Instantiation for Quantified Formulas in SMT: Techniques and Practical Aspects

Andrew Reynolds

June 24, 2016

The University of Iowa

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:

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

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

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
  - ***Existential and Universal Quantifiers***

Outline

- Background
- SMT solver architecture

…and how it extends to ∀ reasoning via quantifier instantiation:

$$\forall x . \psi[x] \Rightarrow \psi[t]$$

- Recent strategies for quantifier instantiation:
  - E-matching, conflict-based, model-based, counterexample-guided
- Challenges, future work

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:
    - Conjectures $\forall i{:}input.\exists o{:}output.R[o,i]$
  - Planning:
    - Specifications $\exists p{:}plan.\forall t{:}time.F[P,t]$

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:
    - Conjectures $\forall i{:}input.\exists o{:}output.R[o,i]$
  - Planning:
    - Specifications $\exists p{:}plan.\forall t{:}time.F[P,t]$
- Are very challenging in theory:
  - Establishing T-satisfiability of formulas with ∀ is generally undecidable

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:
    - Conjectures $\forall i:input.\exists o:output.R[o,i]$
  - Planning:
    - Specifications $\exists p:plan.\forall t:time.F[P,t]$
- Are very challenging in theory:
  - Establishing T-satisfiability of formulas with ∀ is generally undecidable
- Can be handled well in practice:
  - Efficient decision procedures for decidable fragments, e.g. Bernays-Shonfinkel
  - Heuristic techniques have high success rates in the general case

Background: *Theory*

- A *theory* T is a pair (Σ_T, $\mathbf{I}_T$), where:
  - Σ_T is set of function symbols, the *signature* of T
    - E.g. Σ_{LIA} = { `+`, `-`, `<`, `≤`, `>`, `≥`, `0`, `1`, `2`, `3`, … }
  - $\mathbf{I}_T$ is a set of *interpretations*
    - E.g. each $\mathbf{I} \in \mathbf{I}_{LIA}$ interpret functions in Σ_{LIA} in standard way:
      - `1+1=2`, `1+2=3`, `1>0` = $\top$, `0>1` = $\bot$, …
    - Interpretation of free constants chosen arbitrarily
- A formula Ψ is T-satisfiable if there is an $\mathbf{I} \in \mathbf{I}_T$ that interprets Ψ as T
  - We call **I** a *model* of Ψ
    - E.g the formula (`a+1>b`) is LIA-satisfiable with a model **I** where **I** `(a)=2` and **I** `(b)=0`

Background: *Quantifiers*

- Universal quantification:

$$\forall \texttt{x:Int.P(x)}$$

P is true for all integers x

- Existential quantification:

$$\exists \texttt{x:Int.}\neg\texttt{Q(x)}$$

Q is false for some integer x

Background: *Quantifiers*

- Universal quantification:

$$\forall \texttt{x:Int.P(x)}$$

P is true for all integers x

- Existential quantification:

$$\exists \texttt{x:Int.}\neg\texttt{Q(x)} \quad \rightarrow \quad \neg\forall \texttt{x:Int.Q(x)}$$

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

Theoretical Complexity

- Checking T-satisfiability of:

$$(\forall x. P(x) \lor Q(x) \lor x{=}a) \land P(b) \land Q(c)$$

  - Bernays-Shonfinkel (function-free + equality) is decidable (NEXPTIME)

$$(\forall xy. \exists z. x{+}y{+}z{>}2 \lor 0 \leq z{+}x)$$

  - Case of ∀ in pure theories is often decidable, e.g. linear arithmetic

$$(\forall x. P(x) \Rightarrow P(x{+}1)) \land P(a) \land \neg P(b) \land a{<}b$$

  - However, general case is undecidable!

Approaches for Satisfiability of ∀ in Tools

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

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

Approaches for Satisfiability of ∀ in Tools

- 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 in Vampire **[Voronkov 14, Reger et al 15]**
  - **iProver**
    - InstGen calculus **[Ganzinger/Korovin 03]**
  - **Princess, Beagle**, …
- SMT solvers focus mostly on ground theory reasoning
  …but have been extended in the past decade to ∀ reasoning:

Approaches for Satisfiability of ∀ in Tools

- 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 in Vampire **[Voronkov 14, Reger et al 15]**
  - **iProver**
    - InstGen calculus **[Ganzinger/Korovin 03]**
  - **Princess, Beagle**, …
- SMT solvers focus mostly on ground 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 03, deMoura et al 07, Ge et al 07, …]**

Approaches for Satisfiability of ∀ in Tools

- 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 in Vampire **[Voronkov 14, Reger et al 15]**
  - **iProver**
    - InstGen calculus **[Ganzinger/Korovin 03]**
  - **Princess, Beagle**, …
- SMT solvers focus mostly on ground 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 03, deMoura et al 07, Ge et al 09, …]**

$\Rightarrow$ Focus of this talk

Quantified Formulas in DPLL(T): Basics

$$(P(a) \lor f(b)=a+1)$$
$$(\neg\forall x.P(x) \lor \forall y.\neg P(y) \lor R(y))$$
$$(\forall x.f(x)=g(x)+h(x) \lor \neg R(a))$$

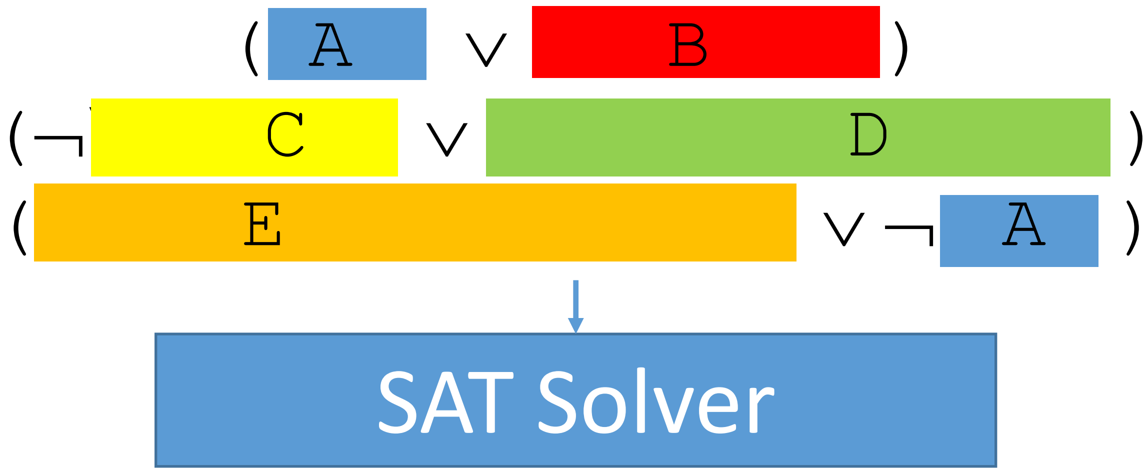
$\Rightarrow$ Given the above input

Quantified Formulas in DPLL(T): Basics

$$(P(a) \lor f(b) > a+1)$$
$$(\neg \forall x. P(x) \lor \forall y. \neg P(y) \lor R(y))$$
$$(\forall x. f(x) = g(x) + h(x) \lor \neg P(a))$$

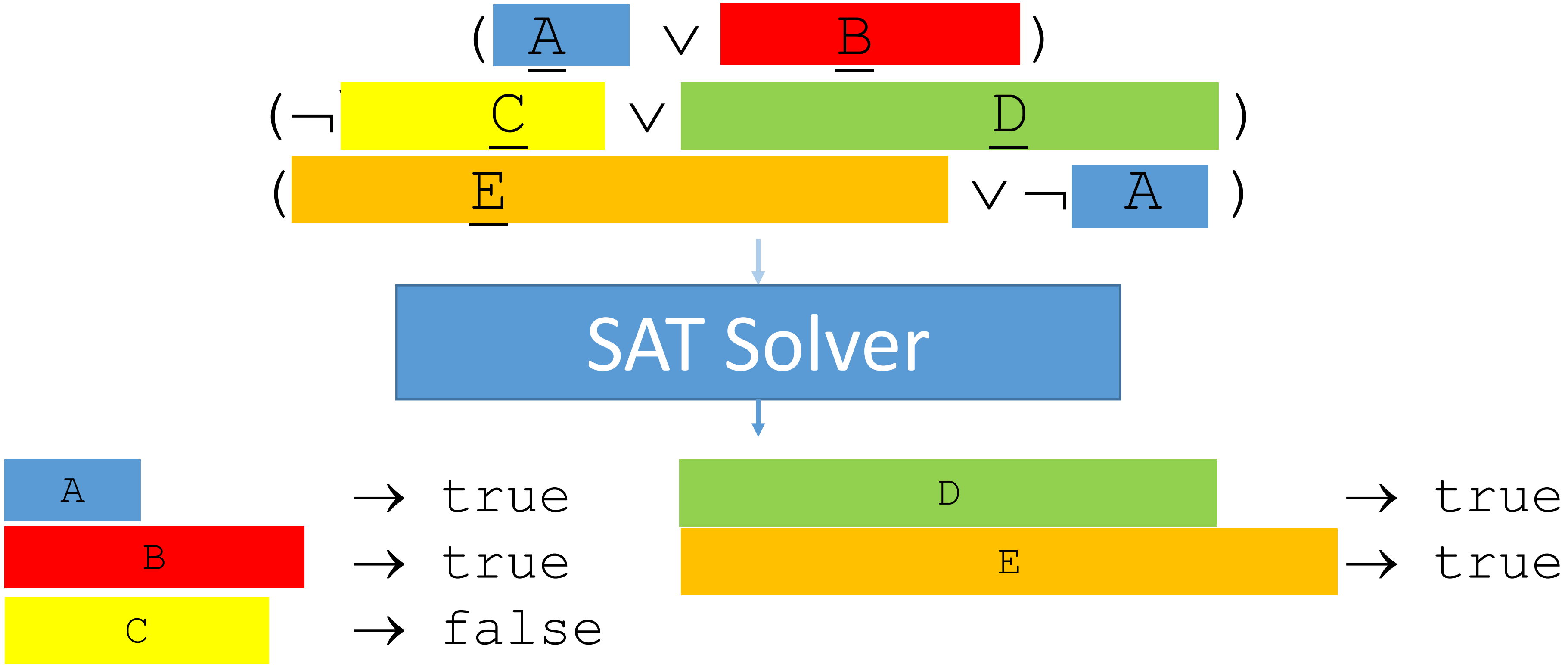
- Consider the propositional abstraction of the formula
  - Atoms may encapsulate quantified formulas with Boolean structure
    - E.g. $\forall y. \neg P(y) \lor R(y)$

Quantified Formulas in DPLL(T): Basics



- Find propositional satisfying assignment via off-the-shelf SAT solver

Quantified Formulas in DPLL(T): Basics



- Find propositional satisfying assignment via off-the-shelf SAT solver

Quantified Formulas in DPLL(T): Basics

$(P(a) \vee f(b)>a+1)$

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

$(\forall x.f(x)=g(x)+h(x) \vee \neg P(a))$

↓

SAT Solver

↓

$P(a) \rightarrow$ true		$\forall y.\neg P(y) \vee R(y) \rightarrow$ true	
$f(b)>a+1 \rightarrow$ true		$\forall x.f(x)=g(x)+h(x) \rightarrow$ true	
$\forall x.P(x) \rightarrow$ false			

$\Rightarrow$ Consider original atoms

Quantified Formulas in DPLL(T): Basics

(P(a) ∨ f(b)>a+1)

(¬∀x.P(x) ∨ ∀y.¬P(y) ∨ R(y))

(∀x.f(x)=g(x)+h(x) ∨ ¬P(a))

↓

SAT Solver

↓

P(a), f(b)>a+1, ¬∀x.P(x), ∀x.f(x)=g(x)+h(x), ∀y.¬P(y) ∨ R(y)

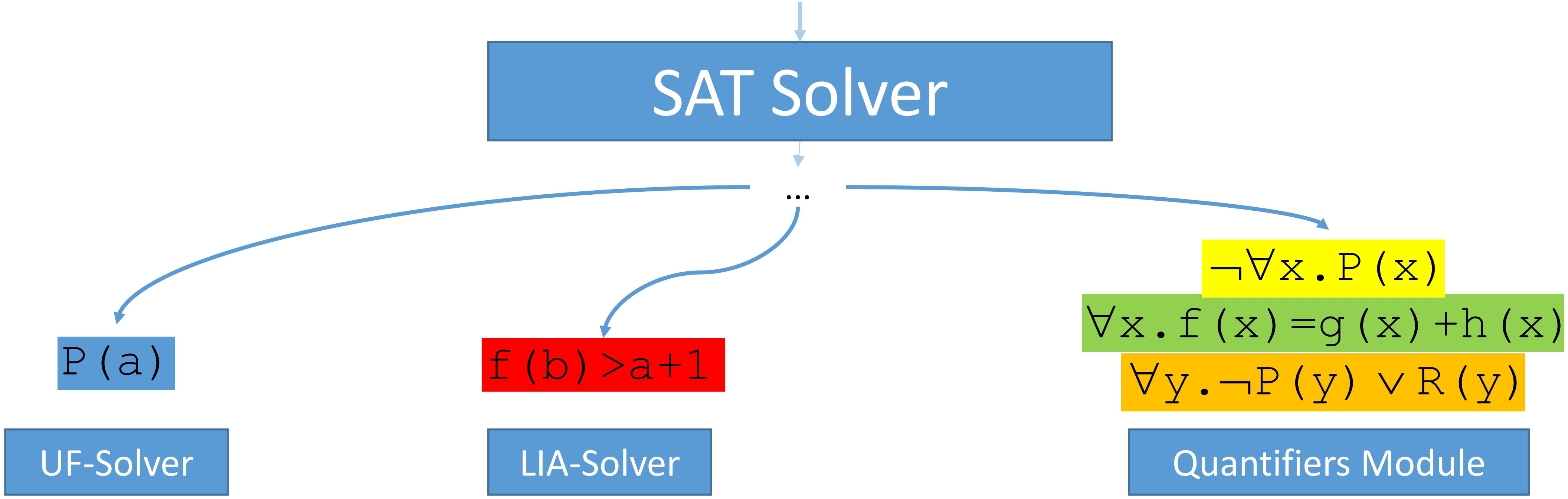
M

⇒ Propositional assignment can be seen as a set of T-literals M

- Must check if M is T-satisfiable

Quantified Formulas in DPLL(T): Basics

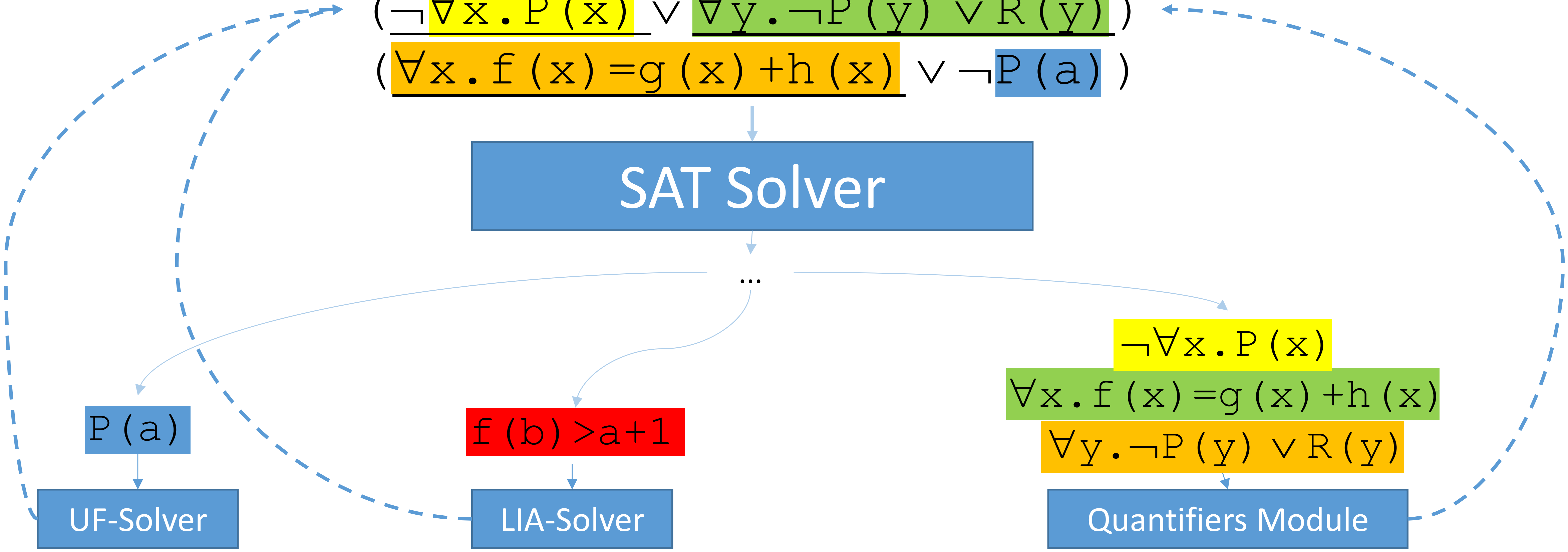
$$(P(a) \lor f(b)>a+1)$$
$$(\neg\forall x.P(x) \lor \forall y.\neg P(y) \lor R(y))$$
$$(\forall x.f(x)=g(x)+h(x) \lor \neg P(a))$$



⇒ Distribute ground literals to T-solvers, ∀ literals to quantifiers module

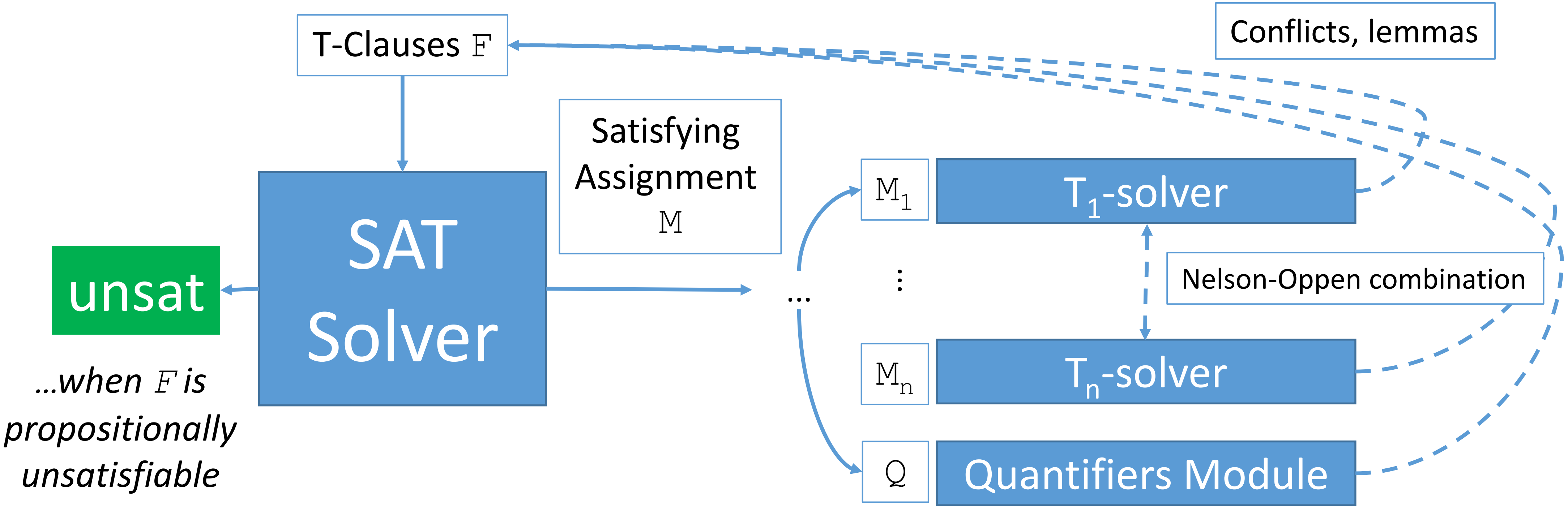
Quantified Formulas in DPLL(T): Basics

$$(P(a) \vee f(b)>a+1)$$
$$(\neg \forall x.P(x) \vee \forall y.\neg P(y) \vee R(y))$$
$$(\forall x.f(x)=g(x)+h(x) \vee \neg P(a))$$



$\Rightarrow$ These solvers may choose to add conflicts/lemmas to clause set

DPLL(T_1+..+T_n)+Quantifiers: Overview

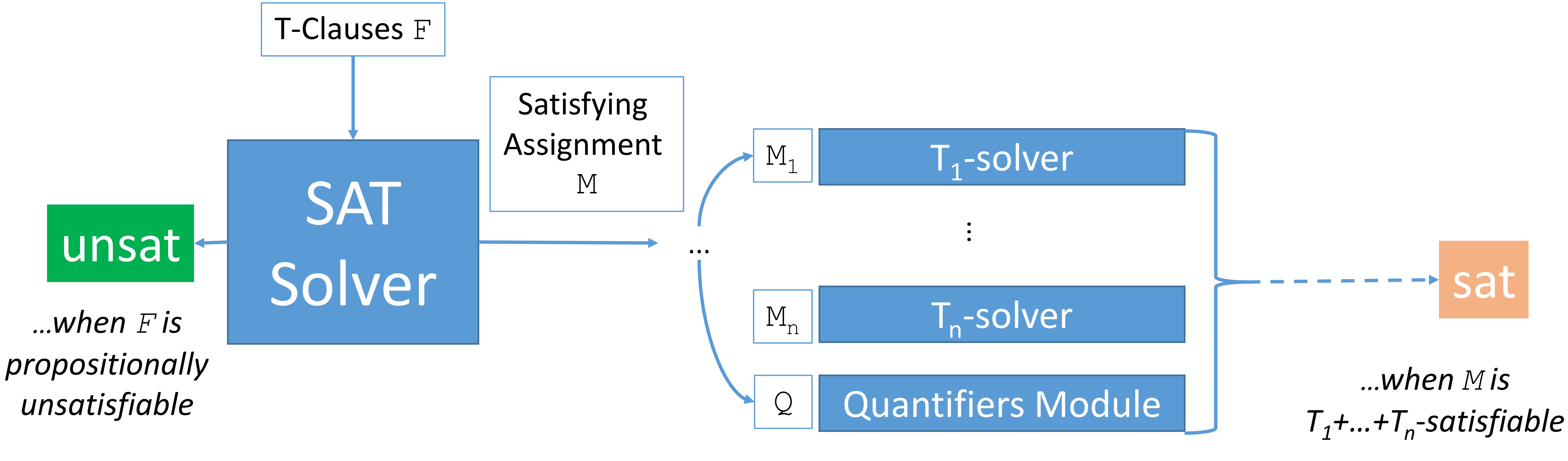


⇒ Each of these components may:

- Report M is T-unsatisfiable by reporting conflict clauses
- Report lemmas if they are unsure

[Nieuwenhuis/Oliveras/Tinelli 06]

DPLL(T_1+..+T_n)+Quantifiers: Overview



⇒ If no component adds a lemma, then it must be the case that M is T_1+…+T_n-satisfiable

[Nieuwenhuis/Oliveras/Tinelli 06]

DPLL(T_1+..+T_n)+Quantifiers: Overview

T-Clauses $\mathtt{F}$

Unlike the ground case where decision procedures exist for T_1, …, T_n,
…there is **no general decision procedure** for ∀-formulas $\mathtt{Q}$, thus:

- This procedure **may not terminate**!
- Regardless, we want techniques that:
  - Are refutation-sound (“unsat” can be trusted)
  - Are model-sound (“sat” can be trusted)
  - Terminate for many $\mathtt{F}$

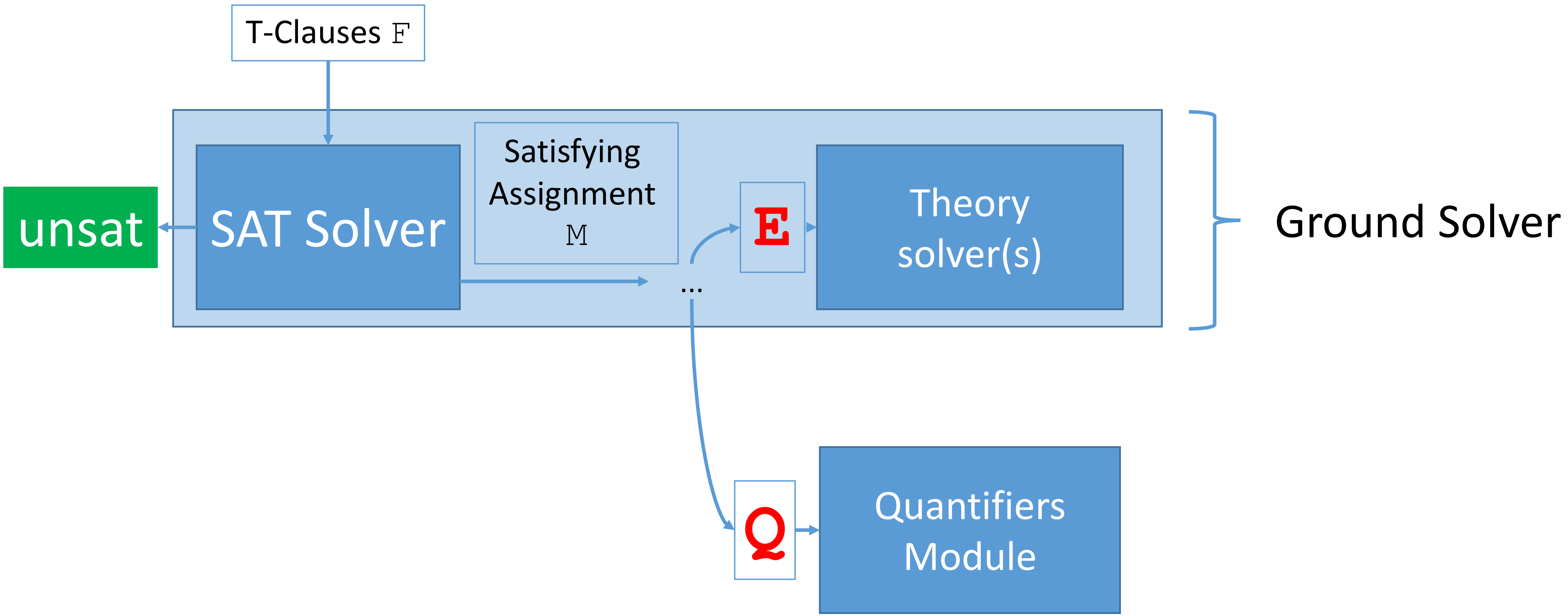
unsat

…when F is propositionally unsatisfiable

sat

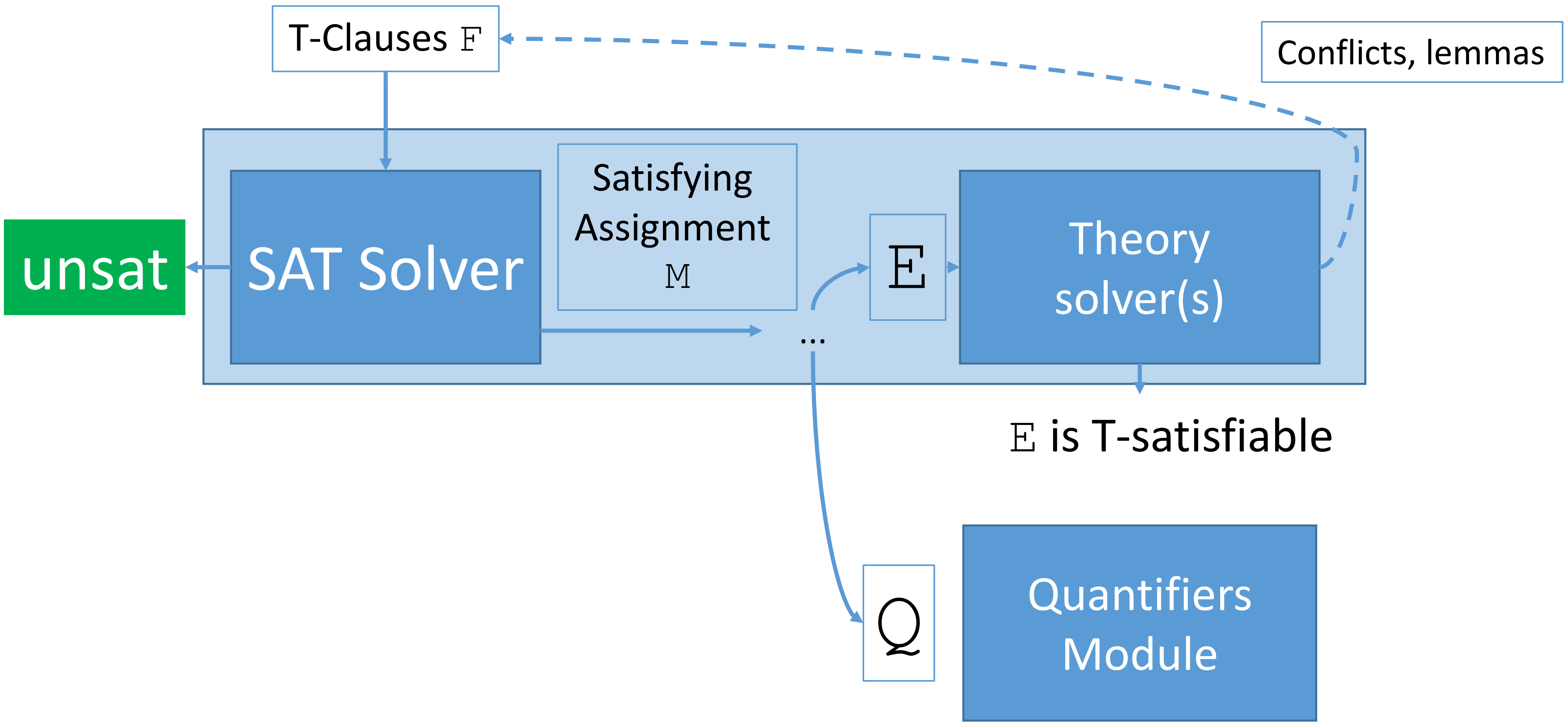
…when M is T_1+…+T_n-satisfiable

In this talk: DPLL(T)+Quantifiers, simplified



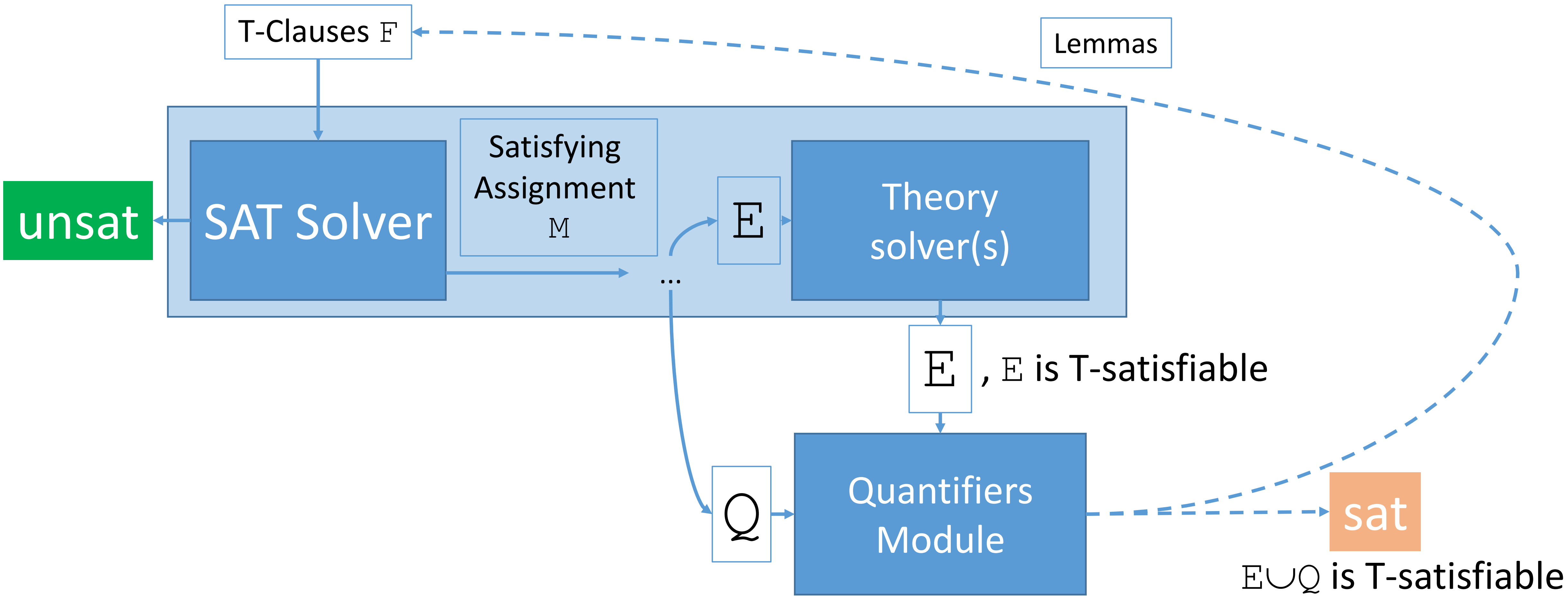
⇒ For purposes of this talk, partition M into quantifier-free part **E**, and set of ∀ formulas **Q**

In this talk: DPLL(T)+Quantifiers, simplified



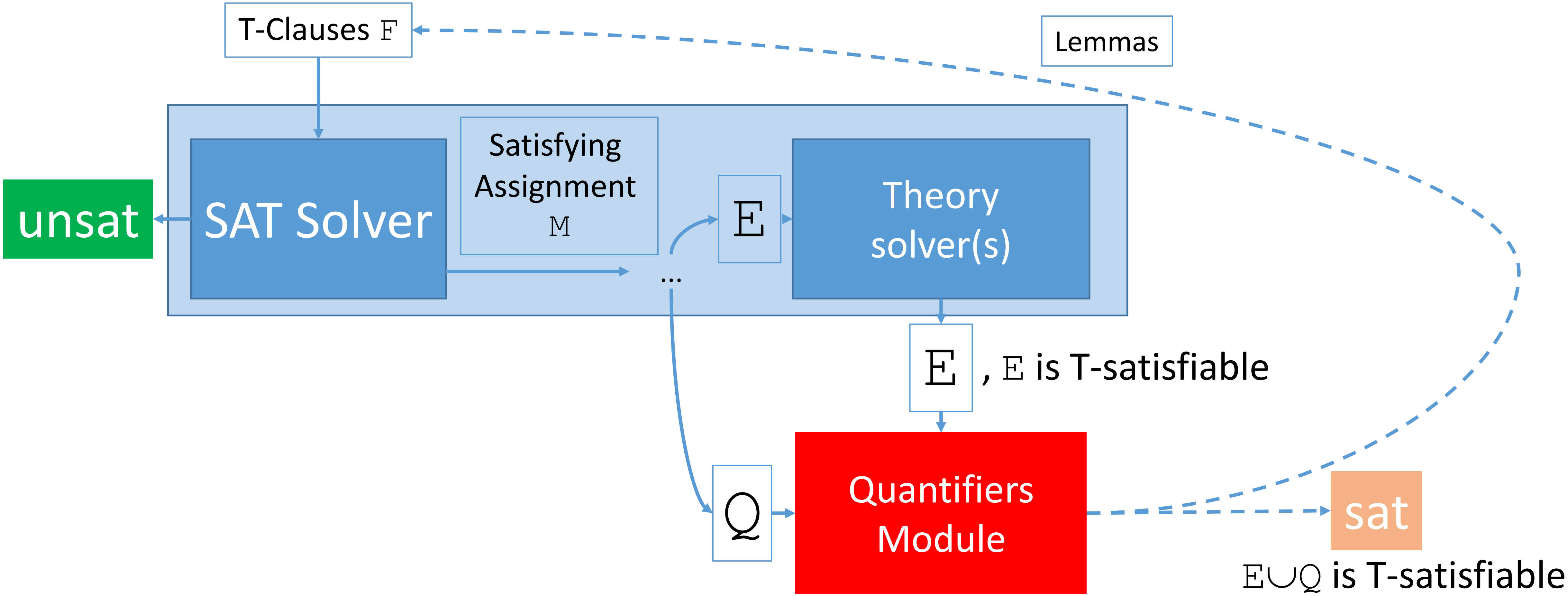
⇒ Theory solvers determine whether E is T-(un)satisfiable

In this talk: DPLL(T)+Quantifiers, simplified



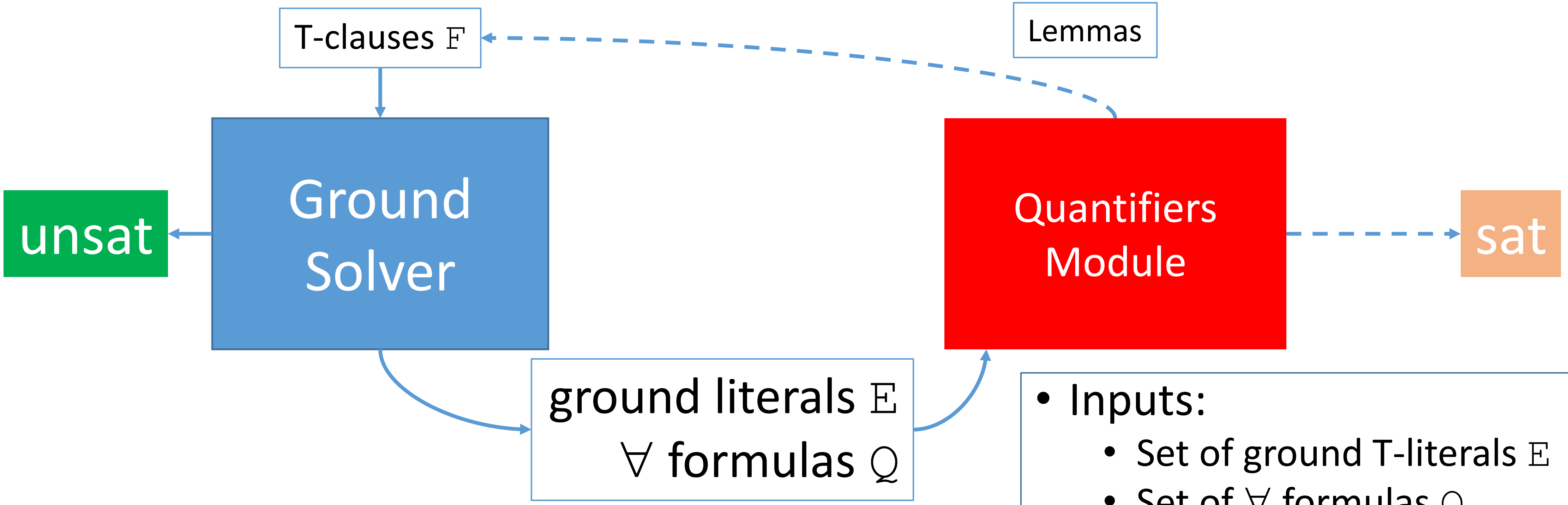
⇒ If E is T-satisfiable, quantifiers module may be invoked

In this talk: DPLL(T)+Quantifiers, simplified



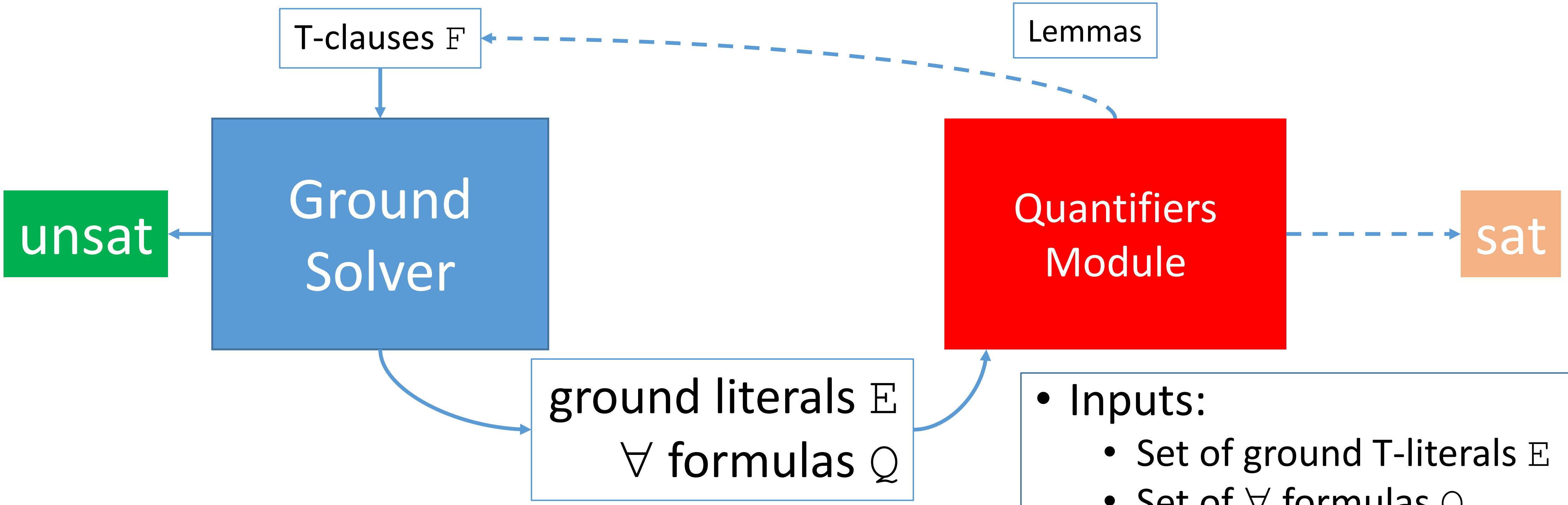
⇒ *The remainder of the talk will discuss how the quantifiers module is implemented*

DPLL(T)+Quantifiers, further simplified



- Inputs:
  - Set of ground T-literals E
  - Set of $\forall$ formulas Q
- Outputs:
  - "$E \cup Q$ is T-satisfiable", or
    $\Rightarrow$ F is T-satisfiable
  - Set of lemmas to add to F

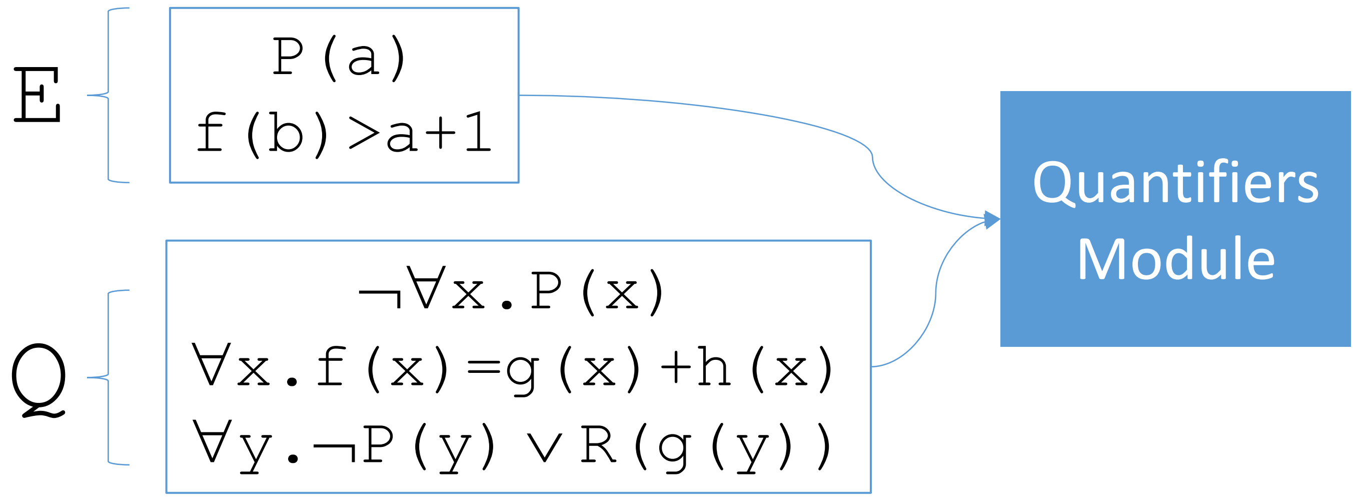
DPLL(T)+Quantifiers, further simplified



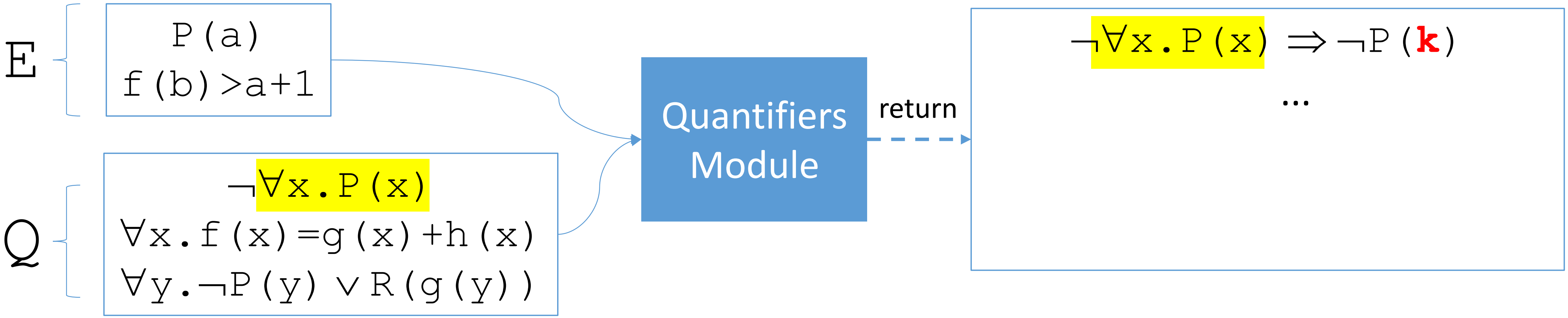
- Recurrent Questions:
  - Which lemmas do we add?
  - How do we know $E \cup Q$ is T-satisfiable?
  - When do we invoke it?

- Inputs:
  - Set of ground T-literals E
  - Set of $\forall$ formulas Q
- Outputs:
  - "$E \cup Q$ is T-satisfiable", or
    $\Rightarrow F$ is T-satisfiable
  - Set of lemmas to add to F

Which lemmas do we add: Basics

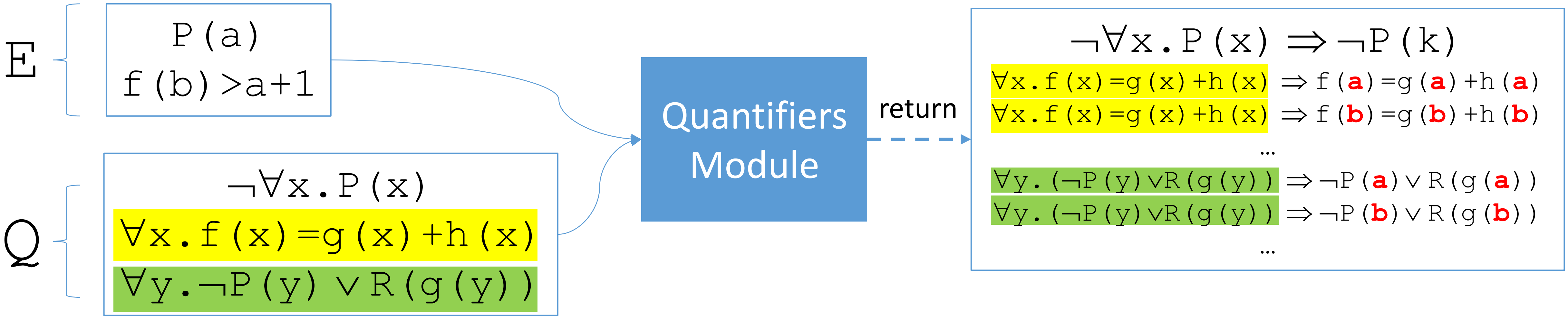


Which lemmas do we add: Basics



- Existential quantification (negated universals) handled by Skolemization
  - Introduce a fresh witness **k**, lemma says ∃x.¬P(x) implies ¬P(**k**)
  - Need only be applied once

Which lemmas do we add: Basics



- Universal quantification handled by Instantiation
  - Choose ground term(s) **t**, lemma(s) say $\forall x.f(x)=g(x)+h(x)$ implies $f(\mathbf{t})=g(\mathbf{t})+h(\mathbf{t})$

$\Rightarrow$ May be applied ad infinitum!

Quantifiers Module : Recurrent Questions

- Which *instances* do we add?
  - E-matching **[Detlefs et al 03]**
  - Conflict-based quantifier instantiation **[Reynolds et al FMCAD14]**
  - Model-based quantifier instantiation **[Ge,de Moura CAV09]**
  - Counterexample-guided quantifier instantiation **[Reynolds et al CAV15]**
  - …

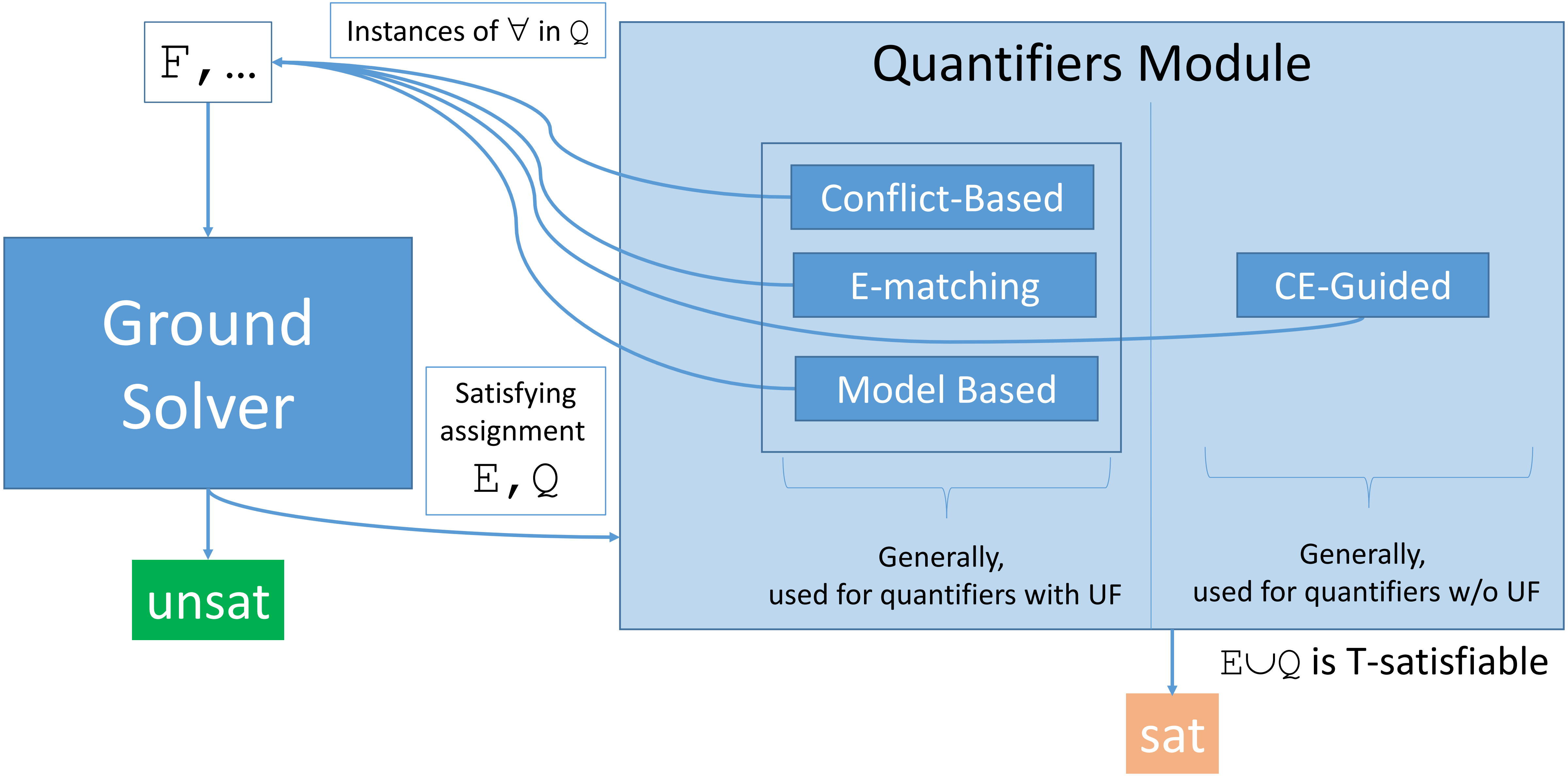
Quantifiers Module : Recurrent Questions

- Which *instances* do we add?
  - E-matching **[Detlefs et al 03]**
  - Conflict-based quantifier instantiation **[Reynolds et al FMCAD14]**
  - Model-based quantifier instantiation **[Ge,de Moura CAV09]**
  - Counterexample-guided quantifier instantiation **[Reynolds et al CAV15]**
  - …
- How do we know $E \cup Q$ *is satisfiable*?
  - For some strategies and fragments, saturation $\Rightarrow$ $E \cup Q$ is satisfiable
    - E.g. model-based, counterexample-guided

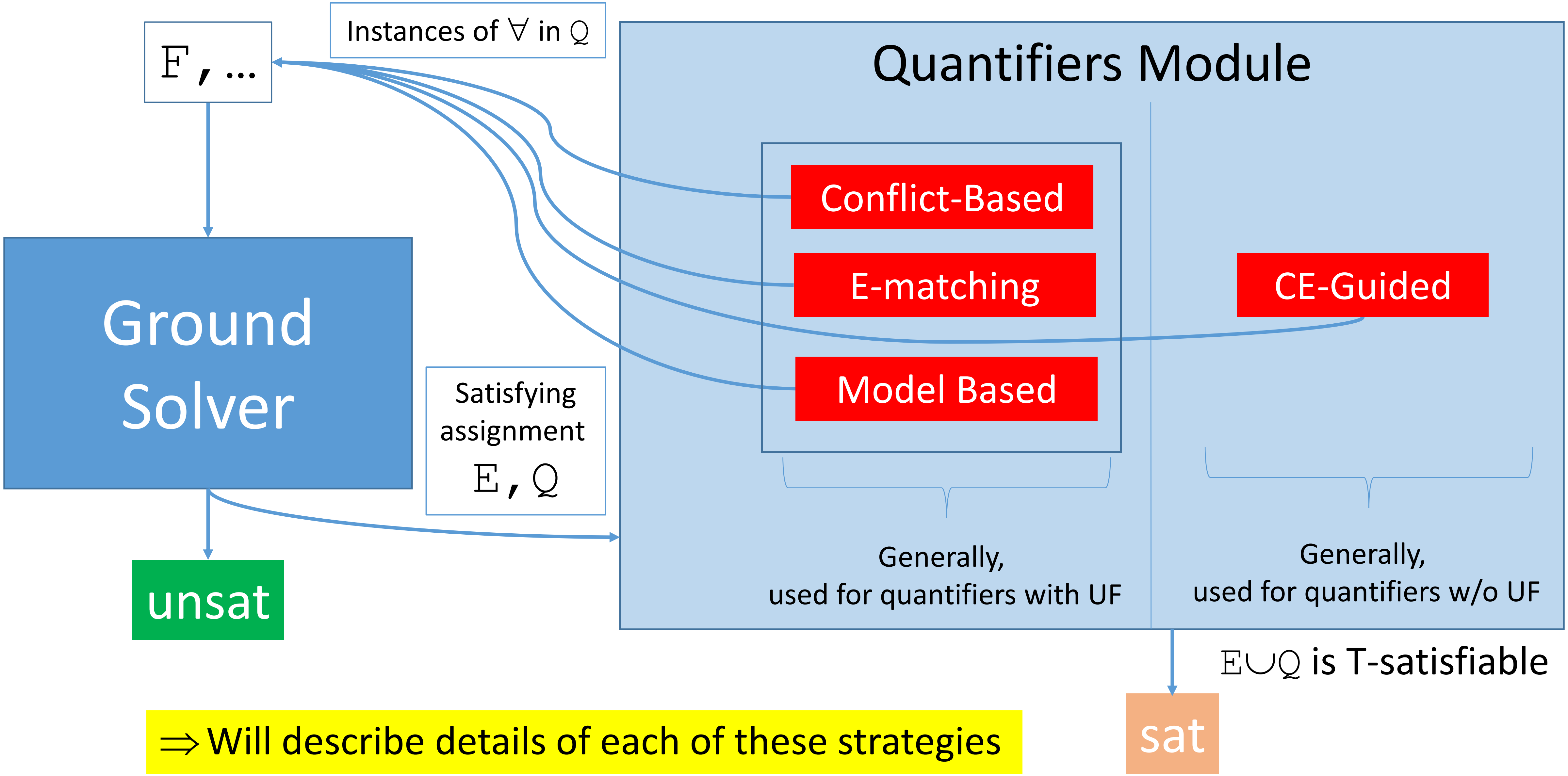
Quantifiers Module : Recurrent Questions

- Which *instances* do we add?
  - E-matching **[Detlefs et al 03]**
  - Conflict-based quantifier instantiation **[Reynolds et al FMCAD14]**
  - Model-based quantifier instantiation **[Ge,de Moura CAV09]**
  - Counterexample-guided quantifier instantiation **[Reynolds et al CAV15]**
  - …
- How do we know $\mathtt{E} \cup \mathtt{Q}$ is satisfiable?
  - For some strategies and fragments, saturation $\Rightarrow$ $\mathtt{E} \cup \mathtt{Q}$ is satisfiable
    - E.g. model-based, counterexample-guided
- *When* do we invoke the quantifiers module?
  - Eagerly, during the DPLL(T) search **[Detlefs et al 03, deMoura/Bjorner CAV07]**, or
  - Lazily, only if $\mathtt{E} \cup \mathtt{Q}$ is a *complete* satisfying assignment

Techniques for Quantifier Instantiation: Overview



Techniques for Quantifier Instantiation: Overview



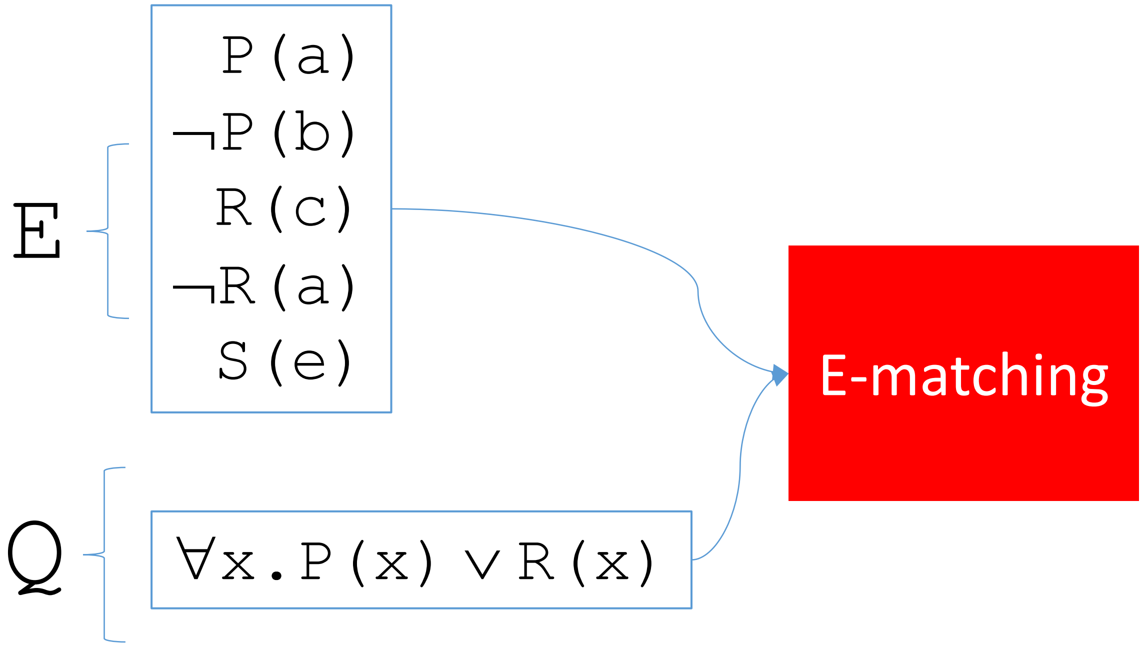
⇒ Will describe details of each of these strategies

E-matching

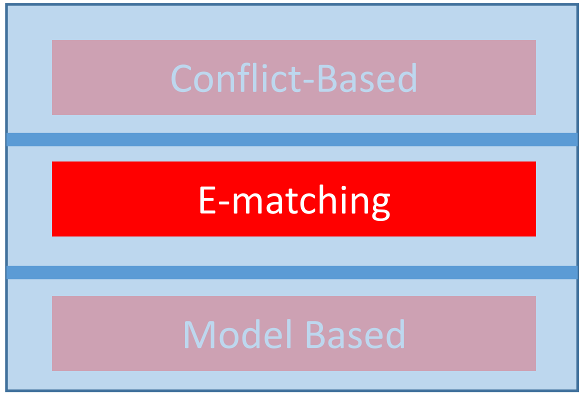
- Introduced in Nelson's Phd Thesis **[Nelson 80]**
  - Implemented in early SMT solvers, e.g. Simplify **[Detlefs et al 03]**
- Most widely used and successful technique for quantifiers in SMT
  - Software verification
    - Boogie/Dafny, Leon, SPARK, Why3
  - Automated Theorem Proving
    - Sledgehammer
- Variants implemented in numerous solvers:
  - Z3 **[deMoura et al 07]**, CVC3 **[Ge et al 07]**, CVC4, Princess **[Ruemmer 12]**, VeriT, Alt-Ergo

