Implementing Branch and Bound Algorithms in SMT

Andrew Reynolds

Two Sigma

July 12, 2016

THE UNIVERSITY OF IOWA

Overview

- Satisfiability Modulo Theories and DPLL(T)
- Finite Model Finding in SMT
  - Branch and bound for finding small models
  - Variants of the approach
  - Relationship to Optimization
- Recent trends, future work

Satisfiability Modulo Theories (SMT)

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

- We are often interested in establishing *T-satisfiability* of formulas with:

Satisfiability Modulo Theories (SMT)

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

- We are often interested in establishing *T-satisfiability* of formulas with:
  - Boolean structure

Satisfiability Modulo Theories (SMT)

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

- We are often interested in establishing *T-satisfiability* of formulas with:
  - Boolean structure
  - Constraints in a background theory T, e.g. UFLIA

Satisfiability Modulo Theories (SMT)

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

- We are often interested in establishing *T-satisfiability* of formulas with:
  - Boolean structure
  - Constraints in a background theory T, e.g. UFLIA
  - …even existential and universal quantifiers

DPLL(T): Basics

$$(P(a) \lor f(b) > a+1)$$
$$(\neg P(b) \lor \forall x.\, P(x))$$
$$(f(b) = a-5 \lor \neg P(a))$$

DPLL(T): Basics

$$(P(a) \vee f(b)>a+1)$$
$$(\neg P(b) \vee \forall x. P(x))$$
$$(f(b)=a-5 \vee \neg P(a))$$

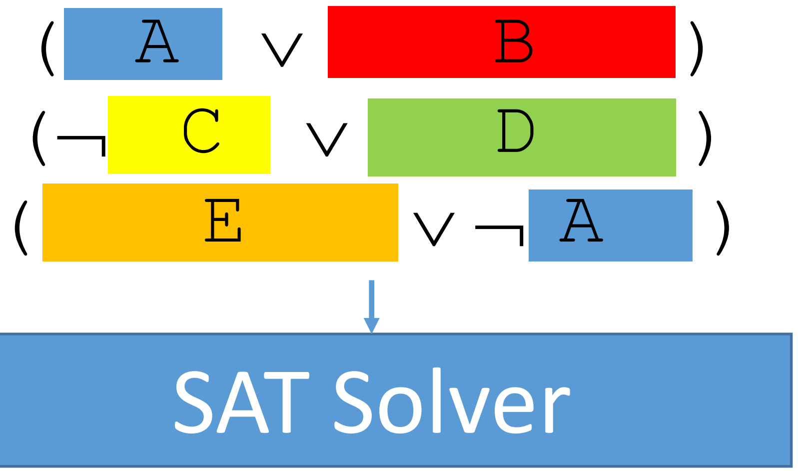
- Consider the propositional abstraction of the formula

DPLL(T): Basics

$$(A \lor B)$$
$$(\neg C \lor D)$$
$$(E \lor \neg A)$$

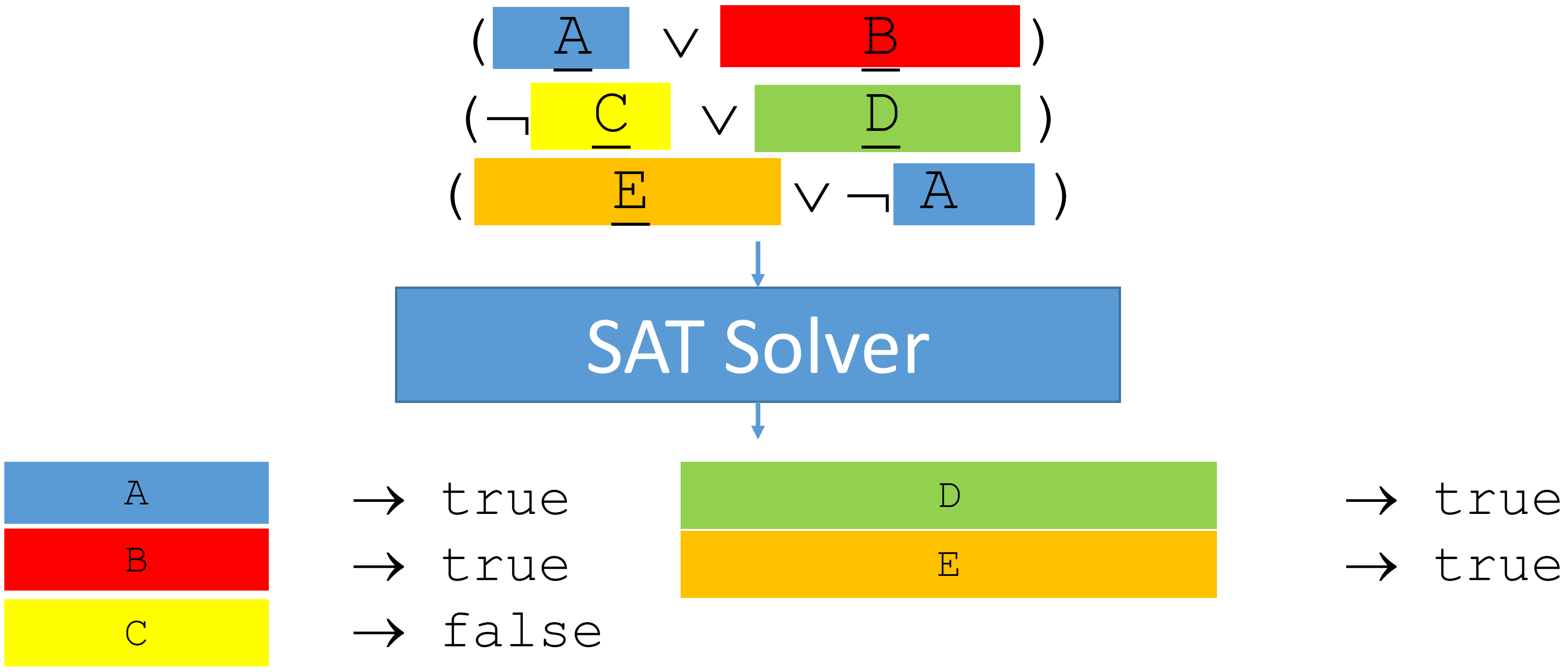
- Consider the propositional abstraction of the formula

DPLL(T): Basics



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

DPLL(T): Basics



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

DPLL(T): Basics

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

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

$(f(b)=a-5 \lor \neg P(a))$

SAT Solver

P(a)	→ true	∀x.P(x)	→ true
f(b)>a+1	→ true	f(b)=a-5	→ true
P(b)	→ false		

- Consider the original atoms

DPLL(T): Basics

$$(P(a) \vee f(b)>a+1)$$
$$(\neg P(b) \vee \forall x.P(x))$$
$$(f(b)=a-5 \vee \neg P(a))$$

$$\downarrow$$

SAT Solver

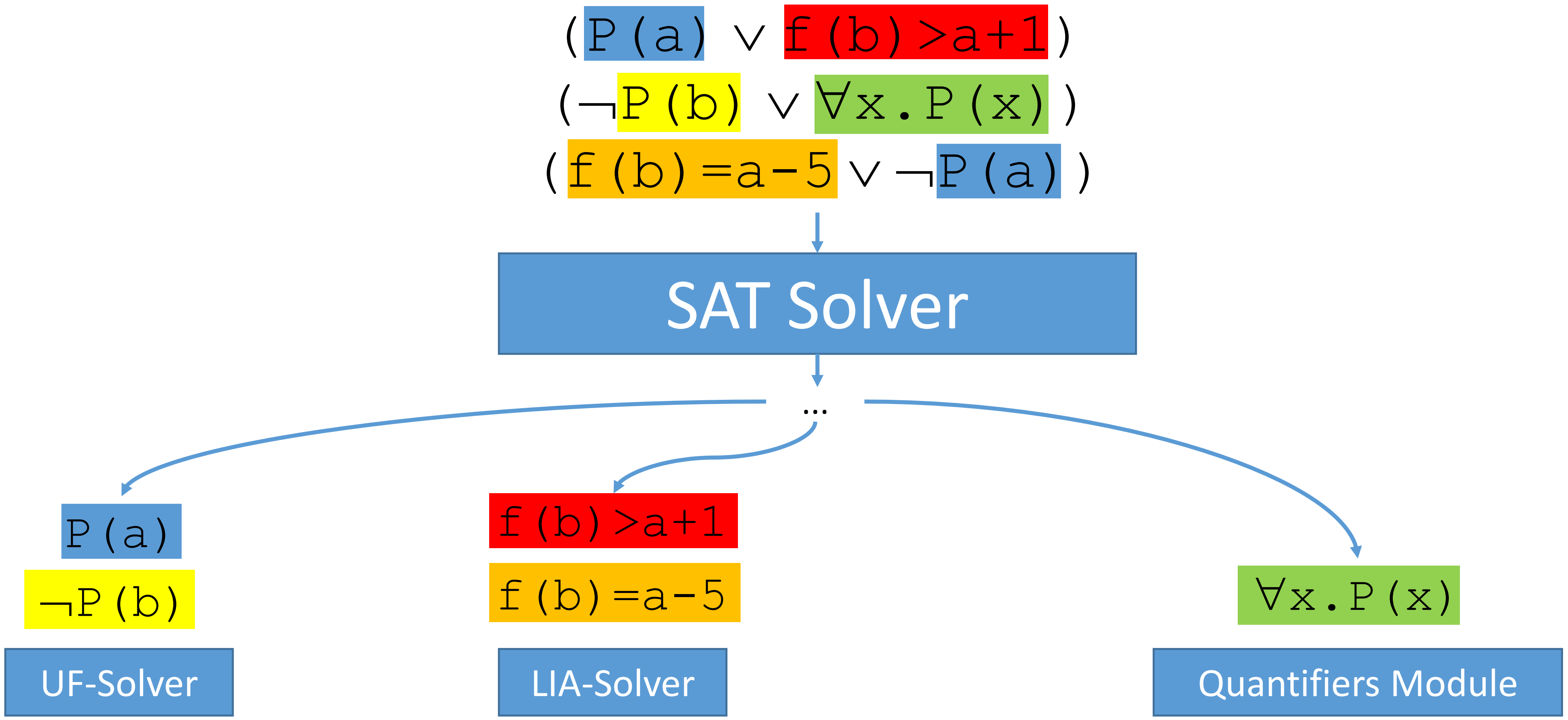
$$\downarrow$$

$$\underbrace{P(a),\ f(b)>a+1,\ \neg P(b),\ \forall x.P(x),\ f(b)=a-5}_{M}$$

$\Rightarrow$ Propositional assignment can be seen as a set of T-literals M

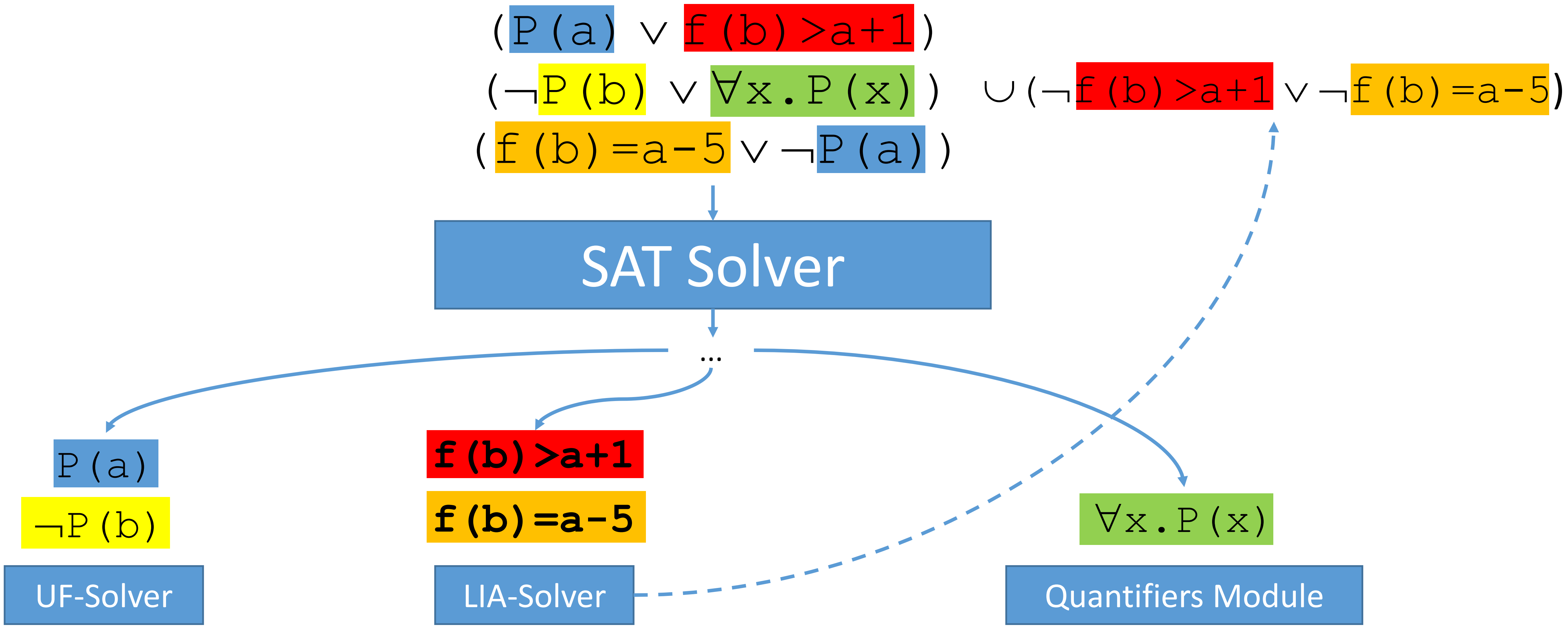
- Must check if M is T-satisfiable

DPLL(T): Basics



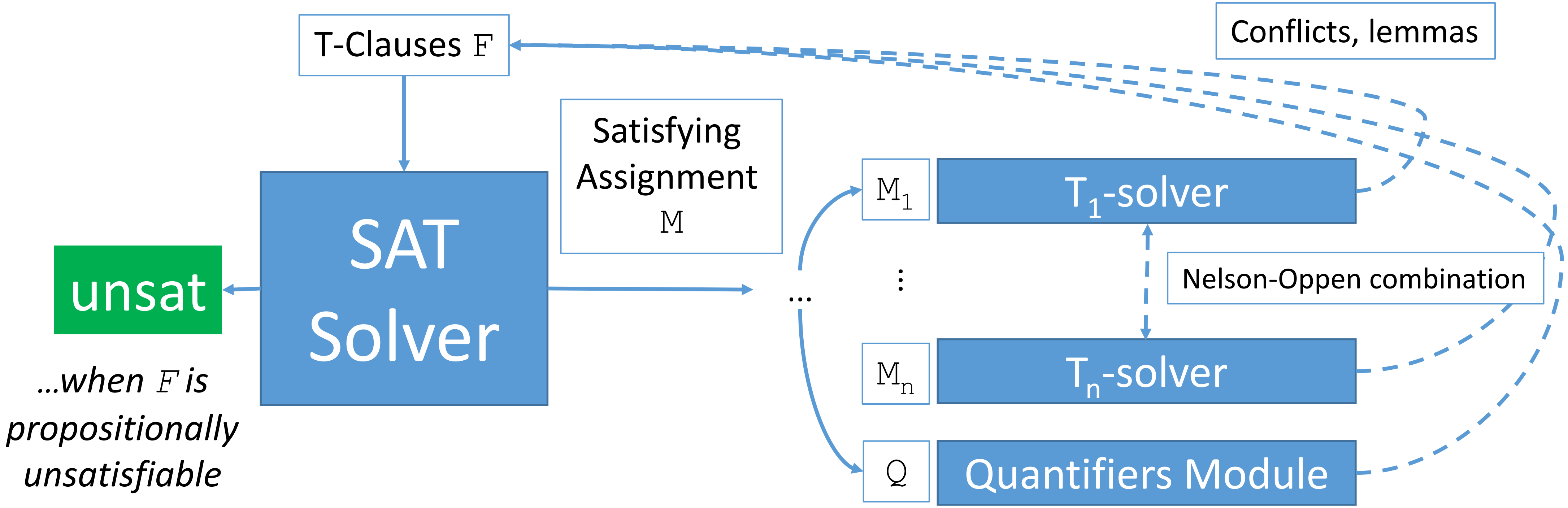
$\Rightarrow$ Distribute ground literals to T-solvers, ∀ literals to quantifiers module

DPLL(T): Basics



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

DPLL(T_1+..+T_n): Overview

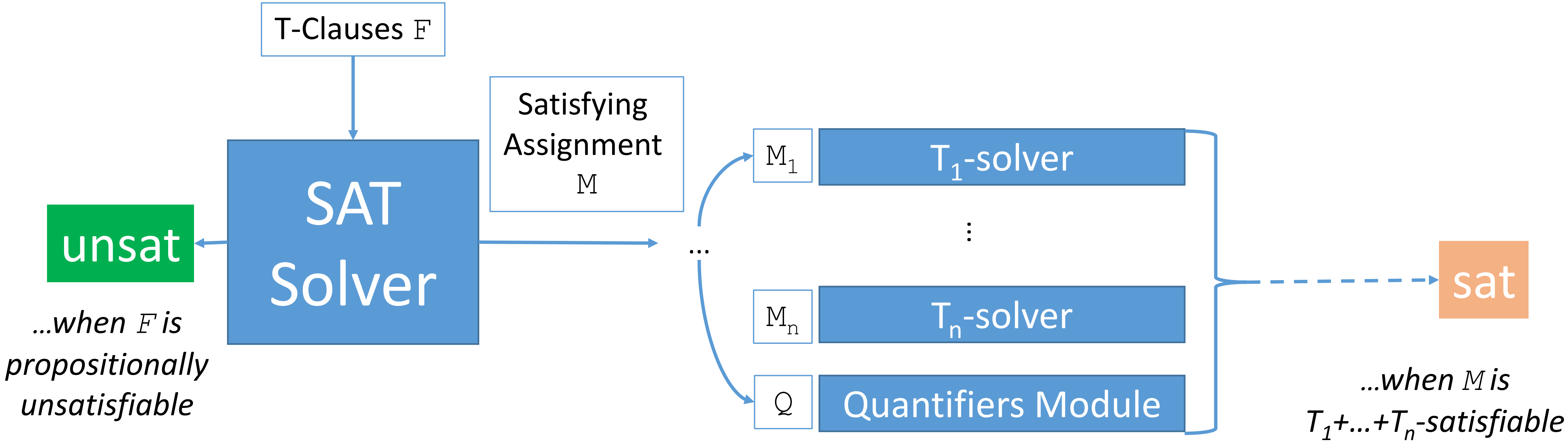


$\Rightarrow$ Each of these components may:

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

[Nieuwenhuis/Oliveras/Tinelli 06]

DPLL(T_1+..+T_n): 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]

Common Theories Supported by SMT Solvers

- SMT solvers support:
  - Arbitrary Boolean combinations of ground *theory* constraints
  - Examples of supported theories:
    - Uninterpreted functions: `f(a)=g(b,c)`
    - Linear real/integer arithmetic: `a≥b+2*c+3`
    - Arrays: `select(A,i)=select(store(A,i+1,3),i)`
    - BitVectors: `bvule(x,#xFF)`
    - Algebraic Datatypes: `x,y:List; tail(x)=cons(0,y)`
    - …
  - ∀ over each of these

Common Theories Supported by SMT Solvers

- SMT solvers support:
  - Arbitrary Boolean combinations of ground *theory* constraints
  - Examples of supported theories:
    - Uninterpreted functions: $\Rightarrow$ Congruence Closure **[Nieuwenhuis/Oliveras 2005]**
    - Linear real/integer arithmetic: $\Rightarrow$ Simplex **[deMoura/Dutertre 2006]**
    - Arrays: $\Rightarrow$ **[deMoura/Bjorner 2009]**
    - BitVectors: $\Rightarrow$ Bitblasting, lazy approaches **[Bruttomesso et al 2007,Hadarean et al 2014]**
    - Algebraic Datatypes: $\Rightarrow$ **[Barrett et al 2007]**
    - …
  - $\forall$ over each of these

SMT Solvers have Partial Support for ∀

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

- Satisfiability problem for ∀ is generally undecidable
- Heuristic Techniques for "unsat":
  - E-matching [Detlefs et al 2003, Ge et al 2007, de Moura/Bjorner 2007]
- Limited Techniques have completeness guarantees:
  - Local theory extensions [Sofronie-Stokkermans 2005]
  - Array fragments [Bradley et al 2006, Alberti et al 2014]
  - Complete Instantiation [Ge/de Moura 2009]
  - Finite Model Finding [Reynolds et al 2013]

