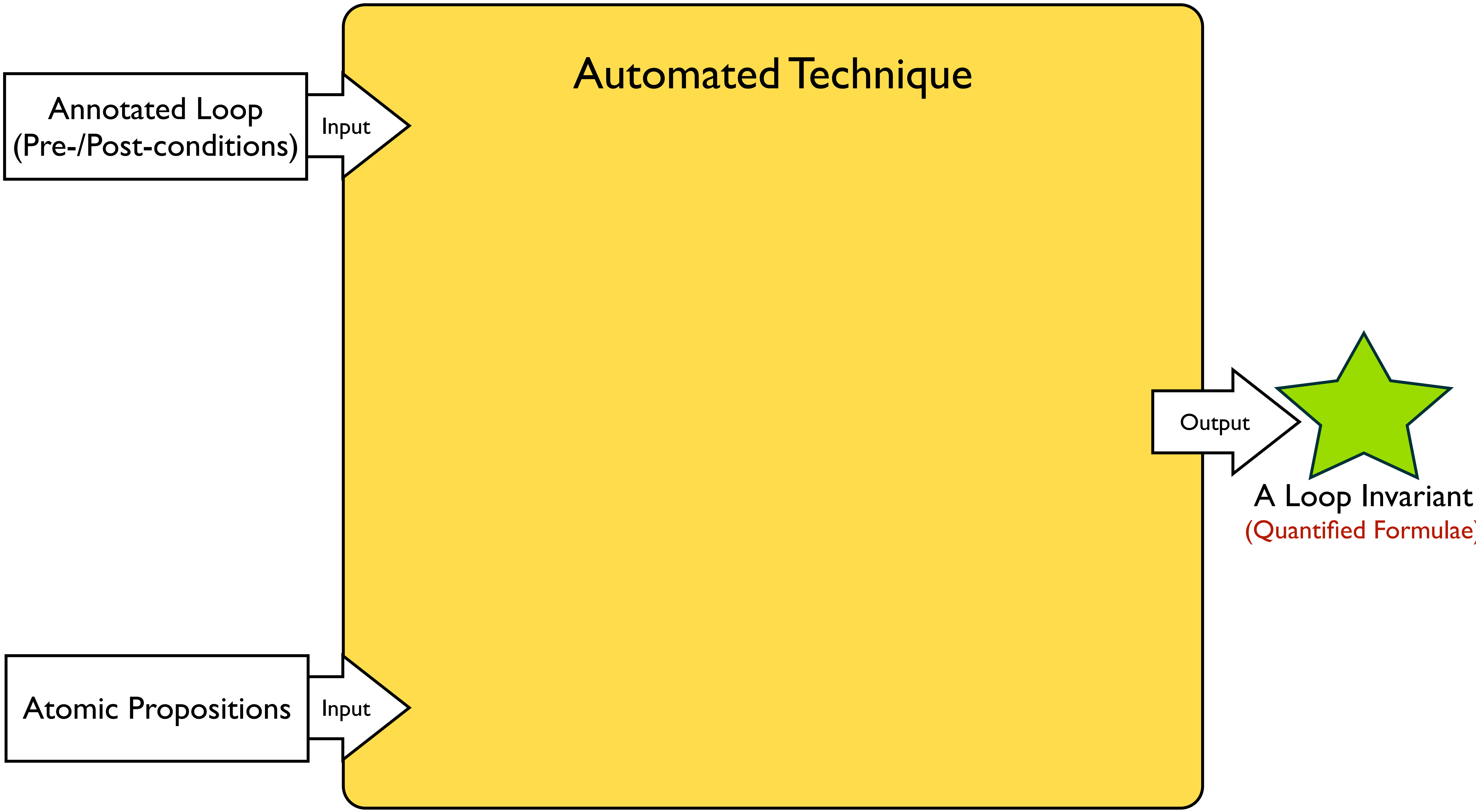
Automatically Inferring Quantified Loop Invariants by Algorithmic Learning from Simple Templates

Soonho Kong[1] Yungbum Jung[1] Cristina David[2]
Bow-Yaw Wang[3] Kwangkeun Yi[1]

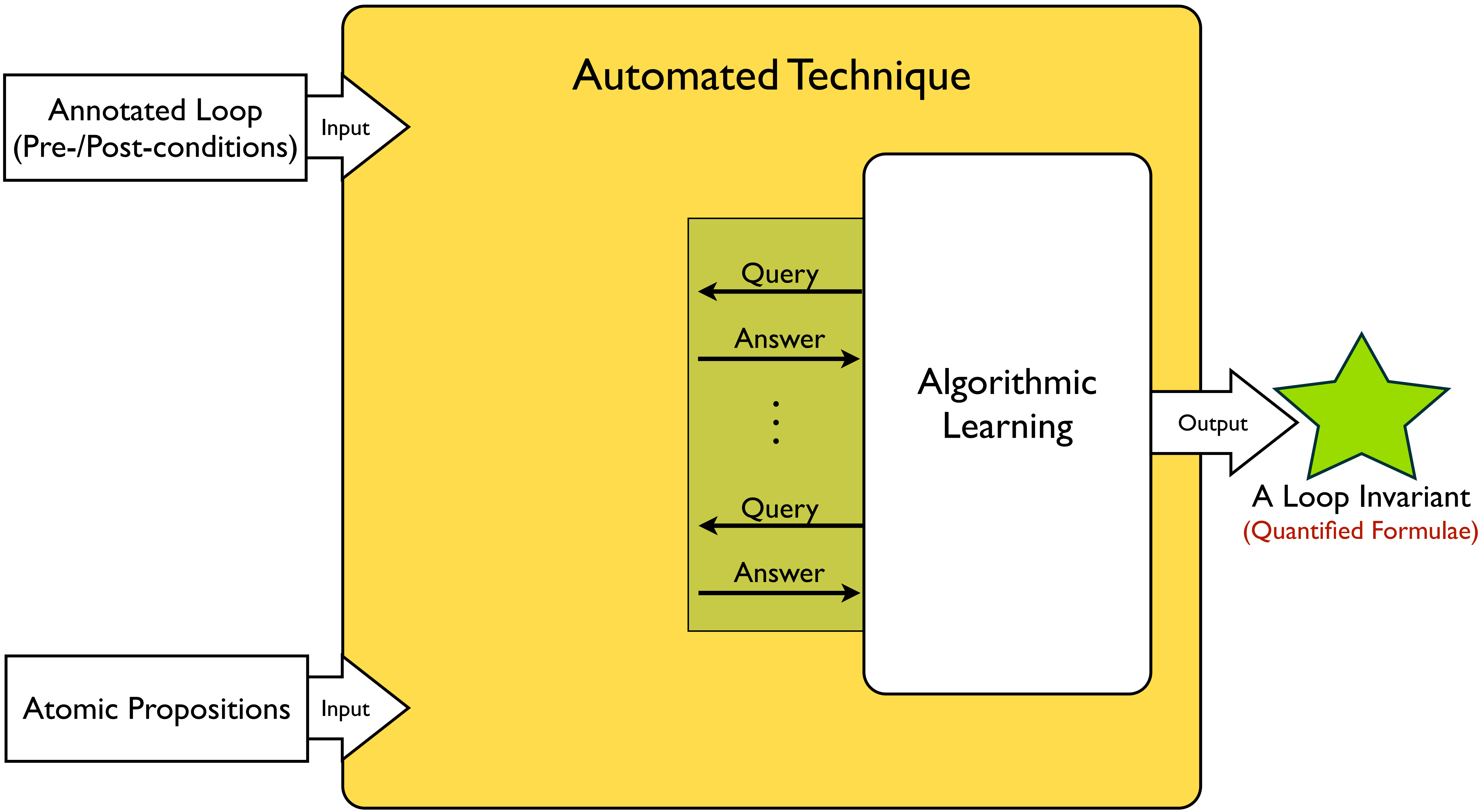
[1] Seoul National University
[2] National University of Singapore
[3] INRIA, Tsinghua University, and Academia Sinica

APLAS'10@Shanghai

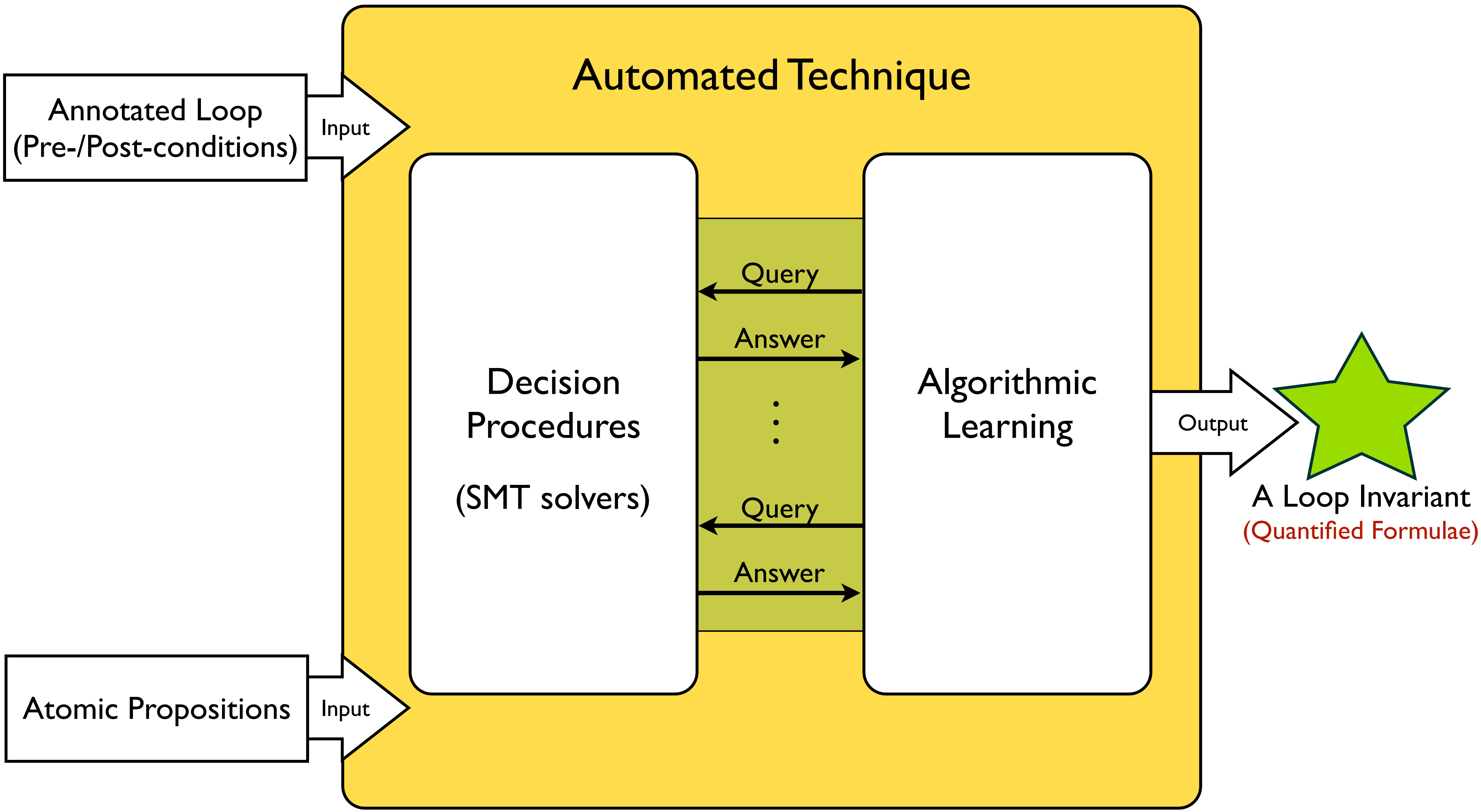
Overview


Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Atomic Propositions
Input
Output
A Loop Invariant
(Quantified Formulae)

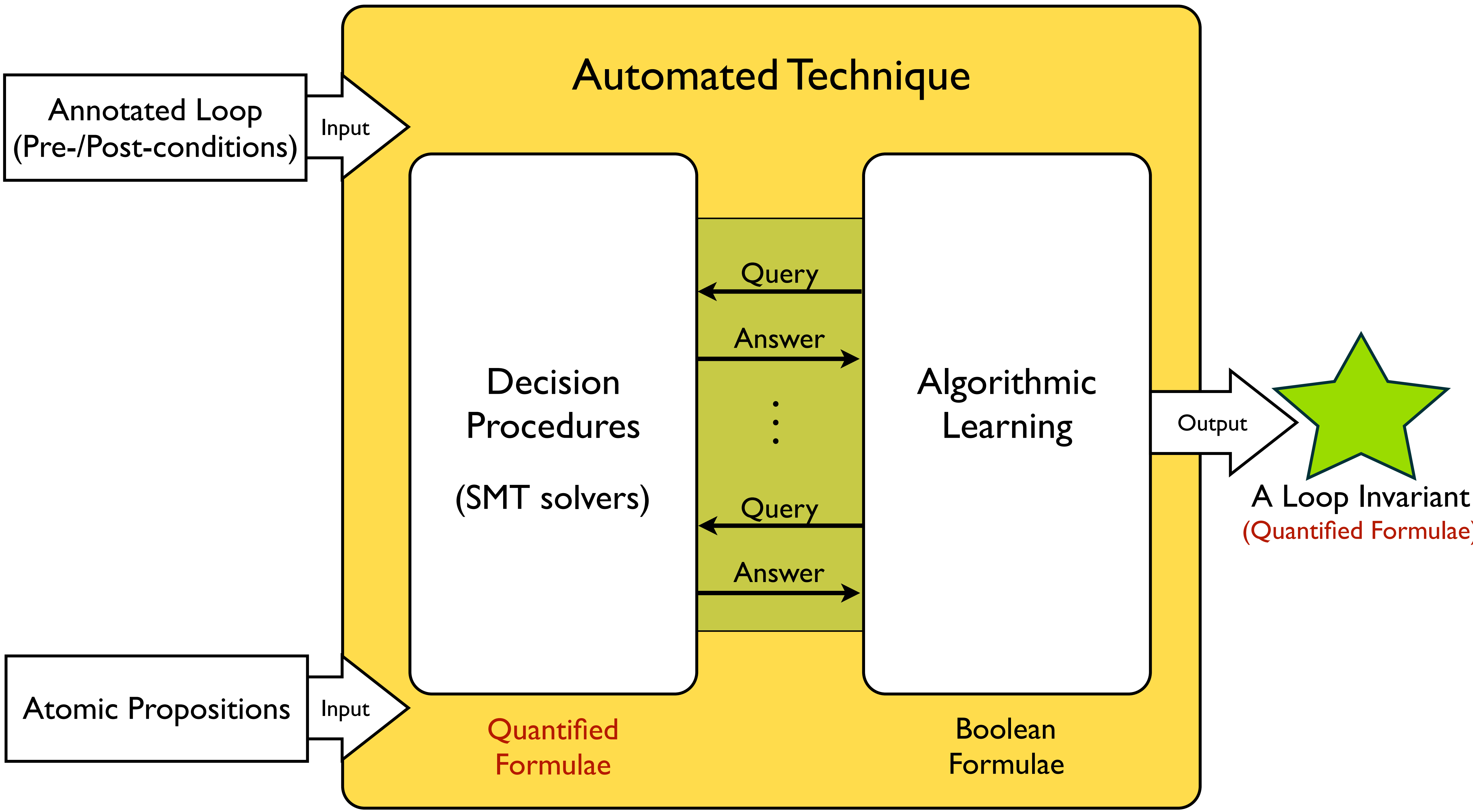
Overview


Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Atomic Propositions
Input
Query
Answer
Query
Answer
Algorithmic
Learning
Output
A Loop Invariant
(Quantified Formulae)

