Abstraction and Refinement for Timed Automata

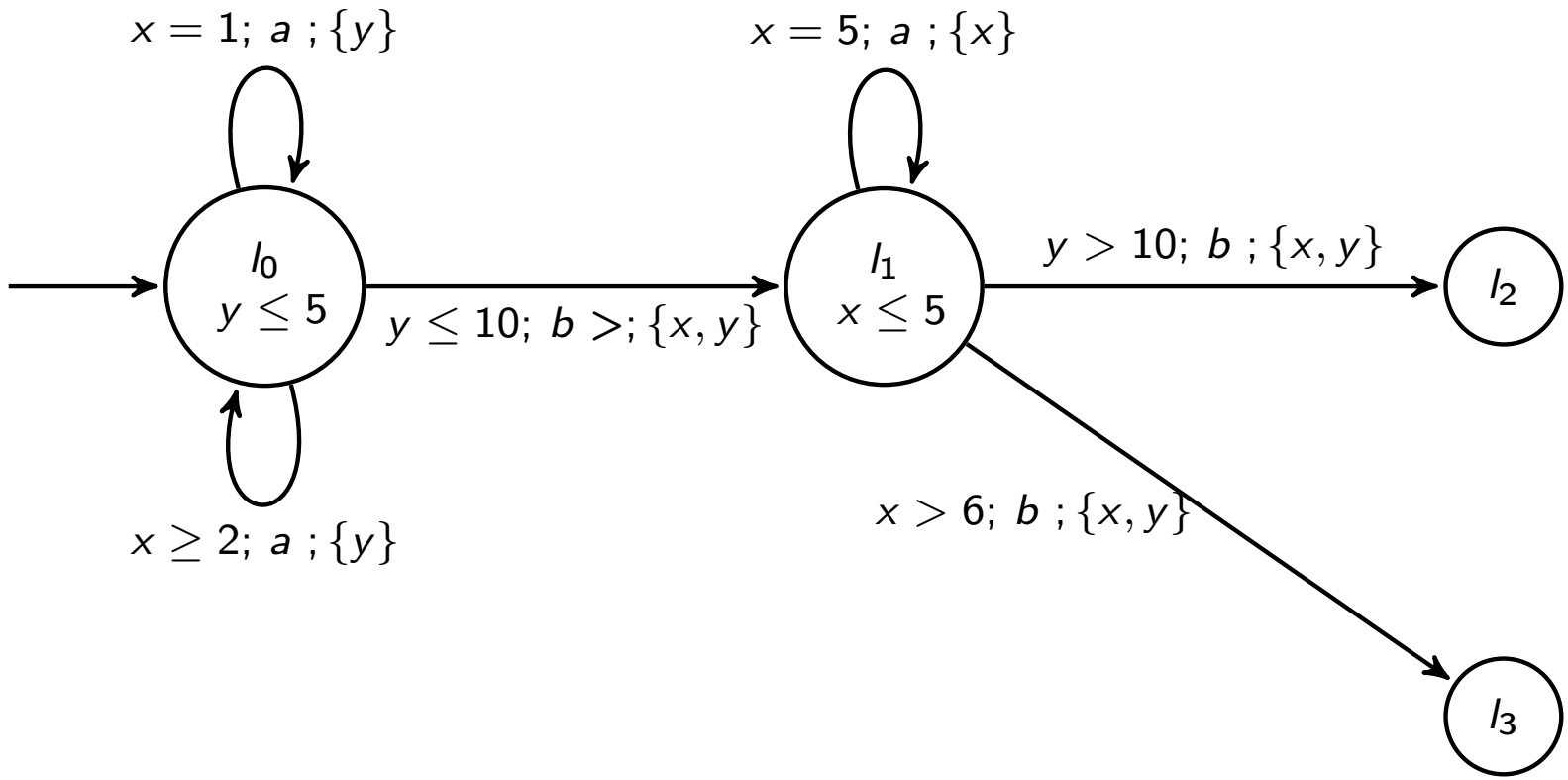
Victor Roussanaly

April 5, 2019

IRISA, INRIA, Univ. Rennes 1
SUMO, Supervised by Nicolas Markey and Ocan Sankur

Timed automata

Constraints $\mathcal{B}(C)$: a conjunction of $x \sim n$ or $x - y \sim n$ ($\sim \in \{<, \leq, =, \geq, >\}$).

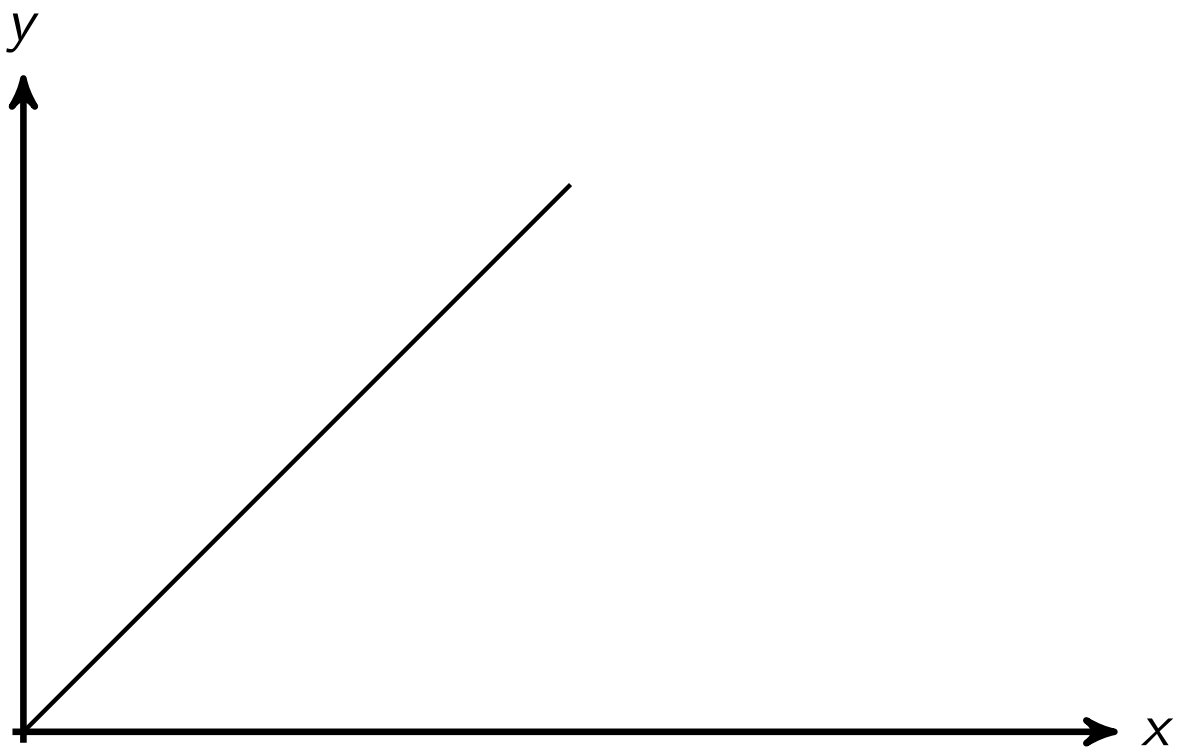


Problem: reachability of a state. PSPACE.

Zones

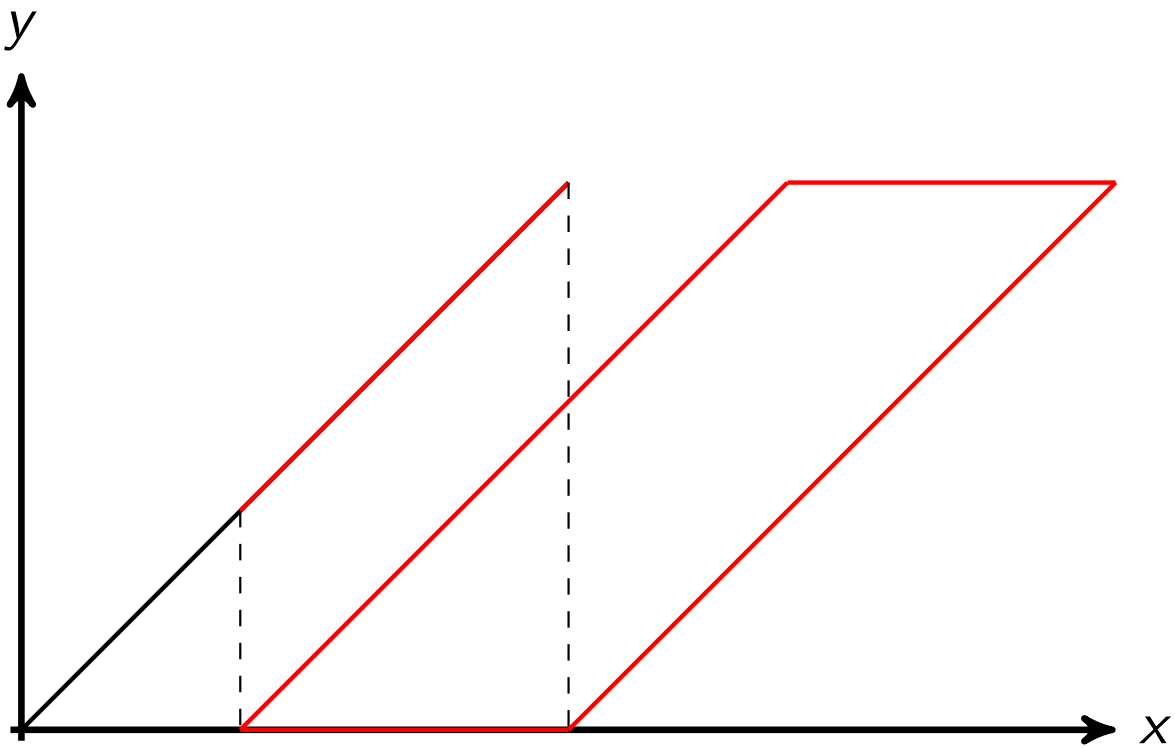
Zone: A set of clock valuations that respect a conjunction of clock constraint
Easy to encode (using DBM or minimal graphs).
We can use simple successor and predecessor functions ($Post_e$ and Pre_e).

