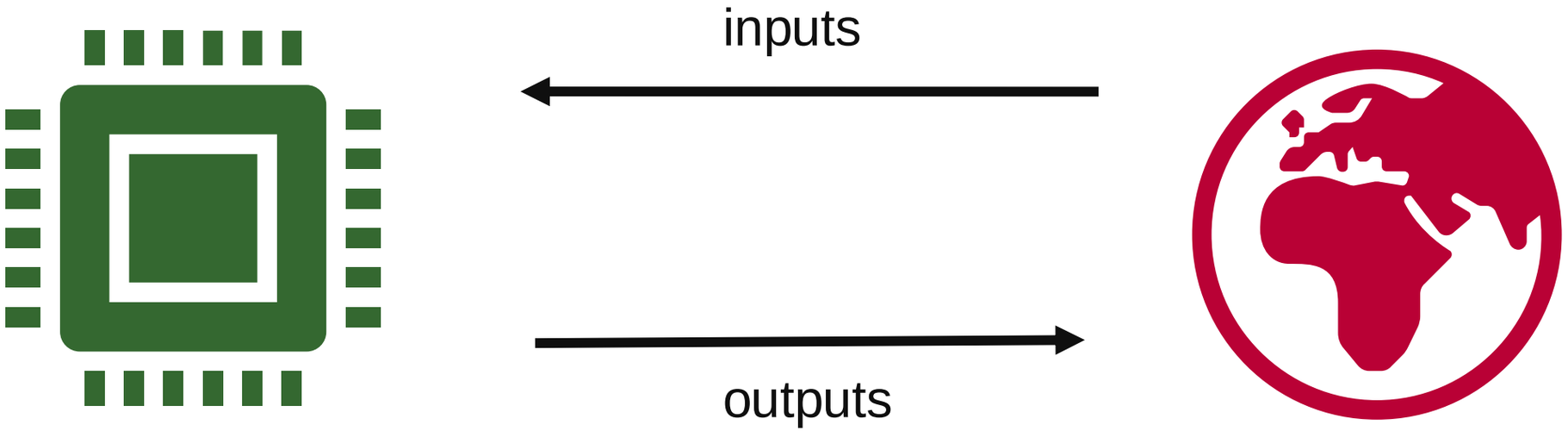
Reactive Synthesis modulo Theories using Abstraction Refinement

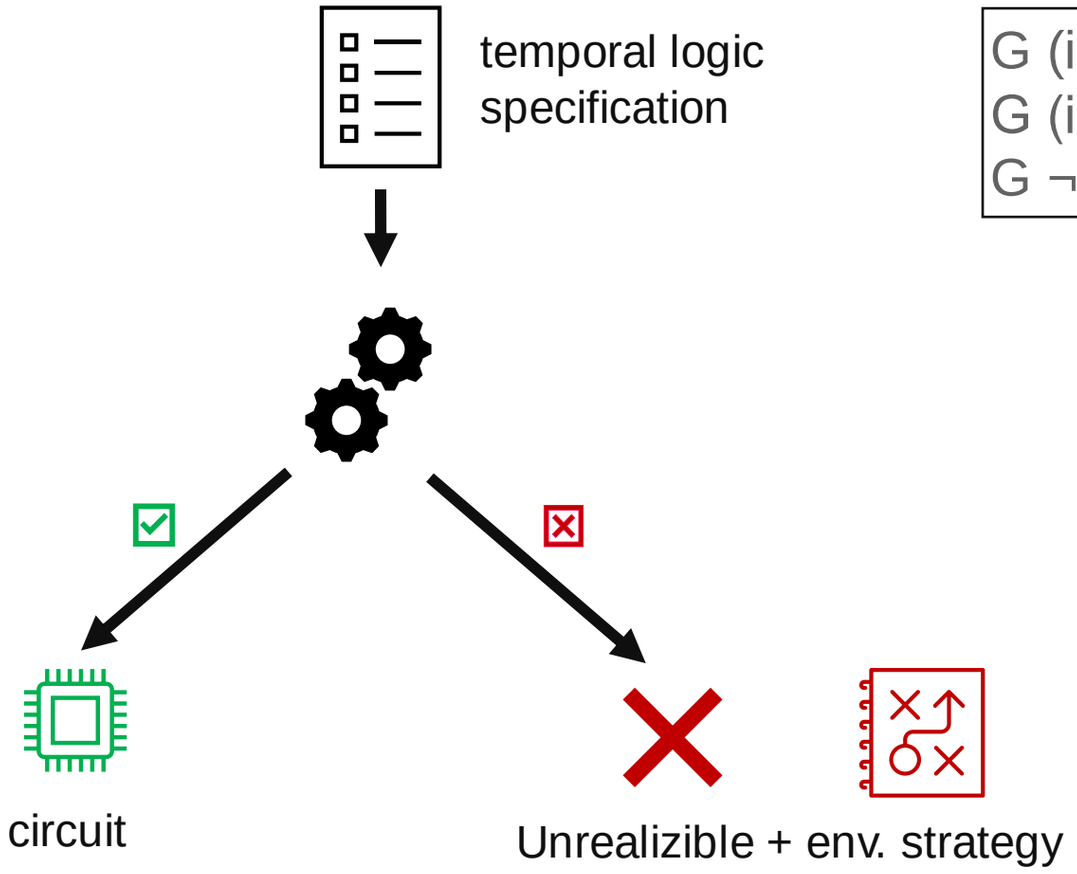
Benedikt Maderbacher and Roderick Bloem
Graz University of Technology

FMCAD 22
21st October 2022

Reactive Systems



Reactive Synthesis



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

Temporal Stream Logic modulo 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) {
    in := 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) {
    in := 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;
}
```

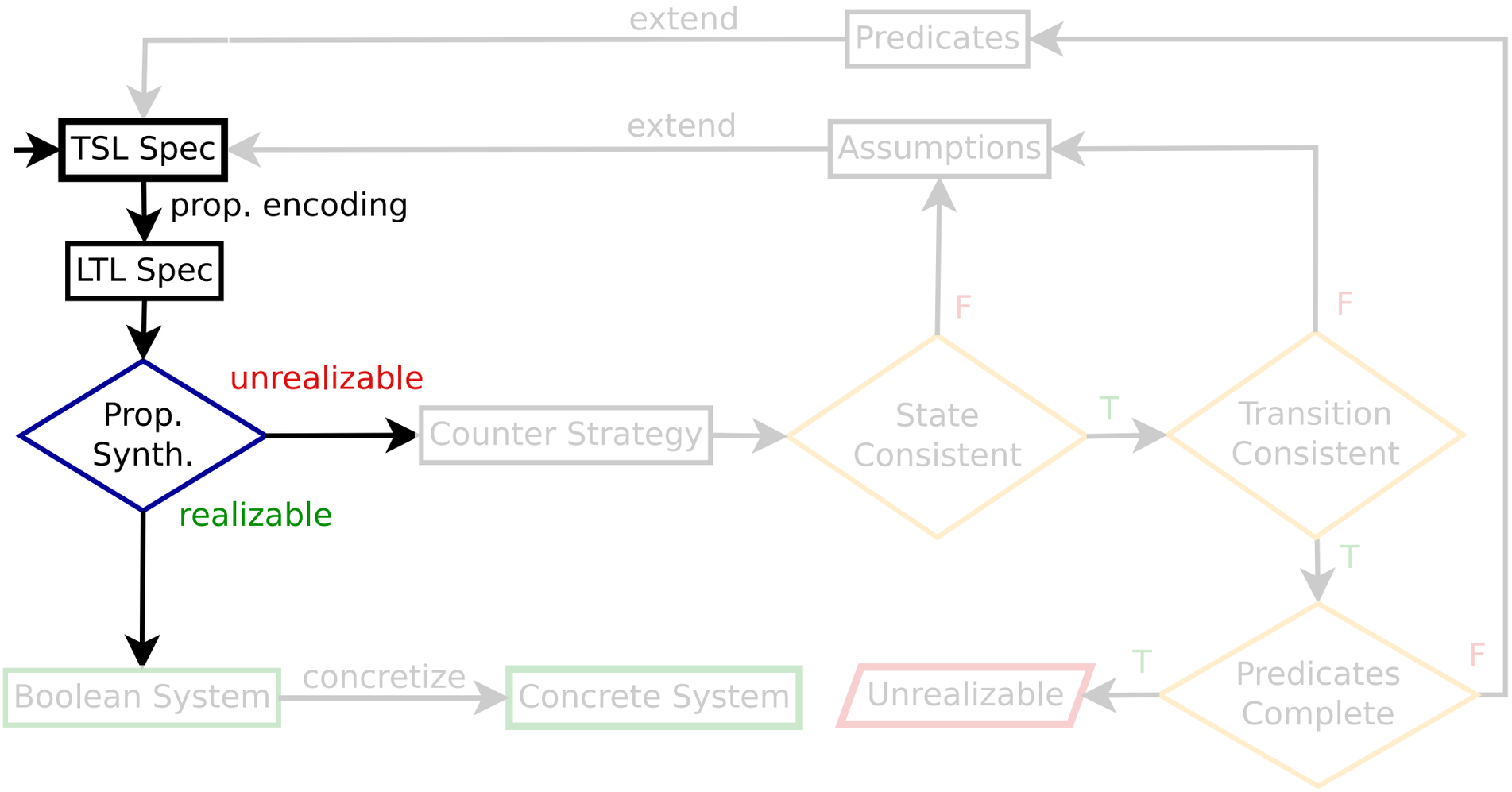
Implementation alternative

```
while (true) {
    in := readInput();
    if(x+i<100) {
        x := x + i;
    }
    else {
        x := x - i;
    }
    sendOutput(x);
}
```

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

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

Synthesis Algorithm



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];
}
```

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

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

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

Propositional Encoding

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

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

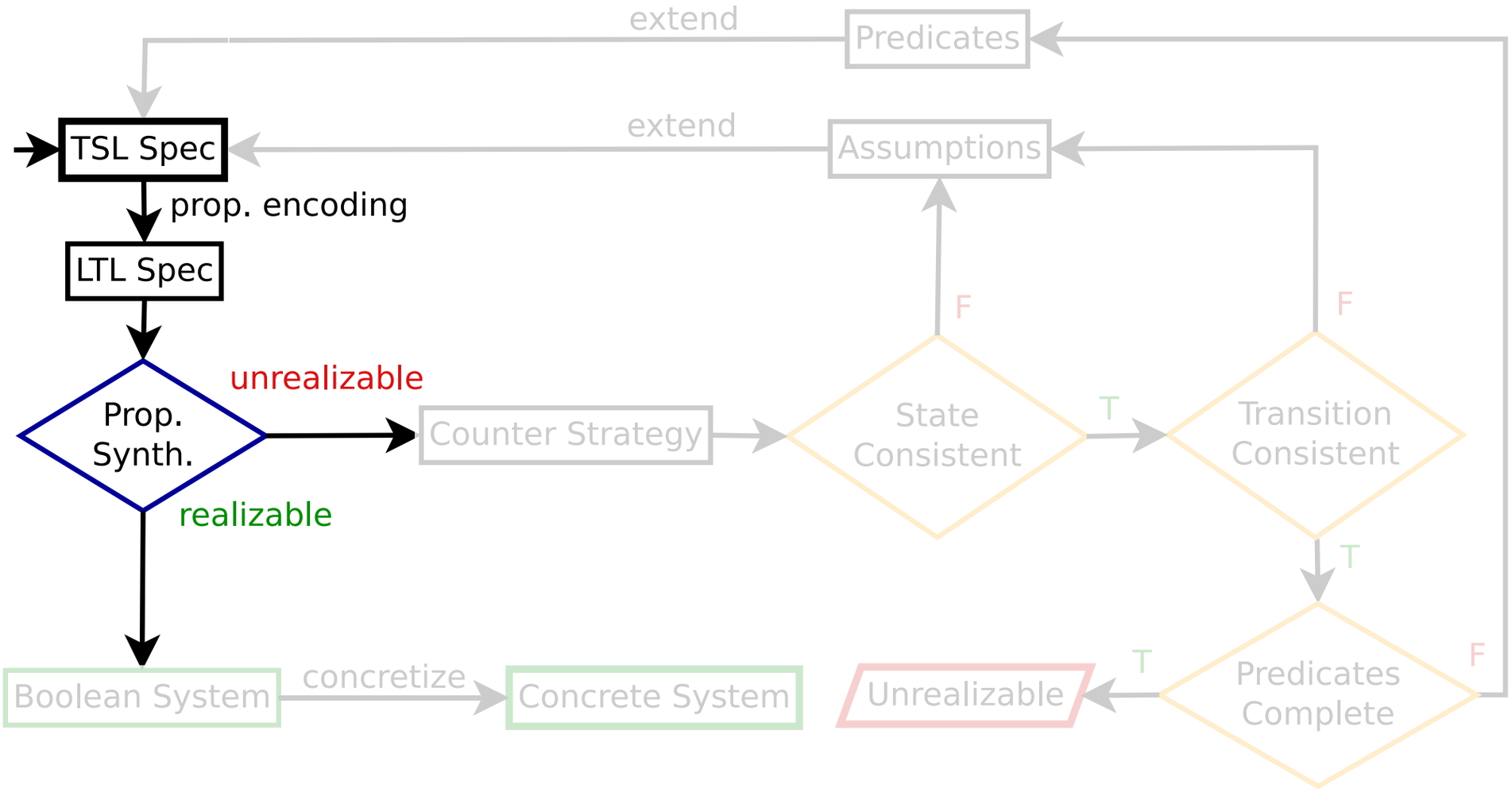
```
always guarantee {
  x >= 0;
  x < 100;
  [x <- x+i] || [x <- x-i];
}
```

