Recent Advances in Instantiation-Based Techniques and their Implementation in CVC4

Andrew Reynolds

July 2, 2016

THE UNIVERSITY OF IOWA

Outline

- CVC4
- SMT solver architecture

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

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

- Recent strategies for quantifier instantiation in CVC4:
  - E-matching, conflict-based, model-based, counterexample-guided
- Challenges, future work

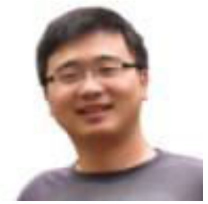
CVC4: Past and Present Team Members









Clark Barrett (NYU)
Cesare Tinelli (U Iowa)
Morgan Deters (NYU)

Kshitij Bansal (Google)
François Bobot (CEA)
Chris Conway (Google)

Liana Hadarean (Mentor Graphics)
Dejan Jovanović (SRI)
Tim King (Google)

Tianyi Liang (Two Sigma)
Andrew Reynolds (U Iowa)
Nestan Tsiskaridze (U Iowa)
Martin Brain (U Oxford)
Guy Katz (Stanford)
Paul Meng (U Iowa)

CVC4: Past and Present Support

CVC4 is Expressive and Featureful

- Boolean combinations of theory constraints
  - UF, Arrays
  - Linear real/integer arithmetic
  - Bitvectors
  - (Co)inductive datatypes
  - Strings
  - Sets with Cardinality
- Mixed constraints over all built-in theories
- Quantifiers $\forall$
- Models, proofs, unsat cores

CVC4 is Expressive and Featureful

- Boolean combinations of theory constraints
  - UF, Arrays
  - Linear real/integer arithmetic
  - Bitvectors
  - (Co)inductive datatypes
  - Strings
  - Sets with Cardinality
- Mixed constraints over all built-in theories
- Quantifiers ∀ ⇒ Focus of this talk
- Models, proofs, unsat cores

Approaches for Satisfiability of $\forall$ in Tools

- First order theorem provers focus on $\forall$ 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 $\forall$ 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, iProver**
    - First-order resolution + superposition **[Robinson 65, Nieuwenhuis/Rubio 99]**
    - AVATAR in Vampire **[Voronkov 14, Reger et al 15]**
    - Some instantiation-based **[Ganzinger/Korovin 03]**

- SMT solvers focus mostly on ground theory reasoning

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

Approaches for Satisfiability of $\forall$ in Tools

- First order theorem provers focus on $\forall$ reasoning

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

  - **Vampire, E, SPASS, iProver**
    - First-order resolution + superposition **[Robinson 65, Nieuwenhuis/Rubio 99]**
    - AVATAR in Vampire **[Voronkov 14, Reger et al 15]**
    - Some instantiation-based **[Ganzinger/Korovin 03]**

- SMT solvers focus mostly on ground theory reasoning

  …but have been extended in the past decade to $\forall$ 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, …]**

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, iProver**
    - First-order resolution + superposition **[Robinson 65, Nieuwenhuis/Rubio 99]**
    - AVATAR in Vampire **[Voronkov 14, Reger et al 15]**
    - Some instantiation-based **[Ganzinger/Korovin 03]**
- 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, …]**

Quantified Formulas in DPLL(T): Basics

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

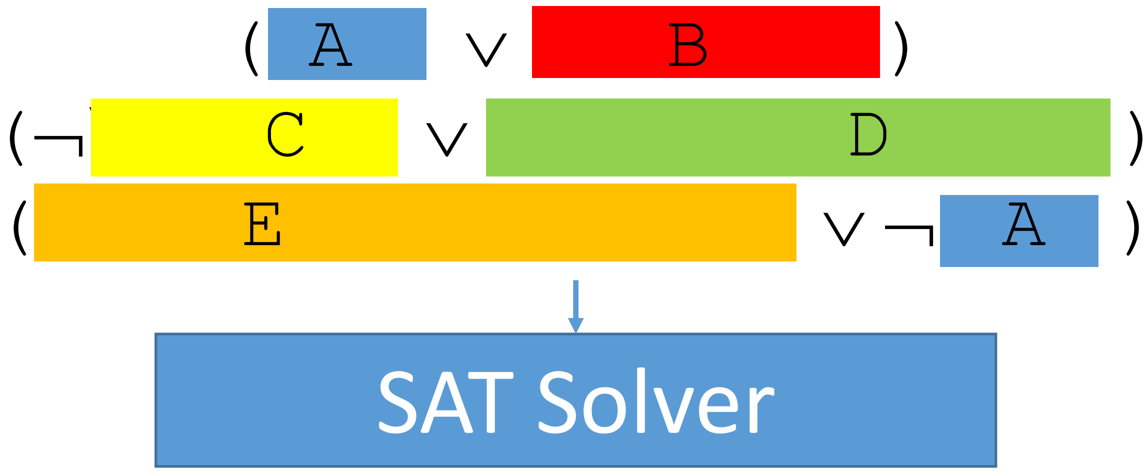
$\Rightarrow$ Given the above input

Quantified Formulas in DPLL(T): Basics

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

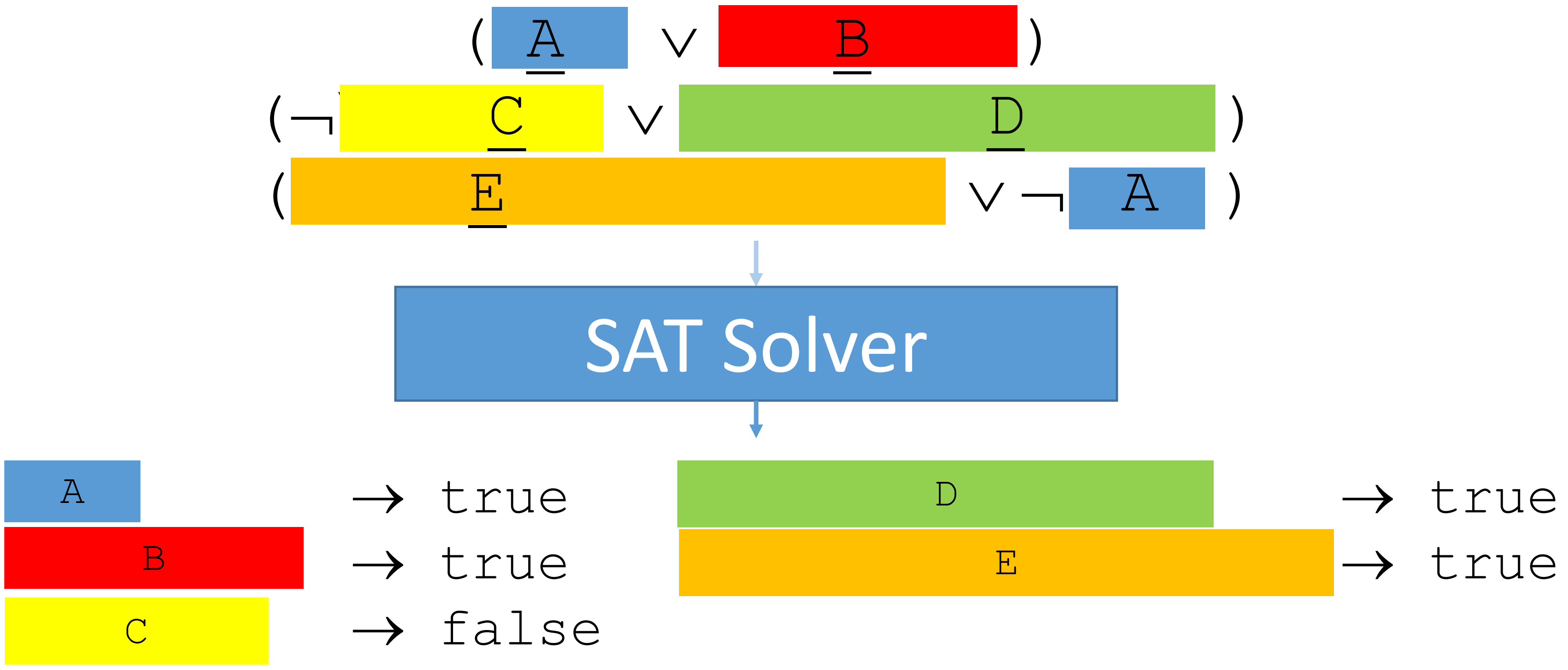
- Consider the propositional abstraction of the formula
  - Atoms may encapsulate quantified formulas with Boolean structure
    - E.g. $\forall y. \neg P(y) \lor R(y)$

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

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

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

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

↓

SAT Solver

↓

$P(a)$	$\rightarrow$ true	$\forall y. \neg P(y) \lor R(y)$	$\rightarrow$ 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

↓

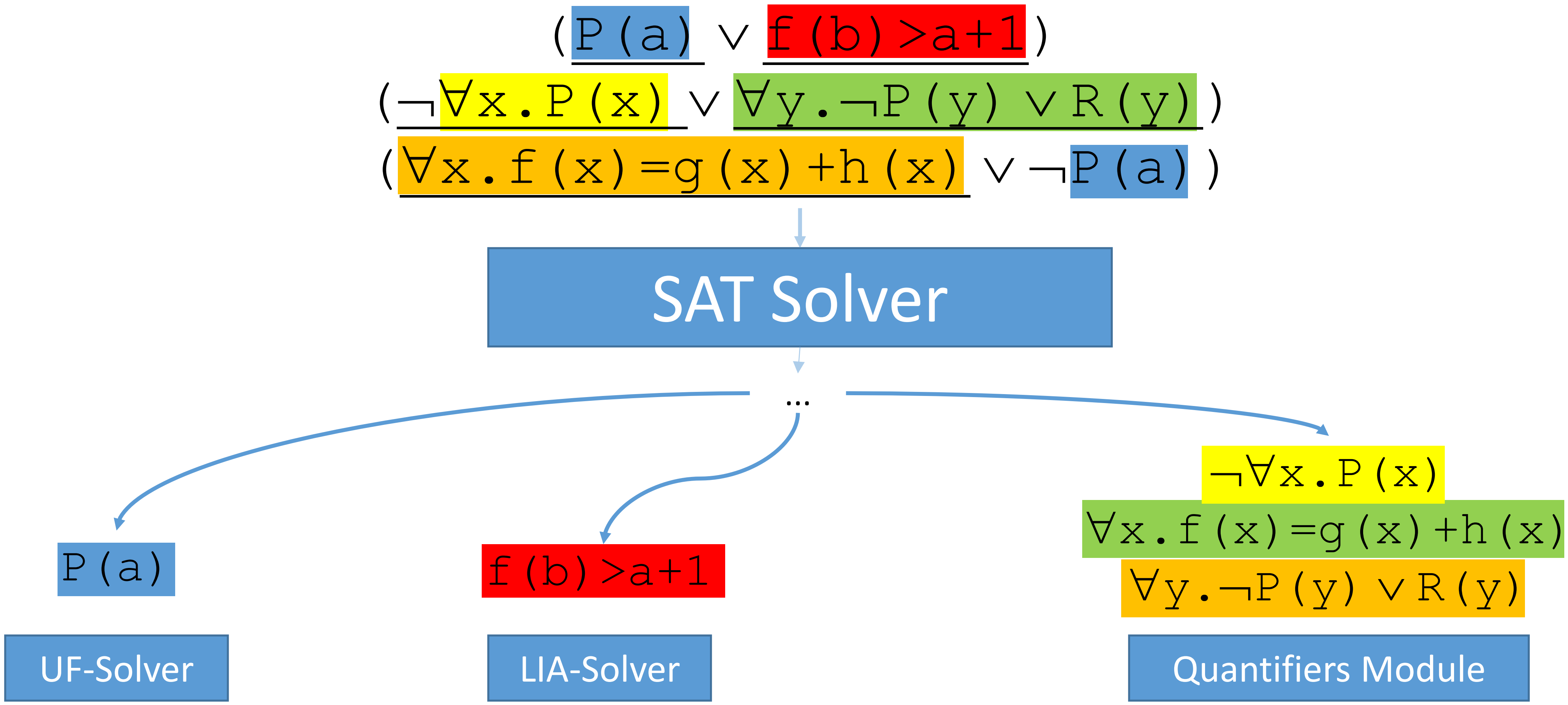
P(a), f(b)>a+1, ¬∀x.P(x), ∀x.f(x)=g(x)+h(x), ∀y.¬P(y) ∨ R(y)

M

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

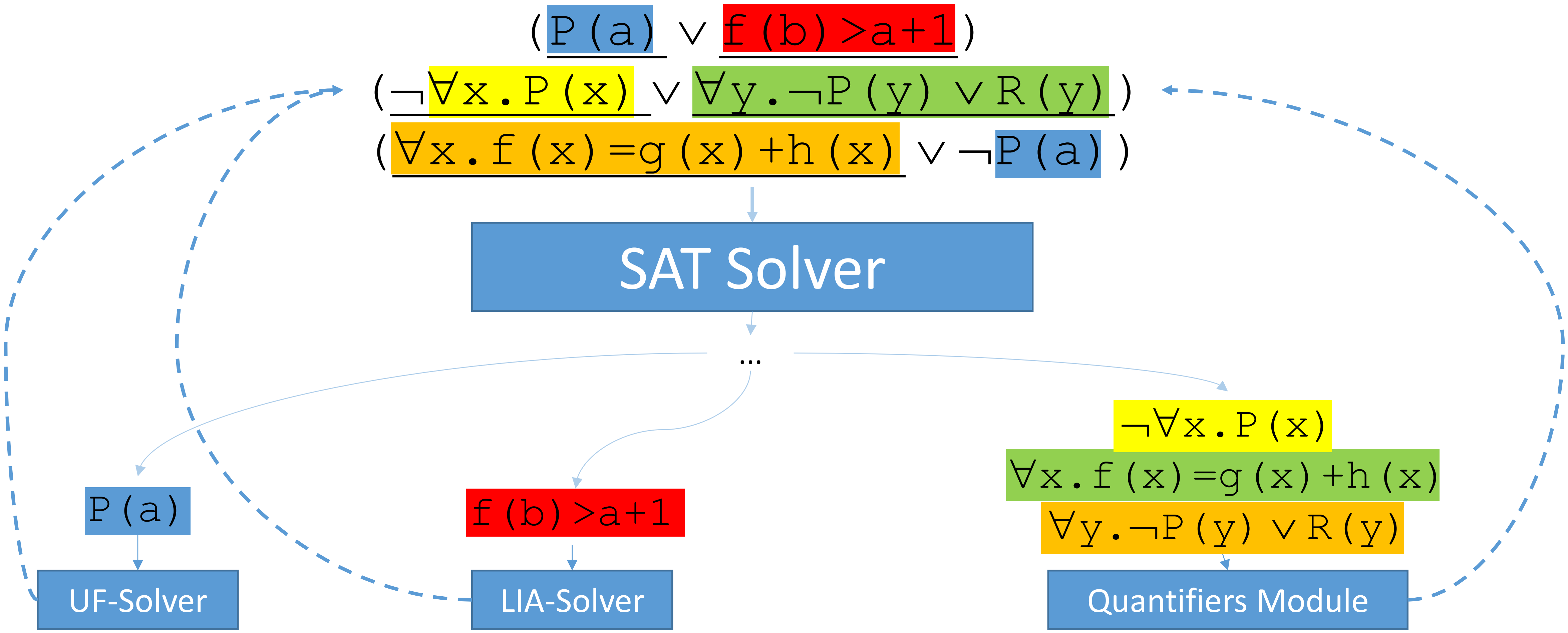
- Must check if M is T-satisfiable

Quantified Formulas in DPLL(T): Basics



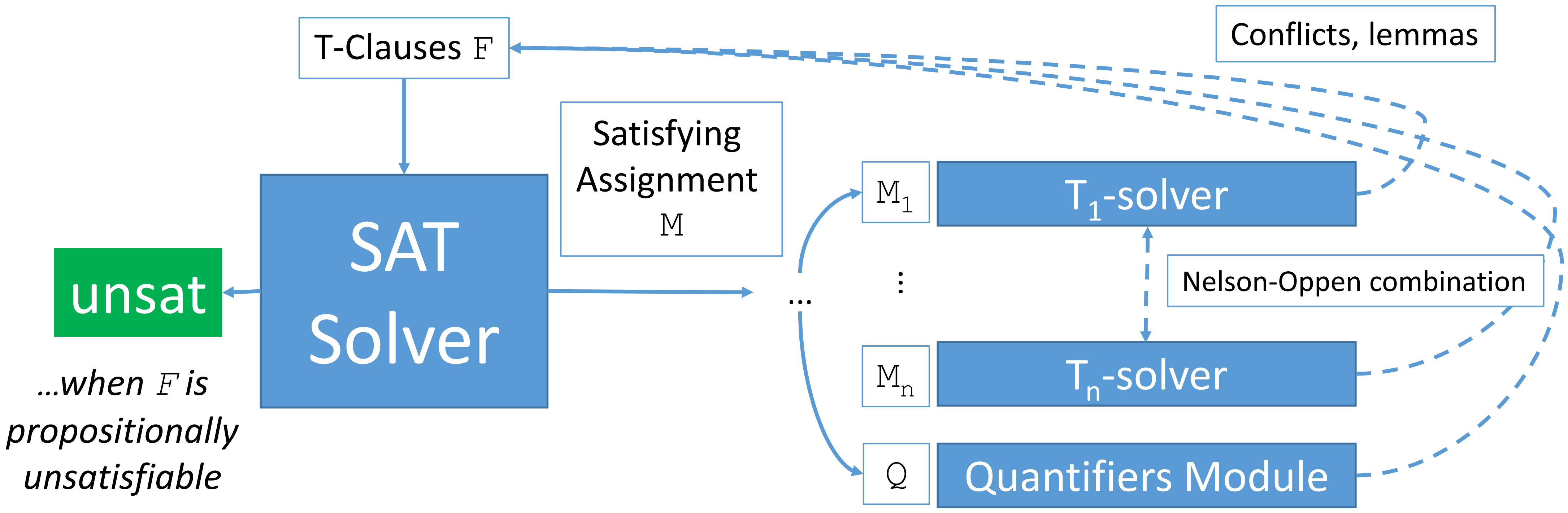
⇒ Distribute ground literals to T-solvers, ∀ literals to quantifiers module

Quantified Formulas in DPLL(T): Basics



⇒ These solvers may choose to add conflicts/lemmas to clause set

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

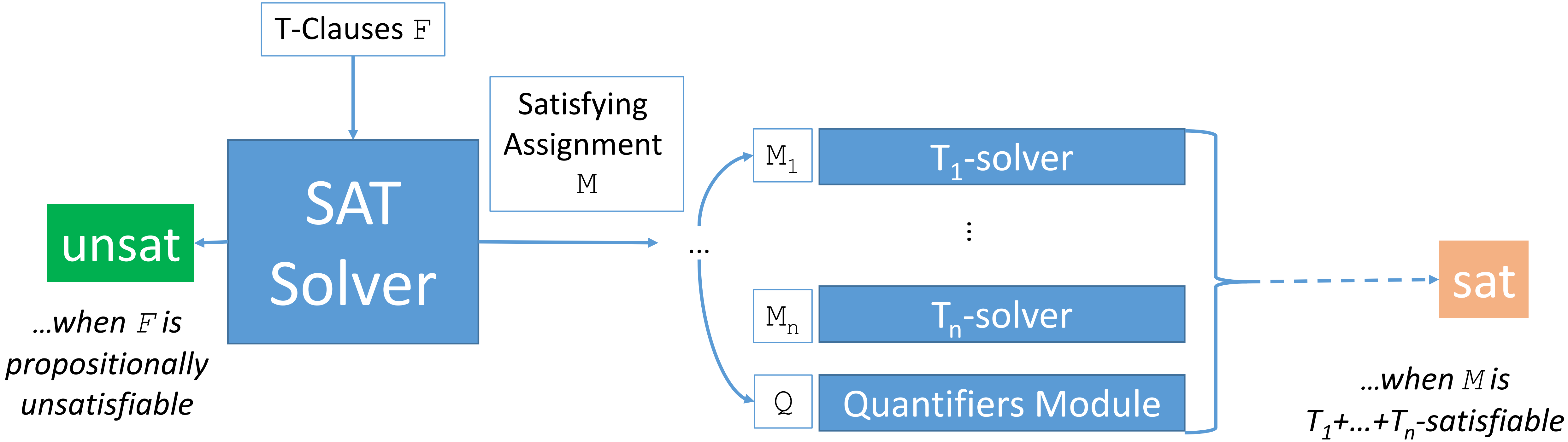


⇒ Each of these components may:

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

[Nieuwenhuis/Oliveras/Tinelli 06]

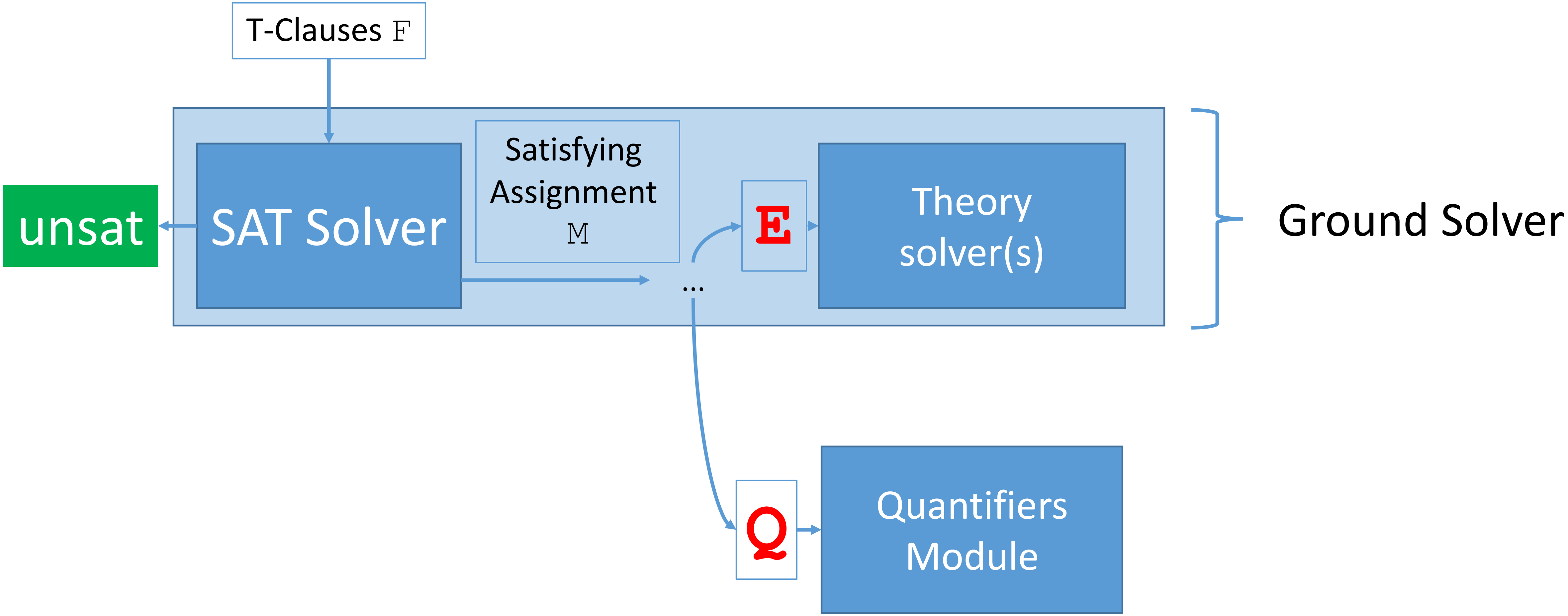
DPLL(T_1+..+T_n)+Quantifiers: Overview



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

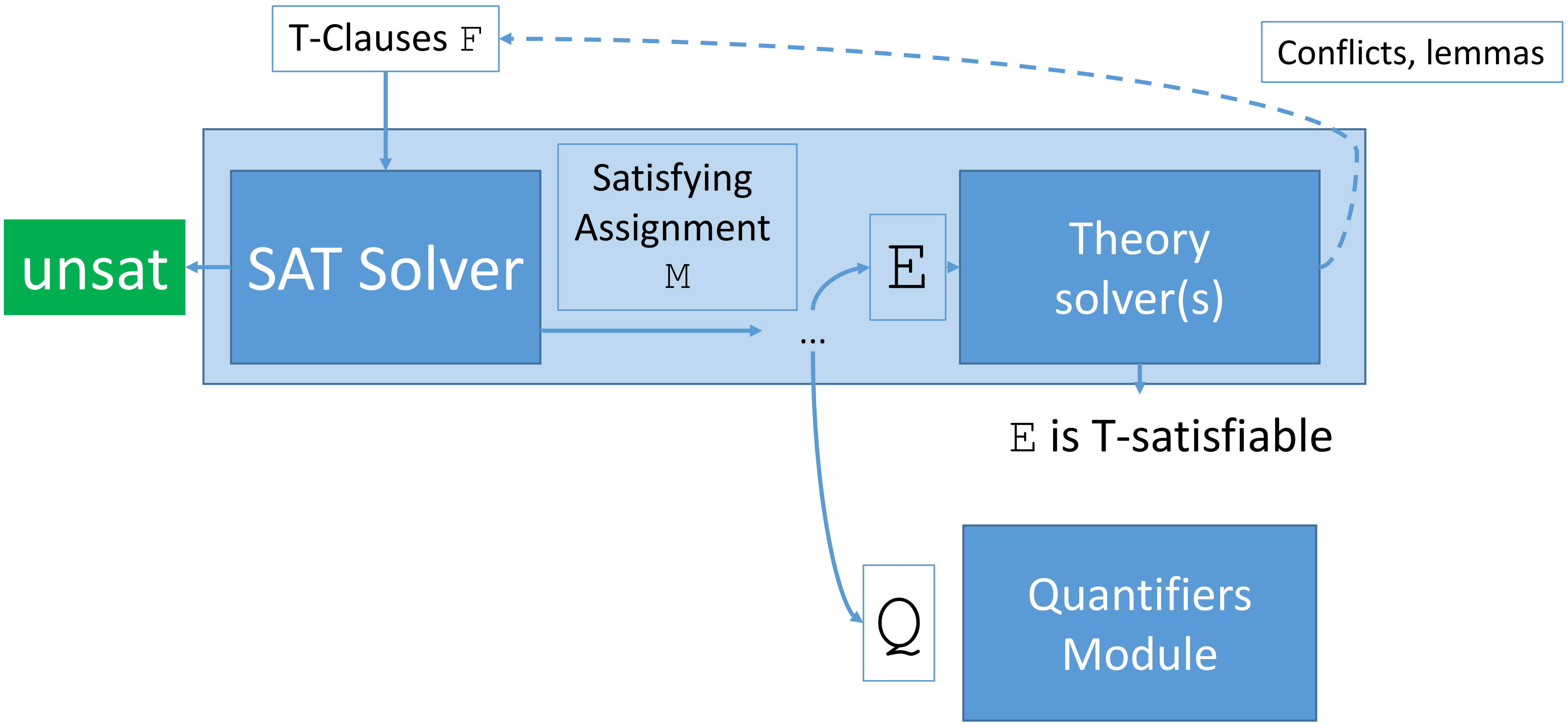
[Nieuwenhuis/Oliveras/Tinelli 06]

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



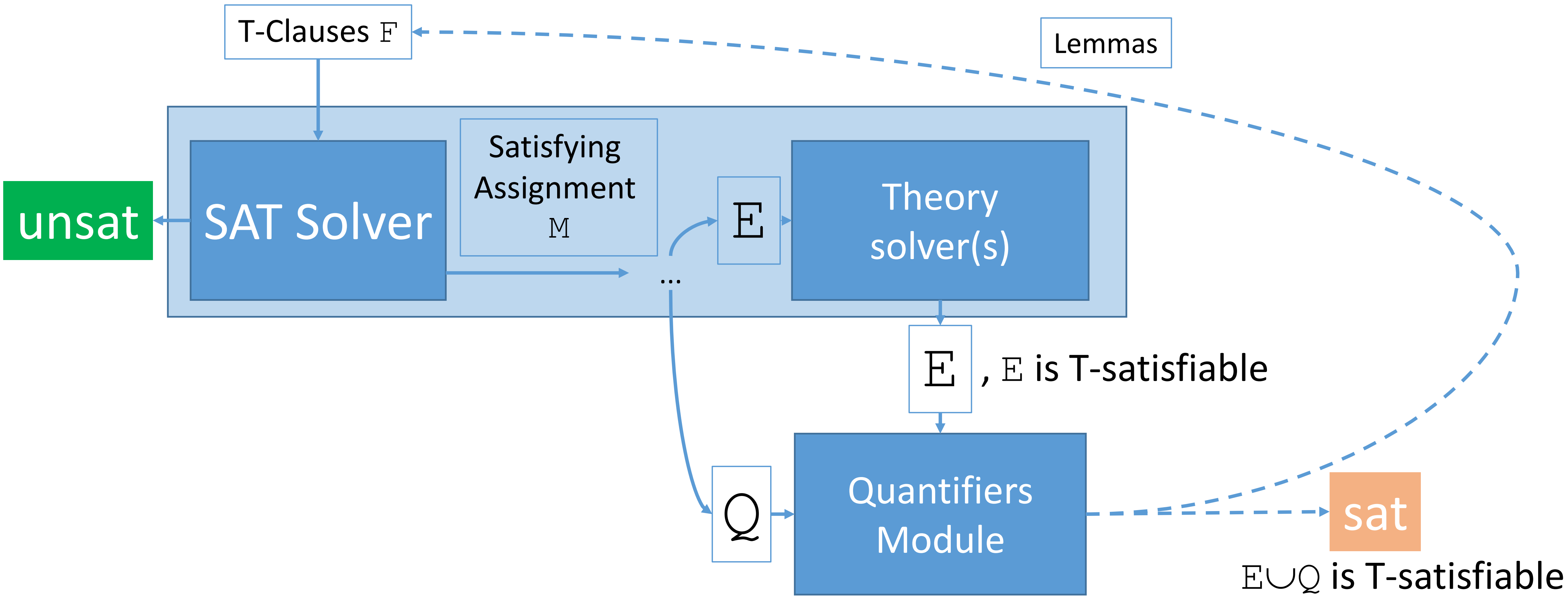
⇒ For purposes of this talk, partition M into quantifier-free part **E**, and set of ∀ formulas **Q**

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



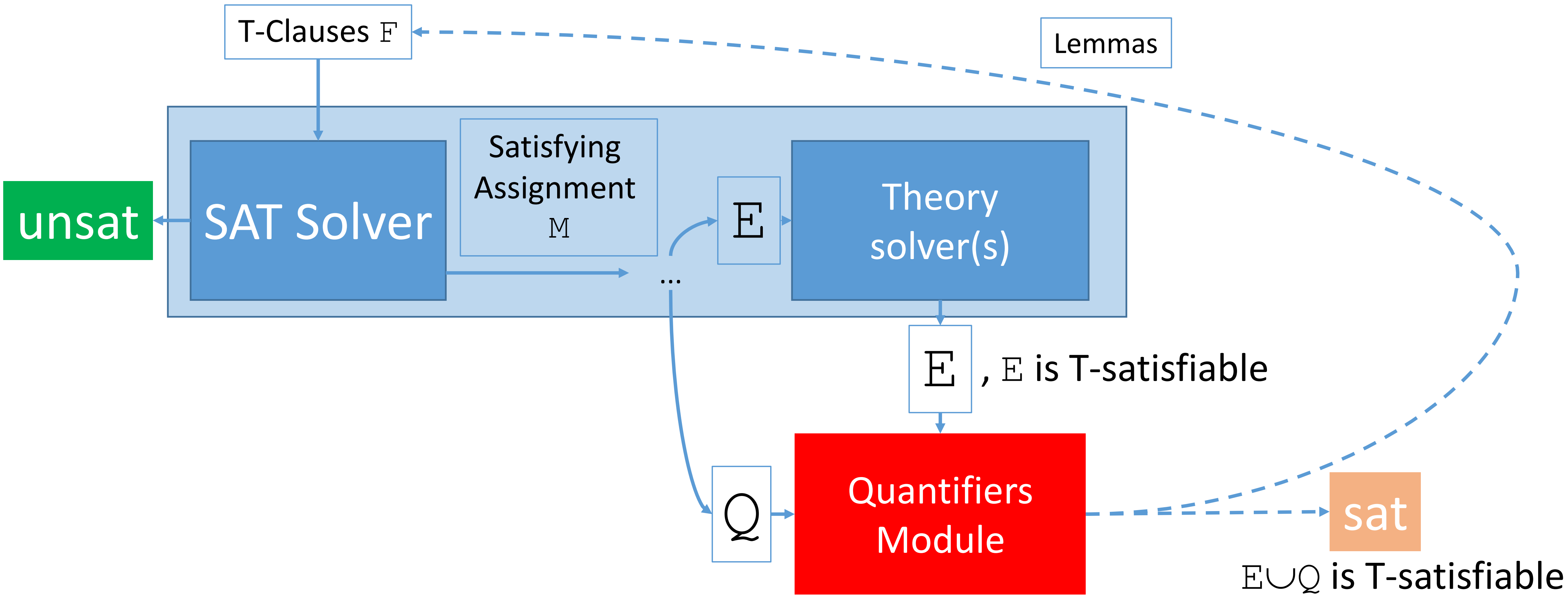
⇒ Theory solvers determine whether E is T-(un)satisfiable

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



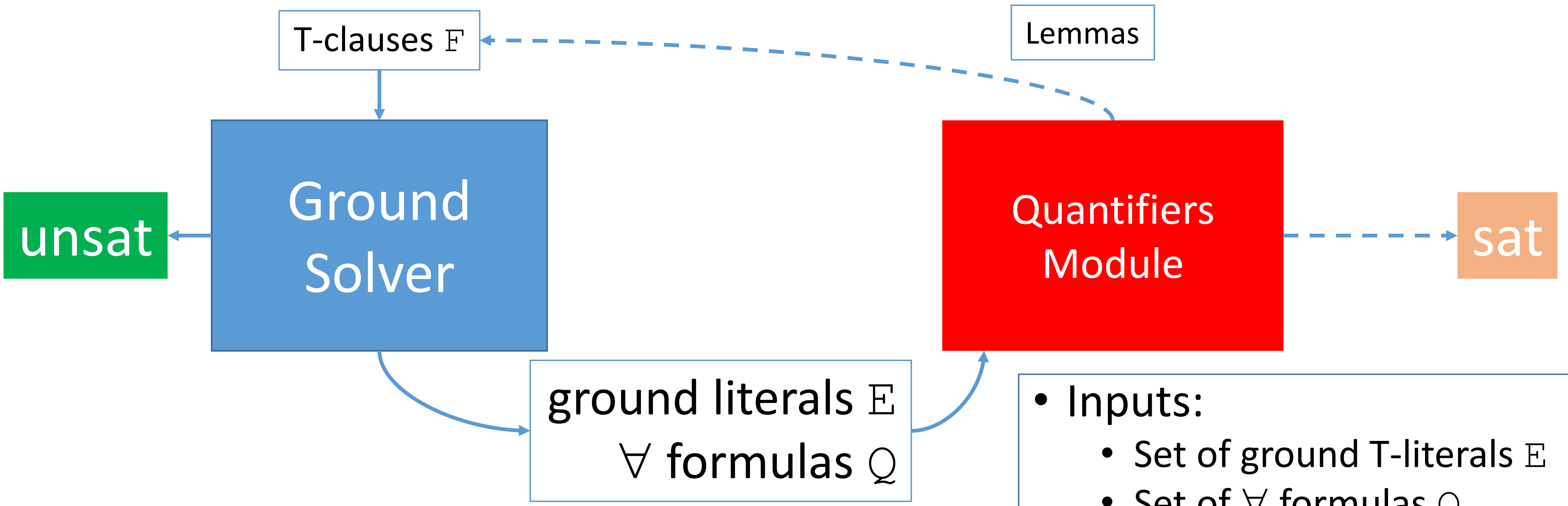
$\Rightarrow$ If E is T-satisfiable, quantifiers module may be invoked

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



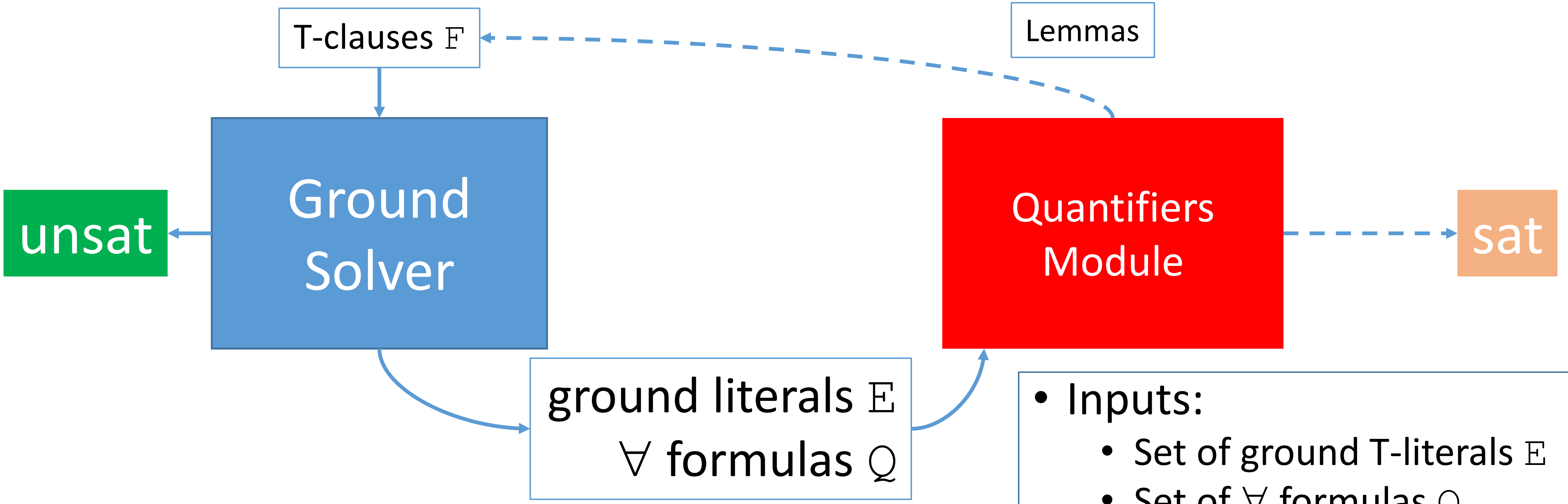
⇒ *Will discuss how the quantifiers module is implemented*

DPLL(T)+Quantifiers, further simplified



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

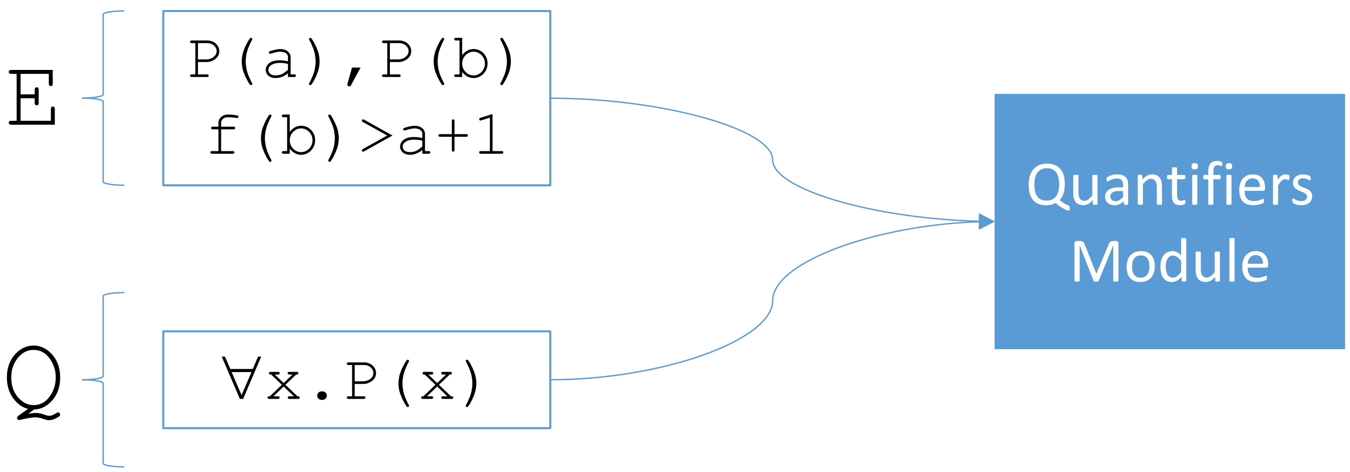
DPLL(T)+Quantifiers, further simplified



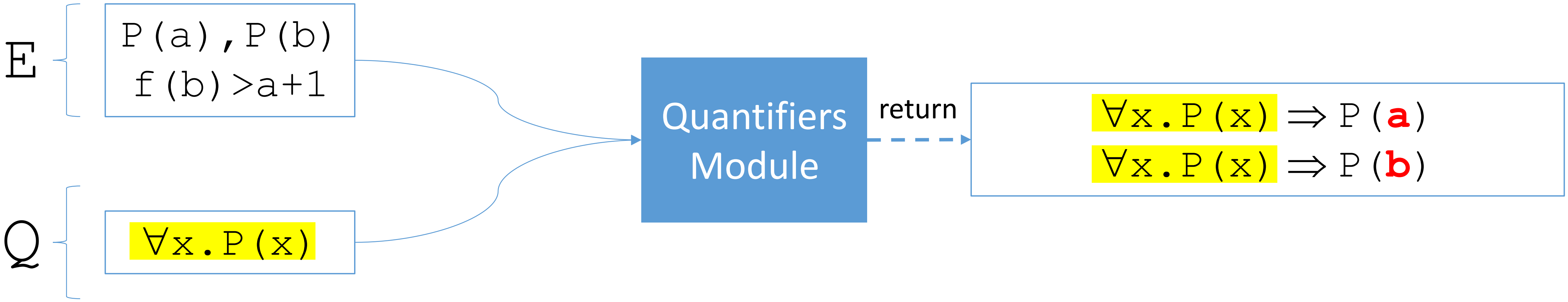
- Recurrent Questions:
  - Which lemmas do we add?
  - How do we know $\mathbb{E} \cup \mathbb{Q}$ is T-satisfiable?
  - When do we invoke it?

