Verification – Lecture 27
Abstraction

Bernd Finkbeiner – Sven Schewe
Rayna Dimitrova – Lars Kuhtz – Anne Proetzsch

Wintersemester 2007/2008

Abstraction

α

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** $(y_2 = 0 \lor y_1 \leq y_2)$
- ℓ_3: **critical**
- ℓ_4: $y_1 := 0$

||

loop forever do
- m_0: **noncritical**
- m_1: $y_2 := y_1 + 1$
- m_2: **await** $(y_1 = 0 \lor y_2 < y_1)$
- m_3: **critical**
- m_4: $y_2 := 0$

local b_1, b_2, b_3 : **boolean** **where** b_1, b_2, b_3

loop forever do
- ℓ_0: **noncritical**
- ℓ_1: $(b_1, b_3) := (\mathit{false}, \mathit{false})$
- ℓ_2: **await** $(b_2 \lor b_3)$
- ℓ_3: **critical**
- ℓ_4: $(b_1, b_3) := (\mathit{true}, \mathit{true})$

||

loop forever do
- m_0: **noncritical**
- m_1: $(b_2, b_3) := (\mathit{false}, \mathit{true})$
- m_2: **await** $(b_1 \lor \neg b_3)$
- m_3: **critical**
- m_4: $(b_2, b_3) := (\mathit{true}, b_1)$

Abstract Interpretation: Galois Connection

Let $(A, \leq)$ and $(C, \subseteq)$ be partially ordered sets.

A pair (α,γ) is a Galois connection iff the following hold

1. $\alpha: C \to A$, $\gamma: A \to C$
2. α and γ are monotone
3. $S \subseteq \gamma(\alpha(S))$ for all $S \in C$
4. $\alpha(\gamma(x)) \leq x$ for all $x \in A$

If $\alpha(\gamma(x)) = x$, then (α,γ) is a Galois insertion.

Example

- Concrete system: Multiplication of integers
- Question: is the result of the multiplication less than, greater than, or equal to zero?

Concrete domain: sets of integers $C = 2^{\mathbf{Z}}$
Multiplication of sets of integers:
$S1 * S2 = \{ s3 \in \mathbf{Z} \mid \exists s1 \in S1, \exists s2 \in S2 \,.\, s1 * s2 = s3 \}$

Abstract domain: $A = \{ neg, zero, pos \}$
Concretization function: $\gamma: A \to C$
$\gamma(neg) = \{ x \in \mathbf{Z} \mid x < 0 \}$
$\gamma(zero) = \{ 0 \}$
$\gamma(pos) = \{ x \in \mathbf{Z} \mid x > 0 \}$

Abstraction Function

Abstraction function $\alpha: C \rightarrow A$

$\alpha(\{0\})$ = zero
$\alpha(S)$ = pos if $\forall x \in S \,.\, x > 0$
$\alpha(S)$ = neg if $\forall x \in S \,.\, x < 0$
$\alpha(S)$ = ??? otherwise

Introduce $\top$ (top) with $\gamma(\top) = \mathbf{Z}$
and $\bot$ (bottom) with $\gamma(\bot) = \varnothing$

$\alpha(\varnothing) = \bot$
$\alpha(S) = \top$ otherwise.
$\Sigma_A = \{\bot, neg, zero, pos, \top\}$

Abstract Multiplication

$*^A$	$\bot$	neg	zero	pos	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
neg	$\bot$	pos	zero	neg	$\top$
zero	$\bot$	zero	zero	zero	zero
pos	$\bot$	neg	zero	pos	$\top$
$\top$	$\bot$	$\top$	zero	$\top$	$\top$

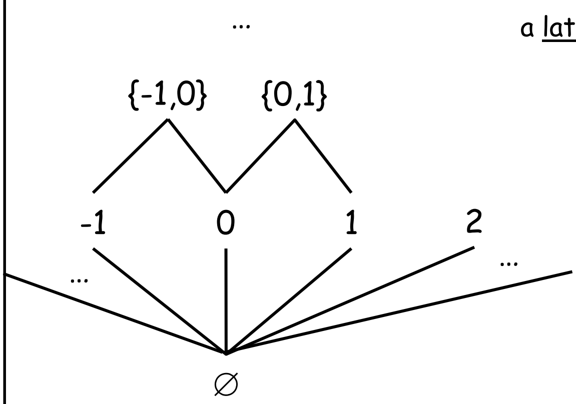
Example cont'd

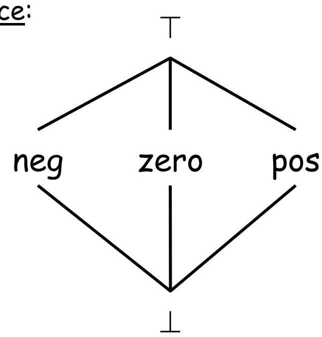
Is result of $x * y$ less than, greater than, or equal to zero?

1. Abstract x and y: $x^A = \alpha(\{x\})$, $y^A = \alpha(\{y\})$
2. Do abstract multiplication $z^A = x^A *^A y^A$
3. Concretize z^A: $S = \gamma(z^A)$
   - if $\forall s \in S \,.\, s > 0$, then $x * y > 0$
   - if $\forall s \in S \,.\, s < 0$, then $x * y < 0$
   - if $S = \{0\}$, then $x * y = 0$
   - (otherwise: not enough information to answer question)

Concrete and Abstract Domains

We impose an order on the abstract domain to obtain a <u>lattice</u>:





The concrete domain is a lattice ordered by $\subseteq$.

Observations

- α,γ are monotone:
  $S1 \subseteq S2 \quad \rightarrow \quad \alpha(S1) \leq \alpha(S2)$
  $a \leq b \quad \rightarrow \quad \gamma(a) \subseteq \gamma(b)$
- The result of abstraction followed by concretization is something larger or equal:
  $S \subseteq \gamma(\alpha(S))$
- The result of concretization followed by abstraction is the same object:
  $\alpha(\gamma(x)) = x$
- (α, γ) is a Galois insertion.

Abstracting Systems

- Concrete domain:
  Sets of states of the concrete system $C = 2^{\Sigma}$
- Abstract domain:
  States of some abstract system $A = \Sigma_A$

System S^A is an LTL property-preserving abstraction of system S if $L(S) \subseteq \gamma(L(S^A))$.