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

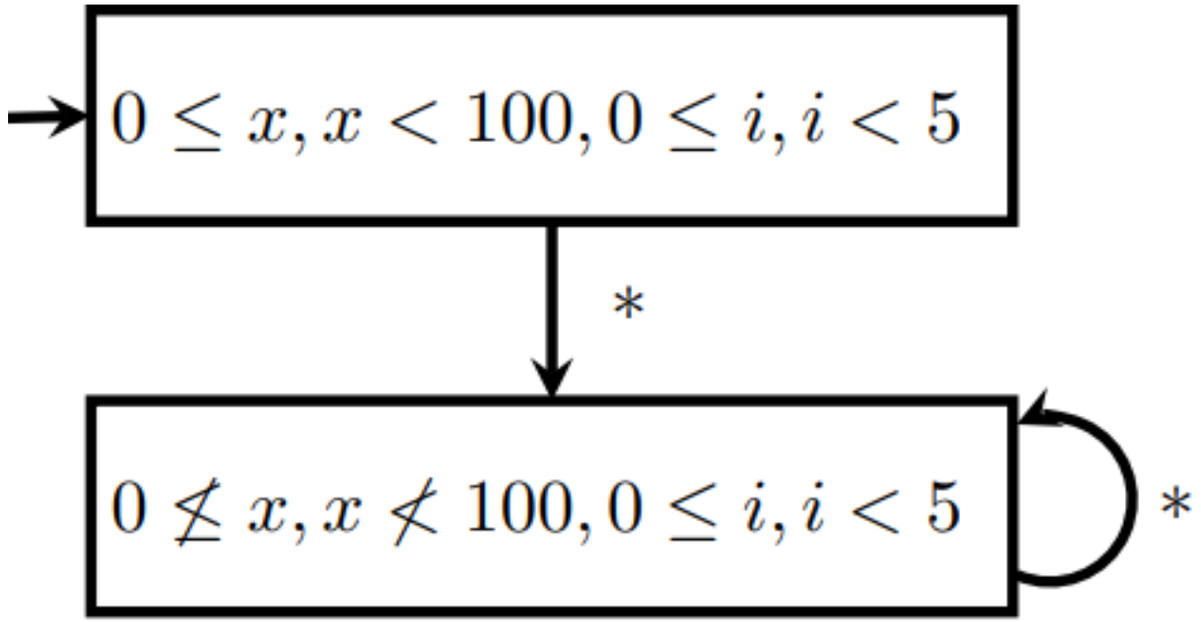
$$\big((p_{0\leq x} \wedge p_{x<100} \wedge \square(p_{0\leq i} \wedge p_{i<5})) \rightarrow$$

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

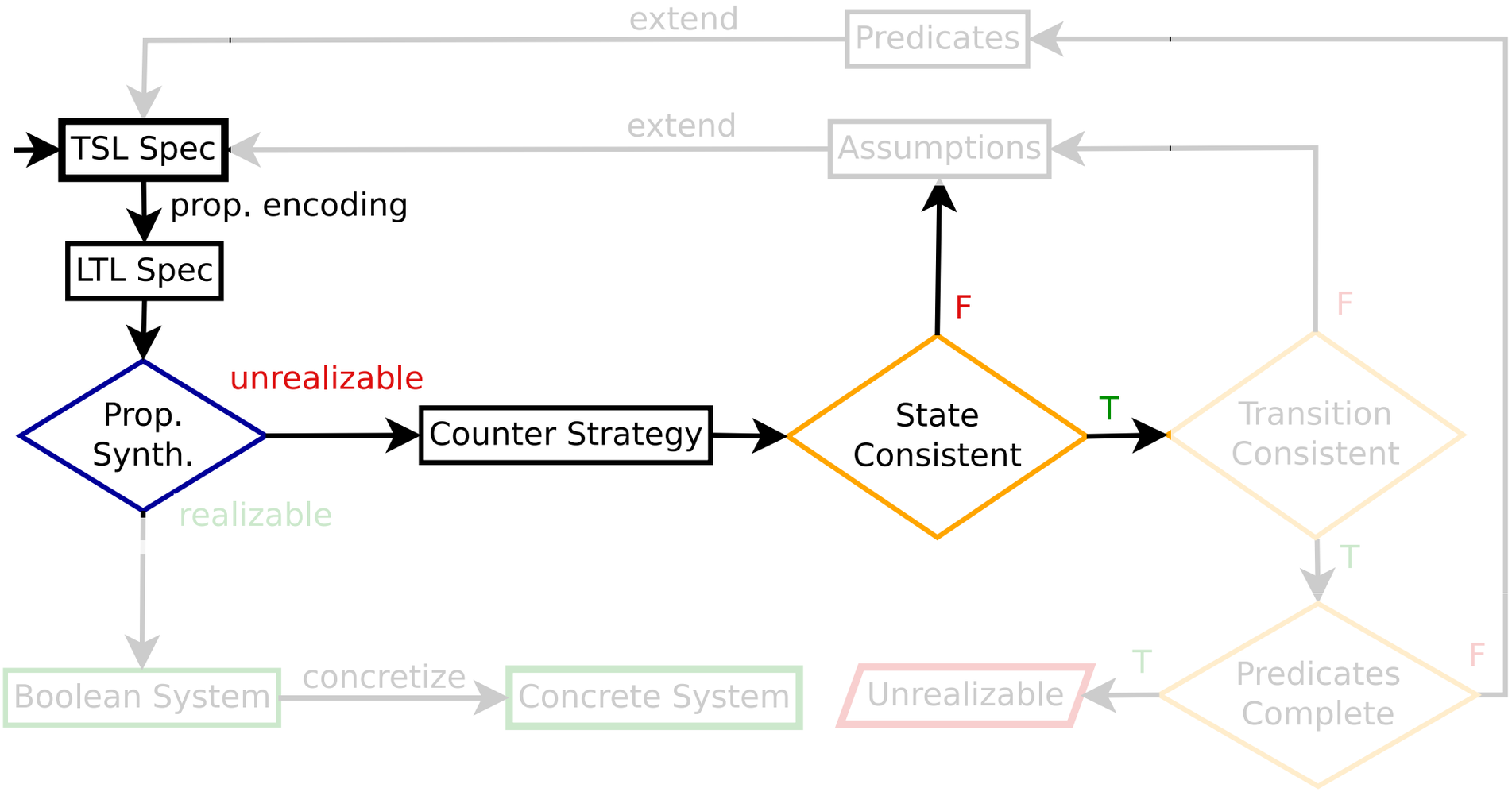
Synthesis Algorithm



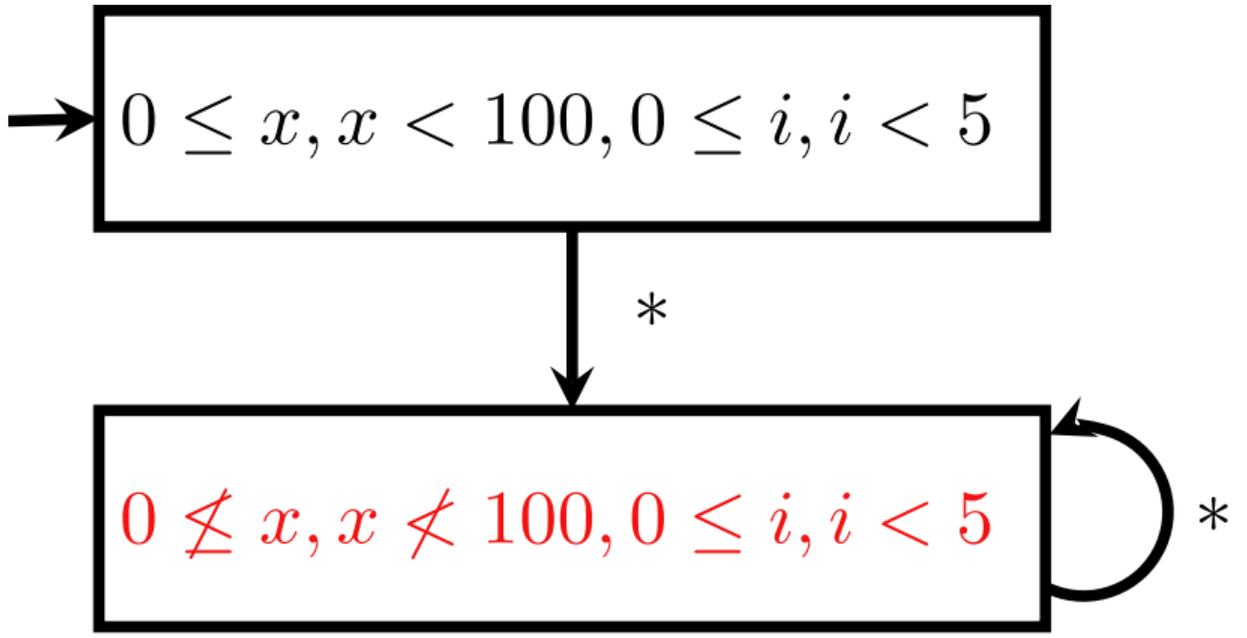
Counter Strategy



Synthesis Algorithm



Counter Strategy (Inconsistent State Outputs)

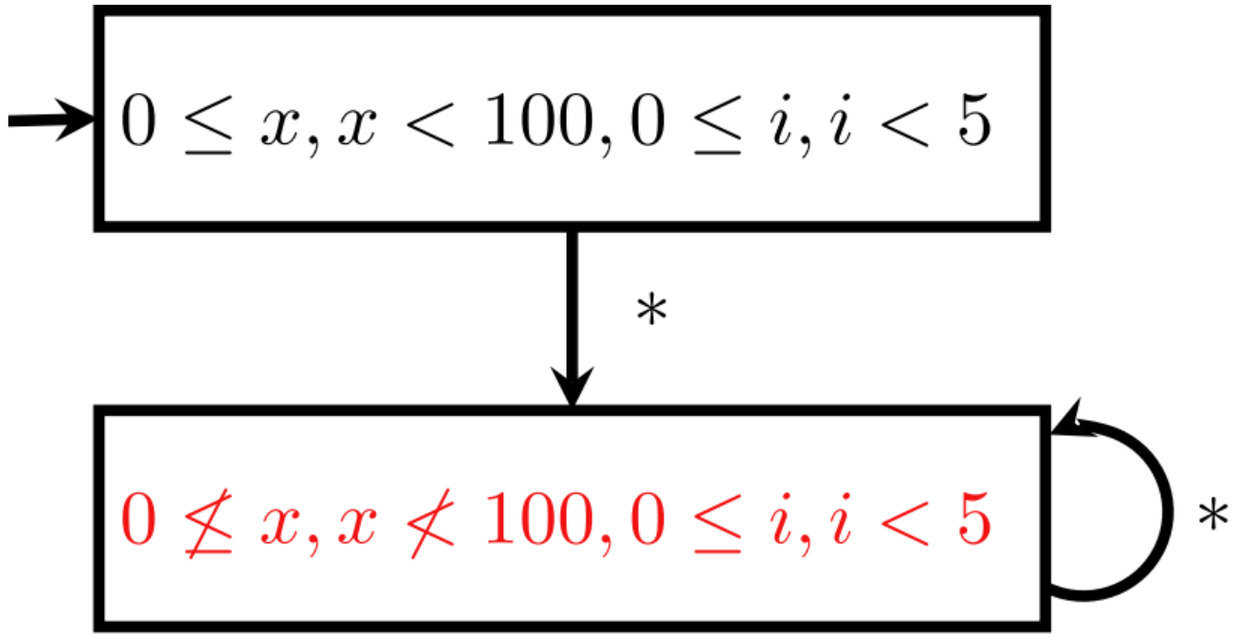


Contradiction:

