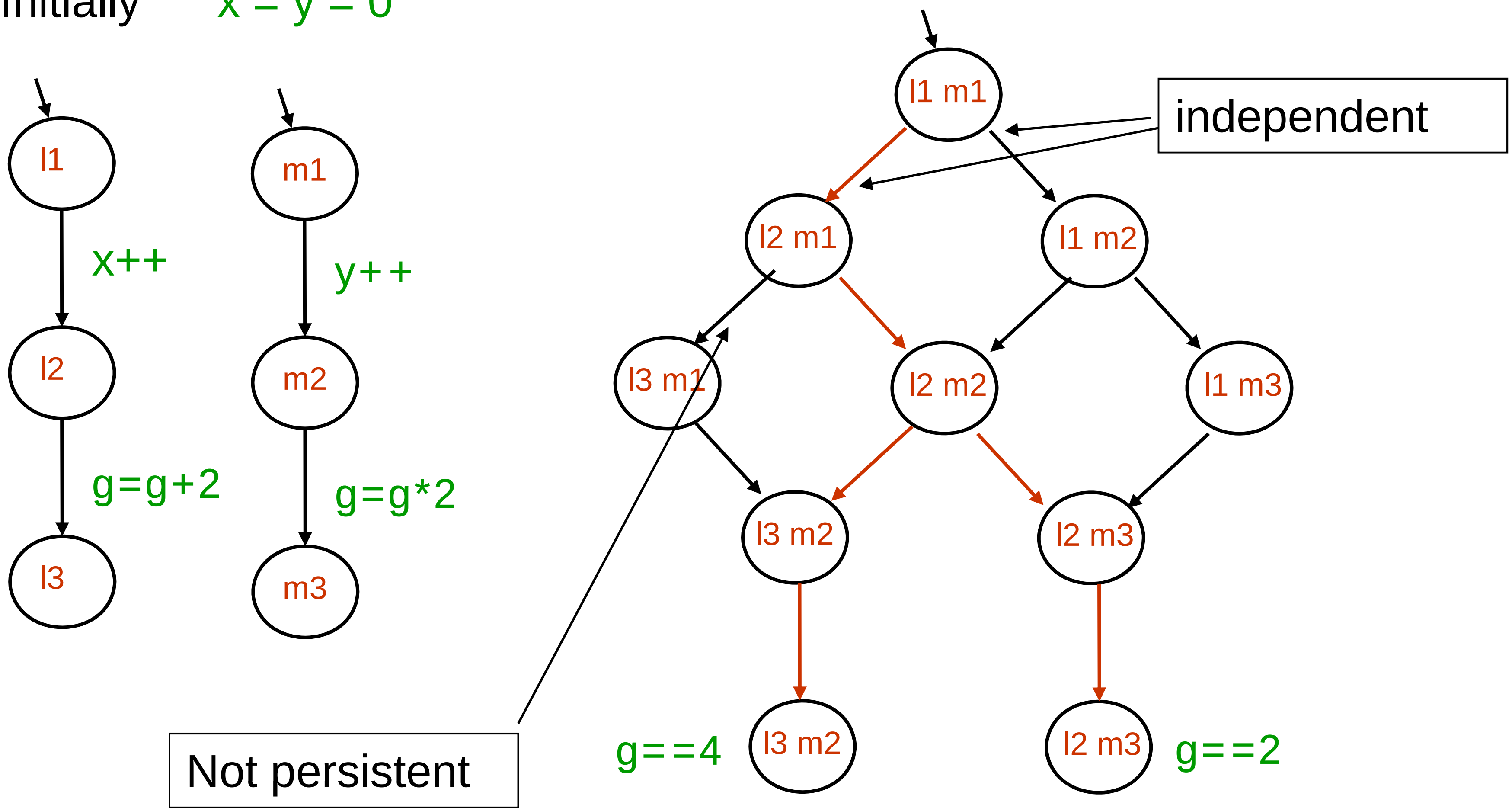
is y>x reachable?



