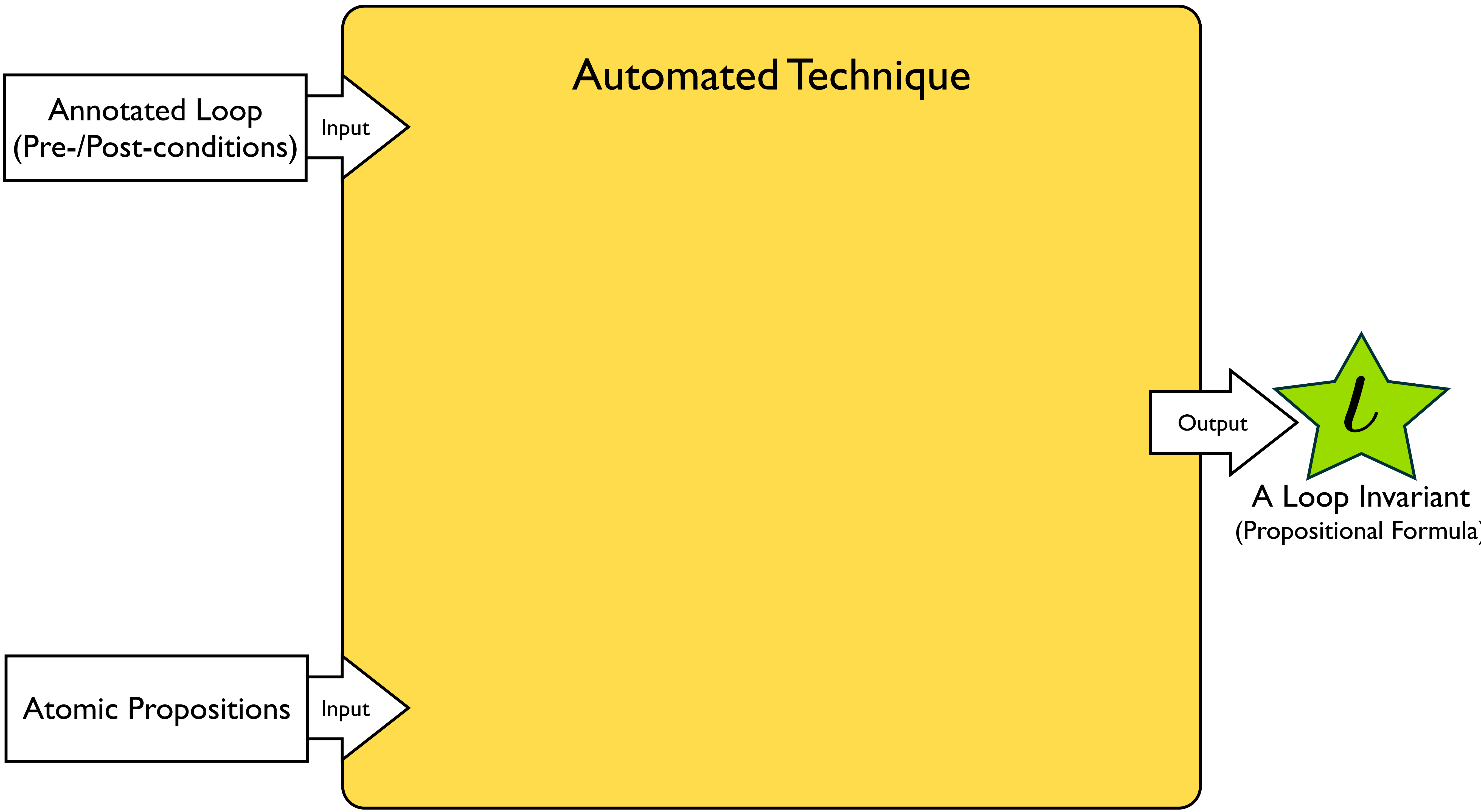
# Deriving Invariants by Algorithmic Learning, Decision Procedure, and Predicate Abstraction

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

[1] Seoul National University [2] Academia Sinica

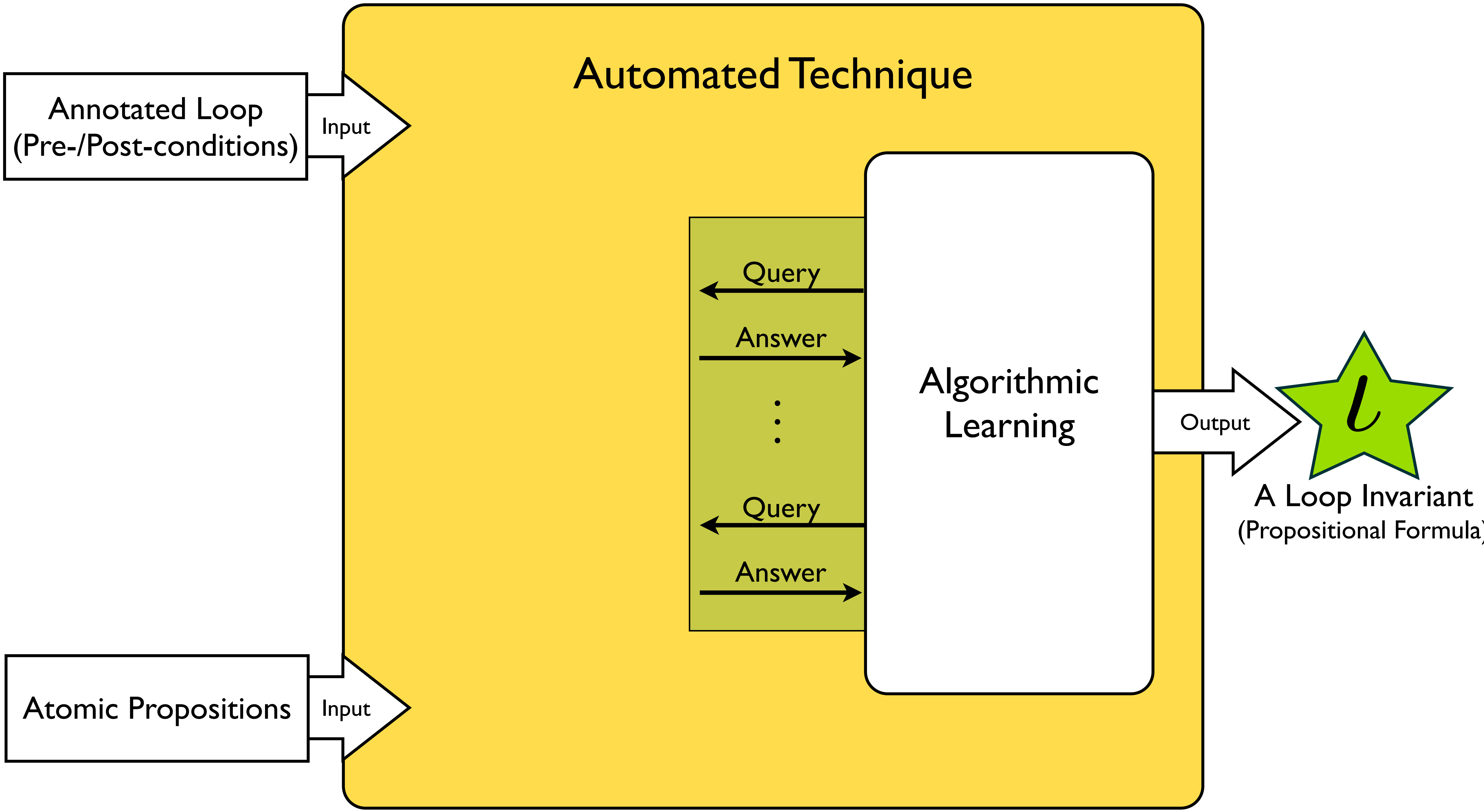
VMCAI'10, 2010/01/18 @ Madrid, Spain

# Overview

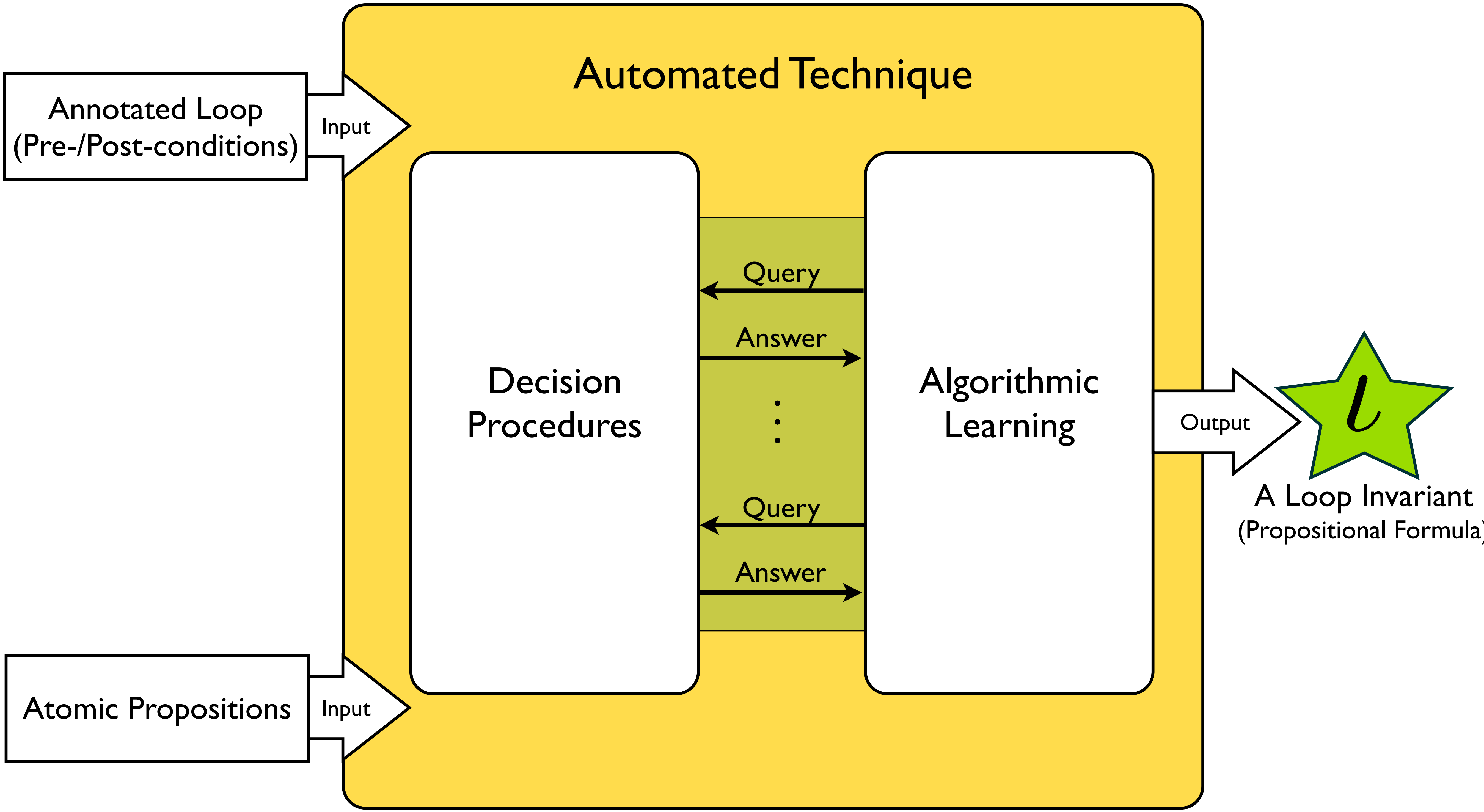


Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Atomic Propositions
Input
Output
$\iota$
A Loop Invariant
(Propositional Formula)

