Extending Satisfiability Modulo Theories to Quantifed Formulas

Andrew Reynolds
University of Iowa
September 24, 2012

Overview

- Satisfiability Modulo Theories (SMT)
  - Challenge of quantifiers in SMT
- SMT approaches to quantifiers
  - Heuristic Instantiation/E-matching
  - Model-Based Quantifier Instantiation
  - Finite Model Finding
- Automated Theorem Proving
- Current Research
  - CVC4 + Finite Model Finding

Satisfiability Modulo Theories (SMT)

- SMT solvers:
  - Are powerful tools for determining satisfiability of ground formulas
    - Built-in decision procedures for many theories
      - Arithmetic, Arrays, BitVectors, Datatypes, …
  - Have applications in:
    - Software/Hardware verification
    - Planning and scheduling
    - Design automation
  - Had significant performance improvement in past 10 years
  - Key to success of many industrial verification applications

Strengths of SMT Solvers

- Performance
  - Built on top of high performance SAT solvers
  - Use fast decision procedures for theories
  - Designed to work incrementally
- Usability
  - Enable rich encodings of problems
  - Accept SMT LIB v2 common language
  - Produce more than SAT/UNSAT answer:
    - Models, proofs, unsat cores, interpolants, …

What is SMT?

$(a = 5 \vee \text{select}(R, a) = b) \wedge g(5) \geq g(a) + 1$

- Satisfiability Modulo Theories:
  - Determine if there exists satisfying assignment
    - If so, return SAT
    - Return UNSAT if none can be found
  - Satisfying assignment must be *T*-consistent

$$(a = 5 \vee select(R, a) = b) \wedge g(5) \geq g(a) + 1$$

Abstract to boolean satisfiability problem

$$\Downarrow$$

$$(A \vee B) \wedge C$$

$$(a = 5 \lor \text{select}(R, a) = b) \land g(5) \geq g(a) + 1$$

$$\Downarrow$$

$$(A \lor B) \land C$$

A: True

C: True

Find satisfying assignment: A, C

$$(a = 5 \lor \text{select}(R, a) = b) \land g(5) \geq g(a) + 1$$

$$\Downarrow$$

$$(A \lor B) \land C$$

A: True C: True

- *However, A and C are inconsistent according to theory*
  - a = 5 and g(5) ≥ g(a) + 1 cannot both be true according to UF + Int
- Can add additional clause:

$$(\neg A \lor \neg C)$$

$$(a = 5 \lor \mathrm{select}(R, a) = b) \land g(5) \geq g(a) + 1$$

$$\Downarrow$$

$$(A \lor B) \land C \land$$

A: False

B: True

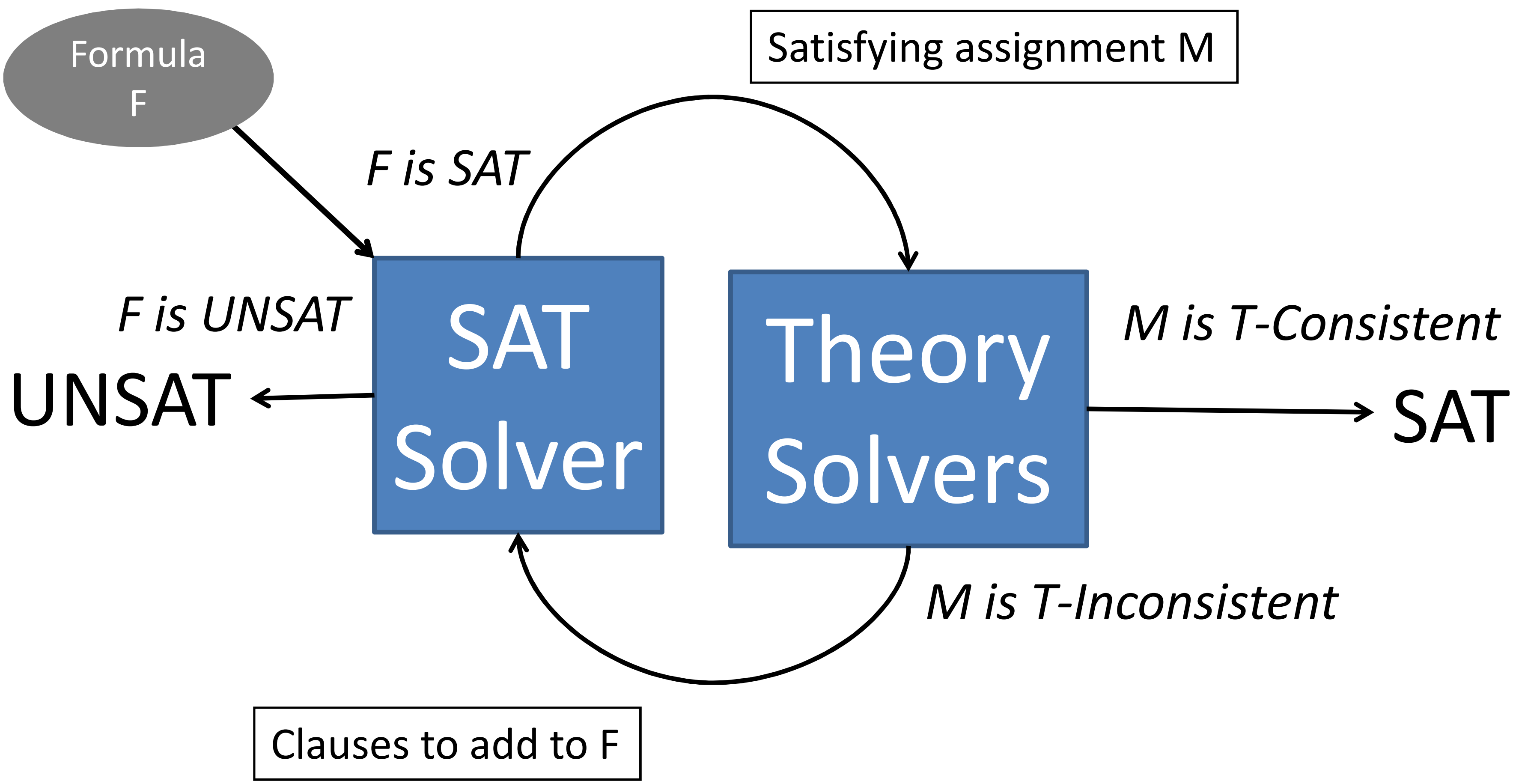
C: True

$$(\neg A \lor \neg C)$$

$\neg A$: True

$\Rightarrow$ answer SAT

DPLL(T) Architecture [Nieuwenhuis et al 03]



Challenge: Quantifiers in SMT

$\forall x.\ f(x+1) \geq f(x) + 1 \wedge (f(2) = 5 \vee select(R, a) = b)$

For all integers x…

- Treat each quantified formula as literal, as before

$$\forall x.\ f(x+1) \geq f(x) + 1 \wedge (f(2) = 5 \vee select(R, a) = b)$$

$$\Downarrow$$

$$A \wedge (B \vee C)$$

True True

- Find satisfying assignment: A, B

$\Rightarrow$ *Problem:* In general, determining consistency of quantified formulas is undecidable

Quantifier Instantiation

- Divide problem into:
  - Ground portion G, and quantified portion Q:

$$\underbrace{\ldots,\ f(\ 2\ ) = 5,\ \ldots.}_{G} \qquad \underbrace{\ldots,\ \forall x.\ f(\ x{+}1\ ) \geq f(\ x\ ) + 1,\ \ldots.}_{Q}$$

- Determine if G is T-inconsistent
  - If not, *instantiate* Q with some set of ground terms

Quantifier Instantiation

- Check again if G is T-inconsistent
  - If not, repeat

…, $f(2) = 5$, ….

$f(1) \geq f(0) + 1$

$f(2) \geq f(1) + 1$

$f(3) \geq f(2) + 1$

….

