Generating Small Countermodels using SMT

Andrew Reynolds

Intel

August 30, 2012

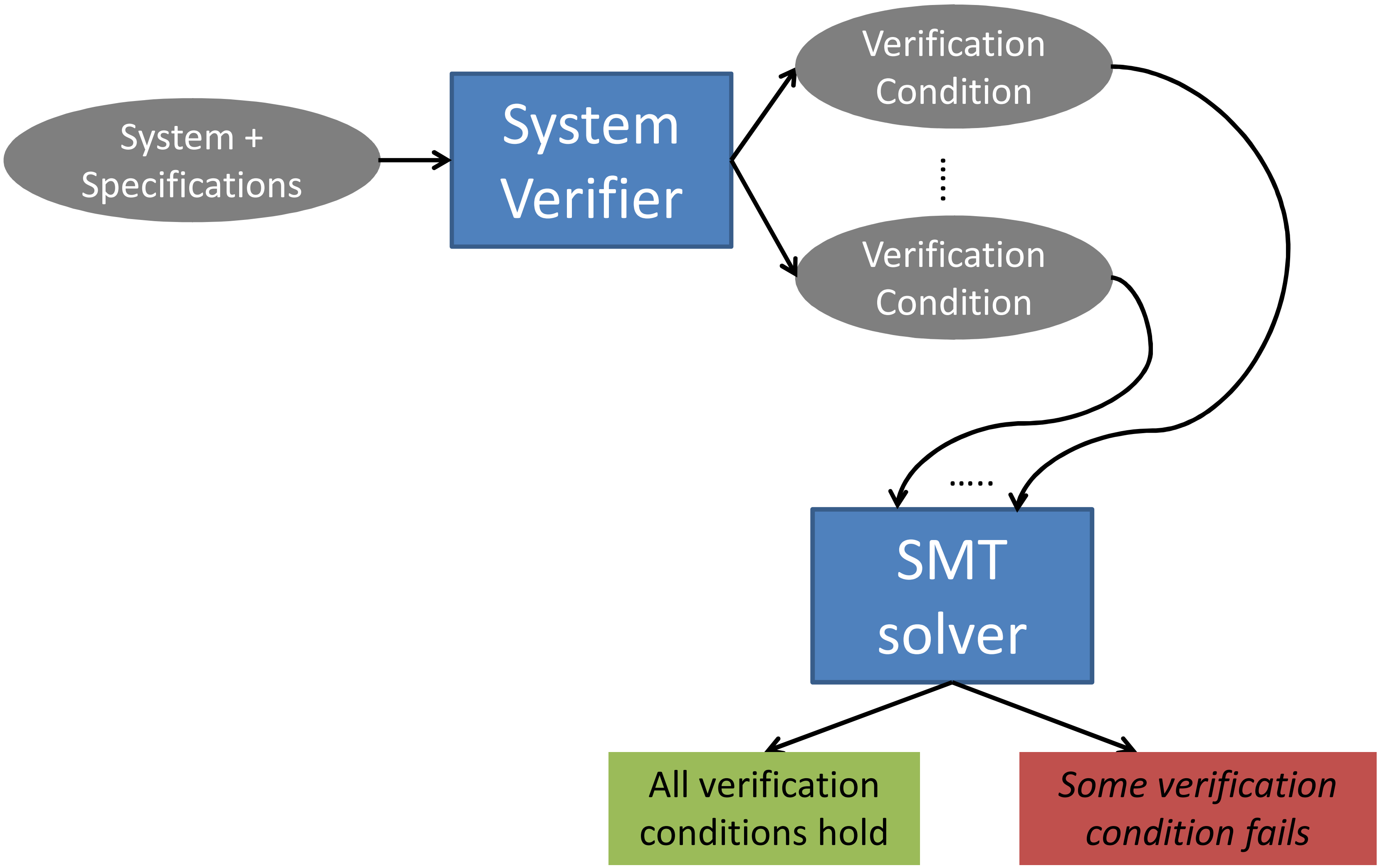
Acknowledgements

- Intel Corporation
  - Amit Goel, Sava Krstic
- University of Iowa
  - Cesare Tinelli, Francois Bobot
- New York University
  - Clark Barrett, Morgan Deters, Dejan Jovanovic

Overview

- SMT-Based System Verification
  - Deductive Verification Framework (DVF)
- SMT Overview
- Challenge of quantifiers in SMT
- Finite Model Finding:
  - Searching for small models
  - Checking models against quantifiers
- Experimental Results

SMT-Based System Verification



DVF

- Deductive Verification Framework
- Used for:
  - Architecture Validation
  - SOC Security Validation
- Language tailors to constraints SMT solvers can handle
  - Arithmetic, arrays, datatypes (enumerations, sum types, …)
- This allows:
  - Tight integration with SMT solver
    - DVF program annotations can help SMT solver
    - SMT solver responses correspond to original program

DVF Example

Definitions

```
type resource
const resource null
type process
var array(resource, bool) valid = mk_array[resource](false)
var array(resource, int) count
var array(process, resource) ref = mk_array[process](null)
…
module S = Set<type process>
```

Transition System

```
transition create (resource r)
require (r != null, !valid[r]){
  valid[r] := true;
  count[r] := 0;
}
…
```

Properties

```
def bool prop = forall (process p) (ref[p] != null => valid[ref[p]])
def bool refs_non_zero = forall (process p) (ref[p] != null => count[ref[p]] > 0)
…
```

Goals

```
goal main = invariant prop assuming refs_non_zero
…
goal rnz  = formula (… && prop && … => refs_non_zero)
```

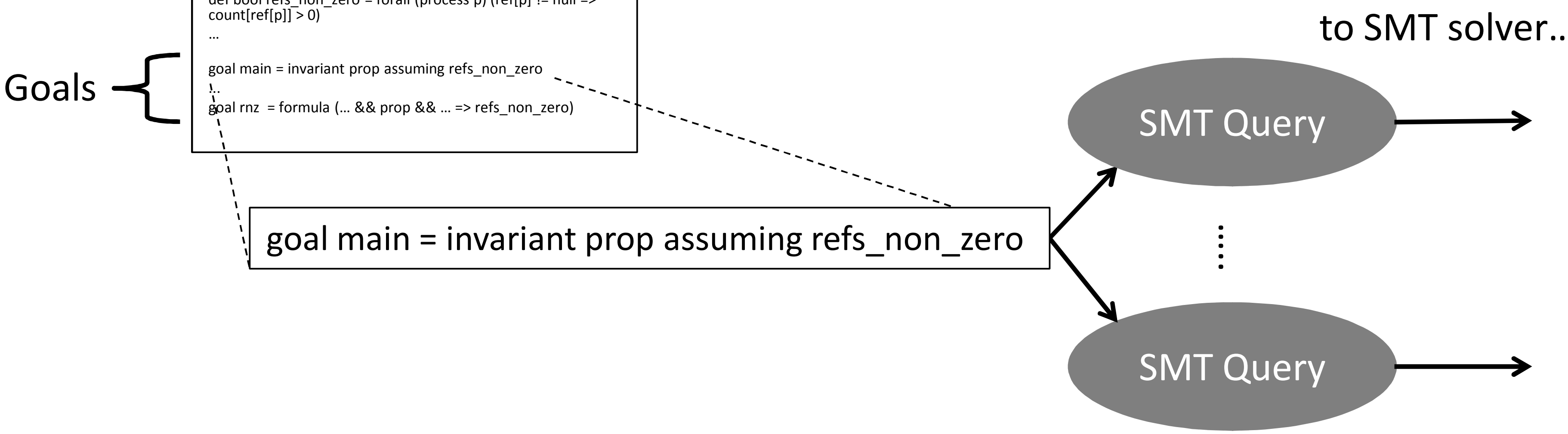
DVF SMT Backend

```
type resource
const resource null
type process
var array(resource, bool) valid = mk_array[resource](false)
var array(resource, int) count
var array(process, resource) ref = mk_array[process](null)
…
module S = Set<type process>

transition create (resource r)
require (r != null, !valid[r]){
  valid[r] := true;
  count[r] := 0;
}
…

def bool prop = forall (process p) (ref[p] != null => valid[ref[p]])
def bool refs_non_zero = forall (process p) (ref[p] != null =>
count[ref[p]] > 0)
…

goal main = invariant prop assuming refs_non_zero
…
goal rnz  = formula (… && prop && … => refs_non_zero)
```



- Goals translated into (possibly multiple) SMT queries
  - Example: base/induction cases for proofs

SMT Query

Definitions

S, P, R : type
null : R
valid: Array(R, Bool)
count: Array(R, Int)
ref: Array(P, R)
empty : S
mem : (S, P) -> Bool
add, remove : (S, P) -> S
…

