Deriving Quantified Invariants Using Algorithmic Learning*

Soonho Kong
soon@ropas.snu.ac.kr

18 Dec 2009
ROPAS Show & Tell

* Joint work with Yungbum, Cristina, Prof. Wang, and Prof. Yi

Invariants

Invariants

Problem : Find an invariant I for the Loop

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

Invariants

Problem : Find an invariant I for the Loop

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

Invariant must satisfy the following conditions:

Invariants

Problem : Find an invariant I for the Loop

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

Invariant must satisfy the following conditions:

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

Invariants

Problem : Find an invariant I for the Loop

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

Invariant must satisfy the following conditions:

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

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

precondition that guarantees I after execution of the statement S

Invariants

Problem : Find an invariant I for the Loop

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

Invariant must satisfy the following conditions:

(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 ϵ after leaving the loop)

Invariants

Problem : Find an invariant I for the Loop

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

Invariant must satisfy the following conditions:

(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 ϵ after leaving the loop)

Note that we have $(I \wedge \neg\rho \Rightarrow \epsilon) \equiv (I \Rightarrow \epsilon \vee \rho)$

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

under approximation of an invariant

over approximation of an invariant

The CDNF Algorithm

The CDNF Algorithm

Nader Bshouty

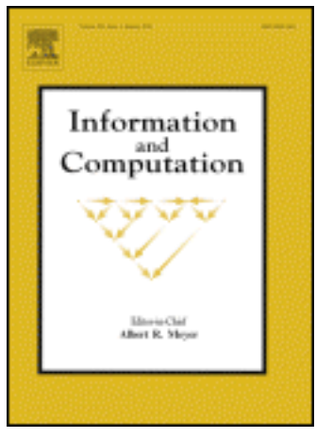
The CDNF Algorithm

Nader Bshouty

UNIVERSITY OF CALGARY

The CDNF Algorithm

Nader Bshouty



Journal of Information and Computation
1995

The CDNF Algorithm

Nader Bshouty

Exact Learning Boolean Functions via the Monotone Theory

Nader H. Bshouty*

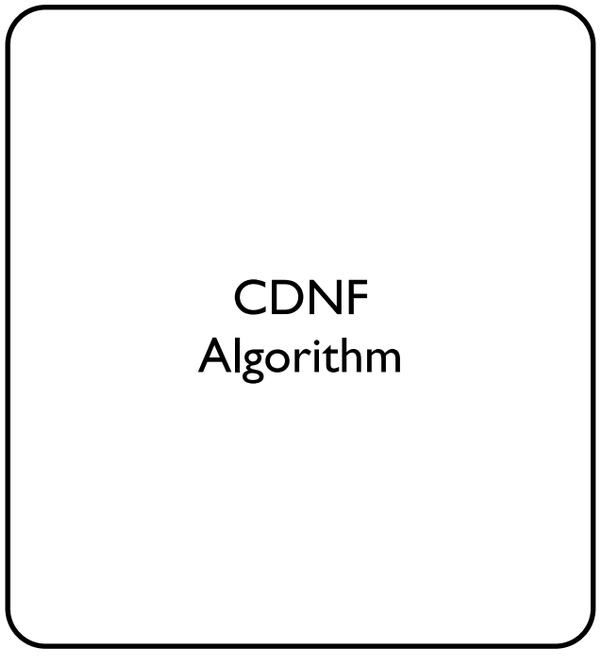
Department of Computer Science

The University of Calgary

Calgary, Alberta, Canada T2N 1N4

e-mail: bshouty@cpsc.ucalgary.ca

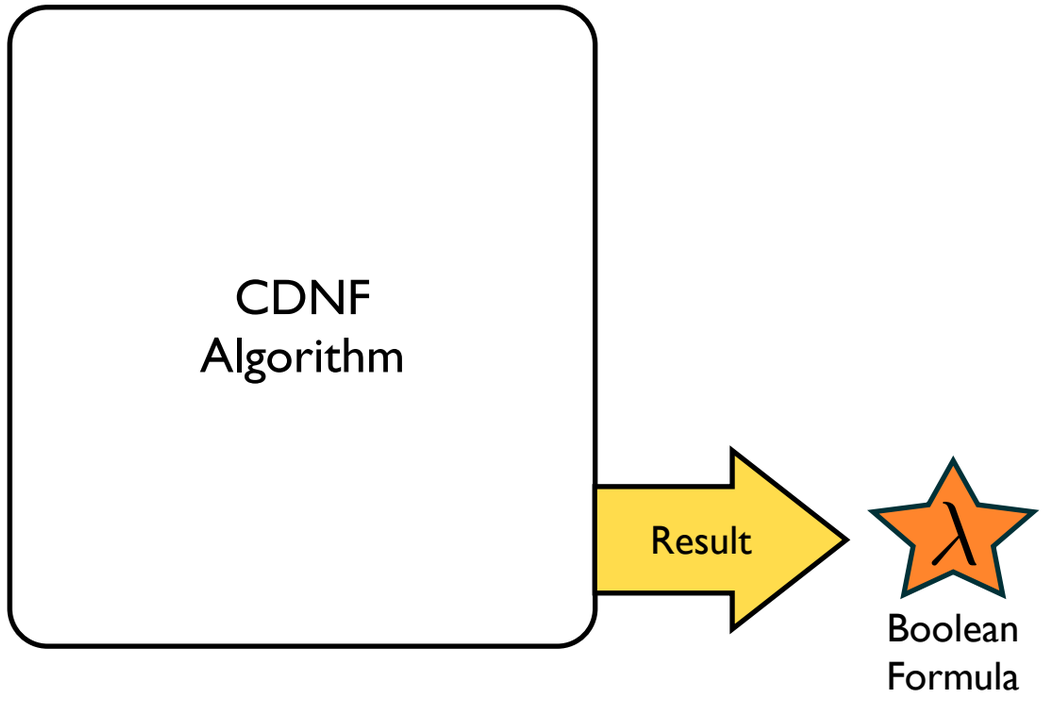
The CDNF Algorithm



Learning

Actively a Boolean formula
from membership and equivalence queries
in polynomial time

The CDNF Algorithm

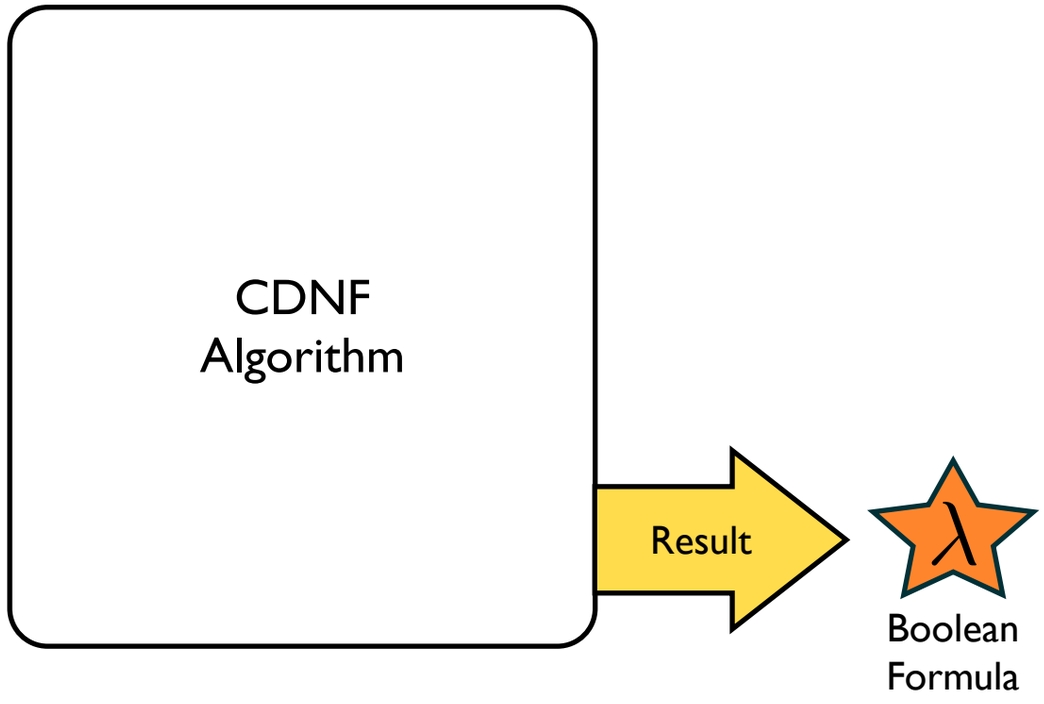


Actively Learning a Boolean formula

from membership and equivalence queries

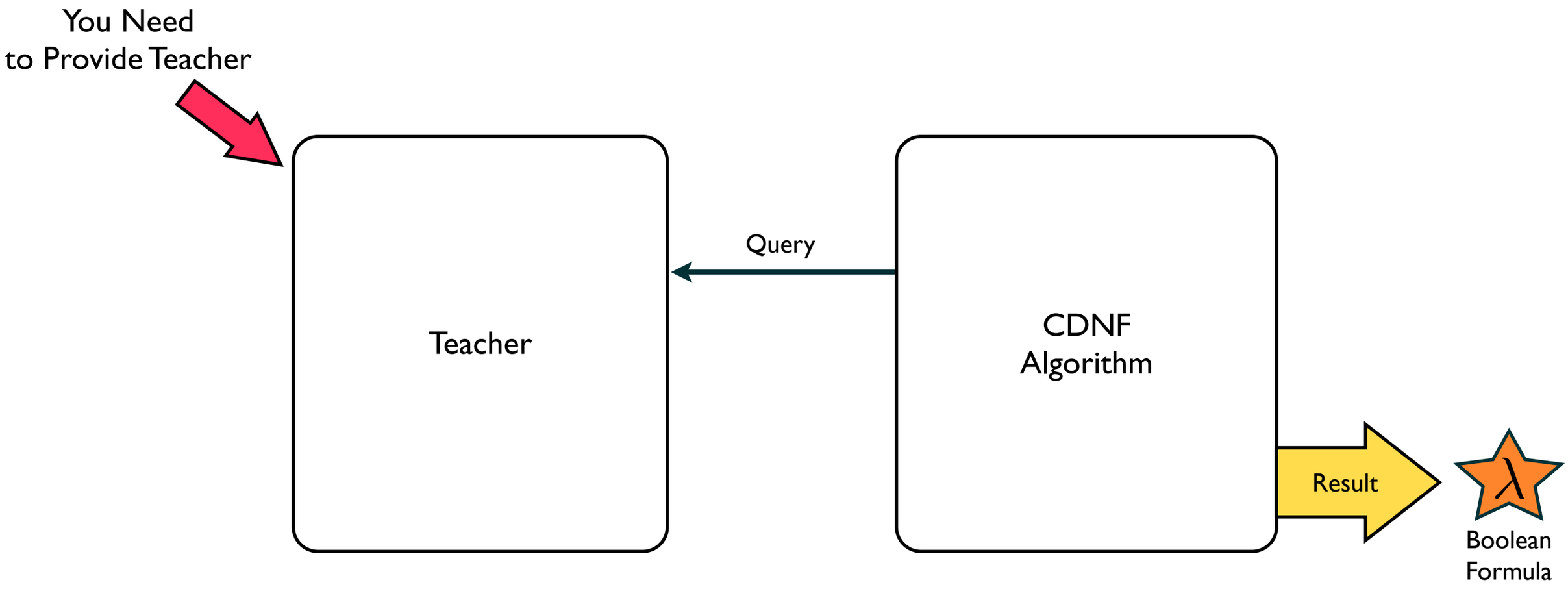
in polynomial time

The CDNF Algorithm



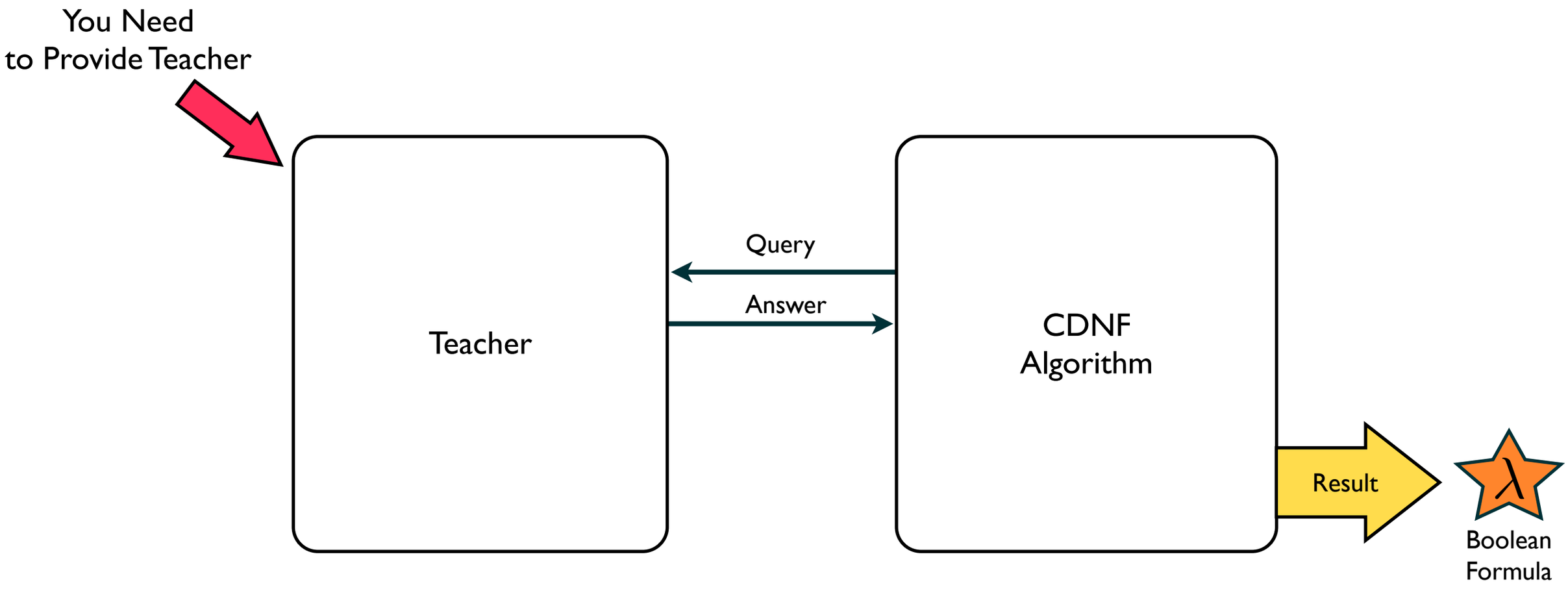
Actively Learning a Boolean formula

The CDNF Algorithm



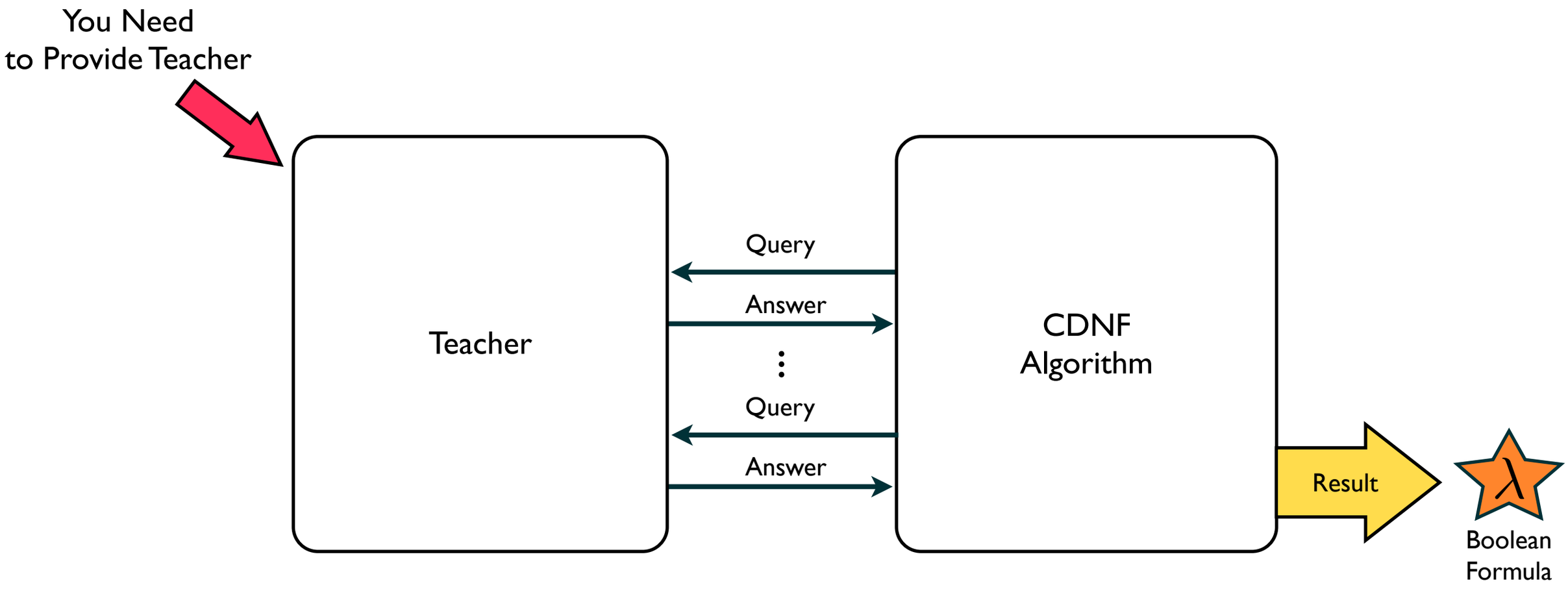
Actively Learning a Boolean formula
from membership and equivalence queries

The CDNF Algorithm



Actively Learning a Boolean formula
from membership and equivalence queries

The CDNF Algorithm



Actively Learning a Boolean formula
from membership and equivalence queries
in polynomial time