G

…, $\forall x.\ f(x+1) \geq f(x) + 1$, ….

instantiate

Q

$\Rightarrow$ *Sound but incomplete procedure*

Instantiation-Based Approaches

- Given set of literals (G, Q):
  - Set of ground constraints G
  - Set of quantified assertions Q
- Questions:
  - (1) How to choose instantiations for Q
  - (2) When can we answer SAT?

Pattern-Based Quantifier Instantiation

[Detlefs et al 05]

- *Idea:* Determine instantiations heuristically
  - Find terms in Q with same shape as ground terms in G
- Example:

$$\underbrace{a = b,\ f(a, a) \neq b,}_{G}\ \underbrace{\forall x.\ f(x, b) = a}_{Q}$$

  - Consider f(x, b) as *trigger* term
  - Determine if f(a, a) and f(x, b) match,
    - Modulo set of background equalities E = { a=b }
  - Here, f(x, b) E-matches f(a, a) with { $x \rightarrow a$ }
    - Add instantiation [a/x] for quantifier
      - Adds constraint f(a, b) = a, leading to T-inconsistency

Pattern-Based Quantifier Instantiation

- Challenges:
  - Trigger selection is highly non-trivial
  - Sensitive to syntactic changes in formulas
  - Matching loops can occur
    - Repeating pattern of generated terms, f(a), f(f(a)), f(f(f(a))), …
  - # instantiations may explode
  - It is an incomplete procedure, i.e. cannot answer SAT
- As a result, tends to:
  - Discover easy conflicts if they exist
  - Otherwise, overloads SMT solver with instances
    - Run indefinitely or answer unknown

Model-Based Quantifier Instantiation (MBQI) [Ge, deMoura 08]

- *Idea:* Try to show that no instance of Q falsifies the current model M for G
- To check if an instance of $\forall x.\ F$ falsifies M:
  - $\Rightarrow$ Suffices to check if $\neg F^M[e/x]$ is satisfable
- If unsat, then no instance of $\forall x.\ F$ falsifies M
- Otherwise, we must refine M
  - – Instantiate $\forall x.\ F$ using sat assignment to $\neg F^M[e/x]$

MBQI : Example

$$\underbrace{P(a, a),\ a \neq b,}_{G}\ \underbrace{\forall z.\ \neg P(z, b)}_{Q}$$

Find model M : $\{ a, b \}$, } representatives

$P^M := \lambda\, xy.\ (x=a \wedge y=a)$ } interpretations for uninterpreted symbols in Q

MBQI : Example

$$\underbrace{P(a, a), a \neq b,}_{G} \; \underbrace{\forall z. \neg P(z, b)}_{Q}$$

Find model M : $\{ a, b \}$,

$P^M := \lambda xy. (x=a \wedge y=a)$

$\neg P^M(z, b) \equiv \neg(z=a \wedge b=a) \equiv \text{true}$

- Is $(\neg \text{true})[e/z] \equiv \text{false}$ satisfiable?

$\Rightarrow$ *unsat, therefore Q does not falsify M*

MBQI as Model Refinement

$$\underbrace{P(a, a), a \neq b,}_{G} \ \underbrace{\forall z. \neg P(z, b)}_{Q}$$

Find model M' : $\{ a, b \}$,
$P^{M'} := \lambda xy.\ x = a$

$$\neg P^{M'}(z, b) \equiv \neg (z = a)$$

- Is $(\neg\neg (z = a))[e/z] \equiv (z = a)[e/z] \equiv (e = a)$ satisfiable?
  - $\Rightarrow$ *sat with valuation* $\{ e \rightarrow a \}$
- Add instantiation [a/z], add $\neg P(a, b)$ to G
  - – Guaranteed to rule out M' on subsequent iterations

Model-Based Quantifier Instantiation

- Challenges:
  - Hard to determine interpretations in M
    - Default values chosen heuristically
  - External model checking calls are expensive
- Typically:
  - Is effective at answering SAT for simple cases
  - Can be paired with E-matching to improve coverage

Finite Model Finding

- *Idea:* Build model for G that is small enough to test Q exhaustively
- Given set of literals (G, Q):
  - Find a "smallest" model for G
    - One with fewest # of ground equivalence classes
  - Try *every* instance of Q in the model
    - Feasible if the number of instances is *finite*
  - If every instance is true in model, answer SAT

Why Small Models?

