Using CVC4 for Proofs by Induction

Andrew Reynolds

May 28th, 2015

Overview

- Satisfiability Modulo Theories (SMT) solvers
  - Lack support for inductive reasoning
- "Induction for SMT solvers"

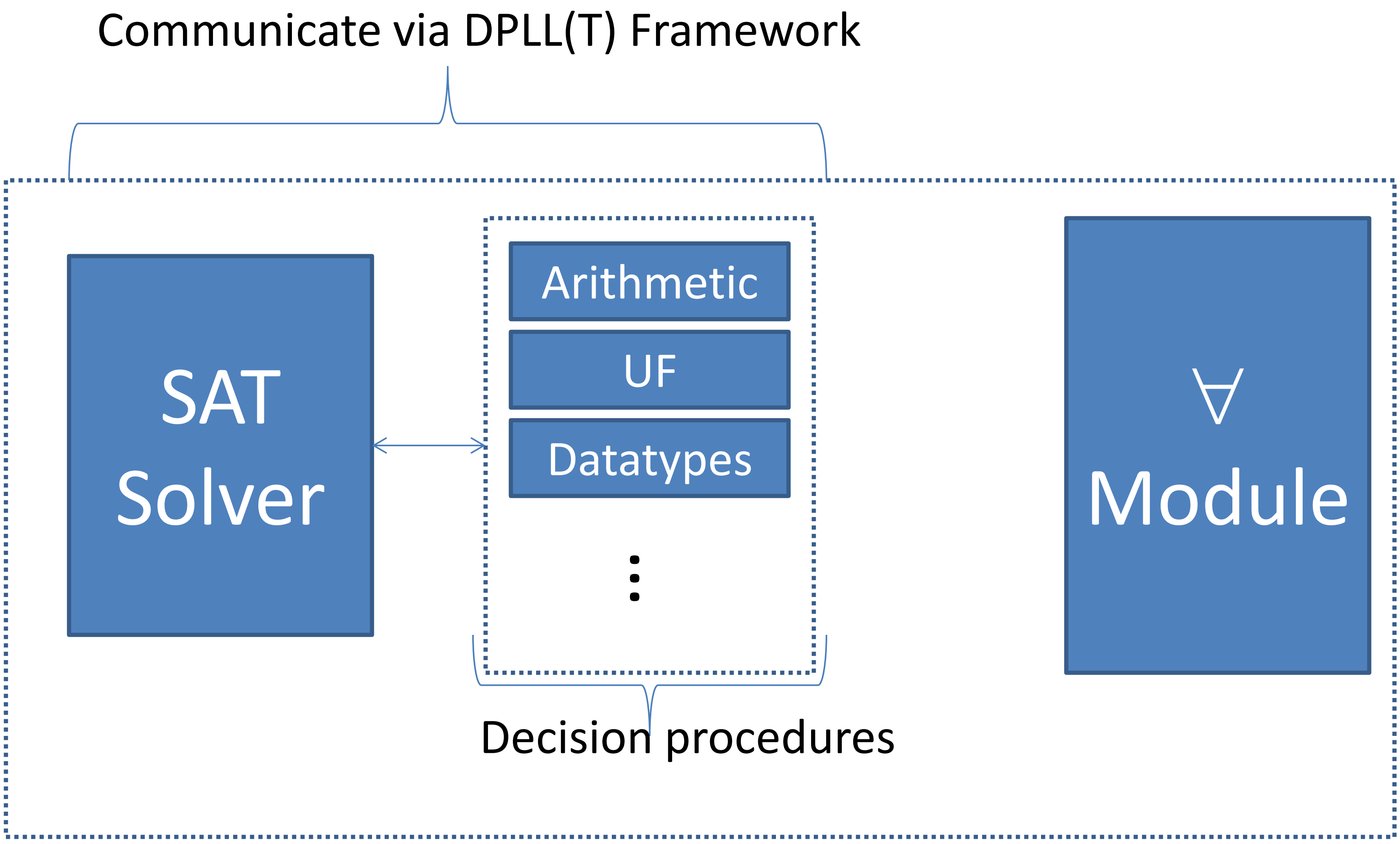
  With Viktor Kuncak, VMCAI 2015

  - Techniques for induction in SMT solvers
    - Subgoal generation
    - Encodings that leverage theory reasoning
    - Benchmarks/Evaluation

SMT Solvers

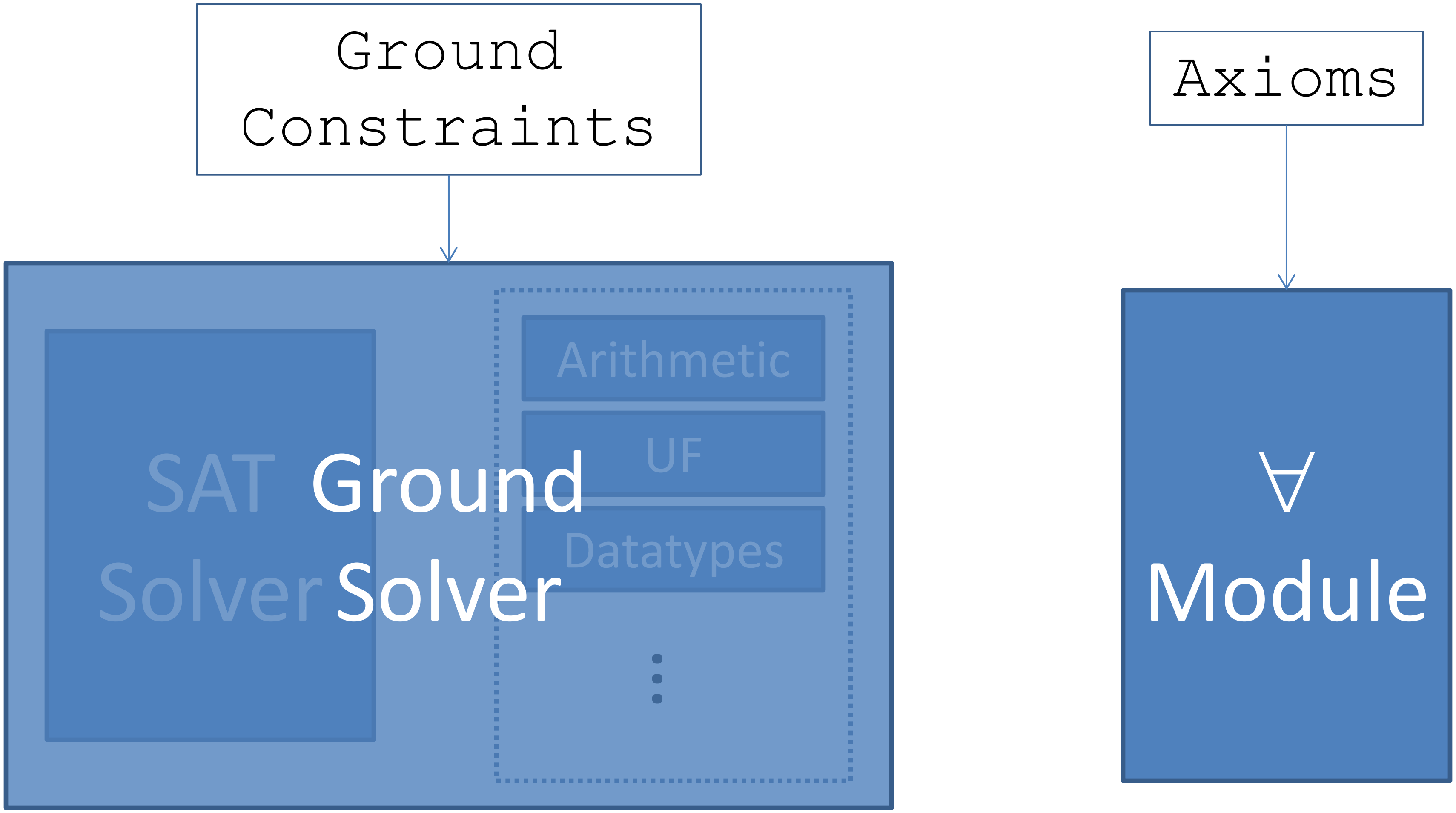
- SMT solvers:
  - Used in formal methods applications:
    - Software verification, automated theorem proving
  - Determine the satisfiability of:
    - Boolean combinations of ground theory constraints
      - Linear arithmetic, BitVectors, Arrays, Datatypes, etc.
  - Have limited support for quantified formulas $\forall$
    - Approaches tend to be heuristic (e.g. E-matching)
    - Often fail on simple examples
      - Notably for problems requiring inductive reasoning

SMT Solver

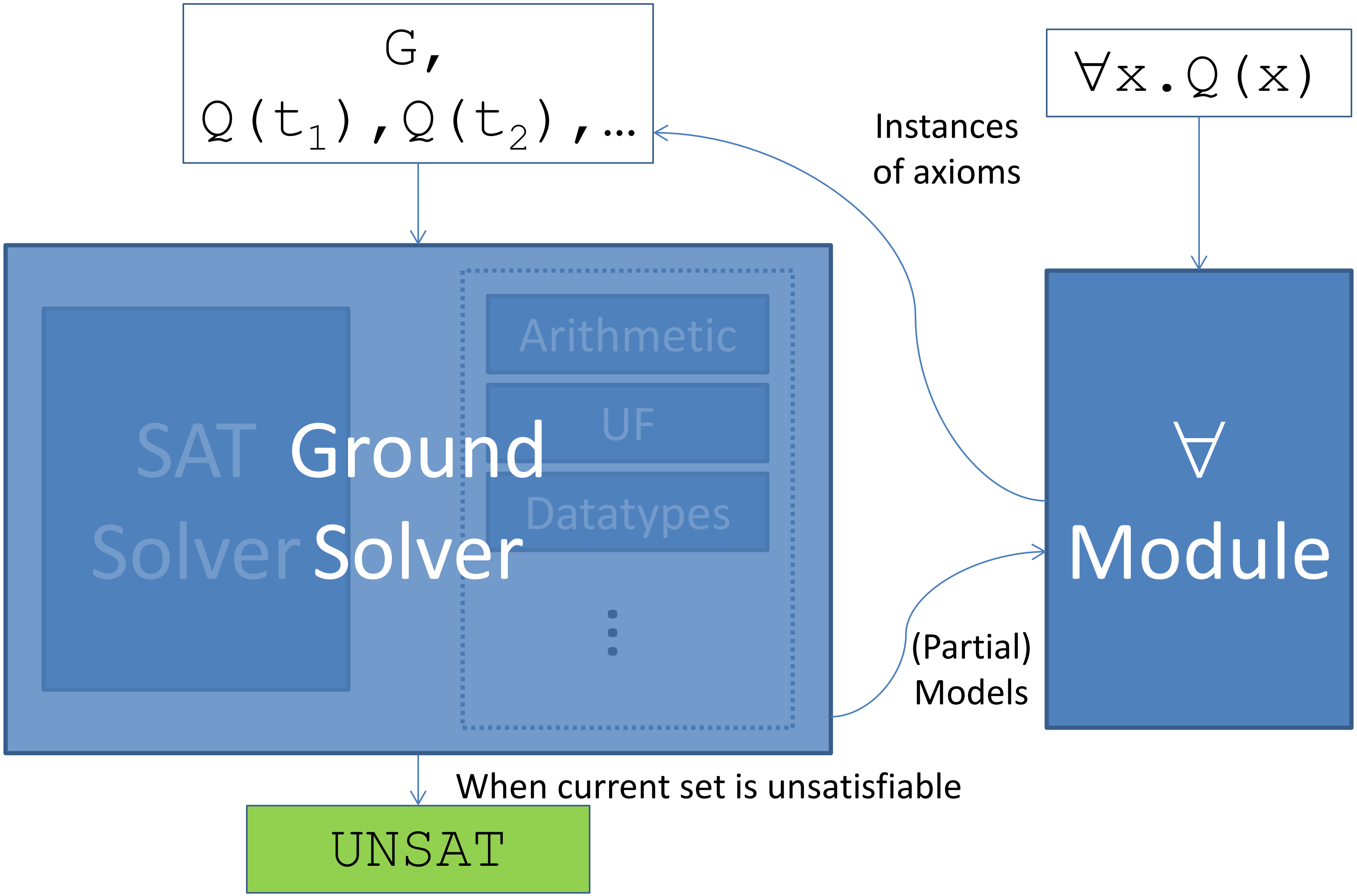


Communicate via DPLL(T) Framework
SAT Solver
Arithmetic
UF
Datatypes
⋮
∀ Module
Decision procedures