Zones

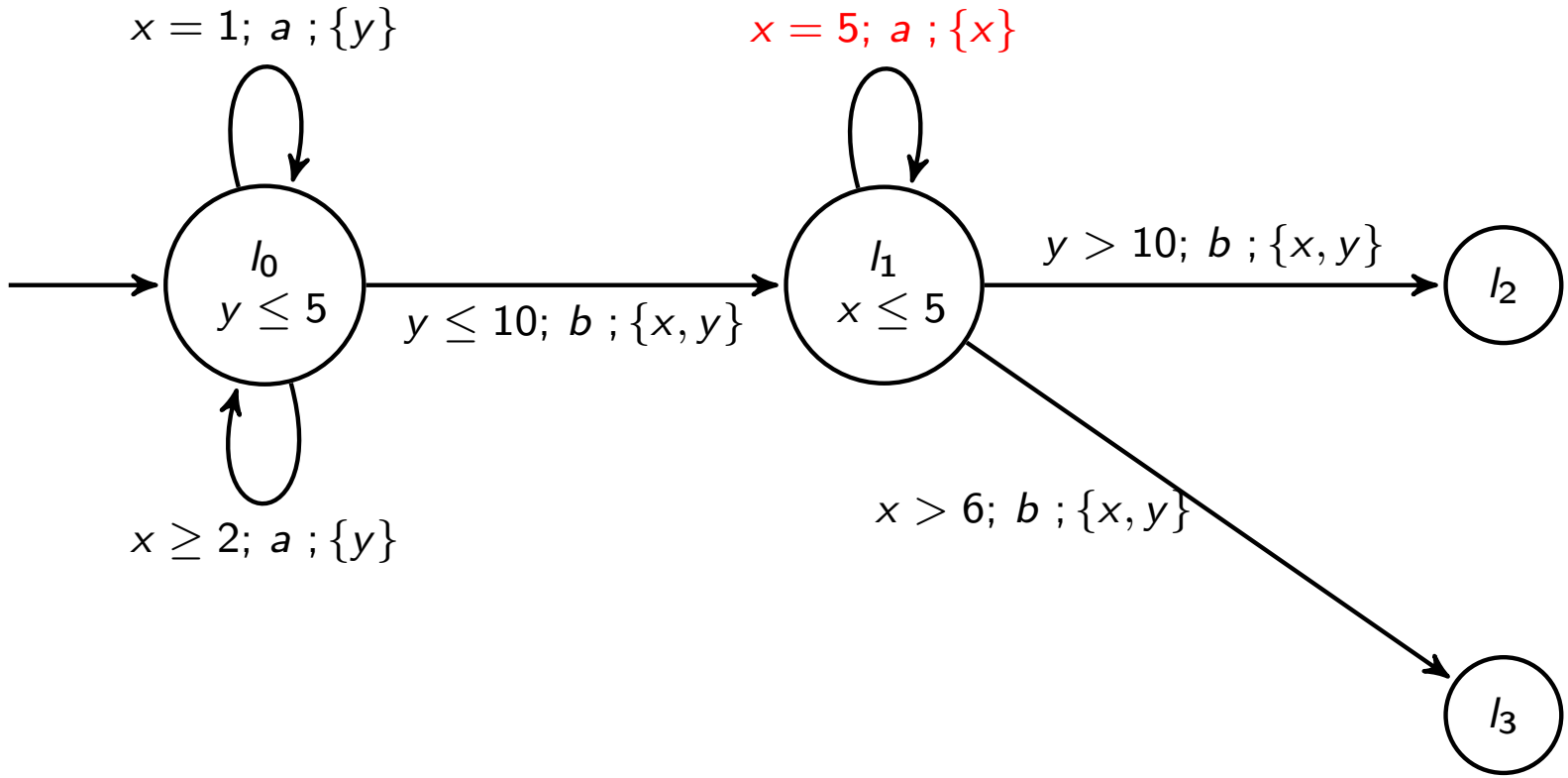


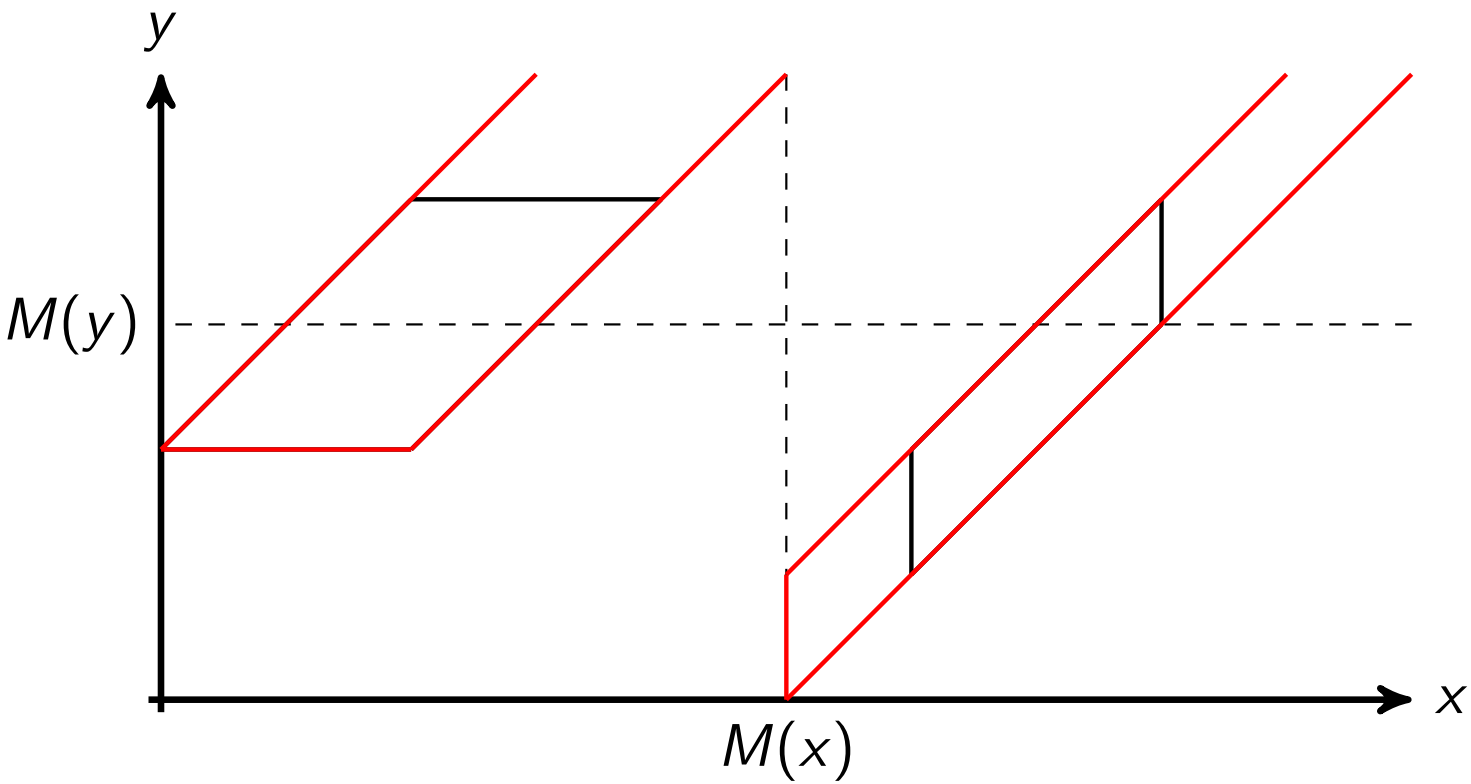
$e = (l, g, a, r, l')$. $Post_e(Z) = \pi_{r=0}(Z \cap g)^{\uparrow} \cap I(l')$

Zones



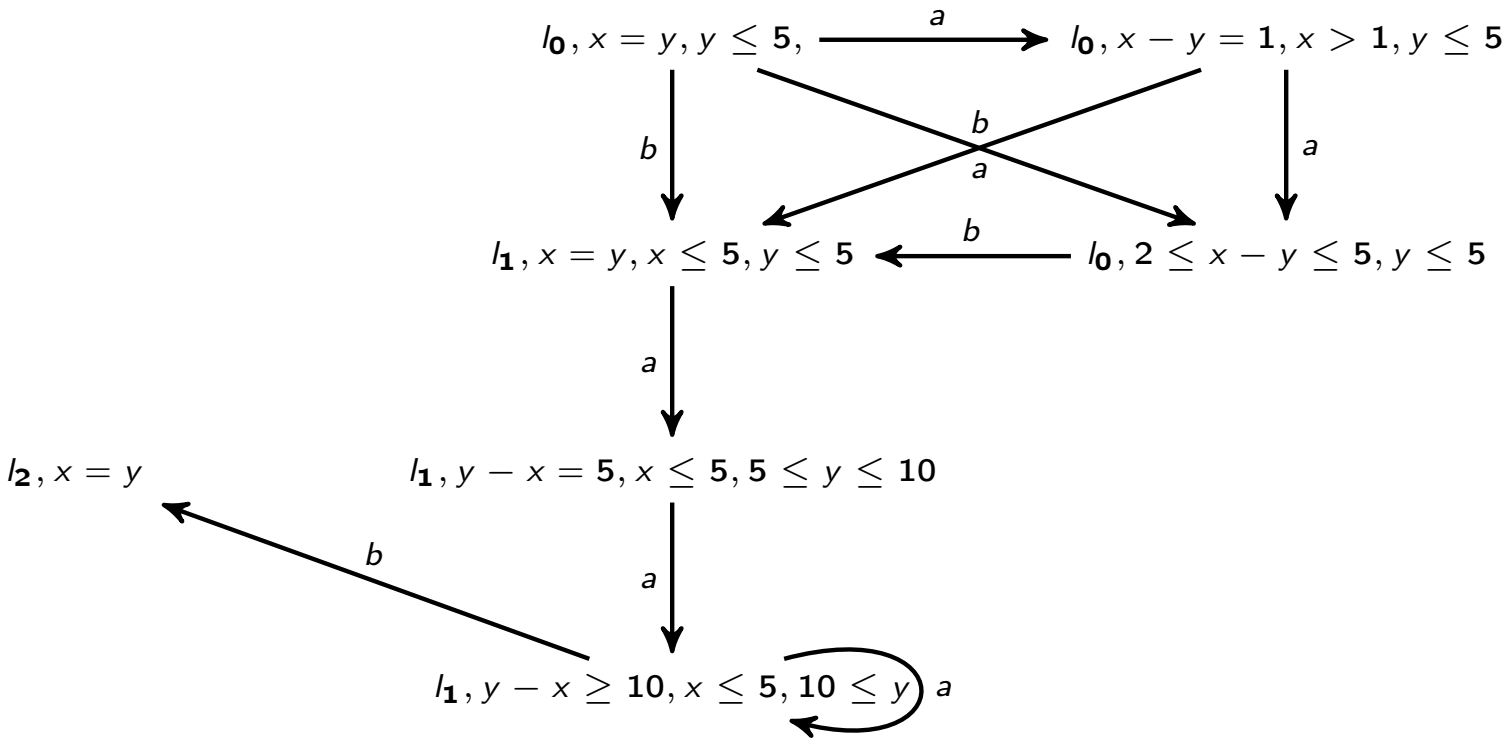
$e = (l, g, a, r, l').\ Post_e(Z) = \pi_{r=0}(Z \cap g)^{\uparrow} \cap I(l')$


$x = 1;\ a\ ;\{y\}$
l_0
$y \leq 5$
$x \geq 2;\ a\ ;\{y\}$
$y \leq 10;\ b\ ;\{x, y\}$
$x = 5;\ a\ ;\{x\}$
l_1
$x \leq 5$
$y > 10;\ b\ ;\{x, y\}$
l_2
$x > 6;\ b\ ;\{x, y\}$
l_3



Use of M-extrapolation (or LU-extrapolation).

Zone Graph


$l_0, x = y, y \leq 5,$
a
$l_0, x - y = 1, x > 1, y \leq 5$
b
b
a
a
b
$l_1, x = y, x \leq 5, y \leq 5$
$l_0, 2 \leq x - y \leq 5, y \leq 5$
a
$l_2, x = y$
$l_1, y - x = 5, x \leq 5, 5 \leq y \leq 10$
b
a
$l_1, y - x \geq 10, x \leq 5, 10 \leq y$
a

This is how UPPAAL works
We want coarser abstraction to reduce the space we explore.
Abstraction not necessarly exact. Need to refine the abstraction.

Counter-Example Guided Abstraction Refinement (CEGAR)

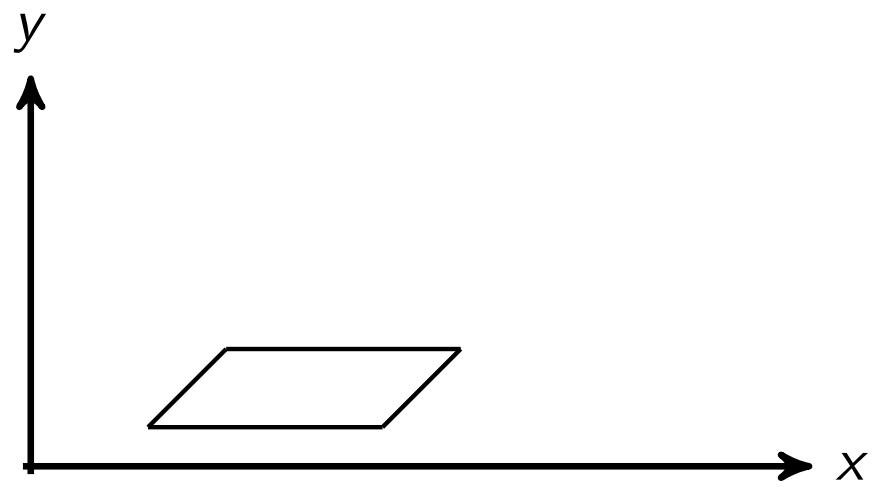
Start with a coarse abstraction.

- Build the model, check for the desired property.
- If the property is not verified, generate a counter-example.
- If the counter-example is not possible in the concrete model, use it to refine the abstraction.

