Induction in CVC4

Andrew Reynolds

December 3rd, 2014

Overview

- Satisfiability Modulo Theories (SMT)
  - SMT solver CVC4
- Induction techniques in CVC4 [Reynolds/Kuncak, 2015]
  - Inductive strengthening
  - Subgoal generation
  - Leveraging theory reasoning
- Experiments

Automated Reasoning

- Historically, automated reasoning meant uniform proof procedures for FOL
- More recent trend is decidable fragments
  - Equality
  - Arithmetic
  - Data structures (arrays, lists, records)
  - …

Automated Reasoning

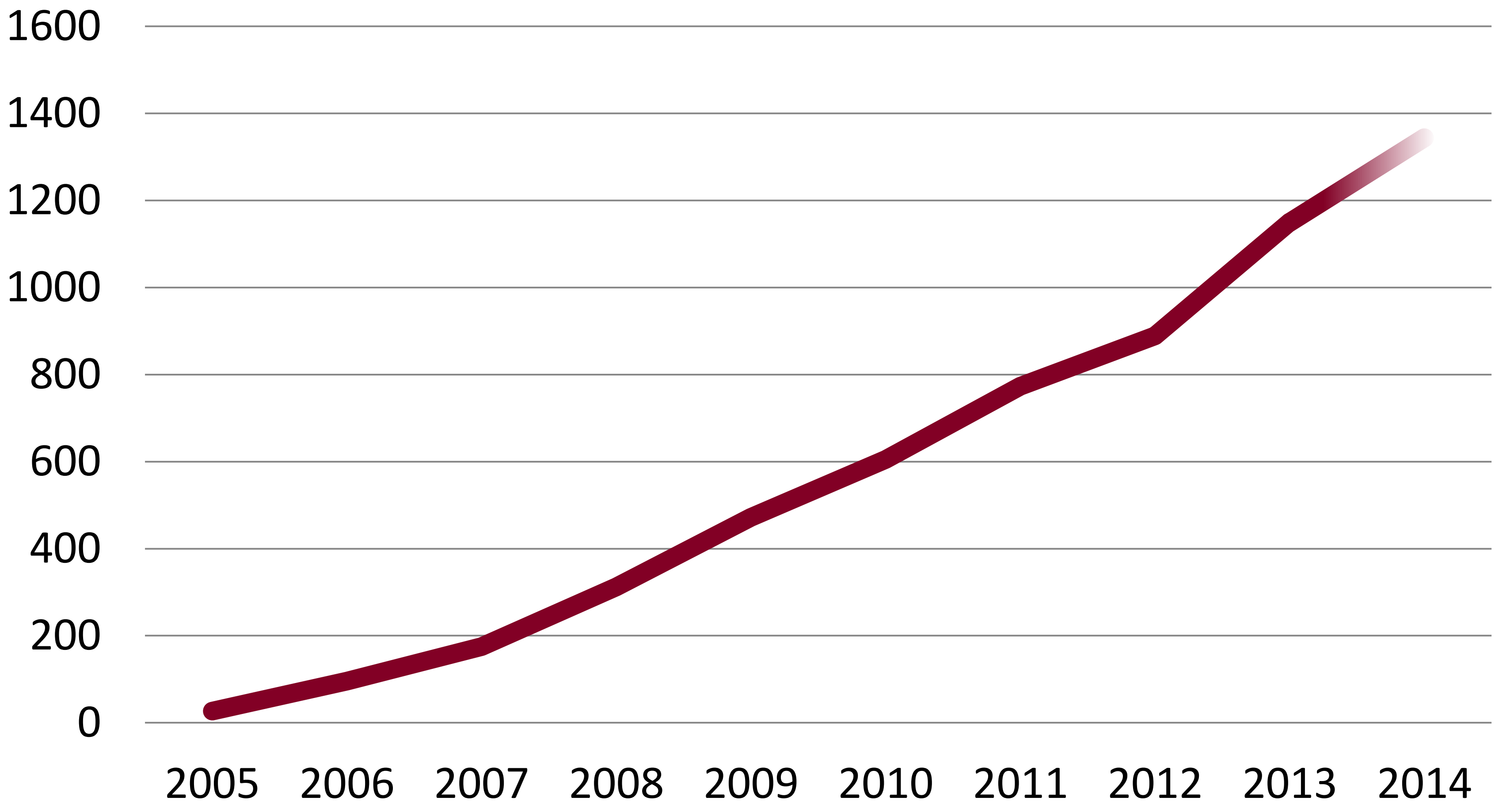
- Examples
  - SAT – propositional, Boolean reasoning
    - Efficient
    - Expressive (NP) but involved encodings

$$(A \vee \neg B) \wedge (\neg A \vee C \vee D)$$

  - SMT – first order, Boolean + DS reasoning
    - Loss of efficiency
    - Improves expressivity and scalability

$$(x+1>0 \vee \neg a[x]=b) \wedge (\neg P(y) \vee y=z)$$

Articles mentioning SMT over time



Applications of SMT

- Extended static checking
- Predicate abstraction
- Model checking
- Scheduling
- Test generation
- Synthesis
- Verification

CVC4

- State-of-the-art SMT Solver
  - Developed over last 5 years as successor of CVC3
- Supports many theories:
  - Arithmetic, Arrays, Bitvectors, UF
  - Inductive/Co-inductive Datatypes
  - New : Strings, Floating Points, Finite Sets
- Has strong performance:
  - Placed 1st in 14 of 32 divisions of SMT-COMP
  - Won TFA division of CASC J7
  - Competitive for many common SMT uses

The CVC4 Team

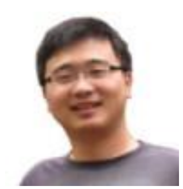

Clark Barrett (NYU)
Cesare Tinelli (U Iowa)



Kshitij Bansal (NYU)
François Bobot (CEA)
Chris Conway (Google)



Morgan Deters (NYU)
Liana Hadarean (NYU)
Dejan Jovanović (SRI)


Tim King (Verimag)
Tianyi Liang (U Iowa)
Andrew Reynolds (EPFL)

CVC4 : Quantifiers

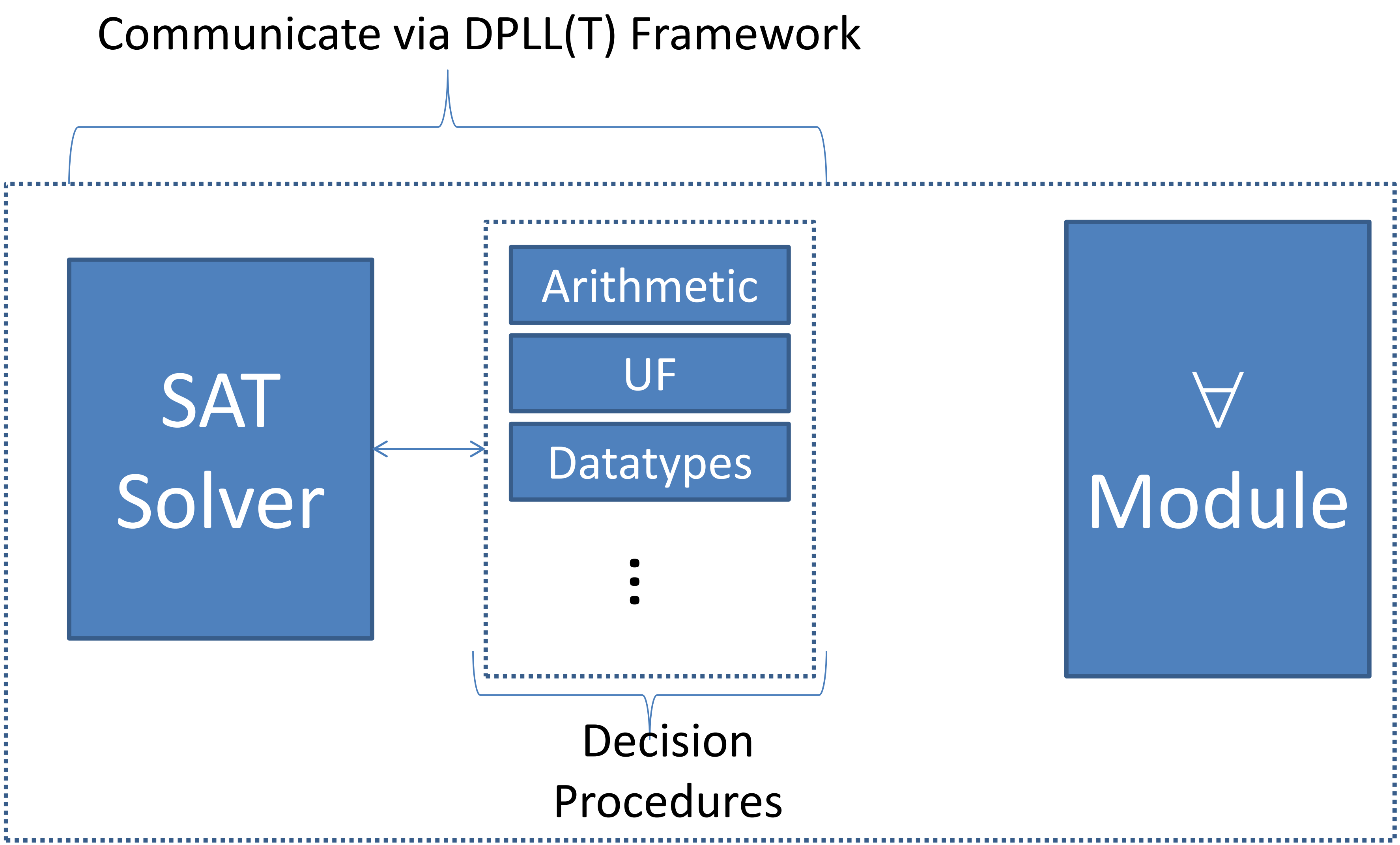
- Handles (universally) quantified formulas

