Reactive Synthesis modulo Theories

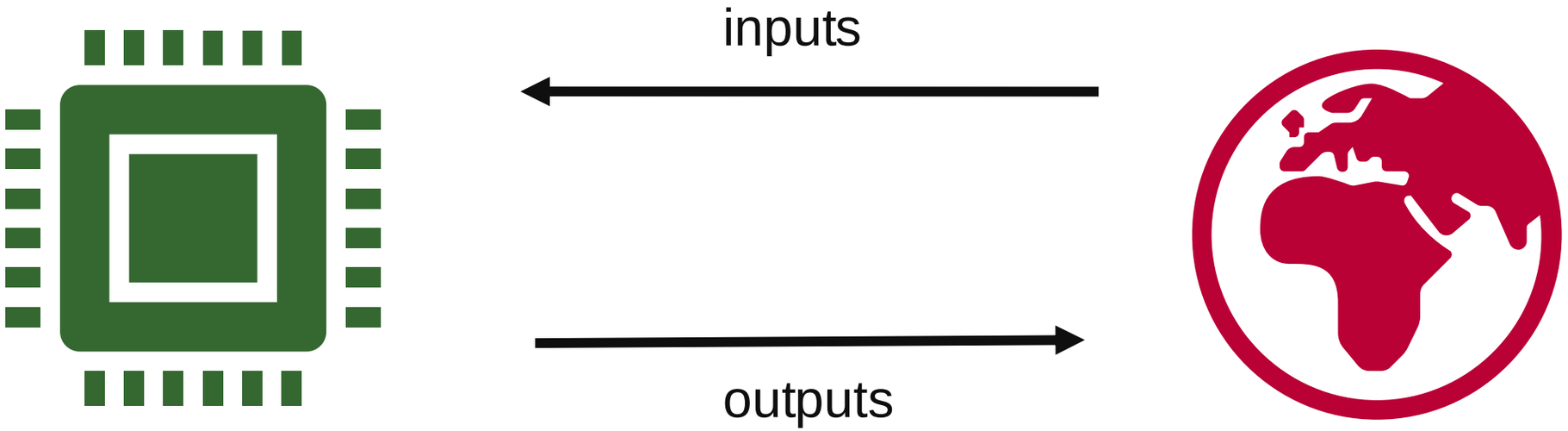
Benedikt Maderbacher
Graz University of Technology

AVM 2024
5 September 2024

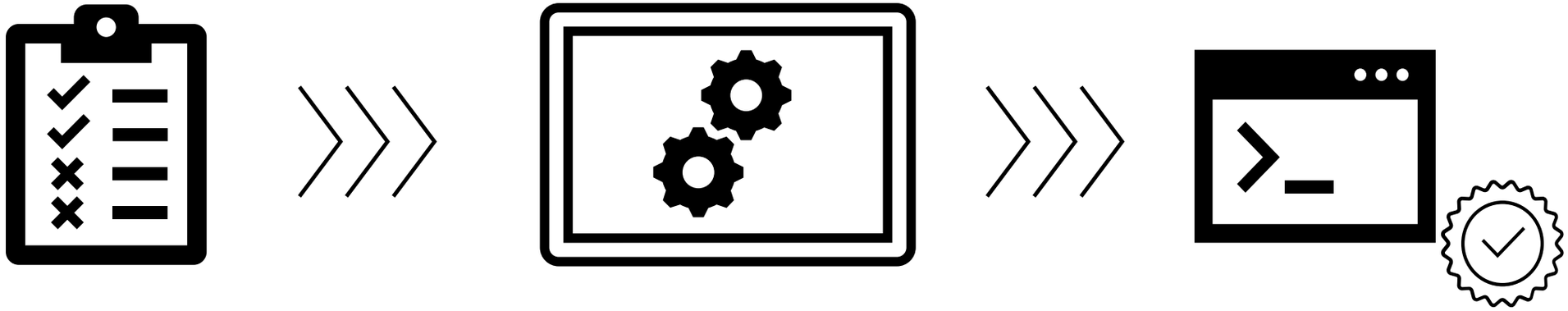
Outline

- Reactive Synthesis
- Abstraction Refinement
- Game Solving
- Parameterized Synthesis (WIP)

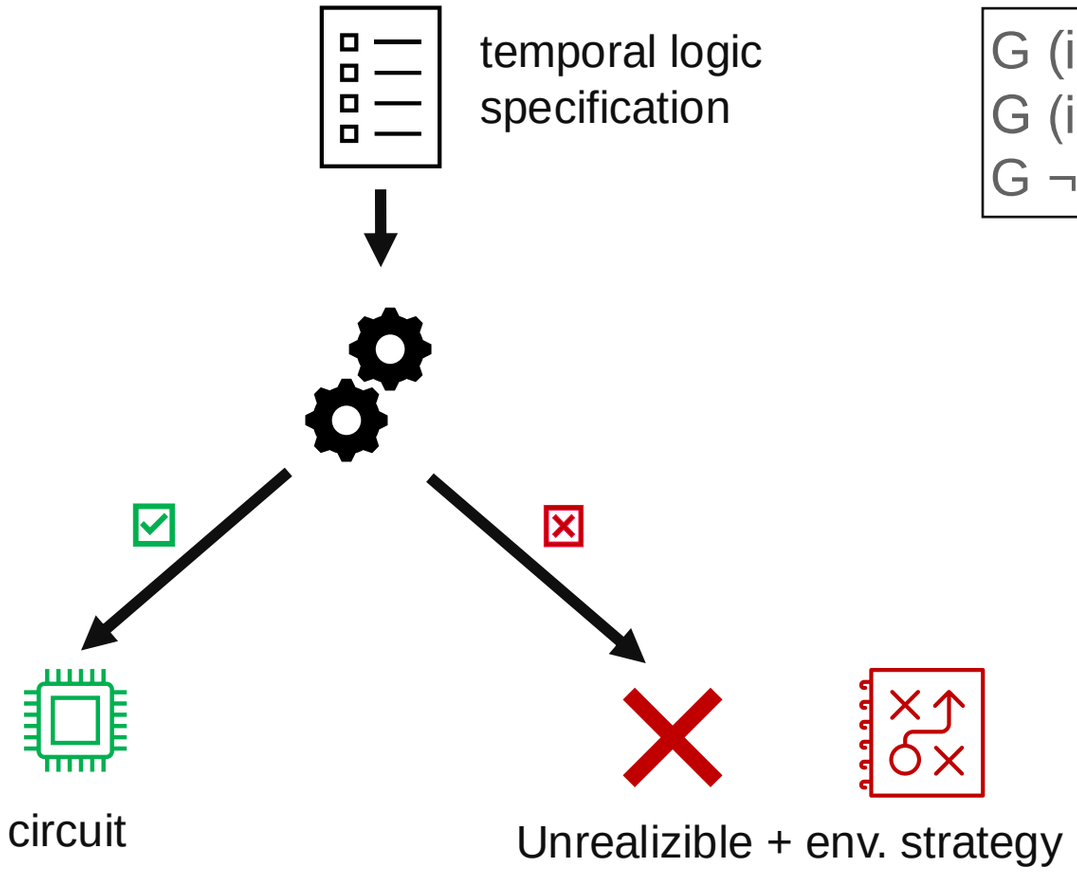
Reactive Systems



Synthesis



Reactive Synthesis



Propositional Reactive Synthesis

- Specification: Linear Temporal Logic

- Limitations:
  - Expressiveness
  - Struggles with large state spaces.

Temporal Logic + Memory + Theories

```
assume {
  x >= 0;
  x < 100;
}

always assume {
  i >= 0;
  i < 5;
}
```

```
always guarantee {
  x >= 0;
  x < 100;
  [x <- x+i] || [x <- x-i];
}
```

Implementation

```
while (true) {
    i := readInput();
    if( _ ) {
        x := x – i;
    }
    else {
        x := x + i;
    }
    sendOutput(x);
}
```

```
always assume {
  i >= 0;
  i < 5;
}

always guarantee {
 x >= 0;
 x < 100;
}
```

Implementation

```
while (true) {
    i := readInput();
    if(x>i) {
        x := x – i;
    }
    else {
        x := x + i;
    }
    sendOutput(x);
}
```

```
always assume {
  i >= 0;
  i < 5;
}

always guarantee {
 x >= 0;
 x < 100;
}
```

Abstraction

Abstraction

- Reuse existing LTL synthesis tools
- Create an LTL under approximation
  - If the LTL specification is realizable so is the TSL specification.
- Refine LTL abstraction using counter examples

Temporal Stream Logic

Specify

- complex systems
- reactive programs

Temporal logic plus:

- memory cells
- predicates over memory cells
- memory updates

Bernd Finkbeiner, Felix Klein, Ruzica Piskac, Mark Santolucito:
Temporal Stream Logic: Synthesis Beyond the Bools. CAV 2019

Propositional Encoding

- Atomic propositions become Boolean variables
  - $x+y>10 \Rightarrow p_{x+y>10}$
- Updates become Boolean variables
  - $[x \leftarrow x+i] \Rightarrow p_{[x \leftarrow x+i]}$
- Always exactly one update per variable
  - $\neg(p_{[x \leftarrow x+i]} \wedge p_{[x \leftarrow x-1]})$
    $p_{[x \leftarrow x+i]} \vee p_{[x \leftarrow x-1]}$

- System controls updates, environment controls everything else.

