Verification

Lecture 27

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

Abstraction

$$
\begin{array}{l}
\textbf{local}\quad y_1, y_2 : \textbf{integer} \\
\qquad\qquad\qquad \textbf{where } y_1 = y_2 = 0 \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
\ell_0: & \textbf{noncritical} \\
\ell_1: & y_1 := y_2 + 1 \\
\ell_2: & \textbf{await } (y_2 = 0 \vee y_1 \leq y_2) \\
\ell_3: & \textbf{critical} \\
\ell_4: & y_1 := 0
\end{array}\right]
\end{array}\right] \\
\| \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
m_0: & \textbf{noncritical} \\
m_1: & y_2 := y_1 + 1 \\
m_2: & \textbf{await } (y_1 = 0 \vee y_2 < y_1) \\
m_3: & \textbf{critical} \\
m_4: & y_2 := 0
\end{array}\right]
\end{array}\right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{local}\quad b_1, b_2, b_3 : \textbf{boolean} \\
\qquad\qquad\qquad \textbf{where } b_1, b_2, b_3 \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
\ell_0: & \textbf{noncritical} \\
\ell_1: & (b_1, b_3) := (\mathit{false}, \mathit{false}) \\
\ell_2: & \textbf{await } (b_2 \vee b_3) \\
\ell_3: & \textbf{critical} \\
\ell_4: & (b_1, b_3) := (\mathit{true}, \mathit{true})
\end{array}\right]
\end{array}\right] \\
\| \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
m_0: & \textbf{noncritical} \\
m_1: & (b_2, b_3) := (\mathit{false}, \mathit{true}) \\
m_2: & \textbf{await } (b_1 \vee \neg b_3) \\
m_3: & \textbf{critical} \\
m_4: & (b_2, b_3) := (\mathit{true}, b_1)
\end{array}\right]
\end{array}\right]
\end{array}
$$

REVIEW: Simulation order

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$, i=1, 2, be two transition systems over AP.

A simulation for (TS_1, TS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that:

1. $\forall q_1 \in I_1 \; \exists q_2 \in I_2. \; (q_1, q_2) \in \mathcal{R}$
2. for all $(q_1, q_2) \in \mathcal{R}$ it holds:

   2.1 $L_1(q_1) = L_2(q_2)$

   2.2 if $q_1' \in Post(q_1)$
   then there exists $q_2' \in Post(q_2)$ with $(q_1', q_2') \in \mathcal{R}$

REVIEW: Simulation order and ∀CTL*

Let TS be a finite transition system (without terminal states) and q, q' states in TS.

The following statements are equivalent:

(1) $q \preceq_{TS} q'$

(2) for all $\forall\mathrm{CTL}^*$-formulas Φ: $q' \models \Phi$ implies $q \models \Phi$

(3) for all $\forall\mathrm{CTL}$-formulas Φ: $q' \models \Phi$ implies $q \models \Phi$

Proof Rules as Abstractions

For assertions q, φ

$$
\begin{array}{ll}
\text{I1.} & P \Vdash \varphi \rightarrow q \\
\text{I2.} & P \Vdash \Theta \rightarrow \varphi \\
\text{I3.} & P \Vdash \{\varphi\}\ \mathcal{T}\ \{\varphi\} \\
\hline
 & P \vDash \Box q
\end{array}
\qquad \text{INV}
$$

- $AP = \{q\}$
- TA: $S = I = \{s_q\}$; $s_q \rightarrow s_q$
- Simulation: $R=((t, s_q) \mid t \models \varphi\}$

Predicate Abstraction

- Abstraction is determined by a set of predicates,

  $P=\{\phi_1, \phi_2, \dots \phi_N\}$

- Abstract state space: subsets of P

- Abstraction function $f(q) = \{\phi_i \mid q \models \phi_i\}$

Example

local y_1, y_2 : **integer**
where $y_1 = y_2 = 0$

loop forever do

ℓ_0: **noncritical**

ℓ_1: $y_1 := y_2 + 1$

ℓ_2: **await** $(y_2 = 0 \vee y_1 \leq y_2)$

ℓ_3: **critical**

ℓ_4: $y_1 := 0$

||

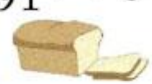
loop forever do

m_0: **noncritical**

m_1: $y_2 := y_1 + 1$

m_2: **await** $(y_1 = 0 \vee y_2 < y_1)$

m_3: **critical**



m_4: $y_2 := 0$

Predicates:
guards of transitions

P = $\{b_1, b_2, b_3\}$ +
control predicates

with

b_1: y1 = 0
b_2: y2 = 0
b_3: y1 $\leq$ y2

Example

$$
\begin{array}{l}
\textbf{local}\ \ y_1, y_2 : \textbf{integer} \\
\qquad\qquad\qquad \textbf{where}\ y_1 = y_2 = 0 \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
\ell_0: & \textbf{noncritical} \\
\ell_1: & y_1 := y_2 + 1 \\
\ell_2: & \textbf{await}\ (y_2 = 0 \lor y_1 \leq y_2) \\
\color{red}{\ell_3:} & \color{red}{\textbf{critical}} \\
\ell_4: & y_1 := 0
\end{array}\right]
\end{array}\right] \\
\| \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
m_0: & \textbf{noncritical} \\
m_1: & y_2 := y_1 + 1 \\
m_2: & \textbf{await}\ (y_1 = 0 \lor y_2 < y_1) \\
\color{red}{m_3:} & \color{red}{\textbf{critical}} \\
m_4: & y_2 := 0
\end{array}\right]
\end{array}\right]
\end{array}
$$

$$
\begin{array}{l}
\textbf{local}\ \ b_1, b_2, b_3 : \textbf{boolean} \\
\qquad\qquad\qquad \textbf{where}\ b_1, b_2, b_3 \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
\ell_0: & \textbf{noncritical} \\
\ell_1: & (b_1, b_3) := (\mathit{false}, \mathit{false}) \\
\ell_2: & \textbf{await}\ (b_2 \lor b_3) \\
\color{red}{\ell_3:} & \color{red}{\textbf{critical}} \\
\ell_4: & (b_1, b_3) := (\mathit{true}, \mathit{true})
\end{array}\right]
\end{array}\right] \\
\| \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
m_0: & \textbf{noncritical} \\
m_1: & (b_2, b_3) := (\mathit{false}, \mathit{true}) \\
m_2: & \textbf{await}\ (b_1 \lor \neg b_3) \\
\color{red}{m_3:} & \color{red}{\textbf{critical}} \\
m_4: & (b_2, b_3) := (\mathit{true}, b_1)
\end{array}\right]
\end{array}\right]
\end{array}
$$

Example

This abstraction allows us to prove

- mutual exclusion
- bounded overtaking

using a model checker, since it is a finite-state program.

$$
\begin{array}{l}
\textbf{local}\quad b_1, b_2, b_3 : \textbf{boolean} \\
\qquad\qquad\qquad\qquad \textbf{where } b_1, b_2, b_3 \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
\ell_0: & \textbf{noncritical} \\
\ell_1: & (b_1, b_3) := (\mathit{false}, \mathit{false}) \\
\ell_2: & \textbf{await } (b_2 \vee b_3) \\
\ell_3: & \textbf{critical} \\
\ell_4: & (b_1, b_3) := (\mathit{true}, \mathit{true})
\end{array}\right]
\end{array}\right] \\
\| \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
m_0: & \textbf{noncritical} \\
m_1: & (b_2, b_3) := (\mathit{false}, \mathit{true}) \\
m_2: & \textbf{await } (b_1 \vee \neg b_3) \\
m_3: & \textbf{critical} \\
m_4: & (b_2, b_3) := (\mathit{true}, b_1)
\end{array}\right]
\end{array}\right]
\end{array}
$$

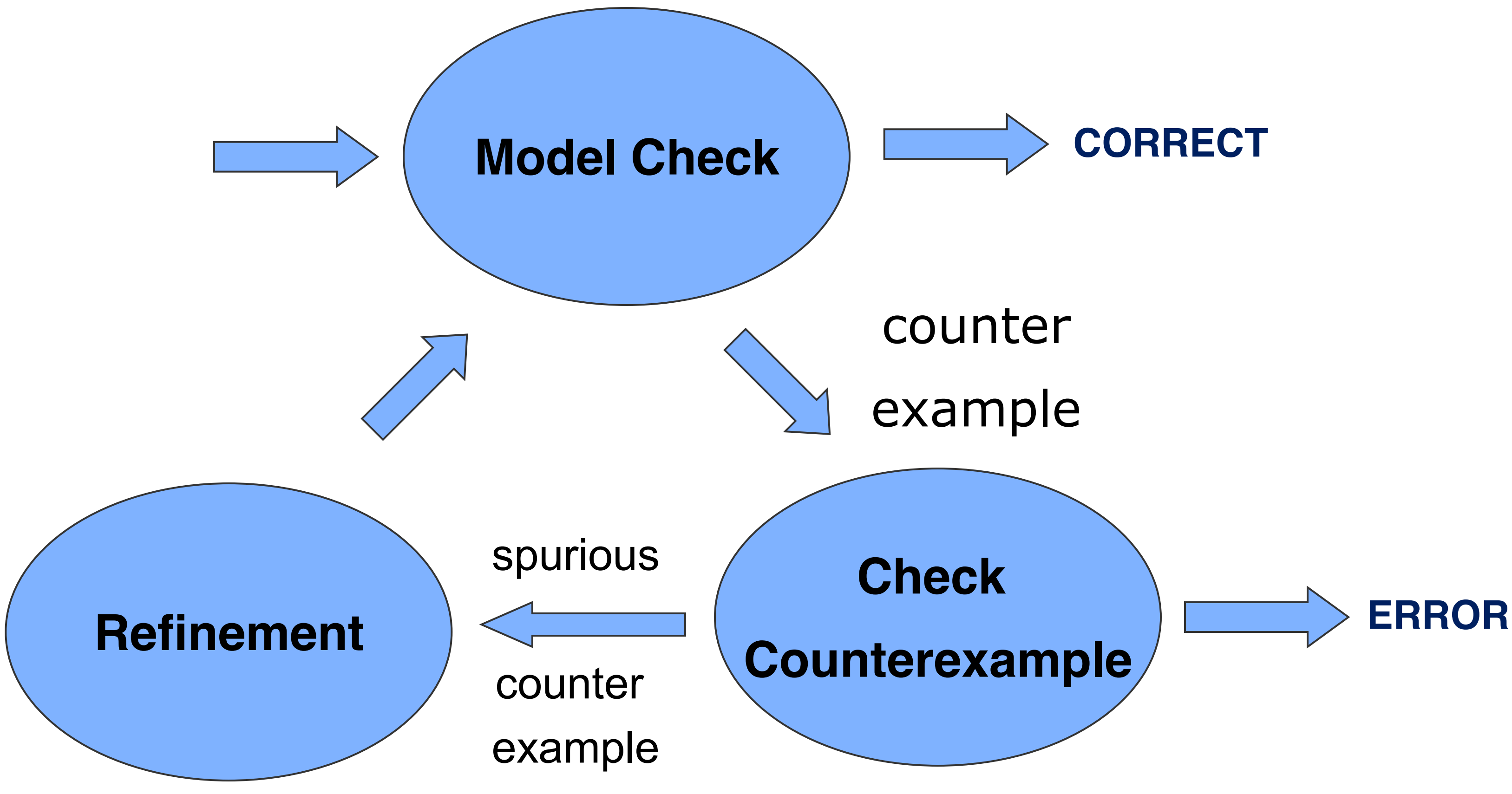
How To Determine the Basis?

A good starting set:

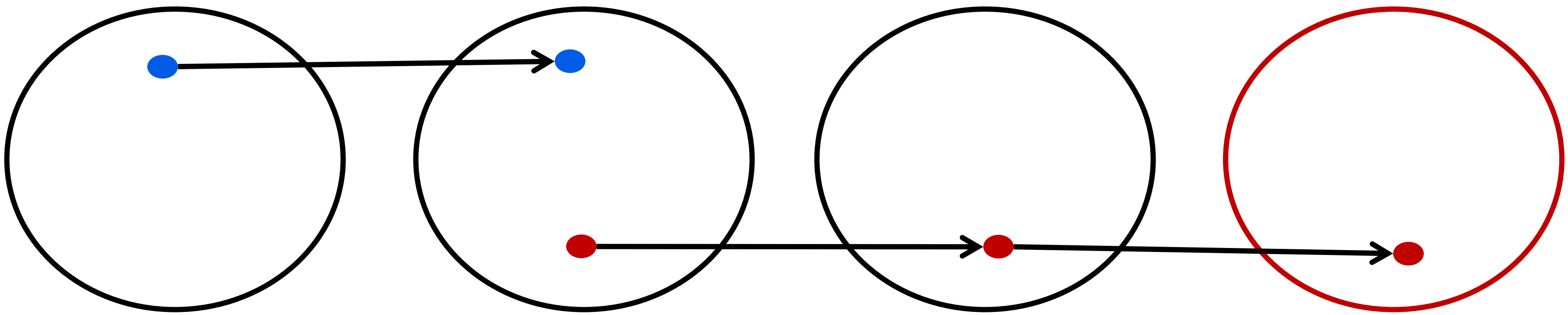
- The atomic assertions appearing in the guards of the transitions ($\rightarrow$ enabling conditions can be represented exactly, and thus fairness carries over)
- The atomic assertions appearing in the property to be proven ($\rightarrow$ the property abstraction is exact)

Analysis of counterexamples may lead to refinement of the abstraction by adding more assertions to the basis.

Counter Example Guided Abstraction Refinement (CEGAR)



Spurious counter examples



Checking abstract error paths

Let E be an assertion indicating an error state.

An abstract counter example $x_0\ x_1\ \ldots\ x_k$ is **concretizable** if there exists a sequence of concrete states $s_0\ s_1\ \ldots\ s_k$ such that

1. For each $0 \leq i \leq k$, $f(s_i)=x_k$.
2. $s_0 \models \Theta$ and $s_k \models E$
3. For each $0 \leq i < k$, $(s_i, s_{i+1}) \models \rho$

Checking abstract error paths

1. For each $0 \leq i \leq k$, $f(s_i) = x_k$.
2. $s_0 \models \Theta$ and $s_k \models E$
3. For each $0 \leq i < k$, $(s_i, s_{i+1}) \models \rho$

represented as a formula:

$$\Theta(V^0) \wedge \bigwedge_{i=0..k} \bigwedge_{\phi \in x_i} \phi(V^i) \wedge \bigwedge_{i=0..k-1} \rho(V^i, V^{i+1}) \wedge E(V^k)$$

Craig Interpolation

For a given pair of formulas $\varphi(X)$ and $\psi(Y)$
such that $\varphi \wedge \psi$ is unsatisfiable,

a **Craig interpolant** $\Delta(X \cap Y)$ is a formula
over the common variables
such that

φ implies Δ and
$\Delta \wedge \psi$ is unsatisfiable.

Craig interpolants can be automatically generated for many first-order theories.

