Induction for SMT Solvers

Andrew Reynolds
Viktor Kuncak
EPFL, Switzerland

January 12th, 2015

Overview

- Satisfiability Modulo Theories (SMT)
- Existing SMT solvers:
  - Lack support for inductive reasoning
- Contributions :
  - Techniques for induction in SMT solvers
    - Subgoal generation
    - Leveraging theory reasoning
  - Implementation in CVC4
  - Benchmarks/Experiments

SMT Solvers

- SMT solvers:
  - Used in numerous formal methods applications:
    - Software verification, automated theorem proving
  - Determine the satisfiability of:
    - Boolean combinations of ground theory constraints
      - Linear arithmetic, BitVectors, Arrays, Datatypes, etc.
  - Have limited support for quantified formulas

SMT Solvers : Quantifiers

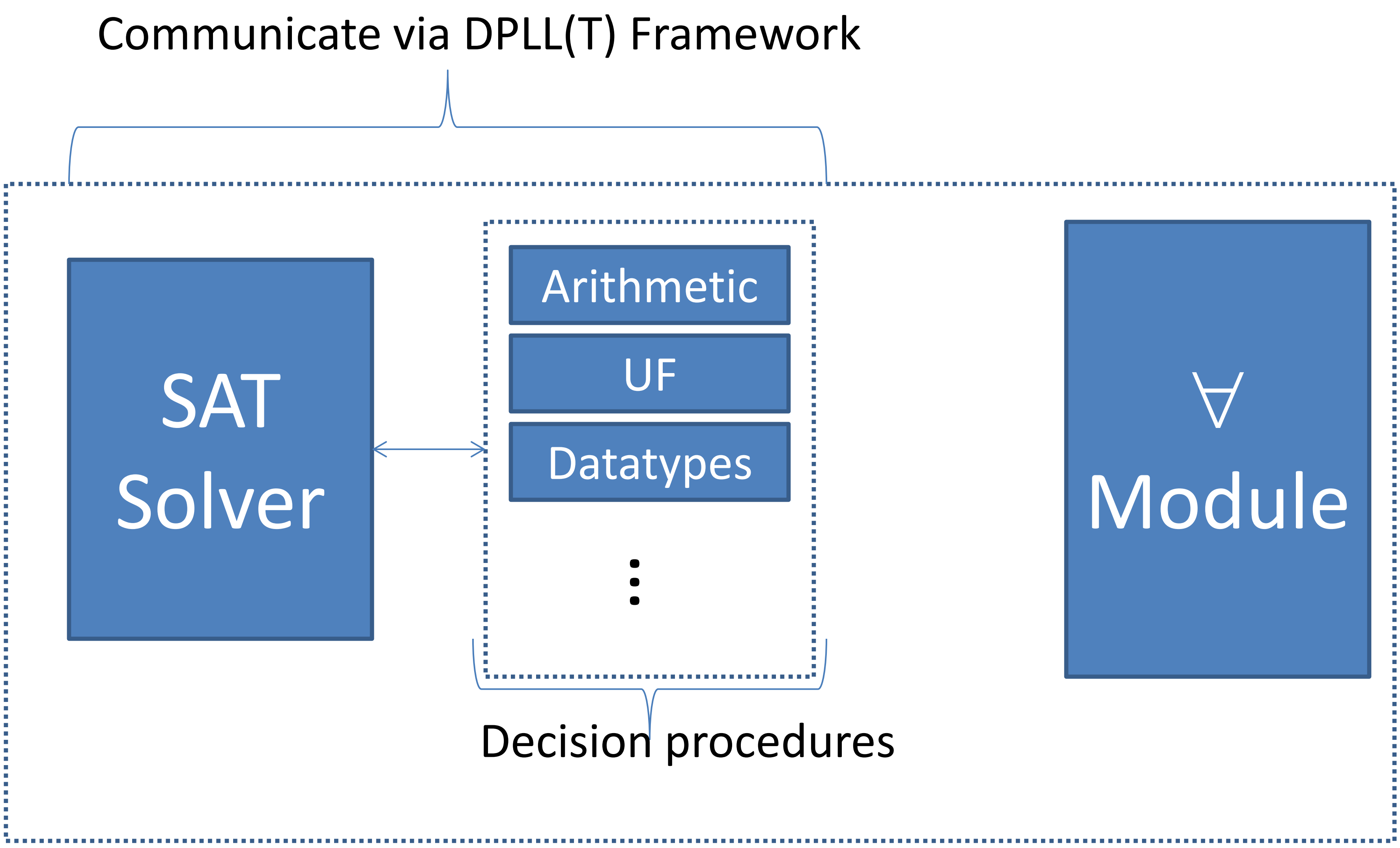
- Handle (universally) quantified formulas

$$\underbrace{\forall x : T.}_{\text{for all x of type T}} P(\, x\,)$$

$\Rightarrow$ SAT problem with $\forall$ is generally undecidable

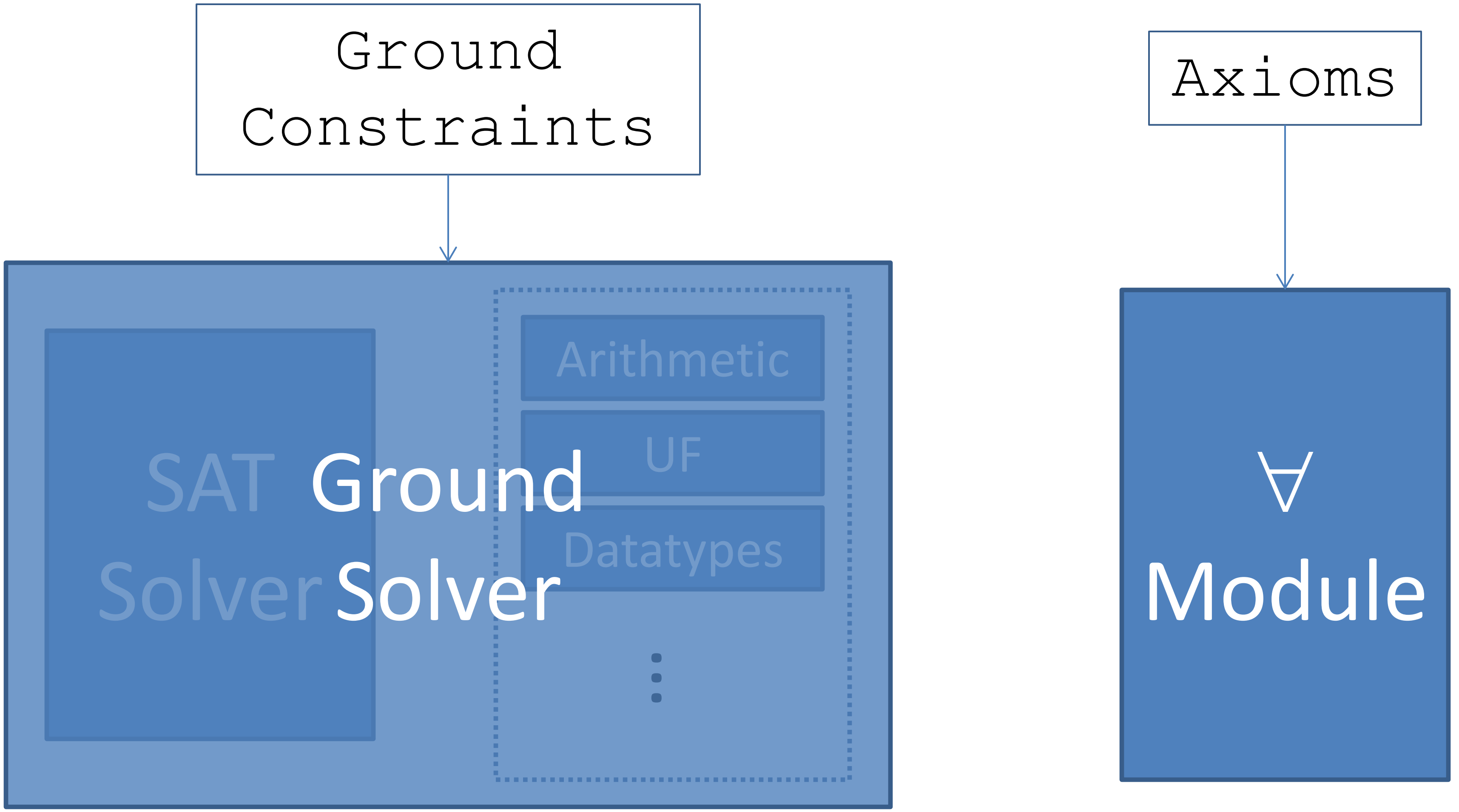
- Existing approaches for $\forall$ in SMT are:
  - Heuristic (E-matching)
  - Incomplete in general
  - Often fail on simple examples
    - Notably, for problems requiring *inductive* reasoning

SMT Solver

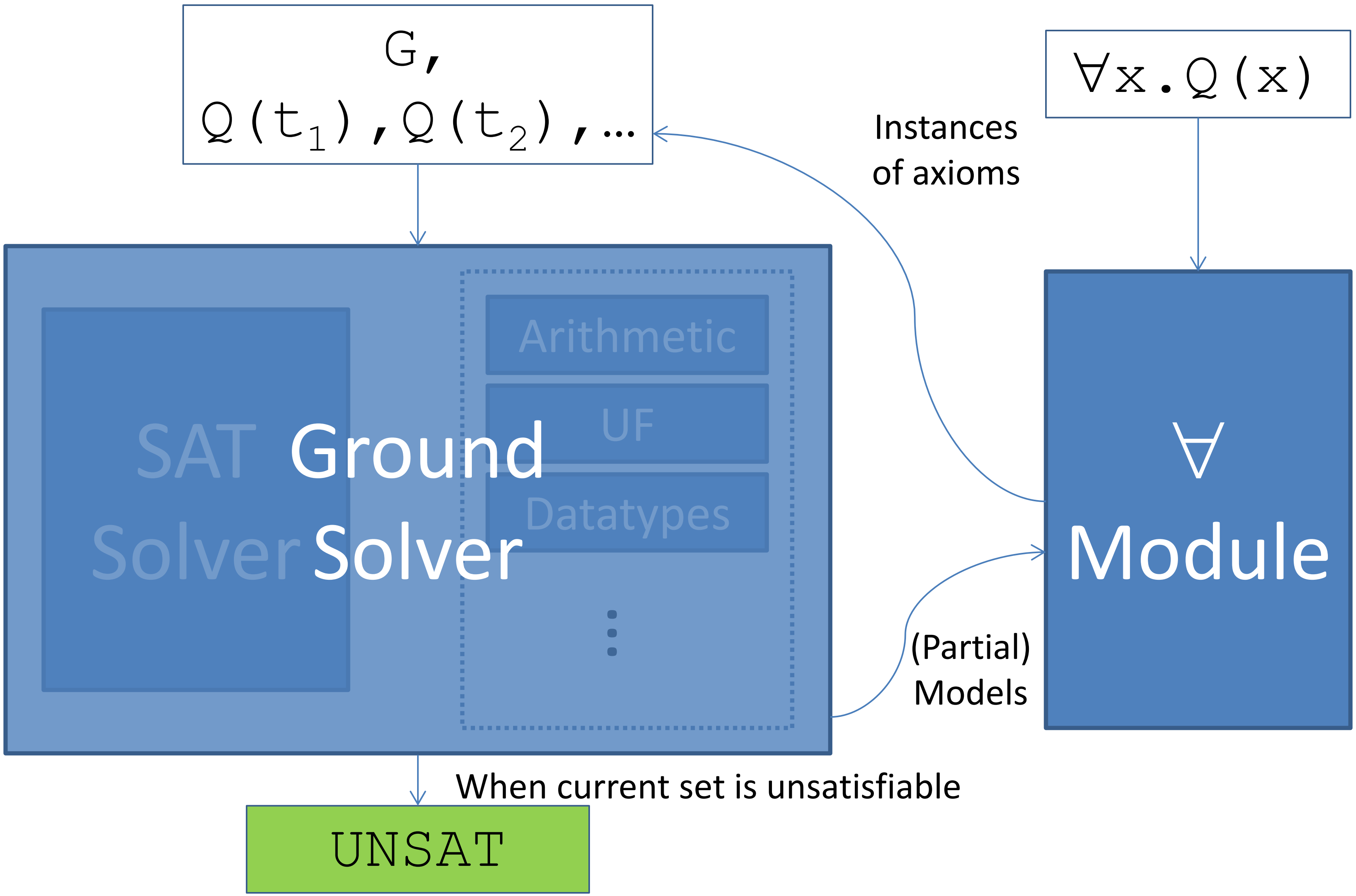


Communicate via DPLL(T) Framework
SAT Solver
Arithmetic
UF
Datatypes
⋮
Decision procedures
∀ Module

