UPPSALA
UNIVERSITET

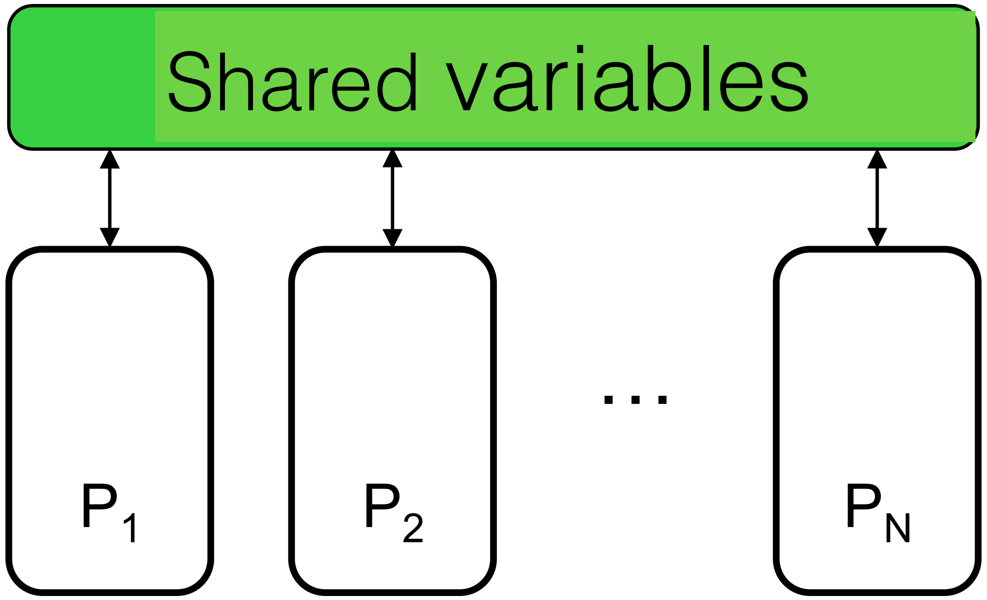
Counter-Example Guided Program Verification

Parosh Aziz Abdulla, Mohamed Faouzi Atig, and **Bui Phi Diep**

Uppsala University, Sweden

Concurrent Programs

1. Parallel processes with shared variables



2. Interleaving (Sequentially Consistent) semantics:

- Computations of different processes are shuffled
- Program order is preserved for each process

Verification of Concurrent Programs

For a program ***P***, and a (control + variable values) state ***s*** :

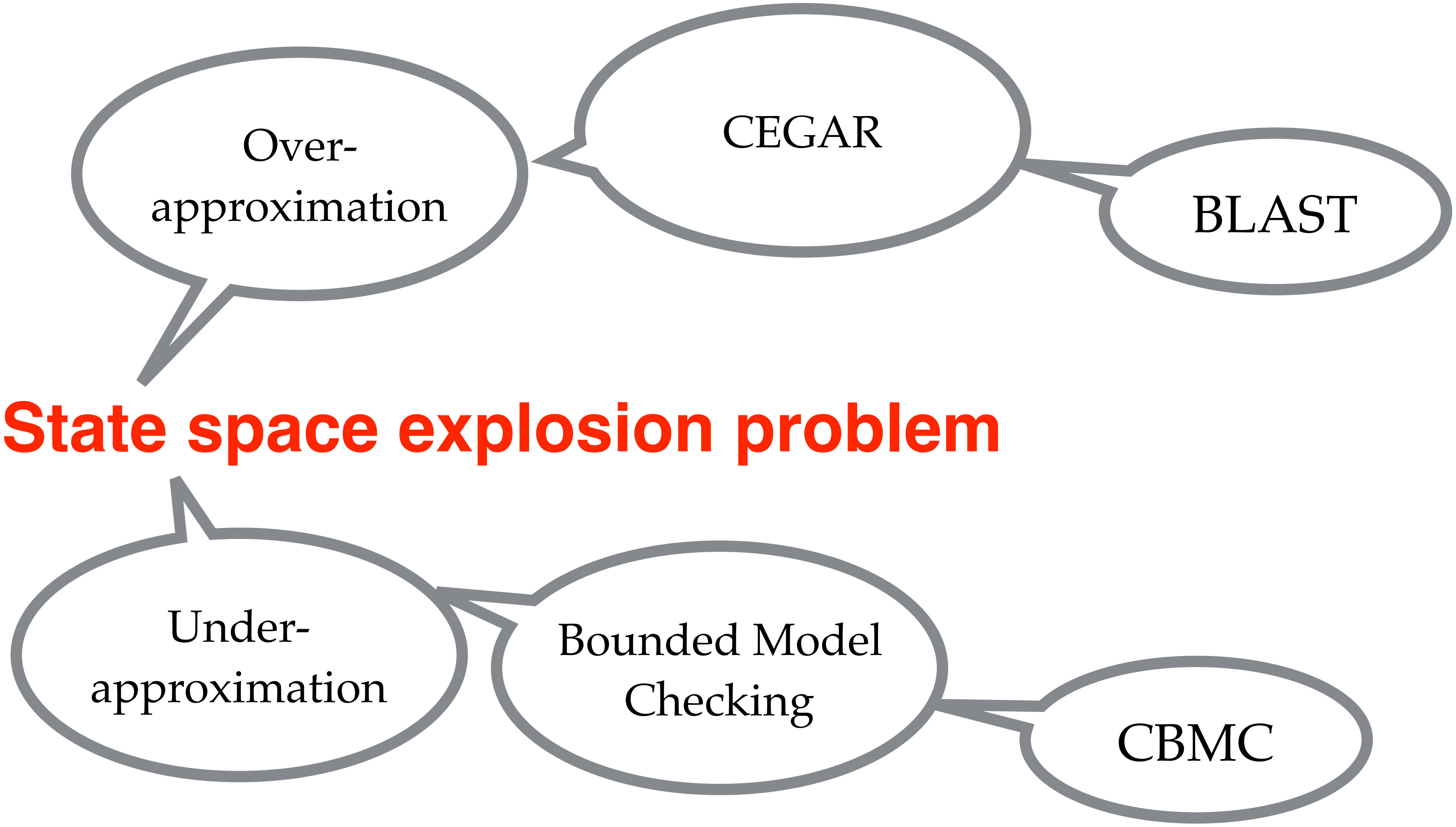
State Reachability Problem (Safety)

s *is reachable in* ***P*** *?*

State space explosion problem

Verification of Concurrent Programs

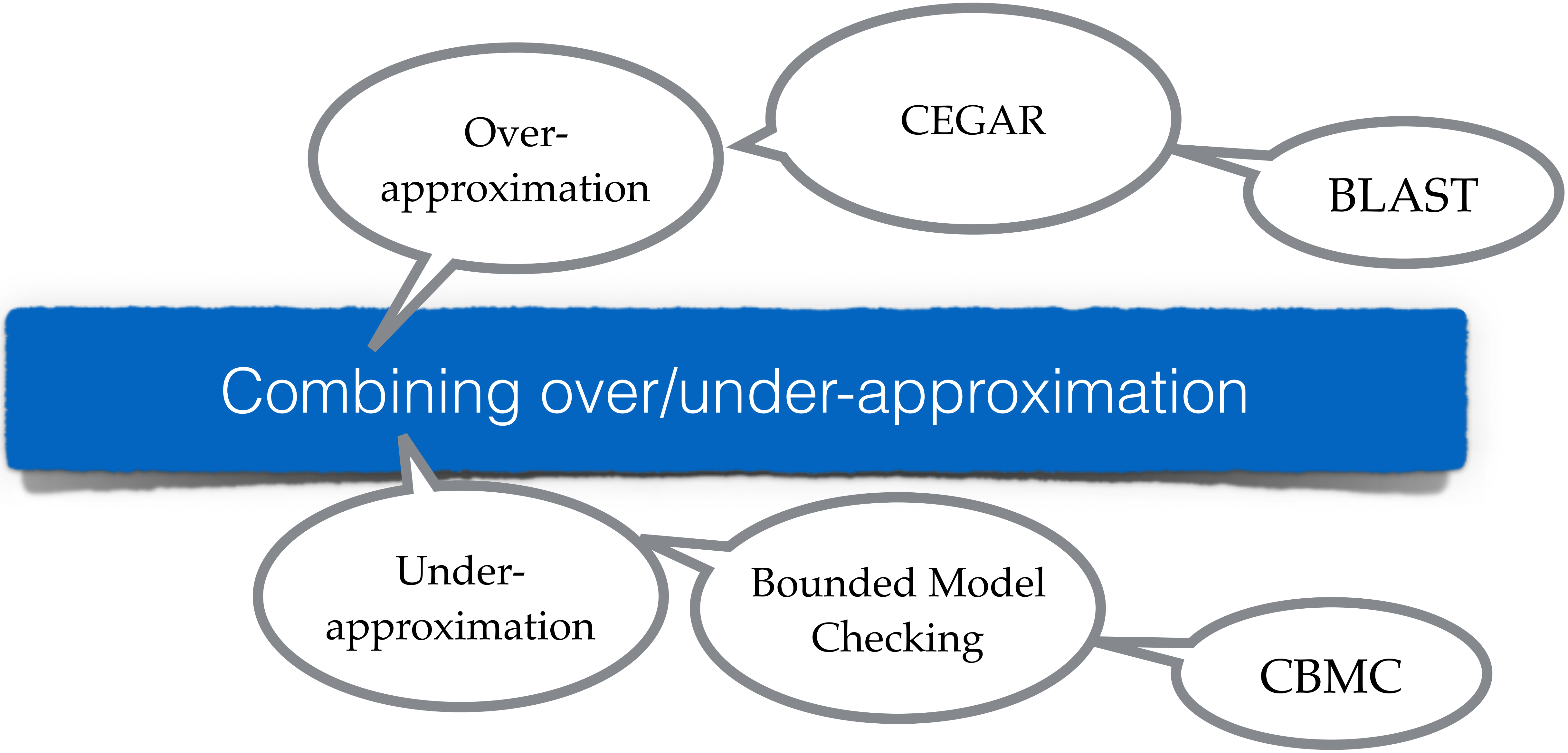
State Reachability Problem (Safety)



State space explosion problem

Verification of Concurrent Programs

State Reachability Problem (Safety)



Over-approximation

CEGAR

BLAST

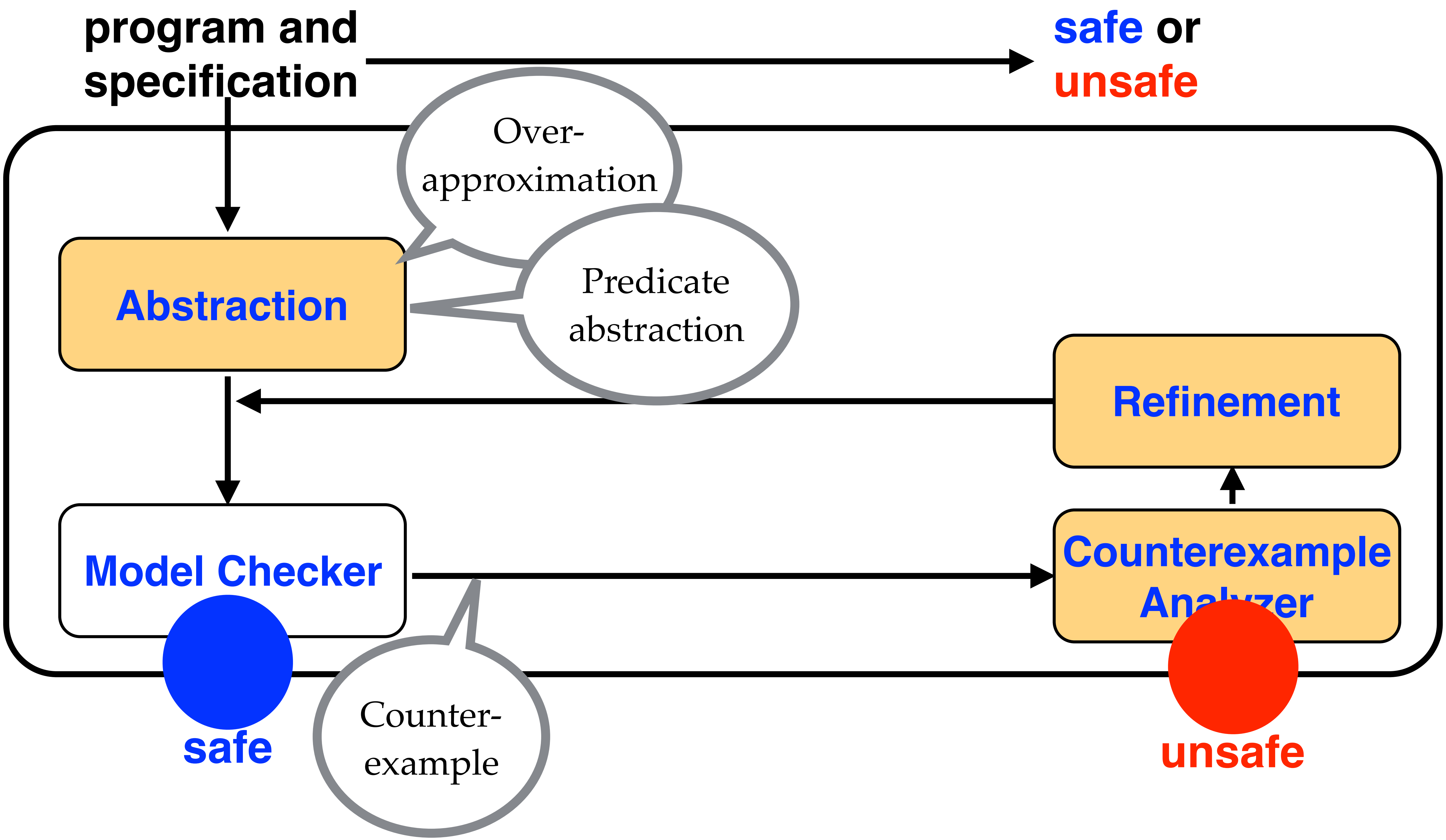
Combining over/under-approximation

Under-approximation

Bounded Model Checking

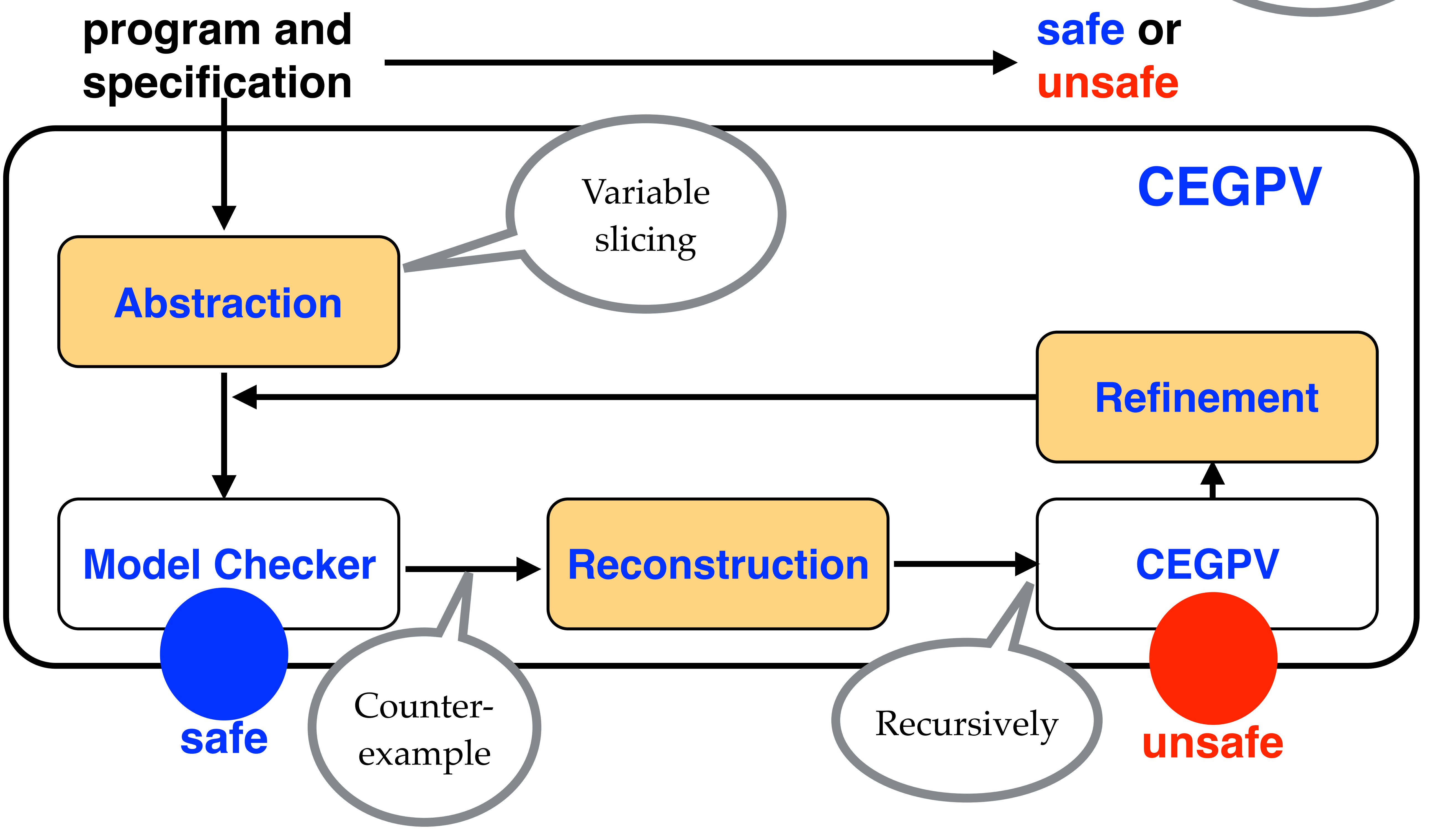
CBMC

CEGAR


program and specification
safe or unsafe
Over-approximation
Abstraction
Predicate abstraction
Refinement
Model Checker
Counterexample Analyzer
Counter-example
safe
unsafe

Our contribution

CEGAR extension



program and specification

safe or unsafe

CEGPV

Abstraction

Variable slicing

Refinement

Model Checker

Reconstruction

CEGPV

safe

Counter-example

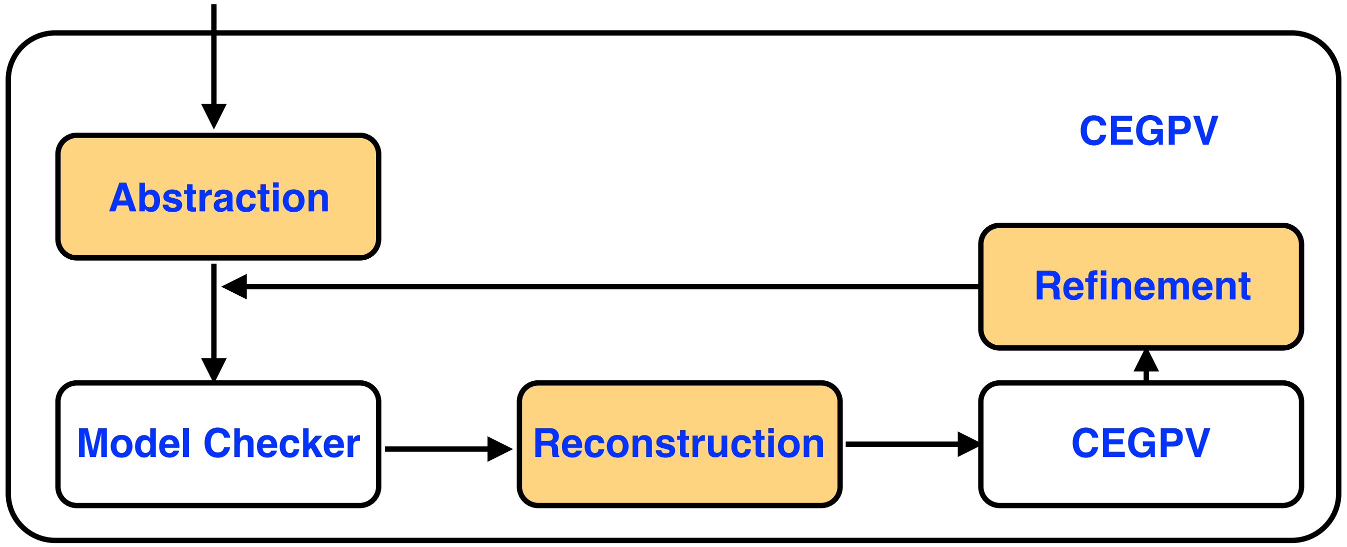
Recursively

unsafe

Our contribution

1. Deal with the state-space explosion problem
2. CEGAR extension for program verification
3. Code to code translation

Can run on any back-end tools



Example

SV-COMP

var: x, y, z, t1, t2

```
P1:  x = y?z?0:1:1
P2:  y = z
P3:  z = 0
P4:  t1 = x
P5:  assert(t1 + t2 != 1)
```

```
Q1:  x = y?0:z?0:1
Q2:  y = !z
Q3:  z = 1
Q4:  t2 = x
```

```
if (y)
   x = 0
else if (z)
   x = 0
else
   x = 1
```

Safety property

Variable dependency

var: x, y, z, t1, t2

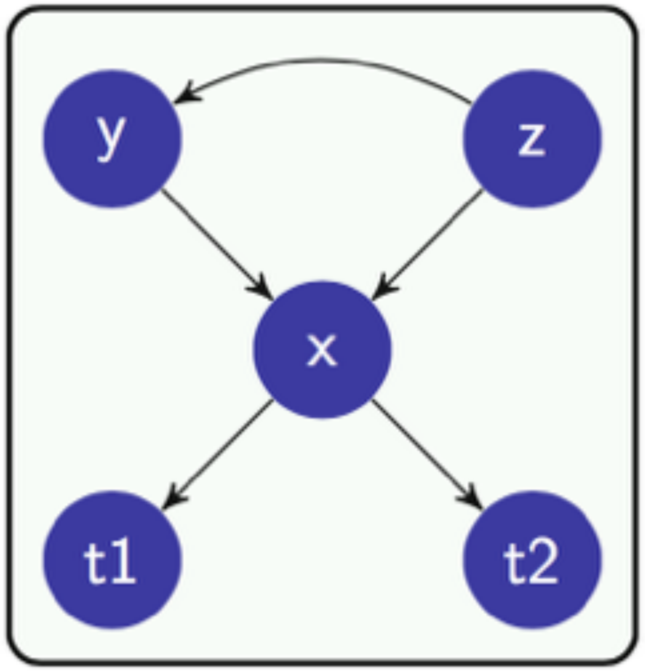
```
P1:  x = y?z?0:1:1
P2:  y = z
P3:  z = 0
P4:  t1 = x
P5:  assert(t1 + t2 != 1)
```

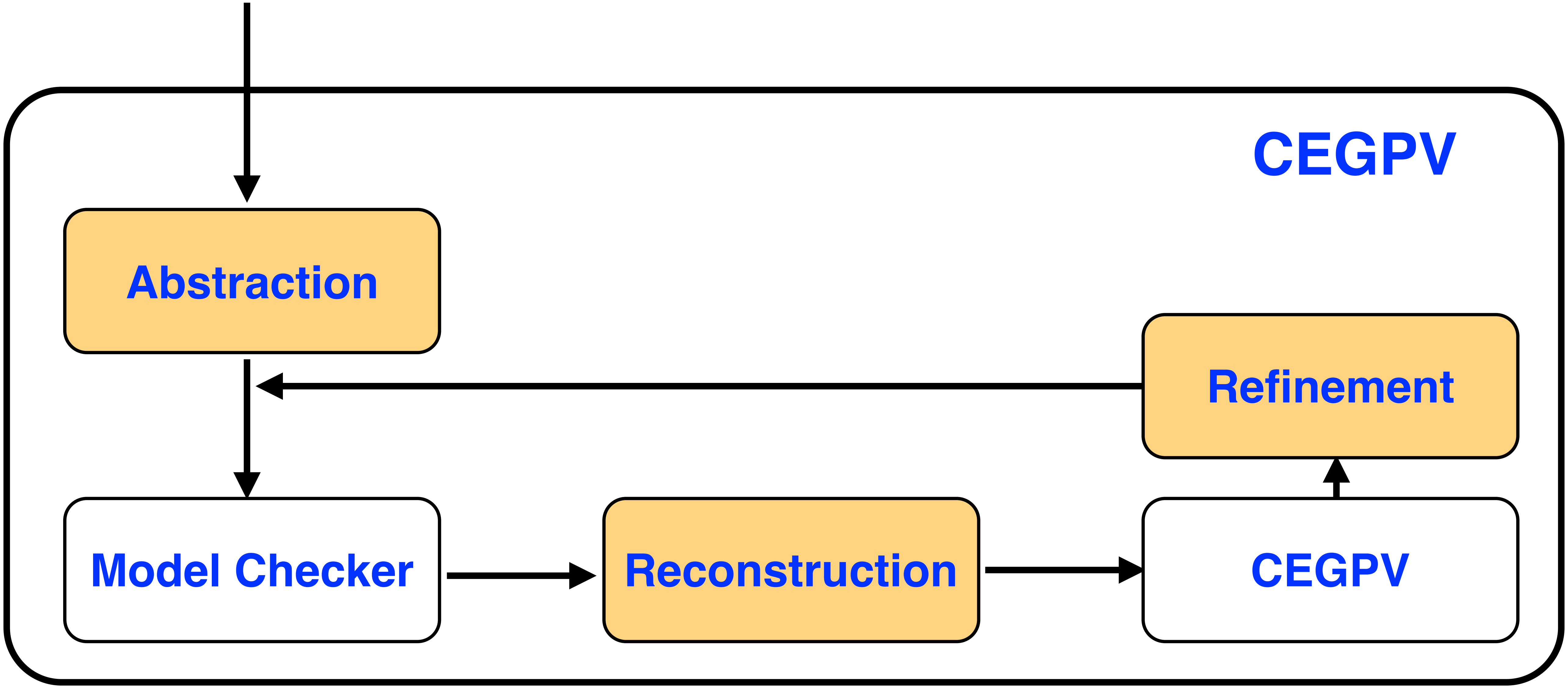
```
Q1:  x = y?0:z?0:1
Q2:  y = !z
Q3:  z = 1
Q4:  t2 = x
```

