Finding Conflicting Instances of Quantified Formulas in SMT

Andrew Reynolds
Cesare Tinelli
Leonardo De Moura
July 18, 2014

Outline of Talk

- SMT solvers:
  - Efficient methods for ground constraints
  - Heuristic methods for quantified formulas

  $\Rightarrow$ *Can we reduce dependency on heuristic methods?*
- New method for quantifiers in SMT
  - Finds conflicting instances of quantified formulas
- Experimental results
- Summary and Future Work

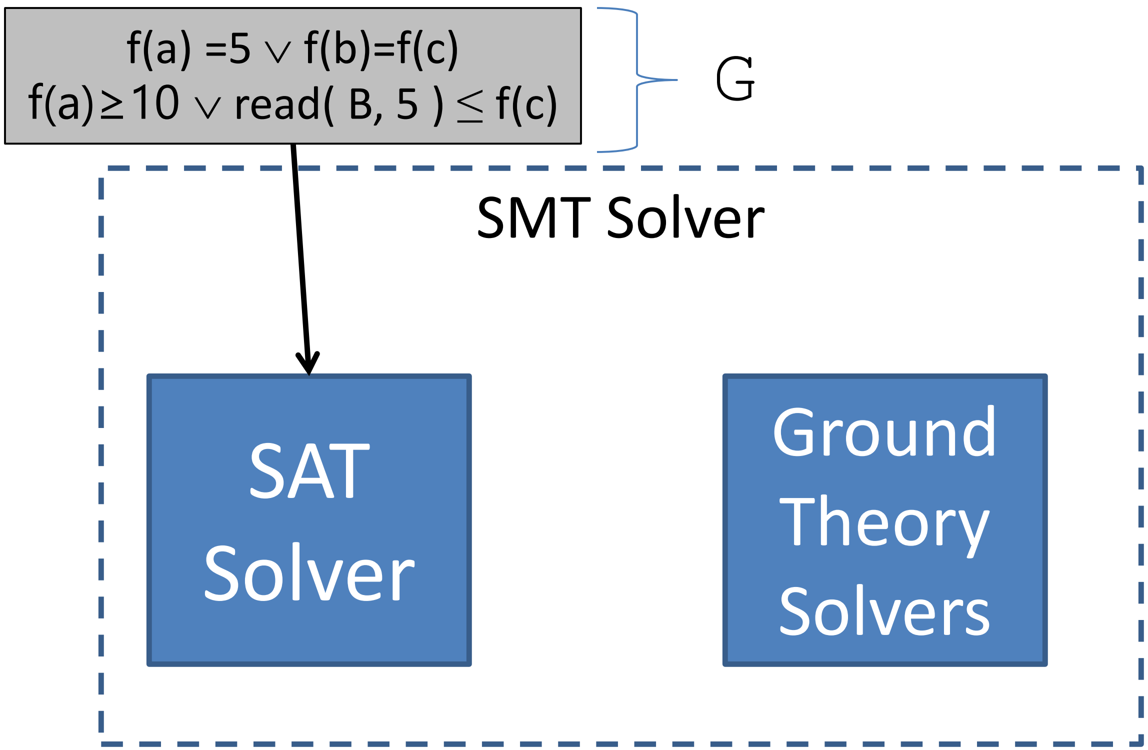
Satisfiability Modulo Theories (SMT)

- SMT solvers
  - Are efficient for problems over ground constraints $\mathbb{G}$
  - Determine the satisfiability of $\mathbb{G}$ using a combination of:
    - Off-the-shelf SAT solver
    - Efficient ground decision procedures, e.g.
      - Uninterpreted Functions
      - Linear arithmetic
      - Arrays
      - Datatypes
      - …

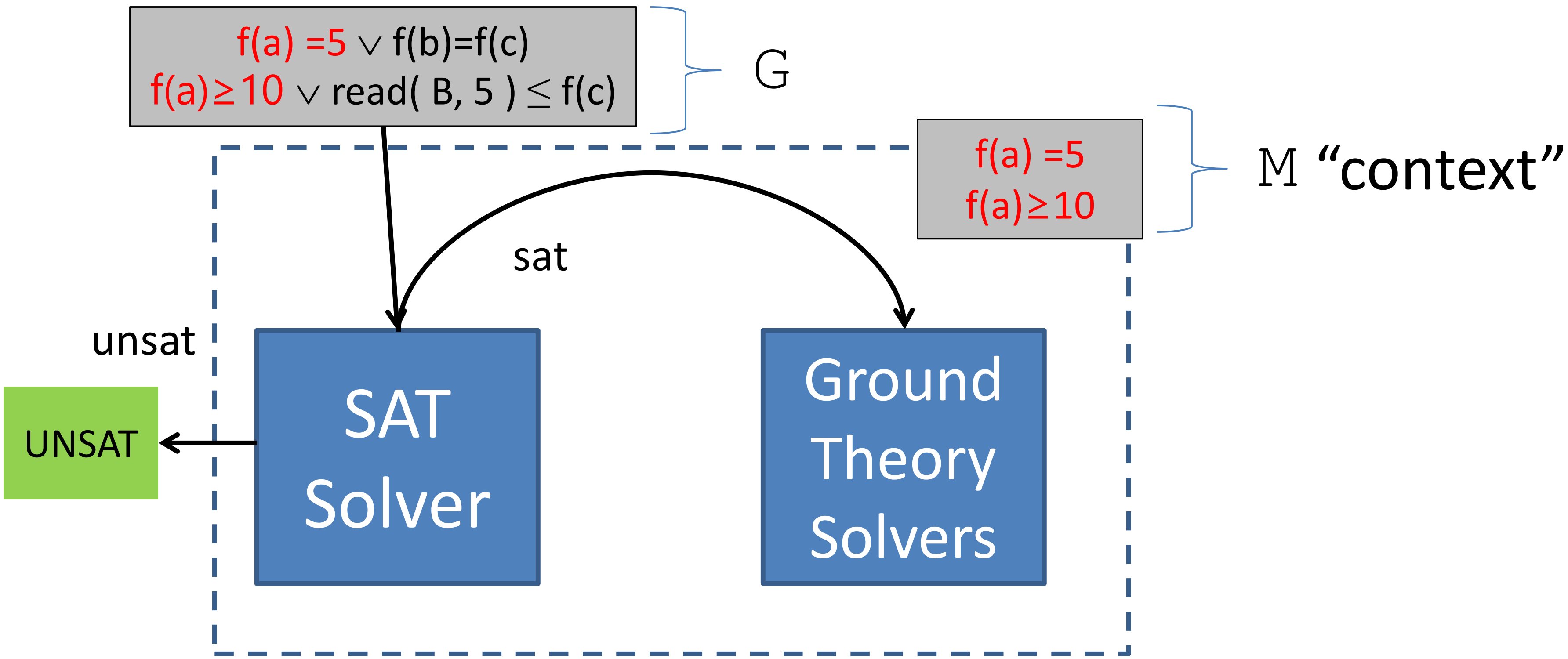
$$
\mathbb{G} = \left\{ \begin{array}{c} f(3) \neq f(c) \\ c=2 \vee c+1 \leq 0 \\ a+1 = read(A,b) \\ tail(l_1)=cons(a,l_2) \end{array} \right.
$$

- Used in many applications:
  - Software/hardware verification
  - Scheduling and Planning
  - Automated Theorem Proving

