An Introduction to SMT Solvers and Their Applications (Part 1)

Andrew Reynolds

University of Iowa

October 13, 2017

THE UNIVERSITY OF IOWA

Satisfiability Modulo Theories (SMT) solvers

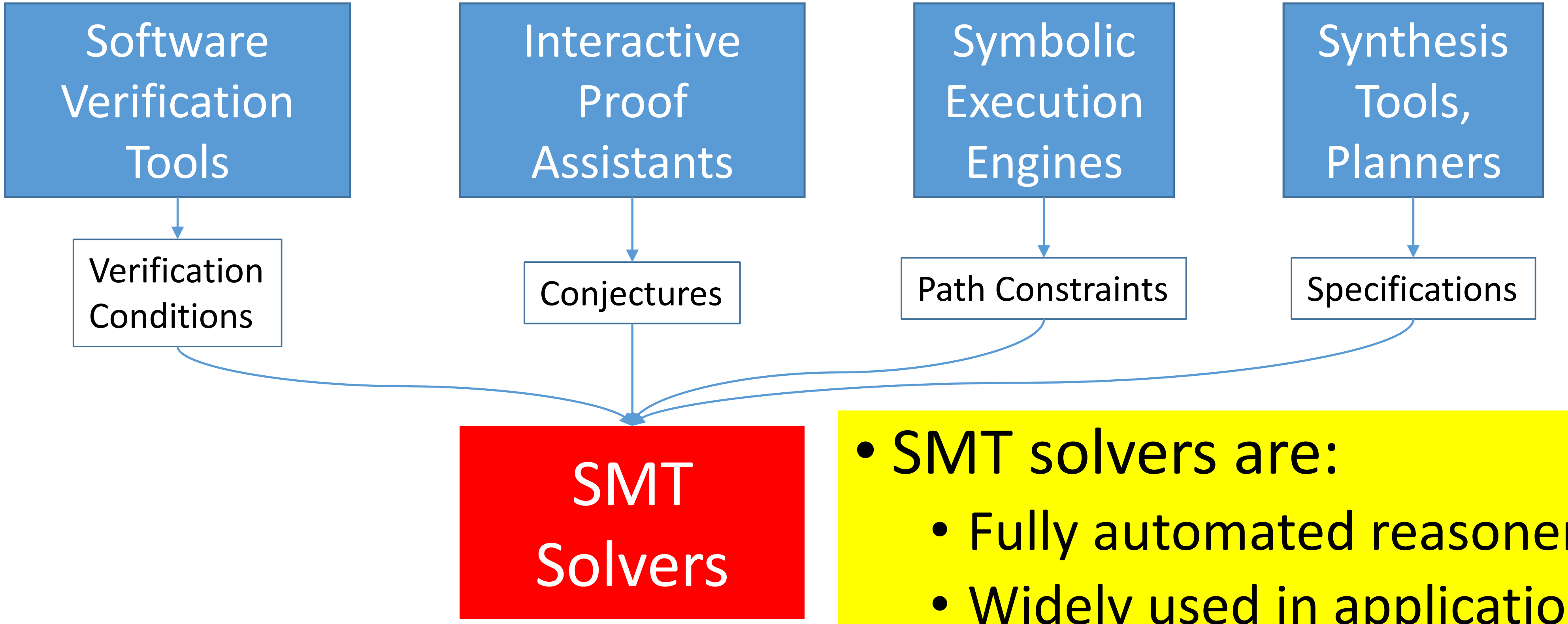
- *Useful tools* for
  - Verification
  - Interactive theorem proving
  - Symbolic execution
  - Synthesis
  - …(your application here)
- Examples of current SMT solvers:
  - Z3, CVC4, Yices2, Boolector, MathSAT, VeriT

    $\Rightarrow$ I develop the CVC4 SMT solver

Acknowledgements

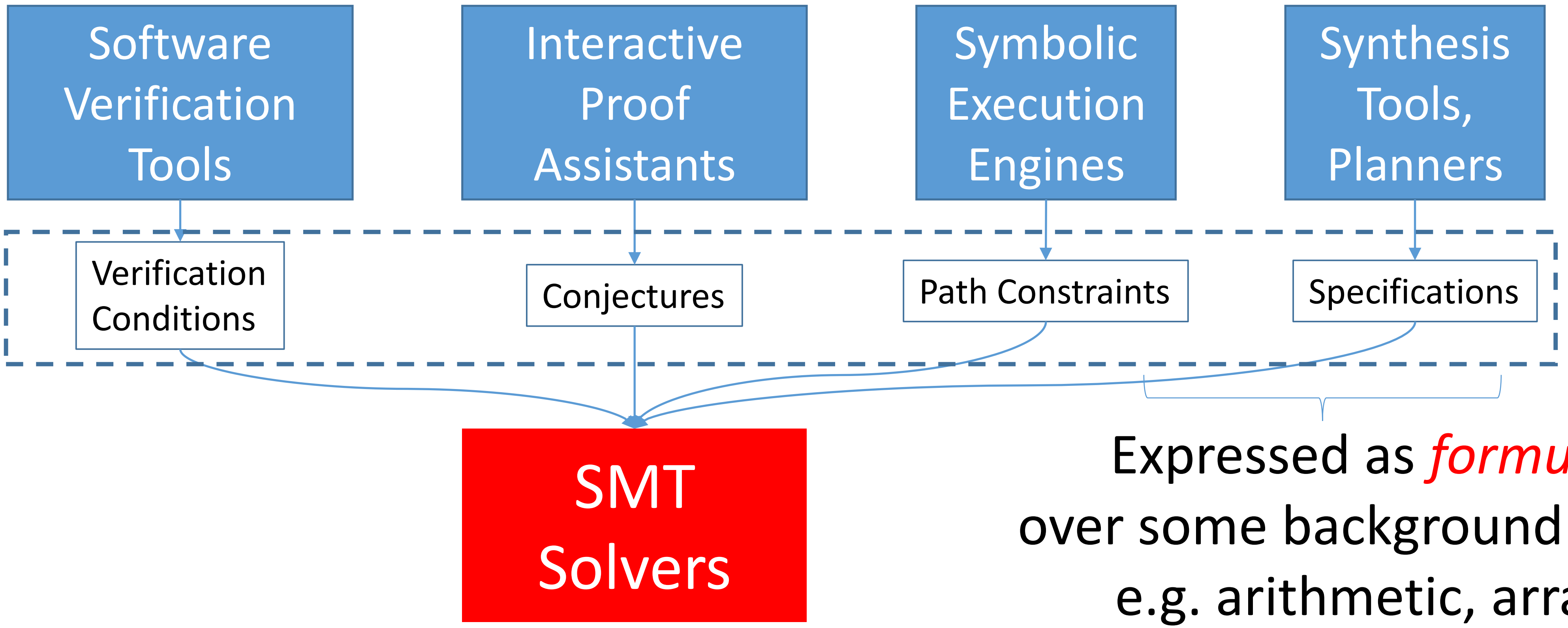
- Thanks to development team of CVC4:
  - Clark Barrett, Cesare Tinelli, Tim King, Andres Noetzli, Paul Meng, Aina Niemetz, Mathias Preiner, Arjun Viswanathan
  - Past members: Morgan Deters, Dejan Jovanovic, Liana Hadarean, Kshitij Bansal, Tianyi Liang, Nestan Tsiskardidze, Christopher Conway, Francois Bobot, Guy Katz, Alain Mebsout, Burak Ekici

Satisfiability Modulo Theories (SMT) Solvers



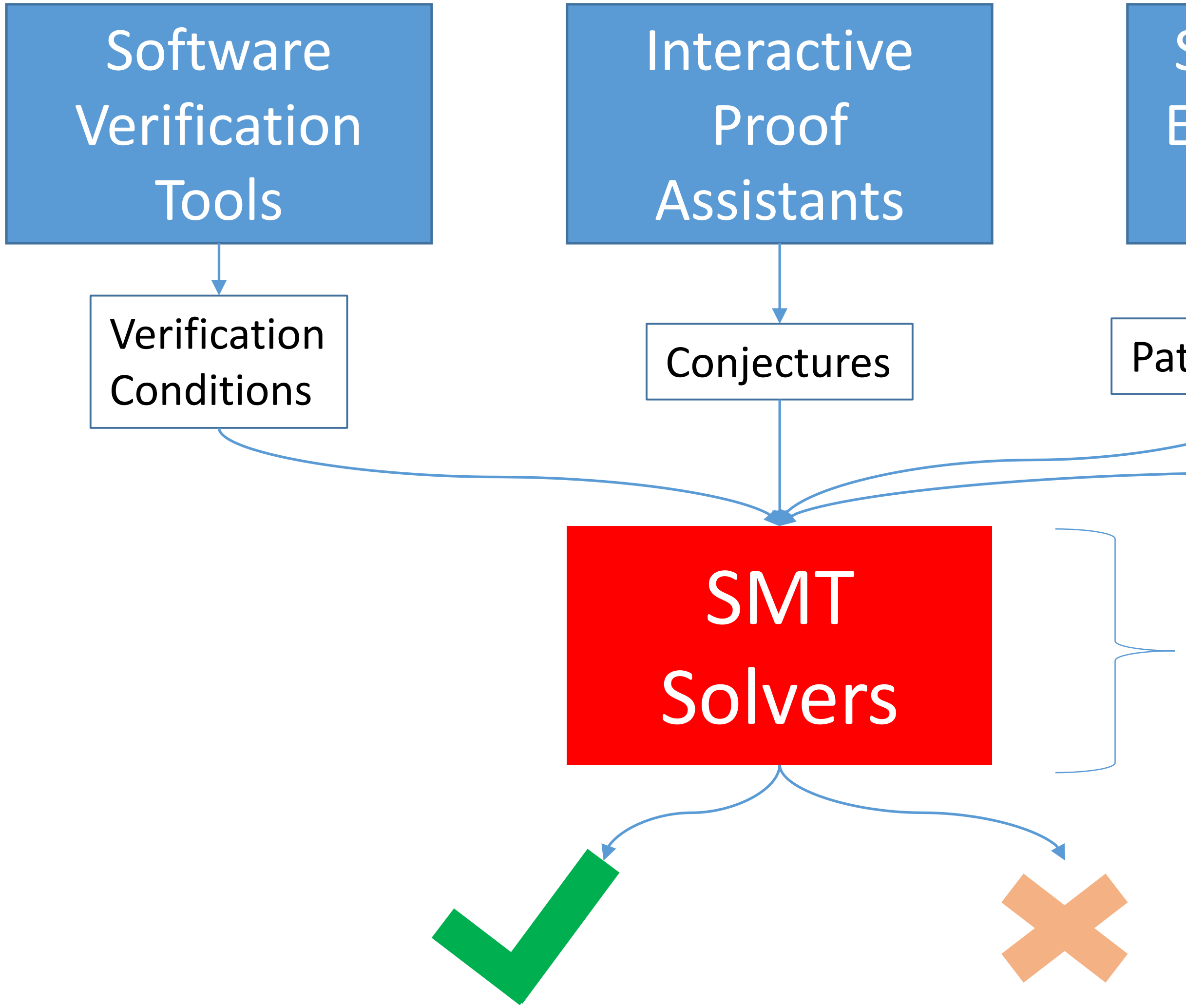
- SMT solvers are:
  - Fully automated reasoners
  - Widely used in applications

Satisfiability Modulo Theories (SMT) Solvers



Expressed as *formulas*
over some background *theory*
e.g. arithmetic, arrays

Satisfiability Modulo Theories (SMT) Solvers



SMT solvers are *general-purpose tools* for handling each case

Contract-Based Software Verification

@precondition: $x_{in} > y_{in}$

```
void swap(int x, int y)
{
      x := x + y;
      y := x - y;
      x := x - y;
}
```

…does this function ensure that $x_{out} = y_{in} \land y_{out} = x_{in}$?

