CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Synthesis of Infinite-State Reactive Systems (Part 2)

Rayna Dimitrova | VTSA 2025 | 05.09.2025

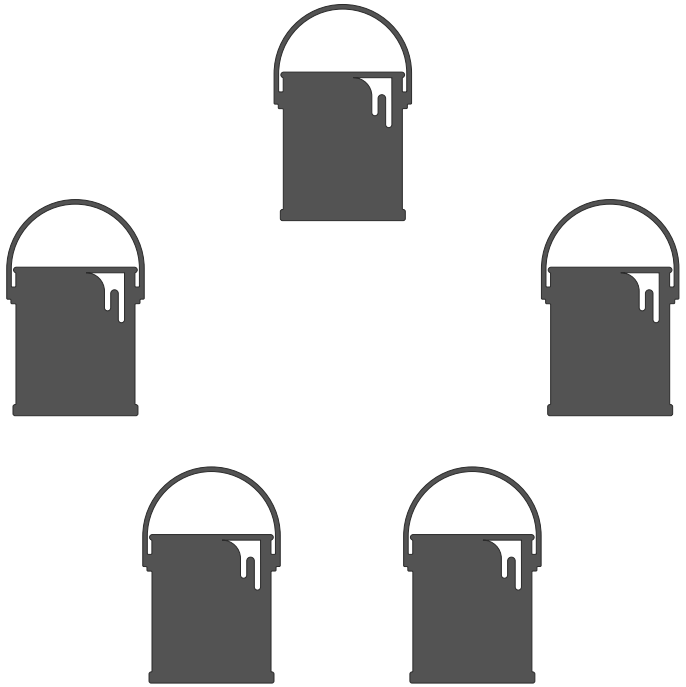
Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

[Bodlaender/Hurkens/Kusters/Staals/Woeginger/Zantema, IFIP TCS, 2012]

Let’s Start with a Puzzle

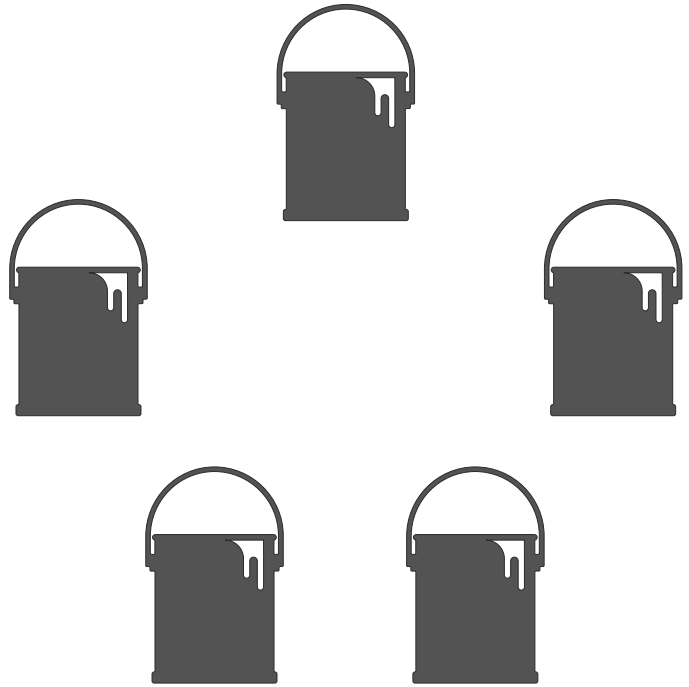
Cinderella and the wicked stepmother play a game



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

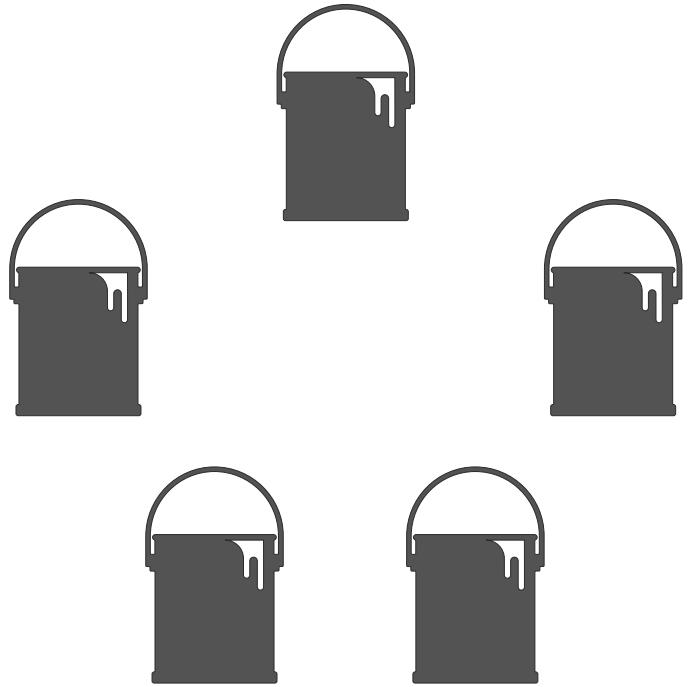
- **Five buckets** arranged in a circle



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

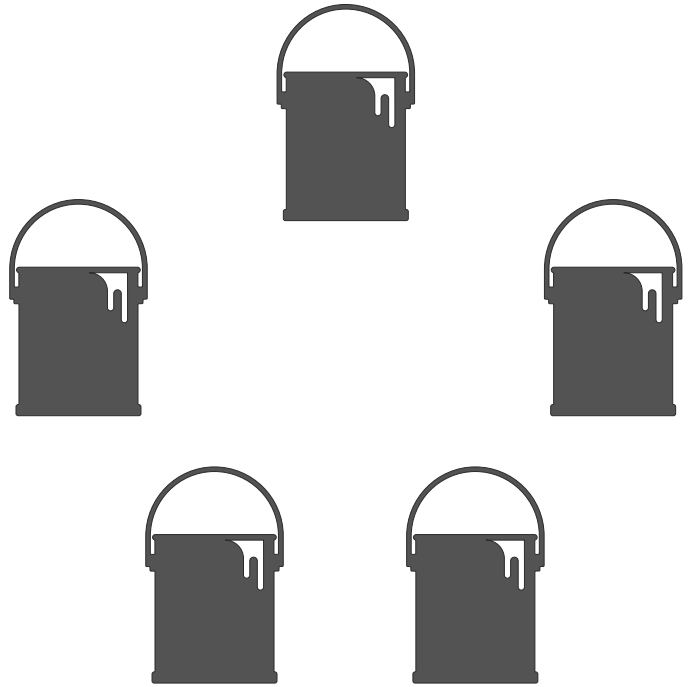
- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

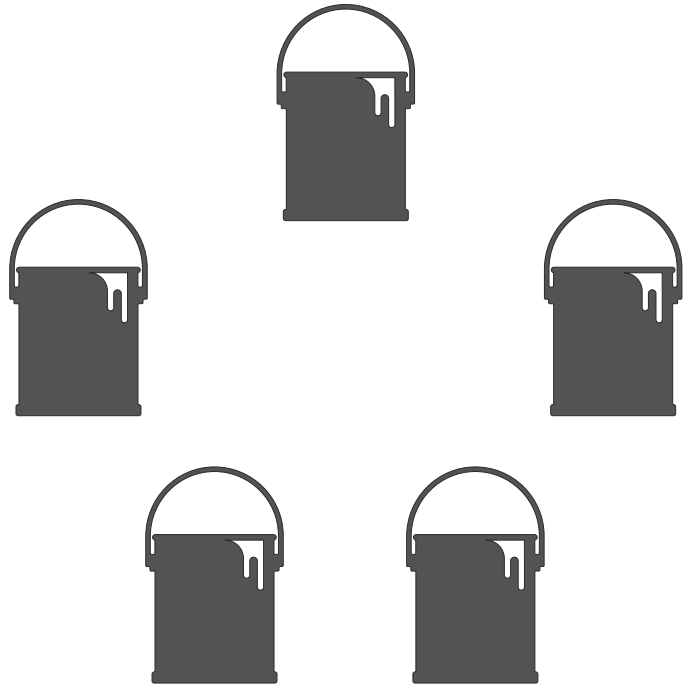
- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

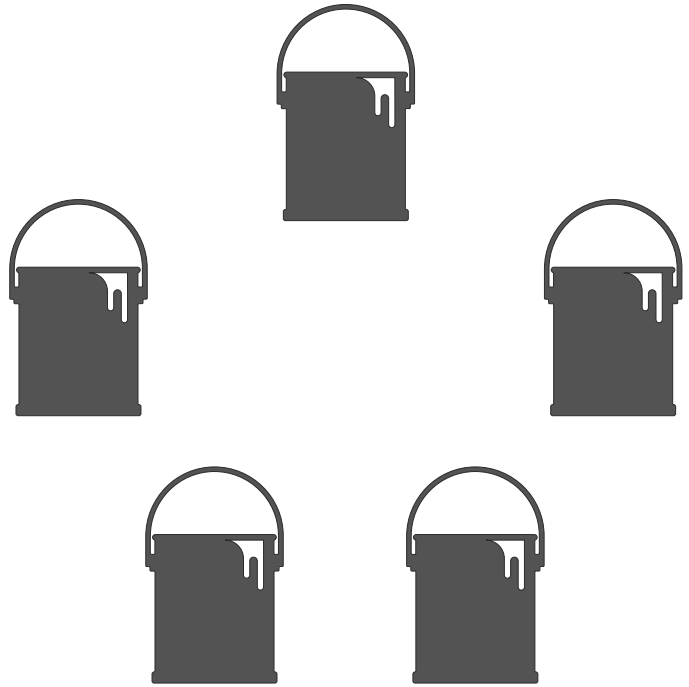
- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

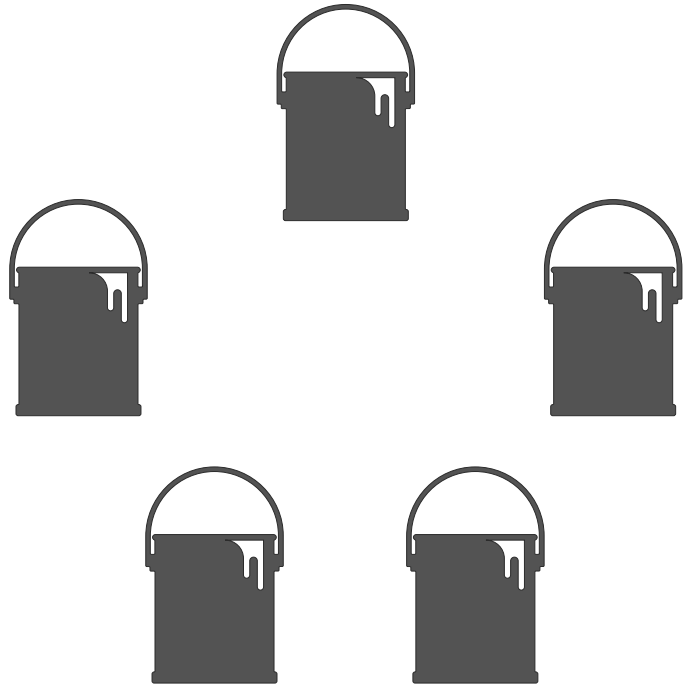
- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

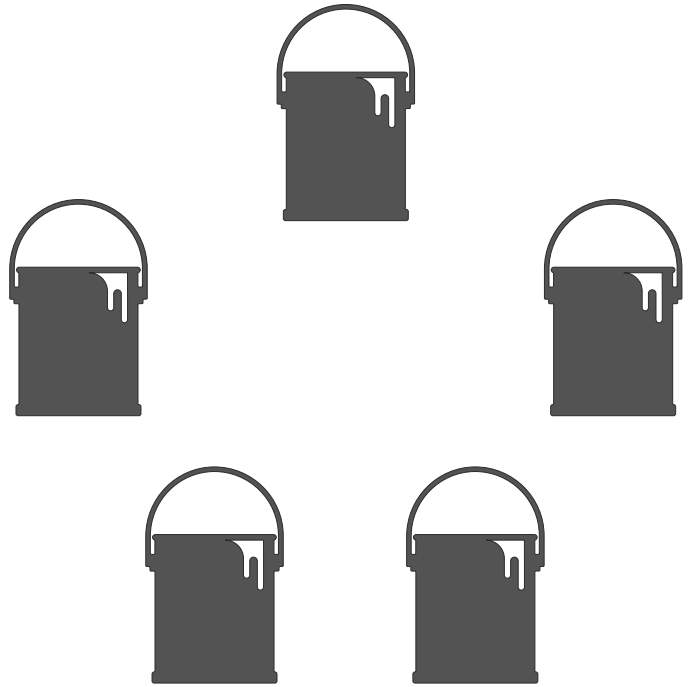
- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

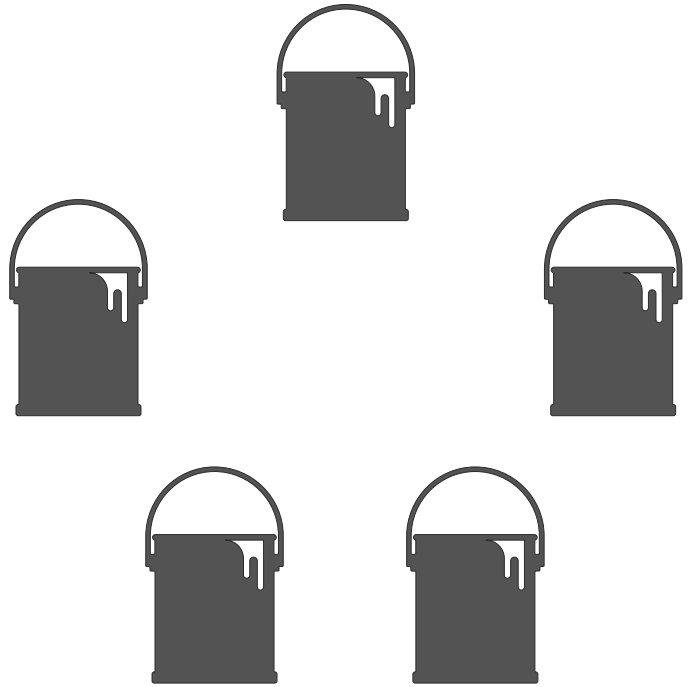
- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!
  - Cinderella **empties two adjacent buckets**



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

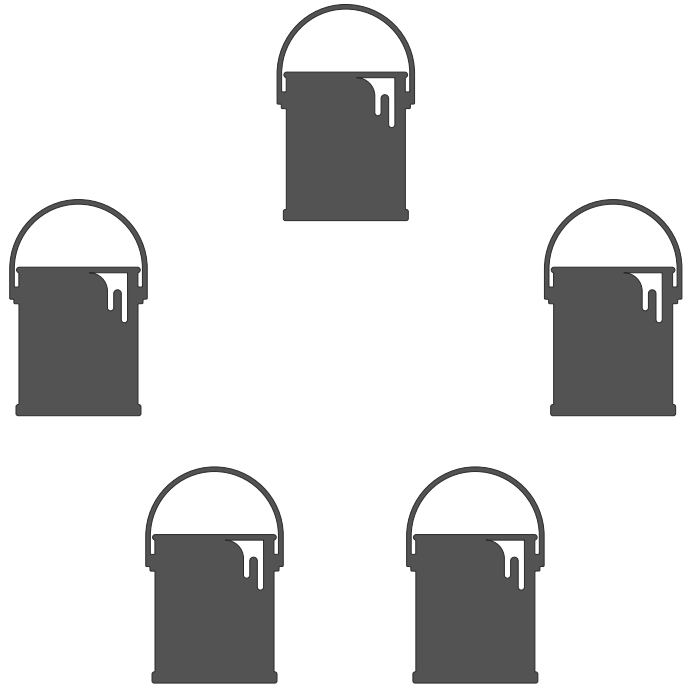
- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!
  - Cinderella **empties two adjacent buckets**
    If the game goes on forever, cinderella wins!



Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!
  - Cinderella **empties two adjacent buckets**
    If the game goes on forever, cinderella wins!

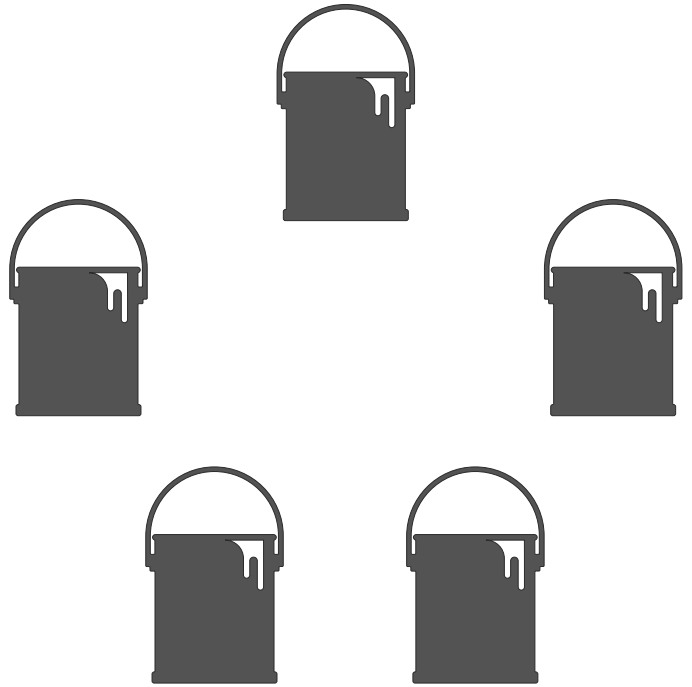


Who wins?

Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!
  - Cinderella **empties two adjacent buckets**
    If the game goes on forever, cinderella wins!

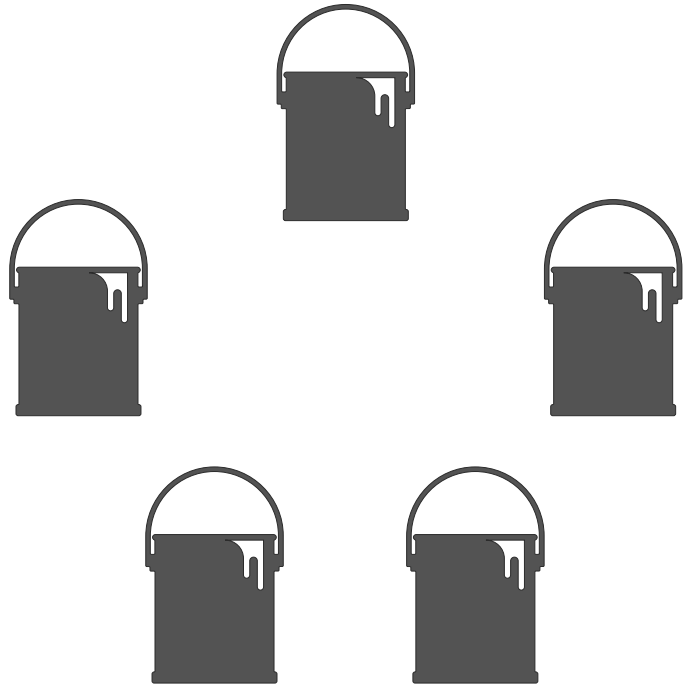


Who wins? Depends on the capacity **B** of the buckets.

Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!
  - Cinderella **empties two adjacent buckets**
    If the game goes on forever, cinderella wins!



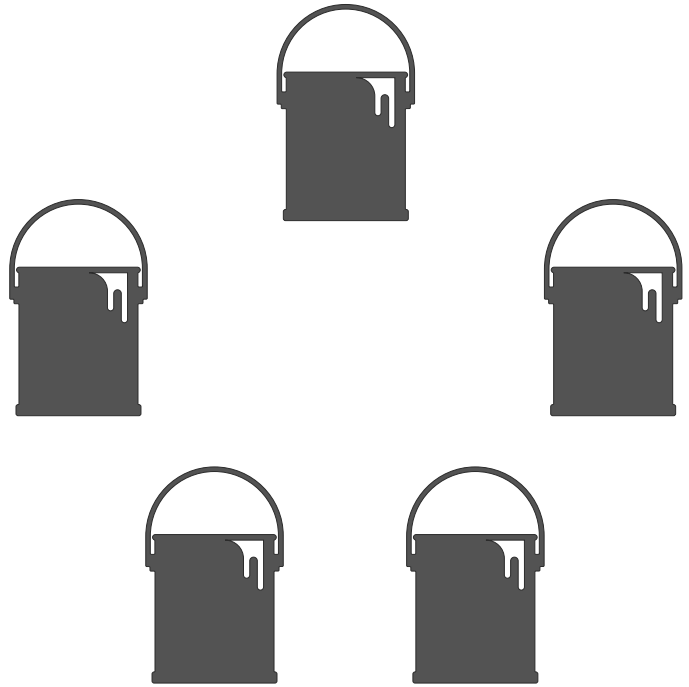
Who wins? Depends on the capacity **B** of the buckets.

Minimal B such that cinderella wins?

Let’s Start with a Puzzle

Cinderella and the wicked stepmother play a game

- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!
  - Cinderella **empties two adjacent buckets**
    If the game goes on forever, cinderella wins!



Claim: Cinderella wins if B = 2.

Let’s Start with a Puzzle

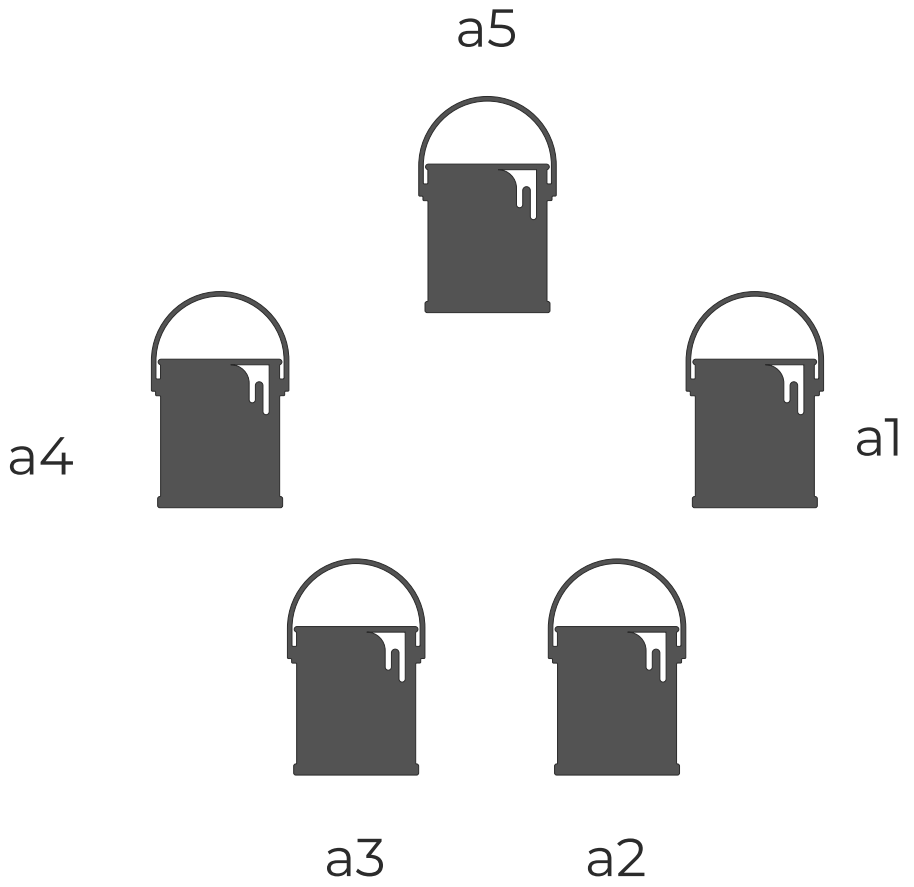
Claim: Cinderella wins if B = 2.

Let’s see that Cinderella can enforce that

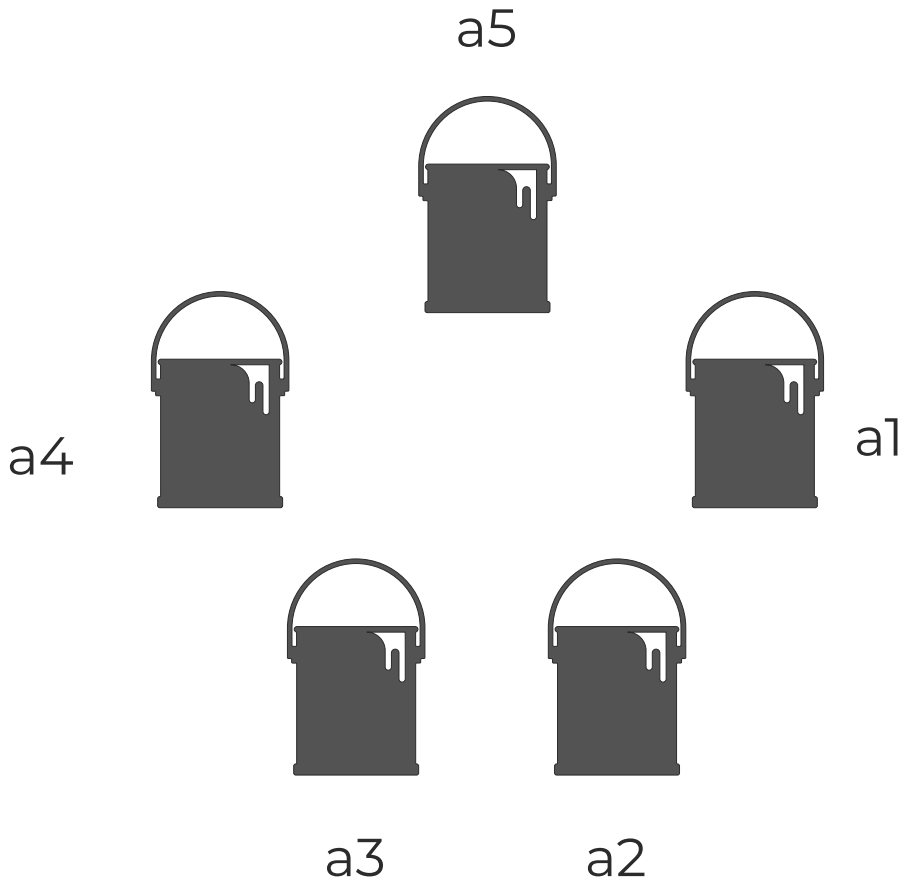
$$(a1 + a3 < 1) \wedge (a2 \leq 1) \wedge (a4 = 0) \wedge (a5 = 0)$$

always holds for the water in some buckets a1, a2, a3, a4, a5 (ordered starting from a1)

Clearly she wins if this is the case.