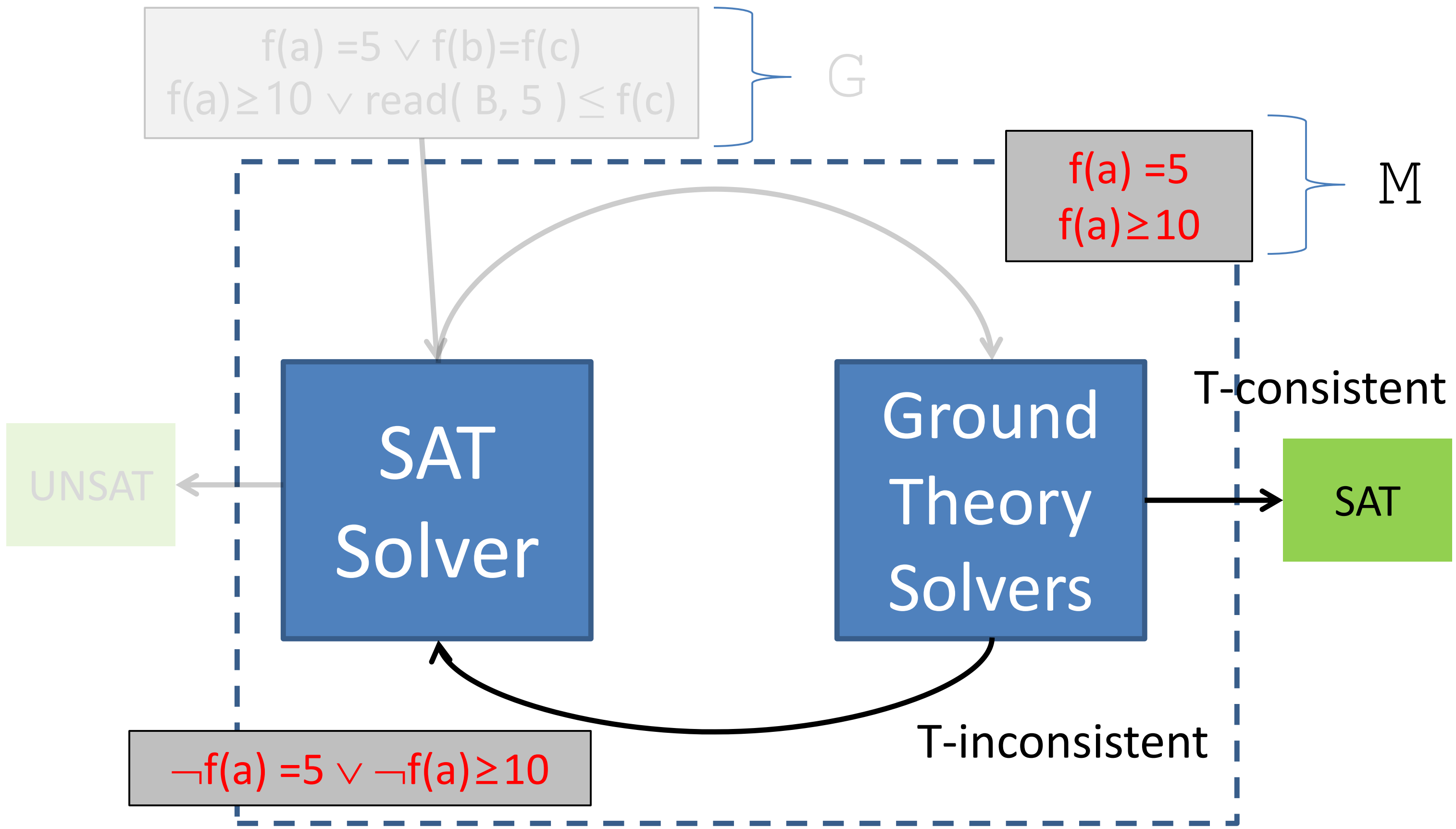
DPLL(T)-Based SMT Solver



DPLL(T)-Based SMT Solver



DPLL(T)-Based SMT Solver



SMT + Quantified Formulas

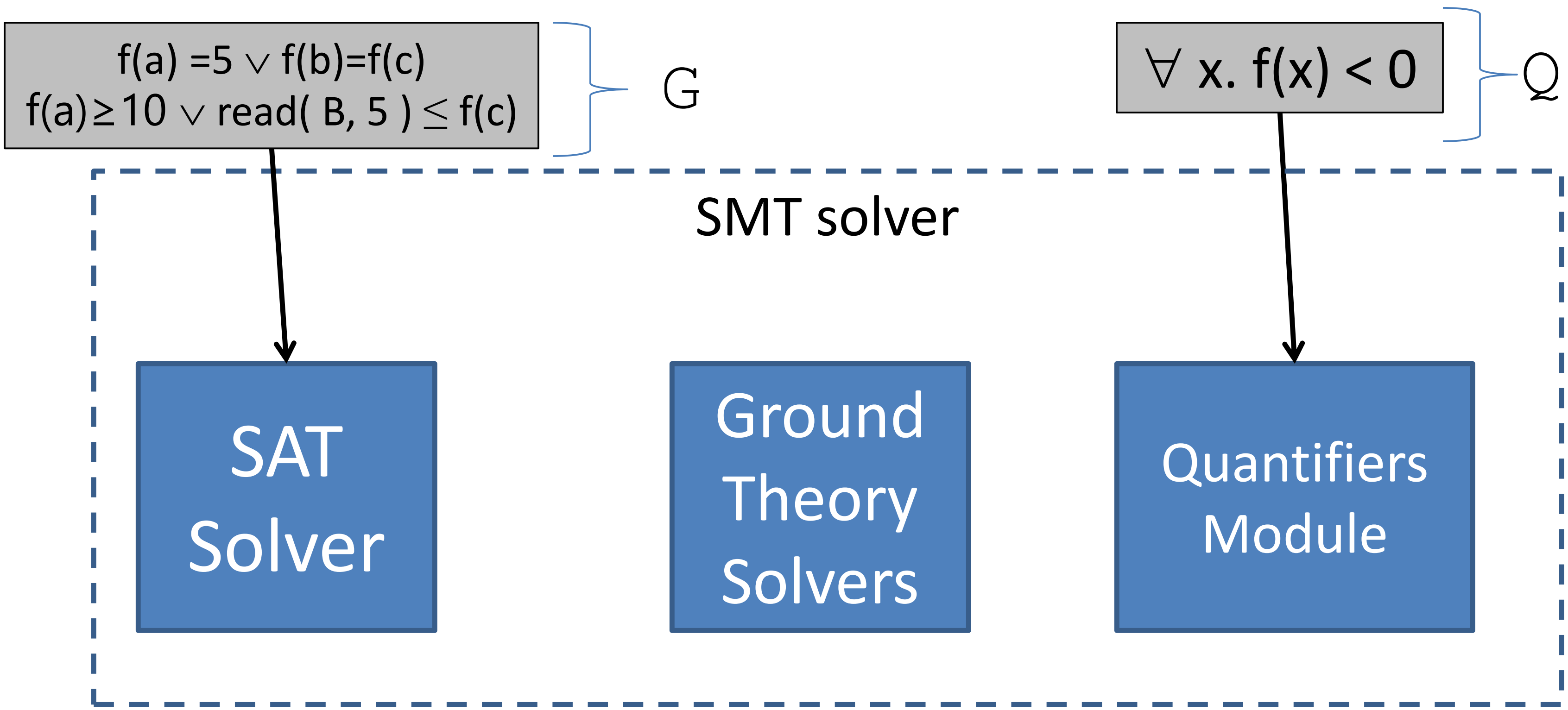
- SMT solvers have limited support for:
  - First-order universally quantified formulas $\mathbb{Q}$

$$\left.\begin{array}{c} f(a) = 5 \vee f(b) = f(c) \\ f(a) \geq 10 \vee \mathrm{read}(B, 5) \leq f(c) \end{array}\right\} \mathbb{G}$$