$x < 0$,

$x \geq 100$

Counter Strategy (Inconsistent State Outputs)



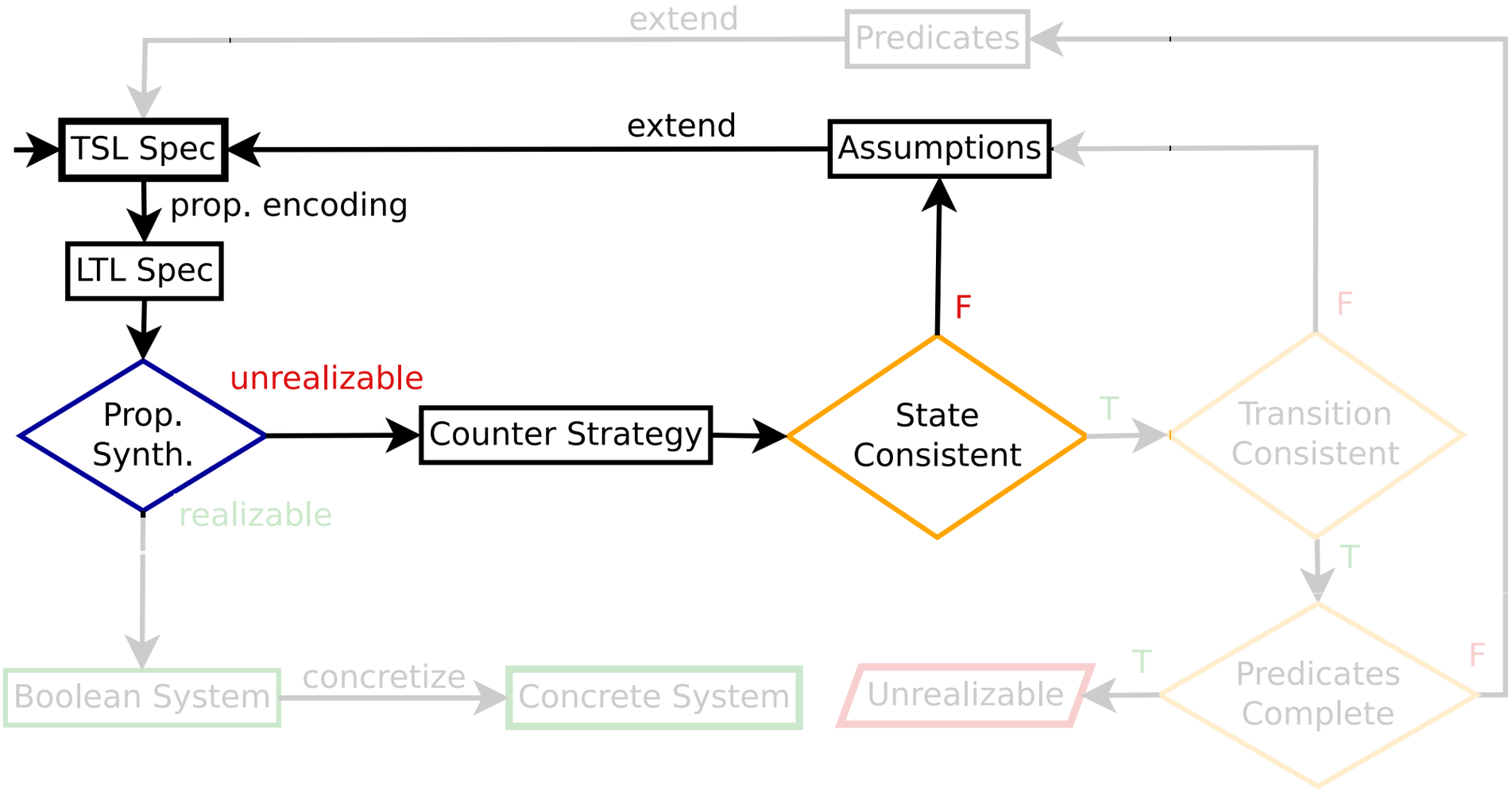
Contradiction:

$x < 0$,

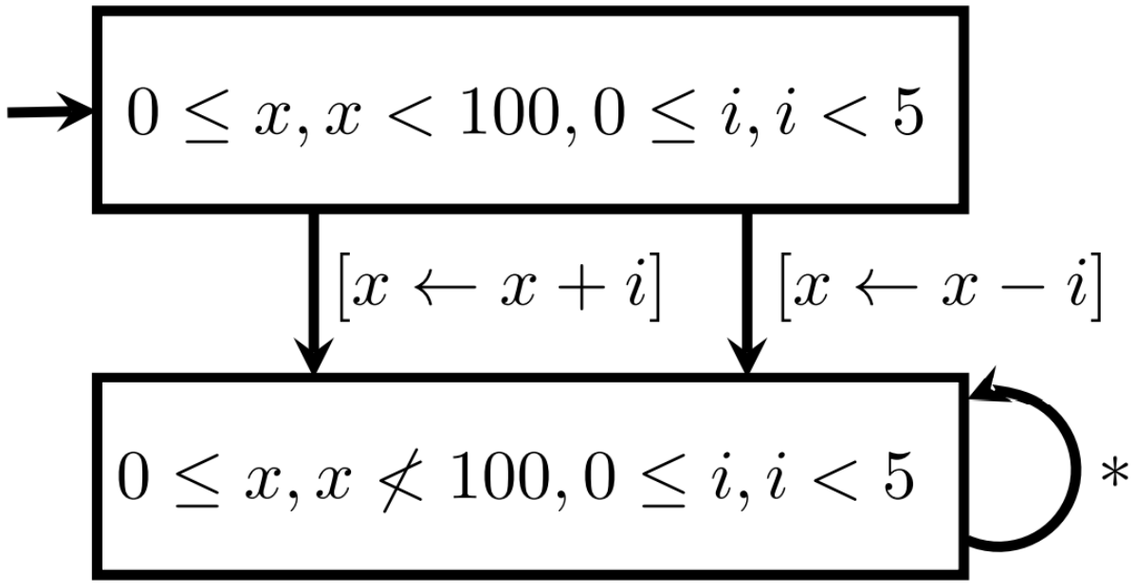
$x \geq 100$

Added assumption: $G \neg(x<0 \wedge x \geq 100)$

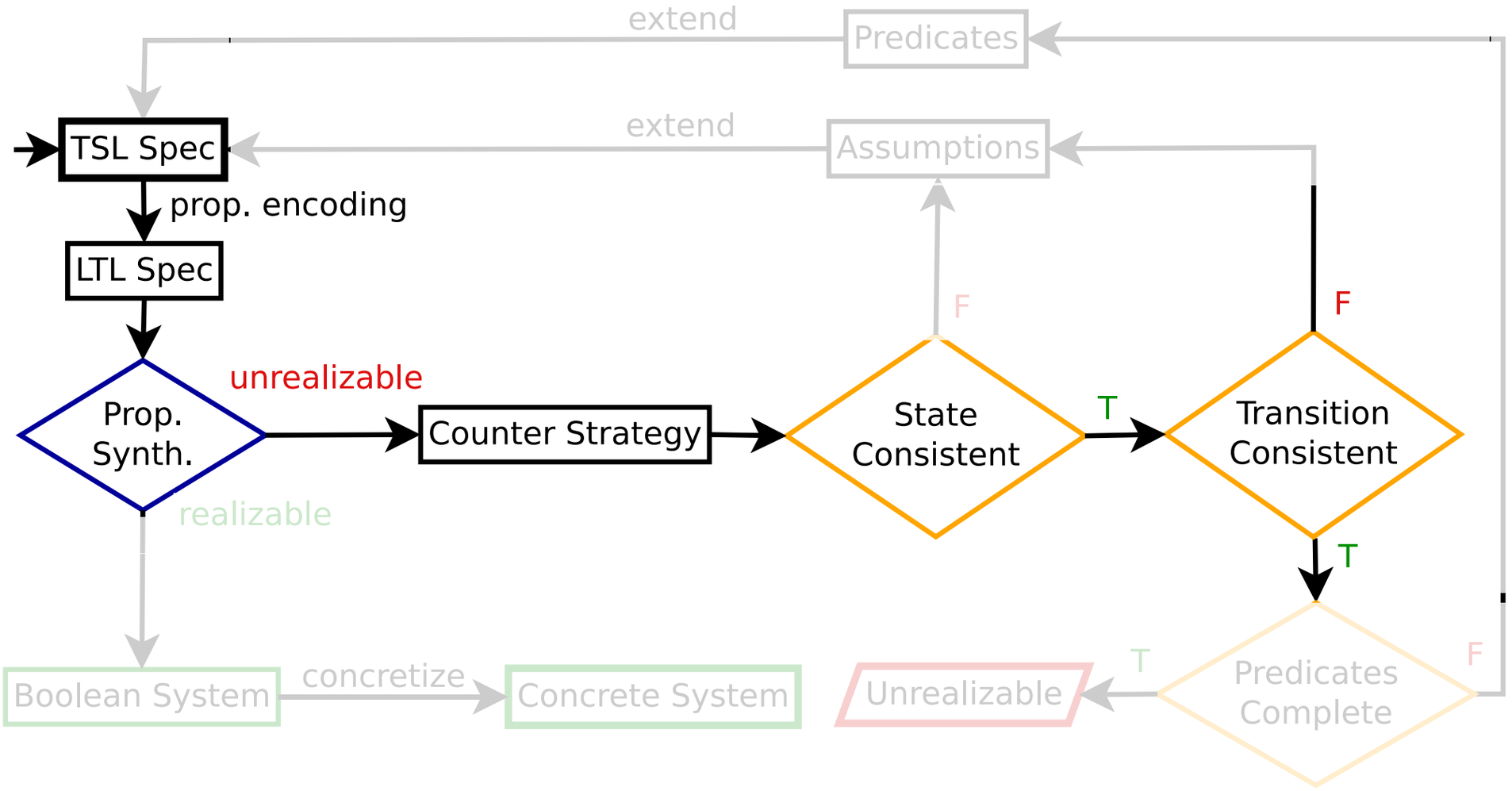
Synthesis Algorithm



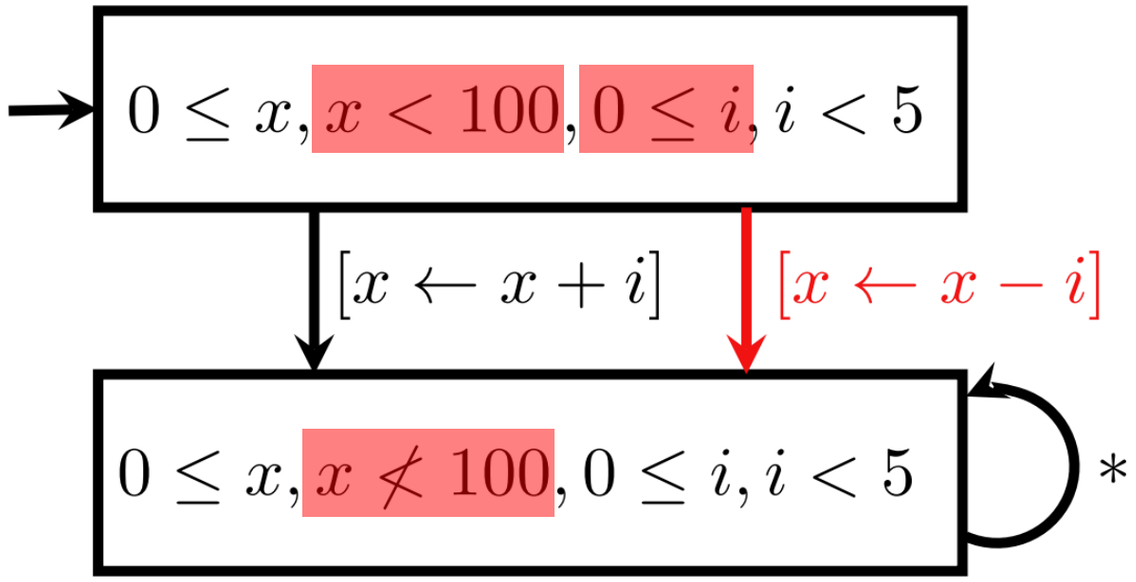
Counter Strategy 2



Synthesis Algorithm



Counter Strategy 2 (Inconsistent Transition)