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)
Arithmetic
UF
Datatypes
SAT
Solver
Ground
Solver
∀
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

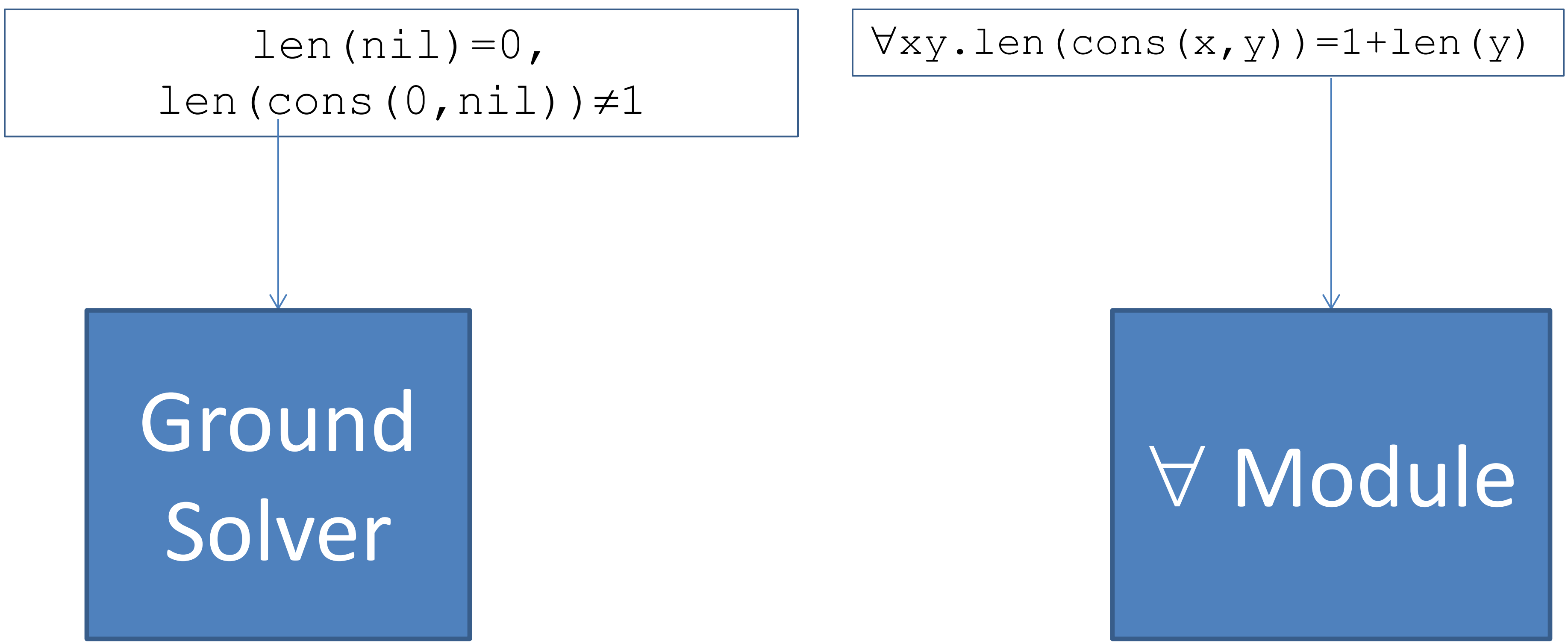
```
¬len(cons(0,nil))=1
```

(Negated) Conjecture

Ground Solver

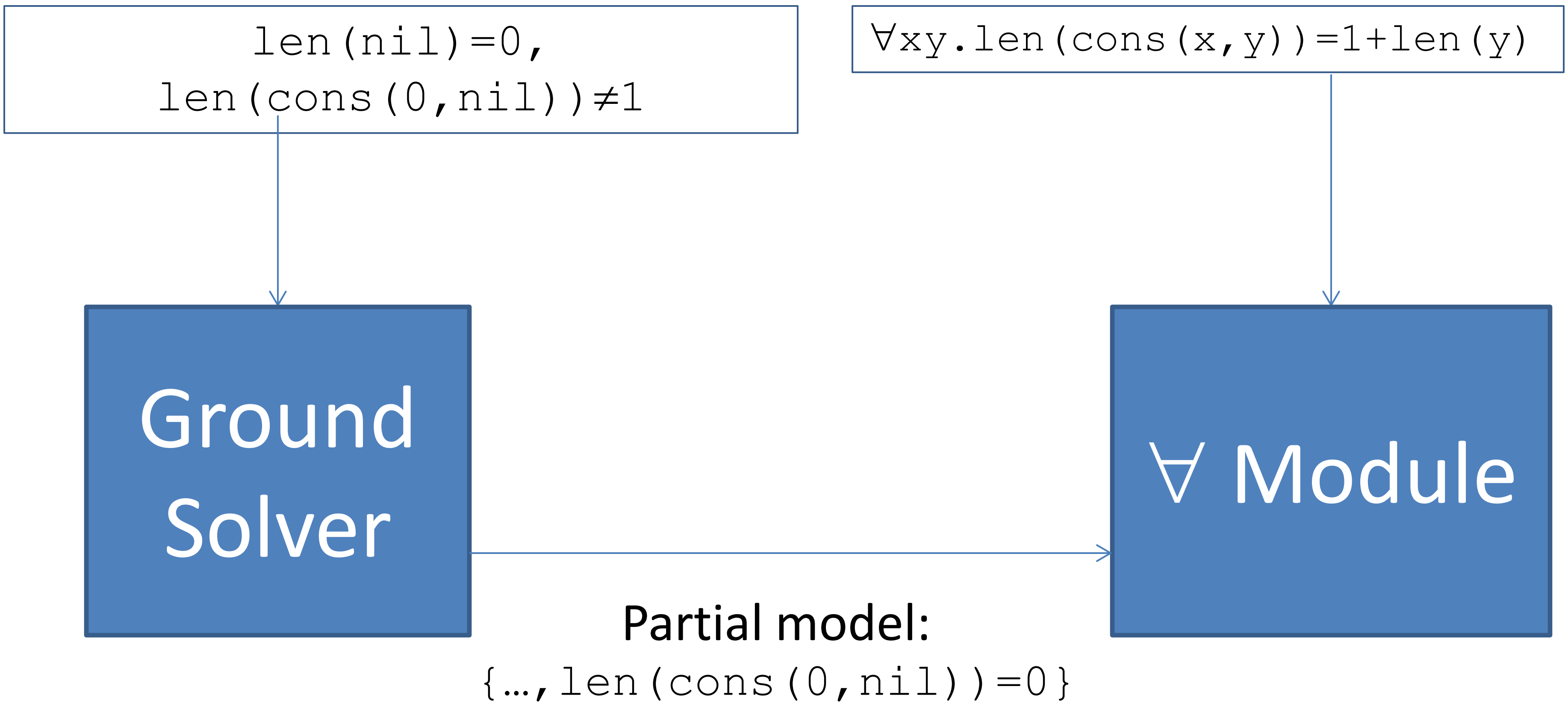
∀ Module

Example #1

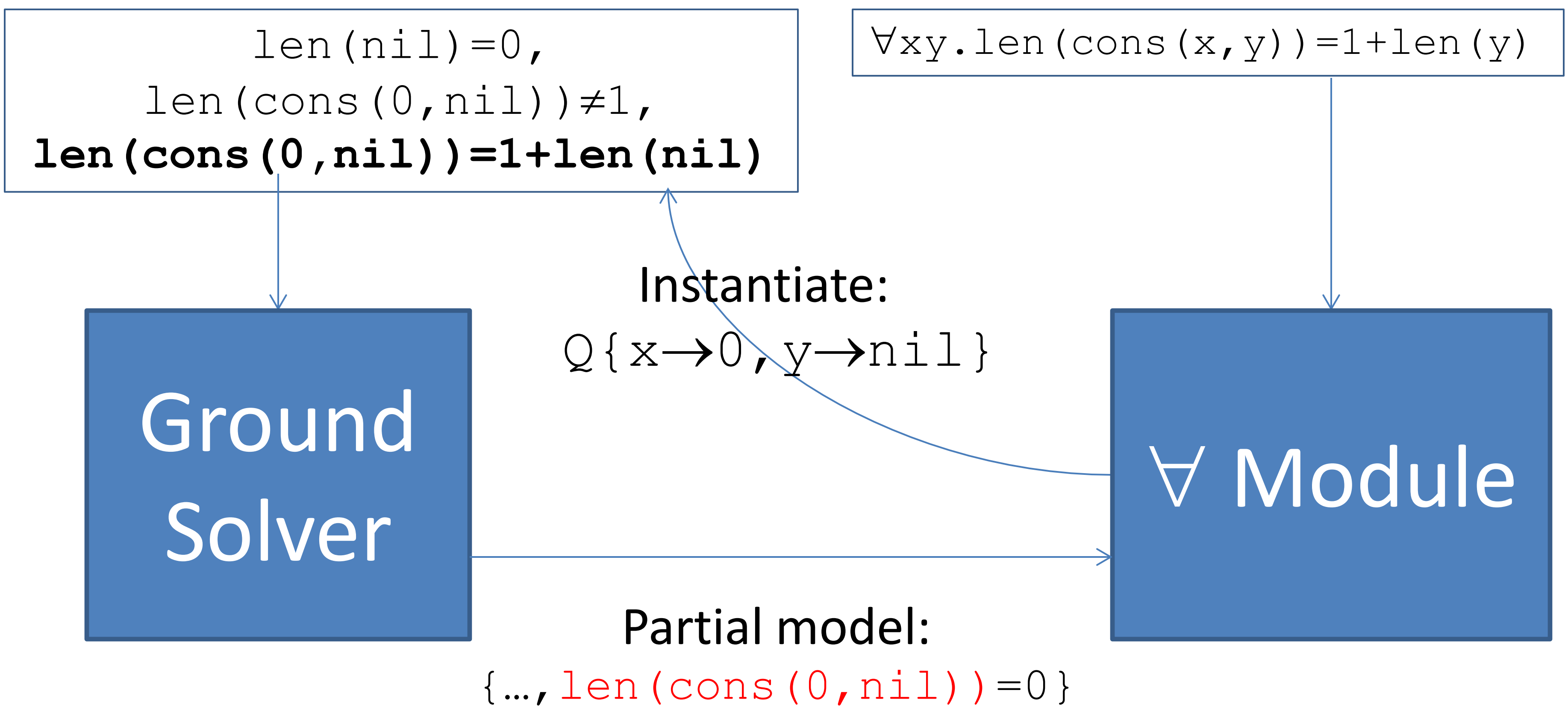


len(nil)=0,
len(cons(0,nil))≠1
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module