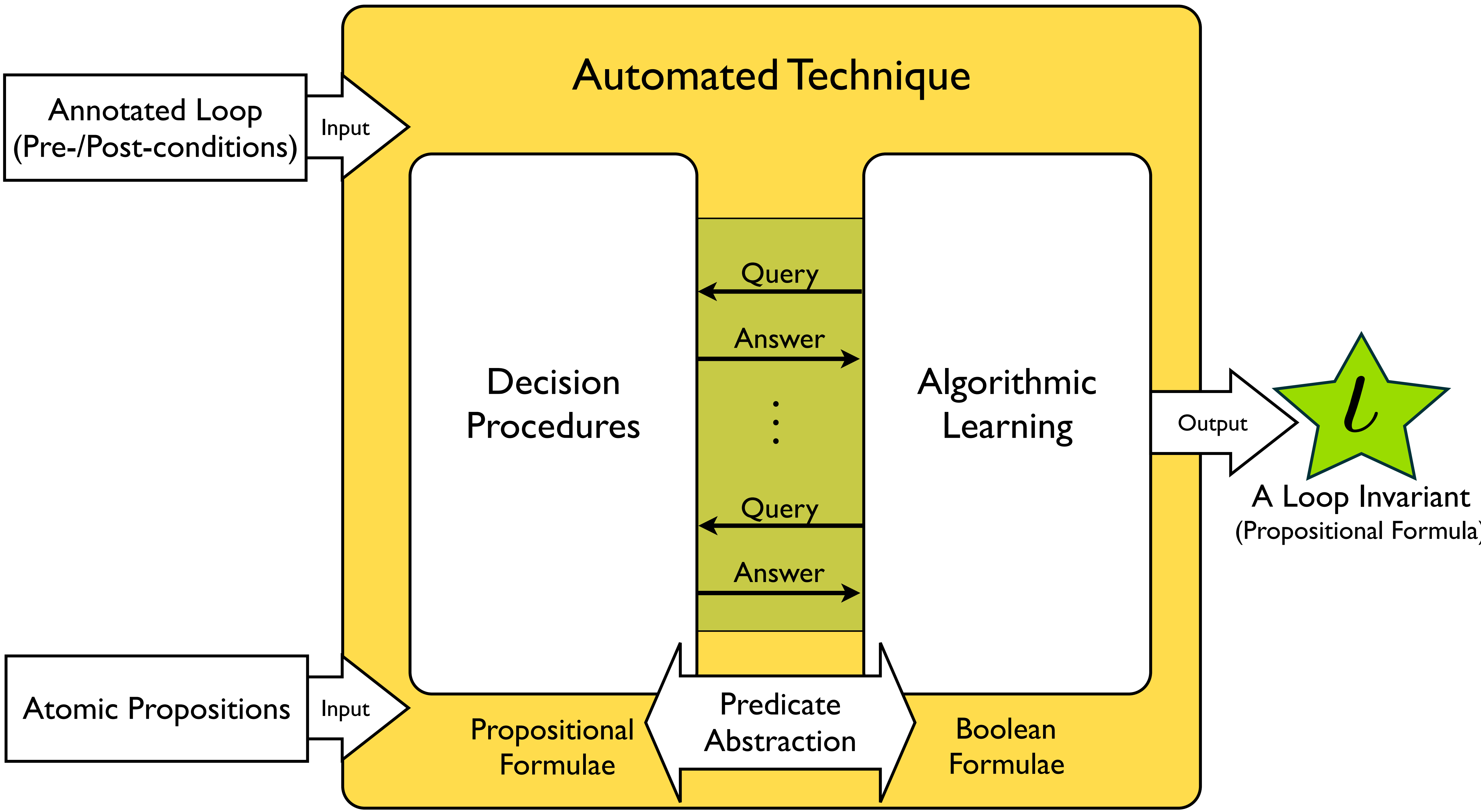
# Overview


Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Atomic Propositions
Input
Query
Answer
Query
Answer
Algorithmic
Learning
Output
$\iota$
A Loop Invariant
(Propositional Formula)

# Overview


Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Atomic Propositions
Input
Decision
Procedures
Query
Answer
Query
Answer
Algorithmic
Learning
Output
$\iota$
A Loop Invariant
(Propositional Formula)

# Overview


Automated Technique
Annotated Loop
(Pre-/Post-conditions)
Input
Decision
Procedures
Query
Answer
Query
Answer
Algorithmic
Learning
Output
A Loop Invariant
(Propositional Formula)
Atomic Propositions
Input
Propositional
Formulae
Predicate
Abstraction
Boolean
Formulae

$\{\ phase = \mathtt{F} \wedge success = \mathtt{F} \wedge give\_up = \mathtt{F} \wedge cutoff = 0 \wedge count = 0\ \}$

```
while ¬(success ∨ give_up) do
    entered_phase := F;
    if ¬phase then
        if cutoff = 0 then cutoff := 1;
        else if cutoff = 1 ∧ maxcost > 1 then cutoff := maxcost;
            else phase := T; entered_phase := T; cutoff := 1000;
        if cutoff = maxcost ∧ ¬search then give_up := T;
    else
        count := count + 1;
        if count > words then give_up := T;
    if entered_phase then count := 1;
    linkages := nondet;
    if linkages > 5000 then linkages := 5000;
    canonical := 0; valid := 0;
    if linkages ≠ 0 then
        valid := nondet; assume 0 ≤ valid ∧ valid ≤ linkages;
        canonical := linkages;
    if valid > 0 then success := T;
end
```

$\{\ (valid > 0 \vee count > words \vee (cutoff = maxcost \wedge \neg search)) \wedge$
$valid \leq linkages \wedge canonical = linkages \wedge linkages \leq 5000\ \}$

**Fig. 3.** A Sample Loop in SPEC2000 Benchmark PARSER

with 20 Atomic Propositions (Building Blocks)

$\{\ phase = \mathtt{F} \wedge success = \mathtt{F} \wedge give\_up = \mathtt{F} \wedge cutoff = 0 \wedge count = 0\ \}$

```
while ¬(success ∨ give_up) do
    entered_phase := F;
    if ¬phase then
        if cutoff = 0 then cutoff := 1;
        else if cutoff = 1 ...
```

$$
\begin{aligned}
& success \Rightarrow (valid \leq linkages \wedge linkages \leq 5000 \wedge canonical = linkages) \bigwedge \\
& success \Rightarrow (\neg search \vee count > words \vee valid \neq 0) \bigwedge \\
& success \Rightarrow (count > words \vee cutoff = maxcost \vee (canonical \neq 0 \wedge valid \neq 0 \wedge linkages \neq 0)) \bigwedge \\
& give\_up \Rightarrow ((valid = 0 \wedge linkages = 0 \wedge canonical = linkages) \vee \\
& \qquad (canonical \neq 0 \wedge valid \leq linkages \wedge linkages \leq 5000 \wedge canonical = linkages)) \bigwedge \\
& give\_up \Rightarrow (cutoff = maxcost \vee count > words \vee \\
& \qquad (canonical \neq 0 \wedge valid \neq 0 \wedge linkages \neq 0)) \bigwedge \\
& give\_up \Rightarrow (\neg search \vee count > words \vee valid \neq 0)
\end{aligned}
$$

Find this Invariant in 33 sec

```
        valid := nondet; assume 0 ≤ valid ∧ valid ≤ linkages;
        canonical := linkages;
    if valid > 0 then success := T;
end
```

$\{\ (valid > 0 \vee count > words \vee (cutoff = maxcost \wedge \neg search)) \wedge$
$valid \leq linkages \wedge canonical = linkages \wedge linkages \leq 5000\ \}$

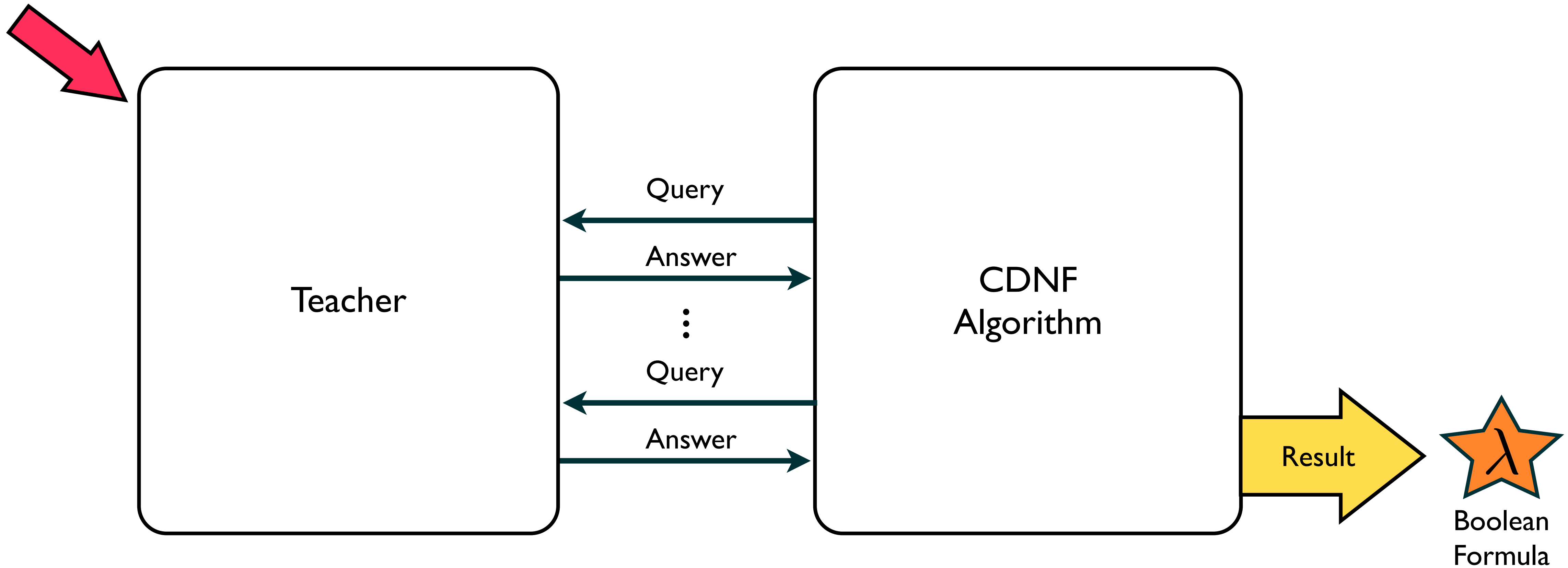
**Fig. 3.** A Sample Loop in SPEC2000 Benchmark PARSER

with 20 Atomic Propositions (Building Blocks)

# Algorithmic Learning: CDNF Algorithm

# CDNF Algorithm[†]

Teacher is required




Actively learning a Boolean formula
from membership and equivalence queries
(polynomial # of queries in $|\lambda|$ and # of variables)

† 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 $\mu$ satisfies the Boolean formula $\lambda$

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

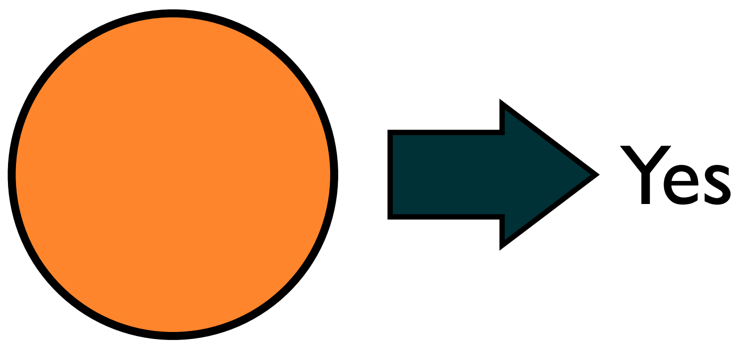
Example: $\lambda =$ XOR function $b_1 \oplus b_2$

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

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

# Equivalence Query

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



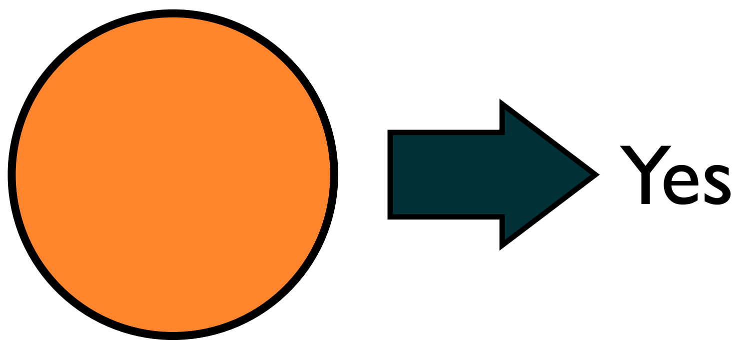

if $\beta \equiv \lambda$

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 $\beta$ is equivalent to $\lambda$.

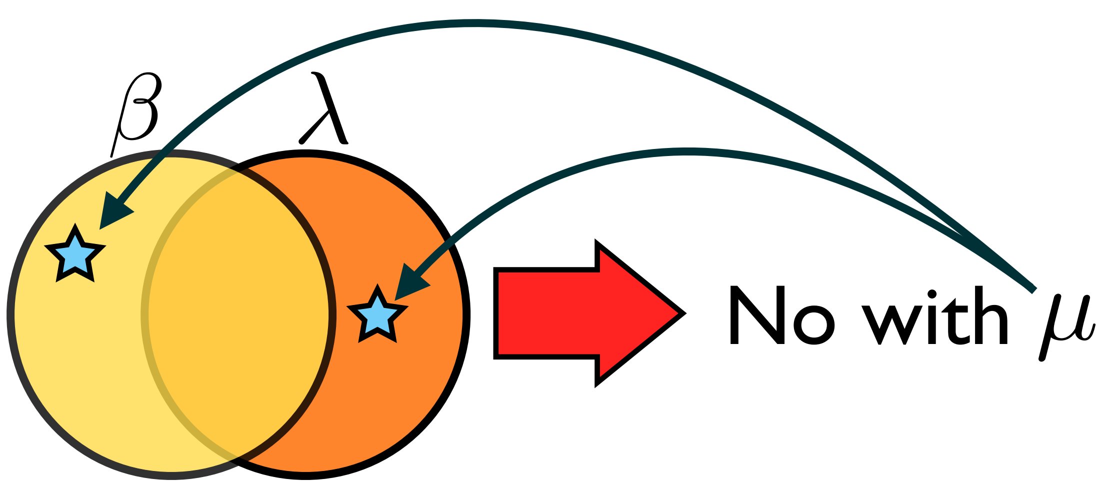



if $\beta \equiv \lambda$

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 $\mu$.