Contradiction:

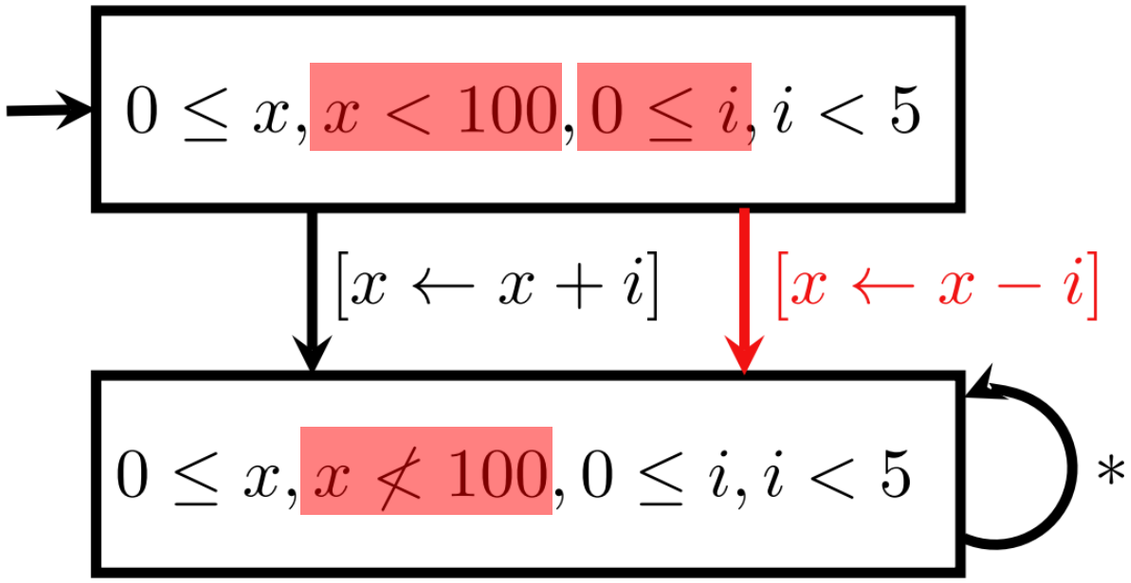
$x < 100,$

$0 \leq i,$

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

$x' \geq 100$

Counter Strategy 2 (Inconsistent Transition)



Contradiction:

$x < 100,$

$0 \leq i,$

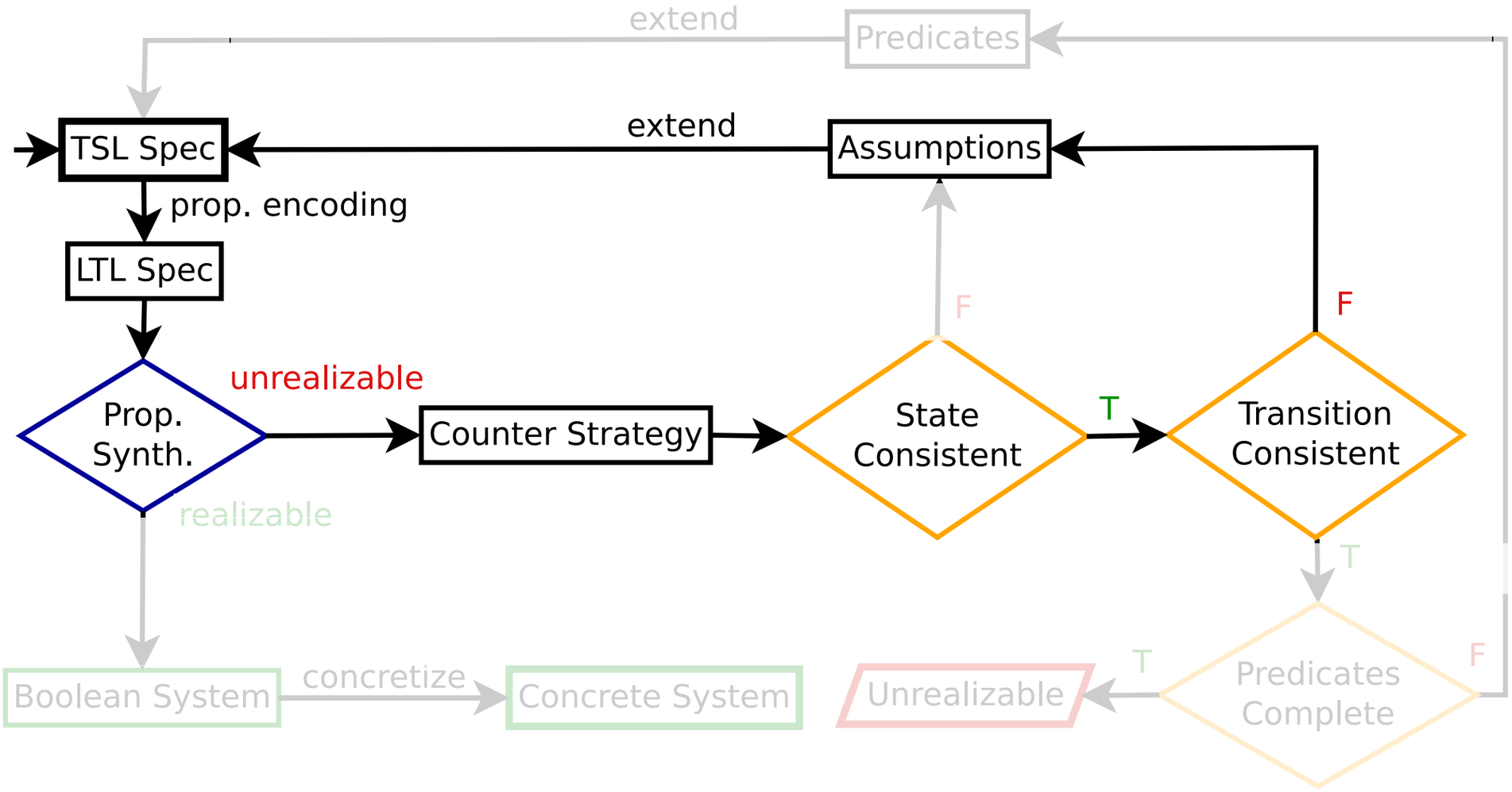
$x' = x\text{-}i,$

$x' \geq 100$

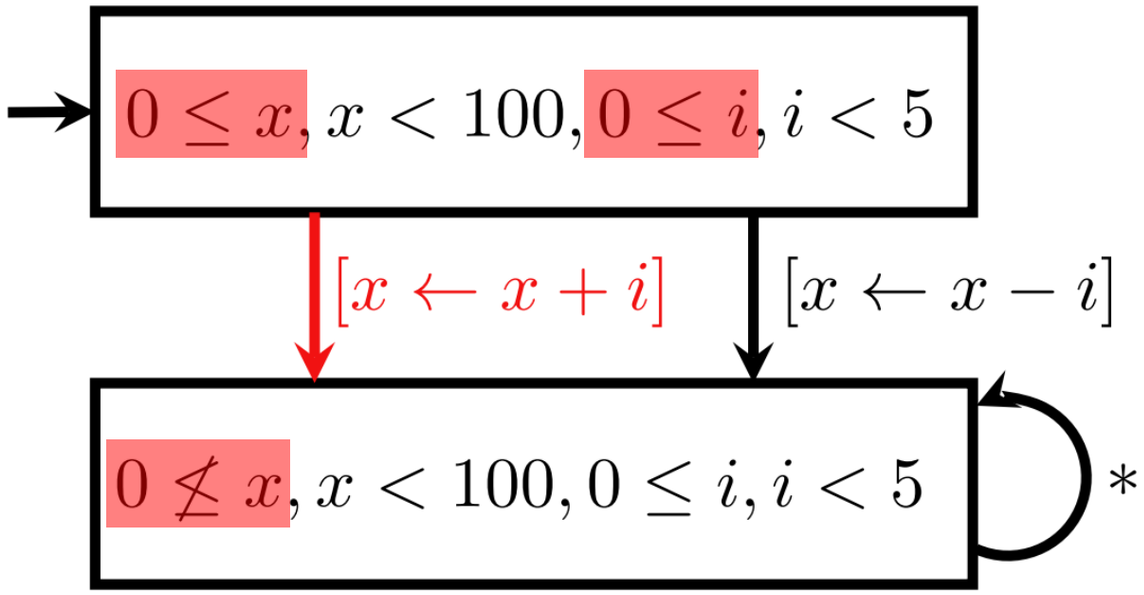
Added assumption:

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

Synthesis Algorithm



Counter Strategy 3 (Inconsistent Transition)



Contradiction:

$0 \leq x,$

$0 \leq i,$

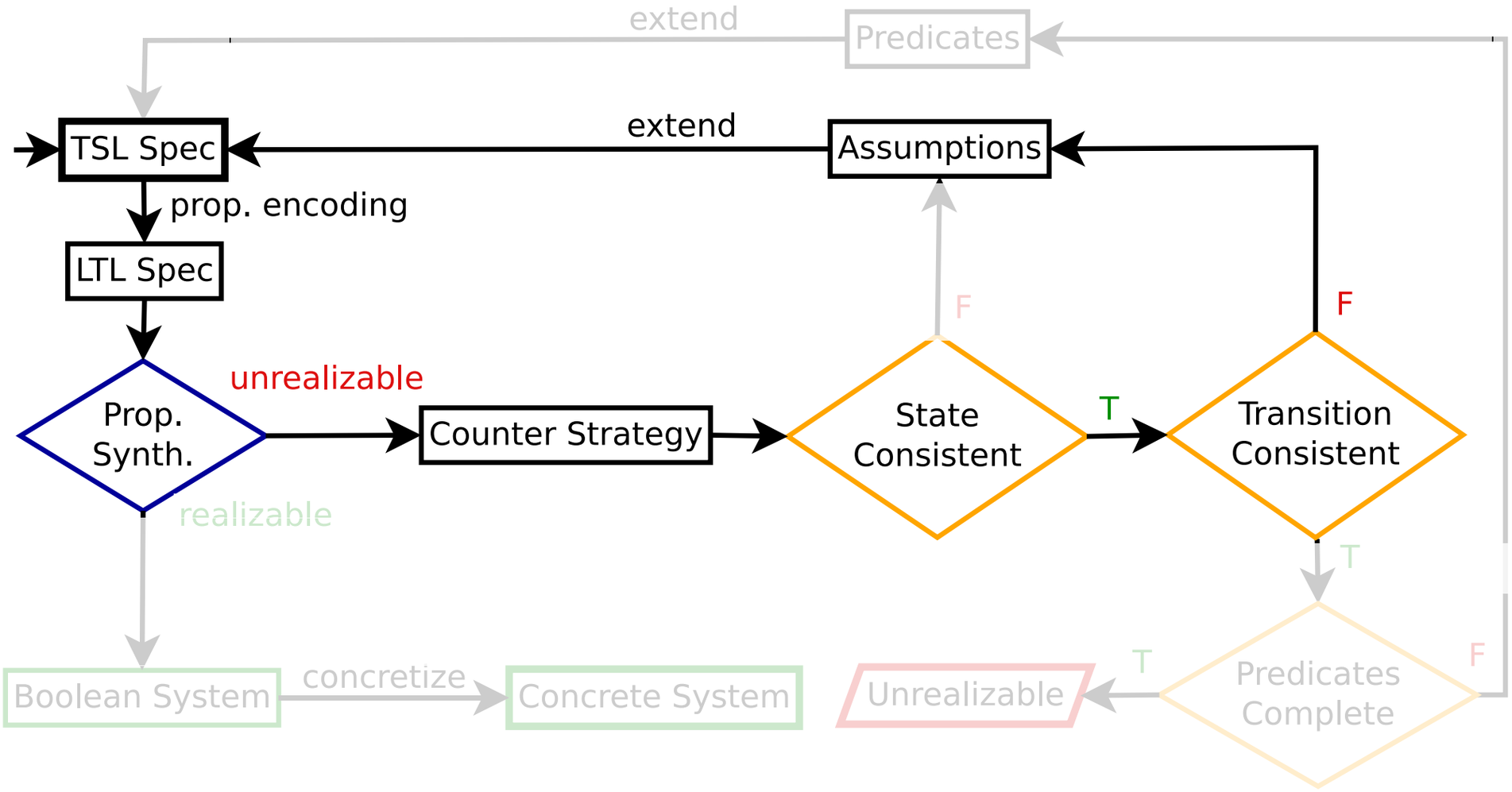
$x' = x+i,$

$x' < 0$

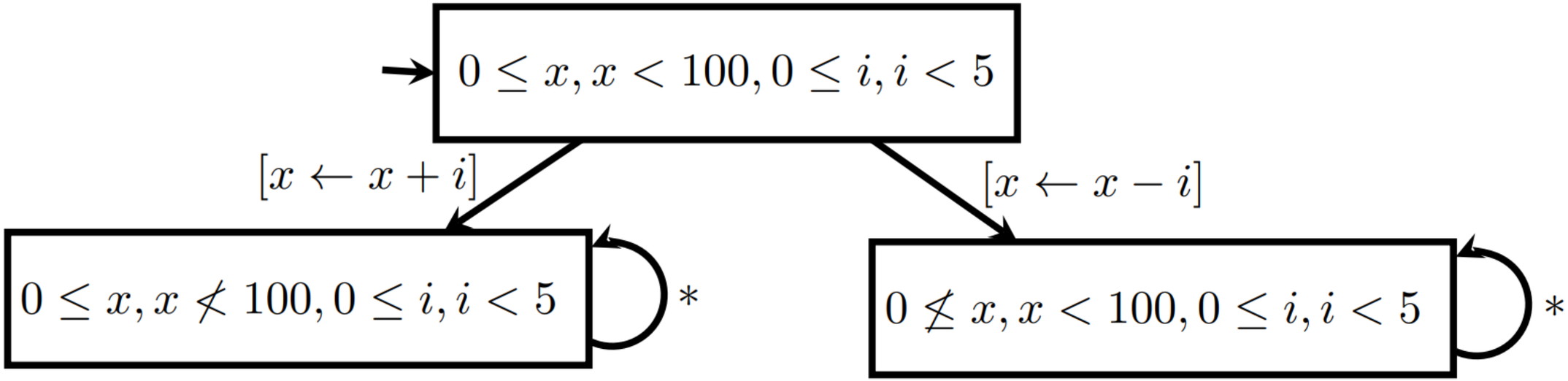
Added assumption:

$G((0 \leq x \wedge 0 \leq i \wedge [x \leftarrow x+i]) \rightarrow X(0 \leq x))$