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

Quantifier Instantiation



Quantifier Instantiation



- Universal quantification handled by Instantiation
  - Choose ground term(s) **t**, lemma(s) say ∀x.P(x) implies P(**a**)

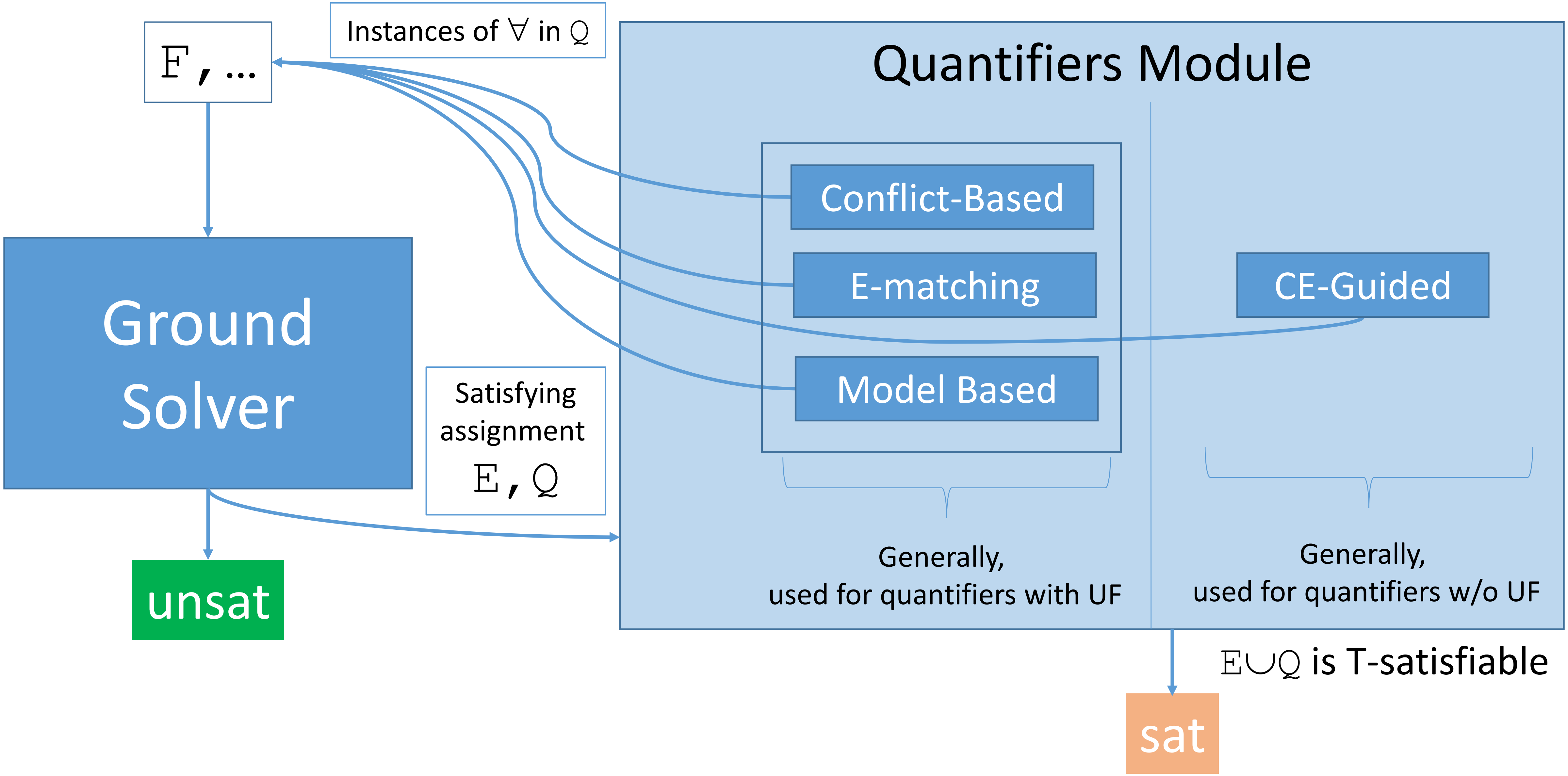
⇒May be applied ad infinitum, for $x \rightarrow$ a,b,c,d,…

- Selection of instances is the core challenge

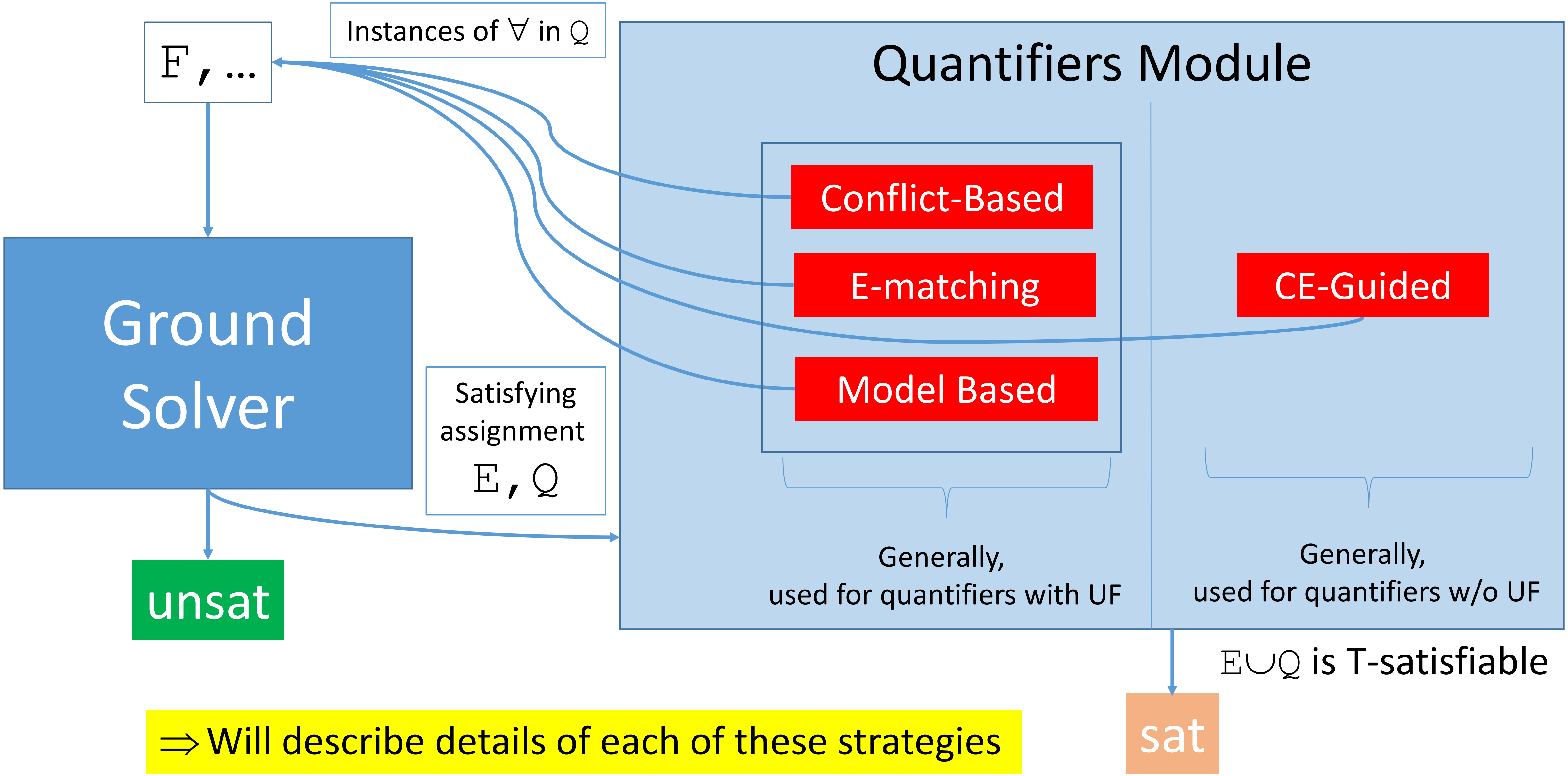
Quantifiers Module : Recurrent Question

- 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]**

Techniques for Quantifier Instantiation: Overview

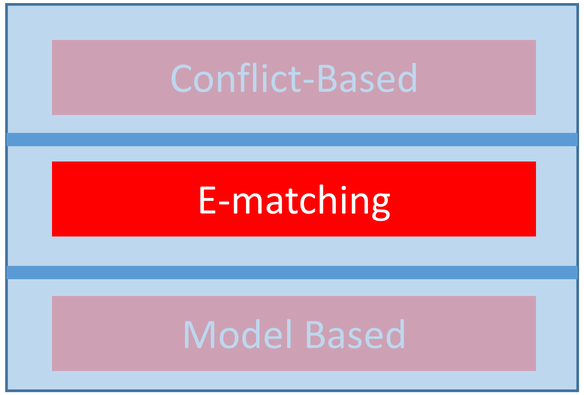


Techniques for Quantifier Instantiation: Overview



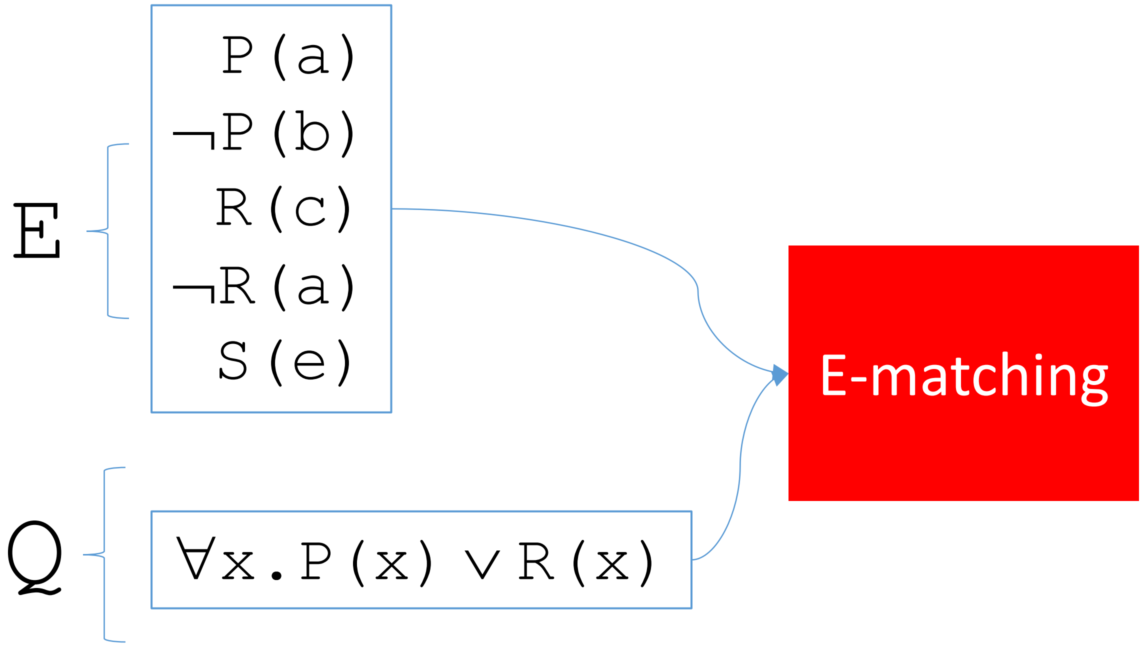
⇒ Will describe details of each of these strategies

E-matching

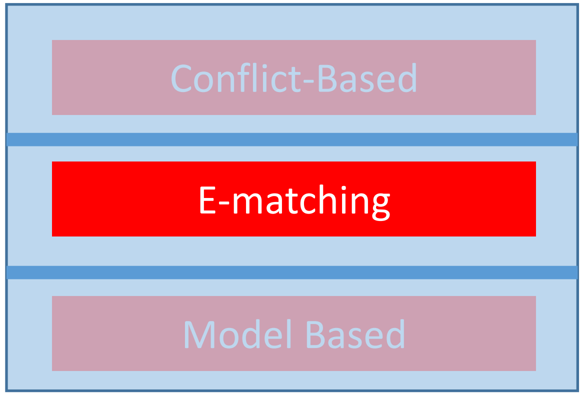


- Introduced in Nelson's Phd Thesis [Nelson 80]
  - Implemented in early SMT solvers, e.g. Simplify [Detlefs et al 03]
- Most widely used and successful technique for quantifiers in SMT
  - Software verification
    - Boogie/Dafny, Leon, SPARK, Why3
  - Automated Theorem Proving
    - Sledgehammer
-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

E-matching

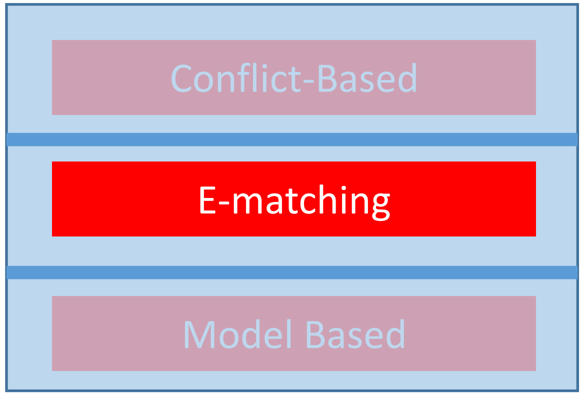


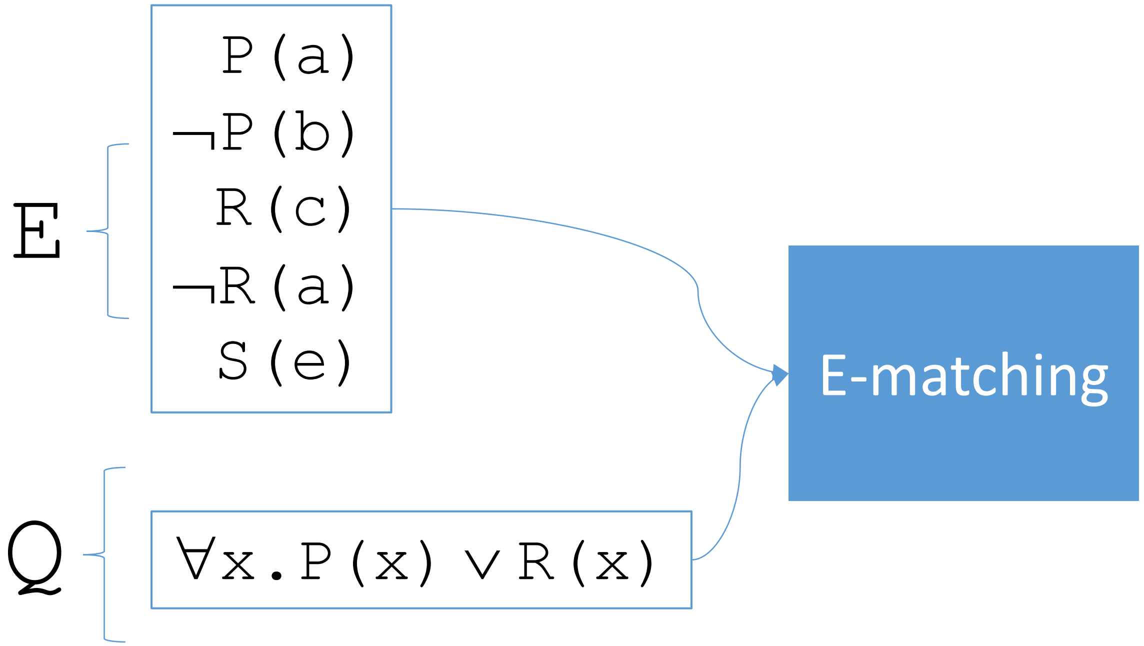
P(a)
¬P(b)
R(c)
¬R(a)
S(e)
E
Q
∀x.P(x) ∨ R(x)
E-matching

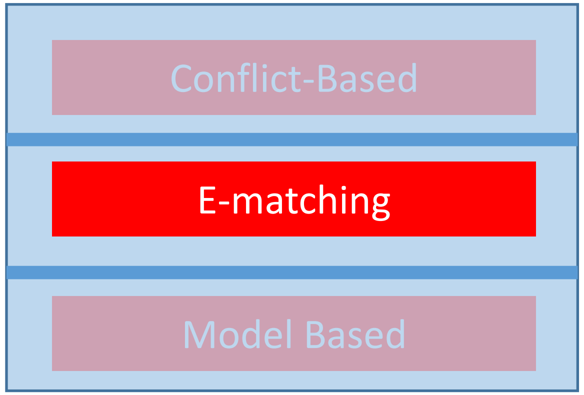


Conflict-Based
E-matching
Model Based

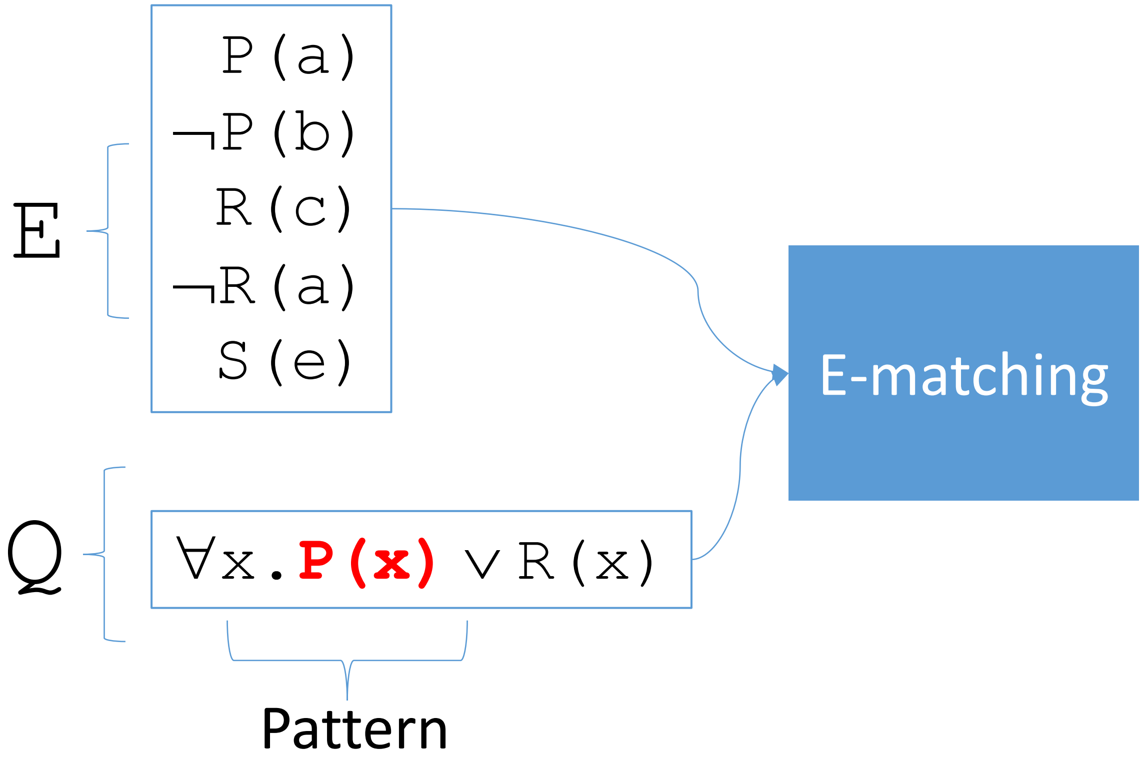
E-matching


Conflict-Based
E-matching
Model Based


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



E-matching



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

E-matching

Conflict-Based

E-matching

Model Based

E:

$\neg P(a)$

$\neg P(b)$

$R(c)$

$\neg R(a)$

$S(e)$

Q:

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

Pattern

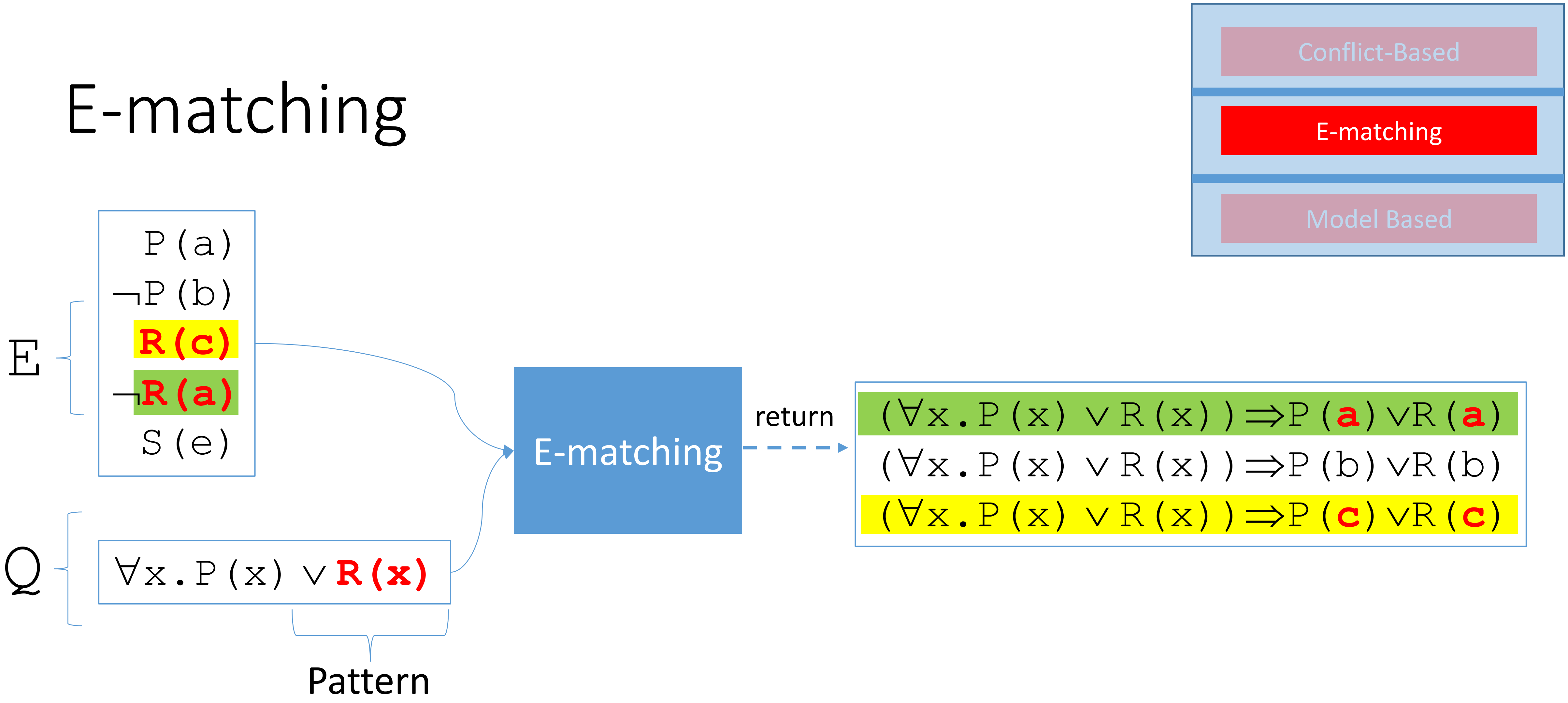
E-matching

return

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

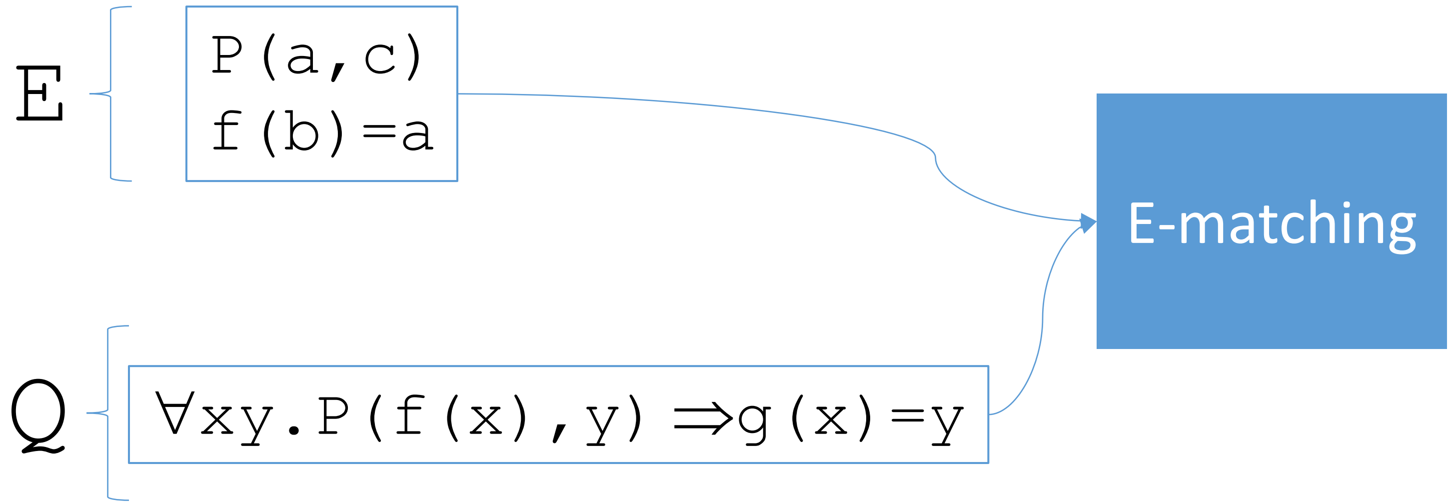
$(\forall x. P(x) \vee R(x)) \Rightarrow P(\mathbf{b}) \vee R(\mathbf{b})$

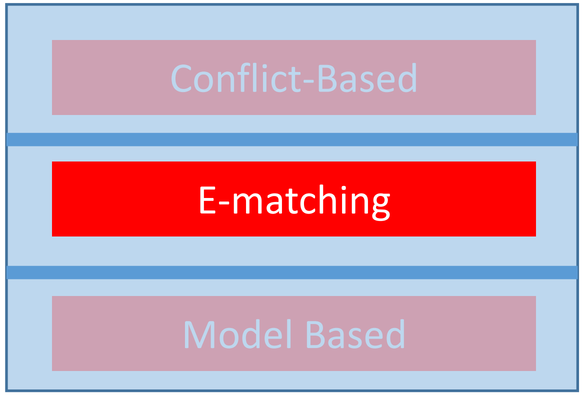
E-matching


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)

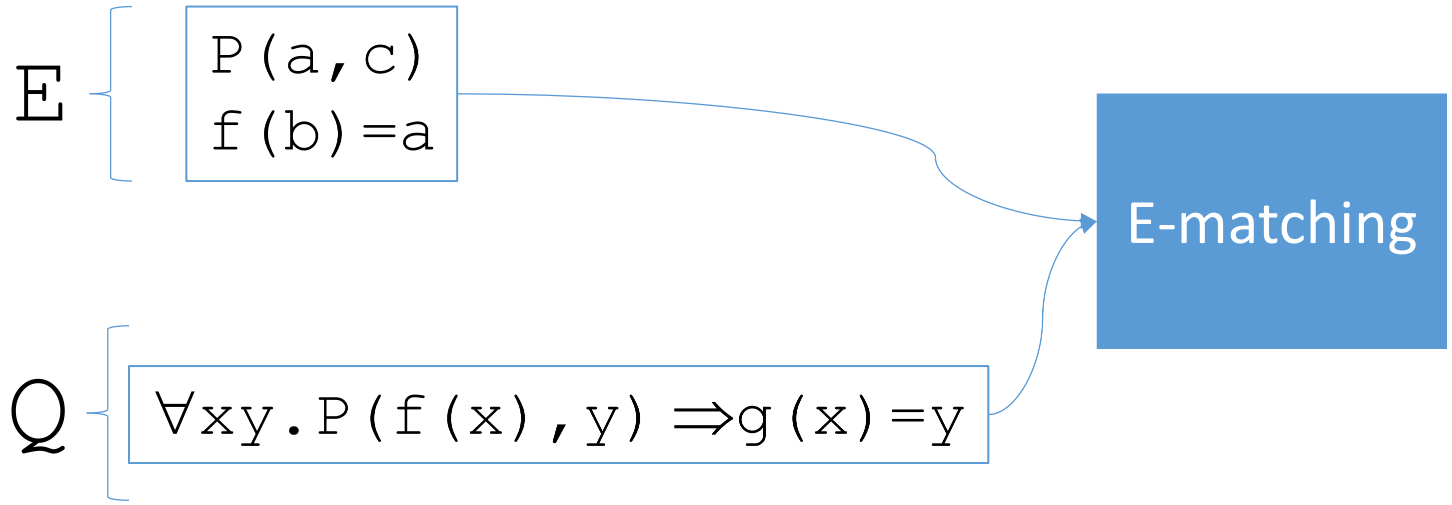
E-matching: Functions, Equality

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





E-matching: Functions, Equality



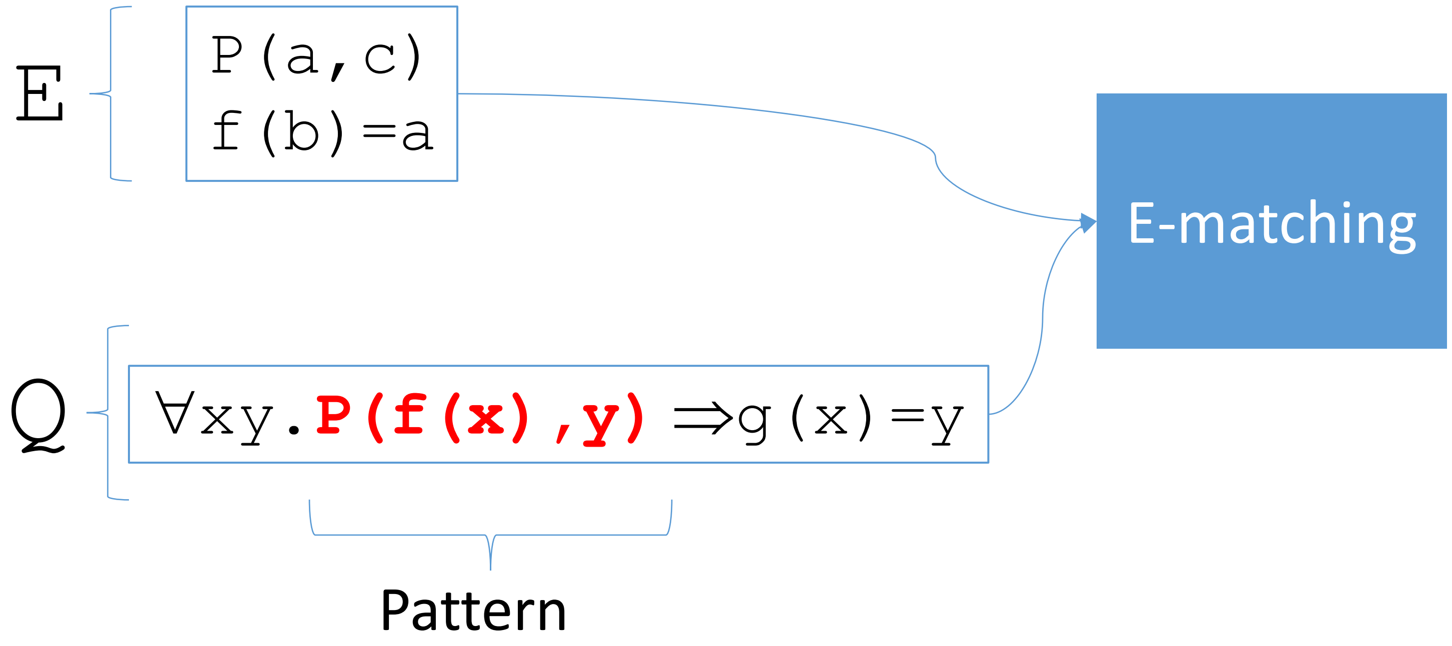
⇒ In E-matching, Pattern ***matching*** takes into account equalities in ***E***

E-matching: Functions, Equality

Conflict-Based

E-matching

Model Based

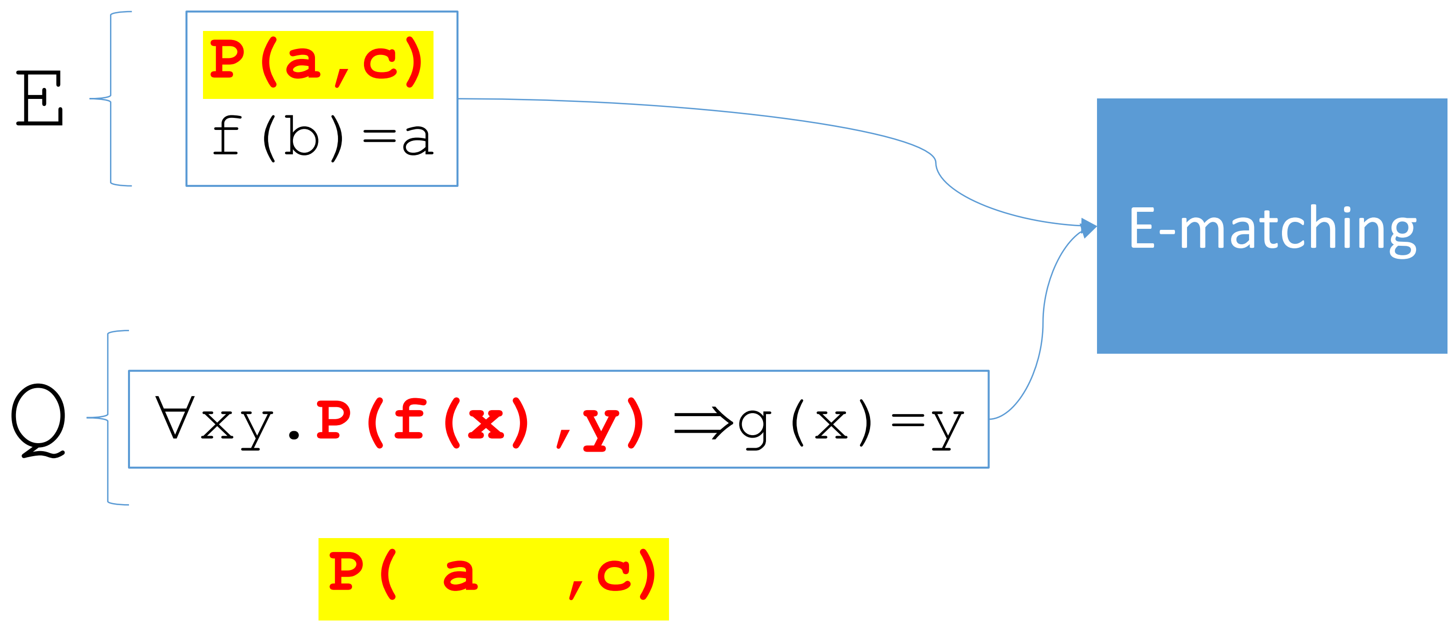


E-matching: Functions, Equality

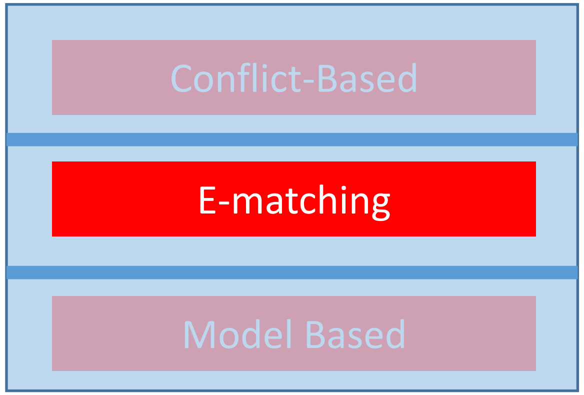
Conflict-Based

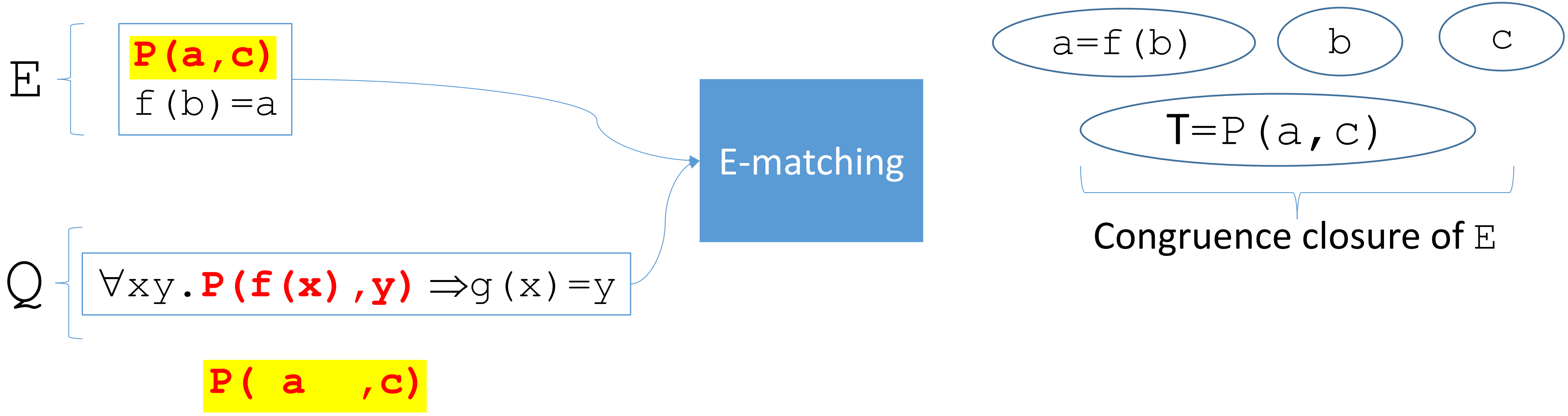
E-matching

Model Based



E-matching: Functions, Equality





E-matching: Functions, Equality

Conflict-Based

E-matching

Model Based

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

Q { ∀xy.P(f(x),y) ⇒g(x)=y

E-matching

a=f(b) | b | c

T=P(a,c)

P(f(b),c)

…E implies P(a,c) ⇔P(f(b),c)

E-matching: Functions, Equality

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: Intuition

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

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

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

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:** $\mathbf{\Psi[g]}$ is implied by $E \cup \{\mathbf{\Psi[p]\{x\rightarrow t\}}\}$

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 $E \cup \{\Psi[p]\{x \rightarrow t\}\}$

  $\Rightarrow$ ***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 $E\cup\{\Psi[p]\ \{\mathbf{x}\rightarrow\mathbf{t}\}\}$

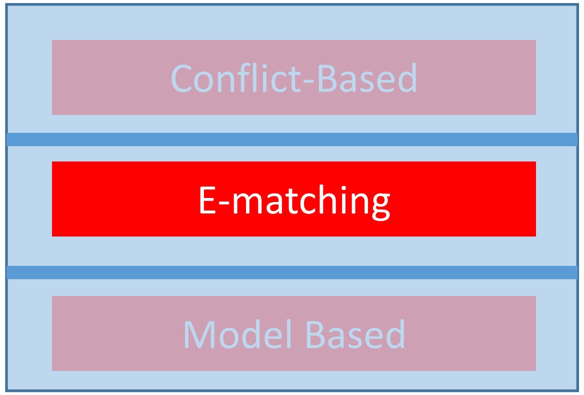
  $\Rightarrow$***In other words, from Q, we learn information Ψ[g] about a term g from E***

Learn $\mathbf{P(a,c)} \Rightarrow g(b)=c$ as a result of

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

E with new instance

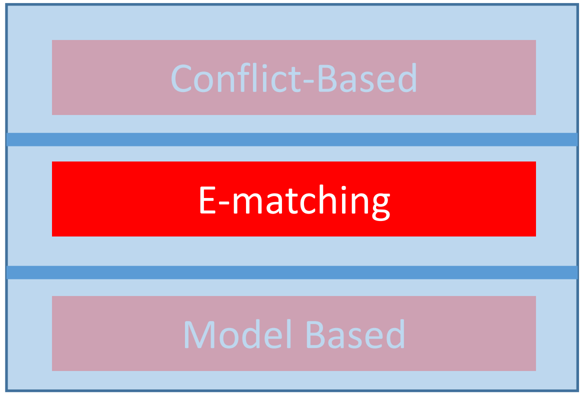
E-matching: Challenges



- 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

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]**

Conflict-Based

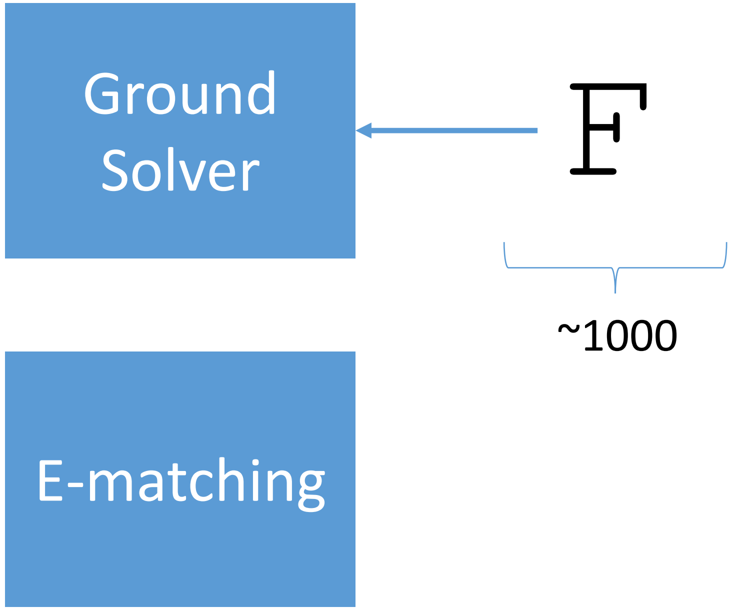
E-matching

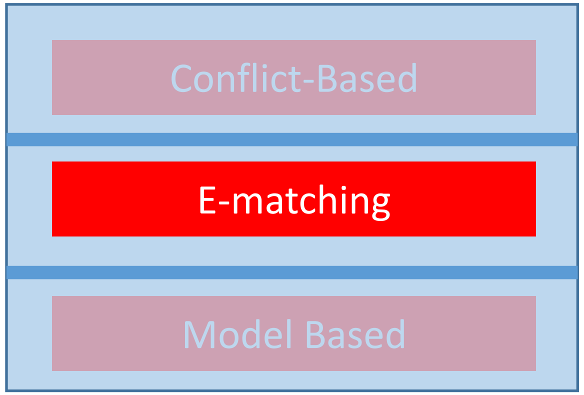
Model Based

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]**
  - …and may share similarities with literal selection heuristics in ATP, a la **[Reger et al 16]**?

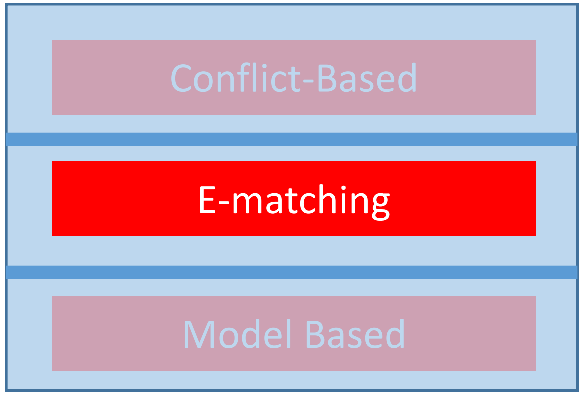
E-matching: Too Many Instances

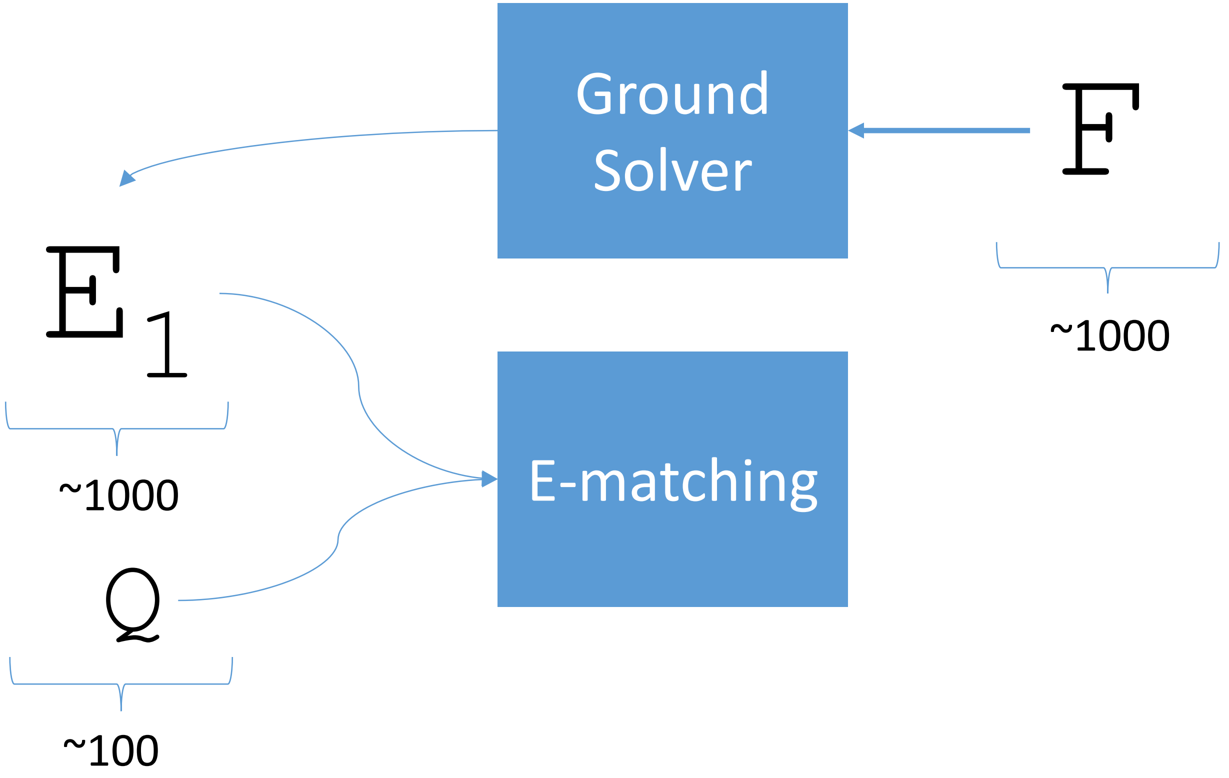




- Typical problems in applications:
  - $\mathtt{F}$ contains 1000s of clauses

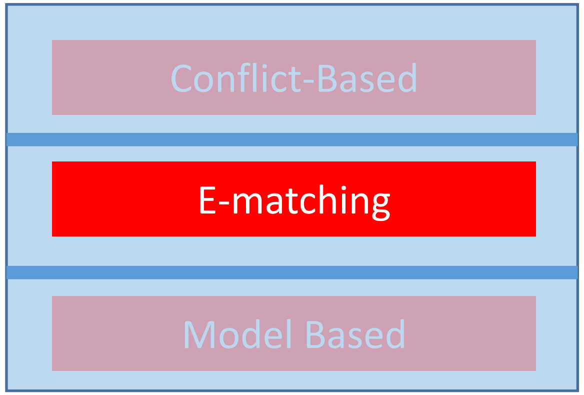
E-matching: Too Many Instances

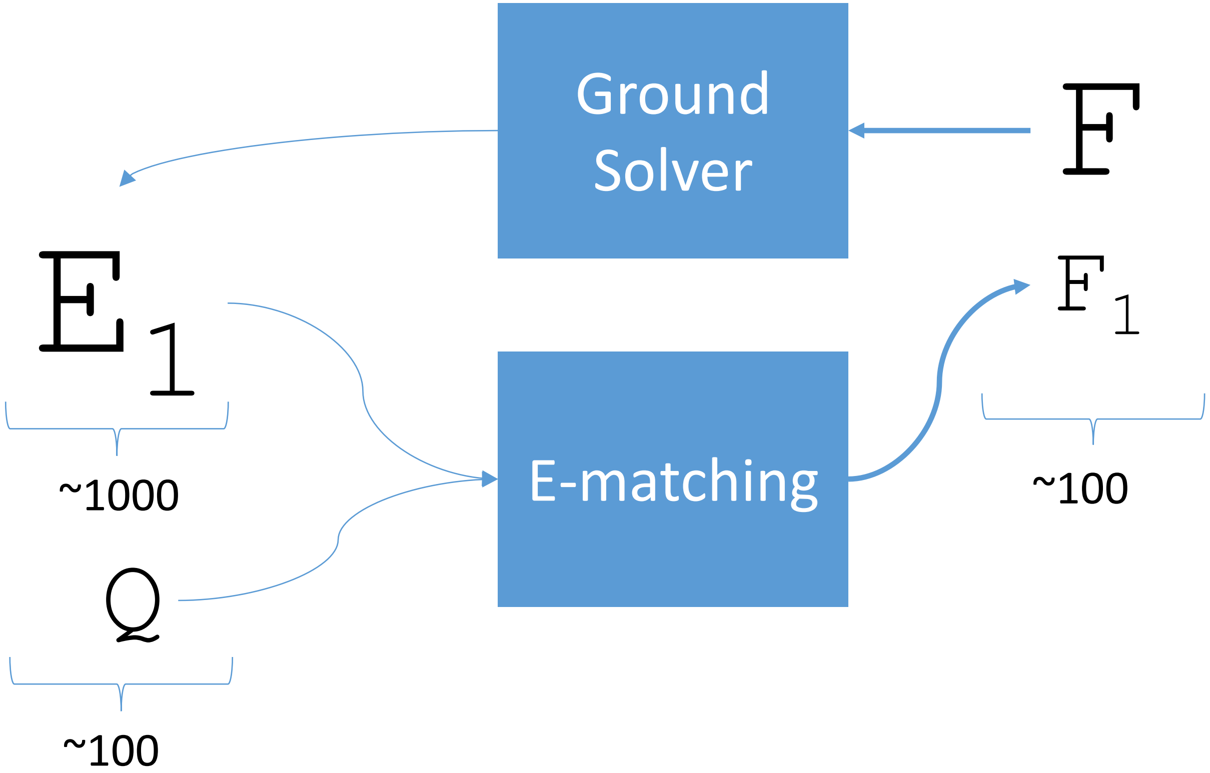




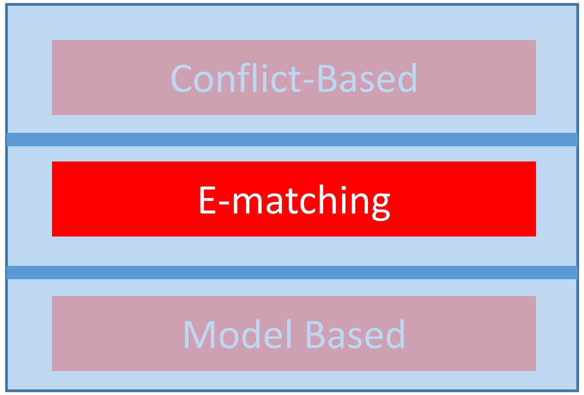
- Typical problems in applications:
  - F contains 1000s of clauses
  - Satisfying assignments contain 1000s of terms in E, 100s of ∀ in Q