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

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

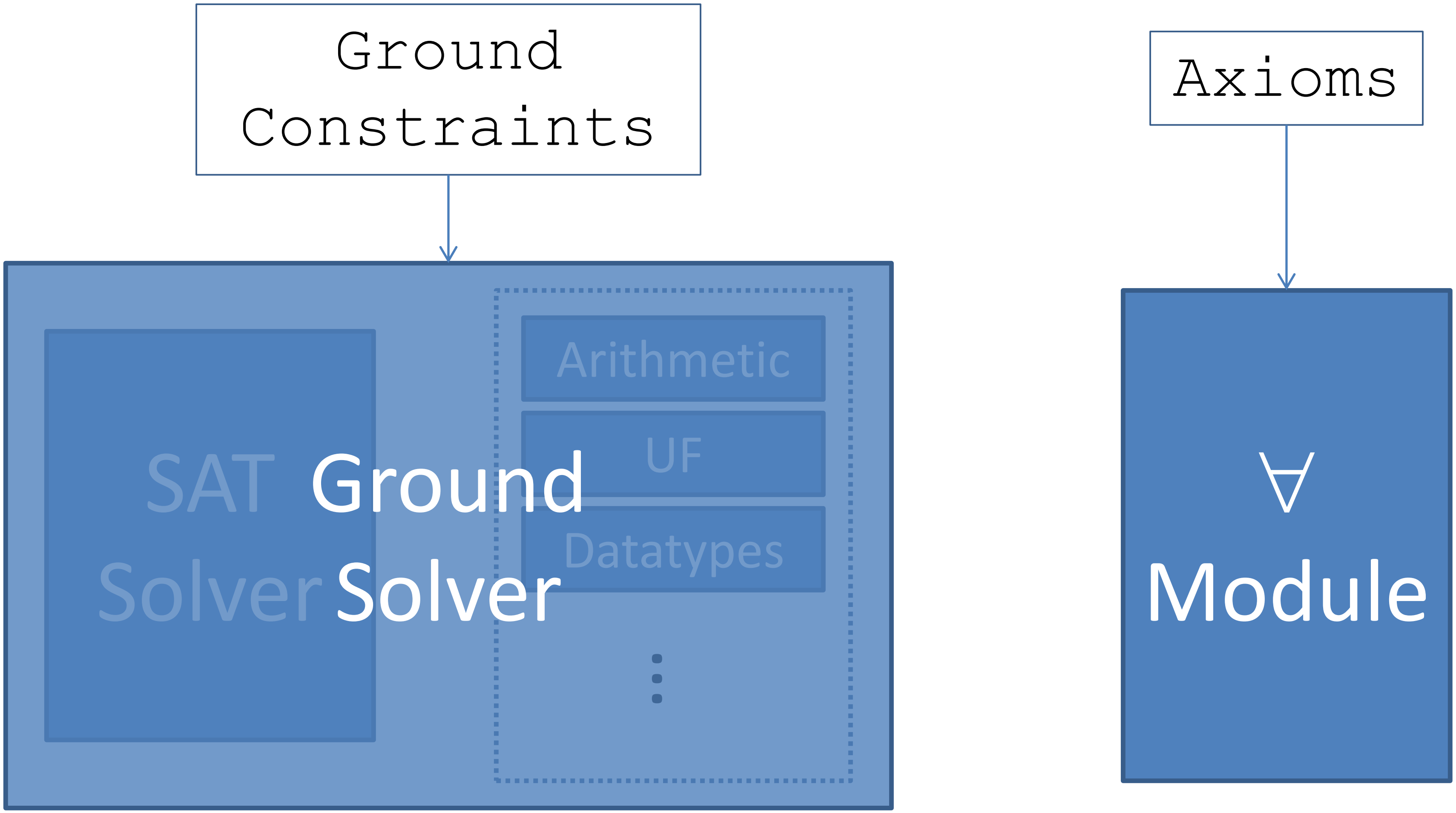
- CVC4 handles quantifiers by:
  - Heuristic instantiation (E-matching)
  - Conflict-based instantiation [FMCAD 2014]
  - Finite model finding/model-based instantiation [CADE/CAV 2013]
  - Rewrite Rules
  - New: Syntax-Guided Synthesis, Induction [VMCAI 2015]

SMT Solver

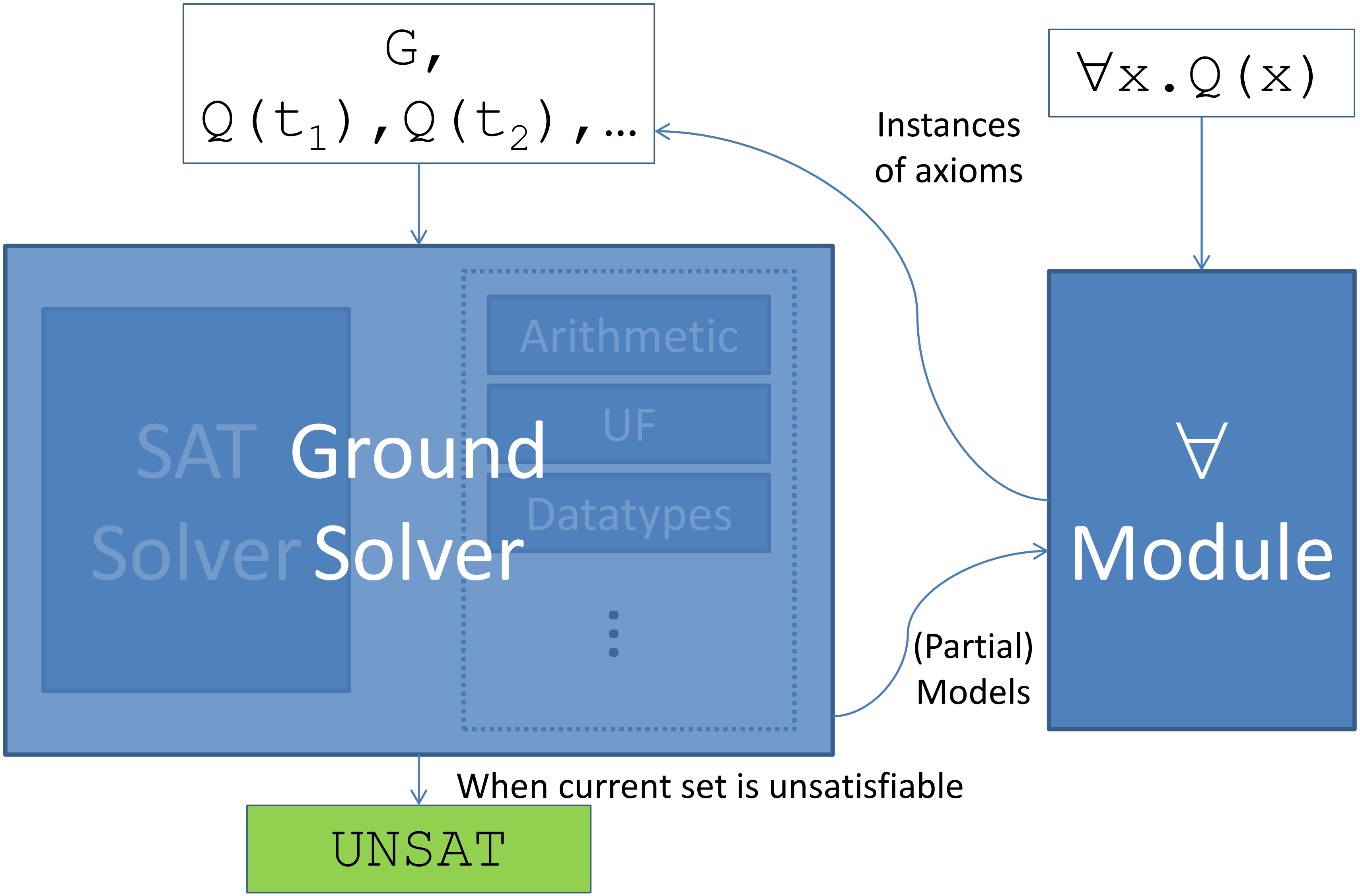


Communicate via DPLL(T) Framework
SAT Solver
Arithmetic
UF
Datatypes
⋮
∀ Module
Decision Procedures

SMT Solver


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

SMT Solver


G,
Q(t1),Q(t2),…
∀x.Q(x)
Instances
of axioms
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