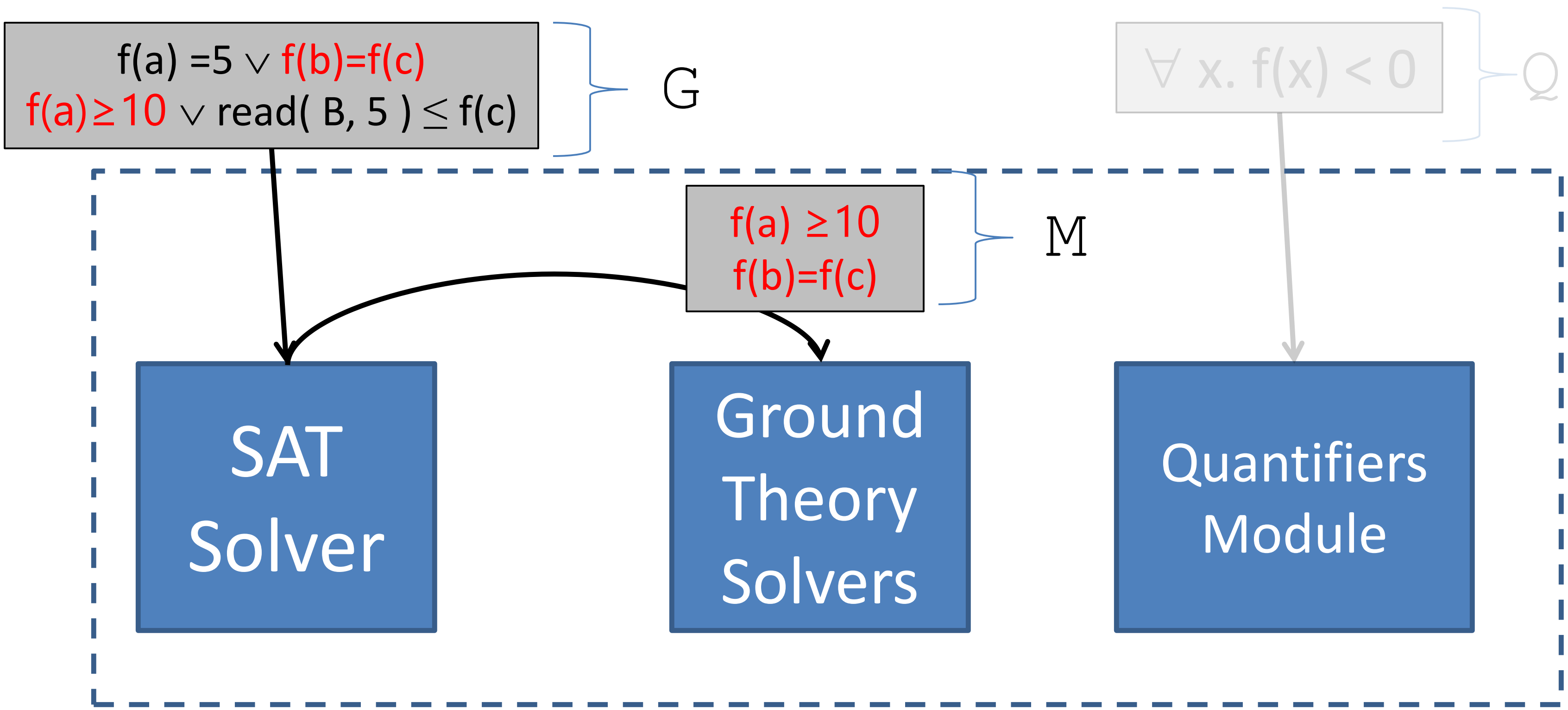
$$\left.\forall x.\ f(x) < 0\right\} \mathbb{Q}$$

- Used in an increasing number of applications, for:
  - Defining axioms for symbols not supported natively
  - Encoding frame axioms, transition systems, …
  - Universally quantified conjectures
- When universally quantified formulas $\mathbb{Q}$ are present, problem is generally undecidable
  - Approaches for $G \cup Q$ in SMT are usually heuristic