Software Verification Tools

Contract-Based Software Verification

```
@precondition: x_in>y_in
void swap(int x, int y)
{
        x := x + y;
        y := x - y;
        x := x - y;
}
```

…does this function ensure that $x_{out}=y_{in} \wedge y_{out}=x_{in}$?

Software Verification Tools

$x_{in}>y_{in}$ — **Pre-condition**

$x_2=x_{in}+y_{in} \wedge y_2=y_{in}$

$x_3=x_2 \wedge y_3=x_2-y_2$

$x_{out}=x_3-y_3 \wedge y_{out}=y_3$ — **Function Body**

$(x_{out}\neq y_{in} \vee y_{out}\neq x_{in})$ — **(Negated) Post-condition**

Contract-Based Software Verification

@precondition: $x_{in} > y_{in}$

```
void swap(int x, int y)
{
      x := x + y;
      y := x - y;
      x := x - y;
}
```

@ensures

$x_{out}=y_{in} \wedge y_{out}=x_{in}$

Software Verification Tools

$x_{in} > y_{in}$ — **Pre-condition**

$x_2 = x_{in} + y_{in} \wedge y_2 = y_{in}$

$x_3 = x_2 \wedge y_3 = x_2 - y_2$

$x_{out} = x_3 - y_3 \wedge y_{out} = y_3$ — **Function Body**

$(x_{out} \neq y_{in} \vee y_{out} \neq x_{in})$ — **(Negated) Post-condition**

SMT Solver

Interactive Proof Assistants

```
Theorem app_rev:
  forall (x : list) (y : list), rev append x y = append (rev y) (rev x).
 Proof.
    ….does this theorem hold? What is the proof?
```

→ Interactive Proof Assistant

Interactive Proof Assistants

```
Theorem app_rev:
  forall (x : list) (y : list), rev append x y = append (rev y) (rev x).
Proof.
```

....does this theorem hold? What is the proof?

Interactive Proof Assistant

```
List := cons( head : Int, tail : List ) | nil
```

Signature

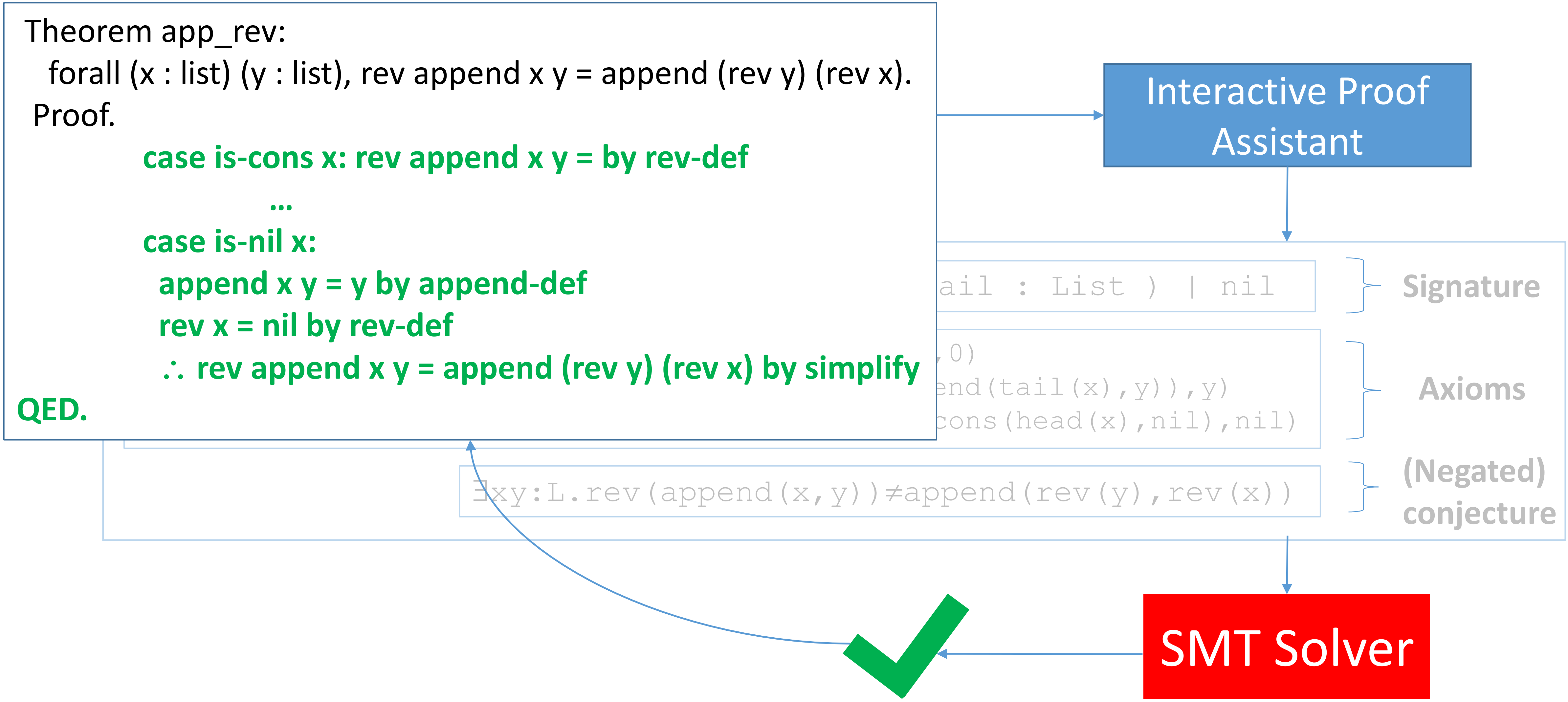
```
∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)
```

Axioms

```
∃xy:L.rev(append(x,y))≠append(rev(y),rev(x))
```

(Negated) conjecture

Interactive Proof Assistants



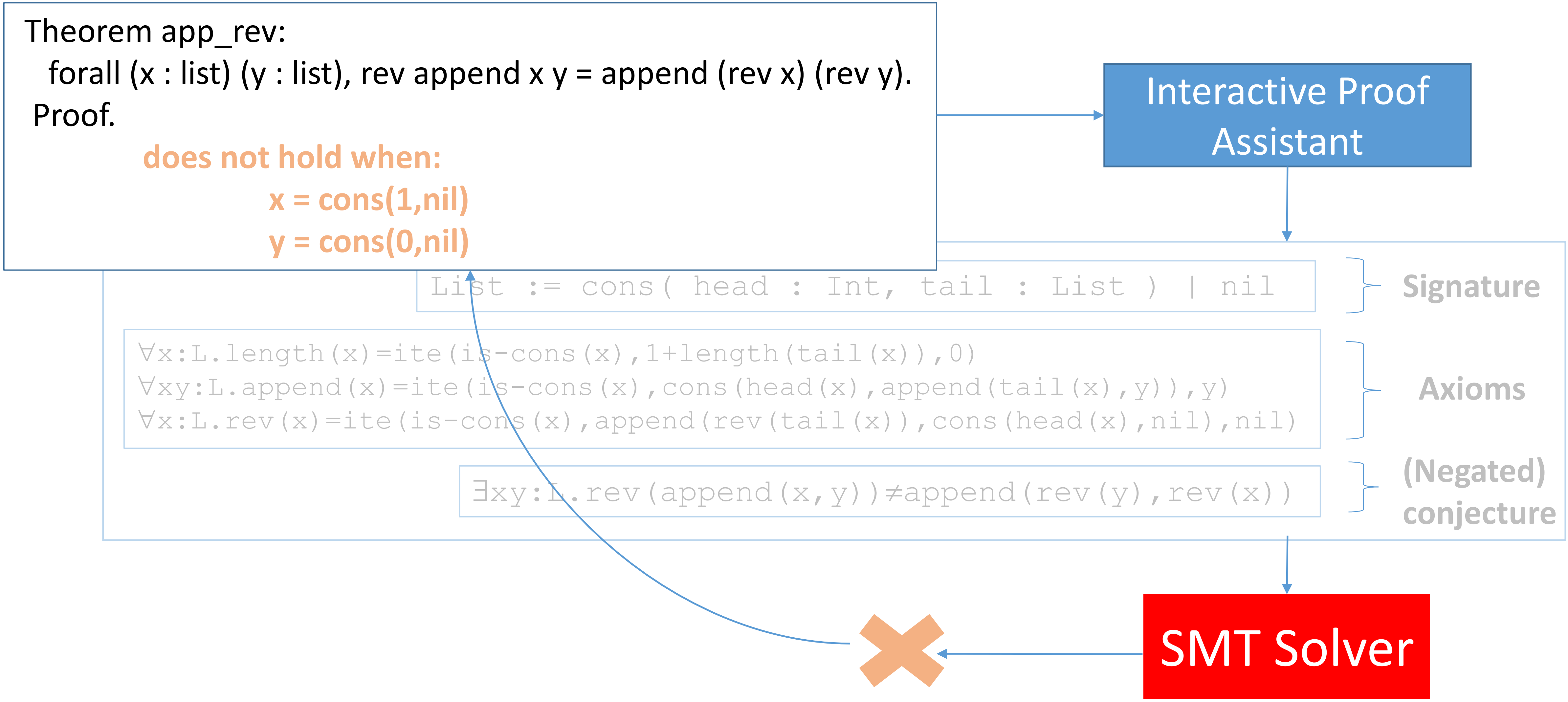
Interactive Proof Assistants

Theorem app_rev:
  forall (x : list) (y : list), rev append x y = append (rev x) (rev y).
 Proof.

  ….does this theorem hold? What is the proof?

→ Interactive Proof Assistant

Interactive Proof Assistants



Symbolic execution

```
char buff[15];
char pass;
cout << "Enter the password :";
gets(buff);
if (regex_match(buff, std::regex("([A-Z]+)") )) {
    if(strcmp(buff, “PASSWORD")) {
        cout << "Wrong Password";
    } else {
        cout << "Correct Password";
        pass = 'Y';
    }
    if(pass == 'Y') {
         grant_root_permission();
        Assert(strcmp(buff,”PASSWORD”)==0);
    }
}
```

Does this assertion hold for all executions?

Symbolic Execution Engine

Symbolic execution

```
char buff[15];
char pass;
cout << "Enter the password :";
gets(buff);
if (regex_match(buff, std::regex("([A-Z]+)") )) {
    if(strcmp(buff, “PASSWORD")) {
        cout << "Wrong Password";
    } else {
        cout << "Correct Password";
        pass = 'Y';
    }
    if(pass == 'Y') {
        grant_root_permission();
        Assert(strcmp(buff,”PASSWORD”)==0);
    }
}
```

Does this assertion hold for all executions?

Symbolic Execution Engine

```
…
(assert (and (= (str.len buff) 15)) (= (str.len pass1) 1)))
(assert (or (< (str.len input) 15)  (= input (str.++ buff pass0 rest)))
(assert (str.in.re buff (re.+ (re.range "A" "Z"))))
(assert (and (not (= buff "PASSWORD")) (= pass1 pass0)))
(assert (= pass1 "Y"))
(assert (not (= buff "PASSWORD")))
```

Symbolic execution

```
char buff[15];
char pass;
cout << "Enter the password :";
gets(buff);
if (regex_match(buff, std::regex("([A-Z]+)") )) {
   if(strcmp(buff, “PASSWORD")) {
      cout << "Wrong Password";
   } else {
      cout << "Correct Password";
      pass = 'Y';
   }
   if(pass == 'Y') {
       grant_root_permission();
      Assert(strcmp(buff,”PASSWORD”)==0);
   }
}
```

Does this assertion hold for all executions?