CEGAR for timed automata:

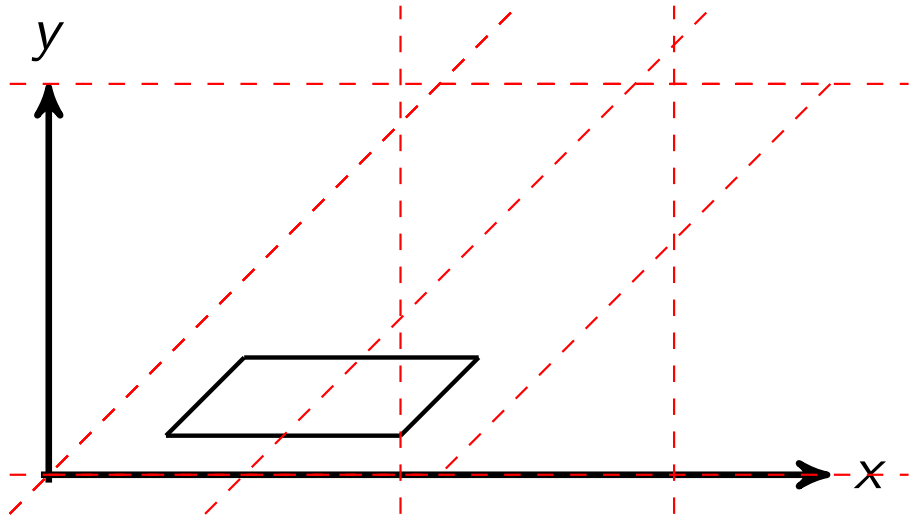
- reducing the number of clocks (Okano, Bordbar, Nagaoka, 2011)
- removing guards (Nagaoka, Okano, Kusumoto, 2010)
- using better LU functions (Herbreteau, Srivathsan, Walukiewicz, 2013)

Abstraction



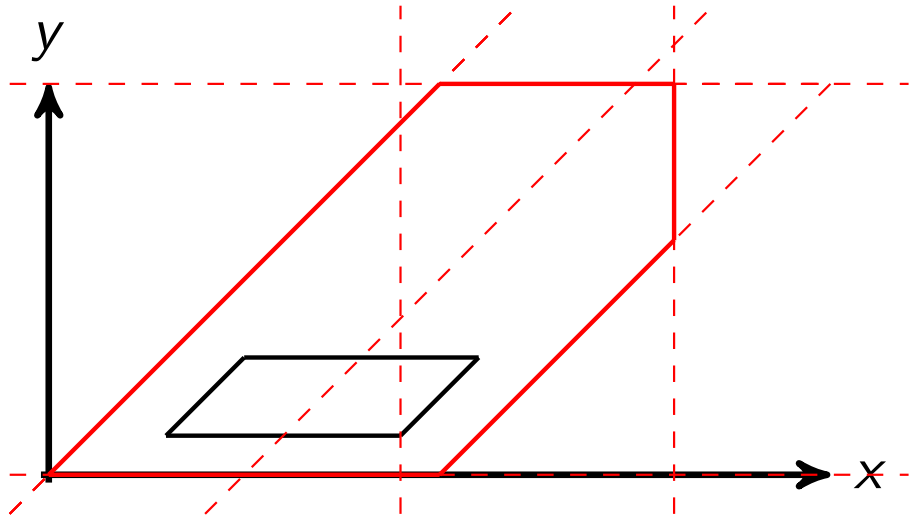
A zone is a conjunction of $x - y \sim n$ ($\sim \in \{<, \leq\}$, $n \in \mathbb{Z}, x, y \in C_0$)

Abstraction



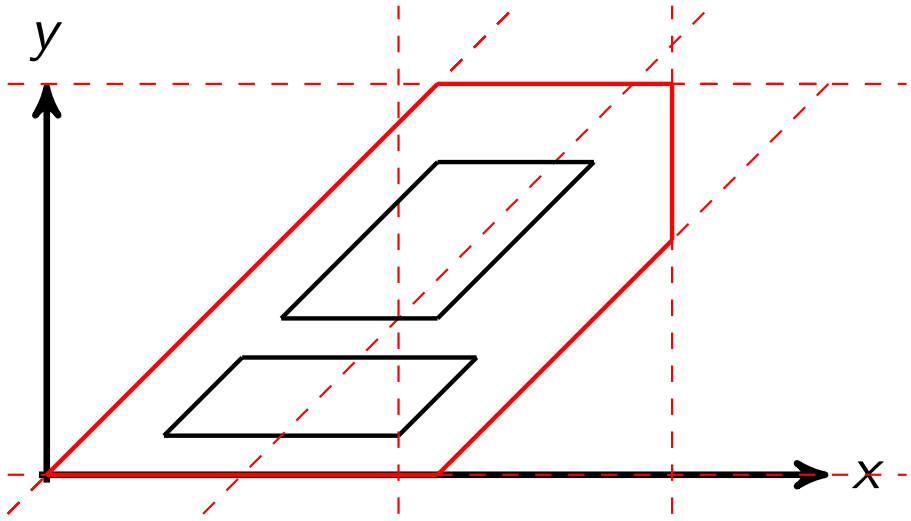
An abstracted zone is a conjunction of $x - y \sim n$
($\sim \in \{<, \leq\}$, $n \in T(x, y) \subseteq \mathbb{Z}, x, y \in C_0$)

Abstraction



An abstracted zone is a conjunction of $x - y \sim n$
($\sim \in \{<, \leq\}$, $n \in T(x, y) \subseteq \mathbb{Z}, x, y \in C_0$)

Abstraction



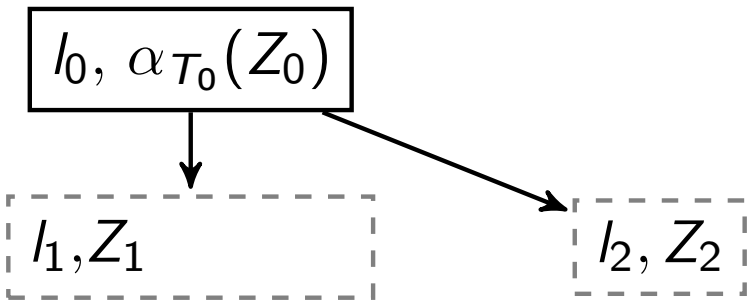
An abstracted zone is a conjunction of $x - y \sim n$
($\sim \in \{<, \leq\}$, $n \in T(x, y) \subseteq \mathbb{Z}, x, y \in C_0$)

Building an exploration tree

An exploration tree where nodes are (I, Z).

Building an exploration tree

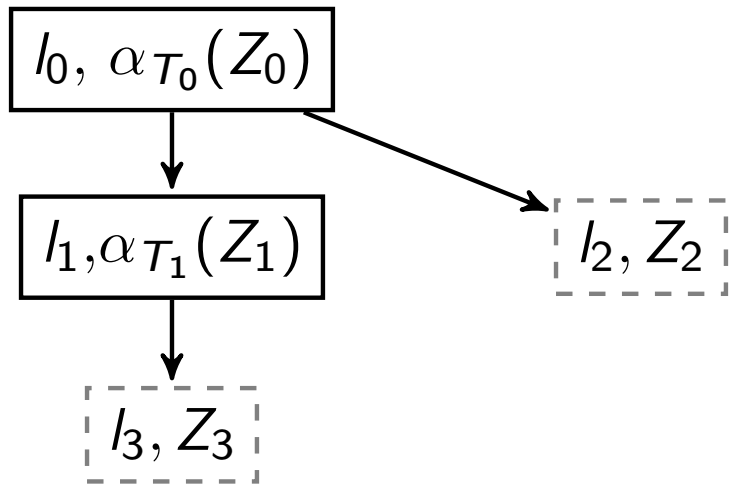
An exploration tree where nodes are (l, Z).



$Z_1 = Post_{e_1}(\alpha_{T_0}(Z_0)),\ Z_2 = Post_{e_2}(\alpha_{T_0}(Z_0)).$

Building an exploration tree

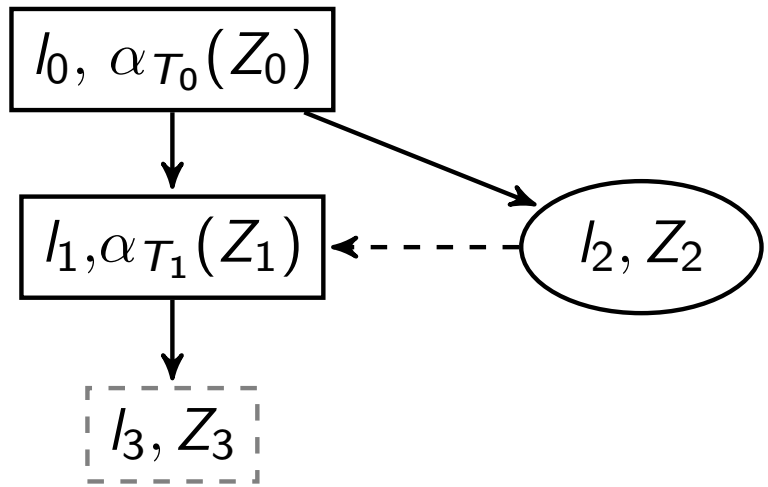
An exploration tree where nodes are (l, Z).



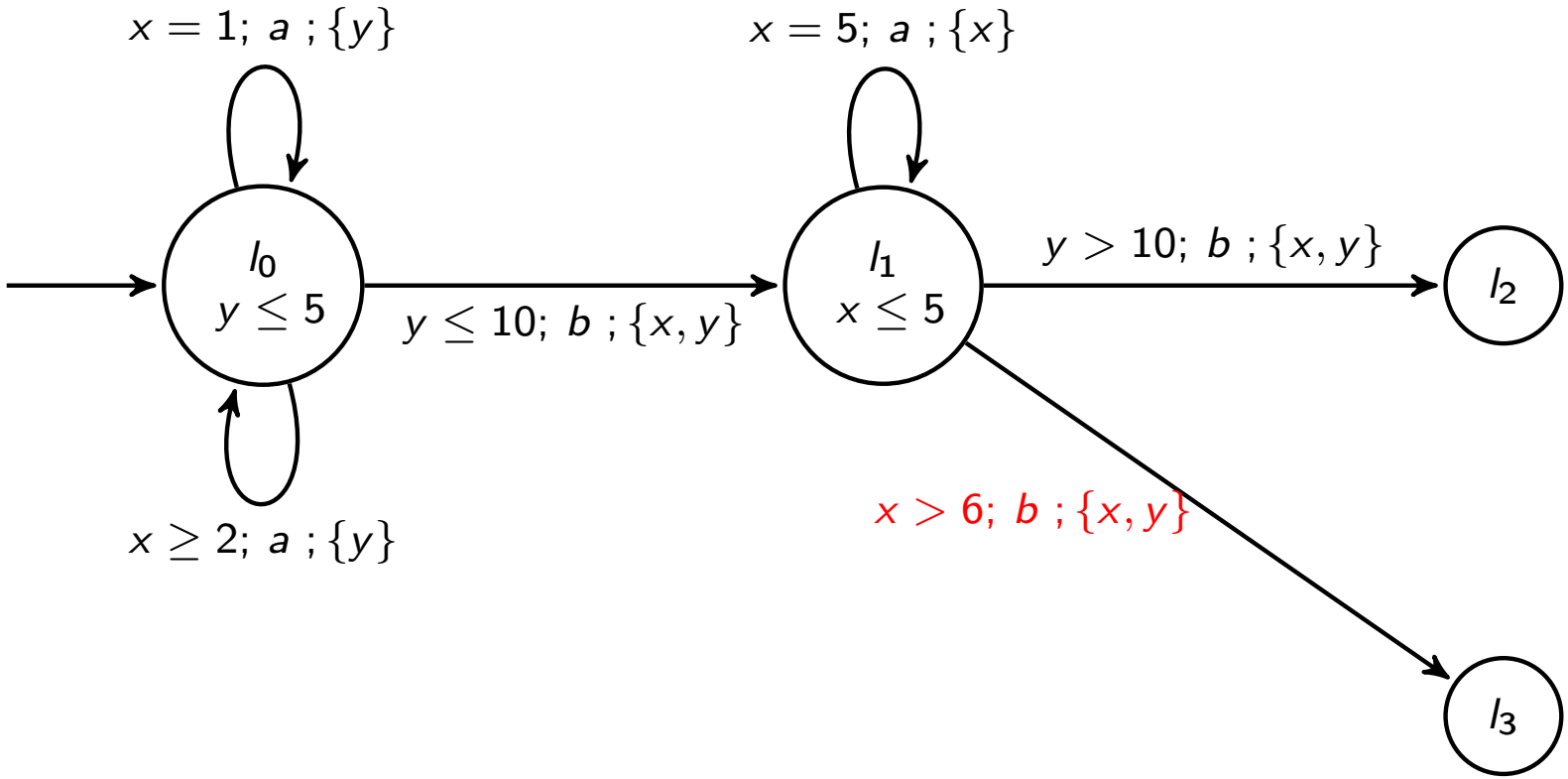
$Z_1 = Post_{e_1}(\alpha_{T_0}(Z_0))$, $Z_2 = Post_{e_2}(\alpha_{T_0}(Z_0))$.

Building an exploration tree

An exploration tree where nodes are (l, Z). Some nodes are covered.