SMT Solvers have Partial Support for ∀

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

- Satisfiability problem for ∀ is generally undecidable
- Heuristic Techniques for “unsat”:
  - E-matching [Detlefs et al 2003, Ge et al 2007, de Moura/Bjorner 2007]
- Limited Techniques have completeness guarantees:
  - Local theory extensions [Sofronie-Stokkermans 2005]
  - Array fragments [Bradley et al 2006, Alberti et al 2014]
  - Complete Instantiation [Ge/de Moura 2009]
  - **Finite Model Finding** [Reynolds et al 2013]

$\Rightarrow$ Focus of next slides

Finite Model Finding : Motivation

```
List := cons( head : Int, tail : List ) | nil
```

Signature

```
∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)
…
```

Axioms

```
∃xy:List.rev(append(x,y))≠append(rev(y),rev(x))
```

(Negated) conjecture

↓

CVC4

↓

Conjecture holds

Finite Model Finding : Motivation

```
List := cons( head : Int, tail : List ) | nil
```

Signature

```
∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)
…
```

Axioms

```
∃xy:List.rev(append(x,y))≠append(rev(y),rev(x))
```

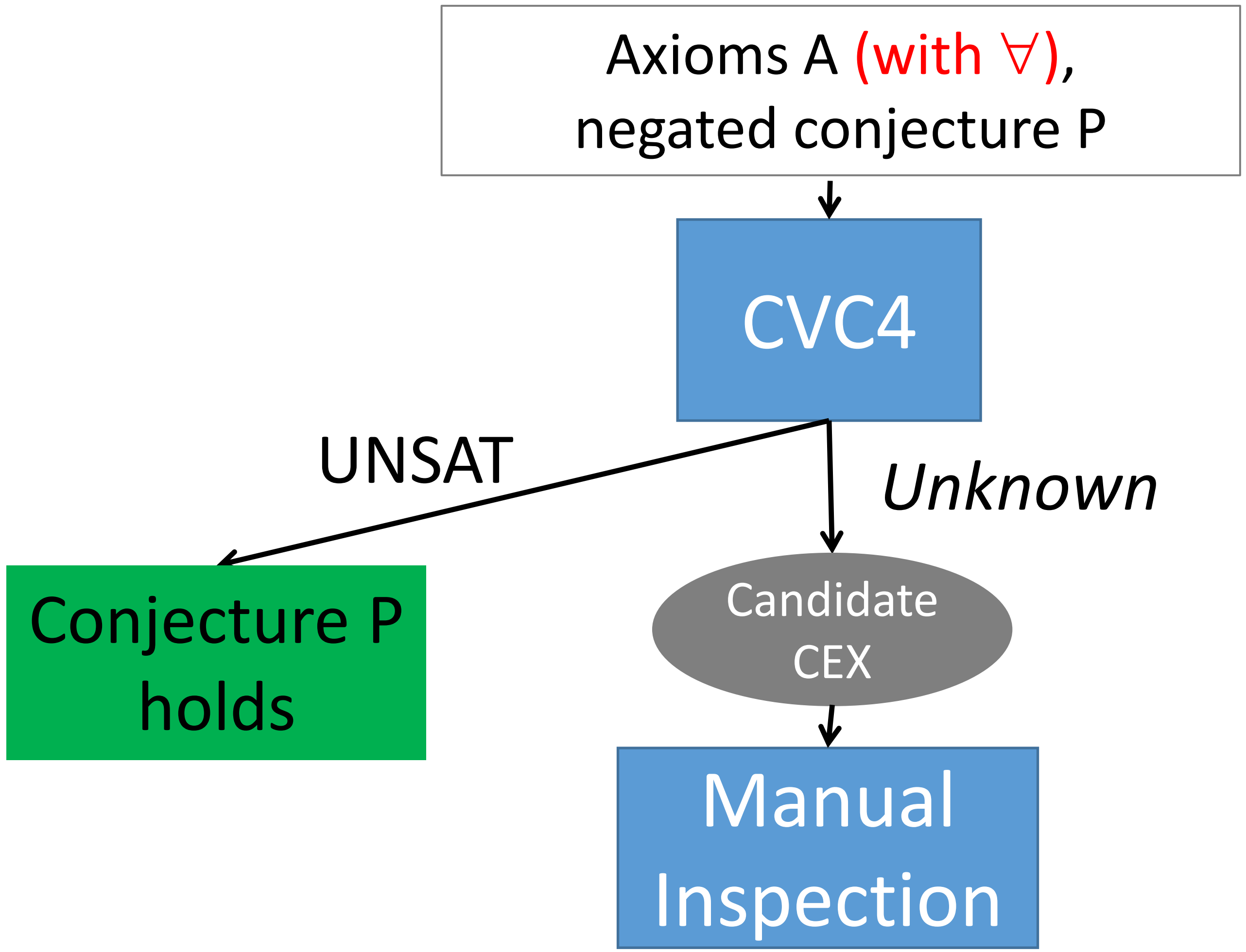
(Negated) conjecture

CVC4

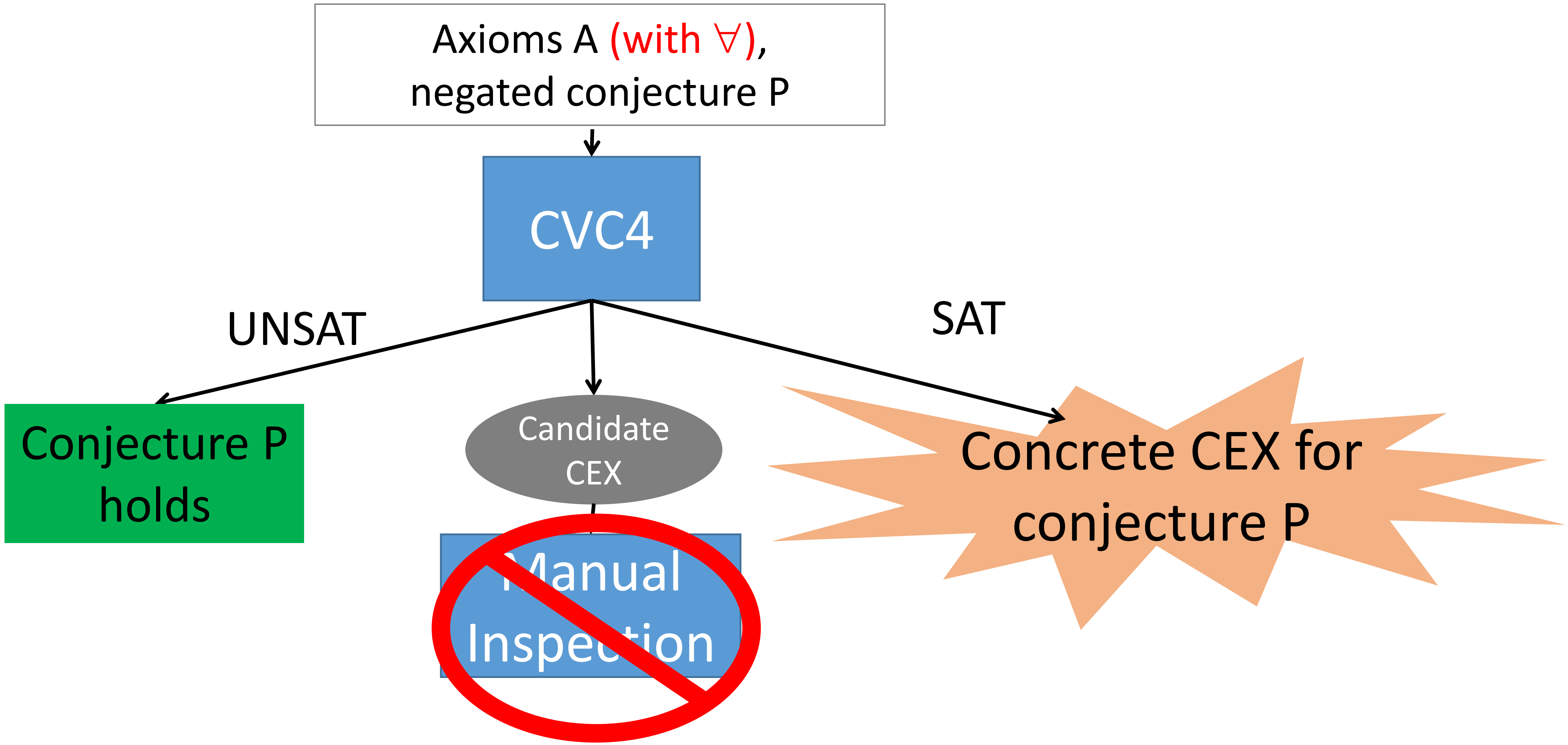
Conjecture holds

…but what if the conjecture does not hold?

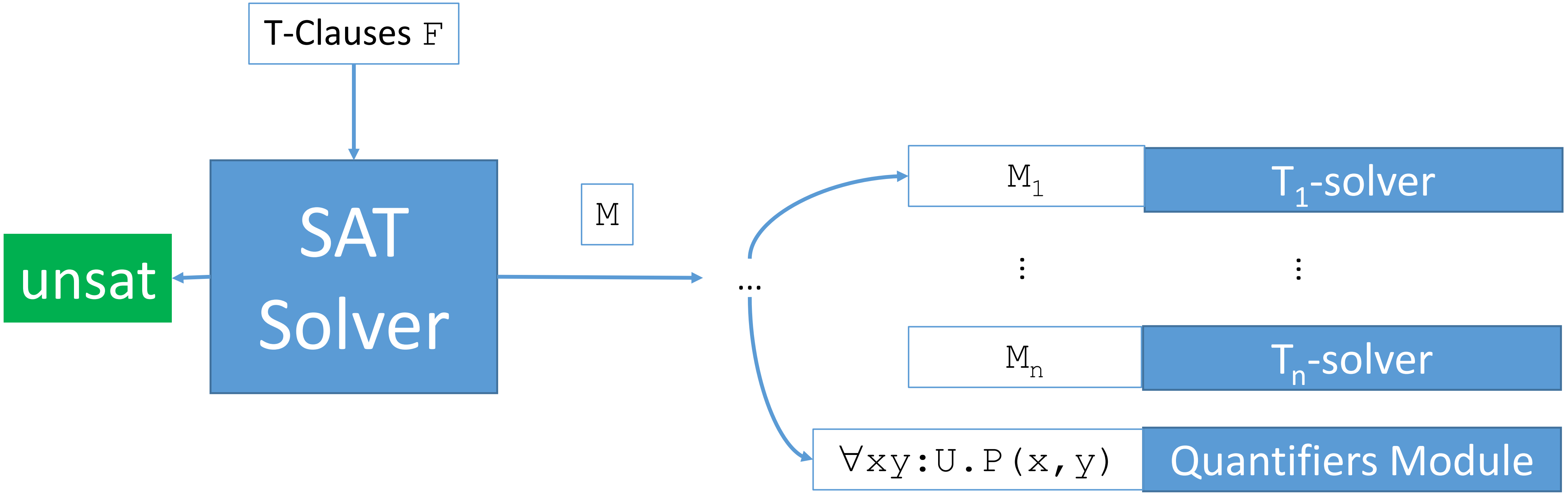
Finite Model Finding : Motivation



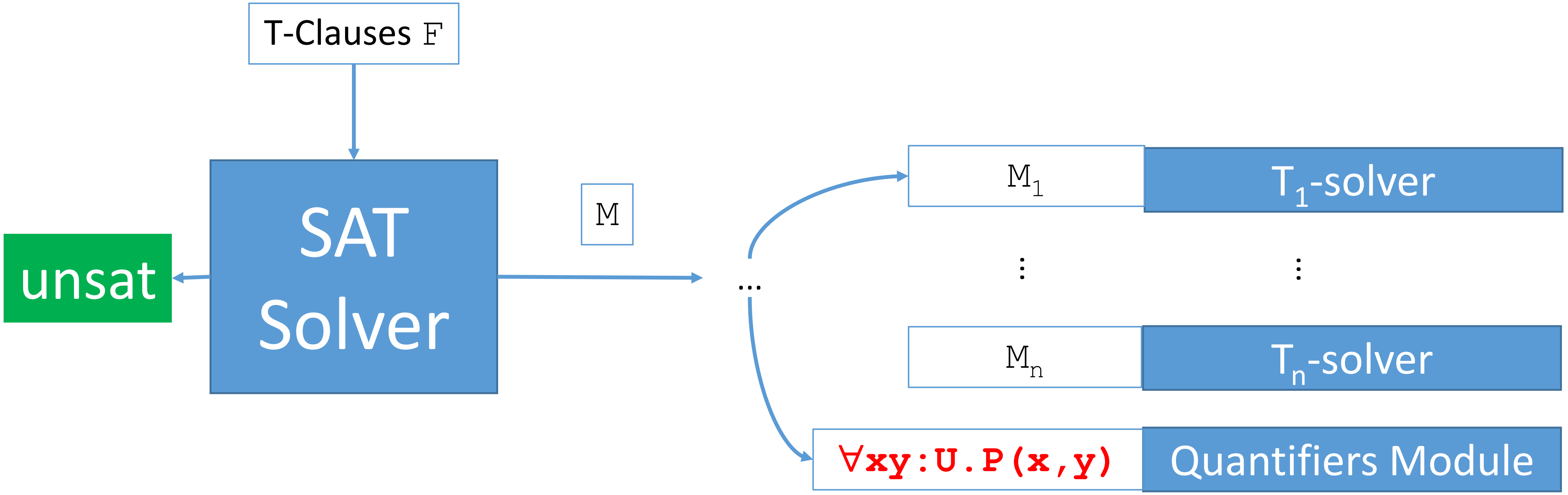
Finite Model Finding : Motivation



Finite Model Finding in DPLL(T)

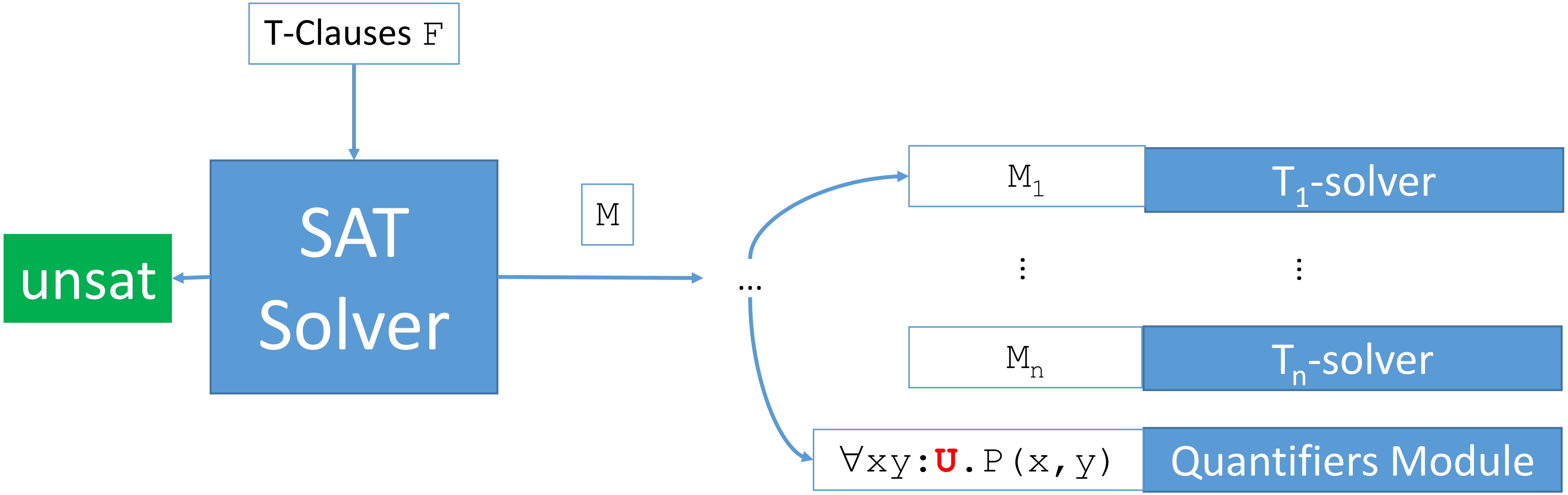


Finite Model Finding in DPLL(T)



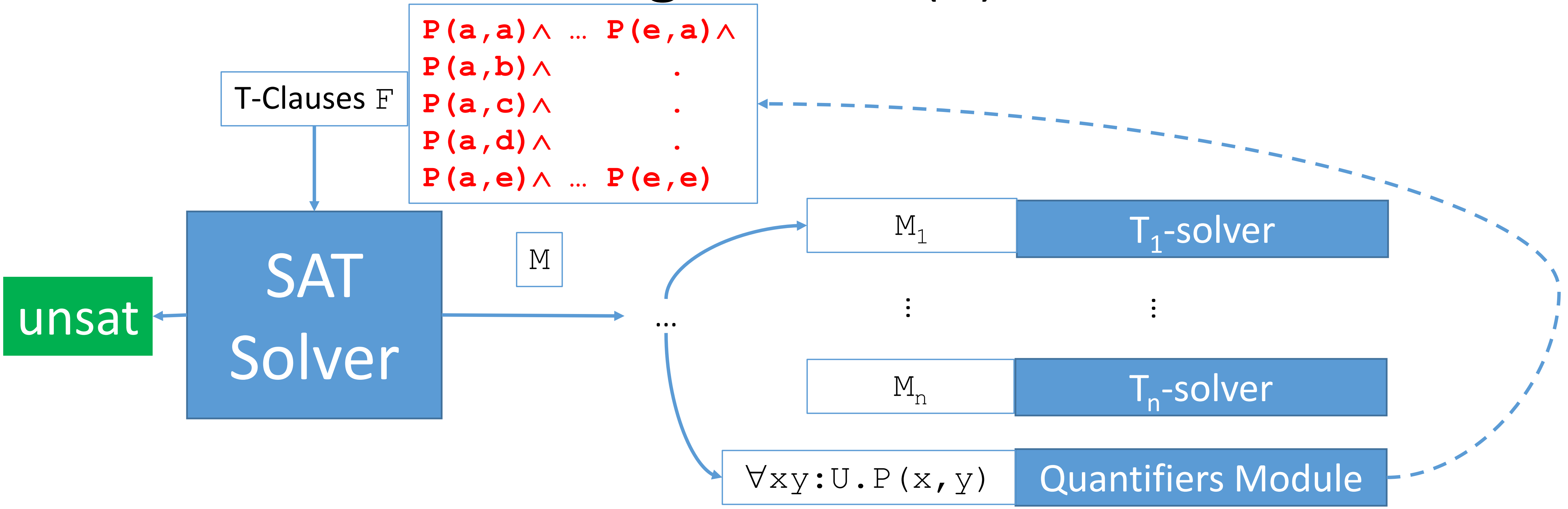
- Given universally quantified formula $\forall$**xy:U.P(x,y)**

Finite Model Finding in DPLL(T)



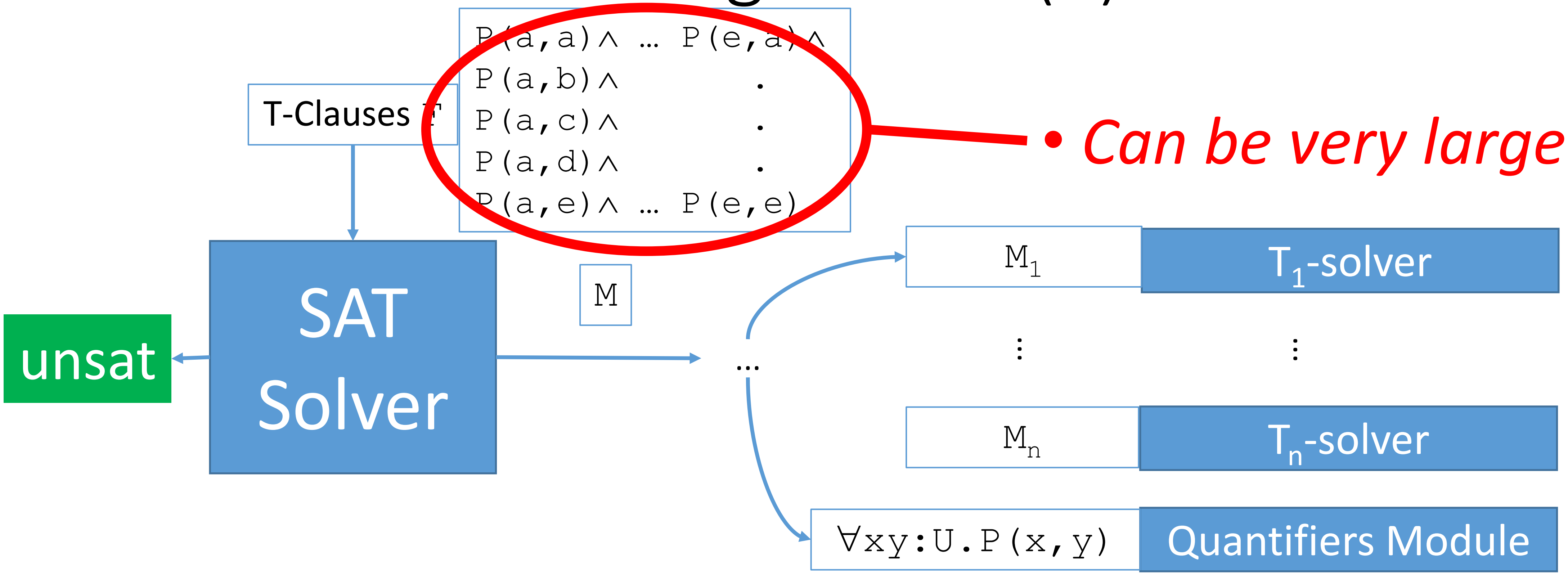
- Given universally quantified formula $\forall$`xy:`**`U`**`.P(x,y)`
  - If **`U`** can be interpreted as finite, e.g. `{`**`a,b,c,d,e`**`}:`