SMT Solver + Quantified Formulas

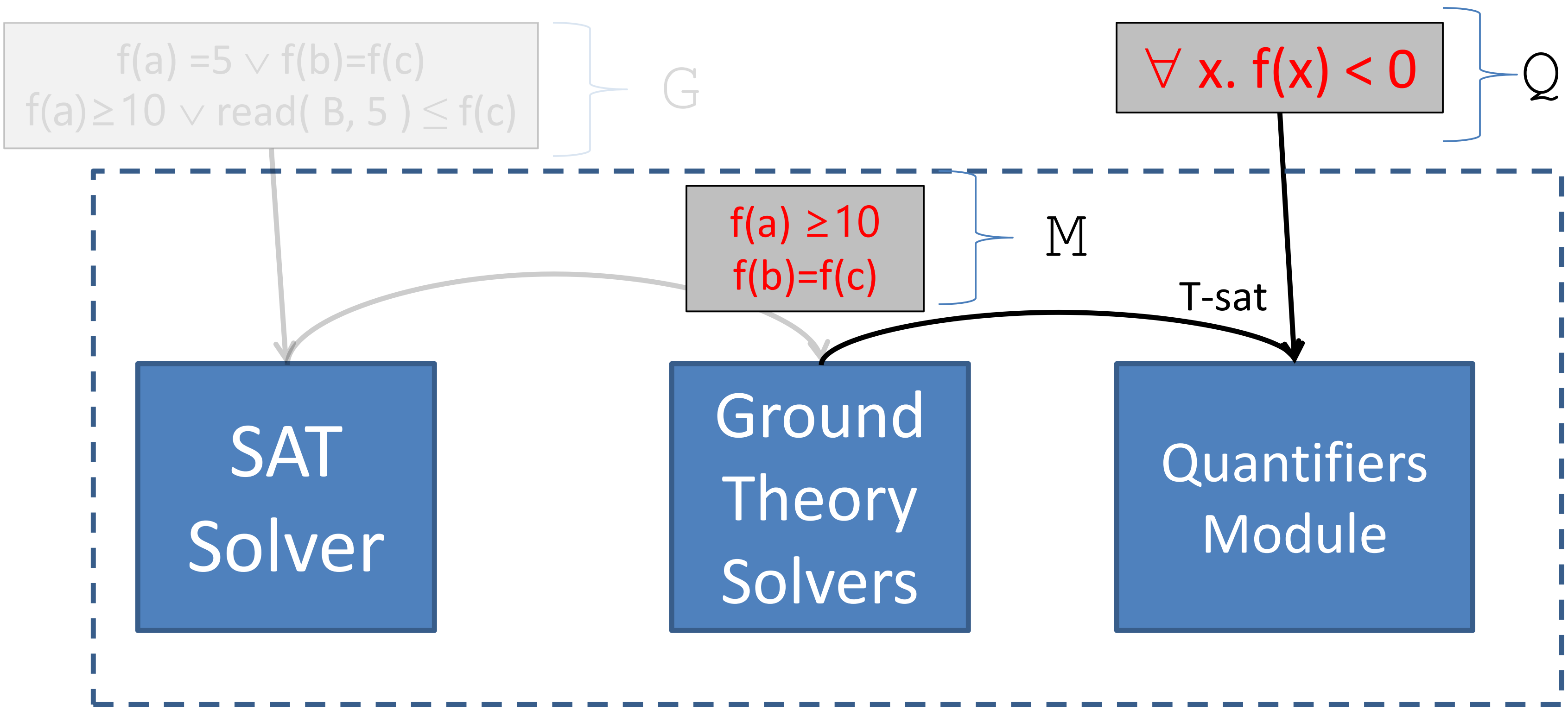


SMT Solver + Quantified Formulas



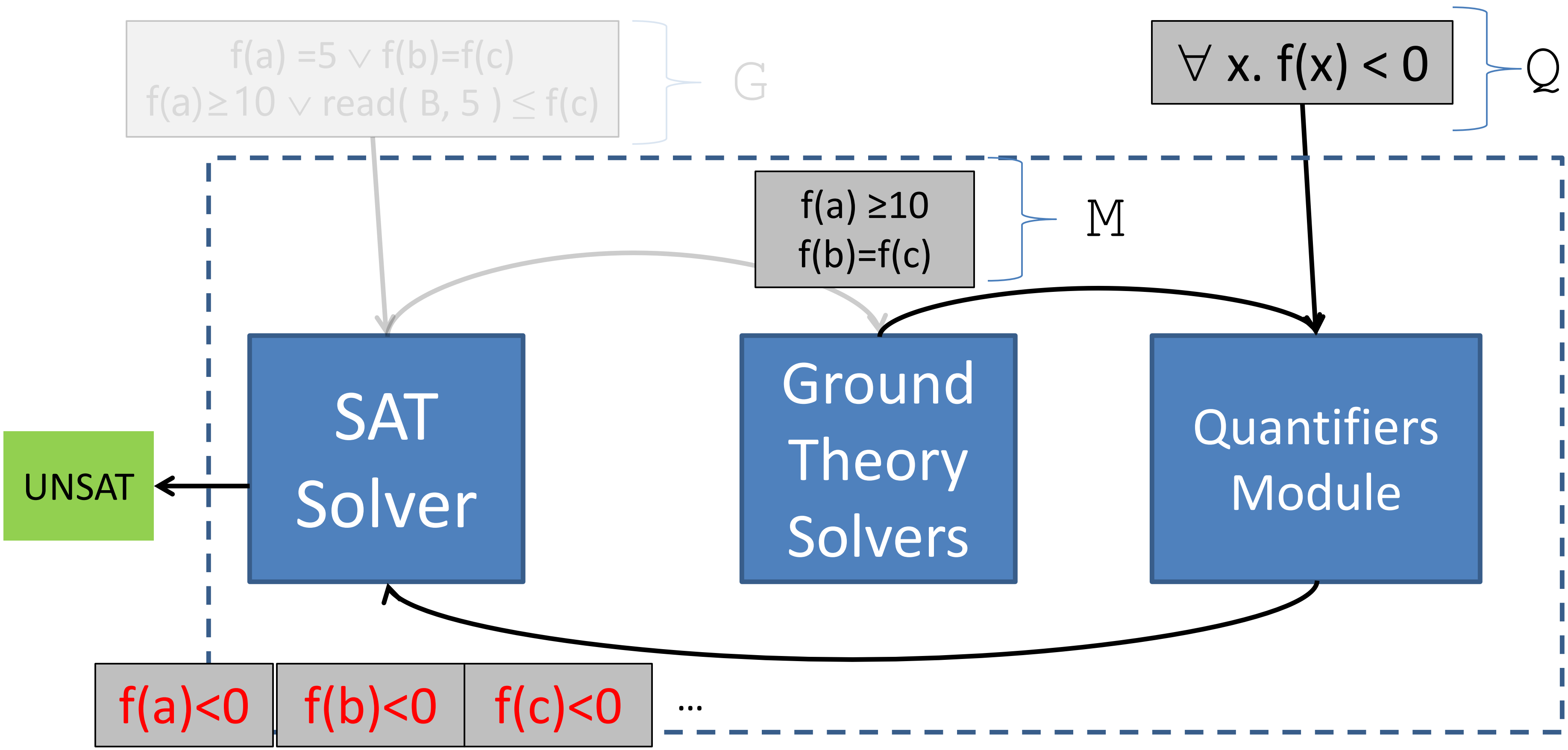
- Find (T-consistent) context $\mathbb{M}$

SMT Solver + Quantified Formulas



- We must answer: *“is $M \cup Q$ consistent?”*
  - Problem is generally undecidable

Quantifier Instantiation



- Instantiation-based approaches:
  - Add instances of quantified formulas, based on some strategy
    - E.g. based on patterns (known as "E-matching")

