Part 2 :
DPLL(T) + Quantified Formulas

Andrew Reynolds

VTSA summer school

August 3, 2017

THE UNIVERSITY
OF IOWA

In this Talk

$$(\forall x.P(x) \lor f(b)=b+1) \land \exists y.(\neg P(y) \land f(y)<y)$$

- Focus on techniques for establishing *T-satisfiability* of formulas with:

In this Talk

$$(\forall x.P(x) \vee f(b)=b+1) \wedge \exists y.(\neg P(y) \wedge f(y)<y)$$

- Focus on techniques for establishing *T-satisfiability* of formulas with:
  - Boolean structure

In this Talk

$$(\forall x.P(x) \lor f(b)=b+1) \land \exists y.(\neg P(y) \land f(y)<y)$$

- Focus on techniques for establishing *T-satisfiability* of formulas with:
  - Boolean structure
  - Constraints in a background theory T, e.g. UFLIA

In this Talk

$$(\forall x.P(x) \vee f(b)=b+1) \wedge \exists y.(\neg P(y) \wedge f(y)<y)$$

- Focus on techniques for establishing *T-satisfiability* of formulas with:
  - Boolean structure
  - Constraints in a background theory T, e.g. UFLIA
  - ***Existential and Universal Quantifiers***

Quantified formulas ∀ in SMT

- Are of importance to applications:
  - Automated theorem proving:
    - Background axioms $\{\forall x.g(e,x)=g(x,e)=x,\ \forall x.g(x,g(y,z))=g(g(x,y),x), \forall x.g(x,i(x))=e\}$
  - Software verification:
    - Unfolding $\forall x.foo(x)=bar(x+1)$, code contracts $\forall x.pre(x)\Rightarrow post(f(x))$
    - Frame axioms $\forall x.x\neq t \Rightarrow A'(x)=A(x)$
  - Function Synthesis: $\forall i:input.\exists o:output.R[o,i]$
  - Planning: $\exists p:plan.\forall t:time.F[P,t]$
  - Knowledge representation: $\forall xy:Person.sibling(x,y)\Rightarrow mother(x)=mother(y)$

Quantified formulas ∀ in SMT

- Are of importance to applications:
  - Automated theorem proving:
    - Background axioms $\{\forall x.g(e,x)=g(x,e)=x,\ \forall x.g(x,g(y,z))=g(g(x,y),x), \forall x.g(x,i(x))=e\}$
  - Software verification:
    - Unfolding $\forall x.foo(x)=bar(x+1)$, code contracts $\forall x.pre(x) \Rightarrow post(f(x))$
    - Frame axioms $\forall x.x \neq t \Rightarrow A'(x)=A(x)$
  - Function Synthesis: $\forall i{:}input.\exists o{:}output.R[o,i]$
  - Planning: $\exists p{:}plan.\forall t{:}time.F[P,t]$
  - Knowledge representation: $\forall xy{:}Person.sibling(x,y) \Rightarrow mother(x)=mother(y)$
- Are very challenging in theory:
  - Establishing T-satisfiability of formulas with ∀ is generally undecidable

Quantified formulas $\forall$ in SMT

- Are of importance to applications:
  - Automated theorem proving:
    - Background axioms $\{\forall x.g(e,x)=g(x,e)=x,\ \forall x.g(x,g(y,z))=g(g(x,y),x), \forall x.g(x,i(x))=e\}$
  - Software verification:
    - Unfolding $\forall x.foo(x)=bar(x+1)$, code contracts $\forall x.pre(x)\Rightarrow post(f(x))$
    - Frame axioms $\forall x.x\neq t \Rightarrow A'(x)=A(x)$
  - Function Synthesis: $\forall i{:}input.\exists o{:}output.R[o,i]$
  - Planning: $\exists p{:}plan.\forall t{:}time.F[P,t]$
  - Knowledge representation: $\forall xy{:}Person.sibling(x,y)\Rightarrow mother(x)=mother(y)$
- Are very challenging in theory:
  - Establishing T-satisfiability of formulas with $\forall$ is generally undecidable
- Can be handled well in practice:
  - Efficient decision procedures for decidable fragments
  - Heuristic techniques have high success rates in the general case

Quantifiers

- Universal quantification:

$$\underbrace{\forall \texttt{x:Int.P(x)}}_{\text{P is true for all integers x}}$$

- Existential quantification:

$$\underbrace{\exists \texttt{x:Int.}\neg\texttt{Q(x)}}_{\text{Q is false for some integer x}}$$

Quantifiers

- Universal quantification:

$$\underbrace{\forall x\texttt{:Int.P(x)}}_{\text{P is true for all integers x}}$$

- Existential quantification:

$$\exists x\texttt{:Int.}\neg\texttt{Q(x)} \quad \rightarrow \quad \text{Ė?}\mathbf{x\texttt{:Int.Q(x)}}$$

⇒ For consistency, assume existential quantification is rewritten as universal quantification

Solvers for ∀

- First order theorem provers focus on ∀ reasoning
  …but have been extended in the past decade to theory reasoning

- SMT solvers focus mostly on quantifier-free theory reasoning
  …but have been extended in the past decade to ∀ reasoning

Solvers for ∀

- First order theorem provers focus on ∀ reasoning

  …but have been extended in the past decade to theory reasoning:

  - **Vampire, E, SPASS**
    - First-order resolution + superposition [Robinson 65, Nieuwenhuis/Rubio 99, Prevosto/Waldman 06]
    - AVATAR [Voronkov 14, Reger et al 15]
  - **iProver**
    - InstGen calculus [Ganzinger/Korovin 03]
  - **Princess, Beagle**, …
- SMT solvers focus mostly on quantifier-free theory reasoning

  …but have been extended in the past decade to ∀ reasoning

Solvers for ∀

- First order theorem provers focus on ∀ reasoning

  …but have been extended in the past decade to theory reasoning:

  - **Vampire, E, SPASS**
    - First-order resolution + superposition [Robinson 65, Nieuwenhuis/Rubio 99, Prevosto/Waldman 06]
    - AVATAR [Voronkov 14, Reger et al 15]
  - **iProver**
    - InstGen calculus [Ganzinger/Korovin 03]
  - **Princess, Beagle**, …
- SMT solvers focus mostly on quantifier-free theory reasoning

  …but have been extended in the past decade to ∀ reasoning:

  - **Z3, CVC4, VeriT, Alt-Ergo**
    - Some superposition-based [deMoura et al 09]
    - Mostly instantiation-based [Detlefs et al 05, deMoura et al 07, Ge et al 07, …]

Solvers for ∀

- First order theorem provers focus on ∀ reasoning

  …but have been extended in the past decade to theory reasoning:
  - **Vampire, E, SPASS**
    - First-order resolution + superposition **[Robinson 65, Nieuwenhuis/Rubio 99, Prevosto/Waldman 06]**
    - AVATAR **[Voronkov 14, Reger et al 15]**
  - **iProver**
    - InstGen calculus **[Ganzinger/Korovin 03]**
  - **Princess, Beagle**, …
- SMT solvers focus mostly on quantifier-free theory reasoning

  …but have been extended in the past decade to ∀ reasoning:
  - **Z3, CVC4, VeriT, Alt-Ergo**
    - Some superposition-based **[deMoura et al 09]**
    - Mostly **instantiation-based** **[Detlefs et al 05, deMoura et al 07, Ge et al 07, …]**

$\Rightarrow$ Focus of the first part of this talk

SMT Solvers for ∀ using Quantifier Instantiation

- Traditionally:
  - E-matching [Detlefs et al 2005,Bjorner et al 2007, Ge et al 2007]

Implemented in

simplify, cvc3, z3, FX7, Alt-Ergo, Princess, cvc4, veriT

SMT Solvers for ∀ using Quantifier Instantiation

- Traditionally:
  - E-matching **[Detlefs et al 2005,Bjorner et al 2007, Ge et al 2007]**
- More recently:
  - Model-Based Instantiation **[Ge et al 2009, Reynolds et al 2013]**
  - Conflict-Based Instantiation **[Reynolds et al 2014, Barbosa et al 2017]**
  - Theory-specific Approaches
    - Linear arithmetic **[Bjorner 2012, Reynolds et al 2015, Janota et al 2015]**
    - Bit-Vectors **[Wintersteiger et al 2013, Dutertre 2015]**

Implemented in

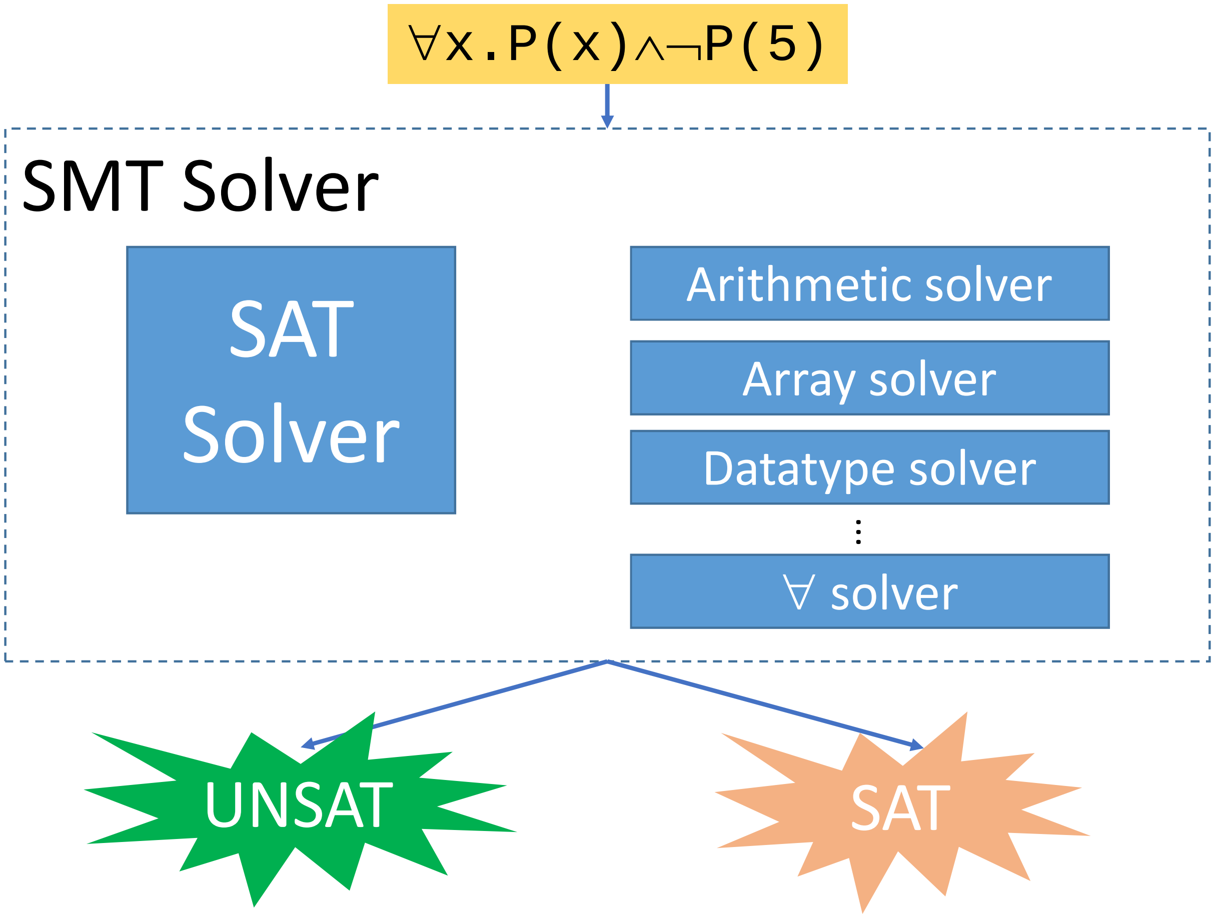
simplify, cvc3, z3, FX7, Alt-Ergo, Princess, cvc4, veriT

z3, cvc4

cvc4, veriT

z3, cvc4, yices, veriT+redlog

Satisfiability Modulo Theories (SMT) Solvers



$\forall x.P(x) \land \neg P(5)$

SMT Solver

SAT Solver

Arithmetic solver

Array solver

Datatype solver

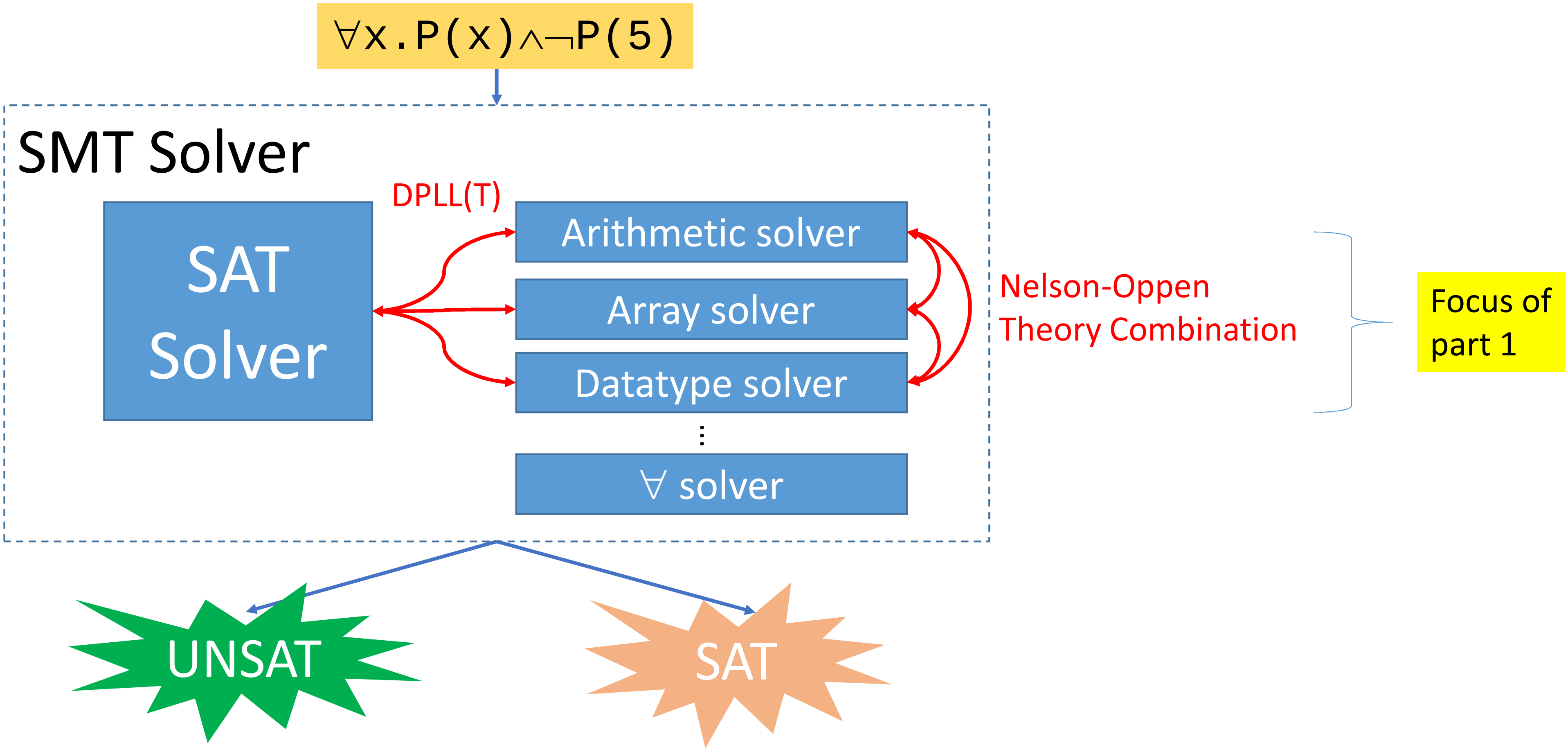
⋮

$\forall$ solver

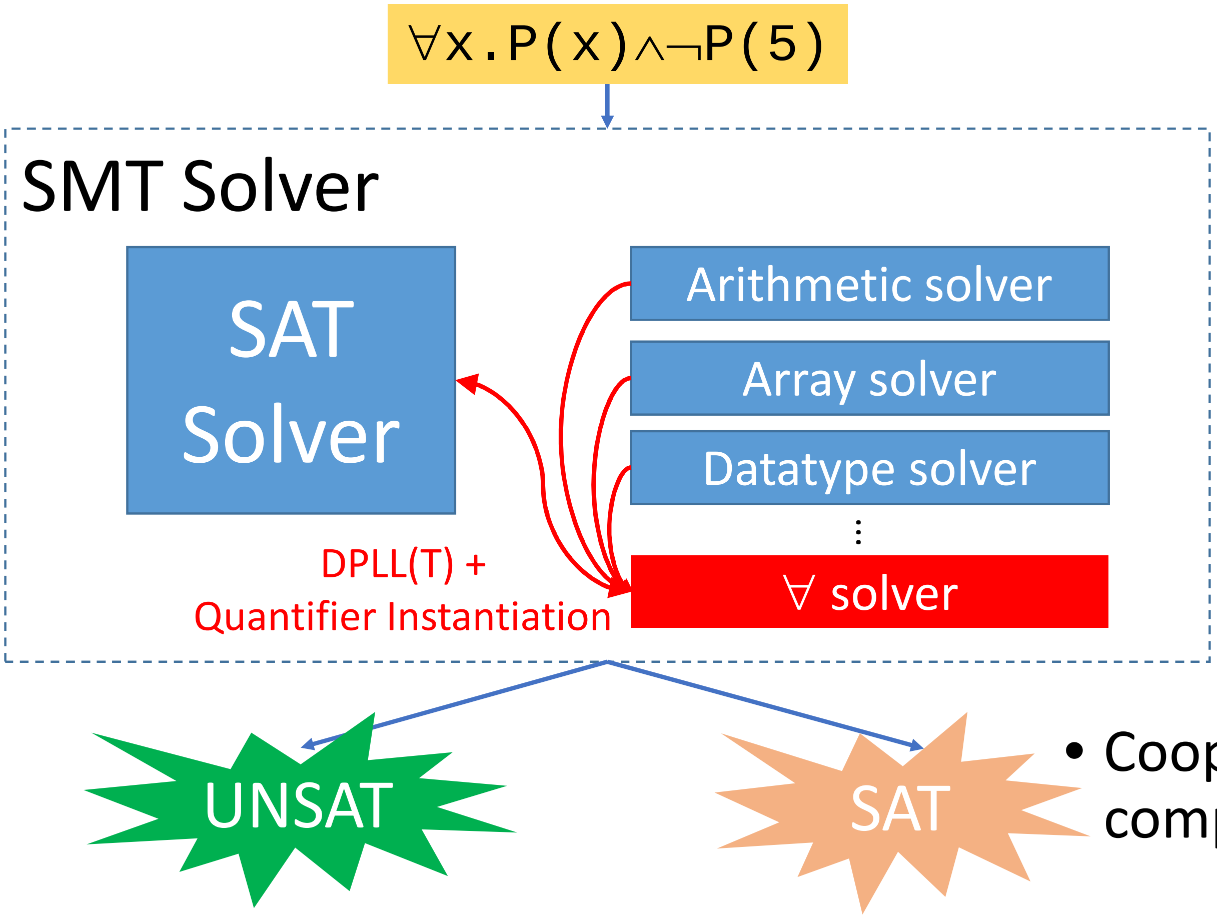
UNSAT

SAT

Satisfiability Modulo Theories (SMT) Solvers



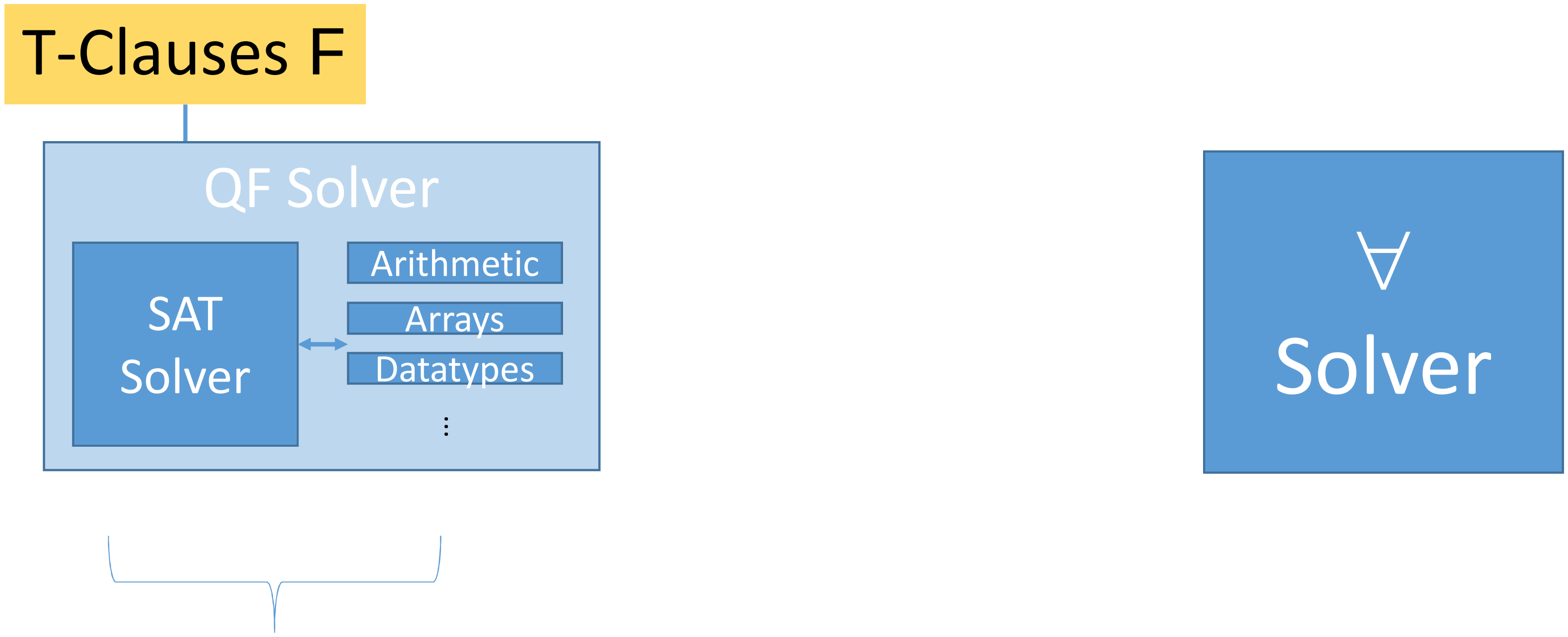
Satisfiability Modulo Theories (SMT) Solvers



Focus of part 2

- Cooperative interaction between components

DPLL(T)-Based SMT Solvers + ∀ Instantiation



T-Clauses F

QF Solver

SAT Solver

Arithmetic

Arrays

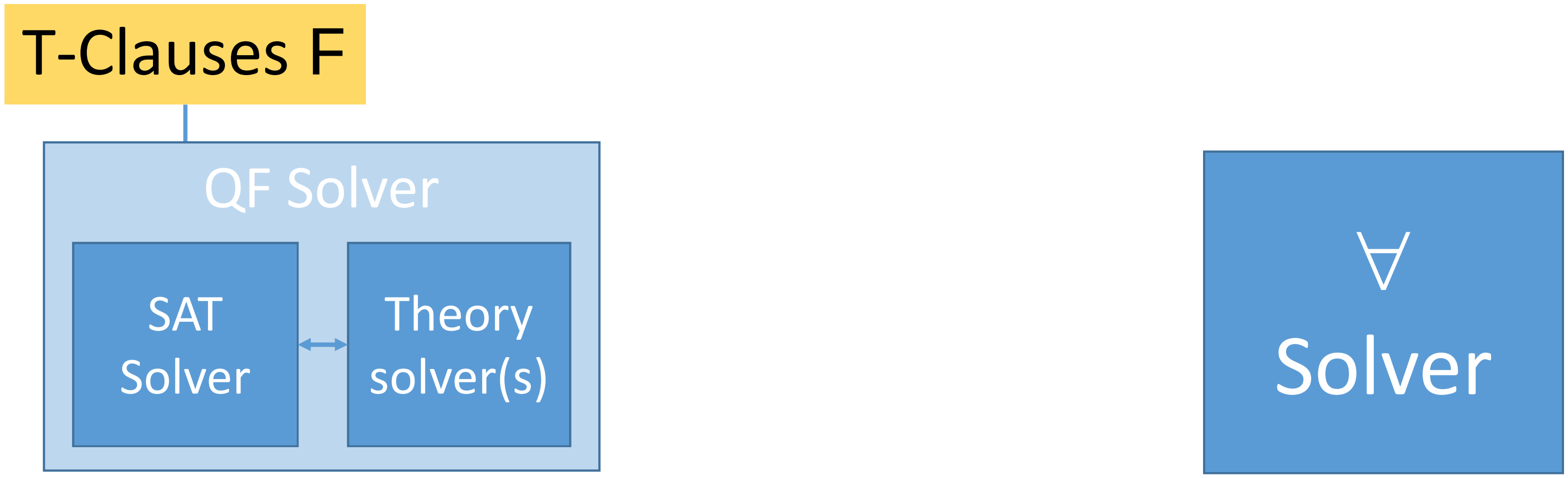
Datatypes

⋮

∀ Solver

- Portion of SMT solver that focuses on quantifier-free reasoning (treats quantified formulas as propositional variables)

DPLL(T)-Based SMT Solvers + ∀ Instantiation



T-Clauses F

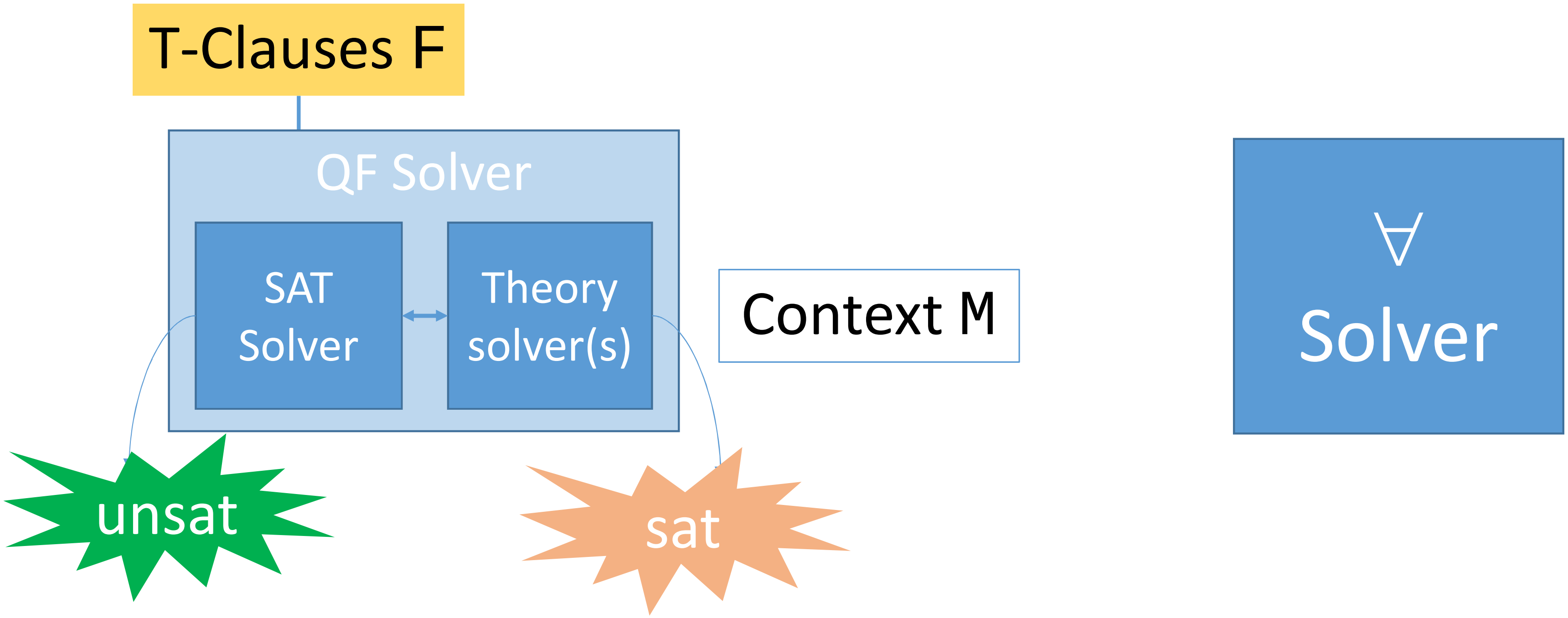
QF Solver

SAT Solver

Theory solver(s)

∀ Solver

DPLL(T)-Based SMT Solvers + ∀ Instantiation



T-Clauses F

QF Solver

SAT Solver

Theory solver(s)

Context M

unsat

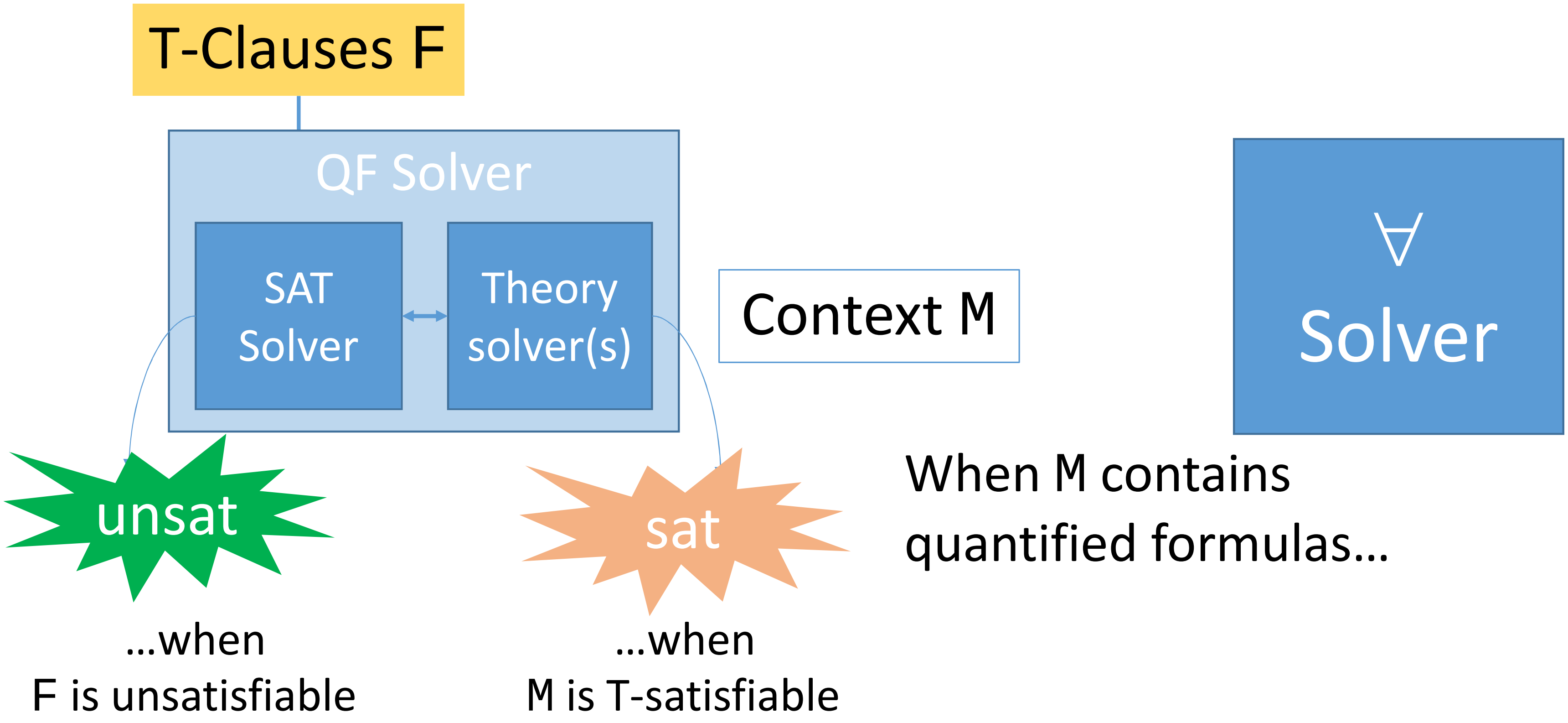
sat

∀ Solver

...when
F is unsatisfiable

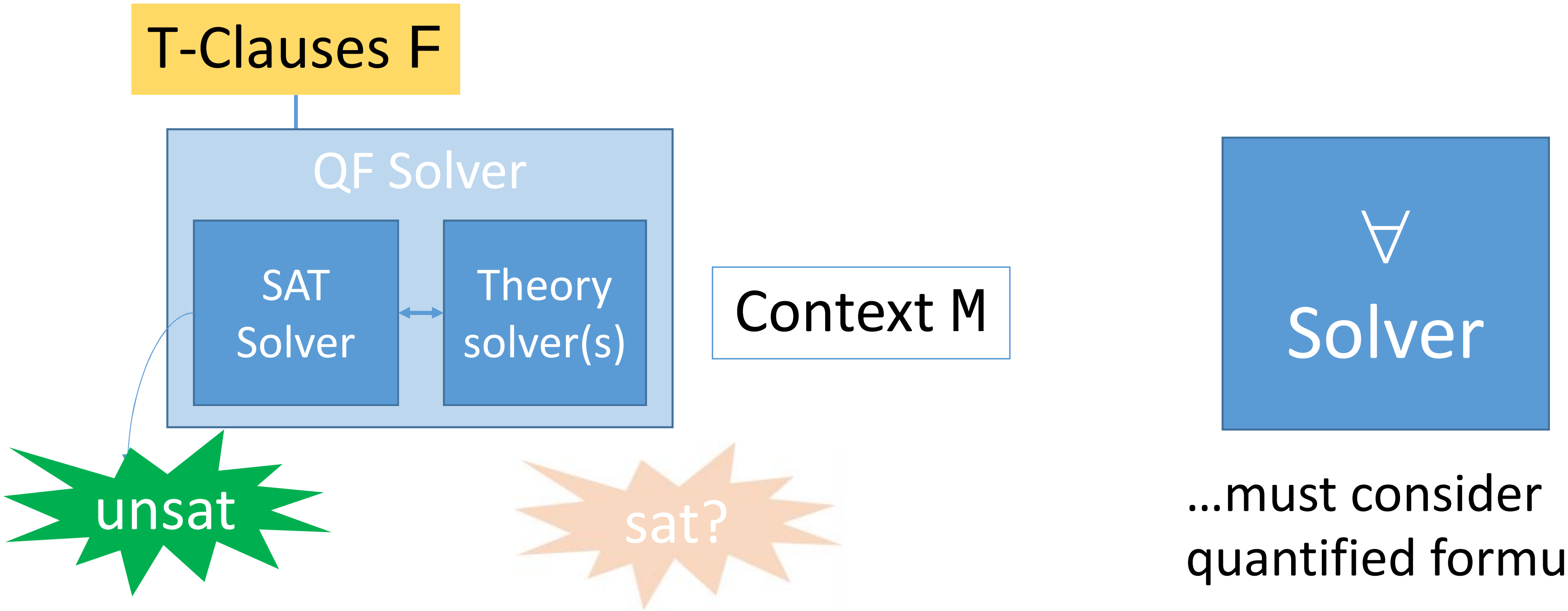
...when
M is T-satisfiable

DPLL(T)-Based SMT Solvers + ∀ Instantiation



When M contains quantified formulas…

DPLL(T)-Based SMT Solvers + ∀ Instantiation



T-Clauses F

QF Solver

SAT Solver

Theory solver(s)

Context M

∀ Solver

unsat

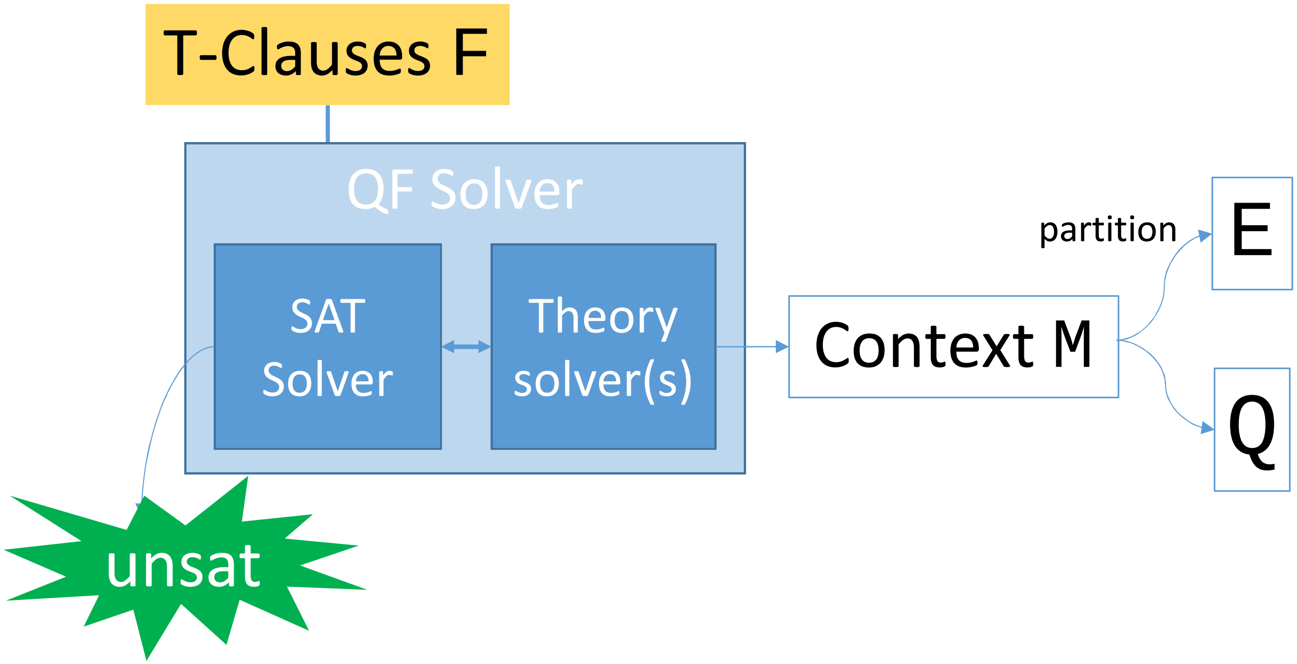
sat?

…when
F is unsatisfiable

…when
M is T-satisfiable

…must consider
quantified formulas in M

DPLL(T)-Based SMT Solvers + ∀ Instantiation



T-Clauses F

QF Solver

SAT Solver

Theory solver(s)

Context M

partition

E

Q

unsat

…when
F is unsatisfiable

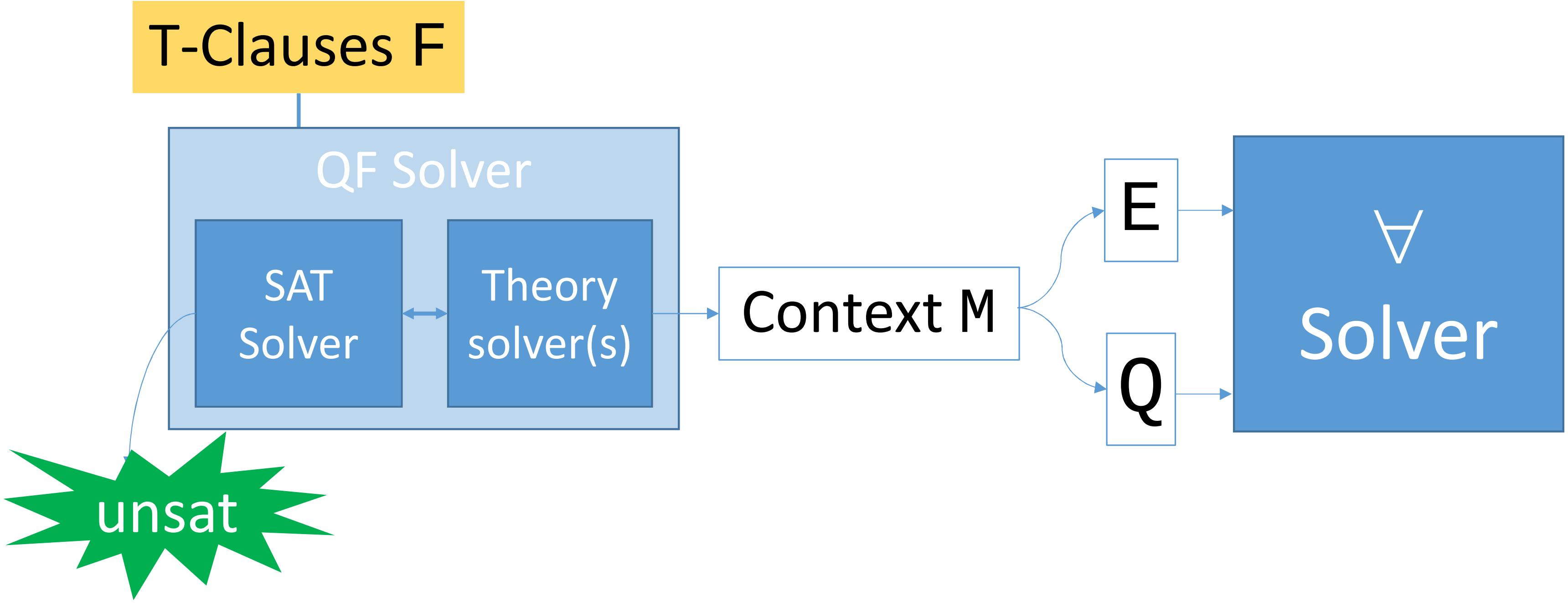
Set of ground equalities and disequalities

- $\{f(a)=b, P(a)=\bot, \ldots\}$

Set of quantified formulas

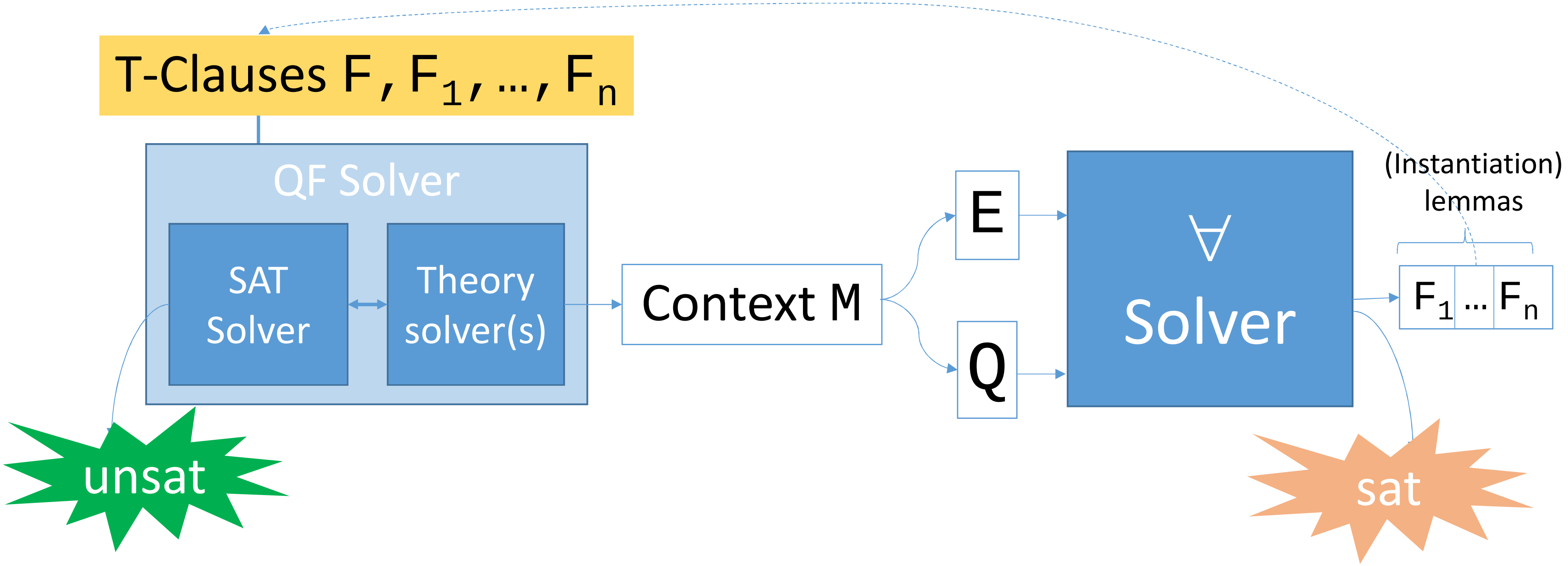
- $\{\forall x.P(x), \ldots\}$

DPLL(T)-Based SMT Solvers + ∀ Instantiation



…when
F is unsatisfiable

DPLL(T)-Based SMT Solvers + ∀ Instantiation



T-Clauses F, F_1, …, F_n

QF Solver

SAT Solver

Theory solver(s)

Context M

E

Q

∀ Solver

(Instantiation) lemmas

F_1 … F_n

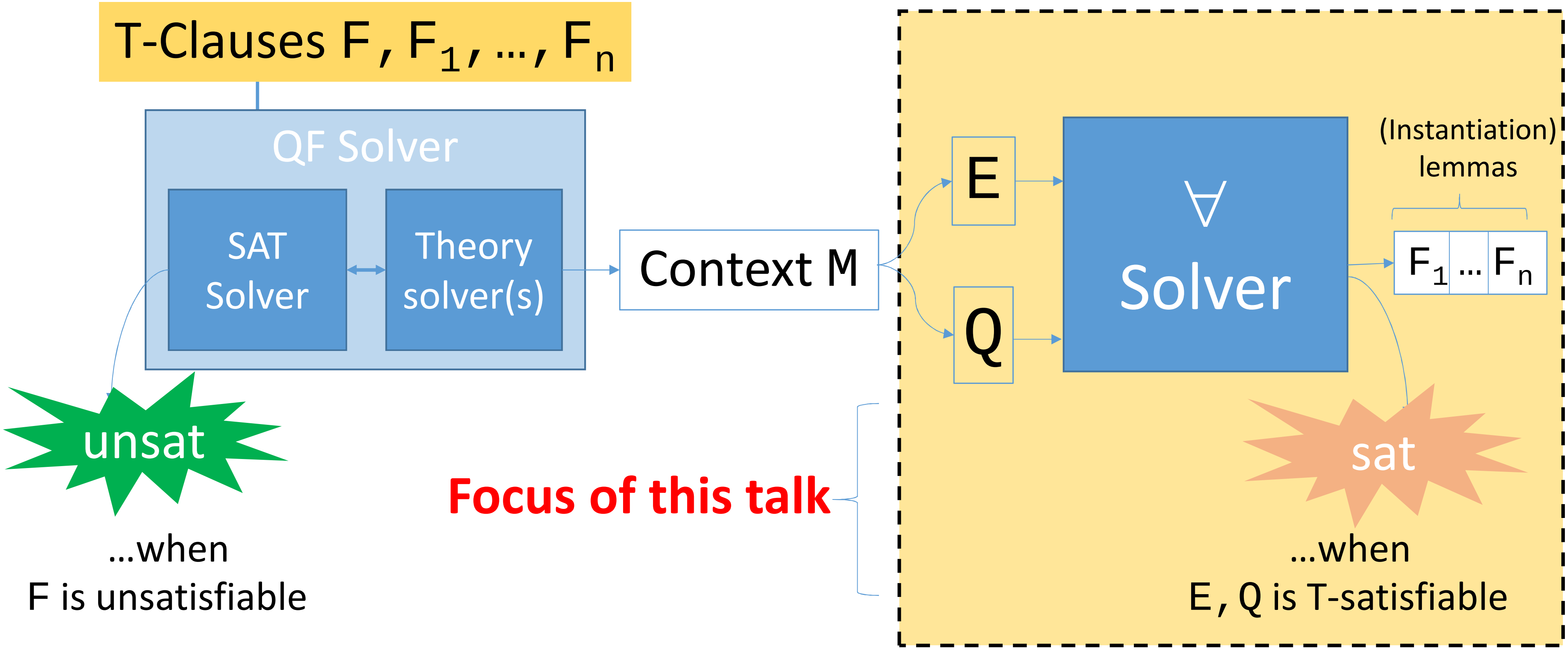
unsat

sat

…when
F is unsatisfiable

…when
E, Q is T-satisfiable

DPLL(T)-Based SMT Solvers + ∀ Instantiation



T-Clauses F, F_1, …, F_n

QF Solver

SAT Solver

Theory solver(s)

Context M

E

Q

∀ Solver

(Instantiation) lemmas

F_1 … F_n

unsat

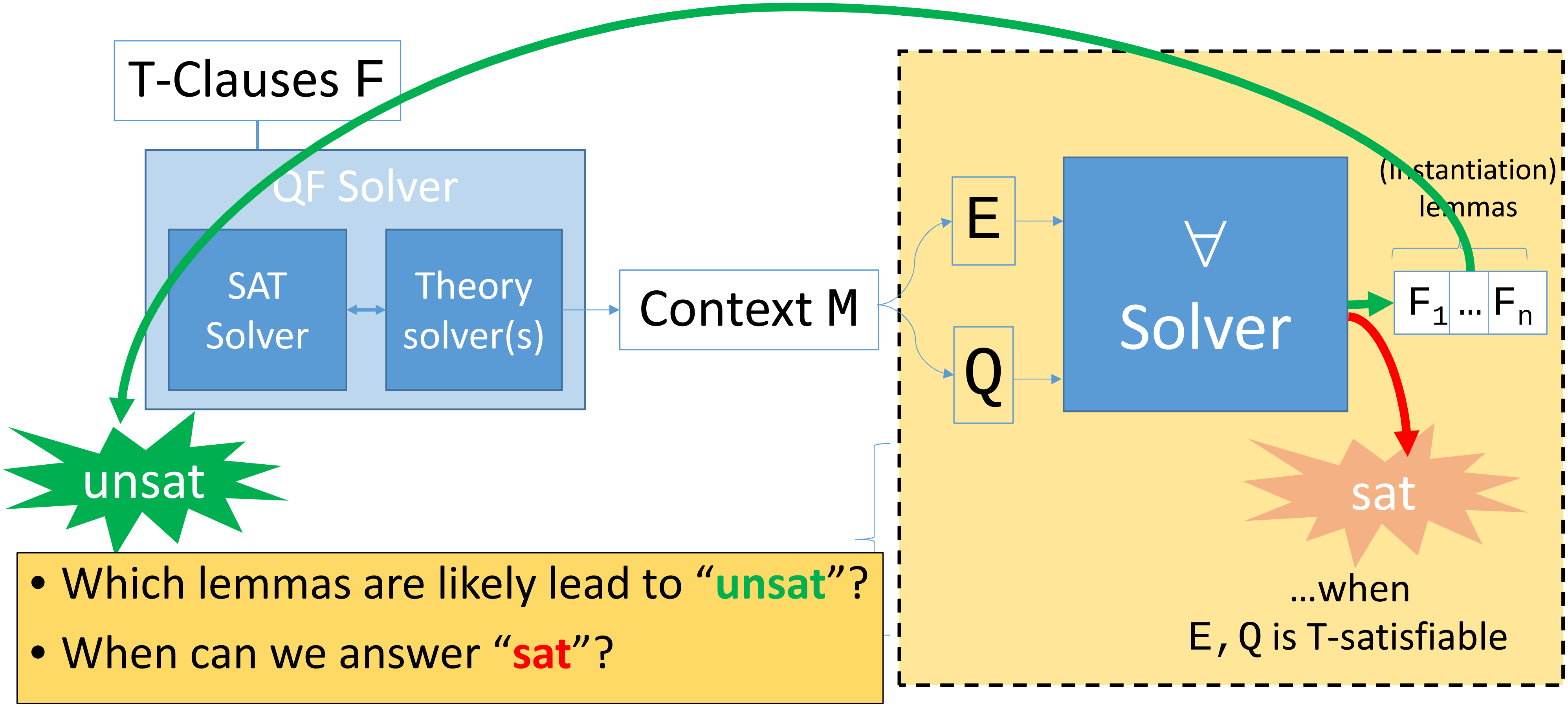
…when F is unsatisfiable

sat

…when E, Q is T-satisfiable

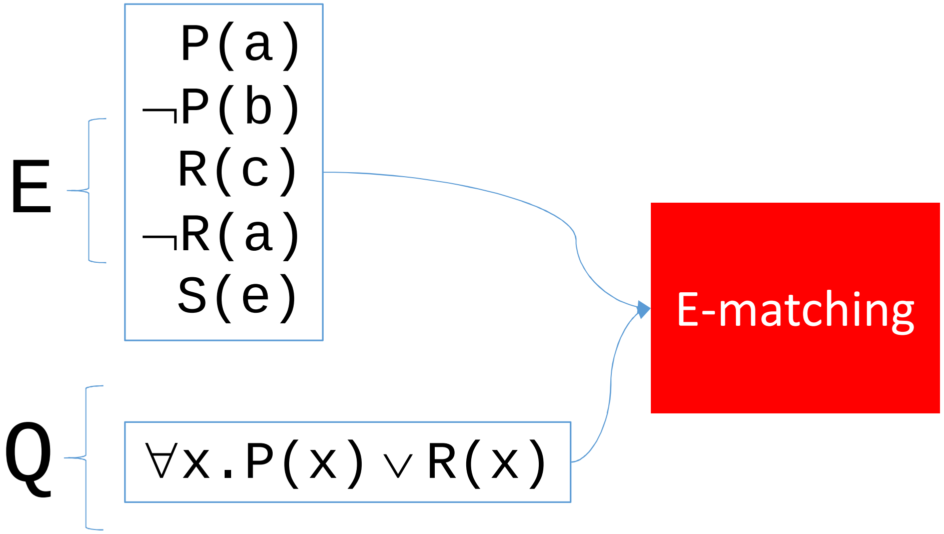
Focus of this talk

DPLL(T)-Based SMT Solvers + ∀ Instantiation



- Which lemmas are likely lead to “**unsat**”?
- When can we answer “**sat**”?

E-matching



E

P(a)
¬P(b)
R(c)
¬R(a)
S(e)

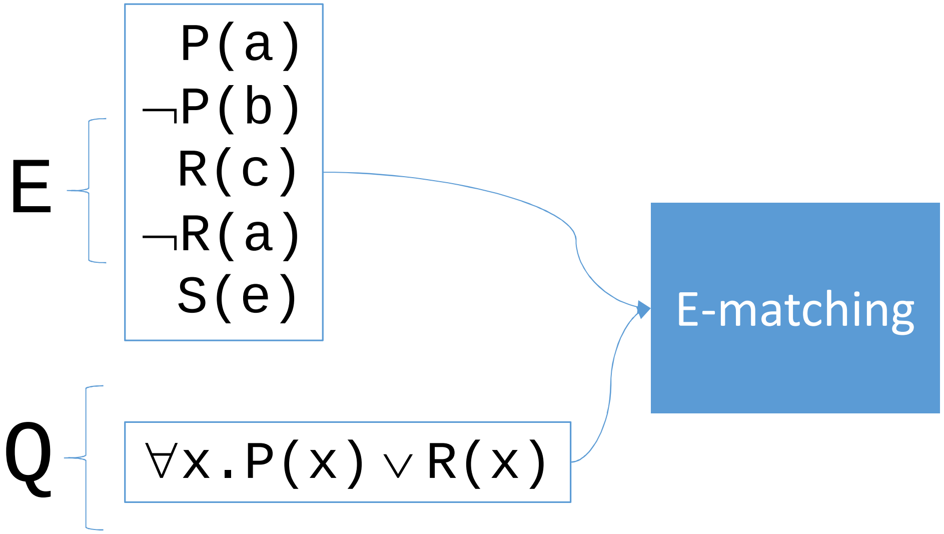
Q

∀x.P(x) ∨ R(x)

E-matching

- Introduced in Nelson’s Phd Thesis [Nelson 80]

E-matching



E

P(a)

¬P(b)

R(c)

¬R(a)

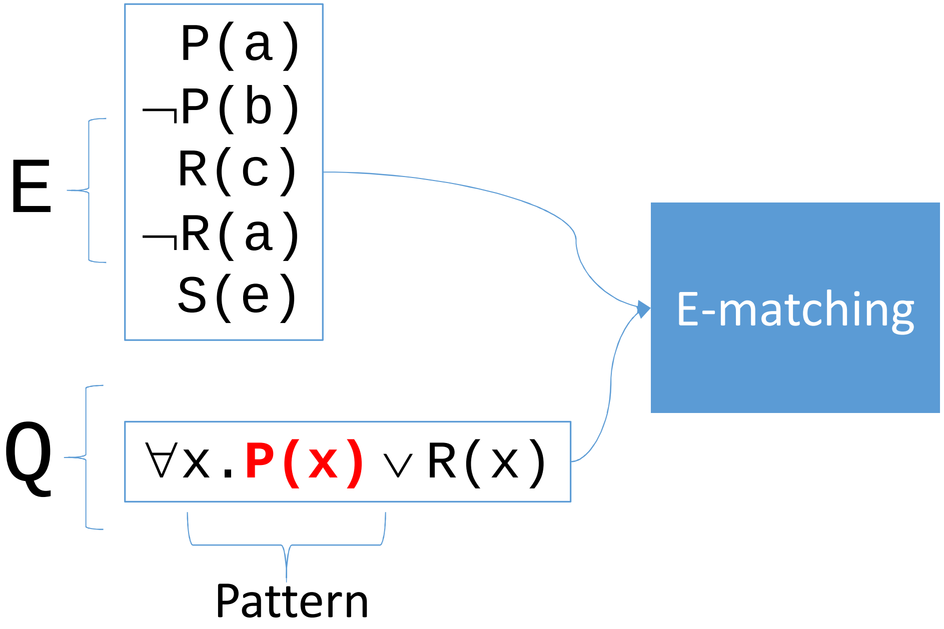
S(e)

Q

∀x.P(x) ∨ R(x)

E-matching

E-matching



E

$P(a)$

$\neg P(b)$

$R(c)$

$\neg R(a)$

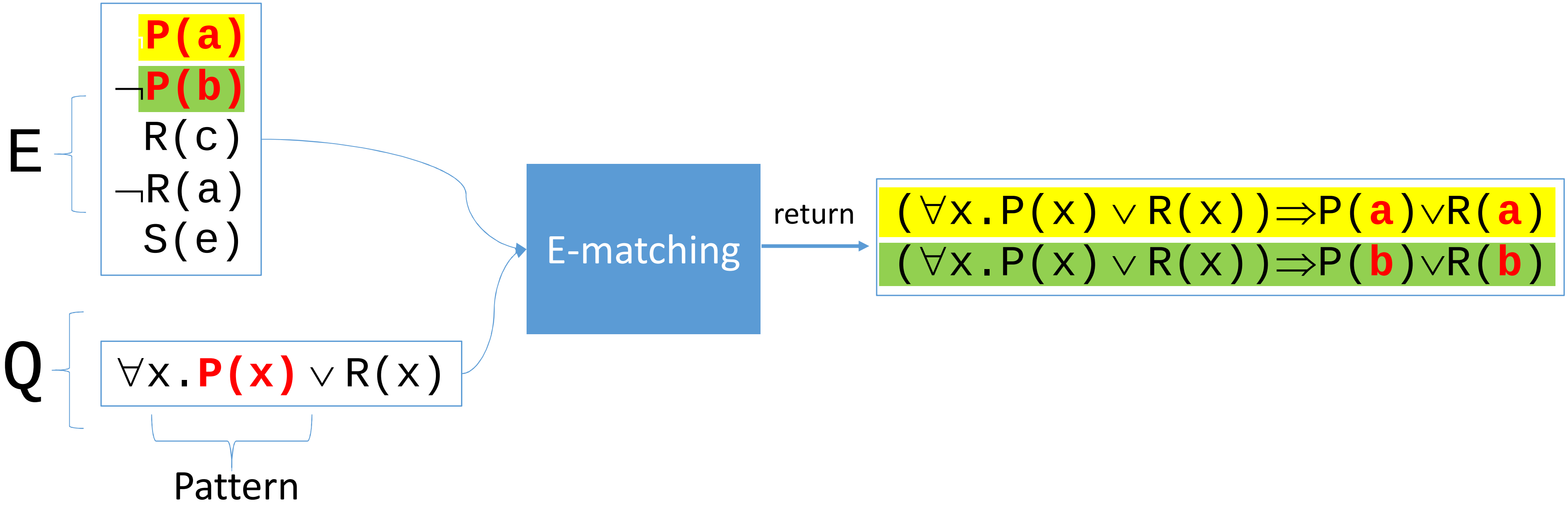
$S(e)$

Q

$\forall x. \underbrace{P(x)}_{\text{Pattern}} \vee R(x)$

Ĝ **Idea**: choose instances based on pattern matching

E-matching



E

P(a)

¬P(b)

R(c)

¬R(a)

S(e)

Q

∀x.P(x) ∨ R(x)

Pattern

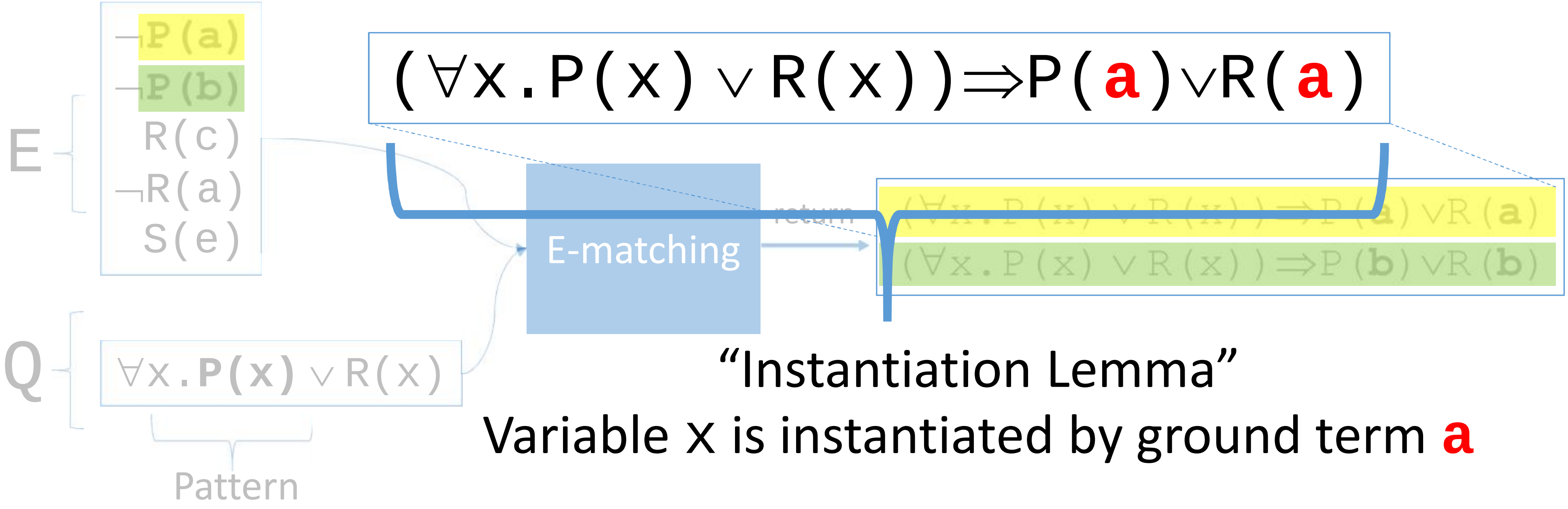
E-matching

return

(∀x.P(x) ∨ R(x))⇒P(a)∨R(a)

(∀x.P(x) ∨ R(x))⇒P(b)∨R(b)

E-matching



E

$\neg$P(a)

$\neg$P(b)

R(c)

$\neg$R(a)

S(e)

Q

$\forall$x.P(x) $\vee$ R(x)

Pattern

E-matching

return

$(\forall x.P(x) \vee R(x)) \Rightarrow P(\mathbf{a}) \vee R(\mathbf{a})$

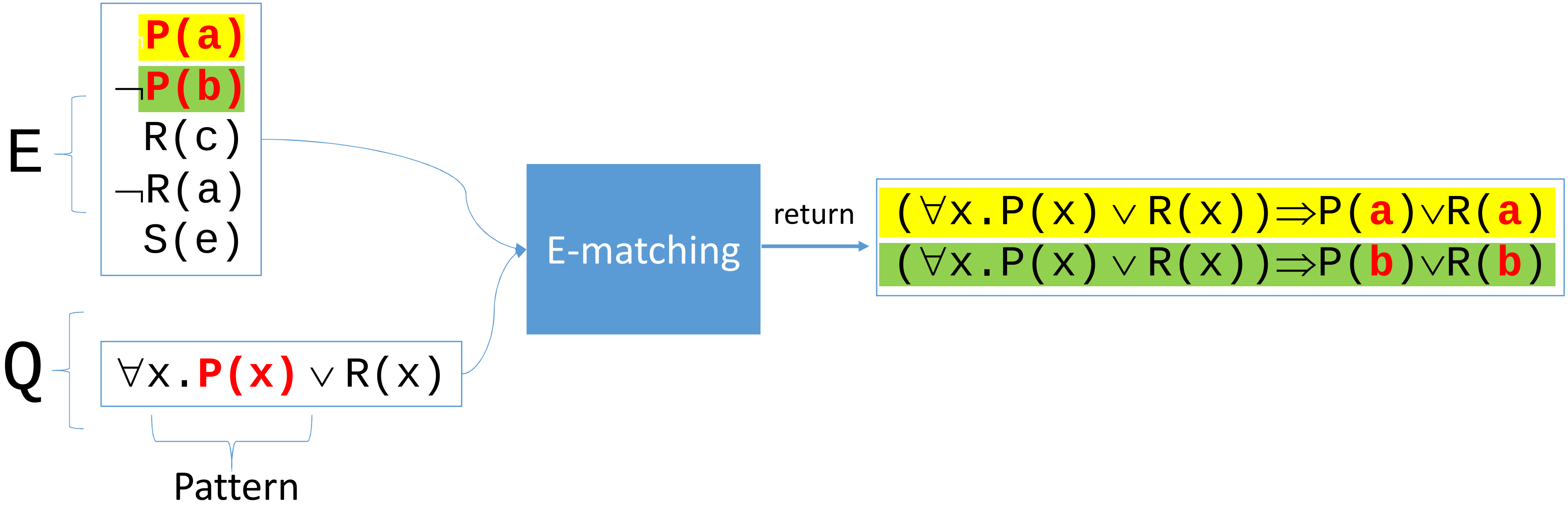
$(\forall x.P(x) \vee R(x)) \Rightarrow P(\mathbf{b}) \vee R(\mathbf{b})$

$$(\forall x.P(x) \vee R(x)) \Rightarrow P(\mathbf{a}) \vee R(\mathbf{a})$$

“Instantiation Lemma”

Variable x is instantiated by ground term **a**

E-matching



E

$P(a)$

$\neg P(b)$

$R(c)$

$\neg R(a)$

$S(e)$

Q

$\forall x. P(x) \vee R(x)$

Pattern

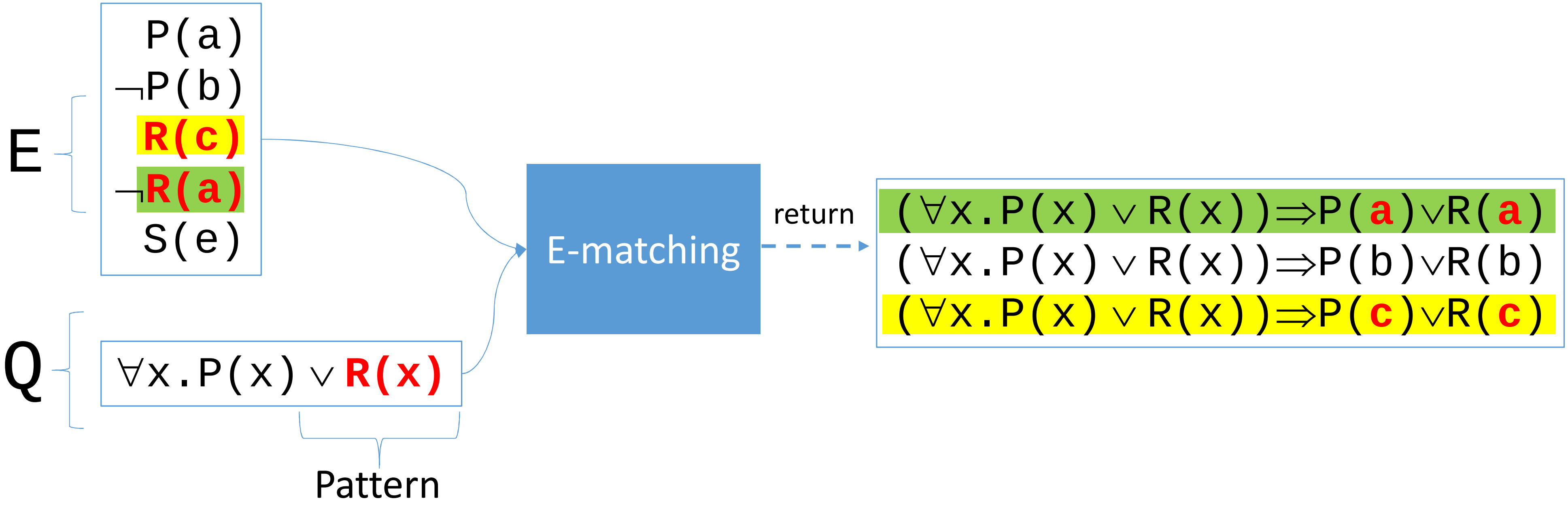
E-matching

return

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$

E-matching



E

P(a)

¬P(b)

R(c)

¬R(a)

S(e)

Q

∀x.P(x) ∨ R(x)

Pattern

E-matching

return

(∀x.P(x) ∨ R(x))⇒P(a)∨R(a)

(∀x.P(x) ∨ R(x))⇒P(b)∨R(b)

(∀x.P(x) ∨ R(x))⇒P(c)∨R(c)

Example

$\forall x.P(x) \land (\neg P(2) \lor \neg P(7))$

- DPLL(UFLIA) + E-Matching

Context

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true

Context

$\forall x.P(x)$

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Decide : $P(2) \rightarrow$ false

Context

$\forall x.P(x)$

$\neg P(2)^d$

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Decide : P(2) $\rightarrow$ false
  - Invoke UF solver for {$\neg P(2)$ }…UF-satisfiable

Context

$\forall x.P(x)$

$\neg P(2)^d$

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Decide : $P(2) \rightarrow$ false
  - Invoke E-matching for E = { $\neg P(2)$ }, Q = { $\forall x.P(x)$ }

Context

$\forall x.P(x)$

$\neg P(2)^d$

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Decide : $P(2) \rightarrow$ false
  - Invoke E-matching for $E = \{ \neg P(2) \}$, $Q = \{ \forall x.\underbrace{P(x)}_{\text{Pattern}} \}$

Context

$\forall x.P(x)$

$\neg P(2)^{d}$

Example

$\forall x.P(x) \land (\neg P(2) \lor \neg P(7))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Decide : $P(2) \rightarrow$ false
  - Invoke E-matching for E = { $\neg P(2)$ }, Q = { $\forall x.P(x)$ }

matches

Context

$\forall x.P(x)$

$\neg P(2)^d$

Example

$$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7)) \wedge$$
$$(\neg \forall x.P(x) \vee P(2))$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Decide : $P(2) \rightarrow$ false
  - matches
  - Invoke E-matching for E = { $\neg P(2)$ }, Q = { $\forall x.P(x)$ }

    $\Rightarrow$ Return instantiation lemma ($\forall x.P(x) \Rightarrow P(2)$)

Context

$\forall x.P(x)$

$\neg P(2)^{d}$

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7)) \wedge$
$(\neg \forall x.P(x) \vee P(2))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x)$ → true
  - …Backtrack

Context

$\forall x.P(x)$

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7)) \wedge$
$(\neg \forall x.P(x) \vee P(2))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Propagate : $P(2) \rightarrow$ true

Context

$\forall x.P(x)$
$P(2)$

Example

$$\forall x.P(x) \land (\neg P(2) \lor \neg P(7)) \land (\neg \forall x.P(x) \lor P(2))$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Propagate : $P(2) \rightarrow$ true
  - Propagate : $P(7) \rightarrow$ false

Context

$\forall x.P(x)$

$P(2)$

$\neg P(7)$

Example

$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7)) \wedge$

$(\neg \forall x.P(x) \vee P(2))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Propagate : $P(2) \rightarrow$ true
  - Propagate : $P(7) \rightarrow$ false
  - Invoke E-matching for E = { $P(2), \neg P(7)$ }, Q = { $\forall x.P(x)$ }

Context

$\forall x.P(x)$

$P(2)$

$\neg P(7)$

Example

$$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7)) \wedge (\neg \forall x.P(x) \vee P(2))$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Propagate : $P(2) \rightarrow$ true
  - Propagate : $P(7) \rightarrow$ false
  - Invoke E-matching for E = { $P(2)$, $\neg P(7)$ }, Q = { $\forall x.P(x)$ }

matches

Context

$\forall x.P(x)$

$P(2)$

$\neg P(7)$

Example

$$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7)) \wedge (\neg\forall x.P(x) \vee P(2)) \wedge (\neg\forall x.P(x) \vee P(7))$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Propagate : $P(2) \rightarrow$ true
  - Propagate : $P(7) \rightarrow$ false

matches

  - Invoke E-matching for $E = \{ P(2), \neg P(7) \}$, $Q = \{ \forall x.P(x) \}$

$\Rightarrow$ Return $(\underbrace{\forall x.P(x) \Rightarrow P(2)}_{\text{(repeated, ignore)}})$, $(\forall x.P(x) \Rightarrow P(7))$

Context

$\forall x.P(x)$

$P(2)$

$\neg P(7)$

Example

$$\forall x.P(x) \wedge (\neg P(2) \vee \neg P(7)) \wedge$$
$$(\neg\forall x.P(x) \vee P(2)) \wedge \underbrace{(\neg\forall x.P(x) \vee P(7))}$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.P(x) \rightarrow$ true
  - Propagate : $P(2) \rightarrow$ true
  - Propagate : $P(7) \rightarrow$ false
  - Invoke E-matching for $E = \{ P(2), \neg P(7) \}$, $Q = \{ \forall x.P(x) \}$

    $\Rightarrow$ Return $(\forall x.P(x) \Rightarrow P(2))$, $(\forall x.P(x) \Rightarrow P(7))$

$\Rightarrow$ Conflicting clause!

…no decision to backtrack

$\Rightarrow$ Input is UFLIA-unsat

Context

$\forall x.P(x)$

$P(2)$

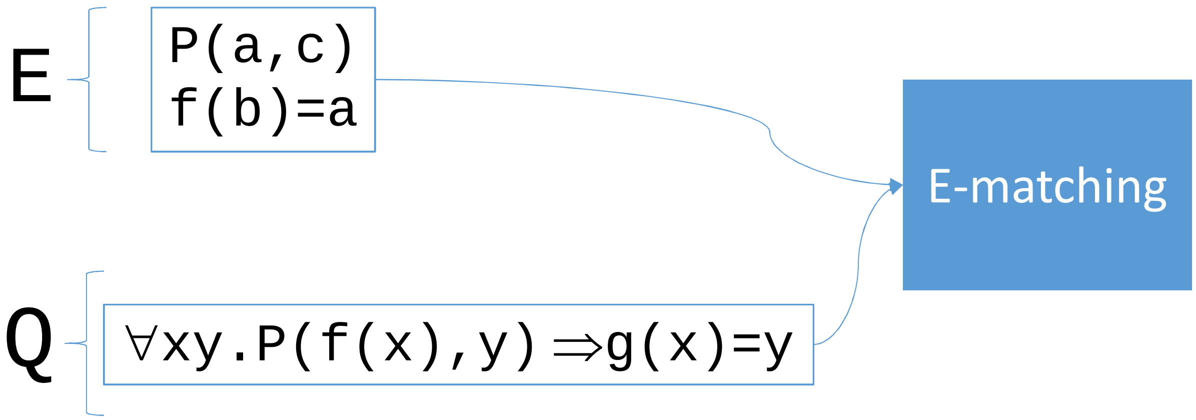
$\neg P(7)$

Encoding in *.smt2

```
(set-logic UFLIA)
(declare-fun P (Int) Bool)
(assert (forall ((x Int)) (P x)))
(assert (or (not (P 2)) (not (P 7))))
(check-sat)
```

EXAMPLE 1…

E-matching: Functions, Equality



E

P(a,c)

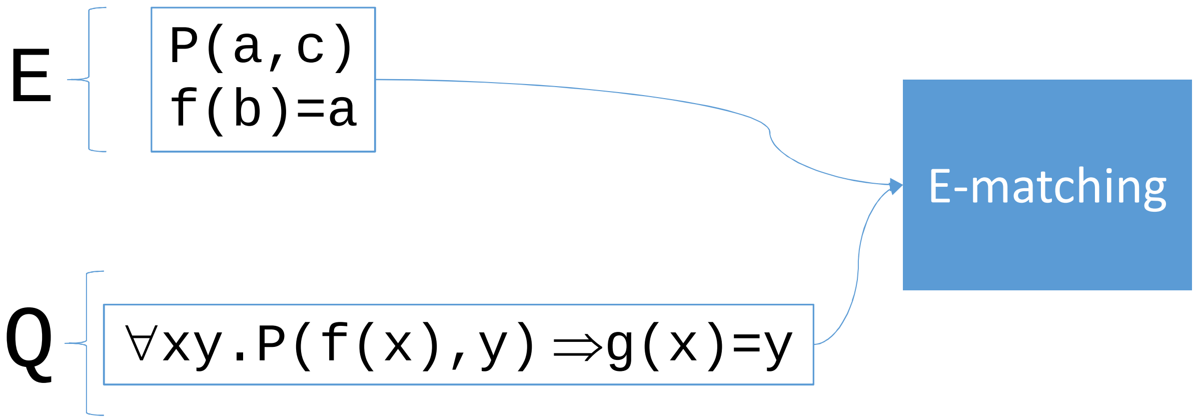
f(b)=a

Q

$\forall xy.P(f(x),y) \Rightarrow g(x)=y$

E-matching

E-matching: Functions, Equality



E

`P(a,c)`
`f(b)=a`

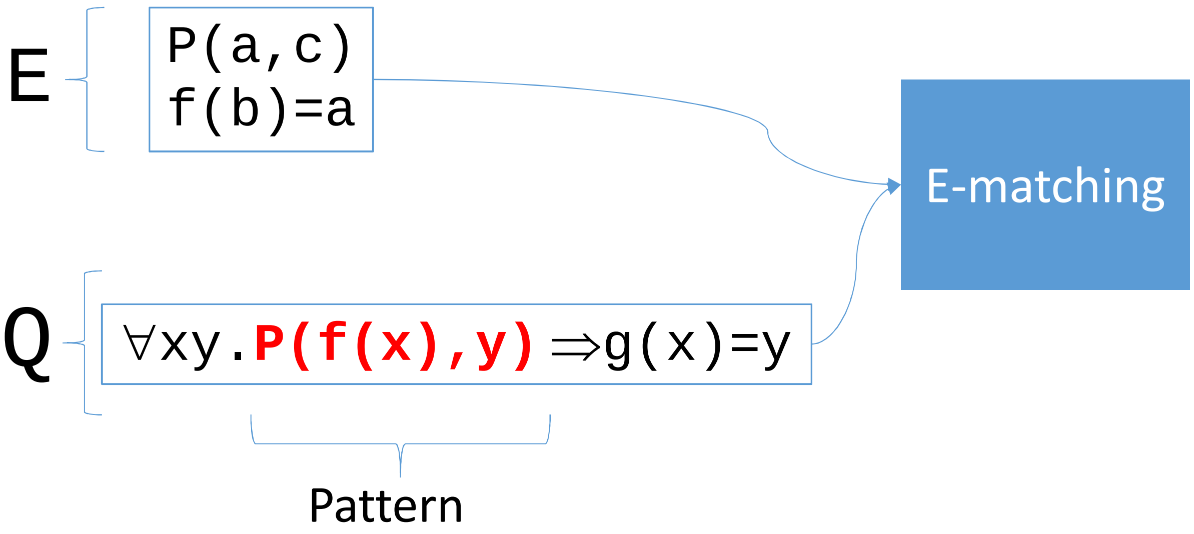
Q

`∀xy.P(f(x),y) ⇒g(x)=y`

E-matching

⇒ In E-matching, Pattern ***matching*** takes into account equalities in ***E***

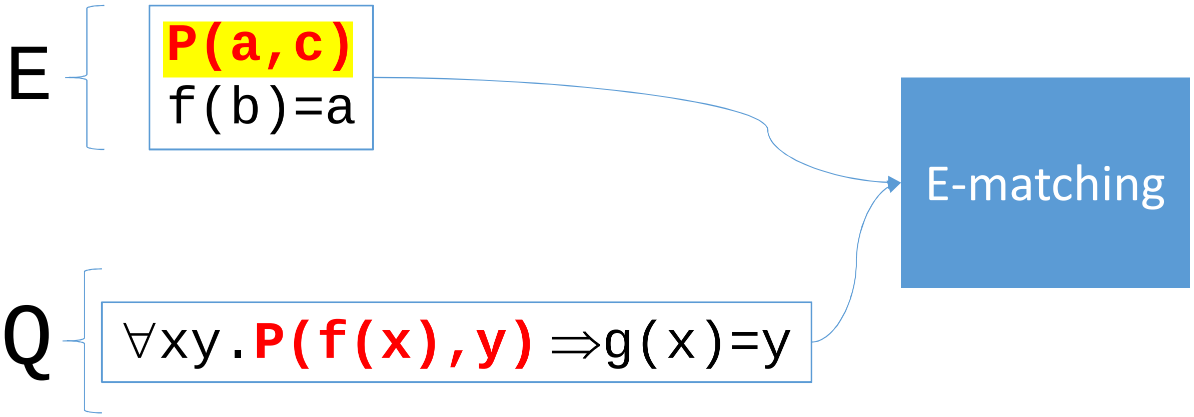
E-matching: Functions, Equality



```
P(a,c)
f(b)=a
```

```
∀xy.P(f(x),y) ⇒g(x)=y
```

E-matching: Functions, Equality



E

P(a,c)

f(b)=a

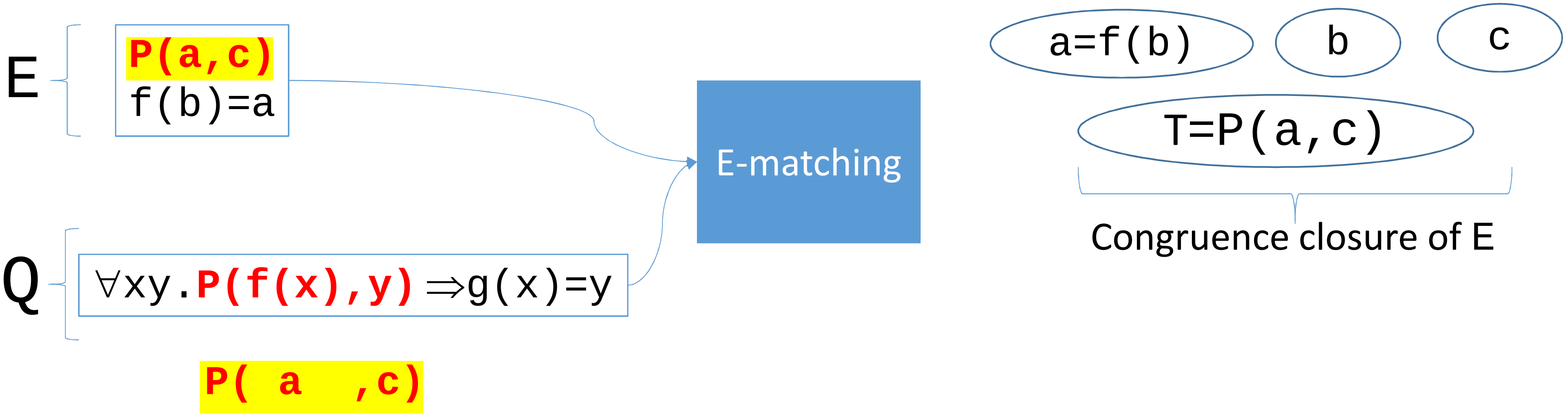
E-matching

Q

∀xy.**P(f(x),y)** ⇒g(x)=y

P(a ,c)

E-matching: Functions, Equality



E

P(a,c)

f(b)=a

Q

∀xy.P(f(x),y) ⇒g(x)=y

P(a ,c)

E-matching

a=f(b)

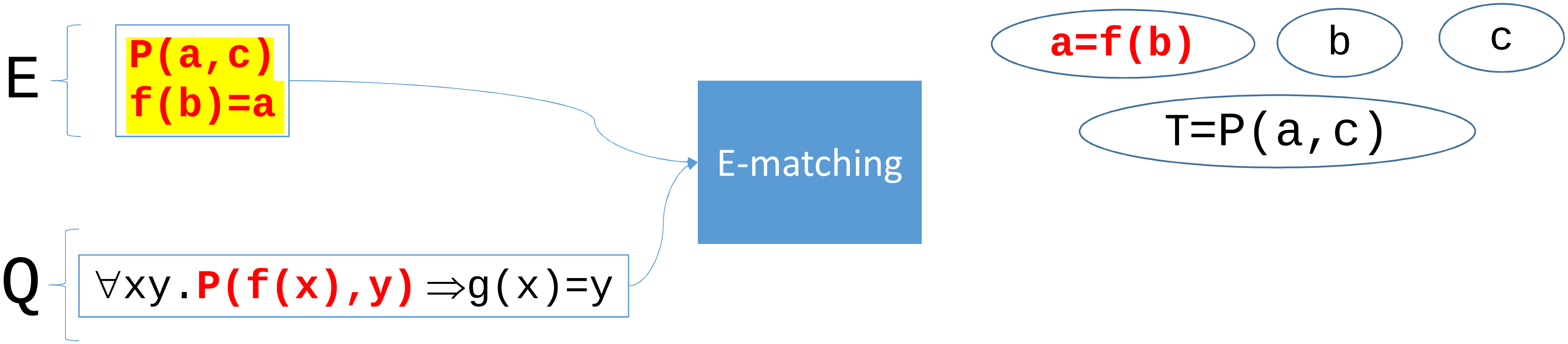
b

c

T=P(a,c)

Congruence closure of E

E-matching: Functions, Equality



E

P(a,c)

f(b)=a

Q

∀xy.P(f(x),y) ⇒g(x)=y

E-matching

a=f(b)

b

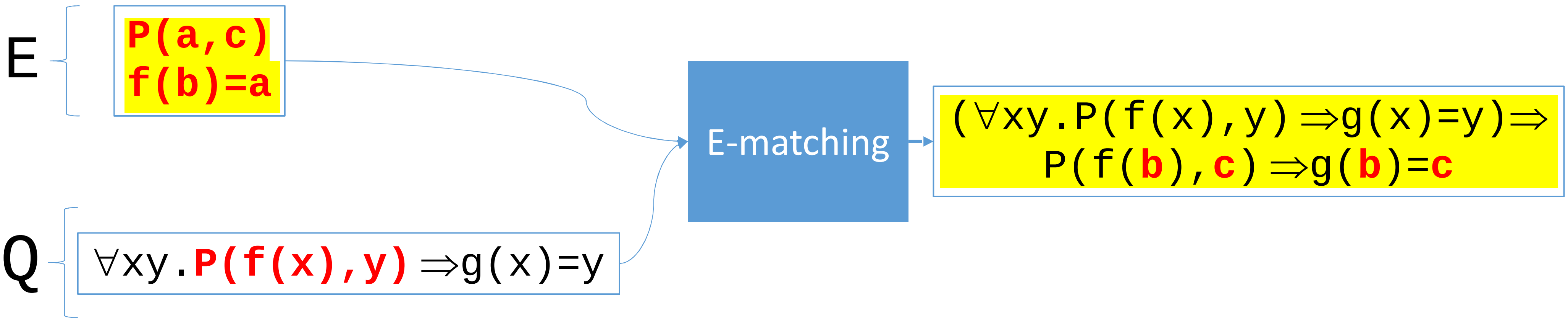
c

T=P(a,c)

P(f(b),c)

...E implies P(a,c)⇔P(f(b),c)

E-matching: Functions, Equality



P(a,c)
f(b)=a

$\forall xy.P(f(x),y) \Rightarrow g(x)=y$

$(\forall xy.P(f(x),y) \Rightarrow g(x)=y) \Rightarrow$
$P(f(b),c) \Rightarrow g(b)=c$

P(f(b),c)

Exercise : E-Matching with UF

$$\forall xyz. P(f(g(x),f(y,z)))$$

$$E \left\{ \quad \neg P(c) \land a=f(b,a) \land b=g(c) \land c=f(b,c) \right.$$

- Find terms t_x, t_y, t_z such that

E implies $P(f(g(x),f(y,z))) \{ x \rightarrow t_x, y \rightarrow t_y, z \rightarrow t_z \} = P(c)$

Exercise : E-Matching with UF

$$\forall xyz.P(f(g(x),f(y,z)))$$

$$E \left\{ \quad \neg P(c) \wedge a=f(b,a) \wedge b=g(c) \wedge c=f(b,c) \right.$$

E implies $P(f(g(x),f(y,z)))\ \{ x \rightarrow c,\ y \rightarrow b,\ z \rightarrow c \} = P(\ c\ )$

Exercise : E-Matching with UF

$$\forall xyz.P(f(g(x),f(y,z)))$$

$$E \left\{ \quad \neg P(c) \wedge a=f(b,a) \wedge b=g(c) \wedge c=f(b,c) \right.$$

E implies $P(f(g(c),f(b,c))) = P(\ c\ )$

Exercise : E-Matching with UF

$$\forall xyz.P(f(g(x),f(y,z)))$$

$$E \left\{ \quad \neg P(c) \wedge a=f(b,a) \wedge b=g(c) \wedge c=f(b,c) \right.$$

E implies $P(f(b,f(b,c))) = P(\ c\ )$

Exercise : E-Matching with UF

$$\forall xyz.P(f(g(x),f(y,z)))$$

$$E \left\{ \quad \neg P(c) \wedge a=f(b,a) \wedge b=g(c) \wedge c=f(b,c) \right.$$

E implies $P(f(b,c)) = P(\ c\ )$

Exercise : E-Matching with UF

$$\forall xyz.P(f(g(x),f(y,z)))$$

$$E \left\{ \quad \neg P(c)\wedge a=f(b,a)\wedge b=g(c)\wedge c=f(b,c) \right.$$

E implies P(c) = P(c)

Challenge : Pattern Selection

- In practice, pattern selection can is done either by:
  - The user, via annotations, e.g. `(! … :pattern ((P x)))`
  - The SMT solver itself (which usually selects all patterns)
- Recurrent questions:
  - Which terms be we permit as patterns? Typically, applications of UF:
    - Use $f(x,y)$ but not $x+y$ for $\forall xy.f(x,y)=x+y$
  - What if multiple patterns exist? Typically use all available patterns:
    - Use both $P(x)$ and $R(x)$ for $\forall x.P(x) \vee R(x)$
  - What if no appropriate term contains all variables? May use “multi-patterns”:
    - $\{R(x,y),R(y,z)\}$ for $\forall xyz.(R(x,y) \wedge R(y,z)) \Rightarrow R(x,z)$
- Pattern selections may impact performance significantly **[Leino et al 16]**

E-matching

- Most widely used technique for unsatisfiable ∀ problems in SMT
  - Variants implemented in:
    - Z3 [deMoura et al 07], CVC3 [Ge et al 07], CVC4, Princess [Ruemmer 12], VeriT, Alt-Ergo
  - Used in:
    - Software verification
      - Boogie, Dafny [Leino 2010], Leon, SPARK, Why3 [Bobot et al 2011], GRASShopper [Wies et al 2013]
    - Automated Theorem Proving
      - Sledgehammer [Blanchette et al 2011]

Exercise

$$\forall x.(P(x) \lor \neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5)$$

- What instantiations do I need to show this is unsatisfiable?