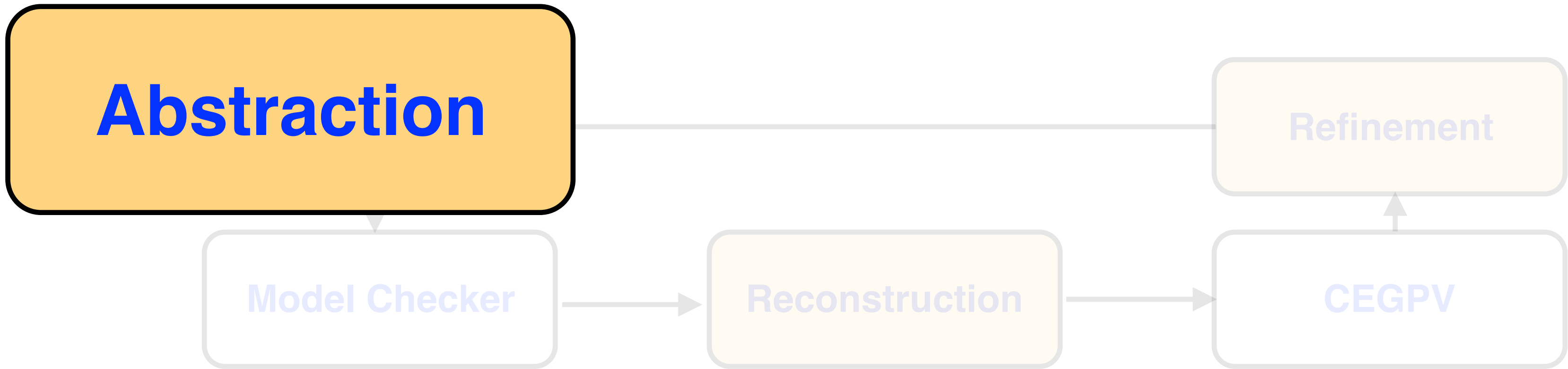
$y \rightarrow x$

$z \rightarrow x$

Variable dependency graph




CEGPV
Abstraction
Refinement
Model Checker
Reconstruction
CEGPV



Abstraction

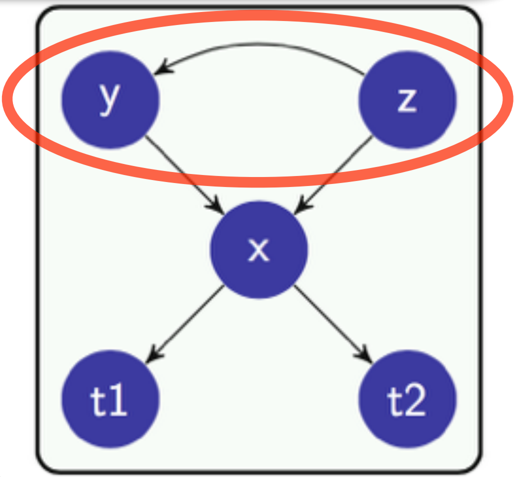
Refinement

Model Checker

Reconstruction

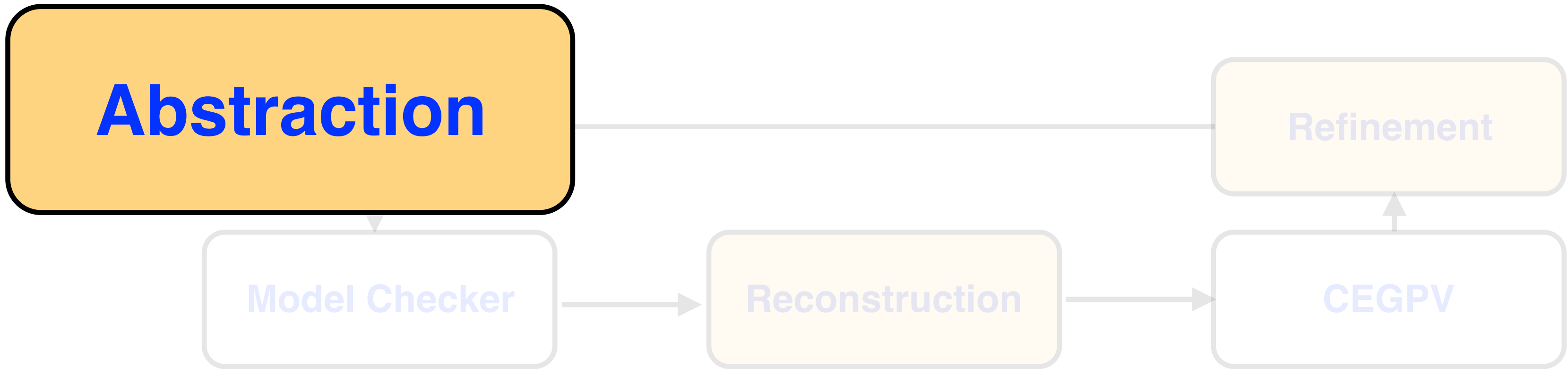
CEGPV

1. Replace some variables by a non-deterministic value (*)
2. Remove assignments of removed variables



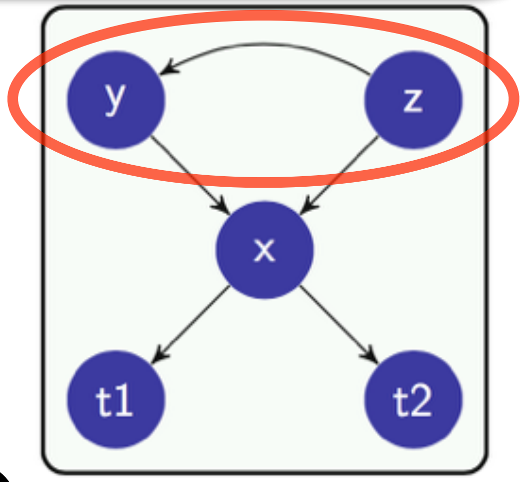
```
var: x, y, z, t1, t2

P1:  x = y?z?0:1:1
P2:  y = z
P3:  z = 0
P4:  t1 = x
P5:  assert(t1 + t2 != 1)
```

```
Q1:  x = y?0:z?0:1
Q2:  y = !z
Q3:  z = 1
Q4:  t2 = x
```



1. Replace some variables by a non-deterministic value (*)

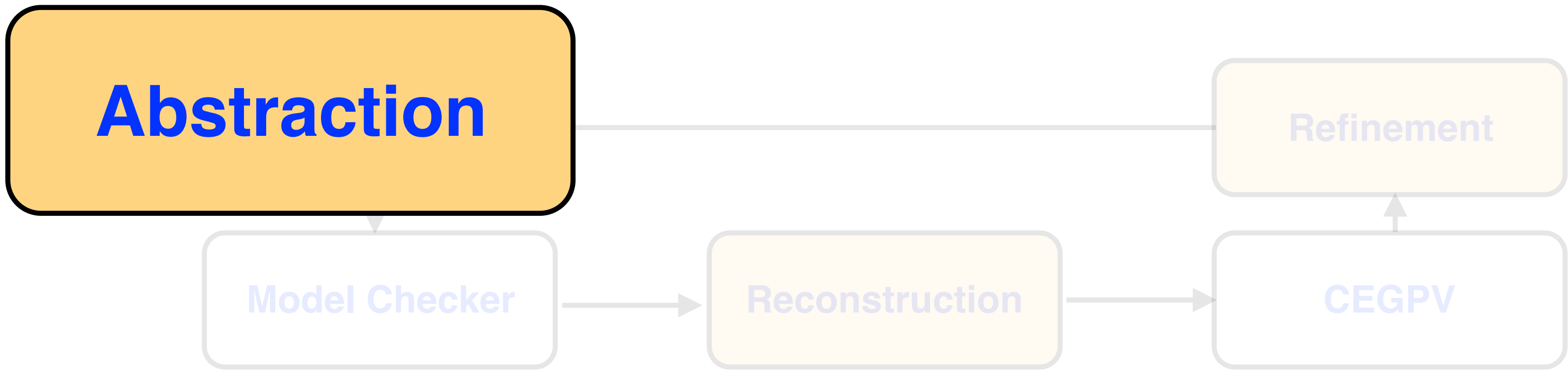
2. Remove assignments of removed variables



removing variables

contain removing variables

```
var: x,        t1, t2

P1:  x = y?z?0:1:1           Q1:  x = y?0:z?0:1
P2:  y = z                   Q2:  y = !z
P3:  z = 0                   Q3:  z = 1
P4:  t1 = x                  Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```



Abstraction

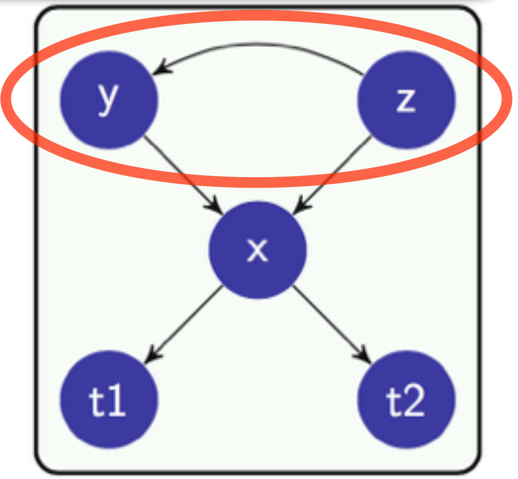
Refinement

Model Checker

Reconstruction

CEGPV

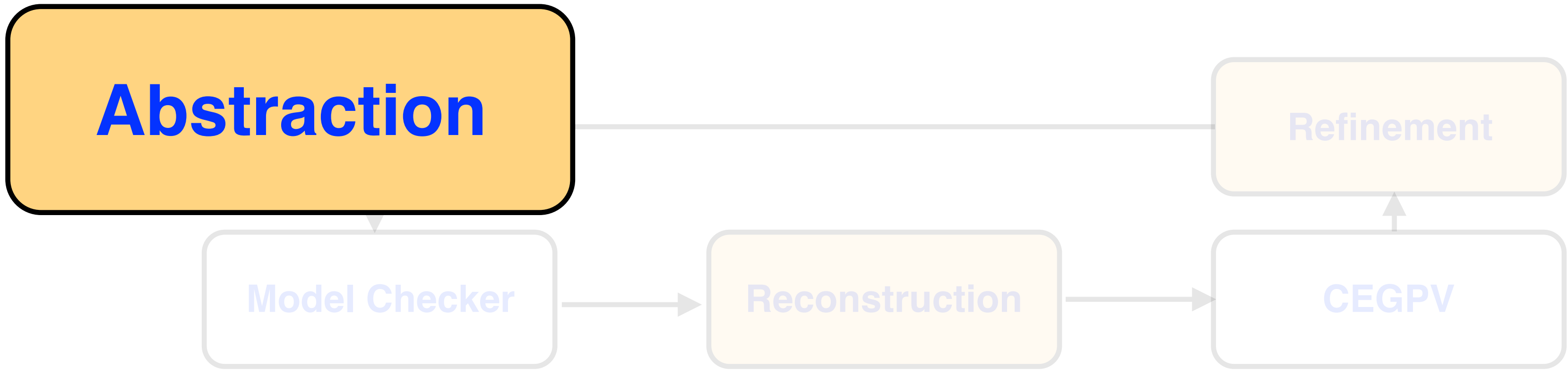
1. Replace some variables by a non-deterministic value (*)
2. Remove assignments of removed variables



removing variables

contain removing variables

```
var: x,      t1, t2

P1:  x = y?z?0:1:1           Q1:  x = y?0:z?0:1
P2:  y = z                   Q2:  y = !z
P3:  z = 0                   Q3:  z = 1
P4:  t1 = x                  Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```



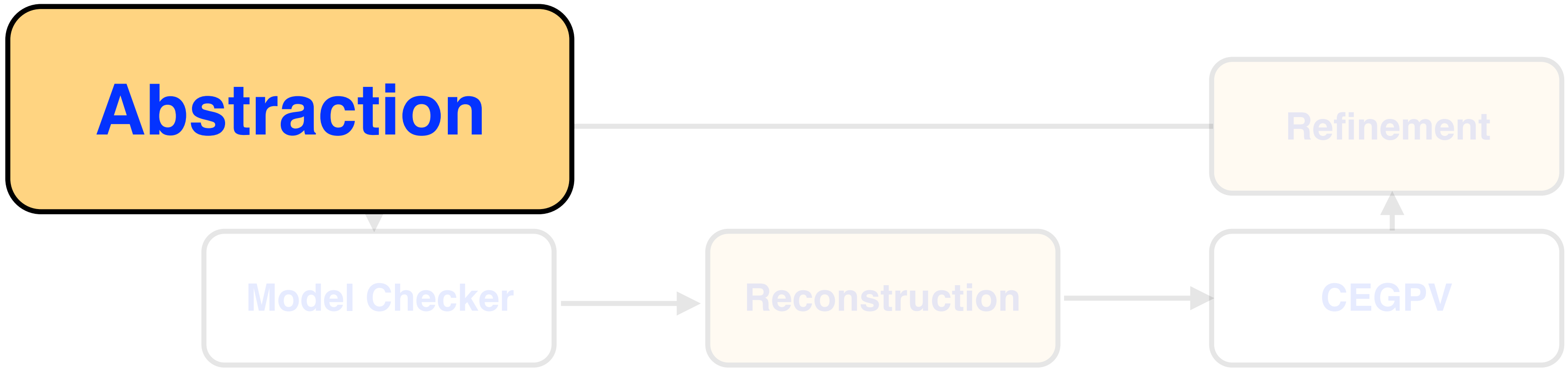
1. Replace some variables by a non-deterministic value (*)

2. Remove assignments of removed variables

removing variables

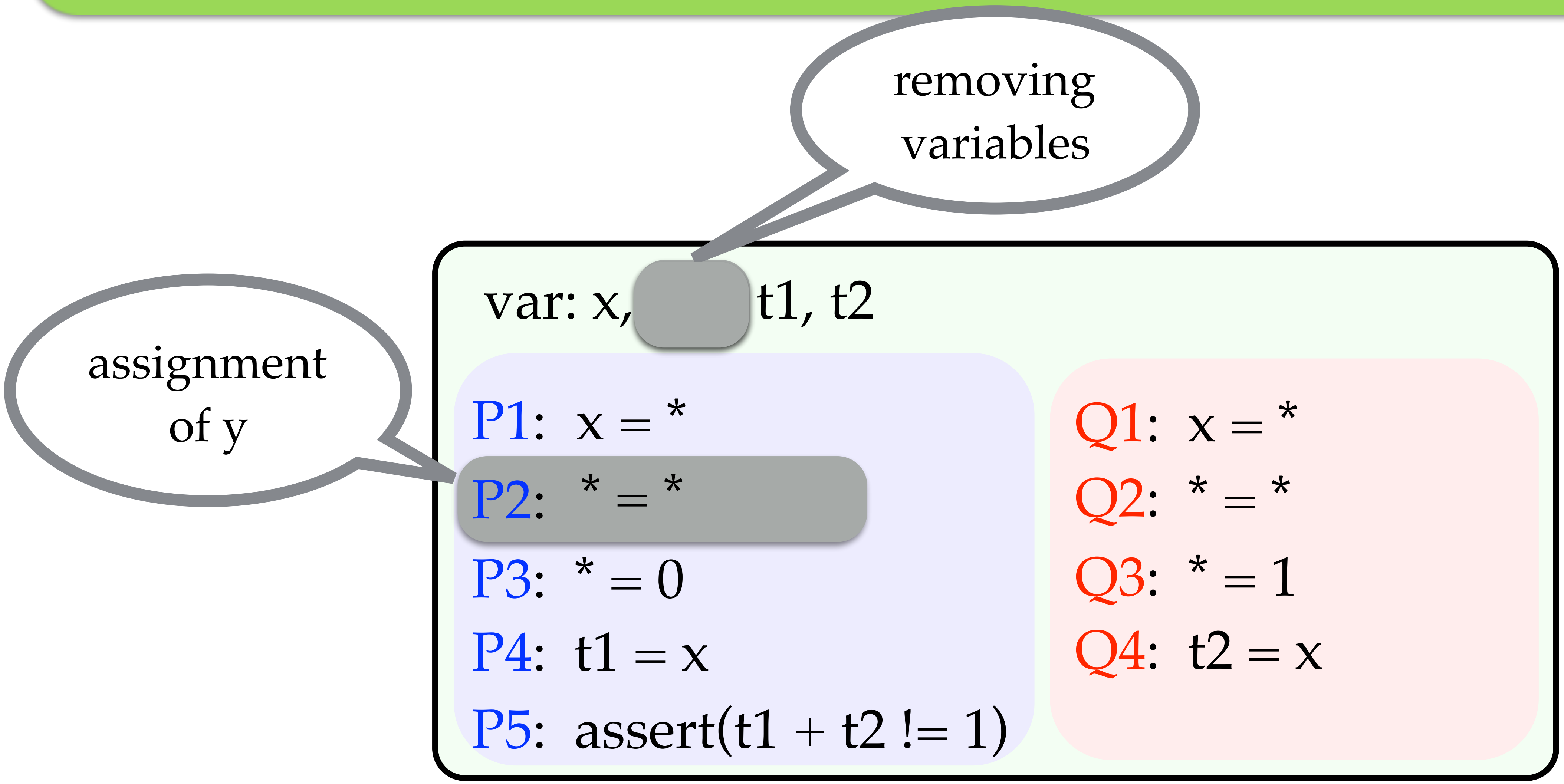
var: x, t1, t2

P1: x = * ✓	Q1: x = * ✓
P2: * = * ✓	Q2: * = * ✓
P3: * = 0 ✓	Q3: * = 1 ✓
P4: t1 = x	Q4: t2 = x
P5: assert(t1 + t2 != 1)	



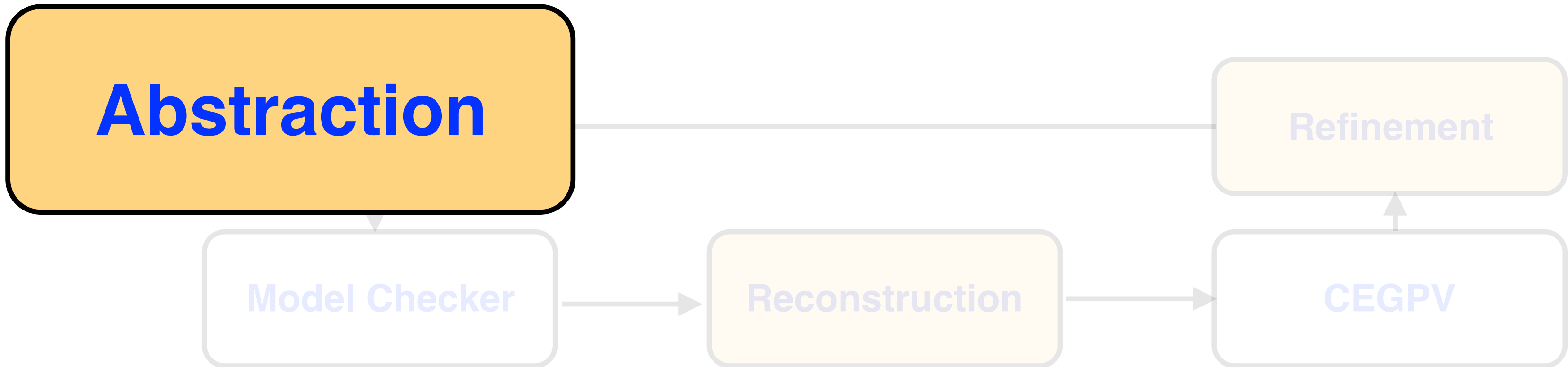
1. Replace some variables by a non-deterministic value (*)

2. Remove assignments of removed variables


removing variables

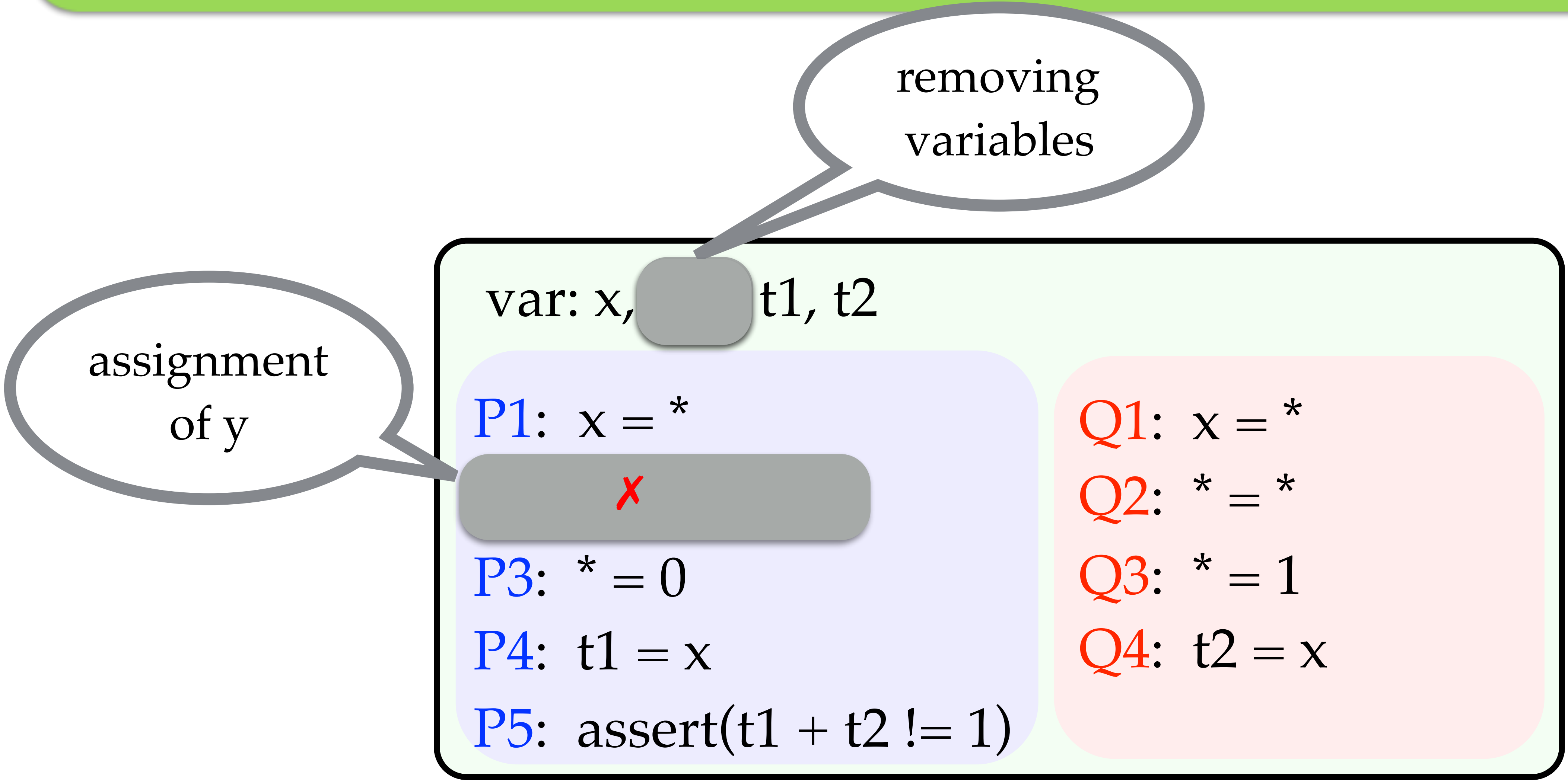
assignment of y

```
var: x,      t1, t2

P1:  x = *
P2:  * = *
P3:  * = 0
P4:  t1 = x
P5:  assert(t1 + t2 != 1)

Q1:  x = *
Q2:  * = *
Q3:  * = 1
Q4:  t2 = x
```

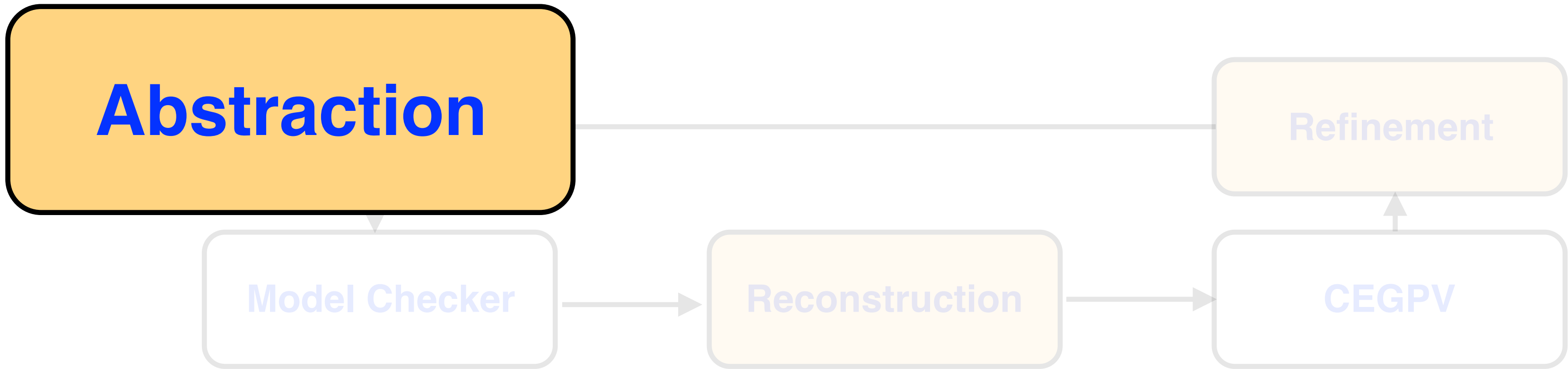


1. Replace some variables by a non-deterministic value (*)

2. Remove assignments of removed variables



```
var: x, t1, t2

P1: x = *
P3: * = 0
P4: t1 = x
P5: assert(t1 + t2 != 1)

Q1: x = *
Q2: * = *
Q3: * = 1
Q4: t2 = x
```

