Quantifier Instantiation Techniques for Finite Model Finding in SMT

Andrew Reynolds, Cesare Tinelli
Amit Goel, Sava Krstic
Morgan Deters, Clark Barrett

Satisfiability Modulo Theories (SMT)

- SMT solvers are powerful tools
  - Used in many formal methods applications
  - Support many background *theories*
    - Arithmetic, bitvectors, arrays, datatypes, …
  - May generate:
    - Proofs
      - Theorem proving, software/hardware verification
    - Models
      - Failing instances of aforementioned applications
      - Invariant synthesis, scheduling, test case generation

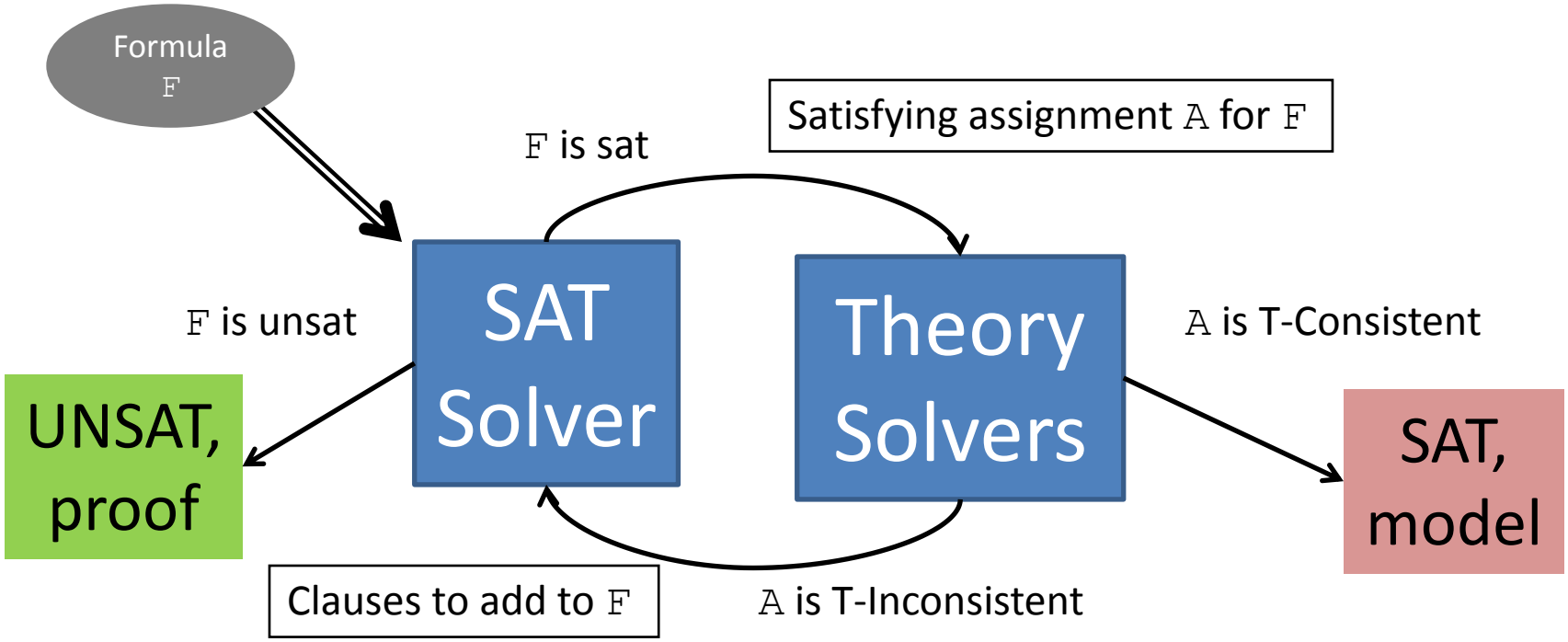
SMT: Limitations

- SMT solvers effective handling *ground* formulas
  - Fast decision procedures for UF, arithmetic, …
- Ongoing challenge: *quantified* formulas
  - Heuristic methods for answering “unsat”
  - Limited capability of answering “sat”
    - Often will return “unknown” after some effort