Path cutting

Split formula

$$\Theta(V^0) \wedge \bigwedge_{i=0..k} \bigwedge_{\phi \in x_i} \phi\,(V^i) \quad \wedge \bigwedge_{i=0..k-1} \rho(V^i,V^{i+1}) \quad \wedge\ E(V^k)$$

into two parts:

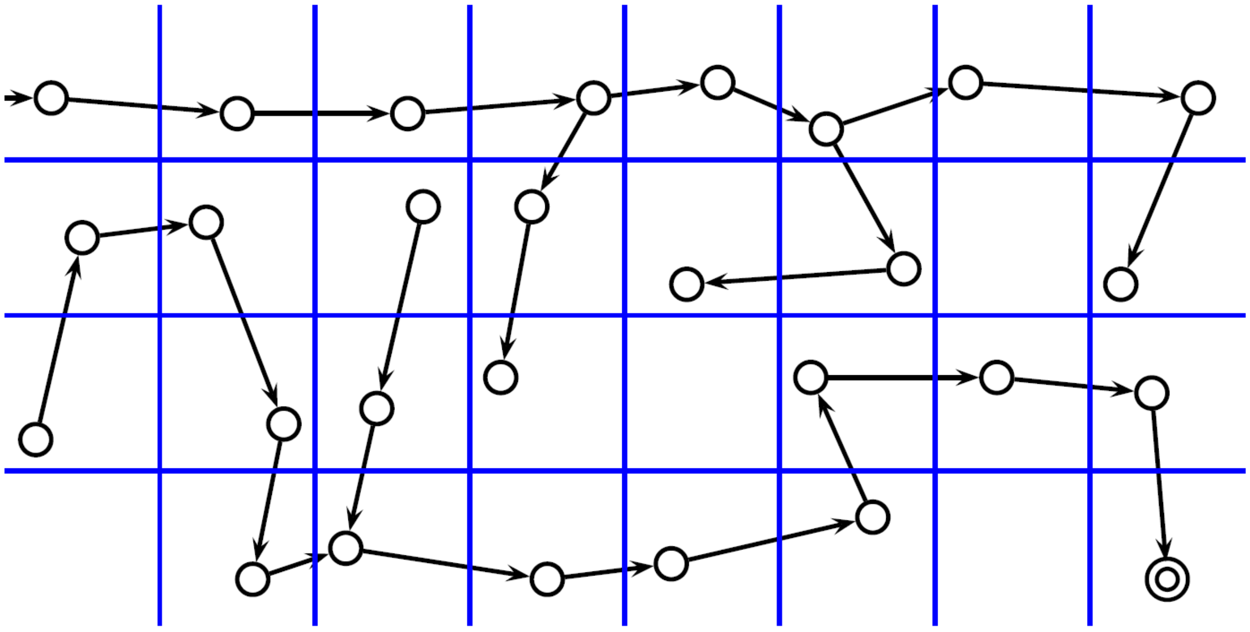
$$\phi_1 = \Theta(V^0) \wedge \bigwedge_{i=0..j-1} \bigwedge_{\phi \in x_i} \phi\,(V^i) \quad \wedge \bigwedge_{i=0..j-2} \rho(V^i,V^{i+1})$$

$$\phi_2 = \bigwedge_{i=j..k} \bigwedge_{\phi \in x_i} \phi\,(V^i) \quad \wedge \bigwedge_{i=j-1..k-1} \rho(V^i,V^{i+1}) \quad \wedge\ E(V^k)$$

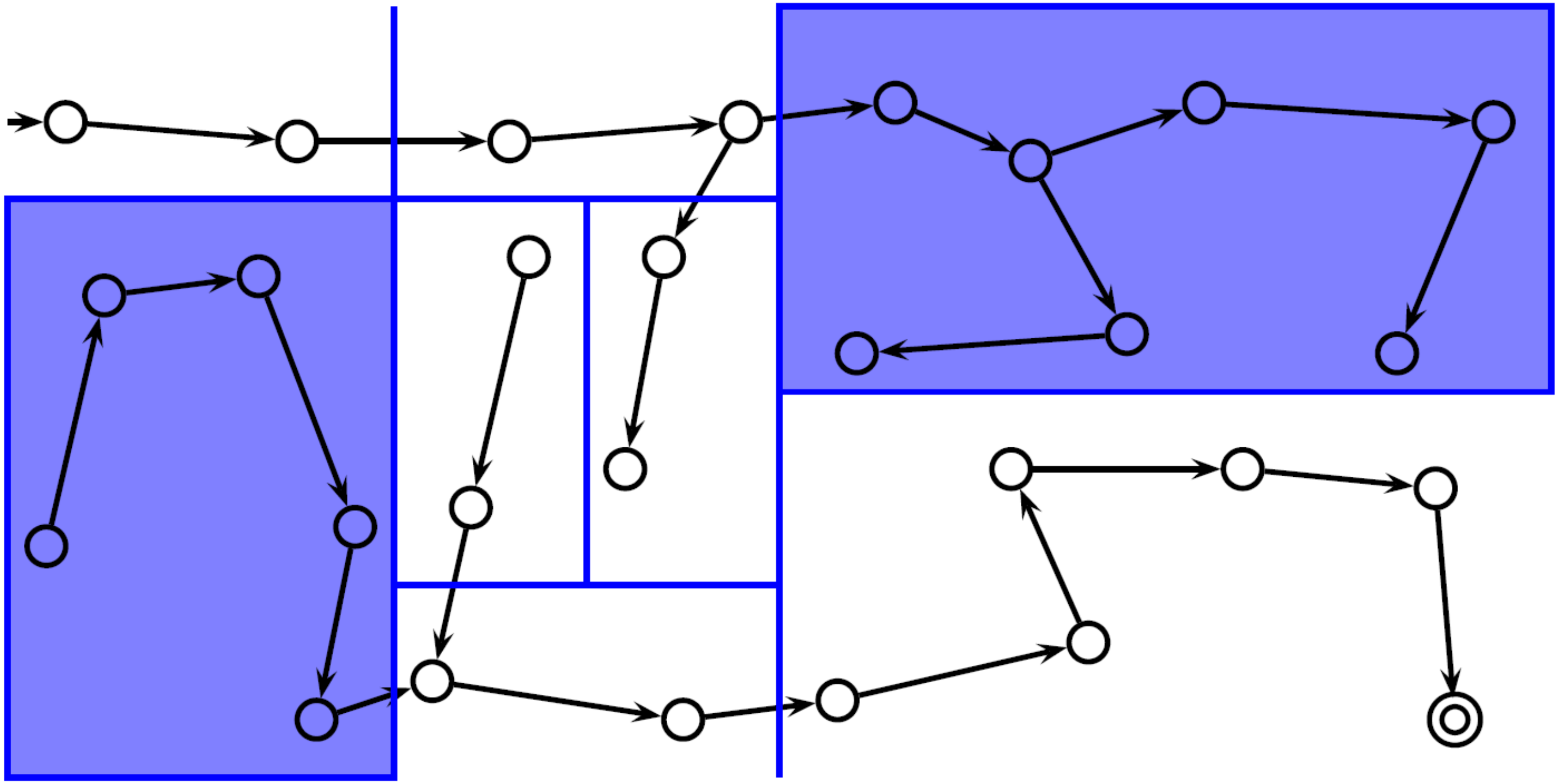
Use interpolant of ϕ_1 and ϕ_2 as new predicate.

Problem: abstract state space explosion

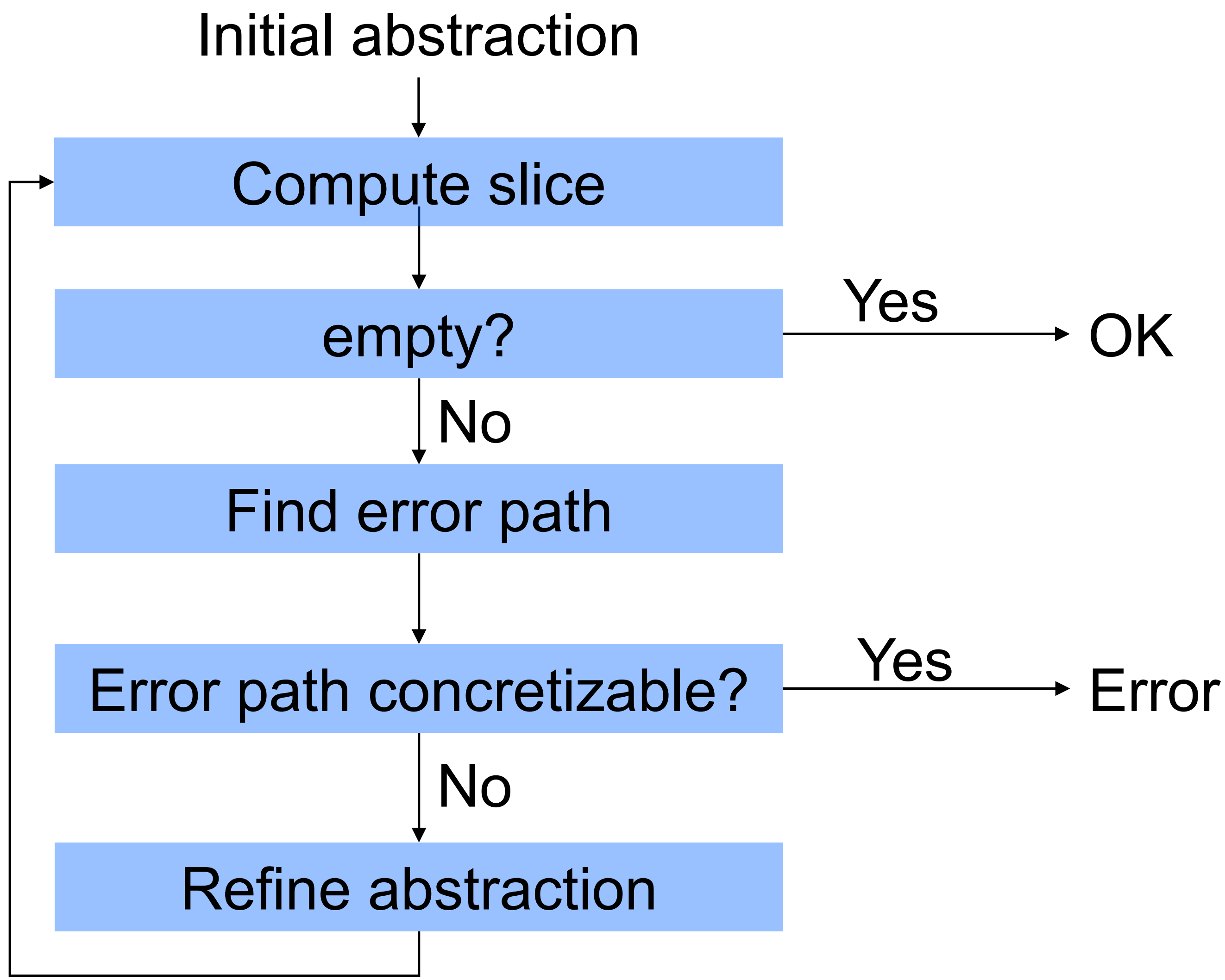
- Abstract state space grows exponentially with number of predicates



Slicing Abstractions

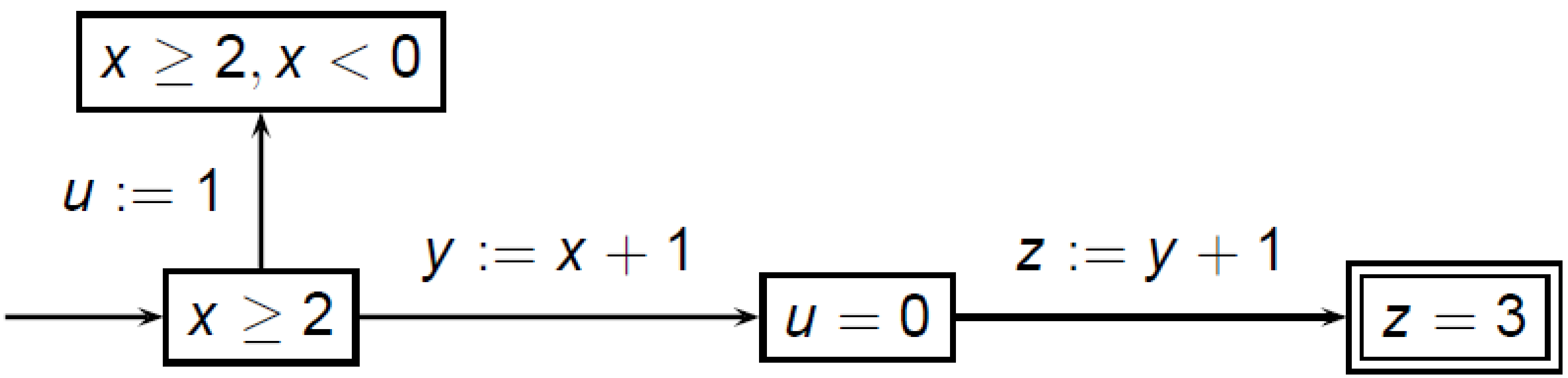


Slicing Abstractions (SLAB)



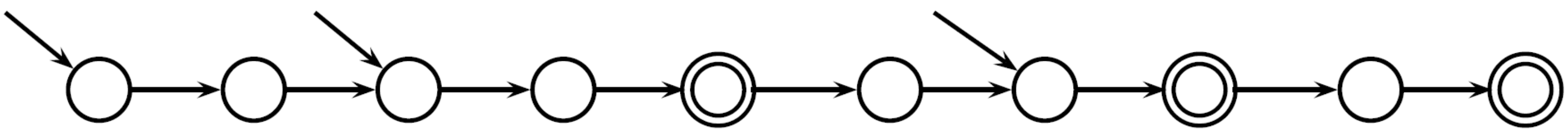
SLAB abstractions

- Finite graphs
- Nodes labeled with sets of literals
- Edges labeled with sets of transitions
- Initial node, error node