E-matching: Too Many Instances

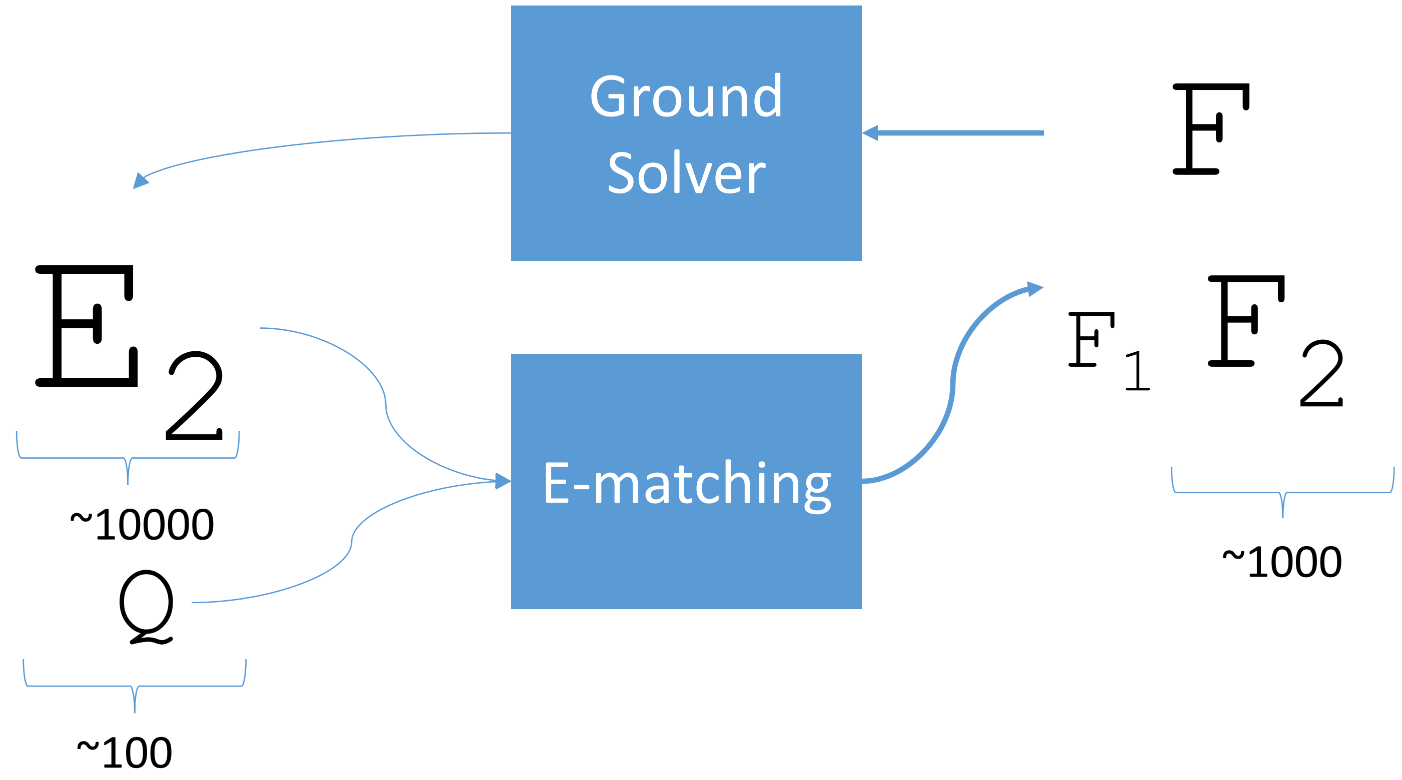




- Typical problems in applications:
  - F contains 1000s of clauses
  - Satisfying assignments contain 1000s of terms in E, 100s of ∀ in Q
  - Leads to 100s

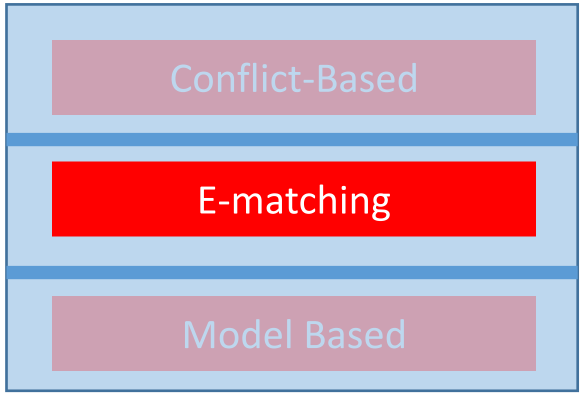


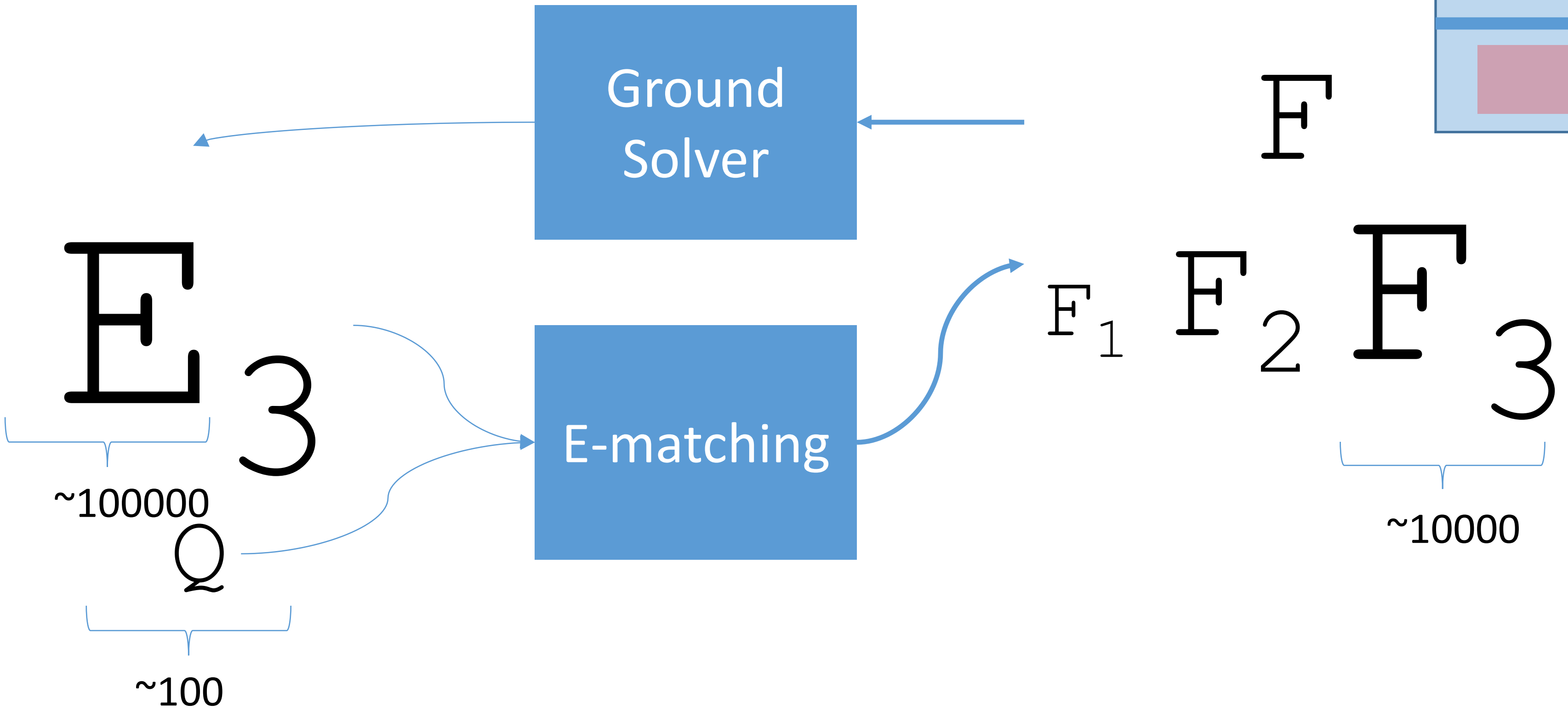
E-matching: Too Many Instances



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

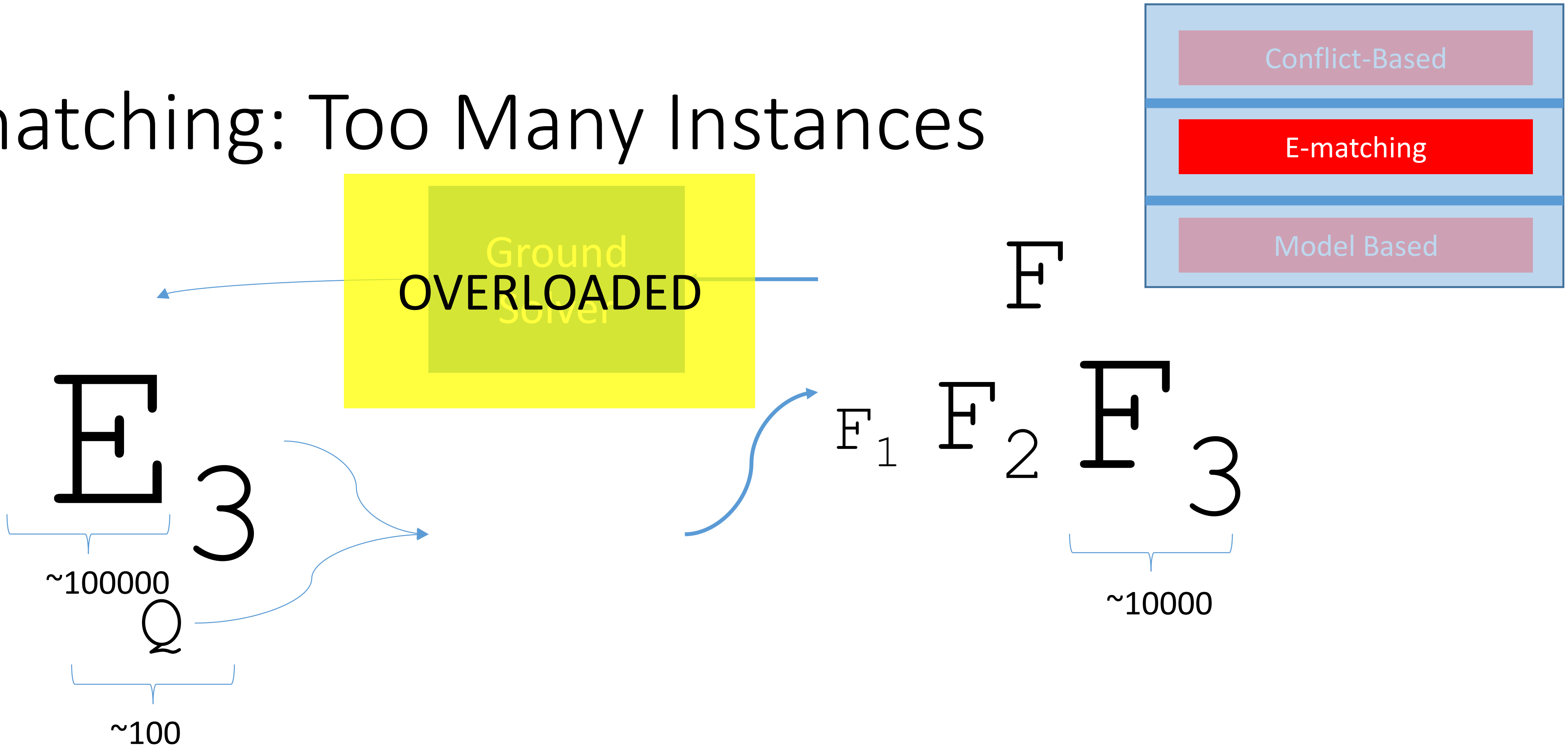
E-matching: Too Many Instances



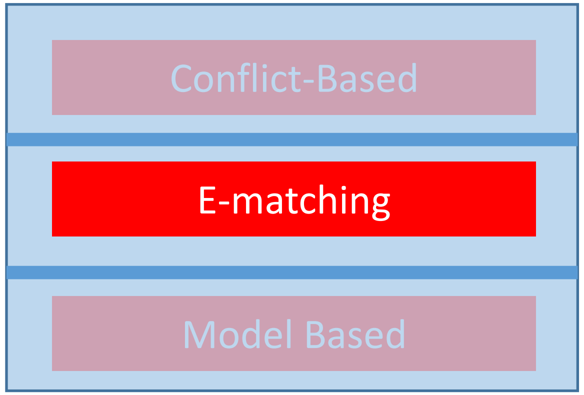


- Typical problems in applications:
  - F contains 1000s of clauses
  - Satisfying assignments contain 1000s of terms in E, 100s of ∀ in Q
  - Leads to 100s, 1000s, 10000s of instances

E-matching: Too Many Instances



$\Rightarrow$ Ground solver is overloaded, loop becomes slow,
…solver times out



E-matching: Too Many Instances

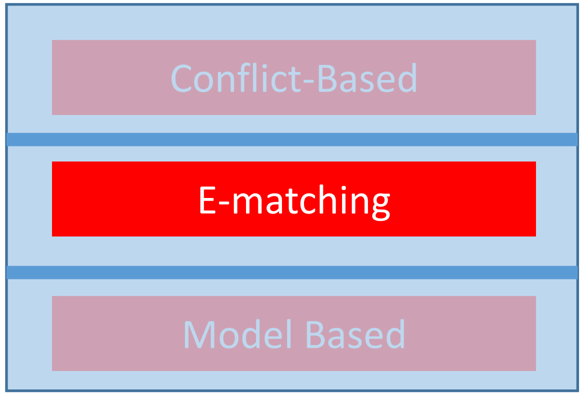
# Instances	cvc3		cvc4		z3	
	#	%	#	%	#	%
1-10	1464	13.49%	1007	8.87%	1321	11.43%
10-100	1755	16.17%	1853	16.31%	2554	22.11%
100-1000	**3816**	35.16%	**3680**	32.40%	**4553**	39.41%
1000-10k	1893	17.44%	2468	21.73%	1779	15.40%
10k-100k	1162	10.71%	1414	12.45%	823	7.12%
100k-1M	560	5.16%	607	5.34%	376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

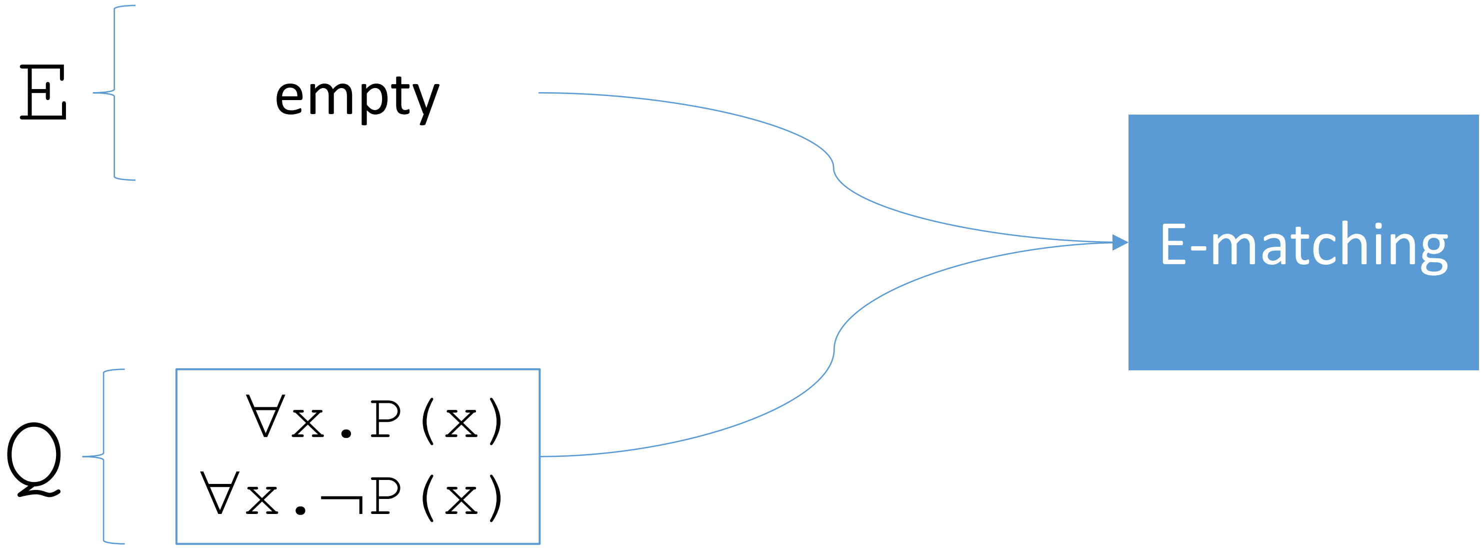
(for 8 of benchmarks z3 solves, its E-matching procedure adds more than 10M instances)

- Evaluation on 33032 SMTLIB, TPTP, Isabelle benchmarks
  - E-matching often requires **many instances**

    (Above, 16.6% required >10k, max 19.5M by z3 on a software verification benchmark from TPTP)

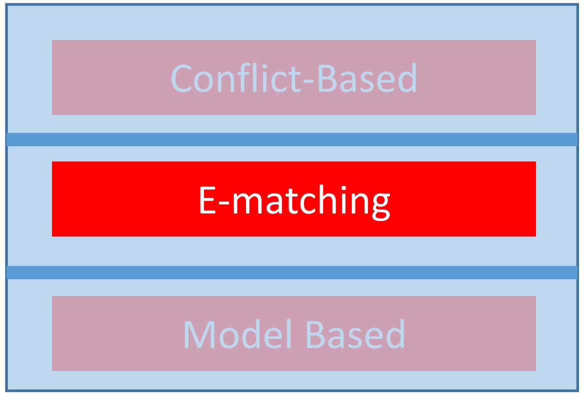
E-matching: Incompleteness

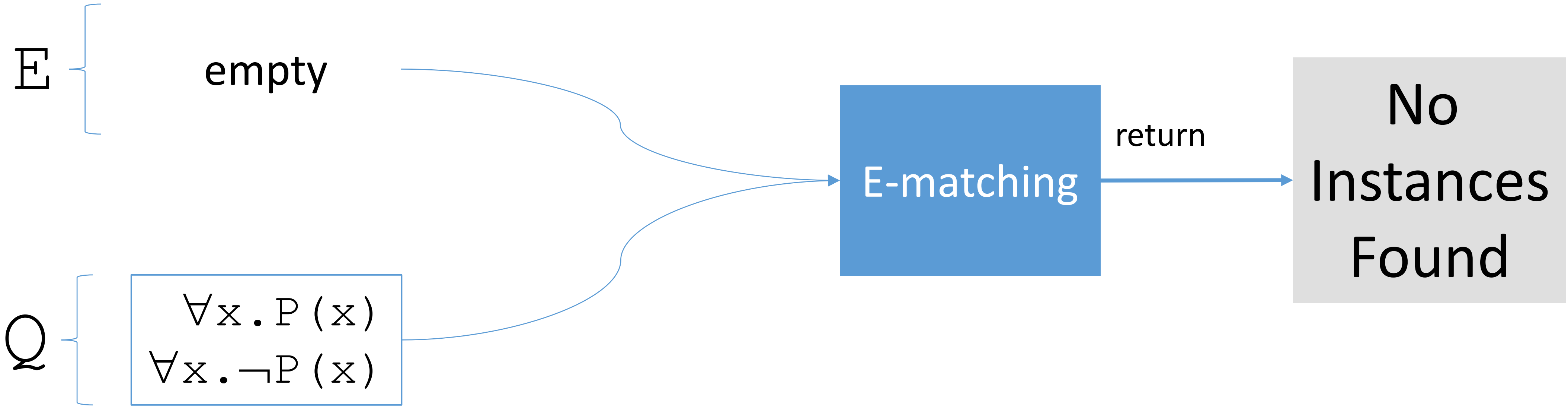




$\Rightarrow$ E-matching is an incomplete procedure

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

Conflict-Based
E-matching
Model Based

- 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

E-matching

Model Based

Conflict-Based Instantiation

E:

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

Q:

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

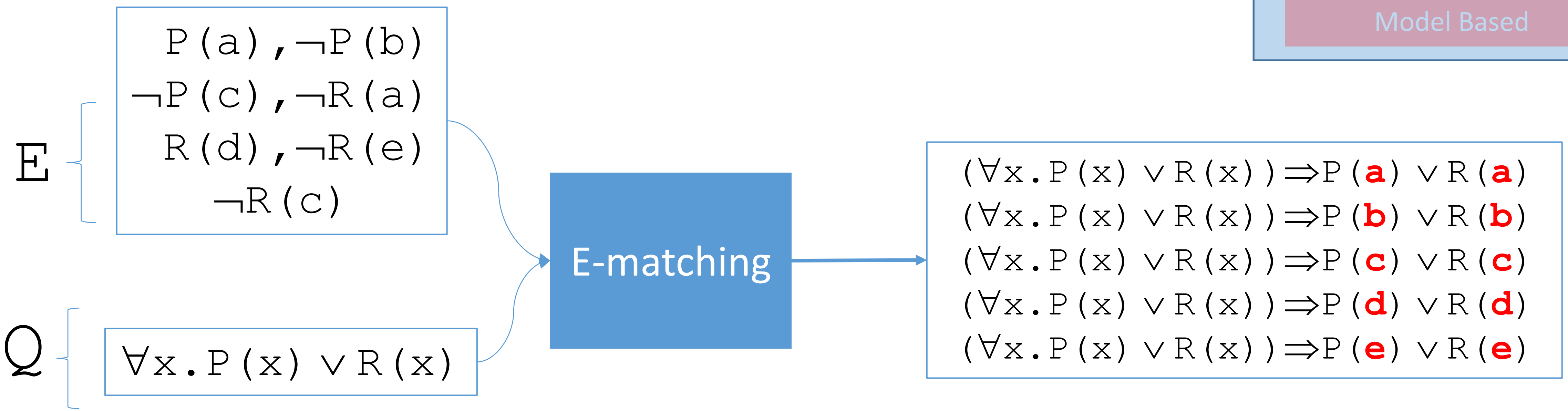
E-matching

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



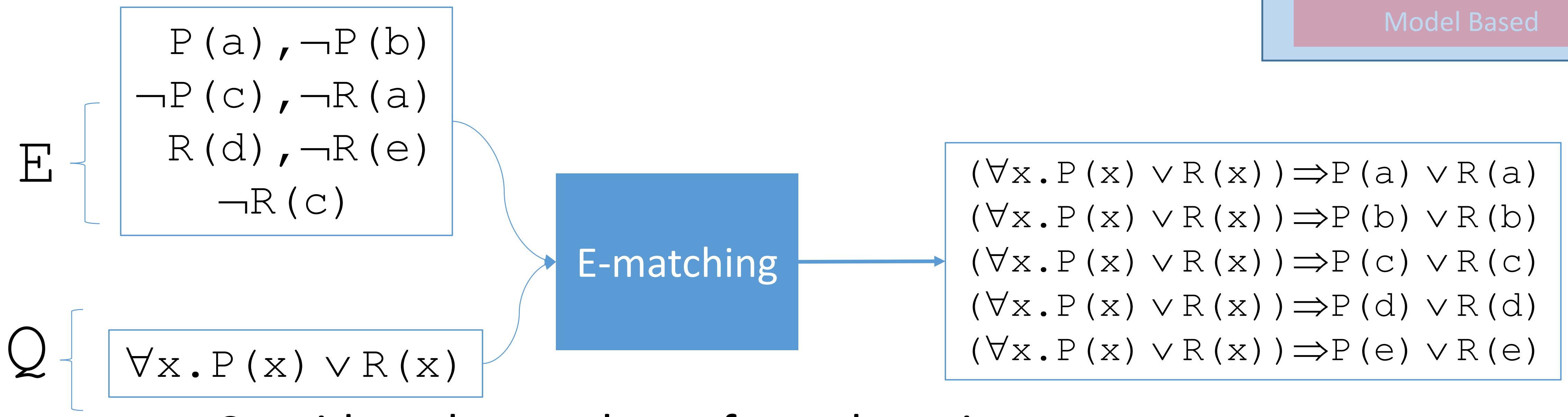
$\Rightarrow$ E-matching would produce {x→**a**}, {x→**b**}, {x→**c**}, {x→**d**}, {x→**e**}

Conflict-Based Instantiation

Conflict-Based

E-matching

Model Based



⇒ Consider what we learn from these instances:

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

Conflict-Based Instantiation

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

Q: ∀x.P(x) ∨ R(x)

E-matching →

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

⇒ Consider what we learn from these instances:

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

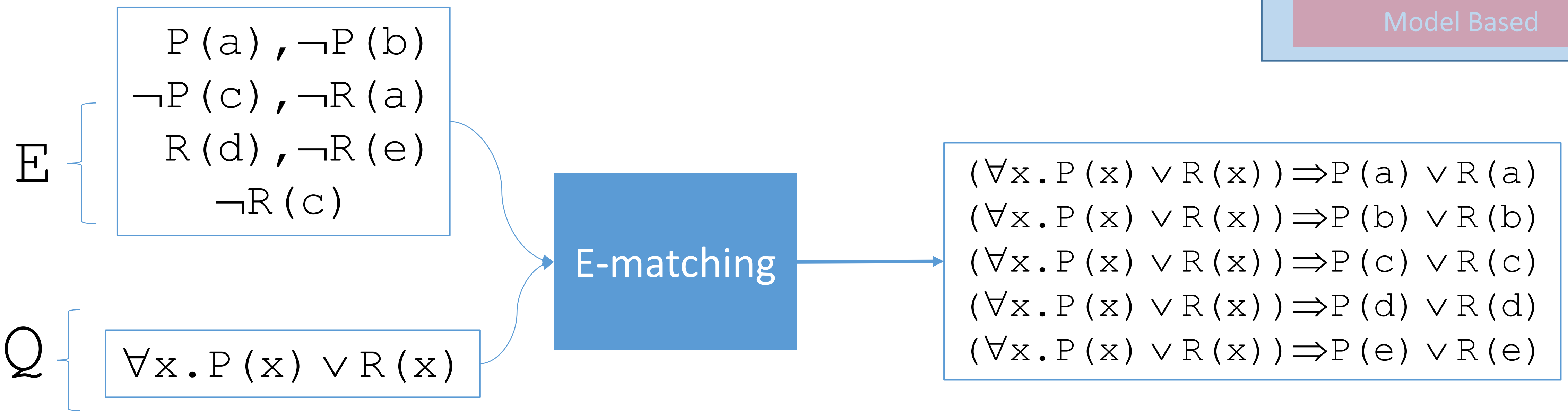
By E, we know **P(a)** ⇔ **T**

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



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

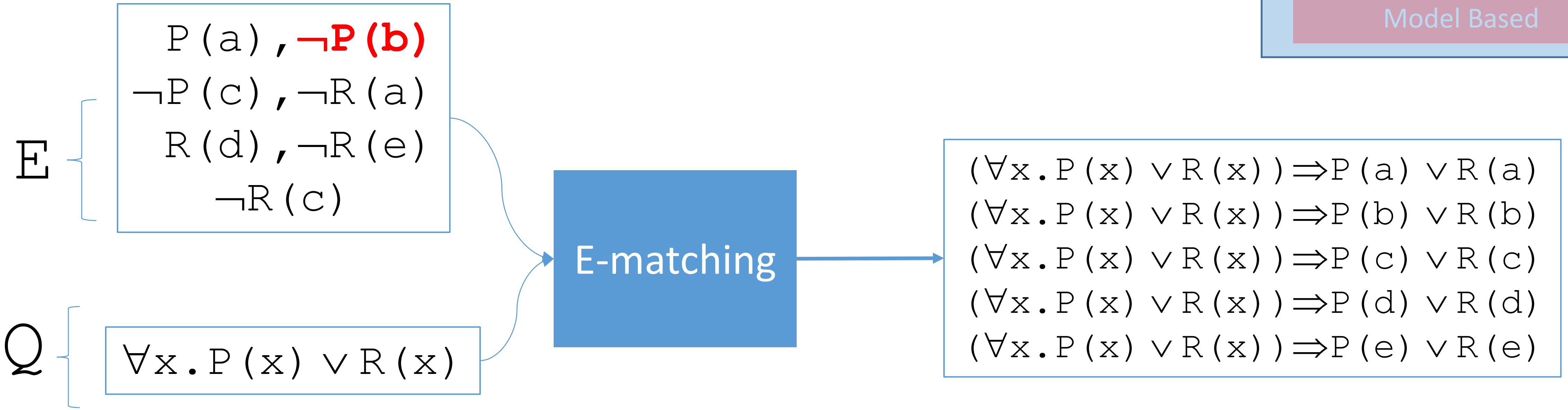
$$E, Q, P(a) \vee R(a) \models \top$$
$$E, Q, P(b) \vee R(b) \models P(b) \vee R(b)$$
$$E, Q, P(c) \vee R(c) \models P(c) \vee R(c)$$
$$E, Q, P(d) \vee R(d) \models P(d) \vee R(d)$$
$$E, Q, P(e) \vee R(e) \models P(e) \vee R(e)$$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



⇒ 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)** $\Leftrightarrow \bot$

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation

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

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

E-matching

Model Based

Conflict-Based Instantiation

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

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:

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

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

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:

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

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

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:

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

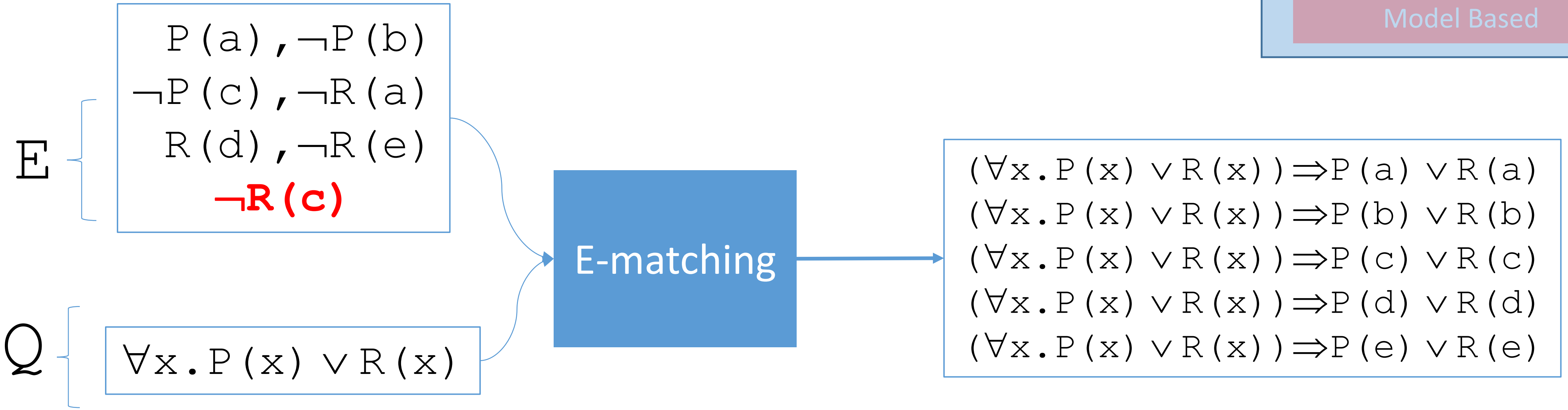
We know **$R(e) \Leftrightarrow \bot$**

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation



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

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:

$E, Q, P(a) \vee R(a) \models \top$

$E, Q, P(b) \vee R(b) \models R(b)$

$\mathbf{E, Q, P(c) \vee R(c) \models \bot}$

$E, Q, P(d) \vee R(d) \models \top$

$E, Q, P(e) \vee R(e) \models P(e)$

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

Conflict-Based Instantiation

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

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, Q → **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

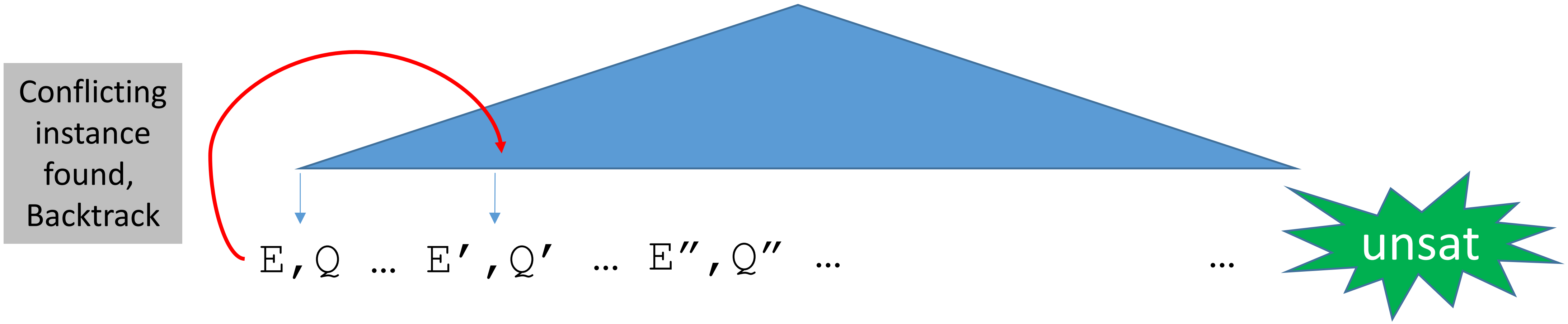
Conflict-Based

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′)



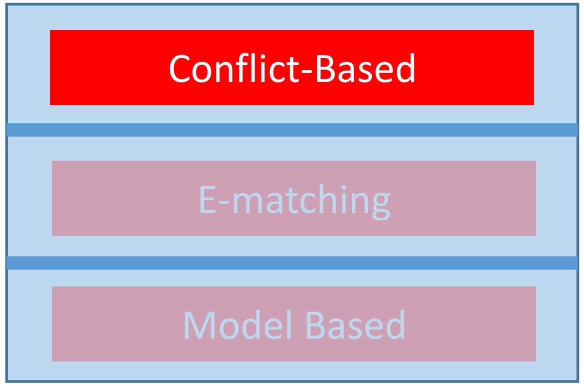
Conflict-Based
E-matching
Model Based

Conflict-Based Instantiation: EUF

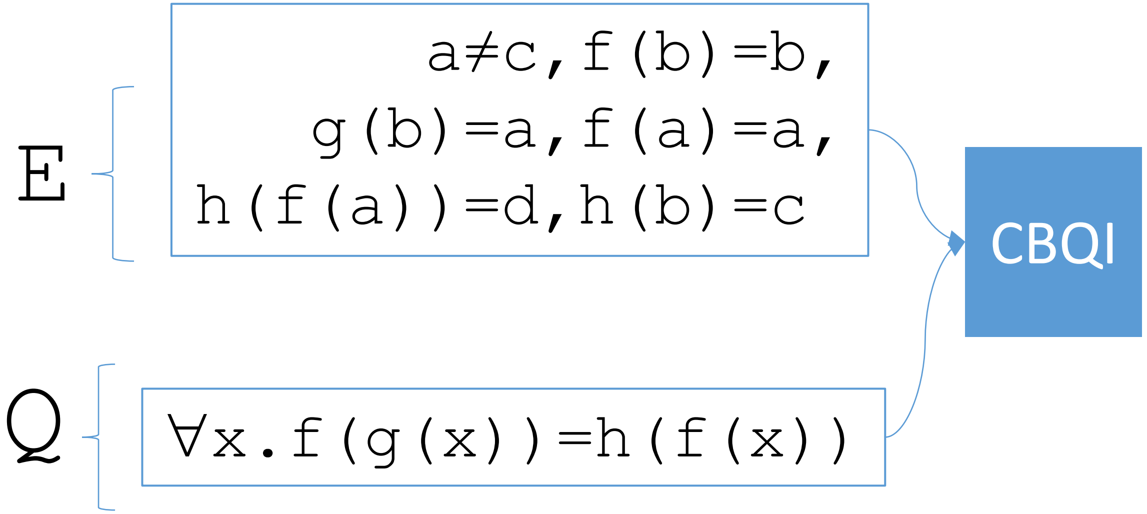
E: $a \neq c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c$

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

CBQI



Conflict-Based Instantiation: EUF



$\Rightarrow$ Consider the instance ∀x.f(g(x))=h(f(x)) $\Rightarrow$ f(g(**b**))=h(f(**b**))

- Is this conflicting for (E,Q)?

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

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

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF

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

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

CBQI

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

$b=f(b)$

$c=h(b)$

$d=h(f(a))$

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))

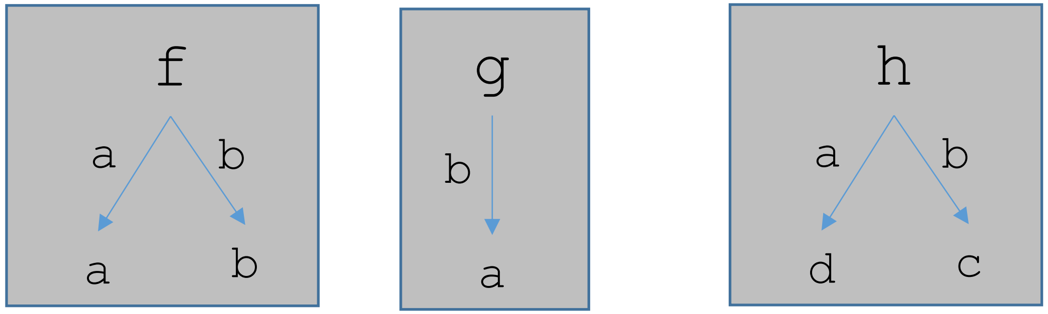
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))

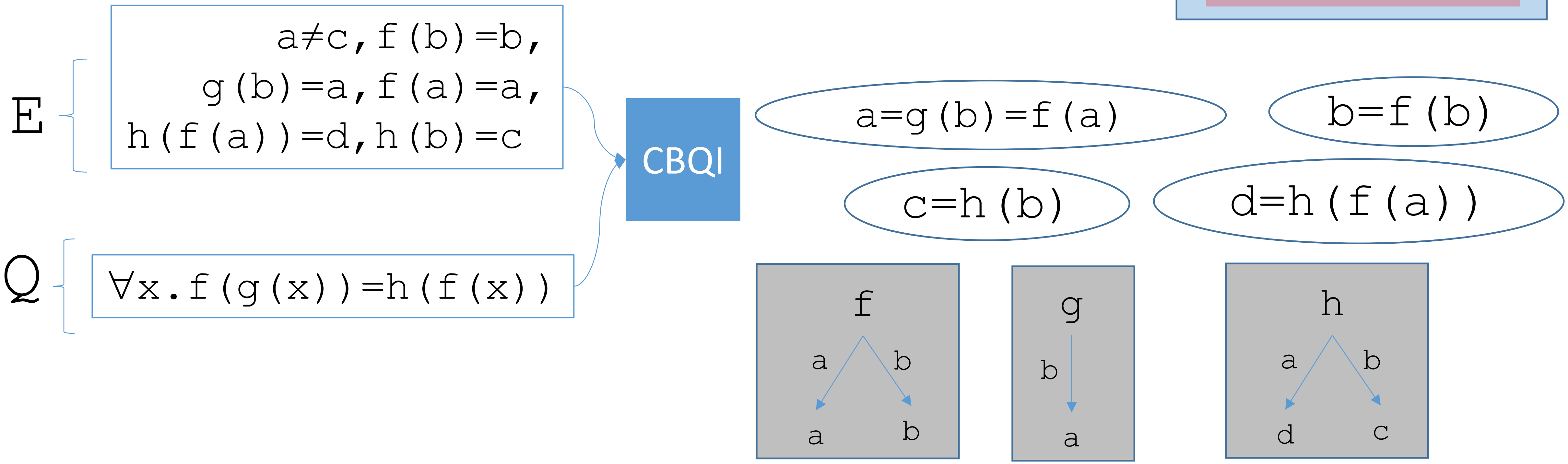


Build partial definitions for functions in terms of *representatives*

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

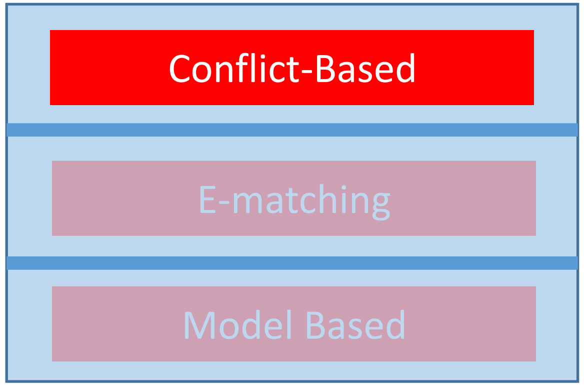
Conflict-Based
E-matching
Model Based

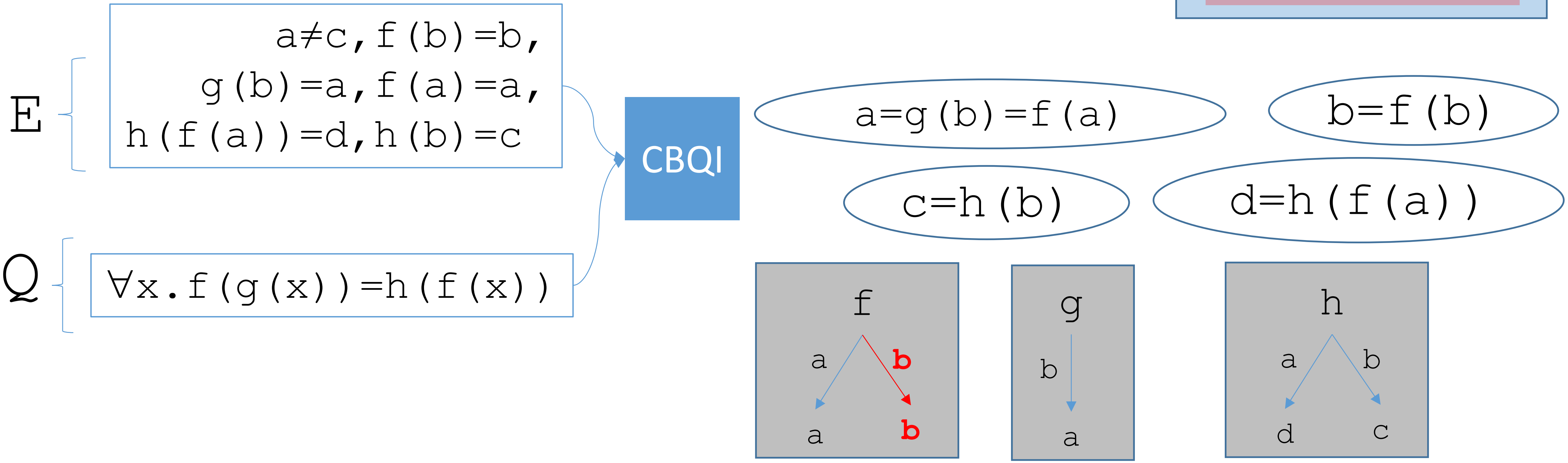
Conflict-Based Instantiation: EUF



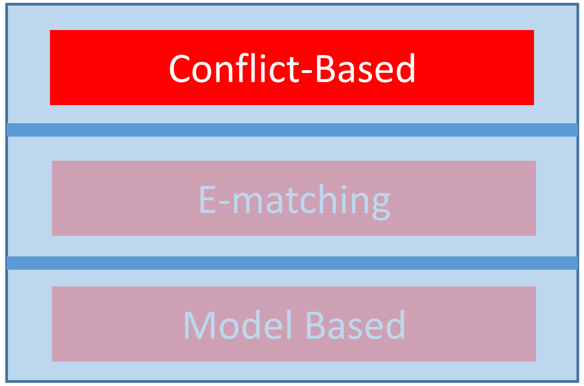
$$E, Q, f(g(b))=h(f(b)) \models_E f(g(b))=h(f(b))$$

Conflict-Based Instantiation: EUF

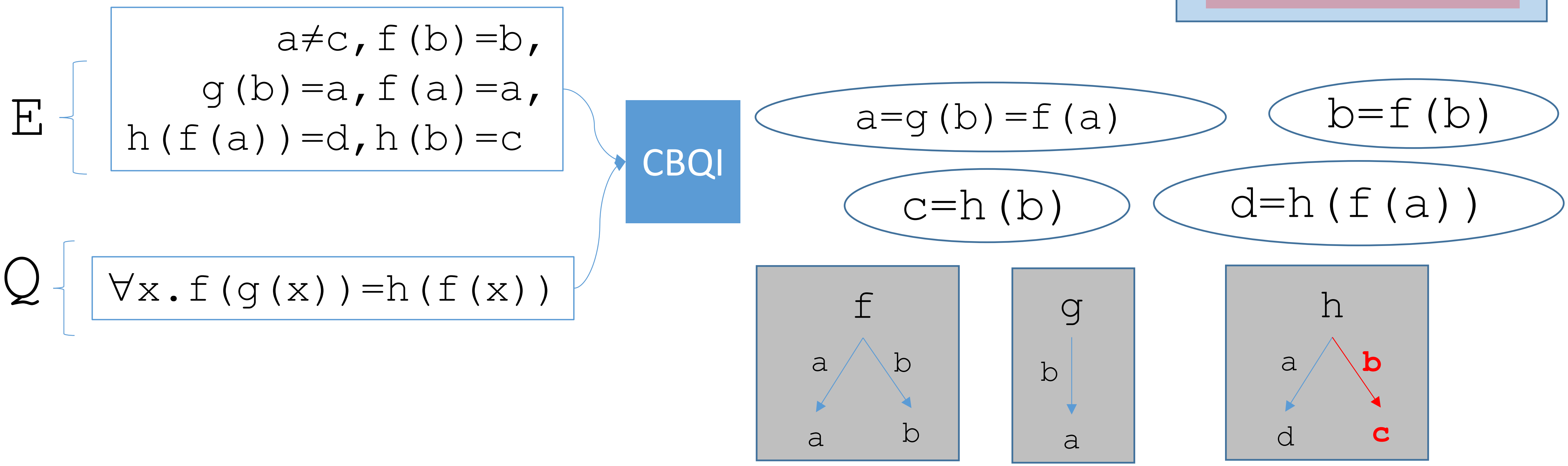




$$E, Q, f(g(b))=h(f(b)) \models_E f(g(b))=h(\ \mathbf{b}\ )$$



Conflict-Based Instantiation: EUF



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

Conflict-Based

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))`

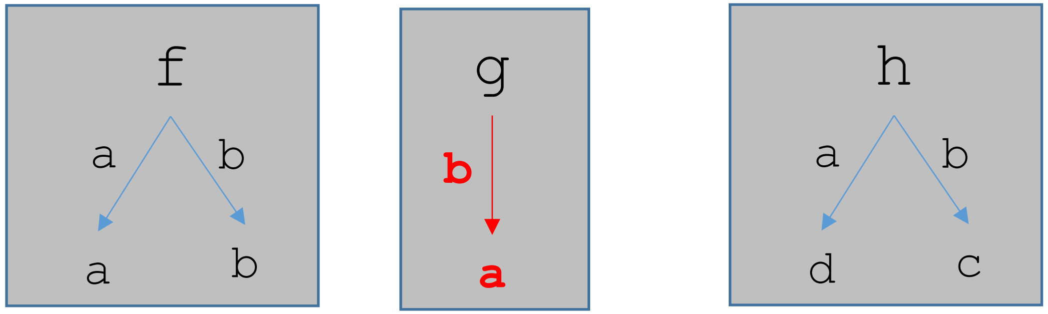
CBQI

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

`b=f(b)`

`c=h(b)`

`d=h(f(a))`



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

Conflict-Based Instantiation: EUF

Conflict-Based

E-matching

Model Based

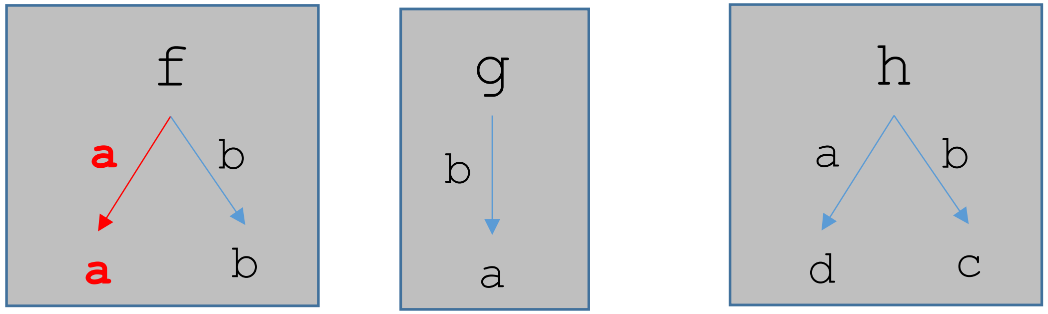
E: $a \neq c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c$