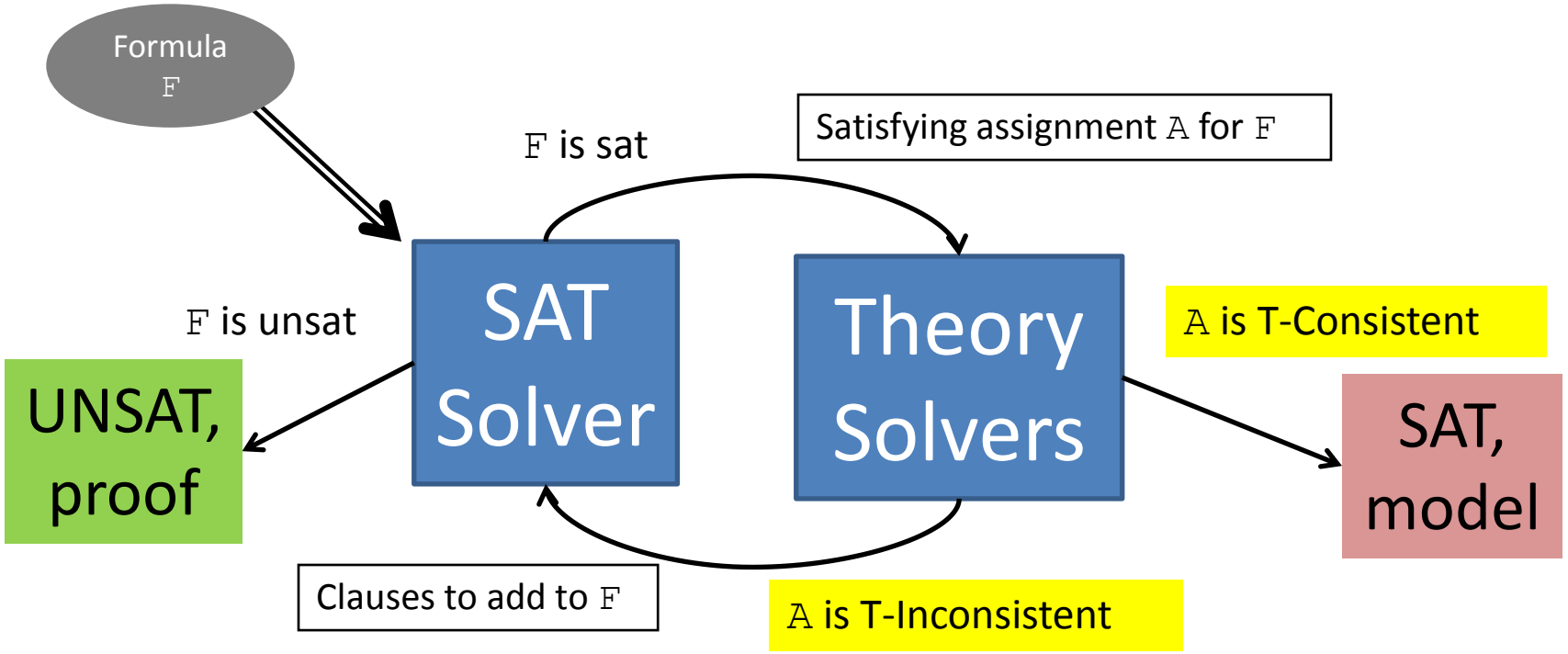
Contributions

- Finite Model Finding in SMT
  - Different from ATP finite model finders:
    - Native support for background (ground) theories
  - Different from SMT solvers:
    - Increased ability to answer “satisfiable”
- New techniques for:
  - Constructing good candidate models
  - Efficiently checking candidate models