Symbolic Execution Engine

```
…
(assert (and (= (str.len buff) 15)) (= (str.len pass1) 1)))
(assert (or (< (str.len input) 15)  (= input (str.++ buff pass0 rest)))
(assert (str.in.re buff (re.+ (re.range "A" "Z"))))
(assert (and (not (= buff "PASSWORD")) (= pass1 pass0)))
(assert (= pass1 "Y"))
(assert (not (= buff "PASSWORD")))
```

SMT Solver

```
(define-fun input () String “AAAAAAAAAAAAAAAY”)
(define-fun buff () String “AAAAAAAAAAAAAAA”)
(define-fun pass () String “Y”)
```

Symbolic execution

```
char buff[15];
char pass;
cout << "Enter the password :";
gets(buff);  ← "AAAAAAAAAAAAAAAY"
if (regex_match(buff, std::regex("([A-Z]+)") )) {
   if(strcmp(buff, "PASSWORD")) {
      cout << "Wrong Password";
   } else {
      cout << "Correct Password";
      pass = 'Y';
   }
   if(pass == 'Y') {
       grant_root_permission();
      Assert(strcmp(buff,"PASSWORD")==0);
   }
}
```

Symbolic Execution Engine

```
…
(assert (and (= (str.len buff) 15)) (= (str.len pass1) 1)))
(assert (or (< (str.len input) 15)  (= input (str.++ buff pass0 rest)))
(assert (str.in.re buff (re.+ (re.range "A" "Z"))))
(assert (and (not (= buff "PASSWORD")) (= pass1 pass0)))
(assert (= pass1 "Y"))
(assert (not (= buff "PASSWORD")))
```

SMT Solver

```
(define-fun input () String "AAAAAAAAAAAAAAAY")
(define-fun buff () String "AAAAAAAAAAAAAAA")
(define-fun pass () String "Y")
```

Synthesis Tools

```
void maxList(List a, List b, List& c)
{
  int max;
  for(i=0;i<a.size();i++){
    max = choose(x => x≥a[i]∧x≥b[i]);
    c := c.append(max);
  }
  return c;
}
@ensures: c_out≥a ∧ c_out≥b ?
```

Find an x that satisfies specification

x≥a[i]∧x≥b[i]

Synthesis Tools

Synthesis Tools

```
void maxList(List a, List b, List& c)
{
  int max;
  for(i=0;i<a.size();i++){
    max = choose(x => x≥a[i]∧x≥b[i]);
    c := c.append(max);
  }
  return c;
}
@ensures: c_out≥a ∧ c_out≥b ?
```

Find an x that satisfies specification $x \geq a[i] \wedge x \geq b[i]$

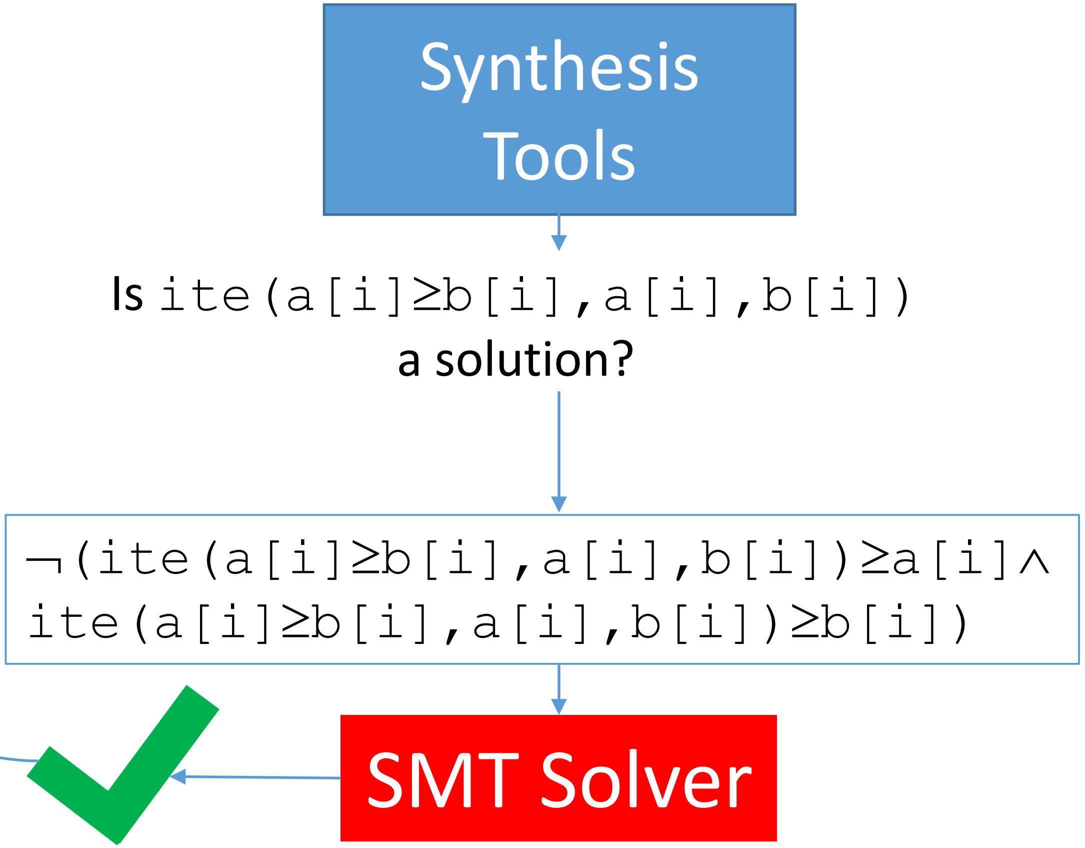
Synthesis Tools

Is $ite(a[i] \geq b[i], a[i], b[i])$ a solution?

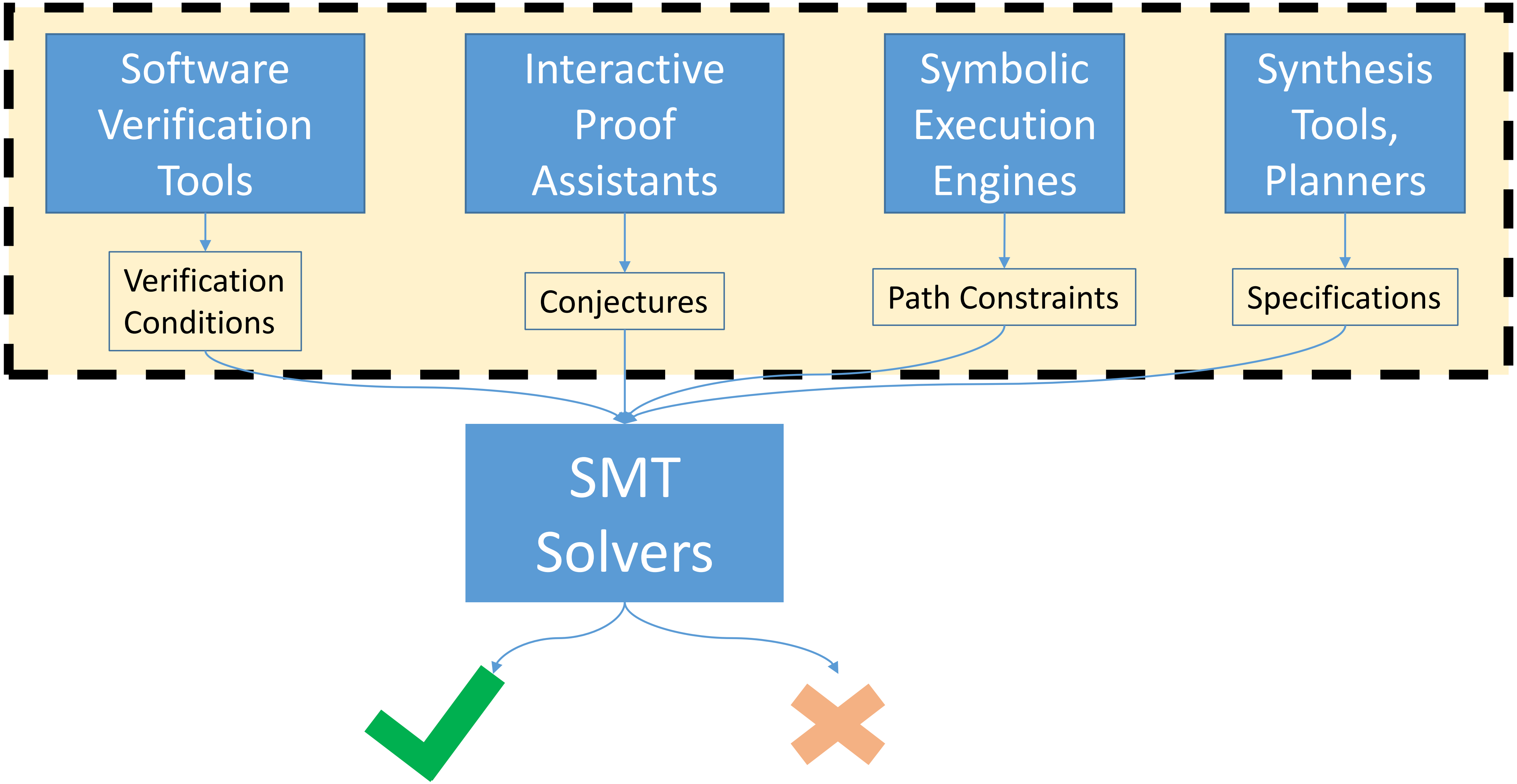
$\neg(ite(a[i] \geq b[i], a[i], b[i]) \geq a[i] \wedge ite(a[i] \geq b[i], a[i], b[i]) \geq b[i])$