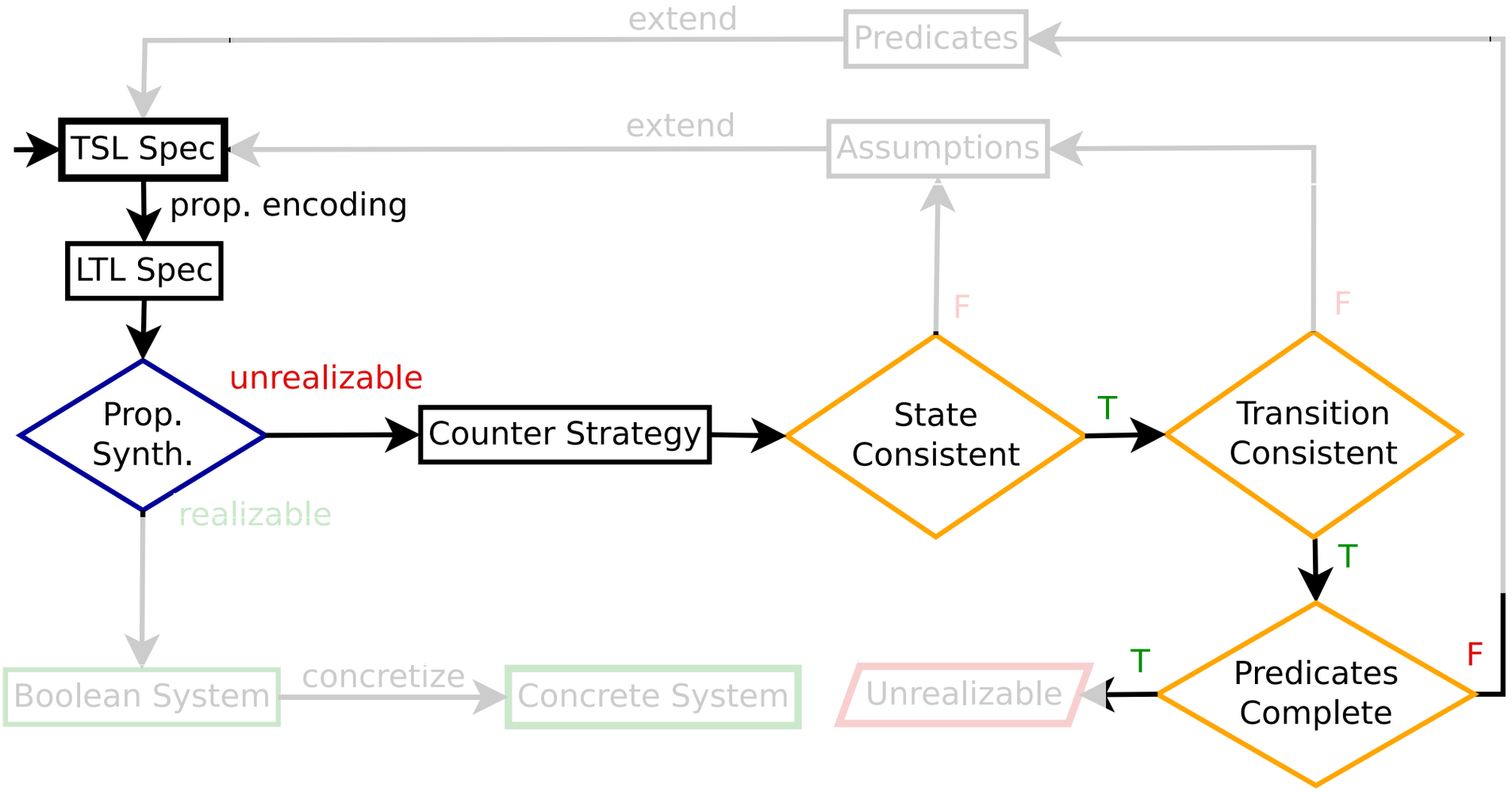
Synthesis Algorithm



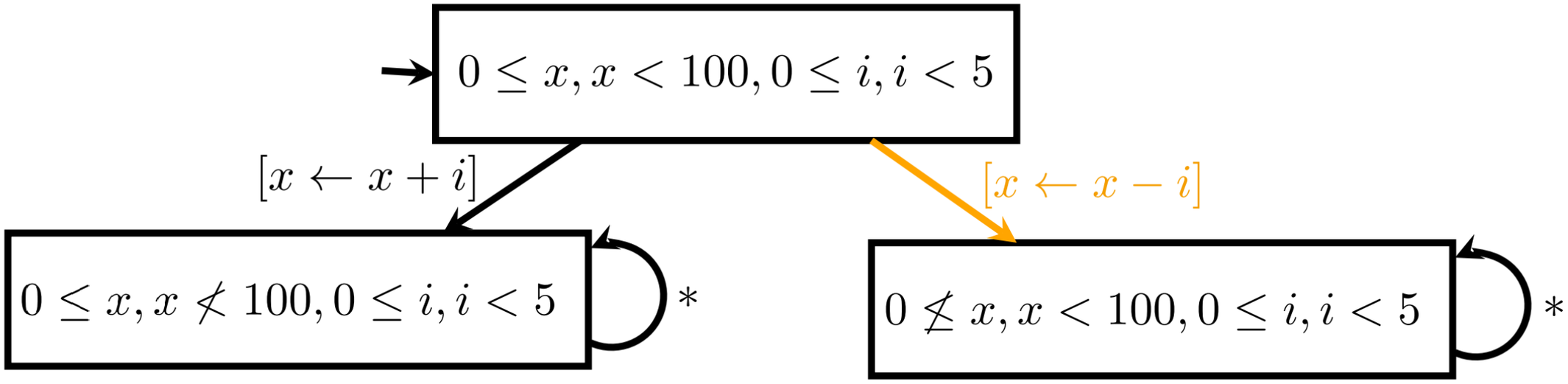
Counter Strategy 4



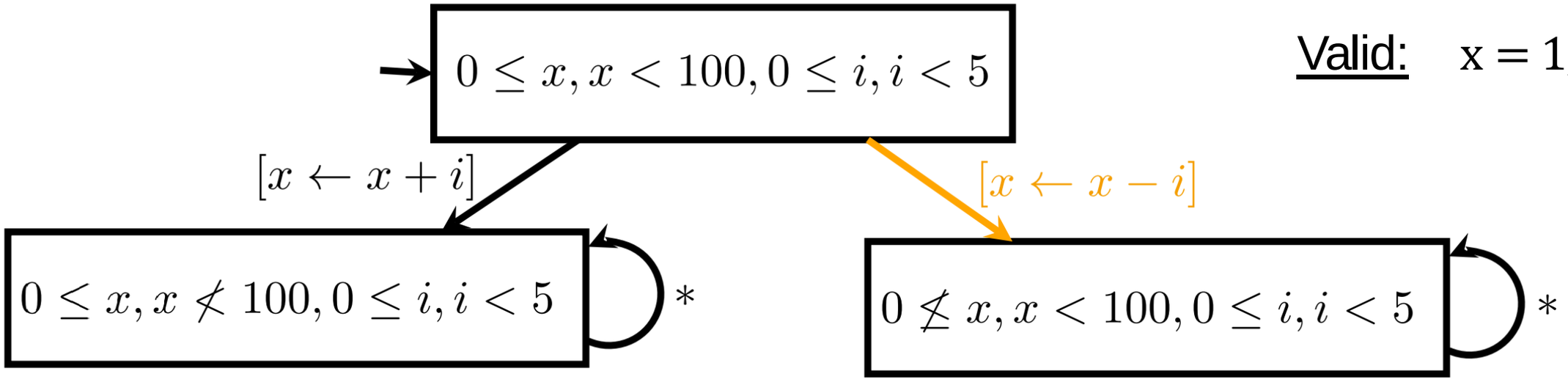
Synthesis Algorithm



Counter Strategy 4

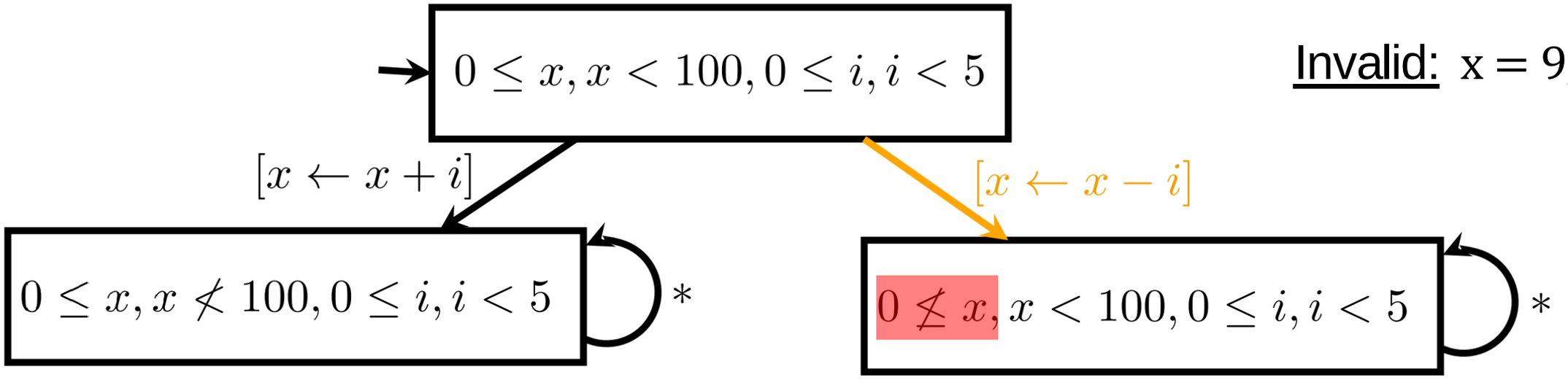


Counter Strategy 4



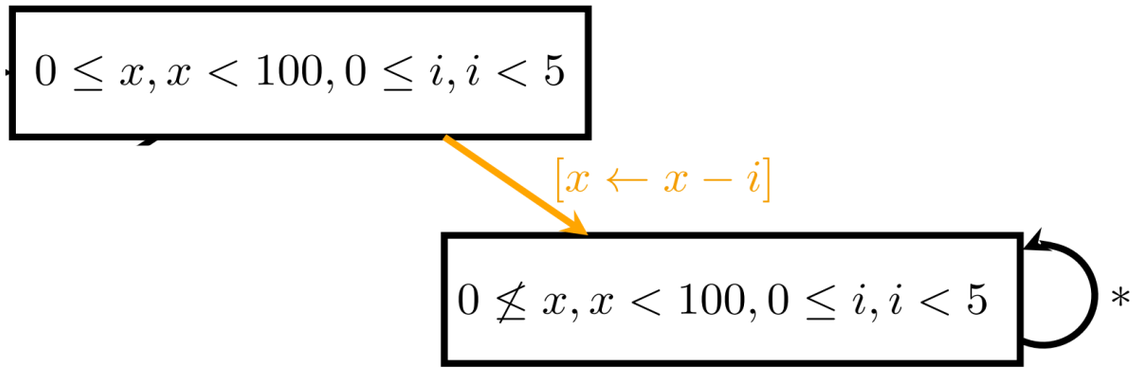
Valid: $x = 1, i = 4$

Counter Strategy 4



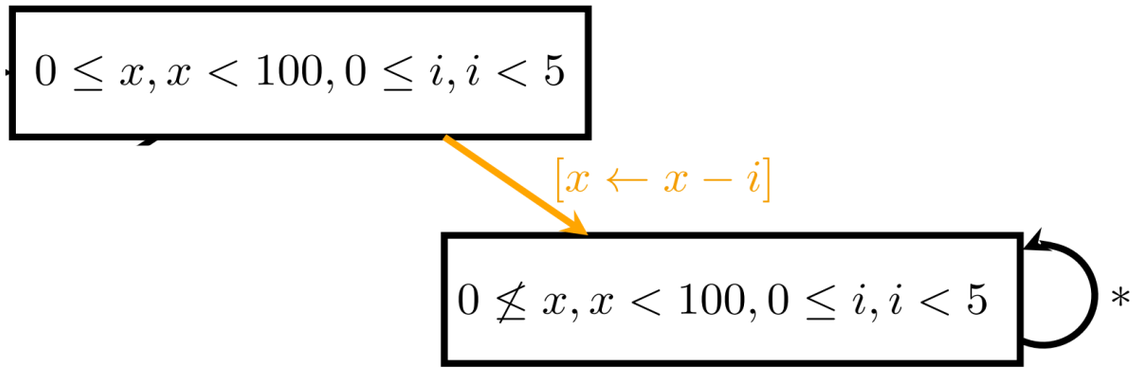
Invalid: $x = 9, i = 1$

Discovering Predicates



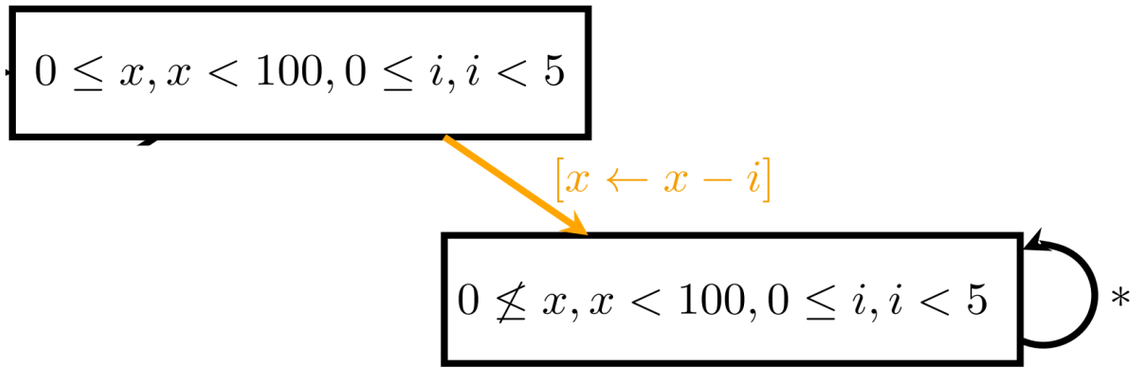
weakest precondition(x ← x - i,
∃ i'. 0>x ∧ x < 100 ∧ 0 ≤ i' ∧ i' < 5)

Discovering Predicates



weakest precondition(x ← x - i,
∃ i'. 0>x ∧ x < 100 ∧ 0 ≤ i' ∧ i' < 5) =
∃ i'. 0>x - i ∧ x - i < 100 ∧ 0 ≤ i' ∧ i' < 5

Discovering Predicates

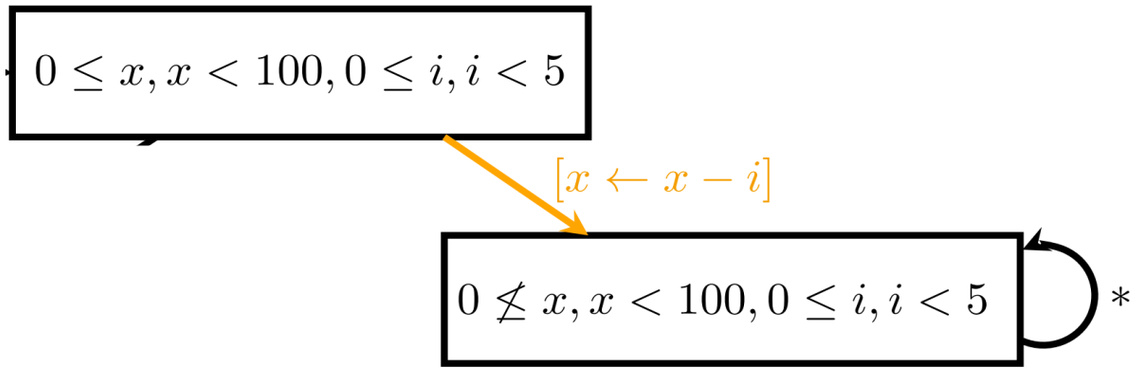


weakest precondition(x ← x - i,
∃ i'. 0>x ∧ x < 100 ∧ 0 ≤ i' ∧ i' < 5) =
∃ i'. 0>x - i ∧ x - i < 100 ∧ 0 ≤ i' ∧ i' < 5
=

quantifier elimination:

0>x - i ∧ x - i < 100

Discovering Predicates

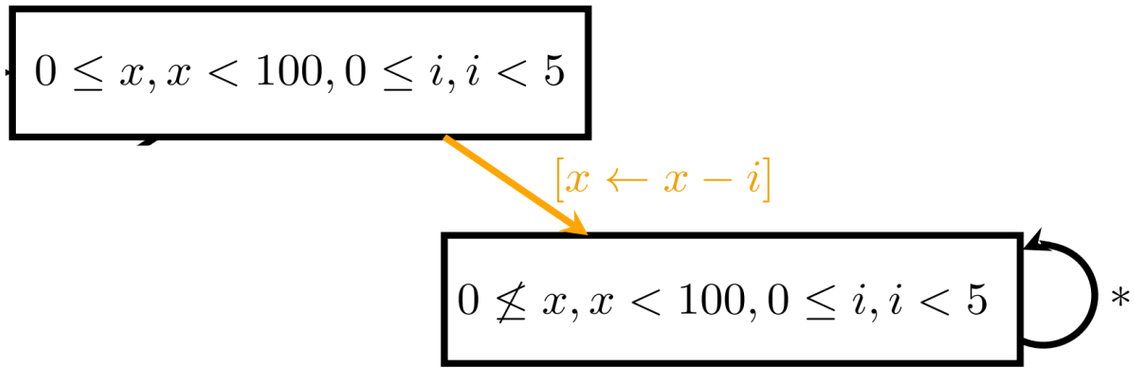


SAT?

$(0 \leq x \wedge x < 100 \wedge 0 \leq i \wedge i < 5) \wedge$
$x - i \geq 100$

$(0 \leq x \wedge x < 100 \wedge 0 \leq i \wedge i < 5) \wedge$
$0 \leq x - i$

Discovering Predicates



$(0 \leq x \wedge x < 100 \wedge 0 \leq i \wedge i < 5) \wedge$
$0 \leq x - i$

$x = 10, \quad i = 1$

Generalizing Assumptions

$$0 \leq x \wedge x < 100 \wedge 0 \leq i \wedge i < 5 \wedge 0 \leq x - i \wedge$$
$$x' = x\text{-}i \wedge$$
$$0 > x' \wedge x' < 100 \wedge 0 \leq i' \wedge i' < 5$$

Generalizing Assumptions

$$0 \leq x \wedge x < 100 \wedge 0 \leq i \wedge i < 5 \wedge 0 \leq x - i \wedge$$
$$x' = x - i \wedge$$
$$0 > x' \wedge x' < 100 \wedge 0 \leq i' \wedge i' < 5$$

UNSAT core

Generalizing Assumptions

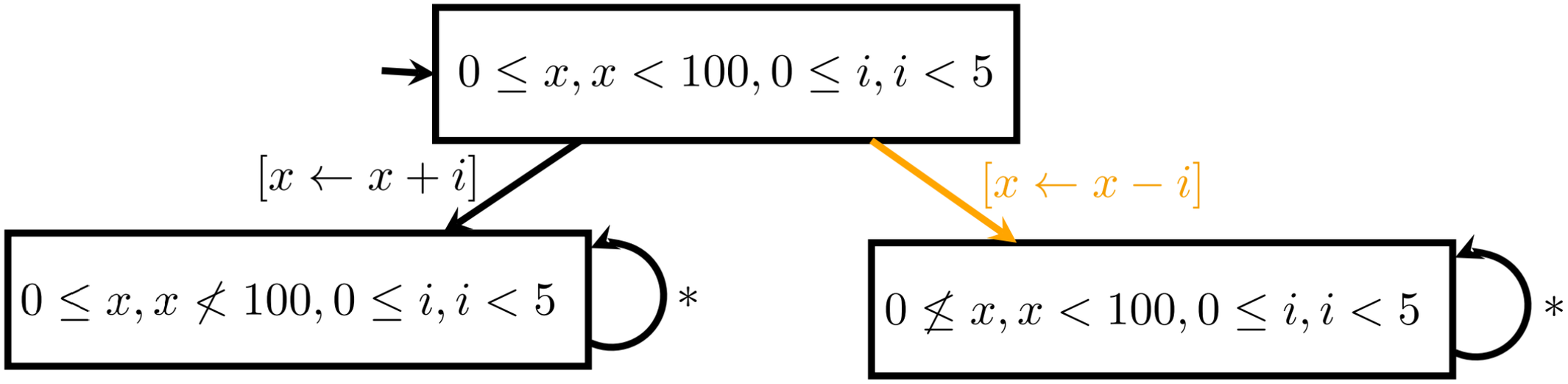
$$0 \leq x \wedge x < 100 \wedge 0 \leq i \wedge i < 5 \wedge 0 \leq x - i \wedge$$
$$x' = x\text{-}i \wedge$$
$$0 > x' \wedge x' < 100 \wedge 0 \leq i' \wedge i' < 5$$