Concretizing Sets and Sequences

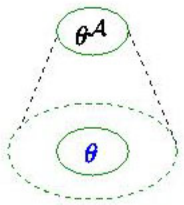
Given $\gamma: \Sigma_A \rightarrow 2^{\Sigma}$

- For $S^A \subseteq \Sigma_A : \gamma(S^A) = \bigcup_{s^A \in S^A} \gamma(s^A)$
- For $\sigma^A : s^A_0 \; s^A_1 \; s^A_2 \; \ldots \in \Sigma_A{}^*$
  $\gamma(\sigma^A) = \{s_0 \; s_1 \; s_2 \; \ldots \mid s_i \in \gamma(s^A{}_i) \text{ for all } i \geq 0\}$
- For a set of sequences L^A:

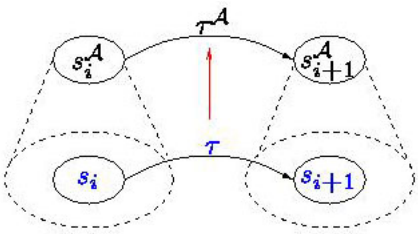
  $\gamma(L^A) = \bigcup_{\sigma^A \in L^A} \gamma(\sigma^A)$

Abstracting Programs

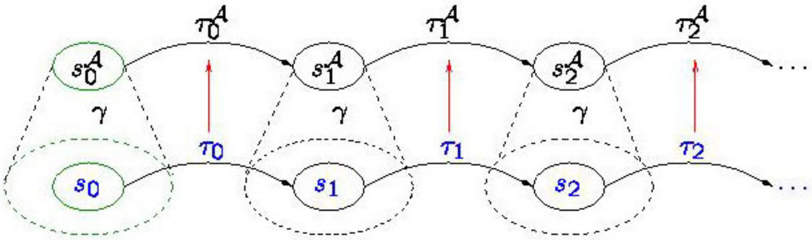
If: initial condition:



and: transitions:



Then:



Concretizing Transitions

- Transition relations $\tau: \Sigma \rightarrow 2^{\Sigma}$ $\tau_A: \Sigma_A \rightarrow 2^{\Sigma A}$

- $\gamma((s_A, S_A)) = \{ (s,S) \mid s \in \gamma(\{s_A\}), S \subseteq \gamma(S_A) \}$

- $\gamma(\tau_A) = \bigcup_{(s_A, S_A) \in \tau_A} \gamma((s_A, S_A))$

Abstracting Systems

Given S: (V, Θ, T),
construct some S^A: (V^A, Θ^A, T^A)
such that $L(S) \subseteq \gamma(L(S^A))$:

1. $\Theta \subseteq \gamma(\Theta^A)$,
2. $\tau \subseteq \gamma(\tau^A)$ for all $\tau^A \in T^A$.

Then: $L(S) \subseteq \gamma(L(S^A))$

Abstracting the Property

- The goal is to show $L(S) \subseteq L(\varphi)$.
- We are planning to prove $L(S^A) \subseteq L(\varphi^A)$.
- By monotonicity, $\gamma(L(S^A)) \subseteq \gamma(L(\varphi^A))$.
- To be able to infer from $L(S^A) \subseteq L(\varphi^A)$ that $L(S) \subseteq L(\varphi)$ we need

$$L(S) \subseteq \gamma(L(S^A)) \subseteq \boxed{\gamma(L(\varphi^A)) \subseteq L(\varphi)}$$

Program ANY

$$\text{local } x,\ y\text{: integer where } x = y = 0$$

$$P_1 :: \begin{bmatrix} \ell_0\text{: while } x = 0 \text{ do} \\ \quad \ell_1\text{: } y := y + 1 \\ \ell_2\text{:} \end{bmatrix} \quad || \quad P_2 :: \begin{bmatrix} m_0\text{: } x := 1 \\ m_1\text{:} \end{bmatrix}$$

To prove φ: $\Box$ $(y \geq 0)$,

we introduce two abstract variables:

X : **boolean**; Y : {neg,zero,pos}

Abstraction function:

α : {X = (x=1); Y = (if (y<0) then neg
else if (y=0) then zero else pos)}

Abstract-ANY

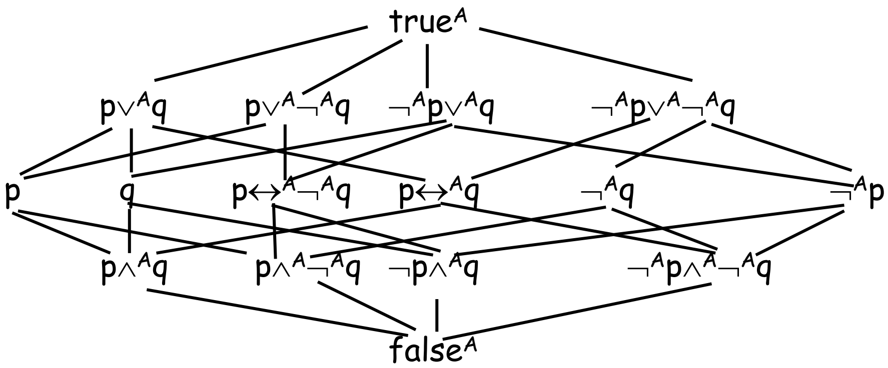
X: **boolean** **initially** $X = 0$
Y: $\{neg, zero, pos\}$ **initially** $Y = zero$

$$P_1 :: \left[\begin{array}{l} \ell_0 : \ \textbf{while } X = 0 \textbf{ do} \\ \quad [\ell_1 : Y := \left(\begin{array}{ll} \textbf{if} & Y = neg \\ \textbf{then} & \{neg, zero\} \\ \textbf{else} & pos \end{array} \right)] \\ \ell_2 : \end{array} \right] \quad \| \quad P_2 :: \left[\begin{array}{ll} m_0 : & X := 1 \\ m_1 : & \end{array} \right]$$

φ_A : $\square\, y \in \{zero, pos\}$

Assertion-based Abstraction

- Given basis $B = \{\varphi_1, \ldots, \varphi_n\}$
- Abstract domain Σ_A : boolean algebra over B, with implication as ordering relation