SMT Solver


Ground Constraints
Axioms
SAT Solver
Ground Solver
Arithmetic
UF
Datatypes
∀ Module

SMT Solver


G,
Q(t1),Q(t2),…
Instances
of axioms
∀x.Q(x)
SAT
Solver
Ground
Solver
Arithmetic
UF
Datatypes
⋮
∀
Module
(Partial)
Models
When current set is unsatisfiable
UNSAT

Running Example

- Datatype `List`

```
List:= cons(hd:Int,tl:List) | nil
```

- Length function `len : List -> Int`

```
len(nil)=0,
∀xy.len(cons(x,y))=1+len(y)
```

Example #1 : Ground Conjecture

```
len(nil)=0
∀xy.len(cons(x,y))=1+len(y)
```

Axioms

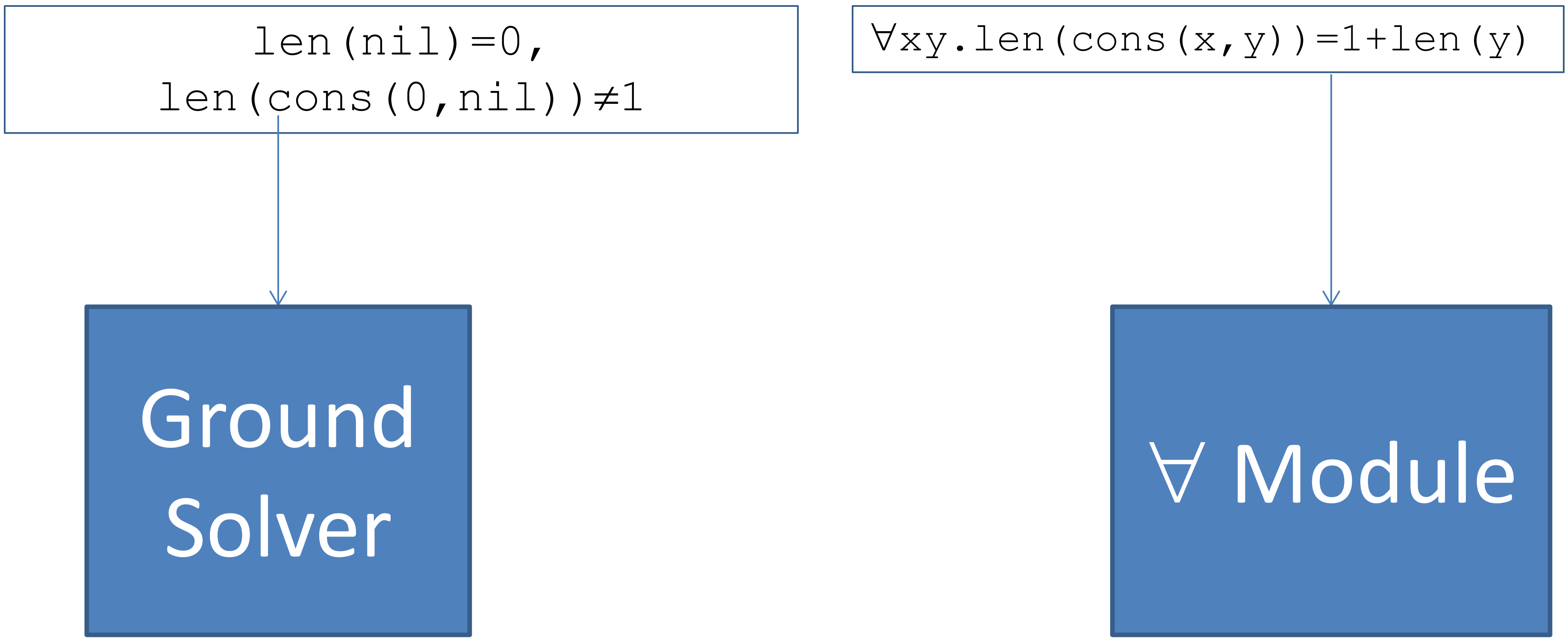
```
¬len(cons(0,nil))=1
```

(Negated) Conjecture

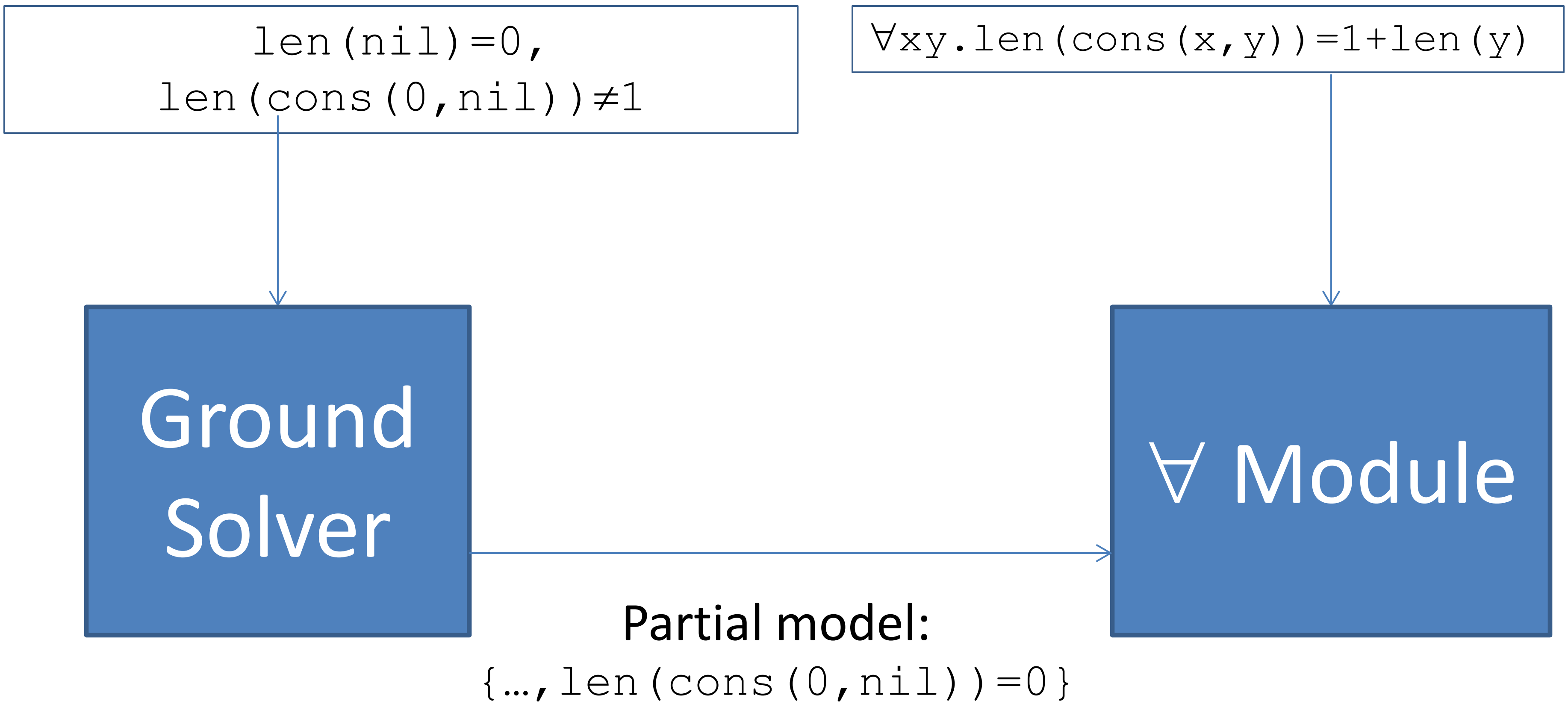
Ground Solver

∀ Module

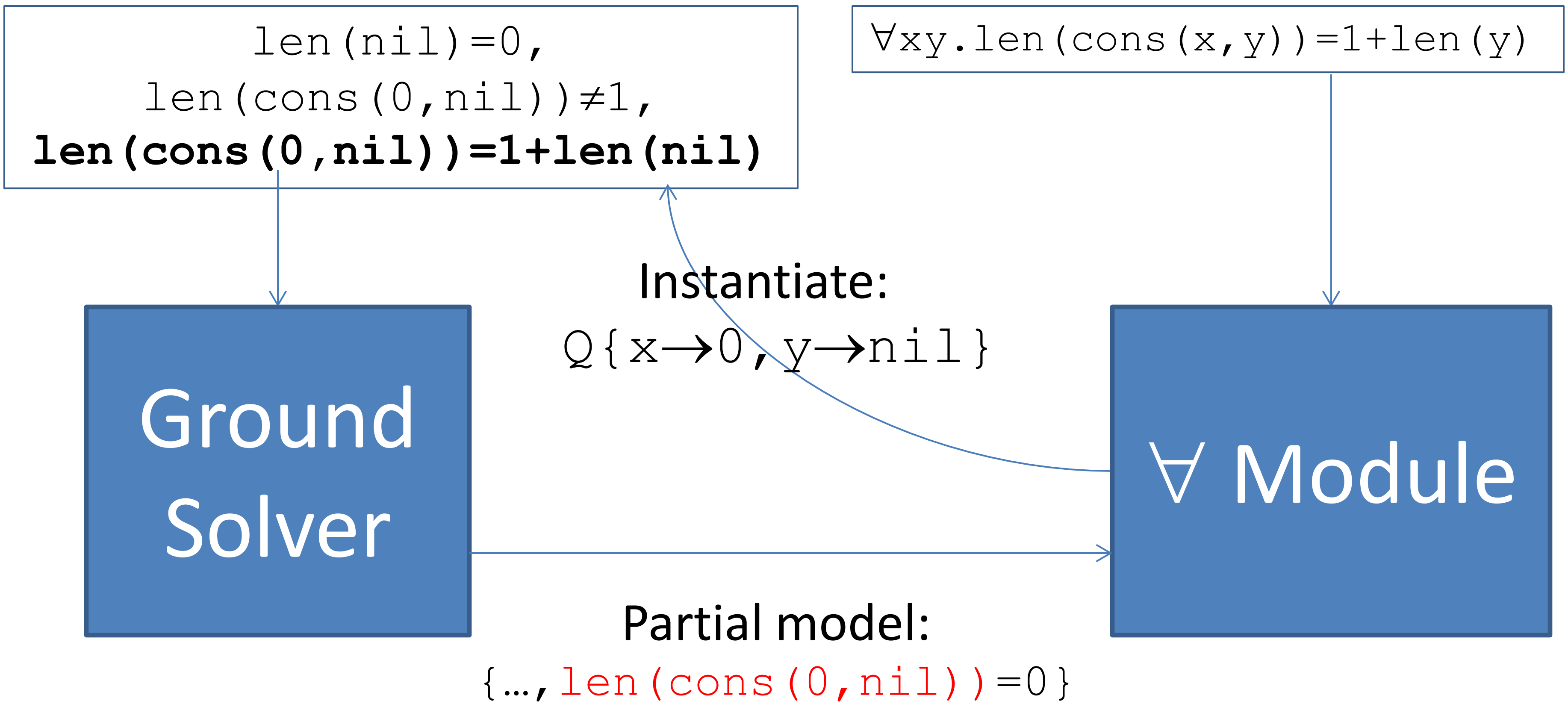
Example #1


len(nil)=0,
len(cons(0,nil))≠1
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module

Example #1


len(nil)=0,
len(cons(0,nil))≠1
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module
Partial model:
{…,len(cons(0,nil))=0}

Example #1



len(nil)=0,
len(cons(0,nil))≠1,
len(cons(0,nil))=1+len(nil)
∀xy.len(cons(x,y))=1+len(y)
Instantiate:
Q{x→0,y→nil}
Ground Solver
∀ Module
Partial model:
{…,len(cons(0,nil))=0}