DPLL(T) Architecture

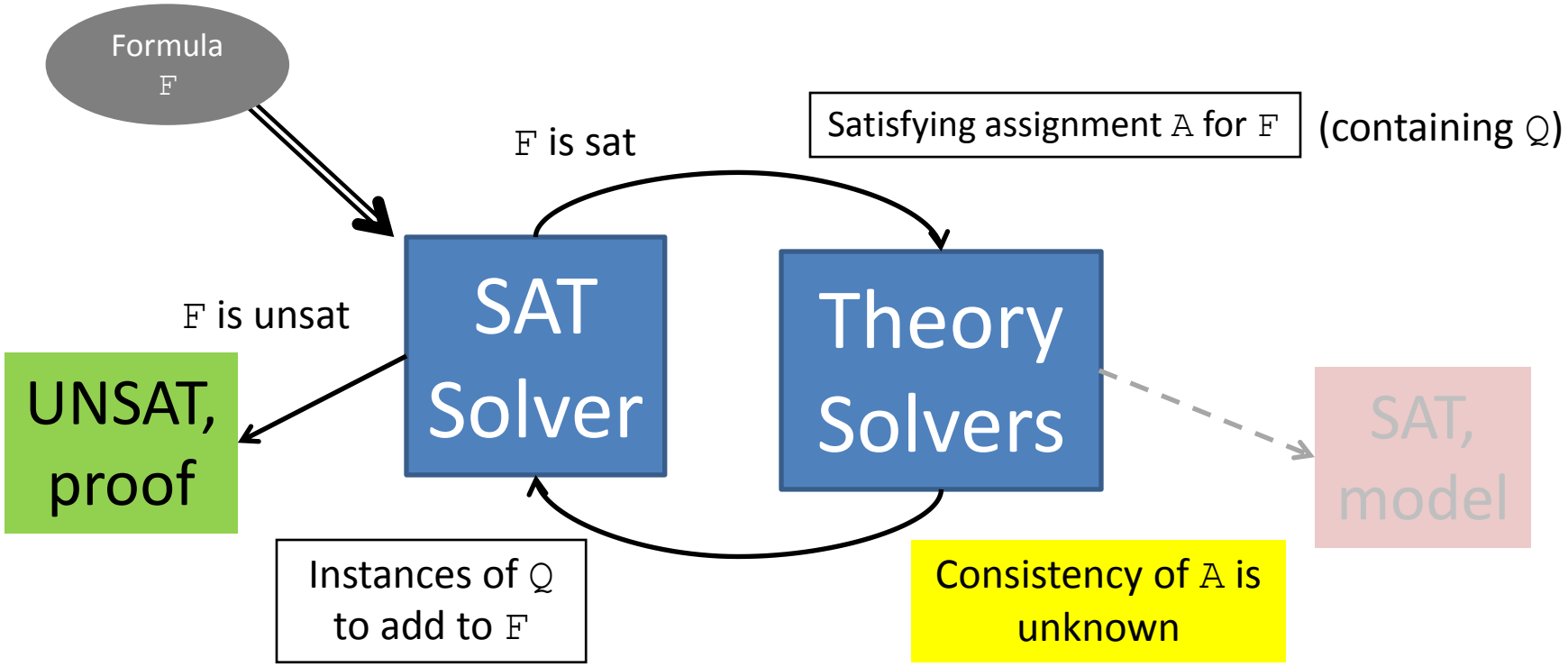


DPLL(T) Architecture : Challenge



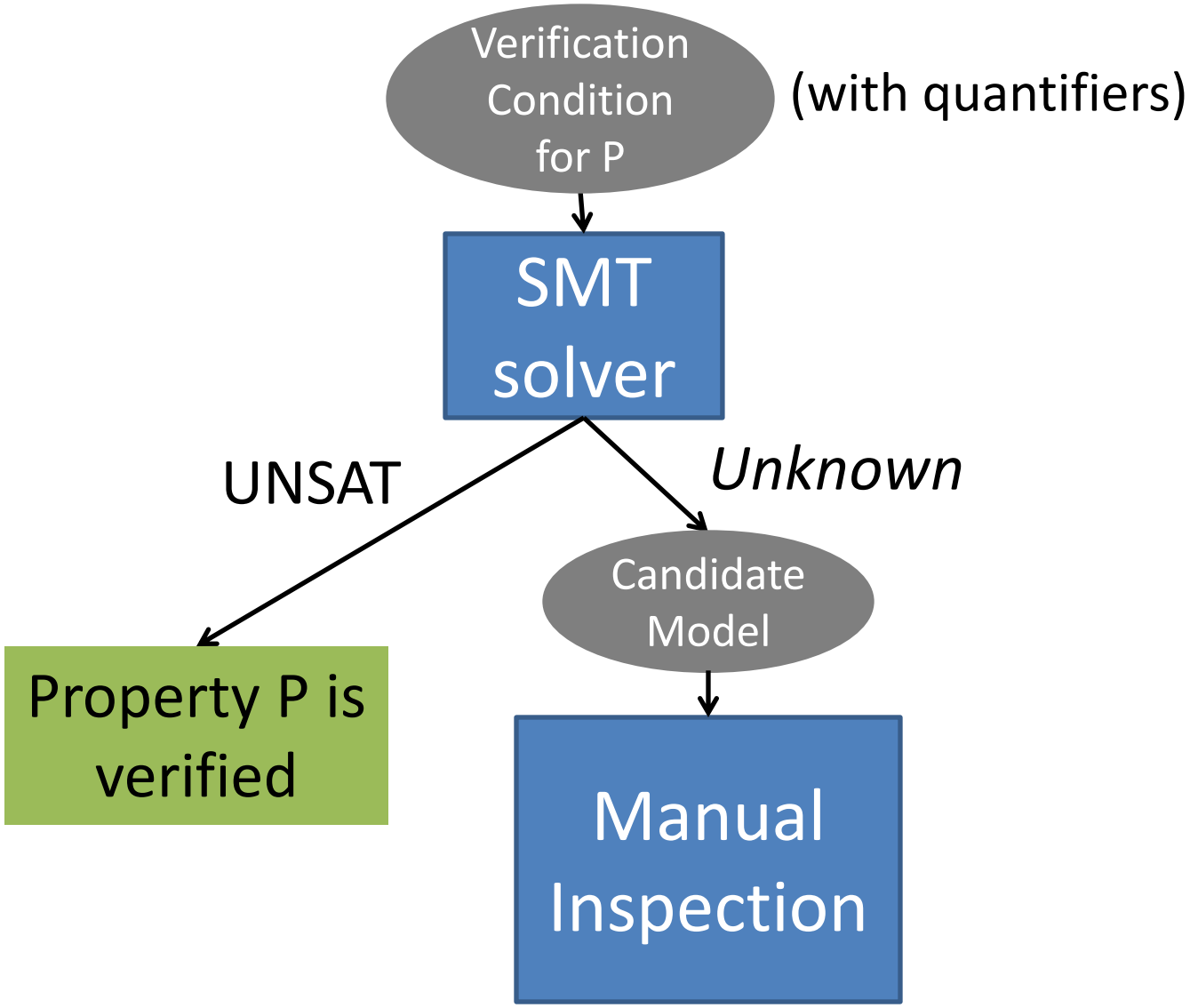
- Challenge: What if determining the consistency of A is difficult?
- For quantified formulas, determining consistency is *undecidable*