Assertion-based Abstraction

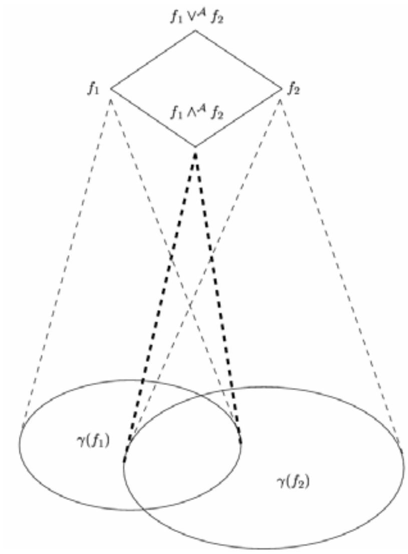
- Concretization function:

  for an assertion $a \in \Sigma_A$:
  $\gamma(a) = \{ s \in \Sigma \mid s \Vdash a \}$

- Abstraction function:

  for a set of concrete states S,

  $\alpha(S) = \bigwedge^A \{ s^A \in \Sigma_A \mid S \subseteq \gamma(s^A) \}$



Boolean Homomorphism

γ is a boolean homomorphism between the abstract and concrete assertion language:

For $s^A \in \Sigma_A$, $\gamma(s^A)$ is characterized by the concrete assertion obtained from s^A by

- replacing $\vee^A, \wedge^A, \neg^A$ by $\vee, \wedge, \neg$
- replacing the boolean variables in s^A by the corresponding formulas (from B)

Proof Rules as Abstractions

For assertions q, φ

$$\text{I1.} \quad P \Vdash \varphi \rightarrow q$$
$$\text{I2.} \quad P \Vdash \Theta \rightarrow \varphi$$
$$\text{I3.} \quad P \Vdash \{\varphi\}\ \mathcal{T}\ \{\varphi\}$$
$$\overline{\quad P \vDash \Box q \quad}$$

INV

Basis $B=\{b_\varphi\}$

Abstract System S^A: $\Theta^A=b_\varphi$, $T^A=\{\tau^A: b_\varphi \rightarrow^A b_\varphi'\}$,

Abstract property: $\Box\ b_\varphi$

NWAIT-Rule

Rule NWAIT (nested waiting-for)

For assertions p, $q_0, q_1, \ldots, q_m$ and $\varphi_0, \varphi_1, \ldots, \varphi_m$

$$\text{N1.} \quad p \rightarrow \bigvee_{j=0}^{m} \varphi_j$$
$$\text{N2.} \quad \varphi_i \rightarrow q_i \qquad \text{for } i = 0, 1, \ldots, m$$
$$\text{N3.} \quad \{\varphi_i\}\mathcal{T}\left\{\bigvee_{j \le i} \varphi_j\right\} \qquad \text{for } i = 0, 1, \ldots, m$$
$$\overline{\quad p \Rightarrow q_m\ \mathcal{W}\ q_{m-1} \cdots q_1\ \mathcal{W}\ q_0 \quad}$$

Basis $B=\{b_{\varphi 0}, \ldots, b_{\varphi m}\}$, $\Theta^A=\text{true}^A$, τ^A: $\bigwedge^A_i \ b_{\varphi_i} \rightarrow^A \bigvee^A_{j \le i} b_{\varphi_j}'$

Abstract property: $\Box \bigvee^A_j b_{\varphi_j} \rightarrow^A \ b_{\varphi m}\ W \ldots W\ b_{\varphi 0}$

Computing Abstractions

Given: basis $B=\{\varphi_1, \ldots, \varphi_n\}$.

Abstract system:

- $V^A = \{b_1, \ldots, b_n\}$ (one variable for each assertion)
- $\Theta^A = \alpha^+(\Theta)$
- $T^A = \{\tau^A | \tau \in T$ and $\rho_{\tau^A} = \alpha^+(\rho_\tau)\ \}$

Abstract property:

Replace atomic assertions p in φ
that have positive polarity with $\alpha^-(p)$,
and those with negative polarity with $\alpha^+(p)$.

Computing Abstractions

- Abstraction functions for formulas:

$\alpha^+(p) = \bigwedge^A \{ a \in \Sigma^A \mid p \rightarrow \gamma(a)\}$ $\alpha^-(p) = \bigvee^A \{ a \in \Sigma^A \mid \gamma(a) \rightarrow p\}$

$\alpha^+(\varphi_1 \wedge \varphi_2) = \alpha^+(\varphi_1) \wedge^A \alpha^+(\varphi_2)$ $\alpha^-(\varphi_1 \wedge \varphi_2) = \alpha^-(\varphi_1) \wedge^A \alpha^-(\varphi_2)$

$\alpha^+(\varphi_1 \vee \varphi_2) = \alpha^+(\varphi_1) \vee^A \alpha^+(\varphi_2)$ $\alpha^-(\varphi_1 \vee \varphi_2) = \alpha^-(\varphi_1) \vee^A \alpha^-(\varphi_2)$

$\alpha^+(\neg\varphi) = \neg^A\alpha^-(\varphi)$ $\alpha^-(\neg\varphi) = \neg^A\alpha^+(\varphi)$

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: $y_2 := 0$

Basis: guards of transitions

B = {b_1, b_2, b_3 } +
  control predicates

with

b_1: y1 = 0
b_2: y2 = 0
b_3: y1 ≤ y2

Example

α

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: $y_2 := 0$

local b_1, b_2, b_3 : boolean
  where b_1, b_2, b_3

loop forever do
- ℓ_0: noncritical
- ℓ_1: $(b_1, b_3) := (\mathit{false}, \mathit{false})$
- ℓ_2: await $(b_2 \vee b_3)$
- ℓ_3: critical
- ℓ_4: $(b_1, b_3) := (\mathit{true}, \mathit{true})$

||

loop forever do
- m_0: noncritical
- m_1: $(b_2, b_3) := (\mathit{false}, \mathit{true})$
- m_2: await $(b_1 \vee \neg b_3)$
- m_3: critical
- m_4: $(b_2, b_3) := (\mathit{true}, b_1)$

Example

This abstraction allows us to prove

- mutual exclusion
- bounded overtaking

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

local b_1, b_2, b_3 : **boolean** **where** b_1, b_2, b_3

loop forever do
- ℓ_0: **noncritical**
- ℓ_1: $(b_1, b_3) := (\mathit{false}, \mathit{false})$
- ℓ_2: **await** $(b_2 \vee b_3)$
- ℓ_3: **critical**
- ℓ_4: $(b_1, b_3) := (\mathit{true}, \mathit{true})$

||

loop forever do
- m_0: **noncritical**
- m_1: $(b_2, b_3) := (\mathit{false}, \mathit{true})$
- m_2: **await** $(b_1 \vee \neg b_3)$
- m_3: **critical**
- m_4: $(b_2, b_3) := (\mathit{true}, b_1)$

How To Determine the Basis?