Example #1

`len(nil)=0,`
`len(cons(0,nil))≠1,`
`len(cons(0,nil))=1+len(nil)`

`∀xy.len(cons(x,y))=1+len(y)`

Ground Solver

∀ Module

`UNSAT`

Since `len(cons(0,nil))=1+len(nil)=1+0=1≠1`

Example #2 : Quantified Conjecture

```
len(nil)=0
∀xy.len(cons(x,y))=1+len(y)
```

Axioms

```
¬∀x.len(x)≥0
```

(Negated) Conjecture

Ground Solver

∀ Module

Example #2

```
len(nil)=0
∀xy.len(cons(x,y))=1+len(y)
```

```
¬∀x.len(x)≥0
```

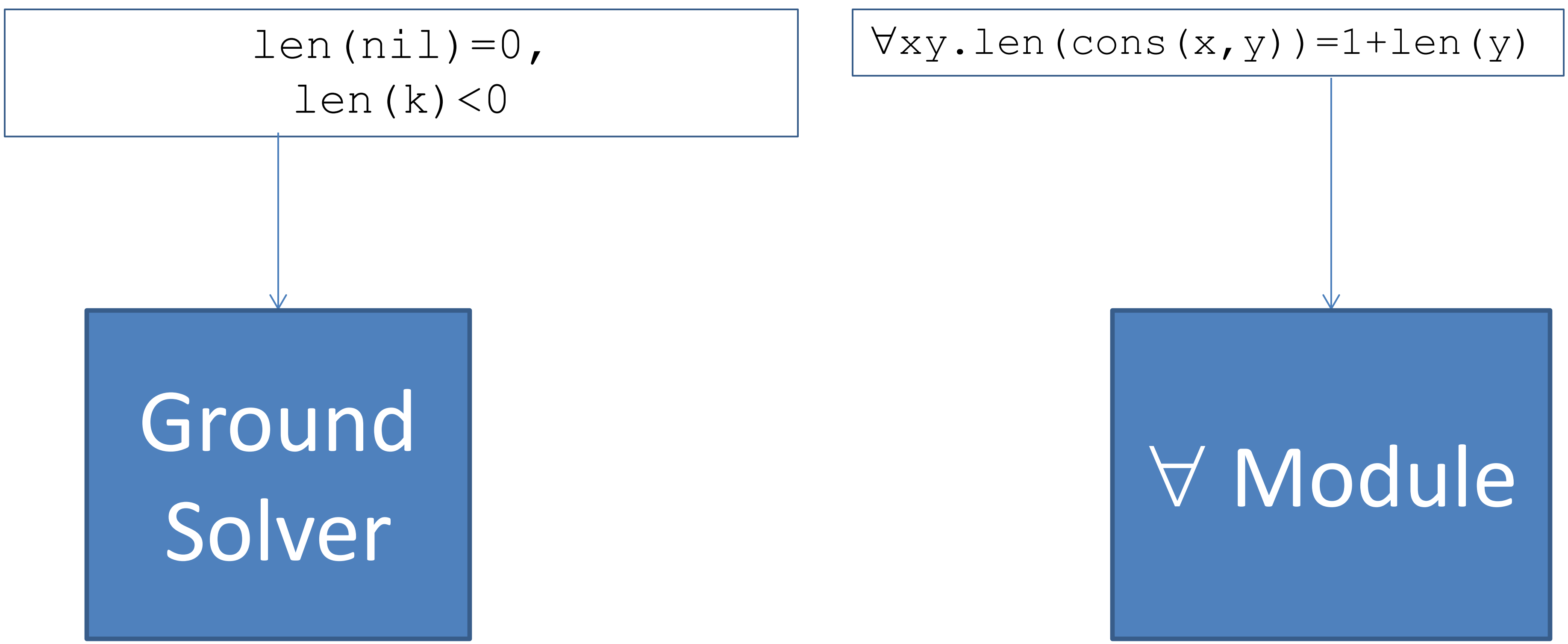
Skolemize : statement (does not) hold for fresh constant **k**

```
¬  len(k)≥0
```