Propositional Encoding

```
assume {
  x >= 0;
  x < 100;
}
```

```
always assume {
  i >= 0;
  i < 5;
}
```

```
always guarantee {
  x >= 0;
  x < 100;
  [x <- x+i] || [x <- x-i];
}
```

$$\square\big(p_{[x\leftarrow x-i]} \wedge \neg\, p_{[x\leftarrow x+i]} \vee p_{[x\leftarrow x+i]} \wedge \neg p_{[x\leftarrow x-i]}\big) \wedge$$

$$\big((p_{0\leq x} \wedge p_{x<100} \wedge \square(p_{0\leq i} \wedge p_{i<5})) \rightarrow$$

$$\square(p_{0\leq x} \wedge p_{x<100} \wedge (p_{[x\leftarrow x-i]} \vee p_{[x\leftarrow x+i]})))$$

Propositional Encoding

```
assume {
  x >= 0;
  x < 100;
}
```

```
always assume {
  i >= 0;
  i < 5;
}
```

```
always guarantee {
  x >= 0;
  x < 100;
  [x <- x+i] || [x <- x-i];
}
```

$$\square\big(p_{[x\leftarrow x-i]} \wedge \neg\, p_{[x\leftarrow x+i]} \vee p_{[x\leftarrow x+i]} \wedge \neg p_{[x\leftarrow x-i]}\big) \wedge$$

$$\big(\big(p_{0\leq x} \wedge p_{x<100} \wedge \square\big(p_{0\leq i} \wedge p_{i<5}\big)\big) \rightarrow$$

$$\square\big(p_{0\leq x} \wedge p_{x<100} \wedge \big(p_{[x\leftarrow x-i]} \vee p_{[x\leftarrow x+i]}\big)\big)\big)$$

Propositional Encoding

```
assume {
  x >= 0;
  x < 100;
}
```

```
always assume {
  i >= 0;
  i < 5;
}
```

```
always guarantee {
  x >= 0;
  x < 100;
  [x <- x+i] || [x <- x-i];
}
```

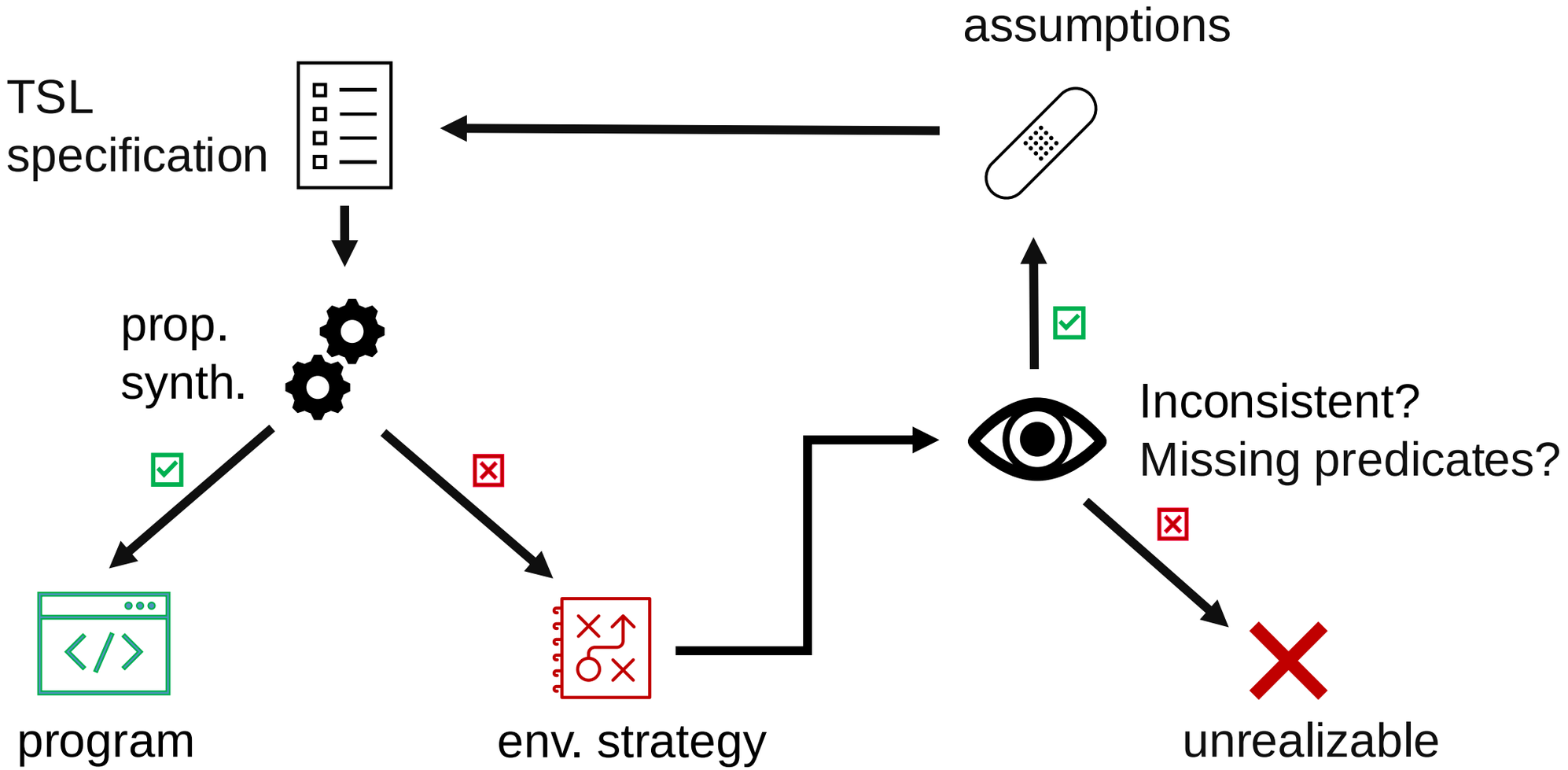
$$\square\big(p_{[x\leftarrow x-i]} \wedge \neg\, p_{[x\leftarrow x+i]} \vee p_{[x\leftarrow x+i]} \wedge \neg p_{[x\leftarrow x-i]}\big) \wedge$$

$$\big((p_{0\leq x} \wedge p_{x<100} \wedge \square(p_{0\leq i} \wedge p_{i<5})) \rightarrow$$

$$\square(p_{0\leq x} \wedge p_{x<100} \wedge (p_{[x\leftarrow x-i]} \vee p_{[x\leftarrow x+i]})))$$

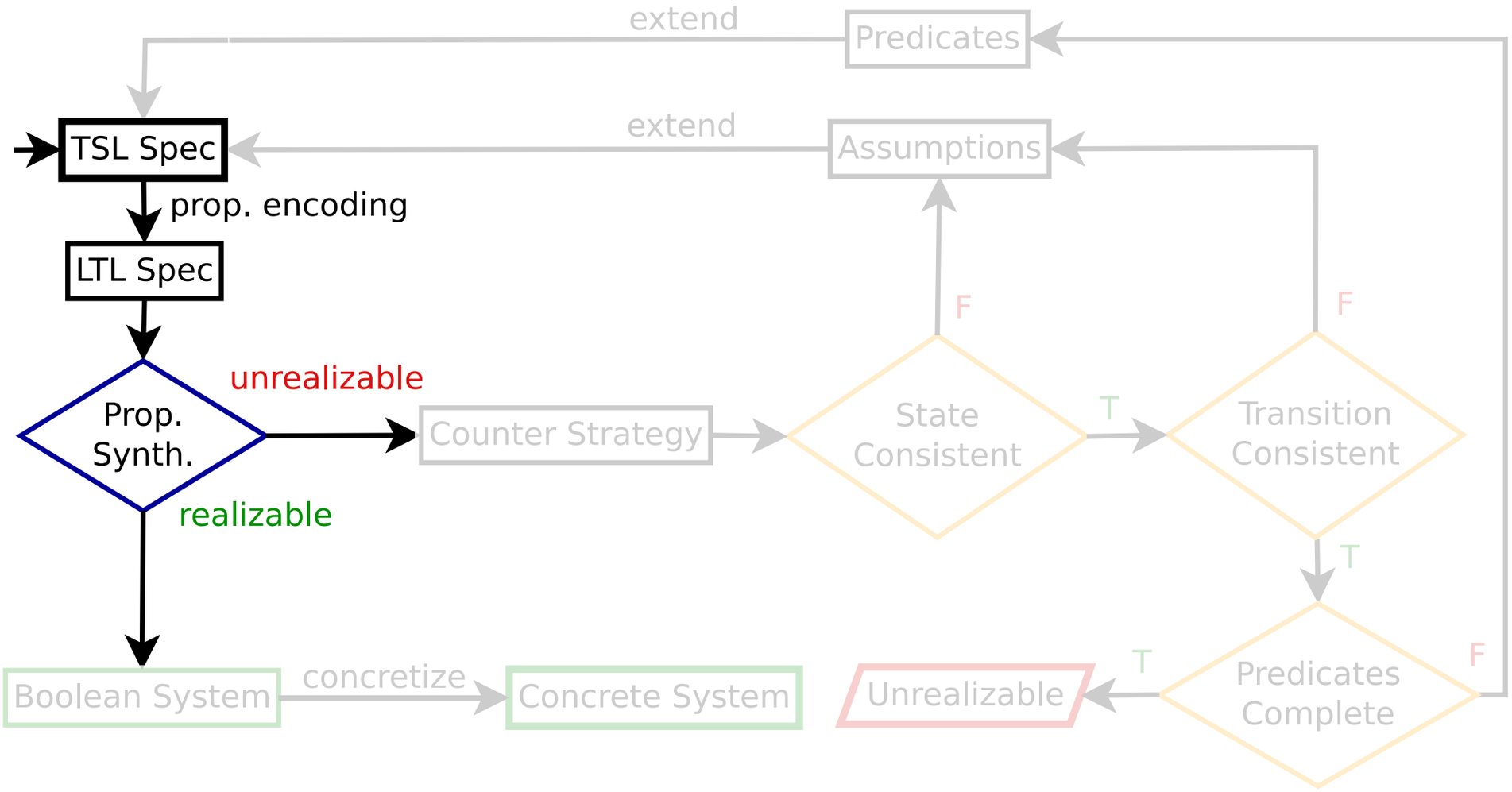
- A single abstraction step works for very simple specifications.
- Arithmetic requires more involved abstractions including new predicates