Heuristic Instantiation for Q

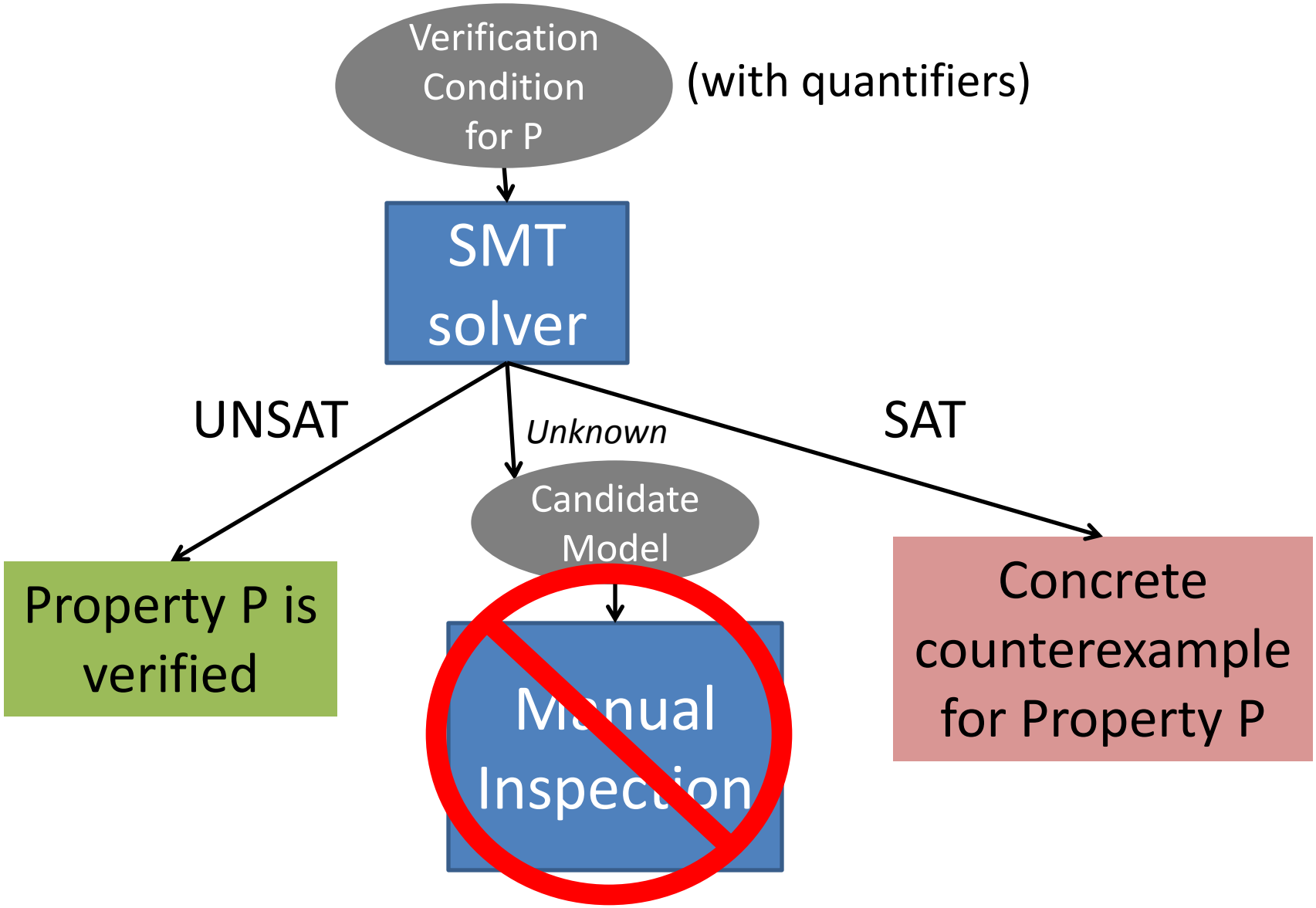


- If sat assignment contains quantified formula Q,
  - Heuristically add instances of Q to F
    - Typically based on pattern matching
  - May discover refutation, if right instances are added

Why Models are Important



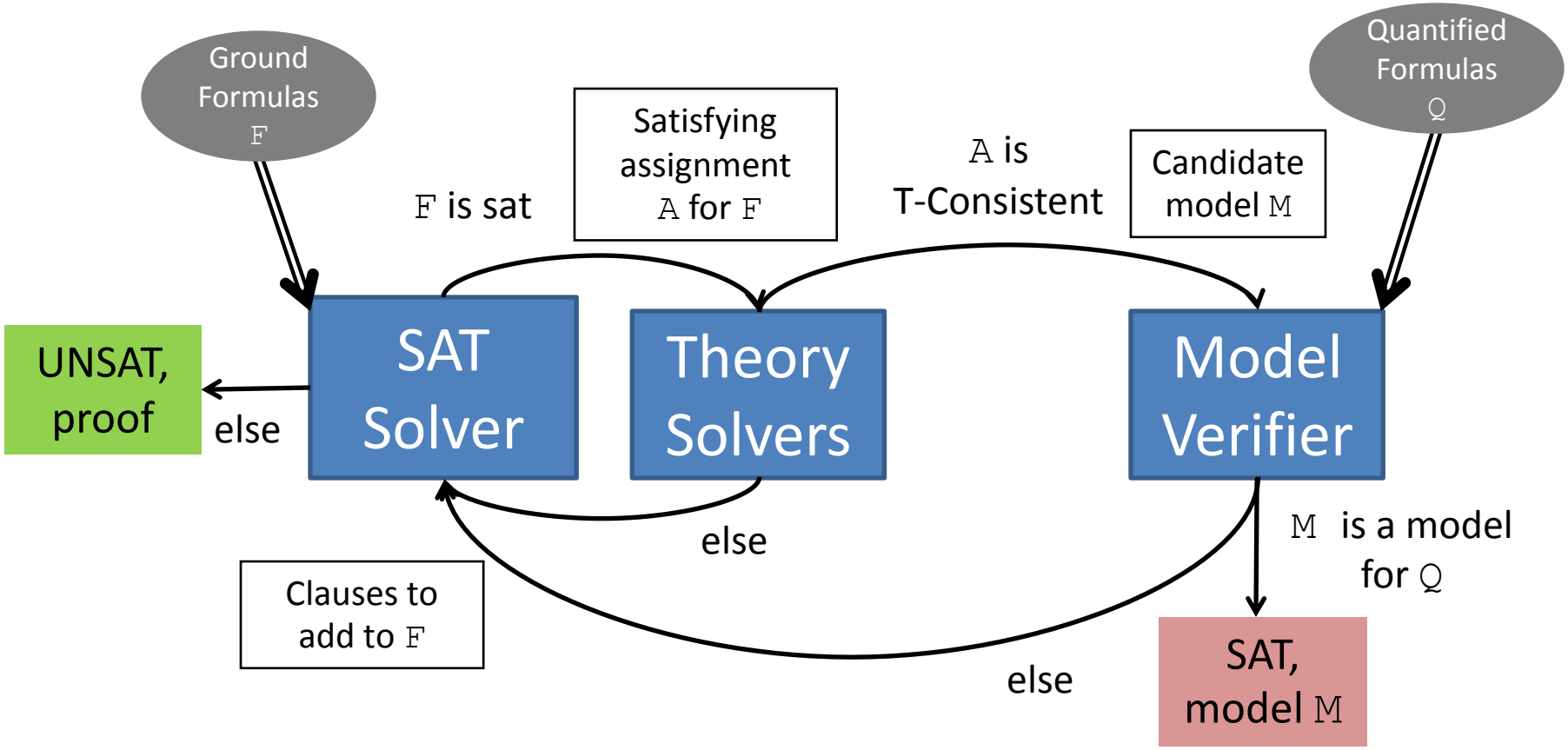
Why Models are Important



Model-Based Approach for Quantifiers

- Given:
  - Set of ground formulas $\mathbb{F}$
  - Set of universally quantified formulas $\mathbb{Q}$
- To determine the satisfiability of $\mathbb{F} \wedge \mathbb{Q}$,
  - Construct candidate models for $\mathbb{Q}$, based on satisfying assignments for $\mathbb{F}$
    - Model-Based Quantifier Instantiation (MBQI)
      - [Ge/deMoura 2009]