Synthesis Tools

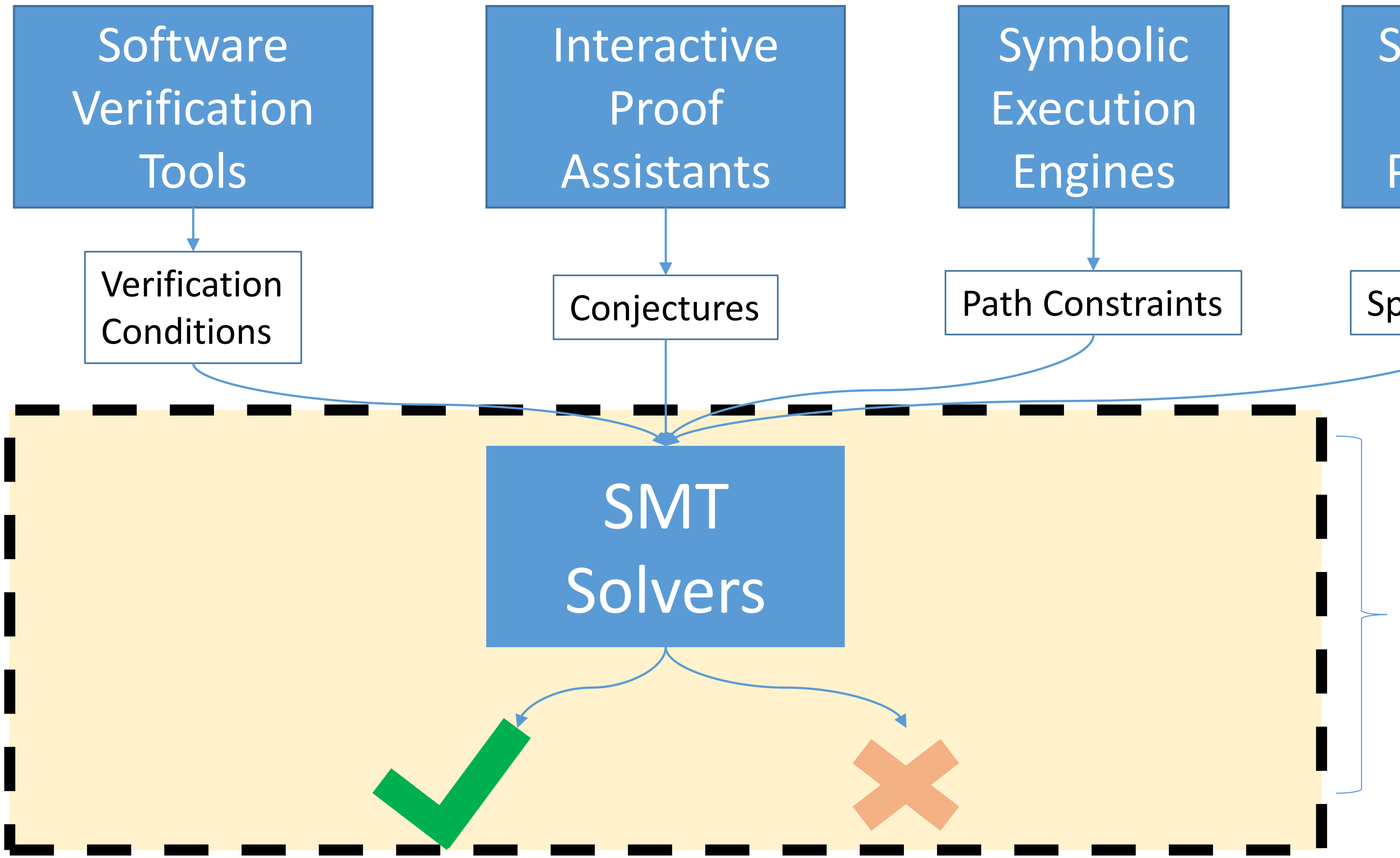
```
void maxList(List a, List b, List& c)
{
  int max;
  for(i=0;i<a.size();i++){
    max = if(a[i]≥b[i]{a[i]}else{b[i]};
    c := c.append(max);
  }
  return c;
}
@ensures: c_out≥a ∧ c_out≥b
```



Satisfiability Modulo Theories (SMT) Solvers



Satisfiability Modulo Theories (SMT) Solvers



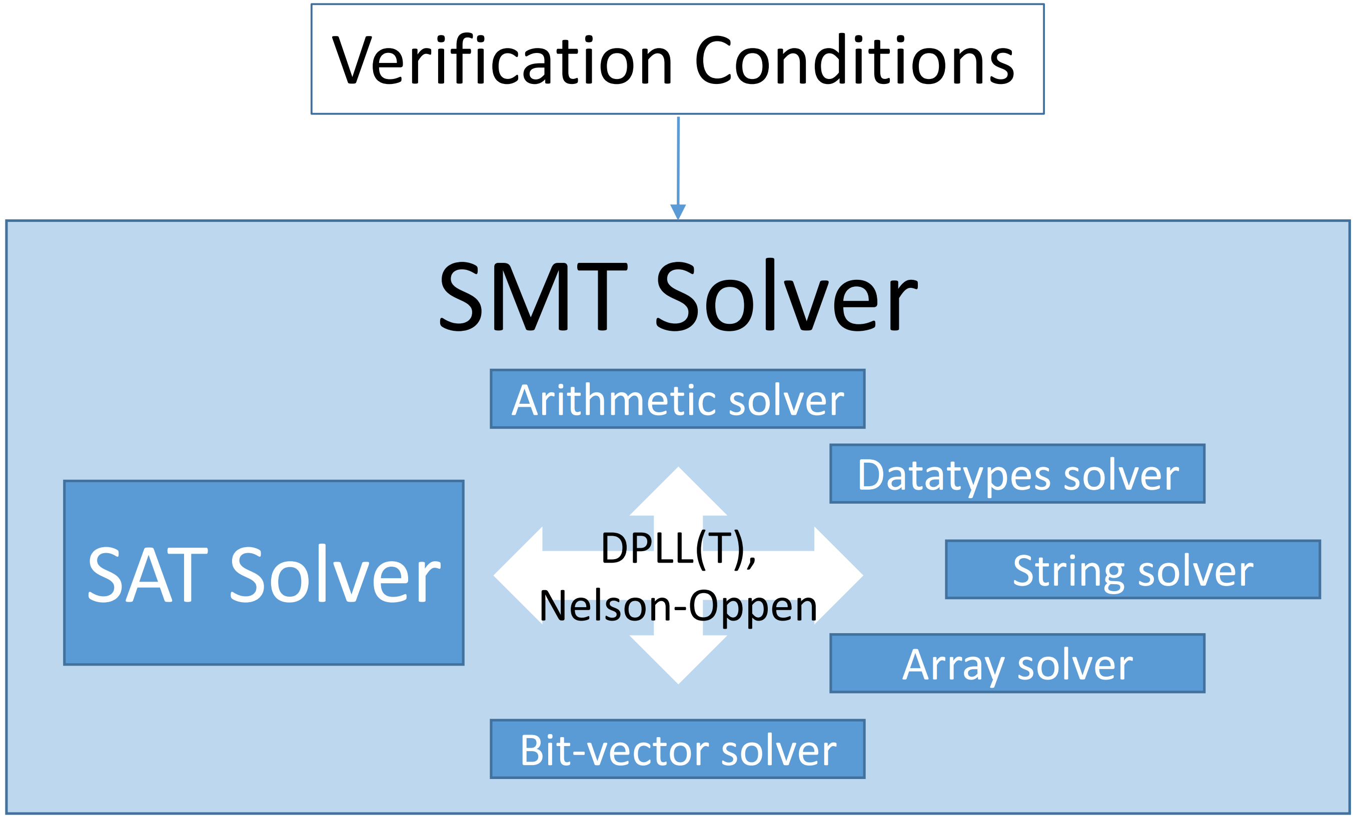
Will first focus on this portion

Overview

- Satisfiability Modulo Theories (SMT) solvers: how they work
  - DPLL, DPLL(T), decision procedures, Nelson-Oppen combination
- How to use SMT solvers
  - `smt2` language, models, proofs, unsat cores, incremental mode
- Things that SMT solvers can (and cannot) do well

SMT solvers

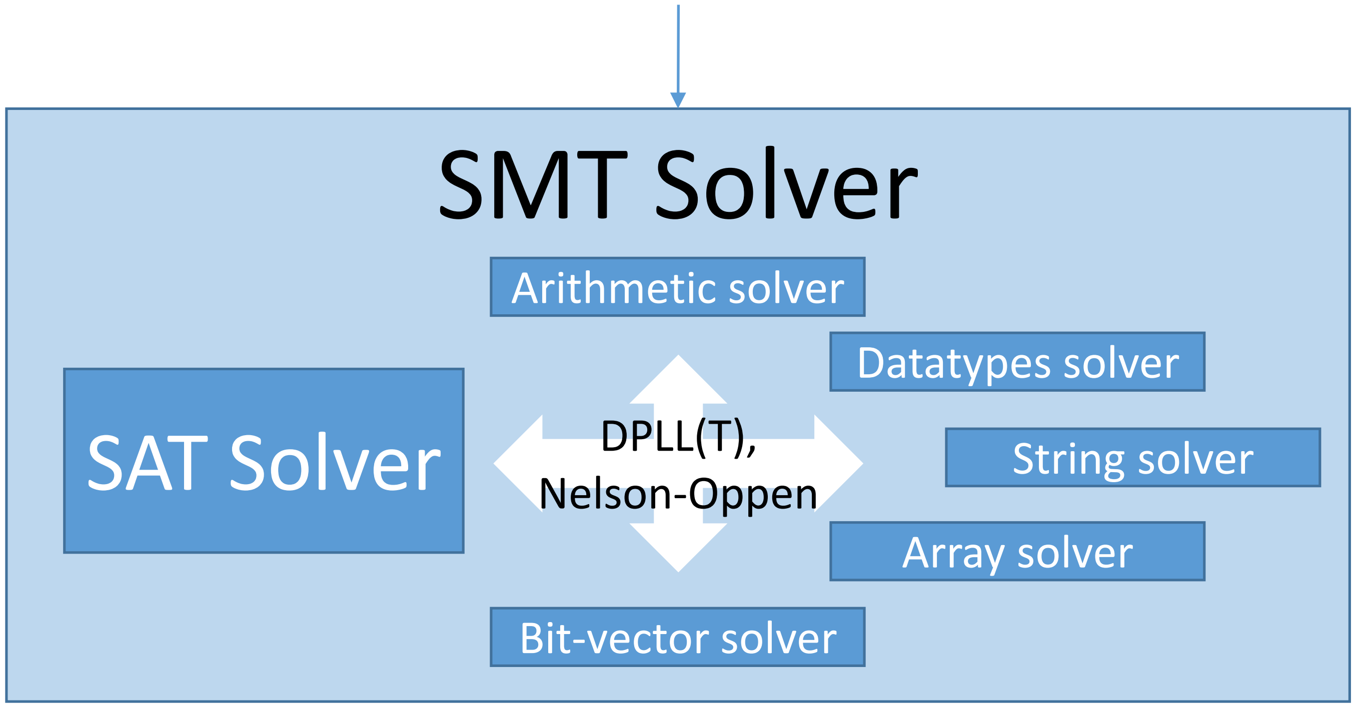
- Efficient tools for satisfiability and satisfiability *modulo theories*



SMT solvers

- Efficient tools for satisfiability and satisfiability *modulo theories*

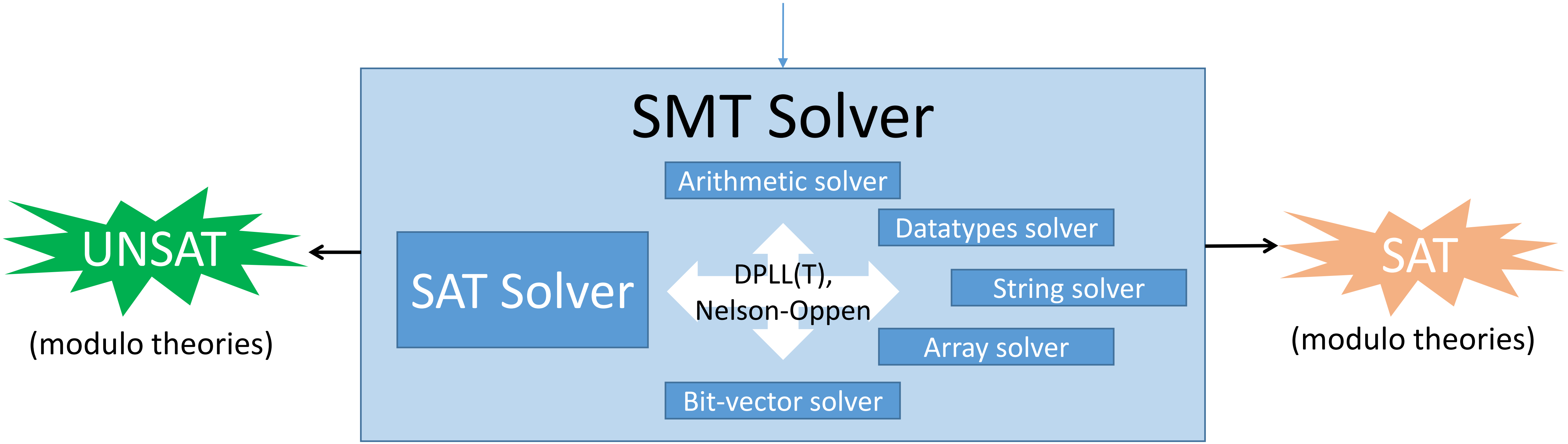
$$(A[x]+B[x]>0 \vee x+y>0) \wedge (cons(\text{"abc"},d_1) \neq d_2 \vee x<0)$$



