

# Using Satisfiability Modulo Theories for Inductive Verification of Lustre Programs

A. Franzén

University of Trento

Workshop on Bounded Model Checking, 2005

# Introduction

- ▶ Verification of Lustre programs
- ▶ Integer and boolean streams
- ▶ Linear fragment
  - ▶ Linear arithmetic expressions
- ▶ k-induction
  - ▶ Base case: Property valid in k initial states
  - ▶ Induction step: If the property is valid in k distinct successive states, it is valid the next state.
- ▶ Incremental decision procedure

# Lustre

```
node Counter( Enable : bool ) returns ( C : int );  
  pC = int;  
let  
  pC = 0 → pre C;  
  C = if Enable then pC + 1 else pC;  
tel
```

```
node Property( Enable : bool ) returns ( P1, P2 : bool );  
  var C : int;  
let  
  C = Counter( Enable );  
  P1 = C ≥ 0;  
  P2 = Enable ⇒ C = pre C + 1;  
tel
```

# Lustre

```
node Counter( Enable : bool ) returns ( C : int );
```

```
  pC = int;
```

```
let
```

```
  pC = 0 → pre C;
```

```
  C = if Enable then pC + 1 else pC;
```

```
tel
```

```
node Property( Enable : bool ) returns ( P1, P2 : bool );
```

```
  var C : int;
```

```
let
```

```
  C = Counter( Enable );
```

```
  P1 = C ≥ 0;
```

```
  P2 = Enable ⇒ C = pre C + 1;
```

```
tel
```

# SMT by example

The formula in CNF

## A simple counter

```
node Counter( X : bool ) returns ( OK : bool );  
    var pC, C : int;  
let  
    pC = 0  $\rightarrow$  pre C;  
    C = if X then pC + 1 else pC;  
    OK = C  $\geq$  0;  
tel
```

- ▶ Translate to logic
- ▶ Assume property invalid
- ▶ Is there a variable assignment satisfying the formula?

# SMT by example

The formula in CNF

$$\begin{aligned} & \{ pC = 0 \} \\ & \{ \neg X, C = pC + 1 \} \\ & \{ X, C = pC \} \\ & \{ \neg \text{OK}, C \geq 0 \} \\ & \{ \text{OK}, C < 0 \} \end{aligned}$$

## A simple counter

```
node Counter( X : bool ) returns ( OK : bool );  
    var pC, C : int;  
let  
    pC = 0  $\rightarrow$  pre C;  
    C = if X then pC + 1 else pC;  
    OK = C  $\geq$  0;  
tel
```

- ▶ Translate to logic
- ▶ Assume property invalid
- ▶ Is there a variable assignment satisfying the formula?

# SMT by example

The formula in CNF

$$\begin{aligned} & \{ pC = 0 \} \\ & \{ \neg X, C = pC + 1 \} \\ & \{ X, C = pC \} \\ & \{ \neg \text{OK}, C \geq 0 \} \\ & \{ \text{OK}, C < 0 \} \\ & \{ \neg \text{OK} \} \end{aligned}$$

## A simple counter

```
node Counter( X : bool ) returns ( OK : bool );  
    var pC, C : int;  
let  
    pC = 0  $\rightarrow$  pre C;  
    C = if X then pC + 1 else pC;  
    OK = C  $\geq$  0;  
tel
```

- ▶ Translate to logic
- ▶ Assume property invalid
- ▶ Is there a variable assignment satisfying the formula?

# SMT by example

The formula in CNF

$$\begin{aligned} & \{ pC = 0 \} \\ & \{ \neg X, C = pC + 1 \} \\ & \{ X, C = pC \} \\ & \{ \neg \text{OK}, C \geq 0 \} \\ & \{ \text{OK}, C < 0 \} \\ & \{ \neg \text{OK} \} \end{aligned}$$

## A simple counter

```
node Counter( X : bool ) returns ( OK : bool );  
    var pC, C : int;  
let  
    pC = 0  $\rightarrow$  pre C;  
    C = if X then pC + 1 else pC;  
    OK = C  $\geq$  0;  
tel
```

- ▶ Translate to logic
- ▶ Assume property invalid
- ▶ Is there a variable assignment satisfying the formula?