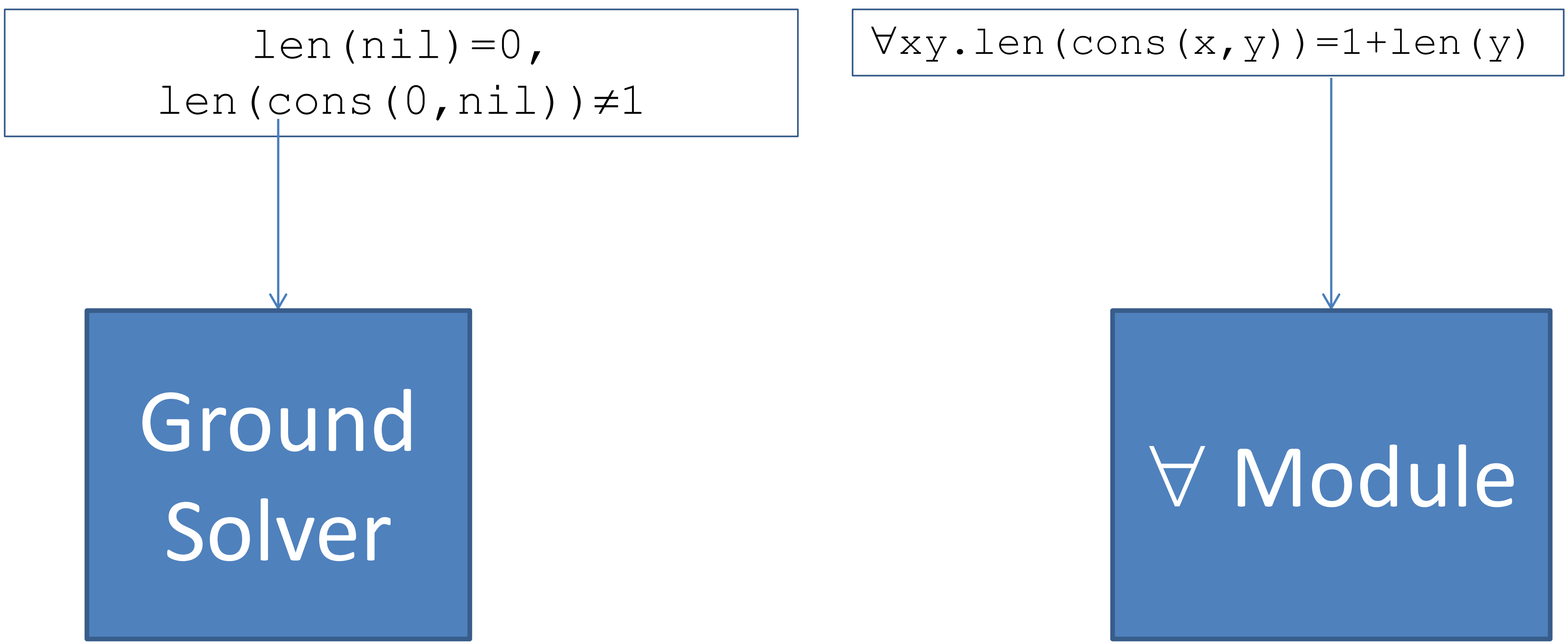
```
¬len(cons(0,nil))=1
```

(Negated) Conjecture

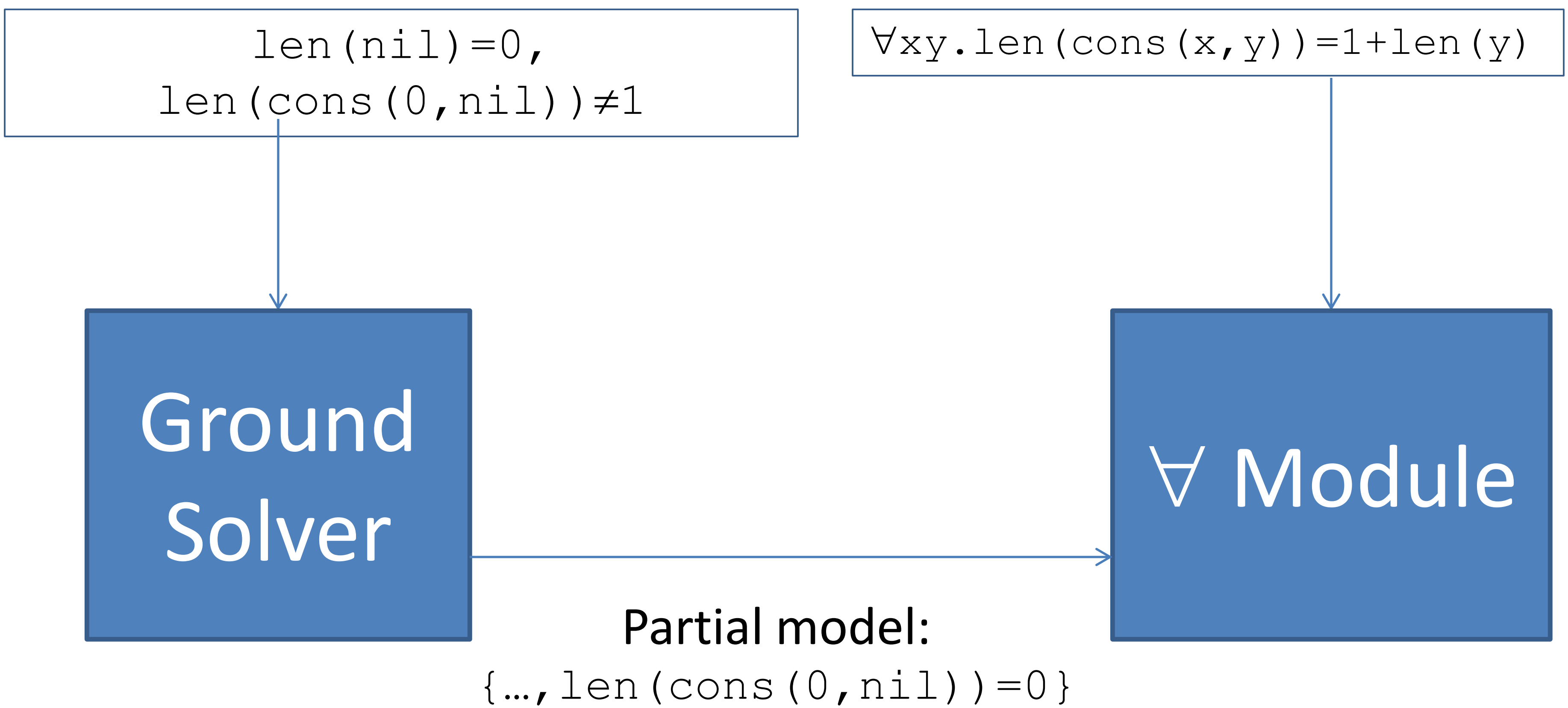
Ground Solver

∀ Module

Example #1

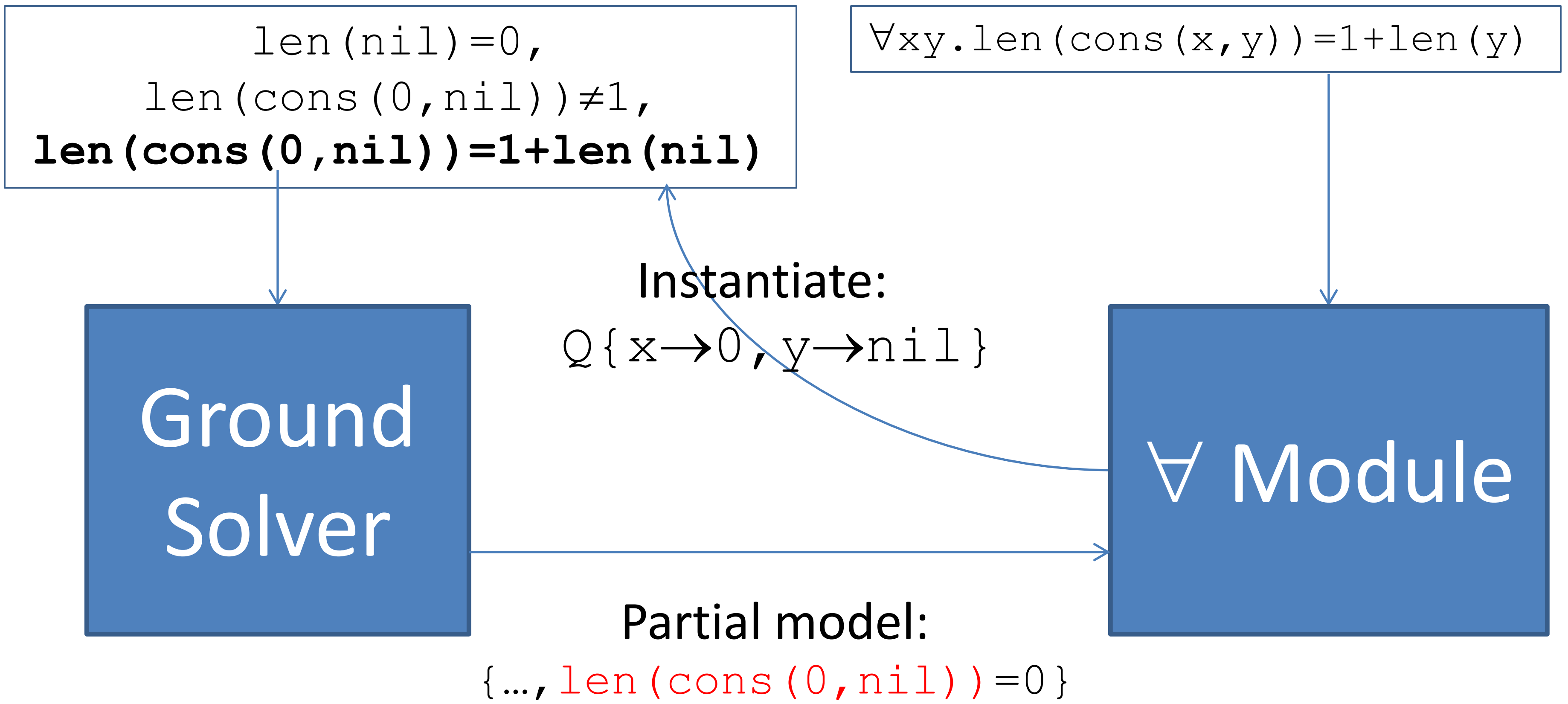

len(nil)=0,
len(cons(0,nil))≠1
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module

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

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

```
¬∀x.len(x)≥0
```

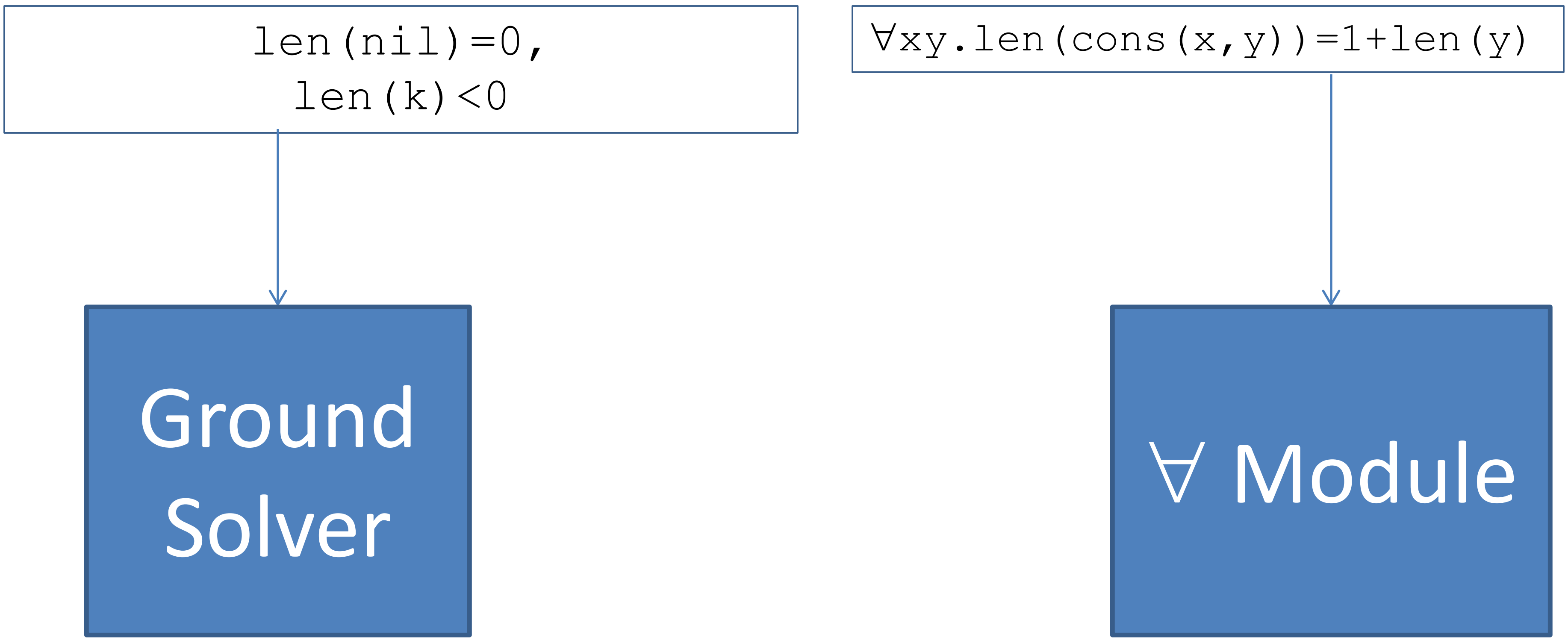
Skolemize : statement (does not) hold for fresh constant **k**

```
¬  len(k)≥0
```

