

Explicit vs. symbolic algorithms for solving ALFP constraints

Piotr Filipiuk, Hanne Riis Nielson and Flemming Nielson

DTU Informatics

September 17. 2010

Outline

- 1 Introduction
- 2 Alternation-free Least Fixed Point Logic
- 3 Algorithms and data structures
- 4 Experiments
- 5 Conclusion

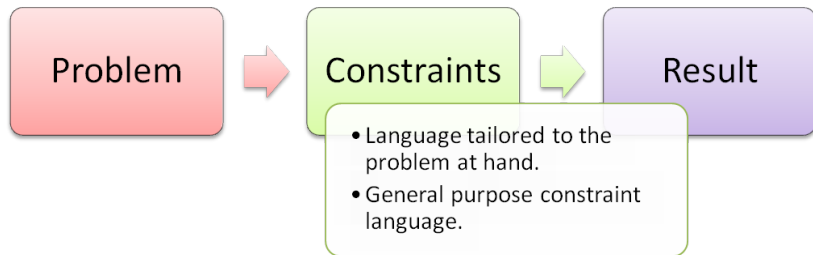
Outline

- 1 Introduction
- 2 Alternation-free Least Fixed Point Logic
- 3 Algorithms and data structures
- 4 Experiments
- 5 Conclusion

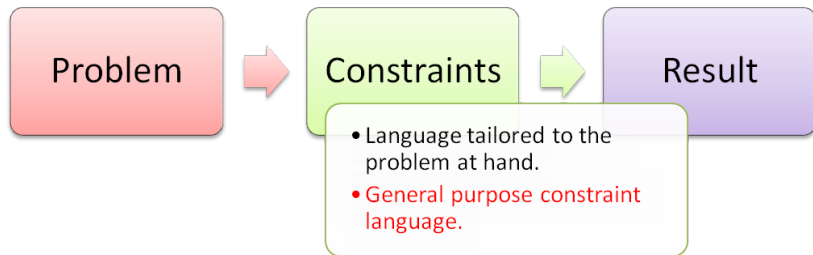
Static Analysis



Static Analysis



Static Analysis



Static Analysis

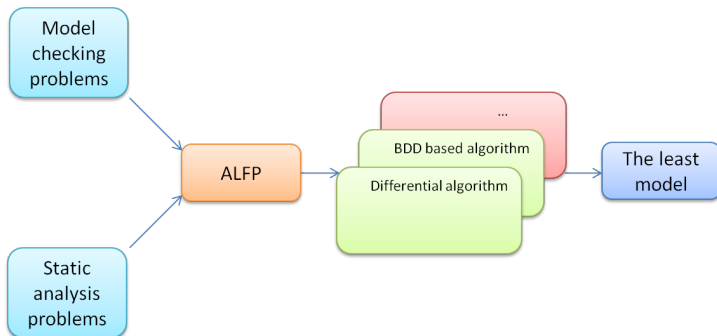


Motivation

Different analysis problems may give rise to ALFP clauses with different characteristics. There are also different approaches to solving ALFP clauses and some of these are better suited for certain kinds of clauses than others.

Motivation

Different analysis problems may give rise to ALFP clauses with different characteristics. There are also different approaches to solving ALFP clauses and some of these are better suited for certain kinds of clauses than others.



Motivation

Different analysis problems may give rise to ALFP clauses with different characteristics. There are also different approaches to solving ALFP clauses and some of these are better suited for certain kinds of clauses than others.

	Differential algorithm	Bdd based algorithm	...
clauses characteristics	?	?	...
relations size	?	?	...
universe size	?	?	...
...

Outline

- 1 Introduction
- 2 Alternation-free Least Fixed Point Logic**
- 3 Algorithms and data structures
- 4 Experiments
- 5 Conclusion

The syntax of ALFP

Definition

Given a fixed countable set \mathcal{X} of variables, a non-empty and finite universe \mathcal{U} and a finite alphabet \mathcal{R} of predicate symbols, we define the set of ALFP formulae (or clause sequences), cls , together with clauses, cl , and preconditions, pre , by the grammar:

$$\begin{aligned}
 pre & ::= R(v_1, \dots, v_k) \mid \neg R(v_1, \dots, v_k) \mid pre_1 \wedge pre_2 \\
 & \quad \mid pre_1 \vee pre_2 \mid \exists x : pre \mid \forall x : pre \\
 cl & ::= R(v_1, \dots, v_k) \mid \mathbf{1} \mid cl_1 \wedge cl_2 \mid pre \Rightarrow cl \mid \forall x : cl \\
 cls & ::= cl_1, \dots, cl_s
 \end{aligned}$$

Here $v_i \in (\mathcal{X} \cup \mathcal{U})$, $x \in \mathcal{X}$, $R \in \mathcal{R}$, $k \geq 0$ and $s \geq 1$.