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

CBQI

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

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



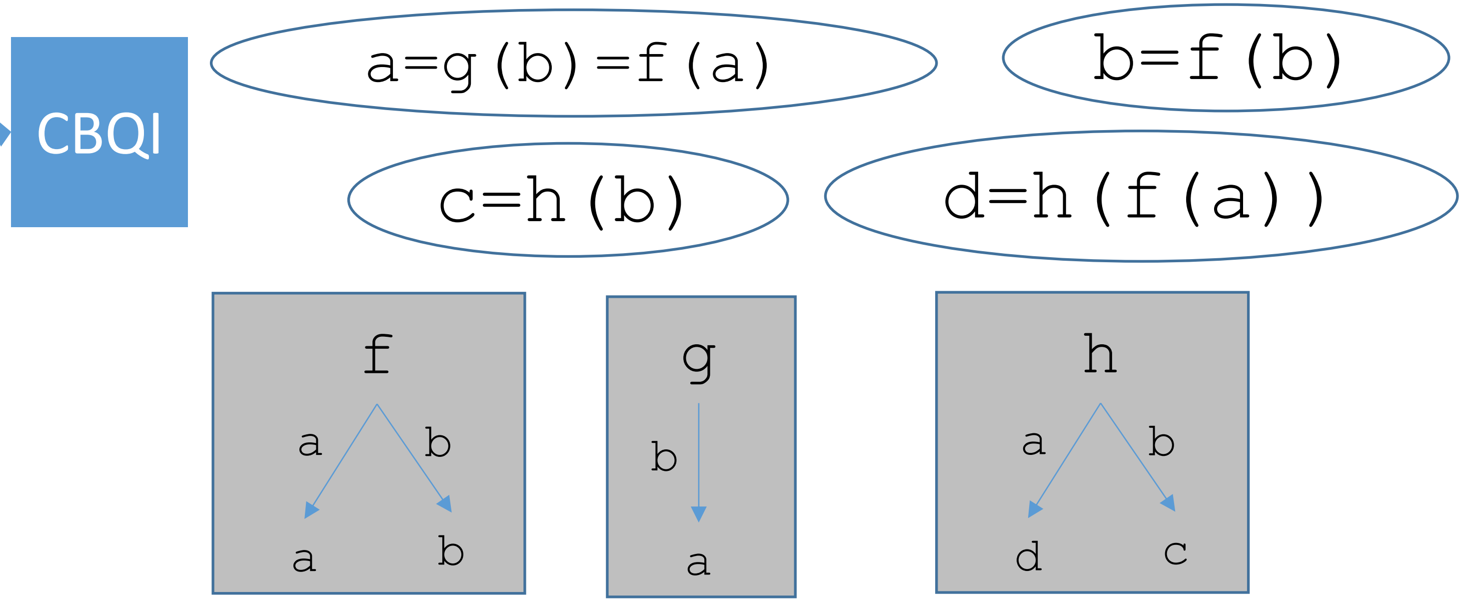
$E, Q, f(g(b))=h(f(b)) \models_E$ **a** $=$ c

Conflict-Based Instantiation: EUF

- Conflict-Based
- E-matching
- Model Based

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

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



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

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))

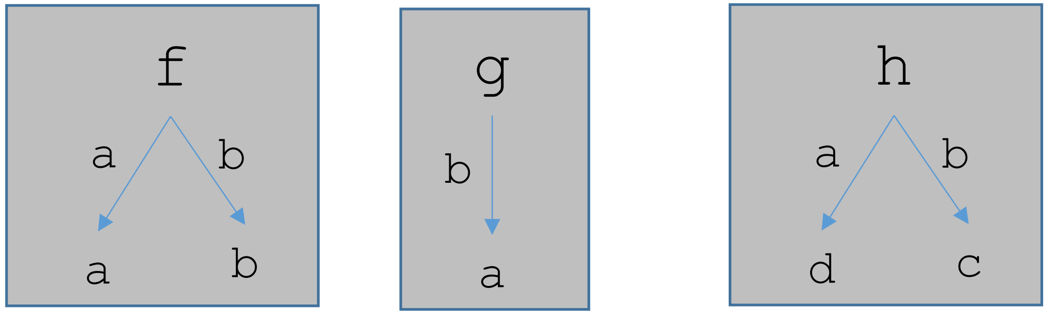
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



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

From E, we know **a≠c**

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation: EUF

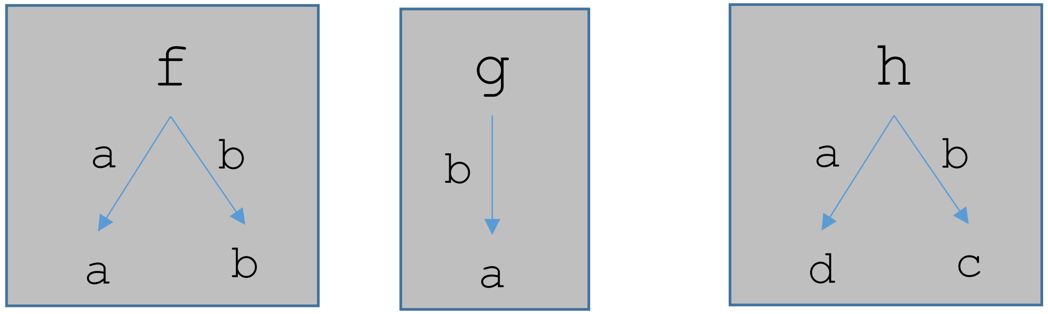
E: `a≠c, f(b)=b, g(b)=a, f(a)=a, h(f(a))=d, h(b)=c`

Q: `∀x.f(g(x))=h(f(x))`

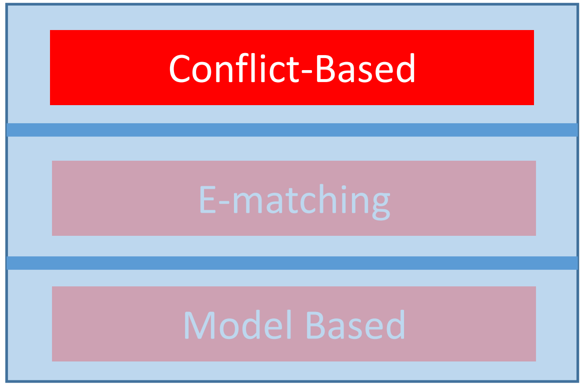
→ CBQI

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

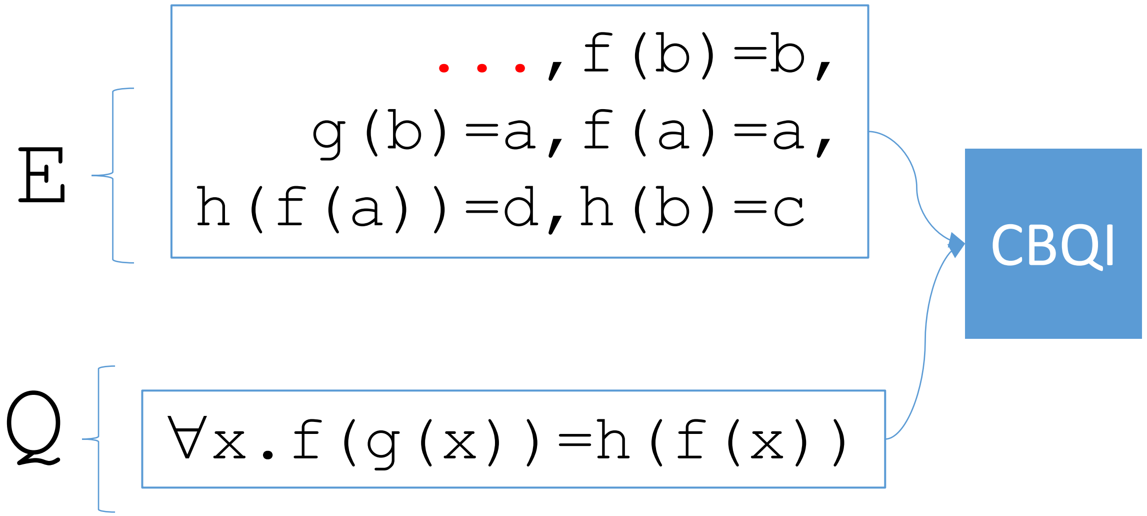
`c=h(b)` `d=h(f(a))`



`E,Q,f(g(b))=h(f(b))` $\models_E$ $\bot$ } `f(g(b))=h(f(b))` is a conflicting instance for (`E,Q`)!



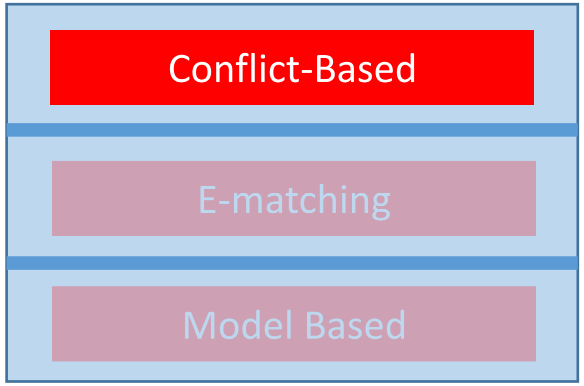
Conflict-Based Instantiation: EUF

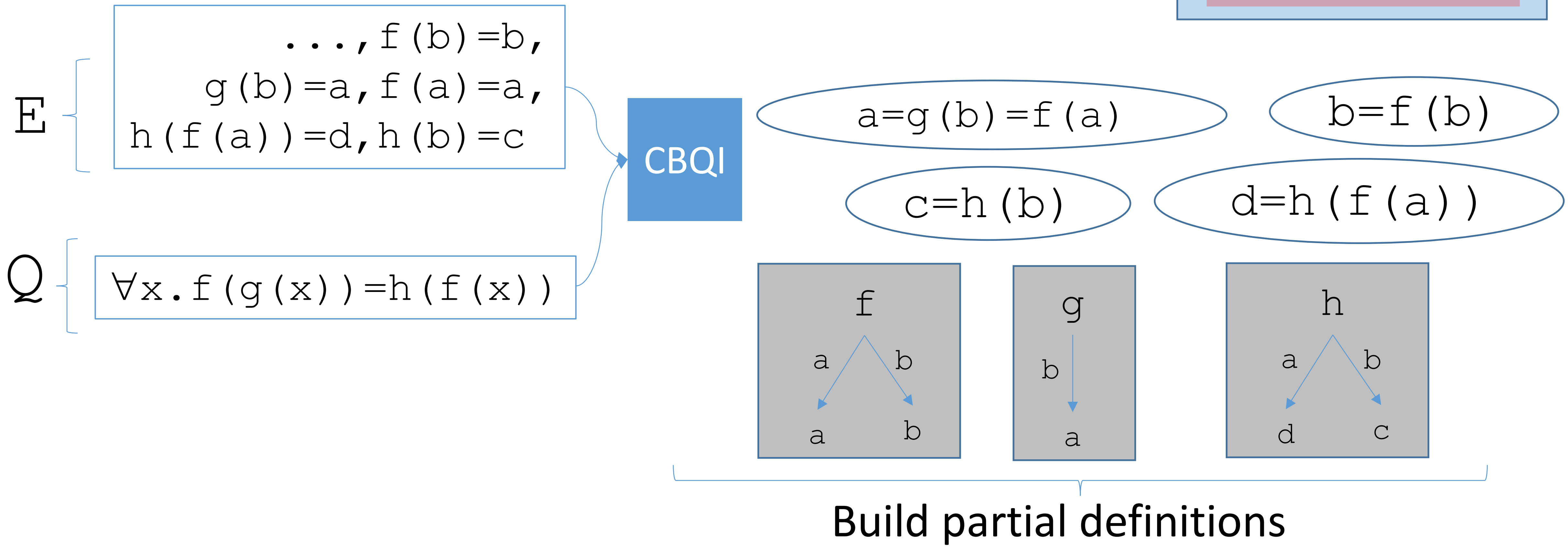


⇒ Consider the same example, but where we don't know a≠c

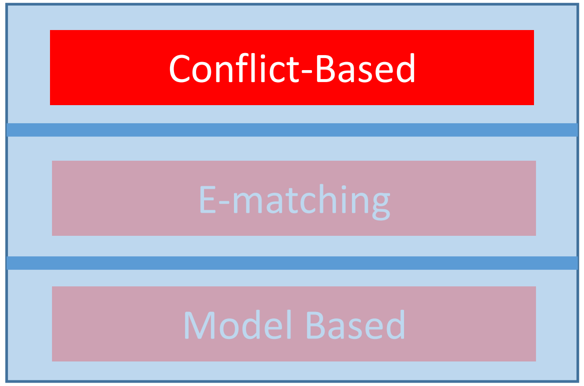
- Is the instance `f(g(b))=h(f(b))` still useful?

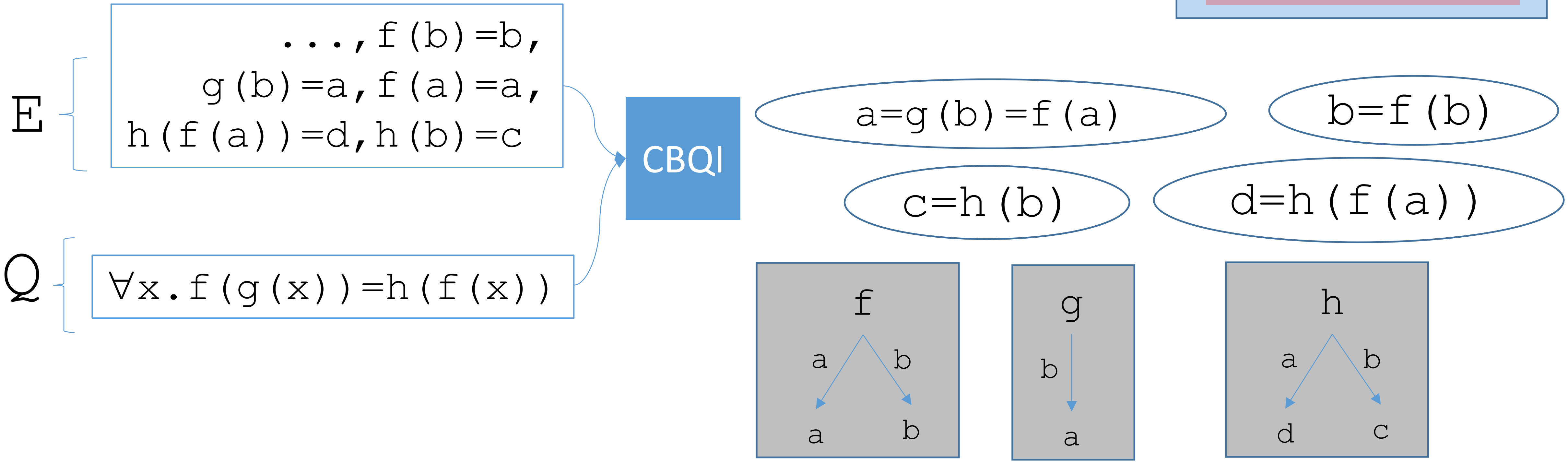
Conflict-Based Instantiation: EUF





Conflict-Based Instantiation: EUF





$\mathrm{E}, \mathrm{Q}, f(g(b))=h(f(b)) \models_{E} f(g(b))=h(f(b))$ } **Check entailment**

Conflict-Based Instantiation: EUF

Conflict-Based

E-matching

Model Based

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

Q: ∀x.f(g(x))=h(f(x))

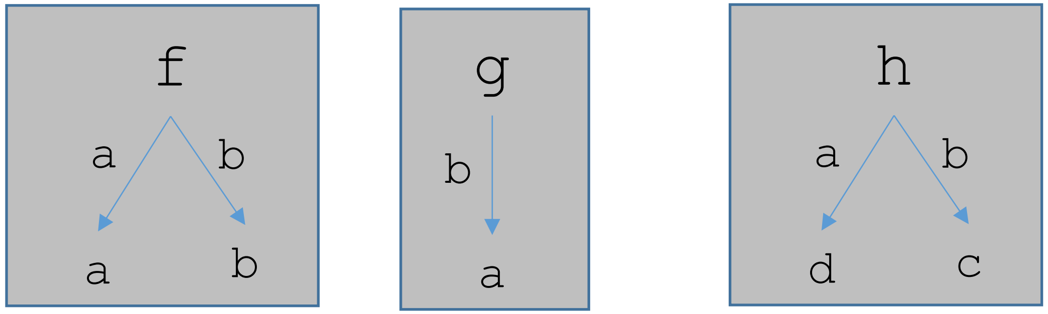
CBQI

a=g(b)=f(a)

b=f(b)

c=h(b)

d=h(f(a))



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

Conflict-Based

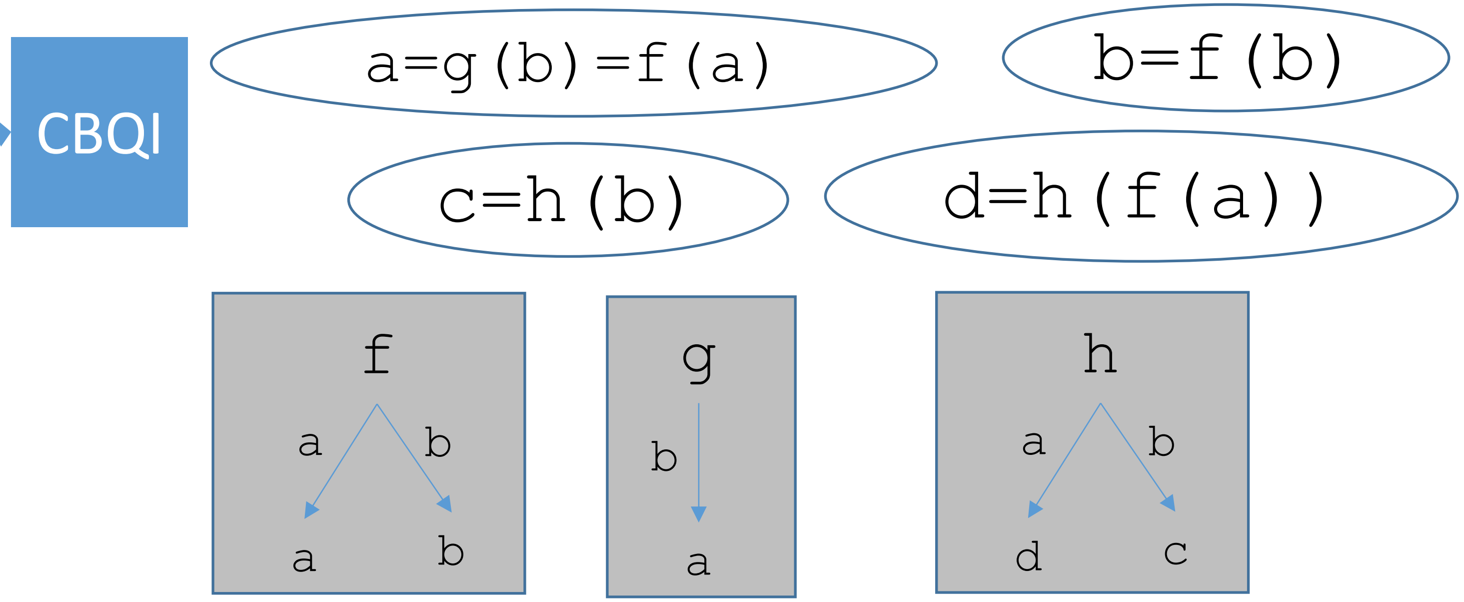
E-matching

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))



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))

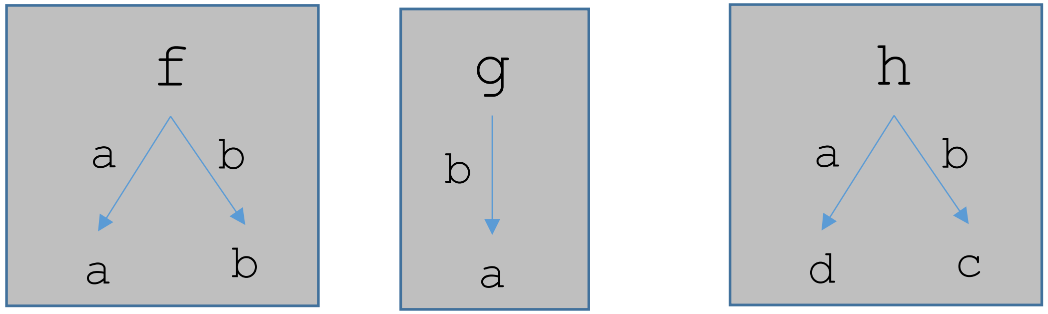
→ CBQI

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)

⇒ *These are also useful*

Conflict-Based

E-matching

Model Based

Conflict-Based Instantiation

Given:

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

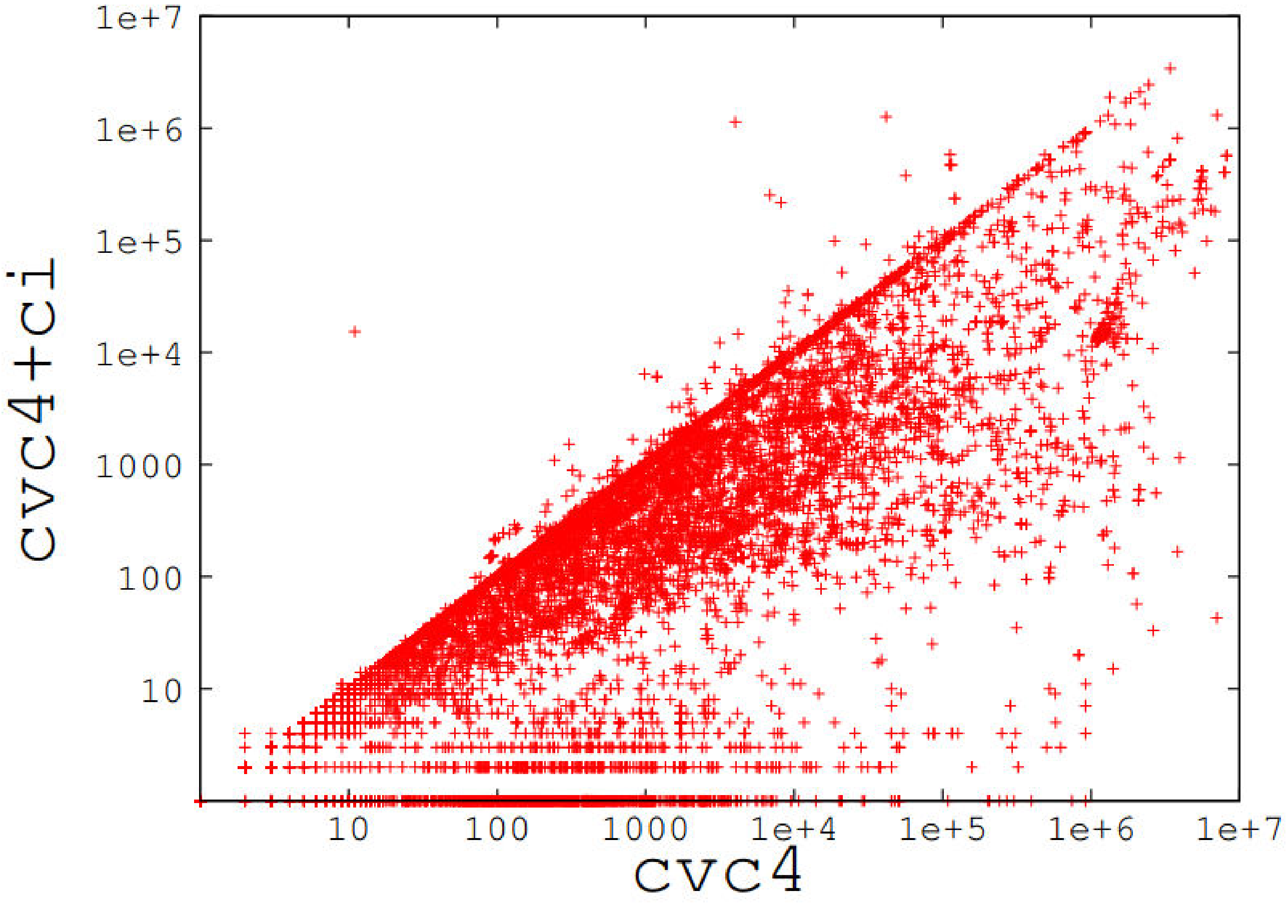
- **Conflict-based instantiation**:
  1. If there exists a *conflicting instance* $\Psi\{\mathbf{x} \rightarrow \mathbf{t}\}$
     - Returns $\{\forall x. \Psi \Rightarrow \Psi\{\mathbf{x} \rightarrow \mathbf{t}\}\}$ only
  2. If there exists *propagating instance(s)*, $\Psi_{\mathbf{i}}\{\mathbf{x} \rightarrow \mathbf{t}_{\mathbf{i}}\}$ for $\mathtt{i=1, \ldots, n}$
     - 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

Conflict-Based

E-matching

Model Based

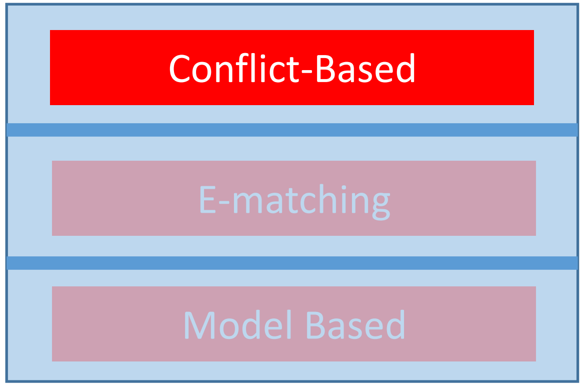


Reported number of instances.

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

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

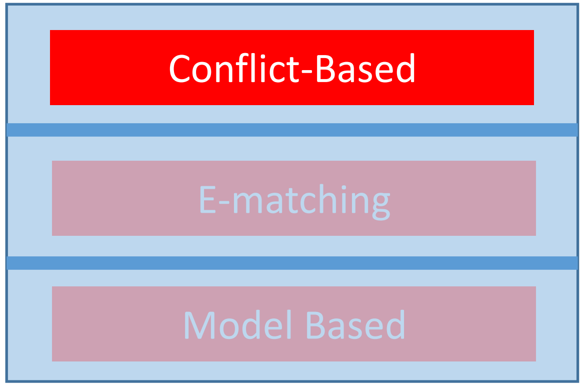
Conflict-Based Instantiation: Impact



- 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

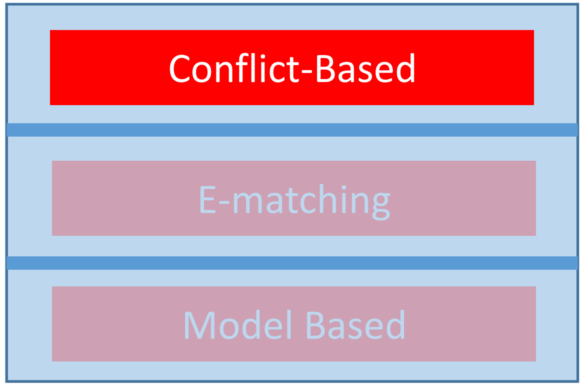


- CVC4 with conflicting instances **`cvc4+ci`**
  - Solves the most benchmarks for TPTP and Isabelle
  - Requires almost an order of magnitude fewer instantiations

	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

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

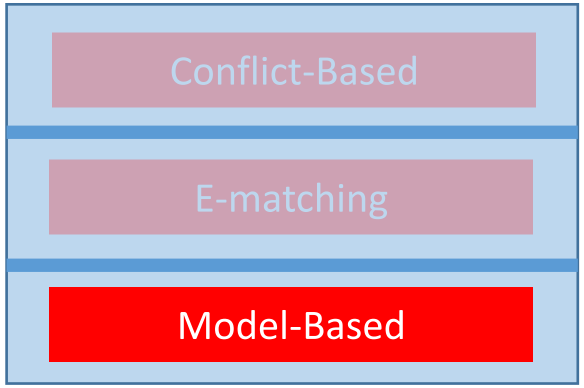
Conflict-Based Instantiation: Challenges



- How do we *find* conflicting instances?
  - Idea: construct instances via a stronger version of matching
    - Intuition: for $\forall x.P(x) \vee Q(x)$, will *only* match $P(x)$ where $P(t) \Leftrightarrow \bot$

      (For technical details, see **[Reynolds et al FMCAD2014]**)
- What about conflicts involving *multiple quantified formulas*?
- What if our quantified formulas that contain *theory symbols*?

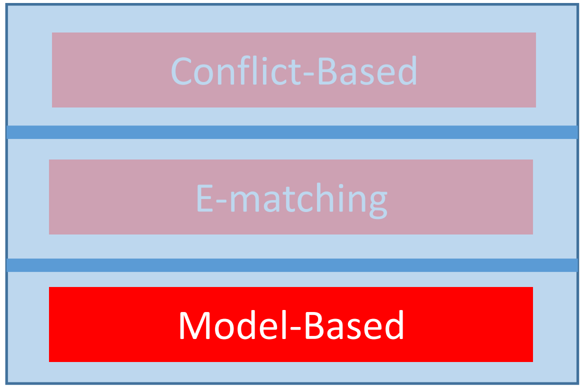
Model-based Instantiation

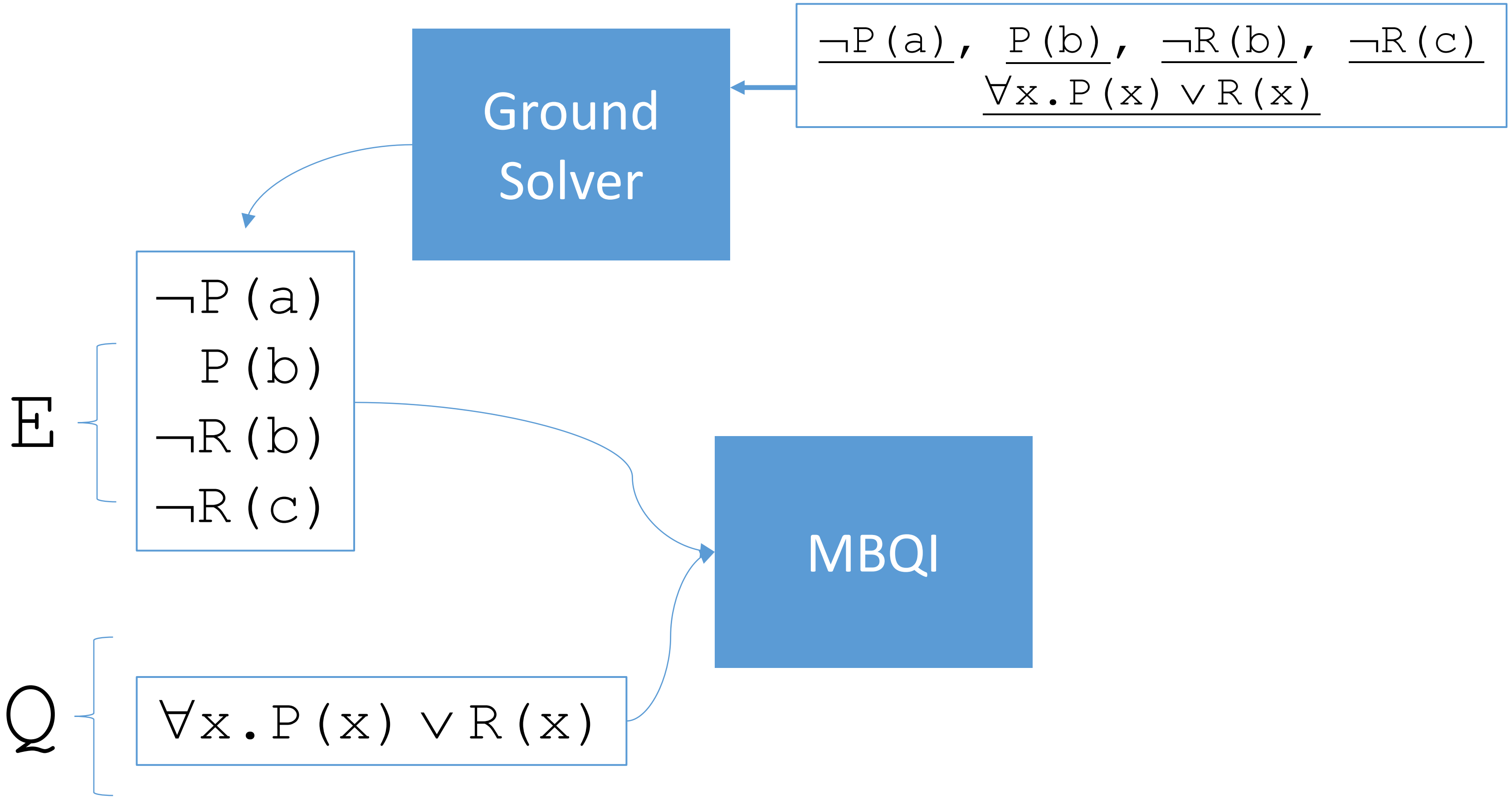


- Implemented in solvers:
  - Z3 **[Ge et al CAV09]**, CVC4 **[Reynolds et al CADE13]**
- Basic idea:
  1. Build interpretation **M** for all uninterpreted functions in the signature
  2. If this interpretation satisfies all formulas in Q, answer "sat"
     - e.g. interpretation $f^M=\lambda x.1$ satisfies $\forall x. f(x)>0$

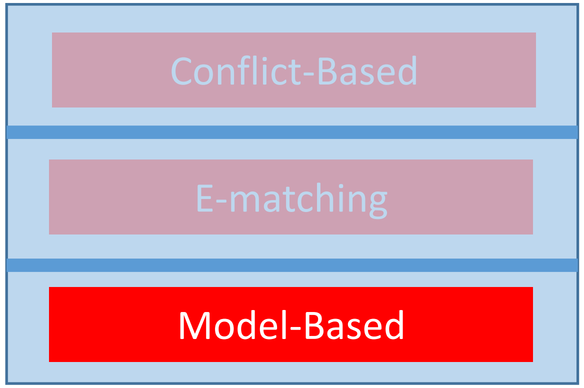
$\Rightarrow$ *Ability to answer "sat"*

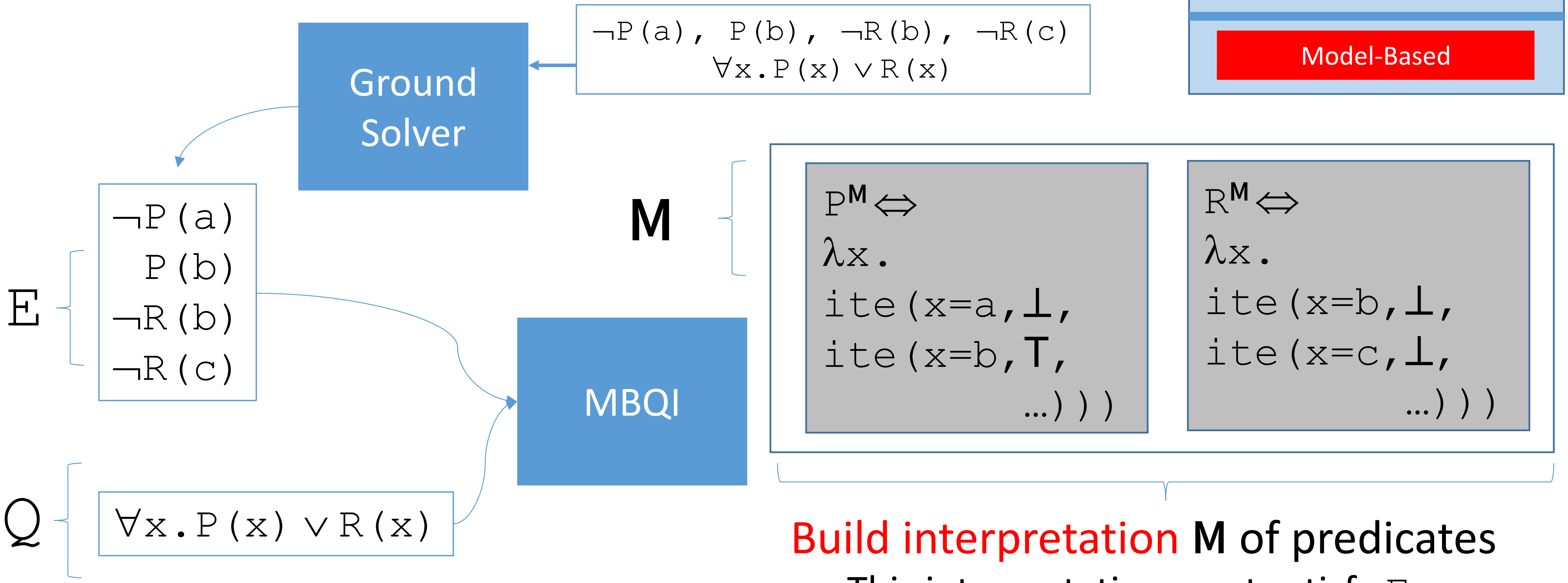
Model-based Instantiation





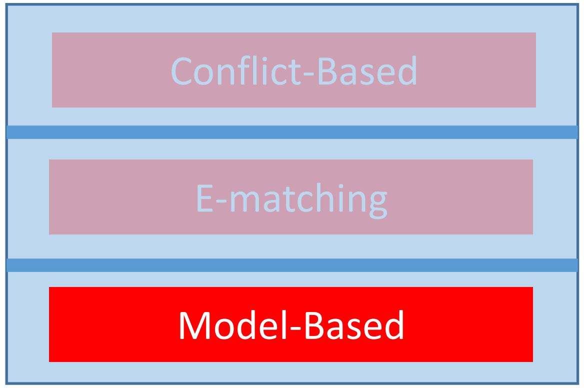
Model-based Instantiation



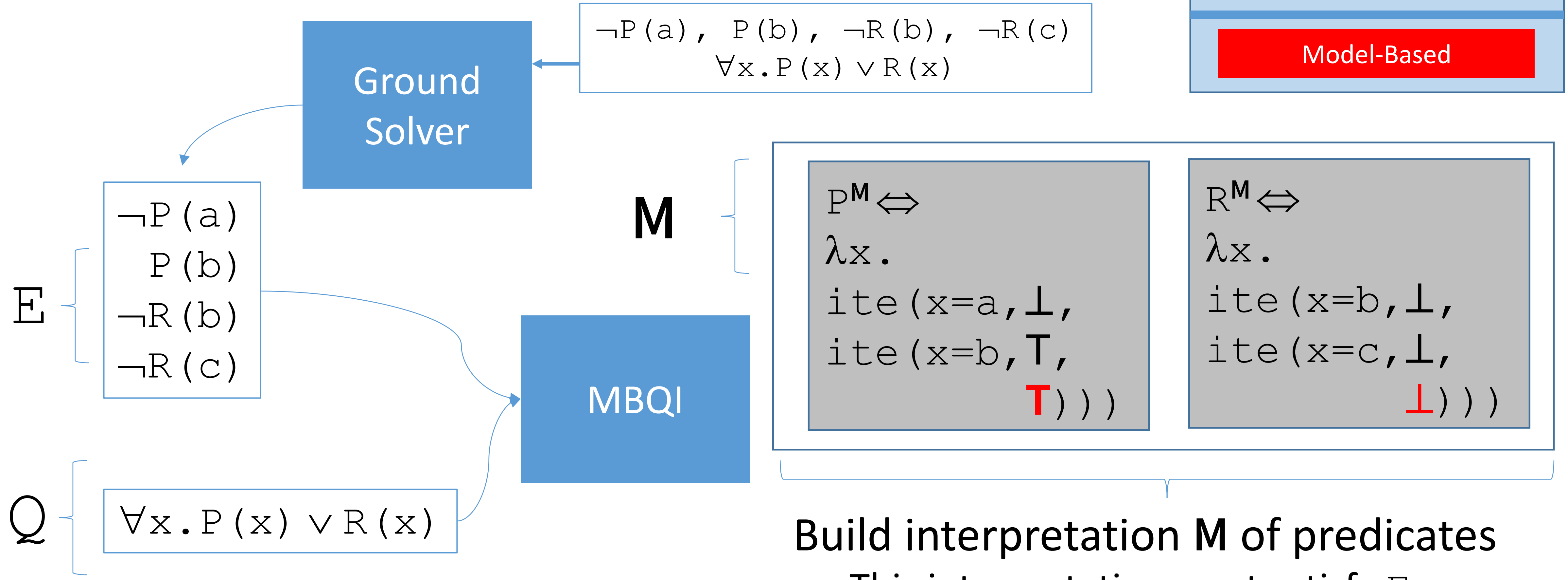


Build interpretation **M** of predicates

- This interpretation must satisfy E



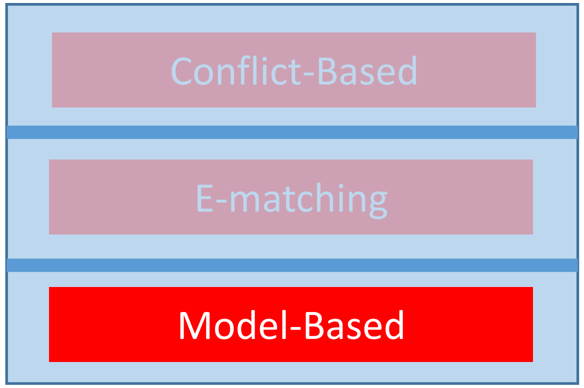
Model-based Instantiation

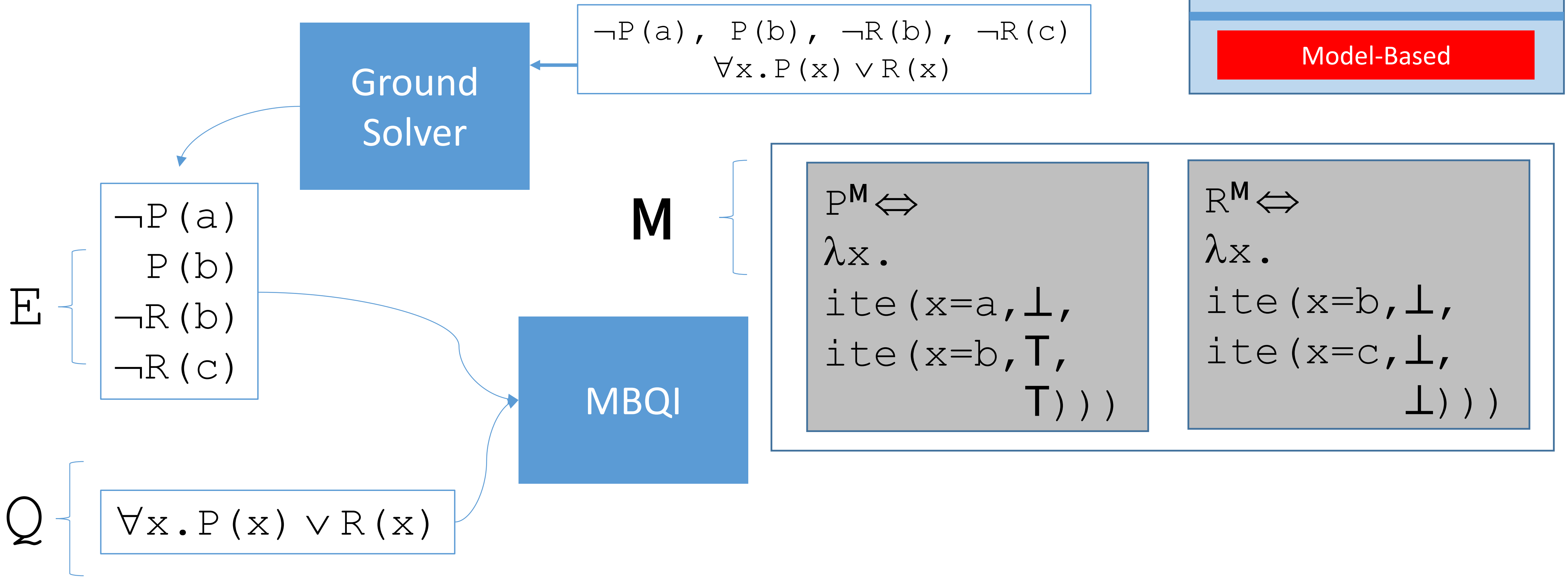


Build interpretation **M** of predicates

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

Model-based Instantiation





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

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

Conflict-Based

E-matching

Model-Based

Model-based Instantiation

$\neg P(a),\ P(b),\ \neg R(b),\ \neg R(c)$
$\forall x.P(x) \vee R(x)$

Ground Solver

E: $\neg P(a)$, $P(b)$, $\neg R(b)$, $\neg R(c)$

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

MBQI

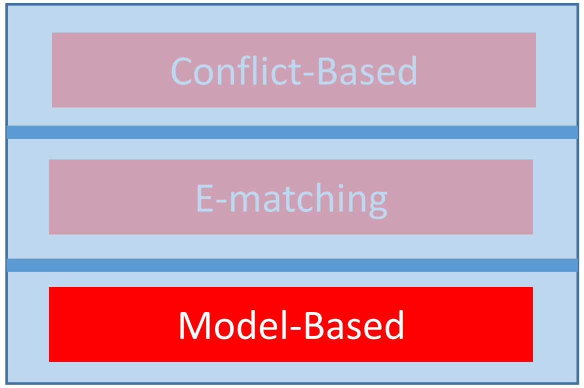
M

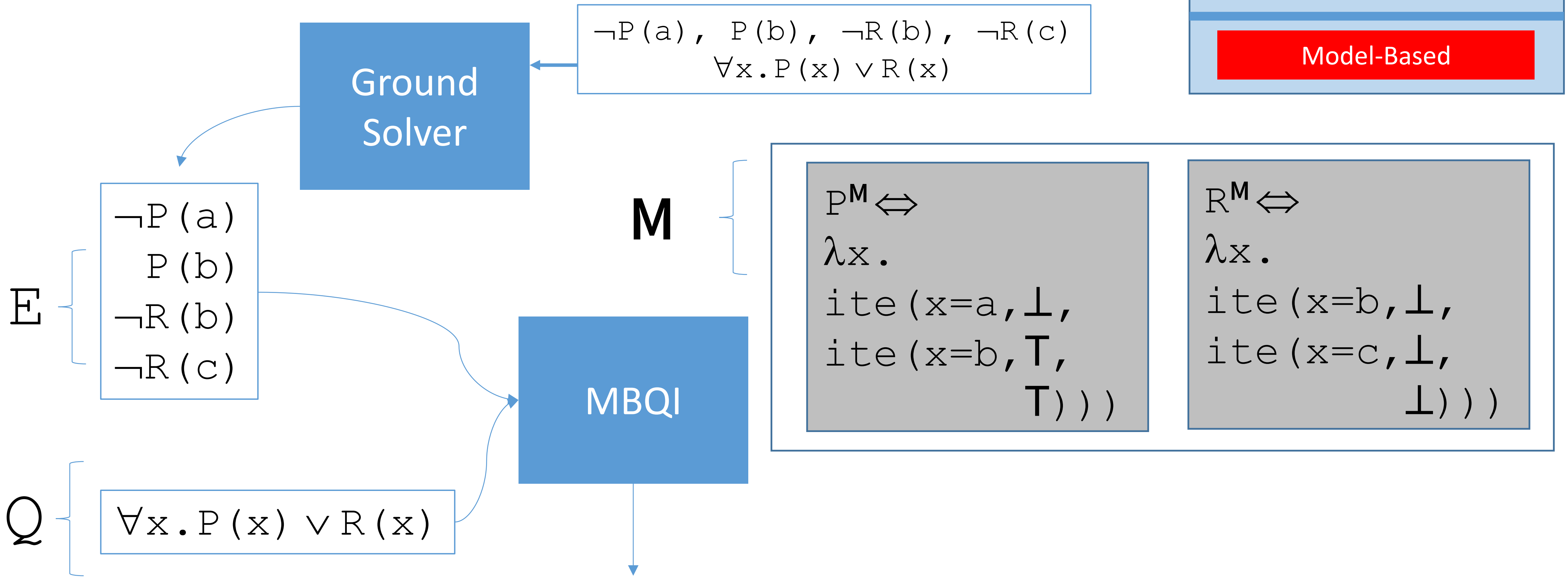
$P^M \Leftrightarrow \lambda x.\ ite(x{=}a, \bot, ite(x{=}b, \top, \top)))$