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

$\texttt{len(nil)=0}$

$\forall \texttt{xy.len(cons(x,y))=1+len(y)}$

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

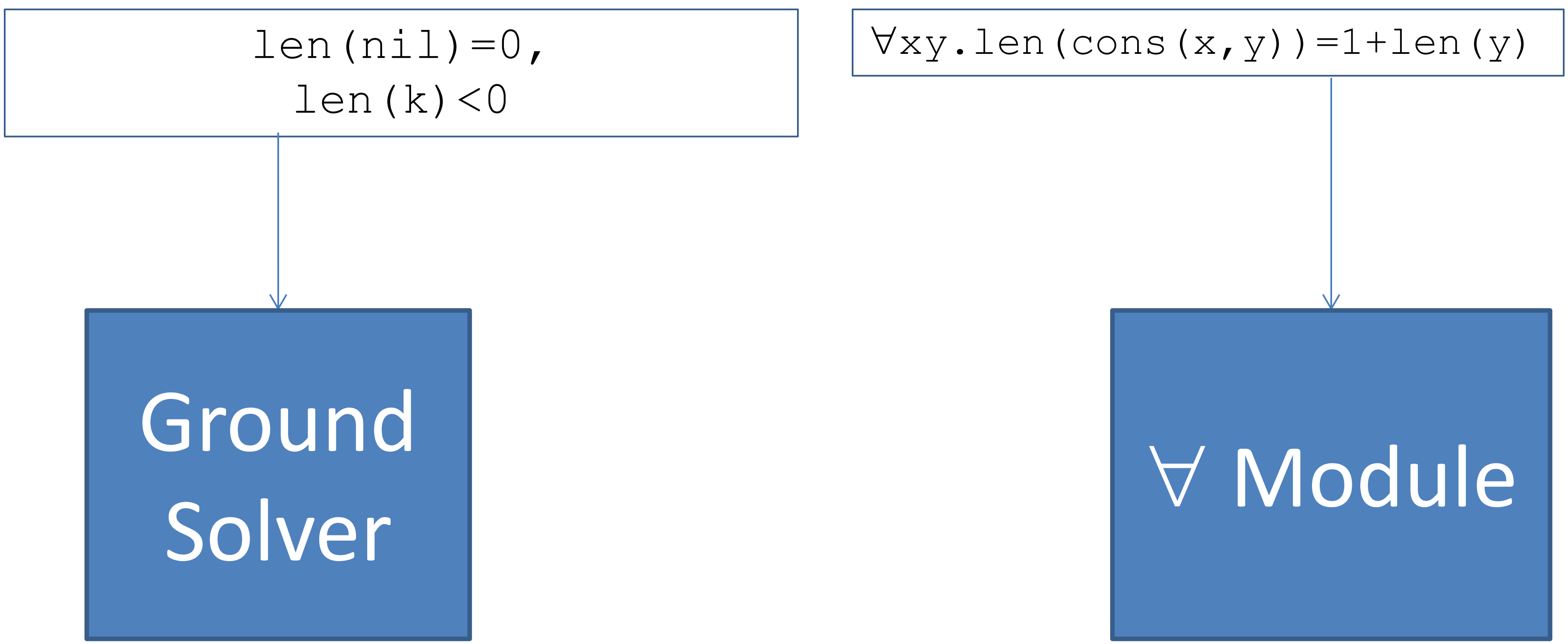
Skolemize : statement (does not) hold for fresh constant **k**