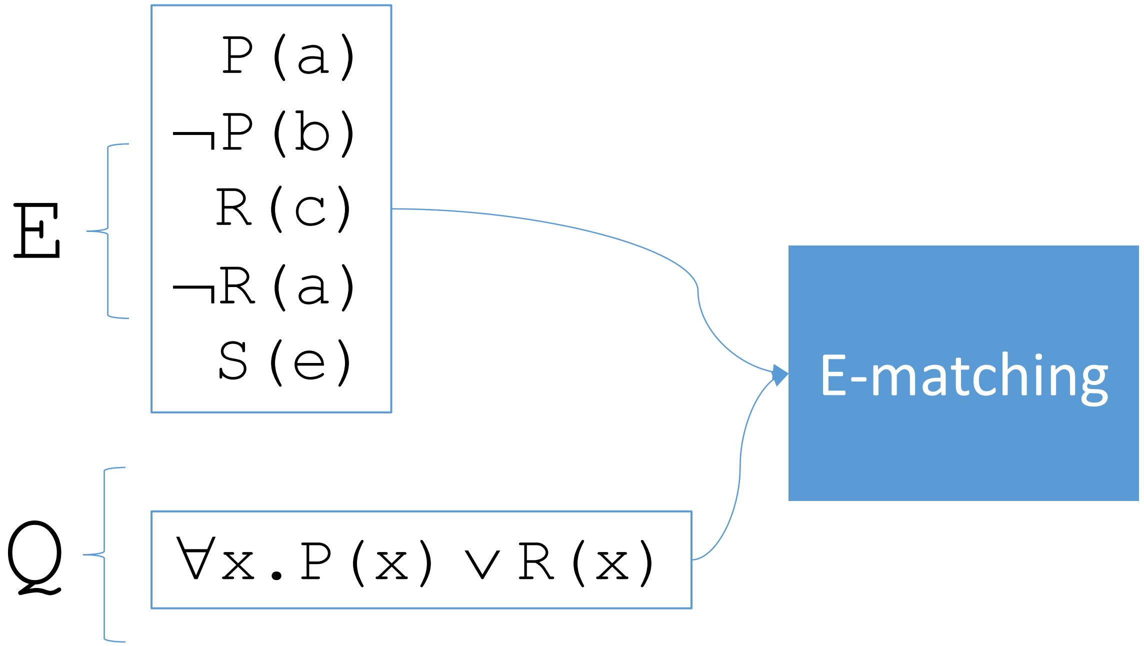
Conflict-Based
E-matching
Model Based

E-matching

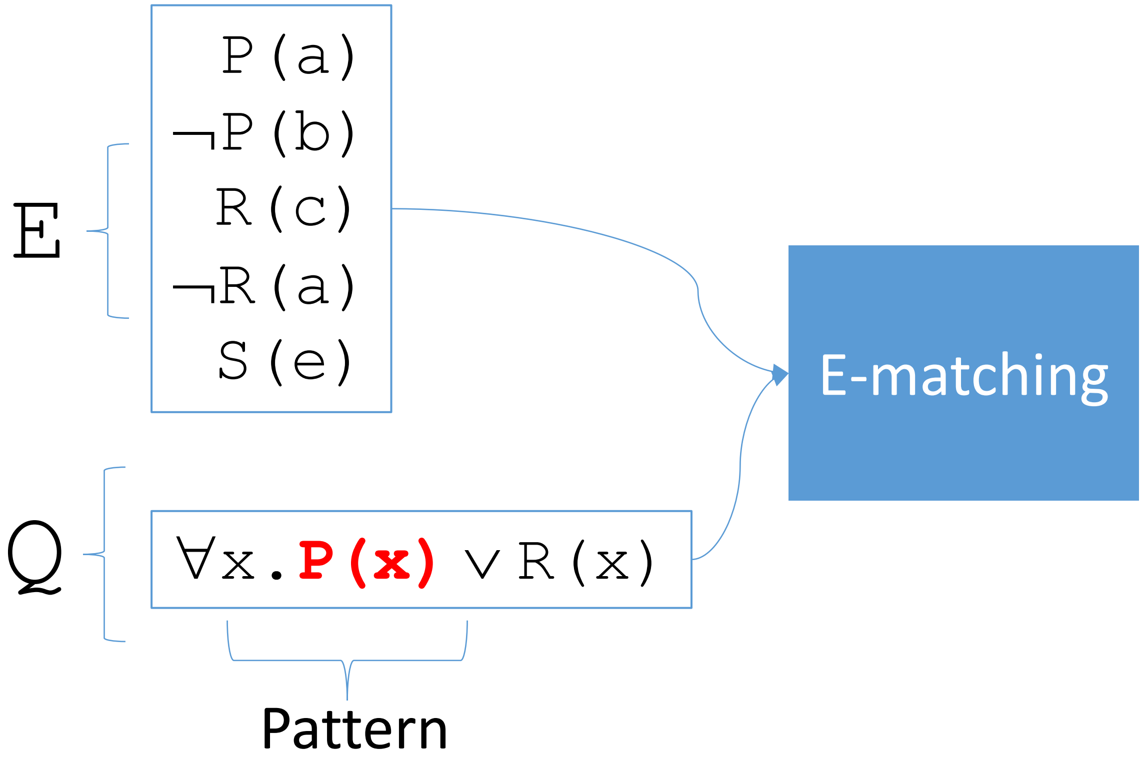


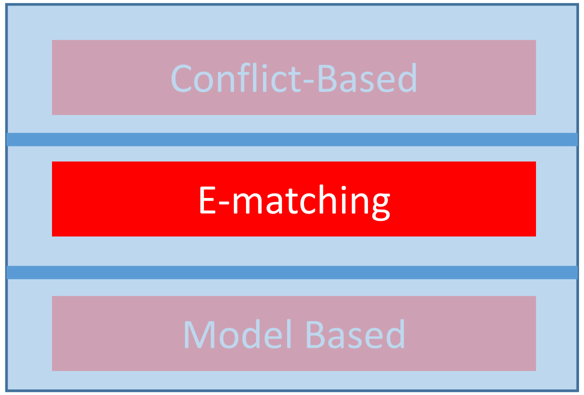
E-matching


Conflict-Based
E-matching
Model Based


E
P(a)
¬P(b)
R(c)
¬R(a)
S(e)
Q
∀x.P(x) ∨ R(x)
E-matching

E-matching

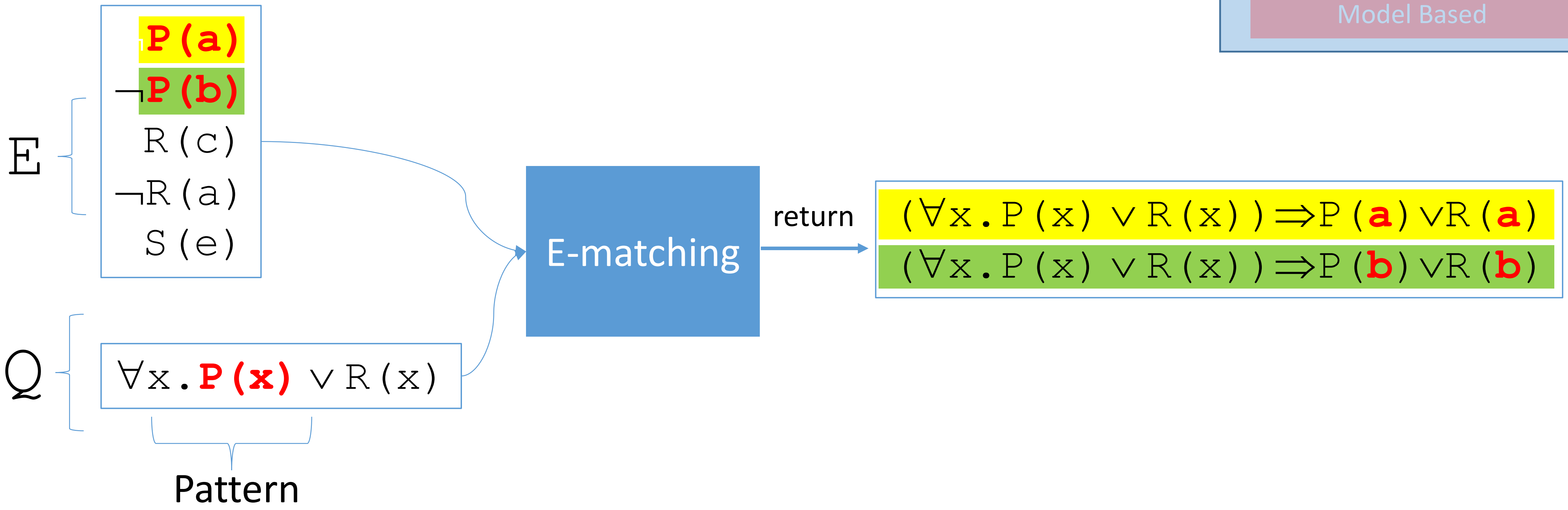




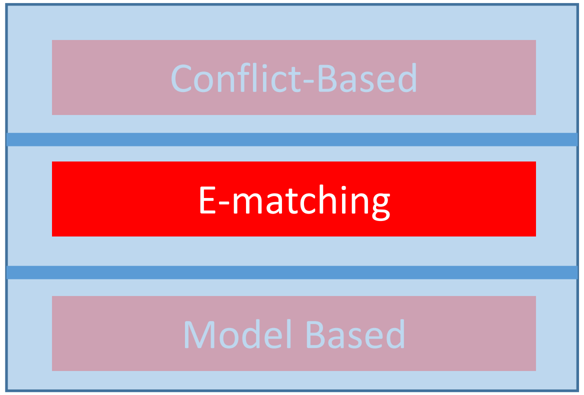
⇒ **Idea**: choose instances based on pattern matching

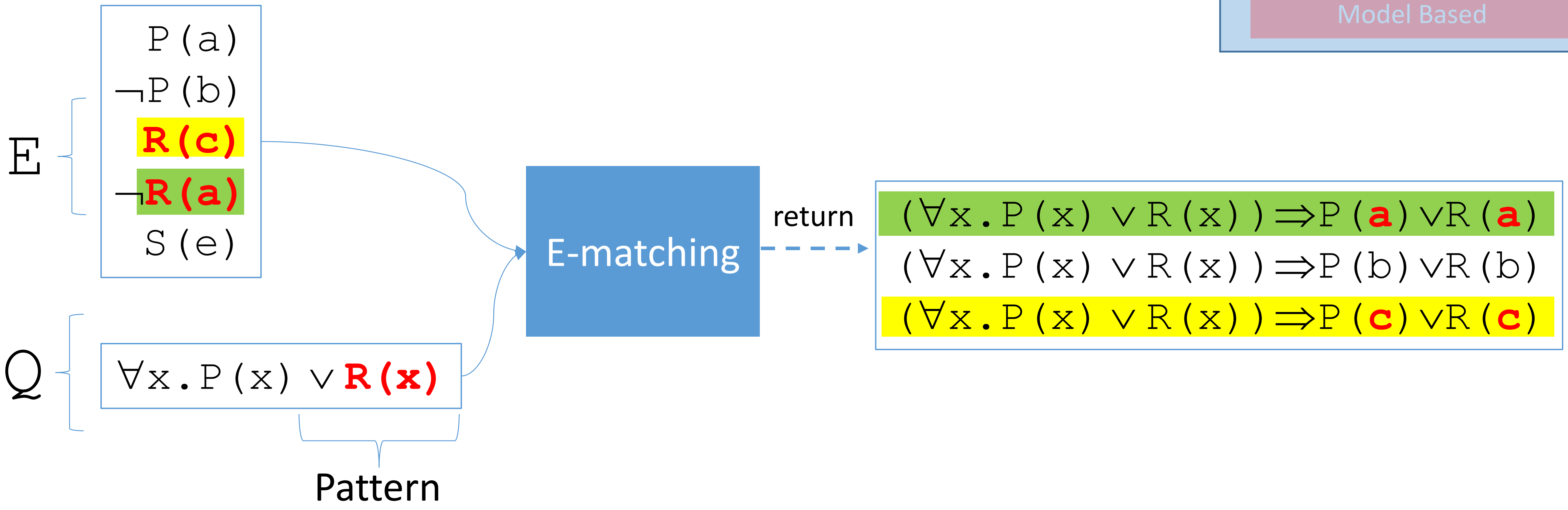
E-matching

Conflict-Based
E-matching
Model Based

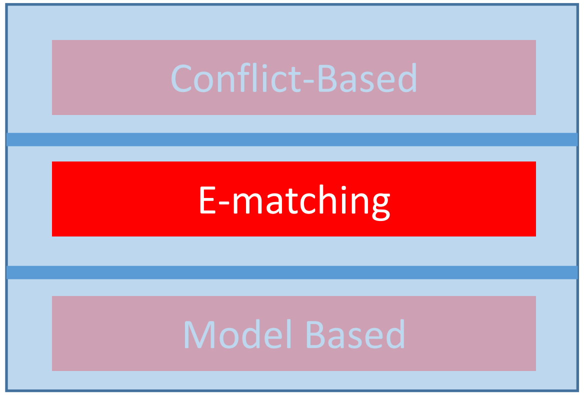


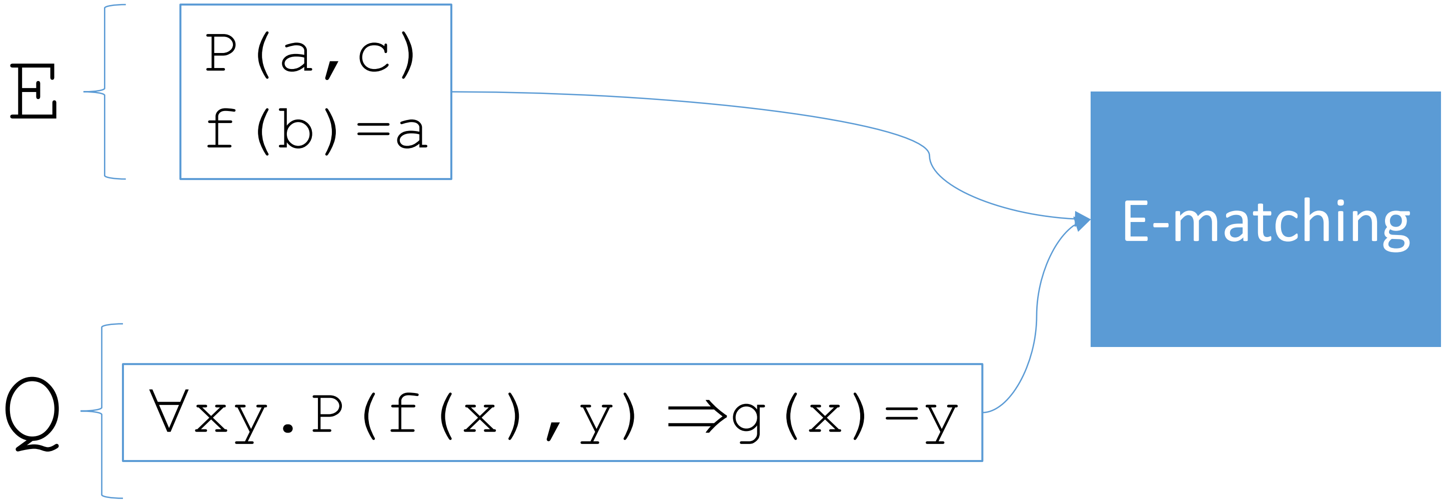
E-matching


Conflict-Based
E-matching
Model Based

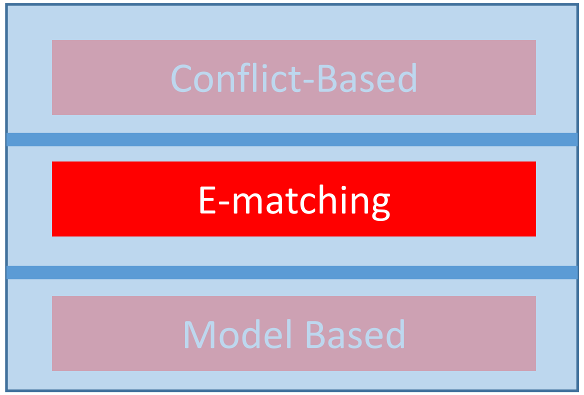

E
P(a)
¬P(b)
R(c)
¬R(a)
S(e)
Q
∀x.P(x) ∨ R(x)
Pattern
E-matching
return
(∀x.P(x) ∨ R(x))⇒P(a)∨R(a)
(∀x.P(x) ∨ R(x))⇒P(b)∨R(b)
(∀x.P(x) ∨ R(x))⇒P(c)∨R(c)

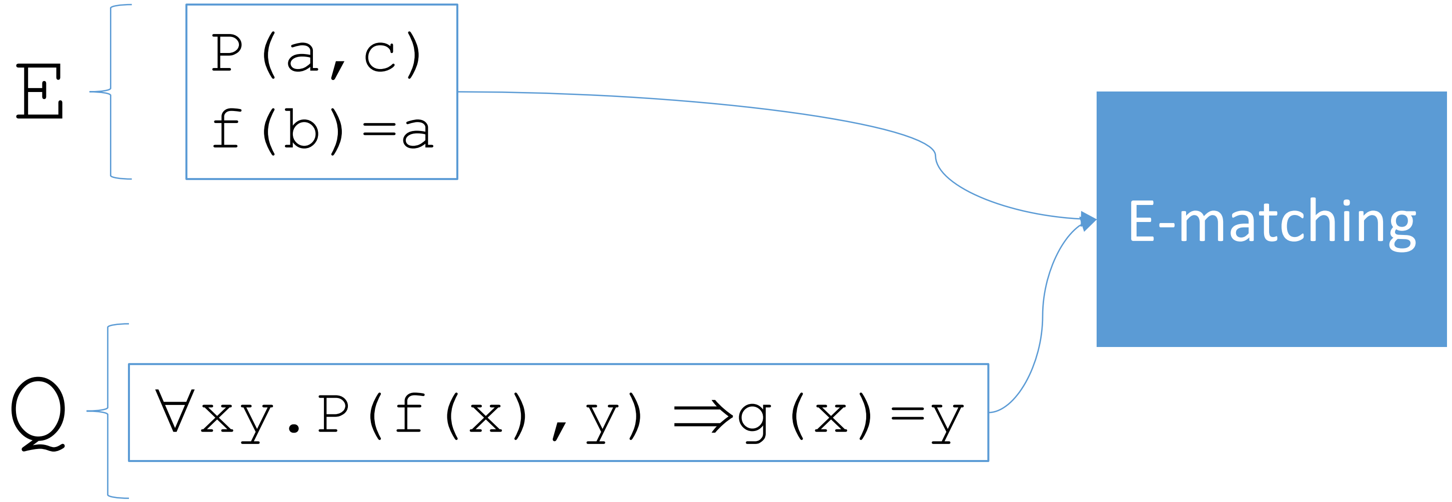
E-matching: Functions, Equality





E-matching: Functions, Equality





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

E-matching: Functions, Equality

- Conflict-Based
- **E-matching**
- Model Based

E:

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

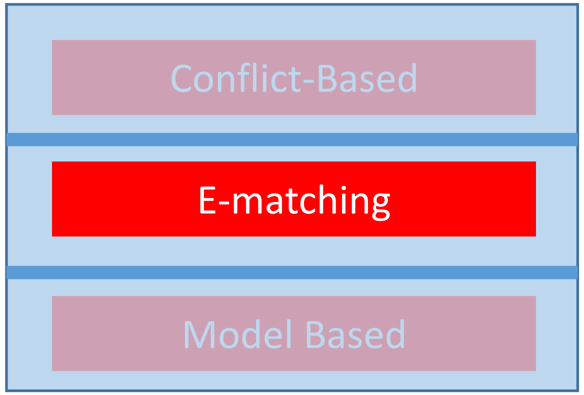
Q:

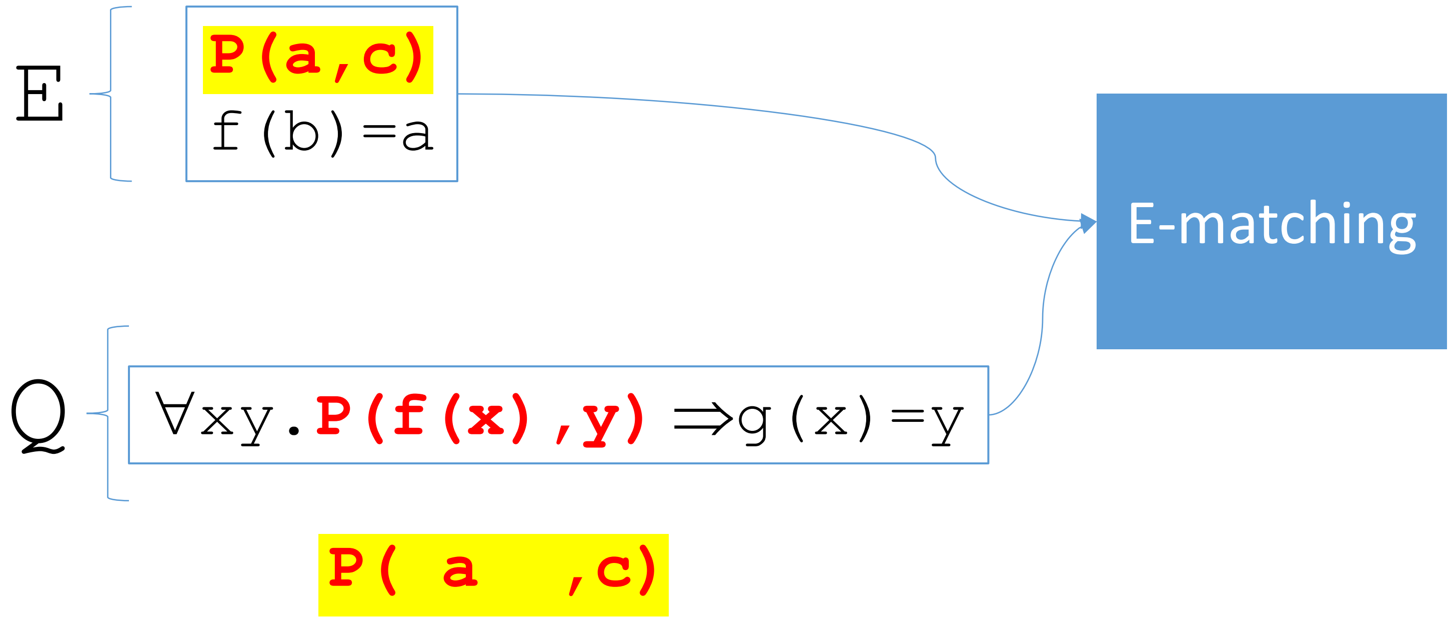
$\forall xy. \mathbf{P(f(x),y)} \Rightarrow g(x)=y$

Pattern: $P(f(x),y)$

→ E-matching

E-matching: Functions, Equality



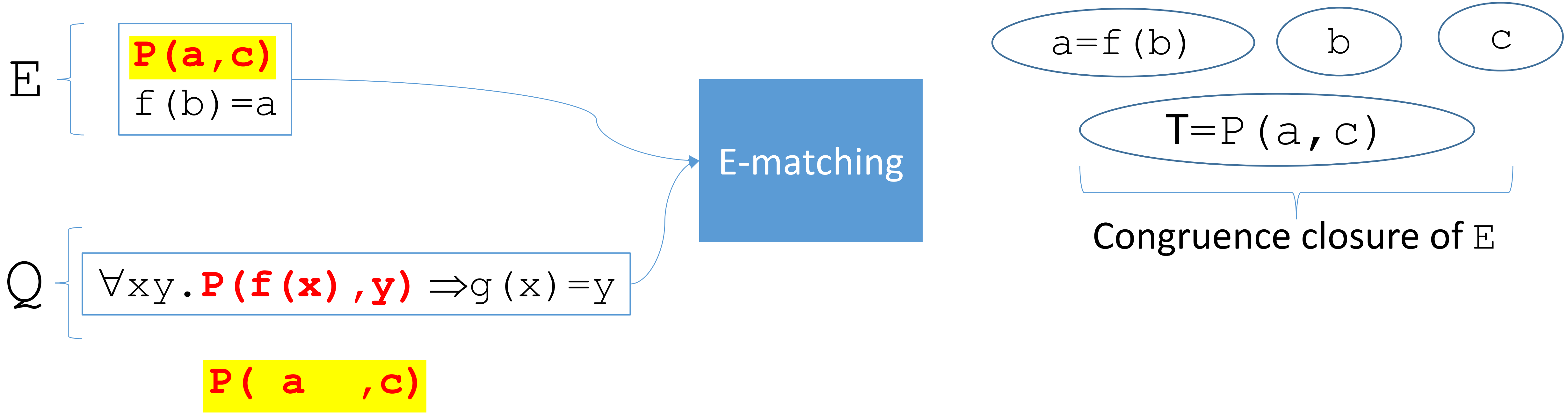


E-matching: Functions, Equality

Conflict-Based

E-matching

Model Based



E-matching: Functions, Equality

Conflict-Based
E-matching
Model Based

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

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

E-matching

a=f(b) | b | c

T=P(a,c)

P(f(b),c)

…E implies P(a,c) $\Leftrightarrow$ P(f(b),c)

E-matching: Functions, Equality

Conflict-Based

E-matching

Model Based

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

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

P(f(b),c)

E-matching

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

E-matching

Conflict-Based

E-matching

Model Based

Given:

- Set of ground T-literals E
- Quantified formula $\forall \mathbf{x}.\Psi$, where $\mathbf{x}$ is a tuple of variables
- A pattern p contain all variables in $\mathbf{x}$
- A ground term g from E

- Formally:
  - We say ***g matches p modulo E*** under the substitution $\{\mathbf{x} \rightarrow \mathbf{t}\}$ if $\mathrm{E} \models_T \mathrm{g}=\mathrm{p}\{\mathbf{x} \rightarrow \mathbf{t}\}$

E-matching

Conflict-Based

E-matching

Model Based

Given:

- Set of ground T-literals E
- Quantified formula $\forall \mathbf{x} . \Psi$, where $\mathbf{x}$ is a tuple of variables
- A pattern p contain all variables in $\mathbf{x}$
- A ground term g from E

- Formally:
  - We say g matches p modulo E under the substitution $\{\mathbf{x} \rightarrow \mathbf{t}\}$ if $\mathrm{E} \models_{T} \mathrm{g} = \mathrm{p}\{\mathbf{x} \rightarrow \mathbf{t}\}$

usually restricted such that
T is theory of equality

Conflict-Based

E-matching

Model Based

E-matching

Given:

- Set of ground T-literals E
- Quantified formula $\forall \mathbf{x}.\Psi$, where $\mathbf{x}$ is a tuple of variables
- A pattern p contain all variables in $\mathbf{x}$
- A ground term g from E

- Formally:
  - We say g matches p modulo E under the substitution $\{\mathbf{x}\to\mathbf{t}\}$ if $\mathrm{E} \models_T \mathrm{g}=\mathrm{p}\{\mathbf{x}\to\mathbf{t}\}$
  - **E-matching**:
    1. Chooses (a set) of patterns $\mathrm{p}_1, \ldots, \mathrm{p}_m$ for $\forall \mathbf{x}.\Psi$
    2. Computes sets of pairs $(\{\mathbf{x}\to\mathbf{t}_{j1}\}, \mathbf{g}_{j1}), \ldots, (\{\mathbf{x}\to\mathbf{t}_{jn}\}, \mathbf{g}_{jn})$ where g_{ji} matches p_j modulo E
    3. Returns the instances $(\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x}\to\mathbf{t}_{11}\}), \ldots, (\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x}\to\mathbf{t}_{nm}\})$

E-matching: Intuition

- Say E-matching returns the instance ($\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x} \rightarrow \mathbf{t}\}$)

  $\Rightarrow$ ***Why is this instance useful?***

Conflict-Based

E-matching

Model Based

E-matching: Intuition

- Say E-matching returns the instance $(\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x}\rightarrow\mathbf{t}\})$

    $\Rightarrow$ ***Why is this instance useful?***

- We are interested in **satisfiability of $\mathtt{E}\cup\mathtt{Q}$**

E-matching: Intuition

- Say E-matching returns the instance $(\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x} \rightarrow \mathbf{t}\})$

  $\Rightarrow$ ***Why is this instance useful?***

- We are interested in satisfiability of $E \cup Q$
- Assume pattern p is a subterm of Ψ, e.g. $\forall \mathbf{x}.\Psi[p]$

Conflict-Based

E-matching

Model Based

E-matching: Intuition

- Say E-matching returns the instance $(\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x}\rightarrow\mathbf{t}\})$

  $\Rightarrow$ ***Why is this instance useful?***

- We are interested in satisfiability of $E \cup Q$
- Assume pattern p is a subterm of Ψ, e.g. $\forall \mathbf{x}.\Psi[p]$
- E-matching finds a ground term g from E, where $\mathbf{g}=\mathbf{p}\{\mathbf{x}\rightarrow\mathbf{t}\}$ is implied by E

Conflict-Based
E-matching
Model Based

E-matching: Intuition

- Say E-matching returns the instance $(\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x} \rightarrow \mathbf{t}\})$

  $\Rightarrow$ ***Why is this instance useful?***

- We are interested in satisfiability of $E \cup Q$
- Assume pattern p is a subterm of Ψ, e.g. $\forall \mathbf{x}.\Psi[p]$
- E-matching finds a ground term g from E, where $g{=}p\{\mathbf{x} \rightarrow \mathbf{t}\}$ is implied by E
- **Thus:** $\Psi[g]$ is implied by $E \cup \{\Psi[p]\{\mathbf{x} \rightarrow \mathbf{t}\}\}$

E-matching: Intuition

Conflict-Based

E-matching

Model Based

- Say E-matching returns the instance $(\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x}\rightarrow\mathbf{t}\})$

  $\Rightarrow$ ***Why is this instance useful?***

- We are interested in satisfiability of $E\cup Q$
- Assume pattern p is a subterm of Ψ, e.g. $\forall \mathbf{x}.\Psi[p]$
- E-matching finds a ground term g from E, where $g=p\{\mathbf{x}\rightarrow\mathbf{t}\}$ is implied by E
- **Thus:** $\Psi[g]$ is implied by $E\cup\{\Psi[p]\{x\rightarrow t\}\}$

  $\Rightarrow$ ***In other words, from Q, we learn information $\Psi[g]$ about a term g from E***

Conflict-Based

E-matching

Model Based

E-matching: Intuition

- Say E-matching returns the instance $(\forall \mathbf{x}.\Psi \Rightarrow \Psi\{\mathbf{x} \rightarrow \mathbf{t}\})$

  $\Rightarrow$ ***Why is this instance useful?***

- We are interested in satisfiability of $E \cup Q$
- Assume pattern p is a subterm of Ψ, e.g. $\forall \mathbf{x}.\Psi[p]$
- E-matching finds a ground term g from E, where $g = p\{\mathbf{x} \rightarrow \mathbf{t}\}$ is implied by E
- **Thus:** $\Psi[g]$ is implied by $E \cup \{\Psi[p]\ \{\mathbf{x} \rightarrow \mathbf{t}\}\}$

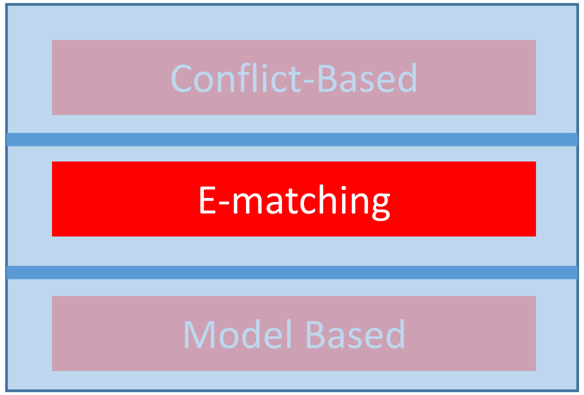
  $\Rightarrow$ ***In other words, from Q, we learn information $\Psi[g]$ about a term g from E***

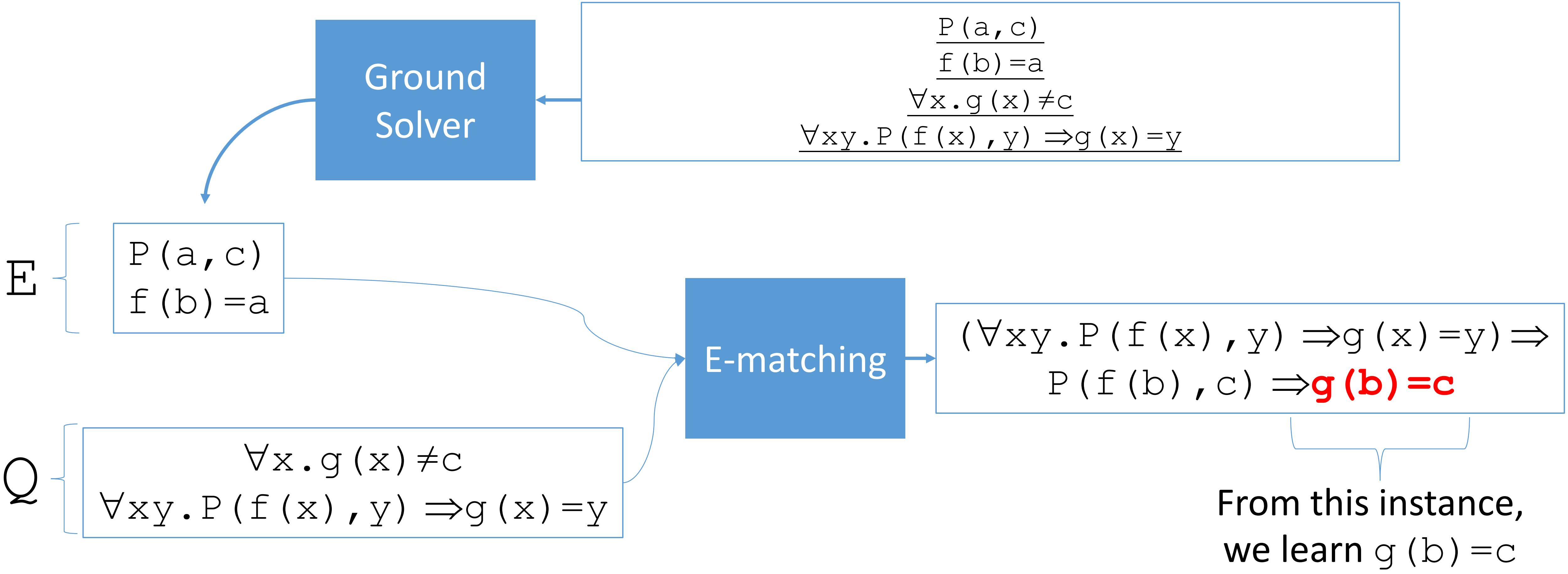
$\mathbf{P(a,c)} \Rightarrow g(b)=c$ is implied by

$\{\mathbf{P(a,c)}, f(b)=a\} \cup \{\mathbf{P(f(b),c)} \Rightarrow g(b)=c\}$

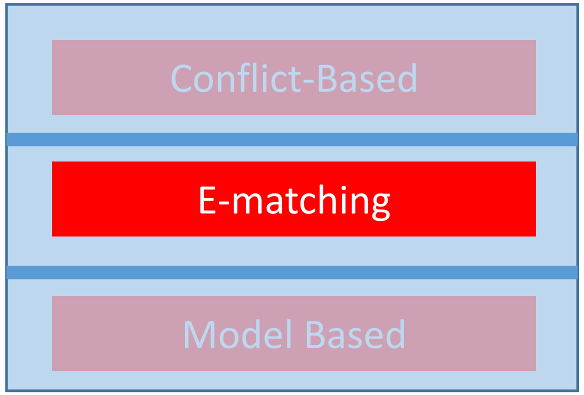
E with new instance

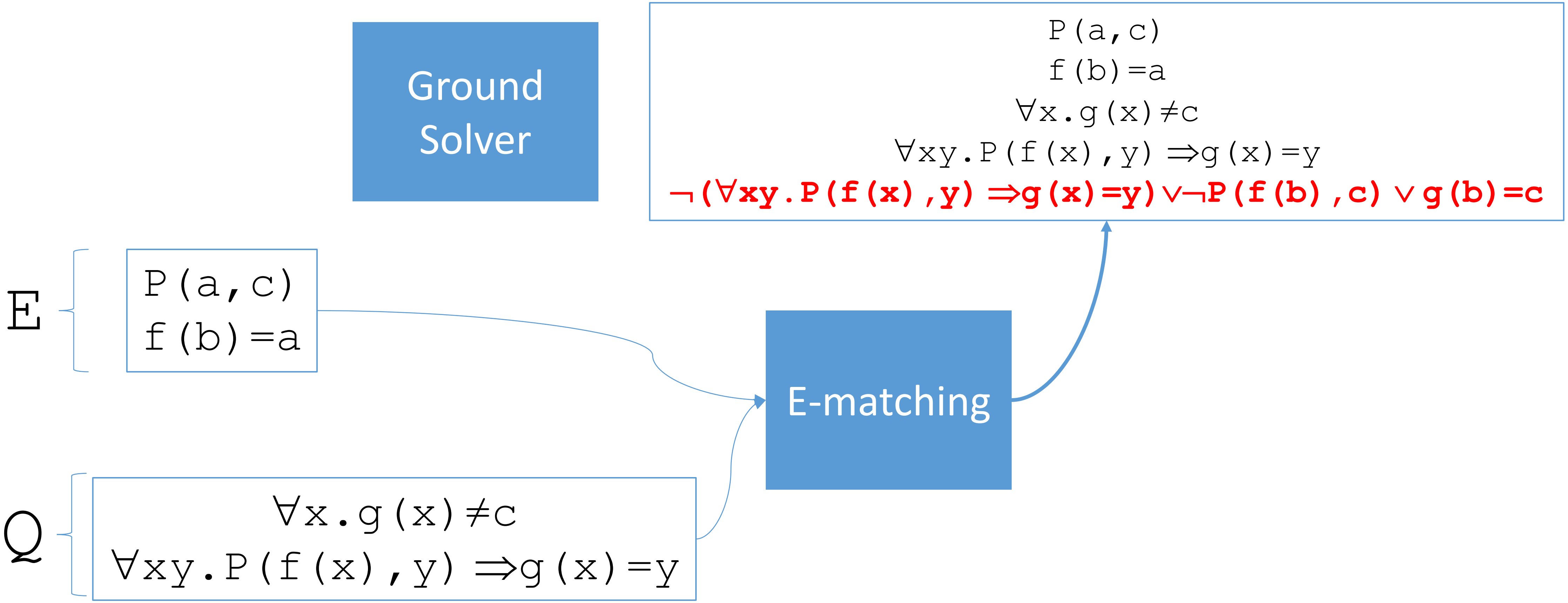
E-matching


Conflict-Based
E-matching
Model Based


Ground Solver
P(a,c)
f(b)=a
∀x.g(x)≠c
∀xy.P(f(x),y)⇒g(x)=y
E
P(a,c)
f(b)=a
Q
∀x.g(x)≠c
∀xy.P(f(x),y)⇒g(x)=y
E-matching
(∀xy.P(f(x),y)⇒g(x)=y)⇒
P(f(b),c)⇒g(b)=c
From this instance,
we learn g(b)=c

E-matching


Conflict-Based
E-matching
Model Based


Ground Solver
P(a,c)
f(b)=a
∀x.g(x)≠c
∀xy.P(f(x),y) ⇒g(x)=y
¬(∀xy.P(f(x),y) ⇒g(x)=y)∨¬P(f(b),c) ∨g(b)=c
E
P(a,c)
f(b)=a
E-matching
Q
∀x.g(x)≠c
∀xy.P(f(x),y) ⇒g(x)=y

E-matching

Conflict-Based

E-matching

Model Based

Ground Solver

$P(a,c)$

$f(b)=a$

$\forall x. g(x) \neq c$

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

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

E′:

$P(a,c)$

$f(b)=a$

$\mathbf{g(b)=c}$

Q′:

$\forall x. g(x) \neq c$

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

E-matching

E-matching

Conflict-Based

E-matching

Model Based

Ground Solver

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

E′:
$P(a,c)$
$f(b)=a$
$\mathbf{g(b)}=c$

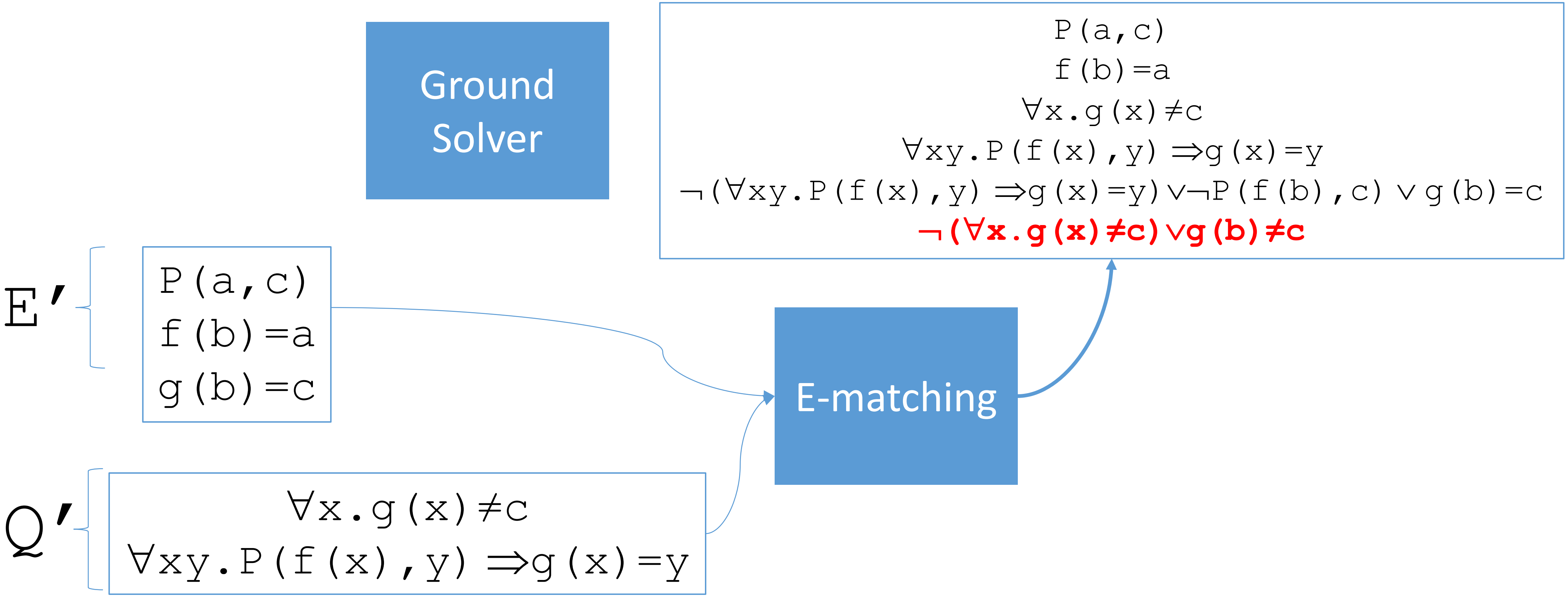
Q′:
$\forall x.\, \mathbf{g(x)} \neq c$
$\forall xy.\, P(f(x),y) \Rightarrow g(x)=y$

E-matching → $(\forall x.\, g(x) \neq c) \Rightarrow g(\mathbf{b}) \neq c$

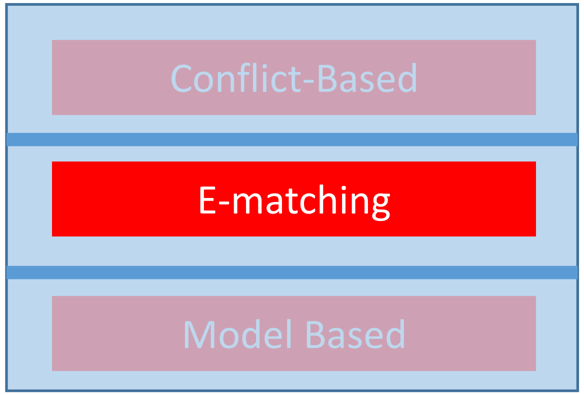
$\Rightarrow$ New terms lead to new instances

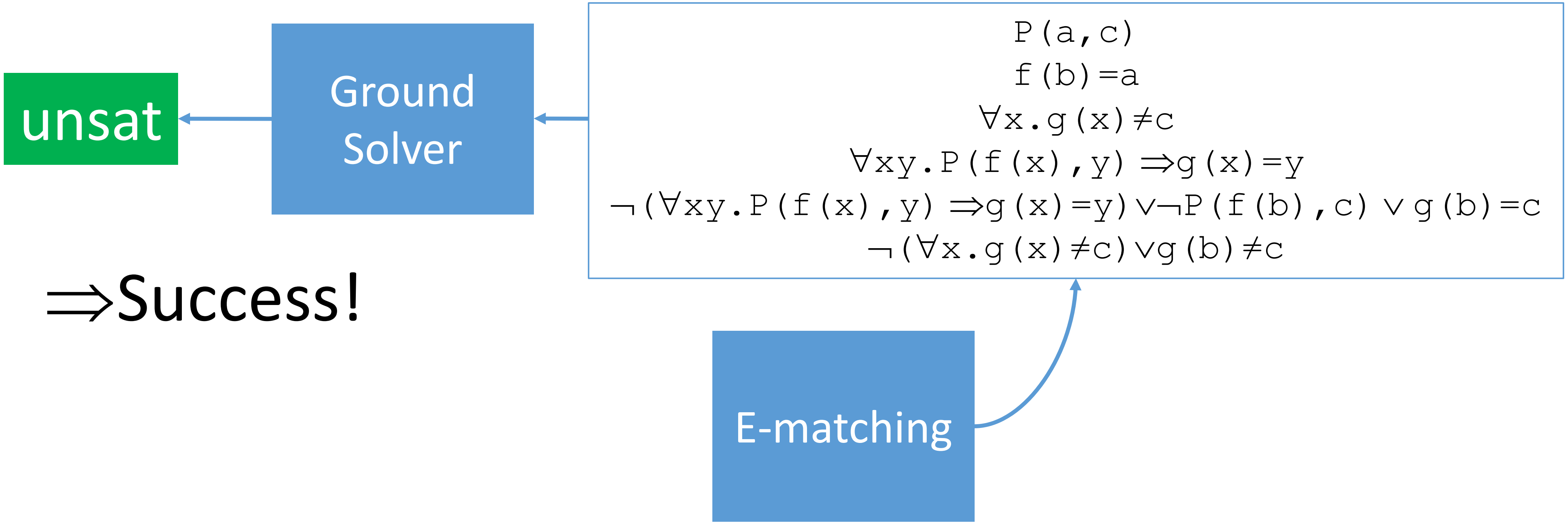
E-matching

Conflict-Based
E-matching
Model Based



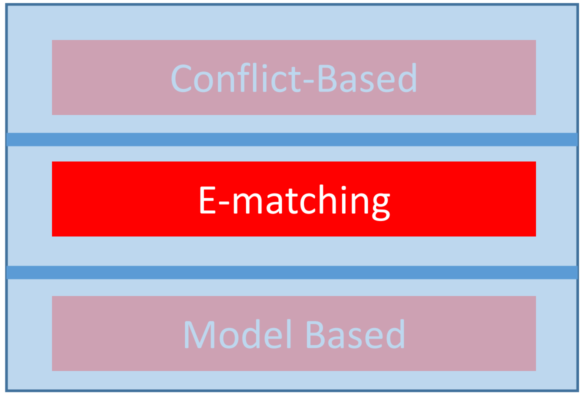
E-matching





⇒Success!

E-matching: Challenges



- E-matching has no standard way of selecting patterns
- E-matching generates too many instances
  - Many instances may overload the ground solver
- 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 $\Leftrightarrow$ model)
      - Decision Procedures via Triggers **[Dross et al 13]**
      - Local Theory Extensions **[Bansal et al 15]**

      $\Rightarrow$ Typically are established by a separate pencil-and-paper proof

Conflict-Based

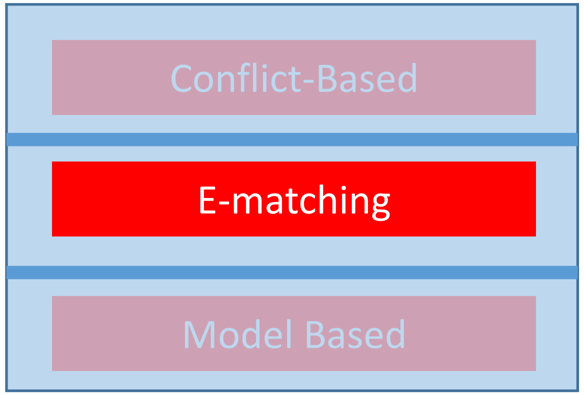
E-matching

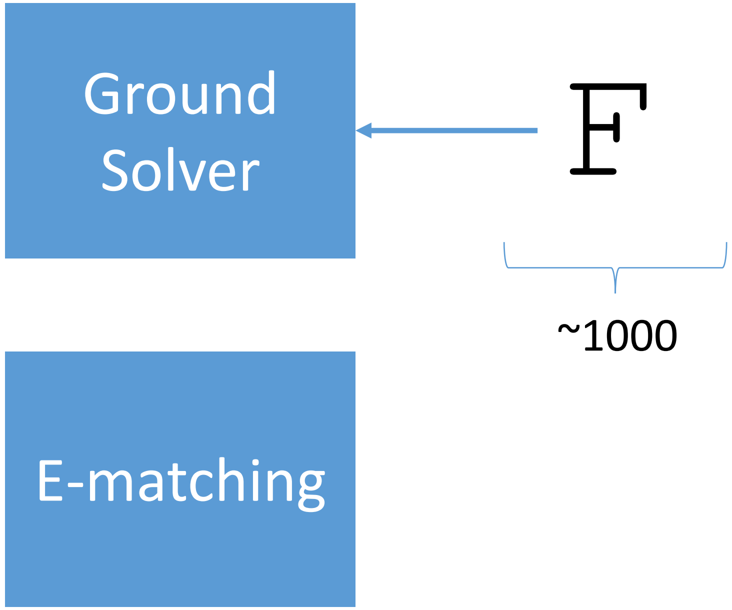
Model Based

E-matching: Pattern Selection

- In practice, pattern selection can is done either by:
  - The user, via annotations, e.g. `(! … :pattern ((P x)))`
  - The SMT solver itself
- 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]**

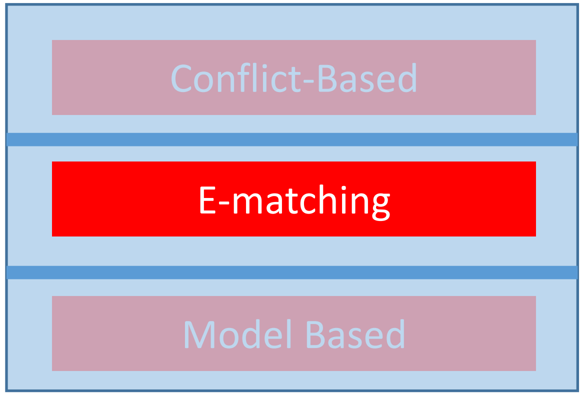
E-matching: Too Many Instances

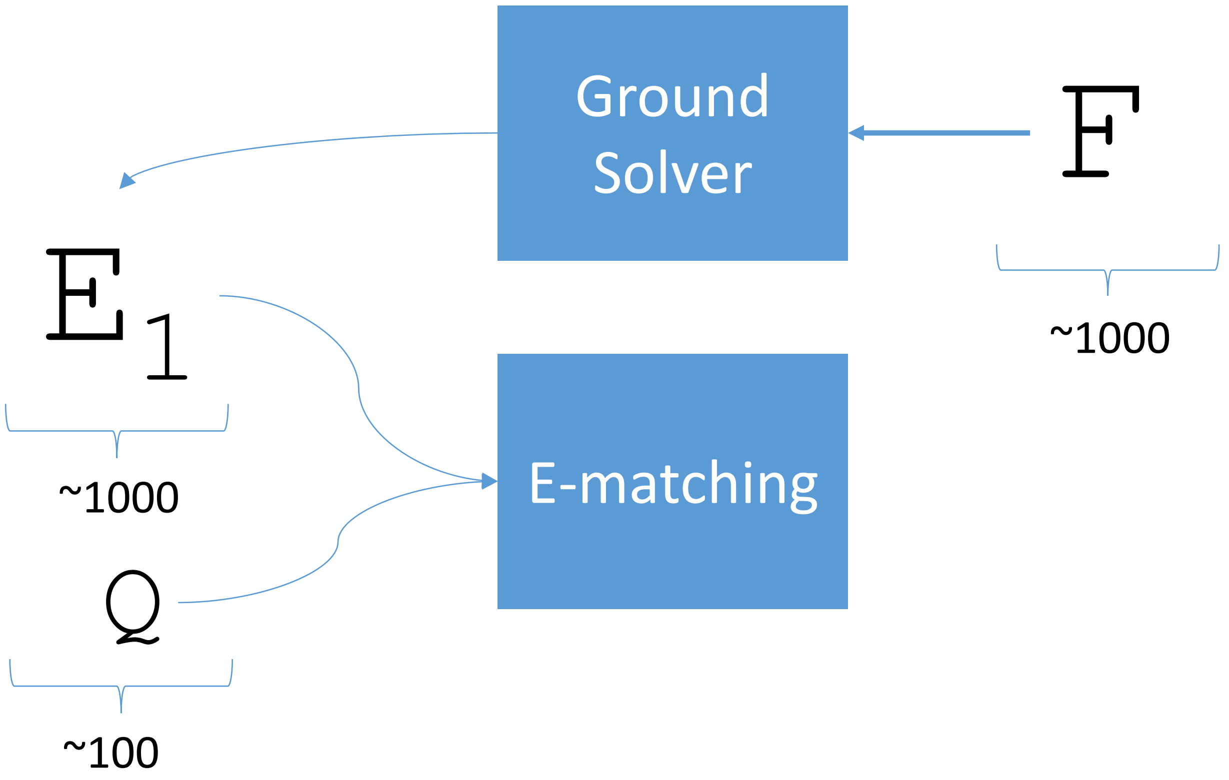




- Typical problems in applications:
  - F contains 1000s of clauses

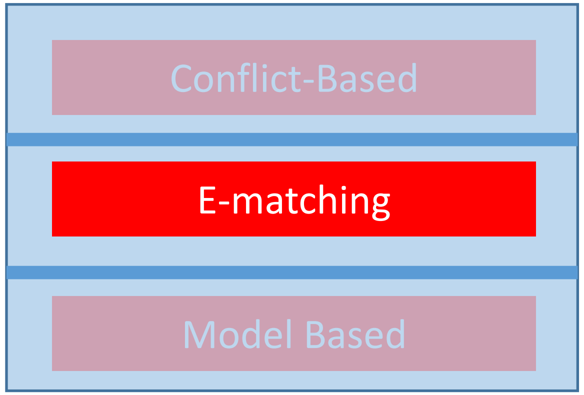
E-matching: Too Many Instances

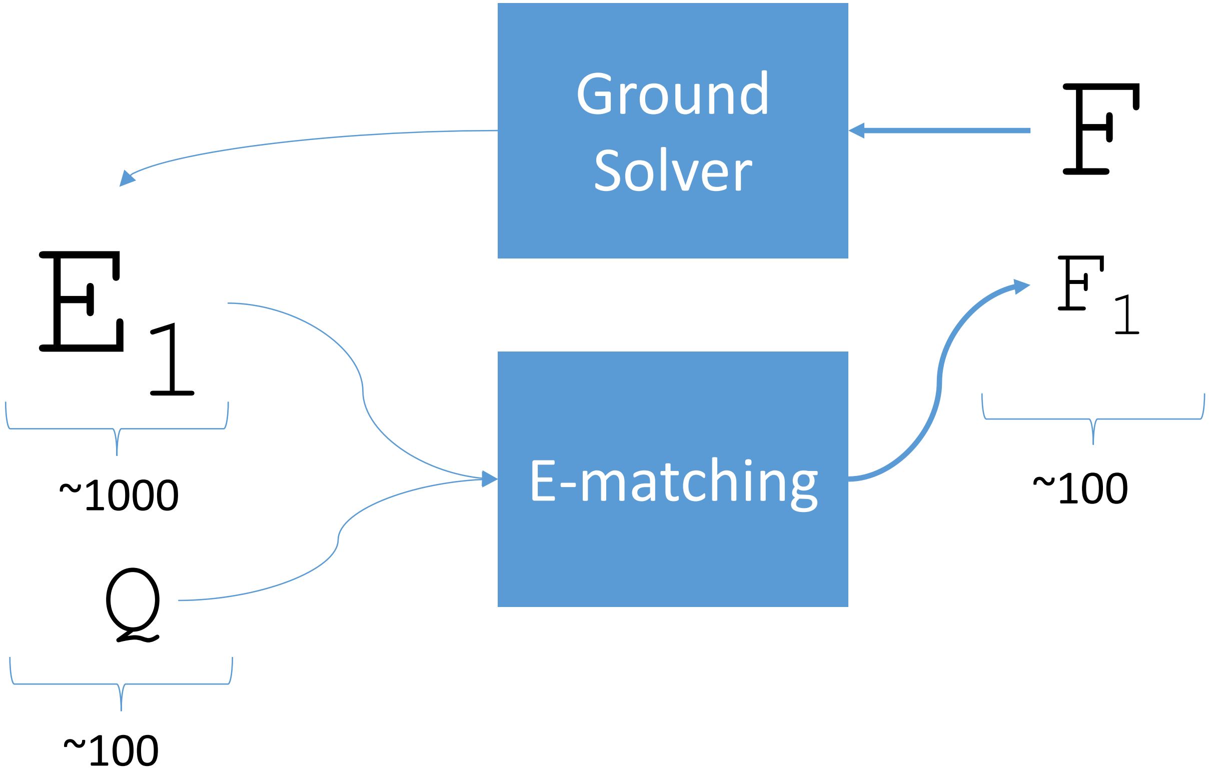




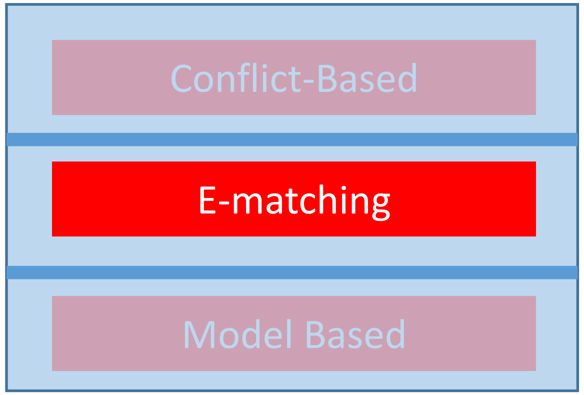
- Typical problems in applications:
  - F contains 1000s of clauses
  - Satisfying assignments contain 1000s of terms in E, 100s of $\forall$ in Q

E-matching: Too Many Instances

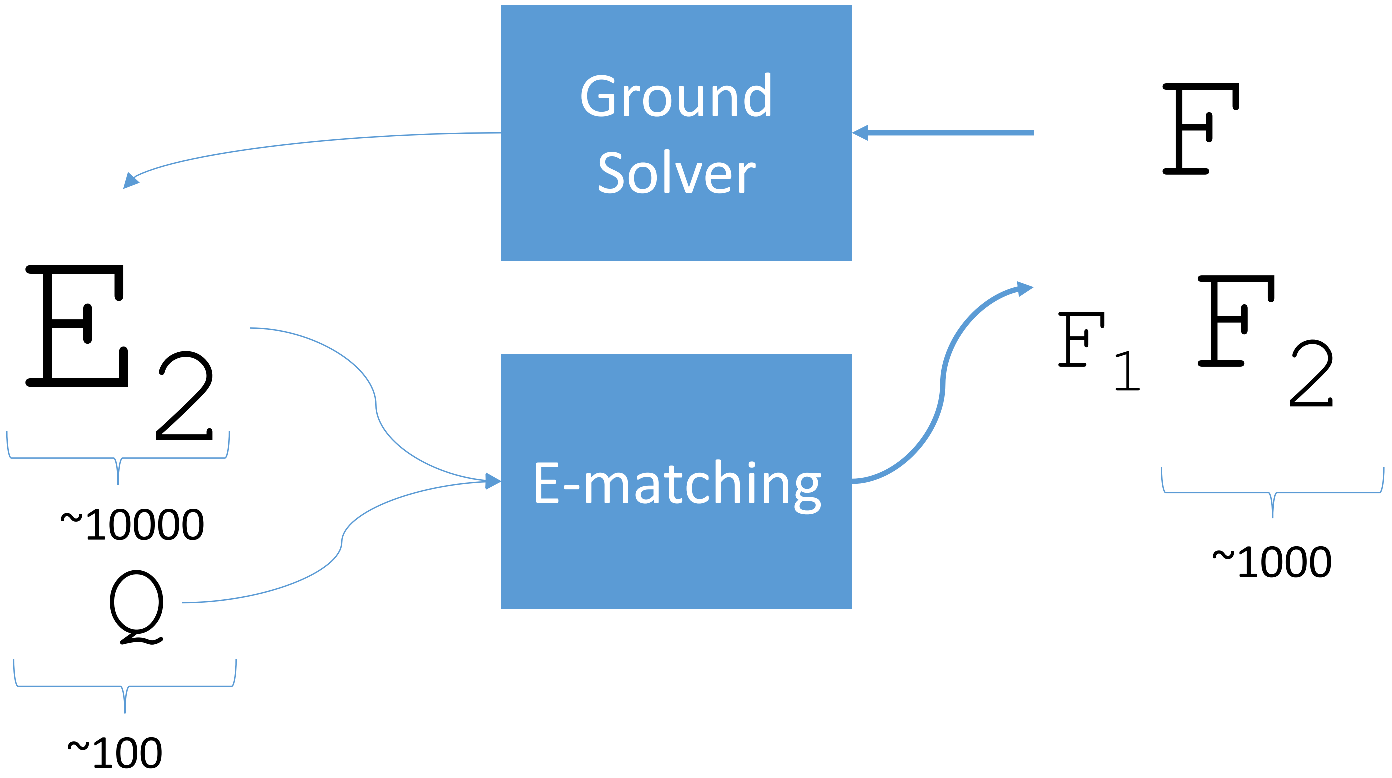




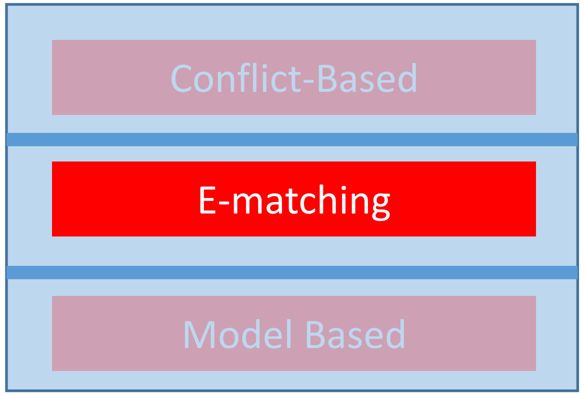
- Typical problems in applications:
  - F contains 1000s of clauses
  - Satisfying assignments contain 1000s of terms in E, 100s of ∀ in Q
  - Leads to 100s