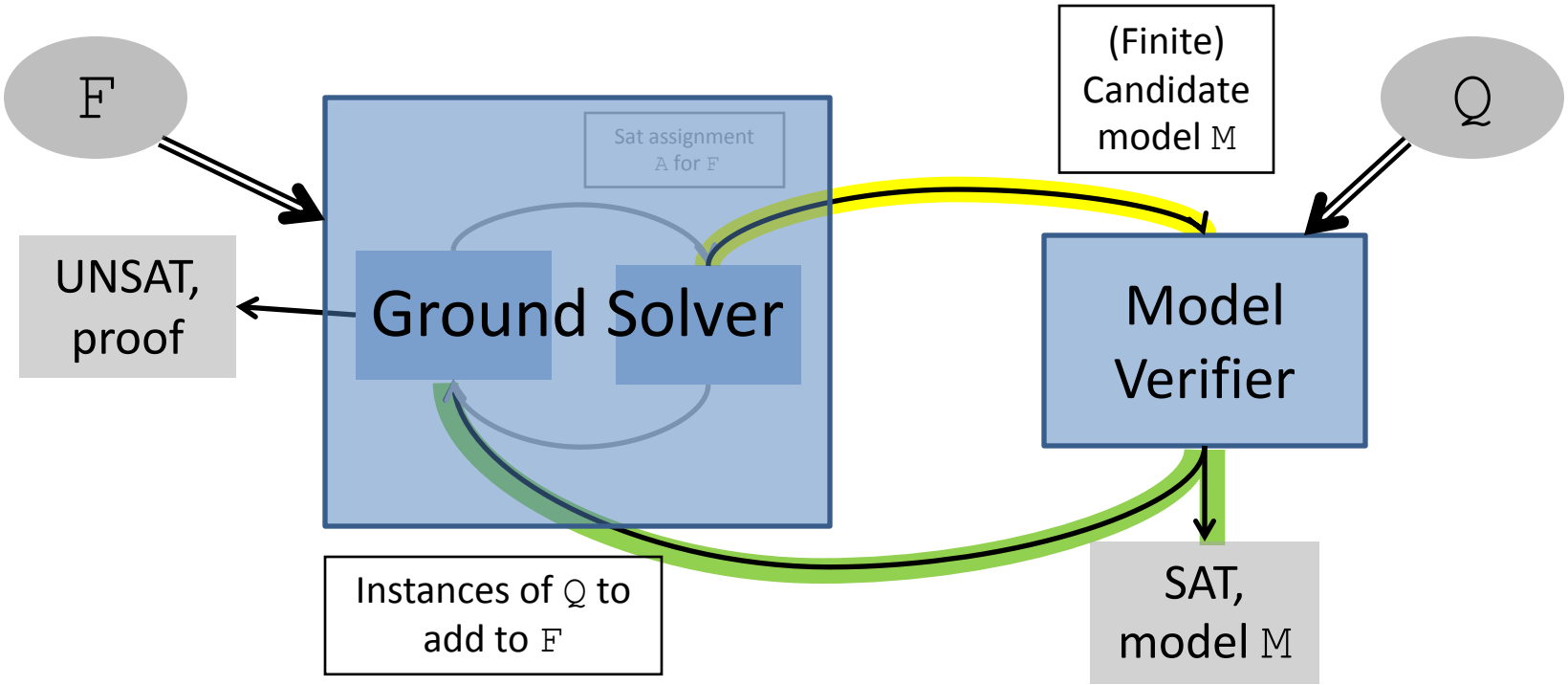
DPLL(T) Architecture (Extended)



When can we represent/check models for Q?

- Focus of talk: Finite Model Finding
  - Limited to quantifiers over:
    - Uninterpreted sorts
      - Can represent memory addresses, values, sets, etc.
    - Other finite sorts
      - Fixed width bitvectors, datatypes, …
- Useful in applications:
  - Software/hardware verification

Constructing/Checking Candidate Finite Models



1. How do we construct good candidate models M?
2. How do we efficiently check if M is a model for Q?
   - If we fail, which instances do we add to F?

Constructing Good Candidate Models

Constructing Good Candidate Models

- Naïvely, to determine whether $\mathtt{M}$ is model for $\mathtt{Q}$:
  - Check if $\mathtt{M}$ satisfies *all instances* $\mathtt{S}$ of $\mathtt{Q}$
- Challenge: $\mathtt{S}$ can be very large
  - For $\mathtt{Q}$ with n variables, domain size d, $|\mathtt{S}|$ can be $O(d^n)$
- Solutions:
  - Find models with small domain sizes
    - Use theory of finite cardinality constraints [CAV 2013]
  - Only consider instances of $\mathtt{Q}$ that are false in $\mathtt{M}$
    - *Construct $\mathtt{M}$ such that most instances of $\mathtt{Q}$ are true*

Constructing Models : Example

$\text{person}_1, \text{person}_2, \text{person}_3$: Person

NewYork, Boston, Seattle : City

salesman: Person $\rightarrow$ Bool

travels : Person $\times$ City $\rightarrow$ Bool

$\mathbb{F}$

distinct(NewYork, Boston, Seattle)

travels(person_1, Boston)

$\neg$salesman(person_2)

salesman(person_3)

$\mathbb{Q}$

$\forall$ x : Person, y : City.

salesman(x) $\Rightarrow$ travels(x,y)