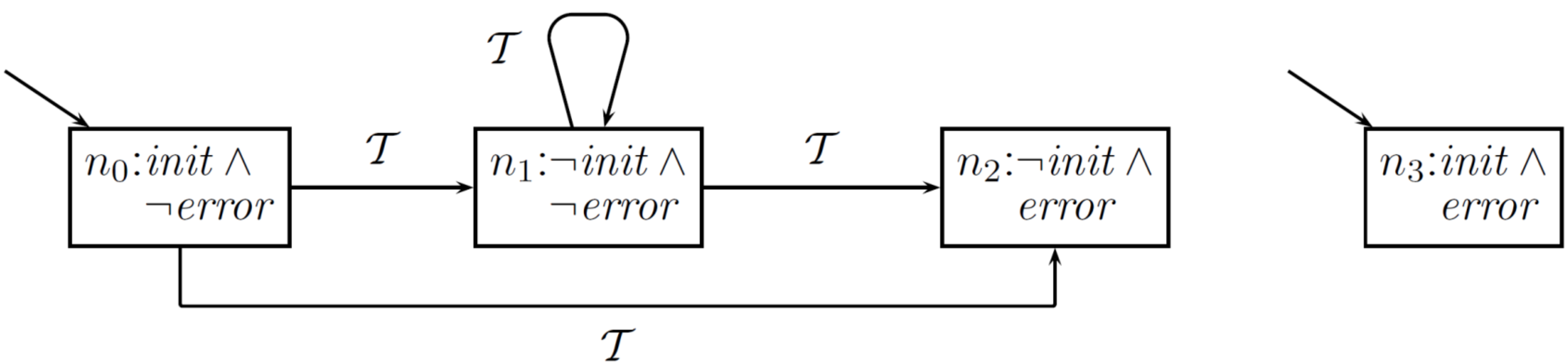


Initial abstraction

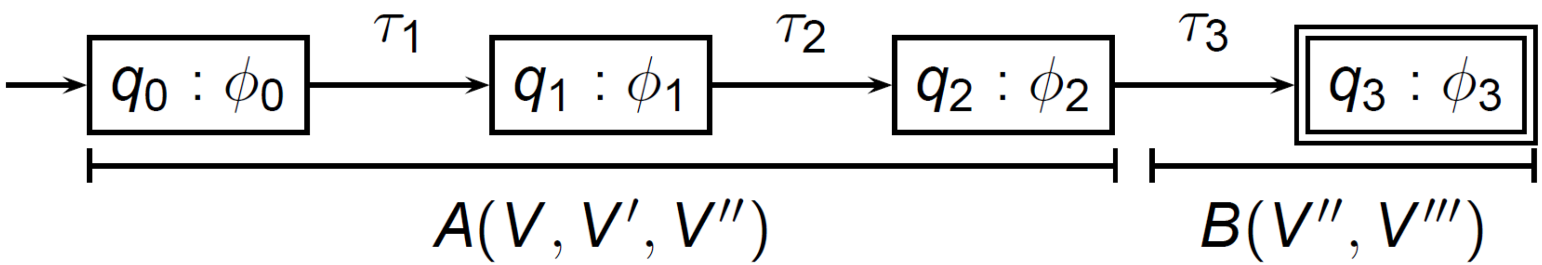
- need only *irreducible* error paths



- Initial abstraction:

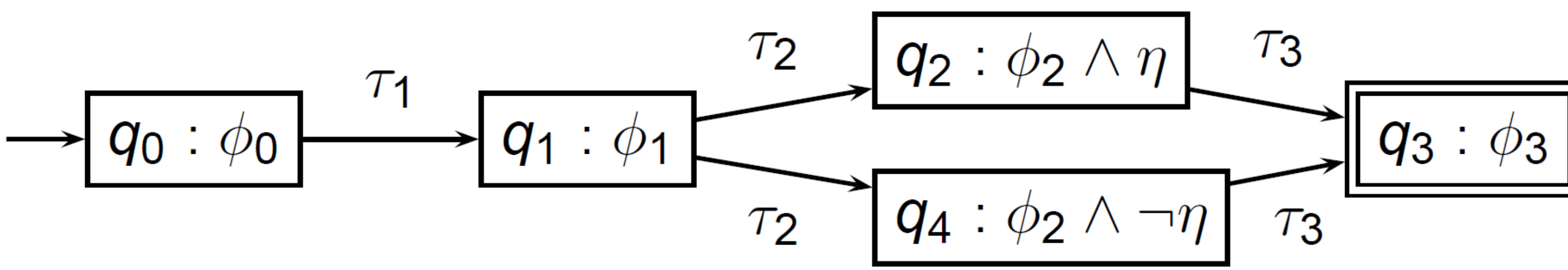


Local refinement by node splitting



- $A \wedge B$ unsat, but A, B sat $\rightsquigarrow$ Craig interpolant η:
  - $A \models \eta$, $B \models \neg\eta$
  - $Var(\eta) \subseteq Var(A) \cap Var(B)$, i.e. values at q_2

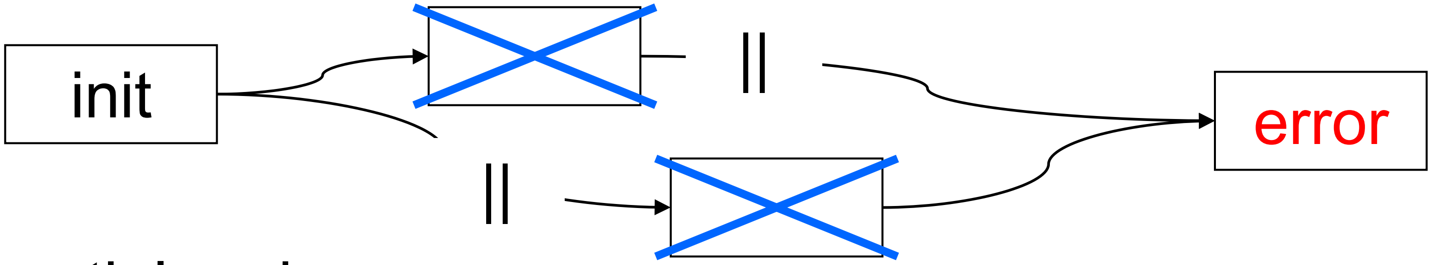
$\rightsquigarrow$ split q_2 with $\eta, \neg\eta$:



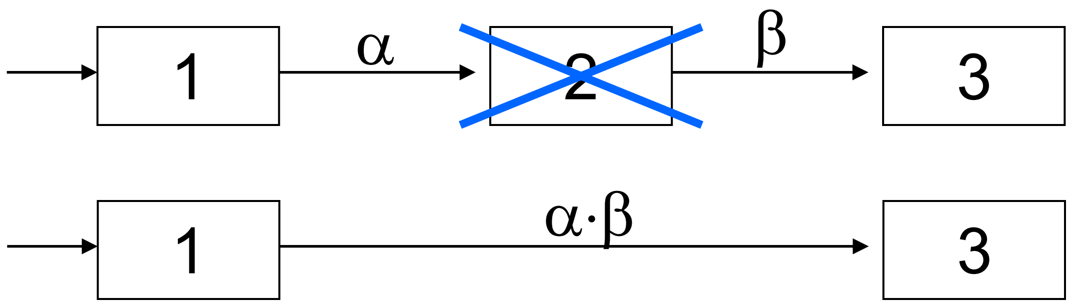
Slicing: Eliminating Nodes

- Inconsistent nodes

- Unreachable nodes

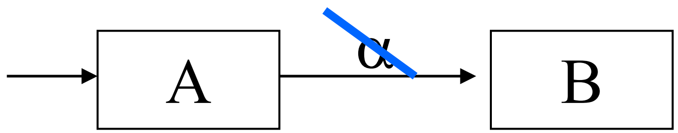


- Sequential nodes



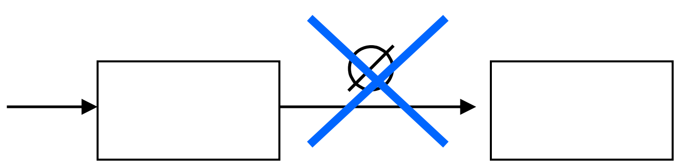
Slicing: Eliminating transitions

- Inconsistent transitions

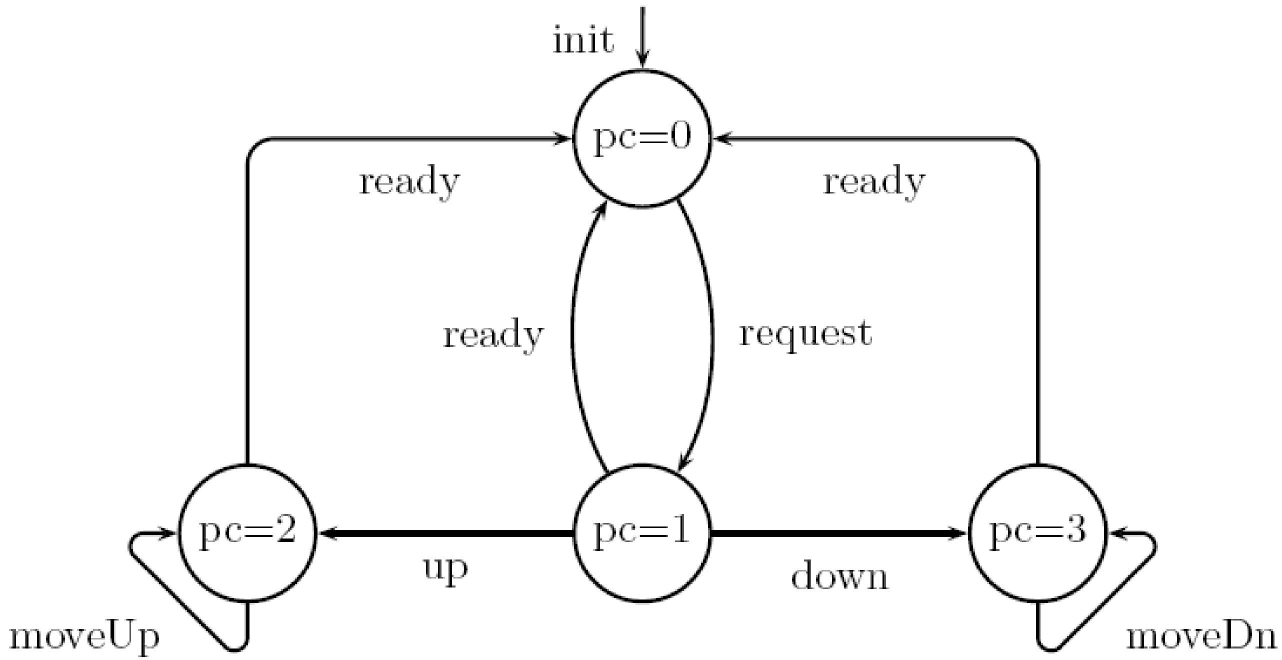


$A(V) \wedge \alpha(V,V') \wedge B(V')$ <u>unsatisfiable</u>

- Empty Edges

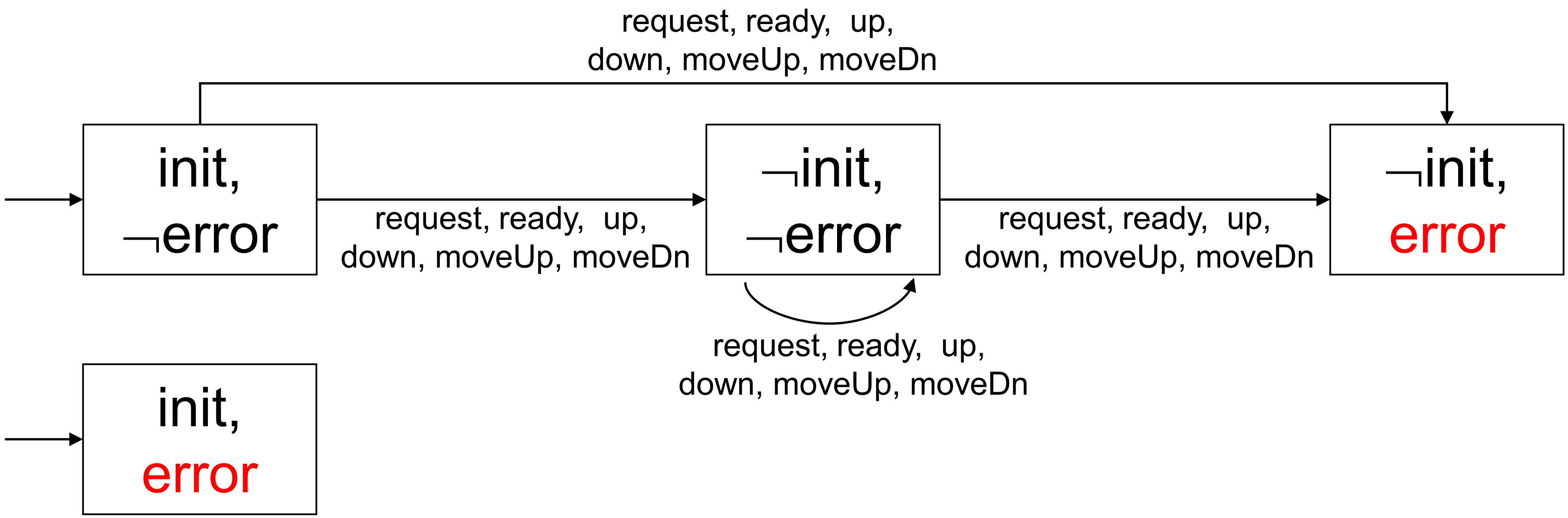


Example

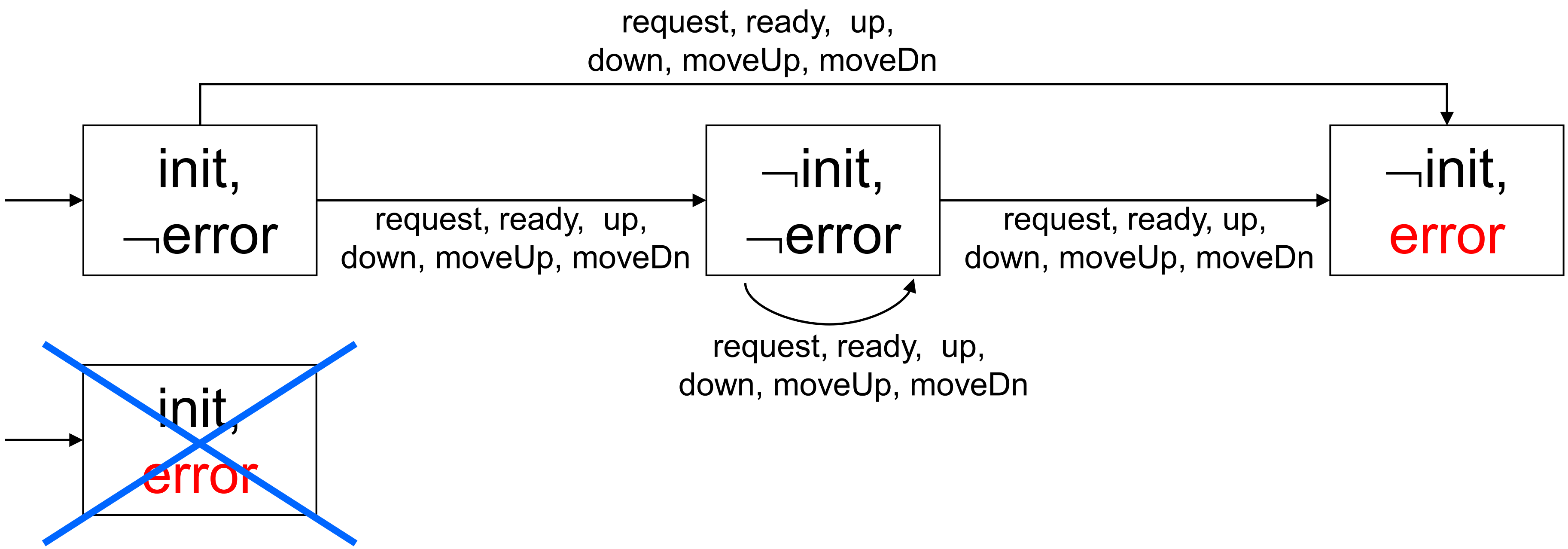


init	$pc{=}0 \wedge current{\le}Max \wedge input{\le}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\le}Max$
ready	$pc \ge 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\le}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
down	$pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$
moveUp	$pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$
moveDn	$pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$