Semantics of ALFP

Definition

Given a non empty and finite universe \mathcal{U} of atomic values together with interpretations ϱ and σ of relations and variables respectively, we define the satisfaction relations for pre-conditions and clauses as follows:

$$(\varrho, \sigma) \models pre \text{ and } (\varrho, \sigma) \models cl$$

Semantics of ALFP pre-conditions

$(\varrho, \sigma) \models R(v_1, \dots, v_n)$	<u>iff</u>	$(\sigma(v_1), \dots, \sigma(v_n)) \in \varrho(R)$
$(\varrho, \sigma) \models \neg R(v_1, \dots, v_n)$	<u>iff</u>	$(\sigma(v_1), \dots, \sigma(v_n)) \notin \varrho(R)$
$(\varrho, \sigma) \models pre_1 \vee pre_2$	<u>iff</u>	$(\varrho, \sigma) \models pre_1$ or $(\varrho, \sigma) \models pre_2$
$(\varrho, \sigma) \models pre_1 \wedge pre_2$	<u>iff</u>	$(\varrho, \sigma) \models pre_1$ and $(\varrho, \sigma) \models pre_2$
$(\varrho, \sigma) \models \forall x : pre$	<u>iff</u>	$(\varrho, \sigma[x \mapsto a]) \models pre$ for all $a \in \mathcal{U}$
$(\varrho, \sigma) \models \exists x : pre$	<u>iff</u>	$(\varrho, \sigma[x \mapsto a]) \models pre$ for some $a \in \mathcal{U}$

Semantics of ALFP clauses

$(\varrho, \sigma) \models R(v_1, \dots, v_n)$	<u>iff</u>	$(\sigma(v_1), \dots, \sigma(v_n)) \in \varrho(R)$
$(\varrho, \sigma) \models true$	<u>iff</u>	true
$(\varrho, \sigma) \models cl_1 \wedge cl_2$	<u>iff</u>	$(\varrho, \sigma) \models cl_1$ and $(\varrho, \sigma) \models cl_2$
$(\varrho, \sigma) \models pre \Rightarrow cl$	<u>iff</u>	$(\varrho, \sigma) \models cl$ whenever $(\varrho, \sigma) \models pre$
$(\varrho, \sigma) \models \forall x : cl$	<u>iff</u>	$(\varrho, \sigma[x \mapsto a]) \models cl$ for all $a \in \mathcal{U}$

Outline

- 1 Introduction
- 2 Alternation-free Least Fixed Point Logic
- 3 Algorithms and data structures**
- 4 Experiments
- 5 Conclusion

Abstract Algorithm

- Differential and Bdd based algorithms
- Both algorithms operate on representations of two interpretations ς and ϱ from the semantics; called `env` and `result`
- Algorithms use different techniques and data structures, however they both share the same overall structure
- For both algorithms we have one function for each of the three syntactic categories
 - `SOLVE(cl_1, \dots, cl_s)`
 - `EXECUTE(cl)env`
 - `CHECK($pre, next$)env`

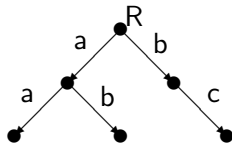
Differential vs. Bdd based algorithm

Differential algorithm	Bdd based algorithm
relations represented as prefix trees	relations represented as Bdd's
works on single tuples at a time	works on entire relations at a time
propagation of differences	propagations of entire relations
use of runtime stack	use of explicit stack

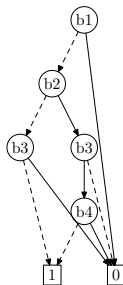
Data structures

Consider the relation: $R = \{(a, a), (a, b), (b, c)\}$

Differential algorithm



Bdd based algorithm



Outline

- 1 Introduction
- 2 Alternation-free Least Fixed Point Logic
- 3 Algorithms and data structures
- 4 Experiments**
- 5 Conclusion

Experiments overview

- Both algorithms have been implemented in F# allowing us to perform experiments.
- We performed experiments for clauses with different characteristics.
- We used problems arising from modal logic representation of analysis problems.
- Properties of programs are expressed in Action Computation Tree Logic (ACTL).
- ACTL properties have been translated into ALFP clauses.

Experiments overview

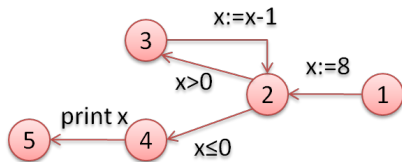
- The connection between iterative data-flow analysis and model checking has already been known.
- As an example the following ACTL formula expresses whether a variable x may have been last assigned at label l :

$$\text{wasLastAssgnAt}(x, l) = \mathbf{EF}_{(a|a \neq \text{mod}_x)}(\mathbf{EX}_{\text{mod}_x}(l))$$

```

[x := 8]1;
while [x > 0]2 do
  [x := x - 1]3;
[print x]4;