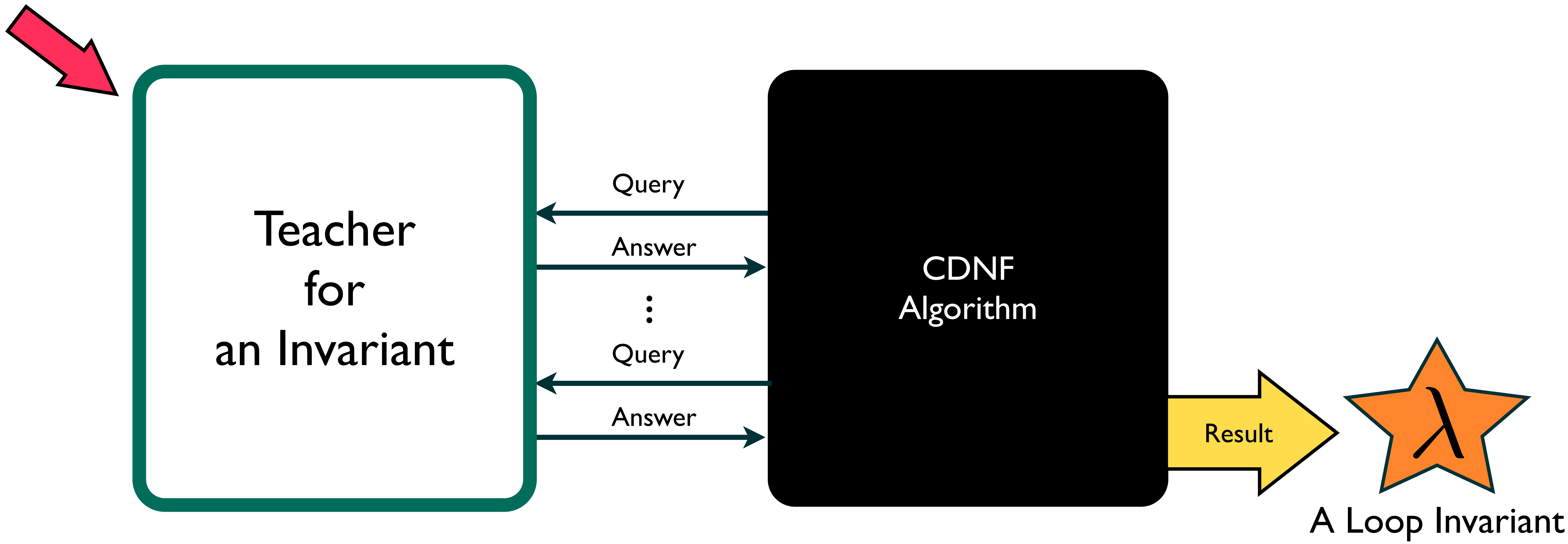

if $\beta \not\equiv \lambda \wedge (\mu \models \beta \oplus \lambda)$

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




Implement a Teacher
to guide CDNF algorithm infer an Invariant

# Predicate Abstraction

# Predicate Abstraction

Problem:

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

$$\begin{array}{ccc} \text{Propositional Formula} & \xrightarrow{?} & \text{Boolean Formula} \\ \mathtt{i} < 10 \vee (\mathtt{i} = 10 \wedge \mathtt{ret}) & \longleftarrow & b_1 \wedge (b_2 \vee b_3) \end{array}$$

# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions.

$$AP = \{i < 10,\ i = 10,\ ret\}$$

$EQ(\beta)$

Boolean Formula

$b_{\mathtt{i<10}} \vee (b_{\mathtt{i=10}} \wedge b_{\mathtt{ret}})$

Learning Algorithm

# Predicate Abstraction

Solution:
Use predicate abstraction with given atomic propositions.

$$AP = \{i < 10,\ i = 10,\ ret\}$$

$EQ(\beta)$

Propositional Formula

$\texttt{i} < 10 \lor (\texttt{i} = 10 \land \texttt{ret})$

$\xrightarrow{\alpha}$

$\xleftarrow{\gamma}$

Boolean Formula

$b_{\texttt{i}<10} \lor (b_{\texttt{i}=10} \land b_{\texttt{ret}})$

SMT Solver

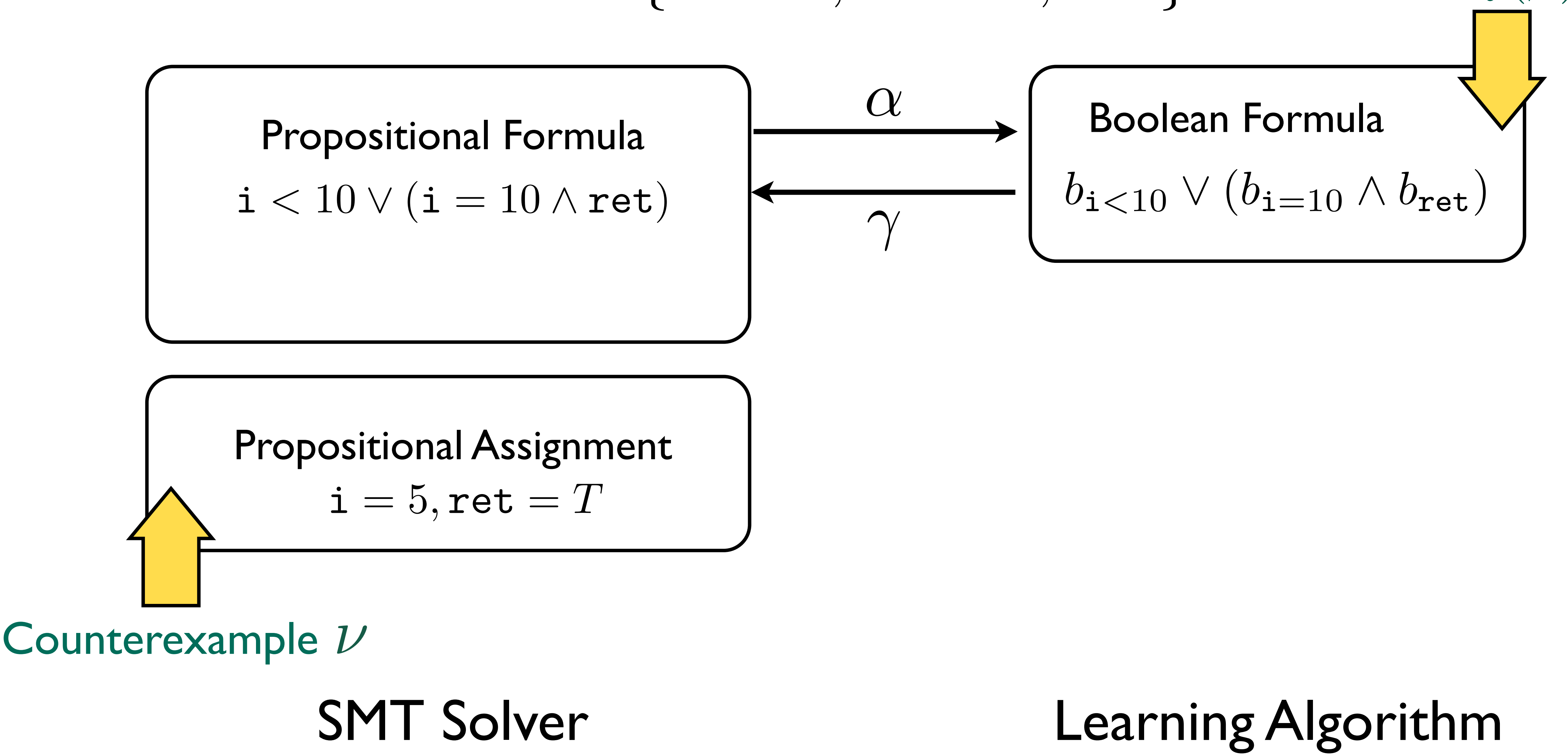
Learning Algorithm

# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions.

$$AP = \{i < 10,\ i = 10,\ ret\}$$

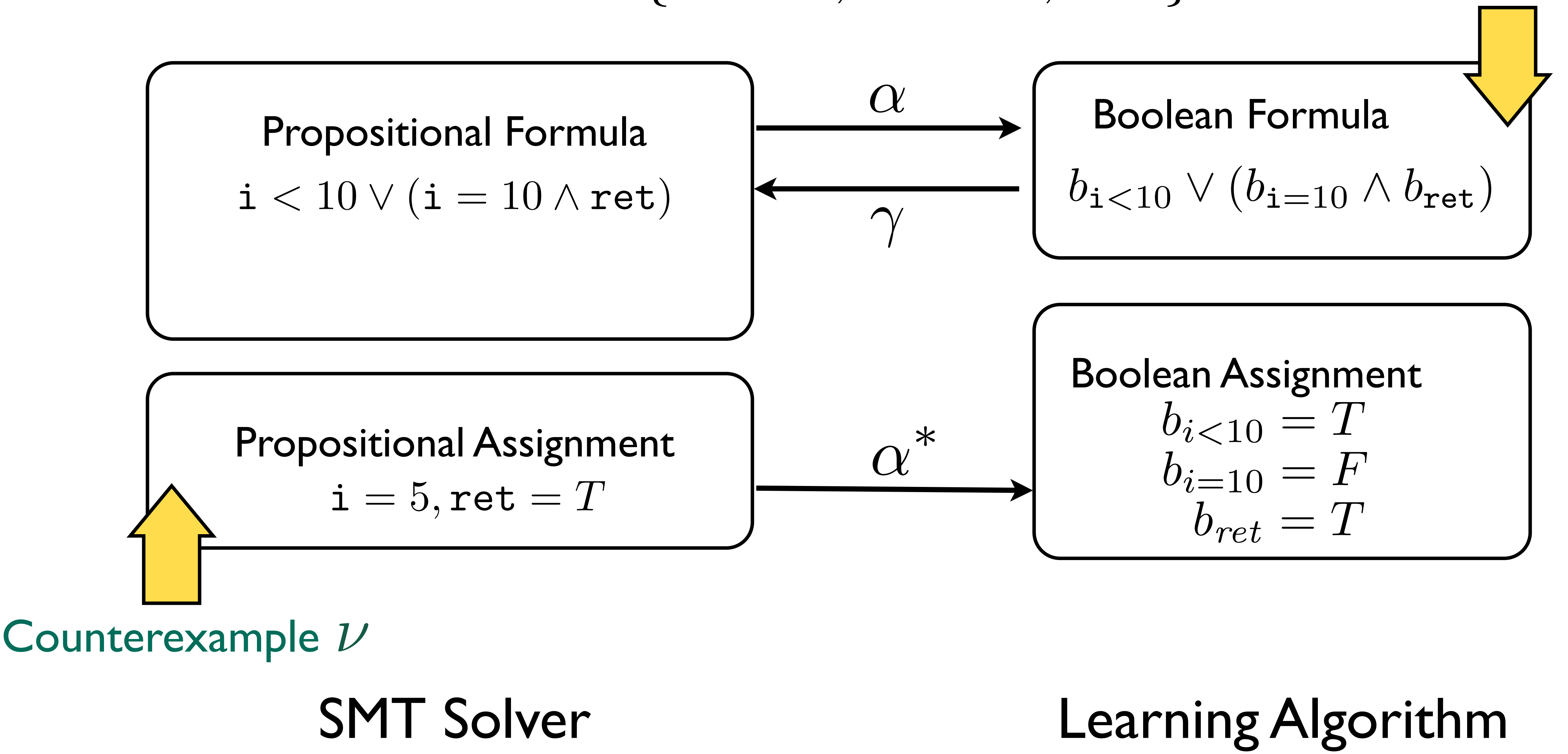


# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions.

$$AP = \{i < 10,\ i = 10,\ ret\}$$

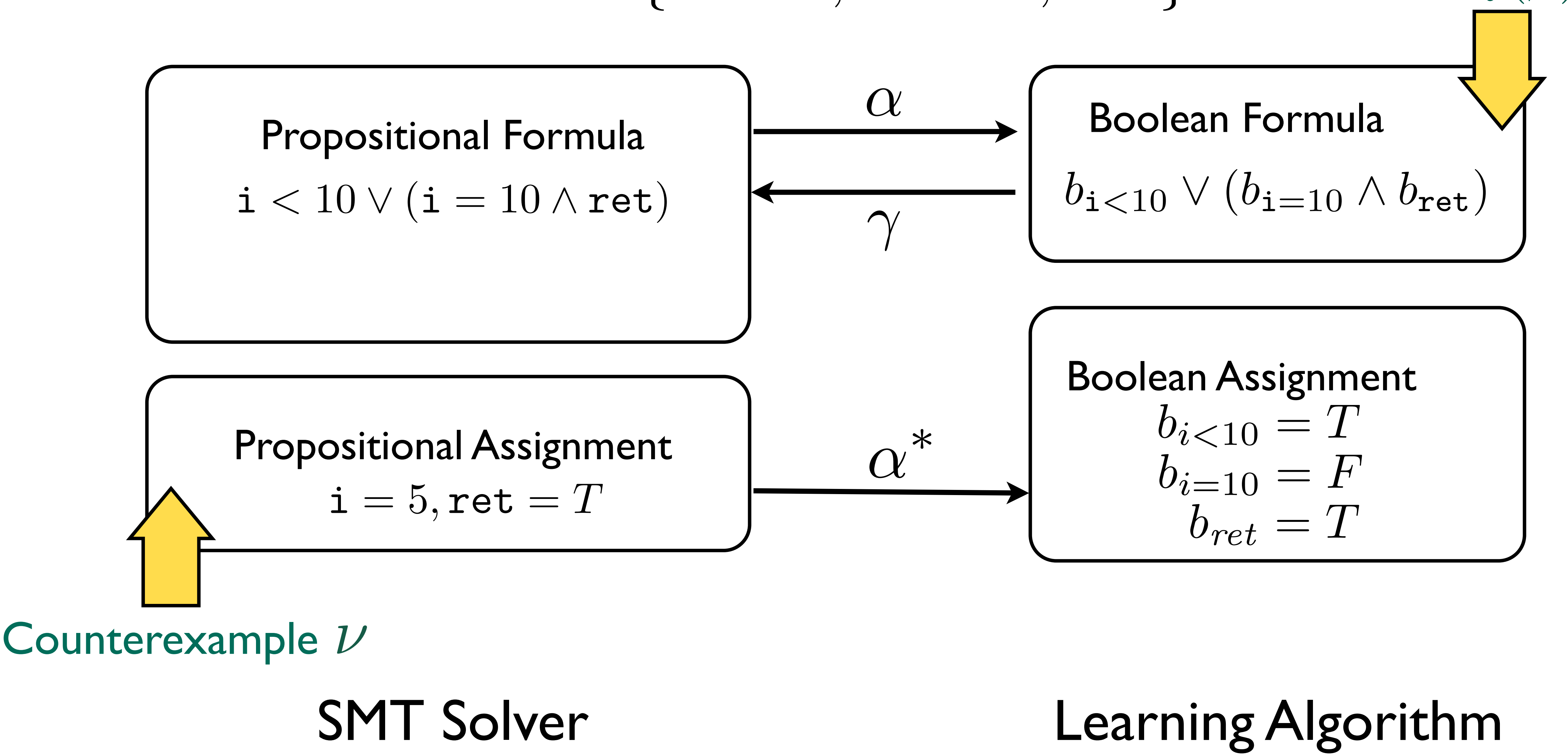


# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions

$$AP = \{i < 10,\ i = 10,\ ret\}$$



# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions.