Overview

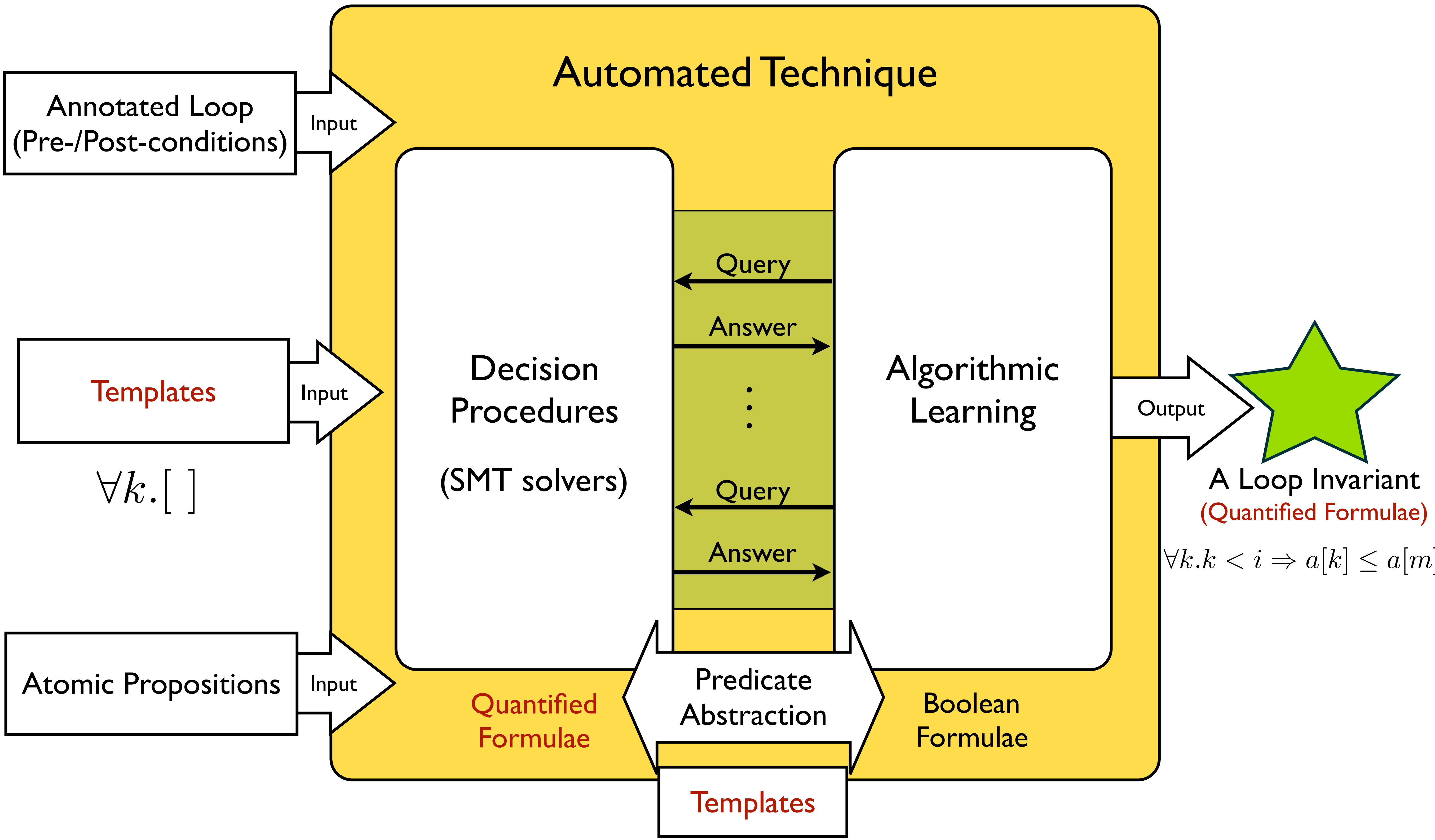

Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Atomic Propositions
Input
Decision
Procedures
(SMT solvers)
Query
Answer
Query
Answer
Algorithmic
Learning
Output
A Loop Invariant
(Quantified Formulae)

Overview



Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Atomic Propositions
Input
Decision
Procedures
(SMT solvers)
Query
Answer
⋮
Query
Answer
Algorithmic
Learning
Quantified
Formulae
Boolean
Formulae
Output
A Loop Invariant
(Quantified Formulae)

Overview



Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Templates
Input
$\forall k.[\ ]$
Atomic Propositions
Input
Decision
Procedures
(SMT solvers)
Query
Answer
Query
Answer
Algorithmic
Learning
Output
A Loop Invariant
(Quantified Formulae)
$\forall k. k < i \Rightarrow a[k] \leq a[m]$
Quantified
Formulae
Predicate
Abstraction
Boolean
Formulae
Templates

$$\{\ i = 0 \land key \neq 0 \land \neg ret \land \neg break\}$$

$\texttt{while}(i < n \ \land\ \neg break)\ \texttt{do}$
    $\texttt{if}(pkeys[i] = key)\ \texttt{then}$
        $pkeyrefs[i]{:=}pkeyrefs[i] - 1;$
        $\texttt{if}(pkeyrefs[i] = 0)\ \texttt{then}$
            $pkeys[i]{:=}0;\ ret{:=}true;$
        $break{:=}true;$
    $\texttt{else}\ i{:=}i + 1;$
$\texttt{done}$

$$\{(\neg ret \land \neg break) \Rightarrow (\forall k.k < n \Rightarrow pkeys[k] \neq key)$$
$$\land(\neg ret \land break) \Rightarrow (pkeys[i] = key \land pkeyrefs[i] \neq 0)$$
$$\land\ ret \Rightarrow (pkeyrefs[i] = 0 \land pkeys[i] = 0)\ \}$$

From Linux InfiniBand Driver

Templates $\forall k.[\ ]$

8 atomic propositions

$$
\begin{array}{l}
\{\ i = 0 \wedge key \neq 0 \wedge \neg ret \wedge \neg break\} \\
\ \ \texttt{while}(i < n\ \wedge\ \neg break)\ \texttt{do} \\
\quad\quad \texttt{if}(pkeys[i] = key)\ \texttt{then} \\
\quad\quad\quad\quad pkeyrefs[i]{:=}pkeyrefs[i] - 1; \\
\quad\quad\quad\quad \texttt{if}(pkeyrefs[i] = 0)\ \texttt{then} \\
\quad\quad\quad\quad\quad\quad pkeys[i]{:=}0;\ ret{:=}true; \\
\quad\quad\quad\quad break{:=}true; \\
\quad\quad \texttt{else}\ i{:=}i + 1; \\
\ \ \texttt{done} \\
\{(\neg ret \wedge \neg break) \Rightarrow (\forall k.k < n \Rightarrow pkeys[k] \neq key) \\
\quad \wedge(\neg ret \wedge break) \Rightarrow (pkeys[i] = key \wedge pkeyrefs[i] \neq 0) \\
\quad \wedge\ ret \Rightarrow (pkeyrefs[i] = 0 \wedge \ldots
\end{array}
$$

Find this invariant in 3 seconds

$$
(\forall k.(k < i) \Rightarrow pkeys[k] \neq key) \wedge (ret \Rightarrow pkeyrefs[i] = 0 \wedge pkeys[i] = 0) \\
\wedge\ (\neg ret \wedge break \Rightarrow pkeys[i] = key \wedge pkeyrefs[i] \neq 0)
$$

$\{\ i = 0 \wedge key \neq 0 \wedge \neg ret \wedge \neg break\}$

```
while(i < n  ∧  ¬break) do
   if(pkeys[i] = key) then
      pkeyrefs[i]:=pkeyrefs[i] − 1;
      if(pkeyrefs[i] = 0) then
         pkeys[i]:=0; ret:=true;
      break:=true;
   else i:=i + 1;
done
```

$\{(\neg ret \wedge \neg break) \Rightarrow (\forall k.k < n \Rightarrow pkeys[k] \neq key)$
$\quad \wedge(\neg ret \wedge break) \Rightarrow (pkeys[i] = key \wedge pkeyrefs[i] \neq 0)$
$\quad \wedge\ ret \Rightarrow (pkeyrefs[i] = 0 \wedge$

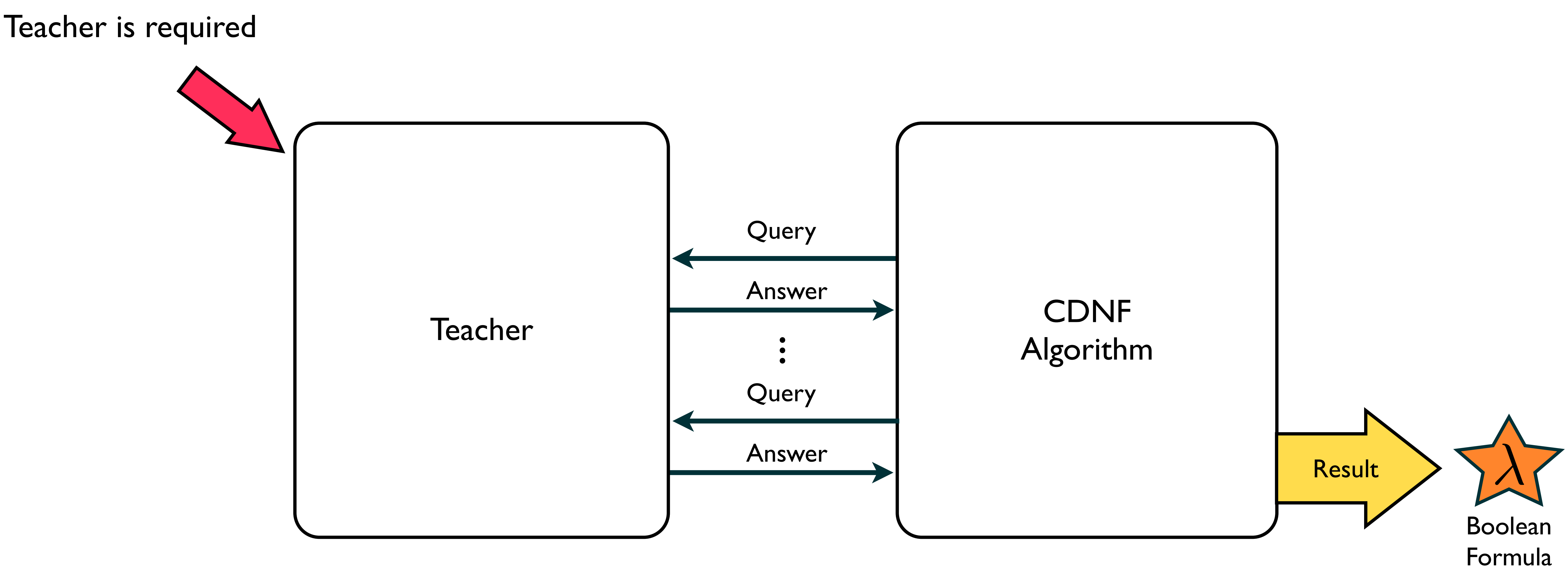
Find this invariant in 3 seconds

$$(\forall k.(k < i) \Rightarrow pkeys[k] \neq key) \wedge (ret \Rightarrow pkeyrefs[i] = 0 \wedge pkeys[i] = 0)$$
$$\wedge\ (\neg ret \wedge break \Rightarrow pkeys[i] = key \wedge pkeyrefs[i] \neq 0)$$

$$(\forall k.(\neg ret \vee \neg break \vee (pkeyrefs[i] = 0 \wedge pkeys[i] = 0)) \wedge (pkeys[k] \neq key \vee k \geq i)$$
$$\wedge(\neg ret \vee (pkeyrefs[i] = 0 \wedge pkeys[i] = 0 \wedge i < n \wedge break))$$
$$\wedge\ (\neg break \vee pkeyrefs[i] \neq 0 \vee ret) \wedge (\neg break \vee pkeys[i] = key \vee ret))$$

Algorithmic Learning: CDNF Algorithm

CDNF Algorithm[†]



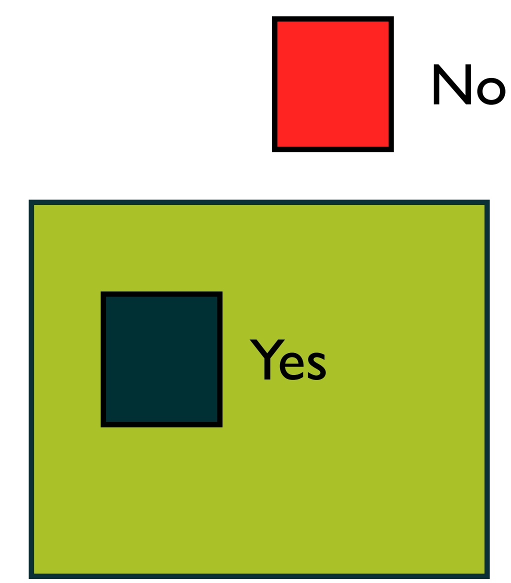
Actively learning a Boolean formula
from membership and equivalence queries

† Bshouty, N.H.: Exact learning boolean functions via the monotone theory. Information and Computation **123** (1995) 146–153

Membership Query

Membership Query $MEM(\mu)$ asks whether Boolean assignment μ satisfies the Boolean formula λ

$$MEM(\mu) = Yes \quad \text{if } \mu \models \lambda$$
$$MEM(\mu) = No \quad \text{if } \mu \not\models \lambda$$



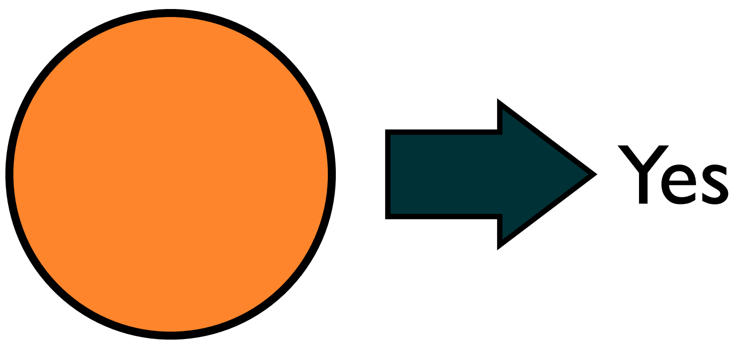
Example: $\lambda =$ XOR function $b_1 \oplus b_2$

$$MEM(\{b_1 = T, b_2 = F\}) = Yes \qquad \because T \oplus F = T$$

$$MEM(\{b_1 = T, b_2 = T\}) = No \qquad \because T \oplus T = F$$

Equivalence Query

Equivalence Query $EQ(\beta)$ asks whether the guessed Boolean formula β is equivalent to λ.