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

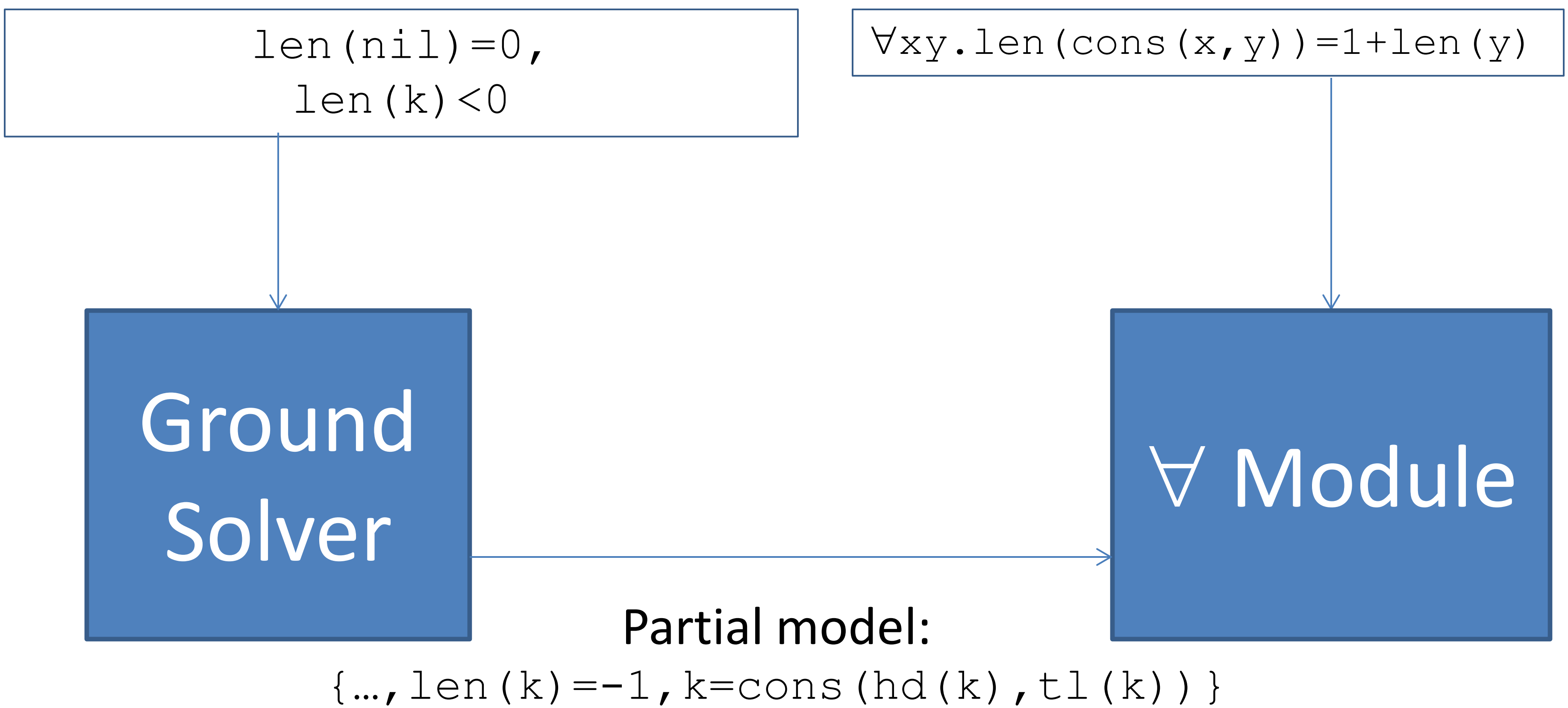
Ground Solver

∀ Module

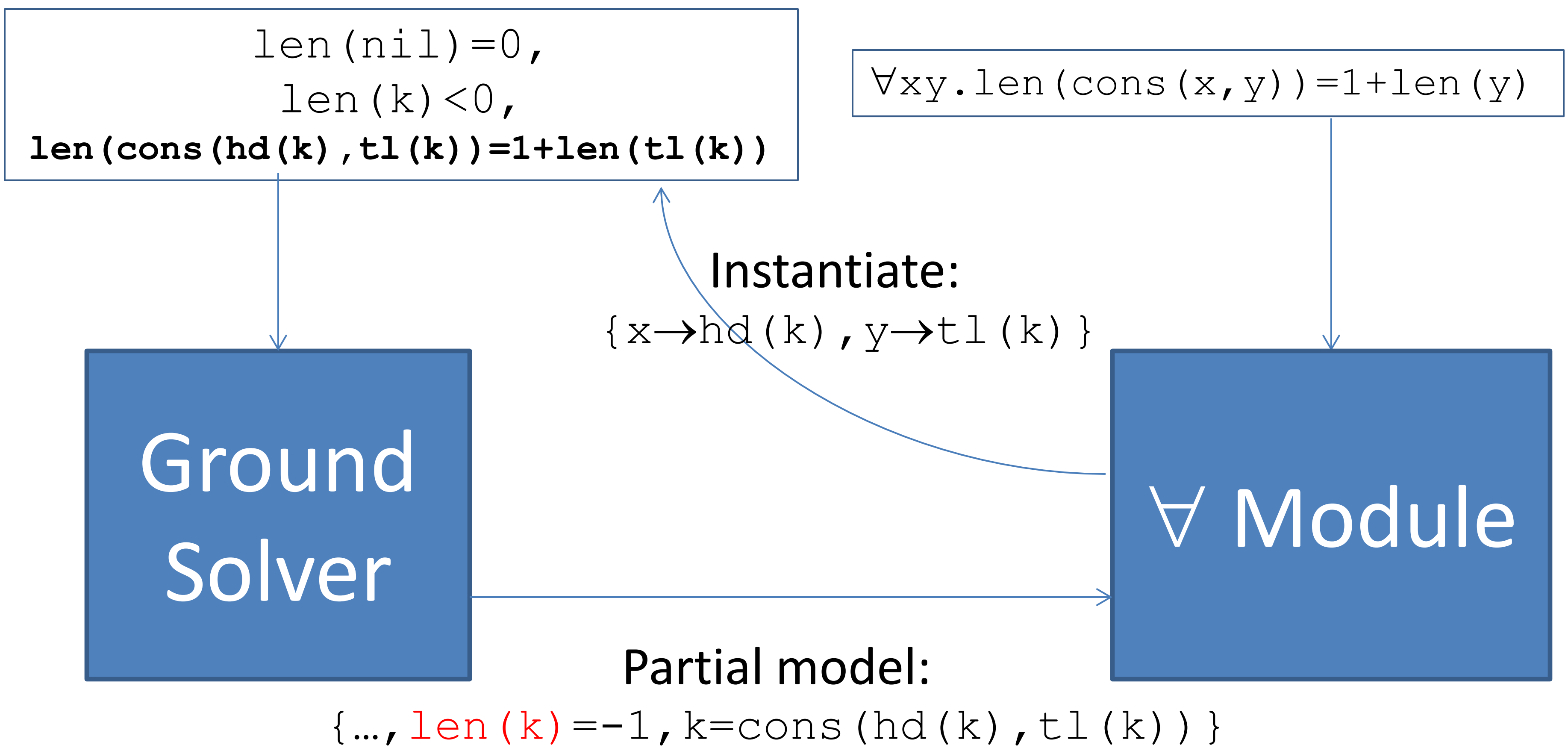
Example #2



Example #2

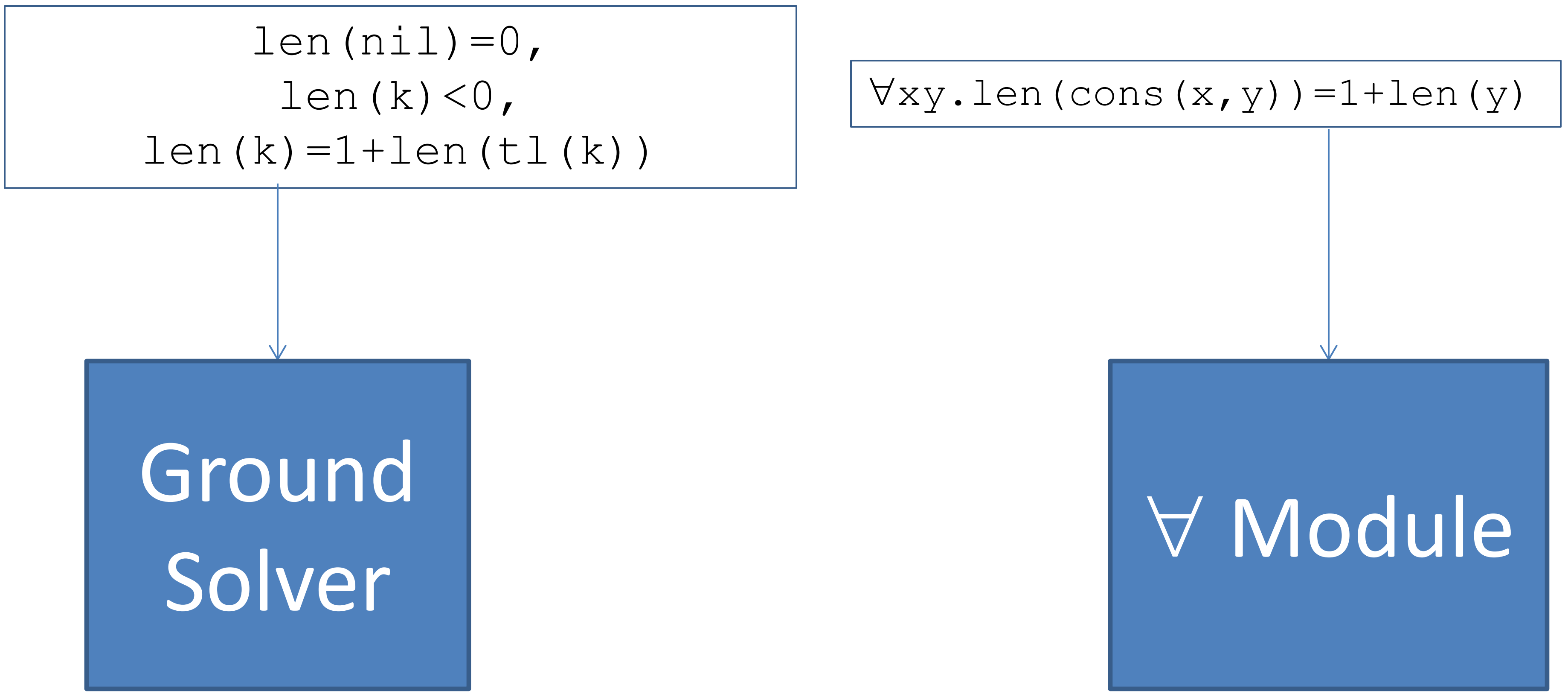

len(nil)=0,
len(k)<0
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module
Partial model:
{…,len(k)=-1,k=cons(hd(k),tl(k))}

Example #2

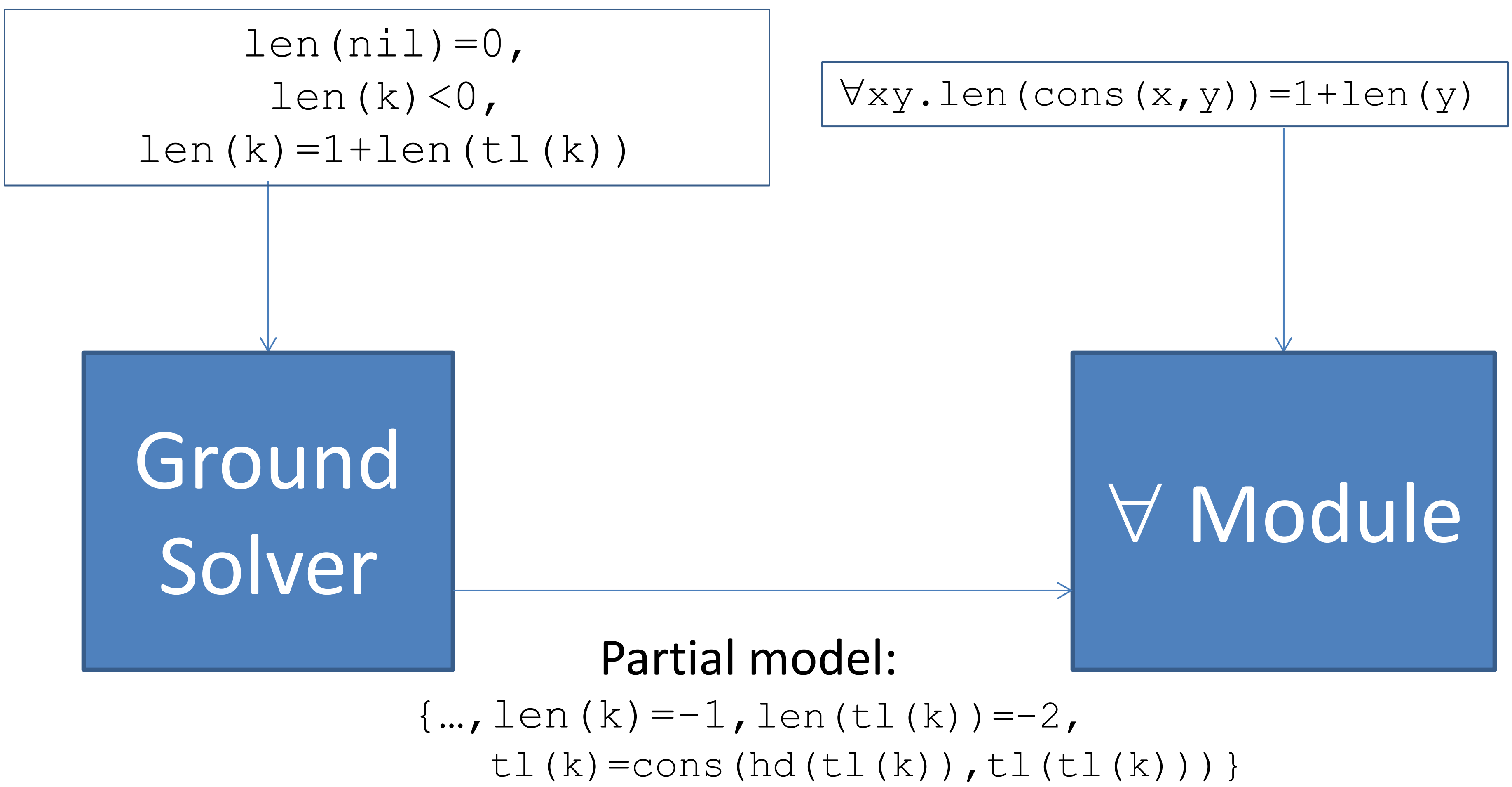


len(nil)=0,
len(k)<0,
len(cons(hd(k),tl(k))=1+len(tl(k))
∀xy.len(cons(x,y))=1+len(y)
Instantiate:
{x→hd(k),y→tl(k)}
Ground Solver
∀ Module
Partial model:
{…,len(k)=-1,k=cons(hd(k),tl(k))}