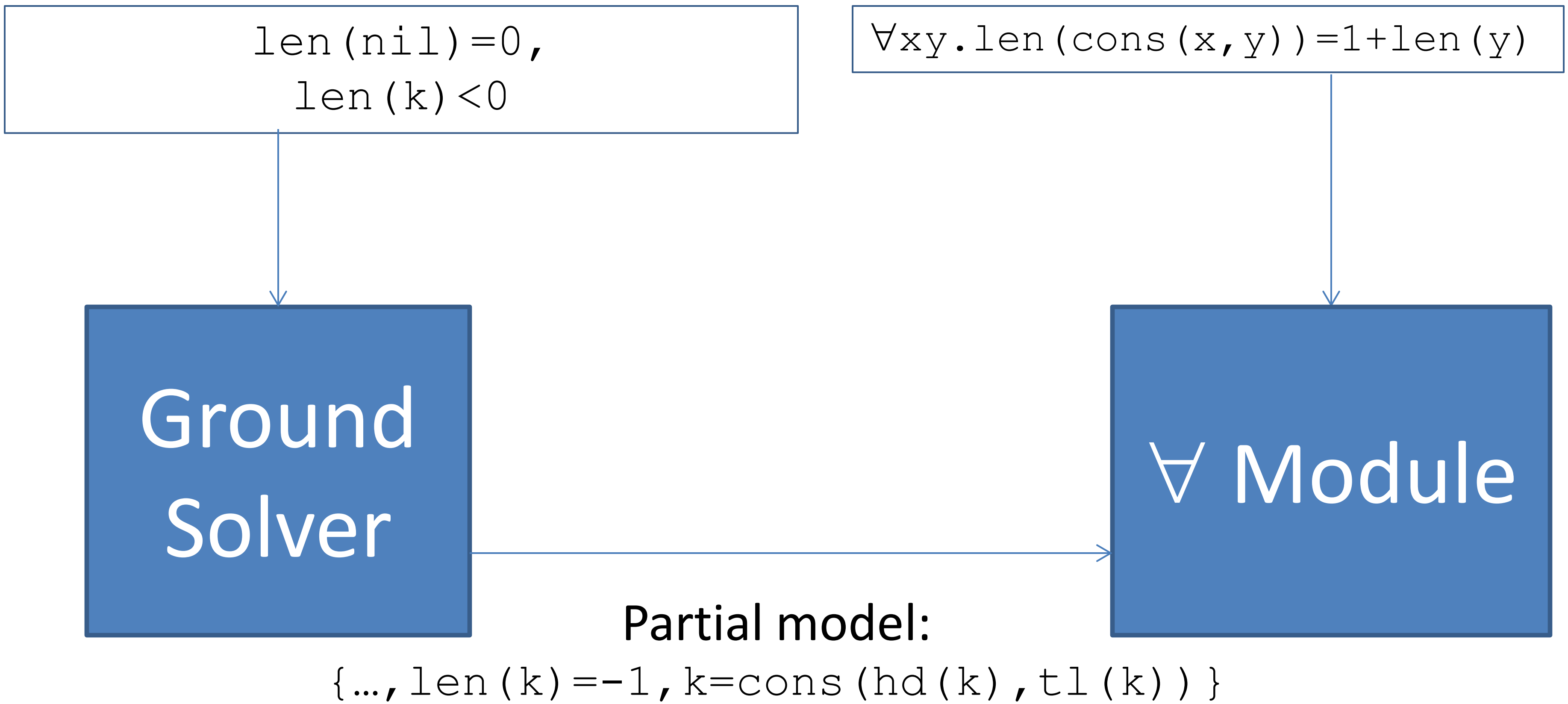
Ground Solver

∀ Module

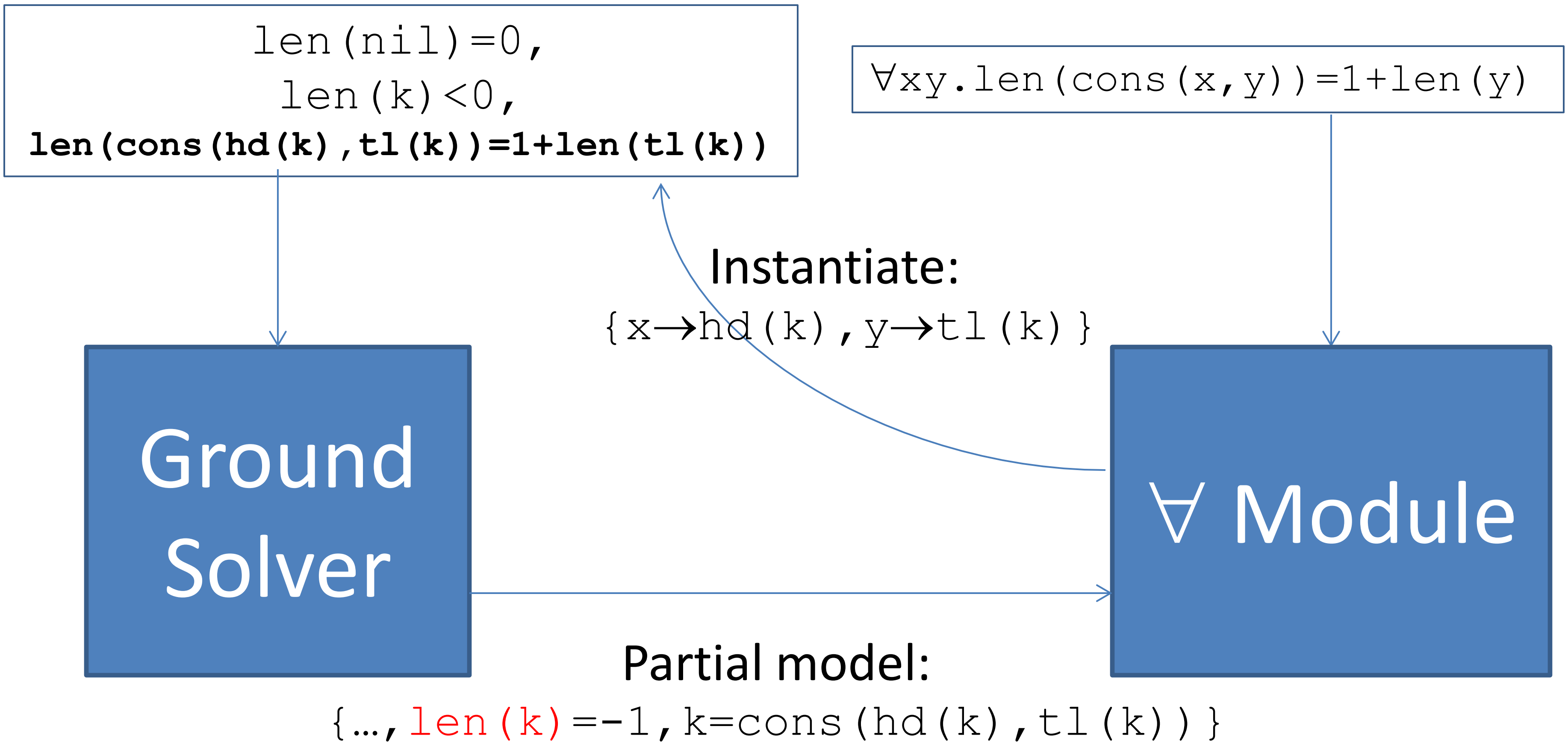
Example #2



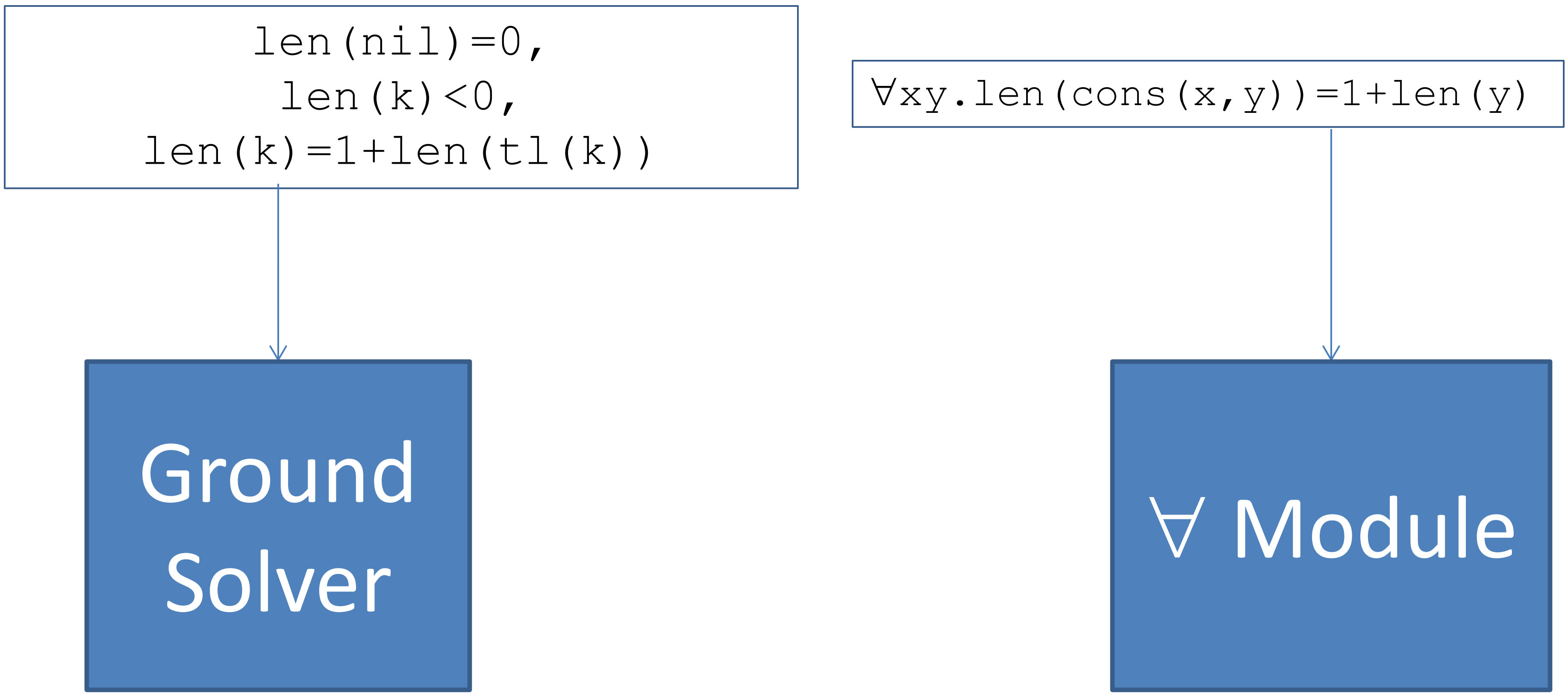
Example #2


len(nil)=0,
len(k)<0
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module
Partial model:
{…,len(k)=-1,k=cons(hd(k),tl(k))}

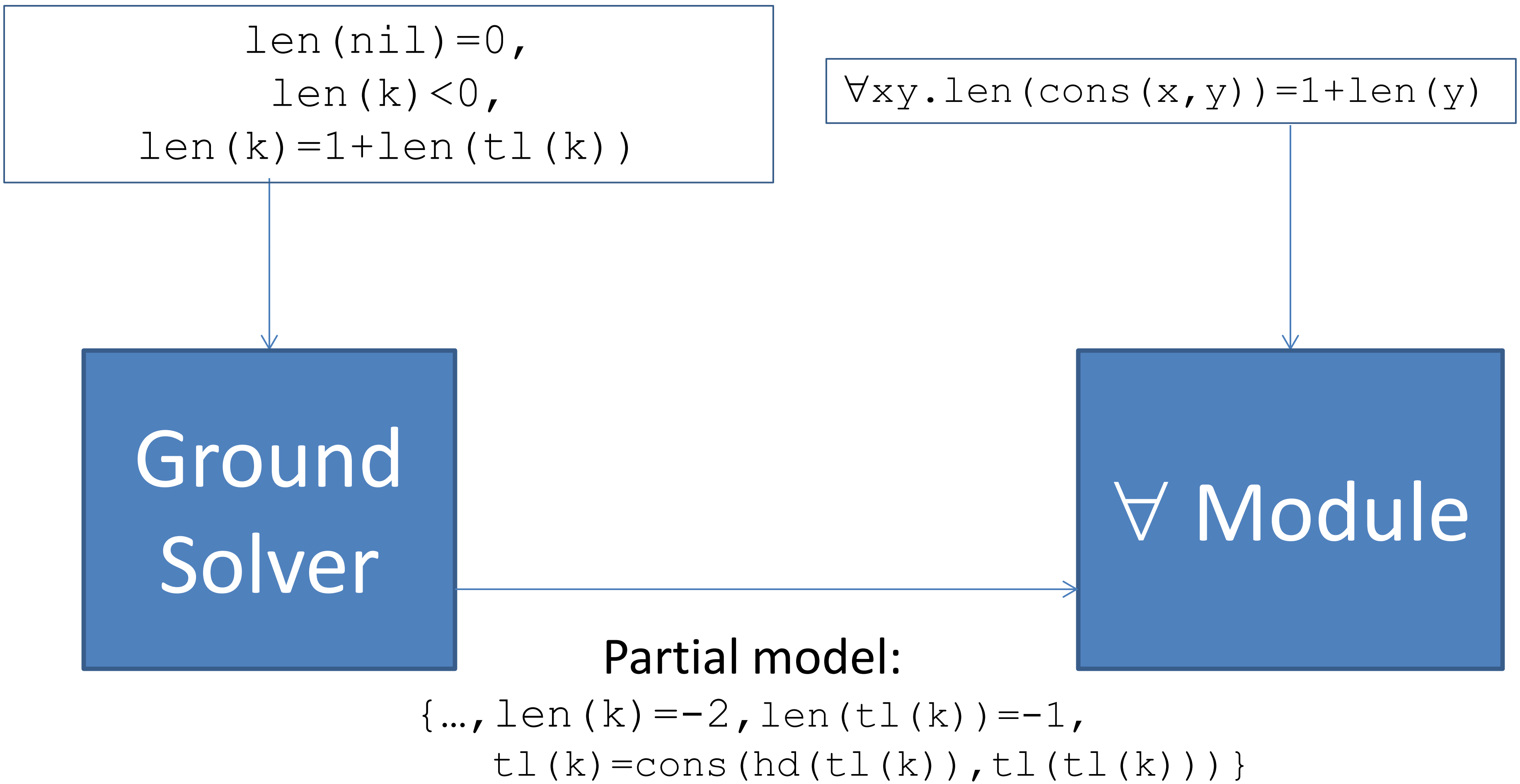
Example #2


len(nil)=0,
len(k)<0,
len(cons(hd(k),tl(k))=1+len(tl(k))
∀xy.len(cons(x,y))=1+len(y)
Instantiate:
{x→hd(k),y→tl(k)}
Ground Solver
∀ Module
Partial model:
{…,len(k)=-1,k=cons(hd(k),tl(k))}