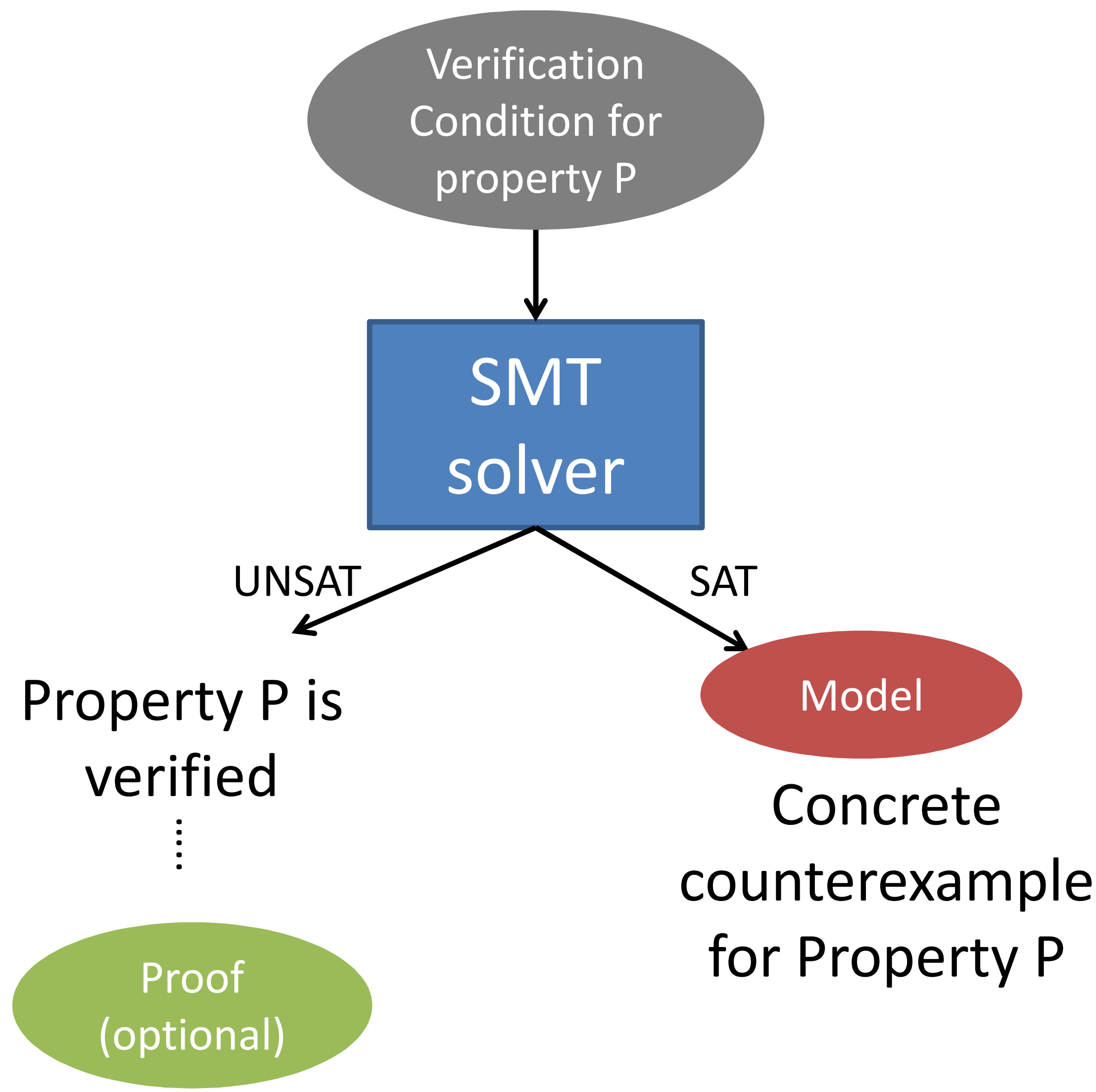
Axioms

$\forall x : R.\ count[x] > 0 \Rightarrow valid[\ x\ ]$
$\forall x : P.\ \neg\ mem(\ empty, x\ )$
$\forall x : S, y, z : P.\ mem(\ add(\ x, y\ ), z\ ) \Rightarrow (\ z = y \vee mem(\ x, z\ )\ )$
$\forall x : S, y, z : P.\ mem(\ remove(\ x, y\ ), z\ ) \Rightarrow (\ z \neq y \wedge mem(\ x, z\ )\ )$
…

$\neg$ (… $\forall$x. (ref[x] != null => valid[ref[x]]) …)

Property to verify

SMT for Verification Conditions



Satisfiability Modulo Theories (SMT)

- SMT solvers:
  - Are powerful tools for determining satisfiability of ground formulas
    - Built-in decision procedures for many theories
  - Have applications in:
    - Software/Hardware verification
    - Planning and scheduling
    - Design automation
  - Had significant performance improvement in past 10 years
  - Many solvers use standard format
    - SMT LIB initiative

CVC4 : SMT Solver

- Support for many theories
  - Equality + Uninterpreted Functions
  - Integer/Real arithmetic
  - Bit Vectors
  - Arrays
  - Datatypes
- Work in progress: Quantifiers
  - Pattern-based instantiation
  - Model-based instantiation
  - Rewrite Rules
  - *Finite Model Finding*
- Highly competitive
  - Won multiple divisions of SMT COMP 2012

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

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

Convert to boolean satisfiability problem

$$\Downarrow$$

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

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

$$\Downarrow$$

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

A: T, C: T

Find satisfying assignment … A, C

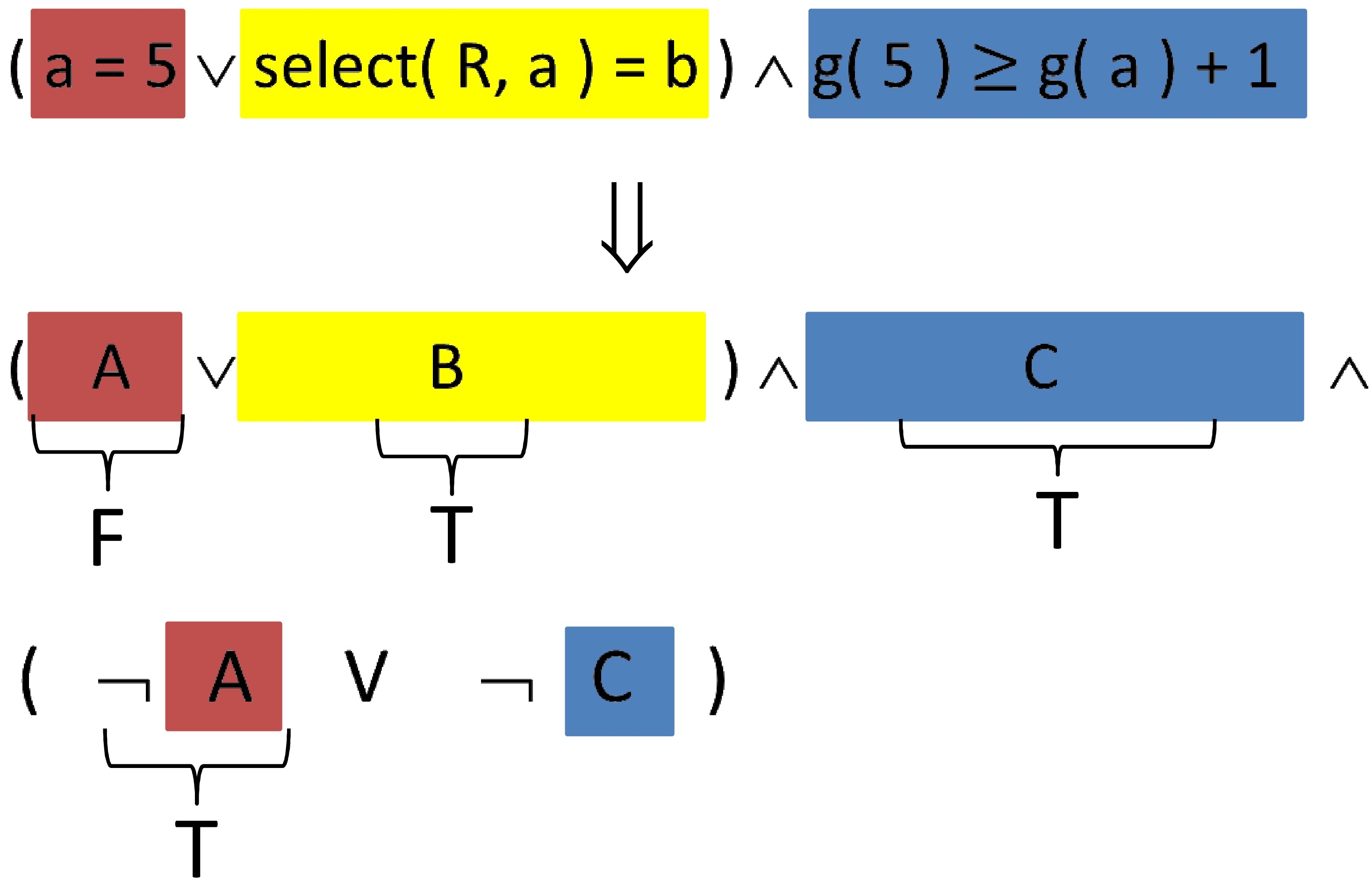
$$(a = 5 \lor \text{select}(R, a) = b) \land g(5) \geq g(a) + 1$$