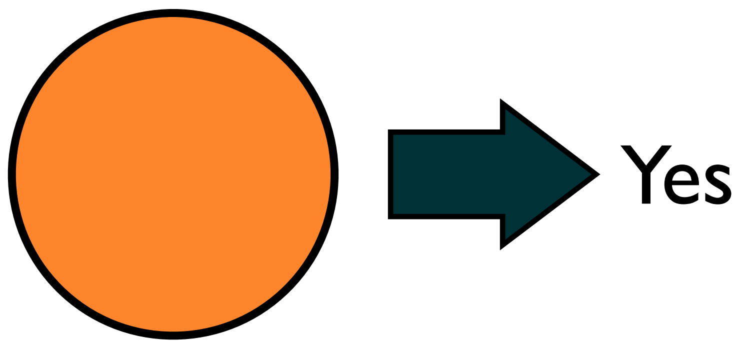
if $\beta \equiv \lambda$

Example: $\lambda =$ XOR function $b_1 \oplus b_2$

$$EQ((b_1 \wedge \neg b_2) \vee (\neg b_1 \wedge b_2)) = Yes$$

Equivalence Query

Equivalence Query $EQ(\beta)$ asks whether the guessed Boolean formula β is equivalent to λ.

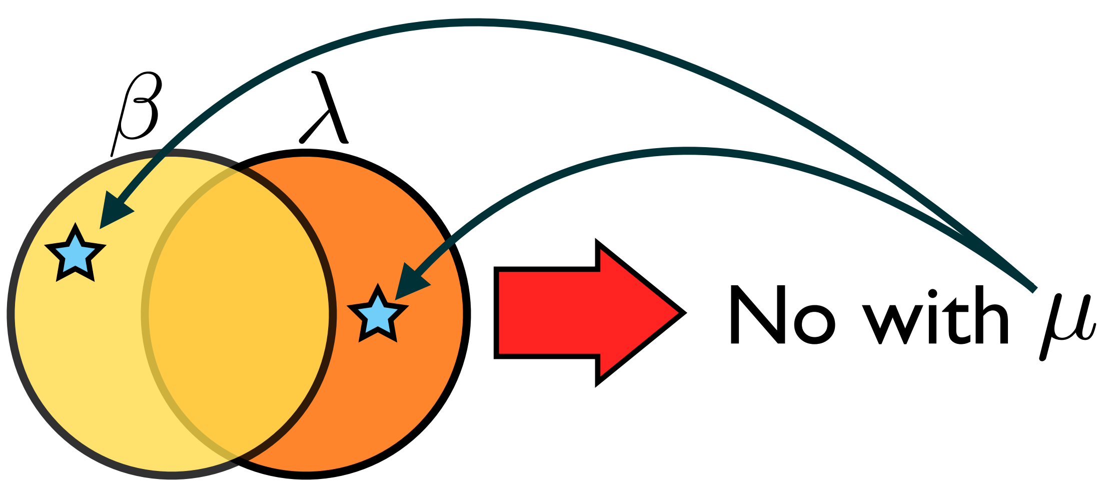


if $\beta \equiv \lambda$

Example: $\lambda =$ XOR function $b_1 \oplus b_2$

$$EQ((b_1 \wedge \neg b_2) \vee (\neg b_1 \wedge b_2)) = Yes$$

Otherwise, the teacher needs to provide a truth assignment as a counterexample μ.



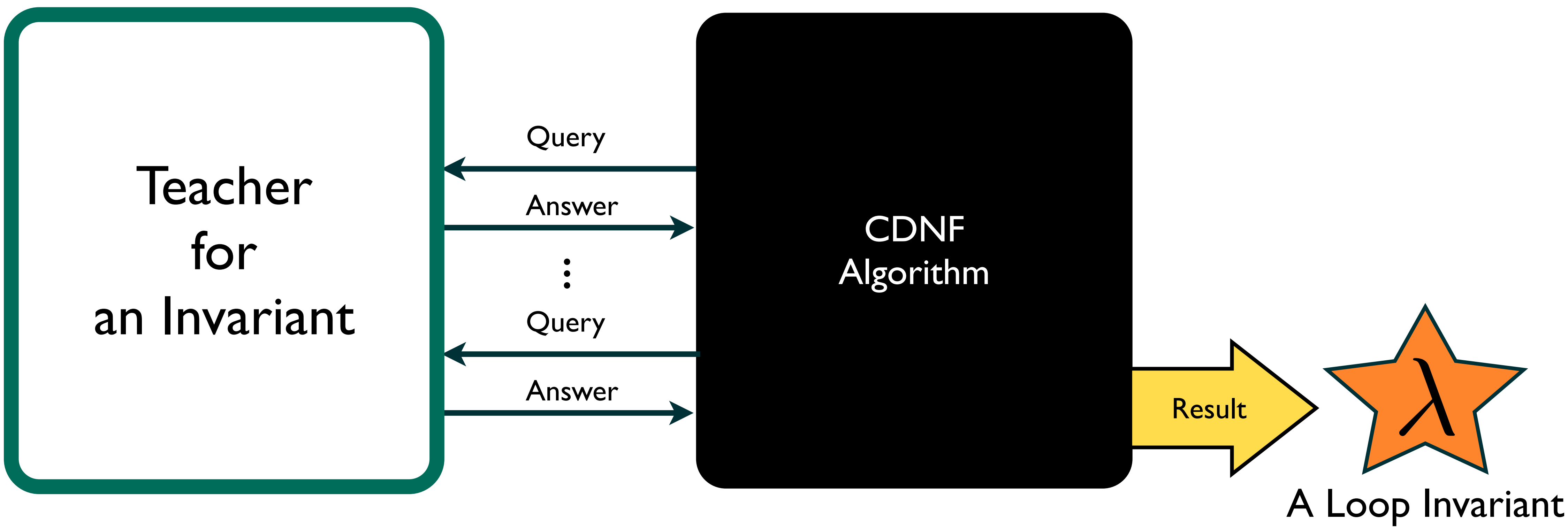
$$\beta \not\equiv \lambda \wedge (\mu \models \beta \oplus \lambda)$$

Example: $\lambda =$ XOR function $b_1 \oplus b_2$

$$EQ(b_1 \vee b_2) = No \text{ with } \{b_1 = T,\ b_2 = T\}$$

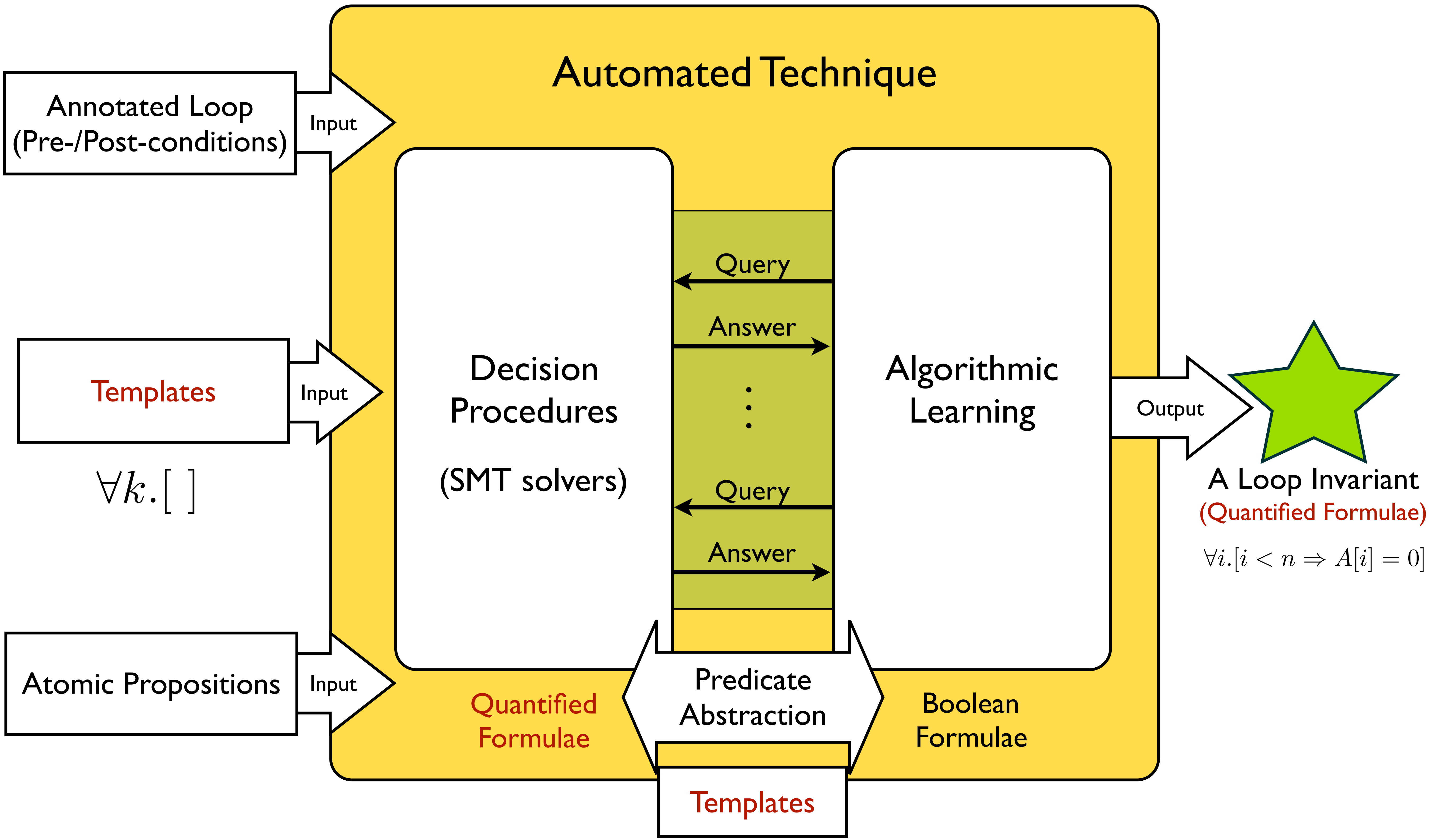
$$\because T \oplus T = F$$
$$T \vee T = T$$

Goal



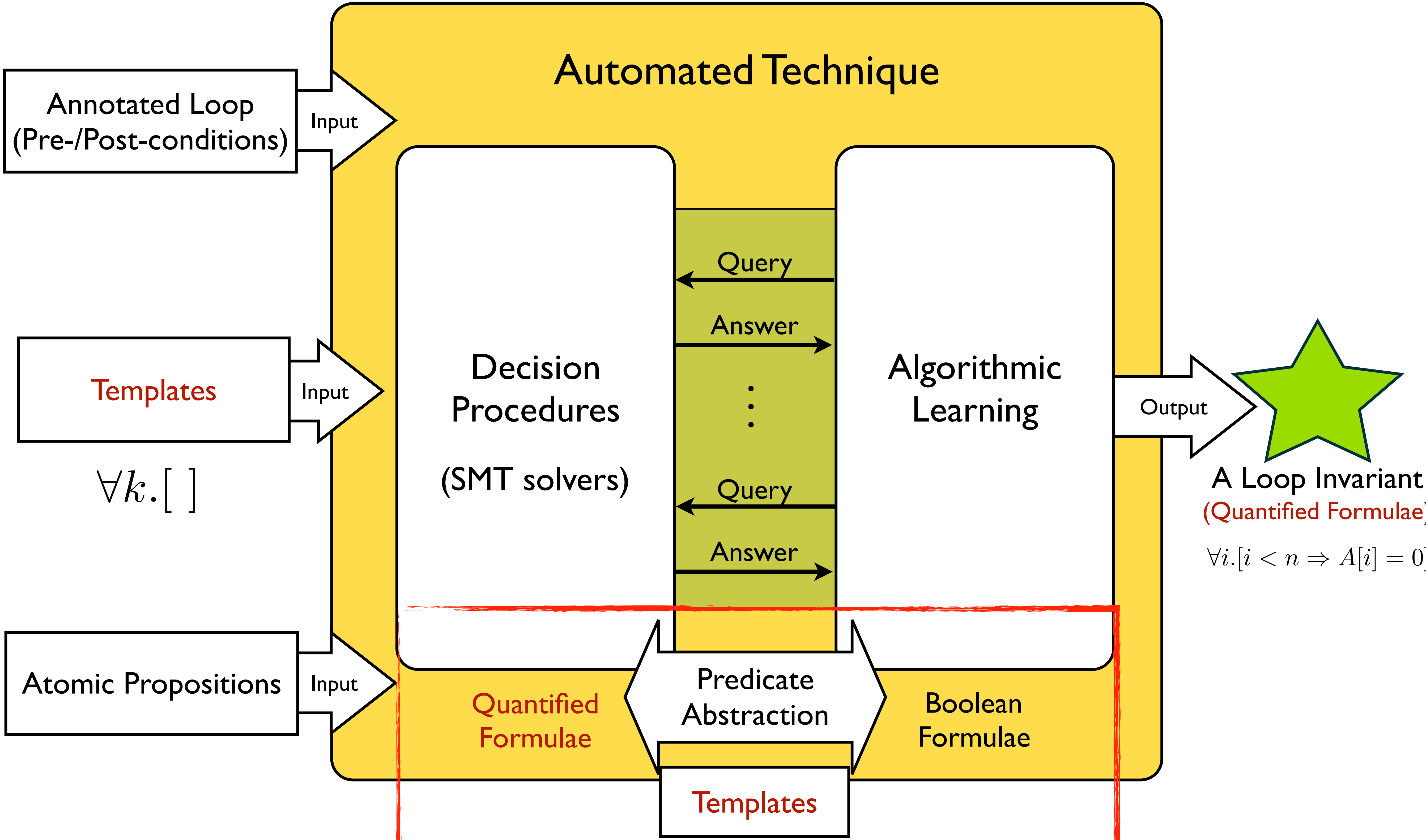
Implementing a Teacher for guiding CDNF algorithm to find a quantified invariant

First Issue



Annotated Loop
(Pre-/Post-conditions)
Input
Automated Technique
Templates
Input
$\forall k.[\;]$
Decision
Procedures
(SMT solvers)
Query
Answer
Query
Answer
Algorithmic
Learning
Output
A Loop Invariant
(Quantified Formulae)
$\forall i.[i < n \Rightarrow A[i] = 0]$
Atomic Propositions
Input
Quantified
Formulae
Predicate
Abstraction
Boolean
Formulae
Templates

First Issue



Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Templates
Input
$\forall k.[\ ]$
Atomic Propositions
Input
Decision
Procedures
(SMT solvers)
Query
Answer
⋮
Query
Answer
Algorithmic
Learning
Output
A Loop Invariant
(Quantified Formulae)
$\forall i.[i < n \Rightarrow A[i] = 0]$
Quantified
Formulae
Predicate
Abstraction
Boolean
Formulae
Templates

Relating Domains

Problem:

We want to find a Quantified invariant
while the CDNF algorithm finds a Boolean formula.