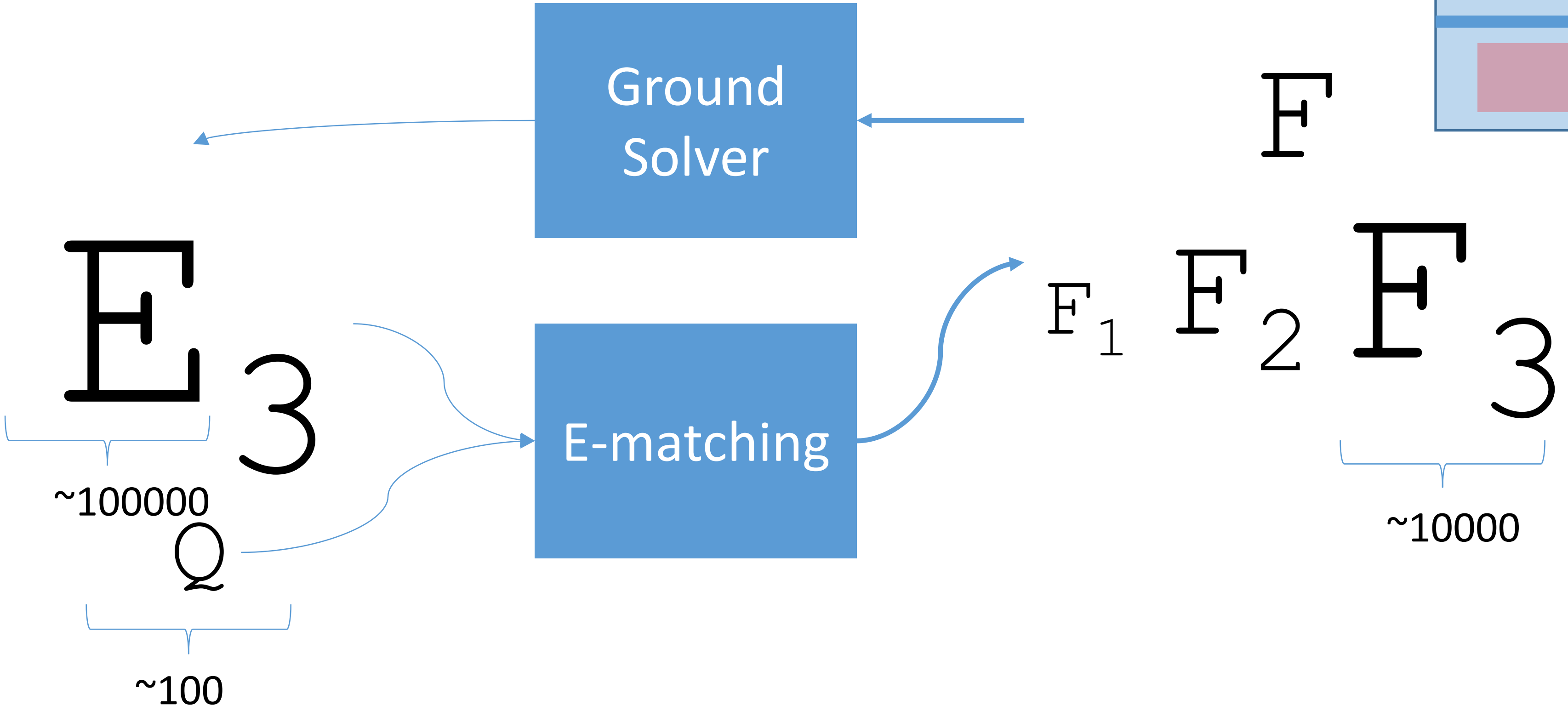
E-matching: Too Many Instances



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

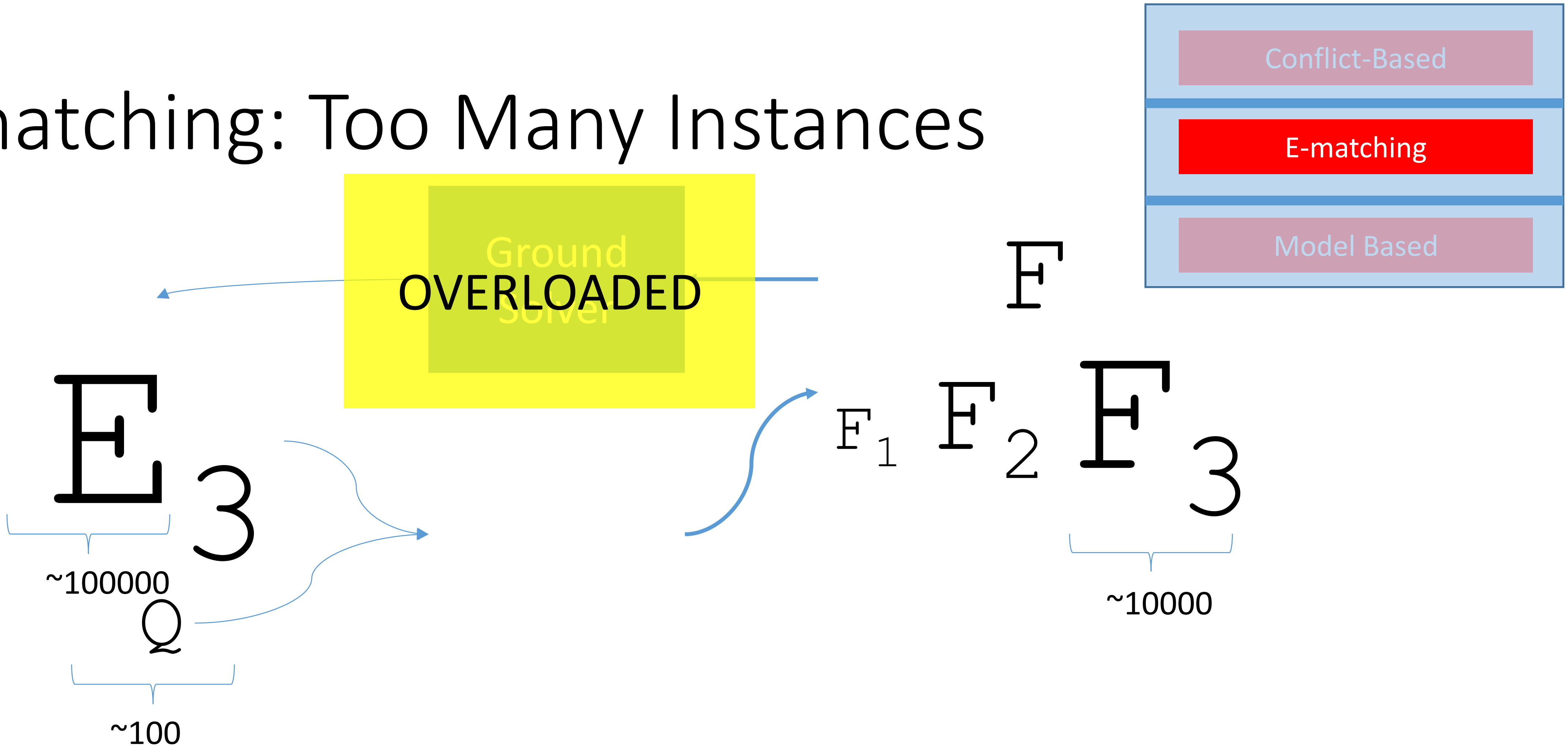


E-matching: Too Many Instances



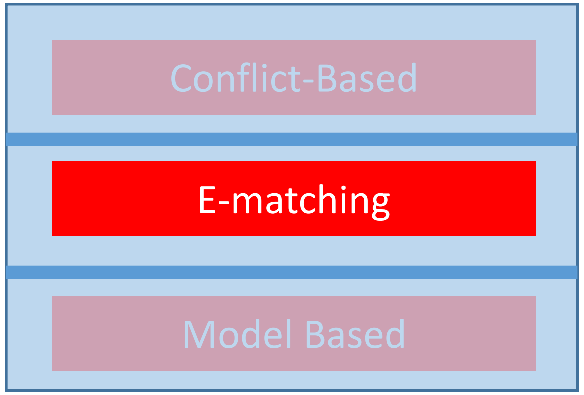
- Typical problems in applications:
  - F contains 1000s of clauses
  - Satisfying assignments contain 1000s of terms in E, 100s of ∀ in Q
  - Leads to 100s, 1000s, 10000s of instances

E-matching: Too Many Instances



⇒ Ground solver is overloaded, loop becomes slow,
...solver times out

E-matching: Too Many Instances



# Instances	cvc3 #	cvc3 %	cvc4 #	cvc4 %	z3 #	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)

E-matching: Too Many Instances

Conflict-Based

E-matching

Model Based

E: a=f(a), a=f(b), P(a,...,a)

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

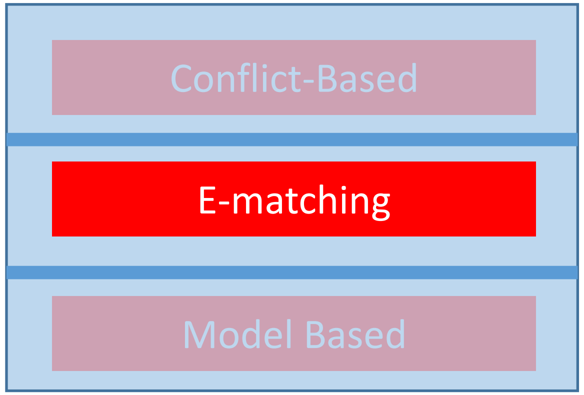
return

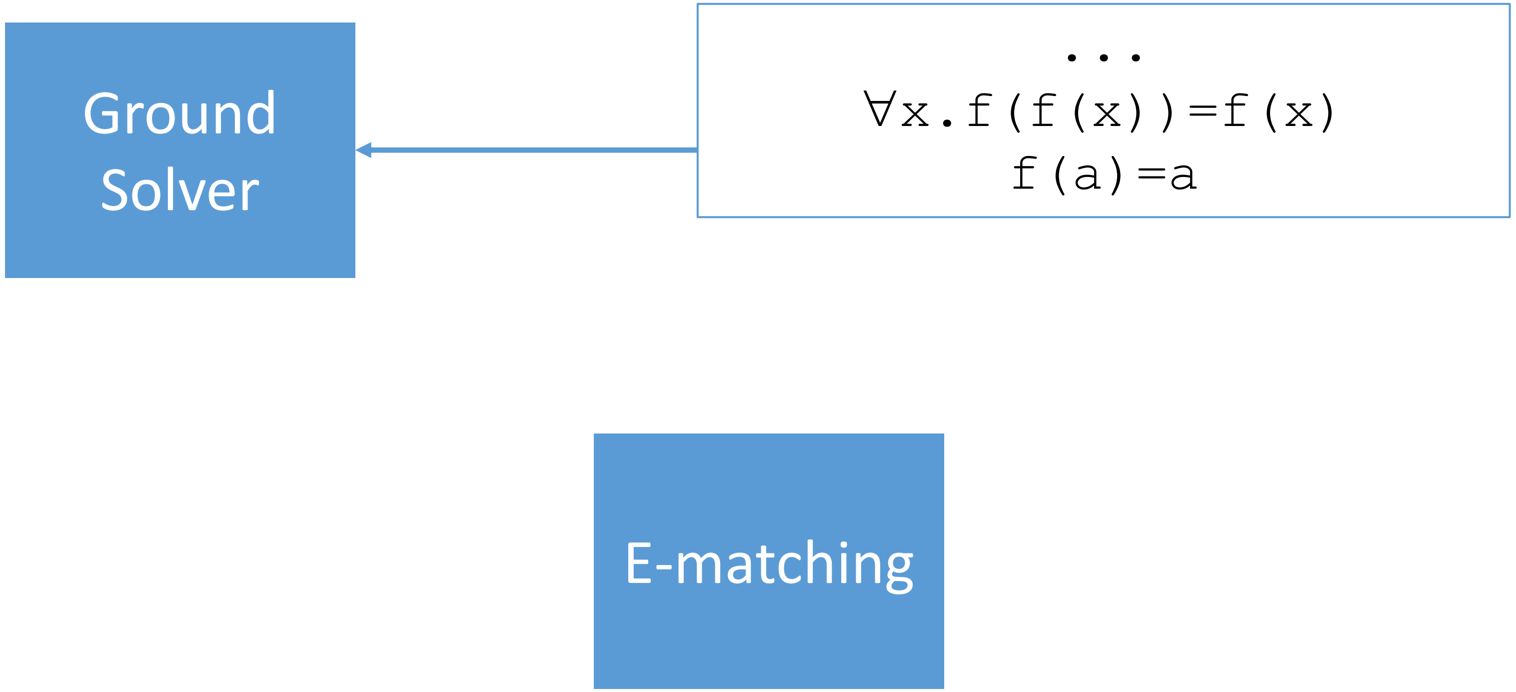
_ ⇒P(...,f(**a**),f(**a**))
_ ⇒P(...,f(**a**),f(**b**))
_ ⇒P(...,f(**b**),f(**a**))
_ ⇒P(...,f(**b**),f(**b**))
...

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

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

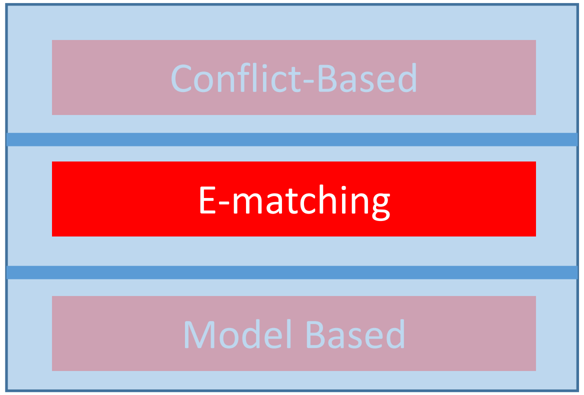
E-matching: Non-termination

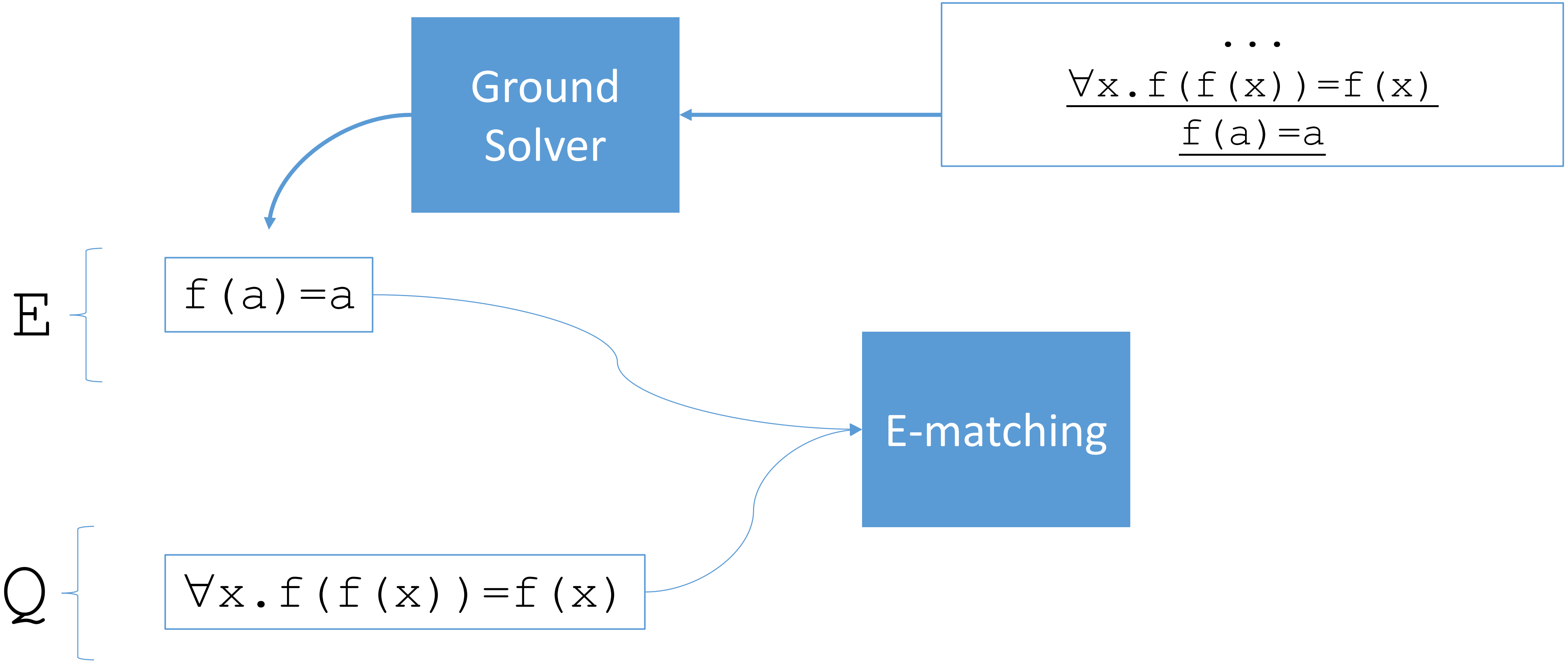




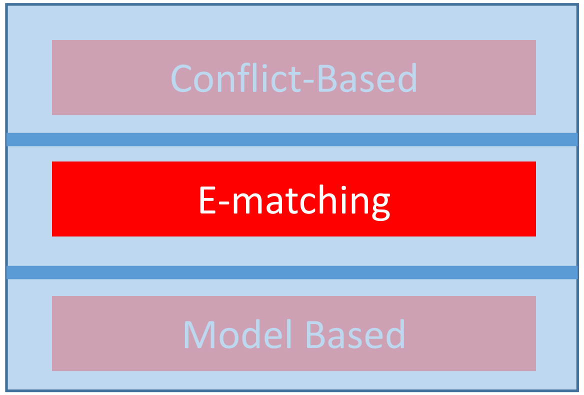
$\Rightarrow$ E-matching may be non-terminating

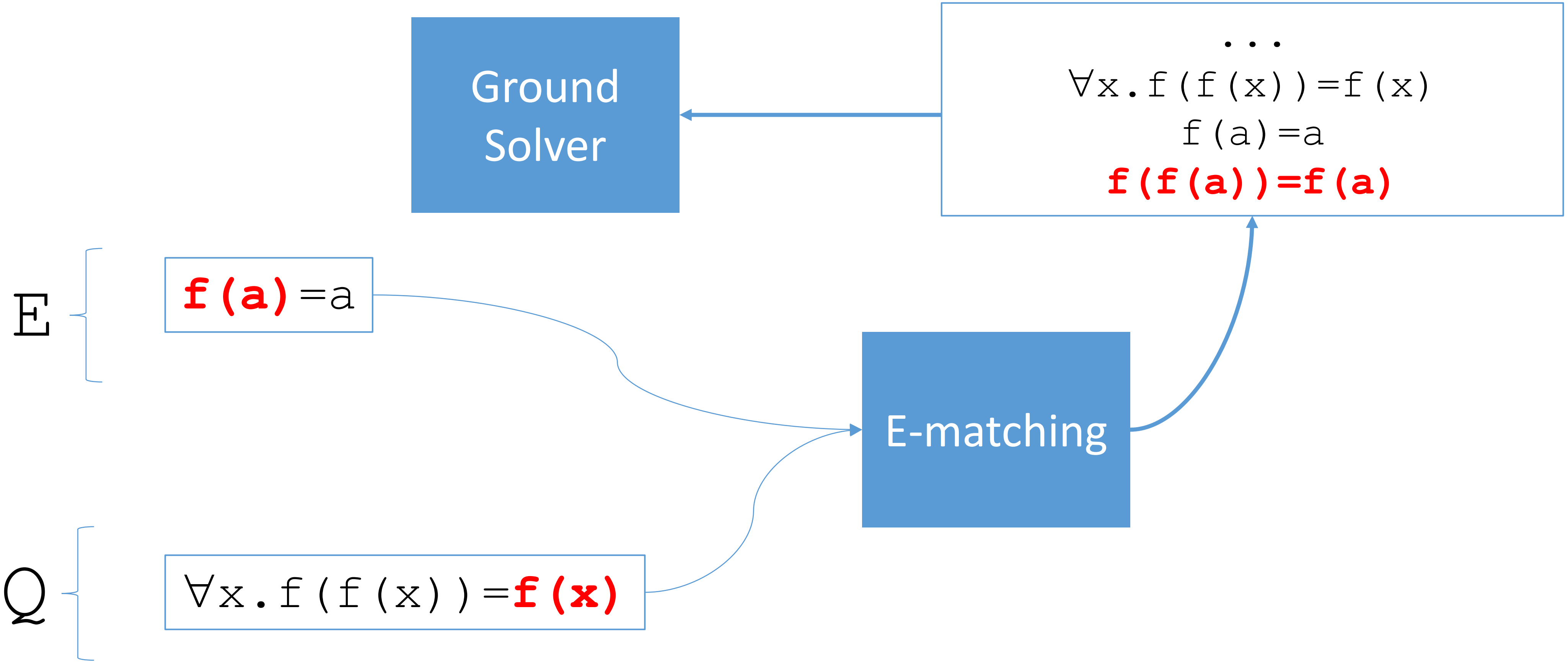
E-matching: Non-termination



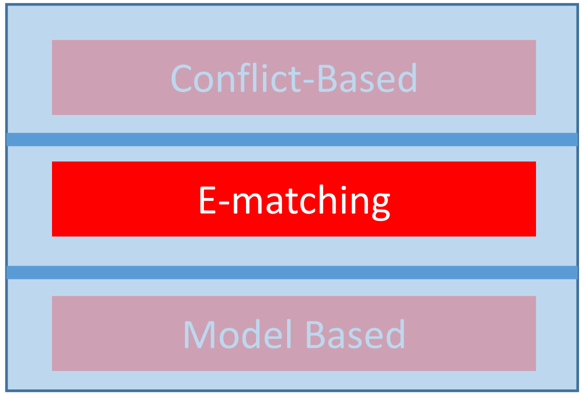


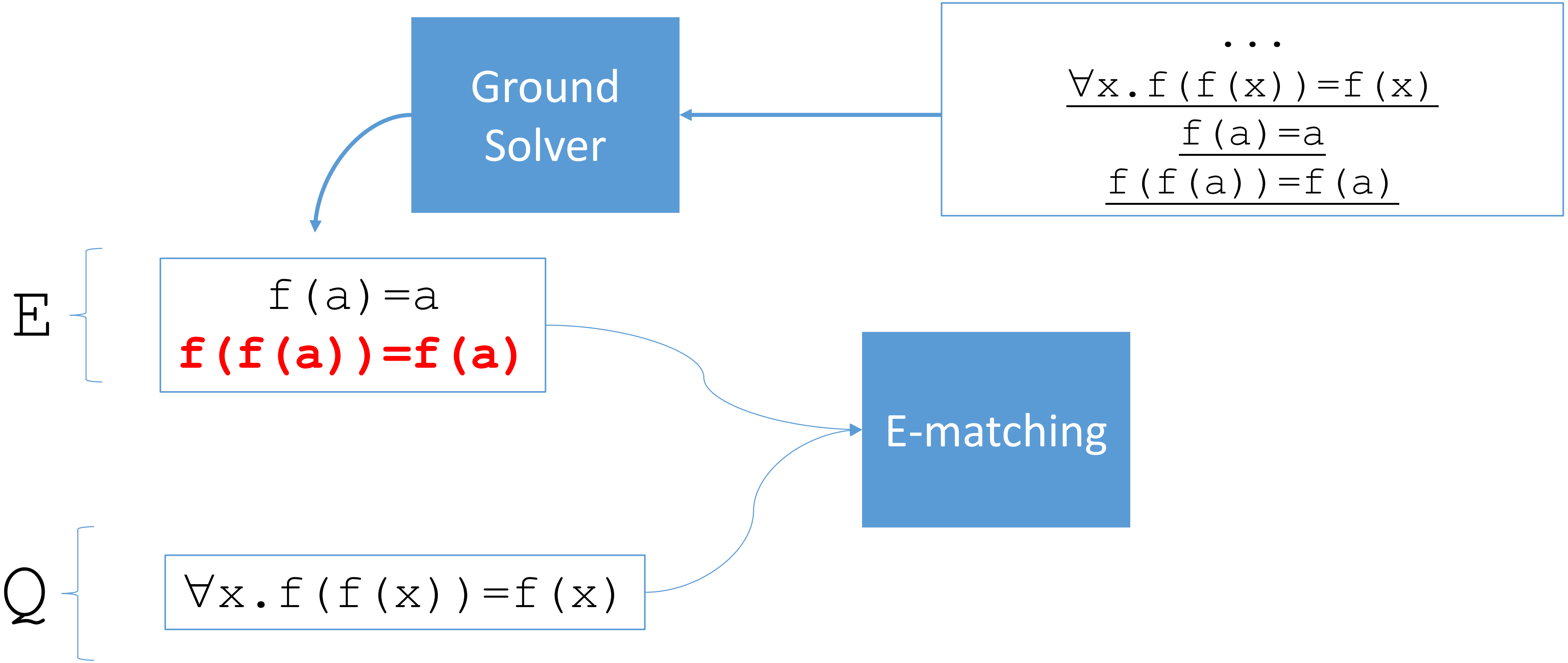
E-matching: Non-termination



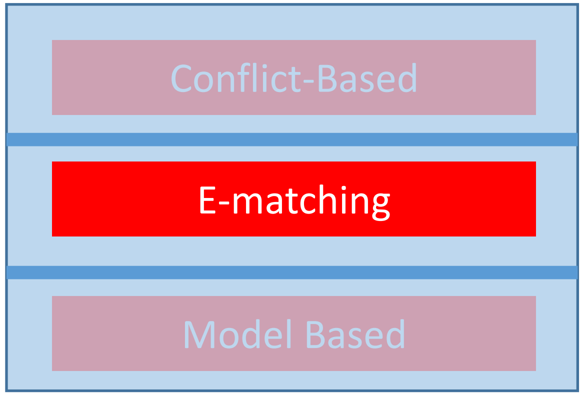


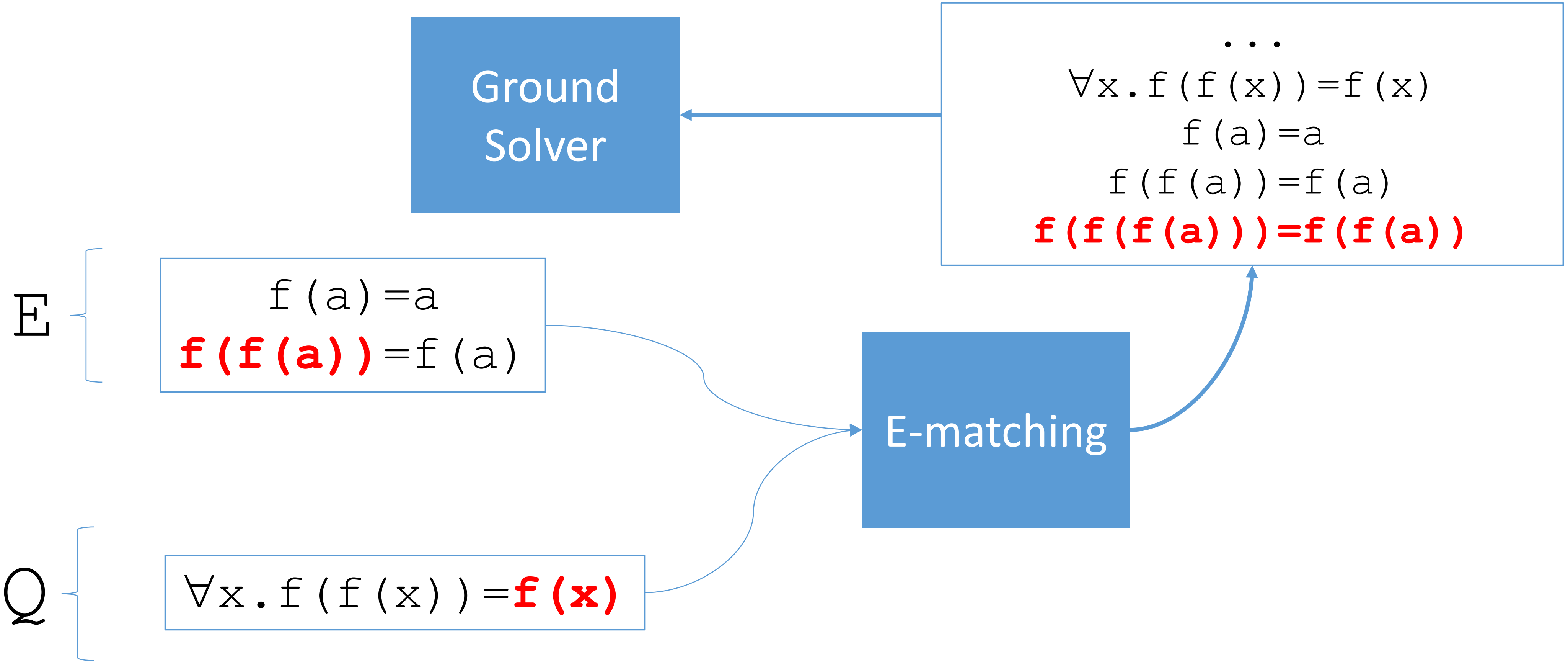
E-matching: Non-termination





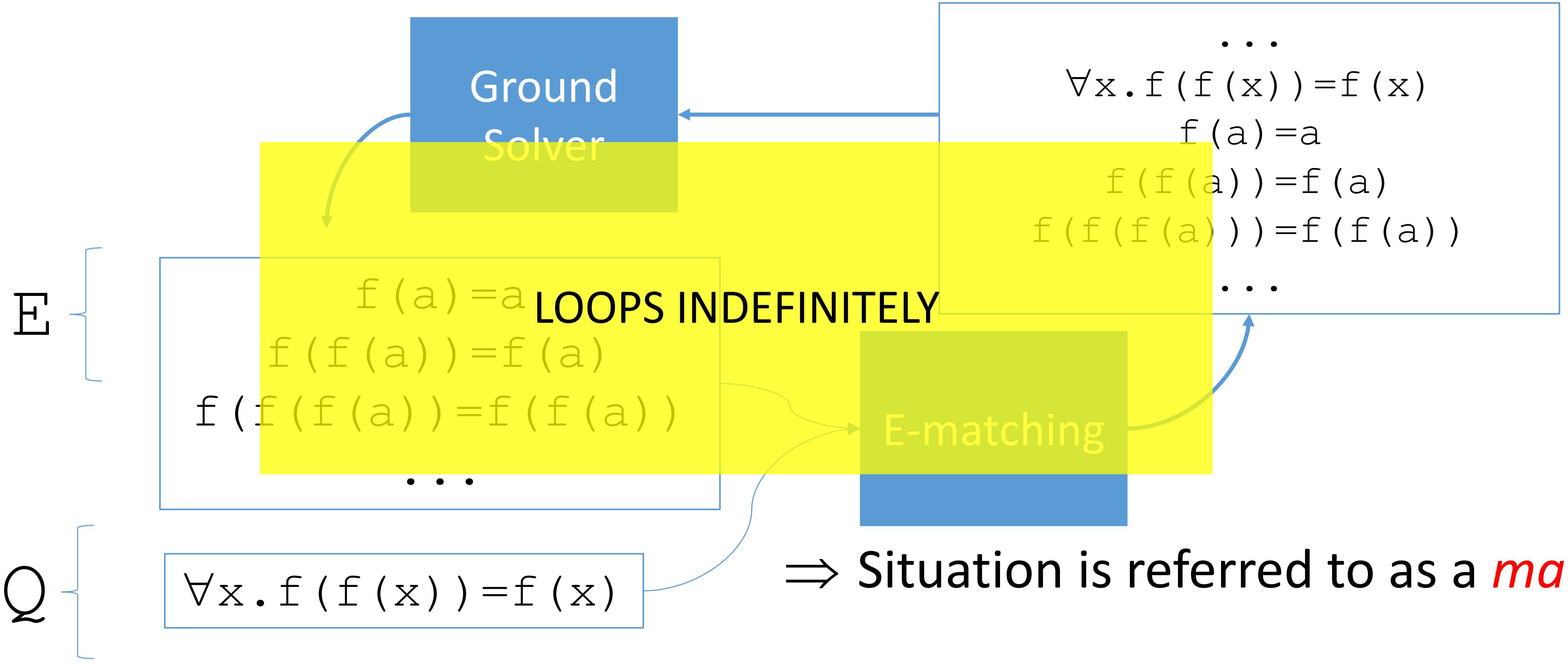
E-matching: Non-termination





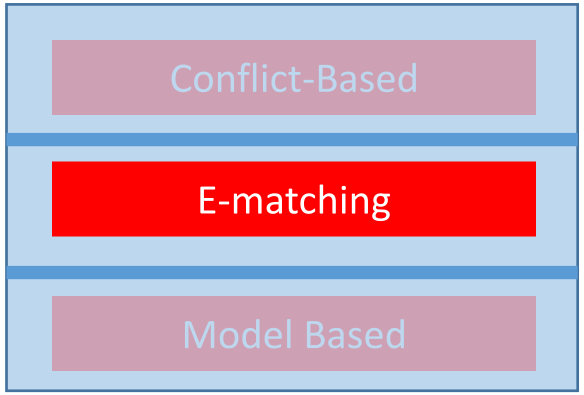
E-matching: Non-termination

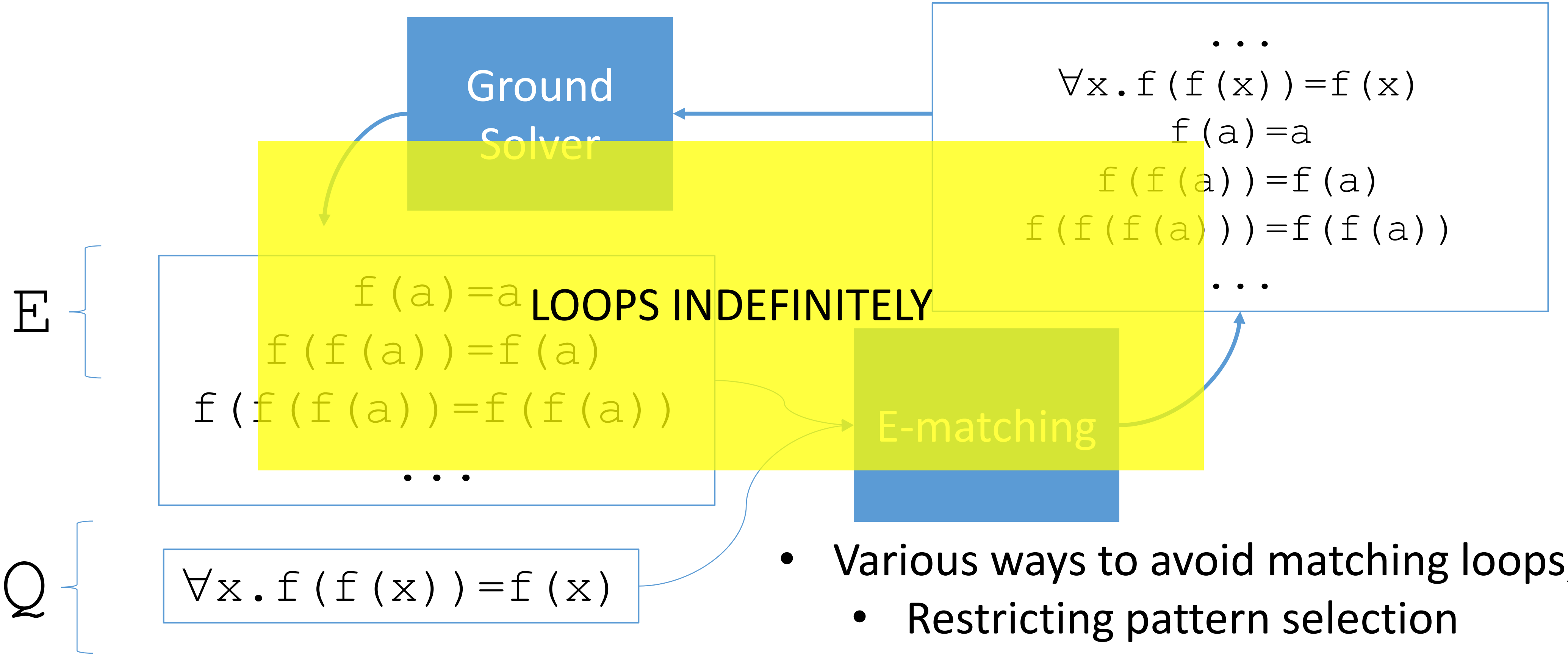
Conflict-Based
E-matching
Model Based



$\Rightarrow$ Situation is referred to as a *matching loop*

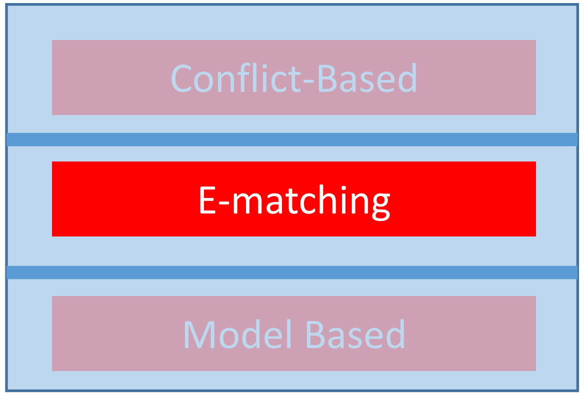
E-matching: Non-termination

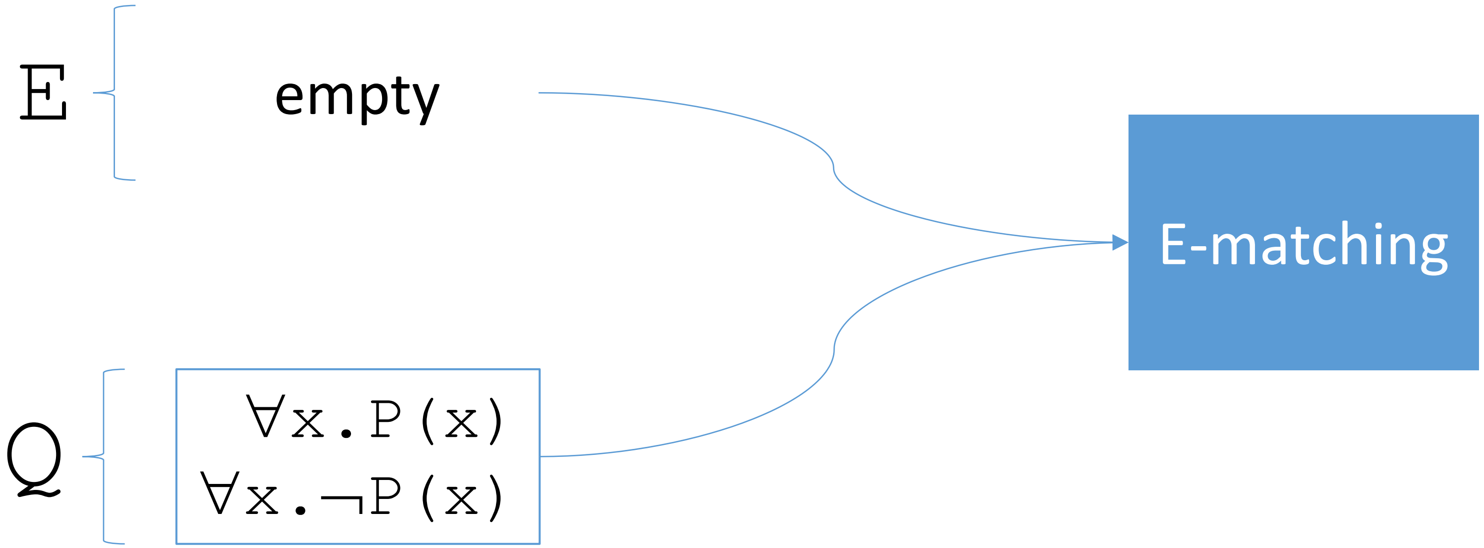




- Various ways to avoid matching loops, e.g. by:
  - Restricting pattern selection
  - Fair instantiations strategies (track "levels")

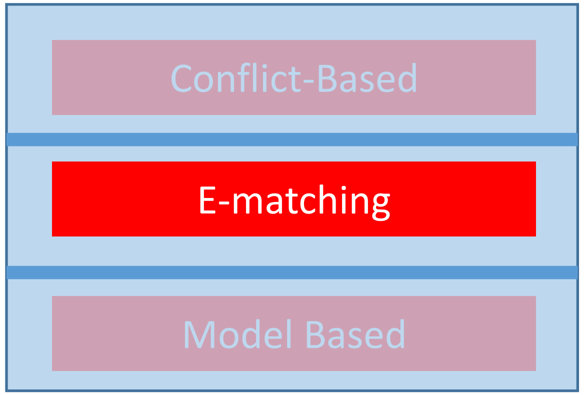
E-matching: Incompleteness

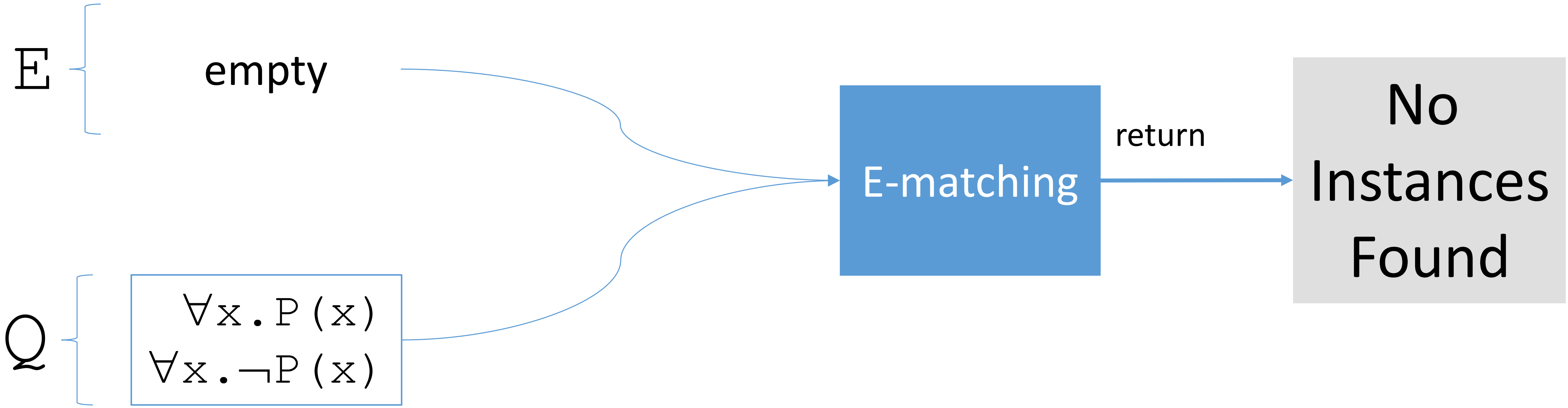




$\Rightarrow$ E-matching is an incomplete procedure

E-matching: Incompleteness





$\Rightarrow$ If E-matching produces no instances,

this *does not guarantee $E \cup Q$ is T-satisfiable*

E-matching: Summary

- Using matching ground terms from `E` against patterns in `Q`:
  - **From `Q`, learn constraints about ground terms `g` from `E`**

E-matching: Summary

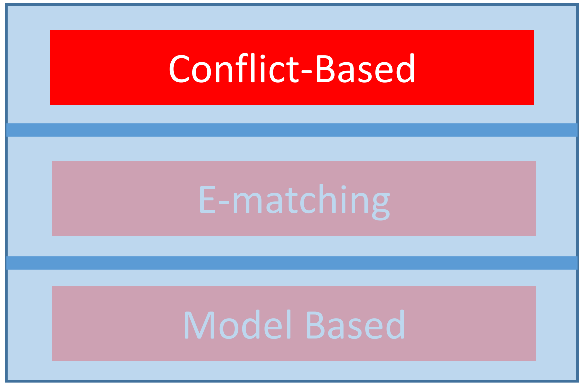
- Using matching ground terms from `E` against patterns in `Q`:
  - From `Q`, learn constraints about ground terms `g` from `E`
- Challenges
  - What can we do when there **too many instances** to add?
  - What can we do when there are **no instances** to add, problem is "**sat**"?

E-matching: Summary

- Using matching ground terms from E against patterns in Q:
  - From Q, learn constraints about ground terms g from E
- Challenges
  - What can we do when there **too many instances** to add?

    $\Rightarrow$*Use conflict-based instantiation* **[Reynolds/Tinelli/deMoura FMCAD14]**
  - What can we do when there are **no instances** to add, problem is "**sat**"?

    $\Rightarrow$*Use model-based instantiation* **[Ge/deMoura CAV09]**



Conflict-Based Instantiation

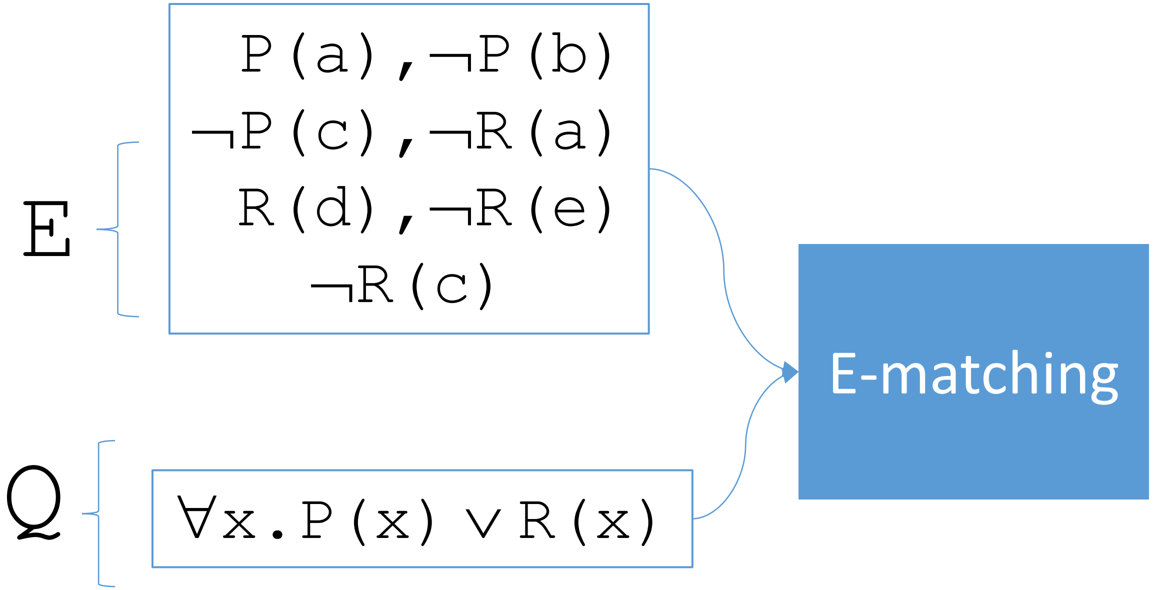
- Implemented in solvers:
  - CVC4 **[Reynolds et al 14]**, recently in VeriT **[Barbosa16]**
- Basic idea:
  1. Try to find a "conflicting" instance such that $E \cup \Psi\{\mathbf{x} \rightarrow \mathbf{t}\}$ implies $\bot$

     (by contrast, E-matching does not distinguish such instances)
  2. If one such instance can be found, return that instance only

     (and do not run E-matching)

$\Rightarrow$ *Leads to fewer instances, improved ability of ground solver to answer "unsat"*

Conflict-Based Instantiation



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** →

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

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

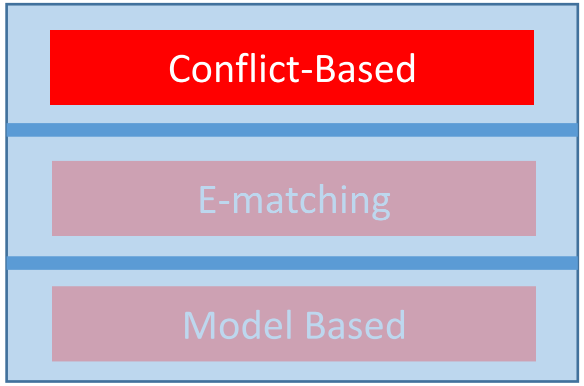
$(\forall x. P(x) \vee R(x)) \Rightarrow P(\mathbf{c}) \vee R(\mathbf{c})$

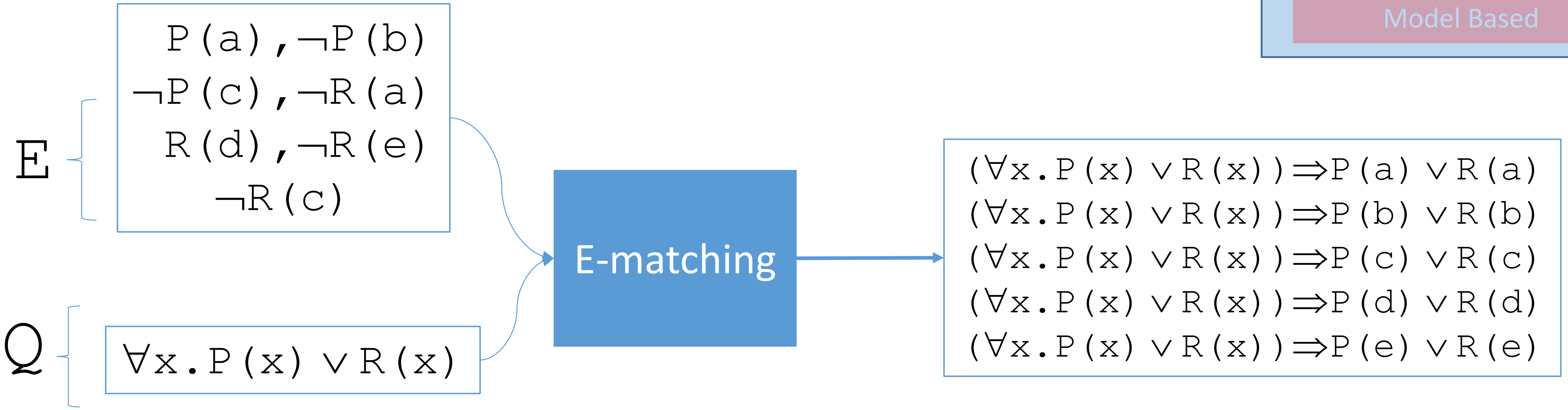
$(\forall x. P(x) \vee R(x)) \Rightarrow P(\mathbf{d}) \vee R(\mathbf{d})$

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

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

Conflict-Based Instantiation





⇒ Consider what we learn from these instances:

$$E, Q, P(a) \vee R(a) \not\models P(a) \vee R(a)$$
$$E, Q, P(b) \vee R(b) \not\models P(b) \vee R(b)$$
$$E, Q, P(c) \vee R(c) \not\models P(c) \vee R(c)$$
$$E, Q, P(d) \vee R(d) \not\models P(d) \vee R(d)$$
$$E, Q, P(e) \vee R(e) \not\models P(e) \vee R(e)$$

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

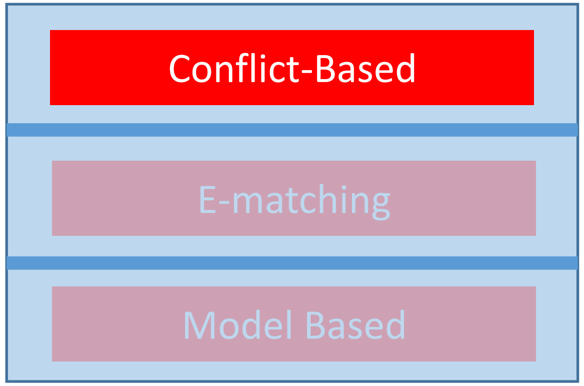
E-matching

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

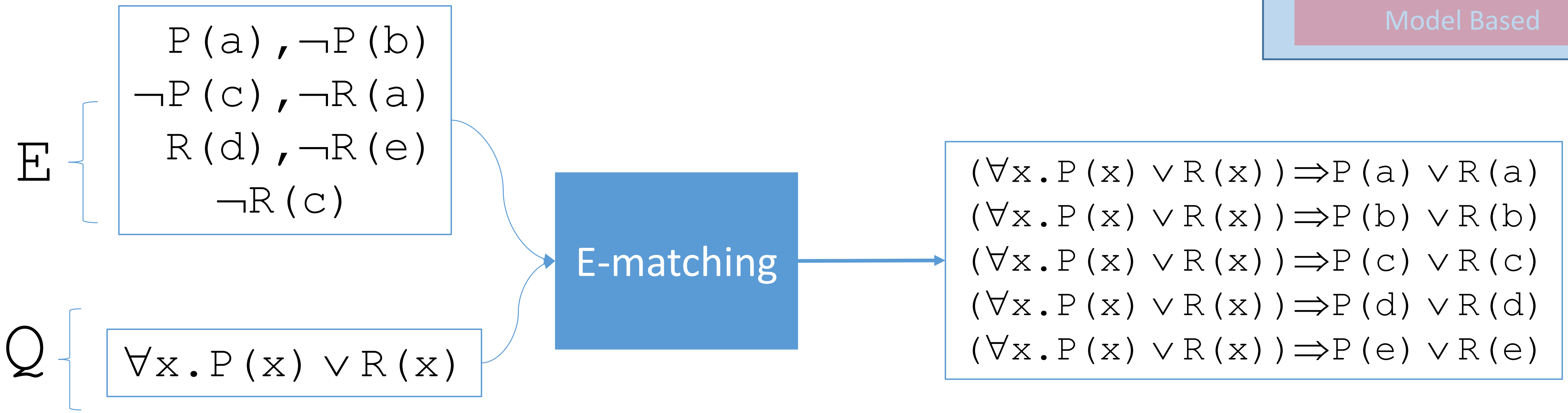
$\Rightarrow$ Consider what we learn from these instances:

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

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



Conflict-Based Instantiation



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

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

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

E-matching

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

⇒ Consider what we learn from these instances:

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

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:

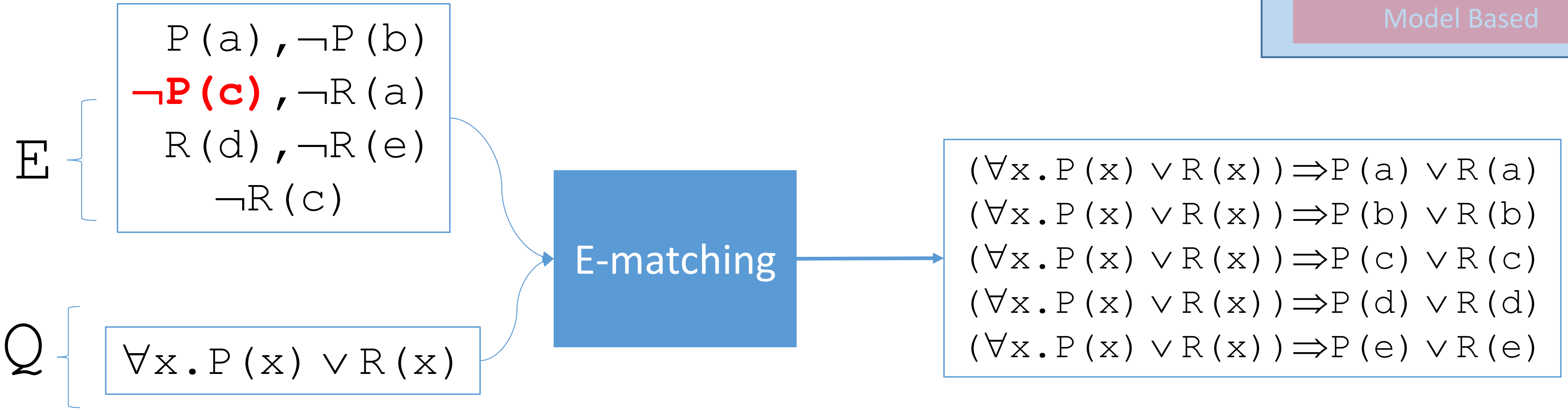
$$E, Q, P(a) \vee R(a) \models \top$$
$$E, Q, P(b) \vee R(b) \models R(b)$$
$$E, Q, P(c) \vee R(c) \models P(c) \vee R(c)$$
$$E, Q, P(d) \vee R(d) \models P(d) \vee R(d)$$
$$E, Q, P(e) \vee R(e) \models P(e) \vee R(e)$$

Conflict-Based Instantiation

Conflict-Based

E-matching

Model Based



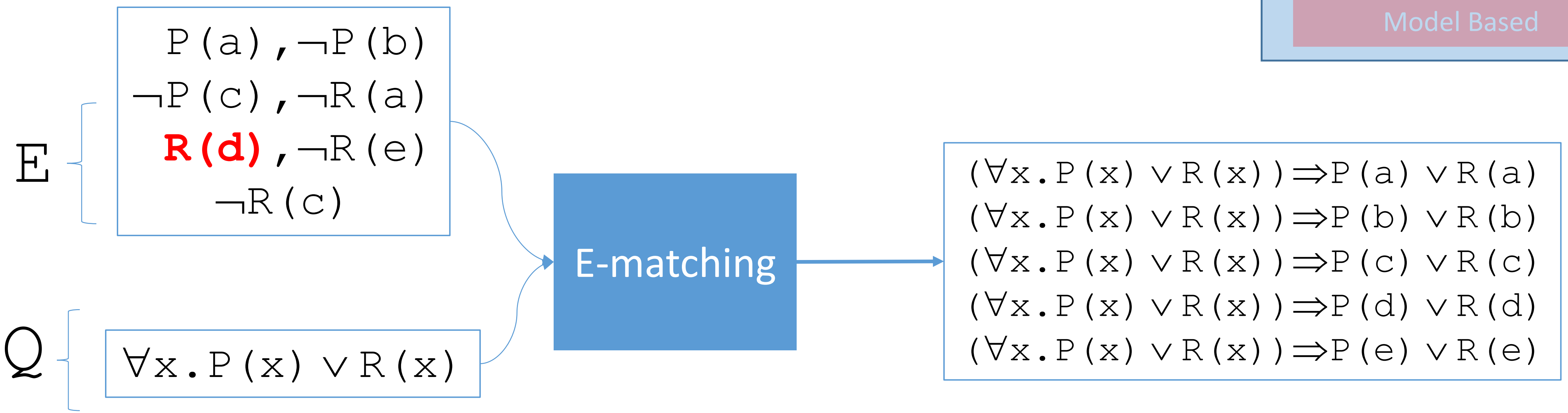
⇒ Consider what we learn from these instances:

$$E, Q, P(a) \lor R(a) \models \top$$
$$E, Q, P(b) \lor R(b) \models R(b)$$
$$E, Q, P(c) \lor R(c) \models R(c)$$
$$E, Q, P(d) \lor R(d) \models P(d) \lor R(d)$$
$$E, Q, P(e) \lor R(e) \models P(e) \lor R(e)$$

We know **P(c)** $\Leftrightarrow \bot$

Conflict-Based Instantiation

Conflict-Based
E-matching
Model Based



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

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

We know **R(d)** $\Leftrightarrow$ **T**

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:

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

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

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation

E: $P(a), \neg P(b)$, $\neg P(c), \neg R(a)$, $R(d), \neg R(e)$, $\neg \mathbf{R(c)}$

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:

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

We know $\mathbf{R(c)} \Leftrightarrow \bot$

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:

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

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:

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

→ 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:

$E, Q, P(a) \vee R(a) \models \top$
$E, Q, P(b) \vee R(b) \models R(b)$
$\mathbf{E, Q, P(c) \vee R(c) \models \bot}$
$E, Q, P(d) \vee R(d) \models \top$
$E, Q, P(e) \vee R(e) \models P(e)$

*$P(c) \vee R(c)$ is a **conflicting instance** for (E, Q)!*

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

→ **Conflict-based Instantiation** →

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

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

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

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

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

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

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

Since $P(c) \vee R(c)$ suffices to derive $\bot$, return ***only*** this instance

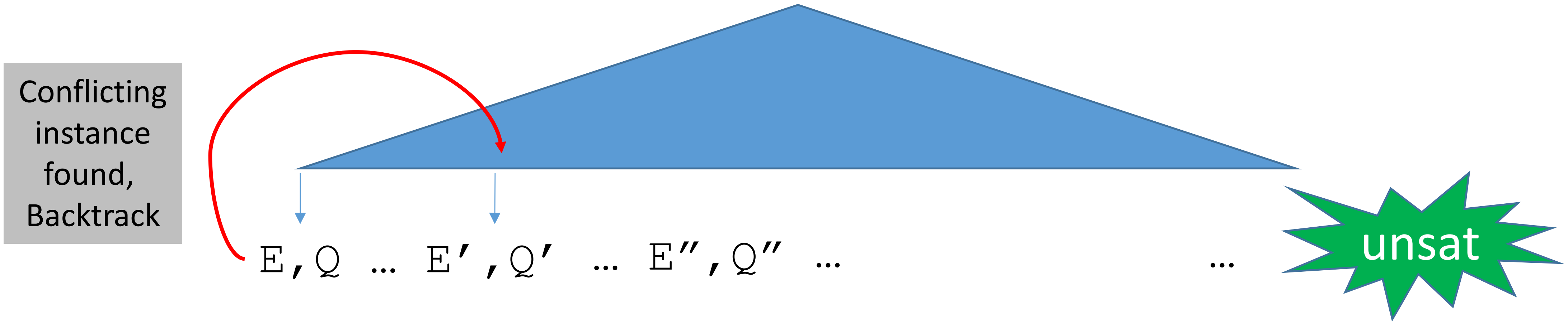
Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation

- Why are conflicts important?
  - As with the ground case, they prune the search space of DPLL(T)
    - Given a conflicting instance for (E, Q) is added to the clause set F
      - Solver is forced to choose a new sat assignment (E′, Q′)

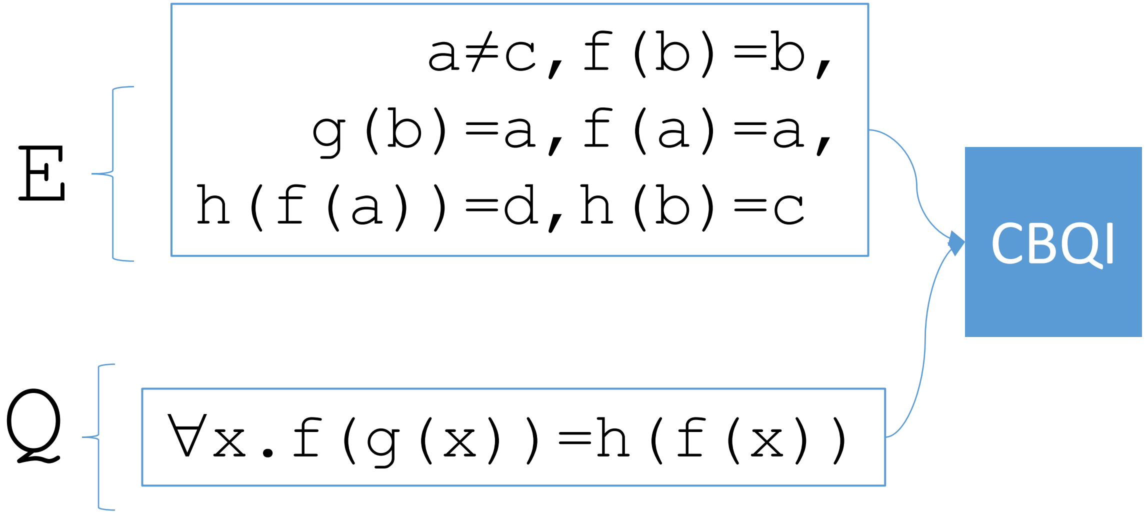


Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF



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

⇒ Consider the instance `∀x.f(g(x))=h(f(x))` ⇒ `f(g(`**`b`**`))=h(f(`**`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

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

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF

E: a≠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))

Consider the *equivalence classes* of E

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

Conflict-Based Instantiation: EUF

- **Conflict-Based**
- E-matching
- Model Based

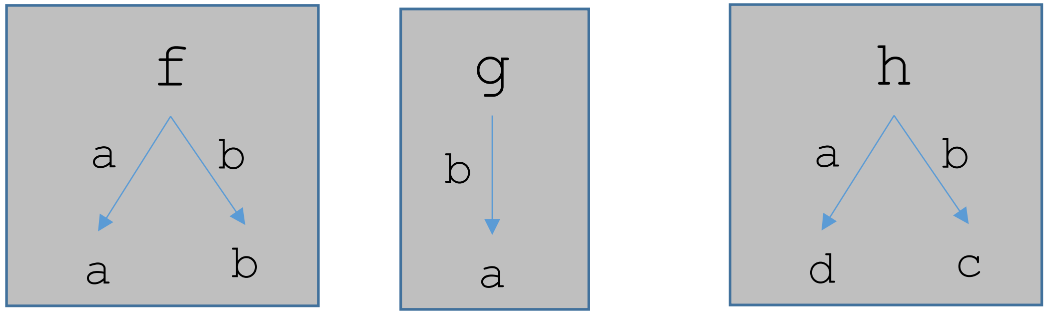
E: a≠c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c

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*

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

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF

E: a≠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))

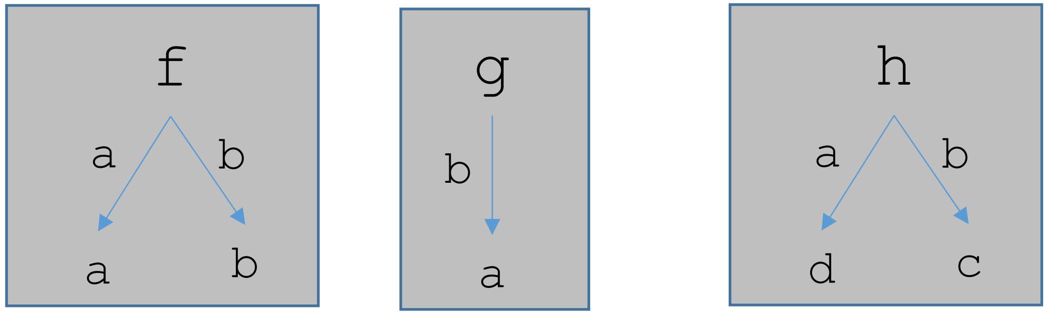
→ CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



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

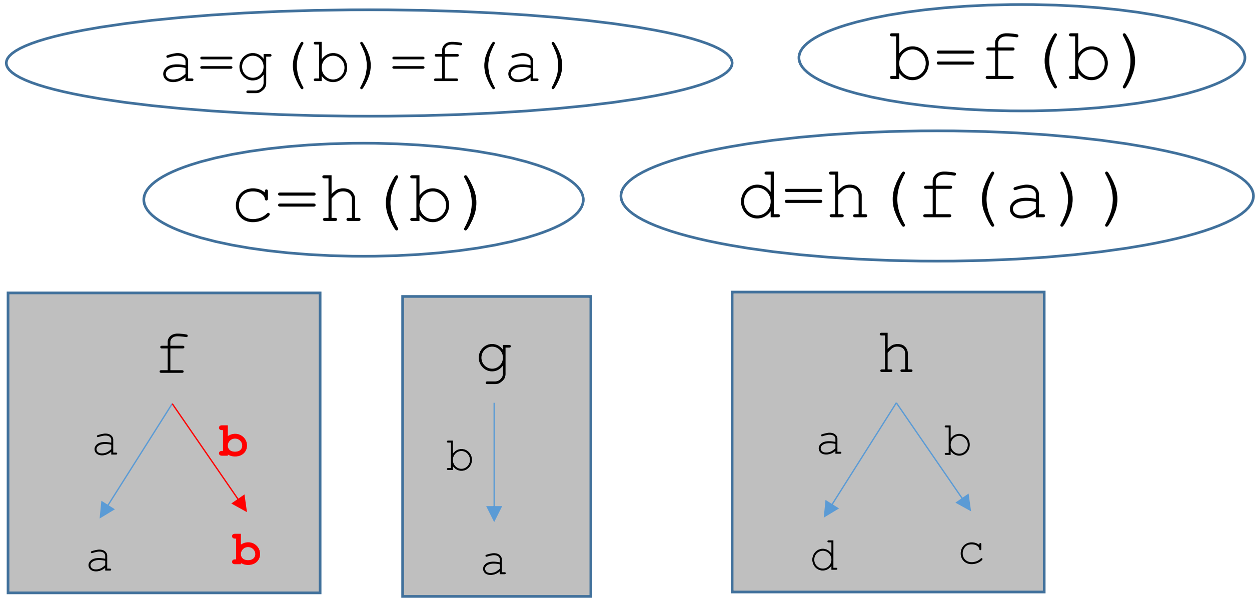
Conflict-Based Instantiation: EUF

Conflict-Based
E-matching
Model Based

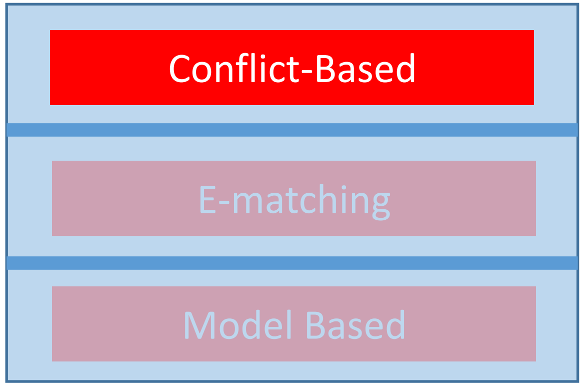
E { a≠c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c

Q { ∀x.f(g(x))=h(f(x))

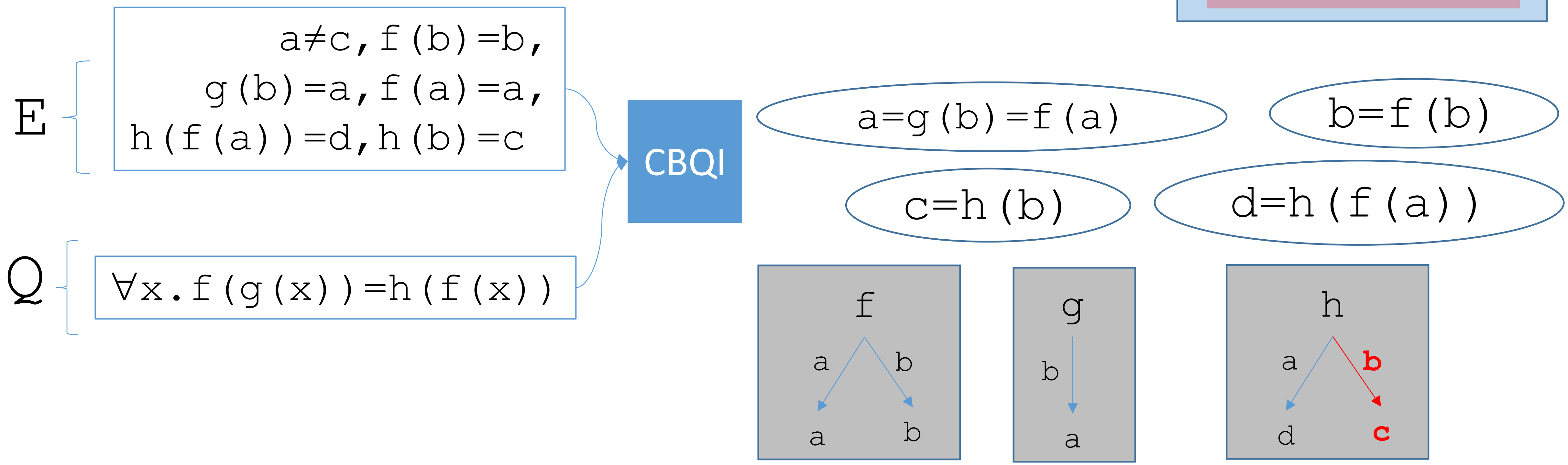
CBQI



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



Conflict-Based Instantiation: EUF



$E, Q, f(g(b))=h(f(b)) \models_E f(g(b)) = $ **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: $\forall x. f(g(x))=h(f(x))$

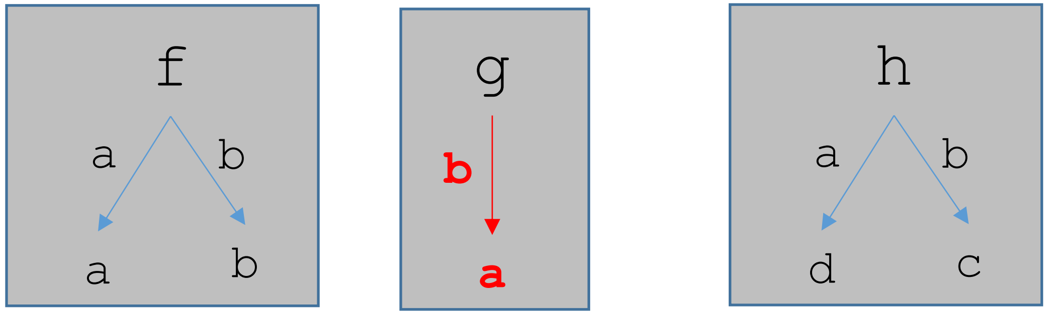
CBQI

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

$b=f(b)$

$c=h(b)$

$d=h(f(a))$



$E, Q, f(g(b))=h(f(b)) \models_E f(\ \mathbf{a}\ ) = \quad c$

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))

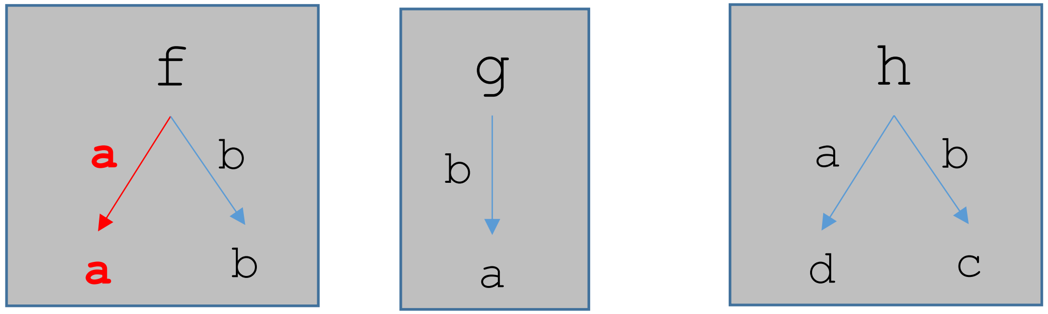
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



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

Conflict-Based

E-matching

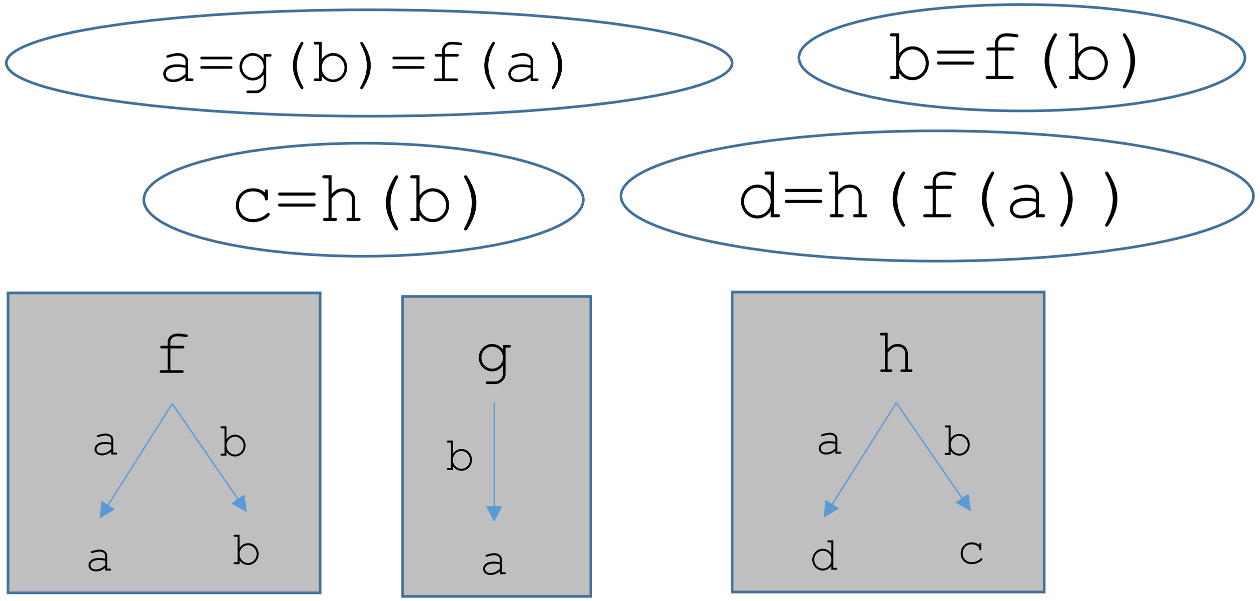
Model Based

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



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

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))

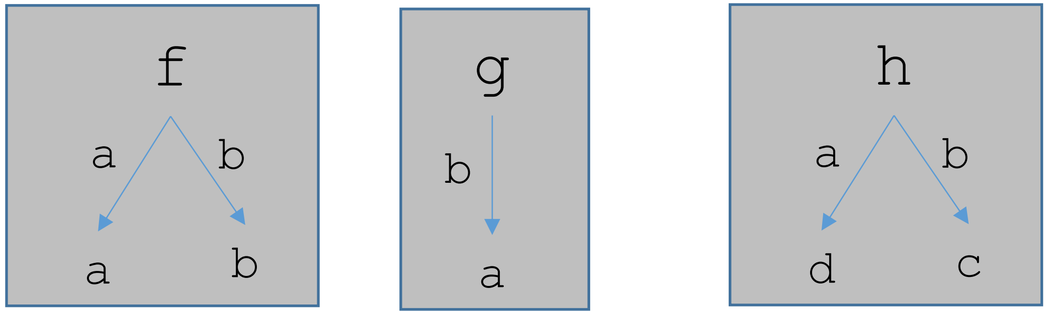
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



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

From E, we know **a≠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))

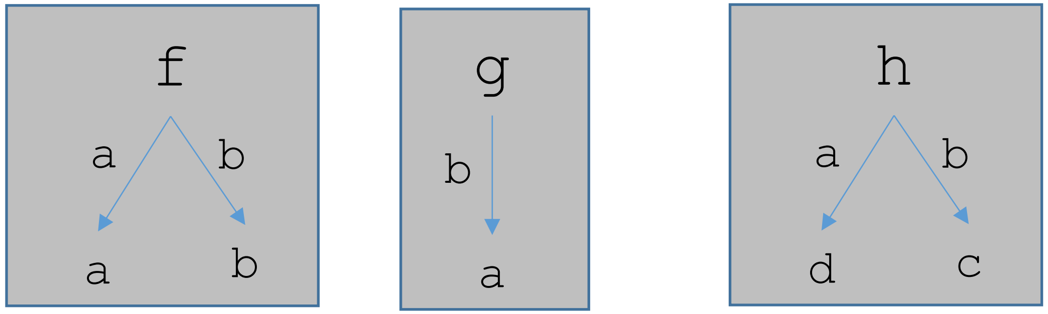
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



E,Q,f(g(b))=h(f(b)) $\models_E$ ⊥

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

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

⇒ 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))

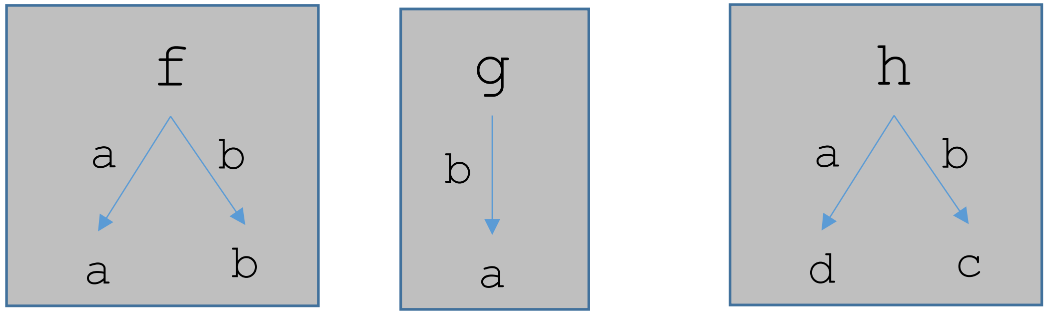
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

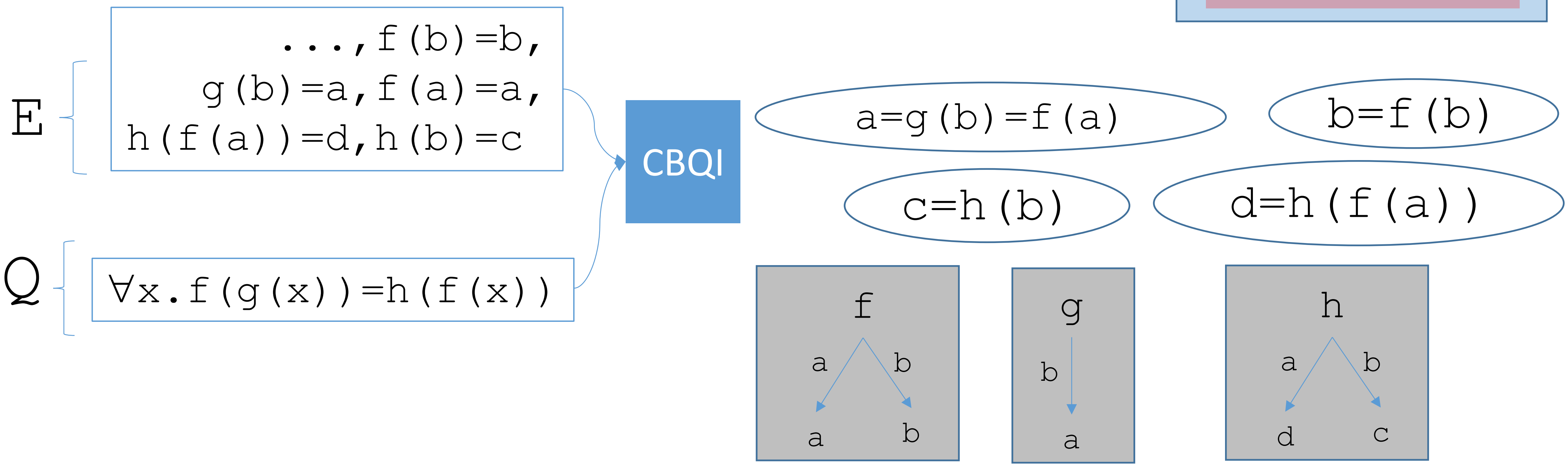
d=h(f(a))



Build partial definitions

Conflict-Based Instantiation: EUF

Conflict-Based
E-matching
Model Based



$E, Q, f(g(b))=h(f(b)) \models_E f(g(b))=h(f(b))$ **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: ∀x.f(g(x))=h(f(x))

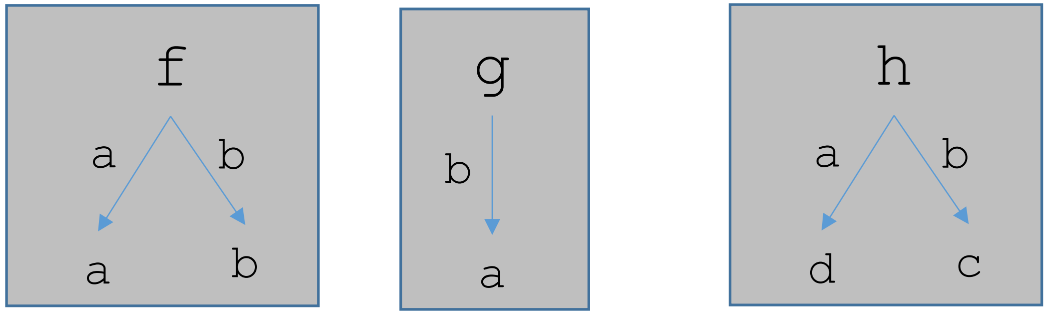
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



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

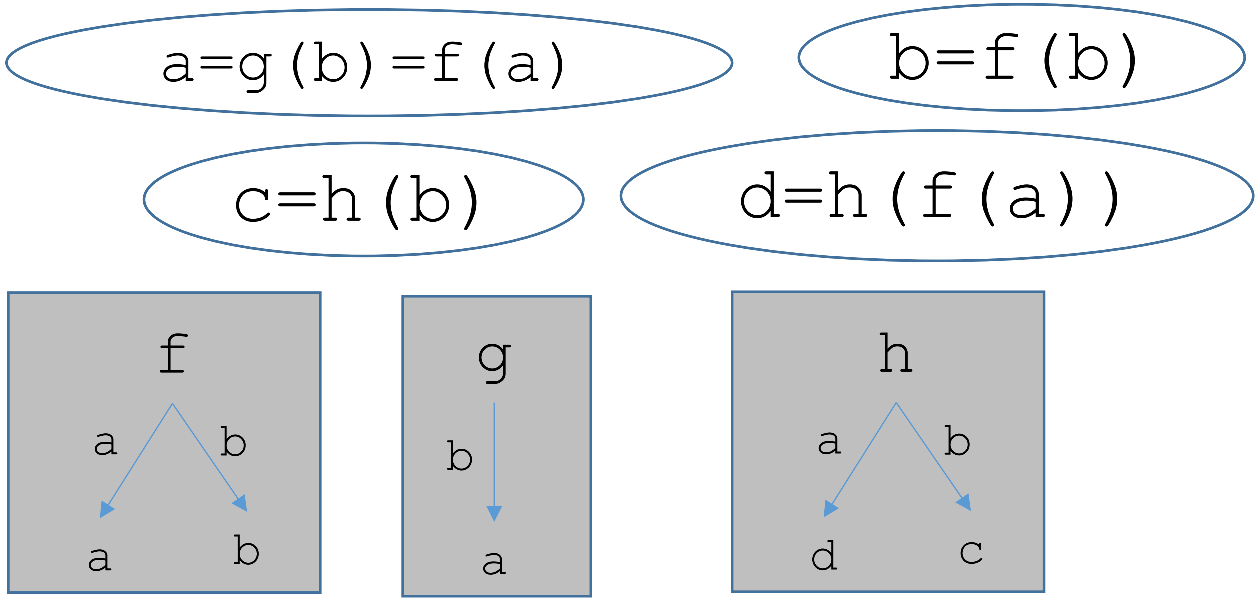
Conflict-Based Instantiation: EUF

- **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



E,Q,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

E-matching

Model Based

Conflict-Based Instantiation: EUF

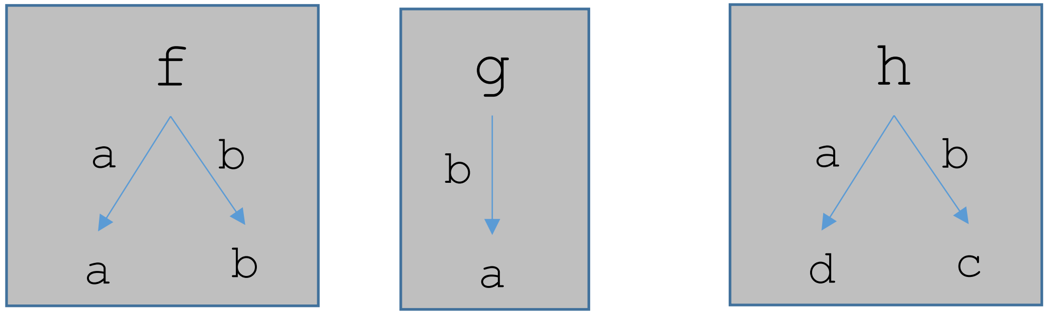
E: `…, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c`

Q: `∀x.f(g(x))=h(f(x))`

→ CBQI

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

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



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

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

$\Rightarrow$ *These are also useful*

Conflict-Based
E-matching
Model Based

Conflict-Based Instantiation

Given:

- Set of ground T-literals E
- Quantified formulas Q

- **Conflict-based instantiation**:
  1. If there exists a *conflicting instance* $\mathrm{E}, \Psi\{\mathbf{x} \rightarrow \mathbf{t}\} \models_T \bot$
     - Returns $\{\forall x. \Psi \Rightarrow \Psi\{\mathbf{x} \rightarrow \mathbf{t}\}\}$ only
  2. If there exists *propagating instance(s)*, $\mathrm{E}, \Psi_{\mathbf{i}}\{\mathbf{x} \rightarrow \mathbf{t}_{\mathbf{i}}\} \models_T s_i = u_i$, for $i=1, \ldots, n$
     - Returns $\{\forall x. \Psi_1 \Rightarrow \Psi_{\mathbf{1}}\{\mathbf{x} \rightarrow \mathbf{t}_{\mathbf{1}}\}, \ldots, \forall x. \Psi_n \Rightarrow \Psi_{\mathbf{n}}\{\mathbf{x} \rightarrow \mathbf{t}_{\mathbf{n}}\}\}$ only
  3. Otherwise:
     - Returns “unknown” (and the quantifiers module will resort to E-matching)

Conflict-Based Instantiation

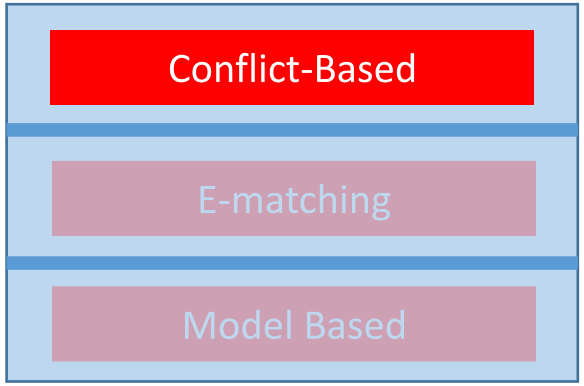
Conflict-Based
E-matching
Model Based

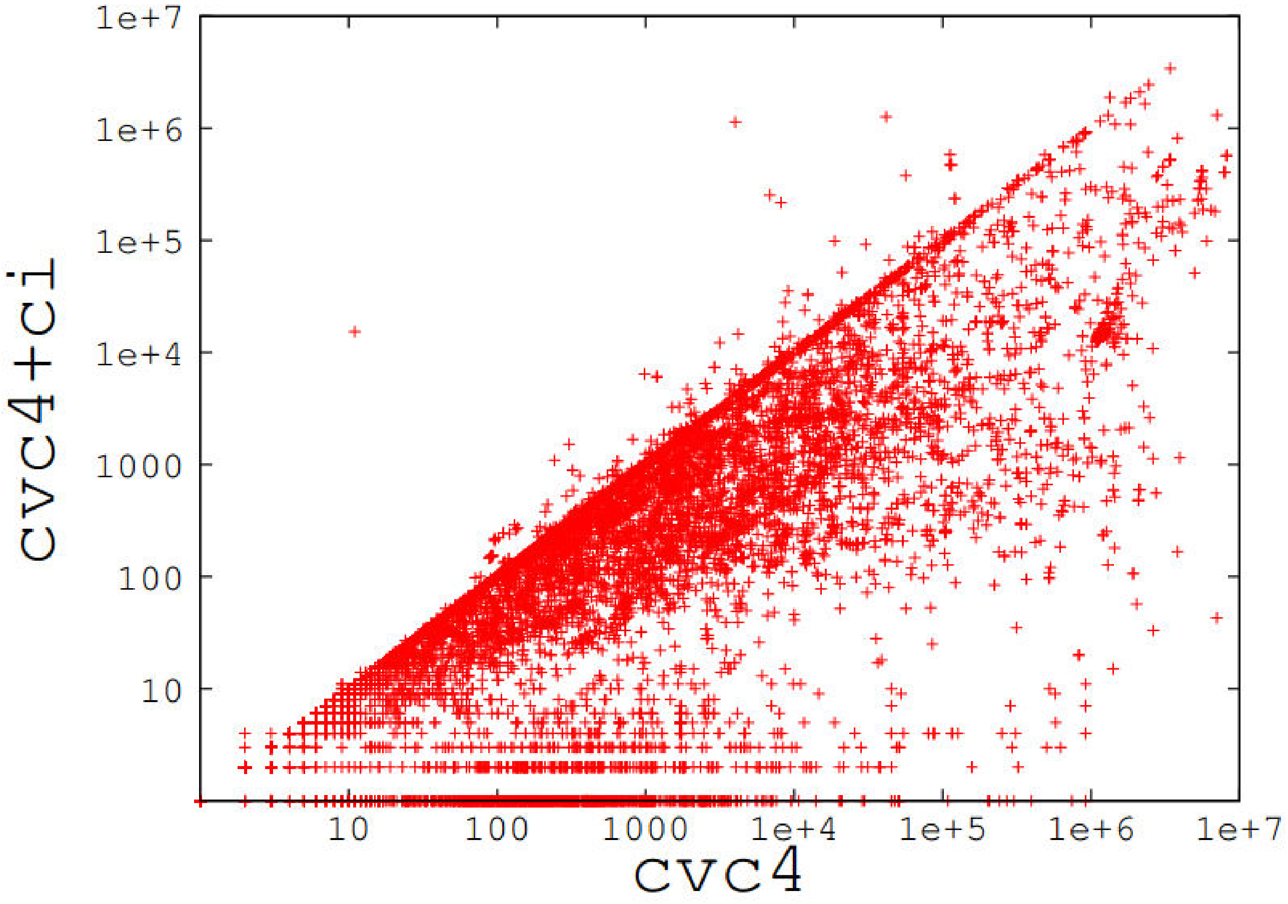
Given:
- Set of ground T-literals $\mathtt{E}$
- Quantified formulas $\mathtt{Q}$

usually restricted such that **T** is theory of equality

- **Conflict-based instantiation**:
  1. If there exists a *conflicting instance* $\mathtt{E}, \Psi\{\mathbf{x} \rightarrow \mathbf{t}\} \models_{T} \bot$
     - Returns $\{\forall x.\Psi \Rightarrow \Psi\{\mathbf{x} \rightarrow \mathbf{t}\}\}$ only
  2. If there exists *propagating instance(s)*, $\mathtt{E}, \Psi_{\mathbf{i}}\{\mathbf{x} \rightarrow \mathbf{t}_{\mathbf{i}}\} \models_{T} s_i = u_i$, for $i=1, \ldots, n$
     - Returns $\{\forall x.\Psi_1 \Rightarrow \Psi_{\mathbf{1}}\{\mathbf{x} \rightarrow \mathbf{t}_{\mathbf{1}}\}, \ldots, \forall x.\Psi_n \Rightarrow \Psi_{\mathbf{n}}\{\mathbf{x} \rightarrow \mathbf{t}_{\mathbf{n}}\}\}$ only
  3. Otherwise:
     - Returns "unknown" (and the quantifiers module will resort to E-matching)

Conflict-Based Instantiation: Impact



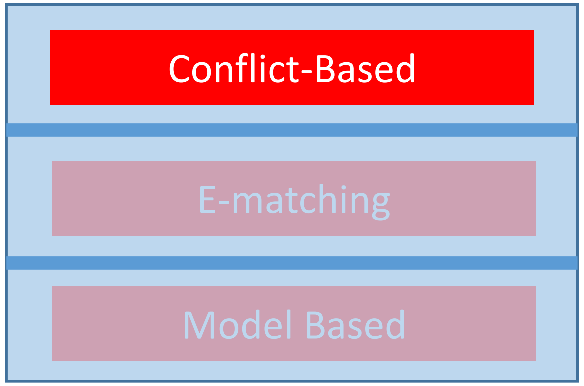


Reported number of instances.

- 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)

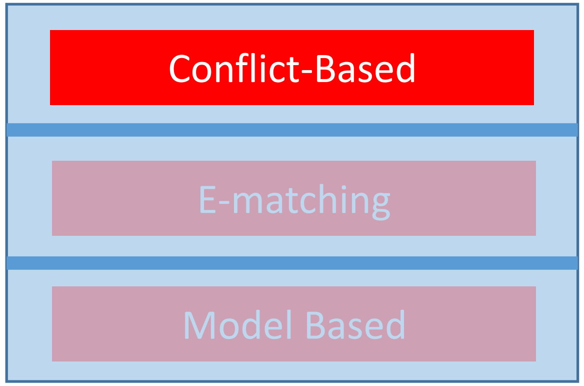
Conflict-Based Instantiation: Impact



- Conflicting instances found on ~75% of rounds (IR)
- Configuration **`cvc4+ci`**:
  - Calls E-matching 1.5x fewer times overall
  - As a result, returns 5x fewer instantiations

			E-matching		Conflict Inst.		Propagating Inst.	
		IR	% IR	# Inst	% IR	# Inst	% IR	# Inst
TPTP	**cvc4**	71,634	100.0	878,957,688				
	cvc4+ci	208,970	20.3	150,351,384	76.4	159,696	3.3	415,772
Isabelle	**cvc4**	6,969	100.0	119,008,834				
	cvc4+ci	21,756	22.4	28,196,846	64.0	13,932	13.6	130,864
SMT-LIB	**cvc4**	14,032	100.0	60,650,746				
	cvc4+ci	58,003	20.0	32,305,788	71.6	41,531	8.4	51,454

Conflict-Based Instantiation: Impact

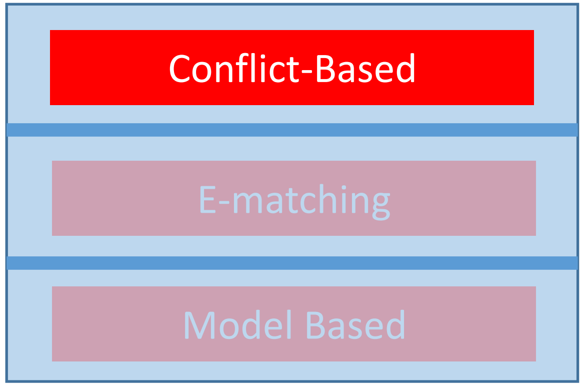


- CVC4 with conflicting instances **`cvc4+ci`**
  - Solves the most benchmarks for TPTP and Isabelle
  - Requires almost an order of magnitude fewer instantiations

	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

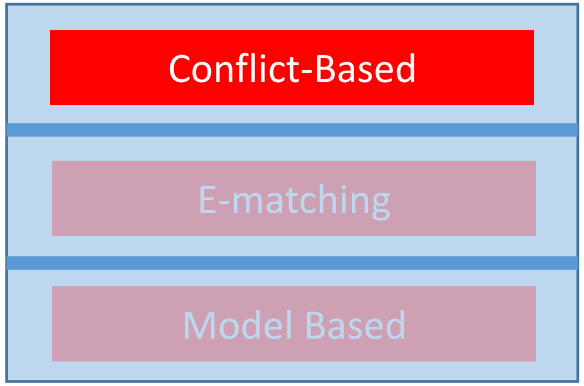
⇒ A number of hard benchmarks can be solved without resorting to E-matching at all

Conflict-Based Instantiation: Challenges



- How do we *find* conflicting instances?
- What about conflicts involving *multiple quantified formulas*?
- What if our quantified formulas that contain *theory symbols*?

Conflict-Based Instantiation: Challenges



- How do we *find* conflicting instances?

Conflict-Based
E-matching
Model Based

Conflict-Based Instantiation: Challenges

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

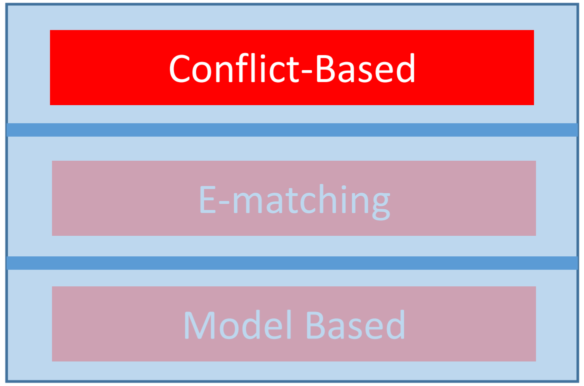
Conflict-Based
E-matching
Model Based

Conflict-Based Instantiation: Challenges

- How do we *find* conflicting instances?
  - Naively:
    1. Produce all instances $\Psi_1, \ldots, \Psi_n$ via E-matching for $(\mathtt{E},\mathtt{Q})$
    2. For $\mathtt{i=1}, \ldots, \mathtt{n}$, check if $\Psi_\mathtt{i}$ is a conflicting instance for $(\mathtt{E},\mathtt{Q})$

    $\Rightarrow$ *but n may be very large!*

Conflict-Based Instantiation: Challenges



- 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]**)

Conflict-Based Instantiation: Challenges

Conflict-Based
E-matching
Model Based

- What about conflicts involving *multiple quantified formulas*?

$$E \left\{ \begin{array}{c} P_0(a) \\ \neg P_{100}(a) \end{array} \right. \qquad Q \left\{ \begin{array}{c} \forall x. P_0(x) \Rightarrow P_1(x) \\ \forall x. P_1(x) \Rightarrow P_2(x) \\ \ldots \\ \forall x. P_{99}(x) \Rightarrow P_{100}(x) \end{array} \right.$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: Challenges

- What about conflicts involving *multiple quantified formulas*?

$$E \left\{ \begin{array}{c} P_0(a) \\ \neg P_{100}(a) \end{array} \right. \qquad Q \left\{ \begin{array}{c} \forall x. P_0(x) \Rightarrow P_1(x) \\ \forall x. P_1(x) \Rightarrow P_2(x) \\ \ldots \\ \forall x. P_{99}(x) \Rightarrow P_{100}(x) \end{array} \right.$$

- Want to find:

$$E, P_0(\mathbf{a}) \Rightarrow P_1(\mathbf{a}), P_1(\mathbf{a}) \Rightarrow P_2(\mathbf{a}), \ldots, P_{99}(\mathbf{a}) \Rightarrow P_{100}(\mathbf{a}) \models_E \bot$$

$\Rightarrow$ Current implementations would take 100 rounds to infer this

Conflict-Based Instantiation: Challenges

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`

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: Challenges

- 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.:
  - $E, f(\mathbf{-3}+\mathbf{4}) > \mathbf{-3}+2*\mathbf{4} \models_{UFLIA} f(\mathbf{-3}+\mathbf{4}) > \mathbf{-3}+2*\mathbf{4}$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: Challenges

- 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` $\models_{UFLIA}$ `f(1)>5`

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: Challenges

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

$\mathbb{E}$ { `f(1)=5`

$\mathbb{Q}$ { `∀xy.f(x+y)>x+2*y`

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

By $\mathbb{E}$, we know **f(1)** = **5**

Conflict-Based Instantiation: Challenges

- 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` $\models_{UFLIA} \bot$

Conflict-Based Instantiation: Challenges

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`

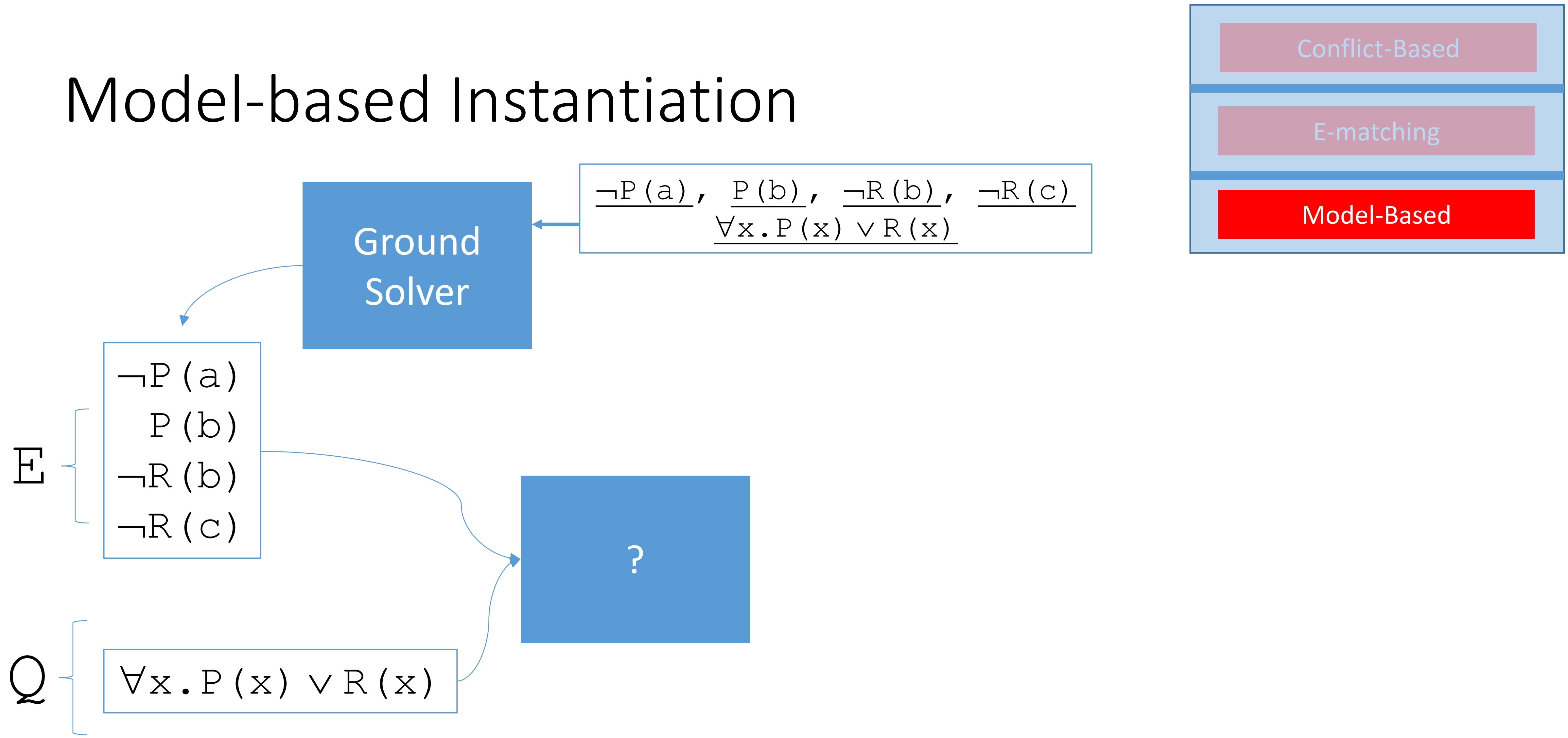
- Want to find, e.g.:
  - $E, f(-3+4) > -3+2*4 \models_{\mathbf{UFLIA}} \bot$

$\Rightarrow$ *In practice, finding such instances cannot be done efficiently*

Conflict-Based Instantiation: Summary

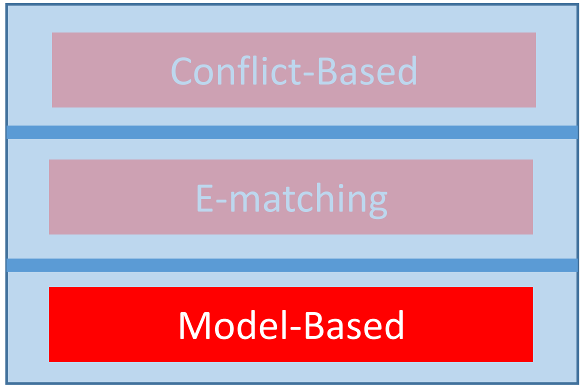
- Instantiation technique for (E, Q), where:
  - ⇒ ***From Q, derive conflicts $\bot$, and equalities $g_1=g_2$ between ground terms g_1, g_2 from E***
- Run with higher priority to E-matching
  - Resort to E-matching only if no conflicting or propagating instances can be found
- Leads to fewer instances, greater ability to answer "unsat"

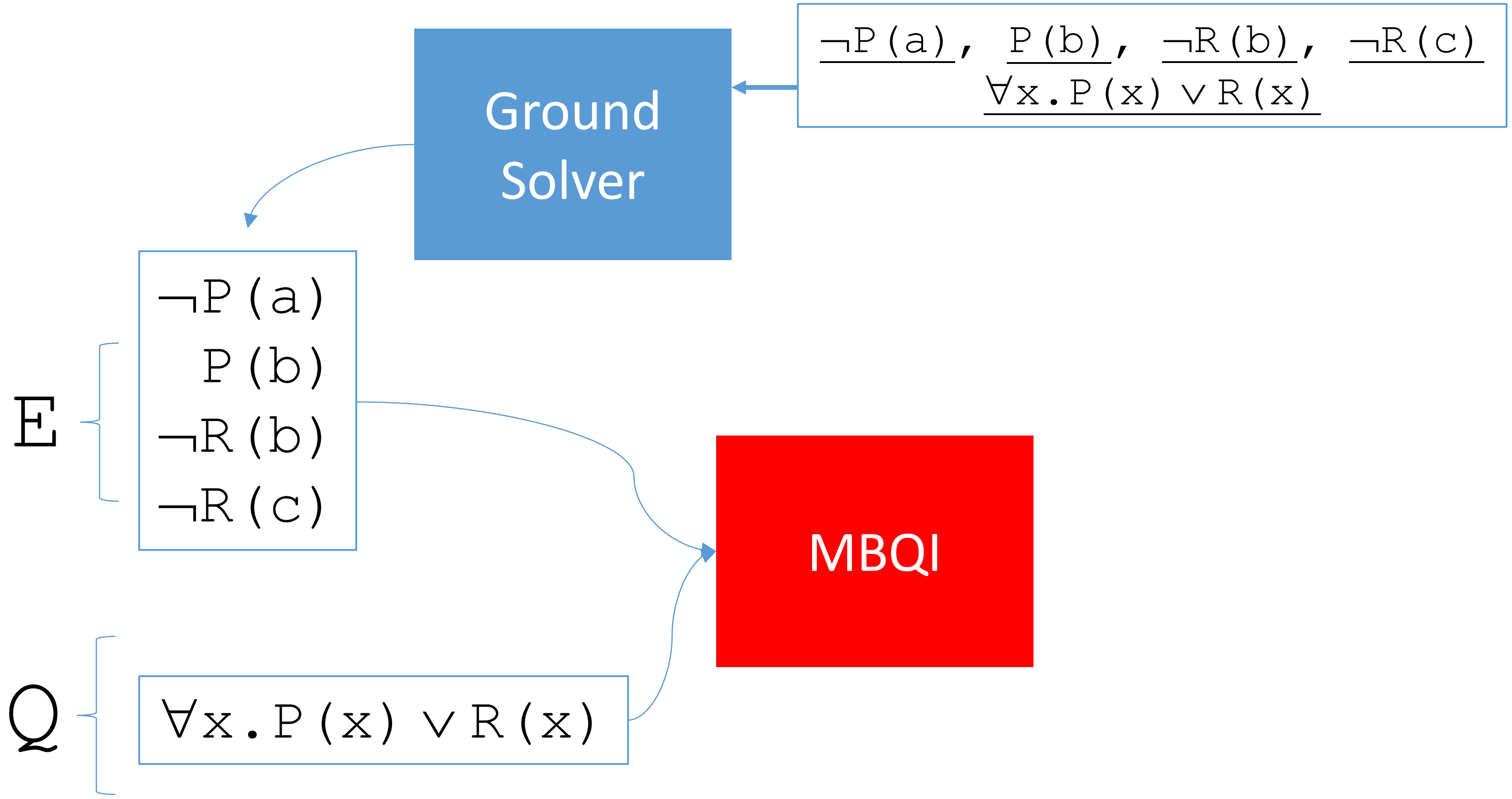
Model-based Instantiation



⇒ What if E∪Q is satisfiable?

Model-based Instantiation

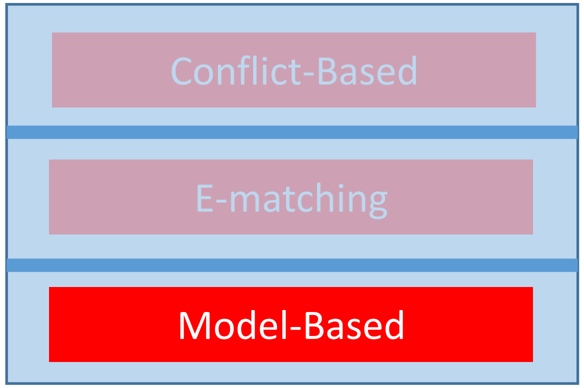




⇒ What if $E \cup Q$ is satisfiable?

- Use model-based quantifier instantiation (MBQI)

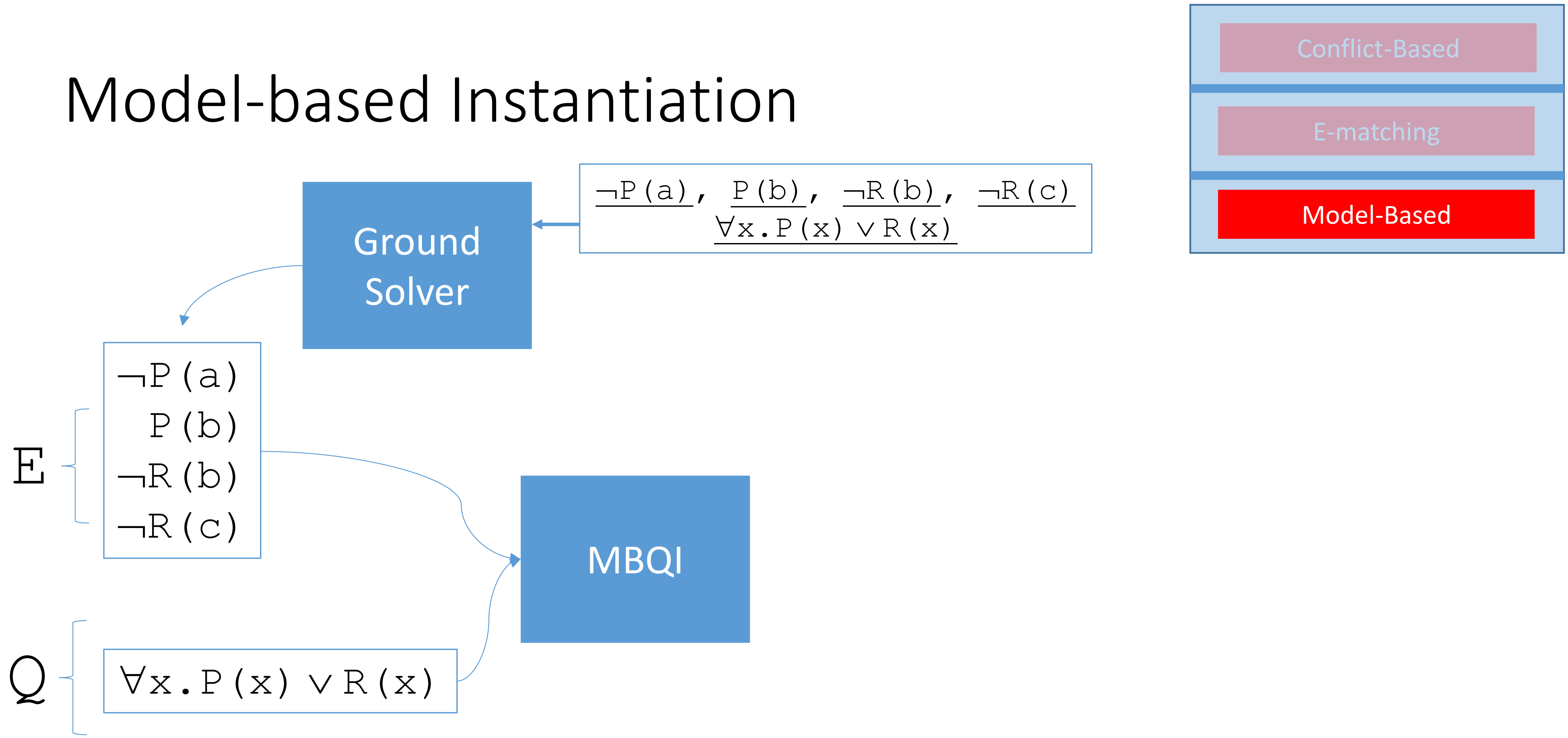
Model-based Instantiation



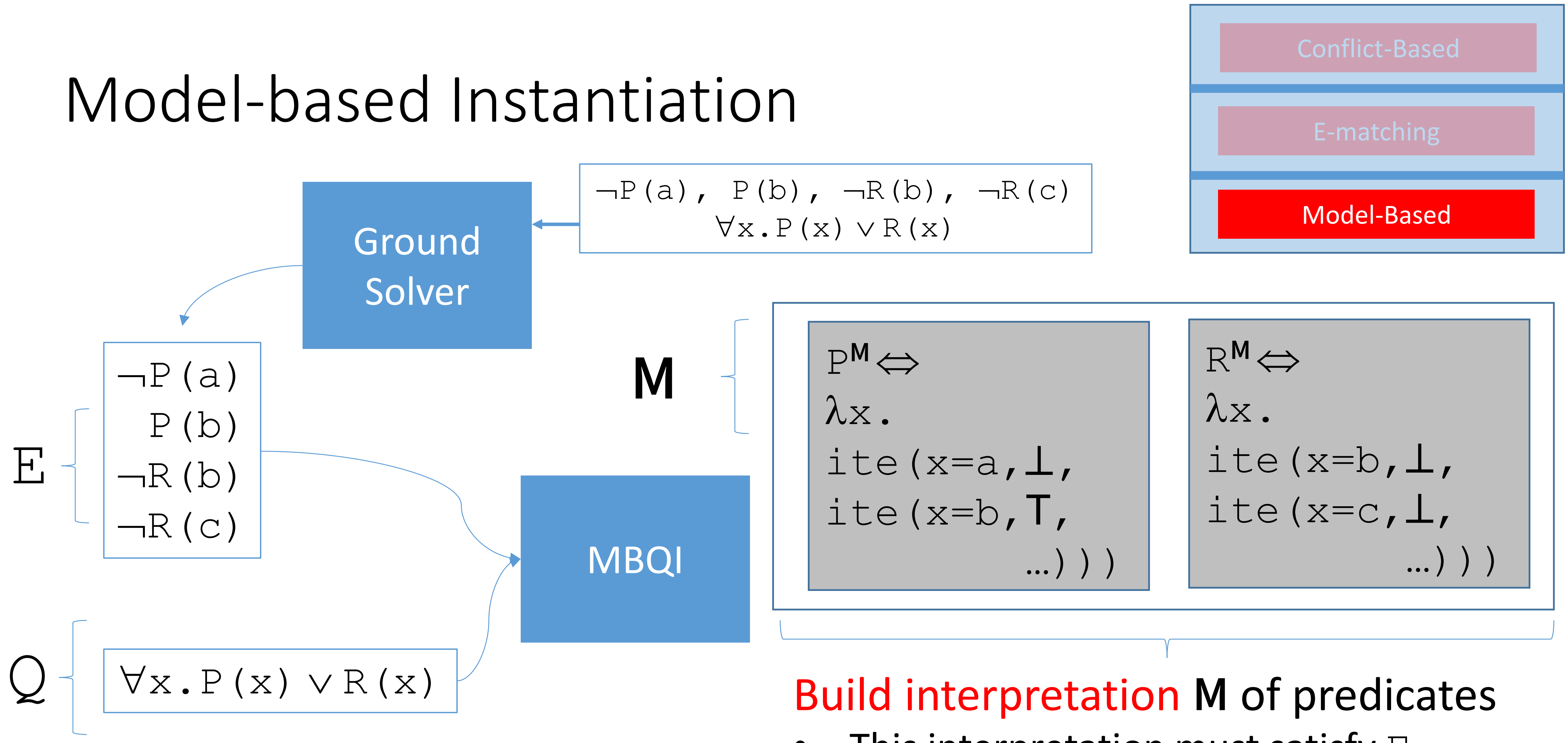
- Implemented in solvers:
  - Z3 **[Ge et al CAV09]**, CVC4 **[Reynolds et al CADE13]**
- Basic idea:
  1. Build interpretation **M** for all uninterpreted functions in the signature
     - e.g. $\mathtt{P}^{\mathbf{M}} \Leftrightarrow \lambda \mathtt{x.ite(x>0,}\top\mathtt{,}\bot\mathtt{)}$
  2. If this interpretation satisfies all formulas in $\mathtt{Q}$, answer "sat"
     - e.g. interpretation **M** satisfies $\forall \mathtt{x.x>4} \Rightarrow \mathtt{P(x)}$

$\Rightarrow$ *Ability to answer "sat"*

Model-based Instantiation



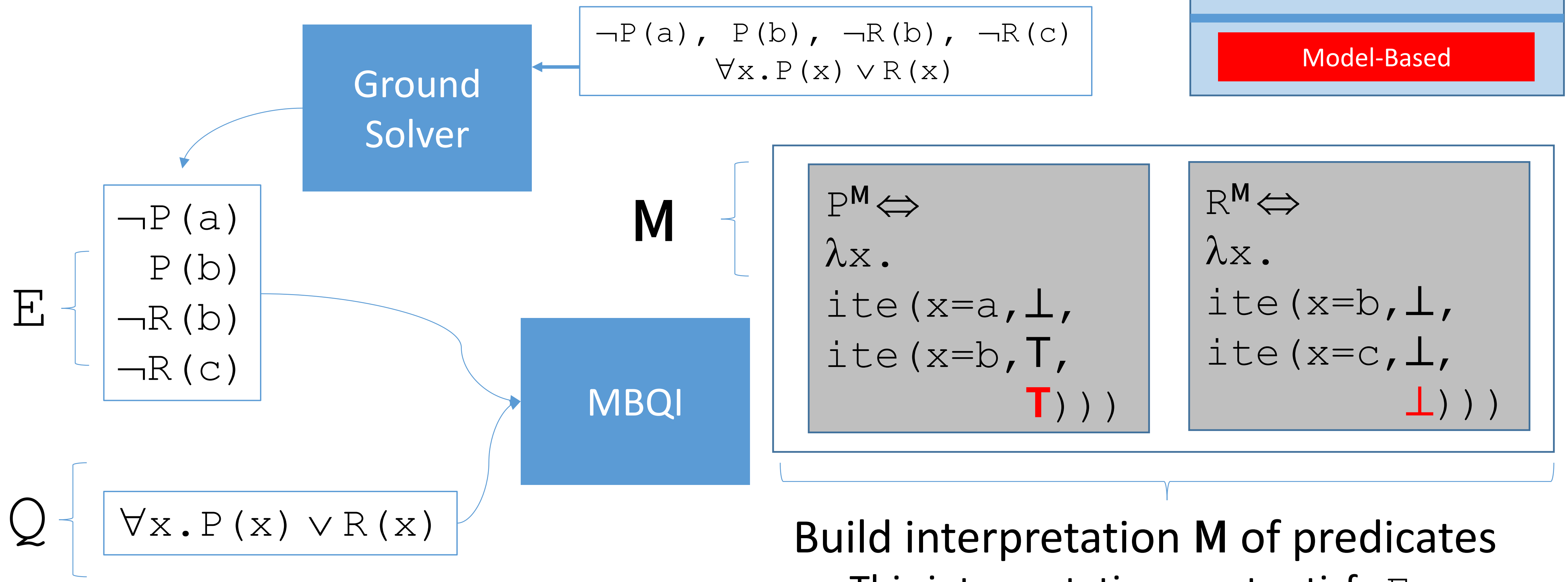
Model-based Instantiation



Build interpretation **M** of predicates

- This interpretation must satisfy E

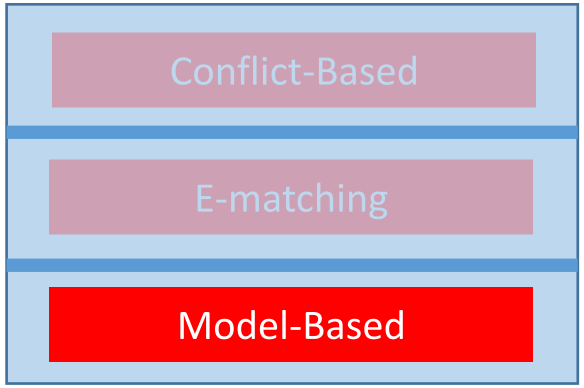
Model-based Instantiation

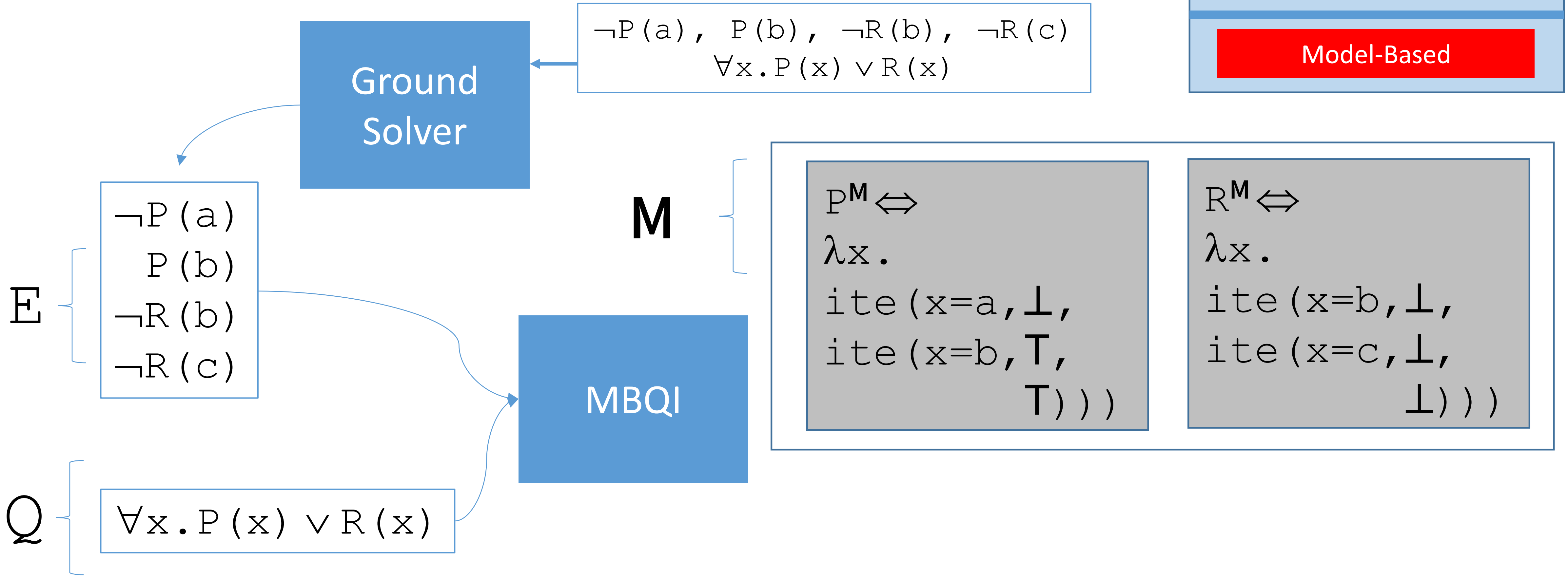


Build interpretation **M** of predicates

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

Model-based Instantiation

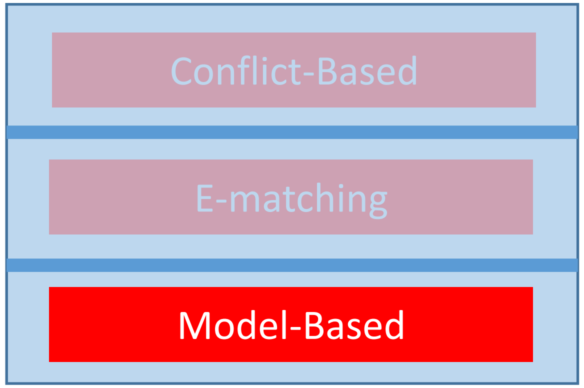


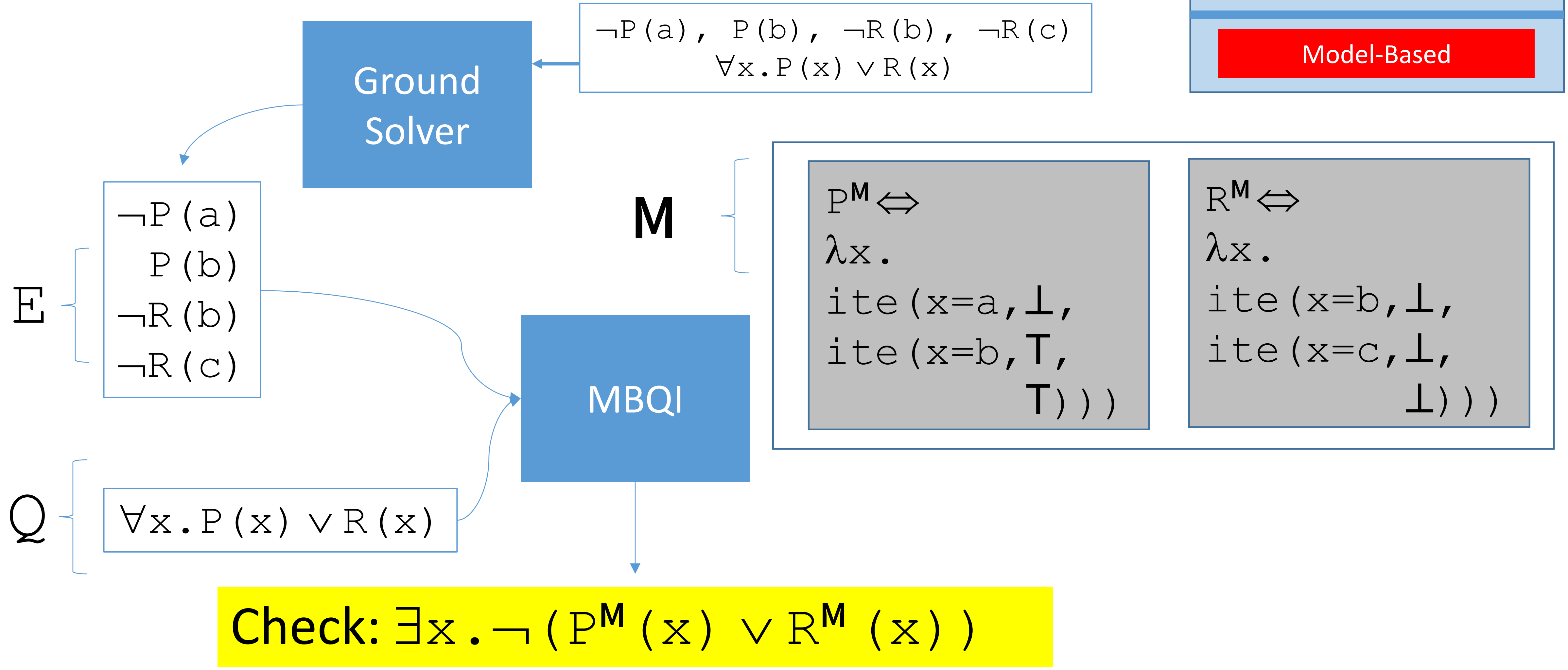


$\Rightarrow$ Does **M** satisfy Q?