$$\Downarrow$$

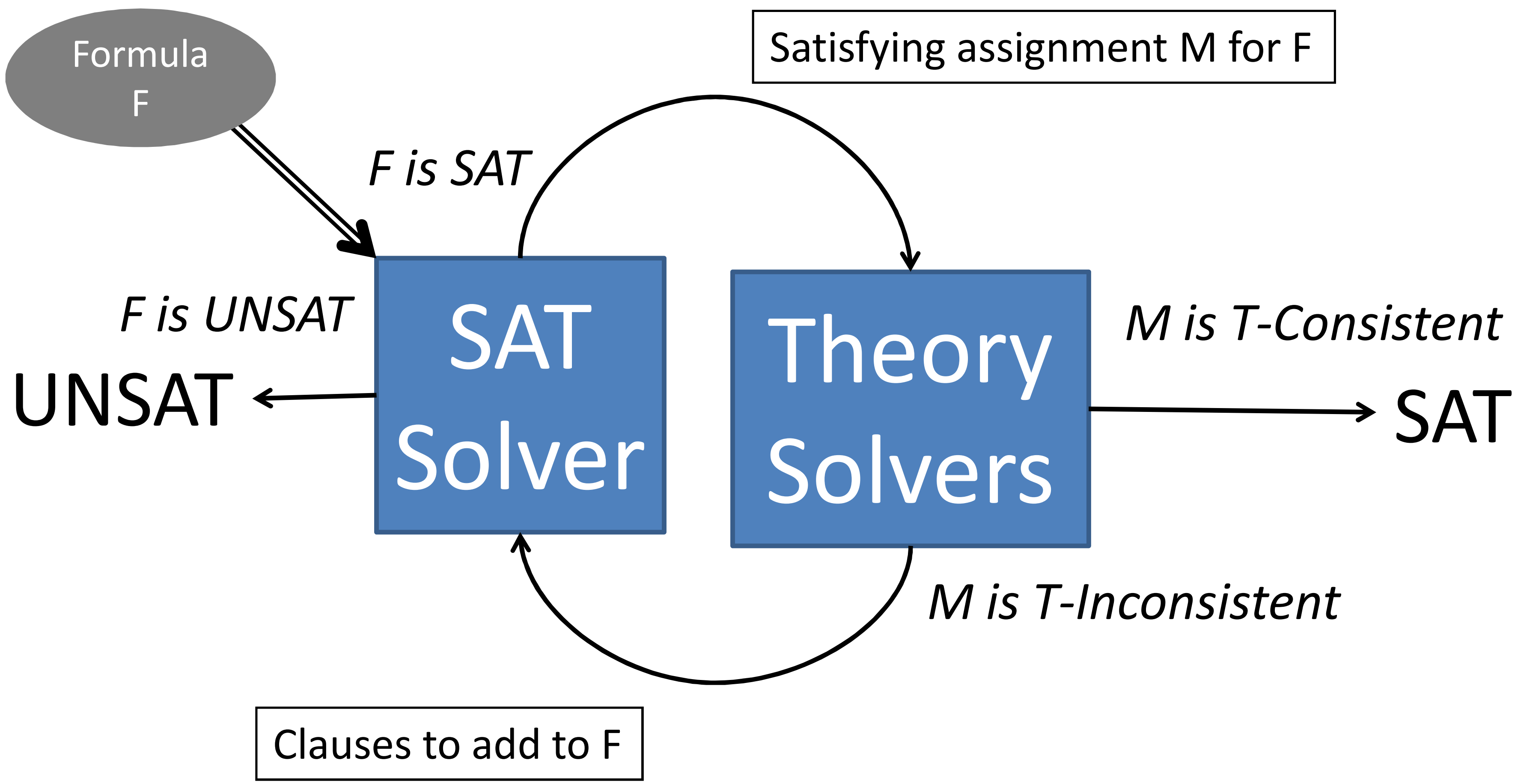
$$(\underbrace{A}_{T} \lor B) \land \underbrace{C}_{T}$$

- *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
  - Must add additional clause:

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


(a = 5 ∨ select(R, a) = b) ∧ g(5) ≥ g(a) + 1
⇓
(A ∨ B) ∧ C ∧
F
T
T
(¬ A V ¬ C)
T

DPLL(T) Architecture



Why Quantifiers?

- Quantifiers exist in verification conditions:

Definitions

S, P, R : type
null : R
valid: Array(R, Bool)
count: Array(R, Int)
ref: Array(P, R)
empty : S
mem : (S, P) -> Bool
add : (S, P) -> S

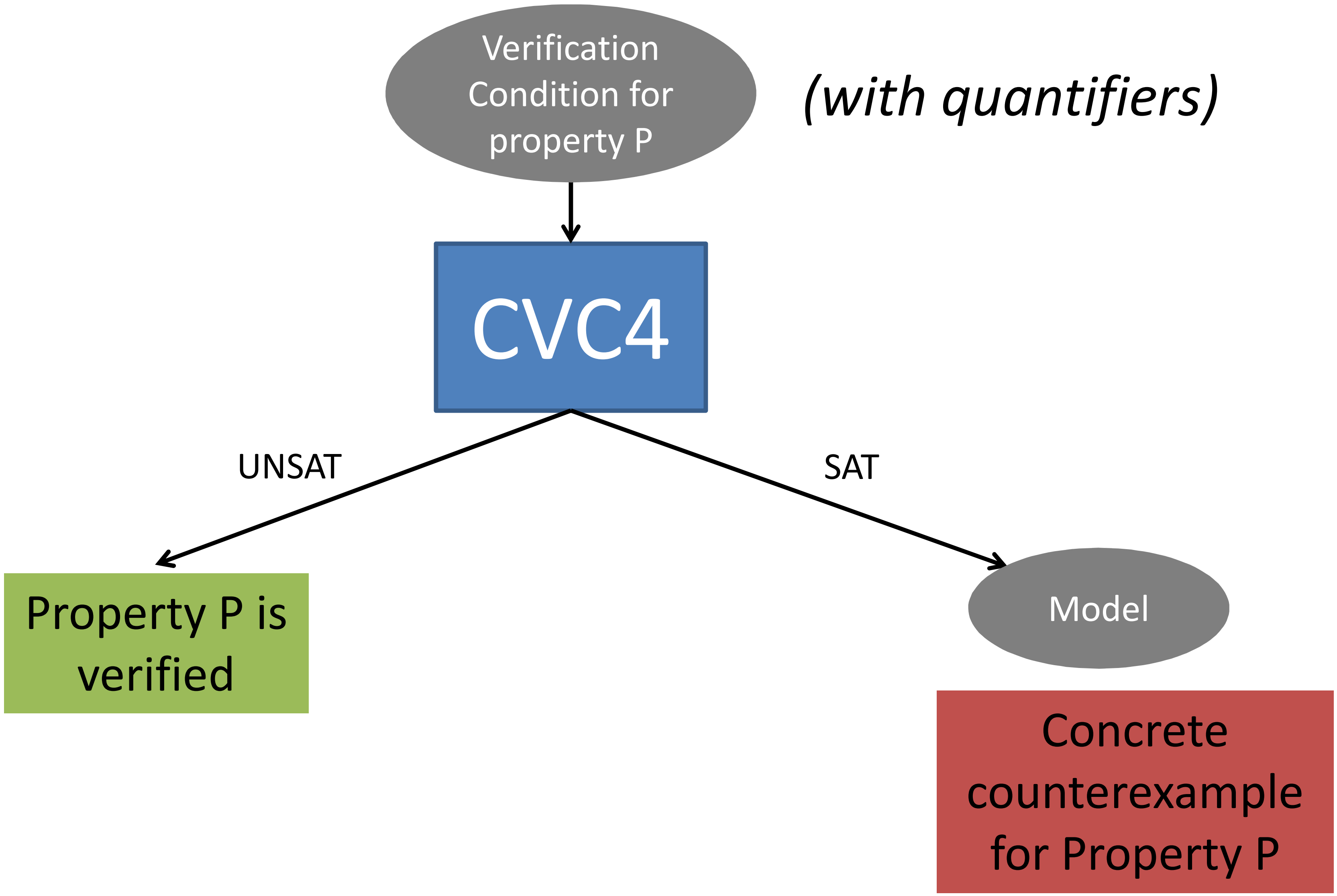
Axioms

$\forall x : R.\ count[x] > 0 \Rightarrow valid[\ x\ ]$
$\forall x : P.\ \neg\ mem(\ empty, x\ )$
$\forall x : S, y, z : P.\ mem(\ add(\ x, y\ ), z\ ) \Rightarrow (\ z = y \vee mem(\ x, z\ )\ )$
$\forall x : S, y, z : P.\ mem(\ remove(\ x, y\ ), z\ ) \Rightarrow (\ z \neq y \wedge mem(\ x, z\ )\ )$
…

$\neg$ (… $\forall$x. (ref[x] != null => valid[ref[x]]) …)

Property to verify

Handling Verification Conditions



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

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

$$\Downarrow$$

$$\underbrace{A}_{T} \wedge (\underbrace{B}_{T} \vee C)$$

- 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 values

Quantifier Instantiation

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

$$\ldots,\ f(2) = 5,\ \ldots.$$

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

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

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

$$\ldots.$$

G

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

instantiate

Q

$\Rightarrow$ *Sound but incomplete procedure*

Quantifiers in SMT

- 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?