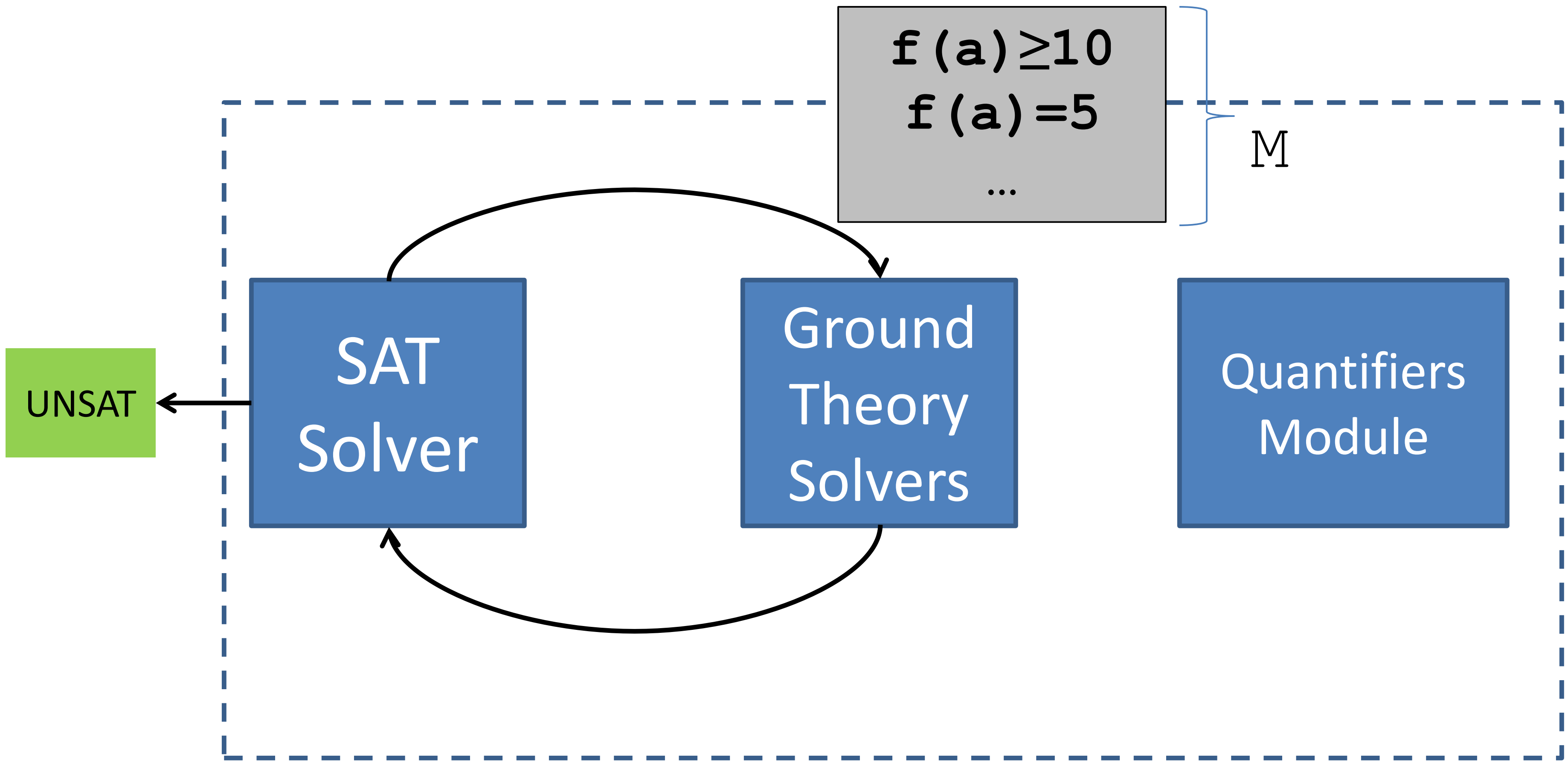
Instantiation-Based Approaches

- Complete approaches:
  - E.g. Complete instantiation, local theory extensions, finite model finding, Inst-Gen
    - Cons: only work for limited fragments
- General approaches:
  - Heuristic E-matching
    - Cons: only for UNSAT, highly heuristic, often inefficient

Motivation

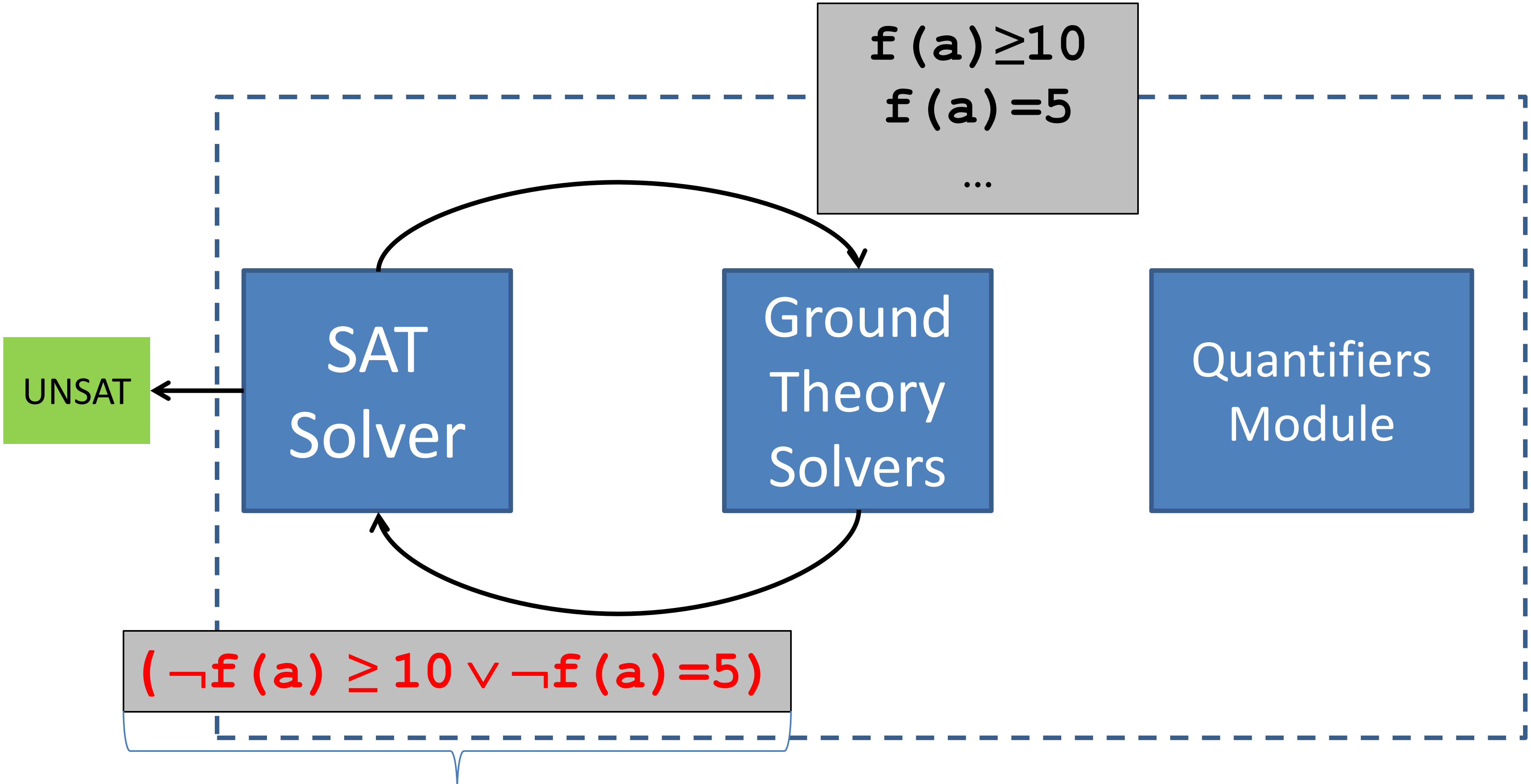
- In this talk: new method for quantified formulas
  - Goals:
    - Reduce dependency on heuristic methods
    - Applicable to arbitrary quantified formulas
  - Not goals:
    - Completeness (thus, focus only on UNSAT)

Ground Theories : Conflicts



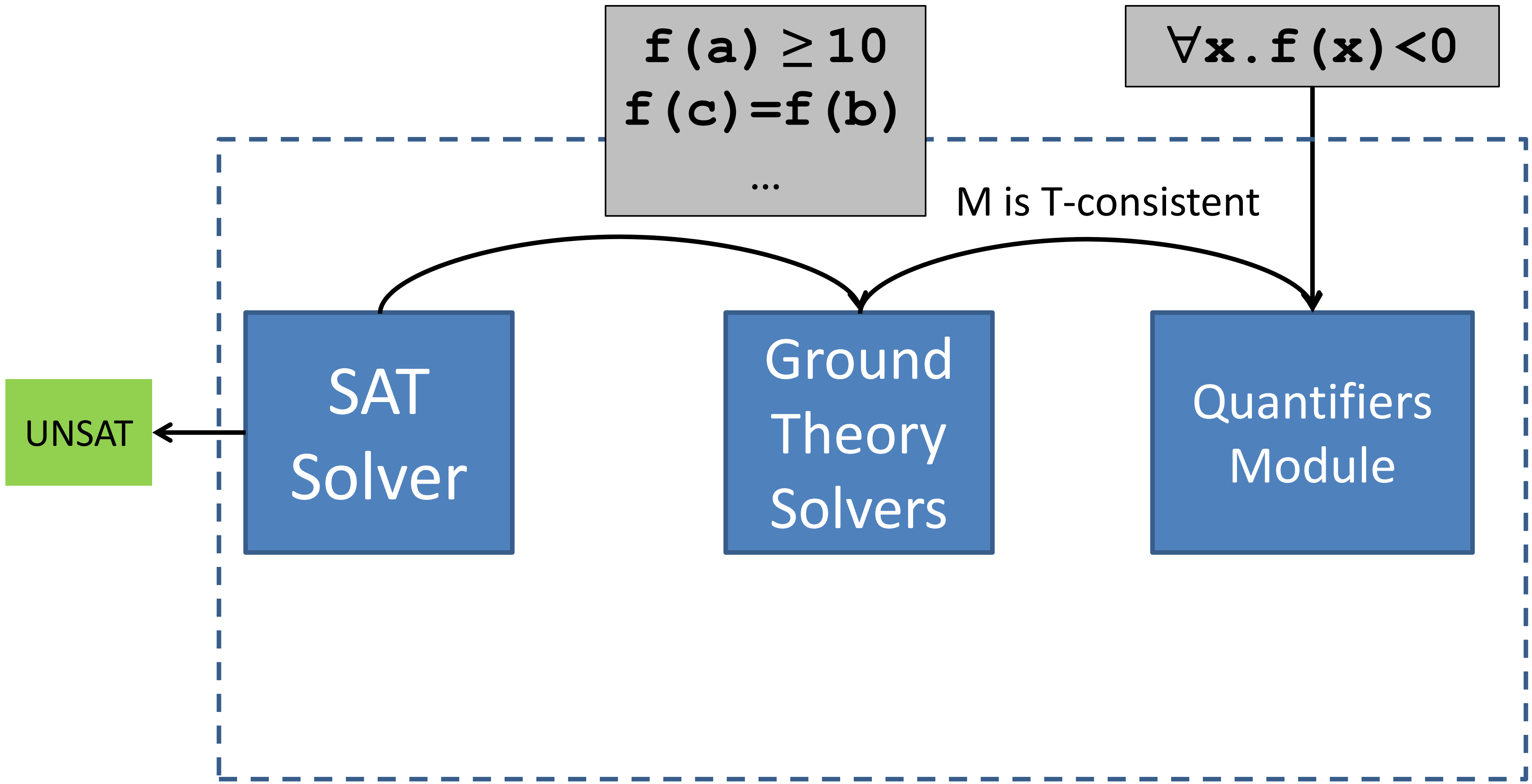
- If M is inconsistent according to ground theory,