Initial Abstraction

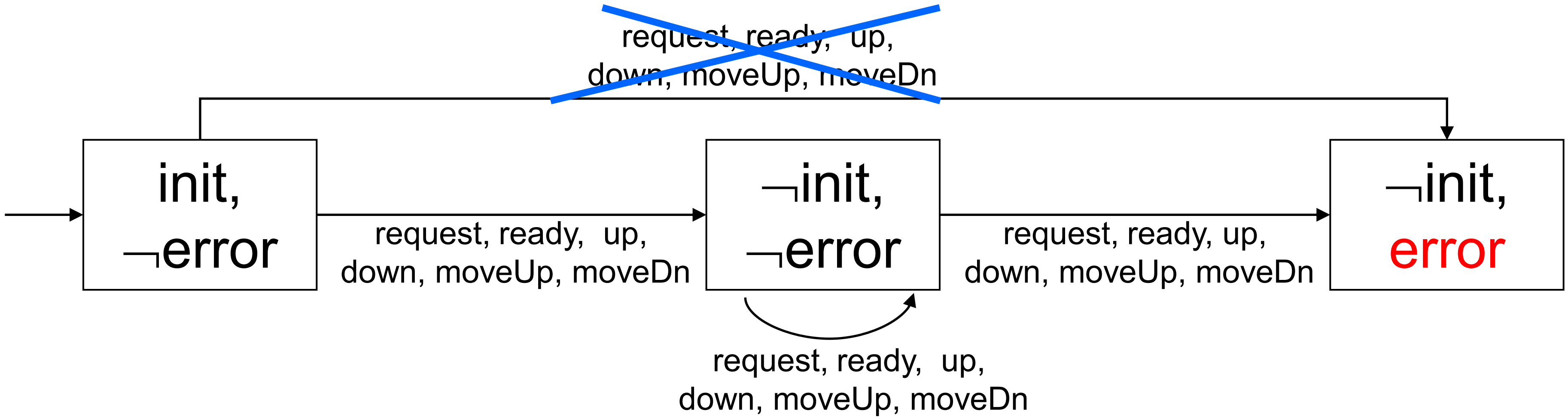


Slicing



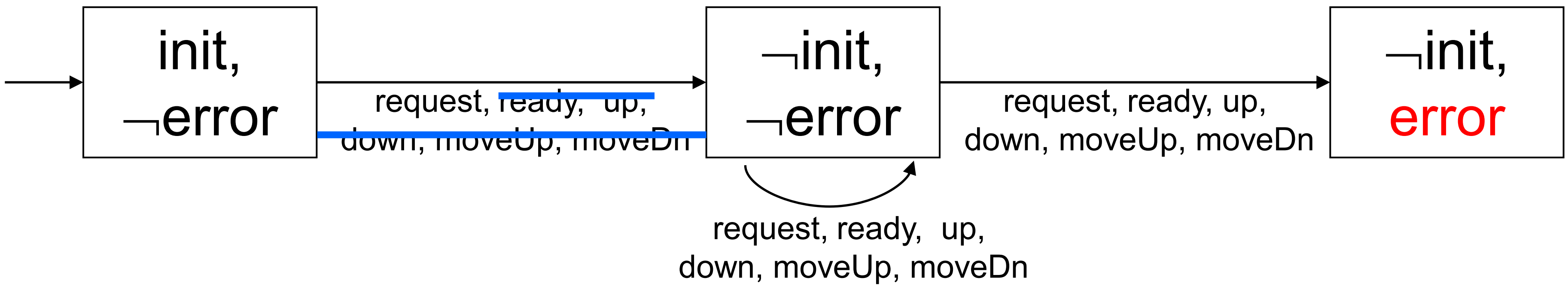
init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
down	$pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$
moveUp	$pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$
moveDn	$pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$

Slicing



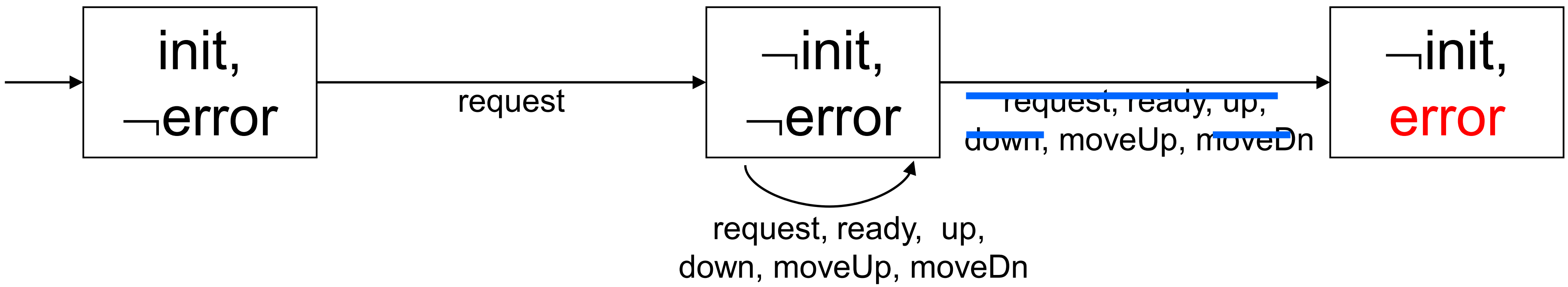
$init$	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
$error$	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
down	$pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$
moveUp	$pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$
moveDn	$pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$

Slicing



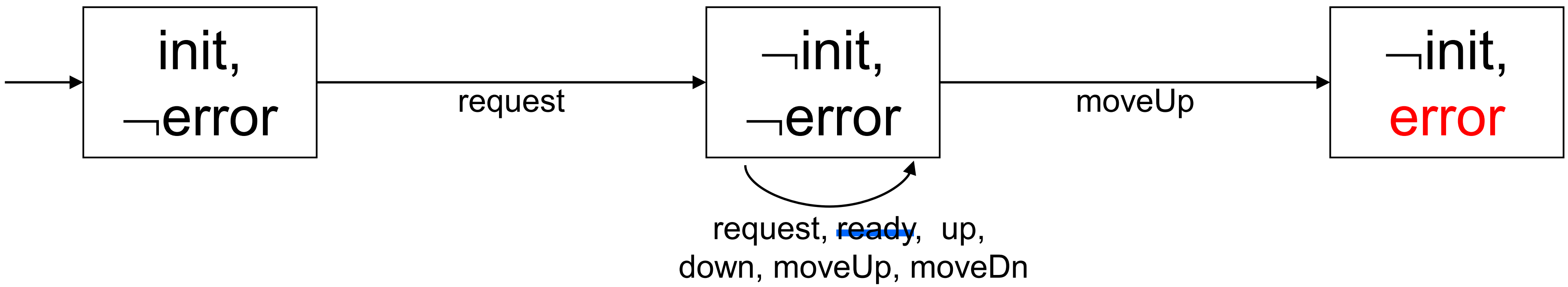
$init$	$pc=0 \wedge current \leq Max \wedge input \leq Max$
$error$	$current > Max$
request	$pc=0 \wedge pc'=1 \wedge current'=current \wedge req'=input \wedge input \leq Max$
ready	$pc \geq 1 \wedge req=current \wedge pc'=0 \wedge current'=current \wedge req'=req \wedge input' \leq Max$
up	$pc=1 \wedge req > current \wedge pc'=2 \wedge current'=current \wedge req'=req$
down	$pc=1 \wedge req < current \wedge pc'=3 \wedge current'=current \wedge req'=req$
moveUp	$pc=2 \wedge req > current \wedge pc'=2 \wedge current'=current+1 \wedge req'=req$
moveDn	$pc=3 \wedge req < current \wedge pc'=3 \wedge current'=current-1 \wedge req'=req$

Slicing



init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
down	$pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$
moveUp	$pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$
moveDn	$pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$

Slicing



$init$	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
$error$	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
down	$pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$
moveUp	$pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$
moveDn	$pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$

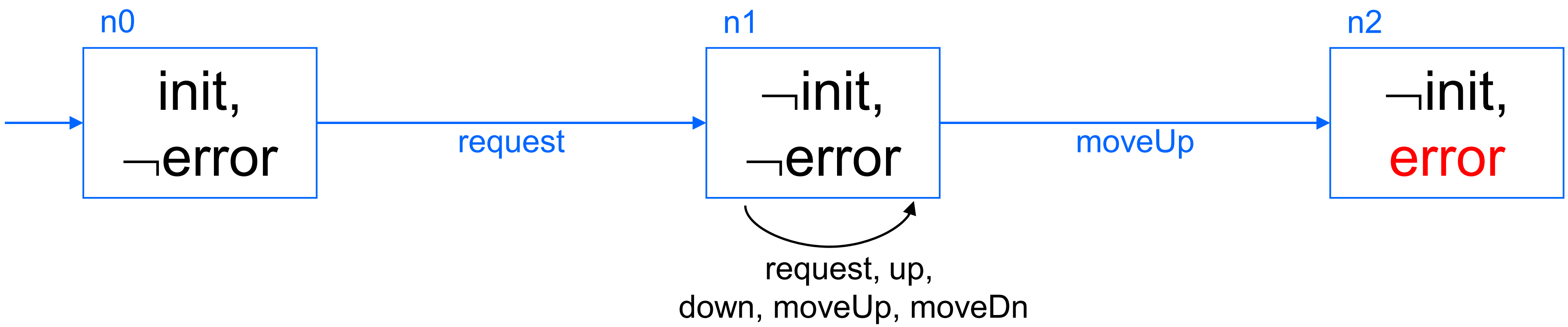
Error Path Analysis

1. Error Path concretizable?

2. If yes: System incorrect

3. If no: Node split
   - Find minimal error path
   - Determine node to split
   - Determine splitting predicate

Error Path Analysis

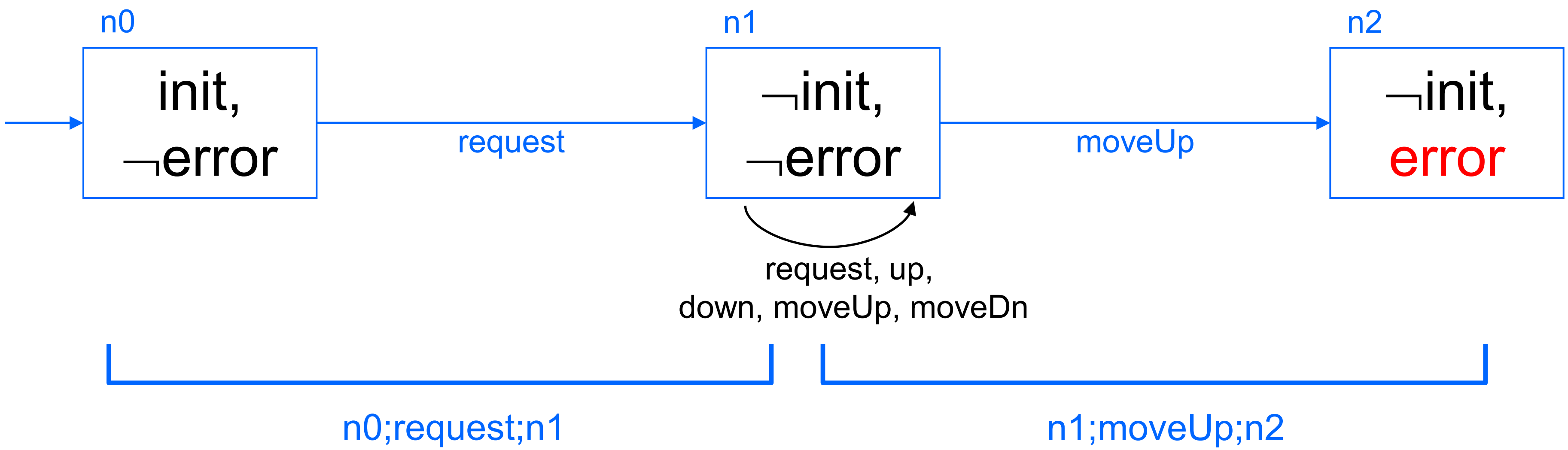


Error path concretizable?

Φ(n0;request;n1;moveUp;n2) =
n0(V^0) ∧ request(V^0,V^1) ∧ n1(V^1) ∧ moveUp(V^1,V^2) ∧ n2(V^2)

is unsatisfiable ⇒ n0;request;n1;moveUp;n2 is not concretizable.

Error Path Analysis



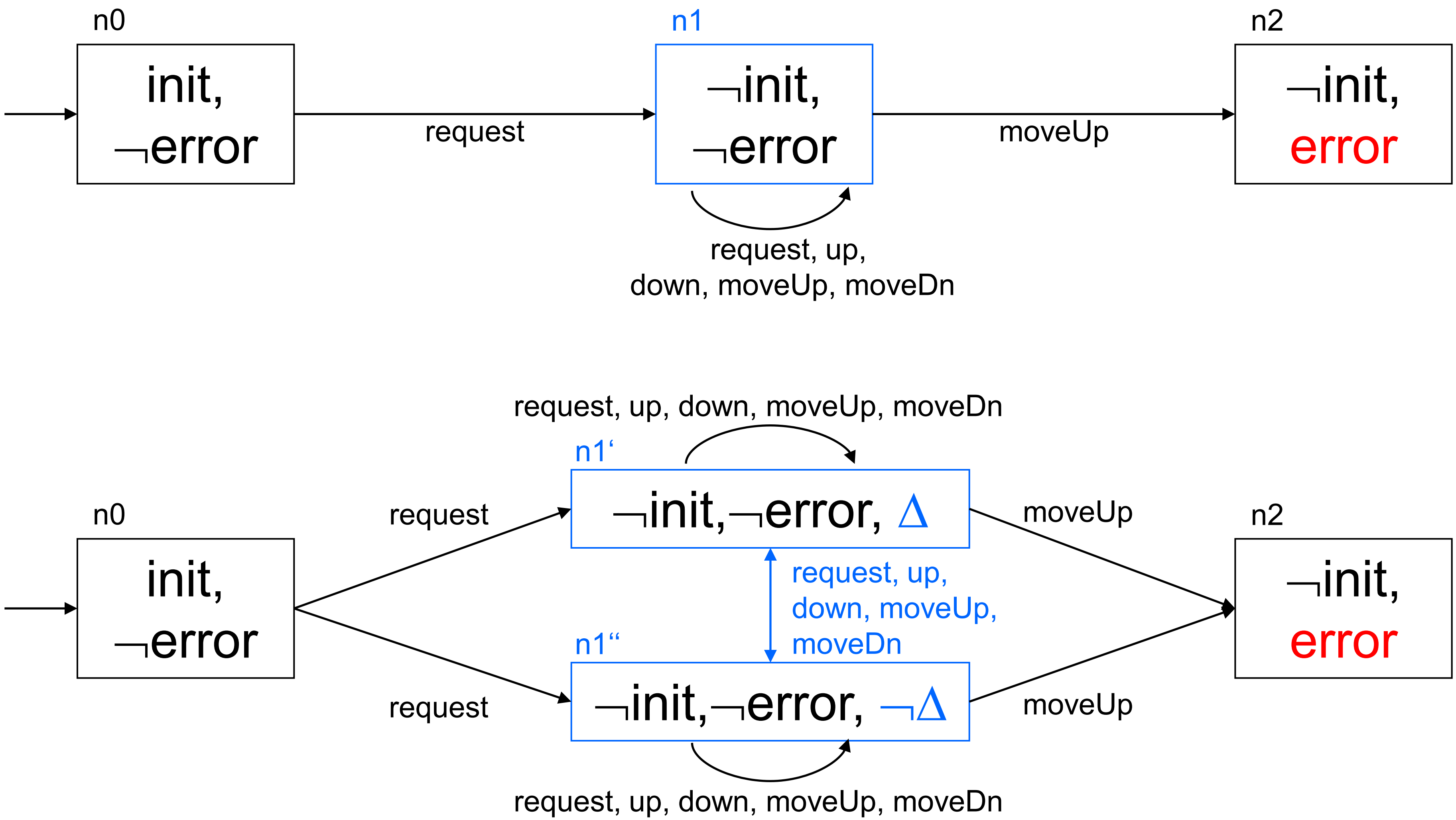
Error path minimal?