UNSAT core

New assumption:

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

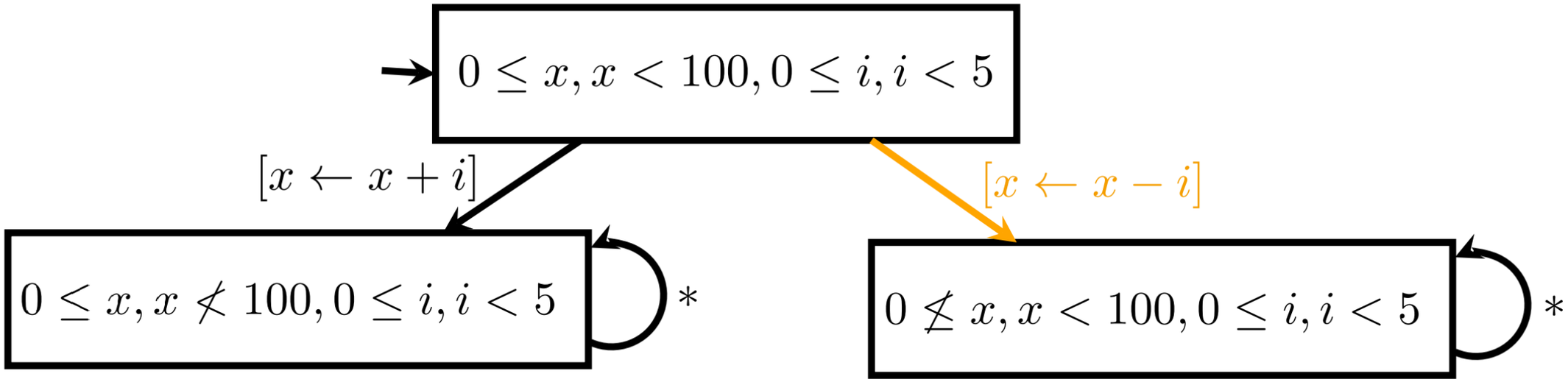
Counter Strategy 4 (Missing Predicates)



Added assumptions:

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

Counter Strategy 4 (Missing Predicates)

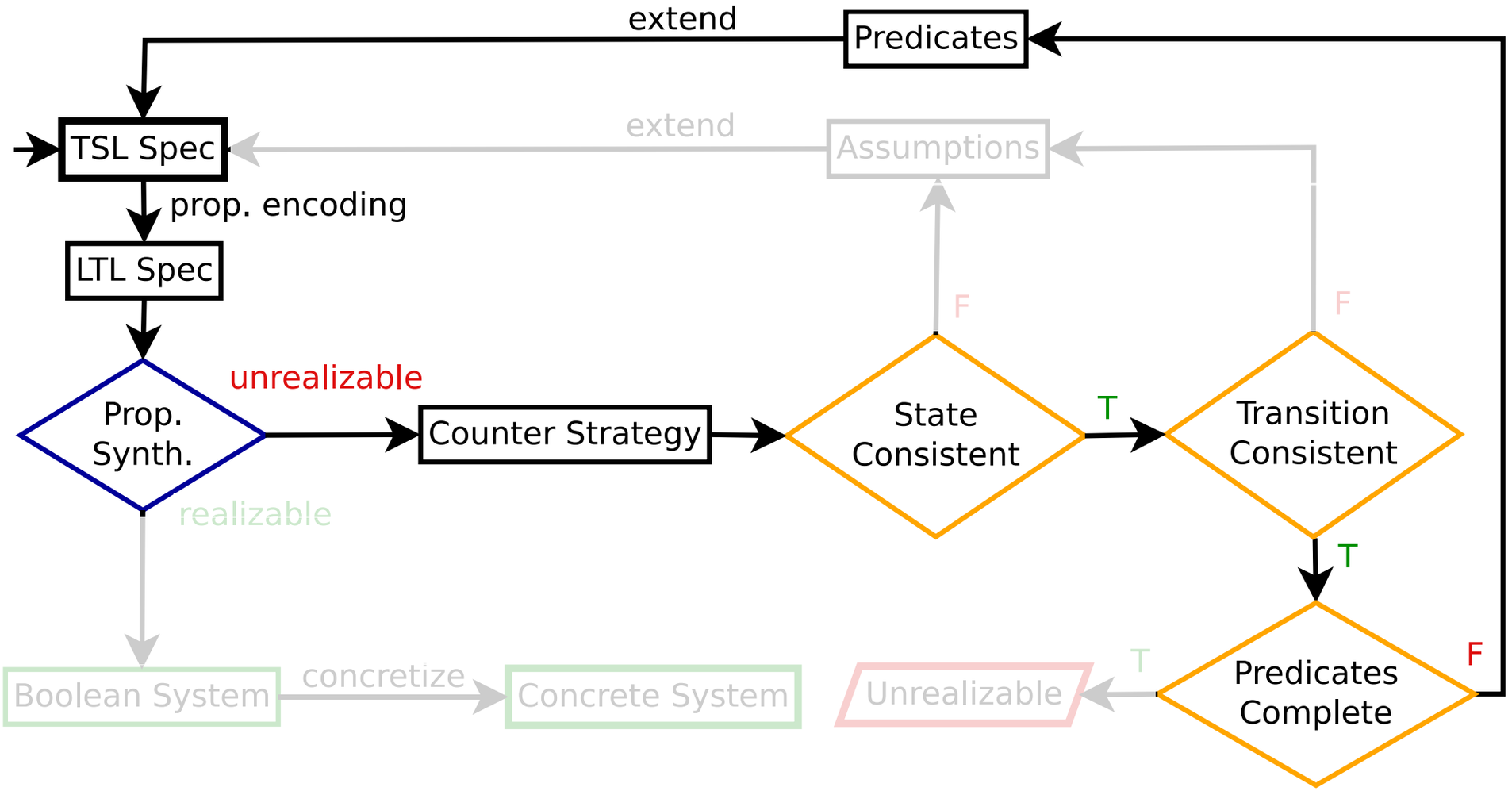


Added assumptions:

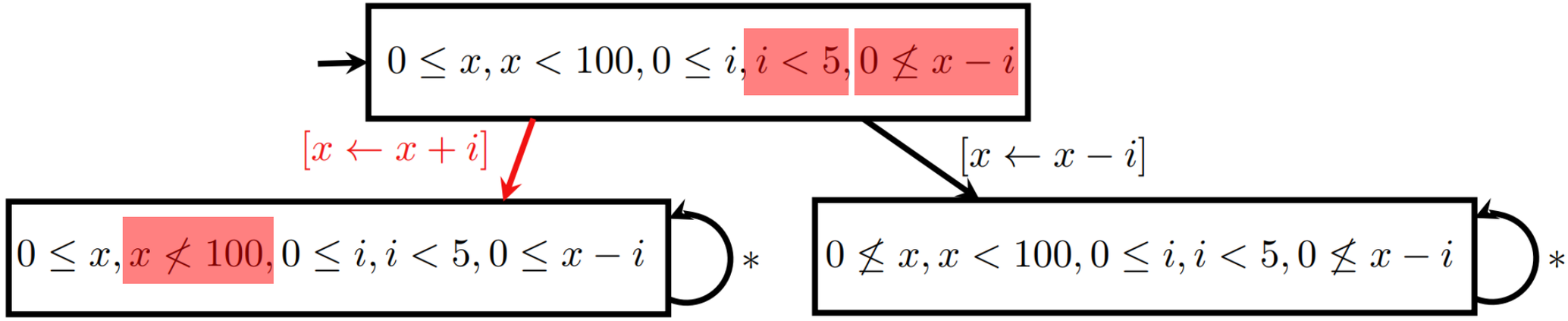
$G((0 \leq x\text{-}i \wedge [x \leftarrow x\text{-}i]) \rightarrow X(x \geq 0)) \wedge$

$G(\neg(0 \leq i \wedge 0 \leq x\text{-}i \wedge 0 > x))$

Synthesis Algorithm



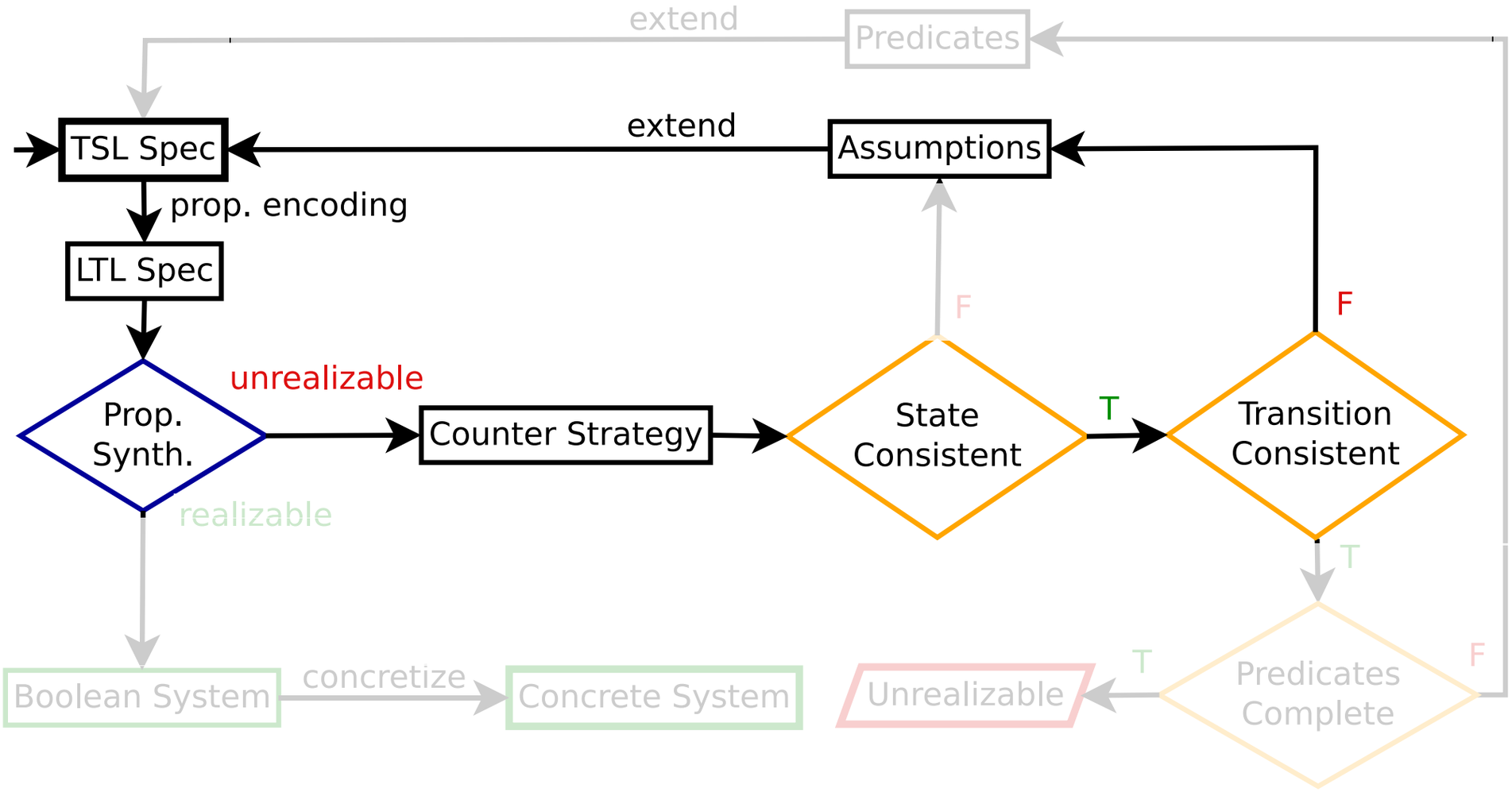
Counter Strategy 5 (Inconsistent Transition)