Finite Model Finding in DPLL(T)



- Given universally quantified formula `∀xy:U.P(x,y)`
  - If `U` can be interpreted as finite, e.g. `{a,b,c,d,e}`:
    - Can be reduced to a **finite set of instances**

Finite Model Finding in DPLL(T)



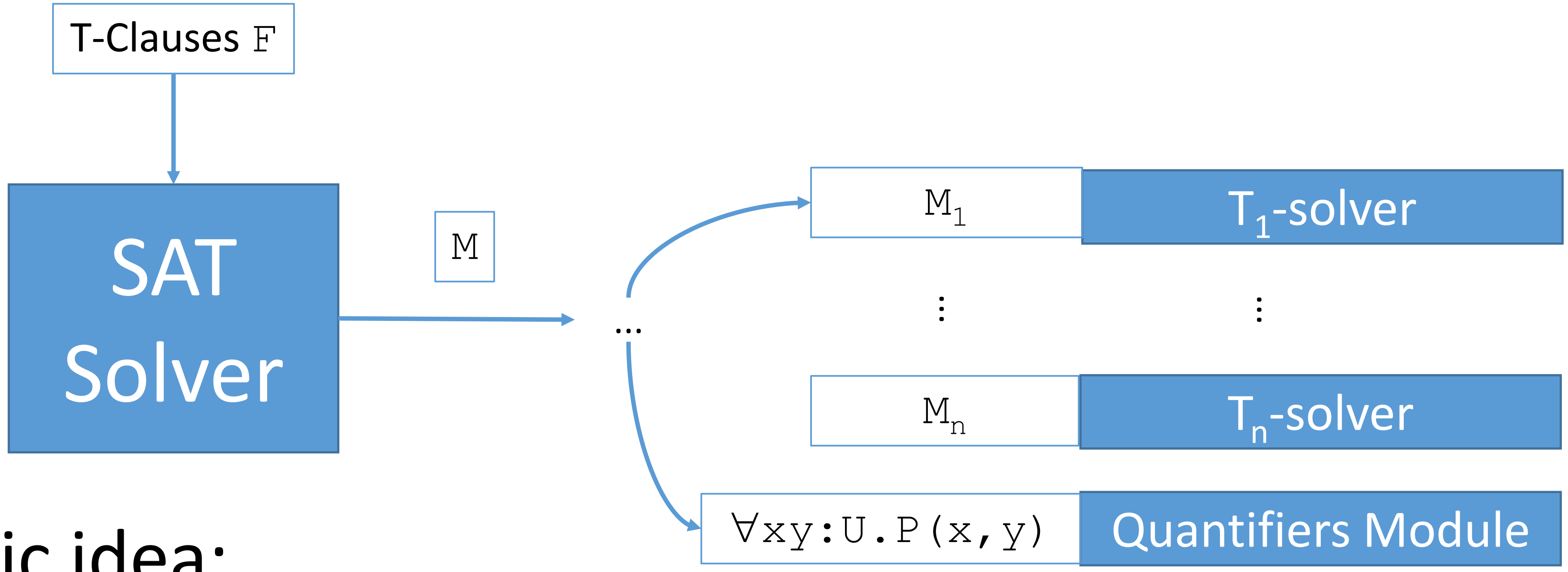
- Given universally quantified formula ∀xy:U.P(x,y)
  - If U can be interpreted as finite, e.g. {a,b,c,d,e}:
    - Can be reduced to a finite set of instances

Finite Model Finding in SMT

- Address large # instantiations by:
  1. Only add instantiations that refine model [Reynolds et al CADE13]
     - Model-based quantifier instantiation [Ge/deMoura CAV 2009]
  2. Minimizing model sizes [Reynolds et al CAV13]
     - Find interpretation that minimizes the #elements in U

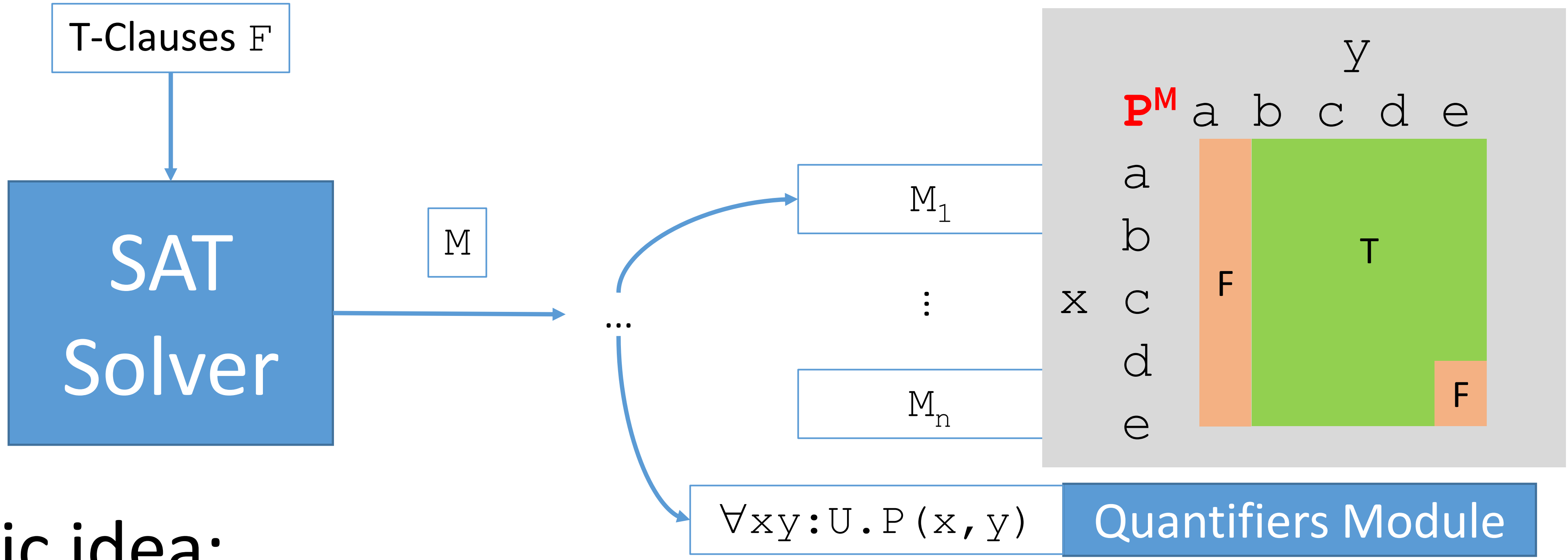
1. Model-Based Quantifier Instantiation

Model-Based Quantifier Instantiation



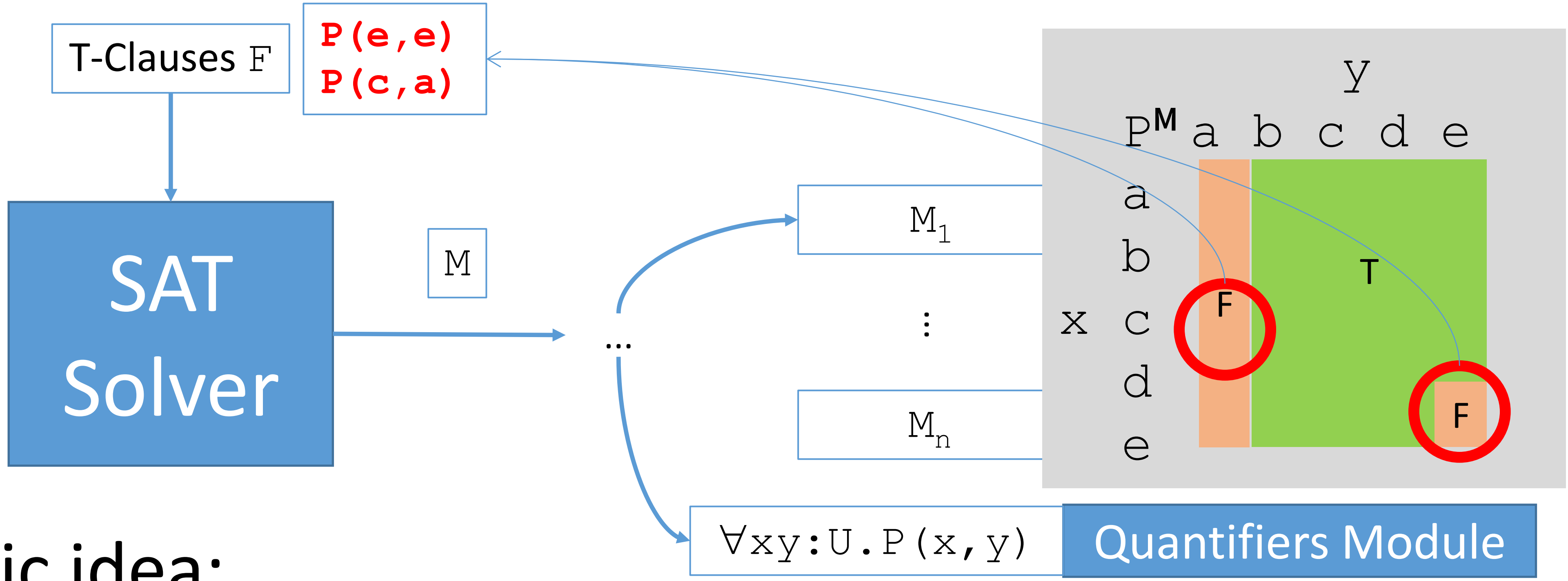
- Basic idea:

Model-Based Quantifier Instantiation



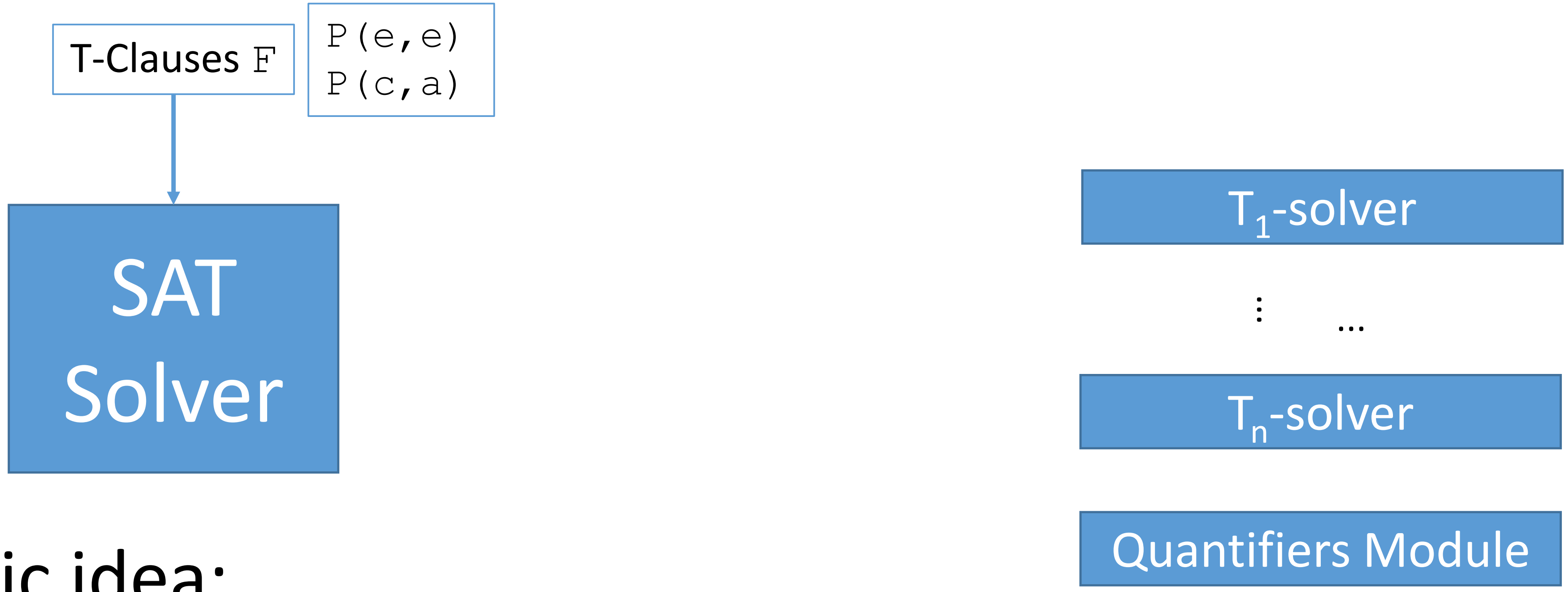
- Basic idea:
  1. Build candidate interpretation **M**, compute $P^M(x,y)$

Model-Based Quantifier Instantiation



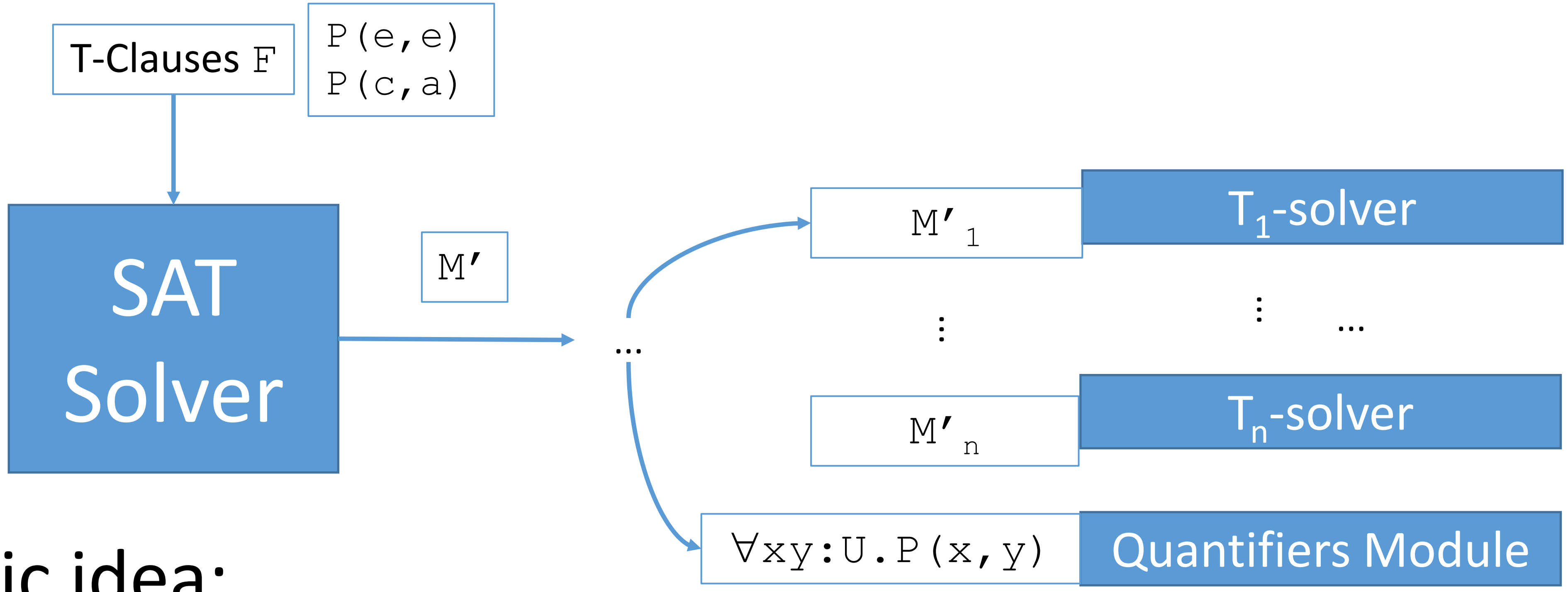
- Basic idea:
  1. Build candidate interpretation **M**, compute $P^M(x,y)$
  2. Add instances (if any) that evaluate to false