Reactive Synthesis modulo Theories using Abstraction Refinement

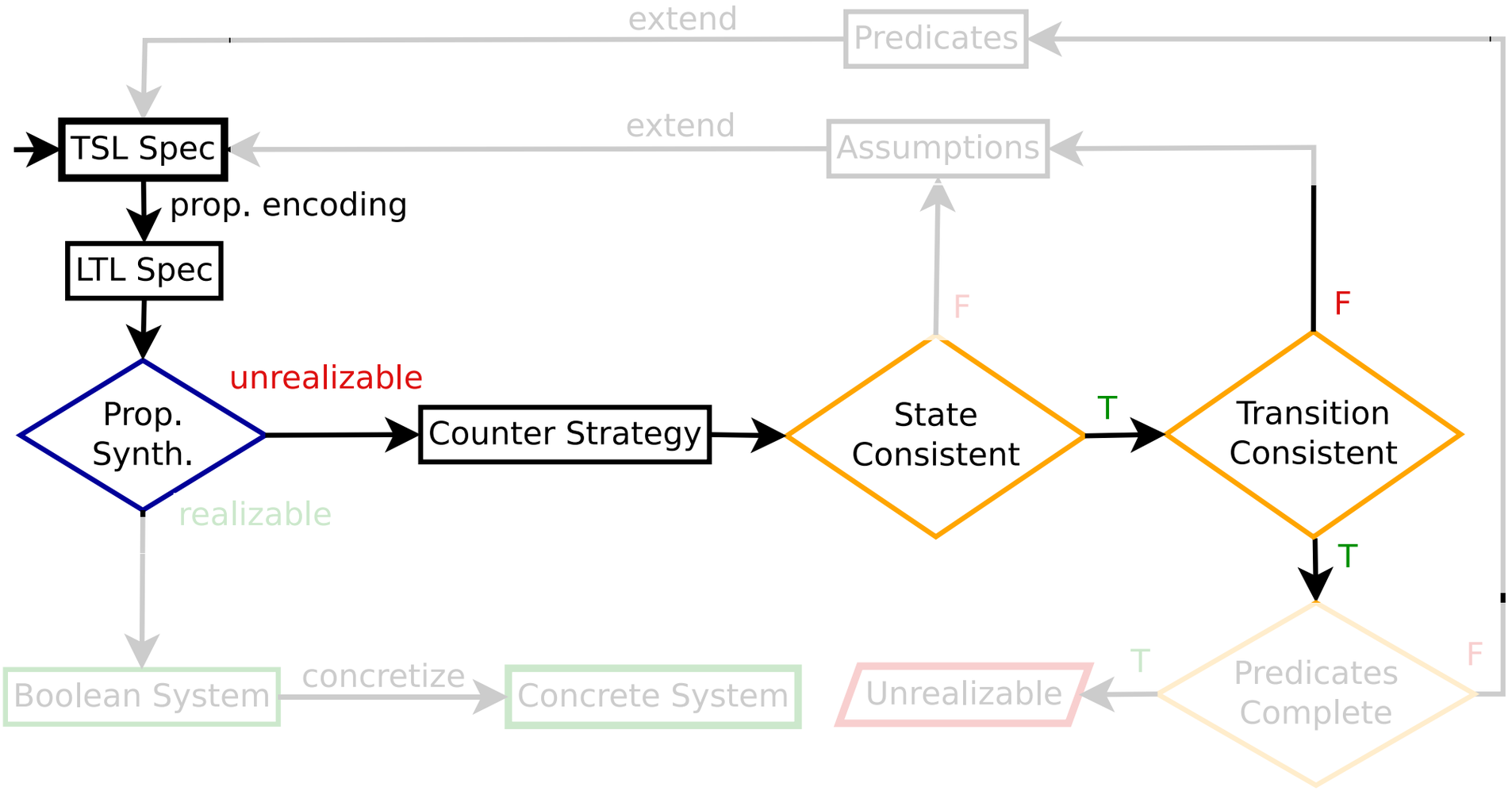


Benedikt Maderbacher, Roderick Bloem:
Reactive Synthesis Modulo Theories using Abstraction Refinement. FMCAD 2022

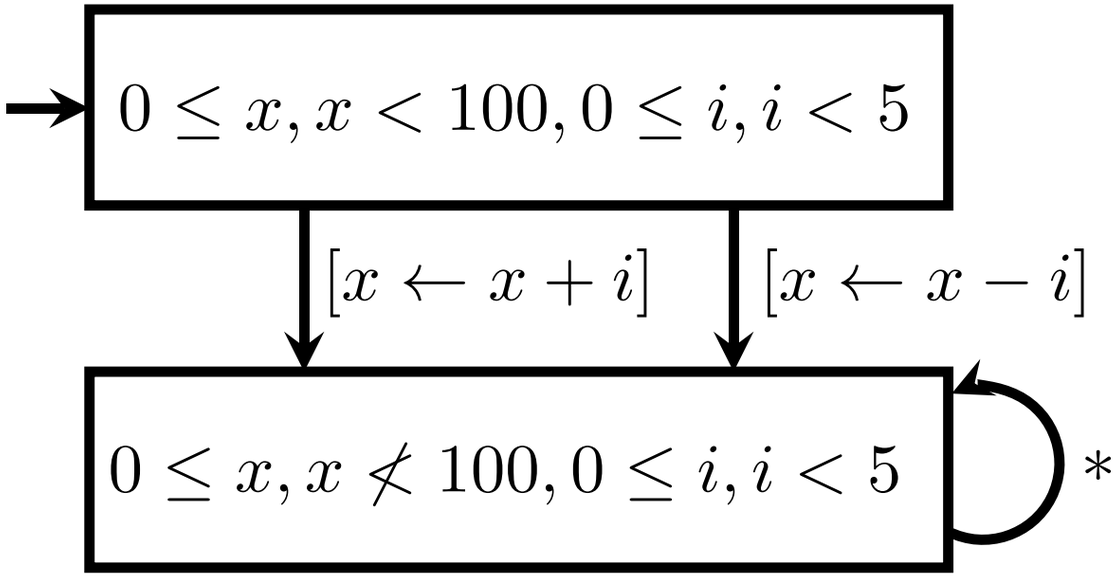
Synthesis Algorithm



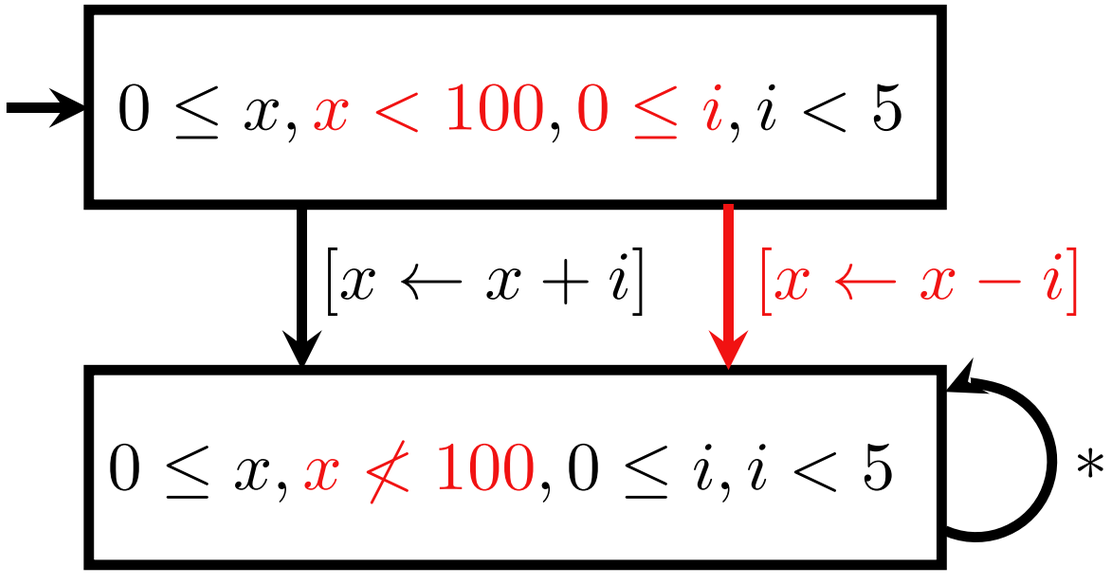
Synthesis Algorithm



Counter Strategy



Counter Strategy (Inconsistent Transition)



Contradiction:

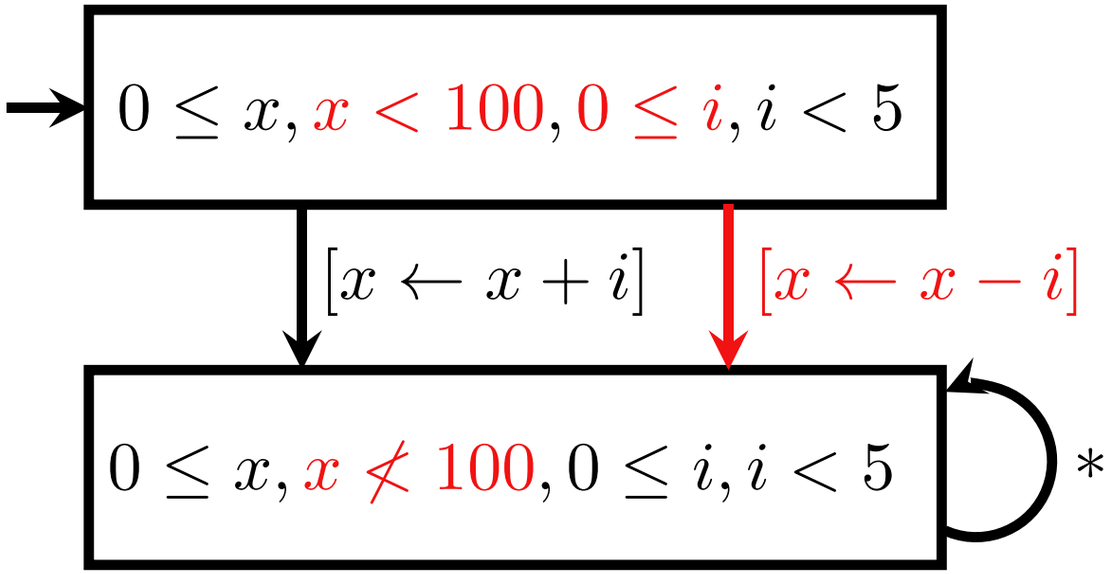
$x < 100,$

$0 \leq i,$

$x' = x\text{-}i,$

$x' \geq 100$

Counter Strategy (Inconsistent Transition)



Contradiction:

$x < 100,$

$0 \leq i,$

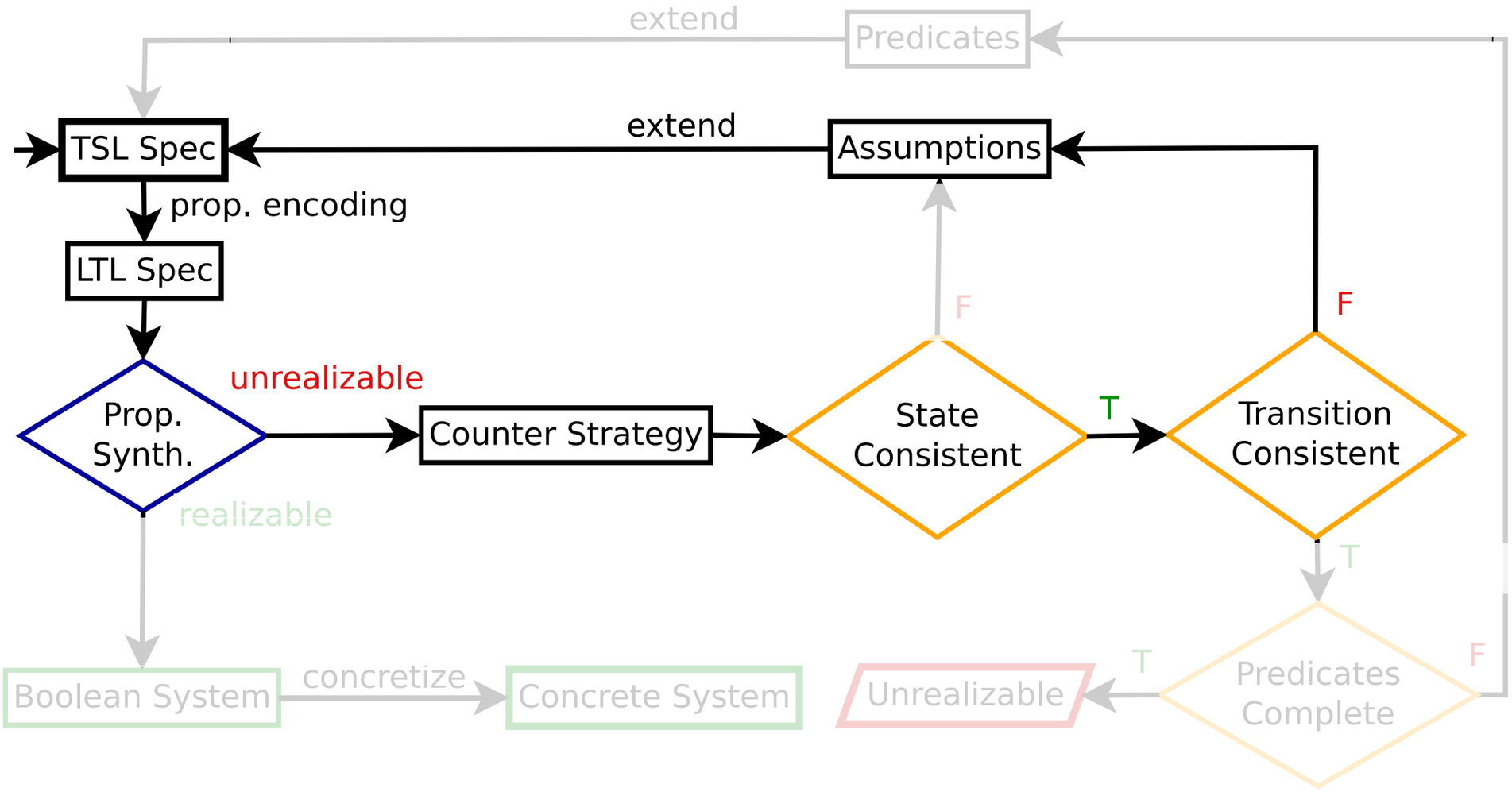
$x' = x\text{-}i,$

$x' \geq 100$

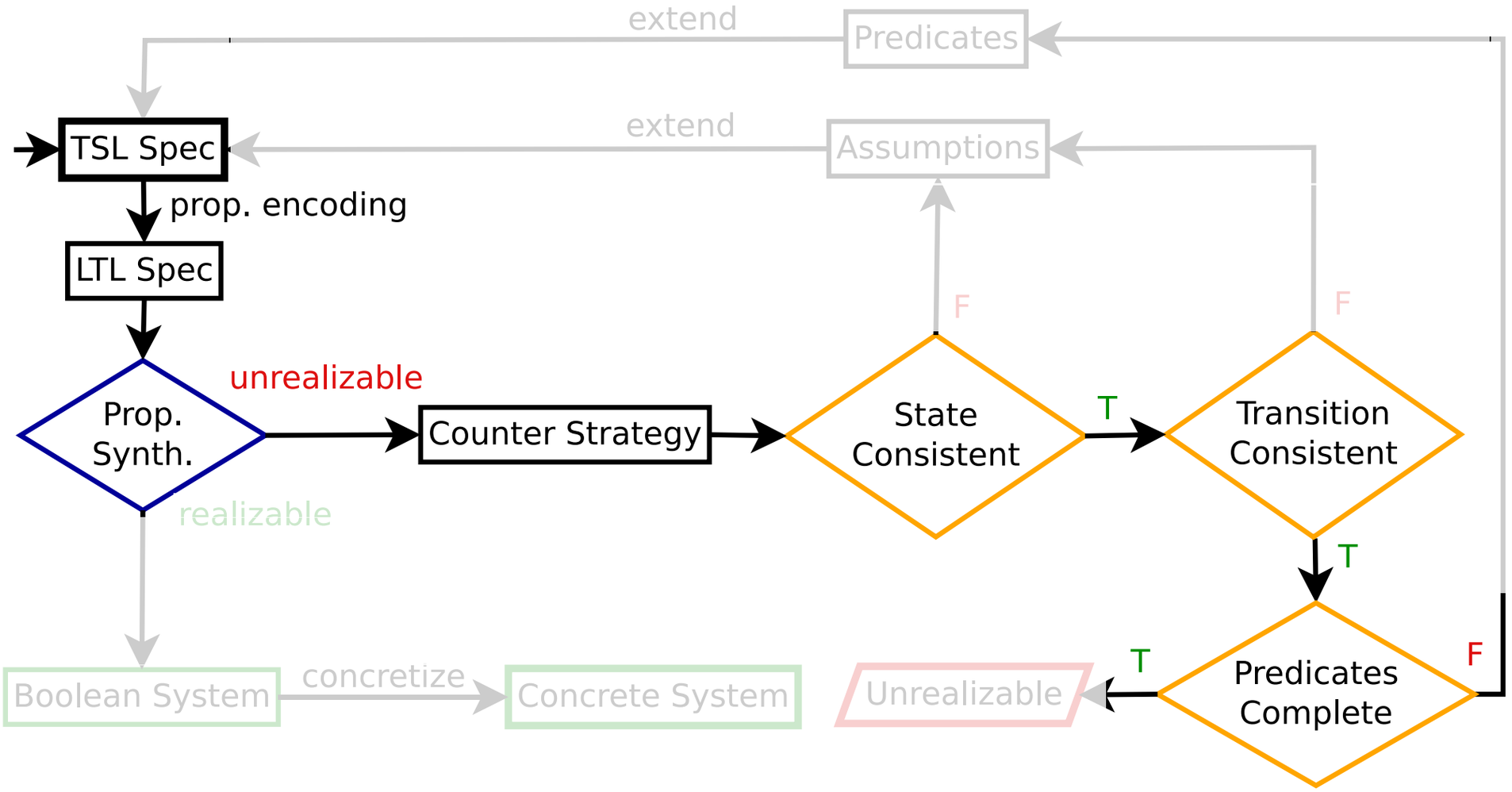
Added assumption:

$G\ ((x < 100 \wedge 0 \leq i \wedge [x \leftarrow x\text{-}i]) \rightarrow X\ (x < 100))$