Φ(n0;request;n1) is satisfiable. Φ(n1;moveUp;n2) is satisfiable.

⇒ **n0;request;n1;moveUp;n2 is minimal.**

⇒ **Split node n1.**

Node Split


n0
init,
¬error
request
n1
¬init,
¬error
moveUp
n2
¬init,
error
request, up,
down, moveUp, moveDn
request, up, down, moveUp, moveDn
n1'
¬init,¬error, Δ
n0
init,
¬error
request
moveUp
n2
¬init,
error
request, up,
down, moveUp,
moveDn
n1''
¬init,¬error, ¬Δ
request
moveUp
request, up, down, moveUp, moveDn

Interpolation

Φ(n0;request;n1) = n0(V^0) $\wedge$ request(V^0,V^1) $\wedge$ n1(V^1) <u>**satisfiable**</u>

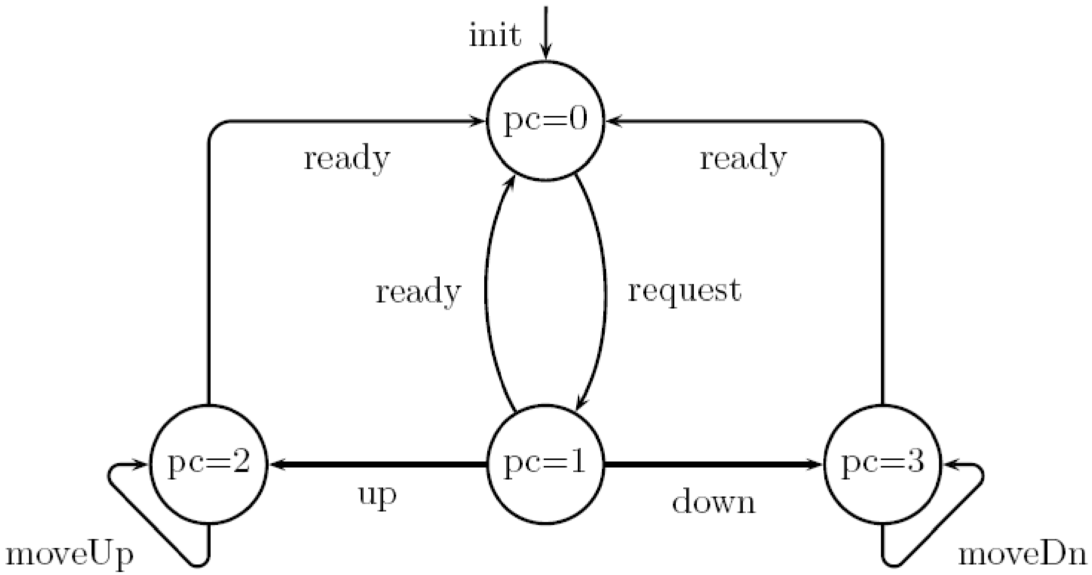
Φ(moveUp;n2) = moveUp(V^1,V^2) $\wedge$ n1(V^2) <u>**satisfiable**</u>

Φ(n0;request;n1;moveUp;n2) = Φ(n0;request;n1) $\wedge$ Φ(moveUp;n2) <u>**unsatisfiable**</u>

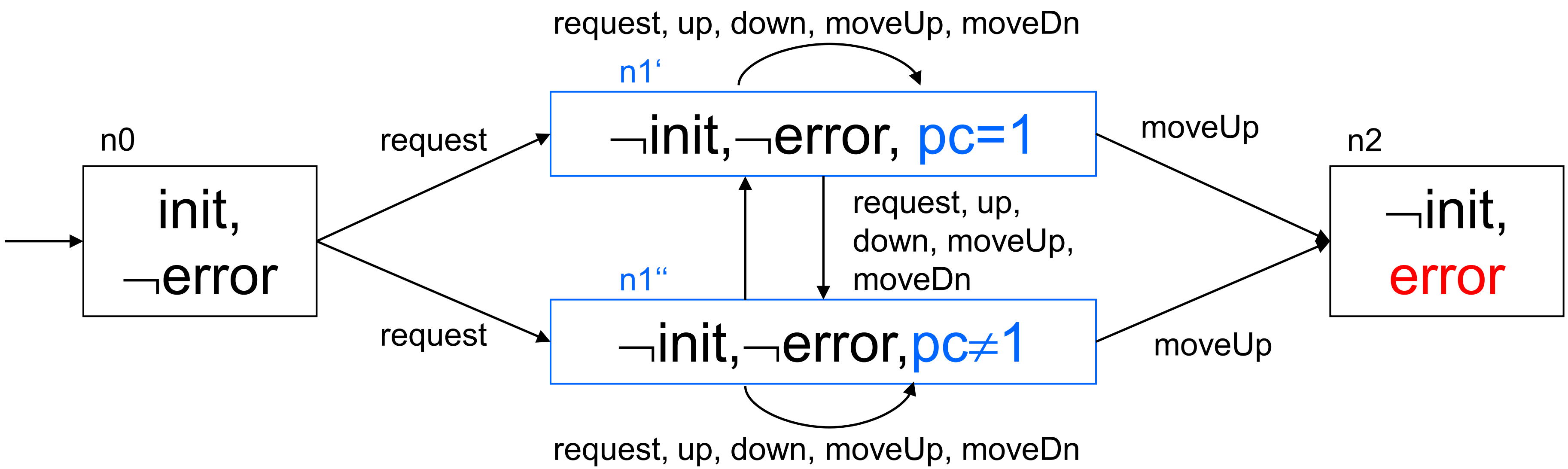
$\Rightarrow$ There exists a Craig interpolant Δ^1, such that

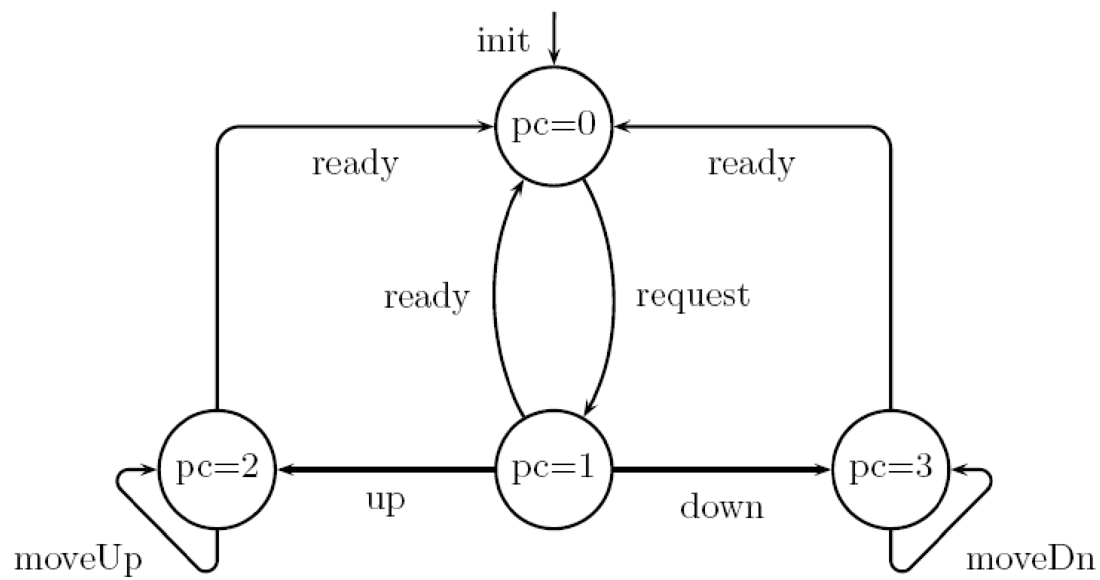
- Φ(n0;request;n1) $\Rightarrow \Delta^1$
- Φ(moveUp;n2) $\Rightarrow \neg\Delta^1$
- Variables(Δ^1) $\subseteq$ V^1

Δ^1 = pc^1=1

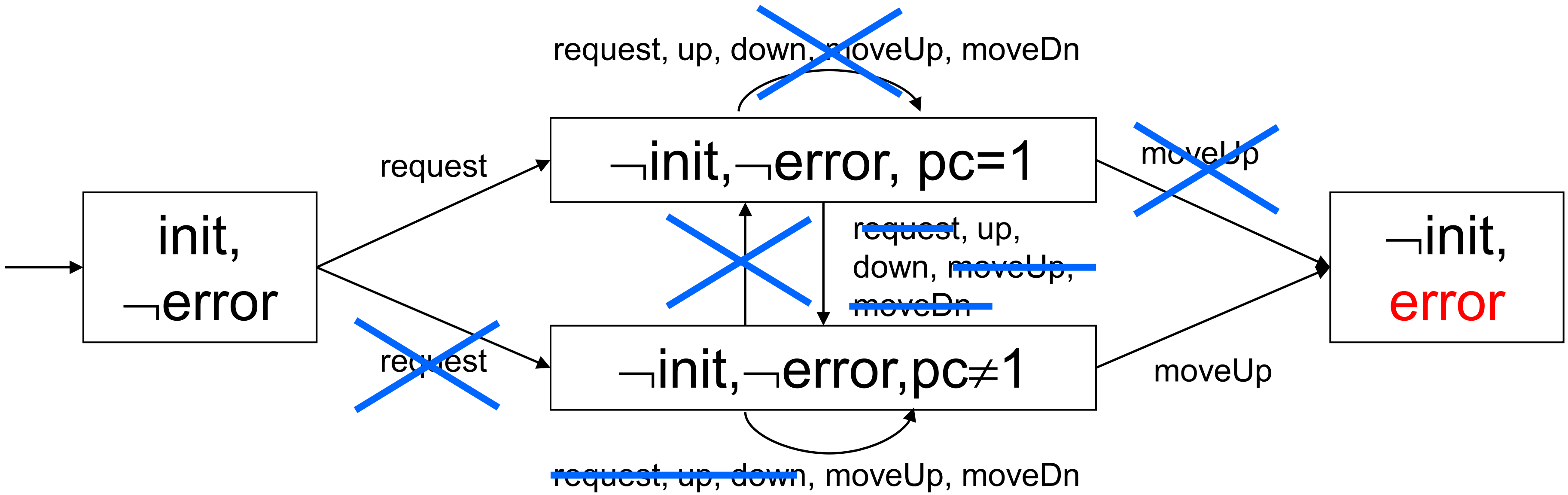


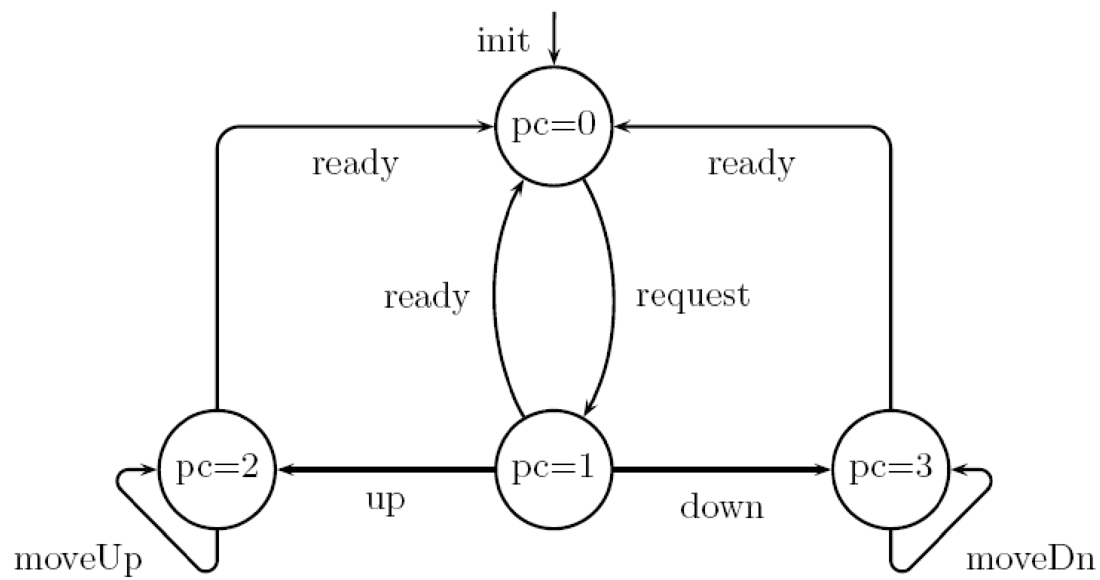
Splitting


request, up, down, moveUp, moveDn
n1'
¬init,¬error, pc=1
n0
init,
¬error
request
request
moveUp
n2
¬init,
error
request, up,
down, moveUp,
moveDn
n1''
¬init,¬error,pc≠1
moveUp
request, up, down, moveUp, moveDn