# The basic algorithm

The formula in CNF

$$\begin{aligned} & \{ pC = 0 \} \\ & \{ \neg X, C = pC + 1 \} \\ & \{ X, C = pC \} \\ & \{ \neg \text{OK}, C \geq 0 \} \\ & \{ \text{OK}, C < 0 \} \\ & \{ \neg \text{OK} \} \end{aligned}$$

Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$$\begin{aligned} p_1 & \mapsto pC = 0 \\ p_2 & \mapsto C = pC + 1 \\ p_3 & \mapsto C = pC \\ p_4 & \mapsto C \geq 0 \\ p_5 & \mapsto C < 0 \end{aligned}$$

And replace all constraints with their in-place variable.

# The basic algorithm

The formula in CNF

$$\begin{aligned} & \{ pC = 0 \} \\ & \{ \neg X, C = pC + 1 \} \\ & \{ X, C = pC \} \\ & \{ \neg \text{OK}, C \geq 0 \} \\ & \{ \text{OK}, C < 0 \} \\ & \{ \neg \text{OK} \} \end{aligned}$$

## Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$$\begin{aligned} p_1 & \mapsto pC = 0 \\ p_2 & \mapsto C = pC + 1 \\ p_3 & \mapsto C = pC \\ p_4 & \mapsto C \geq 0 \\ p_5 & \mapsto C < 0 \end{aligned}$$

And replace all constraints with their in-place variable.

# The basic algorithm

The formula in CNF

$$\begin{aligned} & \{ pC = 0 \} \\ & \{ \neg X, C = pC + 1 \} \\ & \{ X, C = pC \} \\ & \{ \neg \text{OK}, C \geq 0 \} \\ & \{ \text{OK}, C < 0 \} \\ & \{ \neg \text{OK} \} \end{aligned}$$

## Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$$\begin{aligned} p_1 & \mapsto pC = 0 \\ p_2 & \mapsto C = pC + 1 \\ p_3 & \mapsto C = pC \\ p_4 & \mapsto C \geq 0 \\ p_5 & \mapsto C < 0 \end{aligned}$$

And replace all constraints with their in-place variable.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg \text{OK}, p_4 \}$   
 $\{ \text{OK}, p_5 \}$   
 $\{ \neg \text{OK} \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

## Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

And replace all constraints with their in-place variable.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg \text{OK}, p_4 \}$   
 $\{ \text{OK}, p_5 \}$   
 $\{ \neg \text{OK} \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

Step 2: Run through SAT solver

A SAT model is returned

$p_1 = \top$   
 $p_2 = \perp$   
 $p_3 = \top$   
 $p_4 = \perp$   
 $p_5 = \top$   
 $\text{OK} = \perp$

Create a constraint problem based on the in-place variables.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg \text{OK}, p_4 \}$   
 $\{ \text{OK}, p_5 \}$   
 $\{ \neg \text{OK} \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

Step 2: Run through SAT solver

A SAT model is returned

$p_1 = \top$   
 $p_2 = \perp$   
 $p_3 = \top$   
 $p_4 = \perp$   
 $p_5 = \top$   
 $\text{OK} = \perp$

Create a constraint problem based on the in-place variables.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg OK, p_4 \}$   
 $\{ OK, p_5 \}$   
 $\{ \neg OK \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

Step 2: Run through SAT solver

A SAT model is returned

$p_1 = \top$   
 $p_2 = \perp$   
 $p_3 = \top$   
 $p_4 = \perp$   
 $p_5 = \top$   
 $OK = \perp$