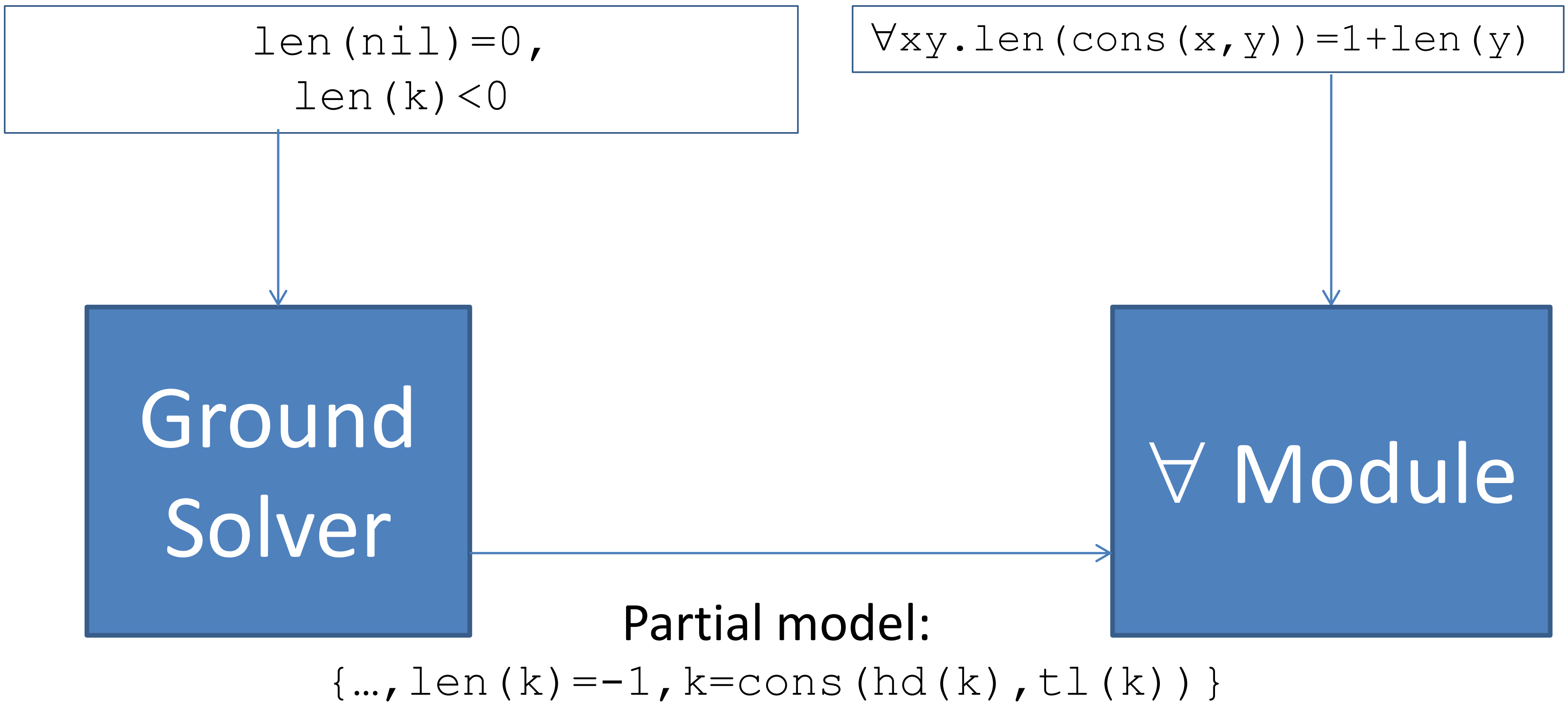
Ground Solver

∀ Module

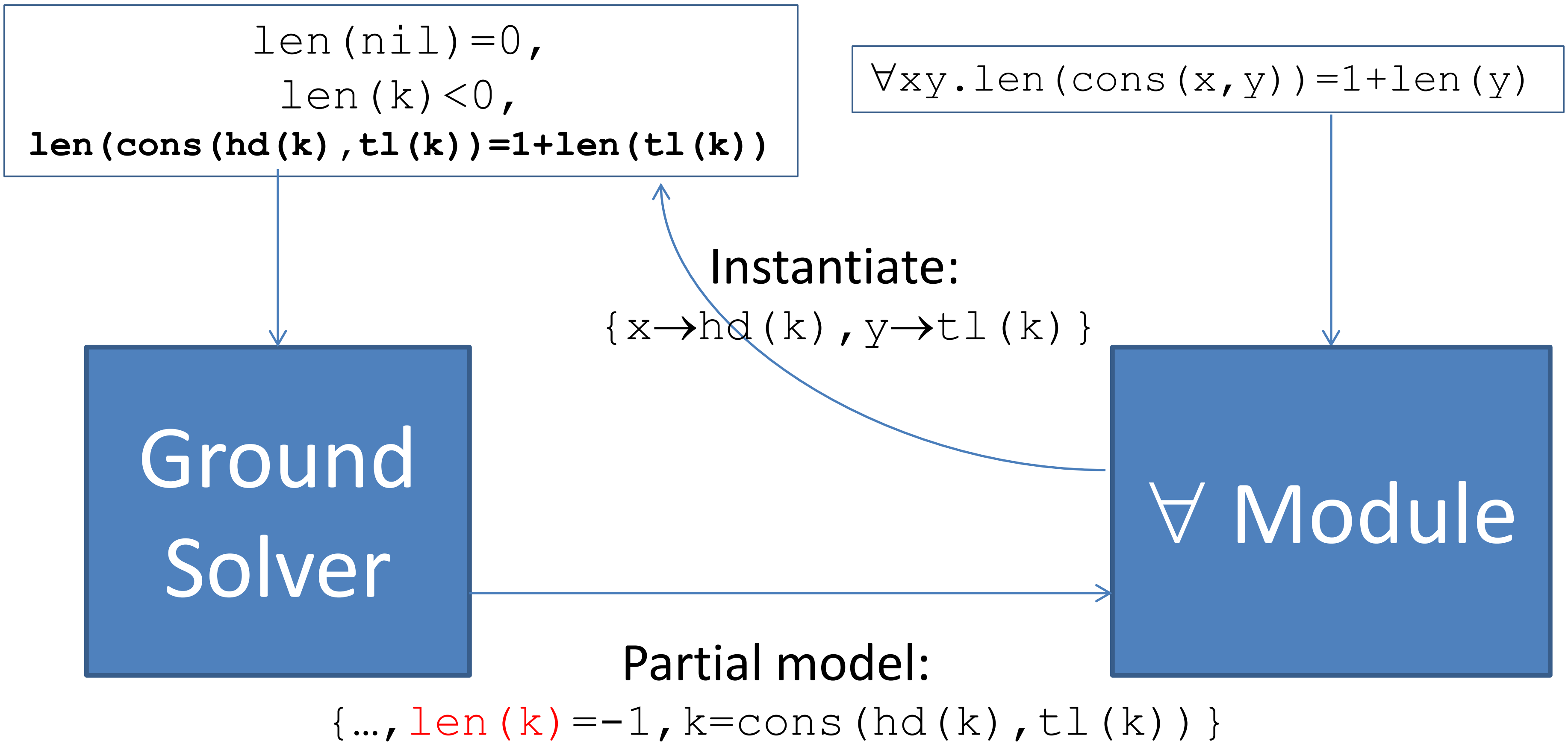
Example #2



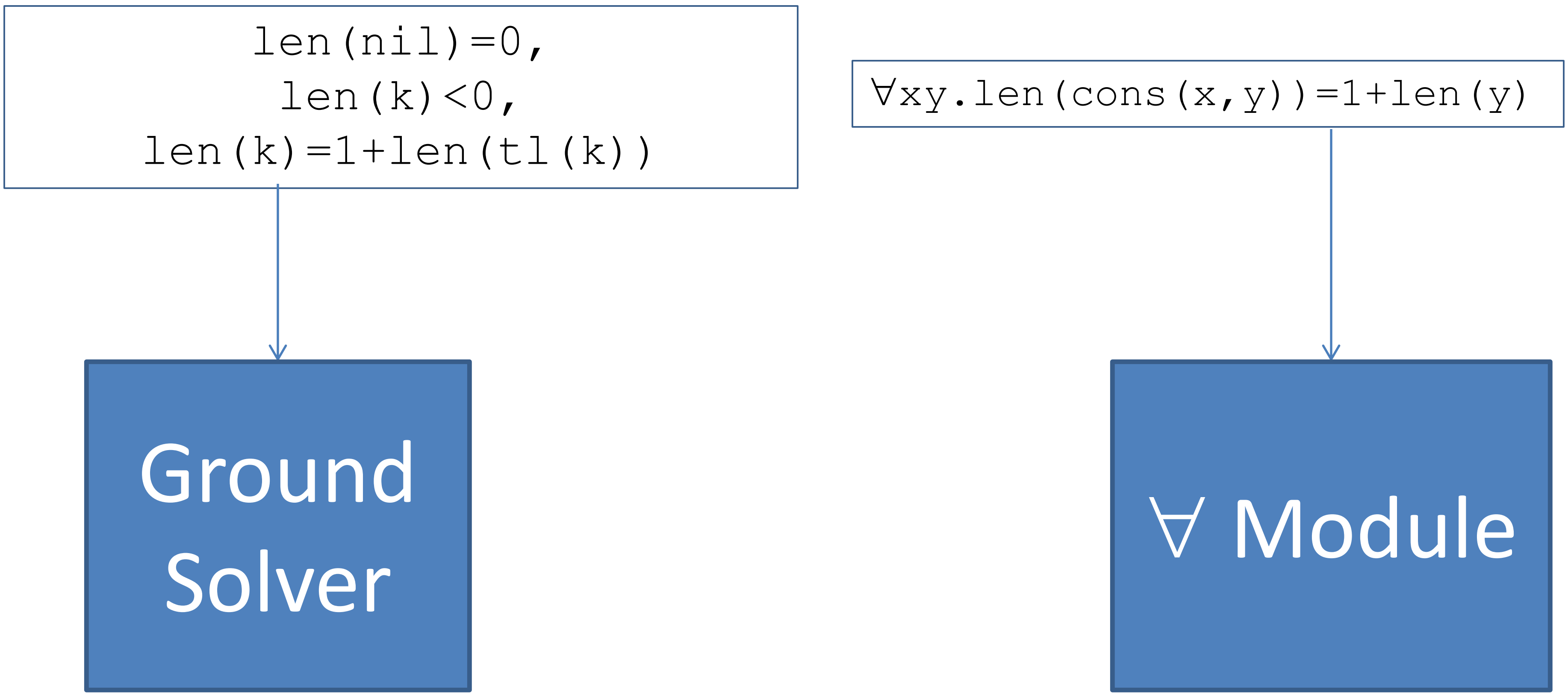
Example #2


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

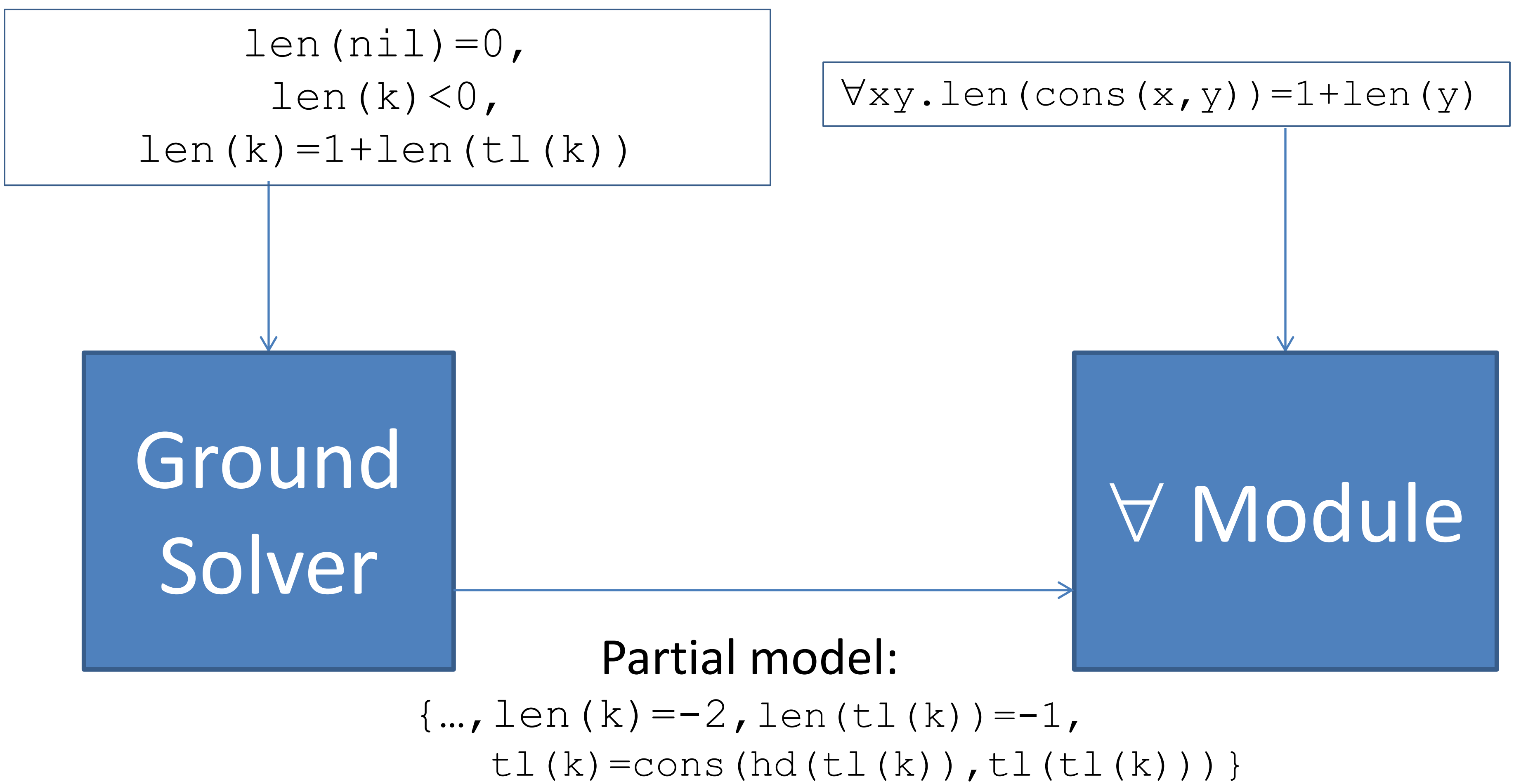

len(nil)=0,
len(k)<0,