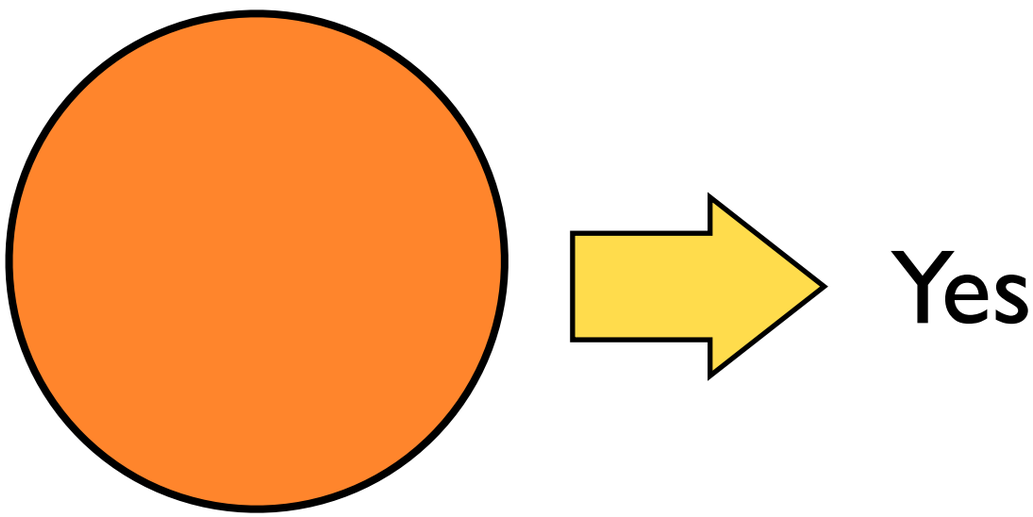
Two Types of Query

$$\mu \overset{?}{\models} \lambda$$

1. Membership Query $MEM(\mu)$ asks if the truth assignment μ satisfies λ.

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

Two Types of Query

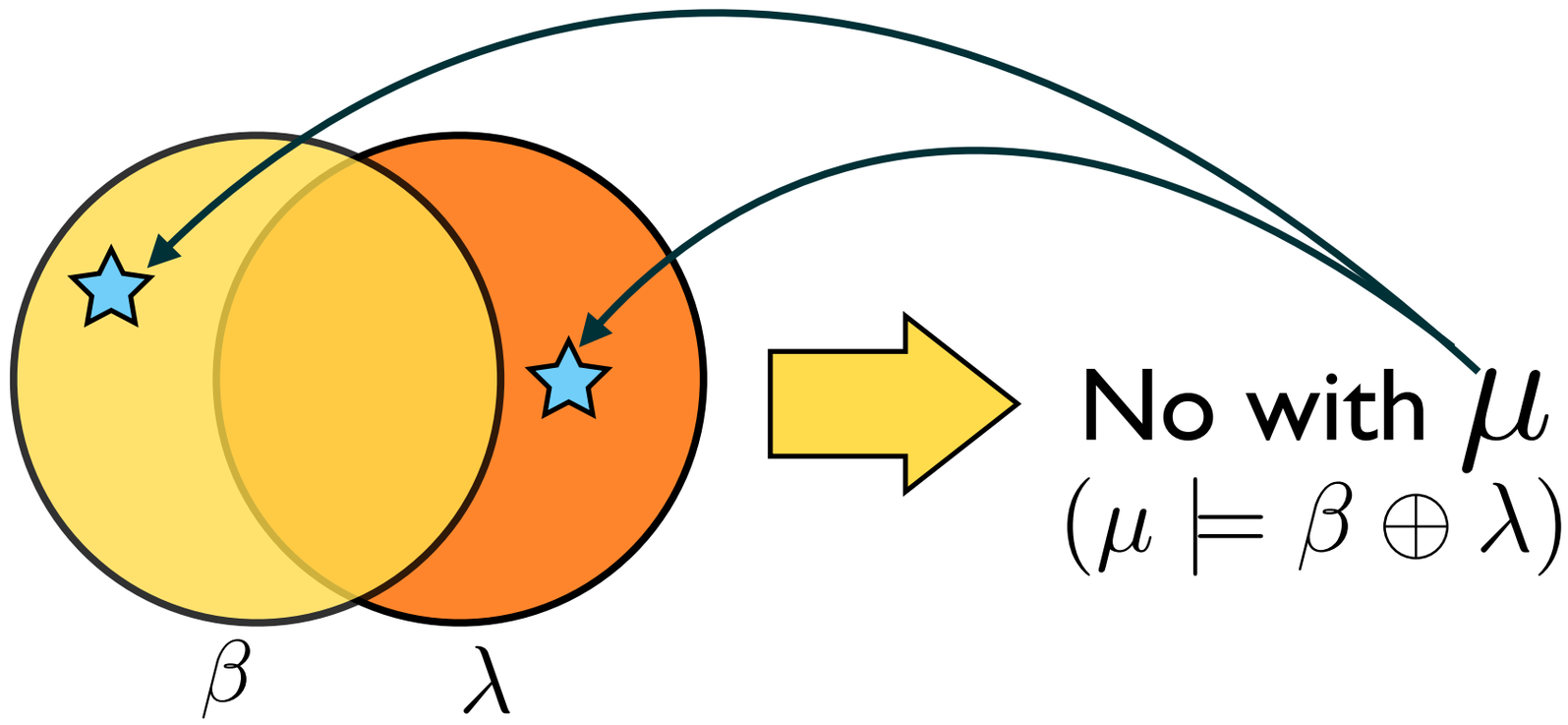


$$\beta \equiv \lambda$$

2. Equivalence Query $EQ(\beta)$ asks
   if the Boolean formula β is equivalent to λ,
   If not, the teacher returns a truth assignment
   as a counterexample γ.

$$EQ(\beta) = Yes \qquad \text{if } \beta \equiv \lambda$$

Two Types of Query



2. Equivalence Query $EQ(\beta)$ asks
   if the Boolean formula β is equivalent to λ,
   If not, the teacher returns a truth assignment
   as a counterexample μ.

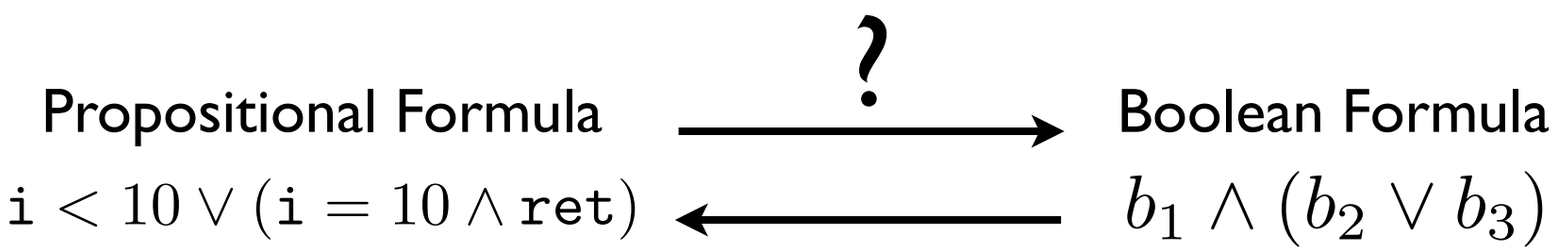
$$EQ(\beta) = Yes \qquad \text{if } \beta \equiv \lambda$$
$$EQ(\beta) = No \text{ with } \mu \qquad \text{if } \beta \not\equiv \lambda \wedge (\mu \models \beta \oplus \lambda)$$

Deriving Propositional Invariants using Algorithmic Learning

Predicate Abstraction

Problem:

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

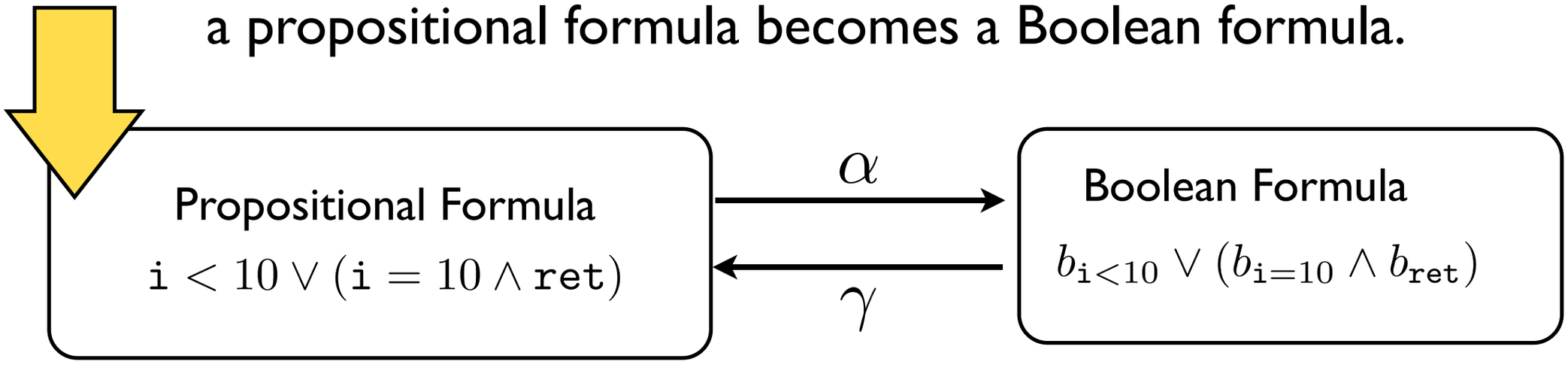


Predicate Abstraction

Solution:

Use predicate abstraction!
If we see atomic propositions as Boolean variables,
a propositional formula becomes a Boolean formula.

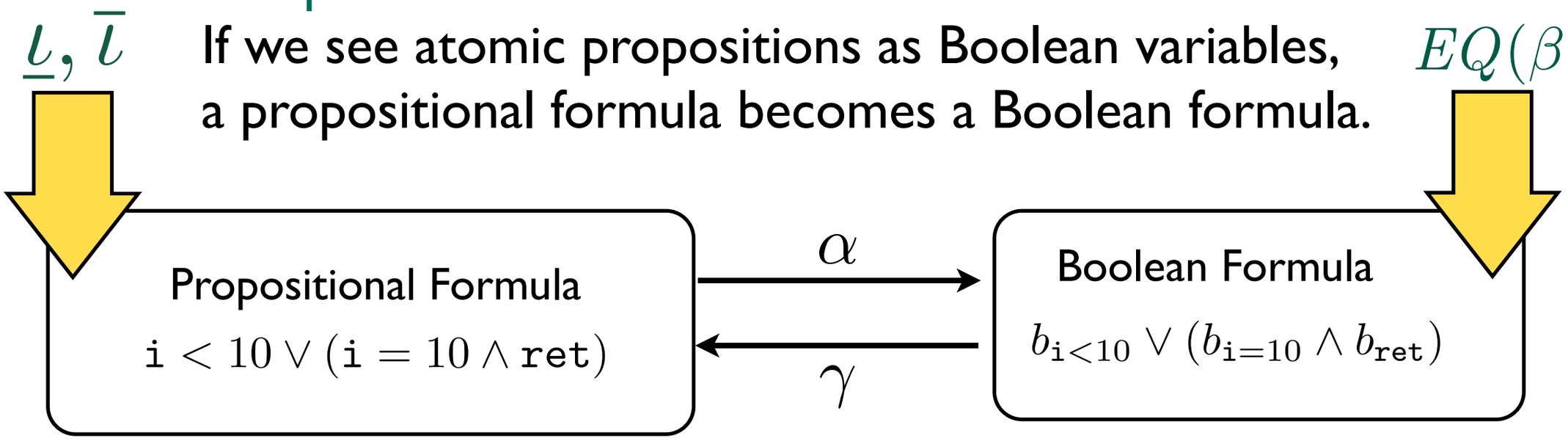
$\underline{\iota}, \overline{\iota}$



Predicate Abstraction

Solution:

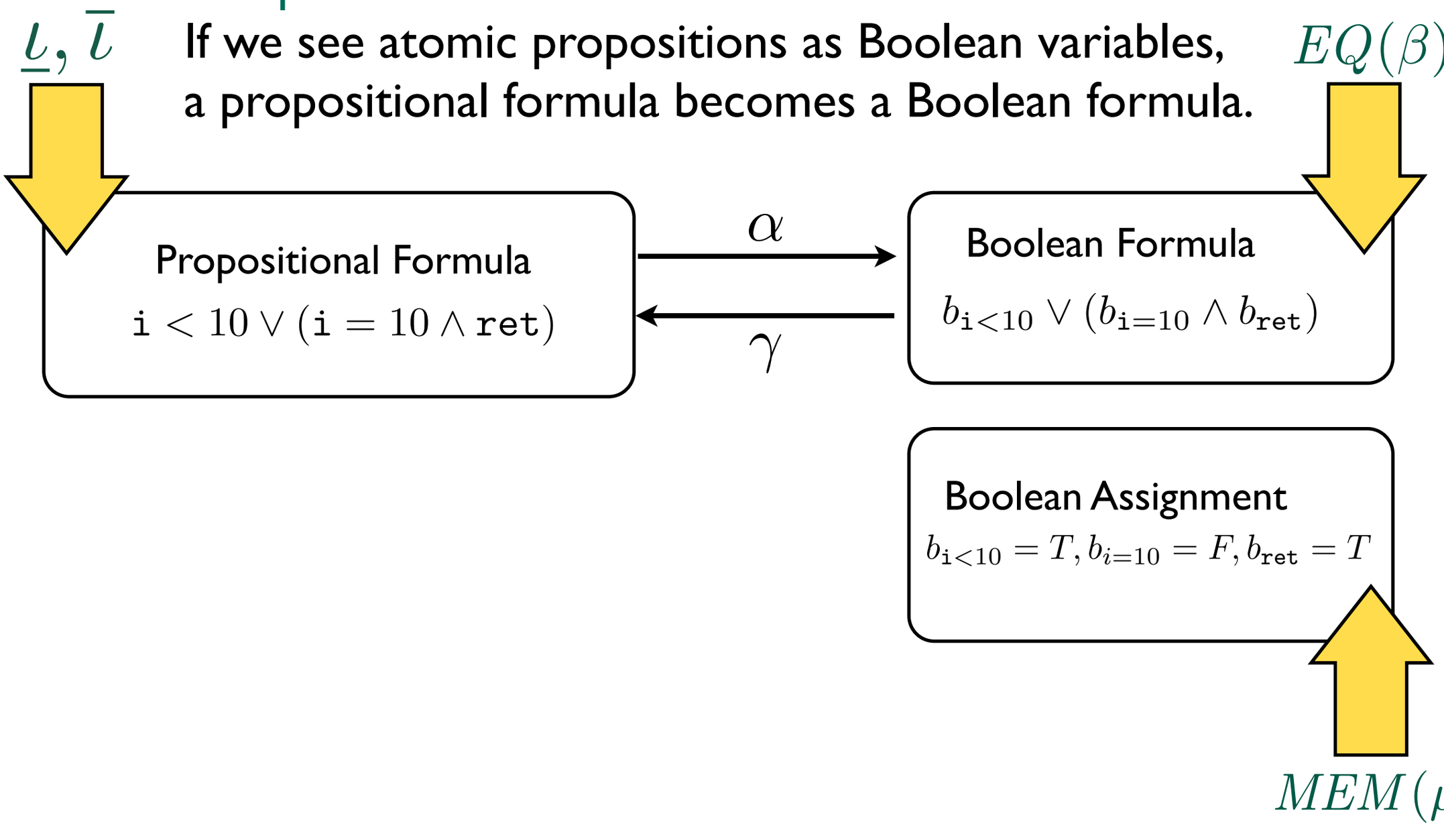
Use predicate abstraction!
If we see atomic propositions as Boolean variables,
a propositional formula becomes a Boolean formula.



Predicate Abstraction

Solution:

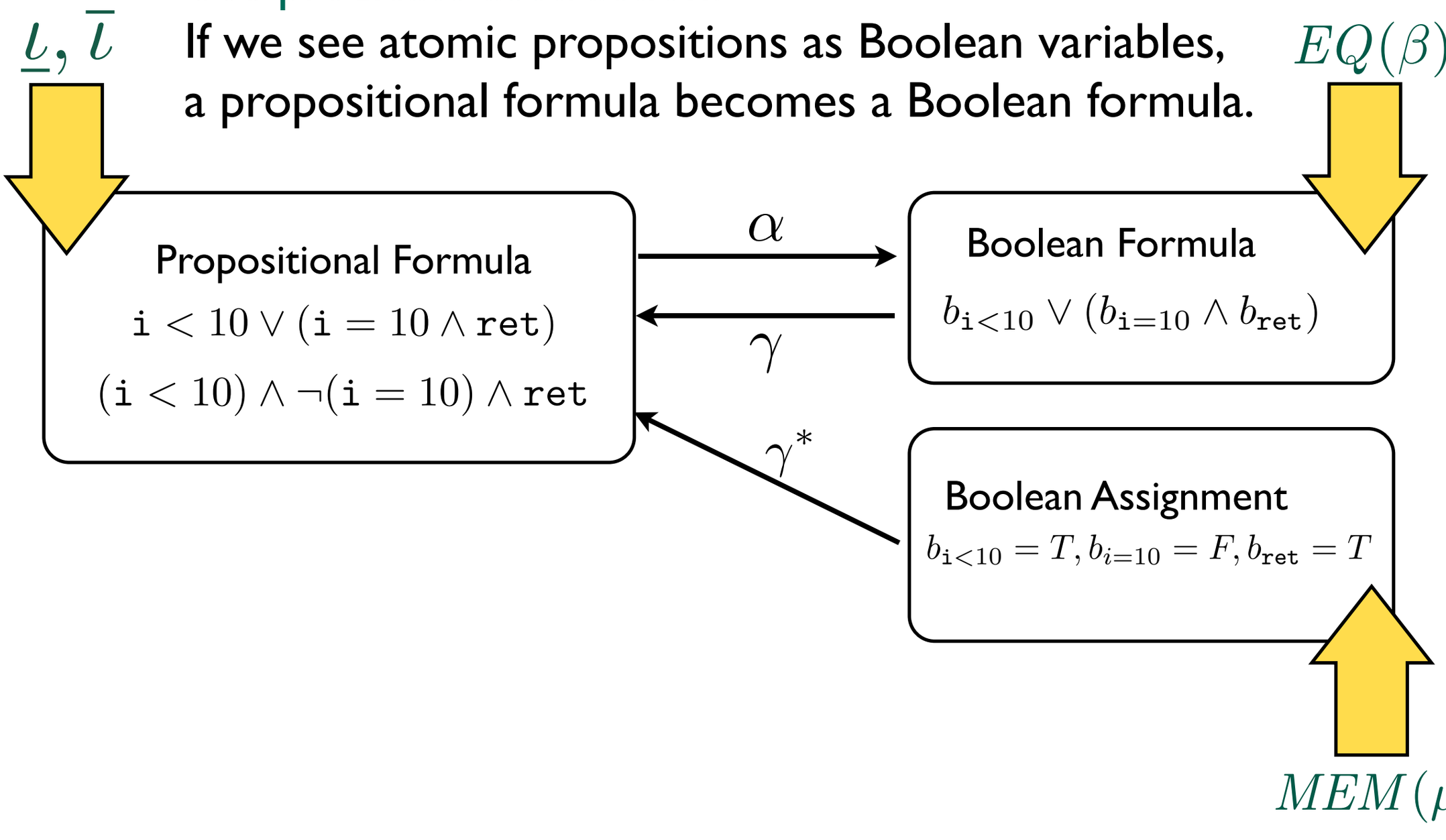
Use predicate abstraction!
If we see atomic propositions as Boolean variables,
a propositional formula becomes a Boolean formula.