- Hints:
  - Literals contain entire scope of quantified formulas
    - E.g. "$\forall x.(P(x) \lor \neg R(x))$" is a literal (assigned true/false)
  - May require multiple iterations of E-matching

Exercise

$\forall x.(P(x) \vee \neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5)$

- DPLL(UFLIA) + E-Matching

Context

Exercise

$\forall x.(P(x) \vee \neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5)$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \vee \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false

Context

$\forall x.P(x) \vee \neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

Exercise

$\forall x.(P(x) \vee \neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5)$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \vee \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Run E-matching on $E = \{ R(3), P(3), \neg P(5) \}$,
    $Q = \{ \forall x.(P(x) \vee \neg R(x)), \forall x.R(x) \}$

Context

$\forall x.P(x) \vee \neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

Exercise

$\forall x.(P(x) \vee \neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5)$

Context
$\forall x.P(x) \vee \neg R(x)$
$\forall x.R(x)$
$R(3)$
$P(3)$
$\neg P(5)$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \vee \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Run E-matching on

matches

$E = \{ R(3), P(3), \neg P(5)\}$,

$Q = \{\forall x.(P(x) \vee \neg R(x)), \forall x.R(x)\}$

Exercise

$\forall x.(P(x) \lor \neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$
$(\neg \forall x.(P(x) \lor \neg R(x)) \lor P(3) \lor \neg R(3)) \land$
$(\neg \forall x.(P(x) \lor \neg R(x)) \lor P(5) \lor \neg R(5))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \lor \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Run E-matching on

matches

$E = \{ R(3), P(3), \neg P(5)\},$
$Q = \{\forall x.(P(x) \lor \neg R(x)), \forall x.R(x)\}$

$\Rightarrow$ Return $\forall x.(P(x) \lor \neg R(x)) \Rightarrow P(3) \lor \neg R(3)$, $\forall x.(P(x) \lor \neg R(x)) \Rightarrow P(5) \lor \neg R(5)$

Context

$\forall x.P(x) \lor \neg R(x)$
$\forall x.R(x)$
$R(3)$
$P(3)$
$\neg P(5)$

Exercise

$\forall x.(P(x) \vee \neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5) \wedge$
$(\neg \forall x.(P(x) \vee \neg R(x)) \vee P(3) \vee \neg R(3)) \wedge$
$(\neg \forall x.(P(x) \vee \neg R(x)) \vee P(5) \vee \neg R(5))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \vee \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Run E-matching on

matches

$E = \{ R(3), P(3), \neg P(5)\},$
$Q = \{\forall x.(P(x) \vee \neg R(x)), \forall x.R(x)\}$

Context
$\forall x.P(x) \vee \neg R(x)$
$\forall x.R(x)$
$R(3)$
$P(3)$
$\neg P(5)$

Exercise

$\forall x.(P(x)\lor\neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$
$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(3)\lor\neg R(3)) \land$
$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(5)\lor\neg R(5))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x)\lor\neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Run E-matching on

matches

$E = \{ R(3), P(3), \neg P(5)\},$

$Q = \{\forall x.(P(x)\lor\neg R(x)), \forall x.R(x)\}$

$\Rightarrow$ Return $\forall x.(P(x)\lor\neg R(x))\Rightarrow P(3)\lor R(3)$ (duplicate)

Context

$\forall x.P(x)\lor\neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

Exercise

$\forall x.(P(x) \vee \neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5) \wedge$
$(\neg \forall x.(P(x) \vee \neg R(x)) \vee P(3) \vee \neg R(3)) \wedge$
$(\neg \forall x.(P(x) \vee \neg R(x)) \vee P(5) \vee \neg R(5))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \vee \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Run E-matching on

    matches

    $E = \{ R(3), P(3), \neg P(5)\}$,
    $Q = \{\forall x.(P(x) \vee \neg R(x)), \forall x.R(x)\}$

Context

$\forall x.P(x) \vee \neg R(x)$
$\forall x.R(x)$
$R(3)$
$P(3)$
$\neg P(5)$

Exercise

$\forall x.(P(x)\lor\neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$
$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(3)\lor\neg R(3)) \land (\neg\forall x.R(x)\lor R(3)) \land$
$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(5)\lor\neg R(5))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x)\lor\neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Run E-matching on

matches

$E = \{ R(3), P(3), \neg P(5)\},$
$Q = \{\forall x.(P(x)\lor\neg R(x)), \forall x.R(x)\}$

$\Rightarrow$ Return $\forall x.R(x)\Rightarrow R(3)$

Context
$\forall x.P(x)\lor\neg R(x)$
$\forall x.R(x)$
$R(3)$
$P(3)$
$\neg P(5)$

Exercise

$$\forall x.(P(x) \lor \neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$$
$$(\neg \forall x.(P(x) \lor \neg R(x)) \lor P(3) \lor \neg R(3)) \land (\neg \forall x.R(x) \lor R(3)) \land$$
$$(\neg \forall x.(P(x) \lor \neg R(x)) \lor P(5) \lor \neg R(5))$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \lor \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false

Context

$\forall x.P(x) \lor \neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

Exercise

$\forall x.(P(x)\vee\neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5) \wedge$
$(\neg\forall x.(P(x)\vee\neg R(x))\vee P(3)\vee\neg R(3)) \wedge (\neg\forall x.R(x)\vee R(3)) \wedge$
$(\neg\forall x.(P(x)\vee\neg R(x))\vee P(5)\vee\neg R(5))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x)\vee\neg R(x))$ → true
  - Propagate : $\forall x.R(x)$ → true
  - Propagate : $R(3)$ → true
  - Propagate : $P(3)$ → true
  - Propagate : $P(5)$ → false
  - Propagate : $R(5)$ → false

Context
$\forall x.P(x)\vee\neg R(x)$
$\forall x.R(x)$
$R(3)$
$P(3)$
$\neg P(5)$
$\neg R(5)$

Exercise

$$\begin{gathered}\forall x.(P(x) \vee \neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5) \wedge \\ (\neg \forall x.(P(x) \vee \neg R(x)) \vee P(3) \vee \neg R(3)) \wedge (\neg \forall x.R(x) \vee R(3)) \wedge \\ (\neg \forall x.(P(x) \vee \neg R(x)) \vee P(5) \vee \neg R(5))\end{gathered}$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x) \vee \neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Propagate : $R(5) \rightarrow$ false
  - Run E-matching on $E = \{ R(3), P(3), \neg P(5), \neg R(5)\}$,
    $Q = \{\forall x.(P(x) \vee \neg R(x)), \forall x.R(x)\}$

Context

$\forall x.P(x) \vee \neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

$\neg R(5)$

Exercise

$\forall x.(P(x)\lor\neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$
$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(3)\lor\neg R(3)) \land (\neg\forall x.R(x)\lor R(3)) \land$
$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(5)\lor\neg R(5))$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x)\lor\neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Propagate : $R(5) \rightarrow$ false
  - Run E-matching on

matches

$E = \{ R(3), P(3), \neg P(5), \neg R(5)\},$
$Q = \{\forall x.(P(x)\lor\neg R(x)), \forall x.R(x)\}$

Context

$\forall x.P(x)\lor\neg R(x)$
$\forall x.R(x)$
$R(3)$
$P(3)$
$\neg P(5)$
$\neg R(5)$

Exercise

$$\forall x.(P(x)\lor\neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$$
$$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(3)\lor\neg R(3)) \land (\neg\forall x.R(x)\lor R(3)) \land$$
$$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(5)\lor\neg R(5)) \land (\neg\forall x.R(x)\lor R(5))$$

- DPLL(UFLIA) + E-Matching
  - Propagate : $\forall x.(P(x)\lor\neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Propagate : $R(5) \rightarrow$ false
  - Run E-matching on $E = \{ R(3), P(3), \neg P(5), \neg R(5)\}$,

    matches

    $Q = \{\forall x.(P(x)\lor\neg R(x)), \forall x.R(x)\}$

$\Rightarrow$ Return $\forall x.R(x)\Rightarrow R(5)$

Context

$\forall x.P(x)\lor\neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

$\neg R(5)$

Exercise

$$\forall x.(P(x)\lor\neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$$
$$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(3)\lor\neg R(3)) \land (\neg\forall x.R(x)\lor R(3)) \land$$
$$(\neg\forall x.(P(x)\lor\neg R(x))\lor P(5)\lor\neg R(5)) \land \underbrace{(\neg\forall x.R(x)\lor R(5))}$$

- DPLL(UFLIA) + E-Matching ⇒ Conflicting clause!
  - Propagate : $\forall x.(P(x)\lor\neg R(x)) \rightarrow$ true
  - Propagate : $\forall x.R(x) \rightarrow$ true
  - Propagate : $R(3) \rightarrow$ true
  - Propagate : $P(3) \rightarrow$ true
  - Propagate : $P(5) \rightarrow$ false
  - Propagate : $R(5) \rightarrow$ false

…no decision to backtrack

⇒ Input is UFLIA-unsat

Context

$\forall x.P(x)\lor\neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

$\neg R(5)$

Exercise

$\forall x.(P(x) \lor \neg R(x)) \land \forall x.R(x) \land R(3) \land P(3) \land \neg P(5) \land$

~~$(\neg \forall x.(P(x) \lor \neg R(x)) \lor P(3) \lor \neg R(3)) \land (\neg \forall x.R(x) \lor R(3)) \land$~~

$\underbrace{(\neg \forall x.(P(x) \lor \neg R(x)) \lor P(5) \lor \neg R(5)) \land (\neg \forall x.R(x) \lor R(5))}$

⇒ Only the latter two instantiation lemmas are necessary

⇒ Input is UFLIA-unsat

Context

$\forall x.P(x) \lor \neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

$\neg R(5)$

Exercise

$\forall x.(P(x)\vee\neg R(x)) \wedge \forall x.R(x) \wedge R(3) \wedge P(3) \wedge \neg P(5) \wedge$

~~$(\neg\forall x.(P(x)\vee\neg R(x))\vee P(3)\vee\neg R(3)) \wedge (\neg\forall x.R(x)\vee R(3)) \wedge$~~

$(\neg\forall x.(P(x)\vee\neg R(x))\vee P(5)\vee\neg R(5)) \wedge (\neg\forall x.R(x)\vee R(5))$

- Takeaways:
  - Instantiation lemmas introduce new literals, e.g. $\neg R(5)$
    - Subsequently used in later invocations of E-matching
  - Not all instantiation lemmas are helpful

$\Rightarrow$ Input is UFLIA-unsat

Context

$\forall x.P(x)\vee\neg R(x)$

$\forall x.R(x)$

$R(3)$

$P(3)$

$\neg P(5)$

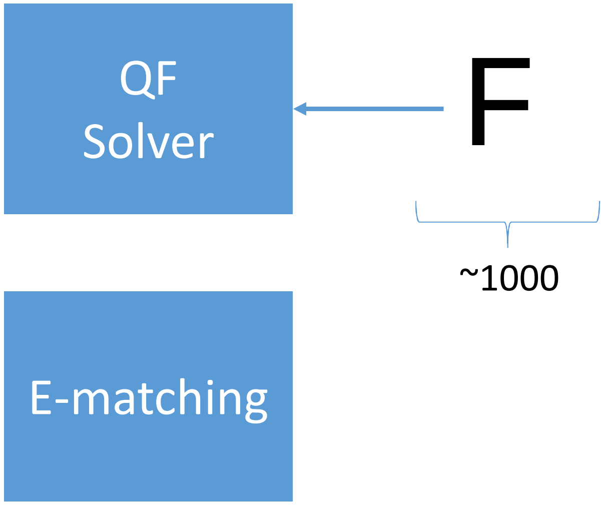
$\neg R(5)$

Encoding in *.smt2

```
(set-logic UFLIA)
(declare-fun P (Int) Bool)
(declare-fun R (Int) Bool)
(assert (forall ((x Int)) (or (P x) (not (R x)))))
(assert (forall ((x Int)) (R x)))
(assert (R 3))
(assert (P 3))
(assert (not (P 5)))
(check-sat)
```

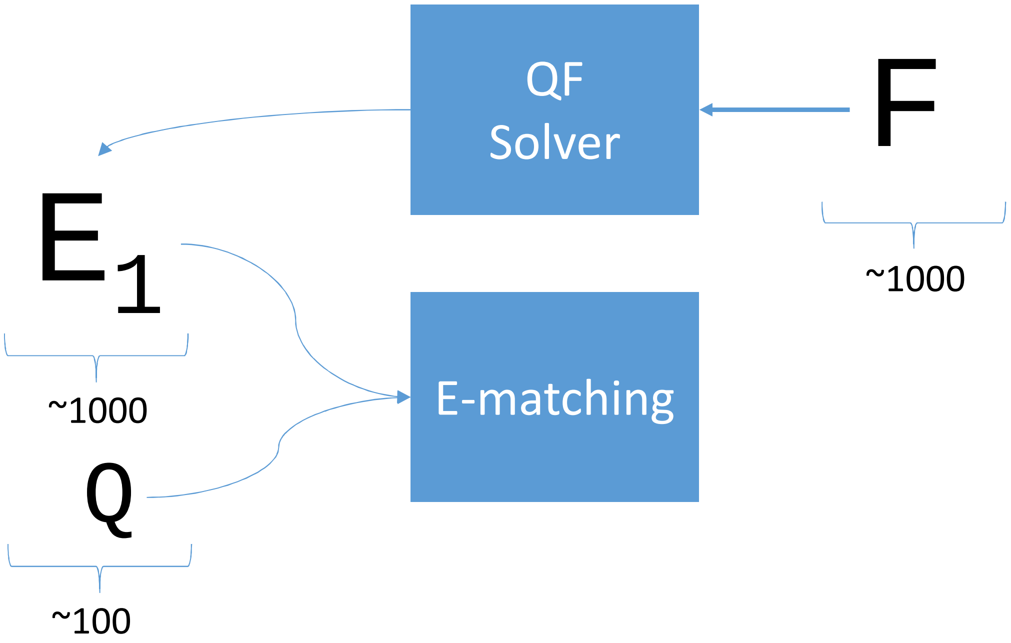
EXAMPLE 2…

Challenge #1 : Too Many Instances



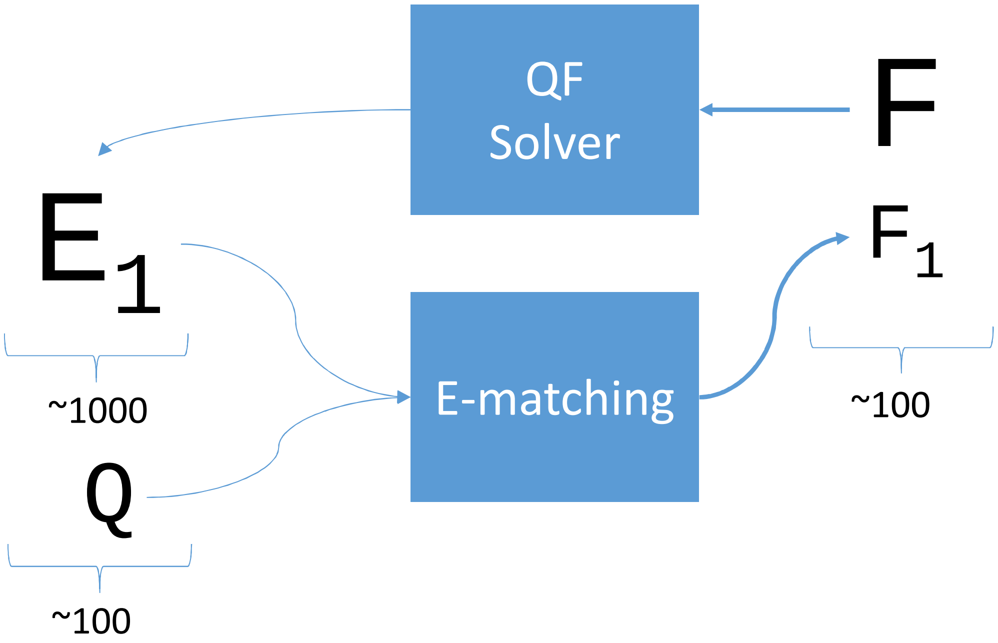
- Typical problems in applications:
  - F contains 1000s of clauses

Challenge #1 : Too Many Instances



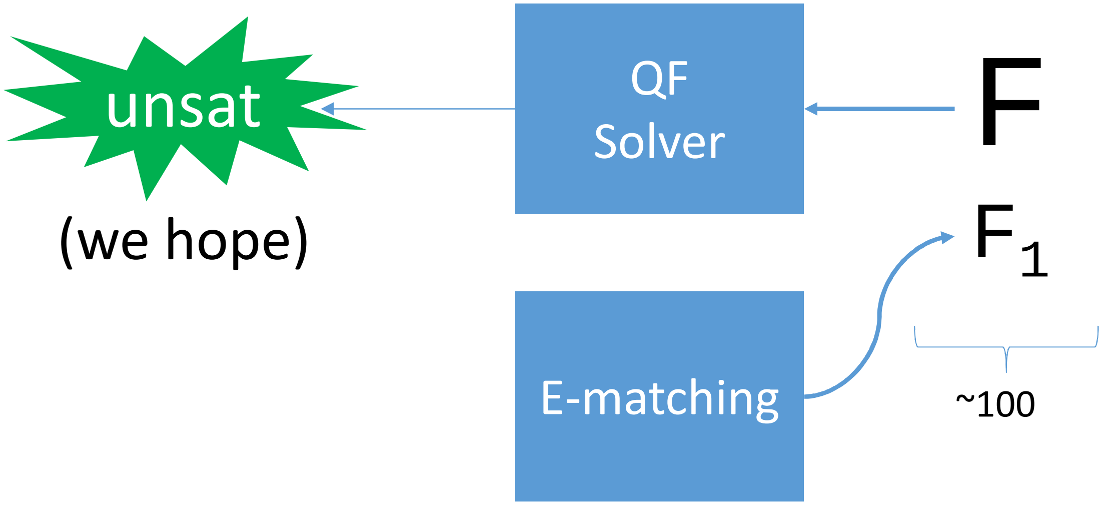
- Typical problems in applications:
  - F contains 1000s of clauses
  - Contexts contain 1000s of terms in E, 100s of ∀ in Q

Challenge #1 : Too Many Instances



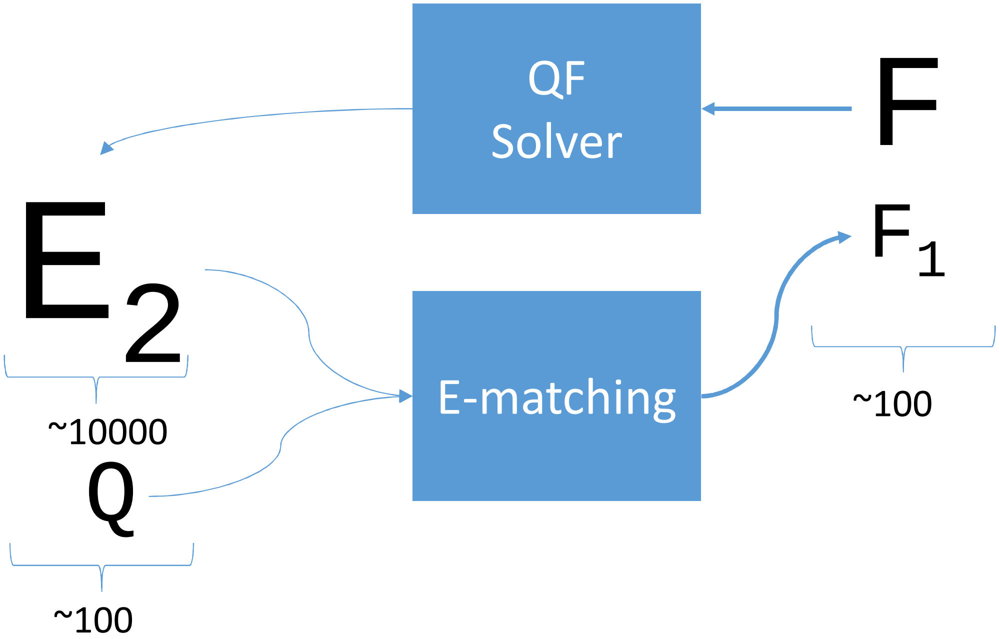
- Typical problems in applications:
  - F contains 1000s of clauses
  - Contexts contain 1000s of terms in E, 100s of ∀ in Q

Challenge #1 : Too Many Instances



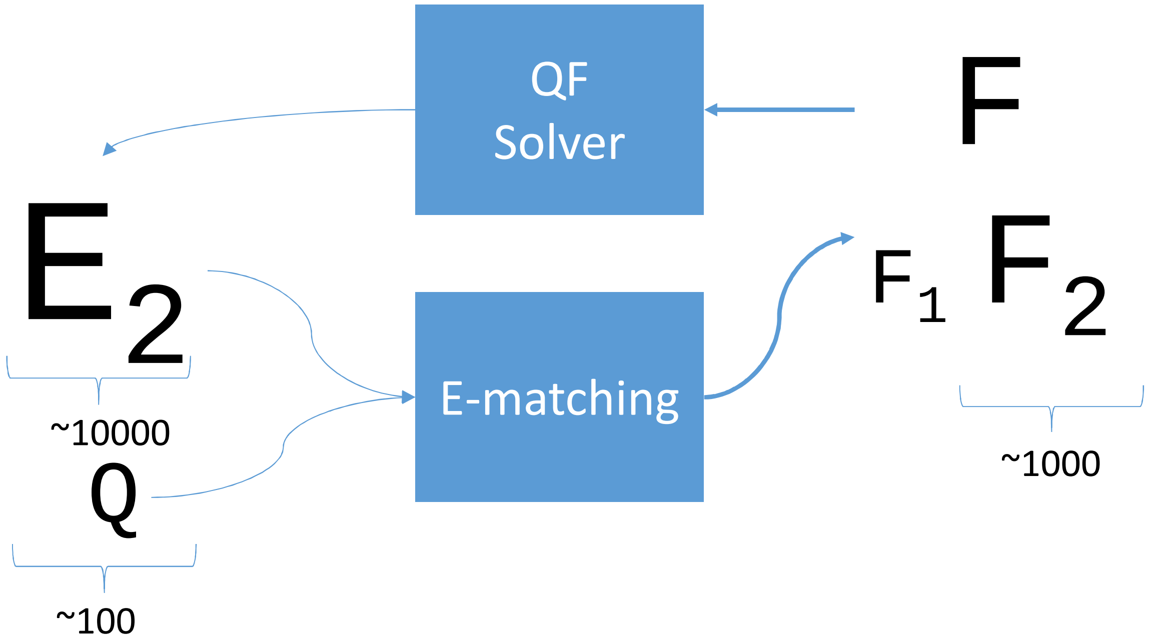
- Typical problems in applications:
  - F contains 1000s of clauses
  - Contexts contain 1000s of terms in E, 100s of ∀ in Q

Challenge #1 : Too Many Instances



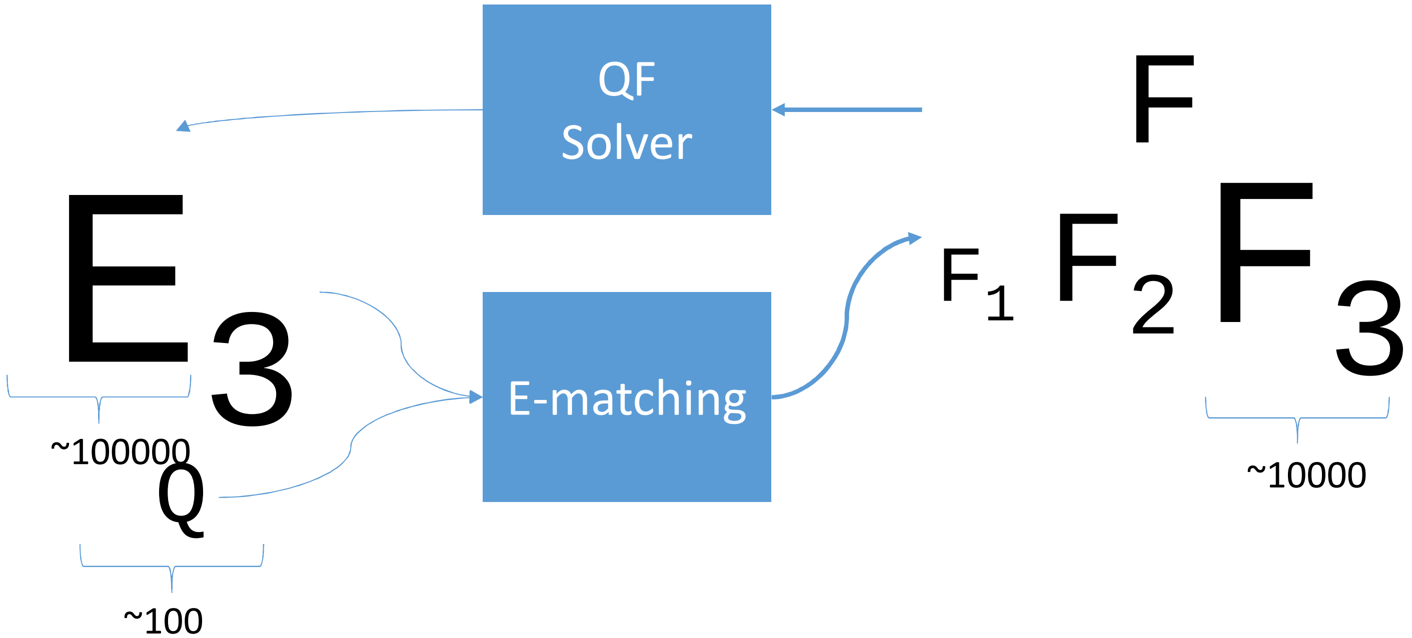
- Typical problems in applications:
  - F contains 1000s of clauses
  - Contexts contain 1000s of terms in E, 100s of ∀ in Q
  - Leads to 100s

Challenge #1 : Too Many Instances



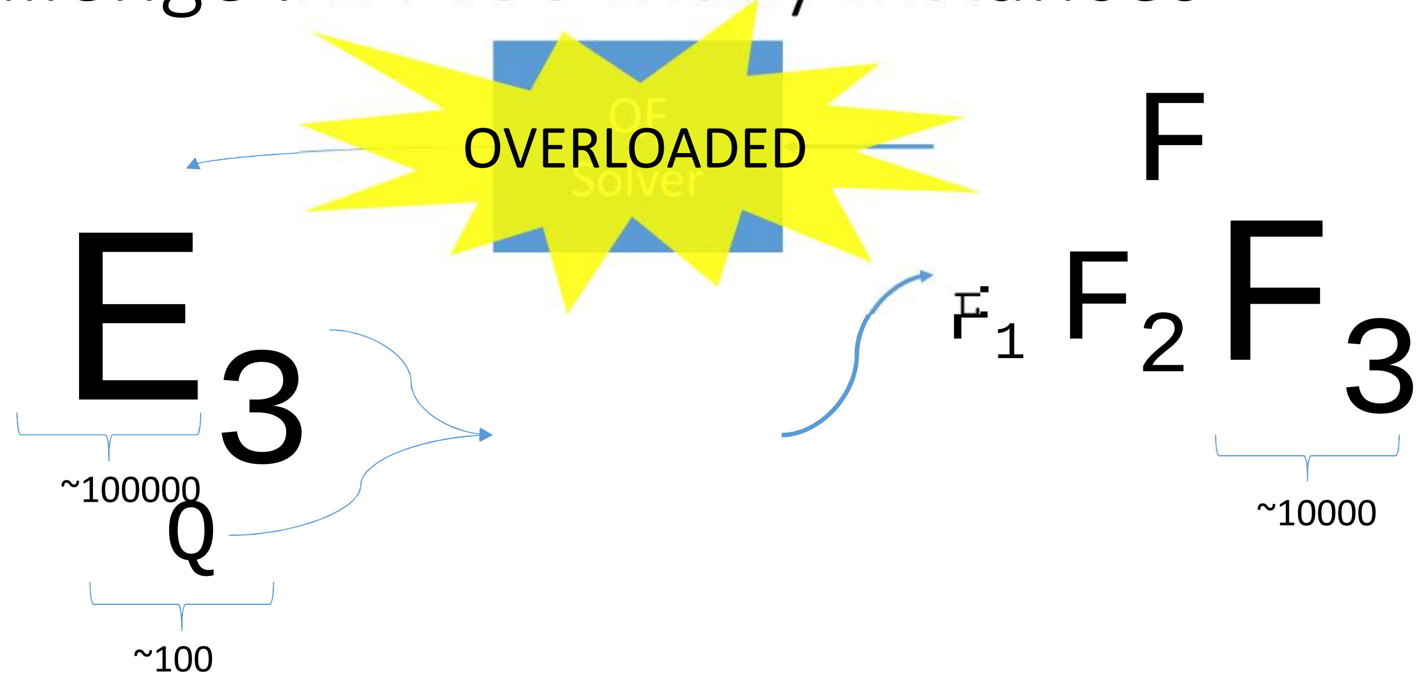
- Typical problems in applications:
  - F contains 1000s of clauses
  - Contexts contain 1000s of terms in E, 100s of ∀ in Q
  - Leads to 100s, 1000s

Challenge #1 : Too Many Instances



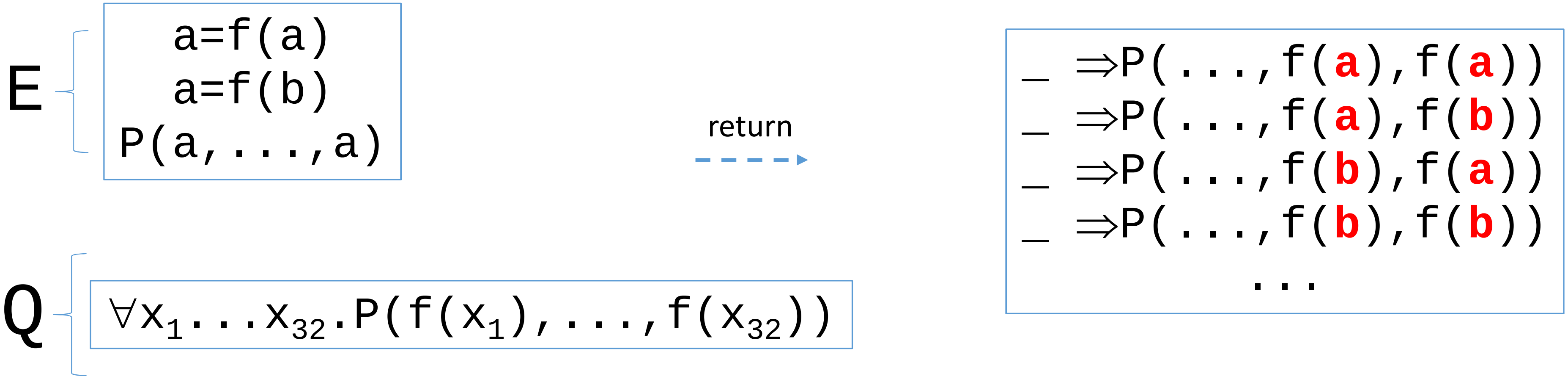
- Typical problems in applications:
  - F contains 1000s of clauses
  - Contexts contain 1000s of terms in E, 100s of ∀ in Q
  - Leads to 100s, 1000s, 10000s of instances

Challenge #1 : Too Many Instances



⇒ QF solver is overloaded …solver times out

Challenge : Too Many Instances



E

$a=f(a)$

$a=f(b)$

$P(a,...,a)$

Q

$\forall x_1 ... x_{32}.P(f(x_1),...,f(x_{32}))$

return

_ $\Rightarrow P(...,f(a),f(a))$

_ $\Rightarrow P(...,f(a),f(b))$

_ $\Rightarrow P(...,f(b),f(a))$

_ $\Rightarrow P(...,f(b),f(b))$

...

⇒ In fact, E-matching may be *exponential*, above produces 2^{32} instances

- Thus, we limit # matches per ground term/pattern pair

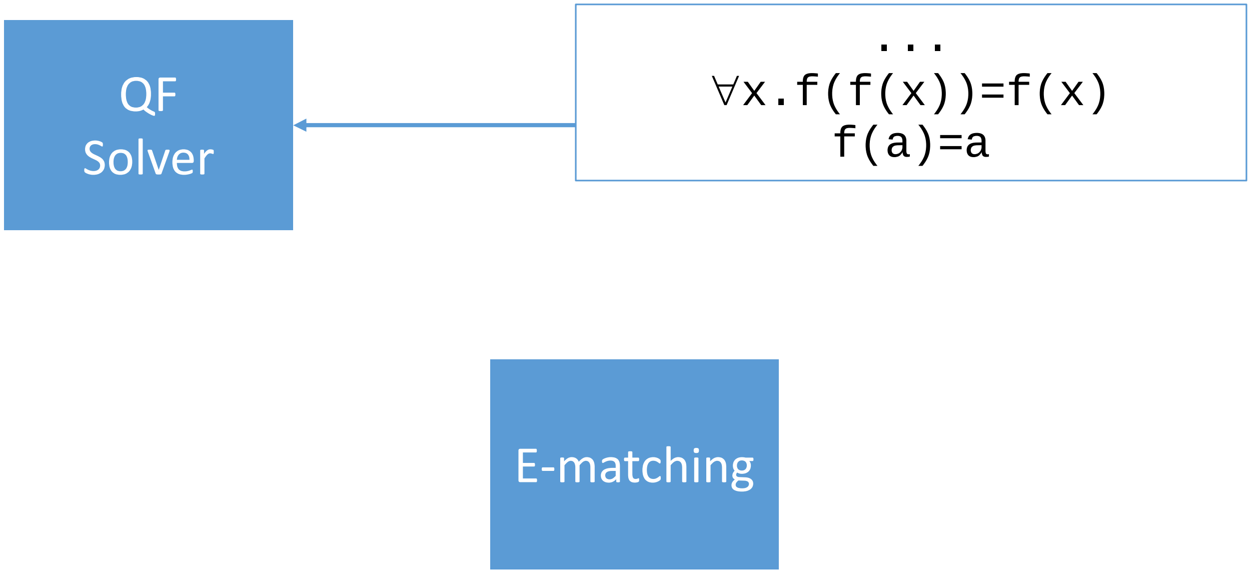
Challenge : Too Many Instances

# Instances	cvc3		cvc4		z3	
	#	%	#	%	#	%
1-10	1464	13.49%	1007	8.87%	1321	11.43%
10-100	1755	16.17%	1853	16.31%	2554	22.11%
100-1000	**3816**	35.16%	**3680**	32.40%	**4553**	39.41%
1000-10k	1893	17.44%	2468	21.73%	1779	15.40%
10k-100k	1162	10.71%	1414	12.45%	823	7.12%
100k-1M	560	5.16%	607	5.34%	376	3.25%
1M-10M	193	1.78%	330	2.91%	139	1.20%
>10M	10	0.09%	0	0.00%	8	0.07%

(for 8 of benchmarks z3 solves,
its E-matching procedure adds
more than 10M instances)

- Evaluation on 33032 SMTLIB, TPTP, Isabelle benchmarks
  - E-matching often requires **many instances**
    (Above, 16.6% required >10k, max 19.5M by z3 on a software verification benchmark from TPTP)

Challenge: Non-termination



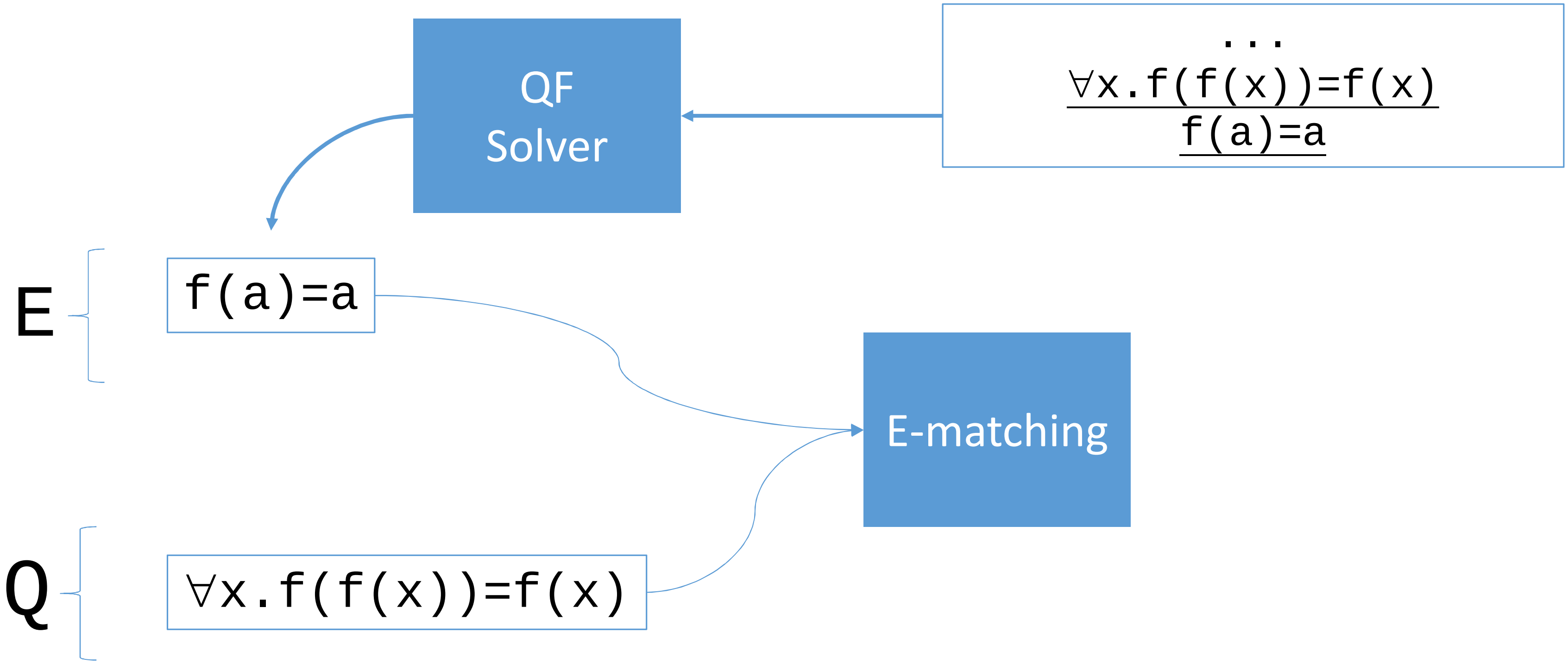
QF Solver

```
...
∀x.f(f(x))=f(x)
f(a)=a
```

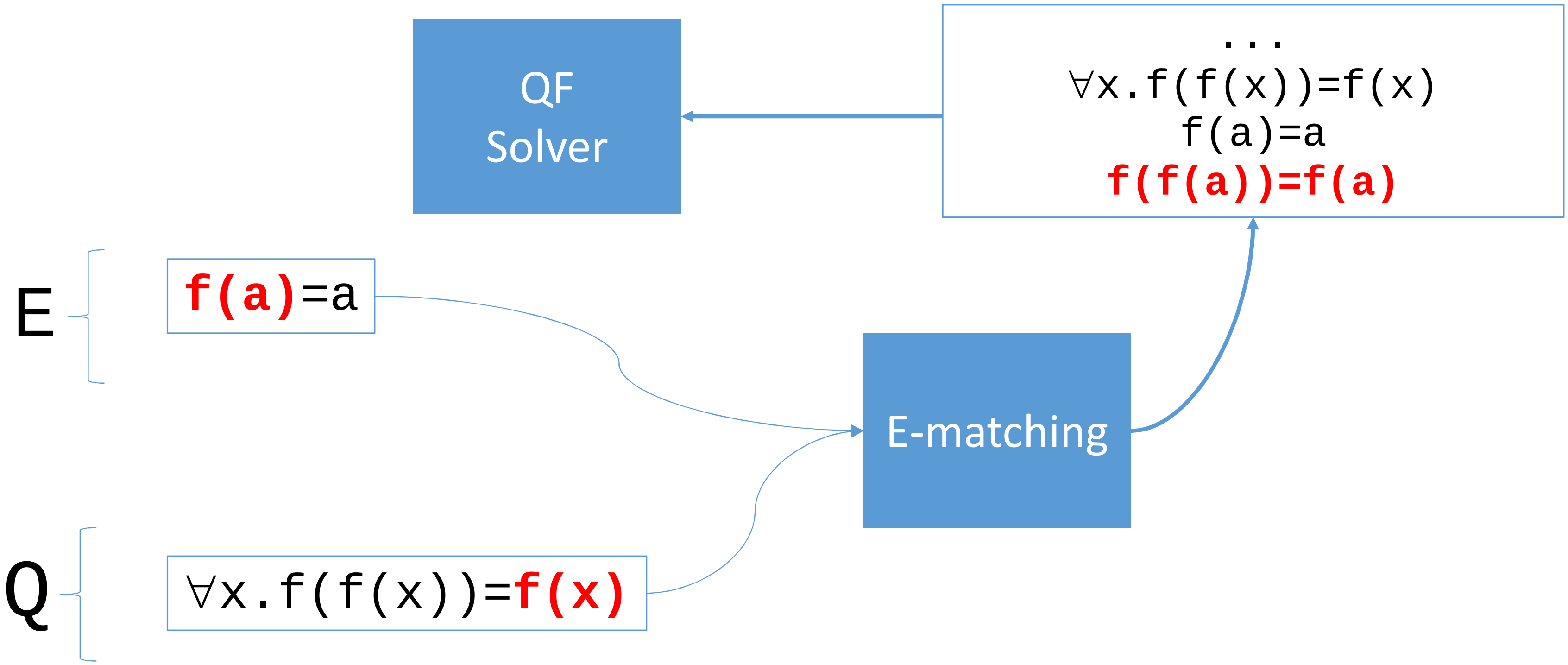
E-matching

⇒ E-matching may be non-terminating

Challenge : Non-termination



Challenge : Non-termination



QF Solver

...

∀x.f(f(x))=f(x)

f(a)=a

f(f(a))=f(a)

E

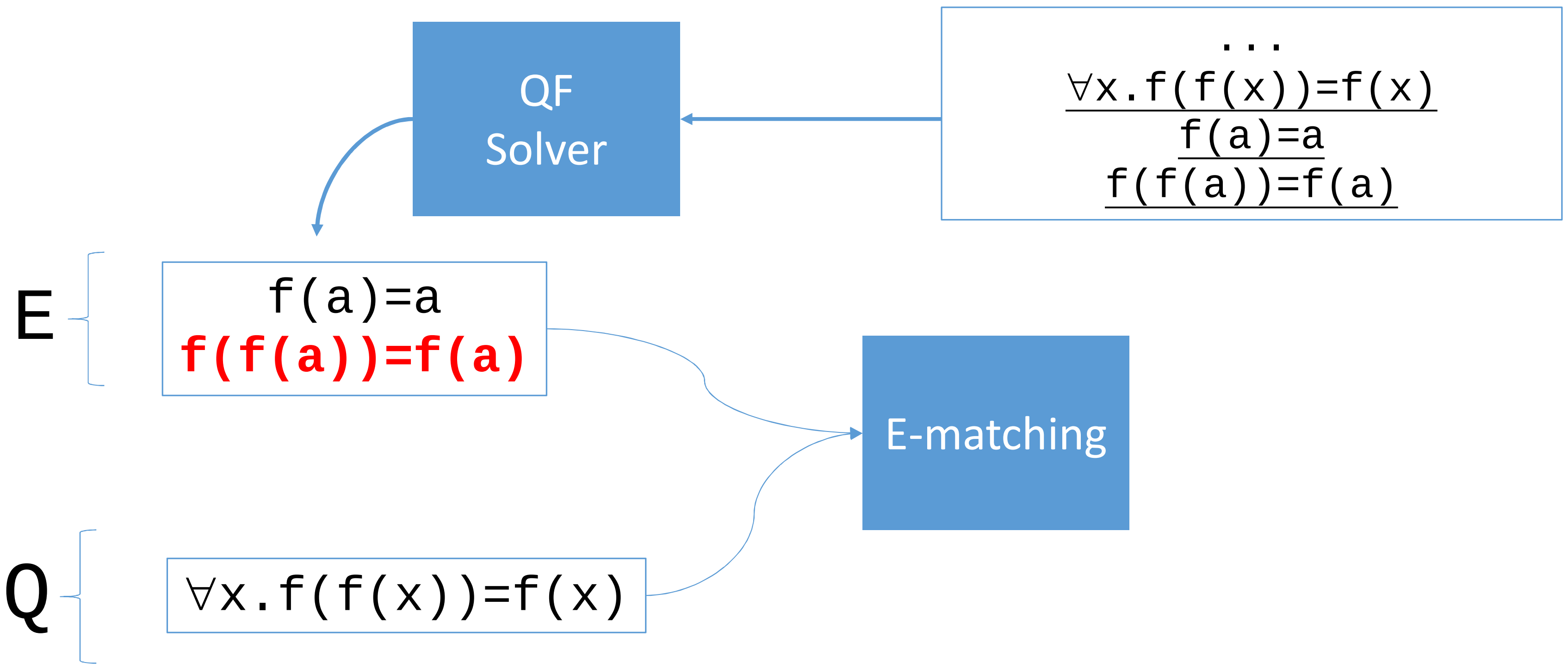
f(a)=a

E-matching

Q

∀x.f(f(x))=**f(x)**

Challenge : Non-termination



QF Solver

...

<u>∀x.f(f(x))=f(x)</u>

<u>f(a)=a</u>

<u>f(f(a))=f(a)</u>

E

f(a)=a

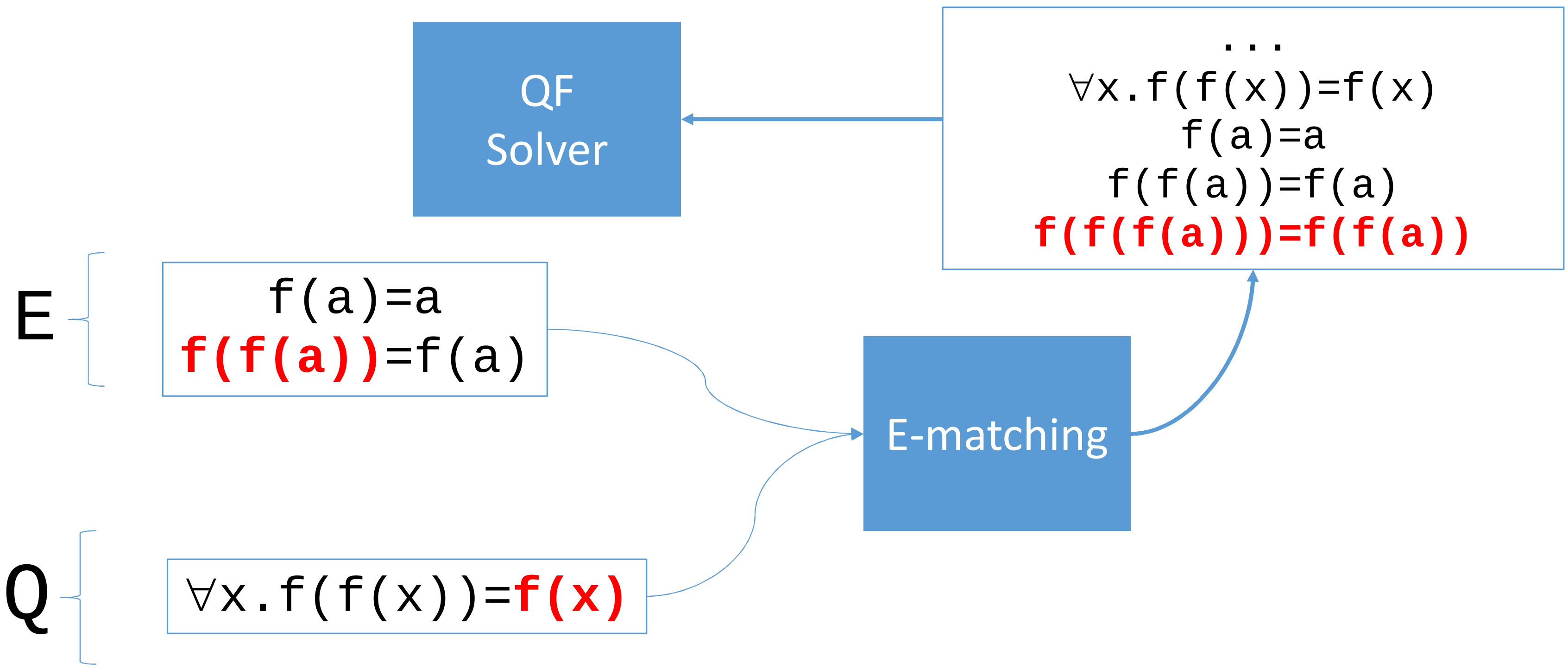
f(f(a))=f(a)

E-matching

Q

∀x.f(f(x))=f(x)

Challenge : Non-termination



QF

Solver

...

∀x.f(f(x))=f(x)

f(a)=a

f(f(a))=f(a)

f(f(f(a)))=f(f(a))

E

f(a)=a

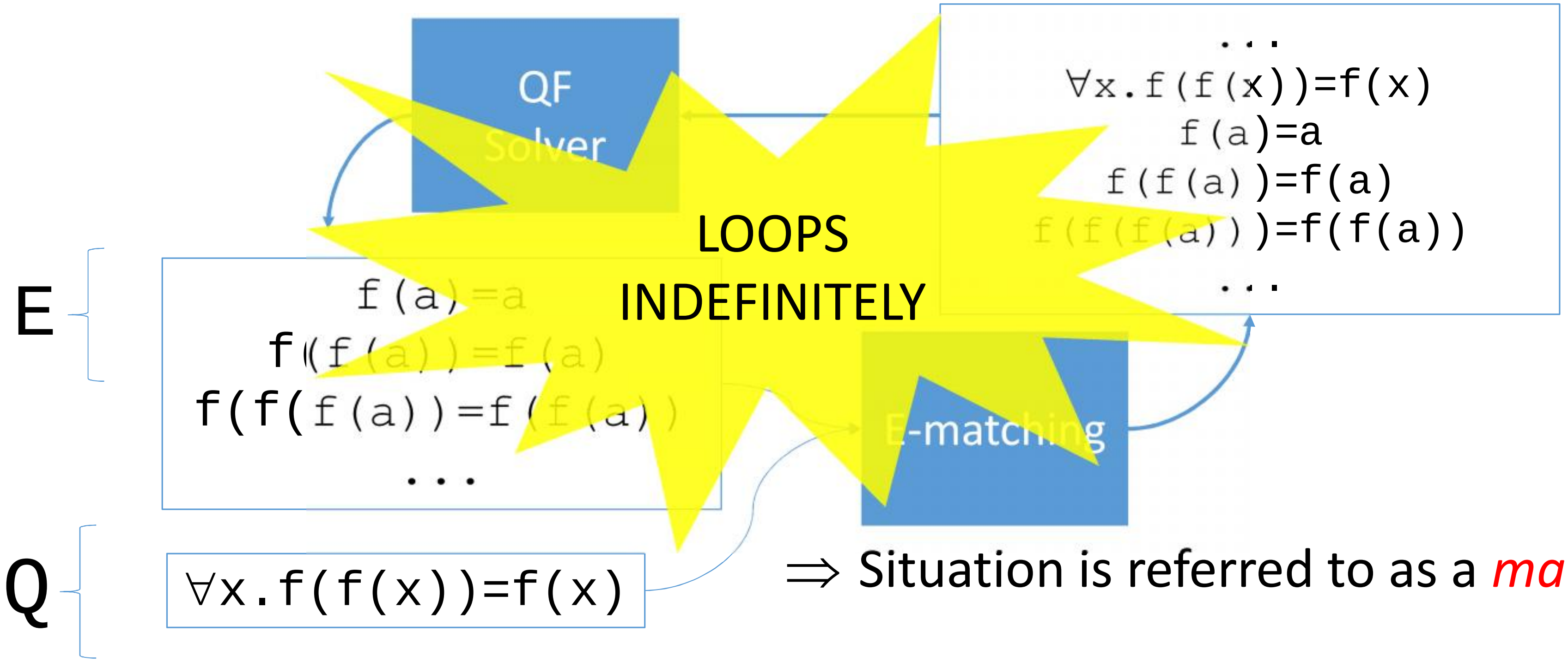
f(f(a))=f(a)

E-matching

Q

∀x.f(f(x))=**f(x)**

Challenge : Non-termination



QF Solver

...

∀x.f(f(x))=f(x)

f(a)=a

f(f(a))=f(a)

f(f(f(a)))=f(f(a))

...

LOOPS INDEFINITELY

E

f(a)=a

f(f(a))=f(a)

f(f(f(a))=f(f(a))

...

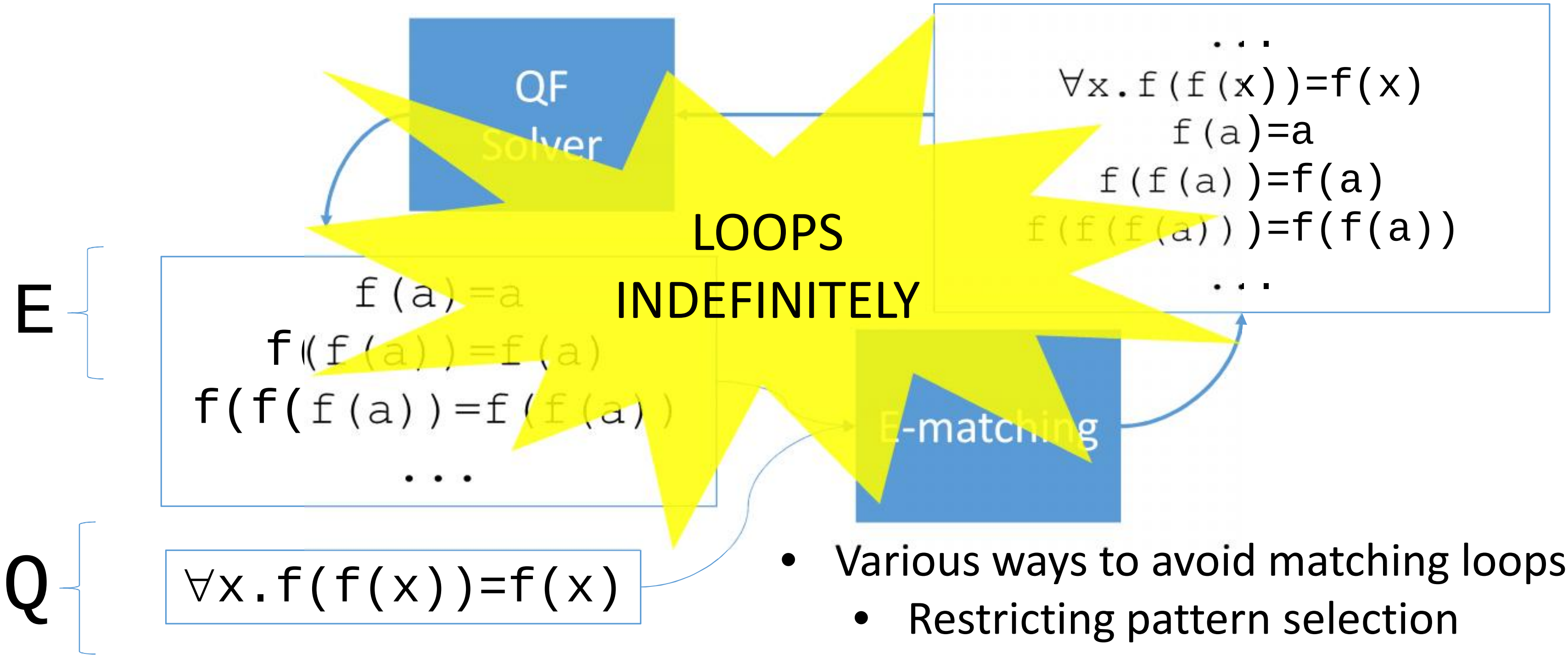
E-matching

Q

∀x.f(f(x))=f(x)

⇒ Situation is referred to as a *matching loop*

Challenge : Non-termination



- Various ways to avoid matching loops, e.g. by:
  - Restricting pattern selection
  - Fair instantiations strategies (track “levels”)

Exercise

$\forall x.(P(x) \vee R(x)) \wedge \neg P(a) \wedge \neg R(5) \wedge (a=5 \vee \neg a=5)$

- Is this input satisfiable or unsatisfiable?

Exercise

$$\forall x.(P(x) \vee R(x)) \wedge \neg P(a) \wedge \neg R(5) \wedge (a=5 \vee \neg a=5)$$

- Propagate $\forall x.P(x) \vee R(x) \rightarrow$ true
- Propagate $P(a) \rightarrow$ false
- Propagate $R(5) \rightarrow$ false

Context

$\forall x.P(x) \vee R(x)$

$\neg P(a)$

$\neg R(5)$

Exercise

$$\forall x.(P(x) \lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5 \lor \neg a=5)$$

- Propagate $\forall x.P(x) \lor R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Decide $a=5$ → true

Context

$\forall x.P(x) \lor R(x)$

$\neg P(a)$

$\neg R(5)$

$a=5^{d}$

Exercise

$$\forall x.(P(x) \lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5 \lor \neg a=5) \land$$
$$(\neg \forall x.(P(x) \lor R(x)) \lor P(a) \lor R(a)) \land$$
$$(\neg \forall x.(P(x) \lor R(x)) \lor P(5) \lor R(5))$$

- Propagate $\forall x.P(x) \lor R(x) \rightarrow$ true
- Propagate $P(a) \rightarrow$ false
- Propagate $R(5) \rightarrow$ false
- Decide $a=5 \rightarrow$ true

$\Rightarrow$ Instantiate $\{ x \rightarrow a \}, \{ x \rightarrow 5 \}$

Context

$\forall x.P(x) \lor R(x)$

$\neg P(a)$

$\neg R(5)$

$a=5^{d}$

Exercise

$$\forall x.(P(x)\lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5\lor\neg a=5) \land$$
$$(\neg\forall x.(P(x)\lor R(x))\lor P(a)\lor R(a)) \land$$
$$(\neg\forall x.(P(x)\lor R(x))\lor P(5)\lor R(5))$$

- Propagate $\forall x.P(x)\lor R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Decide $a=5$ → true
- Propagate $R(a)$ → true
- Propagate $P(5)$ → true

Context

$\forall x.P(x)\lor R(x)$

$\neg P(a)$

$\neg R(5)$

$a=5^{d}$

$R(a)$

$P(5)$

Exercise

$$\forall x.(P(x)\lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5\lor\neg a=5) \land$$
$$(\neg\forall x.(P(x)\lor R(x))\lor P(a)\lor R(a)) \land$$
$$(\neg\forall x.(P(x)\lor R(x))\lor P(5)\lor R(5))$$

- Propagate $\forall x.P(x)\lor R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Decide $a=5$ → true
- Propagate $R(a)$ → true
- Propagate $P(5)$ → true
- Run UF solver on $\{\neg P(a),\neg R(5),a=5,R(a),P(5)\}$

Context

$\forall x.P(x)\lor R(x)$

$\neg P(a)$

$\neg R(5)$

$a=5^{d}$

$R(a)$

$P(5)$

Exercise

$$\forall x.(P(x) \vee R(x)) \wedge \neg P(a) \wedge \neg R(5) \wedge (a=5 \vee \neg a=5) \wedge$$
$$(\neg \forall x.(P(x) \vee R(x)) \vee P(a) \vee R(a)) \wedge$$
$$(\neg \forall x.(P(x) \vee R(x)) \vee P(5) \vee R(5)) \wedge (\neg a=5 \vee P(a) \vee \neg P(5))$$

- Propagate $\forall x.P(x) \vee R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Decide $a=5$ → true
- Propagate $R(a)$ → true
- Propagate $P(5)$ → true
- Run UF solver on $\{ \neg P(a), \neg R(5), a=5, R(a), P(5)\}$…conflict

Context

$\forall x.P(x) \vee R(x)$

$\neg P(a)$

$\neg R(5)$

$a=5^{d}$

$R(a)$

$P(5)$

Exercise

$\forall x.(P(x) \vee R(x)) \wedge \neg P(a) \wedge \neg R(5) \wedge (a=5 \vee \neg a=5) \wedge$
$(\neg \forall x.(P(x) \vee R(x)) \vee P(a) \vee R(a)) \wedge$
$(\neg \forall x.(P(x) \vee R(x)) \vee P(5) \vee R(5)) \wedge (\neg a=5 \vee P(a) \vee \neg P(5))$

- Propagate $\forall x.P(x) \vee R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Backtrack $a=5$ → false

Context

$\forall x.P(x) \vee R(x)$

$\neg P(a)$

$\neg R(5)$

$\neg a=5$

Exercise

$$\forall x.(P(x)\lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5\lor\neg a=5) \land$$
$$(\neg\forall x.(P(x)\lor R(x))\lor P(a)\lor R(a)) \land$$
$$(\neg\forall x.(P(x)\lor R(x))\lor P(5)\lor R(5)) \land (\neg a=5\lor P(a)\lor\neg P(5))$$

- Propagate $\forall x.P(x)\lor R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Backtrack $a=5$ → false
- Propagate $R(a)$ → true
- Propagate $P(5)$ → true

Context

$\forall x.P(x)\lor R(x)$

$\neg P(a)$

$\neg R(5)$

$\neg a=5$

$R(a)$

$P(5)$

Exercise

$\forall x.(P(x)\lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5\lor\neg a=5) \land$
$(\neg\forall x.(P(x)\lor R(x))\lor P(a)\lor R(a)) \land$
$(\neg\forall x.(P(x)\lor R(x))\lor P(5)\lor R(5)) \land (\neg a=5\lor P(a)\lor\neg P(5))$

- Propagate $\forall x.P(x)\lor R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Backtrack $a=5$ → false
- Propagate $R(a)$ → true
- Propagate $P(5)$ → true
- Have $\forall x.(P(x)\lor R(x))$ …what else to instantiate for x?

Context

$\forall x.P(x)\lor R(x)$
$\neg P(a)$
$\neg R(5)$
$\neg a=5$
$R(a)$
$P(5)$

Exercise

$$\forall x.(P(x)\lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5 \lor \neg a=5) \land$$
$$(\neg\forall x.(P(x)\lor R(x)) \lor P(a) \lor R(a)) \land$$
$$(\neg\forall x.(P(x)\lor R(x)) \lor P(5) \lor R(5)) \land (\neg a=5 \lor P(a) \lor \neg P(5))$$

- Propagate $\forall x.P(x)\lor R(x)$ → true
- Propagate $P(a)$ → false
- Propagate $R(5)$ → false
- Backtrack $a=5$ → false
- Propagate $R(a)$ → true
- Propagate $P(5)$ → true
- Have $\forall x.(P(x)\lor R(x))$ …what else to instantiate for x?

⇒ This problem is "SAT", but E-matching cannot tell you that it is!

Context

$\forall x.P(x)\lor R(x)$

$\neg P(a)$

$\neg R(5)$

$\neg a=5$

$R(a)$

$P(5)$

Exercise

$$\forall x.(P(x) \lor R(x)) \land \neg P(a) \land \neg R(5) \land (a=5 \lor \neg a=5) \land$$
$$(\neg \forall x.(P(x) \lor R(x)) \lor P(a) \lor R(a)) \land$$
$$(\neg \forall x.(P(x) \lor R(x)) \lor P(5) \lor R(5)) \land (\neg a=5 \lor P(a) \lor \neg P(5))$$

- Takeaways:
  - E-matching cannot tell you a problem is “SAT”
    - E.g. it is an *incomplete* procedure

⇒ This problem is “SAT”, but E-matching cannot tell you that it is!

Context

$\forall x.P(x) \lor R(x)$

$\neg P(a)$

$\neg R(5)$

$\neg a=5$

$R(a)$

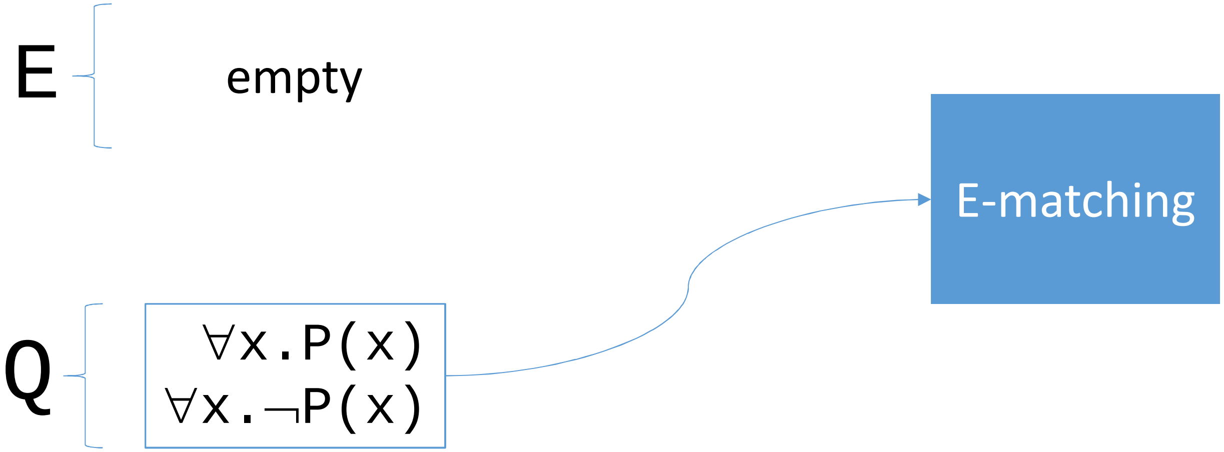
$P(5)$

Encoding in *.smt2

```
(set-logic UFLIA)
(declare-fun P (Int) Bool)
(declare-fun R (Int) Bool)
(declare-fun a () Int)
(assert (forall ((x Int)) (or (P x) (R x))))
(assert (not (P a)))
(assert (not (R 5)))
(assert (or (= a 5) (not (= a 5))))
(check-sat)
```

EXAMPLE 3…

Challenge : Incompleteness



E

empty

Q

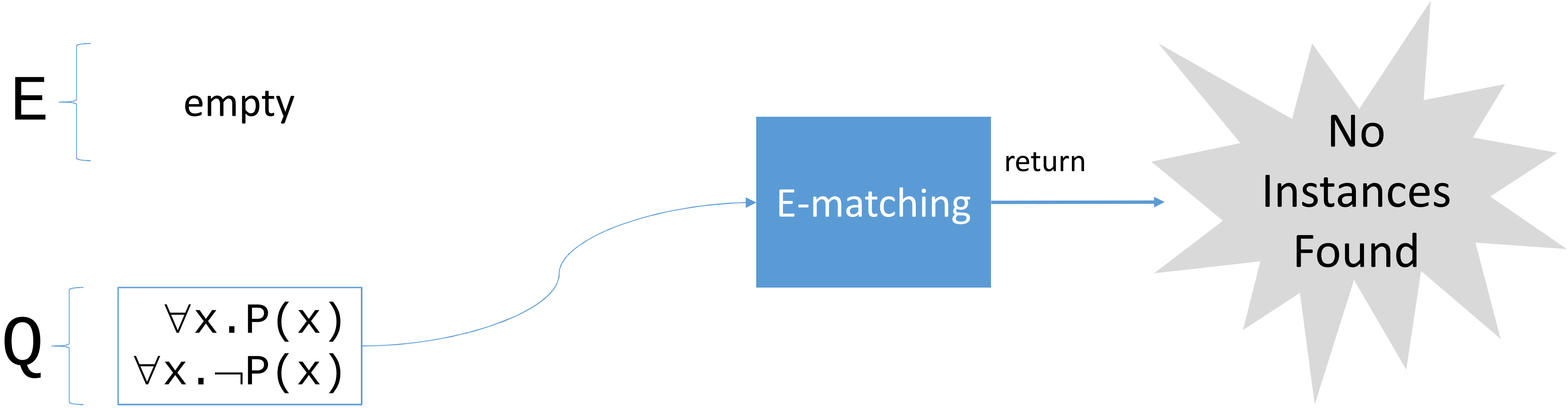
$\forall x.P(x)$

$\forall x.\neg P(x)$

E-matching

$\Rightarrow$ E-matching is an incomplete procedure

Challenge : Incompleteness



E

empty

Q

$\forall x. P(x)$

$\forall x. \neg P(x)$

E-matching

return

No Instances Found

⇒ If E-matching produces no instances,
this *does not guarantee EĆQ is T-satisfiable*

Challenge : Incompleteness

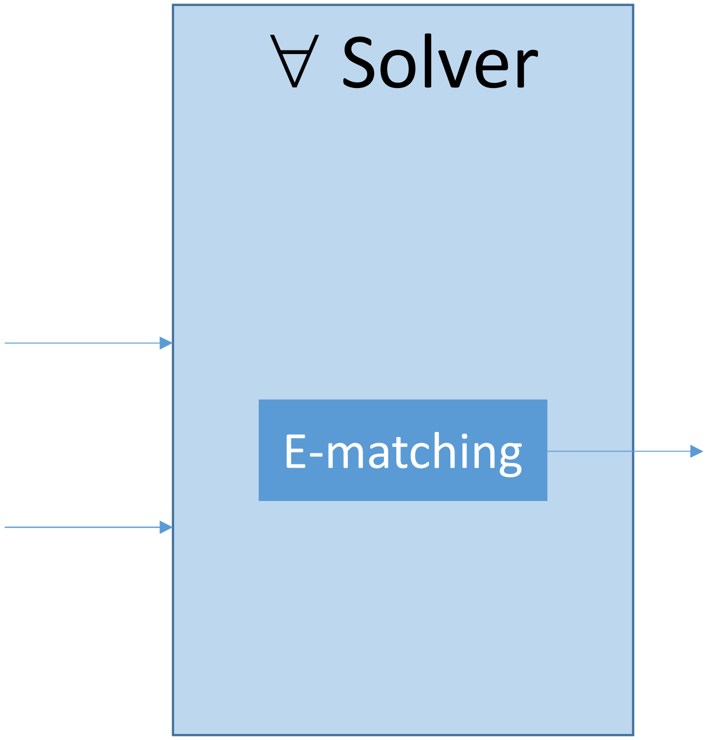
- E-matching is incomplete
  - It may be non-terminating
  - When it terminates, we generally cannot answer “$E \cup Q$ is T-satisfiable”
    - Although for some fragments+variants, we may guarantee (termination ⇔ “sat”)
      - Decision Procedures via Triggers **[Dross et al 13]**
      - Local Theory Extensions **[Bansal et al 15]**

      Ĝ Typically are established by a separate pencil-and-paper proof

E-matching : Challenges Addressed

- What if there are **too many instances**?

- What if there are **no instances**, and problem maybe “**sat**”?

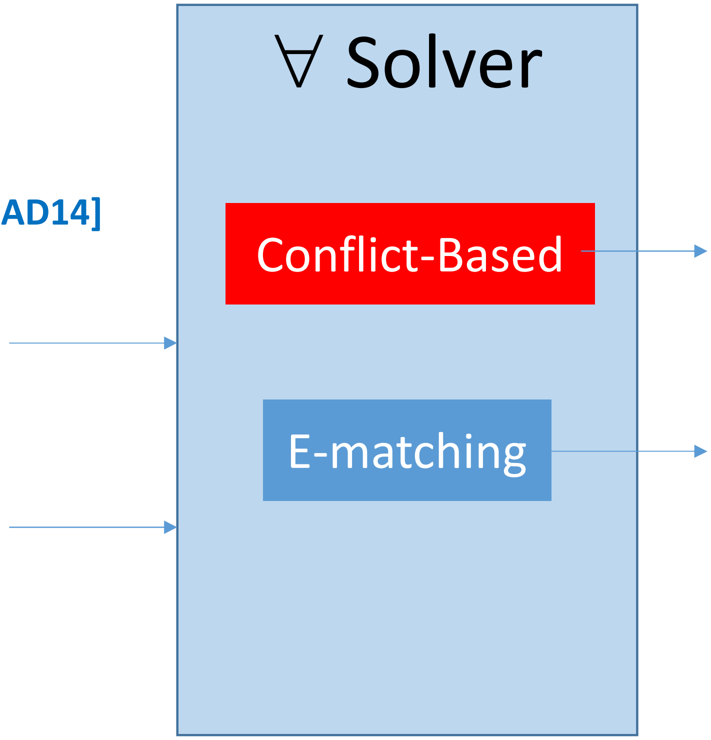


E-matching : Challenges Addressed

- What if there are **too many instances**?

  ⇒*Use conflict-based instantiation* **[Reynolds et al FMCAD14]**

- What if there are no instances, and problem maybe "sat"?



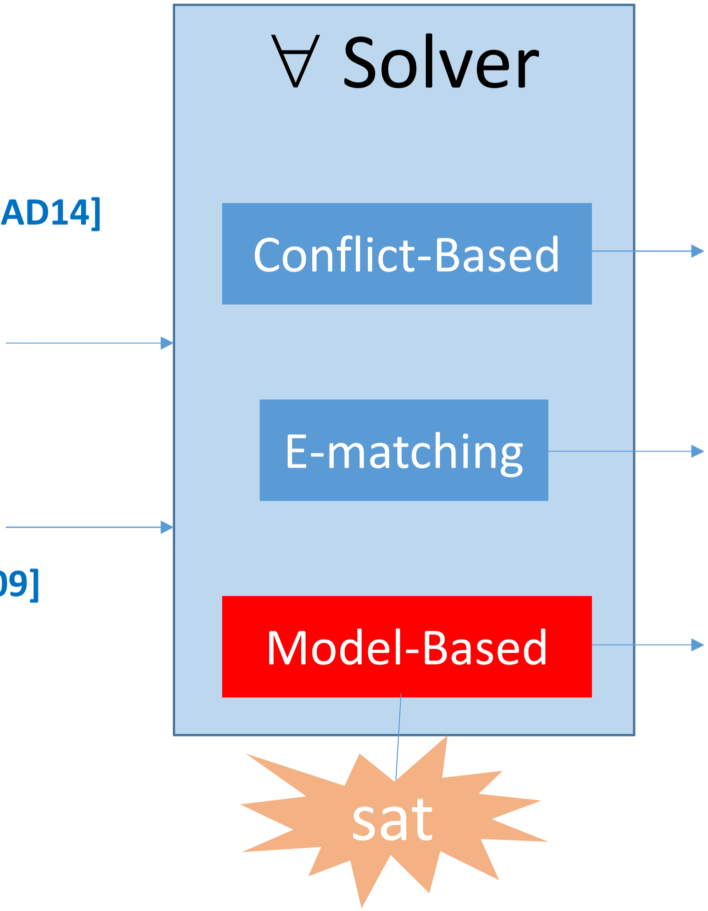
E-matching : Challenges Addressed

- What if there are too many instances?

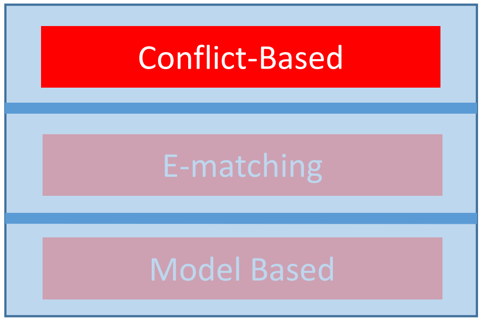
  ⇒Use conflict-based instantiation [Reynolds et al FMCAD14]

- What if there are **no instances**, and problem maybe “**sat**”?