Model-Based Quantifier Instantiation



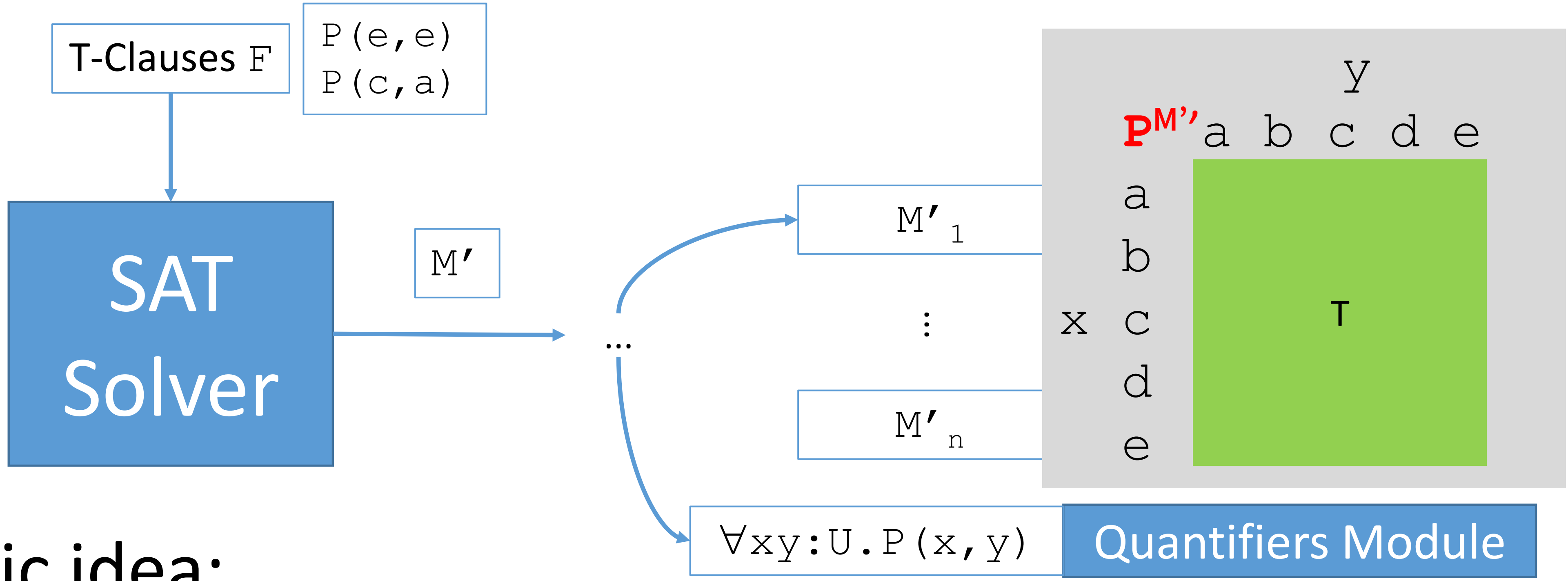
- Basic idea:
  - …and repeat

Model-Based Quantifier Instantiation



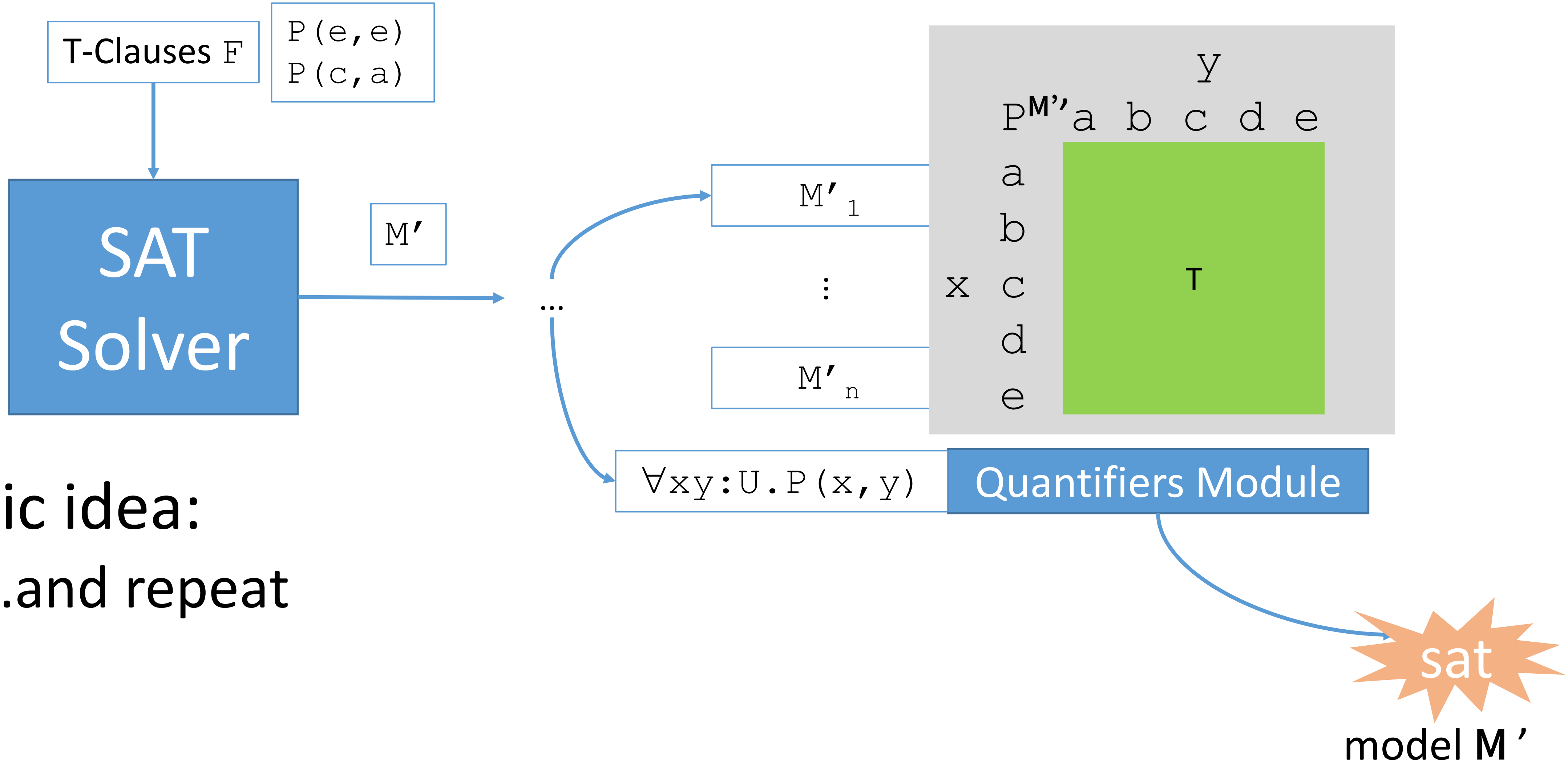
- Basic idea:
  - …and repeat

Model-Based Quantifier Instantiation



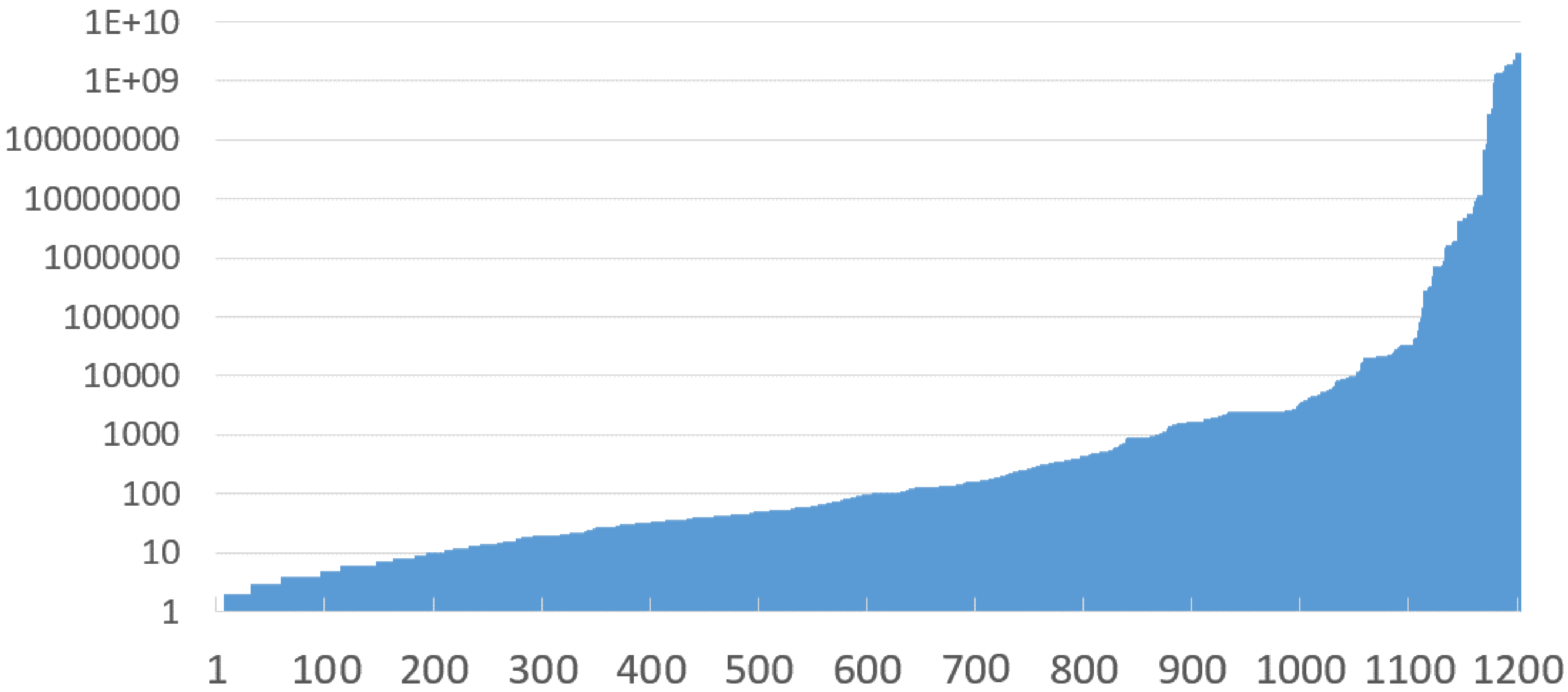
- Basic idea:
  - …and repeat

Model-Based Quantifier Instantiation



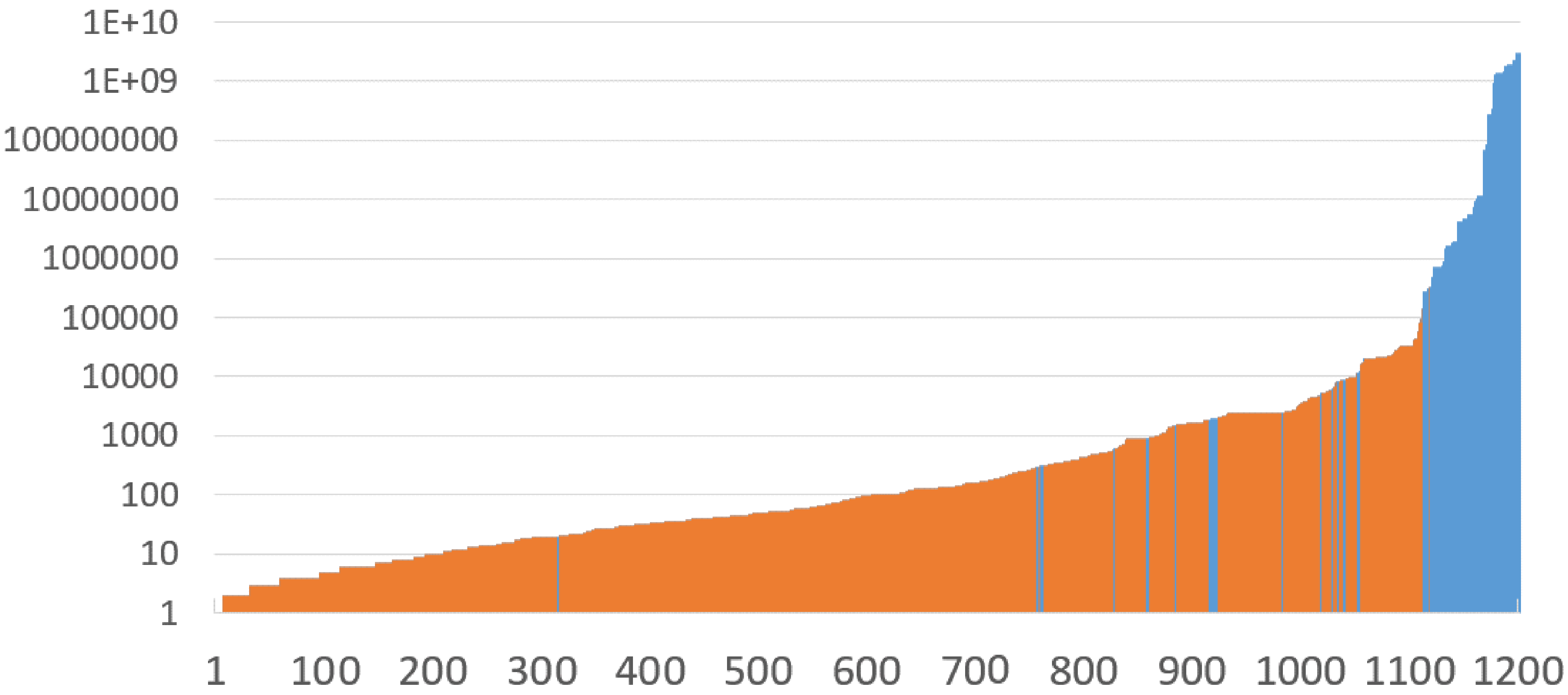
- Basic idea:
  - …and repeat

Model-based Instantiation: Impact



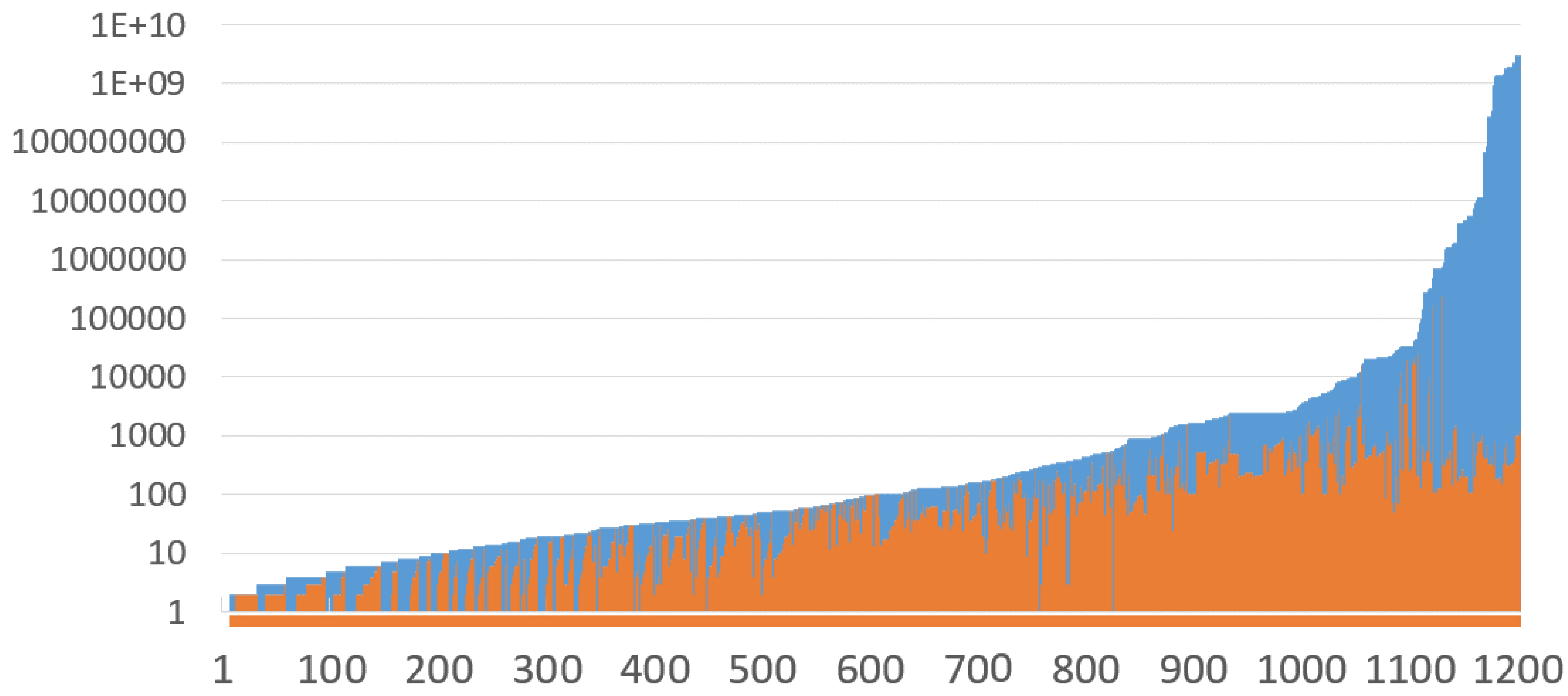
- 1203 satisfiable benchmarks from the TPTP library
  - Graph shows # instances required by exhaustive instantiation
    - E.g. $\forall xyz{:}U.\,P(x,y,z)$, if $|U|=4$, requires $4^3=64$ instances

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

2. Minimizing Model Sizes

Minimizing Model Sizes

- Finding small models is important
  (leads to exponentially fewer possible instances of $\forall$)

To establish T-satisfiability of:

$$\mathtt{G} \wedge \forall \mathtt{x:U.P(x)}$$

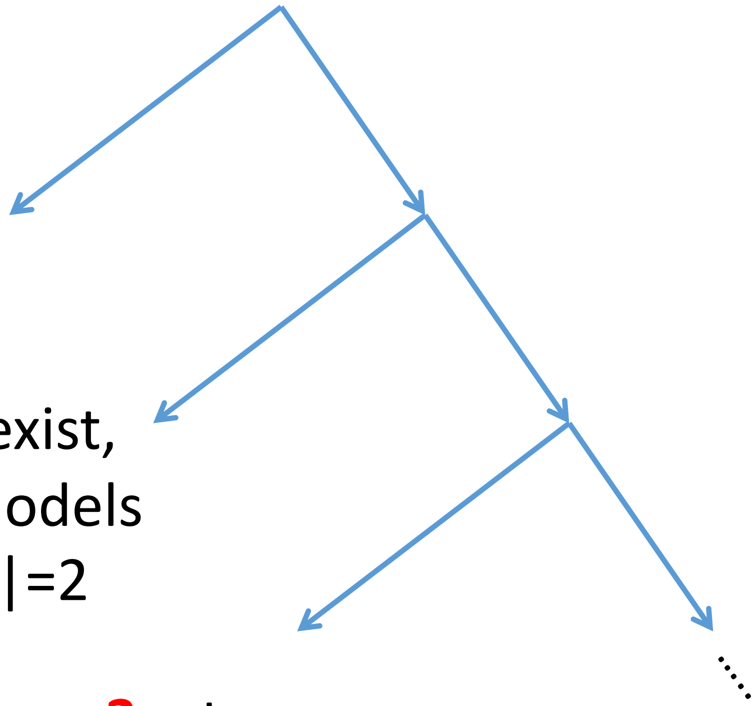
…where $\mathtt{G}$ is a set of ground constraints, and $\mathtt{U}$ is an uninterpreted sort

First, find a model **M** of $\mathtt{G}$ such that $|\mathtt{U}^{\mathbf{M}}|$ is minimized

- To minimize $|\mathtt{U}^{\mathbf{M}}|$:
  - Modifications to the DPLL search procedure in the SAT solver
  - Additional theory solver for cardinality constraints

Minimizing Model Sizes

- Abstractly, organize DPLL search by fixing the cardinality of U



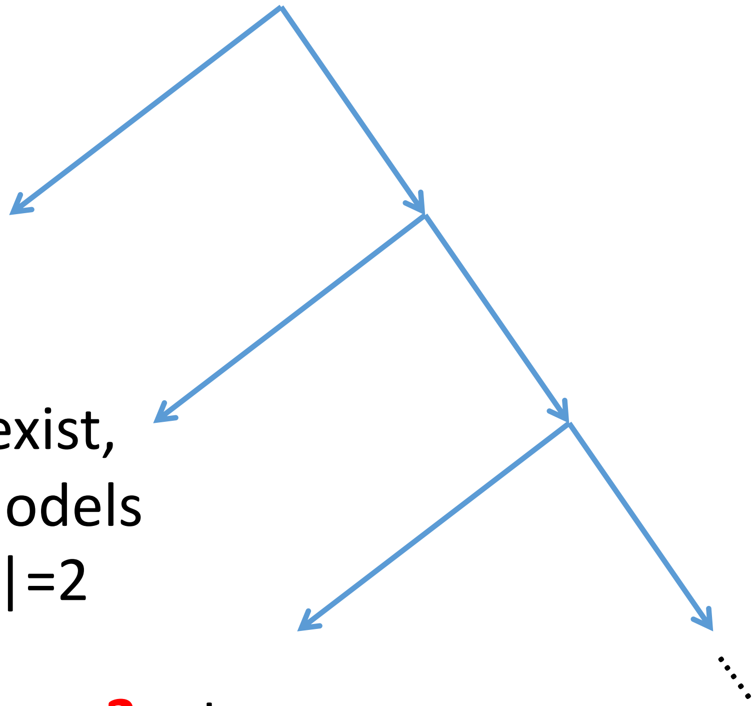
1. Search for models where |U|=1

2. If none exist, search for models where |U|=2

3. etc.

Minimizing Model Sizes

- Abstractly, organize DPLL search by fixing the cardinality of $\mathbb{U}$



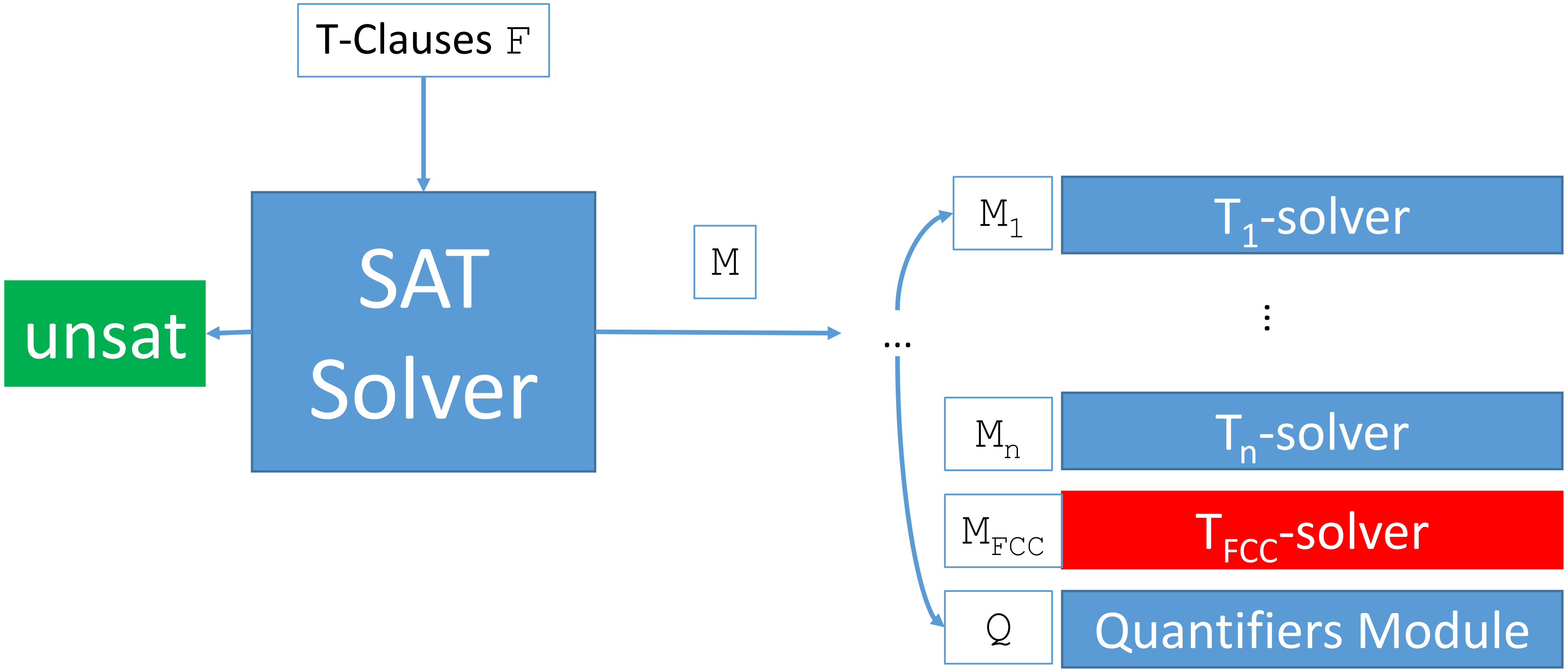
1. Search for models where $|\mathbb{U}|=1$

2. If none exist, search for models where $|\mathbb{U}|=2$

3. etc.