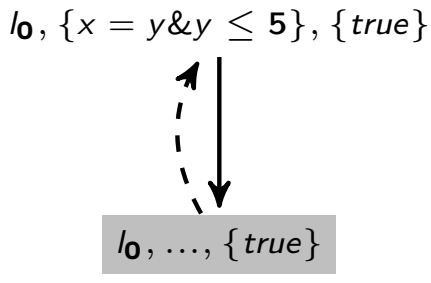
$Z_1 = Post_{e_1}(\alpha_{T_0}(Z_0))$, $Z_2 = Post_{e_2}(\alpha_{T_0}(Z_0))$.
Covering condition: $l_2 = l_1$ and $Z_2 \subseteq \alpha_{T_1}(Z_1)$


$x = 1;\ a\ ;\{y\}$
$x = 5;\ a\ ;\{x\}$
l_0
$y \leq 5$
$y \leq 10;\ b\ ;\{x, y\}$
l_1
$x \leq 5$
$y > 10;\ b\ ;\{x, y\}$
l_2
$x > 6;\ b\ ;\{x, y\}$
l_3
$x \geq 2;\ a\ ;\{y\}$

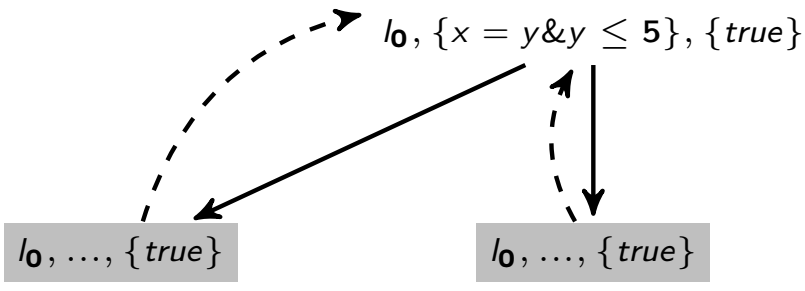
ASG for l_3

$l_{\mathbf{0}}, \{x = y \& y \leq \mathbf{5}\}, \{true\}$

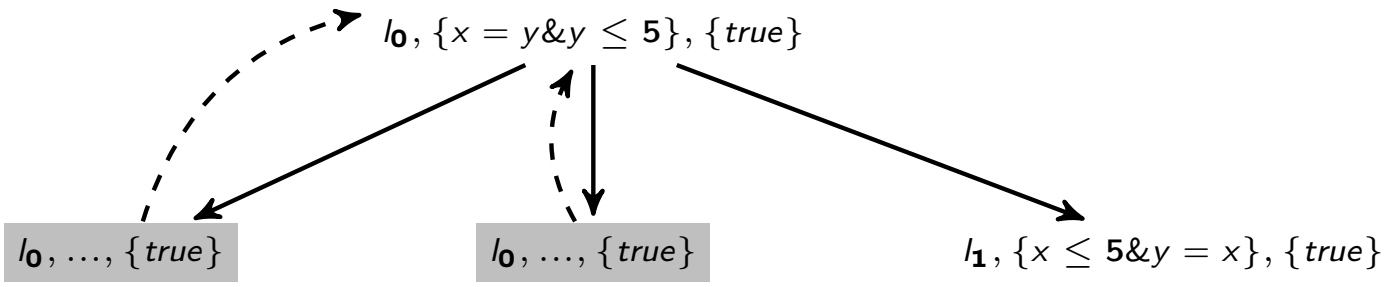
ASG for l_3



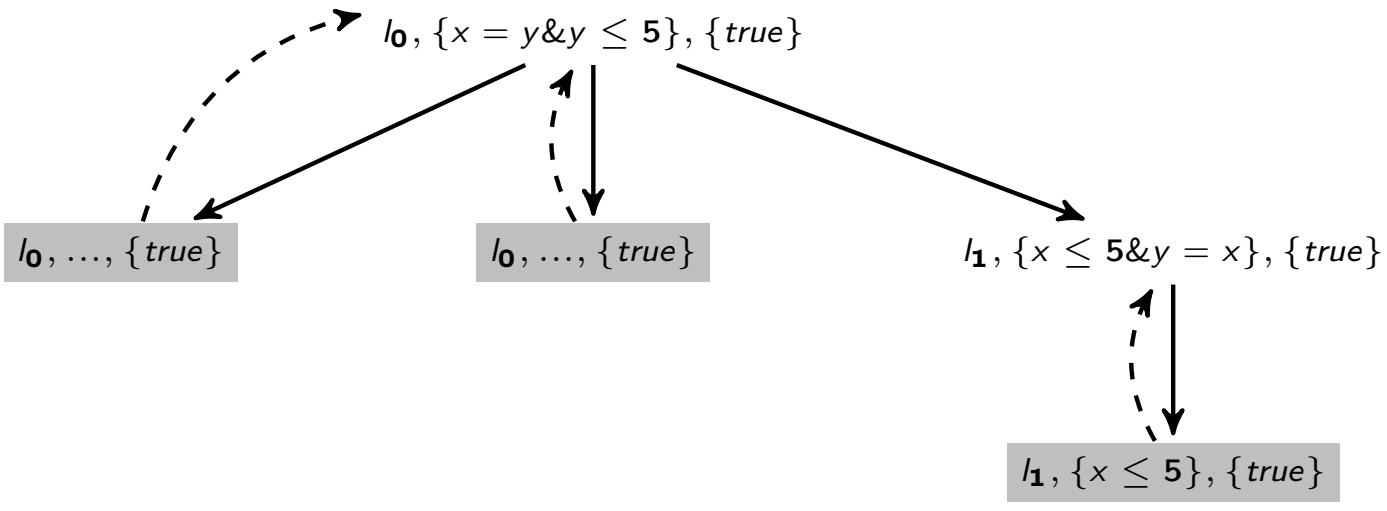
ASG for l_3



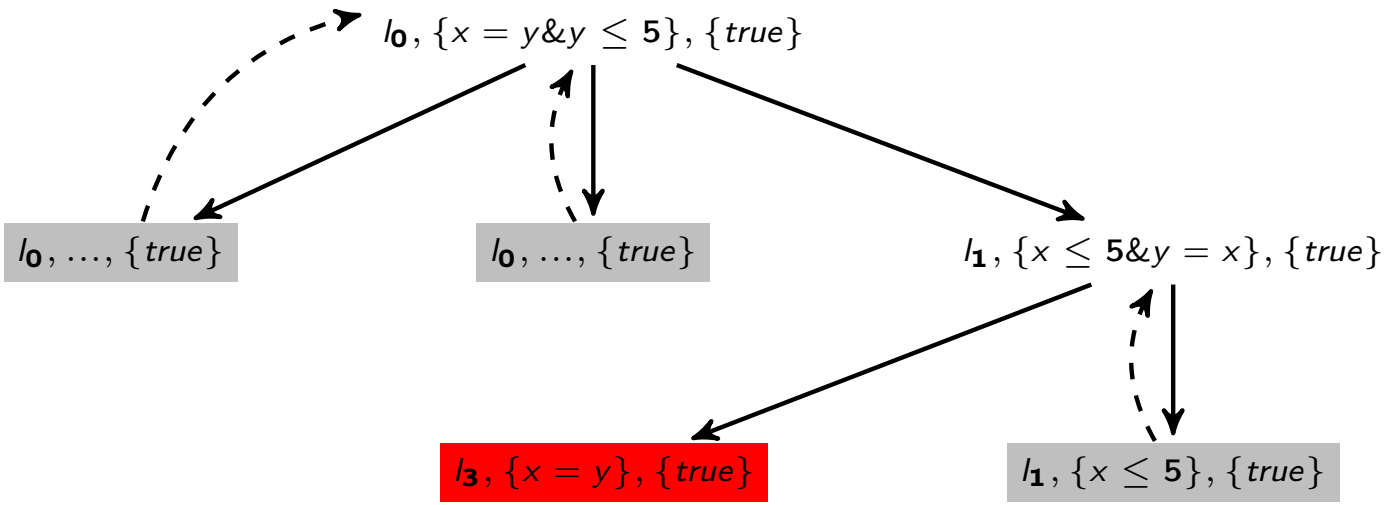
ASG for l_3



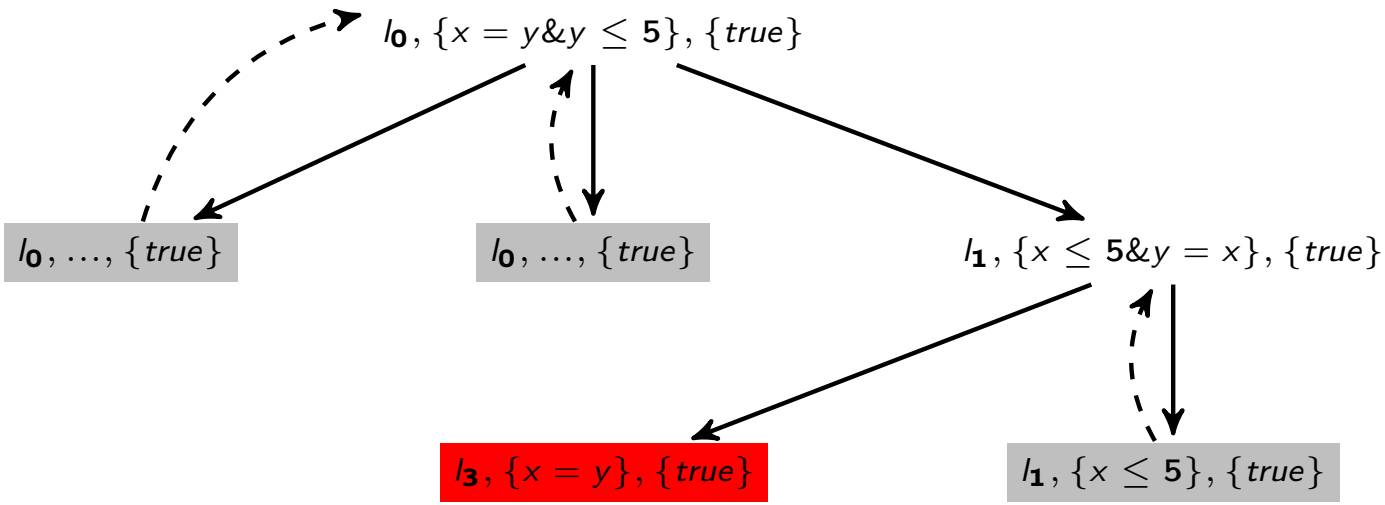
ASG for l_3


$l_0, \{x = y \& y \leq 5\}, \{true\}$
$l_0, \ldots, \{true\}$
$l_0, \ldots, \{true\}$
$l_1, \{x \leq 5 \& y = x\}, \{true\}$
$l_1, \{x \leq 5\}, \{true\}$

ASG for l_3

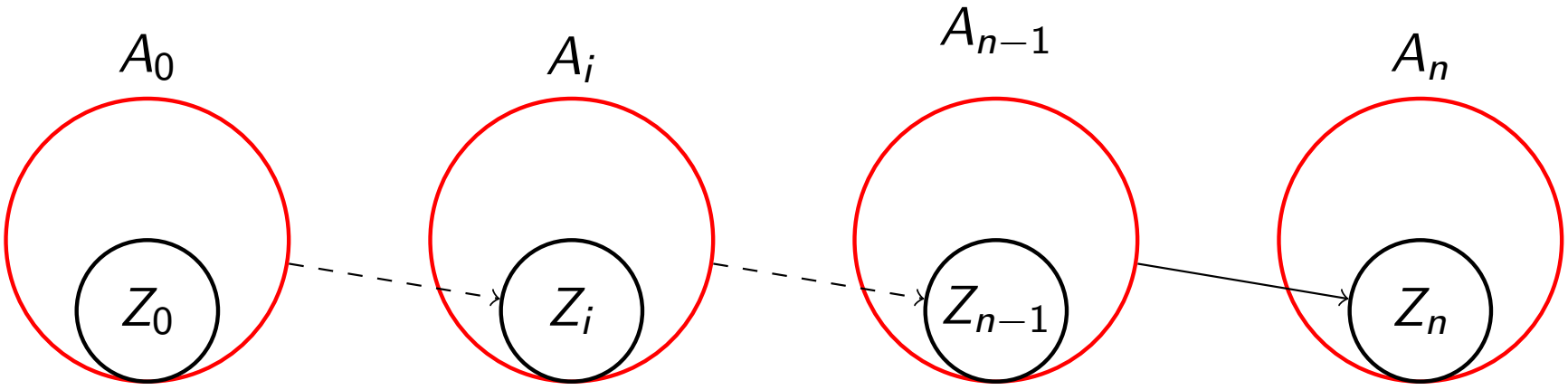

$l_0, \{x = y \& y \leq 5\}, \{true\}$
$l_0, \ldots, \{true\}$
$l_0, \ldots, \{true\}$
$l_1, \{x \leq 5 \& y = x\}, \{true\}$
$l_3, \{x = y\}, \{true\}$
$l_1, \{x \leq 5\}, \{true\}$