Constructing Models : Example

$person_1$, $person_2$, $person_3$: Person
NewYork, Boston, Seattle : City
salesman: Person $\rightarrow$ Bool
travels : Person $\times$ City $\rightarrow$ Bool

$\mathbb{F}$

distinct(NewYork, Boston, Seattle)
travels($person_1$, Boston)
$\neg$salesman($person_2$)
salesman($person_3$)

$\mathbb{Q}$

$\forall$ x : Person, y : City.
salesman(x) $\Rightarrow$ travels(x,y)

Interpretation of "salesman":

salesman($person_2$) $\approx \perp$,
salesman($person_3$) $\approx$ T,
salesman(x_1) $\approx \perp$

Interpretation of "travels":

travels($person_1$, Boston) $\approx$ T,
travels(x_1, x_2)$\approx \perp$

Model Representation

- Represent functions/predicates using *defining maps*
- Given sort S with domain V,
  - A defining map for $f : S \times \ldots \times S \rightarrow S$ is:
    - Set of equations Δ_f of the form $f(t_1, \ldots, t_n) \approx v$, where
      - $v \in V$
      - Each t_i is either a unique variable or in V
    - If $t_1 \approx v_1$ and $t_2 \approx v_2 \in \Delta_f$, unifiable with mgu σ, then:
      - σ is non-empty
      - $t_1 \sigma \approx v \in \Delta_f$ for some v
    - $f(x_1, \ldots, x_n) \approx v \in \Delta_f$ for some v

Interpretation in Model

- Interpretation $\mathtt{f}(\ \mathtt{t}_1, \ldots, \mathtt{t}_n\ )$ is $\mathtt{v}$, where:
  - $\mathtt{t} \approx \mathtt{v} \in \Delta_{\mathtt{f}}$
  - $\mathtt{t}$ is most specific generalization of $\mathtt{f}(\ \mathtt{t}_1, \ldots, \mathtt{t}_n\ )$ among LHS in $\Delta_{\mathtt{f}}$
    - Guaranteed to exist and be unique

Constructing Models

- Defining map Δ_f is a union of:
  - Entailed ground equalities
  - Non-ground equalities for defining *default* values
- How to chose default values?
  - Guided by sat assignment for *distinguished elements*
    - See how one instance is satisfied, generalize this for all

Constructing Models : Example

$person_1, person_2, person_3$: Person
NewYork, Boston, Seattle : City
salesman: Person $\rightarrow$ Bool
travels : Person $\times$ City $\rightarrow$ Bool

$\mathbb{F}$

distinct(NewYork, Boston, Seattle)
$\neg$travels($person_1$, Boston)
$\neg$salesman($person_2$)
salesman($person_3$)

$\mathbb{Q}$

$\forall$ x : Person, y: City.
salesman(x) $\Rightarrow$ travels(x,y)

salesman($person_1$) $\Rightarrow$ travels($person_1$, NewYork)

- Instantiate $\mathbb{Q}$ with distinguished elements

Constructing Models : Example

$person_1, person_2, person_3$: Person
NewYork, Boston, Seattle : City
salesman: Person → Bool
travels : Person × City → Bool

F

distinct(NewYork, Boston, Seattle)
¬travels($person_1$, Boston)
¬salesman($person_2$)
salesman($person_3$)

Q

∀ x : Person, y: City.
salesman(x) ⇒ travels(x,y)

salesman($person_1$) ⇒ travels($person_1$, NewYork) } T

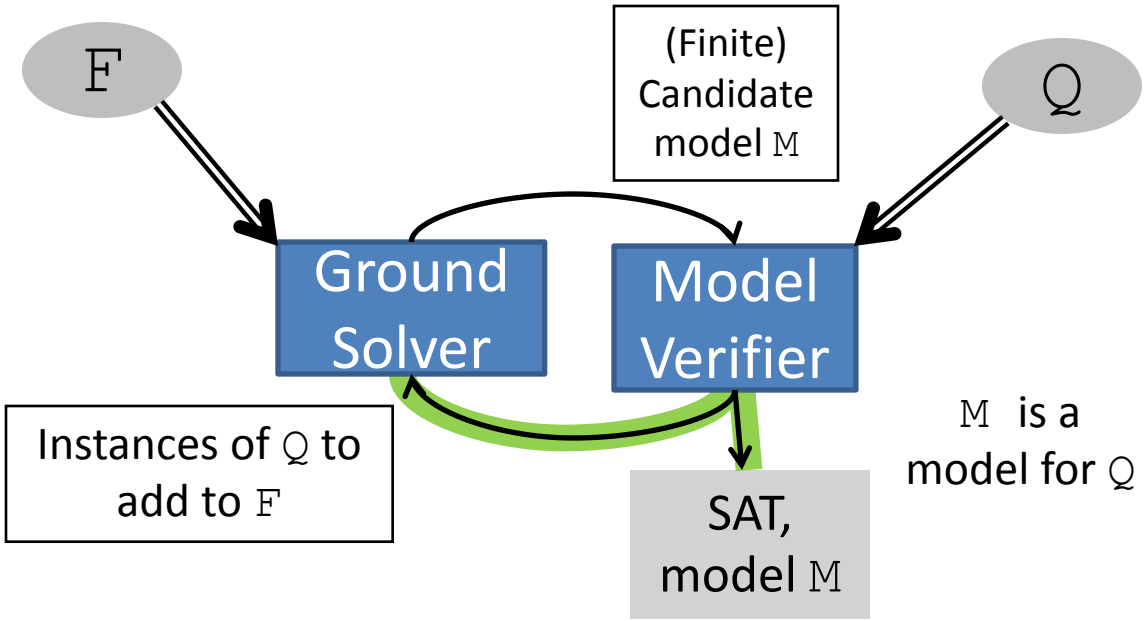
- Choose defaults based on distinguished instance
  - Analagous to Inst Gen Calculus [Korovin 2008]