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

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

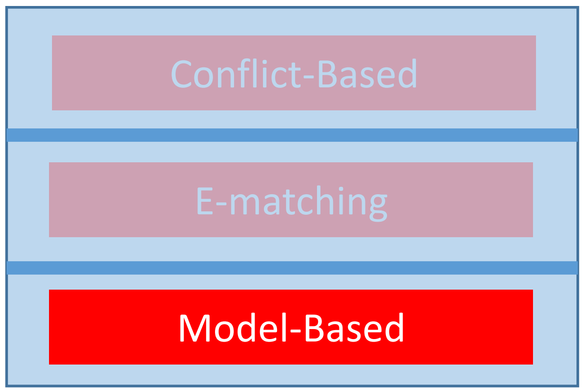
Model-based Instantiation

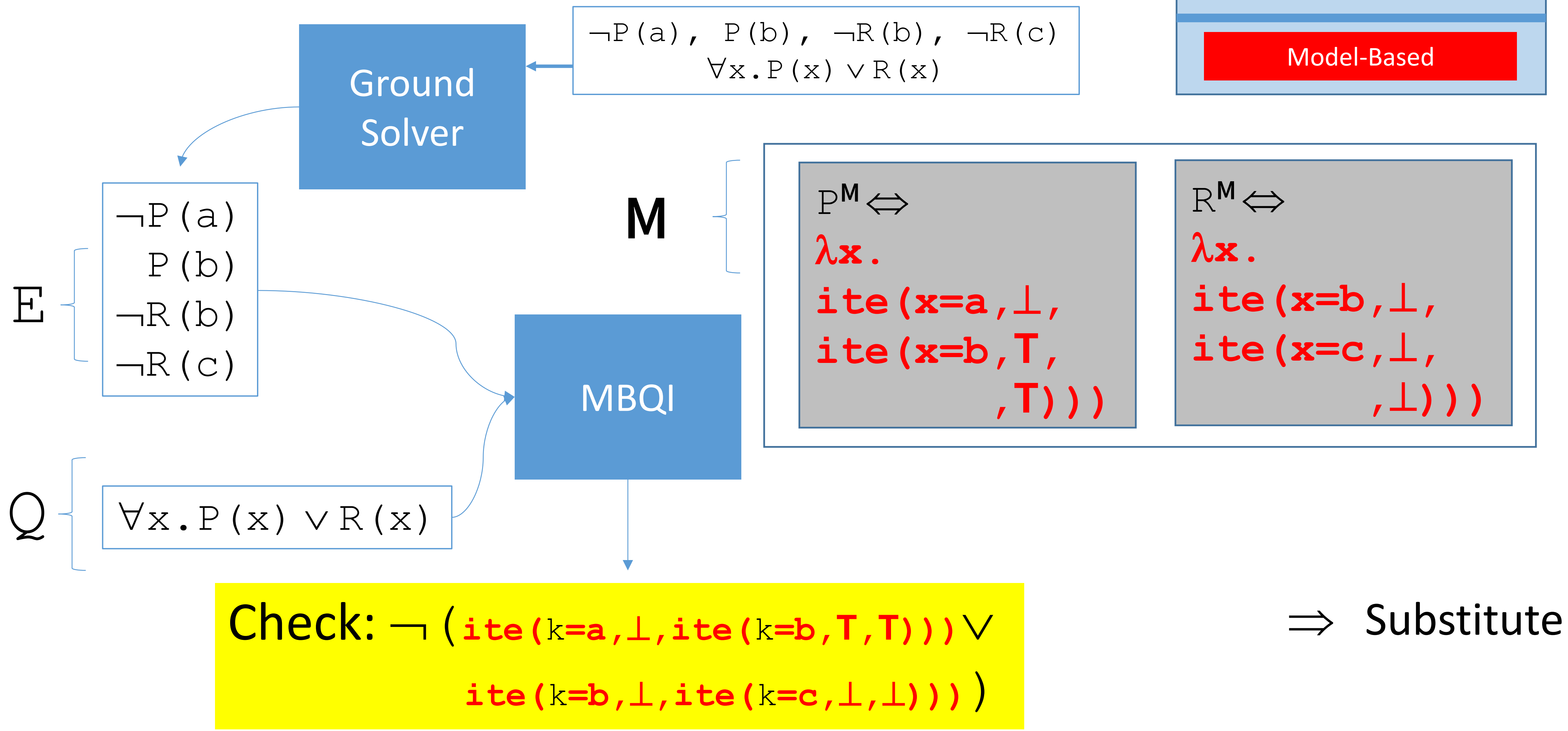




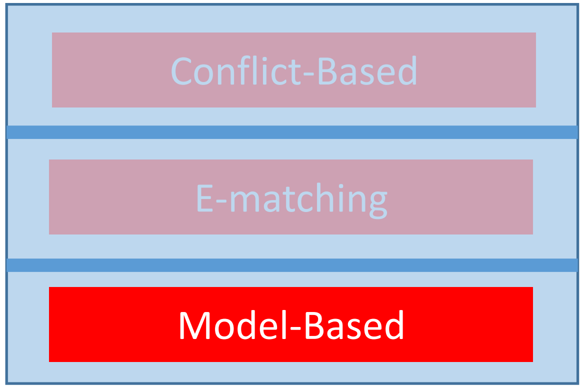
Check: $\neg(P^M(k) \vee R^M(k))$ $\Rightarrow$ Skolemize

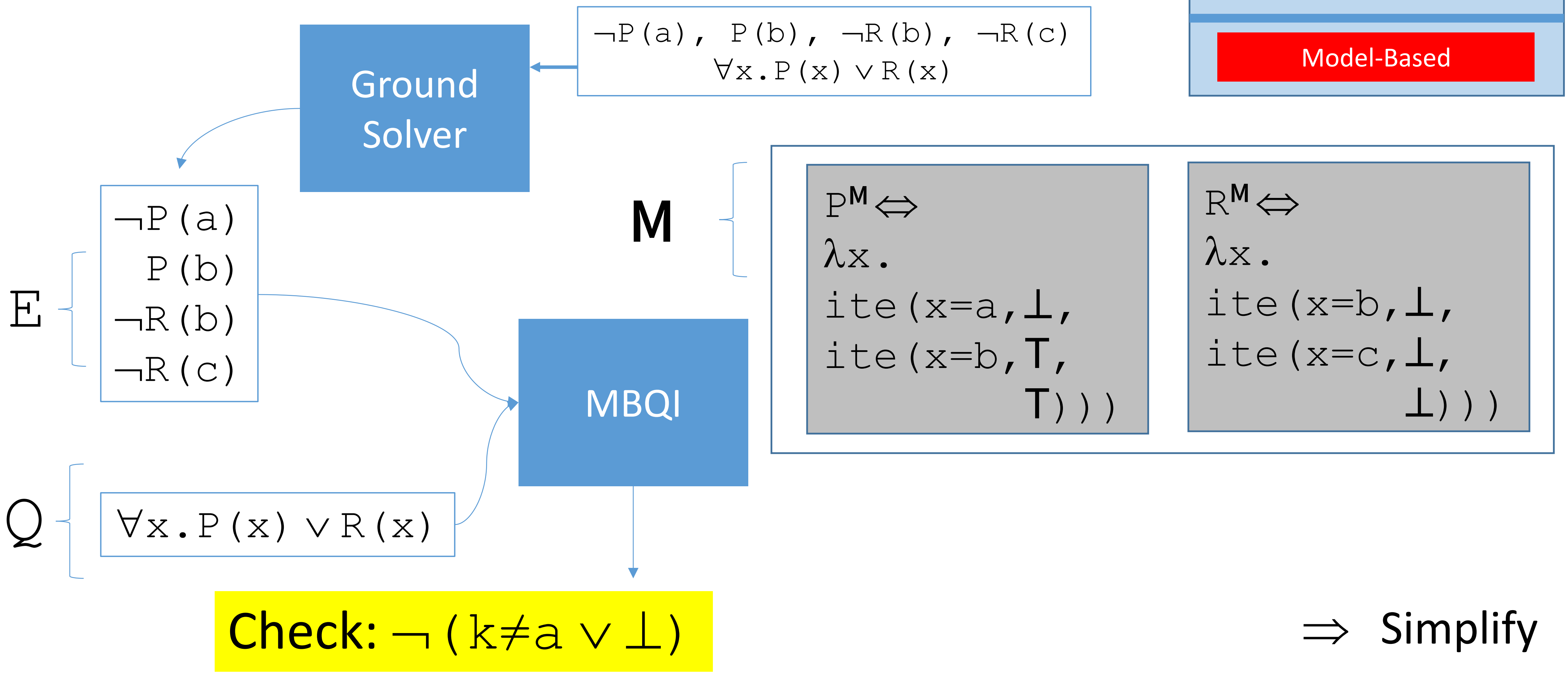
Model-based Instantiation





Model-based Instantiation



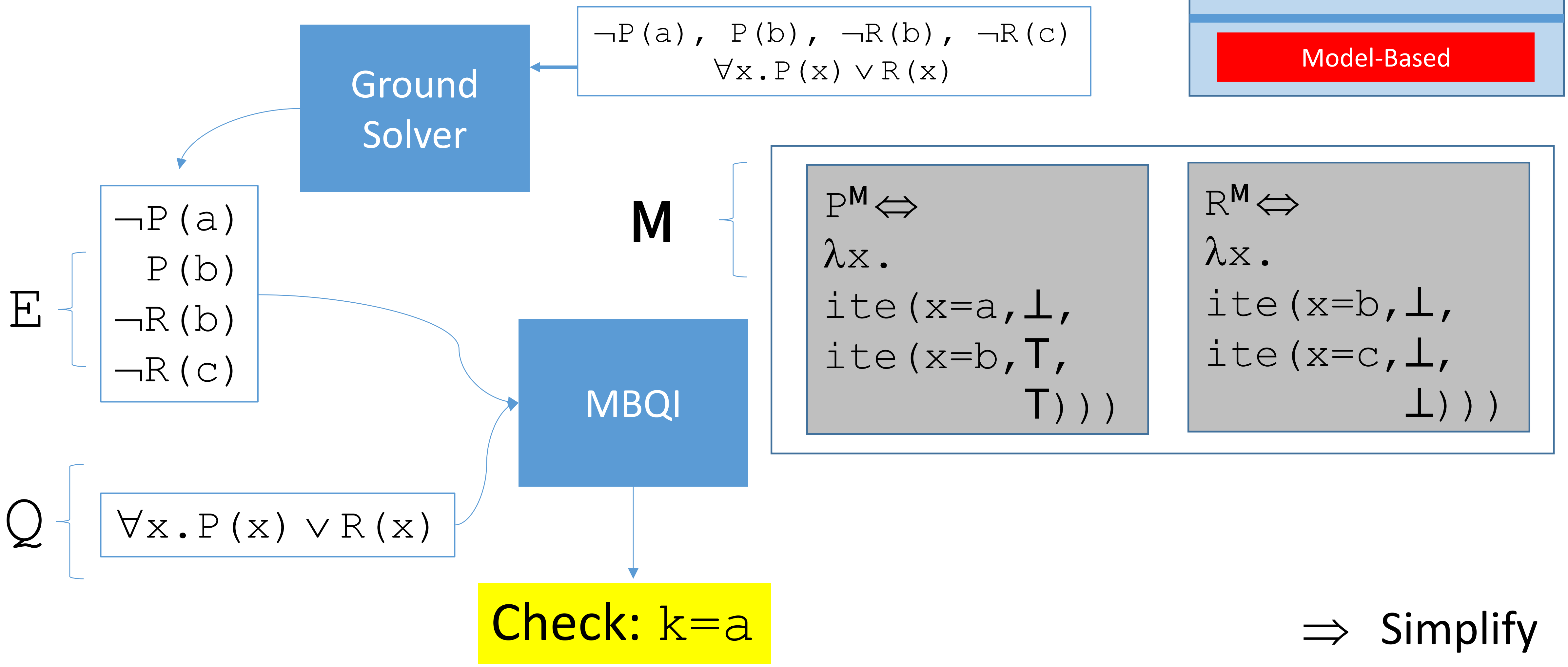


Model-based Instantiation

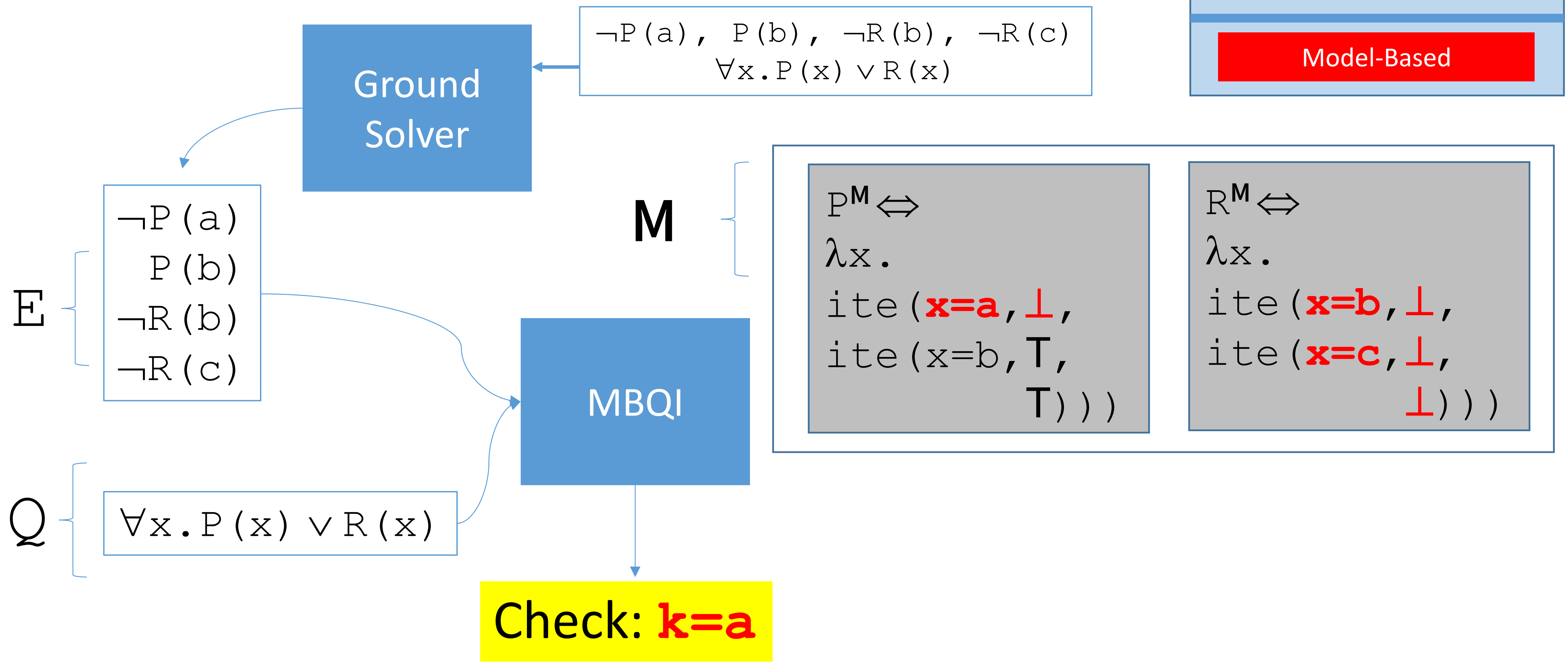
Conflict-Based

E-matching

Model-Based

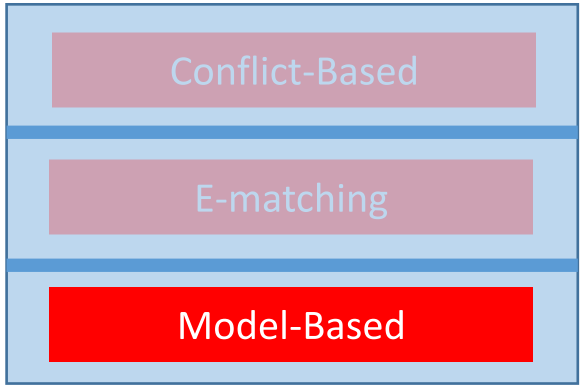


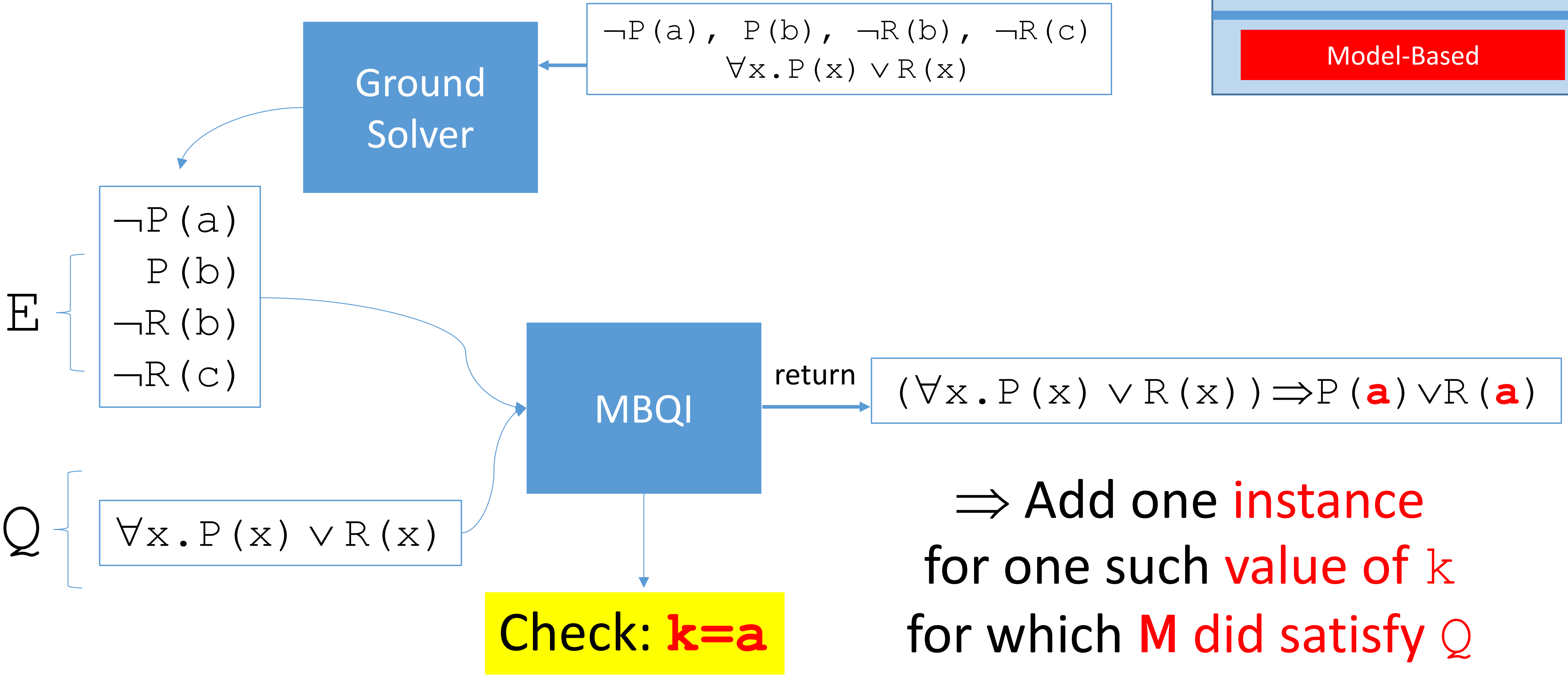
Model-based Instantiation



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

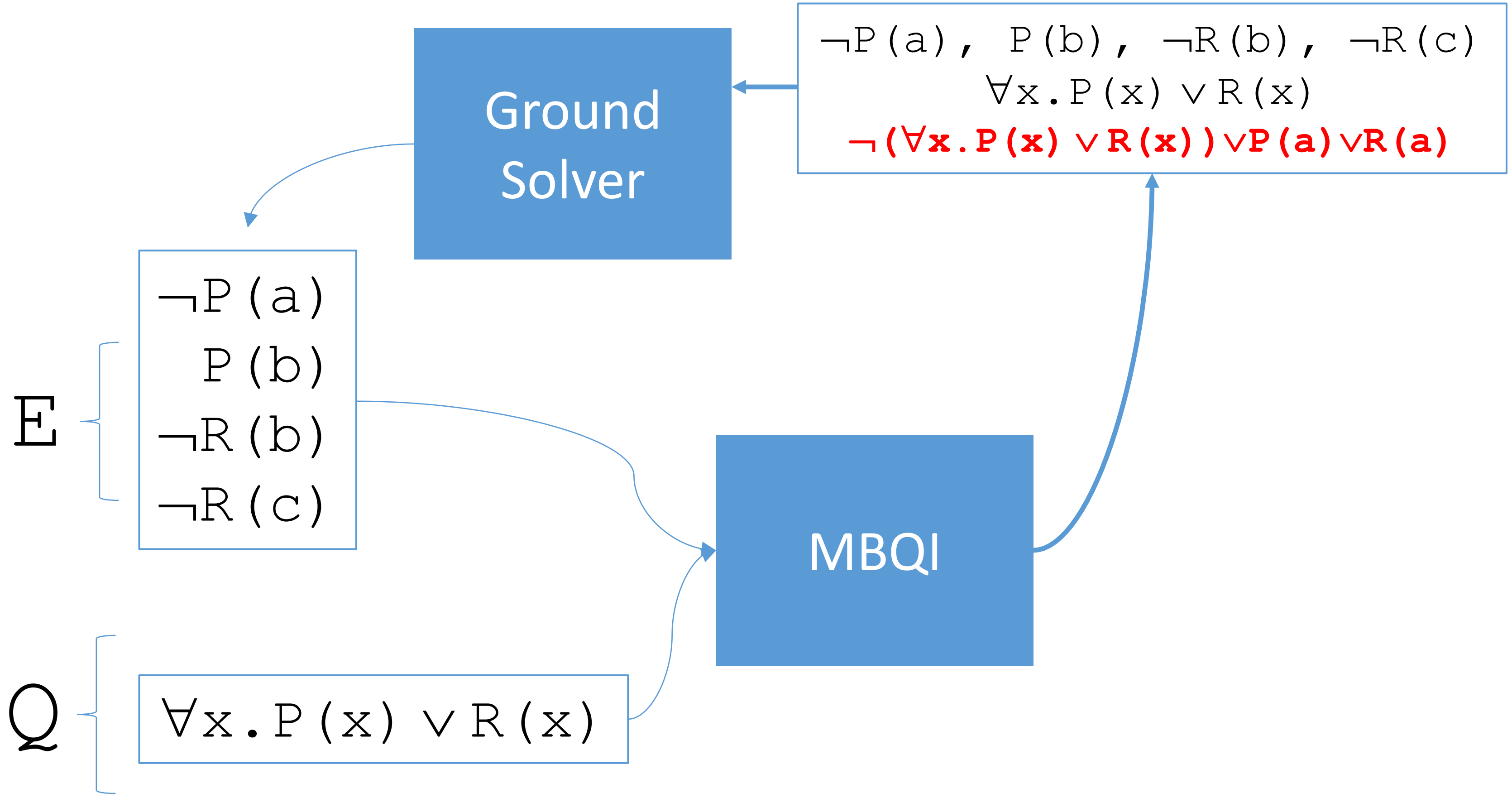
Model-based Instantiation





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

Model-based Instantiation



Conflict-Based

E-matching

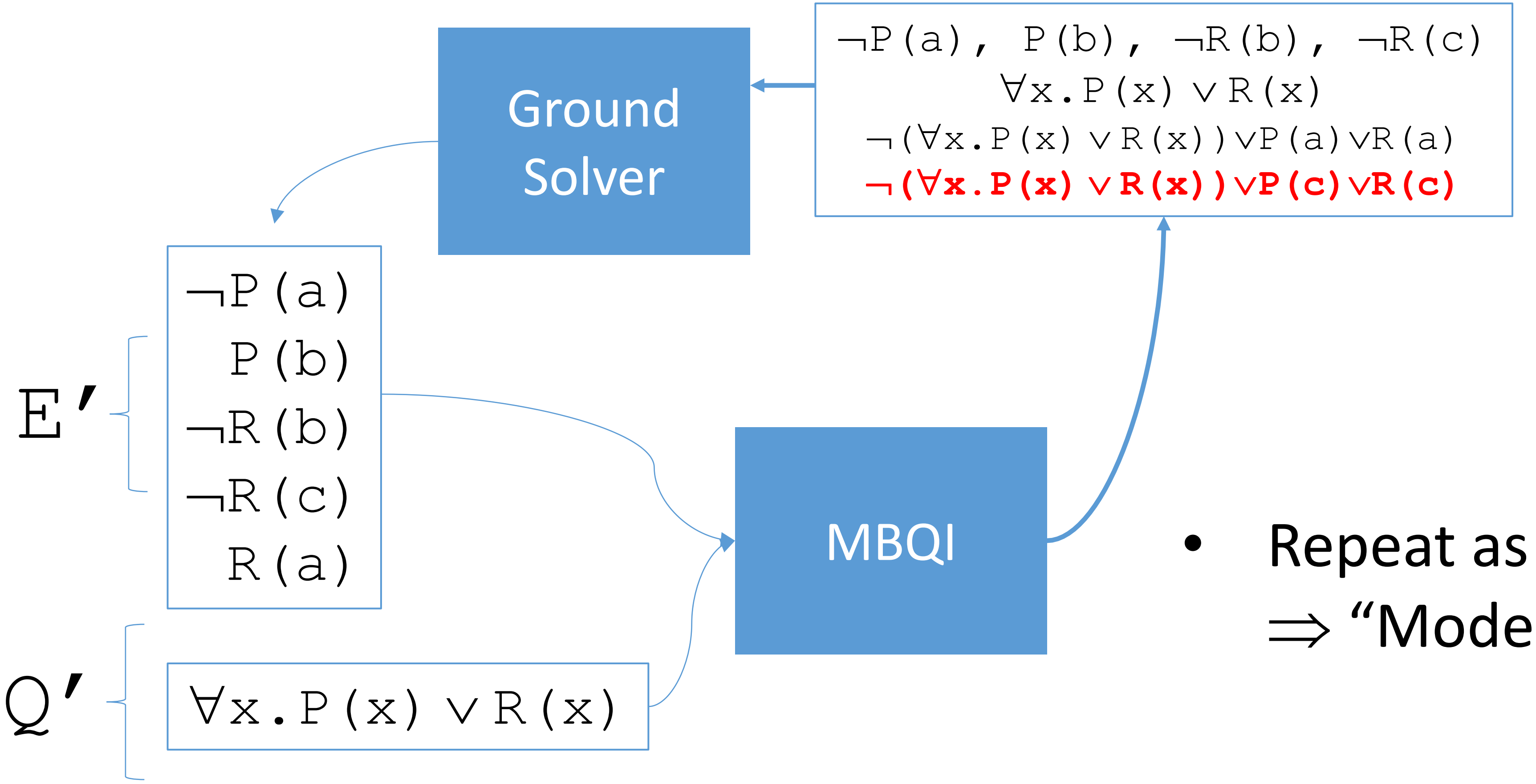
Model-Based

Conflict-Based

E-matching

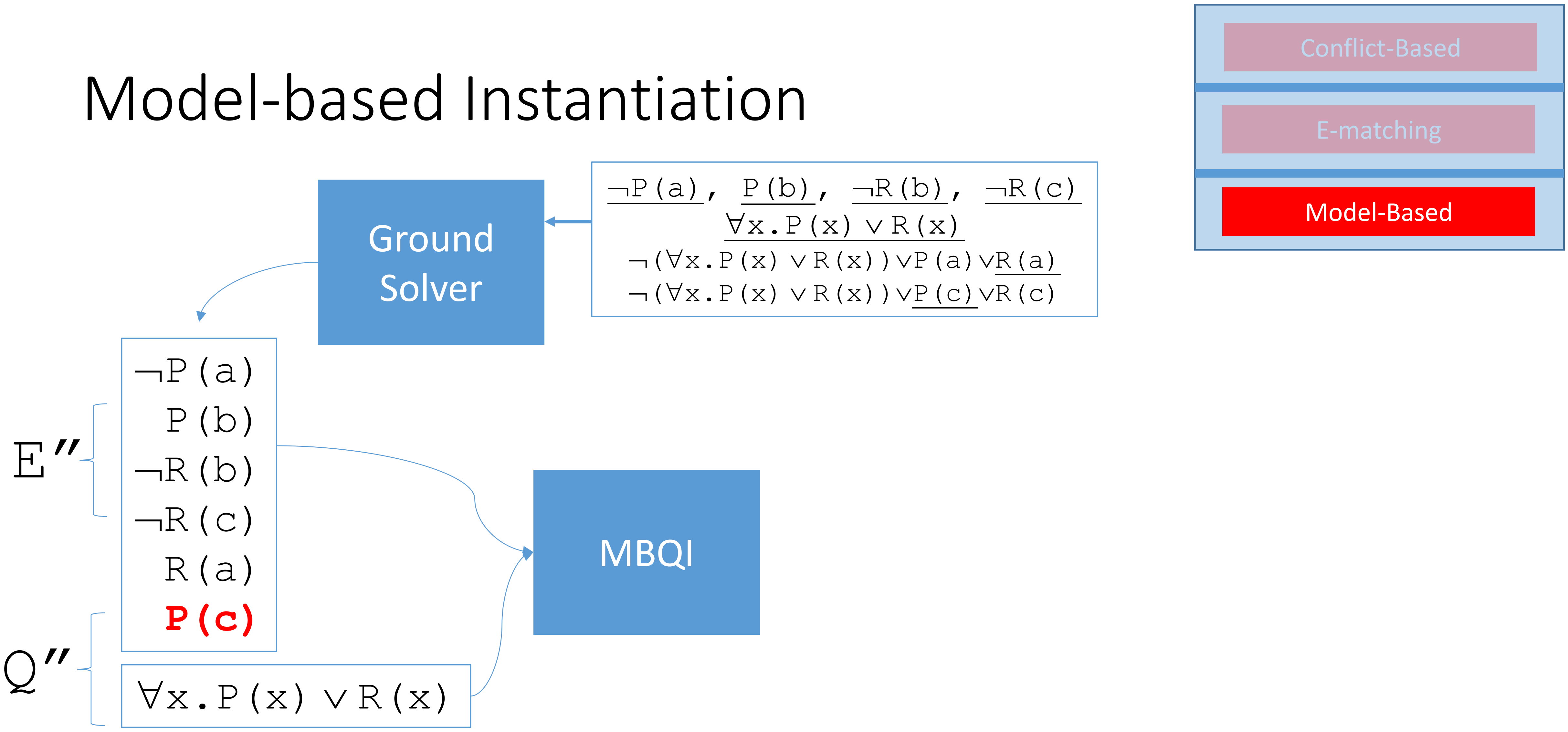
Model-Based

Model-based Instantiation

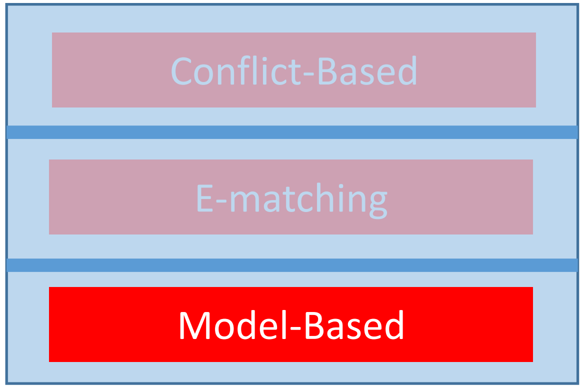


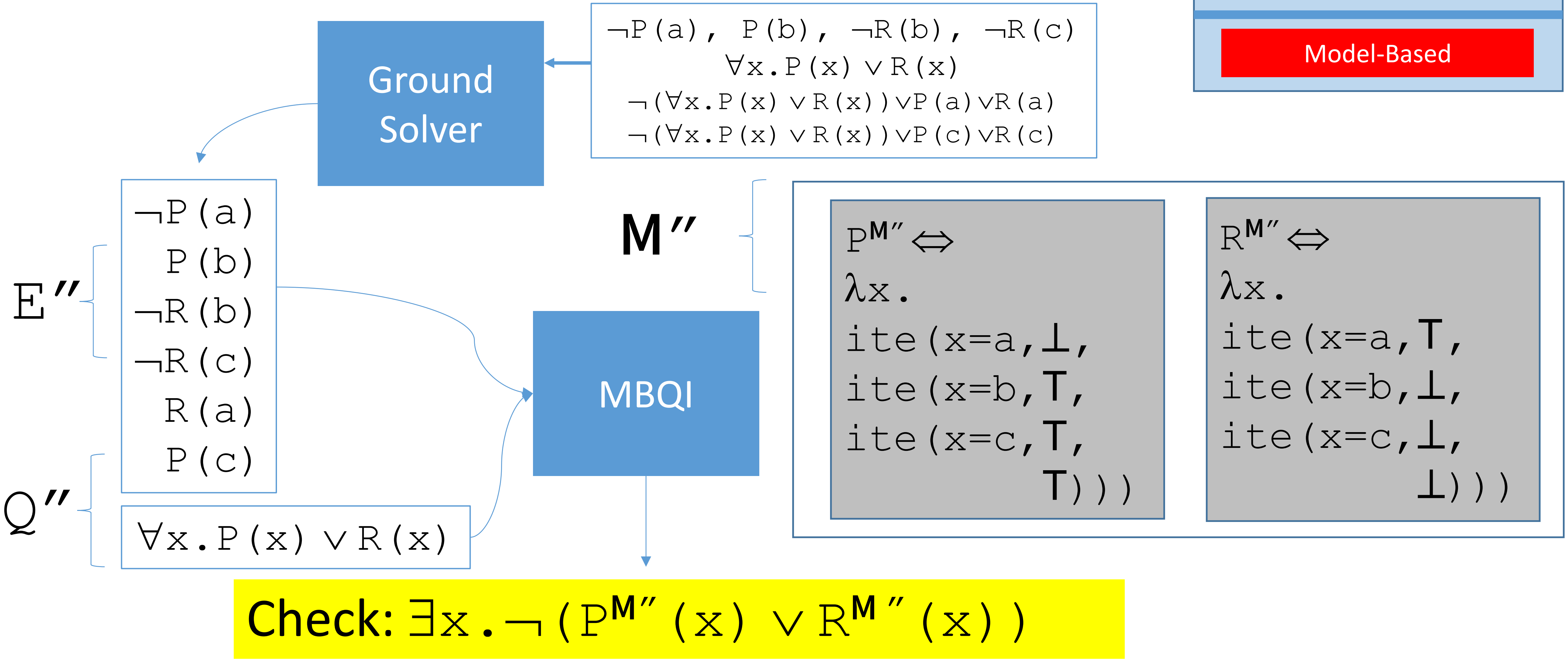
- Repeat as necessary
  ⇒ “Model refinement loop”

Model-based Instantiation

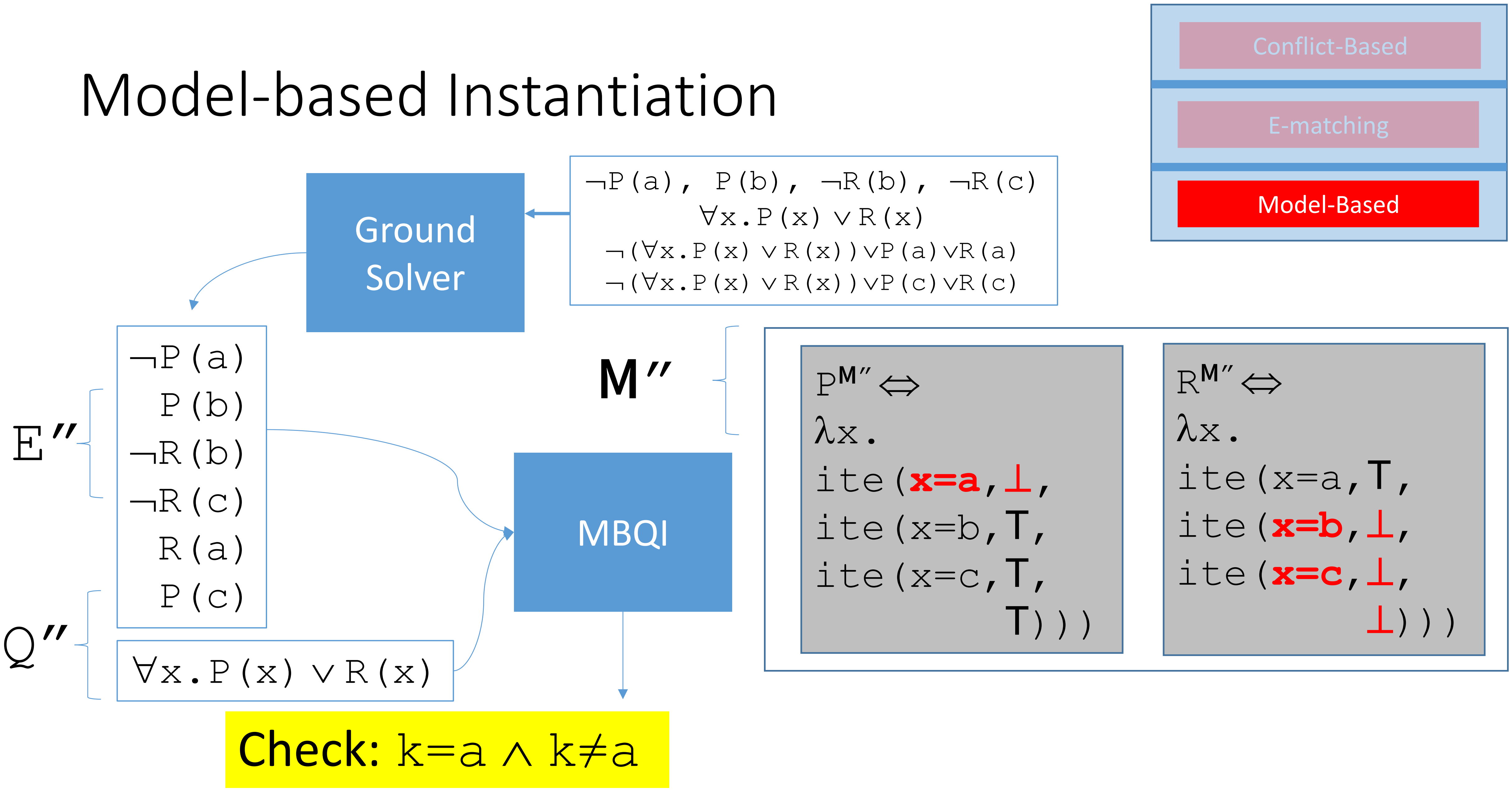


Model-based Instantiation

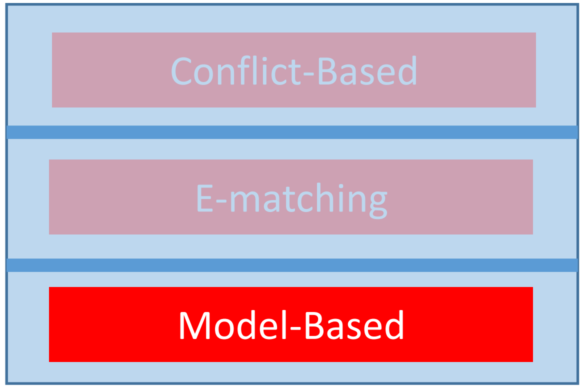


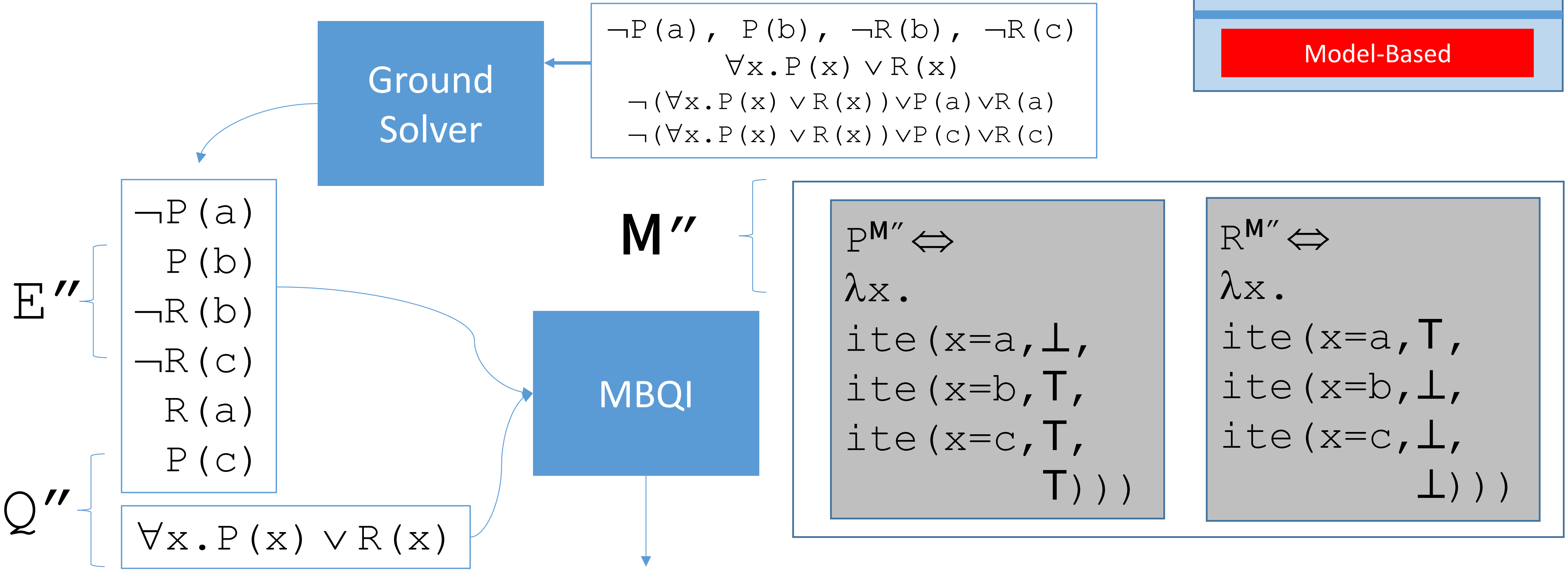


Model-based Instantiation



Model-based Instantiation

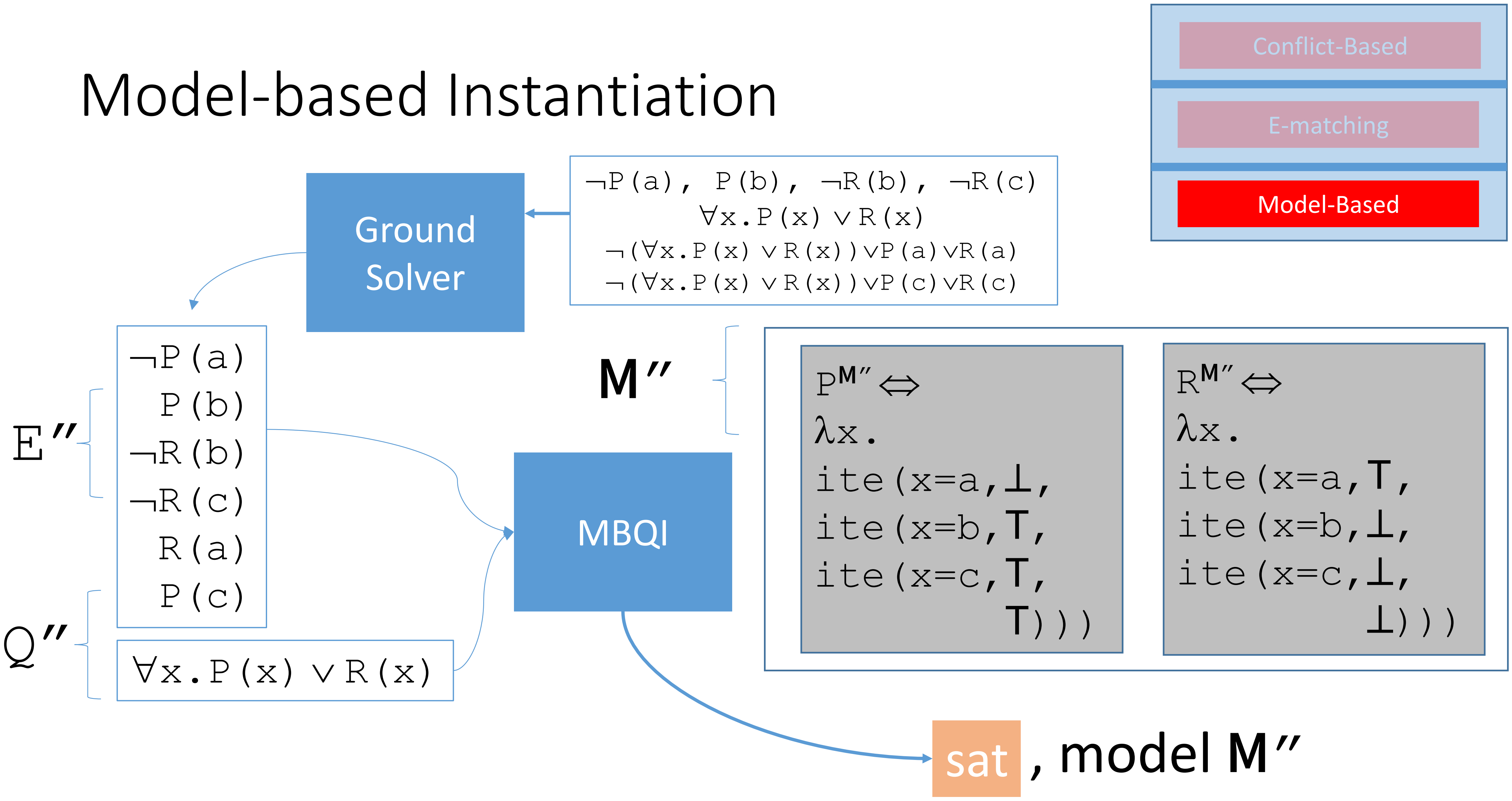




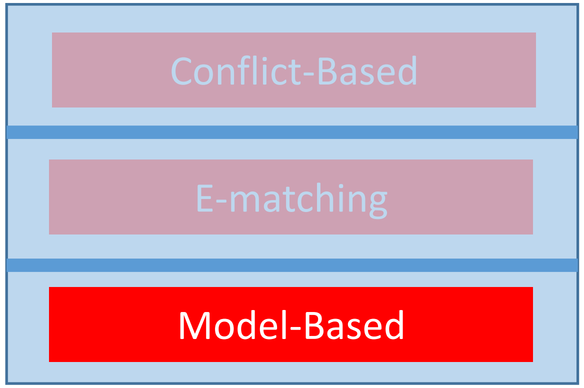
Check: $k=a \wedge k \neq a$

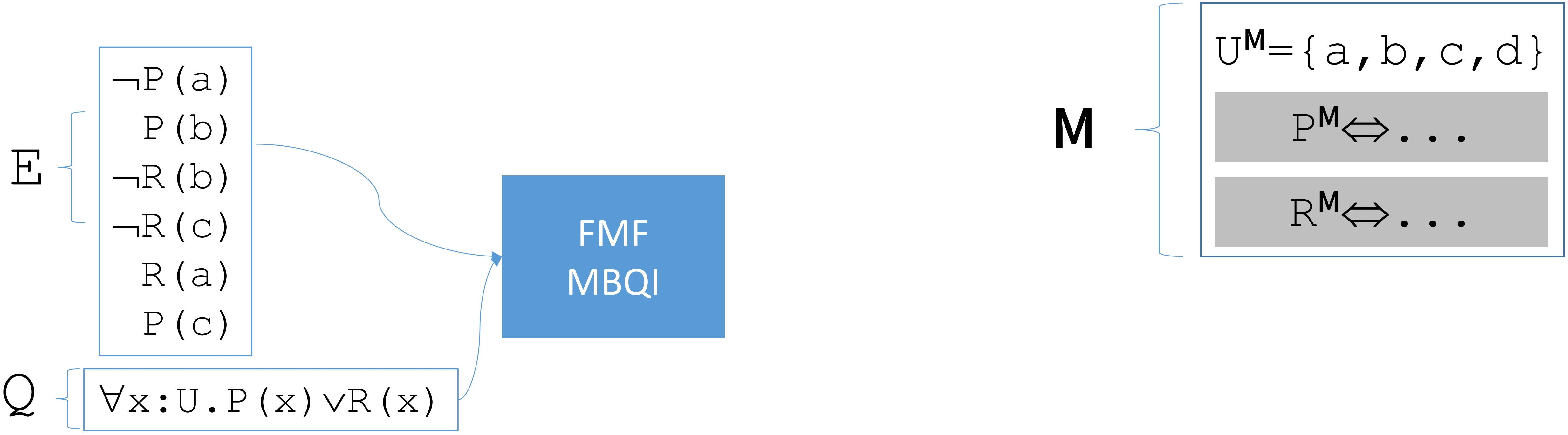
$\Rightarrow$ Unsatisfiable, there are no values k for which **M**″ does not satisfy Q