SMT solvers

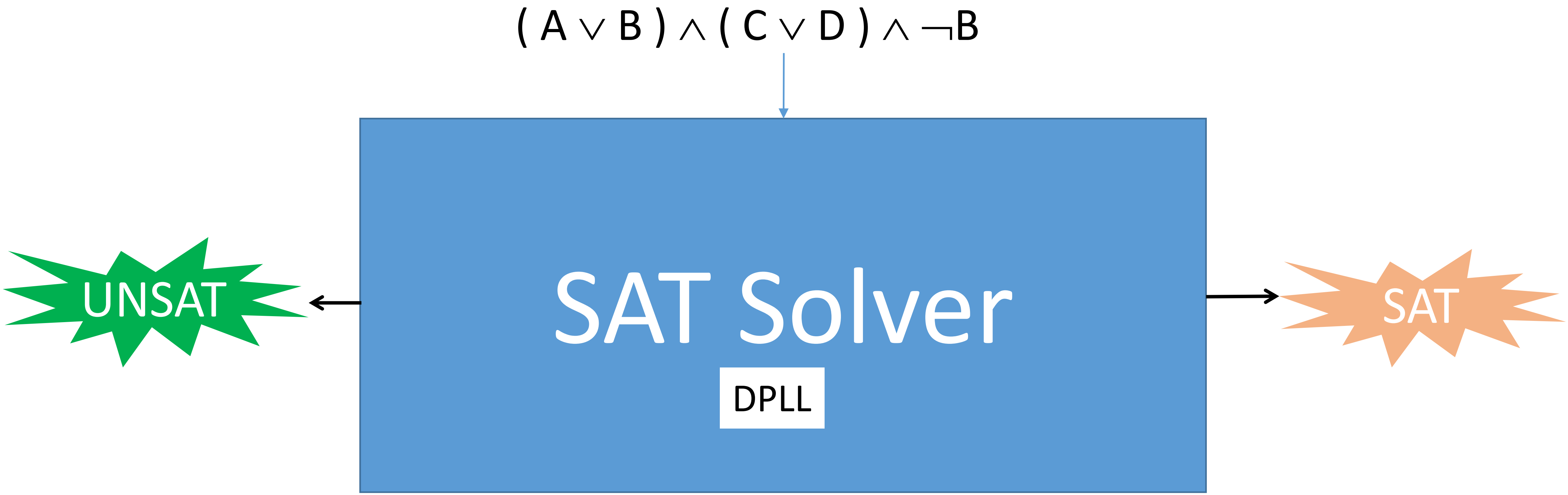
- Efficient tools for satisfiability and satisfiability *modulo theories*

$$(A[x]+B[x]>0 \vee x+y>0) \wedge (cons(\text{“abc”},d_1) \neq d_2 \vee x<0)$$



...but first : SAT solvers

- Efficient tools for *satisfiability*



DPLL

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

DPLL

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

Convert to clausal normal form (CNF)

- A formula is CNF if it is a conjunction of clauses
- A *clause* is a disjunction of literals e.g. $(A \vee B)$
- A *literal* is an atom or its negation e.g. A, $\neg$B,…

DPLL

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

- Alternate between:
  - Propagations : assign values to atoms whose value is forced
  - Decisions : choose an arbitrary value for an unassigned atom

DPLL

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

DPLL

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

Context

$B \rightarrow \bot$

- DPLL algorithm
  - Propagate : B $\rightarrow$ false

DPLL

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

- DPLL algorithm
  - Propagate : B $\rightarrow$ false
  - Propagate : A $\rightarrow$ true

Context

$B \rightarrow \bot$

$A \rightarrow T$

DPLL

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

- DPLL algorithm
  - Propagate : B $\rightarrow$ false
  - Propagate : A $\rightarrow$ true
  - Decide : C $\rightarrow$ true

Context

$B \rightarrow \bot$

$A \rightarrow T$

$C \rightarrow T^d$

DPLL

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

- DPLL algorithm
  - Propagate : B $\rightarrow$ false
  - Propagate : A $\rightarrow$ true
  - Decide : C $\rightarrow$ true

$\Rightarrow$ Input is SAT by interpretation where $\{A \rightarrow T, B \rightarrow \bot, C \rightarrow T\}$

Context

$B \rightarrow \bot$

$A \rightarrow T$

$C \rightarrow T^d$

DPLL

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

Context

DPLL

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

- DPLL algorithm
  - Decide : A $\rightarrow$ true

Context

$A \rightarrow T^{d}$

DPLL

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

- DPLL algorithm
  - Decide : $A \to$ true

Alternatively, can view context as set of literals

Context
A^d

DPLL

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

- DPLL algorithm
  - Decide : A $\rightarrow$ true
  - Propagate : B $\rightarrow$ true

Context
A^d
B

DPLL

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

- DPLL algorithm
  - Decide : A $\rightarrow$ true
  - Propagate : B $\rightarrow$ true
  - Propagate : C $\rightarrow$ false

Context
A^d
B
$\neg C$

DPLL

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

$\Rightarrow$ Conflicting clause!
(all literals are false)

- DPLL algorithm
  - Decide : A $\rightarrow$ true
  - Propagate : B $\rightarrow$ true
  - Propagate : C $\rightarrow$ false

Context
A^d
B
$\neg C$

DPLL

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

- DPLL algorithm
  - Decide : A $\rightarrow$ true
  - Propagate : B $\rightarrow$ true
  - Propagate : C $\rightarrow$ false

$\Rightarrow$ Conflicting clause!

(all literals are false)

…*backtrack* on a decision

Context
A^d
B
$\neg C$

DPLL

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

- DPLL algorithm
  - Backtrack : A $\rightarrow$ false

Context

$\neg A$

DPLL

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

- DPLL algorithm
  - Backtrack : A $\rightarrow$ false
  - Propagate : D $\rightarrow$ true

Context

$\neg A$

D

DPLL

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

- DPLL algorithm
  - Backtrack : A $\rightarrow$ false
  - Propagate : D $\rightarrow$ true
  - Decide : B $\rightarrow$ false

Context
$\neg A$
D
B^d

DPLL

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

- DPLL algorithm
  - Backtrack : A $\rightarrow$ false
  - Propagate : D $\rightarrow$ true
  - Decide : B $\rightarrow$ false

$\Rightarrow$ Input is SAT by interpretation where $\{A \rightarrow \bot, B \rightarrow \bot, D \rightarrow T\}$

Context

$\neg A$

D

B^d

DPLL

- Important optimizations:
  - Two watched literals
  - Non-chronological backtracking
  - Conflict-driven clause learning (CDCL)
  - Decision heuristics
  - Preprocessing / in-processing

SAT

- Using an encoding of problems into propositional logic:
  - **Pros** : Decidable, very efficient CDCL-based SAT solvers available
  - **Cons** : Not expressive, may require exponentially large encoding

    $\Rightarrow$ Motivation for Satisfiability *Modulo Theories*

SMT solvers handle formulas like:

$(x+1>0 \vee x+y>0) \wedge (x<0 \vee x+y>4) \wedge \neg x+y>0$

SMT solvers handle formulas like:

$(x+1>0 \vee x+y>0) \wedge (x<0 \vee x+y>4) \wedge \neg x+y>0$

- …using DPLL(**T**) algorithm for satisfiability modulo T
  - Extends DPLL algorithm to incorporate reasoning about a theory **T**
  - Combines:
    - Off-the-shelf CDCL-based SAT solver
    - *Theory Solver for T*

DPLL(T)

$$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0$$

- DPLL(LIA) algorithm

Invoke DPLL(T) for theory T = LIA (linear integer arithmetic)

DPLL(T)

$(x+1>0 \vee x+y>0) \wedge (x<0 \vee x+y>4) \wedge \neg x+y>0$

- DPLL(LIA) algorithm

Context

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false

Context
$\neg x+y>0$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true

Context

$\neg x+y>0$

$x+1>0$

DPLL(T)

$(\, x+1>0 \lor x+y>0 \,) \land (\, x<0 \lor x+y>4) \land \neg x+y>0$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Decide : $x<0 \rightarrow$ true

Context
$\neg x+y>0$
$x+1>0$
$x<0^d$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Decide : $x<0 \rightarrow$ true

$\Rightarrow$ ***Unlike propositional SAT case, we must check T-satisfiability of context***

Context
$\neg x+y>0$
$x+1>0$
$x<0^d$

DPLL(T)

$(x+1>0 \vee x+y>0) \wedge (x<0 \vee x+y>4) \wedge \neg x+y>0$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Decide : $x<0 \rightarrow$ true
  - Invoke theory solver for LIA on context : $\{ x+1>0, \neg x+y>0, x<0 \}$

Context

$\neg x+y>0$

$x+1>0$

$x<0^d$

DPLL(T)

$$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0$$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Decide : $x<0 \rightarrow$ true
  - Invoke theory solver for LIA on context : $\{ x+1>0, \neg x+y>0, x<0 \}$

Context is LIA-unsatisfiable!

$\Rightarrow$ one of $x+1>0$, $x<0$ must be false

Context
$\neg x+y>0$
$x+1>0$
$x<0^d$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land (\neg x+1>0 \lor \neg x<0)$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Decide : $x<0 \rightarrow$ true
  - Invoke theory solver for LIA on context : $\{ x+1>0, \neg x+y>0, x<0 \}$
    - Add *theory lemma* $(\neg x+1>0 \lor \neg x<0)$

Context
$\neg x+y>0$
$x+1>0$
$x<0^d$

DPLL(T)

($x+1>0 \lor x+y>0$) $\land$ ($x<0 \lor x+y>4$) $\land$ $\neg x+y>0$ $\land$
($\neg x+1>0 \lor \neg x<0$) $\Rightarrow$ Conflicting clause!