$\Delta_{salesman}$:
{ salesman($person_2$) ≈ ⊥,
salesman($person_3$) ≈ T,
salesman(x_1) ≈ T }

$\Delta_{travels}$:
{ travels($person_1$, NewYork) ≈ T
travels($person_1$, Boston) ≈ ⊥,
travels(x_1, x_2)≈ T }

Efficiently Checking Candidate Models

Checking Candidate Models



- To check if M is a model for Q:
  - Naively, add every instance of Q to F
  - *Alternatively*, only add instances that are false in M
    - Identify sets of instances of Q that are equisatisfiable

Checking Candidate Models

$person_1$, $person_2$, $person_3$: Person
NewYork, Boston, Seattle : City
salesman: Person → Bool
travels : Person × City → Bool

$\mathbb{F}$

distinct(NewYork, Boston, Seattle)
¬travels($person_1$, Boston)
¬salesman($person_2$)
salesman($person_3$)

salesman($person_1$) ⇒ travels($person_1$, NewYork)

$\mathbb{Q}$

∀ x : Person, y: City.
salesman(x) ⇒ travels(x,y)

$\Delta_{salesman}$:

{ salesman($person_2$) ≈ ⊥,
salesman($person_3$) ≈ T,
salesman(x_1) ≈ T }

$\Delta_{travels}$:

{ travels($person_1$, NewYork) ≈ T
travels($person_1$, Boston) ≈ ⊥,
travels(x_1, x_2)≈ T }

$\mathbb{Q}$[$person_1$, NewYork]
$\mathbb{Q}$[$person_1$, Boston]
$\mathbb{Q}$[$person_1$, Seattle]
$\mathbb{Q}$[$person_2$, NewYork]
$\mathbb{Q}$[$person_2$, Boston]
$\mathbb{Q}$[$person_2$, Seattle]
$\mathbb{Q}$[$person_3$, NewYork]
$\mathbb{Q}$[$person_3$, Boston]
$\mathbb{Q}$[$person_3$, Seattle]

Checking Candidate Models

$person_1$, $person_2$, $person_3$: Person
NewYork, Boston, Seattle : City
salesman: Person → Bool
travels : Person × City → Bool

$\mathbb{F}$

distinct(NewYork, Boston, Seattle)
¬travels($person_1$, Boston)
¬salesman($person_2$)
salesman($person_3$)

salesman($person_1$) ⇒ travels($person_1$, NewYork)

$\mathbb{Q}$

∀ x : Person, y: City.
salesman(x) ⇒ travels(x,y)

$\Delta_{salesman}$:

{ salesman($person_2$) ≈ ⊥,
salesman($person_3$) ≈ T,
salesman(x_1) ≈ T }

$\Delta_{travels}$:

{ travels($person_1$, NewYork) ≈ T
travels($person_1$, Boston) ≈ ⊥,
travels(x_1, x_2)≈ T }

$\mathbb{Q}$[$person_1$, NewYork] — true
$\mathbb{Q}$[$person_1$, Boston] — false
$\mathbb{Q}$[$person_1$, Seattle] — true
$\mathbb{Q}$[$person_2$, NewYork] — true
$\mathbb{Q}$[$person_2$, Boston]
$\mathbb{Q}$[$person_2$, Seattle]
$\mathbb{Q}$[$person_3$, NewYork] — true
$\mathbb{Q}$[$person_3$, Boston]
$\mathbb{Q}$[$person_3$, Seattle]

Enhancement: Heuristic Instantiation

- Idea:
  - First see if instantiations based on heuristics exist
  - If not, resort to model-based instantiation
- May lead to:
  - Discovering easy conflicts, if they exist
  - Arriving at model faster
    - Instantiations rule out spurious models

Experiments

- DVF Benchmarks
  - Taken from verification tool DVF used by Intel
  - Both SAT/UNSAT benchmarks
    - SAT benchmarks generated by removing necessary pf assumptions
  - Many theories: UF, arithmetic, arrays, datatypes
  - Quantifiers only over free sorts
    - Memory addresses, Values, Sets, …
- TPTP Benchmarks
- Isabelle Benchmarks
  - Provable and unprovable goals, contains some arithmetic

Results: DVF

SAT	german	refcount	agree	apg	bmk	Total	Time
#	45	6	42	19	37	149	
z3	**45**	1	0	0	0	46	8.1
cvc4+i	2	0	0	0	0	2	0.0
cvc4+f	**45**	**6**	**42**	18	36	147	1413.1
cvc4+fi	**45**	**6**	**42**	**19**	36	148	1333.9
cvc4+fm	**45**	**6**	**42**	**19**	**37**	**149**	605.4
cvc4+fmi	**45**	**6**	**42**	**19**	**37**	**149**	409.8

UNSAT	german	refcount	agree	apg	bmk	Total	Time
#	145	40	488	304	244	1221	
z3	**145**	**40**	**488**	**304**	**244**	**1221**	31.0
cvc4+i	**145**	**40**	484	**304**	**244**	1217	21.3
cvc4+f	**145**	**40**	476	298	242	1201	7512.2
cvc4+fi	**145**	**40**	**488**	302	**244**	1219	1181.4
cvc4+fm	**145**	**40**	471	300	242	1198	6949.7
cvc4+fmi	**145**	**40**	**488**	302	**244**	1219	1185.0

cvc4 :