len(cons(hd(k),tl(k))=1+len(tl(k))
∀xy.len(cons(x,y))=1+len(y)
Instantiate:
{x→hd(k),y→tl(k)}
Ground Solver
∀ Module
Partial model:
{…,len(k)=-1,k=cons(hd(k),tl(k))}

Example #2



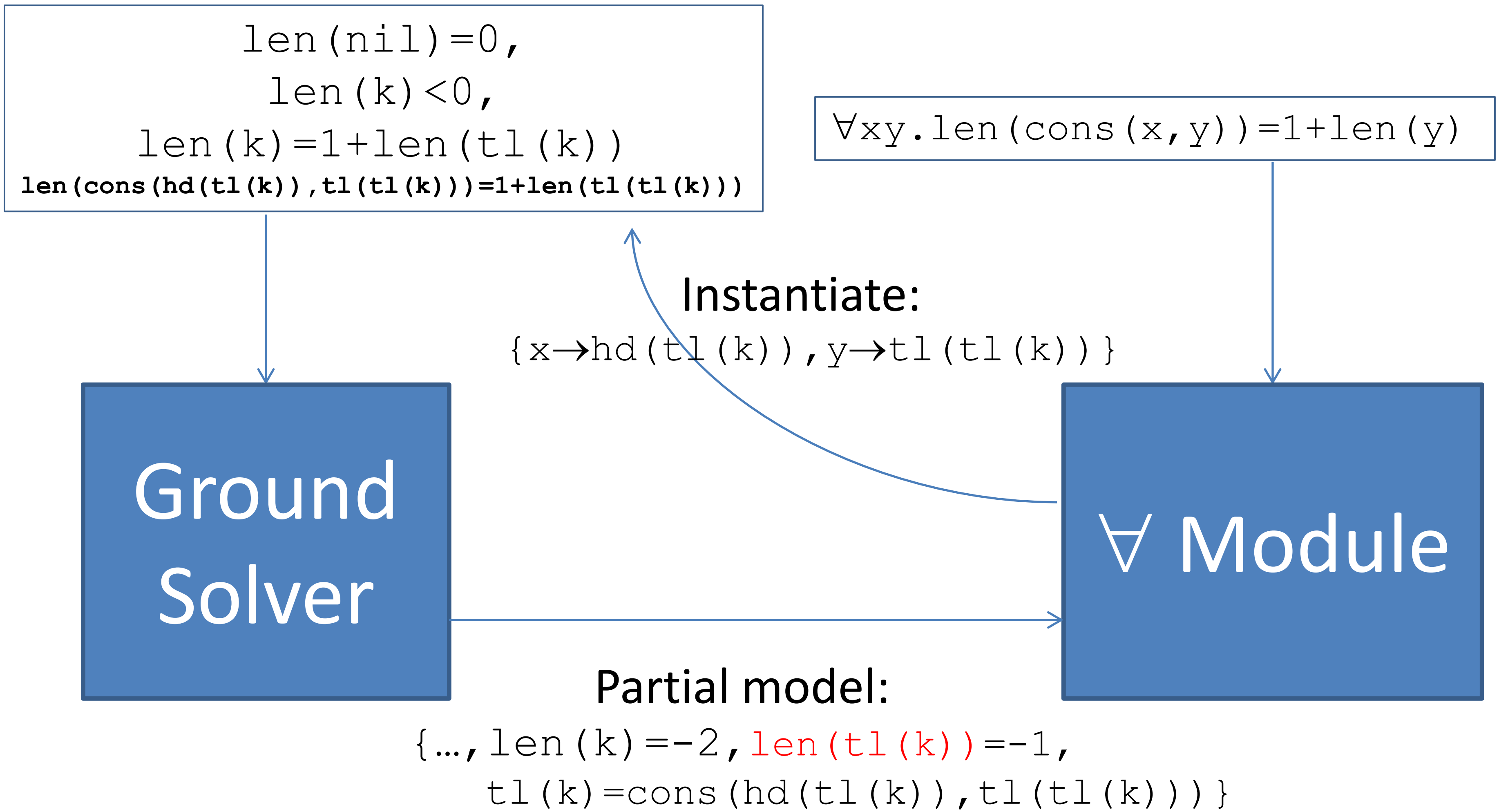
len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module