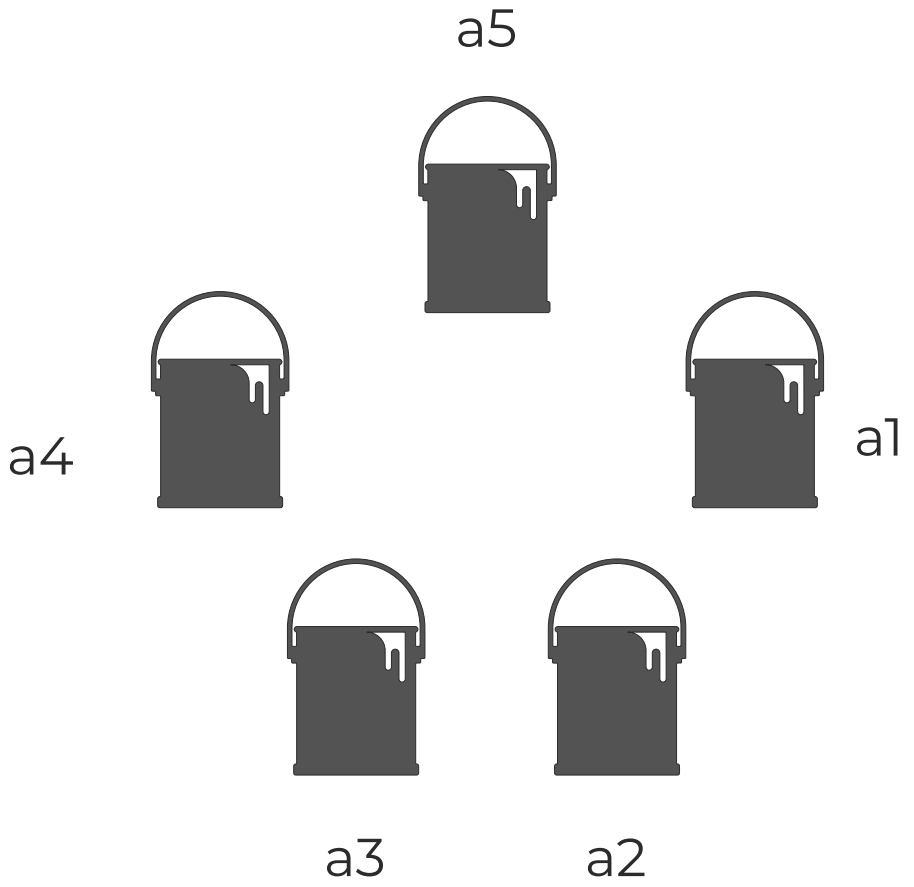


Let’s Start with a Puzzle



Let’s Start with a Puzzle

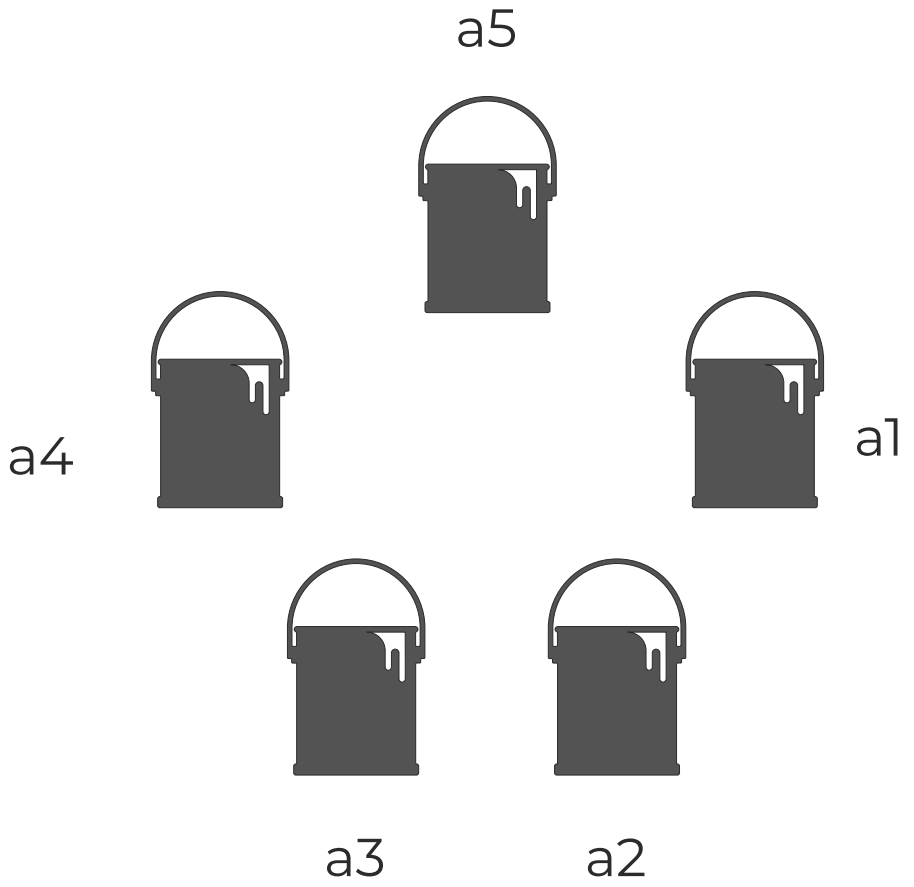
Suppose $(a1 + a3 < 1) \wedge (a2 \leq 1) \wedge (a4 = 0) \wedge (a5 = 0)$ at the begging of some round.



Let’s Start with a Puzzle

Suppose $(a1 + a3 < 1) \wedge (a2 \leq 1) \wedge (a4 = 0) \wedge (a5 = 0)$ at the begging of some round.

Stepmother adds 1 liter in total resulting in $a1', a2', a3', a4', a5'$.

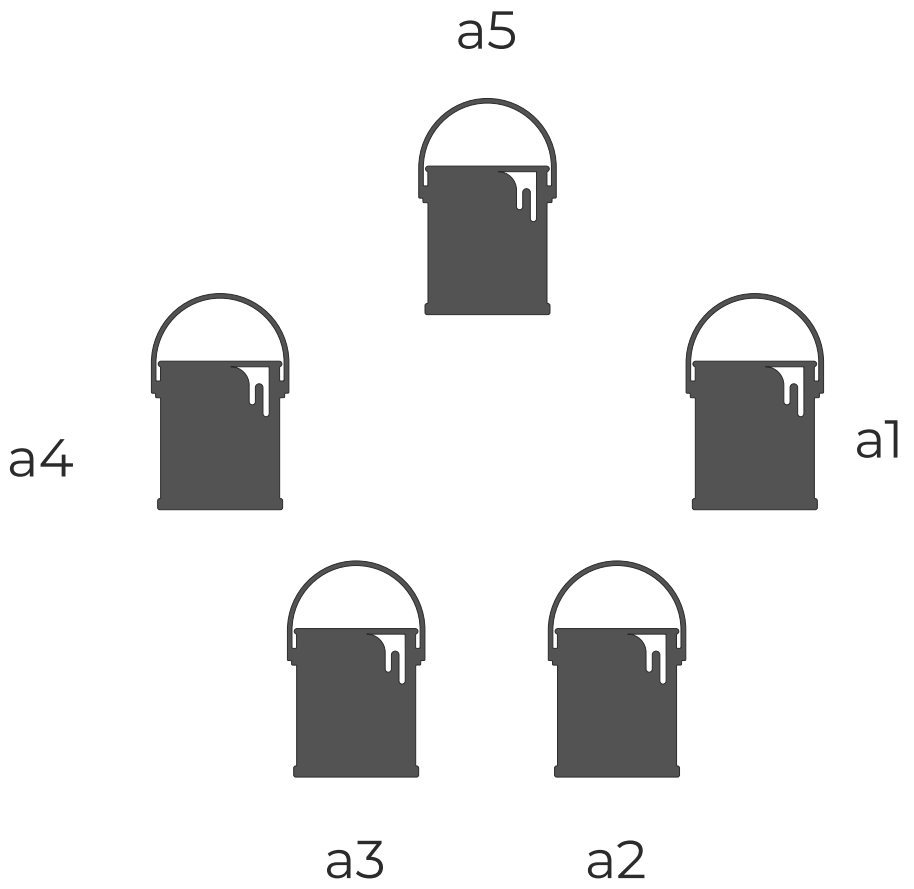


Let’s Start with a Puzzle

Suppose $(a1 + a3 < 1) \wedge (a2 \leq 1) \wedge (a4 = 0) \wedge (a5 = 0)$ at the begging of some round.

Stepmother adds 1 liter in total resulting in $a1', a2', a3', a4', a5'$.

Note that either $(a1' + a4' < 1)$ or $(a3' + a5' < 1)$.



Let's Start with a Puzzle

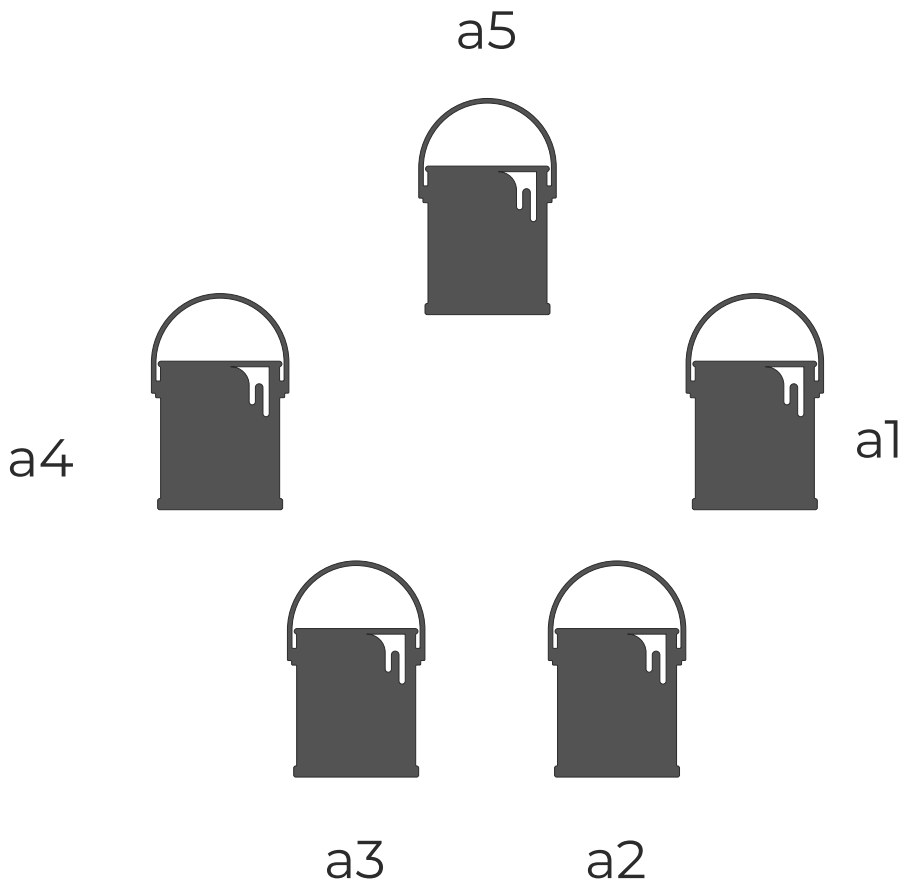
Suppose $(a1 + a3 < 1) \wedge (a2 \leq 1) \wedge (a4 = 0) \wedge (a5 = 0)$ at the begging of some round.

Stepmother adds 1 liter in total resulting in $a1', a2', a3', a4', a5'$.

Note that either $(a1' + a4' < 1)$ or $(a3' + a5' < 1)$.

Then:

- If $(a1' + a4' < 1)$, Cinderella empties 2 and 3.



Let's Start with a Puzzle

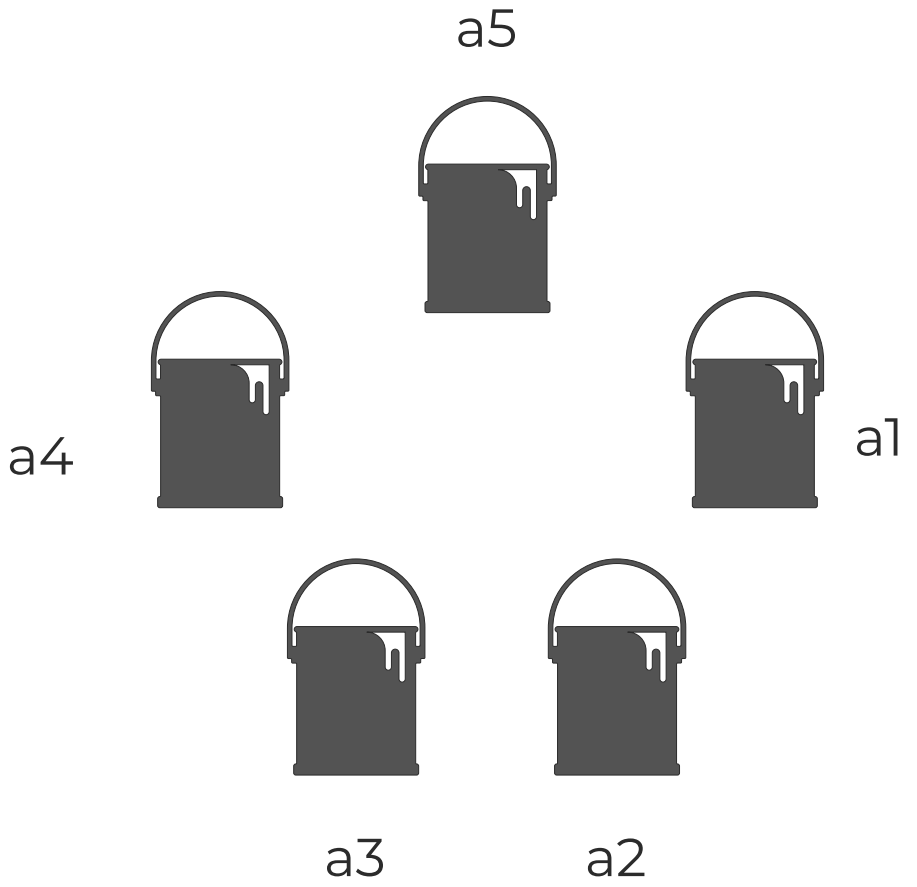
Suppose $(a1 + a3 < 1) \wedge (a2 \leq 1) \wedge (a4 = 0) \wedge (a5 = 0)$ at the begging of some round.

Stepmother adds 1 liter in total resulting in $a1', a2', a3', a4', a5'$.

Note that either $(a1' + a4' < 1)$ or $(a3' + a5' < 1)$.

Then:

- If $(a1' + a4' < 1)$, Cinderella empties 2 and 3.
- If $(a3' + a5' < 1)$, Cinderella empties 1 and 2.



Let's Start with a Puzzle

Suppose $(a1 + a3 < 1) \wedge (a2 \leq 1) \wedge (a4 = 0) \wedge (a5 = 0)$ at the begging of some round.

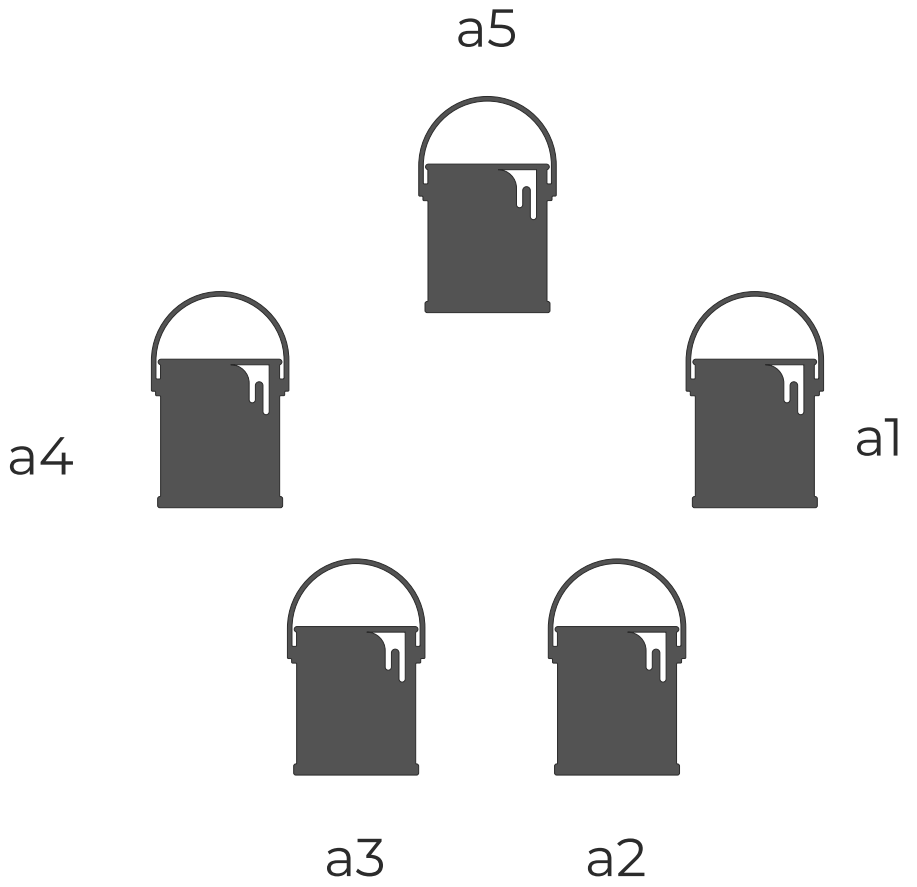
Stepmother adds 1 liter in total resulting in $a1', a2', a3', a4', a5'$.

Note that either $(a1' + a4' < 1)$ or $(a3' + a5' < 1)$.

Then:

- If $(a1' + a4' < 1)$, Cinderella empties 2 and 3.
- If $(a3' + a5' < 1)$, Cinderella empties 1 and 2.

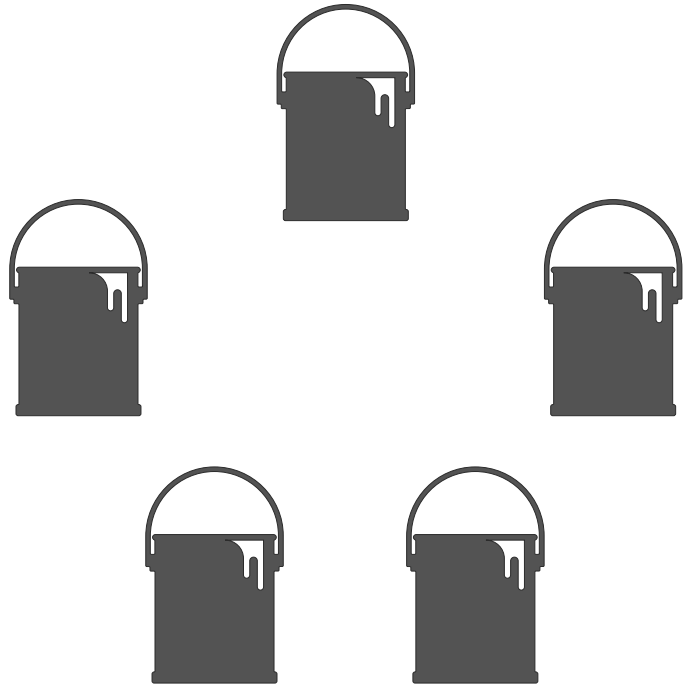
In both cases she guarantees that invariant is enforced regardless of stepmother.



Let’s Start with a Puzzle and Use Synthesis

Cinderella and the wicked stepmother play a game

- **Five buckets** arranged in a circle
- Each can hold $\leq$ **B** liters of water
- Initially all buckets are empty
- The stepmother and cinderella take turns:
  - Stepmother **brings 1 liter of water** and splits it as she likes into the five buckets.
    If any bucket overflows, stepmother wins!
  - Cinderella **empties two adjacent buckets**
    If the game goes on forever, cinderella wins!



Claim: Cinderella wins if B = 2.

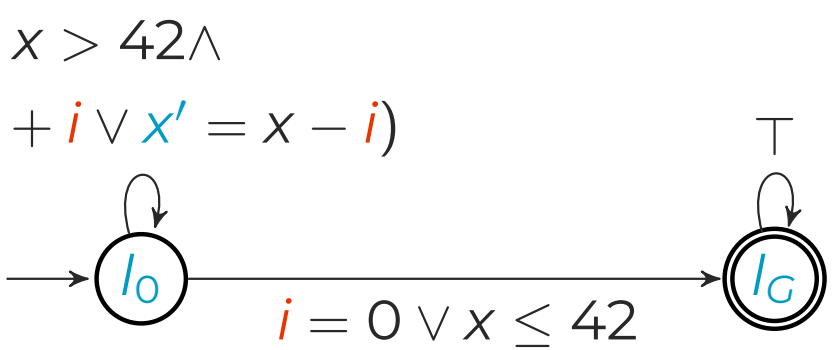
Let’s use synthesis to check that there exists a strategy for cinderella when B=2

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations

- **Symbolic methods enhanced with logical reasoning**
  - **for solving realizability/synthesis games**
  - **for translation of temporal logic formulas**
  - **combining symbolic methods and abstraction**

Naive Symbolic Methods Fail on Simple Problems

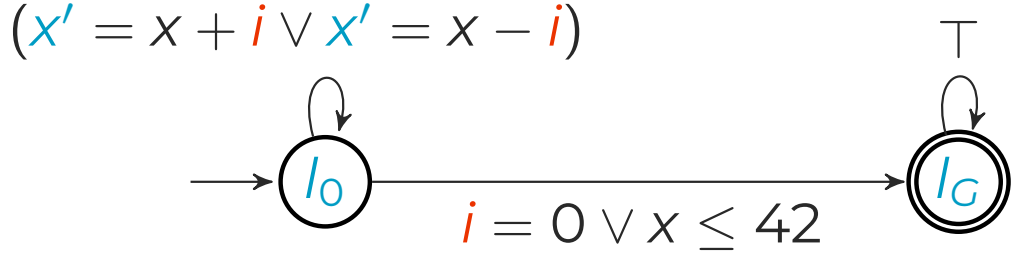


- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

Trying to compute W_{Sys} the set of states from which the system wins

Naive Symbolic Methods Fail on Simple Problems



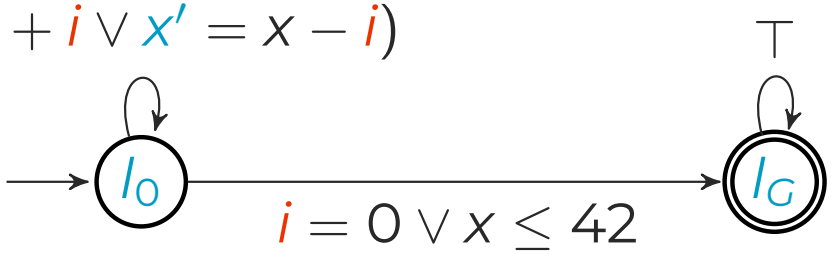
- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Trying to compute W_{Sys} the set of states from which the system wins

$$Attr^0_{Sys} := \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$$

Spec: reach l_G from l_0 for any initial value in x

Naive Symbolic Methods Fail on Simple Problems



- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

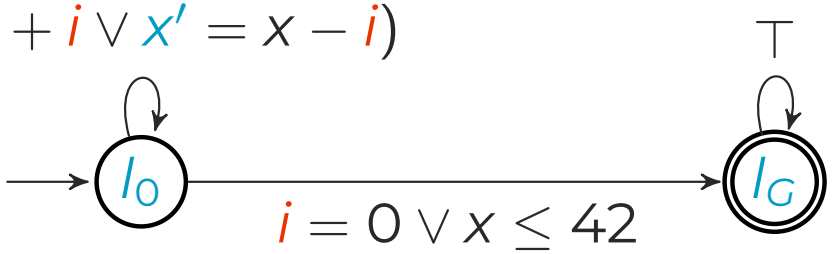
Trying to compute W_{Sys} the set of states from which the system wins

$$Attr^0_{Sys} := \{l_0 \mapsto \bot\ ,\ l_G \mapsto \top\ \}$$

$$Attr^1_{Sys} := \{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\ \}$$

Naive Symbolic Methods Fail on Simple Problems

$i \neq 0 \wedge x > 42 \wedge$
$(x' = x + i \vee x' = x - i)$



- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

Trying to compute W_{Sys} the set of states from which the system wins

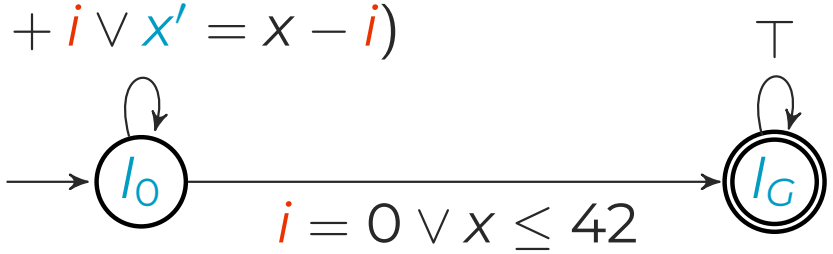
$$Attr^0_{Sys} := \{l_0 \mapsto \bot\ ,\ l_G \mapsto \top\ \}$$

$$Attr^1_{Sys} := \{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\ \}$$

$$Attr^2_{Sys} := \{l_0 \mapsto x \leq 43,\ l_G \mapsto \top\ \}$$

Naive Symbolic Methods Fail on Simple Problems

$i \neq 0 \wedge x > 42 \wedge$
$(x' = x + i \vee x' = x - i)$



- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

Trying to compute W_{Sys} the set of states from which the system wins

$$Attr^0_{Sys} := \{l_0 \mapsto \bot\ ,\ l_G \mapsto \top\ \}$$

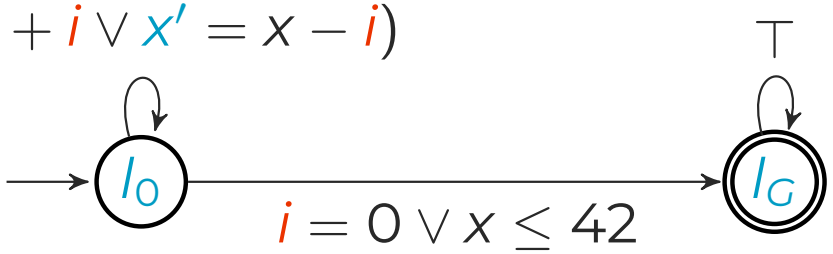
$$Attr^1_{Sys} := \{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\ \}$$

$$Attr^2_{Sys} := \{l_0 \mapsto x \leq 43,\ l_G \mapsto \top\ \}$$

Does not terminate ...

Naive Symbolic Methods Fail on Simple Problems

$i \neq 0 \wedge x > 42 \wedge$
$(x' = x + i \vee x' = x - i)$



- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

Trying to compute W_{Sys} the set of states from which the system wins

$$Attr^0_{Sys} := \{l_0 \mapsto \bot\ ,\ l_G \mapsto \top\ \}$$

$$Attr^1_{Sys} := \{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\ \}$$

$$Attr^2_{Sys} := \{l_0 \mapsto x \leq 43,\ l_G \mapsto \top\ \}$$

Does not terminate ...