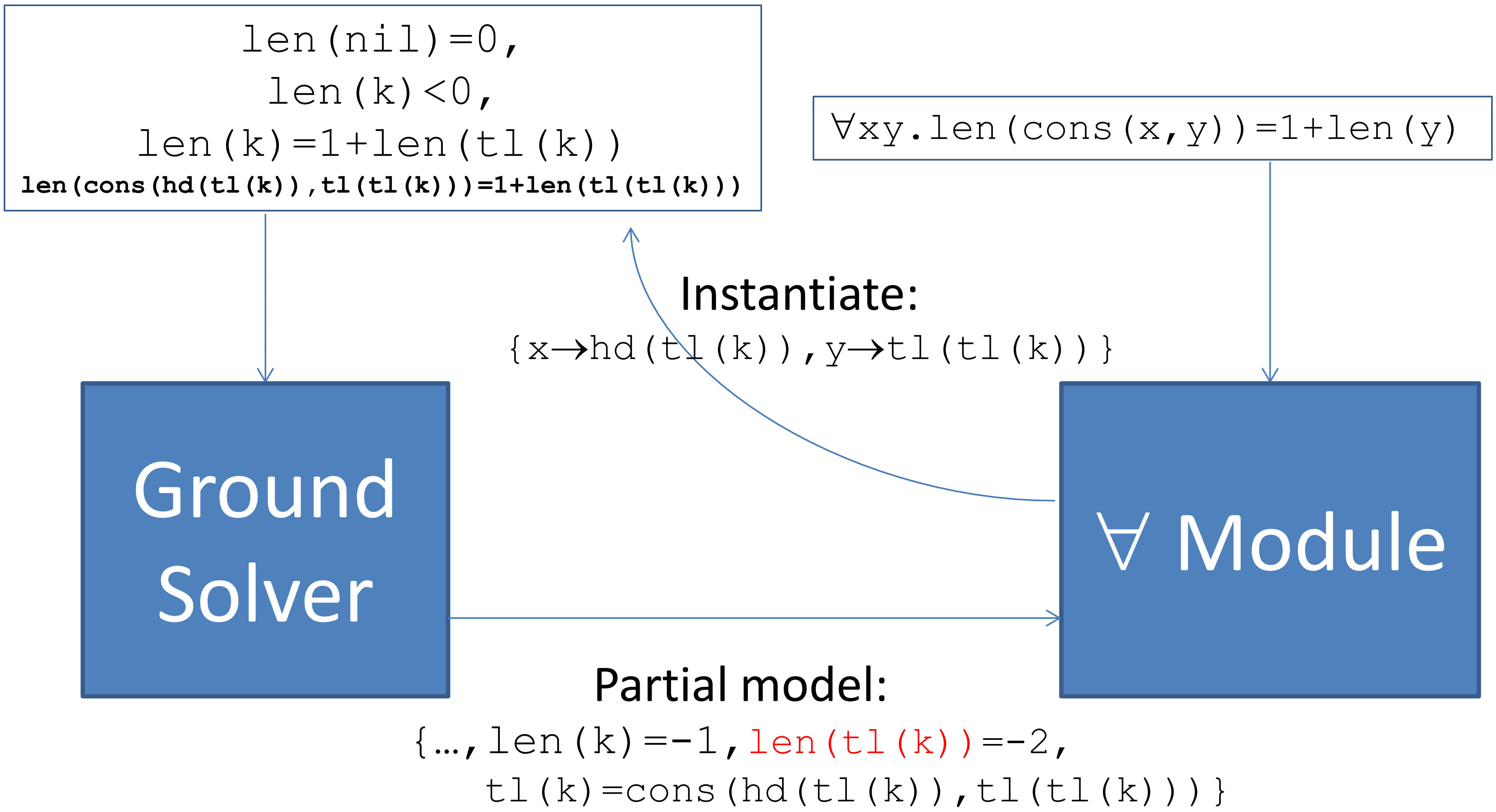
Example #2


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

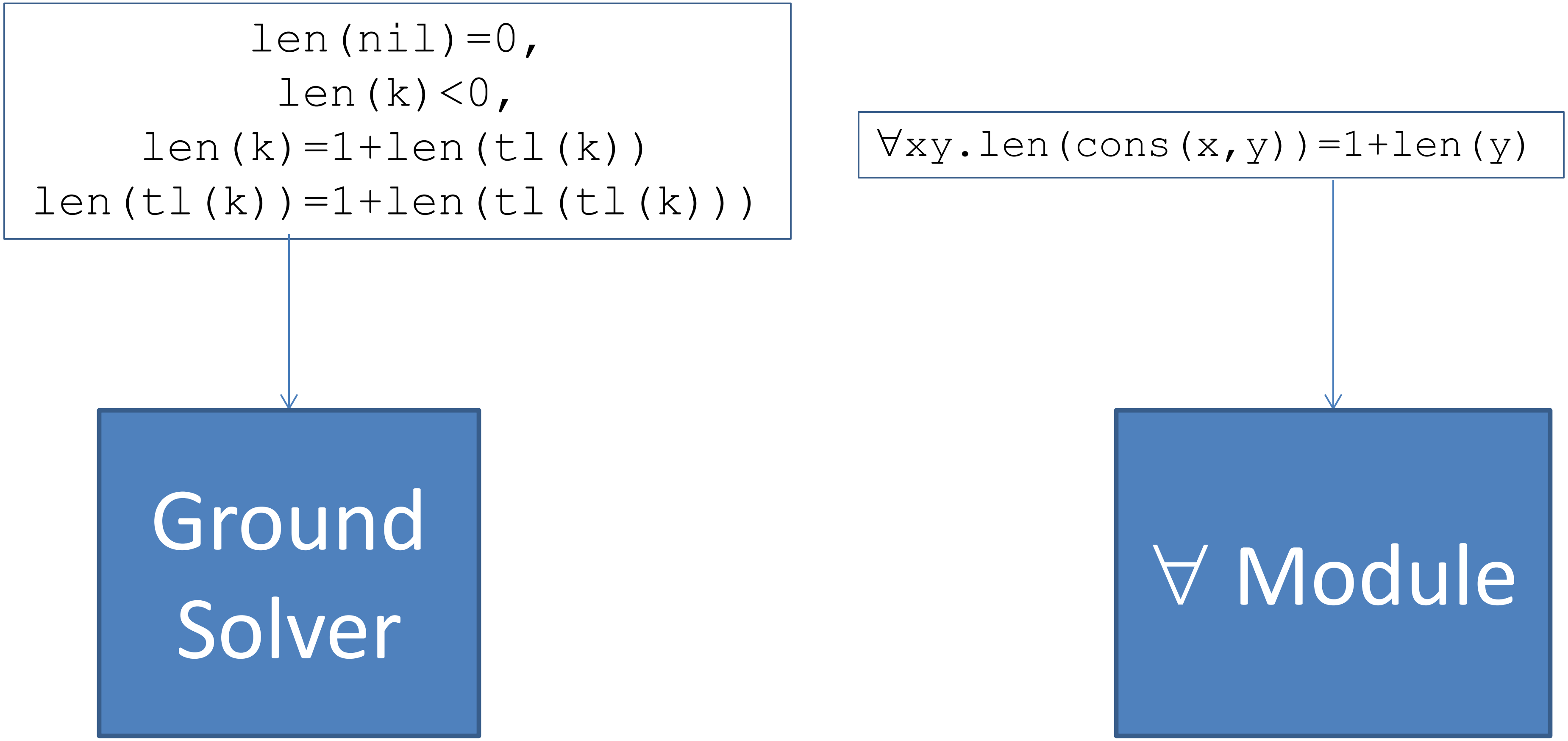
Example #2


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)=-1,len(tl(k))=-2,
tl(k)=cons(hd(tl(k)),tl(tl(k)))}

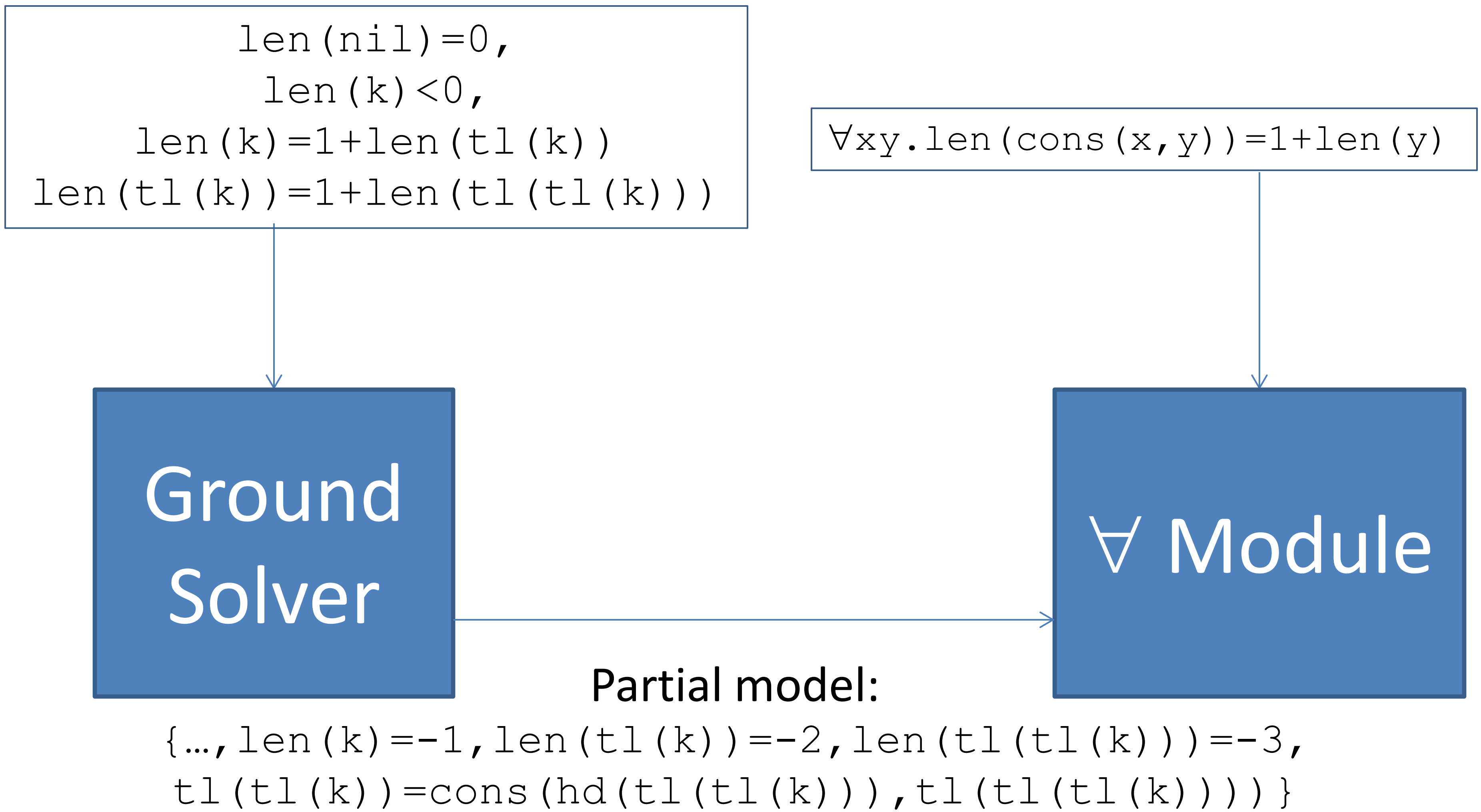
Example #2


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)=-1,len(tl(k))=-2,
tl(k)=cons(hd(tl(k)),tl(tl(k)))}

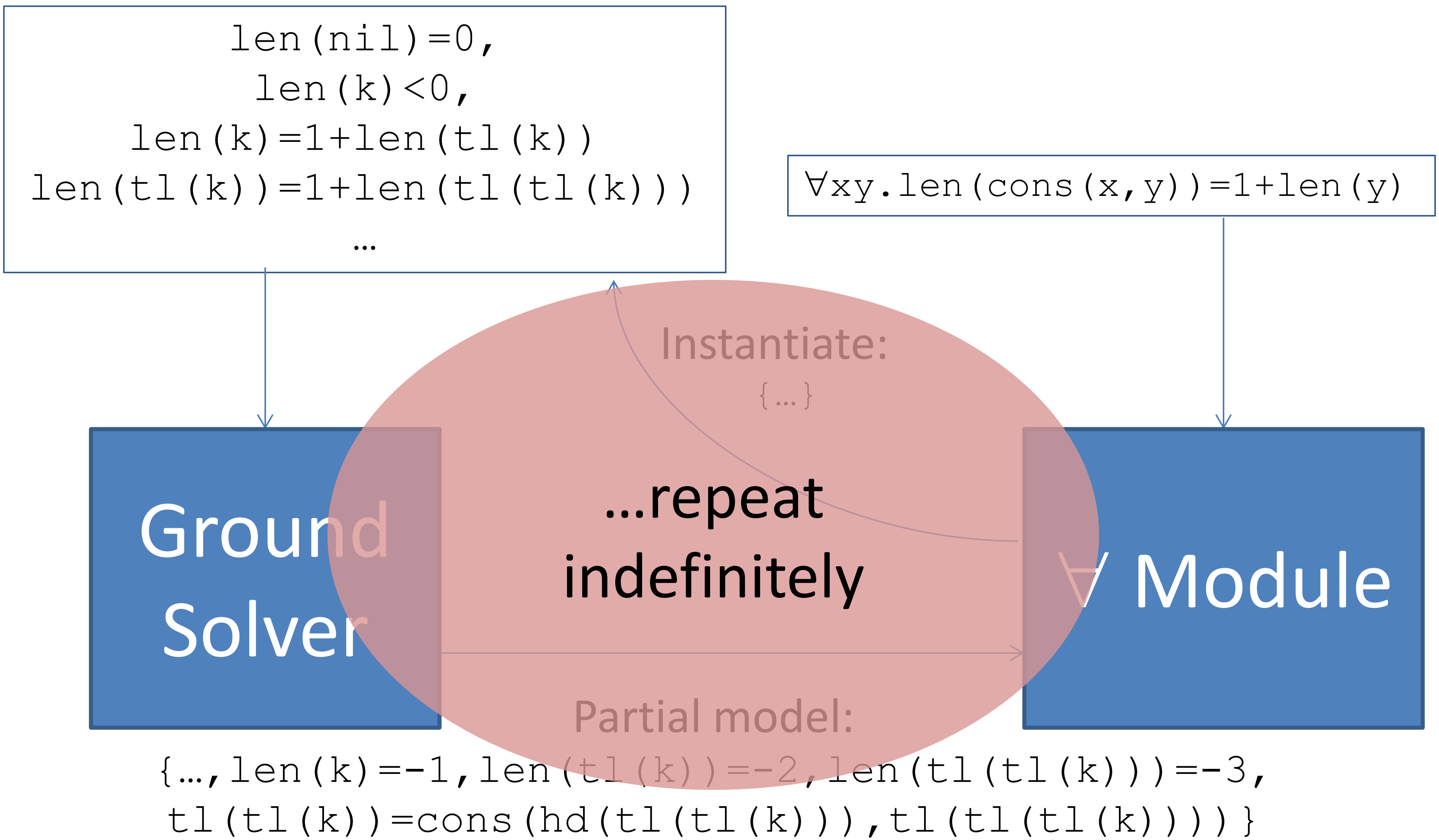
Example #2


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


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
Partial model:
{…,len(k)=-1,len(tl(k))=-2,len(tl(tl(k)))=-3,
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)=-1,len(tl(k))=-2,len(tl(tl(k)))=-3,
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]
    - SPASS/Pirate
    - …
  - 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:

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

- Assume property does not for fresh k:

$$\neg\ \mathtt{len(k)} \geq 0$$

AND

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

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

Example #2: revised

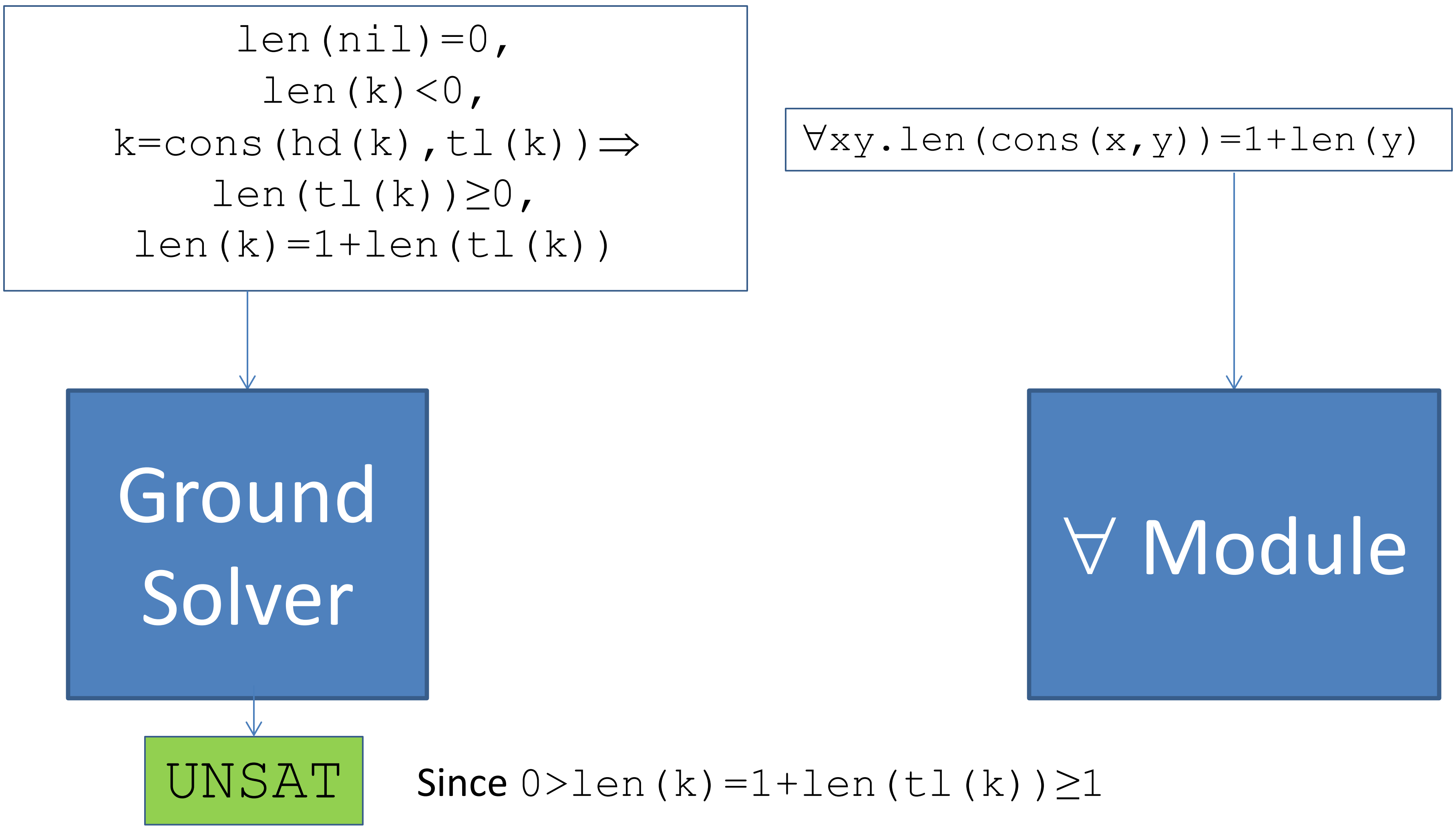
`len(nil)=0,`
`len(k)<0,`
`k=cons(hd(k),tl(k))⇒`
`len(tl(k))≥0,`
`len(k)=1+len(tl(k))`

`∀xy.len(cons(x,y))=1+len(y)`

Ground Solver

∀ Module

Example #2: revised



Skolemization with Inductive Strengthening

- General form:

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

  - 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))
```

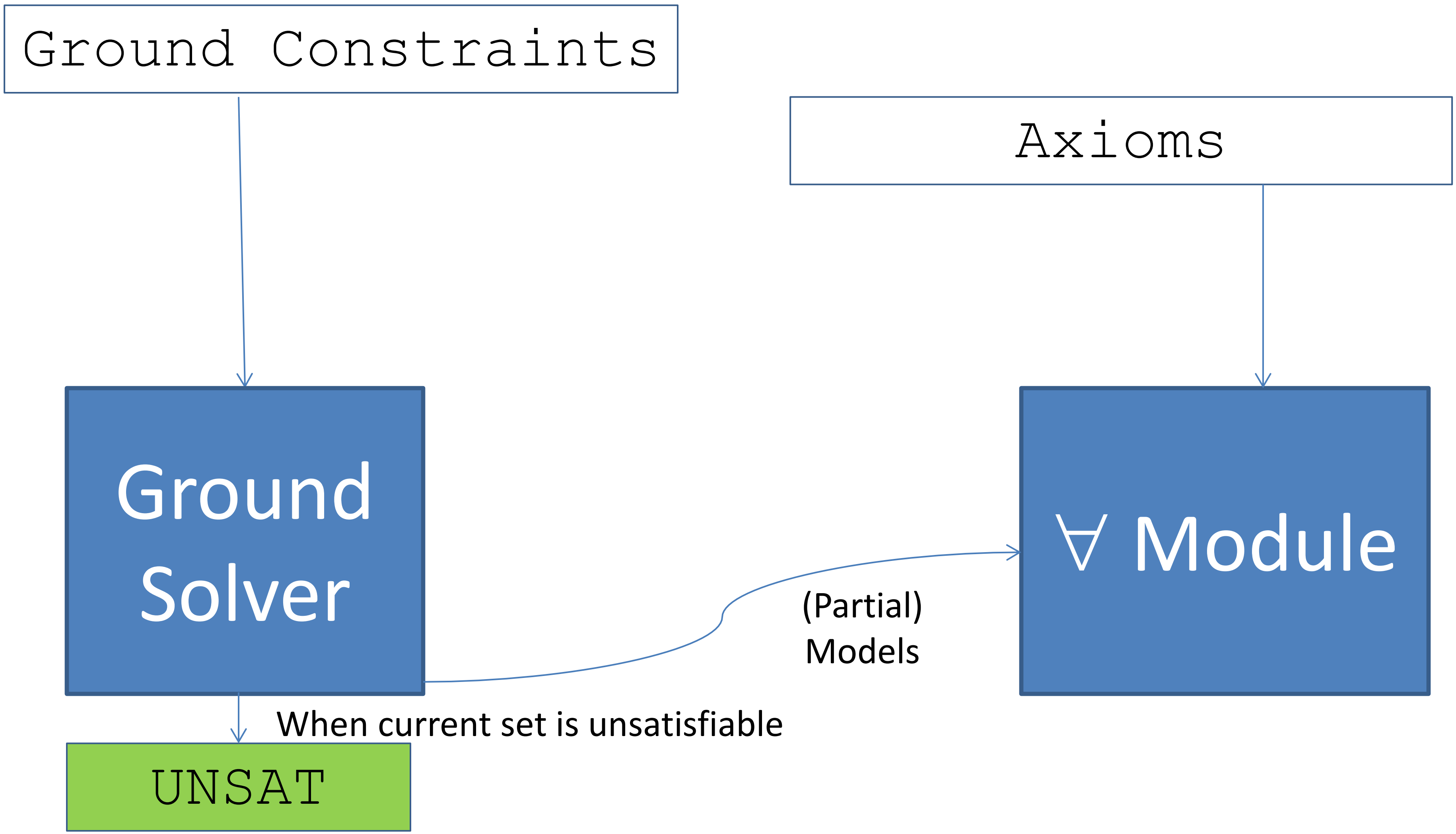
```
∀xy.plus(x,y)=plus(y,x)
```

Generating candidate subgoals

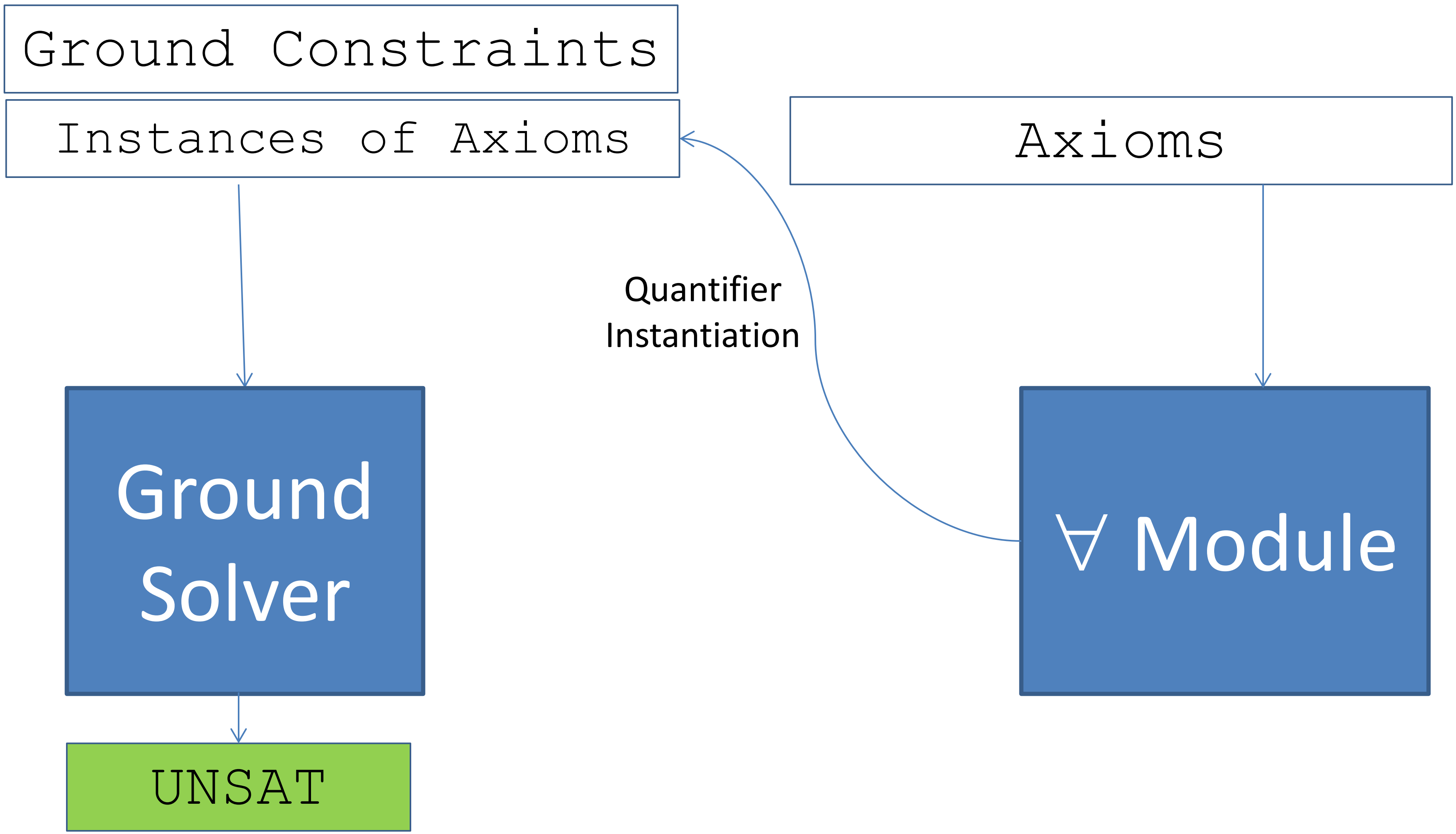
- How to generate necessary subgoals?
  - Idea: Enumerate/prove them in a principled way
    - HipSpec [Claessen et al 2013]