Current Approaches for Quantifiers

- *Most widely used*: Pattern-Based Instantiation
  - Determine instantiations heuristically
    - Based on finding ground terms in G with same shape as terms in Q

$$\ldots, b \neq a, \boxed{f(a)} = b, \ldots, \forall x. \boxed{f(x)} = x$$

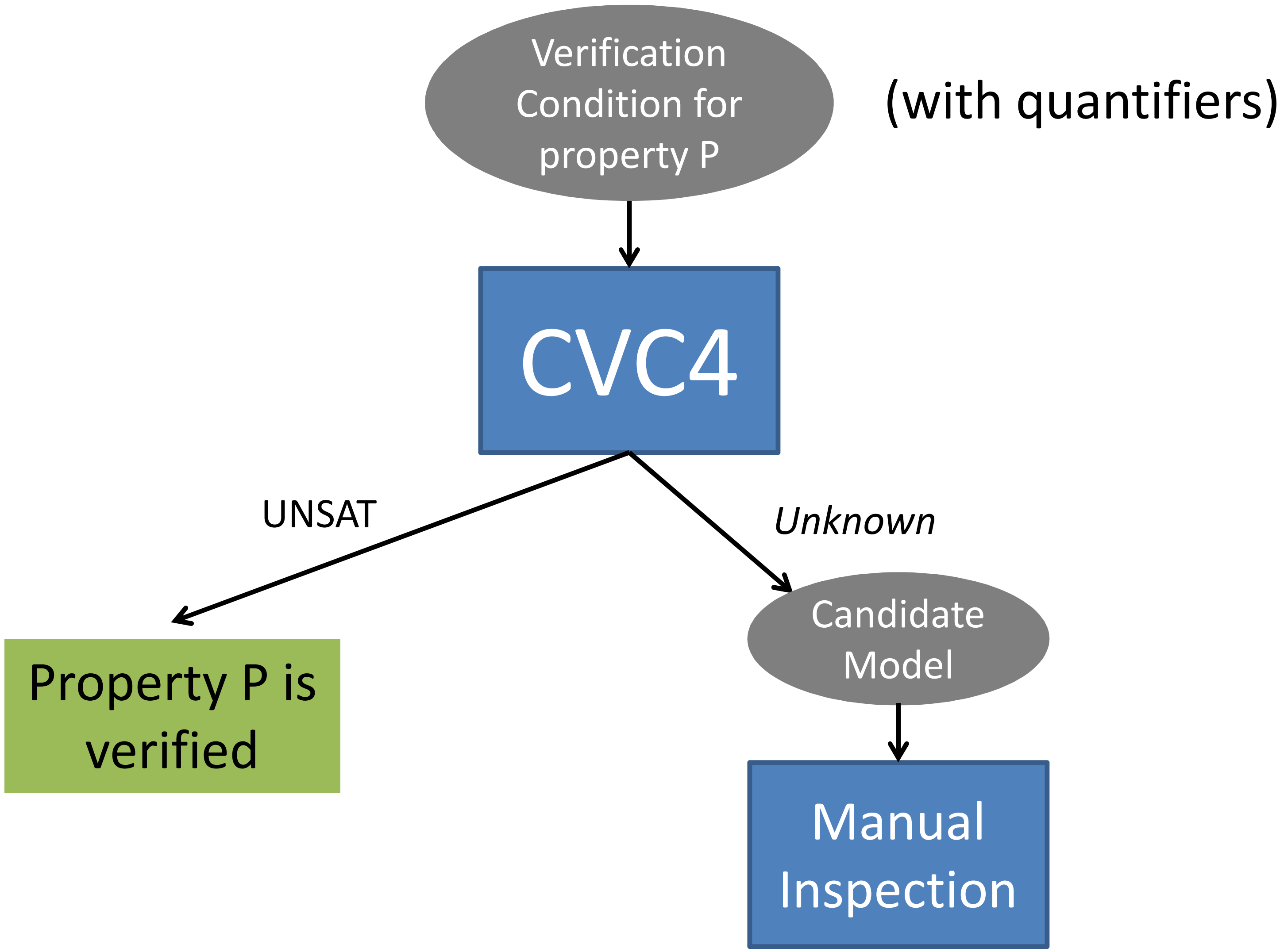
matches

$\Rightarrow$ instantiate [a/x]: $f(a) = a$,

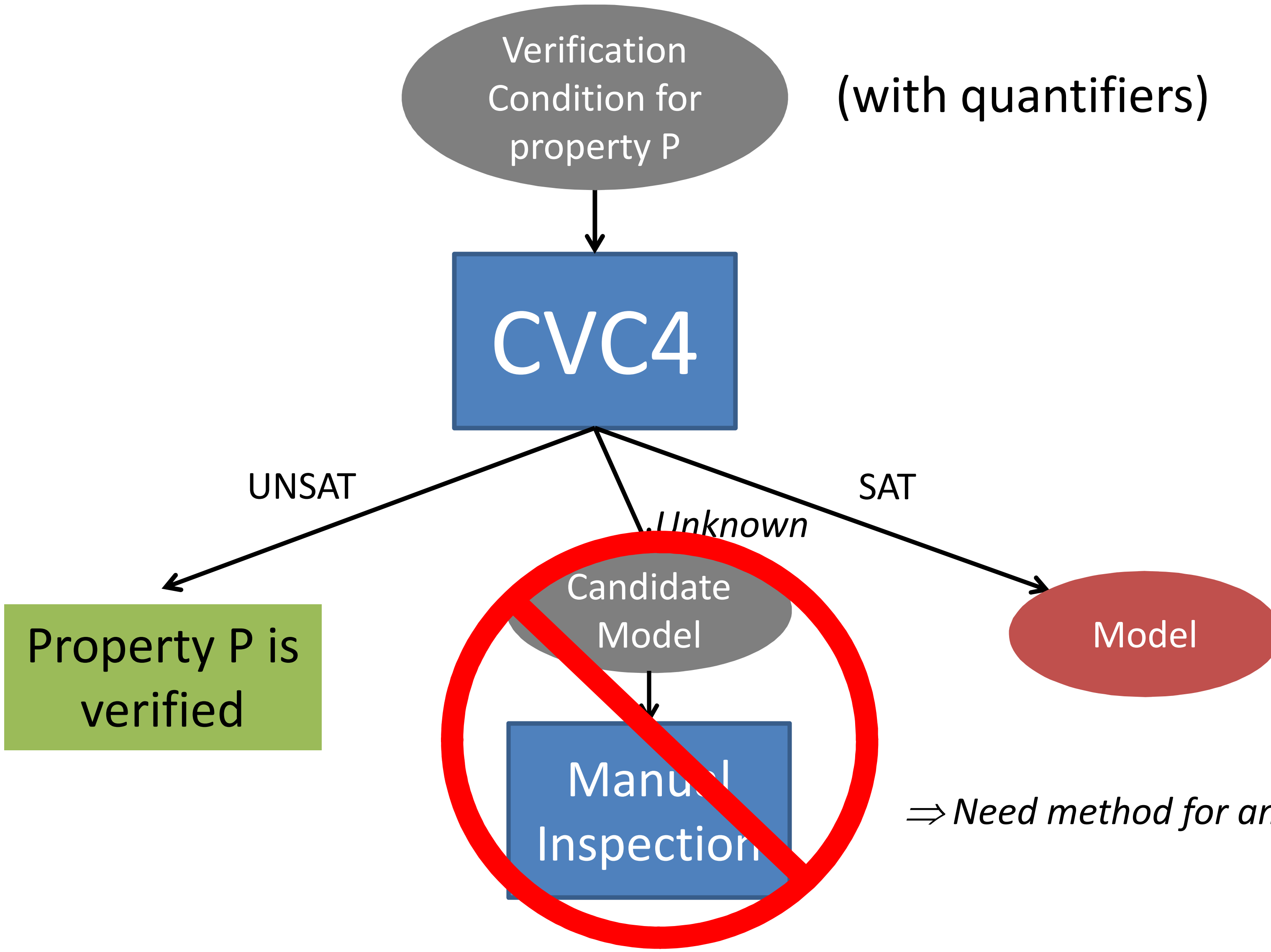
T-inconsistent : $a = f(a) = b \neq a$

- *However, If pattern matching fails, must answer “unknown”*

Handling Verification Conditions



Handling Verification Conditions



⇒ *Need method for answering SAT*

Finite Model Finding

- Method to answer SAT in presence of quantifiers
- Given set of literals (G, Q):
  - Find a "smallest" model for G
  - Try *every* instantiation of Q in the model
    - Feasible if the domain we need to consider is *finite*
  - If every instantiation true in model, answer SAT

Finite Model Finding (for EUF)

- For now, consider quantifiers over uninterpreted sorts:

  $\forall x : S. \neg \text{mem}(\text{empty}, x)$

  for all x of type S…

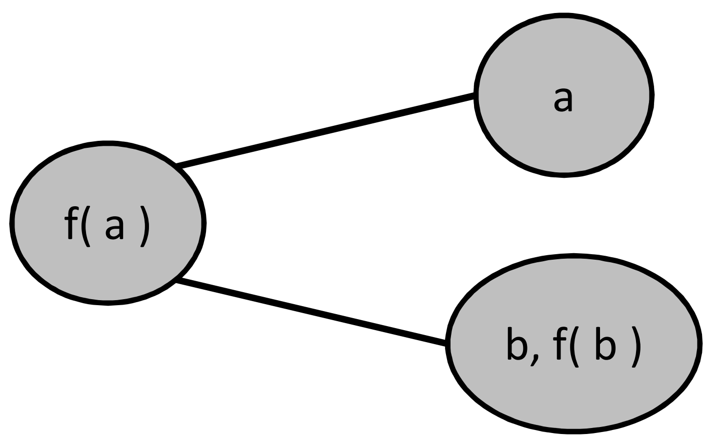
  – Example uses:
    - Values, Addresses, Processes, Resources, Sets, …

Finding Small Models

- What is a small model?
  - SMT solvers maintain a set of equivalence classes internally
  - "Smallest" model for sort S means:
    - Fewest # equivalence classes of sort S
- To find small models:
  - Impose *cardinality constraints* on (uninterpreted) sorts S
    - Predicate $C_{S, k}$, meaning "sort S has at most k equivalence classes"
  - Try to find models of size 1, 2, 3, … etc.
- What this requires:
  - Control to DPLL(T) search for postulating cardinalities
  - Solver for UF + cardinality constraints