ASG for l_3

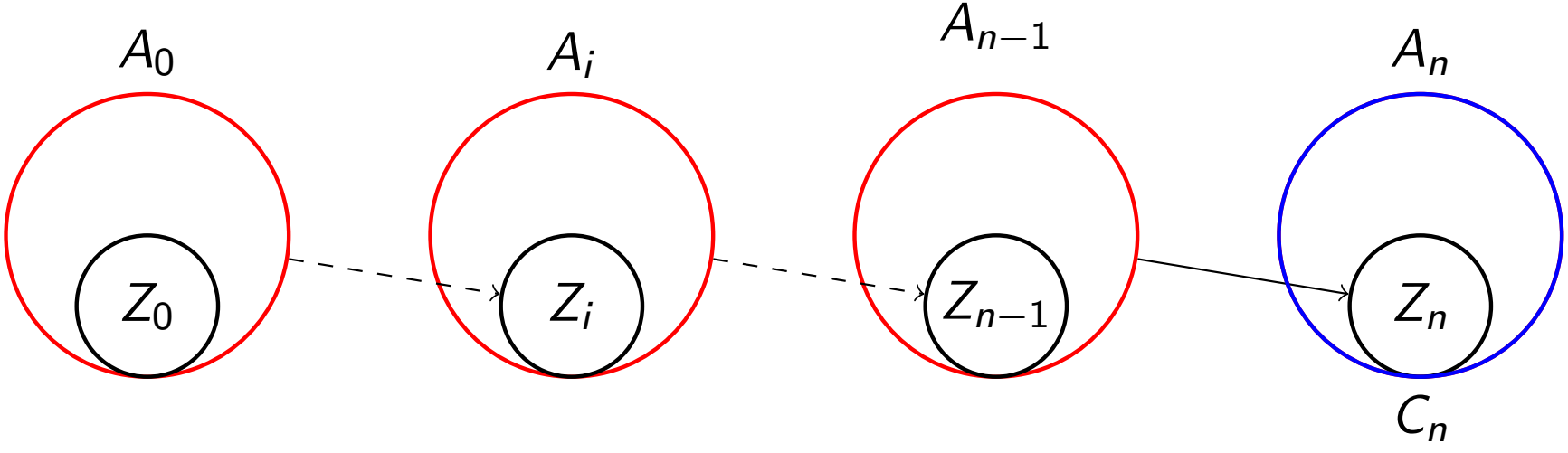


Refinement is needed

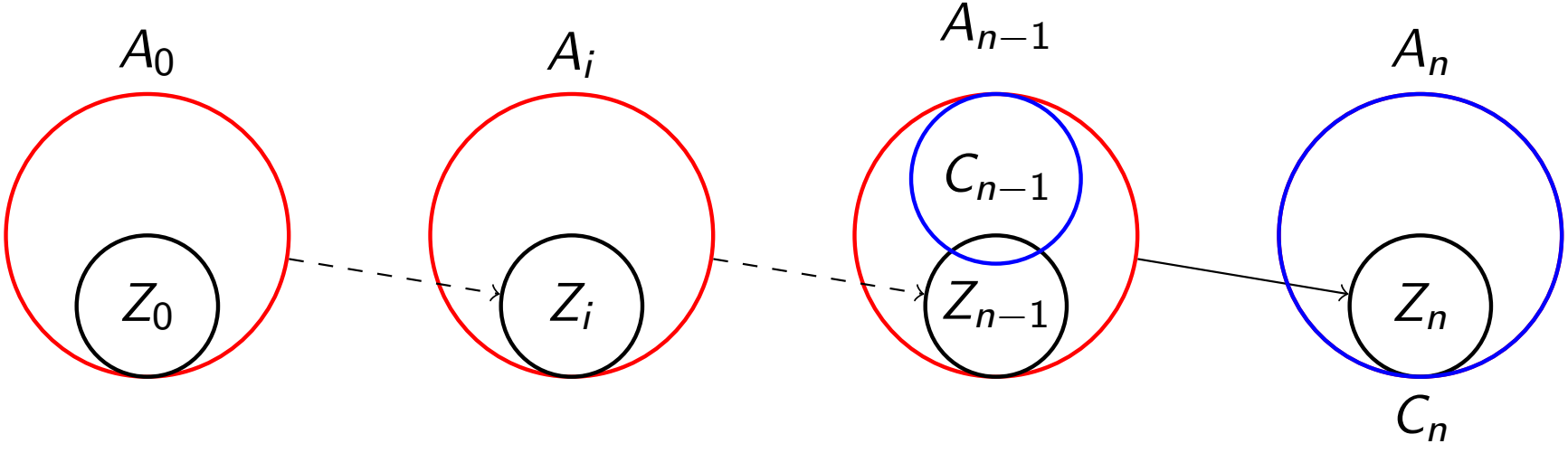
Detecting a spurious run



Detecting a spurious run

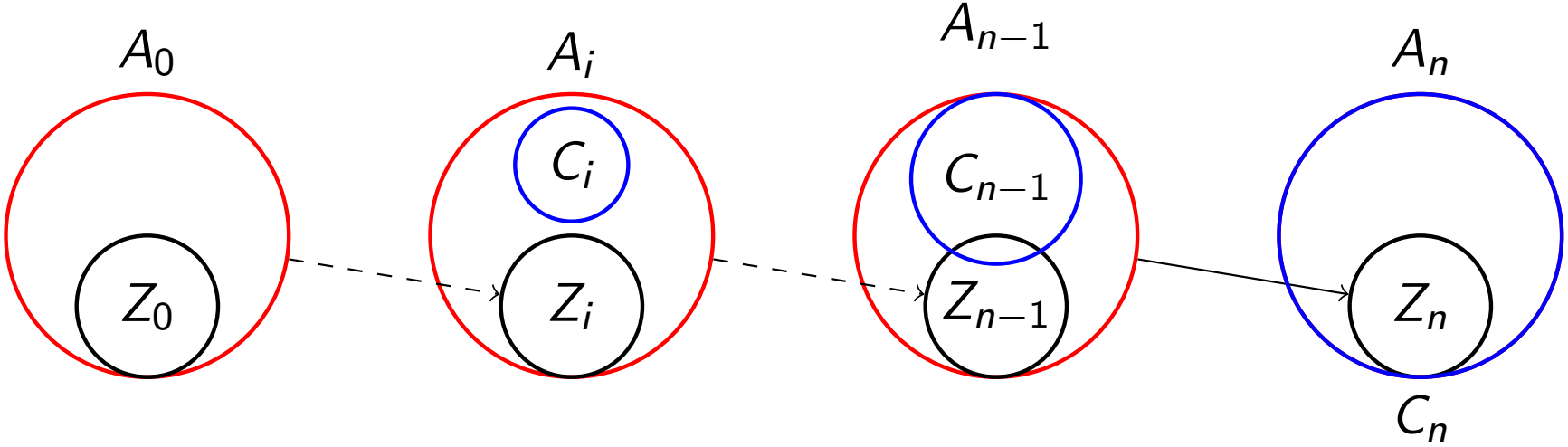


Detecting a spurious run



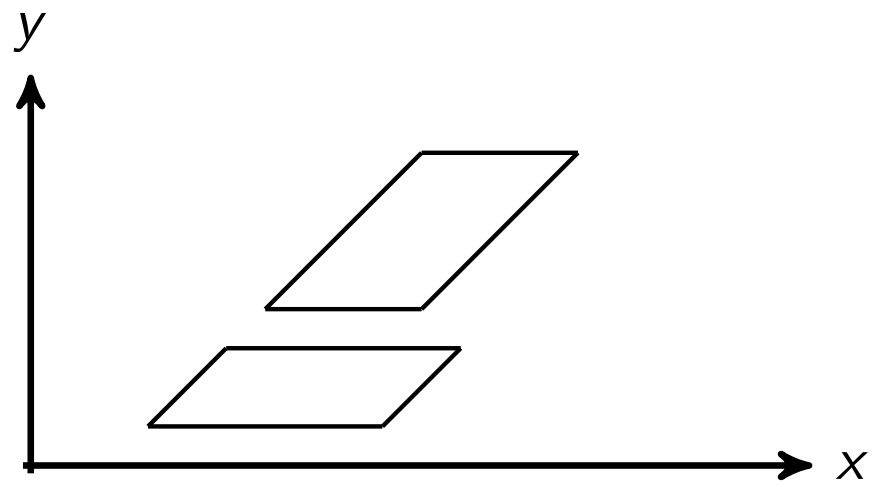
$C_i = Pre(C_{i+1}) \cap A_i.$

Detecting a spurious run



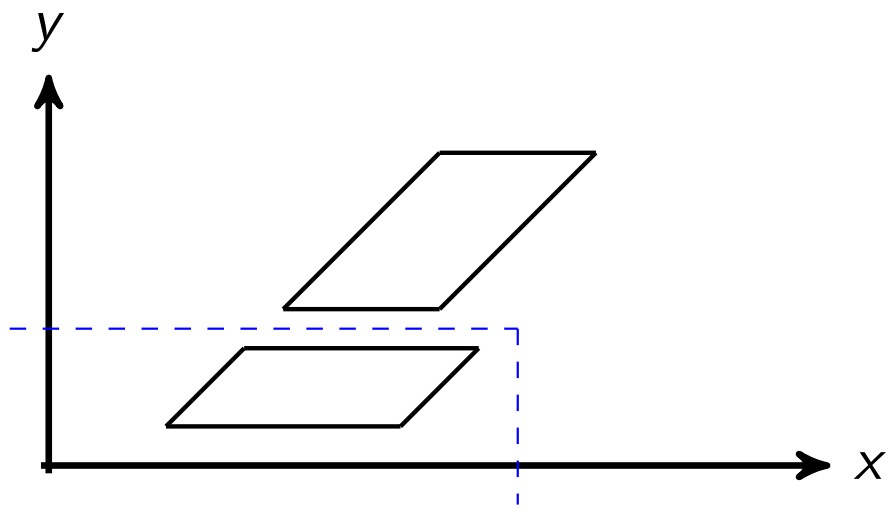
$C_i = Pre(C_{i+1}) \cap A_i$.
If $C_i \cap Z_i = \emptyset$, then the run is spurious.

Interpolant



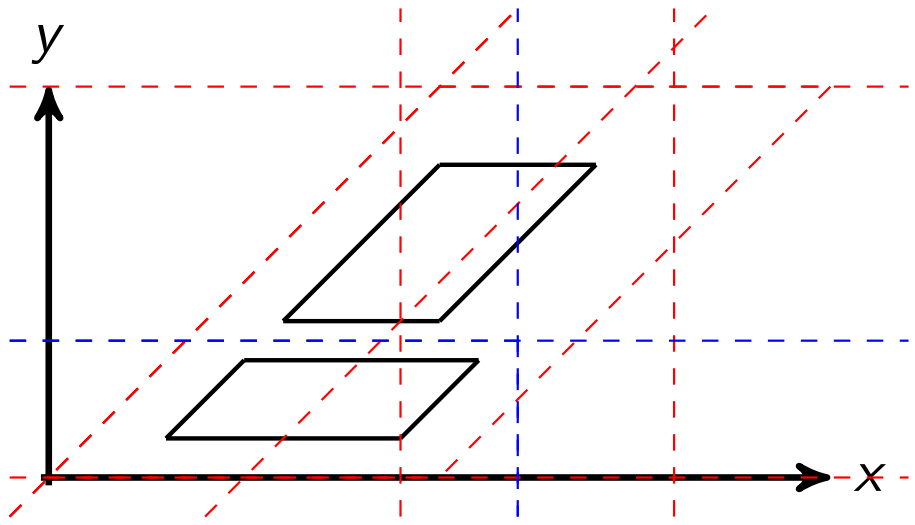
An interpolant I of (A, B), a zone s.t. $B \subseteq I$, $A \cap I = \emptyset$.

Interpolant



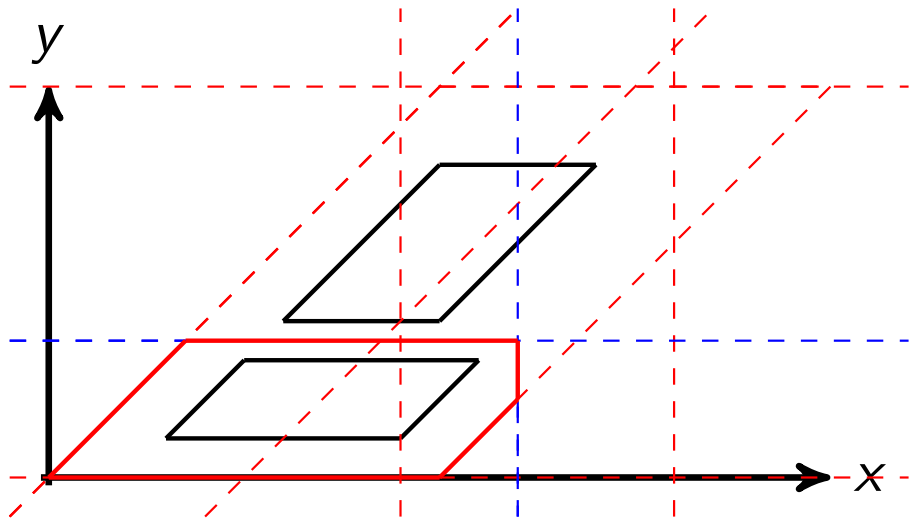
An interpolant I of (A, B), a zone s.t. $B \subseteq I$, $A \cap I = \emptyset$.

Interpolant



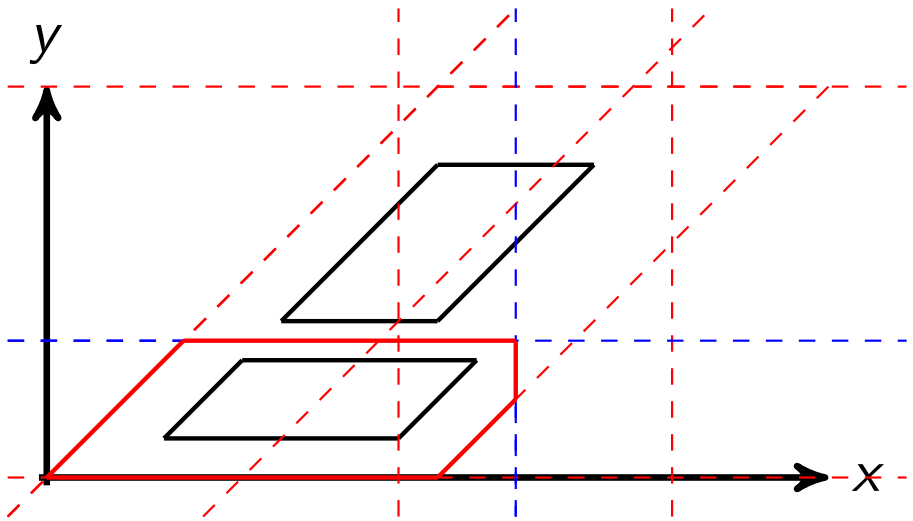
An interpolant I of (A, B), a zone s.t. $B \subseteq I$, $A \cap I = \emptyset$.
Add the constraint of the interpolant to the abstraction table.