Create a constraint problem based on the in-place variables.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg OK, p_4 \}$   
 $\{ OK, p_5 \}$   
 $\{ \neg OK \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

Step 2: Run through SAT solver

A SAT model is returned

$p_1 = \top$   
 $p_2 = \perp$   
 $p_3 = \top$   
 $p_4 = \perp$   
 $p_5 = \top$   
 $OK = \perp$

Create a constraint problem based on the in-place variables.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg OK, p_4 \}$   
 $\{ OK, p_5 \}$   
 $\{ \neg OK \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

## Step 3: Solve constraint problem

Run constraint problem through T-solver

- (1)  $pC = 0$
- (2)  $C \neq pC + 1$
- (3)  $C = pC$
- (4)  $C < 0$
- (5)  $C < 0$

Constraints 1, 3 and 5 contradict each other. Add explanation to SAT problem.  
Goto step 2.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg OK, p_4 \}$   
 $\{ OK, p_5 \}$   
 $\{ \neg OK \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

## Step 3: Solve constraint problem

Run constraint problem through T-solver

- (1)  $pC = 0$
- (2)  $C \neq pC + 1$
- (3)  $C = pC$
- (4)  $C < 0$
- (5)  $C < 0$

Constraints 1, 3 and 5 contradict each other. Add explanation to SAT problem.

Goto step 2.

# The basic algorithm

The formula in CNF

$\{ p_1 \}$   
 $\{ \neg X, p_2 \}$   
 $\{ X, p_3 \}$   
 $\{ \neg \text{OK}, p_4 \}$   
 $\{ \text{OK}, p_5 \}$   
 $\{ \neg \text{OK} \}$   
 $\{ \neg p_1, \neg p_3, \neg p_5 \}$

$p_1 \mapsto pC = 0$   
 $p_2 \mapsto C = pC + 1$   
 $p_3 \mapsto C = pC$   
 $p_4 \mapsto C \geq 0$   
 $p_5 \mapsto C < 0$

## Step 3: Solve constraint problem

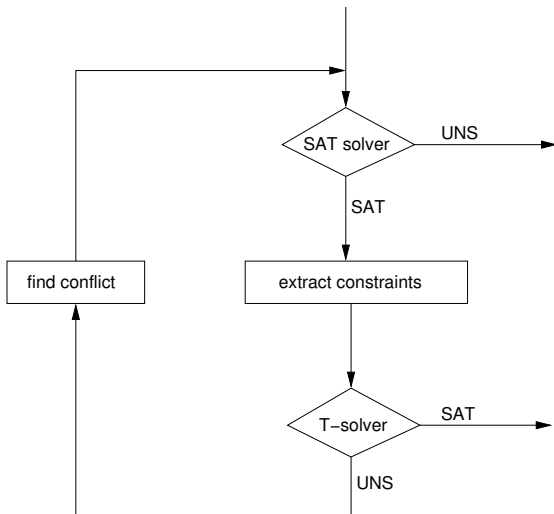
Run constraint problem through T-solver

- (1)  $pC = 0$
- (2)  $C \neq pC + 1$
- (3)  $C = pC$
- (4)  $C < 0$
- (5)  $C < 0$

Constraints 1, 3 and 5 contradict each other. Add explanation to SAT problem.  
Goto step 2.

# Satisfiability Modulo Theories

Informally



# Inconsistent subsets

A minimal inconsistent set of constraints is one

- ▶ which is inconsistent
- ▶ where removing any one constraint makes it consistent

## Examples

$$(1) \quad x > 0$$

$$(2) \quad y - 2x = 1$$

$$(3) \quad y \leq 1$$

$$(1) \quad x > 0$$

$$(2) \quad y - 2x = 1$$

$$(3) \quad y \leq 1$$

$$(4) \quad x < 0$$

Minimal set

Minimal subsets:  $\{1, 2, 3\}$ ,  $\{1, 4\}$

# Finding inconsistent subsets of constraint problems

- ▶ There are typically many (minimal) inconsistent subsets
- ▶ We want the one which prunes the most spurious SAT models
- ▶ Idea: Find one which is as closely related to the property being verified as possible

Here we find a minimal reason using deletion filtering, i.e. by removing one constraint at a time from the inconsistent set.