$$AP = \{i < 10,\ i = 10,\ ret\}$$

Boolean Assignment

$$b_{i<10} = T$$
$$b_{i=10} = F$$
$$b_{ret} = T$$

$MEM(\mu)$

SMT Solver

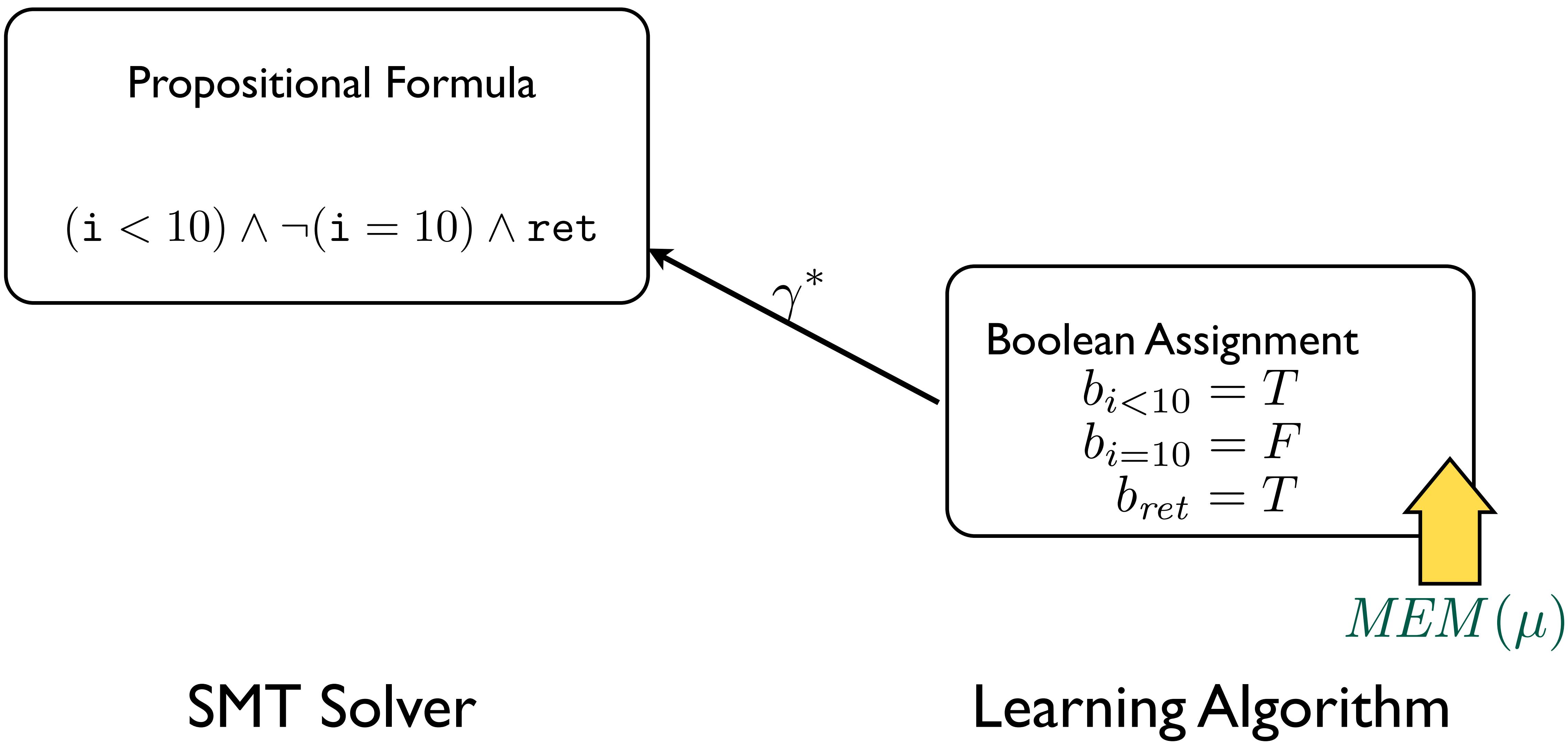
Learning Algorithm

# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions.

$$AP = \{i < 10,\ i = 10,\ ret\}$$

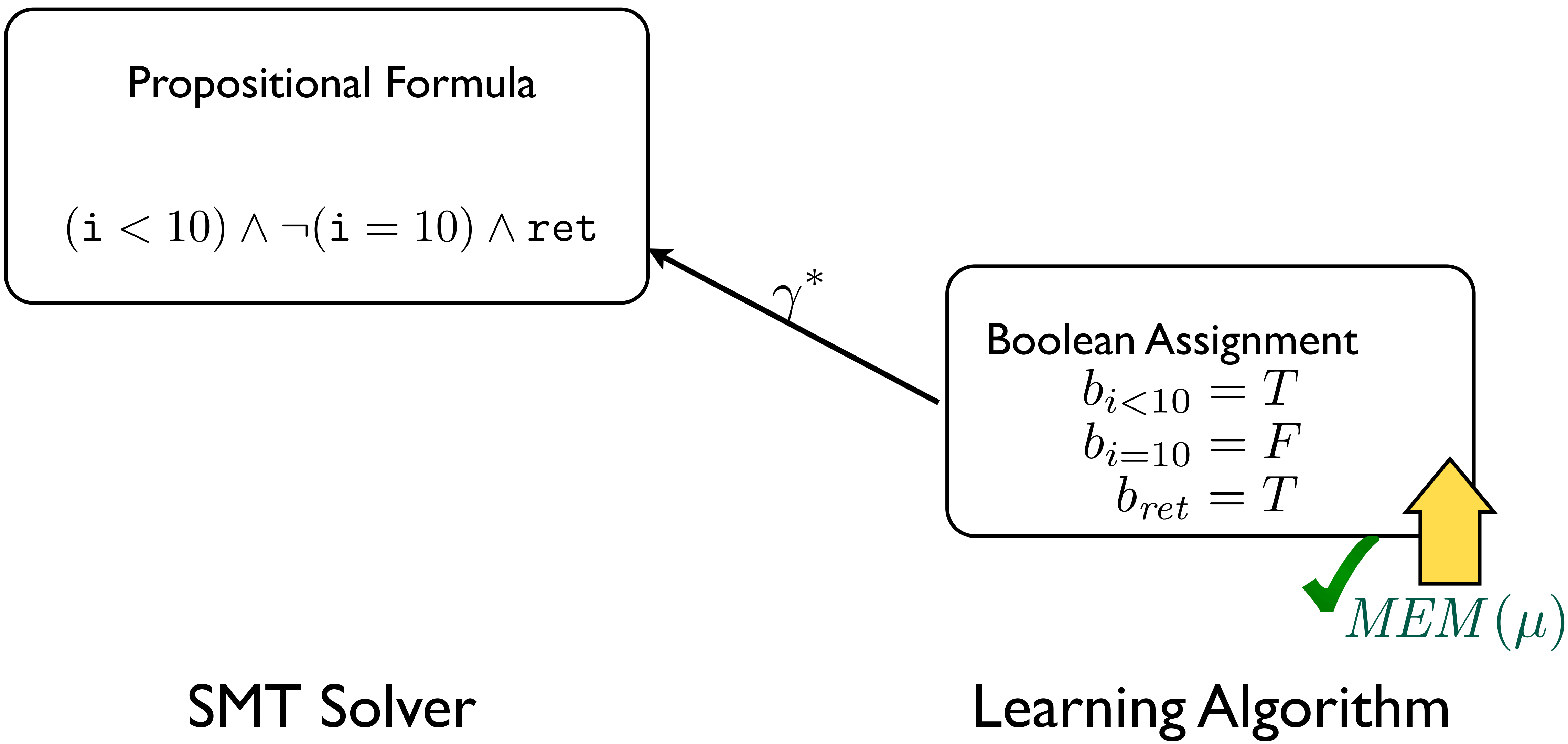


# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions.

$$AP = \{i < 10,\ i = 10,\ ret\}$$

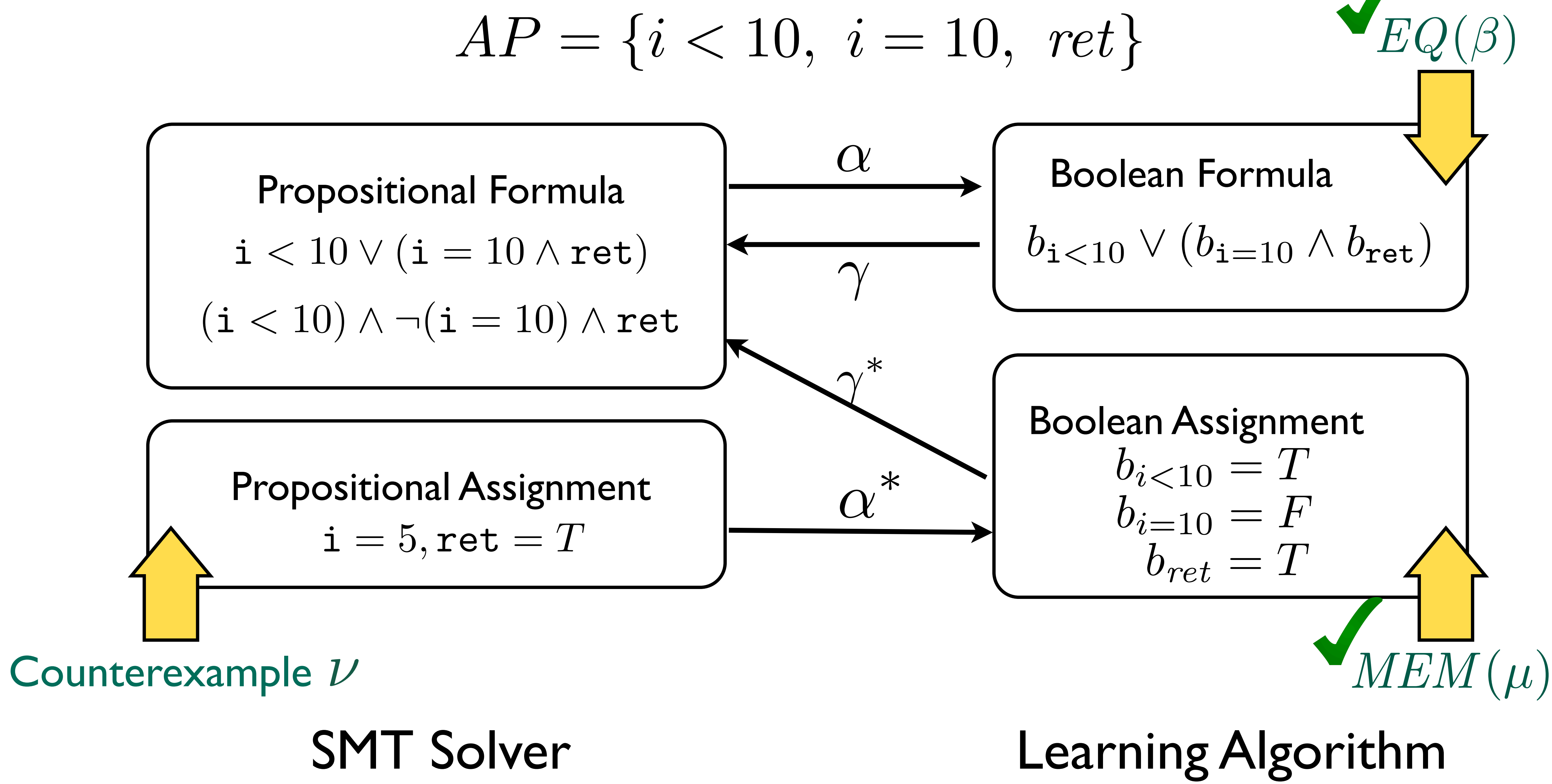


# Predicate Abstraction

Solution:

Use predicate abstraction with given atomic propositions

$$AP = \{i < 10,\ i = 10,\ ret\}$$



# How to Answer Queries

# Problem

The teacher is asked to answer
the question about invariants without knowing invariants..