Interpolant



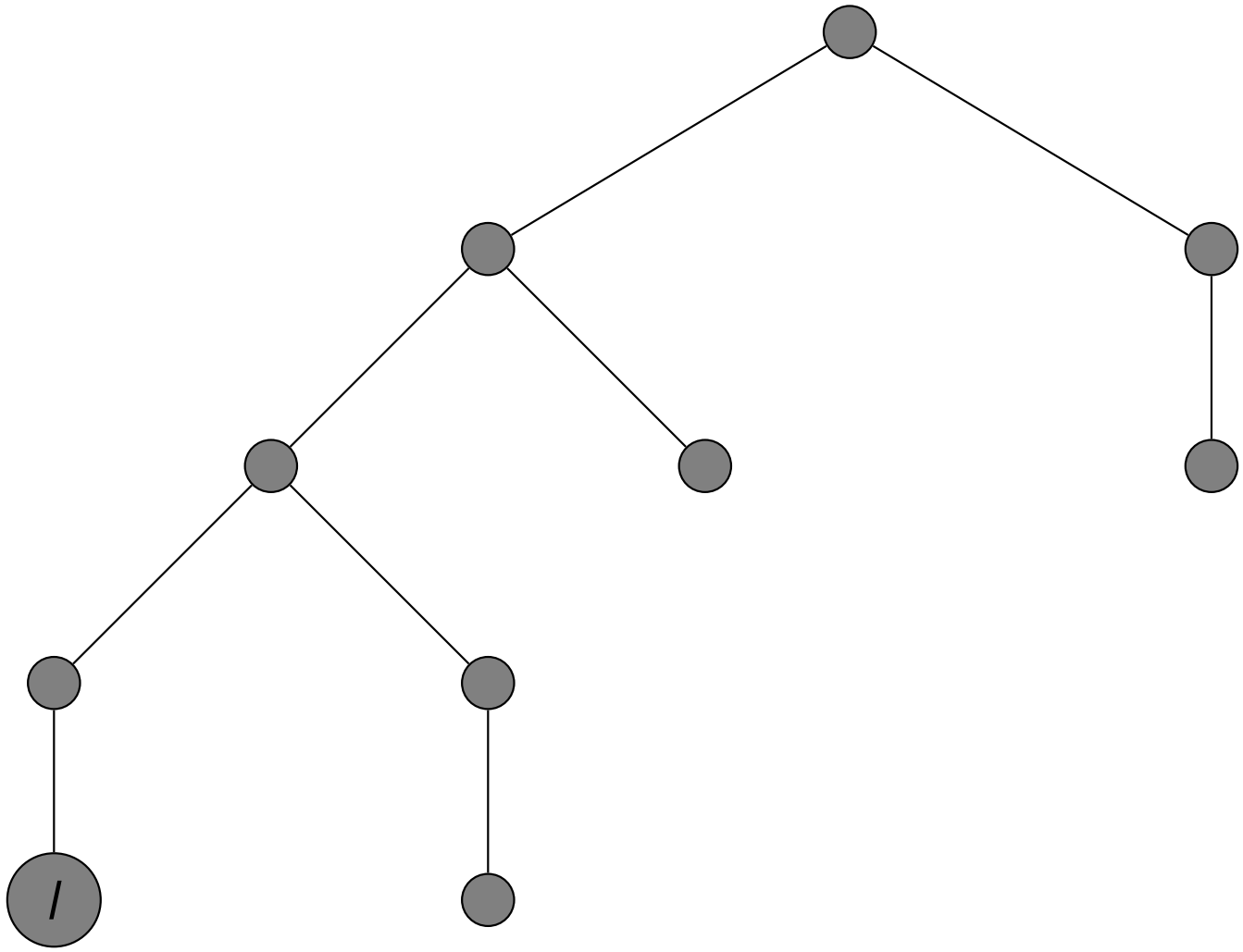
An interpolant I of (A, B), a zone s.t. $B \subseteq I$, $A \cap I = \emptyset$.
Add the constraint of the interpolant to the abstraction table.

Interpolant

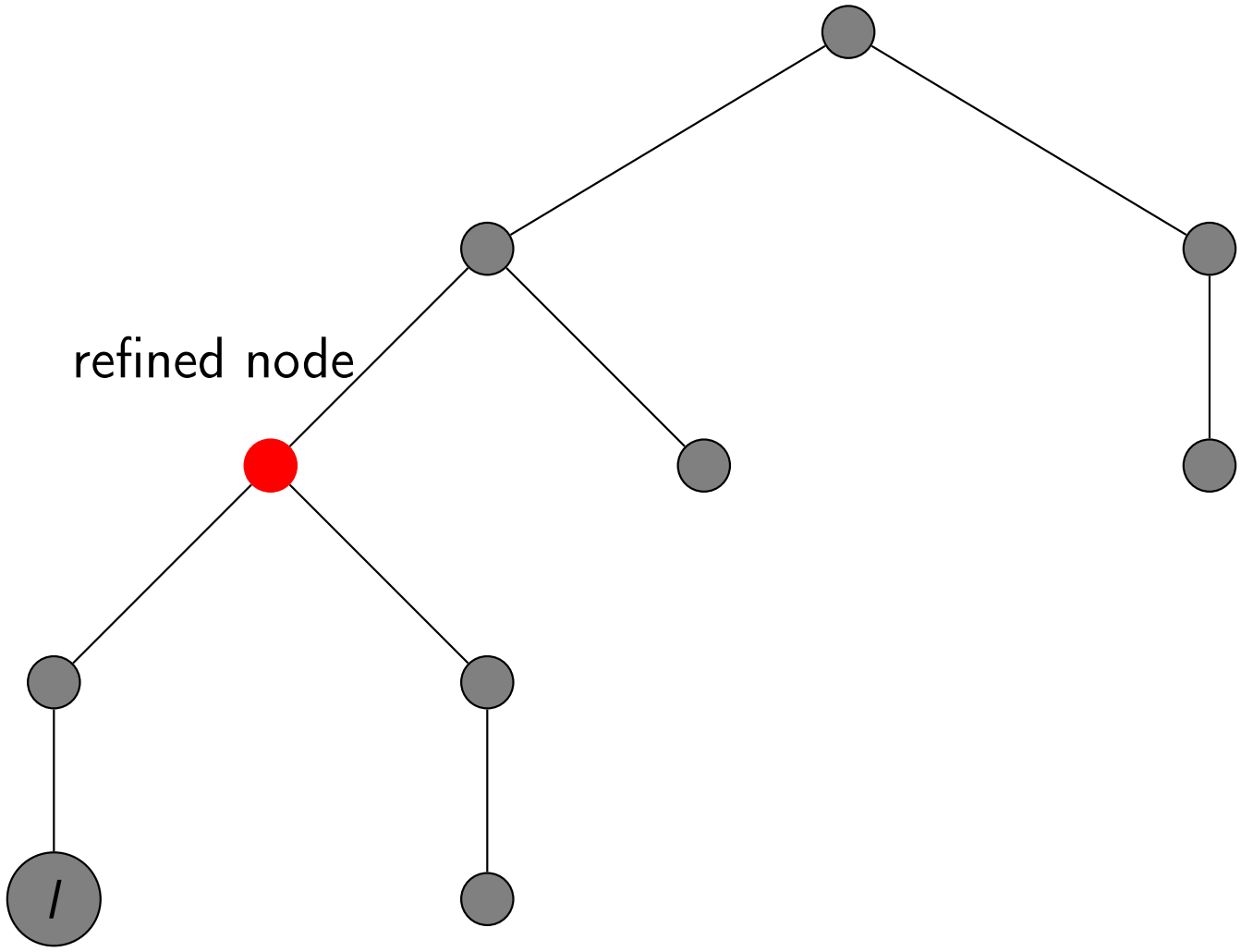


An interpolant I of (A, B), a zone s.t. $B \subseteq I$, $A \cap I = \emptyset$.
Add the constraint of the interpolant to the abstraction table.
Find the minimal interpolant.