Idea:

accelerate to compute

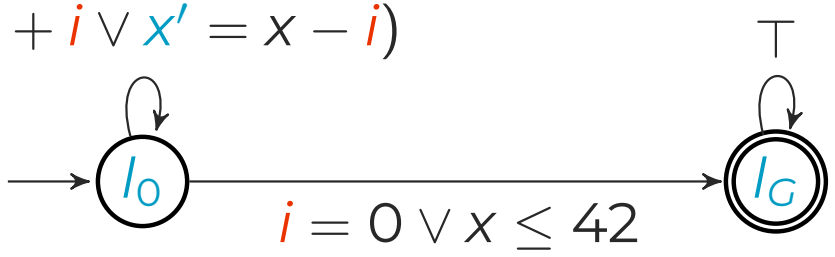
$$W_{Sys} = \{l_0 \mapsto \top\ ,\ l_G \mapsto \top\ \}$$

Idea: Accelerate Symbolic Game Solving

[Heim/Dimitrova, POPL 2024]

$i \neq 0 \wedge x > 42 \wedge$

$(x' = x + i \vee x' = x - i)$



Spec: reach l_G from l_0 for any initial value in x

Intuitive argument

Whatever the starting value of x is at l_0,

if the system can enforce that it can

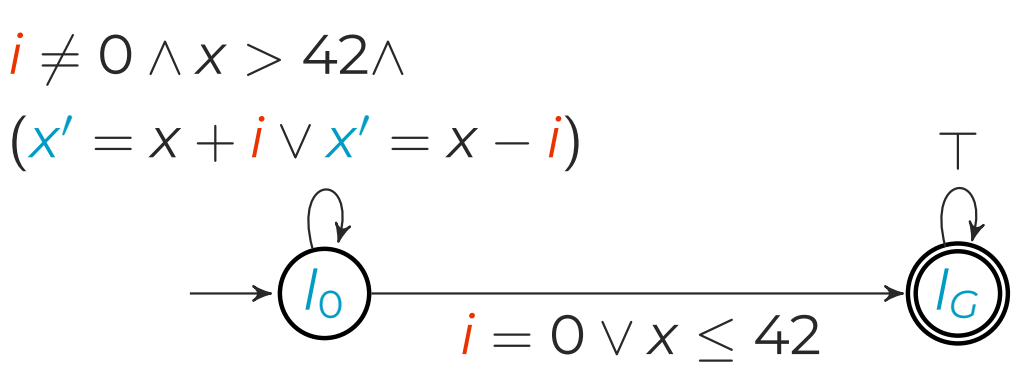
either reach l_G in one step

*or can **decrement x and get back to l_0,***

*then **eventually** $x \leq 42$ or l_G will be reached.*

Idea: Accelerate Symbolic Game Solving

[Heim/Dimitrova, POPL 2024]



Spec: reach l_G from l_0 for any initial value in x

Intuitive argument

Whatever the starting value of x is at l_0,

if the system can enforce that it can

either reach l_G in one step

*or can **decrement x and get back to l_0**,*

*then **eventually** $x \leq 42$ or l_G will be reached.*

Attractor acceleration informally

- From $x \leq 42$ in l_0 the system can reach the goal
- The system can ensure to decrement x from l_0 back to l_0 (or reach l_G)

➡ From $x > 42$ in l_0 the system can also reach the goal

Acceleration Lemmas

Formalize the intuition: Whenever $x > 42$, if the system can ensure that it can decrement x, then eventually $x \leq 42$ will be reached.

Acceleration Lemma [Heim/Dimitrova, POPL 2024]

Triple $(\mathit{base}, \mathit{step}, \mathit{conc})$ of FOL formulas

- $\mathit{conc} = \top$
- $\mathit{step} = x' < x$
- $\mathit{base} = x \leq 42$

For infinite sequence of states α

- $\alpha[0] \vDash \mathit{conc}$
- $\forall i \,.\, \langle \alpha[i], \alpha[i+1] \rangle \vDash \mathit{step}$

➡ $\exists k \,.\, \alpha[k] \vDash \mathit{base}$

Condition does not depend on game!

Applying Acceleration Lemmas

Non-terminating attractor computation:

$$A_0 = \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$$

$$A_1 = \{ l_0 \mapsto x \leq 42, \, l_G \mapsto \top \}$$

...

Applying Acceleration Lemmas

Non-terminating attractor computation:

$A_0 = \{ l_0 \mapsto \bot \, , \, l_G \mapsto \mathsf{T} \, \}$

$A_1 = \{ l_0 \mapsto x \leq 42, \, l_G \mapsto \mathsf{T} \, \}$

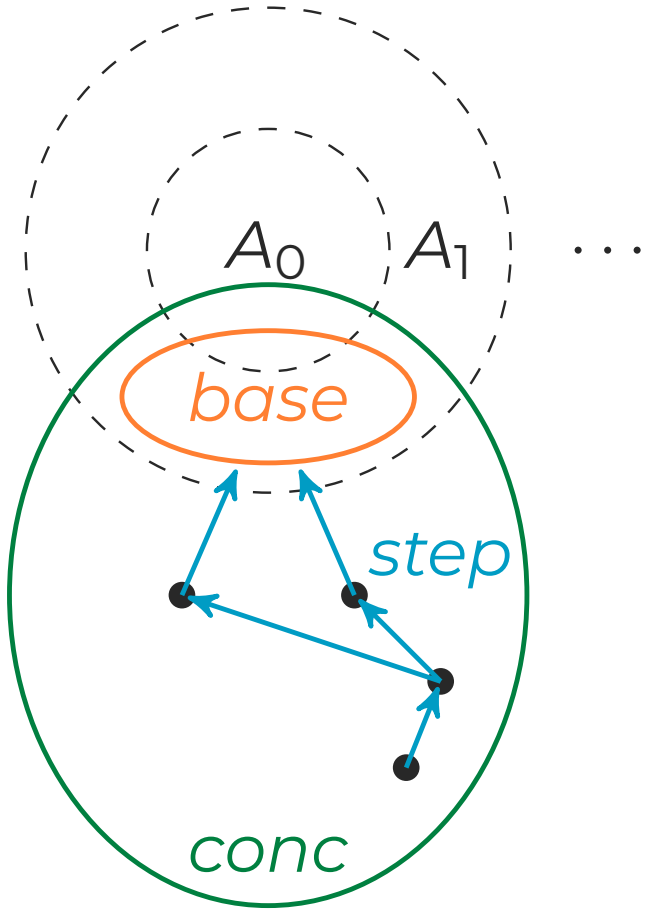
...

Attractor acceleration

Lemma $(x \leq 42, x' < x, \mathsf{T})$

- system reaches goal from $x \leq 42$ in l_0
- system can enforce $x' < x$ from l_0 to l_0

➡ system can reach the goal from T in l_0



Applying Acceleration Lemmas

Non-terminating attractor computation:

$A_0 = \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$

$A_1 = \{ l_0 \mapsto x \leq 42, \, l_G \mapsto \top \}$

…

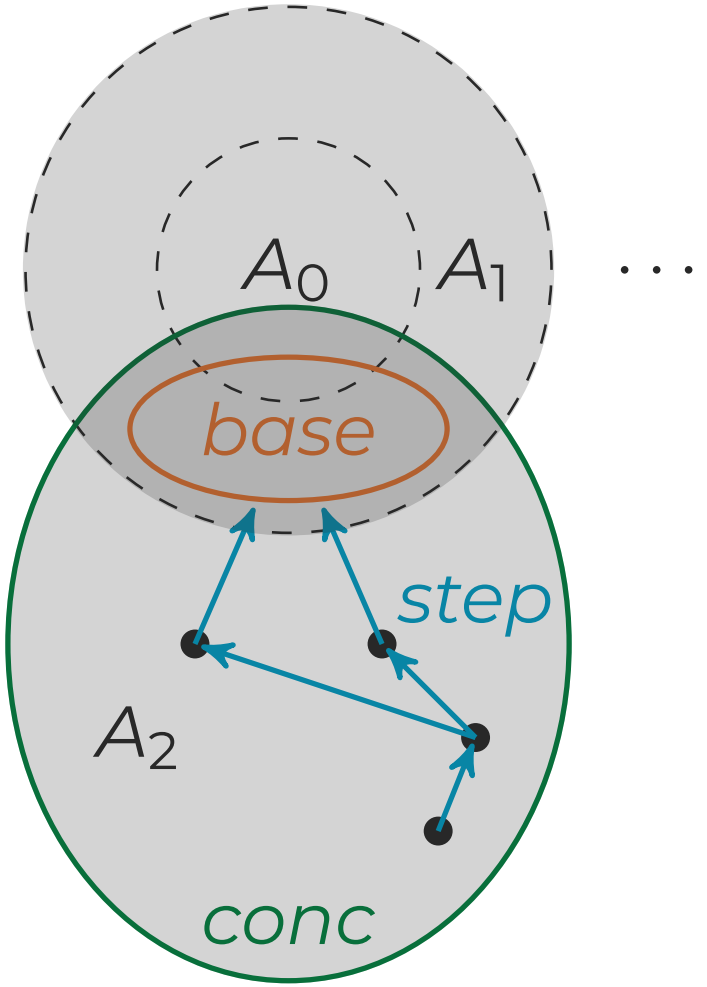
Attractor acceleration

Lemma $(x \leq 42, x' < x, \top)$

- system reaches goal from $x \leq 42$ in l_0
- system can enforce $x' < x$ from l_0 to l_0

➡ system can reach the goal from $\top$ in l_0

➡ add $\top$ to attractor in l_0



Applying Acceleration Lemmas

Non-terminating attractor computation:

$A_0 = \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$

$A_1 = \{ l_0 \mapsto x \leq 42, \, l_G \mapsto \top \}$

…

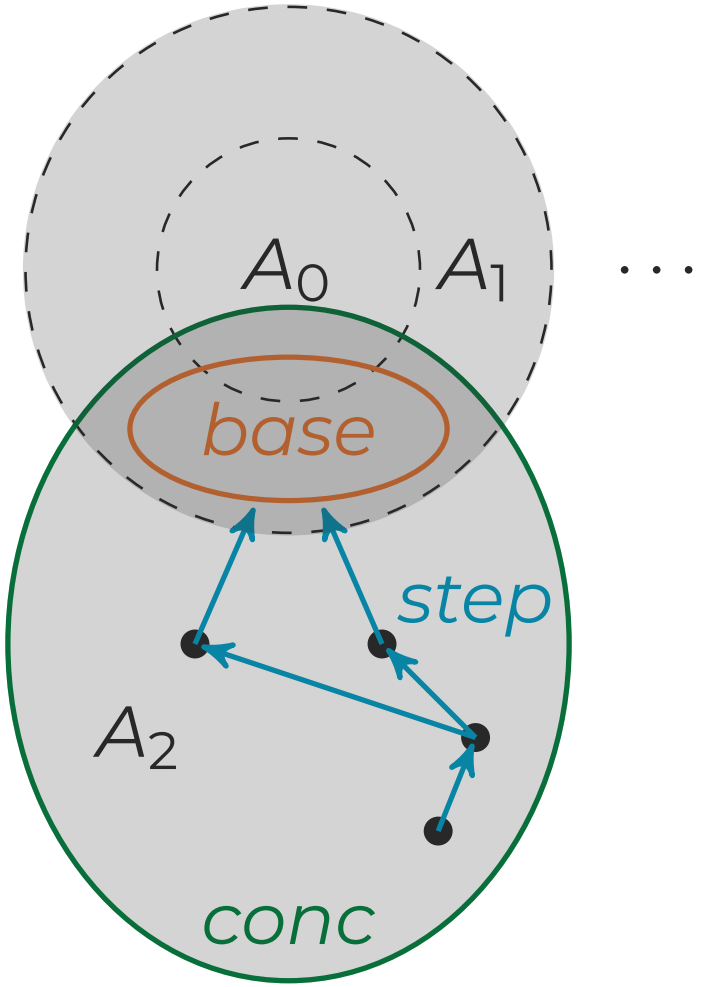
Attractor acceleration

Lemma $(x \leq 42, x' < x, \top)$

- system reaches goal from $x \leq 42$ in l_0
- **system can enforce $x' < x$ from l_0 to l_0**

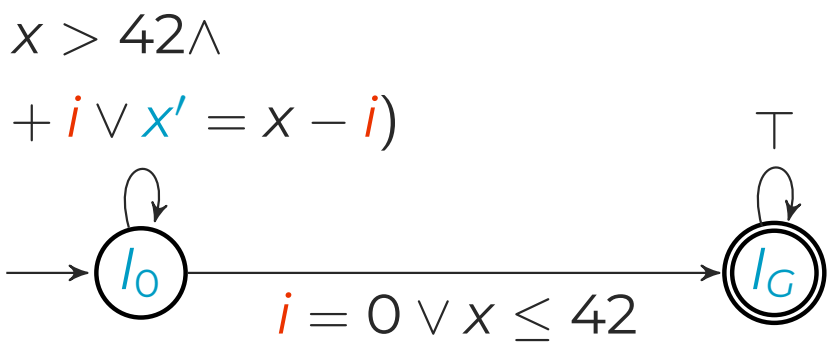
➡ system can reach the goal from $\top$ in l_0

➡ add $\top$ to attractor in l_0



Checking Acceleration Lemmas: Loop Games

To establish: **system can enforce** $step = x' < x$ **from** l_0 **to** l_0

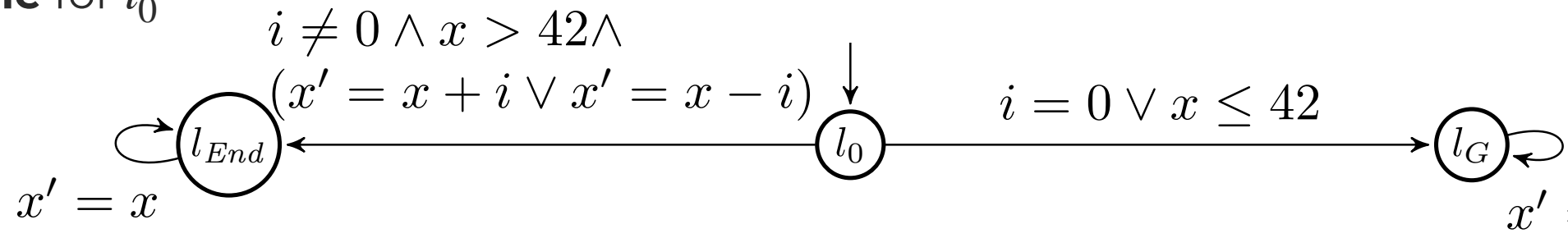


Checking Acceleration Lemmas: Loop Games

To establish: **system can enforce** $step = x' < x$ **from** l_0 **to** l_0

to apply lemma $(base, step, conc)$ at location l_0 when $Attr^1_{Sys} := \{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\}$

loop game for l_0



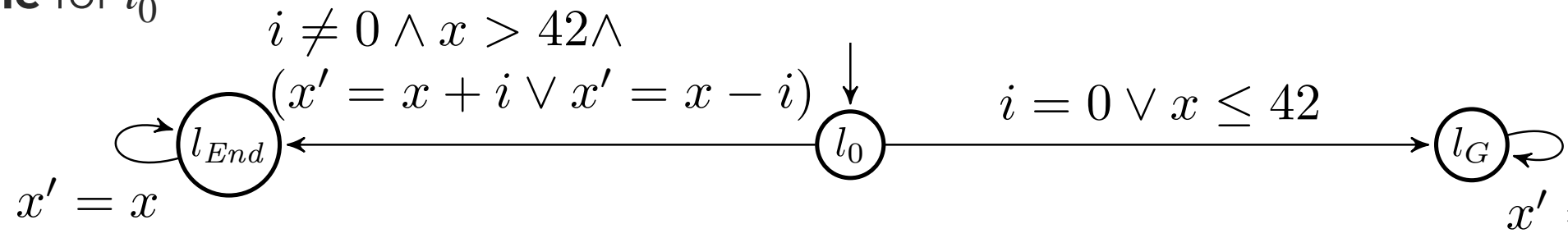
winning condition: reaching l_{End} with $step$ satisfied, or reach $\{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\}$

Checking Acceleration Lemmas: Loop Games

To establish: **system can enforce** $step = x' < x$ **from** l_0 **to** l_0

to apply lemma $(base, step, conc)$ at location l_0 when $Attr^1_{Sys} := \{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\}$

loop game for l_0

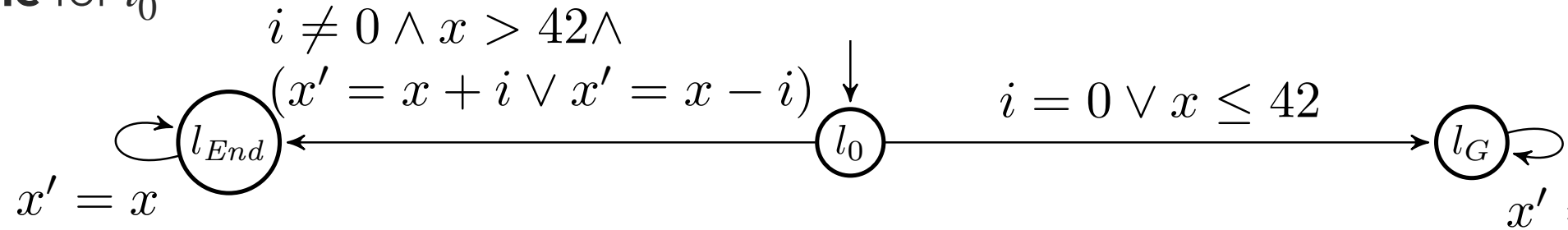


winning condition: reaching l_{End} with $step$ satisfied, or reach $\{l_0 \mapsto x \leq 42,\ l_G \mapsto \top\}$

Checking Acceleration Lemmas: Loop Games

To establish: **system can enforce** $step = x' < x$ **from** l_0 **to** l_0

loop game for l_0

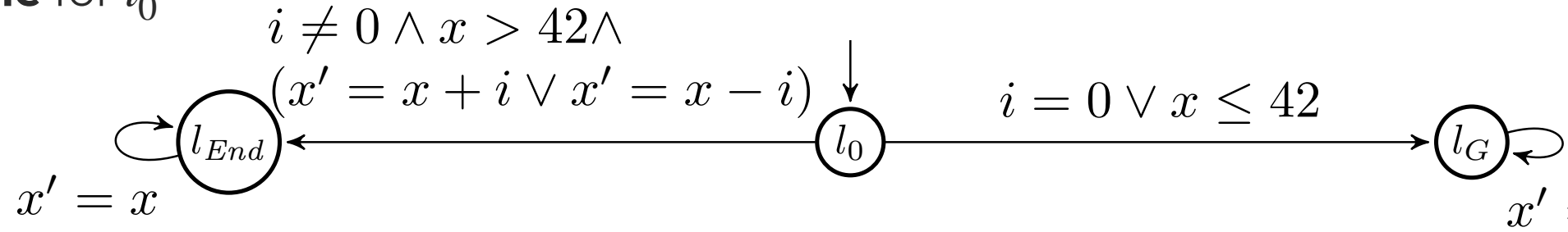


winning condition: reaching l_{End} with $step$ satisfied, or reach $\{ l_0 \mapsto x \leq 42,\ l_G \mapsto \mathsf{T} \}$

Checking Acceleration Lemmas: Loop Games

To establish: **system can enforce** $step = x' < x$ **from** l_0 **to** l_0

loop game for l_0



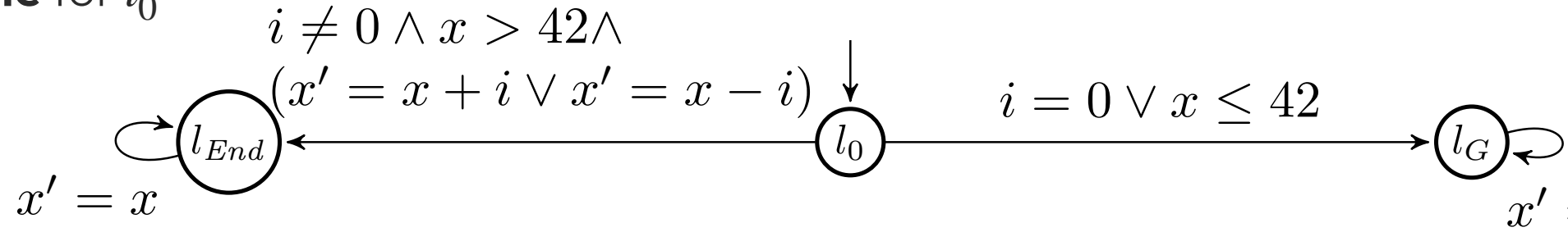
winning condition: reaching l_{End} with $step$ satisfied, or reach $\{l_0 \mapsto x \leq 42,\ l_G \mapsto \mathsf{T}\ \}$

Intuition: strategy in the loop game corresponds to a sub-procedure to enforce $step$.

Checking Acceleration Lemmas: Loop Games

To establish: **system can enforce** $step = x' < x$ **from** l_0 **to** l_0

loop game for l_0



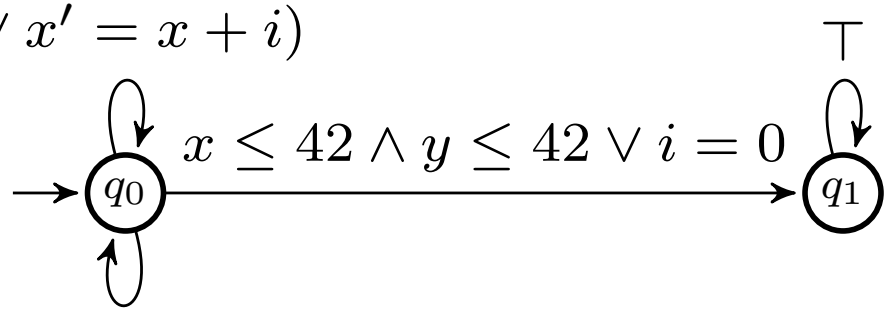
winning condition: reaching l_{End} with $step$ satisfied, or reach $\{l_0 \mapsto x \leq 42,\ l_G \mapsto \mathsf{T}\ \}$

Intuition: strategy in the loop game corresponds to a sub-procedure to enforce $step$.

Solving the loop game might require acceleration. **Nested acceleration needed.**

It Is Not Always “That Simple”

$$i \neq 0 \wedge x > 42 \wedge$$
$$(x' = x - i \vee x' = x + i)$$

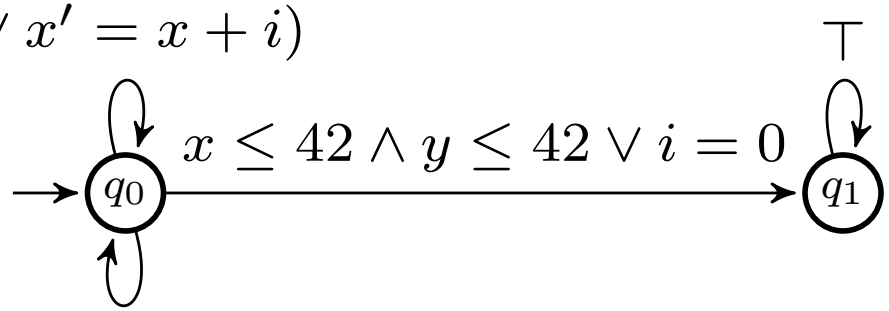


$$i \neq 0 \wedge y > 42 \wedge$$
$$(y' = y - i \vee y' = y + i)$$

Spec: reach q_1 from q_0 for any initial value in x, y

It Is Not Always “That Simple”

$$i \neq 0 \wedge x > 42 \wedge$$
$$(x' = x - i \vee x' = x + i)$$



$$i \neq 0 \wedge y > 42 \wedge$$
$$(y' = y - i \vee y' = y + i)$$

Spec: reach q_1 from q_0 for any initial value in x, y

Apply acceleration twice, for example

- first for x with **invariant** $y \leq 42$
- then for y with invariant $\top$

Computing Acceleration Lemmas

Computing Acceleration Lemmas

Naive approach:

- generate acceleration lemmas based on templates
- if the application is not successful, backtrack

Computing Acceleration Lemmas

Naive approach:

- generate acceleration lemmas based on templates
- if the application is not successful, backtrack

Problem: large search space, nested acceleration relates lemmas

Computing Acceleration Lemmas

Naive approach:

- generate acceleration lemmas based on templates
- if the application is not successful, backtrack

Problem: large search space, nested acceleration relates lemmas

Our solution: **[Heim/Dimitrova, POPL 2024]**

- perform attractor computation using **uninterpreted lemmas**

Computing Acceleration Lemmas

Naive approach:

- generate acceleration lemmas based on templates
- if the application is not successful, backtrack

Problem: large search space, nested acceleration relates lemmas

Our solution: **[Heim/Dimitrova, POPL 2024]**

- perform attractor computation using **uninterpreted lemmas**
- collect **constraints on the unknown lemmas** during computation

Computing Acceleration Lemmas

Naive approach:

- generate acceleration lemmas based on templates
- if the application is not successful, backtrack

Problem: large search space, nested acceleration relates lemmas

Our solution: **[Heim/Dimitrova, POPL 2024]**

- perform attractor computation using **uninterpreted lemmas**
- collect **constraints on the unknown lemmas** during computation
- generate lemmas from **templates** that satisfy the collected constraints

Instantiating Uninterpreted Lemmas

The generated acceleration lemmas must satisfy

- the conditions from the definition of acceleration lemma
- the constraint generated during symbolic acceleration

Instantiating Uninterpreted Lemmas

The generated acceleration lemmas must satisfy

- **the conditions from the definition of acceleration lemma**
- the constraint generated during symbolic acceleration

Search for lemmas of the form

$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

- r is a function term of real or integer type and $\epsilon > 0$ is a constant
- inv is a formula (representing an invariant where the lemma is applicable)

Instantiating Uninterpreted Lemmas

Search for lemmas of the form

$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

Use templates for r and inv.

Use SMT solver to find instantiations that satisfy the constraints.

Instantiating Uninterpreted Lemmas

Search for lemmas of the form

$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

Use templates for r and inv.

Use SMT solver to find instantiations that satisfy the constraints.

Possible template for r

$$r = \sum_x p_x x + d,$$ where x ranges over program variables with a numerical type

$$d \in \mathbb{R}, p_x \in \{-1, 0, 1\}$$

Instantiating Uninterpreted Lemmas

Search for lemmas of the form

$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

Use templates for r and inv.

Use SMT solver to find instantiations that satisfy the constraints.

Possible template for r

$$r = \sum_x p_x x + d, \text{ where } x \text{ ranges over program variables with a numerical type}$$

$$d \in \mathbb{R}, p_x \in \{-1, 0, 1\}$$

Possible template for inv: conjunctions of linear inequalities with B variables

$$\sum_x p_x x \leq d, \text{ where } d \in \mathbb{R}, p_x \in \{-1, 0, 1\} \text{ and } \sum_x |p_x| \leq B$$

Instantiating Uninterpreted Lemmas

Search for lemmas of the form

$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

Use templates for r and inv.

Use SMT solver to find instantiations that satisfy the constraints.

Instantiating Uninterpreted Lemmas

Search for lemmas of the form

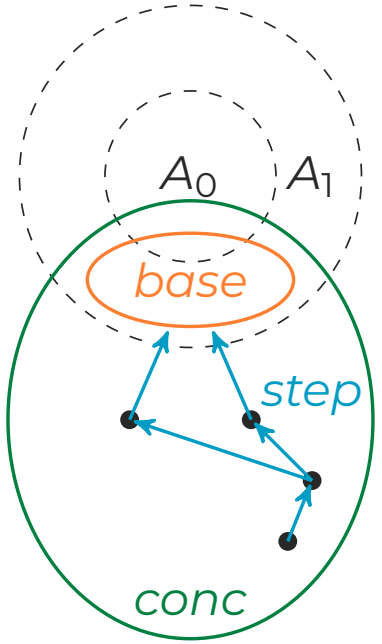
$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

Use templates for r and inv.