  ⇒*Use model-based instantiation* [Ge/deMoura CAV2009]



Conflict-Based Instantiation



Conflict-Based

E-matching

Model Based

- Basic idea:
  - Since we are interested in whether e.g. $E, \forall x.P(x)$ is satisfiable,
    - Try to find one “conflict instance” such that $E, P(a) \models \bot$
    - If this is possible, don’t run E-matching

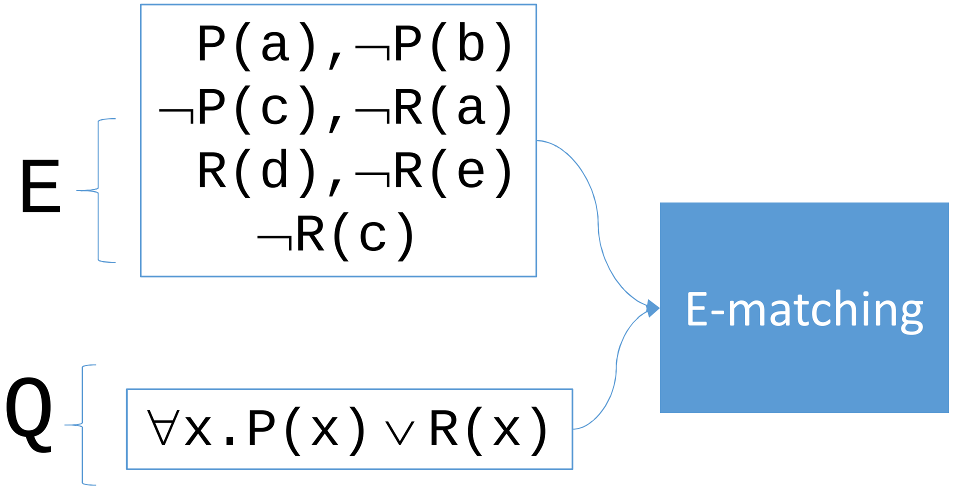
Ĝ *Leads to fewer instances, improved ability to answer* unsat

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

$P(a), \neg P(b)$

$\neg P(c), \neg R(a)$

$R(d), \neg R(e)$

$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

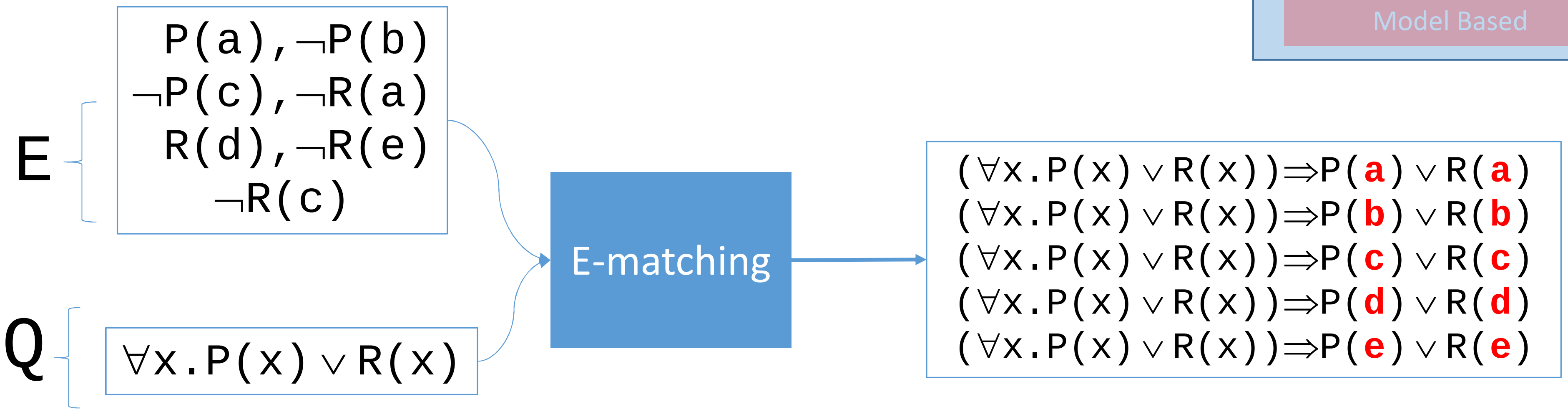
E-matching

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

$P(a), \neg P(b)$

$\neg P(c), \neg R(a)$

$R(d), \neg R(e)$

$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

E-matching

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

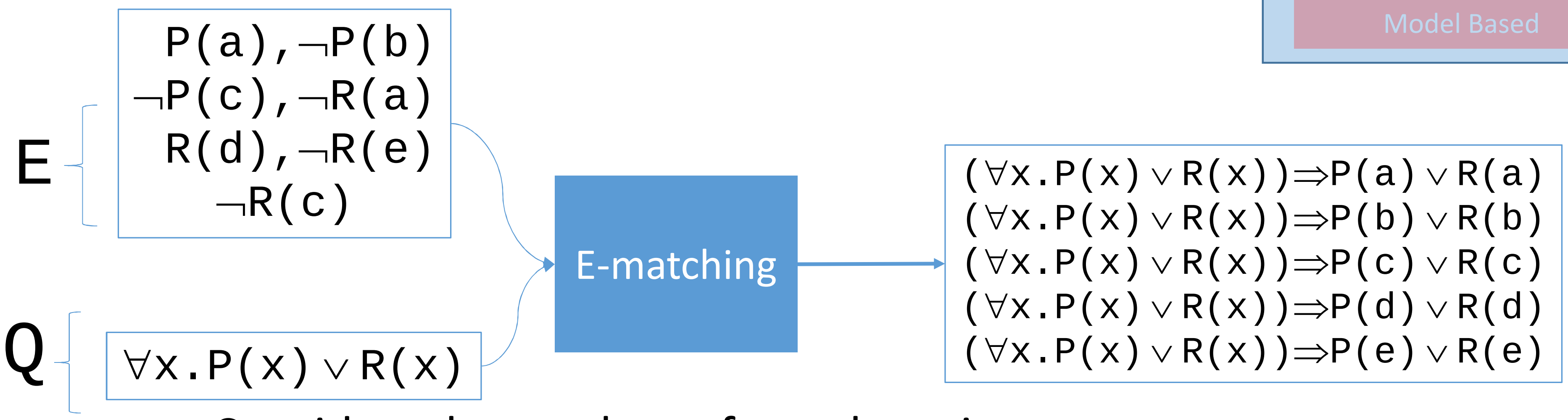
$\Rightarrow$ E-matching would produce $\{x \rightarrow a\}$, $\{x \rightarrow b\}$, $\{x \rightarrow c\}$, $\{x \rightarrow d\}$, $\{x \rightarrow e\}$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

$P(a), \neg P(b)$
$\neg P(c), \neg R(a)$
$R(d), \neg R(e)$
$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

E-matching

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

$\Rightarrow$ Consider what we learn from these instances:

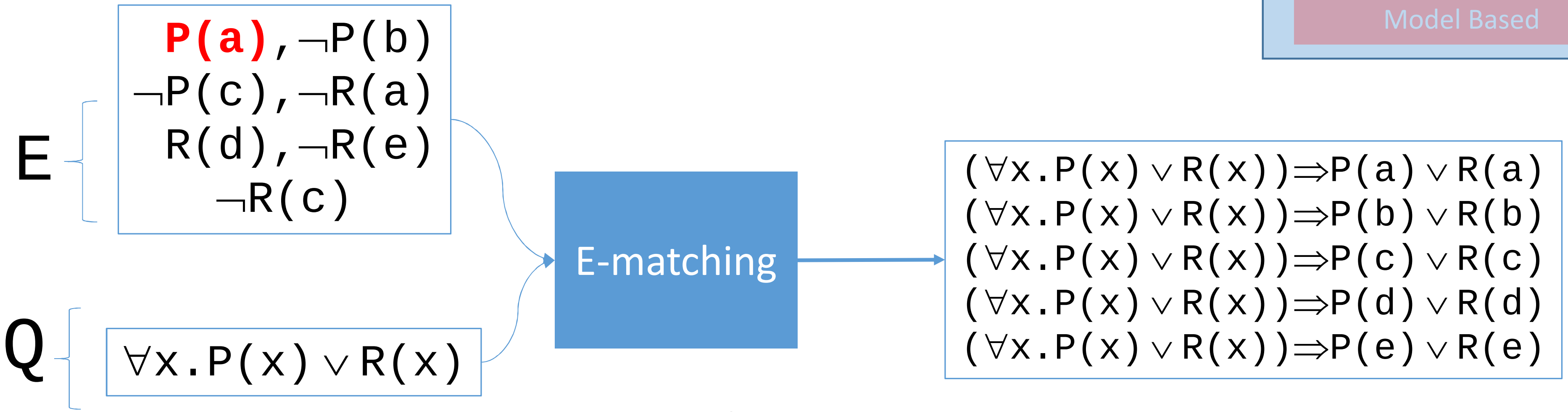
$$E, P(a) \vee R(a) \models P(a) \vee R(a)$$
$$E, P(b) \vee R(b) \models P(b) \vee R(b)$$
$$E, P(c) \vee R(c) \models P(c) \vee R(c)$$
$$E, P(d) \vee R(d) \models P(d) \vee R(d)$$
$$E, P(e) \vee R(e) \models P(e) \vee R(e)$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation


E
$P(a), \neg P(b)$
$\neg P(c), \neg R(a)$
$R(d), \neg R(e)$
$\neg R(c)$
Q
$\forall x. P(x) \vee R(x)$
E-matching
$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

$\Rightarrow$ Consider what we learn from these instances:

$$
\begin{aligned}
E, P(a) \vee R(a) &\models T \vee R(a) \\
E, P(b) \vee R(b) &\models P(b) \vee R(b) \\
E, P(c) \vee R(c) &\models P(c) \vee R(c) \\
E, P(d) \vee R(d) &\models P(d) \vee R(d) \\
E, P(e) \vee R(e) &\models P(e) \vee R(e)
\end{aligned}
$$

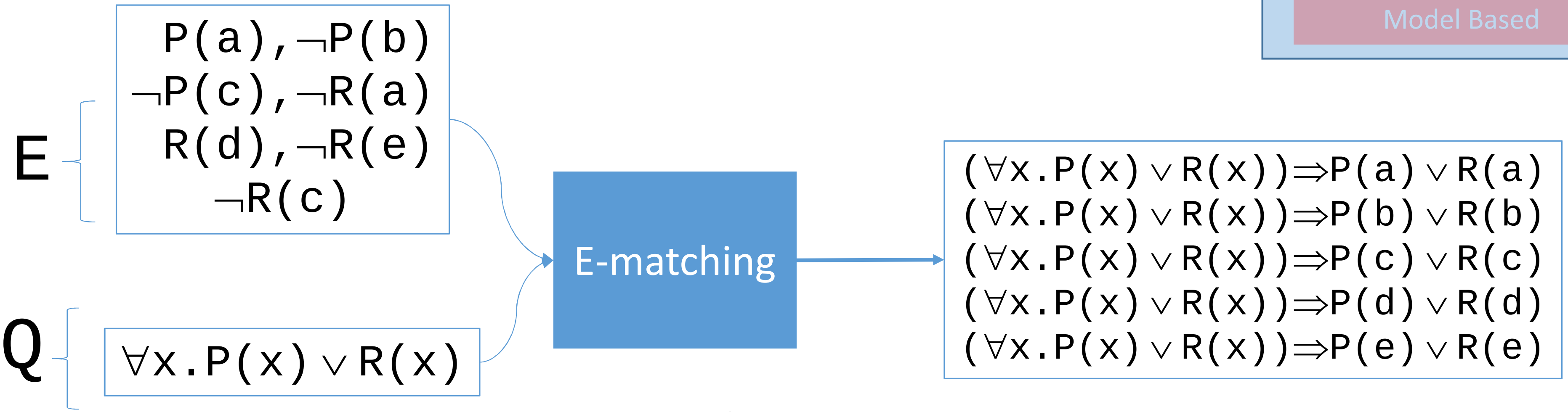
By E, we know $P(a) \Leftrightarrow T$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E
$P(a), \neg P(b)$
$\neg P(c), \neg R(a)$
$R(d), \neg R(e)$
$\neg R(c)$
Q
$\forall x. P(x) \vee R(x)$
E-matching
$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

$\Rightarrow$ Consider what we learn from these instances:

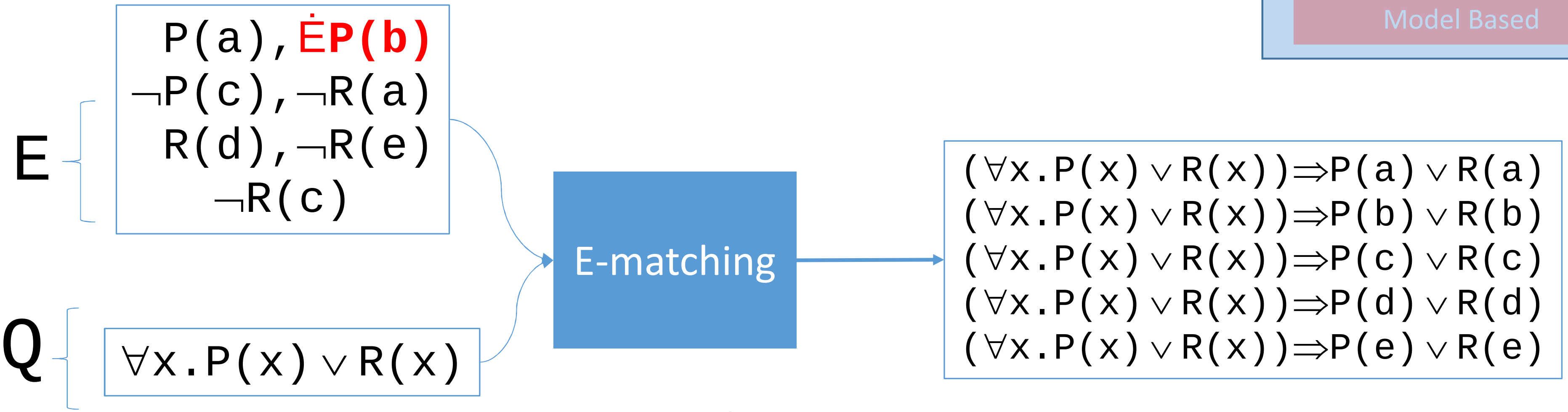
$$
\begin{array}{lcl}
E, P(a) \vee R(a) & \models & \top \\
E, P(b) \vee R(b) & \models & P(b) \vee R(b) \\
E, P(c) \vee R(c) & \models & P(c) \vee R(c) \\
E, P(d) \vee R(d) & \models & P(d) \vee R(d) \\
E, P(e) \vee R(e) & \models & P(e) \vee R(e)
\end{array}
$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

P(a),ĖP(b)
¬P(c),¬R(a)
R(d),¬R(e)
¬R(c)

Q

∀x.P(x)∨R(x)

E-matching

(∀x.P(x)∨R(x))⇒P(a)∨R(a)
(∀x.P(x)∨R(x))⇒P(b)∨R(b)
(∀x.P(x)∨R(x))⇒P(c)∨R(c)
(∀x.P(x)∨R(x))⇒P(d)∨R(d)
(∀x.P(x)∨R(x))⇒P(e)∨R(e)

⇒ Consider what we learn from these instances:

$$
\begin{array}{lcl}
E, P(a) \vee R(a) & \models & T \\
E, P(b) \vee R(b) & \models & \bot \vee R(b) \\
E, P(c) \vee R(c) & \models & P(c) \vee R(c) \\
E, P(d) \vee R(d) & \models & P(d) \vee R(d) \\
E, P(e) \vee R(e) & \models & P(e) \vee R(e)
\end{array}
$$

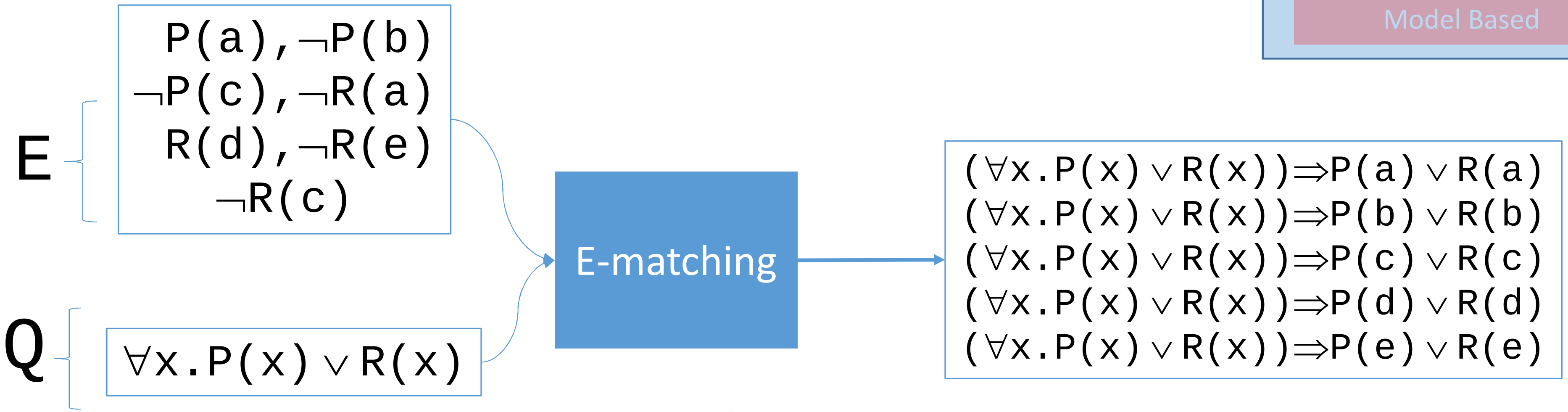
We know **P(b)** ⇔ {

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

$P(a), \neg P(b)$
$\neg P(c), \neg R(a)$
$R(d), \neg R(e)$
$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

E-matching

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

$\Rightarrow$ Consider what we learn from these instances:

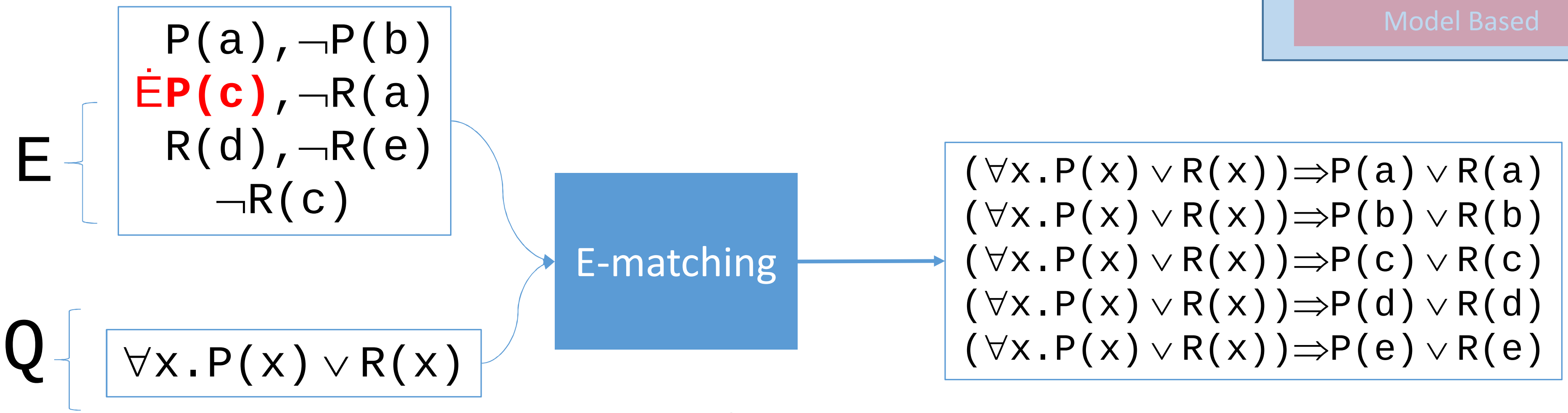
$$\begin{array}{lcl} E, P(a) \vee R(a) & \models & \top \\ E, P(b) \vee R(b) & \models & R(b) \\ E, P(c) \vee R(c) & \models & P(c) \vee R(c) \\ E, P(d) \vee R(d) & \models & P(d) \vee R(d) \\ E, P(e) \vee R(e) & \models & P(e) \vee R(e) \end{array}$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

P(a), ¬P(b)
Ė**P(c)**, ¬R(a)
R(d), ¬R(e)
¬R(c)

Q

$\forall x.P(x) \lor R(x)$

E-matching

$(\forall x.P(x) \lor R(x)) \Rightarrow P(a) \lor R(a)$
$(\forall x.P(x) \lor R(x)) \Rightarrow P(b) \lor R(b)$
$(\forall x.P(x) \lor R(x)) \Rightarrow P(c) \lor R(c)$
$(\forall x.P(x) \lor R(x)) \Rightarrow P(d) \lor R(d)$
$(\forall x.P(x) \lor R(x)) \Rightarrow P(e) \lor R(e)$

$\Rightarrow$ Consider what we learn from these instances:

$$\begin{array}{lcl} E, P(a) \lor R(a) & \models & T \\ E, P(b) \lor R(b) & \models & R(b) \\ E, P(c) \lor R(c) & \models & R(c) \\ E, P(d) \lor R(d) & \models & P(d) \lor R(d) \\ E, P(e) \lor R(e) & \models & P(e) \lor R(e) \end{array}$$

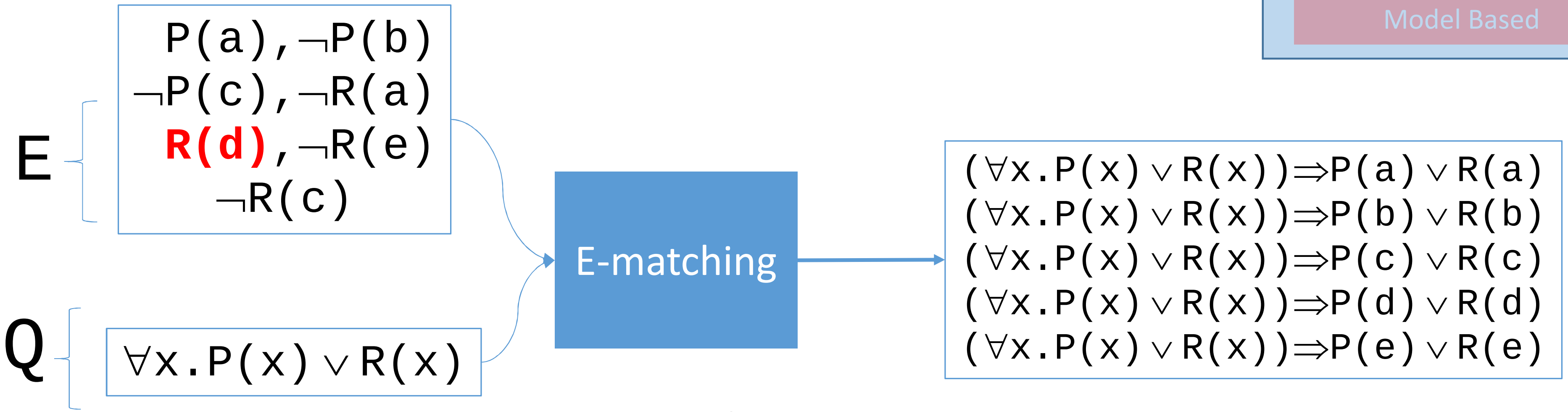
We know **P(c)** ⇔ {

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

$P(a), \neg P(b)$

$\neg P(c), \neg R(a)$

$\mathbf{R(d)}, \neg R(e)$

$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

E-matching

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

$\Rightarrow$ Consider what we learn from these instances:

$$E, P(a) \vee R(a) \models T$$
$$E, P(b) \vee R(b) \models R(b)$$
$$E, P(c) \vee R(c) \models R(c)$$
$$E, P(d) \vee R(d) \models T$$
$$E, P(e) \vee R(e) \models P(e) \vee R(e)$$

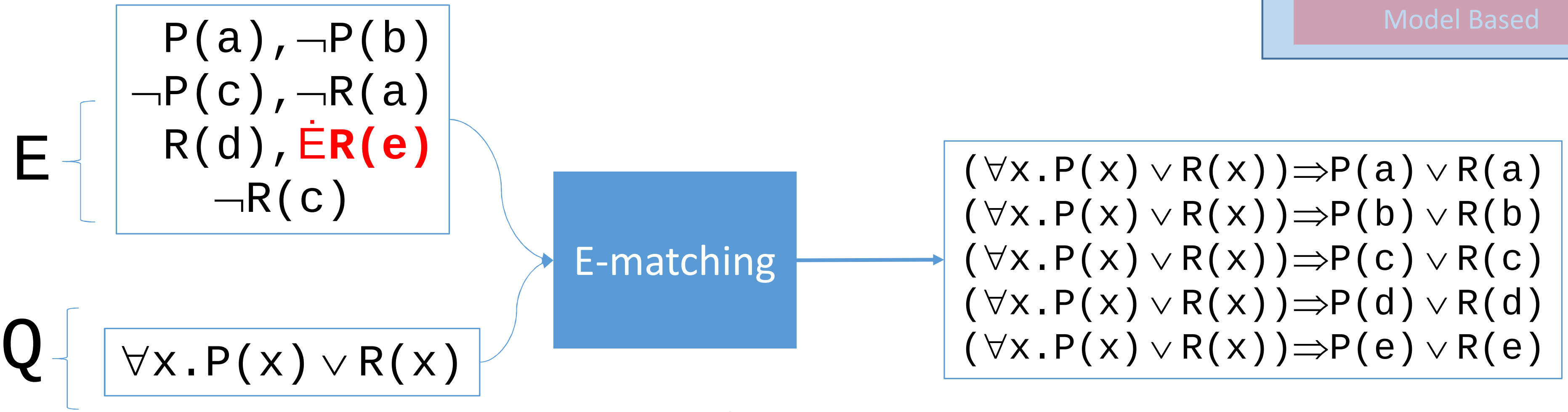
We know $\mathbf{R(d)} \Leftrightarrow \mathbf{T}$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E
$P(a), \neg P(b)$
$\neg P(c), \neg R(a)$
$R(d),$ **$\dot{E}R(e)$**
$\neg R(c)$
Q
$\forall x. P(x) \vee R(x)$
E-matching
$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

$\Rightarrow$ Consider what we learn from these instances:

$$\begin{array}{ll}
E, P(a) \vee R(a) & \models \top \\
E, P(b) \vee R(b) & \models R(b) \\
E, P(c) \vee R(c) & \models R(c) \\
E, P(d) \vee R(d) & \models \top \\
E, P(e) \vee R(e) & \models P(e)
\end{array}$$

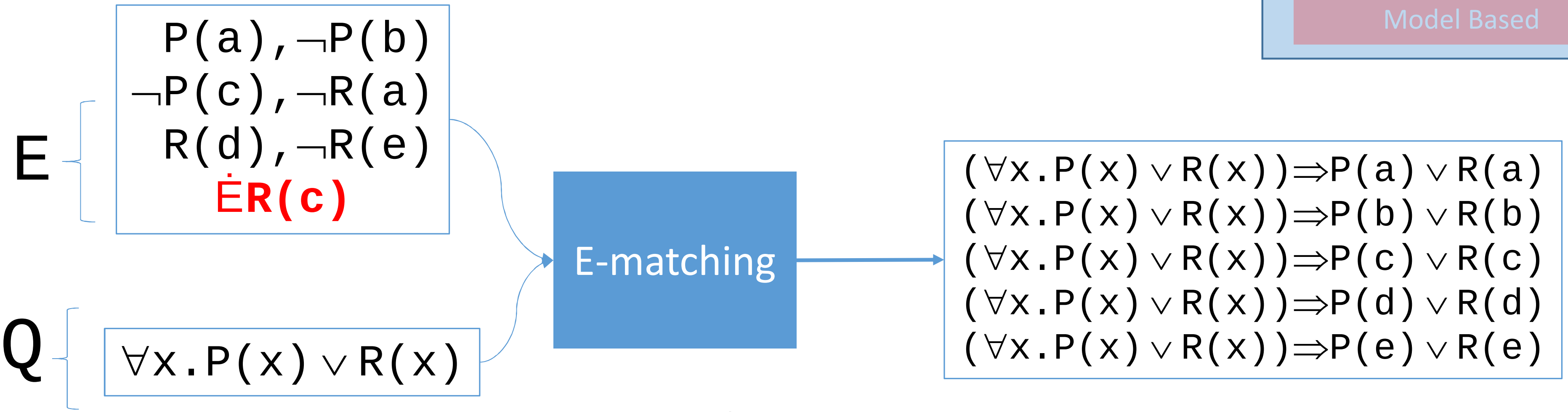
We know **R(e)** $\Leftrightarrow$ {

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



$P(a), \neg P(b)$
$\neg P(c), \neg R(a)$
$R(d), \neg R(e)$
ĖR(c)

$\forall x. P(x) \vee R(x)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

⇒ Consider what we learn from these instances:

$E, P(a) \vee R(a) \models T$
$E, P(b) \vee R(b) \models R(b)$
$E, P(c) \vee R(c) \models \{$
$E, P(d) \vee R(d) \models T$
$E, P(e) \vee R(e) \models P(e)$

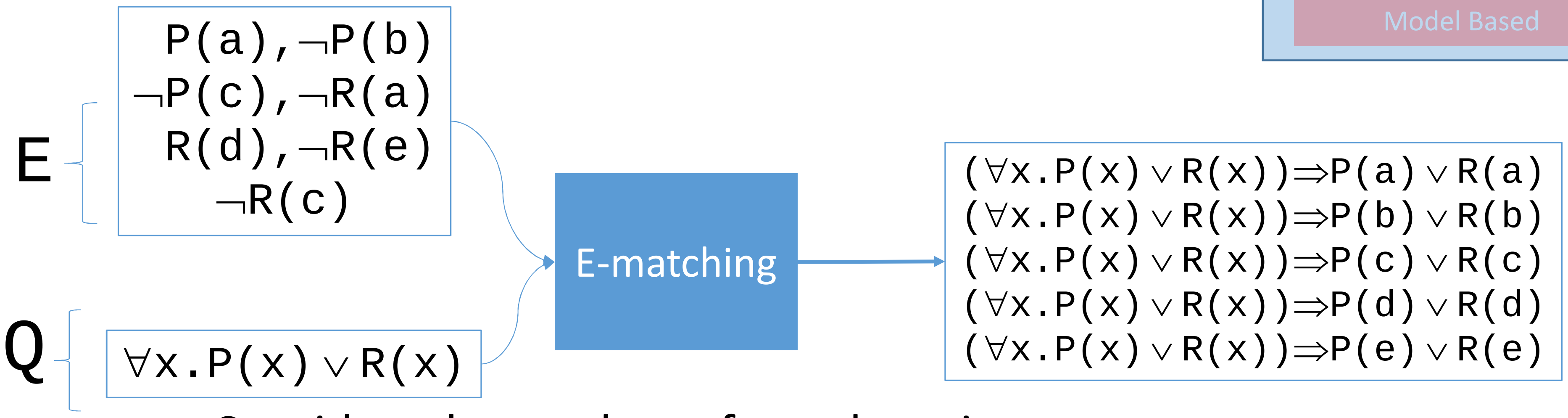
We know **R(c)**⇔ {

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

$P(a), \neg P(b)$
$\neg P(c), \neg R(a)$
$R(d), \neg R(e)$
$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

E-matching

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(b) \vee R(b)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(d) \vee R(d)$
$(\forall x. P(x) \vee R(x)) \Rightarrow P(e) \vee R(e)$

$\Rightarrow$ Consider what we learn from these instances:

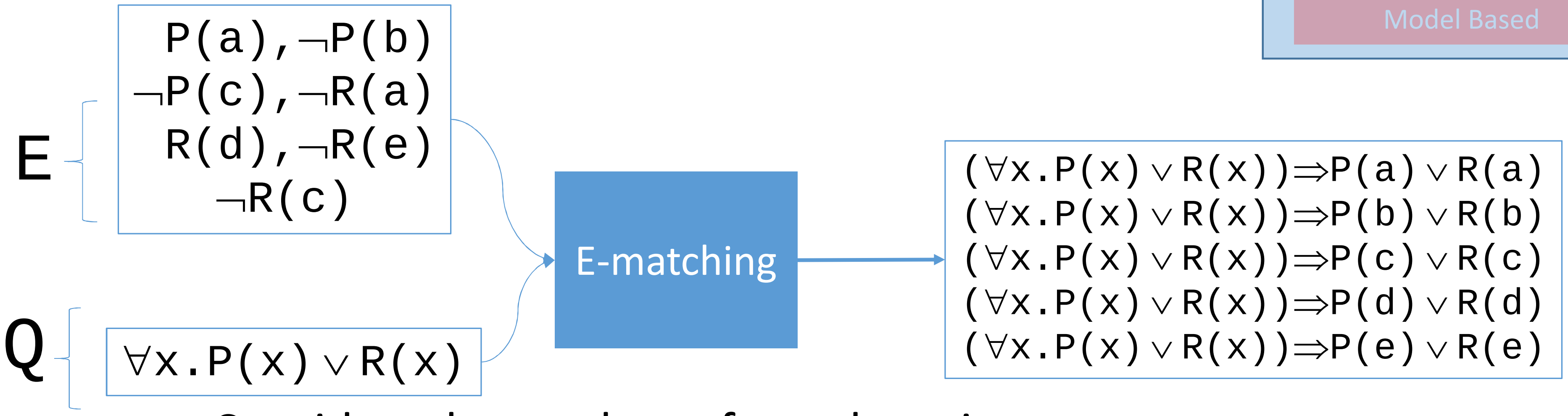
$$E, P(a) \vee R(a) \models T$$
$$E, P(b) \vee R(b) \models R(b)$$
$$E, P(c) \vee R(c) \models \{$$
$$E, P(d) \vee R(d) \models T$$
$$E, P(e) \vee R(e) \models P(e)$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

P(a), ¬P(b)
¬P(c), ¬R(a)
R(d), ¬R(e)
¬R(c)

Q

∀x.P(x) ∨ R(x)

E-matching

(∀x.P(x) ∨ R(x))⇒P(a) ∨ R(a)
(∀x.P(x) ∨ R(x))⇒P(b) ∨ R(b)
(∀x.P(x) ∨ R(x))⇒P(c) ∨ R(c)
(∀x.P(x) ∨ R(x))⇒P(d) ∨ R(d)
(∀x.P(x) ∨ R(x))⇒P(e) ∨ R(e)

⇒ Consider what we learn from these instances:

E,P(a) ∨ R(a) ╞ T
E,P(b) ∨ R(b) ╞ R(b)
E,P(c) Ę R(c) ╞ {
E,P(d) ∨ R(d) ╞ T
E,P(e) ∨ R(e) ╞ P(e)

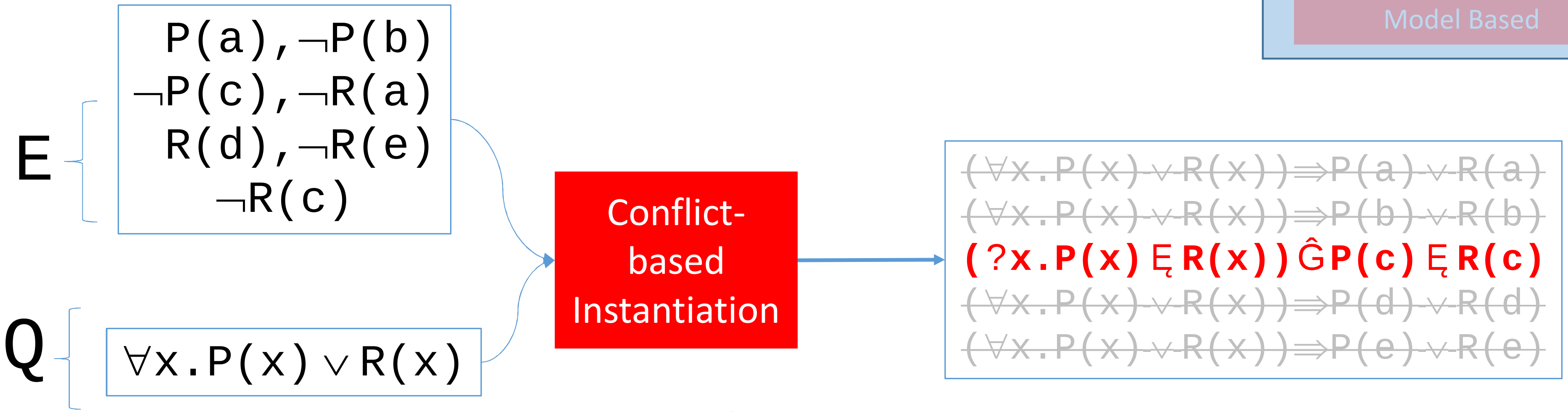
*P(c) Ę R(c) is a **conflicting instance** for (E, Q)!*

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



E

P(a),¬P(b)
¬P(c),¬R(a)
R(d),¬R(e)
¬R(c)

Q

∀x.P(x)∨R(x)

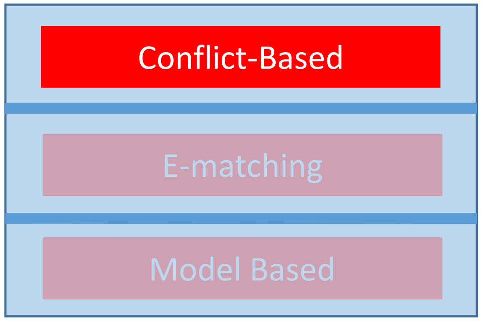
Conflict-based Instantiation

~~(∀x.P(x)∨R(x))⇒P(a)∨R(a)~~
~~(∀x.P(x)∨R(x))⇒P(b)∨R(b)~~
(?x.P(x) Ę R(x)) ĜP(c) Ę R(c)
~~(∀x.P(x)∨R(x))⇒P(d)∨R(d)~~
~~(∀x.P(x)∨R(x))⇒P(e)∨R(e)~~

⇒ Consider what we learn from these instances:

E,P(a)∨R(a) ⊨ T
E,P(b)∨R(b) ⊨ R(b)
E,P(c)∨R(c) ⊨ {
E,P(d)∨R(d) ⊨ T
E,P(e)∨R(e) ⊨ P(e)

Since P(c)∨R(c)
suffices to derive {,
return ***only*** this instance

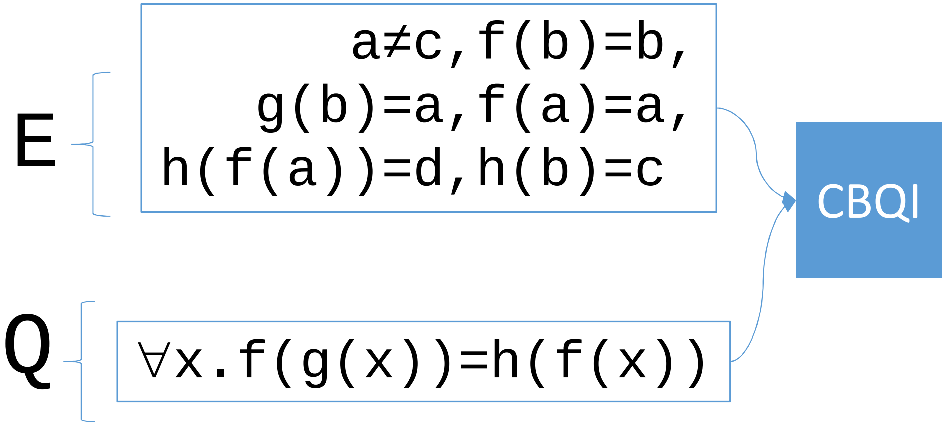


Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



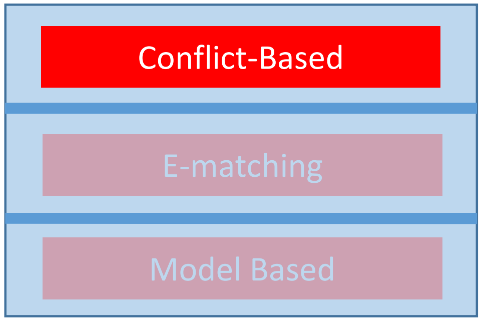
E

a≠c,f(b)=b,
g(b)=a,f(a)=a,
h(f(a))=d,h(b)=c

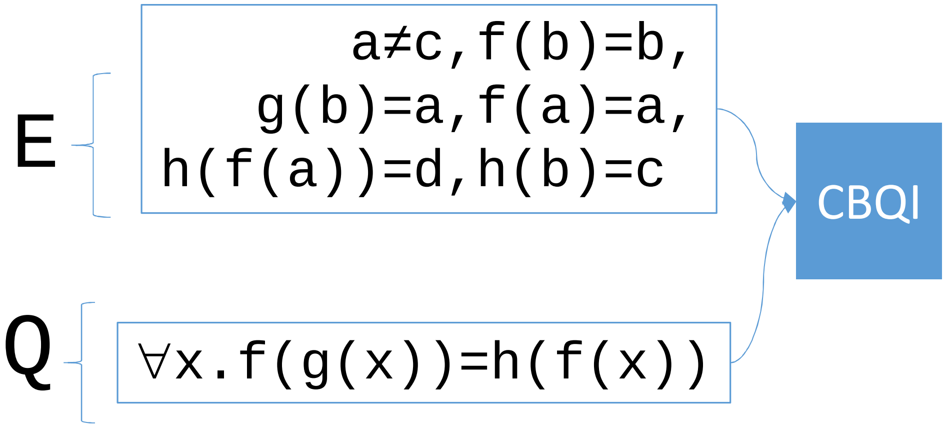
Q

∀x.f(g(x))=h(f(x))

CBQI



Conflict-Based Instantiation: EUF

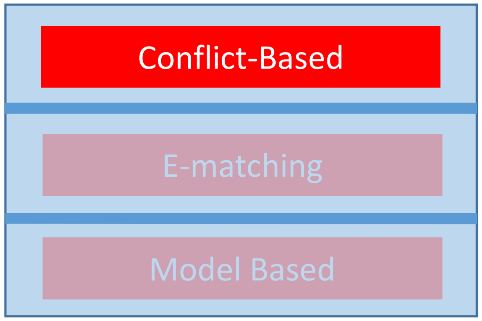


$a \neq c, f(b)=b,$
$g(b)=a, f(a)=a,$
$h(f(a))=d, h(b)=c$

$\forall x. f(g(x))=h(f(x))$

$\Rightarrow$ Consider the instance $\forall x. f(g(x))=h(f(x)) \Rightarrow f(g(\mathbf{b}))=h(f(\mathbf{b}))$

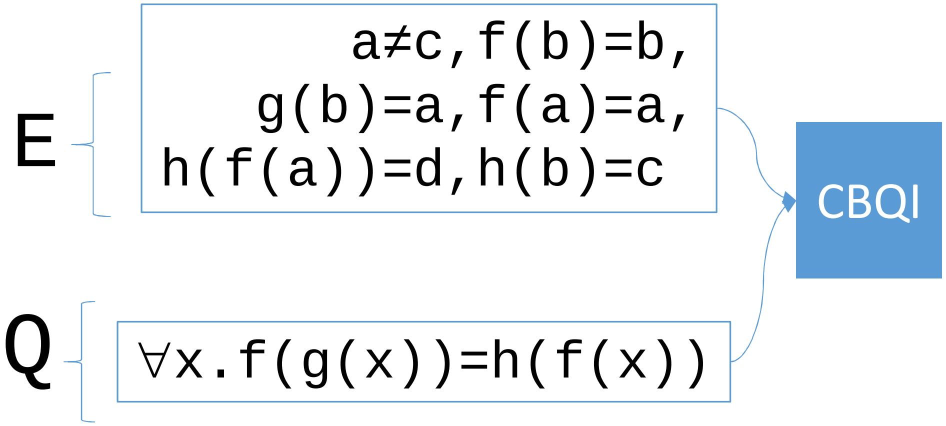
- Is this conflicting for (E, Q)?


Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF


E

$a \neq c, f(b)=b,$
$g(b)=a, f(a)=a,$
$h(f(a))=d, h(b)=c$

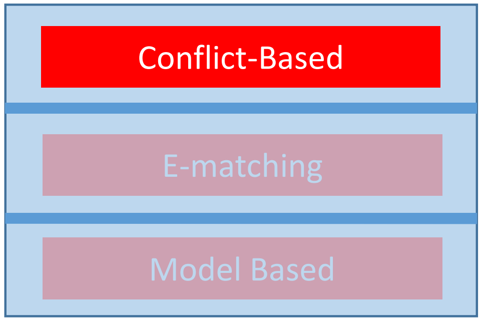
Q

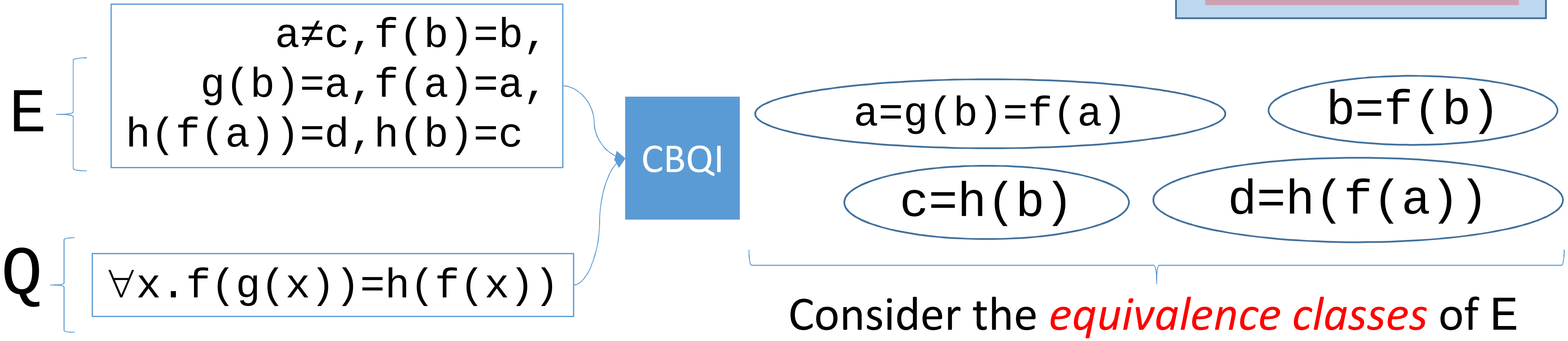
$\forall x. f(g(x))=h(f(x))$

CBQI

$$E, f(g(b))=h(f(b)) \models_E f(g(b))=h(f(b))$$

Conflict-Based Instantiation: EUF





$a \neq c, f(b)=b,$
$g(b)=a, f(a)=a,$
$h(f(a))=d, h(b)=c$

$\forall x. f(g(x))=h(f(x))$

$a=g(b)=f(a)$

$b=f(b)$

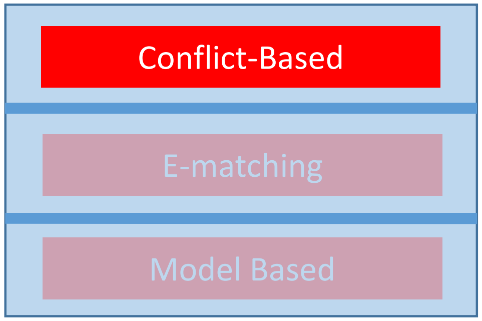
$c=h(b)$

$d=h(f(a))$

Consider the *equivalence classes* of E

$$E, f(g(b))=h(f(b)) \models_E f(g(b))=h(f(b))$$

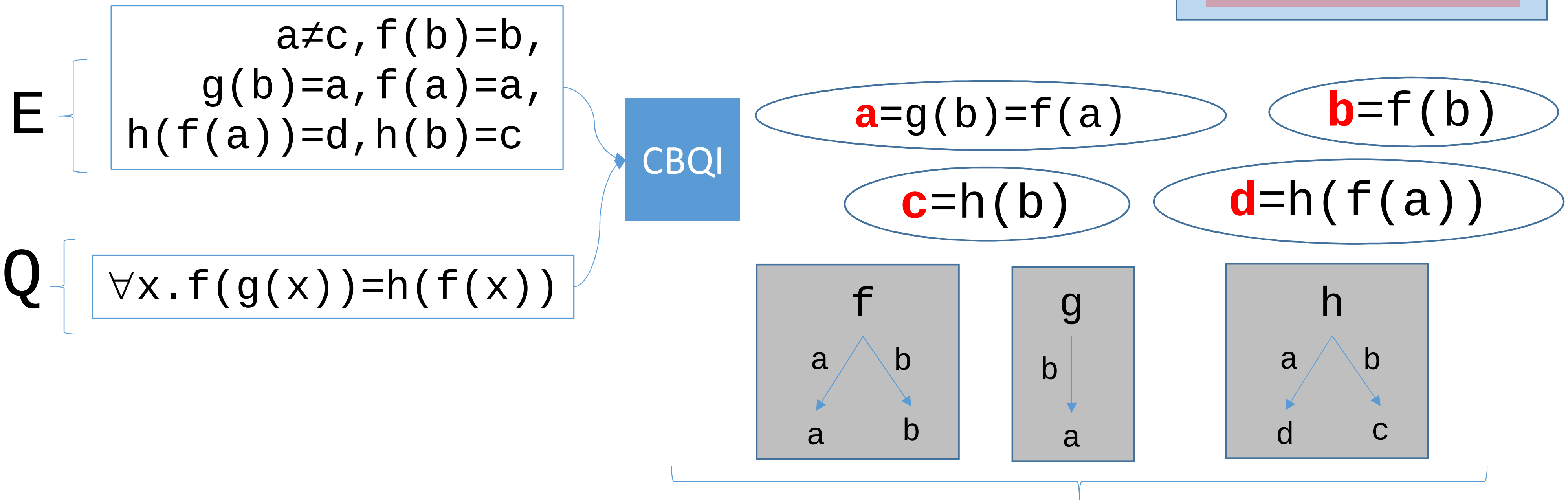
Conflict-Based Instantiation: EUF



Conflict-Based

E-matching

Model Based



E

a≠c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c

Q

∀x.f(g(x))=h(f(x))

CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))

Build partial definitions for functions in terms of *representatives*

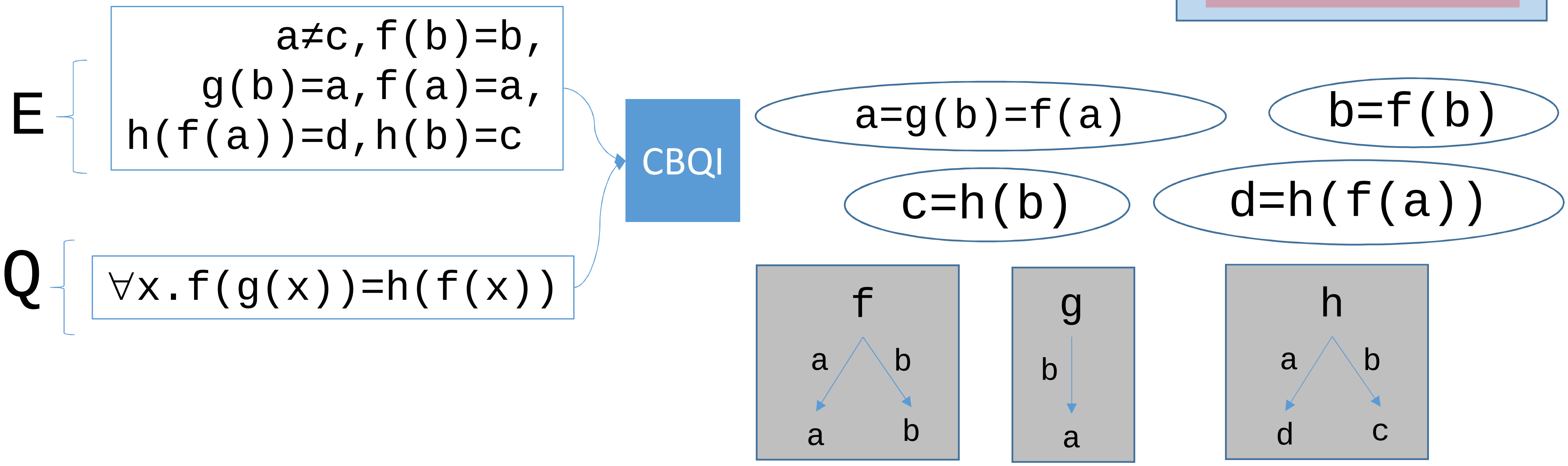
$$E, f(g(b))=h(f(b)) \models_E f(g(b))=h(f(b))$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



E

$a \neq c, f(b)=b,$

$g(b)=a, f(a)=a,$

$h(f(a))=d, h(b)=c$

Q

$\forall x. f(g(x))=h(f(x))$

CBQI

$a=g(b)=f(a)$

$b=f(b)$

$c=h(b)$

$d=h(f(a))$

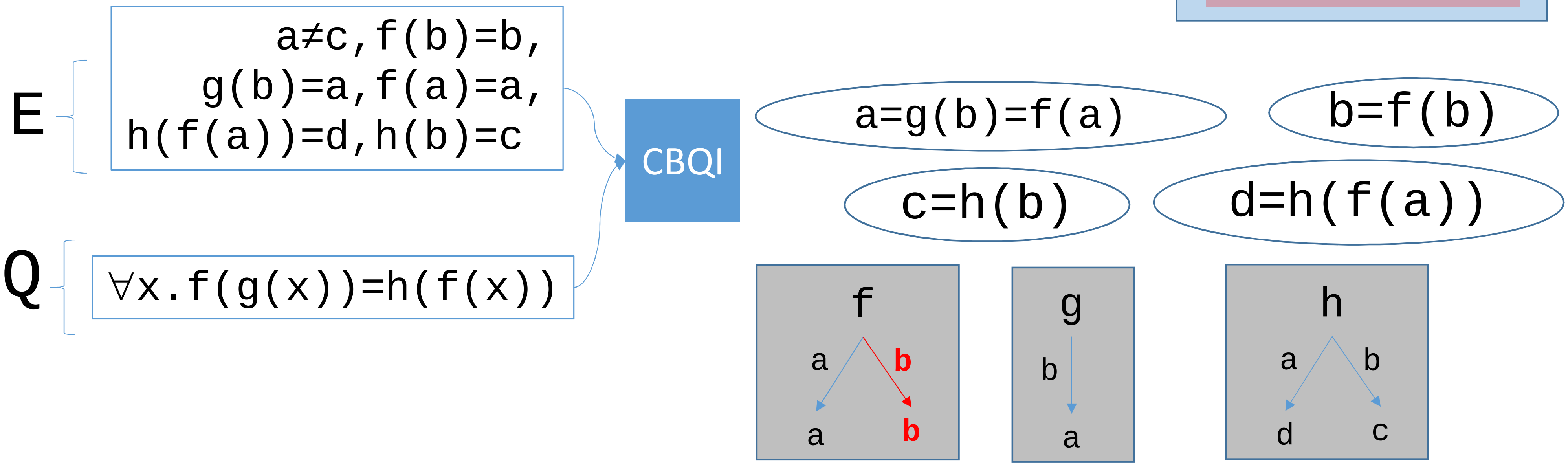
$$E, f(g(b))=h(f(b)) \models_E f(g(b))=h(f(b))$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



E

$a \neq c, f(b)=b,$
$g(b)=a, f(a)=a,$
$h(f(a))=d, h(b)=c$

Q

$\forall x. f(g(x))=h(f(x))$

CBQI

$a=g(b)=f(a)$

$b=f(b)$

$c=h(b)$

$d=h(f(a))$

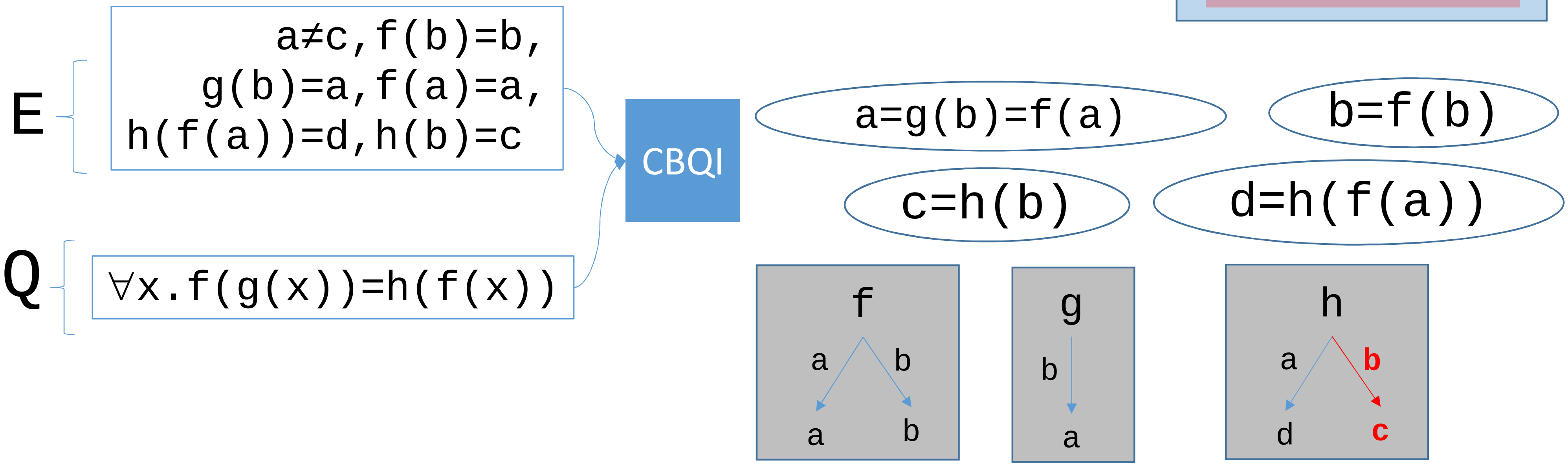
$$E, f(g(b))=h(f(b)) \models_E f(g(b))=h(\ b\ )$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



E: $a \neq c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c$

Q: $\forall x. f(g(x))=h(f(x))$

CBQI

$a=g(b)=f(a)$

$b=f(b)$

$c=h(b)$

$d=h(f(a))$

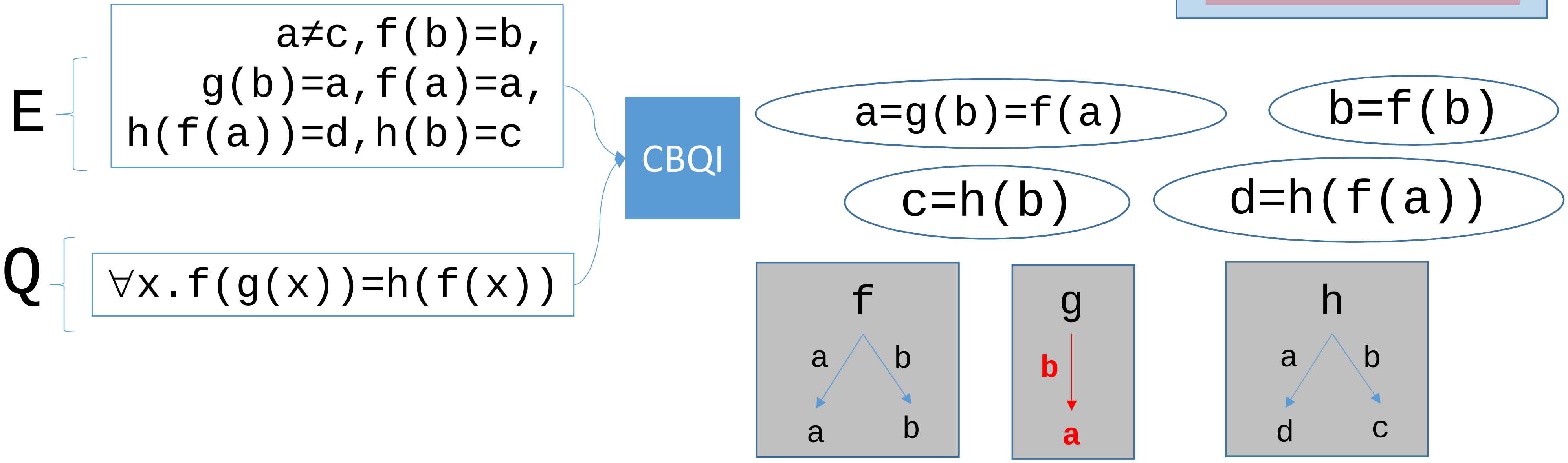
$$E, f(g(b))=h(f(b)) \models_E f(g(b))= \quad c$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



E

a≠c, f(b)=b,
g(b)=a, f(a)=a,
h(f(a))=d, h(b)=c

Q

∀x.f(g(x))=h(f(x))

CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))

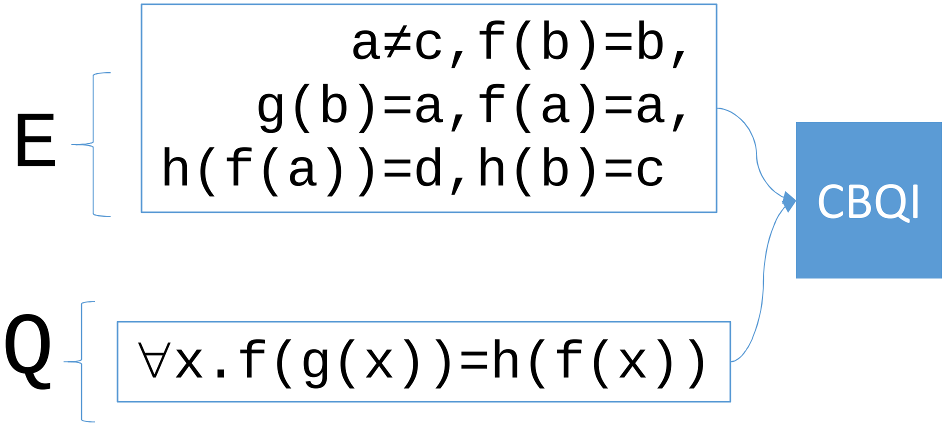
$E, f(g(b))=h(f(b)) \models_E f(\ a\ )=\ c$

Conflict-Based Instantiation: EUF

Conflict-Based

E-matching

Model Based



E

a≠c, f(b)=b,
g(b)=a, f(a)=a,
h(f(a))=d, h(b)=c

Q

∀x.f(g(x))=h(f(x))

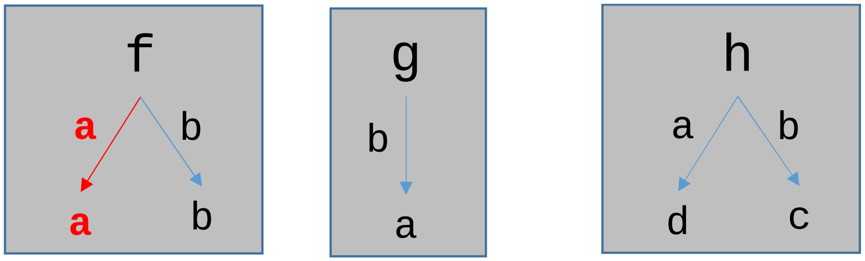
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



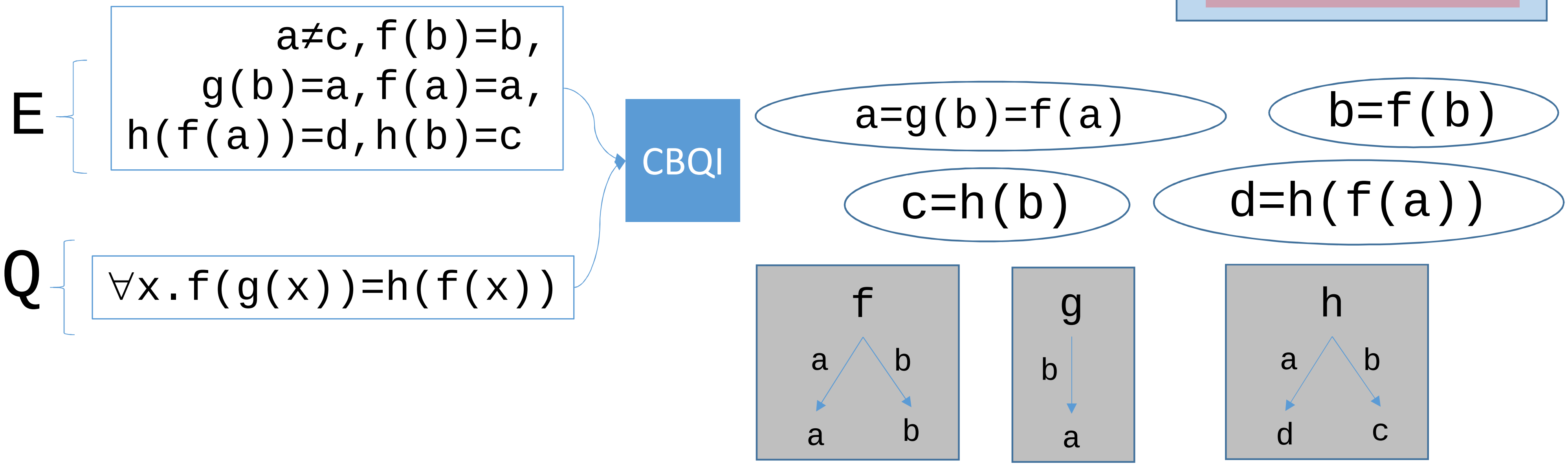
$E, f(g(b))=h(f(b)) \models_E \quad a \quad = \quad c$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



E

$a \neq c, f(b)=b,$
$g(b)=a, f(a)=a,$
$h(f(a))=d, h(b)=c$

Q

$\forall x. f(g(x))=h(f(x))$

CBQI

$a=g(b)=f(a)$

$b=f(b)$

$c=h(b)$

$d=h(f(a))$

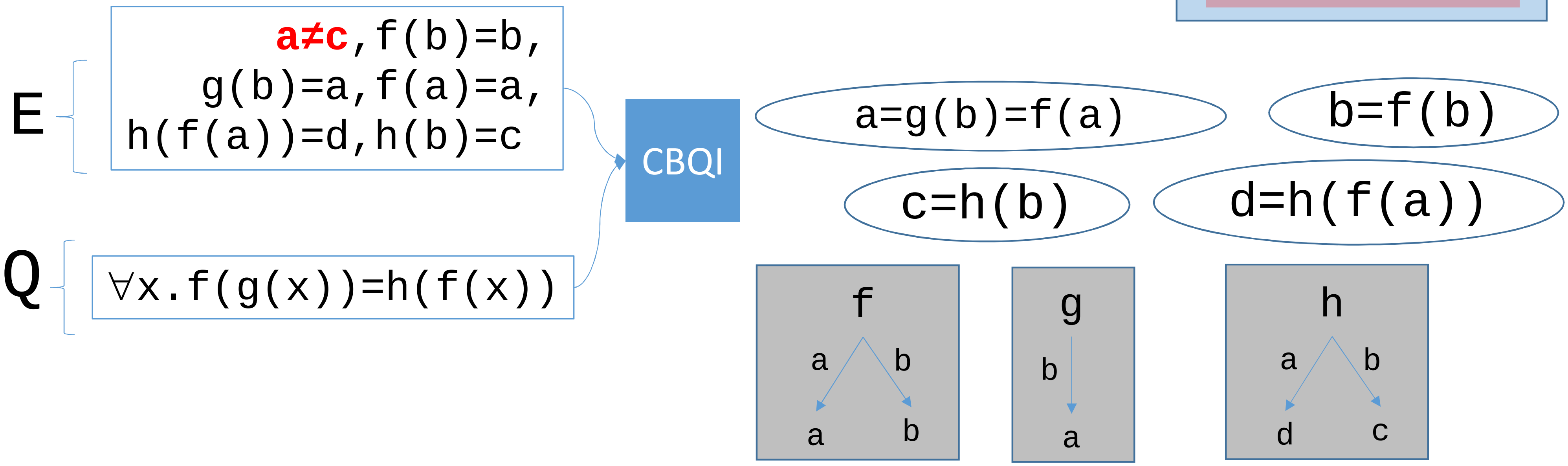
$$E, f(g(b))=h(f(b)) \models_E \quad a=c$$

Conflict-Based Instantiation: EUF

Conflict-Based

E-matching

Model Based



E

$a \neq c, f(b)=b,$
$g(b)=a, f(a)=a,$
$h(f(a))=d, h(b)=c$

Q

$\forall x. f(g(x))=h(f(x))$

CBQI

$a=g(b)=f(a)$

$b=f(b)$

$c=h(b)$

$d=h(f(a))$

$E, f(g(b))=h(f(b)) \models_E$ {

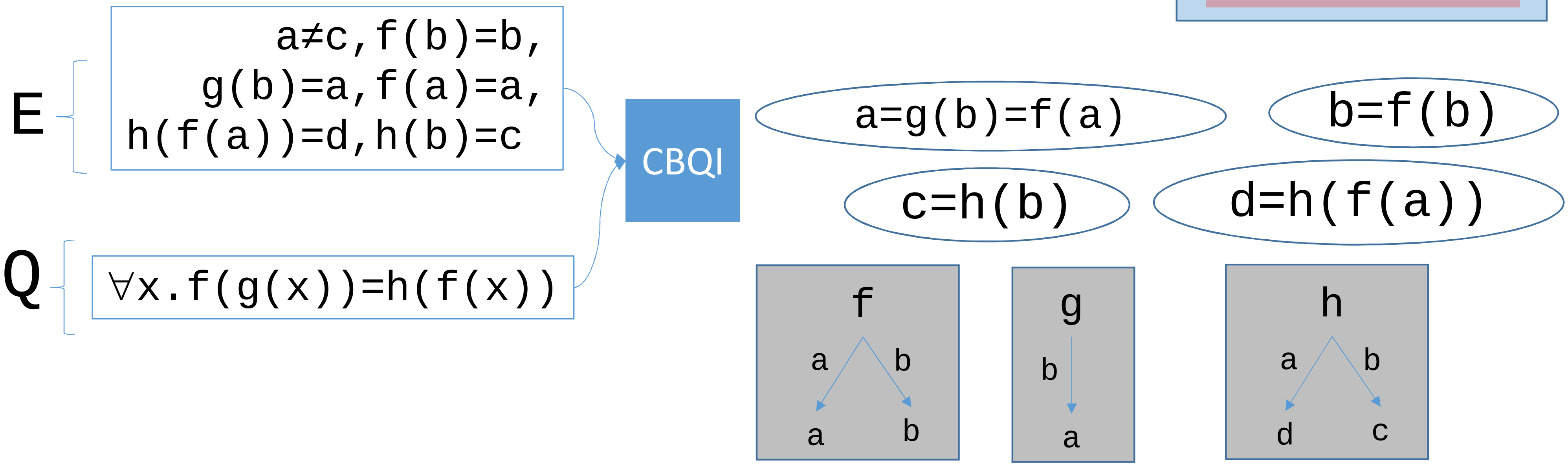
From E, we know **a≠c**

Conflict-Based Instantiation: EUF

Conflict-Based

E-matching

Model Based



E: a≠c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c

Q: ∀x.f(g(x))=h(f(x))

CBQI

a=g(b)=f(a)

b=f(b)

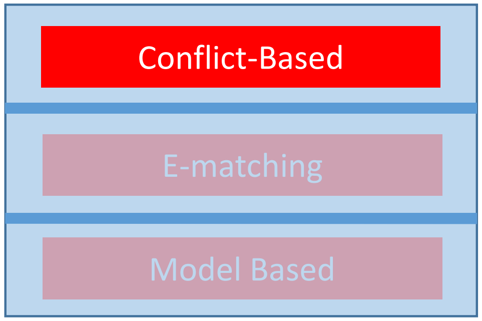
c=h(b)

d=h(f(a))

$E, f(g(b))=h(f(b)) \models_E \bot$

f(g(b))=h(f(b)) is a conflicting instance for (E,Q)!

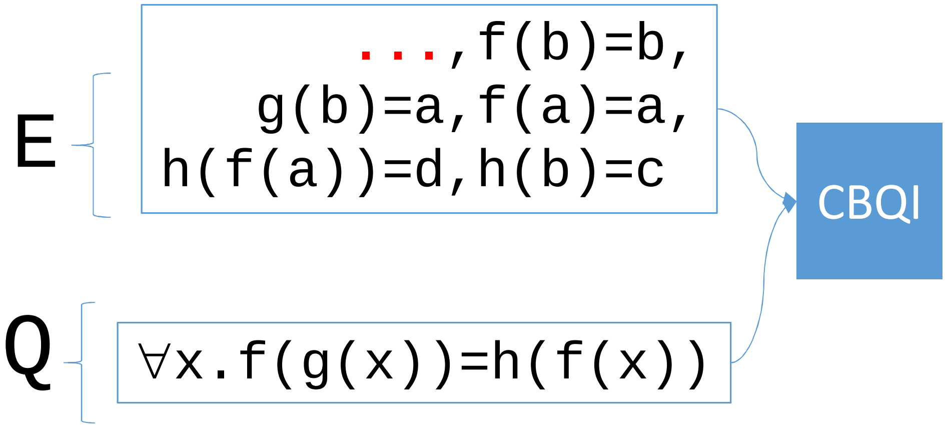
Conflict-Based Instantiation: EUF



Conflict-Based

E-matching

Model Based



E

`...,f(b)=b,`
`g(b)=a,f(a)=a,`
`h(f(a))=d,h(b)=c`

Q

`∀x.f(g(x))=h(f(x))`

CBQI

⇒ Consider the same example, but where we don't know a≠c

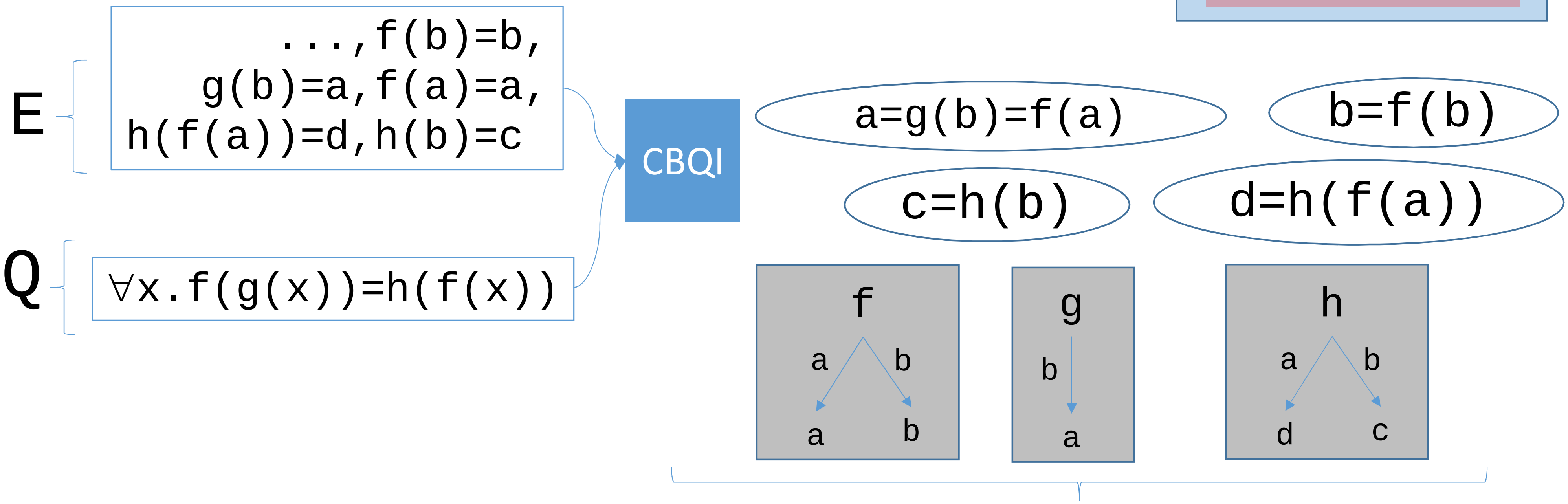
- Is the instance `f(g(b))=h(f(b))` still useful?

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



E

```
...,f(b)=b,
g(b)=a,f(a)=a,
h(f(a))=d,h(b)=c
```

Q

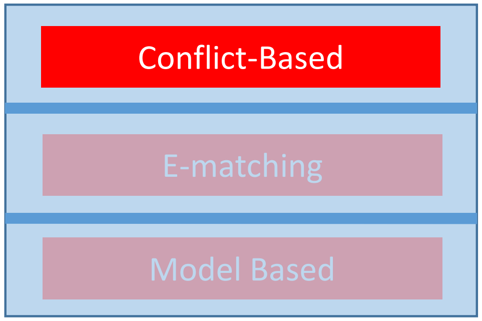
```
∀x.f(g(x))=h(f(x))
```

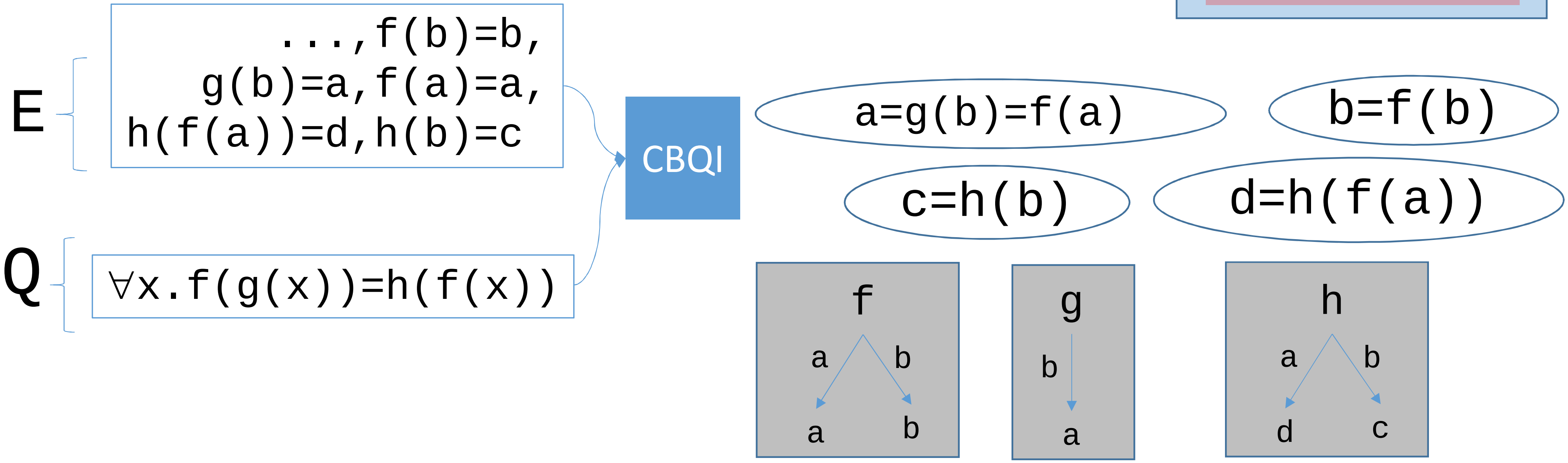
CBQI

```
a=g(b)=f(a)
b=f(b)
c=h(b)
d=h(f(a))
```

Build partial definitions

Conflict-Based Instantiation: EUF





`...,f(b)=b,`
`g(b)=a,f(a)=a,`
`h(f(a))=d,h(b)=c`

`∀x.f(g(x))=h(f(x))`

`a=g(b)=f(a)`

`b=f(b)`

`c=h(b)`

`d=h(f(a))`

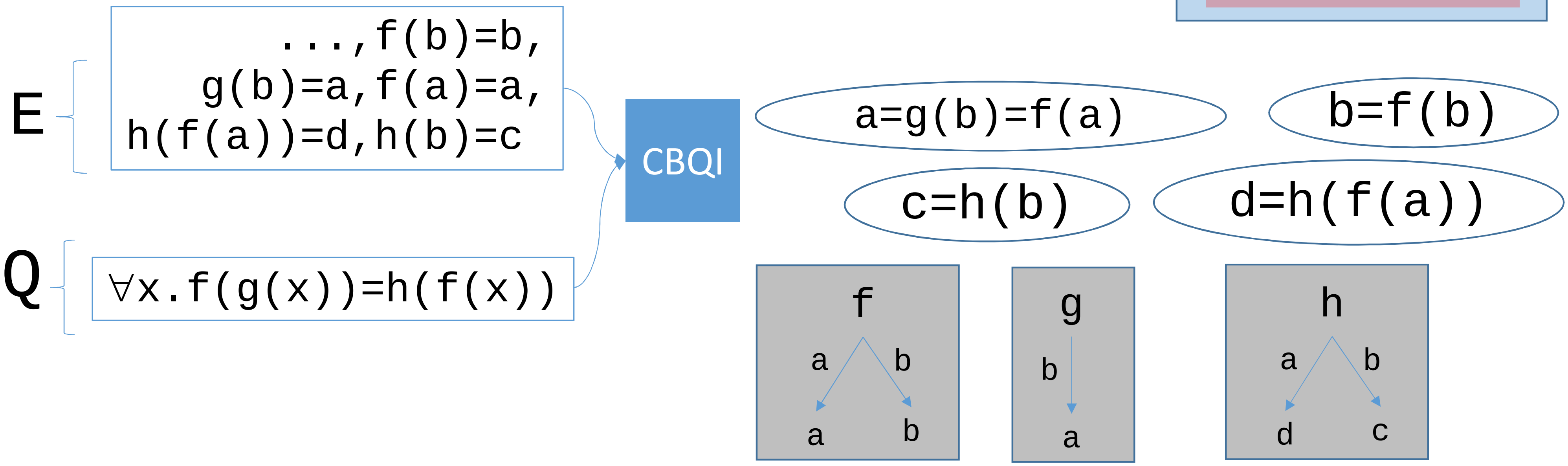
$E, f(g(b))=h(f(b)) \models_E f(g(b))=h(f(b))$ Check entailment

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



E

```
...,f(b)=b,
g(b)=a,f(a)=a,
h(f(a))=d,h(b)=c
```

Q

$\forall$x.f(g(x))=h(f(x))

CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))

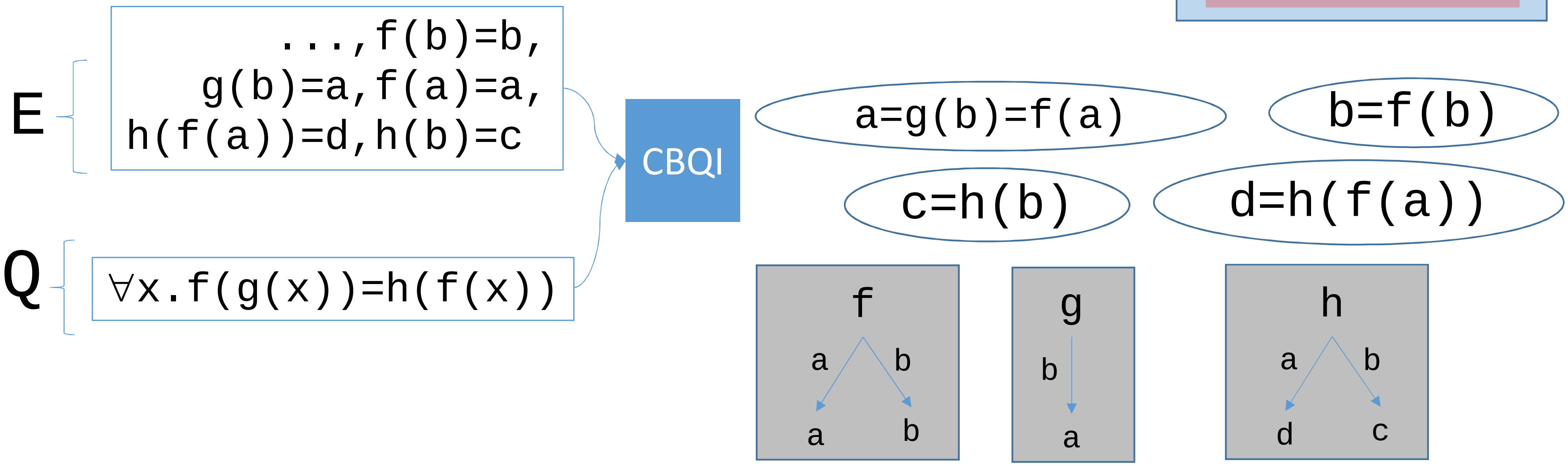
$$E, f(g(b))=h(f(b)) \models_E a=c$$

Conflict-Based Instantiation: EUF

Conflict-Based

E-matching

Model Based



E

```
    ...,f(b)=b,
  g(b)=a,f(a)=a,
h(f(a))=d,h(b)=c
```

Q

```
∀x.f(g(x))=h(f(x))
```

CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))

$\texttt{E,f(g(b))=h(f(b))} \models_E$ **a=c**

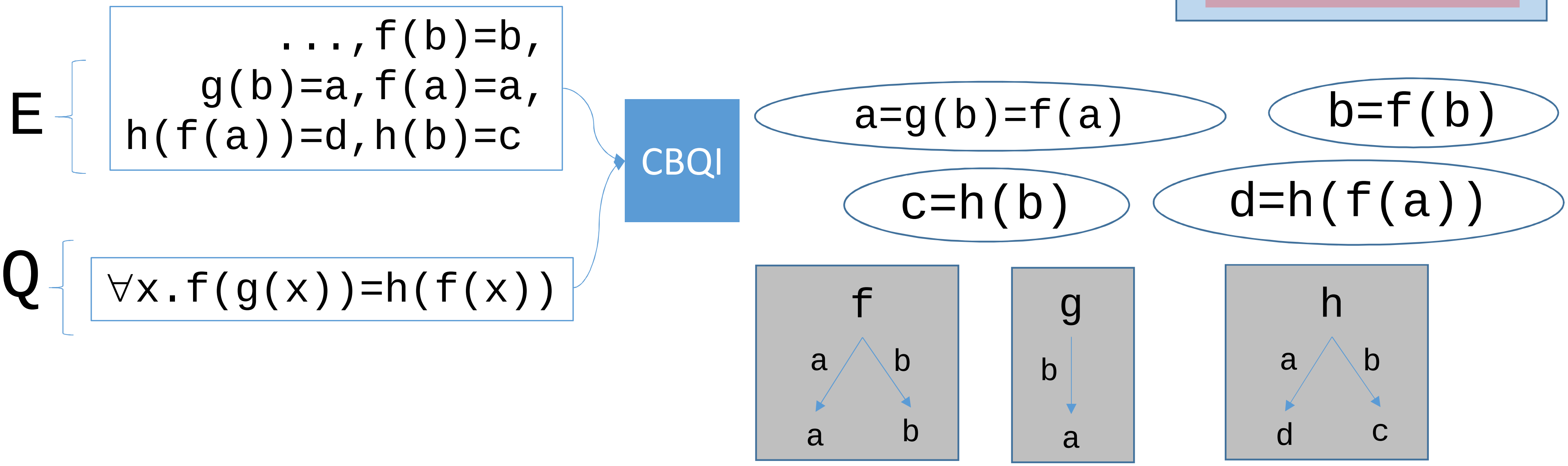
Instance is *not conflicting*, but *propagates* an equality between two existing terms in E

Conflict-Based Instantiation: EUF

Conflict-Based

E-matching

Model Based



E

...,f(b)=b,
g(b)=a,f(a)=a,
h(f(a))=d,h(b)=c

Q

∀x.f(g(x))=h(f(x))

CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

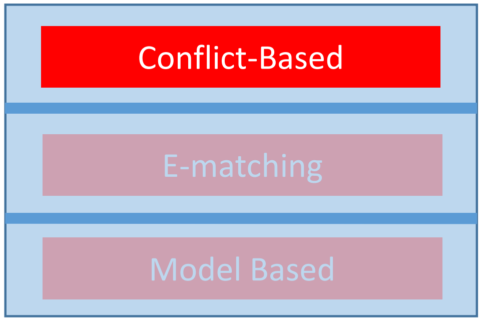
d=h(f(a))

E,f(g(b))=h(f(b)) $\models_E$ a=c

f(g(b))=h(f(b)) is a propagating instance for (E,Q)

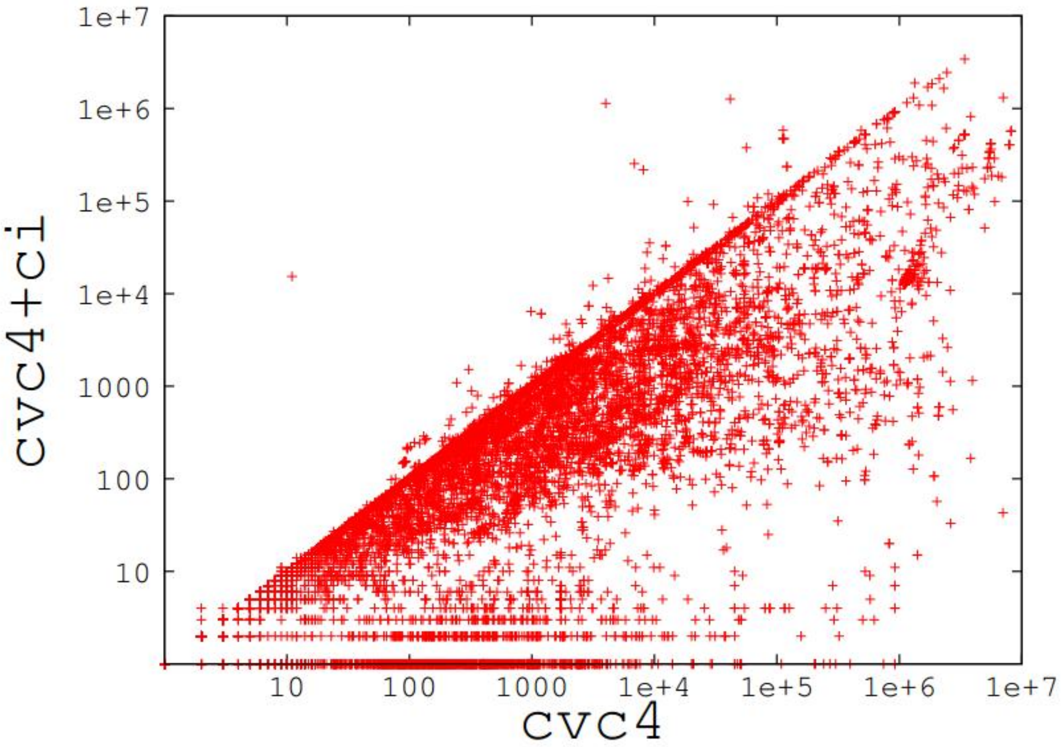
Ĝ *These are also useful*

Conflict-Based Instantiation: Impact


Conflict-Based

E-matching

Model Based

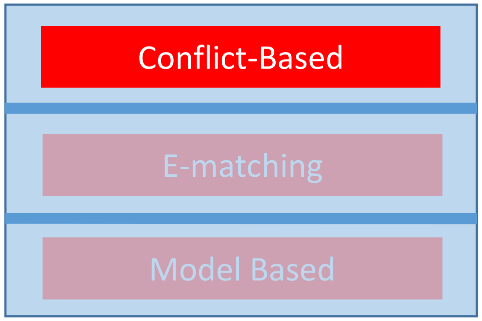


Reported number of instances.