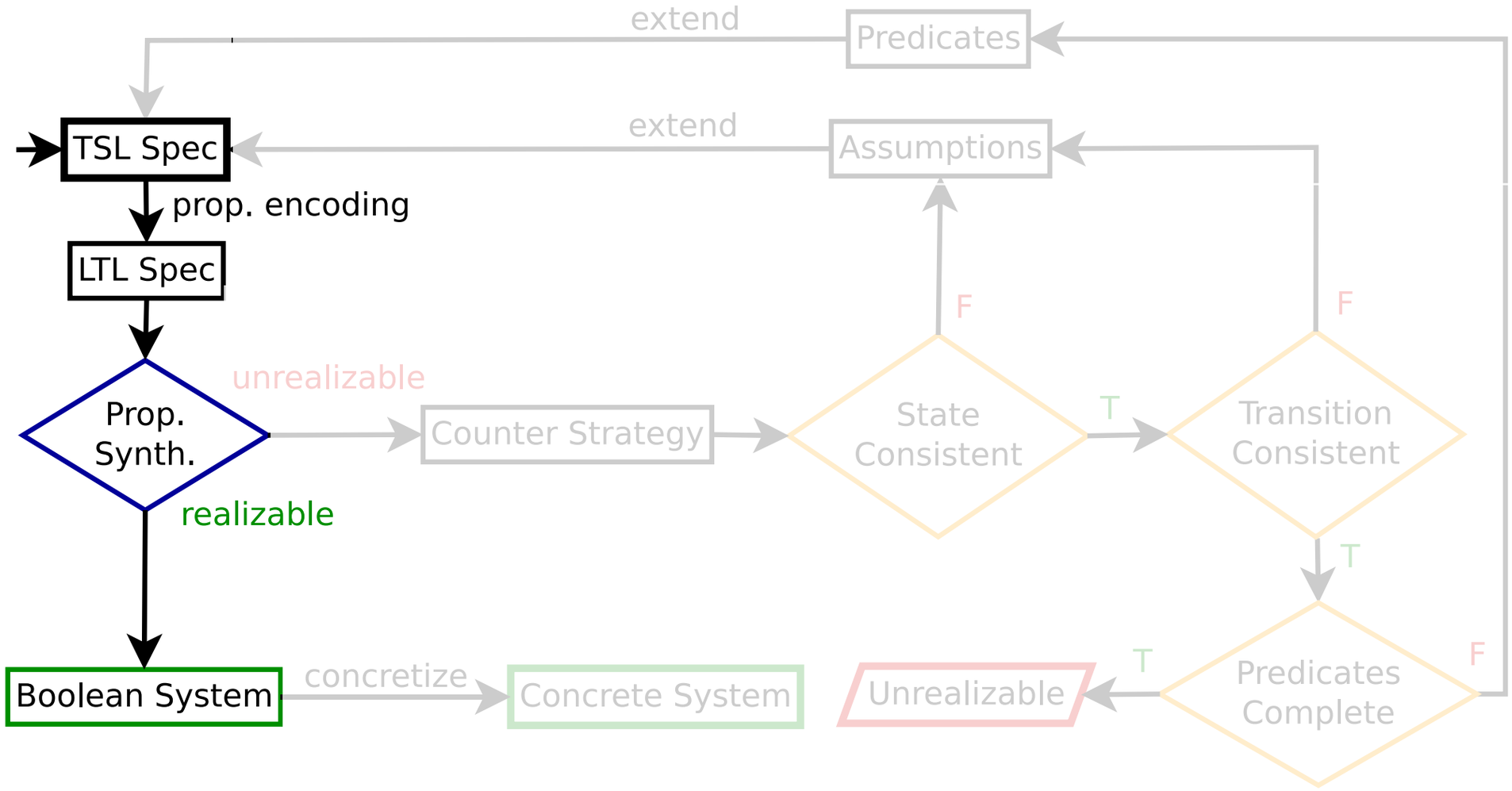
Added assumption:

$G((i > x \wedge i < 5 \wedge [x \leftarrow x+i]) \rightarrow X(x < 100))$

Synthesis Algorithm

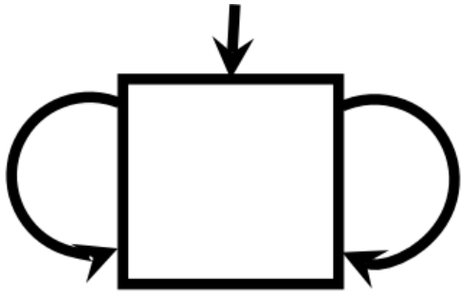


Synthesis Algorithm



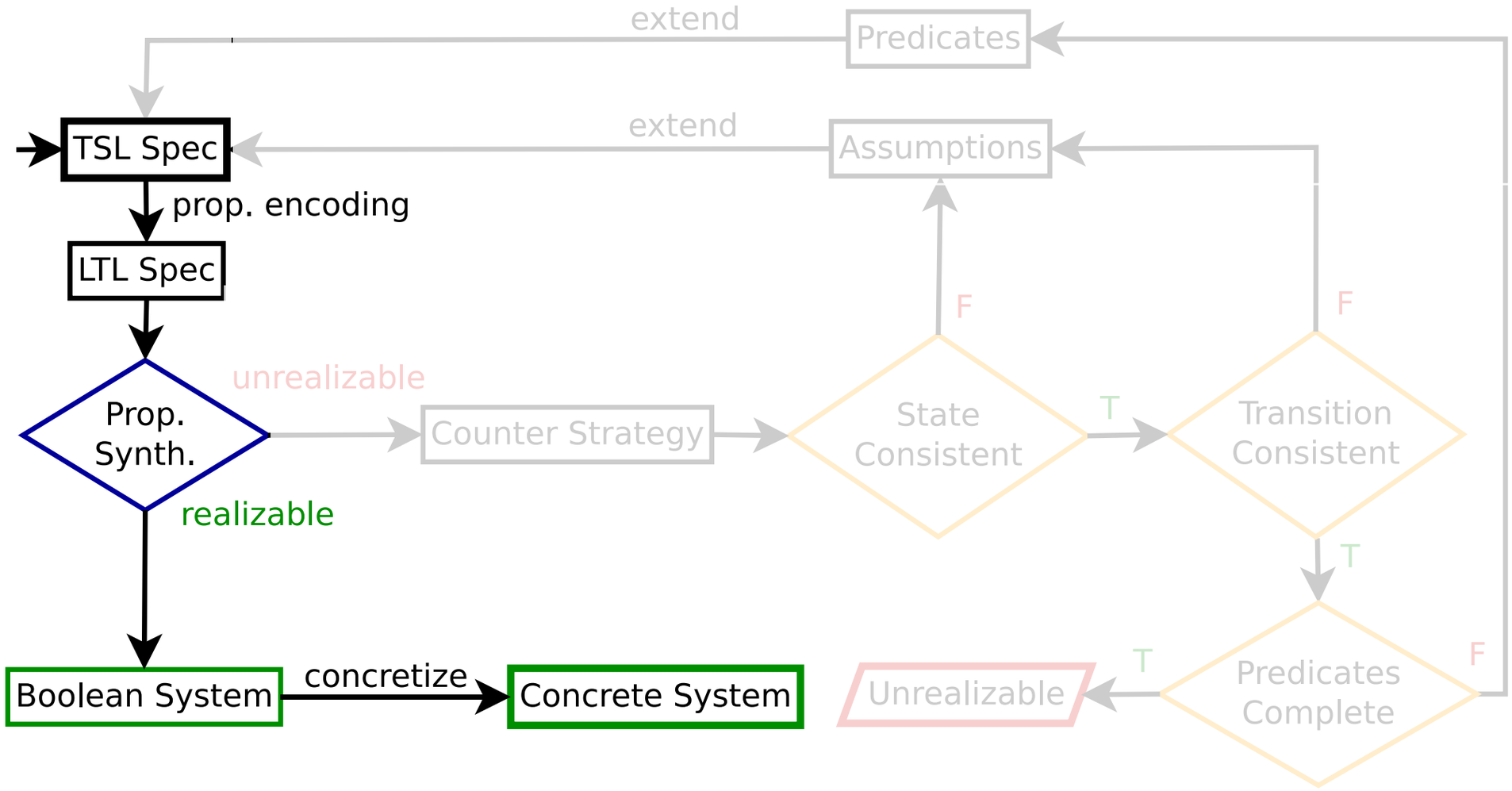
Mealy Machine realizing the Specification

$0 \not\leq x - i, */[x \leftarrow x + i]$



$0 \leq x - i, */[x \leftarrow x - i]$

Synthesis Algorithm



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];
}
```

Advantages

- Expressive specification language
- Symbolic state representation
  state spaces that are intractable for LTL synthesis.
- Decidable for equality logic

Limitations

- Undecidable for theories with arithmetic
- Cannot handle unbounded reachability
  - $(x>0 \land G\,[x \leftarrow x\text{-}1]) \rightarrow F\,x < 0$

Evaluation

c	x_{max}	i_{max}	# refinements	# states	# learned predicates	time [s]
1	100	5	4	1	2	1.0
2	100	5	5	2	2	1.3
2	100 000	50	5	2	2	1.3
3	100	5	9	2	4	2.9
3	100 000	50	10	2	4	3.1
1	100	110	6	unrealizable	3	1.0
2	100	60	4	unrealizable	2	0.8

Table 1. Results for the extended running example.

Evaluation

Name	# refinements	# new predicates	# states	time [s]
grid world: box	5	4	4	4.35
grid world: track	6	0	1	91.04
grid world: evasion	5	0	3	13.78
grid world: falling rocks	4	10	5	11.5
elevator: 5 floors	15	0	4	8.2
elevator: 8 floors	21	0	4	45
elevator signal 3 floors	5	0	1	35
tanks safety	4	13	1	31
tank liveness	18	9	2	95

Summary

- Synthesis procedure for temporal stream logic with theories.
- Based on abstraction refinement and SMT solving.
- Works with all theories that support quantifier elimination
- Finds new required predicates

Evaluation (Elevators)

version	# floors	# refinements	# states	time [s]
simple	3	13	2	3.1
simple	4	11	3	3.7
simple	5	15	4	8.2
simple	8	21	4	45
simple	10	24	4	185
input signal	3	5	1	35
input signal	4	5	1	217
input signal	5	6	1	1424

Table 2. Results for the elevator.

Evaluation

TABLE III
INFINITE GRID WORLD BENCHMARKS FROM [13]. ALL TIMES IN SECONDS — DENOTES A TIME-OUT AFTER 900S. TOOL ABBREVIATIONS: C (CONSYNTH), J (JSYN-VG), D (DT-SYNTH), S (SAT-SYNTH), R (RPNI-SYNTH), G (GENSYS)

Benchmark	C	J	D	S	R	G	Raboniel
Box	3.7	0.6	0.3	0.3	0.1	0.3	1.9
Box Limited	0.4	1.7	0.1	0.4	0.5	0.2	0.6
Diagonal	1.9	4.0	2.4	1.3	0.5	0.2	10.9
Evasion	1.5	0.5	0.2	81	0.1	0.7	5.4
Follow	—	1.2	0.3	88.9	—	0.7	—
Solitary Box	0.4	0.9	0.1	0.3	0.1	0.3	0.5
Square 5x5	—	6.5	2.5	0.6	0.2	0.3	75.1

Local Analysis