# Invariant Properties

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \rho\ \texttt{do}\ S\ \texttt{end}\ \{\epsilon\}$$

An Invariant $I$ must satisfy the following conditions:

| | |
|---|---|
| (A) $\delta \Rightarrow \iota$ | ( $\iota$ holds when entering the loop) |
| (B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ | ( $\iota$ holds at each iteration) |
| (C) $\iota \wedge \neg\rho \Rightarrow \epsilon$ | ( $\iota$ gives $\epsilon$ after leaving the loop) |

Observation #1

In $EQ(\beta)$ we can say "YES" by checking three conditions.

# Invariant Properties

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \rho\ \texttt{do}\ S\ \texttt{end}\ \{\epsilon\}$$

An Invariant $I$ must satisfy the following conditions:

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

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ ( $\iota$ holds at each iteration)

(C) $\iota \wedge \neg\rho \Rightarrow \epsilon$ ( $\iota$ gives $\epsilon$ after leaving the loop)

Observation #1
In $EQ(\beta)$ we can say "YES" by checking three conditions.

Observation #2

$$\delta \Rightarrow I \Rightarrow \epsilon \vee \rho$$

strongest under-approximation of an invariant

weakest over-approximation of an invariant

# Equivalence Query Resolution: $EQ(\beta)$

1. "YES", if $\gamma(\beta)$ satisfies invariant conditions.

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

   (B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ ( $\iota$ holds at each iteration)

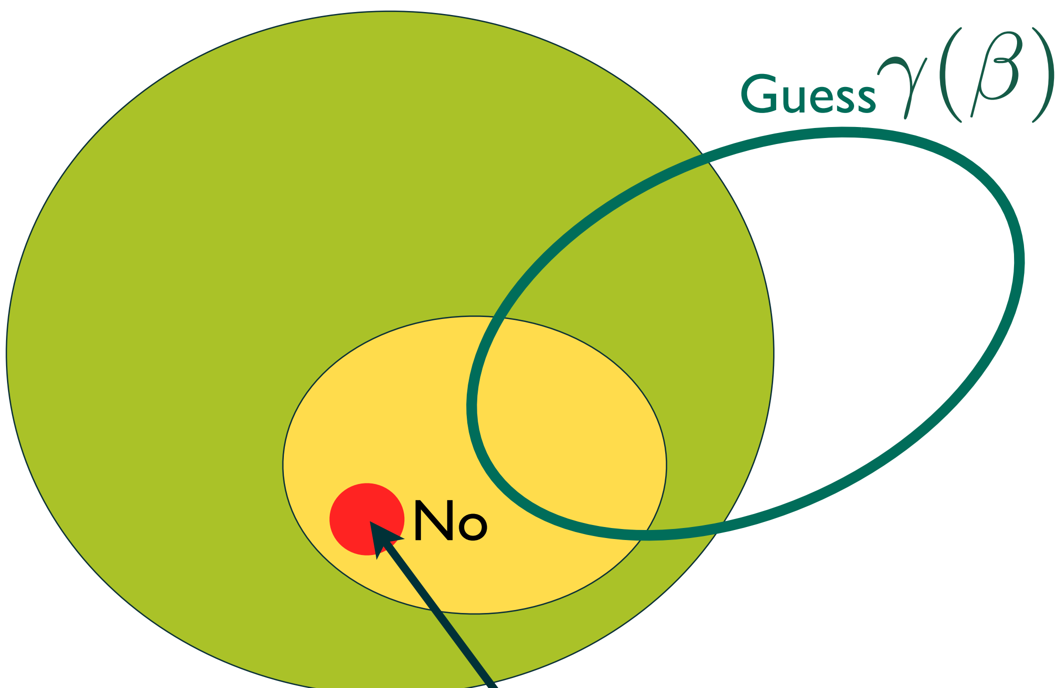
   (C) $\iota \wedge \neg\rho \Rightarrow \epsilon$ ( $\iota$ gives $\epsilon$ after leaving the loop)

   Then, we find an invariant!

# Equivalence Query Resolution: $EQ(\beta)$

1. "YES", if $\gamma(\beta)$ satisfies invariant conditions.
2. Otherwise, we need a counter example to answer "No".

Case 1




No,
with found a counterexample $\alpha^*(\nu)$.

$$\nu \models \gamma(\beta) \oplus \underline{\iota}$$
$$\Rightarrow \nu \models \gamma(\beta) \oplus \iota$$

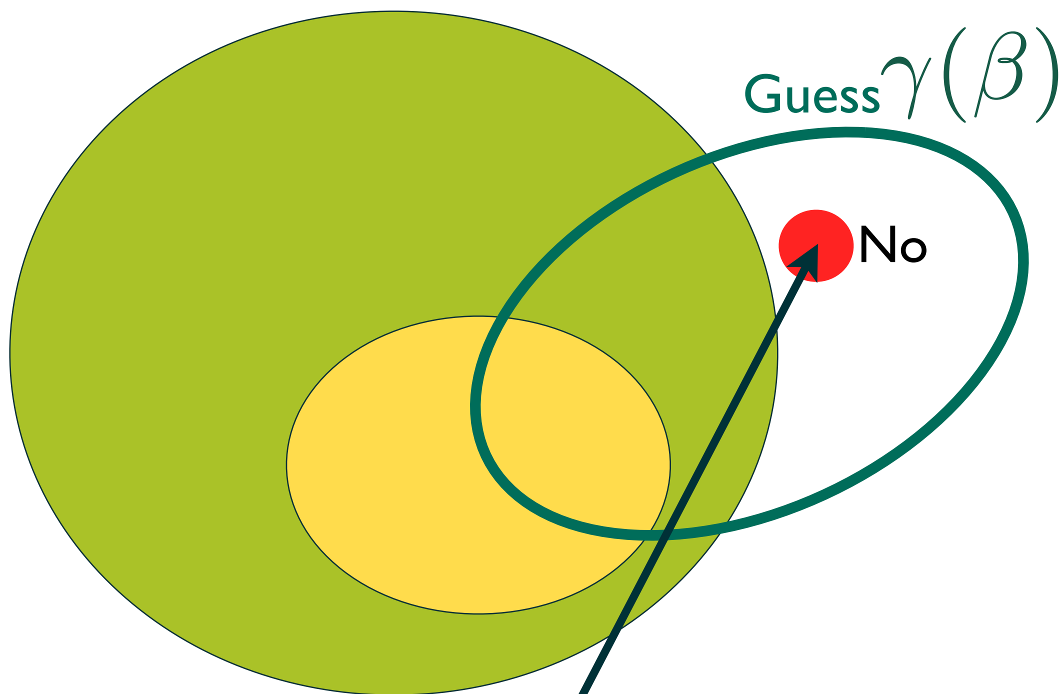
Over Approximation $\overline{\iota}$

Under Approximation $\underline{\iota}$

# Equivalence Query Resolution: $EQ(\beta)$

1. "YES", if $\gamma(\beta)$ satisfies invariant conditions.
2. Otherwise, we need a counter example to answer "No".

Case 2




No,
with found a counterexample $\alpha^*(\nu)$ .

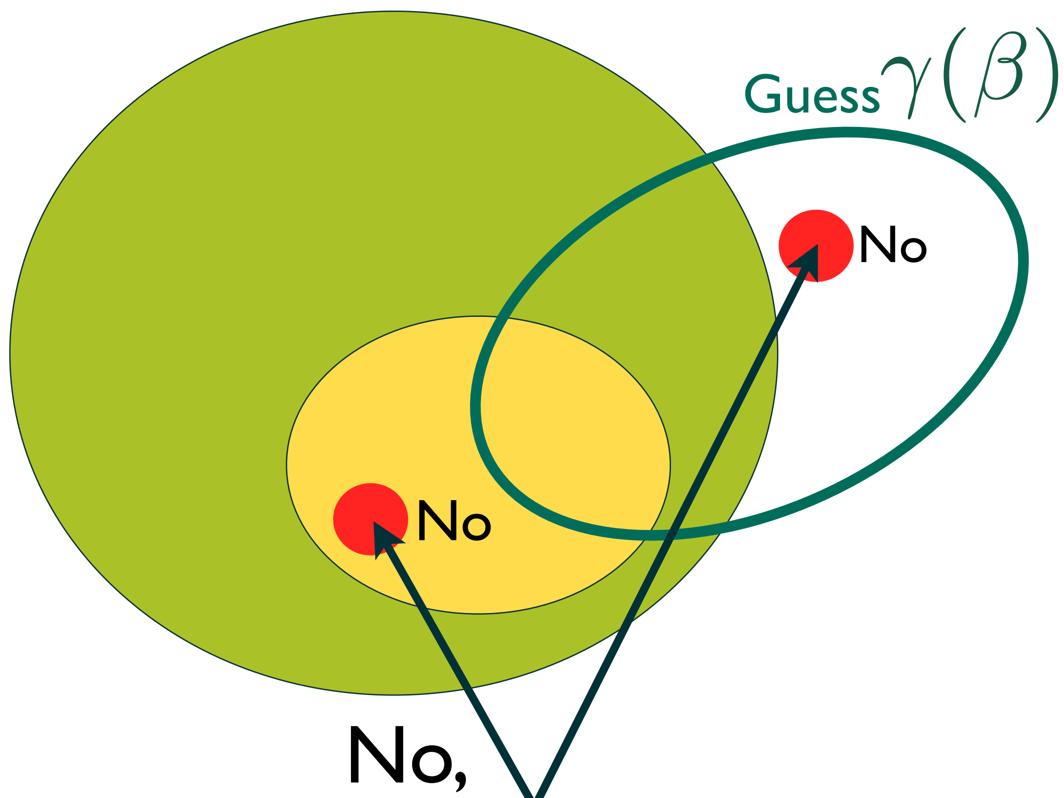
$$\nu \models \gamma(\beta) \oplus \bar{\iota}$$
$$\Rightarrow \nu \models \gamma(\beta) \oplus \iota$$

Over Approximation $\bar{\iota}$
Under Approximation $\underline{\iota}$

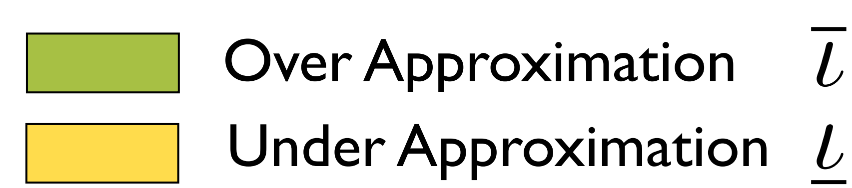
# Equivalence Query Resolution: $EQ(\beta)$

1. "YES", if $\gamma(\beta)$ satisfies invariant conditions.
2. Otherwise, we need a counter example to answer "No".

Case 1 & 2




with found a counterexample $\alpha^*(\nu)$ .

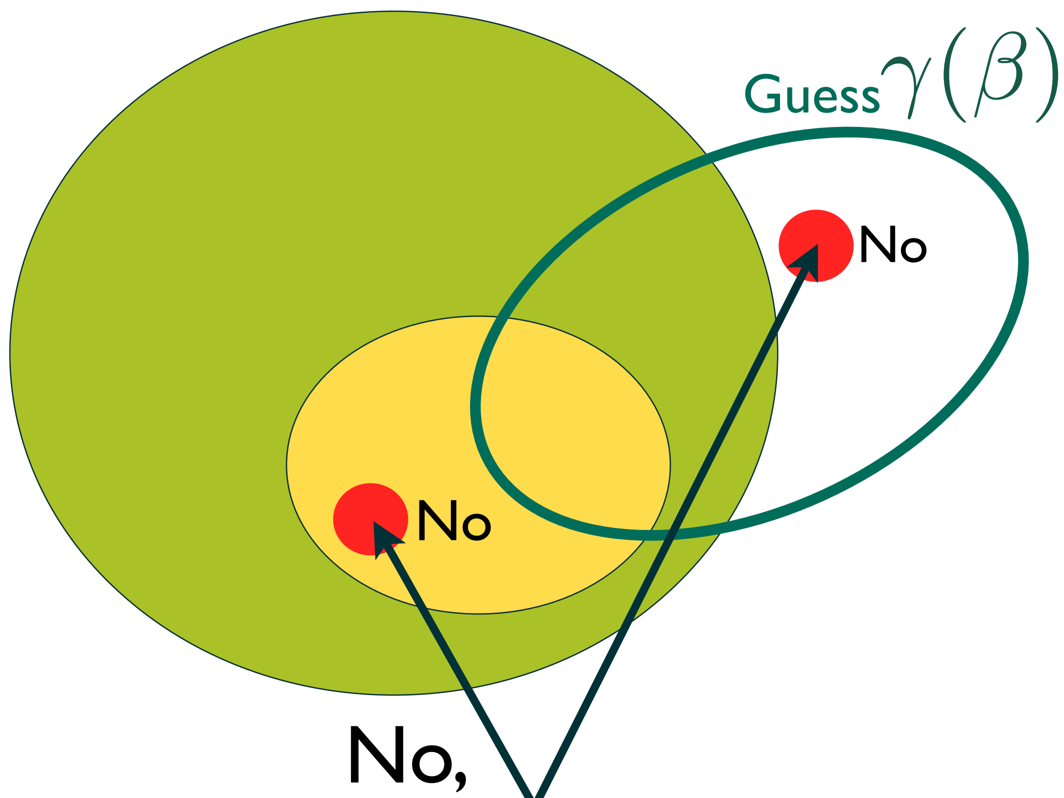


# Equivalence Query Resolution: $EQ(\beta)$

1. "YES", if $\gamma(\beta)$ satisfies invariant conditions.