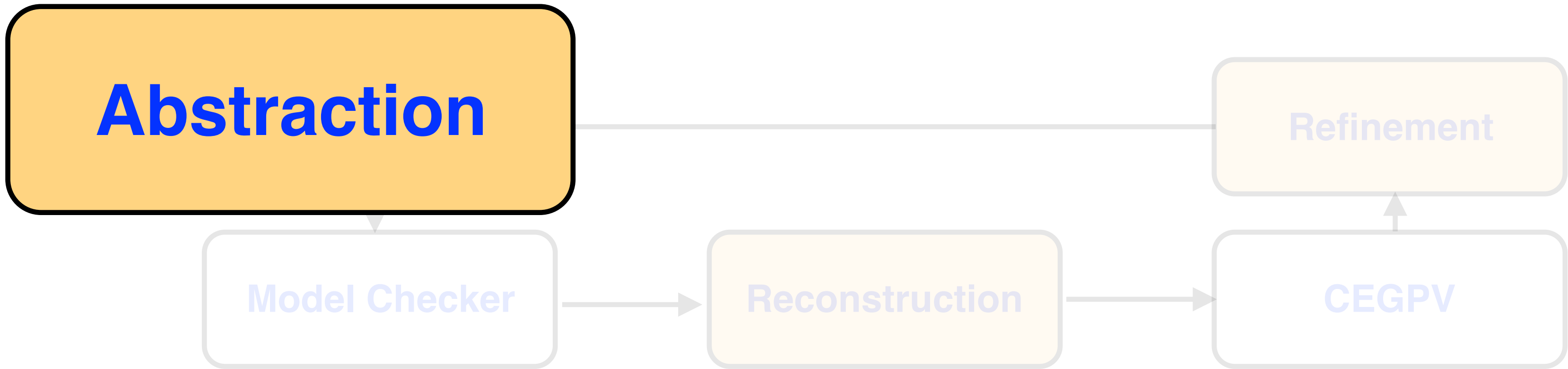


1. Replace some variables by a non-deterministic value (*)

2. Remove assignments of removed variables

removing variables

```
var: x,      t1, t2

P1:  x = *
       ✗
       ✗
P4:  t1 = x
P5:  assert(t1 + t2 != 1)

Q1:  x = *
       ✗
       ✗
Q4:  t2 = x
```

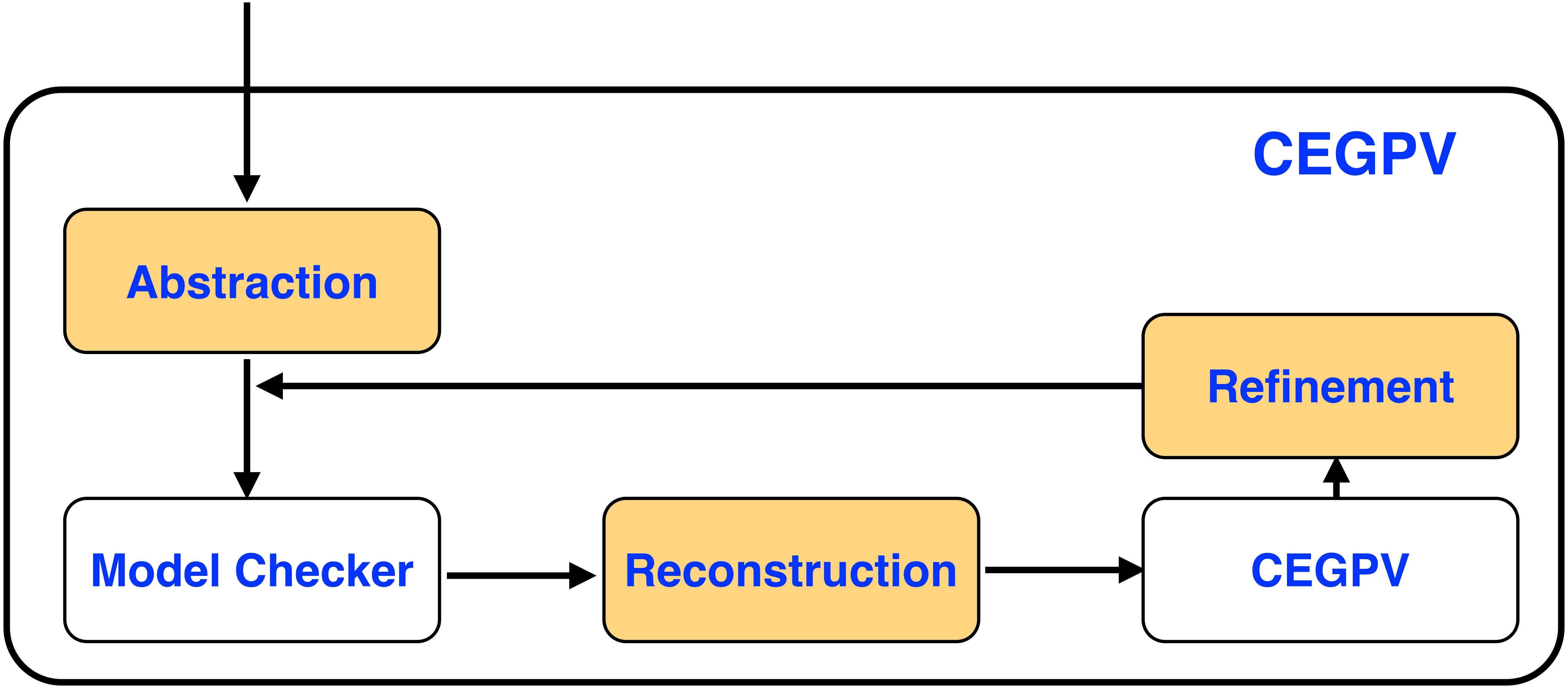


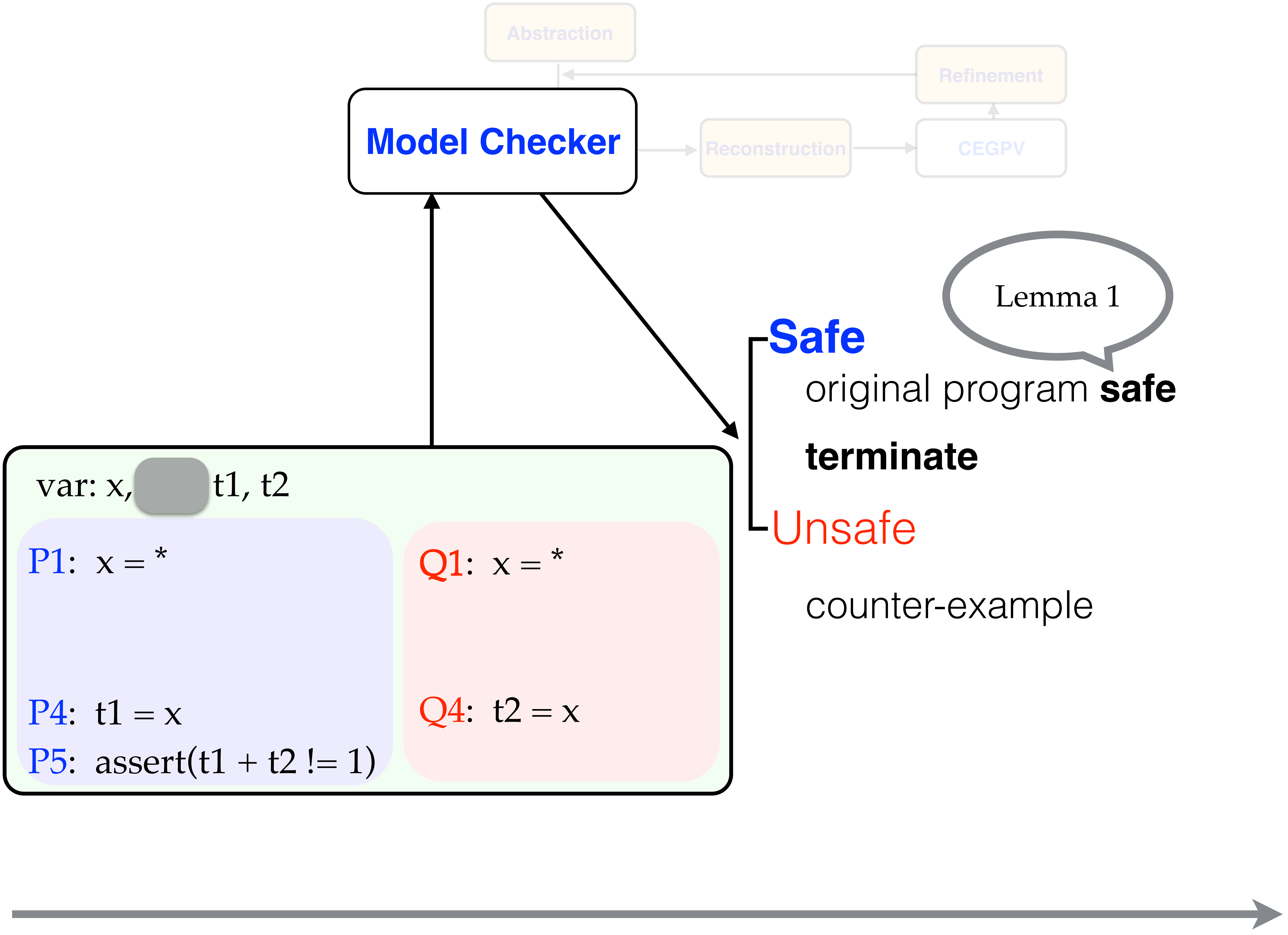
Lemma 1: if the abstracted program is **safe**, then the original program is **safe**.

```
var: x, y, z, t1, t2
P1:  x = y?z?0:1:1        Q1:  x = y?0:z?0:1
P2:  y = z                Q2:  y = !z
P3:  z = 0                Q3:  z = 1
P4:  t1 = x               Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

```
var: x,      t1, t2
P1:  x = *                Q1:  x = *



P4:  t1 = x               Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

Abstracted program is an **over-approximation** of original program


CEGPV
Abstraction
Refinement
Model Checker
Reconstruction
CEGPV



Safe

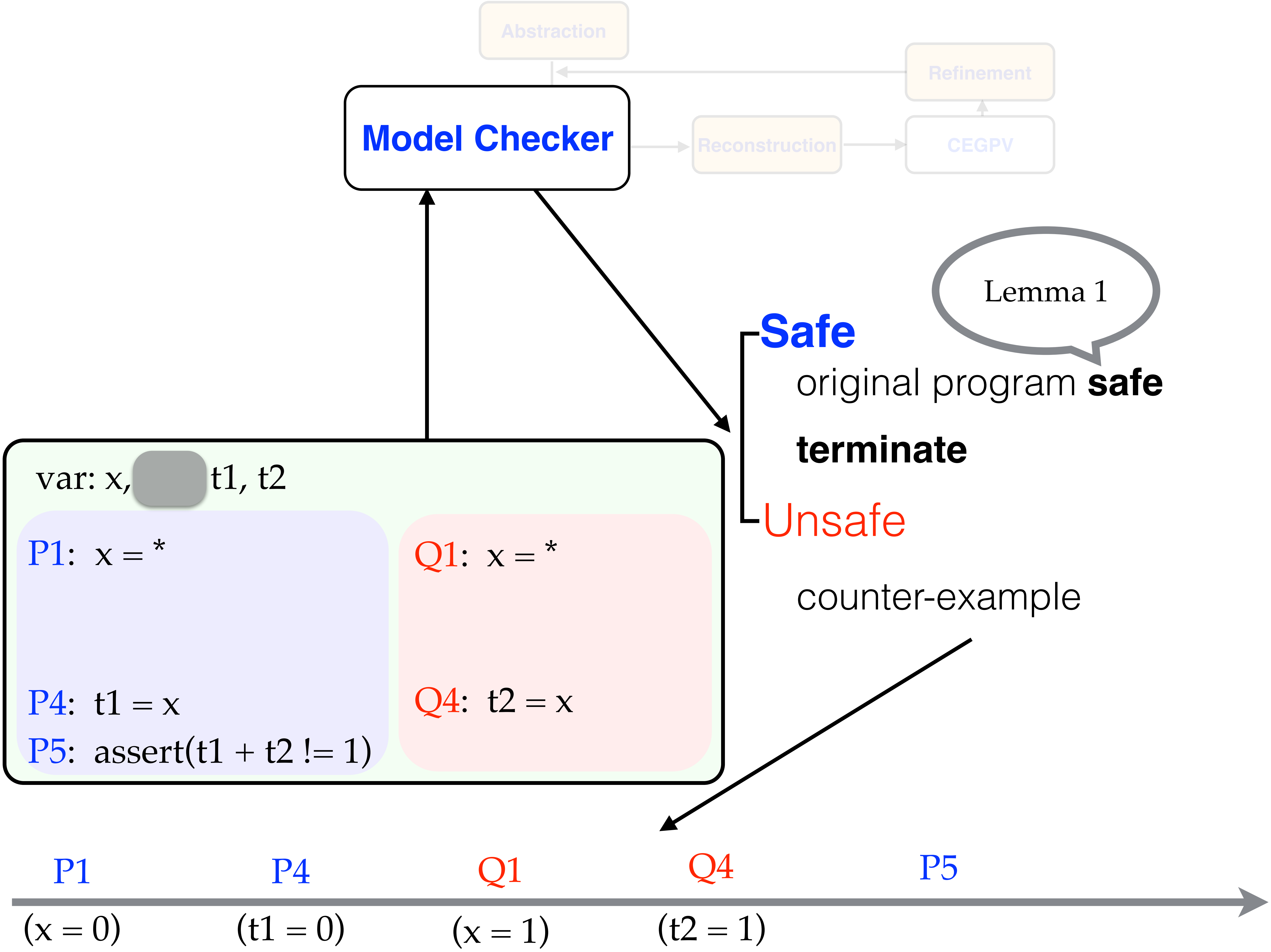
original program **safe**

terminate

Unsafe

counter-example

```
var: x,     t1, t2

P1:  x = *              Q1:  x = *

P4:  t1 = x             Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```



Safe

original program **safe**

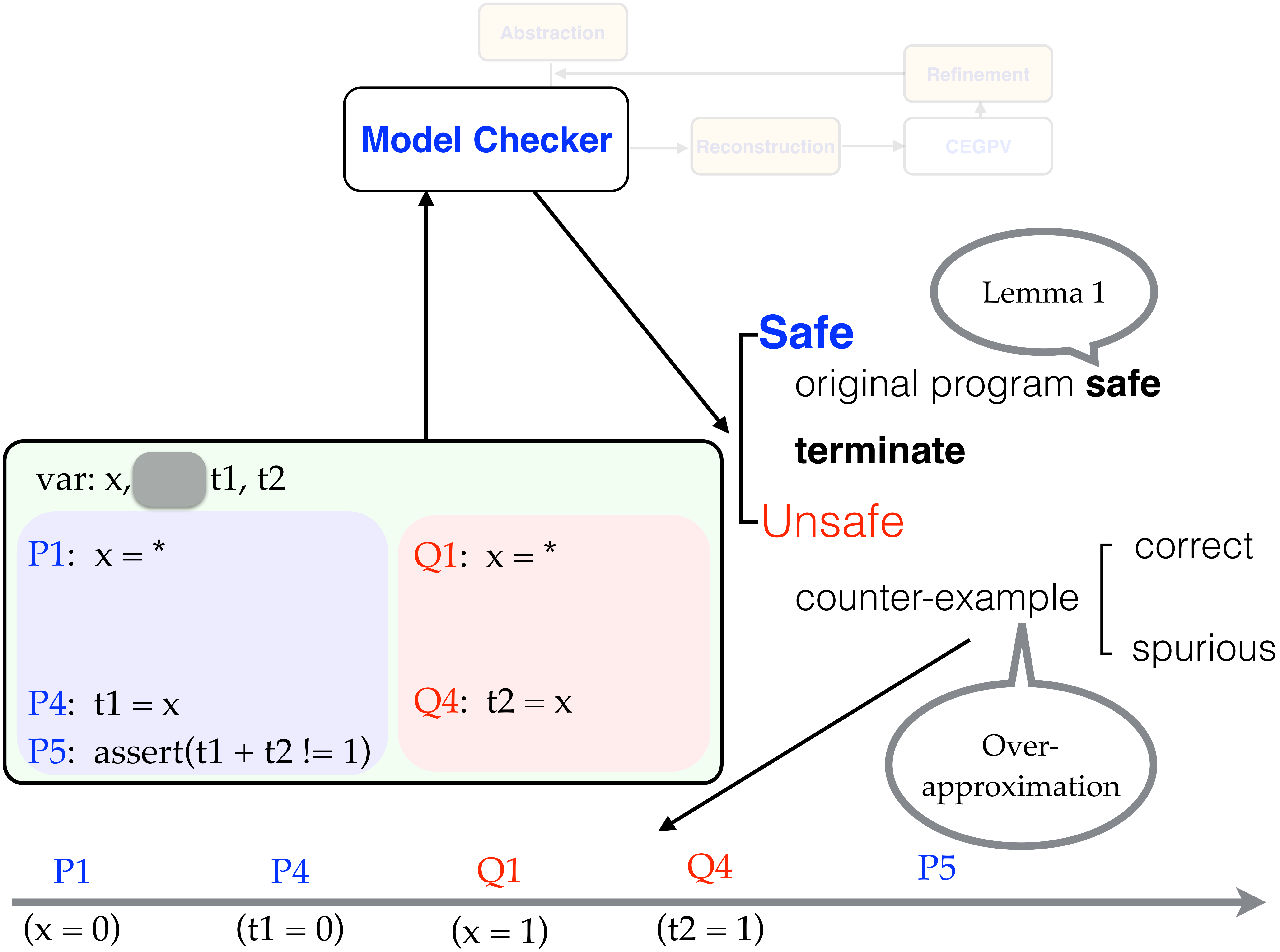
terminate

Unsafe

counter-example

```
var: x,      t1, t2

P1:  x = *                    Q1:  x = *

P4:  t1 = x                   Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

P1	P4	Q1	Q4	P5
(x = 0)	(t1 = 0)	(x = 1)	(t2 = 1)	