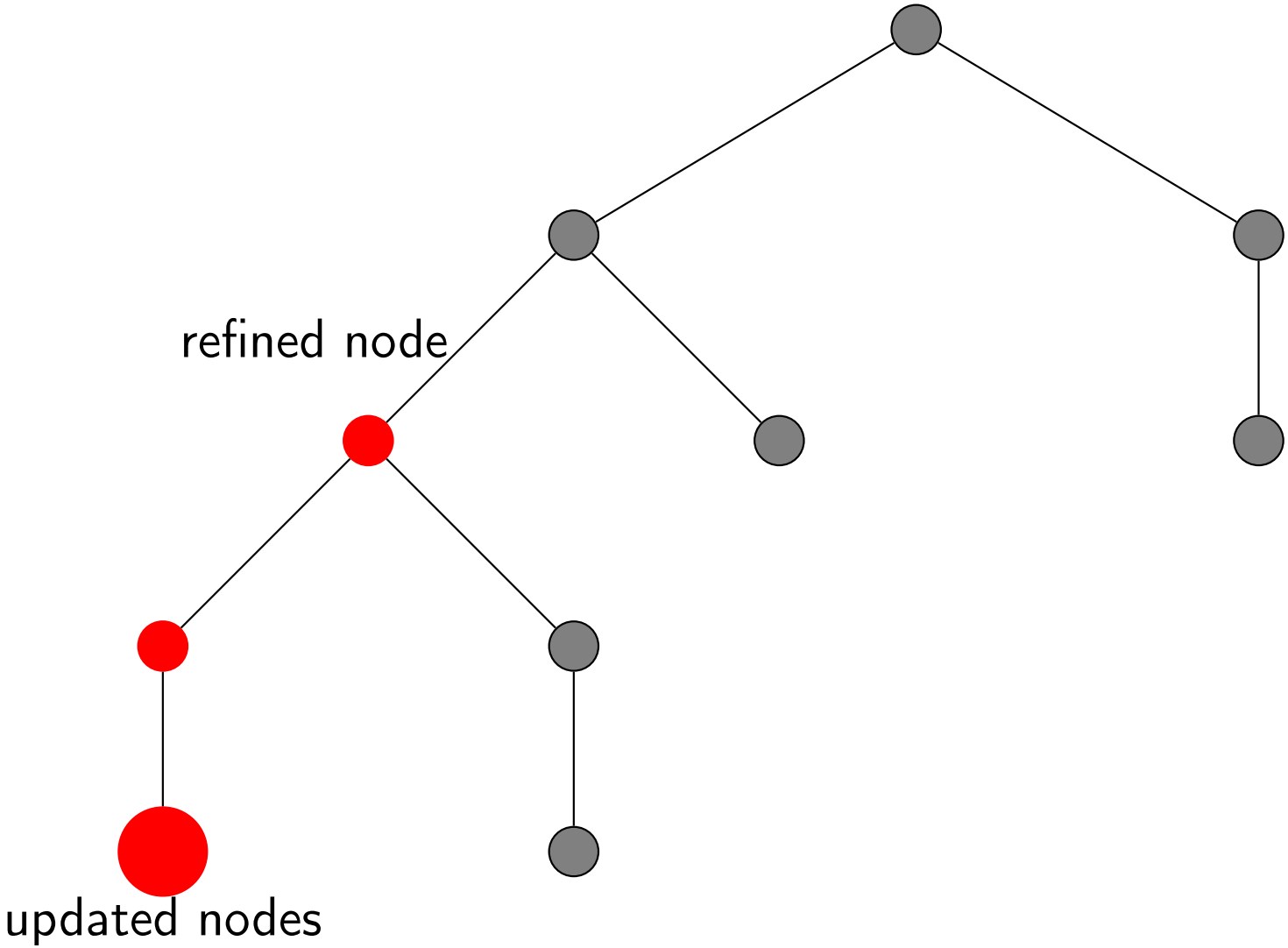
Updating nodes



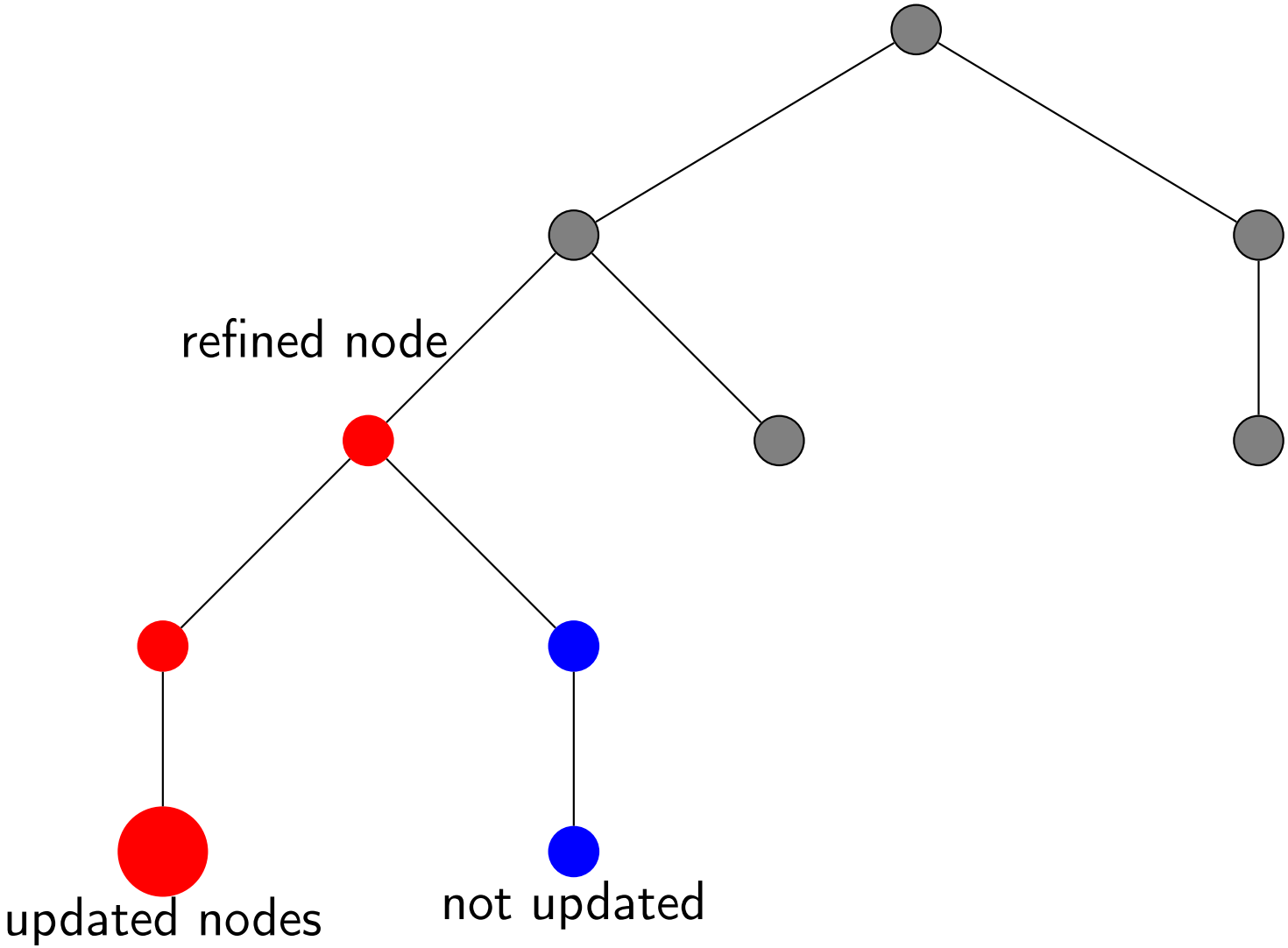
Updating nodes



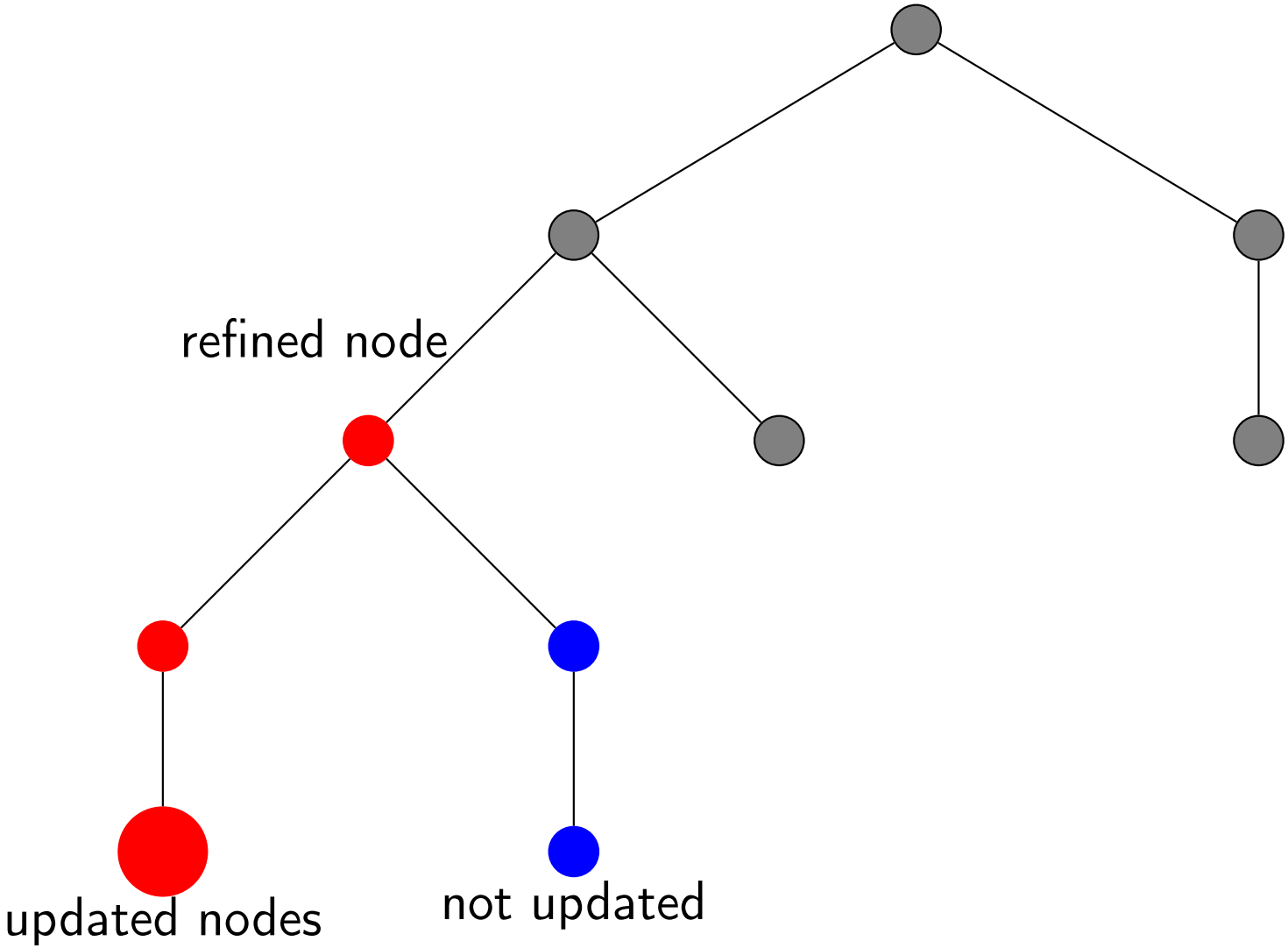
Updating nodes

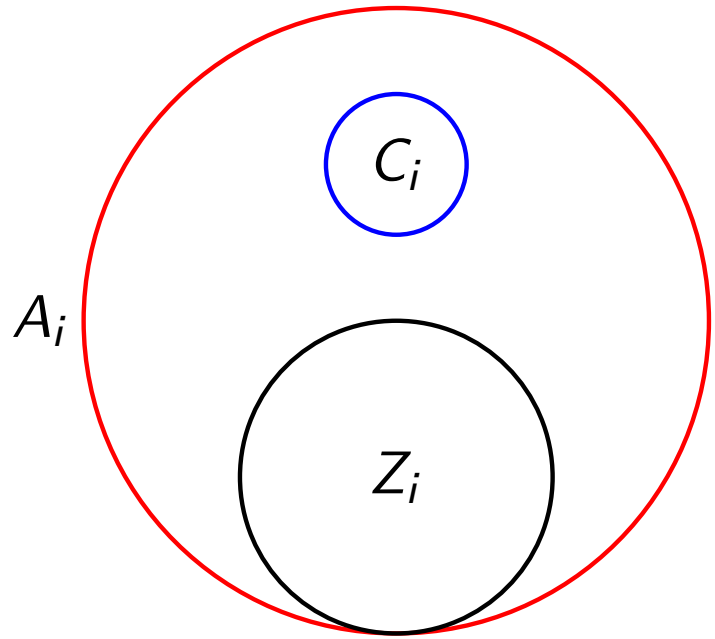


Updating nodes

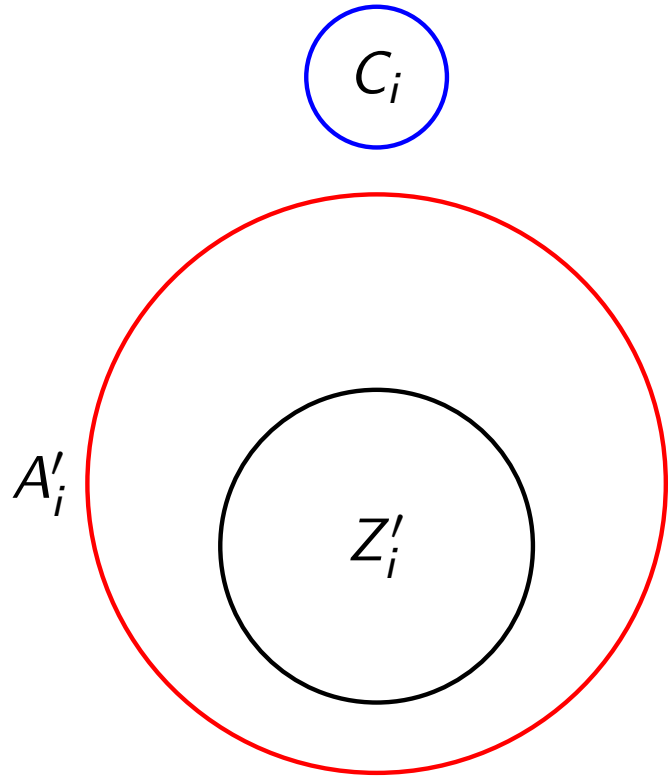


Updating nodes

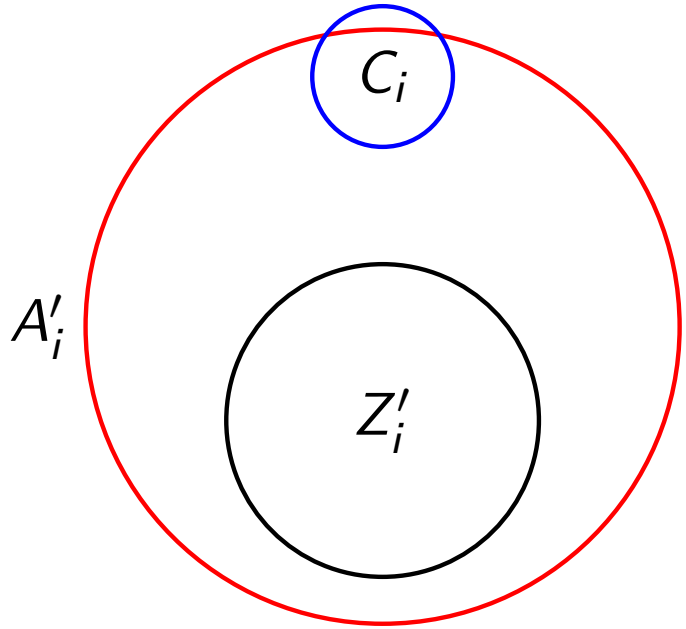




Needs to compute $Z'_i = Post(A_{i-1})$ and $A'_i = \alpha_{T_i}(Z'_i)$



Needs to compute $Z'_i = Post(A_{i-1})$ and $A'_i = \alpha_{T_i}(Z'_i)$



Needs to compute $Z'_i = Post(A_{i-1})$ and $A'_i = \alpha_{T_i}(Z'_i)$

Implementation

Implemented in TChecker.
To ensure termination, we need to update (or cut) the subtree after a refinement.
Several heuristics that ensure termination.
Different uses for abstraction table:

- A unique shared abstraction table.
- One abstraction table per node.
- One abstraction table per location.

Symbolic Algorithm

Use BDD: zones as a Boolean formula over predicates $p_{x-y \prec k}$.
If P is a set of predicates, then $\mathbb{B}^P$ the set of minterms.
A minterm f defines a zone $[\![f]\!] = \{v \mid \forall p \in P, v \models p \text{ iff } f(p)\}$.

Symbolic Algorithm

Use BDD: zones as a Boolean formula over predicates $p_{x-y \prec k}$.
If P is a set of predicates, then $\mathbb{B}^P$ the set of minterms.
A minterm f defines a zone $[\![f]\!] = \{v \mid \forall p \in P, v \models p \text{ iff } f(p)\}$.
Example: $P = \{p_{x-y \leq 0}, p_{x-y<0}\}$,$f = \{p_{x-y \leq 0}\}$, then $[\![f]\!]$ is $x = y$.