Predicate Abstraction

Solution:

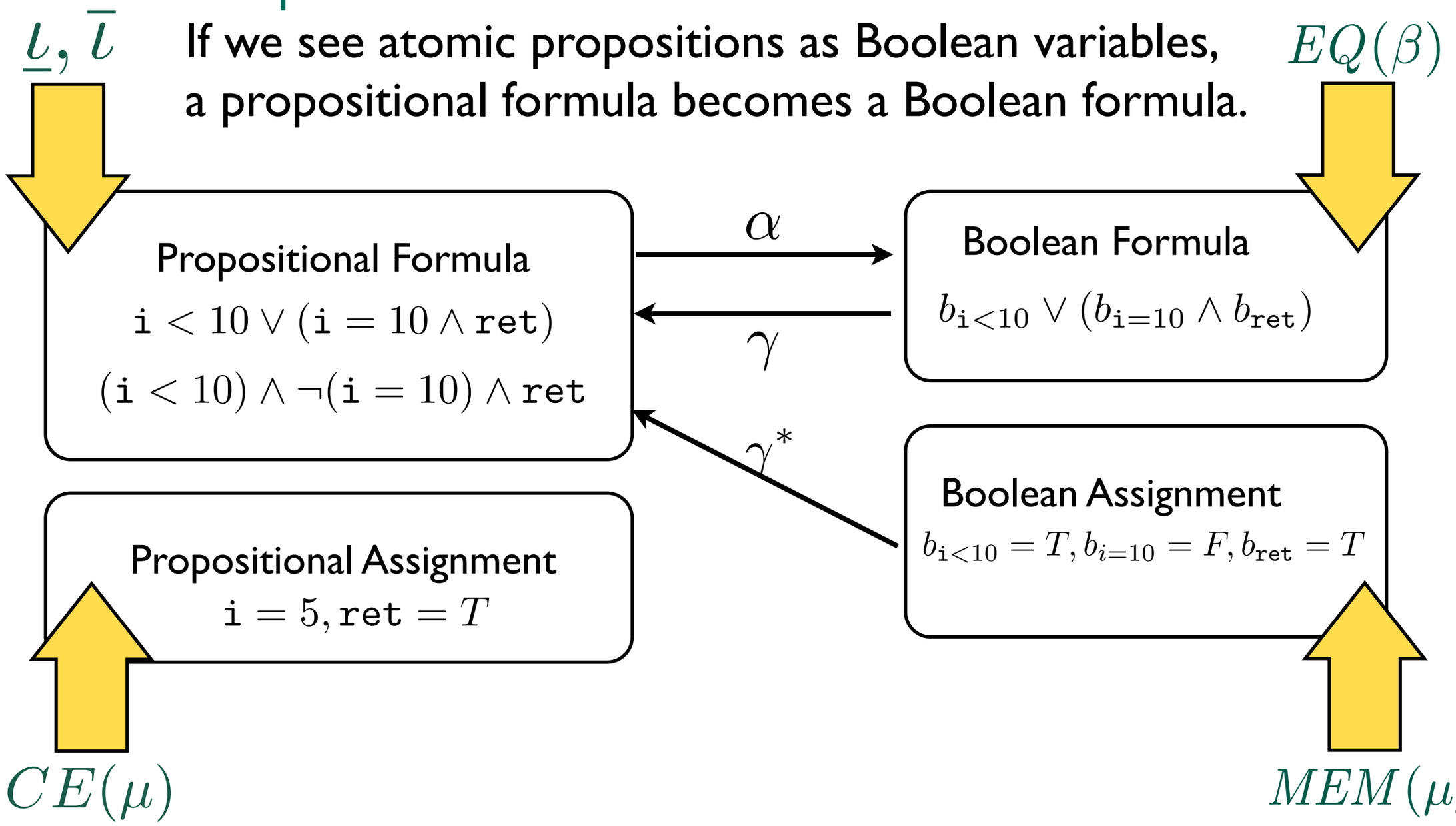
Use predicate abstraction!
If we see atomic propositions as Boolean variables,
a propositional formula becomes a Boolean formula.



Predicate Abstraction

Solution:

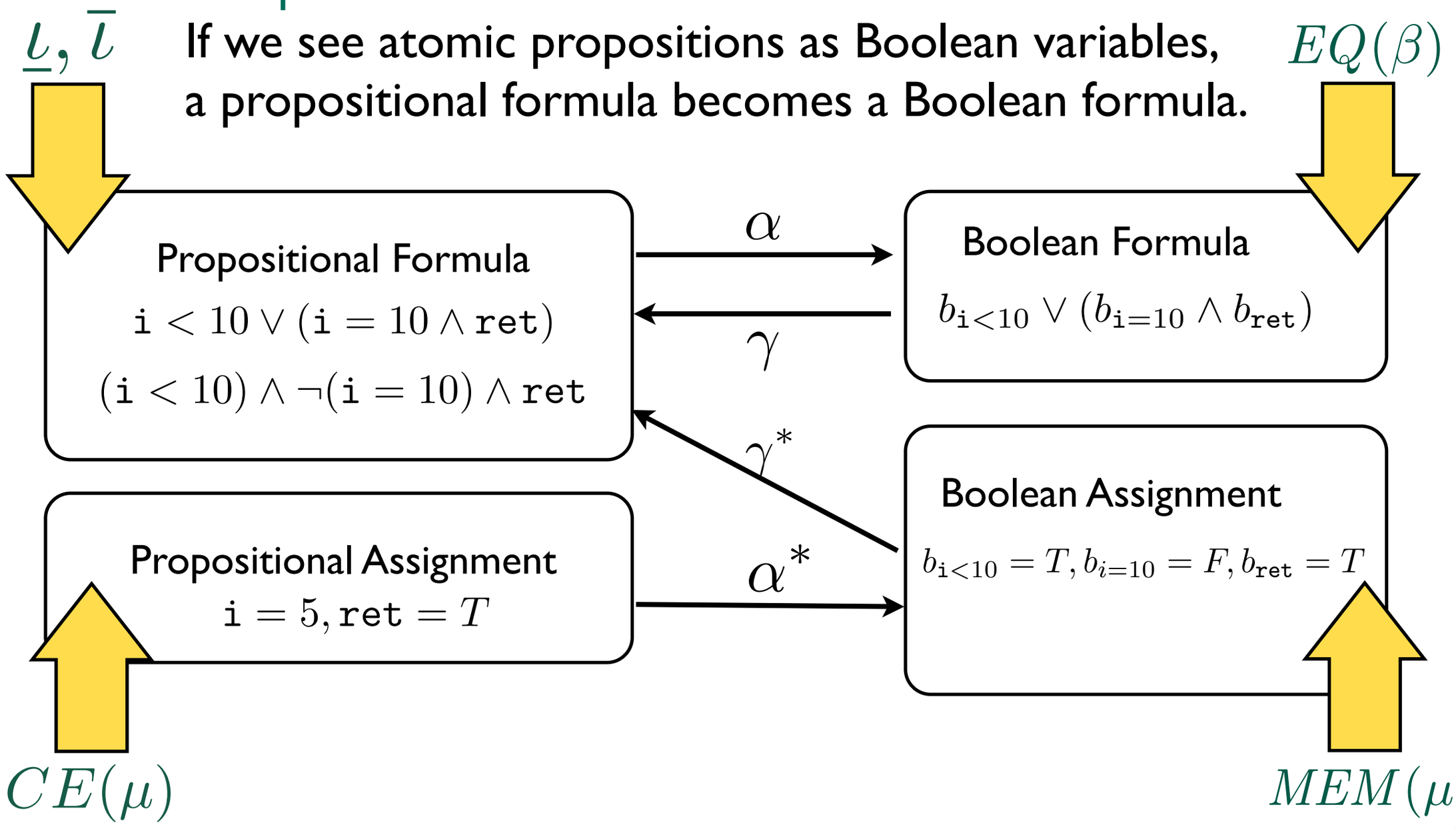
Use predicate abstraction!
If we see atomic propositions as Boolean variables,
a propositional formula becomes a Boolean formula.



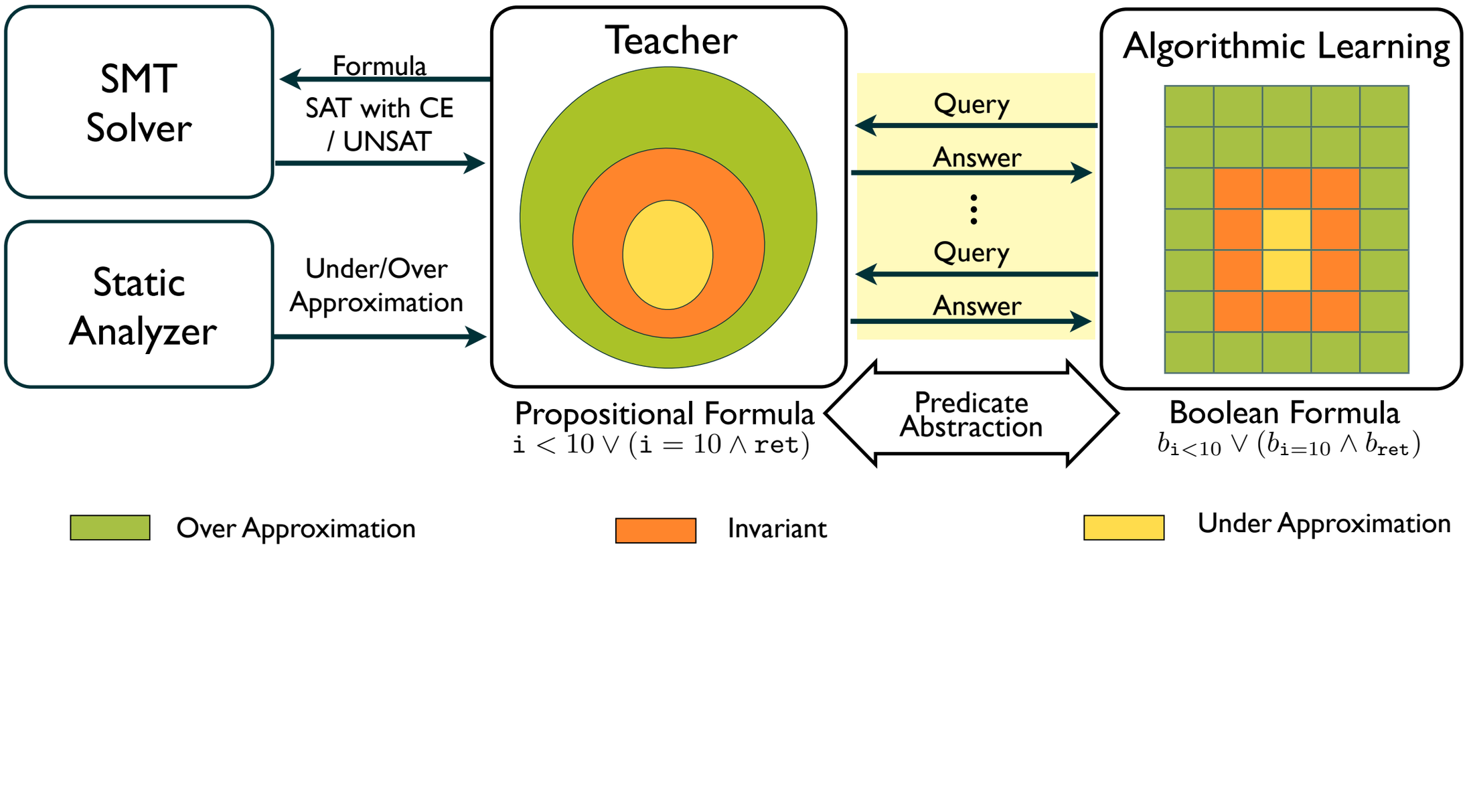
Predicate Abstraction

Solution:

Use predicate abstraction!
If we see atomic propositions as Boolean variables,
a propositional formula becomes a Boolean formula.



Overview


SMT Solver
Formula
SAT with CE / UNSAT
Static Analyzer
Under/Over Approximation
Teacher
Query
Answer
Query
Answer
Algorithmic Learning
Propositional Formula
$\mathtt{i} < 10 \vee (\mathtt{i} = 10 \wedge \mathtt{ret})$
Predicate Abstraction
Boolean Formula
$b_{\mathtt{i}<10} \vee (b_{\mathtt{i}=10} \wedge b_{\mathtt{ret}})$
Over Approximation
Invariant
Under Approximation

Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma(\beta)$ is an invariants (check the three conditions), then "Yes"

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

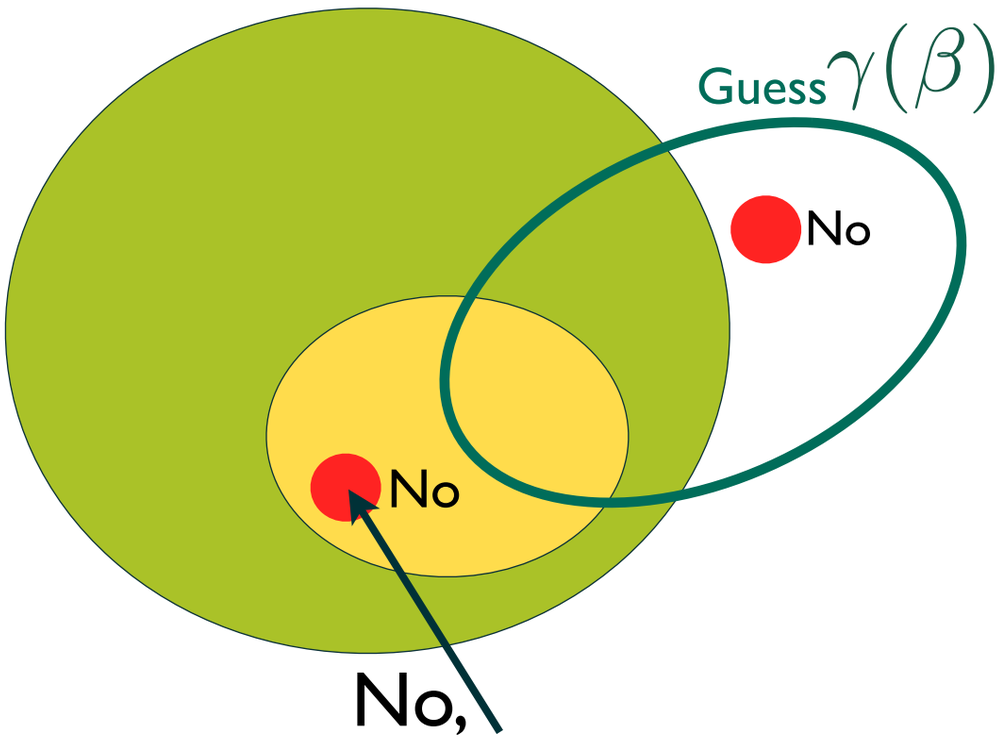
(B) $I \wedge \rho \Rightarrow Pre(I, S)$ (I holds at each iteration)

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

Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



with found a counterexample μ.

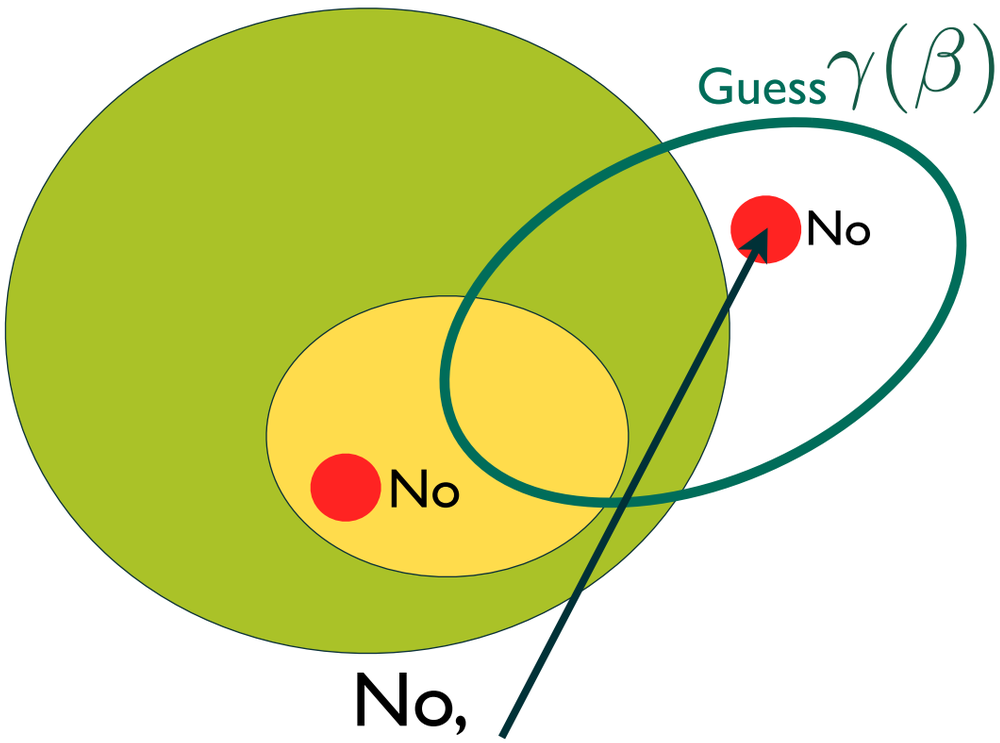
$$\mu \models \gamma(\beta) \oplus \underline{\iota}$$

Over Approximation Under Approximation

Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



with found a counterexample μ.