$\Rightarrow$ Extend the SMT solver with a theory solver for cardinality constraints

Theory of finite cardinality constraints



- Theory solver for T_{FCC}
  - FCC = finite cardinality constraints

Theory of finite cardinality constraints

- Theory of finite cardinality constraints T_{FCC}
  - Signature Σ_{FCC}:
    - Predicates $|\mathtt{U}|\leq\mathtt{k}$ for each uninterpreted sort $\mathtt{U}$ and positive numeral $\mathtt{k}$

Theory of finite cardinality constraints

- Theory of finite cardinality constraints T_{FCC}
  - Signature Σ_{FCC}:
    - Predicates $|U| \leq k$ for each uninterpreted sort U and positive numeral k

- Examples:

  $a, b, c : U$

  - $a \neq b \wedge |U| \leq 1$ … T_{FCC}-unsatisfiable
  - $a \neq b \wedge a \neq c \wedge |U| \leq 2$ … T_{FCC}-satisfiable (where $b^M = c^M$)

Theory of finite cardinality constraints

- Decision procedure for T_{FCC}:
  - Given input $\mathtt{G}$

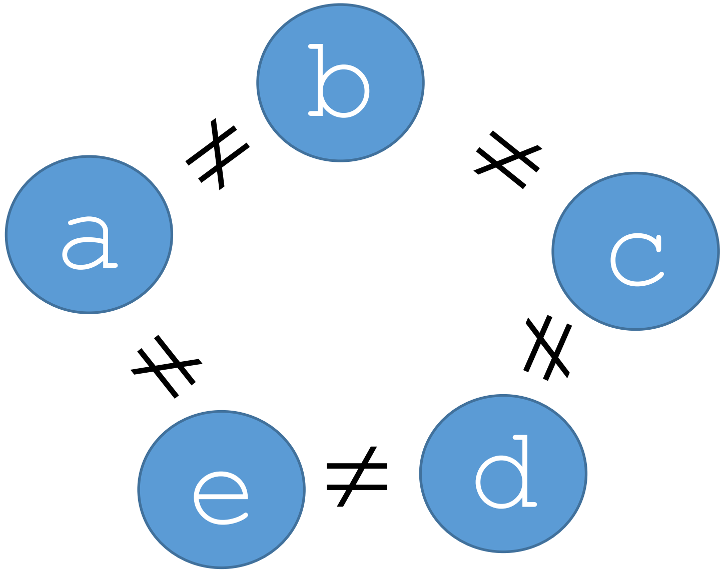
    …where $\mathtt{G}$ is a set of equalities and disequalities
  - Consider the disequality graph (V,E) induced by $\mathtt{G}$:
    - Vertices V are equivalence classes
    - Edges E are disequalities

Theory of finite cardinality constraints

- Decision procedure for T_{FCC}:
  - Given input G

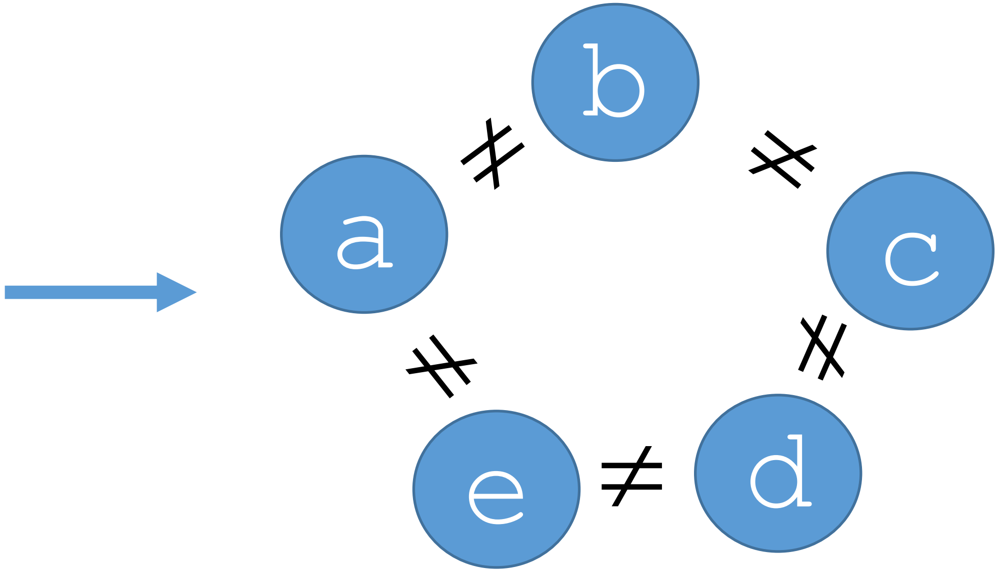
    …where G is a set of equalities and disequalities
  - Consider the disequality graph (V,E) induced by G:
    - Vertices V are equivalence classes
    - Edges E are disequalities

$a \neq b,\ b \neq c,\ c \neq d,\ d \neq e,\ e \neq a$
$|U| \leq 3$



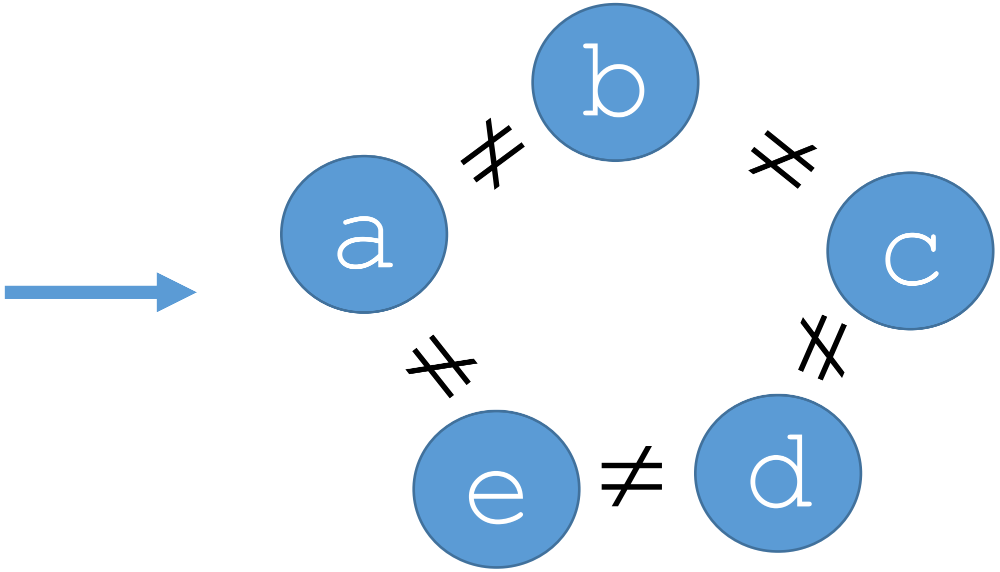
Theory of finite cardinality constraints

a≠b, b≠c, c≠d, d≠e, e≠a
|U|≤3



Theory of finite cardinality constraints

$$a \neq b,\ b \neq c,\ c \neq d,\ d \neq e,\ e \neq a$$
$$|U| \leq 3$$



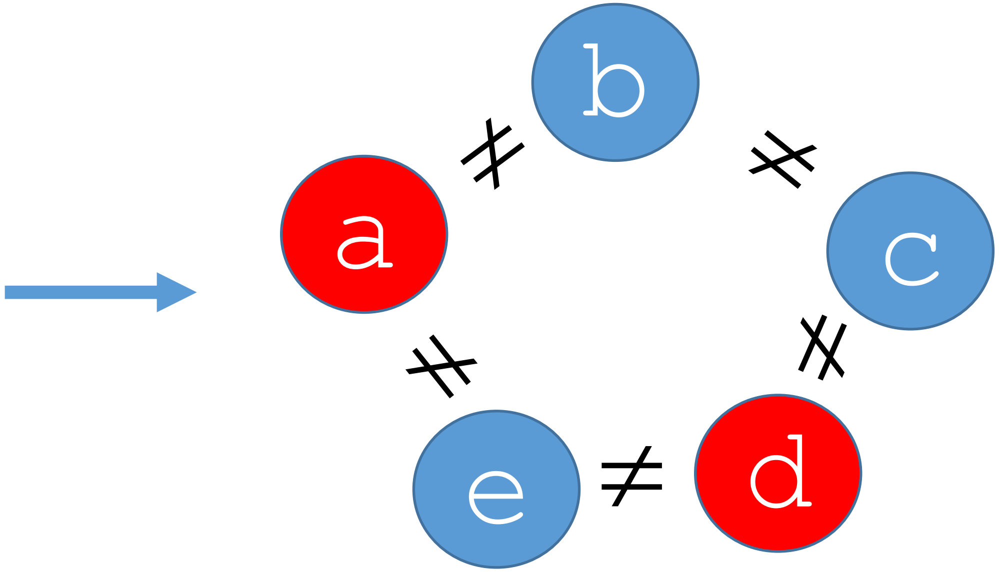
- Decision procedure for T_{FCC} :

Let k be the smallest k such that $|U| \leq k$

- If there is a $(k+1)$-clique, answer “unsat”
- If there are k or fewer vertices, answer “sat”
- Otherwise, split the problem: $t_1 = t_2 \vee t_1 \neq t_2$ for some vertices t_1, t_2

Theory of finite cardinality constraints

a≠b, b≠c, c≠d, d≠e, e≠a
|U|≤3



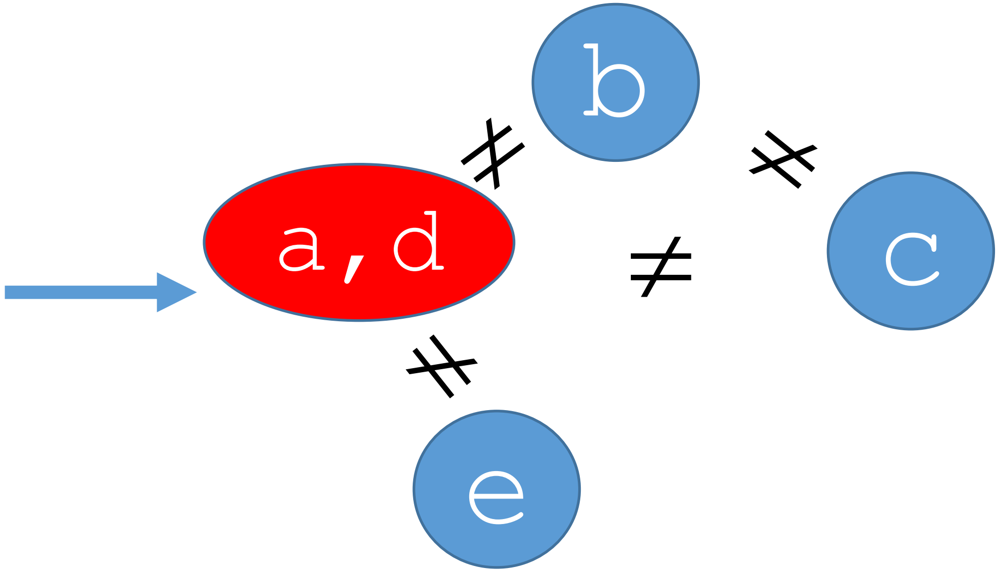
Split: a=d ∨ a≠d

Theory of finite cardinality constraints

$a \neq b,\ b \neq c,\ c \neq d,\ d \neq e,\ e \neq a$

$\mathbf{a=d}$

$|U| \leq 3$



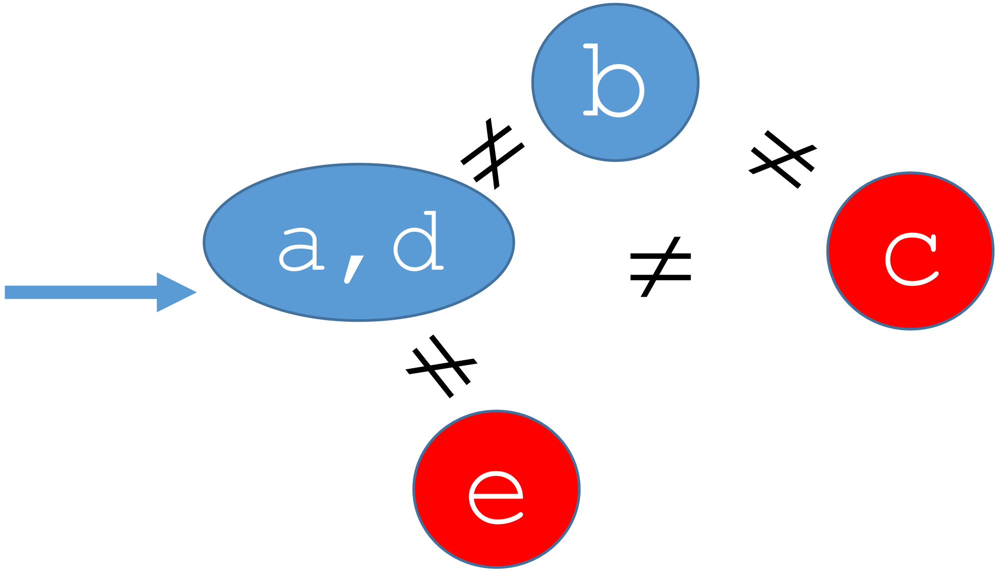
Split: $\underline{\mathbf{a=d}}\ \vee\ a \neq d$

Theory of finite cardinality constraints

$a \neq b,\ b \neq c,\ c \neq d,\ d \neq e,\ e \neq a$

$a = d$

$|U| \leq 3$

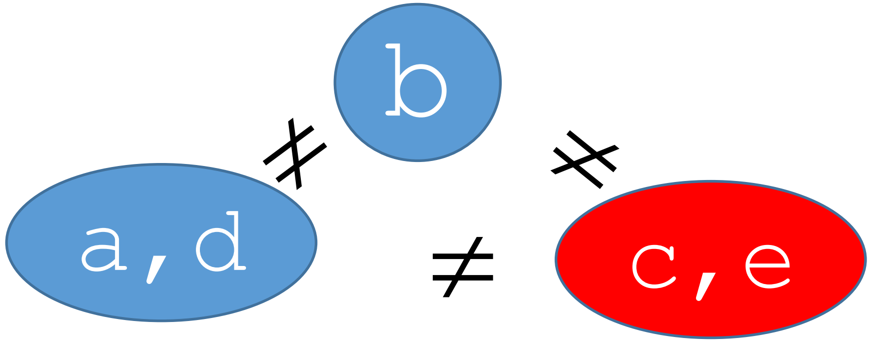


Split: $\underline{a = d} \ \vee\ a \neq d$

Split: $e = c \ \vee\ e \neq c$

Theory of finite cardinality constraints

$a \neq b,\ b \neq c,\ c \neq d,\ d \neq e,\ e \neq a$
$a = d,$ **e=c**
$|U| \leq 3$

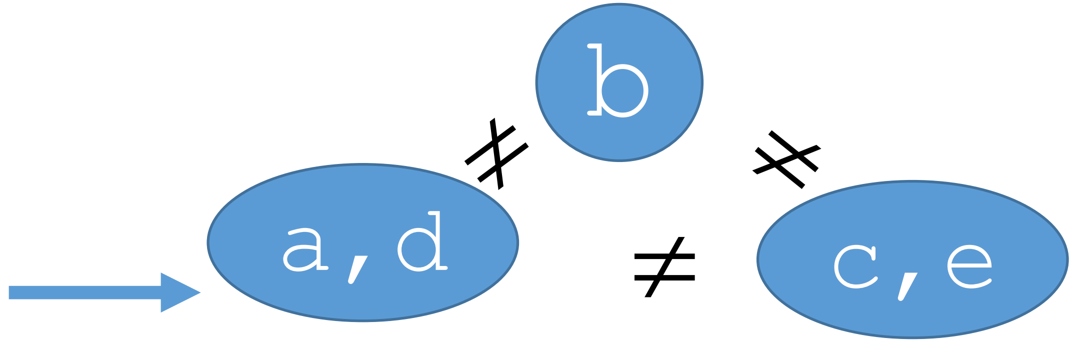


Split: $\underline{a=d} \vee a \neq d$

Split: $\underline{e=c} \vee e \neq c$

Theory of finite cardinality constraints

$a{\neq}b,\ b{\neq}c,\ c{\neq}d,\ d{\neq}e,\ e{\neq}a$
$a{=}d,\ e{=}c$
$|U|\leq 3$



Split: $\underline{a{=}d} \ \vee \ a{\neq}d$
Split: $\underline{e{=}c} \ \vee \ e{\neq}c$
3 equivalence classes … answer "sat"

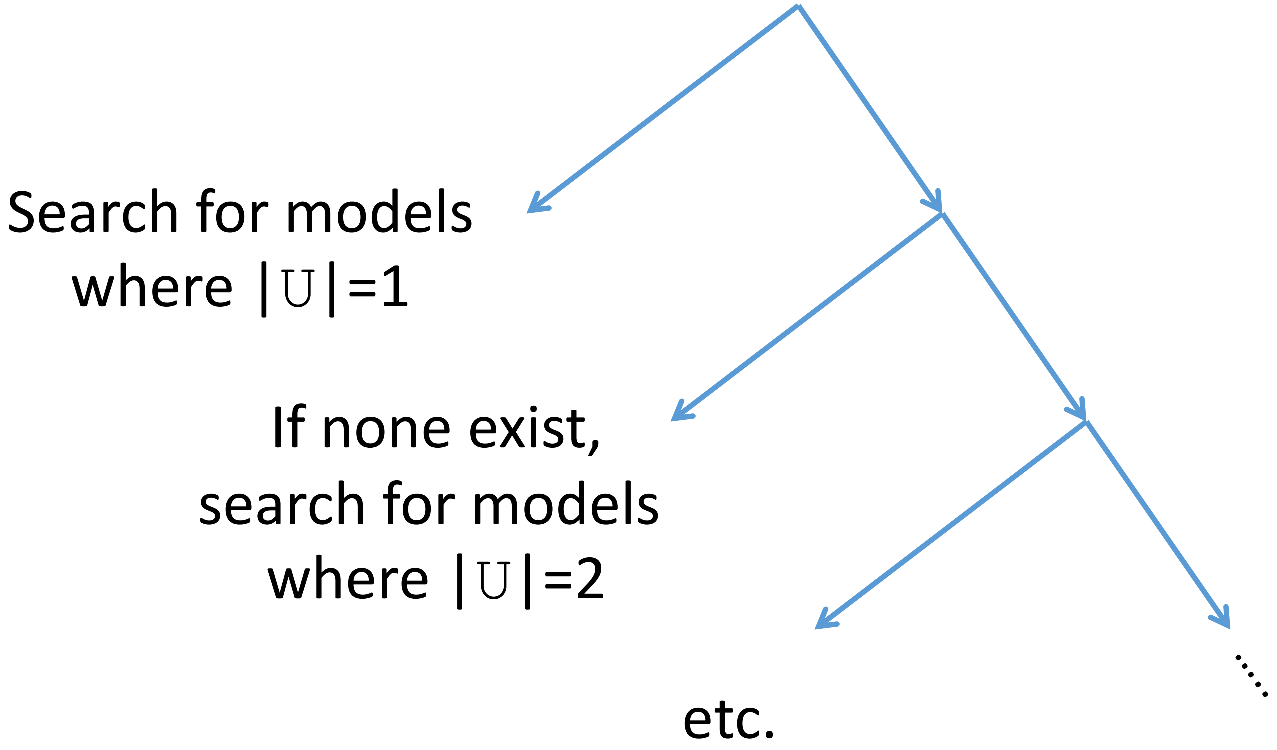
Theory of finite cardinality constraints

- Decision procedure for T_{FCC}
  - Sound, complete and terminating for T_{FCC}-satisfiability
  - Fully integrated into DPLL(T) framework
    - Incremental, generates conflict clauses

- Incorporates optimizations: **[Reynolds et al CAV13]**
  - Finds k-cliques (an NP-hard problem) via a fast incomplete check
  - Heuristics for which vertices to split