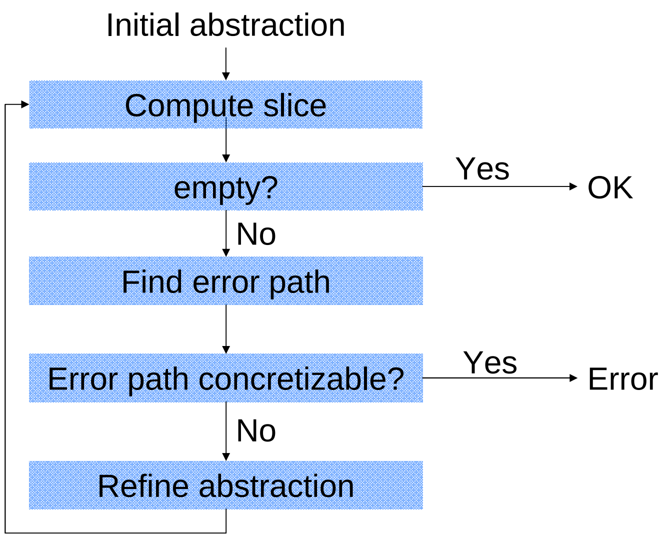
A good starting set:

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

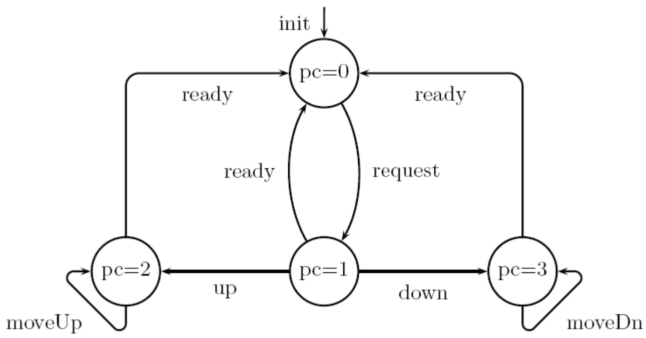
Analysis of counterexamples may lead to refinement of the abstraction by adding more assertions to the basis.

→ Automatic abstraction refinement in tools like SLAM (Microsoft), BLAST (UC Berkeley), SLAB (UdS).

SLAB (Slicing Abstractions)

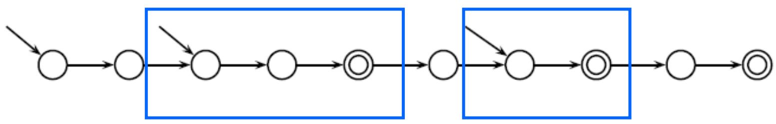


Example

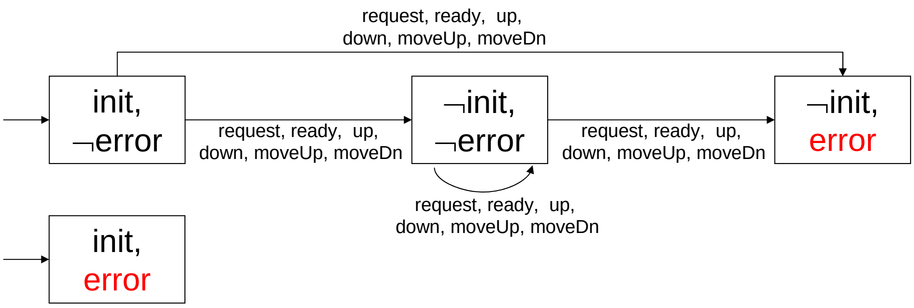


init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req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

Initial Abstraction - Intuition



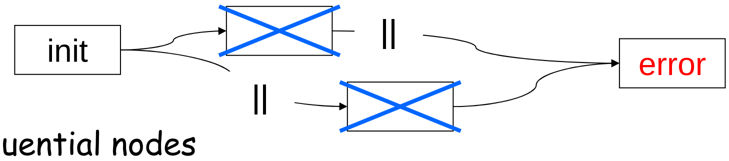
Initial Abstraction



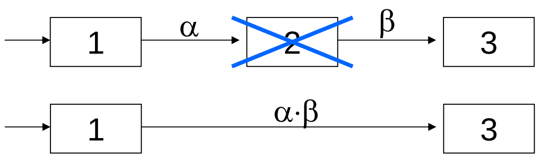
Slicing: Eliminating Nodes

- Inconsistent nodes

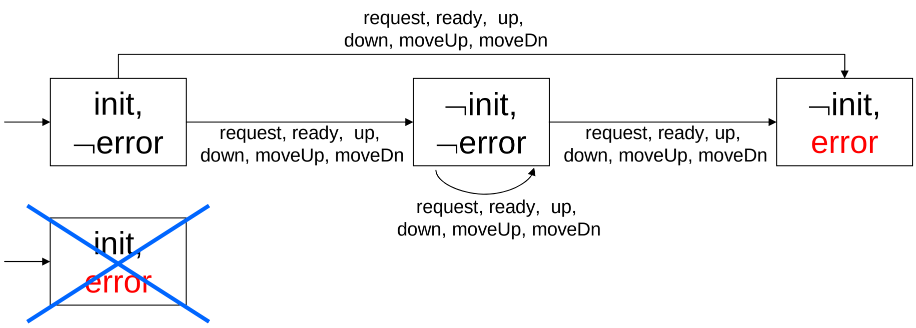
- Unreachable nodes



- Sequential nodes



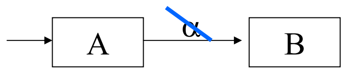
Slicing



$init$	$pc=0 \wedge current \leq Max \wedge input \leq Max$
$error$	$current > Max$
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	$pc=3 \wedge req < current \wedge pc'=3 \wedge current'=current-1 \wedge req'=req$