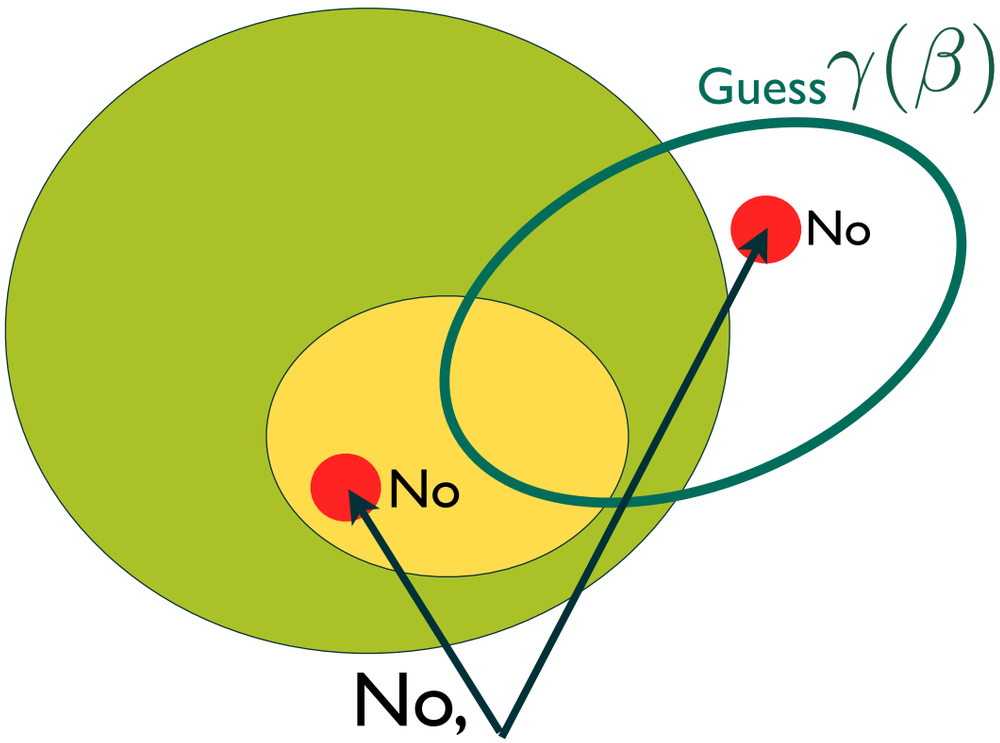
$$\mu \models \gamma(\beta) \oplus \bar{\iota}$$

Over Approximation Under Approximation

Resolving Equivalence Query $EQ(\beta)$

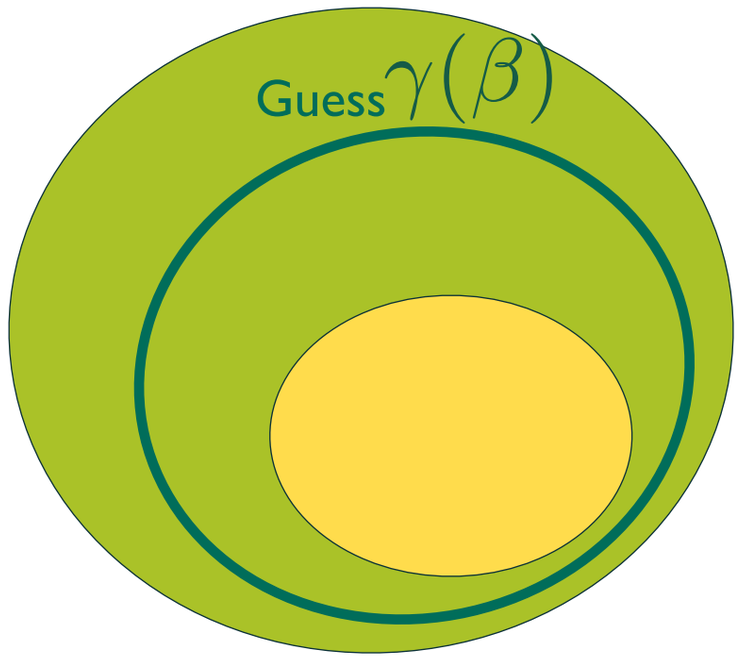
1. If $\gamma(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



with found a counterexample μ.

Case 2 $\underline{\iota} \Rightarrow \gamma(\beta) \Rightarrow \bar{\iota}$



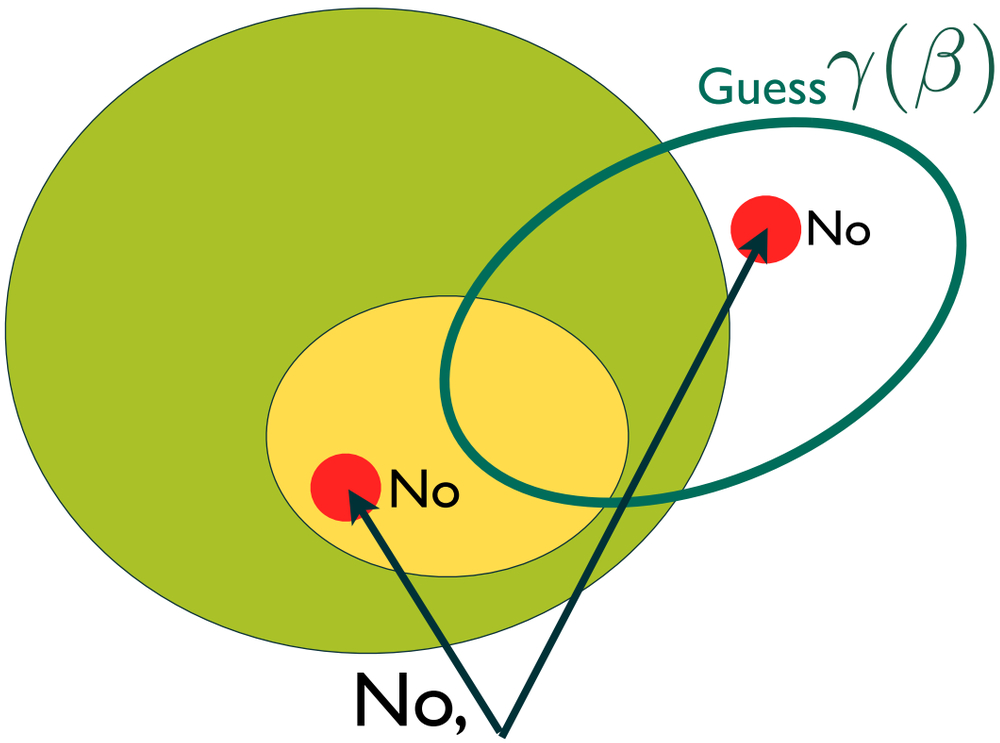
Cannot find a counterexample.

Over Approximation Under Approximation

Resolving Equivalence Query $EQ(\beta)$

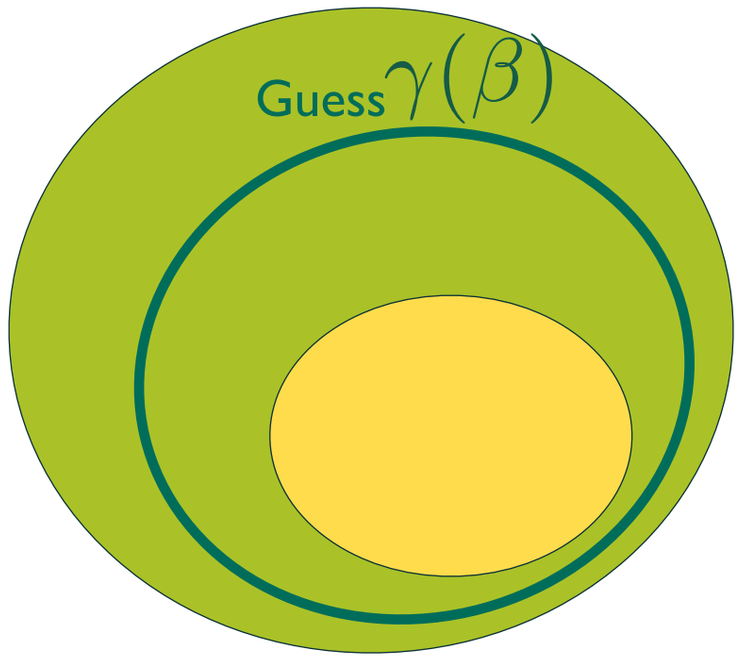
1. If $\gamma(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



with found a counterexample μ.

Case 2 $\underline{\iota} \Rightarrow \gamma(\beta) \Rightarrow \bar{\iota}$



Restart!

Cannot find a counterexample.

Over Approximation Under Approximation

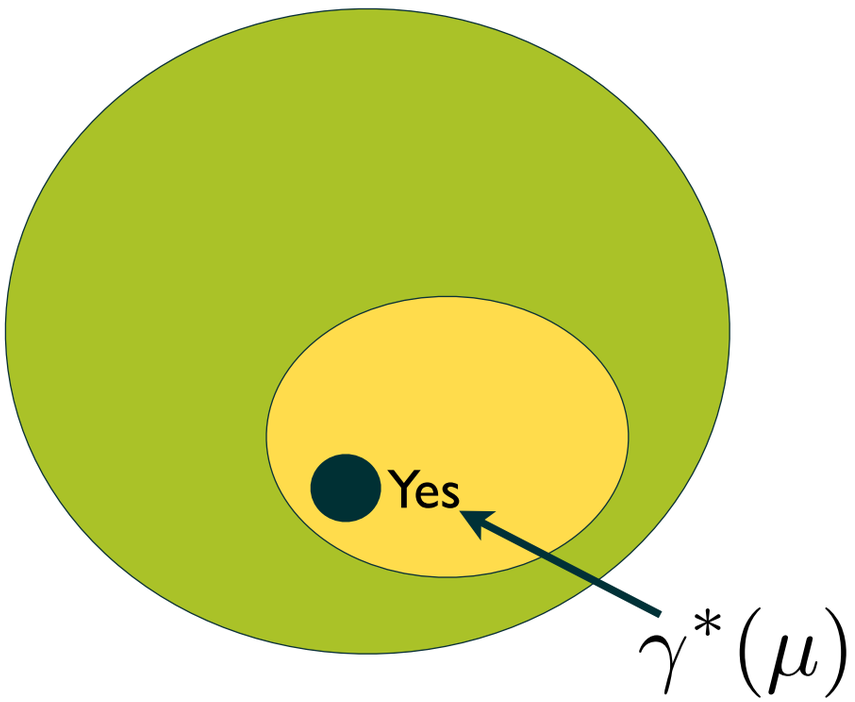
Resolving Membership Query $MEM(\mu)$

Use $\gamma^*(\mu)$ to answer membership query.

Resolving Membership Query $MEM(\mu)$

Use $\gamma^*(\mu)$ to answer membership query.

Case 1



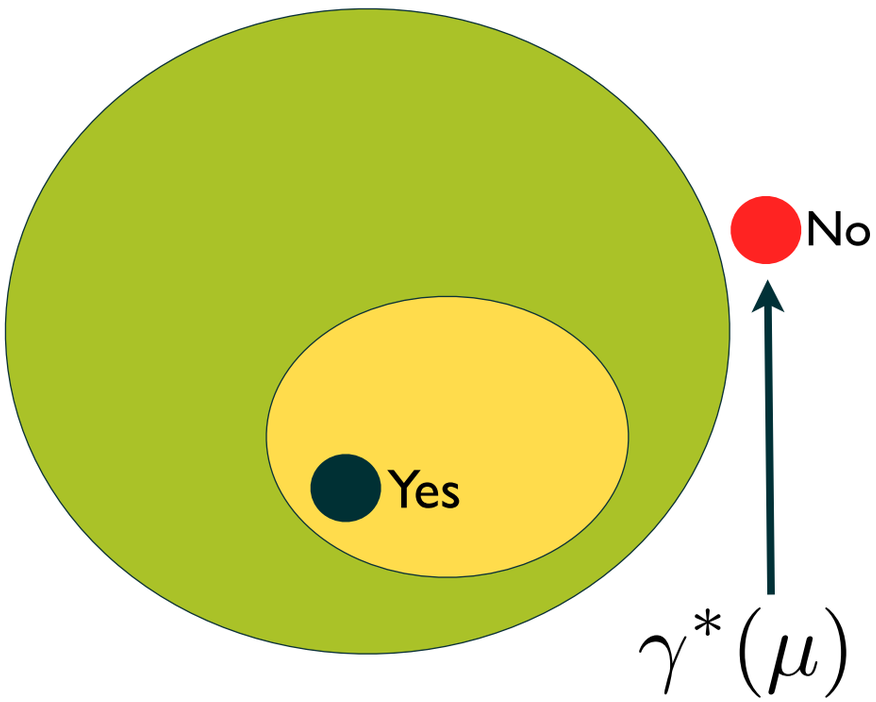
Answer Yes

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

Resolving Membership Query $MEM(\mu)$

Use $\gamma^*(\mu)$ to answer membership query.

Case 1



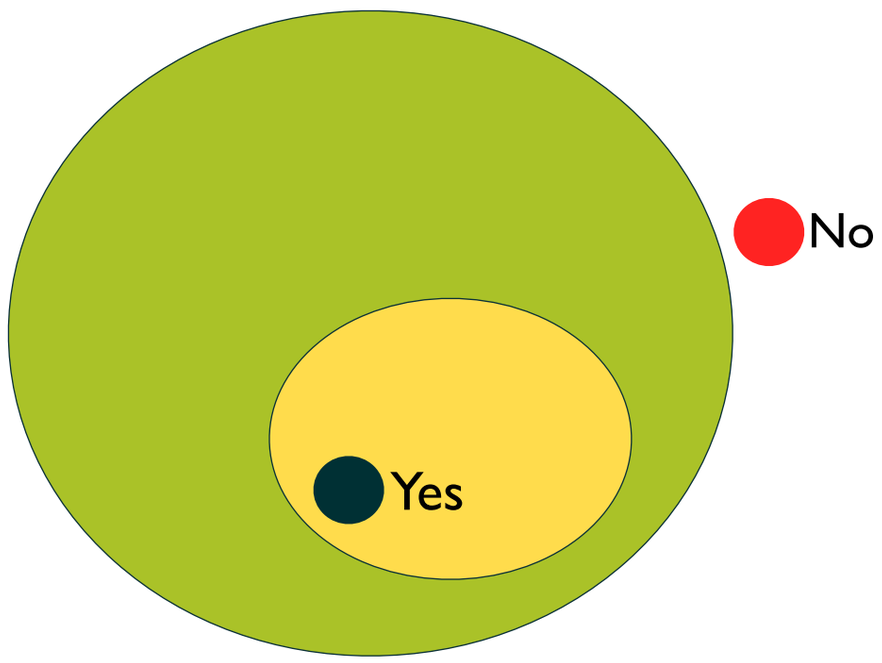
Answer No

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

Resolving Membership Query $MEM(\mu)$

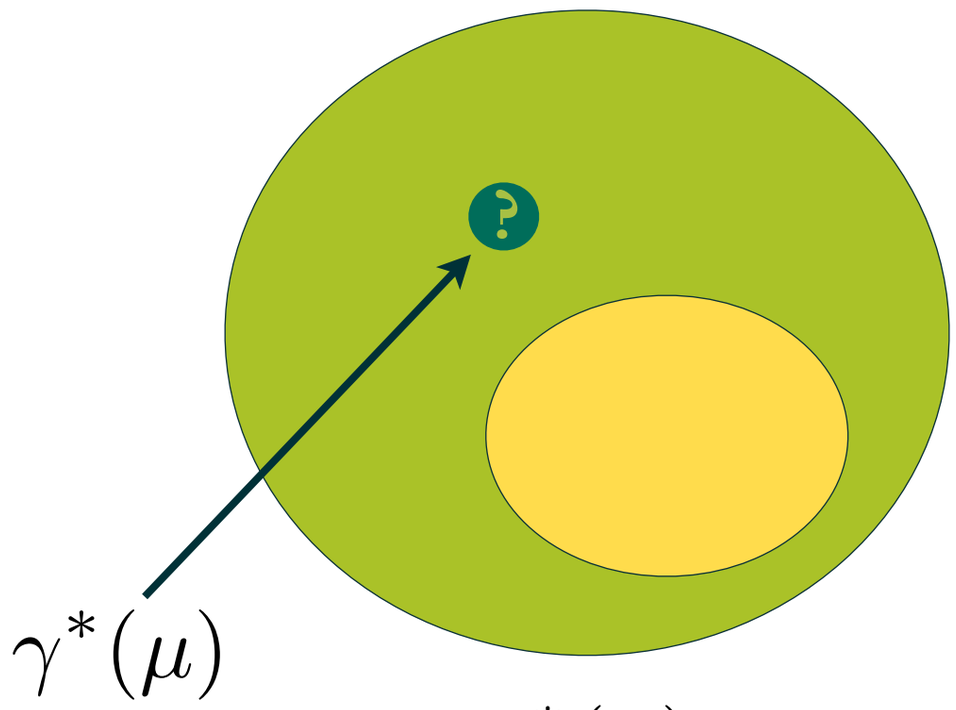
Use $\gamma^*(\mu)$ to answer membership query.