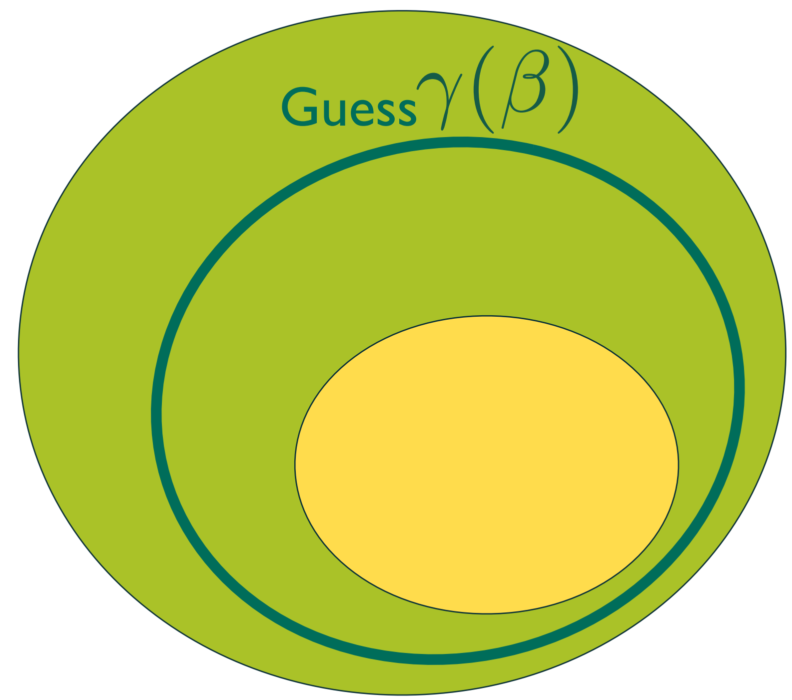
2. Otherwise, we need a counter example to answer "No".

Case 1 & 2



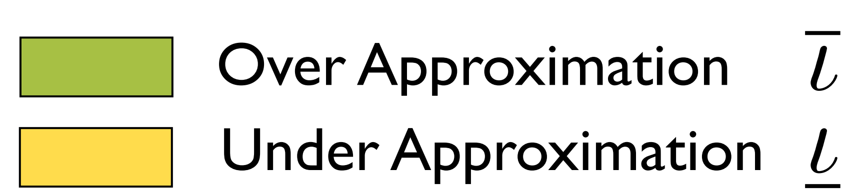

No,
with found a counterexample $\alpha^*(\nu)$.

Case 3




$\underline{\iota} \Rightarrow \gamma(\beta) \Rightarrow \overline{\iota}$

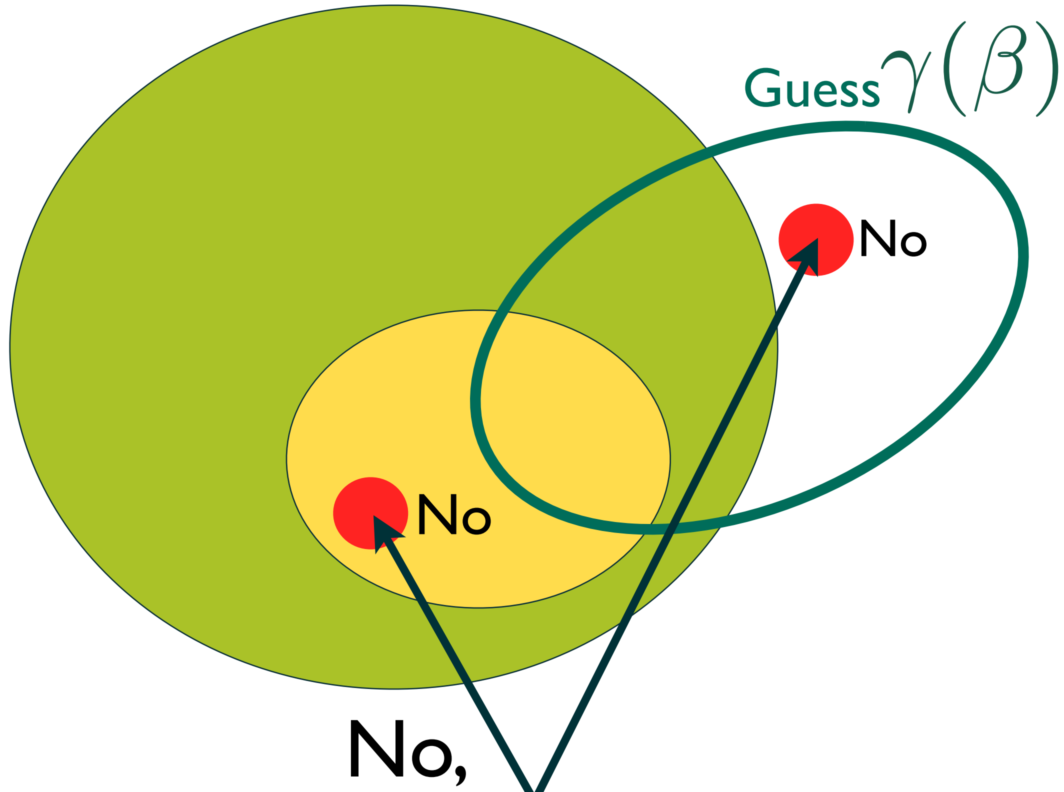
Cannot find a counterexample.



# Equivalence Query Resolution: $EQ(\beta)$

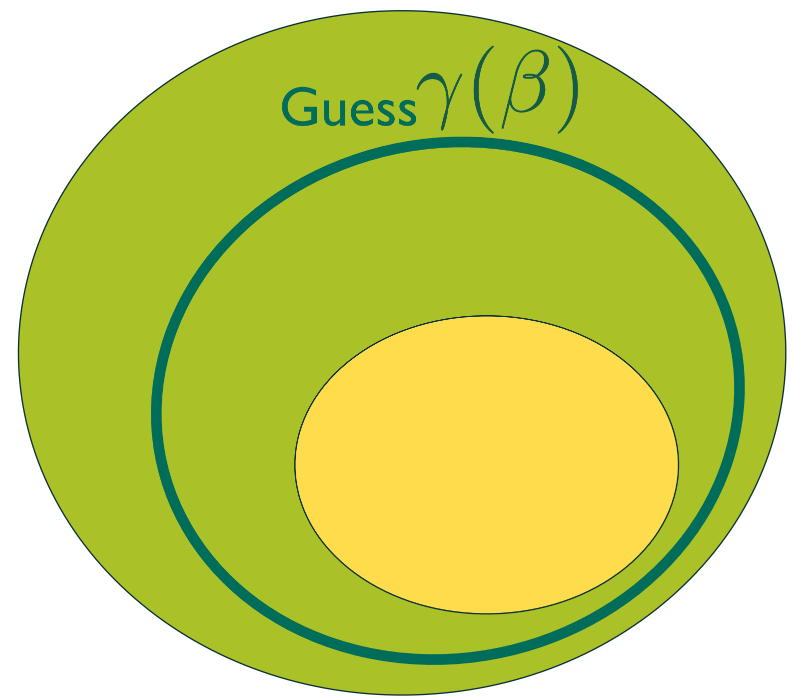
1. "YES", if $\gamma(\beta)$ satisfies invariant conditions.
2. Otherwise, we need a counter example to answer "No".

Case 1 & 2




No,
with found a counterexample $\alpha^*(\nu)$ .

Case 3




Restart the learning algorithm!
Cannot find a counterexample.

Over Approximation $\overline{\iota}$

Under Approximation $\underline{\iota}$

# Membership Query Resolution: $MEM(\mu)$

1. "NO", if $\gamma^*(\mu)$ is unsatisfiable.

$$\mu = \{b_{i=0} = T,\ b_{i<10} = F\}$$

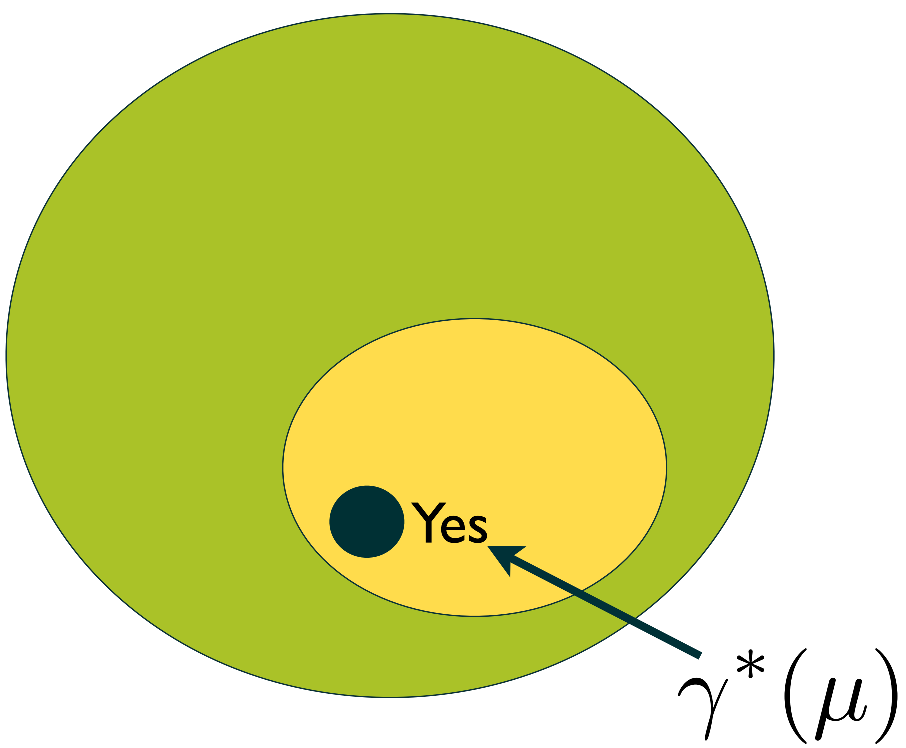
$$\gamma^*(\mu) = (i = 0 \wedge \neg(i < 10))$$

# Membership Query Resolution: $MEM(\mu)$

1. "NO", if $\gamma^*(\mu)$ is unsatisfiable.

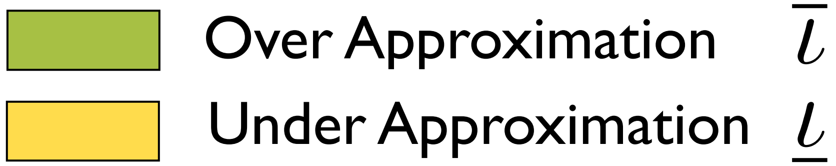
2. Use approximations to answer the query.

Case 1




Answer Yes

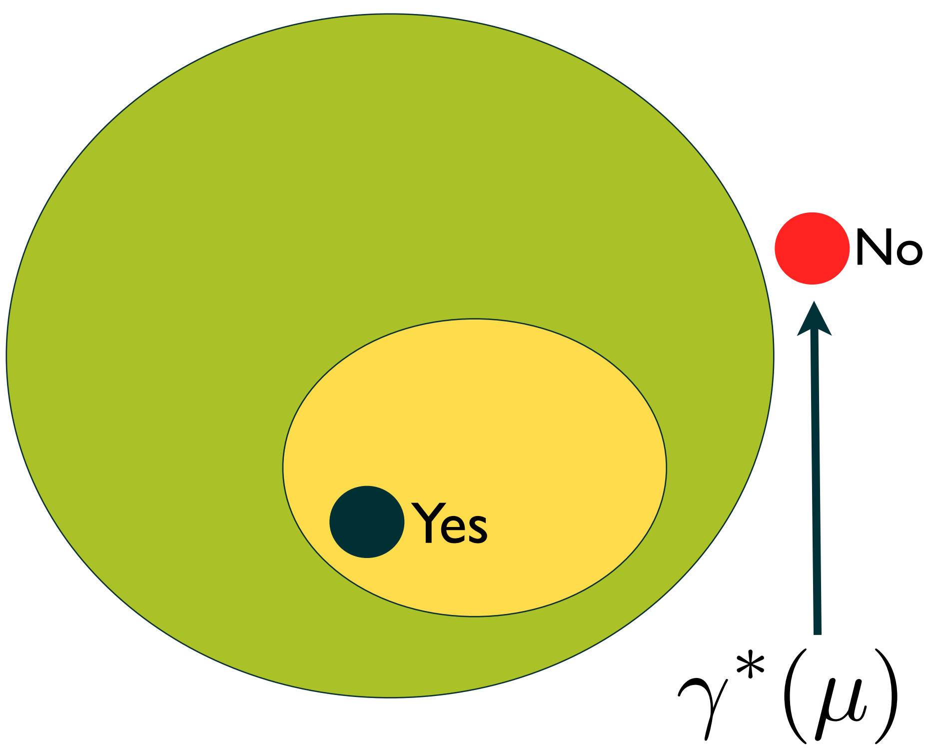
$$\gamma^*(\mu) \Rightarrow \underline{\iota}$$
$$\gamma^*(\mu) \Rightarrow \iota$$



# Membership Query Resolution: $MEM(\mu)$

1. "NO", if $\gamma^*(\mu)$ is unsatisfiable.
2. Use approximations to answer the query.