- Using conflict-based instantiation (**`cvc4+ci`**), require an order of magnitude fewer instances for showing "UNSAT" wrt E-matching alone

(taken from **[Reynolds et al FMCAD14]**, evaluation On SMTLIB, TPTP, Isabelle benchmarks)

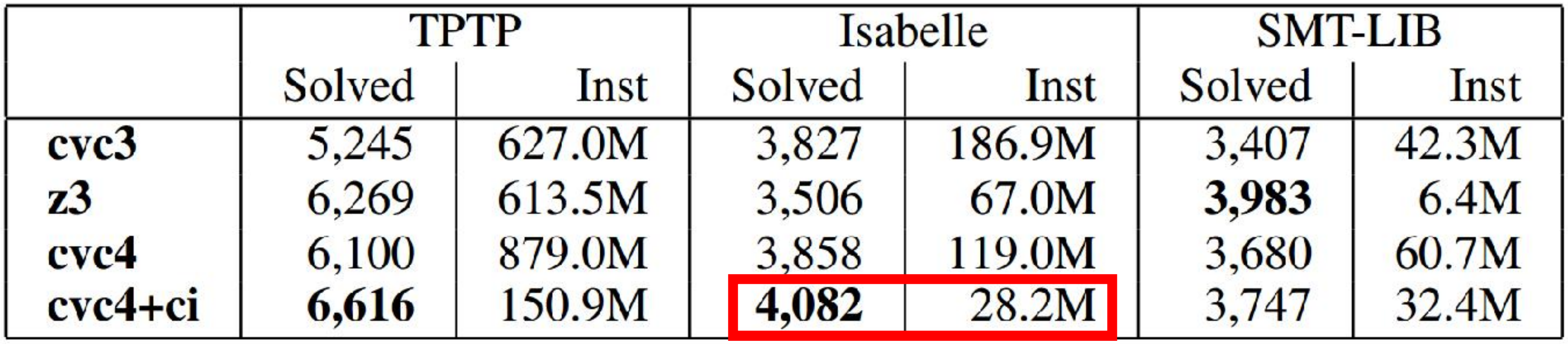
Conflict-Based Instantiation: Impact


Conflict-Based

E-matching

Model Based

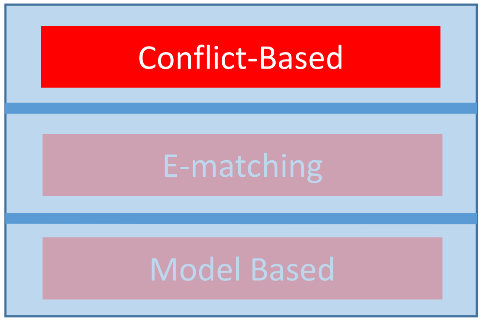
- CVC4 with conflicting instances **`cvc4+ci`**
  - Solves the most benchmarks for TPTP and Isabelle
  - Requires almost an order of magnitude fewer instantiations



	TPTP		Isabelle		SMT-LIB	
	Solved	Inst	Solved	Inst	Solved	Inst
cvc3	5,245	627.0M	3,827	186.9M	3,407	42.3M
z3	6,269	613.5M	3,506	67.0M	**3,983**	6.4M
cvc4	6,100	879.0M	3,858	119.0M	3,680	60.7M
cvc4+ci	**6,616**	150.9M	**4,082**	28.2M	3,747	32.4M

Ĝ *A number of hard benchmarks can be solved without resorting to E-matching at all*

Challenge : Finding Conflicting Instances



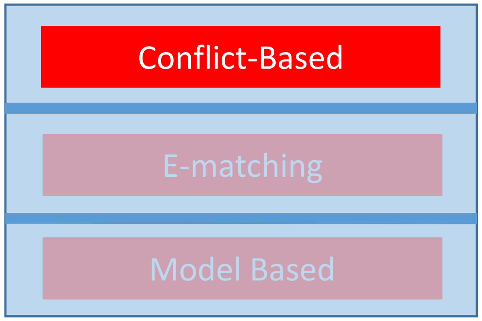
Conflict-Based

E-matching

Model Based

- How do we *find* conflicting instances?

Challenge : Finding Conflicting Instances



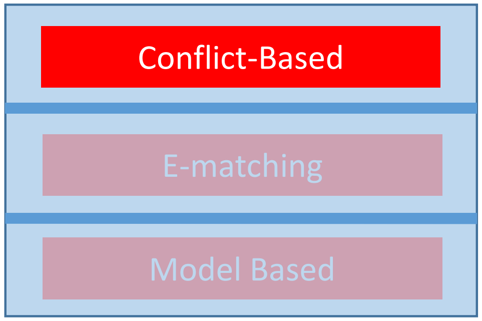
Conflict-Based

E-matching

Model Based

- How do we *find* conflicting instances?
  - Naively:
    1. Produce all instances Ψ_1, …, Ψ_n via E-matching for (E,Q)
    2. For `i=1,` …, n, check if Ψ_i is a conflicting instance for (E,Q)

Challenge : Finding Conflicting Instances


Conflict-Based

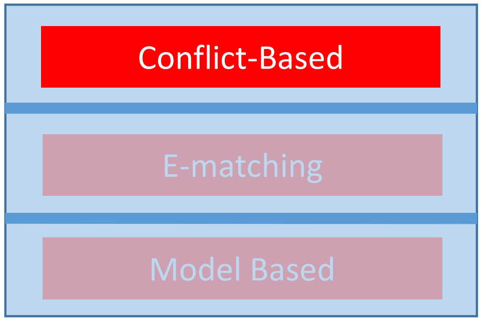
E-matching

Model Based

- How do we *find* conflicting instances?
  - Naively:
    1. Produce all instances Ψ_1, …, Ψ_n via E-matching for (E,Q)
    2. For `i=1,`…, n, check if Ψ_i is a conflicting instance for (E,Q)

    ⇒ *but n may be very large!*

Challenge : Finding Conflicting Instances


Conflict-Based

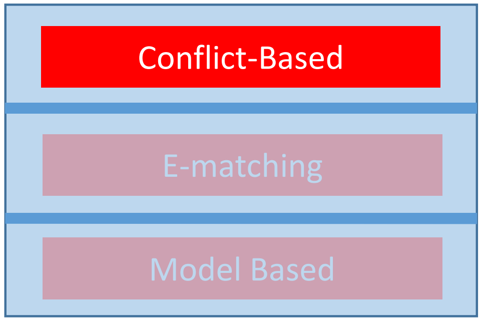
E-matching

Model Based

- How do we *find* conflicting instances?
  - Naively:
    1. Produce all instances $\Psi_1, \ldots, \Psi_n$ via E-matching for (E,Q)
    2. For $i=1,\ldots,n$, check if Ψ_i is a conflicting instance for (E,Q)
  - In practice: it can be done more efficiently:
    - Basic idea: construct instances via a stronger version of matching
      - Intuition: for $\forall x.P(x) \vee Q(x)$, will *only* match $P(x)$ with $P(t) \Leftrightarrow \bot$

        (For technical details, see **[Reynolds et al FMCAD2014]**)
    - Generalized to calculus based on E-(dis)unification **[Barbosa et al TACAS2017]**

Challenge : Theory Symbols

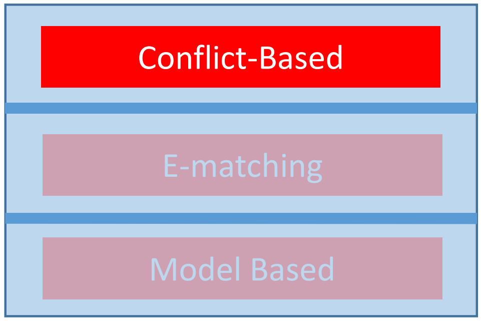


- What about quantified formulas that contain *theory symbols*?

E `f(1)=5`

Q `∀xy.f(x+y)>x+2*y`

Challenge : Theory Symbols


Conflict-Based

E-matching

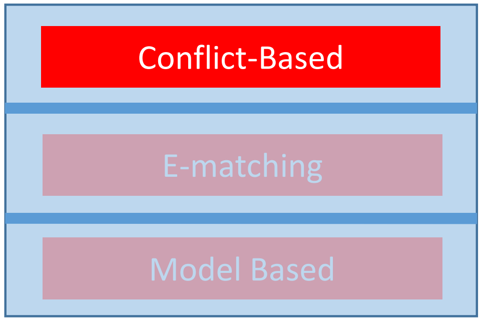
Model Based

- What about quantified formulas that contain *theory symbols*?

$$E \left\{ \boxed{\texttt{f(1)=5}} \right. \qquad Q \left\{ \boxed{\texttt{$\forall$xy.f(x+y)>x+2*y}} \right.$$

- Want to find, e.g.:
  - $\texttt{E,f(}\mathbf{-3}\texttt{+}\mathbf{4}\texttt{)>}\mathbf{-3}\texttt{+2*}\mathbf{4} \models_{UFLIA} \texttt{f(}\mathbf{-3}\texttt{+}\mathbf{4}\texttt{)>}\mathbf{-3}\texttt{+2*}\mathbf{4}$

Challenge : Theory Symbols

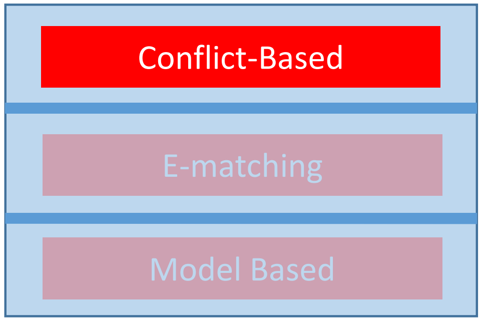


- What about quantified formulas that contain *theory symbols*?

$$E \left\{ \boxed{f(1)=5} \right. \qquad Q \left\{ \boxed{\forall xy.f(x+y)>x+2*y} \right.$$

- Want to find, e.g.:
  - $E, f(-3+4)>-3+2*4 \models_{UFLIA} f(1)>5$

Challenge : Theory Symbols


Conflict-Based

E-matching

Model Based

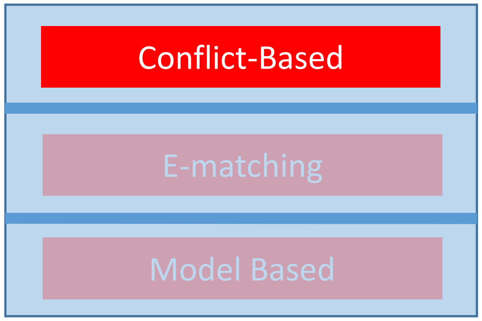
- What about quantified formulas that contain *theory symbols*?

E $\left\{ \boxed{\texttt{f(1)=5}} \right.$ Q $\left\{ \boxed{\forall \texttt{xy.f(x+y)>x+2*y}} \right.$

- Want to find, e.g.:
  - $\texttt{E,f(-3+4)>-3+2*4} \models_{\text{UFLIA}} \mathbf{5}\texttt{>5}$

By E, we know **`f(1)`**= **5**

Challenge : Theory Symbols



Conflict-Based

E-matching

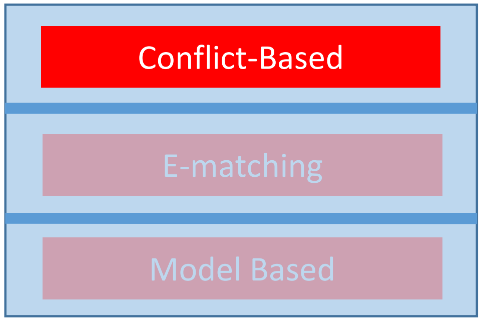
Model Based

- What about quantified formulas that contain *theory symbols*?

$$E \; \{ \; \boxed{\texttt{f(1)=5}} \qquad Q \; \{ \; \boxed{\forall\texttt{xy.f(x+y)>x+2*y}}$$

- Want to find, e.g.:
  - $\texttt{E,f(-3+4)>-3+2*4} \models_{\text{UFLIA}} \bot$

Challenge : Theory Symbols


Conflict-Based

E-matching

Model Based

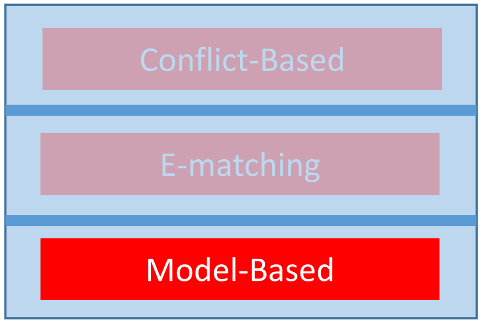
- What about quantified formulas that contain *theory symbols*?

E { `f(1)=5`   Q { `∀xy.f(x+y)>x+2*y`

- Want to find, e.g.:
  - `E,f(-3+4)>-3+2*4` $\models_{\text{UFLIA}} \bot$

Ĝ *In practice, finding such instances cannot be done efficiently*

Model-based Instantiation



Conflict-Based

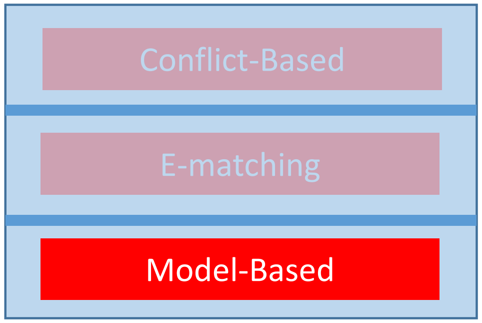
E-matching

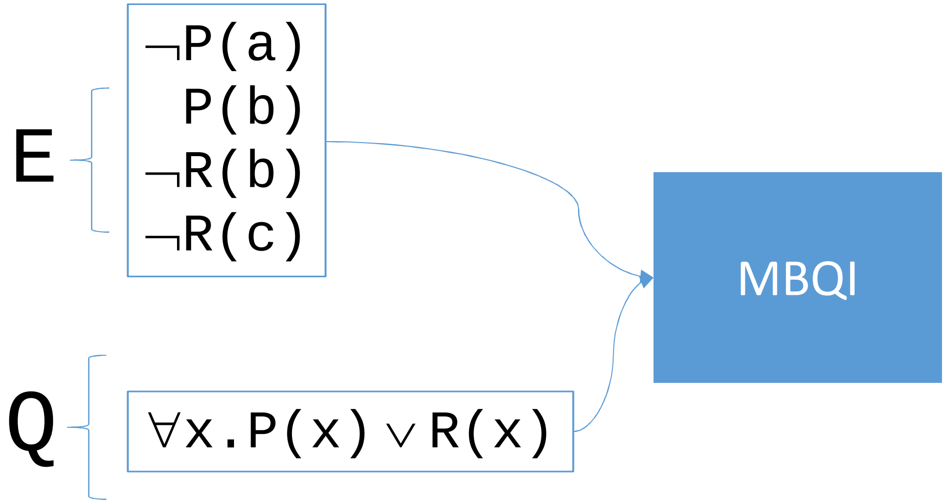
Model-Based

- Basic idea:
  - If E-matching saturates, build “candidate model” $\mathscr{M}$ satisfying E
    - Check if $\mathscr{M}$ also satisfies Q
      (using a quantifier-free satisfiability query)

Ĝ *Ability to answer* sat

Model-based Instantiation





E

$\neg P(a)$

$P(b)$

$\neg R(b)$

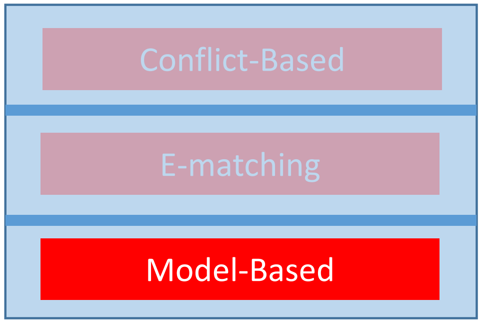
$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

MBQI

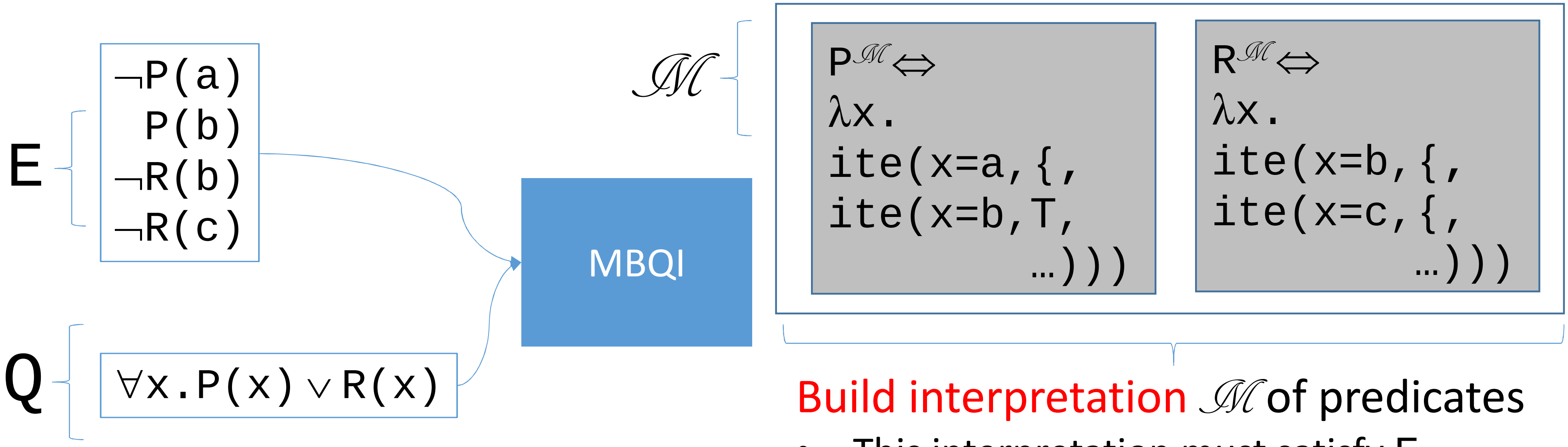
Model-based Instantiation



Conflict-Based

E-matching

Model-Based



E

$\neg P(a)$

$P(b)$

$\neg R(b)$

$\neg R(c)$

Q

$\forall x.P(x) \vee R(x)$

MBQI

$\mathscr{M}$

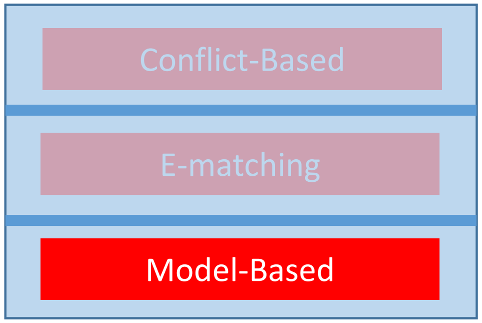
$P^{\mathscr{M}} \Leftrightarrow$
$\lambda x.$
ite(x=a,{,
ite(x=b,T,
…)))

$R^{\mathscr{M}} \Leftrightarrow$
$\lambda x.$
ite(x=b,{,
ite(x=c,{,
…)))

Build interpretation $\mathscr{M}$ of predicates

- This interpretation must satisfy E

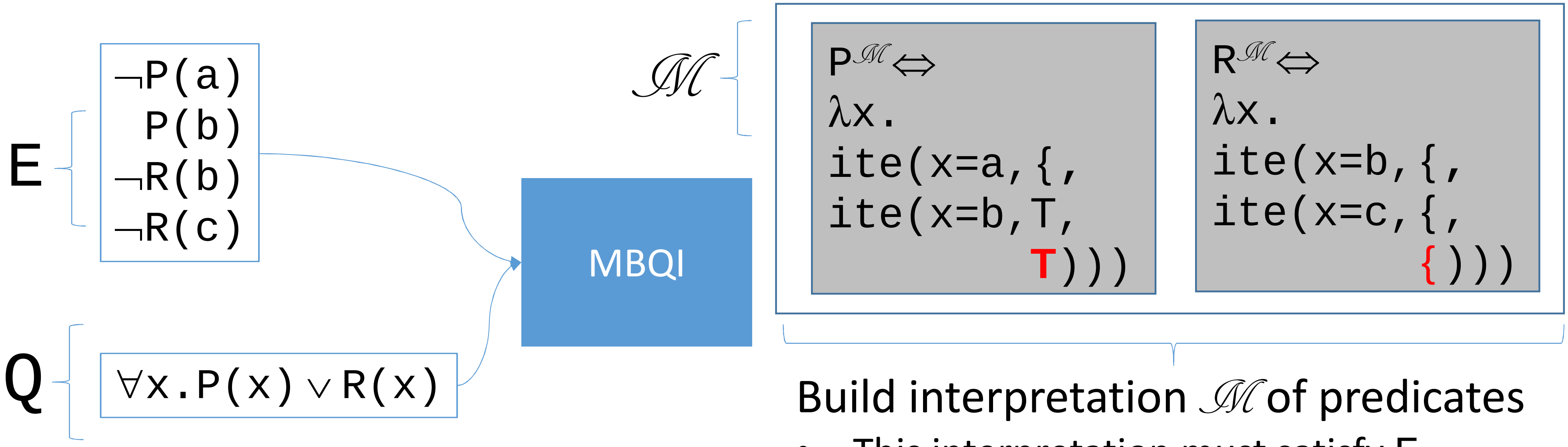
Model-based Instantiation



Conflict-Based

E-matching

Model-Based



E

¬P(a)
 P(b)
¬R(b)
¬R(c)

Q

∀x.P(x) ∨ R(x)

MBQI

$\mathscr{M}$

$P^{\mathscr{M}}$ ⇔
λx.
ite(x=a,{,
ite(x=b,T,
T)))

$R^{\mathscr{M}}$ ⇔
λx.
ite(x=b,{,
ite(x=c,{,
{)))

Build interpretation $\mathscr{M}$ of predicates

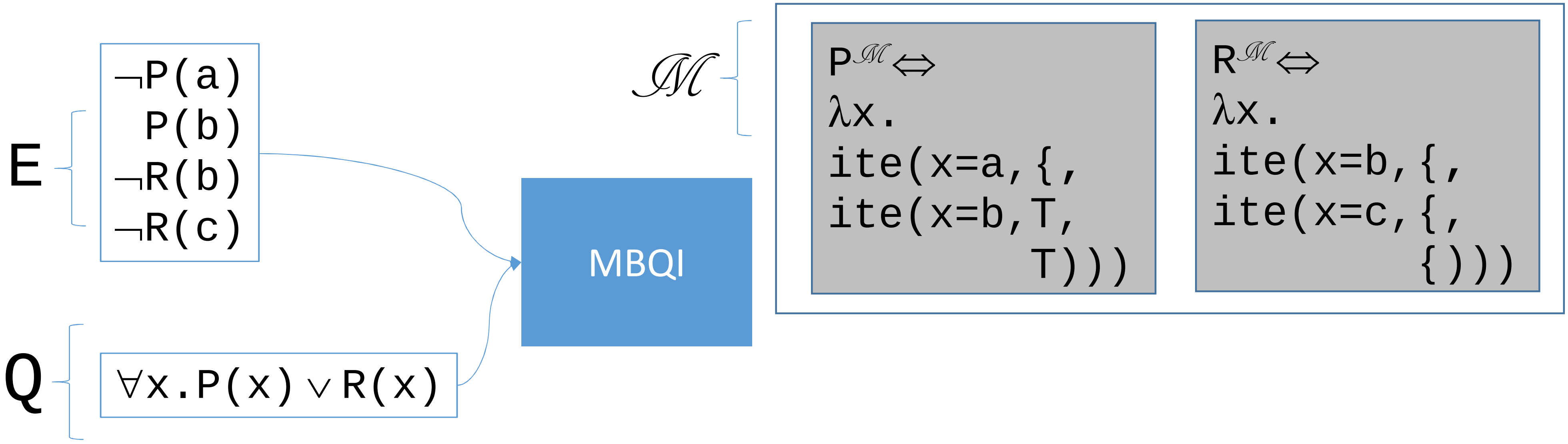
- This interpretation must satisfy E
- Missing values may be filled in arbitrarily

Model-based Instantiation

Conflict-Based

E-matching

Model-Based



E

$\neg$P(a)
P(b)
$\neg$R(b)
$\neg$R(c)

Q

$\forall$x.P(x) $\vee$ R(x)

MBQI

$\mathscr{M}$

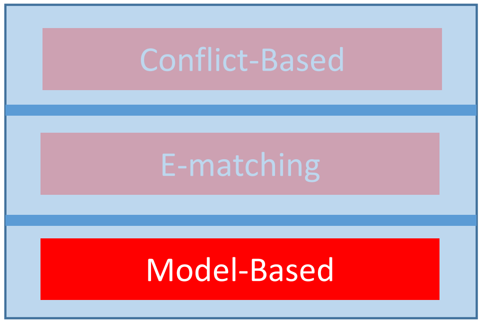
P$^{\mathscr{M}}$ $\Leftrightarrow$
λx.
ite(x=a,{,
ite(x=b,T,
T)))

R$^{\mathscr{M}}$ $\Leftrightarrow$
λx.
ite(x=b,{,
ite(x=c,{,
{)))

$\Rightarrow$ Does $\mathscr{M}$ satisfy Q?

- Check (un)satisfiability of: $\exists$x.$\neg$(P$^{\mathscr{M}}$(x) $\vee$ R$^{\mathscr{M}}$(x))

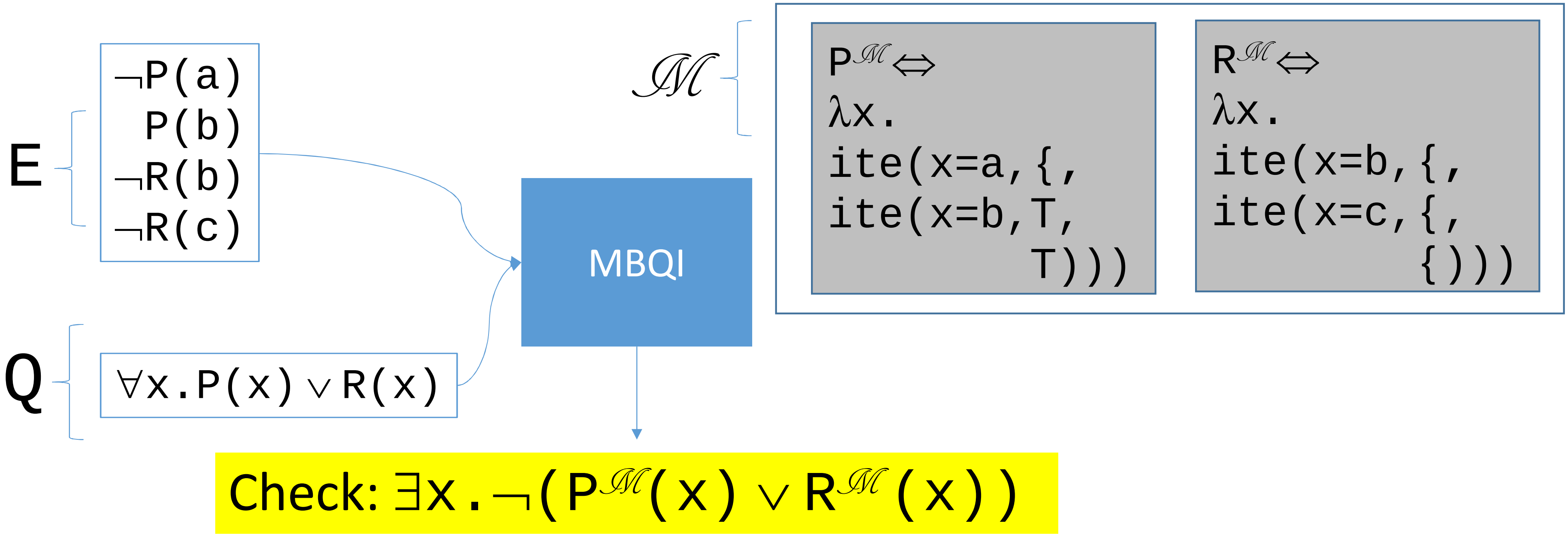
Model-based Instantiation



Conflict-Based

E-matching

Model-Based



E

$\neg P(a)$

$P(b)$

$\neg R(b)$

$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

MBQI

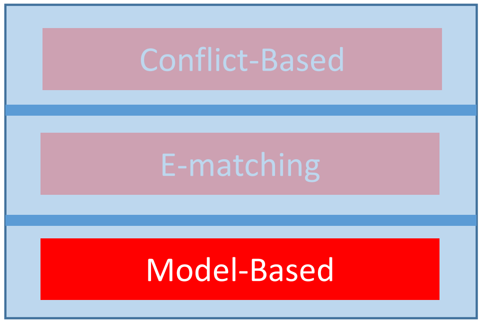
$\mathscr{M}$

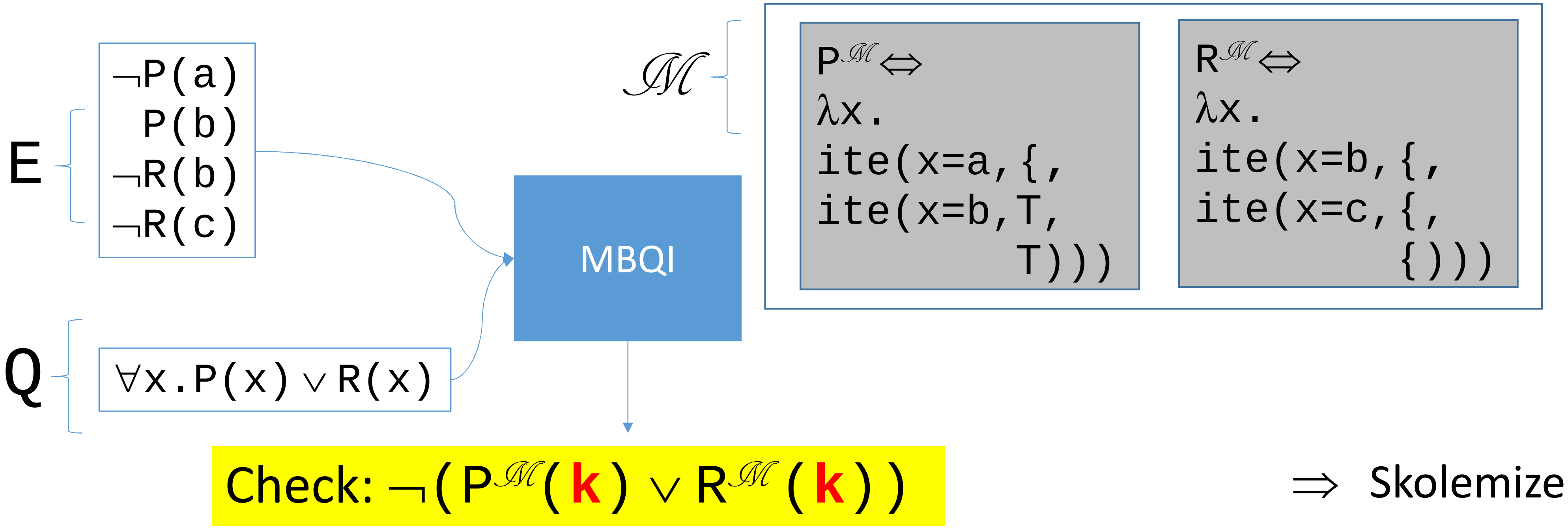
$P^{\mathscr{M}} \Leftrightarrow$
$\lambda x.$
ite(x=a,{,
ite(x=b,T,
T)))

$R^{\mathscr{M}} \Leftrightarrow$
$\lambda x.$
ite(x=b,{,
ite(x=c,{,
{)))

Check: $\exists x. \neg(P^{\mathscr{M}}(x) \vee R^{\mathscr{M}}(x))$

Model-based Instantiation

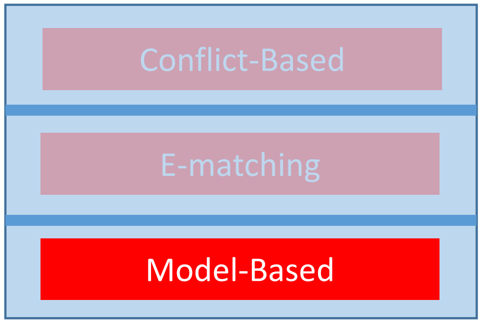




Check: $\neg(P^{\mathscr{M}}(k) \vee R^{\mathscr{M}}(k))$

⇒ Skolemize

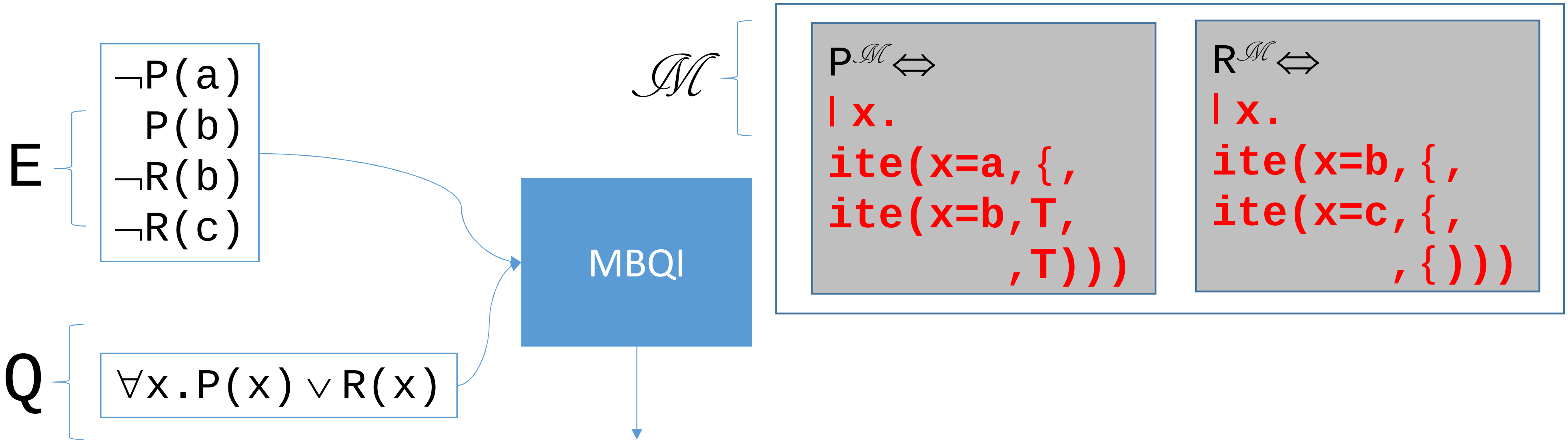
Model-based Instantiation



Conflict-Based

E-matching

Model-Based



E

$\neg$P(a)

P(b)

$\neg$R(b)

$\neg$R(c)

Q

$\forall$x.P(x)$\vee$R(x)

MBQI

$\mathscr{M}$

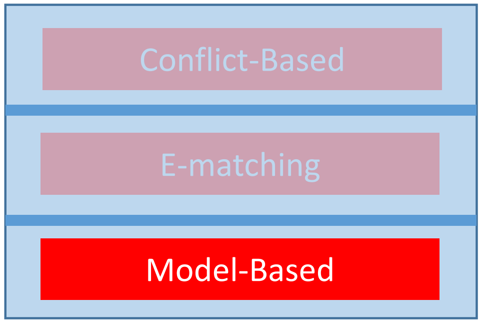
$P^{\mathscr{M}} \Leftrightarrow$
l x.
ite(x=a,{,
ite(x=b,T,
,T)))

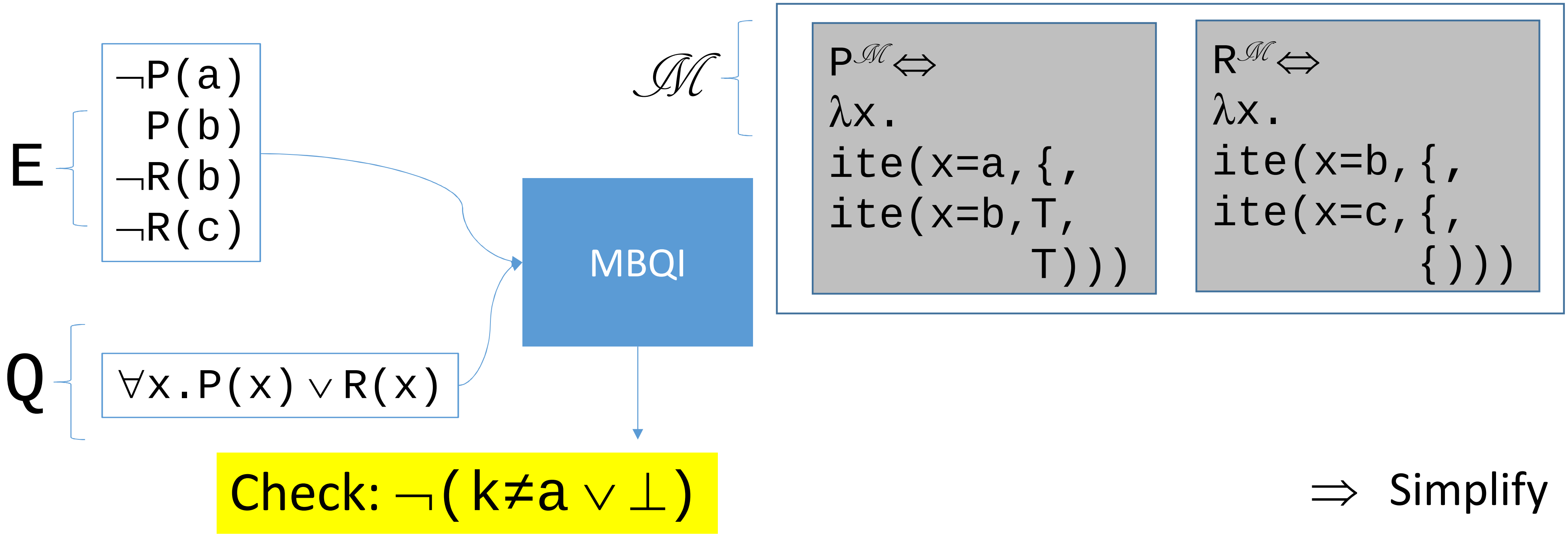
$R^{\mathscr{M}} \Leftrightarrow$
l x.
ite(x=b,{,
ite(x=c,{,
,{)))

Check: $\neg($ ite(k=a,{,ite(k=b,T,T)))$\vee$
ite(k=b,{,ite(k=c,{,{))) $)$

$\Rightarrow$ Substitute

Model-based Instantiation





E

¬P(a)
P(b)
¬R(b)
¬R(c)

Q

∀x.P(x)∨R(x)

MBQI

$\mathscr{M}$

$P^{\mathscr{M}} \Leftrightarrow$

```
λx.
ite(x=a,{,
ite(x=b,T,
         T)))
```

$R^{\mathscr{M}} \Leftrightarrow$

```
λx.
ite(x=b,{,
ite(x=c,{,
         {)))
```

Check: ¬(k≠a ∨ ⊥)

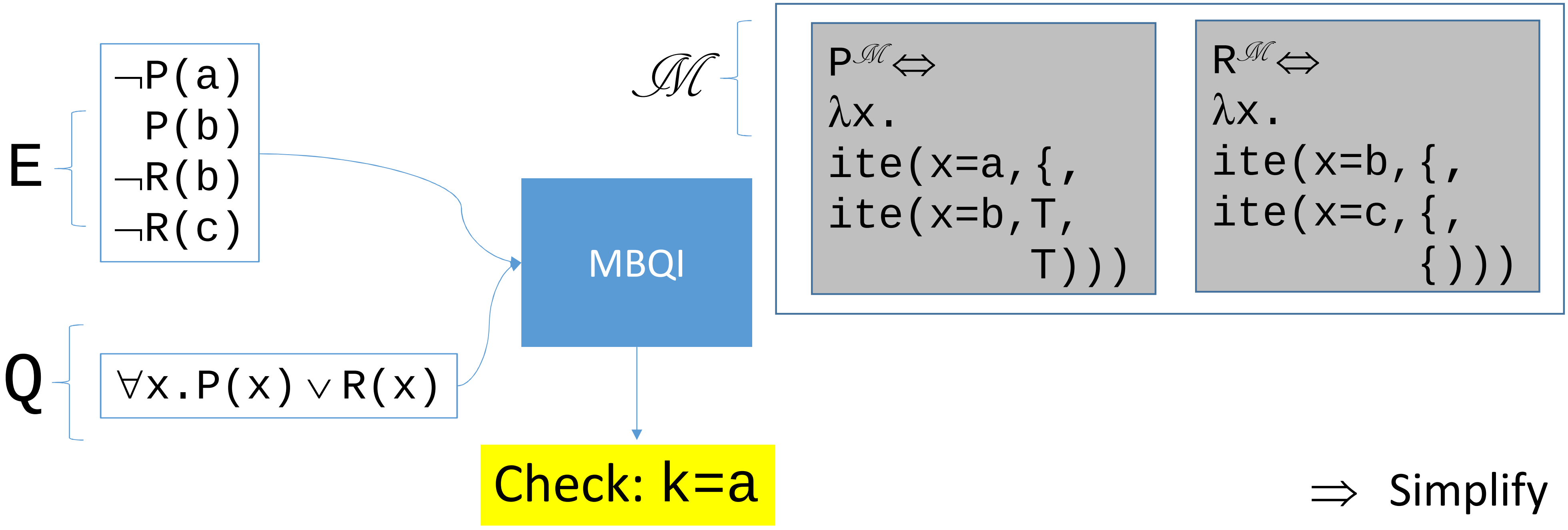
⇒ Simplify

Conflict-Based

E-matching

Model-Based

Model-based Instantiation



E

¬P(a)
 P(b)
¬R(b)
¬R(c)

Q

∀x.P(x) ∨ R(x)

MBQI

Check: k=a

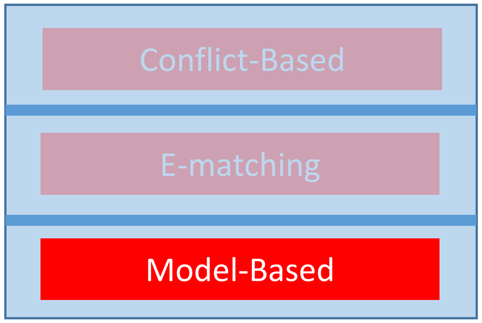
$\mathscr{M}$

$P^{\mathscr{M}}$ ⇔
λx.
ite(x=a,{,
ite(x=b,T,
T)))

$R^{\mathscr{M}}$ ⇔
λx.
ite(x=b,{,
ite(x=c,{,
{)))

⇒ Simplify

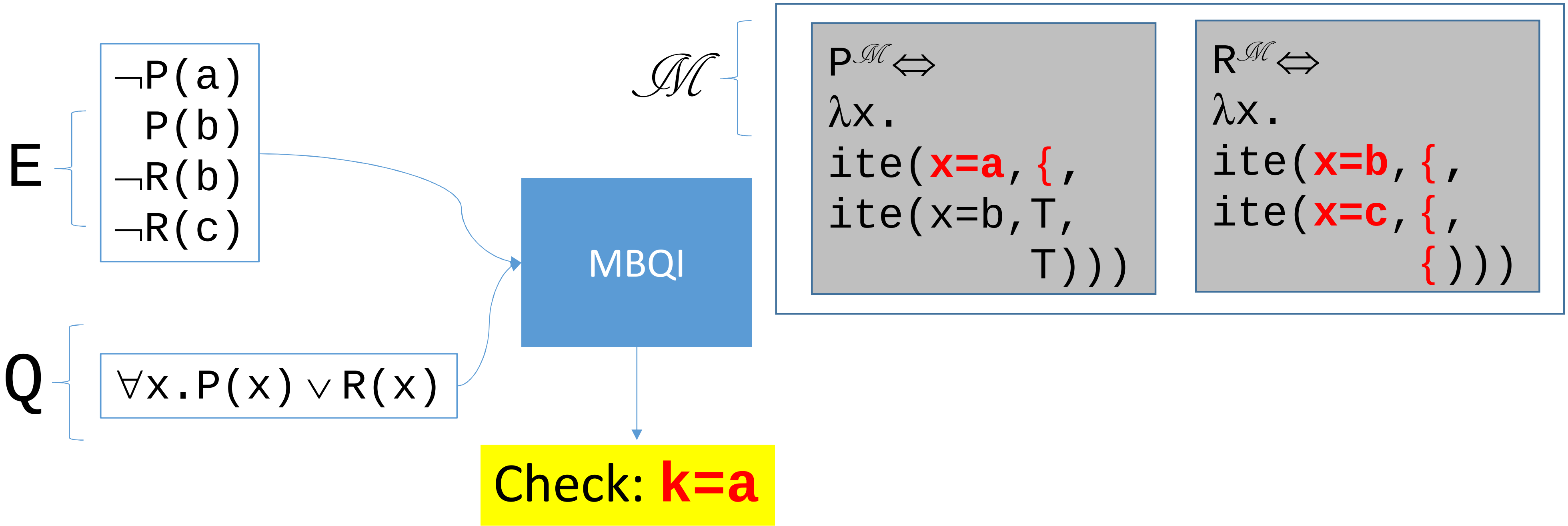
Model-based Instantiation



Conflict-Based

E-matching

Model-Based



E

¬P(a)
 P(b)
¬R(b)
¬R(c)

Q

∀x.P(x) ∨ R(x)

MBQI

$\mathscr{M}$

$P^{\mathscr{M}}$ ⇔
λx.
ite(**x=a**, {,
ite(x=b, T,
          T)))

$R^{\mathscr{M}}$ ⇔
λx.
ite(**x=b**, {,
ite(**x=c**, {,
          {)))

Check: **k=a**

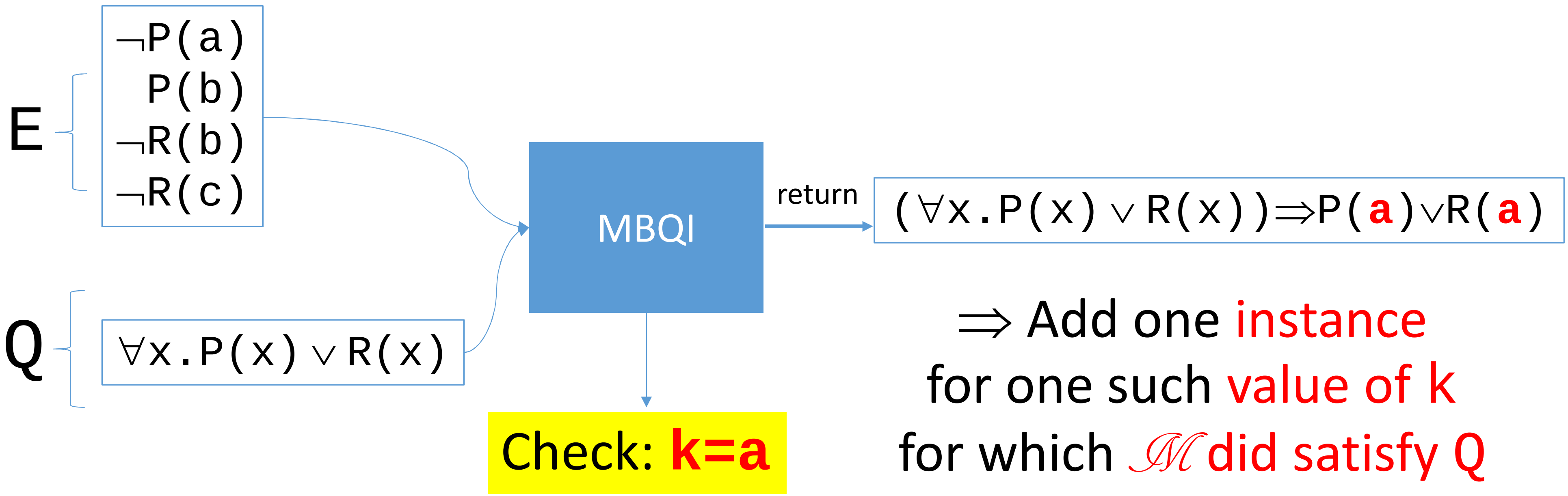
⇒ *Satisfiable! There are values k for which $\mathscr{M}$ does not satisfy Q*

Model-based Instantiation

Conflict-Based

E-matching

Model-Based



E

$\neg P(a)$

$P(b)$

$\neg R(b)$

$\neg R(c)$

Q

$\forall x. P(x) \vee R(x)$

MBQI

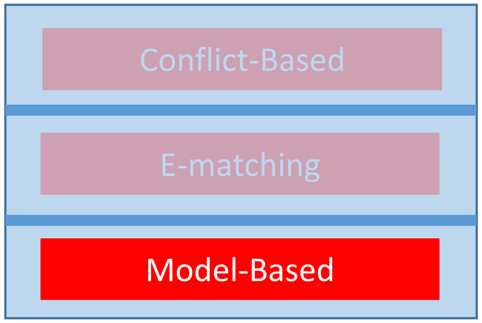
return

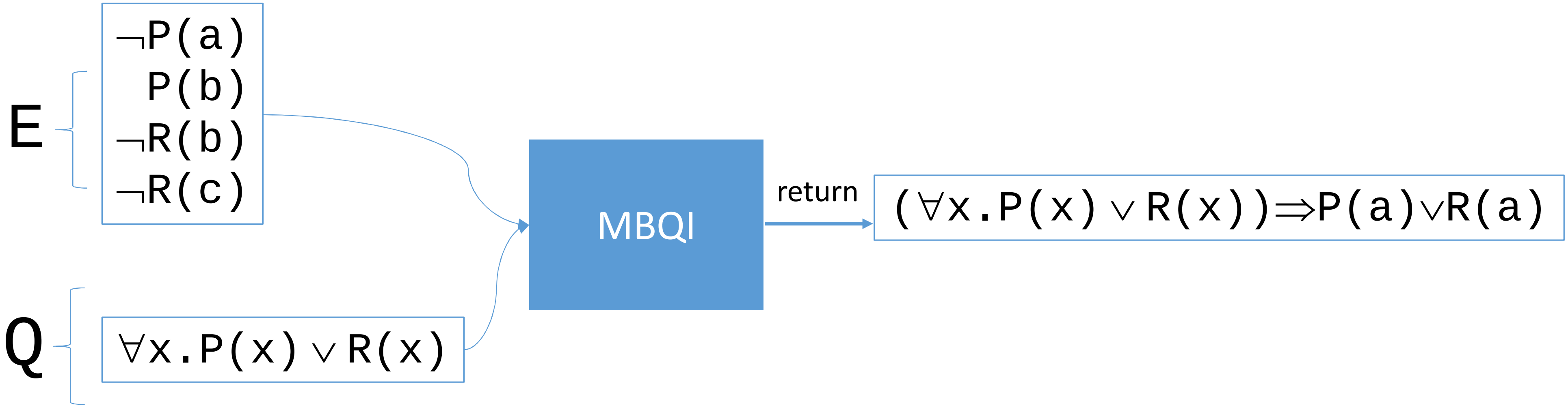
$(\forall x. P(x) \vee R(x)) \Rightarrow P(\mathbf{a}) \vee R(\mathbf{a})$

Check: **k=a**

⇒ Add one instance
for one such value of k
for which $\mathscr{M}$ did satisfy Q

Model-based Instantiation





$\neg P(a)$

$P(b)$

$\neg R(b)$

$\neg R(c)$

$\forall x. P(x) \vee R(x)$

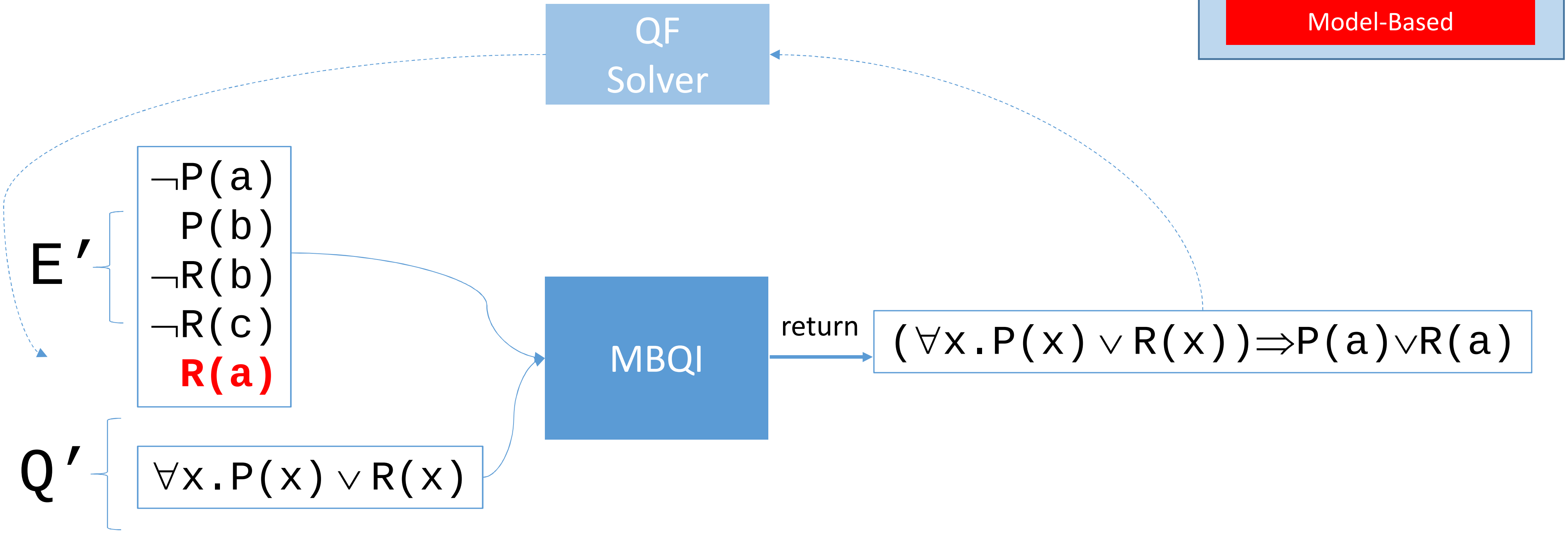
$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$

Conflict-Based

E-matching

Model-Based

Model-based Instantiation



QF Solver

E'

$\neg P(a)$
$P(b)$
$\neg R(b)$
$\neg R(c)$
$\mathbf{R(a)}$

Q'

$\forall x. P(x) \vee R(x)$

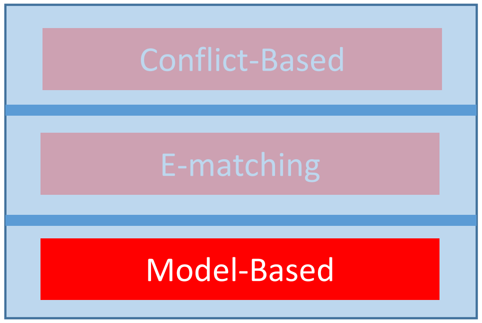
MBQI

return

$(\forall x. P(x) \vee R(x)) \Rightarrow P(a) \vee R(a)$

$\Rightarrow$ Subsequent models must satisfy $P(a) \vee \mathbf{R(a)}$

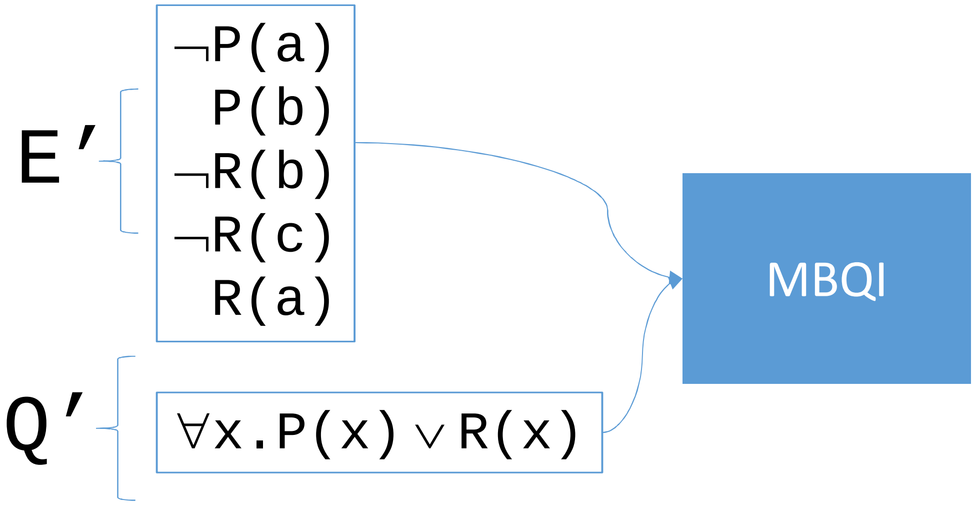
Model-based Instantiation



Conflict-Based

E-matching

Model-Based



E′

$\neg P(a)$

$P(b)$

$\neg R(b)$

$\neg R(c)$

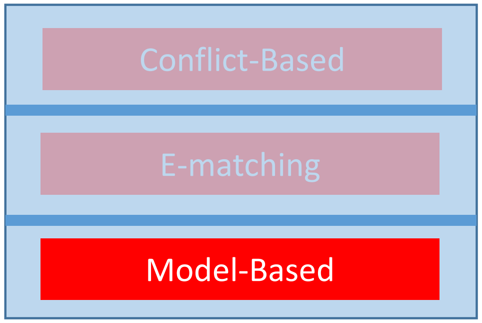
$R(a)$

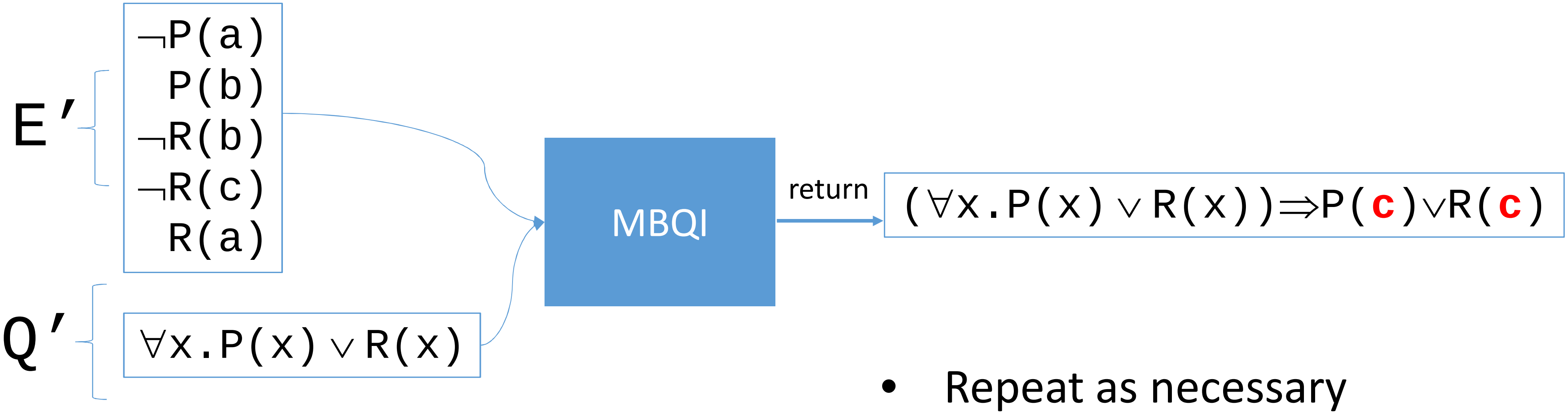
Q′

$\forall x. P(x) \vee R(x)$

MBQI

Model-based Instantiation





$\neg P(a)$

$P(b)$

$\neg R(b)$

$\neg R(c)$

$R(a)$

$\forall x. P(x) \vee R(x)$

$(\forall x. P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$

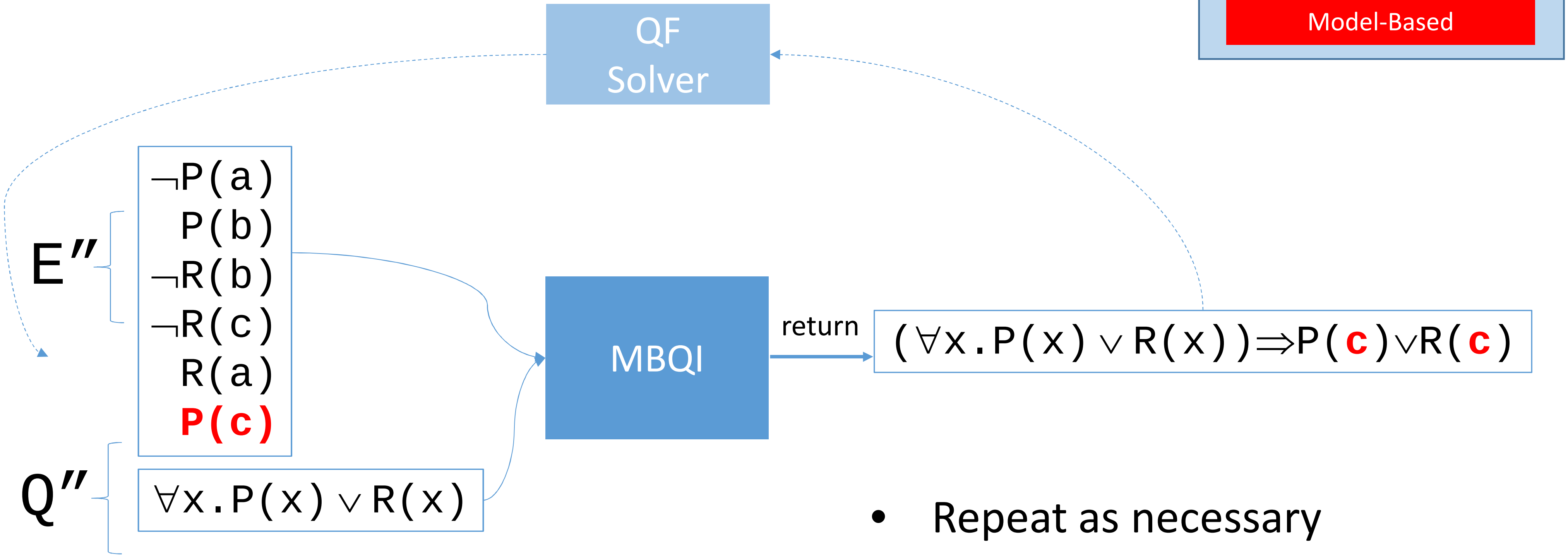
- Repeat as necessary
  ⇒ "Model refinement loop"

Conflict-Based

E-matching

Model-Based

Model-based Instantiation



QF Solver

E″

$\neg P(a)$

$P(b)$

$\neg R(b)$

$\neg R(c)$

$R(a)$

$P(c)$

Q″

$\forall x.P(x) \vee R(x)$

MBQI

return

$(\forall x.P(x) \vee R(x)) \Rightarrow P(c) \vee R(c)$

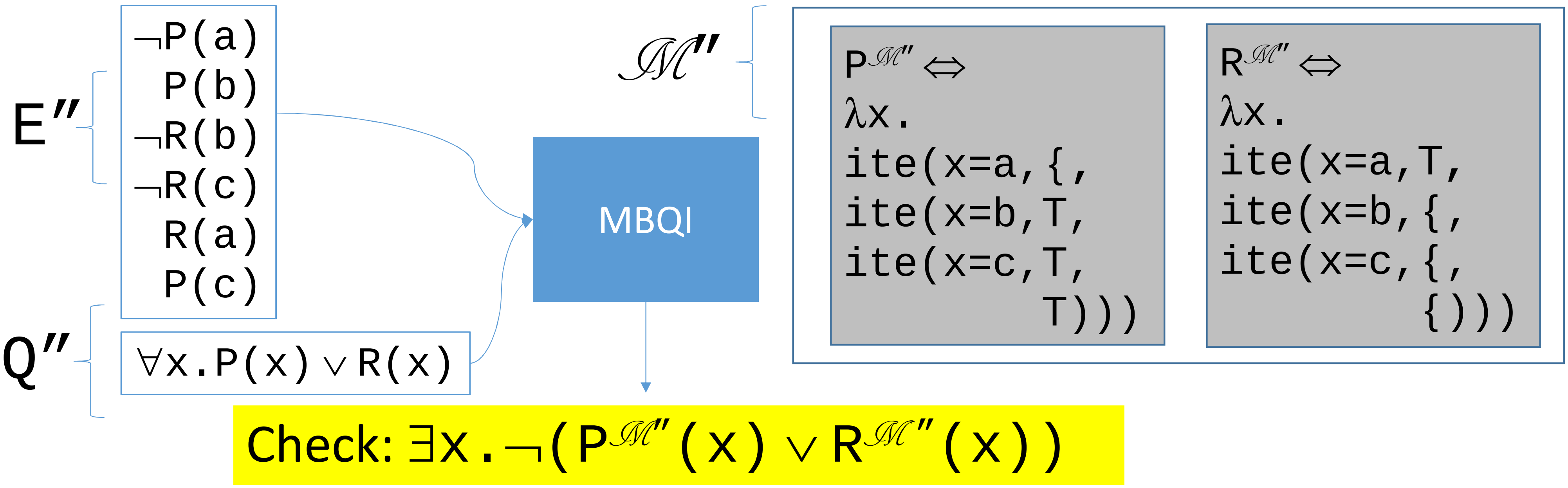
- Repeat as necessary
  ⇒ “Model refinement loop”

Model-based Instantiation

Conflict-Based

E-matching

Model-Based



E''

$\neg$P(a)
P(b)
$\neg$R(b)
$\neg$R(c)
R(a)
P(c)

Q''

$\forall$x.P(x) $\vee$ R(x)

MBQI

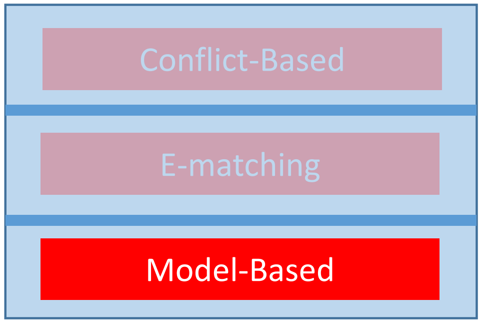
$\mathscr{M}''$

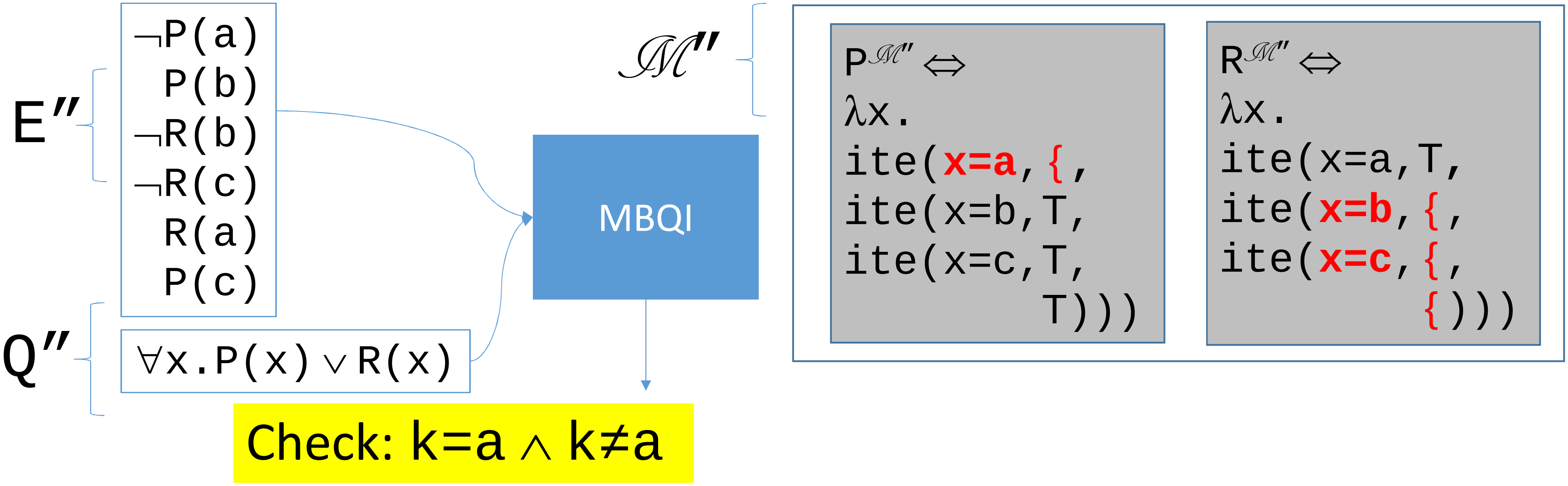
$P^{\mathscr{M}''} \Leftrightarrow$
λx.
ite(x=a,{,
ite(x=b,T,
ite(x=c,T,
T)))

$R^{\mathscr{M}''} \Leftrightarrow$
λx.
ite(x=a,T,
ite(x=b,{,
ite(x=c,{,
{)))

Check: $\exists x.\neg(P^{\mathscr{M}''}(x) \vee R^{\mathscr{M}''}(x))$

Model-based Instantiation




E″

¬P(a)
P(b)
¬R(b)
¬R(c)
R(a)
P(c)

Q″

∀x.P(x) ∨ R(x)

MBQI

Check: k=a ∧ k≠a

$\mathscr{M}''$

```
P^M″ ⇔
λx.
ite(x=a,{,
ite(x=b,T,
ite(x=c,T,
         T)))
```

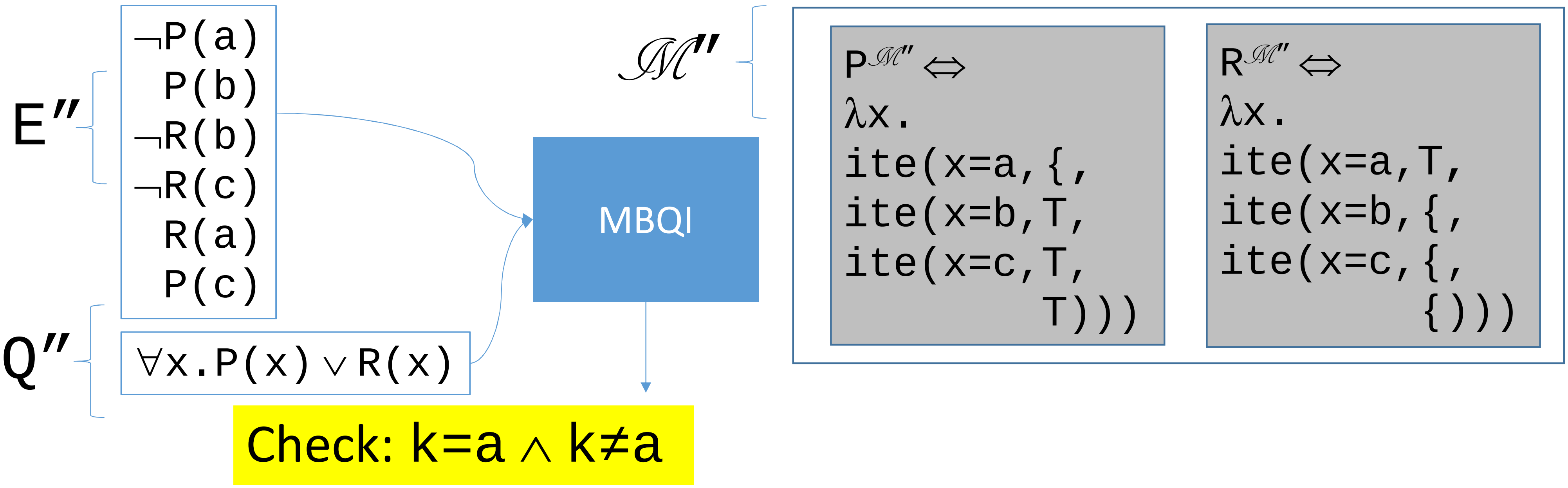
```
R^M″ ⇔
λx.
ite(x=a,T,
ite(x=b,{,
ite(x=c,{,
         {)))
```

Model-based Instantiation

Conflict-Based

E-matching

Model-Based



E″

$\neg$P(a)

P(b)

$\neg$R(b)

$\neg$R(c)

R(a)

P(c)

Q″

$\forall$x.P(x) $\vee$ R(x)

MBQI

$\mathscr{M}''$

$P^{\mathscr{M}''} \Leftrightarrow$
λx.
ite(x=a,{,
ite(x=b,T,
ite(x=c,T,
T)))

$R^{\mathscr{M}''} \Leftrightarrow$
λx.
ite(x=a,T,
ite(x=b,{,
ite(x=c,{,
{)))

Check: k=a $\wedge$ k≠a

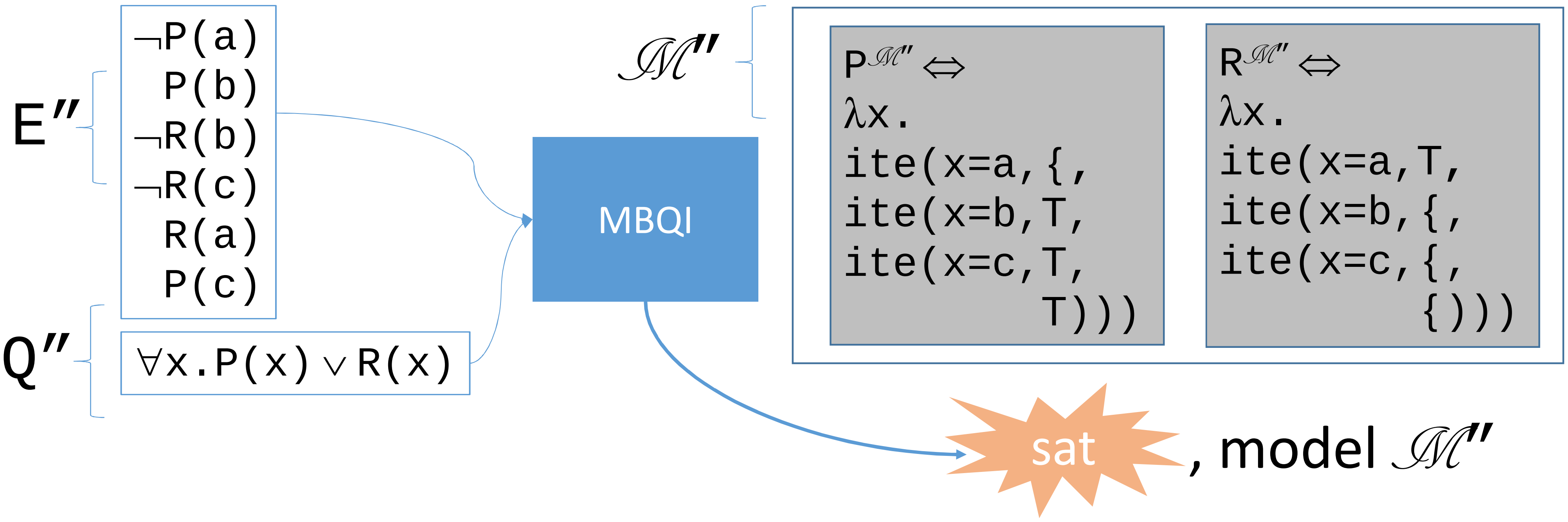
$\Rightarrow$ Unsatisfiable, there are no values k for which $\mathscr{M}''$ does not satisfy Q

Model-based Instantiation

Conflict-Based

E-matching

Model-Based



E″

¬P(a)
P(b)
¬R(b)
¬R(c)
R(a)
P(c)

Q″

∀x.P(x) ∨ R(x)

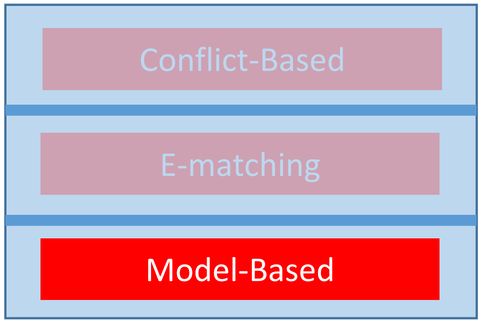
MBQI

$\mathscr{M}''$

P$^{\mathscr{M}''}$ ⇔
λx.
ite(x=a,{,
ite(x=b,T,
ite(x=c,T,
T)))

R$^{\mathscr{M}''}$ ⇔
λx.
ite(x=a,T,
ite(x=b,{,
ite(x=c,{,
{)))

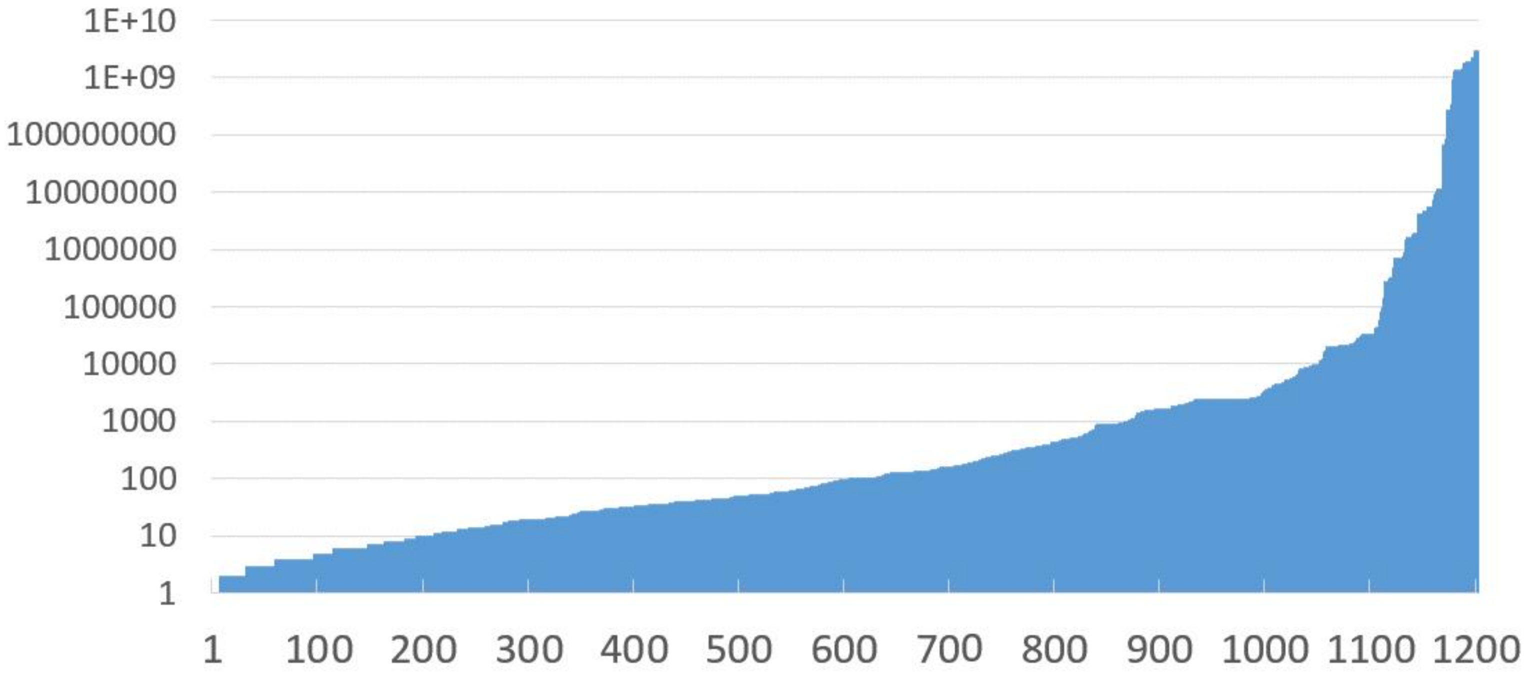
sat, model $\mathscr{M}''$


Conflict-Based

E-matching

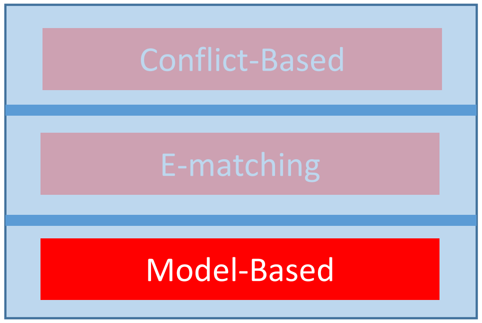
Model-Based

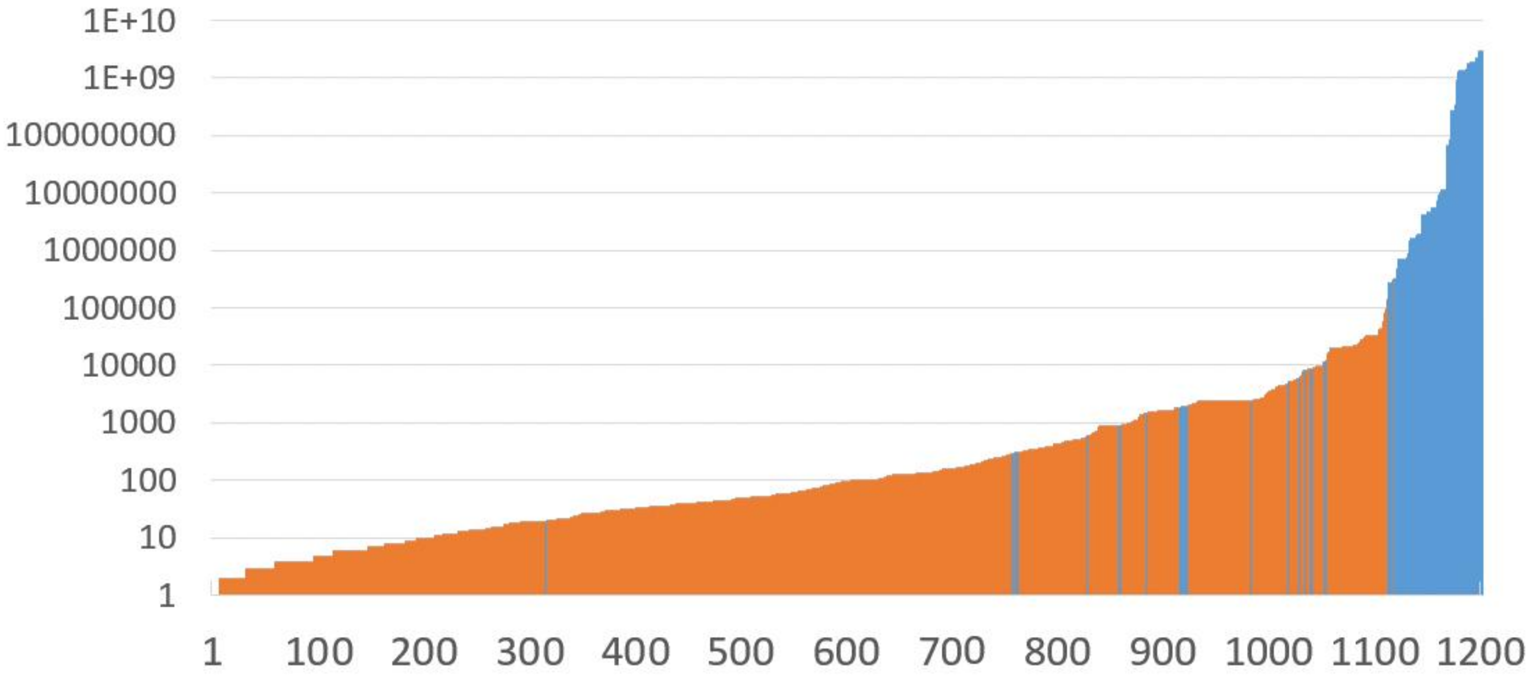
Model-based Instantiation: Impact



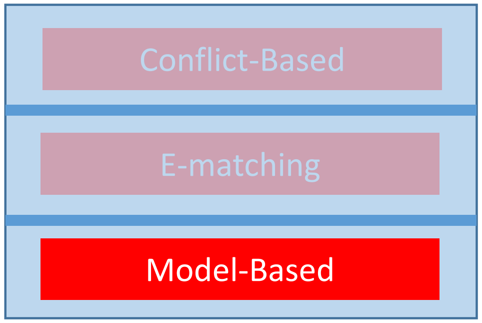
- 1203 satisfiable benchmarks from the TPTP library
  - Graph shows # instances required by exhaustive instantiation
    - E.g. $\forall$`xyz:U.P(x,y,z)`, if `|U|`=4, requires 4^3=64 instances

Model-based Instantiation: Impact





- CVC4 Finite Model Finding + Exhaustive instantiation
  - Scales only up to ~150k instances with a 30 sec timeout

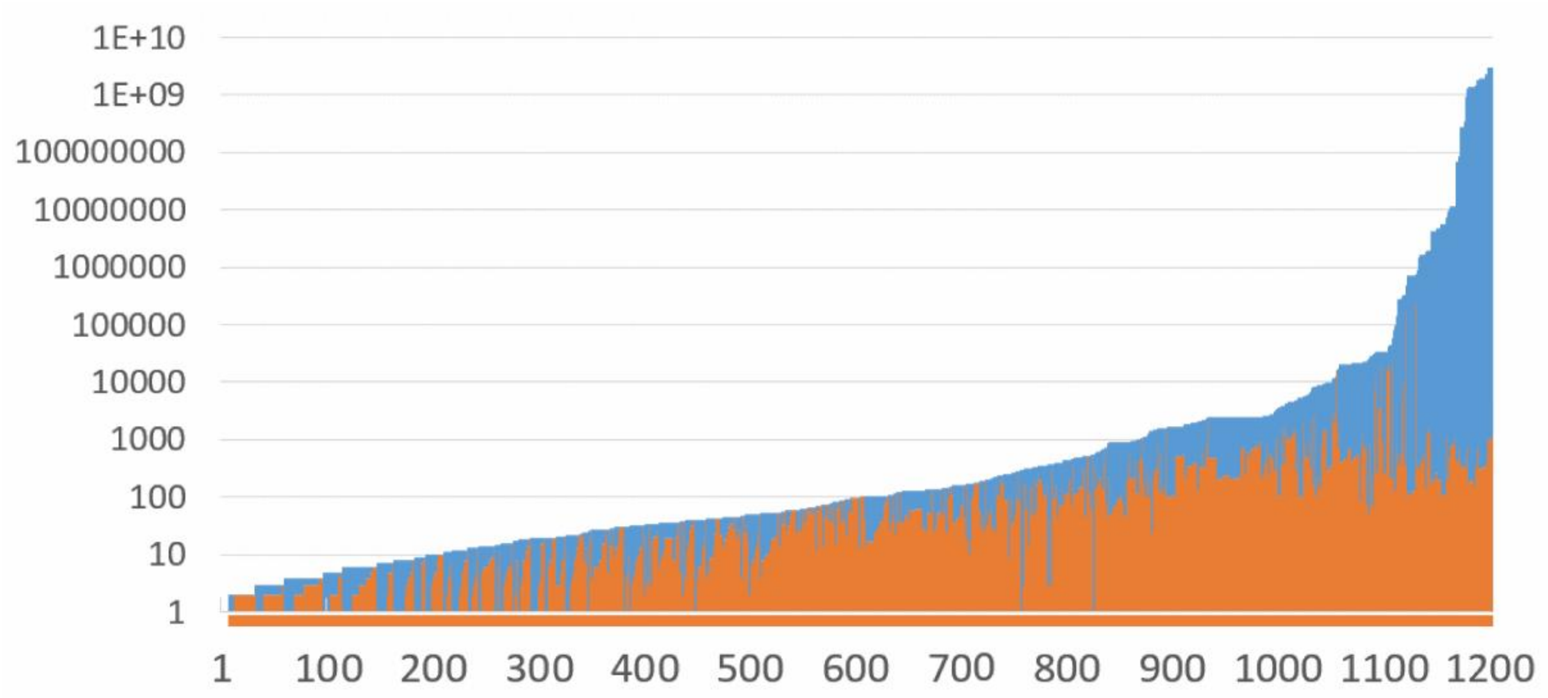


Conflict-Based

E-matching

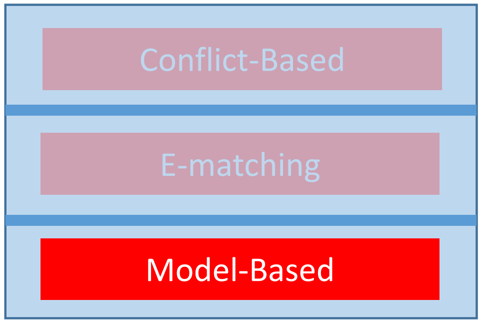
Model-Based

Model-based Instantiation: Impact



- CVC4 Finite Model Finding + Model-Based instantiation **[Reynolds et al CADE13]**
  - Scales to >2 billion instances with a 30 sec timeout, only adds fraction of possible instances

Challenge : Building Interpretations



- How do we build interpretations $\mathscr{M}$?
  - Typically, build interpretations $\mathtt{f}^{\mathscr{M}}$ that are almost constant:
    - e.g. $\mathtt{f}^{\mathscr{M}} := \lambda\mathtt{x.ite(x=t}_1\mathtt{,v}_1\mathtt{,ite(x=t}_2\mathtt{,v}_2\mathtt{,}\ldots\mathtt{,ite(x=t}_n\mathtt{,v}_n\mathtt{,v}_{\mathtt{def}}\mathtt{)}\ldots\mathtt{))}$

Challenge : Building Interpretations

Conflict-Based

E-matching

Model-Based

- How do we build interpretations $\mathscr{M}$?
  - Typically, build interpretations $\mathtt{f}^{\mathscr{M}}$ that are almost constant:
    - e.g. $\mathtt{f}^{\mathscr{M}} := \lambda\mathtt{x.ite(x=t_1,v_1,ite(x=t_2,v_2,\ldots,ite(x=t_n,v_n,v_{def})\ldots))}$

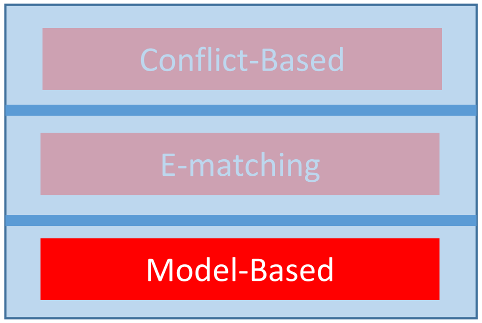
…but models may need to be more complex when *theories are present*:

$\forall\mathtt{xy{:}Int.(f(x,y)\geq x \wedge f(x,y)\geq y)}$ → $\mathtt{f}^{\mathscr{M}} := \lambda\mathtt{xy.ite(x\geq y,x,y)}$

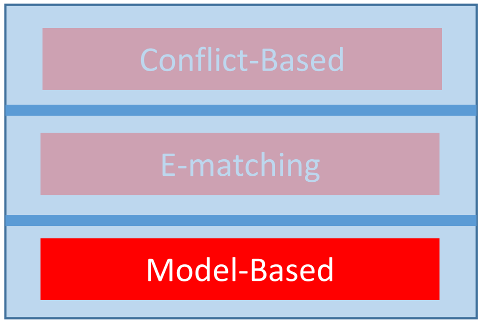
$\forall\mathtt{x{:}Int.3*g(x)+5*h(x)=x}$ → $\mathtt{g}^{\mathscr{M}} := \lambda\mathtt{x.-3*x}$
$\mathtt{h}^{\mathscr{M}} := \lambda\mathtt{x.2*x}$

$\forall\mathtt{xy{:}Int.u(x+y)+11*v(w(x))=x+y}$ → ???