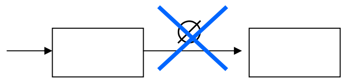
Slicing: Eliminating transitions

- Inconsistent transitions

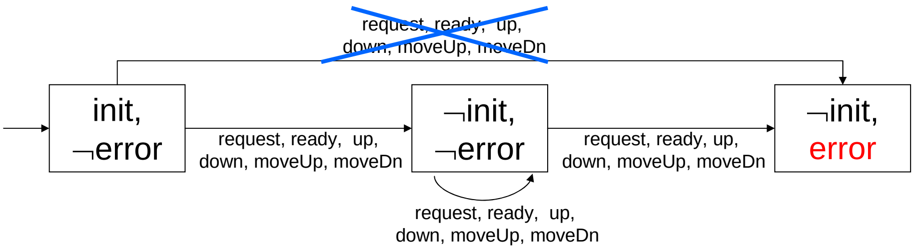


A(V) ∧ α(V,V') ∧ B(V') <u>unsatisfiable</u>

- Empty Edges

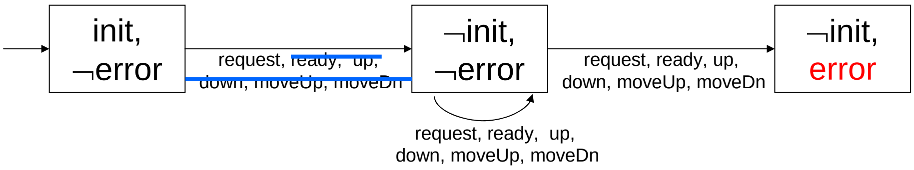


Slicing



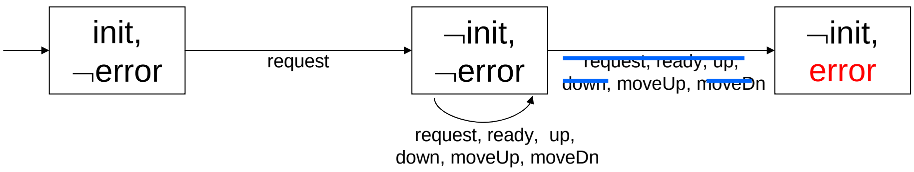
init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
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



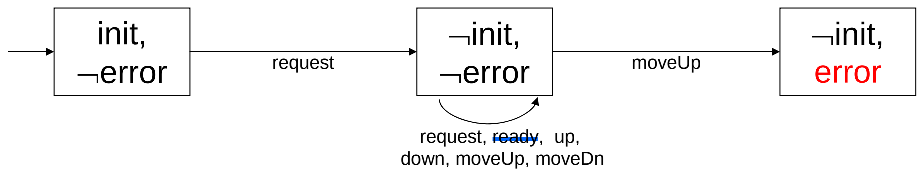
init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req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

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

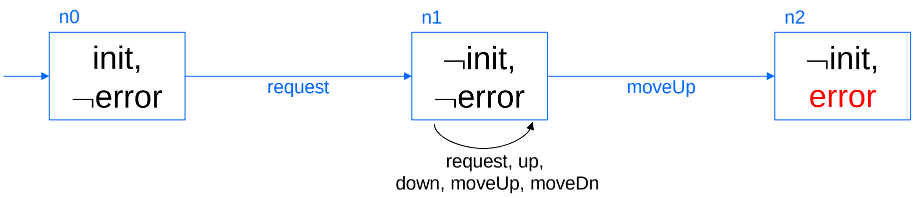


init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req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

Error Path Analysis

1. Error Path realizable?
2. If yes: System incorrect
3. If no: Node split
   - Find minimal error path
   - Determine node to split
   - Determine splitting predicate

Error Path Analysis

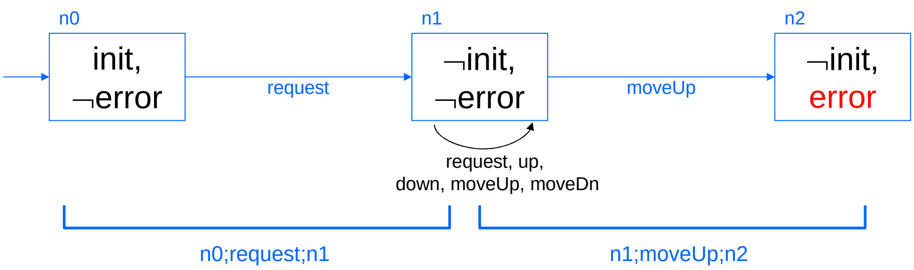


Error path realizable?

Φ(n0;request;n1;moveUp;n2) =
n0(V0) ∧ request(V0,V1) ∧ n1(V1) ∧ moveUp(V1,V2) ∧ n1(V2)

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

Error Path Analysis

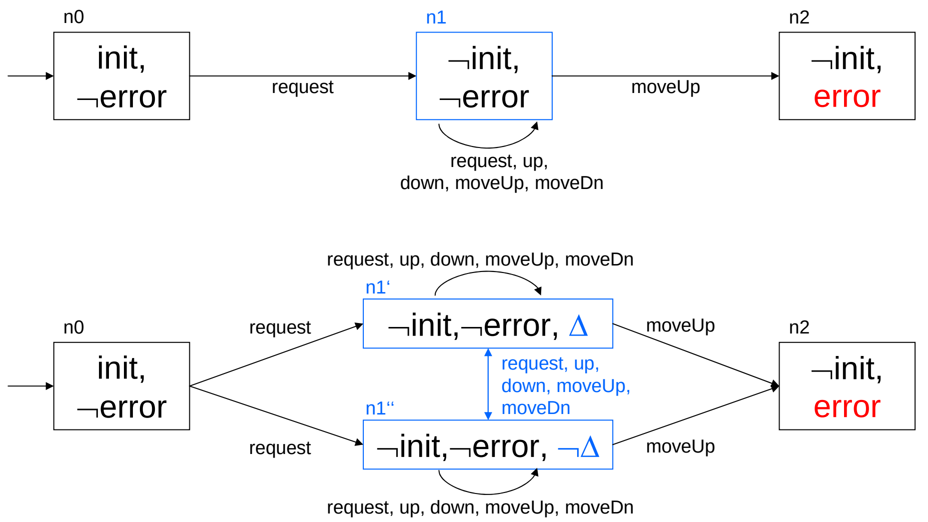


Error path minimal?