…*backtrack* on a decision

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Decide : $x<0 \rightarrow$ true
  - Invoke theory solver for LIA on context : { $x+1>0$, $\neg x+y>0$, $x<0$ }
    - Add *theory lemma* ($\neg x+1>0 \lor \neg x<0$)

Context

$\neg x+y>0$

$x+1>0$

$x<0^d$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land (\neg x+1>0 \lor \neg x<0)$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true

Context
$\neg x+y>0$
$x+1>0$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land$
$(\neg x+1>0 \lor \neg x<0)$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - *Propagate* : $x<0 \rightarrow$ false

Context

$\neg x+y>0$

$x+1>0$

$\neg x<0$

DPLL(T)

$(\ x+1>0 \lor x+y>0\ ) \land (\ x<0 \lor x+y>4) \land \neg x+y>0 \land$
$(\neg x+1>0 \lor \neg x<0\ )$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Propagate : $x<0 \rightarrow$ false
  - Propagate : $x+y>4 \rightarrow$ true

Context

$\neg x+y>0$
$x+1>0$
$\neg x<0$
$x+y>4$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land$
$(\neg x+1>0 \lor \neg x<0)$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Propagate : $x<0 \rightarrow$ false
  - Propagate : $x+y>4 \rightarrow$ true
  - Invoke theory solver for LIA on: $\{ x+1>0, \neg x+y>0, \neg x<0, x+y>4 \}$

Context
$\neg x+y>0$
$x+1>0$
$\neg x<0$
$x+y>4$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land (\neg x+1>0 \lor \neg x<0)$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Propagate : $x<0 \rightarrow$ false
  - Propagate : $x+y>4 \rightarrow$ true
  - Invoke theory solver for LIA on: $\{ x+1>0, \neg x+y>0, \neg x<0, x+y>4 \}$

Context is LIA-unsatisfiable!
$\Rightarrow$ one of $\neg x+y>0$, $x+y>4$ must be false

Context
$\neg x+y>0$
$x+1>0$
$\neg x<0$
$x+y>4$

DPLL(T)

$$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land$$
$$(\neg x+1>0 \lor \neg x<0) \land (x+y>0 \lor \neg x+y>4)$$

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Propagate : $x<0 \rightarrow$ false
  - Propagate : $x+y>4 \rightarrow$ true
  - Invoke theory solver for LIA on: $\{ x+1>0, \neg x+y>0, \neg x<0, x+y>4 \}$
    - Add *theory lemma* $(x+y>0 \lor \neg x+y>4)$

Context
$\neg x+y>0$
$x+1>0$
$\neg x<0$
$x+y>4$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land$
$(\neg x+1>0 \lor \neg x<0) \land (x+y>0 \lor \neg x+y>4)$

$\Rightarrow$ Conflicting clause!

...no decision to backtrack

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Propagate : $x<0 \rightarrow$ false
  - Propagate : $x+y>4 \rightarrow$ true
  - Invoke theory solver for LIA on: $\{ x+1>0, \neg x+y>0, \neg x<0, x+y>4 \}$
    - Add *theory lemma* $(x+y>0 \lor \neg x+y>4)$

Context

$\neg x+y>0$

$x+1>0$

$\neg x<0$

$x+y>4$

DPLL(T)

$(x+1>0 \lor x+y>0) \land (x<0 \lor x+y>4) \land \neg x+y>0 \land$
$(\neg x+1>0 \lor \neg x<0) \land (x+y>0 \lor \neg x+y>4)$

$\Rightarrow$ Conflicting clause!

…no decision to backtrack

- DPLL(LIA) algorithm
  - Propagate : $x+y>0 \rightarrow$ false
  - Propagate : $x+1>0 \rightarrow$ true
  - Propagate : $x<0 \rightarrow$ false
  - Propagate : $x+y>4 \rightarrow$ true
  - Invoke theory solver for LIA on: $\{ x+1>0, \neg x+y>0, \neg x<0, x+y>4 \}$
    - Add *theory lemma* $(x+y>0 \lor \neg x+y>4)$

$\Rightarrow$ *Input is* LIA-unsat

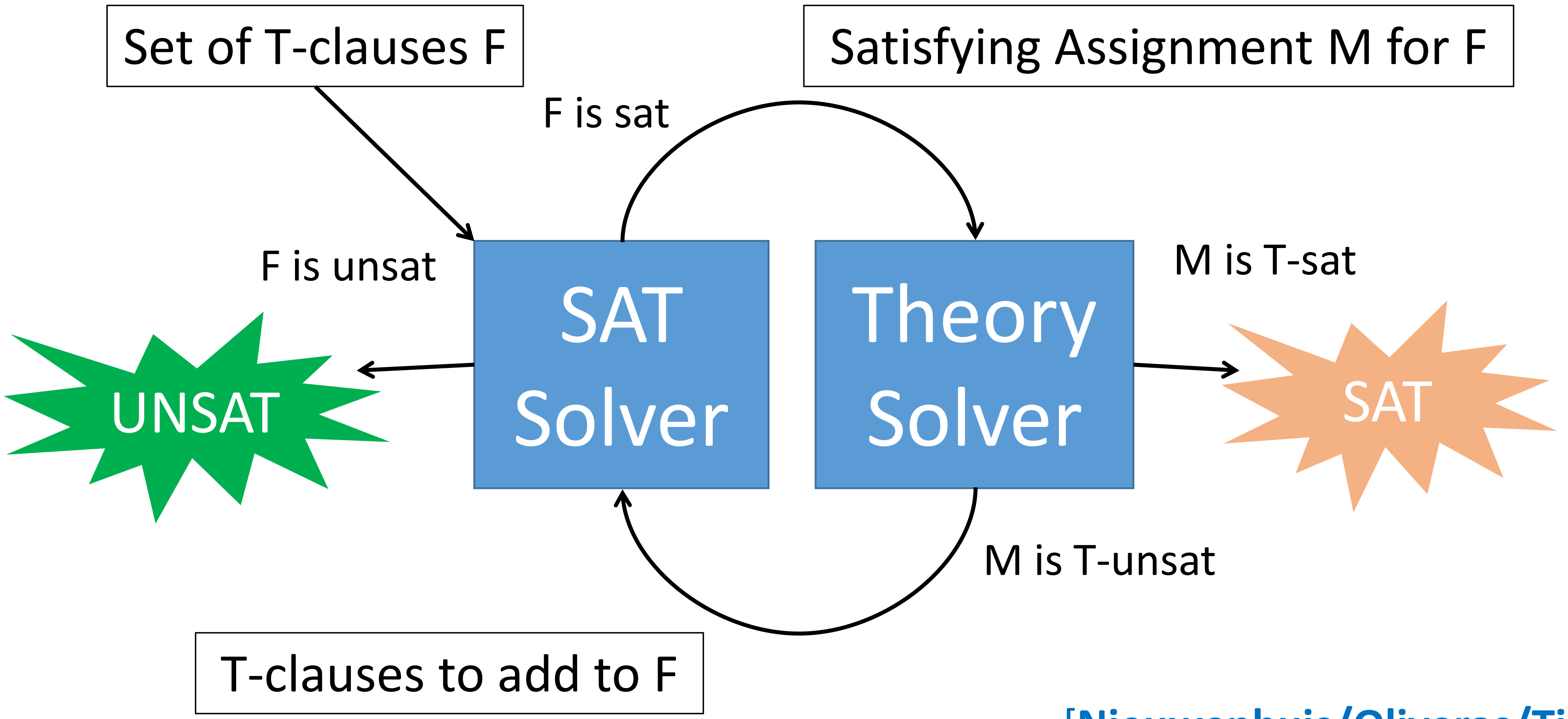
Context

$\neg x+y>0$
$x+1>0$
$\neg x<0$
$x+y>4$

Encoding in *.smt2 format

```
(set-logic QF_LIA)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (or (> (+ x 1) 0) (> (+ x y) 0)))
(assert (or (< x 0) (> (+ x y) 4)))
(assert (not (> (+ x y) 0)))
(check-sat)
```

DPLL(T)



[Nieuwenhuis/Oliveras/Tinelli 2006]

Design of DPLL(T) Theory Solvers

- A DPLL(T) theory solver:
  - Should produce models when M is T-satisfiable
  - Should produce T-conflicts of minimal size when M is T-unsatisfiable
  - Should be designed to work *incrementally*
    - M is constantly being appended to/backtracked upon
  - Should cooperate with other theory solvers when combining theories

DPLL(T) Theory Solvers : Examples

- SMT solvers incorporate:
  - Theory solvers that are *decision procedures* for e.g.:
    - Theory of Equality and Uninterpreted Functions (EUF)
    - Theory of Linear Integer/Real Arithmetic
    - Theory of Arrays
    - Theory of Bit Vectors
    - Theory of Inductive Datatypes
    - …and many others
  - Theory solvers that are *incomplete procedures* for e.g.:
    - Theory of Non-Linear Integer Arithmetic
    - Theory of Strings + Length constraints

DPLL(T) Theory Solvers : Examples

- SMT solvers incorporate:
  - Theory solvers that are *decision procedures* for e.g.:
    - Theory of Equality and Uninterpreted Functions (EUF)
    - Theory of Linear Integer/Real Arithmetic
    - Theory of Arrays
    - Theory of Bit Vectors
    - **Theory of Inductive Datatypes** } Focus of the next part
    - …and many others
  - Theory solvers that are *incomplete procedures* for e.g.:
    - Theory of Non-Linear Integer Arithmetic
    - Theory of Strings + Length constraints

Theory of Inductive Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

Theory of Inductive Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

- Theory of Inductive Datatypes (DT)
  1. Terms with different constructors are distinct
     - red $\neq$ green
  2. Constructors are injective
     - If cons(c_1, l_1) = cons(c_2, l_2), then $c_1 = c_2$ and $l_1 = l_2$
  3. Terms of a datatype must have one of its constructors as its topmost symbol
     - Each c is such that c = red or c = green or c = blue
  4. Selectors access subfields
     - head(cons(c, l)) = c
  5. Terms do not contain themselves as subterms
     - l $\neq$ cons(c, l)

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

Context

$cons(x,nil)=cons(y,z) \wedge (x=red \vee \neg x=y) \wedge y = green$

- DPLL(DT) algorithm

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) $\wedge$ (x=red $\vee$ $\neg$x=y) $\wedge$ y = green

Context

cons(x,nil)=cons(y,z)
y=green

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) $\wedge$ (x=red $\vee$ $\neg$x=y) $\wedge$ y = green

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true
  - Decide : x=red $\rightarrow$ true

Context
cons(x,nil)=cons(y,z) y=green x=red^{d}

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) $\wedge$ (x=red $\vee$ $\neg$x=y) $\wedge$ y = green

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true
  - Decide : x=red $\rightarrow$ true
  - Invoke DT solver on {cons(x,nil)=cons(y,z),y=green,x=red}

Context

cons(x,nil)=cons(y,z)
y=green
$x=red^d$

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

$cons(x,nil)=cons(y,z) \wedge (x=red \vee \neg x=y) \wedge y = green$

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true
  - Decide : x=red $\rightarrow$ true
  - Invoke DT solver on {cons(x,nil)=cons(y,z),y=green,x=red}

    $\Rightarrow$DT-unsatisfiable

    Since cons(x, nil) = cons(y, nil), it must be that x = y, but x = red and y = green and red $\neq$ green

Context
cons(x,nil)=cons(y,z)
y=green
$x=red^d$

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) $\wedge$ (x=red $\vee$ $\neg$x=y) $\wedge$ y = green
($\neg$cons(x,nil)=cons(y,z) $\vee$ $\neg$ y=green $\vee$ $\neg$ x=red)

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true
  - Decide : x=red $\rightarrow$ true
  - Invoke DT solver on {cons(x,nil)=cons(y,z),y=green,x=red}

    $\Rightarrow$ …add theory lemma

Context
cons(x,nil)=cons(y,z) y=green x=red^{d}

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) ∧ (x=red ∨ ¬x=y) ∧ y = green
(¬cons(x,nil)=cons(y,z) ∨ ¬ y=green ∨ ¬ x=red)

⇒ Conflicting clause!

