Combining SAT solving with Integer Programming for Inductive Verification of Lustre Programs

3rd December 2004

Outline

Outline

1. Introduction
   - The Lustre programming language
   - Temporal induction
   - Propositional logic
2. Verification
   - The decision procedure (SAT + Integer Programming)
   - Variants of the basic algorithm
3. Analysis
   - Test plan
   - Comparison with Luke

Outline

1. Introduction
   - The Lustre programming language
   - Temporal induction
   - Propositional logic
2. Verification
   - The decision procedure (SAT + Integer Programming)
   - Variants of the basic algorithm
3. Analysis
   - Test plan
   - Comparison with Luke

Lustre

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

node Prop( X : bool ) returns ( OK : bool );
let
    OK = Counter( X ) ≥ 0;
tel
```

Verification by induction

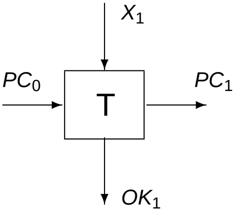
- Prove property valid in initial time point

- Assume property valid at time n, prove property valid at time $n+1$

- Induction incomplete for unbounded integers
- Lustre with unbounded integers Turing-complete

Verification by induction

- Prove property valid in initial time point

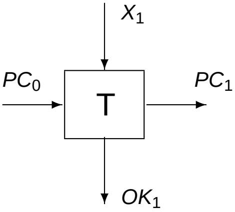


- Assume property valid at time n, prove property valid at time $n+1$

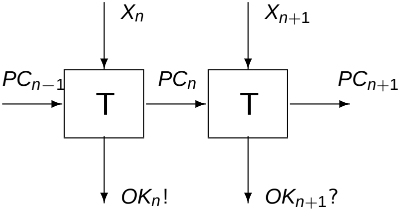
- Induction incomplete for unbounded integers
- Lustre with unbounded integers Turing-complete

Verification by induction

- Prove property valid in initial time point



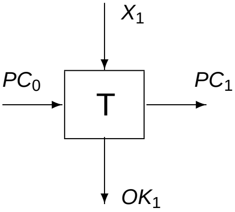
- Assume property valid at time n, prove property valid at time $n+1$



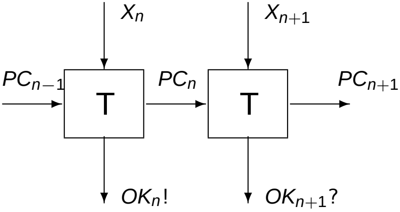
- Induction incomplete for unbounded integers
- Lustre with unbounded integers Turing-complete

Verification by induction

- Prove property valid in initial time point



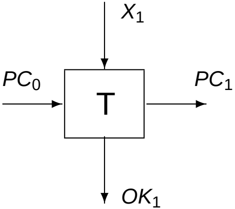
- Assume property valid at time n, prove property valid at time $n+1$



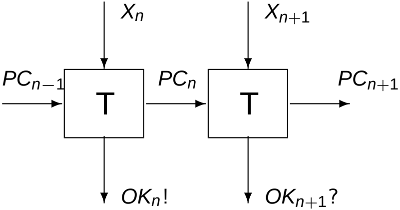
- Induction incomplete for unbounded integers
- Lustre with unbounded integers Turing-complete

Verification by induction

- Prove property valid in initial time point



- Assume property valid at time n, prove property valid at time $n+1$



- Induction incomplete for unbounded integers
- Lustre with unbounded integers Turing-complete

Propositional logic

Example

$$\{p, q\}$$

Short introduction

- A clause is a set of literals. At least one literal must be true.
- A formula is a set of clauses. All clauses must be true.

Propositional logic

Example

$$\{p, q\}$$
$$\{p, \neg q, r\}$$
$$\{\neg q, \neg r\}$$

Short introduction

- A clause is a set of literals. At least one literal must be true.
- A formula is a set of clauses. All clauses must be true.