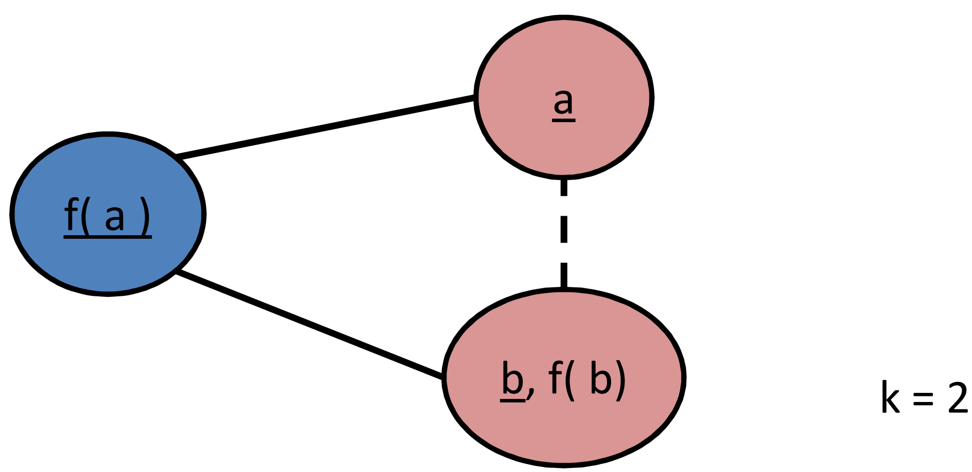
UF + Cardinality Constraints

- Given (G, $C_{S,k}$)
  - Set of ground constraints G over sort S
  - Cardinality constraint $C_{S,k}$
- Maintain disequality graph D_S = (V, E)
  - V are equivalence classes of sort S
  - E are disequalities between terms of sort S
- D_S induced by asserted set of literals in G
  - So, $f(a) \neq a$, $f(a) \neq b$, $b = f(b)$ becomes:



UF + Cardinality Constraints

- We are interested in whether D_S is k-colorable
  - If *no*, then we have a conflict ($F \Rightarrow \neg C_{S,k}$)
    - where F is explanation of sub-graph of D_S that is not k-colorable
  - If *yes*, then we merge nodes with same color

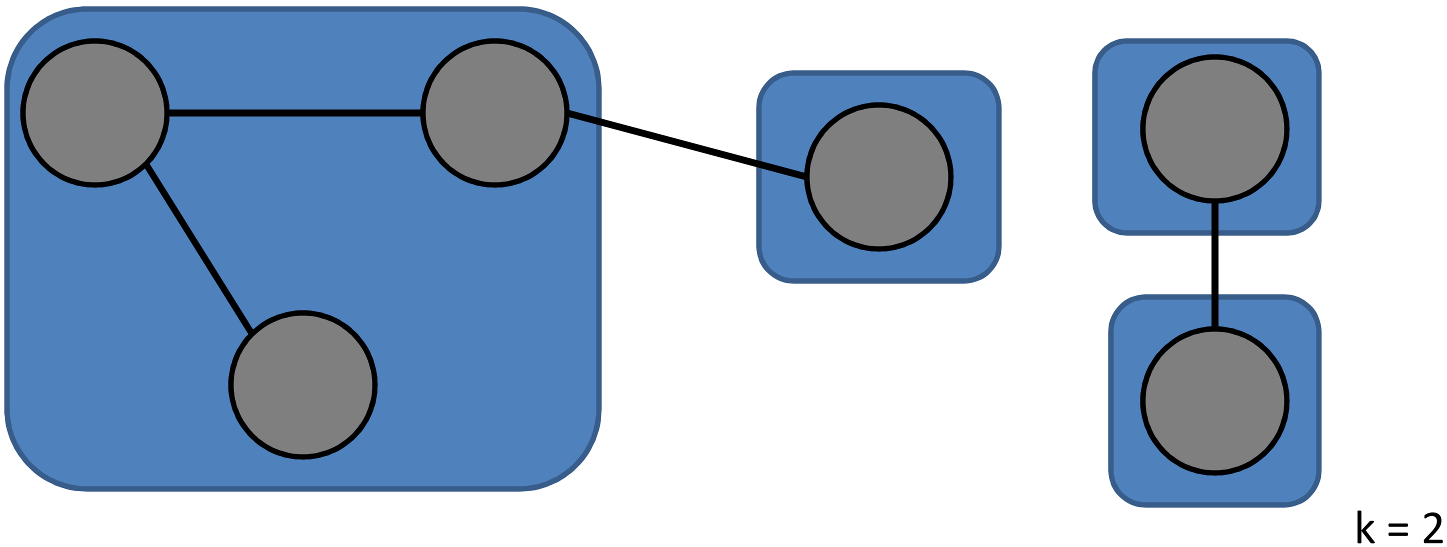


UF + Cardinality Constraints

- Challenges:
  - Determining k-colorability is NP-hard
  - Analysis must be incremental
- Solution: use a *region-based approach*
  - Partition nodes in *regions* with high edge density
  - *Quickly* recognize when D_S is *not* k-colorable
  - Helpful for suggesting relevant nodes to merge

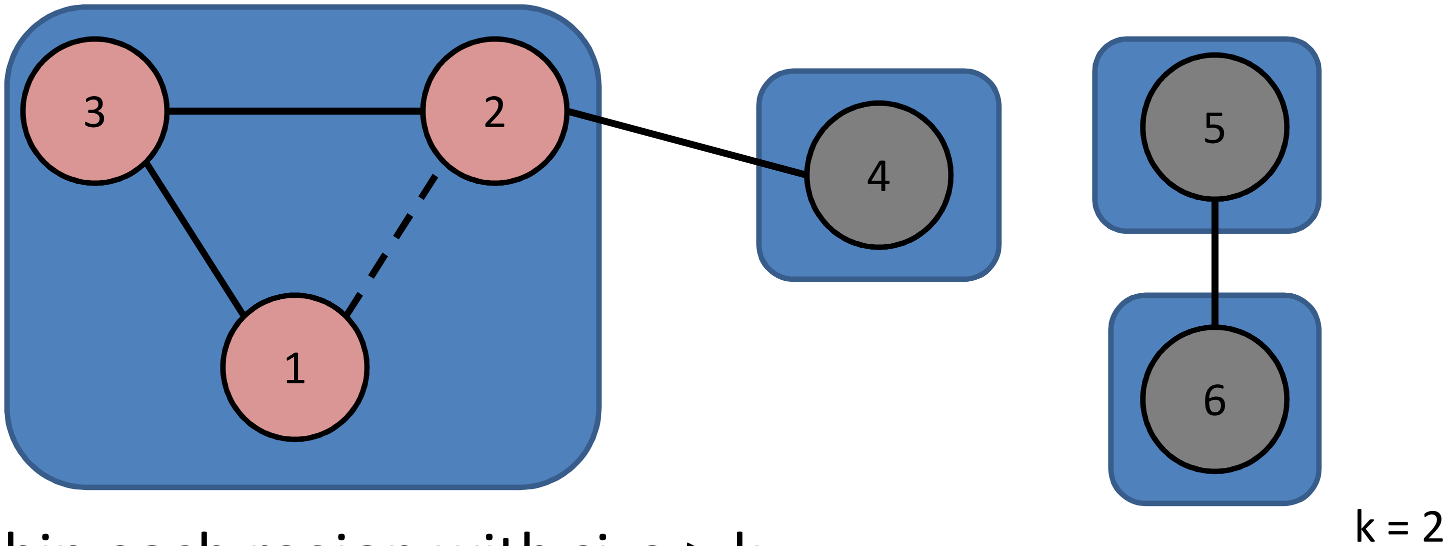
Region-Based Approach

- Partition nodes V of D_S into *regions*



- Invariant: need only search for (k+1)-cliques local to regions
- Region can be ignored if it has ≤ k terms

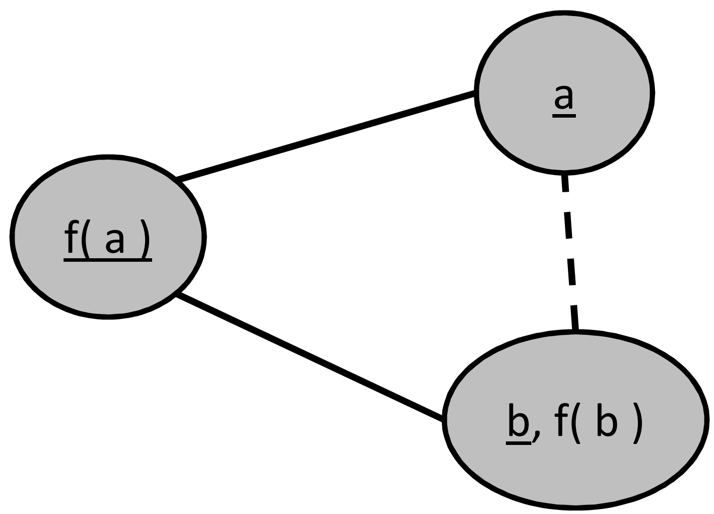
Region-Based Approach



- Within each region with size > k:
  - Maintain a watched set N of k+1 nodes
  - Record pairs of nodes in N that are not linked
    - If this set is empty, N is a clique $\Rightarrow$ report conflict
    - Otherwise, merge unlinked nodes in N