Example #2

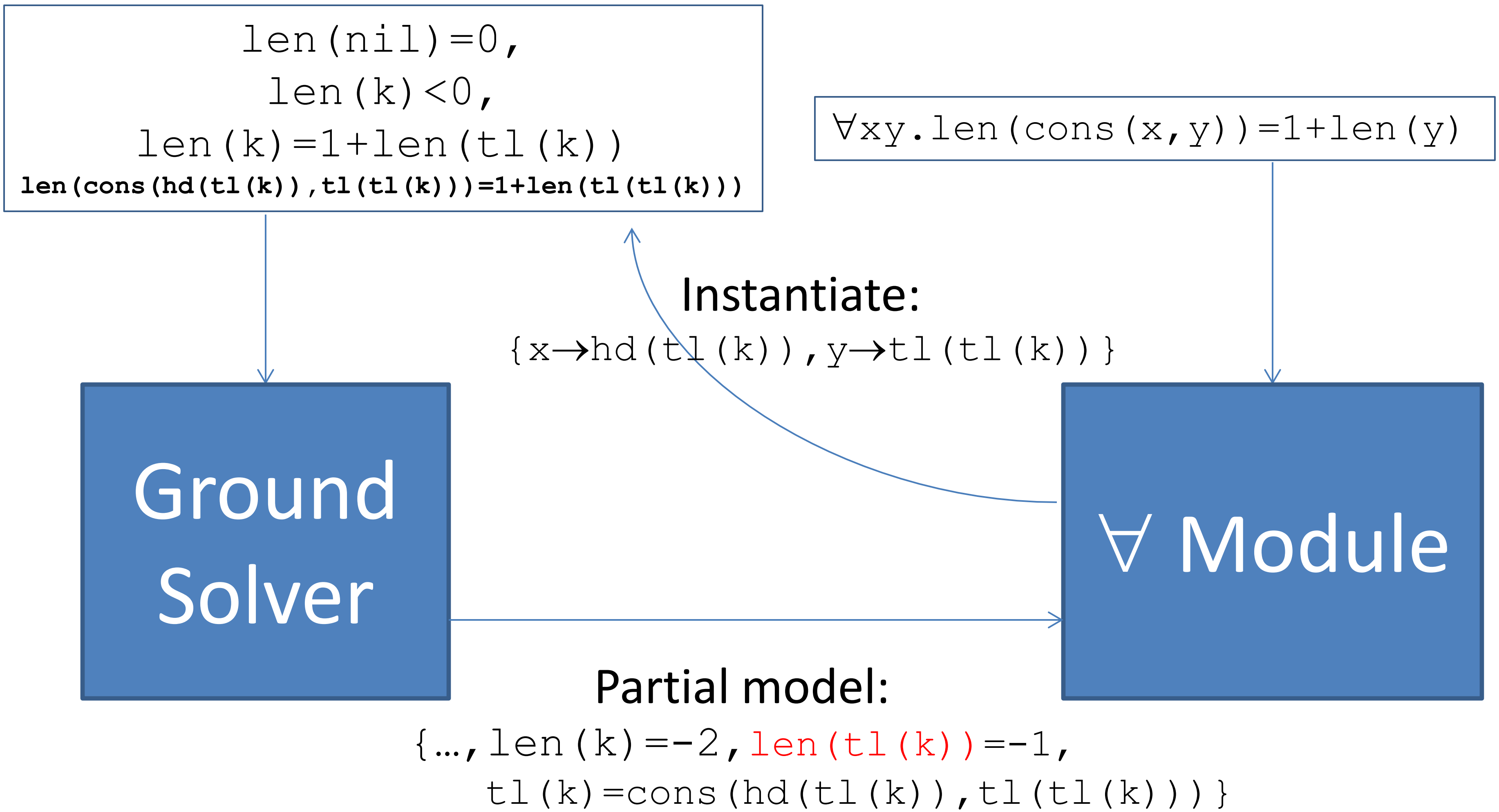

len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module

Example #2

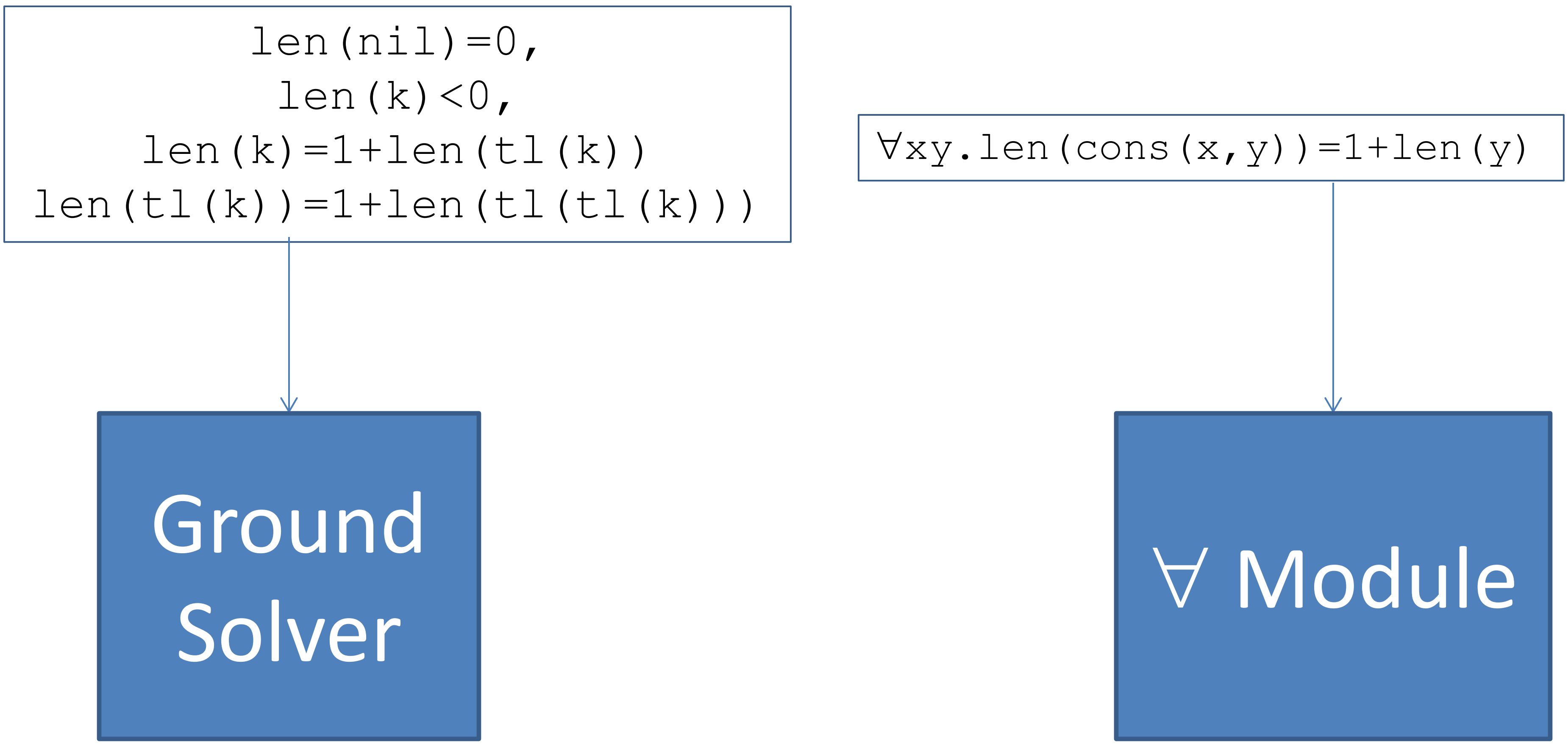


len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module
Partial model:
{…,len(k)=-2,len(tl(k))=-1,
tl(k)=cons(hd(tl(k)),tl(tl(k)))}

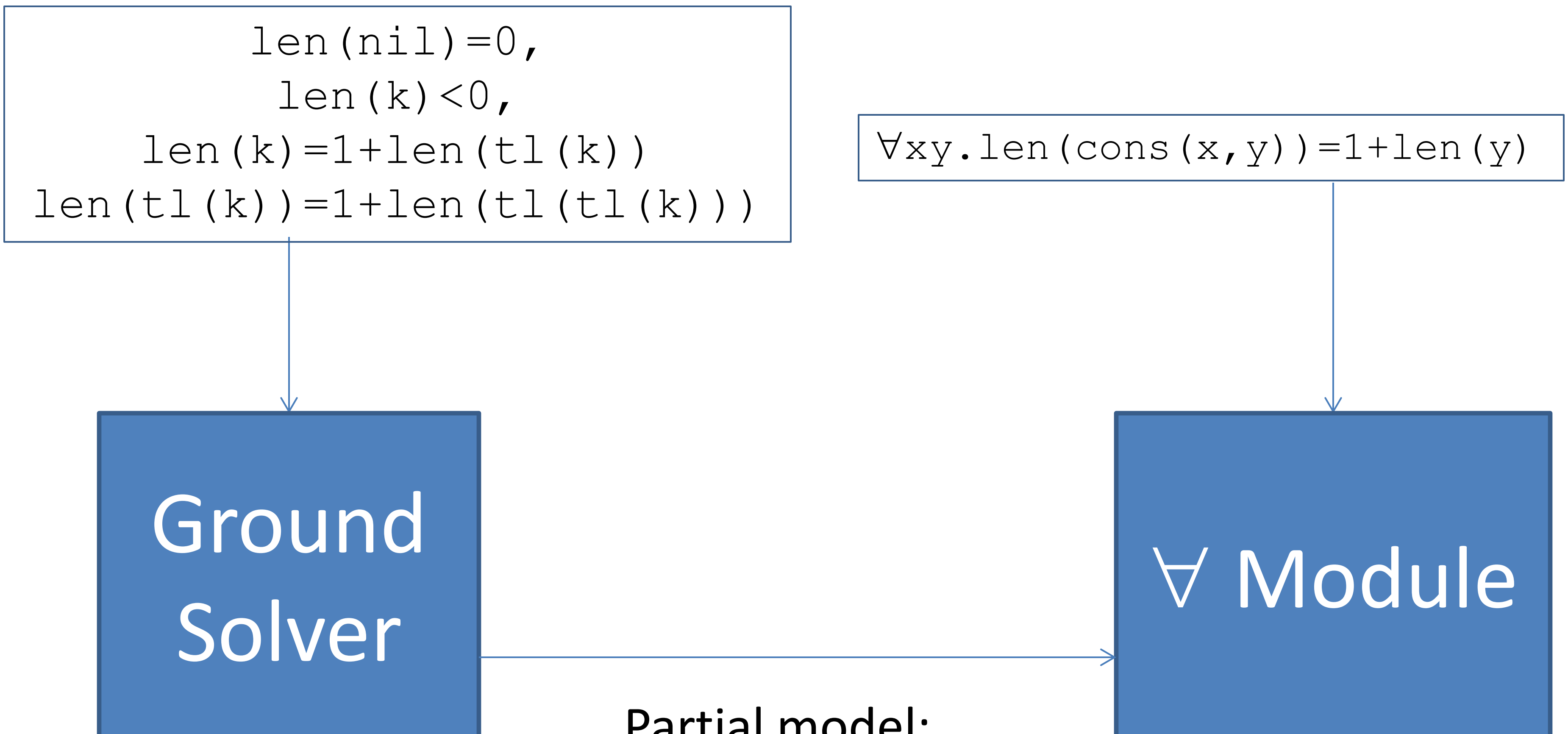
Example #2


len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
len(cons(hd(tl(k)),tl(tl(k)))=1+len(tl(tl(k)))
∀xy.len(cons(x,y))=1+len(y)
Instantiate:
{x→hd(tl(k)),y→tl(tl(k))}
Ground
Solver
∀ Module
Partial model:
{…,len(k)=-2,len(tl(k))=-1,
tl(k)=cons(hd(tl(k)),tl(tl(k)))}

Example #2


len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
len(tl(k))=1+len(tl(tl(k)))
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module

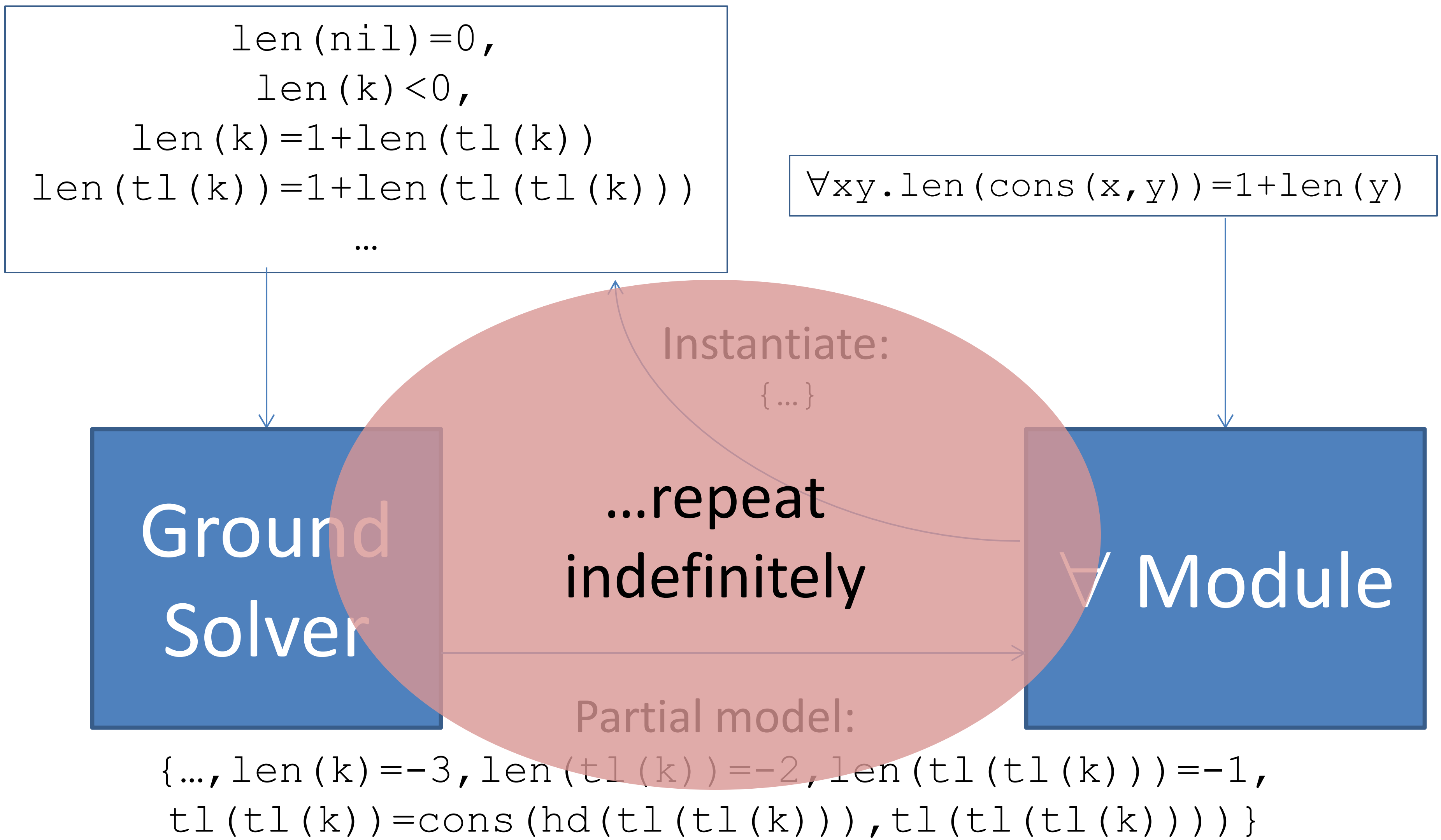
Example #2



Partial model:

`{…,len(k)=-3,len(tl(k))=-2,len(tl(tl(k)))=-1,`
`tl(tl(k))=cons(hd(tl(tl(k))),tl(tl(tl(k))))}`

Example #2


len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
len(tl(k))=1+len(tl(tl(k)))
…
∀xy.len(cons(x,y))=1+len(y)
Instantiate:
{…}
Ground Solver
…repeat indefinitely
∀ Module
Partial model:
{…,len(k)=-3,len(tl(k))=-2,len(tl(tl(k)))=-1,
tl(tl(k))=cons(hd(tl(tl(k))),tl(tl(tl(k))))}

Challenge: Inductive Reasoning

- This example requires induction
- Existing techniques
  - Within inductive theorem provers:
    - ACL2 [Chamarthi et al 2012]
    - HipSpec [Claessen et al 2013]
    - IsaPlanner [Johansson et al 2010]
    - Zeno [Sonnex et al 2012]
  - Induction as preprocessing step to SMT solver:
    - Dafny [Leino 2012]
- No SMT solvers support induction *natively*