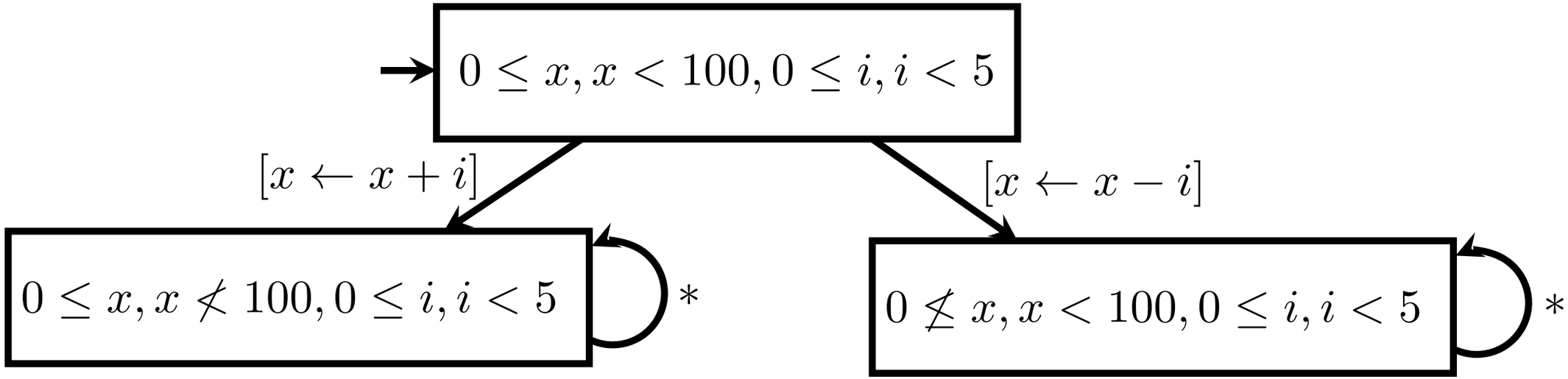
Synthesis Algorithm



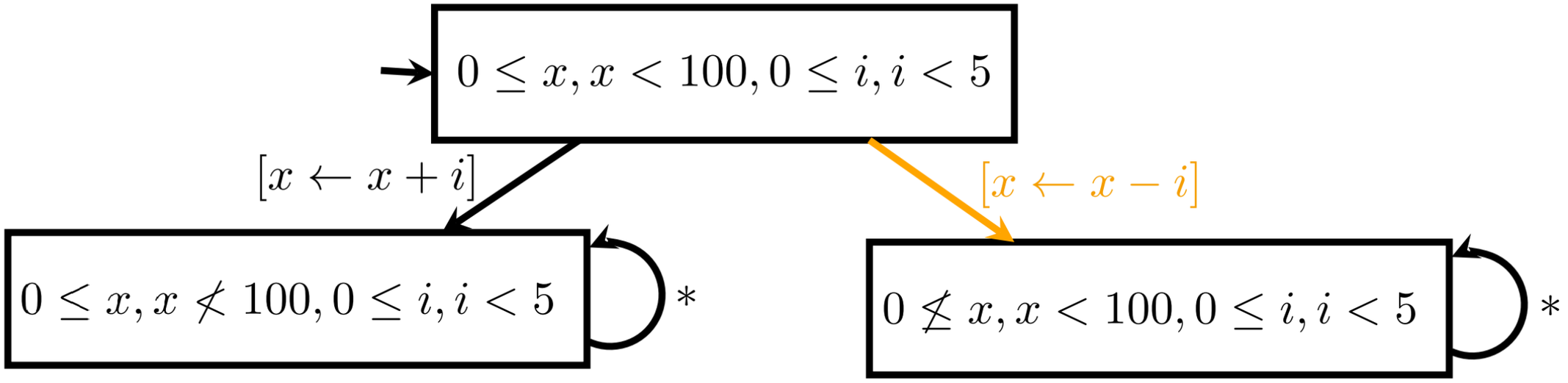
Synthesis Algorithm



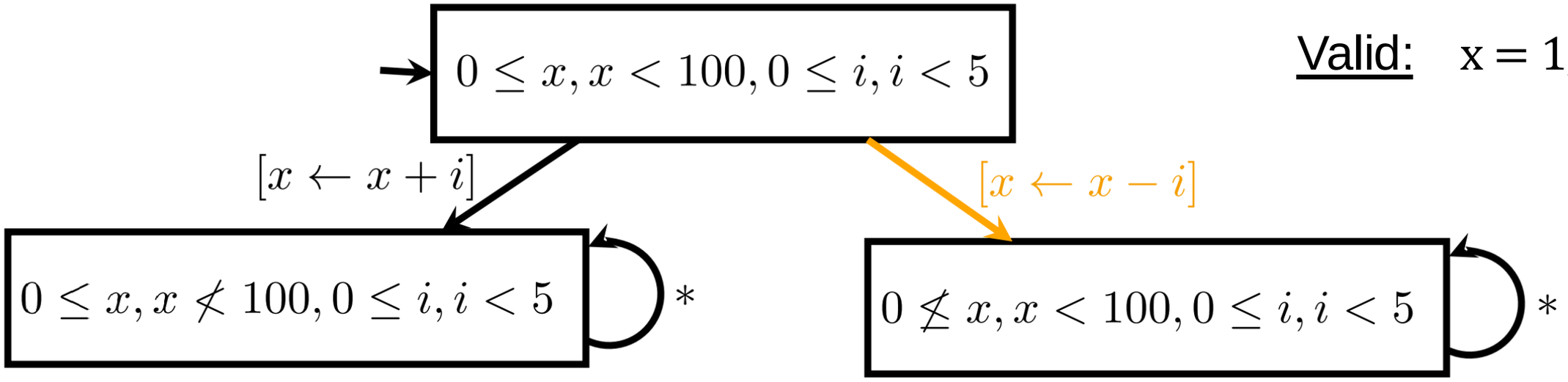
Counter Strategy



Counter Strategy

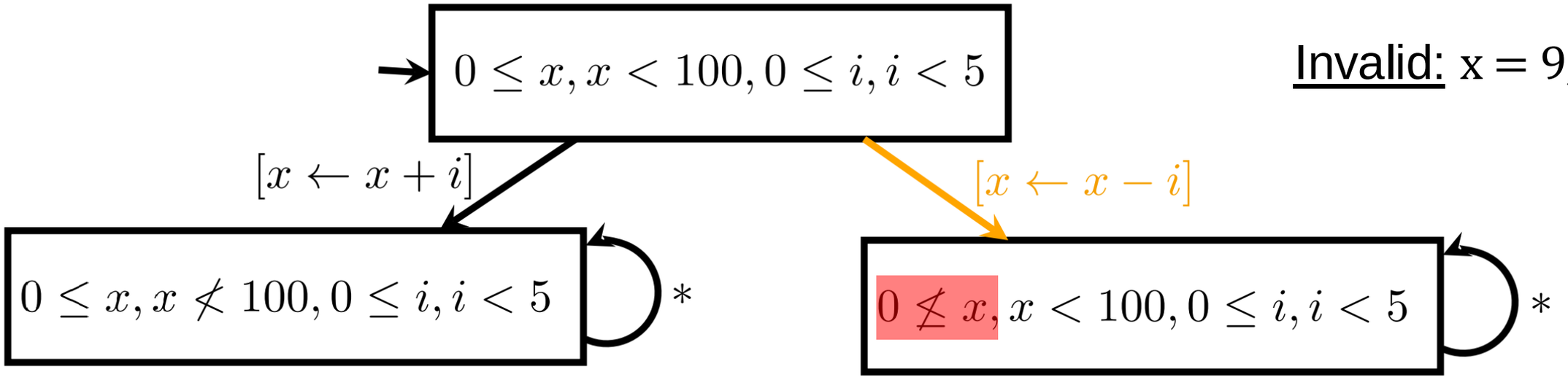


Counter Strategy



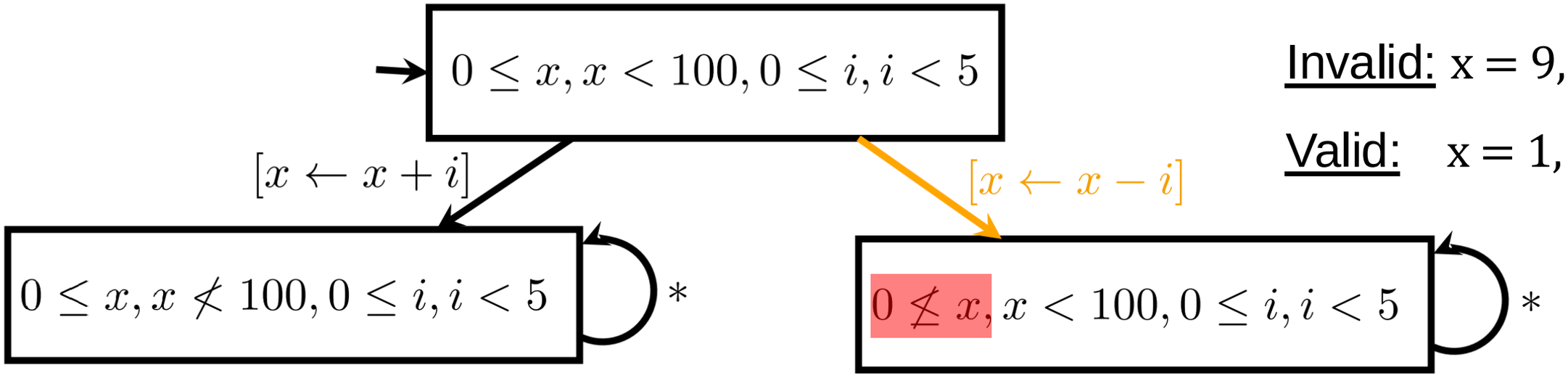
Valid: $x = 1, i = 4$

Counter Strategy



Invalid: x = 9, i = 1

Counter Strategy



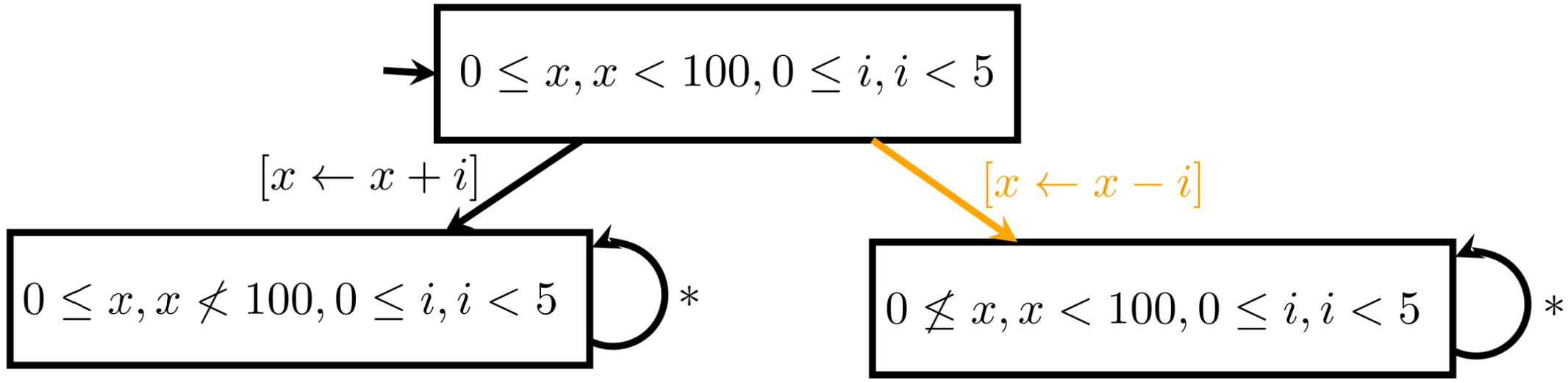
Invalid: x = 9, i = 1

Valid: x = 1, i = 4

Find a distinguishing predicate:

$i \leq x$

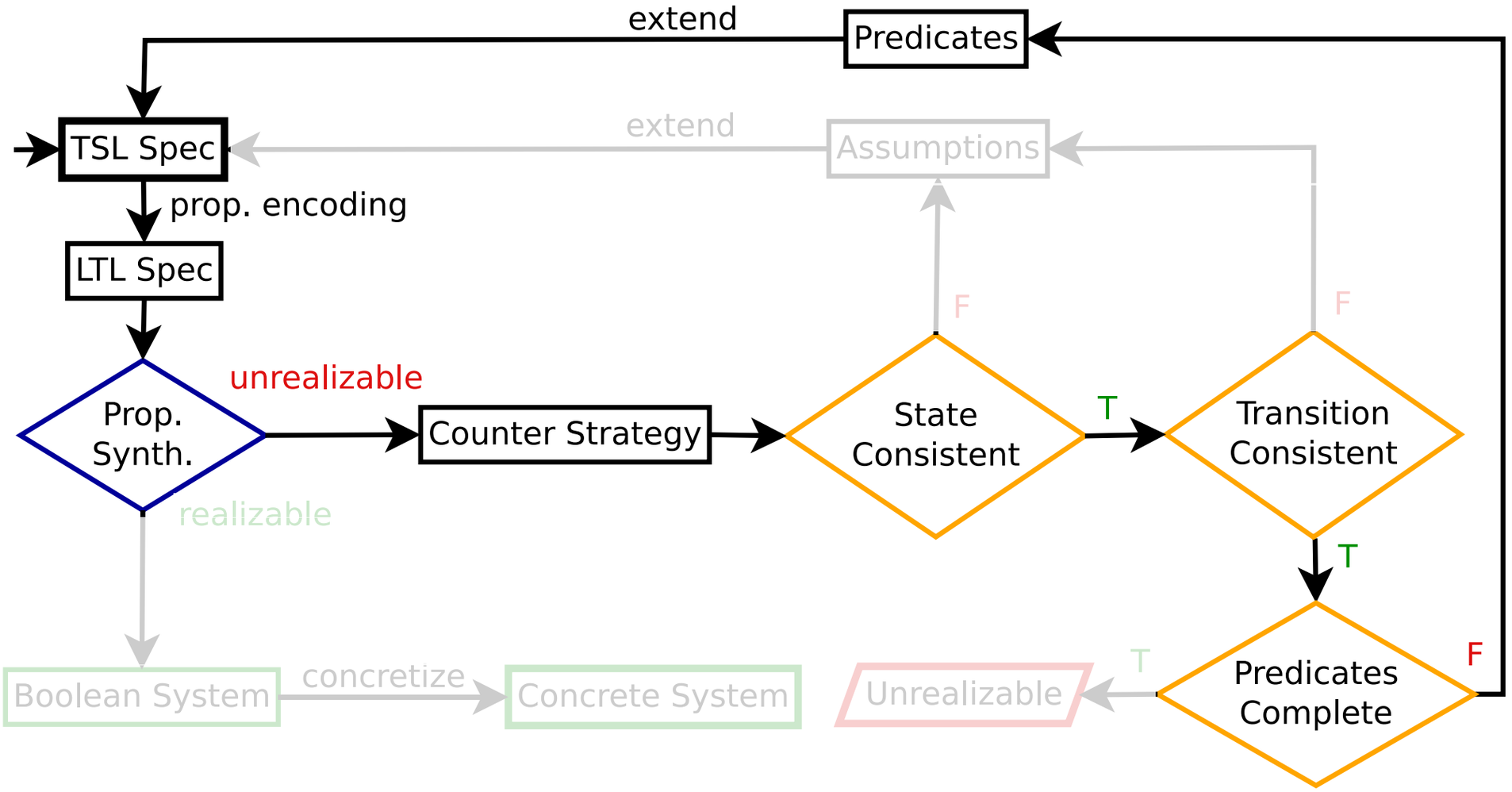
Counter Strategy (Missing Predicates)