Abstraction

Refinement

Model Checker

Reconstruction

CEGPV

Lemma 1

Safe

original program **safe**

terminate

Unsafe

counter-example

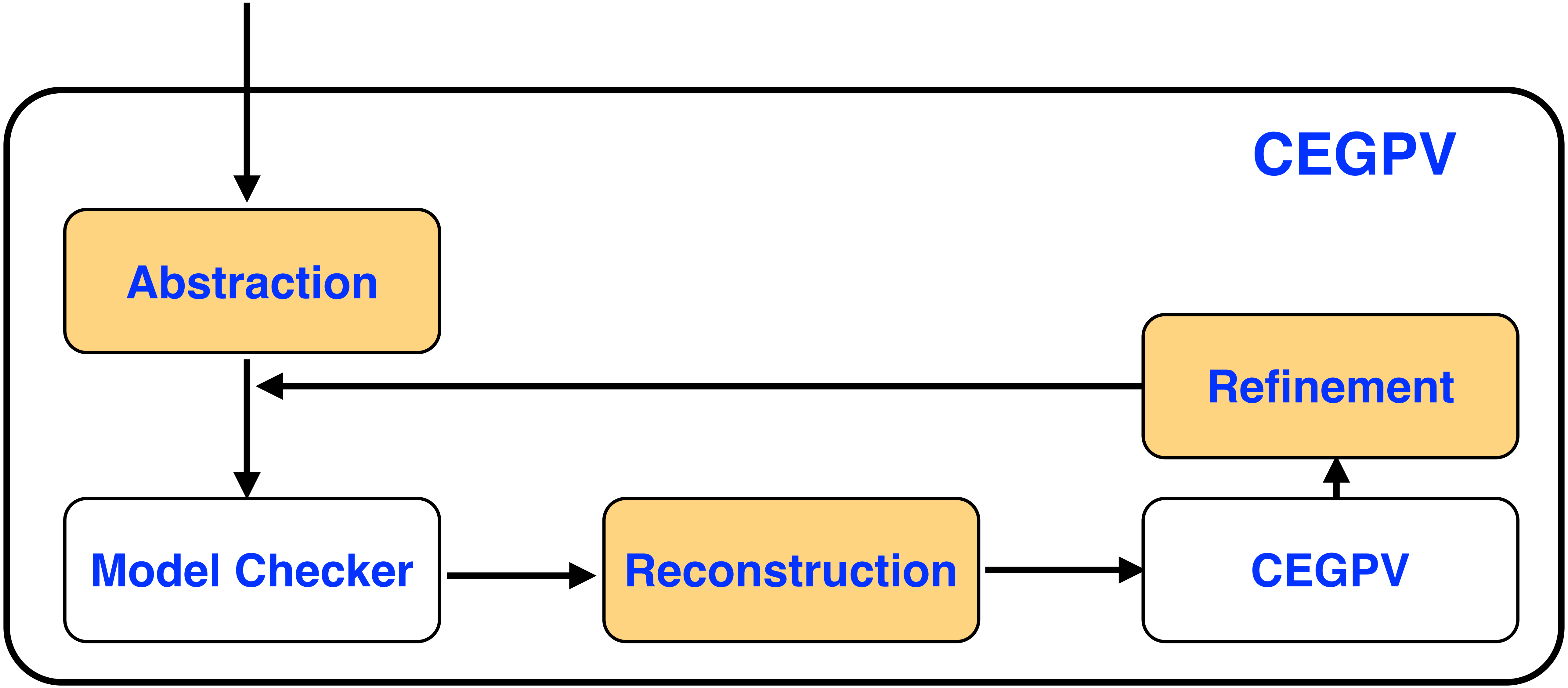
correct

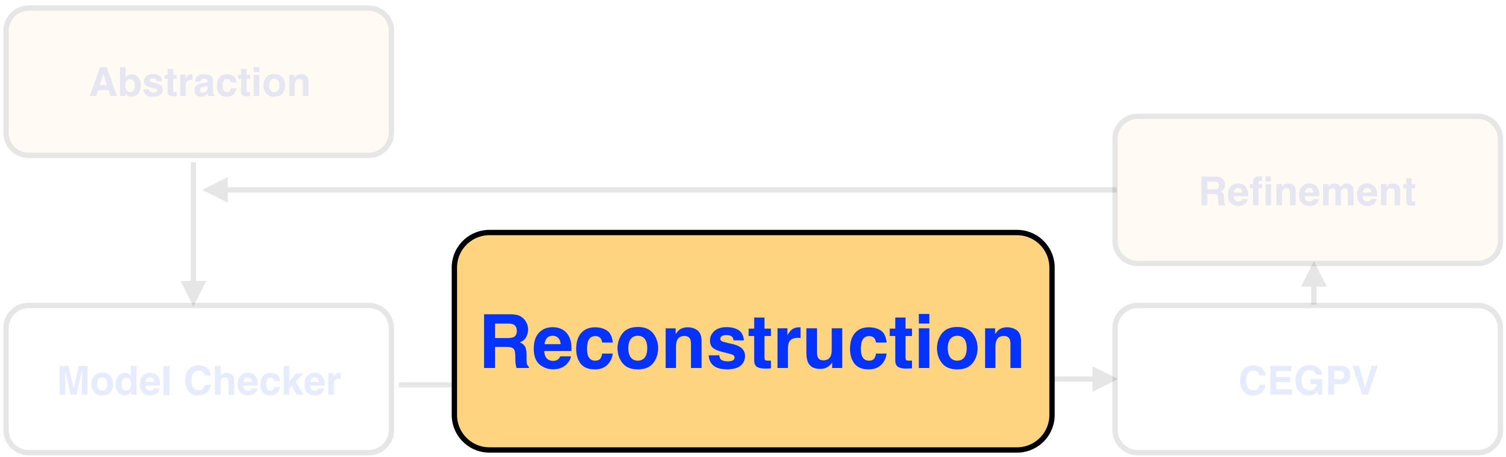
spurious

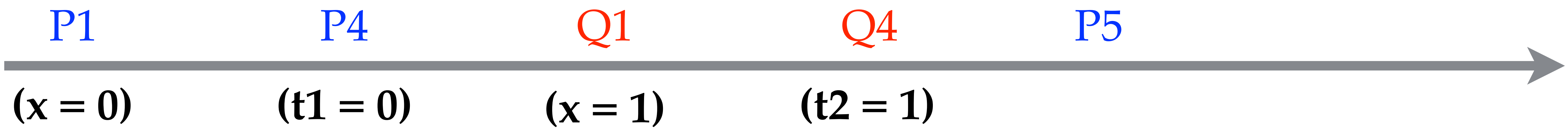
Over-approximation

```
var: x,      t1, t2

P1:  x = *                      Q1:  x = *

P4:  t1 = x                     Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

P1	P4	Q1	Q4	P5
(x = 0)	(t1 = 0)	(x = 1)	(t2 = 1)	


CEGPV
Abstraction
Refinement
Model Checker
Reconstruction
CEGPV





P1 (x = 0)

P4 (t1 = 0)

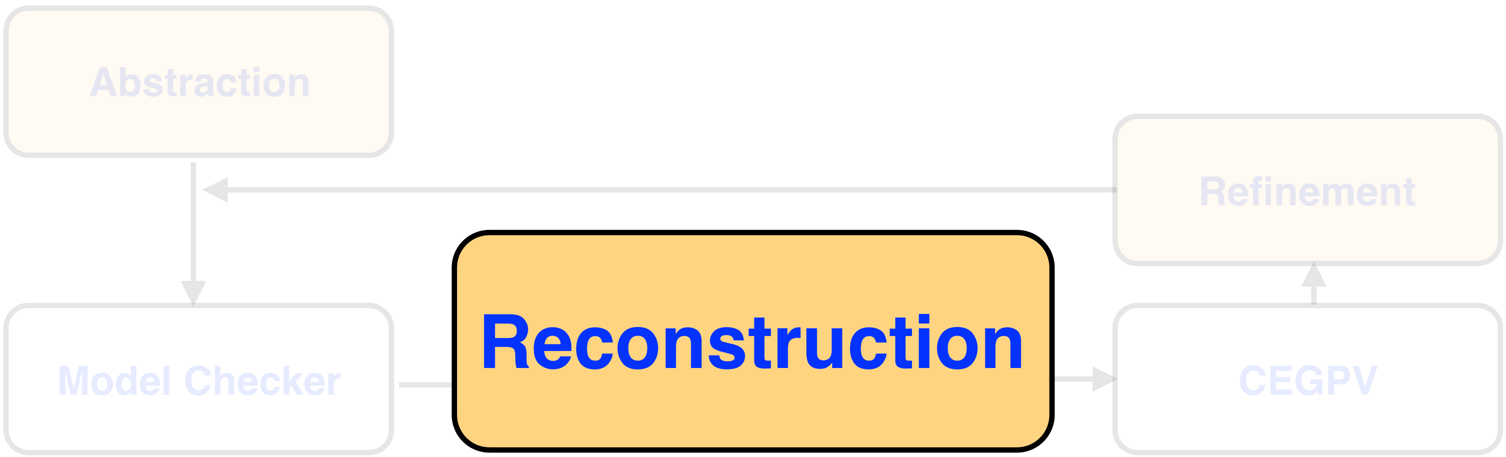
Q1 (x = 1)

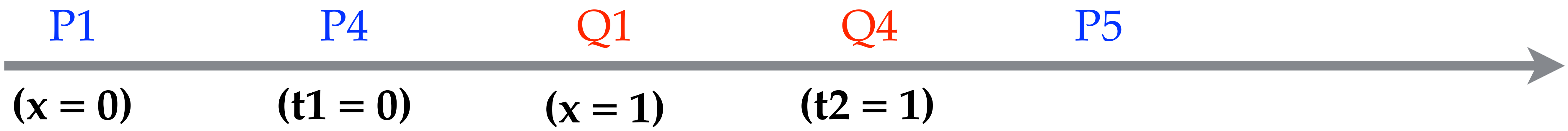
Q4 (t2 = 1)

P5

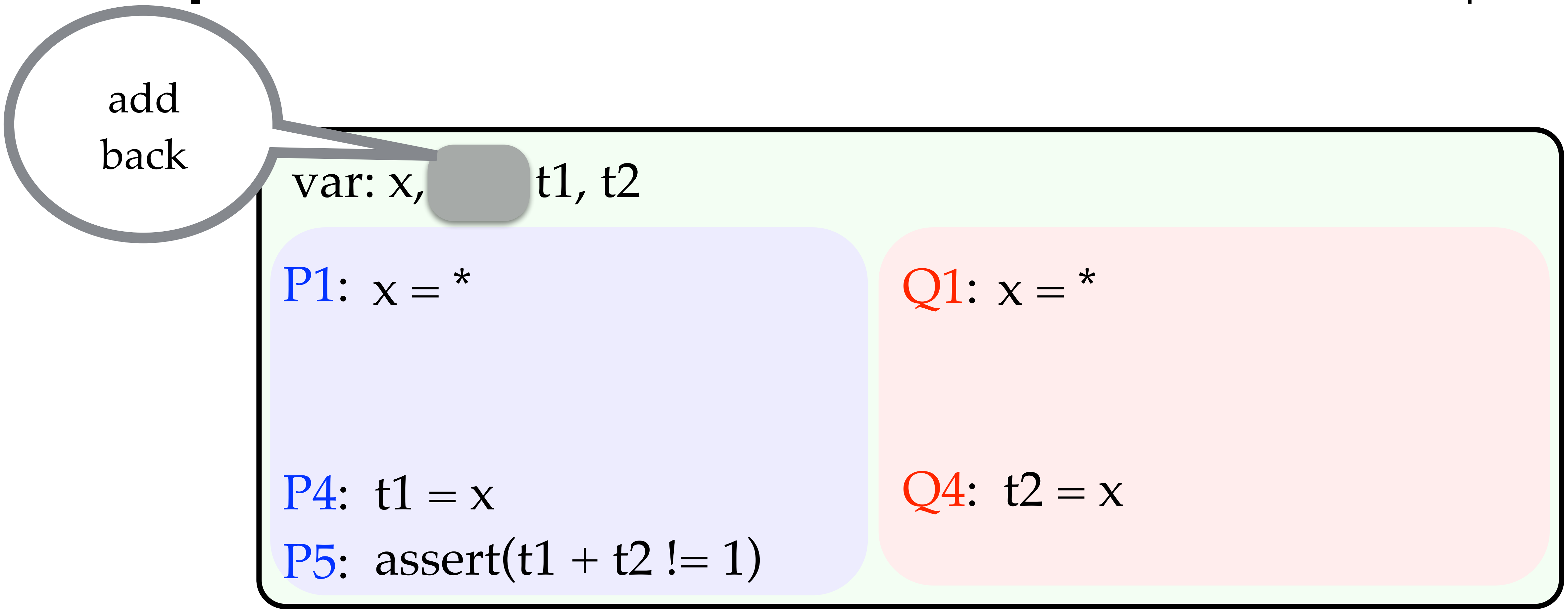
1. **Add back**, **update** variables and instructions
2. **Respect the flow** of instructions in counter-example

```
var: x,  t1, t2

P1:  x = *                      Q1:  x = *



P4:  t1 = x                     Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

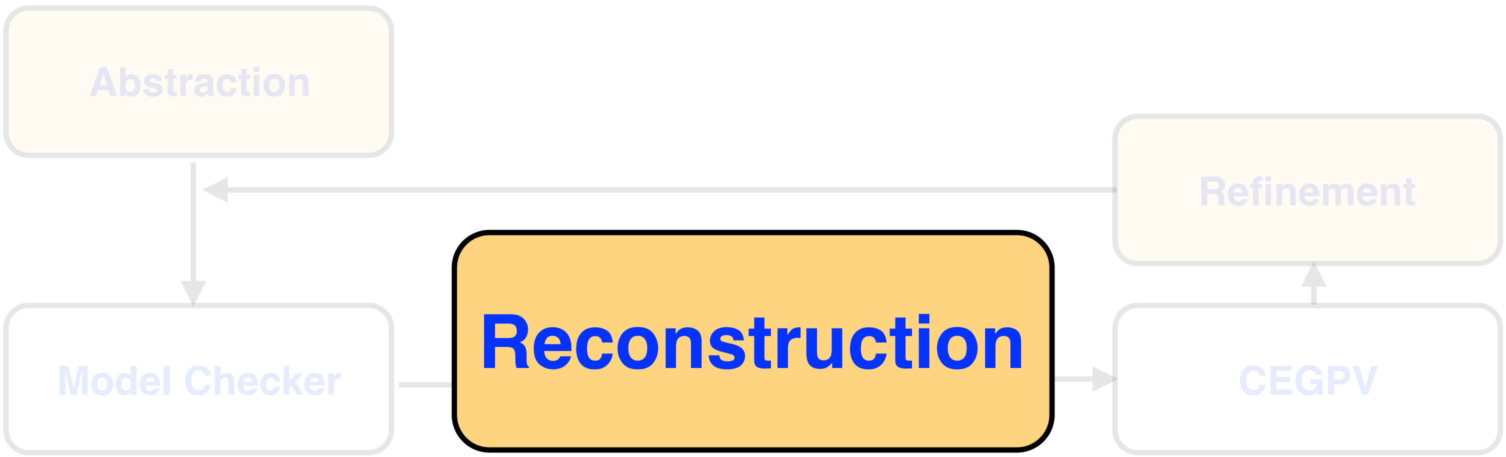




1. **Add back**, **update** variables and instructions
2. **Respect the flow** of instructions in counter-example



```
var: x,      t1, t2

P1:  x = *                     Q1:  x = *

P4:  t1 = x                    Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

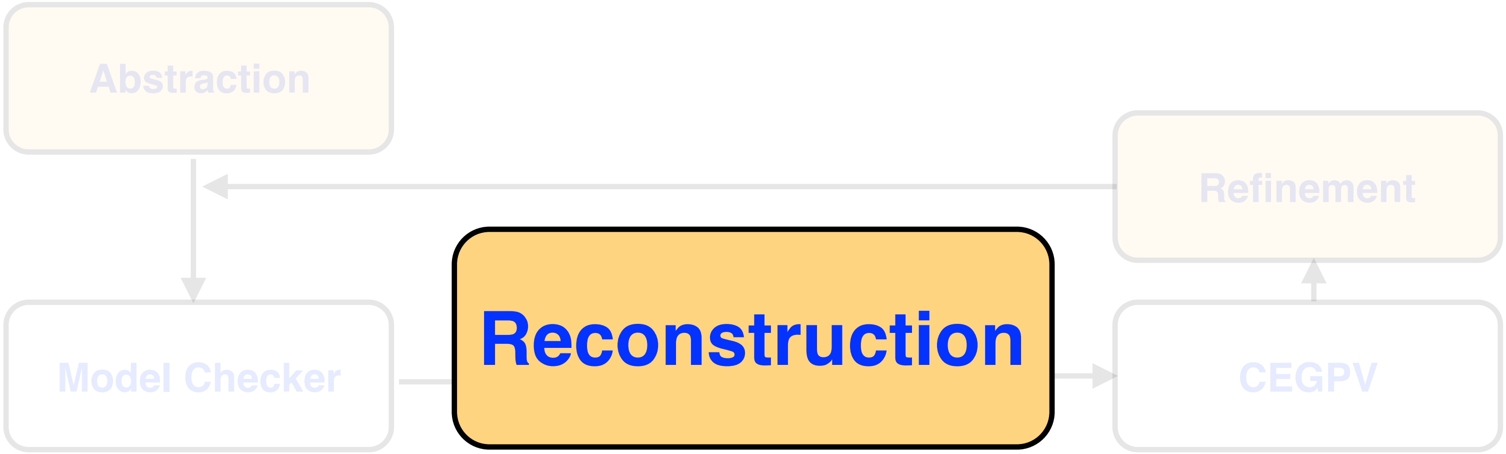


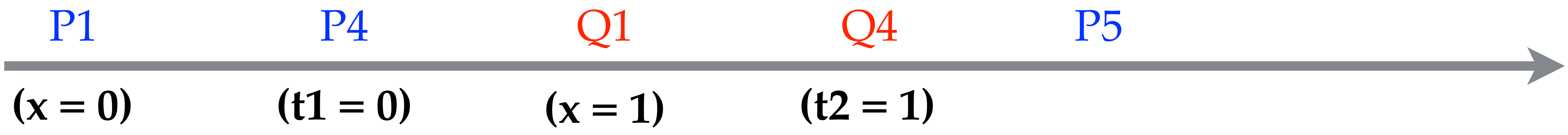
P1	P4	Q1	Q4	P5
(x = 0)	**(t1 = 0)**	**(x = 1)**	**(t2 = 1)**	

1. **Add back**, **update** variables and instructions
2. **Respect the flow** of instructions in counter-example

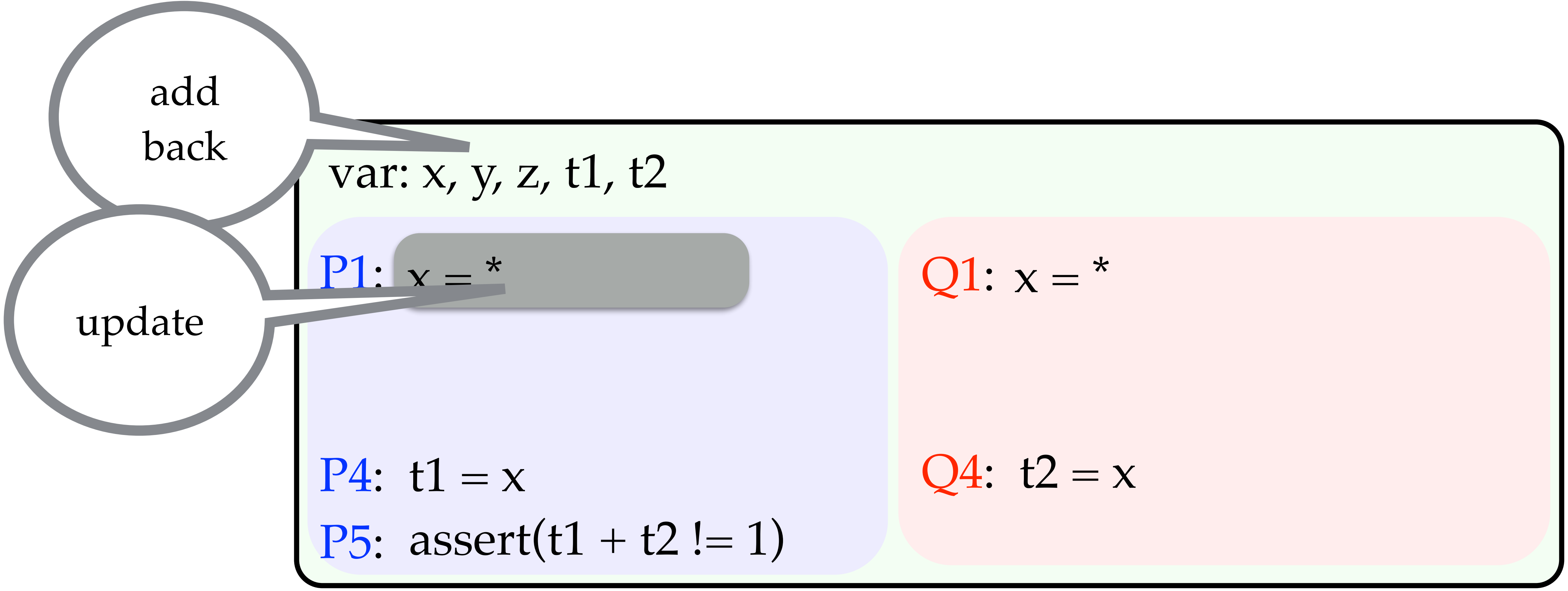
add back

```
var: x, y, z, t1, t2

P1:  x = *                          Q1:  x = *


P4:  t1 = x                         Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

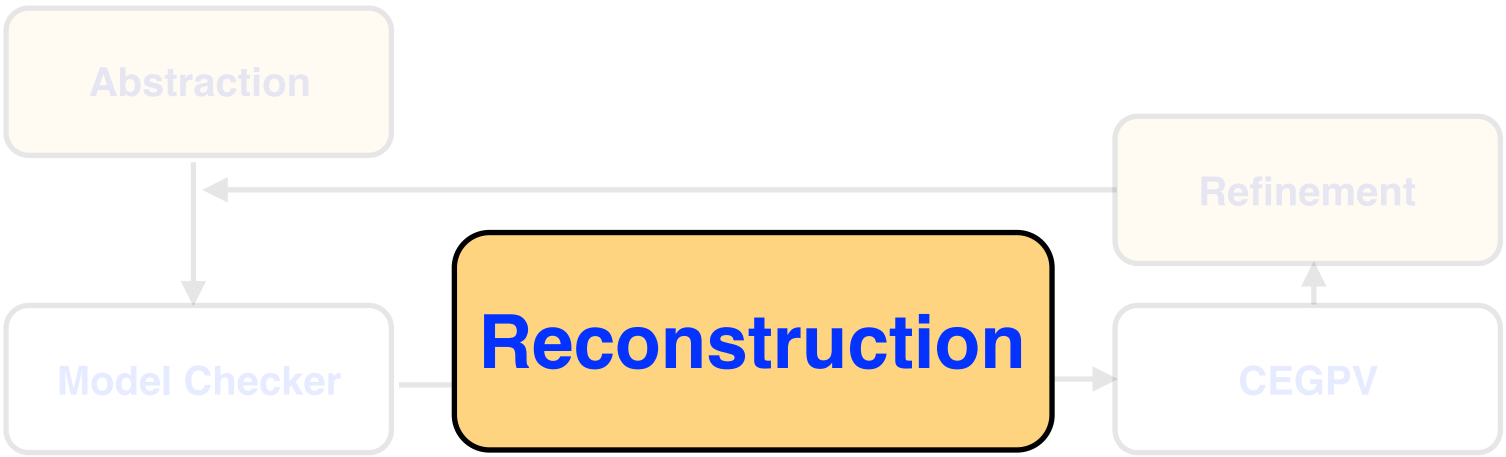




1. **Add back**, **update** variables and instructions
2. **Respect the flow** of instructions in counter-example

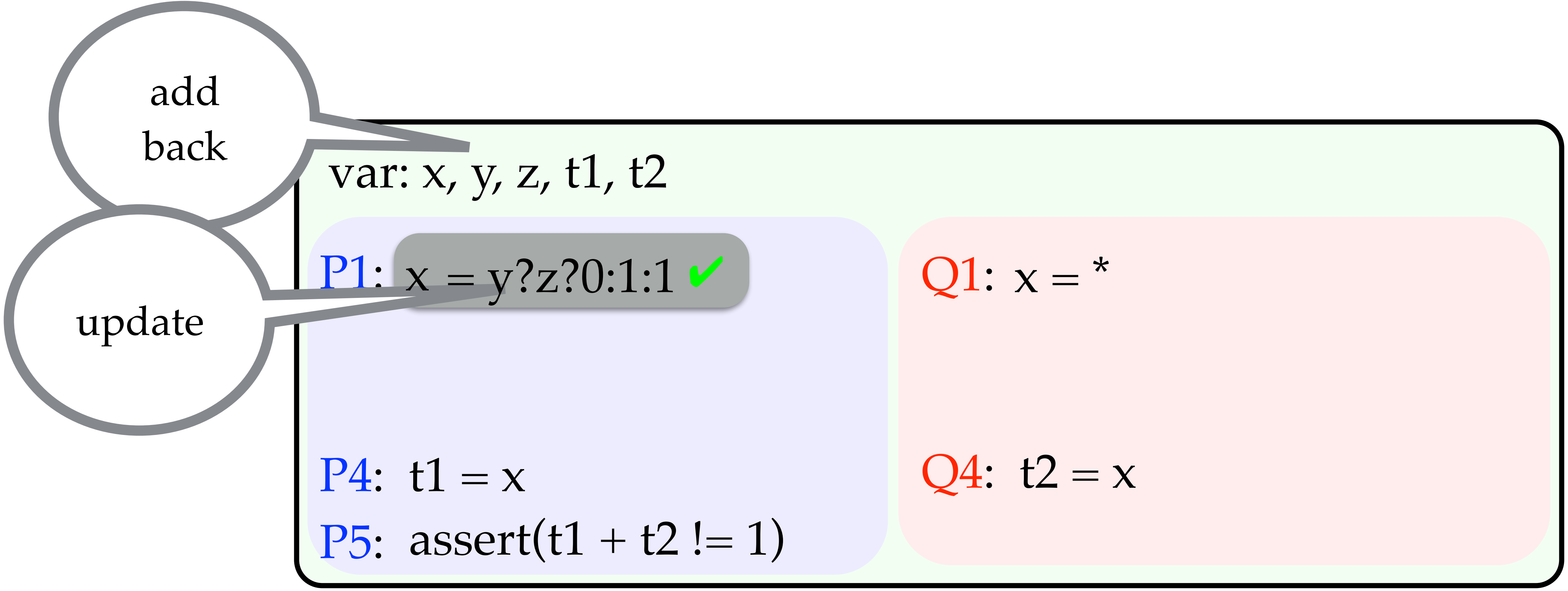


```
var: x, y, z, t1, t2
P1: x = *                Q1: x = *

P4: t1 = x               Q4: t2 = x
P5: assert(t1 + t2 != 1)
```

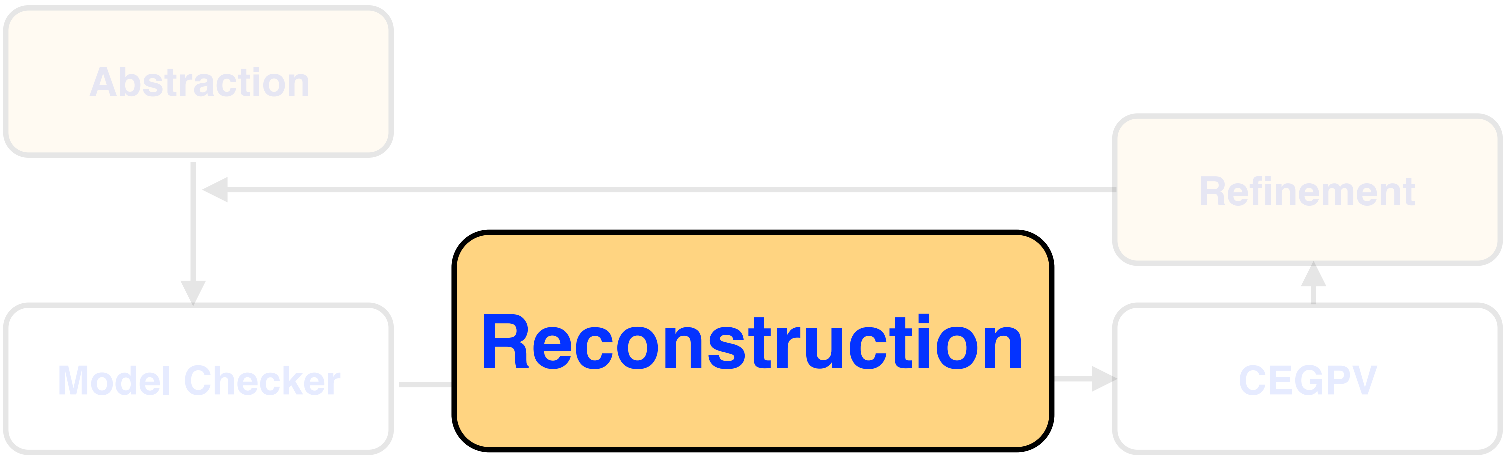


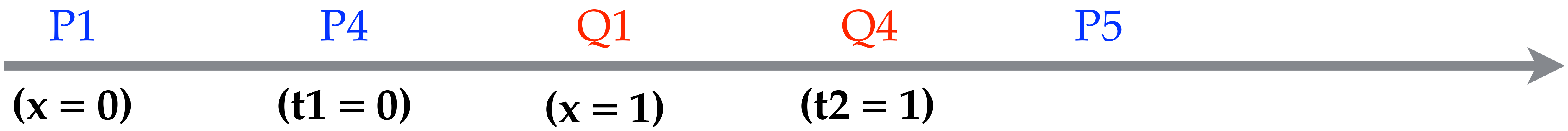
P1	P4	Q1	Q4	P5
(x = 0)	**(t1 = 0)**	**(x = 1)**	**(t2 = 1)**	

1. **Add back**, **update** variables and instructions
2. **Respect the flow** of instructions in counter-example



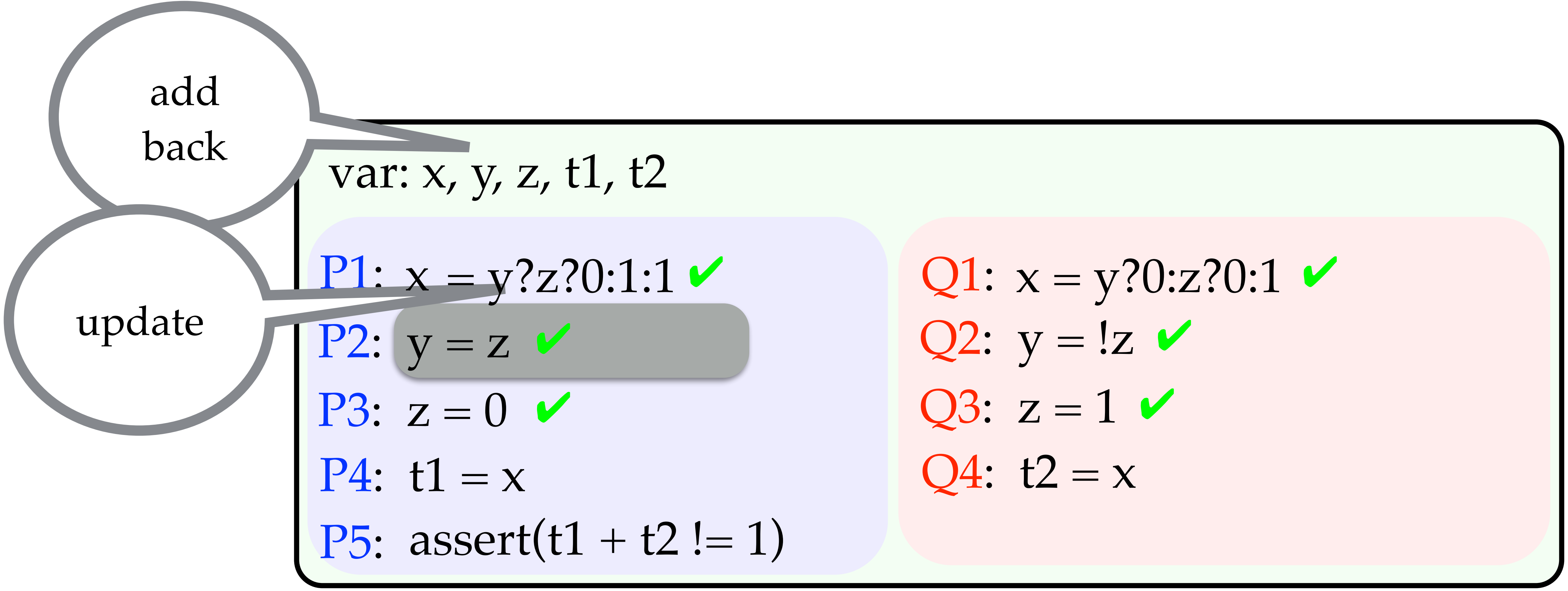
```
var: x, y, z, t1, t2
P1: x = y?z?0:1:1 ✓        Q1: x = *
P4: t1 = x                 Q4: t2 = x
P5: assert(t1 + t2 != 1)
```