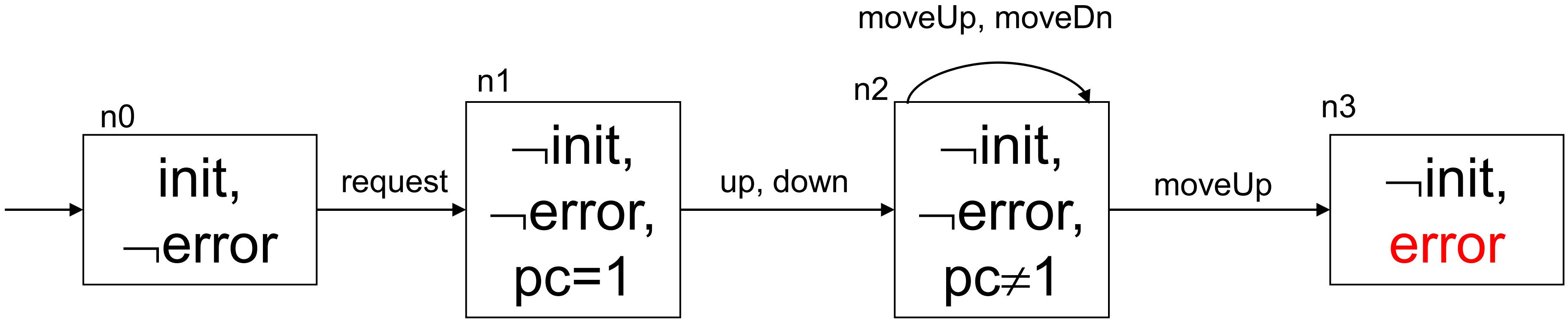

init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

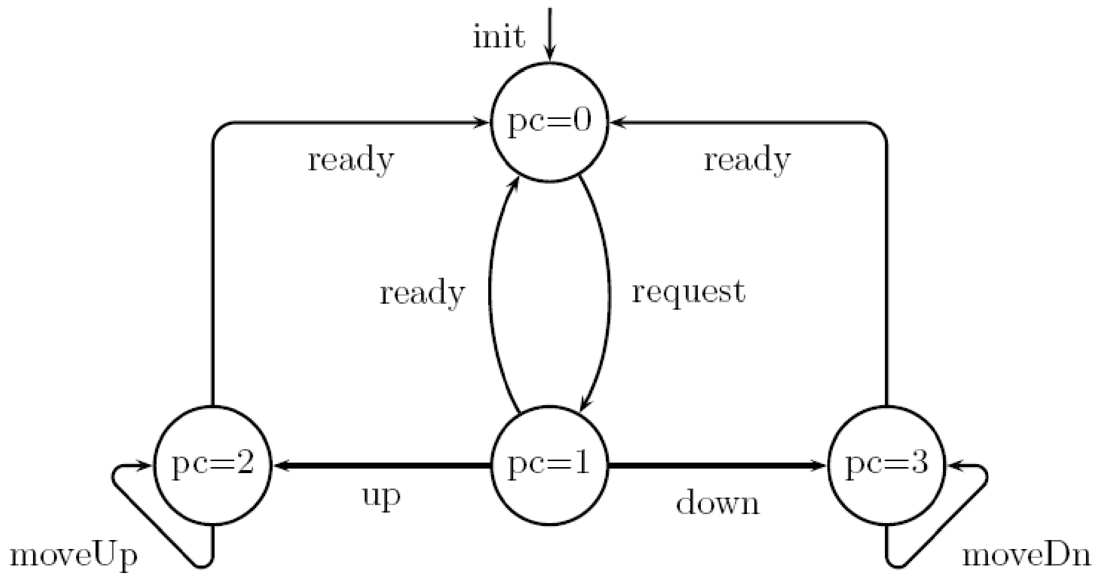
Slicing


request, up, down, moveUp, moveDn
request
¬init,¬error, pc=1
moveUp
init,
¬error
~~request~~, up,
down, ~~moveUp,~~
~~moveDn~~
¬init,
error
request
¬init,¬error,pc≠1
moveUp
~~request, up, down~~, moveUp, moveDn

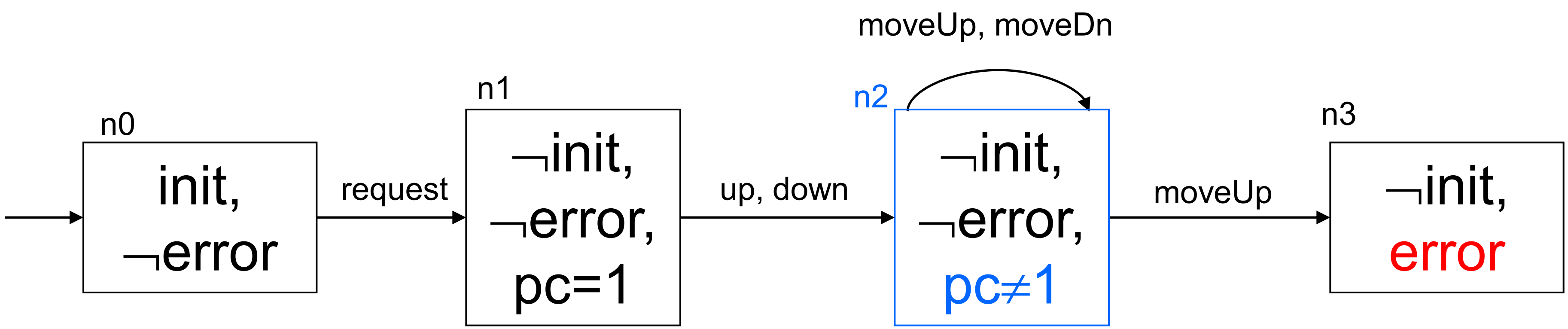

init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

Error Path Analysis

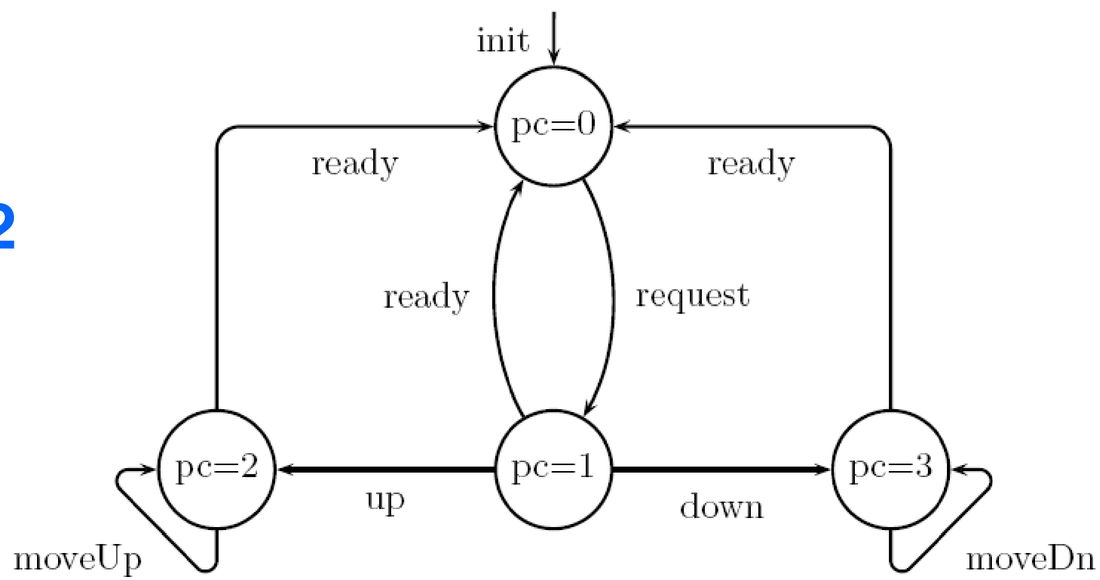




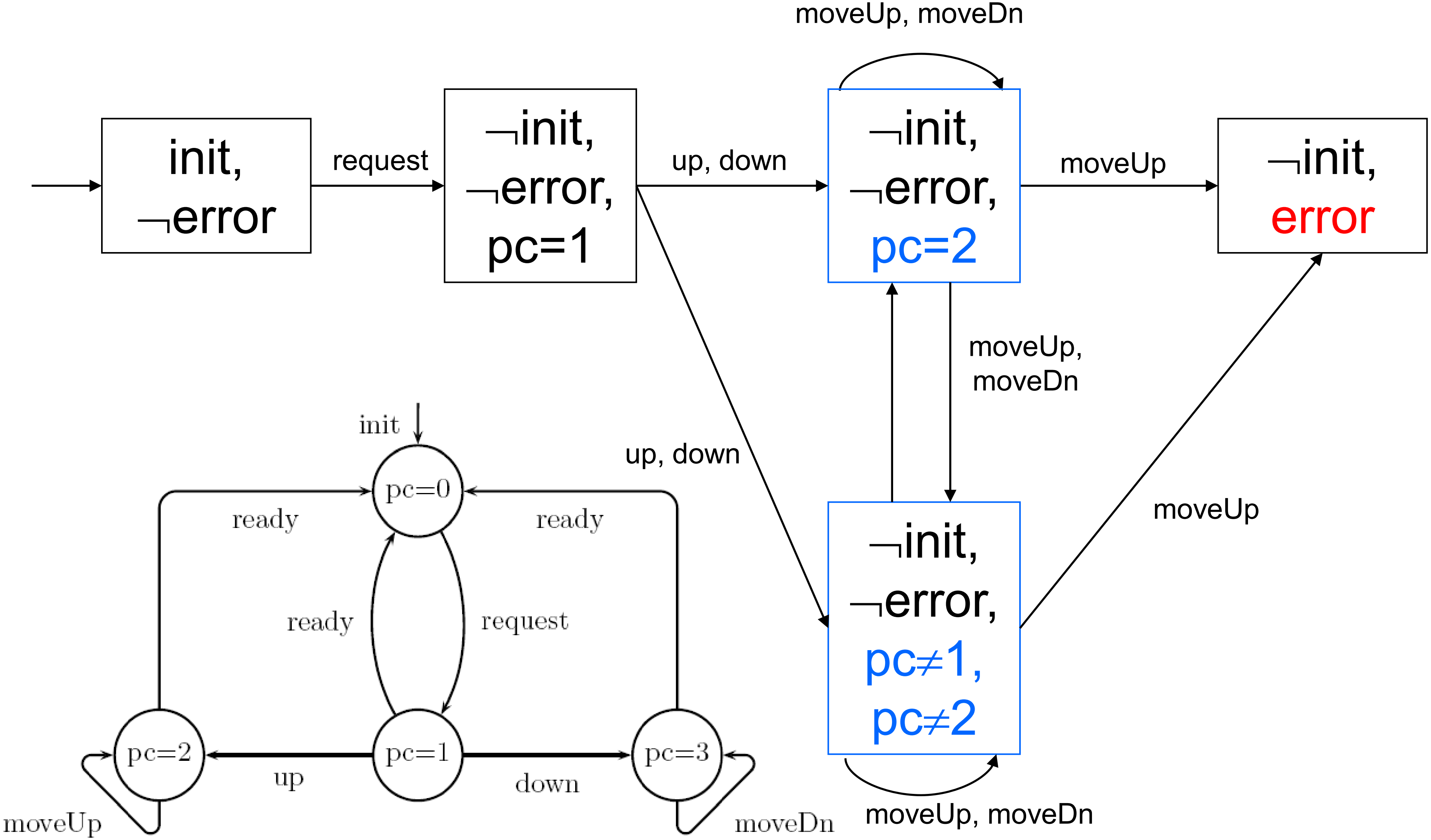
Error Path Analysis



Split node n2 with pc=2

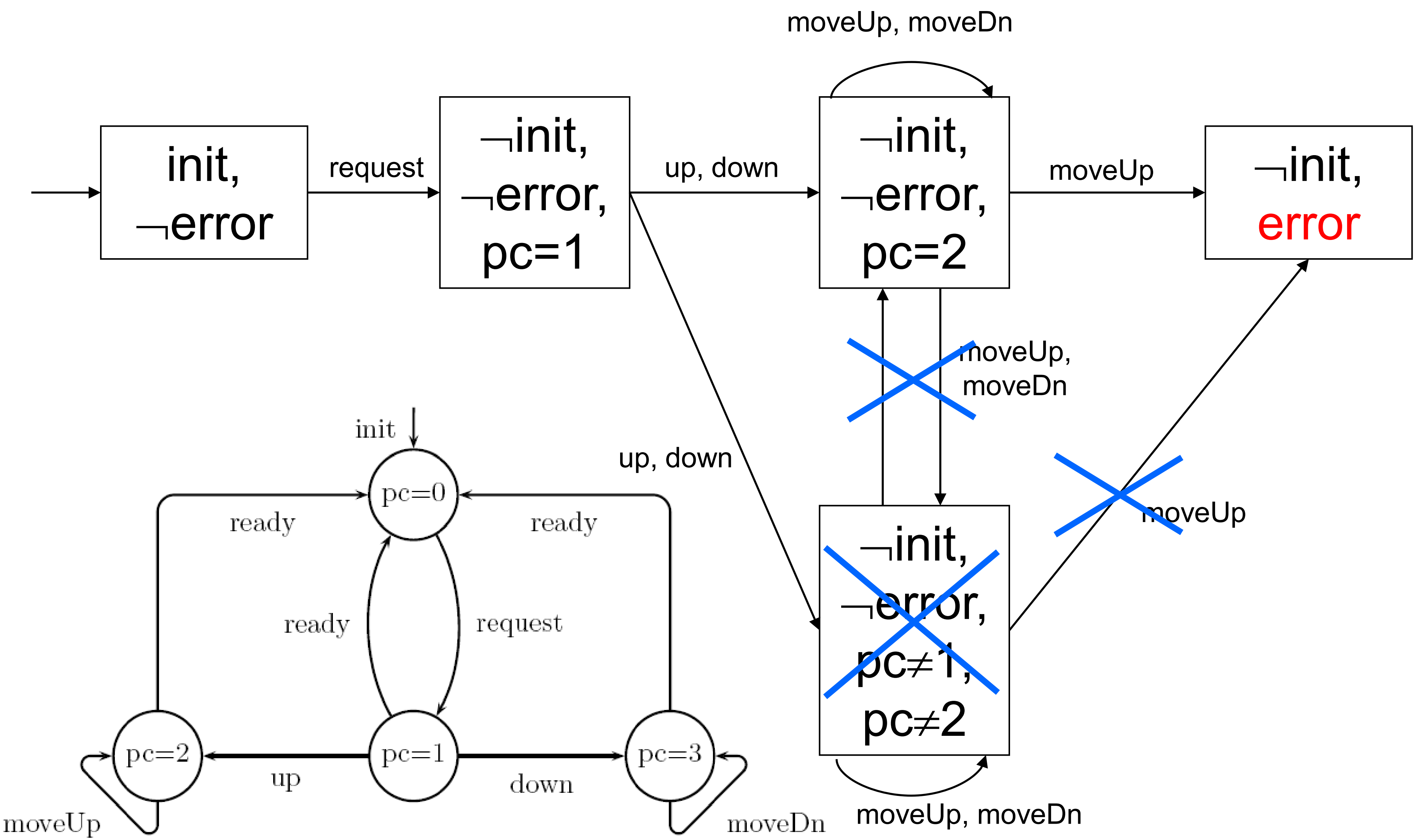


Splitting

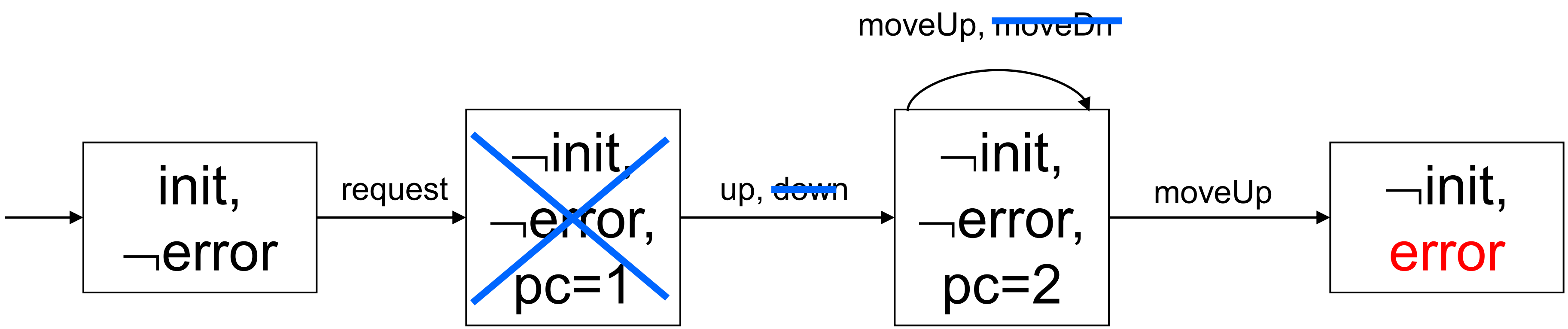


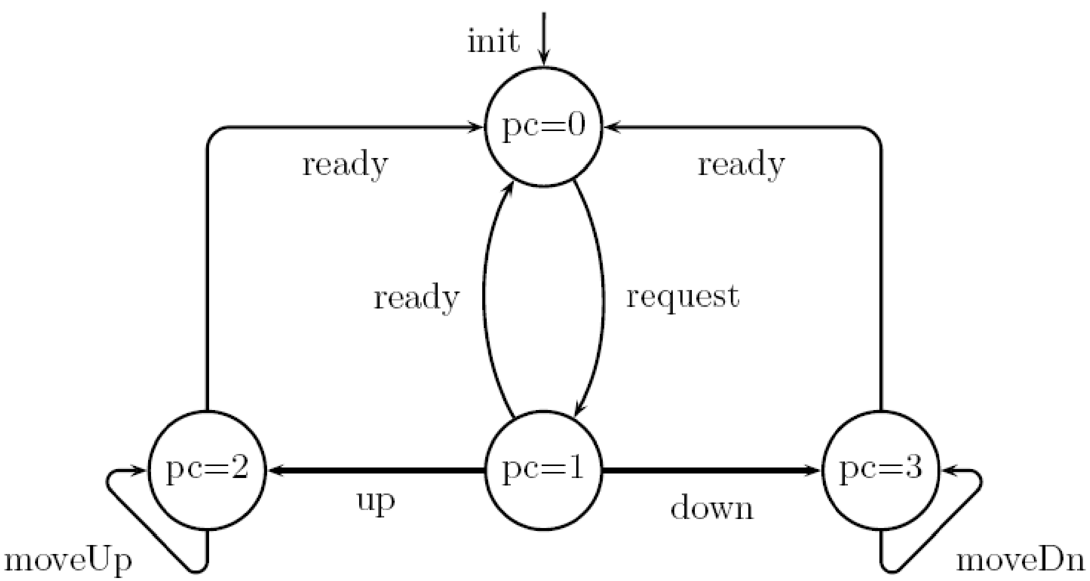
moveUp, moveDn
init,
¬error
request
¬init,
¬error,
pc=1
up, down
¬init,
¬error,
pc=2
moveUp
¬init,
error
moveUp,
moveDn
up, down
moveUp
¬init,
¬error,
pc≠1,
pc≠2
moveUp, moveDn
init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