Case 1



Answer Yes/No

Case 2

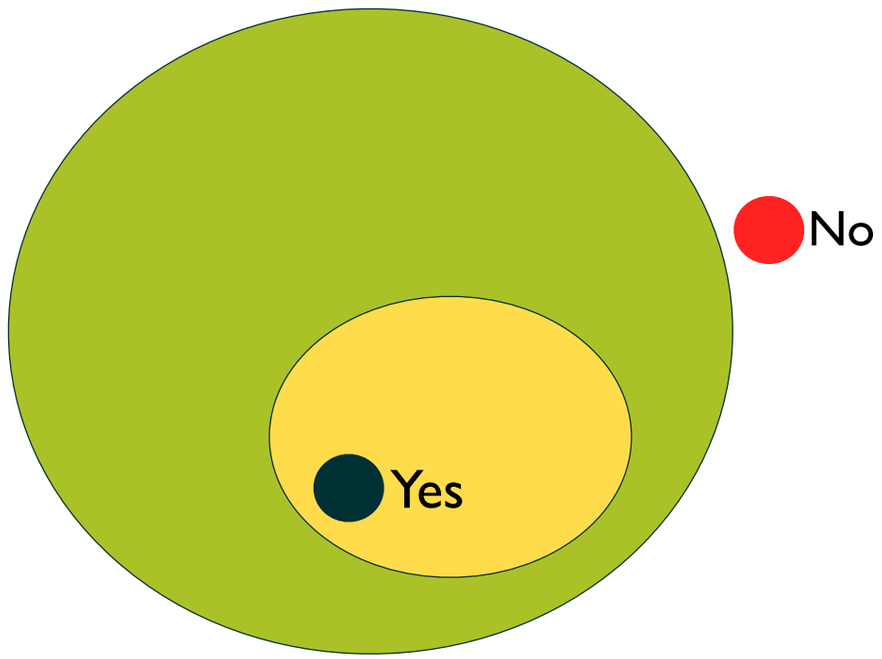


$\underline{\iota} \Rightarrow \gamma^*(\mu) \Rightarrow \bar{\iota}$

Resolving Membership Query $MEM(\mu)$

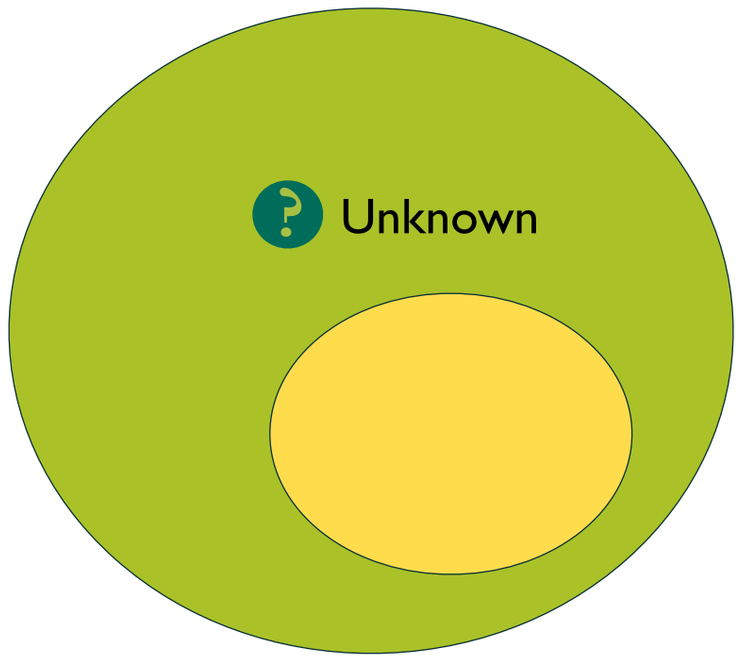
Use $\gamma^*(\mu)$ to answer membership query.

Case 1



Answer Yes/No

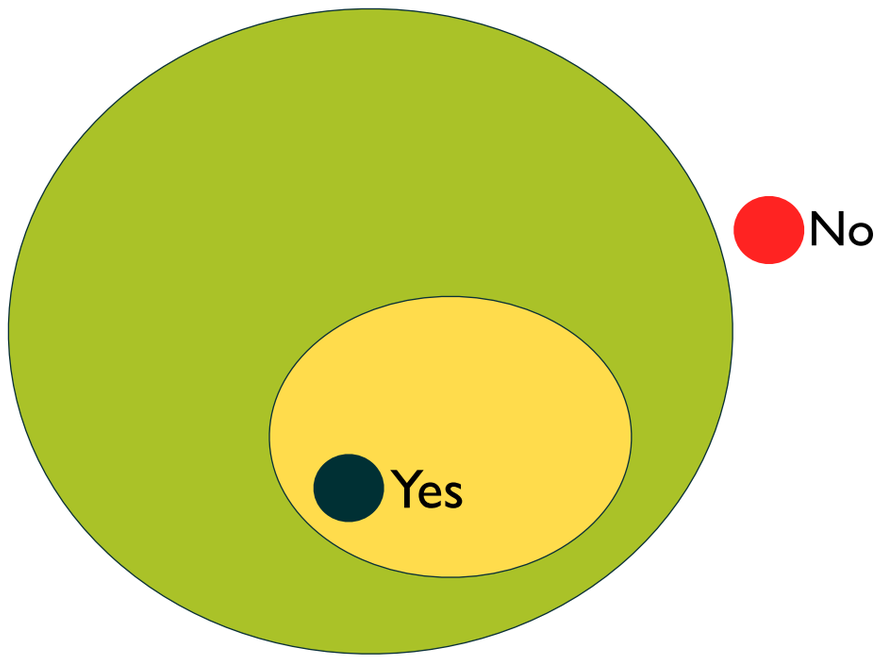
Case 2



Resolving Membership Query $MEM(\mu)$

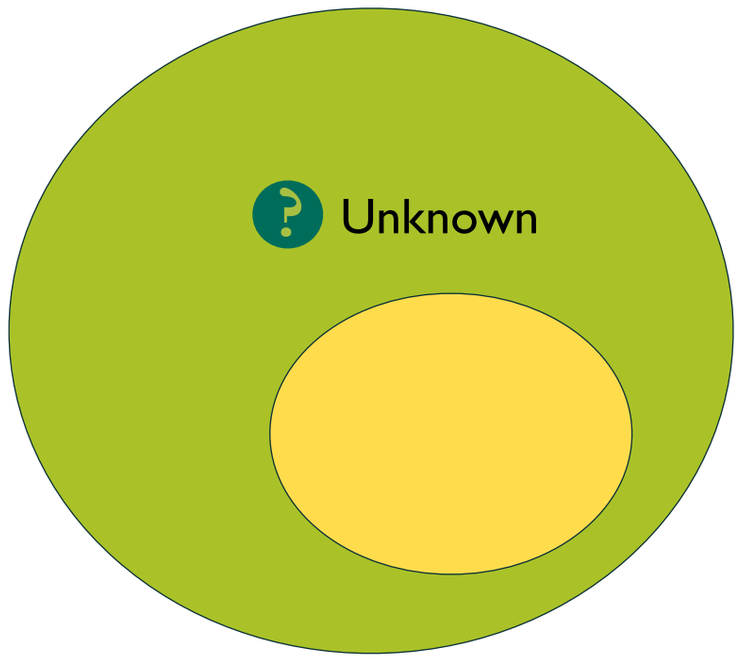
Use $\gamma^*(\mu)$ to answer membership query.

Case 1



Answer Yes/No

Case 2



Random Answer!

Q: Random Answer, Sound?

Q: Random Answer, Sound?

Yes, it is sound. Found invariants are always real invariants.

Why Sound?

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 ϵ after leaving the loop)

1. If $\gamma(\beta)$ is an invariants (check the three conditions), then "Yes"

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

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

Deriving Quantified Invariants using Algorithmic Learning

Problems

1. Incompleteness in SMT

2. Predicate Abstraction between
   First-order formulae and Boolean formulae

3. Query Resolution

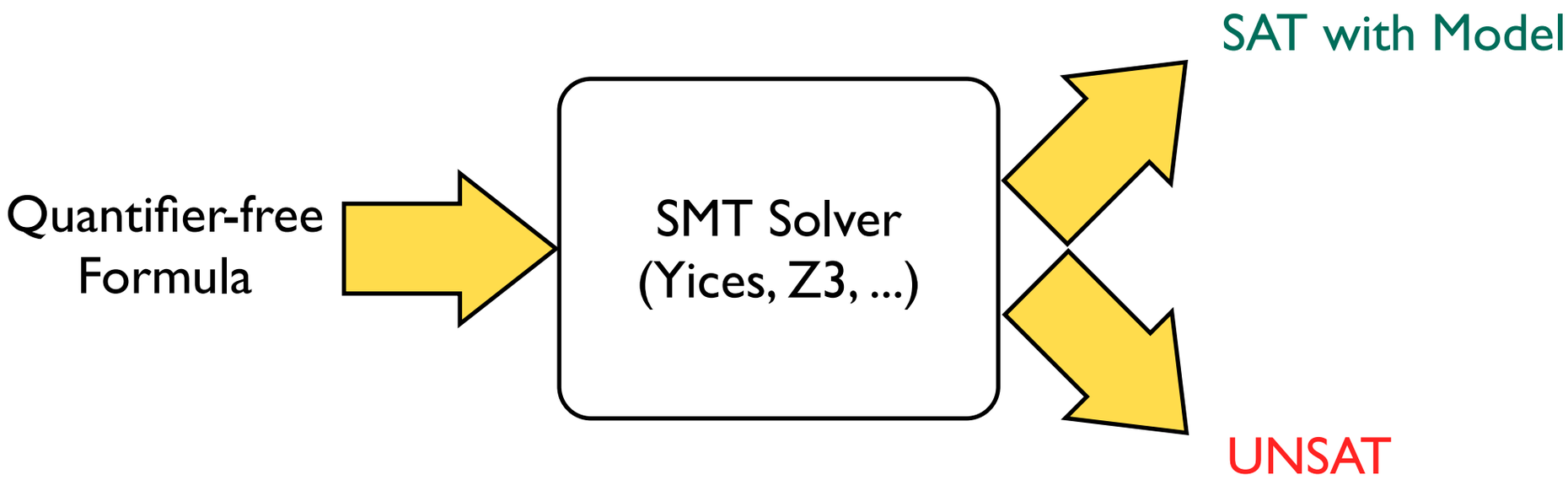
Problems

1. Incompleteness in SMT

2. Predicate Abstraction between
   First-order formulae and Boolean formulae

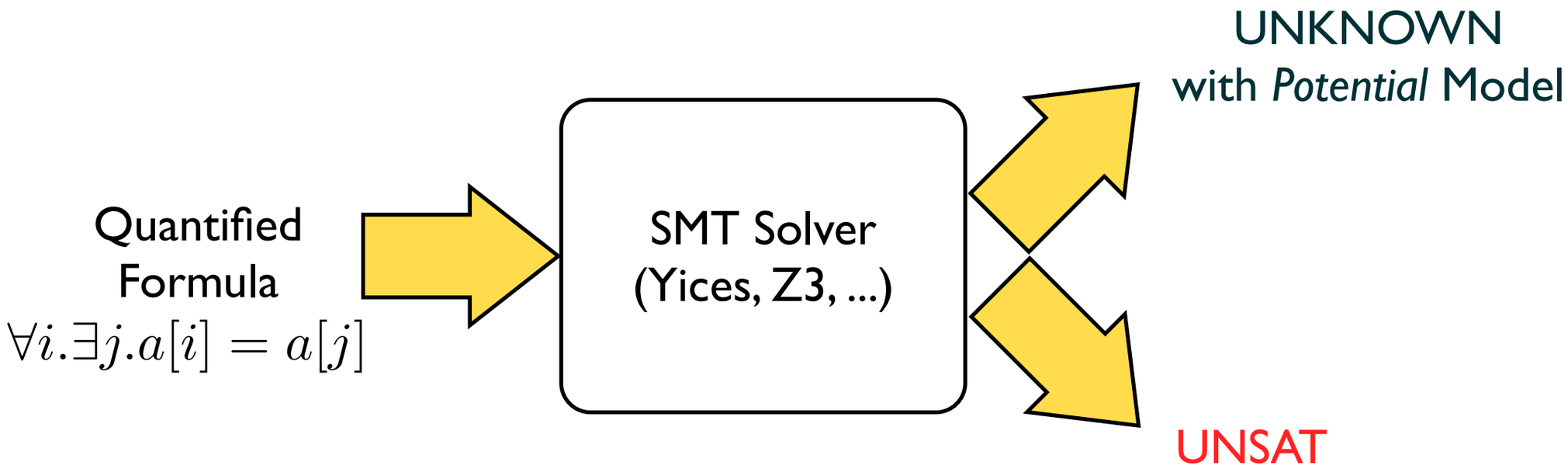
3. Query Resolution

Incompleteness in SMT



SMT solver is sound and complete for the quantifier-free formulae

Incompleteness in SMT



SMT solver is sound but incomplete for the quantified formulae

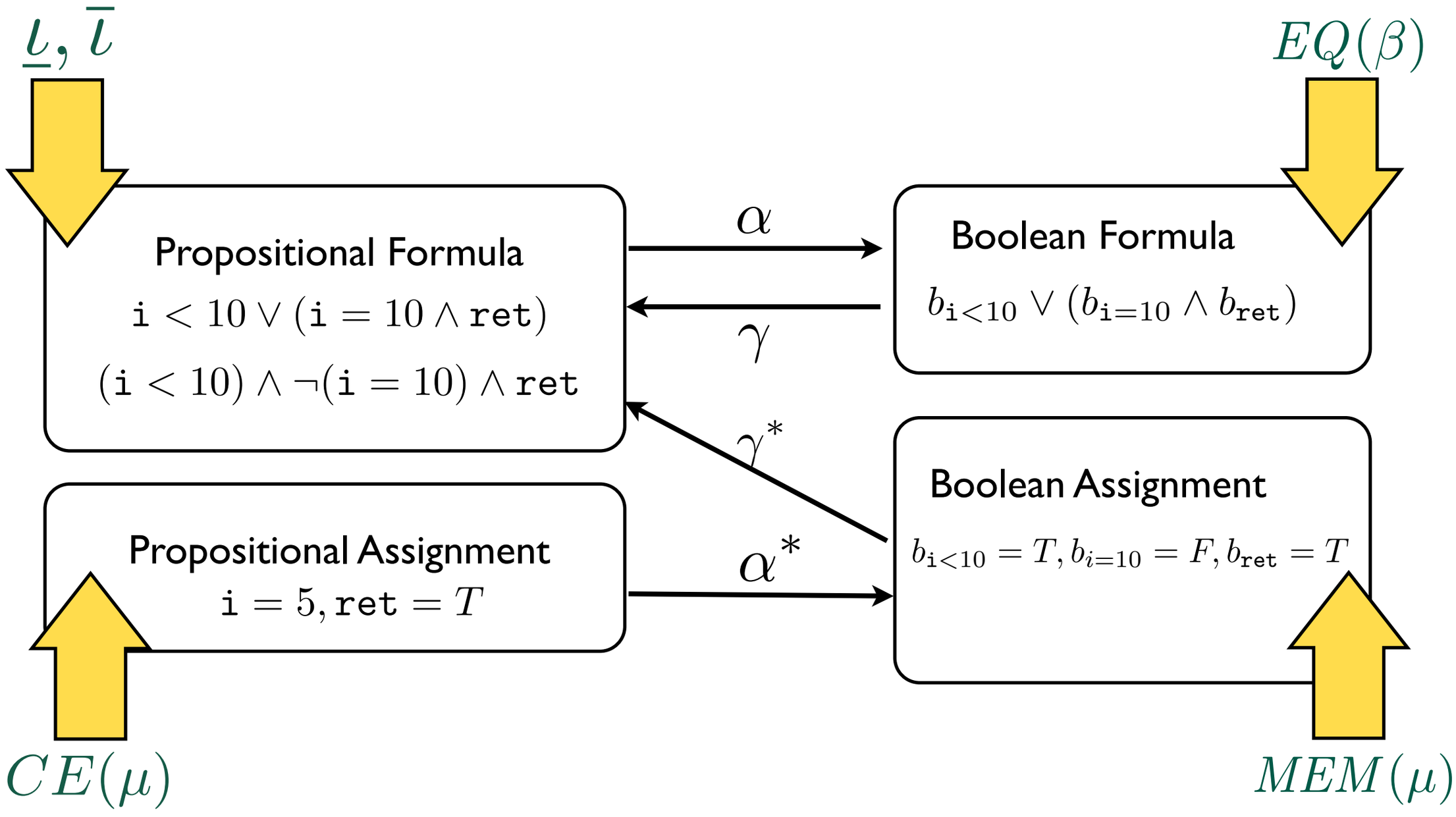
Problems

1. Incompleteness in SMT

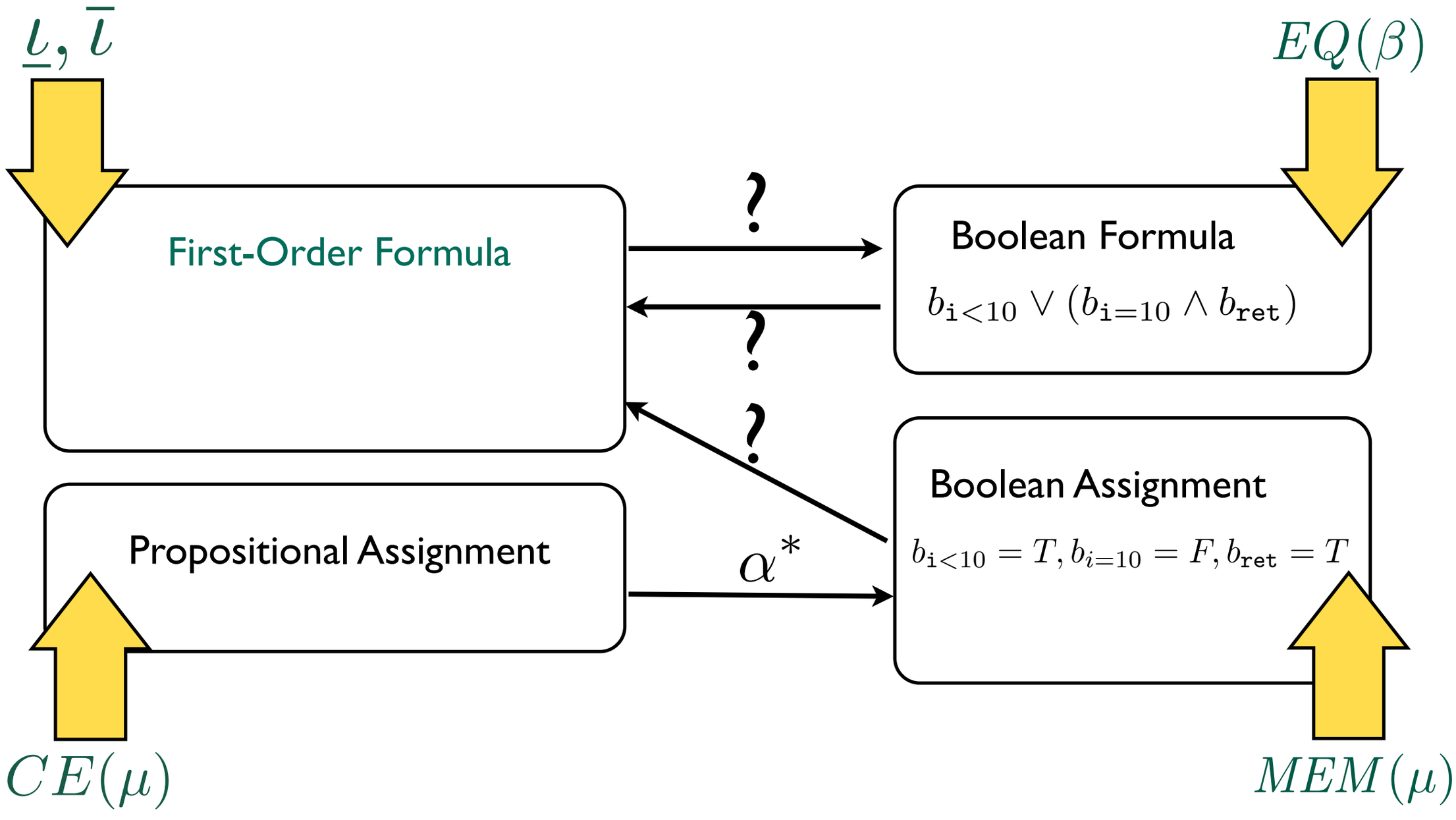
2. Predicate Abstraction between
   First-order formulae and Boolean formulae