Symbolic Algorithm

Use BDD: zones as a Boolean formula over predicates $p_{x-y \prec k}$.
If P is a set of predicates, then $\mathbb{B}^P$ the set of minterms.
A minterm f defines a zone $[\![f]\!] = \{v \mid \forall p \in P, v \models p \text{ iff } f(p)\}$.
Example: $P = \{p_{x-y \leq 0}, p_{x-y<0}\}$,$f = \{p_{x-y \leq 0}\}$, then $[\![f]\!]$ is $x = y$.
Smallest zones possible with P. If P has all possible predicates, then we have regions.
A boolean formula F is an union of minterms and $[\![F]\!]$ can be defined.

Abstraction

Limit the number of predicates using abstraction.
Abstract domain D s.t. if $x < y$, $D_{x,y} \subseteq \mathbb{Z} \times \{<, \leq\}$ defines the sets of predicates P^D.
Abstraction function $\alpha_D(Z) = \{f \mid f \in \mathbb{B}^{P^D} \text{ and } Z \cap [\![f]\!] \neq \emptyset\}$.

Abstraction

Limit the number of predicates using abstraction.
Abstract domain D s.t. if $x < y$, $D_{x,y} \subseteq \mathbb{Z} \times \{<, \leq\}$ defines the sets of predicates P^D.
Abstraction function $\alpha_D(Z) = \{f \mid f \in \mathbb{B}^{P^D} \text{ and } Z \cap [\![f]\!] \neq \emptyset\}$.
$[\![\alpha_D(\cdot)]\!]$ is almost the same abstraction function as in the enumerative algorithm.

Reduction

Similar to closure operation in DBM.
Needed to apply successor operations.
Eliminate unsatisfiable minterms (negative cycles).
We only look triangle inequality of size 2 (2-reduction).

Reduction

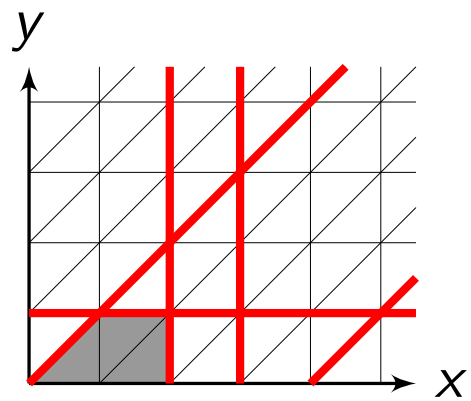
Similar to closure operation in DBM.
Needed to apply successor operations.
Eliminate unsatisfiable minterms (negative cycles).
We only look triangle inequality of size 2 (2-reduction).

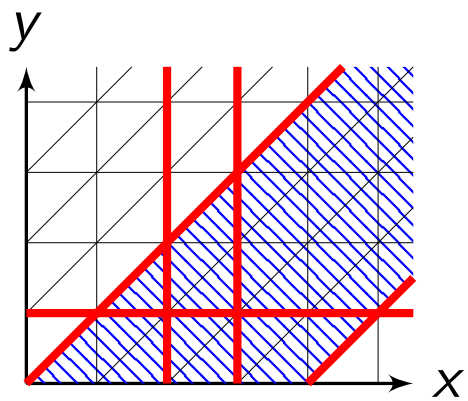
$$\bigwedge_{\substack{(x,y)\in C_{\mathbf{0}}^{\mathbf{2}} \\ (k,\prec_k)\in D_{x,y}}} \left[\left(p_{x-y\prec_k k} \leftarrow \left(\bigvee_{\substack{(l,\prec)\in D_{x,y} \\ (l,\prec)\leq(k,\prec_k)}} p_{x-y\prec l} \vee \bigvee_{\substack{z\in C_{\mathbf{0}},(l,\prec)\in D_{x,z}, \\ (l',\prec')\in D_{z,y} \\ (l,\prec)+(l',\prec')\leq(k,\prec_k)}} p_{x-z\prec l} \wedge p_{z-y\prec' l'} \right) \right) \right]$$

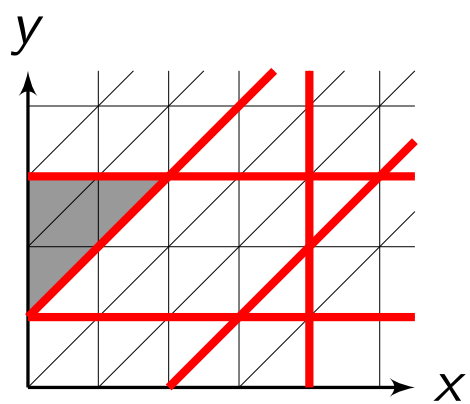
We can refine D if this is not enough.

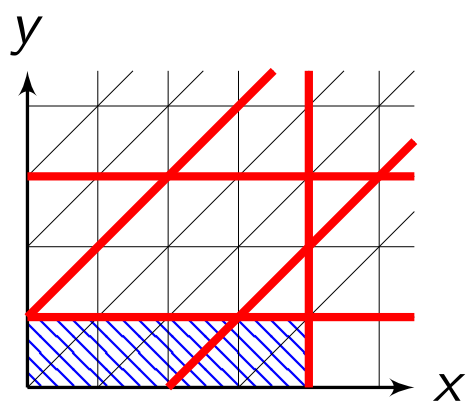
Successor operator

There is an Up_D and a $reset_{D,x}$ operator such that $\alpha_D([\![F]\!]\uparrow) \subseteq Up_D(F)$ and $\alpha_D([\![F]\!][x \leftarrow 0]) \subseteq reset_{D,x}(F)$









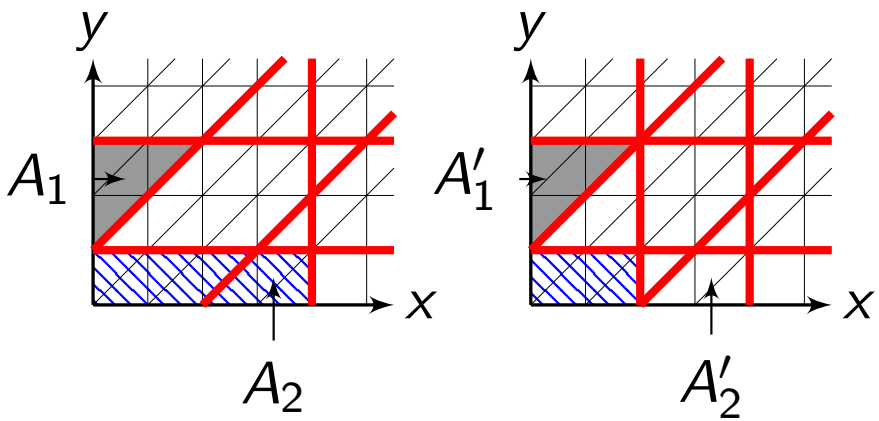
Algorithm

Implemented in Symrob.

- Exploration with Boolean formula.
- If target state is reached, check the feasability of the trace.
- Refine and restart by adding the needed value to the domain if the trace was spurious.

Refinement

Same method as enumerative algorithm.
We might need to refine because the reduction was not correct.
Or the succesor function might need a refinement.

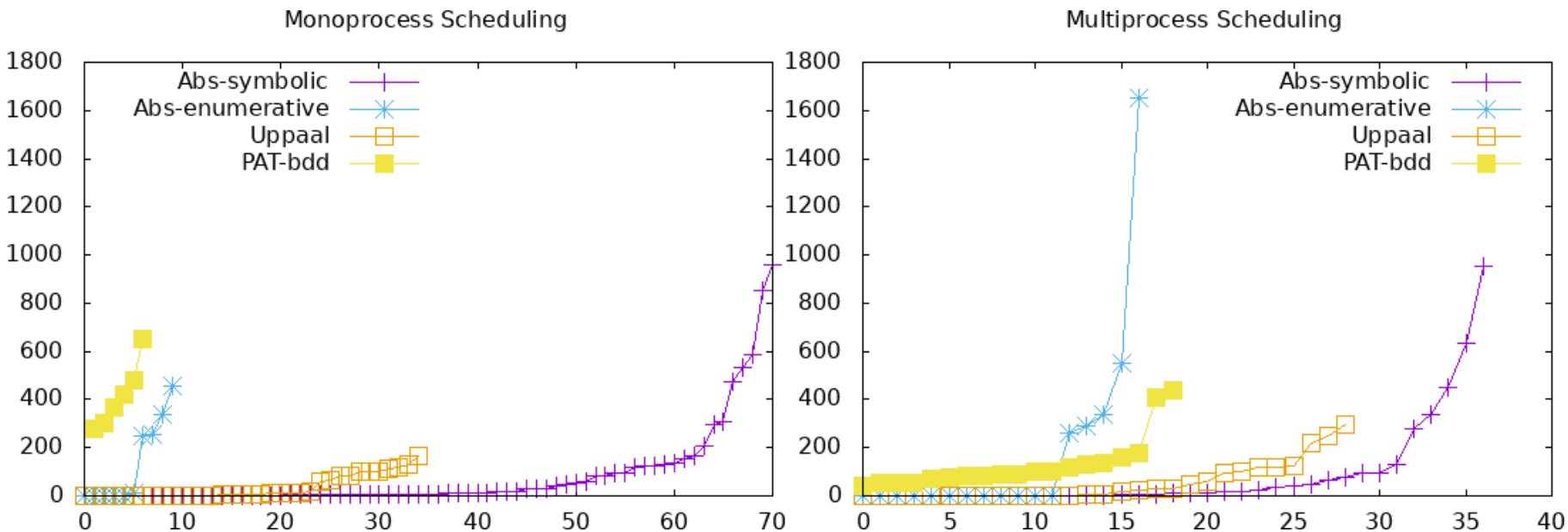


Benchmarks

Comparison with UPPAAL and PAT on 4 models

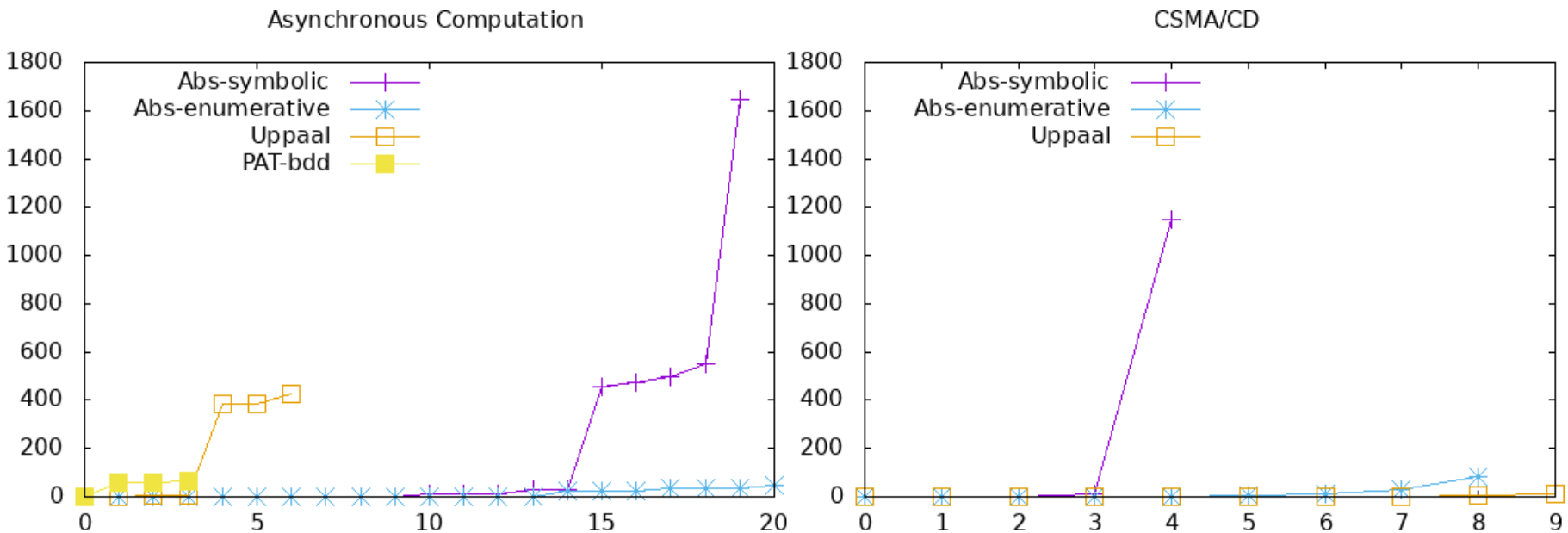
- CSMA-CD
- Monoprocesse scheduling. Measure maximum execution time on a single process. Synchronous circuits.
- Multiprocesses scheduling. 3 processes, round robin. Check that every deadline is met.
- Asynchronous Comptation. Network of logic gates, active during a time period.

Results



A point (X,Y) means X benchmarks were solved within time bound Y.

Results



A point (X,Y) means X benchmarks were solved within time bound Y.

Future work

Understanding why some methods work better than other.
Some work on the choice of the interpolant. Is minimal interpolant the best?
Extension to priced automata.