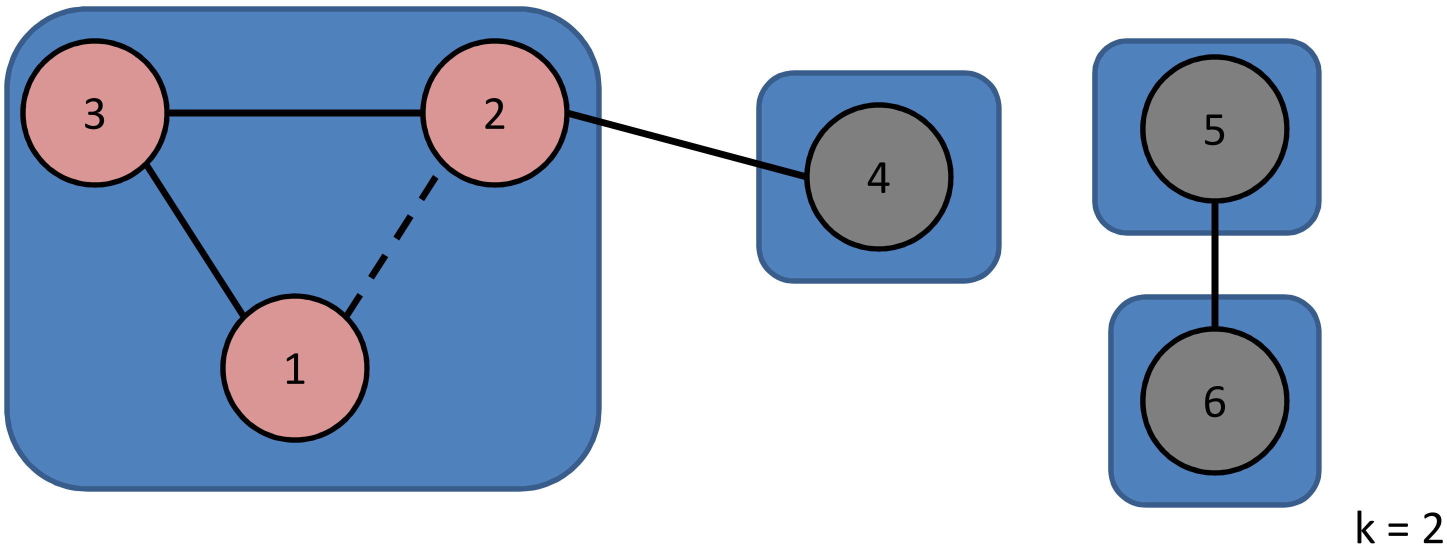
Region-Based Approach

- Merging nodes may lead to T-inconsistency
  - For example, congruence axioms in UF:



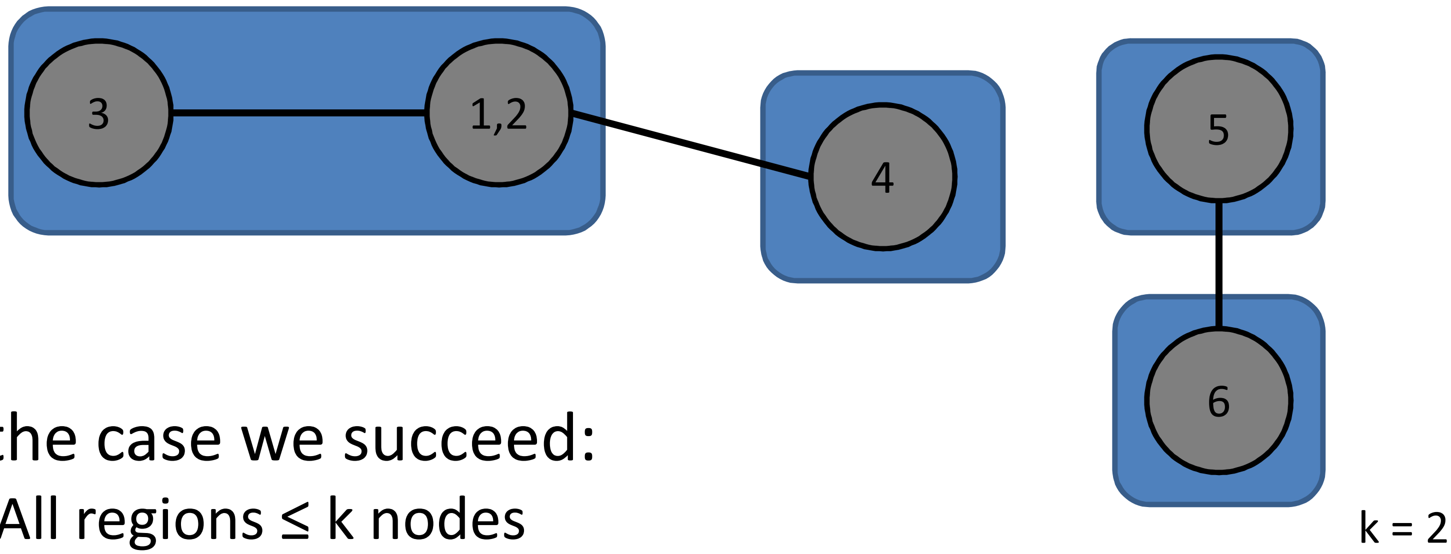
$\Rightarrow$ *In this case, we cannot merge a = b*

Region-Based Approach



- Merging nodes 1 and 2 may:
  - Lead to T-inconsistency
  - Lead to a cardinality conflict (force a clique), or
  - *Succeed*

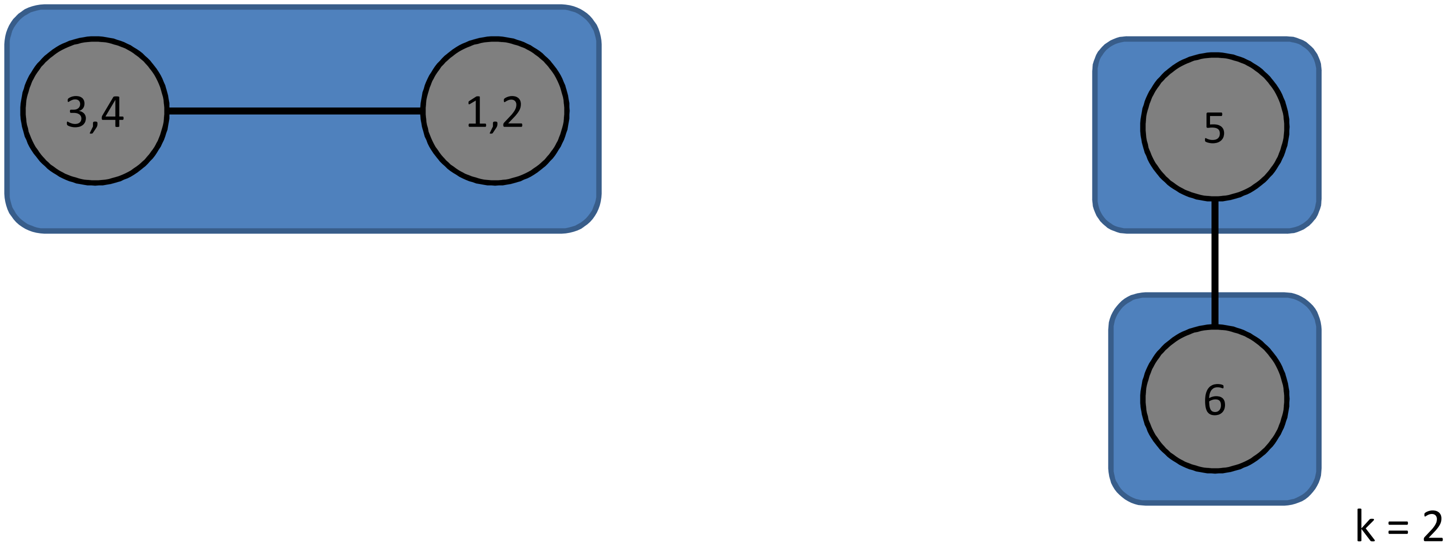
Region-Based Approach



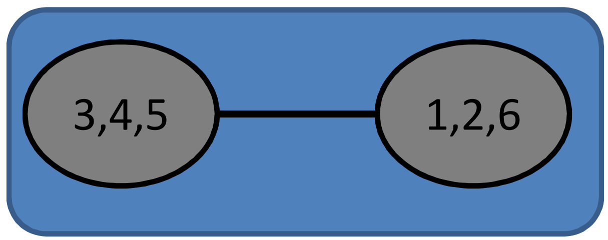
- In the case we succeed:
  - All regions ≤ k nodes
    - We are ensured k-colorability
  - However, still unsure a model of size k exists
    - Due to possible T-inconsistency

  ⇒ *Must shrink model explicitly*

Region-Based Approach



Region-Based Approach



k = 2

- Merge until we have until ≤ k nodes overall

  $\Rightarrow$ Guaranteed a model of size k exists

Finite Model Finding

- Given set of literals (G, Q):
  1. Find smallest model M for G
     - i.e. M with smallest # of equivalence classes
  2. Instantiate Q with all combinations of terms in M
  3. If all instantiations are true in model, and model size did not grow, then answer SAT

Finite Model Finding : Example

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

1. Smallest model for G, size 2 : { $\underline{a}$ }, { $\underline{b}$, c }
2. Instantiate Q with [a/x, b/x]:
   - f(a) = a, f(b) = b added to G
3. After instantiation : { $\underline{a}$, f(a) }, { $\underline{b}$, c, f(b) }
   - All instantiations are true, model size did not grow

$\Rightarrow$ 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 $O(k^n)$ 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/eliminate redundant instantiations
      - *Use Model-Based Quantifier Instantiation* [Ge/deMoura 09]

Model-Based Quantifier Instantiation (MBQI)

- Idea : Do not consider instantiations that are already true in current model
- Strategy for (G, Q):

1. Build model M for G, consisting of:
   - Set of representatives R
   - Interpretation for all symbols in Q
2. For all quantifiers $\forall$x. F[x] in Q:
   - Construct F^M[x] according to interpretations in M
   - Add instantiations F[t] to G, for all t $\in$ R such that:
     - F^M[t] is not true in M