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

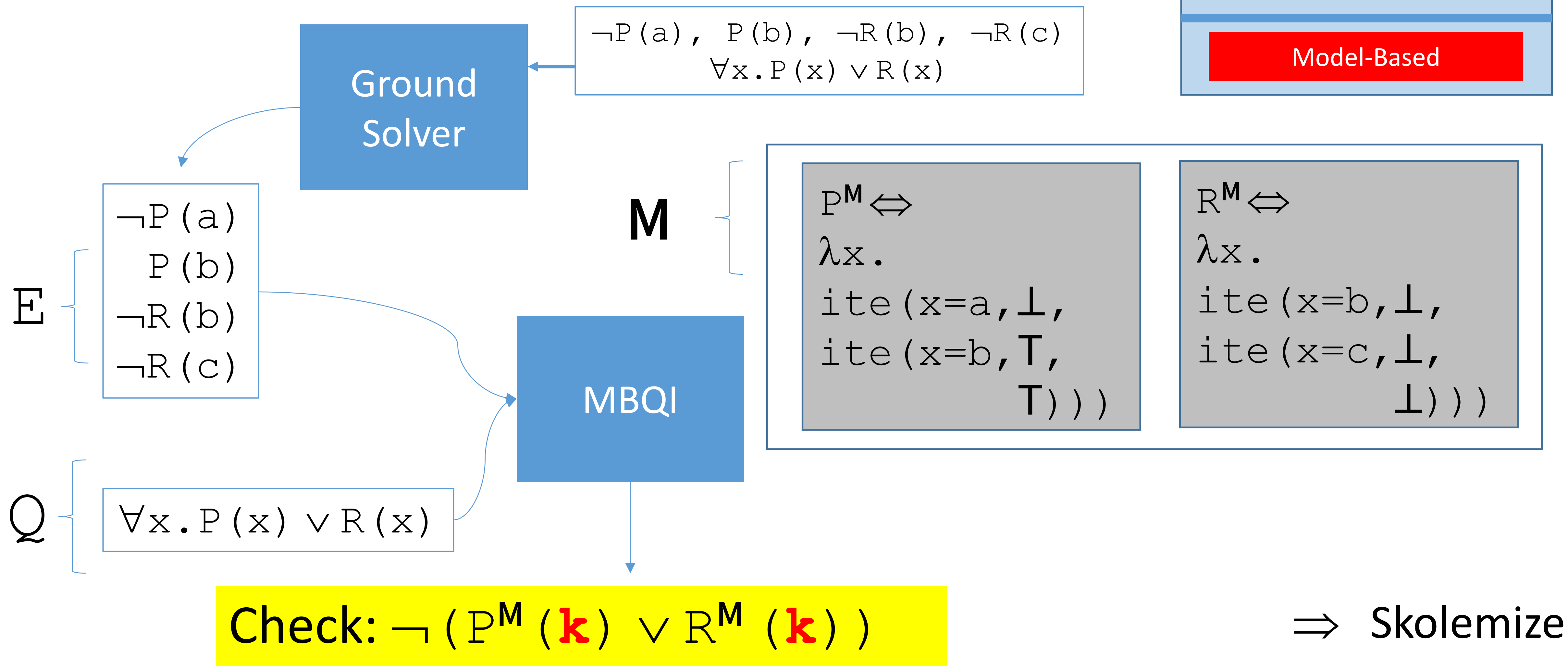
Model-based Instantiation



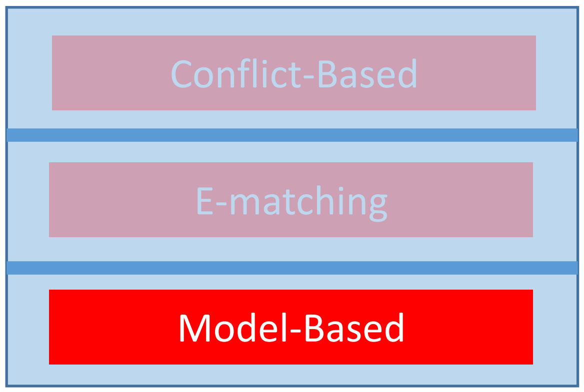


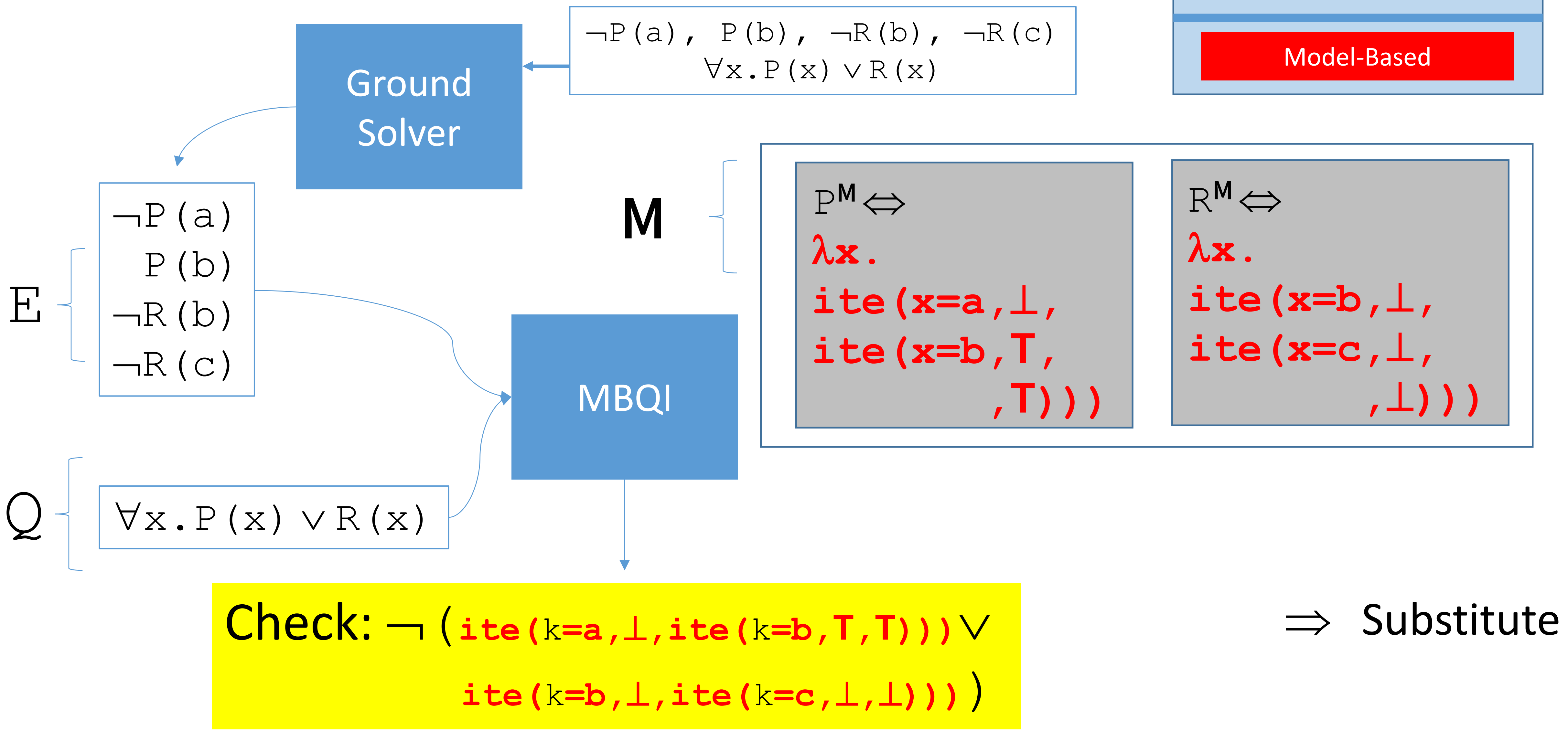
Model-based Instantiation

Conflict-Based
E-matching
Model-Based



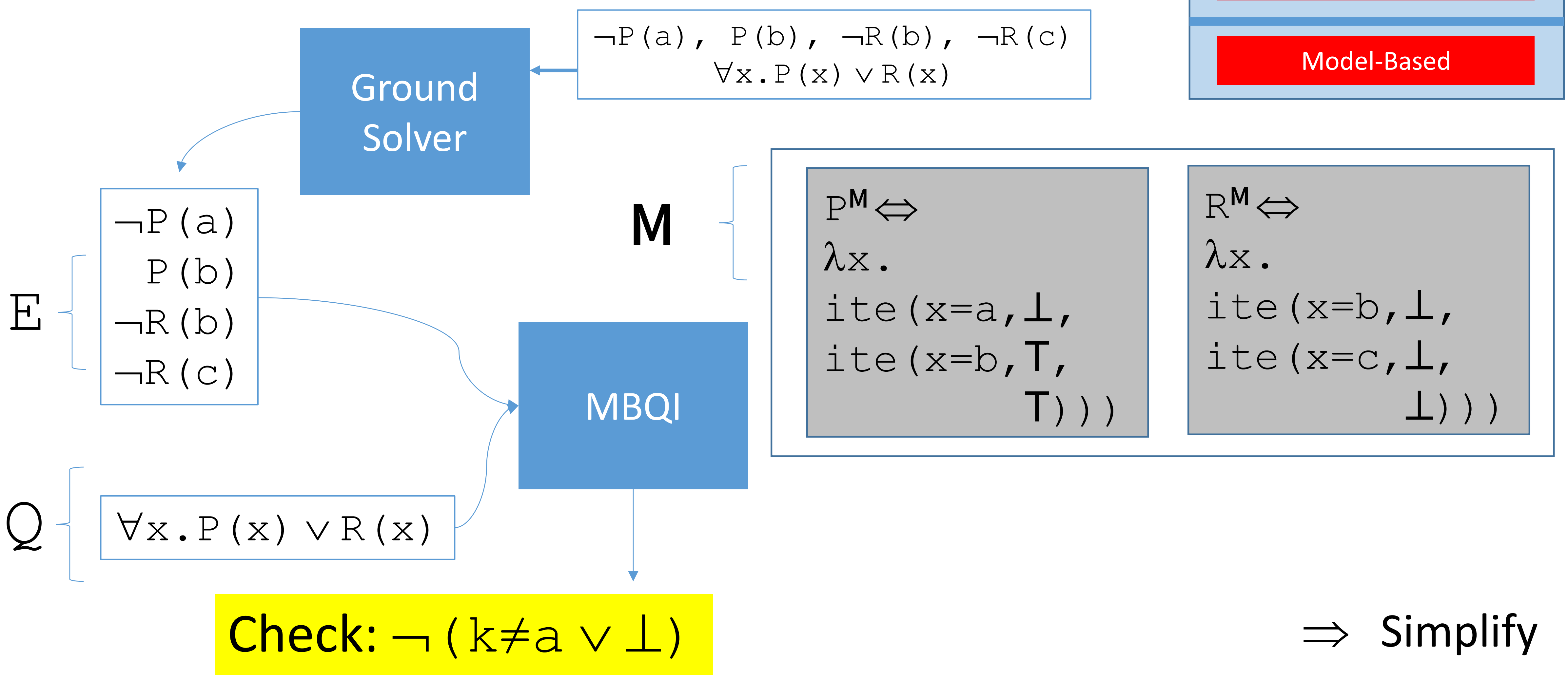
Model-based Instantiation



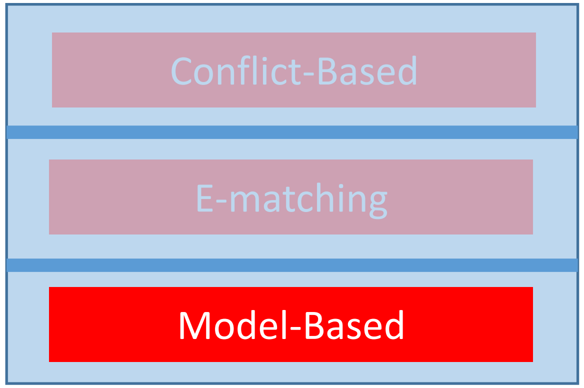


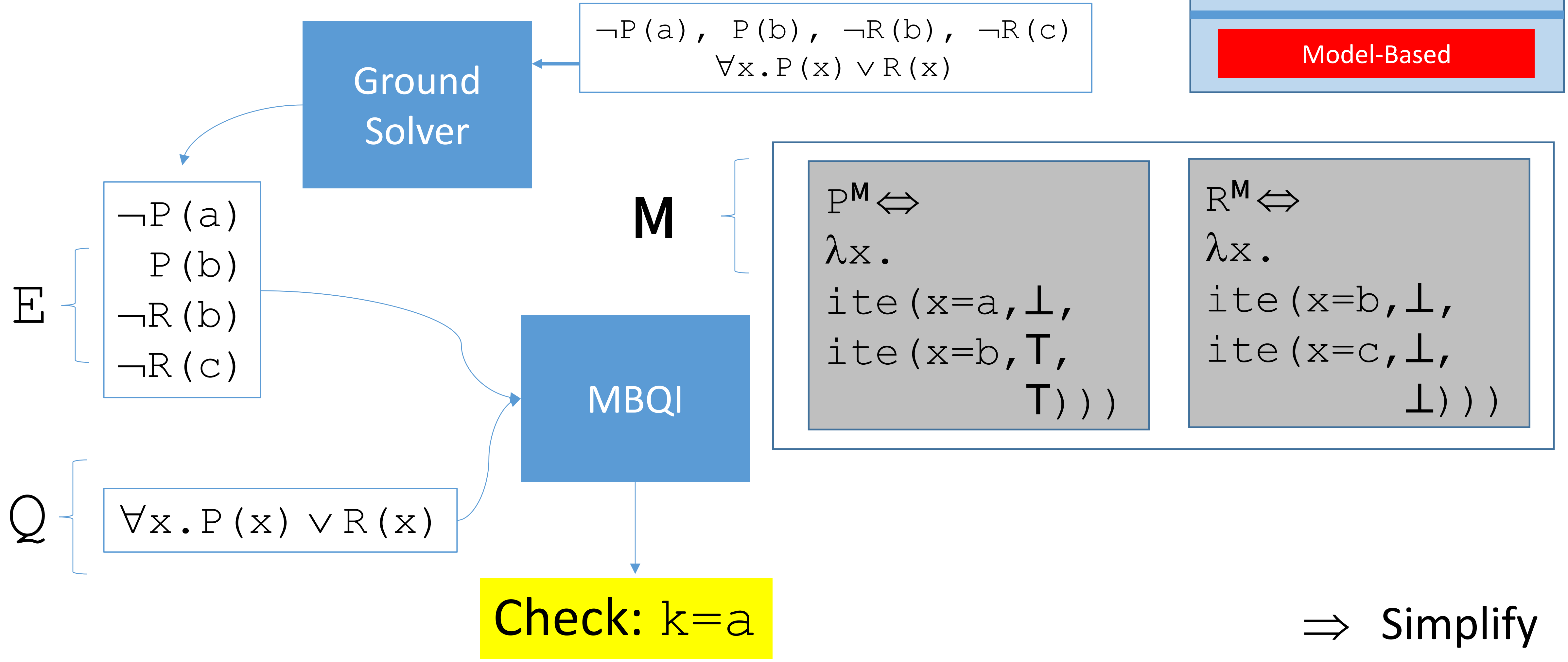
Model-based Instantiation

- Conflict-Based
- E-matching
- **Model-Based**



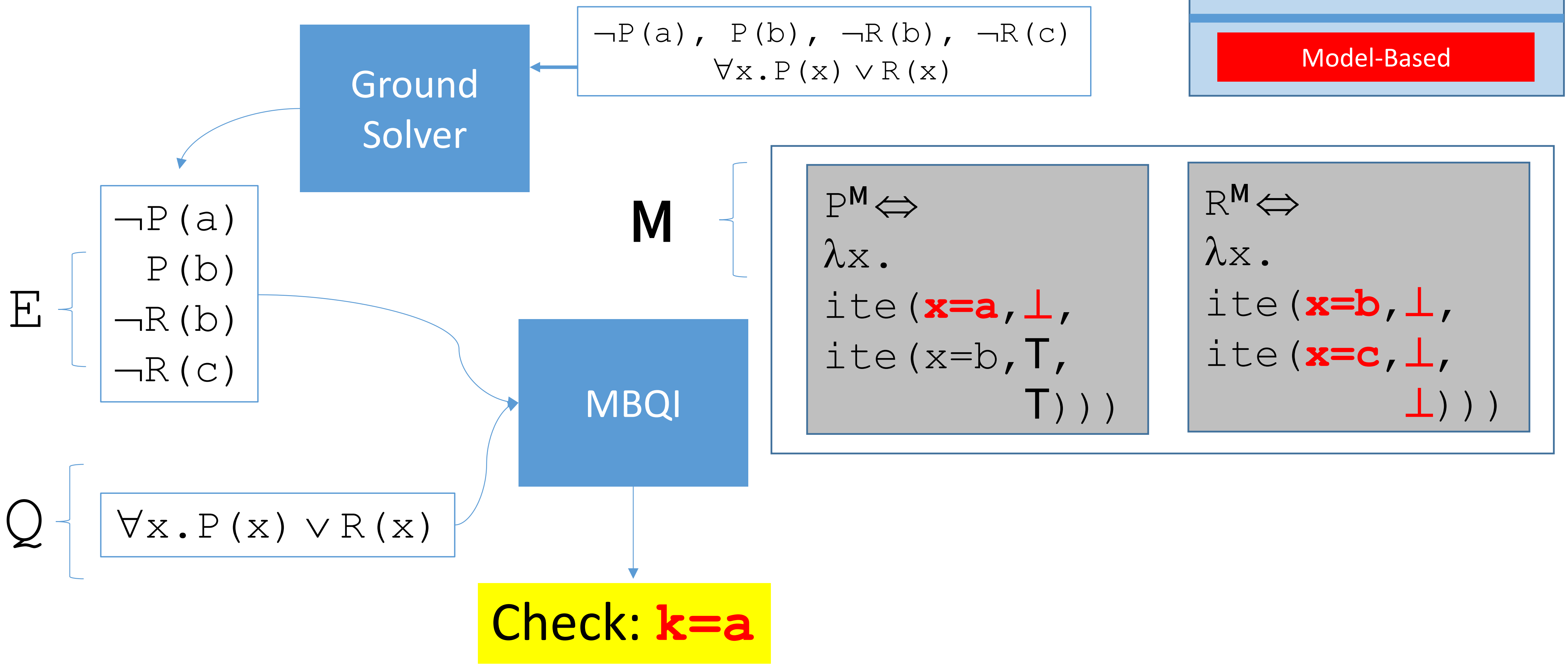
Model-based Instantiation





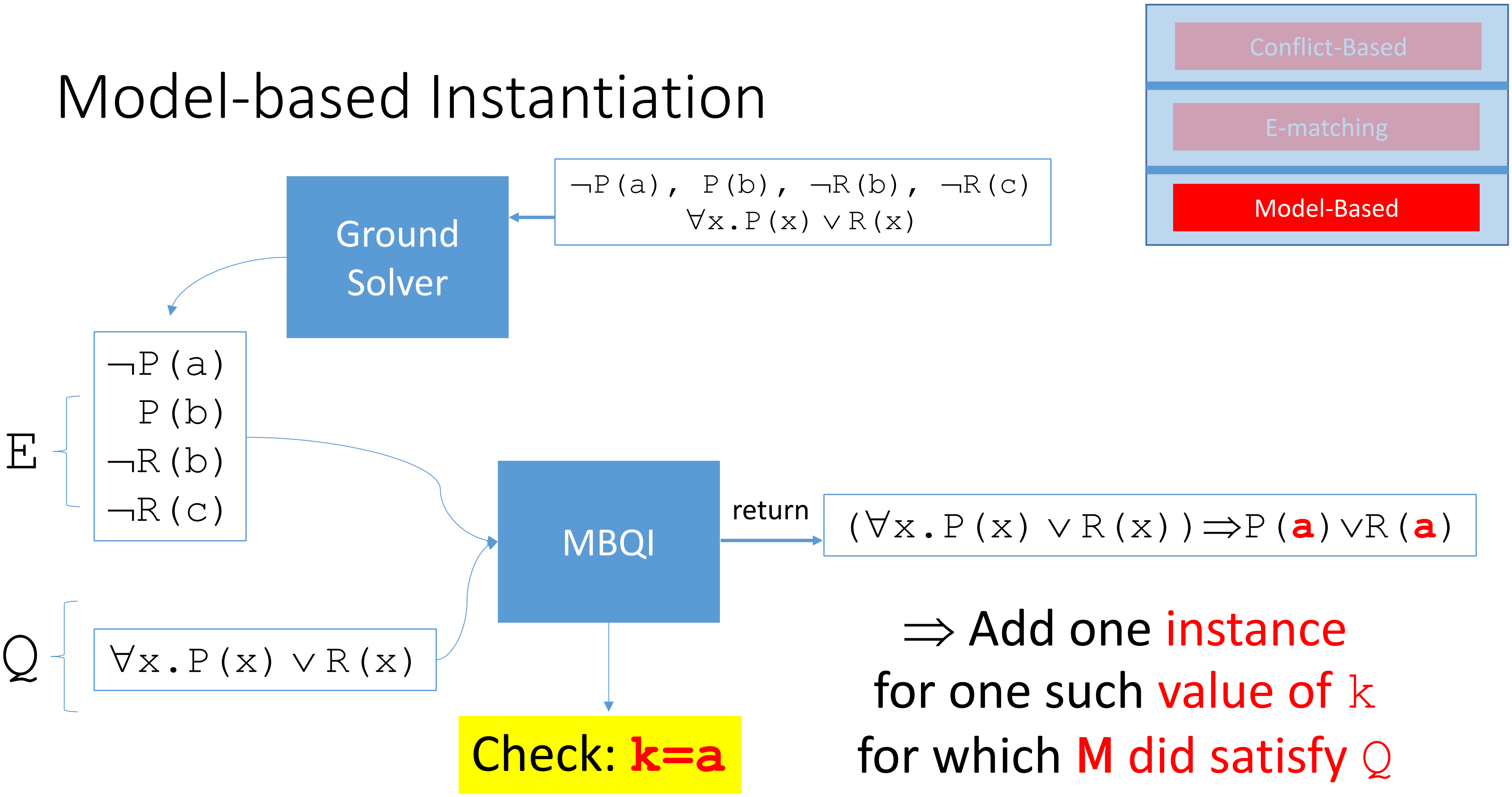
Model-based Instantiation

Conflict-Based
E-matching
Model-Based



⇒ *Satisfiable! There are values k for which M does not satisfy Q*

Model-based Instantiation



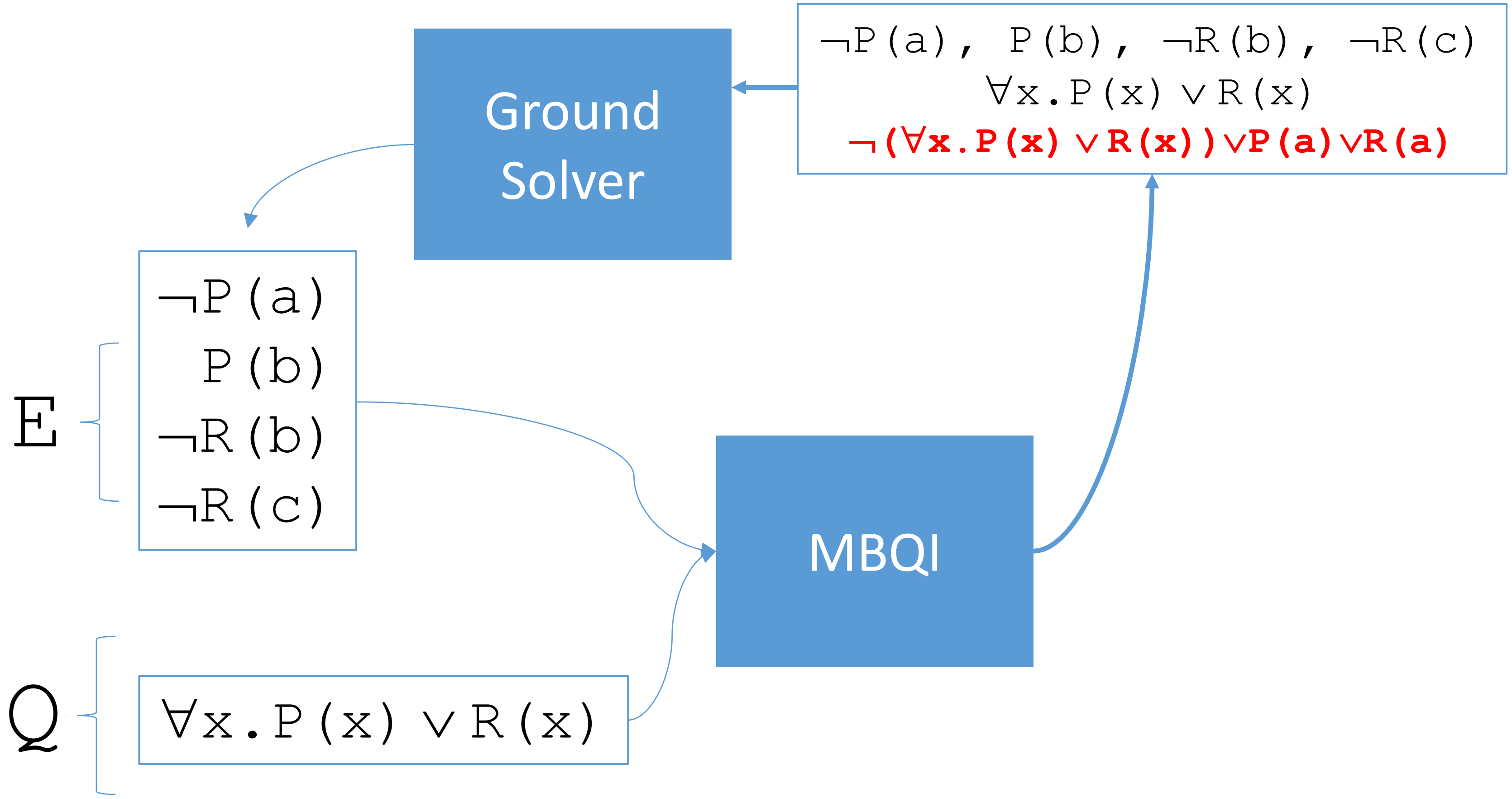
⇒ Add one instance for one such value of k for which M did satisfy Q

Conflict-Based

E-matching

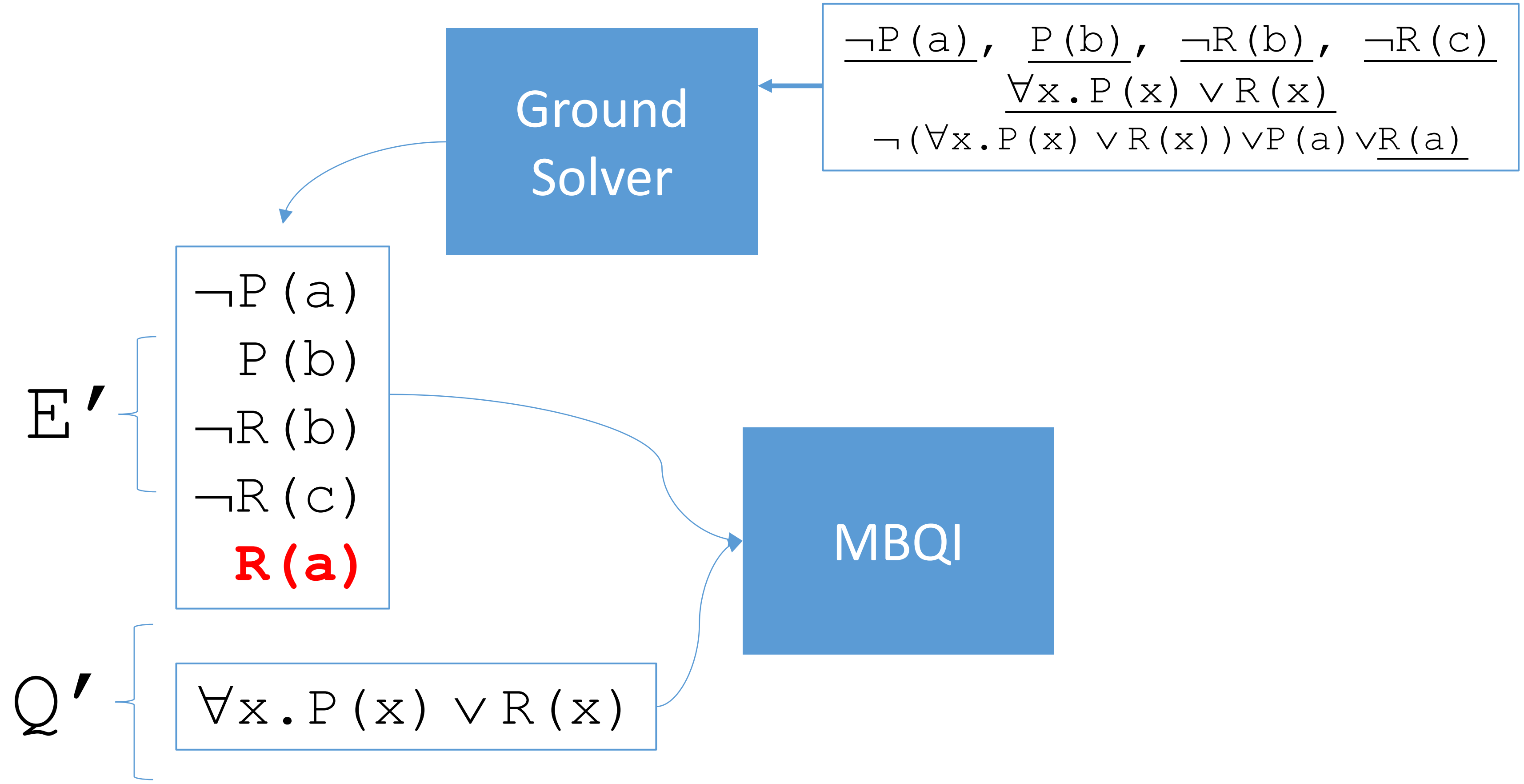
Model-Based

Model-based Instantiation



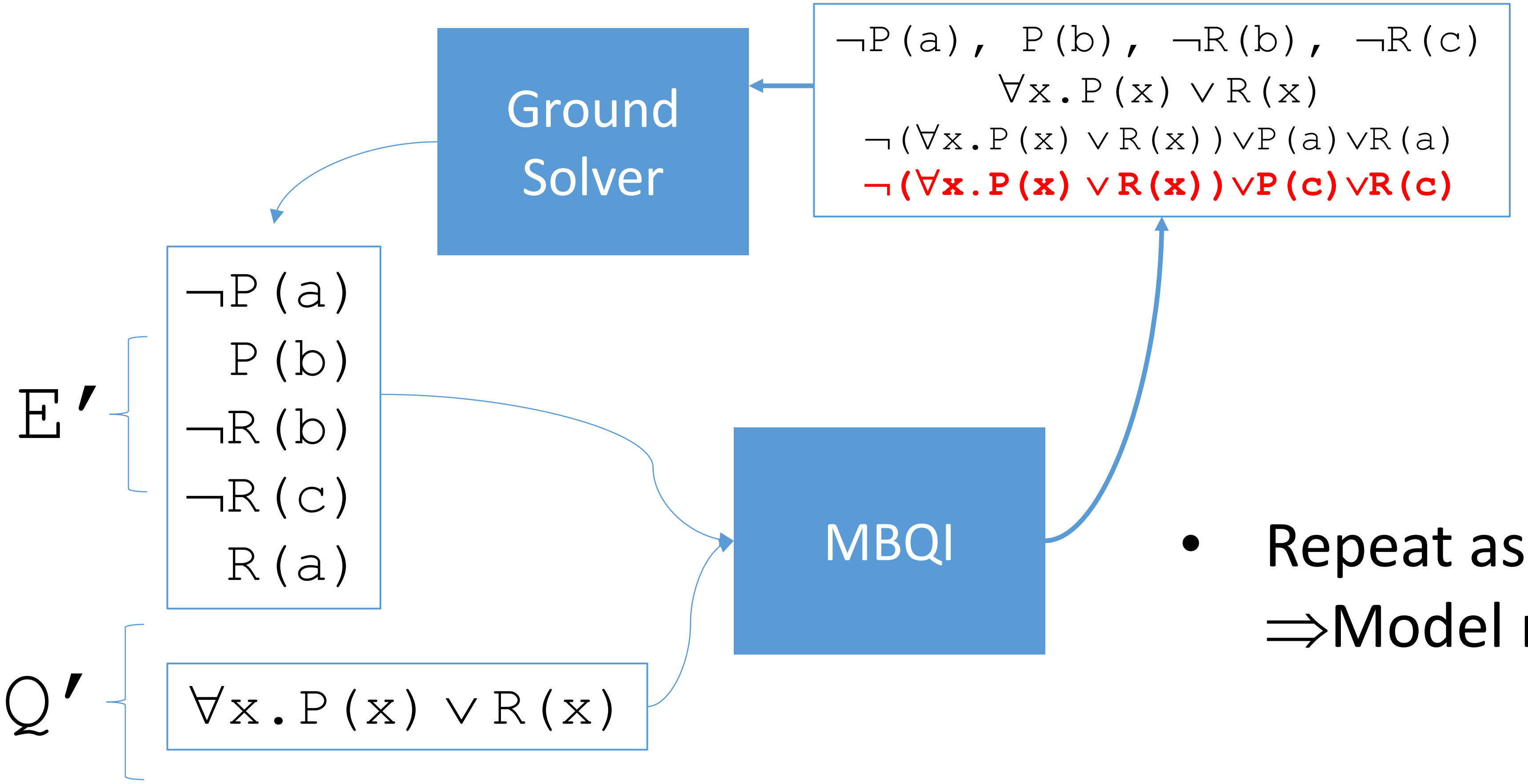
Model-based Instantiation

Conflict-Based
E-matching
Model-Based



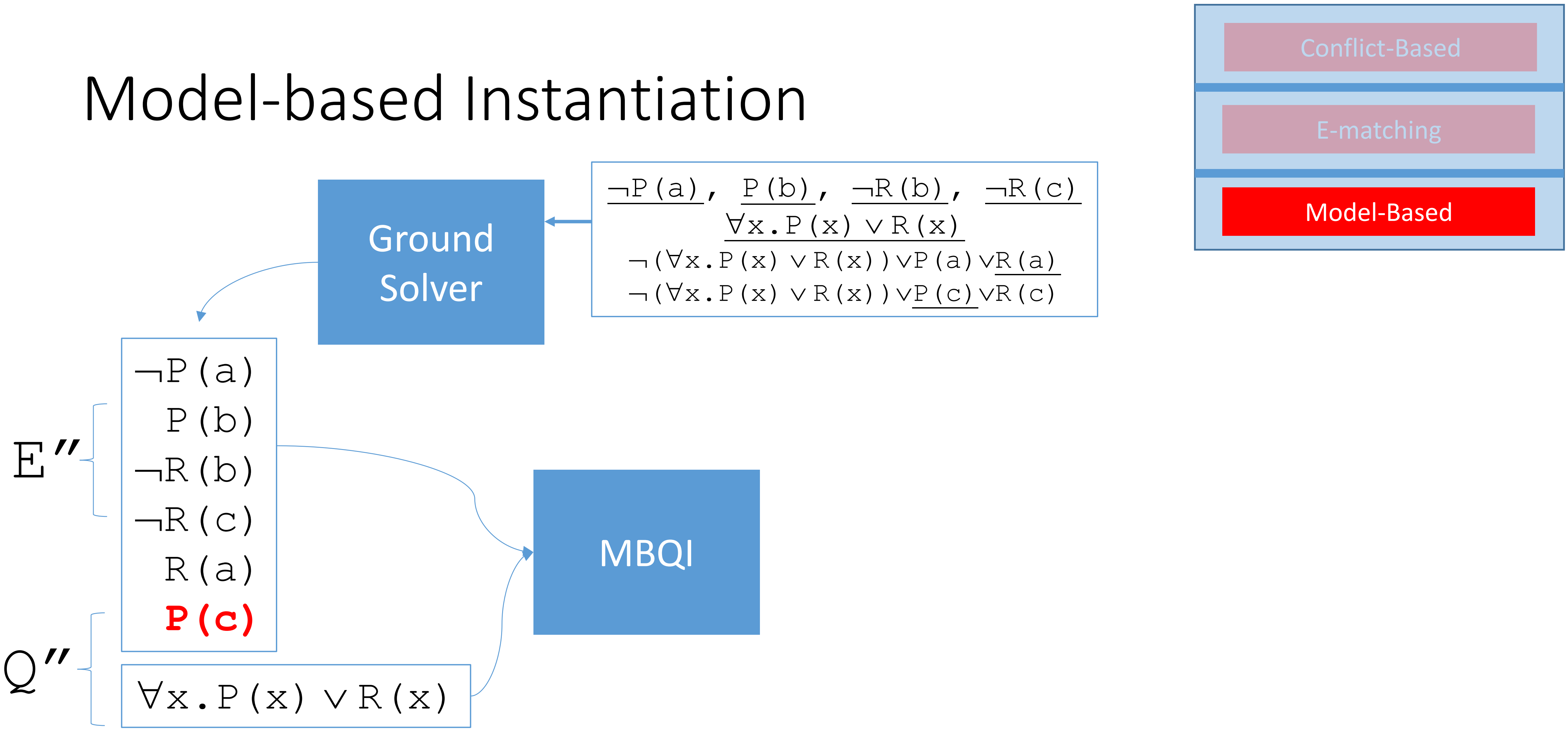
$\Rightarrow$ Subsequent models must satisfy $P(x) \lor R(x)$ for $x \rightarrow a$

Model-based Instantiation

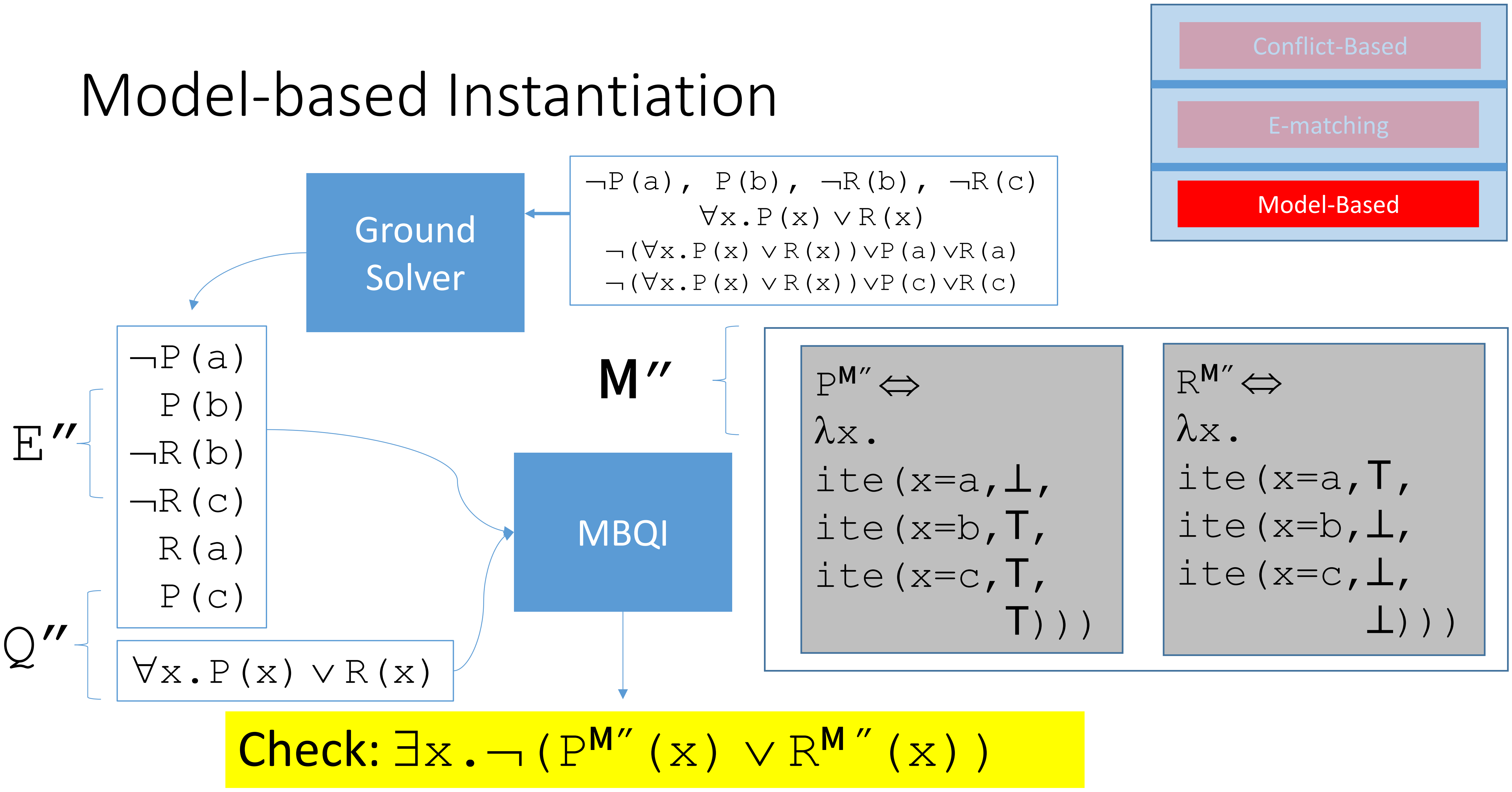


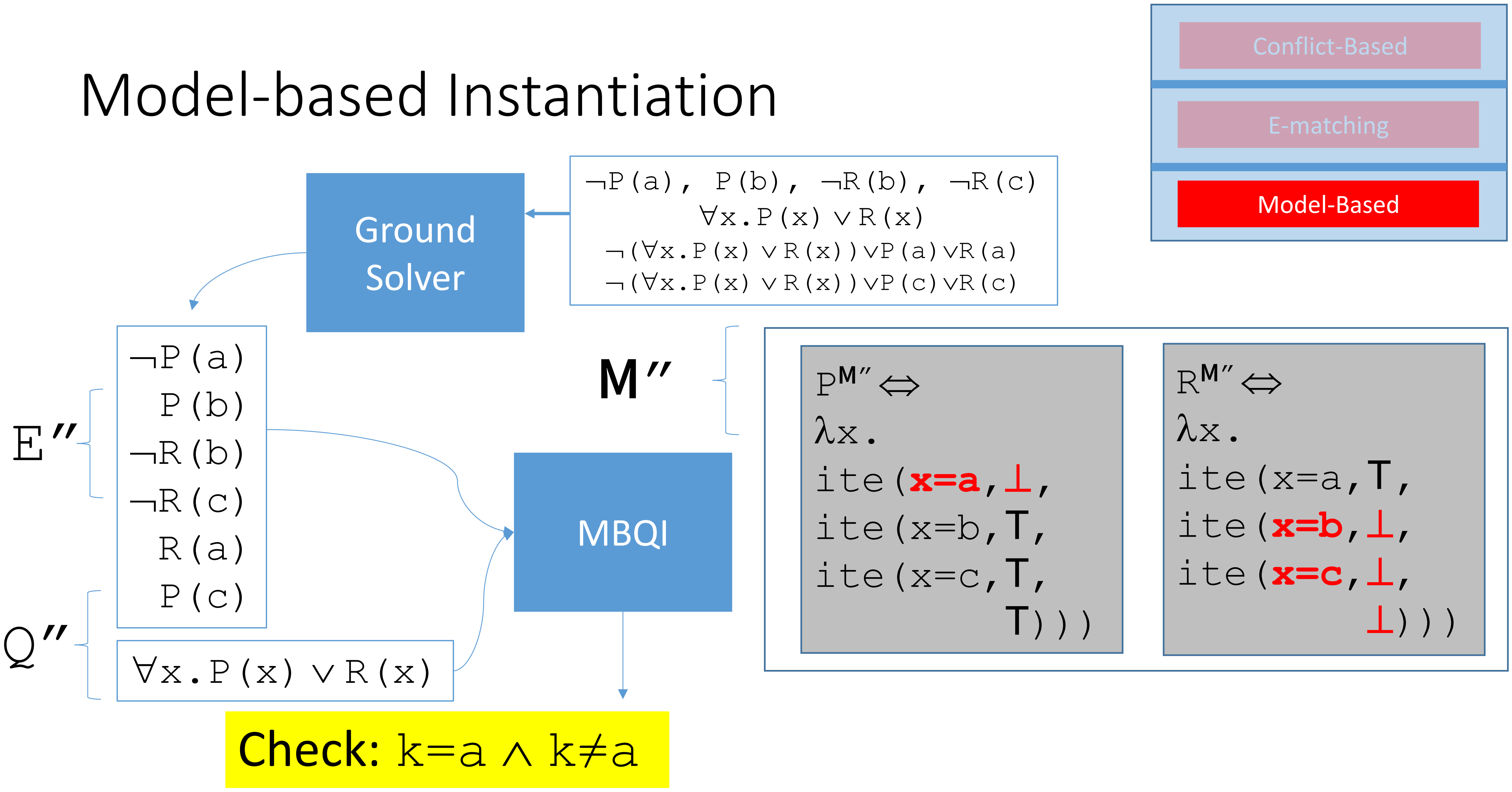
- Repeat as necessary
  ⇒Model refinement loop

Model-based Instantiation



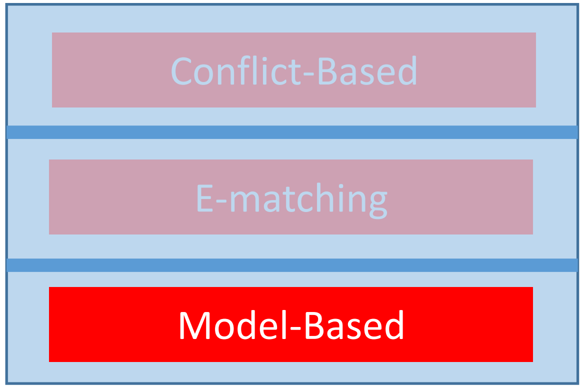
Model-based Instantiation

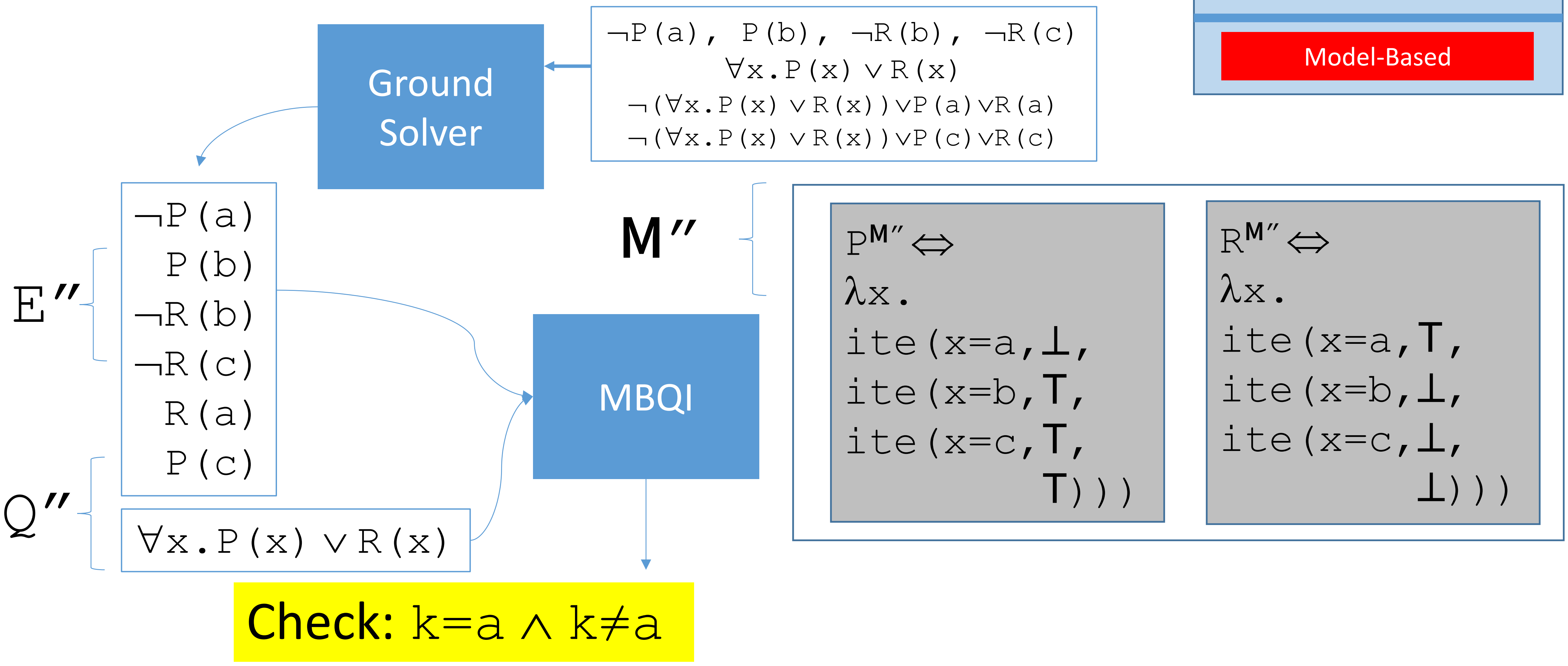




Model-based Instantiation
Conflict-Based
E-matching
Model-Based
¬P(a), P(b), ¬R(b), ¬R(c)
∀x.P(x) ∨ R(x)
¬(∀x.P(x) ∨ R(x))∨P(a)∨R(a)
¬(∀x.P(x) ∨ R(x))∨P(c)∨R(c)
Ground Solver
E″
¬P(a)
P(b)
¬R(b)
¬R(c)
R(a)
P(c)
Q″
∀x.P(x) ∨ R(x)
MBQI
M″
P^M″ ⇔ λx. ite(x=a,⊥, ite(x=b,T, ite(x=c,T, T)))
R^M″ ⇔ λx. ite(x=a,T, ite(x=b,⊥, ite(x=c,⊥, ⊥)))
Check: k=a ∧ k≠a

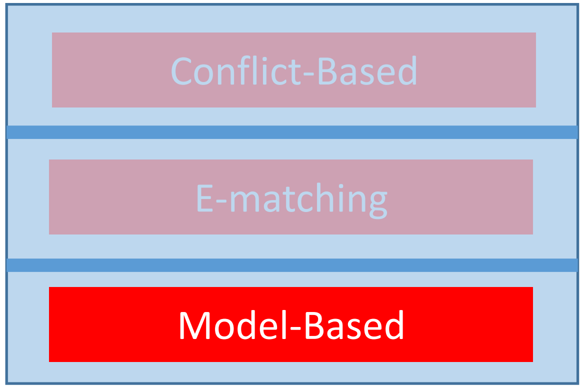
Model-based Instantiation

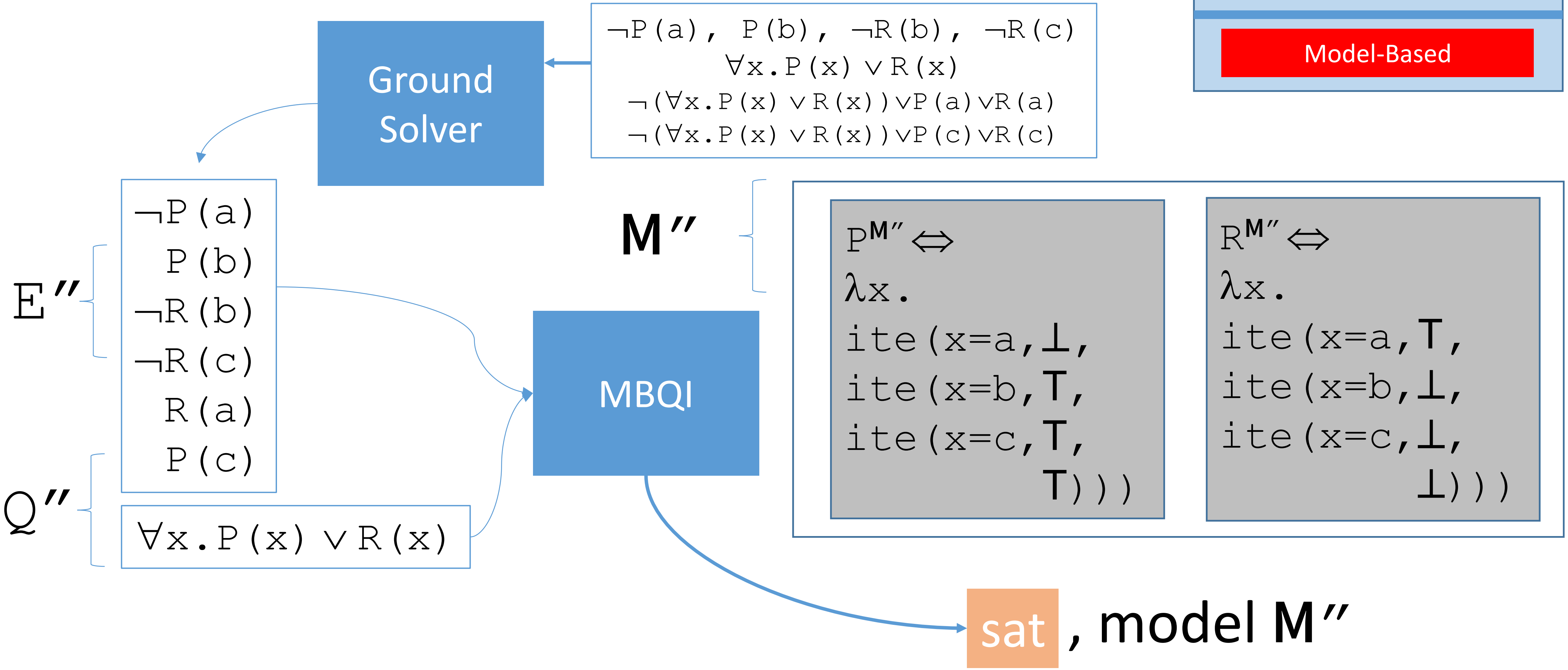




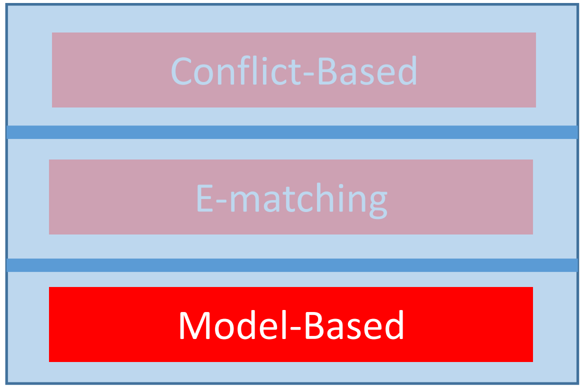
⟹ Unsatisfiable, there are no values k for which **M** ″ does not satisfy Q