…*backtrack* on a decision

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) → true
  - Propagate : y=green → true
  - Decide : x=red → true
  - Invoke DT solver on {cons(x,nil)=cons(y,z),y=green,x=red}

    ⇒ …add theory lemma

Context
cons(x,nil)=cons(y,z)
y=green
$x=red^d$

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) $\wedge$ (x=red $\vee$ $\neg$x=y) $\wedge$ y = green
($\neg$cons(x,nil)=cons(y,z) $\vee$ $\neg$ y=green $\vee$ $\neg$ x=red)

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true

Context

cons(x,nil)=cons(y,z)
y=green

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) ∧ (x=red ∨ ¬x=y) ∧ y = green
(¬cons(x,nil)=cons(y,z) ∨ ¬ y=green ∨ ¬ x=red)

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) → true
  - Propagate : y=green → true
  - Propagate : x=red → false

Context
cons(x,nil)=cons(y,z) y=green ¬x=red

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

$cons(x,nil)=cons(y,z) \wedge (x=red \vee \neg x=y) \wedge y = green$
$(\neg cons(x,nil)=cons(y,z) \vee \neg y=green \vee \neg x=red)$

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true
  - Propagate : x=red $\rightarrow$ false
  - Propagate : x=y $\rightarrow$ false

Context
cons(x,nil)=cons(y,z)
y=green
$\neg$x=red
$\neg$x=y

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

$cons(x,nil)=cons(y,z) \wedge (x=red \vee \neg x=y) \wedge y = green$
$(\neg cons(x,nil)=cons(y,z) \vee \neg\, y=green \vee \neg\, x=red)$

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true
  - Propagate : x=red $\rightarrow$ false
  - Propagate : x=y $\rightarrow$ false
  - Invoke DT solver on {cons(x,nil)=cons(y,z), y=green, x=red, ¬x=y}

Context
cons(x,nil)=cons(y,z)
y=green
¬x=red
¬x=y

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) ∧ (x=red ∨ ¬x=y) ∧ y = green
(¬cons(x,nil)=cons(y,z) ∨ ¬ y=green ∨ ¬ x=red)
(¬cons(x,nil)=cons(y,z) ∨ x=y)

- DPLL(DT) algorithm
  - Propagate : cons(x,nil)=cons(y,z) → true
  - Propagate : y=green → true
  - Propagate : x=red → false
  - Propagate : x=y → false
  - Invoke DT solver on {cons(x,nil)=cons(y,z), y=green, x=red, ¬x=y}

    ⇒ DT-unsatisfiable, add theory lemma

Context
cons(x,nil)=cons(y,z)
y=green
¬x=red
¬x=y

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

cons(x,nil)=cons(y,z) ∧ (x=red ∨ ¬x=y) ∧ y = green
(¬cons(x,nil)=cons(y,z) ∨ ¬ y=green ∨ ¬ x=red)
(¬cons(x,nil)=cons(y,z) ∨ x=y) ⇒ Conflicting clause!

- DPLL(DT) algorithm *...no decisions*
  - Propagate : cons(x,nil)=cons(y,z) → true
  - Propagate : y=green → true
  - Propagate : x=red → false
  - Propagate : x=y → false
  - Invoke DT solver on {cons(x,nil)=cons(y,z), y=green, x=red, ¬x=y}

    ⇒ DT-unsatisfiable, add theory lemma

Context
cons(x,nil)=cons(y,z)
y=green
¬x=red
¬x=y

Datatypes : Example

ClrList := cons(head : Clr, tail : ClrList) | nil
Clr := red | green | blue

$\text{cons}(x,\text{nil})=\text{cons}(y,z) \wedge (x=\text{red} \vee \neg x=y) \wedge y = \text{green}$

$(\neg \text{cons}(x,\text{nil})=\text{cons}(y,z) \vee \neg\, y=\text{green} \vee \neg\, x=\text{red})$

$(\neg \text{cons}(x,\text{nil})=\text{cons}(y,z) \vee x=y)$ $\Rightarrow$ Conflicting clause!

- DPLL(DT) algorithm *...no decisions*
  - Propagate : cons(x,nil)=cons(y,z) $\rightarrow$ true
  - Propagate : y=green $\rightarrow$ true
  - Propagate : x=red $\rightarrow$ false
  - Propagate : x=y $\rightarrow$ false
  - Invoke DT solver on {cons(x,nil)=cons(y,z), y=green, x=red, $\neg$x=y}

$\Rightarrow$ Input is DT-unsat

Context
cons(x,nil)=cons(y,z)
y=green
$\neg$x=red
$\neg$x=y

Encoding in *.smt2

```
(set-logic QF_DT)
(declare-datatypes ((ClrList 0) (Clr 0))(
 ((cons (head Clr) (tail ClrList)) (nil))
 ((red) (green) (blue))))
(declare-fun x () Clr)
(declare-fun y () Clr)
(declare-fun z () ClrList)
(assert (= (cons x nil) (cons y z)))
(assert (or (= x red) (not (= x y))))
(assert (= y green))
(check-sat)
```

Theory Combination

- What if we have:

$$\text{IntList} := \text{cons}(\ \text{head} : \text{Int},\ \text{tail} : \text{IntList}\ )\ |\ \text{nil}$$

- Example input:

$$(\ \text{head}(\ x\ )+3 = y \vee x = \text{cons}(\ y+1,\ \text{nil}\ )\ ) \wedge \text{head}(\ x\ ) > y+1$$

$\Rightarrow$Requires reasoning about datatypes *and* integers!

Theory Combination

- What if we have:

$$\text{IntList} := \text{cons}(\ \text{head} : \text{Int},\ \text{tail} : \text{IntList}\ )\ |\ \text{nil}$$

- Example input:

$$(\ \text{head}(\ x\ )+3 = y \vee x = \text{cons}(\ y+1, \text{nil}\ )\ ) \wedge \text{head}(\ x\ ) > y+1$$

- Idea:
  - Use DPLL(LIA+DT): find satisfying assignments $M = M_{LIA} \cup M_{DT}$
    - Use existing solver for LIA to check if M_{LIA} is LIA-satisfiable
    - Use existing solver for DT to check if M_{DT} is DT-satisfiable

Do not need to write a new theory solver for LIA+DT

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

(head(x)+3 = y ∨ x = cons(y+1, nil)) ∧ head(x) > y+1

Context

- DPLL(LIA+DT) algorithm

$IntList := cons(head : Int, tail : IntList) \mid nil$

Theory Combination

$(\quad u_1 \quad +3 = y \lor x = cons(\; u_2 \; , nil)) \land \quad u_1 \quad > y+1 \land$
$u_1 = head(x) \land u_2 = y+1$

Context

- DPLL(LIA+DT) algorithm
  - ⇒First, purify the input
    - Introduce *shared variables* u_1, u_2

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

Context

- DPLL(LIA+DT) algorithm

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true

Context
$u_1>y+1$ $u_1=head(x)$ $u_2=y+1$

IntList := cons(head : Int, tail : IntList) | nil

Theory Combination

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

Context

$u_1>y+1$

$u_1=head(x)$

$u_2=y+1$

$u_1+3=y^d$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Decide : $u_1+3=y \rightarrow$ true

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \lor x = cons(u_2, nil)) \land u_1>y+1 \land u_1=head(x) \land u_2=y+1$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Decide : $u_1+3=y \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x)\}$ … DT-satisfiable

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$u_1+3=y^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \lor x = \text{cons}(u_2, \text{nil})) \land u_1>y+1 \land u_1=\text{head}(x) \land u_2=y+1$

Context
$u_1>y+1$
$u_1=\text{head}(x)$
$u_2=y+1$
$u_1+3=y^d$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=\text{head}(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Decide : $u_1+3=y \rightarrow$ true
  - Invoke DT solver on $\{u_1=\text{head}(x)\}$ … DT-satisfiable
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, u_1+3=y\}$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

$(\neg u_1>y+1 \vee \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Decide : $u_1+3=y \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x)\}$ … DT-satisfiable
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, u_1+3=y\}$…LIA-unsatisfiable

    $\Rightarrow$ Add theory lemma

Context

$u_1>y+1$

$u_1=head(x)$

$u_2=y+1$

$u_1+3=y^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

$(\neg u_1>y+1 \vee \neg u_1+3=y)$

$\Rightarrow$ Conflicting clause!

…*backtrack* on a decision

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Decide : $u_1+3=y \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x)\}$ … DT-satisfiable
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, u_1+3=y\}$…LIA-unsatisfiable

    $\Rightarrow$ Add theory lemma

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$u_1+3=y^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \lor x = cons(u_2, nil)) \land u_1>y+1 \land u_1=head(x) \land u_2=y+1$

$(\neg u_1>y+1 \lor \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true

Context
$u_1>y+1$ $u_1=head(x)$ $u_2=y+1$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \lor x = \text{cons}(u_2, \text{nil})) \land u_1>y+1 \land u_1=\text{head}(x) \land u_2=y+1$

$(\neg u_1>y+1 \lor \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=\text{head}(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false

Context
$u_1>y+1$
$u_1=\text{head}(x)$
$u_2=y+1$
$\neg u_1+3=y$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1 \wedge$
$(\neg u_1>y+1 \vee \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil)\}$ … DT-satisfiable

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil)\}$ … DT-satisfiable
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, \neg u_1+3=y\}$ … LIA-satisfiable

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

$(\neg u_1>y+1 \vee \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil)\}$ … DT-satisfiable
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, \neg u_1+3=y\}$ … LIA-satisfiable

$\Rightarrow$Theory solvers must agree on shared variables u_1, u_2

Context

$u_1>y+1$

$u_1=head(x)$

$u_2=y+1$

$\neg u_1+3=y$

$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

$(\neg u_1>y+1 \vee \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil)\}$ … DT-satisfiable, $u_1=u_2$
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, \neg u_1+3=y\}$ … LIA-satisfiable

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \lor x = cons(u_2, nil)) \land u_1>y+1 \land u_1=head(x) \land u_2=y+1$

$(\neg u_1>y+1 \lor \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil)\}$ … DT-satisfiable, $u_1=u_2$
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, \neg u_1+3=y\}$ … LIA-satisfiable, $u_1 \neq u_2$

Context

$u_1>y+1$

$u_1=head(x)$

$u_2=y+1$

$\neg u_1+3=y$

$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil)\}$ … DT-satisfiable, $\mathbf{u_1=u_2}$
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, \neg u_1+3=y\}$ … LIA-satisfiable, $\mathbf{u_1 \neq u_2}$

$\Rightarrow$ *Theory solvers do not agree on $u_1=u_2$!*

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

$(\neg u_1>y+1 \vee \neg u_1+3=y) \wedge (u_1=u_2 \vee \neg u_1=u_2)$

Context

$u_1>y+1$

$u_1=head(x)$

$u_2=y+1$

$\neg u_1+3=y$

$x=cons(u_2, nil)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil)\}$ … DT-satisfiable, $u_1=u_2$
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, \neg u_1+3=y\}$ … LIA-satisfiable, $u_1 \neq u_2$

$\Rightarrow$ *Theory solvers do not agree on $u_1=u_2$ … add splitting lemma for u_1, u_2*

Theory Combination

$IntList := cons(\ head : Int,\ tail : IntList\ )\ |\ nil$

$(u_1+3=y \lor x = cons(u_2, nil)\ ) \land u_1>y+1 \land u_1=head(x) \land u_2=y+1$

$(\neg u_1>y+1 \lor \neg u_1+3=y) \land (\ u_1=u_2 \lor \neg u_1=u_2\ )$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$

$(\neg u_1>y+1 \vee \neg u_1+3=y) \wedge (u_1=u_2 \vee \neg u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Decide : $u_1=u_2 \rightarrow$ true

Context

$u_1>y+1$

$u_1=head(x)$

$u_2=y+1$

$\neg u_1+3=y$

$x=cons(u_2, nil)$

$u_1=u_2{}^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \lor x = cons(u_2, nil)) \land u_1>y+1 \land u_1=head(x) \land u_2=y+1$
$(\neg u_1>y+1 \lor \neg u_1+3=y) \land (u_1=u_2 \lor \neg u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Decide : $u_1=u_2 \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil), u_1=u_2\}$ … DT-satisfiable