  $\Rightarrow$ Until now, in CVC4

Solution: Inductive Strengthening

- Given negated conjecture:

$$\boxed{\neg\forall \texttt{x.len(x)} \geq 0}$$

- Assume property does not for fresh k:

$$\boxed{\neg\ \texttt{len(k)} \geq 0}$$

AND

- Assume k is the *smallest* CE to property:

$$\boxed{\texttt{k=cons(hd(k),tl(k))} \Rightarrow \texttt{len(tl(k))} \geq 0}$$

Example #2: revised

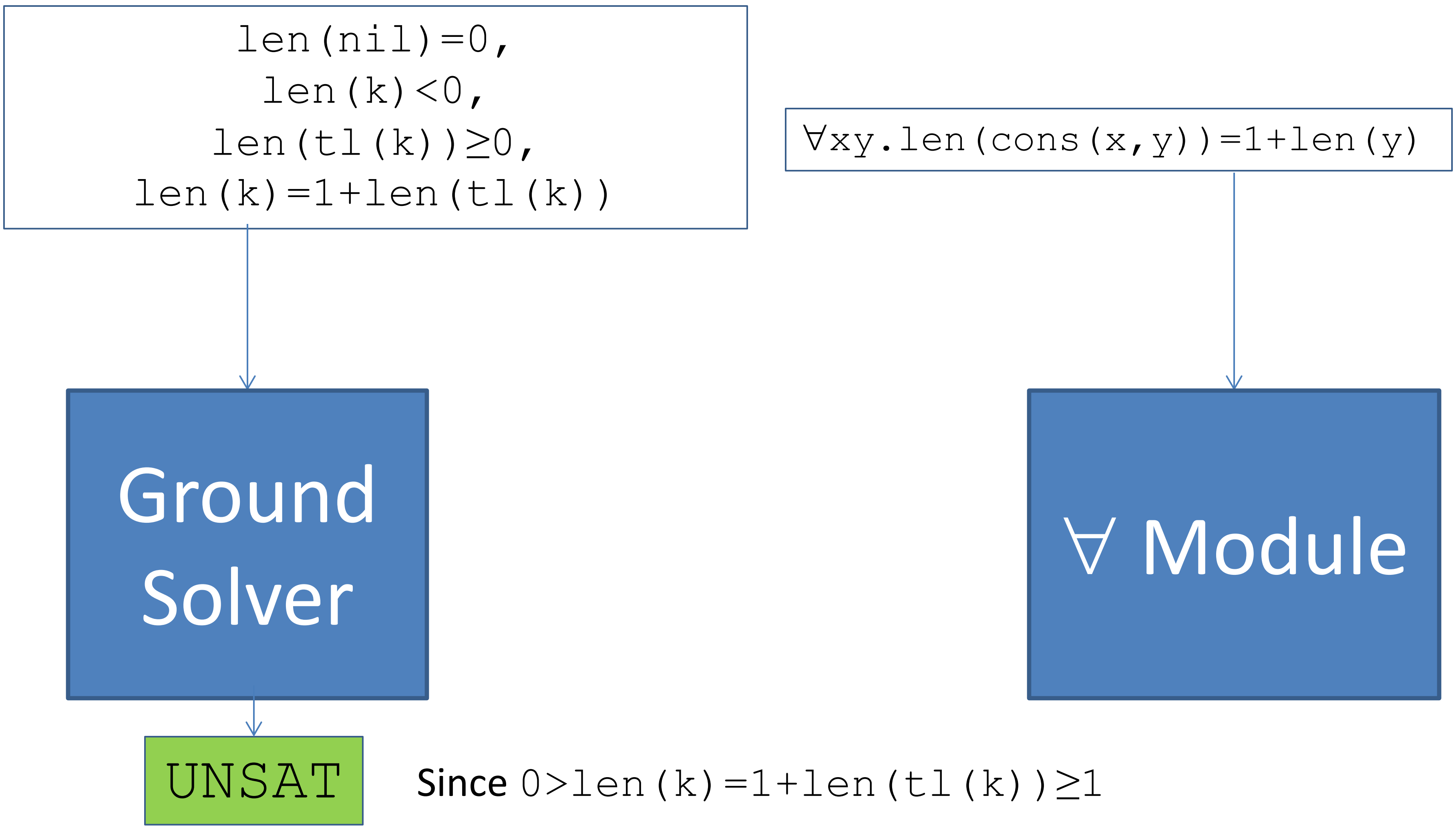
```
len(nil)=0,
len(k)<0,
len(tl(k))≥0,
len(k)=1+len(tl(k))
```

Ground Solver

```
∀xy.len(cons(x,y))=1+len(y)
```

∀ Module

Example #2: revised



Skolemization with Inductive Strengthening

- General form:

$$\boxed{\forall x.P(x) \vee (\neg P(k) \wedge \forall y.(y < k \Rightarrow P(y)))}$$

  - For well-founded relation "**<**"
- Extends for multiple variables
- Common examples of "**<**" in SMT:
  - (Weak) structural induction on inductive datatypes
    - Assume property holds for direct children of k of same type
  - (Weak) well-founded induction on integers
    - Assume property holds for (k-1), with base case 0

Challenge: Subgoal Generation

- Unfortunately, inductive strengthening is not enough
- Consider conjecture:

```
∀x.len(rev(x))=len(x)
```

  - where `rev` is axiomatized by:

```
rev(nil)=nil,
∀xy.rev(cons(x,y))=app(rev(y),cons(x,nil))
```

- To prove, requires induction, and "subgoals":

```
∀xy.len(app(x,y))=plus(len(x),len(y))
```

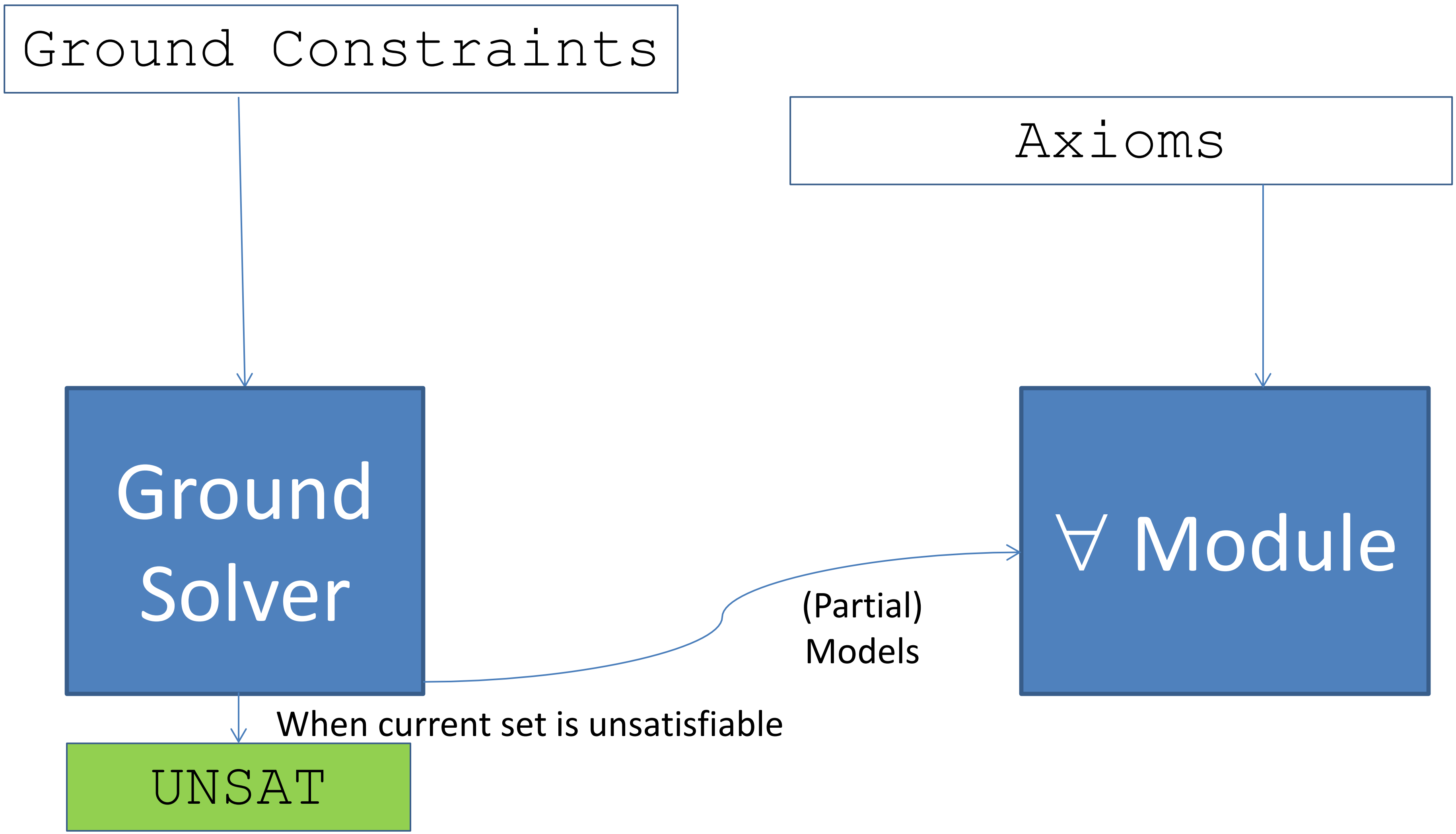
```
∀xy.plus(x,y)=plus(y,x)
```

Generating candidate subgoals

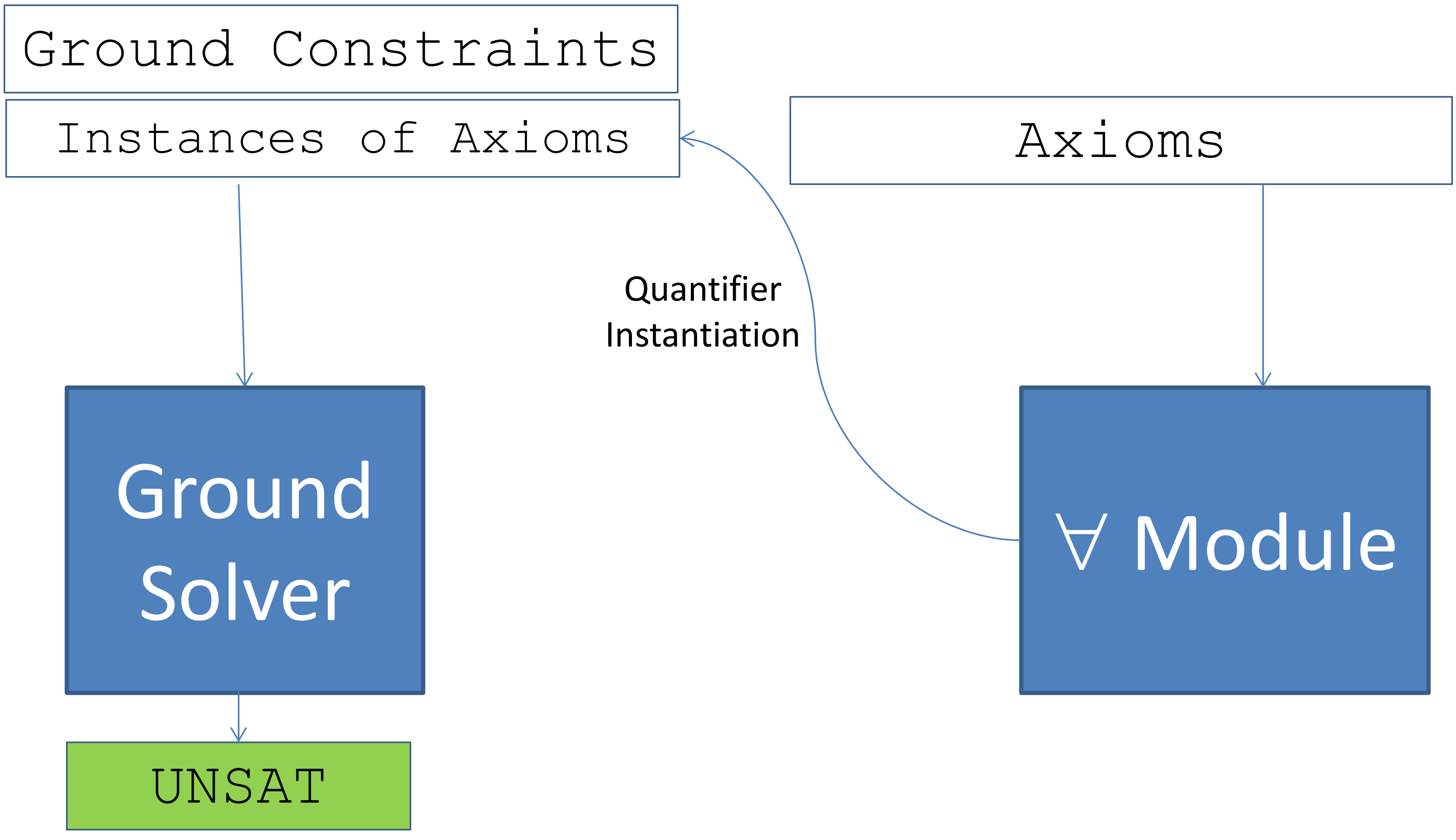
- How to generate necessary subgoals?
  - Idea: Enumerate/prove them in a principled way
    - QuickSpec [Claessen et al 2010]

```
∀x.len(x)=Z
∀x.len(x)=S(Z)
∀x.app(x,nil)=nil
∀x.app(x,nil)=x
∀x.app(x,nil)=cons(0,x)
…
∀xy.plus(x,y)=plus(x,0)
∀xy.plus(x,y)=plus(y,x)
…
∀xy.len(app(x,y))=plus(len(x),len(y))
…
```