Model-based Instantiation



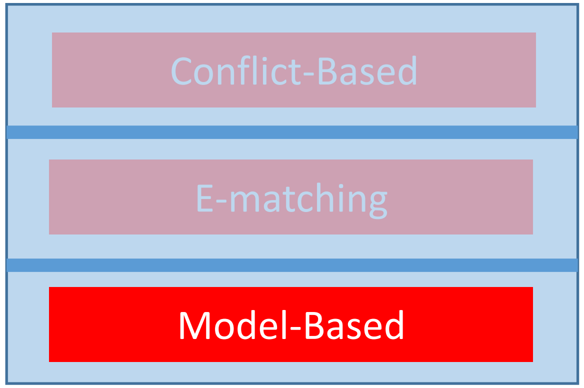


Model-based Instantiation: Completeness



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

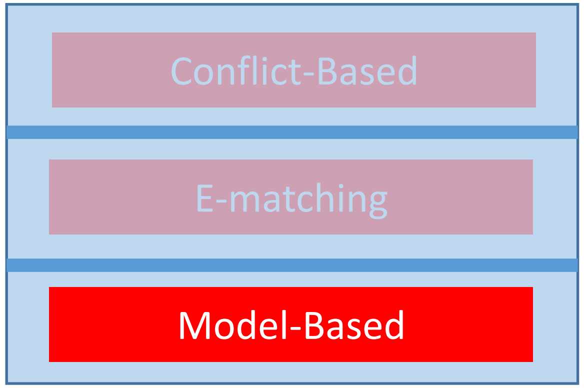
Model-based Instantiation: Completeness



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

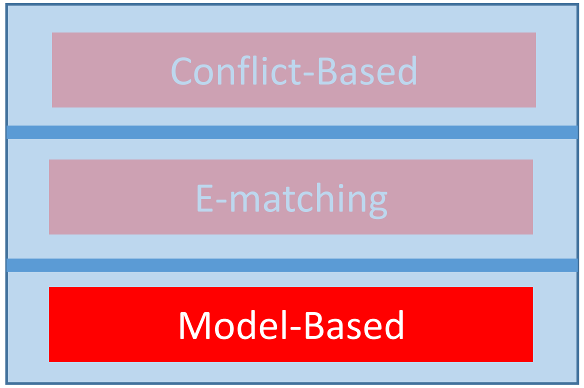
- Under what conditions are these techniques *terminating?*
  A. If the domains of ∀ are interpreted as finite
     - E.g. quantified bitvectors **[Wintersteiger et al 13]**

Model-based Instantiation: 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]**

Model-based Instantiation: 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

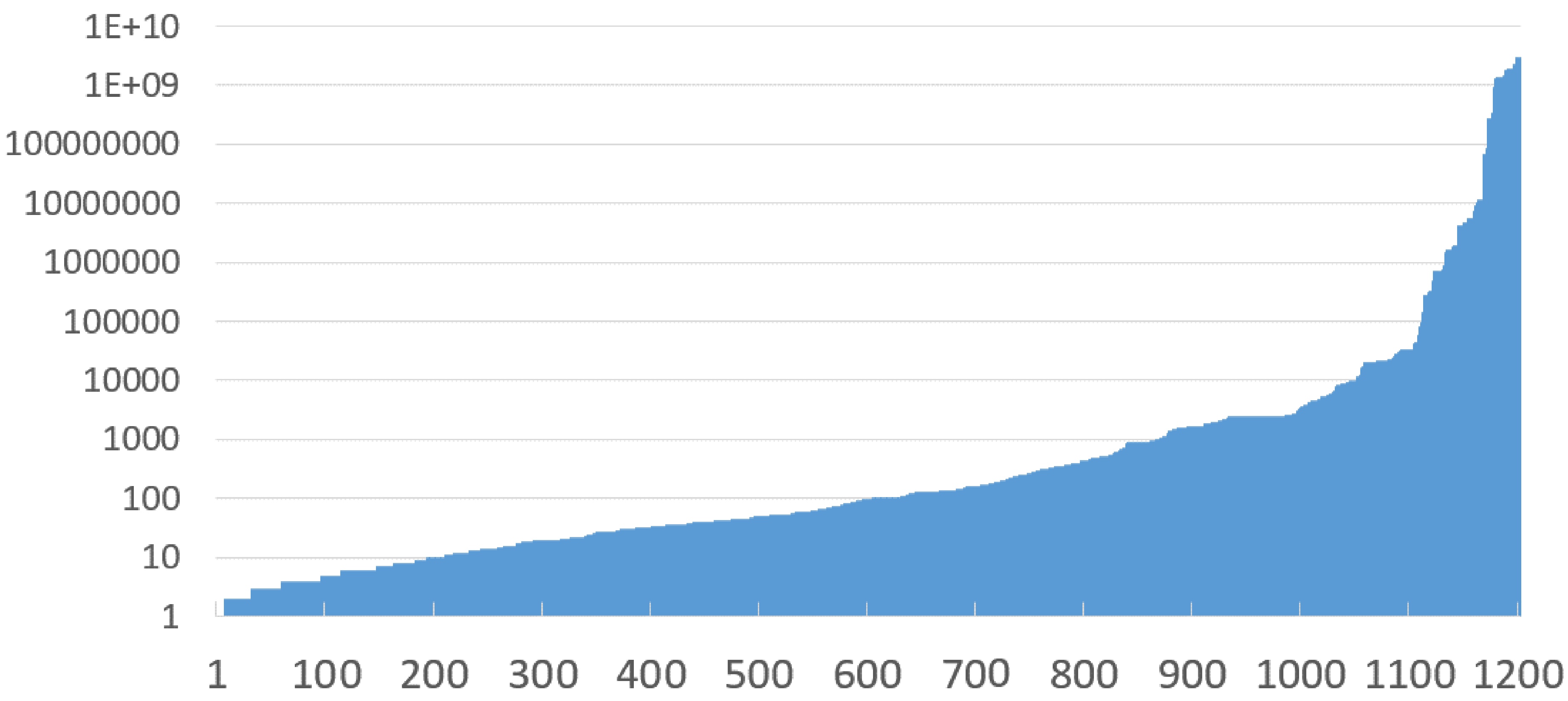
     …but it can be argued that only finitely many instances will be generated
     - E.g. essentially uninterpreted fragment **[Ge+deMoura 09]**, …

Conflict-Based

E-matching

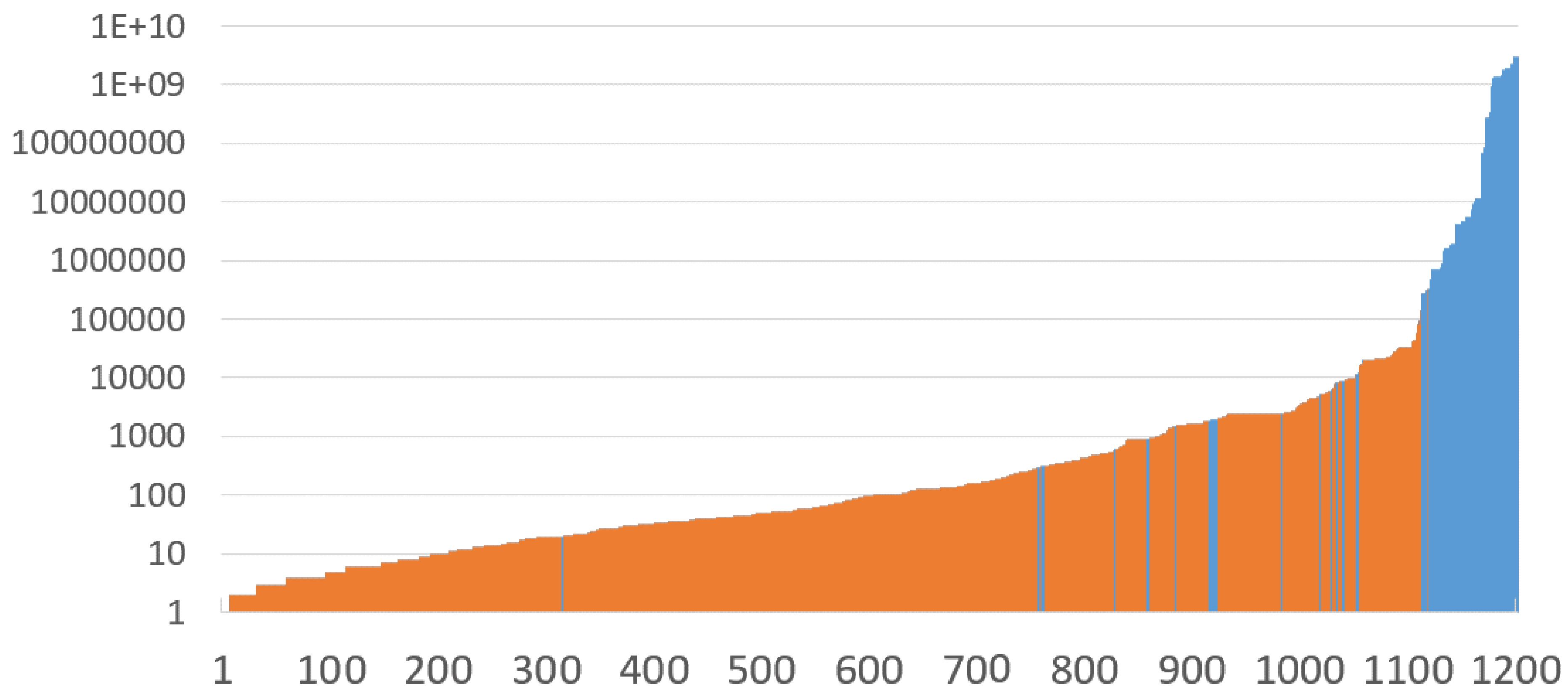
Model-Based

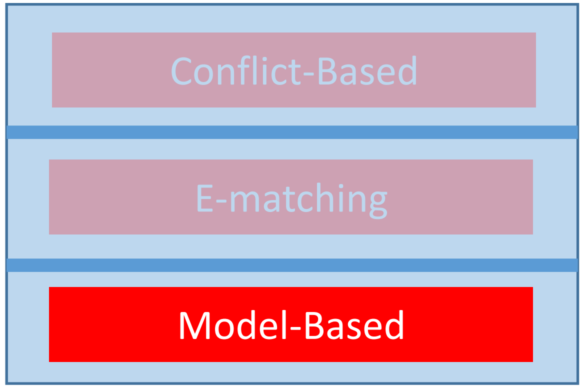
Model-based Instantiation: Impact



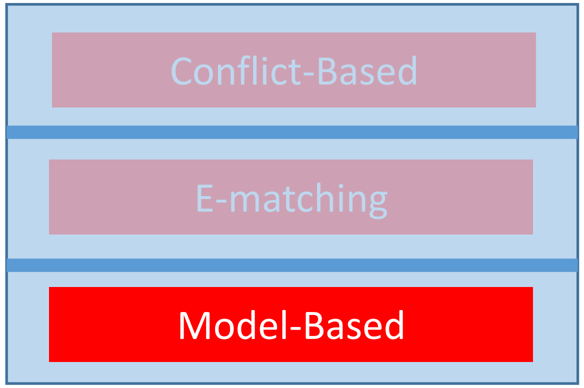
- 1203 satisfiable benchmarks from the TPTP library
  - Graph shows # instances required by exhaustive instantiation
    - E.g. $\forall xyz : U . P(x,y,z)$, if $|U|=4$, requires $4^3=64$ instances

Model-based Instantiation: Impact

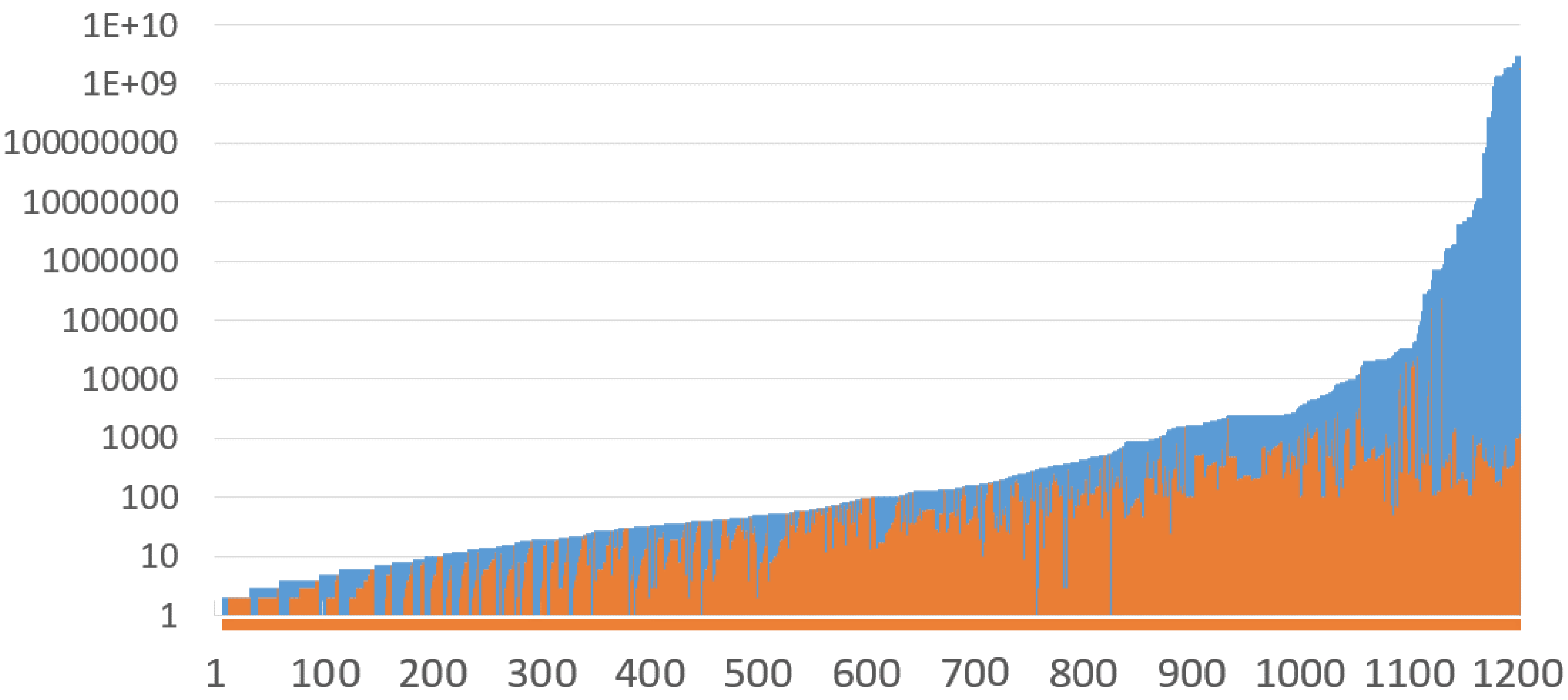




- CVC4 Finite Model Finding + Exhaustive instantiation
  - Scales only up to ~150k instances with a 30 sec timeout

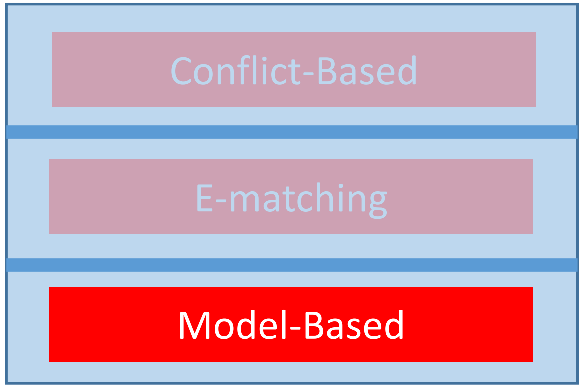


Model-based Instantiation: Impact

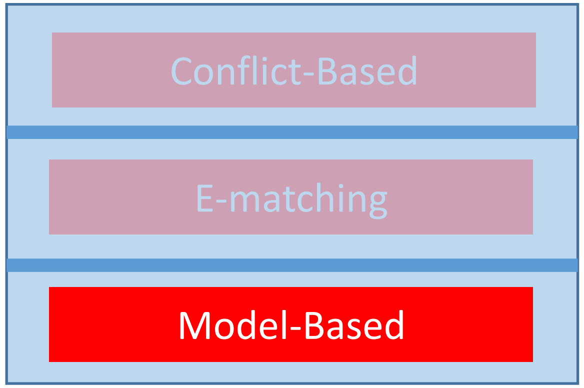


- CVC4 Finite Model Finding + Model-Based instantiation **[Reynolds et al CADE13]**
  - Scales to >2 billion instances with a 30 sec timeout, only adds fraction of possible instances

Model-based Instantiation: Challenges



Model-based Instantiation: Challenges



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

Model-based Instantiation: Challenges

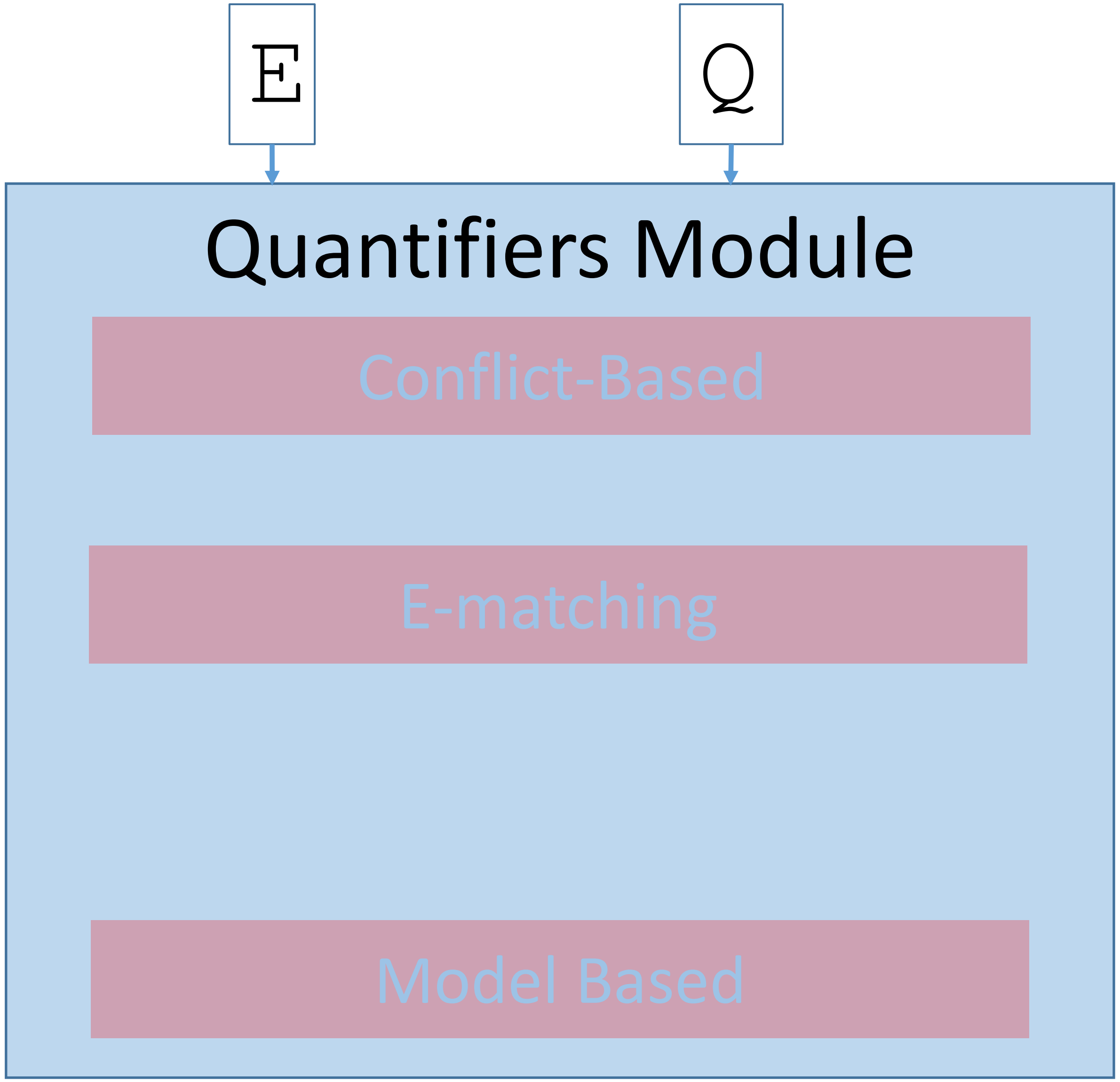
Conflict-Based
E-matching
Model-Based

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

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

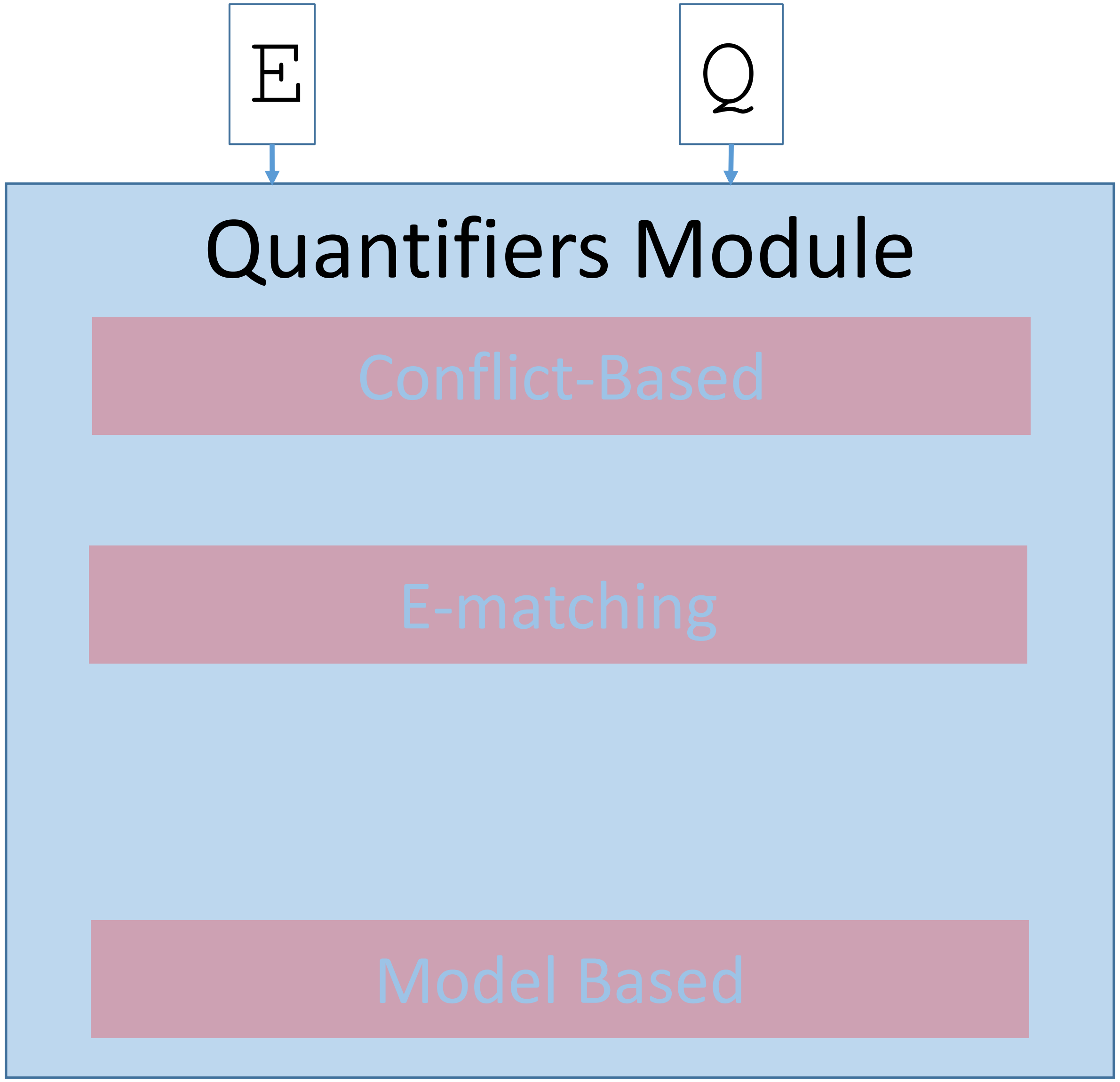
$\forall xy{:}Int.(f(x,y) \geq x \land f(x,y) \geq y)$	→	$f^M := \lambda xy.\mathtt{ite}(x \geq y, x, y)$
$\forall x{:}Int.3*g(x)+5*h(x)=x$	→	$g^M := \lambda x.5*x$ $h^M := \lambda x.-3*x$
$\forall xy{:}Int.u(x+y)+11*v(w(x))=x+y$	→	???

Putting it Together

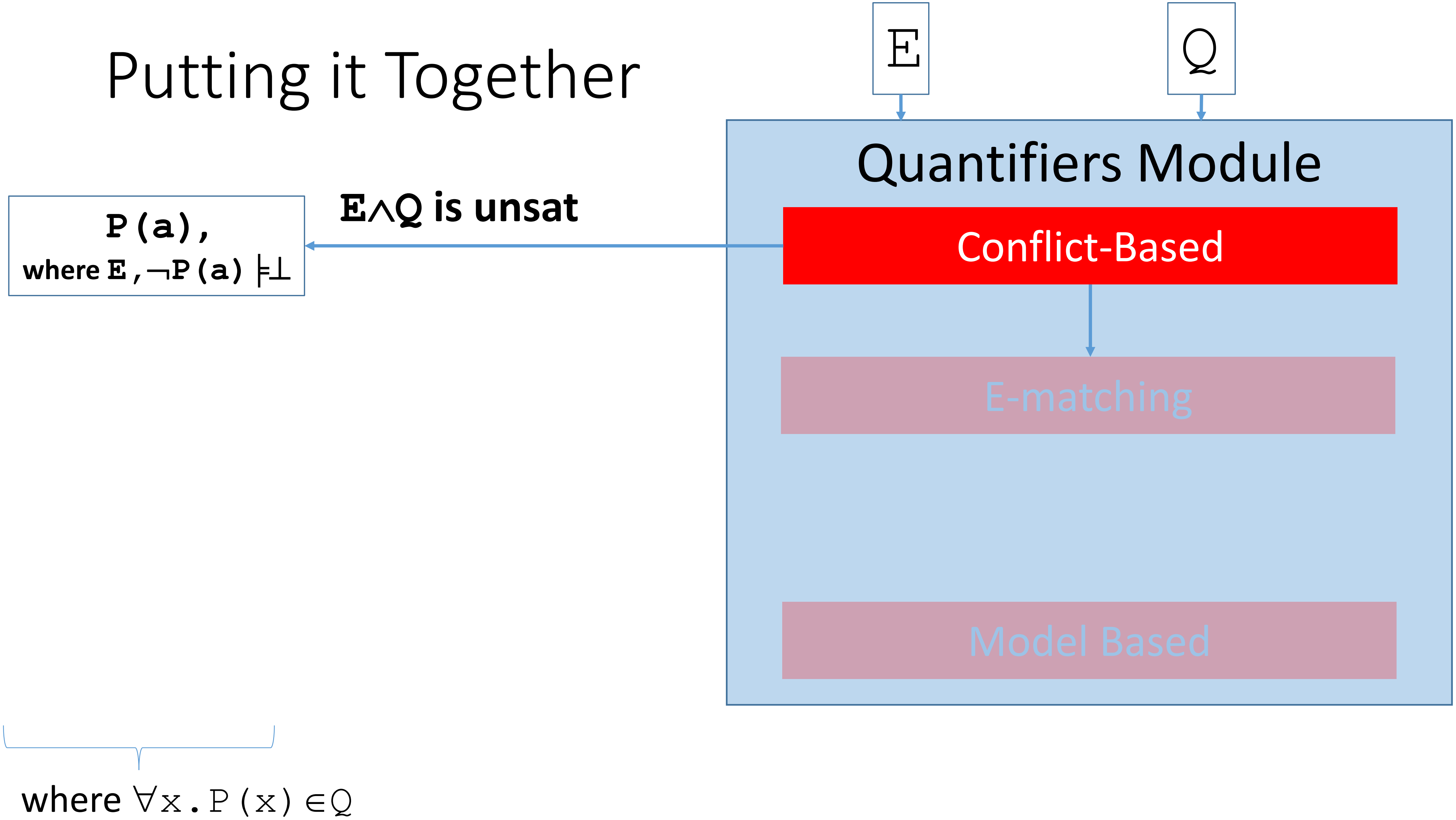


Putting it Together

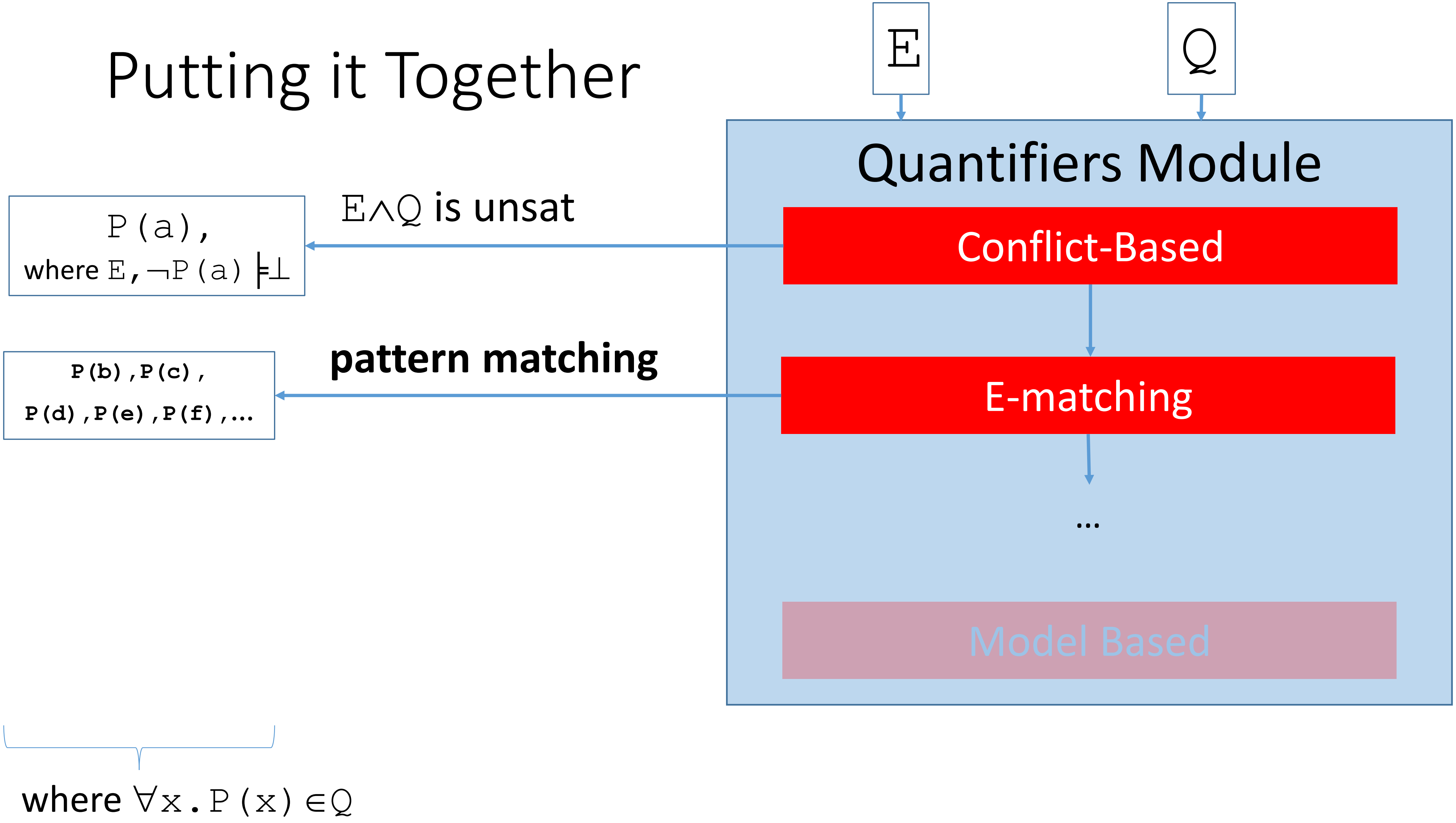
- Input:
  - Ground literals $\mathtt{E}$
  - Quantified formulas $\mathtt{Q}$



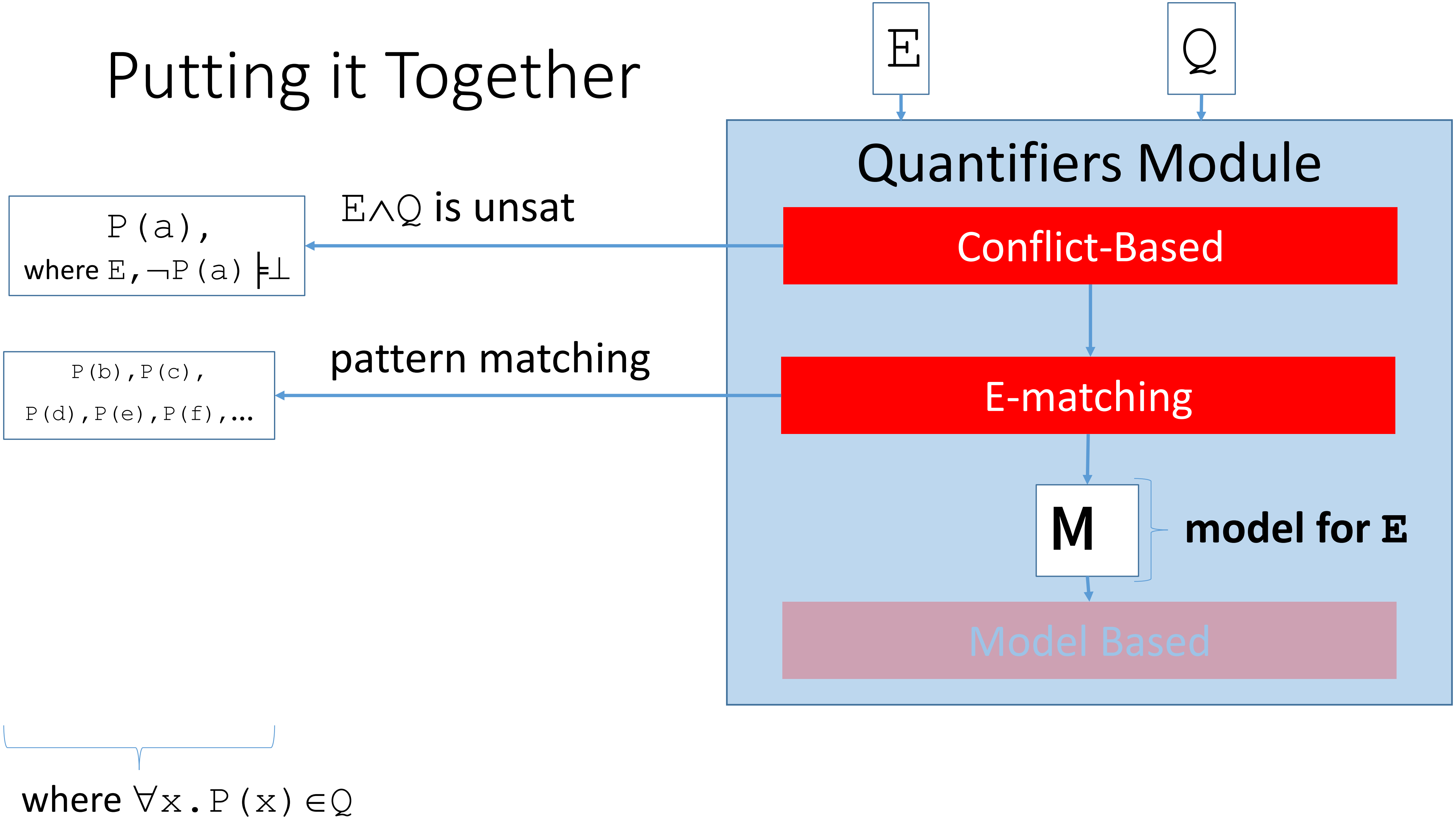
Putting it Together



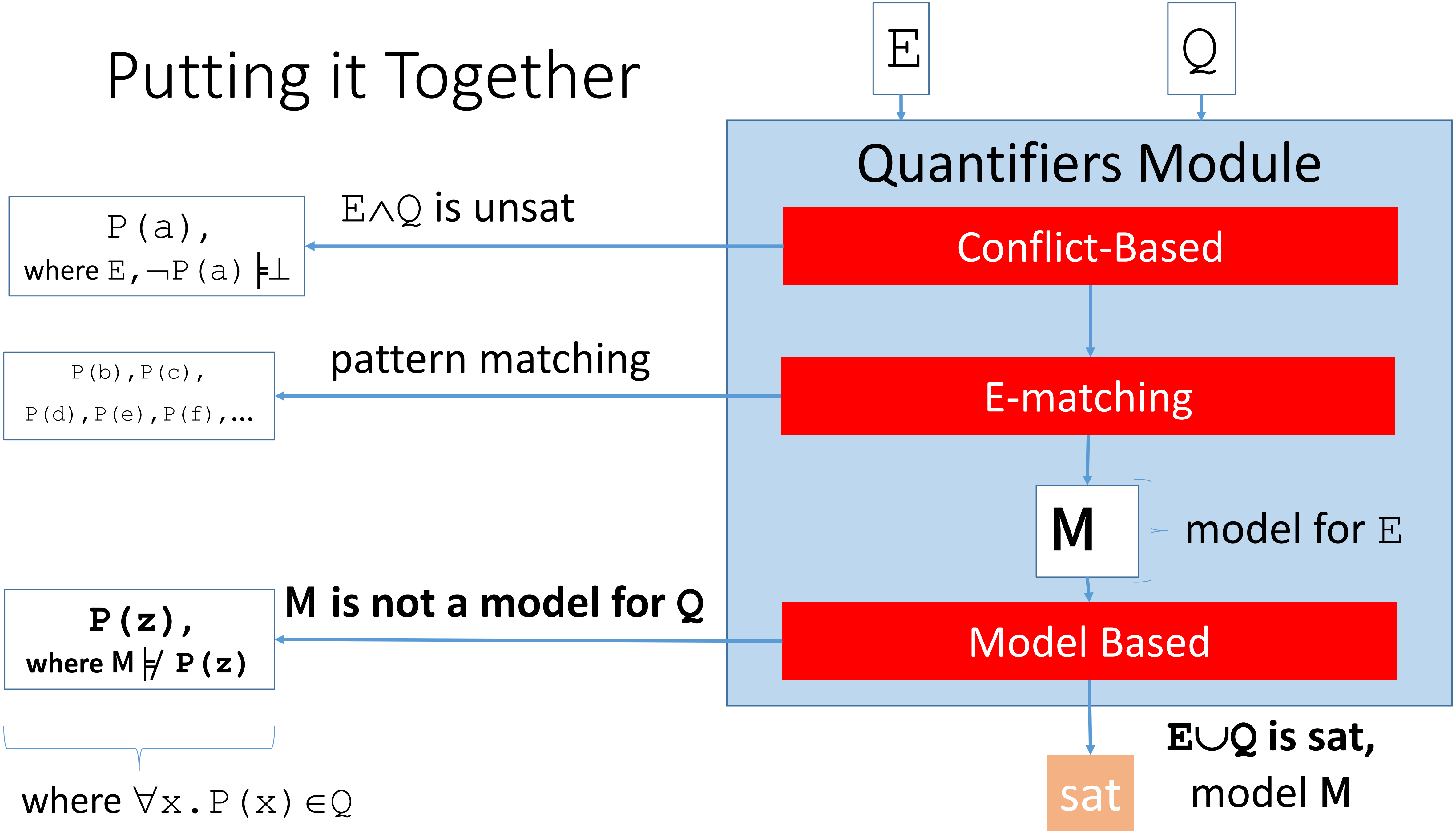
Putting it Together



Putting it Together



Putting it Together



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 ∀ + *pure* theories isn't as bad:

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

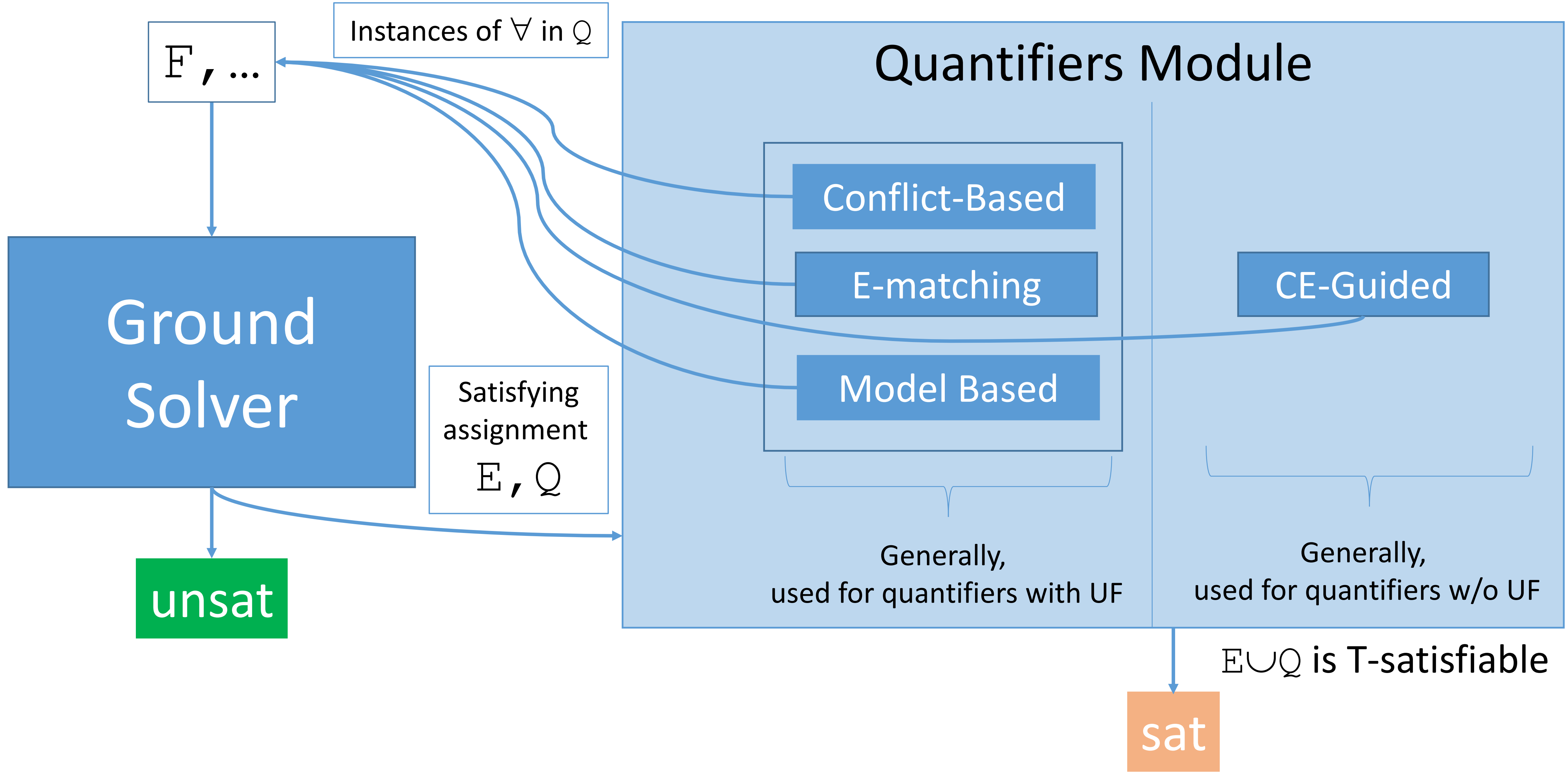
E-matching, Conflict-Based, Model-based:

- **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

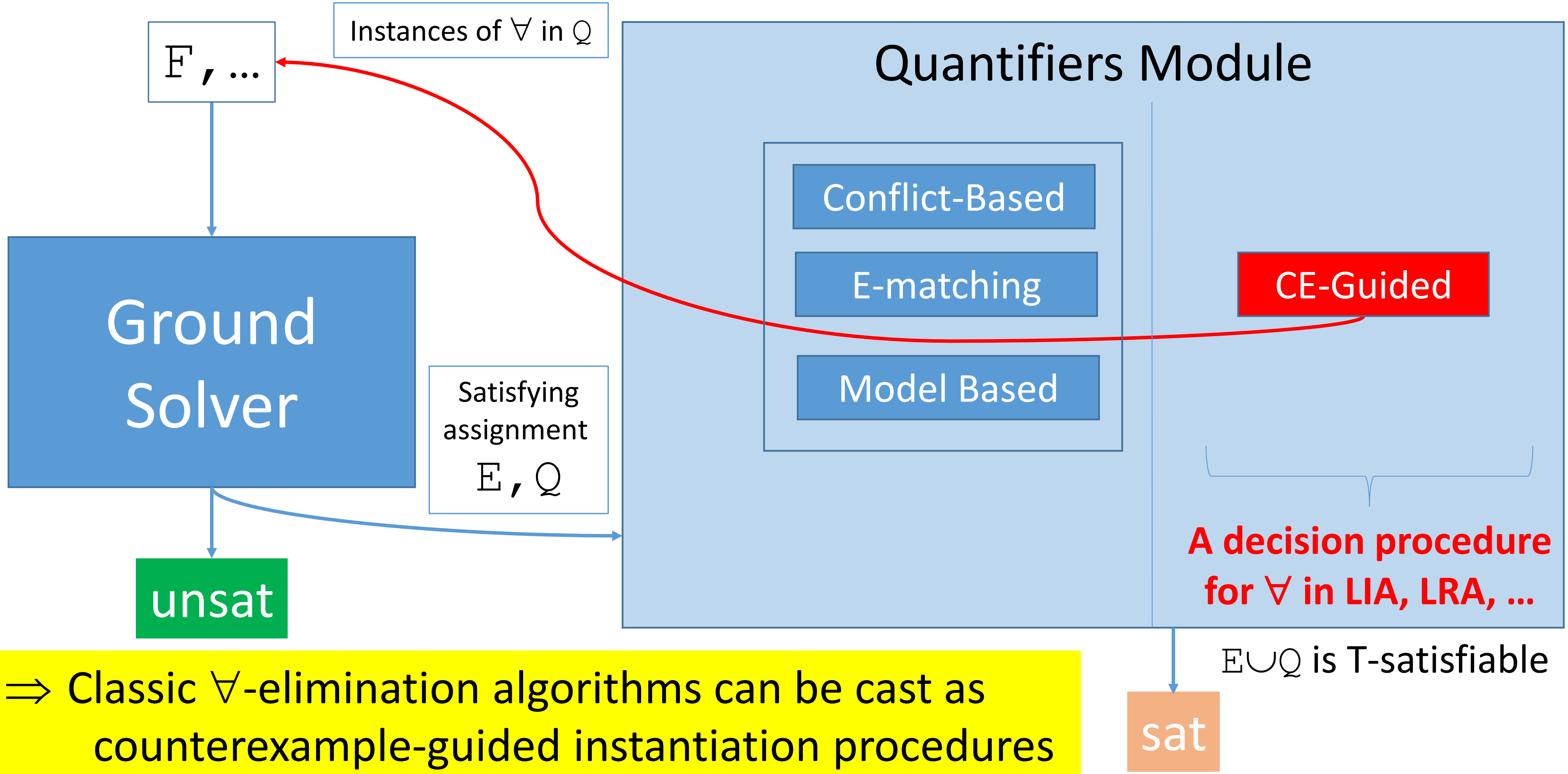
⇒ But reasoning about ∀ + *pure* theories isn't as bad:

- Classic ∀-elimination algorithms are decision procedures for ∀ in:
  - LRA [Ferrante+Rackoff 79, Loos+Wiespfenning 93] , LIA [Cooper 72], datatypes, …
- Can classic ∀-elimination algorithms be implemented in an SMT context?
  - Yes: [Monniaux 2010, Bjorner 2012, Komuravelli et al 2014, Reynolds et al 2015, Bjorner/Janota 2016]

Techniques for Quantifier Instantiation

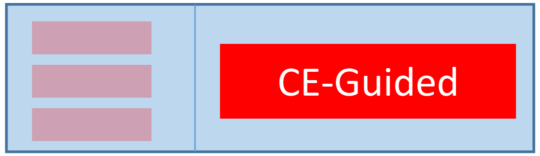


Techniques for Quantifier Instantiation



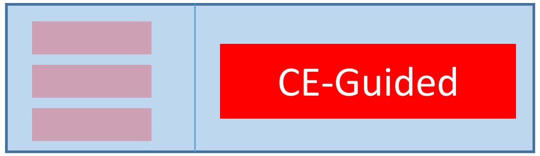
⇒ Classic ∀-elimination algorithms can be cast as counterexample-guided instantiation procedures

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]**
- High-level idea:
  - Quantifier elimination (e.g. for LIA) says: $\exists x \,.\, \psi[x] \Leftrightarrow \psi[t_1] \vee \ldots \vee \psi[t_n]$ for finite n

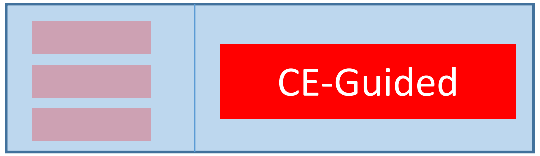
Counterexample-Guided Instantiation



- 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]
- High-level idea:
  - Quantifier elimination (e.g. for LIA) says: $\forall x.\neg\psi[x] \Leftrightarrow \neg\psi[t_1] \wedge \ldots \wedge \neg\psi[t_n]$ for finite n

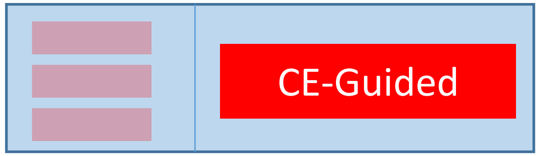
(consider the dual)

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]**
- High-level idea:
  - Quantifier elimination (e.g. for LIA) says: $\forall x . \neg\psi[x] \Leftrightarrow \neg\psi[\mathbf{t_1}] \wedge \ldots \wedge \neg\psi[\mathbf{t_n}]$ for finite n
  - Enumerate these instances lazily, via a counterexample-guided loop, that is:
    - Terminating: enumerate at most n instances
    - Efficient in practice: typically terminates after $m << n$ instances

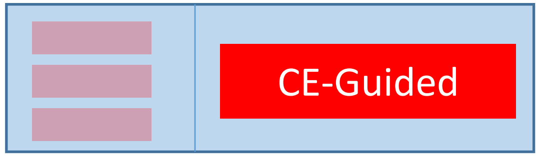
Counterexample-Guided Instantiation

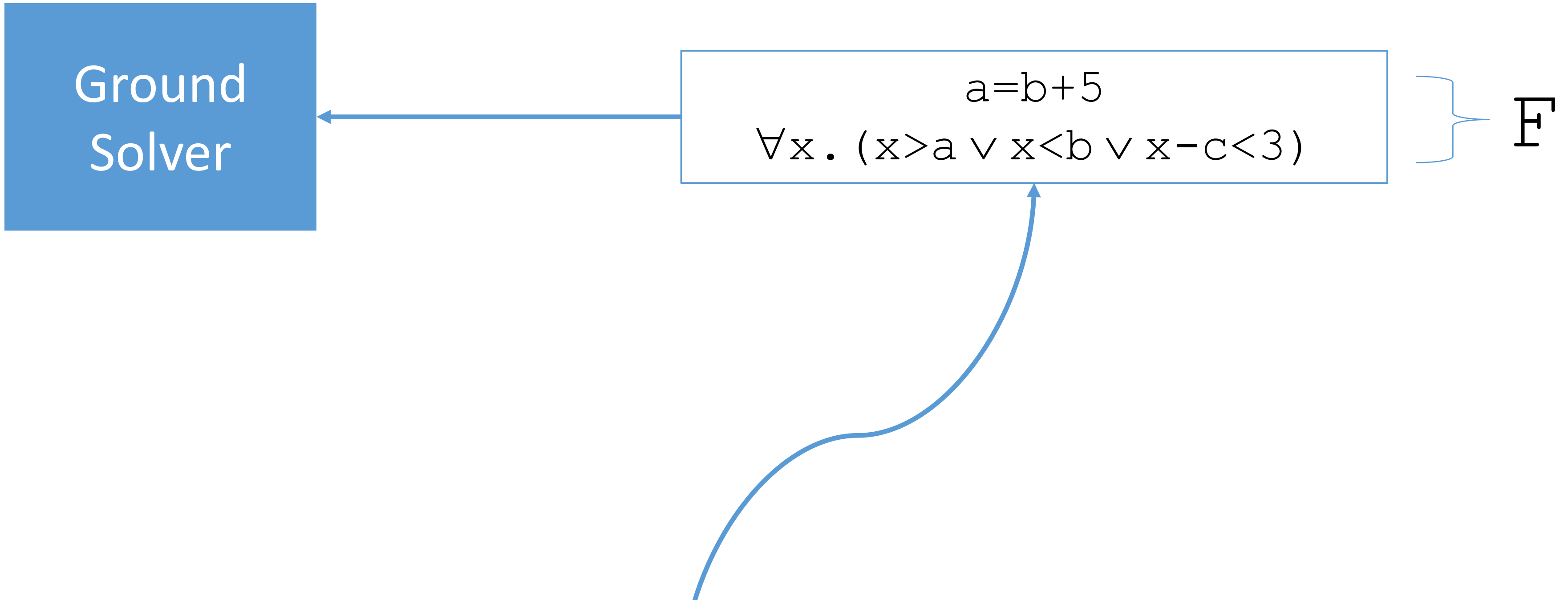


$\Rightarrow$ Consider $\forall$ in the theory of linear integer arithmetic LIA:

$$\exists abc.(a=b+5 \wedge \forall x.(x>a \vee x<b \vee x-c<3))$$

Counterexample-Guided Instantiation



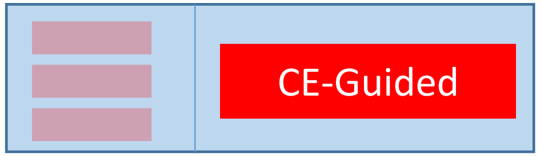


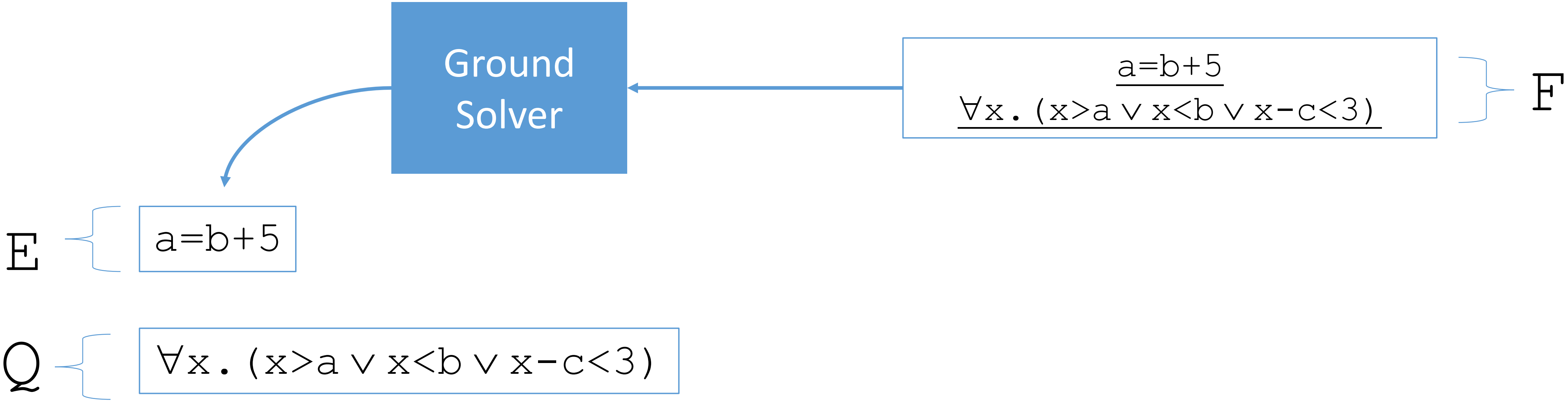
⇒ Consider ∀ in the theory of linear integer arithmetic LIA:

~~∃abc~~. (**a=b+5 ∧ ∀x. (x>a ∨ x<b ∨ x-c<3)**)

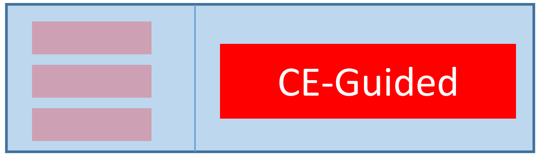
- Outermost existentials `a`, `b`, `c` are treated as *free constants*

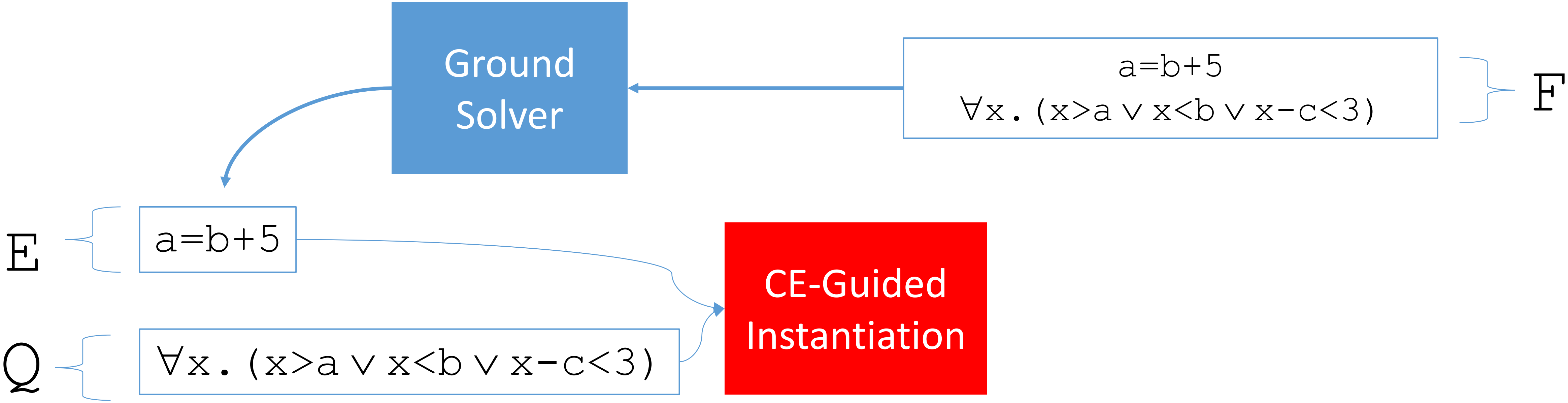
Counterexample-Guided Instantiation





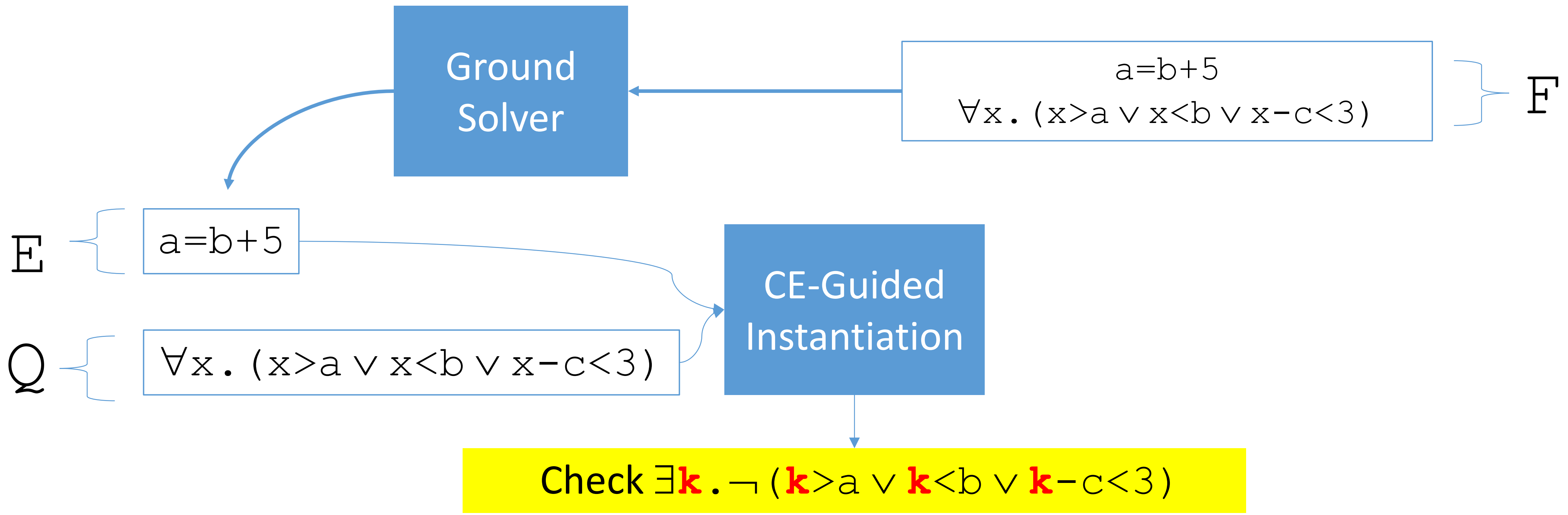
Counterexample-Guided Instantiation





⇒ Use counterexample-guided instantiation

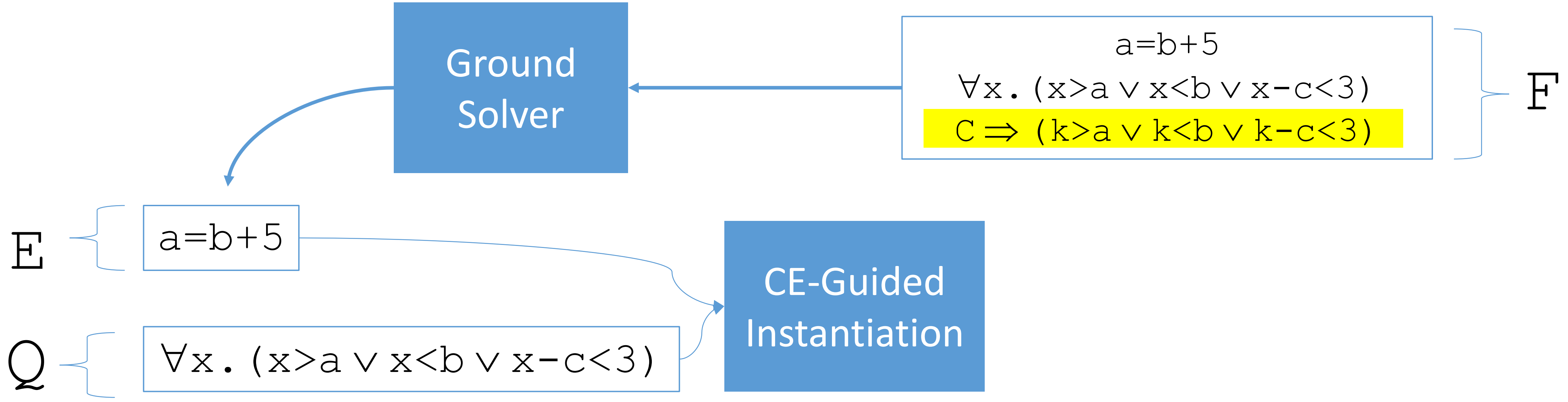
Counterexample-Guided Instantiation



⇒With respect to *model-based instantiation*:

- Similar: check satisfiability of $\exists \mathbf{k}.\neg(\mathbf{k}>a \vee \mathbf{k}<b \vee \mathbf{k}-c<3)$

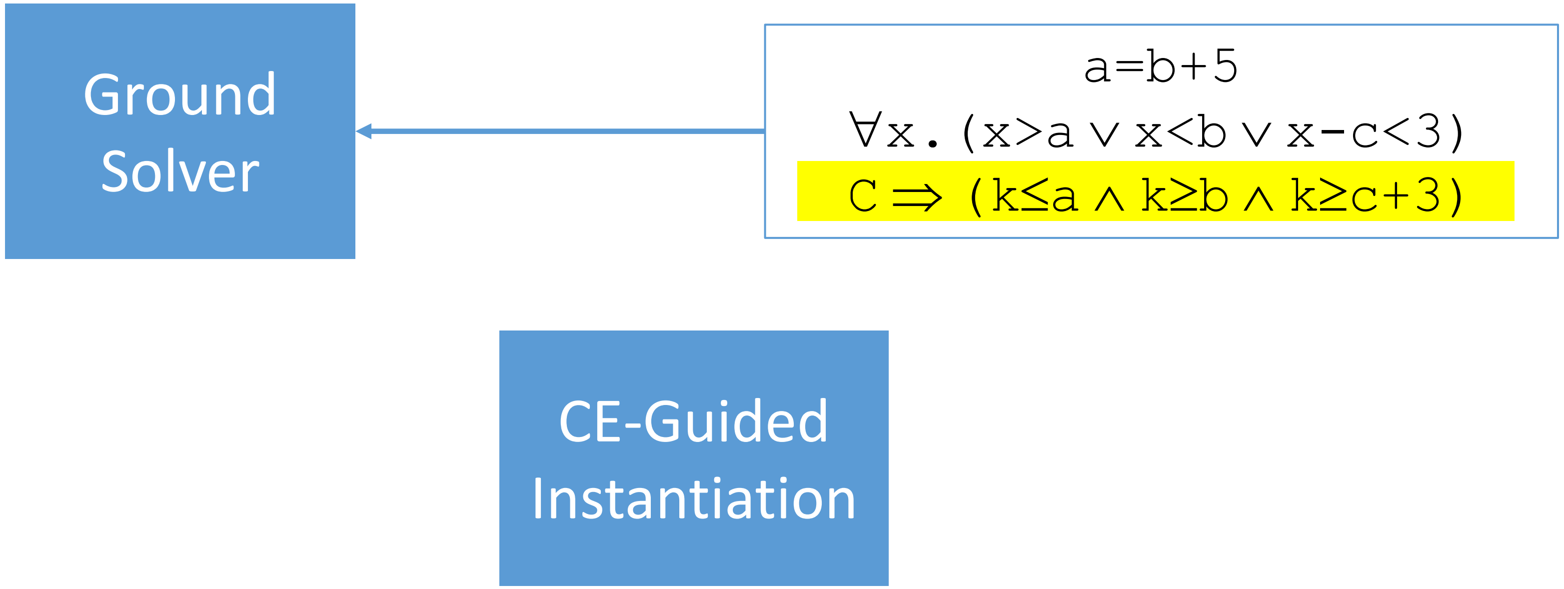
Counterexample-Guided Instantiation



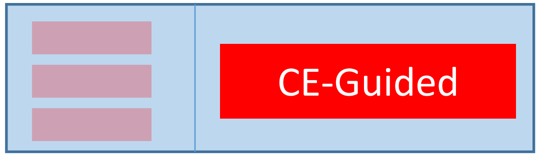
⇒With respect *to model-based instantiation*:

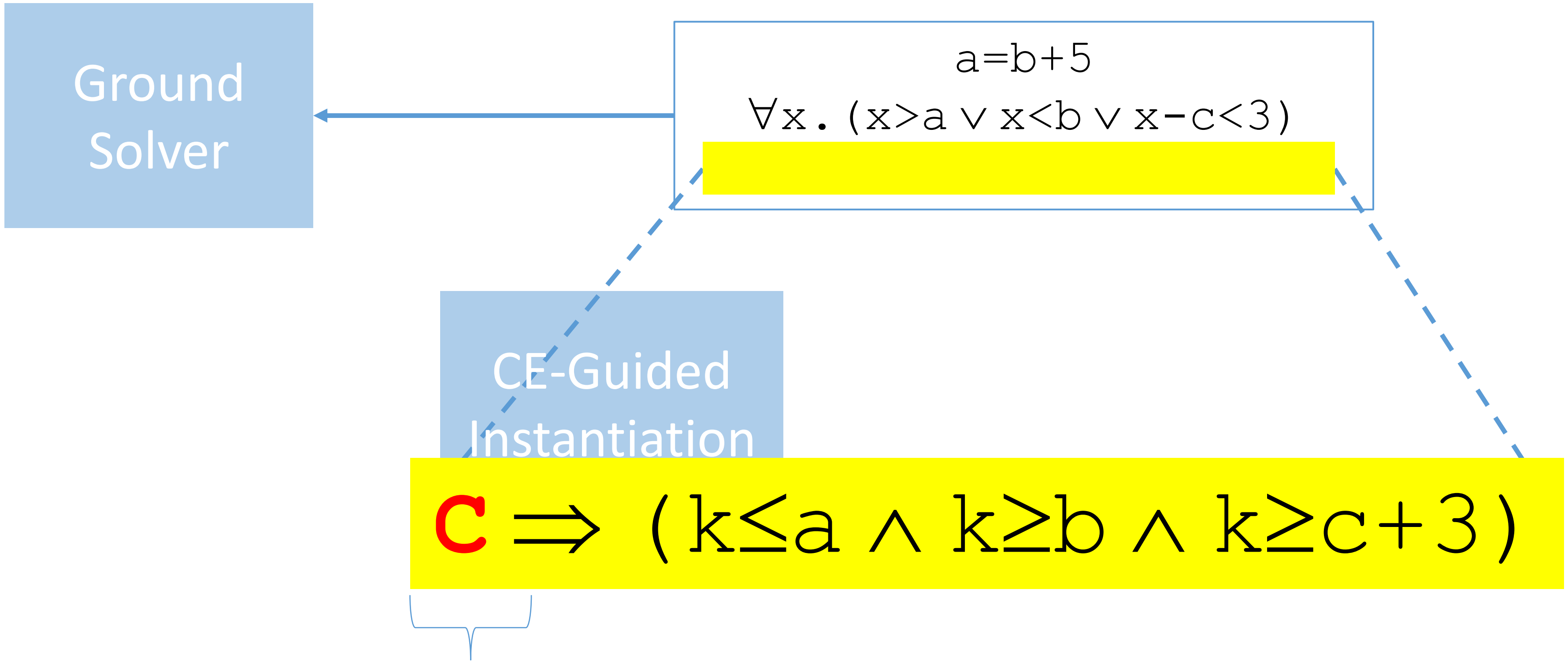
- Similar: check satisfiability of $\exists k. \neg (k>a \vee k<b \vee k-c<3)$
- *Key difference:* use the same (ground) solver for F and *counterexample* k *for* Q

Counterexample-Guided Instantiation



Counterexample-Guided Instantiation

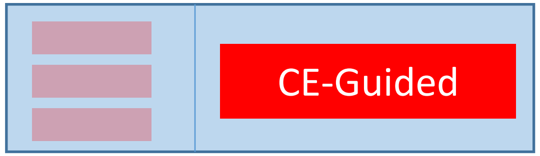


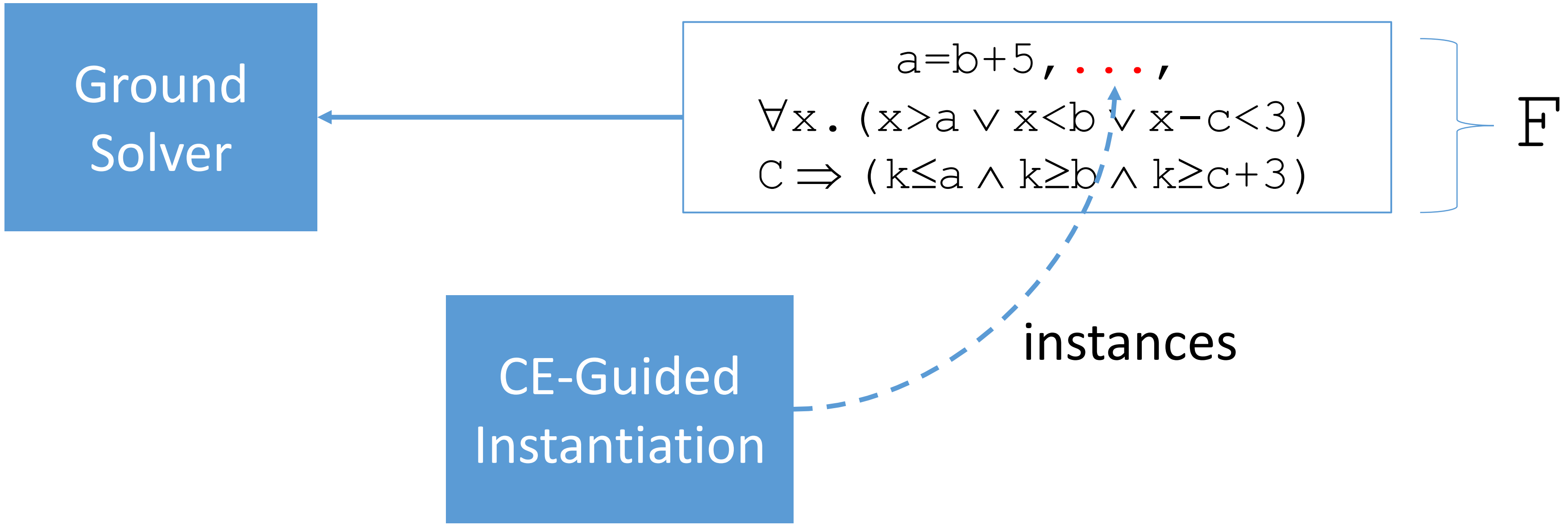


C is a fresh Boolean variable:

"A counterexample k exists for $\forall x.(x>a \lor x<b \lor x-c<3)$"

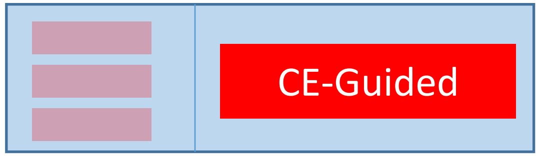
Counterexample-Guided Instantiation

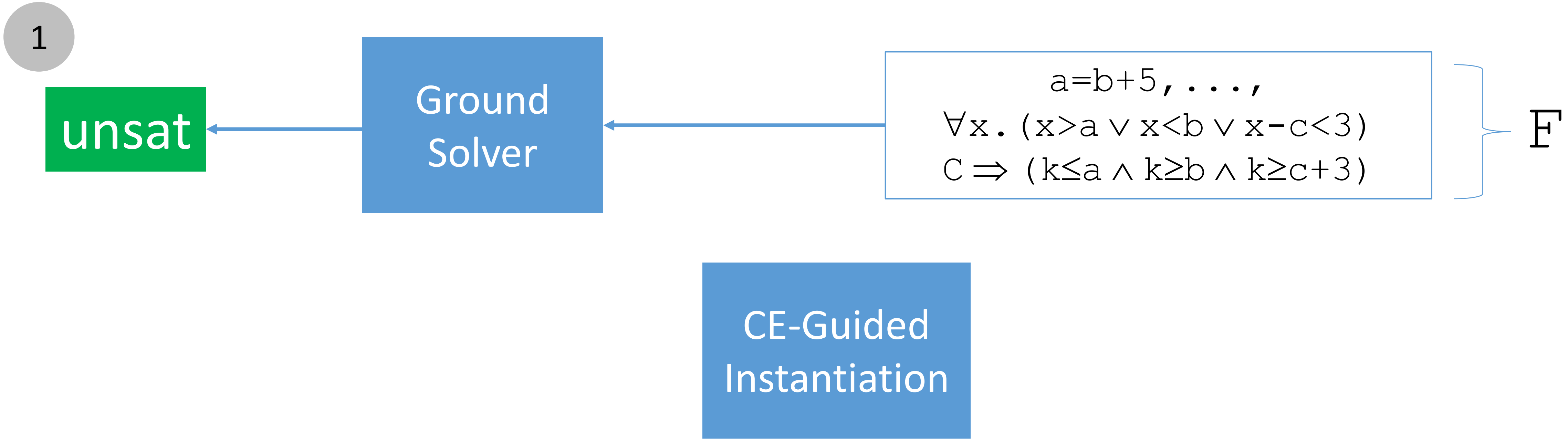




- Three cases:

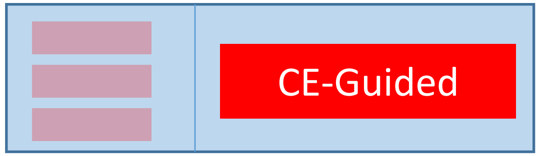
Counterexample-Guided Instantiation

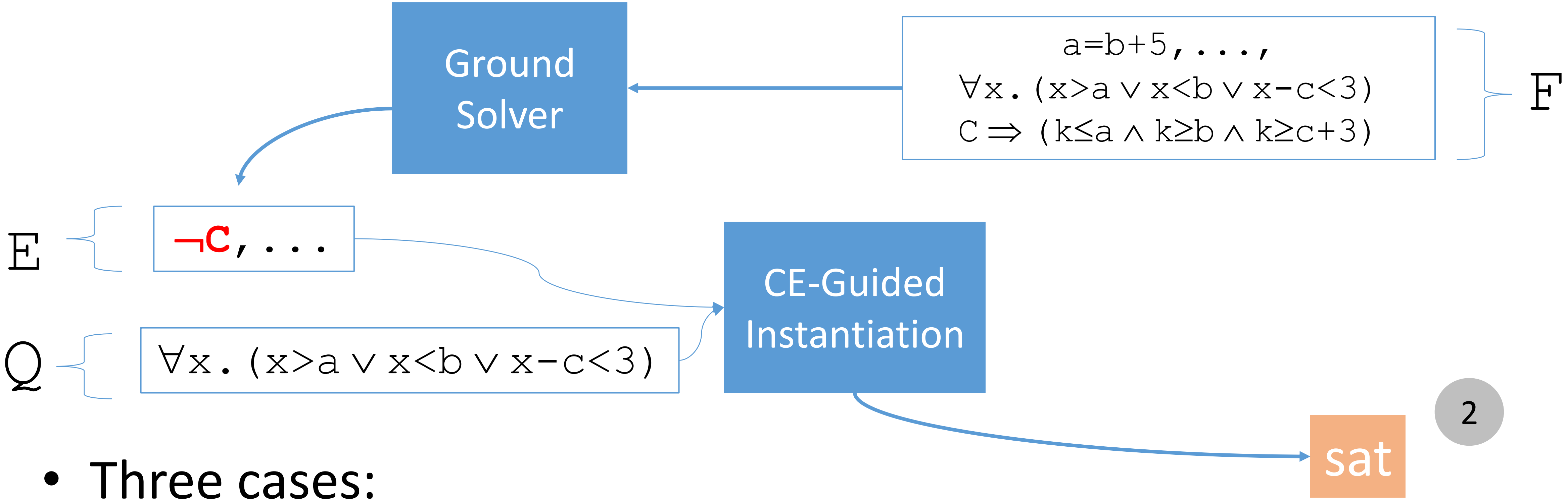




- Three cases:
  1. **F is unsatisfiable** ⇒ **answer "unsat"**

Counterexample-Guided Instantiation



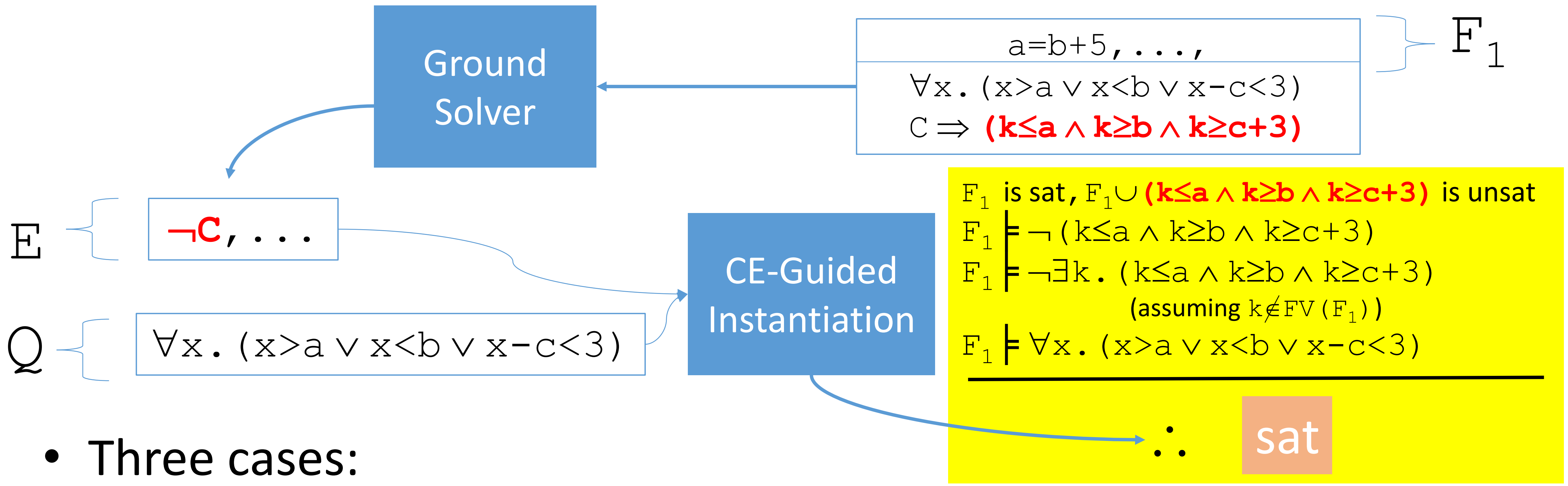


- Three cases:

2. F is satisfiable, ¬C∈E for *all* assignments E ⇒ answer "sat"

Counterexample-Guided Instantiation

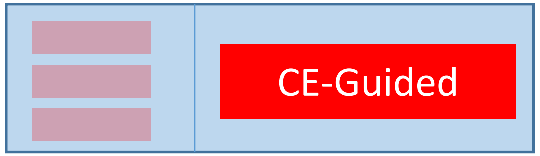
CE-Guided

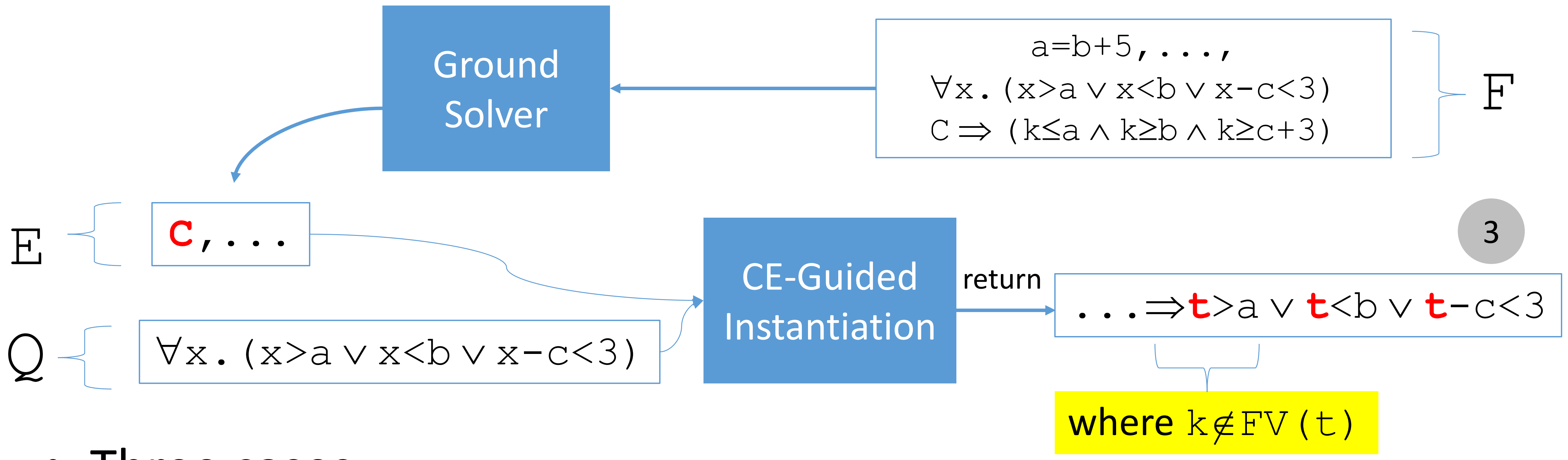


- Three cases:

2. F is satisfiable, ¬C∈E for *all* assignments E ⇒ answer "sat"

Counterexample-Guided Instantiation

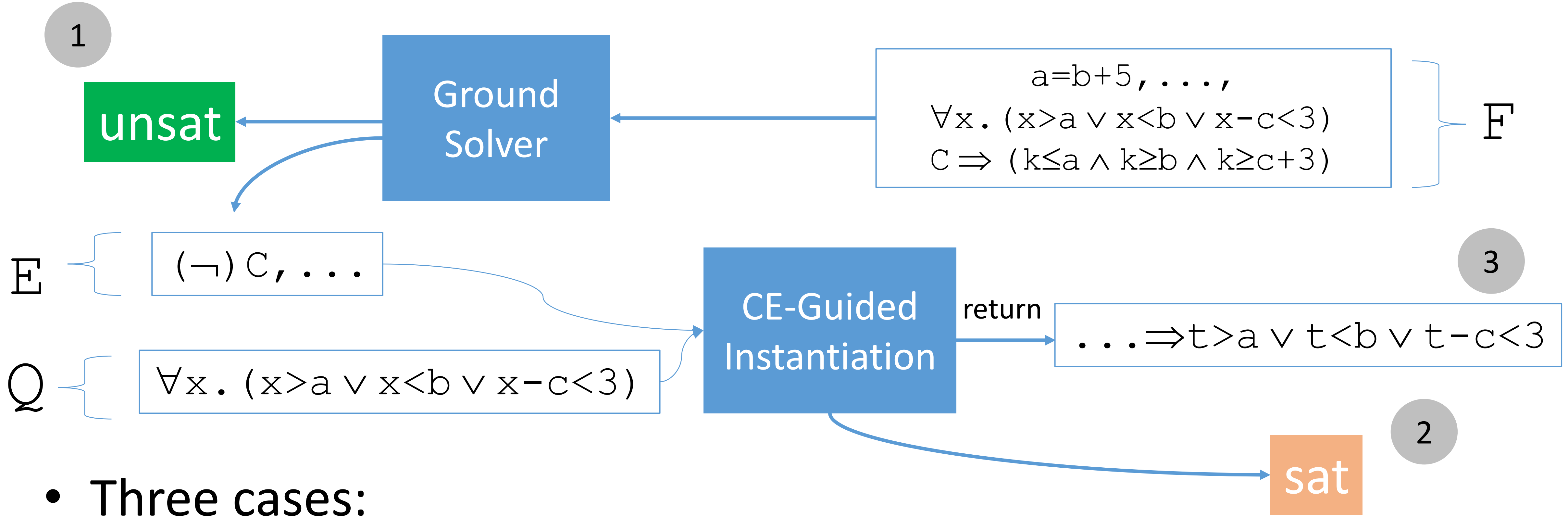




- Three cases:

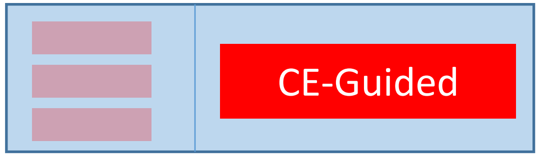
3. F is satisfiable, C∈E for *some* assignment E ⇒ add an instance to F

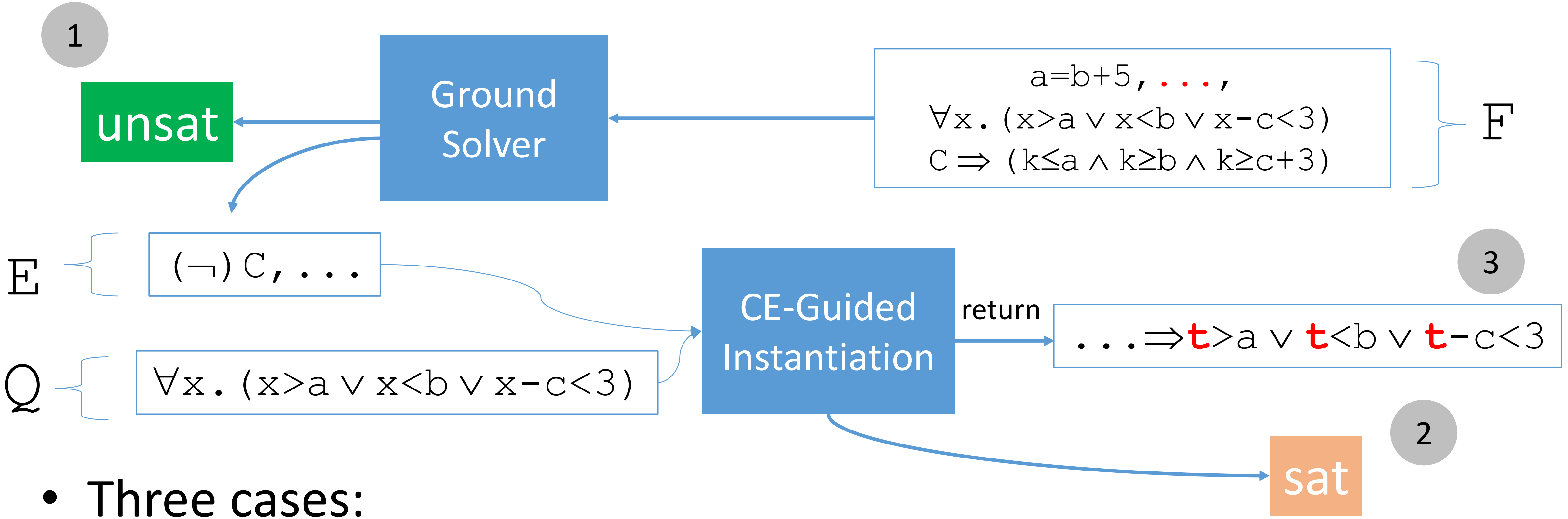
Counterexample-Guided Instantiation



- Three cases:
  1. F is unsatisfiable ⇒ answer "unsat"
  2. F is satisfiable, ¬C∈E for all assignments E ⇒ answer "sat"
  3. F is satisfiable, C∈E for some assignment E ⇒ add an instance to F

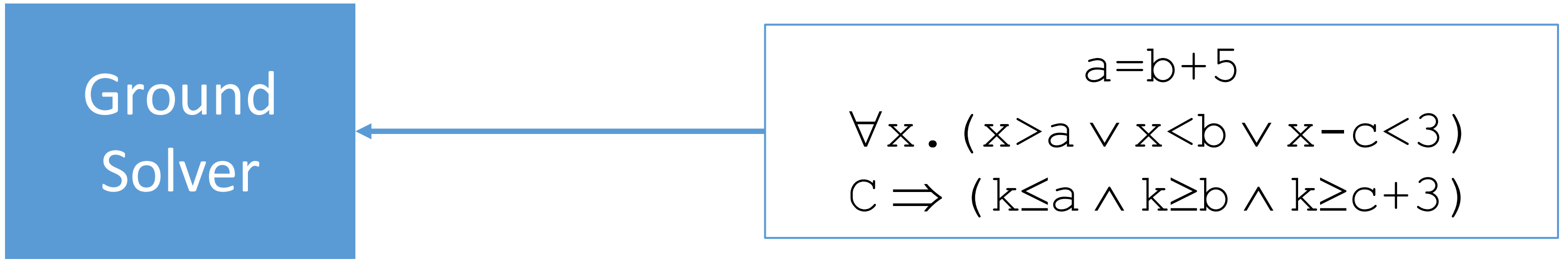
Counterexample-Guided Instantiation



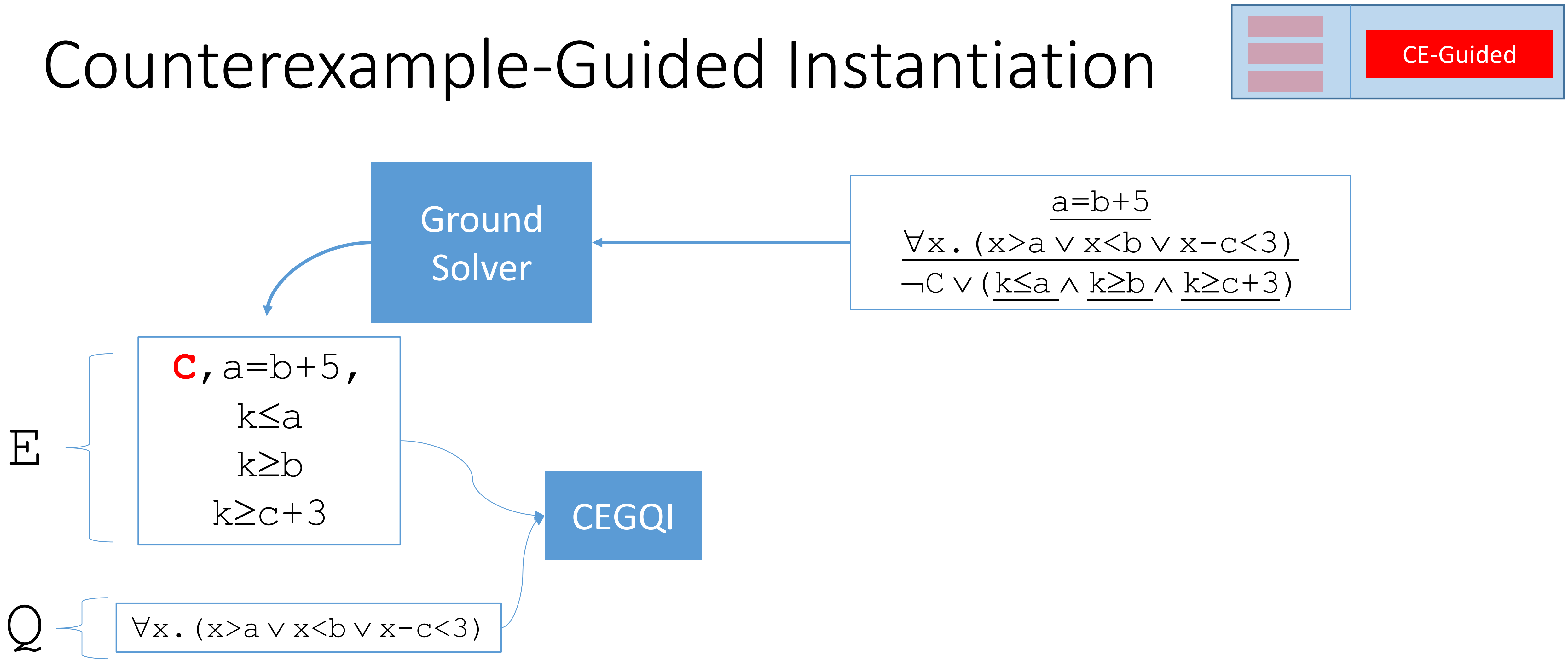


- Three cases:
  1. F is unsatisfiable ⇒ answer “unsat”
  2. F is satisfiable, ¬C∈E for all assignments E ⇒ answer “sat”
  3. F is satisfiable, C∈E for some assignment E ⇒ add **an instance** to F (…which **t**?)

Counterexample-Guided Instantiation

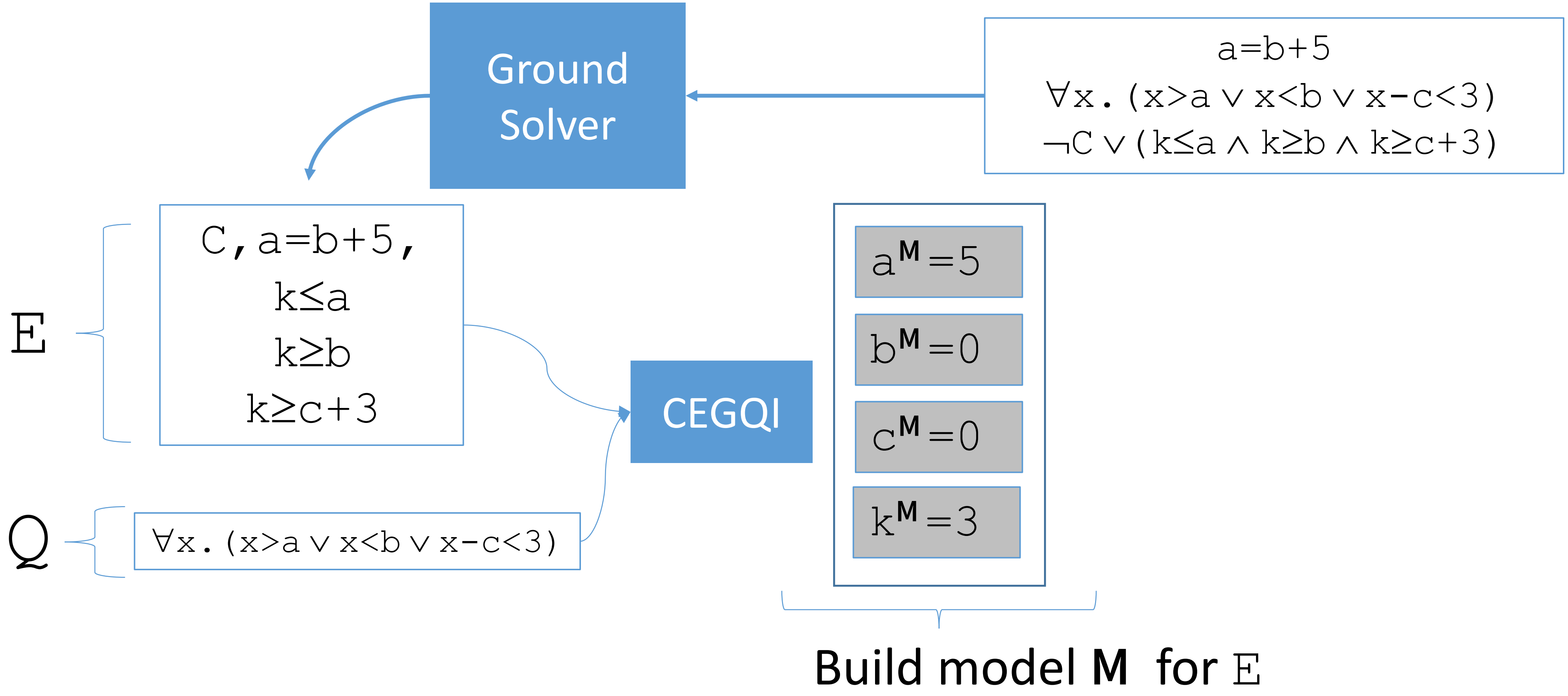


Counterexample-Guided Instantiation



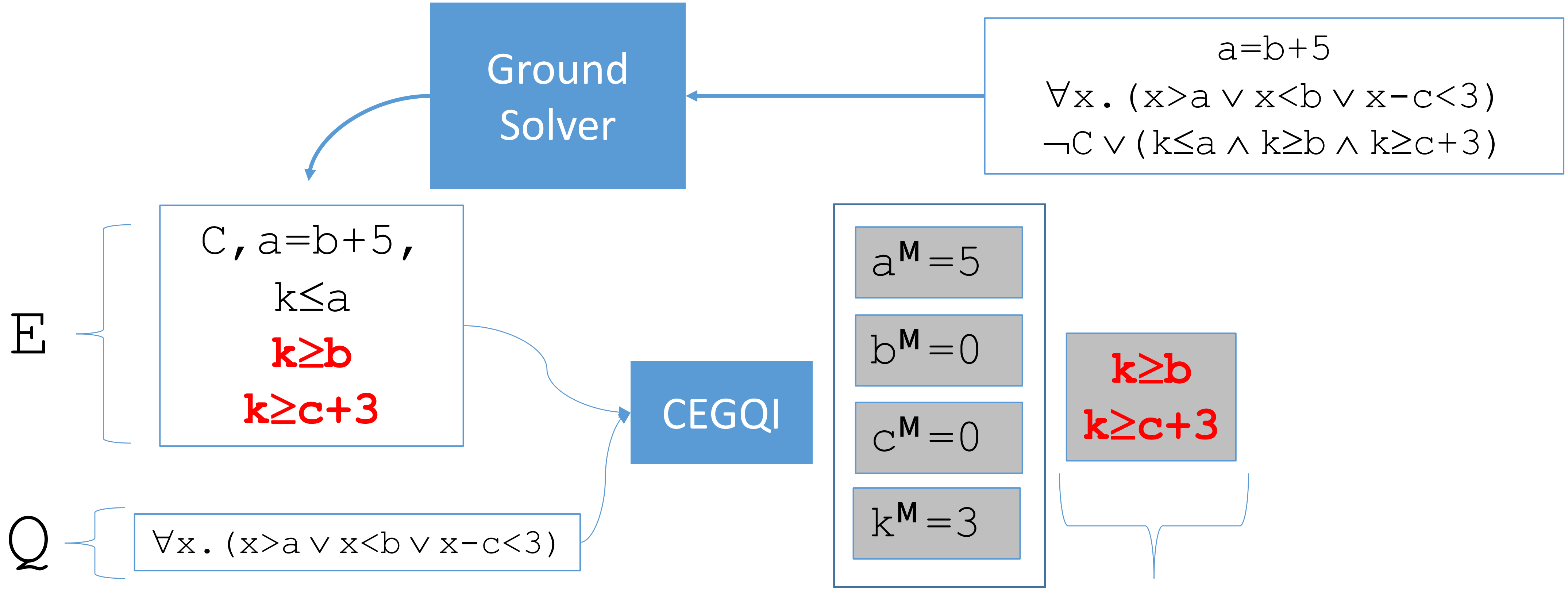
Counterexample-Guided Instantiation

CE-Guided



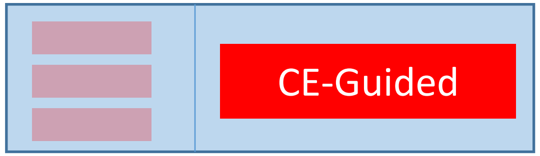
Counterexample-Guided Instantiation

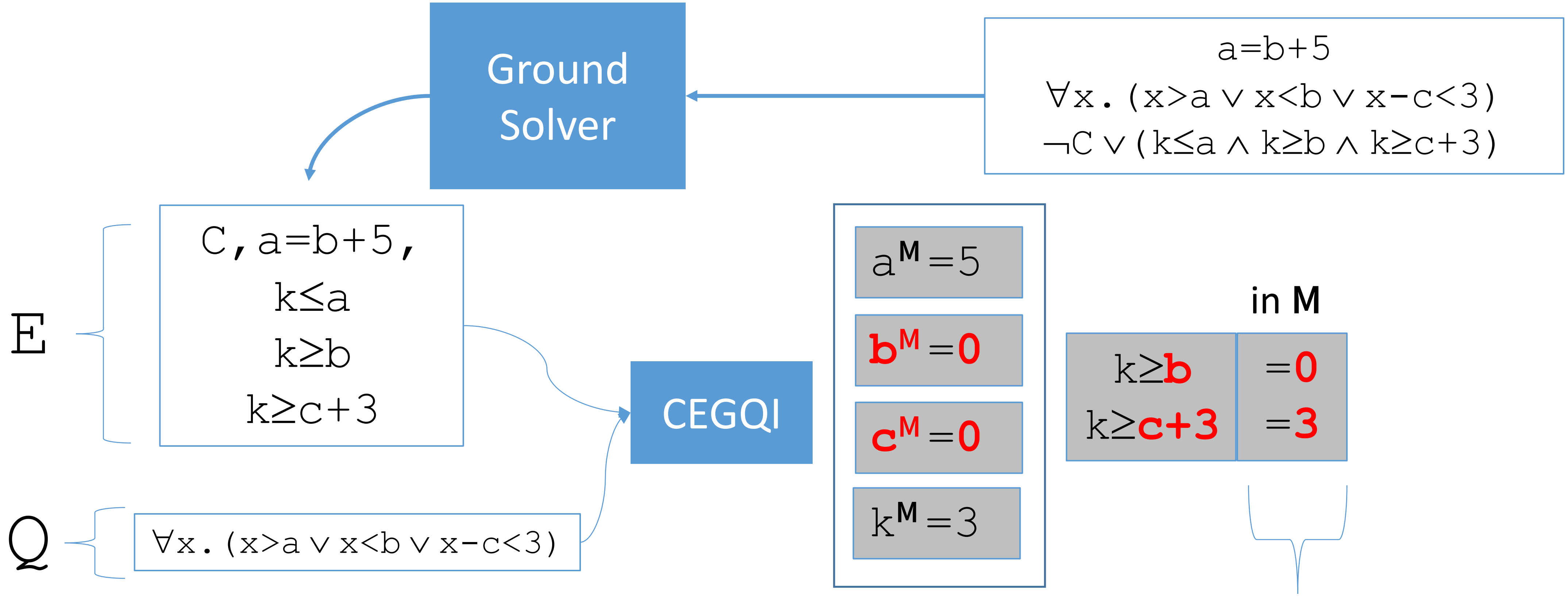
CE-Guided



Take lower bounds of k in E

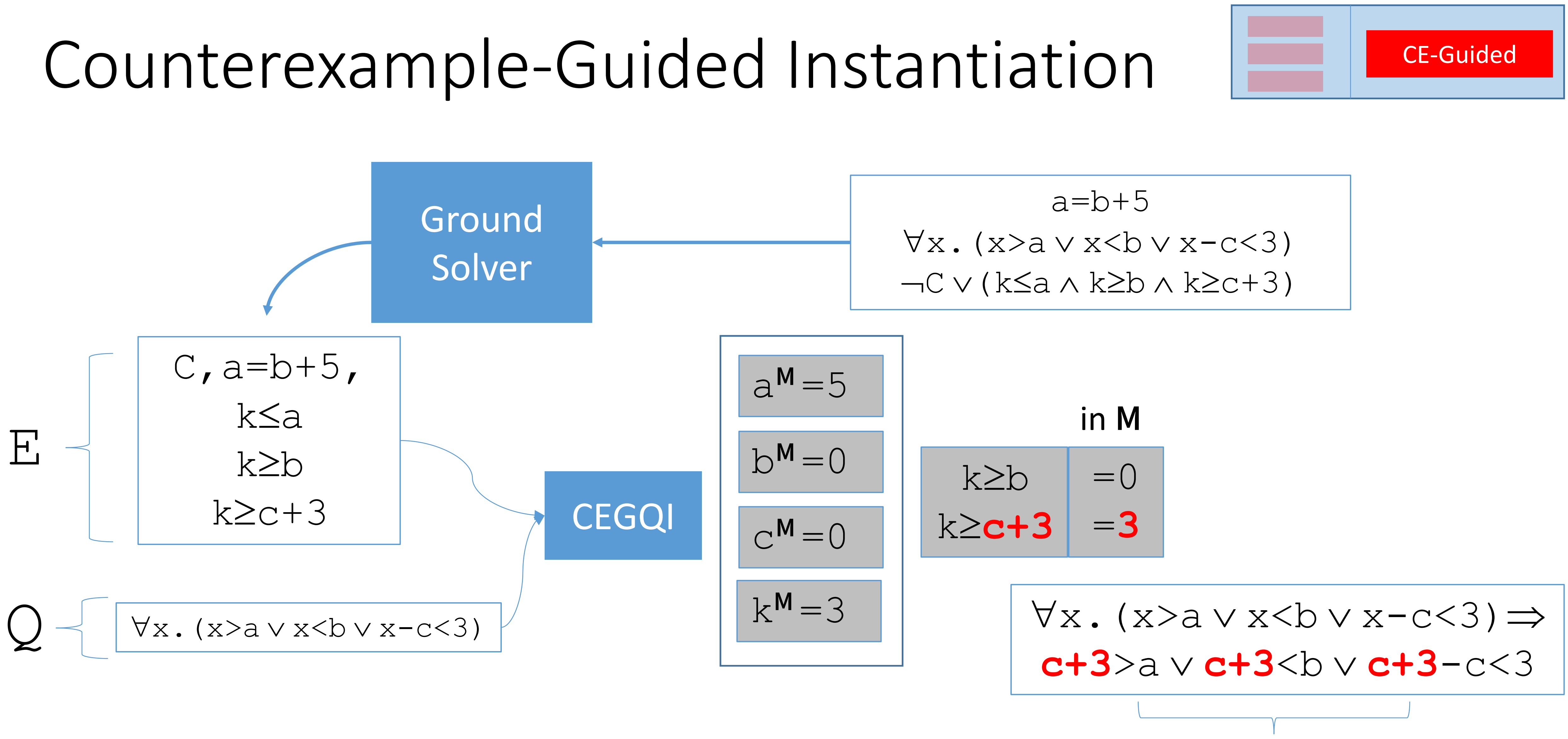
Counterexample-Guided Instantiation





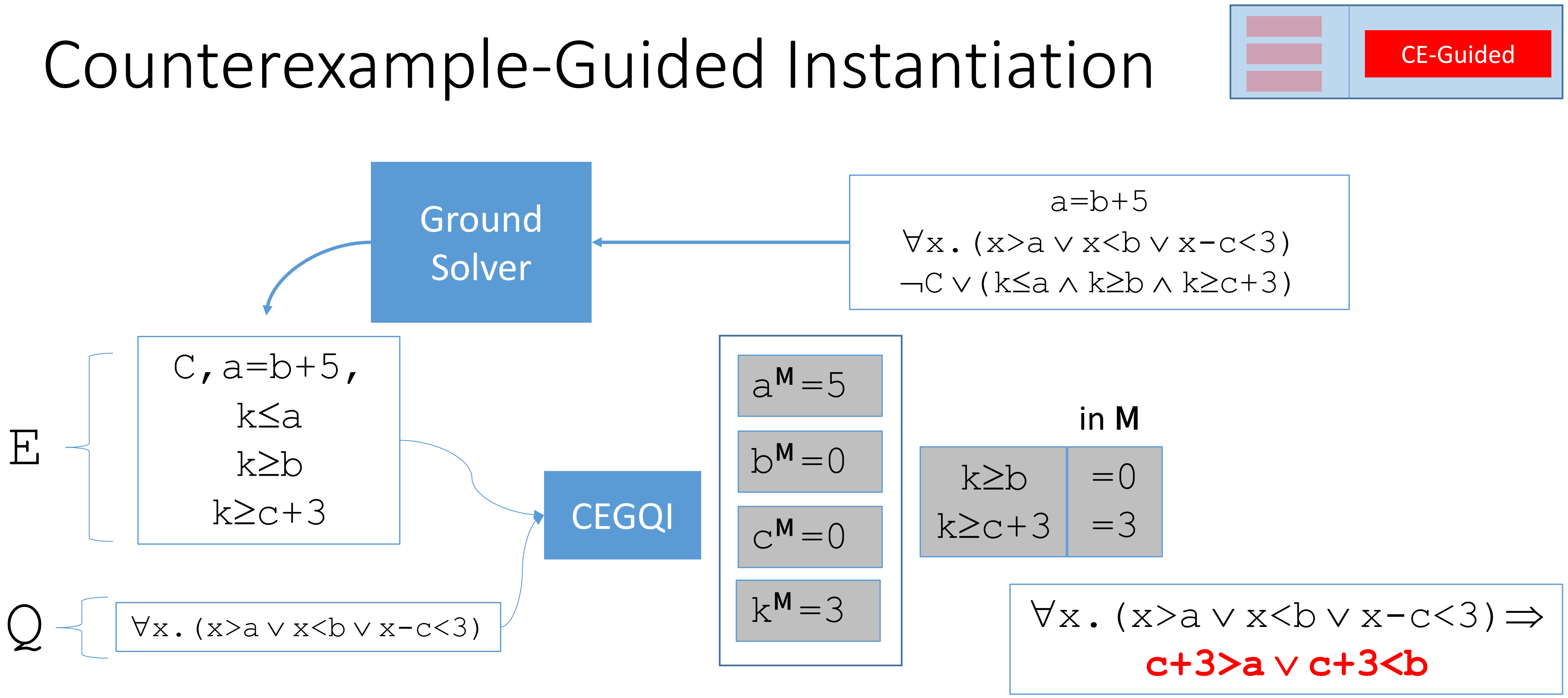
Compute their value in M

Counterexample-Guided Instantiation

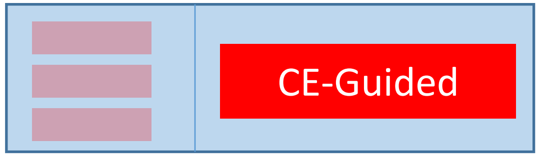


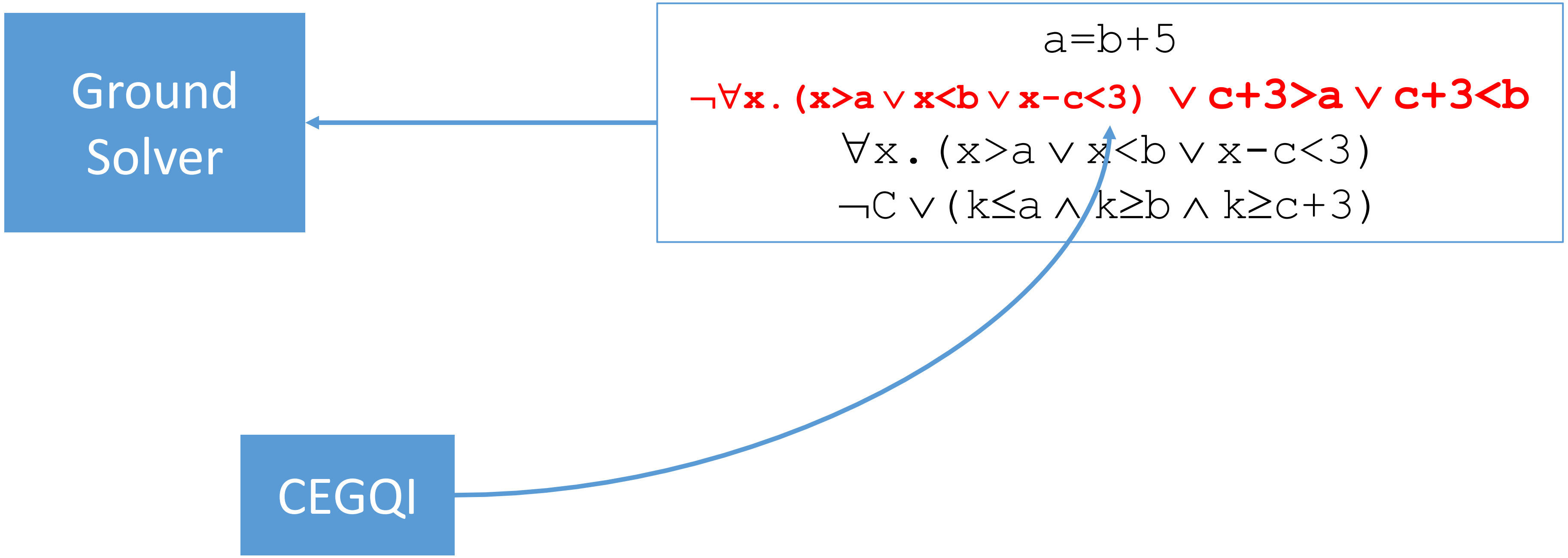
Add instance for lower bound that is maximal in **M**

