

Tree Automata for Program Analysis

Helmut Seidl



CIAA 2009

Motivation

- ▶ Finite tree automata allow to reason about (certain) sets of trees.

Motivation

- ▶ Finite tree automata allow to reason about (certain) sets of trees.
- ▶ Program analysis tries to statically infer properties of the runtime behavior of a program,

Motivation

- ▶ Finite tree automata allow to reason about (certain) sets of trees.
- ▶ Program analysis tries to statically infer properties of the runtime behavior of a program, e.g.,
 - values of variables;
 - reachable configurations.

Motivation

- ▶ Finite tree automata allow to reason about (certain) sets of trees.
- ▶ Program analysis tries to statically infer properties of the runtime behavior of a program, e.g.,
 - values of variables;
 - reachable configurations.
- ▶ Often, such analyses **result** in tree automata.
- ▶ A formalism is required to conveniently express and perform **operations** on tree automata.

Overview

- Set-based Analysis of Functional Programs
- Set-based Analysis of Logic Programs
- Reachability for DPNs

1. Functional Programs

Idea

- ▶ Determine for every sub-expression e in the program a (safe super-) set of the terms to which e evaluates.

Idea

- ▶ Determine for every sub-expression e in the program a (safe super-) set of the terms to which e evaluates.
- ▶ Describe these sets by means of regular **tree grammars** Jones 1987

Idea

- ▶ Determine for every sub-expression e in the program a (safe super-) set of the terms to which e evaluates.
- ▶ Describe these sets by means of regular **tree grammars** Jones 1987
- ▶ Use **set constraints** instead Heintze 1994

Idea

- ▶ Determine for every sub-expression e in the program a (safe super-) set of the terms to which e evaluates.
- ▶ Describe these sets by means of regular **tree grammars** Jones 1987
- ▶ Use **set constraints** instead Heintze 1994
- ▶ Why not **Horn clauses** ???

The append Function

```
let rec a = fun x → match x with
              [] → fun y → y
              h::t → fun y → h::a t y
in a [1; 2] [3]
```

The Horn Clauses

$\text{val}_a(\text{fun}(x, \text{match})) \Leftarrow$

$\text{val}_{\text{match}}(\text{fun}(y, y)) \Leftarrow \text{val}_x([])$

$\text{val}_{\text{match}}(\text{fun}(y, \text{cons})) \Leftarrow \text{val}_x(- :: -)$

$\text{val}_h(Z) \Leftarrow \text{val}_x(Z :: -)$

$\text{val}_{\text{cons}}(H :: T) \Leftarrow \text{val}_h(H), \text{val}_{a.t.y}(T)$

$\text{val}_{a.t.y}(Z) \Leftarrow \text{val}_{a.t}(\text{fun}(x, \text{match})), \text{val}_{\text{match}}(Z)$

...

Discussion

- ▶ All heads of the generated constraints are linear !
No variable occurs repeatedly in bodies !!

$\implies \mathcal{H}_3$

Discussion

- ▶ All heads of the generated constraints are linear !
No variable occurs repeatedly in bodies !!

$\implies \mathcal{H}_3$

- ▶ For every set of \mathcal{H}_3 -clauses, an equivalent set of **normal** clauses can be found in **polynomial** time.

Discussion

- ▶ All heads of the generated constraints are linear !
No variable occurs repeatedly in bodies !!

$\implies \mathcal{H}_3$

- ▶ For every set of \mathcal{H}_3 -clauses, an equivalent set of **normal** clauses can be found in **polynomial** time.
- ▶ **Normal** clauses are finite tree automata ...

The Result

$$\text{val}_{\text{let}}(Z_1 :: Z_2) \quad \Leftarrow \quad \text{val}_3(Z_1), \text{val}_{[]} (Z_2)$$

$$\text{val}_{\text{let}}(Z_1 :: Z_2) \quad \Leftarrow \quad \text{val}_h(Z_1), \text{val}_{\text{a.t.y}}(Z_2)$$

$$\text{val}_{\text{a.t.y}}(Z_1 :: Z_2) \quad \Leftarrow \quad \text{val}_3(Z_1), \text{val}_{[]} (Z_2)$$

$$\text{val}_{\text{a.t.y}}(Z_1 :: Z_2) \quad \Leftarrow \quad \text{val}_h(Z_1), \text{val}_{\text{a.t.y}}(Z_2)$$

$$\text{val}_h(1) \quad \Leftarrow$$

$$\text{val}_h(2) \quad \Leftarrow$$

$$\text{val}_3(3) \quad \Leftarrow$$

$$\text{val}_{[]} ([]) \quad \Leftarrow$$

Discussion

- ▶ The analysis finds:

$$\{[x_1; \dots; x_k; 3] \mid x_i \in \{1, 2\}\}$$

- ▶ The analysis does not discover that the set of all occurring lists is finite :-)