Use SMT solver to find instantiations that satisfy the constraints.

Problem: we can end up with $(x \leq 42, x' < x \wedge x \leq 100, x \leq 100)$

which is not very helpful



Instantiating Uninterpreted Lemmas

Search for lemmas of the form

$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

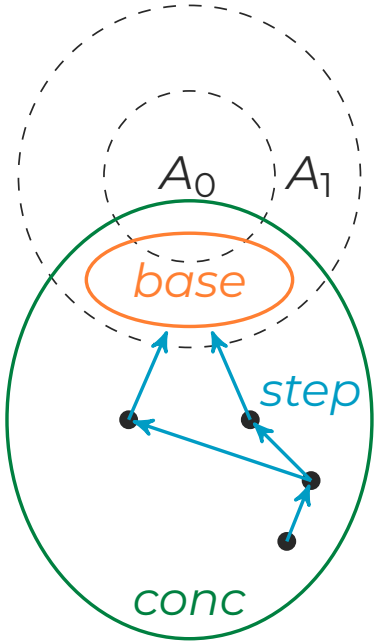
Use templates for r and inv.

Use SMT solver to find instantiations that satisfy the constraints.

Problem: we can end up with $(x \leq 42, x' < x \wedge x \leq 100, x \leq 100)$

which is not very helpful

Approach: apply **quantifier elimination** to obtain general $conc$



Instantiating Uninterpreted Lemmas

Search for lemmas of the form

$$(r(\mathbb{X}) \leq 0 \wedge inv,\ r(\mathbb{X}') \leq r(\mathbb{X}) - \epsilon \wedge inv[\mathbb{X} \mapsto \mathbb{X}'],\ inv)$$

Use templates for r and inv.

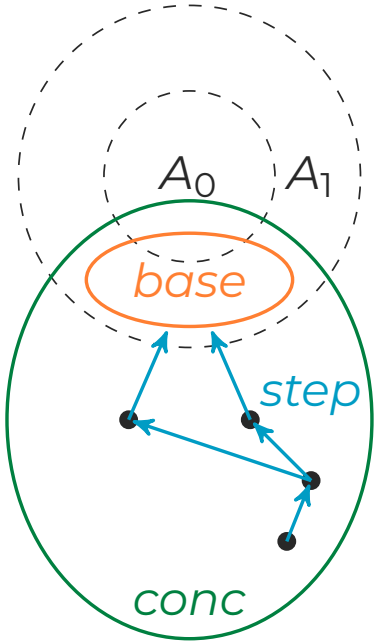
Use SMT solver to find instantiations that satisfy the constraints.

Problem: we can end up with $(x \leq 42, x' < x \wedge x \leq 100, x \leq 100)$

which is not very helpful

Approach: apply **quantifier elimination** to obtain general $conc$

Intuition: find **all states** such that **there exists a lemma**



What About Program Extraction?

What About Program Extraction?

- Classical attractor steps ⟶ book-keeping

What About Program Extraction?

- Classical attractor steps ⟶ book-keeping
- Uninterpreted lemmas and quantifier elimination ⟶ **function synthesis**

What About Program Extraction?

- Classical attractor steps ⟶ book-keeping
- Uninterpreted lemmas and quantifier elimination ⟶ **function synthesis**

Challenges:

What About Program Extraction?

- Classical attractor steps ⟶ book-keeping
- Uninterpreted lemmas and quantifier elimination ⟶ **function synthesis**

Challenges:

- Skolem function computation

What About Program Extraction?

- Classical attractor steps $\longrightarrow$ book-keeping
- Uninterpreted lemmas and quantifier elimination $\longrightarrow$ **function synthesis**

Challenges:

- Skolem function computation
- size of generated formulas $\Longrightarrow$ existing function synthesis tools not applicable in practice

An Alternative Approach Implemented in Issy

An Alternative Approach Implemented in Issy

Geometric acceleration [Heim/Dimitrova, CAV 2025]

An Alternative Approach Implemented in Issy

Geometric acceleration [Heim/Dimitrova, CAV 2025]

- Use high-level information from the game when trying to apply acceleration

An Alternative Approach Implemented in Issy

Geometric acceleration [Heim/Dimitrova, CAV 2025]

- Use high-level information from the game when trying to apply acceleration
- Search for acceleration lemma guided by current subset of the attractor

An Alternative Approach Implemented in Issy

Geometric acceleration [Heim/Dimitrova, CAV 2025]

- Use high-level information from the game when trying to apply acceleration
- Search for acceleration lemma guided by current subset of the attractor
- Analyze variable behavior

An Alternative Approach Implemented in Issy

Geometric acceleration [Heim/Dimitrova, CAV 2025]

- Use high-level information from the game when trying to apply acceleration
- Search for acceleration lemma guided by current subset of the attractor
- Analyze variable behavior

➡ In many cases performs better

An Alternative Approach Implemented in Issy

Geometric acceleration [Heim/Dimitrova, CAV 2025]

- Use high-level information from the game when trying to apply acceleration
- Search for acceleration lemma guided by current subset of the attractor
- Analyze variable behavior

➡ In many cases performs better

➡ Easier to extract implementations

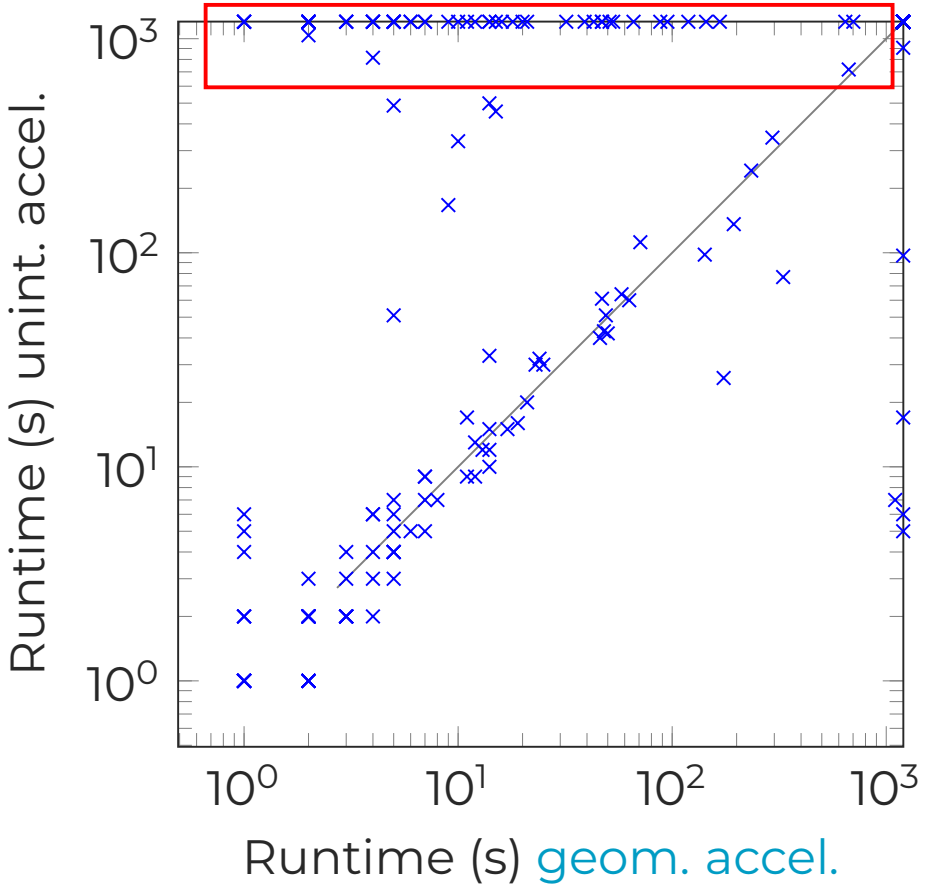
An Alternative Approach Implemented in Issy

Geometric acceleration [Heim/Dimitrova, CAV 2025]

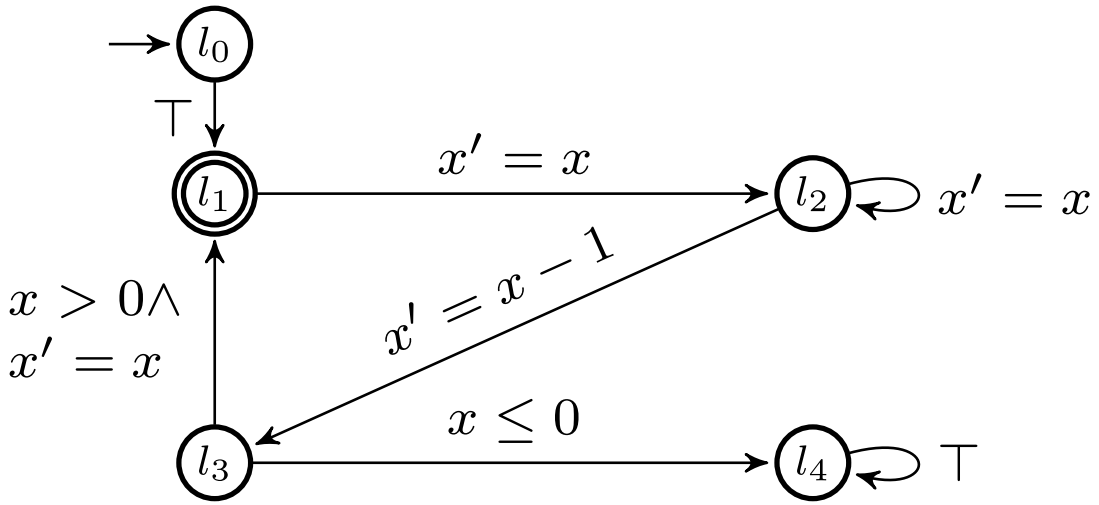
- Use high-level information from the game when trying to apply acceleration
- Search for acceleration lemma guided by current subset of the attractor
- Analyze variable behavior

➡ In many cases performs better

➡ Easier to extract implementations

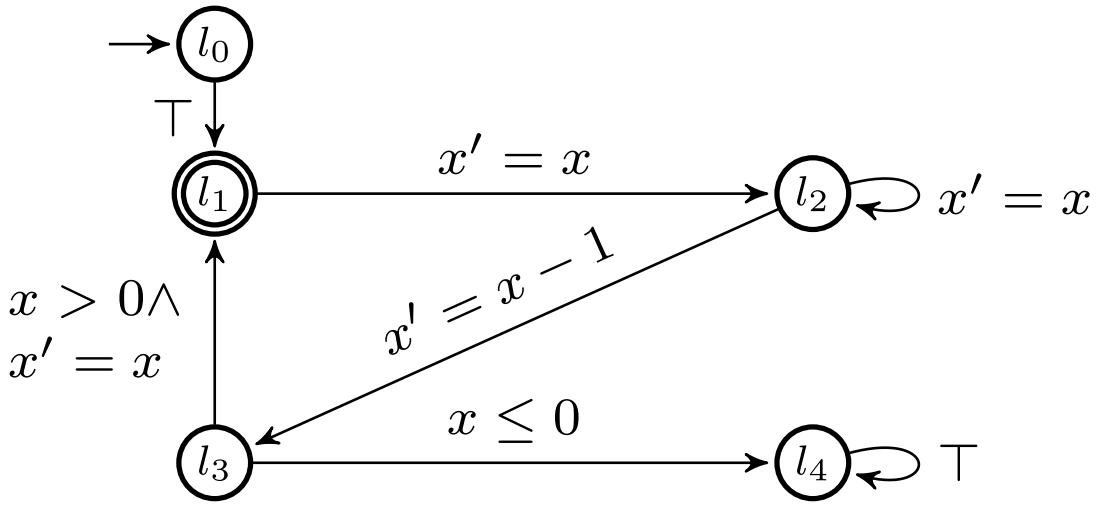


Acceleration Beyond Attractors: Büchi Games



Spec: visit location l_1 infinitely often

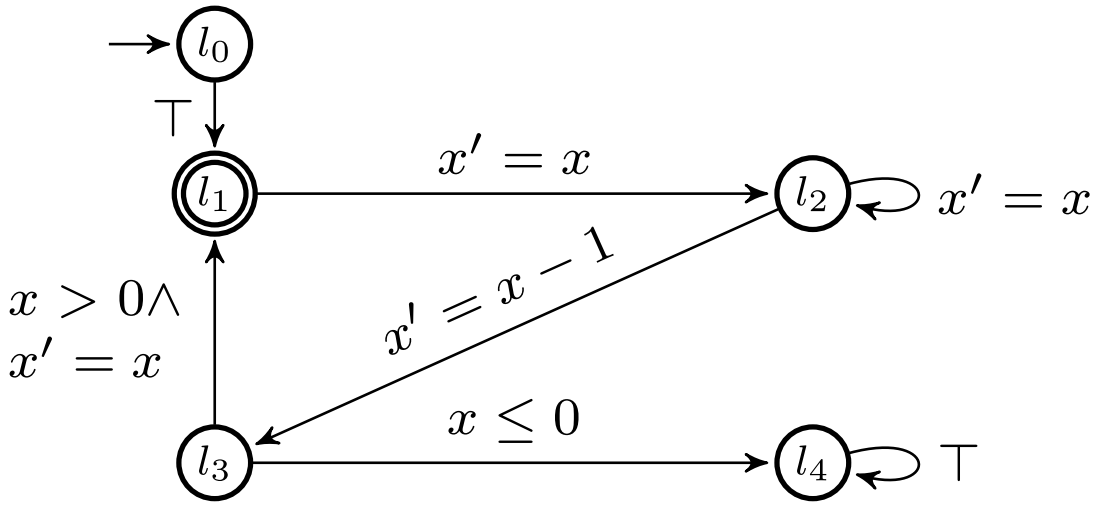
Acceleration Beyond Attractors: Büchi Games



Spec: visit location l_1 infinitely often

The **environment wins** the game for every possible state, since **the value of** x chosen in l_0 **bounds the number of visits to** l_1.

Acceleration Beyond Attractors: Büchi Games

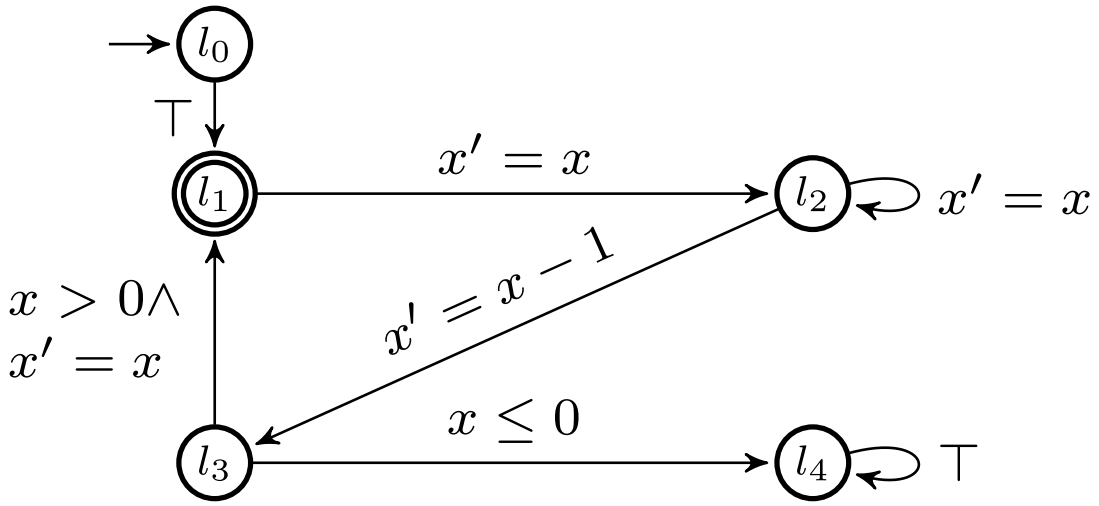


Spec: visit location l_1 infinitely often

The **environment wins** the game for every possible state, since **the value of** x chosen in l_0 **bounds the number of visits to** l_1.

This argument cannot be captured with attractor acceleration, as the environment **cannot force** the system to decrease x.

Acceleration Beyond Attractors: Büchi Games



Player p with Büchi objective

function SolveBüchiGame($\mathcal{G}$, p, f)

1 $f^0 := \lambda l.\ \top$; $f^1 := f$; $w^0_{1-p} := \lambda l.\bot$

2 **for** $n = 1, 2, \ldots$ **do**

3 **if** $f^n \equiv_T f^{n-1}$ **then return** w^{n-1}_{1-p}

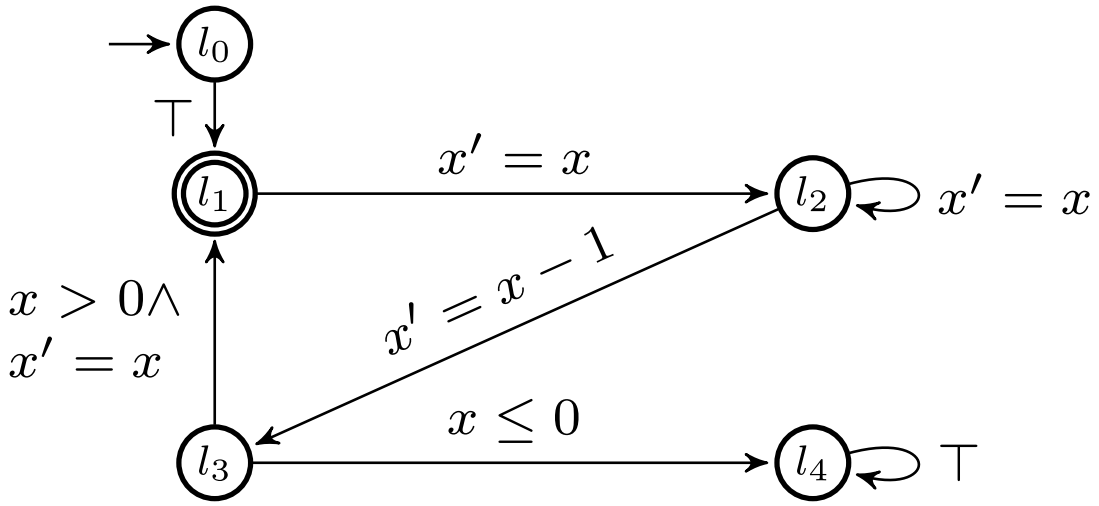
4 $a^n := \text{Attractor}(\mathcal{G}, p, f^n)$

5 $w^n_{1-p} := \text{Attractor}(\mathcal{G}, 1-p, \neg CPre_{\mathcal{G},p}(a^n))$

6 $f^{n+1} := f^n \wedge \neg w^n_{1-p}$

w^n_{1-p}: states from which Player $1-p$ wins with at most n visits to the set of states f

Acceleration Beyond Attractors: Büchi Games



	f^0	a^0	w^1_{1-p}	f^1	a^1	w^2_{1-p}	...
l_1	$\top$	$\top$	$x \leq 1$	$x > 1$	$x > 1$	$x \leq 2$	...
l_2	$\bot$	$x > 1$	$x \leq 1$	$\bot$	$x > 2$	$x \leq 2$	...
l_3	$\bot$	$x > 0$	$x \leq 1$	$\bot$	$x > 1$	$x \leq 2$	...
l_4	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	...

Player p with Büchi objective

function SolveBüchiGame($\mathcal{G}, p, f$)

1 $f^0 := \lambda l.\ \top$; $f^1 := f$; $w^0_{1-p} := \lambda l.\bot$

2 **for** $n = 1, 2, \ldots$ **do**

3 **if** $f^n \equiv_T f^{n-1}$ **then return** w^{n-1}_{1-p}

4 $a^n := \text{Attractor}(\mathcal{G}, p, f^n)$

5 $w^n_{1-p} := \text{Attractor}(\mathcal{G}, 1-p, \neg CPre_{\mathcal{G},p}(a^n))$

6 $f^{n+1} := f^n \wedge \neg w^n_{1-p}$

w^n_{1-p}: states from which Player $1-p$ wins with at most n visits to the set of states f

Acceleration Beyond Attractors: Büchi Games

Acceleration Beyond Attractors: Büchi Games

- In many cases, attractor acceleration is helpful for Büchi and parity games.
- It is clear that we need acceleration methods specialized for those games.

Acceleration Beyond Attractors: Büchi Games

- In many cases, attractor acceleration is helpful for Büchi and parity games.
- It is clear that we need acceleration methods specialized for those games.

co-Büchi acceleration

Acceleration Beyond Attractors: Büchi Games

- In many cases, attractor acceleration is helpful for Büchi and parity games.
- It is clear that we need acceleration methods specialized for those games.

co-Büchi acceleration

- Establish arguments of the form: "If Player p with the Büchi objective keeps visiting a given Büchi accepting location, then the set of winning states of Player $(1-p)$ must be reached eventually".

Acceleration Beyond Attractors: Büchi Games

- In many cases, attractor acceleration is helpful for Büchi and parity games.
- It is clear that we need acceleration methods specialized for those games.

co-Büchi acceleration

- Establish arguments of the form: "If Player p with the Büchi objective keeps visiting a given Büchi accepting location, then the set of winning states of Player $(1-p)$ must be reached eventually".
- "Reaching the winning states eventually" is captured by the acceleration lemmas.

Acceleration Beyond Attractors: Büchi Games

- In many cases, attractor acceleration is helpful for Büchi and parity games.
- It is clear that we need acceleration methods specialized for those games.

co-Büchi acceleration

- Establish arguments of the form: "If Player p with the Büchi objective keeps visiting a given Büchi accepting location, then the set of winning states of Player $(1-p)$ must be reached eventually".
- "Reaching the winning states eventually" is captured by the acceleration lemmas.
- In contrast to attractor acceleration, we do not use that the step relation is enforceable by a player, but that it is unavoidable by Player p upon revisiting the considered accepting location.

Acceleration Beyond Attractors: Büchi Games

- In many cases, attractor acceleration is helpful for Büchi and parity games.
- It is clear that we need acceleration methods specialized for those games.

co-Büchi acceleration

- Establish arguments of the form: "If Player p with the Büchi objective keeps visiting a given Büchi accepting location, then the set of winning states of Player $(1-p)$ must be reached eventually".
- "Reaching the winning states eventually" is captured by the acceleration lemmas.
- In contrast to attractor acceleration, we do not use that the step relation is enforceable by a player, but that it is unavoidable by Player p upon revisiting the considered accepting location.
- The loop games have Büchi winning conditions.

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations
- **Symbolic methods enhanced with logical reasoning**
  - for solving realizability/synthesis games
  - **for translation of temporal logic formulas**
  - **combining symbolic methods and abstraction**

Reactive Program LTL (RP-LTL)

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi\mathbf{U}\varphi$$

where α is a quantifier-free formula over **inputs, current and next-state variables**

Formulas α are quantifier-free first-order formulas over

- state variables $\mathbb{X}$,
- input variables $\mathbb{I}$, controlled by the environment
- next-step state variables $\mathbb{X}'$, controlled by the system

Synthesis from RP-LTL formulas: translate to a symbolic game and solve the game

Direct Translation via LTL: Example

$$\mathbf{G}\ (x' = x + 1)\ \wedge\ \mathbf{GF}\ (x > 0)$$

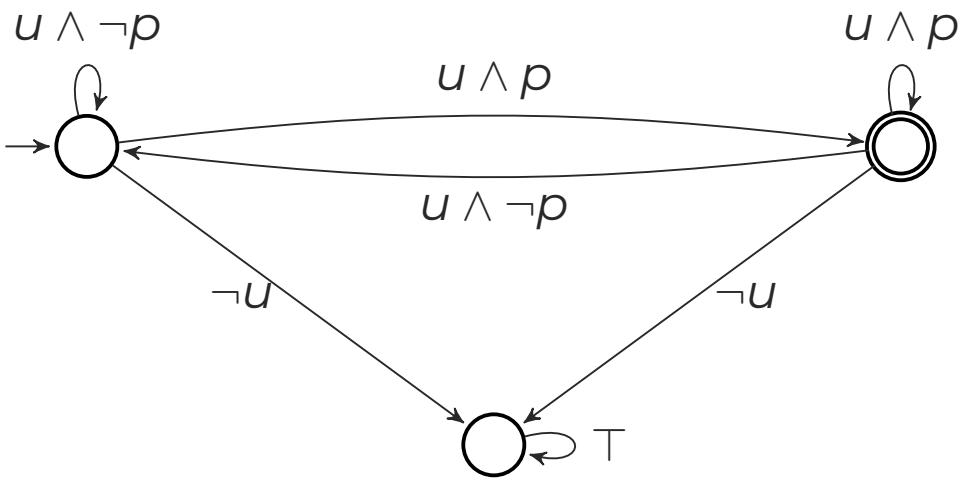
Step 1: Propositionalize to LTL formula

$$\mathbf{G}\ u\ \wedge\ \mathbf{GF}\ p$$

Direct Translation via LTL: Example

$$\mathbf{G}\, u \;\wedge\; \mathbf{GF}\, p$$

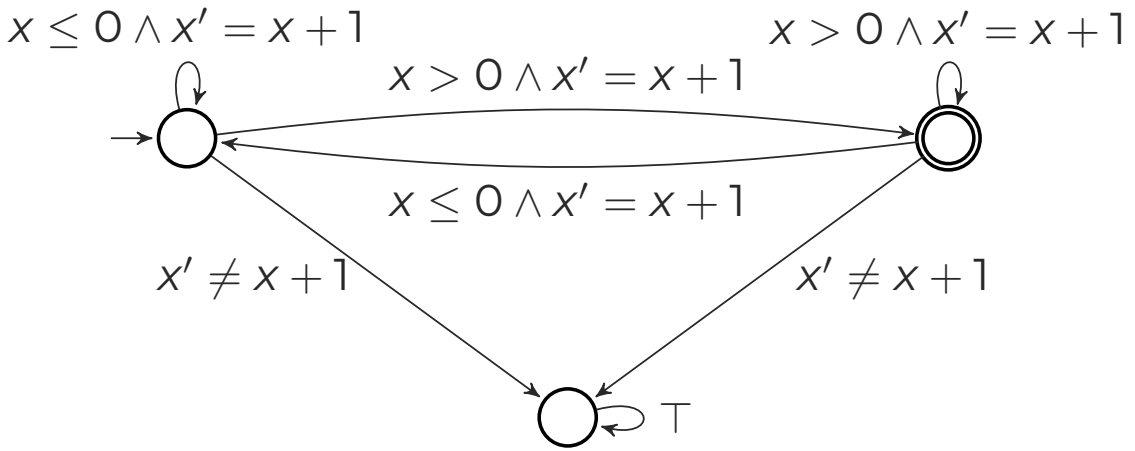
Step 2: Construct deterministic automaton for LTL formula



Direct Translation via LTL: Example

$$\mathbf{G}\,(x' = x + 1) \;\wedge\; \mathbf{GF}\,(x > 0)$$

Step 3: Transform back and construct game

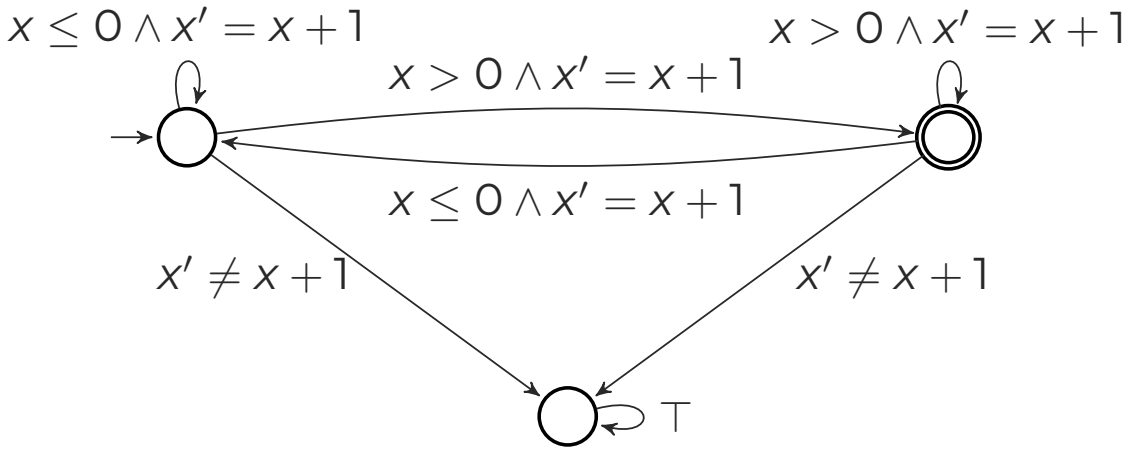


(In this example, a one-player game)

Direct Translation via LTL: Example

$$\mathbf{G}\ (x' = x + 1)\ \wedge\ \mathbf{GF}\ (x > 0)$$

Step 3: Transform back and construct game



(In this example, a one-player game)

⇒Lose high-level information!