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

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

Node Split



Interpolation

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

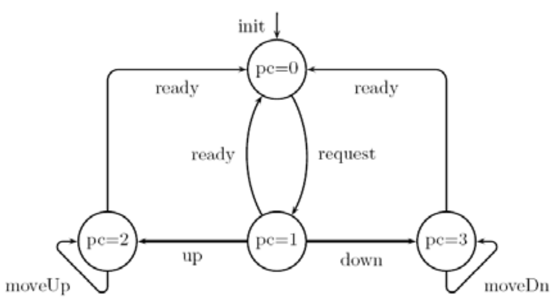
Φ(n1;moveUp;n2) = n1(V^1) $\wedge$ moveUp(V^1,V^2) $\wedge$ n1(V^2) **satisfiable**

Φ(n0;request;n1;moveUp;n2) = Φ(n0;request;n1) $\wedge$ Φ(n1;moveUp;n2) **unsatisfiable**

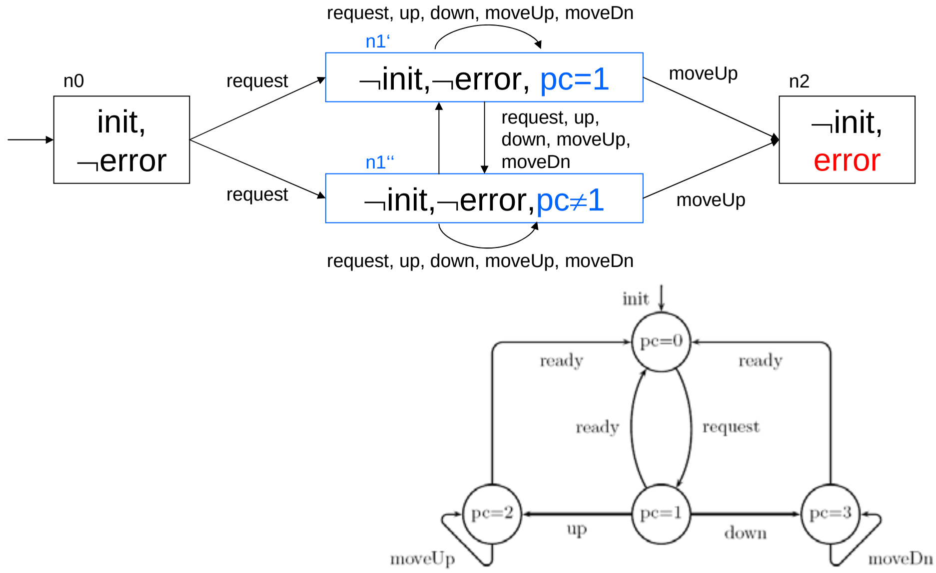
$\Rightarrow$ There exists a Craig interpolant Δ^1, such that