Reducing search

Variables: x, y, g: integer
Initially x = y = 0

is y>x reachable?



Commutativity of statements (actions)

- Statement s commutes left with statement t if

$$t;s \subseteq s;t$$

  - s does not disable t
  - t does not enable s
  - order of execution irrelevant
- Statement s commutes right with statement t if

$$s;t \subseteq t;s$$

- s and t independent = commute left and right
- A statement is visible if it may change the value in some state formula that we are checking

Commutativity (examples)

- `x = 8`   `y = 7`
- `x == 8`   `y == 7`
- `x == 8`   `y = x`
- `chan?5`   `chan!5`
- `chan?5`   `chan!3`

Idea of Partial order reduction

In state space exploration, whenever s commutes left with t then we can ``defer'' the exploration of t, since it works just as fine to explore s

note that by deferring t , we implicitly defer any sequence of actions that starts with the action t

s1 ,... , sk is a persistent set if:

- when performing any sequence t1 t2 … tn of other actions,
- then any s1 ,... , sk is independent of any t1, … , tn

Idea of Partial order reduction

In State-space exploration, it is enough that

1. In each state, explore a persistent set of actions,
2. If the explored set contains a visible action, then explore all enabled actions.
3. Along any explored cycle, each action must be either disabled once or explored once.
   - can be achieved by requiring that if any explored action leads to an already visited state, then explore all enabled actions.

Finding persistent sets: one strategy

One strategy: find a set of processes whose “potentially enabled” statements are independent of any other process, i.e., “local”

A ``local'' statement is either

- accessing only local variables
- A receive from a queue (or test for emptiness), from which no other process receives
- A send to a queue (or test for being full), to which no other process sends.
- Last two operations must not be disabled within a selection.

Implementation in SPIN

Precompute when P can execute only local actions.

In each global state, if a ``locally executing'' P is found,
such that all statements lead to unvisited states,
then it is enough to explore Ps transitions.

Promela Extension:

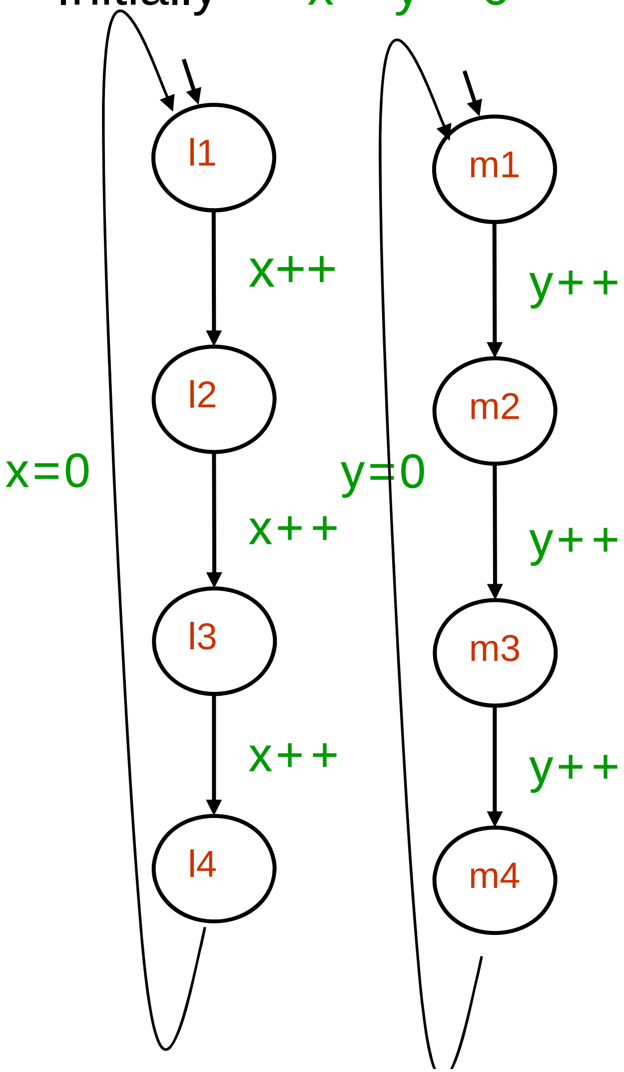
Hard to statically determine exclusive access to
channels