MBQI : Example

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

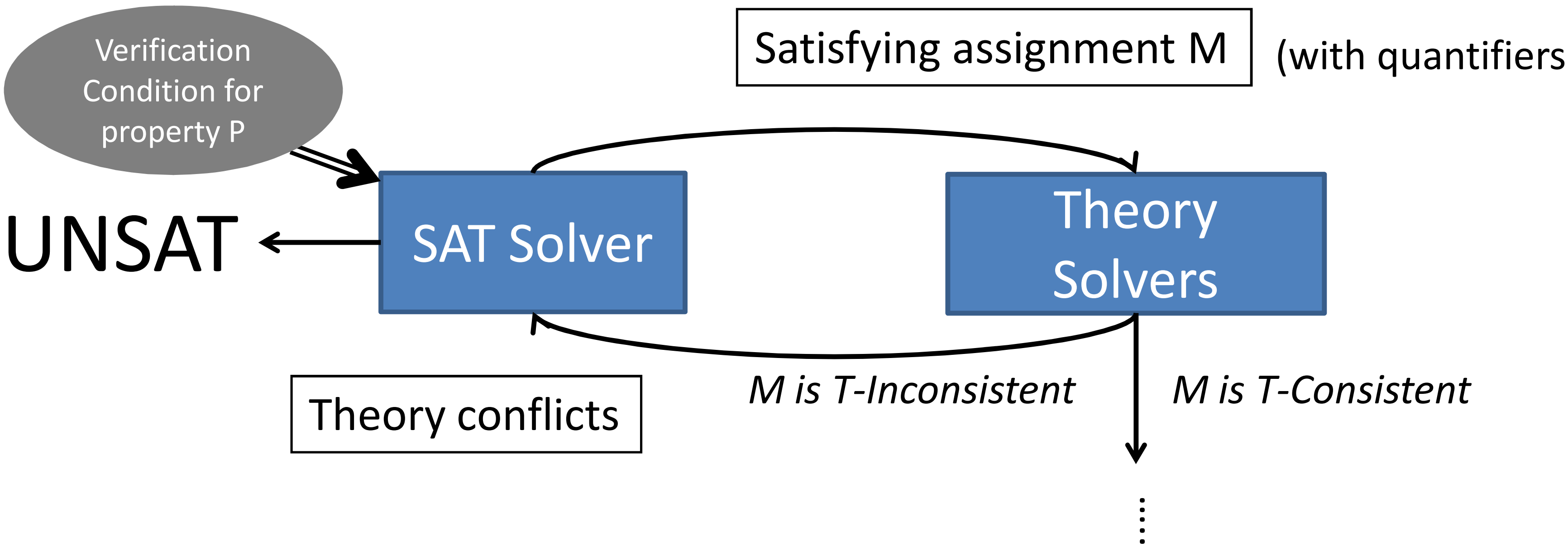
Find model M : $\{a\}, \{b\},$

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

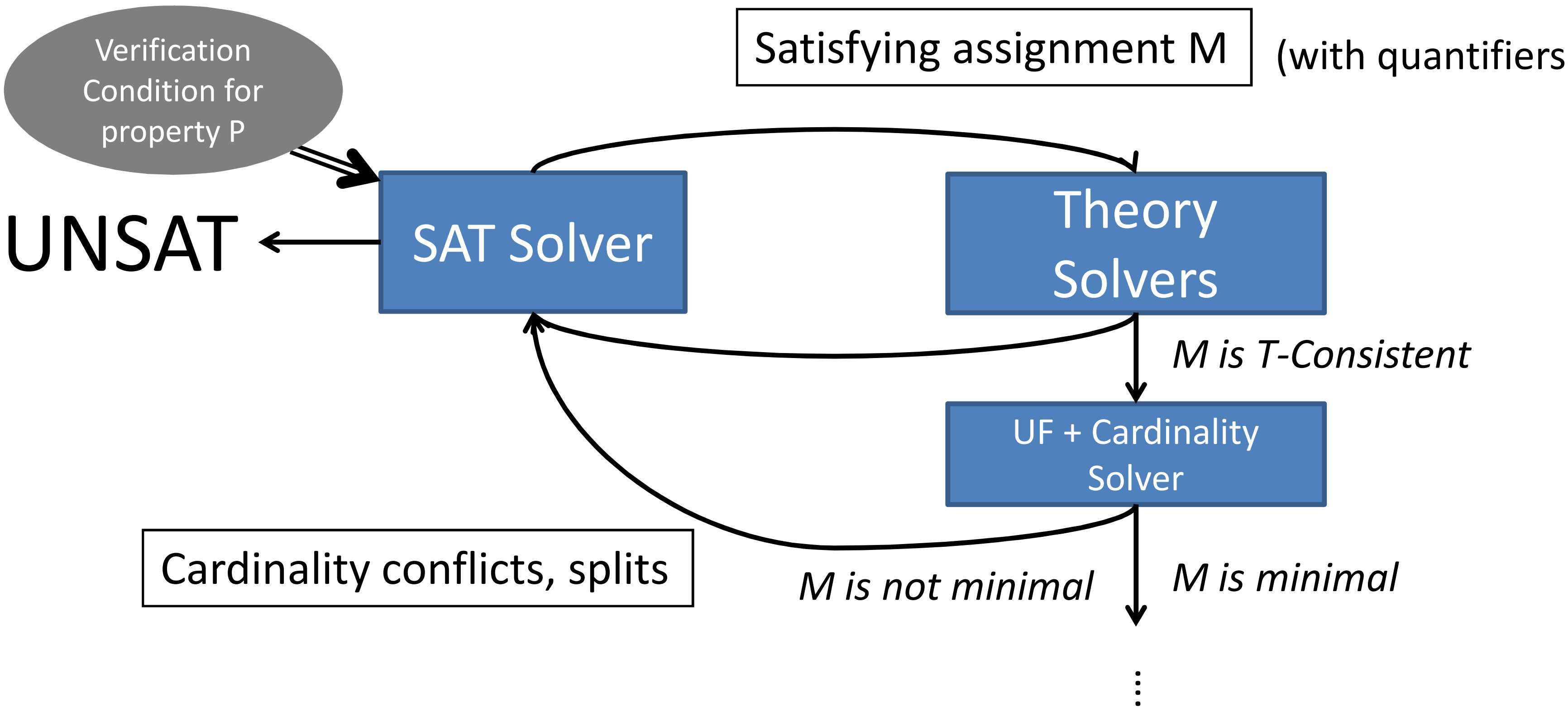
$$\neg P^M(x, b) \equiv \neg(x{=}a \wedge b{=}a) \equiv \text{true}$$