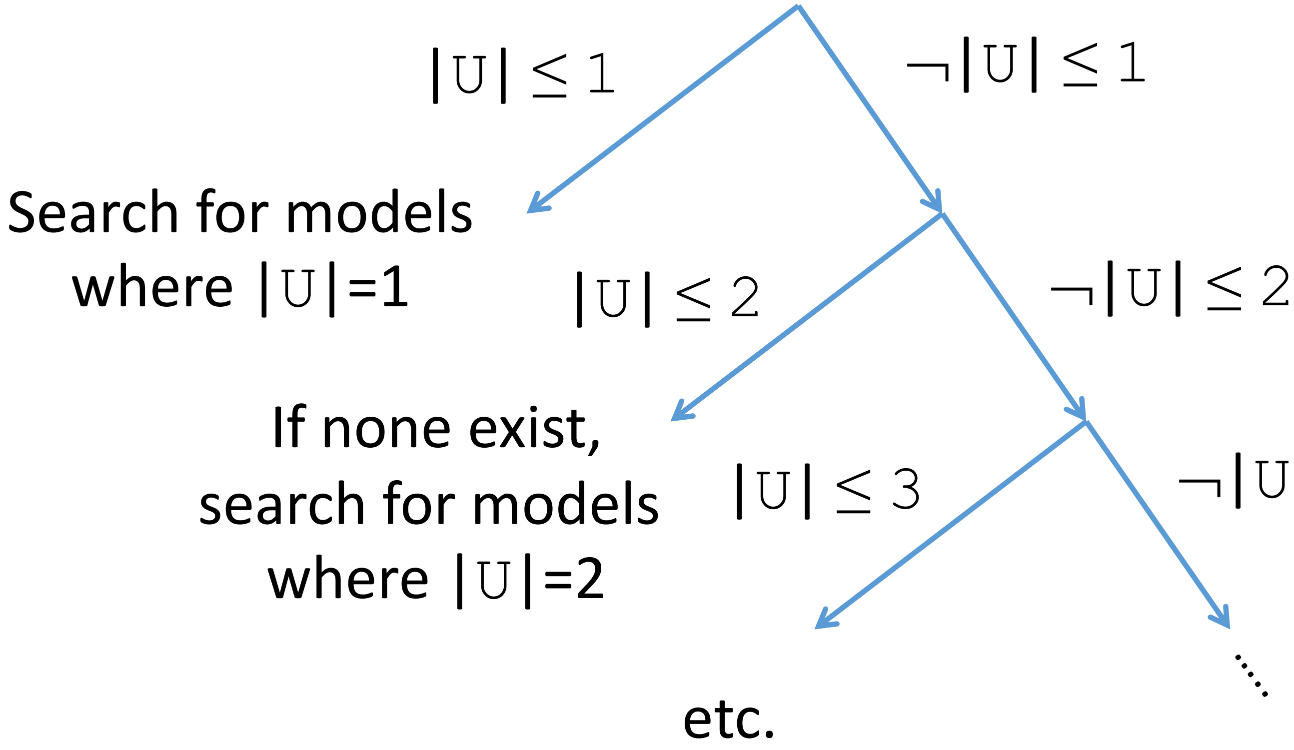
Minimizing Model Sizes with T_{FCC}

- Theory solver for T_{FCC} can be used in part for finding minimal models



Minimizing Model Sizes with T_{FCC}

- Theory solver for T_{FCC} can be used in part for finding minimal models
  - Introduce incremental bounds on cardinality in DPLL search



$\Rightarrow$ DPLL solver chooses tightest bound as first decision

Finding Minimal Counterexamples for ITP

```
List := cons( head : Int, list : Tail ) | nil
L: "subterm-closed structure" of List
```

Signature

```
∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x,y)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)
…
```

Axioms

∃xy:L.rev(append(x,y))≠append(rev(y),rev(x))

(Negated) conjecture

CVC4

Finding Minimal Counterexamples for ITP

```
List := cons( head : Int, list : Tail ) | nil
L: “subterm-closed structure” of List
```

Signature

```
∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x,y)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)
…
```

Axioms

```
∃xy:L.rev(append(x,y))≠append(rev(y),rev(x))
```

(Negated) conjecture

CVC4

UNSAT

```
∀xy:L.rev(append(x,y))=append(rev(y),rev(x))
```

holds

Finding Minimal Counterexamples for ITP

```
List := cons( head : Int, list : Tail ) | nil
L: "subterm-closed structure" of List
```

Signature

```
∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x,y)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)
…
```

Axioms

∃xy:L.rev(append(x,y))≠append(rev(x),rev(y))

(Negated) conjecture

CVC4

Finding Minimal Counterexamples for ITP

```
List := cons( head : Int, list : Tail ) | nil
L: "subterm-closed structure" of List
```

Signature

```
∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x,y)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)
…
```

Axioms

∃xy:L.rev(append(x,y))≠append(rev(x),rev(y))

(Negated) conjecture

CVC4

Counterexample M:

M (x)=cons(0,nil)

M (y)=cons(1,nil)

```
rev(append(cons(0,nil),cons(1,nil)))=
      cons(1,cons(0,nil))≠
      cons(0,cons(1,nil))=
      append(rev(x),rev(y))
```

Finding Minimal Counterexamples: Challenge

```
Tree := node( left : Tree, data : Int, right : Tree ) | leaf
T: "subterm-closed structure" of Tree
```

```
∀x:T.depth(x)=ite(is-node(x),1+max(depth(left(x)),depth(right(x))),0)
```

```
∃k:T.depth(k)≥4
```

CVC4

- Find a tree with depth at least 4

Finding Minimal Counterexamples: Challenge

```
Tree := node( left : Tree, data : Int, right : Tree ) | leaf
T: "subterm-closed structure" of Tree
```

```
∀x:T.depth(x)=ite(is-node(x),1+max(depth(left(x)),depth(right(x))),0)
```

$\exists \mathbf{k}:T.\mathrm{depth}(\mathbf{k})\geq 4$

- Find a tree with depth at least 4

Consider all k of **depth 0**

Consider all k of **depth 1**

Consider all k of **depth 2**

Consider all k of **depth 3**

…

Combinatorial explosion
$\Rightarrow$solver is **slow!**

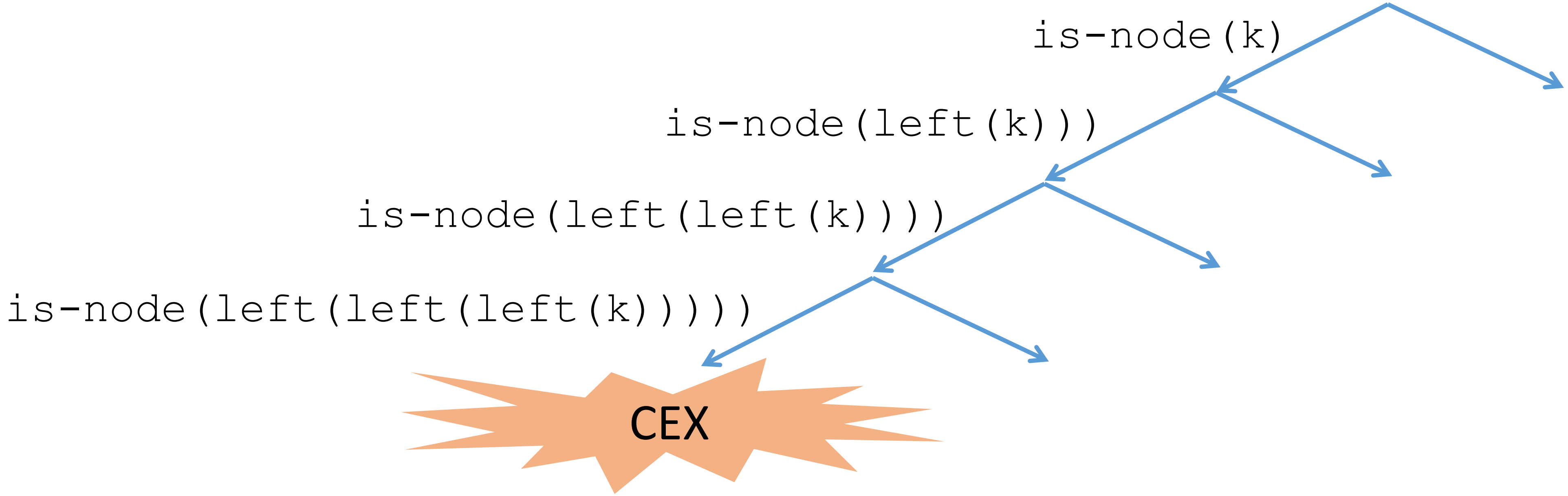
Finding ~~Minimal~~ Counterexamples: Challenge

```
Tree := node( left : Tree, data : Int, right : Tree ) | leaf
T: "subterm-closed structure" of Tree
```

```
∀x:T.depth(x)=ite(is-node(x),1=max(depth(left(x)),depth(right(x))),0)
```

```
∃k:T.depth(k)≥4
```

- **Find a tree with depth at least** 4



Finding (Non-Minimal) CEX: Challenge

```
List := cons( head : Int, list : Tail ) | nil
L: "subterm-closed structure" of List
```

```
∀x:L.all-pos(x)=ite(is-cons(x),head(x)>0∧all-pos(tail(x)),true)
```

```
∃k:L.is-cons(k) ∧ all-pos(k)
```

- Find a non-empty list of positive integers

CVC4

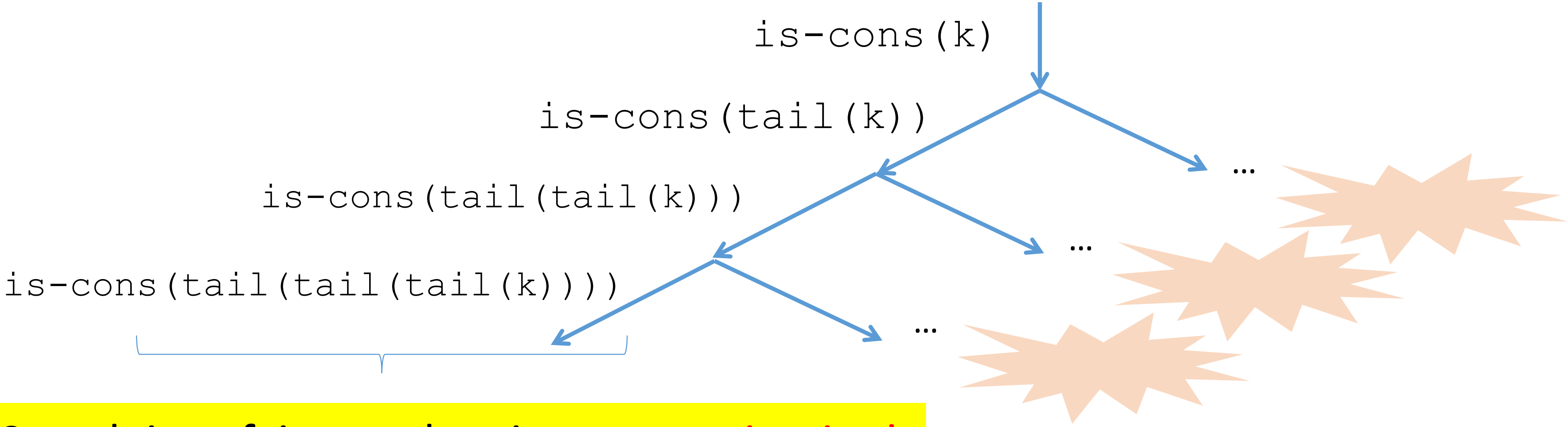
Finding (Non-Minimal) CEX: Challenge

```
List := cons( head : Int, list : Tail ) | nil
L: "subterm-closed structure" of List
```

```
∀x:L.all-pos(x)=ite(is-cons(x),head(x)>0∧all-pos(tail(x)),true)
```

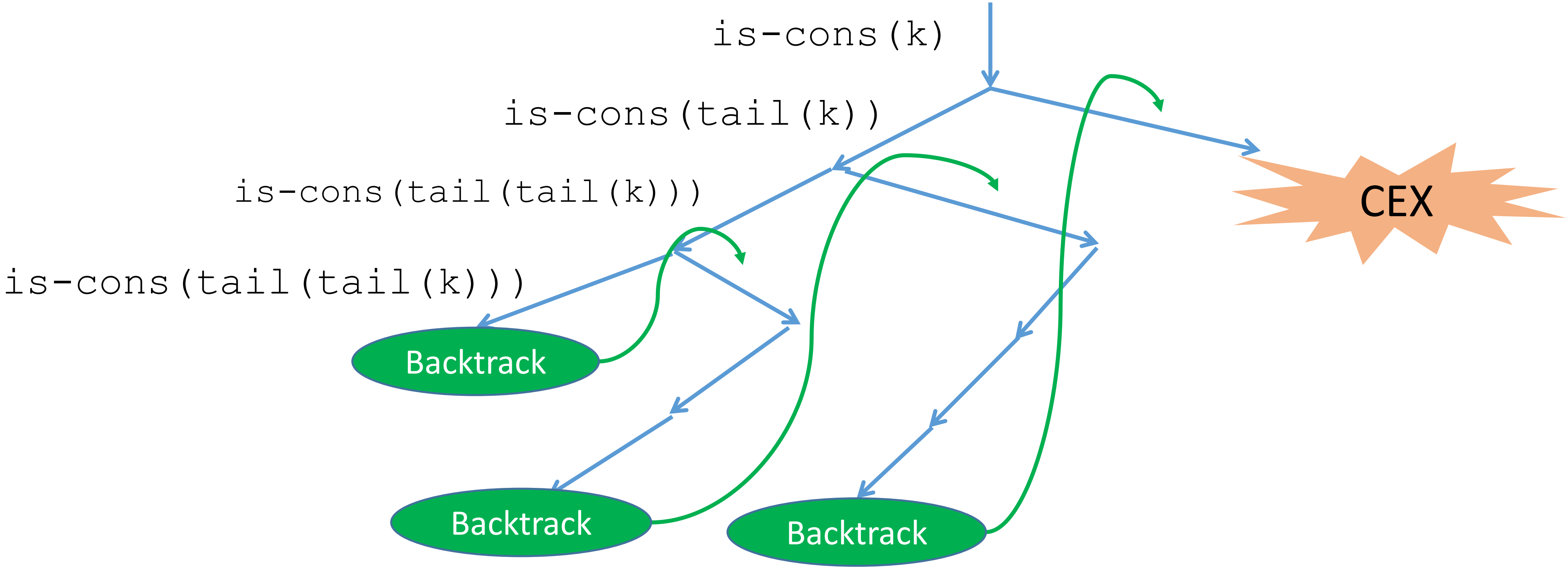
```
∃k:L.is-cons(k) ∧ all-pos(k)
```

- Find a non-empty list of positive integers



Search is unfair ⇒ solver is non-terminating!

Branch and Bound: Hybrid Approach?



- Guide search so that eventually it will consider small models
  - $\Rightarrow$ In development

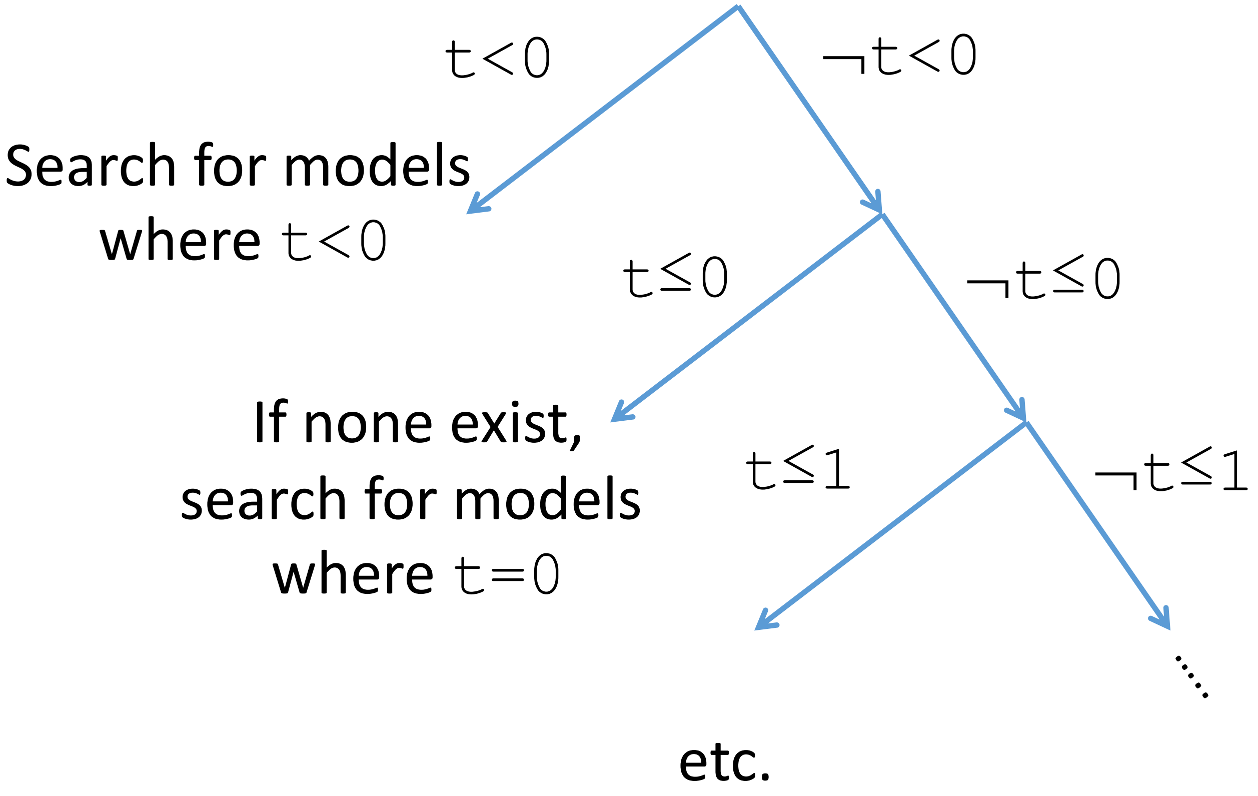
Branch and Bound: Use Cases

- Similar approach can be used for:
  1. $\forall$ bounded by symbolic numeric (integer) range
  2. $\forall$ bounded by set membership
  3. Model finding for theory of strings + length
  4. Syntax-Guided Synthesis

Use case #1: Bounded Integer ∀

Variant: Bounded Integer ∀

- $\forall x{:}\mathrm{Int}.\ 0 \leq x < t \Rightarrow P(x)$



⇒ Incrementally bound the value of term t

Use case #2: Sets + Cardinality

Theory of Finite Sets + Cardinality

- Parametric theory of finite sets of elements $\mathtt{E}$
- Signature Σ_{Set}:
  - Empty set $\emptyset$, Singleton $\{a\}$
  - Membership $\in : \mathtt{E} \times \mathtt{Set} \rightarrow \mathtt{Bool}$
  - Subset $\subseteq : \mathtt{Set} \times \mathtt{Set} \rightarrow \mathtt{Bool}$
  - Set connectives $\cup, \cap, \setminus : \mathtt{Set} \times \mathtt{Set} \rightarrow \mathtt{Set}$

- Example input: $\boxed{x = y \cap z \wedge a+5 \in x \wedge y \subseteq w}$

- Applications in programming languages, e.g. Alloy

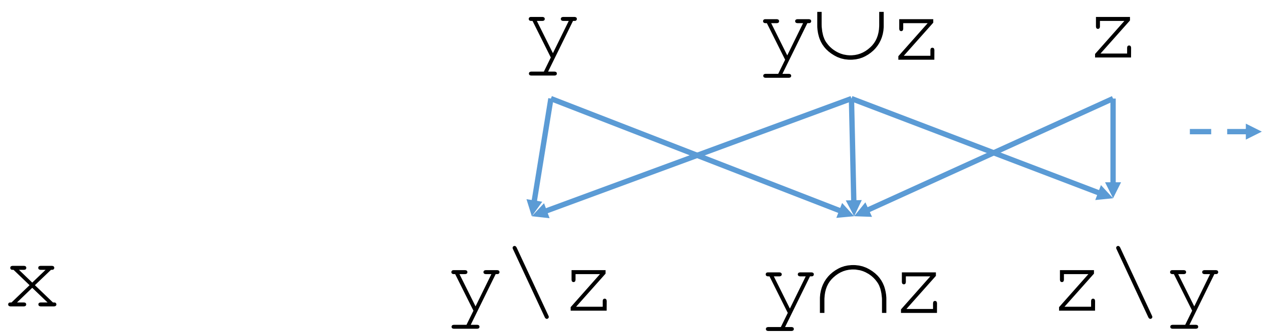
Theory of Finite Sets + Cardinality

- Recently:
  - Extended signature of theory to include:
    - Cardinality $|.|$: $\texttt{Set} \rightarrow \texttt{Int}$
  - Extended decision procedure for cardinality constraints
    - Fully integrated component in DPLL(T) [Bansal et al IJCAR2016]

- Example input: $\boxed{x = y \cup z \wedge |x| = 14 \wedge |y| \geq |z| + 5}$

Theory of Finite Sets + Cardinality

- Decision procedure builds cardinality graph where
  - Cardinality of leaves are disjoint sum of parents