3. Query Resolution

Predicate Abstraction in VMCAI'10

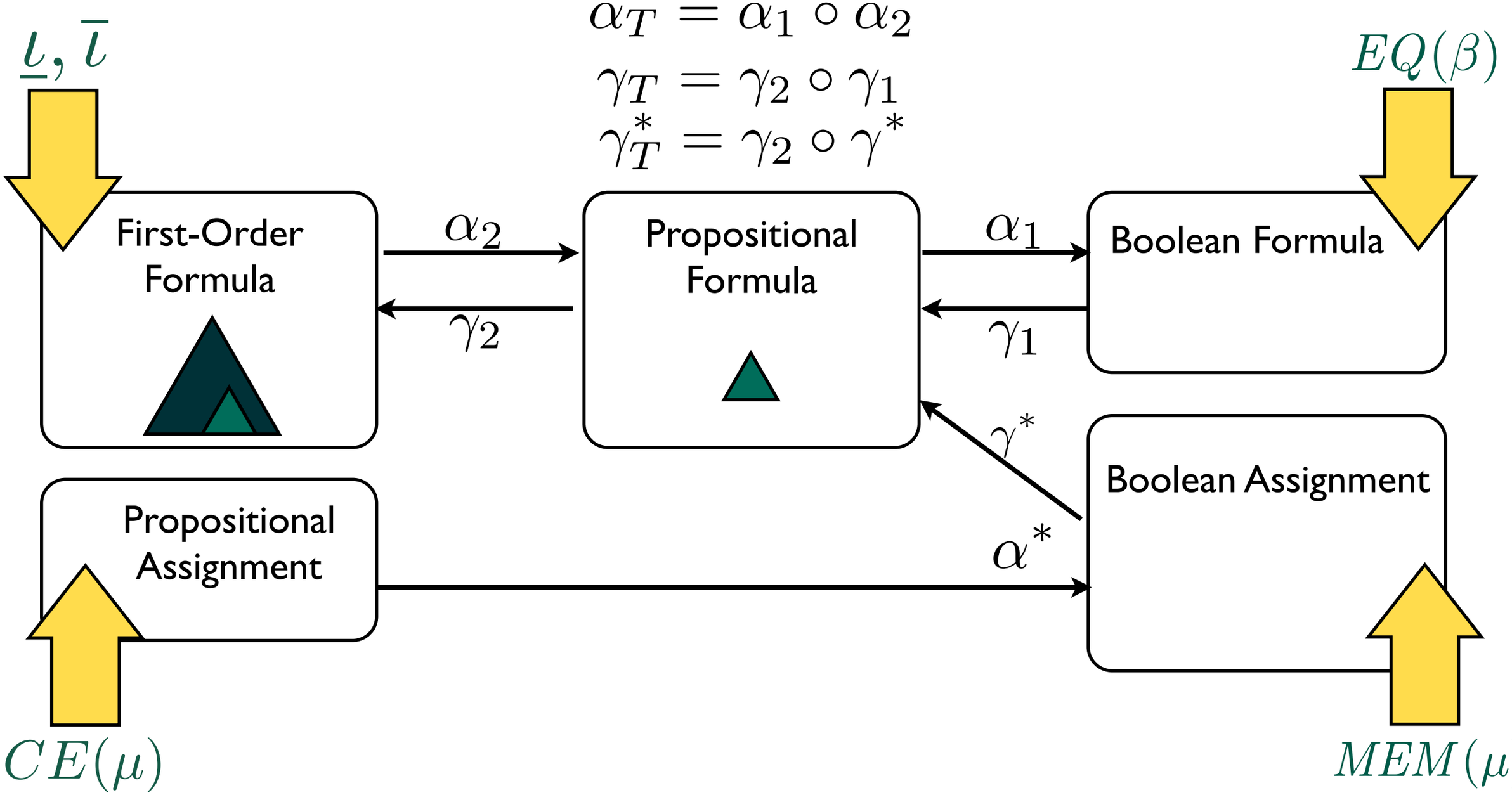


Need New Predicate Abstraction



Predicate Abstraction in Template Based Approach

$$\alpha_T = \alpha_1 \circ \alpha_2$$
$$\gamma_T = \gamma_2 \circ \gamma_1$$
$$\gamma_T^* = \gamma_2 \circ \gamma^*$$



Predicate Abstraction in Template Based Approach

Predicate Abstraction does not established with any arbitrary templates.

$$T[x] = x \vee true$$

$$(p_1 \Rightarrow p_2) \Rightarrow (T[p_1] \Rightarrow T[p_2])$$
$$(p_1 \Rightarrow p_2) \Rightarrow (T[p_2] \Rightarrow T[p_1])$$

$$T[x] = x \wedge false$$

$$(p_1 \Rightarrow p_2) \Rightarrow (T[p_1] \Rightarrow T[p_2])$$
$$(p_1 \Rightarrow p_2) \Rightarrow (T[p_2] \Rightarrow T[p_1])$$

Predicate Abstraction in Template Based Approach

Predicate Abstraction does not established with any arbitrary templates.
We need a notion of *well-formedness* for templates.

Predicate Abstraction in Template Based Approach

Predicate Abstraction does not established with any arbitrary templates.
We need a notion of *well-formedness* for templates.

Def 3 Well-formed template

A template t with unknown x is well-formed with respect to $p_1 \in \mathsf{Prop}_A$ and a valuation S of the free variables under the following conditions ($t[p]$ is a shorthand for $t[x \mapsto p]$):

- $t[p_1] = p_1$ *is well-formed*
- $t[p_1] = \forall I.t'[p_1]$ *is well-formed if* $t'[p_1]$ *is well-formed*
- $t[p_1] = t'[p_1] \vee p$ *is well-formed if* $S \models t'[p_1] \not\Rightarrow p$ *and* $t'[p_1]$ *is well-formed*
- *if* $t[p_1] = t'[p_1] \wedge p$ *is well-formed if* $S \models t\_pos'[p_1] \wedge p$ *and* $t'[p_1]$ *is well-formed*
- $t[p_1] = \neg t'[p_1]$ *is well-formed if* $t'[p_1]$ *is well-formed*

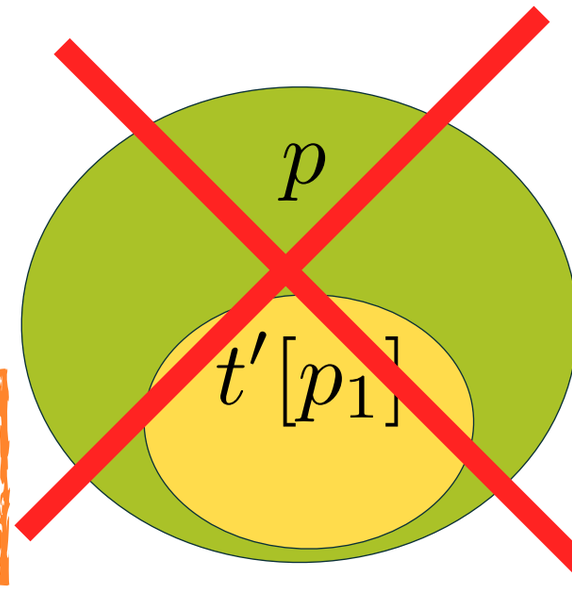
Predicate Abstraction in Template Based Approach

Predicate Abstraction does not established with any arbitrary templates.
We need a notion of *well-formedness* for templates.

Def 3 Well-formed template

A template t with unknown x is well-formed with respect to $p_1 \in \mathsf{Prop}_A$ and a valuation S of the free variables under the following conditions ($t[p]$ is a shorthand for $t[x \mapsto p]$):

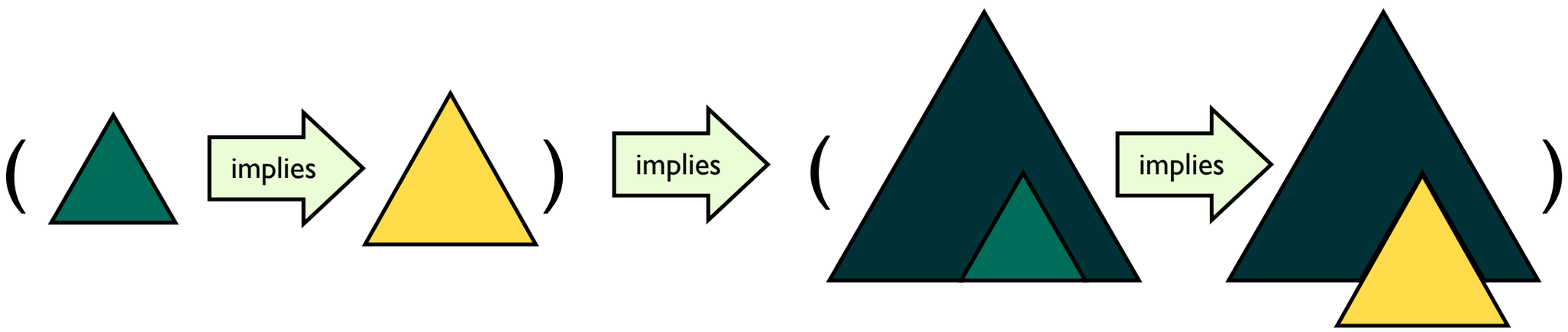
- $t[p_1] = p_1$ *is well-formed*
- $t[p_1] = \forall I.t'[p_1]$ *is well-formed if* $t'[p_1]$ *is well-formed*
- $t[p_1] = t'[p_1] \vee p$ *is well-formed if* $S \models t'[p_1] \not\Rightarrow p$ *and* $t'[p_1]$ *is well-formed*
- *if* $t[p_1] = t'[p_1] \wedge p$ *is well-formed if* $S \models t\_pos'[p_1] \wedge p$ *and* $t'[p_1]$ *is well-formed*
- $t[p_1] = \neg t'[p_1]$ *is well-formed if* $t'[p_1]$ *is well-formed*

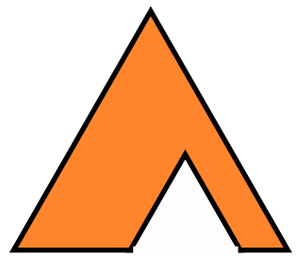


Positive Template 

Def 1 Positive unknown *(this definition applies to general templates with more than one hole)*
Given a template t, an unknown v is said to be positive if, given any substitution σ_v that maps all unknown variables in t different from v to some set of predicates, the following holds:

$$\forall \sigma_v, p_1, p_2 : (p_1 \Rightarrow p_2) \Rightarrow (t\sigma_v[v \mapsto p_1] \Rightarrow t\sigma_v[v \mapsto p_2]).$$

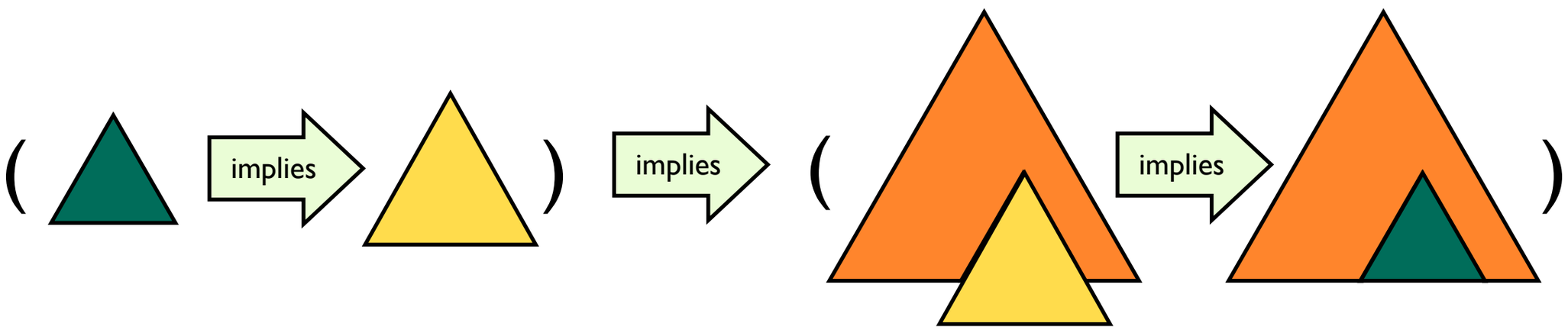


Negative Template 

Def 2 Negative unknown

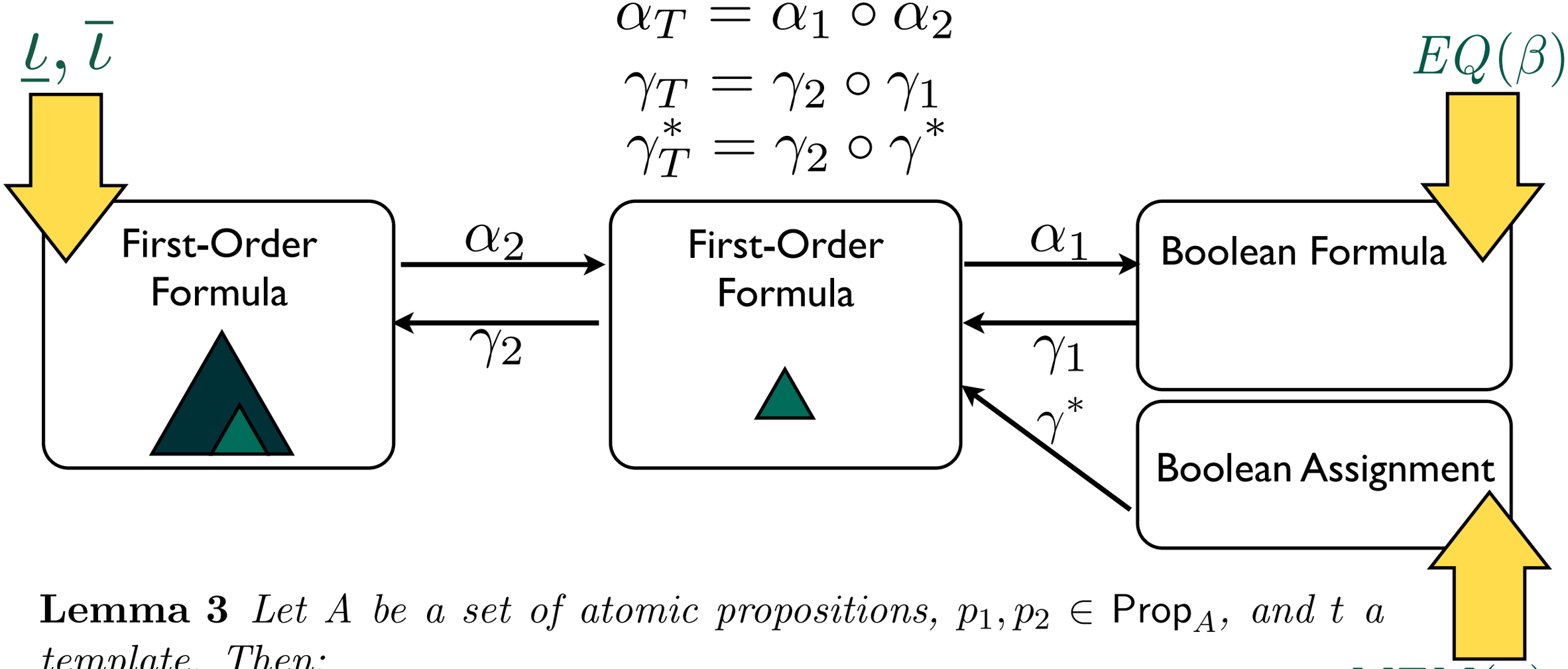
Given a template t, an unknown v is said to be negative if, given any substitution σ_v that maps all unknown variables in t different from v to some set of predicates, the following holds:

$$\forall \sigma_v, p_1, p_2 : (p_1 \Rightarrow p_2) \Rightarrow (t\sigma_v[v \mapsto p_2] \Rightarrow t\sigma_v[v \mapsto p_1]).$$



Predicate Abstraction

$$\alpha_T = \alpha_1 \circ \alpha_2$$
$$\gamma_T = \gamma_2 \circ \gamma_1$$
$$\gamma_T^* = \gamma_2 \circ \gamma^*$$



Lemma 3 *Let A be a set of atomic propositions, $p_1, p_2 \in \mathsf{Prop}_A$, and t a template. Then:*

$S \models t[p_1] \Rightarrow t[p_2]$ *implies* $S \models p_1 \Rightarrow p_2$, *if t has positive unknown and it is well−formed with respect to p_1 and and model S,*
$S \models t[p_1] \Rightarrow t[p_2]$ *implies* $S \models p_2 \Rightarrow p_1$, *if t has positive unknown and it is well−formed with respect to p_2 and and model S.*

where S gives valuations for the free variables.

Predicate Abstraction

Lemma 3 *Let A be a set of atomic propositions, $p_1, p_2 \in \mathsf{Prop}_A$, and t a template. Then:*

$$S \models t[p_1] \Rightarrow t[p_2] \text{ implies } S \models p_1 \Rightarrow p_2, \text{ if } t \text{ has positive unknown and it is well-formed with respect to } p_1 \text{ and and model } S,$$
$$S \models t[p_1] \Rightarrow t[p_2] \text{ implies } S \models p_2 \Rightarrow p_1, \text{ if } t \text{ has positive unknown and it is well-formed with respect to } p_2 \text{ and and model } S.$$

where S gives valuations for the free variables.