Challenge : Completeness



- Seen techniques for which:
  - Ground Solver may answer unsat
  - Quantifiers Module (+ model-based instantiation) may answer sat
- Under what conditions are these techniques *terminating?*



Conflict-Based

E-matching

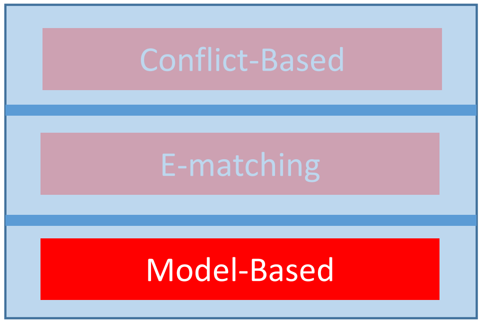
Model-Based

Challenge : Completeness

- Seen techniques for which:
  - Ground Solver may answer unsat
  - Quantifiers Module (+ model-based instantiation) may answer sat

- Under what conditions are these techniques *terminating?*

  A. If the domains of ∀ are interpreted as finite
     - E.g. quantified bitvectors **[Wintersteiger et al 13]**



Conflict-Based

E-matching

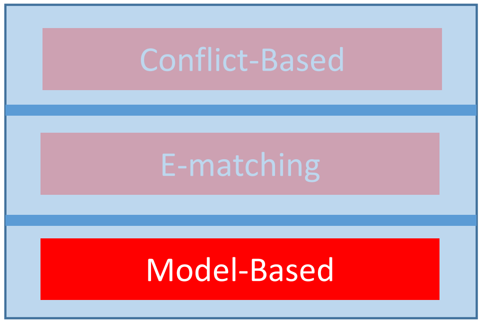
Model-Based

Challenge : Completeness

- Seen techniques for which:
  - Ground Solver may answer unsat
  - Quantifiers Module (+ model-based instantiation) may answer sat
- Under what conditions are these techniques *terminating?*
  A. If the domains of ∀ are interpreted as finite
     - E.g. quantified bitvectors **[Wintersteiger et al 13]**

  B. If the domains of ∀ may be interpreted as finite in a model
     - Finite model finding **[Reynolds et al 13]**

Challenge : Completeness

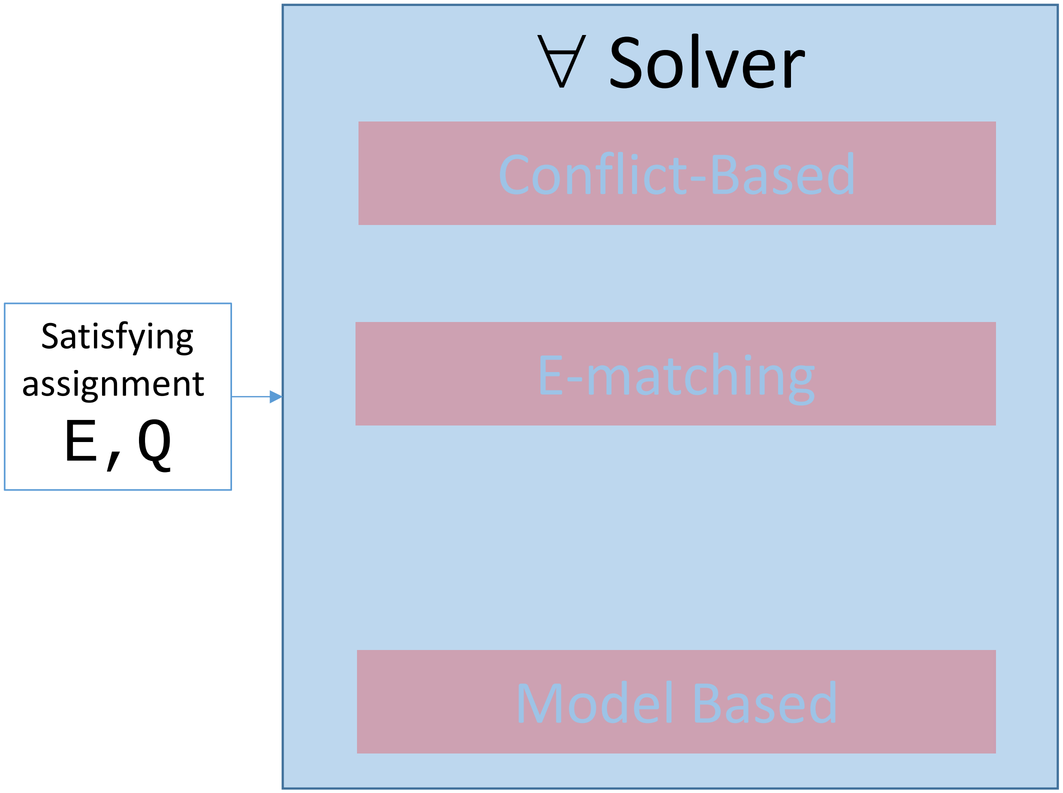


- Seen techniques for which:
  - Ground Solver may answer unsat
  - Quantifiers Module (+ model-based instantiation) may answer sat

- Under what conditions are these techniques *terminating?*
  A. If the domains of ∀ are interpreted as finite
     - E.g. quantified bitvectors **[Wintersteiger et al 13]**
  B. If the domains of ∀ may be interpreted as finite in a model
     - Finite model finding **[Reynolds et al 13]**
  C. If the domains of ∀ are infinite

     …but it can be argued that only finitely many instances will be generated
     - E.g. essentially uninterpreted fragment **[Ge+deMoura 09]**, …

Putting it Together



∀ Solver

Conflict-Based

E-matching

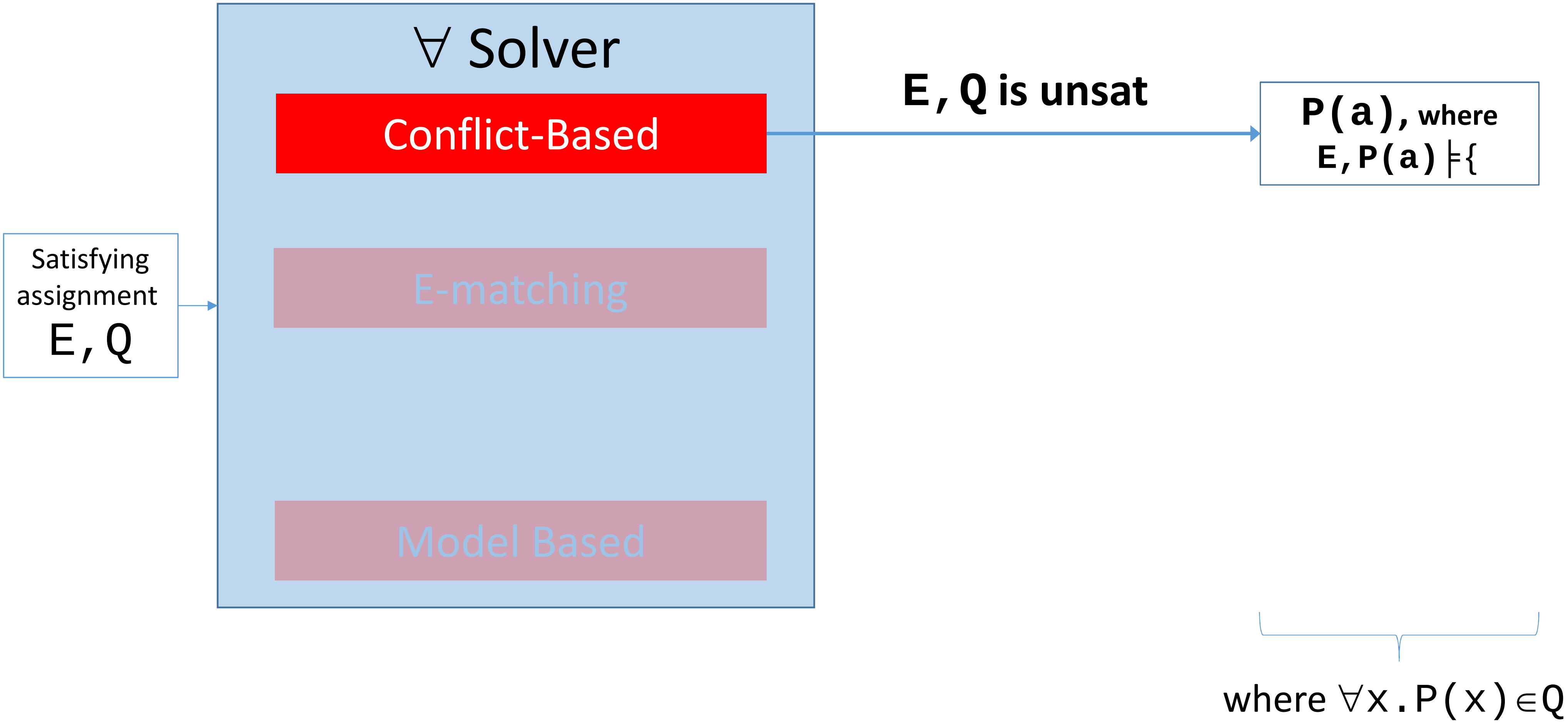
Model Based

Satisfying assignment

E, Q

- Input:
  - Ground literals E
  - Quantified formulas Q

Putting it Together



∀ Solver

Conflict-Based

E-matching

Model Based

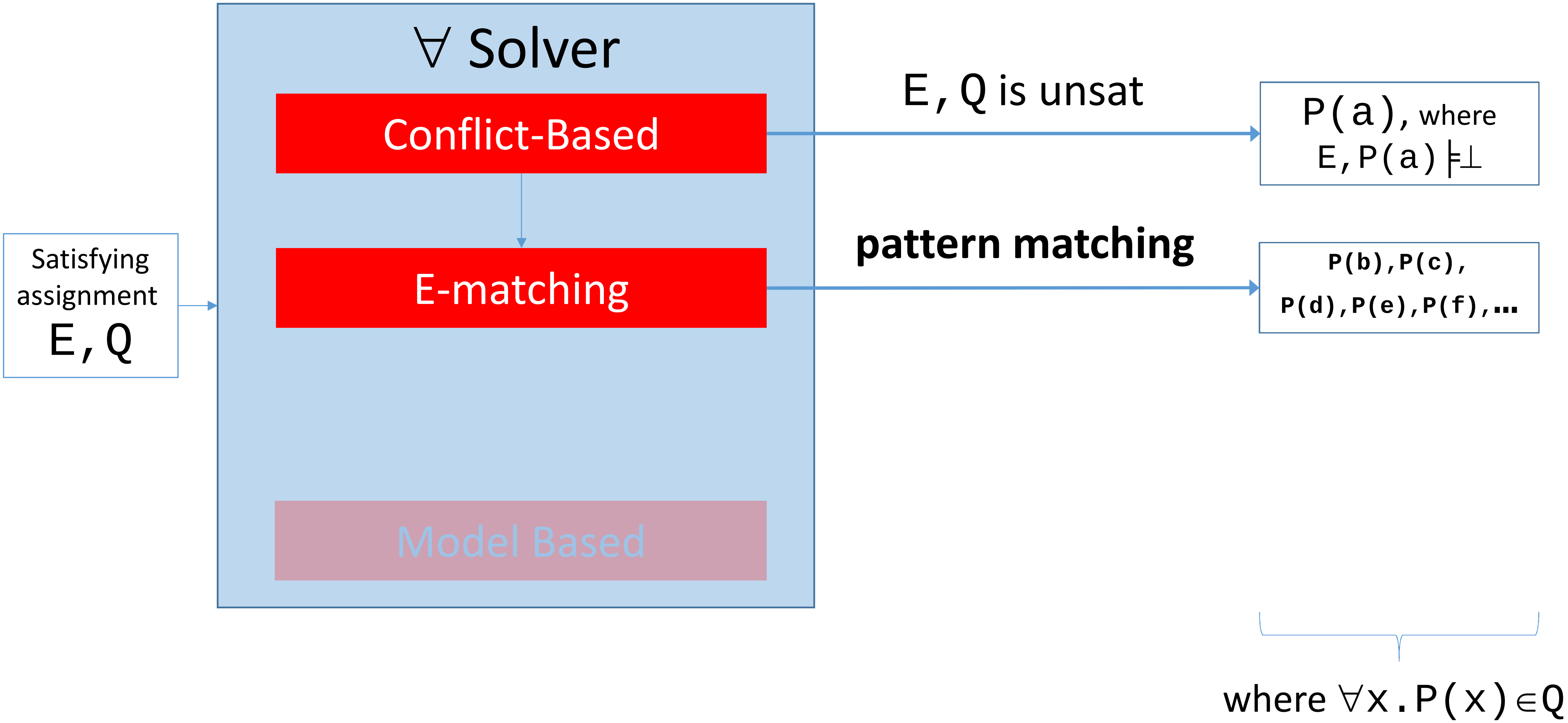
Satisfying assignment
E, Q

E, Q is unsat

P(a), where
E, P(a) ⊨{

where ∀x.P(x)∈Q

Putting it Together



Satisfying assignment

E, Q

$\forall$ Solver

Conflict-Based

E-matching

Model Based

E, Q is unsat

$P(a)$, where $E, P(a) \models \bot$

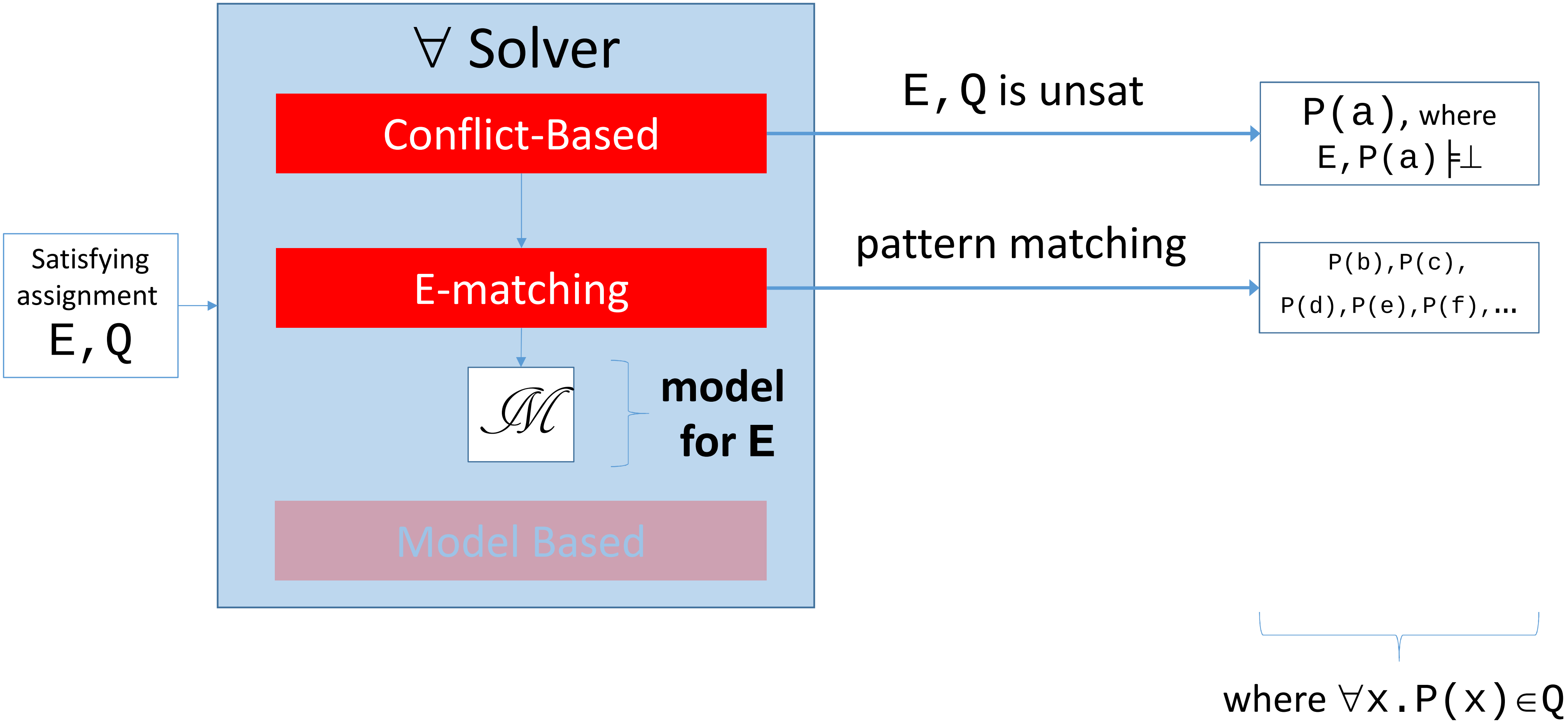
pattern matching

P(b),P(c),

P(d),P(e),P(f),...

where $\forall x . P(x) \in Q$

Putting it Together



Satisfying assignment E, Q

$\forall$ Solver

Conflict-Based

E-matching

model for E

Model Based

E, Q is unsat

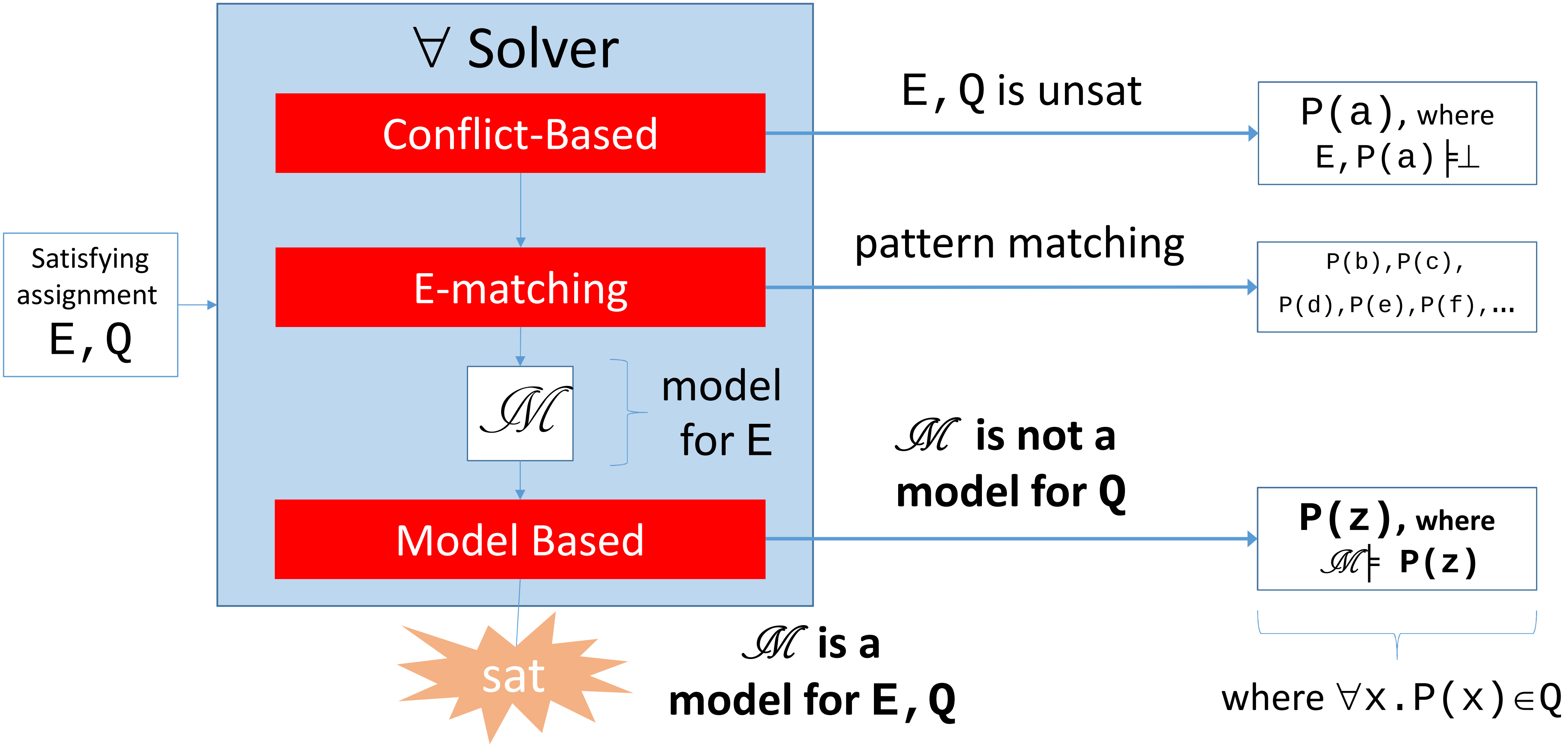
$P(a)$, where $E, P(a) \models \bot$

pattern matching

$P(b), P(c), P(d), P(e), P(f), \ldots$

where $\forall x. P(x) \in Q$

Putting it Together



Satisfying assignment

E, Q

$\forall$ Solver

Conflict-Based

E-matching

$\mathscr{M}$

model for E

Model Based

sat

E, Q is unsat

$P(a)$, where $E, P(a) \models \bot$

pattern matching

P(b), P(c), P(d), P(e), P(f), ...

$\mathscr{M}$ is not a model for Q

$P(z)$, where $\mathscr{M} \models P(z)$

where $\forall x. P(x) \in Q$

$\mathscr{M}$ is a model for E, Q

Topics Not Covered

- Eager Quantifier Instantiation
- Relevancy
- Preprocessing
- Theory-specific instantiation procedures
- Superposition-based techniques

Exercise

$\forall x.(P(x) \vee R(x-5)) \wedge \forall x.Q(f(x),x) \wedge \forall xy.(P(f(x)) \vee Q(x,y))$
$(\neg P(a+5) \vee a=f(b)) \wedge (\neg R(a) \vee \neg Q(a,b)) \wedge R(f(b)) \wedge a=f(a)-5$

- Is this satisfiable or unsatisfiable?

EXAMPLE 4 (optional)…

E-matching, Conflict-Based, Model-based:

- **Common thread:** satisfiability of ∀ + UF + theories is hard!
  - E-matching:
    - Pattern selection, matching modulo theories
  - Conflict-based:
    - Matching is incomplete, entailment tests are expensive
  - Model-based:
    - Models are complex, interpreted domains (e.g. Int) may be infinite

E-matching, Conflict-Based, Model-based:

- **Common thread:** satisfiability of ∀ + UF + theories is hard!
  - E-matching:
    - Pattern selection, matching modulo theories
  - Conflict-based:
    - Matching is incomplete, entailment tests are expensive
  - Model-based:
    - Models are complex, interpreted domains (e.g. Int) may be infinite

⇒ But reasoning about ∀ + theories without UF isn't as bad:

- Classic ∀-elimination algorithms are decision procedures for ∀ in:
  - LRA **[Ferrante+Rackoff 79, Loos+Wiespfenning 93]** , LIA **[Cooper 72]**, datatypes, …

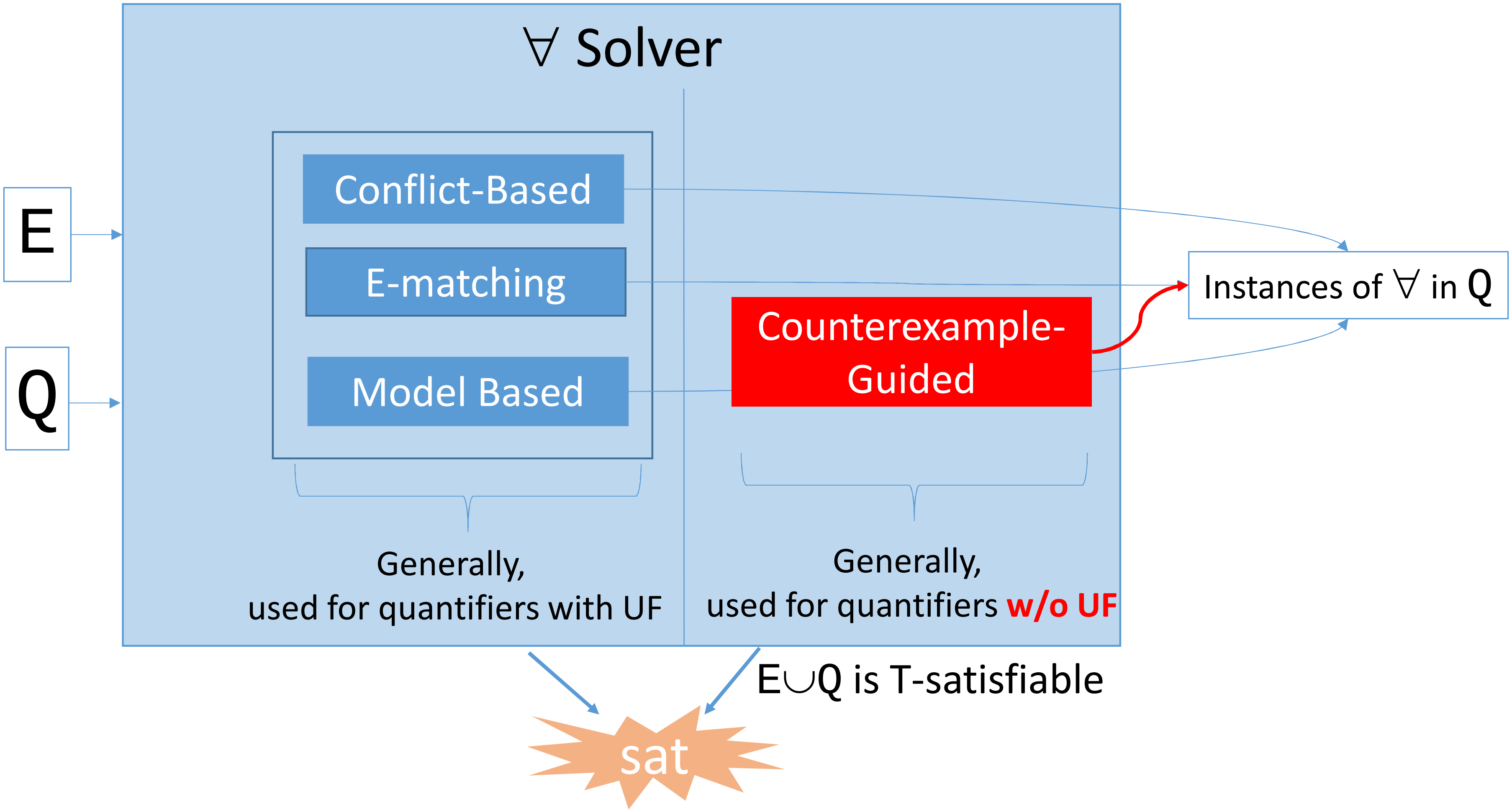
E-matching, Conflict-Based, Model-based:

- **Common thread:** satisfiability of ∀ + UF + theories is hard!
  - E-matching:
    - Pattern selection, matching modulo theories
  - Conflict-based:
    - Matching is incomplete, entailment tests are expensive
  - Model-based:
    - Models are complex, interpreted domains (e.g. Int) may be infinite

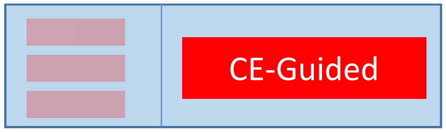
⇒ But reasoning about ∀ + theories without UF isn't as bad:

- Classic ∀-elimination algorithms are decision procedures for ∀ in:
  - LRA **[Ferrante+Rackoff 79, Loos+Wiespfenning 93]** , LIA **[Cooper 72]**, datatypes, …
- Can classic ∀-elimination algorithms be leveraged in an DPLL(T) context?
  - Yes: **[Monniaux 2010, Bjorner 2012, Reynolds et al 2015, Bjorner/Janota 2016]**

Techniques for Quantifier Instantiation

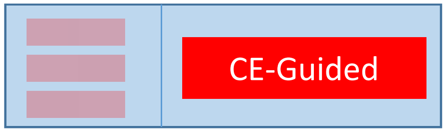


Counterexample-Guided Instantiation



- Variants implemented in number of tools:
  - **Z3** **[Bjorner 2012, Bjorner/Janota 2016]**
  - Tools using Z3 as backend: **SPACER** **[Komuravelli et al 2014]** **UFO** **[Fedyukovich et al 2016]**
  - **Yices** **[Dutertre 2015]**
  - **CVC4** **[Reynolds et al 2015]**
  - **Boolector** **[Preiner et al 2017]**

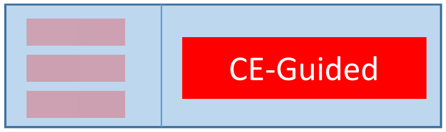
Counterexample-Guided Instantiation



- High-level idea:
  - Quantifier elimination (e.g. for LIA) says:

$$\exists \mathtt{x}.\psi[\mathtt{x}] \Leftrightarrow \psi[\mathtt{t}_1] \vee \ldots \vee \psi[\mathtt{t}_n] \text{ for finite } n$$

Counterexample-Guided Instantiation

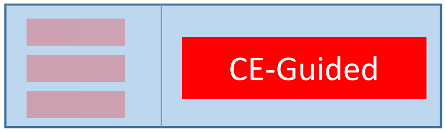


- High-level idea:
  - Quantifier elimination (e.g. for LIA) says:

?x.Ėy[x] ę Ėy[t_1] ė … ė Ėy[t_n] for finite n

(consider the dual)

Counterexample-Guided Instantiation

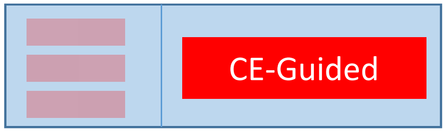


- High-level idea:
  - Quantifier elimination (e.g. for LIA) says:

$$\forall x . \neg \psi[x] \Leftrightarrow \neg \psi[t_1] \wedge \ldots \wedge \neg \psi[t_n] \text{ for finite } n$$

  - Enumerate these instances via a counterexample-guided loop, that is:
    - Terminating: enumerate at most n instances
    - Efficient in practice: typically terminates after $<<n$ instances

Counterexample-Guided Instantiation

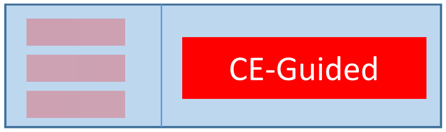


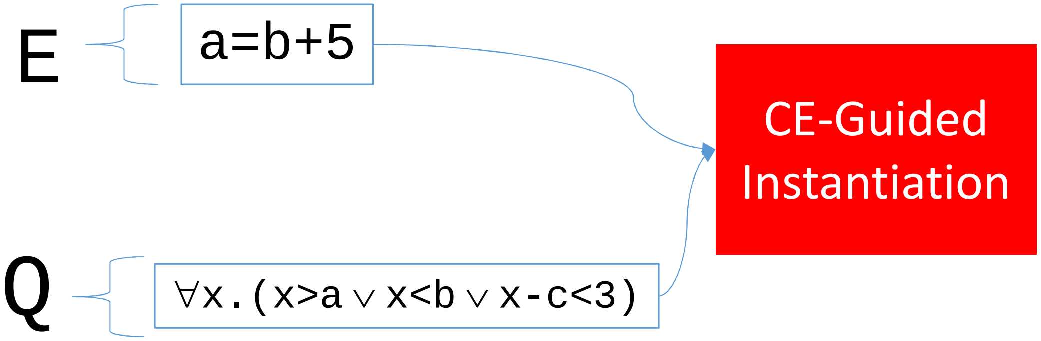
$$E \left\{ \boxed{a=b+5} \right.$$

$$\underbrace{Q \left\{ \boxed{\forall x.(x>a \vee x<b \vee x-c<3)} \right.}_{}$$

E, Q contain no uninterpreted functions, only linear arithmetic symbols

Counterexample-Guided Instantiation





E

$a=b+5$

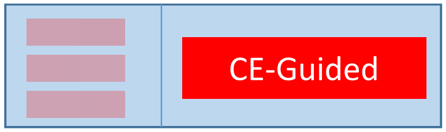
Q

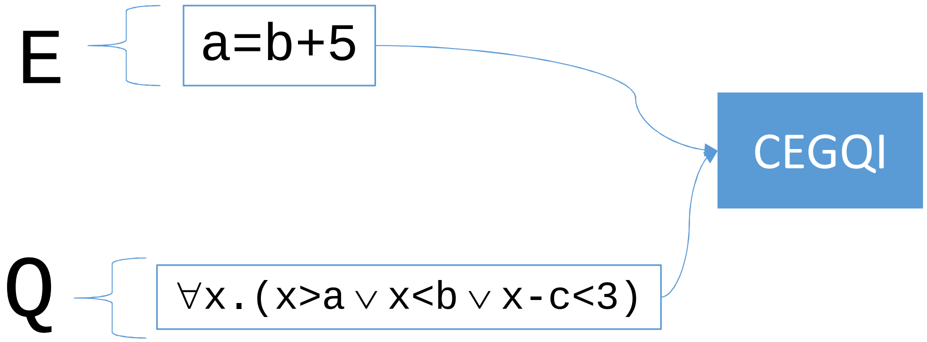
$\forall x.(x>a \lor x<b \lor x-c<3)$

CE-Guided Instantiation

⇒Use *counterexample-guided instantiation*

Counterexample-Guided Instantiation





E

a=b+5

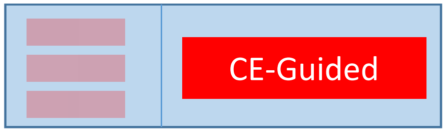
Q

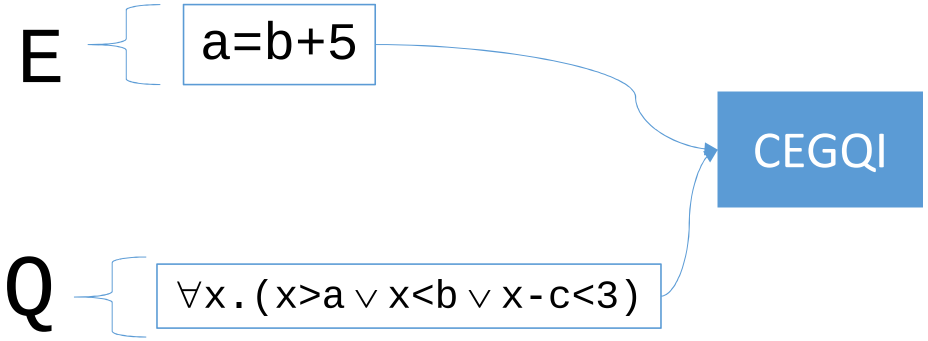
$\forall x.(x>a \lor x<b \lor x-c<3)$

CEGQI

⇒Use *counterexample-guided instantiation*

Counterexample-Guided Instantiation





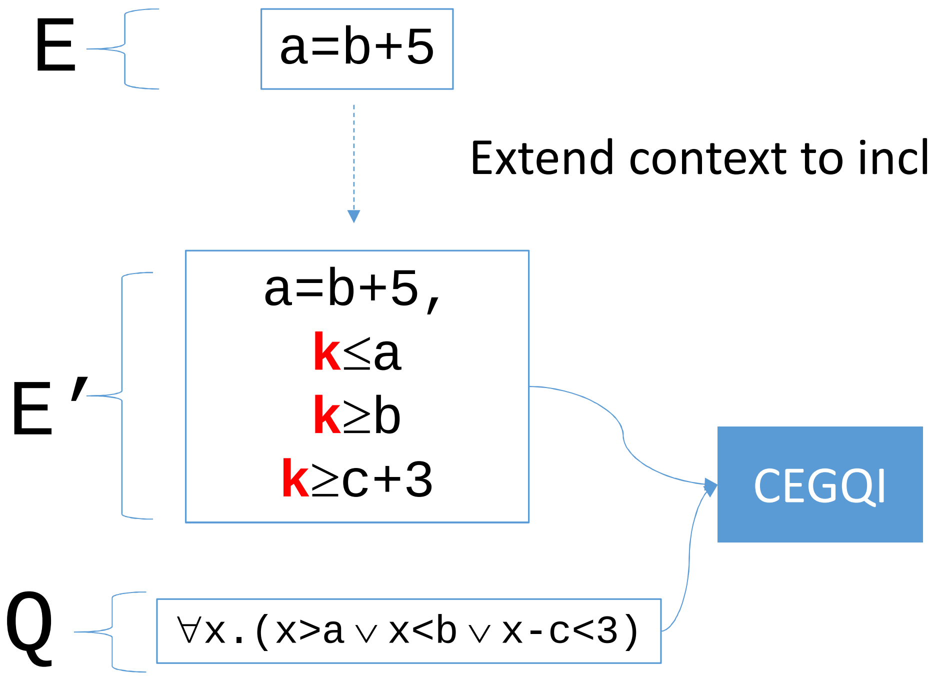
$\forall x.(x>a \vee x<b \vee x-c<3)$

⇒With respect to *model-based instantiation*:

- Similar: based on finding models for Q’s negation

Counterexample-Guided Instantiation

CE-Guided



E

$a=b+5$

Extend context to include counterexample

E'

$a=b+5,$
$k \leq a$
$k \geq b$
$k \geq c+3$

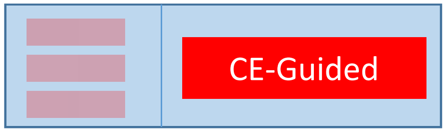
CEGQI

Q

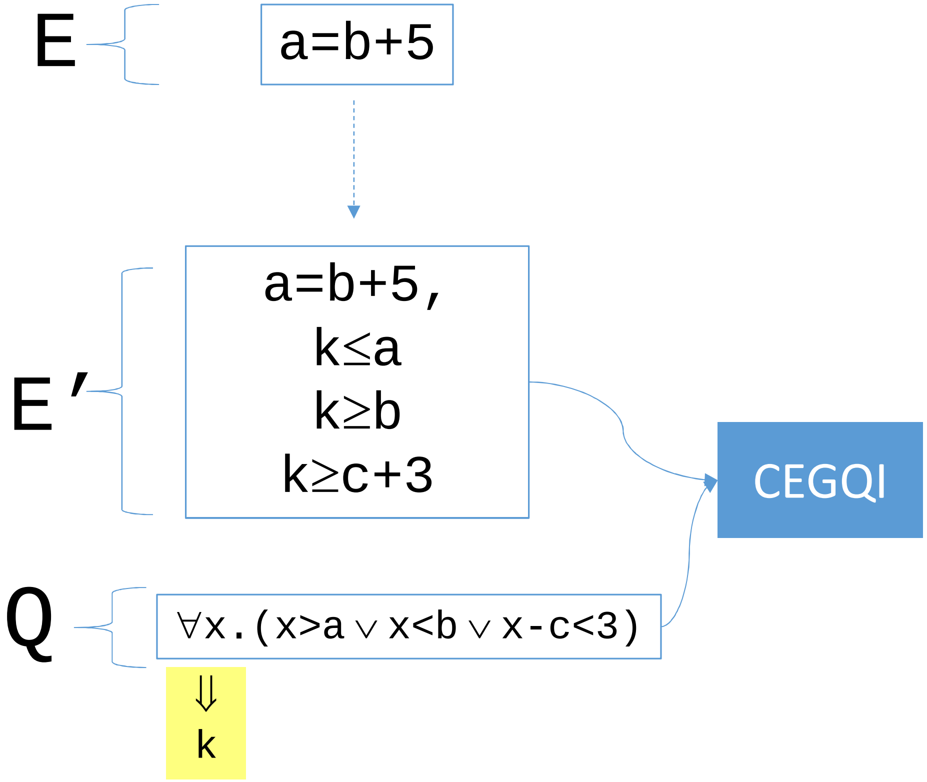
$\forall x.(x>a \vee x<b \vee x-c<3)$

...has counterexample k iff $\exists k.\neg(k>a \vee k<b \vee k-c<3)$

Counterexample-Guided Instantiation



CE-Guided



E

$a=b+5$

E′

$a=b+5,$
$k\leq a$
$k\geq b$
$k\geq c+3$

CEGQI

Q

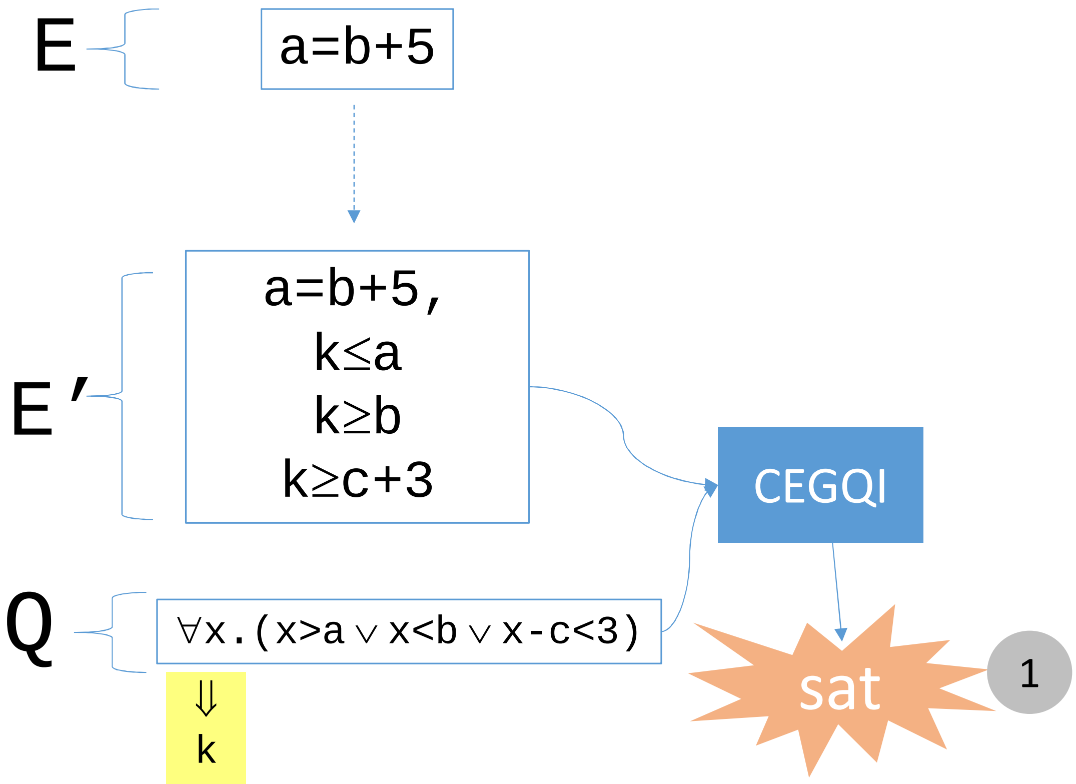
$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$
k

One of two cases…

Counterexample-Guided Instantiation

CE-Guided

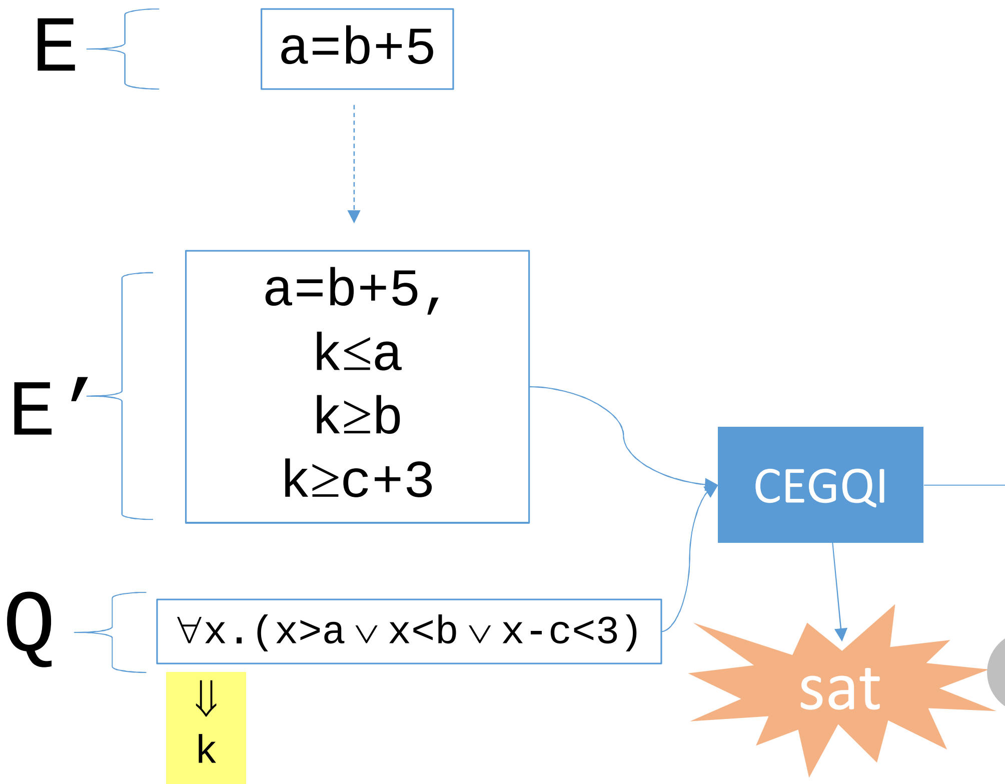


If E′ is T-unsatisfiable,
all models of E also satisfy Q

(since $E' \equiv E \cup \neg Q \models_T \bot$ implies $E \models_T Q$)

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5

E′

a=b+5,
k≤a
k≥b
k≥c+3

Q

$\forall x.(x>a \lor x<b \lor x-c<3)$

⇓
k

CEGQI

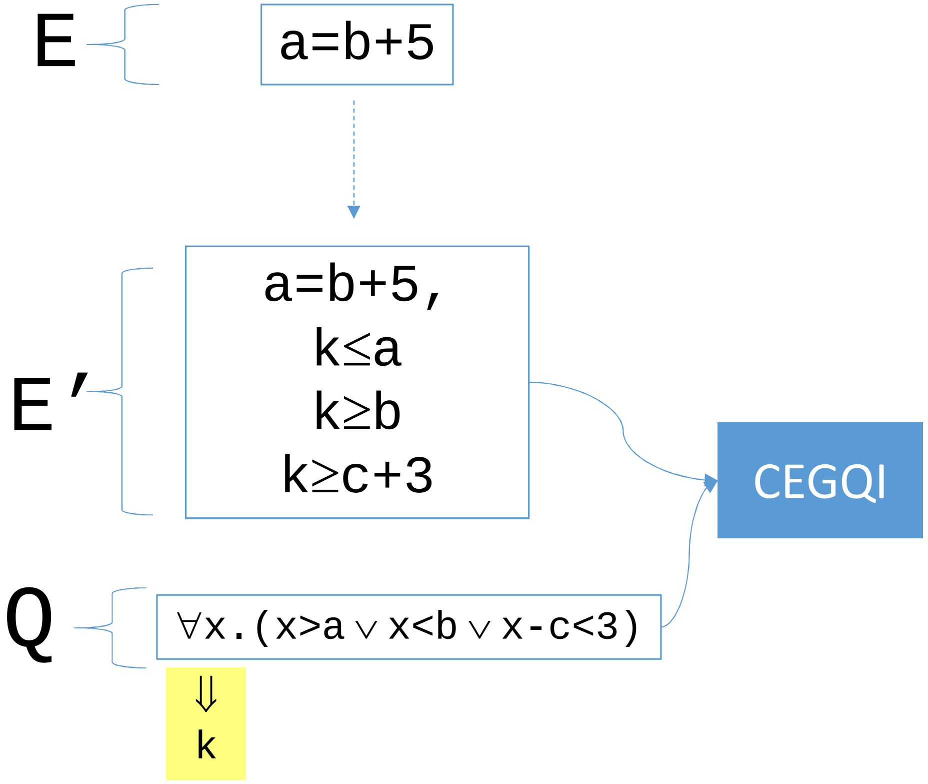
sat

2 Return an instance based on model for E′

1 If E′ is T-unsatisfiable,
all models of E also satisfy Q

(since $E' \equiv E \cup \neg Q \models_T \bot$ implies $E \models_T Q$)

Counterexample-Guided Instantiation



E

$a=b+5$

E′

$a=b+5,$

$k \leq a$

$k \geq b$

$k \geq c+3$

CEGQI

Q

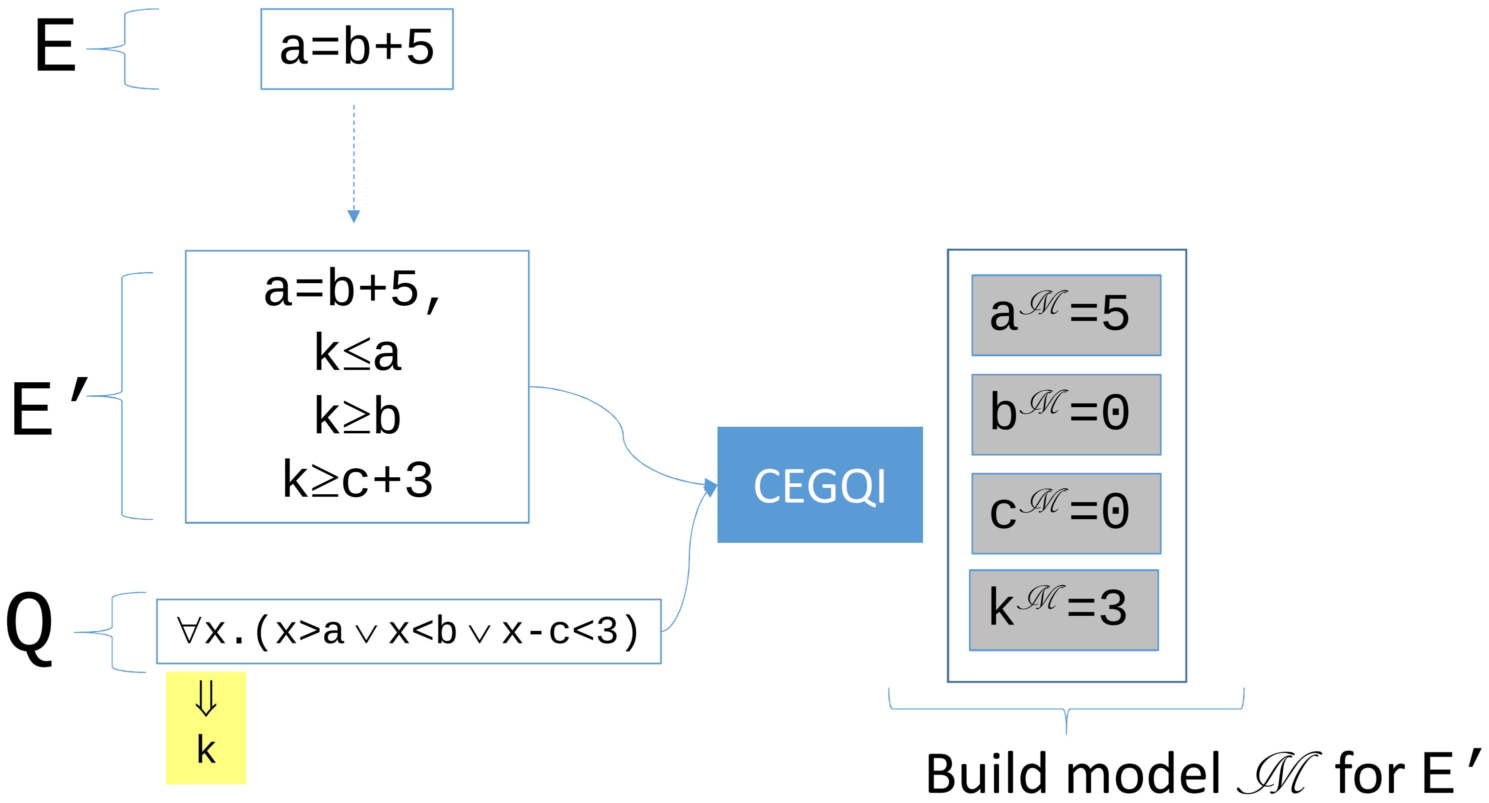
$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$

k

Counterexample-Guided Instantiation

CE-Guided



E

$a=b+5$

E′

$a=b+5,$
$k\leq a$
$k\geq b$
$k\geq c+3$

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$
k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

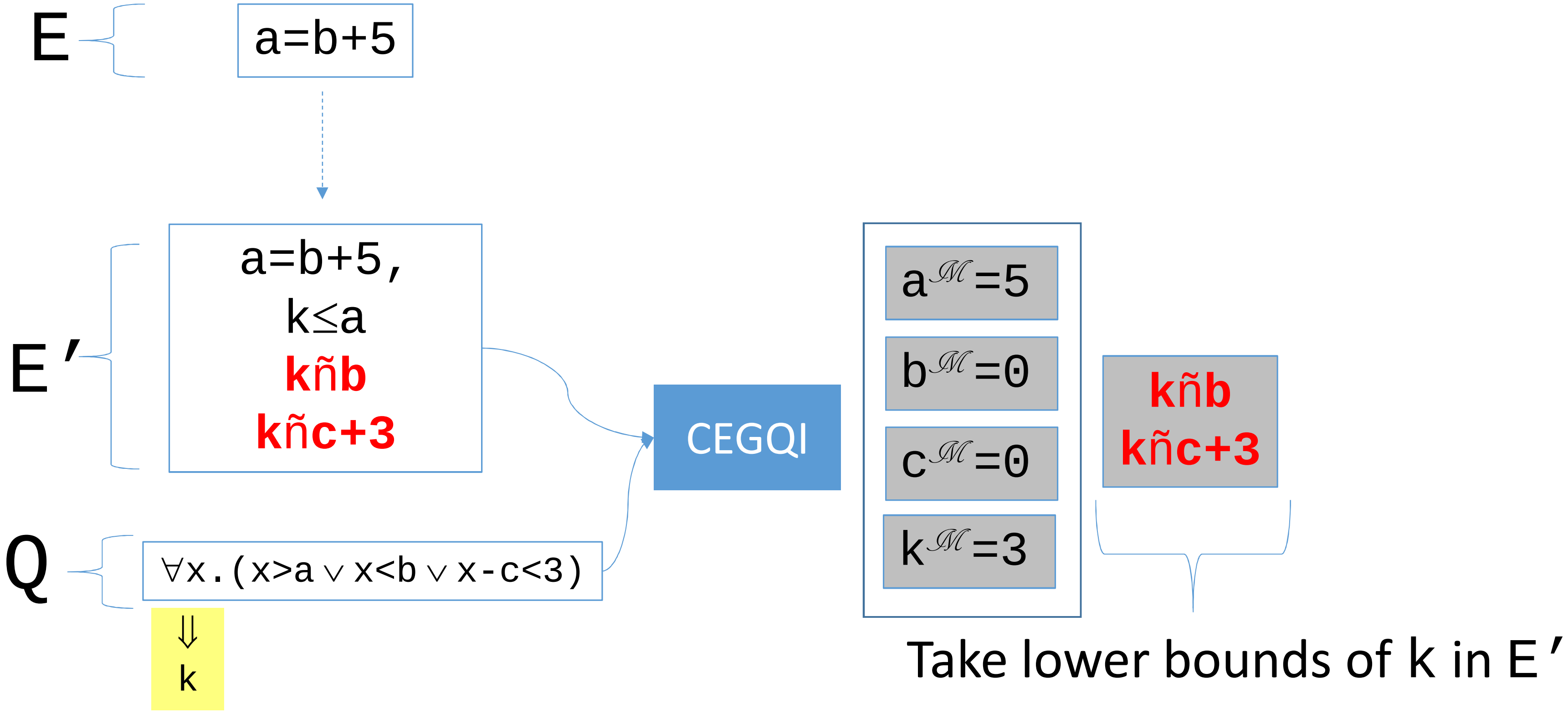
$c^{\mathscr{M}}=0$

$k^{\mathscr{M}}=3$

Build model $\mathscr{M}$ for E′

Counterexample-Guided Instantiation

CE-Guided



E: a=b+5

E′: a=b+5, k≤a **kñb** **kñc+3**

Q: ∀x.(x>a ∨ x<b ∨ x-c<3)

⇓ k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=0$

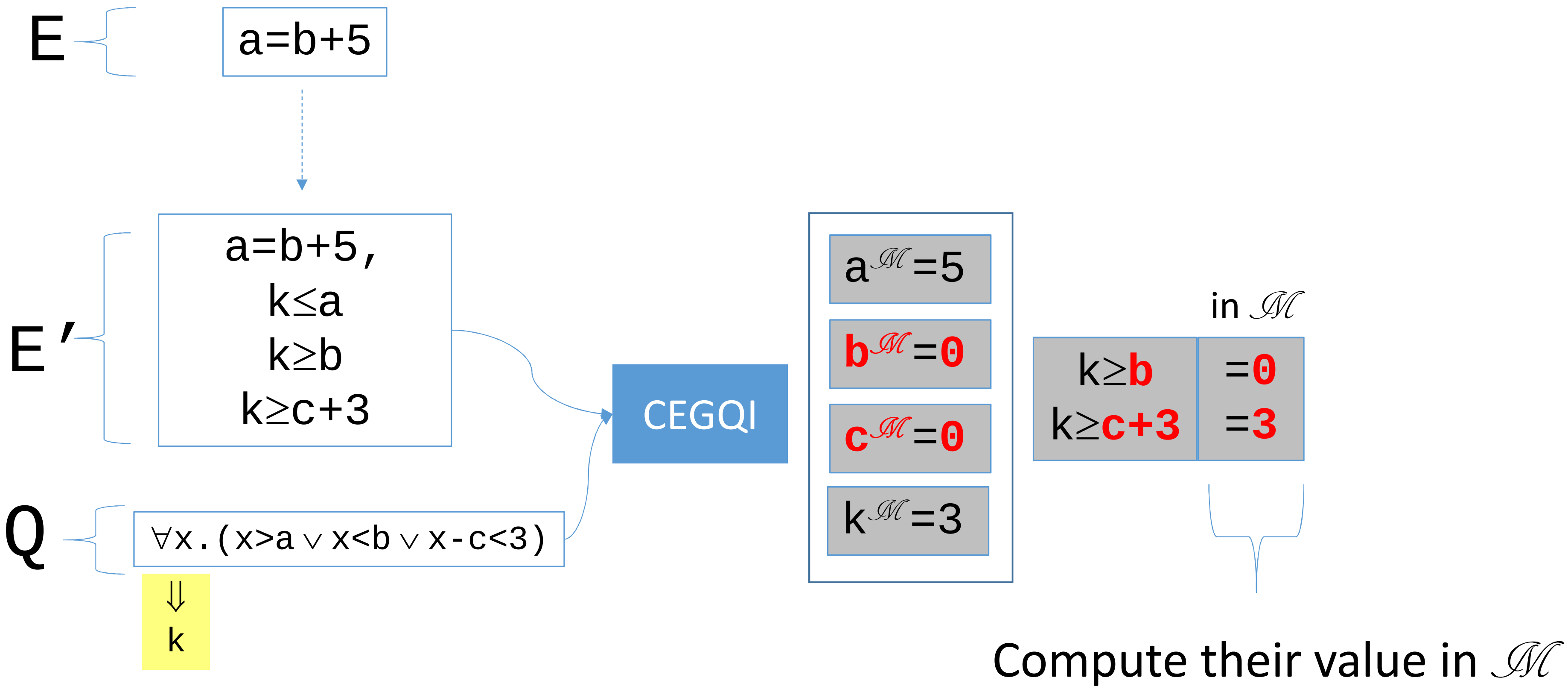
$k^{\mathscr{M}}=3$

kñb **kñc+3**

Take lower bounds of k in E′

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5

E′

a=b+5,
k≤a
k≥b
k≥c+3

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

⇓
k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=0$

$k^{\mathscr{M}}=3$

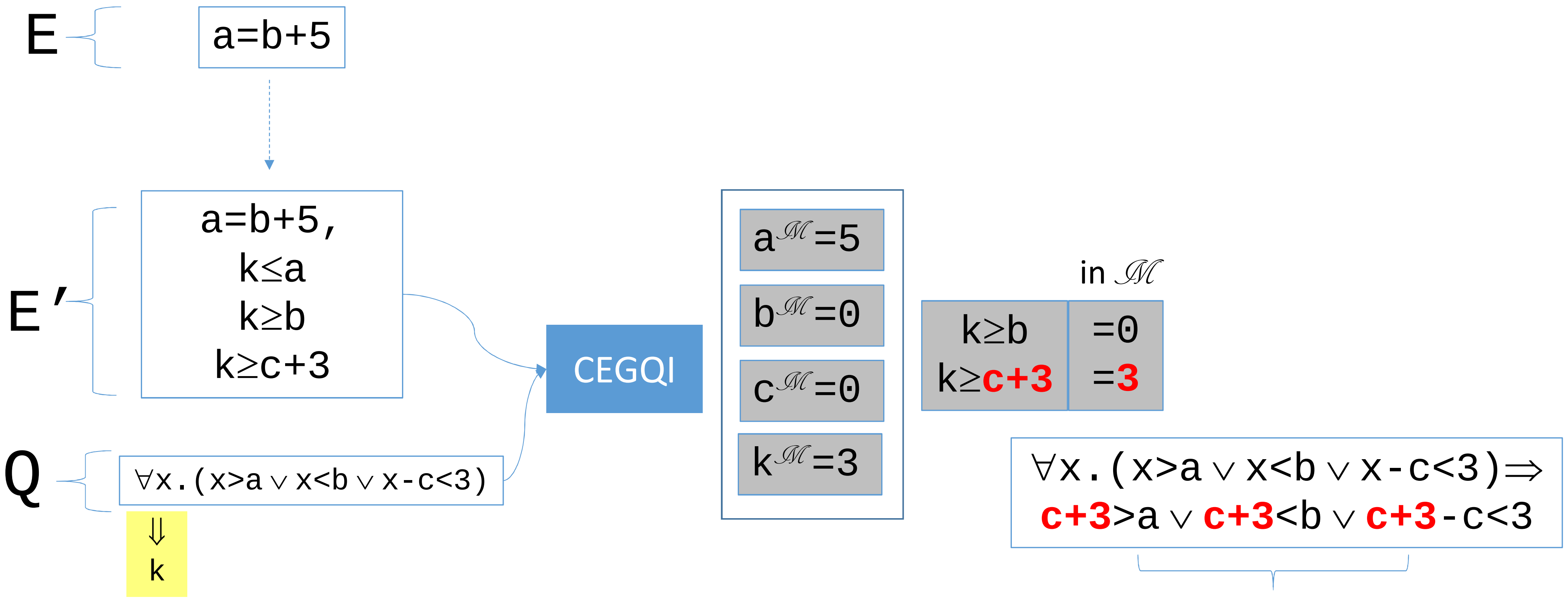
in $\mathscr{M}$

k≥b	=0
k≥c+3	=3

Compute their value in $\mathscr{M}$

Counterexample-Guided Instantiation

CE-Guided



E: $a=b+5$

E′: $a=b+5,$ $k \le a$ $k \ge b$ $k \ge c+3$

Q: $\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$ k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=0$

$k^{\mathscr{M}}=3$

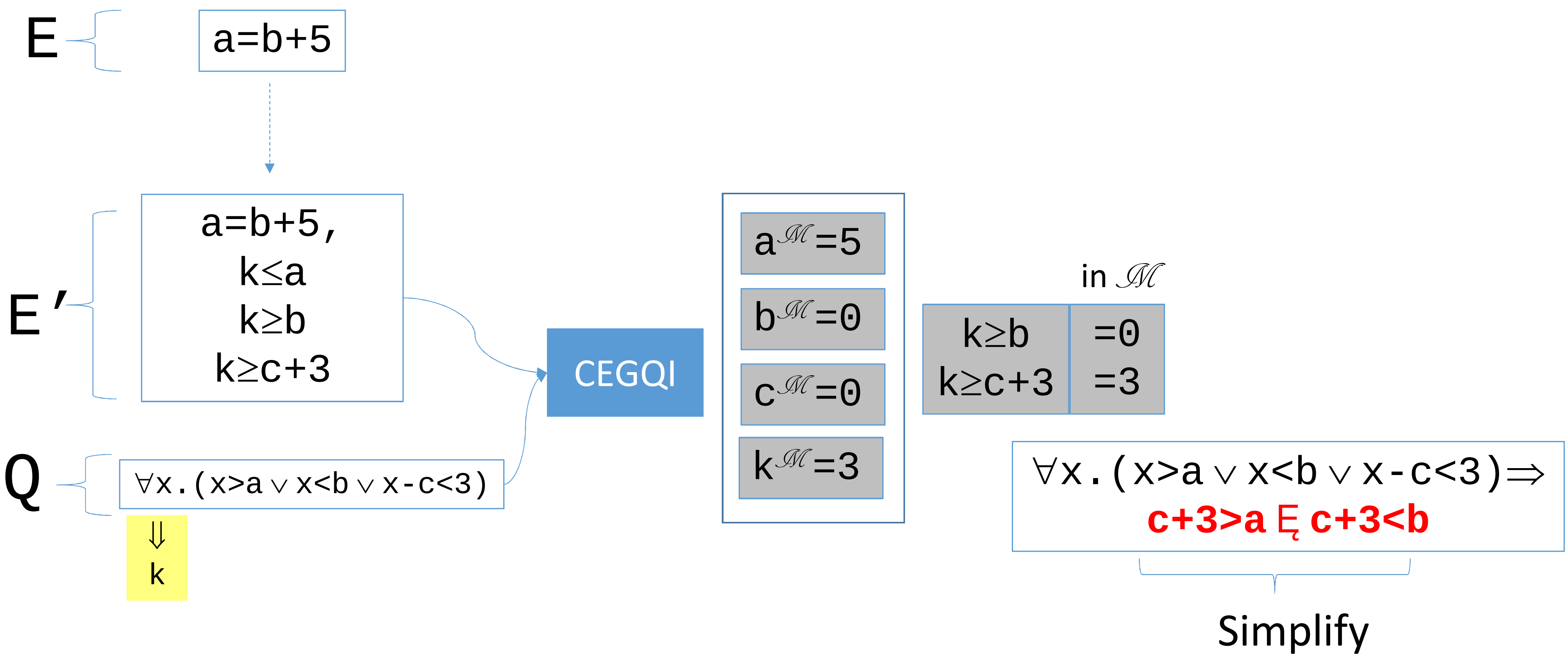
	in $\mathscr{M}$
$k \ge b$	$=0$
$k \ge \mathbf{c+3}$	$=\mathbf{3}$

$\forall x.(x>a \vee x<b \vee x-c<3) \Rightarrow$
$\mathbf{c+3}>a \vee \mathbf{c+3}<b \vee \mathbf{c+3}-c<3$

Add instance for lower bound that is maximal in $\mathscr{M}$

Counterexample-Guided Instantiation

CE-Guided



E: $a=b+5$

E′: $a=b+5$, $k \leq a$, $k \geq b$, $k \geq c+3$

Q: $\forall x.(x>a \vee x<b \vee x\text{-}c<3)$

$\Downarrow$ k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=0$

$k^{\mathscr{M}}=3$

in $\mathscr{M}$

$k \geq b$	$=0$
$k \geq c+3$	$=3$

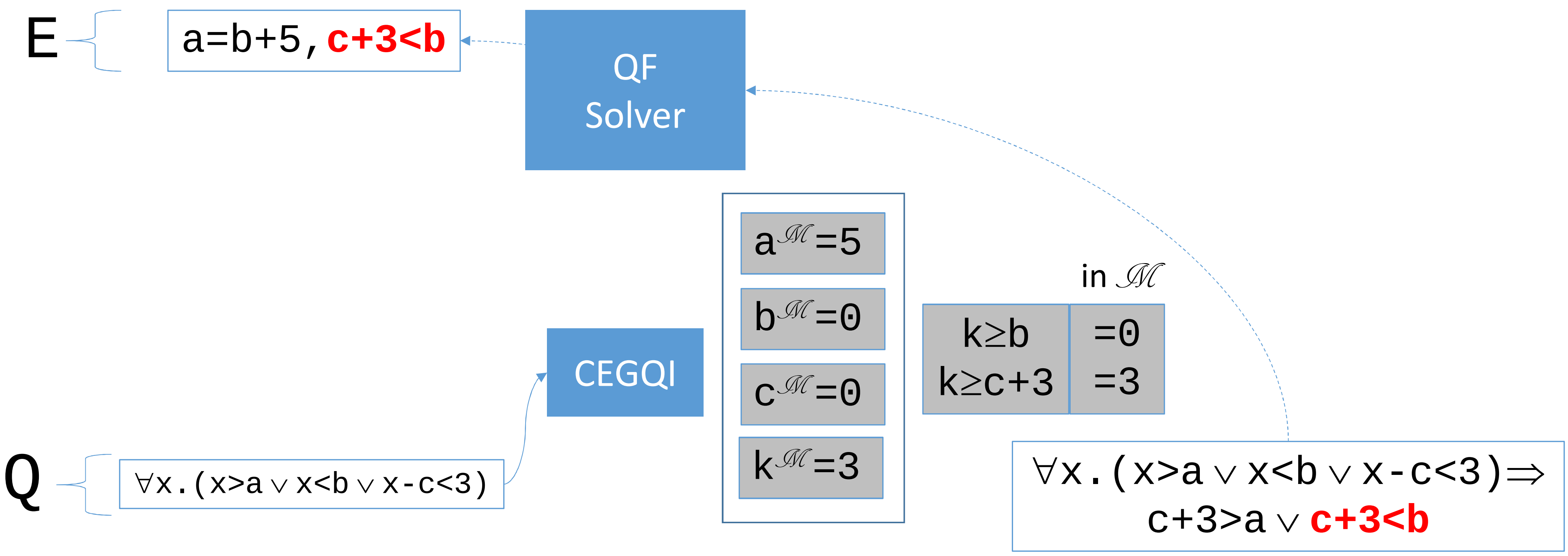
$\forall x.(x>a \vee x<b \vee x\text{-}c<3) \Rightarrow$

c+3>a Ę c+3<b

Simplify

Counterexample-Guided Instantiation

CE-Guided



E

$a=b+5$, $c+3<b$

QF Solver

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=0$

$k^{\mathscr{M}}=3$

in $\mathscr{M}$

$k\geq b$	$=0$
$k\geq c+3$	$=3$

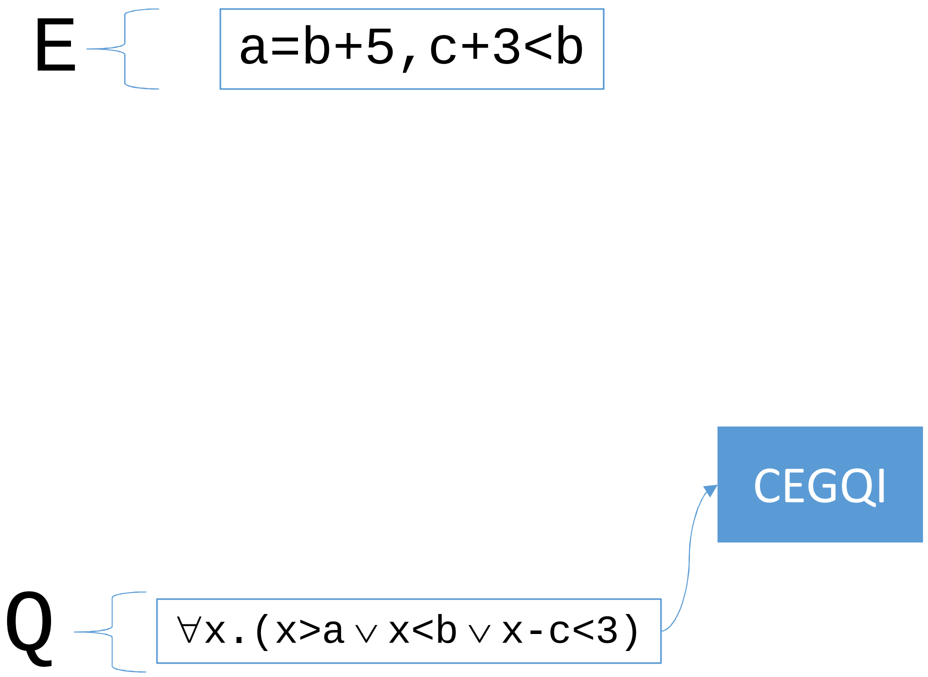
CEGQI

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

$\forall x.(x>a \vee x<b \vee x-c<3) \Rightarrow c+3>a \vee c+3<b$

Counterexample-Guided Instantiation



E

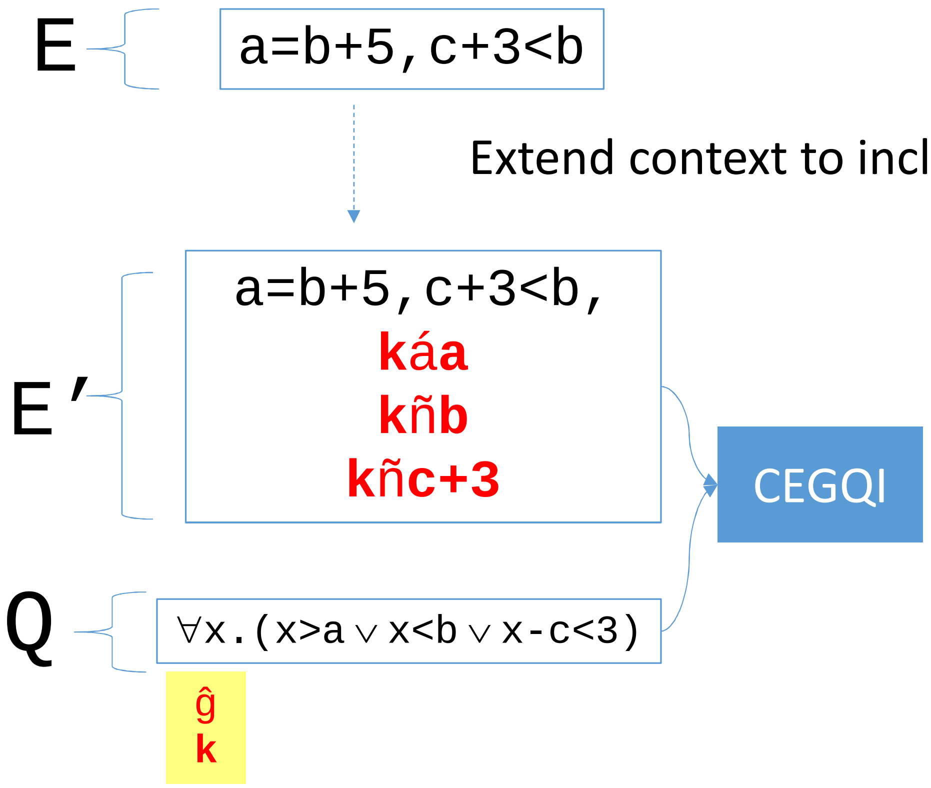
$a=b+5, c+3<b$

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5,c+3<b

Extend context to include counterexample

E'

a=b+5,c+3<b,

káa

kñb

kñc+3

Q

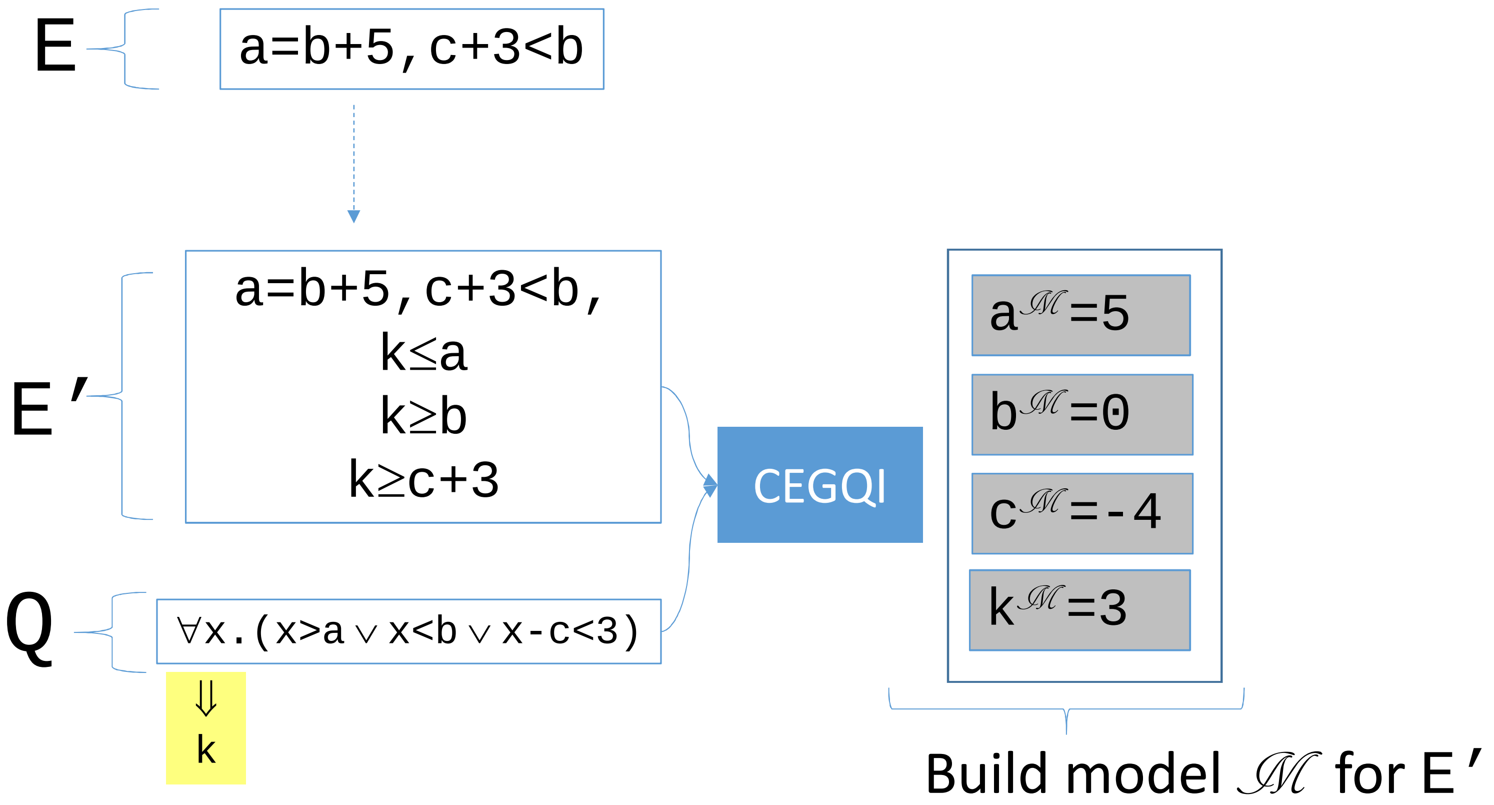
∀x.(x>a ∨ x<b ∨ x-c<3)