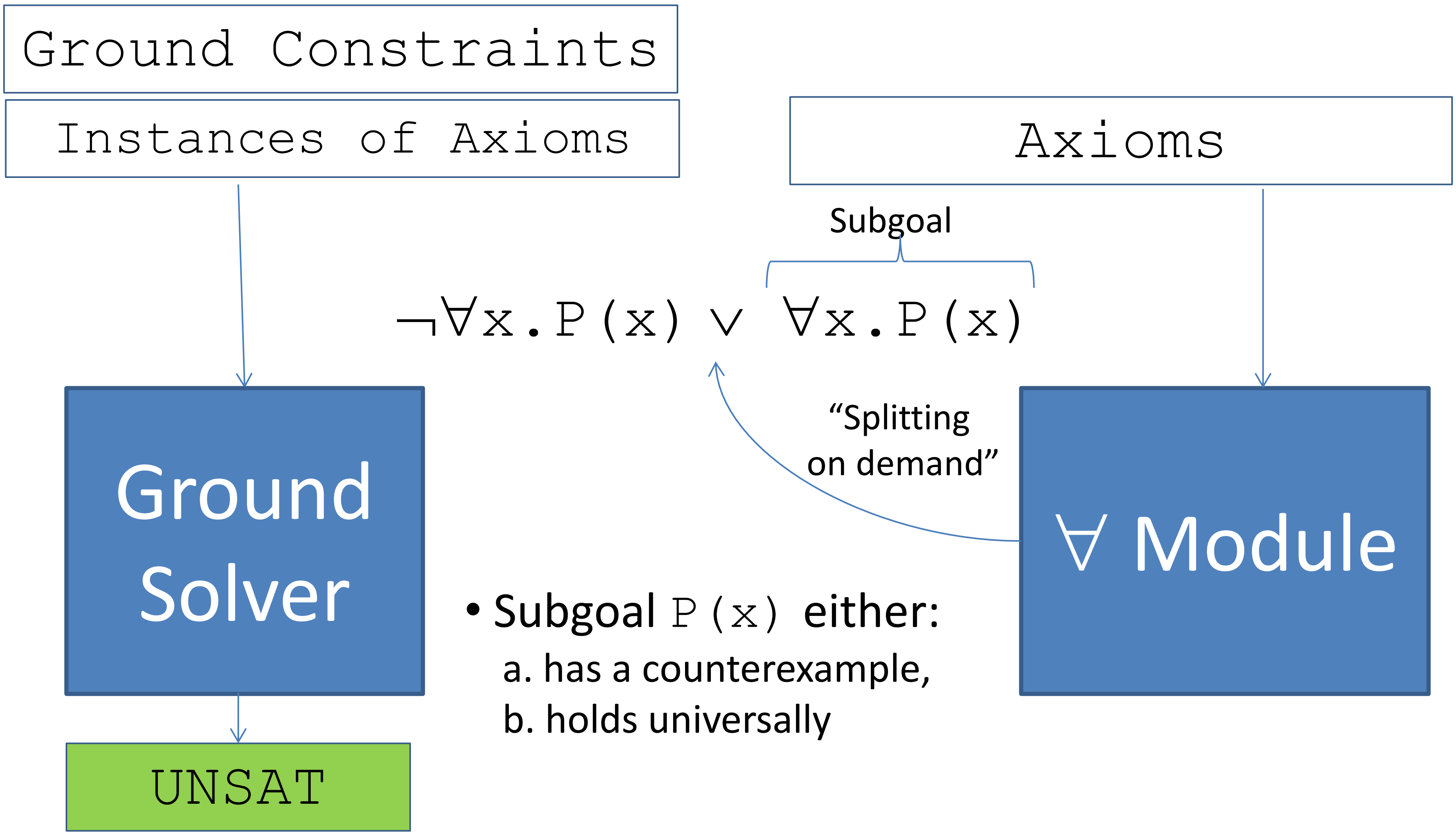
Subgoal Generation in SMT



Subgoal Generation in SMT

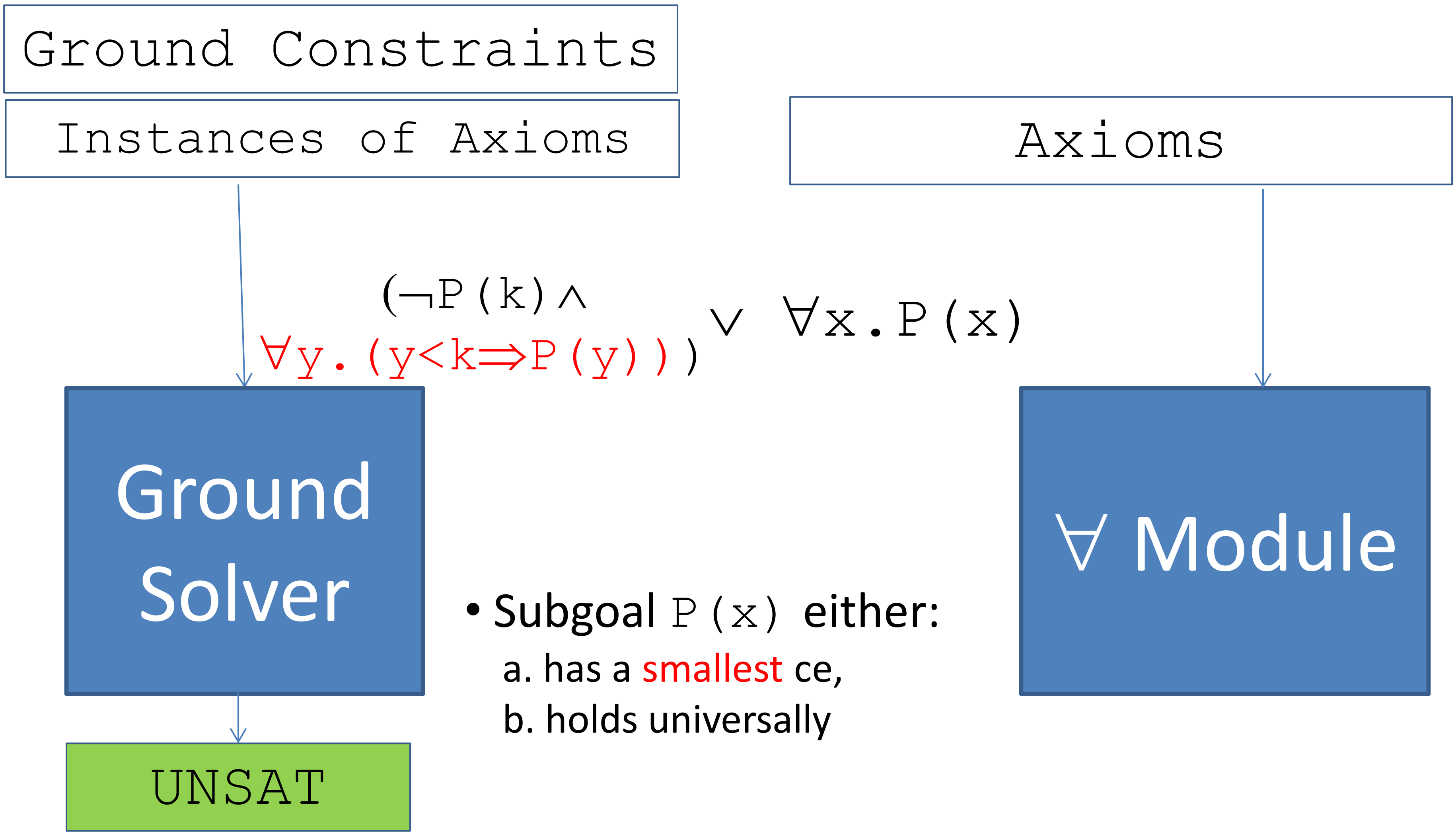


Subgoal Generation in SMT



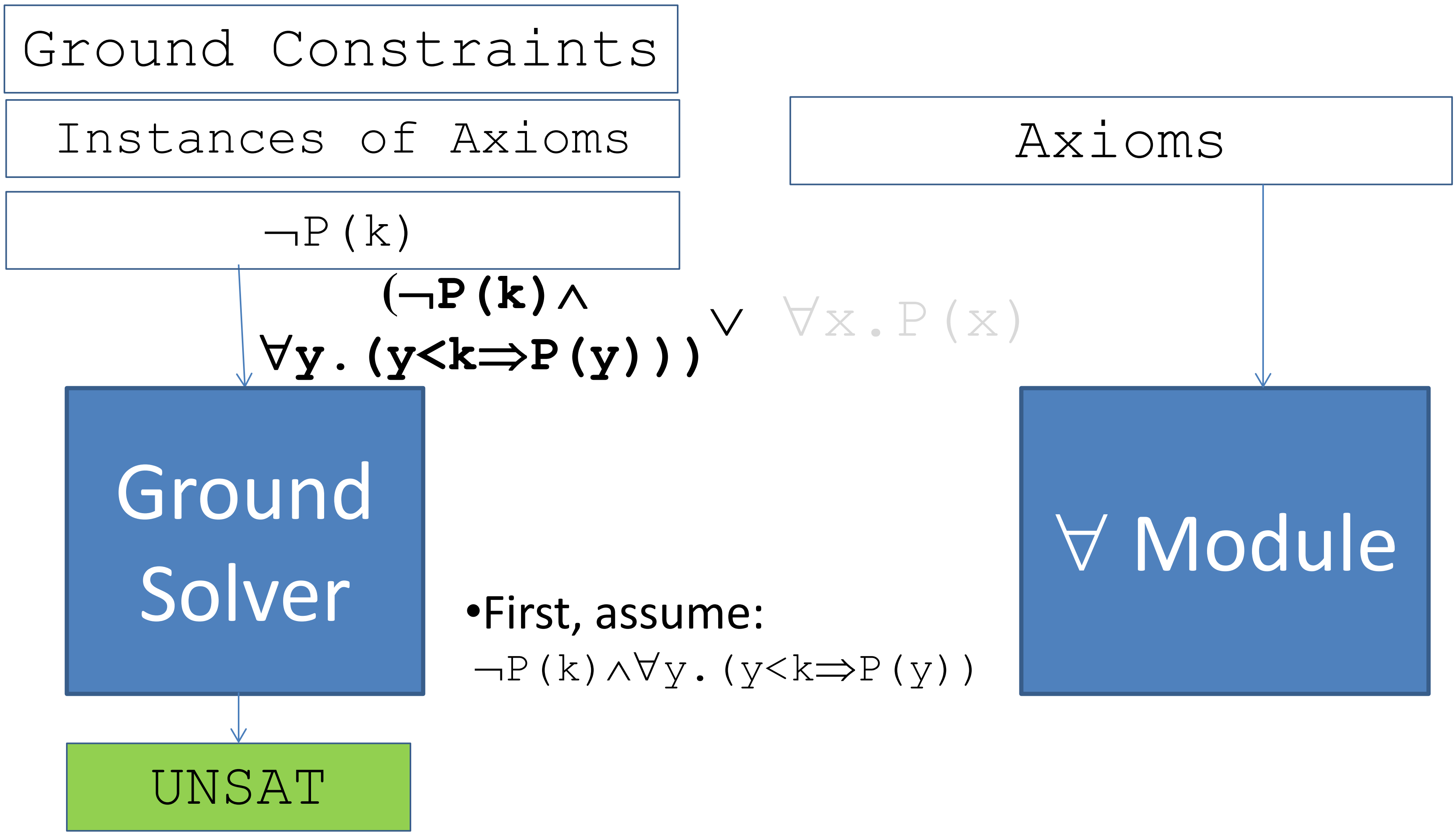
- Subgoal `P(x)` either:
  a. has a counterexample,
  b. holds universally