```
∀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))
…
```

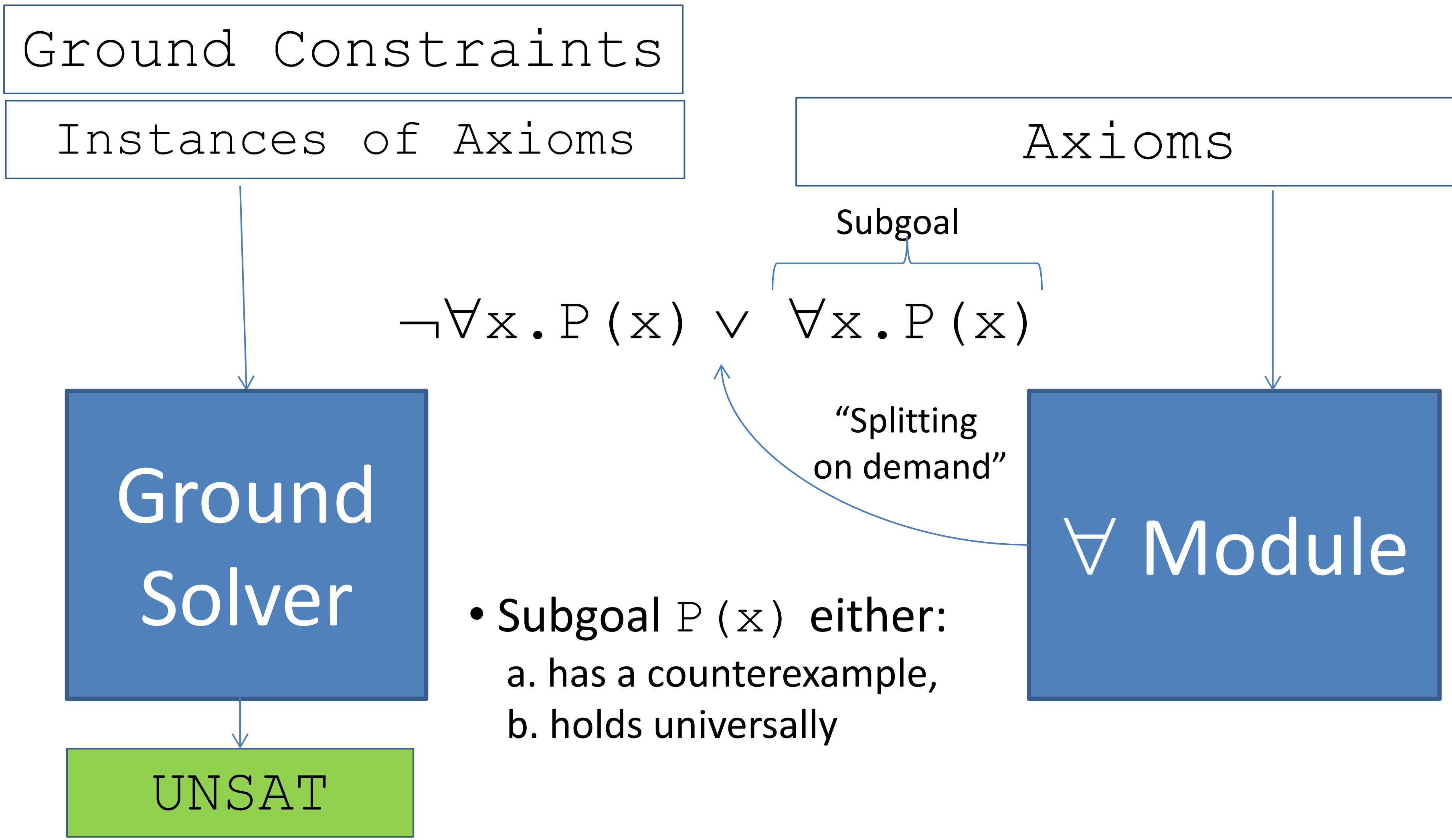
Subgoal Generation in SMT



Subgoal Generation in SMT

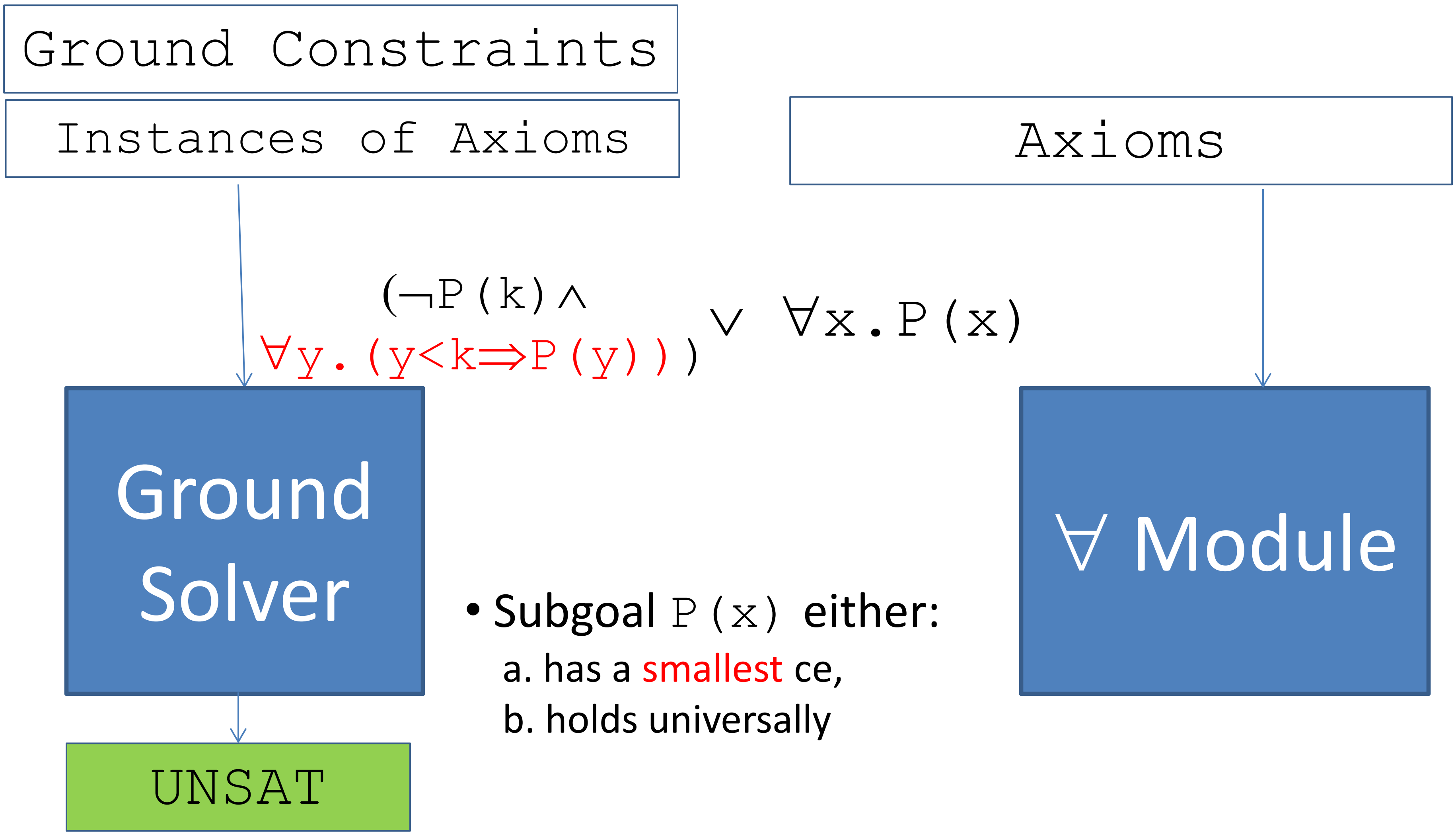


Subgoal Generation in SMT



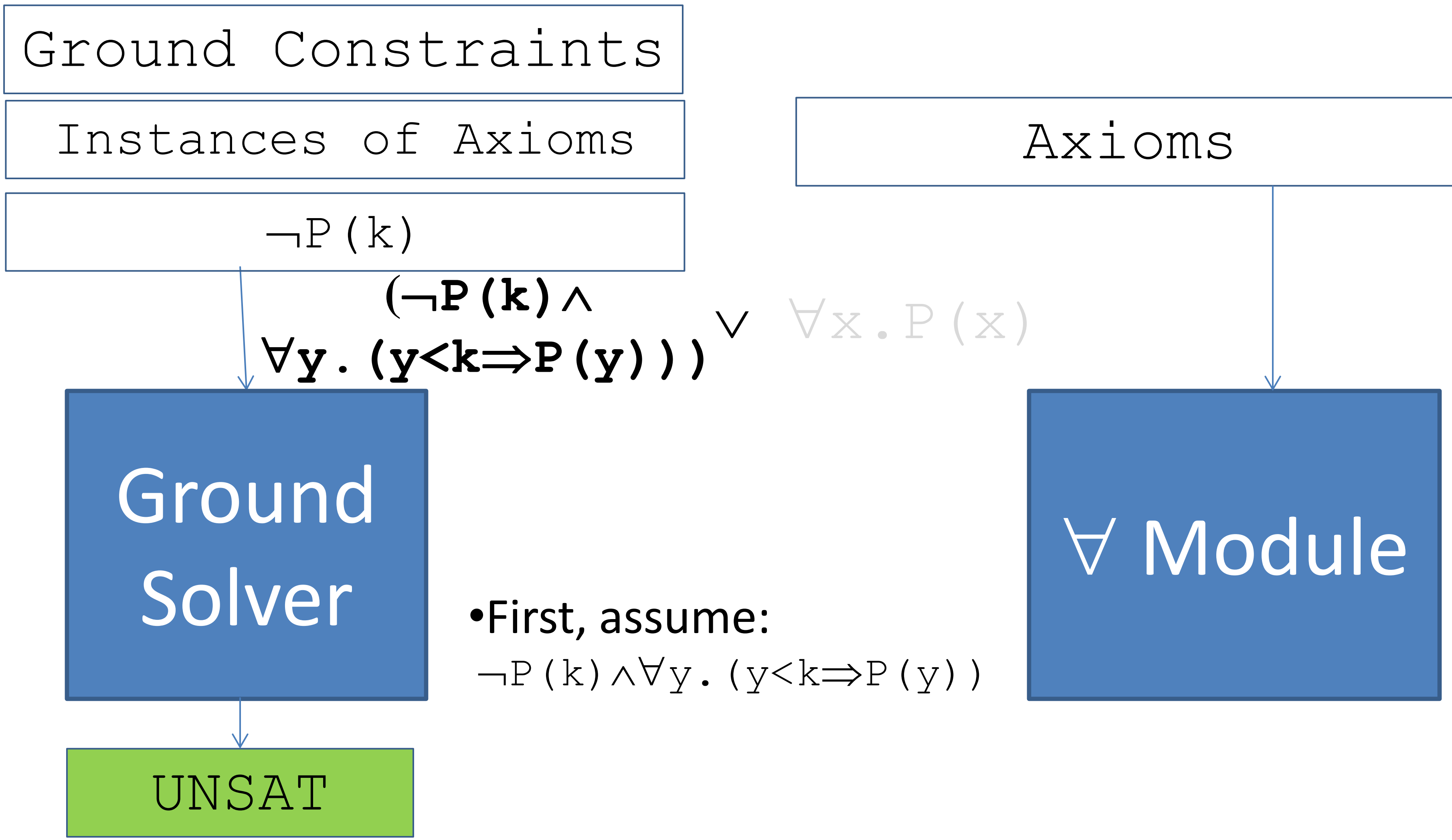
- Subgoal `P(x)` either:
  a. has a counterexample,
  b. holds universally

Subgoal Generation in SMT



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

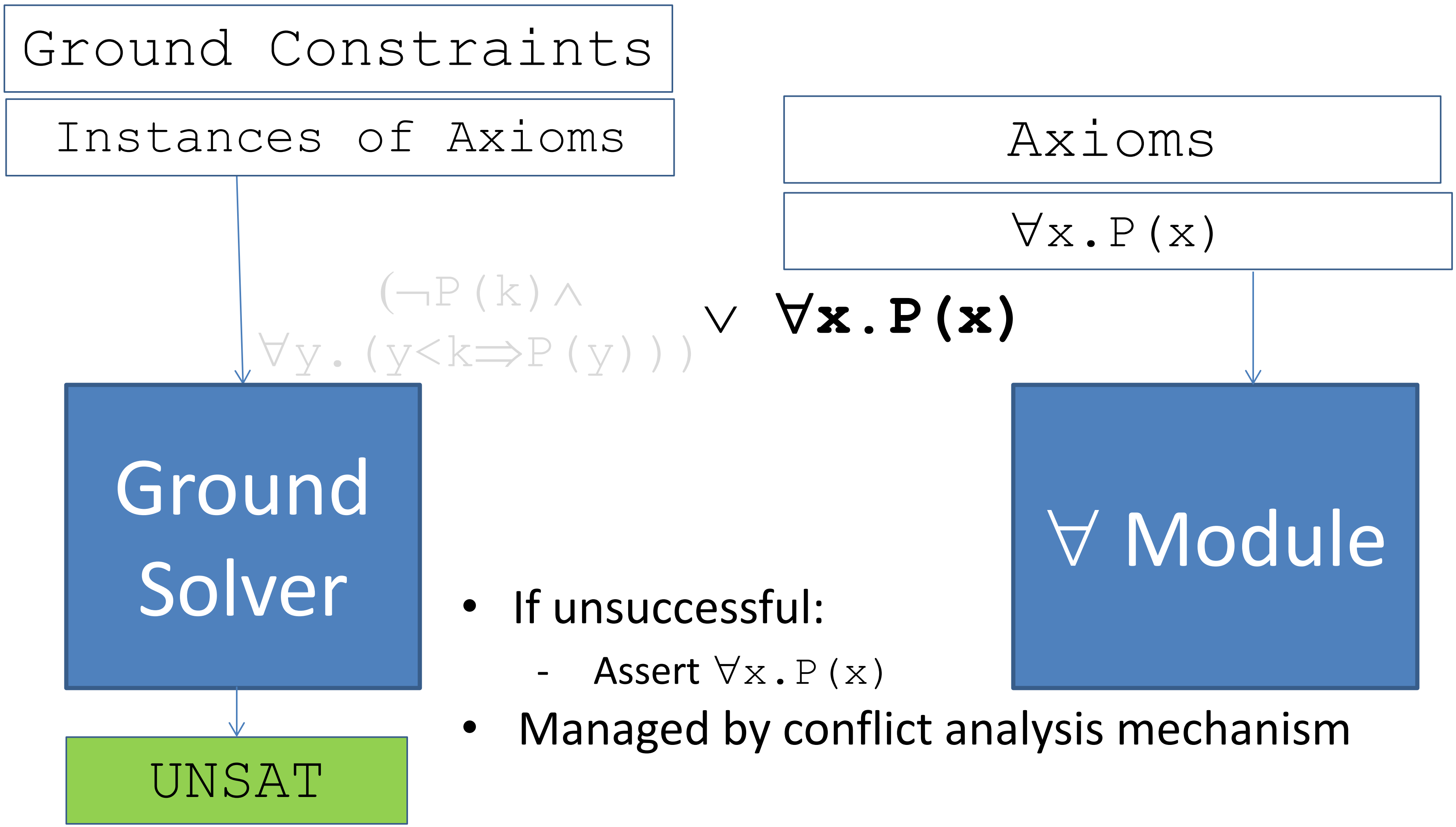
Subgoal Generation in SMT



- First, assume:

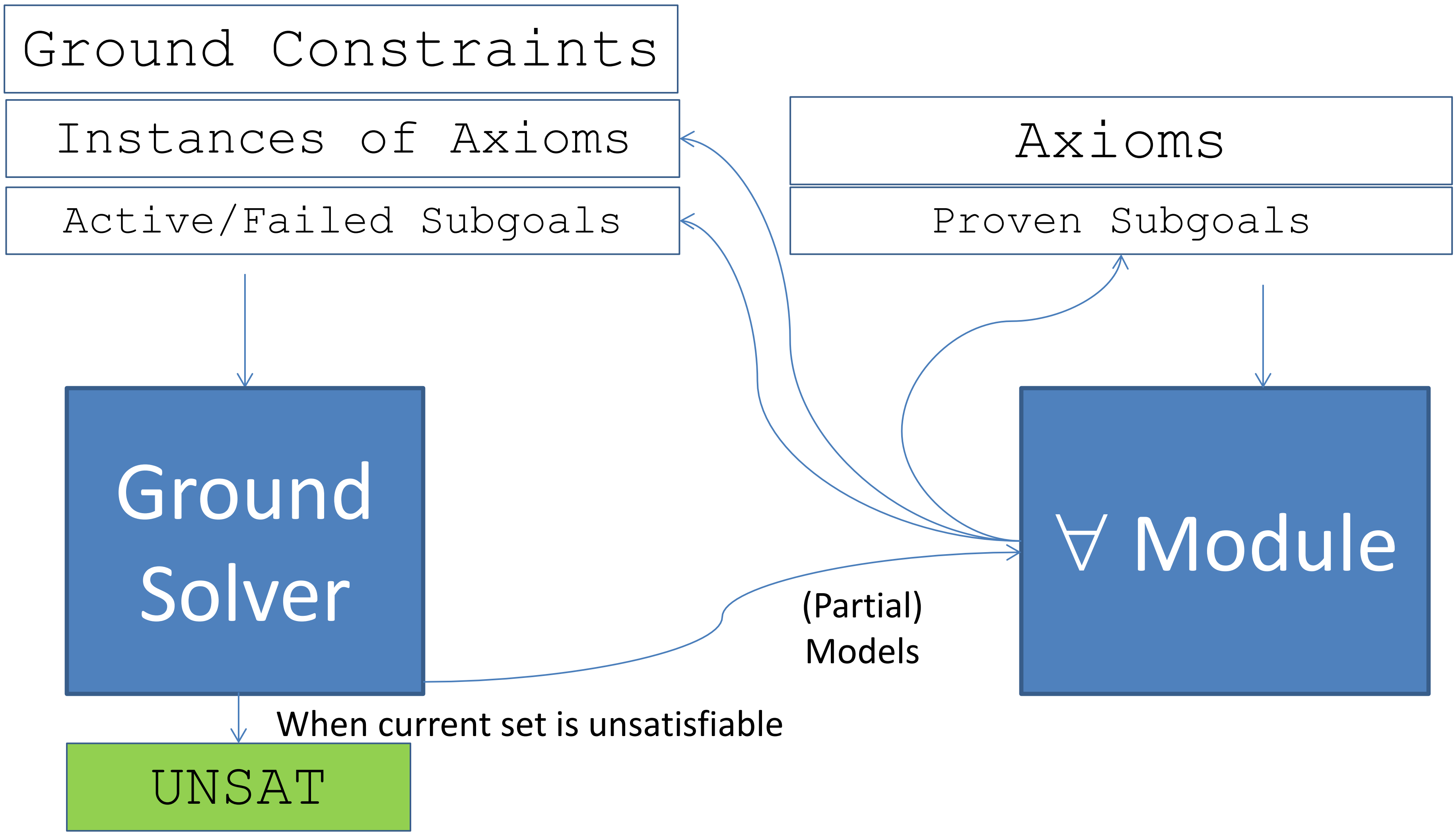
$\neg P(k) \wedge \forall y.(y<k \Rightarrow P(y))$

Subgoal Generation in SMT



- If unsuccessful:
  - 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)
  - Do not consider subgoals that are not useful

Subgoal Filtering

- Given: $\boxed{\texttt{∀x.len(rev(x))=len(x)}}$
- Filtering based on "active" symbols:

  `∀xy.count(x,y)=count(rev(x),y)`
  - Irrelevant, if conjecture is not related to "`count`"
- Filtering based on canonicity:

  `∀x.len(x)=len(app(x,nil))`
  - Redundant, if we know `∀x.x=app(x,nil)`
- Filtering based on counterexamples:

  `∀x.len(x)=len(app(x,x))`
  - Falsified, e.g. if partial model contains len(t) $\neq$ len(app(t, t))

$\Rightarrow$ Typically can remove >95-99% subgoals

Evaluation : Benchmarks

- Four benchmark sets (in SMT2):
  1. IsaPlanner [Johansson et al 2010]
  2. Clam [Ireland 1996]
  3. HipSpec [Claessen et al 2013]
  4. Leon
     - Amortized Queues, Binary search trees, Leftist Heaps
- Three encodings:
  - Base encoding
  - (2 variants of) Theory encoding
    - *Take advantage of builtin theory reasoning of SMT solver*

Base Encoding

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