SAT solving

Example

$$\{p, q\}$$
$$\{p, \neg q, r\}$$
$$\{\neg q, \neg r\}$$

Search for a satisfying variable assignment

- Choose a variable, and assign at value to it
- Infer consequences
- Repeat until all variables assigned, or a *conflict* found

SAT solving

Example

$$\{\mathbf{p}, q\}$$
$$\{\mathbf{p}, \neg q, r\}$$
$$\{\neg q, \neg r\}$$

$$p = \perp$$

Search for a satisfying variable assignment

- Choose a variable, and assign at value to it
- Infer consequences
- Repeat until all variables assigned, or a *conflict* found

SAT solving

Example

$$\{\mathbf{p}, q\}$$
$$\{\mathbf{p}, \neg q, r\}$$
$$\{\neg q, \neg r\}$$

$$p \quad = \quad \perp$$

Search for a satisfying variable assignment

- Choose a variable, and assign at value to it
- Infer consequences
- Repeat until all variables assigned, or a *conflict* found

SAT solving

Example

$$\{\mathbf{p}, \mathbf{q}\}$$
$$\{\mathbf{p}, \neg\mathbf{q}, r\}$$
$$\{\neg\mathbf{q}, \neg r\}$$

$$\begin{aligned} p &= \bot \\ q &= \top \end{aligned}$$

Search for a satisfying variable assignment

- Choose a variable, and assign at value to it
- Infer consequences
- Repeat until all variables assigned, or a *conflict* found

SAT solving

Example

$$\{\mathbf{p}, \mathbf{q}\}$$
$$\{\mathbf{p}, \neg\mathbf{q}, \mathbf{r}\}$$
$$\{\neg\mathbf{q}, \neg\mathbf{r}\}$$

$$\begin{aligned} p &= \bot \\ q &= \top \\ r &= \top \textbf{ and } \bot?? \end{aligned}$$

Search for a satisfying variable assignment

- Choose a variable, and assign at value to it
- Infer consequences
- Repeat until all variables assigned, or a *conflict* found

SAT solving

Example

$$\{p, q\}$$
$$\{p, \neg q, r\}$$
$$\{\neg q, \neg r\}$$

$$\begin{aligned} p &= \bot \\ q &= \top \\ r &= \top \textbf{ and } \bot?? \end{aligned}$$

Search for a satisfying variable assignment

- Analyze reason for conflict
- Add *conflict clause*
- Backtrack and continue

SAT solving

Example

$$\{p, q\}$$
$$\{p, \neg q, r\}$$
$$\{\neg q, \neg r\}$$

$$p = \bot$$
$$q = \top$$
$$r = \top \textbf{ and } \bot??$$

Search for a satisfying variable assignment

- Analyze reason for conflict
- Add *conflict clause*
- Backtrack and continue

SAT solving

Example

$$\{p, q\}$$
$$\{p, \neg q, r\}$$
$$\{\neg q, \neg r\}$$
$$\{p\}$$

$$\begin{aligned} p &= \bot \\ q &= \top \\ r &= \top \textbf{ and } \bot?? \end{aligned}$$

Search for a satisfying variable assignment

- Analyze reason for conflict
- Add *conflict clause*
- Backtrack and continue

SAT solving

Example

$$\{p, q\}$$
$$\{p, \neg q, r\}$$
$$\{\neg q, \neg r\}$$
$$\{p\}$$

Search for a satisfying variable assignment

- Analyze reason for conflict
- Add *conflict clause*
- Backtrack and continue

SAT solving

Example

$$\{\mathbf{p}, q\}$$
$$\{\mathbf{p}, \neg q, r\}$$
$$\{\neg q, \neg r\}$$
$$\{\mathbf{p}\}$$

$$p = \top$$

Search for a satisfying variable assignment

- Analyze reason for conflict
- Add *conflict clause*
- Backtrack and continue

SAT solving

Example

$$\{\mathbf{p}, \mathbf{q}\}$$
$$\{\mathbf{p}, \neg\mathbf{q}, r\}$$
$$\{\neg\mathbf{q}, \neg r\}$$
$$\{\mathbf{p}\}$$

$$\begin{aligned} p &= \top \\ q &= \bot \end{aligned}$$

Search for a satisfying variable assignment

- Analyze reason for conflict
- Add *conflict clause*
- Backtrack and continue

SAT solving

Example

$$\{p, q\}$$
$$\{p, \neg q, r\}$$
$$\{\neg q, \neg r\}$$