- ▶ Requires solving of one constraint problem per constraint.  
**Expensive.**
- ▶ Faster algorithms do not necessarily find the wanted subset.

# Gaussian elimination

$$(1) \quad x > 0$$

$$(2) \quad y - 2x = 1$$

$$(3) \quad y \leq 1$$

Simple idea:

- ▶ eliminate variables
- ▶ if we have two constraints  $c_1 \leq x \leq c_2$  where  $c_1 > c_2$ , we have a conflict.

# Gaussian elimination

$$(1) \quad x > 0$$

$$(2) \quad y = 2x + 1$$

$$(3) \quad y \leq 1$$

Simple idea:

- ▶ eliminate variables
- ▶ if we have two constraints  $c_1 \leq x \leq c_2$  where  $c_1 > c_2$ , we have a conflict.

# Gaussian elimination

$$\begin{array}{l} (1) \quad x > 0 \\ (2) \quad y = 2x + 1 \\ (3, 2) \quad 2x + 1 \leq 1 \end{array}$$

Simple idea:

- ▶ eliminate variables
- ▶ if we have two constraints  $c_1 \leq x \leq c_2$  where  $c_1 > c_2$ , we have a conflict.

# Gaussian elimination

$$\begin{array}{l} (1) \quad x > 0 \\ (2) \quad y = 2x + 1 \\ (3,2) \quad x \leq 0 \end{array}$$

Simple idea:

- ▶ eliminate variables
- ▶ if we have two constraints  $c_1 \leq x \leq c_2$  where  $c_1 > c_2$ , we have a conflict.

# Gaussian elimination

$$\begin{aligned}(1) \quad & x > 0 \\(2) \quad & y = 2x + 1 \\(3, 2) \quad & x \leq 0\end{aligned}$$

Simple idea:

- ▶ eliminate variables
- ▶ if we have two constraints  $c_1 \leq x \leq c_2$  where  $c_1 > c_2$ , we have a conflict.

# Encoding of Lustre

- ▶ Nested  $e = \mathbf{if\ } b \mathbf{\ then\ } e_1 \mathbf{\ else\ } e_2$  expressions are common.
- ▶ Only one of  $e_1, e_2$  are relevant, depending on the truth assignment of  $b$ .

We will use *guarded* constraints to encode this into the SAT problem.

- ▶ All constraints in the formula has a guard, i.e. on the form  $g \rightarrow c$ .
- ▶ A constraint is relevant only when its guard is true.

$$g_1 \rightarrow e = e_1$$

$$g_2 \rightarrow e = e_2$$

$$b \rightarrow g_1 \wedge \neg g_2$$

$$\neg b \rightarrow \neg g_1 \wedge g_2$$

$$g_1 \mapsto e = e_1$$

$$g_2 \mapsto e = e_2$$

# Encoding of Lustre

- ▶ Nested  $e = \mathbf{if\ } b \mathbf{\ then\ } e_1 \mathbf{\ else\ } e_2$  expressions are common.
- ▶ Only one of  $e_1, e_2$  are relevant, depending on the truth assignment of  $b$ .

We will use *guarded* constraints to encode this into the SAT problem.

- ▶ All constraints in the formula has a guard, i.e. on the form  $g \rightarrow c$ .
- ▶ A constraint is relevant only when its guard is true.

$$g_1 \rightarrow e = e_1$$

$$g_2 \rightarrow e = e_2$$

$$b \rightarrow g_1 \wedge \neg g_2$$

$$\neg b \rightarrow \neg g_1 \wedge g_2$$

$$g_1 \mapsto e = e_1$$

$$g_2 \mapsto e = e_2$$

# Encoding of Lustre

- ▶ Nested  $e = \mathbf{if\ } b \mathbf{\ then\ } e_1 \mathbf{\ else\ } e_2$  expressions are common.
- ▶ Only one of  $e_1, e_2$  are relevant, depending on the truth assignment of  $b$ .