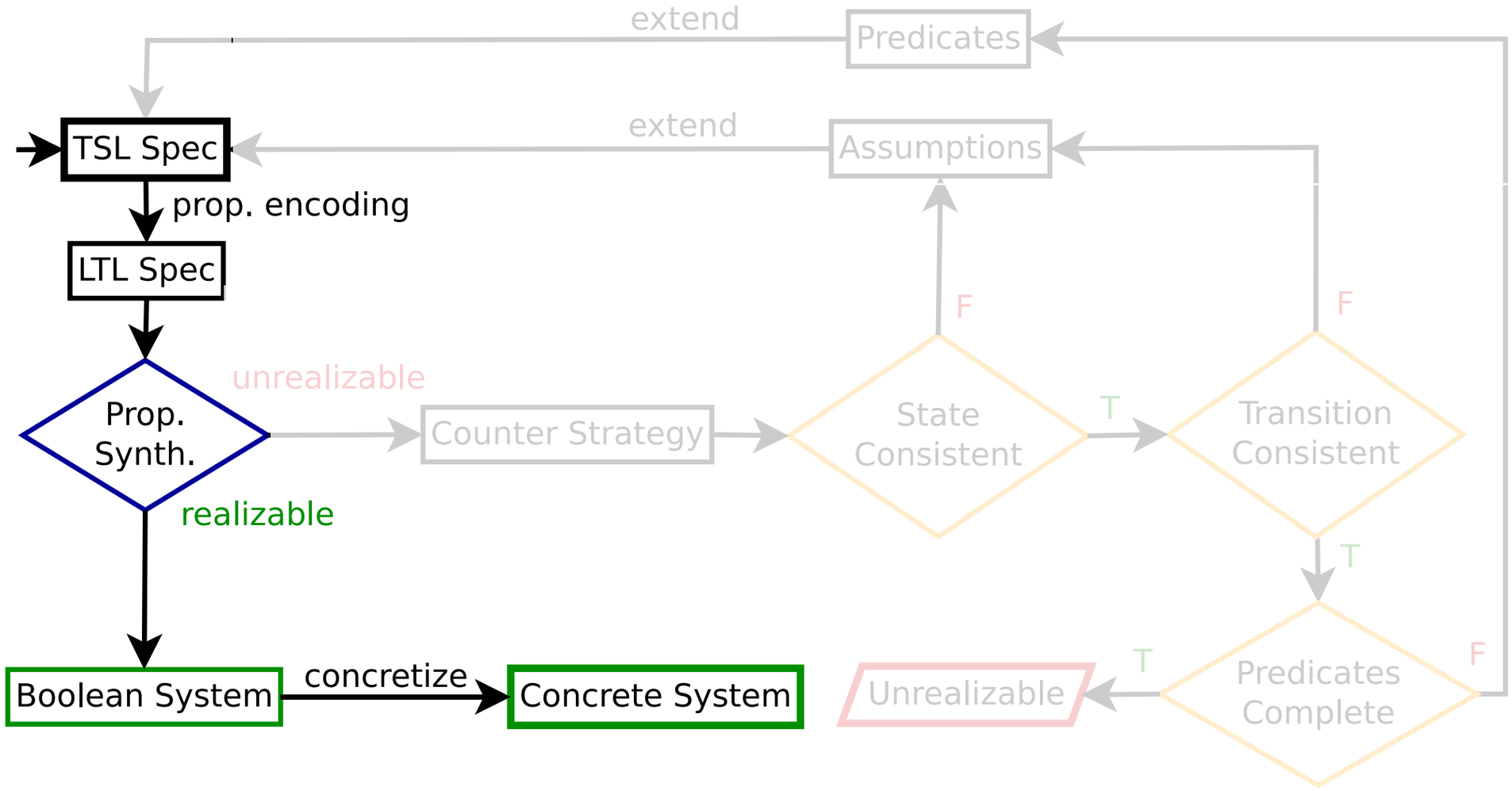
Added assumption:

$G((i \leq x \wedge [x \leftarrow x\text{-}i]) \rightarrow X(x \geq 0))$

Synthesis Algorithm

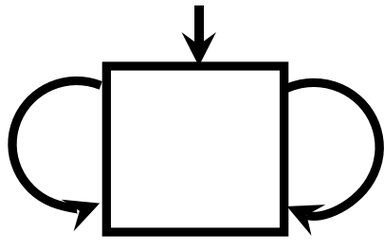


Synthesis Algorithm



Mealy Machine realizing the Specification

$i \not\leq x, * \;/\; [x \leftarrow x + i]$



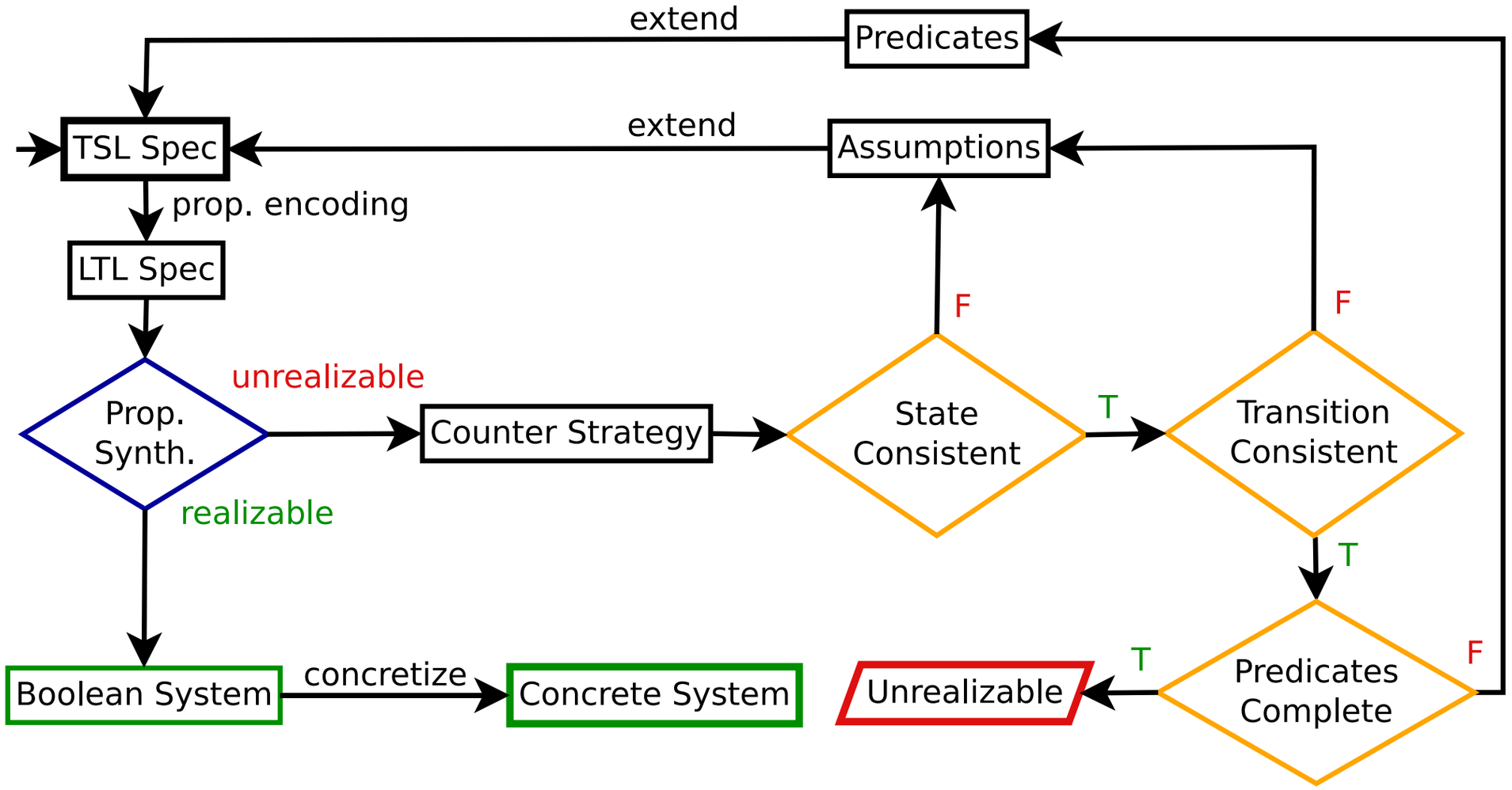
$i \leq x, * \;/\; [x \leftarrow x - i]$

Implementation

```
while (true) {
    in := readInput();
    if(0 <= x-i) {
        x := x – i;
    }
    else {
        x := x + i;
    }
    sendOutput(x);
}
```

```
assume {
  x >= 0;
  x < 100;
}
always assume {
  i >= 0;
  i < 5;
}
always guarantee {
 x >= 0;
 x < 100;
 [x <- x+i] || [x <- x-i];
}
```