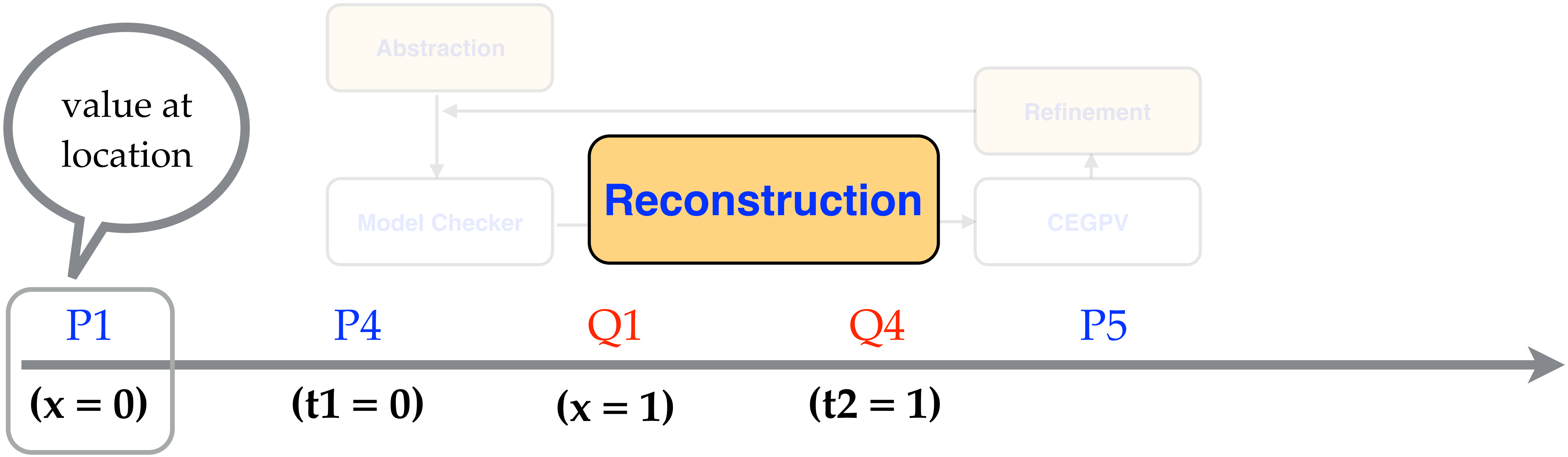


1. **Add back**, **update** variables and instructions

2. **Respect the flow** of instructions in counter-example



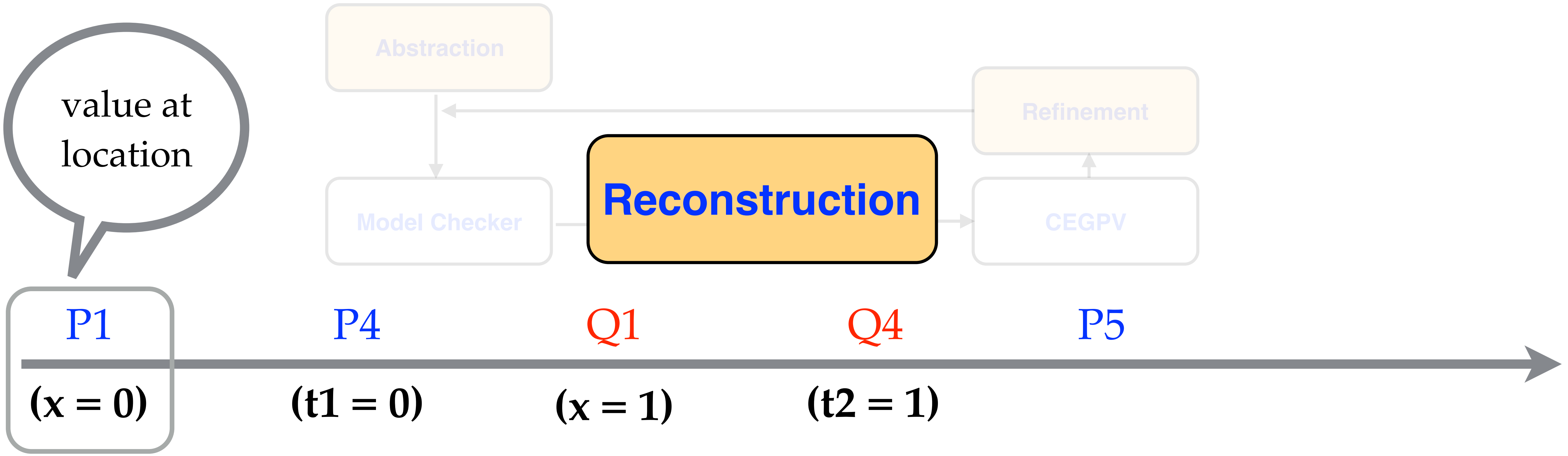
```
var: x, y, z, t1, t2

P1: x = y?z?0:1:1 ✓          Q1: x = y?0:z?0:1 ✓
P2: y = z ✓                  Q2: y = !z ✓
P3: z = 0 ✓                  Q3: z = 1 ✓
P4: t1 = x                   Q4: t2 = x
P5: assert(t1 + t2 != 1)
```



1. **Add back**, **update** variables and instructions
2. **Respect the flow** of instructions in counter-example

check value

```
var: x, y, z, t1, t2

P1:  x = y?z?0:1:1           Q1:  x = y?0:z?0:1
P2:  y = z                   Q2:  y = !z
P3:  z = 0                   Q3:  z = 1
P4:  t1 = x                  Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

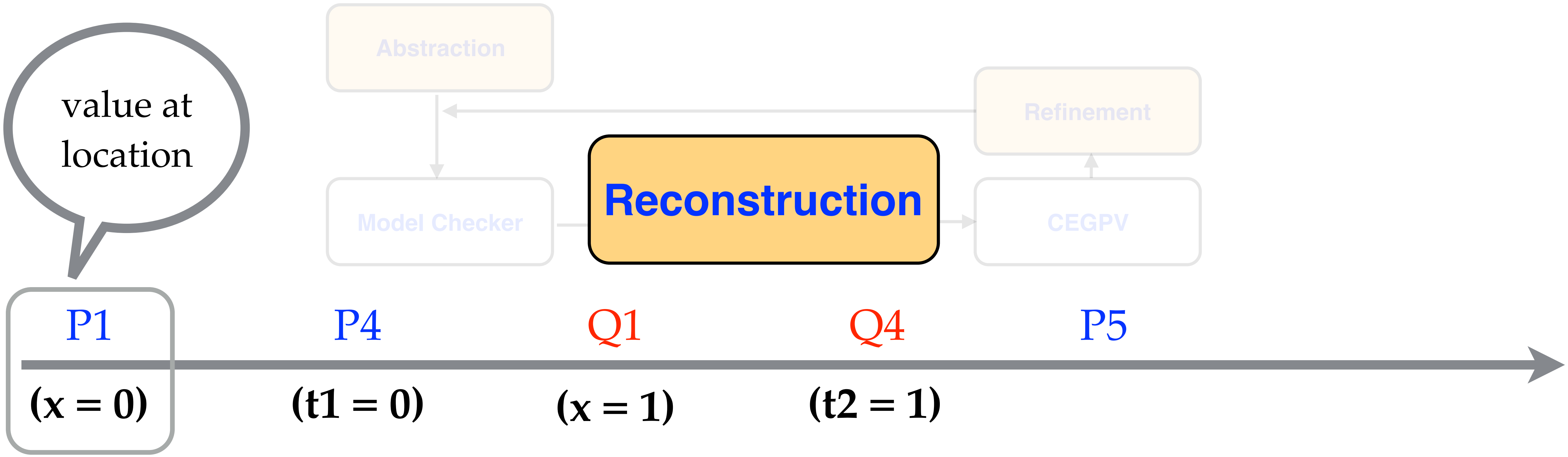


1. **Add back**, **update** variables and instructions

2. **Respect the flow** of instructions in counter-example

check value

```
var: x, y, z, t1, t2

P1:  assume 0 == y?z?0:1:1 ✓
P2:  y = z
P3:  z = 0
P4:  t1 = x
P5:  assert(t1 + t2 != 1)

Q1:  x = y?0:z?0:1
Q2:  y = !z
Q3:  z = 1
Q4:  t2 = x
```



value at location

Abstraction

Model Checker

Reconstruction

Refinement

CEGPV

P1 (x = 0)

P4 (t1 = 0)

Q1 (x = 1)

Q4 (t2 = 1)

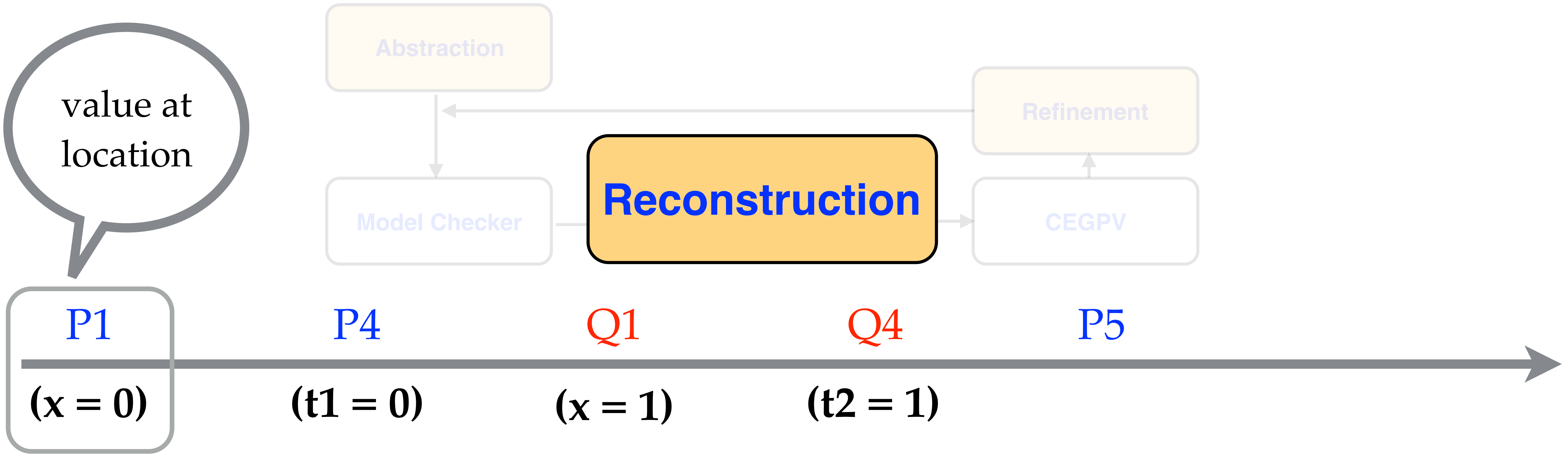
P5

1. **Add back**, **update** variables and instructions

2. **Respect the flow** of instructions in counter-example

check value

```
var: x, y, z, t1, t2

P1:  assume 0 == y?z?0:1:1 ✓    Q1:  assume 1 == y?0:z?0:1 ✓
P2:  y = z                      Q2:  y = !z
P3:  z = 0                      Q3:  z = 1
P4:  assume 0 == 0   ✓          Q4:  assume 1 == 1 ✓
P5:  assert(t1 + t2 != 1)
```



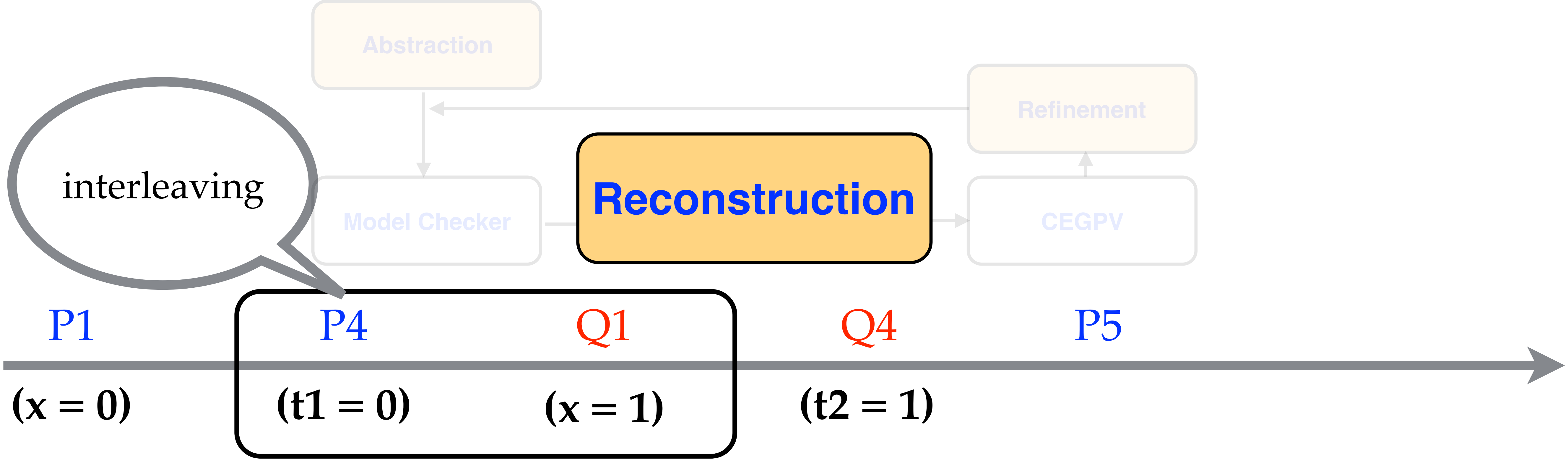
1. **Add back**, **update** variables and instructions

2. **Respect the flow** of instructions in counter-example

removed

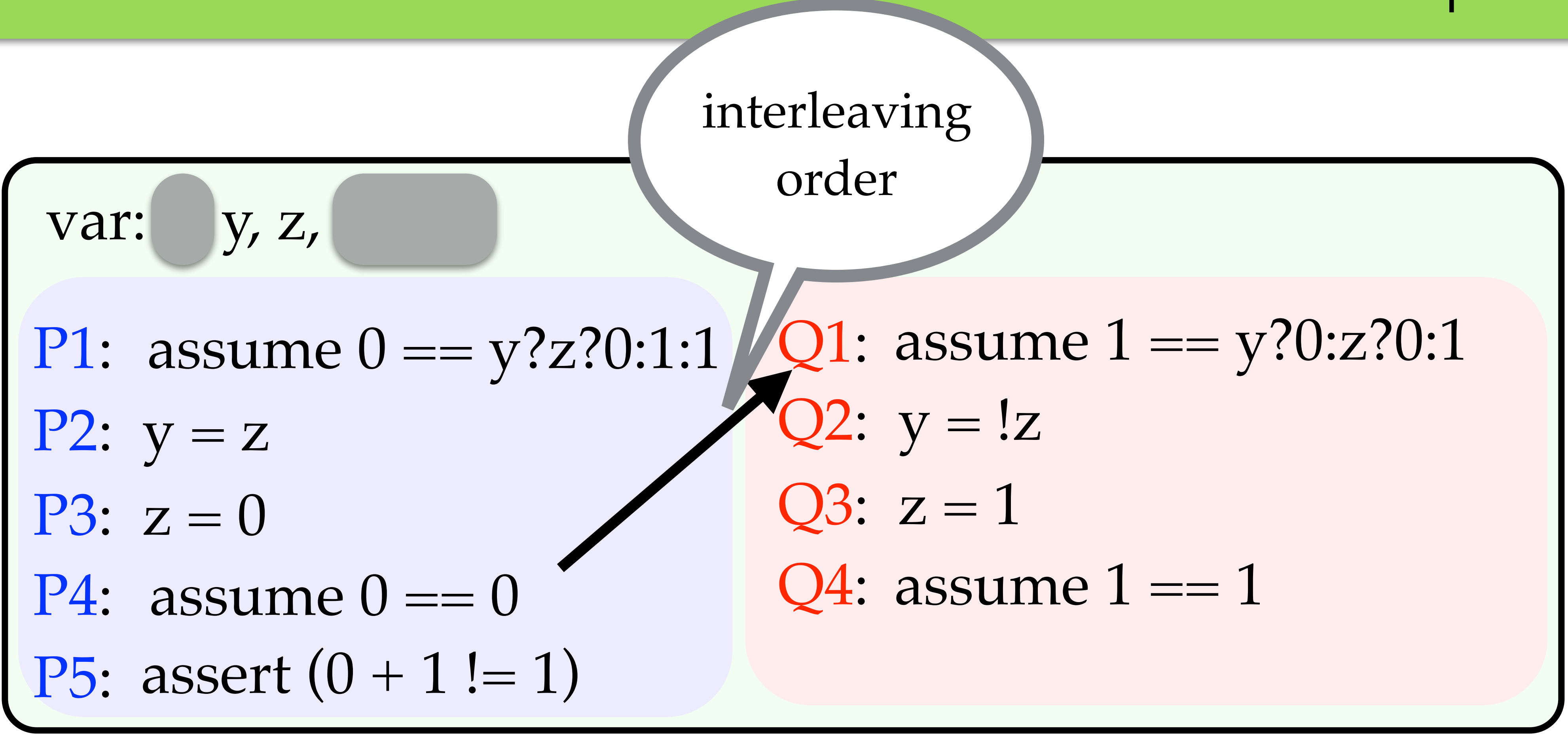
check value

```
var:   y, z,

P1:  assume 0 == y?z?0:1:1 ✓     Q1:  assume 1 == y?0:z?0:1 ✓
P2:  y = z                       Q2:  y = !z
P3:  z = 0                       Q3:  z = 1
P4:  assume 0 == 0  ✓            Q4:  assume 1 == 1 ✓
P5:  assert (0 + 1 != 1)  ✓
```

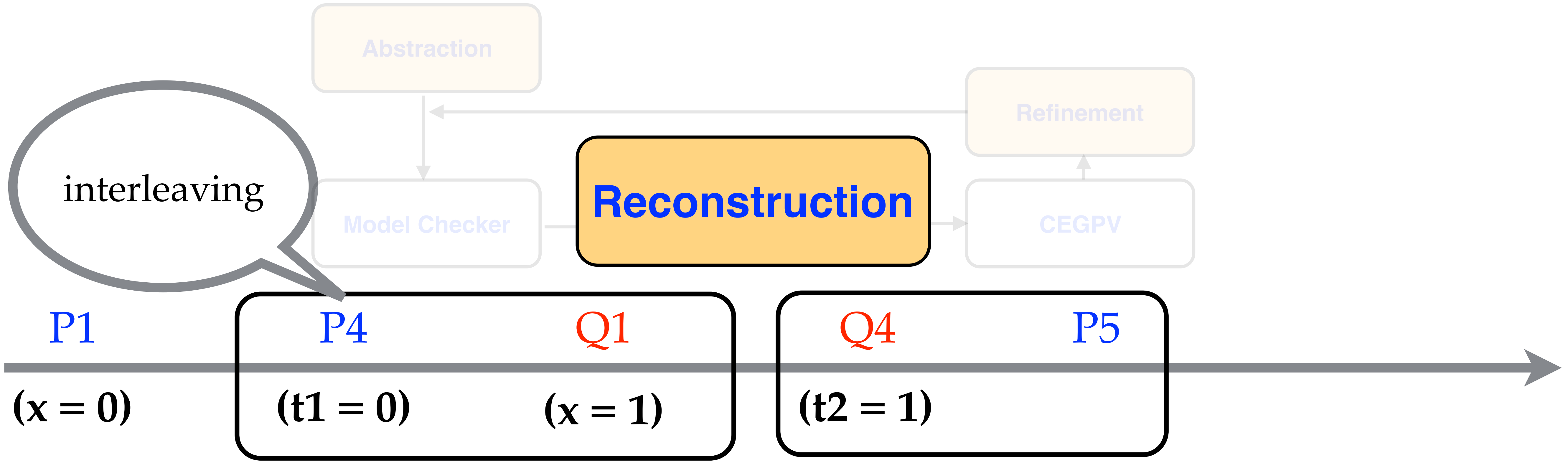


1. Add back, update variables and instructions

2. **Respect the flow** of instructions in counter-example

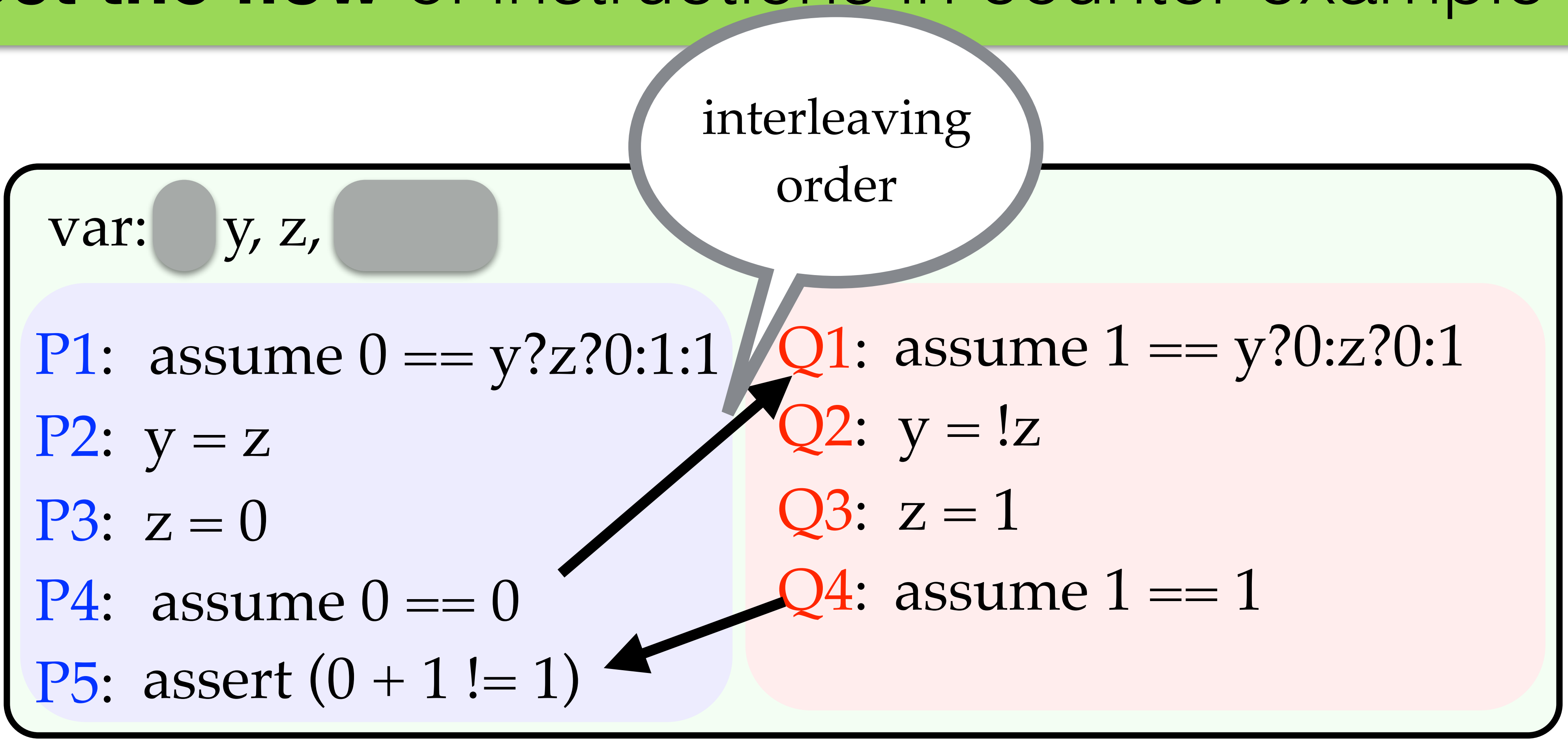


```
var: y, z,

P1:  assume 0 == y?z?0:1:1
P2:  y = z
P3:  z = 0
P4:  assume 0 == 0
P5:  assert (0 + 1 != 1)

Q1:  assume 1 == y?0:z?0:1
Q2:  y = !z
Q3:  z = 1
Q4:  assume 1 == 1
```



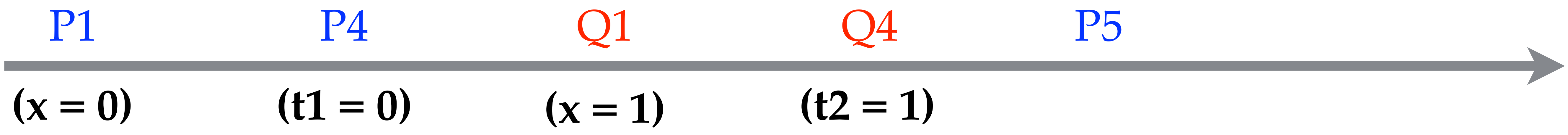
1. Add back, update variables and instructions

2. **Respect the flow** of instructions in counter-example