Discussion

- ▶ The analysis finds:

$$\{[x_1; \dots; x_k; 3] \mid x_i \in \{1, 2\}\}$$

- ▶ The analysis does not discover that the set of all occurring lists is finite :-(

▶ The clauses are **program-specific**



The analysis consists of two steps:

- (1) Clause generation
- (2) Clause normalization

Alternative: Set Constraints

Heintze 1994

$$\begin{aligned}
 \text{val}_a &\supseteq \{\mathbf{fun}(x, \text{match})\} \\
 \text{val}_{\text{match}} &\supseteq (\{\llbracket _ \rrbracket\} \cap \text{val}_x \neq \emptyset); \{\mathbf{fun}(y, y)\} \\
 \text{val}_{\text{match}} &\supseteq (_ :: _ \cap \text{val}_x \neq \emptyset); \{\mathbf{fun}(y, \text{cons})\} \\
 \text{val}_h &\supseteq \pi_{::,1} \text{val}_x \\
 \text{val}_{\text{cons}} &\supseteq \text{val}_h :: \text{val}_{a.t.y} \\
 \text{val}_{a.t.y} &\supseteq (\{\mathbf{fun}(x, \text{match})\} \cap \text{val}_{a.t} \neq \emptyset); \text{val}_{\text{match}} \\
 &\dots
 \end{aligned}$$

Discussion

- ▶ The analysis requires the set operators:

Projection $\pi_{::,j} X$

Constructor application $X :: Y$

Check $(e \cap X \neq \emptyset); Y$

Discussion

- ▶ The analysis requires the set operators:

Projection $\pi_{::,j} X$

Constructor application $X :: Y$

Check $(e \cap X \neq \emptyset); Y$

- ▶ All these can be modelled by \mathcal{H}_3 -clauses :-)

Program-Independent Formulation

`val(fun(X, E), fun(X, E))` \Leftarrow

`val(H :: T, Z1 :: Z2)` \Leftarrow

`val(H, Z1), val(T, Z2)`

Program-Independent Formulation

$\text{val}(A, Z) \quad \Leftarrow \quad \text{val}(E, Z)$

...

$\text{val}(\text{fun}(X, E), \text{fun}(X, E)) \quad \Leftarrow$

$\text{val}(H :: T, Z_1 :: Z_2) \quad \Leftarrow \quad \text{val}(H, Z_1), \text{val}(T, Z_2)$

Program-Independent Formulation

$\text{reach}(\text{let}(a, \dots, \text{app}(\text{app}(a, [1; 2]), [3]))) \Leftarrow$

$\text{reach}(E) \Leftarrow \text{reach}(\text{let}(-, E, -))$

$\text{val}(A, Z) \Leftarrow$
 $\text{val}(E, Z)$

...

$\text{val}(\text{fun}(X, E), \text{fun}(X, E)) \Leftarrow$

$\text{val}(H :: T, Z_1 :: Z_2) \Leftarrow$
 $\text{val}(H, Z_1), \text{val}(T, Z_2)$

Program-Independent Formulation

$\text{reach}(\text{let}(a, \dots, \text{app}(\text{app}(a, [1; 2]), [3]))) \Leftarrow$

$\text{reach}(E) \Leftarrow \text{reach}(\text{let}(_, E, _))$

$\text{val}(A, Z) \Leftarrow \text{reach}(\text{let}(A, E, _)),$
 $\text{val}(E, Z)$

...

$\text{val}(\text{fun}(X, E), \text{fun}(X, E)) \Leftarrow \text{reach}(\text{fun}(X, E))$

$\text{val}(H :: T, Z_1 :: Z_2) \Leftarrow \text{reach}(H :: T),$
 $\text{val}(H, Z_1), \text{val}(T, Z_2)$

Discussion

- ▶ The rules both for functions and constructor applications violate the restrictions of our favorite class \mathcal{H}_1 :-)

Discussion

- ▶ The rules both for functions and constructor applications violate the restrictions of our favorite class \mathcal{H}_1 :-(
- ▶ Finiteness analysis of the predicate `reach/1` on the other hand, allows to instantiate variables with finite range !