Model-based Instantiation



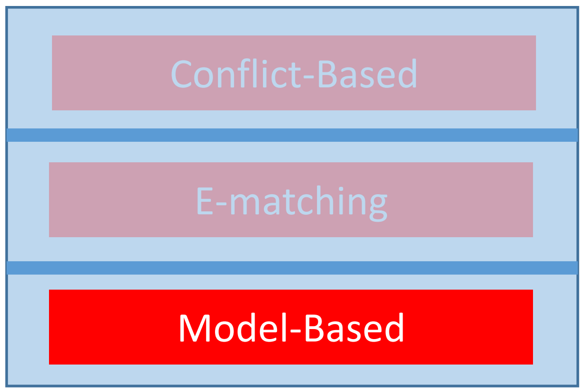
Finite Model Finding in CVC4

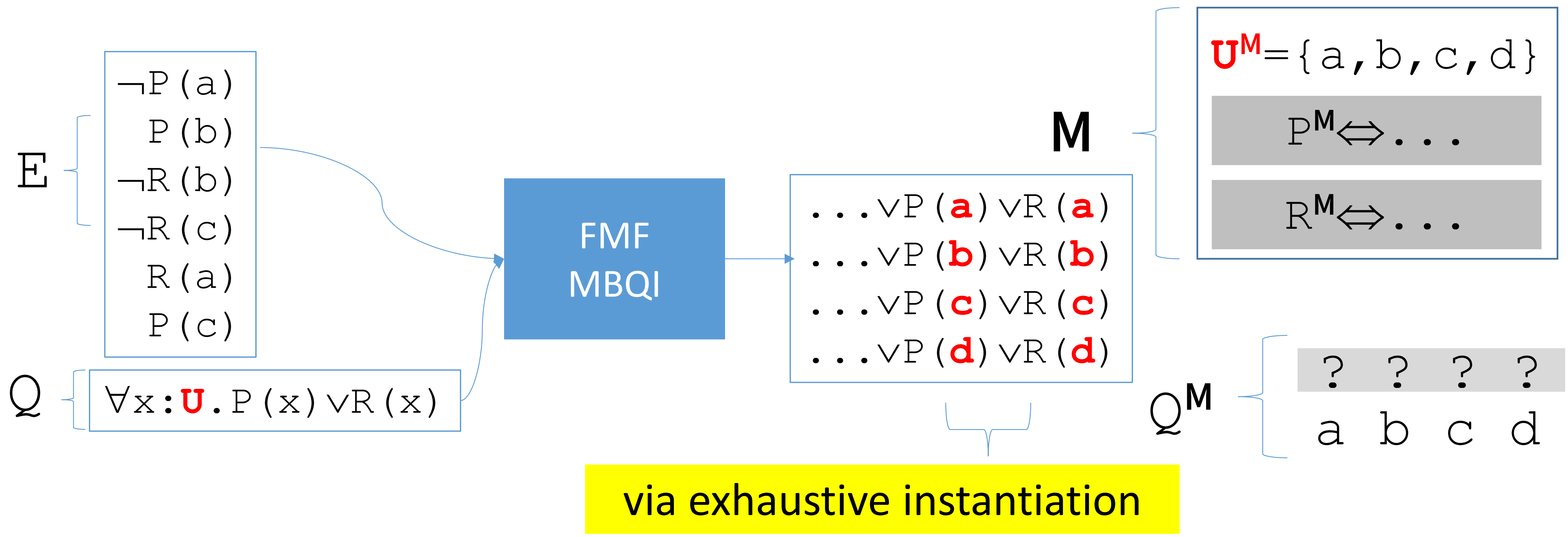




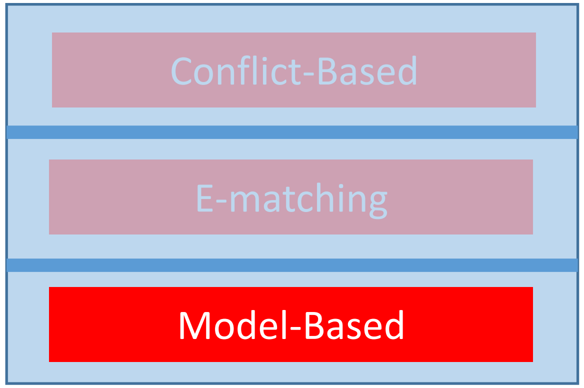
In CVC4, model-based Instantiation used for improving scalability of FMF

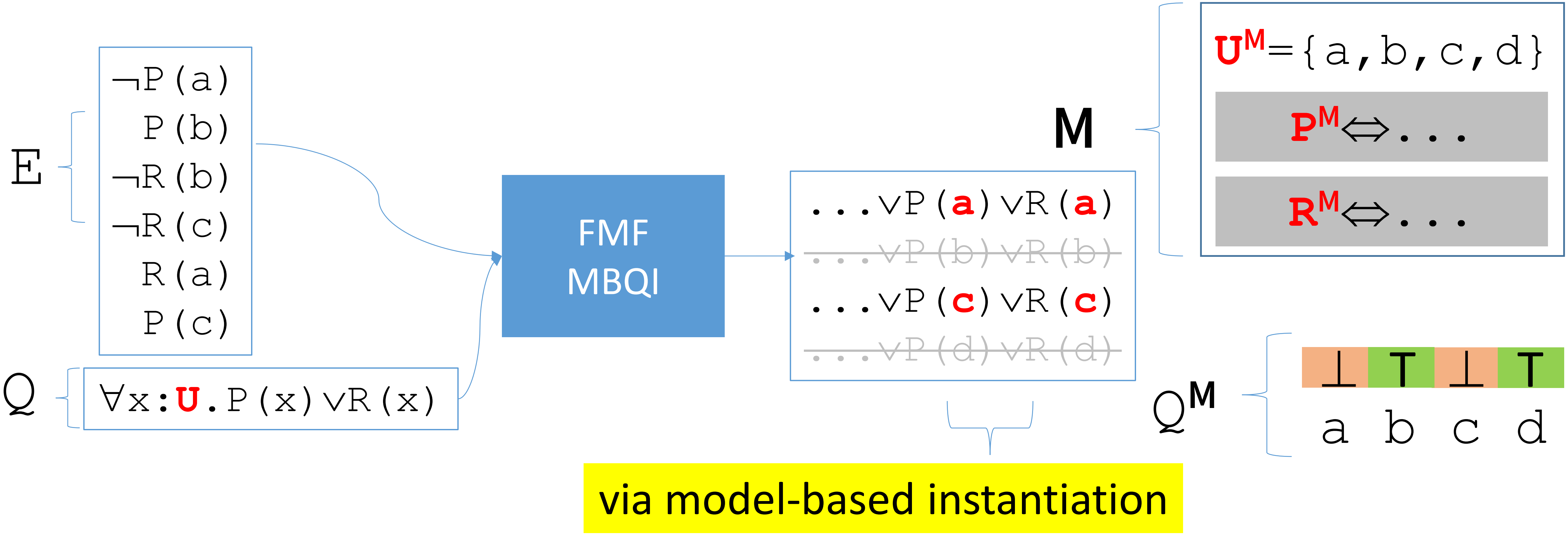
Finite Model Finding in CVC4





Finite Model Finding in CVC4



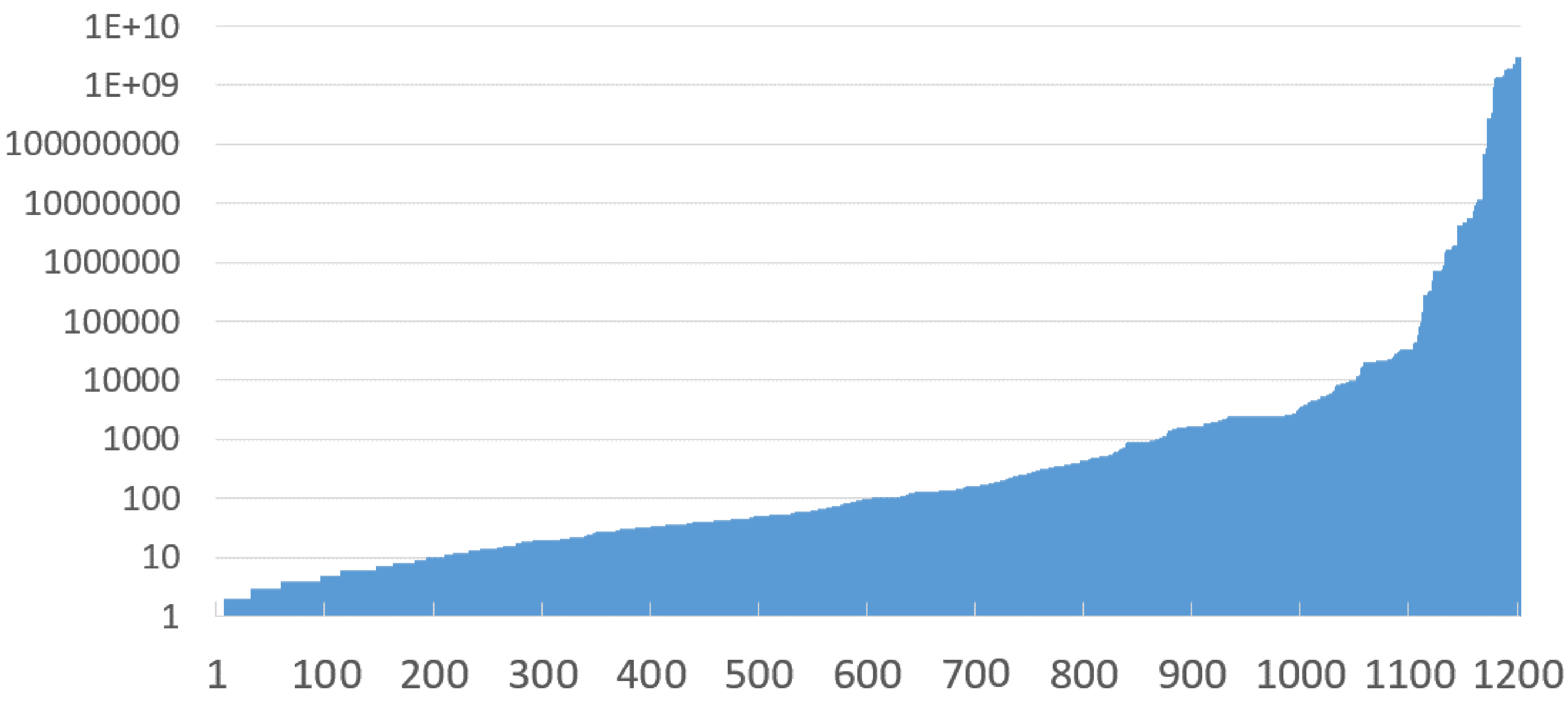


Conflict-Based

E-matching

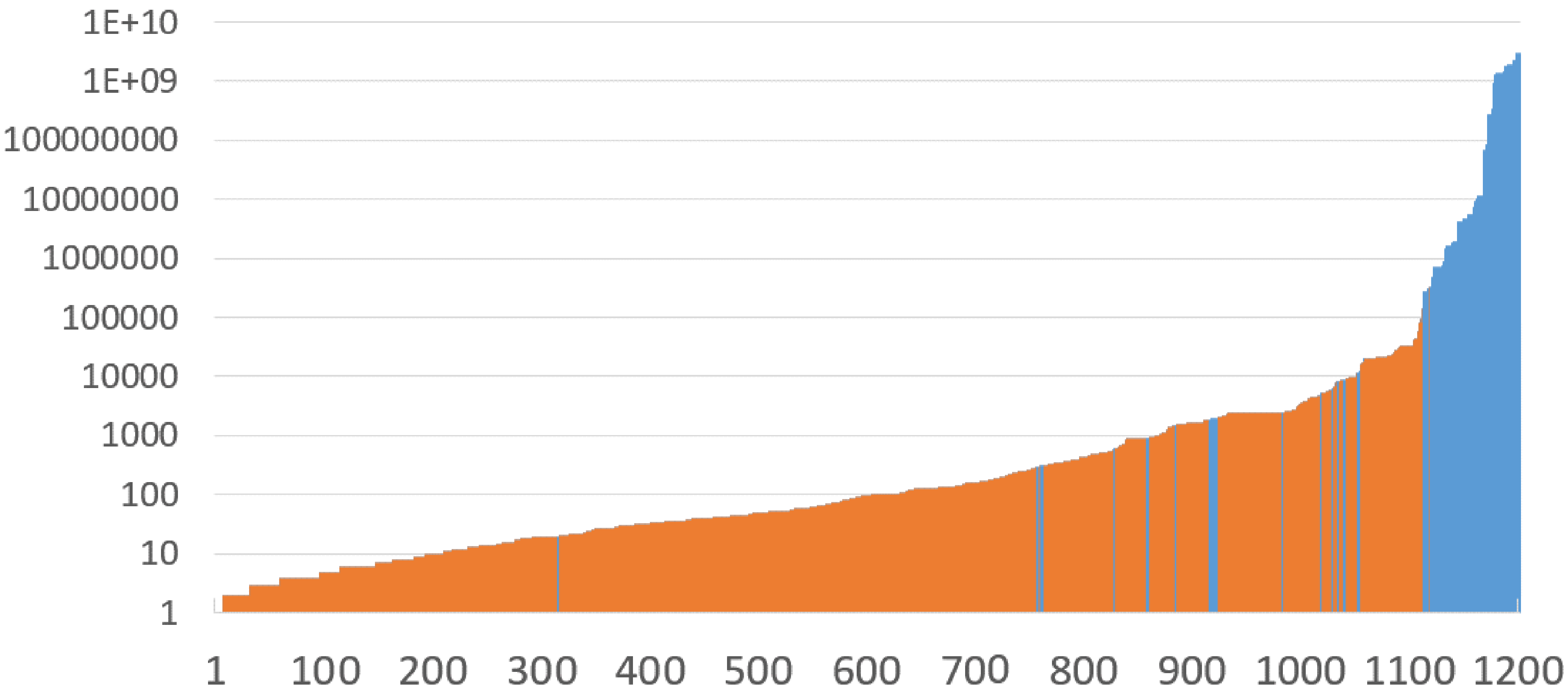
Model-Based

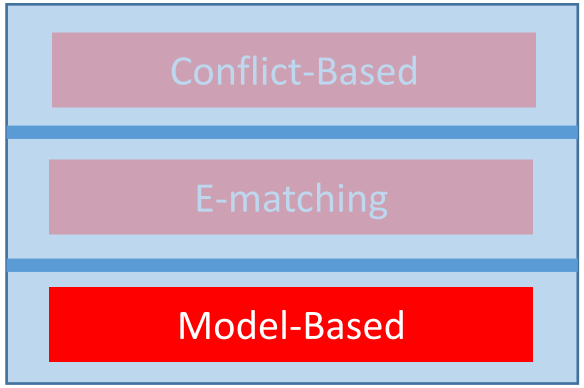
Model-based Instantiation: Impact



- 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

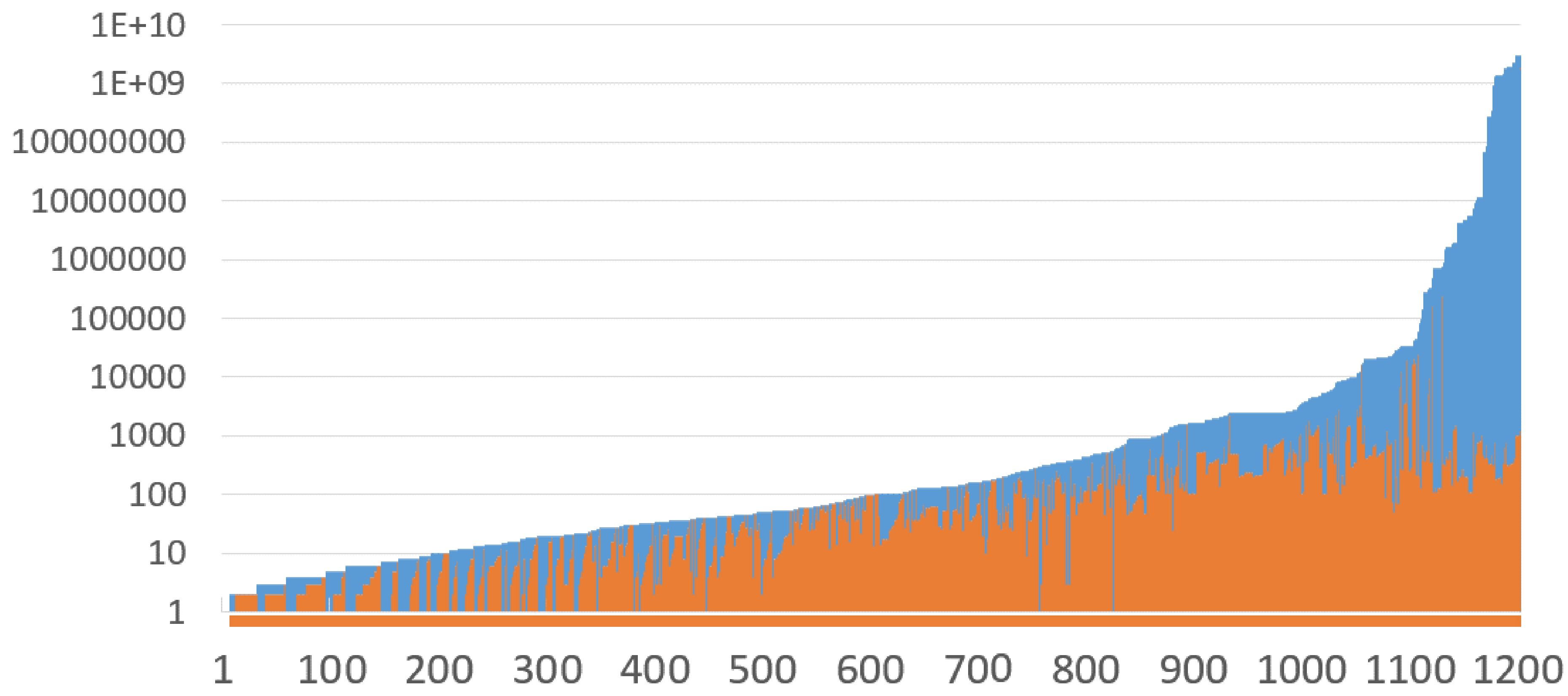
Model-based Instantiation: Impact

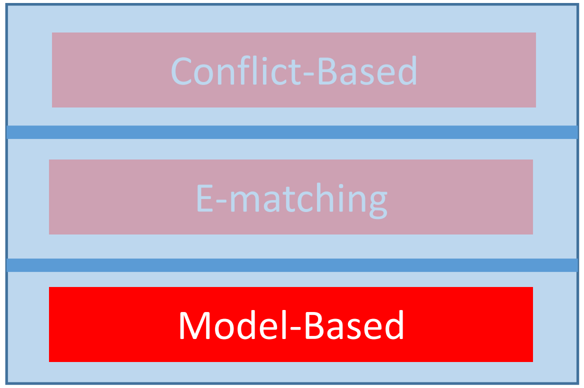




- CVC4 Finite Model Finding + Exhaustive instantiation
    - Scales only up to ~150k instances with a 30 sec timeout

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

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, …

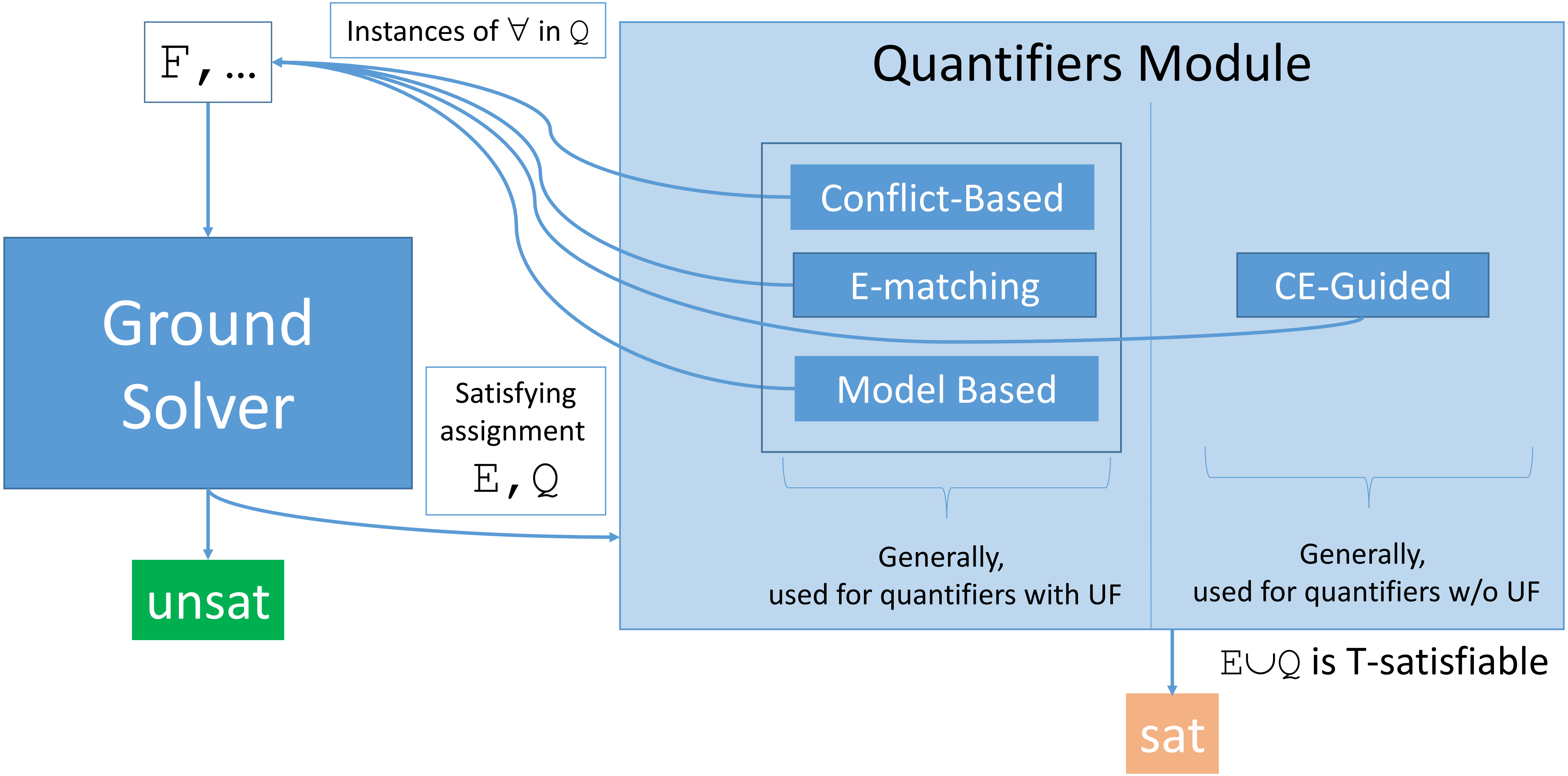
E-matching, Conflict-Based, Model-based:

- **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

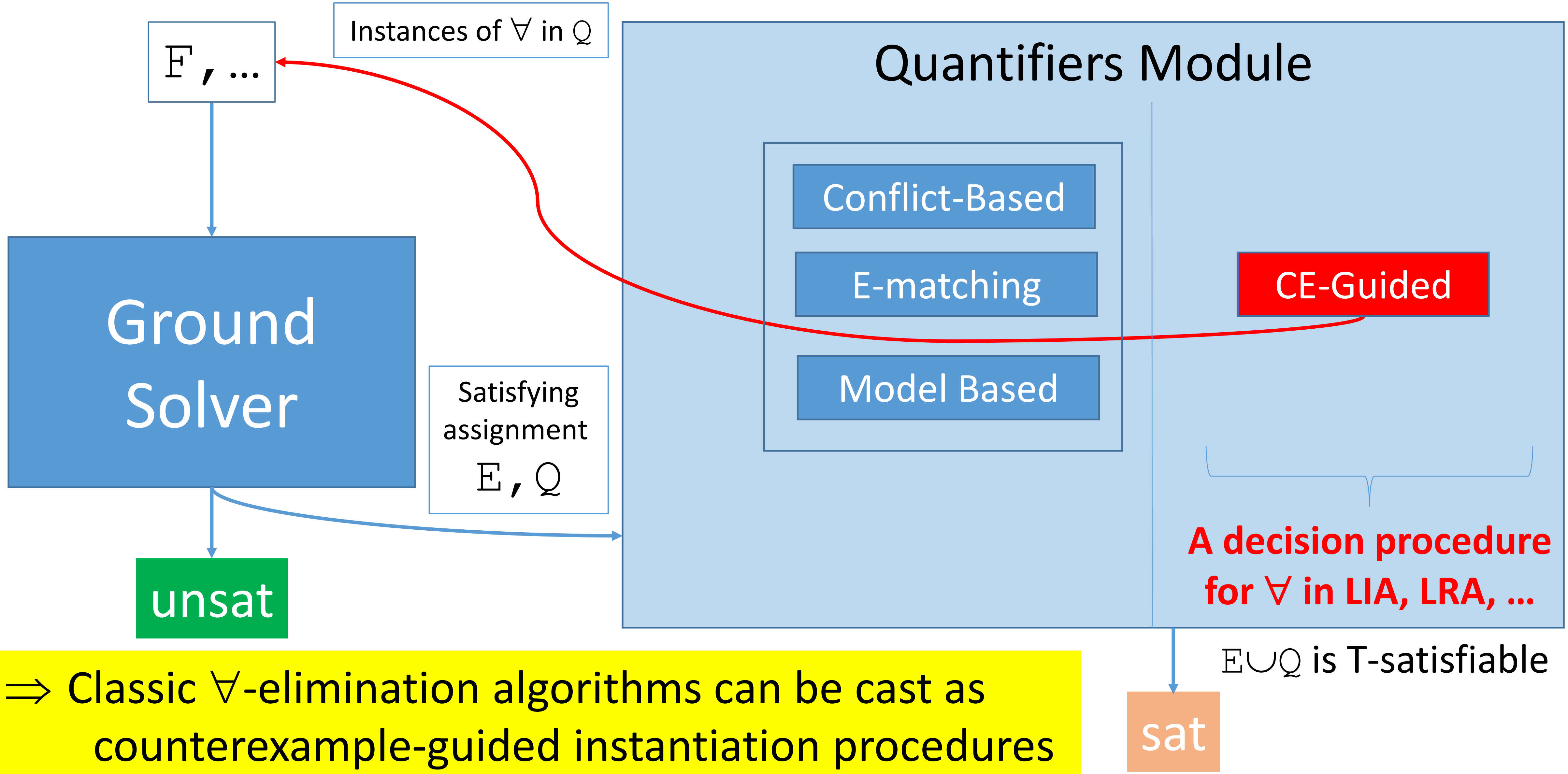
⇒ But reasoning about ∀ + *pure* theories isn't as bad:

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

Techniques for Quantifier Instantiation

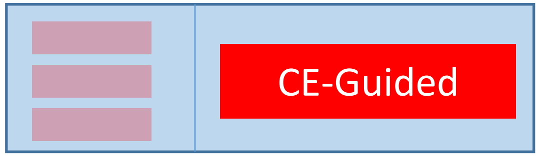


Techniques for Quantifier Instantiation



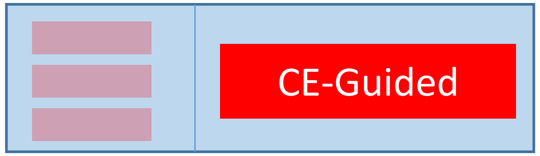
⇒ Classic ∀-elimination algorithms can be cast as counterexample-guided instantiation procedures

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

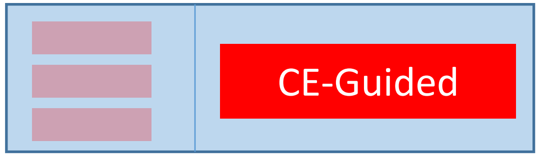
Counterexample-Guided Instantiation



- 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: $\forall \mathbf{x}. \neg\psi[\mathbf{x}] \Leftrightarrow \neg\psi[\mathbf{t}_1] \wedge \ldots \wedge \neg\psi[\mathbf{t}_n]$ for finite n

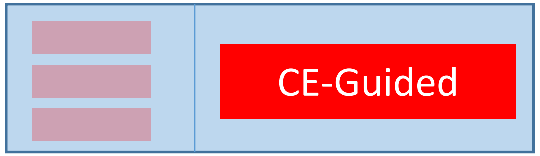
(consider the dual)

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: $\forall x . \neg\psi[x] \Leftrightarrow \neg\psi[t_1] \wedge \ldots \wedge \neg\psi[t_n]$ 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 $m \ll n$ instances

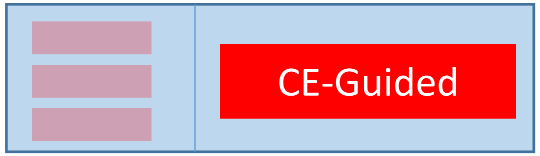
Counterexample-Guided Instantiation

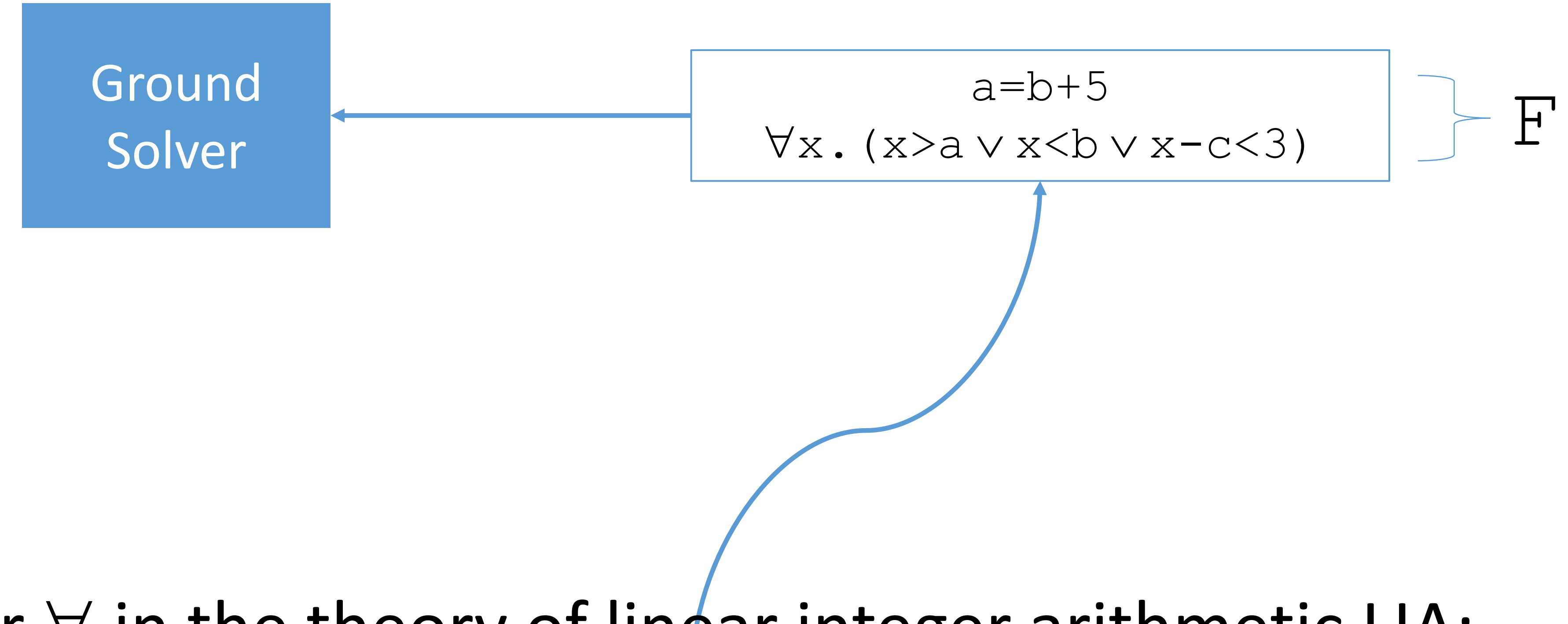


$\Rightarrow$ **Consider $\forall$ in the theory of linear integer arithmetic LIA:**

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

Counterexample-Guided Instantiation



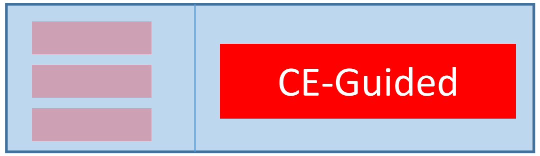


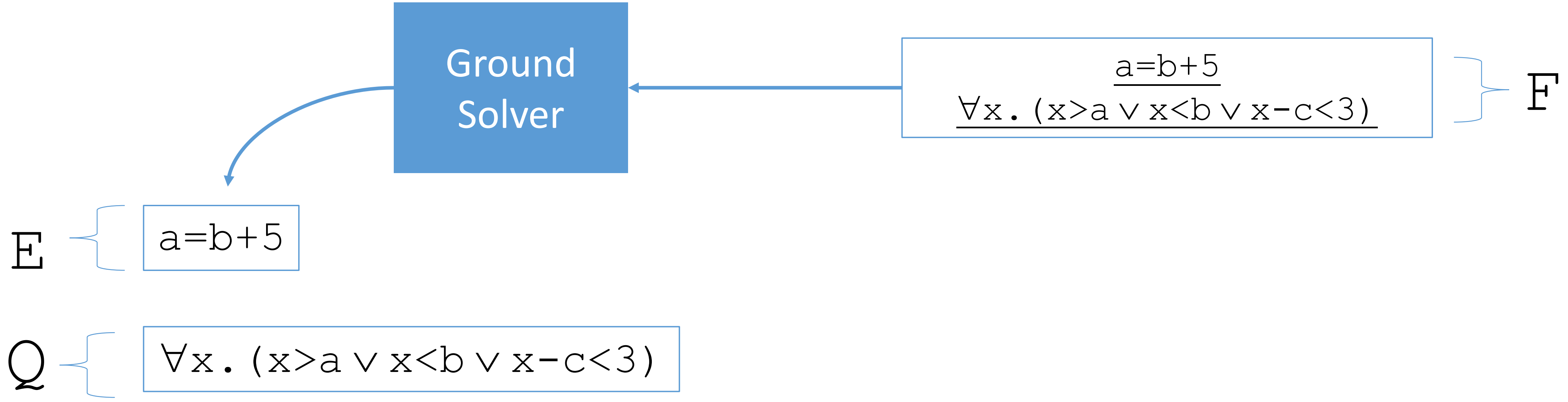
⇒ Consider ∀ in the theory of linear integer arithmetic LIA:

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

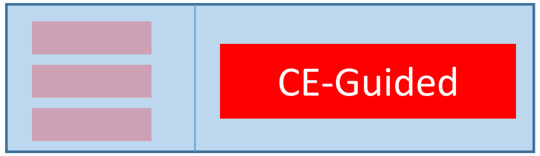
- Outermost existentials `a`, `b`, `c` are treated as *free constants*

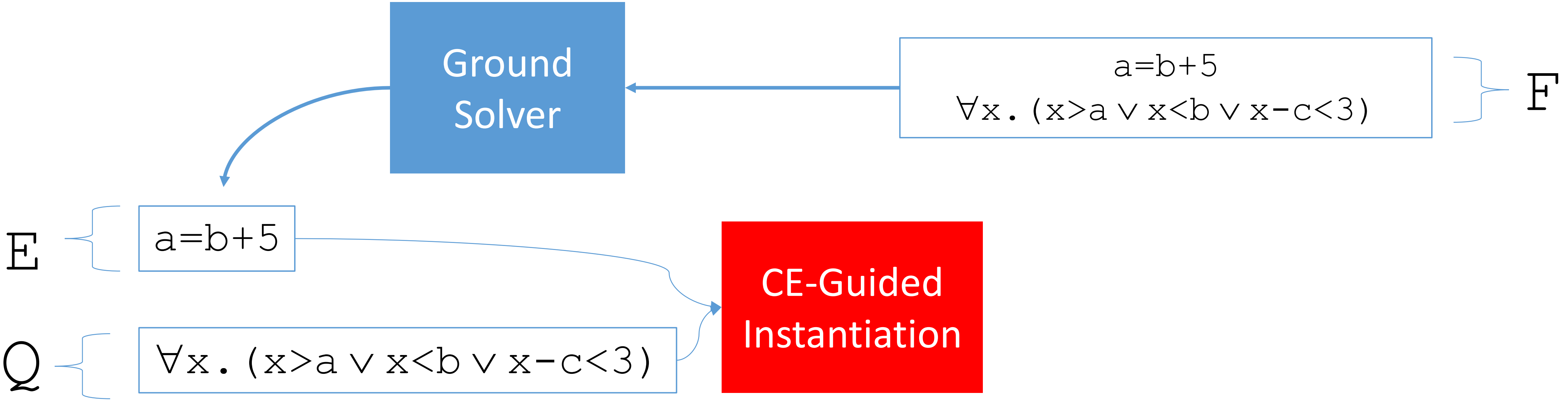
Counterexample-Guided Instantiation





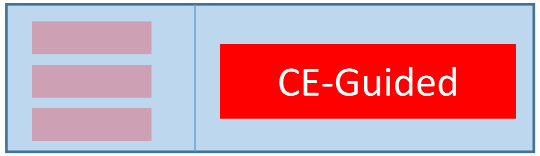
Counterexample-Guided Instantiation

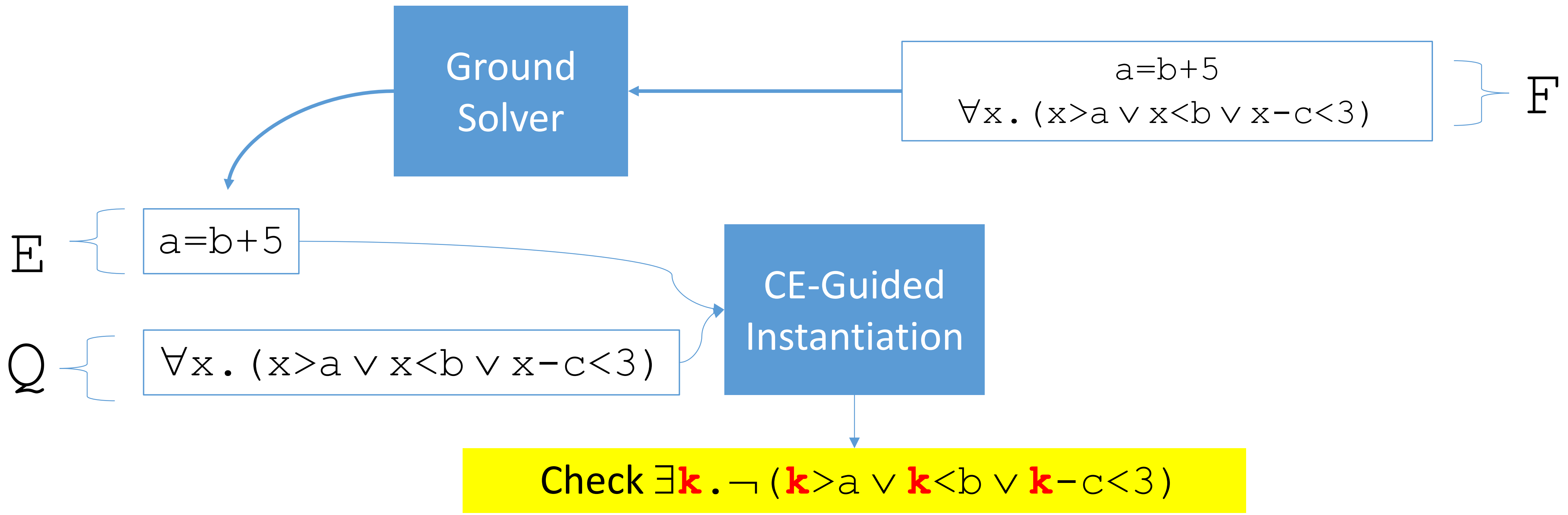




$\Rightarrow$ Use counterexample-guided instantiation

Counterexample-Guided Instantiation

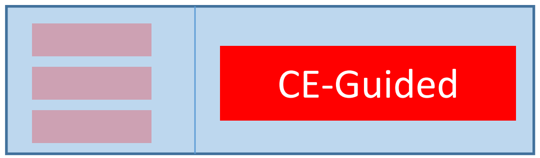


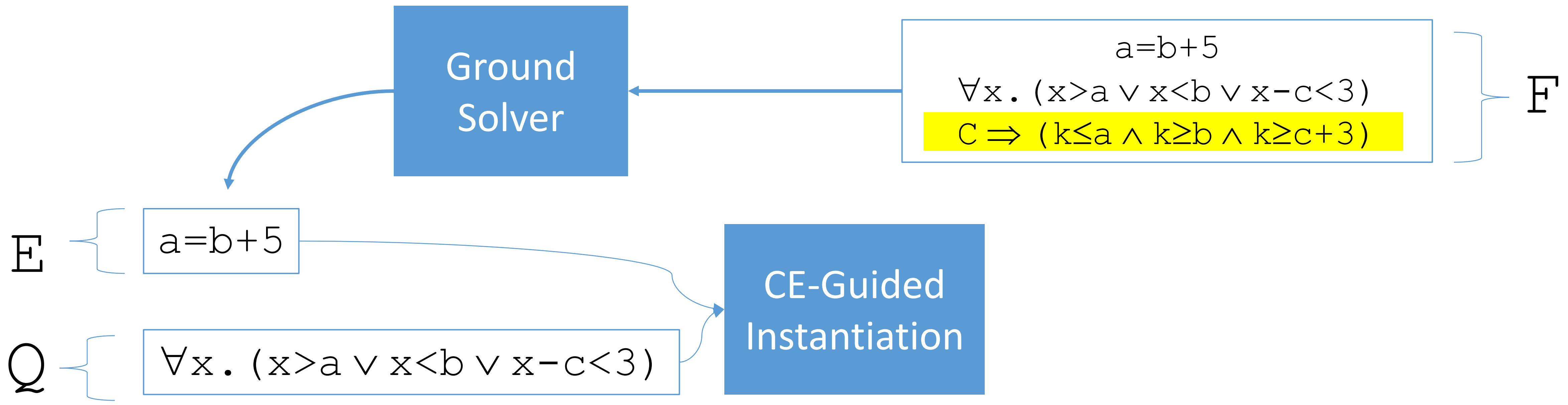


⇒With respect to *model-based instantiation*:

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

Counterexample-Guided Instantiation

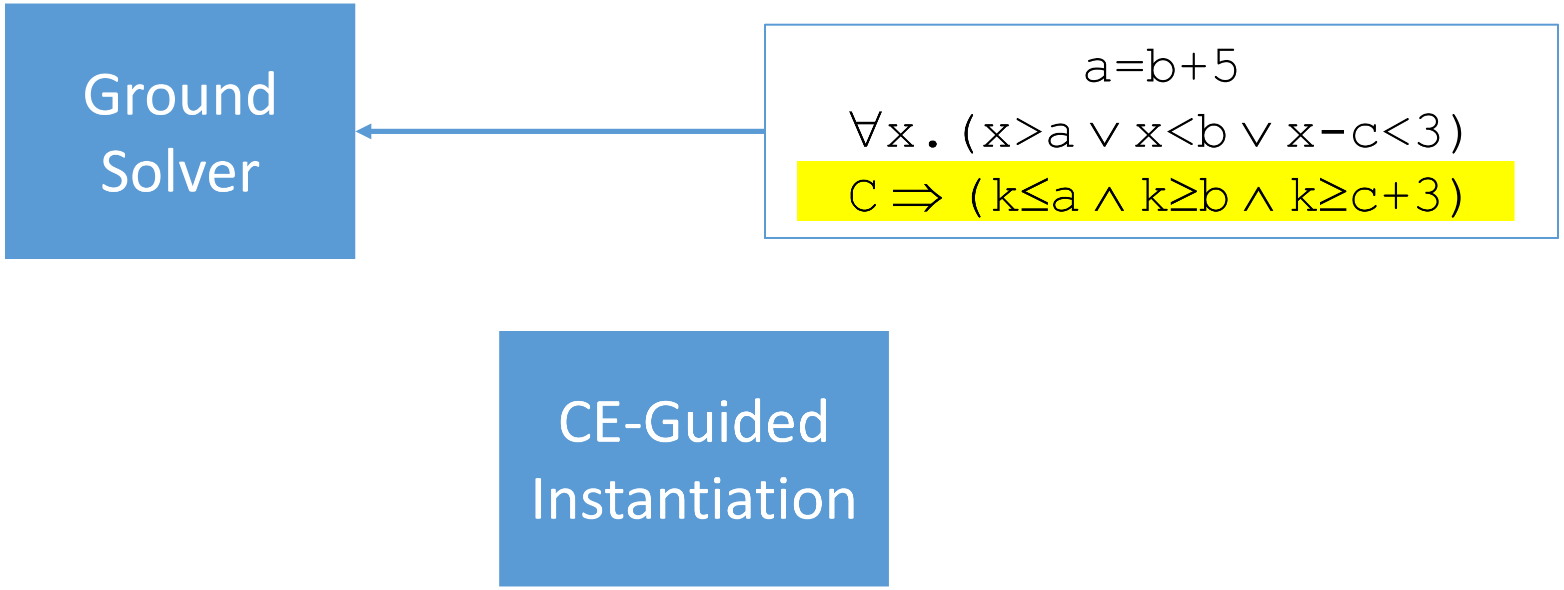




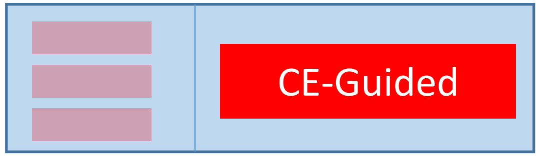
$\Rightarrow$With respect *to model-based instantiation*:

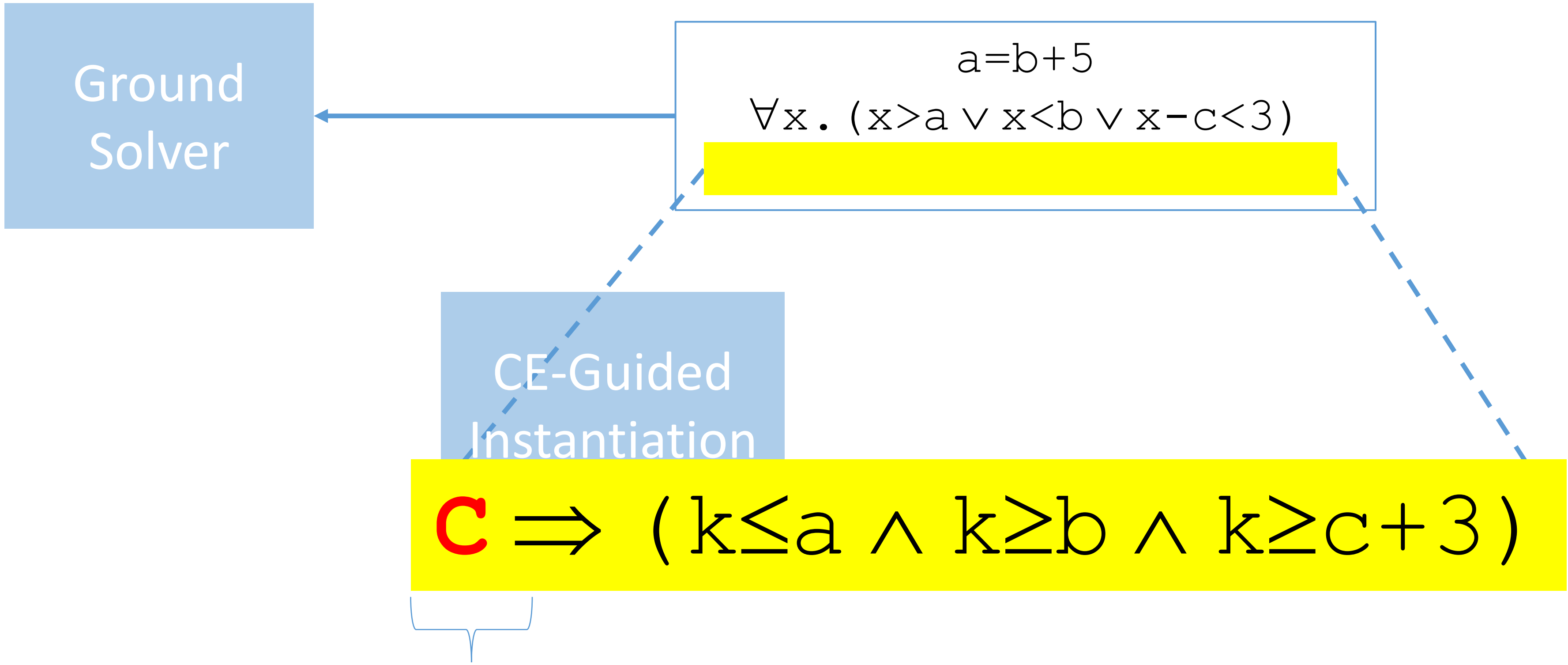
- Similar: check satisfiability of $\exists k.\neg(k>a \vee k<b \vee k-c<3)$
- *Key difference:* use the same (ground) solver for F and *counterexample* k *for* Q

Counterexample-Guided Instantiation



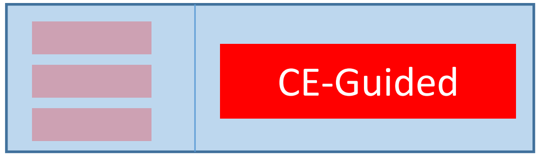
Counterexample-Guided Instantiation

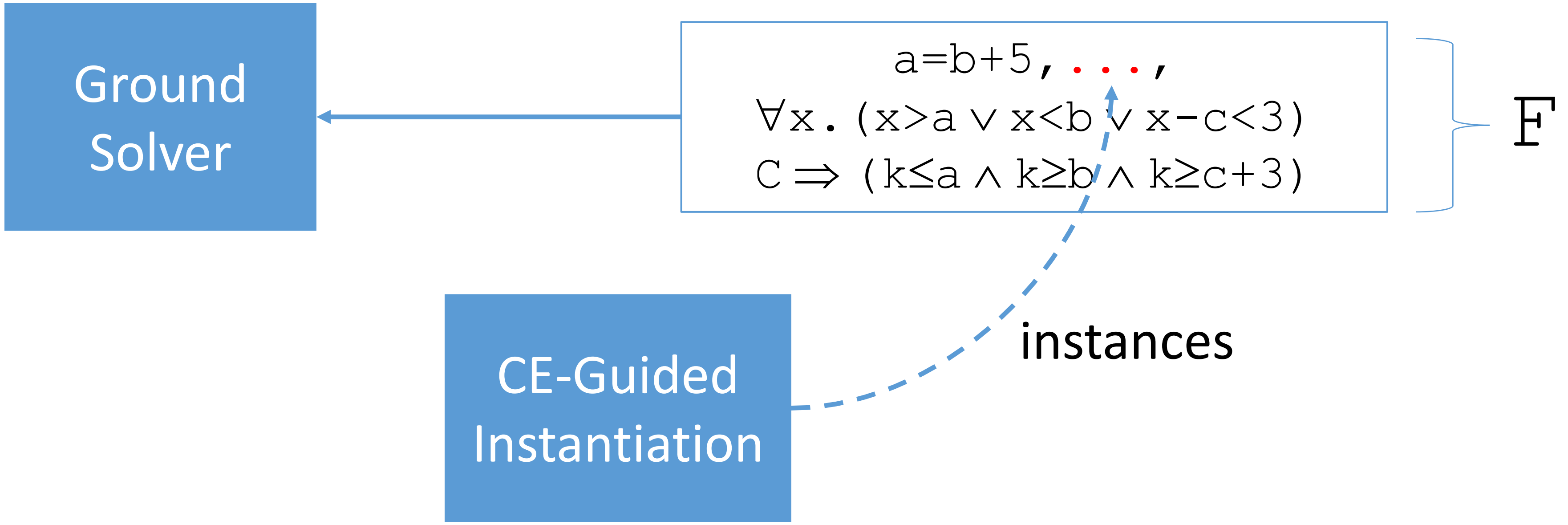




C is a fresh Boolean variable:
"A counterexample k exists for ∀x.(x>a ∨ x<b ∨ x−c<3)"

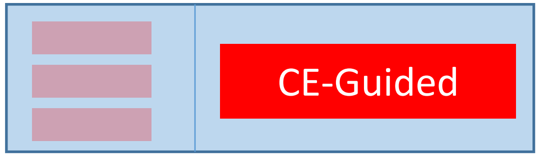
Counterexample-Guided Instantiation

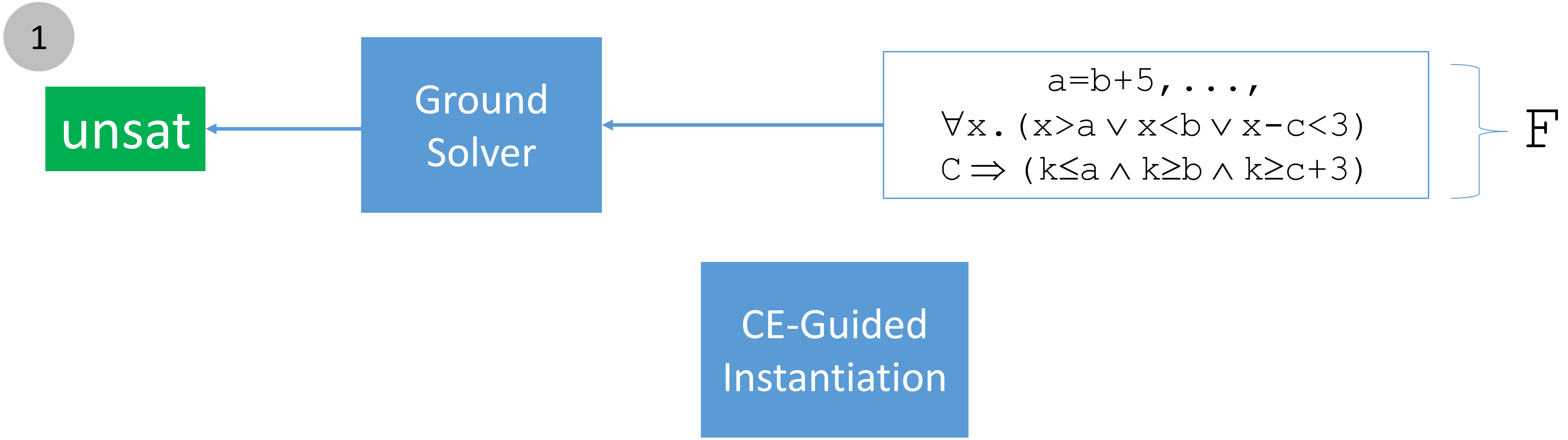




- Three cases:

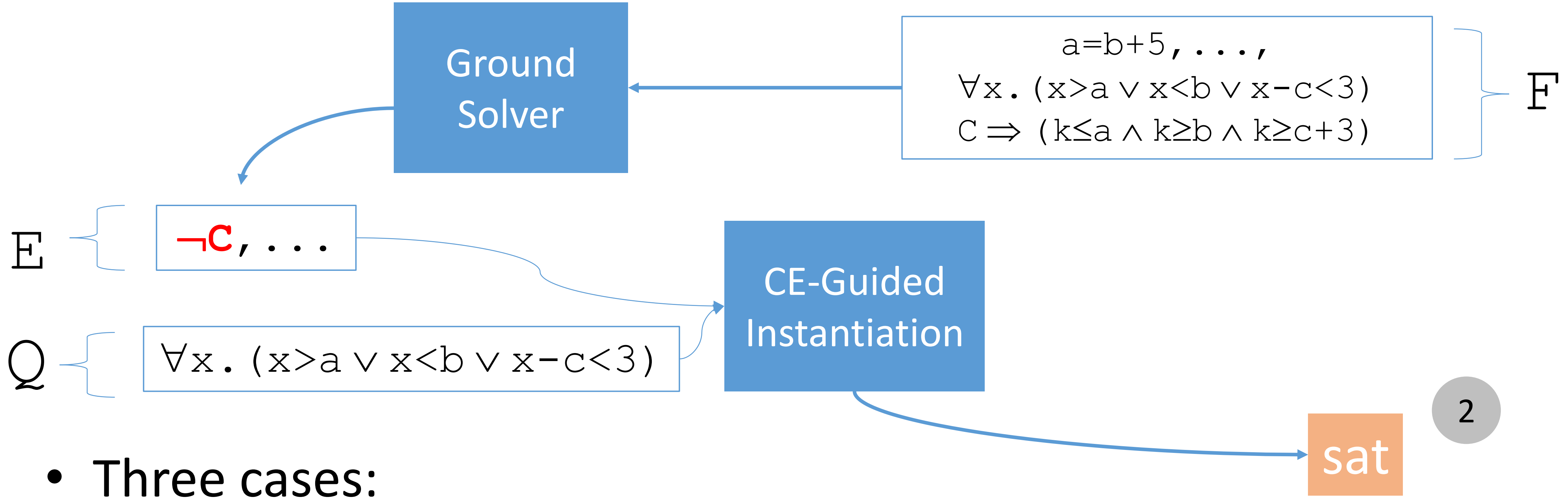
Counterexample-Guided Instantiation





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

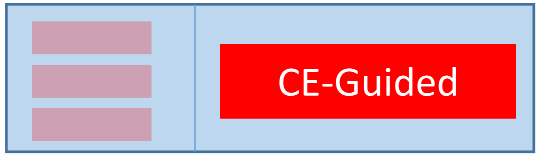
Counterexample-Guided Instantiation

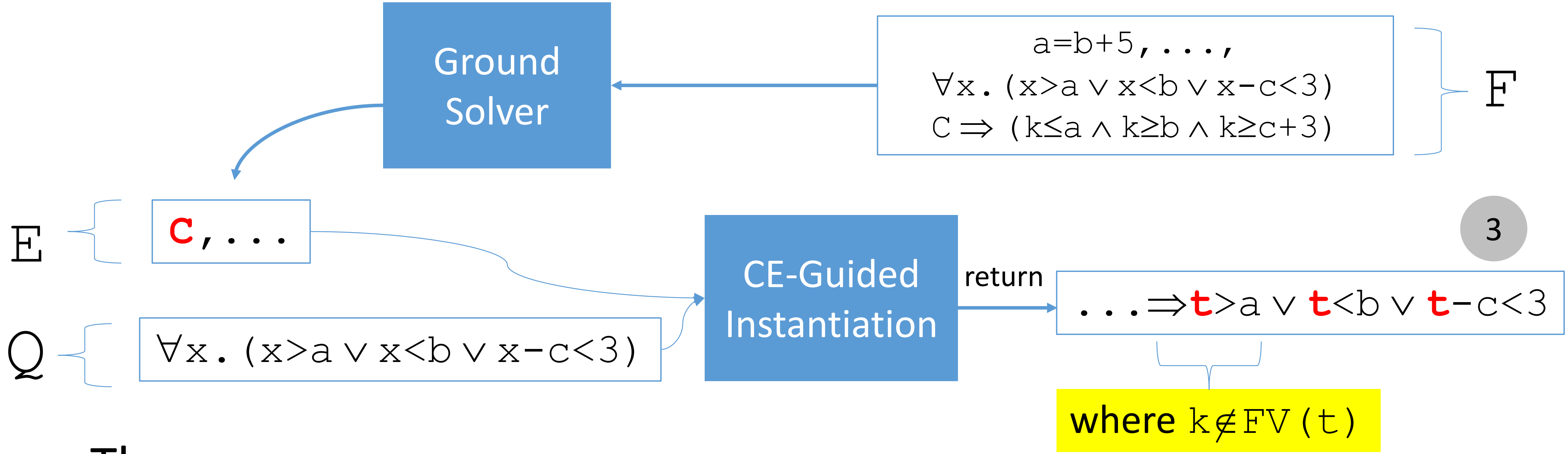


- Three cases:

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

Counterexample-Guided Instantiation

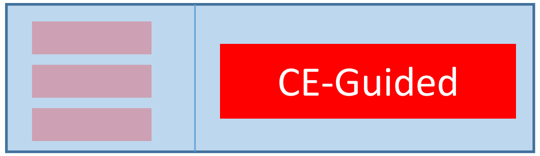


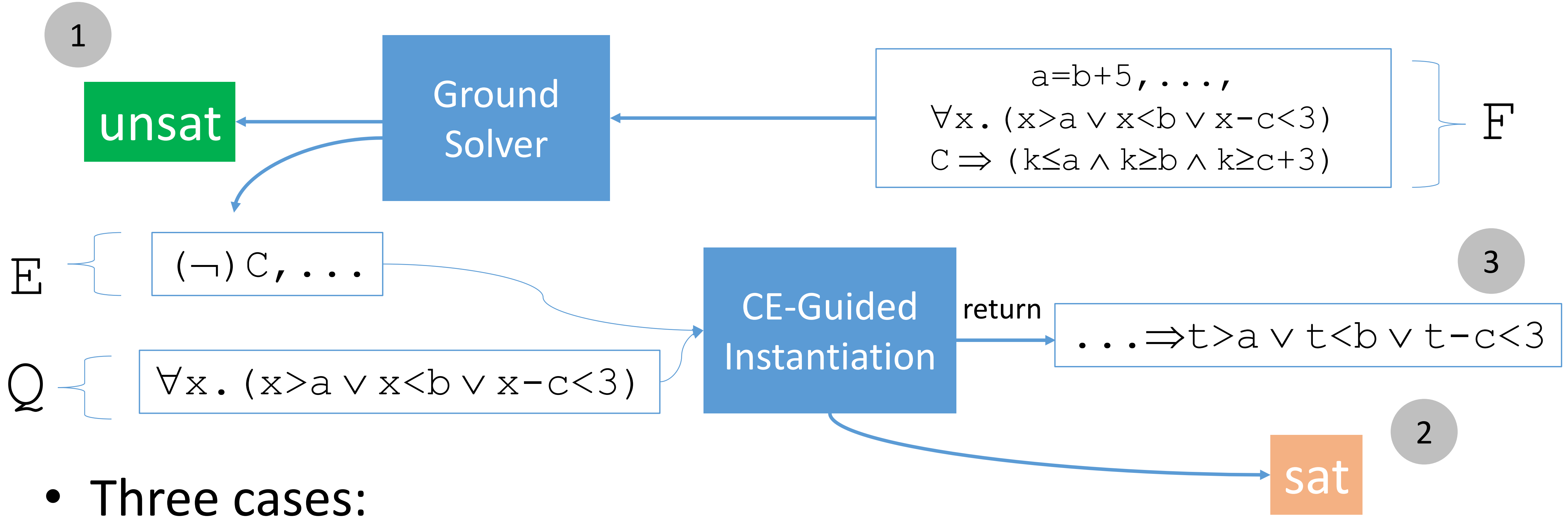


- Three cases:

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

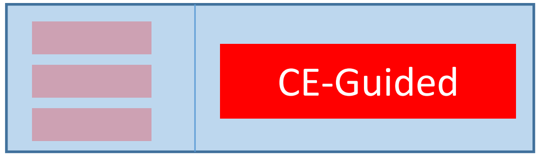
Counterexample-Guided Instantiation

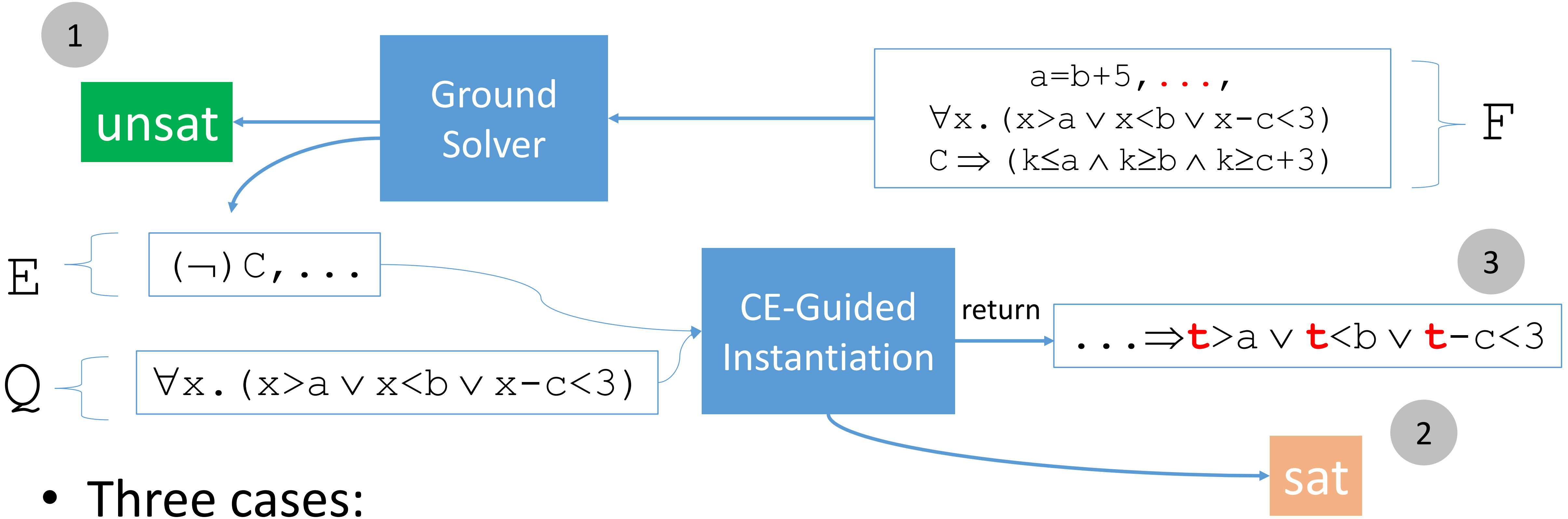




- Three cases:
  1. F is unsatisfiable ⇒ answer "unsat"
  2. F is satisfiable, $\neg C \in E$ for all assignments E ⇒ answer "sat"
  3. F is satisfiable, $C \in E$ for some assignment E ⇒ add an instance to F

Counterexample-Guided Instantiation

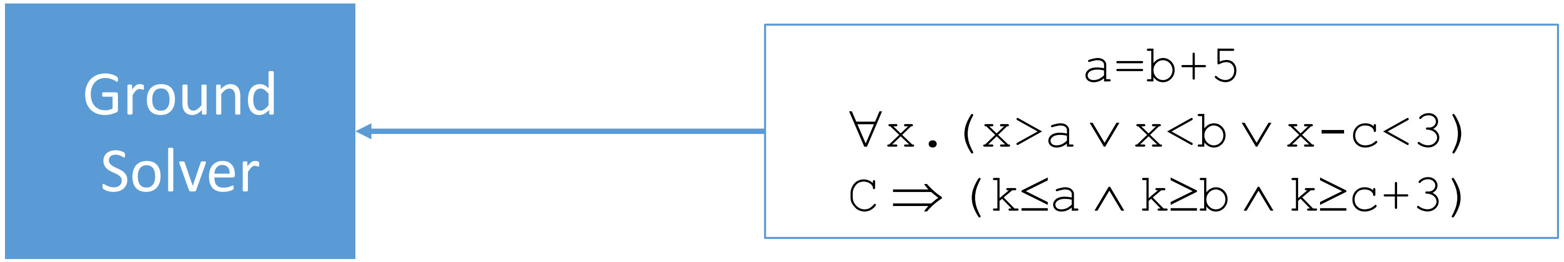




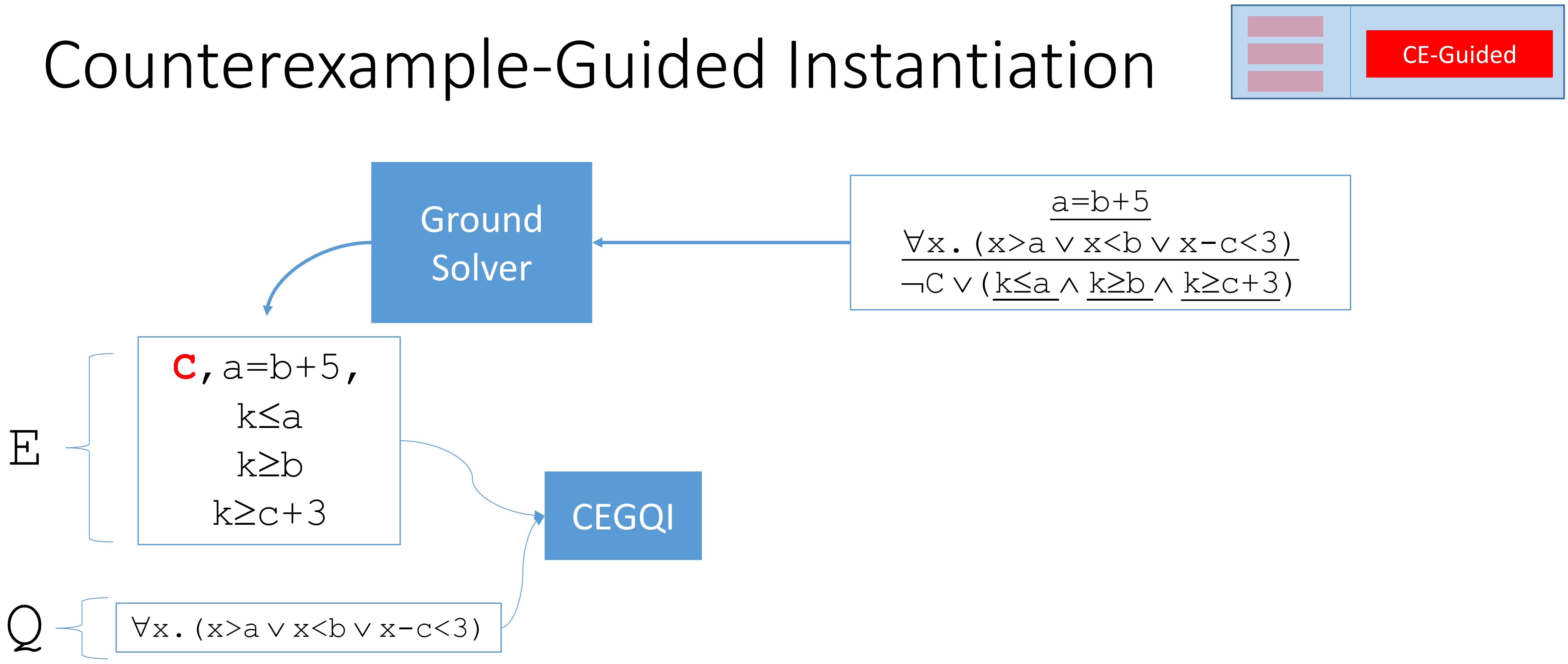
- Three cases:
  1. F is unsatisfiable ⇒ answer "unsat"
  2. F is satisfiable, ¬C∈E for all assignments E ⇒ answer "sat"
  3. F is satisfiable, C∈E for some assignment E ⇒ add **an instance** to F (...which **t**?)

Counterexample-Guided Instantiation

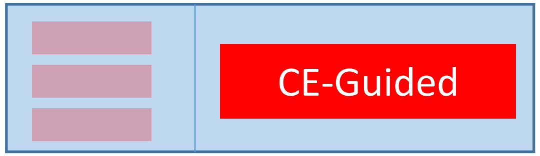
CE-Guided

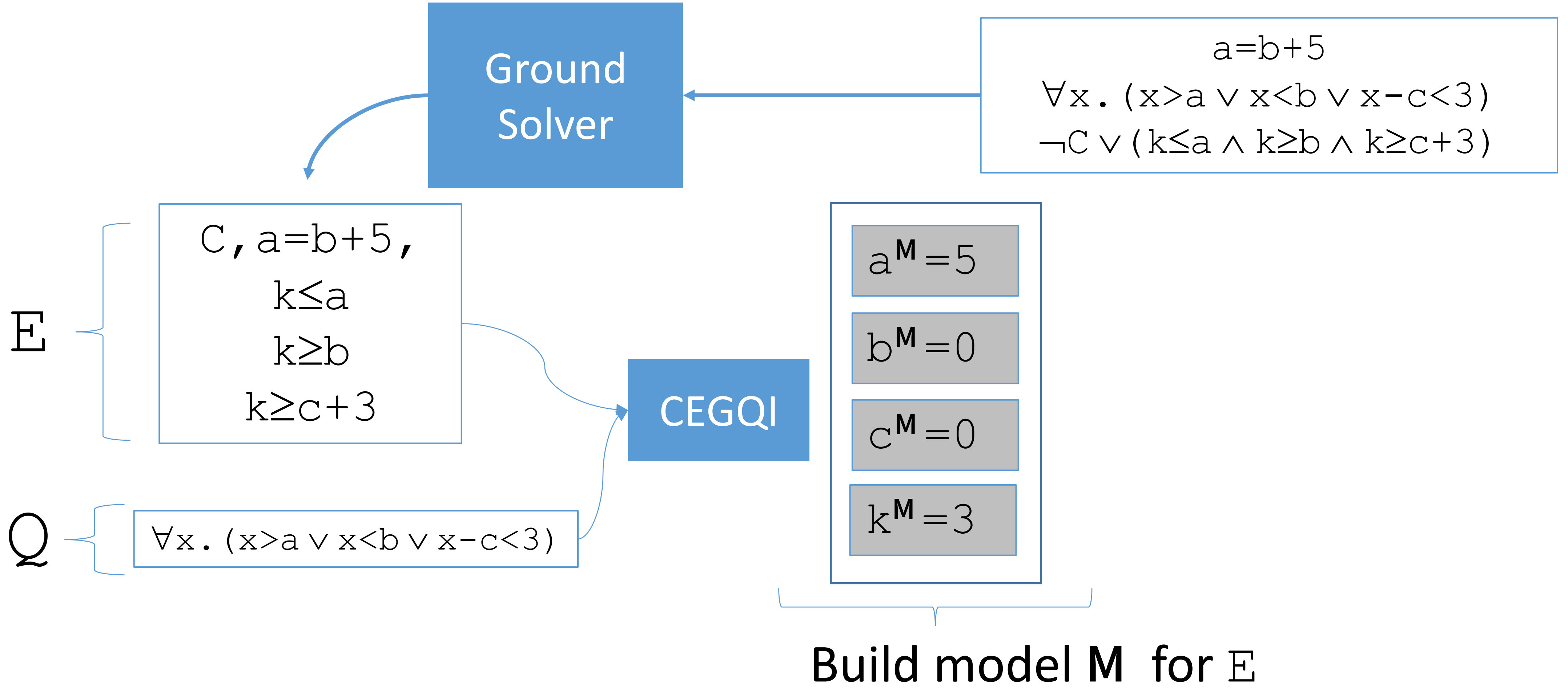


Counterexample-Guided Instantiation

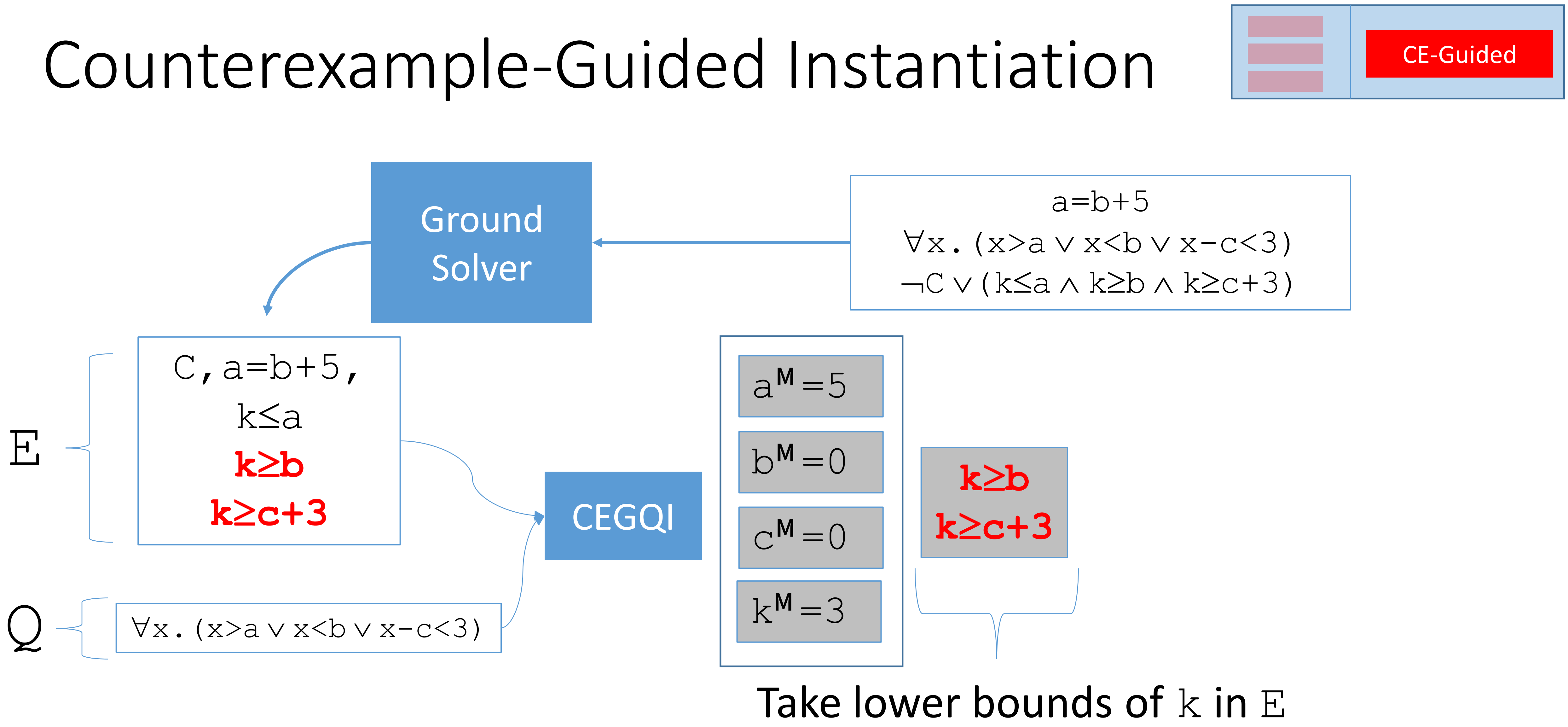


Counterexample-Guided Instantiation

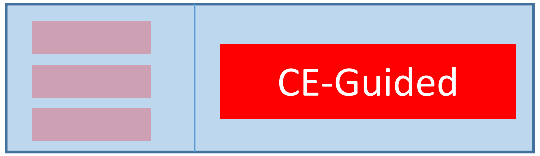


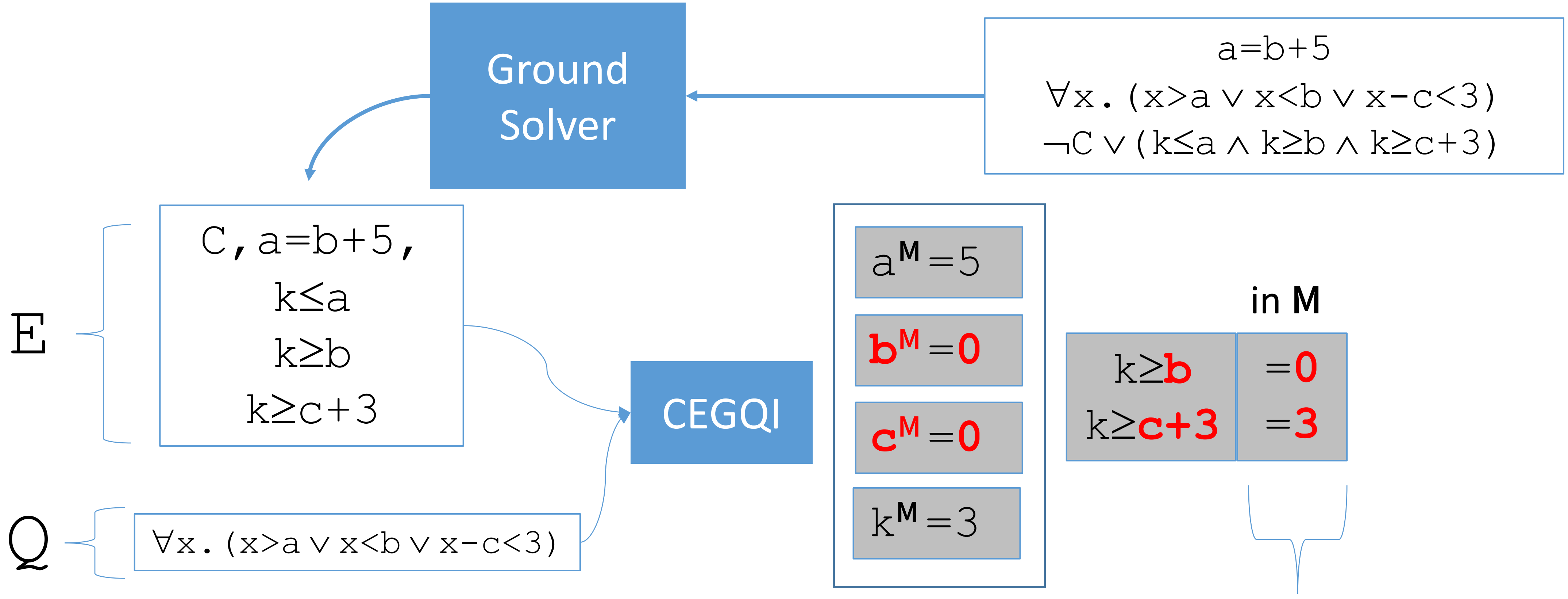


Counterexample-Guided Instantiation



Counterexample-Guided Instantiation

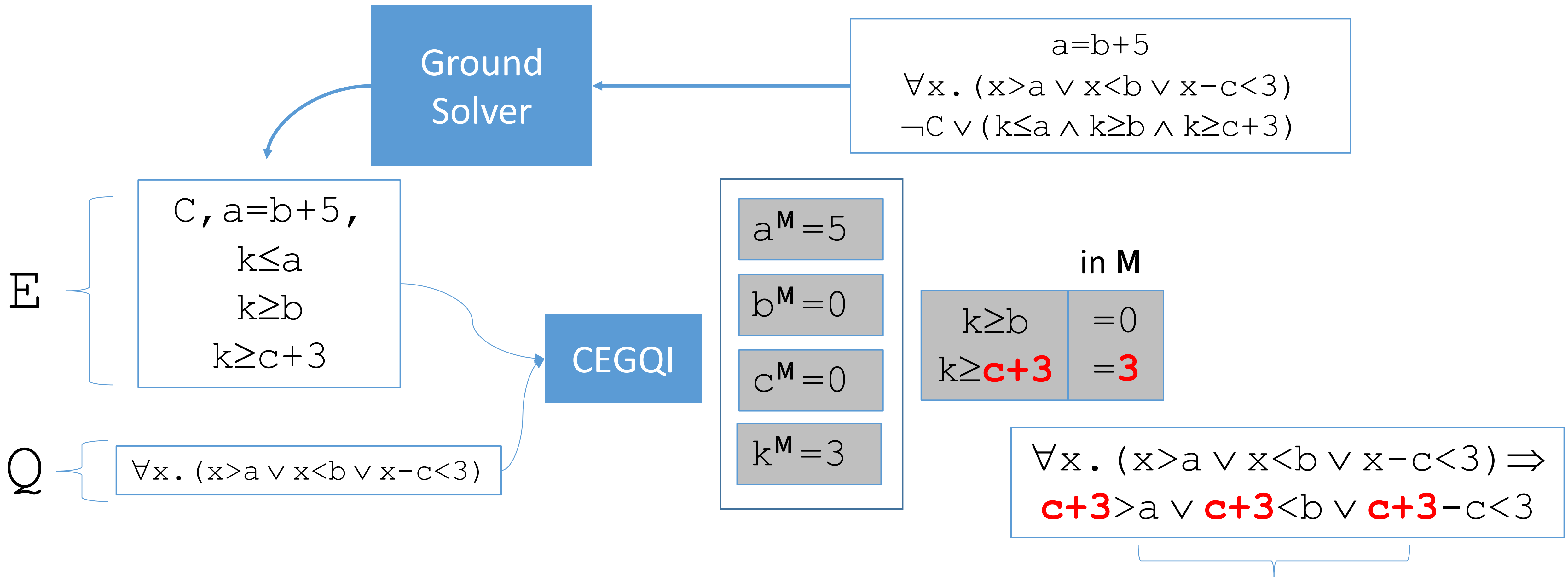




Compute their value in M

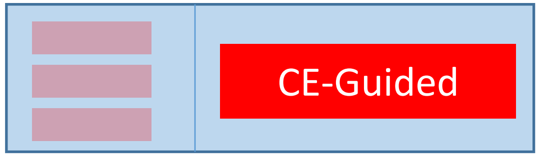
Counterexample-Guided Instantiation

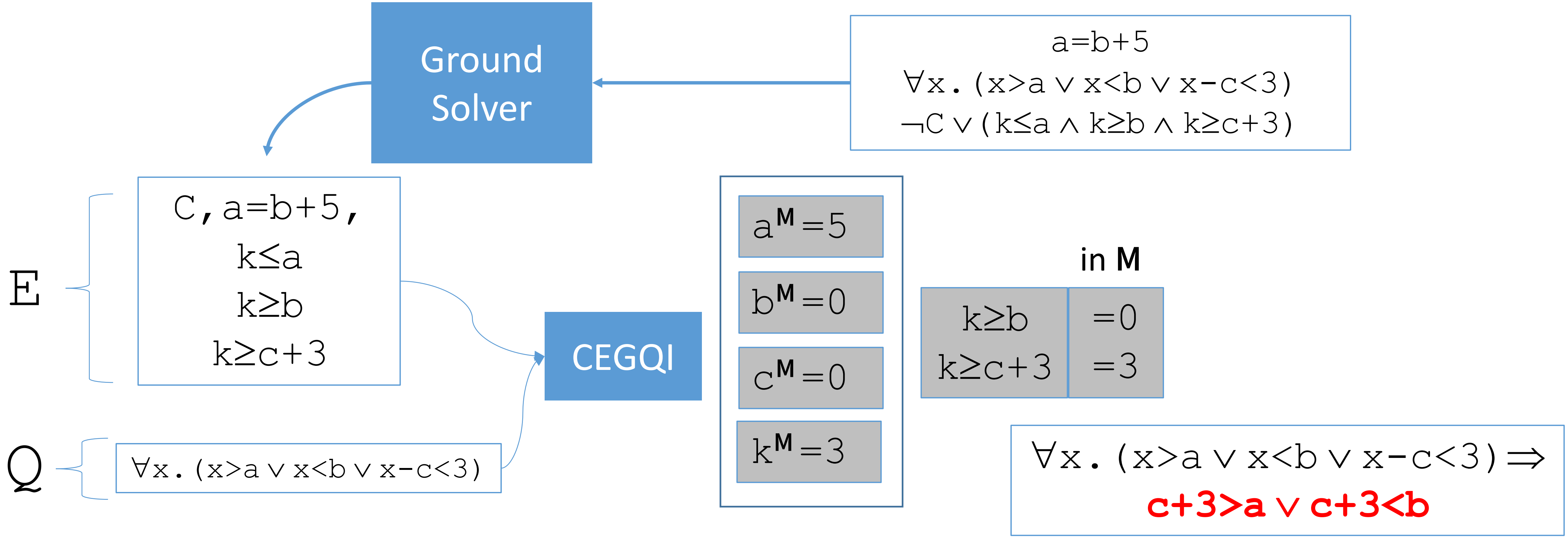
CE-Guided



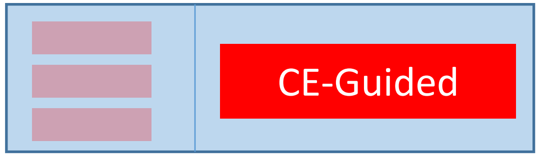
Add instance for lower bound that is maximal in **M**