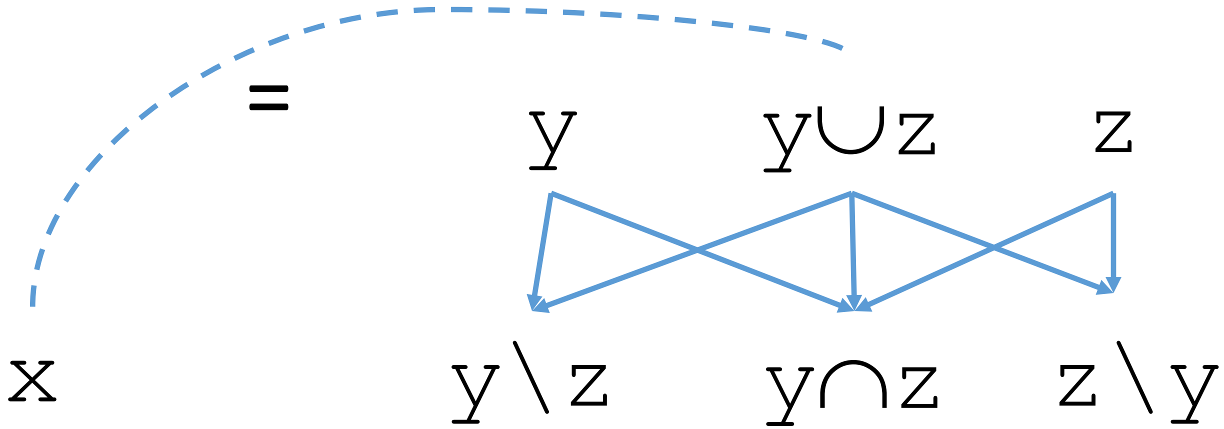


$$|y| = |y \setminus z| + |y \cap z|$$
$$|z| = |z \setminus y| + |y \cap z|$$
$$|y \cup z| = |y \setminus z| + |y \cap z| + |z \setminus y|$$

[Bansal/Reynolds/Barrett/Tinelli IJCAR2016]

Theory of Finite Sets + Cardinality

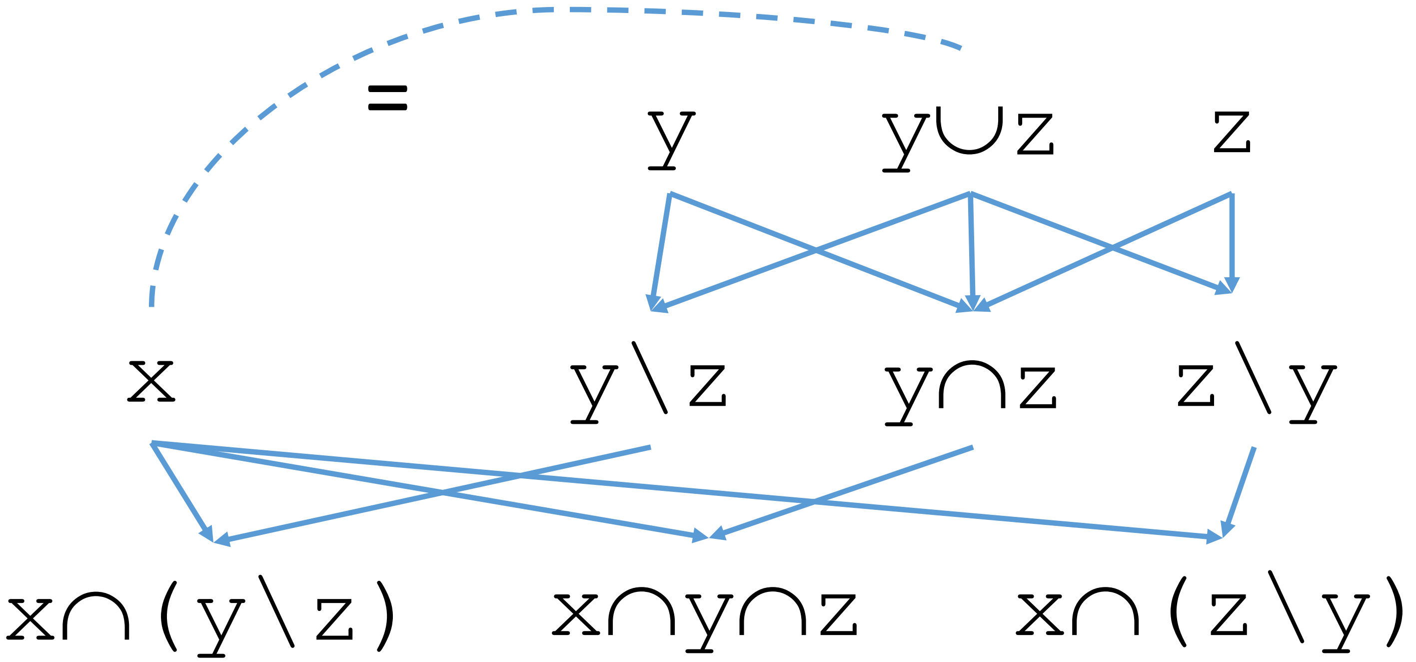
- Decision procedure builds cardinality graph where
  - Cardinality of leaves are disjoint sum of parents
    - Equalities between sets



[Bansal/Reynolds/Barrett/Tinelli IJCAR2016]

Theory of Finite Sets + Cardinality

- Decision procedure builds cardinality graph where
  - Cardinality of leaves are disjoint sum of parents
    - Equalities between sets $\rightarrow$ merge leaves

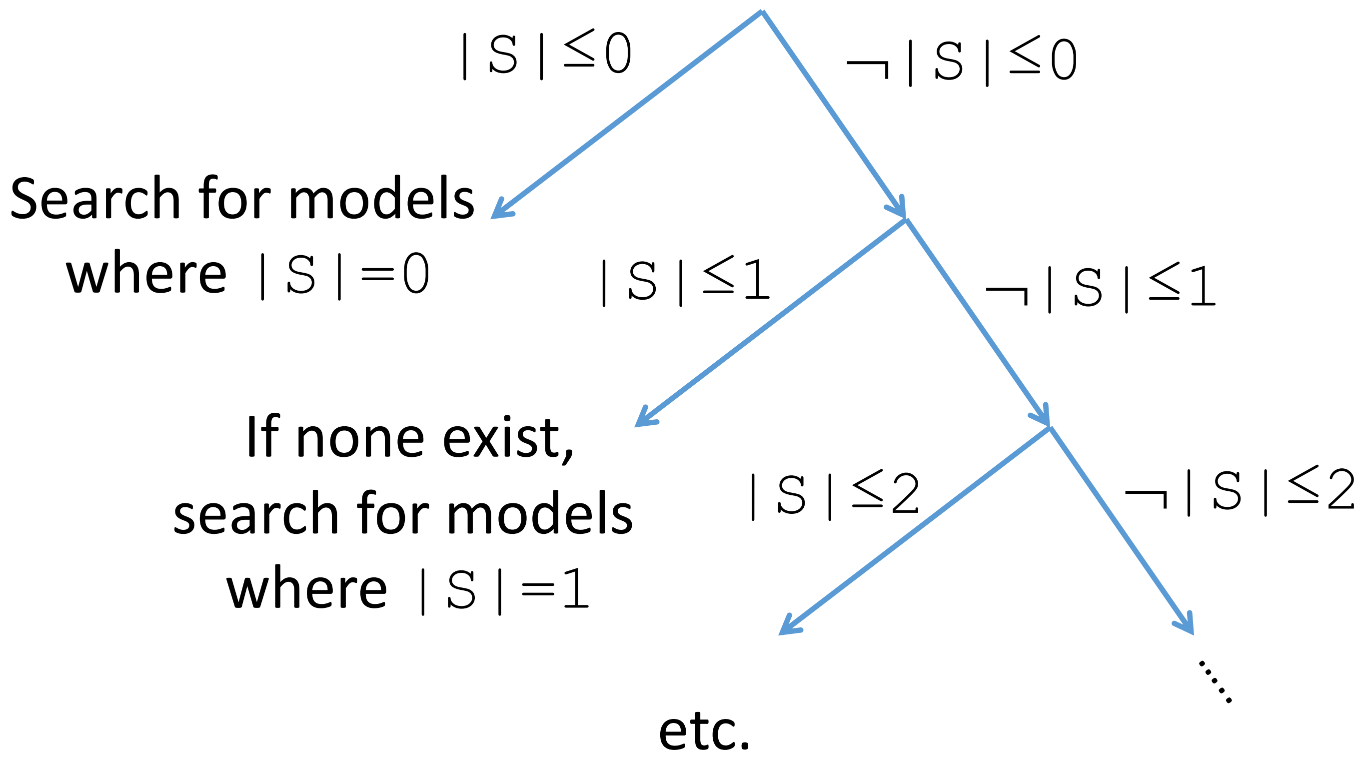


$$x = y \cup z \Rightarrow$$
$$|x| = |x \cap (y \setminus z)| +$$
$$|x \cap y \cap z| + |x \cap (z \setminus y)|$$

[Bansal/Reynolds/Barrett/Tinelli IJCAR2016]

Branch and Bound: Set Membership ∀

- $\forall$`x:Int.x`$\in$`S`$\Rightarrow$`P(x)`



$\Rightarrow$ Make use of native set cardinality operator $|.|:\mathrm{Set}\rightarrow\mathrm{Int}$

Set Membership $\forall$

- Increased power to encode:

$\forall x. x \in S \Rightarrow P(x) \wedge \lvert S \rvert \geq k$	…	P holds for at least k points
$\forall x. x \in S \Rightarrow x < 10$	…	All elements of S are < 10
$\forall xy. x \in S \wedge y \in T \Rightarrow x < y$	…	All elements of S are $<$ those in T

Use case #3: Theory of Strings

Theory of Strings + Length

- Signature Σ_S :
  - Constants from a fixed finite alphabet e.g. `“a”`, `“ab”`, …
  - String concatenation $\_ \cdot \_$: $\mathtt{Str} \times \mathtt{Str} \rightarrow \mathtt{Str}$
  - Length `len`(_) : $\mathtt{Str} \rightarrow \mathtt{Int}$
  - Extended functions `str.substr`, `str.contains`, `str.to.int`, `int.to.str`, `str.replace`, `str.indexof`
- Example input:

$$\mathtt{len(x)>len(y)} \ \wedge \ \mathtt{str.contains(y,"ab")}$$

Theory of Strings + Length : Models

```
char buff[15];
char pass;
cout << "Enter the password :";
gets(buff);
if (regex_match(buff, std::regex("([A-Z]+)") )) {
   if(strcmp(buff, "PASSWORD")) {
      cout << "Wrong Password";
   } else {
      cout << "Correct Password";
      pass = 'Y';
   }
   if(pass == 'Y') {
      /* Grant the root permission*/
   }
}
```

Encode

```
(set-logic QF_S)
(declare-const input String)
(declare-const buff String)
(declare-const pass0 String)
(declare-const rest String)
(declare-const pass1 String)
(assert (= (str.len buff) 15))
(assert (= (str.len pass1) 1))
(assert (or (< (str.len input) 15)
   (= input (str.++ buff pass0 rest)))
(assert (str.in.re buff
         (re.+ (re.range "A" "Z"))))
(assert (ite (= buff "PASSWORD")
             (= pass1 "Y")
             (= pass1 pass0)))
(assert (not (= buff "PASSWORD")))
(assert (= pass1 "Y"))
```

Solve

Extract

```
tiliang@milner:~/workspace/security/benchmarks/homemade$ ~/CVC4/bin/pt-cvc4 propsalex.smt2
sat
(model
(define-fun input () String "AAAAAAAAAAAAAAAY")
(define-fun buff () String "AAAAAAAAAAAAAAA")
(define-fun pass0 () String "Y")
(define-fun rest () String "")
(define-fun pass1 () String "Y")
)
```

- Models may correspond to security vulnerabilities

Theory of Strings + Length

- Theoretical complexity of:
  - Word equation problem is in PSPACE
  - …with length constraints is OPEN
  - …with extended functions is UNDECIDABLE
- Instead, focus on:
  - Solver that is efficient in practice
    - Often, for applications like symbolic execution, able to find models

Theory of Strings + Length

…

$$\text{F-Unify}\ \frac{\mathsf{F}\, s = (w, u, u_1) \quad \mathsf{F}\, t = (w, v, v_1) \quad s \approx t \in \mathcal{C}(\mathsf{S}) \quad \mathsf{S} \models \mathsf{len}\, u \approx \mathsf{len}\, v}{\mathsf{S} := \mathsf{S}, u \approx v}$$

$$\text{F-Split}\ \frac{\mathsf{F}\, s = (w, u, u_1) \quad \mathsf{F}\, t = (w, v, v_1) \quad s \approx t \in \mathcal{C}(\mathsf{S}) \quad \mathsf{S} \models \mathsf{len}\, u \not\approx \mathsf{len}\, v \quad u \notin \mathcal{V}(v_1) \quad v \notin \mathcal{V}(u_1)}{\mathsf{S} := \mathsf{S}, u \approx \mathsf{con}(v, z) \quad \| \quad \mathsf{S} := \mathsf{S}, v \approx \mathsf{con}(u, z)}$$

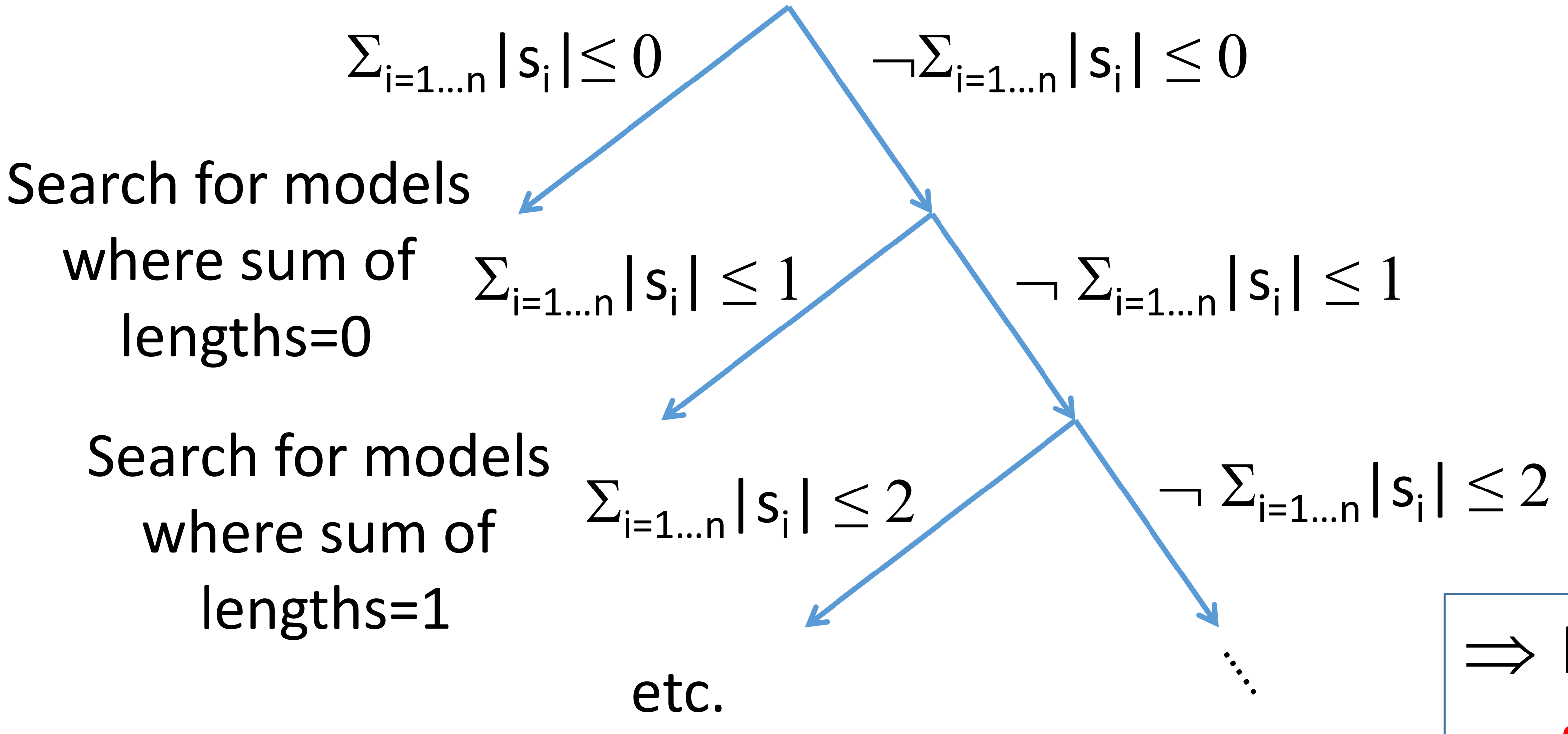
$$\text{F-Loop}\ \frac{\mathsf{F}\, s = (w, x, u_1) \quad \mathsf{F}\, t = (w, v, v_1, x, v_2) \quad s \approx t \in \mathcal{C}(\mathsf{S}) \quad x \notin \mathcal{V}((v, v_1))}{\begin{array}{c}\mathsf{S} := \mathsf{S},\ x \approx \mathsf{con}(z_2, z),\ \mathsf{con}(v, v_1) \approx \mathsf{con}(z_2, z_1),\ \mathsf{con}(u_1) \approx \mathsf{con}(z_1, z_2, v_2) \\ \mathsf{R} := \mathsf{R}, z\ \mathsf{in}\ \mathsf{star}(\mathsf{set}\ \mathsf{con}(z_1, z_2)) \quad \mathsf{C} := \mathsf{C}, t\end{array}}$$

…

- Rule-based algebraic calculus **[Liang et al 2014]**:
  - Handled unbounded strings
    - E.g. HAMPI **[Kiezun et al 2009]** reduces to fixed-width Bit Vectors
  - Refutation-**sound** and model-**sound**, e.g. “unsat” and “sat” can be trusted
  - Refutation-**incomplete**, not guaranteed to terminate for “unsat”
  - Finite-model **complete**
    - …assuming a branch and bound strategy

Branch and Bound: Theory of Strings + Length

- Given input $\texttt{F}[s_1, \ldots, s_n]$ for strings $s_1 \ldots s_n$:



$\Rightarrow$ Incrementally bound the sum of lengths of strings

Use case #4: Syntax-Guided Synthesis

Syntax-Guided Synthesis

$$\exists \texttt{f}:\texttt{Prog}.\forall \texttt{i}.\texttt{S(f,i)}$$

- Interested in synthesis conjectures of the above form:

  There exists a program `f`,

  ...such that for all inputs `i`,

  ...a (universal) specification `S(f,i)` holds

Syntax-Guided Synthesis

$$\exists \texttt{f:Prog}.\forall \texttt{i}.\texttt{S(f,i)}$$

- Problem is UNDECIDABLE
  - Involves second-order $\forall$ on `f`, universal $\forall$ on `i`

Syntax-Guided Synthesis

$$\exists \mathtt{f}:\mathtt{Prog}.\forall \mathtt{i}.\mathtt{S}(\mathtt{f},\mathtt{i})$$

```
P = ite(C,P,P)|+(P,P)|-(P,P)|0|1|i
C = ≥(P,P)|=(P,P)|not(C)
```

- Problem is UNDECIDABLE
  - Involves second-order $\forall$ on `f`, universal $\forall$ on `i`
- A way to simplify the problem is to restrict the space of solutions
  - Solutions belong to a grammar `P` specifying syntax for `f`

Syntax-Guided Synthesis

$$\exists f{:}\mathbf{P}.\forall i.\mathbf{S_P}(f,i)$$

```
P = ite(C,P,P)|+(P,P)|-(P,P)|0|1|i
C = ≥(P,P)|=(P,P)|not(C)
```

- Problem is UNDECIDABLE
  - Involves second-order $\forall$ on `f`, universal $\forall$ on `i`
- A way to simplify the problem is to restrict the space of solutions
  - Solutions belong to a grammar `P` specifying syntax for `f`
- Grammar P can be seen in SMT as an inductive datatype
  - Use deep embedding into specification S_P, solve for `f` as `P` **[Reynolds et al CAV15]**

Syntax-Guided Synthesis

$$\exists f{:}P.\forall i.S_P(f,i)$$

$$\boxed{\begin{array}{l} P \;=\; \mathtt{ite}(C,P,P)\,|\,+(P,P)\,|\,-(P,P)\,|\,0\,|\,1\,|\,i \\ C \;=\; \geq(P,P)\,|\,=(P,P)\,|\,\mathtt{not}(C) \end{array}}$$

- Consider solutions (naively) by enumeration:

$f^M = 0$ check $\forall i.S_P(0,i)$

$f^M = 1$ check $\forall i.S_P(1,i)$

$f^M = \ldots$ …

$f^M = 1+1$ check $\forall i.S_P(1+1,i)$

$f^M = i+1$ check $\forall i.S_P(i+1,i)$

$f^M = \ldots$ …

$f^M = \mathtt{ite}(\geq(i,0),i,0)$ check $\forall i.S_P(\mathtt{ite}(\geq(i,0),i,0),i)$

- In practice, guided via CE-guided inductive synthesis loop **[Solar-Lezama 2013]**

Syntax-Guided Synthesis

$$\exists f{:}P.\forall i.S_P(f,i)$$

$$\boxed{\begin{array}{l} P \; = \; \mathtt{ite}(C,P,P)\,|+(P,P)\,|-(P,P)\,|\,0\,|\,1\,|\,i \\ C \; = \; \geq(P,P)\,|=(P,P)\,|\,\mathtt{not}(C) \end{array}}$$