```
var: y, z,

P1: assume 0 == y?z?0:1:1
P2: y = z
P3: z = 0
P4: assume 0 == 0
P5: assert (0 + 1 != 1)

Q1: assume 1 == y?0:z?0:1
Q2: y = !z
Q3: z = 1
Q4: assume 1 == 1
```

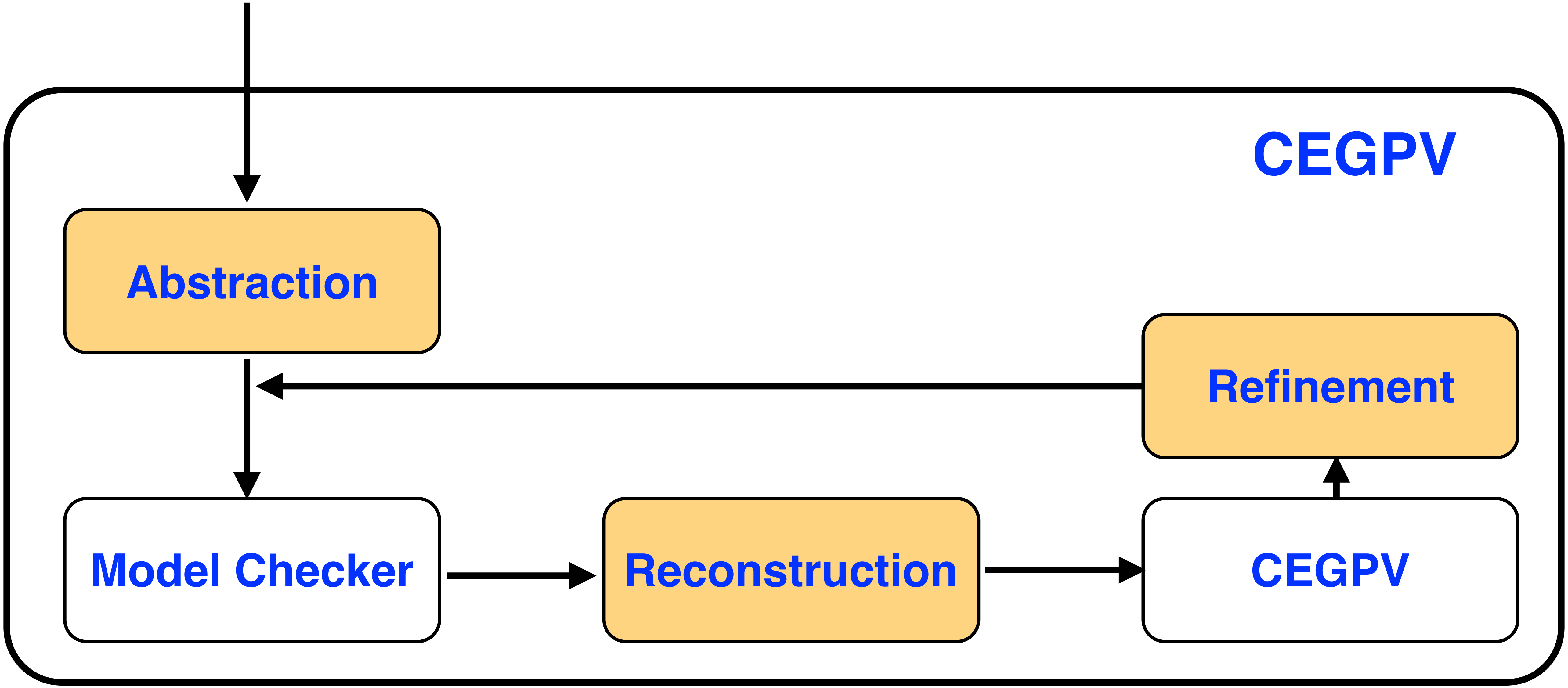
Abstraction
Refinement
Model Checker
Reconstruction
CEGPV

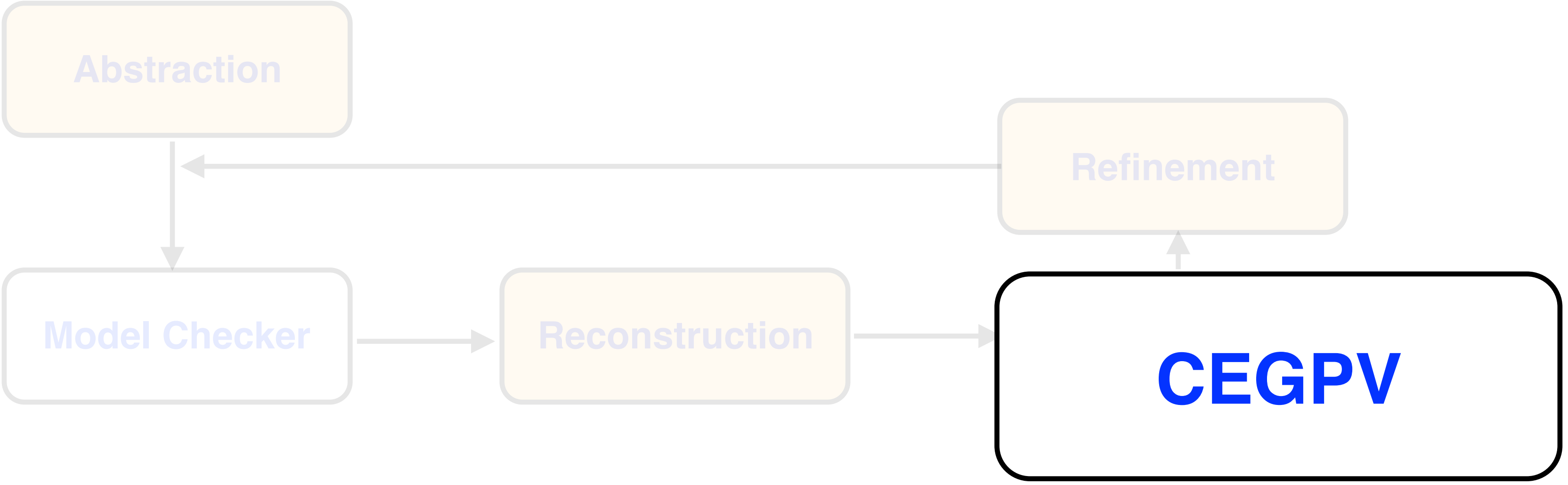


Lemma 2: If the constructed program is **unsafe**, then the original program is **unsafe**.

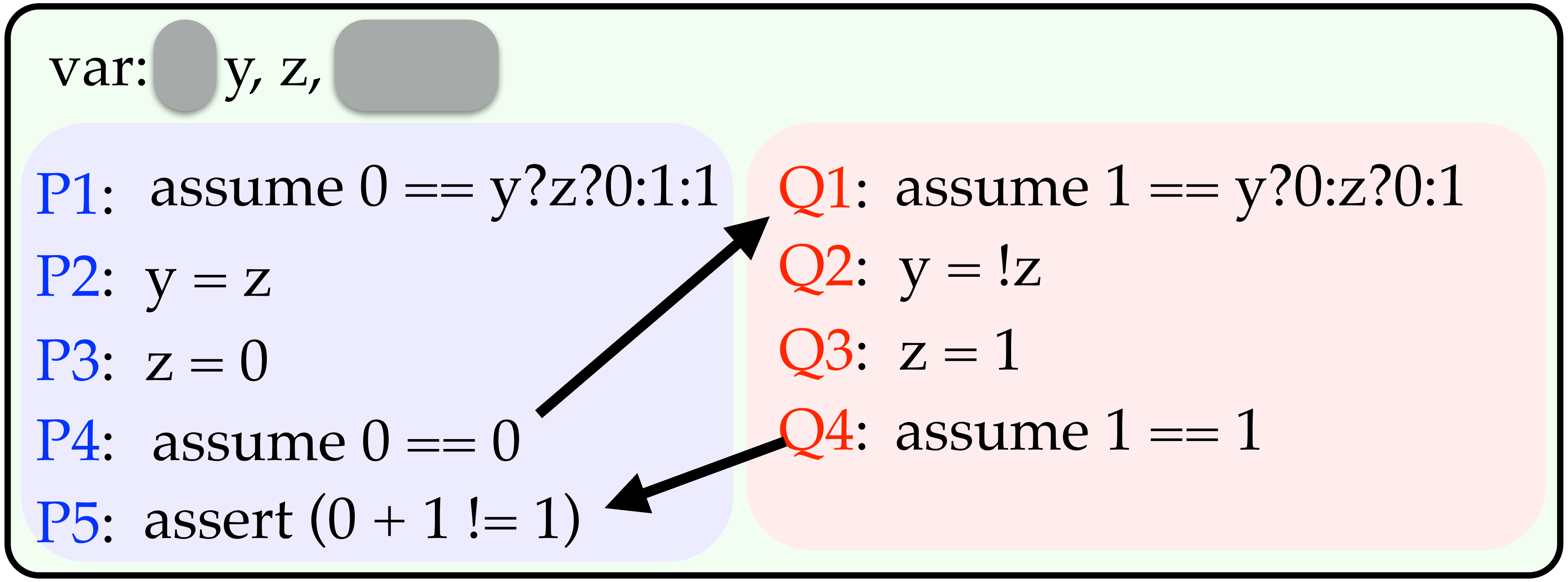
```
var: x, y, z, t1, t2
P1:  x = y?z?0:1:1        Q1:  x = y?0:z?0:1
P2:  y = z                Q2:  y = !z
P3:  z = 0                Q3:  z = 1
P4:  t1 = x               Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

```
var: [redacted] y, z, [redacted]
P1:  assume 0 == y?z?0:1:1      Q1:  assume 1 == y?0:z?0:1
P2:  y = z                      Q2:  y = !z
P3:  z = 0                      Q3:  z = 1
P4:  assume 0 == 0              Q4:  assume 1 == 1
P5:  assert (0 + 1 != 1)
```

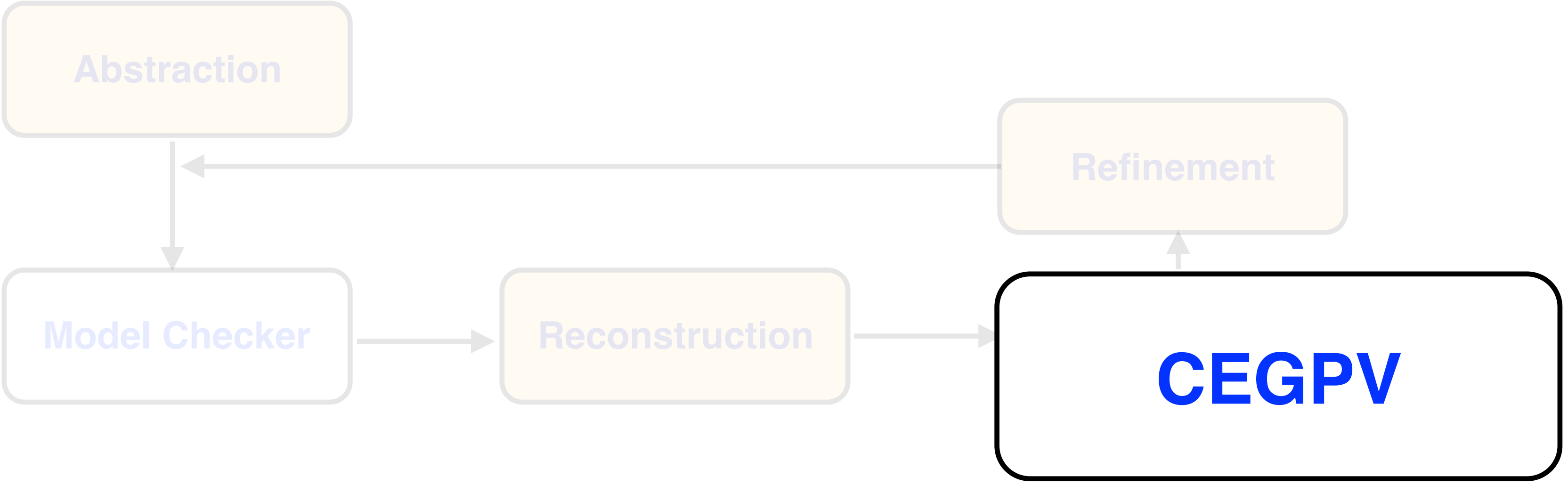

CEGPV
Abstraction
Refinement
Model Checker
Reconstruction
CEGPV



Recursively check the program



```
var: [redacted] y, z, [redacted]

P1: assume 0 == y?z?0:1:1        Q1: assume 1 == y?0:z?0:1
P2: y = z                        Q2: y = !z
P3: z = 0                        Q3: z = 1
P4: assume 0 == 0                Q4: assume 1 == 1
P5: assert (0 + 1 != 1)
```



Abstraction

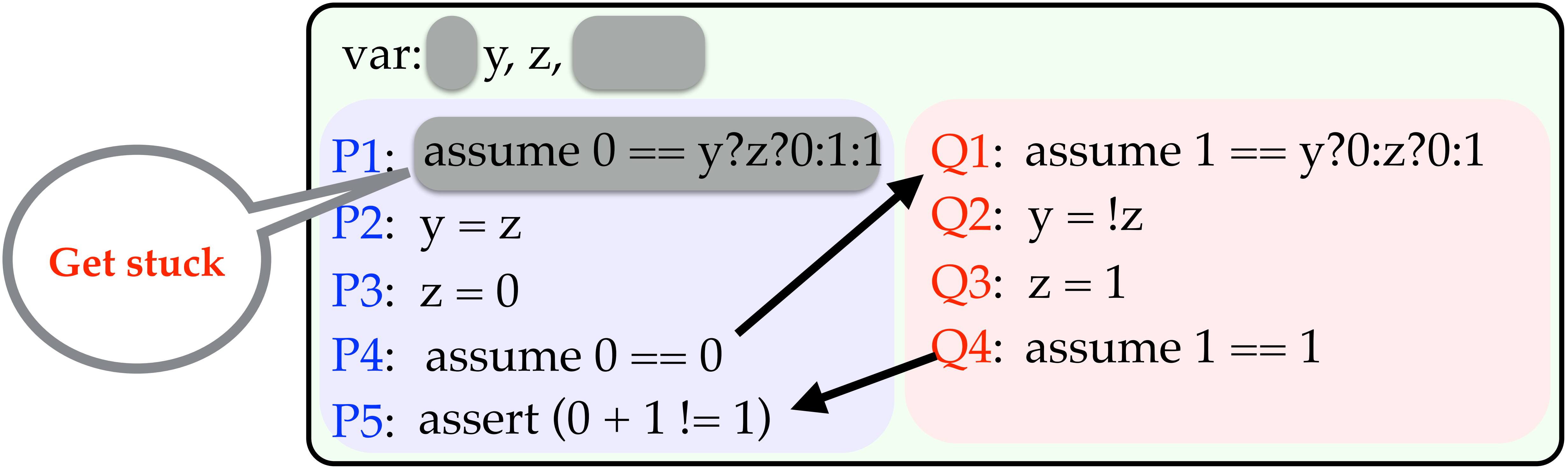
Refinement

Model Checker

Reconstruction

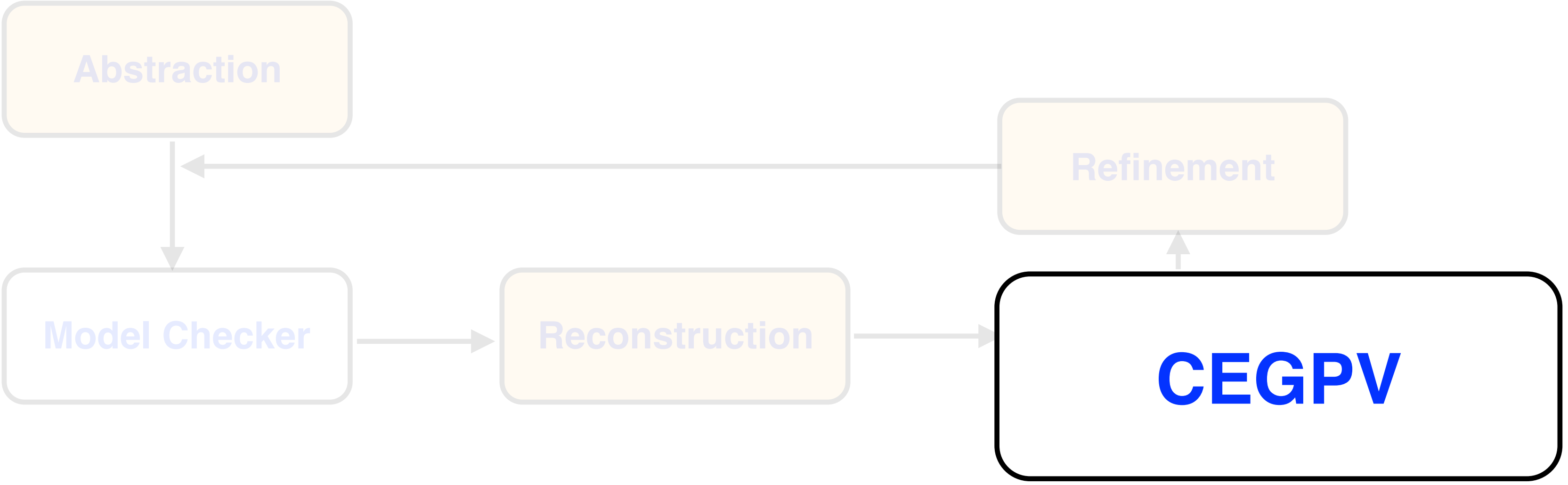
CEGPV

Recursively check the program

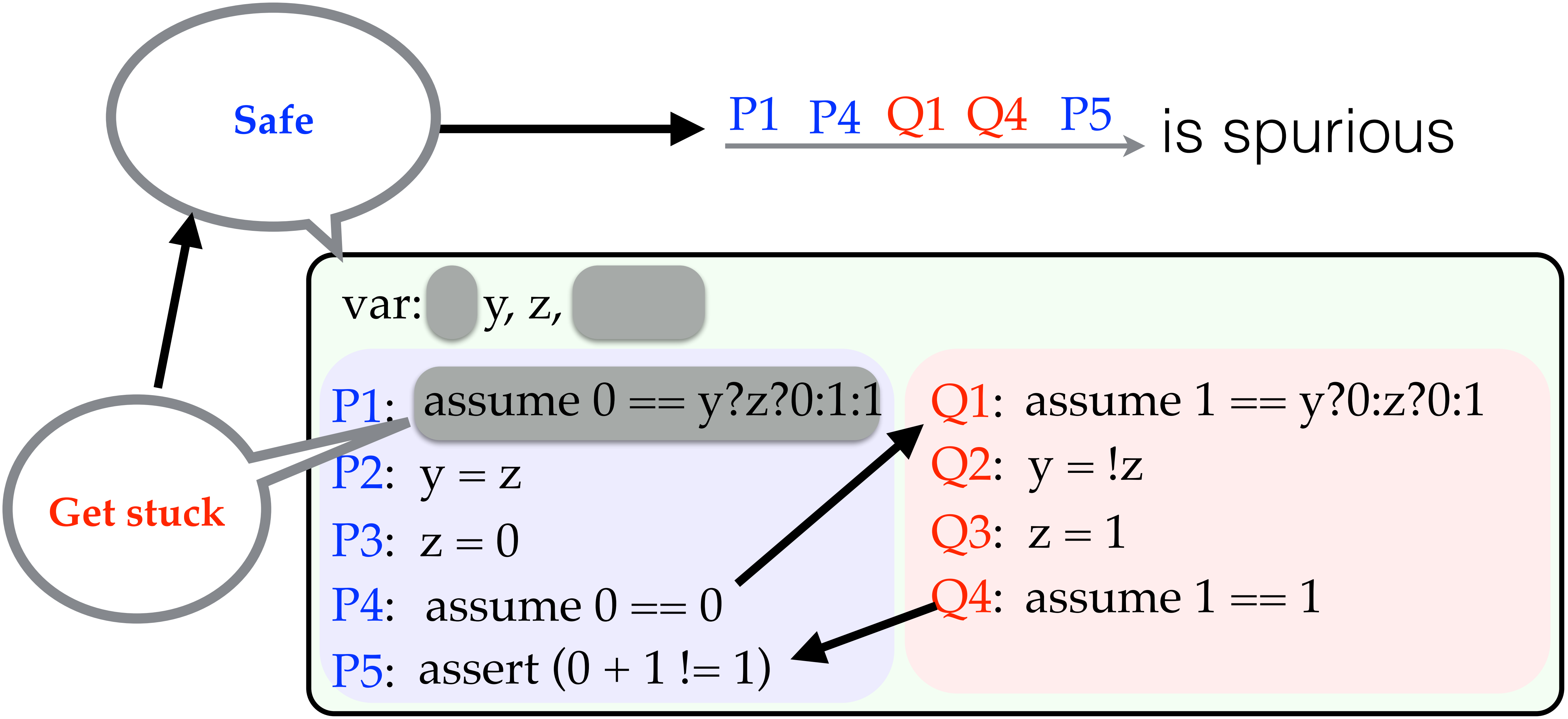


```
var:  y, z,

P1: assume 0 == y?z?0:1:1        Q1: assume 1 == y?0:z?0:1
P2: y = z                        Q2: y = !z
P3: z = 0                        Q3: z = 1
P4: assume 0 == 0                Q4: assume 1 == 1
P5: assert (0 + 1 != 1)
```

Get stuck

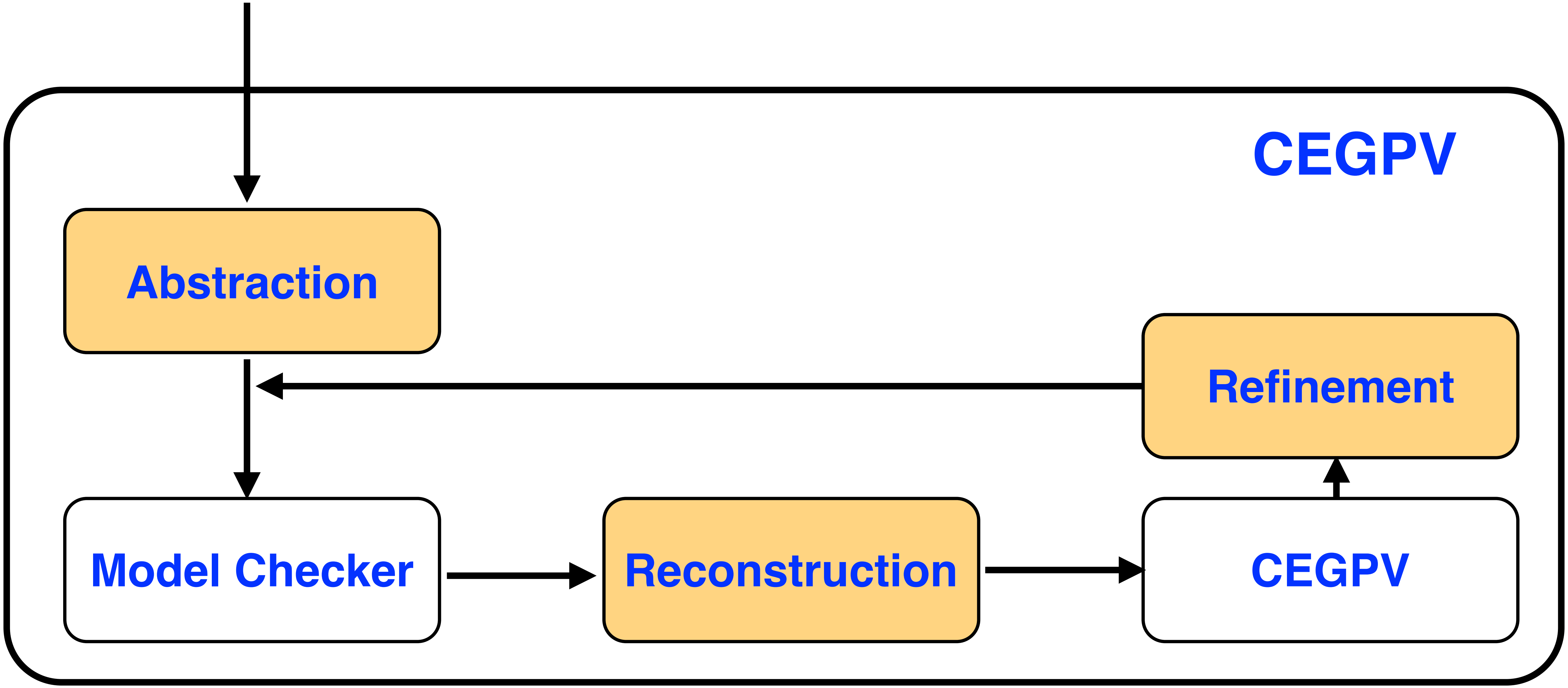


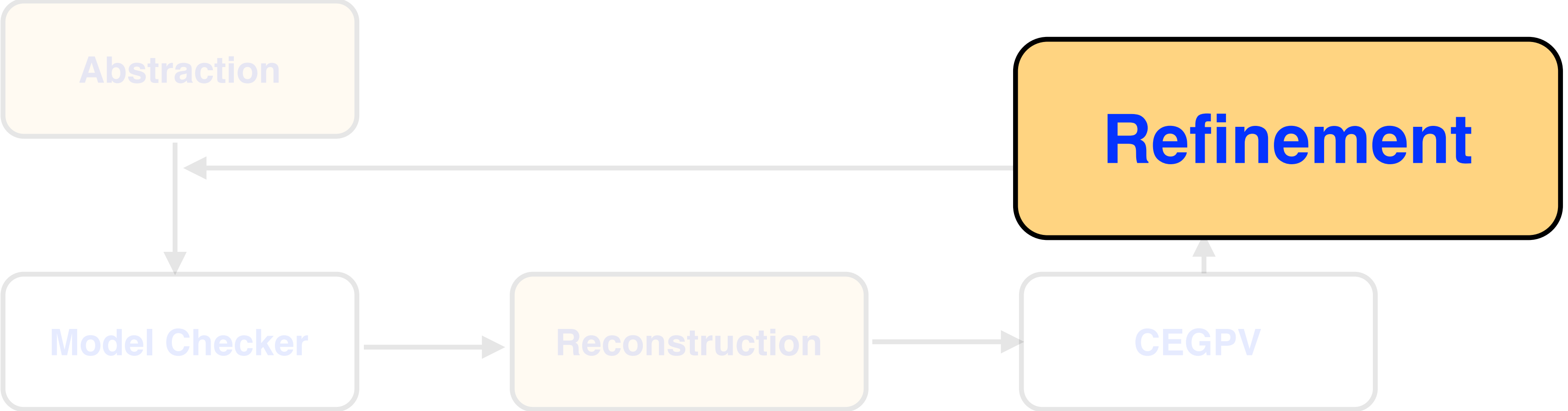
Recursively check the program

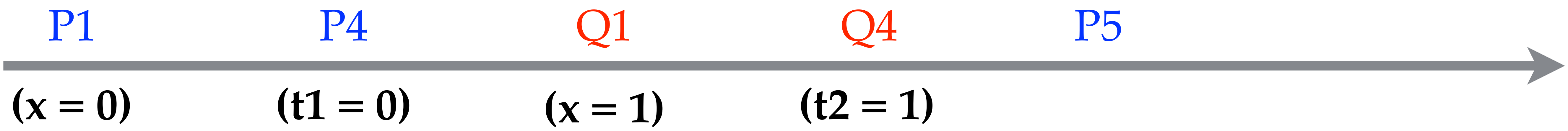


P1 P4 Q1 Q4 P5 is spurious

```
var: y, z,

P1: assume 0 == y?z?0:1:1        Q1: assume 1 == y?0:z?0:1
P2: y = z                        Q2: y = !z
P3: z = 0                        Q3: z = 1
P4: assume 0 == 0                Q4: assume 1 == 1
P5: assert (0 + 1 != 1)
```