```
¬∀x.len(rev(x))=len(x)
```

Negated Conjecture

Base Encoding

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

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

Necessary Subgoals for

UNSAT

```
¬∀x.len(rev(x))=len(x)
```

Theory Encoding

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

?

```
¬∀x.len(rev(x))=len(x)
```

Negated Conjecture

Theory Encoding #1

- All functions over datatypes:

```
Nat := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

$\Rightarrow$ Replace uninterp. functions with theory functions, e.g. plus $\rightarrow$ +

```
¬∀x.len(rev(x))=len(x)
```

Negated Conjecture

Theory Encoding #1

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

$\Rightarrow$ Replace uninterp. functions with theory functions, e.g. plus $\rightarrow$ +

Downside: quantifiers + theory symbols can be hard

```
¬∀x.len(rev(x))=len(x)
```

Negated Conjecture

Theory Encoding

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

?

```
¬∀x.len(rev(x))=len(x)
```

Negated Conjecture

Theory Encoding #2

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

```
toInt(zero)=0,∀x.toInt(S(x))=1+toInt(x)
∀xy.toInt(plus(x,y))=toInt(x)+toInt(y)
...
```

Mapping
toInt : Nat→Int

```
¬∀x.len(rev(x))=len(x)
```

Negated Conjecture

Theory Encoding #2

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

```
toInt(zero)=0,∀x.toInt(S(x))=1+toInt(x)
∀xy.toInt(plus(x,y))=toInt(x)+toInt(y)
...
```

Mapping toInt : Nat→Int

$\Rightarrow$ Allows SMT solver to make use of theory reasoning on demand

Above axioms imply, e.g. `∀xy.plus(x,y)=plus(y,x)`

```
¬∀x.len(rev(x))=len(x)
```

Negated Conjecture

Theory Encoding #2

- All functions over datatypes:

```
Nat  := S(P:Nat) | Z
List:= cons(hd:Int,tl:List) | nil
```

Datatype Definitions

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

Function Definitions

```
toInt(zero)=0,∀x.toInt(S(x))=1+toInt(x)
∀xy.toInt(plus(x,y))=toInt(x)+toInt(y)
...
```

Mapping toInt : Nat→Int

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

Necessary Subgoals for

```
¬∀x.len(rev(x))=len(x)
```

UNSAT

Results : SMT solvers

	base	th1	th2
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 **th2** 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)

Results: Subgoal Generation

- Conjecture:

$$\boxed{\forall xn.\mathtt{count(n,sort(x))=count(n,x)}}$$

$\Rightarrow$ *Number of times n occurs in a list is unchanged after sorting*

- We thought it would require subgoals:

$$\forall xn.\mathtt{count(n,insert(n,x))=count(n,x)+1}$$
$$\forall xnm.\mathtt{n}\neq\mathtt{m}\Rightarrow\mathtt{count(n,insert(m,x))=count(n,x)}$$

- CVC4 instead found the sufficient subgoal:

$$\boxed{\forall xnm.\mathtt{count(n,insert(m,x))=count(n,cons(m,x))}}$$

$\Rightarrow$ *Proved original conjecture fully automatically with a simpler proof*

Comparison with Other Provers

	Isaplanner	Clam	HipSpec	Leon
cvc4+ig:th2	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
- CVC4 fairly competitive

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
- Better comparison with other tools
- 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