Synthesis with Abstraction Refinement

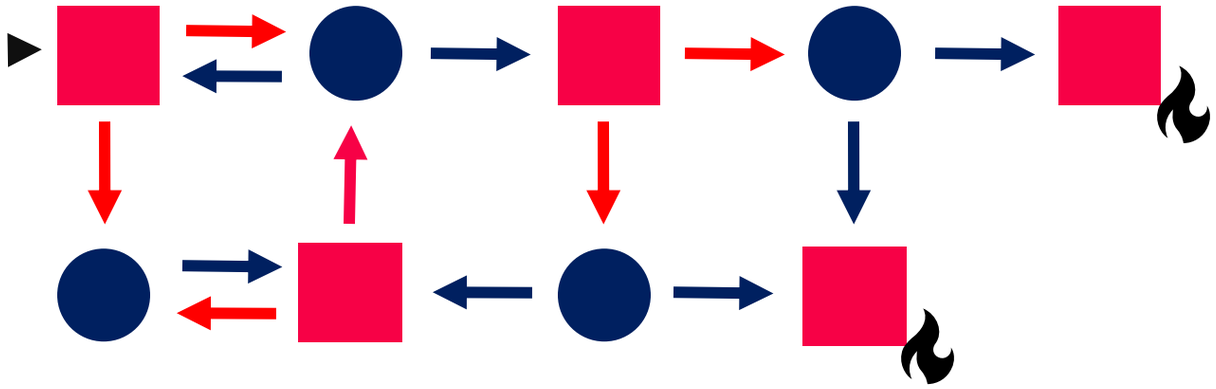


Game Solving

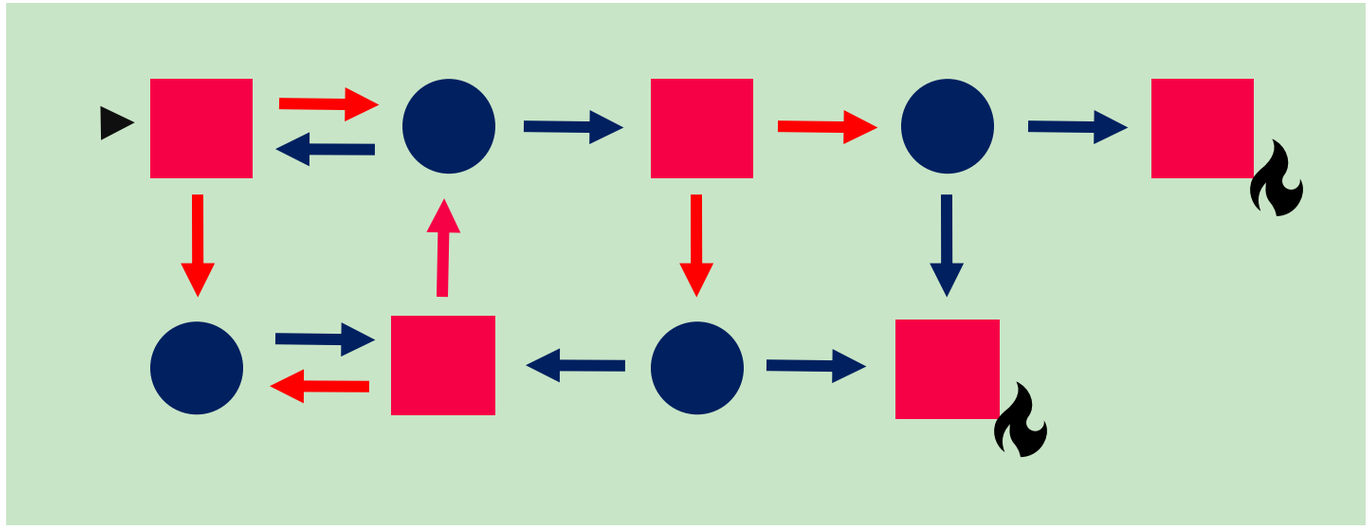
Reactive Synthesis as a Game

- 2-player game
- Infinite plays
- Winning conditions: safety, Büchi, parity, …
- Boolean reactive synthesis uses OBDD to represent game sates.

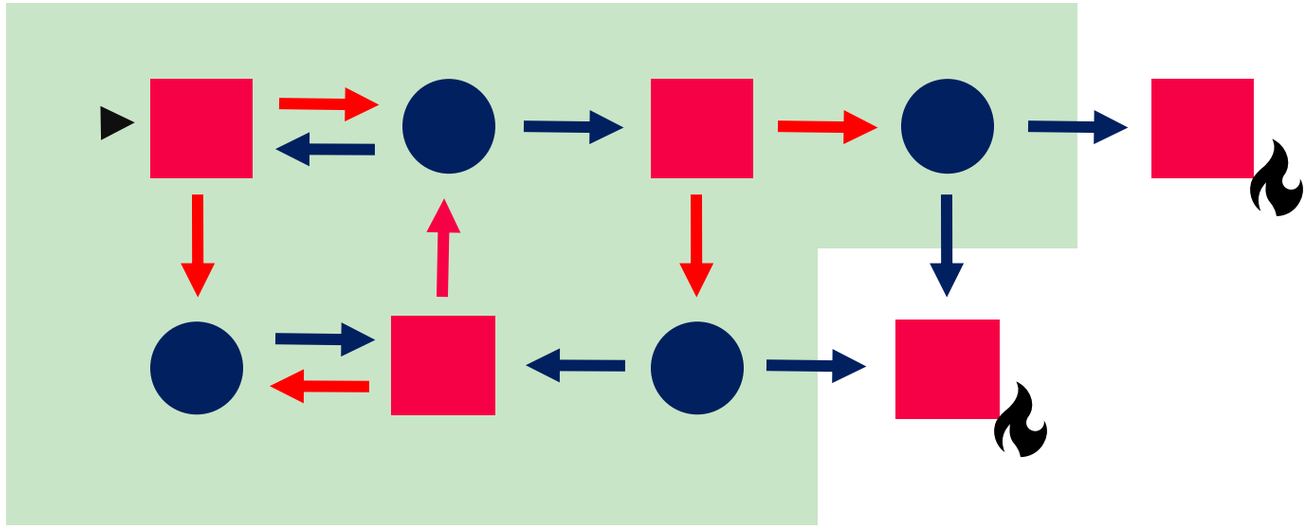
Fixpoints for safety game solving



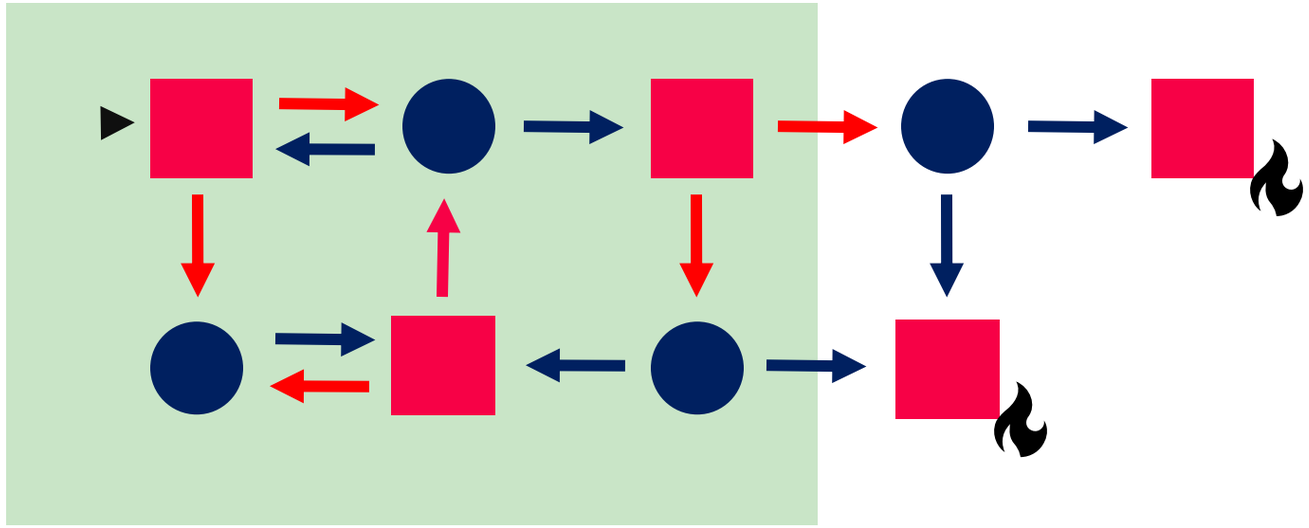
Fixpoints for safety game solving



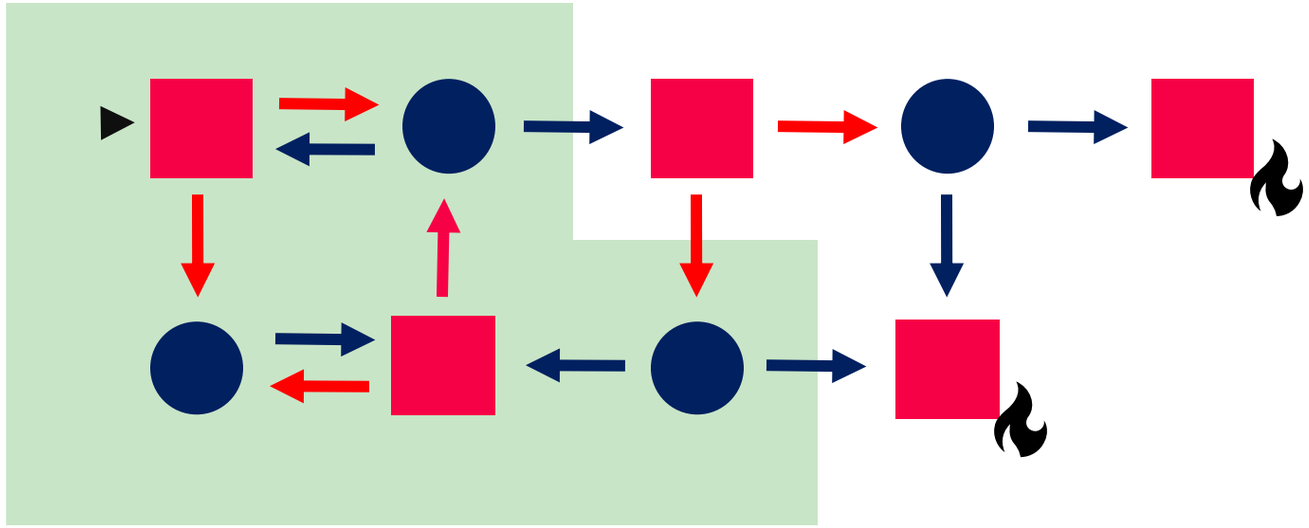
Fixpoints for safety game solving



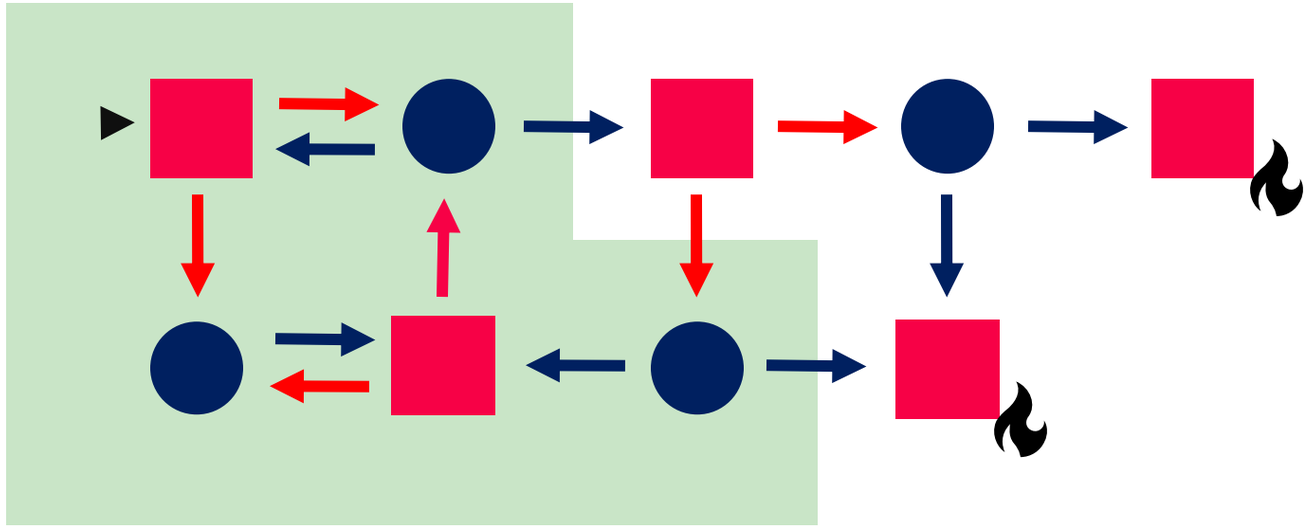
Fixpoints for safety game solving



Fixpoints for safety game solving



Fixpoints for safety game solving



Fixpoint reached

Infinite State Games

State variables: x

Input variables: i

Transition Predicates:

env(x,i) …..... how the environment can choose inputs i

sys(x,i,x') ….. how the system can choose the next state x'