$$\{p\}$$

$$\begin{aligned} p &= \top \\ q &= \bot \\ r &= \bot \end{aligned}$$

Search for a satisfying variable assignment

- Analyze reason for conflict
- Add *conflict clause*
- Backtrack and continue

A small example

The formula in CNF

A simple counter

node Counter() **returns** (OK : bool);
    var C : int;
let
    C = 0 $\rightarrow$ **pre** C + 1;
    OK = C $\geq$ 0;
tel

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

A small example

The formula in CNF

$$\{\ C_1 \leq 0\ \}$$
$$\{\ C_1 \geq 0\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ C_1 \geq 0\ \}$$
$$\{\ \mathrm{OK}_1,\ C_1 \leq -1\ \}$$

A simple counter

node Counter() **returns** (OK : bool);
  var C : int;
let
  C = 0 $\rightarrow$ **pre** C + 1;
  OK = C $\geq$ 0;
tel

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

A small example

The formula in CNF

$$\{\ C_1 \leq 0\ \}$$
$$\{\ C_1 \geq 0\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ C_1 \geq 0\ \}$$
$$\{\ \mathrm{OK}_1,\ C_1 \leq -1\ \}$$
$$\{\ \neg\mathrm{OK}_1 \}$$

A simple counter

node Counter() **returns** (OK : bool);
  var C : int;
let
  C = 0 $\rightarrow$ **pre** C + 1;
  OK = C $\geq$ 0;
tel

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

A small example

The formula in CNF

$$\{\ C_1 \leq 0\ \}$$
$$\{\ C_1 \geq 0\ \}$$
$$\{\ \neg \mathrm{OK}_1,\ C_1 \geq 0\ \}$$
$$\{\ \mathrm{OK}_1,\ C_1 \leq -1\ \}$$
$$\{\ \neg \mathrm{OK}_1 \}$$

A simple counter

node Counter() **returns** (OK : bool);
  var C : int;
let
  C = 0 $\rightarrow$ **pre** C + 1;
  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

$$\{\ C_1 \leq 0\ \}$$
$$\{\ C_1 \geq 0\ \}$$
$$\{\ \neg \mathrm{OK}_1,\ C_1 \geq 0\ \}$$
$$\{\ \mathrm{OK}_1,\ C_1 \leq -1\ \}$$
$$\{\ \neg \mathrm{OK}_1 \}$$

Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

And replace all constraints with their in-place variable.

The basic algorithm

The formula in CNF

$$\begin{array}{l} \{\ C_1 \leq 0\ \} \\ \{\ C_1 \geq 0\ \} \\ \{\ \neg \mathrm{OK}_1,\ C_1 \geq 0\ \} \\ \{\ \mathrm{OK}_1,\ C_1 \leq -1\ \} \\ \{\ \neg \mathrm{OK}_1 \} \end{array}$$

Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$$\begin{array}{l} p_1 \mapsto C_1 \leq 0 \\ p_2 \mapsto C_1 \geq 0 \\ p_3 \mapsto C_1 \geq 0 \\ p_4 \mapsto C_1 \leq -1 \end{array}$$

And replace all constraints with their in-place variable.

The basic algorithm

The formula in CNF

$$\{\ C_1 \leq 0\ \}$$
$$\{\ C_1 \geq 0\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ C_1 \geq 0\ \}$$
$$\{\ \mathrm{OK}_1,\ C_1 \leq -1\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$

Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$$\begin{aligned} p_1 &\mapsto C_1 \leq 0 \\ p_2 &\mapsto C_1 \geq 0 \\ p_3 &\mapsto C_1 \geq 0 \\ p_4 &\mapsto C_1 \leq -1 \end{aligned}$$

And replace all constraints with their in-place variable.

The basic algorithm

The formula in CNF

$$\{\ p_1\ \}$$
$$\{\ p_2\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ p_3\ \}$$
$$\{\ \mathrm{OK}_1,\ p_4\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$

$$p_1 \mapsto C_1 \leq 1$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 1: Create *in-place* variables

Create a fresh propositional variable for each constraint

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

And replace all constraints with their in-place variable.