- Easier to test against quantifiers
  - Given quantified formula $\forall x_1 \ldots x_n.\ F(x_1 \ldots x_n)$
    - Naively, we require k^n instantiations
      - Where k is the cardinality of sort($x_1 \ldots x_n$)
  - Feasible if either:
    - Both n and k are small
    - We can recognize redundant instantiations
      - Use Model-Based Quantifier Instantiation

SMT vs ATP

- SMT Solvers
  - Strengths:
    - Efficient decision procedures for theories
    - Theories increase expressivity
  - Weaknesses:
    - Ability to handle quantifiers is limited
- Automated Theorem Provers (ATP)
  - Strengths:
    - Advanced methods for quantified clauses
  - Weaknesses:
    - Nearly no support for theories
      - Omission is intentional, as this leads to undecidability

Resolution-Based Theorem Proving

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \text{ Res}$$
where $\sigma = mgu(A, B)$.

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \text{ Factor}$$
where $\sigma = mgu(A, B)$.

- Sound and complete
  - If input is unsat, we will eventually derive $\bot$
  - When clause set is saturated wrt rules, input is sat
- Additional rules for equational reasoning
  - Paramodulation, superposition
- Optimizations
  - Term Indexing
  - Redundancy Elimination (i.e. clause subsumption)

ATP Approaches

- Deciding fragments of first-order logic (EPR):
  - Model evolution calculus [Baumgartner, Tinelli 03]
    - Darwin [Fuchs et al 04]
  - Inst-Gen [Korovin, Ganzinger 03]
    - iProver [Korovin 06]
- Finite model finding:
  - SEM-style model finding [Zhang, Zhang 96]
  - *MACE-style model finding* [McCune 94]
    - Paradox [Clausen, Sorenson 03]

MACE-Style Model Finding

- *Idea:* Check for models of fixed size by generating a corresponding ground queries
- Given (G, Q):
  - First, create ground problem G, $F_{G,Q,1}$
    - If sat, then model of size 1 exists
  - If unsat, create ground problem G, $F_{G,Q,2}$
    - If sat, then model of size 2 exists
    - …
- Will eventually find *finite* model, if one exists

MACE-Style Model Finding : Example

$$\underbrace{a \neq b,\ b = c,}_{G}\ \underbrace{\forall x.\ f(x) = x}_{Q}$$

- No model of size 1 can be found…
- Generate ground problem G, $F_{G,Q,2}$:
  - *Use domain constants d_1, d_2*

a ≠ b, b = c,

$(a = d_1 \lor a = d_2)$, …
$(f(d_1) = d_1 \lor f(d_1) = d_2)$,
$(f(d_2) = d_1 \lor f(d_2) = d_2)$,
— each term must be equal to some d_i

$f(d_1) = d_1$, $f(d_2) = d_2$ — Q is true for all d_i

$\Rightarrow$ *SAT*

MACE-Style Model Finding

- Challenges:
  - Introducing constants leads to value symmetries
    - Find identical models modulo renaming of constants

    $\Rightarrow$ Can use static symmetry breaking techniques
  - May produce large # of clauses
    - Must test all instances of quantified clauses

    $\Rightarrow$ Use sort inference to determine a subset of instances that are relevant

    $\Rightarrow$ Use clause splitting to reduce # variables per clause

My Current Research

- New approaches to quantifiers in SMT
- *In this talk:* Finite Model Finding in CVC4
- Approach for (G, Q) consists of:
  - Finding minimal models for G
  - Model checking Q by exhaustive instantiation

Finite Model Finding for SMT

- Similar to MACE-style approaches for (G, Q),
  - Search for models of size 1, 2, 3, etc.
  - Naively, test all instances of Q for fixed model size
- In contrast to MACE-style approaches,
  - Search for models is integrated into DPLL(T)
  - Do not introduce domain constants explicitly
    - Use internal union-find data structure in SMT solver