ĝ

k

Counterexample-Guided Instantiation

CE-Guided



E

$a=b+5, c+3<b$

E′

$a=b+5, c+3<b,$

$k \leq a$

$k \geq b$

$k \geq c+3$

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$

k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

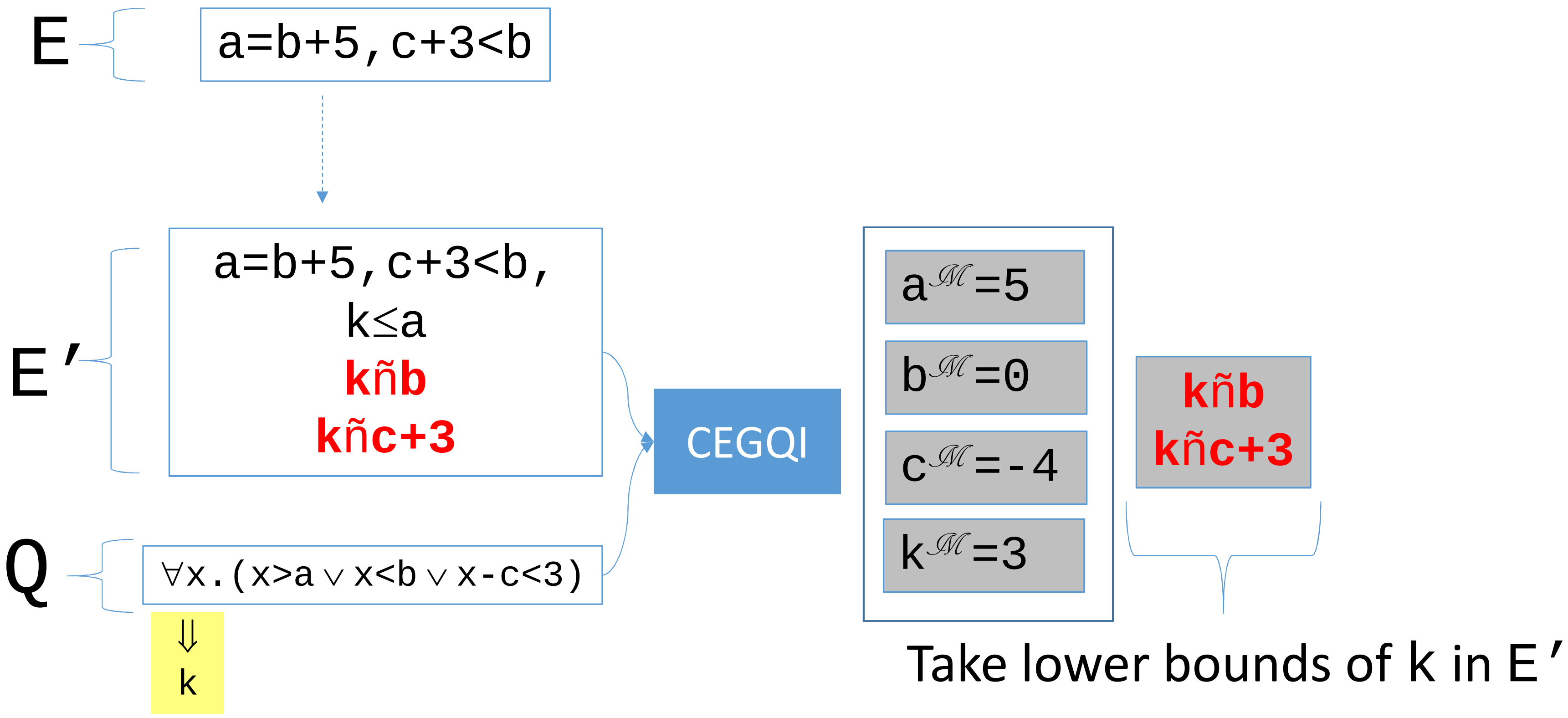
$c^{\mathscr{M}}=-4$

$k^{\mathscr{M}}=3$

Build model $\mathscr{M}$ for E′

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5,c+3<b

E′

a=b+5,c+3<b,
k≤a
kñb
kñc+3

Q

∀x.(x>a ∨ x<b ∨ x-c<3)

⇓
k

CEGQI

$a^{\mathscr{M}}$=5

$b^{\mathscr{M}}$=0

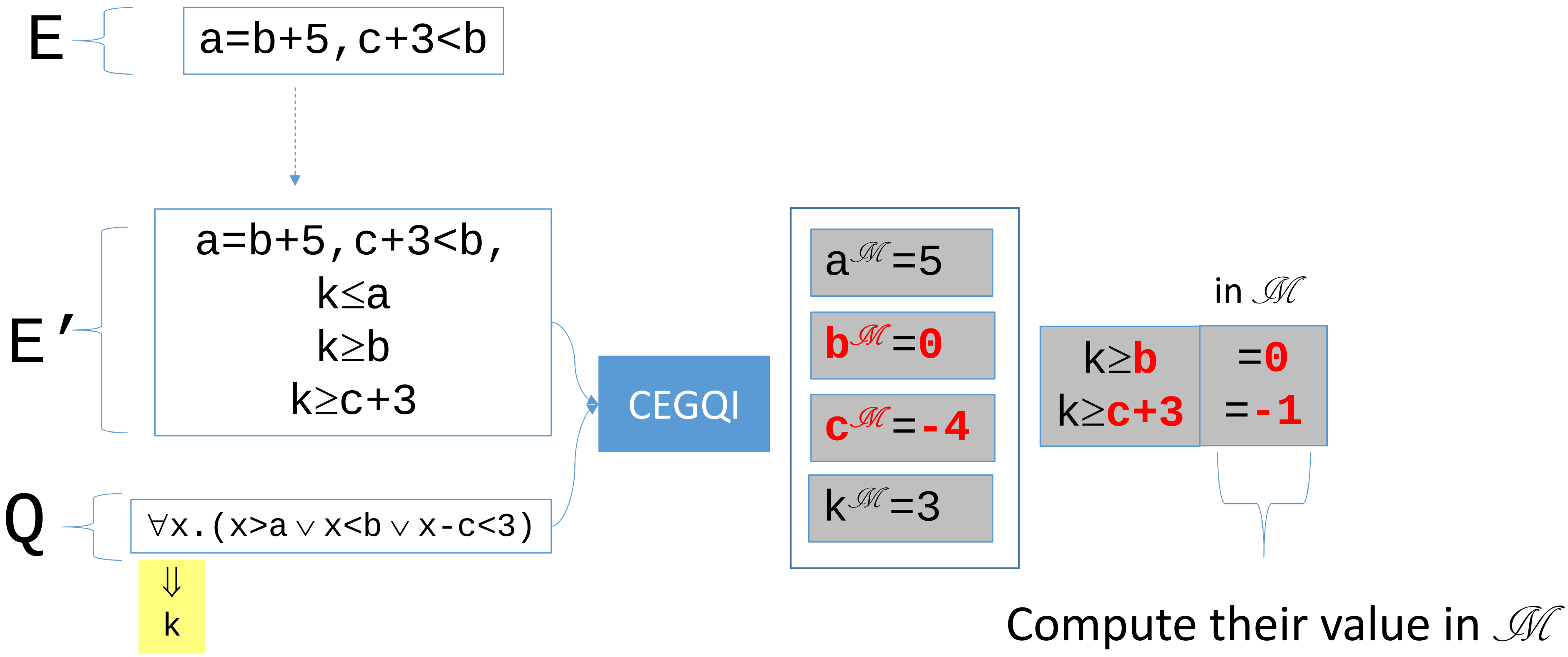
$c^{\mathscr{M}}$=-4

$k^{\mathscr{M}}$=3

kñb
kñc+3

Take lower bounds of k in E′

Counterexample-Guided Instantiation



E

a=b+5,c+3<b

E′

a=b+5,c+3<b,
k≤a
k≥b
k≥c+3

Q

∀x.(x>a ∨ x<b ∨ x-c<3)

⇓
k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=-4$

$k^{\mathscr{M}}=3$

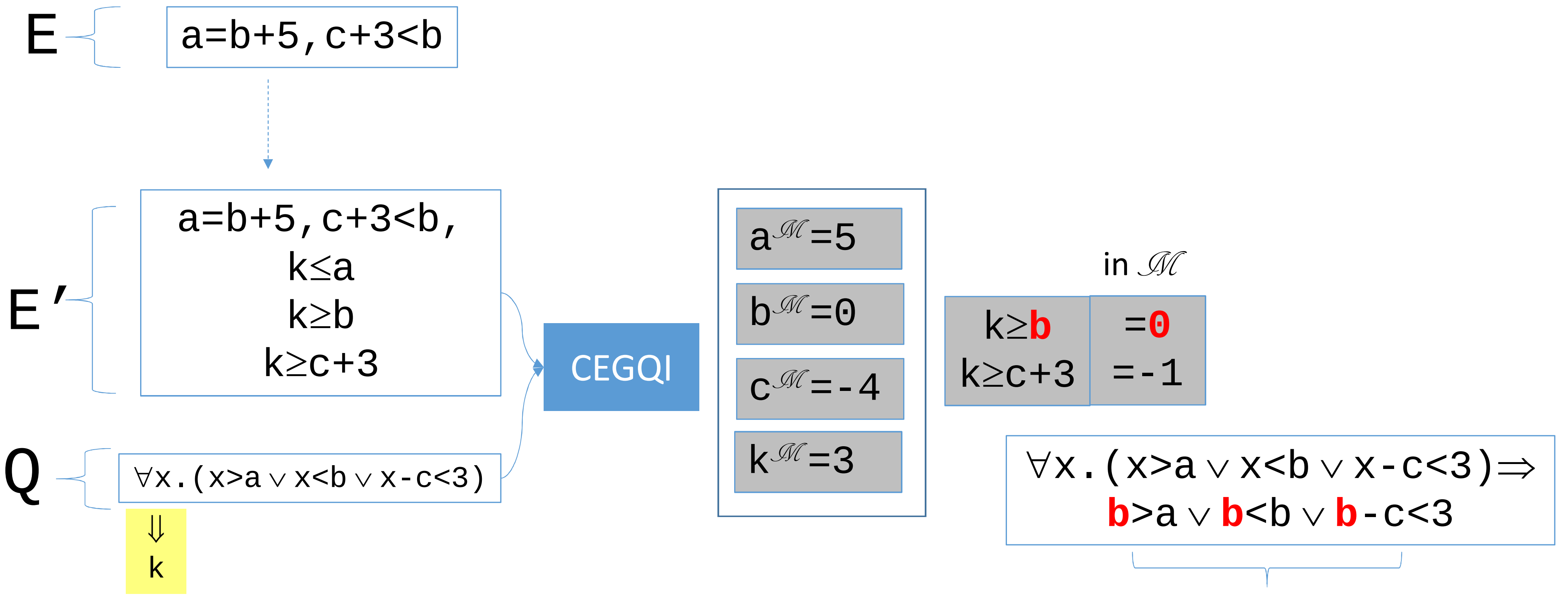
in $\mathscr{M}$

k≥**b**	=**0**
k≥**c+3**	=**-1**

Compute their value in $\mathscr{M}$

Counterexample-Guided Instantiation

CE-Guided



E

$a=b+5, c+3<b$

E′

$a=b+5, c+3<b,$
$k\leq a$
$k\geq b$
$k\geq c+3$

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$
k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=-4$

$k^{\mathscr{M}}=3$

in $\mathscr{M}$

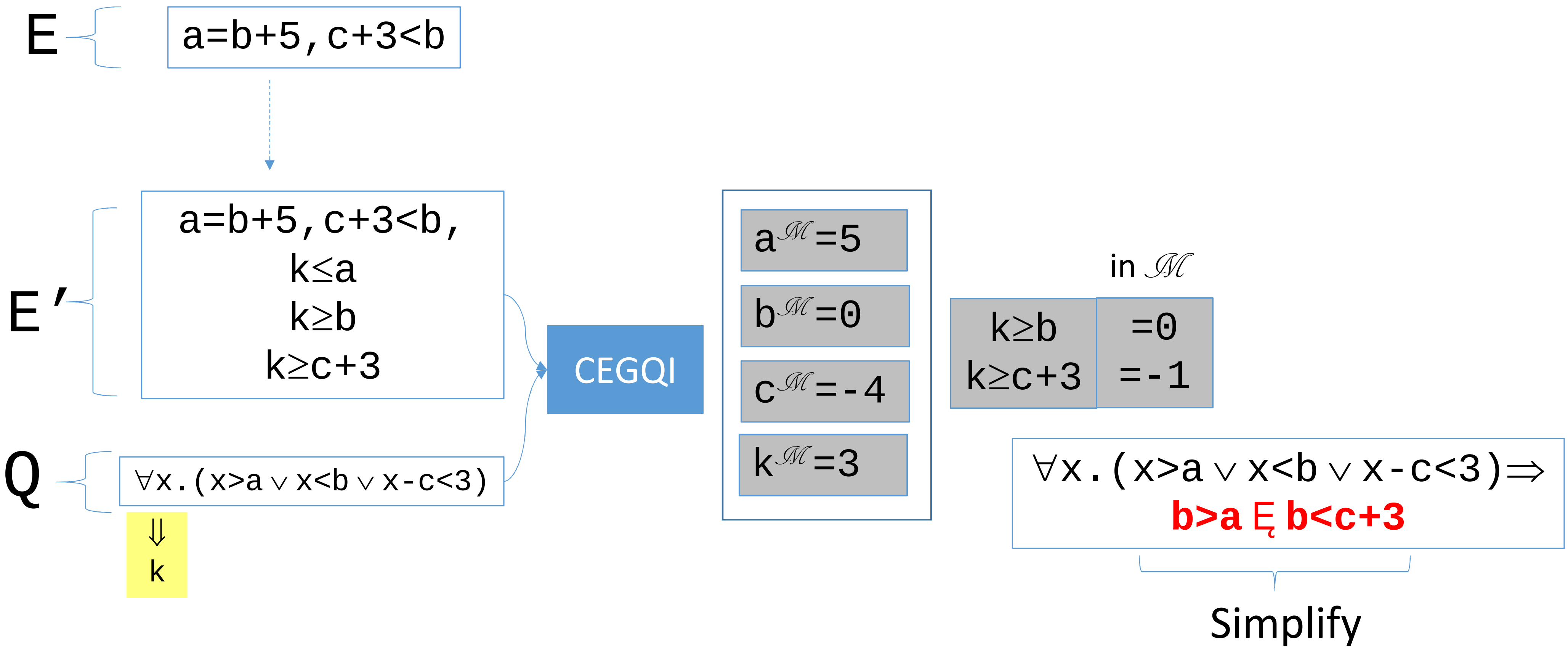
$k\geq b$	$=0$
$k\geq c+3$	$=-1$

$\forall x.(x>a \vee x<b \vee x-c<3) \Rightarrow$
$b>a \vee b<b \vee b-c<3$

Add instance for lower bound that is maximal in $\mathscr{M}$

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5,c+3<b

E′

a=b+5,c+3<b,
k≤a
k≥b
k≥c+3

Q

∀x.(x>a ∨ x<b ∨ x-c<3)

⇓
k

CEGQI

$a^{\mathscr{M}}=5$

$b^{\mathscr{M}}=0$

$c^{\mathscr{M}}=-4$

$k^{\mathscr{M}}=3$

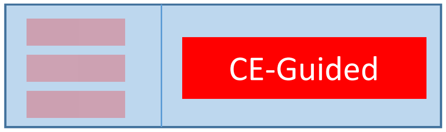
in $\mathscr{M}$

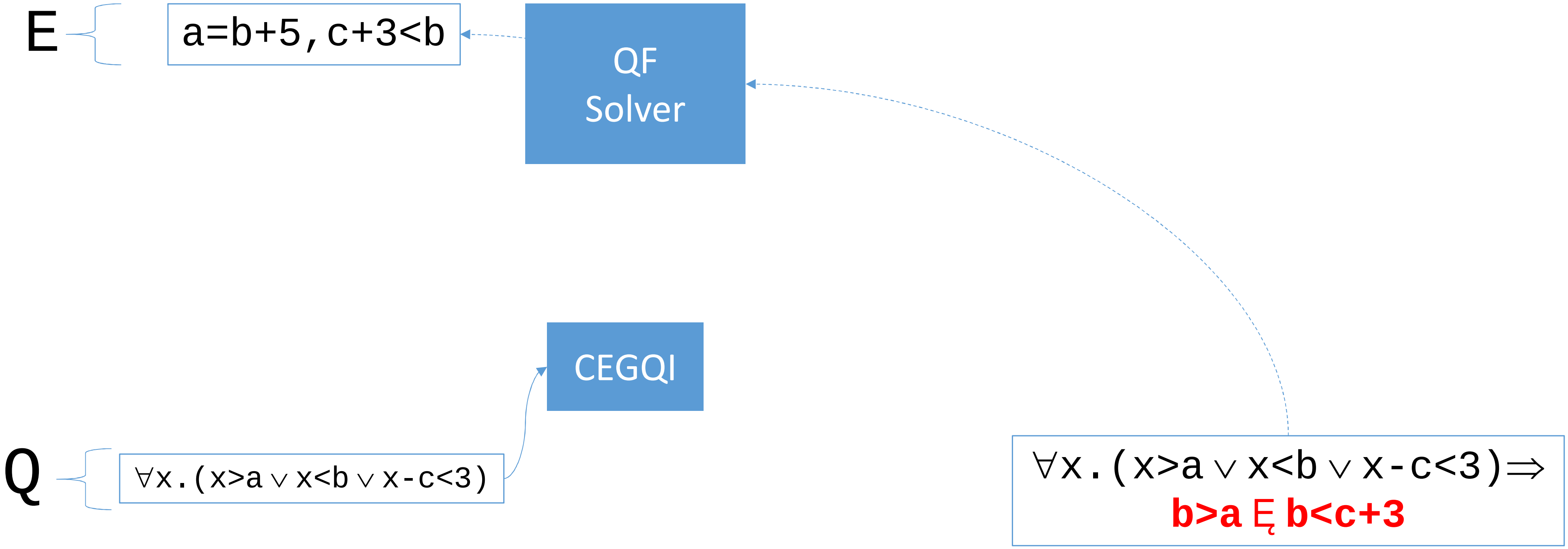
| k≥b | =0 |
| k≥c+3 | =-1 |

∀x.(x>a ∨ x<b ∨ x-c<3)⇒
b>a Ę b<c+3

Simplify

Counterexample-Guided Instantiation





E

a=b+5,c+3<b

QF Solver

CEGQI

Q

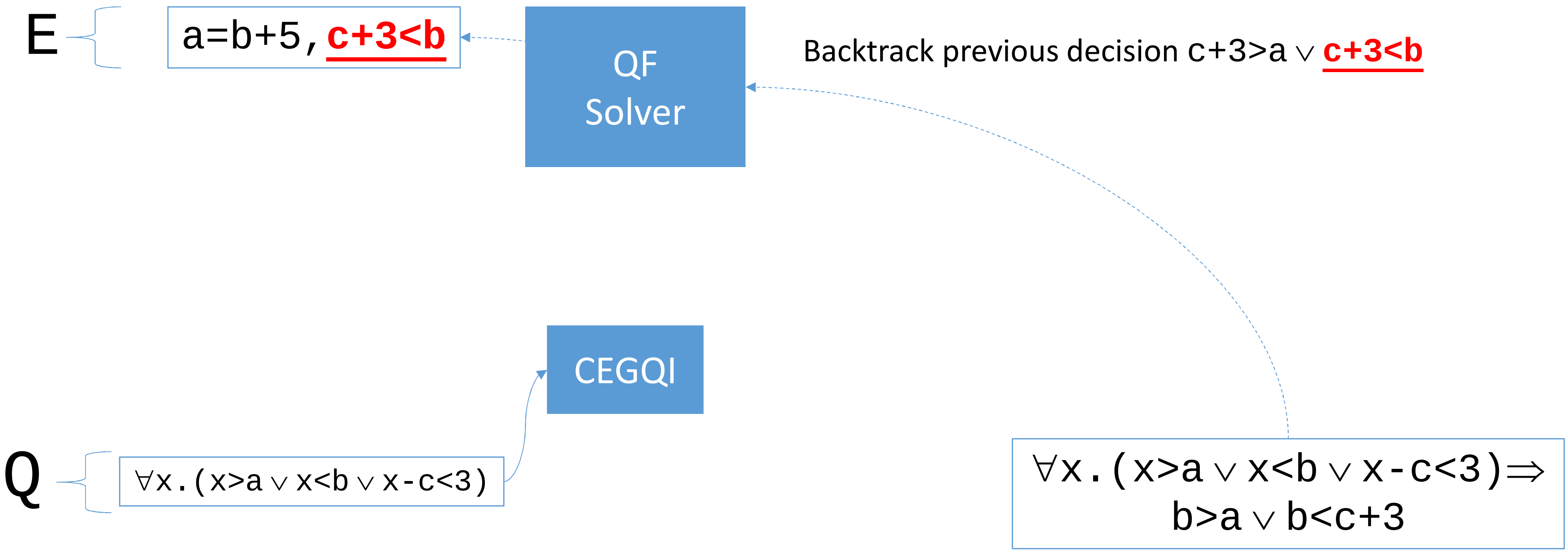
$\forall$x.(x>a $\vee$ x<b $\vee$ x-c<3)

$\forall$x.(x>a $\vee$ x<b $\vee$ x-c<3)$\Rightarrow$

b>a Ę b<c+3

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5, **c+3<b**

QF Solver

Backtrack previous decision c+3>a ∨ **c+3<b**

CEGQI

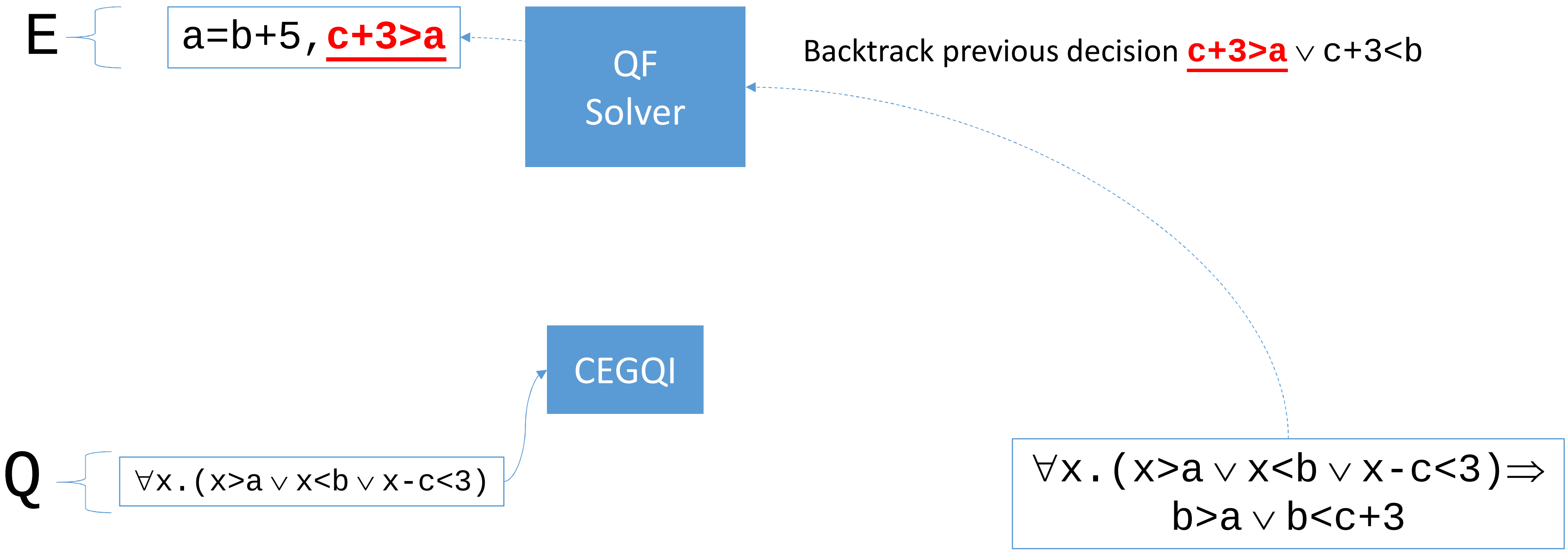
Q

∀x.(x>a ∨ x<b ∨ x-c<3)

∀x.(x>a ∨ x<b ∨ x-c<3)⇒
b>a ∨ b<c+3

Counterexample-Guided Instantiation

CE-Guided



E

$a=b+5$, **$c+3>a$**

QF Solver

Backtrack previous decision **$c+3>a$** $\vee$ $c+3<b$

CEGQI

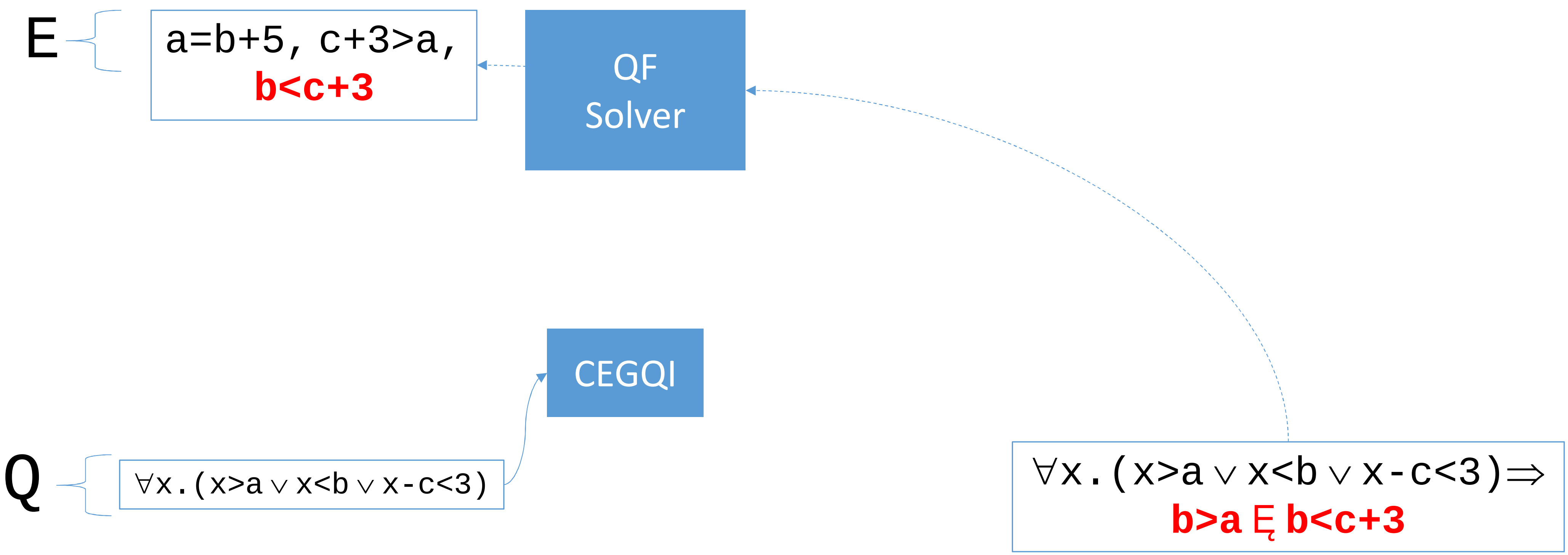
Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

$\forall x.(x>a \vee x<b \vee x-c<3) \Rightarrow$
$b>a \vee b<c+3$

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5, c+3>a,

b<c+3

QF Solver

CEGQI

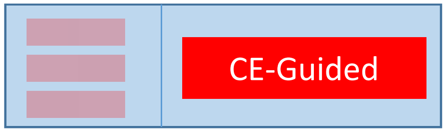
Q

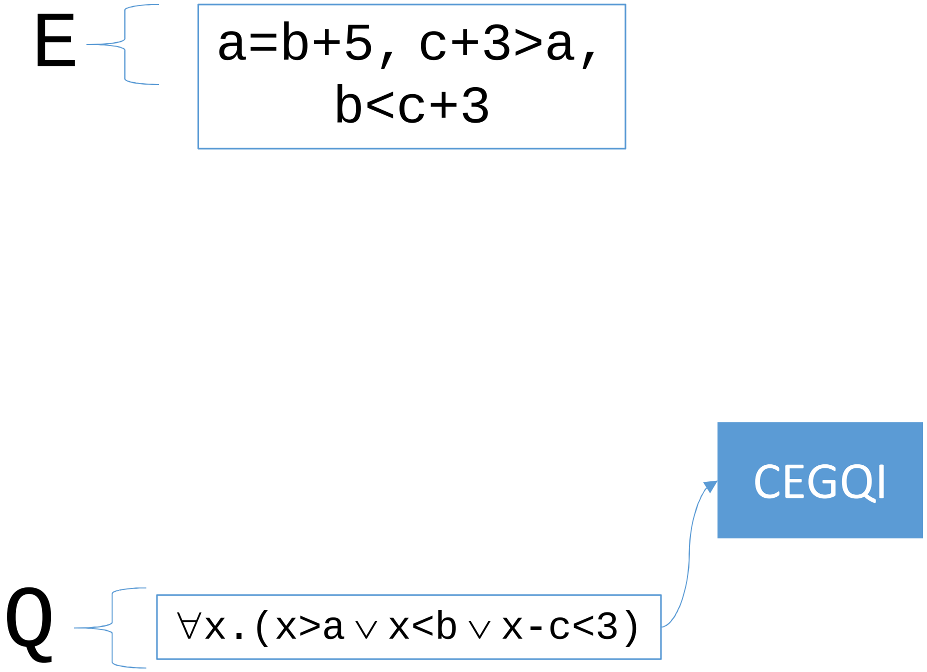
∀x.(x>a ∨ x<b ∨ x-c<3)

∀x.(x>a ∨ x<b ∨ x-c<3)⇒

b>a Ę b<c+3

Counterexample-Guided Instantiation



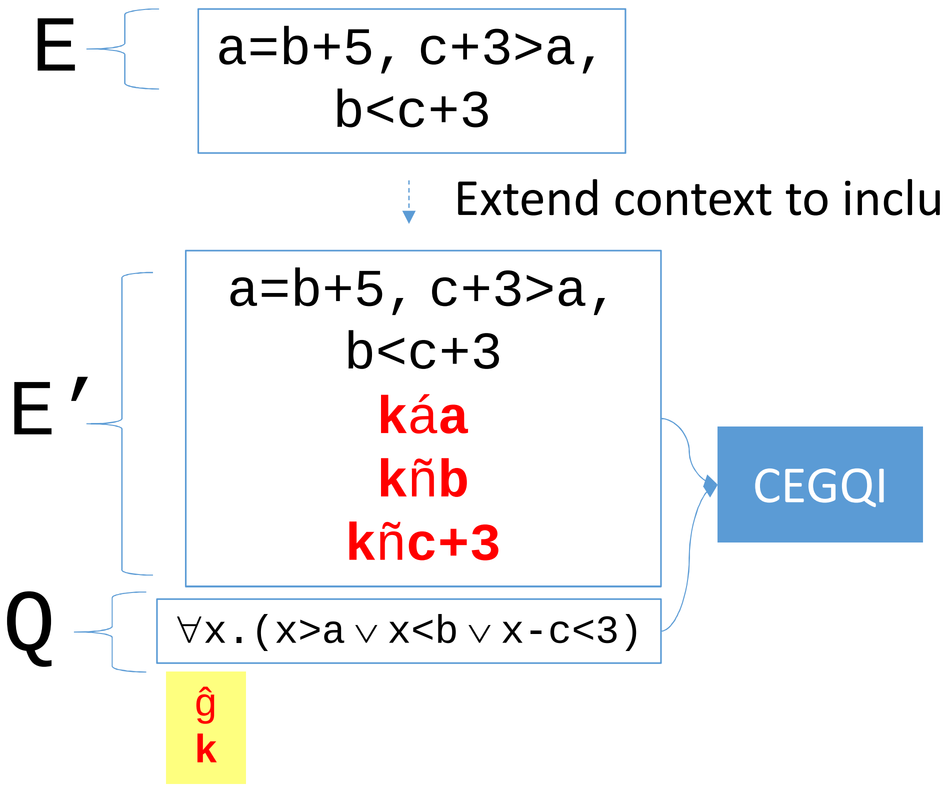


a=b+5, c+3>a, b<c+3

∀x.(x>a ∨ x<b ∨ x-c<3)

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5, c+3>a, b<c+3

Extend context to include counterexample

E'

a=b+5, c+3>a, b<c+3

káa

kñb

kñc+3

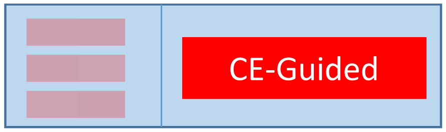
Q

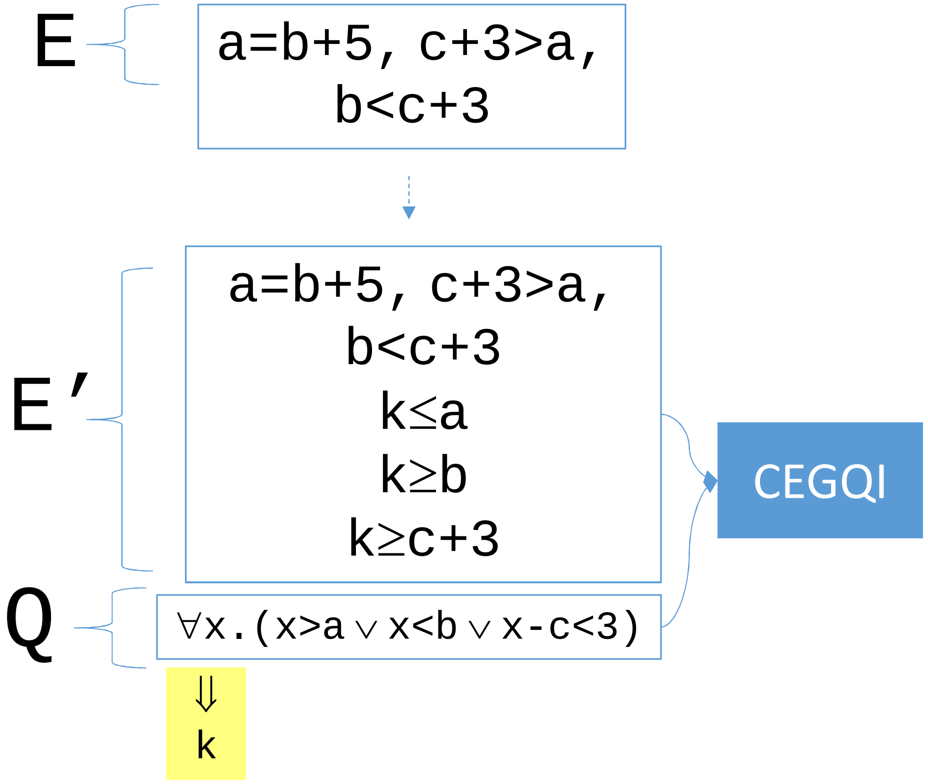
∀x.(x>a ∨ x<b ∨ x-c<3)

ĝ
k

CEGQI

Counterexample-Guided Instantiation





E

$a=b+5,\ c+3>a,$
$b<c+3$

E′

$a=b+5,\ c+3>a,$
$b<c+3$
$k \leq a$
$k \geq b$
$k \geq c+3$

Q

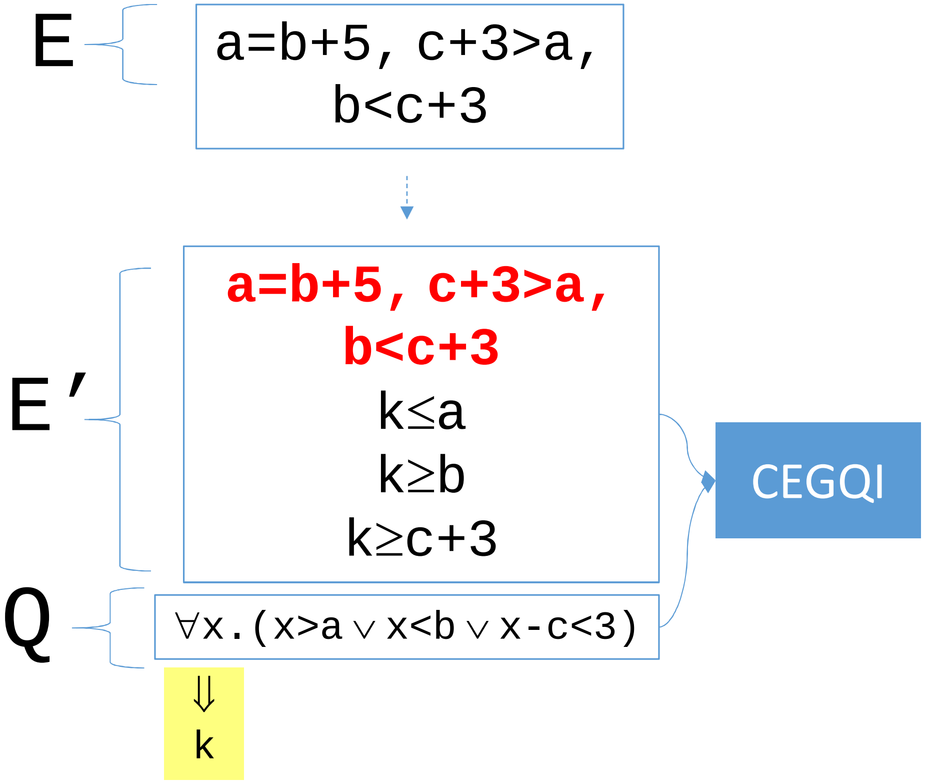
$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$
k

CEGQI

Counterexample-Guided Instantiation

CE-Guided



E

$a=b+5,\ c+3>a,$
$b<c+3$

E′

$a=b+5,\ c+3>a,$
$b<c+3$

$k \leq a$

$k \geq b$

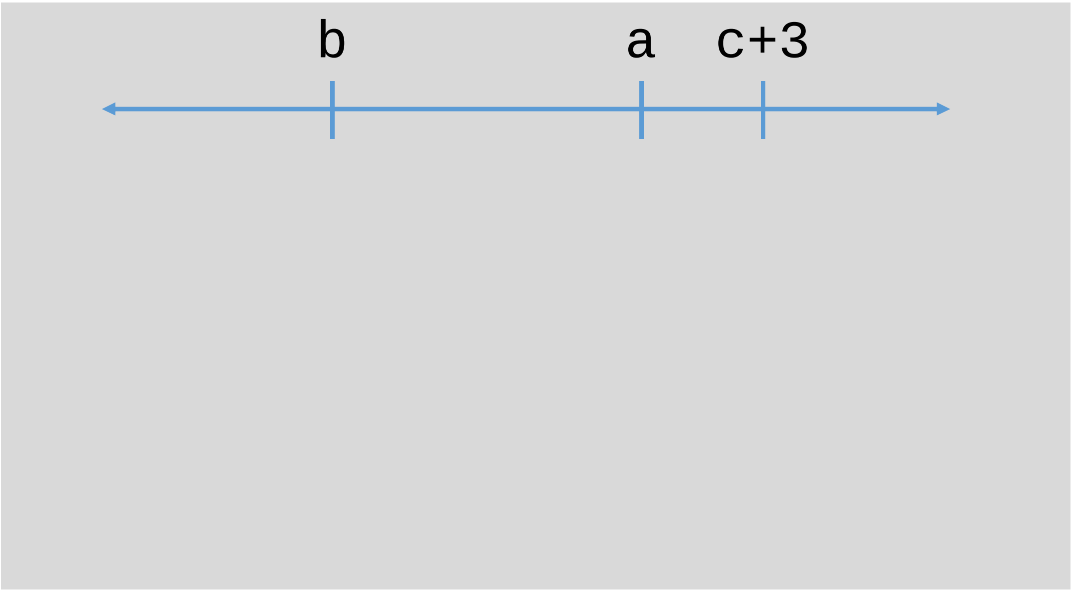
$k \geq c+3$

CEGQI

Q

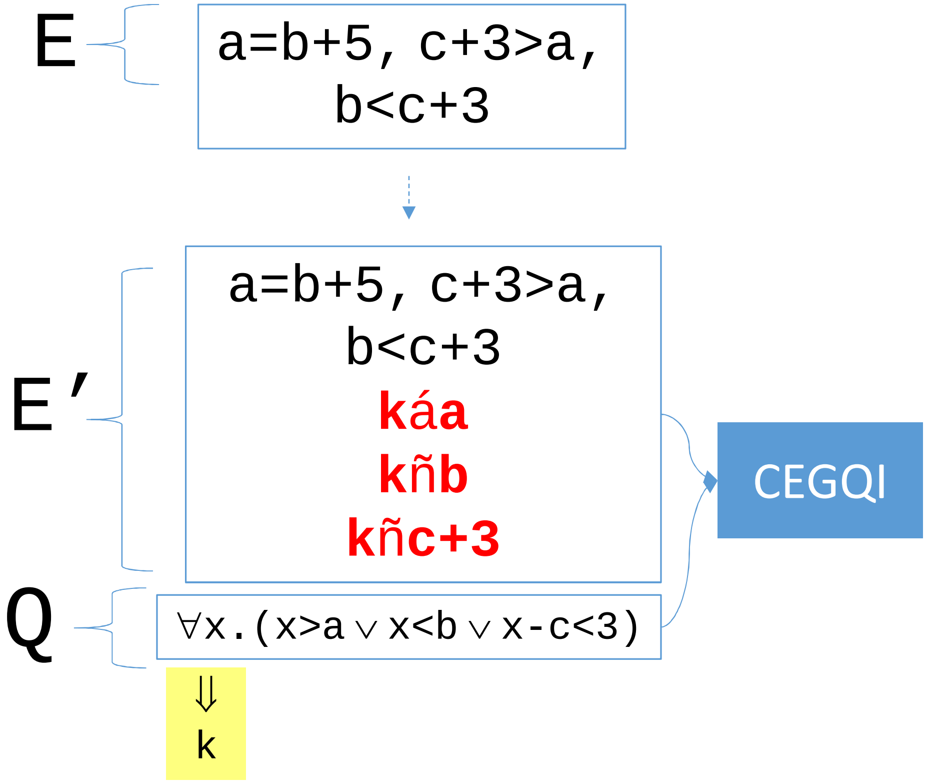
$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$
k



Counterexample-Guided Instantiation

CE-Guided



E

a=b+5, c+3>a,
b<c+3

E′

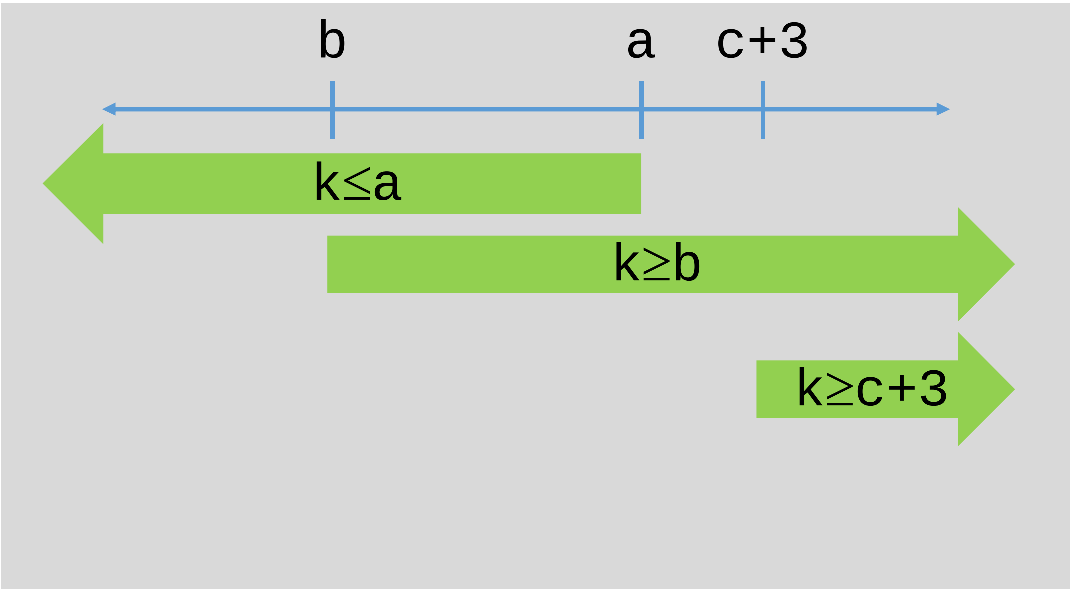
a=b+5, c+3>a,
b<c+3
káa
kñb
kñc+3

Q

∀x.(x>a ∨ x<b ∨ x-c<3)

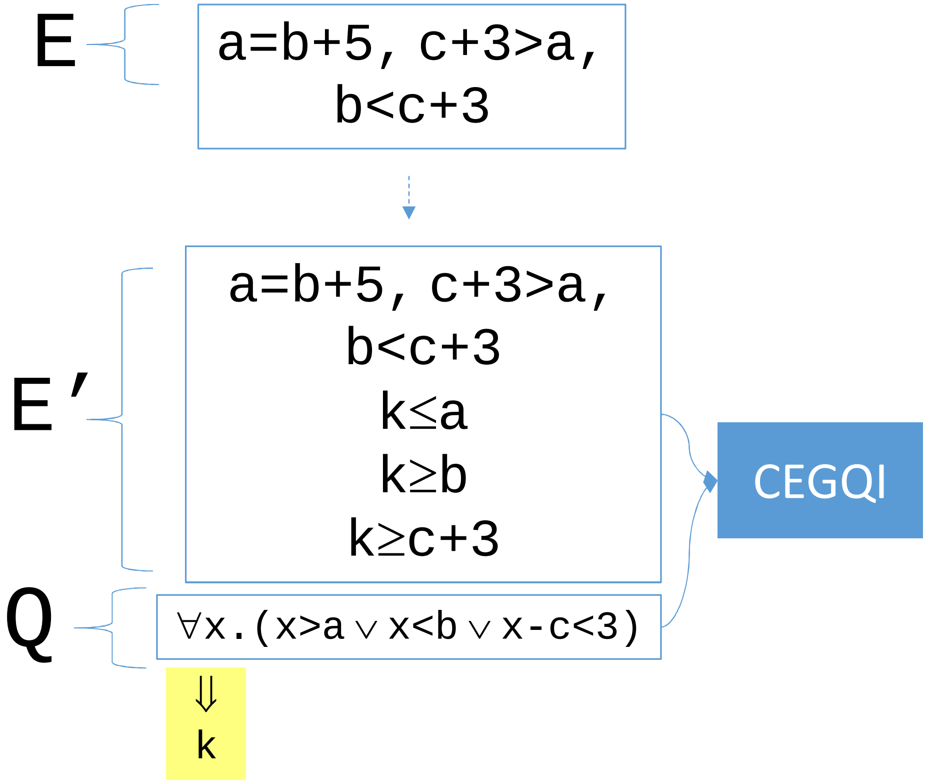
⇓
k

CEGQI



Counterexample-Guided Instantiation

CE-Guided



E

a=b+5, c+3>a,
b<c+3

E′

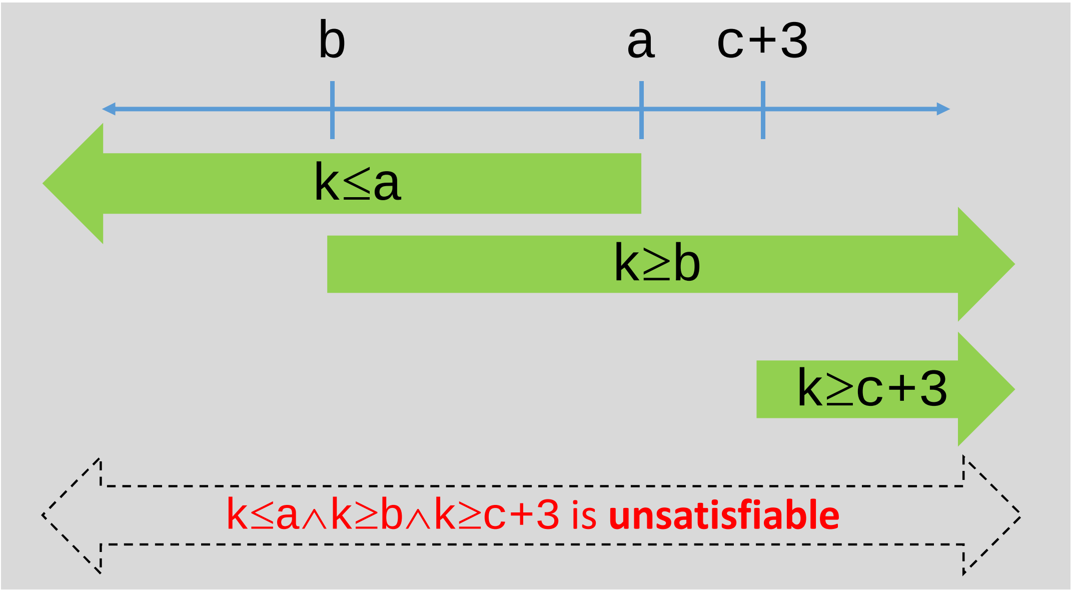
a=b+5, c+3>a,
b<c+3
k≤a
k≥b
k≥c+3

Q

∀x.(x>a ∨ x<b ∨ x-c<3)

⇓
k

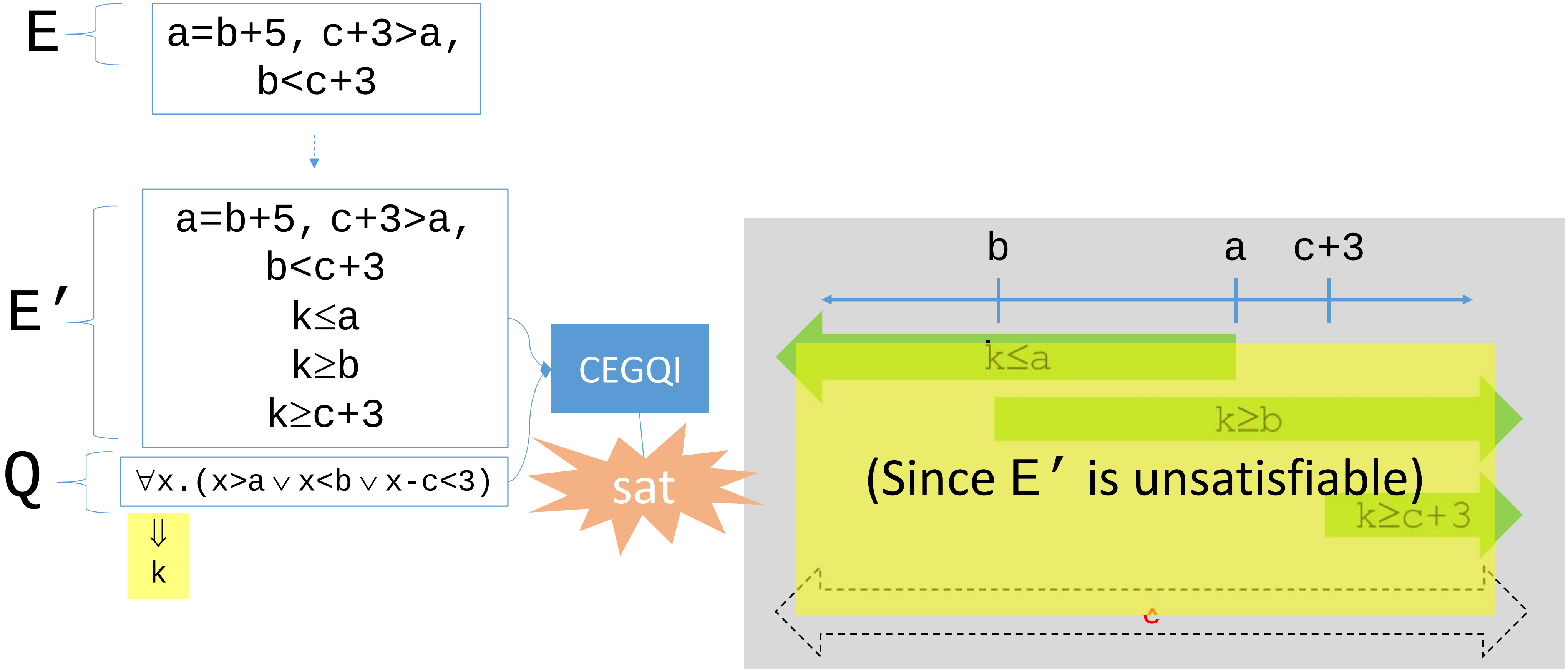
CEGQI



k≤a∧k≥b∧k≥c+3 is **unsatisfiable**

Counterexample-Guided Instantiation

CE-Guided



E: a=b+5, c+3>a, b<c+3

E′: a=b+5, c+3>a, b<c+3, k≤a, k≥b, k≥c+3

Q: ∀x.(x>a ∨ x<b ∨ x-c<3)

⇓ k

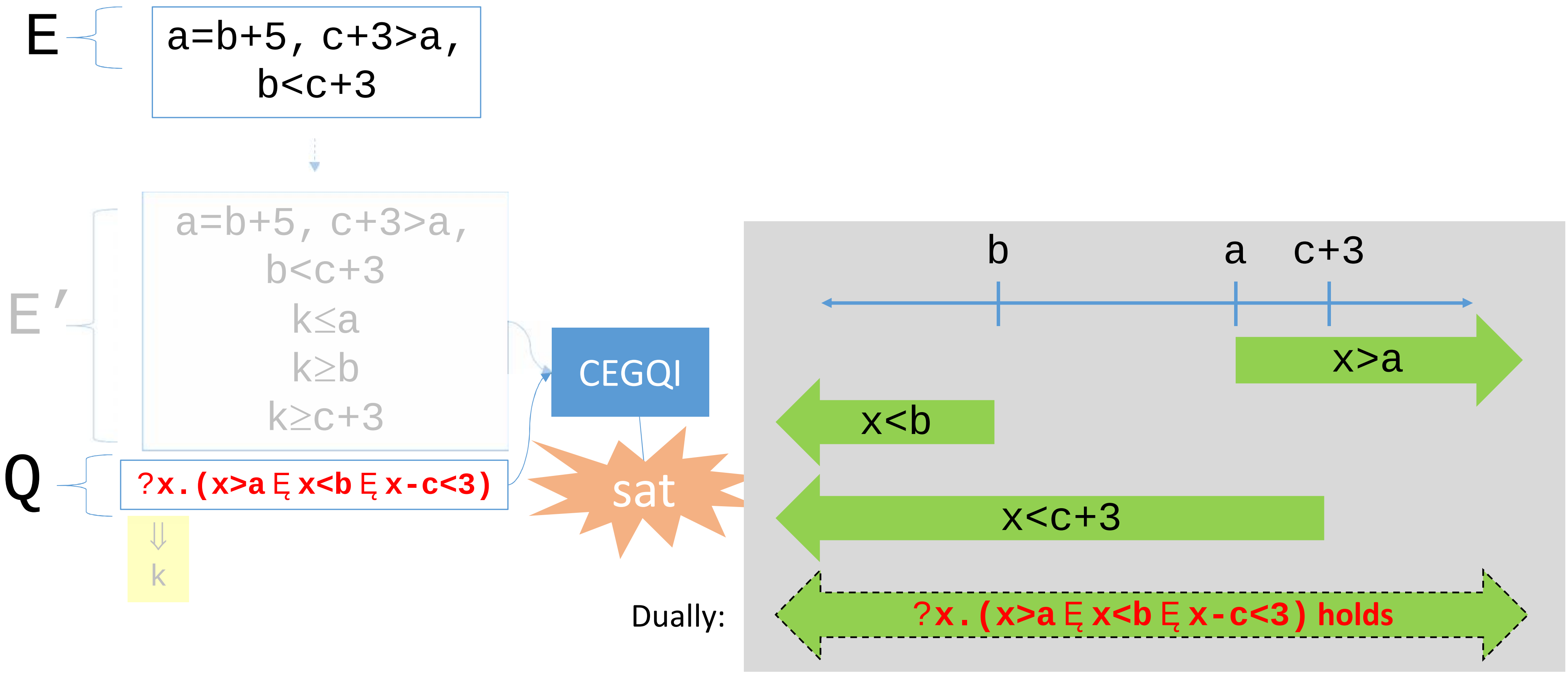
CEGQI

sat

(Since E′ is unsatisfiable)

Counterexample-Guided Instantiation

CE-Guided



E

a=b+5, c+3>a, b<c+3

E′

a=b+5, c+3>a, b<c+3

k≤a

k≥b

k≥c+3

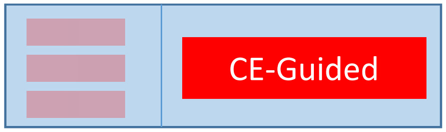
Q

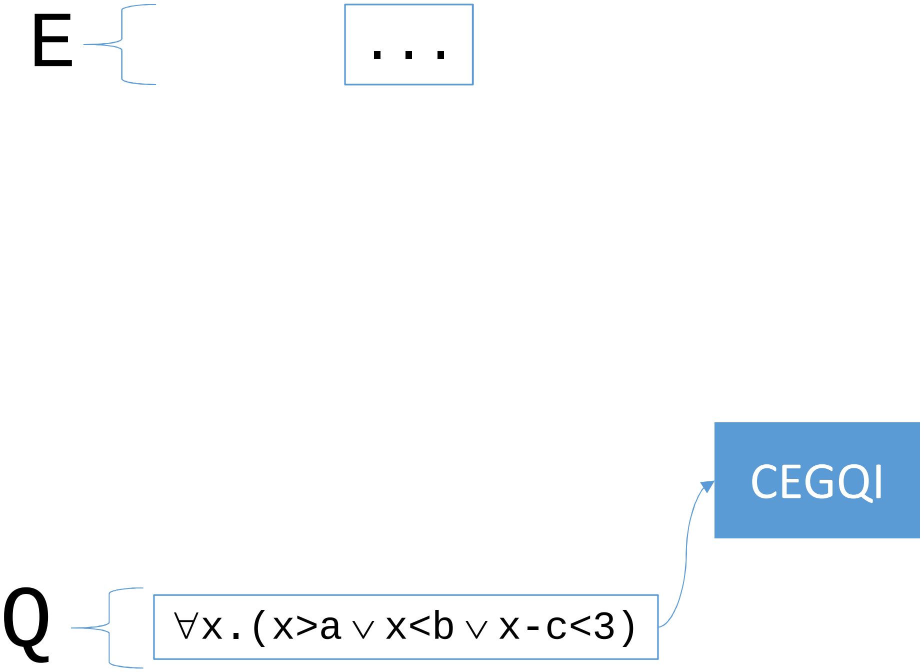
?x.(x>a Ę x<b Ę x-c<3)

⇓ k

Dually: ?x.(x>a Ę x<b Ę x-c<3) holds

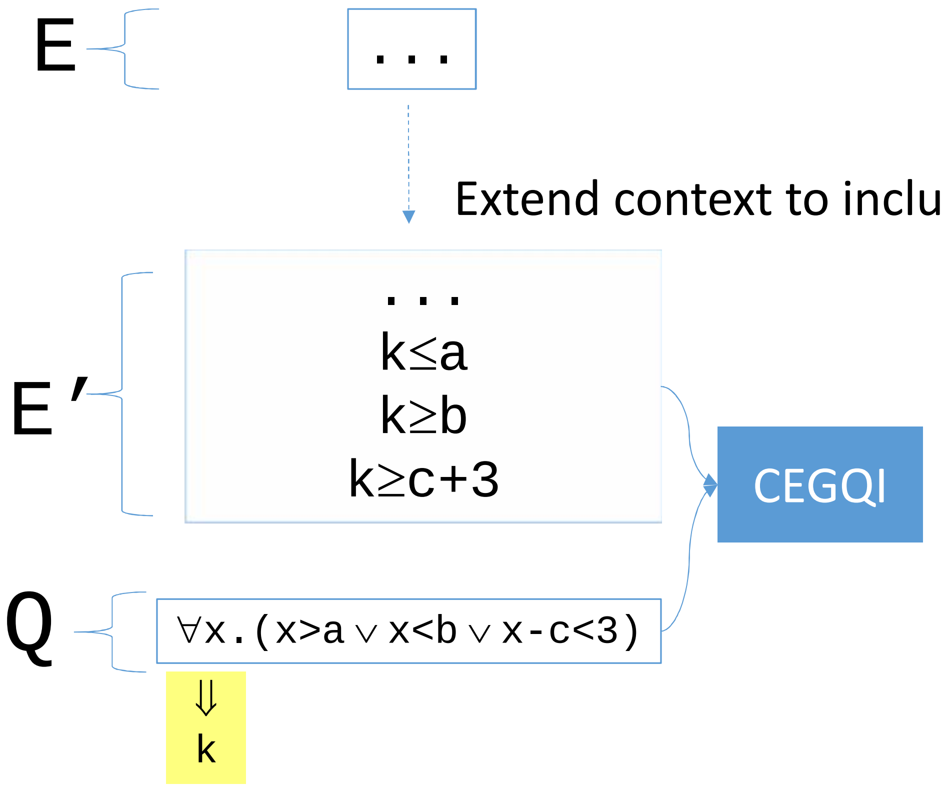
Summary





CE-Guided

Summary



E

...

Extend context to include counterexample

E′

...

$k \leq a$

$k \geq b$

$k \geq c+3$

CEGQI

Q

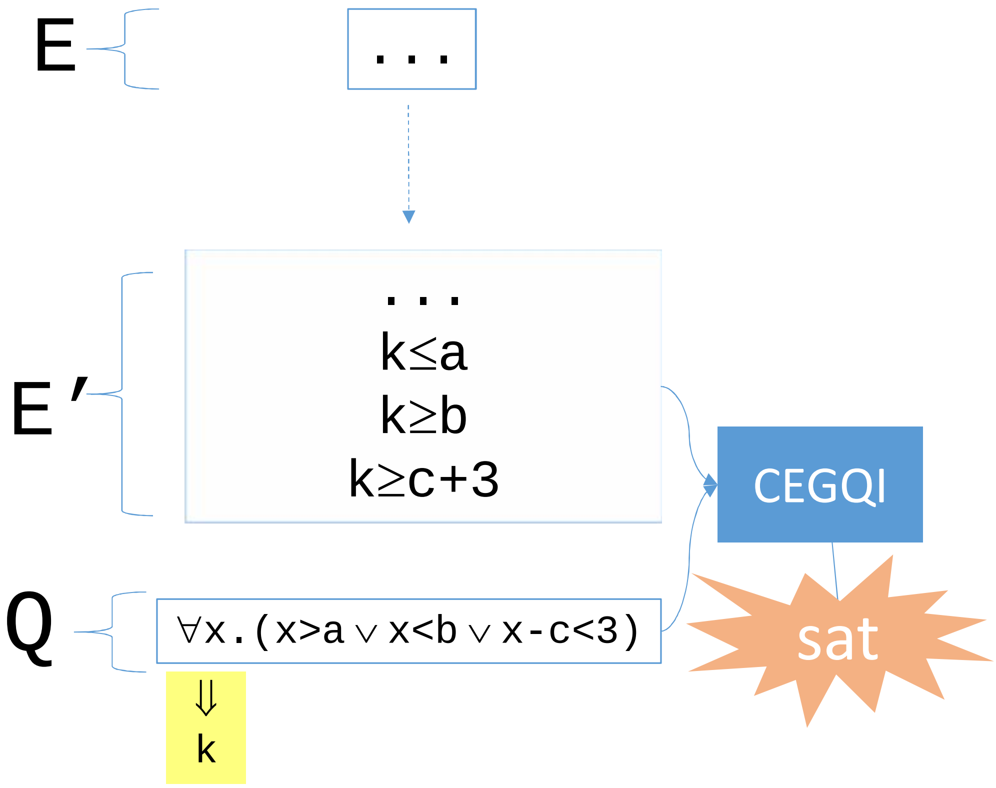
$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$

k

Summary

CE-Guided



E

. . .

E′

. . .

$k \le a$

$k \ge b$

$k \ge c+3$

CEGQI

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

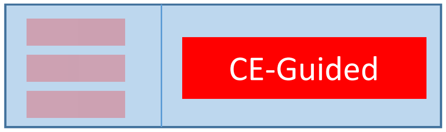
$\Downarrow$

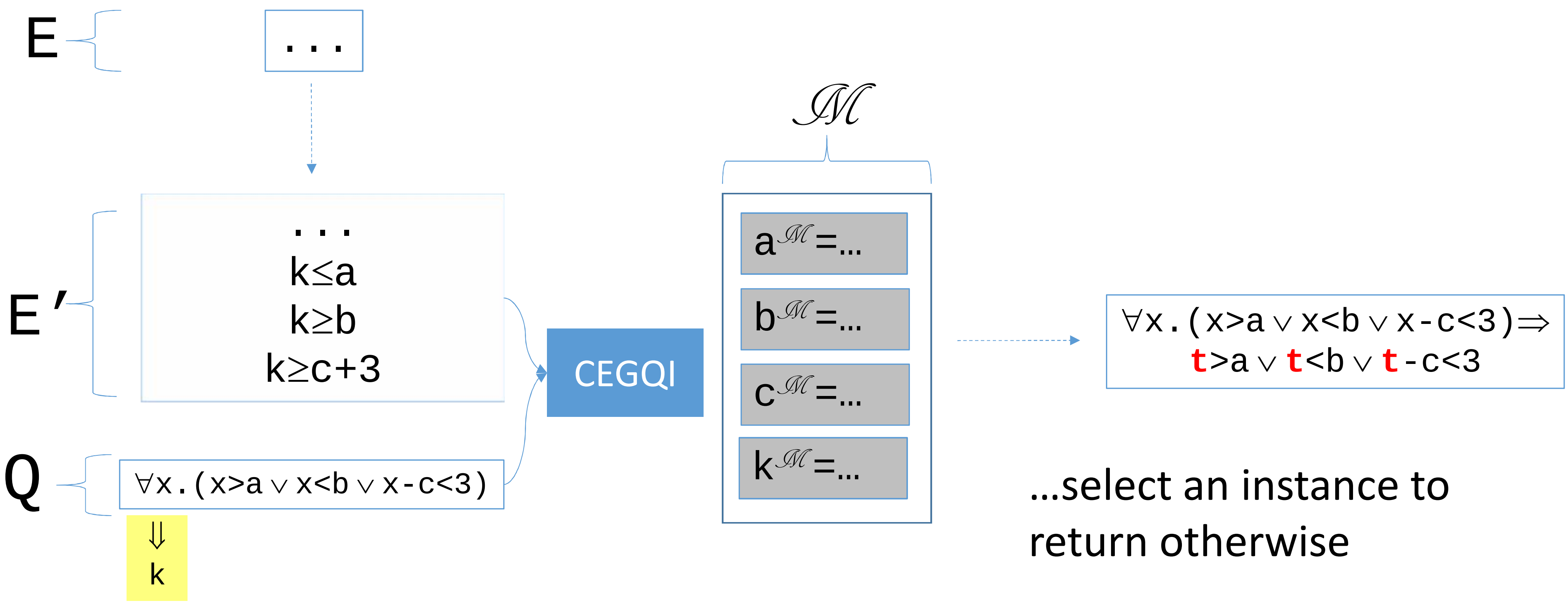
k

sat

…if E′ is unsatisfiable

Summary





E

. . .

E′

. . .

$k \leq a$

$k \geq b$

$k \geq c+3$

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$

k

CEGQI

$\mathscr{M}$

$a^{\mathscr{M}}=\ldots$

$b^{\mathscr{M}}=\ldots$

$c^{\mathscr{M}}=\ldots$

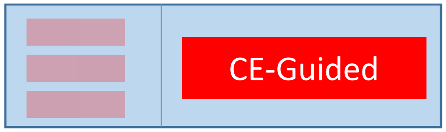
$k^{\mathscr{M}}=\ldots$

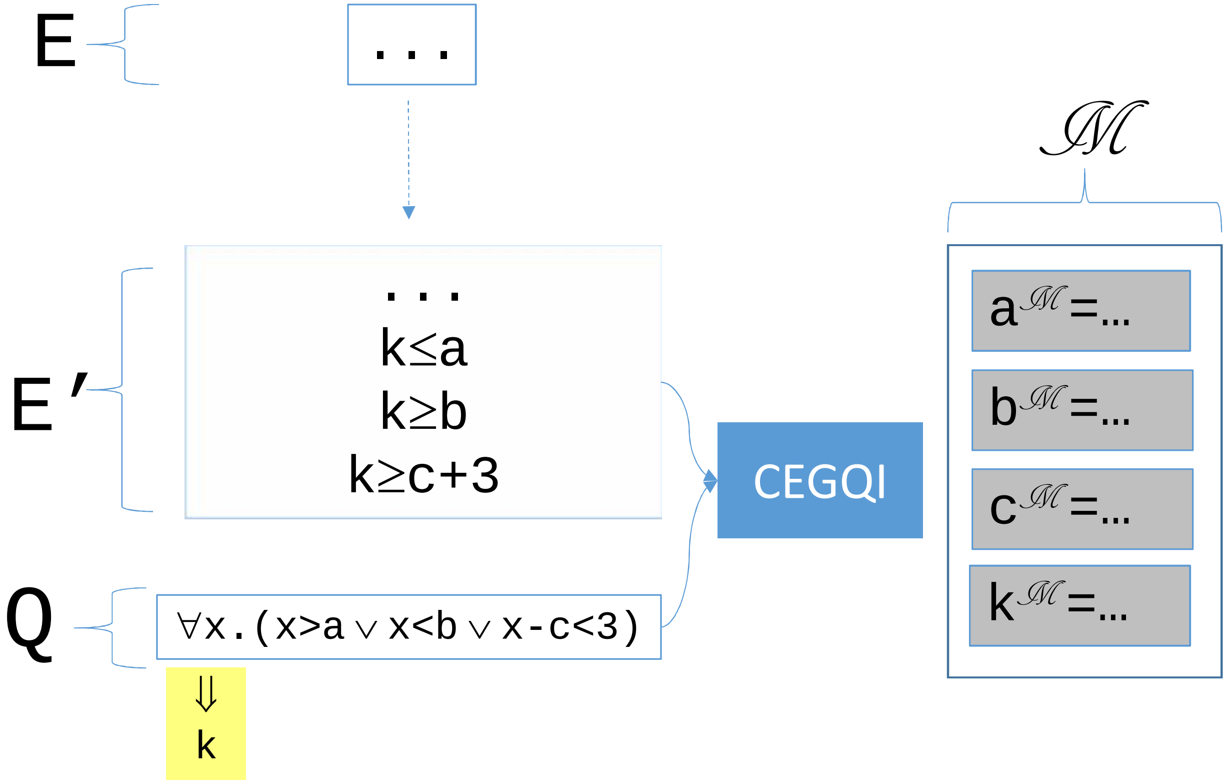
$\forall x.(x>a \vee x<b \vee x-c<3) \Rightarrow$

$\mathbf{t}>a \vee \mathbf{t}<b \vee \mathbf{t}-c<3$

…select an instance to return otherwise

Summary





E

...

E'

...

$k \leq a$

$k \geq b$

$k \geq c+3$

Q

$\forall x.(x>a \vee x<b \vee x-c<3)$

$\Downarrow$

k

CEGQI

$\mathcal{M}$

$a^{\mathcal{M}}=\ldots$

$b^{\mathcal{M}}=\ldots$

$c^{\mathcal{M}}=\ldots$

$k^{\mathcal{M}}=\ldots$

$\forall x.(x>a \vee x<b \vee x-c<3) \Rightarrow$

$\mathbf{t}>a \vee \mathbf{t}<b \vee \mathbf{t}-c<3$

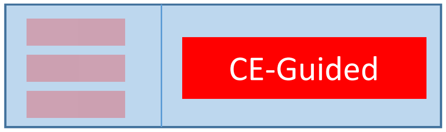
⇒ *In general:*

Requires a

selection function

$Sel_T : E', \mathcal{M}, k \rightarrow t$

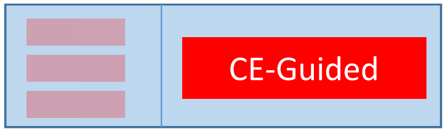
Selection Functions



- Selection function gives decision procedure for ∀ in various theories:
  - Linear real arithmetic (LRA)
    - Maximal lower (minimal upper) bounds $l_1 < k, \ldots, l_n < k \rightarrow \{x \rightarrow l_{max} + \delta\}$
      [Loos+Wiespfenning 93] …*may involve virtual terms* d, ã
    - Interior point method: $l_{max} < k < u_{min} \rightarrow \{x \rightarrow (l_{max} + u_{min})/2\}$
      [Ferrante+Rackoff 79]
  - Linear integer arithmetic (LIA)
    - Maximal lower (minimal upper) bounds (+c) $l_1 < k, \ldots, l_n < k \rightarrow \{x \rightarrow l_{max} + c\}$
      [Cooper 72]
  - Bitvectors/finite domains
    - Value instantiations … $\rightarrow \{x \rightarrow k^{\mathscr{M}}\}$
  - Datatypes, …

Ĝ ***Termination* argument for each**: *enumerate at most a finite number of instances*

Current Work



- In current work [Reynolds/King/Kuncak FMSD 2017]
  - Finite selection functions for LRA, LIA, LIRA
  - Extension to arbitrary quantifier alternations

  - Instantiation can be used for quantifier elimination:

CEGQI terminates with $\psi[t_1] \wedge \ldots \wedge \psi[t_n] \Rightarrow$

$\exists x.\neg\psi[x]$ is equivalent to $\neg\psi[t_1] \vee \ldots \vee \neg\psi[t_n]$

  - Instantiation can be used as basis of synthesis procedures:

CEGQI finds $\psi[t_1] \wedge \ldots \wedge \psi[t_n]$ is unsat $\Rightarrow$

$\lambda x.\mathtt{ite}(\psi[t_1], t_1, \ldots, \mathtt{ite}(\psi[t_{n-1}], t_{n-1}, t_n)\ldots)$ is a solution for f in $\forall x.\psi[f(x)]$

$\Rightarrow$Used in CVC4's synthesis solver [Reynolds et al CAV 2015]

Applications / Examples

Contract-Based Verification : Unfolding

```
int len(List x){
  if(is-nil(x)){
    return 0;
  }else{
    return 1+len(tail(x))
  }
}
```

```
List append(List x, List y){
  …
}
@ensures len(@ret)=len(x_in)+len(y_in)
```

```
List shift(List x){
  if(is-nil(x)){
    return x;
  }else{
    return append(tail(x),cons(head(x),nil));
  }
}
@ensures len(@ret)=len(x_in) ?
```

EXAMPLE A1…

Contract-Based Verification : Unfolding

```
int len(List x){
  if(is-nil(x)){
    return 0;
  }else{
    return 1+len(tail(x))
  }
}
```

```
List append(List x, List y){
  …
}
@ensures len(@ret)=len(x_in)+len(y_in)
```

```
List shift(List x){
  if(is-nil(x)){
    return x;
  }else{
    return append(tail(x),cons(head(x),nil));
  }
}
@ensures len(@ret)=len(x_in)
```

Contract-Based Verification : Unfolding

```
∀x.len(x)=ite(is-nil(x),0,1+len(tail(x)))
∀xy.len(append(x,y))=len(x)+len(y)
∀x.shift(x)=ite(is-nil(x),nil,append(tail(x),cons(head(x),nil)))
∃k.len(shift(k))≠len(k)
```

Contract-Based Verification : Unfolding

```
∀x.len(x)=ite(is-nil(x),0,1+len(tail(x)))
∀xy.len(append(x,y))=len(x)+len(y)
∀x.shift(x)=ite(is-nil(x),nil,append(tail(x),cons(head(x),nil)))
len(shift(k))≠len(k)
```

(Skolemize)

Contract-Based Verification : Unfolding

```
∀x.len(x)=ite(is-nil(x),0,1+len(tail(x)))
∀xy.len(append(x,y))=len(x)+len(y)
∀x.shift(x)=ite(is-nil(x),nil,append(tail(x),cons(head(x),nil)))
len(shift(k))≠len(k)
```

if `is-nil(k)`…

Contract-Based Verification : Unfolding

`∀x.len(x)=ite(is-nil(x),0,1+len(tail(x)))`

`∀xy.len(append(x,y))=len(x)+len(y)`

`∀x.shift(x)=ite(is-nil(x),nil,append(tail(x),cons(head(x),nil)))`

`len(shift(k))≠len(k)`

(E-Matching, shift) `||` `||` (E-Matching, len)

`len(nil)` `0`

(E-Matching, len) `||`

`0`

if `is-nil(k)`…

Contract-Based Verification : Unfolding

```
∀x.len(x)=ite(is-nil(x),0,1+len(tail(x)))
∀xy.len(append(x,y))=len(x)+len(y)
∀x.shift(x)=ite(is-nil(x),nil,append(tail(x),cons(head(x),nil)))
```

`len(shift(k))` ≠ `len(k)`

if `is-cons(k)`…

Contract-Based Verification : Unfolding

```
∀x.len(x)=ite(is-nil(x),0,1+len(tail(x)))
∀xy.len(append(x,y))=len(x)+len(y)
∀x.shift(x)=ite(is-nil(x),nil,append(tail(x),cons(head(x),nil)))
```

`len(shift(k))` ≠ `len(k)`

(E-Matching, shift) || || (E-Matching, len)

`len(append(tail(k),cons(head(k),nil))`   `1+len(tail(k))`

(E-Matching, append) ||

`len(tail(k))+len(cons(head(k),nil))`

(E-Matching, len) ||

`len(tail(k))+1+len(nil)`

(E-Matching, len) ||

`len(tail(k))+1`

if `is-cons(k)`…

Contract-Based Verification : Recursion

```
@precondition: x≥0
int sum(int x)
{
     if( x==0 ){
       return 0;
     }else{
       return x+sum(x-1);
     }
}
@ensures: @ret≥0 ?
```

EXAMPLE A2…

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
       ∃k.k≥0∧¬sum(k)≥0
```

Contract-Based Verification : Recursion

$\forall$x.sum(x)=ite(x=0,0,x+sum(x-1))

k$\geq$0$\wedge$sum(k)<0

(Skolemize, simplify)

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
          k≥0∧sum(k)<0
  sum(k)=ite(k=0,0,k+sum(k-1))        (E-Matching)
```

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
         k≥0∧sum(k)<0
 sum(k)=ite(k=0,0,k+sum(k-1))
```

$$\overbrace{\boxed{\begin{array}{c} \texttt{k}^{\mathscr{M}}\texttt{=1} \\ \texttt{sum(k)}^{\mathscr{M}}\texttt{=-1} \\ \texttt{sum(k-1)}^{\mathscr{M}}\texttt{=-2} \end{array}}}^{\mathscr{M}}$$

Contract-Based Verification : Recursion

```
  ∀x.sum(x)=ite(x=0,0,x+sum(x-1))
            k≥0∧sum(k)<0
    sum(k)=ite(k=0,0,k+sum(k-1))
sum(k-1)=ite((k-1)=0,0,(k-1)+sum((k-1)-1))      (E-Matching)
```

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
         k≥0∧sum(k)<0
  sum(k)=ite(k=0,0,k+sum(k-1))
 sum(k-1)=ite(k=1,0,(k-1)+sum(k-2))
```

(simplify)

Contract-Based Verification : Recursion

$$\forall x.\mathtt{sum}(x)=\mathtt{ite}(x=0,0,x+\mathtt{sum}(x-1))$$

$$k\geq 0\wedge \mathtt{sum}(k)<0$$

$$\mathtt{sum}(k)=\mathtt{ite}(k=0,0,k+\mathtt{sum}(k-1))$$

$$\mathtt{sum}(k-1)=\mathtt{ite}(k=1,0,(k-1)+\mathtt{sum}(k-2))$$

$$\overbrace{\boxed{\begin{array}{c} k^{\mathscr{M}}=2 \\ \mathtt{sum}(k)^{\mathscr{M}}=-1 \\ \mathtt{sum}(k-1)^{\mathscr{M}}=-3 \\ \mathtt{sum}(k-2)^{\mathscr{M}}=-4 \end{array}}}^{\mathscr{M}}$$

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
          k≥0∧sum(k)<0
  sum(k)=ite(k=0,0,k+sum(k-1))
  sum(k-1)=ite(k=1,0,(k-1)+sum(k-2))
sum(k-2)=ite((k-2)=0,0,(k-2)+sum((k-2)-1))
```

(E-Matching)

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
         k≥0∧sum(k)<0
 sum(k)=ite(k=0,0,k+sum(k-1))
 sum(k-1)=ite(k=1,0,(k-1)+sum(k-2))
 sum(k-2)=ite(k=2,0,(k-2)+sum(k-3))
```

(simplify)

Contract-Based Verification : Recursion

$$\forall x.\mathtt{sum}(x)=\mathtt{ite}(x=0,0,x+\mathtt{sum}(x-1))$$

$$k\geq 0\wedge \mathtt{sum}(k)<0$$

$$\mathtt{sum}(k)=\mathtt{ite}(k=0,0,k+\mathtt{sum}(k-1))$$

$$\mathtt{sum}(k-1)=\mathtt{ite}(k=1,0,(k-1)+\mathtt{sum}(k-2))$$

$$\mathtt{sum}(k-2)=\mathtt{ite}(k=2,0,(k-2)+\mathtt{sum}(k-3))$$

$\mathscr{M}$

$$k^{\mathscr{M}}=3$$

$$\mathtt{sum}(k)^{\mathscr{M}}=-1$$

$$\mathtt{sum}(k-1)^{\mathscr{M}}=-4$$

$$\mathtt{sum}(k-2)^{\mathscr{M}}=-6$$

$$\mathtt{sum}(k-3)^{\mathscr{M}}=-7$$

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
         k≥0∧sum(k)<0
 sum(k)=ite(k=0,0,k+sum(k-1))
 sum(k-1)=ite(k=1,0,(k-1)+sum(k-2))
 sum(k-2)=ite(k=2,0,(k-2)+sum(k-3))
```

…and repeat ad infinitum

Solution: Inductive Strengthening

- Given negated conjecture:

$$\exists k. k \geq 0 \land sum(k) < 0$$

- Assume k is the *smallest* CE to property:

$$\begin{gathered} k \geq 0 \land sum(k) < 0 \land \\ 0 \leq (k-1) \Rightarrow \neg((k-1) \geq 0 \land sum(k-1) < 0) \end{gathered}$$

Weak induction

$$\begin{gathered} k \geq 0 \land sum(k) < 0 \land \\ \forall k'.(0 \leq k' < k \Rightarrow \neg(k' \geq 0 \land sum(k') < 0)) \end{gathered}$$

Strong induction

Skolemization with Inductive Strengthening

- General form:

$$\boxed{\forall x. \neg P(x) \vee (P(k) \wedge \forall y. (y < k \Rightarrow \neg P(y)))}$$

  - For well-founded relation “<“
- Extends for multiple variables
- Common examples of “<“ in SMT:
  - (Weak) structural induction on inductive datatypes
    - Assume property holds for direct children of k of same type
  - (Weak) well-founded induction on integers
    - Assume property holds for (k-1), with base case 0

Contract-Based Verification : Induction

```
@precondition: x_in≥0
int sum(int x)
{
     if( x==0 ){
       return 0;
     }else{
       return x+sum(x-1);
     }
}
@ensures: @ret≥0 ?
```

…requires *induction!*

EXAMPLE A2-ind…

Contract-Based Verification : Induction

```
@precondition: x_in≥0
int sum(int x)
{
     if( x==0 ){
       return 0;
     }else{
       return x+sum(x-1);
     }
}
@ensures: @retñ0
```

…by well-founded induction on (positive) integers

- Can be automated by SMT solver

Contract-Based Verification : Induction

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
         k≥0∧sum(k)<0
```

Contract-Based Verification : Induction

$$\forall x.\mathtt{sum}(x)=\mathtt{ite}(x=0,0,x+\mathtt{sum}(x-1))$$

$$k\geq 0\wedge \mathtt{sum}(k)<0$$

$$0\leq(k-1)\Rightarrow\neg((k-1)\geq 0\wedge \mathtt{sum}(k-1)<0)$$

(strengthen)

Contract-Based Verification : Induction

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
         k≥0∧sum(k)<0
      (k-1)<0∨sum(k-1)≥0                         (simplify)
```

Contract-Based Verification : Induction

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
          k≥0∧sum(k)<0
       (k-1)<0∨sum(k-1)≥0
  sum(k)=ite(k=0,0,k+sum(k-1))              (E-matching)
```

Contract-Based Verification : Recursion

```
∀x.sum(x)=ite(x=0,0,x+sum(x-1))
         k≥0∧sum(k)<0
      (k-1)<0∨sum(k-1)≥0
  sum(k)=ite(k=0,0,k+sum(k-1))
```

unsat

…since

when k=0
 sum(k)<0, and
 sum(k)=0

when k>0,
 sum(k)<0, and
 sum(k)=k+sum(k-1)≥k>0

Contract-Based Verification : Recursion

```
@precondition: x_in≥0
int sum(int x)
{
     if( x==0 ){
       return 0;
     }else{
       return x+sum(x-1);
     }
}
@ensures: @ret<100 ?
```

EXAMPLE A3…

Contract-Based Verification : Recursion

```
@precondition: x_in≥0
int sum(int x)
{
     if( x==0 ){
       return 0;
     }else{
       return x+sum(x-1);
     }
}
@ensures: @ret<100 ?
```

…this conjecture does not hold

- Need (finite) model finding techniques to show “sat”

Finite Model Finding in **CVC4**

- Finite Model-complete method for finite/uninterpreted ∀

$$\forall xy{:}\underbrace{\mathbf{U}.}(x\neq y\Rightarrow f(x)\neq f(y)) \land a\neq b$$

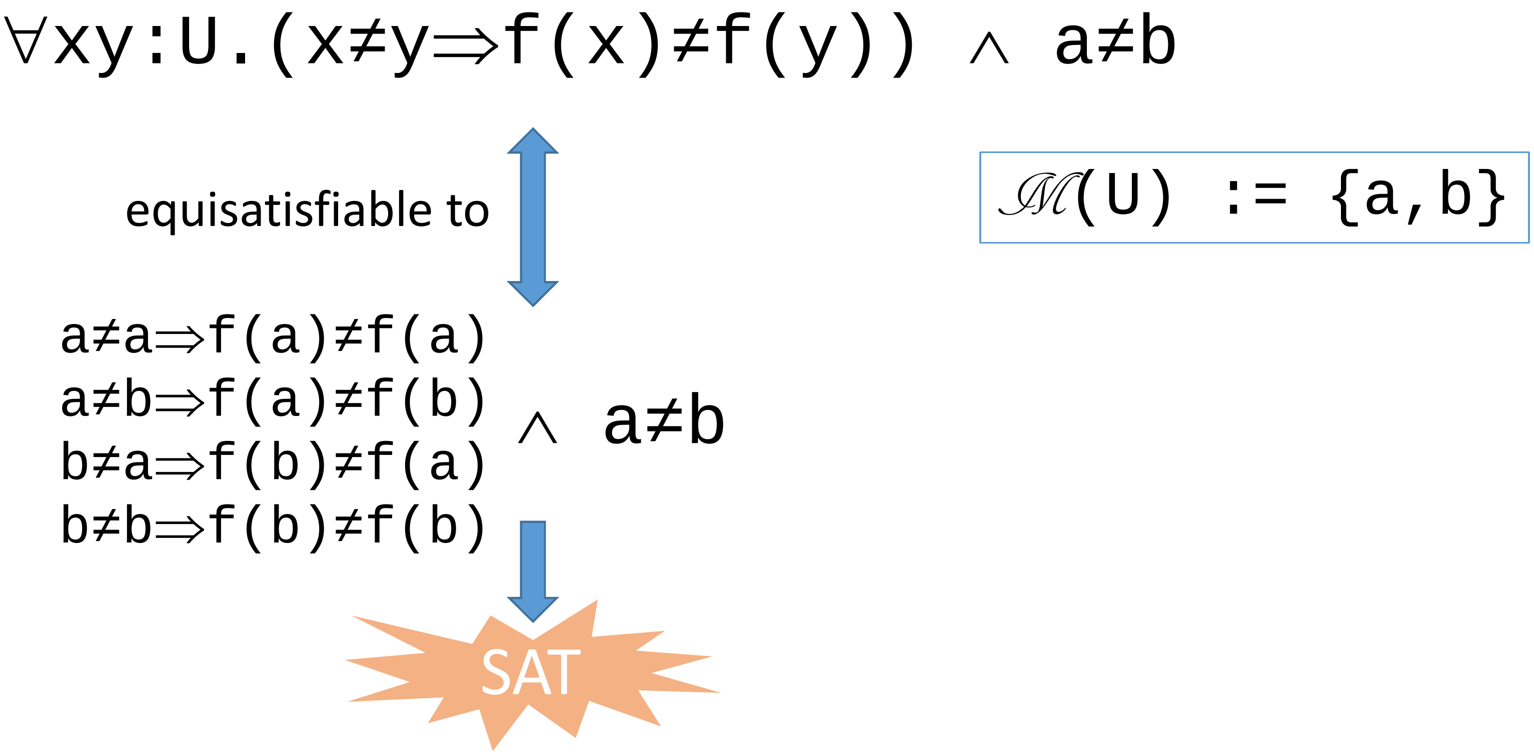
All variables have finite/uninterpreted sort **U**

Finite Model Finding in **CVC4**

$$\forall xy{:}U.(x \neq y \Rightarrow f(x) \neq f(y)) \;\wedge\; a \neq b$$

$$\underbrace{\boxed{\mathscr{M}(U) := \{a,b\}}}_{\text{Model interprets U as the set } \mathscr{M}(U)=\{a,b\}}$$

Finite Model Finding in **CVC4**



$\forall xy{:}U.(x \neq y \Rightarrow f(x) \neq f(y)) \wedge a \neq b$

equisatisfiable to

$\mathscr{M}(U) := \{a,b\}$

$$\begin{array}{l} a \neq a \Rightarrow f(a) \neq f(a) \\ a \neq b \Rightarrow f(a) \neq f(b) \\ b \neq a \Rightarrow f(b) \neq f(a) \\ b \neq b \Rightarrow f(b) \neq f(b) \end{array} \wedge a \neq b$$

SAT

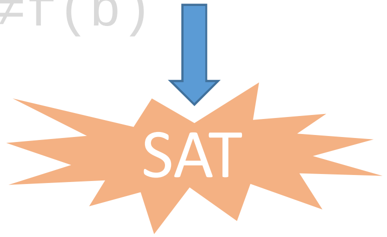
Finite Model Finding in **CVC4**

$\forall xy{:}U.(x\neq y\Rightarrow f(x)\neq f(y)) \land a\neq b$

equisatisfiable to

$\mathscr{M}(U) := \{a,b\}$

$a\neq a\Rightarrow f(a)\neq f(a)$
$a\neq b\Rightarrow f(a)\neq f(b)$ $\land$ $a\neq b$
$b\neq a\Rightarrow f(b)\neq f(a)$
$b\neq b\Rightarrow f(b)\neq f(b)$



⇒ Can be accelerated by model-based quantifier instantiation

For details, see **[Reynolds et al CADE2013]**

…Fails on most Recursive Function Definitions!