Slicing

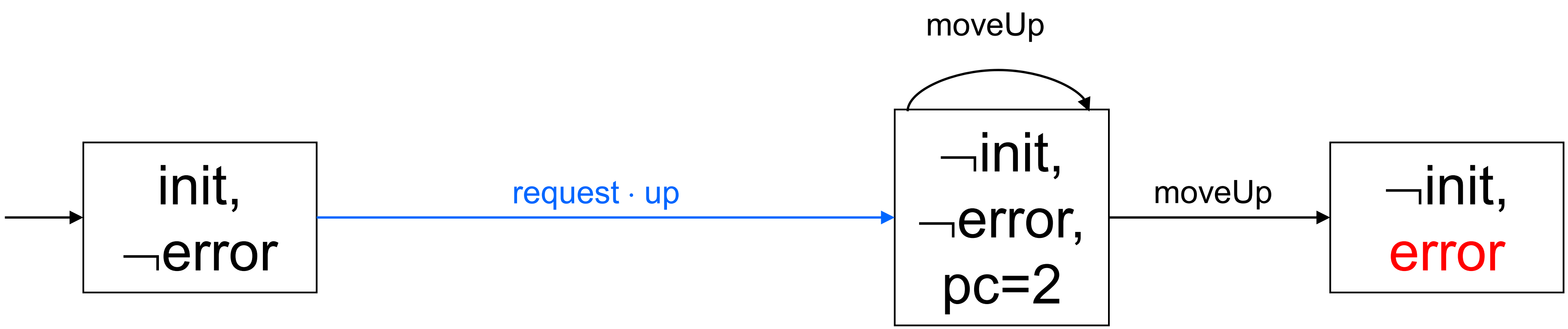

init, ¬error
request
¬init, ¬error, pc=1
up, down
moveUp, moveDn
¬init, ¬error, pc=2
moveUp
¬init, error
moveUp, moveDn
moveUp
up, down
¬init, ¬error, pc≠1, pc≠2
moveUp, moveDn
init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

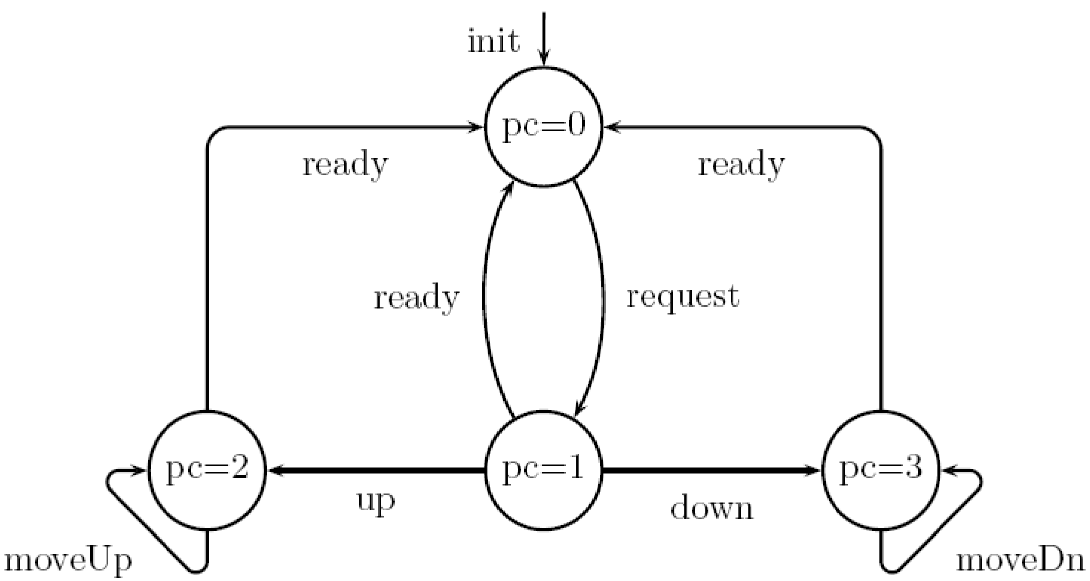
Slicing


init,
¬error
request
¬init,
¬error,
pc=1
up, ~~down~~
moveUp, ~~moveDn~~
¬init,
¬error,
pc=2
moveUp
¬init,
error