Finite Model Finding in SMT : Example

$$\underbrace{a \neq b,\ b = c,}_{G}\ \underbrace{\forall x.\ f(x) = x}_{Q}$$

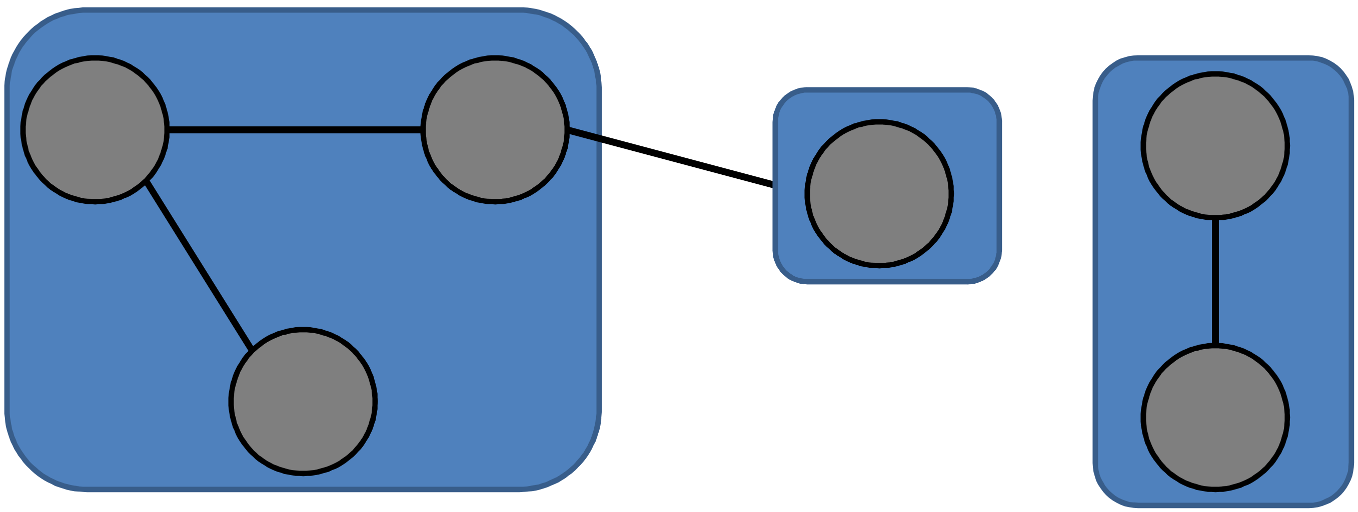
- Using DPLL(T), we find smallest model for G, equivalence classes: { $\underline{a}$ }, { $\underline{b}$, c }
- Instantiate Q with all representative terms:
  - f(a) = a, f(b) = b added to G
- Afterwards : { $\underline{a}$, f(a) }, { $\underline{b}$, c, f(b) }
  - All instances are true in model $\Rightarrow$ answer SAT

Finite Model Finding

- To find small models:
  - Where “smallest” model for sort S means:
    - Fewest # equivalence classes of sort S
  - Try to find models of size 1, 2, 3, … etc.
    - Impose *cardinality constraints*
  - Requires:
    - Control the DPLL(T) search for postulating cardinalities
    - Theory solver for equality + cardinality constraints