- Example:

  $\forall x:\mathbf{Int}.(sum(x)=ite(x\leq 0,0,sum(x-1)+x)) \wedge sum(x_{in})>100$

- Finite Model Finding:
  - Fails, since quantification is over infinite type **Int**

    $\mathscr{M}(Int)=\{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$

…Fails on most Recursive Function Definitions!

- Example:

$$\forall x{:}\mathbf{Int}.(\mathtt{sum}(x)=\mathtt{ite}(x\leq 0,0,\mathtt{sum}(x-1)+x)) \wedge \mathtt{sum}(x_{in})>100$$

$\mathcal{M}$(Int)={…, -3, -2, -1, 0, 1, 2, 3, …}

Impossible

$$\begin{array}{ll}
(\quad \ldots & \wedge \\
(\mathtt{sum}(1)=\mathtt{ite}(1\leq 0,0,\mathtt{sum}(1-1)+1)) & \wedge \\
(\mathtt{sum}(0)=\mathtt{ite}(0\leq 0,0,\mathtt{sum}(0-1)+0)) & \wedge \\
(\mathtt{sum}(-1)=\mathtt{ite}(-1\leq 0,0,\mathtt{sum}(-1-1)+-1)) & \wedge \\
\quad \ldots & \wedge \ \mathtt{sum}(x_{in})>100
\end{array}$$

Model Finding for Recursive Functions [Reynolds et al 2016]

```
∀x:Int.ite(x≤0,
          sum(x)=0,
          sum(x)=sum(x-1)+x))∧
sum(x_in)>100
```

Model Finding for Recursive Functions [Reynolds et al 2016]

```
∀x:a.ite(g(x)≤0,
          sum(g(x))=0,
          sum(g(x))=sum(g(x)-1)+g(x))∧
sum(x_in)>100
```

- Introduce uninterpreted sort a
  - Conceptually, α represents the set of relevant arguments of f
    - Restrict the domain of function definition quantification to a
- Introduce uninterpreted function g: α →Int
  - Maps between abstract and concrete domains

Model Finding for Recursive Functions [Reynolds et al 2016]

```
∀x:a.ite(g(x)≤0,
          sum(g(x))=0,
          sum(g(x))=sum(g(x)-1)+g(x)∧(Az:a.g(z)=g(x)-1))∧
sum(x_in)>100 ∧ (Az:a.g(z)=x_in)
```

- Add appropriate constraints regarding a, g
  - Each relevant concrete value must be mapped to by some abstract value

Model Finding for Recursive Functions [Reynolds et al 2016]

$$\begin{aligned}
&\forall x{:}a.\mathtt{ite}(\gamma(x)\le 0,\\
&\qquad\qquad \mathtt{sum}(\gamma(x))=0,\\
&\qquad\qquad \mathtt{sum}(\gamma(x))=\mathtt{sum}(\gamma(x)-1)+\gamma(x)\wedge(\exists z{:}\alpha.\gamma(z)=\gamma(x)-1))\wedge\\
&\mathtt{sum}(x_{in})>100\ \wedge\ (\exists z{:}\alpha.\gamma(z)=x_{in})
\end{aligned}$$

- ∀ is over finite/uninterpreted sorts
  - ⇒ **CVC4** (finite model finding) finds model for this benchmark in <1 second

Model Finding for Recursive Functions [Reynolds et al 2016]

$$\begin{array}{l}
\forall x{:}\alpha.\mathtt{ite}(\gamma(x)\leq 0,\\
\qquad\qquad \mathtt{sum}(\gamma(x))=0,\\
\qquad\qquad \mathtt{sum}(\gamma(x))=\mathtt{sum}(\gamma(x)-1)+\gamma(x)\wedge(\exists z{:}\alpha.\gamma(z)=\gamma(x)-1))\wedge\\
\mathtt{sum}(\mathbf{x_{in}})>100\ \wedge\ (\exists z{:}\alpha.\gamma(z)=x_{in})
\end{array}$$

- Formula is satisfied by a model $\mathscr{M}$ where:
  - $\mathscr{M}(\mathbf{x_{in}}) := \mathbf{14}$
  - $\mathscr{M}(f) := \lambda x.\mathtt{ite}(x=14,105,\mathtt{ite}(x=13,91,\ldots\ \mathtt{ite}(x=1,1,0)\ldots))$

Model Finding for Recursive Functions [Reynolds et al 2016]

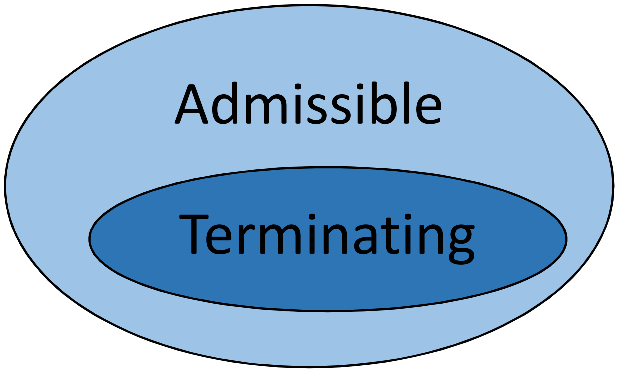
$$
\begin{aligned}
&\forall x{:}\alpha.\,\mathtt{ite}(\gamma(x)\le 0,\\
&\qquad\qquad \mathtt{sum}(\gamma(x))=0,\\
&\qquad\qquad \mathtt{sum}(\gamma(x))=\mathtt{sum}(\gamma(x)-1)+\gamma(x)\wedge(\exists z{:}\alpha.\gamma(z)=\gamma(x)-1))\wedge\\
&\mathtt{sum}(\mathbf{x}_{\mathbf{in}})>100\ \wedge\ (\exists z{:}\alpha.\gamma(z)=x_{in})
\end{aligned}
$$

- Formula is satisfied by a model $\mathscr{M}$ where:
  - $\mathscr{M}(\mathbf{x}_{\mathbf{in}}) := \mathbf{14}$
  - $\mathscr{M}$(**f**) := λ `x.ite(x=14,105,ite(x=13,91,… ite(x=1,1,`**`0`**`)…))`

  ⇒$\mathscr{M}$ *is correct only for relevant inputs of original formula, and not e.g.* *`sum(15)=`**`0`***

Model Finding for Recursive Functions : Properties

- Refutation sound
  - When T(Φ) is unsatisfiable, Φ is unsatisfiable
- Model sound, when function definitions are ***admissible***
  - When T(Φ) is satisfiable, Φ is satisfiable



Contract-Based Verification : Recursion

```
@precondition: $x_{in} \geq 0$
int sum(int x)
{
     if( x==0 ){
       return 0;
     }else{
       return x+sum(x-1);
     }
}
@ensures: ~~@ret<100~~
```

False when x_{in}=14

…by finite model finding for recursive functions

Function Synthesis

```
int max(int x, int y)
{
     ???
}
```

@ensures: $@ret \geq x_{in} \wedge @ret \geq y_{in} \wedge (@ret = x_{in} \vee @ret = y_{in})$

Function Synthesis

```
int max(int x, int y)
{
     ???
}
```

@ensures: $@ret \geq x_{in} \wedge @ret \geq y_{in} \wedge (@ret = x_{in} \vee @ret = y_{in})$

⇒ Can be phrased as a synthesis conjecture

Synthesis conjectures

$$\underbrace{\exists \mathtt{f}.\forall \mathtt{x}.\mathtt{P}(\mathtt{f},\mathtt{x})}_{\text{There exists a function } \mathtt{f} \text{ for which property P holds for all } \mathtt{x}}$$

Synthesis conjecture : Max

$$\exists f.\forall xy.\underbrace{f(x,y)\geq x\land f(x,y)\geq y\land(f(x,y)=x\lor f(x,y)=y)}$$

There exists a function f for which our specification holds for all x, y

Synthesis conjecture : Max

$$\forall xy. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$$

- Naively: treat `f` as a free uninterpreted function
  - Ask SMT solver to find model $\mathscr{M}$ where e.g.

$$f^{\mathscr{M}} = \lambda xy.\mathrm{ite}(x \geq y, x, y)$$

EXAMPLE A4…

Synthesis conjecture : Max

$$\forall xy.\mathtt{f(x,y)}\geq \mathtt{x} \wedge \mathtt{f(x,y)}\geq \mathtt{y} \wedge (\mathtt{f(x,y)=x} \vee \mathtt{f(x,y)=y})$$

- Naively: treat f as a free uninterpreted function
  - Ask SMT solver to find model $\mathscr{M}$ where e.g.

$$\mathtt{f}^{\mathscr{M}} = \lambda \mathtt{xy.ite(x}\geq\mathtt{y,x,y)}$$

⇒ This is hard for SMT solvers! Need to use synthesis techniques.

Syntax-Guided Synthesis [Alur et al 2013]

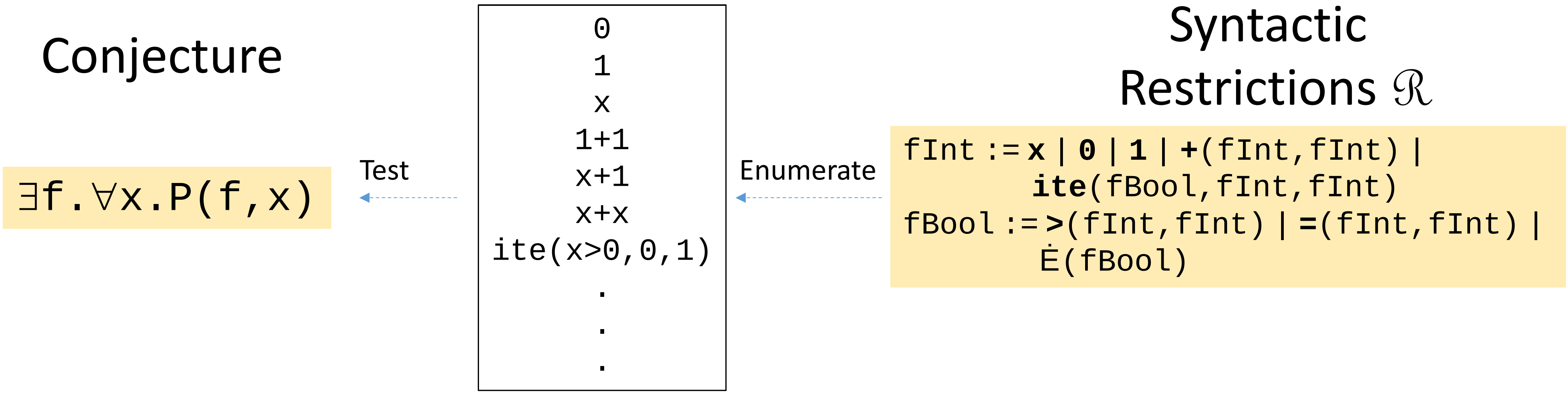
$$\exists \mathtt{f}.\forall \mathtt{xy}.\mathtt{f(x,y)} \geq \mathtt{x} \wedge \mathtt{f(x,y)} \geq \mathtt{y} \wedge (\mathtt{f(x,y)=x} \vee \mathtt{f(x,y)=y})$$

…with syntactic restrictions:

$\mathscr{R}$: fInt := **x** | **y** | **0** | **1** | **+**(fInt,fInt) | **ite**(fBool,fInt,fInt)
fBool := **>**(fInt,fInt) | **=**(fInt,fInt) | Ė(fBool)

Find solutions f = λxy.t, where t is generated by grammar $\mathscr{R}$

Enumerative Syntax-Guided Synthesis



Conjecture

`∃f.∀x.P(f,x)`

Test

```
0
1
x
1+1
x+1
x+x
ite(x>0,0,1)
.
.
.
```

Enumerate

Syntactic Restrictions $\mathfrak{R}$

```
fInt := x | 0 | 1 | +(fInt,fInt) |
        ite(fBool,fInt,fInt)
fBool := >(fInt,fInt) | =(fInt,fInt) |
        Ė(fBool)
```

- Idea: enumerate terms generated by the grammar
- Approach used by number of synthesis solvers [Solar-Lezama 2013,Udupa et al 2013]

Function Synthesis via SyGuS

```
int max(int x, int y)
{
      ???
}
```

@ensures: $@ret \geq x_{in} \wedge @ret \geq y_{in} \wedge (@ret = x_{in} \vee @ret = y_{in})$

EXAMPLE A4-sygus…

Function Synthesis via SyGuS

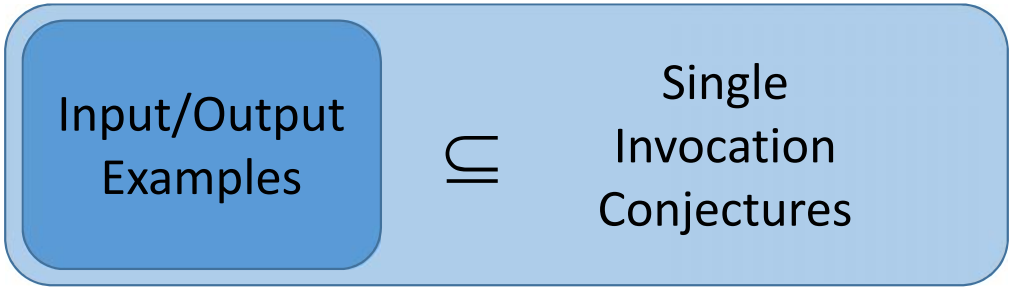
```
int max(int x, int y)
{
     if(xñy){
       return x;
     }else{
       return y;
     }
}
@ensures: @retñx_in ė @retñy_in ė (@ret=x_in Ę @ret=y_in)
```

Types of Synthesis Conjectures

Input/Output Examples

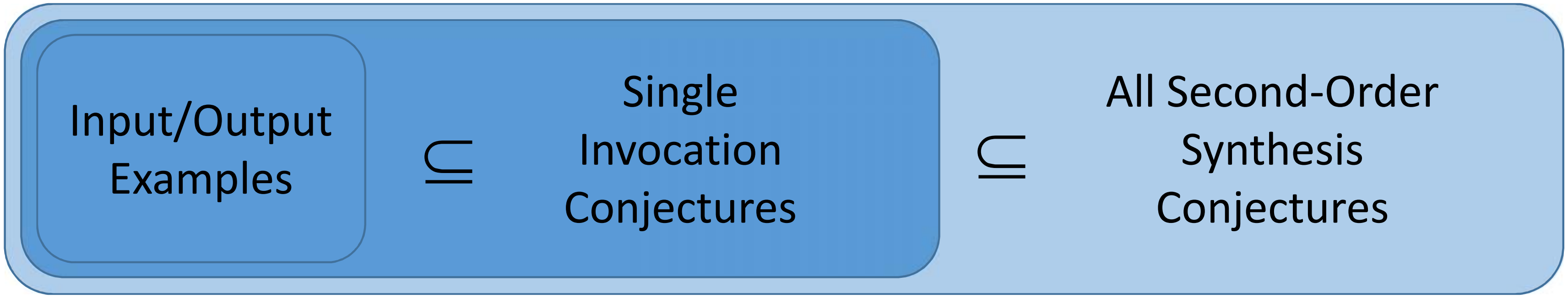
e.g. $\exists f.\forall x.(x=\mathbf{i}_1 \Rightarrow f(x)=\mathbf{o}_1) \land (x=\mathbf{i}_2 \Rightarrow f(x)=\mathbf{o}_2) \land (x=\mathbf{i}_3 \Rightarrow f(x)=\mathbf{o}_3)$

Types of Synthesis Conjectures



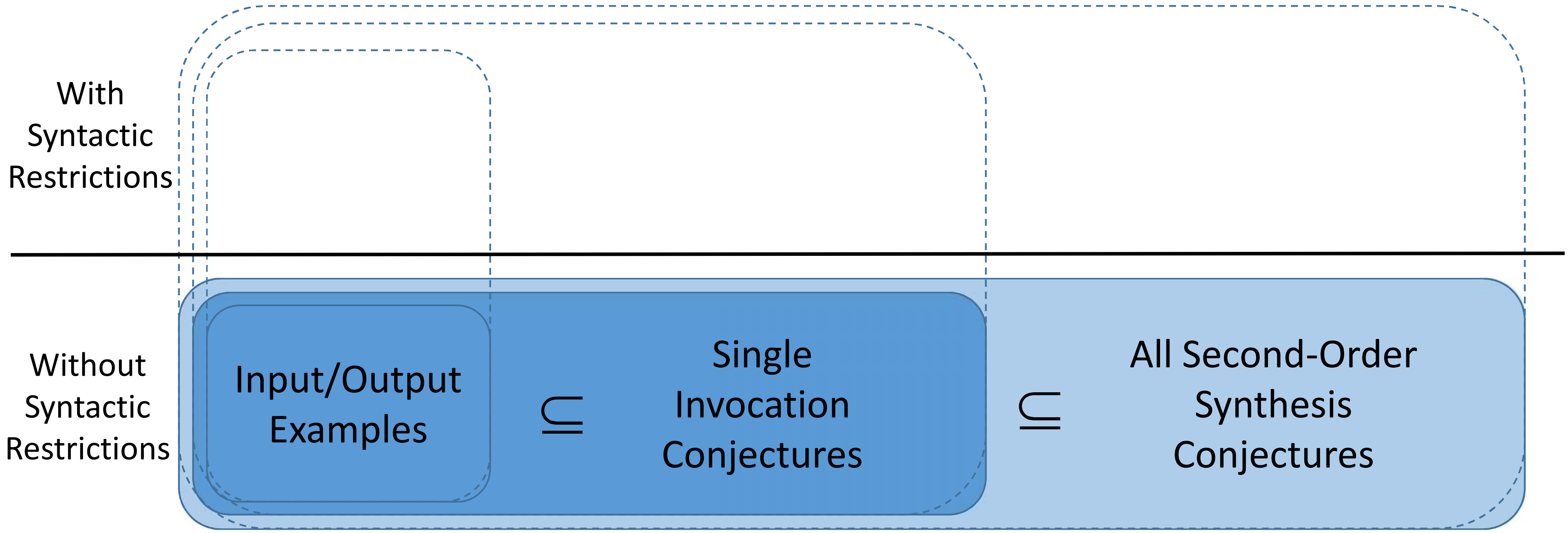
e.g. $\exists f. \forall xy. f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$

Types of Synthesis Conjectures


Input/Output Examples
⊆
Single Invocation Conjectures
⊆
All Second-Order Synthesis Conjectures

e.g. `∃f.∀xy. f(x,y)=f(y,x)`

Types of Synthesis Conjectures



With Syntactic Restrictions

Without Syntactic Restrictions

Input/Output Examples $\subseteq$ Single Invocation Conjectures $\subseteq$ All Second-Order Synthesis Conjectures

Types of Synthesis Conjectures

	Input/Output Examples	Single Invocation Conjectures	Other Second-Order Synthesis Conjectures
With Syntactic Restrictions	?	?	?
Without Syntactic Restrictions	?	?	?

Types of Synthesis Conjectures

	Input/Output Examples	Single Invocation Conjectures	Other Second-Order Synthesis Conjectures
With Syntactic Restrictions	?	?	?
Without Syntactic Restrictions	?	?	?

DPLL(T)-based SMT solvers can be instrumented to handle each class of conjecture

Single Invocation w/o Syntactic Restrictions

With Syntactic Restrictions	?	?	?
Without Syntactic Restrictions	?	?	?
	Input/Output Examples	Single Invocation Conjectures	Other Second-Order Synthesis Conjectures

Function Synthesis via Quantifier Instantiation

- Some synthesis conjectures are *essentially first-order*:

$$\neg\exists f.\forall xy.\ \mathbf{f(x,y)}\geq x \wedge \mathbf{f(x,y)}\geq y \wedge (\mathbf{f(x,y)}=x \vee \mathbf{f(x,y)}=y)$$

"**f(x,y)** is the maximum of x and y"

Function Synthesis via Quantifier Instantiation

$$\neg\exists \texttt{f}.\forall \texttt{xy}.\ \underline{\mathbf{f(x,y)}}\geq \texttt{x} \wedge \underline{\mathbf{f(x,y)}}\geq \texttt{y} \wedge (\underline{\mathbf{f(x,y)}}=\texttt{x} \vee \underline{\mathbf{f(x,y)}}=\texttt{y})$$

$\texttt{Int} \times \texttt{Int} \rightarrow \texttt{Int}$

All occurrence of `f` are in terms of the form **`f(x,y)`**

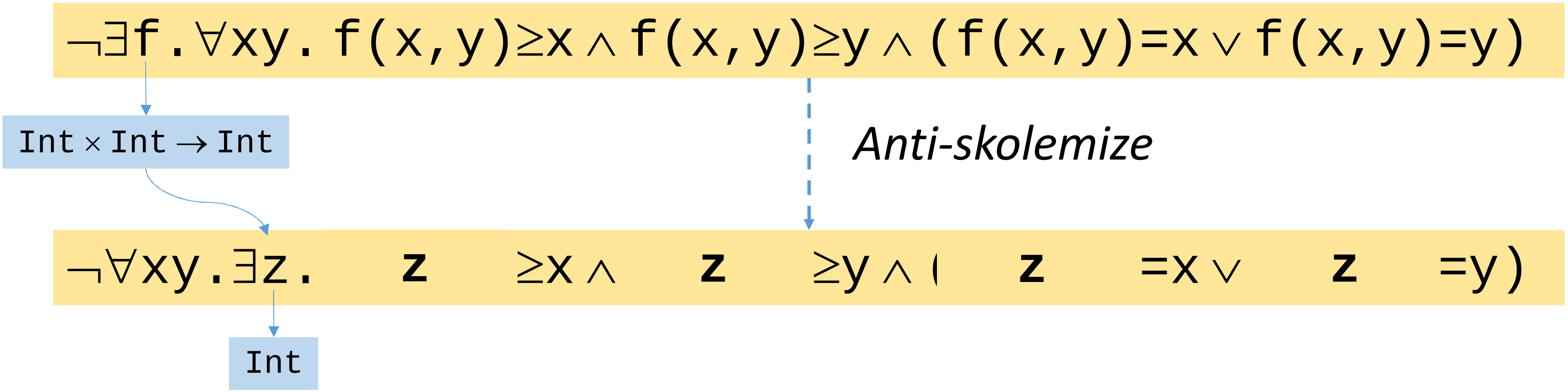
Ĝ “single invocation” synthesis conjectures

Function Synthesis via Quantifier Instantiation

$$\neg\exists f.\forall xy.\ f(x,y)\geq x \land f(x,y)\geq y \land (f(x,y)=x \lor f(x,y)=y)$$

$\mathtt{Int} \times \mathtt{Int} \rightarrow \mathtt{Int}$

Function Synthesis via Quantifier Instantiation



$$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$$

$\mathtt{Int} \times \mathtt{Int} \rightarrow \mathtt{Int}$

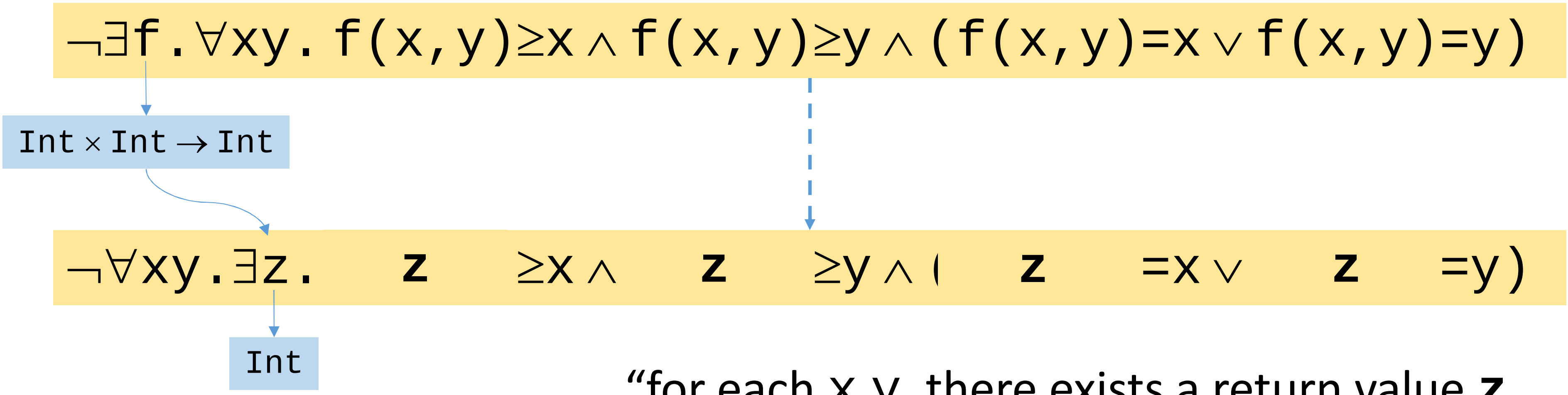
Anti-skolemize

$$\neg\forall xy.\exists z.\quad \mathbf{z} \quad \geq x \wedge \quad \mathbf{z} \quad \geq y \wedge (\quad \mathbf{z} \quad =x \vee \quad \mathbf{z} \quad =y)$$

$\mathtt{Int}$

[Reynolds et al CAV2015]

Function Synthesis via Quantifier Instantiation



$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$

Int × Int → Int

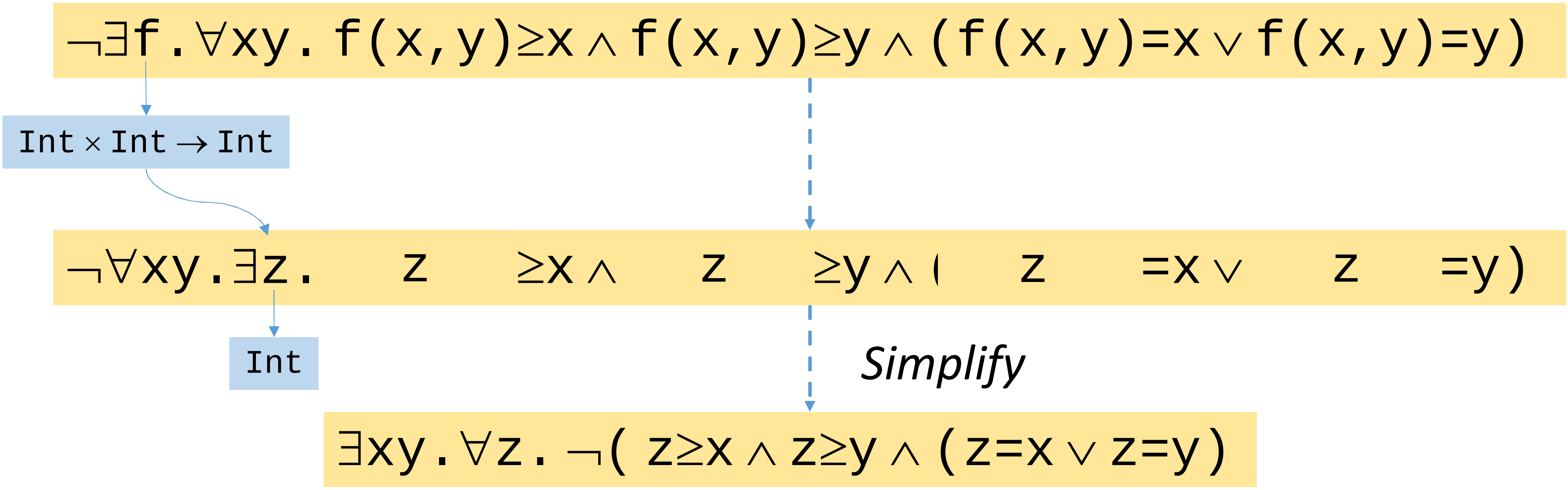
$\neg\forall xy.\exists z.\ \mathbf{z} \geq x \wedge \mathbf{z} \geq y \wedge (\mathbf{z} = x \vee \mathbf{z} = y)$

Int

“for each x,y, there exists a return value **z** that is the maximum of x and y”

[Reynolds et al CAV2015]

Function Synthesis via Quantifier Instantiation



$\neg\exists f.\forall xy.\ f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$

Int × Int → Int

$\neg\forall xy.\exists z.\quad z \quad \geq x \wedge \quad z \quad \geq y \wedge (\quad z \quad = x \vee \quad z \quad = y)$

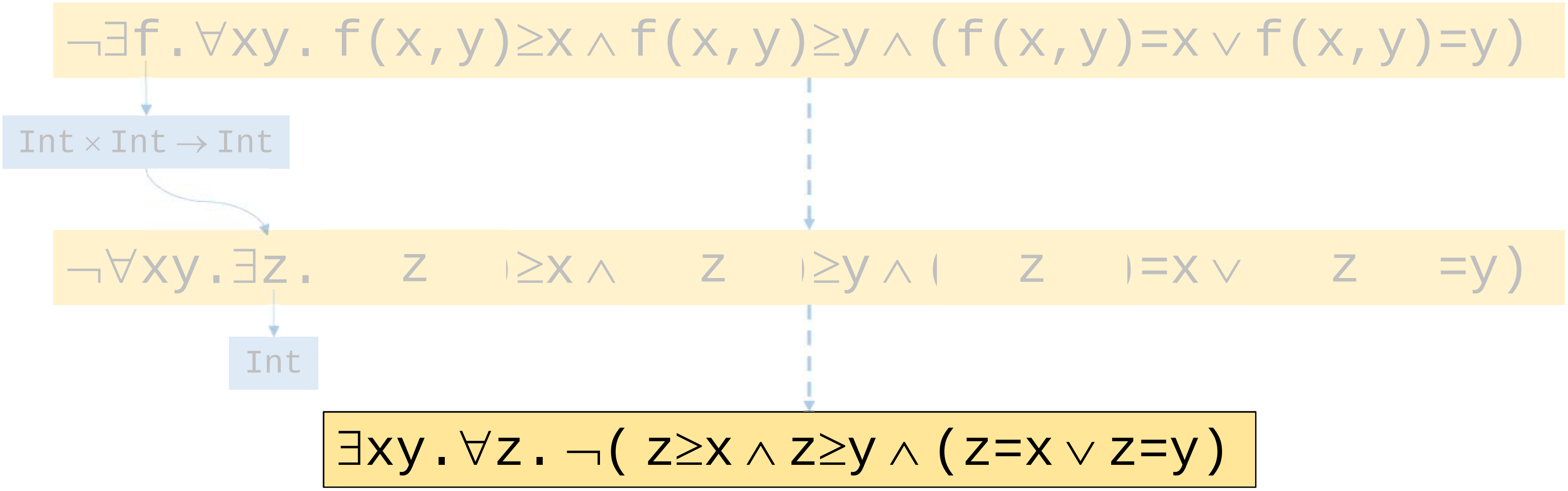
Int

Simplify

$\exists xy.\forall z.\neg(\ z \geq x \wedge z \geq y \wedge (z = x \vee z = y)$

[Reynolds et al CAV2015]

Function Synthesis via Quantifier Instantiation



$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$

Int × Int → Int

$\neg\forall xy.\exists z.\ z \geq x \wedge z \geq y \wedge (z =x \vee z =y)$

Int

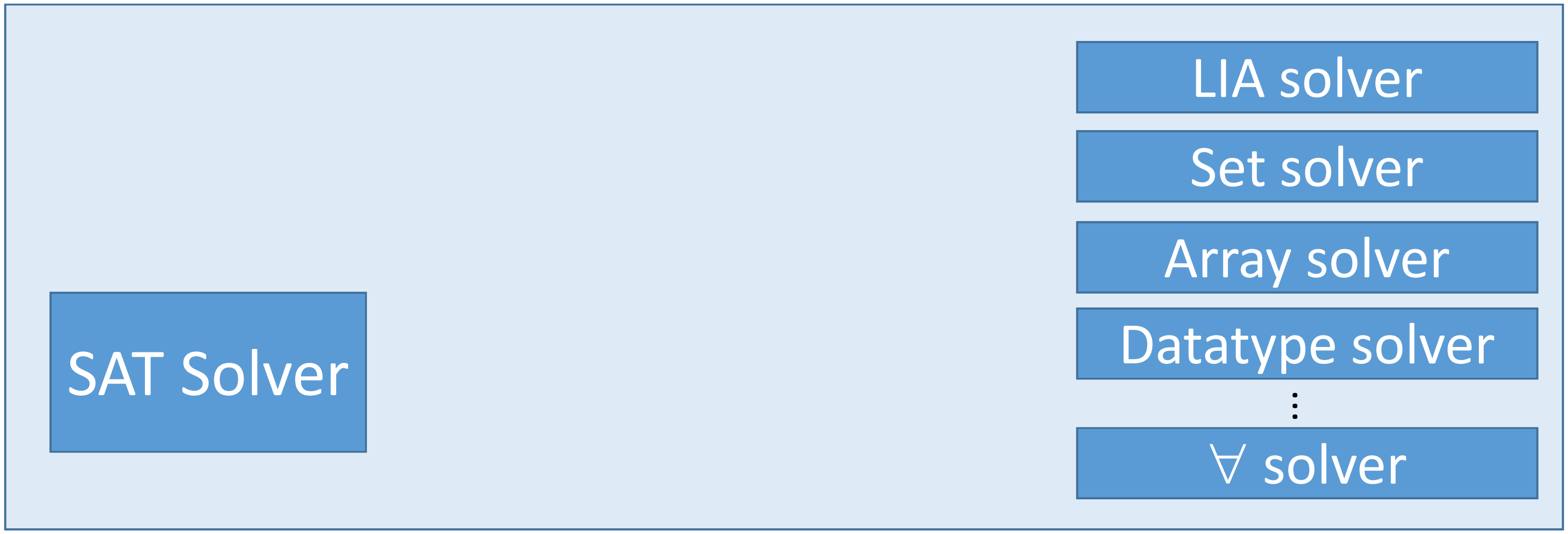
$\exists xy.\forall z.\neg(\ z\geq x \wedge z\geq y \wedge (z=x \vee z=y)$

First-order linear arithmetic Ĝ *Solvable by first-order* ?*-instantiation*

[Reynolds et al CAV2015]

Single Invocation Synthesis in SMT

$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$



SAT Solver

LIA solver

Set solver

Array solver

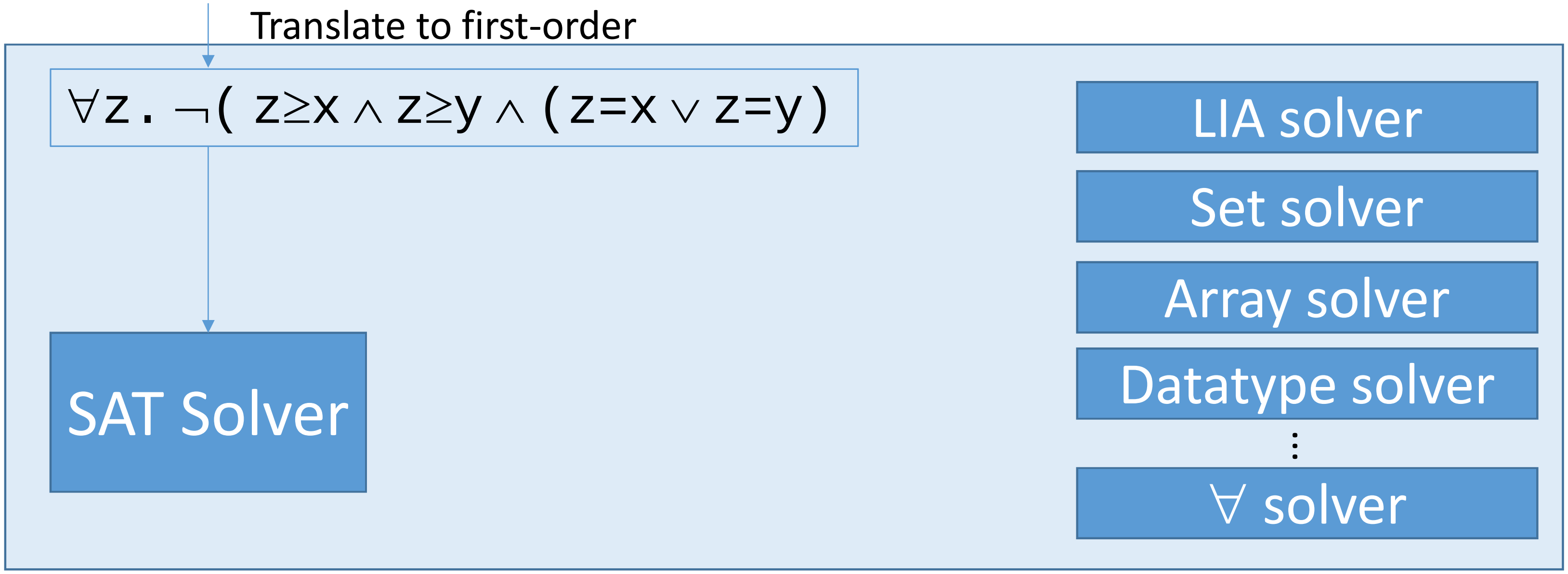
Datatype solver

⋮

∀ solver

Single Invocation Synthesis in SMT

$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$



Translate to first-order

$\forall z.\ \neg(\ z\geq x \wedge z\geq y \wedge (z=x \vee z=y)$

SAT Solver

LIA solver

Set solver

Array solver

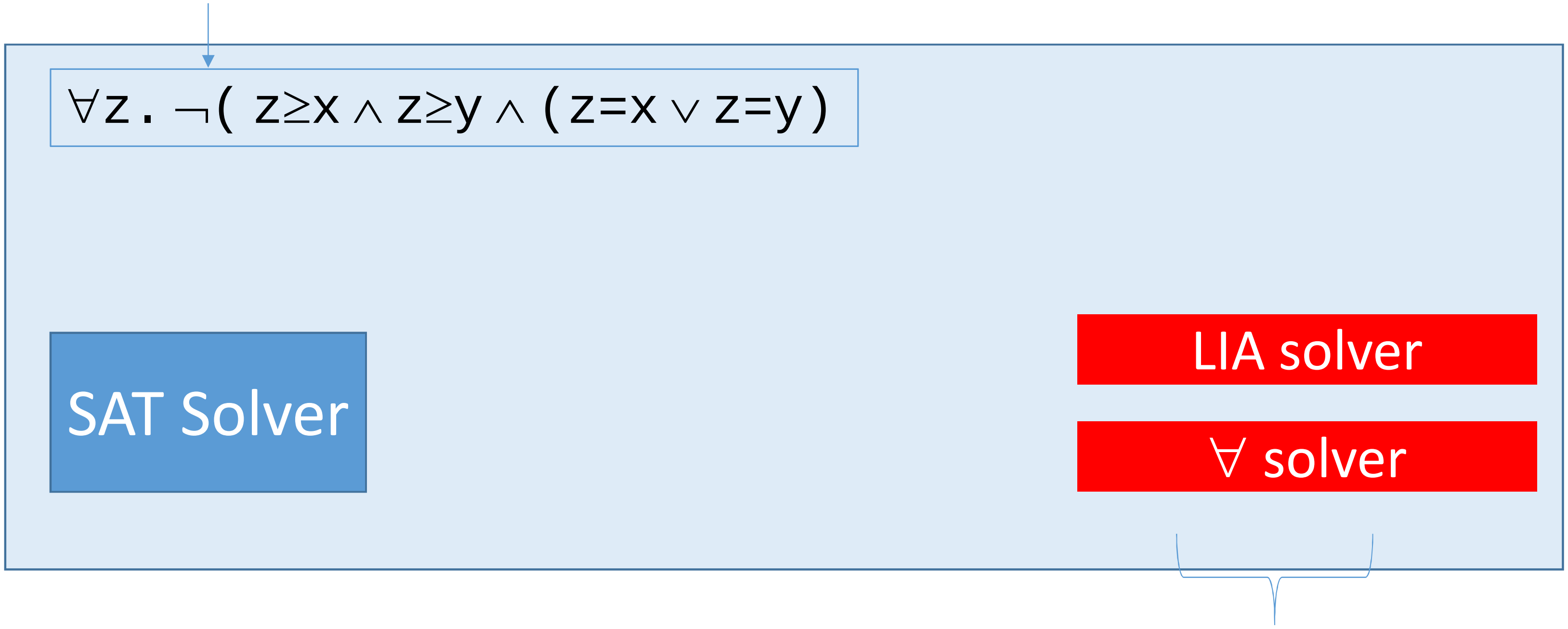
Datatype solver

⋮

∀ solver

Single Invocation Synthesis in SMT

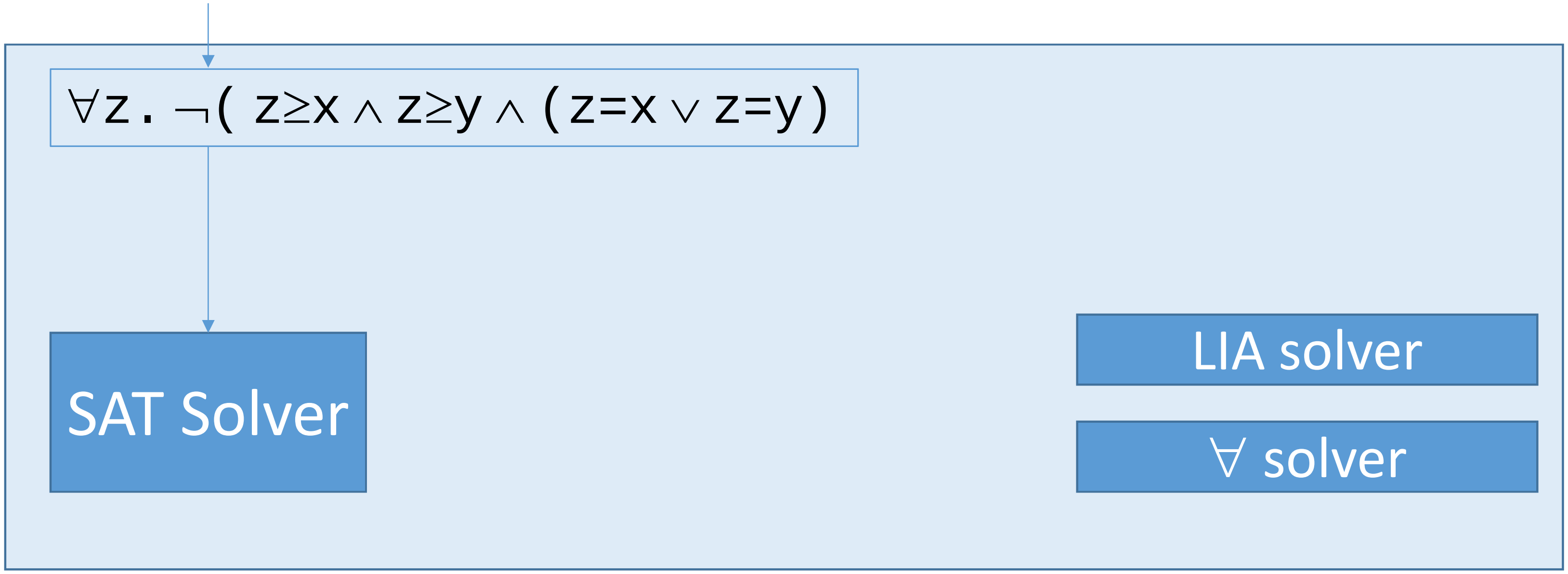
$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$


$\forall z.\ \neg(z\geq x \wedge z\geq y \wedge (z=x \vee z=y)$
SAT Solver
LIA solver
∀ solver

Solve use first-order ∀-instantiation for linear arithmetic (LIA)

Single Invocation Synthesis in SMT

$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$



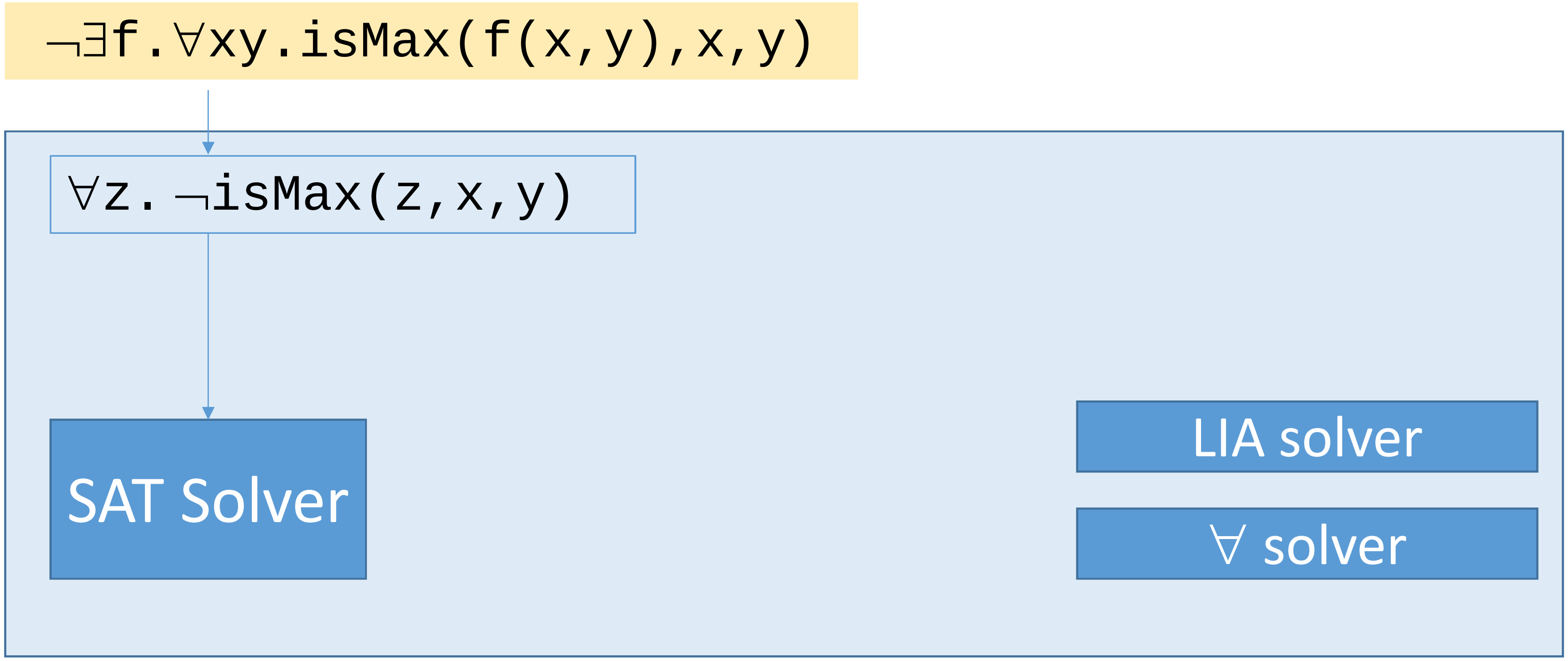
$\forall z.\ \neg(z\geq x \wedge z\geq y \wedge (z=x \vee z=y)$

SAT Solver

LIA solver

∀ solver

Single Invocation Synthesis in SMT



$\neg\exists f.\forall xy.\mathrm{isMax}(f(x,y),x,y)$

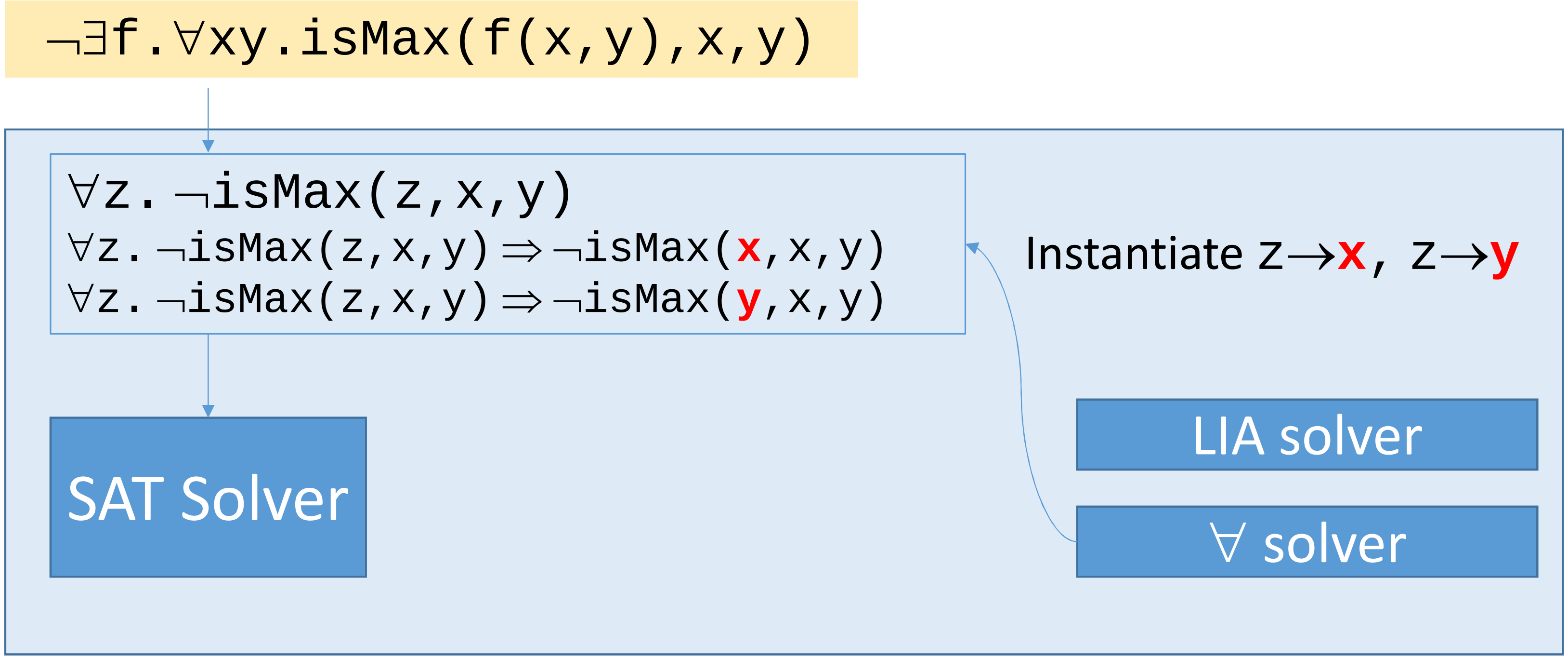
$\forall z.\neg\mathrm{isMax}(z,x,y)$

SAT Solver

LIA solver

∀ solver

Single Invocation Synthesis in SMT



$\neg\exists f.\forall xy.\mathtt{isMax}(f(x,y),x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y) \Rightarrow \neg\mathtt{isMax}(\mathbf{x},x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y) \Rightarrow \neg\mathtt{isMax}(\mathbf{y},x,y)$

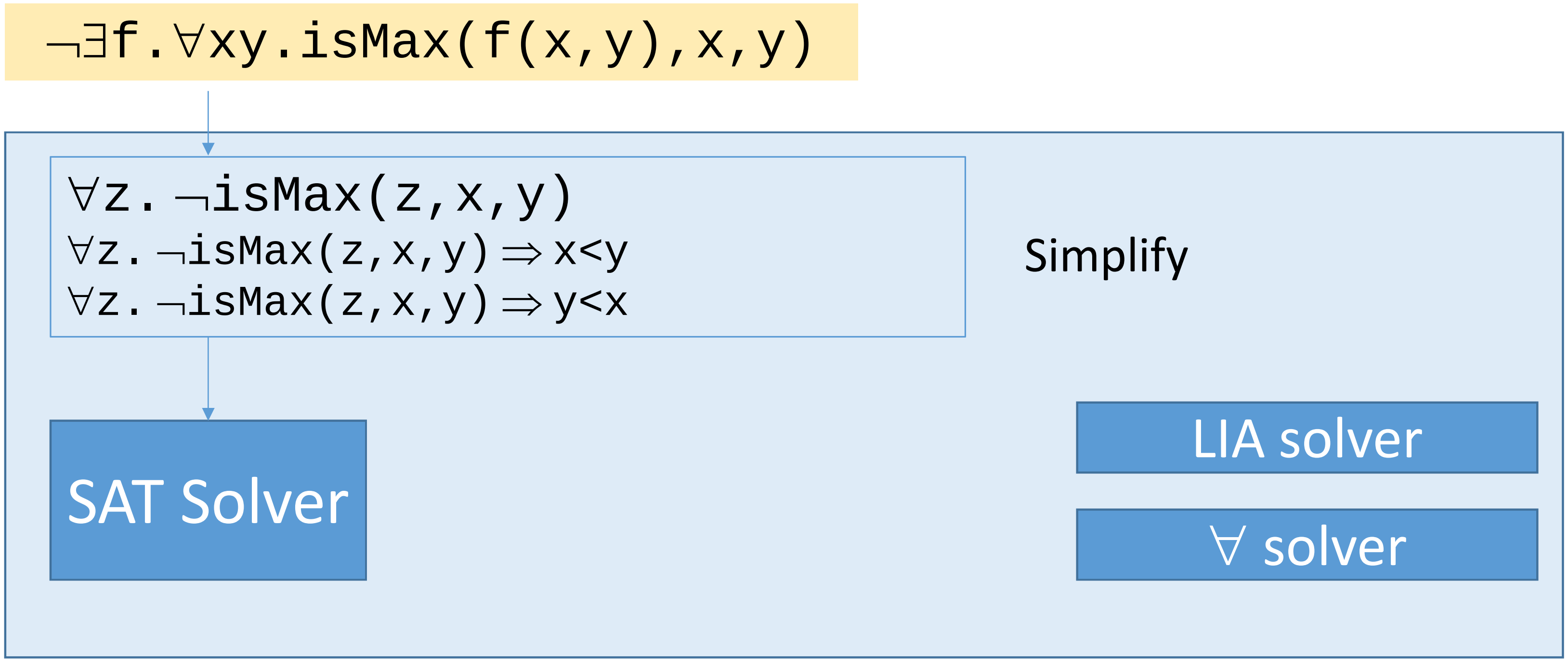
Instantiate $z \rightarrow \mathbf{x}$, $z \rightarrow \mathbf{y}$

SAT Solver

LIA solver

∀ solver

Single Invocation Synthesis in SMT



$\neg\exists f.\forall xy.\mathtt{isMax}(f(x,y),x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y) \Rightarrow x<y$

$\forall z.\neg\mathtt{isMax}(z,x,y) \Rightarrow y<x$

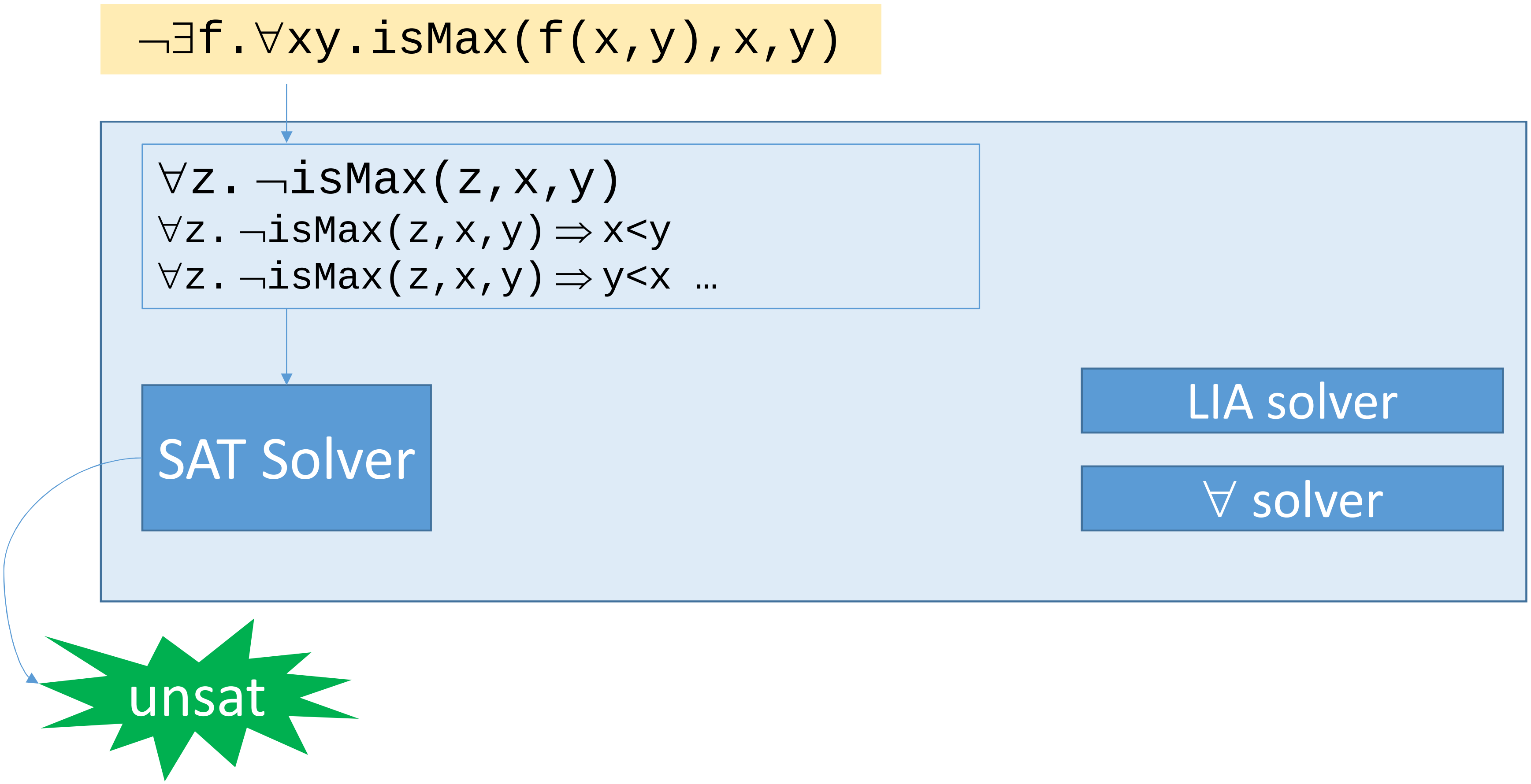
Simplify

SAT Solver

LIA solver

∀ solver

Single Invocation Synthesis in SMT



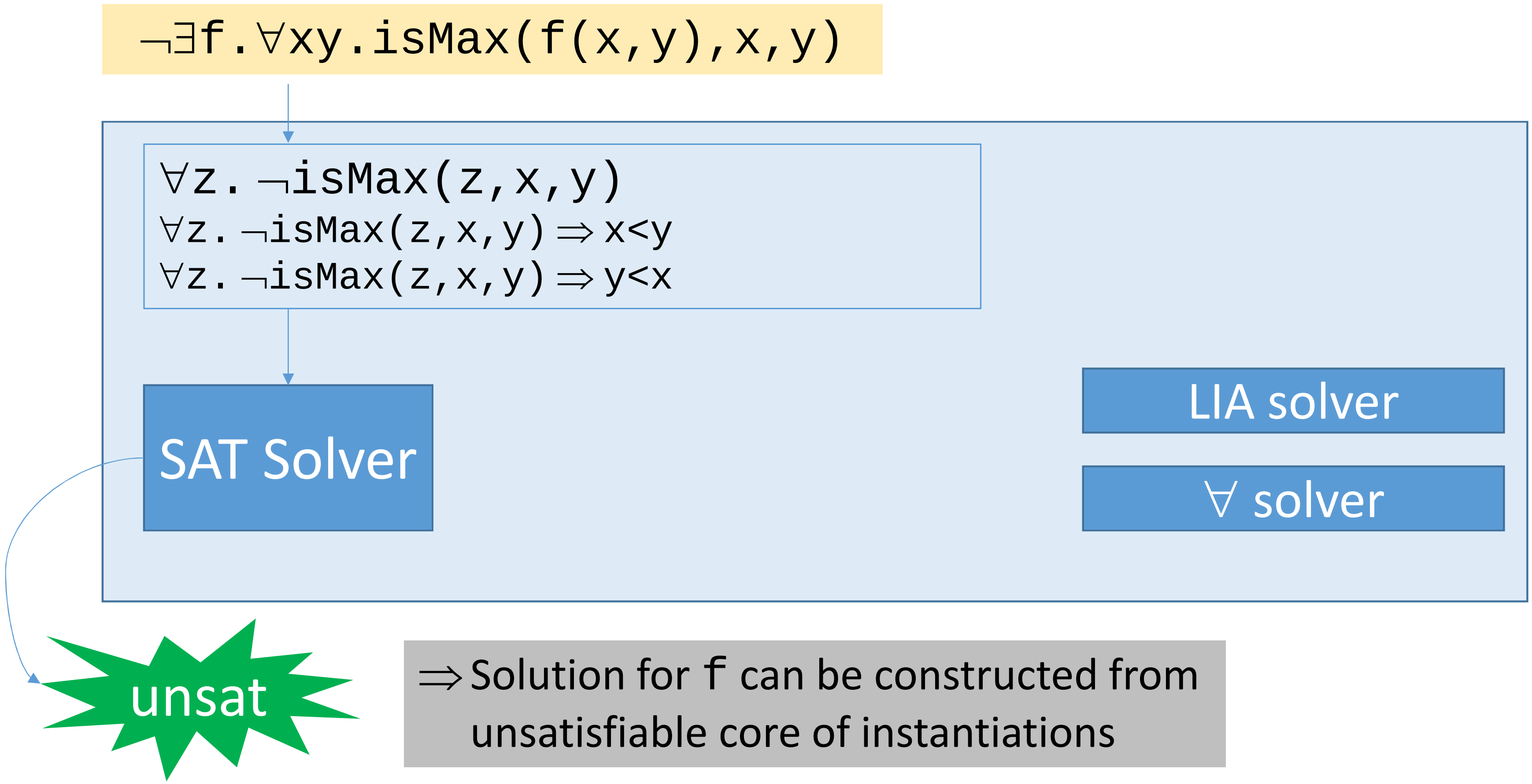
¬∃f.∀xy.isMax(f(x,y),x,y)

∀z.¬isMax(z,x,y)

∀z.¬isMax(z,x,y) ⇒ x<y

∀z.¬isMax(z,x,y) ⇒ y<x …

Single Invocation Synthesis in SMT



¬∃f.∀xy.isMax(f(x,y),x,y)

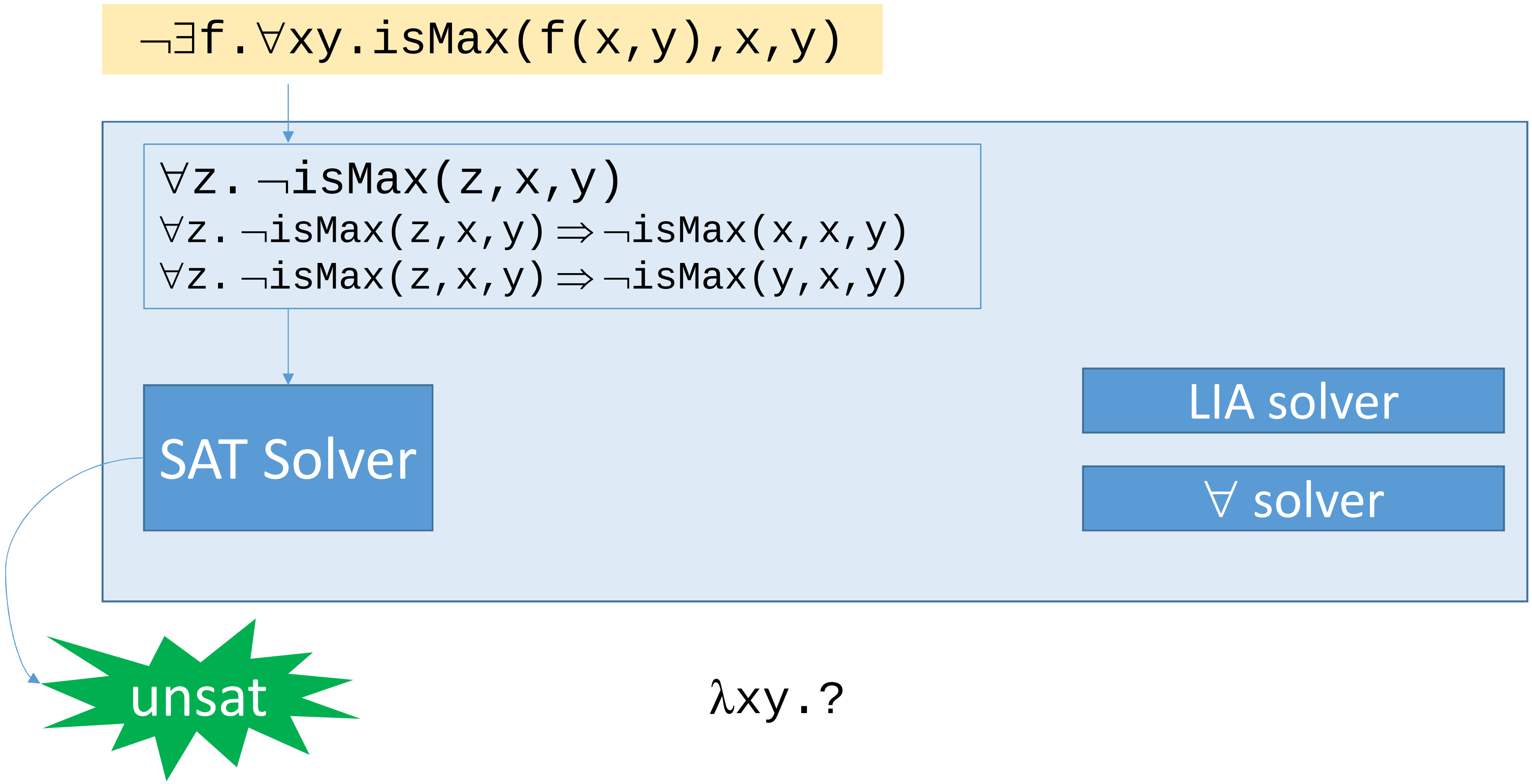
∀z.¬isMax(z,x,y)

∀z.¬isMax(z,x,y) ⇒ x<y

∀z.¬isMax(z,x,y) ⇒ y<x

⇒ Solution for f can be constructed from unsatisfiable core of instantiations

Single Invocation Synthesis in SMT

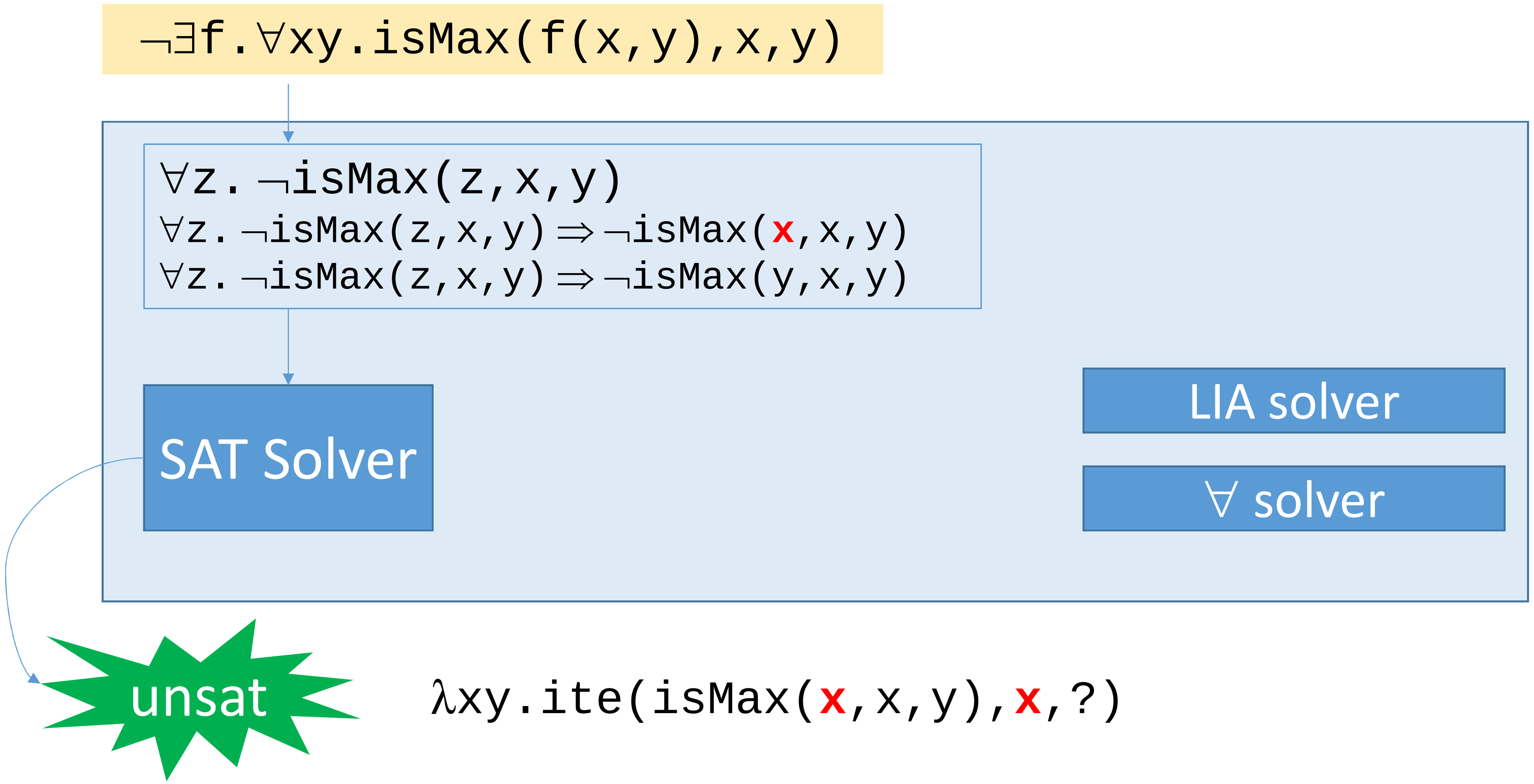


¬∃f.∀xy.isMax(f(x,y),x,y)

∀z.¬isMax(z,x,y)
∀z.¬isMax(z,x,y) ⇒ ¬isMax(x,x,y)
∀z.¬isMax(z,x,y) ⇒ ¬isMax(y,x,y)

λxy.?