Discussion

- ▶ The rules both for functions and constructor applications violate the restrictions of our favorite class \mathcal{H}_1 :-)
- ▶ Finiteness analysis of the predicate `reach/1` on the other hand, allows to instantiate variables with finite range !
- ▶ Instantiation will recover the extensive form :-)

Summary

- ▶ Horn clauses provide a convenient tool for approximating the collecting semantics :-)

Summary

- ▶ Horn clauses provide a convenient tool for approximating the collecting semantics :-)
- ▶ A program-dependent specification can be obtained via \mathcal{H}_3 -clauses — which can be recovered from general Horn clauses through **instantiateion** of finite predicates :-)

Summary

- ▶ Horn clauses provide a convenient tool for approximating the collecting semantics :-)
- ▶ A program-dependent specification can be obtained via \mathcal{H}_3 -clauses — which can be recovered from general Horn clauses through **instantiate** of finite predicates :-)
- ▶ A trivial **finiteness analysis** may suffice.

2. Prolog

Idea

- ▶ Approximate the least model of a program via **set constraints** **Heintze/Jaffar 1990**

Idea

- ▶ Approximate the least model of a program via **set constraints** Heintze/Jaffar 1990
- ▶ Use **uniform** Horn clauses instead! Frühwirth et al. 1991

Idea

- ▶ Approximate the least model of a program via **set constraints** Heintze/Jaffar 1990
- ▶ Use **uniform** Horn clauses instead! Frühwirth et al. 1991
- ▶ Why not approximate with \mathcal{H}_1 ?
Weidenbach 1999
Nielson/Riis-Nielson/S. 2002
Goubault-Larrecq 2005

The Class of \mathcal{H}_1 -Clauses

\mathcal{H}_1 -Clauses are of the form:

$$p(X_1, \dots, X_k) \Leftarrow \text{any} \quad \text{or} \quad p(a(X_1, \dots, X_k)) \Leftarrow \text{any}$$

The Class of \mathcal{H}_1 -Clauses

\mathcal{H}_1 -Clauses are of the form:

$$p(X_1, \dots, X_k) \Leftarrow \text{any} \quad \text{or} \quad p(a(X_1, \dots, X_k)) \Leftarrow \text{any}$$

For any set of \mathcal{H}_1 -clauses an equivalent set of **normal clauses** can be constructed in exponential time :-)

The Class of \mathcal{H}_1 -Clauses

\mathcal{H}_1 -Clauses are of the form:

$$p(X_1, \dots, X_k) \Leftarrow \text{any} \quad \text{or} \quad p(a(X_1, \dots, X_k)) \Leftarrow \text{any}$$

For any set of \mathcal{H}_1 -clauses an equivalent set of **normal clauses** can be constructed in exponential time :-)

Any set of Horn clauses can be **approximated** by a set of \mathcal{H}_1 -clauses ...

Example: append

$\text{app}([], Y, Y) \Leftarrow$
 $\text{app}([H|T], Y, [H|Z]) \Leftarrow \text{app}(T, Y, Z)$
 $\Leftarrow \text{app}([1, 2], [3], Z)$

... results in:

The \mathcal{H}_1 -Approximation

$$\text{app}(Z, Y, Y') \quad \Leftarrow \quad \text{aux}_{[]} (Z)$$

$$\text{aux}_{[]} ([]) \quad \Leftarrow$$

The \mathcal{H}_1 -Approximation

$$\text{app}(Z, Y, Y') \Leftarrow \text{aux}_{[]} (Z)$$

$$\text{aux}_{[]} ([]) \Leftarrow$$

$$\text{app}(Z_1, Y, Z_2) \Leftarrow \text{aux}_{[H|T]} (Z_1), \text{aux}_{[H|Z]} (Z_2), \text{app}(T, Y, Z)$$

The \mathcal{H}_1 -Approximation

$$\text{app}(Z, Y, Y') \Leftarrow \text{aux}_{[]} (Z)$$

$$\text{aux}_{[]} ([]) \Leftarrow$$

$$\text{app}(Z_1, Y, Z_2) \Leftarrow \text{aux}_{[H|T]} (Z_1), \text{aux}_{[H|Z]} (Z_2), \text{app}(T, Y, Z)$$

$$\text{aux}_{[H|T]} ([H|T]) \Leftarrow \text{app}(T, Y, Z)$$

$$\text{aux}_{[H|Z]} ([H|Z]) \Leftarrow \text{app}(T, Y, Z)$$

...