Direct Translation: Problem

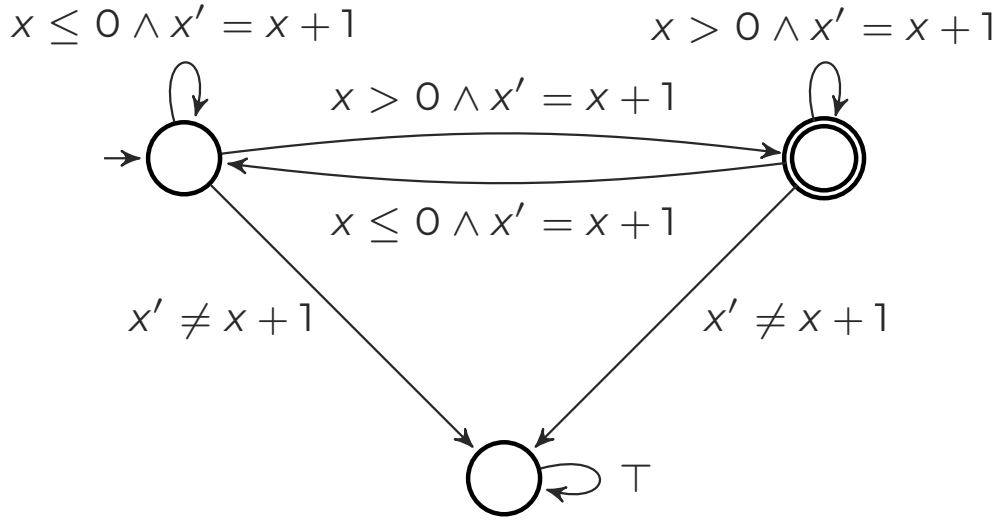
$$\mathbf{G}(x' = x + 1) \ \wedge \ \mathbf{GF}(x > 0)$$

Reasoning on formula ...

- $\mathbf{G}(x' = x + 1)$ implies $\mathbf{F}(x > 0)$
- $\mathbf{G}(x' = x + 1), x > 0$ imply $\mathbf{G}(x > 0)$
- So $\mathbf{G}(x' = x + 1)$ implies $\mathbf{GF}(x > 0)$

Simplify to $\mathbf{G}(x' = x + 1)$

...not clear in game

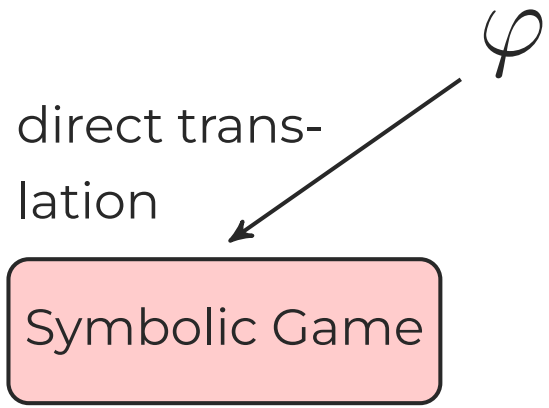


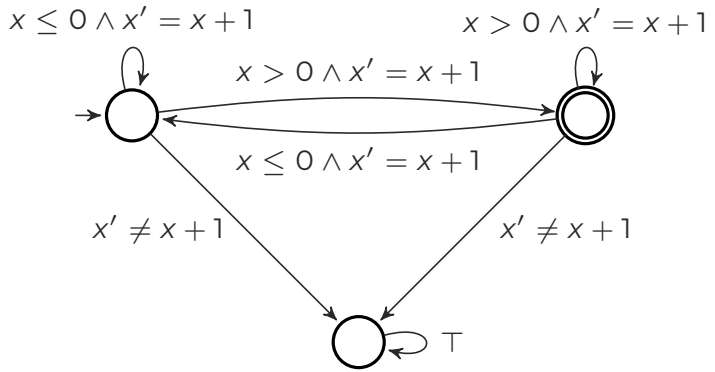
$\Rightarrow$ How to apply to games?

Method Overview

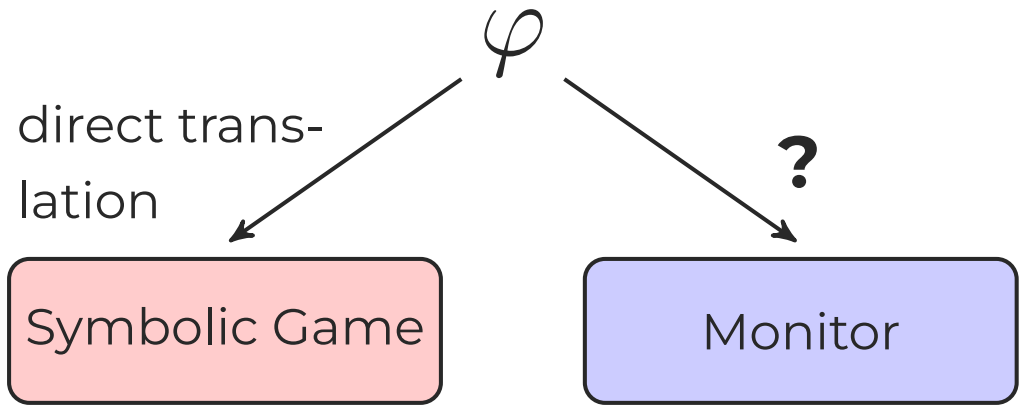
φ

Method Overview

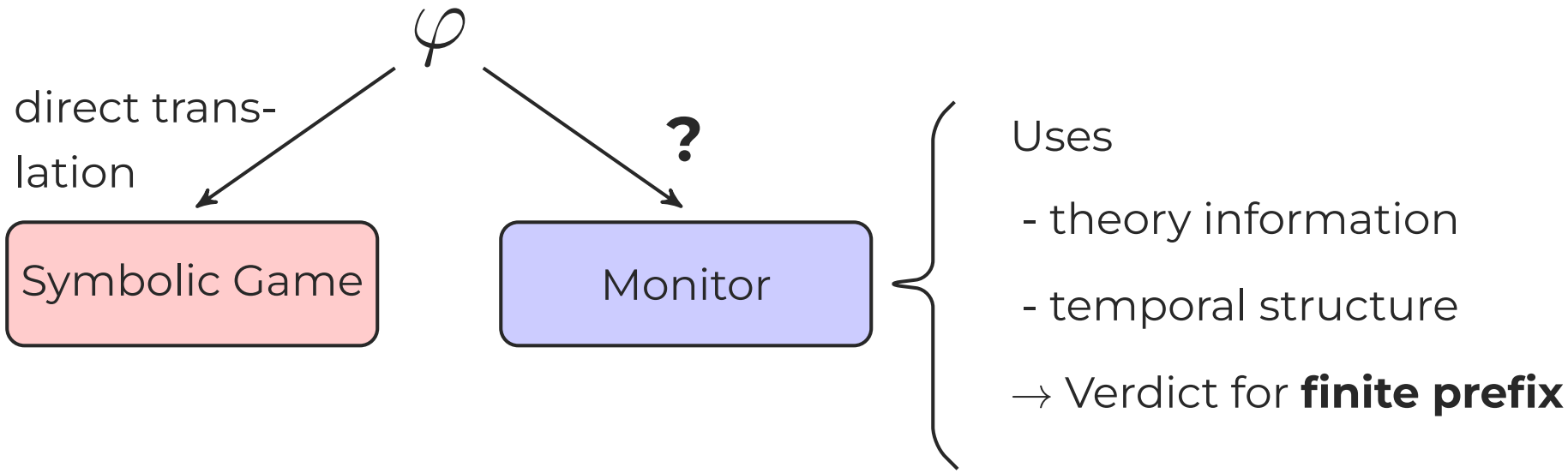




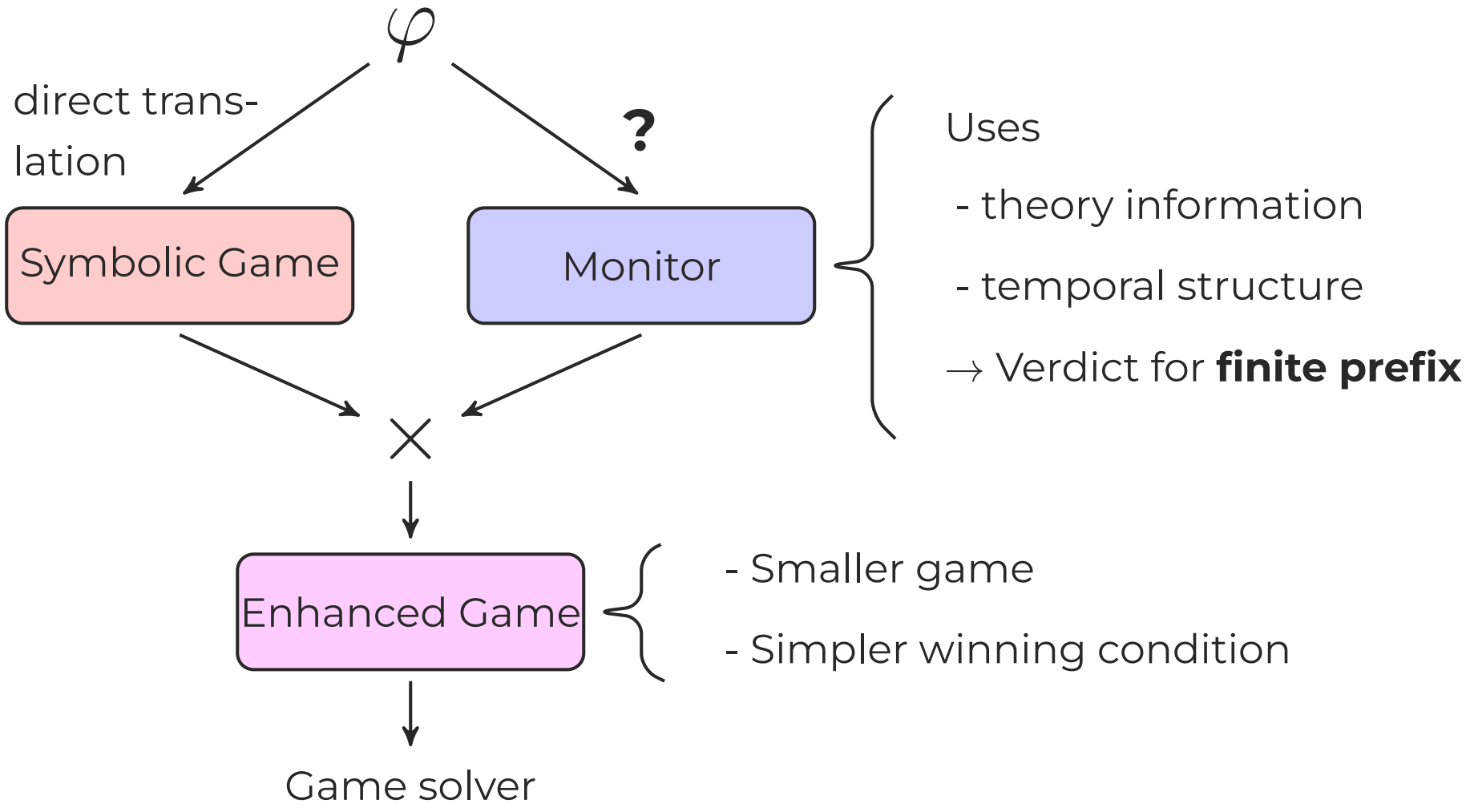
Method Overview



Method Overview



Method Overview



Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Step 1: Expansion & Rules

$\Phi_1 = \{d' = d\}$

$$\boxed{d > 0 \rightarrow \mathbf{G}\Phi_1 \wedge \mathbf{X}(\dots)}$$

Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Step 1: Expansion & Rules

$\Phi_1 = \{d' = d\}$

$\Phi_2 = \Phi_1 \cup \{d > 0\}$

$$d > 0 \rightarrow \mathbf{G}\Phi_2 \wedge \mathbf{X}(\dots)$$

Rule: Invariant Generation

$$\mathbf{G}(d' = d), d > 0 \implies \mathbf{G}(d > 0)$$ (SMT query)

Rule: Invariant Generation

$$\textbf{(Gen-Inv)}\ \frac{\alpha \in QF(\mathbb{X}) \qquad \theta \models_T \alpha \quad \theta \in QF(\mathbb{X}) \qquad \gamma \models_T \theta \quad \gamma \in QF(\mathbb{X} \cup \mathbb{I} \cup \mathbb{X}') \qquad \theta \wedge ImpInv_D(q) \models_T \theta[\mathbb{X} \mapsto \mathbb{X}']}{Imp'_D := Imp_D \cup \{\mathbf{G}(\gamma \rightarrow \mathbf{G}\alpha)\}}$$

Intuition:

Whenever γ is satisfied, then all sequences of valuations of the state variables that satisfy the implied invariants must also satisfy α at every point from the current step onwards.

➡ We can add $\mathbf{G}(\gamma \rightarrow \mathbf{G}\alpha)$ to the implied invariants.

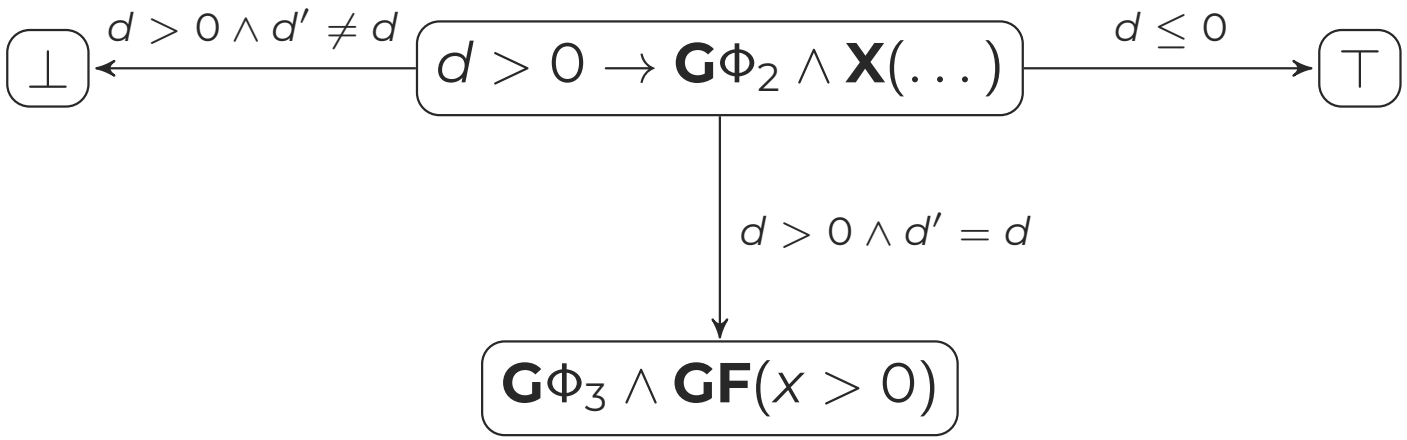
θ is a strengthening of α, which plays the role of an inductive invariant.

Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Step 1: Expansion & Rules

$$\Phi_1 = \{d' = d\}$$
$$\Phi_2 = \Phi_1 \cup \{d > 0\}$$
$$\Phi_3 = \Phi_2 \cup \{x' = x + d\}$$

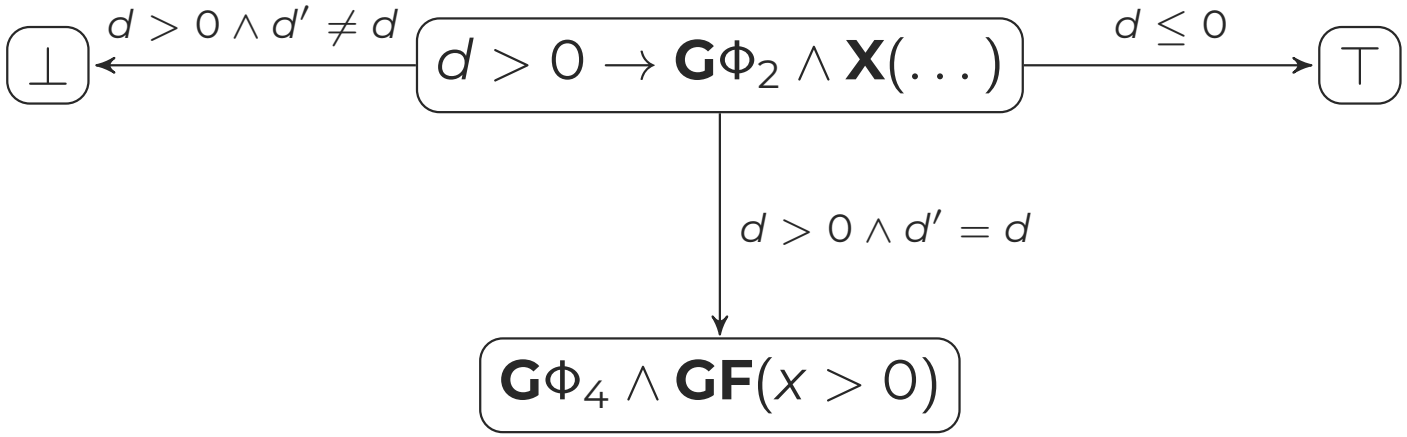


Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Step 1: Expansion & Rules

$$\Phi_1 = \{d' = d\}$$
$$\Phi_2 = \Phi_1 \cup \{d > 0\}$$
$$\Phi_3 = \Phi_2 \cup \{x' = x + d\}$$
$$\Phi_4 = \Phi_3 \cup \{\mathbf{F}(x > 0)\}$$



Rule: Liveness derivation

$$\mathbf{G}(d > 0), \mathbf{G}(x' = x + d) \implies \mathbf{F}(x > 0) \quad \text{(Fixpoint query)}$$

Rule: Liveness Derivation

$$\textbf{(Gen-Reach)}\ \frac{\gamma, \beta \in QF(\mathbb{X}) \qquad \gamma \models_T \mu B.\beta \vee \forall \mathbb{I} \forall \mathbb{X}'.ImpInv_D(q) \rightarrow B(\mathbb{X}')}{Imp'_D := Imp_D \cup \{\mathbf{G}(\gamma \rightarrow \mathbf{F}\beta)\}}$$

Intuition:

- This rule generates implied reachability properties.
- We provide a formula β and the rule computes as a least fixpoint a formula γ representing all the valuations from which all sequences that satisfy the implied invariants eventually satisfy β.

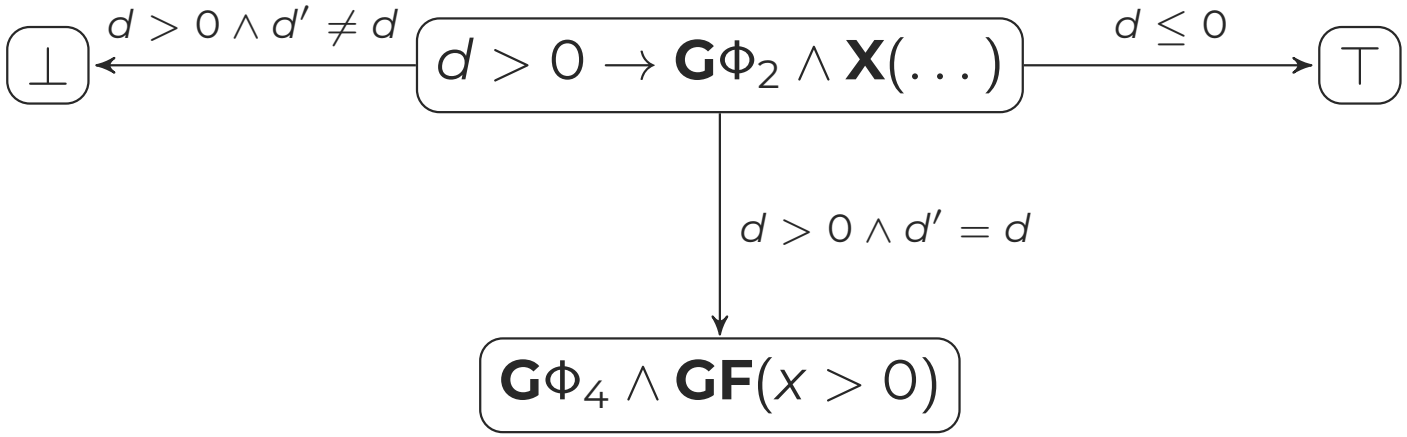
➡ We can add $\mathbf{G}(\gamma \rightarrow \mathbf{F}\beta)$ to the implied invariants.

Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Step 1: Expansion & Rules

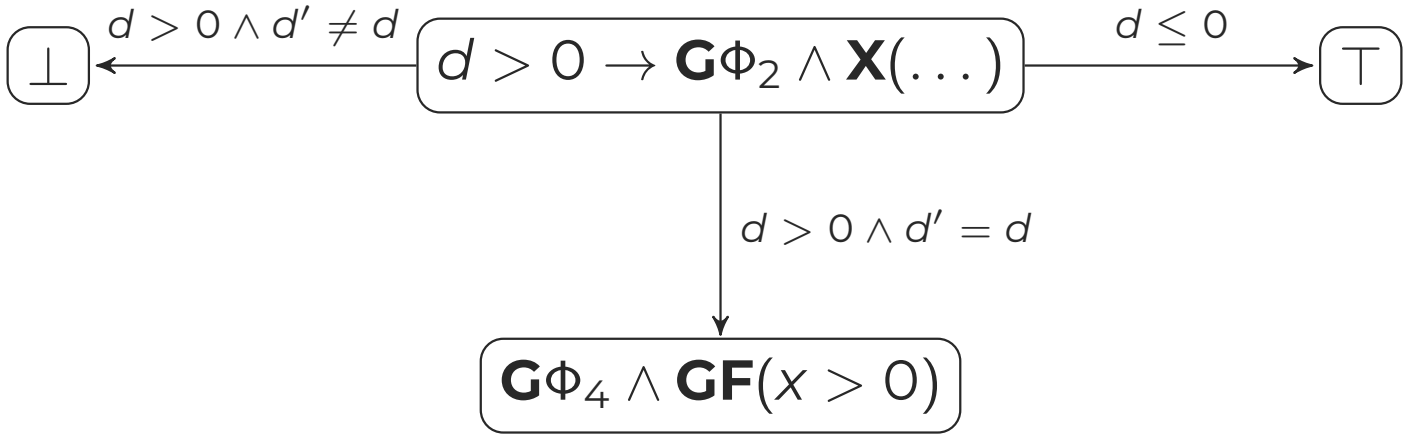
$$\Phi_1 = \{d' = d\}$$
$$\Phi_2 = \Phi_1 \cup \{d > 0\}$$
$$\Phi_3 = \Phi_2 \cup \{x' = x + d\}$$
$$\Phi_4 = \Phi_3 \cup \{\mathbf{F}(x > 0)\}$$



Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

$$\Phi_1 = \{d' = d\}$$
$$\Phi_2 = \Phi_1 \cup \{d > 0\}$$
$$\Phi_3 = \Phi_2 \cup \{x' = x + d\}$$
$$\Phi_4 = \Phi_3 \cup \{\mathbf{F}(x > 0)\}$$



Rule: Subsumption

$$\mathbf{G}\Phi_4 \implies \mathbf{GF}(x > 0)$$

Example: Monitor Construction

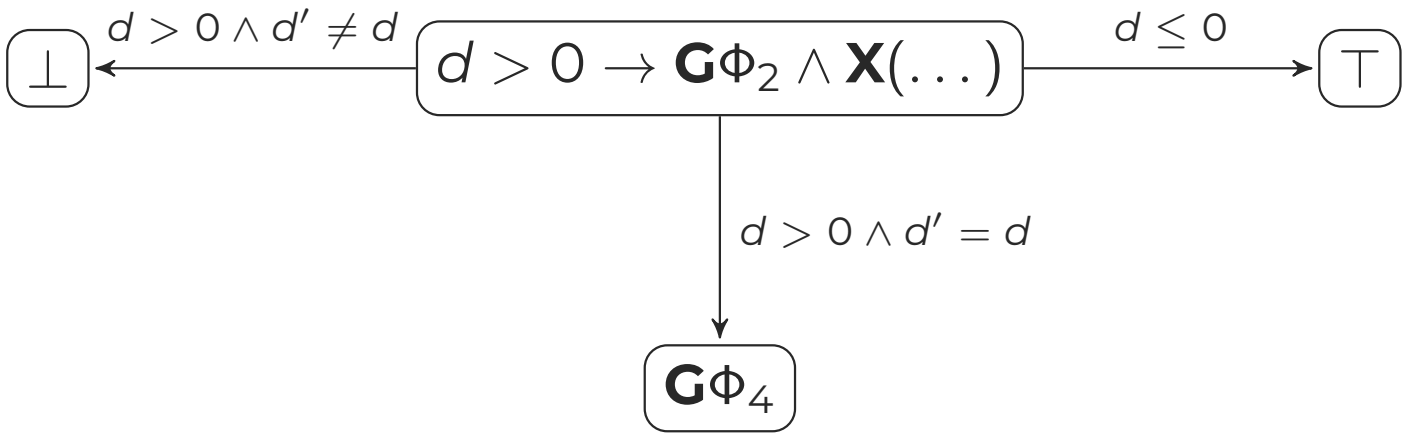
$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