This is weaker than we desired: It checks satisfiability, not validity!

Limitation:

UNSAT means "NOT Valid",
however SAT does not always mean "Valid".

Consequence:

We can say "No" for sure at membership query resolution,
however we cannot say "Yes".

Problems

1. Incompleteness in SMT

2. Predicate Abstraction between
   First-order formulae and Boolean formulae

3. Query Resolution

Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"

(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 ϵ after leaving the loop)

Resolving Equivalence Query $EQ(\beta)$

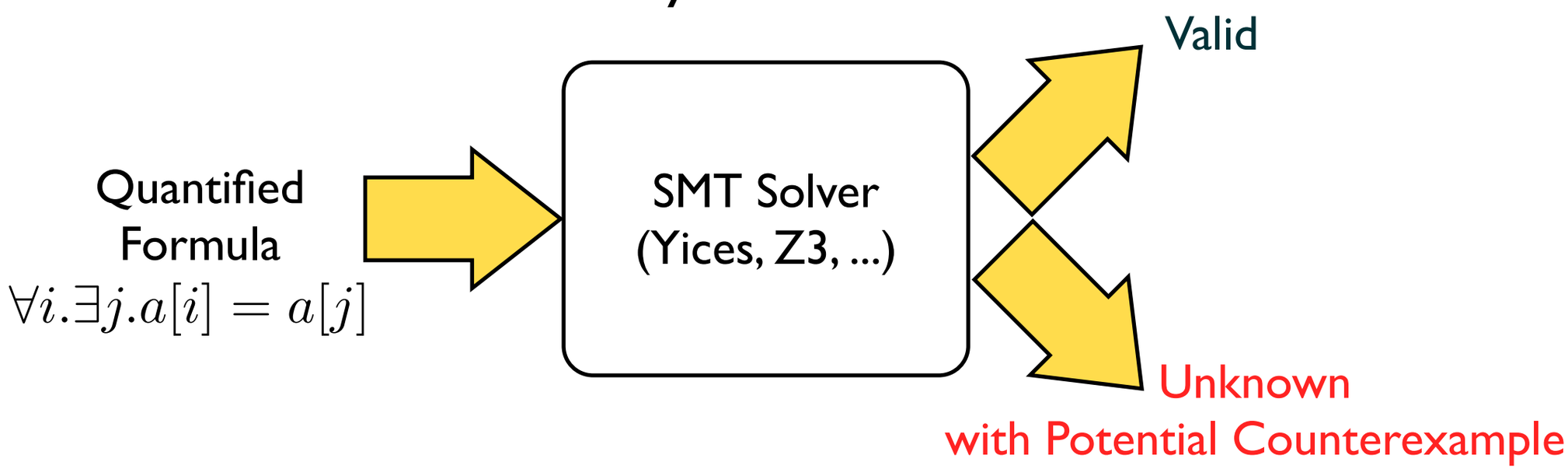
1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"

(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 ϵ after leaving the loop)

Because of the incompleteness of SMT solver,
we have two cases for validity check.



Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"

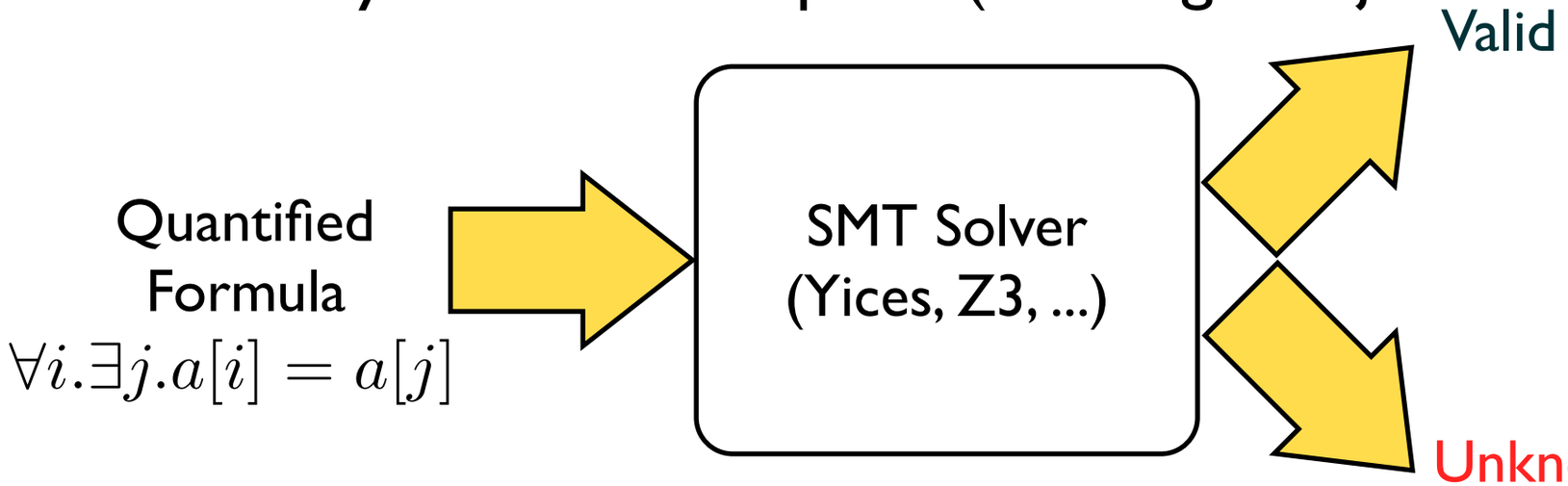
(A) $\delta \Rightarrow I$ (I holds when entering the loop)

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

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

Consequences:

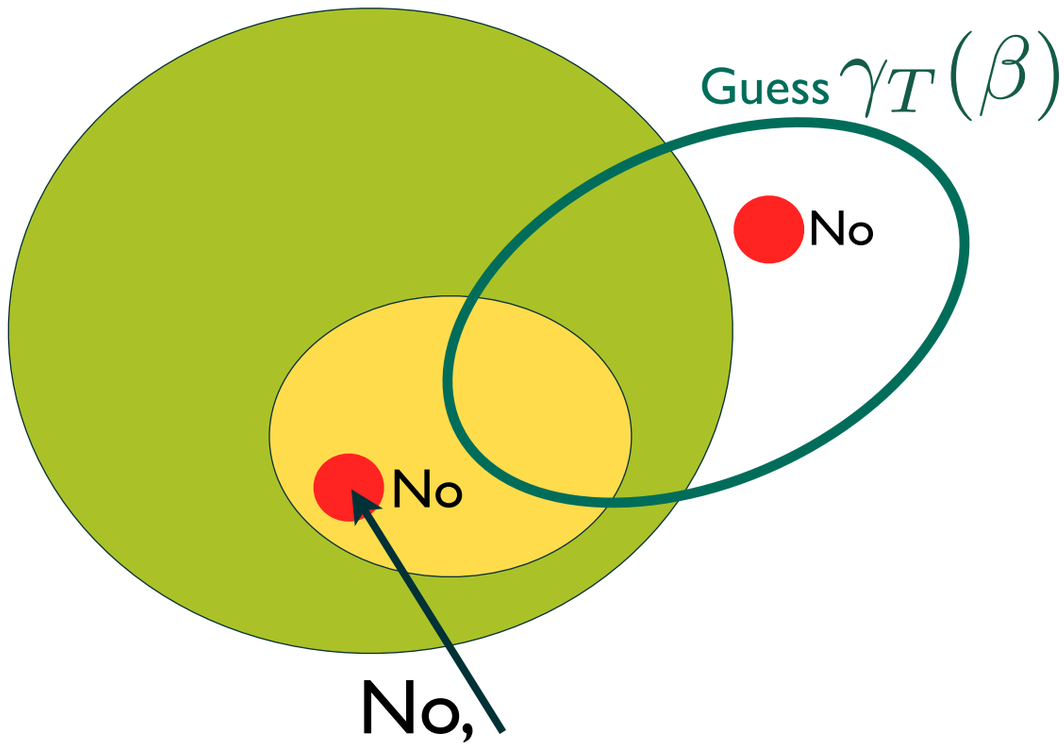
1. The system is sound (Found invariant is the real one)
2. The system is incomplete (We might reject the true invariant.)



Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



with found a potential counterexample μ.

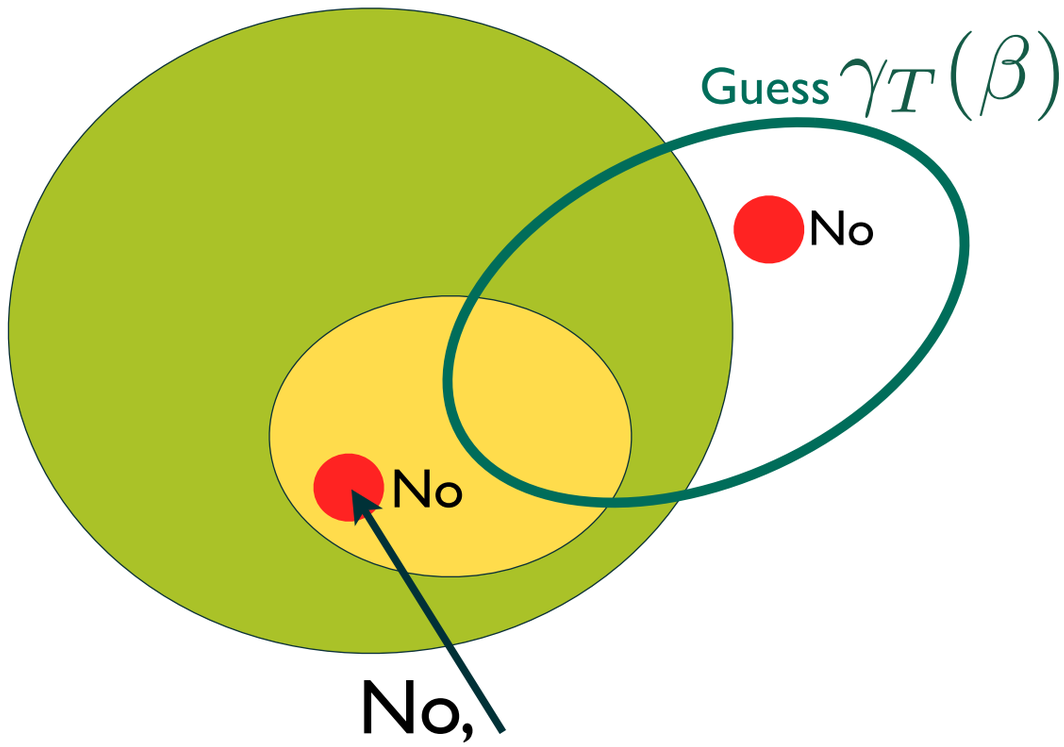
$$\mu \models \gamma_T(\beta) \oplus \underline{\iota}$$

Over Approximation Under Approximation

Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



Potential counterexample:

Could be real => Use it?
Could be false => Restart?

with found a potential counterexample μ.

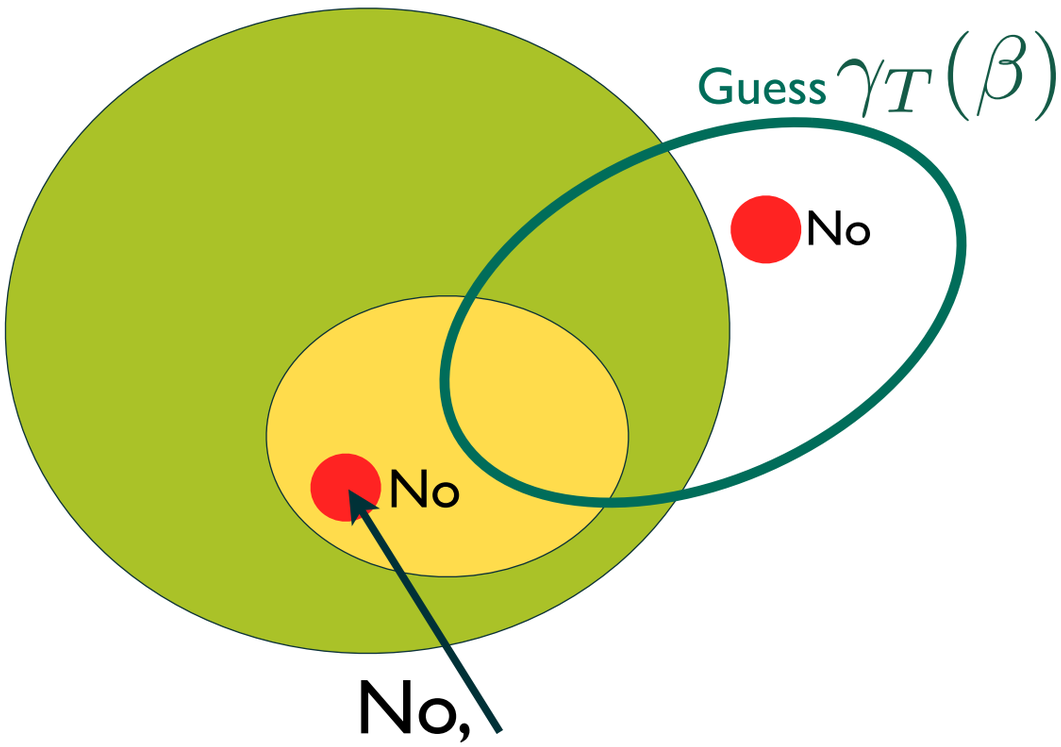
$$\mu \models \gamma_T(\beta) \oplus \underline{\iota}$$

Over Approximation Under Approximation

Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



Potential counterexample:

Just use it!,
It does not affect the soundness.

with found a potential counterexample μ.

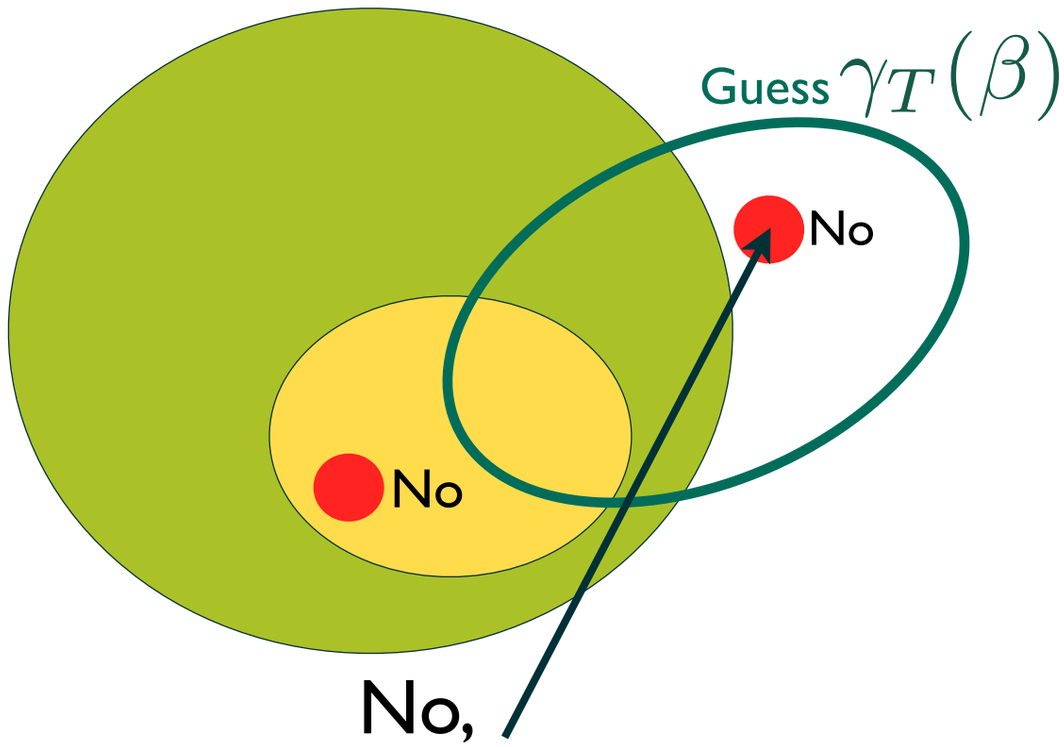
$$\mu \models \gamma_T(\beta) \oplus \underline{\iota}$$

Over Approximation Under Approximation

Resolving Equivalence Query $EQ(\beta)$

1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



with found a potential counterexample μ.