```

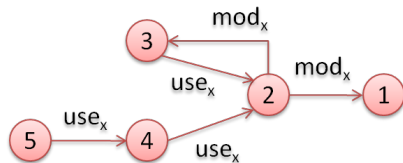


Experiments overview

- The connection between iterative data-flow analysis and model checking has already been known.
- As an example the following ACTL formula expresses whether a variable x *may* have been last assigned at label l :

$$\text{wasLastAssgnAt}(x, l) = \mathbf{EF}_{(a|a \neq \text{mod}_x)}(\mathbf{EX}_{\text{mod}_x}(l))$$

```
[x := 8]1;
while [x > 0]2 do
  [x := x - 1]3;
[print x]4;
```



Example of input clauses

The ALCT formula:

$$\text{wasLastAssgnAt}(x, l) = \mathbf{EF}_{(a|a \neq \text{mod}_x)}(\mathbf{EX}_{\text{mod}_x}(l))$$

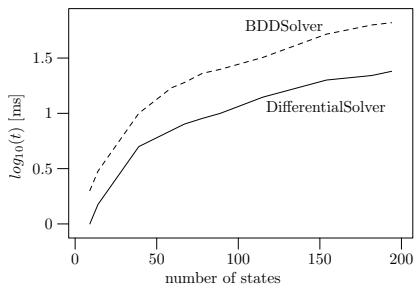
Corresponds to the following ALFP clauses:

$$[\forall s : [\exists a : \exists s' : T(s, a, s') \wedge \text{mod}_x(a) \wedge R_l(s')] \Rightarrow R_{(\mathbf{EX}_{\text{mod}_x}(l))}(s)] \wedge$$

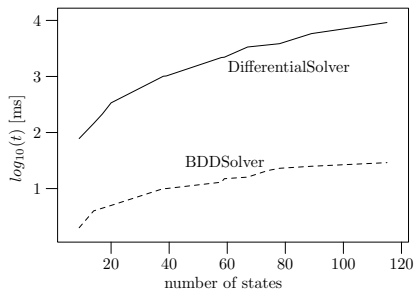
$$[\forall s : [\exists a : \exists s' : T(s, a, s') \wedge \neg \text{mod}_x(a) \wedge R_{(\mathbf{EX}_{\text{mod}_x}(l))}(s')] \Rightarrow F(s)] \wedge$$

$$[\forall s : [\exists a : \exists s' : T(s, a, s') \wedge F(s')] \Rightarrow F(s)]$$

Experiments results

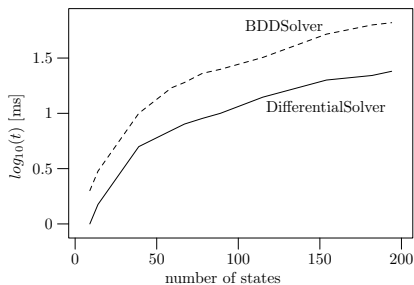


clauses without universal
quantification in preconditions

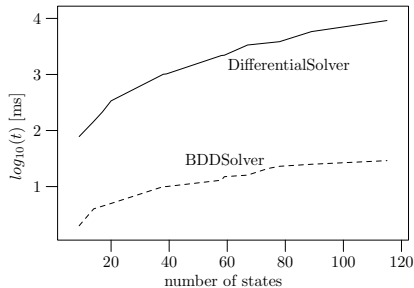


clauses including universal
quantification in preconditions

Experiments results



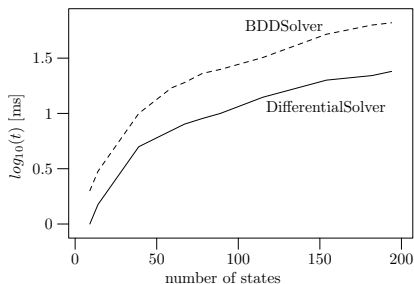
clauses without universal
quantification in preconditions



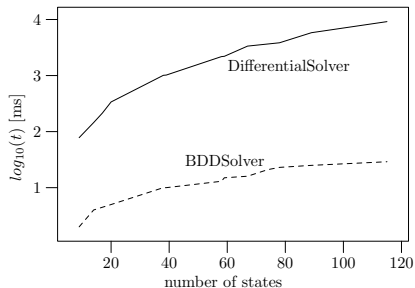
clauses including universal
quantification in preconditions

Due to propagation of differences in differential algorithm, it outperformed Bdd based one.

Experiments results



clauses without universal
quantification in preconditions



clauses including universal
quantification in preconditions

Bdd based algorithm utilizes universal quantification on Bdd, which is computed in $O(n^2)$.

Outline

- 1 Introduction
- 2 Alternation-free Least Fixed Point Logic
- 3 Algorithms and data structures
- 4 Experiments
- 5 Conclusion**

Conclusion

Challenge:

- Development of two algorithms that use ALFP as an input language,
- Show how certain problems can benefit from certain solver algorithms and data structures.

Conclusion

Challenge:

- Development of two algorithms that use ALFP as an input language,
- Show how certain problems can benefit from certain solver algorithms and data structures.

Future Work:

- Employ heuristics for clause tuning and variable ordering,
- Identify other data structures and methods that can be efficiently used in constraint solving,
- Go beyond powersets.