$\Phi_1 = \{d' = d\}$

$\Phi_2 = \Phi_1 \cup \{d > 0\}$

$\Phi_3 = \Phi_2 \cup \{x' = x + d\}$

$\Phi_4 = \Phi_3 \cup \{\mathbf{F}(x > 0)\}$



Example: Monitor Construction

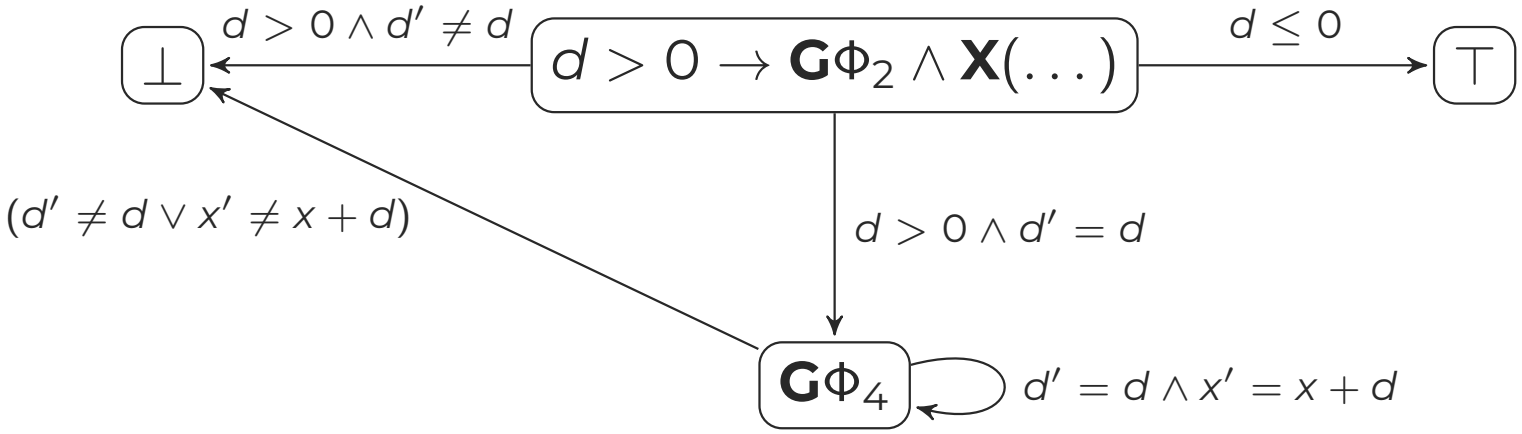
$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

$\Phi_1 = \{d' = d\}$

$\Phi_2 = \Phi_1 \cup \{d > 0\}$

$\Phi_3 = \Phi_2 \cup \{x' = x + d\}$

$\Phi_4 = \Phi_3 \cup \{\mathbf{F}(x > 0)\}$

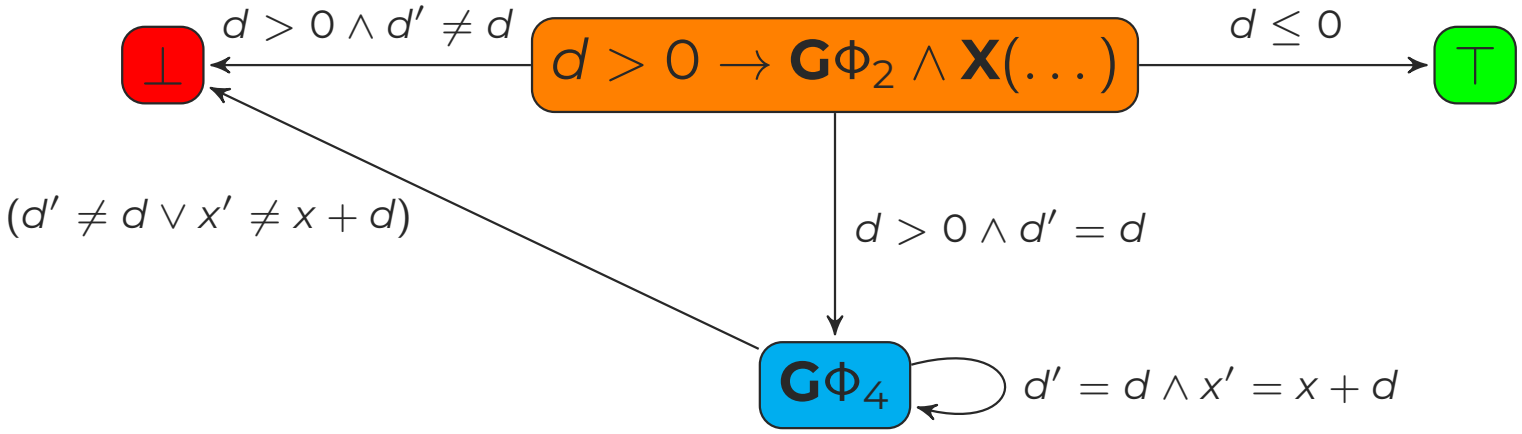


Example: Monitor Construction

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Step 2: Verdict Assignment

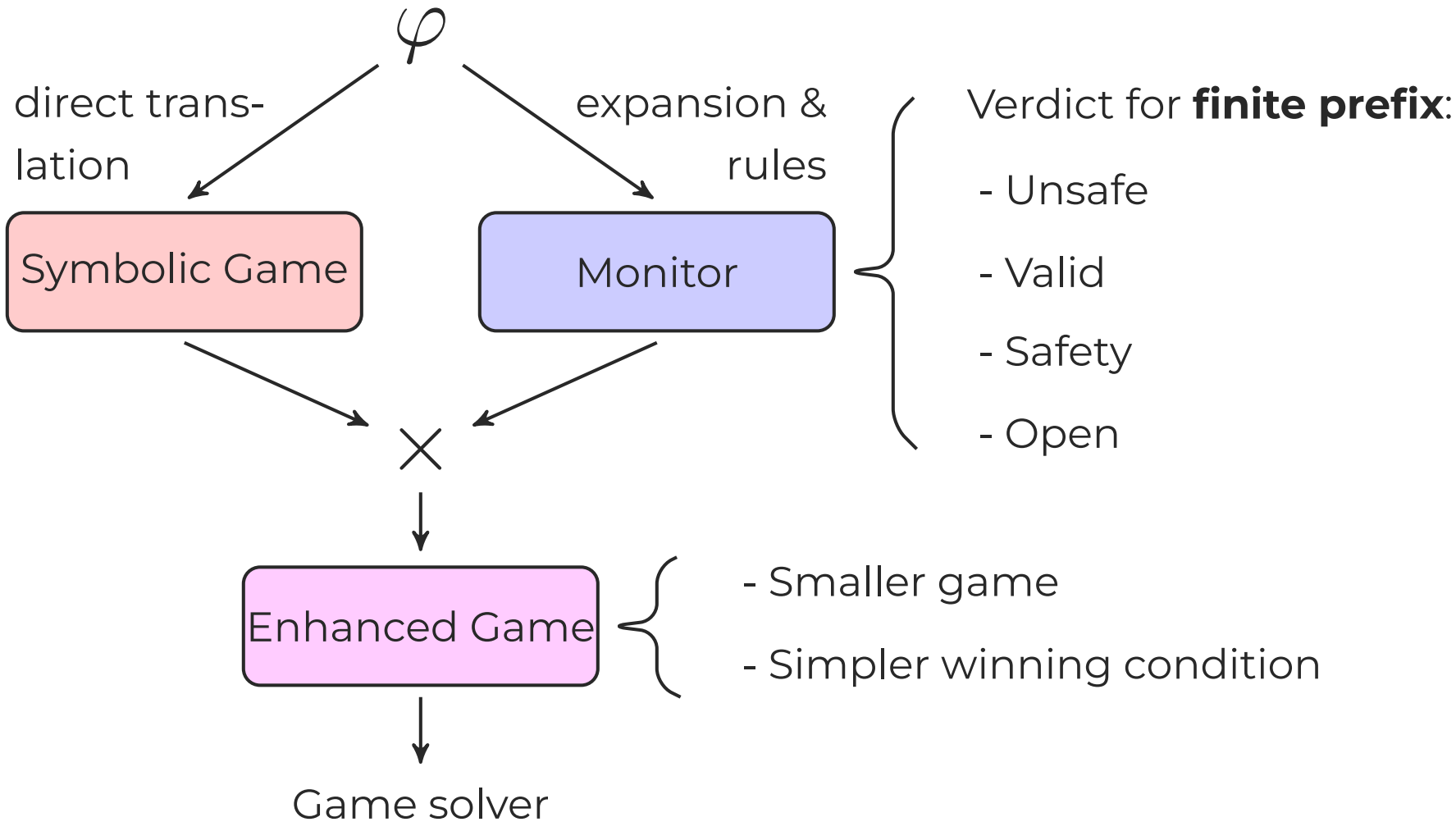
$$\Phi_1 = \{d' = d\}$$
$$\Phi_2 = \Phi_1 \cup \{d > 0\}$$
$$\Phi_3 = \Phi_2 \cup \{x' = x + d\}$$
$$\Phi_4 = \Phi_3 \cup \{\mathbf{F}(x > 0)\}$$



Verdicts:

Unsafe, Valid, Safety, Open

Method Overview

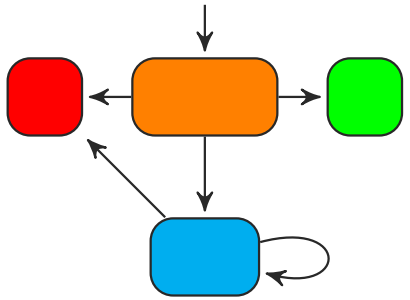


Example: Overall Method

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

Example: Overall Method

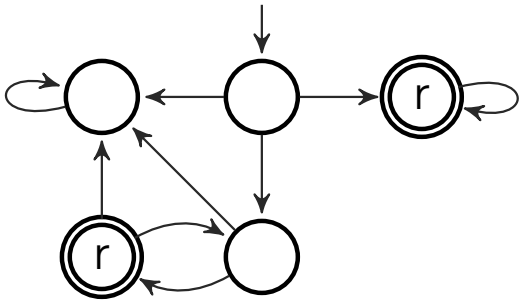
$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$



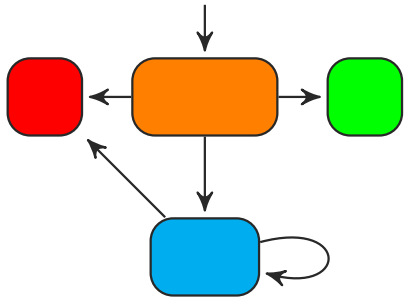
Once in Safety stay there

Example: Overall Method

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$



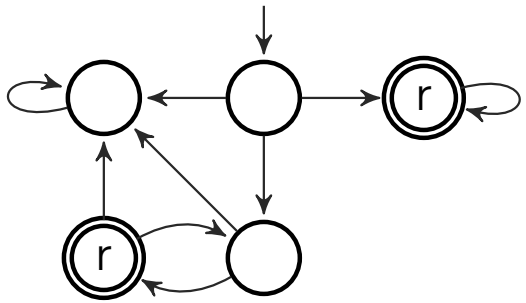
Winning condition: **GF***r*



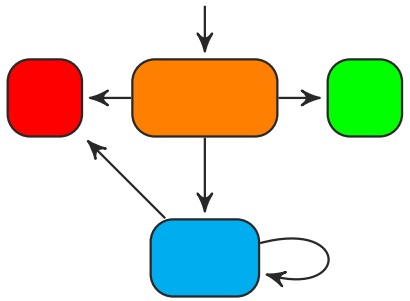
Once in Safety stay there

Example: Overall Method

$$d > 0 \rightarrow \mathbf{G}(d' = d) \wedge \mathbf{X}(\mathbf{G}(x' = x + d) \wedge \mathbf{GF}x > 0)$$

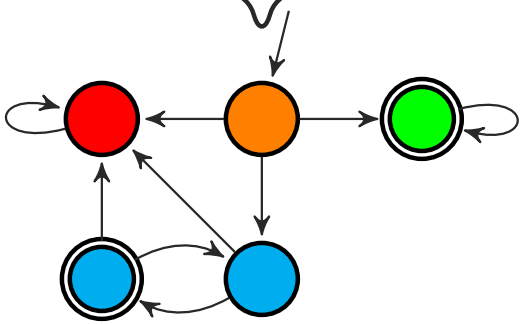


$\times$



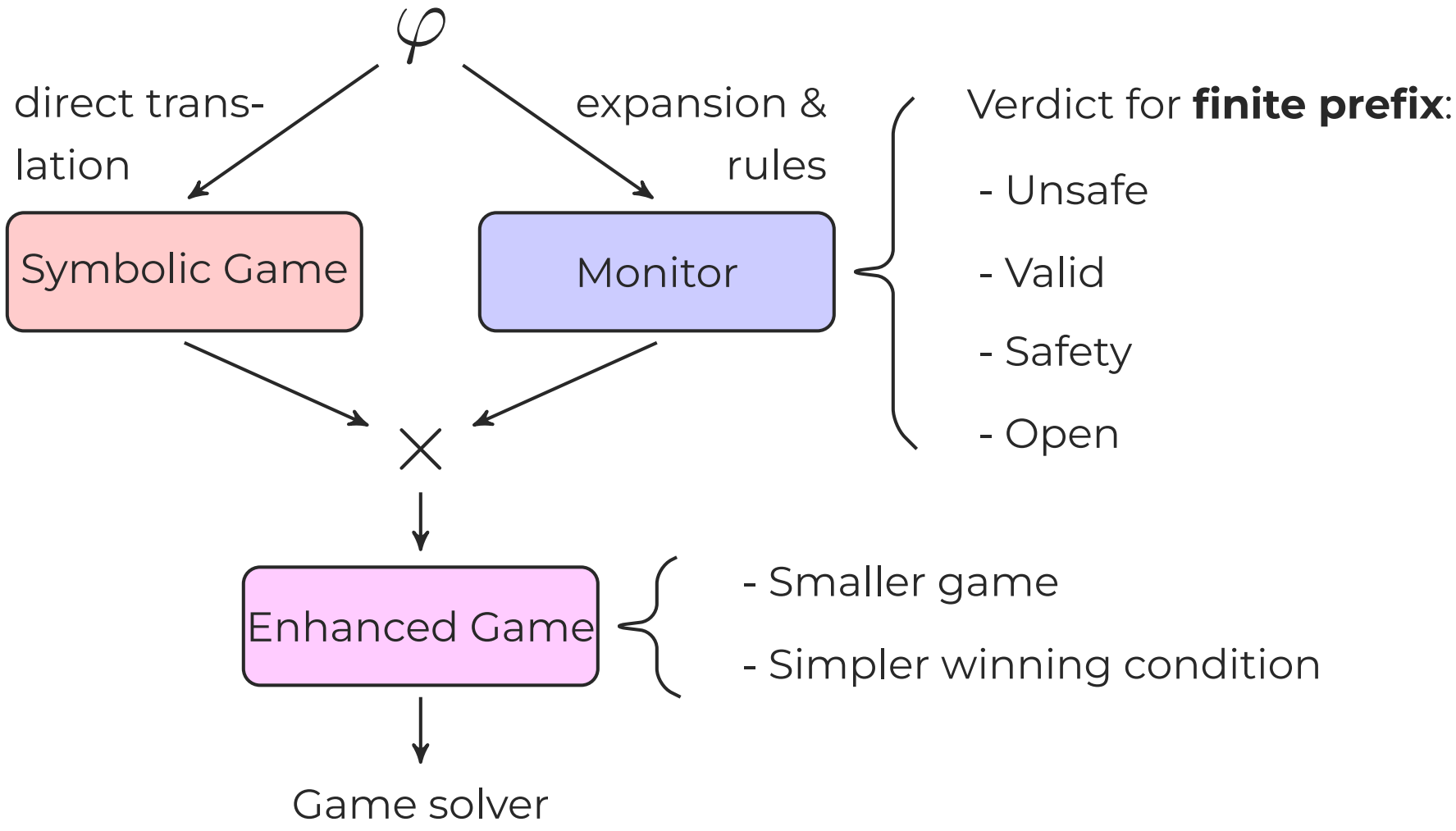
Winning condition: **GF***r*

Once in Safety stay there



$\Rightarrow$ Easier safety winning condition

Method Overview



When Can This Be Helpful?

When Can This Be Helpful?

- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate

When Can This Be Helpful?

- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate
- Detecting **vacuous requirements** — can help with avoiding the need to accelerate

When Can This Be Helpful?

- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate
- Detecting **vacuous requirements** — can help with avoiding the need to accelerate
- **Winning condition simplification** (also locally) — safety games are easier

When Can This Be Helpful?

- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate
- Detecting **vacuous requirements** — can help with avoiding the need to accelerate
- **Winning condition simplification** (also locally) — safety games are easier

Limitations:

When Can This Be Helpful?

- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate
- Detecting **vacuous requirements** — can help with avoiding the need to accelerate
- **Winning condition simplification** (also locally) — safety games are easier

Limitations:

- The symbolic game (essentially the DPA) is still constructed naively

When Can This Be Helpful?

- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate
- Detecting **vacuous requirements** — can help with avoiding the need to accelerate
- **Winning condition simplification** (also locally) — safety games are easier

Limitations:

- The symbolic game (essentially the DPA) is still constructed naively
- No reasoning about strategies

When Can This Be Helpful?

- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate
- Detecting **vacuous requirements** — can help with avoiding the need to accelerate
- **Winning condition simplification** (also locally) — safety games are easier

Limitations:

- The symbolic game (essentially the DPA) is still constructed naively
- No reasoning about strategies
- Computing invariants can be costly

When Can This Be Helpful?

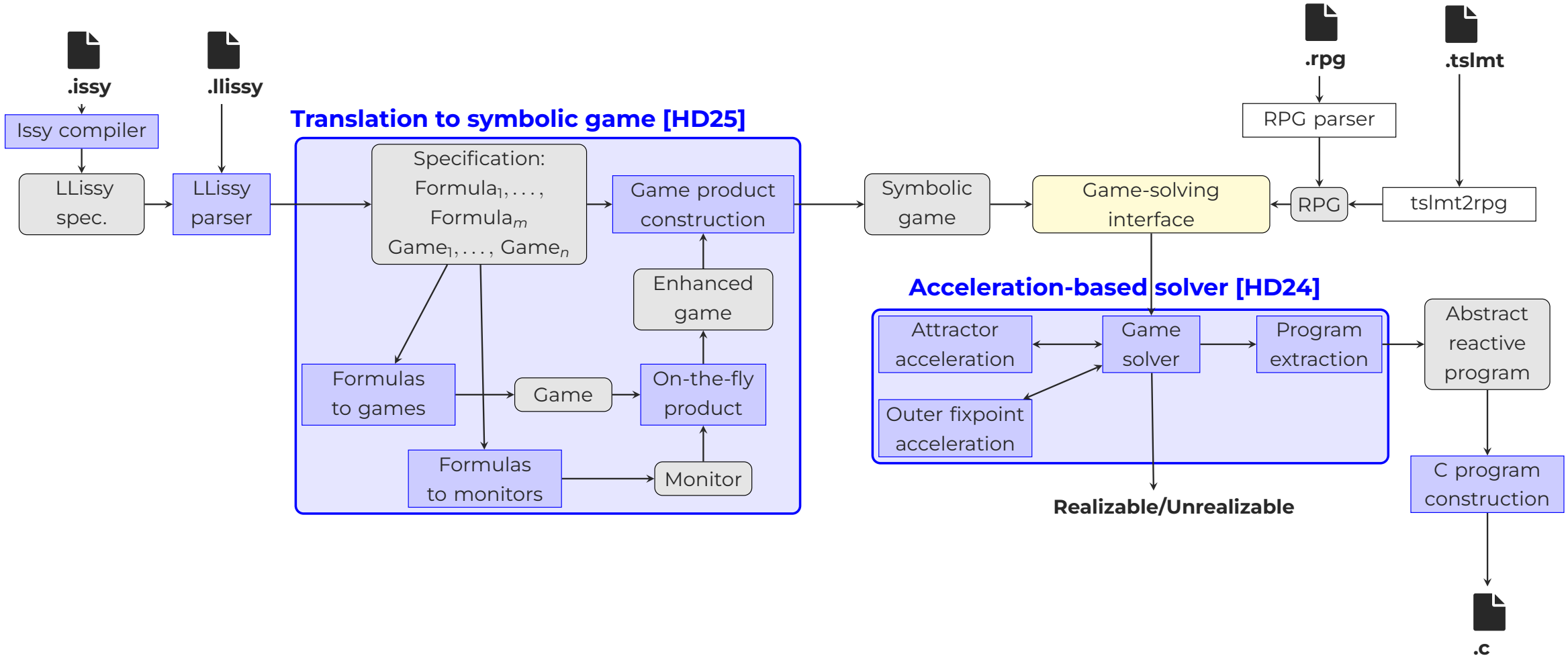
- Detecting **unsatisfiable specifications** — can help with avoiding the need to accelerate
- Detecting **vacuous requirements** — can help with avoiding the need to accelerate
- **Winning condition simplification** (also locally) — safety games are easier

Limitations:

- The symbolic game (essentially the DPA) is still constructed naively
- No reasoning about strategies
- Computing invariants can be costly
- Size of the game can grow after building the product with the monitor

The Issy Framework

github.com/phheim/issy



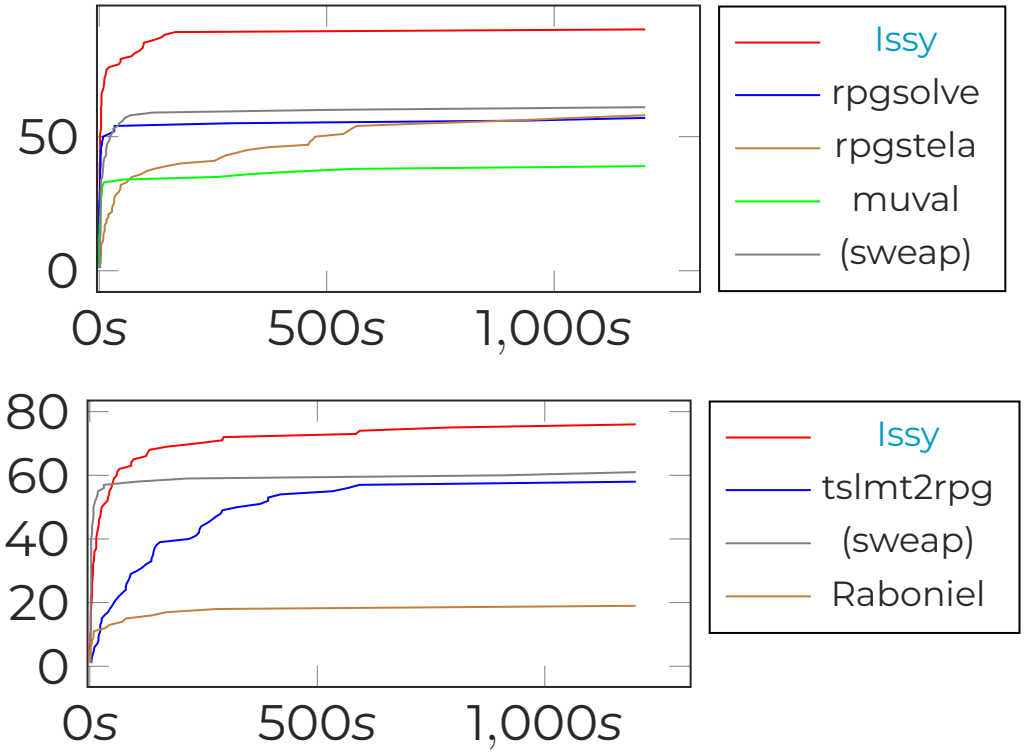
SOTA 2025: Techniques and Tools

Tool	Specification Type	SR	B	LTL	Inf Inp	Inf Out	$\mathbb{B}$	$\mathbb{Z}$	$\mathbb{R}$	Input Format
Issy	Combined games & RP-LTL	✓	✓	✓	✓	✓	✓	✓	✓	Issy, LLissy, RPG, TSL-MT
rpgsolve [HD24]	RPG	✓	✓	✗	✓	✗	✓	✓	✓	RPG
rpg-STeLA [Sch+24]	RPG	✓	✓	✗	✓	✗	✓	✓	✓	RPG
tslmt2rpg [HD25] + [HD24]	TSL-MT	✓	✓	✓	✓	✗	✓	✓	✓	TSL-MT var.
sweap [Azz+25]	Programs + LTL	✓	✓	✓	✗	✗	✓	✓	✗	custom format
Raboniel [MB22]	TSL-MT	✓	✓	✓	✓	✗	✗	✓	✓	TSL-MT var.
temos [Cho+22]	TSL-MT	✓	✓	✓	✓	✗	✓	✓	✗	TSL-MT var.
GenSys [SDK21]	Games	✓	✗	✗	✓	✓	✓	✓	✓	—
GenSys-LTL [SDK23]	Games	✓	✓	✓	✓	✓	✓	✓	✓	—
gr1mT [MWB24]	GR(1) + data	✓	✓	✗	✓	✗	✓	✓	✓	unkown
tools in [RS23; RS24; RGS24]	LTL$_\mathcal{T}$	✓	✓	✓	✓	✓	✓	✓	✓	unkown
MuVal [Unn+23]	μCLP formulas	✓	✓	✓	✓	✓	✗	✓	✓	custom format
SimSynth [FK18]	linear arith. games	✓	✗	✗	✓	✓	✓	✓	✓	custom format

SOTA 2025: Techniques and Tools

Tool	Technique	Unb. Loop	Synthesis	LTL Synt.	SMT	LTL to Aut	Open-Source
Issy	acceleration-based f.p. computation	✓	✓(C code)		•	•	✓
rpgsolve [HD24]	acceleration-based f.p. computation	✓	✓		•		✓
rpg-STeLA [Sch+24]	[HD24] + abstraction	✓	✗		•		✓
tslmt2rpg [HD25] + rpgsolve	monitor-enhanced symb. game constr.	✓	✓		•	•	✓
sweap [Azz+25]	abstraction to LTL	✓	✓	•	•		✓
Raboniel [MB22]	abstraction to LTL	✗	✓(Python)	•	•		✓
temos [Cho+22]	abstraction to LTL	✗	✓(several)	•	•		✓
GenSys [SDK21]	naive f.p. comp.	✗	✓		•		✓
GenSys-LTL [SDK23]	naive f.p. comp.	✗	✓		•	•	✓
gr1mT [MWB24]	GR(1) f.p. comp.	✗	✓		•		✗
tool in [RS23]	abstraction to LTL	—	✗	•	•		✗
tool in [RS24]	abstraction to LTL	—	✓	•	•		✗
tool in [RGS24]	abstraction to LTL + Skolem fun. syn.	—	✓(C code)	•	•		✗
MuVal [Unn+23]	constraint solving	✓	✗		•		✓
SimSynth [FK18]	constraint solving	—	✓		•		✓