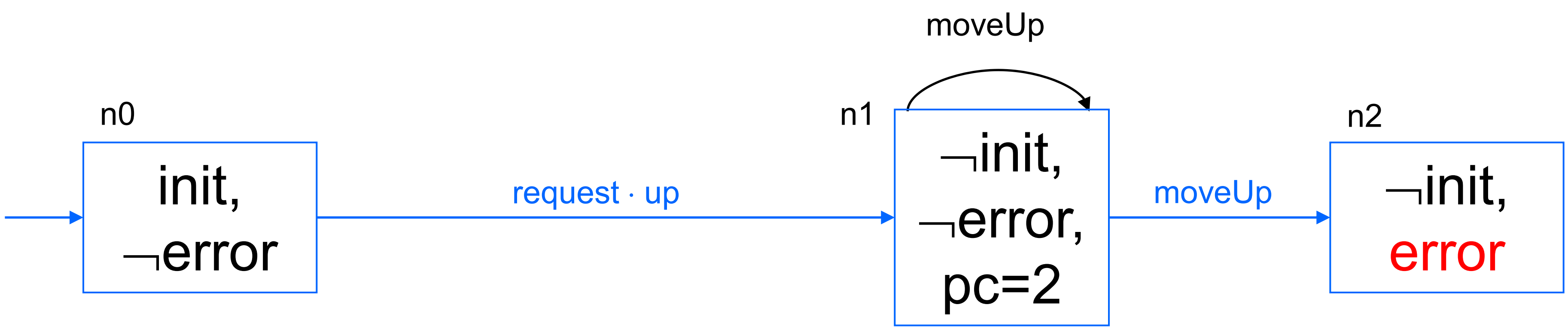

init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

Slicing


moveUp
init,
¬error
request · up
¬init,
¬error,
pc=2
moveUp
¬init,
error

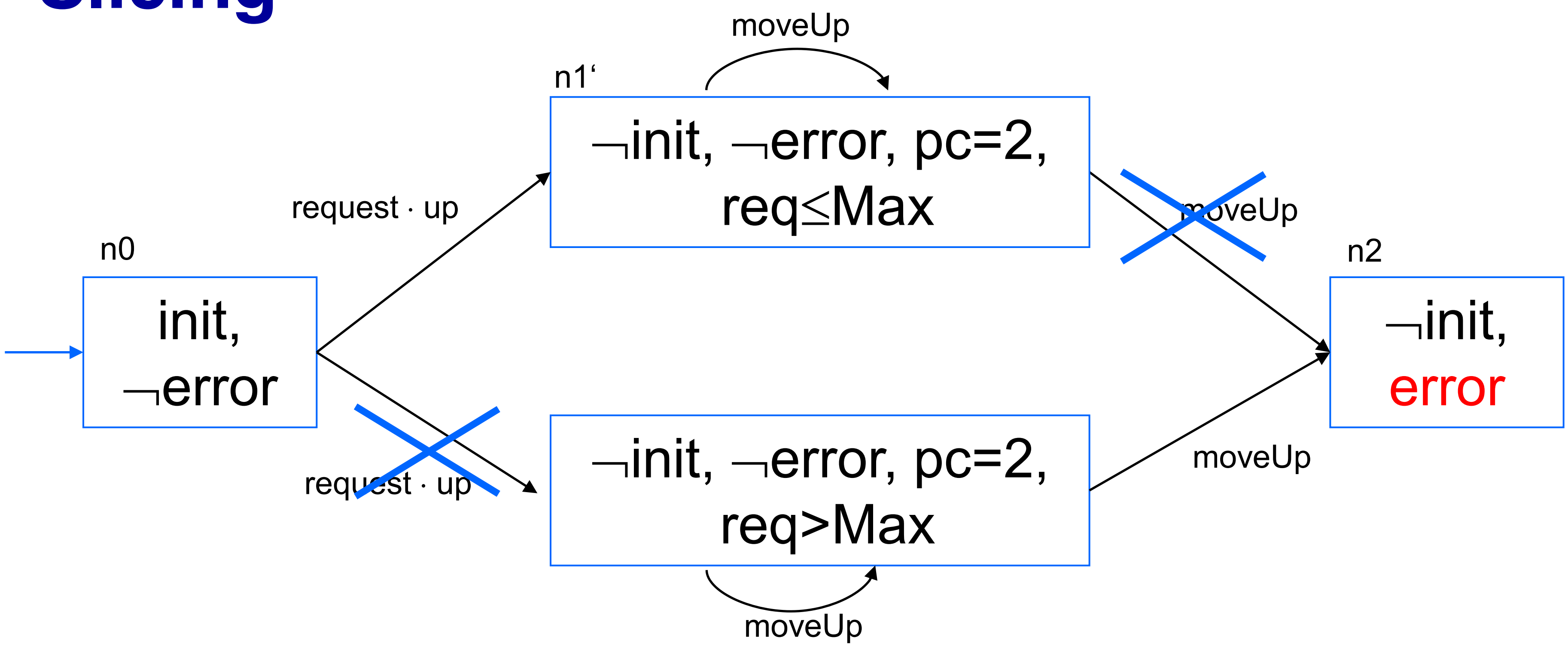

init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

Error Path Analysis



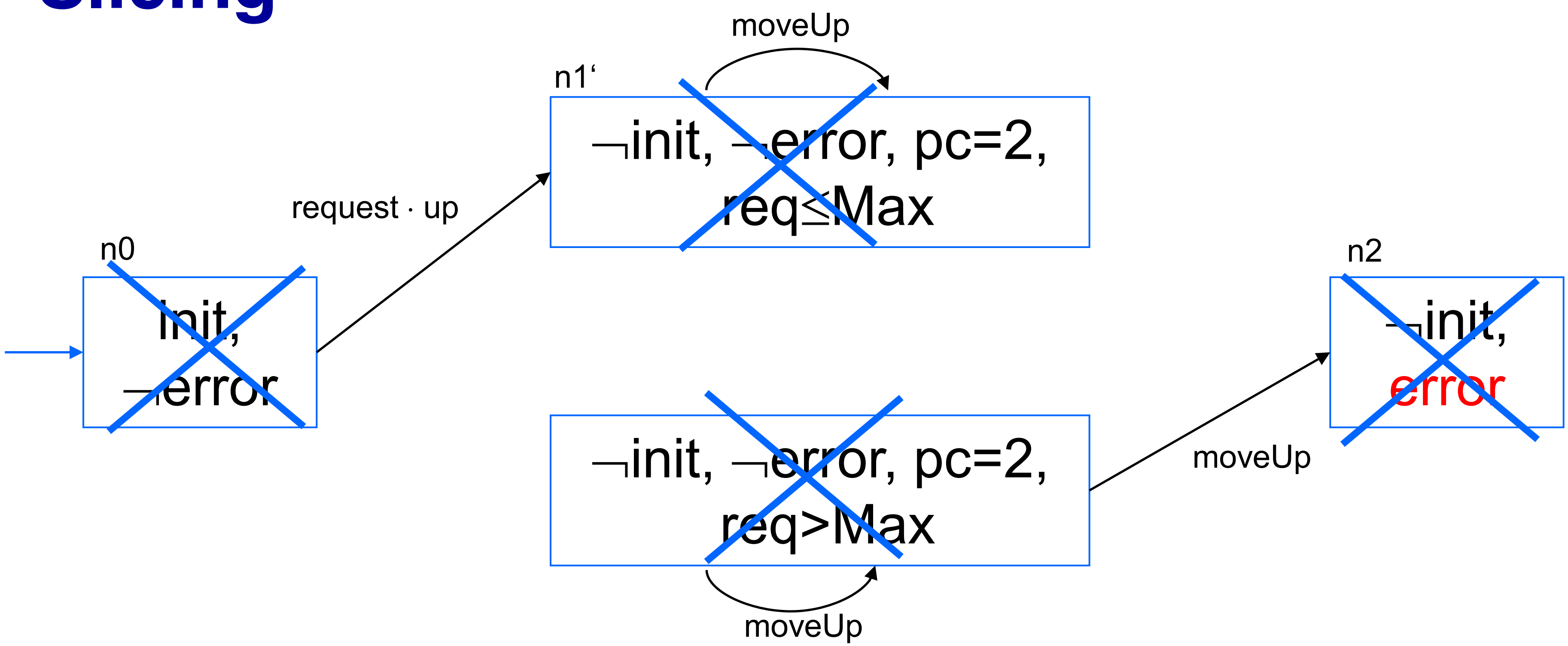
Split node n1 with req≤Max

Slicing



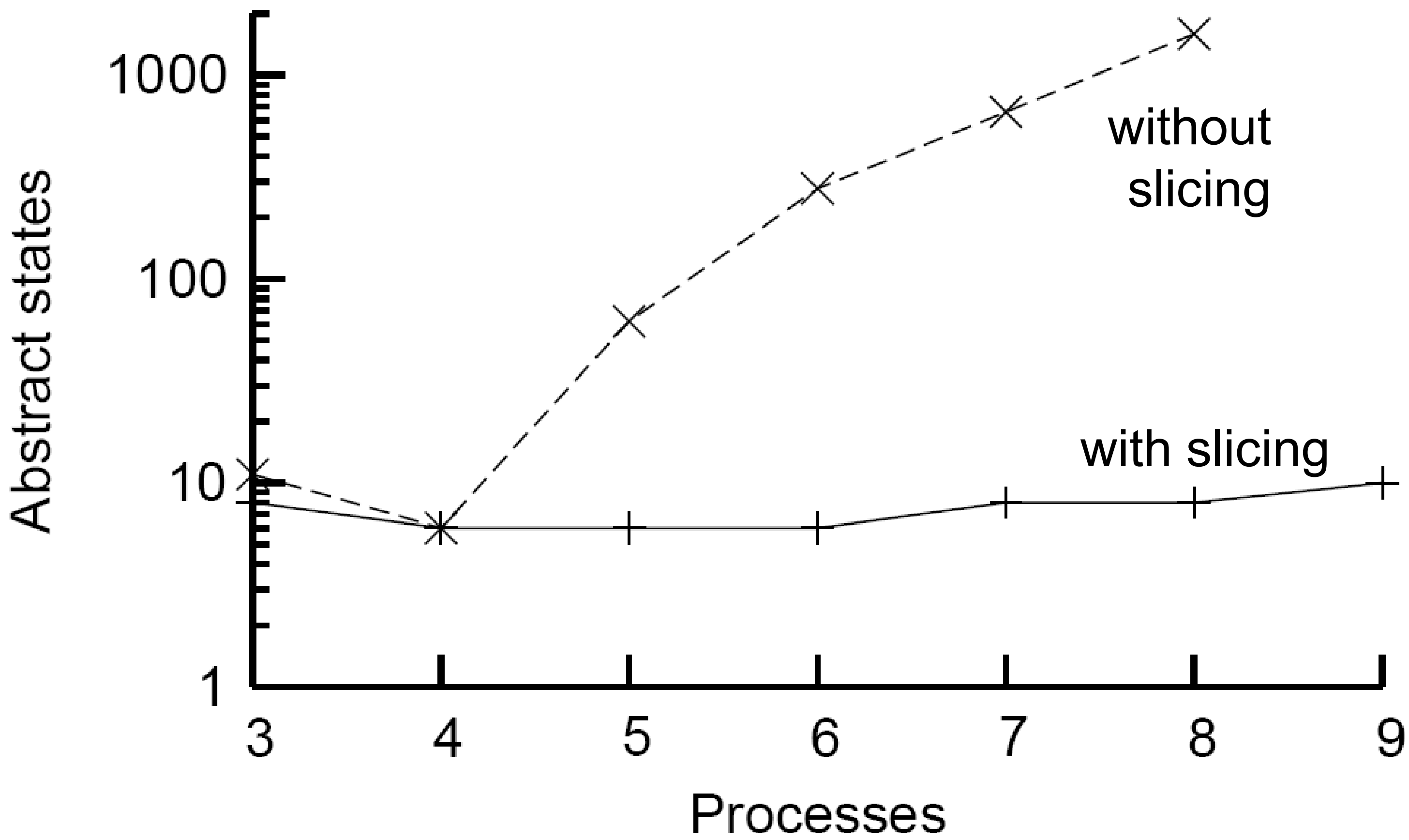
init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
down	$pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$
moveUp	$pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$
moveDn	$pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$

Slicing

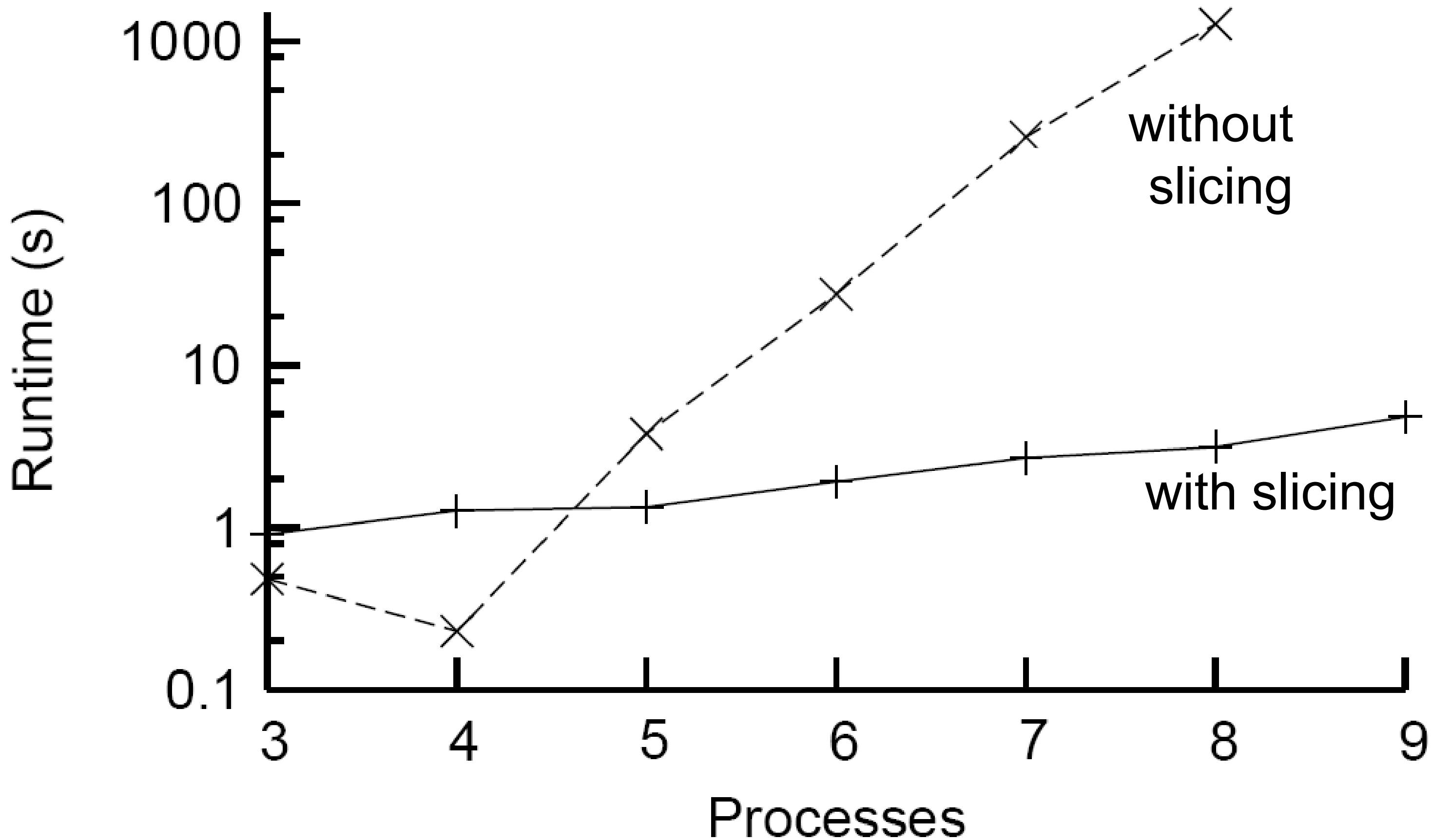


$init$	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
$error$	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
down	$pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$
moveUp	$pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$
moveDn	$pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$

Experiments: State Space

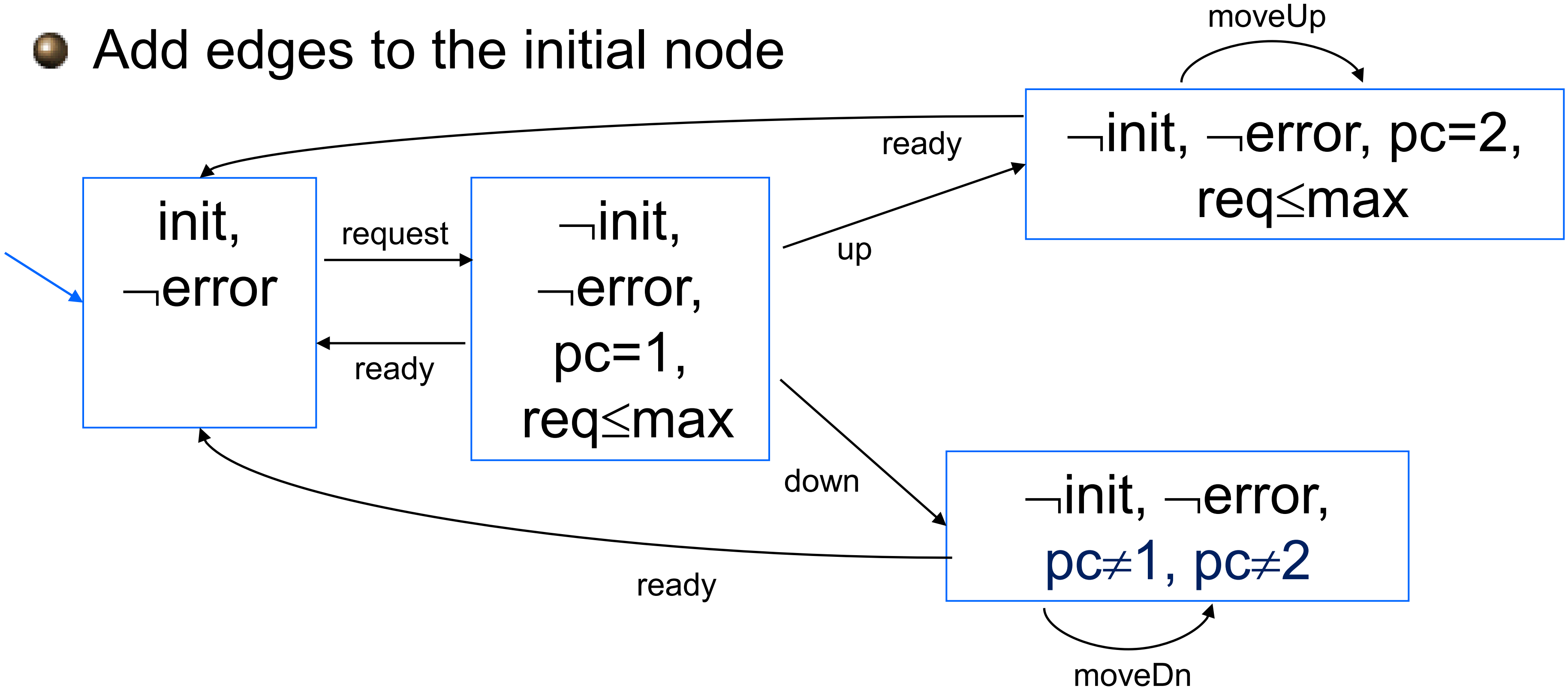


Experiments: Runtime



Verification diagrams as certificates

- Add intermediate nodes for composite transitions (using strongest postcondition)
- Do not remove nodes that are not backward reachable but still forward-reachable
- Add edges to the initial node



Review

Prof. Bernd Finkbeiner, Ph.D.
Peter Faymonville, M.Sc.
Michael Gerke, B.Sc.

Winter term 2011/2012
Problem Set 13

Verification

Please write the names of all group members on the solutions you hand in.

Problem 1: Invariance Diagrams

Consider the transition system DEQUE in Figure 1, representing a ring buffer for a double-ended queue. The buffer consists of five cells (represented by integer variables), which can be either free (0) or occupied (1). Starting with a single occupied cell x_1, we can toggle a cell's state if the states of its neighbors differ.

Θ	$x_1 = 1 \wedge x_2 = 0 \wedge x_3 = 0 \wedge x_4 = 0 \wedge x_5 = 0$
ρ_1	$x_5 + x_2 = 1 \wedge x_1' = 1 - x_1 \wedge pres(x_2, x_3, x_4, x_5)$
ρ_2	$x_1 + x_3 = 1 \wedge x_2' = 1 - x_2 \wedge pres(x_1, x_3, x_4, x_5)$
ρ_3	$x_2 + x_4 = 1 \wedge x_3' = 1 - x_3 \wedge pres(x_1, x_2, x_4, x_5)$
ρ_4	$x_3 + x_5 = 1 \wedge x_4' = 1 - x_4 \wedge pres(x_1, x_2, x_3, x_5)$
ρ_5	$x_4 + x_1 = 1 \wedge x_5' = 1 - x_5 \wedge pres(x_1, x_2, x_3, x_4)$

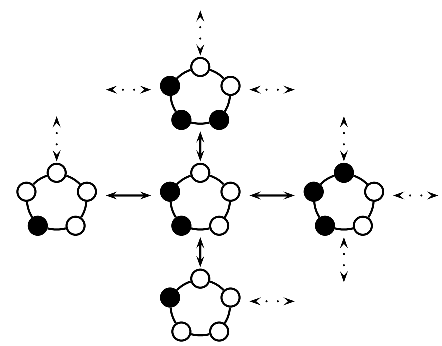


Figure 1: DEQUE transition system.

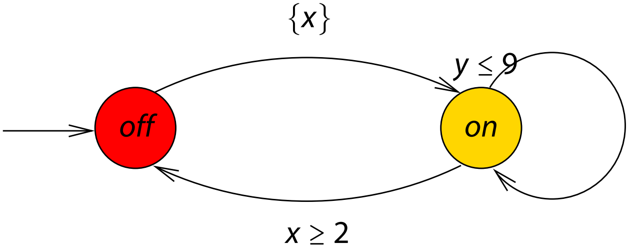
Create an INVARIANCE diagram which proves for the DEQUE system that the state with all cells occupied is not reachable.

Hints:

- Keep it simple - the verification diagram in the sample solution only has five nodes.
- State any auxiliary invariants needed to prove P-validity.
- You do not need to give proofs for individual verification conditions.

True or False?

The following timed automaton satisfies EF *on*:



True or False?

Each nonzeno timed automaton is timelock-free.

True or False?

The state graph and the region graph of a timed automaton are bisimilar over AP'.

True or False?

Clock equivalence is a bisimulation.

True or False?

If there is a P-inductive program annotation, then P is partially correct.

True or False?

It holds that

$$wp(F, \mathtt{assume}\ c) = F \wedge c$$

True or False?

$$f(a) = f(b) \rightarrow a = b$$

is T_E-satisfiable.

True or False?

T_E is decidable.

True or False?

$$a[i] = e \rightarrow a\langle i \triangleleft e\rangle = a$$

is T_A-valid.

True or False?

The quantifier-free fragment of the theory of arrays with extensionality is decidable.

True or False?

The limitations of the Nelson-Oppen method are as follows:
Given formula F in theory $T_1 \cup T_2$.

1. F must be quantifier-free.
2. Signatures Σ_i of the combined theory only share $=$, i.e.,

$$\Sigma_1 \cap \Sigma_2 = \{=\}$$

3. Theories T_1, T_2 must be stably infinite.
4. Theories T_1, T_2 must be convex.

True or False?

The quantifier-free fragment of the theory of arrays with extensionality is stably infinite.

True or False?

The quantifier-free fragment of the theory of arrays with extensionality is convex.

True or False?

A P-valid invariance diagram labeled with assertions $\varphi_1, \varphi_2, \ldots \varphi_n$ establishes that

$$\Box \left(\bigvee_{i=1}^{n} \varphi_i \right)$$

is P-valid.