Subgoal Generation in SMT



- Subgoal `P(x)` either:
  a. has a smallest ce,
  b. holds universally

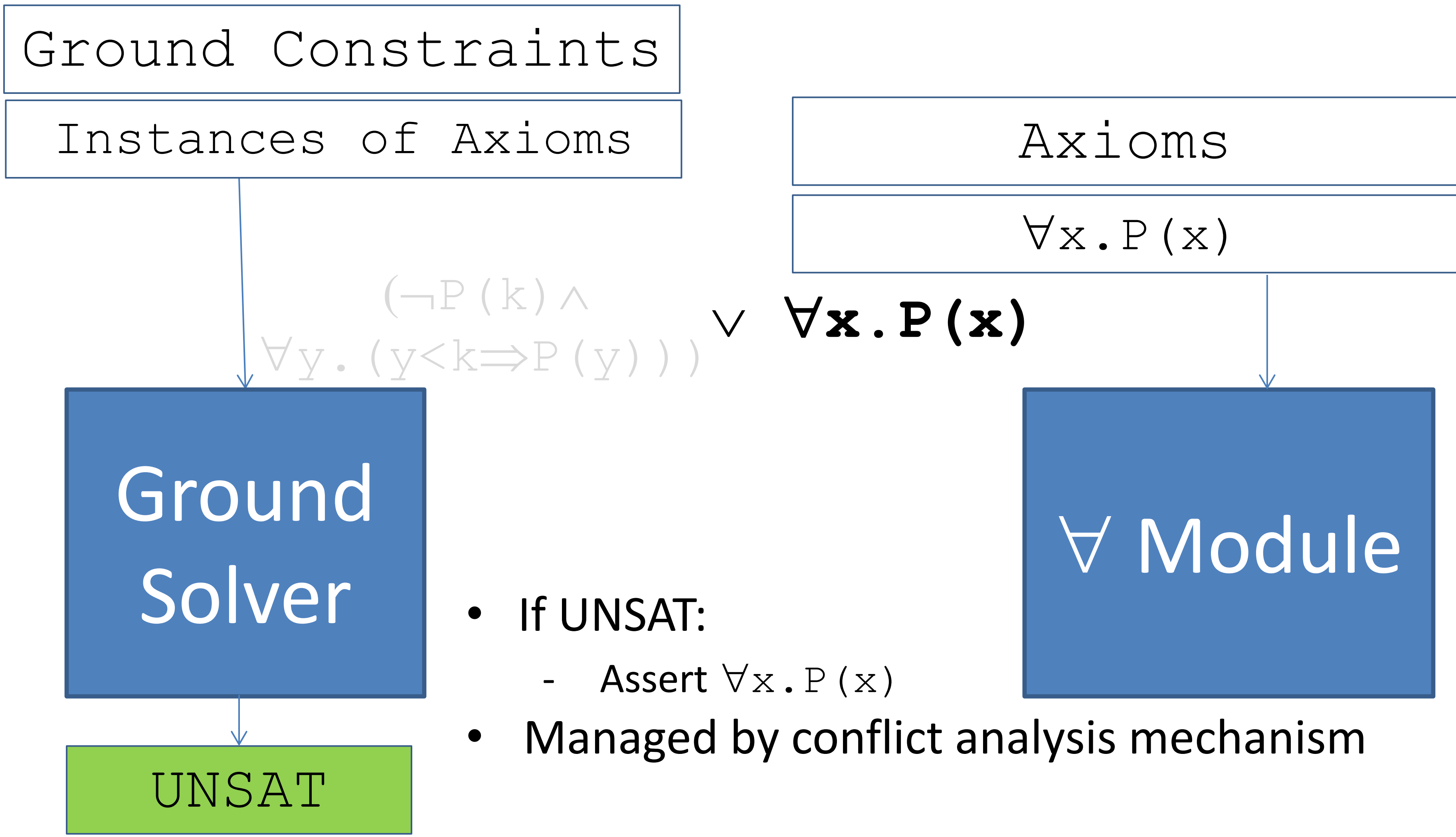
Subgoal Generation in SMT



- First, assume:

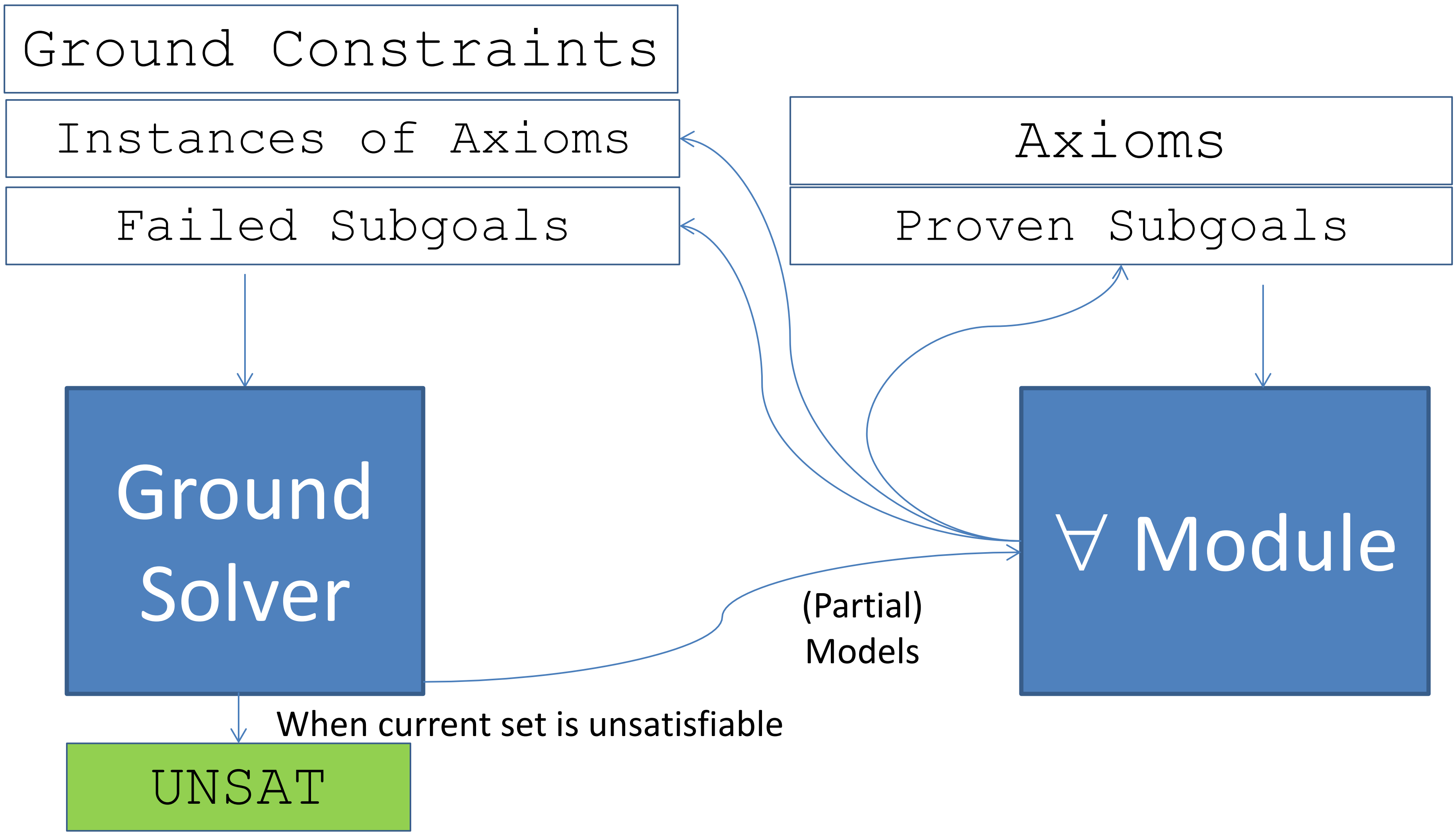
$\neg P(k) \land \forall y.(y<k \Rightarrow P(y))$

Subgoal Generation in SMT



- If UNSAT:
  - Assert ∀x.P(x)
- Managed by conflict analysis mechanism

Subgoal Generation in SMT



Subgoal Generation : Challenges

- Main challenge: scalability
- Keys to success:
  - Enumerate subgoals in a fair manner (smaller first)
  - Filter out subgoals that are not useful

Subgoal Filtering

- Given: $\boxed{\forall x.\,\mathrm{len}(\mathrm{rev}(x))=\mathrm{len}(x)}$
- Filtering based on “active” symbols:

  $\forall xy.\,\mathrm{count}(x,y)=\mathrm{count}(\mathrm{rev}(x),y)$

  – Irrelevant, since conjecture is not related to “count”
- Filtering based on canonicity:

  $\forall x.\,\mathrm{len}(x)=\mathrm{len}(\mathrm{app}(x,\mathrm{nil}))$

  – Redundant, since we know $\forall x.\,x=\mathrm{app}(x,\mathrm{nil})$
- Filtering based on counterexamples:

  $\forall x.\,\mathrm{len}(x)=\mathrm{len}(\mathrm{app}(x,x))$

  – False, since we know $\mathrm{len}(x)\neq\mathrm{len}(\mathrm{app}(x,x))$ for $x\neq\mathrm{nil}$

$\Rightarrow$ Using techniques, typically can remove >95% subgoals

Benchmarks

- Four benchmark sets:
  1. IsaPlanner [Johansson et al 2010]
     - List, Nats, Trees, (some) higher-order functions
  2. Clam [Ireland 1996]
     - Lists, Nats, Sets
       - Designed specifically to require subgoals
  3. HipSpec [Claessen et al 2013]
     - Lists, Nats
       - e.g. : sum of n cubes is square of nth triangle number
  4. Leon
     - Amortized Queues, Binary search trees, Leftist Heaps

Benchmarks : Encodings

1. Datatype encoding (**dt**), e.g. define plus as:

$$\boxed{\begin{array}{l}\forall \texttt{x.plus(Z,x)=x} \\ \forall \texttt{xy.plus(S(x),y)=S(plus(x,y))}\end{array}}$$

2. Theory encoding (**dtt**), using builtin CVC4 theories:
   – Rephrase axioms/conjectures in terms of e.g. "+"
3. Combined, theory-isomorphism encoding (**dti**)
   – Keep encoding, provide mappings to theory symbols:
     - Injection "`toInt`" from dt to int, with axiom:

$$\left.\boxed{\forall \texttt{xy.toInt(plus(x,y))=toInt(x)+toInt(y)}}\right\} A$$

$\Rightarrow$ *2,3 allow SMT solver to leverage theory reasoning*

– Thus, we get subgoals for "free", e.g.:

$$A \models_T \boxed{\forall \texttt{xy.plus(x,y)=plus(y,x)}}$$

Results : SMT solvers

	dt	dtt	dti
z3	35	72	75
cvc4	29	63	68
cvc4+i	204	180	240
cvc4+ig	260	201	277

cvc4+i:
with induction

cvc4+ig:
with induction
+subgoal gen.

- Results for 311 benchmarks from 4 classes
- 300 second timeout

Results: Subgoal Generation

- With subgoals, solved +37 for **dti** encoding
  - Only solved +1 when filtering turned off
- Overhead of subgoal generation was small:
  - 30 cases (out of 933) was 2x slower
  - 9 cases (out of 933) went solved -> unsolved
- Most subgoals were small: term size ≤ 3
  - Some were non-trivial (not discovered manually)

Comparison with Other Provers

Benchmark class

Solvers

	Isaplanner	Clam	HipSpec	Leon
cvc4+ig (dti)	80	39	18	42
ACL2	73			
Clam		41		
Dafny	45			
Hipspec	80	47	26	
Isaplanner	43			
Zeno	82	21		
Total	*85*	*50*	*26*	*45*

- Translated/evaluated in previous studies
- Tools tend to perform well on benchmarks they are tuned for
  - CVC4 competitive with state-of-the-art inductive theorem provers

Summary

- Techniques for Induction in SMT solver CVC4
- Best performance by use of:
  - Theory reasoning (**dti** encoding)
  - Subgoal generation
- Competitive with inductive theorem provers

Future Work

- Improvements to subgoal generation
  - Filtering heuristics
  - Configurable approaches for signature of subgoals
- Incorporate more induction schemes
- Completeness criteria
  - Identify cases approach is guaranteed to succeed
- Applications:
  - Tighter integration with Leon (http://leon.epfl.ch)

Thanks!

- CVC4 publicly available:
  - http://cvc4.cs.nyu.edu/downloads/
  - Induction techniques:
    - Enabled by "`--quant-ind`"
- Benchmarks (SMT2) available:
  - http://lara.epfl.ch/~reynolds/VMCAI2015-ind