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

Δ^1 = $pc^1=1$

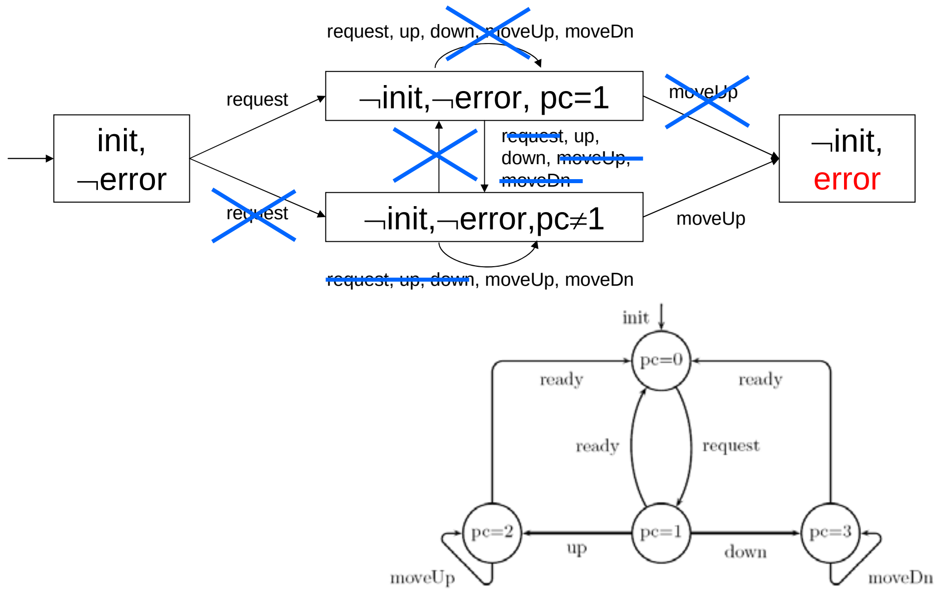


Splitting



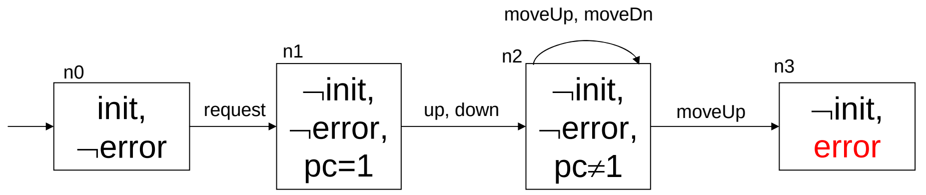
Bernd Finkbeiner

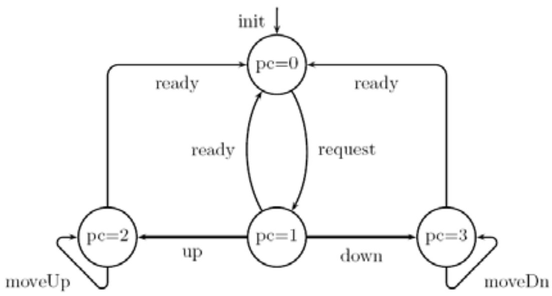
Slicing



Bernd Finkbeiner

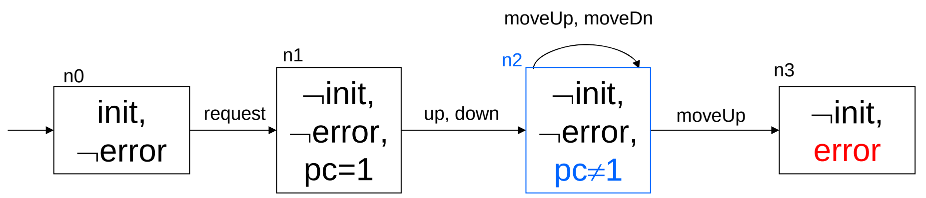
Error Path Analysis



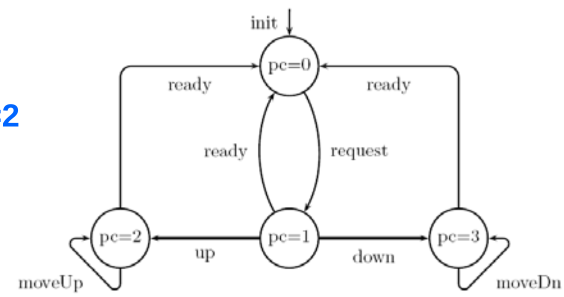


Bernd Finkbeiner

Error Path Analysis

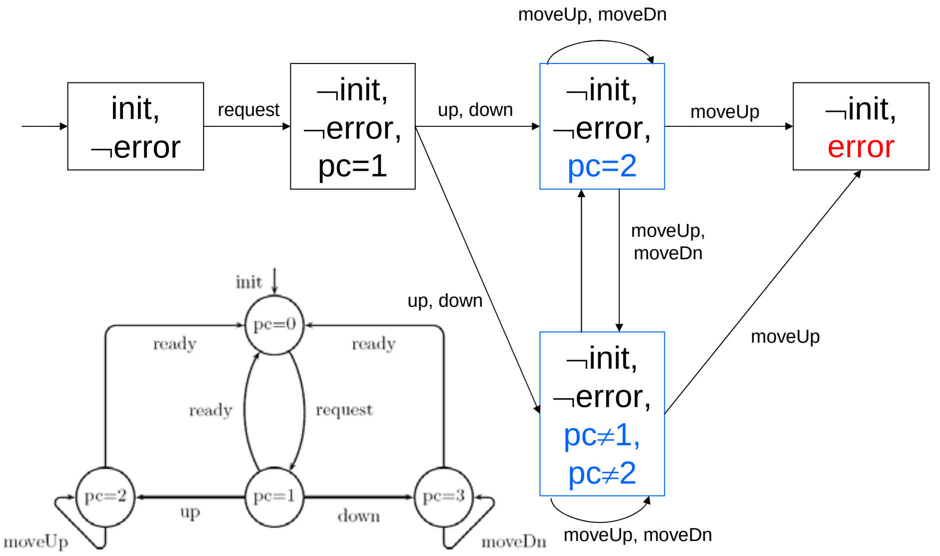


Split node n2 with pc=2



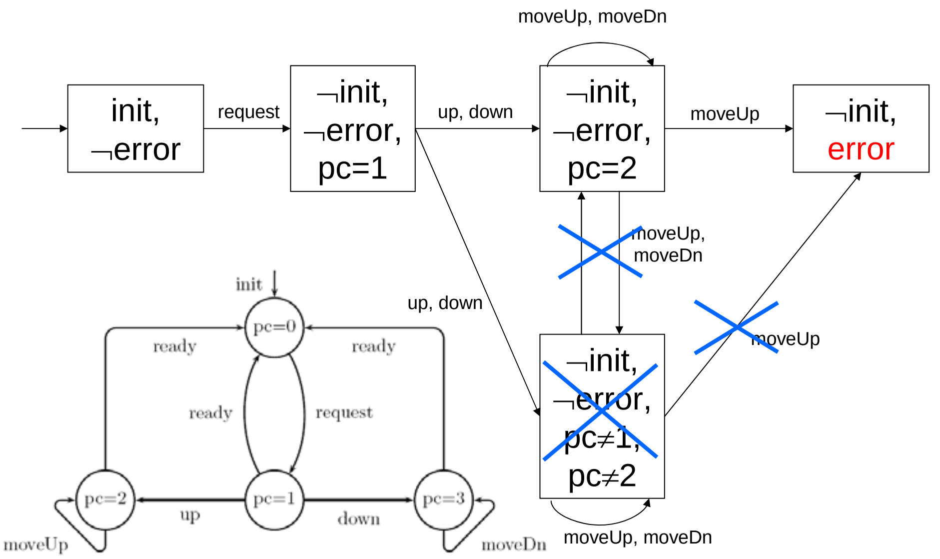
Bernd Finkbeiner

Splitting



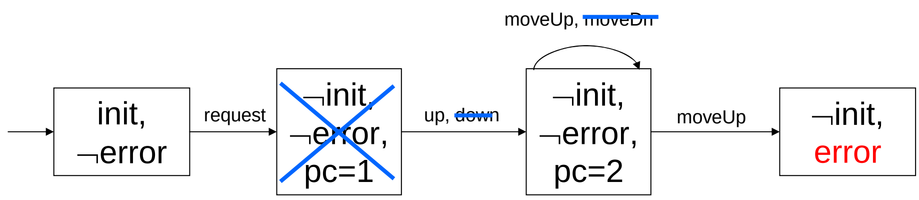
49

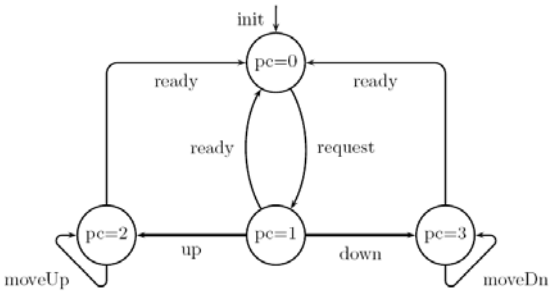
Slicing



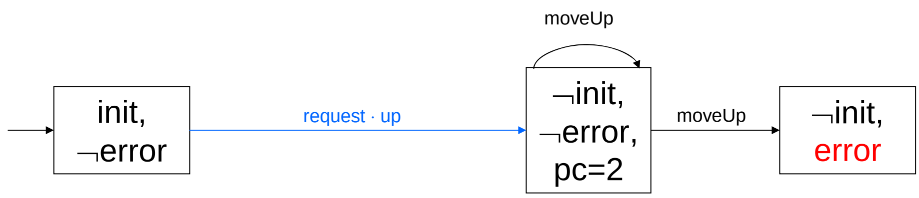
50

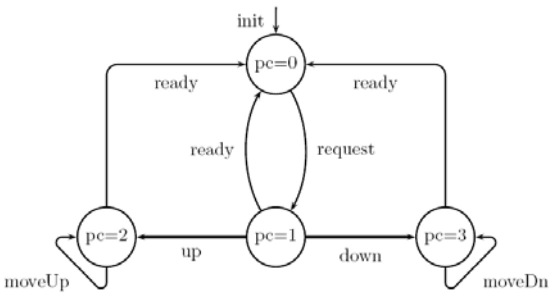
Slicing



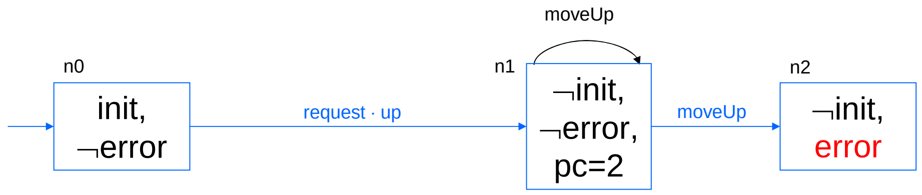


Slicing





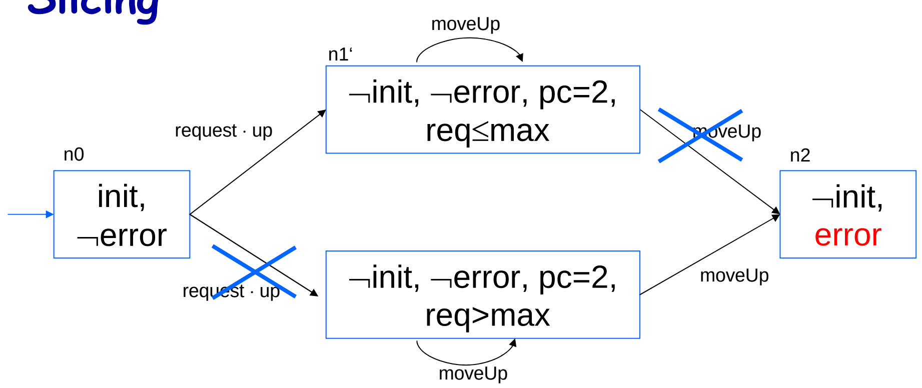