The basic algorithm

The formula in CNF

$$\{\ p_1\ \}$$
$$\{\ p_2\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ p_3\ \}$$
$$\{\ \mathrm{OK}_1,\ p_4\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 2: Run through SAT solver

A SAT model is returned

$$\begin{aligned} p_1 &= \top \\ p_2 &= \top \\ p_3 &= \bot \\ p_4 &= \top \\ \mathrm{OK}_1 &= \bot \end{aligned}$$

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

The basic algorithm

The formula in CNF

$$\begin{array}{l} \{\ p_1\ \} \\ \{\ p_2\ \} \\ \{\ \neg\mathrm{OK}_1,\ p_3\ \} \\ \{\ \mathrm{OK}_1,\ p_4\ \} \\ \{\ \neg\mathrm{OK}_1\} \end{array}$$

$$\begin{array}{l} p_1 \mapsto C_1 \leq 0 \\ p_2 \mapsto C_1 \geq 0 \\ p_3 \mapsto C_1 \geq 0 \\ p_4 \mapsto C_1 \leq -1 \end{array}$$

Step 2: Run through SAT solver

A SAT model is returned

$$\begin{aligned} p_1 &= \top \\ p_2 &= \top \\ p_3 &= \bot \\ p_4 &= \top \\ \mathrm{OK}_1 &= \bot \end{aligned}$$

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

The basic algorithm

The formula in CNF

$$\{\ \mathbf{p_1}\ \}$$
$$\{\ \mathbf{p_2}\ \}$$
$$\{\ \neg \mathrm{OK}_\mathbf{1},\ \mathbf{p_3}\ \}$$
$$\{\ \mathrm{OK}_\mathbf{1},\ \mathbf{p_4}\ \}$$
$$\{\ \neg \mathrm{OK}_\mathbf{1}\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 2: Run through SAT solver

A SAT model is returned

$$\begin{aligned} p_1 &= \top \\ p_2 &= \top \\ p_3 &= \bot \\ p_4 &= \top \\ \mathrm{OK}_1 &= \bot \end{aligned}$$

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

The basic algorithm

The formula in CNF

$$\{\ \mathbf{p_1}\ \}$$
$$\{\ \mathbf{p_2}\ \}$$
$$\{\ \neg\mathrm{OK}_{\mathbf{1}},\ \mathbf{p_3}\ \}$$
$$\{\ \mathrm{OK}_{\mathbf{1}},\ \mathbf{p_4}\ \}$$
$$\{\ \neg\mathrm{OK}_{\mathbf{1}}\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 2: Run through SAT solver

A SAT model is returned

$$\begin{aligned} p_1 &= \top \\ p_2 &= \top \\ p_3 &= \bot \\ p_4 &= \top \\ \mathrm{OK}_1 &= \bot \end{aligned}$$

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

The basic algorithm

The formula in CNF

$$\{\ \mathbf{p_1}\ \}$$
$$\{\ \mathbf{p_2}\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ \mathbf{p_3}\ \}$$
$$\{\ \mathrm{OK}_1,\ \mathbf{p_4}\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 3: Solve constraint problem

Run constraint problem trough ILP solver

$$\begin{array}{ll} (1) & C_1 \leq 0 \\ (2) & C_1 \geq 0 \\ (3) & C_1 < 0 \\ (4) & C_1 \leq -1 \end{array}$$

Constraint 2 and 4 contradict each other. Add explanation to SAT problem. Goto step 2.

The basic algorithm

The formula in CNF

$\{\ \mathbf{p_1}\ \}$
$\{\ \mathbf{p_2}\ \}$
$\{\ \neg\mathrm{OK}_1,\ \mathbf{p_3}\ \}$
$\{\ \mathrm{OK}_1,\ \mathbf{p_4}\ \}$
$\{\ \neg\mathrm{OK}_1\}$

$p_1 \mapsto C_1 \leq 0$
$p_2 \mapsto C_1 \geq 0$
$p_3 \mapsto C_1 \geq 0$
$p_4 \mapsto C_1 \leq -1$

Step 3: Solve constraint problem

Run constraint problem trough ILP solver

$$
\begin{array}{ll}
(1) & C_1 \leq 0 \\
(2) & C_1 \geq 0 \\
(3) & C_1 < 0 \\
(4) & C_1 \leq -1
\end{array}
$$

Constraint 2 and 4 contradict each other. Add explanation to SAT problem. Goto step 2.

The basic algorithm

The formula in CNF

$$\{\ p_1\ \}$$
$$\{\ p_2\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ p_3\ \}$$
$$\{\ \mathrm{OK}_1,\ p_4\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$
$$\{\neg p_2,\ \neg p_4\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 < 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 3: Solve constraint problem

Run constraint problem trough ILP solver

$$\begin{aligned} (1) &\quad C_1 \leq 0 \\ (2) &\quad C_1 \geq 0 \\ (3) &\quad C_1 < 0 \\ (4) &\quad C_1 \leq -1 \end{aligned}$$

Constraint 2 and 4 contradict each other. Add explanation to SAT problem. Goto step 2.