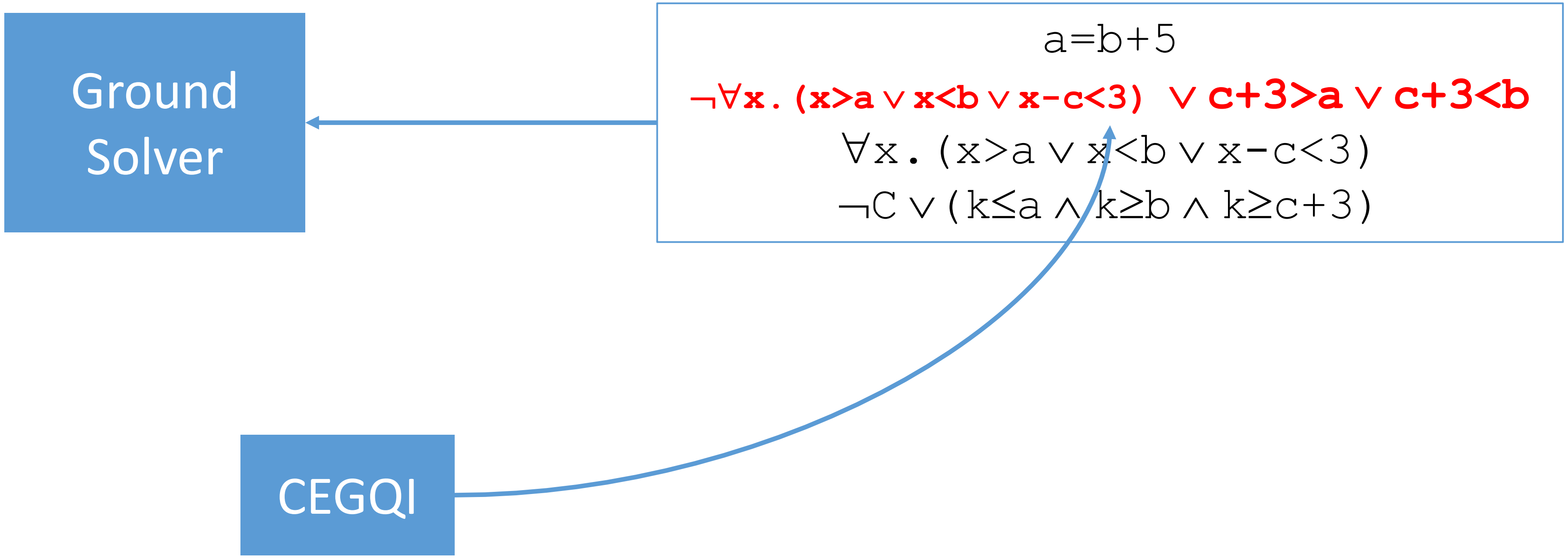
Counterexample-Guided Instantiation



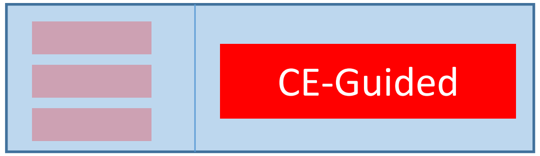


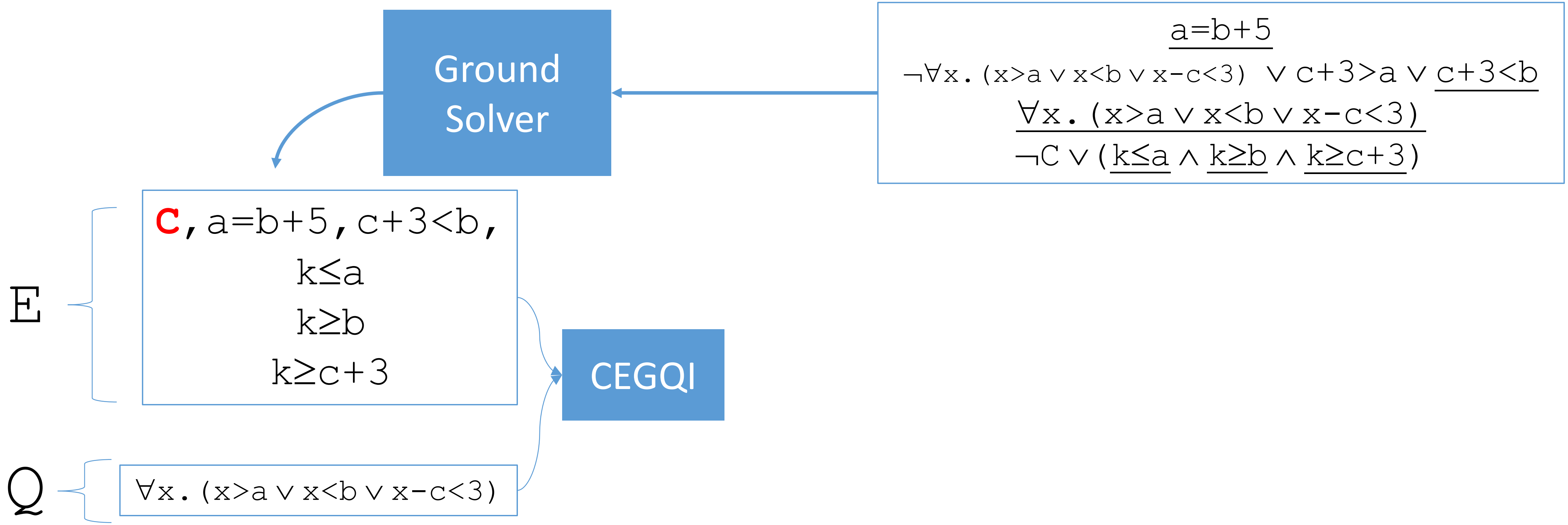
Counterexample-Guided Instantiation



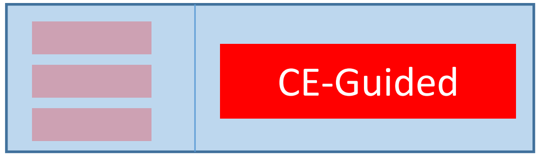


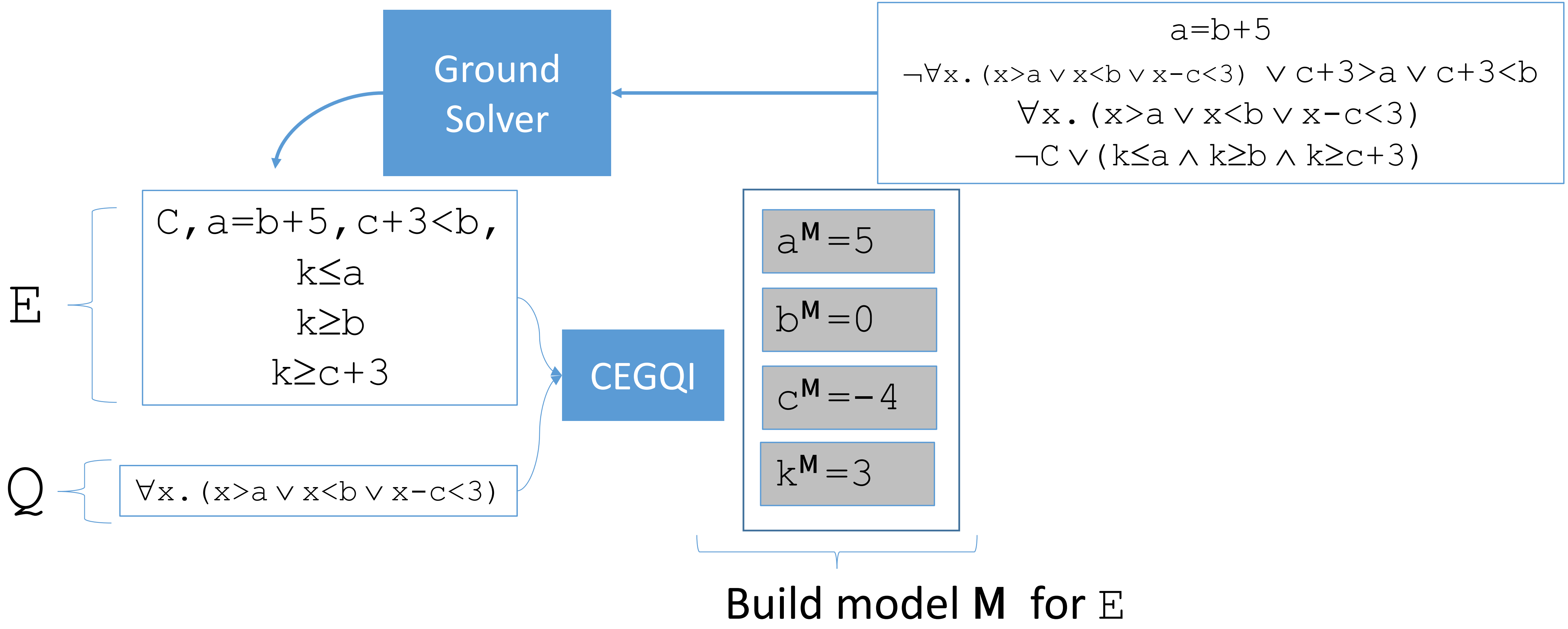
Counterexample-Guided Instantiation





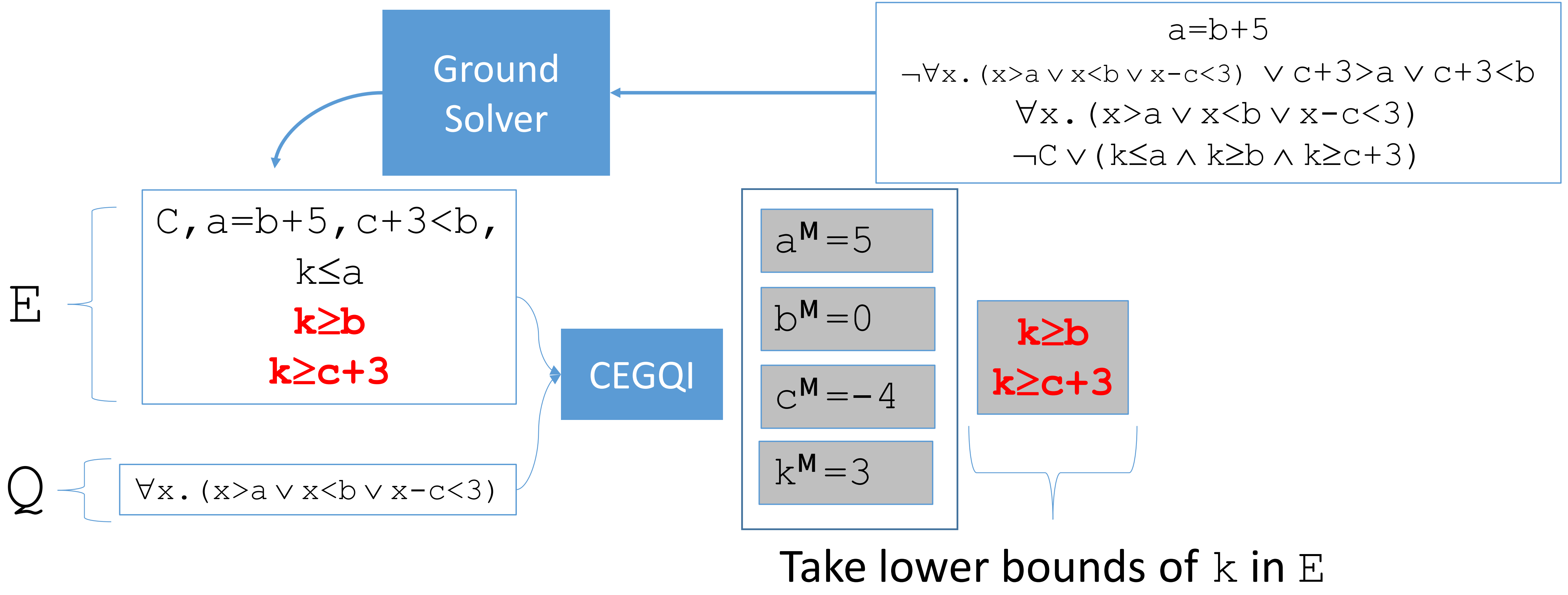
Counterexample-Guided Instantiation



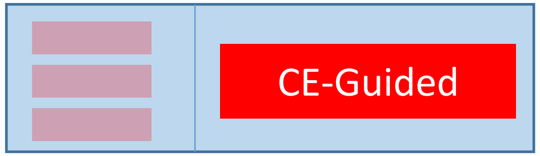


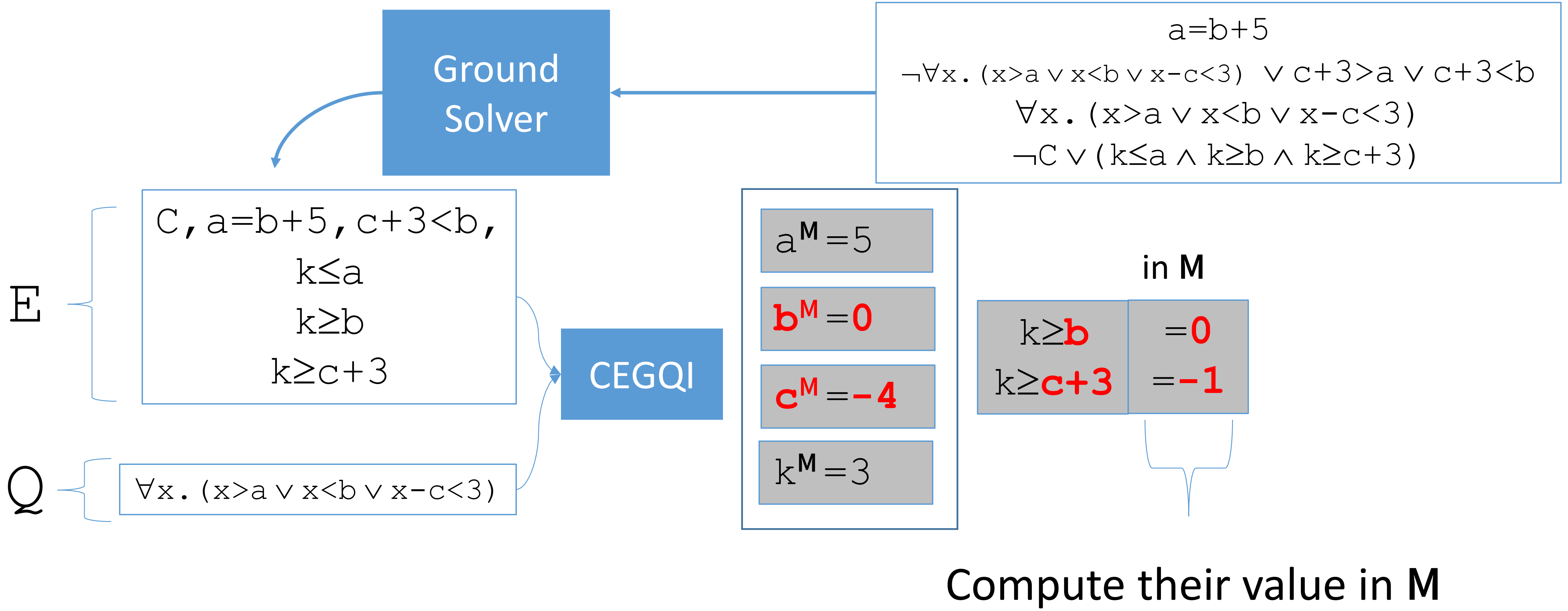
Counterexample-Guided Instantiation

CE-Guided



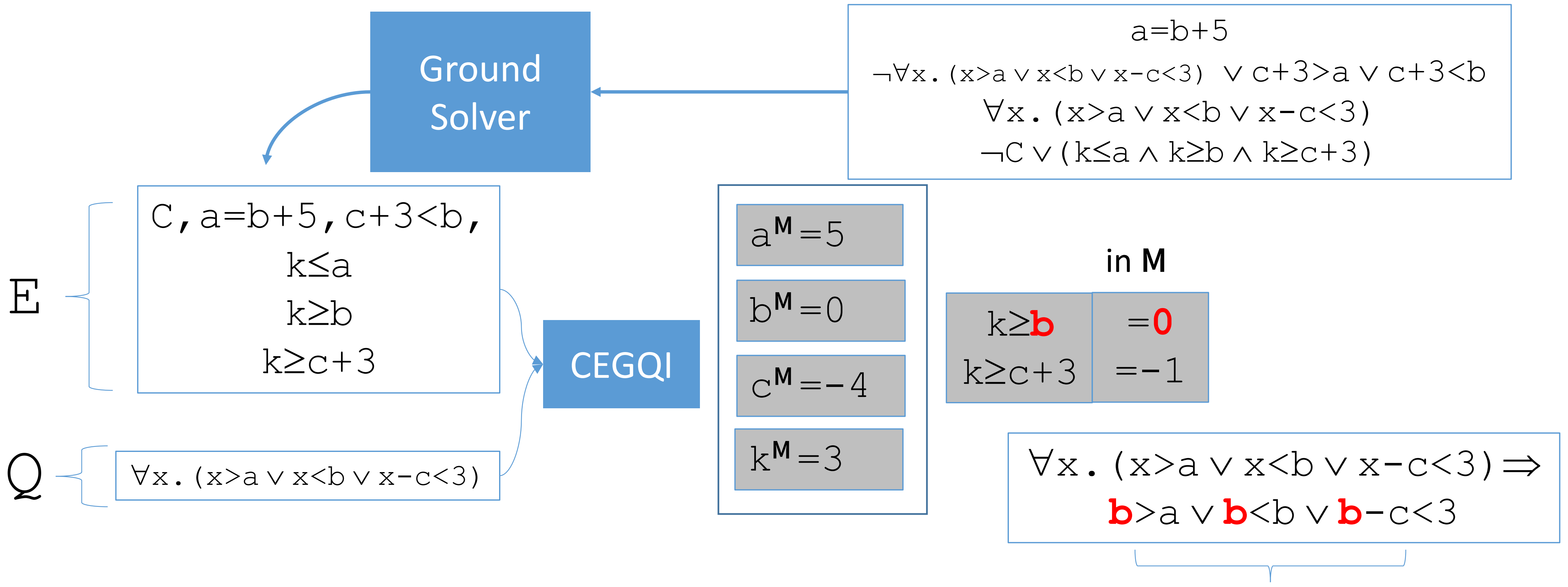
Counterexample-Guided Instantiation





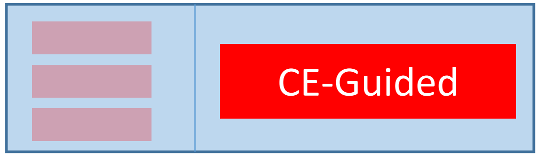
Counterexample-Guided Instantiation

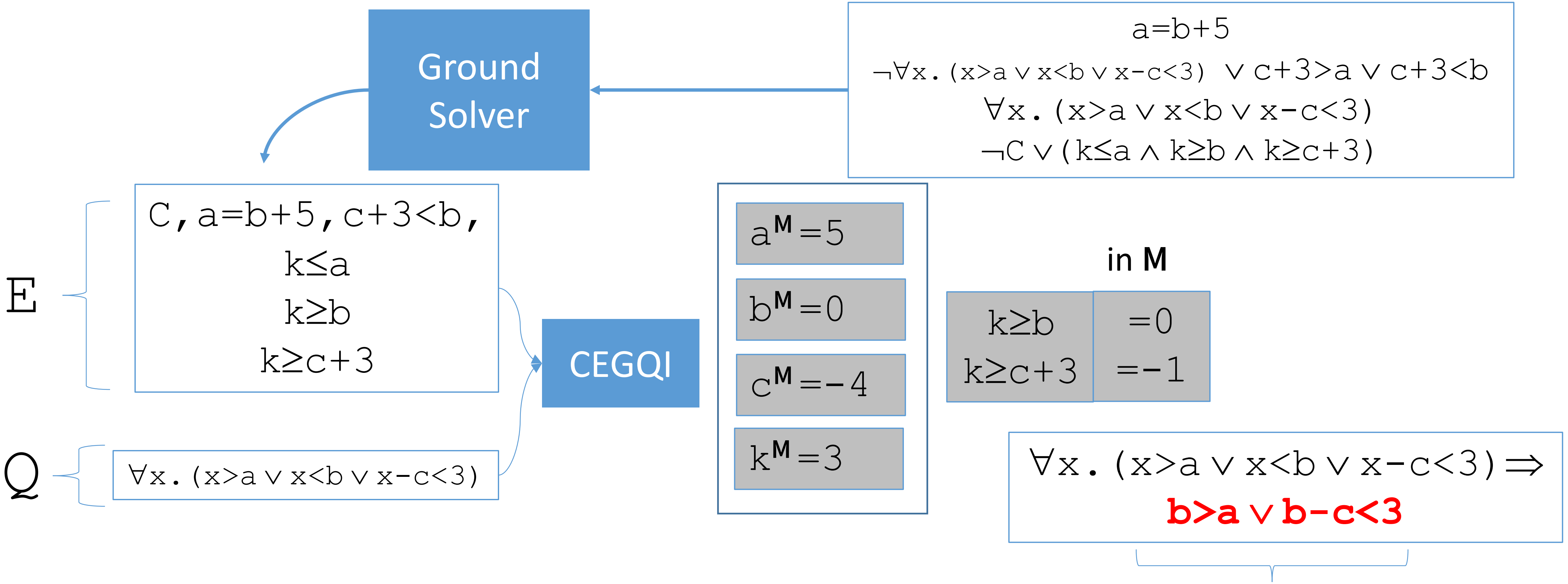
CE-Guided



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

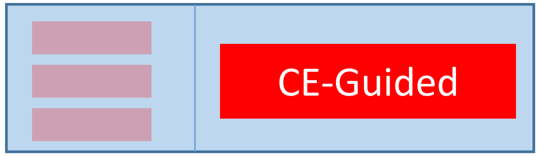
Counterexample-Guided Instantiation

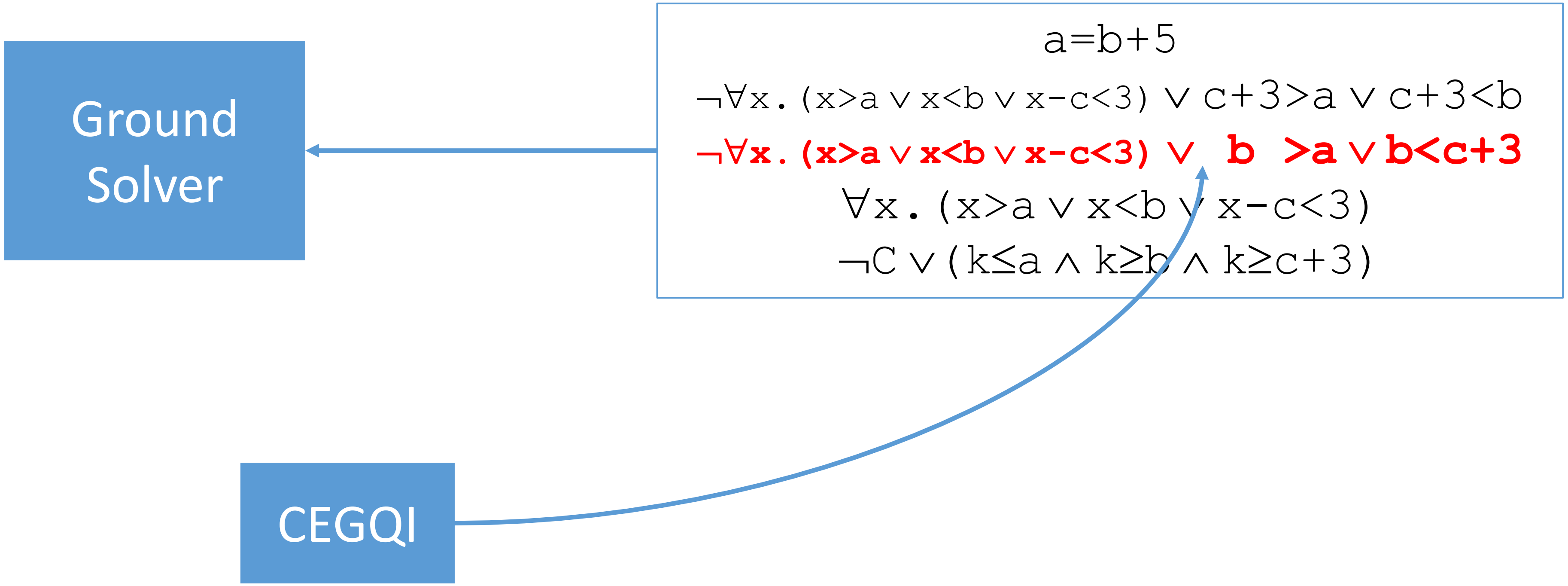




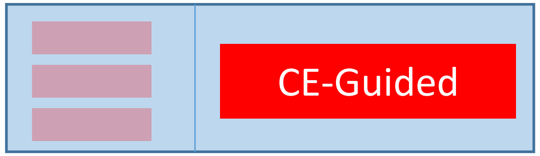
Add instance for lower bound that is maximal in **M**

Counterexample-Guided Instantiation





Counterexample-Guided Instantiation



Ground Solver

$a=b+5$

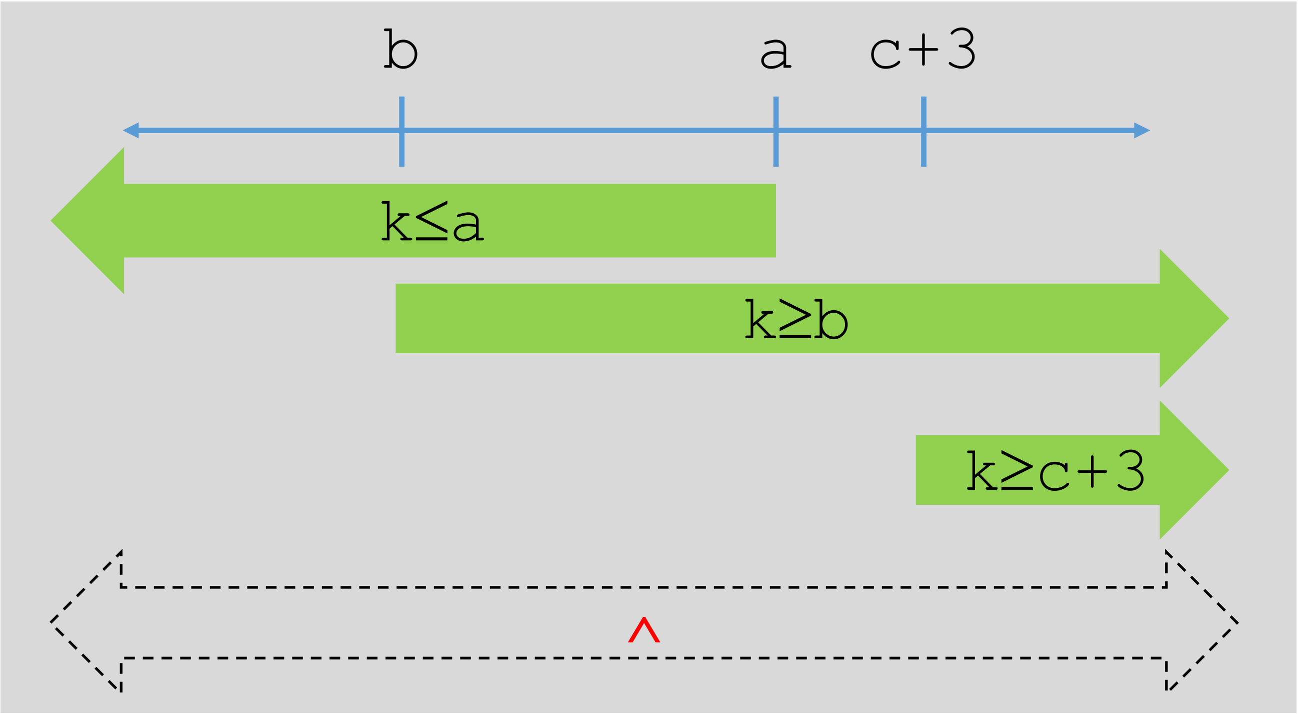
$\neg\forall x.(x>a \vee x<b \vee x-c<3) \vee c+3>a \vee c+3<b$

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

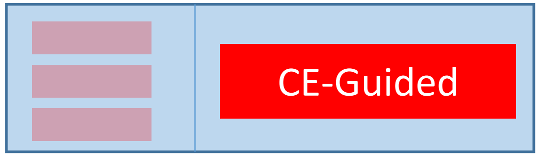
$\forall x.(x>a \vee x<b \vee x-c<3)$

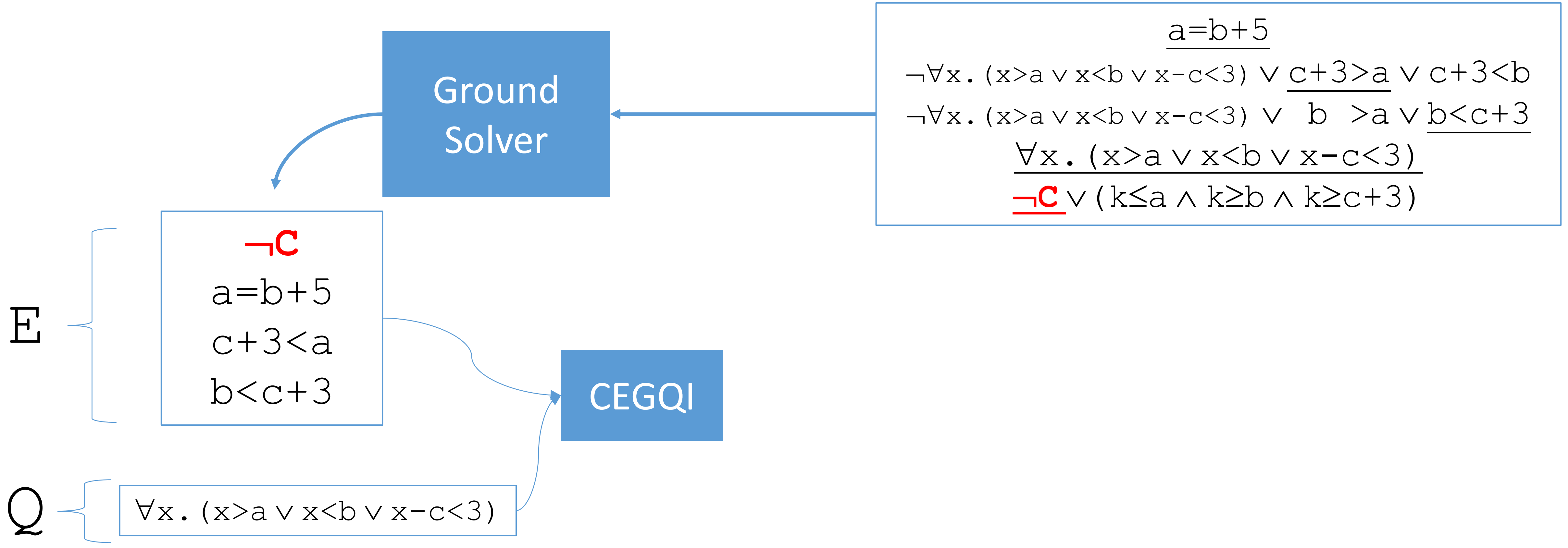
$\neg C \vee (\mathbf{k\leq a} \wedge \mathbf{k\geq b} \wedge \mathbf{k\geq c+3})$

CEGQI



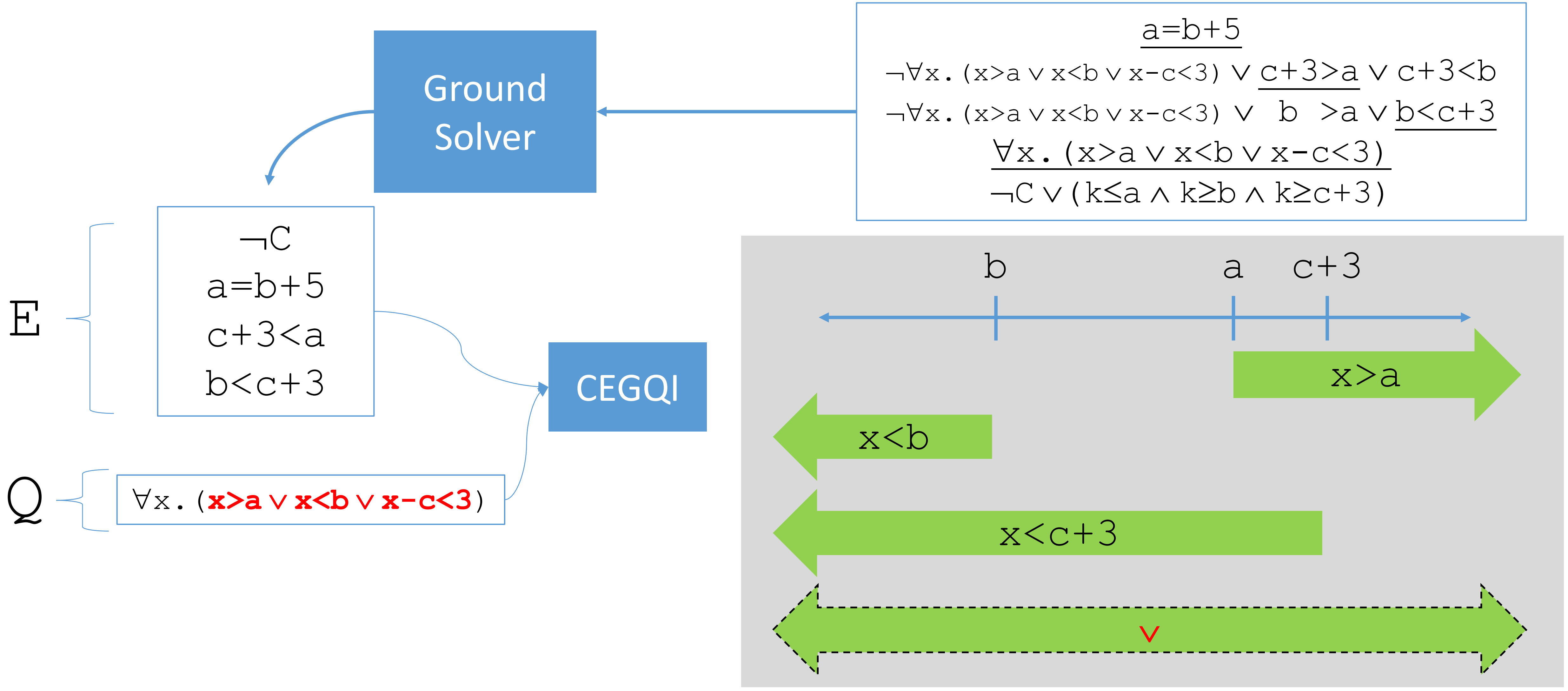
Counterexample-Guided Instantiation



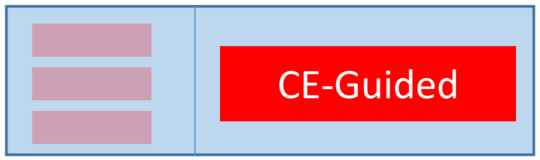


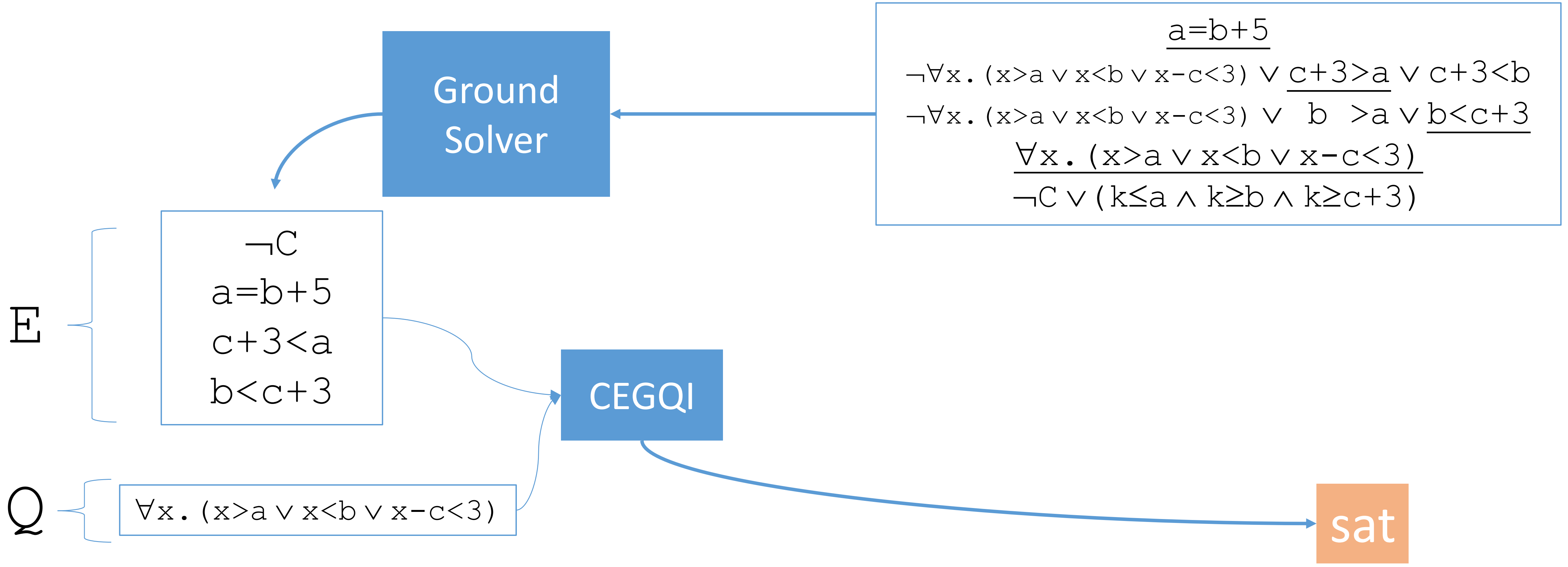
Counterexample-Guided Instantiation

CE-Guided



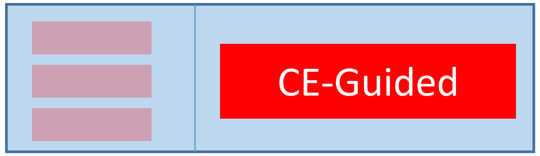
Counterexample-Guided Instantiation





$\Rightarrow \exists abc.(a=b+5 \land \forall x.(x>a \lor x<b \lor x-c<3))$
is LIA-satisfiable

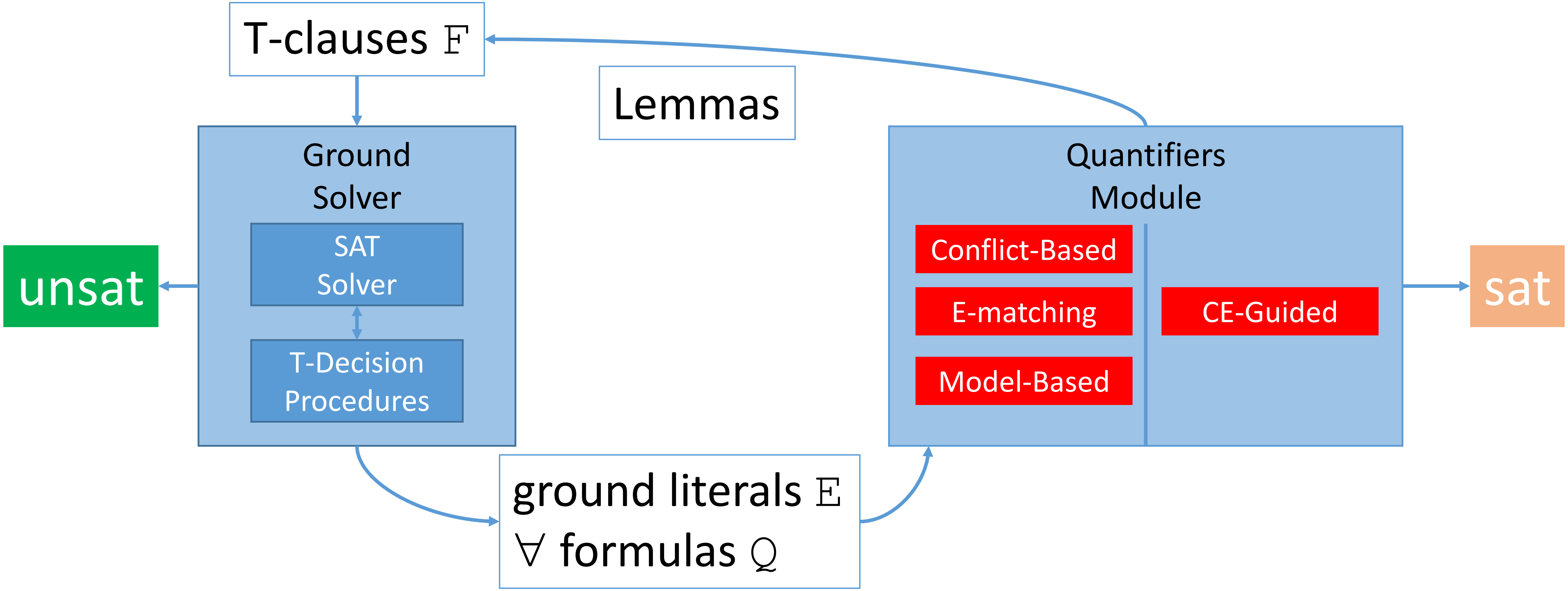
Counterexample-Guided Instantiation



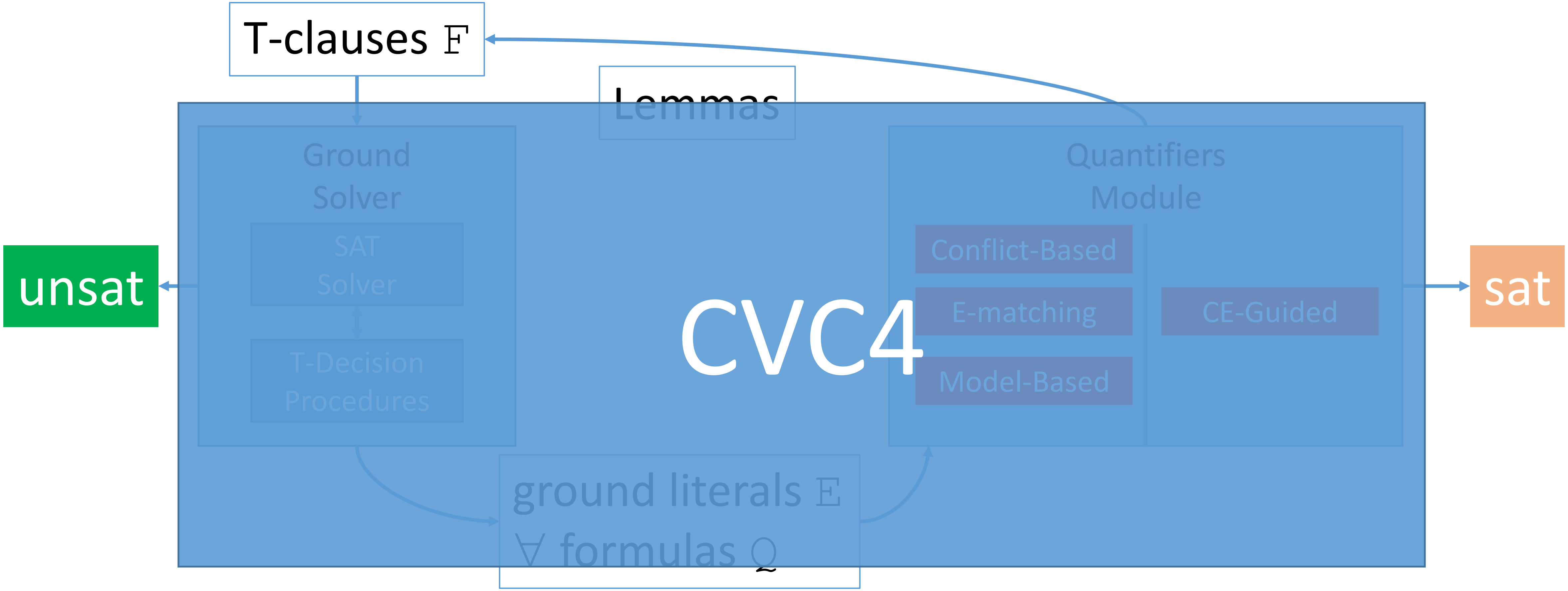
- Decision procedure for $\forall$ in various theories:
  - Linear real arithmetic (LRA)
    - Maximal lower (minimal upper) bounds — $l_1<k, \ldots, l_n<k \rightarrow \{x \rightarrow l_{max}+\delta\}$
      - [Loos+Wiespfenning 93]
      - *…may involve virtual terms* δ, ∞
    - Interior point method: — $l_{max}<k<u_{min} \rightarrow \{x \rightarrow (l_{max}-u_{min})/2\}$
      - [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, …

$\Rightarrow$ ***Termination argument for each****: enumerate at most a finite number of instances*

Summary: DPLL(T)+Instantiation



Summary: DPLL(T)+Instantiation



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

- CVC4:
  - Open source, available at http://cvc4.cs.nyu.edu/downloads/