We will use *guarded* constraints to encode this into the SAT problem.

- ▶ All constraints in the formula has a guard, i.e. on the form  $g \rightarrow c$ .
- ▶ A constraint is relevant only when its guard is true.

$$g_1 \rightarrow e = e_1$$

$$g_2 \rightarrow e = e_2$$

$$b \rightarrow g_1 \wedge \neg g_2$$

$$\neg b \rightarrow \neg g_1 \wedge g_2$$

$$g_1 \mapsto e = e_1$$

$$g_2 \mapsto e = e_2$$

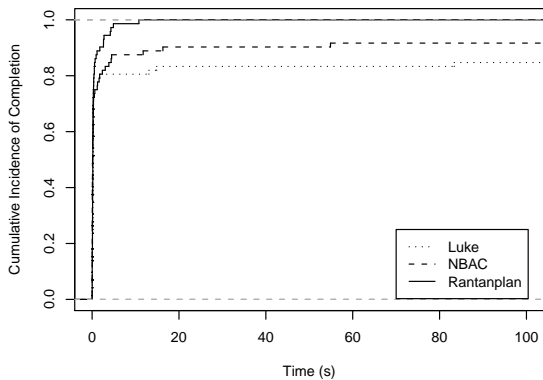
# Incremental SMT

An incremental SMT procedure can be constructed on top of an incremental SAT procedure

- ▶ add constraint  $c$   
returns its guard
- ▶ add clause  $l_1 \vee \dots \vee l_n$
- ▶ solve under assumptions  $l_1 \wedge \dots \wedge l_n$   
returns SAT or UNSAT

# Comparison w. NBAC/Luke

Valid properties



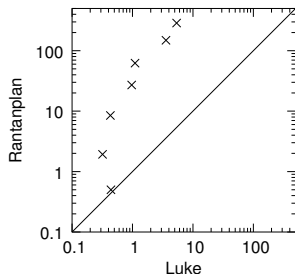
# Comparison w. NBAC/Luke

Valid properties

<b>Tool</b>	<b>Verified</b>	<b>Alone</b>
NBAC	96	18
Luke	98	1
Rantanplan	106	1

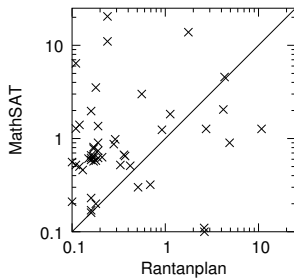
# Comparison w. Luke

## Invalid properties



- ▶ All examples have a single shortest cex
- ▶ Step cases dominate the execution time

# Comparison w. MathSAT



## Rantanplan vs MathSAT 3.2.1

- ▶ No static learning
- ▶ No early pruning
- ▶ Incremental procedure
- ▶ Domain-specific conflicts

# Incompleteness of induction

## Problem

```
node Natural() returns ( N : int );
```

```
let
```

```
    N = 0  $\rightarrow$  pre N + 1;
```

```
tel
```

```
node Property() returns ( OK : bool );
```

```
let
```

```
    OK = Natural()  $\neq$  -1;
```

```
tel
```

- ▶ Can not be verified using induction
- ▶ Unbounded sequence  $-n, -n + 1, \dots, -1$  leading to a state where the property is false
- ▶ Induction incomplete for unbounded integers

# Summary

- ▶ Incremental SMT
  - ▶ Domain-specific conflicts
  - ▶ Cheap detection of commonly occurring conflicts
- 
- ▶ Outlook
    - ▶ Larger depths (better ind. schema/isomorphy inference)
    - ▶ Completeness of branch and bound

Thank you