Ground Theories : Conflicts



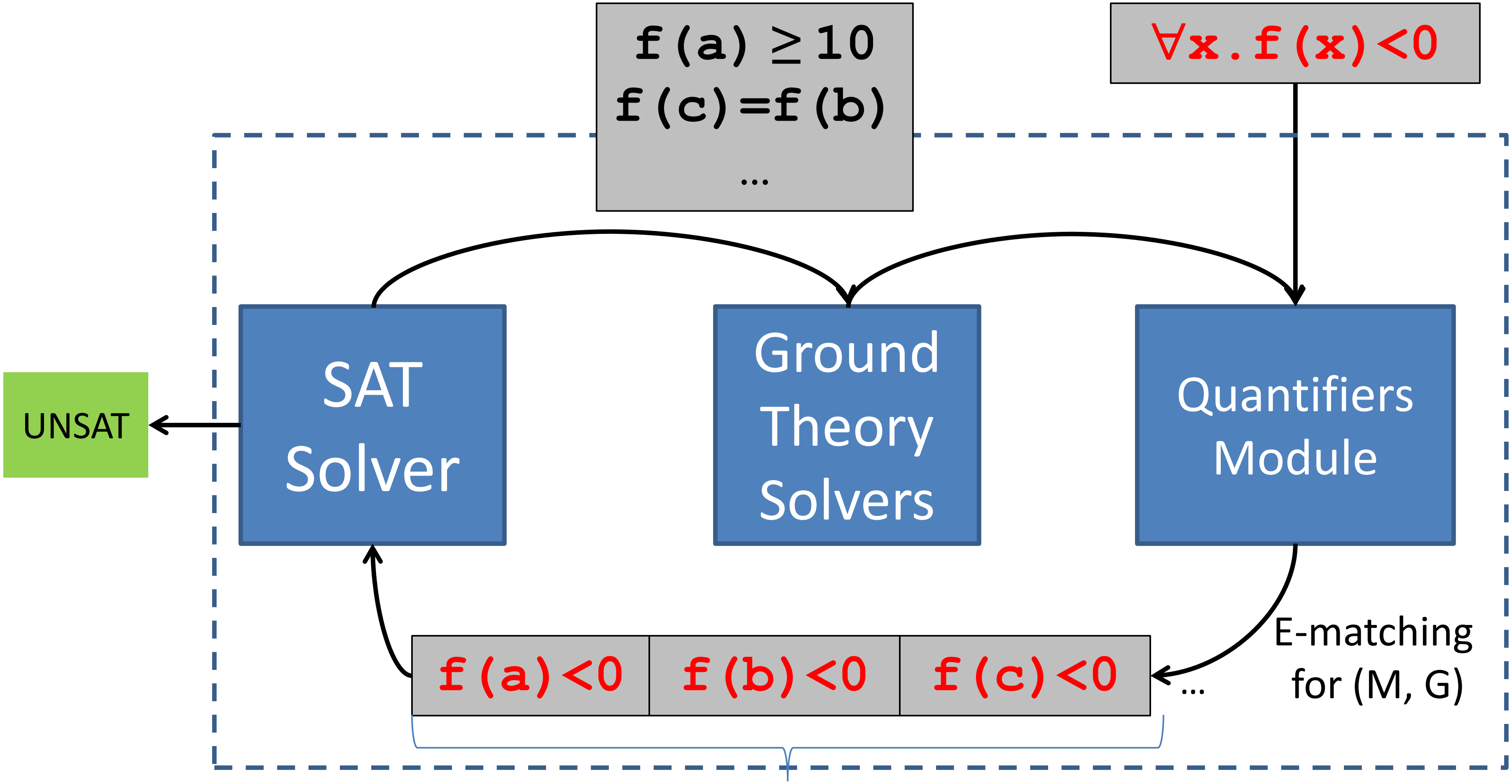
- Ground theory solver reports a single conflict clause
  - Typically, can be determined efficiently

Quantifiers : Heuristic Instantiation?



- The decision problem for $M \cup Q$ is undecidable,

Quantifiers : Heuristic Instantiation?