Single Invocation Synthesis in SMT



$\neg\exists f.\forall xy.\mathrm{isMax}(f(x,y),x,y)$

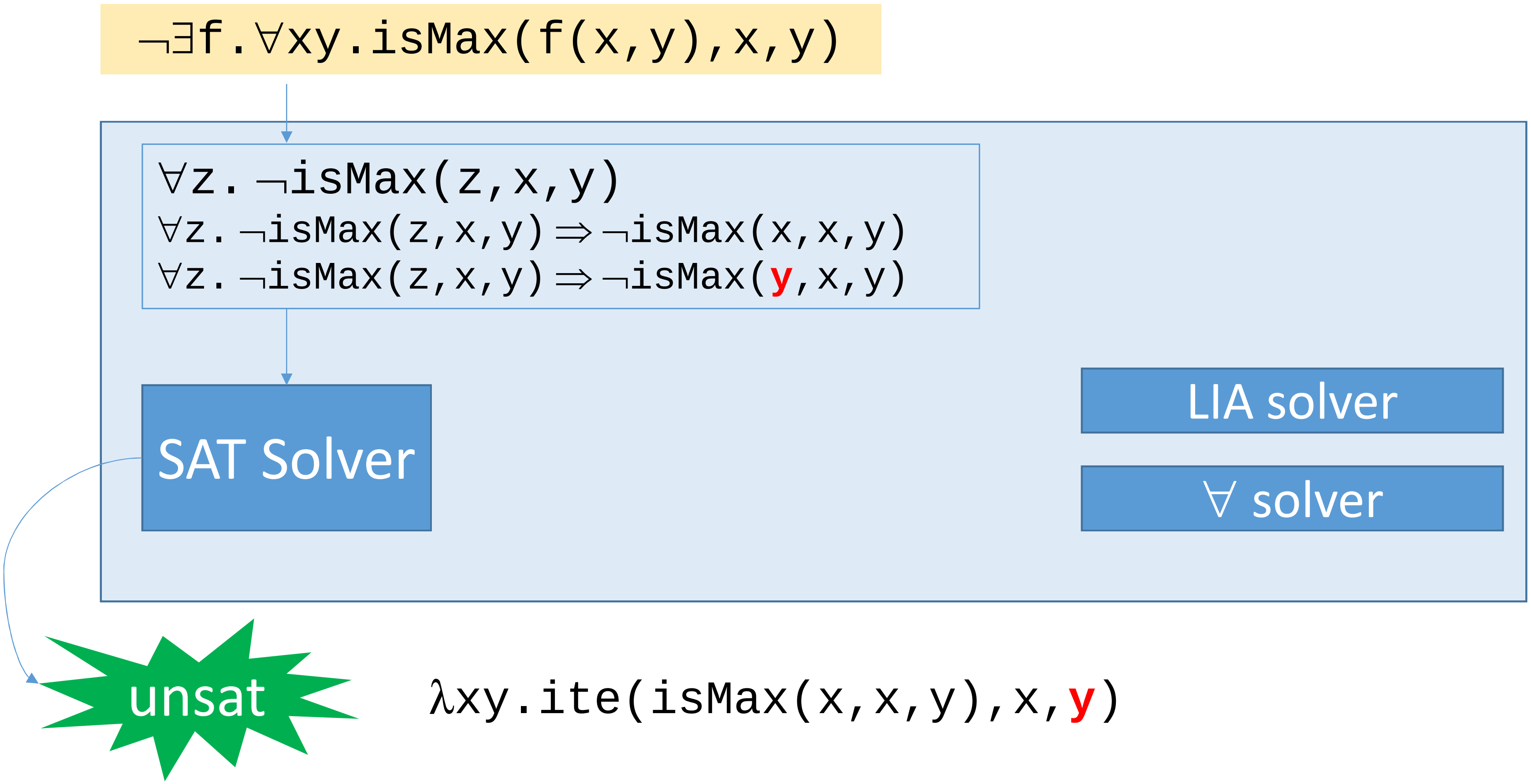
$\forall z.\neg\mathrm{isMax}(z,x,y)$

$\forall z.\neg\mathrm{isMax}(z,x,y) \Rightarrow \neg\mathrm{isMax}(\mathbf{x},x,y)$

$\forall z.\neg\mathrm{isMax}(z,x,y) \Rightarrow \neg\mathrm{isMax}(y,x,y)$

$\lambda xy.\mathrm{ite}(\mathrm{isMax}(\mathbf{x},x,y),\mathbf{x},?)$

Single Invocation Synthesis in SMT

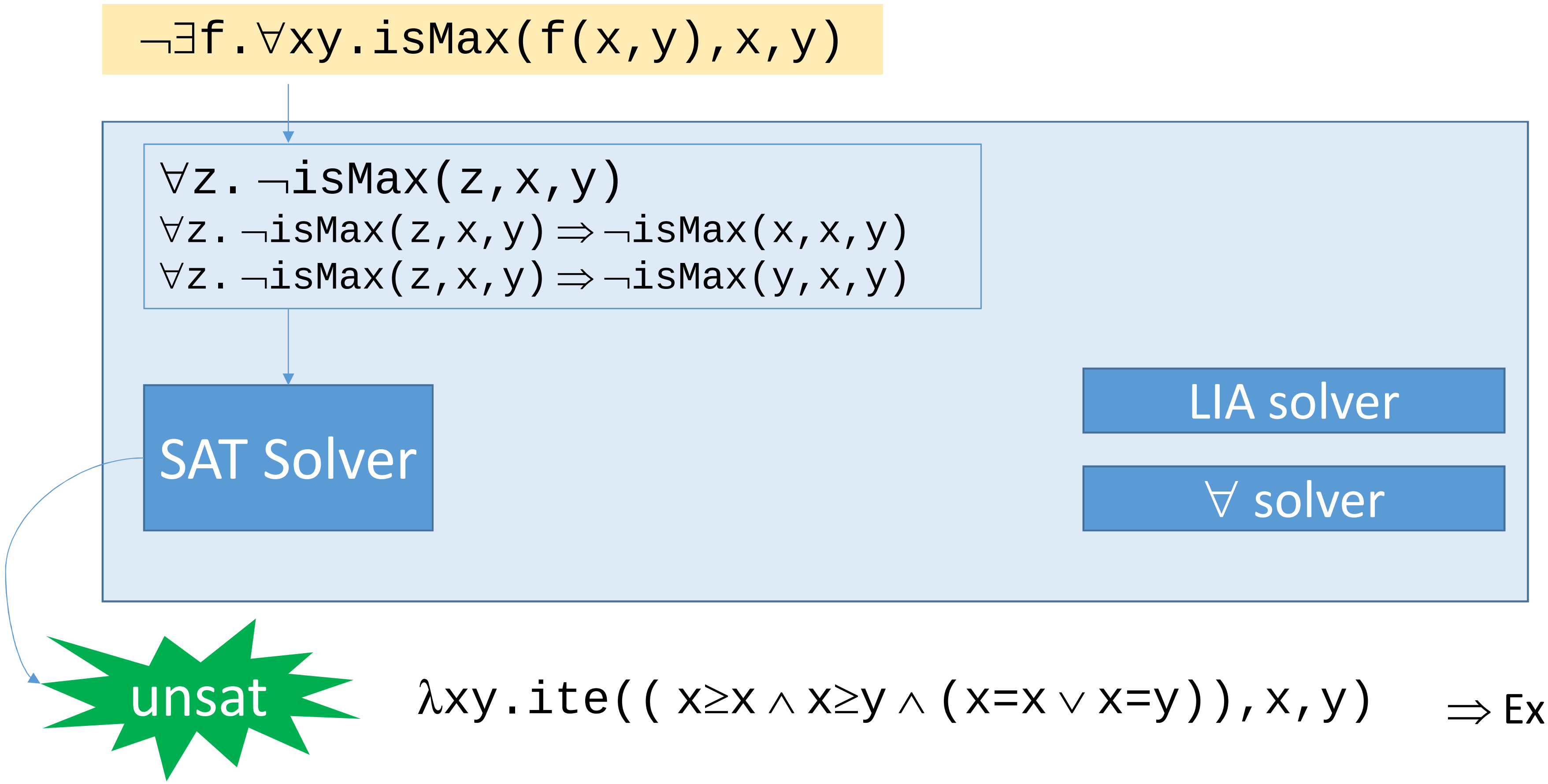


¬∃f.∀xy.isMax(f(x,y),x,y)

∀z.¬isMax(z,x,y)
∀z.¬isMax(z,x,y)⇒¬isMax(x,x,y)
∀z.¬isMax(z,x,y)⇒¬isMax(**y**,x,y)

λxy.ite(isMax(x,x,y),x,**y**)

Single Invocation Synthesis in SMT



$\neg\exists f.\forall xy.\mathtt{isMax}(f(x,y),x,y)$

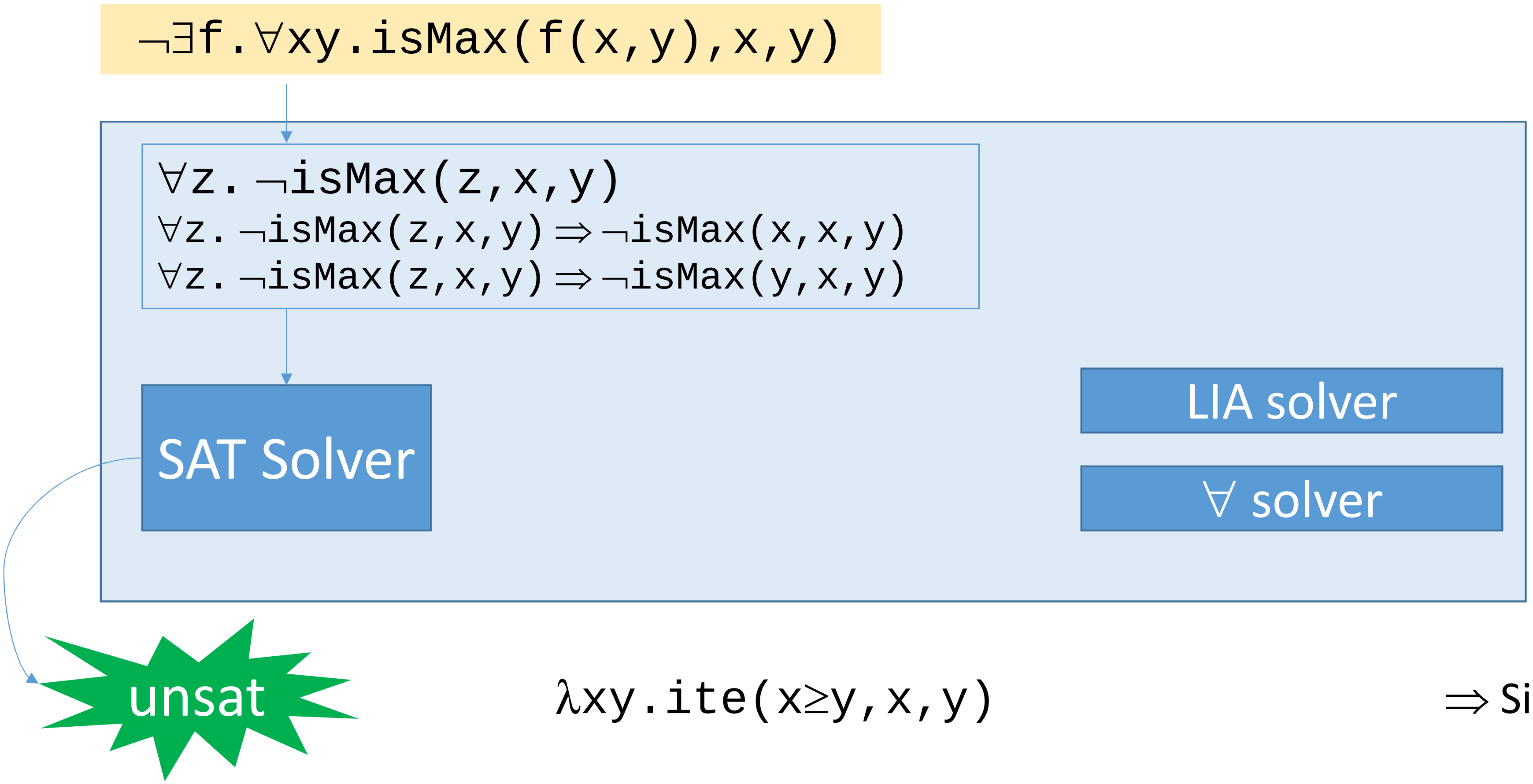
$\forall z.\neg\mathtt{isMax}(z,x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y) \Rightarrow \neg\mathtt{isMax}(x,x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y) \Rightarrow \neg\mathtt{isMax}(y,x,y)$

$\lambda xy.\mathtt{ite}((x \geq x \wedge x \geq y \wedge (x=x \vee x=y)),x,y)$ ⇒ Expand

Single Invocation Synthesis in SMT



$\neg\exists f.\forall xy.\mathtt{isMax}(f(x,y),x,y)$

$\forall z.\neg\mathtt{isMax}(z,x,y)$
$\forall z.\neg\mathtt{isMax}(z,x,y)\Rightarrow\neg\mathtt{isMax}(x,x,y)$
$\forall z.\neg\mathtt{isMax}(z,x,y)\Rightarrow\neg\mathtt{isMax}(y,x,y)$

SAT Solver

LIA solver

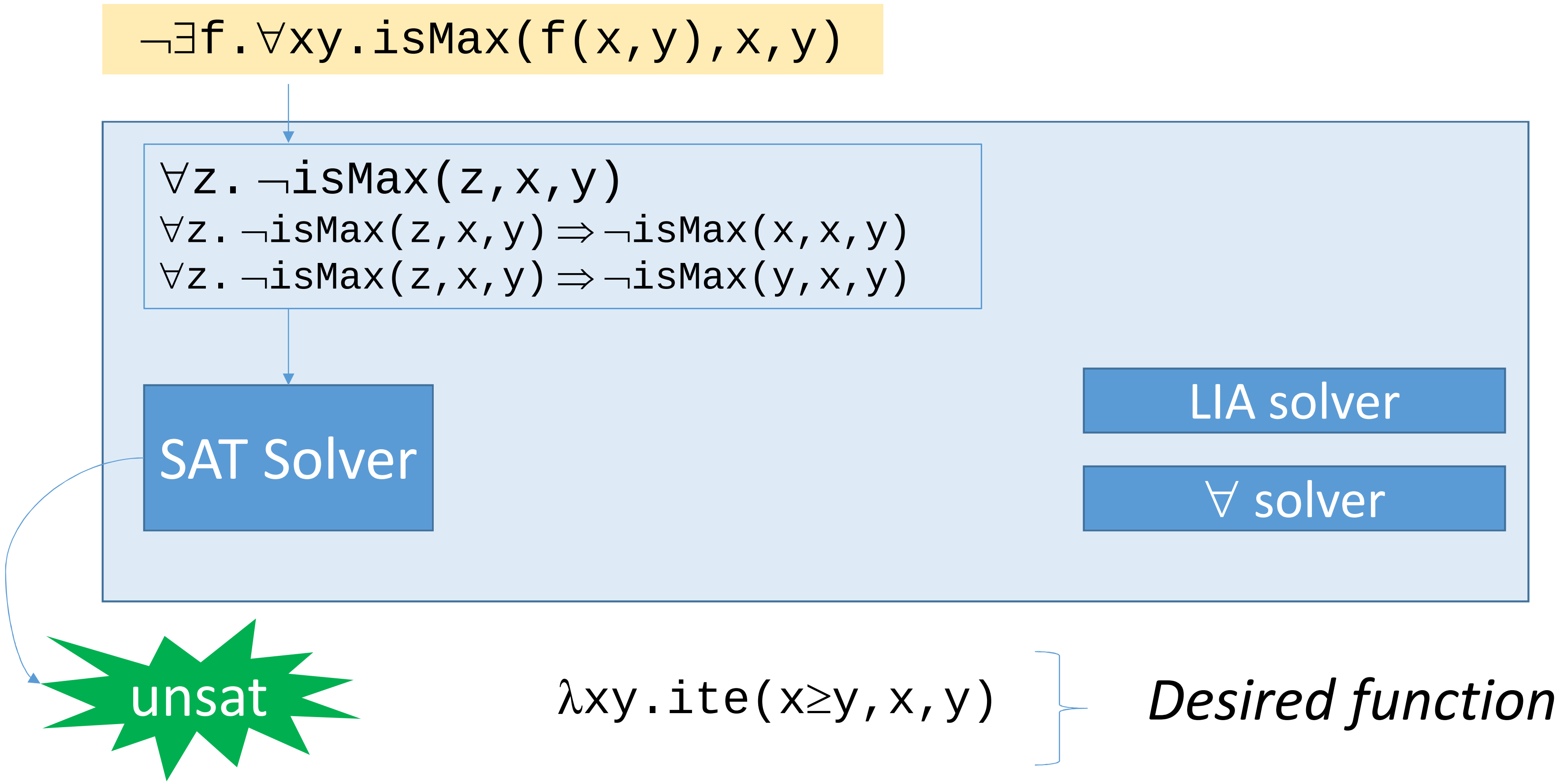
∀ solver

unsat

$\lambda xy.\mathtt{ite}(x\geq y,x,y)$

⇒ Simplify

Single Invocation Synthesis in SMT



$\neg\exists f.\forall xy.\text{isMax}(f(x,y),x,y)$

$\forall z.\neg\text{isMax}(z,x,y)$

$\forall z.\neg\text{isMax}(z,x,y) \Rightarrow \neg\text{isMax}(x,x,y)$

$\forall z.\neg\text{isMax}(z,x,y) \Rightarrow \neg\text{isMax}(y,x,y)$

SAT Solver

LIA solver

∀ solver

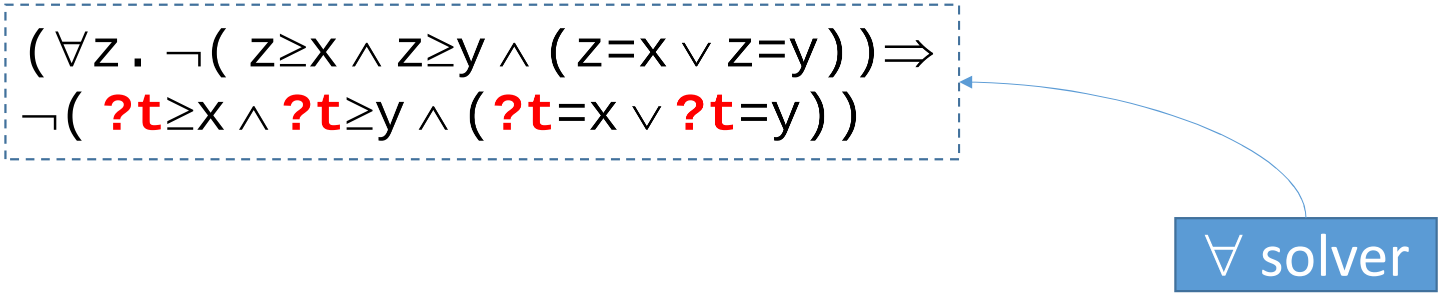
unsat

$\lambda xy.\text{ite}(x \geq y,x,y)$

Desired function

Single Invocation Synthesis in SMT

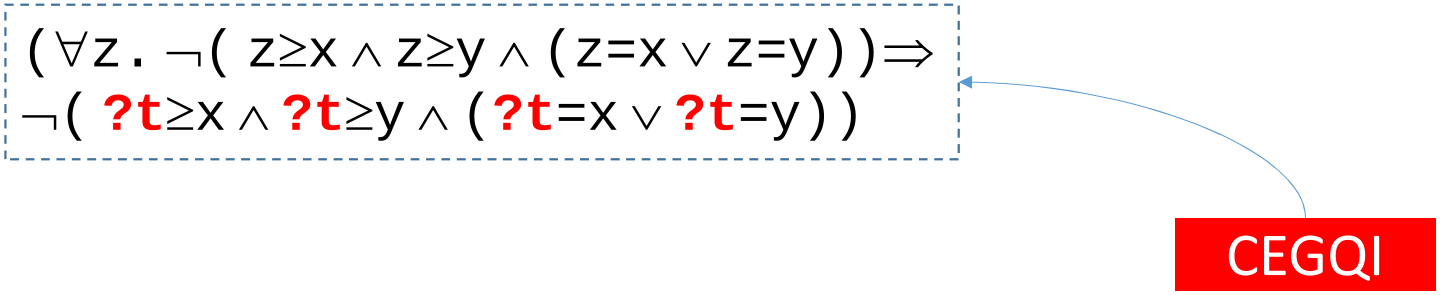
- Requires: method for selecting a term **?t** for instantiation



$$(\forall z.\, \neg(z \geq x \wedge z \geq y \wedge (z = x \vee z = y))) \Rightarrow \neg(?t \geq x \wedge ?t \geq y \wedge (?t = x \vee ?t = y))$$

Single Invocation Synthesis in SMT

- Requires: method for selecting a term **`?t`** for instantiation
  - Use *counterexample-guided quantifier instantiation* (CEGQI)



$$(\forall z.\ \neg(\, z \geq x \wedge z \geq y \wedge (z = x \vee z = y))\Rightarrow$$
$$\neg(\, \mathbf{?t} \geq x \wedge \mathbf{?t} \geq y \wedge (\mathbf{?t} = x \vee \mathbf{?t} = y))$$

Counterexample-Guided $\forall$-Instantiation

Quantifier Elimination Procedures

$$\Leftarrow(\Rightarrow)^{?}$$

Instantiation-Based procedures for $\exists\forall$ formulas

$$\Leftarrow\Rightarrow$$

Synthesis procedures for single-invocation properties

Overview

With Syntactic Restrictions	?	?	?
Without Syntactic Restrictions	?	Counterexample Guided ∀-Instantiation	?
	Input/Output Examples	Single Invocation Conjectures	Other Second-Order Synthesis Conjectures

Function Synthesis via Quantifier Instantiation

```
int max(int x, int y)
{
      ???
}
```

@ensures: $@ret \geq x_{in} \land @ret \geq y_{in} \land (@ret = x_{in} \lor @ret = y_{in})$

If we don’t restrict our syntax,
use single invocation techniques…

EXAMPLE A4-sygus-no-syntax…

Function Synthesis via Quantifier Instantiation

```
int max(int x, int y)
{
     if(x+(-1)*yñ0){
       return x;
     }else{
       return y;
     }
}
```

@ensures: $@ret{\geq}x_{in} \wedge @ret{\geq}y_{in} \wedge (@ret{=}x_{in} \vee @ret{=}y_{in})$

⇒Single invocation techniques are much faster, but typically produce larger or non-optimal solutions

What if we apply CEGQI to I/O Examples?

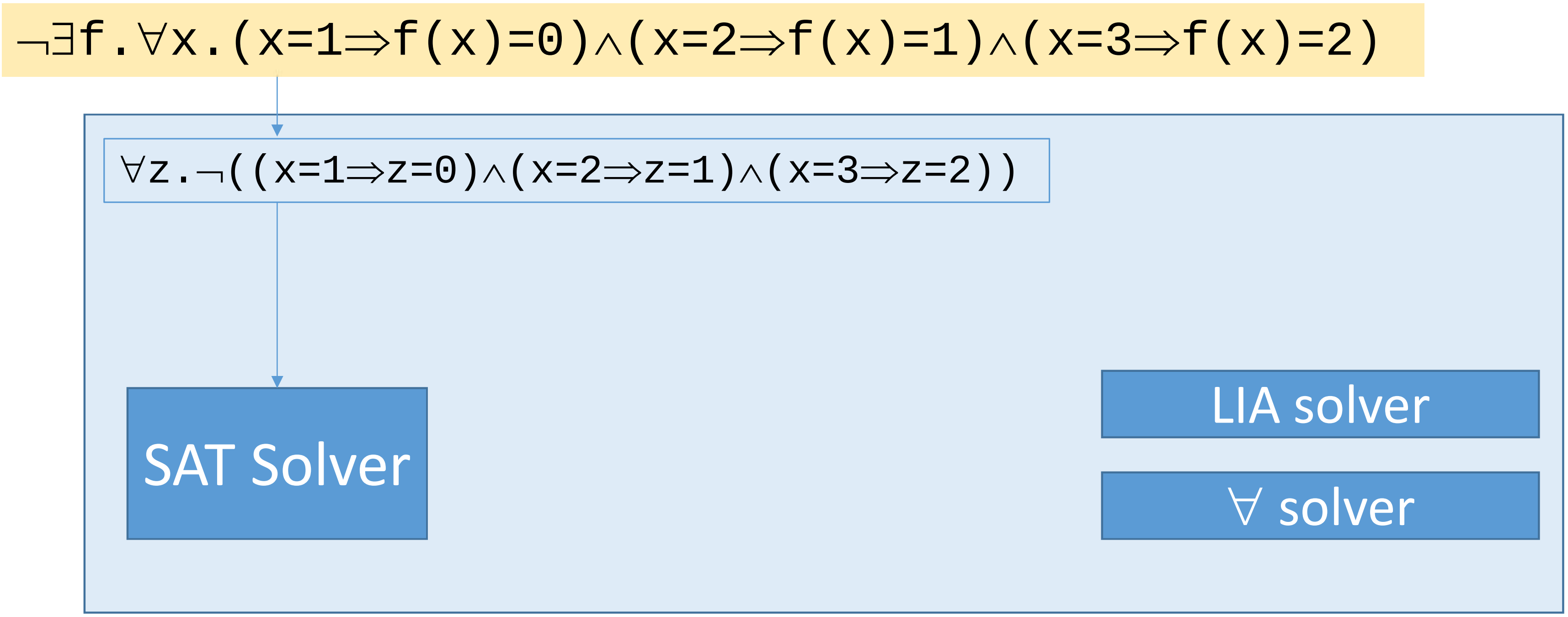
$$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\land(x=2\Rightarrow f(x)=1)\land(x=3\Rightarrow f(x)=2)$$

SAT Solver

LIA solver

∀ solver

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\wedge(x=2\Rightarrow f(x)=1)\wedge(x=3\Rightarrow f(x)=2)$

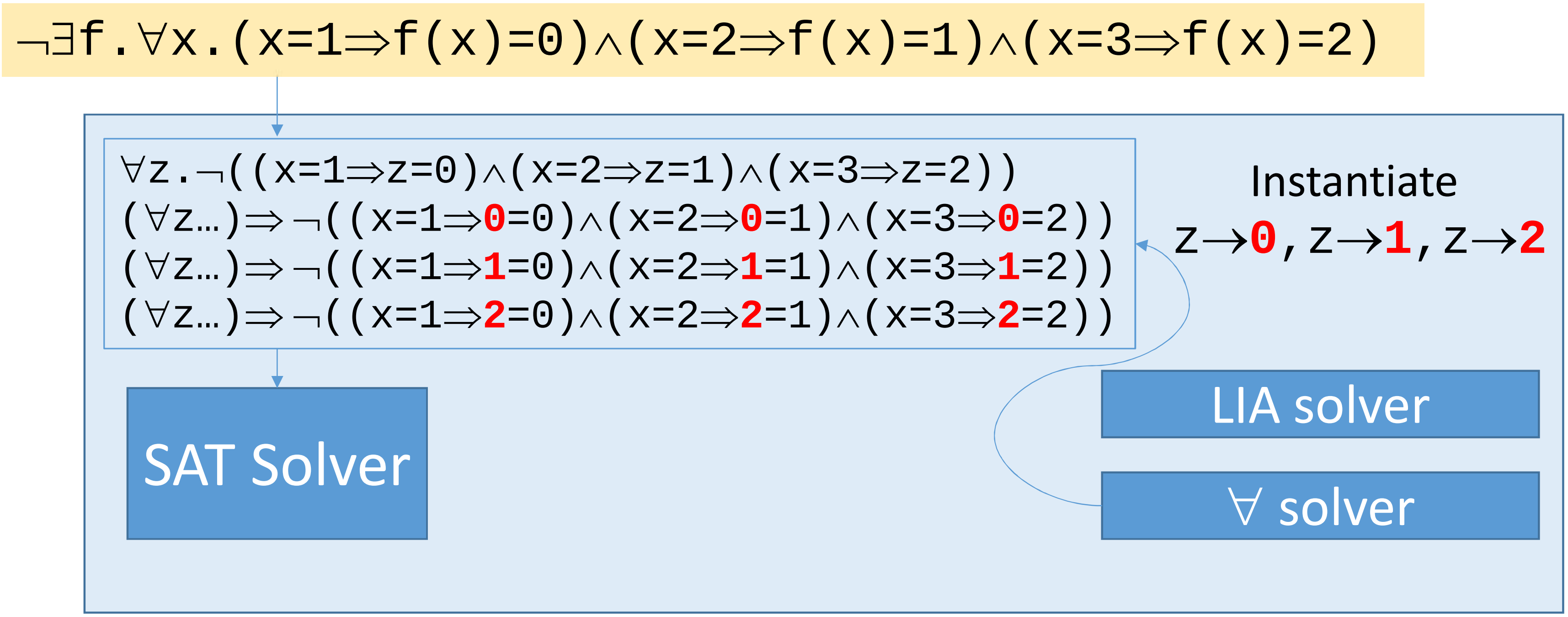
$\forall z.\neg((x=1\Rightarrow z=0)\wedge(x=2\Rightarrow z=1)\wedge(x=3\Rightarrow z=2))$

SAT Solver

LIA solver

∀ solver

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\wedge(x=2\Rightarrow f(x)=1)\wedge(x=3\Rightarrow f(x)=2)$

$\forall z.\neg((x=1\Rightarrow z=0)\wedge(x=2\Rightarrow z=1)\wedge(x=3\Rightarrow z=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow \mathbf{0}=0)\wedge(x=2\Rightarrow \mathbf{0}=1)\wedge(x=3\Rightarrow \mathbf{0}=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow \mathbf{1}=0)\wedge(x=2\Rightarrow \mathbf{1}=1)\wedge(x=3\Rightarrow \mathbf{1}=2))$

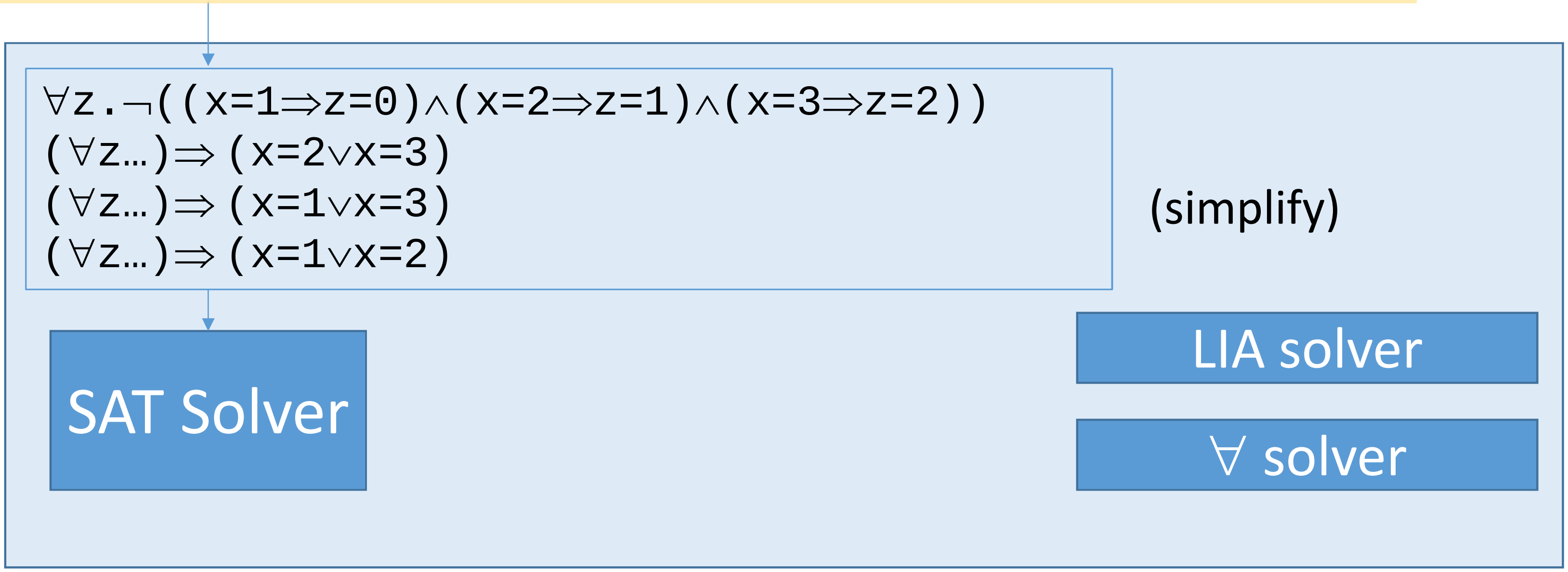
$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow \mathbf{2}=0)\wedge(x=2\Rightarrow \mathbf{2}=1)\wedge(x=3\Rightarrow \mathbf{2}=2))$

Instantiate

$z\rightarrow\mathbf{0}, z\rightarrow\mathbf{1}, z\rightarrow\mathbf{2}$

What if we apply CEGQI to I/O Examples?

$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\wedge(x=2\Rightarrow f(x)=1)\wedge(x=3\Rightarrow f(x)=2)$



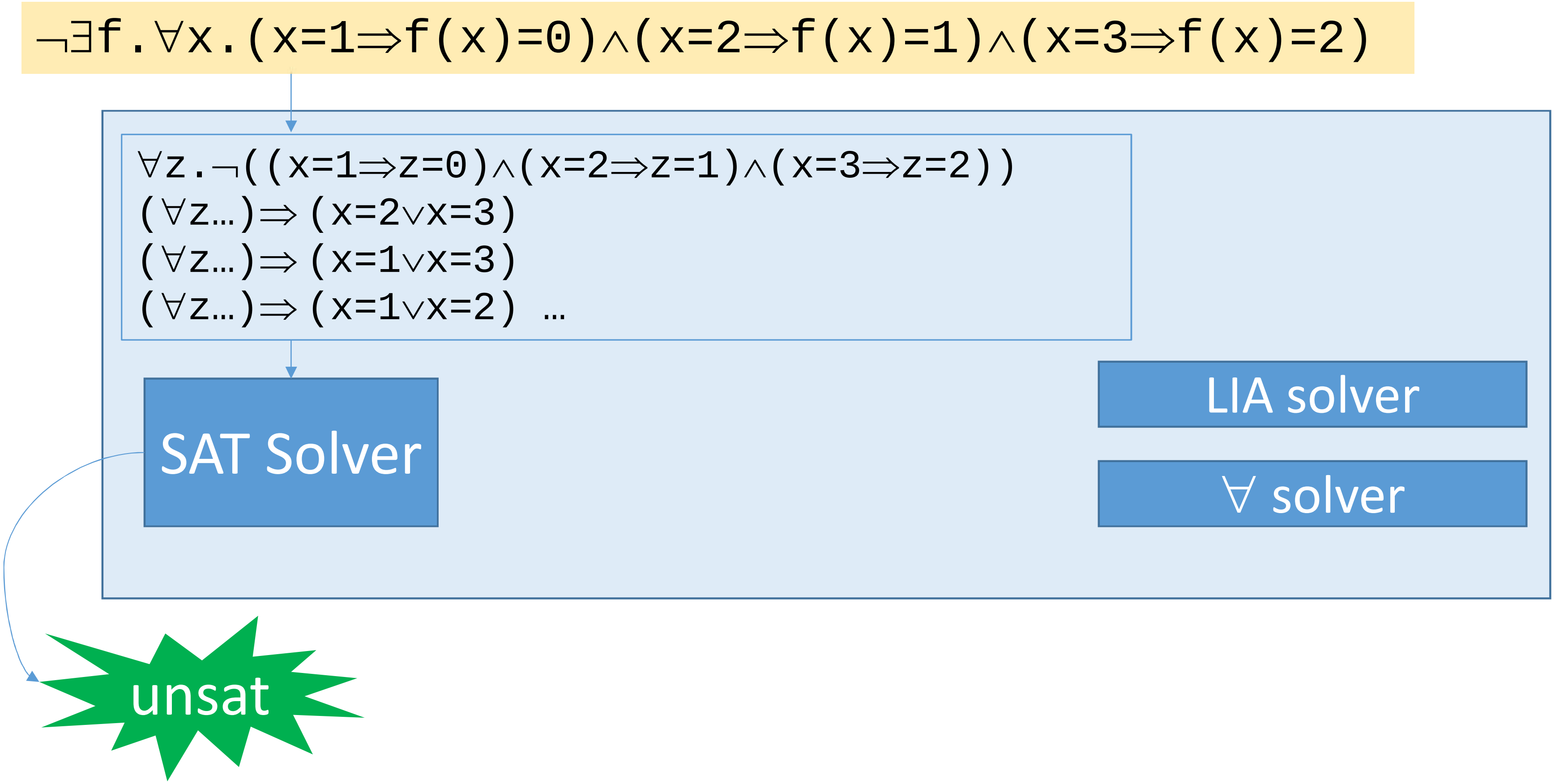
$\forall z.\neg((x=1\Rightarrow z=0)\wedge(x=2\Rightarrow z=1)\wedge(x=3\Rightarrow z=2))$

$(\forall z\ldots)\Rightarrow(x=2\vee x=3)$

$(\forall z\ldots)\Rightarrow(x=1\vee x=3)$

$(\forall z\ldots)\Rightarrow(x=1\vee x=2)$

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\wedge(x=2\Rightarrow f(x)=1)\wedge(x=3\Rightarrow f(x)=2)$

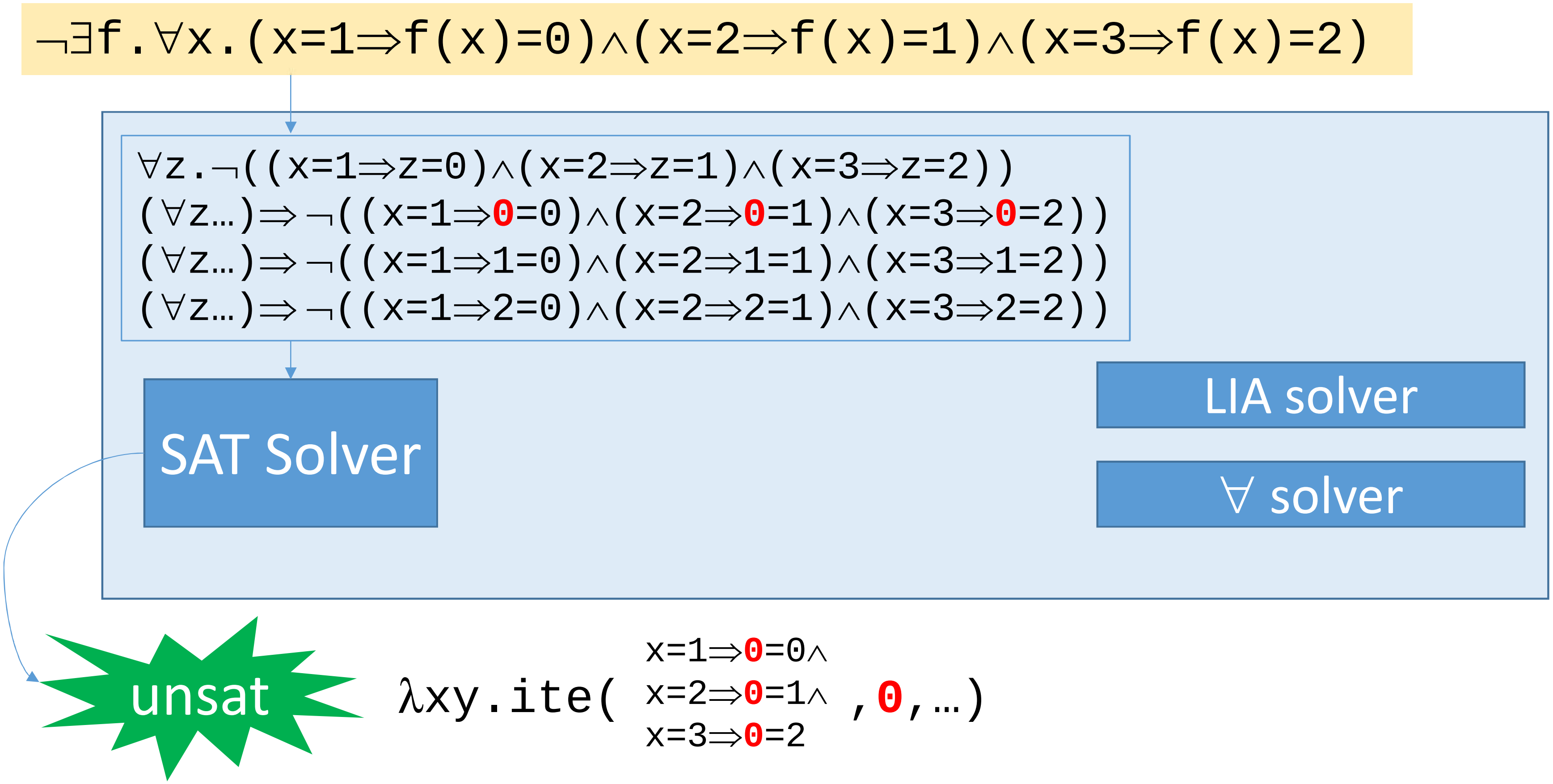
$\forall z.\neg((x=1\Rightarrow z=0)\wedge(x=2\Rightarrow z=1)\wedge(x=3\Rightarrow z=2))$

$(\forall z\ldots)\Rightarrow(x=2\vee x=3)$

$(\forall z\ldots)\Rightarrow(x=1\vee x=3)$

$(\forall z\ldots)\Rightarrow(x=1\vee x=2)$ …

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\land(x=2\Rightarrow f(x)=1)\land(x=3\Rightarrow f(x)=2)$

$\forall z.\neg((x=1\Rightarrow z=0)\land(x=2\Rightarrow z=1)\land(x=3\Rightarrow z=2))$

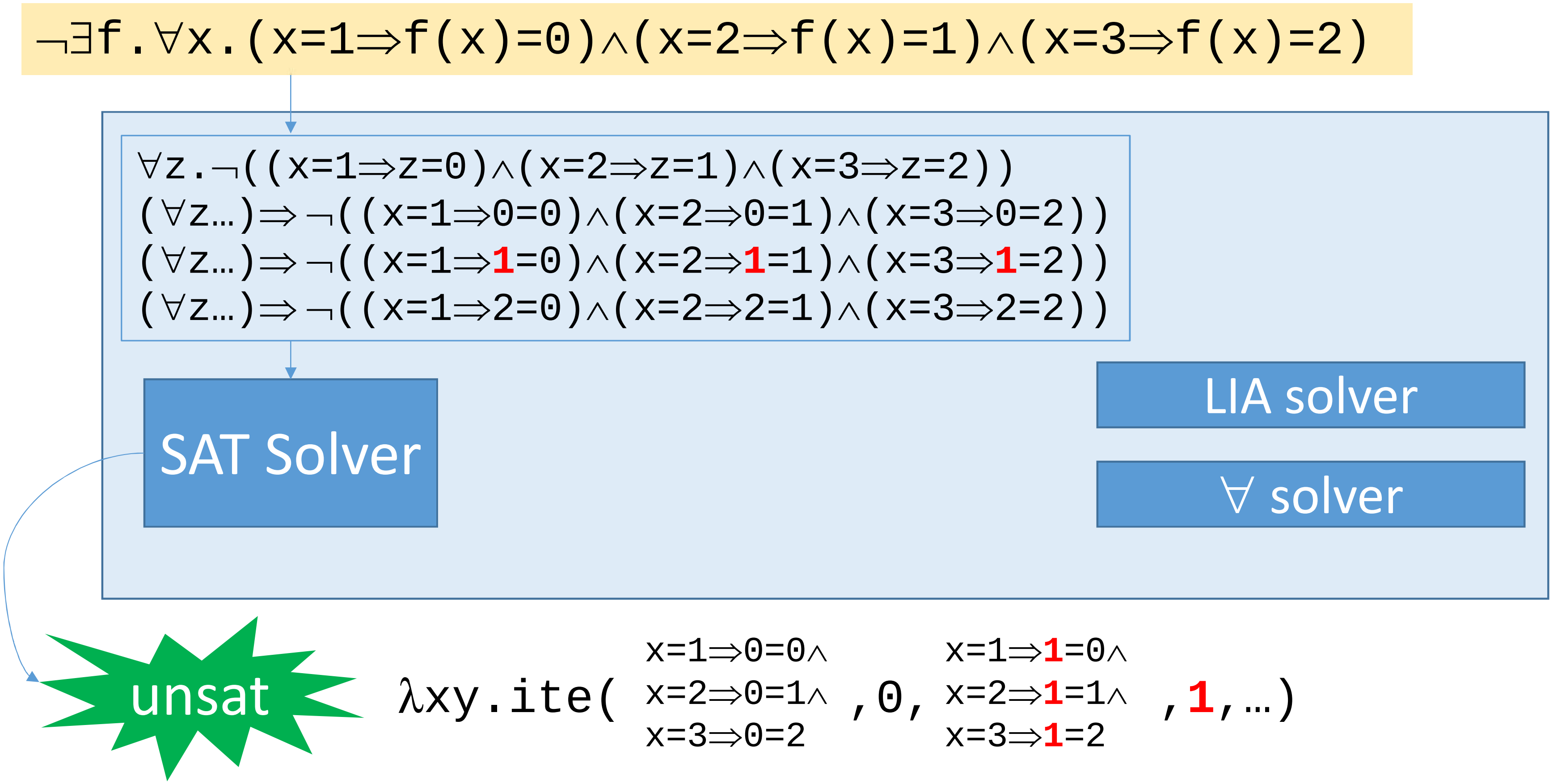
$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 0=0)\land(x=2\Rightarrow 0=1)\land(x=3\Rightarrow 0=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 1=0)\land(x=2\Rightarrow 1=1)\land(x=3\Rightarrow 1=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 2=0)\land(x=2\Rightarrow 2=1)\land(x=3\Rightarrow 2=2))$

$$\lambda xy.\mathtt{ite}(\begin{array}{l}x=1\Rightarrow 0=0\land\\ x=2\Rightarrow 0=1\land\\ x=3\Rightarrow 0=2\end{array},0,\ldots)$$

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\land(x=2\Rightarrow f(x)=1)\land(x=3\Rightarrow f(x)=2)$

$\forall z.\neg((x=1\Rightarrow z=0)\land(x=2\Rightarrow z=1)\land(x=3\Rightarrow z=2))$

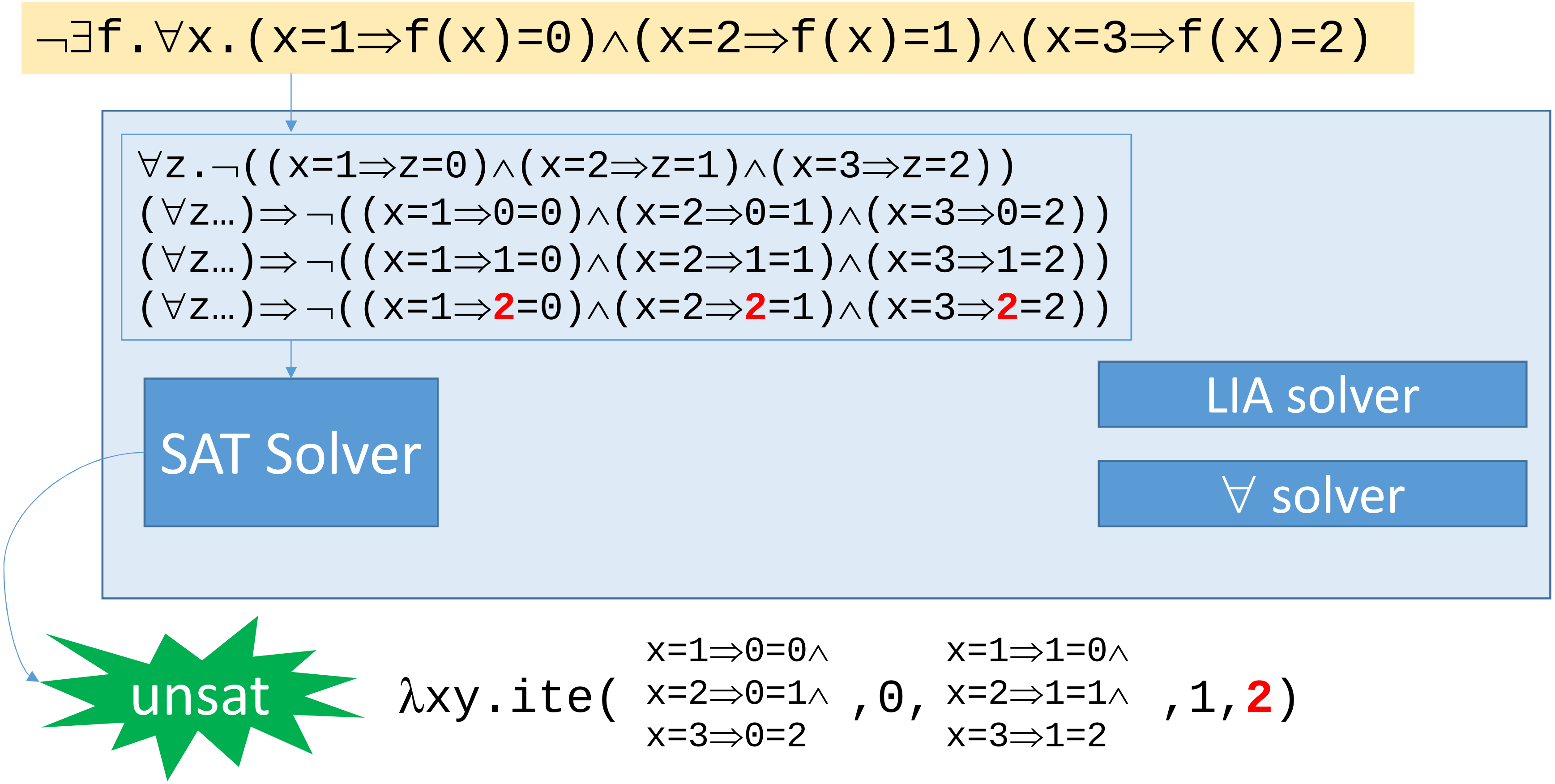
$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 0=0)\land(x=2\Rightarrow 0=1)\land(x=3\Rightarrow 0=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow \mathbf{1}=0)\land(x=2\Rightarrow \mathbf{1}=1)\land(x=3\Rightarrow \mathbf{1}=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 2=0)\land(x=2\Rightarrow 2=1)\land(x=3\Rightarrow 2=2))$

$$\lambda xy.\mathrm{ite}(\begin{array}{l}x=1\Rightarrow 0=0\land\\ x=2\Rightarrow 0=1\land\\ x=3\Rightarrow 0=2\end{array},0,\begin{array}{l}x=1\Rightarrow \mathbf{1}=0\land\\ x=2\Rightarrow \mathbf{1}=1\land\\ x=3\Rightarrow \mathbf{1}=2\end{array},\mathbf{1},\ldots)$$

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\wedge(x=2\Rightarrow f(x)=1)\wedge(x=3\Rightarrow f(x)=2)$

$\forall z.\neg((x=1\Rightarrow z=0)\wedge(x=2\Rightarrow z=1)\wedge(x=3\Rightarrow z=2))$

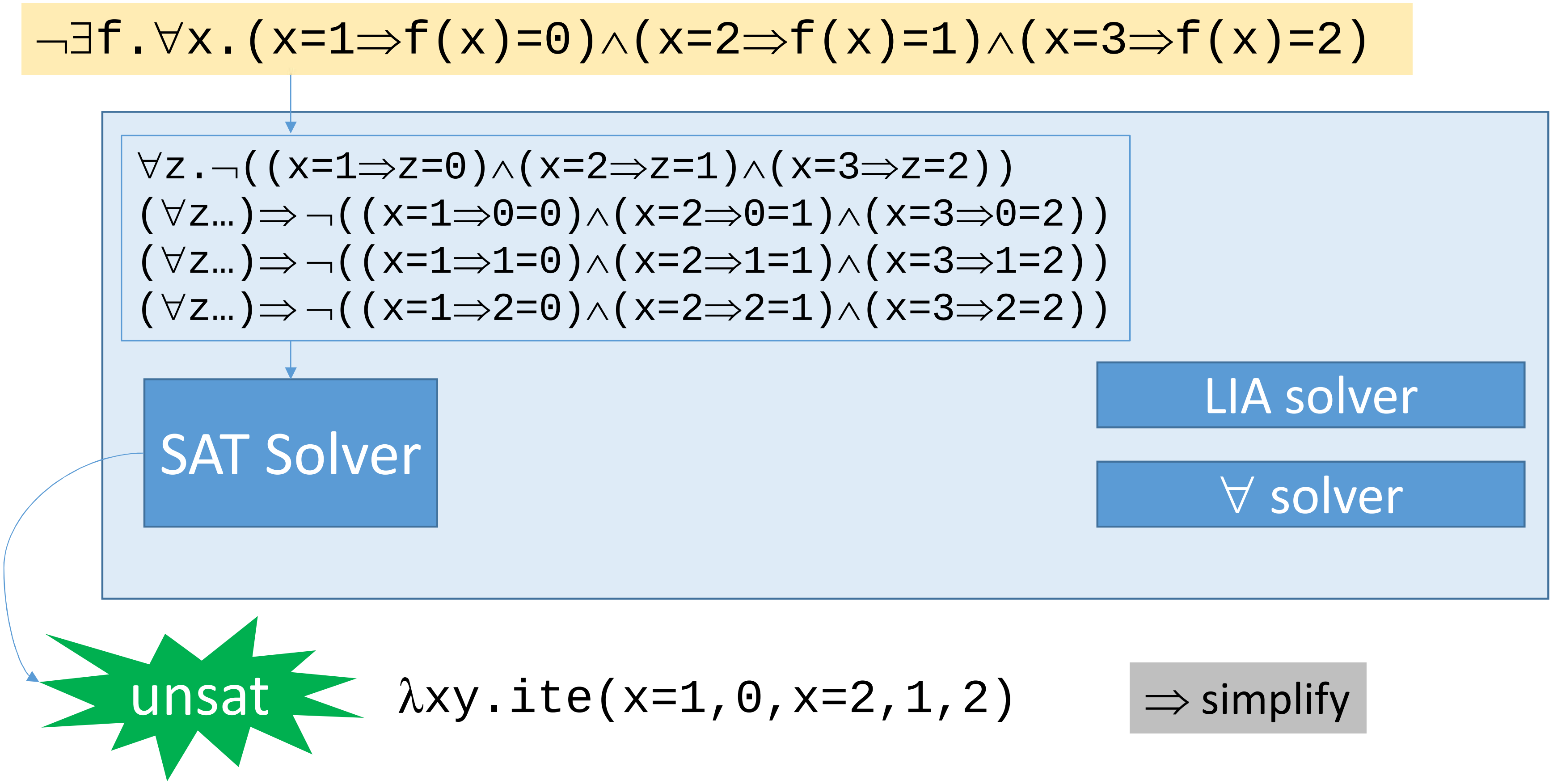
$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 0=0)\wedge(x=2\Rightarrow 0=1)\wedge(x=3\Rightarrow 0=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 1=0)\wedge(x=2\Rightarrow 1=1)\wedge(x=3\Rightarrow 1=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow \mathbf{2}=0)\wedge(x=2\Rightarrow \mathbf{2}=1)\wedge(x=3\Rightarrow \mathbf{2}=2))$

$$\lambda xy.\mathrm{ite}(\begin{array}{l}x=1\Rightarrow 0=0\wedge\\ x=2\Rightarrow 0=1\wedge\\ x=3\Rightarrow 0=2\end{array},0,\begin{array}{l}x=1\Rightarrow 1=0\wedge\\ x=2\Rightarrow 1=1\wedge\\ x=3\Rightarrow 1=2\end{array},1,\mathbf{2})$$

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\wedge(x=2\Rightarrow f(x)=1)\wedge(x=3\Rightarrow f(x)=2)$

$\forall z.\neg((x=1\Rightarrow z=0)\wedge(x=2\Rightarrow z=1)\wedge(x=3\Rightarrow z=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 0=0)\wedge(x=2\Rightarrow 0=1)\wedge(x=3\Rightarrow 0=2))$

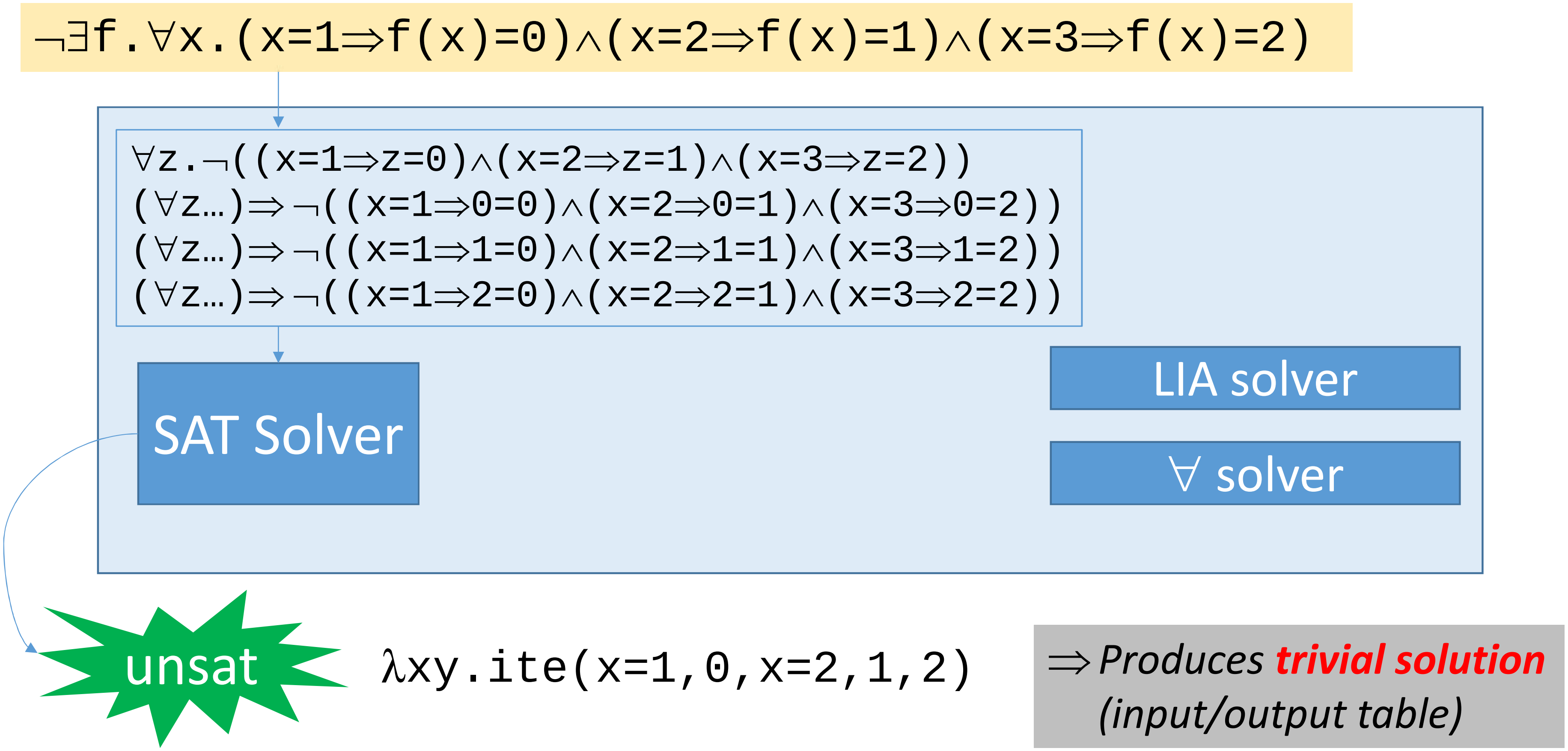
$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 1=0)\wedge(x=2\Rightarrow 1=1)\wedge(x=3\Rightarrow 1=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 2=0)\wedge(x=2\Rightarrow 2=1)\wedge(x=3\Rightarrow 2=2))$

$\lambda xy.\mathtt{ite}(x=1,0,x=2,1,2)$

⇒ simplify

What if we apply CEGQI to I/O Examples?



$\neg\exists f.\forall x.(x=1\Rightarrow f(x)=0)\wedge(x=2\Rightarrow f(x)=1)\wedge(x=3\Rightarrow f(x)=2)$

$\forall z.\neg((x=1\Rightarrow z=0)\wedge(x=2\Rightarrow z=1)\wedge(x=3\Rightarrow z=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 0=0)\wedge(x=2\Rightarrow 0=1)\wedge(x=3\Rightarrow 0=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 1=0)\wedge(x=2\Rightarrow 1=1)\wedge(x=3\Rightarrow 1=2))$

$(\forall z\ldots)\Rightarrow\neg((x=1\Rightarrow 2=0)\wedge(x=2\Rightarrow 2=1)\wedge(x=3\Rightarrow 2=2))$

SAT Solver

LIA solver

∀ solver

unsat

$\lambda xy.\mathrm{ite}(x=1,0,x=2,1,2)$

⇒ *Produces* ***trivial solution***
(input/output table)

Overview

With Syntactic Restrictions	? …although prefer solutions here	?	?
Without Syntactic Restrictions	CEGQI (trivially)	Counterexample Guided ∀-Instantiation	?
	Input/Output Examples	Single Invocation Conjectures	Other Second-Order Synthesis Conjectures

What if there are syntactic restrictions?

```
¬∃f.∀xy.isMax(f(x,y),x,y)
```

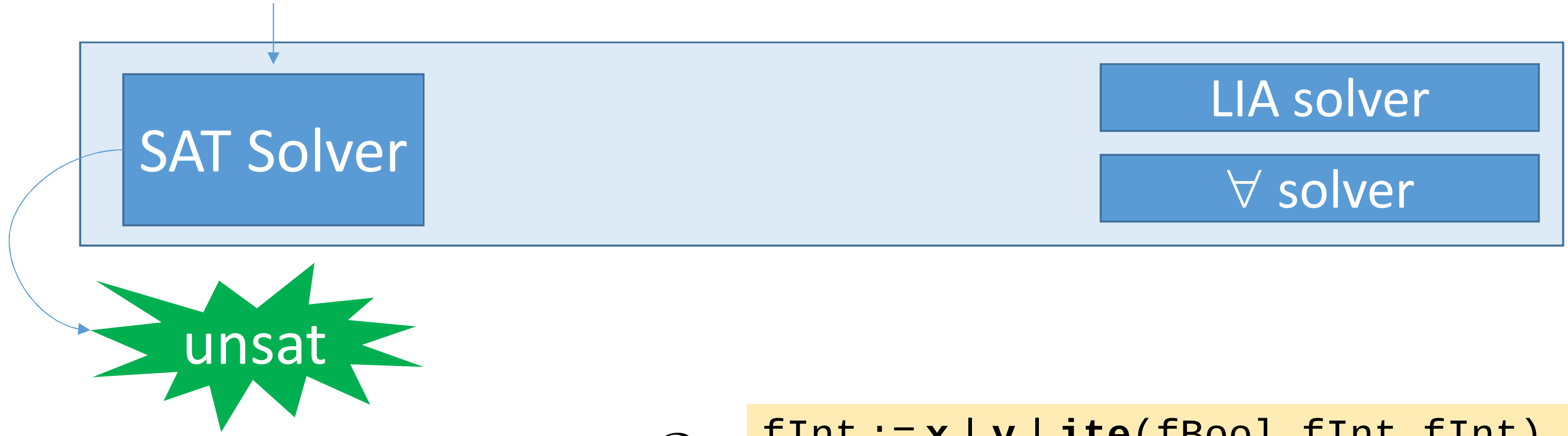
where solution meets syntactic restrictions ℜ :

ℜ :

```
fInt := x | y | ite(fBool,fInt,fInt)
fBool := >(fInt,fInt) | =(fInt,fInt) | Ė(fBool)
```

What if there are syntactic restrictions?

`¬∃f.∀xy.isMax(f(x,y),x,y)`

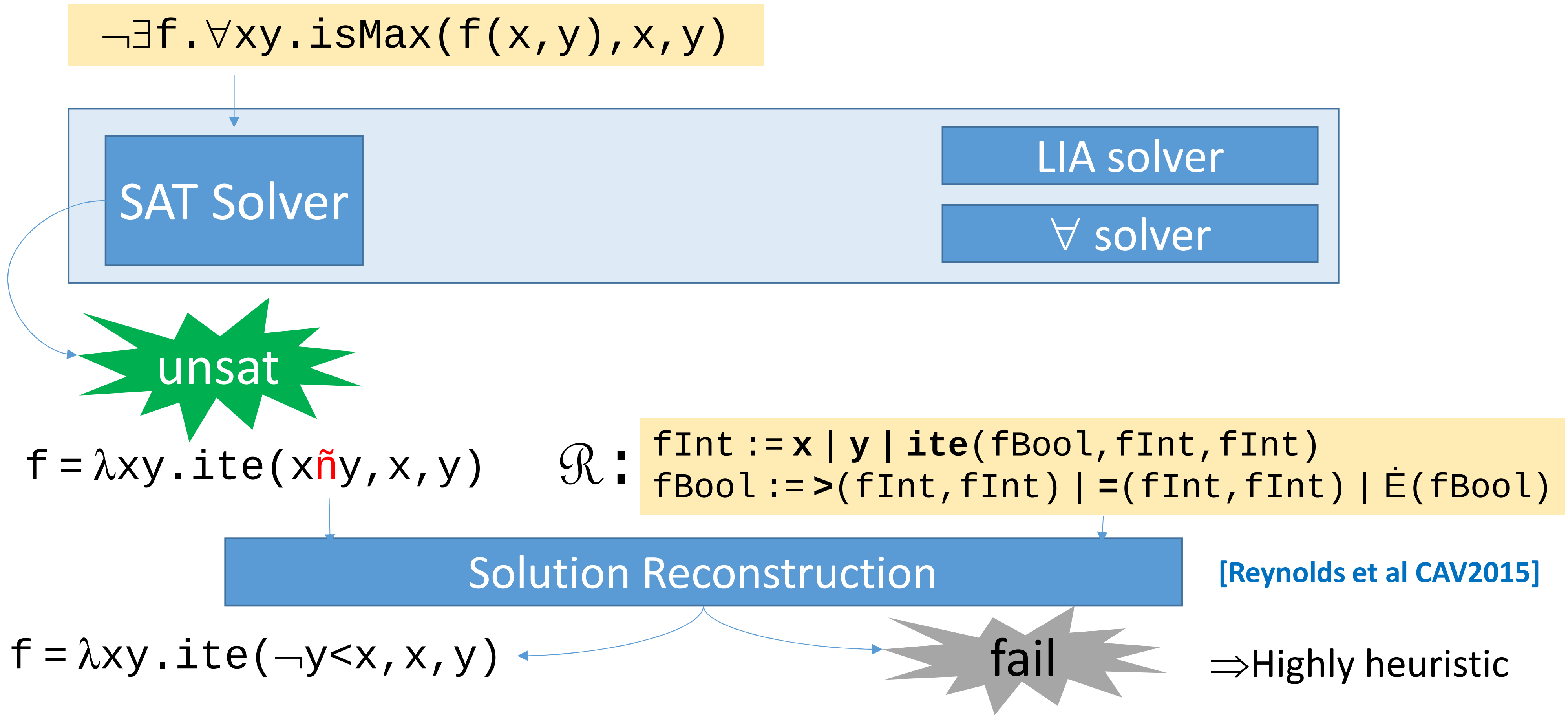


`f = λxy.ite(xñy,x,y)`

ℜ:

fInt := **x** | **y** | **ite**(fBool,fInt,fInt)
fBool := **>**(fInt,fInt) | **=**(fInt,fInt) | Ė(fBool)

What if there are syntactic restrictions?



¬∃f.∀xy.isMax(f(x,y),x,y)

SAT Solver

LIA solver

∀ solver

unsat

f = λxy.ite(xñy,x,y)

$\mathfrak{R}$: fInt := **x** | **y** | **ite**(fBool,fInt,fInt)
fBool := **>**(fInt,fInt) | **=**(fInt,fInt) | Ė(fBool)

Solution Reconstruction

f = λxy.ite(¬y<x,x,y)

fail

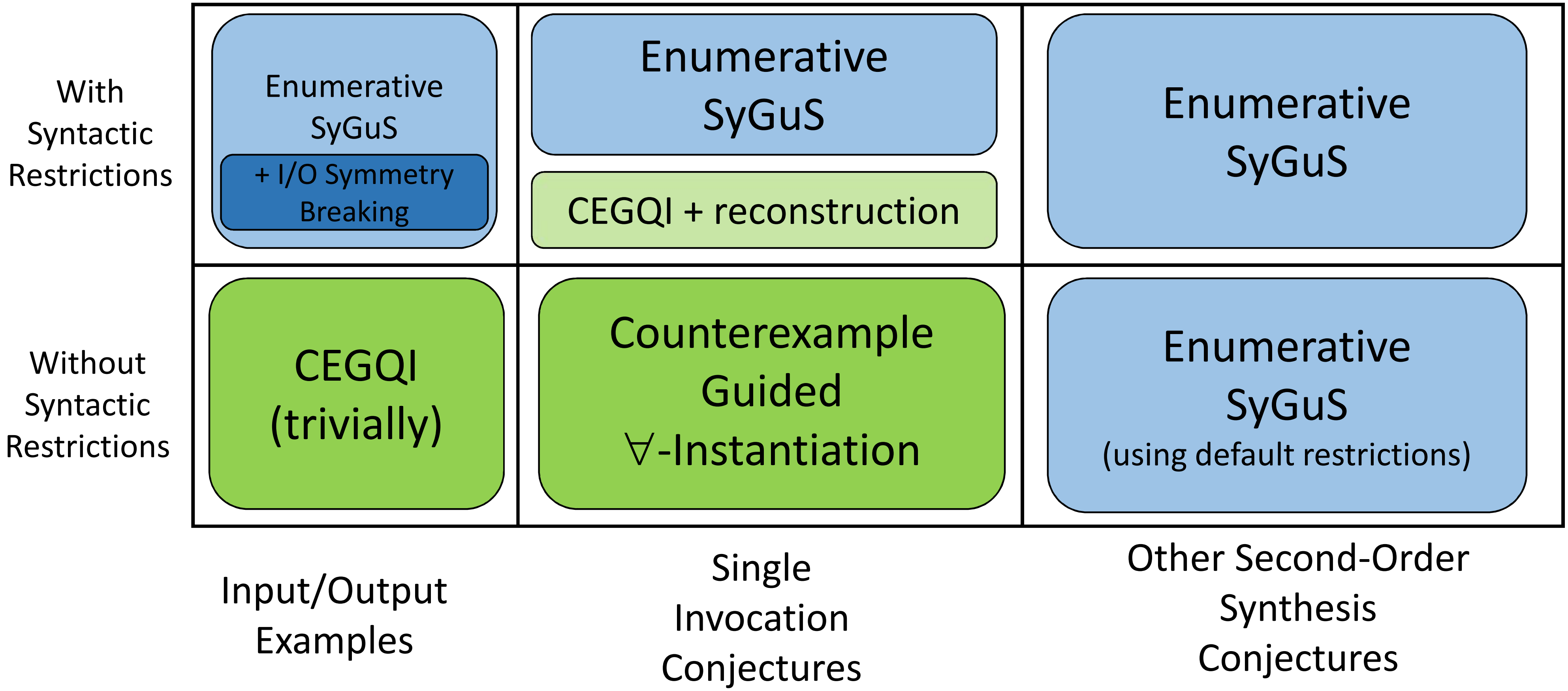
[Reynolds et al CAV2015]

⇒Highly heuristic

Overview

With Syntactic Restrictions	?	? CEGQI + reconstruction	?
Without Syntactic Restrictions	CEGQI (trivially)	Counterexample Guided ∀-Instantiation	?
	Input/Output Examples	Single Invocation Conjectures	Other Second-Order Synthesis Conjectures

Techniques used by CVC4 for Synthesis



	Input/Output Examples	Single Invocation Conjectures	Other Second-Order Synthesis Conjectures
With Syntactic Restrictions	Enumerative SyGuS + I/O Symmetry Breaking	Enumerative SyGuS CEGQI + reconstruction	Enumerative SyGuS
Without Syntactic Restrictions	CEGQI (trivially)	Counterexample Guided ∀-Instantiation	Enumerative SyGuS (using default restrictions)

Function Synthesis

```
int min_comm(int x, int y)
{
     ???
}
```

@ensures: $@ret \leq x_{in}-1 \land \forall xy.\mathtt{min\_comm}(x,y)=\mathtt{min\_comm}(y,x)$

`min_comm` is a commutative function

EXAMPLE A5…

Function Synthesis

```
int min_comm(int x, int y)
{
     if(xáy){
       return x-1;
     }else{
       return y-1;
     }
}
@ensures: @ret≤x_in-1 ∧ ∀xy.min_comm(x,y)=min_comm(y,x)
```

⇒ Use enumerative techniques in the core of the SMT solver **[Reynolds et al 2015]**

Challenge Problem: Invariant Synthesis

@precondition: **I**$[x_{in}, y_{in}]$

```
void update(int& x, int& y){
     x := x+2;
     y := y+1;
}
```

@ensures: **I**$[x_{out}, y_{out}]$

@precondition: $x_{in}=5 \wedge y_{in}=2$

```
void updatew(int& x, int& y){
     while(y<50){
       update(x,y);
     }
}
```

@ensures: **x_{out}ñ100 ?**

EXAMPLE A6…

Challenge Problem: Invariant Synthesis

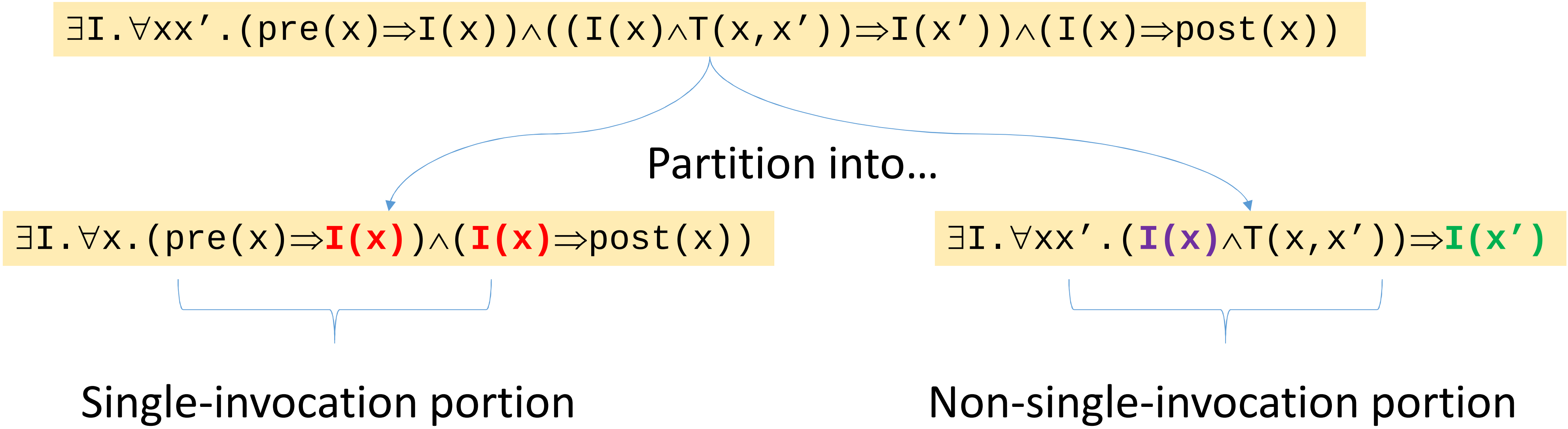
```
@precondition: x_inñ2*y_in
void update(int& x, int& y){
     x := x+2;
     y := y+1;
}
@ensures: x_outñ2*y_out
```

```
@precondition: x_in=5 ∧ y_in=2
void updatew(int& x, int& y){
     while(y<50){
       update(x,y);
     }
}
@ensures: x_outñ100
```

What if conjecture is *Partially Single Invocation?*

$$\underbrace{\exists \mathtt{I}.\forall \mathtt{x}\mathtt{x}'.(\mathtt{pre}(\mathtt{x})\Rightarrow \mathtt{I}(\mathtt{x}))\wedge((\mathtt{I}(\mathtt{x})\wedge \mathtt{T}(\mathtt{x},\mathtt{x}'))\Rightarrow \mathtt{I}(\mathtt{x}'))\wedge(\mathtt{I}(\mathtt{x})\Rightarrow \mathtt{post}(\mathtt{x}))}_{\text{E.g. invariant synthesis problem for } \mathtt{I} \text{ w.r.t } \mathtt{pre}\text{, } \mathtt{T}\text{, } \mathtt{post}}$$

What if conjecture is *Partially Single Invocation?*



$\exists I. \forall x x'. (pre(x) \Rightarrow I(x)) \wedge ((I(x) \wedge T(x,x')) \Rightarrow I(x')) \wedge (I(x) \Rightarrow post(x))$

Partition into…

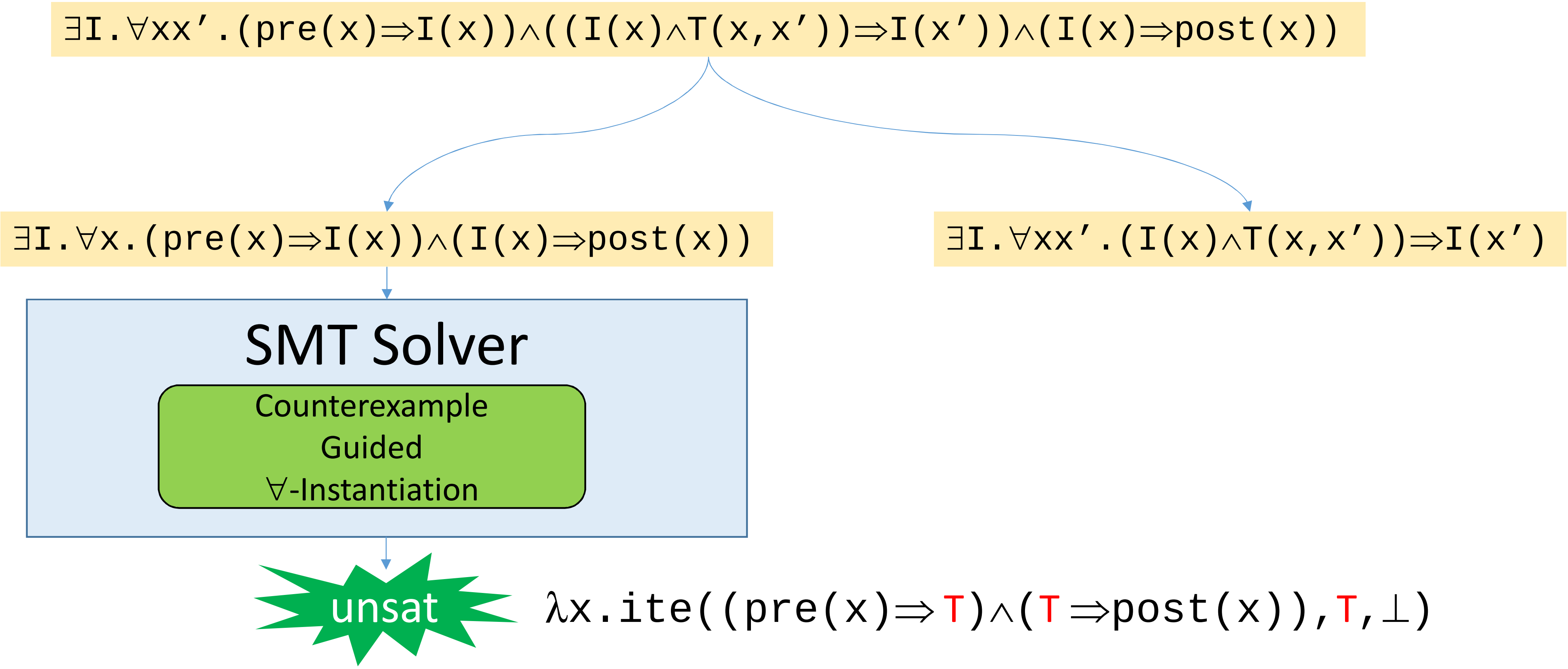
$\exists I. \forall x. (pre(x) \Rightarrow \mathbf{I(x)}) \wedge (\mathbf{I(x)} \Rightarrow post(x))$

$\exists I. \forall x x'. (\mathbf{I(x)} \wedge T(x,x')) \Rightarrow \mathbf{I(x')}$

Single-invocation portion

Non-single-invocation portion

What if conjecture is *Partially Single Invocation?*



$$\exists I. \forall x x'. (pre(x) \Rightarrow I(x)) \wedge ((I(x) \wedge T(x,x')) \Rightarrow I(x')) \wedge (I(x) \Rightarrow post(x))$$

$$\exists I. \forall x. (pre(x) \Rightarrow I(x)) \wedge (I(x) \Rightarrow post(x))$$

$$\exists I. \forall x x'. (I(x) \wedge T(x,x')) \Rightarrow I(x')$$

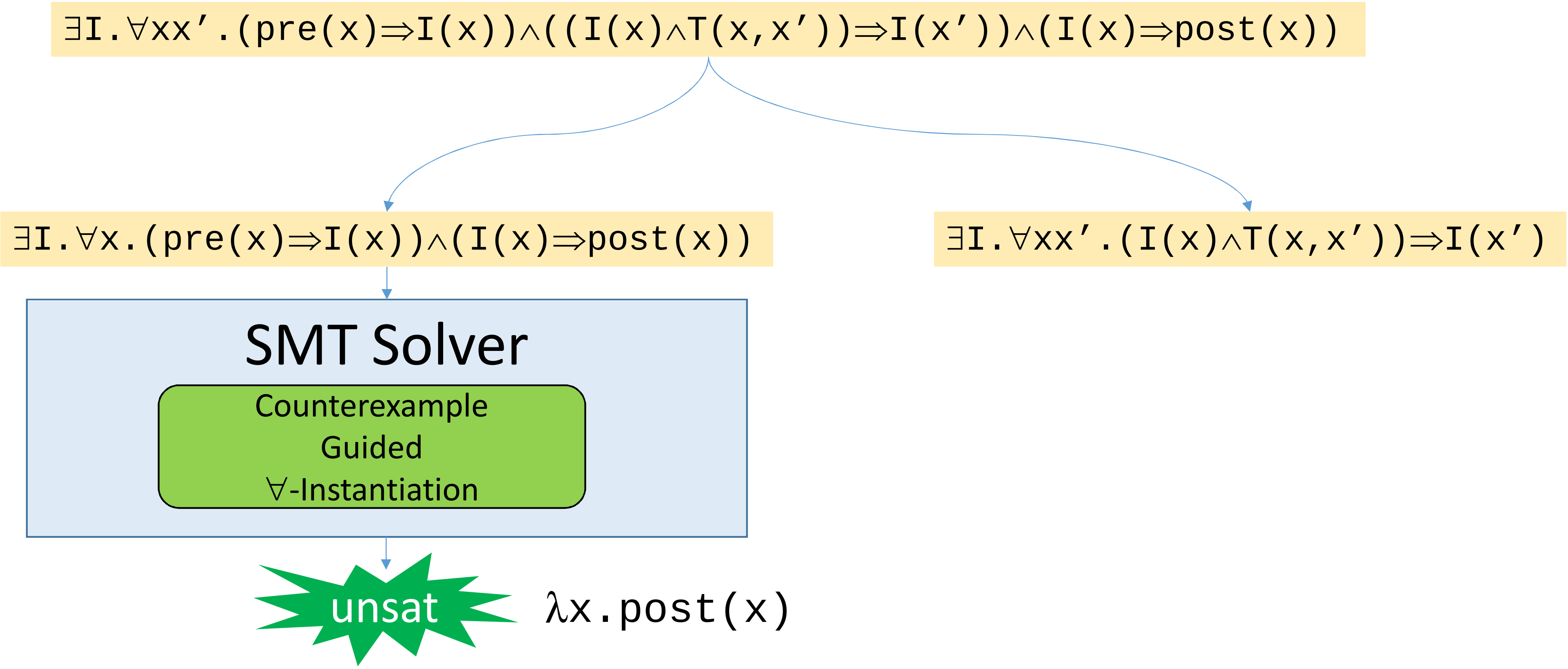
SMT Solver

Counterexample Guided ∀-Instantiation

unsat

$$\lambda x. ite((pre(x) \Rightarrow T) \wedge (T \Rightarrow post(x)), T, \bot)$$

What if conjecture is *Partially Single Invocation?*



$\exists I.\forall x x'.(pre(x)\Rightarrow I(x))\wedge((I(x)\wedge T(x,x'))\Rightarrow I(x'))\wedge(I(x)\Rightarrow post(x))$

$\exists I.\forall x.(pre(x)\Rightarrow I(x))\wedge(I(x)\Rightarrow post(x))$

$\exists I.\forall x x'.(I(x)\wedge T(x,x'))\Rightarrow I(x')$

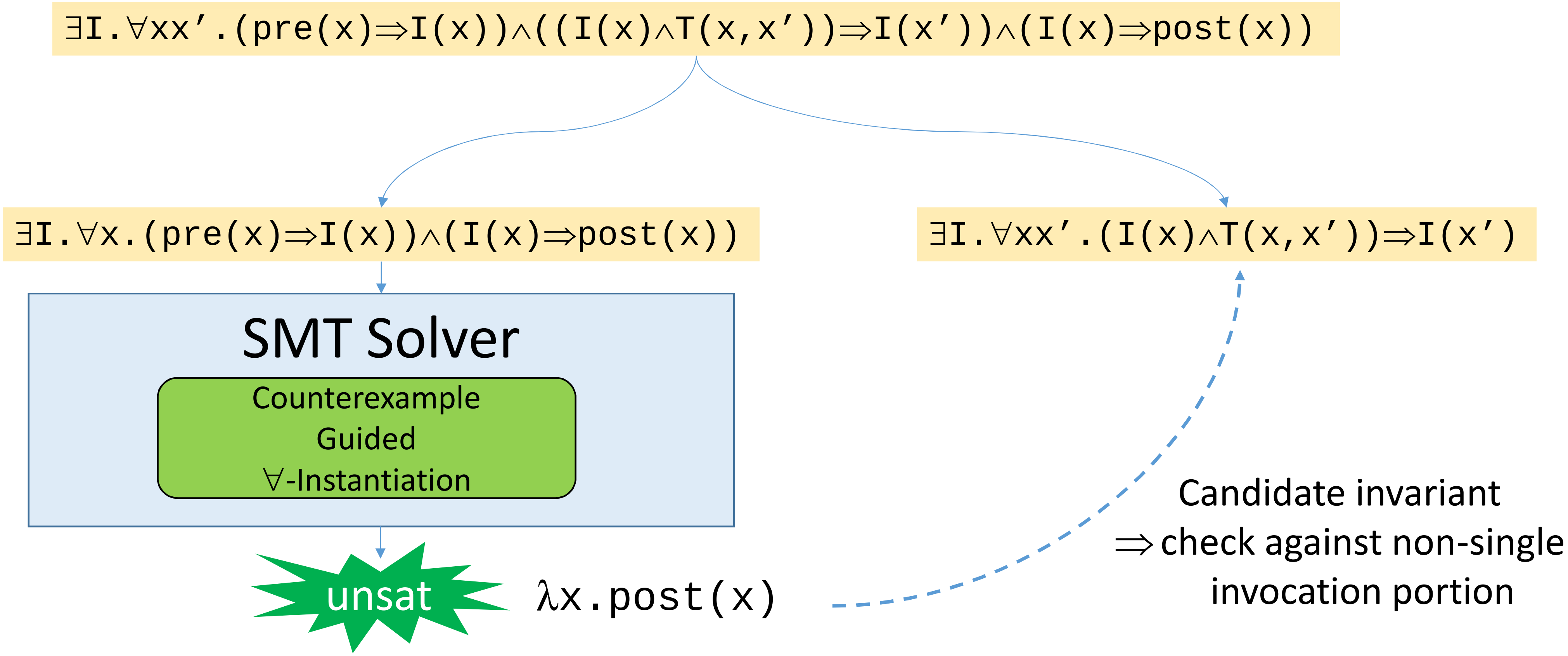
SMT Solver

Counterexample Guided ∀-Instantiation

unsat

$\lambda x.post(x)$

What if conjecture is *Partially Single Invocation*?



$\exists I.\forall xx'.(pre(x)\Rightarrow I(x))\wedge((I(x)\wedge T(x,x'))\Rightarrow I(x'))\wedge(I(x)\Rightarrow post(x))$

$\exists I.\forall x.(pre(x)\Rightarrow I(x))\wedge(I(x)\Rightarrow post(x))$

$\exists I.\forall xx'.(I(x)\wedge T(x,x'))\Rightarrow I(x')$

SMT Solver

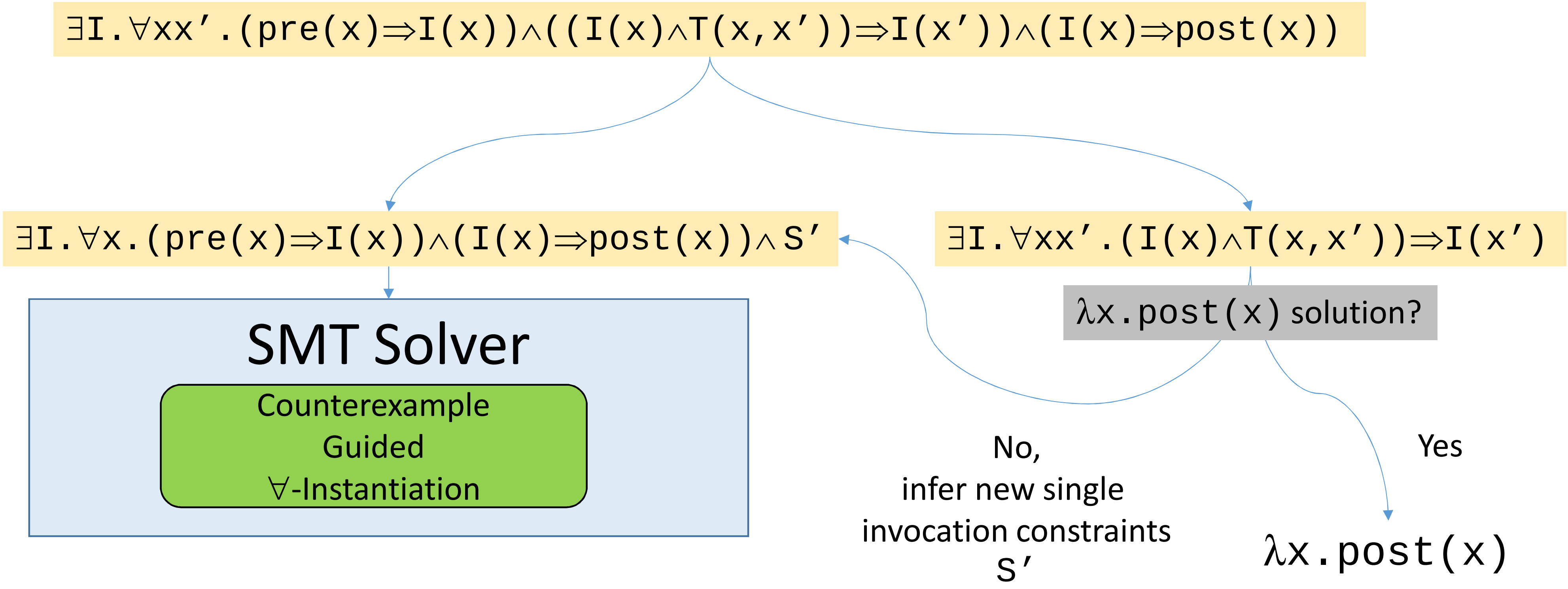
Counterexample Guided ∀-Instantiation

unsat

$\lambda x.post(x)$

Candidate invariant
⇒ check against non-single invocation portion

What if conjecture is *Partially Single Invocation?*



$\exists I.\forall xx'.(pre(x)\Rightarrow I(x))\wedge((I(x)\wedge T(x,x'))\Rightarrow I(x'))\wedge(I(x)\Rightarrow post(x))$

$\exists I.\forall x.(pre(x)\Rightarrow I(x))\wedge(I(x)\Rightarrow post(x))\wedge S'$

$\exists I.\forall xx'.(I(x)\wedge T(x,x'))\Rightarrow I(x')$

SMT Solver

Counterexample Guided ∀-Instantiation

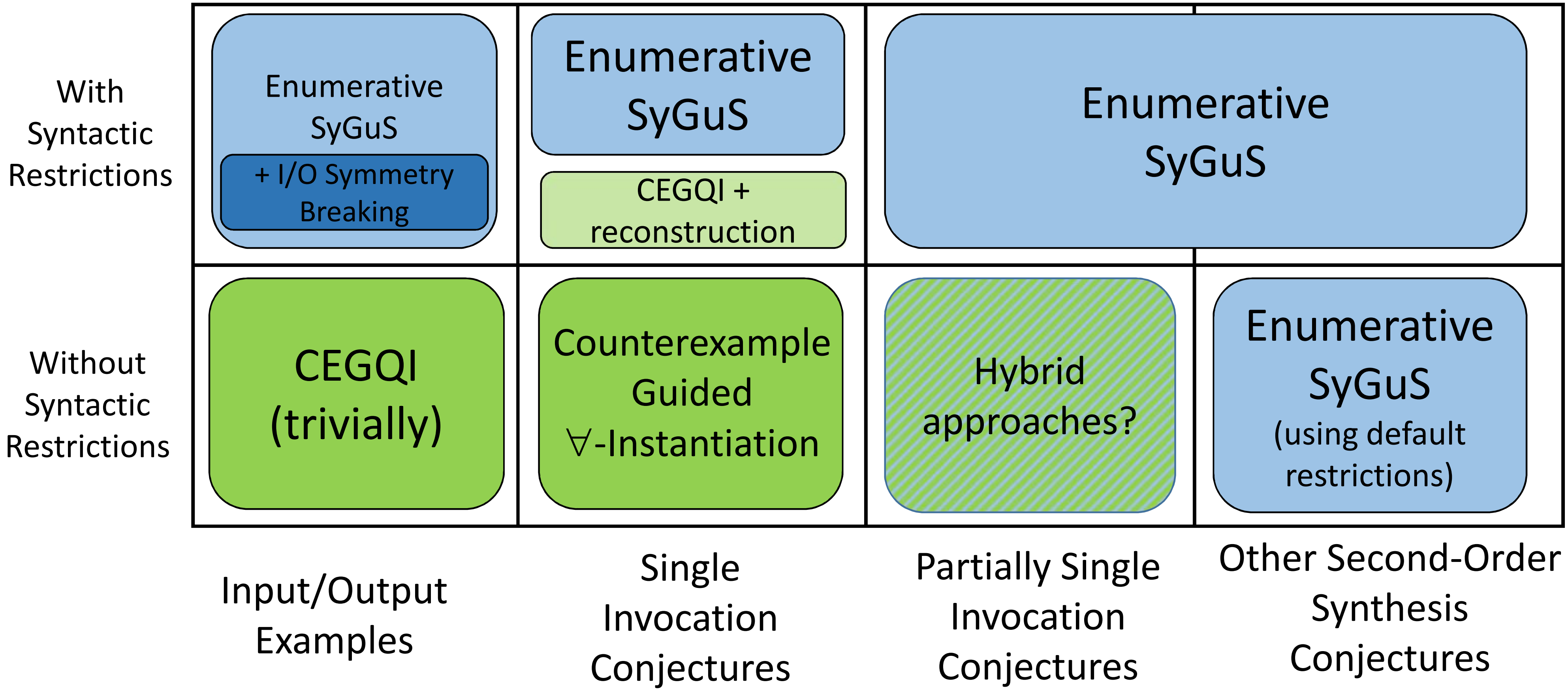
$\lambda x.post(x)$ solution?

No, infer new single invocation constraints S'

Yes

$\lambda x.post(x)$

Techniques for Synthesis



	Input/Output Examples	Single Invocation Conjectures	Partially Single Invocation Conjectures	Other Second-Order Synthesis Conjectures
With Syntactic Restrictions	Enumerative SyGuS + I/O Symmetry Breaking	Enumerative SyGuS CEGQI + reconstruction	Enumerative SyGuS	
Without Syntactic Restrictions	CEGQI (trivially)	Counterexample Guided ∀-Instantiation	Hybrid approaches?	Enumerative SyGuS (using default restrictions)

- …Thanks for listening!

- Techniques from these lectures available in master branch of CVC4:
  - Open source
  - Available at : http://cvc4.cs.stanford.edu/web/
  - Accepts *.smt2, *.sy formats