$\Rightarrow$ *All instantiations of Q are true in M*

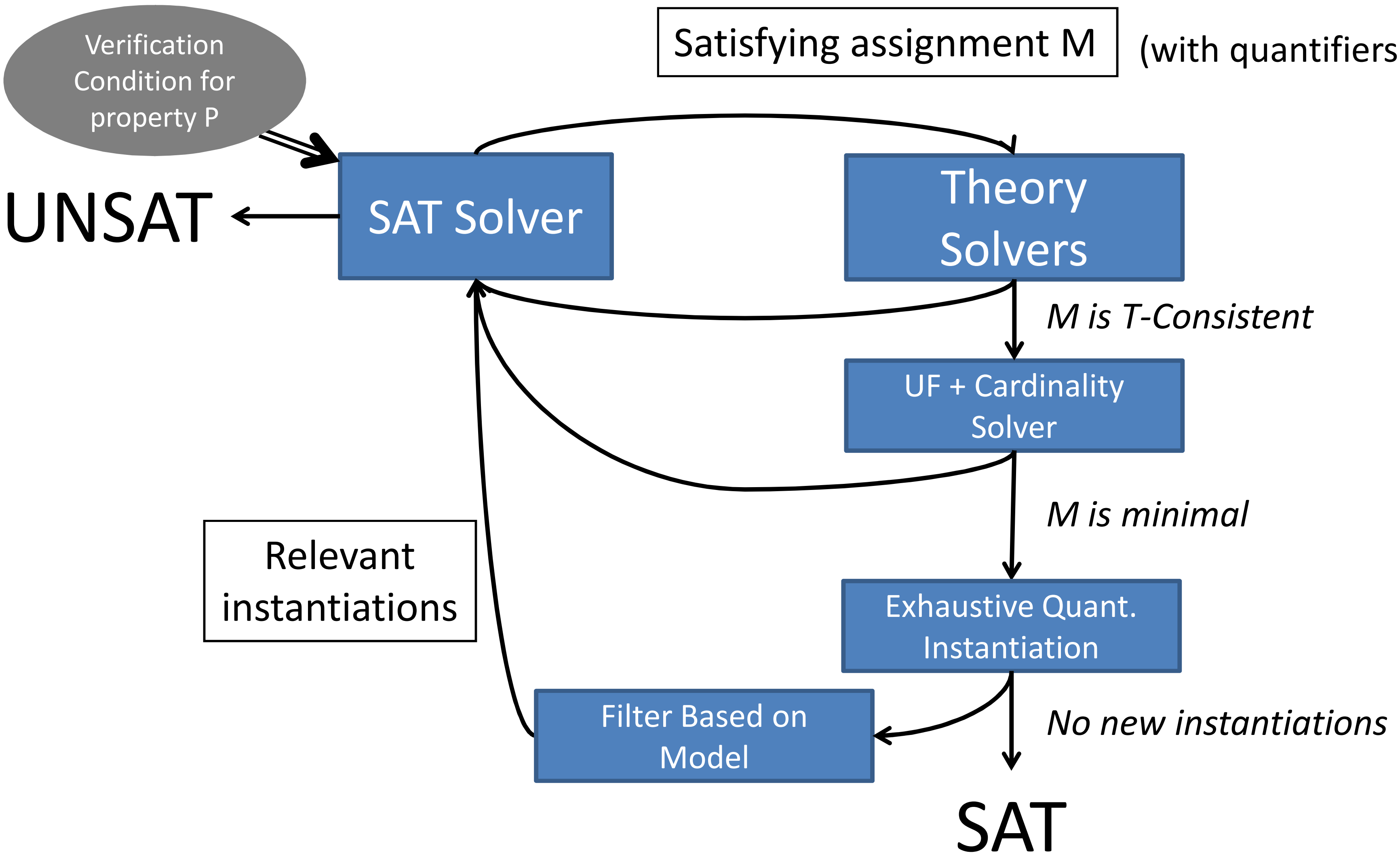
Anatomy of Finite Model Finding



Anatomy of Finite Model Finding



Anatomy of Finite Model Finding



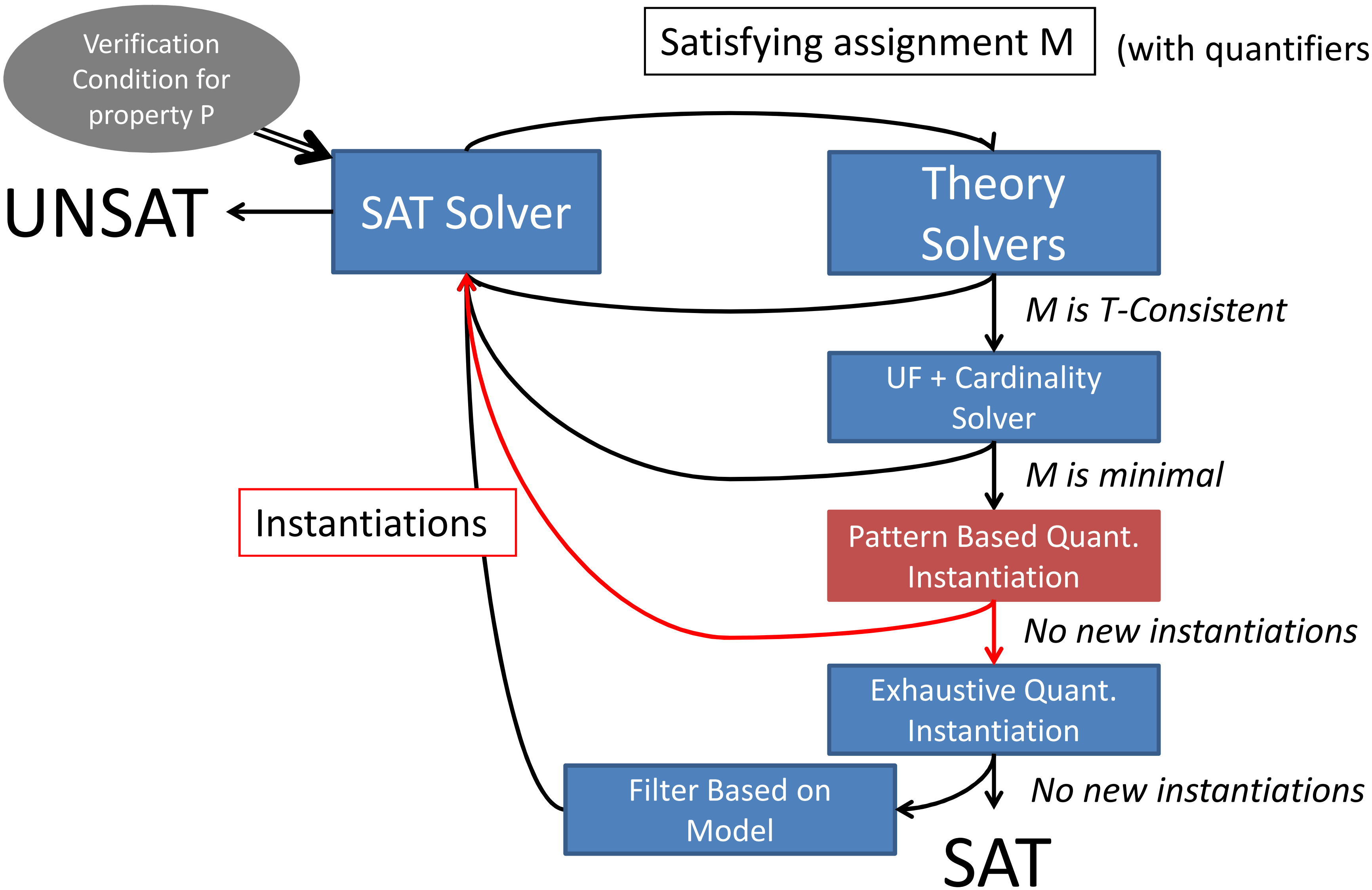
Other Instantiation Strategies

- Sometimes, # instantiations is still very large
- Other strategies:
  - Non-exhaustive instantiation:
    - Only add small # instantiations each round
      - Pro: (possibly) less instantiations added
      - Con: usually slower convergence to model
  - Exhaustive instantiation restricted to non-axioms
    - Rely on other methods for instantiating axioms, e.g…
  - Pattern-Based instantiation

FMF + Pattern-Based Instantiation

- Idea:
  - First see if instantiations based on patterns exist
  - If not, resort to exhaustive instantiation
- May lead to:
  - Answering UNSAT more often
    - Discover easy conflicts, if they exist
  - Arriving at model faster
    - Instantiations rule out spurious models

FMF + Pattern-Based Instantiation



Experimental Results

- DVF Benchmarks
  - Taken from real DVF examples
  - Both SAT/UNSAT benchmarks
    - SAT benchmarks generated by removing necessary pf assumptions
  - Many theories: UF, arithmetic, arrays, datatypes
- TPTP Benchmarks
  - Taken from ATP community
  - Heavily quantified
  - Unsorted logic

Results: DVF

UNSAT	german	refcount	agree	apg	bmk	Total
cvc4	**145**	**40**	600	**304**	**244**	1333
cvc4+fmf	**145**	**40**	**604**	294	236	1319
z3	**145**	**40**	**604**	**304**	**244**	**1337**
	145	40	604	304	244	1337

SAT	german	refcount	agree	apg	bmk	Total
cvc4	2	0	0	0	0	2
cvc4+fmf	**45**	**6**	**62**	**16**	**36**	**165**
z3	**45**	1	0	0	0	46
	45	6	62	19	37	169

- 60 second timeout

Results per Inst Strategy (cvc4+fmf)

UNSAT	german	refcount	agree	apg	bmk	Total
naïve	**145**	**40**	583	272	222	1262
mbqi	**145**	**40**	579	292	**238**	1294
mbqi+pattern-based inst	**145**	**40**	**604**	**294**	236	**1319**
	145	40	604	304	244	1337

SAT	german	refcount	agree	apg	bmk	Total
naïve	**45**	**6**	**62**	**18**	33	164
mbqi	**45**	**6**	60	15	**36**	162
mbqi+pattern-based inst	**45**	**6**	**62**	16	**36**	**165**
	45	6	62	19	37	169

$\Rightarrow$ *Each SAT benchmark is solved by at least one configuration*

Example Model from CVC4

Information regarding sorts

```
; cardinality of R is 2
(declare-sort R 0)
; cardinality of P is 1
(declare-sort P 0)
; cardinality of S is 2
(declare-sort S 0)
```

Definitions of functions and predicates in model

```
(define-fun null () R r2)
(define-fun empty () S s1)
(define-fun mem ((x1 P) (x2 S)) BOOL
        (ite (= x1 p1) (ite (= x2 s2) Truth Falsity) Falsity))
(define-fun add ((x1 P) (x2 S)) S s2)
(define-fun remove ((x1 P) (x2 S)) S s1)
(define-fun cardinality ((x1 S)) Int (ite (= x1 s1) 0 1))
(define-fun count () (Array R Int) (store count r1 0))
(define-fun ref () (Array P R) (store ref p1 r1))
(define-fun valid () (Array R BOOL) (store valid r1 Truth))
(define-fun destroyr () R r1)
(define-fun valid1 () (Array R BOOL) (store valid r1 Truth))
```

Results: TPTP

- 10 second timeout
- 11613 UNSAT benchmarks:
  - z3: **5471** solved
  - cvc4: 4868 solved
  - cvc4+fmf: 2246 solved, but orthogonal
    - 288 solved that cvc4 w/o finite model finding cannot
  - Either cvc4 or cvc4+fmf: 5158 solved
- 1933 SAT benchmarks:
  - z3: 866 solved
  - cvc4+fmf: **920** solved
- Model-Based Quantifier Instantiation is essential

Summary

- Finite model finding in CVC4
  - Uses solver for UF + cardinality constraints
  - Finds minimal models for ground constraints
  - Uses exhaustive instantiation to test models
    - Instantiations filtered by MBQI
  - Optionally, uses pattern-based instantiation

Conclusions

- Finite Model Finding:
  - Practical approach for SMT + quantifiers
  - Can answer SAT quickly
    - Generate simple counterexamples for DVF
  - Improves coverage in UNSAT cases
    - Increased ability to discharge verification conditions
  - Orthogonal to other approaches

Future Work

- Rewrite rules for axiom sets
  - Use rewriting system instead of quant instantiation
- Improvements to MBQI
  - Use ATP techniques for constructing model
  - Model interpretation for theories
    - Equality, Bit Vectors, Arithmetic, etc.
- Encode relationships between cardinalities
- Improvements for Model Output
  - Focus on human readability