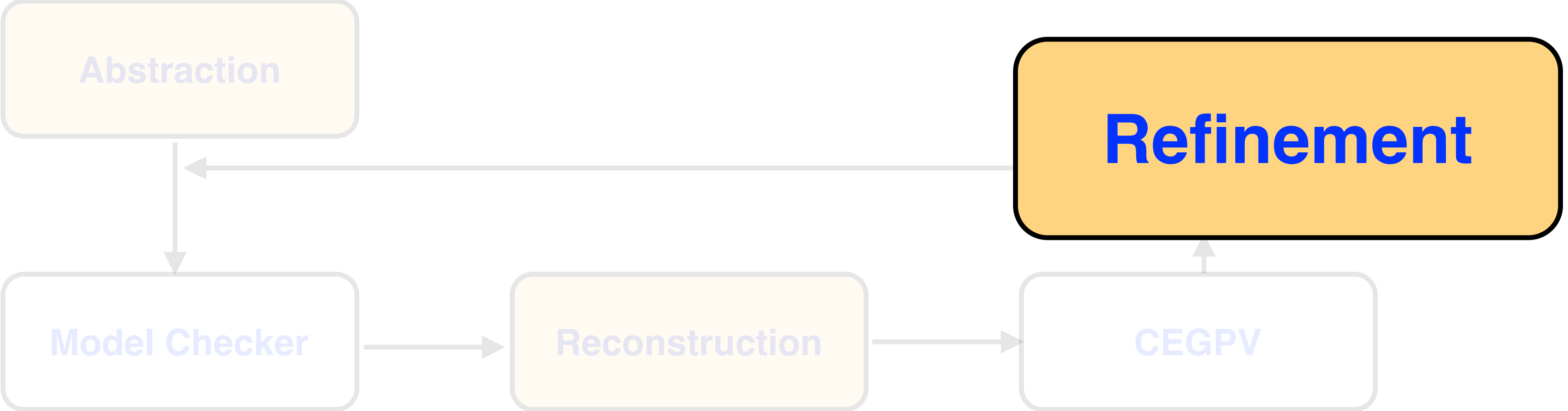

CEGPV
Abstraction
Refinement
Model Checker
Reconstruction
CEGPV



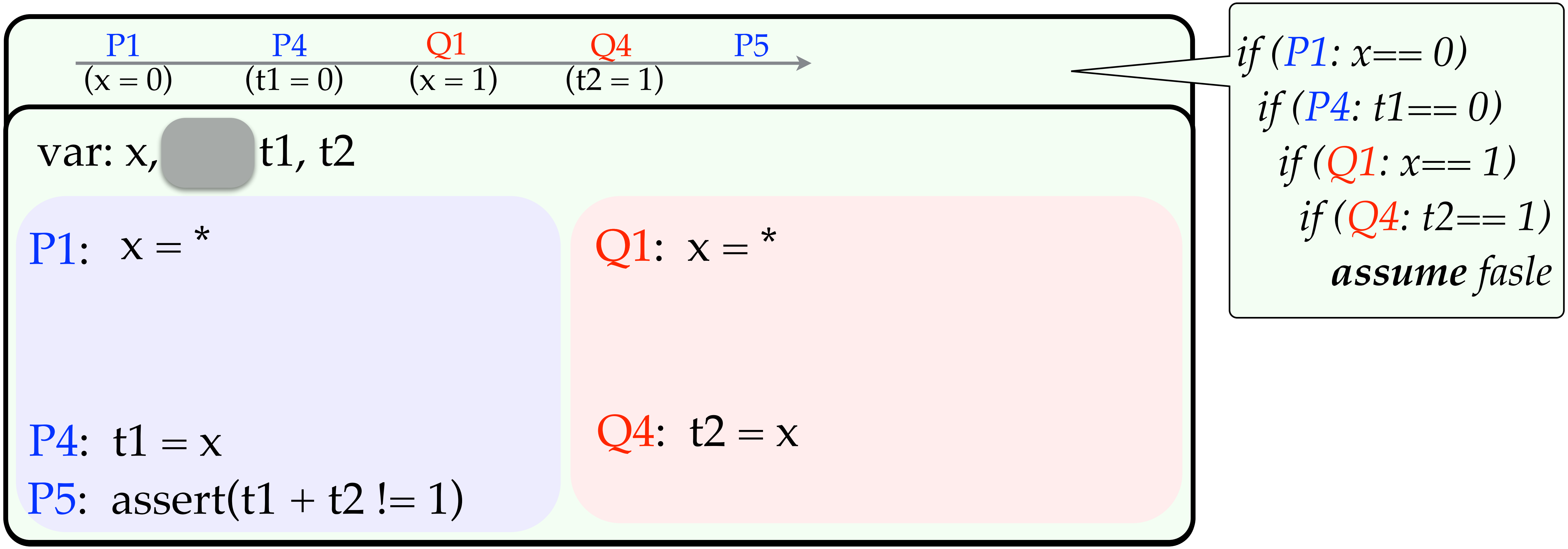


Block the counter-example from the abstraction

```
var: x, t1, t2

P1:  x = *                          Q1:  x = *

P4:  t1 = x                         Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

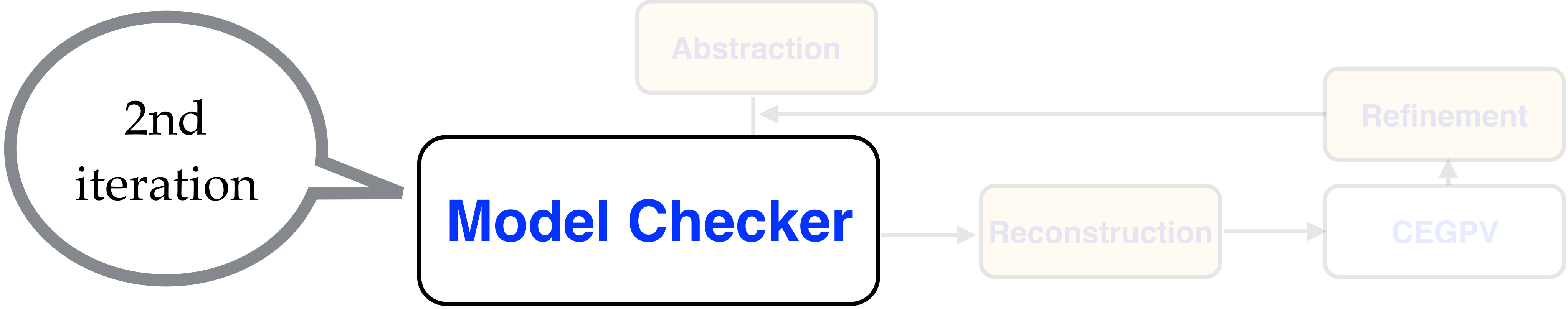


Block the counter-example from the abstraction



```
var: x,  t1, t2

P1:  x = *
P4:  t1 = x
P5:  assert(t1 + t2 != 1)

Q1:  x = *
Q4:  t2 = x
```

```
if (P1: x== 0)
  if (P4: t1== 0)
    if (Q1: x== 1)
      if (Q4: t2== 1)
        assume fasle
```



2nd iteration

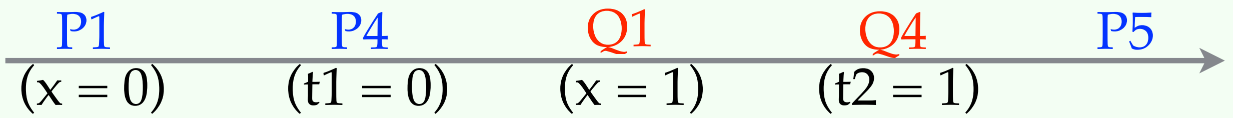
Model Checker

Abstraction

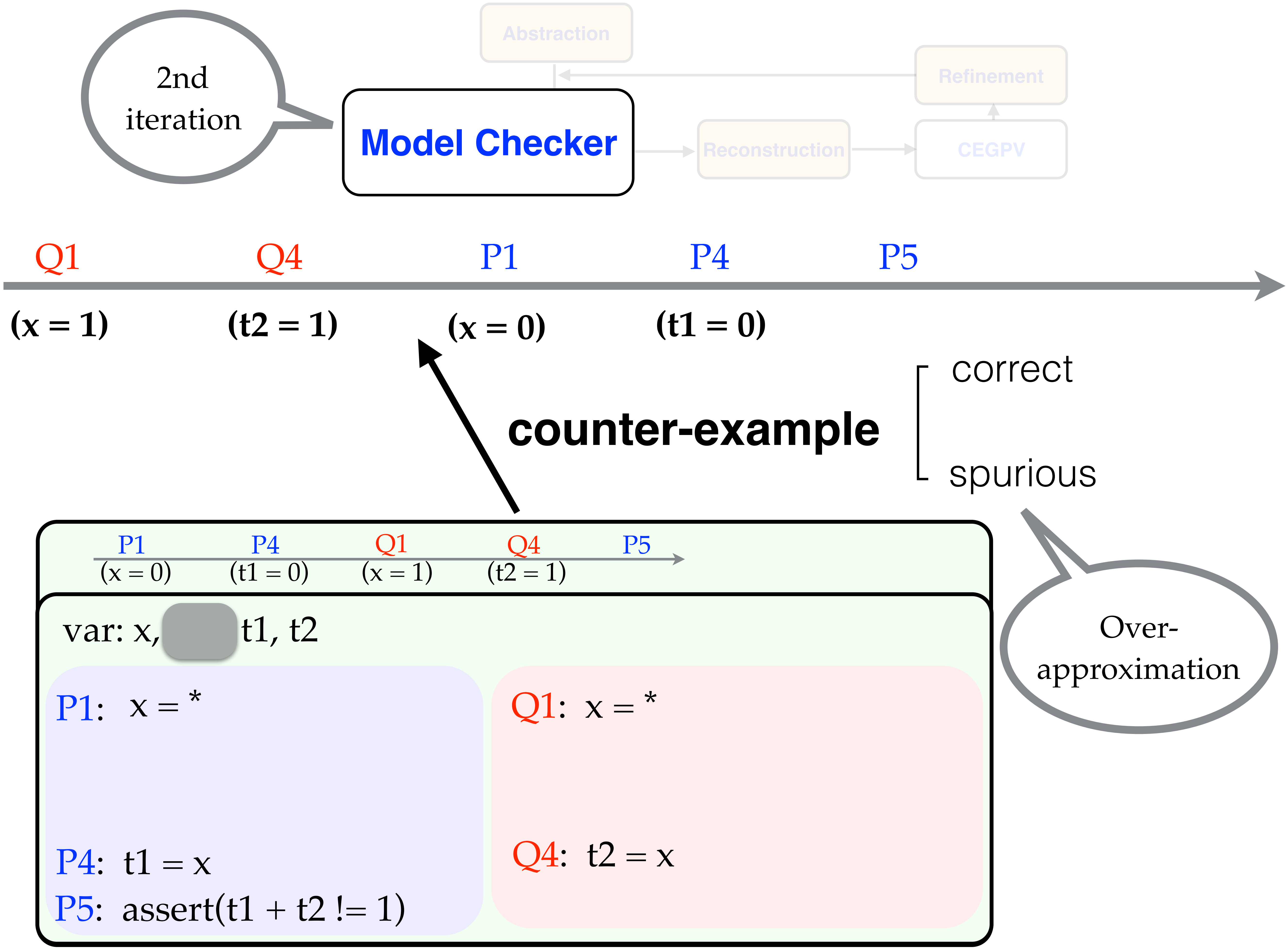
Refinement

Reconstruction

CEGPV



```
var: x,  t1, t2

P1:  x = *                     Q1:  x = *

P4:  t1 = x                    Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```



2nd iteration

Abstraction

Refinement

Model Checker

Reconstruction

CEGPV

Q1 (x = 1)

Q4 (t2 = 1)

P1 (x = 0)

P4 (t1 = 0)

P5

counter-example

correct

spurious

Over-approximation

P1 (x = 0)

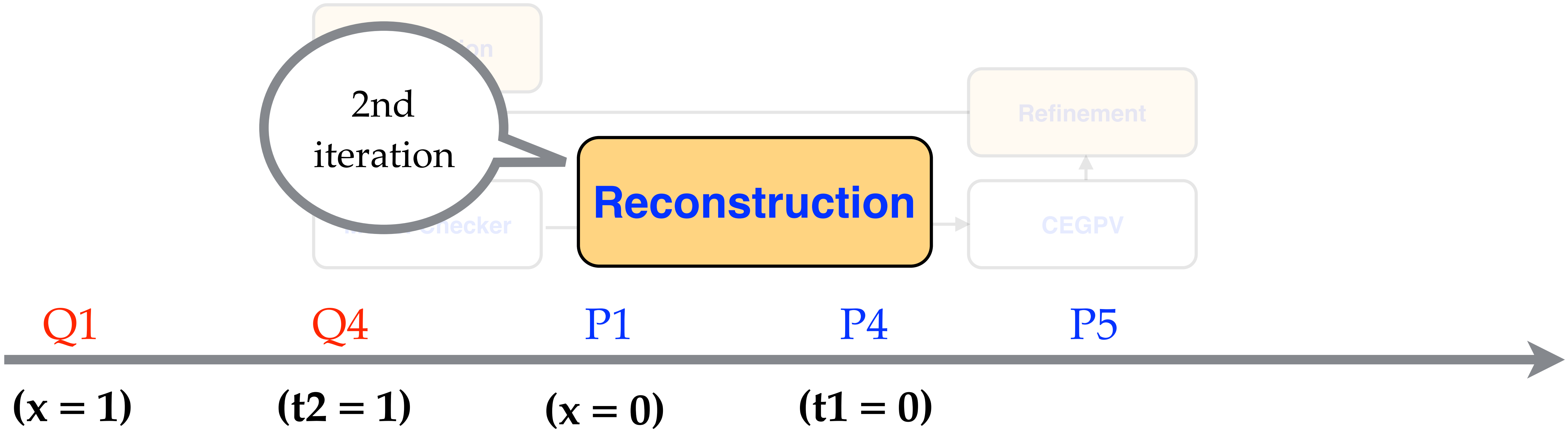
P4 (t1 = 0)

Q1 (x = 1)

Q4 (t2 = 1)

P5

```
var: x,      t1, t2

P1:  x = *                    Q1:  x = *

P4:  t1 = x                   Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```

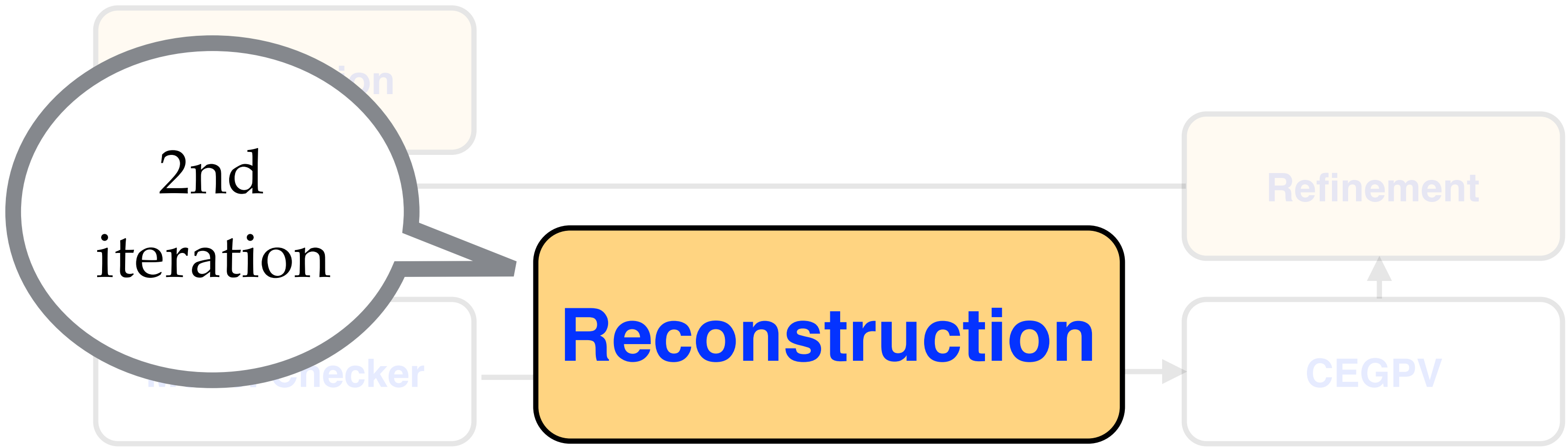


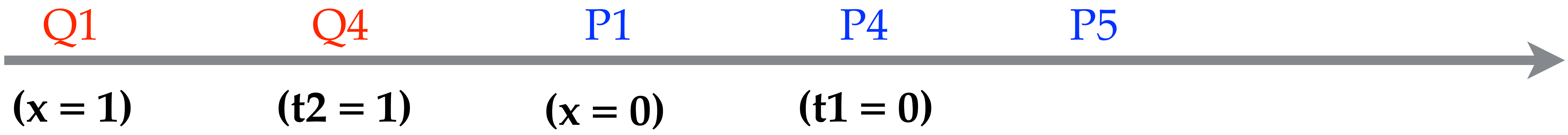
Add back, update variables, instructions

Respect the counter-example

add back

```
var: x, y, z, t1, t2

P1:  x = *                              Q1:  x = *



P4:  t1 = x                             Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```



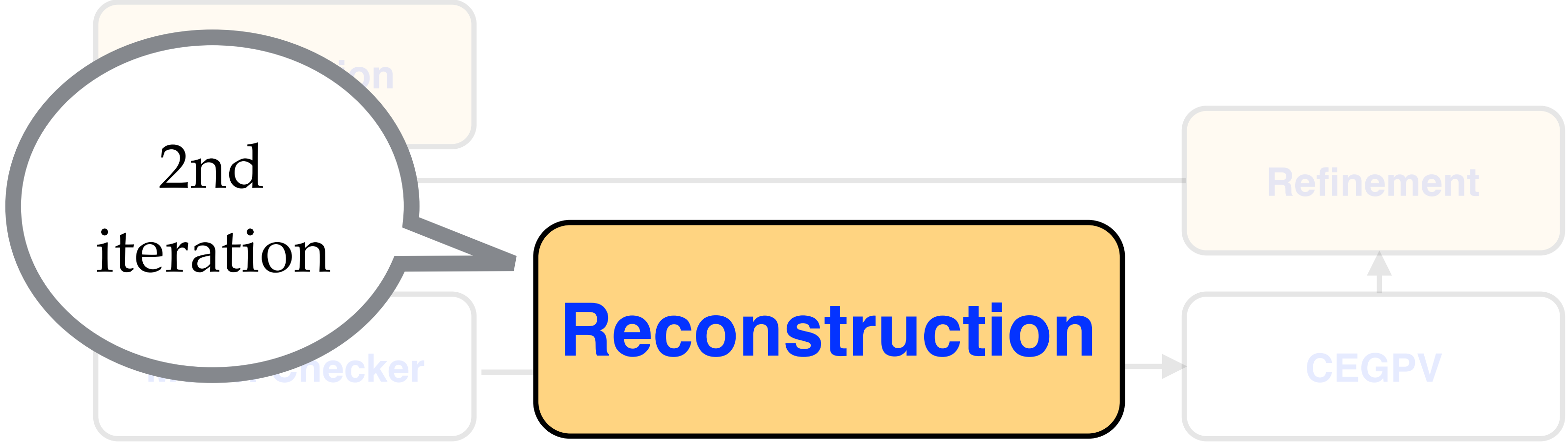


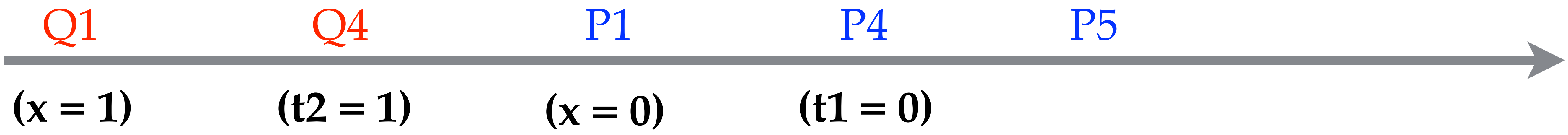
Add back, update variables, instructions

Respect the counter-example

add back

```
var: x, y, z, t1, t2

P1:  assume 0 == y?z?0:1:1 ✔Q1:  x = *


P4:  t1 = x                     Q4:  t2 = x
P5:  assert(t1 + t2 != 1)
```



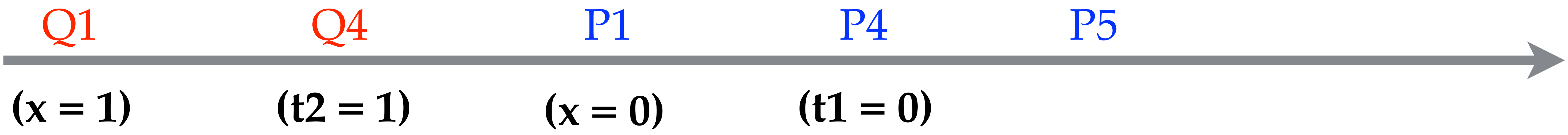


Add back, update variables, instructions

Respect the counter-example

```
var: x, y, z, t1, t2

P1:  assume 0 == y?z?0:1:1 ✓
P2:  y = z ✓
P3:  z = 0 ✓
P4:  assume 0 == 0  ✓
P5:  assert (0 + 1 != 1) ✓

Q1:  assume 1 == y?0:z?0:1  ✓
Q2:  y = !z ✓
Q3:  z = 1 ✓
Q4:  assume 1 == 1 ✓
```

2nd iteration
Reconstruction
Refinement
CEGPV
Checker



Q1 (x = 1)

Q4 (t2 = 1)

P1 (x = 0)

P4 (t1 = 0)

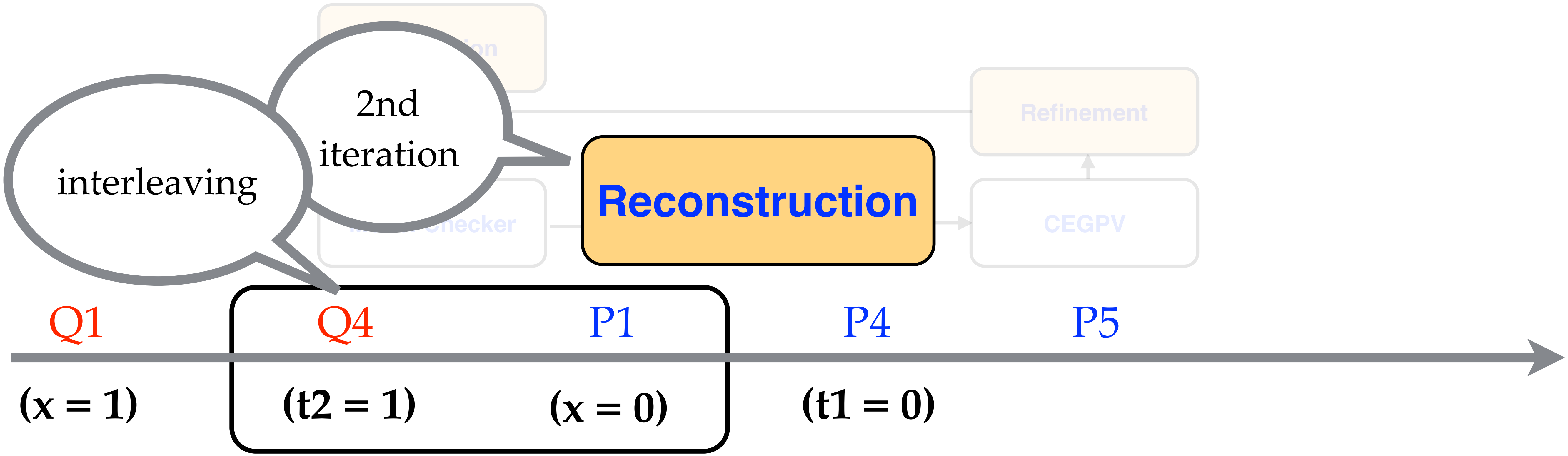
P5

Add back, update variables, instructions

Respect the counter-example

removed

```
var:   y, z,

P1:  assume 0 == y?z?0:1:1 ✓      Q1:  assume 1 == y?0:z?0:1 ✓
P2:  y = z ✓                      Q2:  y = !z ✓
P3:  z = 0 ✓                      Q3:  z = 1 ✓
P4:  assume 0 == 0  ✓             Q4:  assume 1 == 1 ✓
P5:  assert (0 + 1 != 1) ✓
```