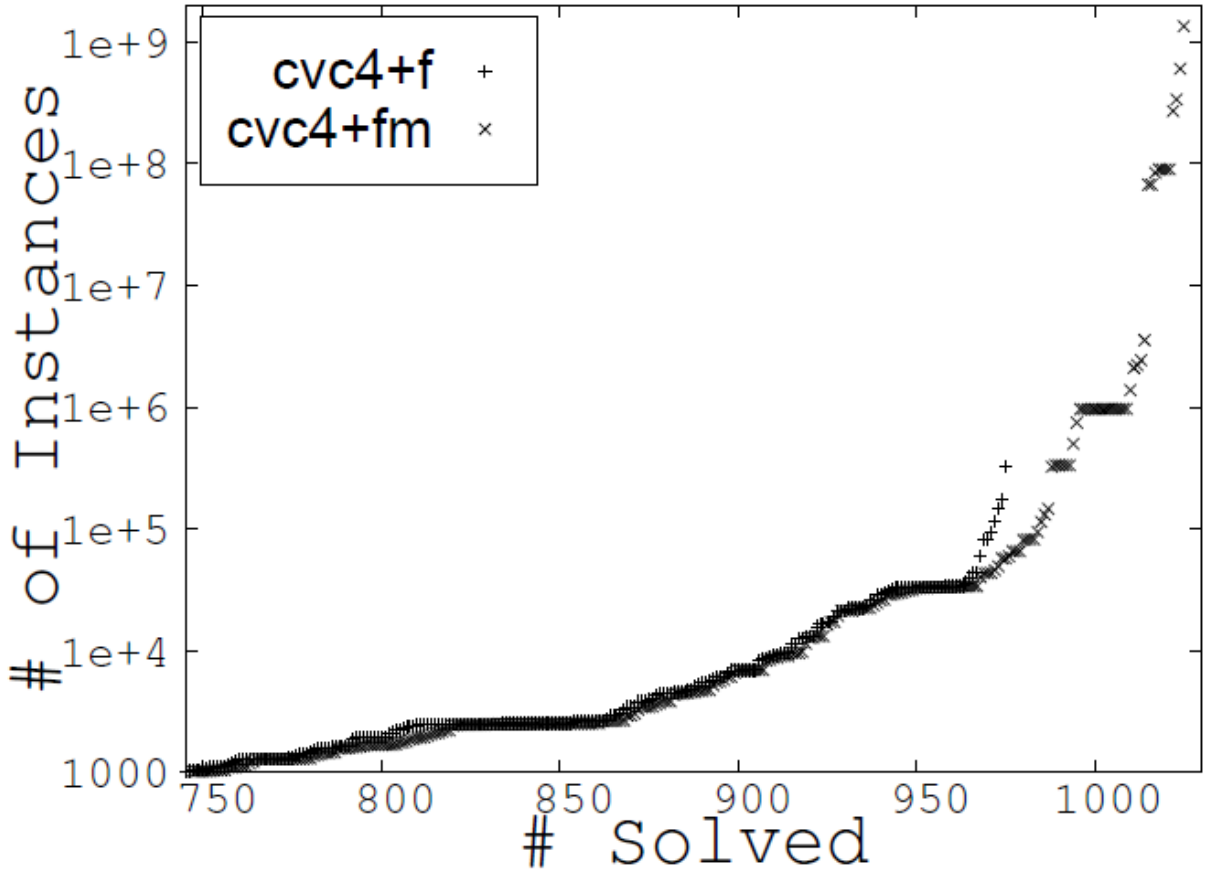
- f : finite model
- i : heuristic
- m : model-based

- cvc4 with finite model finding (cvc4+f)
  - Effective for answering sat
  - Using heuristic instantiation, solves 4 unsat that cvc4 cannot

Results: TPTP

- Using techniques described in this work:
  - Of 1995 satisfiable benchmarks:
    - Paradox solves 1305
    - iProver solves 1231
    - cvc4 solves 1109 (with model-based instantiation)
      - Includes 2 problems with rating 1.0
  - Of 12568 unsatisfiable benchmarks:
    - z3 solves 5934
    - cvc4 solves 3028 (with model-based+heuristic instantiation)
      - Orthogonal, 282 cannot be solved by z3
- Placed 3rd in FNT (non-theorem) division of CASC 24

Results : TPTP



- Model-Based Instantiation is often essential
  - Solves where exh. instantiation require >1 billion instances

Results: Isabelle

SAT	Arrow	FFT	FTA	Hoare	NS	QEP	SNorm	TwoSq	TypeSafe	Total
z3	3	19	24	46	10	49	1	17	11	180
cvc4+i	0	9	0	0	0	0	0	8	0	17
cvc4+f	22	138	172	153	56	79	**12**	59	69	760
cvc4+fm	**26**	139	171	151	49	80	**12**	59	69	756
cvc4+fmi	**26**	**151**	**174**	**159**	**60**	**81**	**12**	**60**	**78**	**801**

UNSAT	Arrow	FFT	FTA	Hoare	NS	QEP	SNorm	TwoSq	TypeSafe	Total
z3	**261**	**224**	**765**	**497**	**135**	236	**240**	451	**325**	**3134**
cvc4+i	199	217	682	456	97	**244**	231	**486**	239	2851
cvc4+f	120	99	298	214	36	105	84	316	132	1404
cvc4+fm	102	91	330	246	26	117	80	310	128	1430
cvc4+fmi	155	170	467	328	42	161	97	411	188	2019

- cvc4+fmi solves 244 unsat that z3 cannot, 164 that cvc4 cannot

cvc4 :

- f : finite model
- i : heuristic
- m : model-based

Summary

- CVC4 with finite model finding:
  - Constructs “good” candidate models
  - Incorporates various instantiation strategies
    - Model-based quantifier instantiation
    - Heuristic instantiation (E-matching)
  - Increased ability to answer “satisfiable”
- Publicly available: http://cvc4.cs.nyu.edu/web/

Further Work

- Further work:
  - Would like to show:
    - Finite model completeness
    - Refutational completeness for certain fragments
  - Improved algorithm for checking candidate models
  - Apply similar techniques to:
    - Bounded integer quantification
    - Datatypes, strings with bounded length

Thank you

- Questions?