APLAS'10

Templates

$\forall k.[\ ]$

VMCAI'10

Predicate Abstraction

Quantified Formula | Propositional Formula | Boolean Formula

$\forall k.k < i \Rightarrow a[k] \leq a[m]$

$k < i \Rightarrow a[k] \leq a[m]$

$\neg b_{k<i} \vee b_{a[k] \leq a[m]}$

Answering Queries

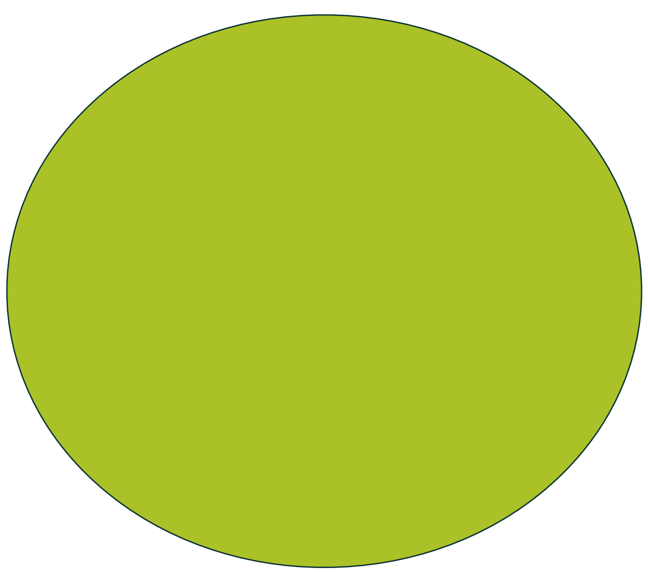
Quantified Formula

$$\forall k. k < i \Rightarrow a[k] \leq a[m]$$

Boolean Formula

$$\neg b_{k<i} \vee b_{a[k] \leq a[m]}$$

Teacher



CDNF Algorithm

Answering Queries

Quantified Formula

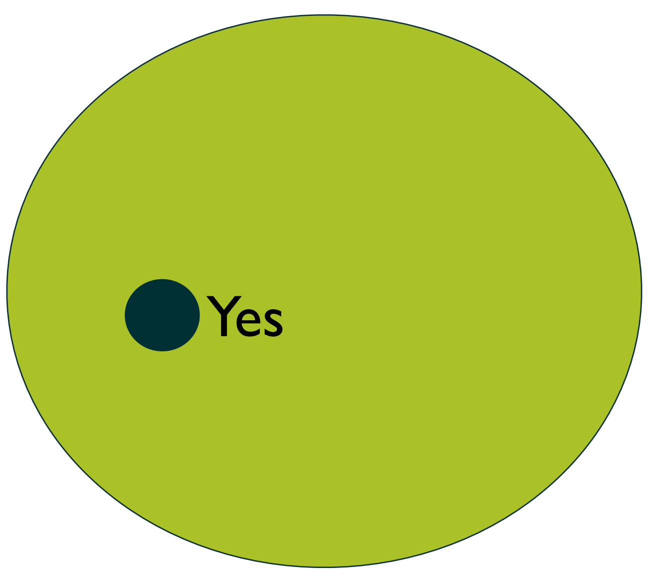
$$\forall k.k < i \Rightarrow a[k] \leq a[m]$$

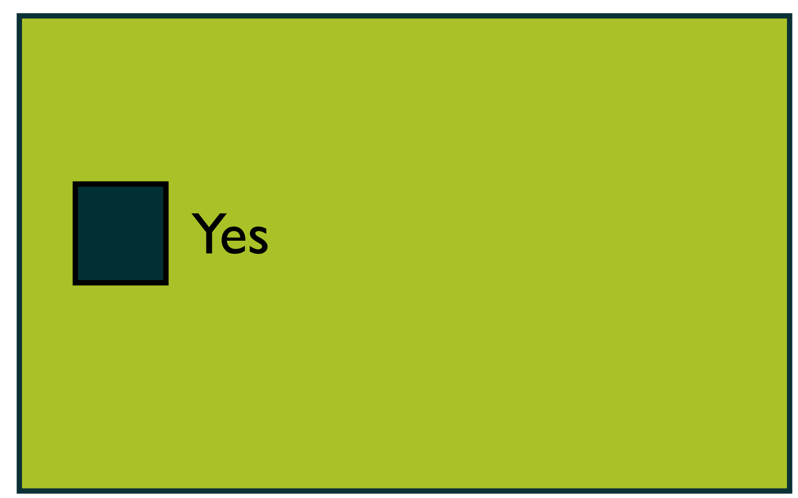
Boolean Formula

$$\neg b_{k<i} \vee b_{a[k]\leq a[m]}$$

Teacher

CDNF Algorithm





If teacher says "Yes" then it should really mean "Yes"

Answering Queries

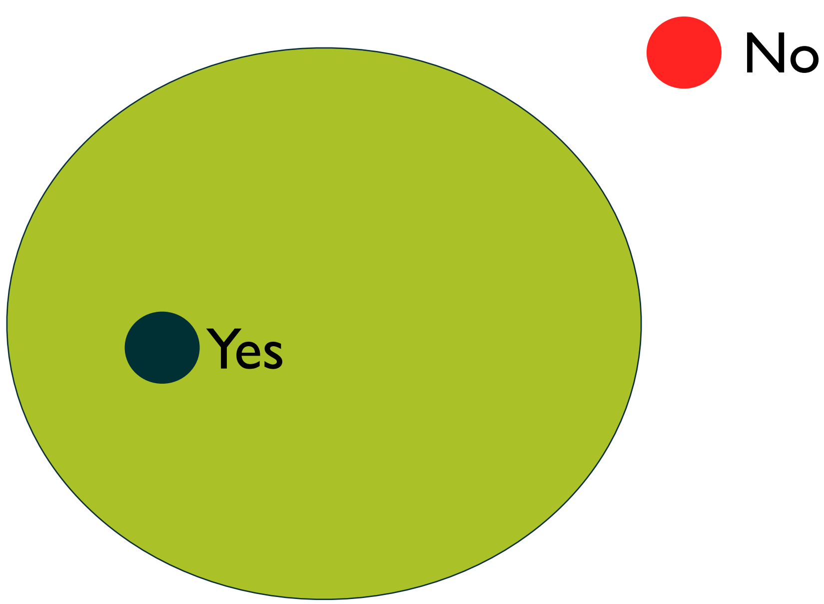
Quantified Formula

$$\forall k. k < i \Rightarrow a[k] \leq a[m]$$

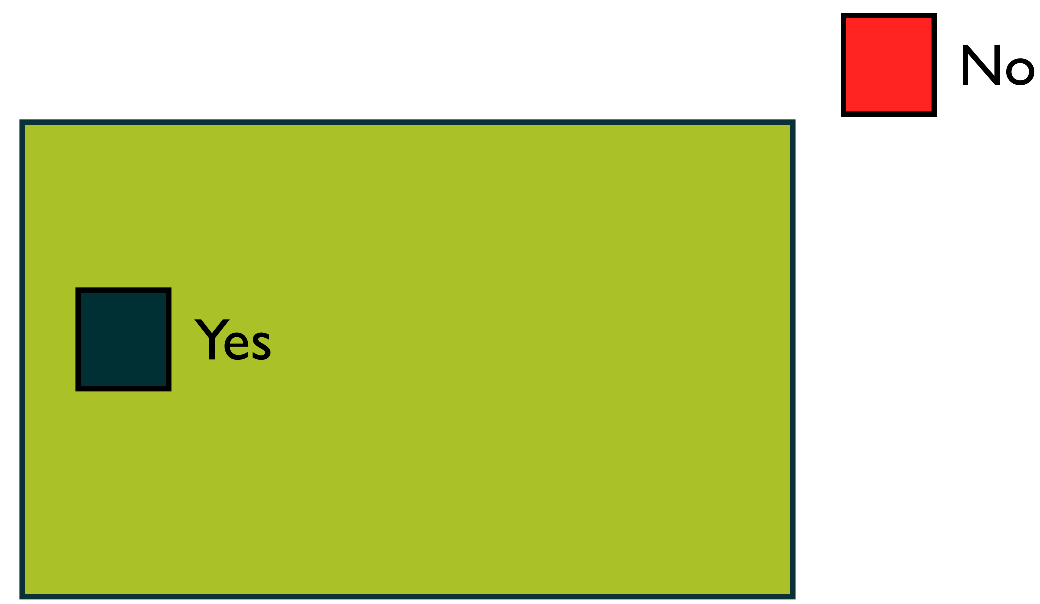
Boolean Formula

$$\neg b_{k<i} \vee b_{a[k] \leq a[m]}$$

Teacher



CDNF Algorithm



If teacher says "Yes" then it should really mean "Yes"

If teacher says "No" then it should really mean "No"

Answering Queries

If teacher says "No" then it should really mean "No"

$$\texttt{if}\ \ t[\theta_1] \not\Rightarrow t[\theta_2]\ \ \texttt{then}\ \ \theta_1 \not\Rightarrow \theta_2$$

If teacher says "Yes" then it should really mean "Yes"

$$\texttt{if}\ \ t[\theta_1] \Rightarrow t[\theta_2]\ \ \texttt{then}\ \ \theta_1 \Rightarrow \theta_2$$

Answering Queries

If teacher says "No" then it should really mean "No"

$$\texttt{if}\ \ t[\theta_1] \not\Rightarrow t[\theta_2]\ \ \texttt{then}\ \ \theta_1 \not\Rightarrow \theta_2$$

If teacher says "Yes" then it should really mean "Yes"

$$\texttt{if}\ \ t[\theta_1] \Rightarrow t[\theta_2]\ \ \texttt{then}\ \ \theta_1 \Rightarrow \theta_2$$

Answering Queries

If teacher says "No" then it should really mean "No"

$$\texttt{if}\ \ t[\theta_1] \not\Rightarrow t[\theta_2]\ \ \texttt{then}\ \ \theta_1 \not\Rightarrow \theta_2$$

If teacher says "Yes" then it should really mean "Yes"

$$\texttt{if}\ \ t[\theta_1] \Rightarrow t[\theta_2]\ \ \texttt{then}\ \ \theta_1 \Rightarrow \theta_2$$

A Counter Example

$$\texttt{if}\ \ \forall i.i < 10 \Rightarrow \forall i.i < 1\ \ \texttt{then}\ \ i < 10 \Rightarrow i < 1$$

Answering Queries

If teacher says "No" then it should really mean "No"

$$\checkmark \quad \texttt{if} \;\; t[\theta_1] \not\Rightarrow t[\theta_2] \;\; \texttt{then} \;\; \theta_1 \not\Rightarrow \theta_2$$

If teacher says "Yes" then it should really mean "Yes"

$$\texttt{if} \;\; t[\theta_1] \Rightarrow t[\theta_2] \;\; \texttt{then} \;\; \theta_1 \Rightarrow \theta_2$$

A Counter Example

$$\texttt{if} \;\; \boxed{\forall i.i < 10 \Rightarrow \forall i.i < 1} \;\; \texttt{then} \;\; i < 10 \Rightarrow i < 1$$

Answering Queries

If teacher says "No" then it should really mean "No"

✓ $\texttt{if} \;\; t[\theta_1] \not\Rightarrow t[\theta_2] \;\; \texttt{then} \;\; \theta_1 \not\Rightarrow \theta_2$

If teacher says "Yes" then it should really mean "Yes"

$\texttt{if} \;\; t[\theta_1] \Rightarrow t[\theta_2] \;\; \texttt{then} \;\; \theta_1 \Rightarrow \theta_2$

A Counter Example

$\texttt{if} \;\; \boxed{\forall i.i < 10 \Rightarrow \forall i.i < 1} \;\; \texttt{then} \;\; \boxed{i < 10 \Rightarrow i < 1}$

Answering Queries

If teacher says "No" then it should really mean "No"

✓ $\texttt{if} \;\; t[\theta_1] \not\Rightarrow t[\theta_2] \;\; \texttt{then} \;\; \theta_1 \not\Rightarrow \theta_2$

If teacher says "Yes" then it should really mean "Yes"

✓ $\texttt{if} \;\; t[\theta_1] \Rightarrow t[\theta_2] \;\; \texttt{then} \;\; \theta_1 \Rightarrow \theta_2$

A Counter Example

$$\texttt{if} \;\; \boxed{\forall i. i < 10 \Rightarrow \forall i. i < 1} \;\; \texttt{then} \;\; \boxed{i < 10 \Rightarrow i < 1}$$

Well-formedness condition

Second Issue

Problem:

The teacher is asked to answer questions about invariants without knowing invariants.