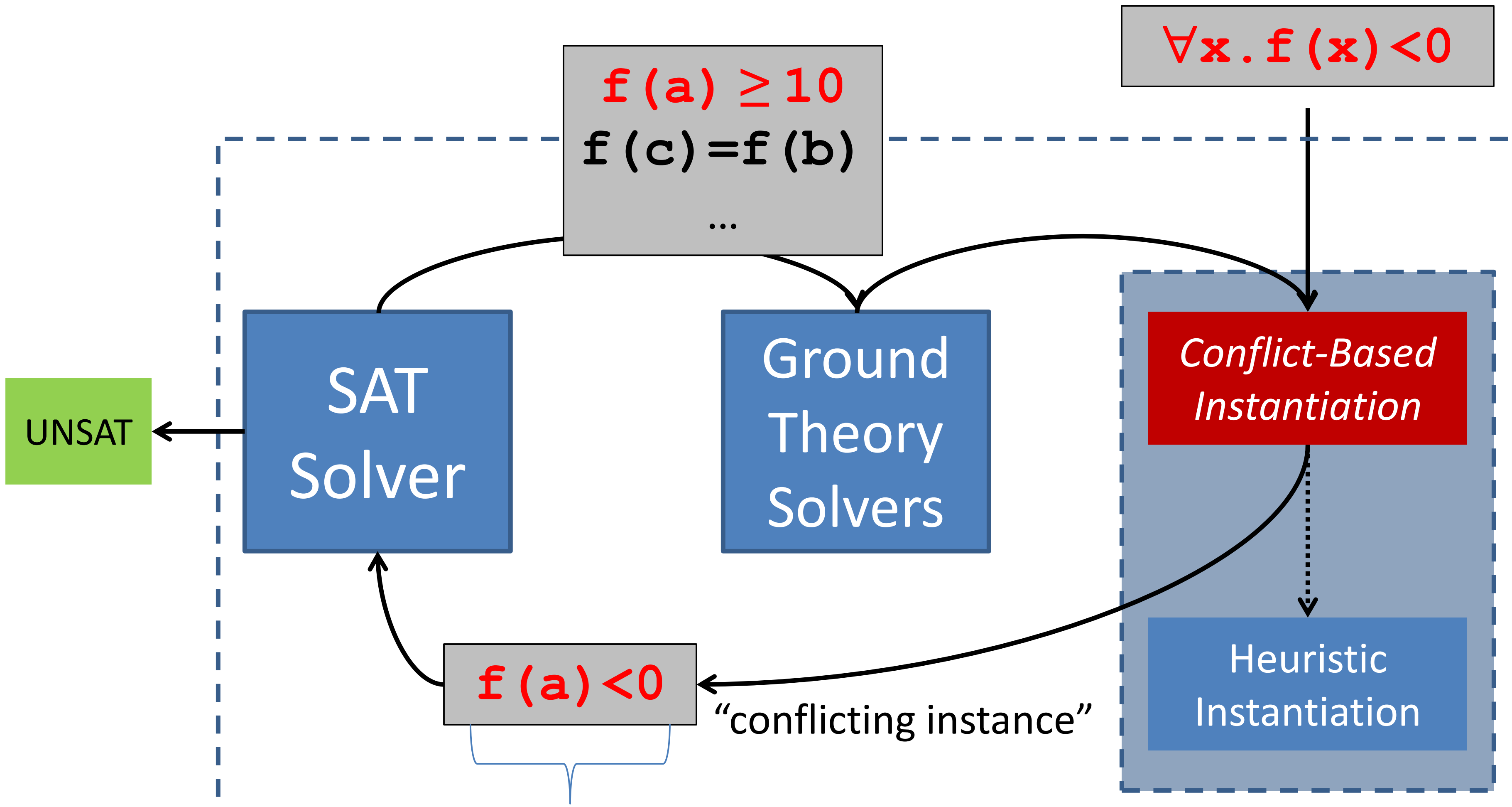
- Add a potentially large set of instances, heuristically
  - This can overload the ground solver

Conflicting Instances

$\Rightarrow$ *Can we make the quantifiers module behave more like a theory solver?*

- Idea: find cases when $\mathrm{M} \cup \mathrm{Q}$ is UNSAT:
  - Find grounding substitution σ
    - Such that $\mathrm{M} \models_T \neg \mathrm{Q}\sigma$
- $\mathrm{Q}\sigma$ is a *conflicting instance*

Conflict-Based Instantiation



- First, determine if a conflicting instance exists
  - If not, resort to heuristic instantiation

Limit of Approach

- *Caveat*: No complete method will determine whether a conflicting instance exists for ($\mathbb{M}$,$\mathbb{Q}$)
- Thus, our approach:
  1. Uses an incomplete procedure to determine a conflicting instance for ($\mathbb{M}$, $\mathbb{Q}$)
  2. If not, resort to E-matching for ($\mathbb{M}$, $\mathbb{Q}$)

  $\Rightarrow$ *In practice, Step 1 succeeds for a majority of ($\mathbb{M}$, $\mathbb{Q}$)*

E-matching vs Conflicting Instances

$$\left.\boxed{\begin{array}{c} \overbrace{g(b)}^{\text{Ground term}} \neq f(a) \\ b = h(a) \end{array}}\right\} \mathbb{M}$$

$$\left.\boxed{\forall x.\ f(x) = \underbrace{g(h(x))}_{\text{Trigger term}}}\right\} \mathbb{Q}$$

- In example, g(h(x)) matches ground term g(b)
  - That is:
    - $\mathbb{M} \models_T g(b)=g(h(x))\sigma$, for $\sigma = \{x \rightarrow a\}$

$\Rightarrow$ *E-matching for (M,Q) returns* σ

E-matching vs Conflicting Instances

$$\underbrace{\boxed{\begin{array}{c} g(b) \neq f(a) \\ b = h(a) \end{array}}}_{\mathrm{M}} \qquad \underbrace{\boxed{\forall x.\ f(x) = g(h(x))}}_{\mathrm{Q}}$$

- In this example, for $\sigma = \{ x \rightarrow a \}$:
  1. Ground terms match each sub-term from Q
     - $\mathrm{M} \models_T g(b)=g(h(x))\sigma$
     - $\mathrm{M} \models_T f(a)=f(x)\sigma$
  2. …and the body of Q is falsified:
     - $\mathrm{M} \models_T f(x) \neq g(h(x))\sigma$

$\Rightarrow M \cup Q\sigma$ *is UNSAT*

E-matching vs Conflicting Instances

$\left.\begin{array}{c} g(b) \neq f(a) \\ b = h(a) \end{array}\right\} M$

$\left.\forall x.\ f(x) = g(h(x))\right\} Q$

- In this example, for $\sigma = \{ x \rightarrow a \}$:
  1. Ground terms match each sub-term from Q
     - $M \models_T g(b)=g(h(x))\sigma$
     - $M \models_T f(a)=f(x)\sigma$
  2. …and the body of Q is falsified:
     - $M \models_T f(x) \neq g(h(x))\sigma$

In paper, limit T to EUF

$\Rightarrow M \cup Q\sigma$ *is UNSAT*

E-matching vs Conflicting Instances

$$\left.\boxed{\begin{array}{c} g(b)\neq f(a) \\ b=h(a) \end{array}}\right\}\mathbb{M} \qquad \left.\boxed{\forall x.\ f(x) = g(h(x))}\right\}\mathbb{Q}$$

- Consider *flat form* of Q:

$$\boxed{\begin{array}{c} \forall x\ y_1\ y_2\ y_3. \\ \underbrace{y_1 = f(x) \wedge y_2 = g(y_3) \wedge y_3 = h(x)}_{\text{Matching constraints } \mu} \Rightarrow \underbrace{y_1 = y_2}_{\text{Flattened body } \Psi} \end{array}}$$

- Conflicting substitution σ for ($\mathbb{M}$, $\mathbb{Q}$) is such that:
  - $\mathbb{M}$ entails $\mu\sigma$
  - $\mathbb{M}$ entails $\neg\Psi\sigma$

Equality-Inducing Instances

$$\left.\begin{array}{c} g(b)\neq c \\ d=f(a) \\ b=h(a) \end{array}\right\} \mathbb{M} \qquad \left.\forall x.\ f(x) = g(h(x))\right\} \mathbb{Q}$$

- *What if we relax constraint 2?*
  - Modified example, for $\sigma = \{ x \rightarrow a \}$:
    1. Ground terms match each sub-term from $\mathbb{Q}$
       - $\mathbb{M} \models_{\mathcal{T}} g(b)=g(h(x))\sigma$
       - $\mathbb{M} \models_{\mathcal{T}} f(a)=f(x)\sigma$
    2. …but the body of $\mathbb{Q}$ is *not* falsified:
       - $\mathbb{M} \not\models_{\mathcal{T}} f(x)\neq g(h(x))\sigma$

Equality-Inducing Instances

$$\left.\boxed{\begin{array}{c} g(b)\neq c \\ d=f(a) \\ b=h(a) \end{array}}\right\} \mathrm{M} \qquad \left.\boxed{\forall x.\ f(x) = g(h(x))}\right\} \mathrm{Q}$$