Case 2




Answer No

$$\gamma^*(\mu) \not\Rightarrow \bar{\iota}$$
$$\gamma^*(\mu) \not\Rightarrow \iota$$

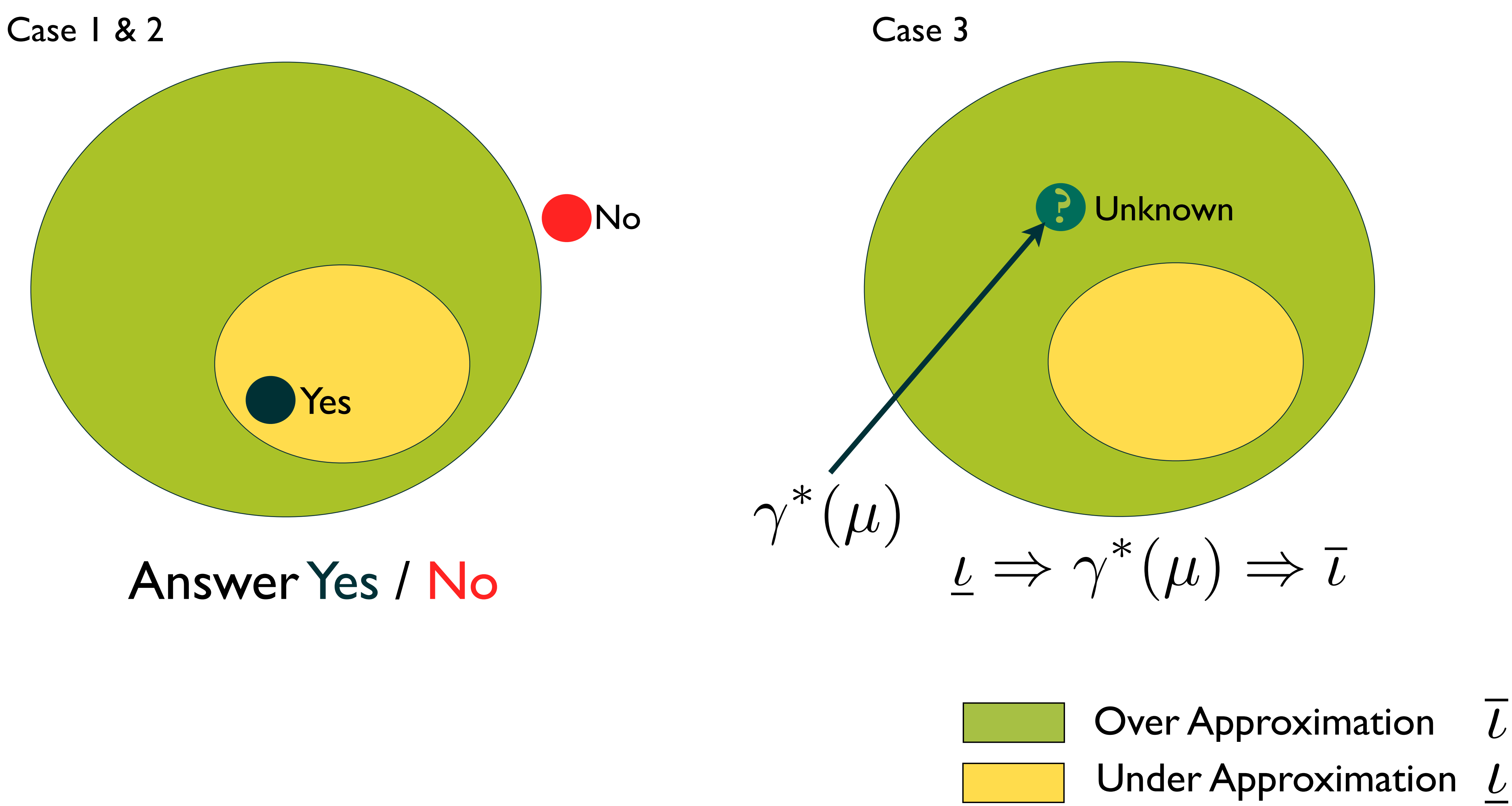
Over Approximation $\bar{\iota}$

Under Approximation $\underline{\iota}$

# Membership Query Resolution: $MEM(\mu)$

1. "NO", if $\gamma^*(\mu)$ is unsatisfiable.

2. Use approximations to answer the query.

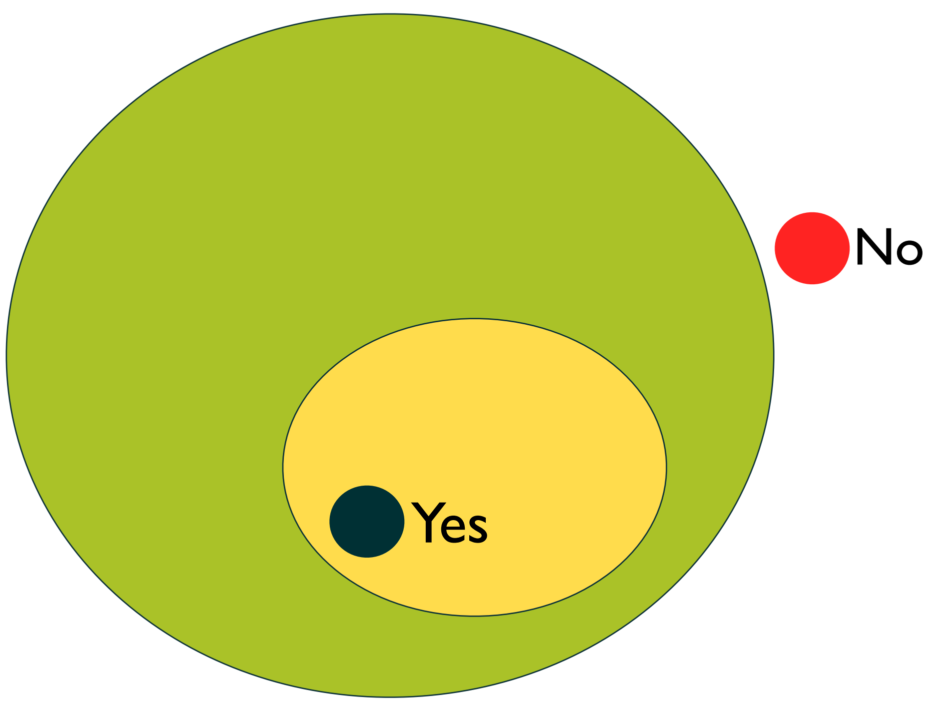


# Membership Query Resolution: $MEM(\mu)$

1. "NO", if $\gamma^*(\mu)$ is unsatisfiable.

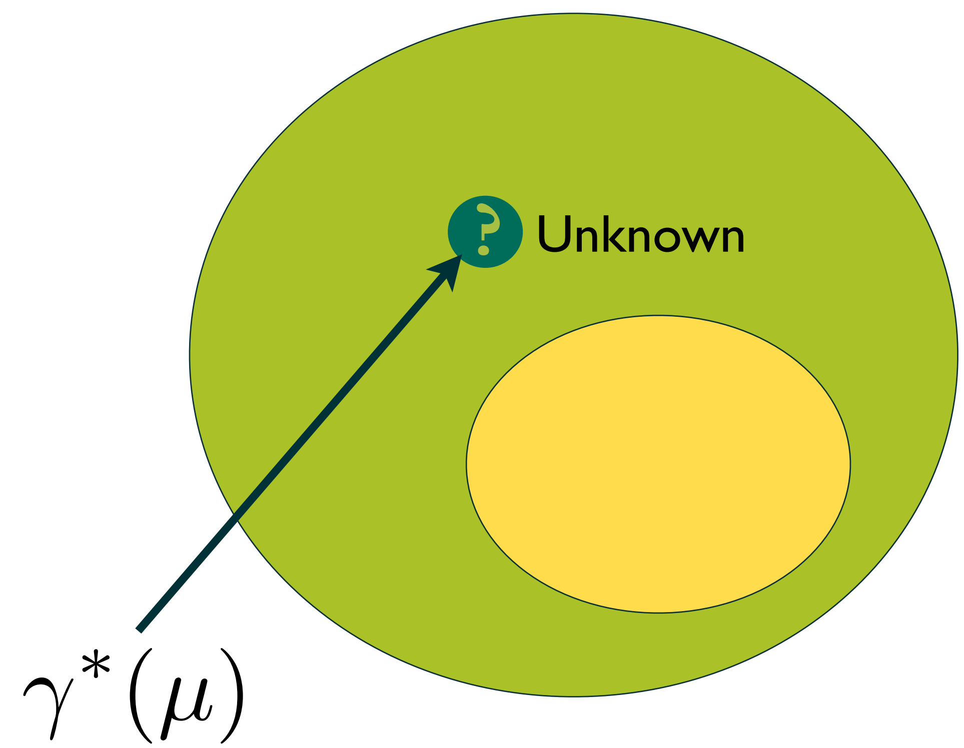
2. Use approximations to answer the query.

Case 1 & 2




Answer Yes / No

Case 3




Random Answer!

# It's still Sound

Why?

When resolving equivalence query $EQ(\beta)$

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

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

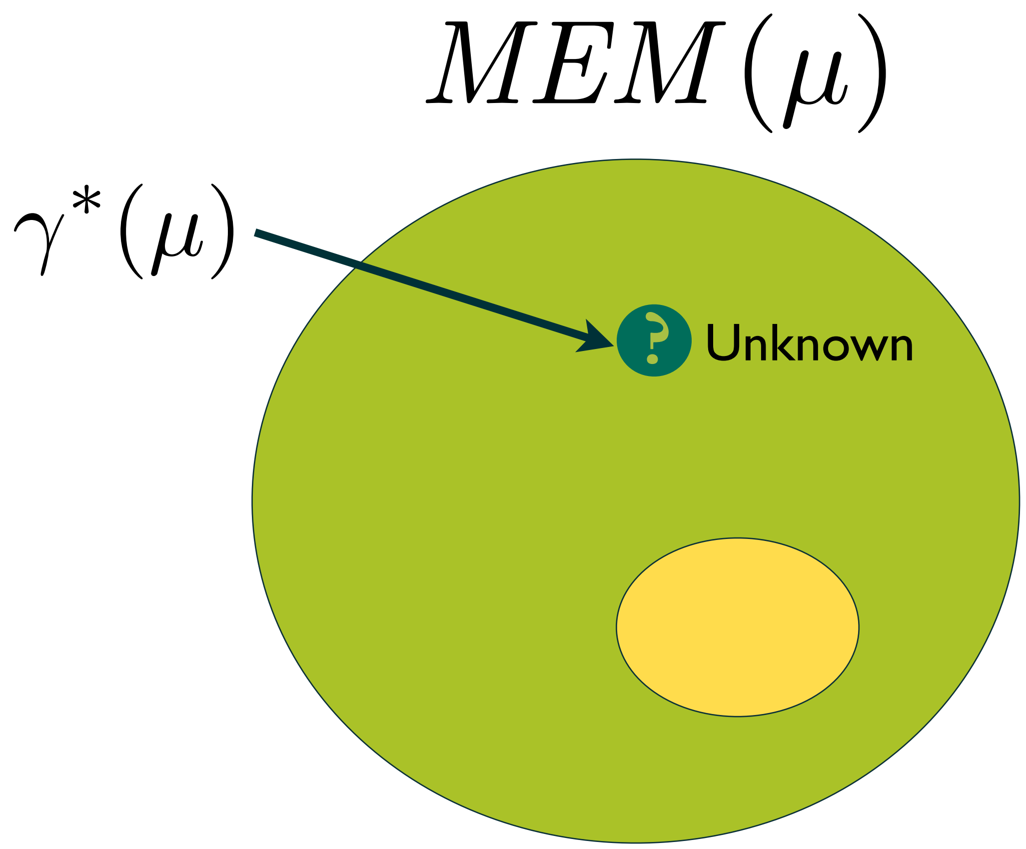
(C) $I \wedge \neg\rho \Rightarrow \epsilon$ ( $I$ gives $\epsilon$ after leaving the loop)

1. "YES", if $\gamma(\beta)$ satisfies invariant conditions.

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

An SMT solver is **sound** and **complete** for **propositional** formulae.