2nd iteration

interleaving

Reconstruction

Refinement

Checker

CEGPV

Q1 (x = 1)

Q4 (t2 = 1)

P1 (x = 0)

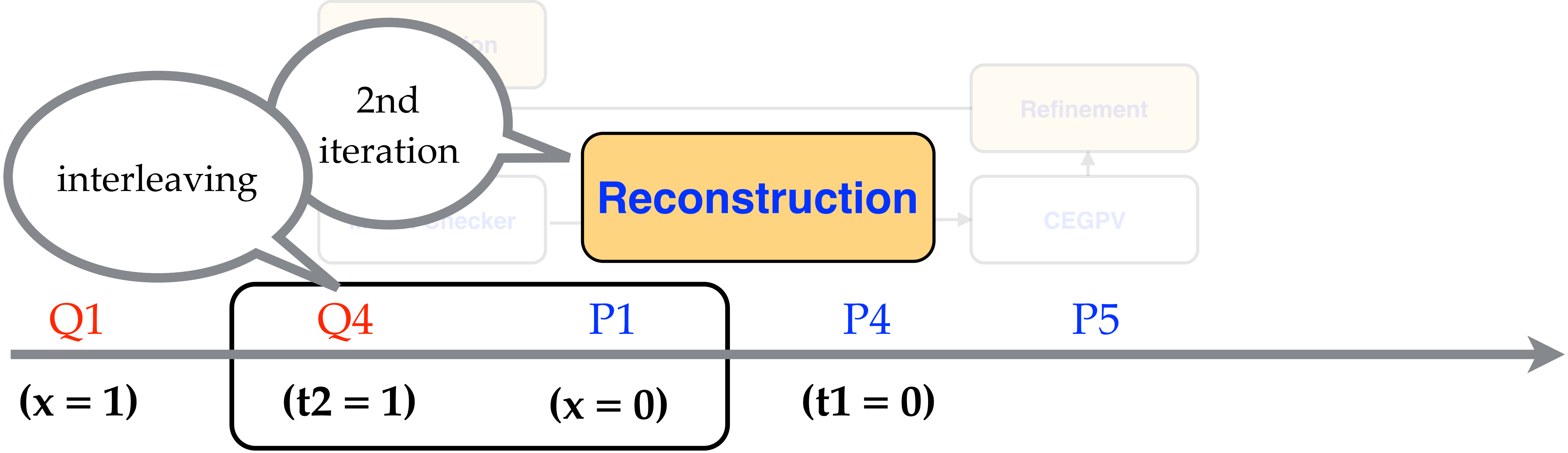
P4 (t1 = 0)

P5

Add back, update variables, instructions

Respect the counter-example

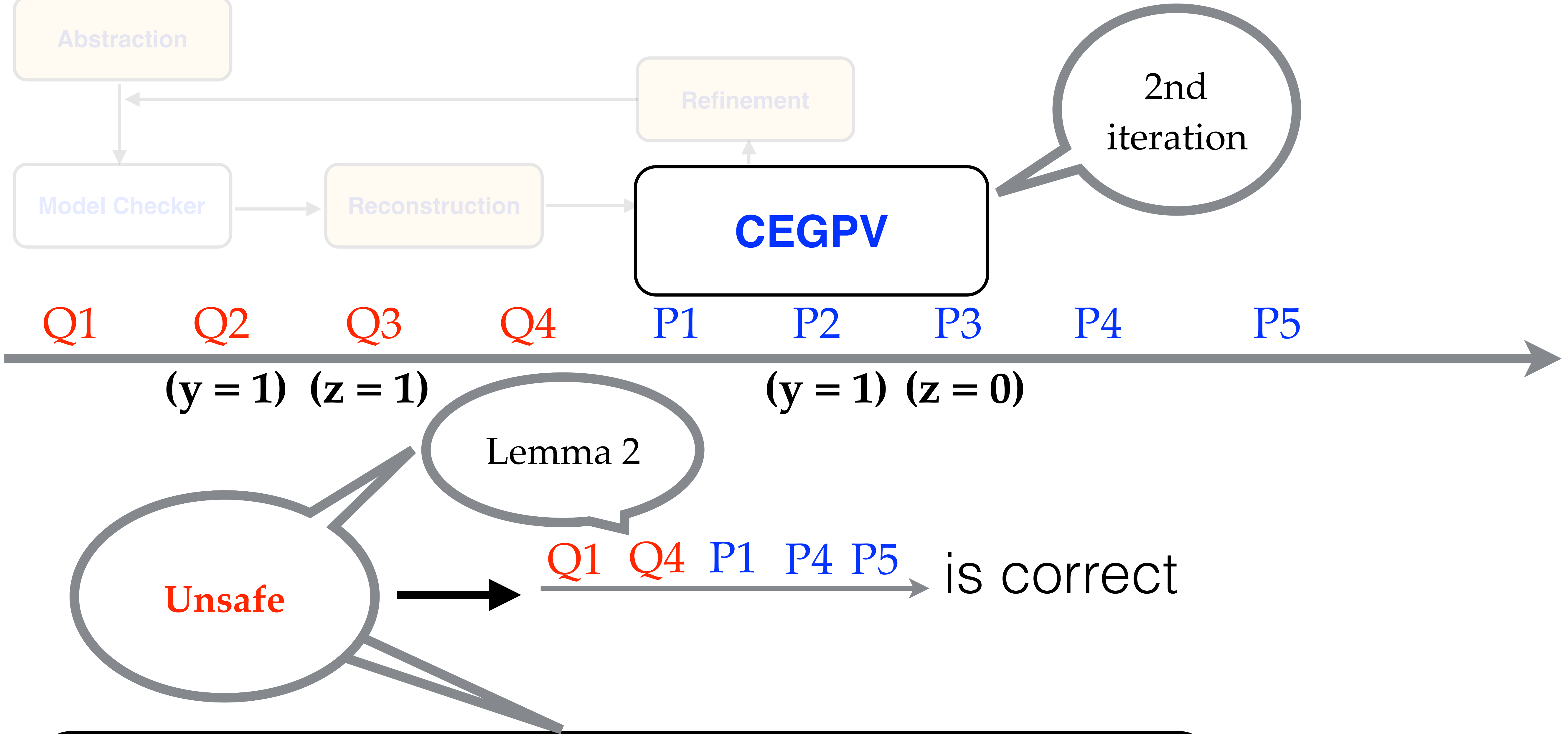
```
var: [redacted] y, z, [redacted]

P1:  assume 0 == y?z?0:1:1 ✔
P2:  y = z ✔
P3:  z = 0 ✔
P4:  assume 0 == 0 ✔
P5:  assert (0 + 1 != 1) ✔

Q1:  assume 1 == y?0:z?0:1 ✔
Q2:  y = !z ✔
Q3:  z = 1 ✔
Q4:  assume 1 == 1 ✔
```



Add back, update variables, instructions

Respect the counter-example

```
var:   y, z,

P1:  assume 0 == y?z?0:1:1 ✓      Q1:  assume 1 == y?0:z?0:1 ✓
P2:  y = z ✓                      Q2:  y = !z ✓
P3:  z = 0 ✓                      Q3:  z = 1 ✓
P4:  assume 0 == 0 ✓              Q4:  assume 1 == 1 ✓
P5:  assert (0 + 1 != 1) ✓
```



var: [redacted] y, z, [redacted]

P1: assume 0 == y?z?0:1:1 ✓

P2: y = z ✓

P3: z = 0 ✓

P4: assume 0 == 0 ✓

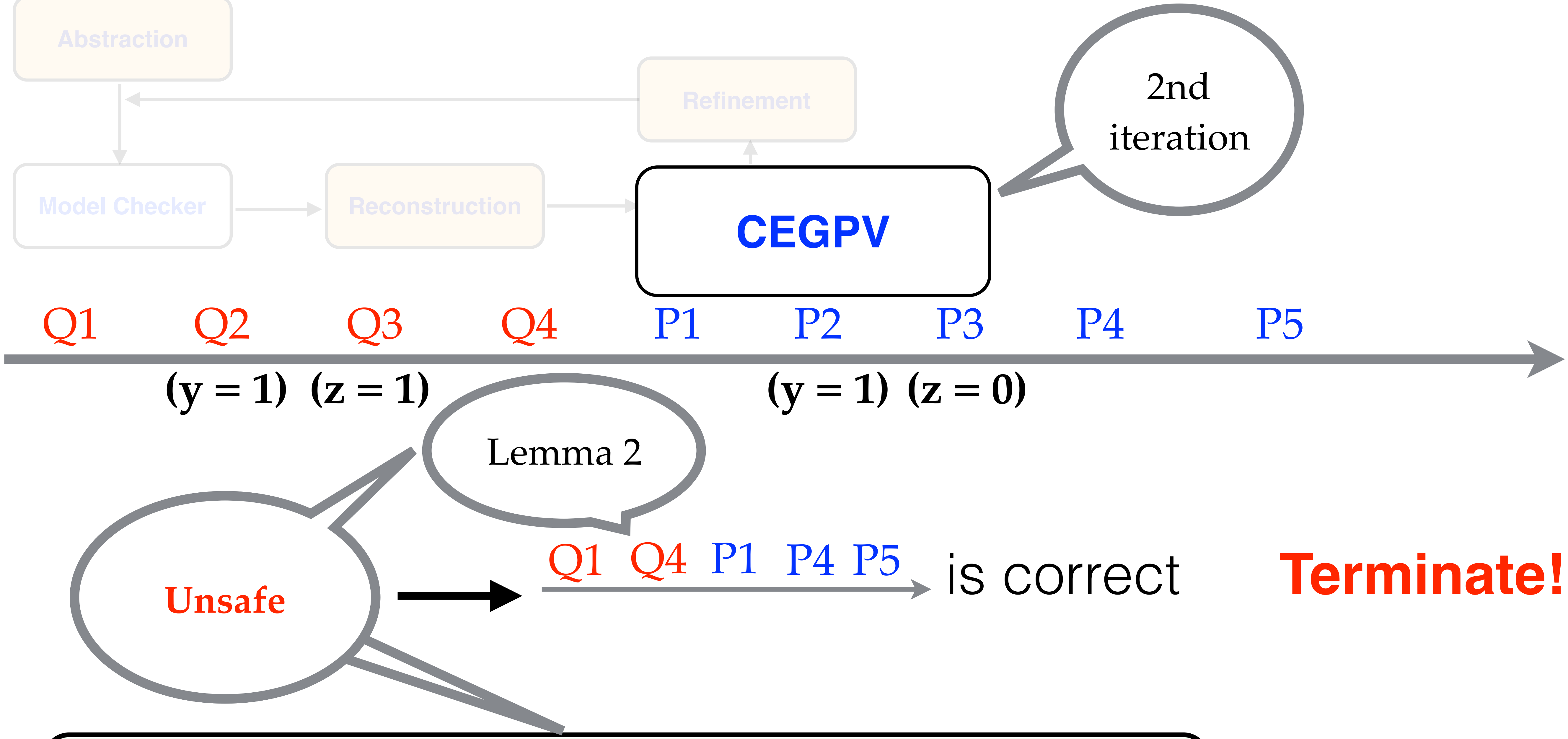
P5: assert (0 + 1 != 1) ✓

Q1: assume 1 == y?0:z?0:1 ✓

Q2: y = !z ✓

Q3: z = 1 ✓

Q4: assume 1 == 1 ✓



Q1 Q4 P1 P4 P5 is correct **Terminate!**

```
var: [redacted] y, z, [redacted]

P1:  assume 0 == y?z?0:1:1 ✔        Q1:  assume 1 == y?0:z?0:1 ✔
P2:  y = z ✔                        Q2:  y = !z ✔
P3:  z = 0 ✔                        Q3:  z = 1 ✔
P4:  assume 0 == 0  ✔               Q4:  assume 1 == 1 ✔
P5:  assert (0 + 1 != 1) ✔
```

Experiment results

Build on top of CBMC

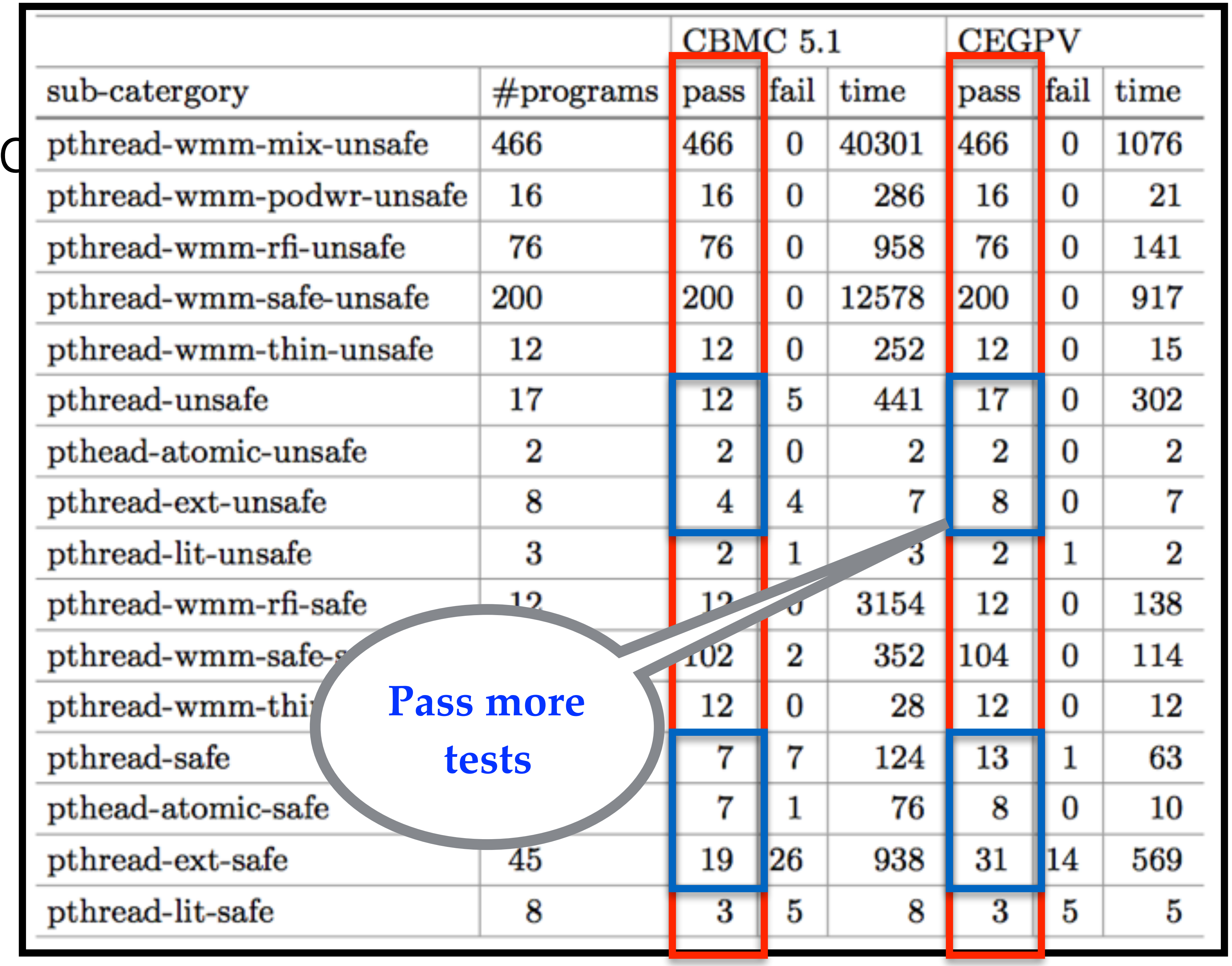
Run on SV-COMP15 benchmarks

> 1000 concurrent C programs

Experiment results

Build

Run



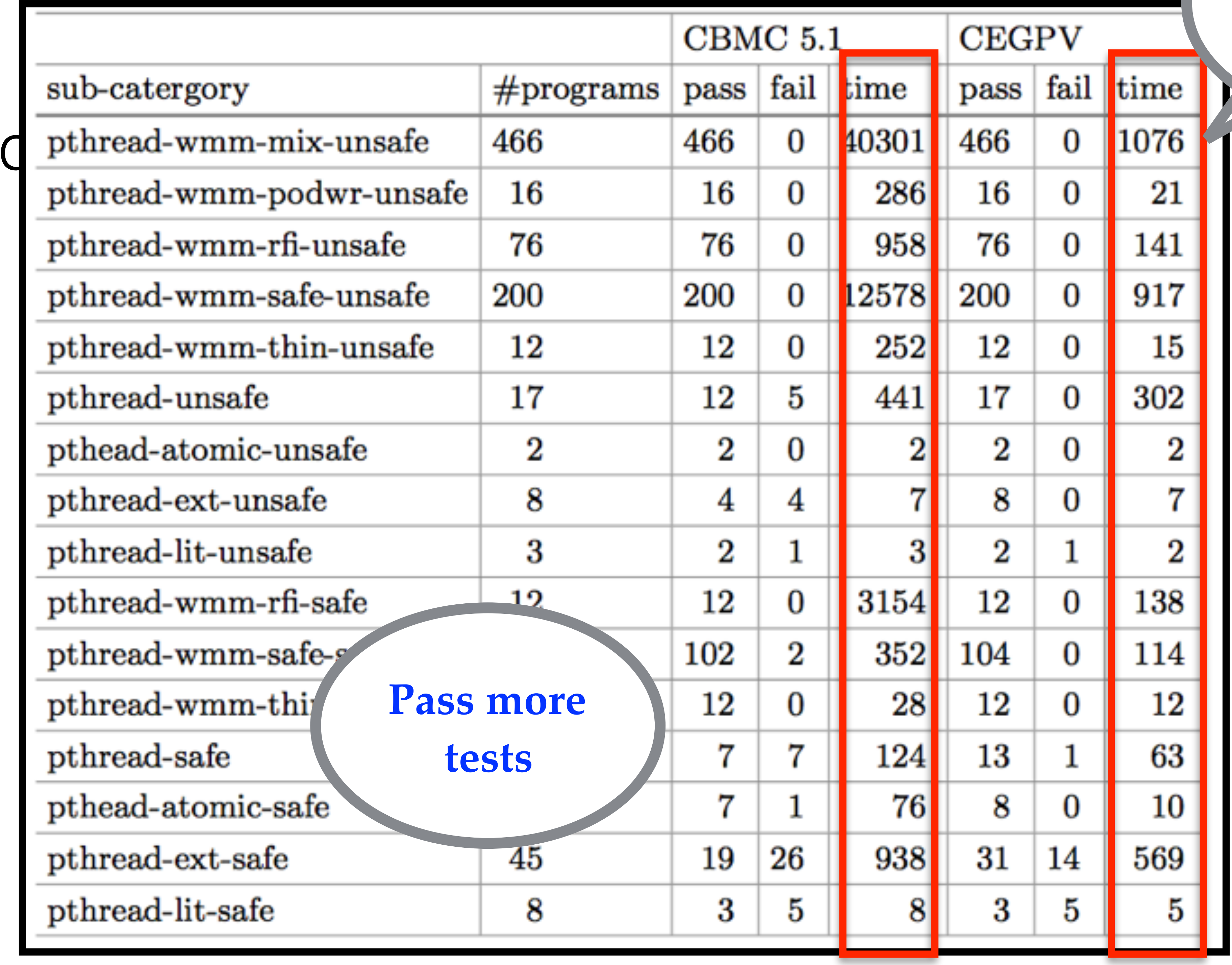
		CBMC 5.1			CEGPV		
sub-catergory	#programs	pass	fail	time	pass	fail	time
pthread-wmm-mix-unsafe	466	466	0	40301	466	0	1076
pthread-wmm-podwr-unsafe	16	16	0	286	16	0	21
pthread-wmm-rfi-unsafe	76	76	0	958	76	0	141
pthread-wmm-safe-unsafe	200	200	0	12578	200	0	917
pthread-wmm-thin-unsafe	12	12	0	252	12	0	15
pthread-unsafe	17	12	5	441	17	0	302
pthead-atomic-unsafe	2	2	0	2	2	0	2
pthread-ext-unsafe	8	4	4	7	8	0	7
pthread-lit-unsafe	3	2	1	3	2	1	2
pthread-wmm-rfi-safe	12	12	0	3154	12	0	138
pthread-wmm-safe-s	[illegible]	102	2	352	104	0	114
pthread-wmm-thi	[illegible]	12	0	28	12	0	12
pthread-safe	[illegible]	7	7	124	13	1	63
pthead-atomic-safe	[illegible]	7	1	76	8	0	10
pthread-ext-safe	45	19	26	938	31	14	569
pthread-lit-safe	8	3	5	8	3	5	5

Pass more tests

Experiment results

Build

Run



		CBMC 5.1			CEGPV		
sub-catergory	#programs	pass	fail	time	pass	fail	time
pthread-wmm-mix-unsafe	466	466	0	40301	466	0	1076
pthread-wmm-podwr-unsafe	16	16	0	286	16	0	21
pthread-wmm-rfi-unsafe	76	76	0	958	76	0	141
pthread-wmm-safe-unsafe	200	200	0	12578	200	0	917
pthread-wmm-thin-unsafe	12	12	0	252	12	0	15
pthread-unsafe	17	12	5	441	17	0	302
pthead-atomic-unsafe	2	2	0	2	2	0	2
pthread-ext-unsafe	8	4	4	7	8	0	7
pthread-lit-unsafe	3	2	1	3	2	1	2
pthread-wmm-rfi-safe	12	12	0	3154	12	0	138
pthread-wmm-safe-s	[illegible]	102	2	352	104	0	114
pthread-wmm-thi	[illegible]	12	0	28	12	0	12
pthread-safe	[illegible]	7	7	124	13	1	63
pthead-atomic-safe	[illegible]	7	1	76	8	0	10
pthread-ext-safe	45	19	26	938	31	14	569
pthread-lit-safe	8	3	5	8	3	5	5

Pass more tests

10x faster

Summary

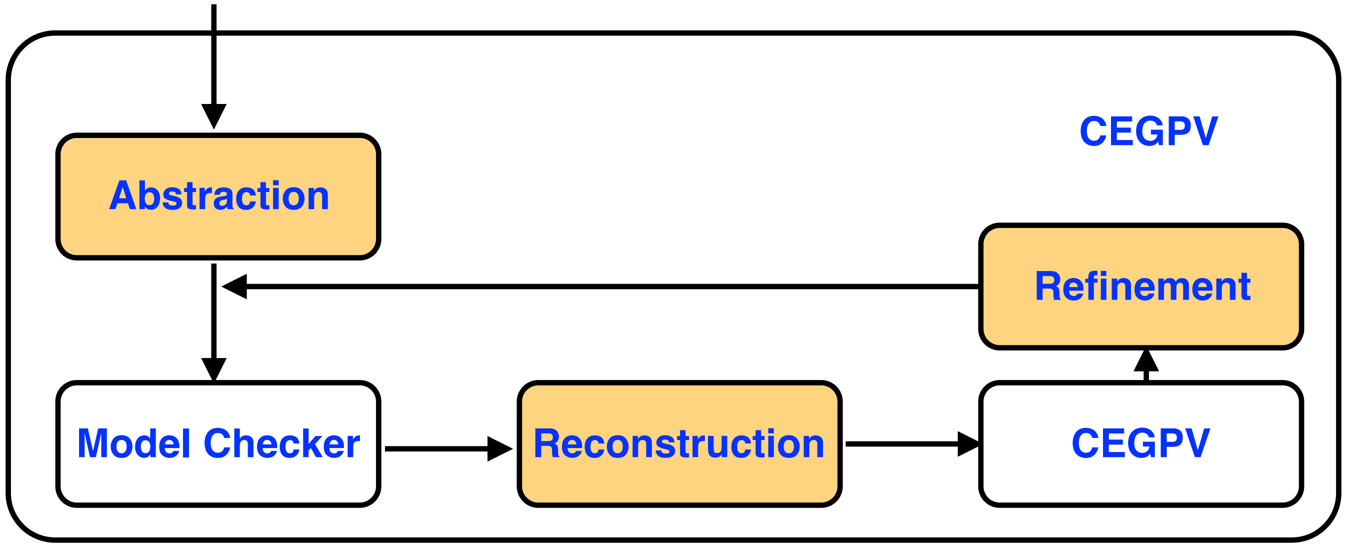
1. Deal with the state-space explosion problem

2. CEGAR extension for program verification

3. Code to code translation

   Can run on any back-end tools

4. Run on top of CBMC, **much** faster



Thank you!