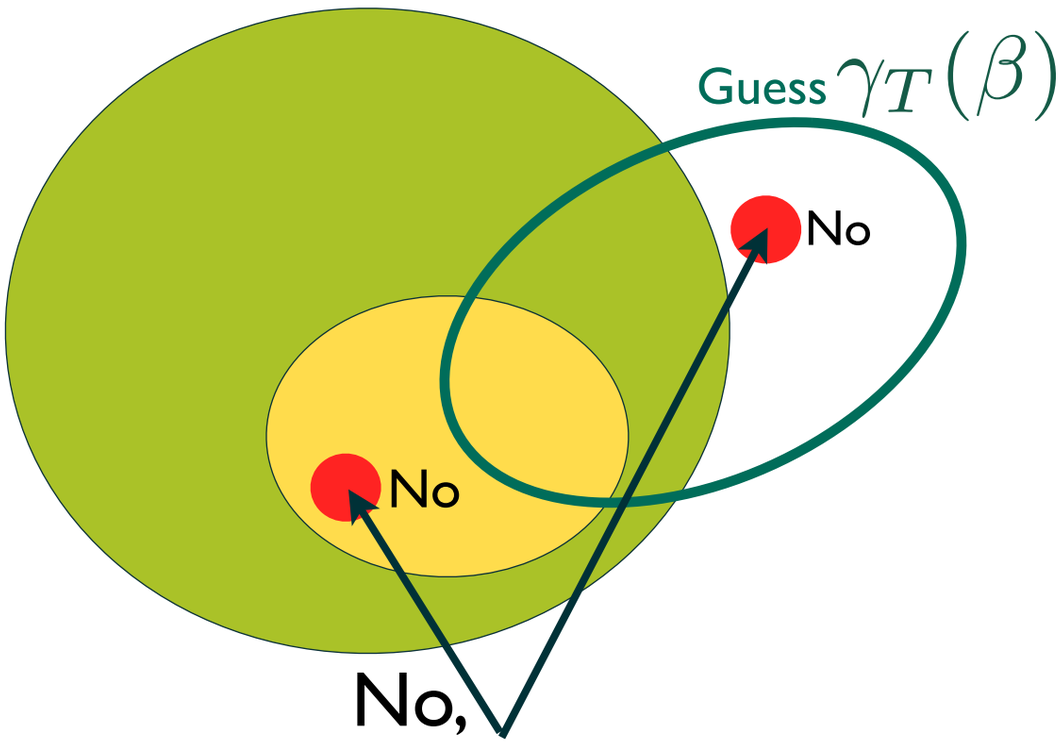
$$\mu \models \gamma_T(\beta) \oplus \bar{\iota}$$

Over Approximation Under Approximation

Resolving Equivalence Query $EQ(\beta)$

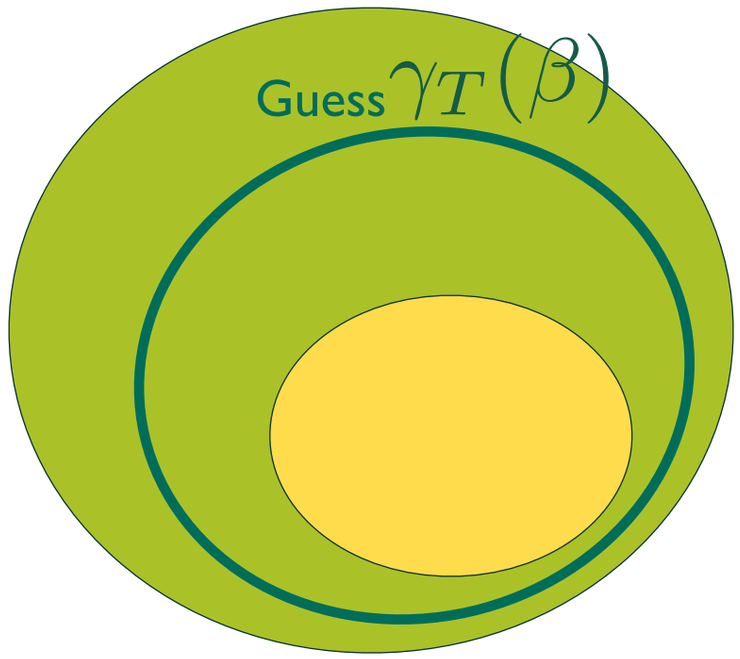
1. If $\gamma_T(\beta)$ is an invariants (check the three conditions), then "Yes"
2. Try to find a counter example.

Case 1



with found a counterexample μ.

Case 2 $\underline{\iota} \Rightarrow \gamma_T(\beta) \Rightarrow \bar{\iota}$



Restart!

Cannot find a counterexample.

Over Approximation Under Approximation

Resolving Membership Query $MEM(\mu)$

1. Check the well-formedness of the template T with $\gamma^*(\mu)$

   => Reject not well-formed template.

Resolving Membership Query $MEM(\mu)$

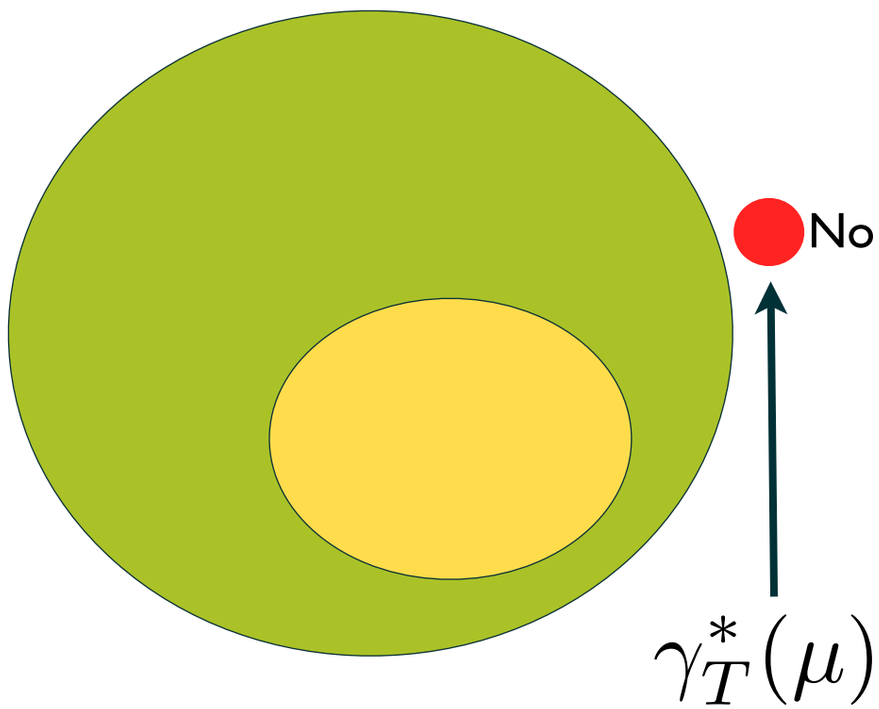
2. Use $\gamma_T^*(\mu)$ to answer membership query.
   Depends on the polarity of the template.

Resolving Membership Query $MEM(\mu)$

2. Use $\gamma_T^*(\mu)$ to answer membership query.

For positive template T

Case 1



Answer No

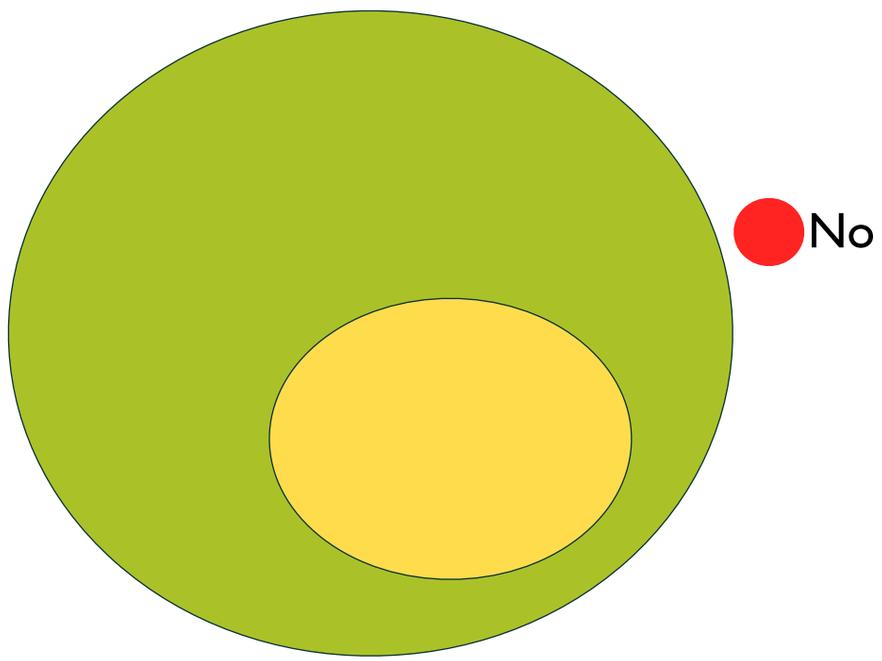
$S \models \gamma_T^*(\mu) \not\Rightarrow \bar{\iota}$

Resolving Membership Query $MEM(\mu)$

2. Use $\gamma_T^*(\mu)$ to answer membership query.

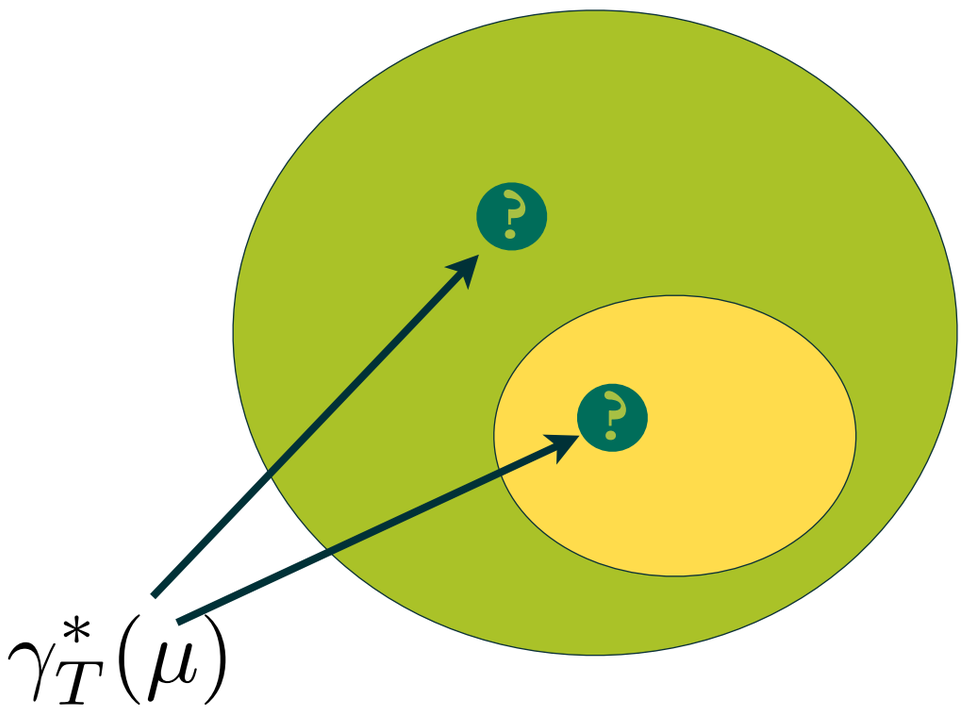
For positive template T

Case 1



Answer No

Case 2

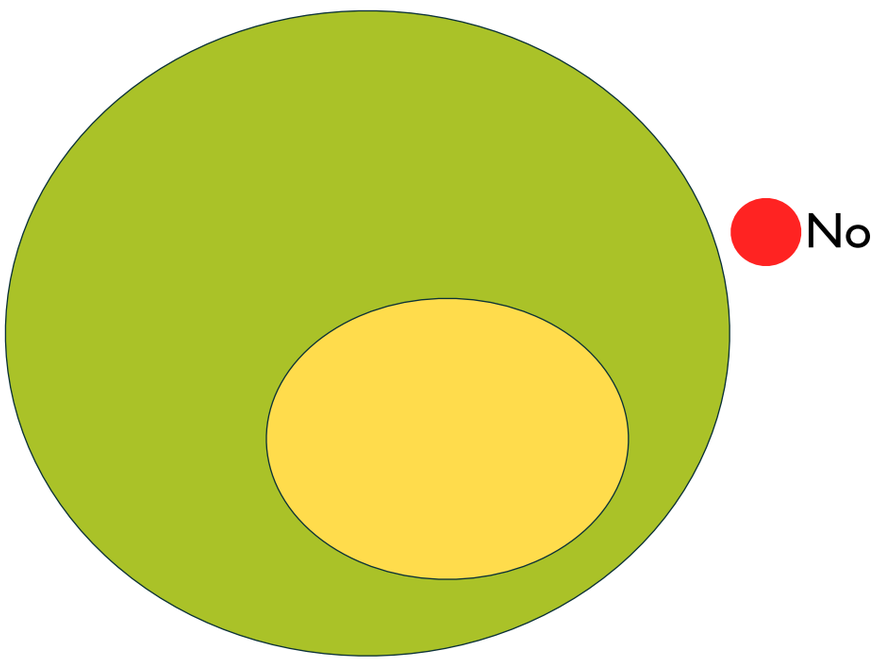


Resolving Membership Query $MEM(\mu)$

2. Use $\gamma_T^*(\mu)$ to answer membership query.

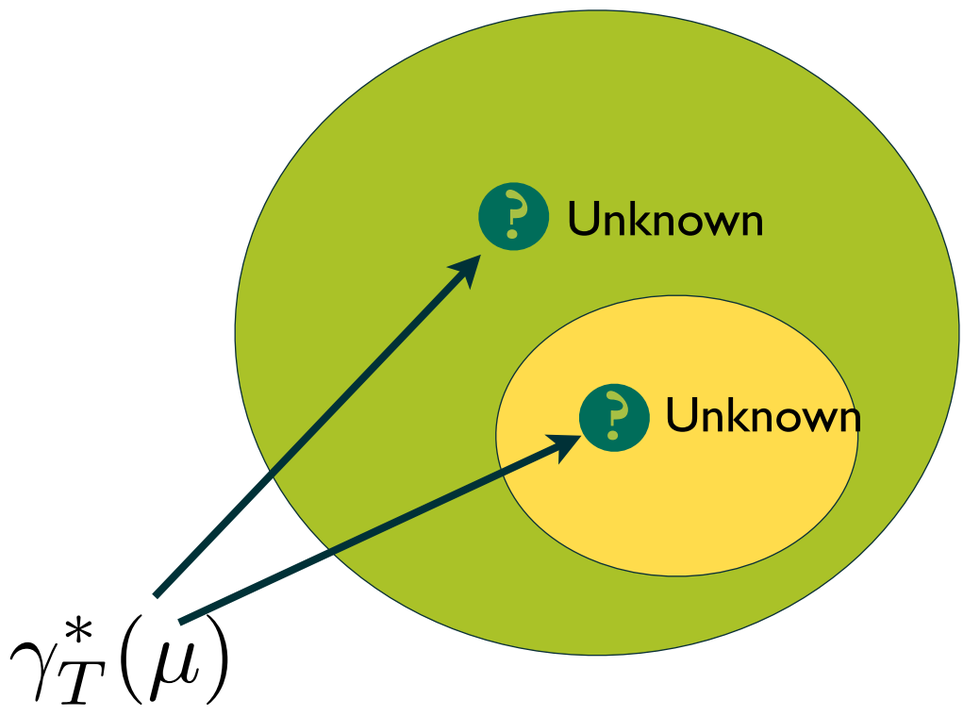
For positive template T

Case 1



Answer No

Case 2



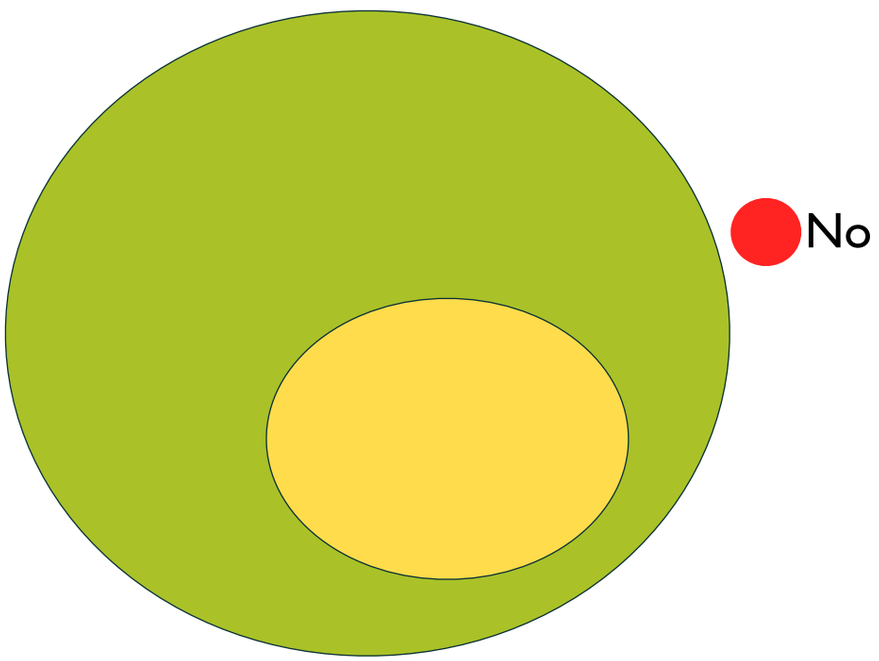
Cannot Answer!

Resolving Membership Query $MEM(\mu)$

2. Use $\gamma_T^*(\mu)$ to answer membership query.

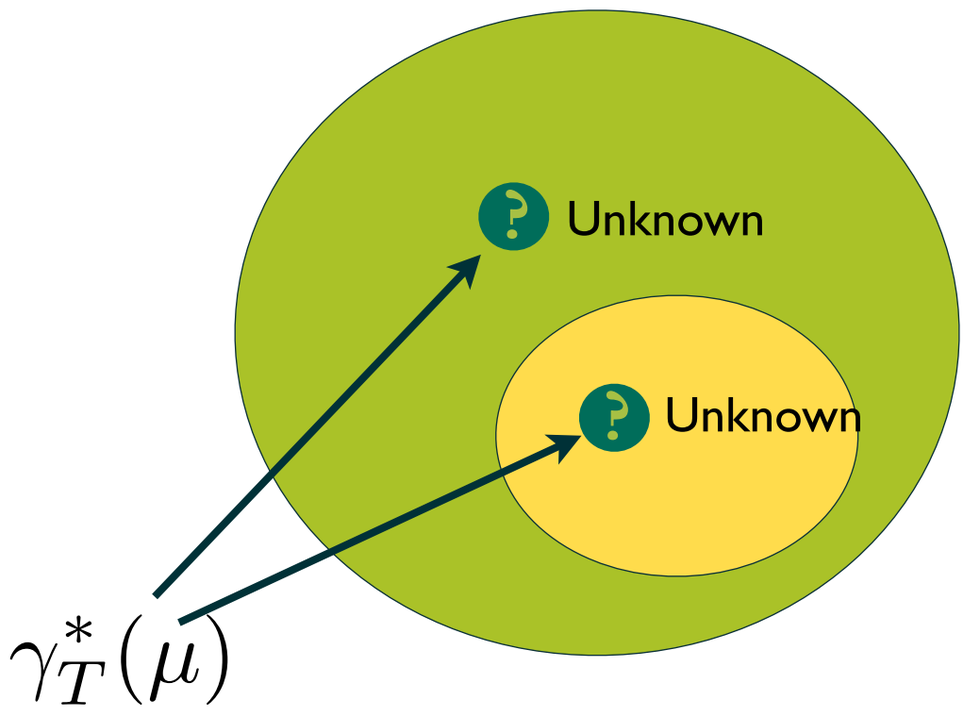
For positive template T

Case 1



Answer No

Case 2



Random Answer!

Implementation

- Basic examples work:
  - Array initialization program
  - Maximal element finding program
- Working on more complex examples
  - Selection Sort
  - Binary Search

Thank You