Counterexample-Guided Instantiation



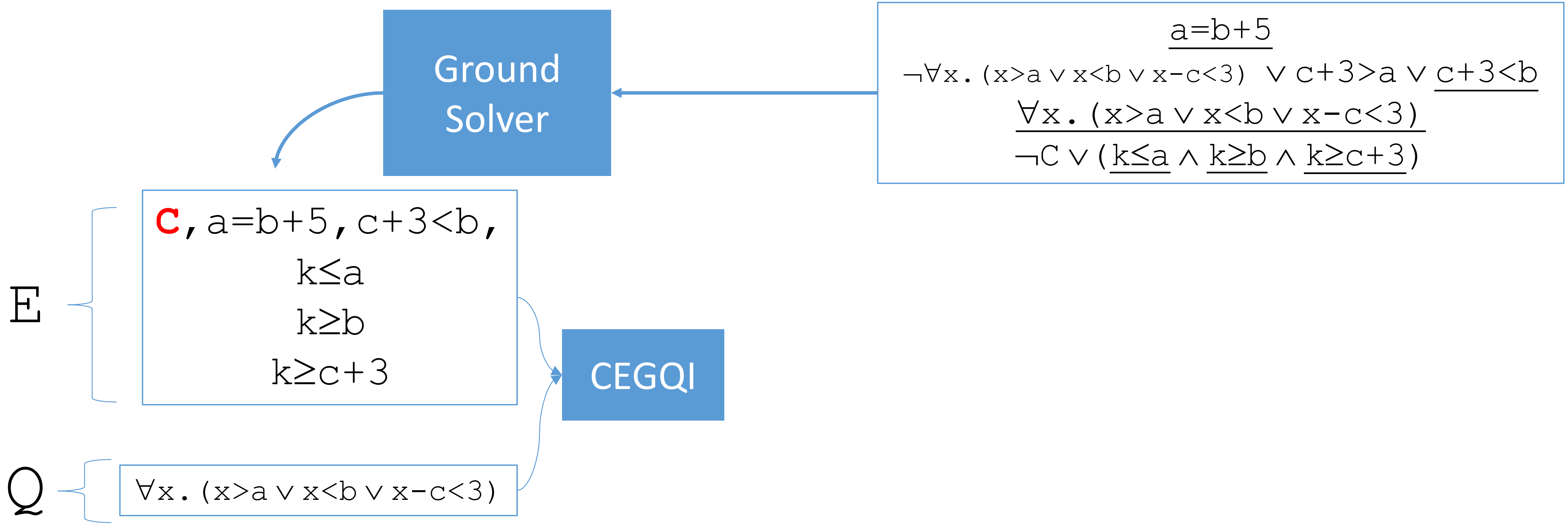
Counterexample-Guided Instantiation



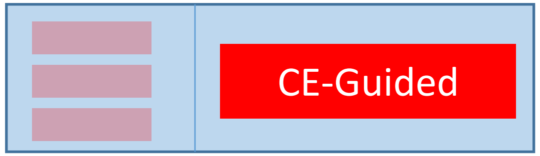


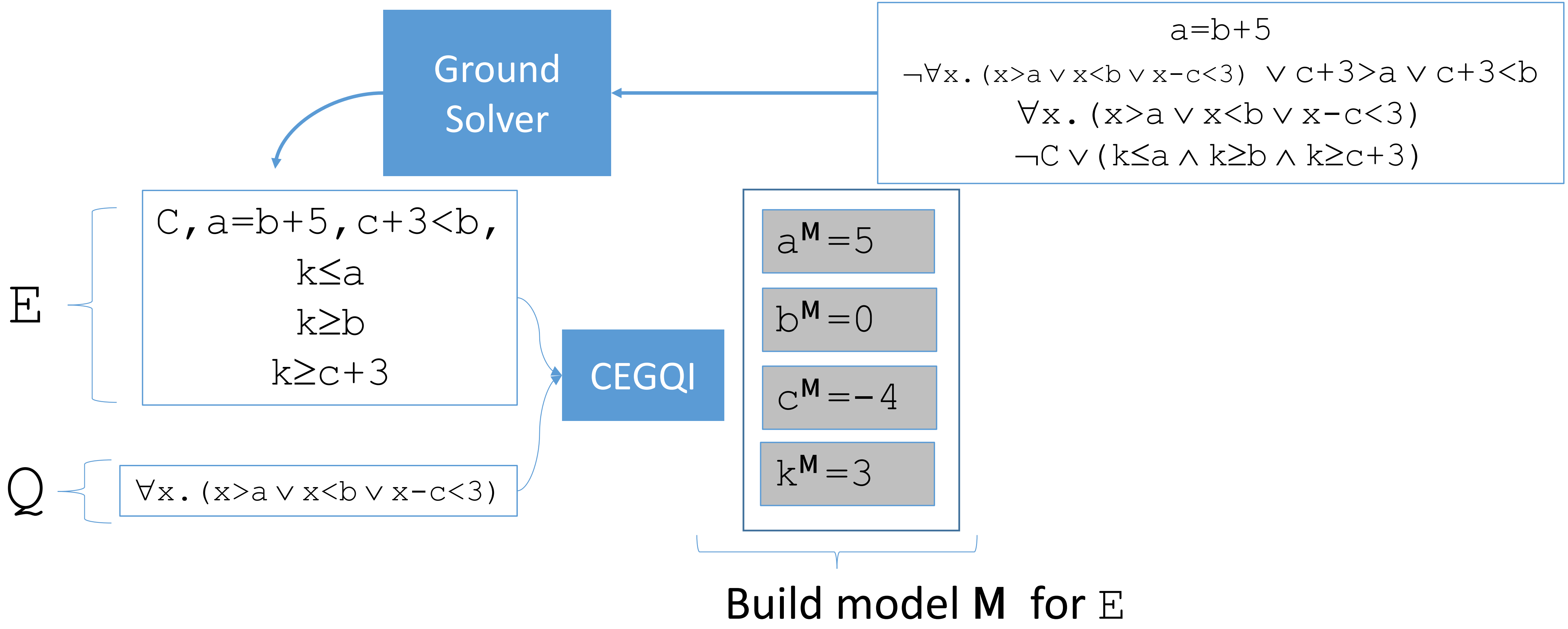
Counterexample-Guided Instantiation

CE-Guided

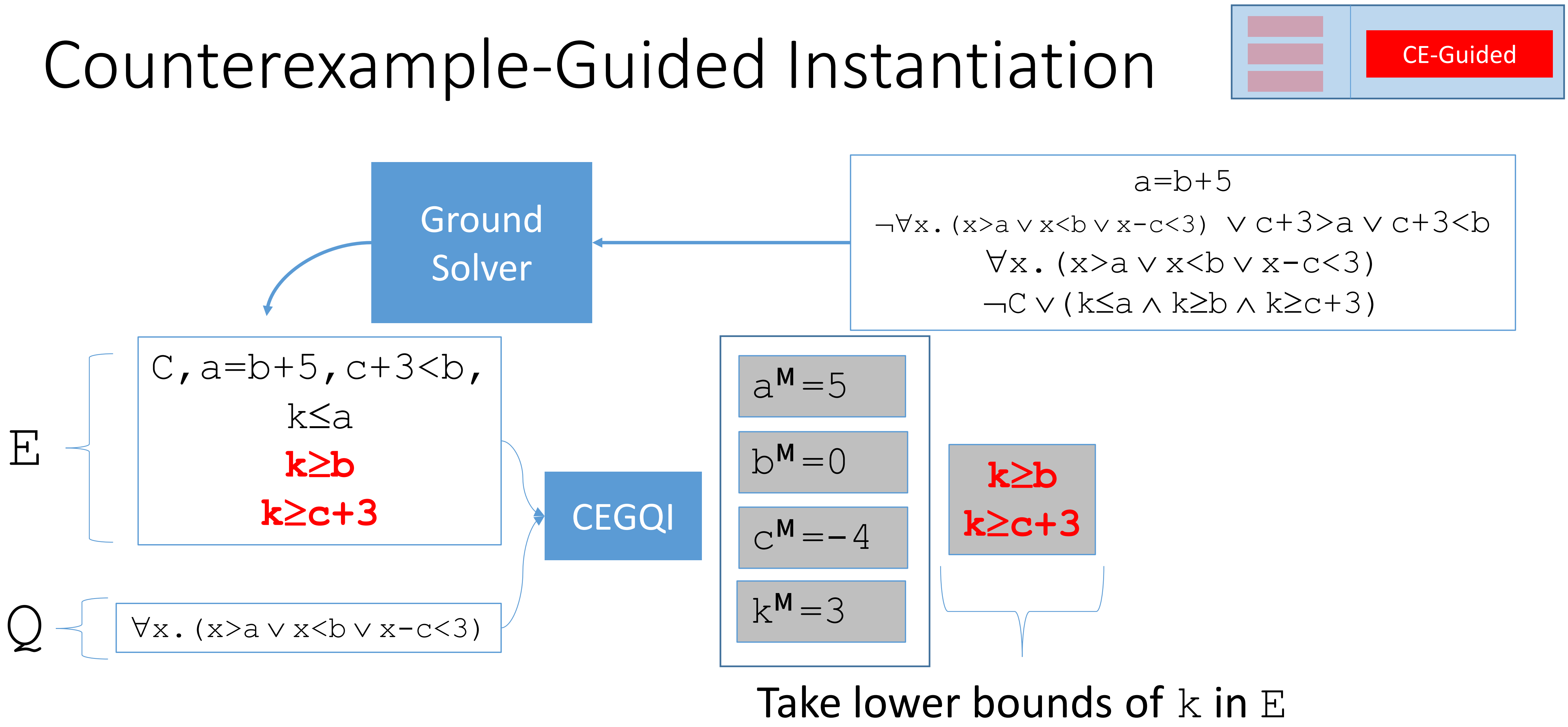


Counterexample-Guided Instantiation

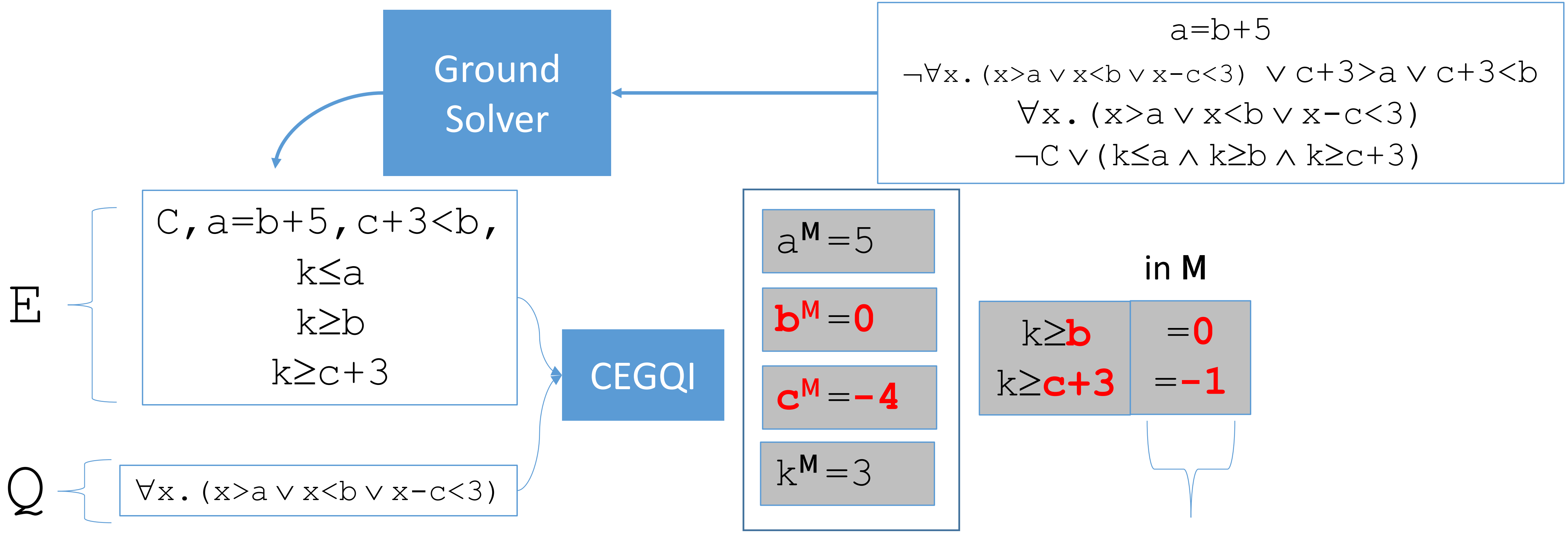




Counterexample-Guided Instantiation

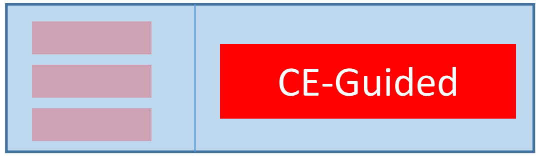


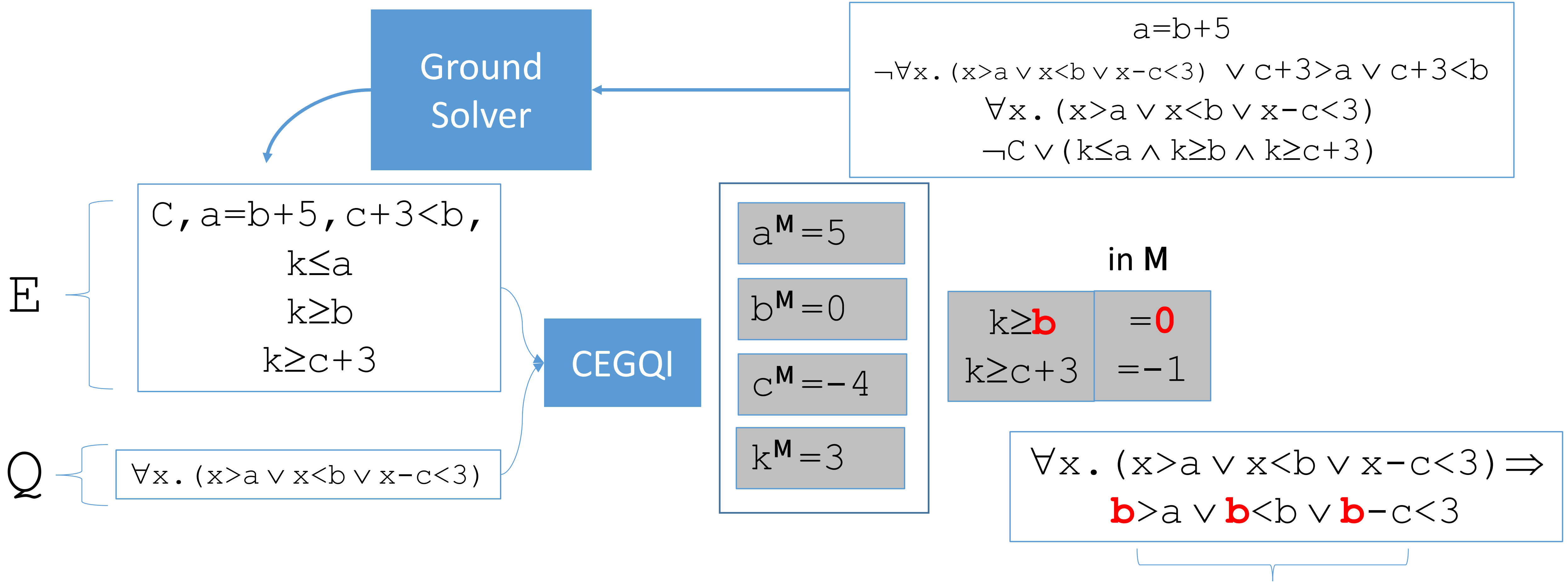
Counterexample-Guided Instantiation



Compute their value in **M**

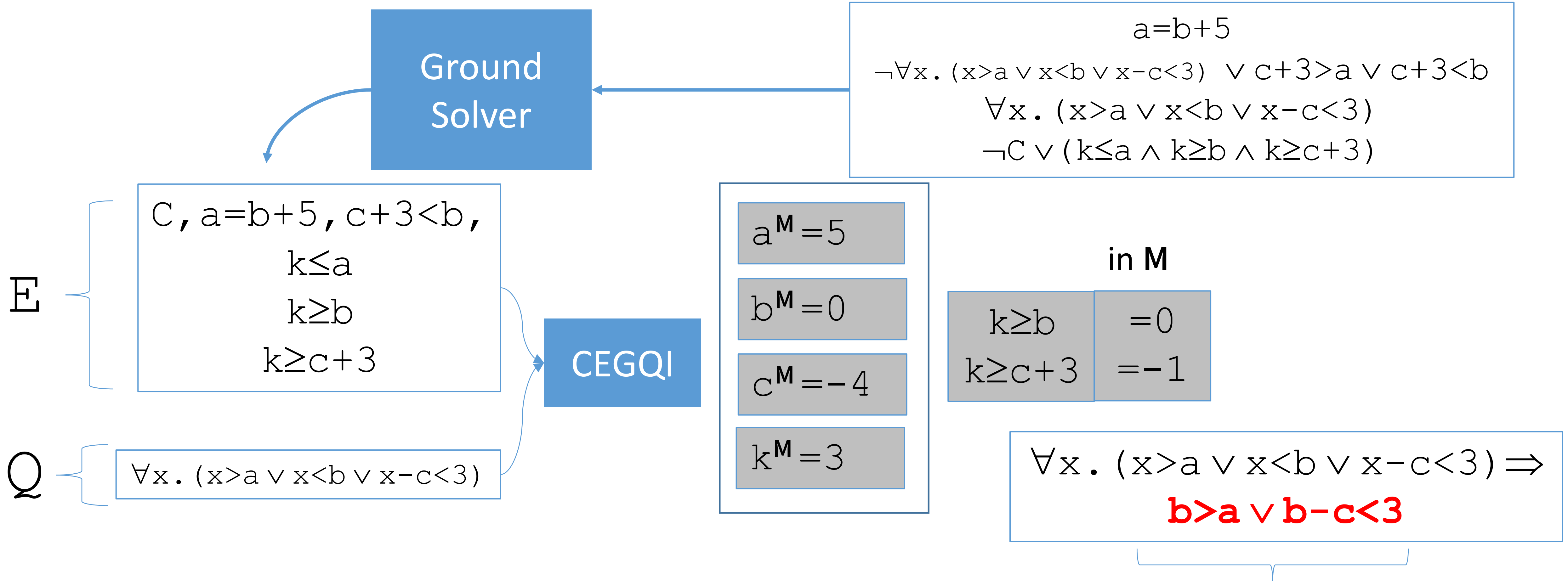
Counterexample-Guided Instantiation





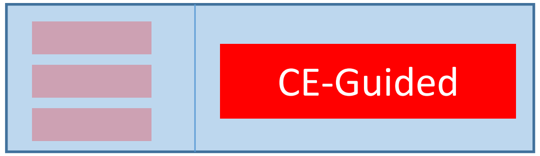
Add instance for lower bound that is maximal in **M**

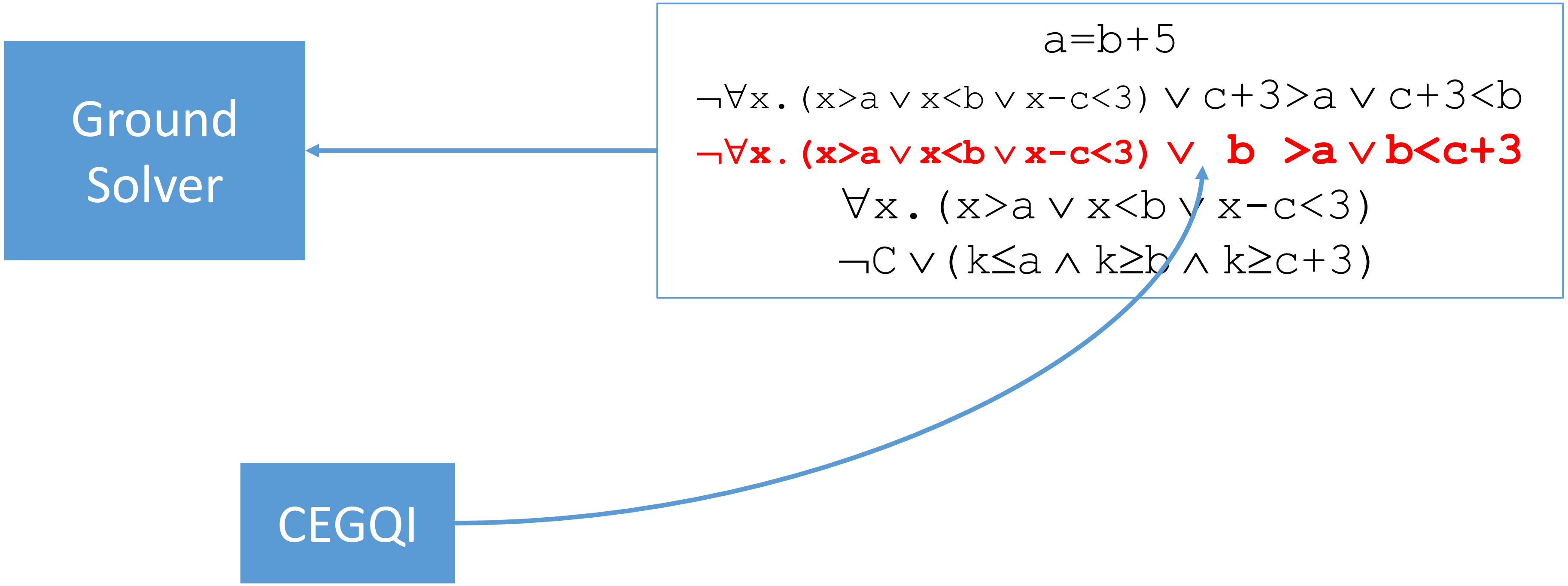
Counterexample-Guided Instantiation



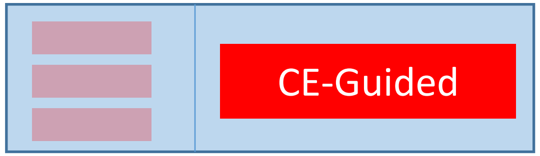
Add instance for lower bound that is maximal in **M**

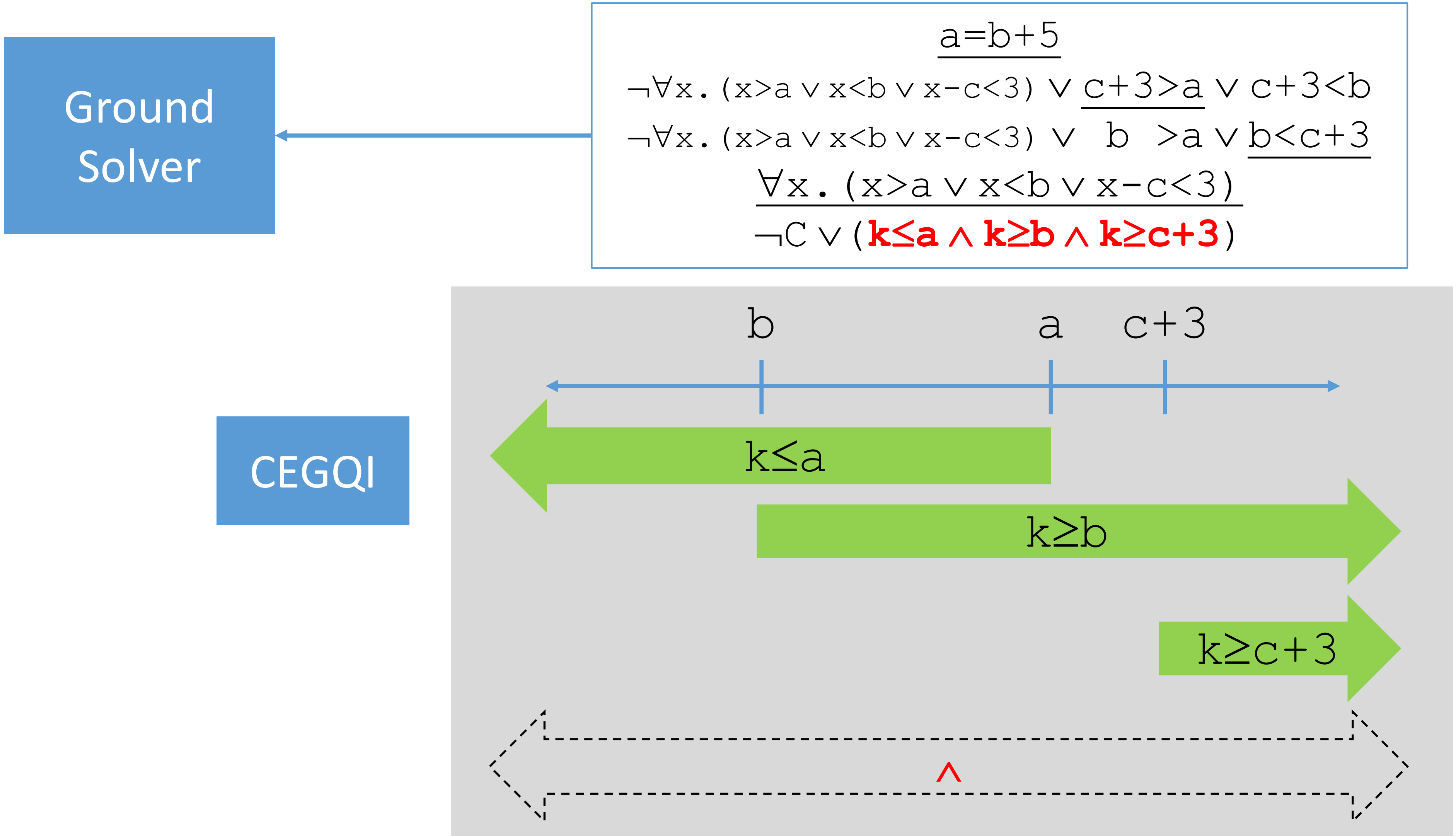
Counterexample-Guided Instantiation



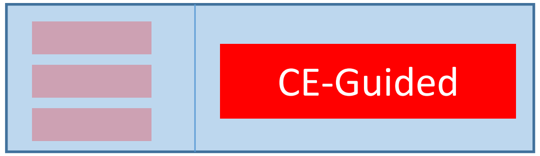


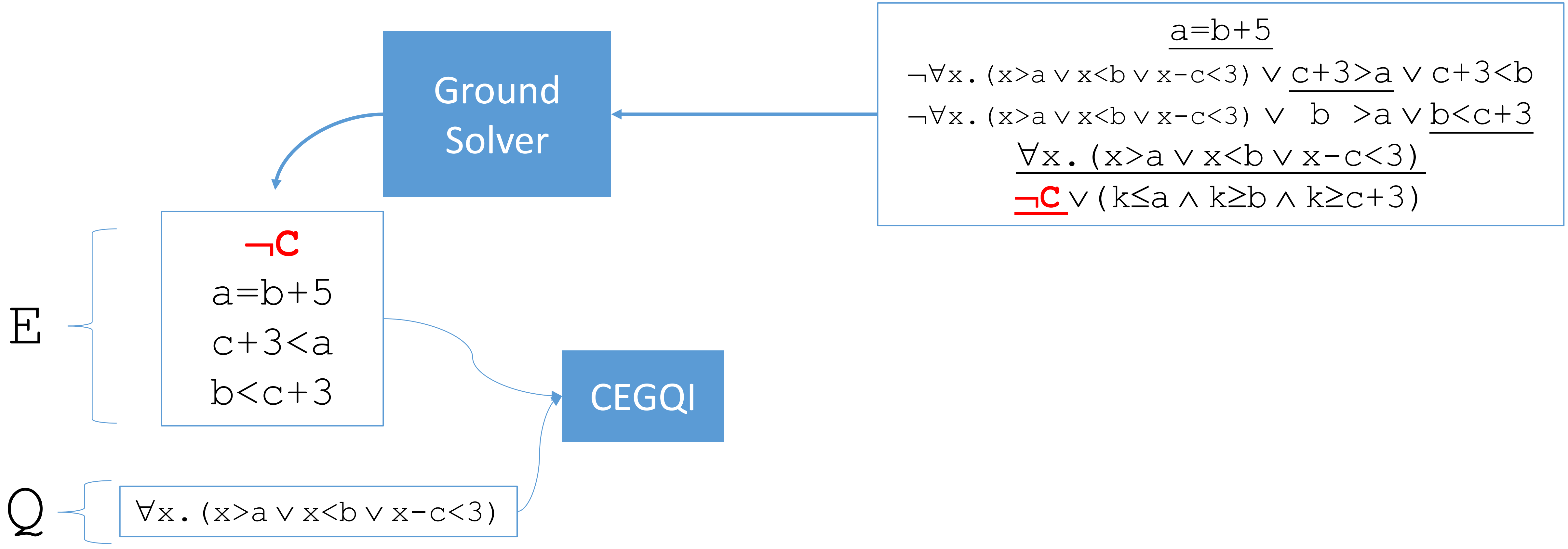
Counterexample-Guided Instantiation



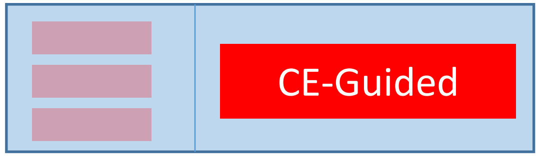


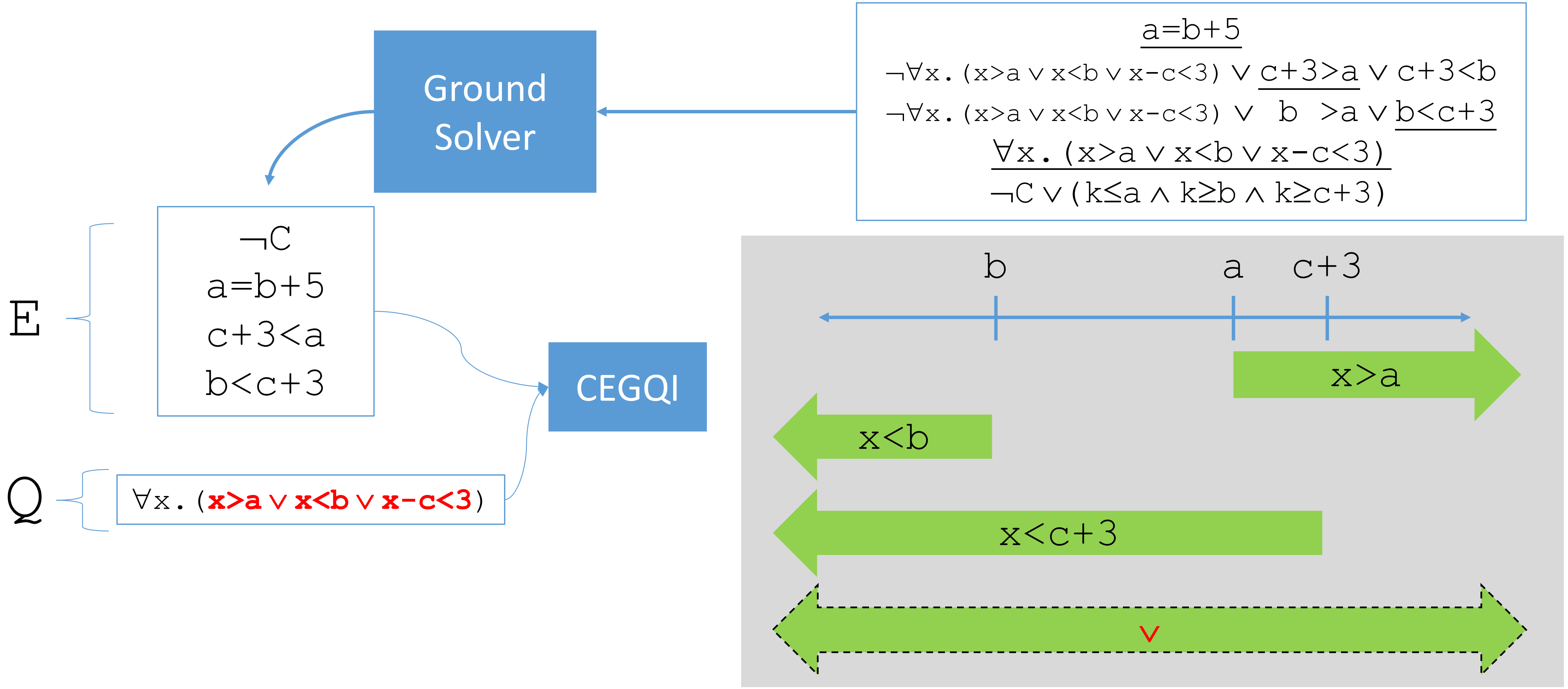
Counterexample-Guided Instantiation





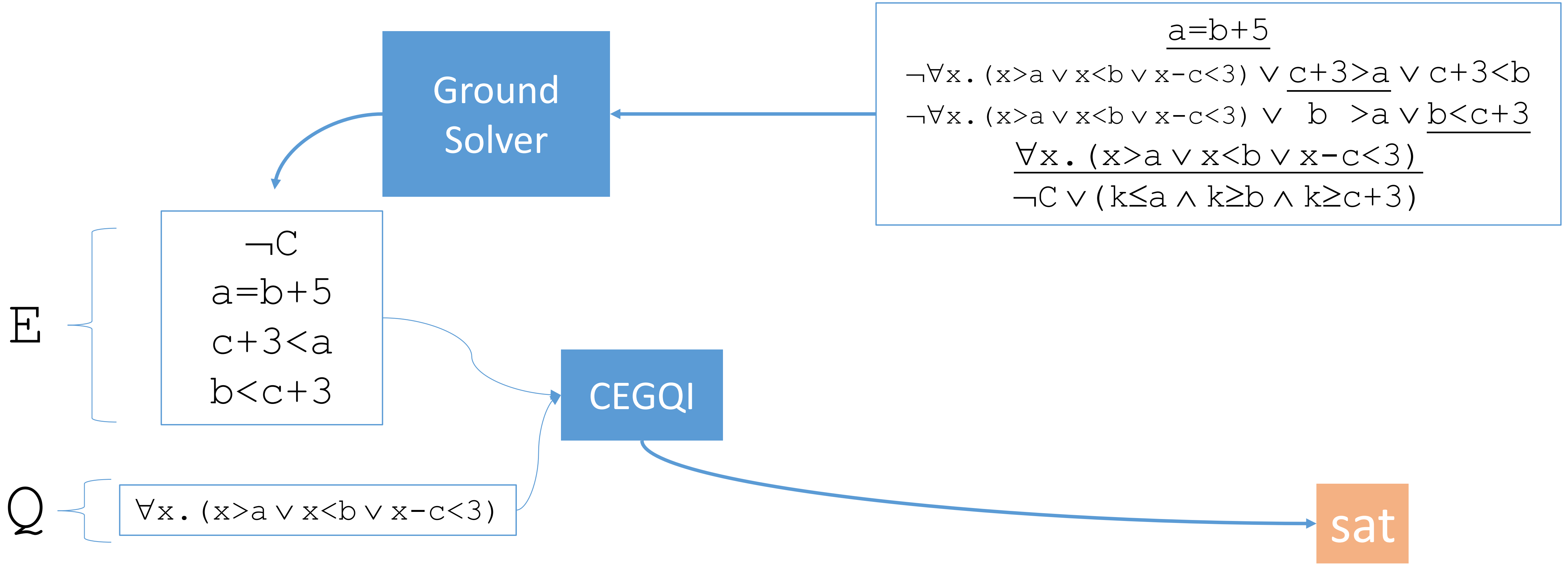
Counterexample-Guided Instantiation





Counterexample-Guided Instantiation

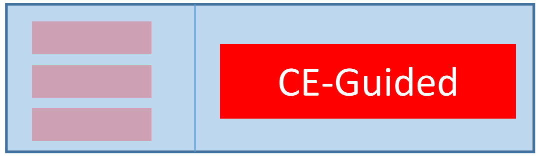
CE-Guided



$\Rightarrow \exists abc.(a=b+5 \wedge \forall x.(x>a \vee x<b \vee x-c<3))$

is LIA-satisfiable

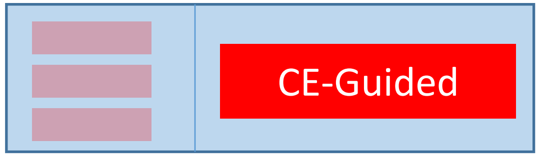
Counterexample-Guided Instantiation



- 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 δ, ∞*
    - 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 — $F[k] \;\rightarrow \{x \rightarrow k^M\}$
  - Datatypes, …

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

Counterexample-Guided Instantiation



$$\forall \mathbf{x} . \psi[\mathbf{x}]$$

- Can be used for:
  - Quantifier elimination

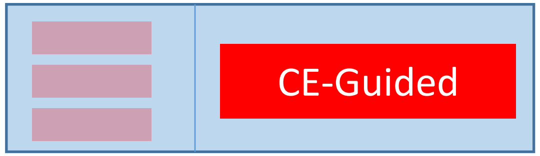
    $$\psi[\mathtt{t}_1] \wedge \ldots \wedge \psi[\mathtt{t}_n] \text{ is (un)sat}$$

    - $\exists \mathbf{x} . \neg\psi[\mathbf{x}]$ **is equivalent to** $\neg\psi[\mathtt{t}_1] \vee \ldots \vee \neg\psi[\mathtt{t}_n]$
  - Function Synthesis

    $$\psi[\mathtt{t}_1] \wedge \ldots \wedge \psi[\mathtt{t}_n] \text{ is unsat}$$

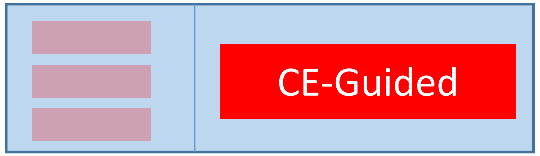
    - $\lambda \mathbf{x}.\mathtt{ite}(\psi[\mathtt{t}_1], \mathtt{t}_1, \ldots, \mathtt{ite}(\psi[\mathtt{t}_{n-1}], \mathtt{t}_{n-1}, \mathtt{t}_n) \ldots)$ **is a solution for** $\mathtt{f}$ **in** $\forall \mathbf{x} . \psi[\mathtt{f}(\mathbf{x})]$

Counterexample-Guided Instantiation

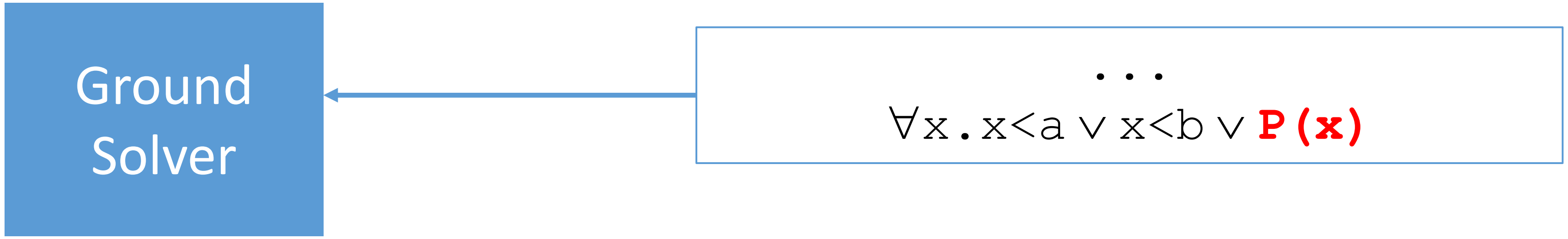


- Challenge:

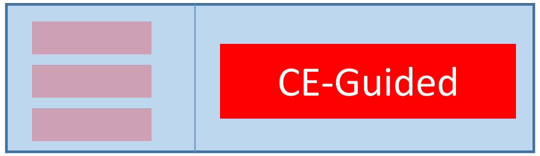
Counterexample-Guided Instantiation



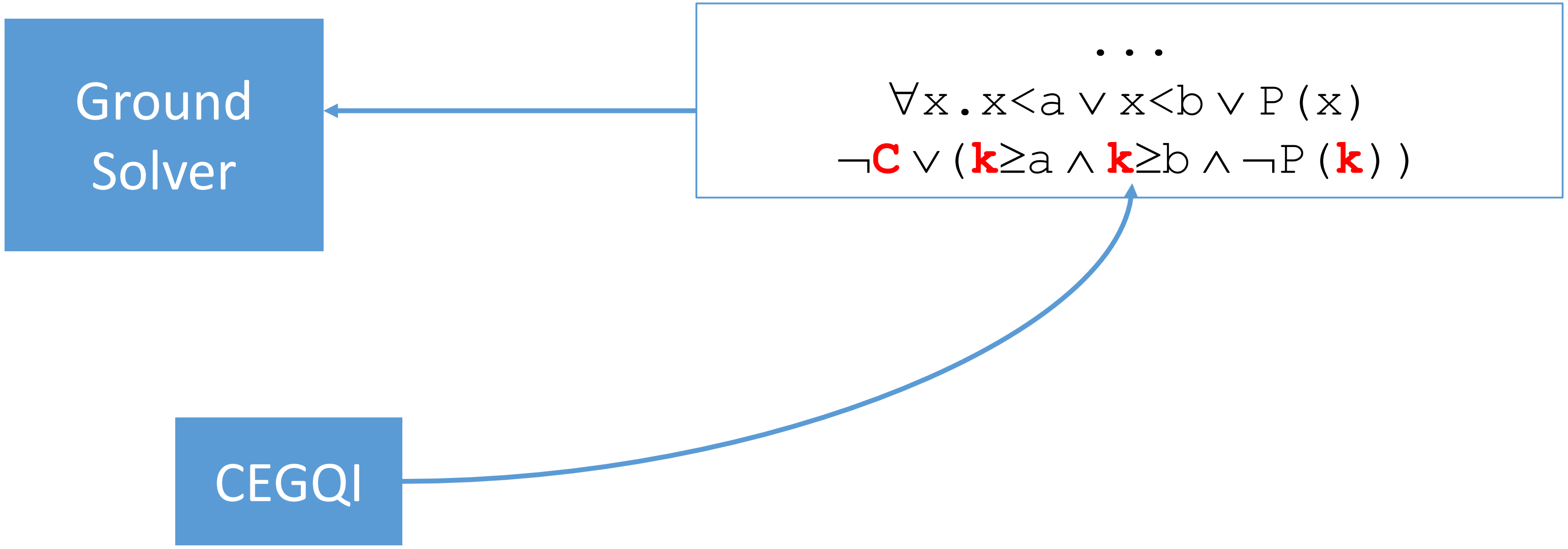
- Challenge: does not work in presence of uninterpreted functions!



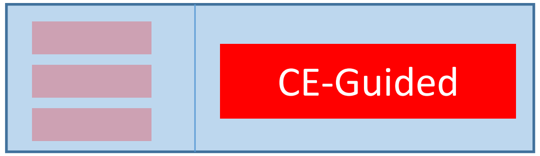
Counterexample-Guided Instantiation



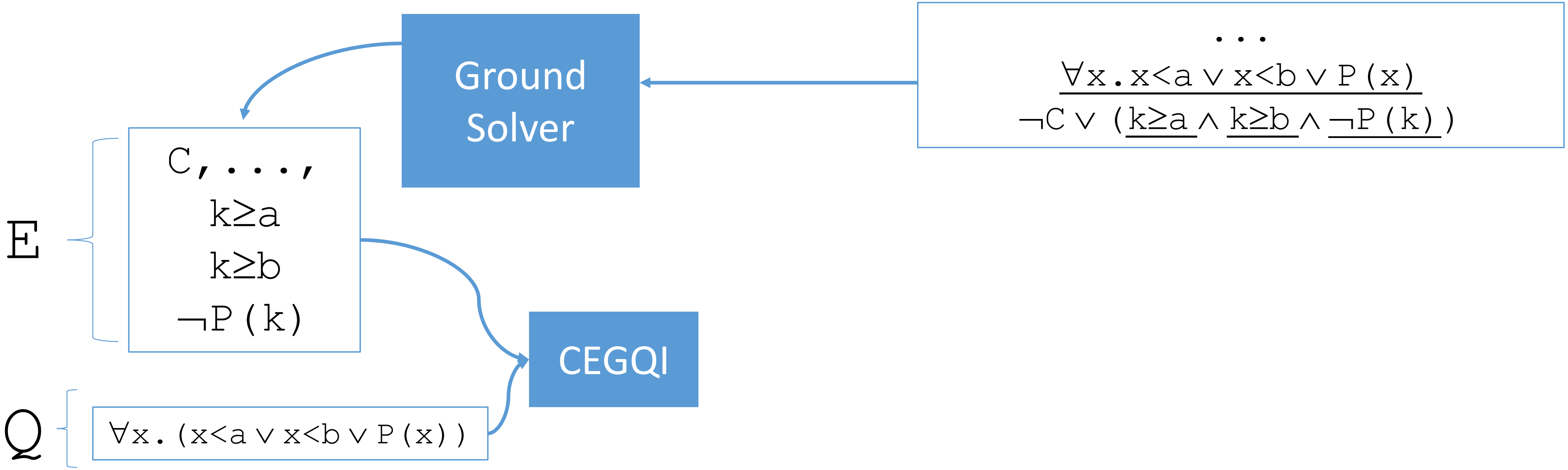
- Challenge: does not work in presence of uninterpreted functions!



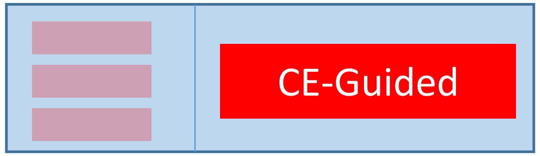
Counterexample-Guided Instantiation



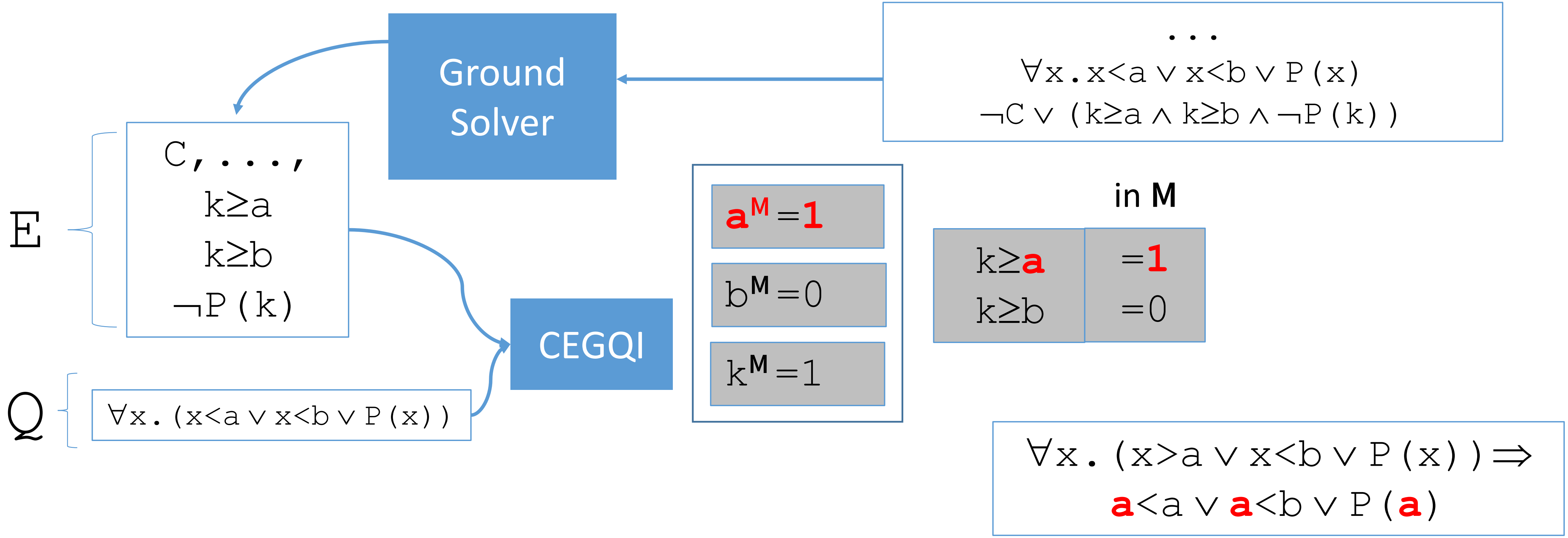
- Challenge: does not work in presence of uninterpreted functions!



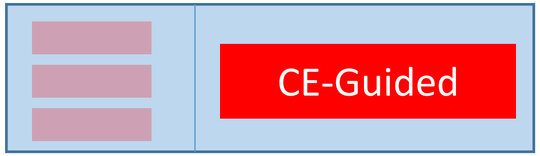
Counterexample-Guided Instantiation



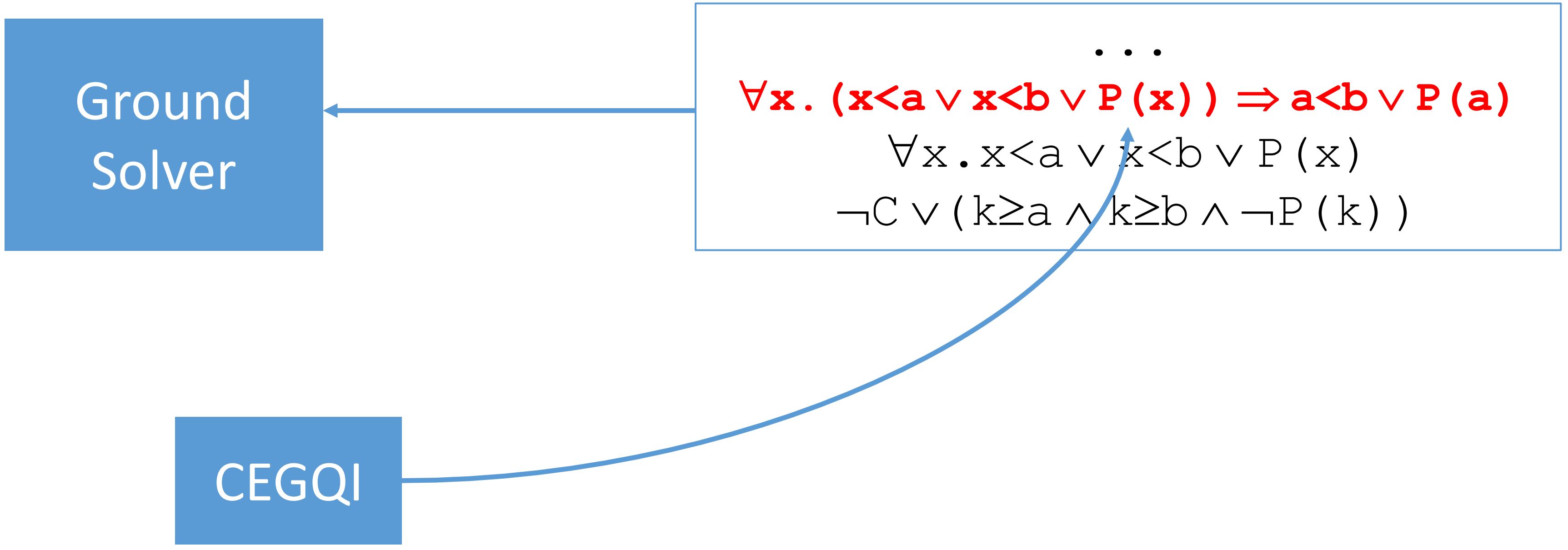
- Challenge: does not work in presence of uninterpreted functions!



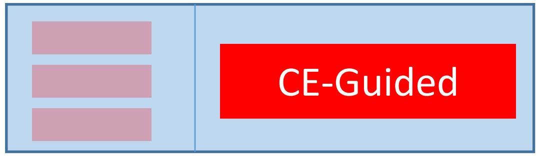
Counterexample-Guided Instantiation



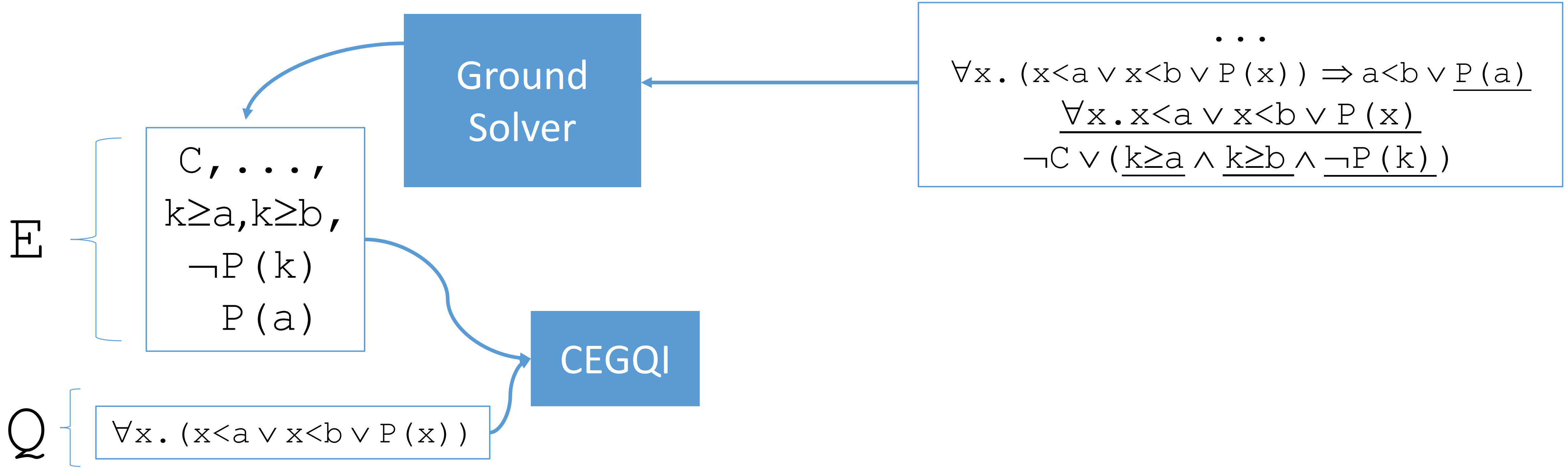
- Challenge: does not work in presence of uninterpreted functions!



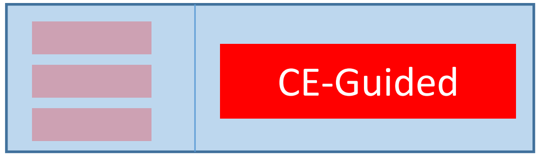
Counterexample-Guided Instantiation



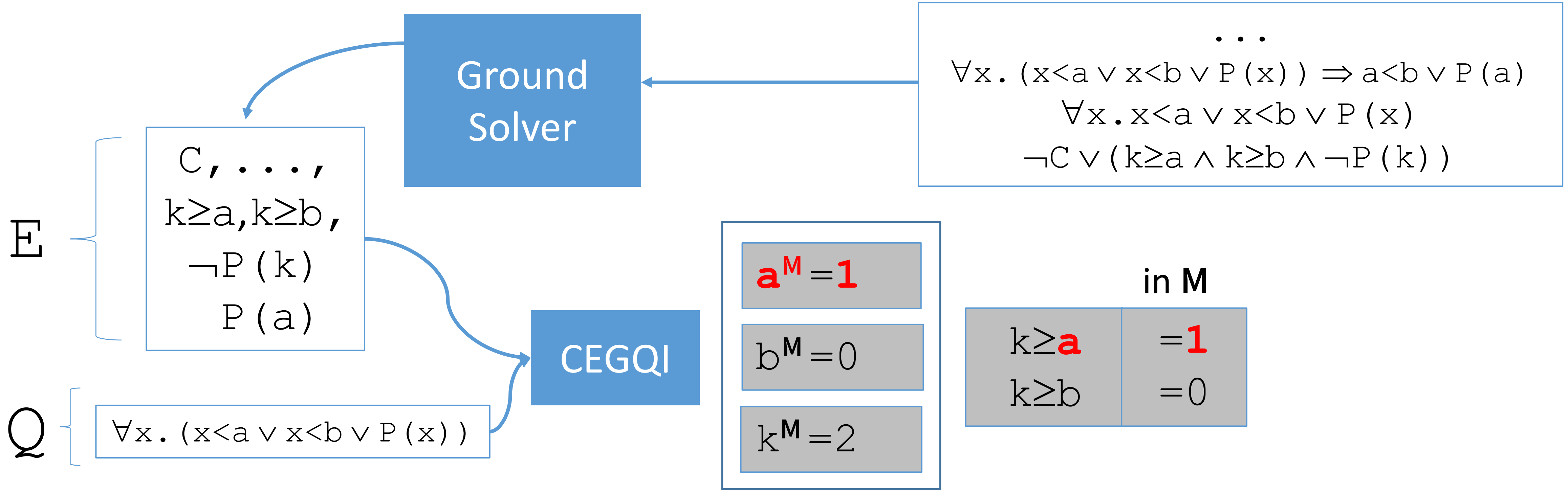
- **Challenge: does not work in presence of uninterpreted functions!**



Counterexample-Guided Instantiation



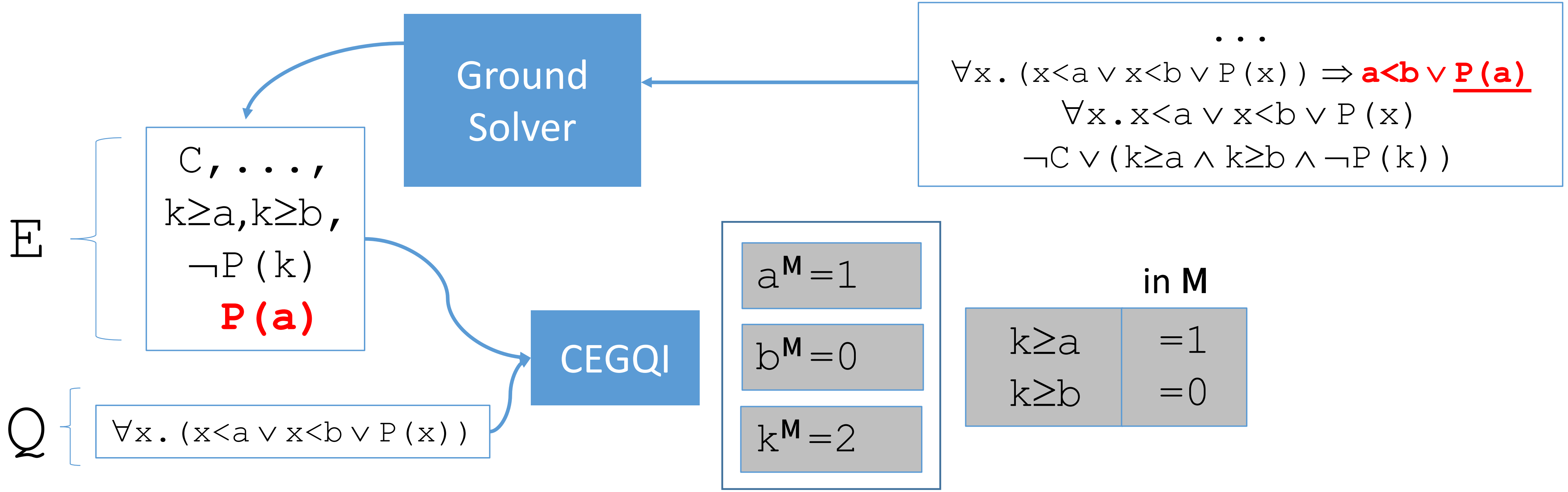
- Challenge: does not work in presence of uninterpreted functions!



$\Rightarrow$ a is still the maximal lower bound in **M** !

Counterexample-Guided Instantiation

- Challenge: does not work in presence of uninterpreted functions!



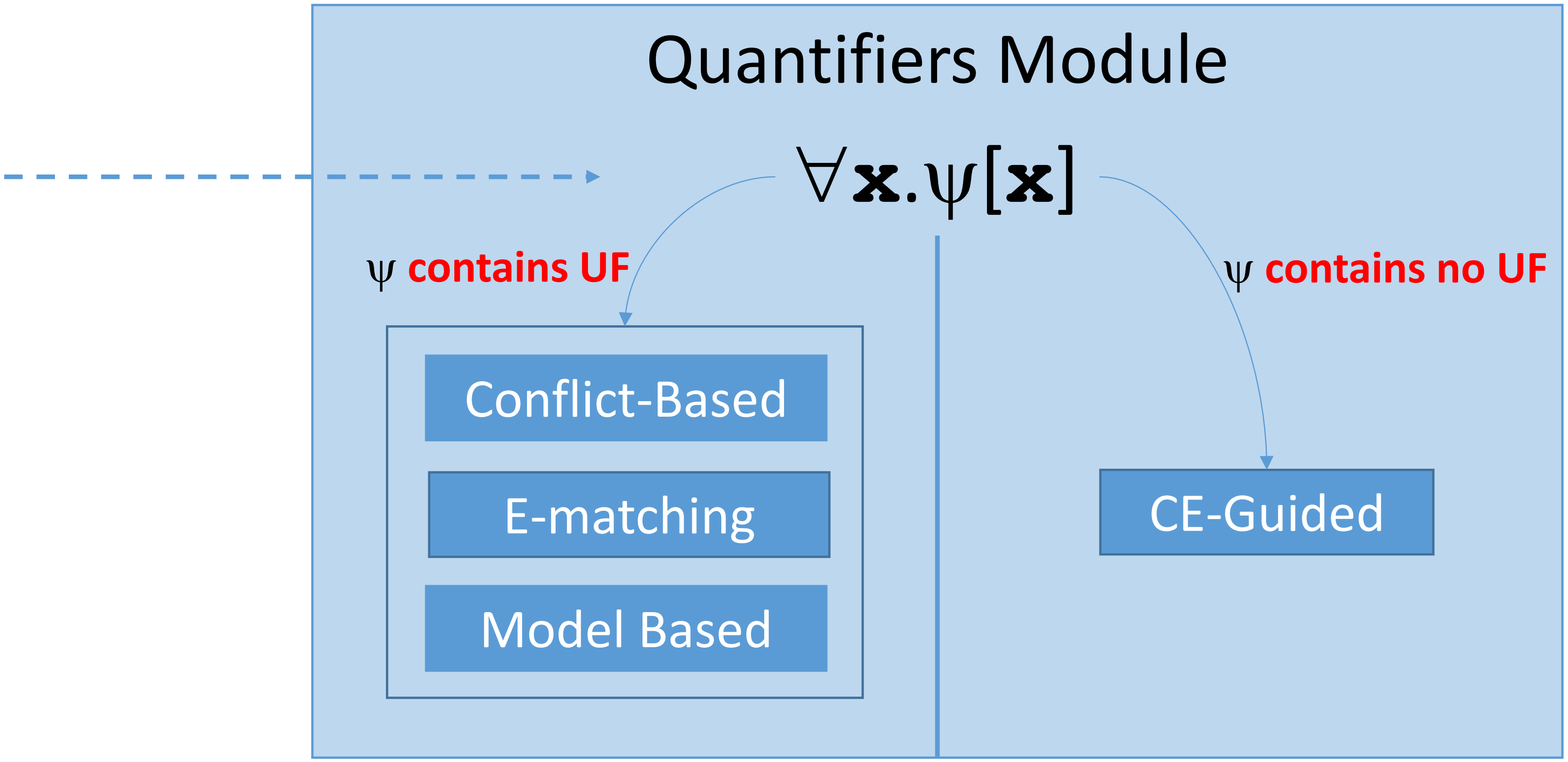
⇒ *Unlike the pure arithmetic case:*

- *Instance does not suffice to rule out* a *as maximal lower bound*

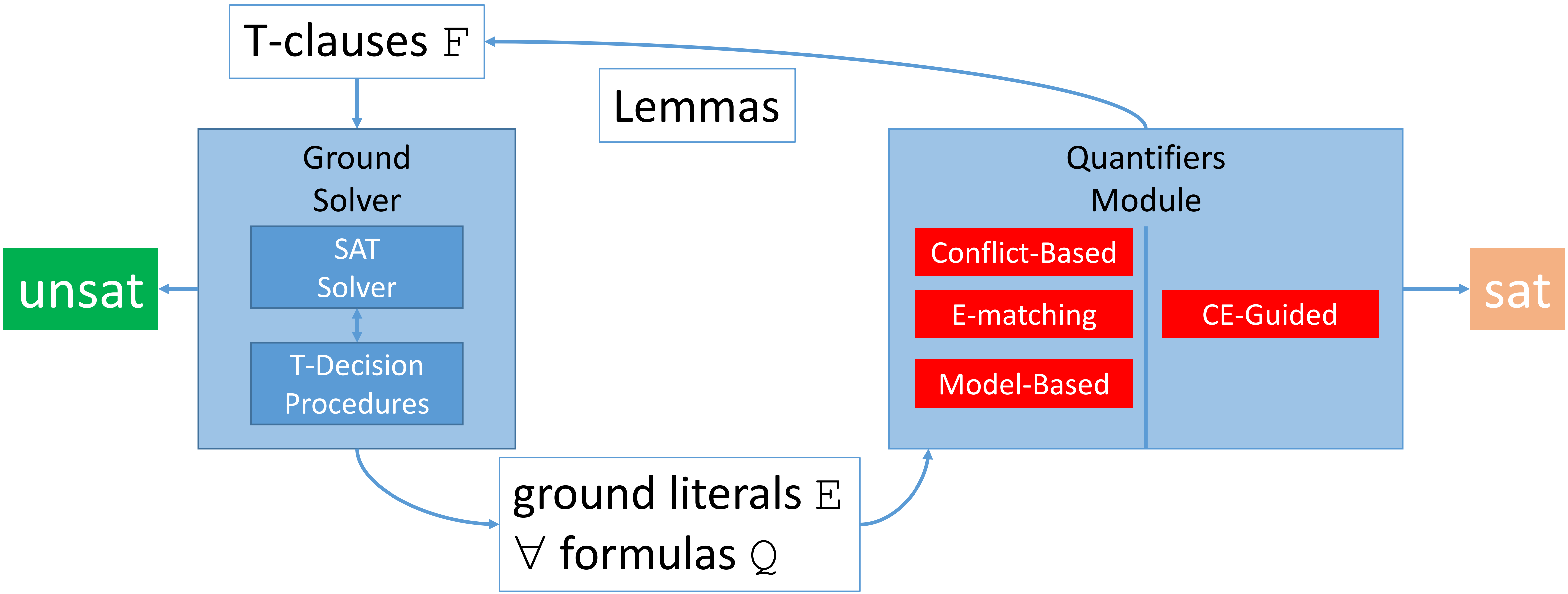
Summary

- SMT solvers handle quantifiers+theories via combination of:
  - DPLL(T)-based ground solver
  - Instantiation via:
    - **Conflict-based, E-matching, Model-Based Instantiation**
      - Effective in practice for ∀+UF, ∀+UFLIA, ∀+UFLRA, …
      - Can be decision procedure for limited fragments, e.g. Bernays-Shonfinkel
      - Conflict-Based, E-matching are useful for “unsat”
      - Model-Based is useful for “sat”
    - **Counterexample-guided Instantiation**
      - Decision procedure for ∀+LRA, ∀+LIA, ∀+BV, …

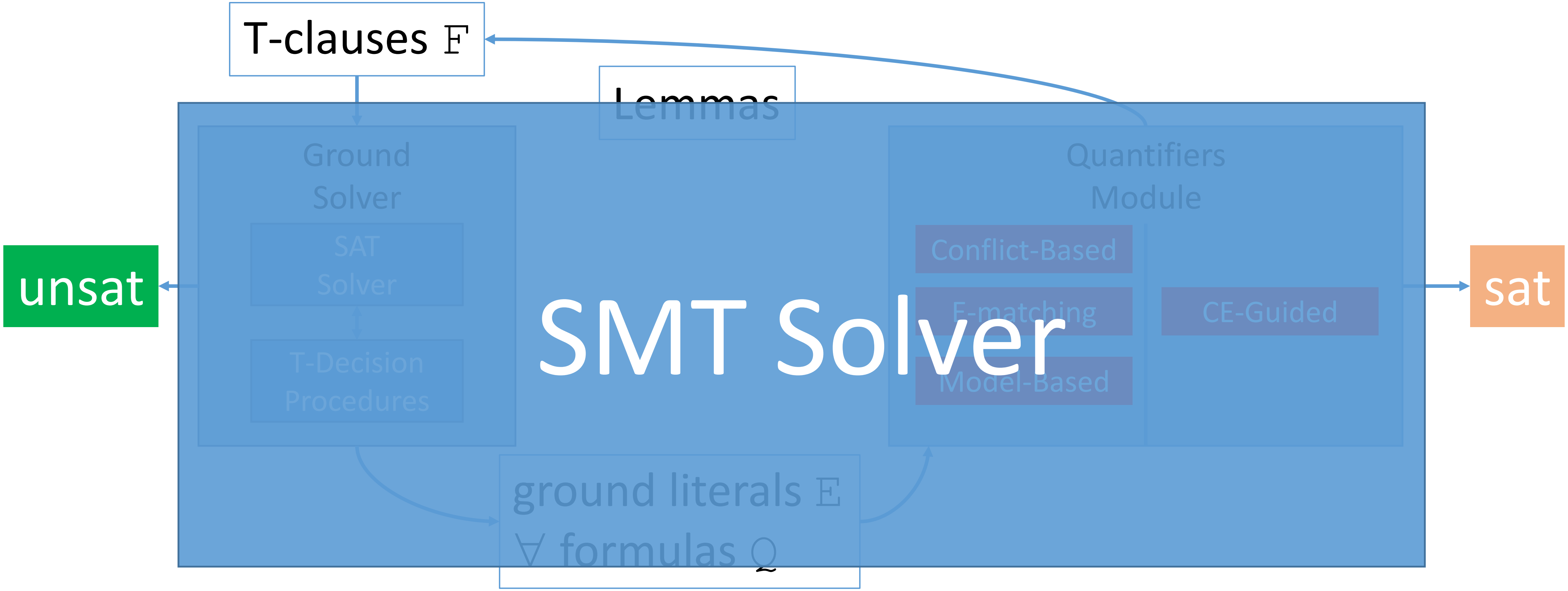
In practice: Distribute ∀ to proper strategy



Summary: DPLL(T)+Instantiation



Summary: DPLL(T)+Instantiation



Other Important Aspects of $\forall$ Not Covered

- Eager Quantifier Instantiation
- Relevancy
- Preprocessing
- Rewriting

Future Challenges

- Improve performance and precision of existing approaches
    - Many engineering challenges when implementing E-matching, conflict-based instantiation
- Develop new approaches for ∀+UF+theories that:
    - Are efficient in practice
        - E-matching is efficient for ∀+UF, ce-guided approaches are efficient for ∀+ theories
            - Under what conditions, and to what degree, can these techniques be combined?
    - Are decision procedures for various fragments
        - Extensions of Bernays-Shonfinkel
        - Array Property fragments
        - Local theory extensions
        - ∀ over pure theories that emit quantifier elimination

Thanks for listening

- ….Questions?