Context

$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$
$u_1=u_2{}^d$

IntList := cons(head : Int, tail : IntList) | nil

Theory Combination

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y) \wedge (u_1=u_2 \vee \neg u_1=u_2) \wedge (\neg u_1>y+1 \vee$
$\neg u_2=y+1 \vee \neg u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Decide : $u_1=u_2 \rightarrow$ true
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil), u_1=u_2\}$ … DT-satisfiable
  - Invoke LIA solver on $\{u_1>y+1, u_2=y+1, \neg u_1+3=y, u_1=u_2\}$ … LIA-unsatisfiable

Context

$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$
$u_1=u_2{}^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y) \wedge (u_1=u_2 \vee \neg u_1=u_2) \wedge (\neg u_1>y+1 \vee$
$\neg u_2=y+1 \vee \neg u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true

Context

$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y) \wedge (u_1=u_2 \vee \neg u_1=u_2) \wedge (\neg u_1>y+1 \vee$
$\neg u_2=y+1 \vee \neg u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Propagate : $u_1=u_2 \rightarrow$ false

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$
$\neg u_1=u_2^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y) \wedge (u_1=u_2 \vee \neg u_1=u_2) \wedge (\neg u_1>y+1 \vee$
$\neg u_2=y+1 \vee \neg u_1=u_2) \wedge (\neg u_1=head(x) \vee \neg x=cons(u_2, nil) \vee u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Propagate : $u_1=u_2 \rightarrow$ false
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil), \neg u_1=u_2\}$ …DT-unsat

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$
$\neg u_1=u_2{}^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \lor x = cons(u_2, nil)) \land u_1>y+1 \land u_1=head(x) \land u_2=y+1$
$(\neg u_1>y+1 \lor \neg u_1+3=y) \land (u_1=u_2 \lor \neg u_1=u_2) \land (\neg u_1>y+1 \lor$
$\neg u_2=y+1 \lor \neg u_1=u_2) \land (\neg u_1=head(x) \lor \neg x=cons(u_2, nil) \lor u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Propagate : $u_1=u_2 \rightarrow$ false
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil), \neg u_1=u_2\}$ …DT-unsat

$\Rightarrow$ Conflicting clause!

…no decisions

Context
$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$
$\neg u_1=u_2{}^d$

Theory Combination

IntList := cons(head : Int, tail : IntList) | nil

$(u_1+3=y \vee x = cons(u_2, nil)) \wedge u_1>y+1 \wedge u_1=head(x) \wedge u_2=y+1$
$(\neg u_1>y+1 \vee \neg u_1+3=y) \wedge (u_1=u_2 \vee \neg u_1=u_2) \wedge (\neg u_1>y+1 \vee$
$\neg u_2=y+1 \vee \neg u_1=u_2) \wedge (\neg u_1=head(x) \vee \neg x=cons(u_2, nil) \vee u_1=u_2)$

- DPLL(LIA+DT) algorithm
  - Propagate : $u_1>y+1 \rightarrow$ true
  - Propagate : $u_1=head(x) \rightarrow$ true
  - Propagate : $u_2=y+1 \rightarrow$ true
  - Propagate : $u_1+3=y \rightarrow$ false
  - Propagate : $x=cons(u_2, nil) \rightarrow$ true
  - Propagate : $u_1=u_2 \rightarrow$ false
  - Invoke DT solver on $\{u_1=head(x), x=cons(u_2, nil), \neg u_1=u_2\}$ …DT-unsat

$\Rightarrow$ Conflicting clause!

…no decisions

$\Rightarrow$ Input is LIA+DT-unsat

Context

$u_1>y+1$
$u_1=head(x)$
$u_2=y+1$
$\neg u_1+3=y$
$x=cons(u_2, nil)$
$\neg u_1=u_2{}^d$

Encoding in *.smt2

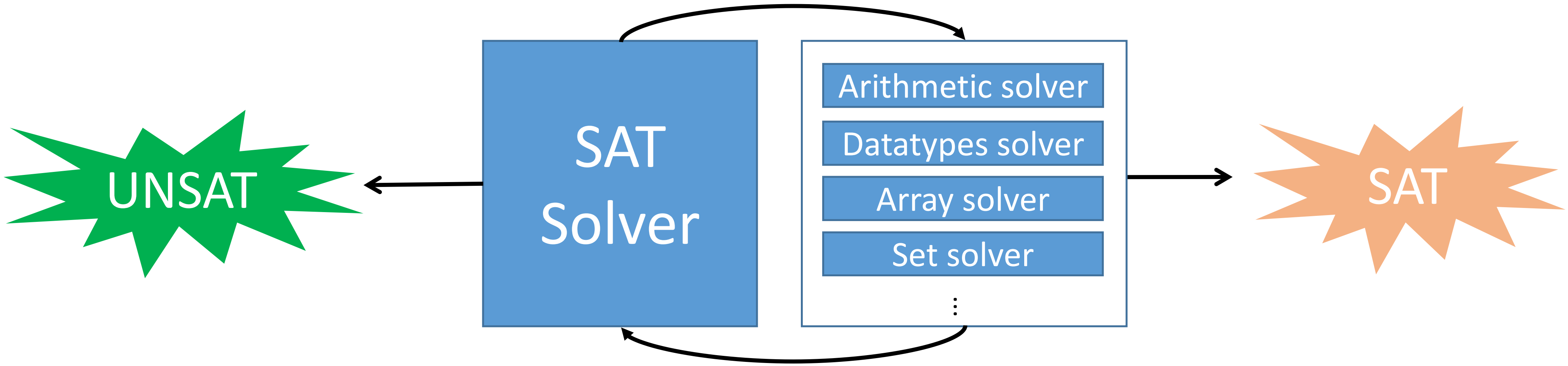
```
(set-logic QF_DTLIA)
(declare-datatypes ((IntList 0))(
 ((cons (head Int) (tail IntList)) (nil))))
(declare-fun x () IntList)
(declare-fun y () Int)
(assert (= (+ (head x) 3) y))
(assert (= x (cons (+ y 1) nil)))
(assert (> (head x) (+ y 1)))
(check-sat)
```

DPLL(T) : Theory Combination

- Nelson-Oppen Theory Combination
  - SMT solvers use preexisting theory solvers for combined theories $T_1 + \ldots + T_n$
  - Partition and distribute context M to T_1-solver, …, T_n-solver
    - If any T_i-solver says "unsat", then M is unsatisfiable
    - If each T_i-solver says "sat", then solvers must agree on shared variables
  - Requires theory solvers to:
    - Have disjoint signatures
      - E.g. arithmetic has functions { +, <, 0, 1, … }, datatypes has functions { cons, head, tail, … }
    - Know equalities/disequalities between shared variables
      - E.g. are $u_1 = u_2$ equal?
    - Theories agree on cardinalities for shared types
      - E.g. LIA and DT may agree that Int has infinite cardinality

DPLL(T) : Summary

- SMT solvers use
  - DPLL(T) algorithm for theory T, which uses:
    - Off-the-shelf SAT solver
    - Theory solver(s) for T
  - Nelson-Oppen theory combination for combined theories $T_1 + T_2$, which uses:
    - Existing theory solvers for T_1 and T_2, combines them using a generic method



Examples

Contract-Based Verification

```
@precondition: P1[ x_in, y_in ]
void f( int& x, int& y )
{
      ...
}
@ensures: P2[ x_in, y_in, x_out, y_out]
```

Property P_1 should hold for all inputs x_{in}, y_{in} to function `f`

Property P_2 is guaranteed to hold for x_{out}, y_{out} (the state of `x`, `y` after calling `f`)

Contract-Based Verification

@precondition: $x_{in} > y_{in}$

```
0 void swap(int& x, int& y)
  {
1       x := x + y;
2       y := x - y;
3       x := x - y;
  }
```

@ensures: $x_{out}=y_{in} \wedge y_{out}=x_{in}$ **?**

EXAMPLE A1…

Contract-Based Verification

@precondition: $x_{in} > y_{in}$

```
0 void swap(int& x, int& y)
  {
1       x := x + y;
2       y := x - y;
3       x := x - y;
  }
```

@ensures: $x_{out} = y_{in} \wedge y_{out} = x_{in}$

Contract-Based Verification

@precondition: $\mathbf{x_{in}>y_{in}}$

```
0 void swap(int& x, int& y)
  {
1       x := x + y;
2       y := x - y;
3       x := x - y;
  }
```

@ensures: $x_{out}=y_{in} \wedge y_{out}=x_{in}$

Is this necessary?

EXAMPLE A1-uc…

Contract-Based Verification

```
  @precondition: ~~x_in>y_in~~
0 void swap(int& x, int& y)
  {
1       x := x + y;
2       y := x - y;
3       x := x - y;
  }
  @ensures: x_out=y_in ∧y_out=x_in
```

Not necessary

$x_{in}>y_{in}$ is not in the unsatisfiable core in the *proof* of $\mathbf{x_{out}=y_{in} \land y_{out}=x_{in}}$

$\Rightarrow$ *precondition is not necessary to show properties of* `swap`

Contract-Based Verification

```
0 void swap(int& x, int& y)
  {
1       x := x + y;
2       y := x - y;
3       x := x - y;
  }
```

@ensures: $x_{out} = y_{in} \wedge y_{out} = x_{in}$

Contract-Based Verification

```
void swap(int& x, int& y)
{
      x := x + y;
      y := x - y;
      x := x - y;
}
```

@ensures: $x_{out}=y_{in} \wedge y_{out}=x_{in}$

```
0 void setMax(int& x, int& y)
  {
1       if( y>x ){
          swap( x, y );
2       }
  }
```

@ensures: $x_{out}>y_{out}$?

EXAMPLE A2…

Contract-Based Verification

```
void swap(int& x, int& y)
{
      x := x + y;
      y := x - y;
      x := x - y;
}
```

@ensures: $x_{out}=y_{in} \wedge y_{out}=x_{in}$

```
0  void setMax(int& x, int& y)
   {
1        if( y>x ){
           swap( x, y );
2        }
   }
```

~~@ensures: $x_{out}>y_{out}$~~

...when $x_{in}=0$ and $y_{in}=0$

Contract-Based Verification

```
void swap(int& x, int& y)
{
      x := x + y;
      y := x - y;
      x := x - y;
}
```

@ensures: $x_{out}=y_{in} \wedge y_{out}=x_{in}$

@precondition: $\mathbf{x_{in} \neq y_{in}}$

```
0 void setMax(int& x, int& y)
  {
1       if( y>x ){
           swap( x, y );
2       }
  }
```

@ensures: $\mathbf{x_{out} > y_{out}}$

Contract-Based Verification

@precondition: x_{in}>5

```
0 void resetX(int& x, int& y)
  {
1       if( x*y==3 ){
           x=-1;
        }
2 }
```

@ensures: x_{out}>5 ?

EXAMPLE A3…

Contract-Based Verification

```
  @precondition: x_in>5
0 void resetX(int& x, int& y)
  {
1       if( x*y==3 ){
          x=-1;
        }
2 }
  @ensures: x_out>5
```

…using heuristic techniques for non-linear arithmetic
(incomplete, can get lucky)

Contract-Based Verification

```
  @precondition: res_in==0
0 void cubes(int a, int b, int c, int& res)
  {
1       if( (a*a*a)+(b*b*b)+(c*c*c)==33 ){
          res = 1;
        }
2 }
  @ensures: res_out==0 ?
```

EXAMPLE A4…

Contract-Based Verification

```
  @precondition: res_in==0
0 void cubes(int a, int b, int c, int& res)
  {
1     if( (a*a*a)+(b*b*b)+(c*c*c)==33 ){
        res = 1;
      }
2 }
  @ensures: res_out==0 ?
```

…the SMT solver will (typically) not solve open problems in mathematics!