Satisfiability Modulo Theories: ABsolver

Michael Tautschnig

Joint work with:
Andreas Bauer
Martin Leucker
Christian Schallhart

Outline

1. Introduction

Outline

1. Introduction

2. ABsolver

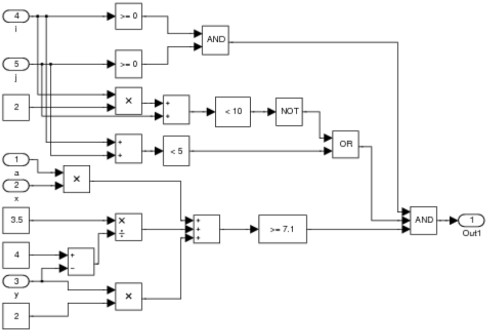
Outline

1. Introduction

2. ABsolver

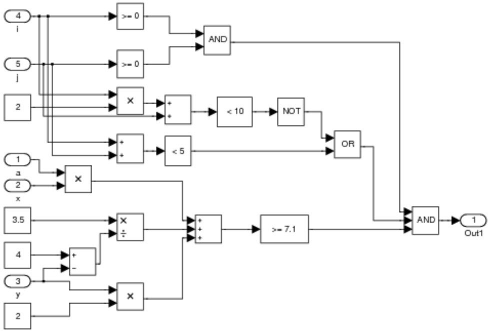
3. Summary

Example (1)



Example (1)

$$
\begin{aligned}
& ((i \geq 0) \wedge (j \geq 0)) \\
\wedge \quad & (\neg(2i + j < 10) \vee (i + j < 5)) \\
\wedge \quad & \left(a \cdot x + \frac{3.5}{4-y} + 2y \geq 7.1\right)
\end{aligned}
$$



Example (1)

$$((i \geq 0) \land (j \geq 0))$$
$$\land \quad (\neg(2i + j < 10) \lor (i + j < 5))$$
$$\land \quad \left(a \cdot x + \frac{3.5}{4-y} + 2y \geq 7.1\right)$$

```
p cnf 4 3
1 0
-2 3 0
4 0

c def int 1 i >= 0
c def int 1 j >= 0
c def int 2 2*i + j < 10
c def int 3 i + j < 5
c def real 4 a * x + 3.5 / ( 4 - y ) + 2 * y >= 7.1
```

Example (1)

$$((i \geq 0) \wedge (j \geq 0))$$
$$\wedge \quad (\neg(2i + j < 10) \vee (i + j < 5))$$
$$\wedge \quad \left(a \cdot x + \frac{3.5}{4-y} + 2y \geq 7.1\right)$$

```
p cnf 4 3
1 0
-2 3 0
4 0

c def int 1 i >= 0
c def int 1 j >= 0
c def int 2 2*i + j < 10
c def int 3 i + j < 5
c def real 4 a * x + 3.5 / ( 4 - y ) + 2 * y >= 7.1
```

Example (1)

$$((i \geq 0) \wedge (j \geq 0))$$
$$\wedge \quad (\neg(2i + j < 10) \vee (i + j < 5))$$
$$\wedge \quad \left(a \cdot x + \frac{3.5}{4-y} + 2y \geq 7.1\right)$$

```
p cnf 4 3
1 0
-2 3 0
4 0

c def int 1 i >= 0
c def int 1 j >= 0
c def int 2 2*i + j < 10
c def int 3 i + j < 5
c def real 4 a * x + 3.5 / ( 4 - y ) + 2 * y >= 7.1
```

Example (1)

```
p cnf 4 3
1 0
-2 3 0
4 0

c def int 1 i >= 0
c def int 1 j >= 0
c def int 2 2*i + j < 10
c def int 3 i + j < 5
c def real 4 a * x + 3.5 / ( 4 - y ) + 2 * y >= 7.1
```

$$
\begin{aligned}
& ((i \geq 0) \wedge (j \geq 0)) \\
\wedge \quad & (\neg(2i + j < 10) \vee (i + j < 5)) \\
\wedge \quad & \left(a \cdot x + \frac{3.5}{4-y} + 2y \geq 7.1\right)
\end{aligned}
$$

Example (2)

```
void fn( int * a, int width, int size )
{
  int i = 0, j = 0;

  for( i = 0; i < size / width; ++i )
    for( j = 0; j <= width; ++j )
      assert( i * width + j < size );
}

int main( int argc, char * argv[] )
{
  int a[ 10 ];

  if( argc > 5 )
    fn( a, 1, 10 );
  else
    fn( a, 2, 10 );

  return 0;
}
```