The basic algorithm

The formula in CNF

$$\{\ p_1\ \}$$
$$\{\ p_2\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ p_3\ \}$$
$$\{\ \mathrm{OK}_1,\ p_4\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$
$$\{\neg p_2,\ \neg p_4\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 2: Run through SAT solver

The formula is unsatisfiable

$$\Rightarrow$$

The original formula is unsatisfiable.

The property is valid in first time point

The basic algorithm

The formula in CNF

$$\{\ p_1\ \}$$
$$\{\ p_2\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ p_3\ \}$$
$$\{\ \mathrm{OK}_1,\ p_4\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$
$$\{\neg p_2,\ \neg p_4\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 2: Run through SAT solver

The formula is unsatisfiable

$\Rightarrow$

The original formula is unsatisfiable.

The property is valid in first time point

The basic algorithm

The formula in CNF

$$
\begin{array}{l}
\{\ p_1\ \} \\
\{\ p_2\ \} \\
\{\ \neg \mathrm{OK}_1,\ p_3\ \} \\
\{\ \mathrm{OK}_1,\ p_4\ \} \\
\{\ \neg \mathrm{OK}_1 \} \\
\{ \neg p_2,\ \neg p_4 \}
\end{array}
$$

$$
\begin{array}{l}
p_1 \mapsto C_1 \leq 0 \\
p_2 \mapsto C_1 \geq 0 \\
p_3 \mapsto C_1 \geq 0 \\
p_4 \mapsto C_1 \leq -1
\end{array}
$$

Step 2: Run through SAT solver

The formula is unsatisfiable

$\Rightarrow$

The original formula is unsatisfiable.

The property is valid in first time point

The basic algorithm

The formula in CNF

$$\{\ p_1\ \}$$
$$\{\ p_2\ \}$$
$$\{\ \neg\mathrm{OK}_1,\ p_3\ \}$$
$$\{\ \mathrm{OK}_1,\ p_4\ \}$$
$$\{\ \neg\mathrm{OK}_1\}$$
$$\{\neg p_2,\ \neg p_4\}$$

$$p_1 \mapsto C_1 \leq 0$$
$$p_2 \mapsto C_1 \geq 0$$
$$p_3 \mapsto C_1 \geq 0$$
$$p_4 \mapsto C_1 \leq -1$$

Step 2: Run through SAT solver

The formula is unsatisfiable

$$\Rightarrow$$

The original formula is unsatisfiable.

The property is valid in first time point

The algorithm

φ is a propositional + constraints formula

loop
    $\mathbb{I}_p \leftarrow \text{Psat}(\varphi)$
    if $\mathbb{I}_p = \emptyset$ **then**
        return *unsatisfiable*
    else
        $C \leftarrow \text{generate}(\varphi, \mathbb{I}_p)$
        if $\text{Csat}(C)$ **then**
            return *satisfiable*
        else
            $\varphi \leftarrow \varphi \cup \text{explain}(C)$
        end if
    end if
end loop

Other ideas

- Check *partial* SAT models
  Everytime the SAT solver assigns an in-place variable, check the constraint problem generated by the set of assigned in-place variables.
- Several methods of creating explanations
  Several algorithms exist. Finding multiple explanations.
- Preprocessing
  Find contradictions in the set of constraints before the decision procedure starts.
- Faster (incomplete) integer programming procedure
  Use a cheap procedure that can find the most commonly occuring contradictions in constraint problems.

Other ideas

- Check *partial* SAT models
  Everytime the SAT solver assigns an in-place variable, check the constraint problem generated by the set of assigned in-place variables.
- Several methods of creating explanations
  Several algorithms exist. Finding multiple explanations.
- Preprocessing
  Find contradictions in the set of constraints before the decision procedure starts.
- Faster (incomplete) integer programming procedure
  Use a cheap procedure that can find the most commonly occuring contradictions in constraint problems.

Other ideas

- Check *partial* SAT models
  Everytime the SAT solver assigns an in-place variable, check the constraint problem generated by the set of assigned in-place variables.
- Several methods of creating explanations
  Several algorithms exist. Finding multiple explanations.
- Preprocessing
  Find contradictions in the set of constraints before the decision procedure starts.
- Faster (incomplete) integer programming procedure
  Use a cheap procedure that can find the most commonly occuring contradictions in constraint problems.