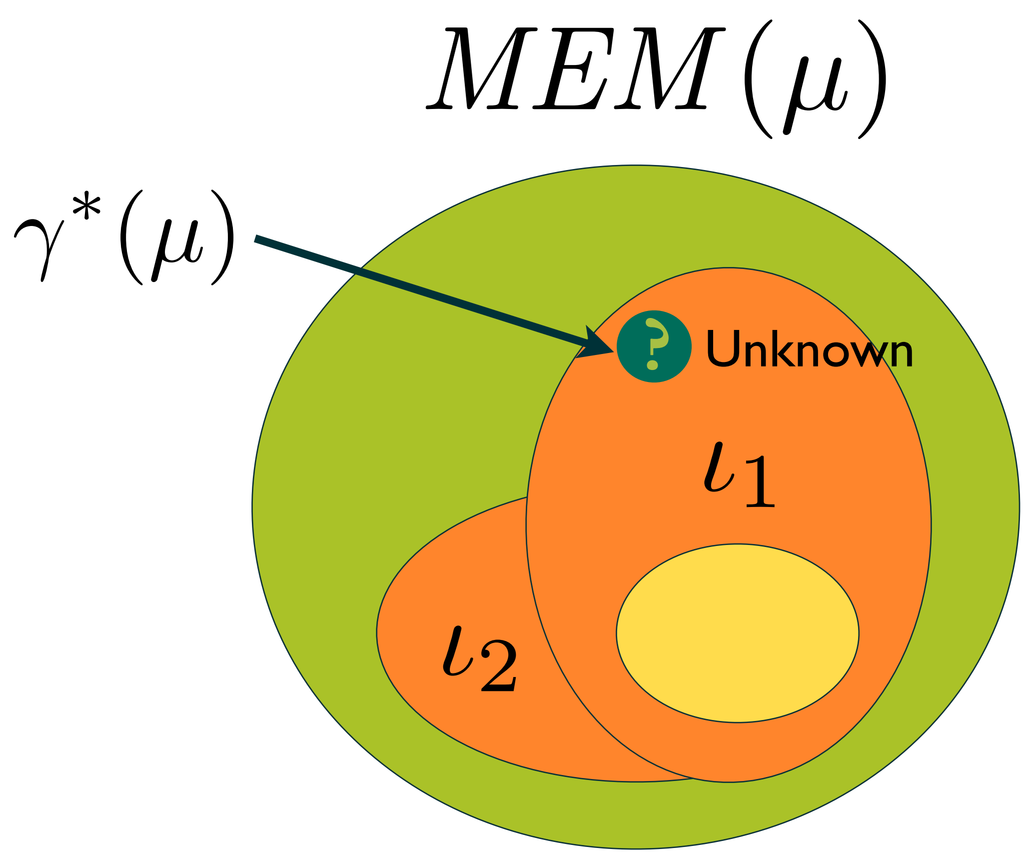
# Effect of Random Answer



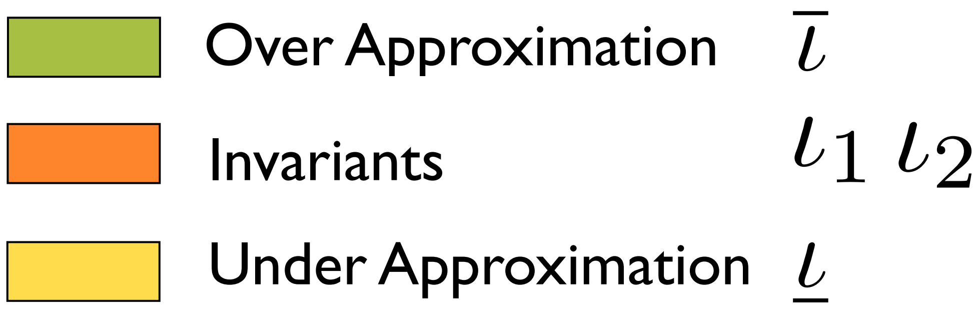

Over Approximation $\overline{\iota}$

Under Approximation $\underline{\iota}$

# Effect of Random Answer

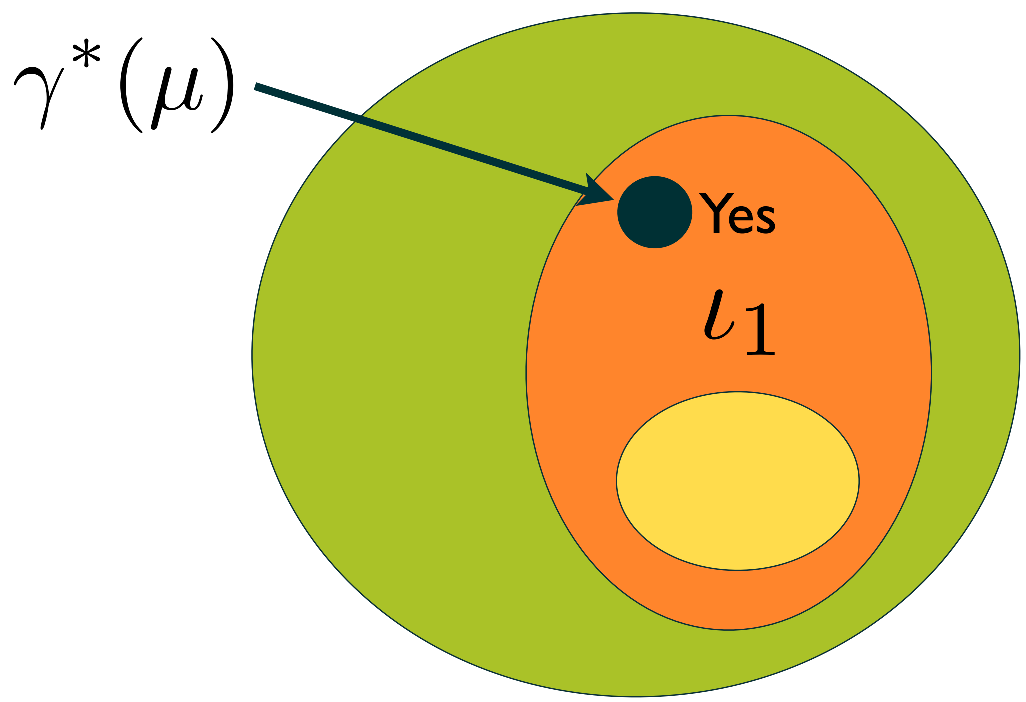



Both of the random answers can lead to an invariant.



# Effect of Random Answer

$$MEM(\mu)$$




$$MEM(\mu) = YES$$

**leads to** $\iota_1$

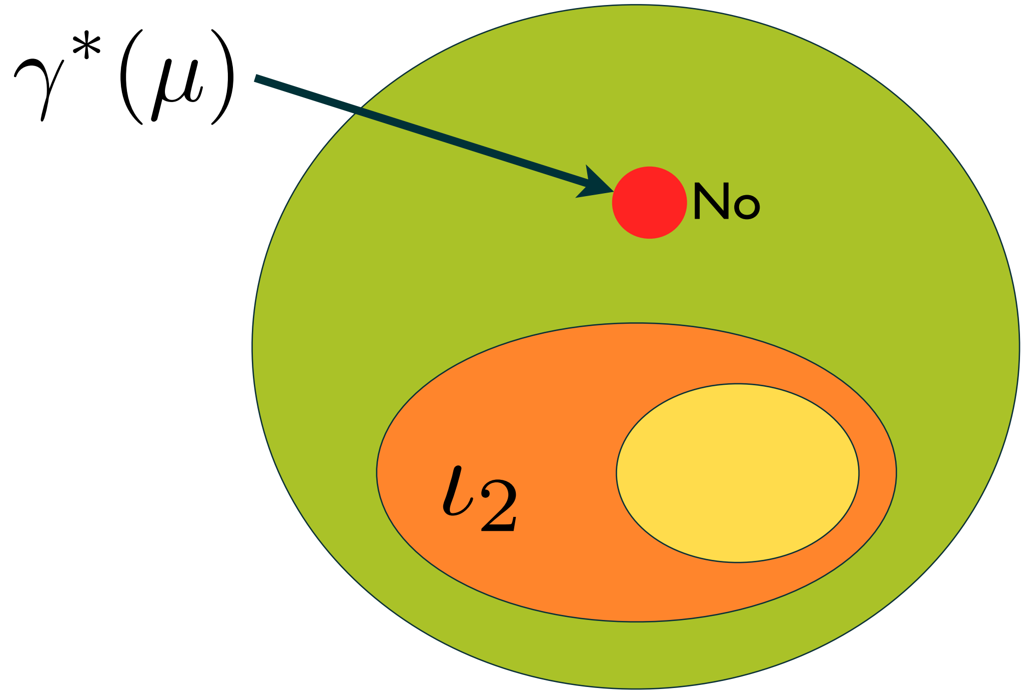
Over Approximation $\overline{\iota}$

Invariants $\iota_1$ $\iota_2$

Under Approximation $\underline{\iota}$

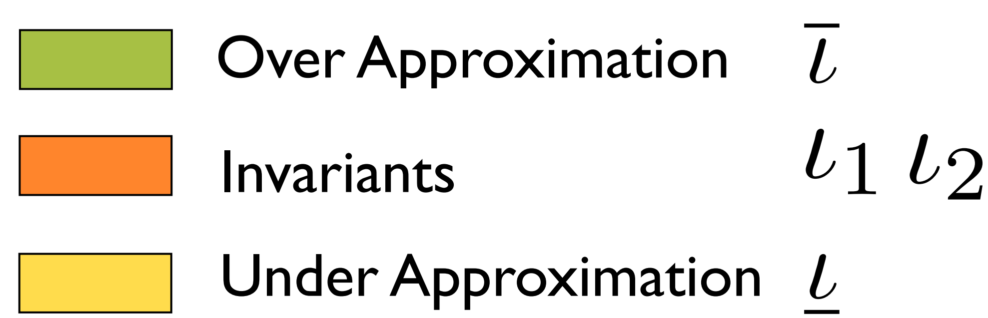
# Effect of Random Answer

$MEM(\mu)$




$MEM(\mu) = NO$

leads to $\iota_2$



# Invariants are Not Unique

$\{\mathtt{locked} \wedge \mathtt{i} = 0\}$

```
while i < entries && status = 0 do
  retval := nondet
  locked := false;
  switch retval do
    case ENXIO: retval := 0;
    case EAGAIN: retval := 0;
    case ENOMEN: retval := 0;
    case 0: i := i + 1;
  end
  locked := true;
  if retval != 0 && (status = 0 || status = ECONNRESET) then
    status := retval;
end
```

$\{\mathtt{locked} \wedge (\mathtt{i} \neq 0 \Rightarrow \mathtt{status} = \mathtt{retval})\}$

1. $((\mathtt{io\_status} = \mathtt{retval} \wedge \mathtt{io\_status} = 0) \vee (\mathtt{io\_status} = \mathtt{retval} \wedge \mathtt{retval} \neq 0 \wedge \mathtt{io\_lock}) \vee (\mathtt{i} = 0 \wedge \mathtt{io\_lock})) \wedge ((\mathtt{i} < \mathtt{entries}) \vee \mathtt{io\_lock})$
2. $\mathtt{io\_lock} \wedge (\mathtt{i} = 0 \vee (\mathtt{io\_status} = 0 \wedge \mathtt{io\_status} = \mathtt{retval}) \vee (\mathtt{retval} \neq 0 \wedge \mathtt{io\_status} = \mathtt{retval}))$
3. $\mathtt{io\_lock} \wedge ((\mathtt{io\_status} = 0 \wedge \mathtt{retval} \neq 0 \wedge \mathtt{i} = 0) \vee (\mathtt{io\_status} = \mathtt{retval} \wedge \mathtt{retval} \neq 0) \vee (\mathtt{io\_status} = \mathtt{retval} \wedge \mathtt{io\_status} = 0) \vee \mathtt{i} = 0)$

...

# Experiments

Performance Table

| Case | AP | MEM | EQ | Coin Toss | Restarts | Time(sec) |
|---|---|---|---|---|---|---|
| ide-ide-tape | 6 | 18.2 | 5.2 | 4.1 | 1.2 | 0.055 |
| ide-wait-ireason | 6 | 216.1 | 111.8 | 47.2 | 9.9 | 0.602 |
| parser | 20 | 6694.5 | 819.4 | 990.3 | 12.5 | 32.120 |
| usb-message | 10 | 20.1 | 6.8 | 1.0 | 1.0 | 0.128 |
| vpr | 7 | 14.5 | 8.9 | 11.8 | 2.9 | 0.055 |

# Experiments

Performance Table

| Case | AP | MEM | EQ | Coin Toss | Restarts | Time(sec) |
|---|---|---|---|---|---|---|
| ide-ide-tape | 6 | 18.2 | 5.2 | 4.1 | 1.2 | 0.055 |
| ide-wait-ireason | 6 | 216.1 | 111.8 | 47.2 | 9.9 | 0.602 |
| parser | 20 | 6694.5 | 819.4 | 990.3 | 12.5 | 32.120 |
| usb-message | 10 | 20.1 | 6.8 | 1.0 | 1.0 | 0.128 |
| vpr | 7 | 14.5 | 8.9 | 11.8 | 2.9 | 0.055 |

1. Search Space is Huge, naive search shouldn't work

$$\#AP = 20 \Rightarrow |S| = 2^{2^{20}}$$

# Experiments

Performance Table

| Case | AP | MEM | EQ | Coin Toss | Restarts | Time(sec) |
|---|---|---|---|---|---|---|
| ide-ide-tape | 6 | 18.2 | 5.2 | 4.1 | 1.2 | 0.055 |
| ide-wait-ireason | 6 | 216.1 | 111.8 | 47.2 | 9.9 | 0.602 |
| parser | 20 | 6694.5 | 819.4 | 990.3 | 12.5 | 32.120 |
| usb-message | 10 | 20.1 | 6.8 | 1.0 | 1.0 | 0.128 |
| vpr | 7 | 14.5 | 8.9 | 11.8 | 2.9 | 0.055 |

1. Search Space is Huge, naive search shouldn't work

$$\#AP = 20 \Rightarrow |S| = 2^{2^{20}}$$

2. Multitude of Invariants

990.3 / 12.5 = 79.5 Coin Toss / Restarts

If there is only one "the invariant",
then we need $2^{79.5}$ restarts.

# Conclusion

- Algorithmic Learning + Decision Procedures + Predicate Abstraction
  => Invariant Generation Technique
- Works in realistic settings (Linux device drivers and SPEC 200 Benchmarks)
- Exploits the flexibility in invariants by randomized mechanism.
- Static/Dynamic Analysis can help.
  More accurate approximations reduce number of restarts(EQ) and random answers(MEM).

Thank You