Second Issue

Problem:

The teacher is asked to answer questions about invariants without knowing invariants.

Solution:

We use approximations and random answers

Invariant Properties

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \kappa\ \texttt{do}\ S\ \{\epsilon\}$$

An Invariant I must satisfy all the following conditions:

(A) $\delta \Rightarrow I$ (I holds when entering the loop)

(B) $I \wedge \kappa \Rightarrow Pre(I, S)$ (I holds at each iteration)

(C) $I \wedge \neg\kappa \Rightarrow \epsilon$ (I gives ϵ after leaving the loop)

Invariant Properties

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \kappa\ \texttt{do}\ S\ \{\epsilon\}$$

An Invariant I must satisfy all the following conditions:

(A) $\delta \Rightarrow I$ (I holds when entering the loop)

(B) $I \wedge \kappa \Rightarrow Pre(I, S)$ (I holds at each iteration)

(C) $I \wedge \neg\kappa \Rightarrow \epsilon$ (I gives ϵ after leaving the loop)

Observation #1
In equivalence query we can say "YES" by checking these conditions.

Invariant Properties

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \kappa\ \texttt{do}\ S\ \{\epsilon\}$$

An Invariant I must satisfy all the following conditions:

(A) $\delta \Rightarrow I$ (I holds when entering the loop)

(B) $I \wedge \kappa \Rightarrow Pre(I, S)$ (I holds at each iteration)

(C) $I \wedge \neg\kappa \Rightarrow \epsilon$ (I gives ϵ after leaving the loop)

Observation #1
In equivalence query we can say "YES" by checking these conditions.

Observation #2

$$\delta \Rightarrow I \Rightarrow \kappa \vee \epsilon$$

strongest under-approximation of an invariant

weakest over-approximation of an invariant

Equivalence Query Resolution

1. "YES", if the guess satisfies invariant conditions.

(A) $\delta \Rightarrow I$ (I holds when entering the loop)

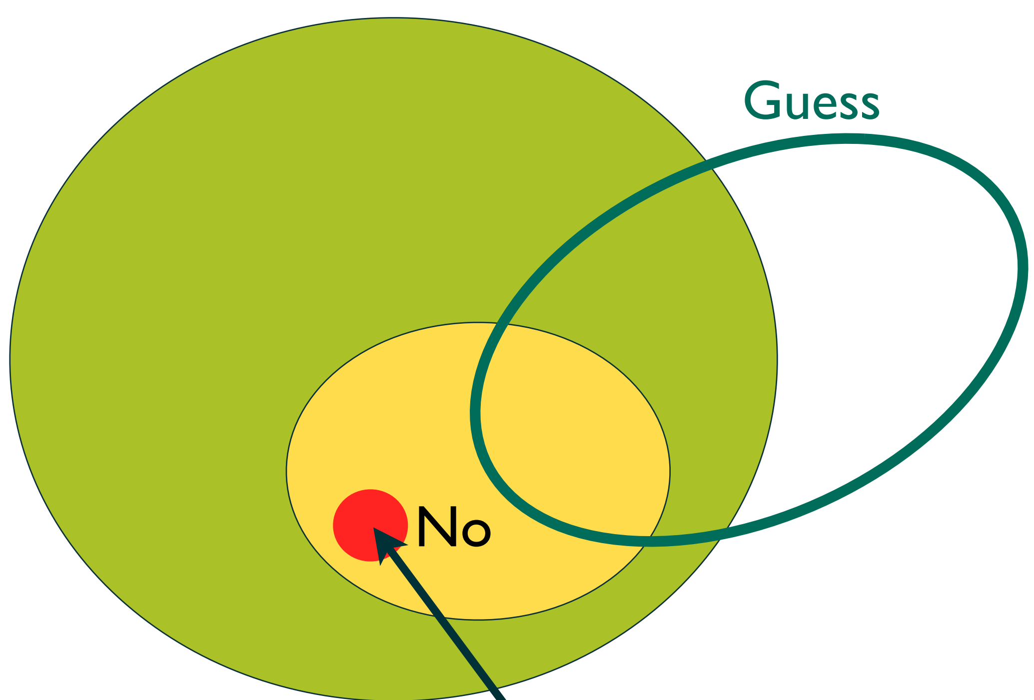
(B) $I \wedge \kappa \Rightarrow Pre(I, S)$ (I holds at each iteration)

(C) $I \wedge \neg\kappa \Rightarrow \epsilon$ (I gives ϵ after leaving the loop)

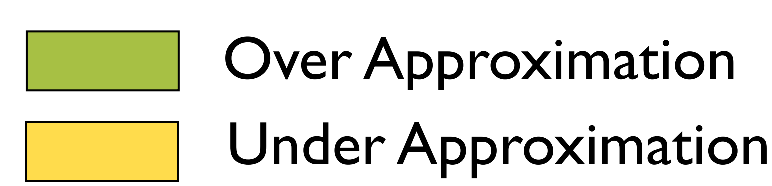
Equivalence Query Resolution

1. “YES”, if the guess satisfies invariant conditions.
2. Otherwise, we need a counter example to answer “No”.

Case 1



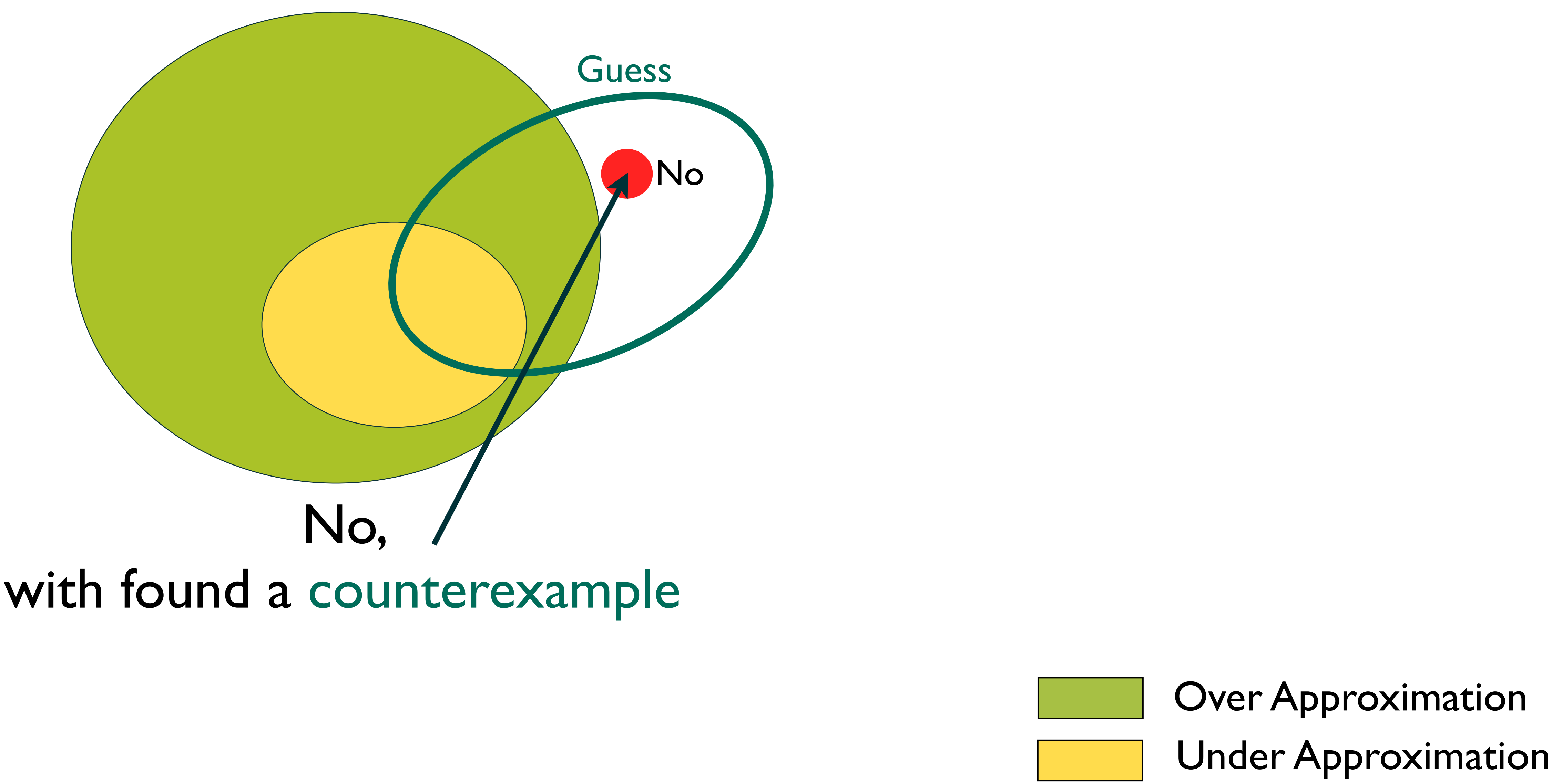
No,
with found a counterexample



Equivalence Query Resolution

1. "YES", if the guess satisfies invariant conditions.
2. Otherwise, we need a counter example to answer "No".

Case 2

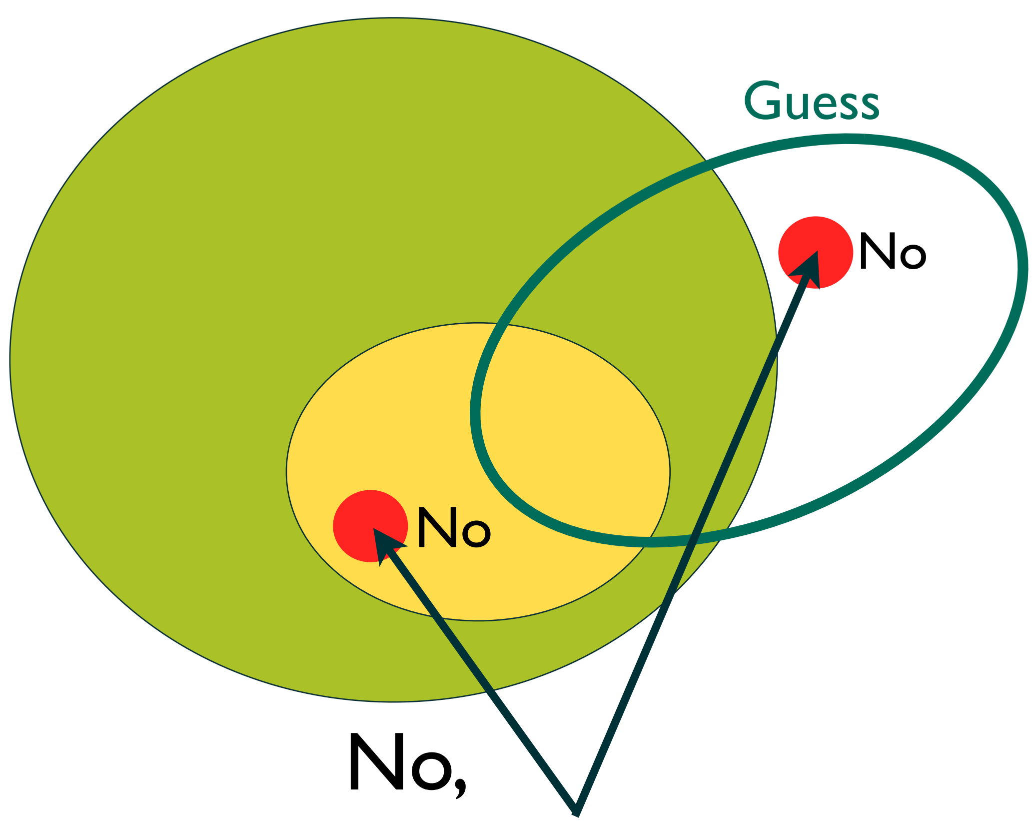


Equivalence Query Resolution

1. “YES”, if the guess satisfies invariant conditions.