Example (2)

$$
\begin{aligned}
&(i \geq 0) \wedge (j \geq 0) \\
\wedge \quad &\Big(\big(\neg(i + j < 10) \wedge (i < 10) \wedge (j \leq 1) \big) \\
&\vee \big(\neg(2i + j < 10) \wedge (i < 5) \wedge (j \leq 2) \big) \Big)
\end{aligned}
$$

```
void fn( int * a, int width, int size )
{
  int i = 0, j = 0;

  for( i = 0; i < size / width; ++i )
    for( j = 0; j <= width; ++j )
      assert( i * width + j < size );
}

int main( int argc, char * argv[] )
{
  int a[ 10 ];

  if( argc > 5 )
    fn( a, 1, 10 );
  else
    fn( a, 2, 10 );

  return 0;
}
```

Satisfiability Modulo Theories

(informal)

- Quantifier-free Boolean formulas using the operators $\vee, \wedge, \neg$

Satisfiability Modulo Theories

(informal)

- Quantifier-free Boolean formulas using the operators $\vee, \wedge, \neg$
- Substitution of Boolean variables by theory constraints

Satisfiability Modulo Theories

(informal)

- Quantifier-free Boolean formulas using the operators $\vee, \wedge, \neg$
- Substitution of Boolean variables by theory constraints
- E. g., arithmetic operators $(+, -, \cdot, \div)$ and comparison using $<, > \leq, \geq, =$ on real or integer variables

Satisfiability Modulo Theories

(informal)

- Quantifier-free Boolean formulas using the operators $\vee, \wedge, \neg$
- Substitution of Boolean variables by theory constraints
- E. g., arithmetic operators $(+, -, \cdot, \div)$ and comparison using $<, > \leq, \geq, =$ on real or integer variables

Satisfiability problem: Is there an assignment to the arithmetic and Boolean variables, such that the formula is satisfied?

Application domain

- Decision procedures for verification
- Test case generation
- Model-based diagnosis
- Puzzles

Possible approaches

- *(Specialized) theorem provers* (CVC 3)

Possible approaches

- *(Specialized) theorem provers* (CVC 3)
- Online approach (MathSAT, yices)
  - \+ Highly integrated
  - \- Hardly extensible – cannot deal with non-linear arithmetic
  - \- Cannot easily benefit from (new) expert knowledge

Possible approaches

- *(Specialized) theorem provers* (CVC 3)
- Online approach (MathSAT, yices)
  - \+ Highly integrated
  - \- Hardly extensible – cannot deal with non-linear arithmetic
  - \- Cannot easily benefit from (new) expert knowledge
- Offline approach (Ario, ABsolver)

Offline approach

```
1:  // Abstract solution via SAT-solver.
2:  ν := find_SAT_solution(𝒞)
3:  if ν = ∅ then
4:      return ff
5:  end if
6:  // Concretisation via constraint solver.
7:  τ := find_concrete_solution(constr(φ, ν))
8:  if τ = ∅ then
9:      // Refinement of the Boolean abstraction.
10:     𝒞 := 𝒞 ∪ ¬ν
11:     goto 2
12: end if
13: return tt
```

ABsolver: Design principles

- Extensibility

ABsolver: Design principles

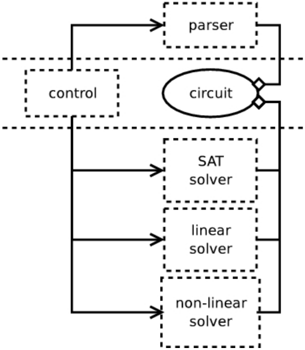
- Extensibility
- Utilization of existing solvers

ABsolver: Design principles

- Extensibility
- Utilization of existing solvers
- Ability to tackle hard problems

ABsolver: Design principles

- Extensibility
- Utilization of existing solvers
- Ability to tackle hard problems



ABsolver: Algorithm

ABsolver: Algorithm

- Obtain a solution to the Boolean abstraction

ABsolver: Algorithm

- Obtain a solution to the Boolean abstraction
- Solve the arithmetic parts while considering the Boolean results

ABsolver: Algorithm

- Obtain a solution to the Boolean abstraction
- Solve the arithmetic parts while considering the Boolean results
- If unsatisfiable, compute conflicts

ABsolver: Algorithm

- Obtain a solution to the Boolean abstraction
- Solve the arithmetic parts while considering the Boolean results
- If unsatisfiable, compute conflicts
- Loop, until the desired number of solutions has been obtained

ABsolver: Algorithm

- Obtain a solution to the Boolean abstraction
- Solve the arithmetic parts while considering the Boolean results
- If unsatisfiable, compute conflicts
- Loop, until the desired number of solutions has been obtained

- Very simple, but still successful
- Allows for the computation of any number of solutions (useful for test case generation)
- May require the computation of all Boolean solutions

Summary

Summary

- Definition arithmetic/Boolean problem

Summary

- Definition arithmetic/Boolean problem
- Application domains

Summary

- Definition arithmetic/Boolean problem
- Application domains
- The approach using ABsolver

Summary

- Definition arithmetic/Boolean problem
- Application domains
- The approach using ABsolver
- Experimental results