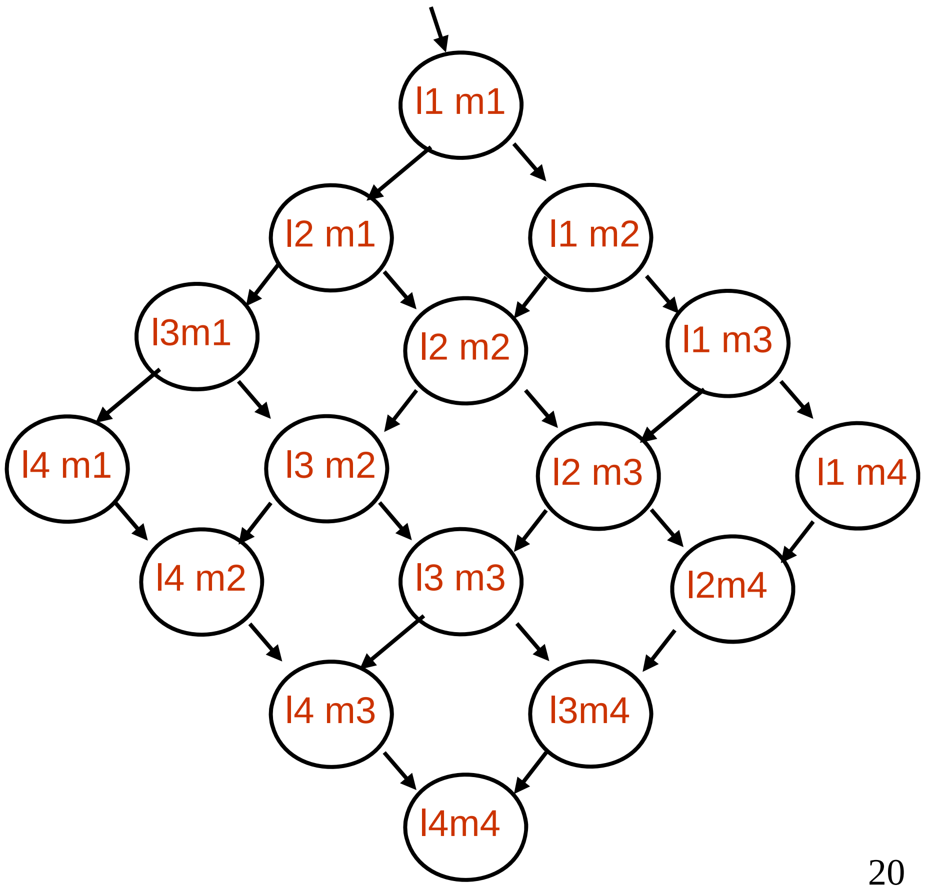
annotate channels by **`xr`** or **`xs`**

How clever is PO reduction really?

Variables: x, y: integer
Initially x = y = 0

is y == 3 reachable?





Limits of PO Reduction

From Table 3 in [Holzmann -97], one can see that

- Partial Order Reduction buys some reduction
- But that manual coarsening buys much more.

Coarsening: Making Steps atomic

When is this correct?

A sequence of steps

t1; … ; tk ; s ; t1' ; … ; tm'

can be made atomic if

- Each action t1 … tk commutes right with any other action that may potentially be executed interleaved with t1; … ; tk
- Each action t1' … tm' commutes left with any other action that may potentially be executed interleaved with t1' ; … ; tm'
- t1' ; … ; tm' is guaranteed to terminate (e.g., if it contains a loop)