Example #2



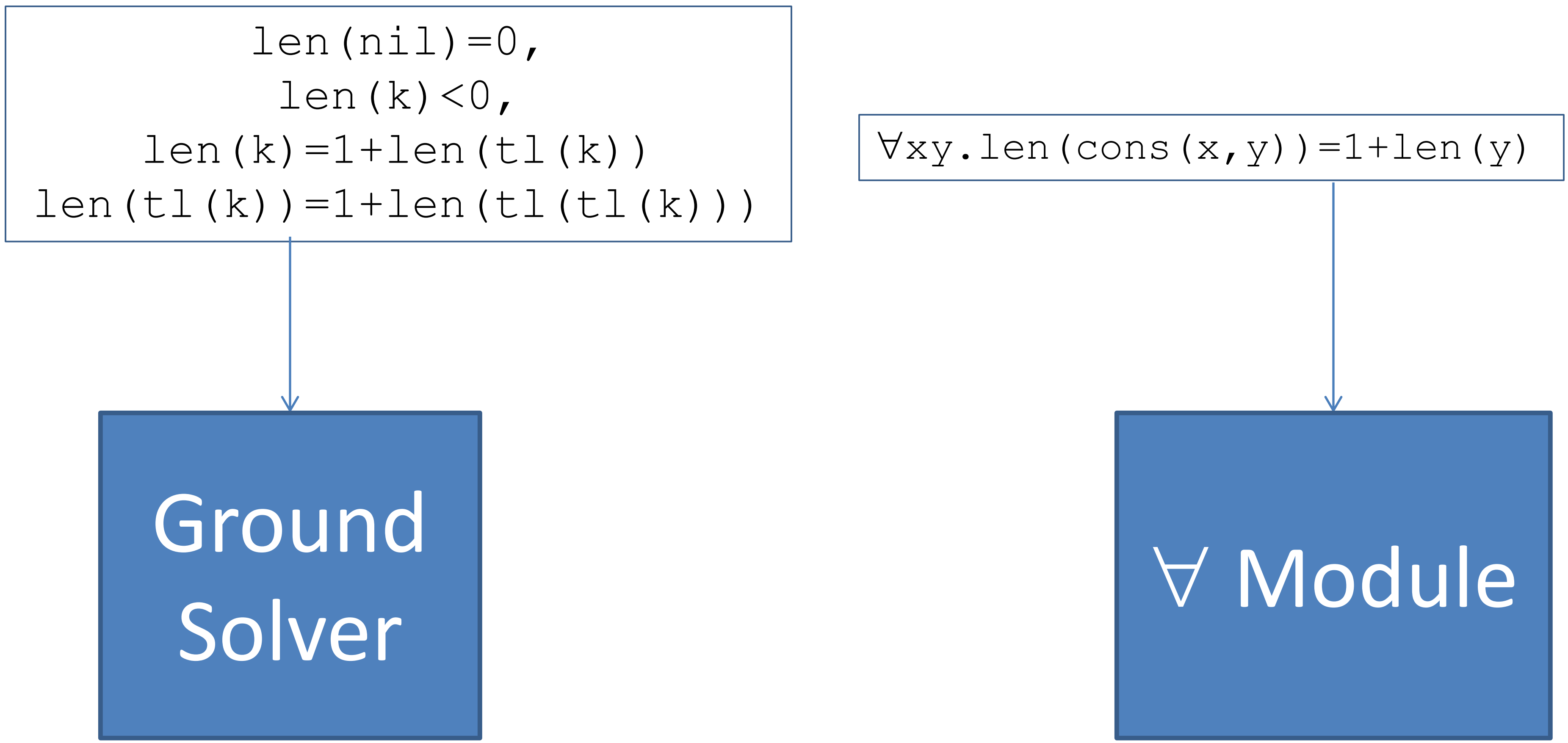
len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module
Partial model:
{…,len(k)=-2,len(tl(k))=-1,
tl(k)=cons(hd(tl(k)),tl(tl(k)))}

Example #2

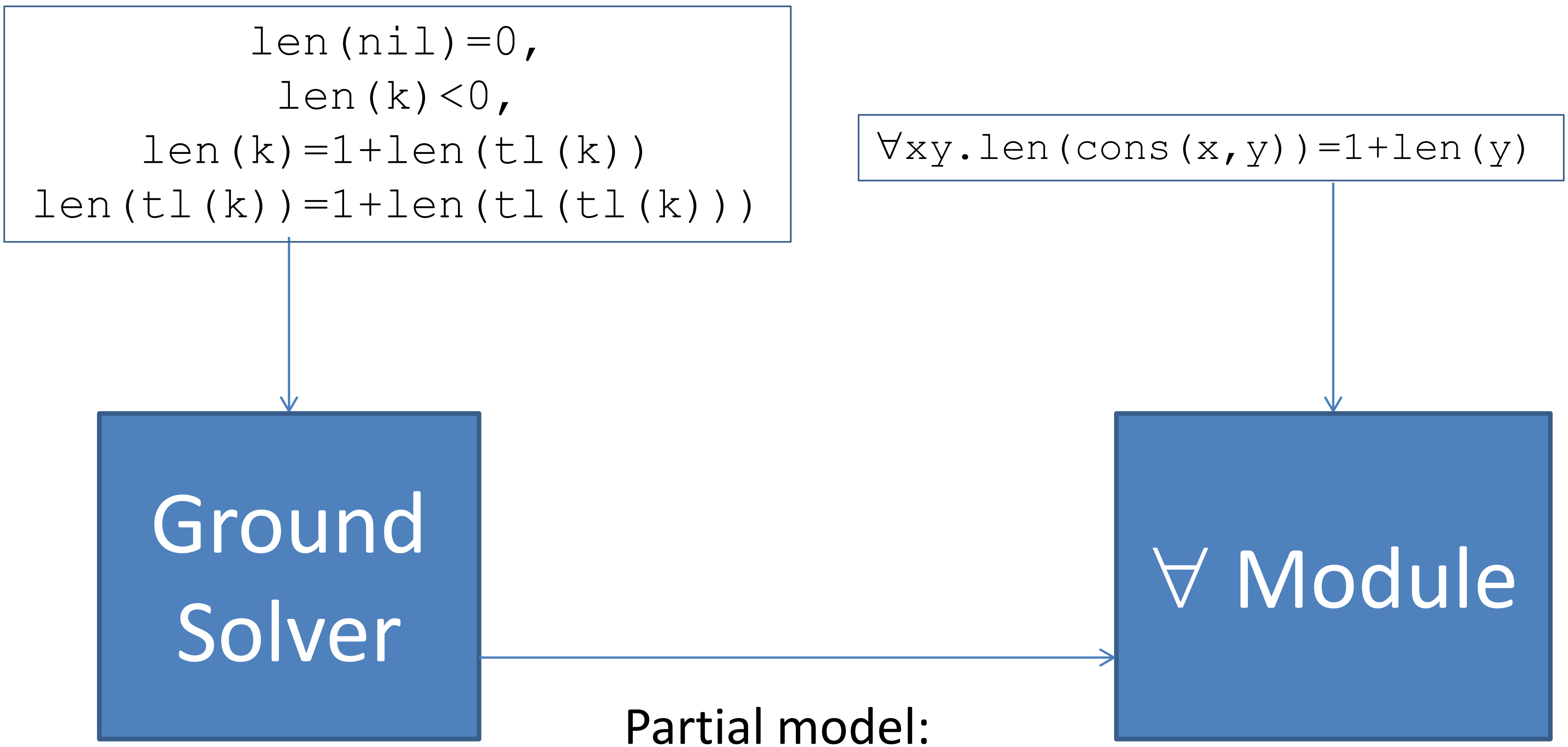


len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
len(cons(hd(tl(k)),tl(tl(k)))=1+len(tl(tl(k)))
∀xy.len(cons(x,y))=1+len(y)
Instantiate:
{x→hd(tl(k)),y→tl(tl(k))}
Ground Solver
∀ Module
Partial model:
{…,len(k)=-2,len(tl(k))=-1,
tl(k)=cons(hd(tl(k)),tl(tl(k)))}

Example #2

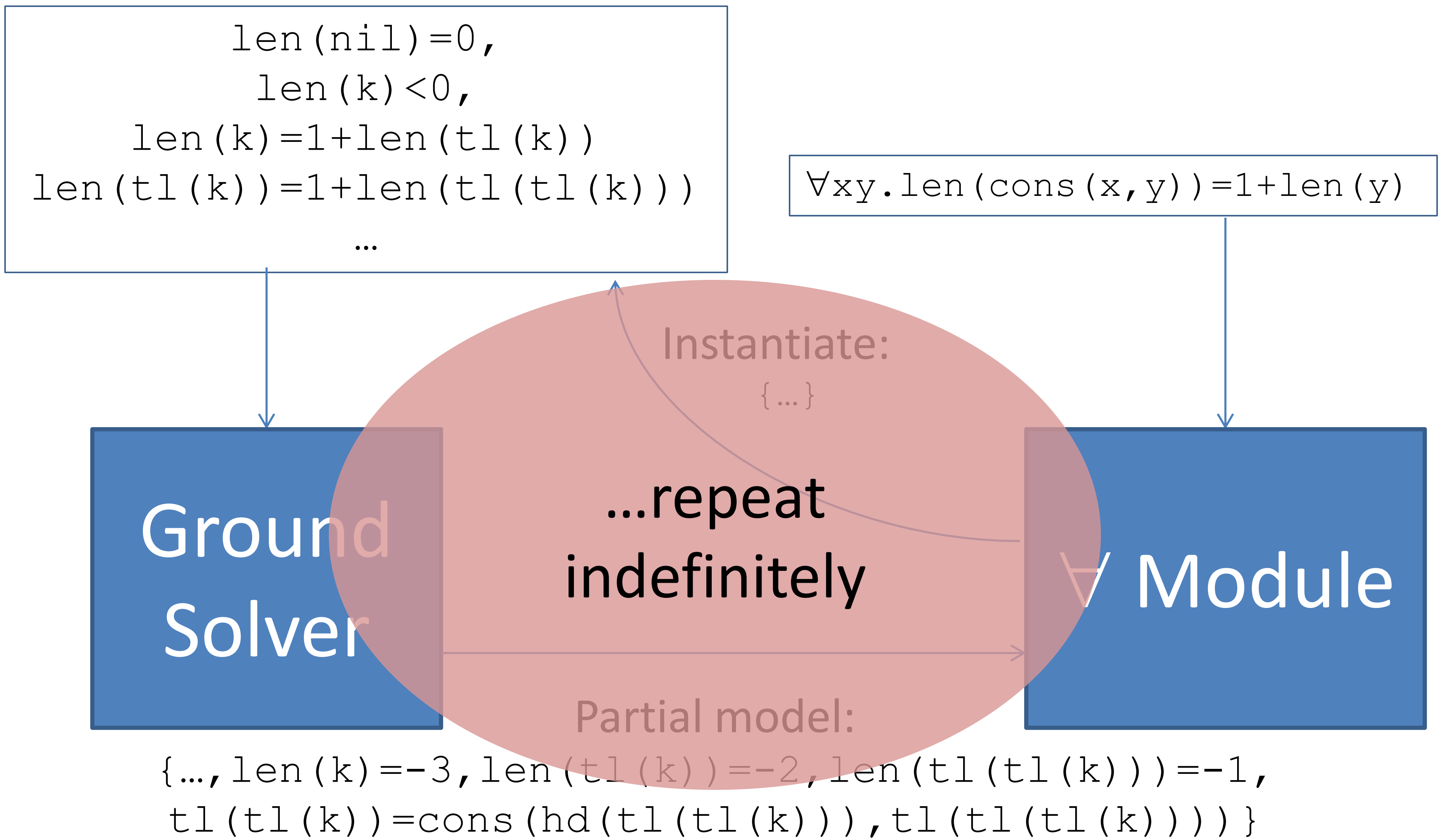

len(nil)=0,
len(k)<0,
len(k)=1+len(tl(k))
len(tl(k))=1+len(tl(tl(k)))
∀xy.len(cons(x,y))=1+len(y)
Ground Solver
∀ Module

Example #2



{…,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:

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

- Assume property does not for fresh k:

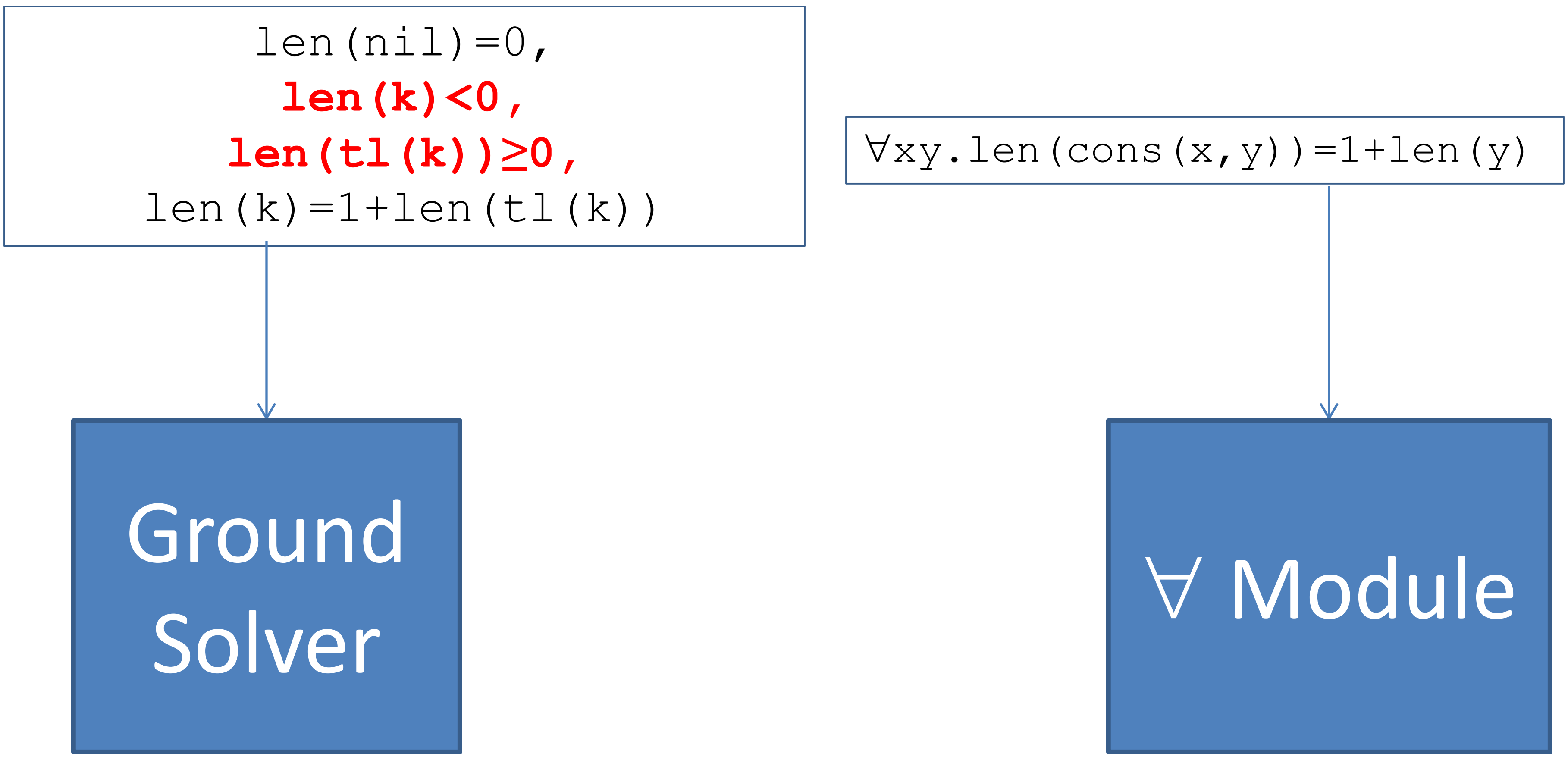
$$\neg\ \texttt{len(k)} \geq 0$$

AND

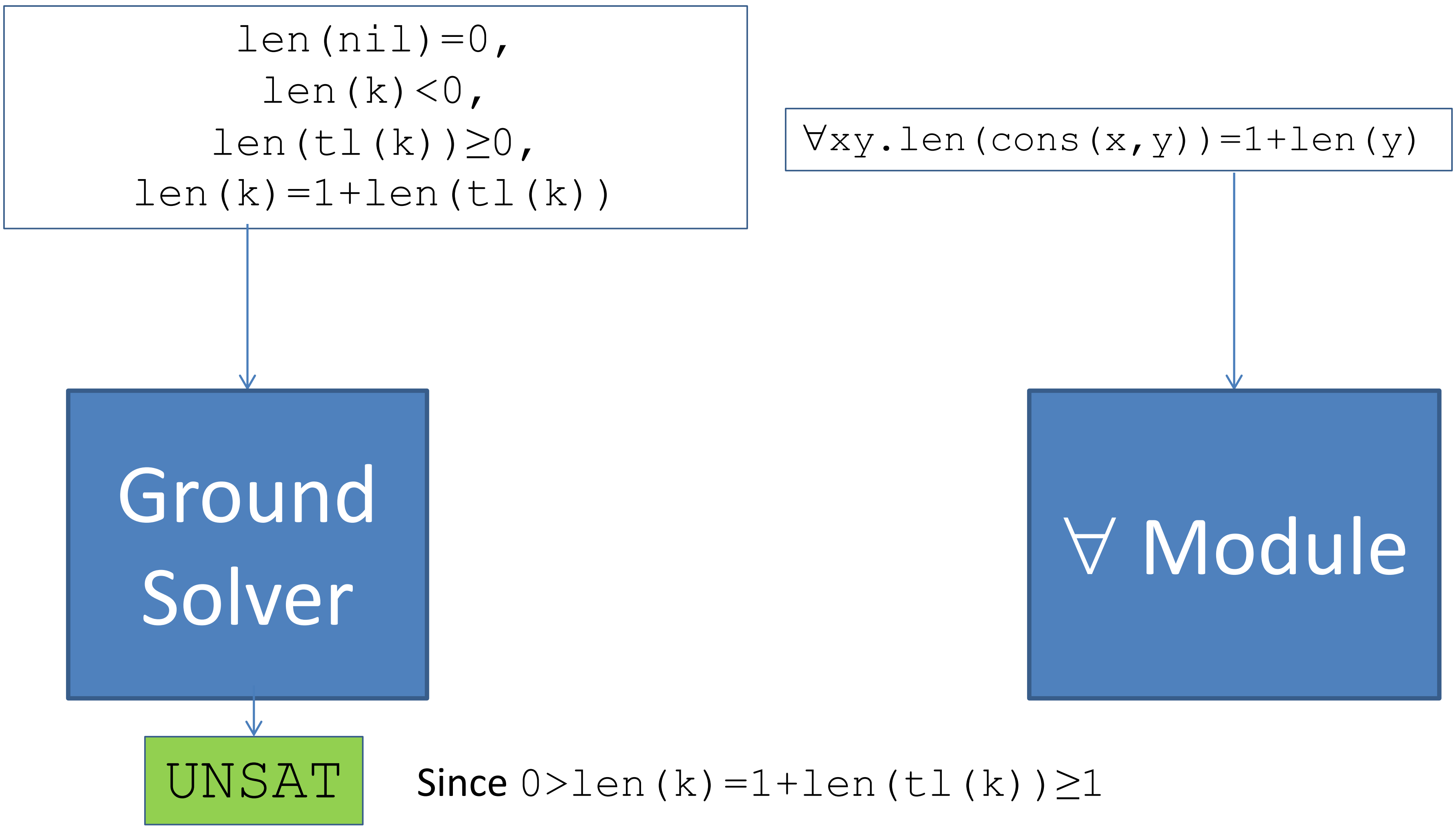
- Assume k is the *smallest* CE to property:

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

Example #2: revised



Example #2: revised



Skolemization with Inductive Strengthening

- General form:

$$\boxed{\forall x. P(x) \lor (\neg P(k) \land \forall y. R(y,k) \Rightarrow P(y))}$$