Error Path Analysis



Split node n1 with req≤max

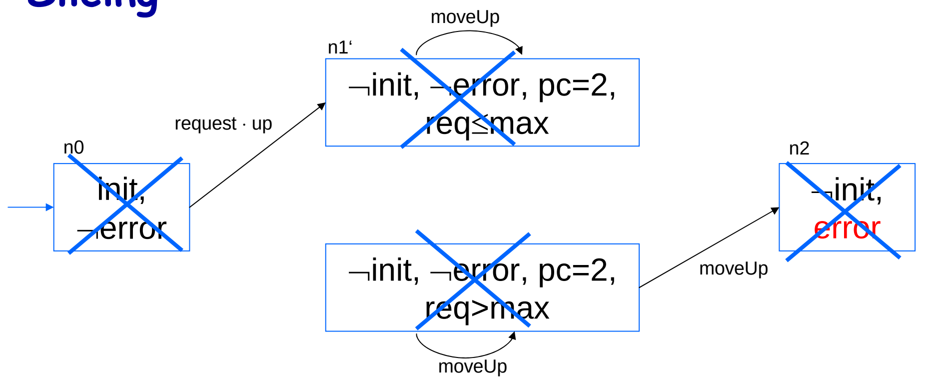
init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
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



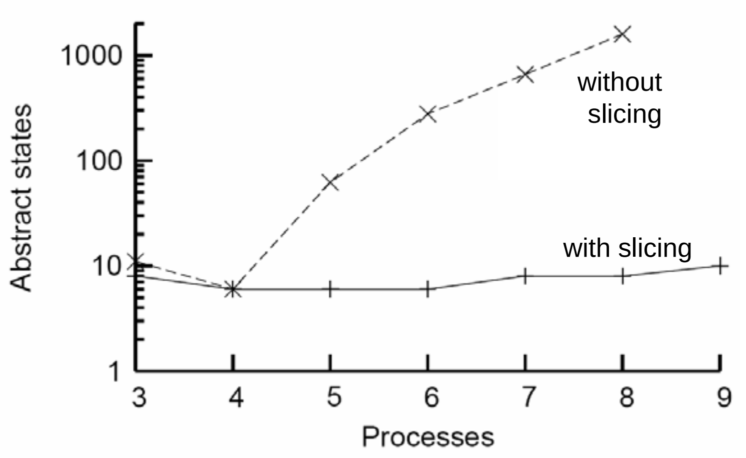
init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req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

Slicing

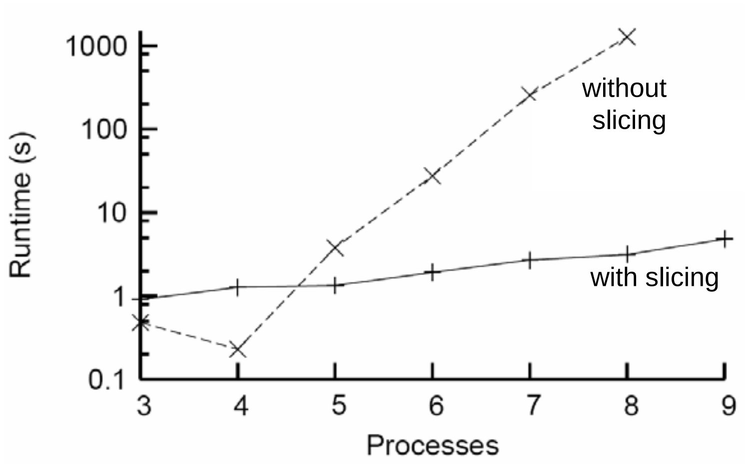


init	$pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$
error	$current{>}Max$
request	$pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$
ready	$pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$
up	$pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$
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



THE END

… see you in

Automata, Games, and Verification
(Sommersemester 2008)