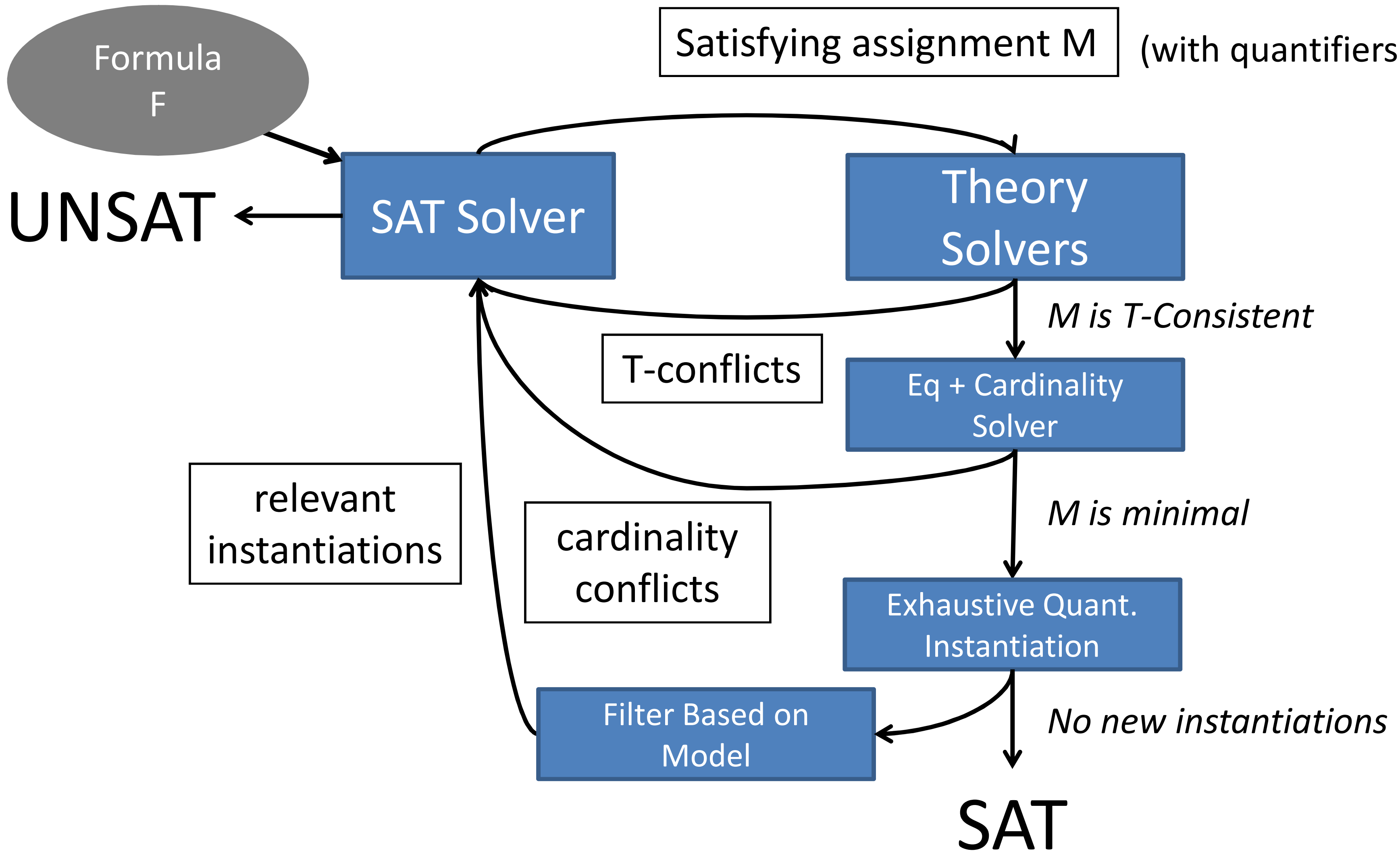
Solver for Eq + Cardinality Constraints

- Maintain disequality graph
  - Nodes are equivalence classes
  - Edges are disequalities
- For cardinality k, interested whether graph is k-colorable



- Partition disequality graph of the solver into *regions* where the edge density is high
  - Discover cliques local to regions
  - Suggest relevant terms to identify

Finite Model Finding for SMT



CVC4 + Finite Model Finding

- Implemented in SMT solver CVC4 [Barrett et al 10]
  - State of the art solver developed by NYU/Iowa
- Preliminary Results
  - Successful as backend to Intel's DVF Tool [Goel et al 12]
    - Effective at finding small countermodels (SAT cases)
    - Added ability to discharge VC's (UNSAT cases)
  - Orthogonal to other approaches
    - Answers SAT in cases where no other solver can

Ongoing Work

- For Equality + Cardinality Constraint Solver:
  - Improved clique finding and reporting
- For Quantifier Instantiation:
  - Incorporate heuristic instantiation
  - Use of iProver’s Inst-Gen calculus
    - Require weaker condition for answering SAT
    - Eliminate the need for exhaustive instantiation

- Questions?