  - For well-founded relation R
- Extends for multiple variables
- Common examples 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))
```

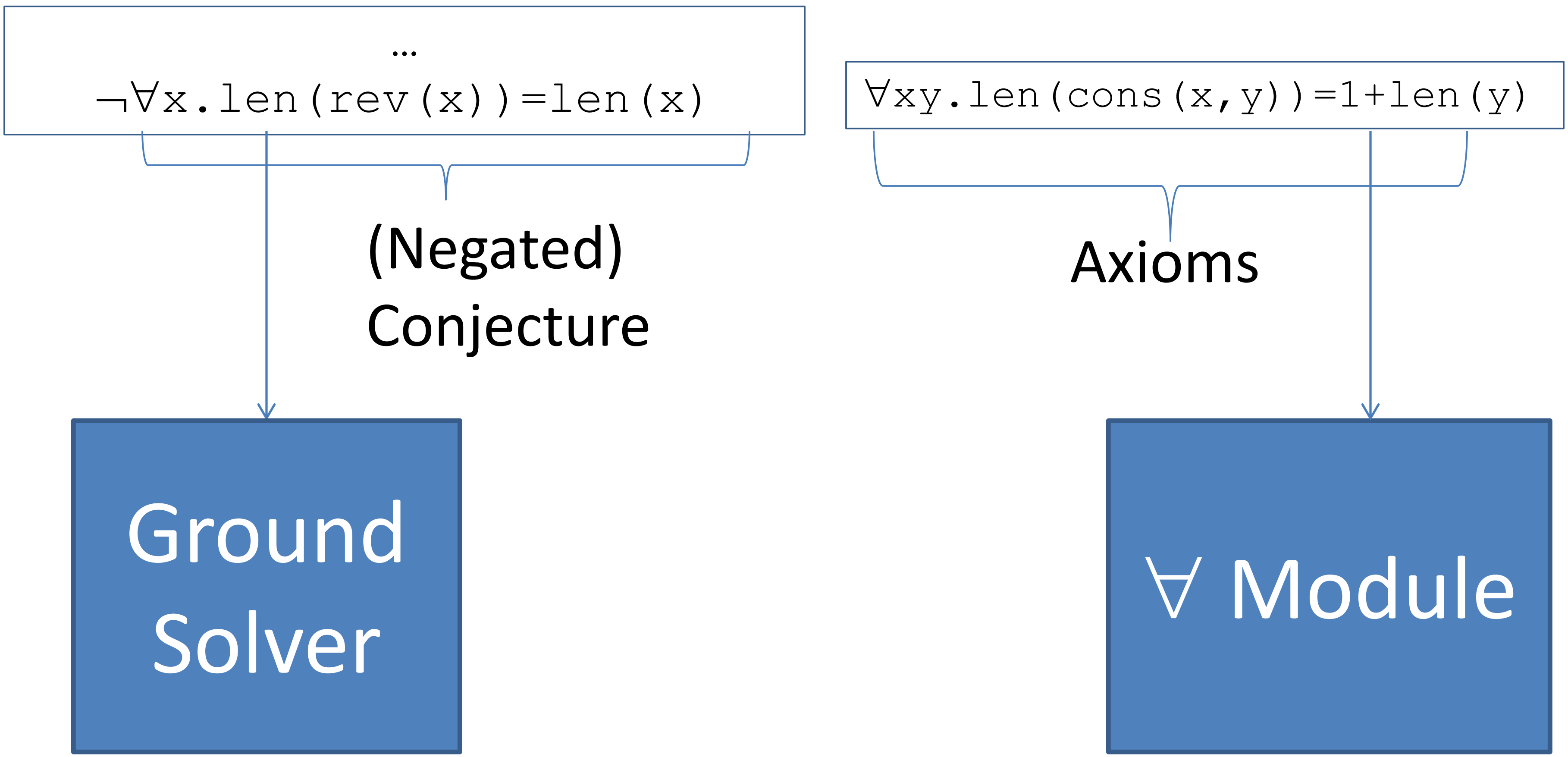
```
∀xy.plus(x,y)=plus(y,x)
```

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

…

$\neg\forall x.$`len(rev(x))=len(x)`

$\forall xy.$`len(cons(x,y))=1+len(y)`

$\neg\forall x.$`len(x)=Z`	$\vee$	$\forall x.$`len(x)=Z`
$\neg\forall x.$`app(x,nil)=x`	$\vee$	$\forall x.$`app(x,nil)=x`
…	$\vee$	…
$\neg\forall xy.$`len(app(x,y))=plus(len(x),len(y))`	$\vee$	$\forall xy.$`len(app(x,y))=plus(len(x),len(y))`

Ground Solver

- Enumerate subgoals

  For each, either:

  (a) it has a c.e.,

  (b) holds universally

$\forall$ Module

$\Rightarrow$ Use "splitting-on-demand"

Subgoal Generation in SMT

...

¬∀x.len(rev(x))=len(x)

∀xy.len(cons(x,y))=1+len(y)

¬∀x.len(x)=Z ∨ ∀x.len(x)=Z

¬∀x.app(x,nil)=x ∨ ∀x.app(x,nil)=x

... ∨ ...

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

Ground Solver

∀ Module

Subgoal Generation in SMT

…

$\mathtt{len(rev(k_0))} \neq \mathtt{len(k_0)},$

$\mathtt{len(rev(tl(k_0)))=len(tl(k_0))}$

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

$\mathtt{len(k_1)} \neq \mathtt{Z}, \mathtt{len(tl(k_1))=Z} \quad \vee \quad \forall x.\mathtt{len(x)=Z}$

$\mathtt{app(k_2,nil)} \neq \mathtt{k_2}, \mathtt{app(tl(k_2),nil)=tl(k_2)} \quad \vee \quad \forall x.\mathtt{app(x,nil)=x}$

$\ldots \quad \vee \quad \ldots$

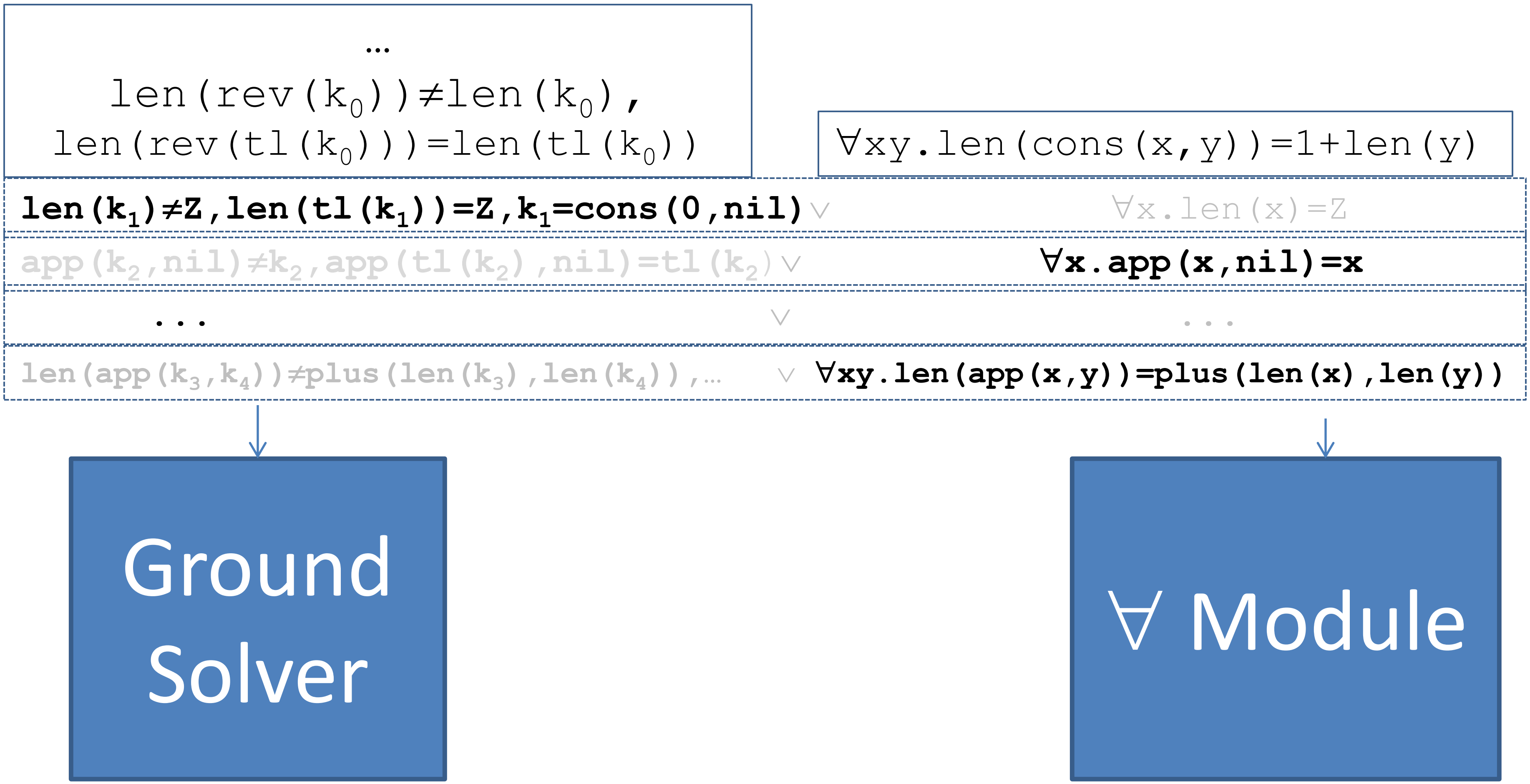
$\mathtt{len(app(k_3,k_4))} \neq \mathtt{plus(len(k_3),len(k_4))}, \ldots \quad \vee \quad \forall xy.\mathtt{len(app(x,y))=plus(len(x),len(y))}$

Ground Solver

∀ Module

- Each subgoal can be inductively strengthened
  - I.e. holds for a smallest counterexample

Subgoal Generation in SMT



- When negated s.g. is SAT, deduce ground information
- When negated s.g. is UNSAT, it can be used as axiom

Subgoal Generation : Challenges

- Generate subgoals by enumeration
- Main challenge: scalability
- Keys to success:
  - Generate s.g. in a fair manner (smaller s.g. first)
  - Filter out s.g. that are not useful

Subgoal Filtering

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

  🚫 $\forall$`xy.count(x,y)=count(rev(x),y)`
  - Irrelevant, since conjecture is not related to "`count`"
- Filtering based on canonicity:

  🚫 $\forall$`x.len(x)=len(app(x,nil))`
  - Redundant, since we know $\forall$`x.x=app(x,nil)`
- Filtering based on counterexamples:

  🚫 $\forall$`x.len(x)=len(app(x,x))`
  - False, since we know `len(x)`$\neq$`len(app(x,x))` for `x`$\neq$`nil`

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

Experiments : 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

Experiments : Encodings

1. Base encoding (**dt**), e.g. defined plus as:

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

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

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

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

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

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

Results : SMT solvers

dt	**Isaplan**	**Clam**	**HSpec**	**Leon**
z3	16	0	0	6
cvc4	15	0	0	7
cvc4+i	68	16	12	29
cvc4+ig	75	38	17	34
dtt	**Isaplan**	**Clam**	**HSpec**	**Leon**
z3	35	9	2	9
cvc4	34	7	2	8
cvc4+i	64	16	11	37
cvc4+ig	67	25	12	39
dti	**Isaplan**	**Clam**	**HSpec**	**Leon**
z3	35	8	2	9
cvc4	34	8	2	9
cvc4+i	76	25	15	41
cvc4+ig	80	39	18	42
Total	*85*	*50*	*26*	*45*

- 300 second timeout

cvc4+i:
with induction

cvc4+ig:
with induction
+subgoal gen.

Results: Subgoal Generation

- With subgoals, solved +37 for **dti** encoding
  - Only solved +1 when filtering turned off
- Most subgoals were small: term size $\leq$ 3
- Can find simpler goals than by manual inspection:
  - For conjecture:

$$\forall xy.\mathtt{count(x,y)=count(insort(x),y)}$$

  - We thought it would require:

$$\forall xy.\mathtt{count(ins(y,x),y)=S(count(x,y))},$$
$$\forall xyz.\mathtt{y}\neq\mathtt{z}\Rightarrow\mathtt{count(ins(y,x),z)=count(x,z)}$$

  - CVC4 found:

$$\forall xyz.\mathtt{count(ins(y,x),z)=count(cons(y,x),z)}$$

$\Rightarrow$ *Suffices to prove conjecture, which CVC4 did fully automatically*

Comparison with Other Provers

	Isaplan	**Clam**	**HSpec**	**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 CVC4
- Best performance by making use of:
  - Theory reasoning (dti encoding)
  - Subgoal generation
- Competitive with inductive theorem provers

Future Work

- Improvements to subgoal generation
  - Filtering heuristics
  - User-guided/interactive approaches
- Incorporate more induction schemes
- Completeness criteria
  - Identify cases approach is guaranteed to succeed
- Standard format for inductive theorem provers
- Applications:
  - Use within Leon verification tool (EPFL)
  - Synthesis of recursive functions

Thanks!

- CVC4 publicly available:
  - http://cvc4.cs.nyu.edu/downloads/
  - Induction techniques:
    - Enabled by "`--quant-ind`"
    - More details, see VMCAI 2015 paper

- Questions?