2. Otherwise, we need a counter example to answer “No”.

Case 1 & 2



No,
with found a counterexample

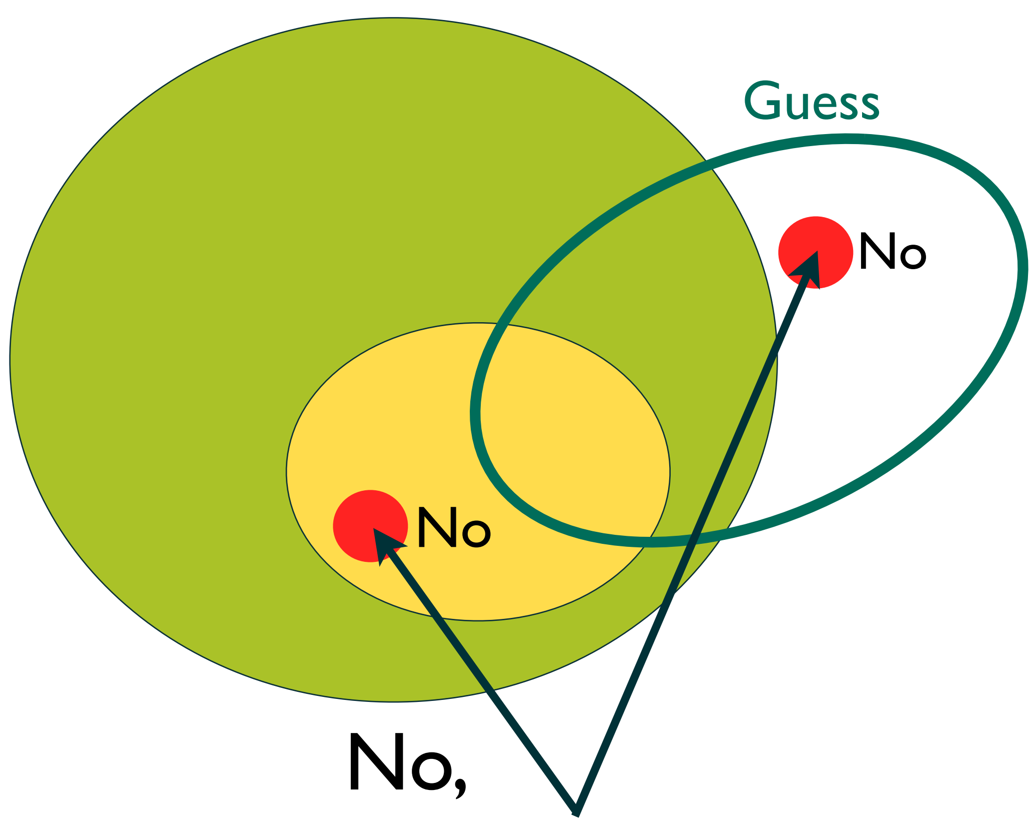
Over Approximation

Under Approximation

Equivalence Query Resolution

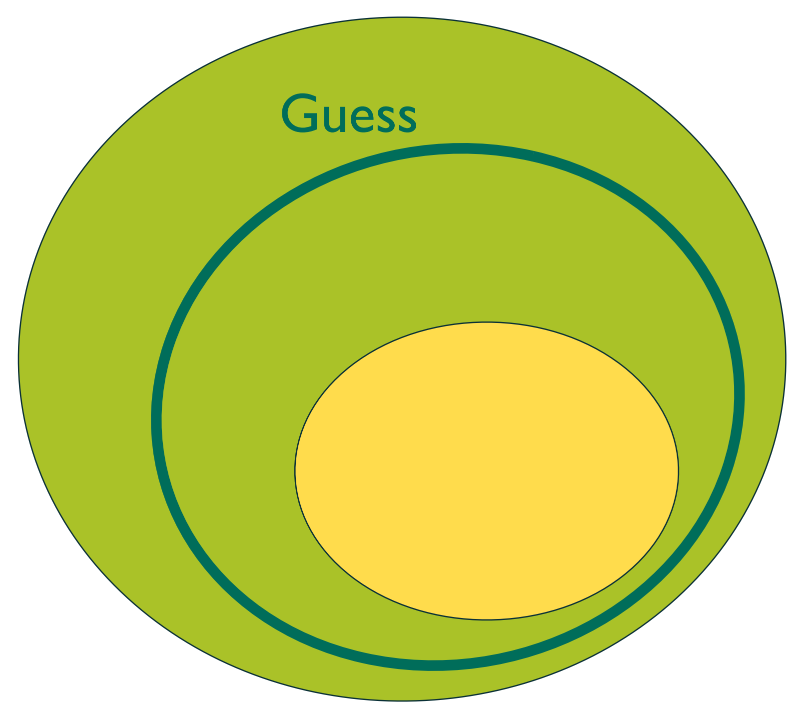
1. "YES", if the guess satisfies invariant conditions.
2. Otherwise, we need a counter example to answer "No".

Case 1 & 2



No,
with found a counterexample

Case 3



Cannot find a counterexample.

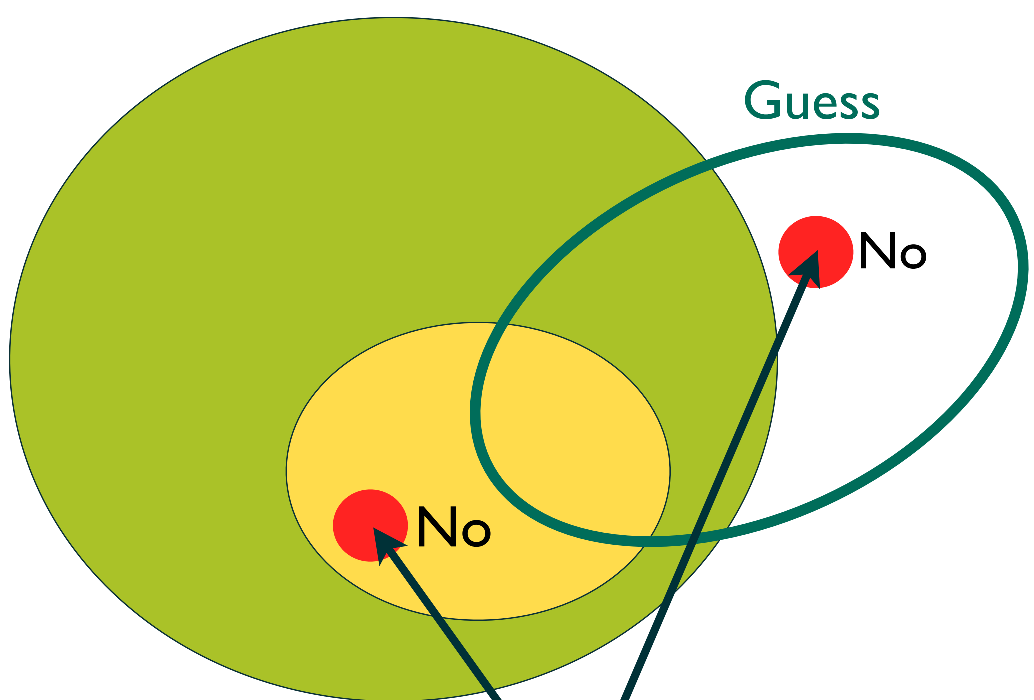
Over Approximation
Under Approximation

Equivalence Query Resolution

1. "YES", if the guess satisfies invariant conditions.

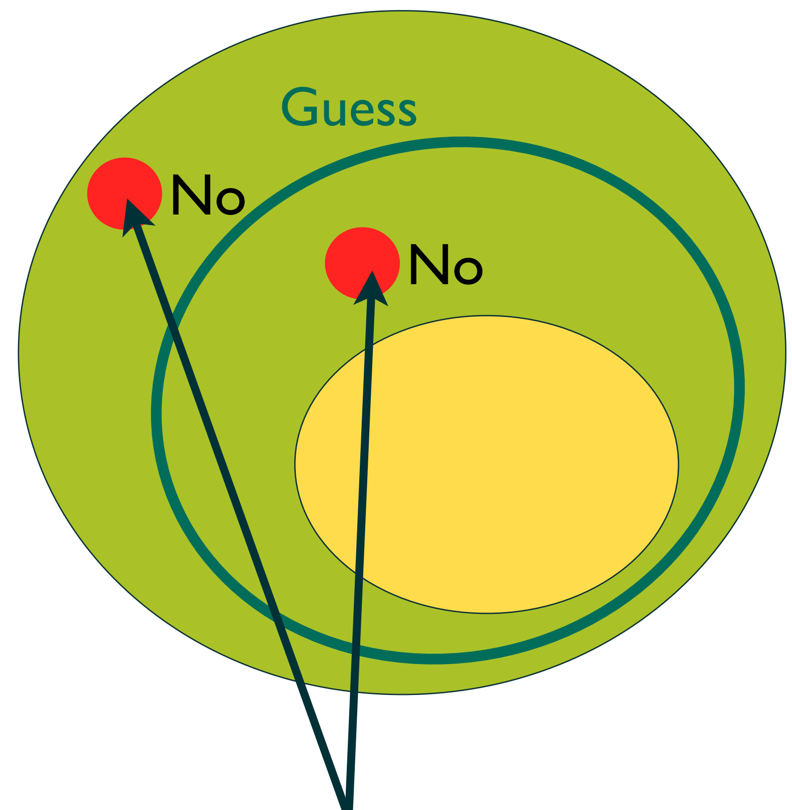
2. Otherwise, we need a counter example to answer "No".

Case 1 & 2



No,
with found a counterexample

Case 3



A random counterexample.

Membership Query Resolution

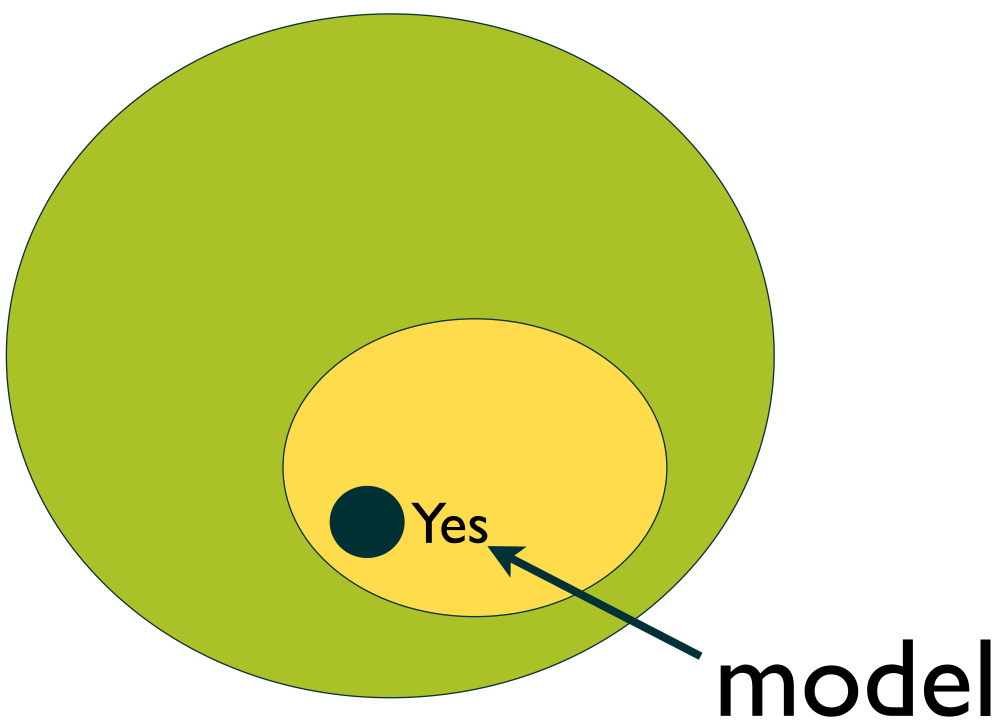
1. "NO", if the model is unsatisfiable. $i = 0 \wedge i = 1$

Membership Query Resolution

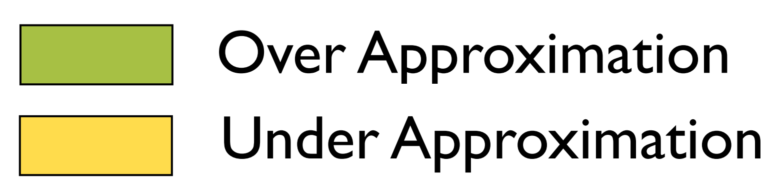
1. “NO”, if the model is unsatisfiable.

2. Use approximations to answer the query.

Case 1



Answer Yes

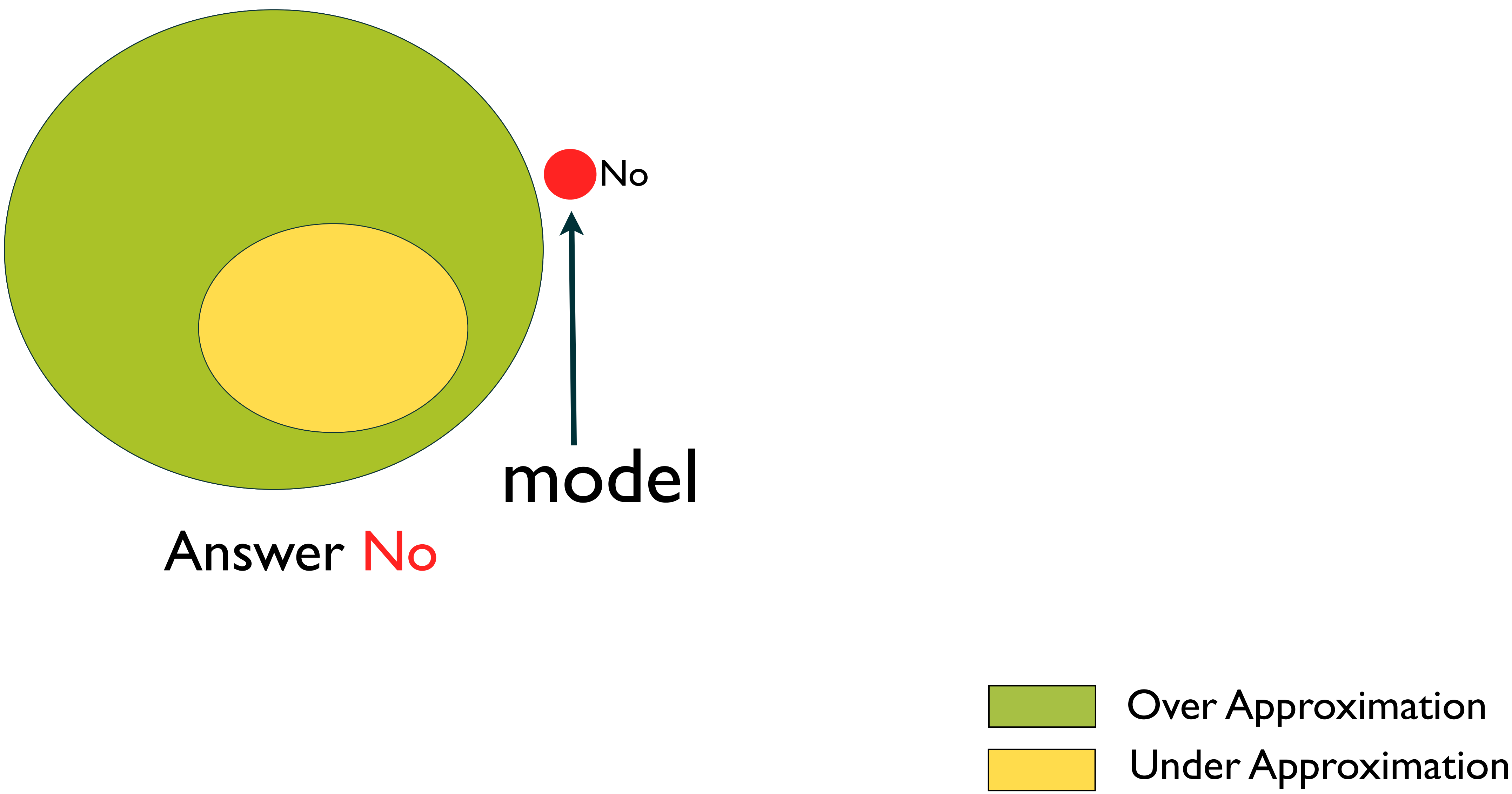


Membership Query Resolution

1. "NO", if the model is unsatisfiable.