SOTA 2025: Experimental Evaluation



Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations
- **Symbolic methods enhanced with logical reasoning**
  - for solving realizability/synthesis games
  - for translation of temporal logic formulas
  - **combining symbolic methods and abstraction**

Attractor Acceleration: Challenges

Finding acceleration lemmas [Heim/Dimitrova, POPL 2024]

- uninterpreted lemmas → uninterpreted functions → constraints
- generate lemmas from grammar/templates with SMT solver

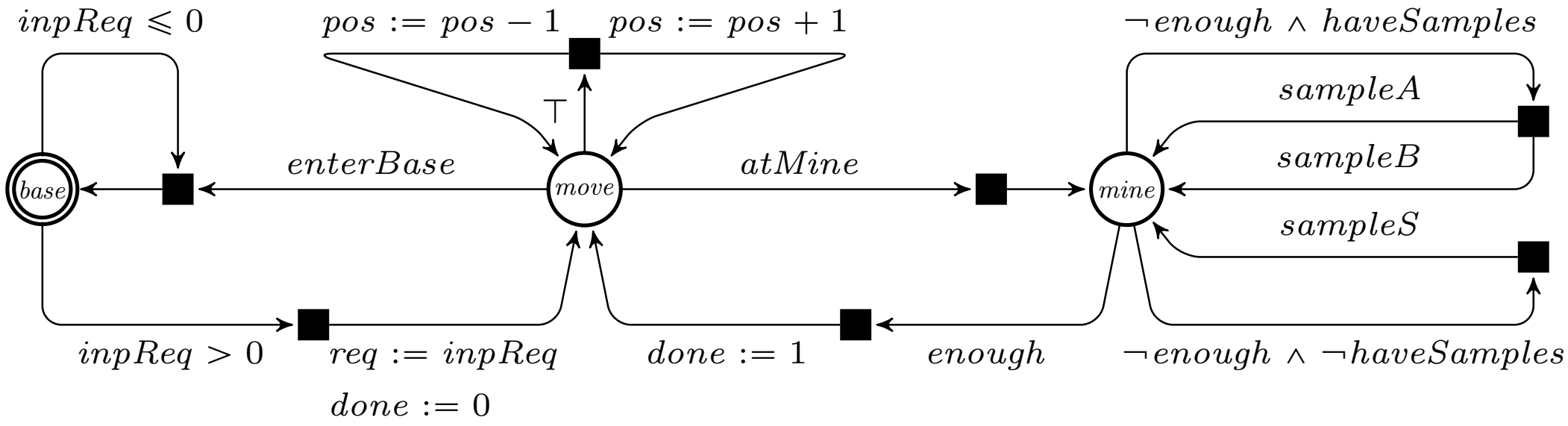
➡ **constraints become complex as the size of the games increases**

Attractor Acceleration: Challenges

Finding acceleration lemmas [Heim/Dimitrova, POPL 2024]

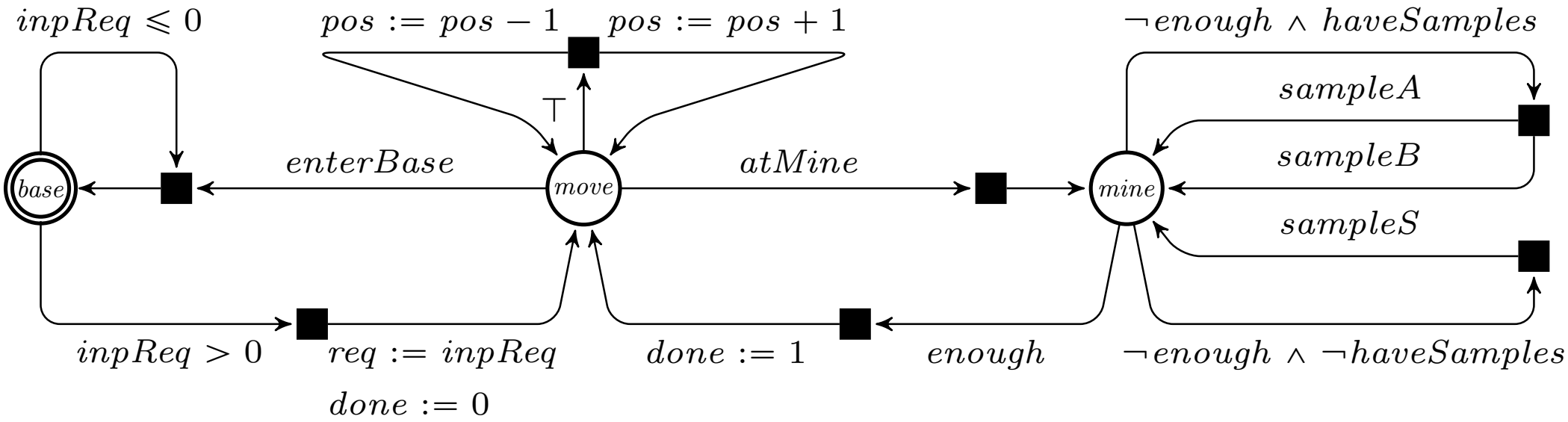
- uninterpreted lemmas → uninterpreted functions → constraints
- generate lemmas from grammar/templates with SMT solver

➡ **constraints become complex as the size of the games increases**



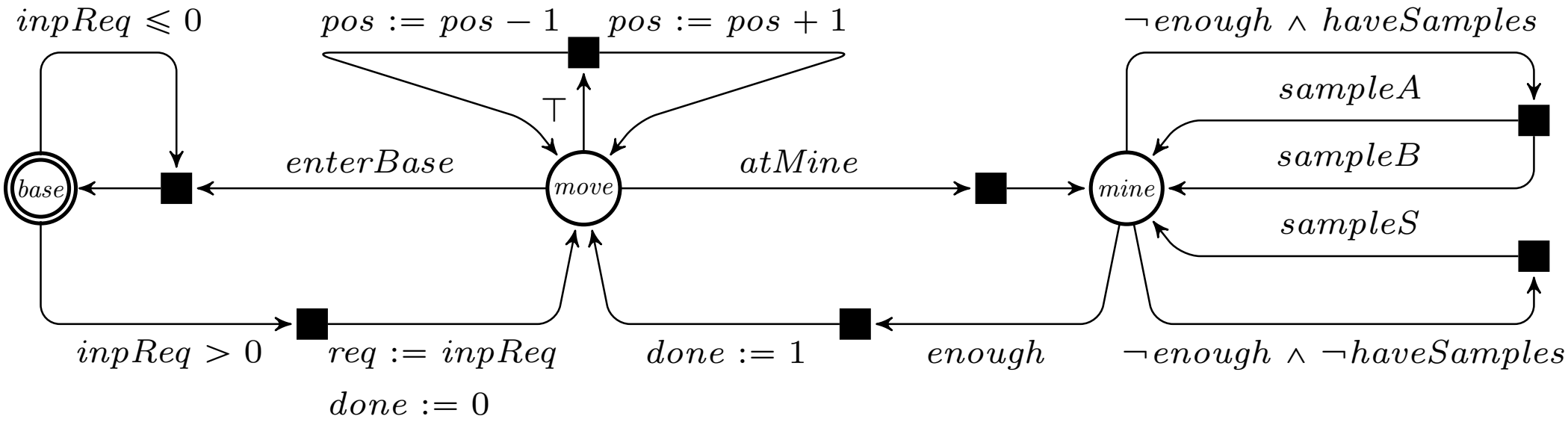
Reactive Program Games: Example

[Heim/Dimitrova, POPL 2024]



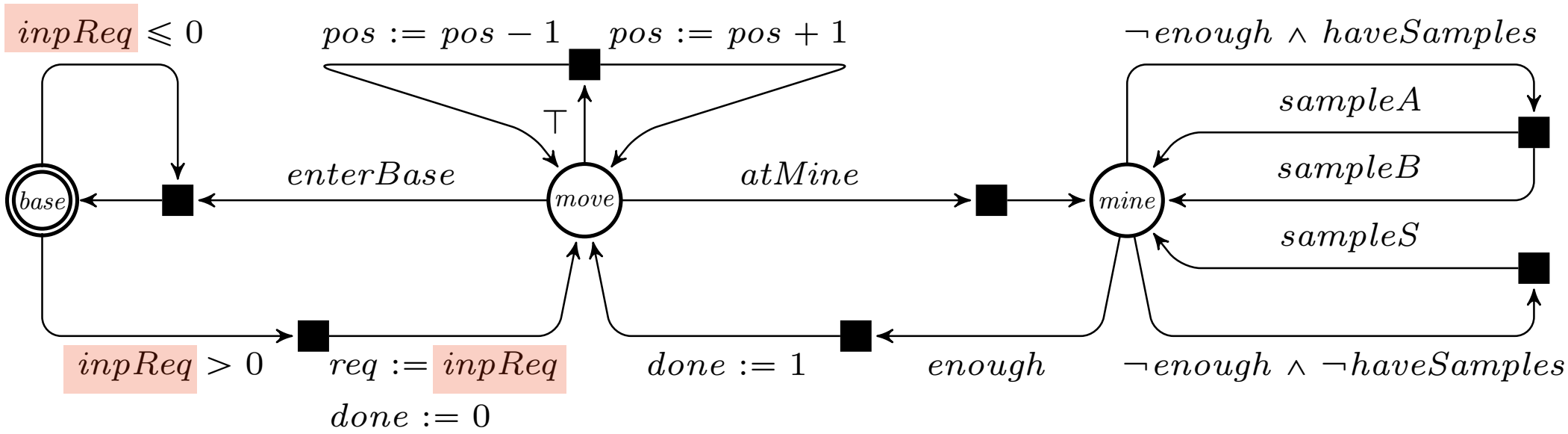
Reactive Program Games: Example

[Heim/Dimitrova, POPL 2024]



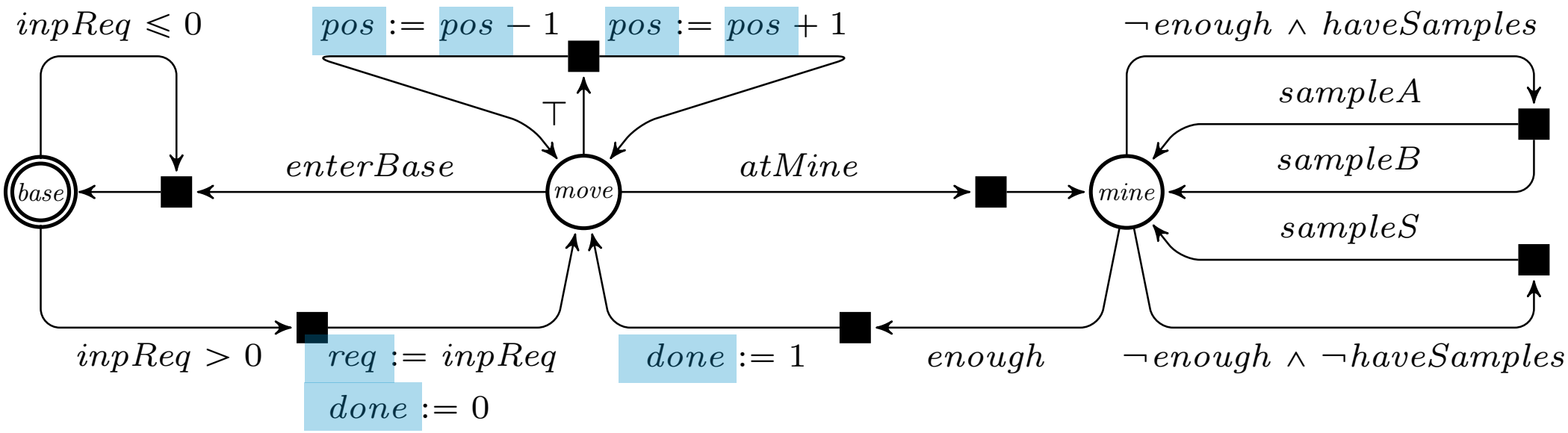
- Locations $\{base, move, mine\}$

Reactive Program Games: Example



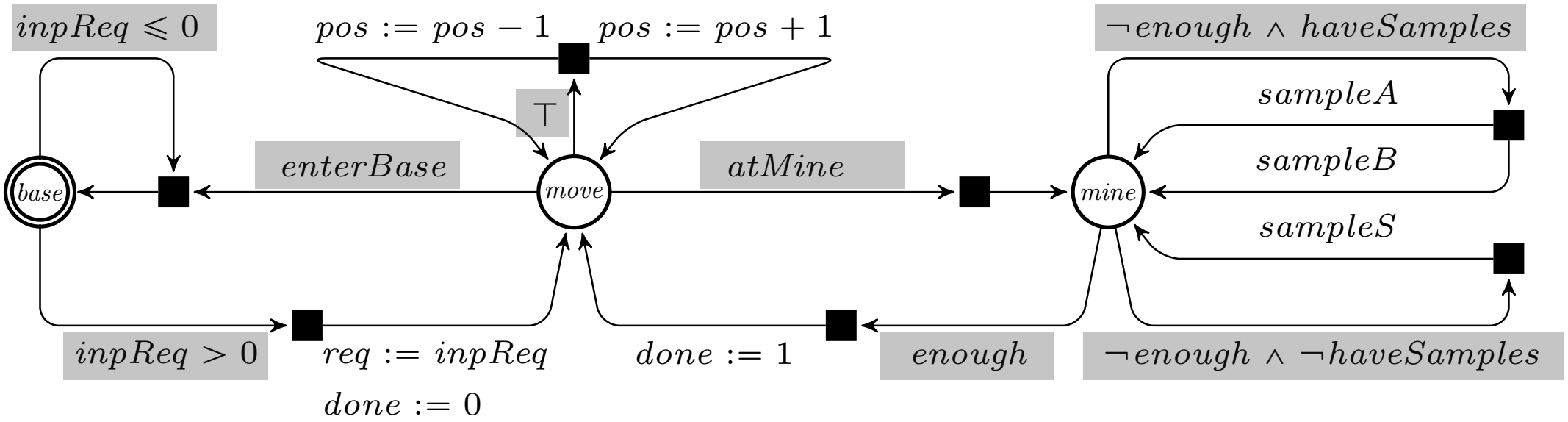
- Locations $\{base, move, mine\}$
- Input variables $\{inpReq, a, b\}$

Reactive Program Games: Example



- Locations $\{base, move, mine\}$
- Input variables $\{inpReq, a, b\}$
- Program variables $\{pos, done, req, samp\}$

Reactive Program Games: Example

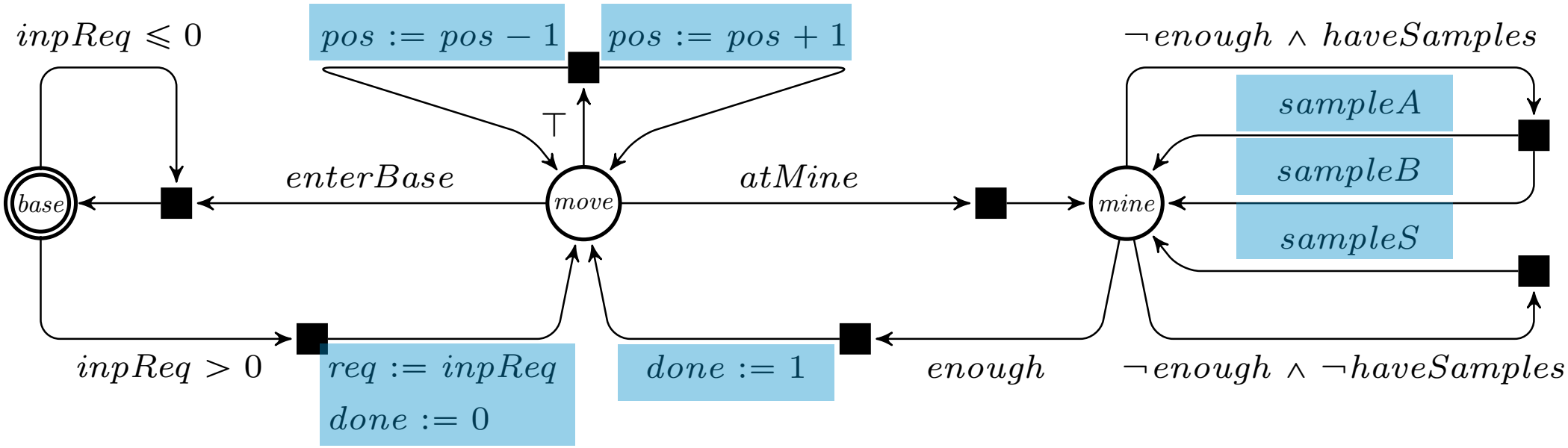


- Locations $\{base, move, mine\}$
- Input variables $\{inpReq, a, b\}$
- Program variables $\{pos, done, req, samp\}$

Transitions labelled with

- Guards

Reactive Program Games: Example

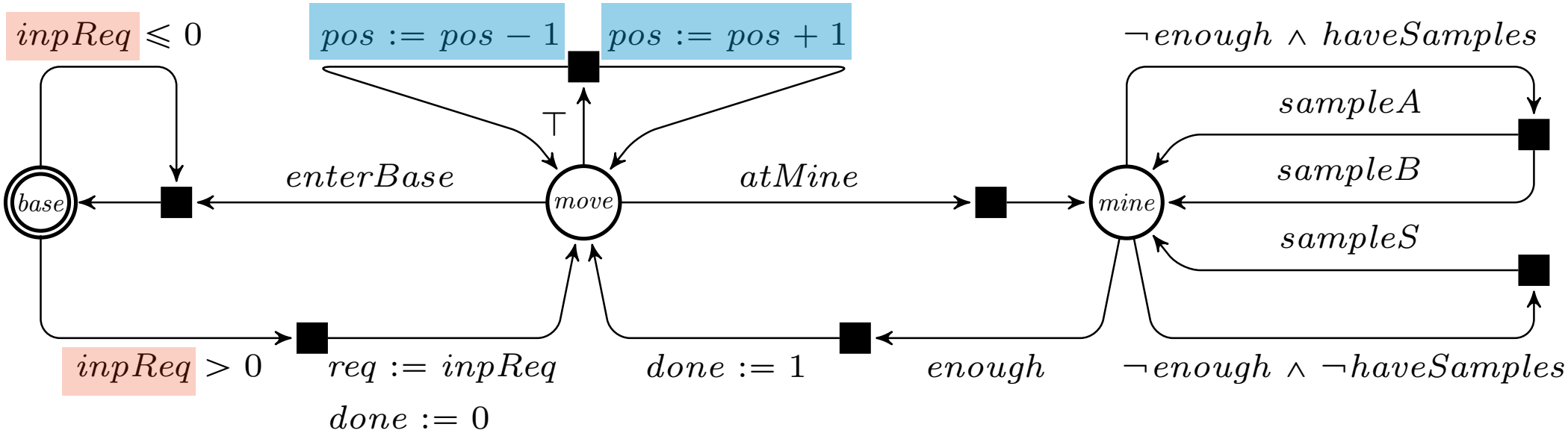


- Locations $\{base, move, mine\}$
- Input variables $\{inpReq, a, b\}$
- Program variables $\{pos, done, req, samp\}$

Transitions labelled with

- Guards
- Updates

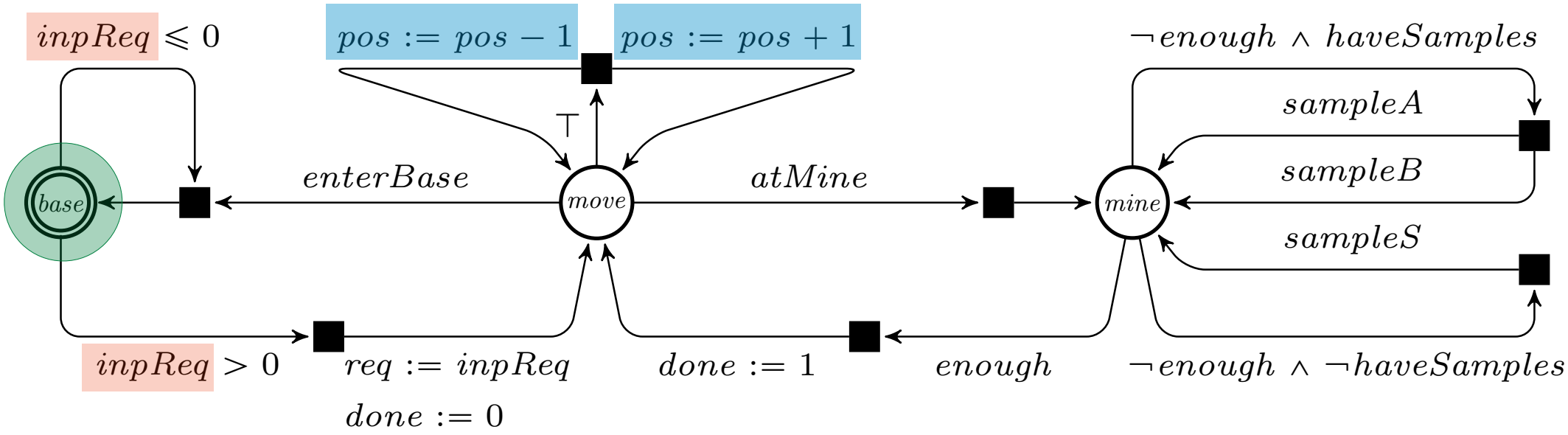
Reactive Program Games: Example



Two-player game

- **Environment** chooses values for input variables vs. **system** chooses update

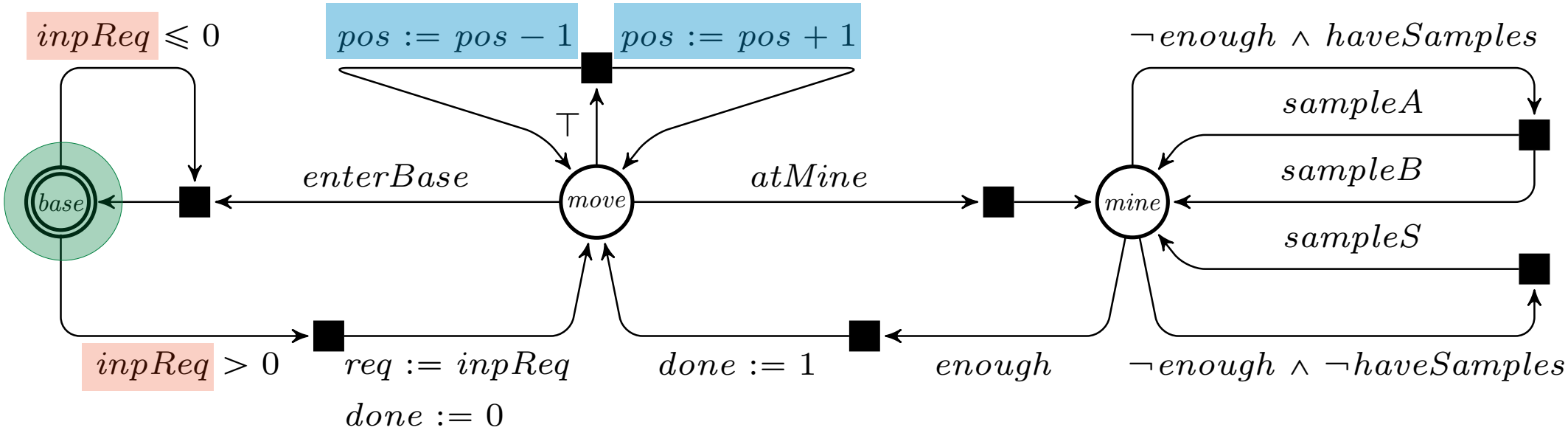
Reactive Program Games: Example



Two-player game

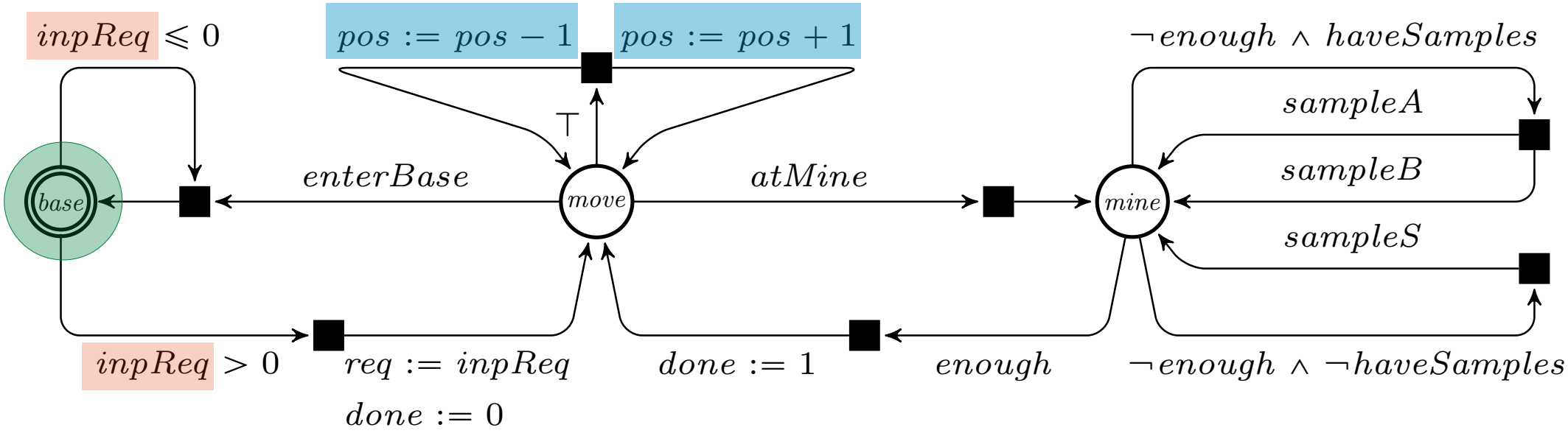
- **Environment** chooses values for input variables vs. **system** chooses update
- **Specification** location-based, e.g. safety, reachability, Büchi, parity