- *Still*, it may be useful to add the instance $\mathrm{Q}\ \{ x \rightarrow a \}$
  - In this example, $\mathrm{Q}\ \{ x \rightarrow a \}$ entails $g(b) = f(a)$

$\Rightarrow \{ x \rightarrow a \}$ is an equality-inducing substitution

- Mimics T-propagation done by theory solvers

Instantiation Strategy

InstantiationRound($\mathbb{Q}$, $\mathbb{M}$)
(1) Return a (single) conflicting instance for ($\mathbb{Q}$, $\mathbb{M}$)
(2) Return a set of equality-inducing instances for ($\mathbb{Q}$, $\mathbb{M}$)
(3) Return instances based on E-matching for ($\mathbb{Q}$, $\mathbb{M}$)

- Three configurations:
  - **cvc4** : step (3)
  - **cvc4+c** : steps (1), (3)
  - **cvc4+ci** : steps (1),(2),(3)

Experimental Results

- Implemented techniques in SMT solver CVC4
- UNSAT benchmarks from:
  - TPTP
  - Isabelle
  - SMT Lib
- Solvers:
  - **cvc3, z3**
  - 3 configurations: **cvc4, cvc4+c, cvc4+ci**

UNSAT Benchmarks Solved

	cvc3	z3	cvc4	cvc4+c	cvc4+ci
TPTP	5234	6268	6100	6413	6616
Isabelle	3827	3506	3858	3983	4082
SMTLIB	3407	3983	3680	3721	3747
Total	12468	13757	13638	14117	14445

- Configuration cvc4+ci solves the most (14,445)
  - Against cvc4 : 1,049 vs 235 (+807)
  - Against z3: 1,998 vs 1,310 (+688)
  - 359 that no implementation of E-matching (cvc3, z3, cvc4) can solve

Instantiations for Solved Benchmarks

	TPTP		Isabelle		SMT lib	
	Solved	Inst	Solved	Inst	Solved	Inst
cvc3	5245	627.0M	3827	186.9M	3407	42.3M
z3	6269	613.5M	3506	67.0M	3983	6.4M
cvc4	6100	879.0M	3858	119.M	3680	60.7M
cvc4+c	6413	190.8M	3983	54.0M	3721	41.1M
cvc4+ci	6616	150.9M	4082	28.2M	3747	32.5M

- cvc4+ci
  - Solves the most benchmarks for TPTP and Isabelle
  - Requires almost an order of magnitude fewer instantiations
- Improvements less noticeable on SMT LIB
  - Due to encodings that make heavy use of theory symbols
    - Method for finding conflicting instances is more incomplete

Instances Produced

InstantiationRound(Q, M)
(1) conflicting instance for (Q, M)
(2) equality-inducing instances for (Q, M)
(3) E-matching for (Q, M)

			E-matching		Conflicting		C-Inducing	
		IR	IR	#	IR	#	IR	#
smtlib	**cvc4**	14032	100.0%	60.7M				
	cvc4+c	51696	24.3%	41.0M	75.7%	39.1K		
	cvc4+ci	58003	20.0%	32.3M	71.6%	41.5K	8.4%	51.5K
TPTP	**cvc4**	71634	100.0%	879.0M				
	cvc4+c	201990	21.7%	190.1M	78.3%	158.2K		
	cvc4+ci	208970	20.3%	150.4M	76.4%	160.0K	3.3%	41.6K
Isabelle	**cvc4**	6969	100.0%	119.0M				
	cvc4+c	18160	28.9%	54.0M	71.1%	12.9K		
	cvc4+ci	21756	22.4%	28.2M	64.0%	13.9K	13.6%	130.1K

- Conflicting instances found on ~75% of IR
- cvc4+ci :
  - Requires 3.1x more instantiation rounds w.r.t. cvc4
  - Calls E-matching 1.5x fewer times overall
    - As a result, adds 5x fewer instantiations

Details on Solved Problems

- For the 30,081 benchmarks we considered:
  - – cvc4+ci solves more (14,445) than any other
  - – 359 are solved *uniquely* by cvc4+c or cvc4+ci
    - Techniques increase precision of SMT solver
  - – cvc4+ci does not use E-matching 21% of the time
    - 94 benchmarks unsolved by E-matching implementations
    - Techniques reduce dependency on heuristic instantiation

Competitions : CASC J7

- Partly due to techniques:
  - Won TFA division
  - Finished only behind Vampire/E(s) in FOF division

Typed First-order Theorems +*-/	CVC4 1.4-TFA	Princess 140704	SPASS+T 2.2.19	SPASS+T 2.2.20	Beagle 0.9	Zipperpos 0.4-TFF
Solved/200	179/200	176/200	173/200	173/200	173/200	80/200
Av. CPU Time	4.47	11.81	3.44	3.57	5.49	6.57
Solutions	0/200	0/200	173/200	173/200	0/200	80/200
µEfficiency	797	307	402	402	623	313
SOTAC	0.22	0.21	0.19	0.19	0.20	0.27
Core Usage	1.30	1.19	1.83	1.79	1.21	0.99
New Solved	33/50	35/50	30/50	30/50	28/[illegible]	44/50

First-order Theorems	Vampire 2.6	ET 0.1	E 1.9	VanHEls 1.0	CVC4 1.4-FOF	iProver 1.4	leanCoP 2.2	Prover9 1109a	Zipperpos 0.4-FOF	Muscadet 4.4	Princess 140704
Solved/400	375/400	339/400	321/400	310/[illegible]00	215/400	216/400	158/400	95/400	73/400	32/400	134/400
Av. CPU Time	13.19	29.31	22.88	17.[illegible]9	46.03	18.11	55.15	41.45	28.81	19.74	69.31
Solutions	372/400	339/400	321/400	310/[illegible]00	215/400	214/400	158/400	95/400	73/400	30/400	0/400
µEfficiency	571	361	466	1[illegible]8	228	216	129	119	75	47	17
SOTAC	0.22	0.18	0.17	0.[illegible]7	0.15	0.16	0.14	0.14	0.13	0.12	0.13
Core Usage	1.00	1.00	1.00	1.0[illegible]	1.00	1.00	1.00	1.00	1.00	0.99	1.22
New Solved	5/6	5/6	0/6	0/6	0/[illegible]	6/6	0/6	0/6	0/6	0/6	0/6

Competitions : SMT COMP 2014

- Partly due to techniques:
  - Official winner in 11 division with quantifiers
  - (Unofficially) beat z3 in AUFLIA, UFLIA, UF, …

UF

Division COMPLETE: The winner is CVC4

Solver	Errors	Solved	Not Solved	Remaining	CPU Time (on solved instances)	Weighted medal score weight = 3.452
CVC4	0	2732	98	0	87682.16	3.217
[Z3]	0	1802	1028	0	21936.93	1.400
CVC3	0	1682	1148	0	31862.96	1.219
veriT	0	1410	1420	0	7880.76	0.857

Summary and Future Work

- Conflict-based method for quantifiers in SMT
  - Supplements existing techniques
  - Improves performance, both in:
    - Number of instantiations required for UNSAT
    - Number of UNSAT benchmarks solved
- Future work:
  - More incremental instantiation strategies
  - Specialize techniques to other theories
    - Handle quantified formulas containing (e.g.) linear arithmetic
  - Completeness criteria

Thank You

- Solver is publicly available:

  `http://cvc4.cs.nyu.edu/`

- Techniques enabled by option:

  “`cvc4 --quant-cf` …”