2. Use approximations to answer the query.

Case 2

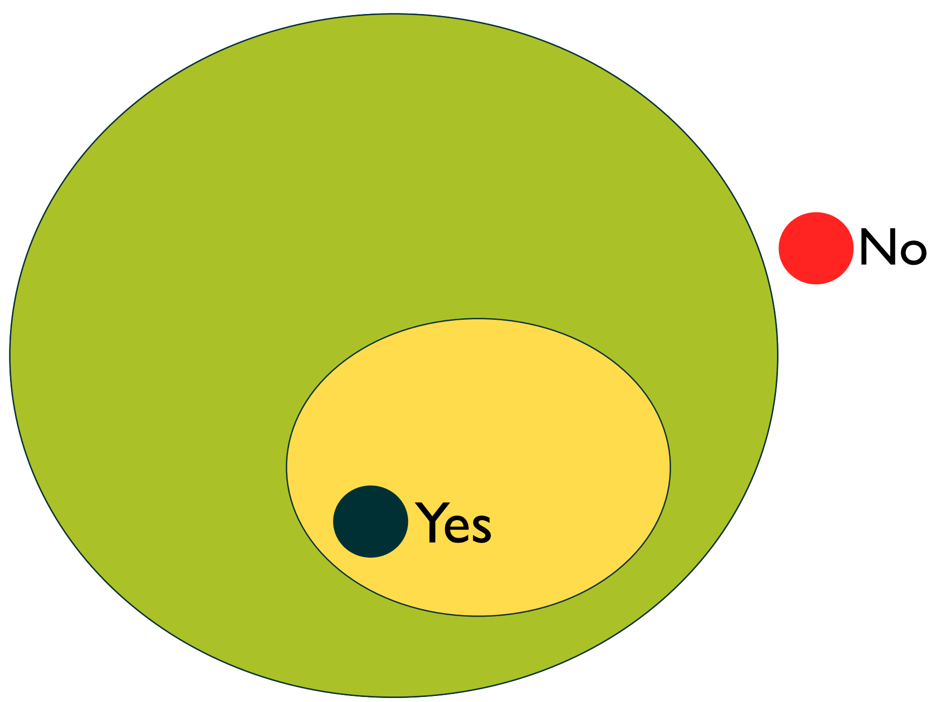


Membership Query Resolution

1. "NO", if the model is unsatisfiable.

2. Use approximations to answer the query.

Case 1 & 2



Answer Yes or No

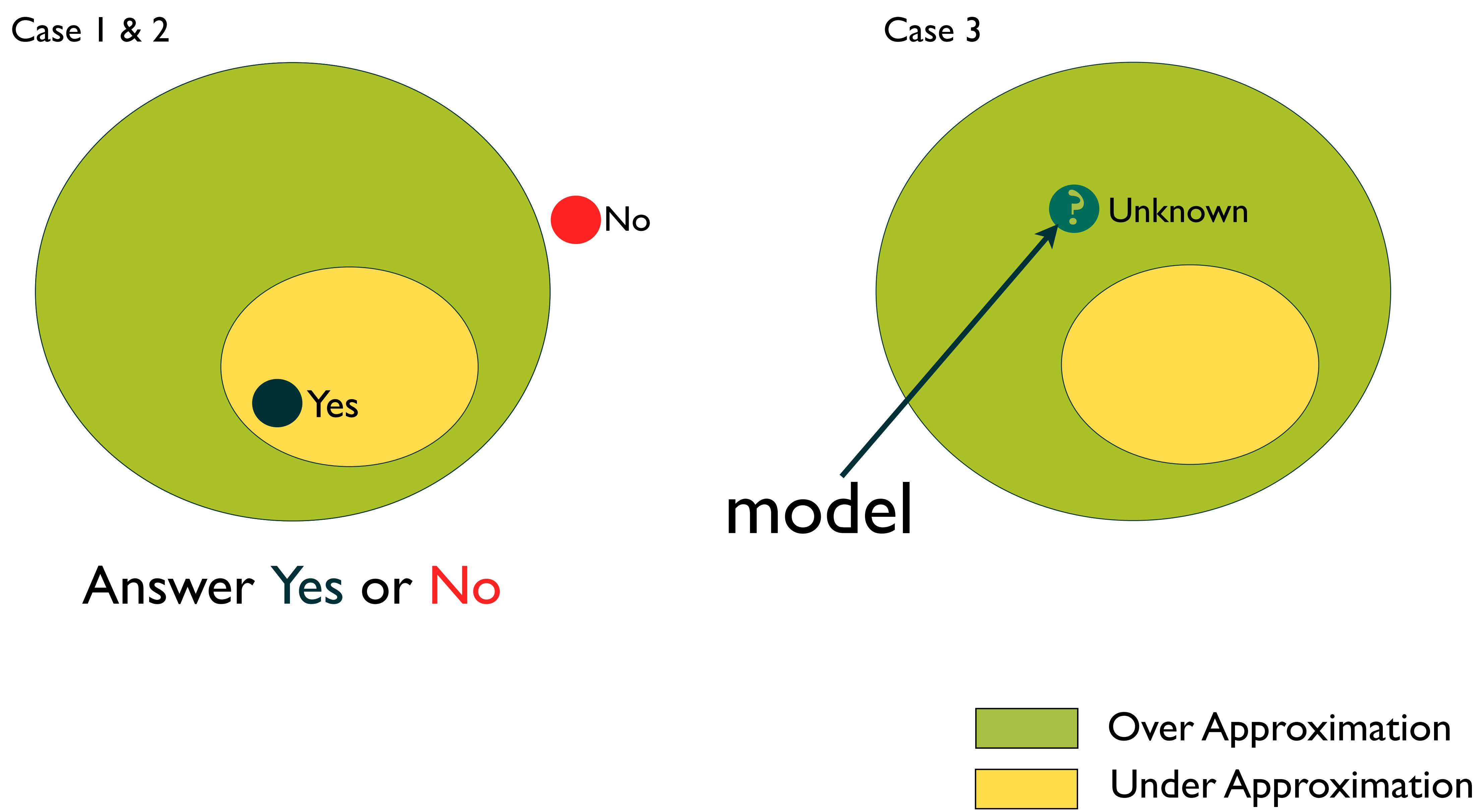
Over Approximation

Under Approximation

Membership Query Resolution

1. "NO", if the model is unsatisfiable.

2. Use approximations to answer the query.

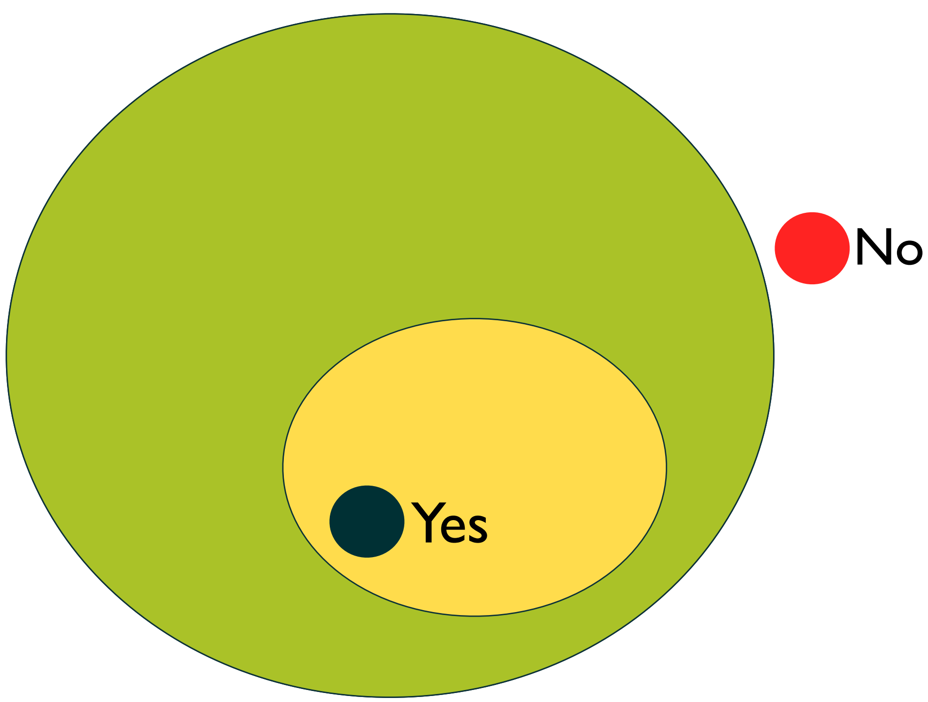


Membership Query Resolution

1. "NO", if the model is unsatisfiable.

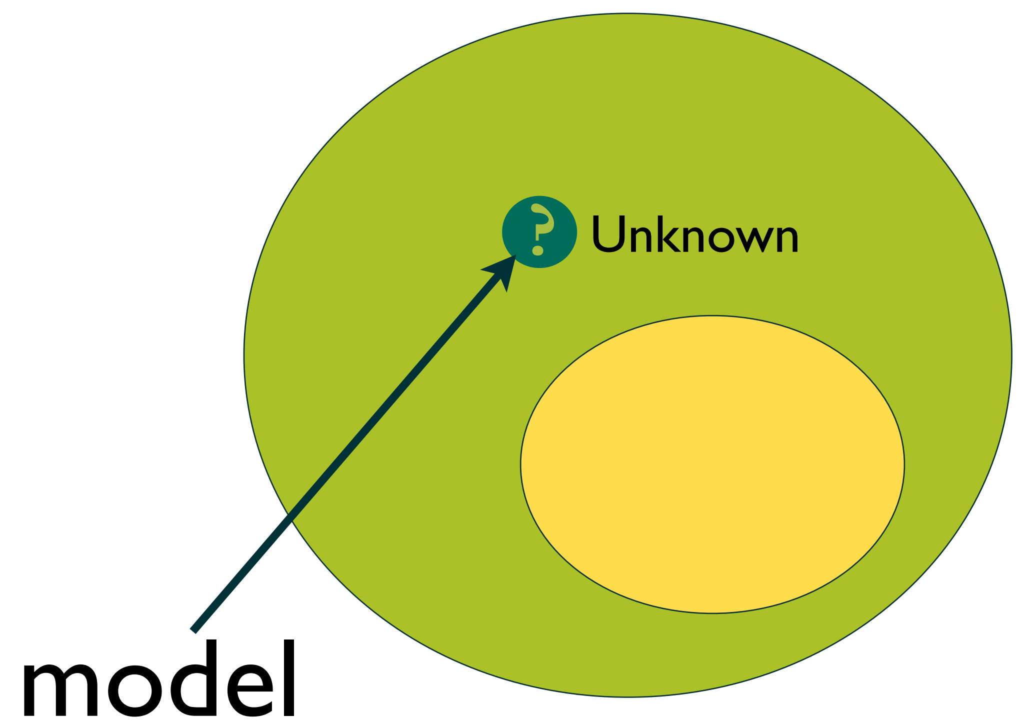
2. Use approximations to answer the query.