Winning condition:

e.g. only visit states in SAFE

Example: Safety Game

$$\text{env} \quad = \quad i >= 0 \wedge i < 5$$

$$\text{sys} \quad = \quad x' = x + I \ \vee \ x' = x - i$$

$$\text{SAFE} \quad = \quad x >= 0 \wedge x < 100$$

```
assume {
  x >= 0;
  x < 100;
}
always assume {
  i >= 0;
  i < 5;
}
always guarantee {
 x >= 0;
 x < 100;
 [x <- x+i] || [x <- x-i];
}
```

Solving Infinite State Safety Games

Represent the sets of the classic game solving algorithm using SMT.

$$W = \text{True}$$

$$S = \text{safe states}$$

$$\text{while } W \text{ changed:}$$

$$\quad W = \text{pre}(W) \wedge S$$

$$\text{pre}(Y) = \text{QE}(\forall\, i: \text{env}(x,i) \Rightarrow \exists\, x' \in Y: \text{sys}(x,i,x'))$$

Stanly Samuel, Deepak D'Souza, Raghavan Komondoor: GenSys: a scalable fixed-point engine for maximal controller synthesis over infinite state spaces. ESEC/SIGSOFT FSE 2021

Example: Infinite State GR(1) Synthesis

$$\text{sys} =$$
$$(x \geq 0 \wedge x \leq 6 \wedge x' = x + 1 + d) \vee$$
$$(x \geq 0 \wedge x \leq 6 \wedge x' = x - 1 + d) \vee$$
$$(x \geq 0 \wedge x \leq 6 \wedge x' = x + d)$$

$$\text{env} = -1 \leq d \leq 1$$

$$\phi = (\square\Diamond(d < 0) \wedge \square\Diamond(d > 0)) \Rightarrow (\square\Diamond(x < 1) \wedge \square\Diamond(x > 5))$$

Example: Infinite State GR(1) Synthesis

```
if ( (k = 1 ∧ x < 1) ∨ (k = 2 ∧ x > 5) ) {
    k := k ⊕ 1
}
if ( k = 1 ) {
    if ( x + d >= 1 ) {
        x := x − 1 + d
    } else {
        x := x + d
    }
}
else if ( k = 2 ) {
    if ( x + d <= 5 ) {
        x := x + 1 + d
    } else {
        x := x + d
    }
}
```

Infinite State GR(1) Synthesis

- Adapted classic GR(1) algorithm to SMT
- Improved scalability using simplification and redundancy elimination
- Minimization reduces size of the target programs

$$
\begin{aligned}
&Z = \mathrm{True}\\
&\text{while } Z \text{ changed:}\\
&\quad \text{for } j \in [1, n]:\\
&\quad\quad Y = \mathrm{False}\\
&\quad\quad \text{while } Y \text{ changed:}\\
&\quad\quad\quad \mathrm{start} = (\mathrm{Js}_j \wedge \mathrm{pre}(Z)) \vee \mathrm{pre}(Y)\\
&\quad\quad\quad Y = \mathrm{False}\\
&\quad\quad\quad \text{for } i \in [1, m]:\\
&\quad\quad\quad\quad X = Z\\
&\quad\quad\quad\quad \text{while } X \text{ changed:}\\
&\quad\quad\quad\quad\quad X = \mathrm{start} \vee (\neg \mathrm{Je}_i \wedge \mathrm{pre}(X))\\
&\quad\quad\quad\quad Y = Y \vee X\\
&\quad\quad Z = Y
\end{aligned}
$$

Benedikt Maderbacher, Felix Windisch, Roderick Bloem: Synthesis from Infinite-State Generalized Reactivity(1) Specifications. ISoLA 2024 (to appear)

Parameterized Synthesis

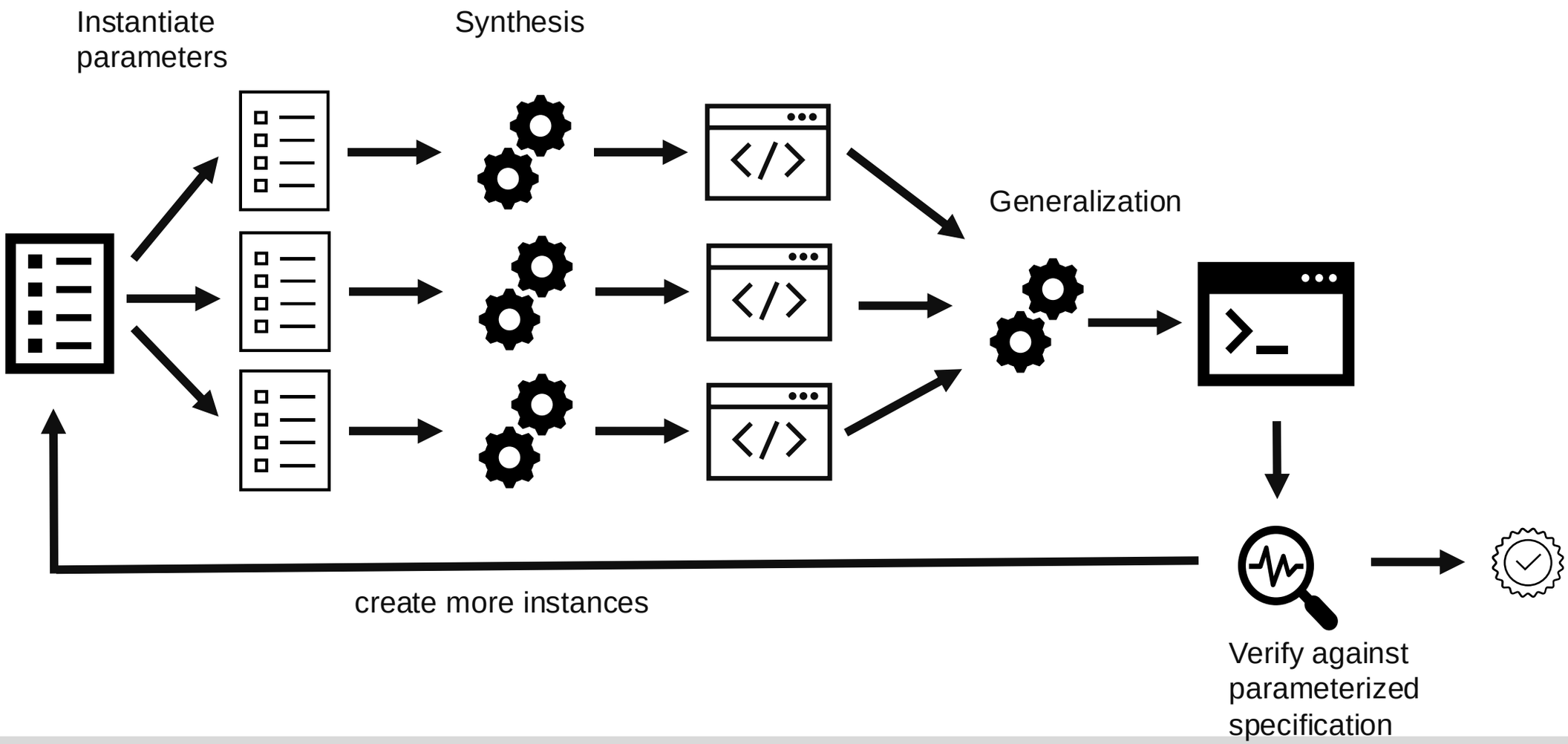
(WIP)

Setting

- Specification with parameters
  - size of an arena
  - number of floors in a building
  - number of clients

- Find one program that works for all instantiations of the parameters.

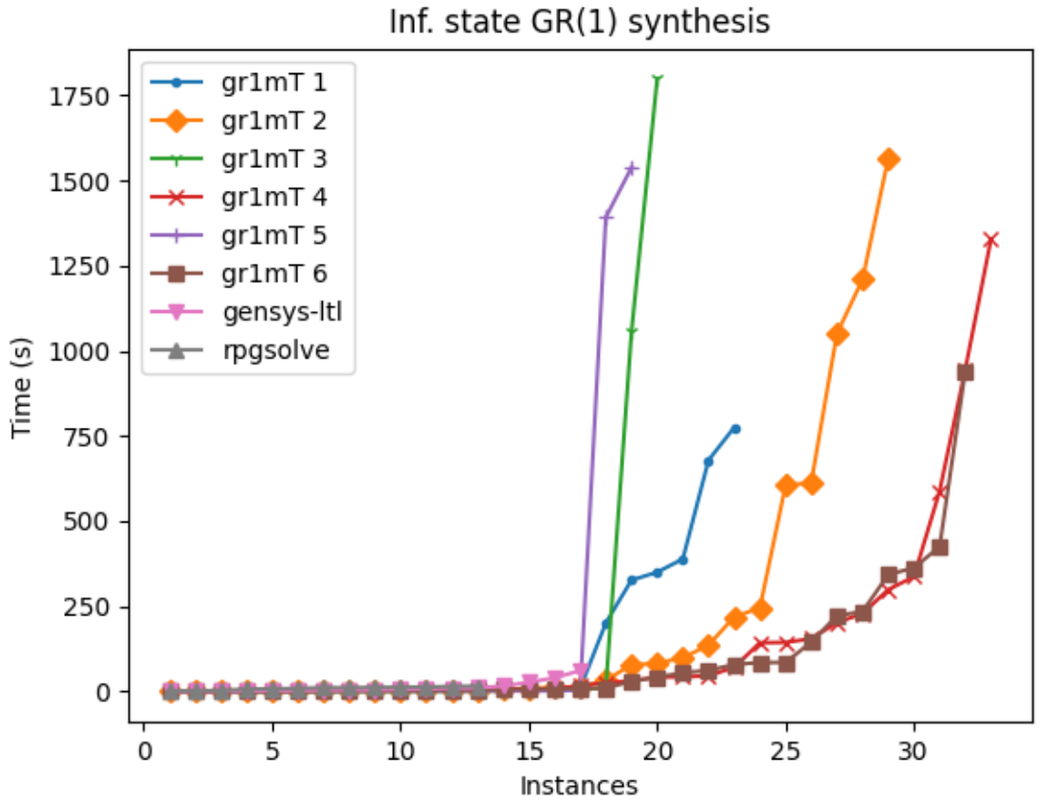
Building Parameterized Strategies



Conclusion

- Reactive synthesis can be extended to non-Boolean settings.
- Two major solving techniques:
  - abstraction to LTL
  - game solving using SMT
- Future work: parameterized synthesis

Experimental Results



Experimental Results