Other ideas

- Check *partial* SAT models
  Everytime the SAT solver assigns an in-place variable, check the constraint problem generated by the set of assigned in-place variables.
- Several methods of creating explanations
  Several algorithms exist. Finding multiple explanations.
- Preprocessing
  Find contradictions in the set of constraints before the decision procedure starts.
- Faster (incomplete) integer programming procedure
  Use a cheap procedure that can find the most commonly occuring contradictions in constraint problems.

Rantanplan

- Implements all ideas outlined here
- Based on Luke
- SAT solver changed to MiniSat
- Integer programming package GLPK

Test plan

Aim

- What combinations of ideas work well?
- How do these ideas compare to Luke and NBAC?

- Find “good” combinations of ideas
- Compare these to Luke & NBAC

Test plan

Aim

- What combinations of ideas work well?
- How do these ideas compare to Luke and NBAC?

- Find “good” combinations of ideas
- Compare these to Luke & NBAC

Test plan

Aim

- What combinations of ideas work well?
- How do these ideas compare to Luke and NBAC?

- Find “good” combinations of ideas
- Compare these to Luke & NBAC

Test suite

Every test should be verifyable by every tool in the tests.

- The test suite consists of 137 tests.
- Some of these are invalid properties. Can not be verified in NBAC.
- Some have too weak properties. Can not be verified in Rantanplan.
- Some used unbounded integers. Can not be verified in Luke.
- Some uses modulo. Can not be verified in NBAC.
- Some generates constraint problems where branch-and-bound does not terminate. Can not be verified in Rantanplan.

We are left with 72 tests.

Test suite

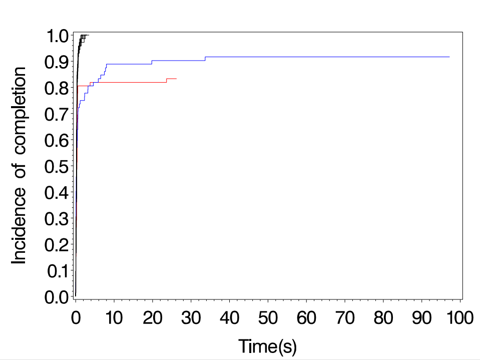
Every test should be verifyable by every tool in the tests.

- The test suite consists of 137 tests.
- Some of these are invalid properties. Can not be verified in NBAC.
- Some have too weak properties. Can not be verified in Rantanplan.
- Some used unbounded integers. Can not be verified in Luke.
- Some uses modulo. Can not be verified in NBAC.
- Some generates constraint problems where branch-and-bound does not terminate. Can not be verified in Rantanplan.

We are left with 72 tests.

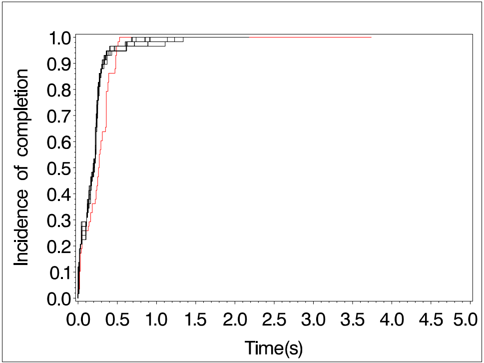
Comparisons

Tests of the 11 best variants against Luke and NBAC



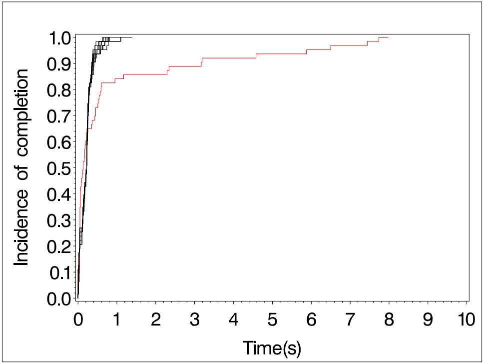
Comparison with Luke

Tests with execution time $>$ 10s in Luke removed (58 remaining)



Comparison with NBAC

Tests with execution time $>$ 10s in NBAC removed (63 remaining)



Summary

- Rantanplan competitive on the test suite used here
- The branch-and-bound algorithm is incomplete
- For longer induction depth (e.g. invalid properties w. long counter-examples), Luke outperforms Rantanplan

- Outlook
  - Complete integer programming procedure
  - Improvements for larger induction depths
  - Invariant strengthening