Case 1 & 2



Answer Yes or No

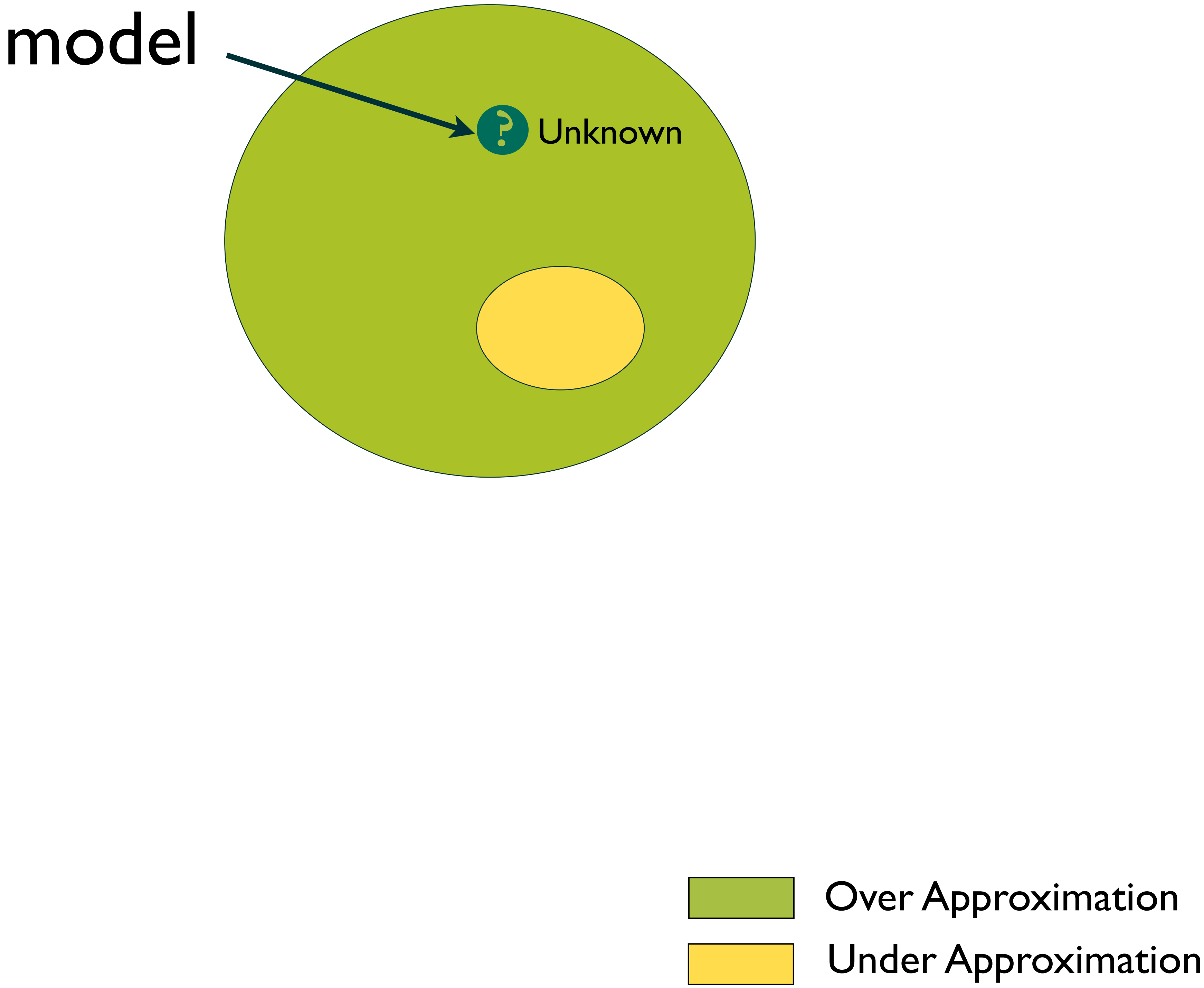
Case 3



Answer Yes or No randomly

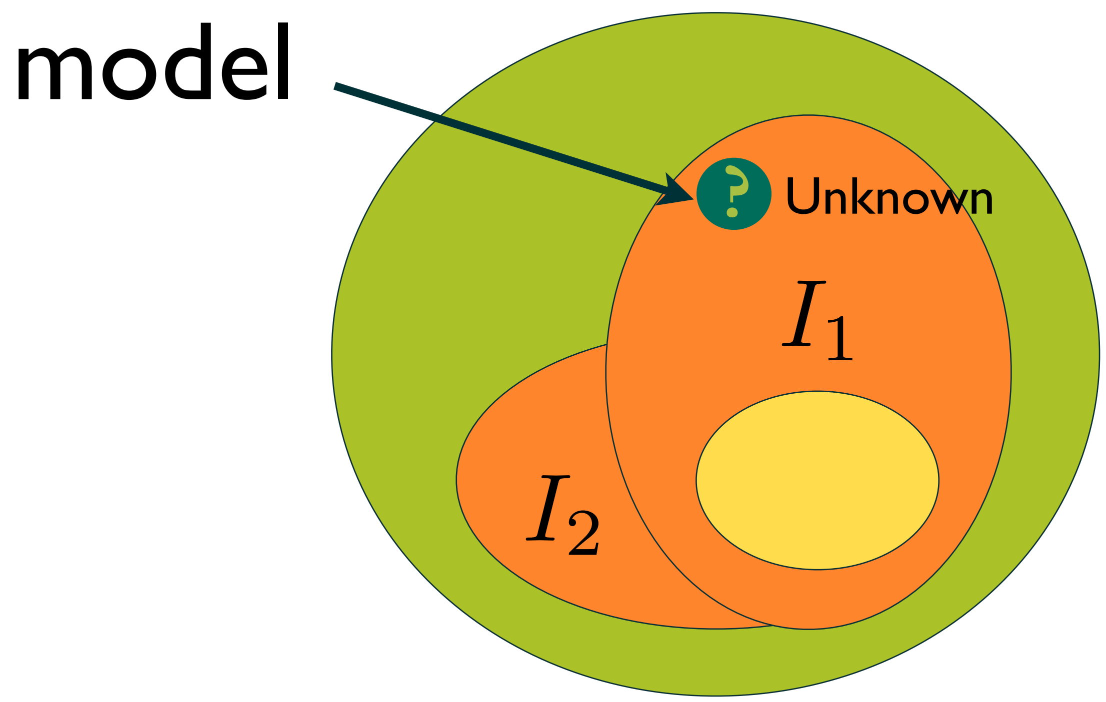
Effect of Random Answer

Membership Query

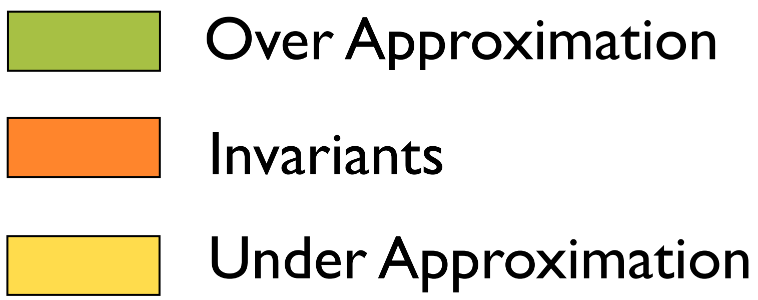


Effect of Random Answer

Membership Query

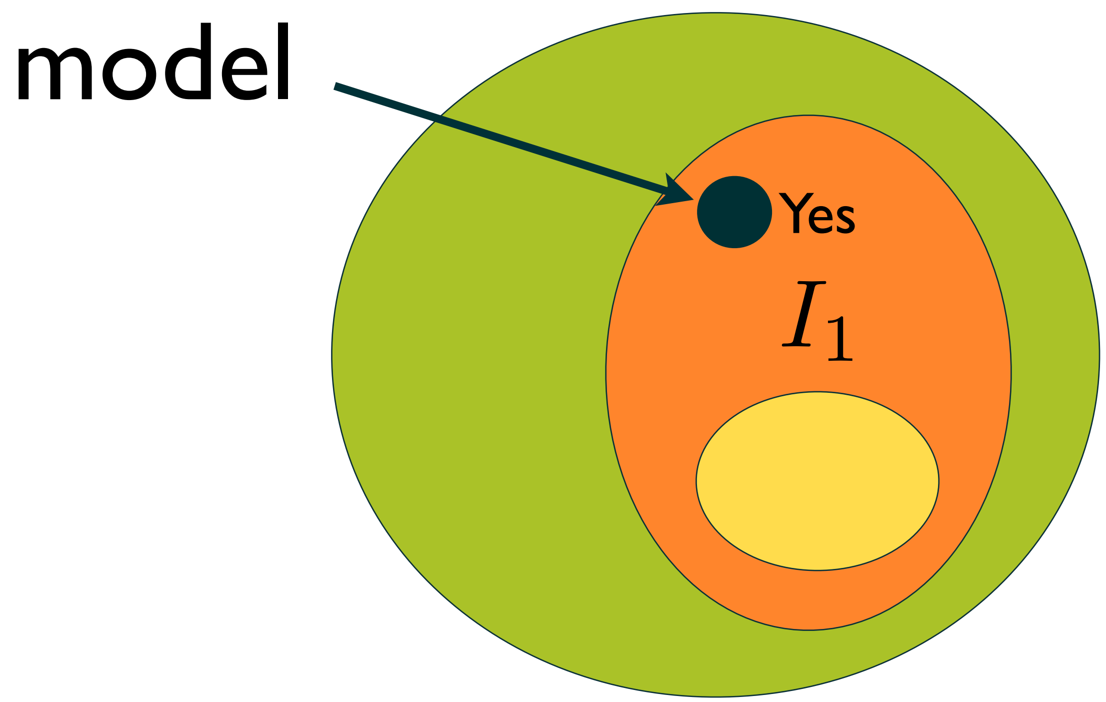


Both of the random answers can lead to an invariant.

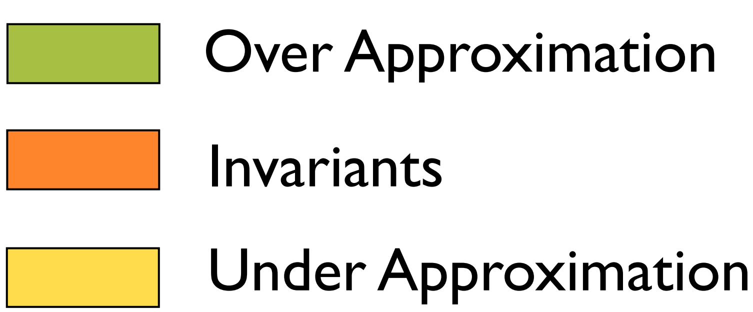


Effect of Random Answer

Membership Query

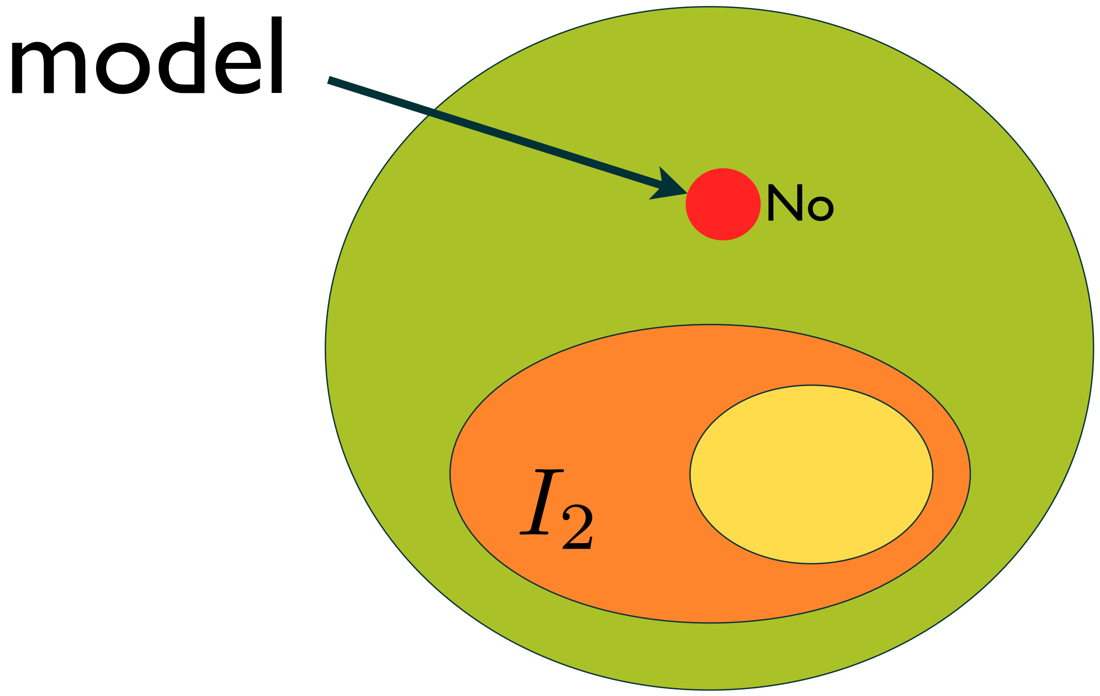


"Yes" leads to I_1

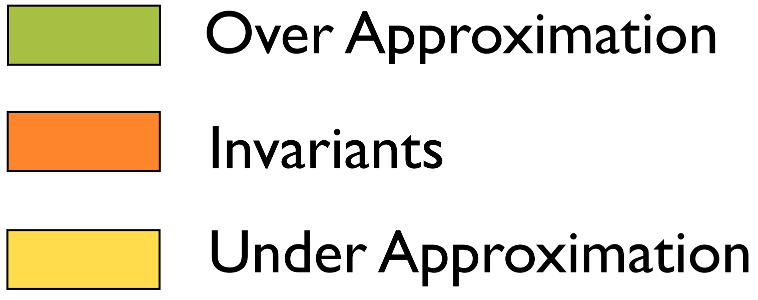


Effect of Random Answer

Membership Query



“No” leads to I_2



Random Algorithm

- Random Membership and Equivalence query resolution causes conflict!
- Then we simply restart the whole algorithm

Random Algorithm

- Random Membership and Equivalence query resolution causes conflict!
- Then we simply restart the whole algorithm

Memoization could not save the time :(
Because the search space is huge

Random Algorithm

- Random Membership and Equivalence query resolution causes conflict!
- Then we simply restart the whole algorithm

Memoization could not save the time :(
Because the search space is huge

$$2^{2^n}$$

where n is #atomic propositions

It's still Sound

Why?

When resolving equivalence query

(A) $\delta \Rightarrow I$ (I holds when entering the loop)

(B) $I \wedge \kappa \Rightarrow Pre(I, S)$ (I holds at each iteration)

(C) $I \wedge \neg\kappa \Rightarrow \epsilon$ (I gives ϵ after leaving the loop)

1. "YES", if the guess satisfies invariant conditions.

We always **verify** the conditions before say "Yes".

SMT solvers are not complete but **sound**

for **quantified** formulae.

Experiment Results

Average of 500 runs

Program	Template	AP	Total MEM	Total EQ	Random MEM	Random EQ	ITER	Time
max	$\forall k.[\ ]$	7	5,968	1,742	65%	26%	269	5.7s
selection_sort	$\forall k_1.\exists k_2.[\ ]$	6	9,630	5,832	100%	4%	1,672	9.6s
devres	$\forall k.[\ ]$	7	2,084	1,214	91%	21%	310	0.9s
rm_pkey	$\forall k.[\ ]$	8	2,204	919	67%	20%	107	2.5s
tracepoint1	$\exists k.[\ ]$	4	246	195	61%	25%	31	0.3s
tracepoint2	$\forall k_1.\exists k_2.[\ ]$	7	33,963	13,063	69%	5%	2,088	157.6s

Experiment Results

Average of 500 runs

Program	Template	AP	Total MEM	Total EQ	Random MEM	Random EQ	ITER	Time
max	$\forall k.[\ ]$	7	5,968	1,742	65%	26%	269	5.7s
selection_sort	$\forall k_1.\exists k_2.[\ ]$	6	9,630	5,832	100%	4%	1,672	9.6s
devres	$\forall k.[\ ]$	7	2,084	1,214	91%	21%	310	0.9s
rm_pkey	$\forall k.[\ ]$	8	2,204	919	67%	20%	107	2.5s
tracepoint1	$\exists k.[\ ]$	4	246	195	61%	25%	31	0.3s
tracepoint2	$\forall k_1.\exists k_2.[\ ]$	7	33,963	13,063	69%	5%	2,088	157.6s

1. Simple templates are enough $\quad \tau \triangleq [] \mid \forall I.\tau \mid \exists I.\tau$

Experiment Results

Average of 500 runs

Program	Template	AP	Total MEM	Total EQ	Random MEM	Random EQ	ITER	Time
max	$\forall k.[\ ]$	7	5,968	1,742	65%	26%	269	5.7s
selection_sort	$\forall k_1.\exists k_2.[\ ]$	6	9,630	5,832	100%	4%	1,672	9.6s
devres	$\forall k.[\ ]$	7	2,084	1,214	91%	21%	310	0.9s
rm_pkey	$\forall k.[\ ]$	8	2,204	919	67%	20%	107	2.5s
tracepoint1	$\exists k.[\ ]$	4	246	195	61%	25%	31	0.3s
tracepoint2	$\forall k_1.\exists k_2.[\ ]$	7	33,963	13,063	69%	5%	2,088	157.6s

1. Simple templates are enough $\quad \tau \triangleq [] \mid \forall I.\tau \mid \exists I.\tau$
2. Random algorithm works well

Conclusion

- Algorithmic Learning + Decision Procedures + Predicate Abstraction + Simple Template => Quantified Invariant Generation Technique
- Exploits the flexibility in invariants by randomized mechanism.
- Static/Dynamic Analysis can help with tighter approximations on invariants.
- Apply the CDNF algorithm to your own problems.

Conclusion

- Algorithmic Learning + Decision Procedures + Predicate Abstraction + Simple Template => Quantified Invariant Generation Technique
- Exploits the flexibility in invariants by randomized mechanism.
- Static/Dynamic Analysis can help with tighter approximations on invariants.
- Apply the CDNF algorithm to your own problems.

Thanks!