Reactive Program Games: Example



Spec: visit location *base* infinitely often

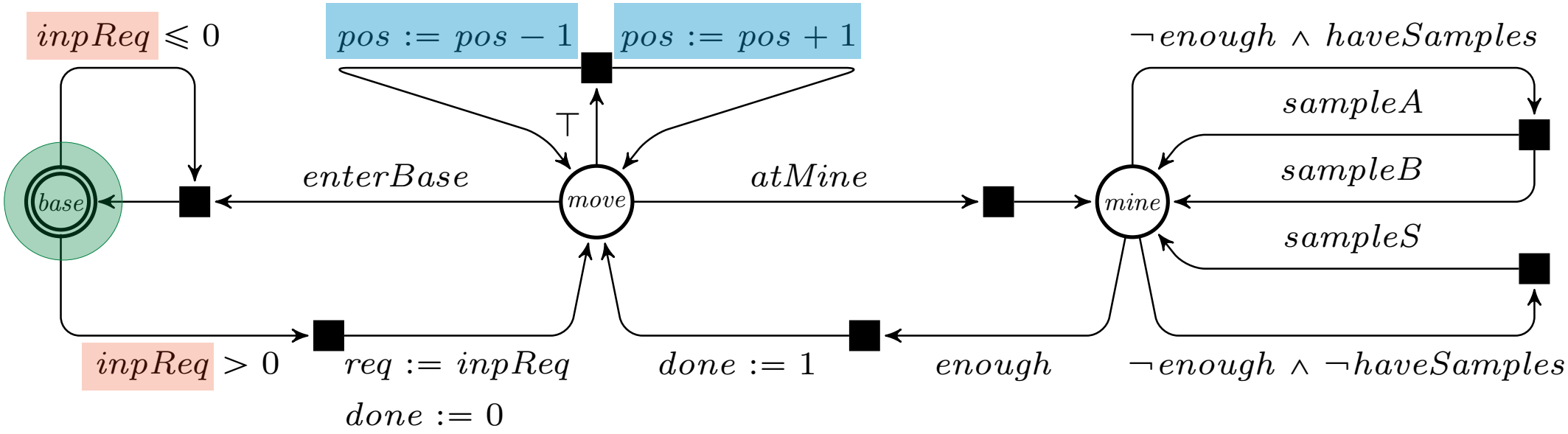
Reactive Program Games: Example



Spec: visit location *base* infinitely often

System wins, but reachability argument needs attractor acceleration

Reactive Program Games: Example



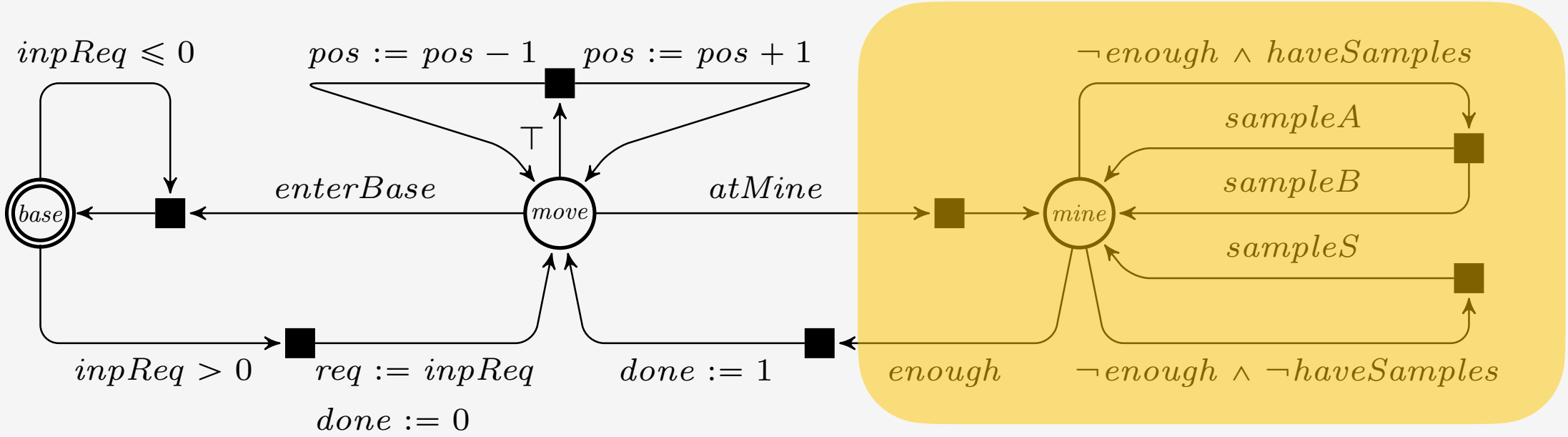
Spec: visit location *base* infinitely often

System wins, but reachability argument needs attractor acceleration

Challenge: size and complexity of game

Idea: Localized Attractor Computation

[Schmuck/Heim/Dimitrova/Nayak, CAV 2024]

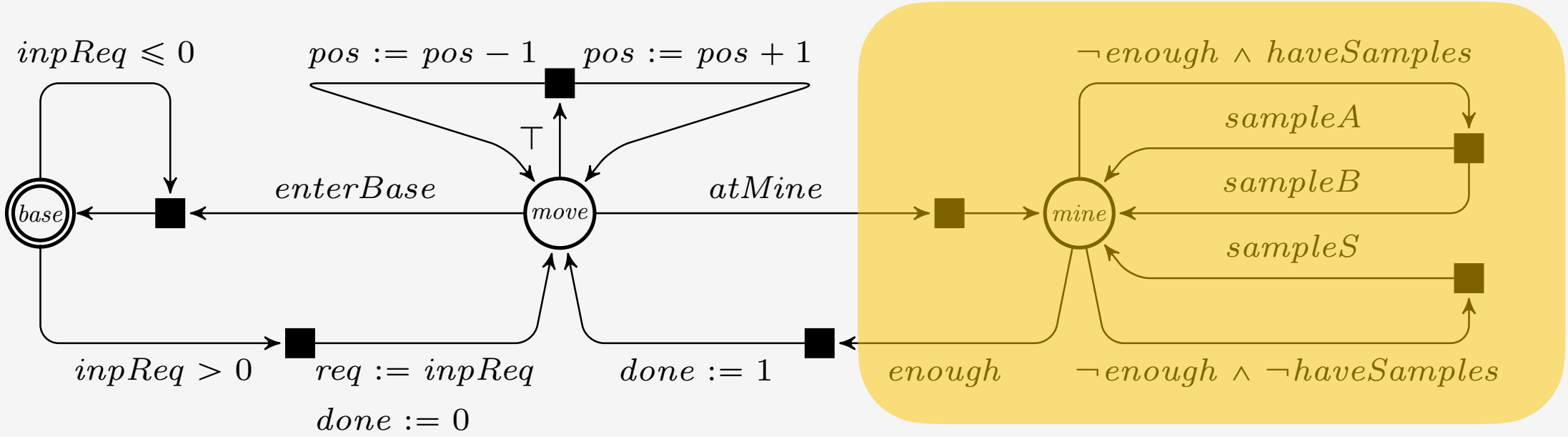


Goal:

- pre-compute **localized attractors** in small and useful sub-games
- utilize the **cached results** when solving the global game

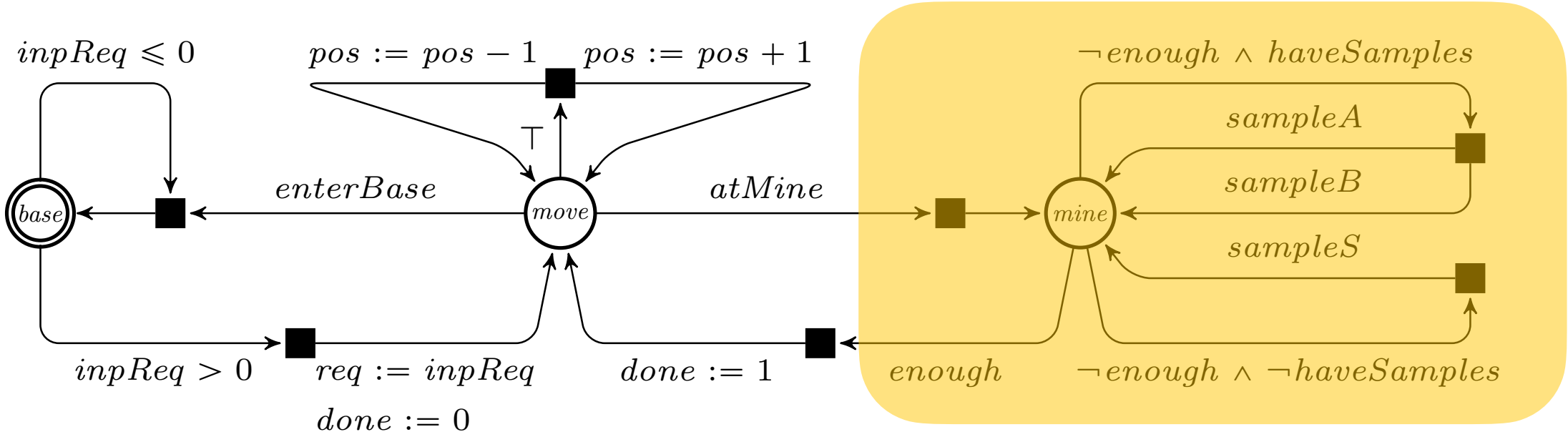
➡ **when possible, avoid attractor acceleration over the complete game**

Idea: Localized Attractor Computation



Attractor acceleration needed for establishing that the system can enforce the satisfaction of the guard $enough \triangleq samp \geq req$

Localized Attractor Computation



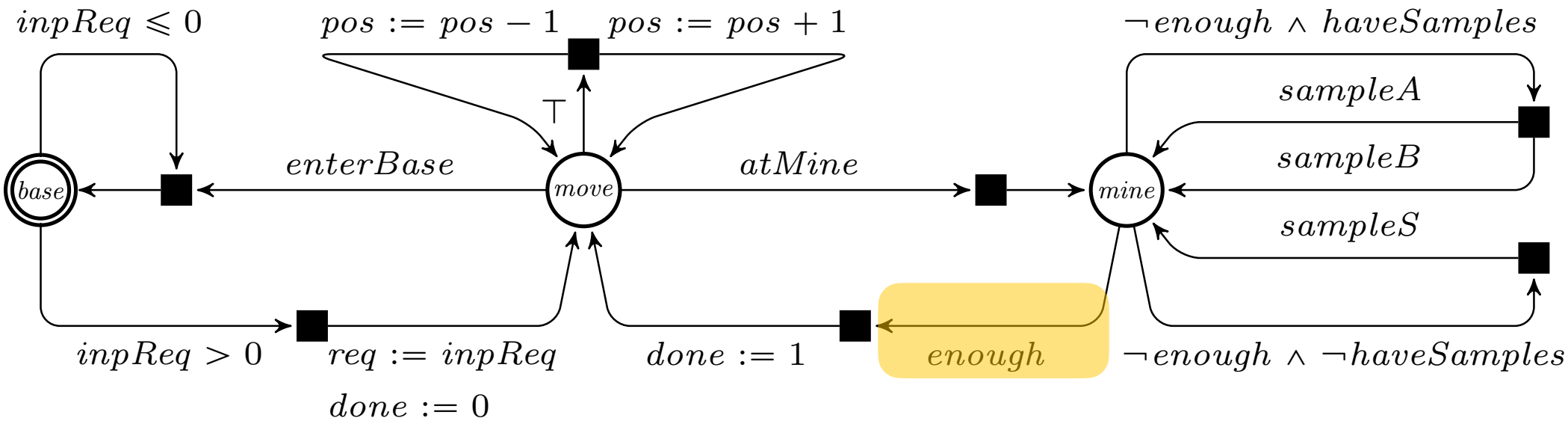
Cache attractor computation results

Result of local acceleration: $(Sys, samp \geq req, samp' > samp \wedge req' = req, \top)$

➡ **cache:** target set :$mine \mapsto samp \geq req$; source set: $mine \mapsto \top$

independent program variables: $done, pos, req$

Small and Useful Sub-Games via Helpful Edges



We consider sub-games induced by **helpful edges.**

Idea: capture transitions that a player might need to take,

hence the game solving procedure has to establish that

the player can eventually enable their guards

Abstraction to The Rescue

Reactive program game G + finite set of predicates $\mathscr{P}$

- $G^{\uparrow}$ over-approximate system player
- $G^{\downarrow}$ under-approximate system player

Abstraction to The Rescue

Reactive program game G + finite set of predicates $\mathscr{P}$

- $G^{\uparrow}$ over-approximate system player
- $G^{\downarrow}$ under-approximate system player

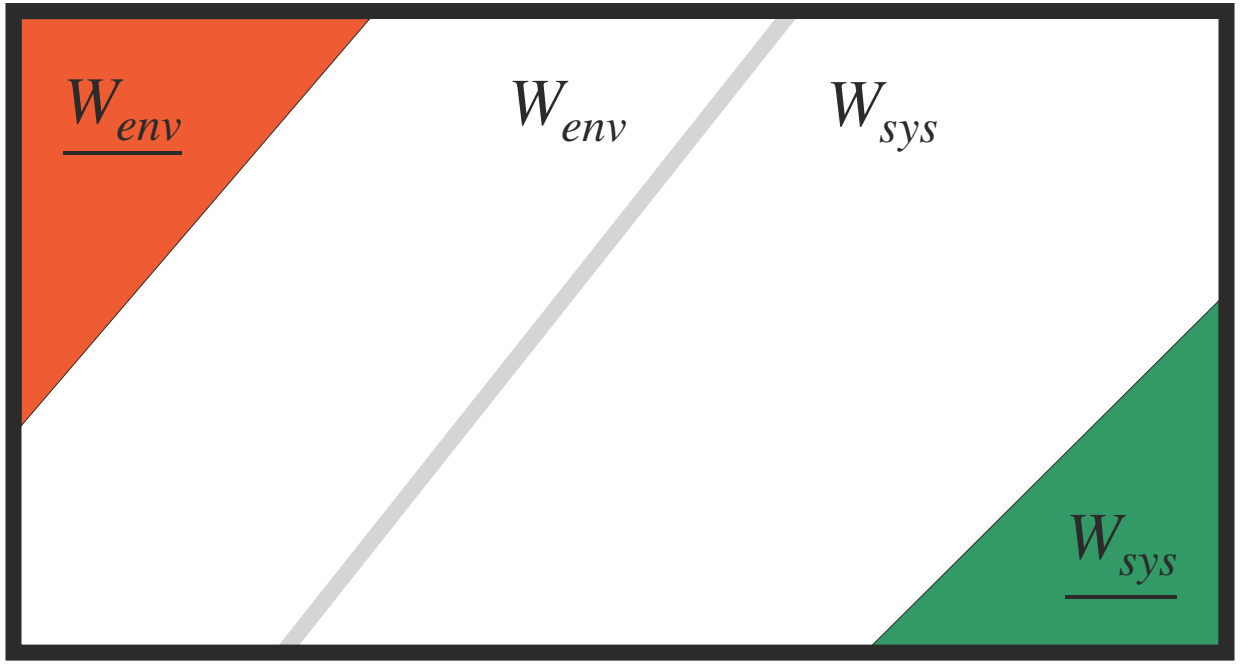
predicates: from the guards in G

Abstraction to The Rescue

Reactive program game G + finite set of predicates $\mathcal{P}$

- $G^{\uparrow}$ over-approximate system player
- $G^{\downarrow}$ under-approximate system player

predicates: from the guards in G



Abstraction to The Rescue: Optimistic Helpful Edges

Reactive program game G + finite set of predicates $\mathscr{P}$

- $G^{\uparrow}$ over-approximate system player
- $G^{\downarrow}$ under-approximate system player

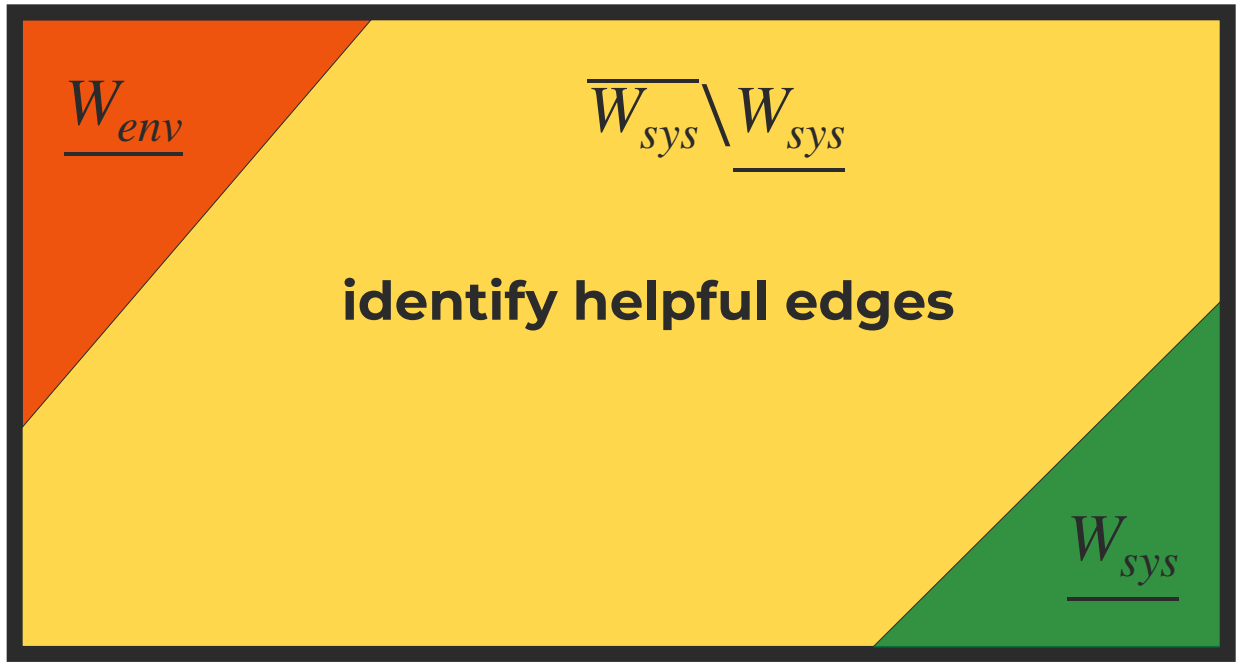
predicates: from the guards in G

Abstraction to The Rescue: Optimistic Helpful Edges

Reactive program game G + finite set of predicates $\mathscr{P}$

- $G^{\uparrow}$ over-approximate system player
- $G^{\downarrow}$ under-approximate system player

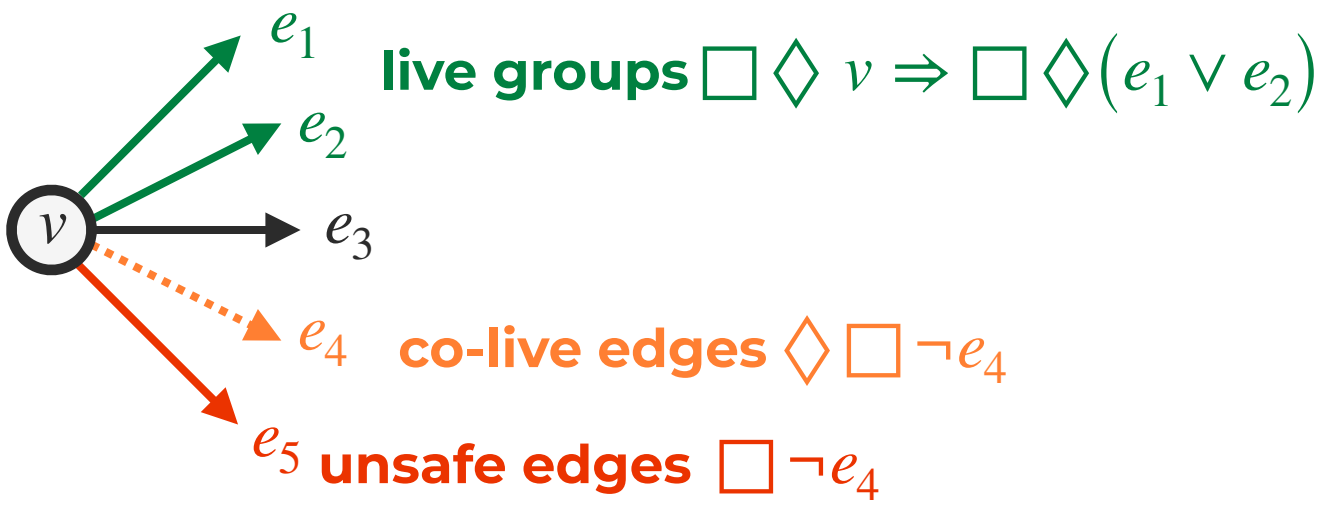
predicates: from the guards in G



Helpful Edges via Permissive Winning Strategy Templates

permissive winning strategy templates [Anand/Mallik/Nayak/Schmuck, TACAS 2023]

represent through **local edge conditions** (potentially infinite) sets of strategies winning for a given player in finite-state games



live groups $\square\lozenge\, v \Rightarrow \square\lozenge(e_1 \vee e_2)$

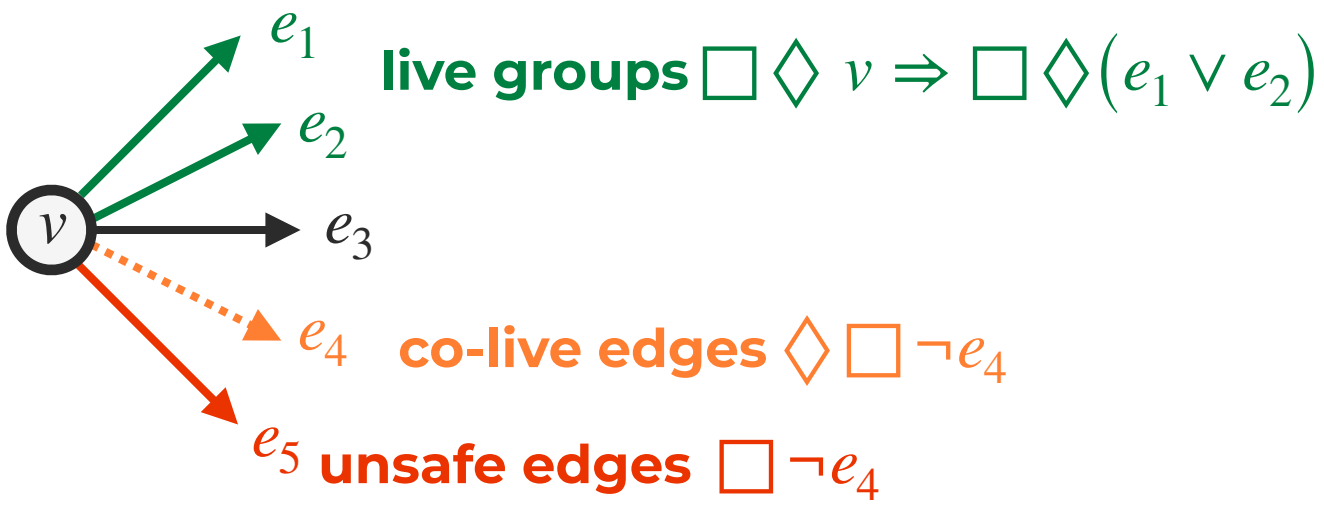
co-live edges $\lozenge\square\neg e_4$

unsafe edges $\square\neg e_4$

Helpful Edges via Permissive Winning Strategy Templates

permissive winning strategy templates [Anand/Mallik/Nayak/Schmuck, TACAS 2023]

represent through **local edge conditions** (potentially infinite) sets of strategies winning for a given player in finite-state games



live groups $\square\lozenge\, v \Rightarrow \square\lozenge(e_1 \vee e_2)$

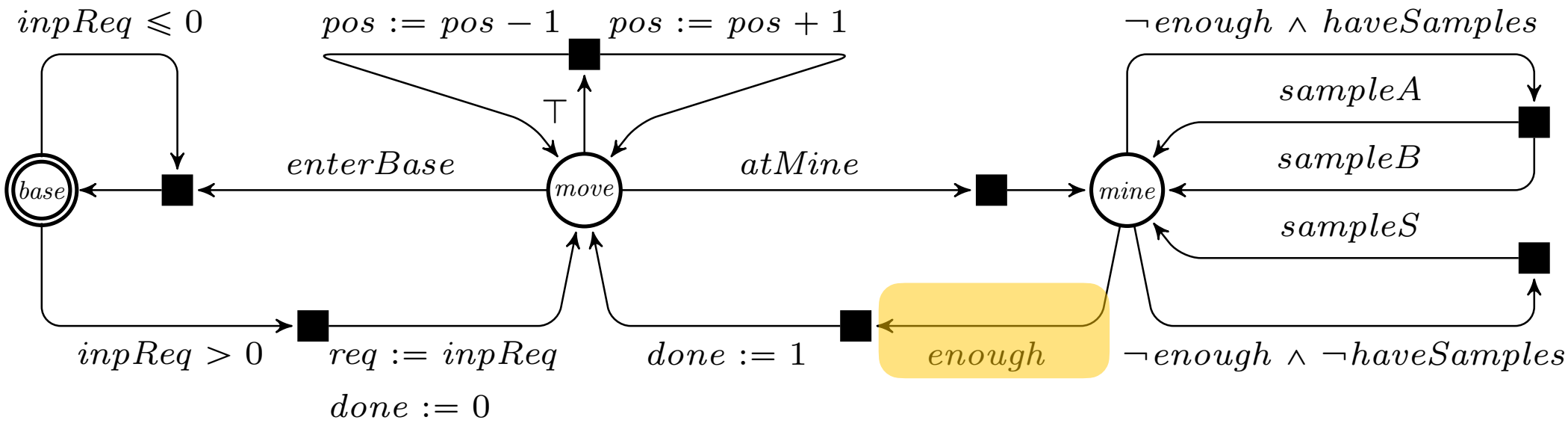
co-live edges $\lozenge\square\neg e_4$

unsafe edges $\square\neg e_4$

helpful edges

- edges that belong to a **live** group
- **safe alternatives to co-live** edges

Helpful Edges via Permissive Winning Strategy Templates



corresponds to an edge belonging to a
live group for system player in $G^{\uparrow}$

$enough \triangleq samp \geq req$ the value of $samp$ finally becomes greater or equal to req

Putting It All Together

1. Given G and finite set of predicates $\mathscr{P}$, construct $G^{\uparrow}$ and $G^{\downarrow}$
2. For each of $G^{\uparrow}$ and $G^{\downarrow}$, compute strategy templates
3. Compute localized attractors in sub-games induced by helpful edges
4. Use cached results in symbolic fixpoint-based method for solving G

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations

- Symbolic methods enhanced with logical reasoning
  - for solving realizability/synthesis games
  - for translation of temporal logic formulas
  - combining symbolic methods and abstraction

References (Part 2)

[Bodlaender/Hurkens/Kusters/Staals/Woeginger/Zantema, IFIP TCS, 2012]

M.H.L. Bodlaender, C.A.J. Hurkens, V.J.J.Kusters, F. Staals, G.J. Woeginger, H. Zantema. **Cinderella versus the Wicked Stepmother**. IFIP TCS 2012

[Heim/Dimitrova, POPL 2024]

P. Heim and R. Dimitrova. **Solving Infinite-State Games via Acceleration**. POPL 2024

[Heim/Dimitrova, POPL 2025]

P. Heim and R. Dimitrova. **Translation of Temporal Logic for Efficient Infinite-State Reactive Synthesis**. POPL 2025

[Heim/Dimitrova, CAV 2025]

P. Heim and R. Dimitrova. **Issy: A Comprehensive Tool for Specification and Synthesis of Infinite-State Reactive Systems**. CAV 2025

References (Part 2)

[Anand/Mallik/Nayak/Schmuck, TACAS 2023]

A. Anand, K. Mallik, S.P. Nayak, and A. Schmuck. **Computing adequately permissive assumptions for synthesis**. TACAS 2023

[Schmuck/Heim/Dimitrova/Nayak, CAV 2024]

A. Schmuck, P. Heim, R. Dimitrova, and S. P. Nayak. **Localized Attractor Computations for Infinite-State Games**. CAV 2024

Acknowledgement

Some of the slides in this lecture are by **Philippe Heim** (CISPA Helmholtz Center for Information Security)