- Consider solutions (naively) by enumeration:

$f^M = 0$	check $\forall i.S_P(0,i)$
$f^M = 1$	check $\forall i.S_P(1,i)$
$f^M = \ldots$	…
$f^M = 1+1$	check $\forall i.S_P(1+1,i)$
$f^M = i+1$	check $\forall i.S_P(i+1,i)$
$f^M = \ldots$	…
$f^M = \mathtt{ite}(\geq(i,0),i,0)$	check $\forall i.S_P(\mathtt{ite}(\geq(i,0),i,0),i)$

- In practice, guided via CE-guided inductive synthesis loop **[Solar-Lezama 2013]**

  $\Rightarrow$ Finite-model completeness if we consider smaller solutions before larger ones

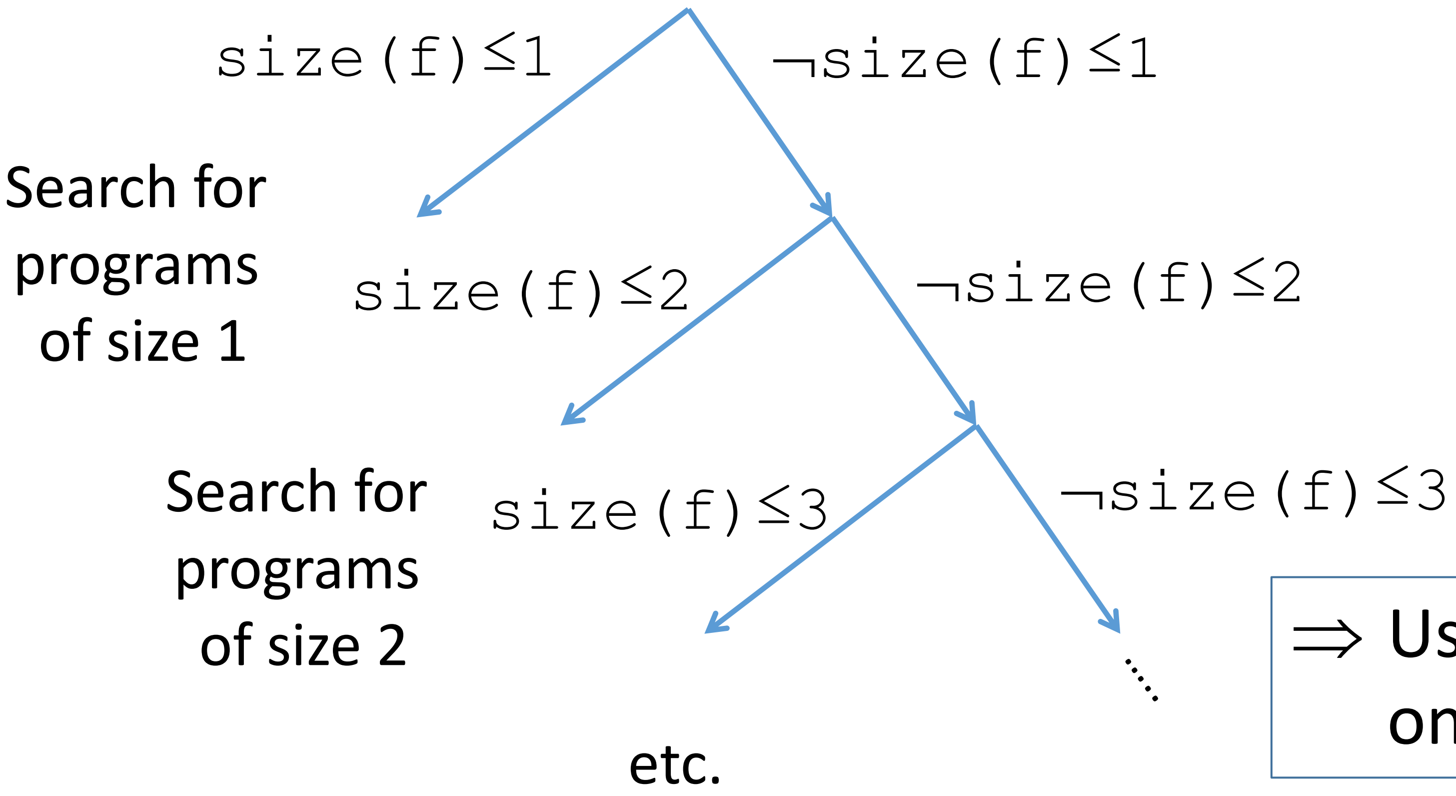
Syntax-Guided Synthesis

- To enumerate smaller solutions before larger ones:
  - Introduce notion of term size of datatype (# constructor applications), e.g.:
    - `size(i)=1`
    - `size(i+1)=3`
    - `size(ite(i≥0,i,i+1))=8`
- Extend theory of datatypes with size bound predicates:
  - `size(t)≤k`

    …where `t` is a datatype term and numeral `k`
  - Decision procedure extends to predicates of this form

Branch and Bound: Syntax-Guided Synthesis

- `∃f:P.∀i.S(f,i)`



$\Rightarrow$ Use size bound predicate on inductive datatypes

Each of these variants:

- Modify DPLL search
  - …to minimize some (numeric) quantity:
    - Finite model finding: cardinality of sorts
    - Bounded integer $\forall$: value of numeric bounds
    - Bounded set membership: cardinality of sets
    - Strings: sum of lengths
    - Syntax-guided synthesis: term size
- Have similar challenges/tradeoffs for strategies:
  - Minimal $\Rightarrow$ finite-model complete, slow
  - Non-minimal $\Rightarrow$ incomplete, can be fast

Current Trends in SMT

- Incorporation of many new theories:
  - Strings and regular expressions
  - Floating point
  - Sets with cardinality constraints
  - Finite Relations
  - …
- Increased support for $\forall$
- New solving algorithms
  - Natural domain SMT, mcSat **[Jovanovic/deMoura 2013]**
- Some work on Optimization Modulo Theories

Optimization Modulo Theories

- Some SMT solvers support optimization queries:
  - νZ (extension of Z3) **[Bjorner/Phan 2014]**
  - OptiMathSAT (extension of MathSat) **[Sebastiani/Tomasi 2014]**

Optimization Modulo Theories

$\texttt{F[cost]} \quad \cup \quad \texttt{l}\leq\texttt{cost}\leq\texttt{u}$

- Given input `F[cost]` where `l≤cost≤u`,
    - Find model that minimizes `cost`

Optimization Modulo Theories

`F[cost]` $\cup$ `l≤cost≤u`

return **`cost=l`**

…if `l=u`

Optimization Modulo Theories

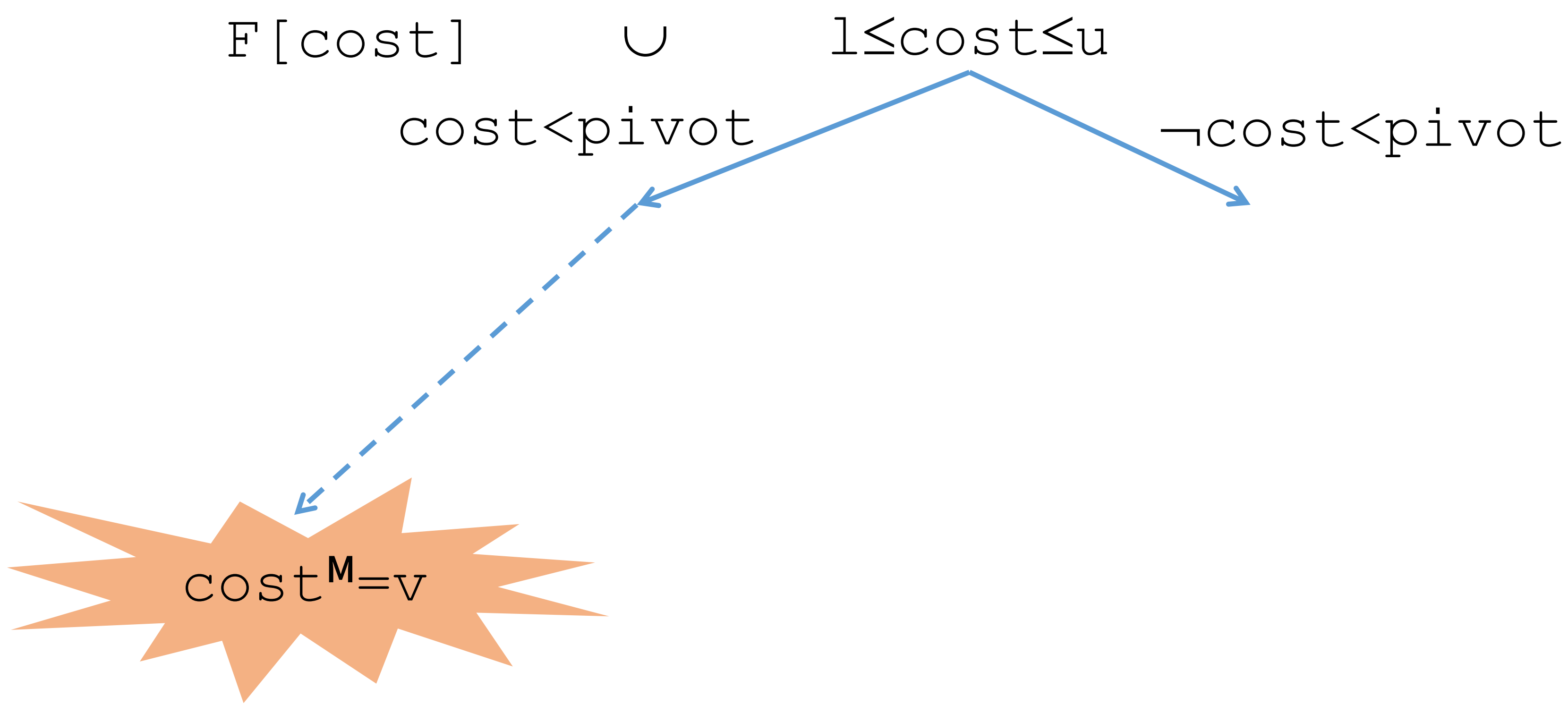
`F[cost]` ∪ `l≤cost≤u`

`cost<pivot`

`¬cost<pivot`

- Otherwise, split on pivot for some `l<pivot<u`

Optimization Modulo Theories



- If we find model where $cost^{M}=v$, update upper bound

Optimization Modulo Theories

`F[cost]` ∪ `l≤cost≤u`

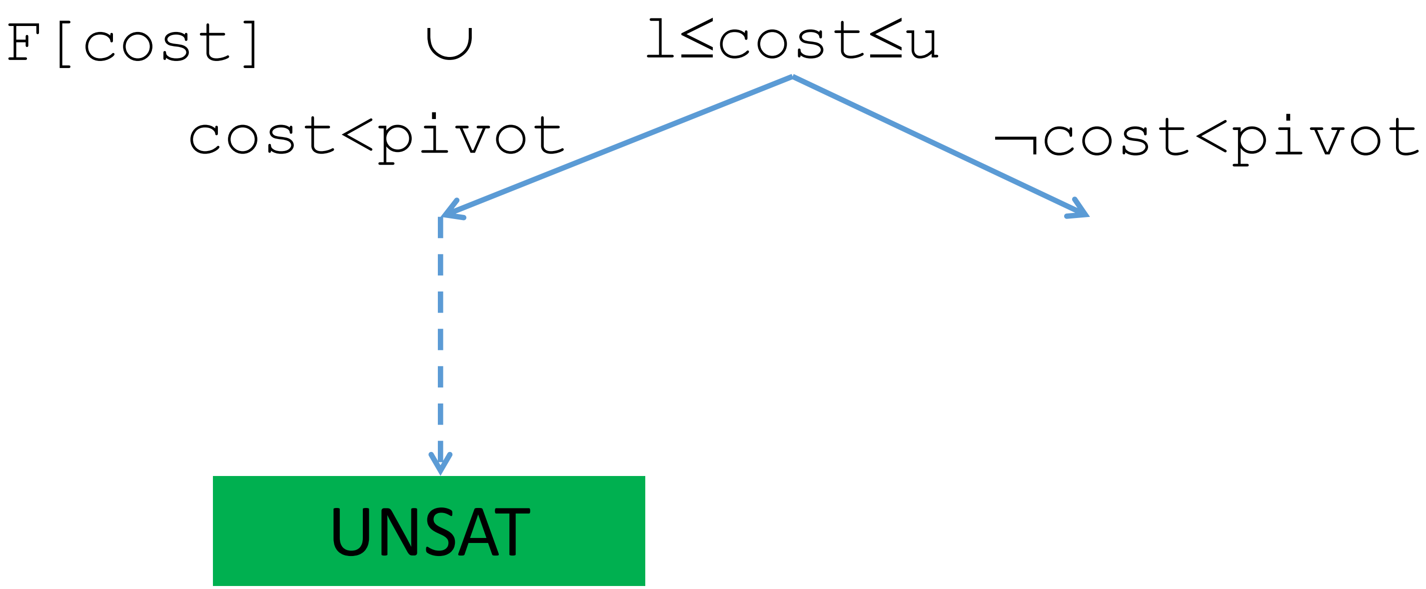
`cost<pivot` `¬cost<pivot`

`l≤cost≤v`

`cost`^{M}`=v`

- If we find model where `cost`^{M}`=v`, update upper bound

Optimization Modulo Theories



- If no model found, update lower bound

Optimization Modulo Theories

F[cost] $\cup$ $l \leq cost \leq u$

$cost < pivot$ $\neg cost < pivot$

$pivot \leq cost \leq u$

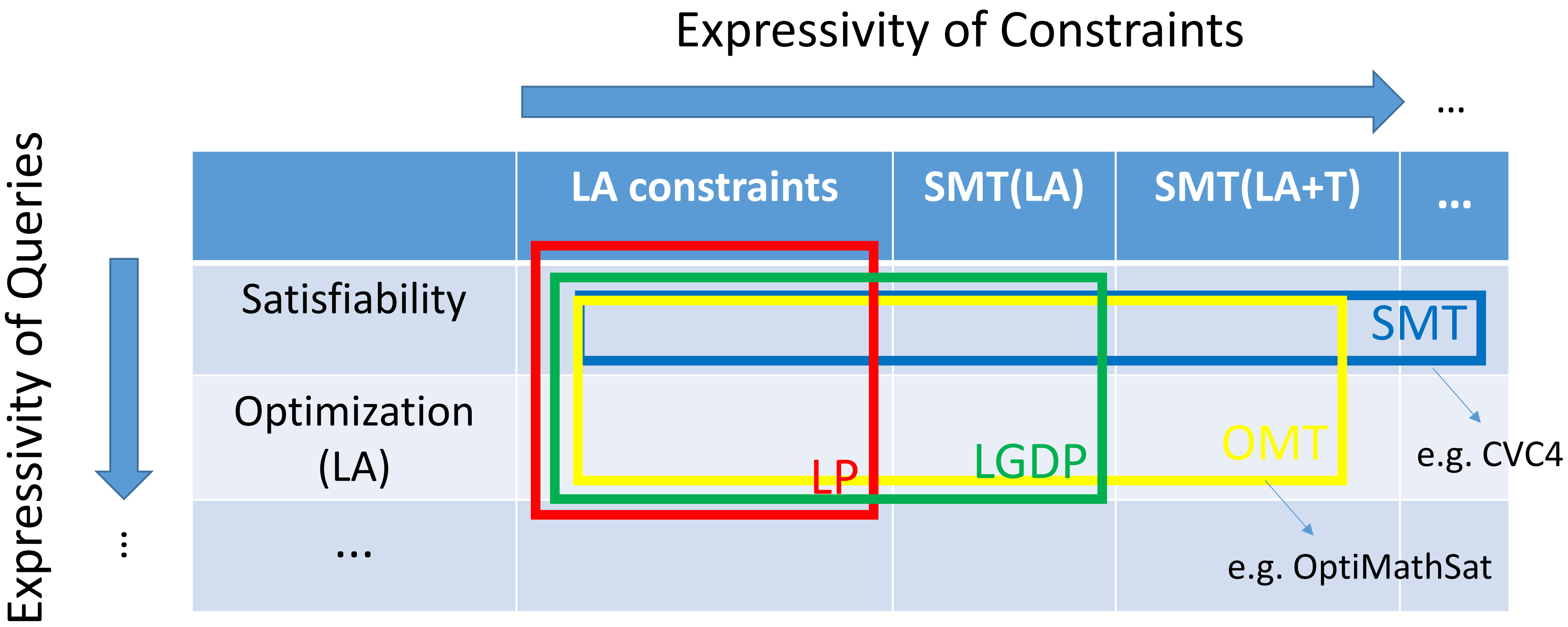
UNSAT

- If no model found, update lower bound

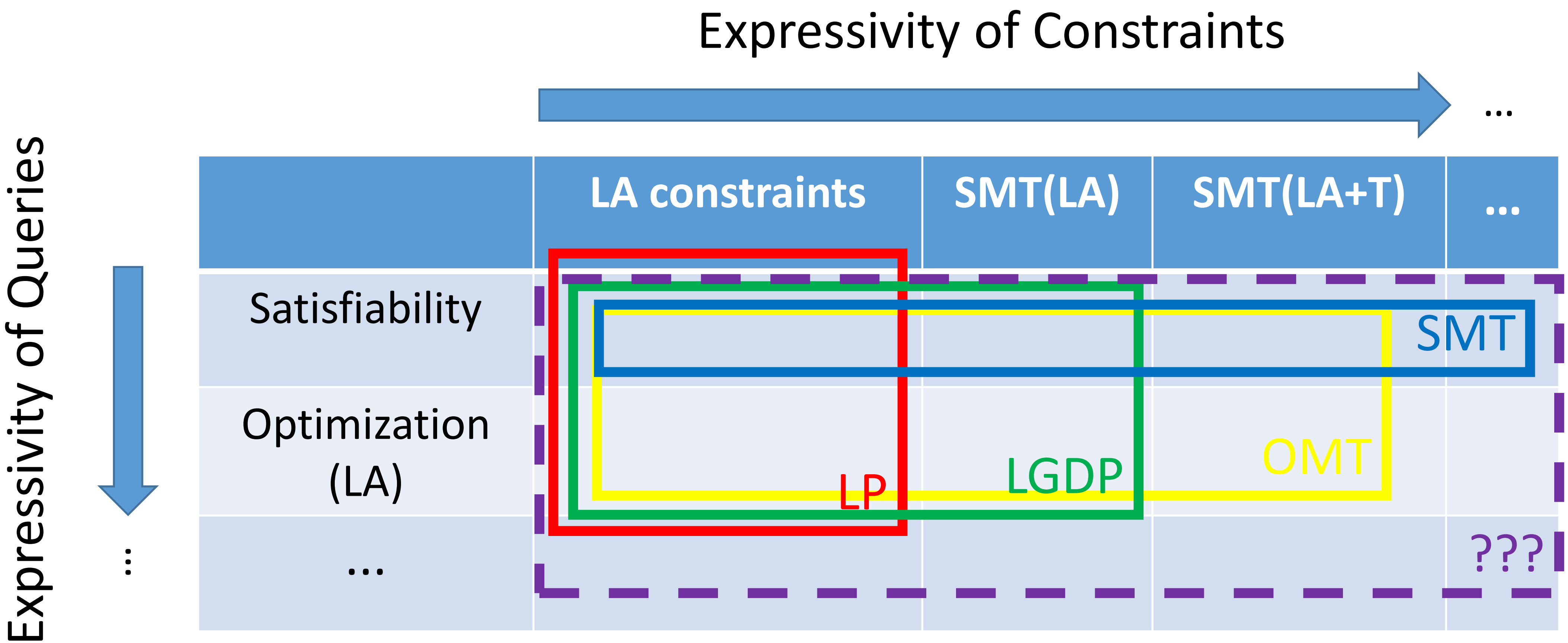
Optimization Modulo Theories

- Similarly, uses branch and bound to minimize cost
  - Modify the behavior of the DPLL search
- Improvements:
  - Use LP solvers to minimize size of cost in models
  - Use conflict analysis to terminate when “unsat” does not depend on cost

Future Work


Expressivity of Constraints
…
LA constraints
SMT(LA)
SMT(LA+T)
…
Expressivity of Queries
Satisfiability
Optimization (LA)
…
SMT
LP
LGDP
OMT
e.g. CVC4
e.g. OptiMathSat

Future Work



⇒Extensions of optimization queries for rich set of theories supported by SMT solvers

Summary

- SMT solvers + DPLL(T) used in many applications
- Can be modified to support model finding and optimization
  - Extensions of theories, e.g. native support for cardinality
  - Modifications to decision heuristics in SAT solver

Thanks for listening!

- SMT Solver CVC4:
  - Open source, available at http://cvc4.cs.nyu.edu/downloads/
    - Supports many theories:
      - UF, Linear arithmetic, Arrays, Strings, Sets, …
    - and techniques mentioned in this talk:
      - Finite model finding, syntax-guided synthesis, etc.