... with the Result:

$\text{val}_Z([Z_1|Z_2]) \Leftarrow \text{top}(Z_1), \text{aux}(Z_2)$

$\text{val}_Z([Z_1|Z_2]) \Leftarrow \text{top}(Z_1), \text{top}(Z_2)$

$\text{aux}([]) \Leftarrow$

$\text{aux}([Z_1|Z_2]) \Leftarrow \text{top}(Z_1), \text{aux}(Z_2)$

... with the Result:

$\text{val}_Z([Z_1|Z_2]) \Leftarrow \text{top}(Z_1), \text{aux}(Z_2)$

$\text{val}_Z([Z_1|Z_2]) \Leftarrow \text{top}(Z_1), \text{top}(Z_2)$

$\text{aux}([]) \Leftarrow$

$\text{aux}([Z_1|Z_2]) \Leftarrow \text{top}(Z_1), \text{aux}(Z_2)$



Listness and all information about the content of the result is lost :-)

Discussion

- ▶ One source of imprecision is the non-linear rule:

$$\text{app}([], Y, Y) \Leftarrow$$

Discussion

- ▶ One source of imprecision is the non-linear rule:

$$\text{app}([], Y, Y) \Leftarrow$$

- ▶ Better results can be obtained if additionally **call patterns** are tracked !

⇒⇒ Magic Set Transformation

Magic Sets

- ▶ For every predicate p/k , we introduce a new predicate called_p/k with the clauses

$$\text{called}_p(\underline{t}) \Leftarrow \text{for the query } \Leftarrow p(\underline{t})$$

Magic Sets

- ▶ For every predicate p/k , we introduce a new predicate called_p/k with the clauses

$$\text{called}_p(\underline{t}) \Leftarrow \text{for the query } \Leftarrow p(\underline{t})$$



$$\text{called}_{p_i}(\underline{t}_i) \Leftarrow \text{called}_p(\underline{t}), p_1(\underline{t}_1), \dots, p_{i-1}(\underline{t}_{i-1})$$

$$p(\underline{t}) \Leftarrow \text{called}_p(\underline{t}), p_1(\underline{t}_1), \dots, p_m(\underline{t}_m)$$

for every clause:

$$p(\underline{t}) \Leftarrow p_1(\underline{t}_1), \dots, p_m(\underline{t}_m)$$

Example: append (Cont.)

`app([], Y, Y)` \Leftarrow `called([], Y, Y)`
`app([H|T], Y, [H|Z])` \Leftarrow `called([H|T], Y, [H|Z])`,
`app(T, Y, Z)`
`called(T, Y, Z)` \Leftarrow `called([H|T], Y, [H|Z])`
`called([1, 2], [3], Z)` \Leftarrow

The \mathcal{H}_1 -Approximation (Cont.)

$$\text{val}_Z([Z_1|Z_2]) \Leftarrow \text{val}_2(Z_1), \text{aux}_1(Z_2)$$

$$\text{val}_Z([Z_1|Z_2]) \Leftarrow \text{val}_1(Z_1), \text{aux}_2(Z_2)$$

$$\text{aux}_1([Z_1|Z_2]) \Leftarrow \text{val}_3(Z_1), \text{val}_{[]} (Z_2)$$

$$\text{aux}_2([Z_1|Z_2]) \Leftarrow \text{val}_2(Z_1), \text{aux}_1(Z_2)$$

$$\text{aux}_2([Z_1|Z_2]) \Leftarrow \text{val}_1(Z_1), \text{aux}_2(Z_2)$$

$$\text{val}_i(i) \Leftarrow$$

$$\text{val}_{[]}([]) \Leftarrow$$

Alternative: Set Constraints

Heintze/Jaffar 1990

- ▶ The analysis requires the set operators:

Projection $\pi_{::,j} X$

Constructor application $X :: Y$

Check $(e \cap X \neq \emptyset); Y$

Intersection $X \cap Y$

Alternative: Set Constraints

Heintze/Jaffar 1990

- ▶ The analysis requires the set operators:

Projection $\pi_{::,j} X$

Constructor application $X :: Y$

Check $(e \cap X \neq \emptyset); Y$

Intersection $X \cap Y$

- ▶ All these can be modelled by \mathcal{H}_1 -clauses :-)

Alternative: Set Constraints

Heintze/Jaffar 1990

- ▶ The analysis requires the set operators:

Projection $\pi_{::,j} X$

Constructor application $X :: Y$

Check $(e \cap X \neq \emptyset); Y$

Intersection $X \cap Y$

- ▶ All these can be modelled by \mathcal{H}_1 -clauses :-)
- ▶ Set constraints still are **less precise ...**

Precision

The \mathcal{H}_1 -clause:

$$p(a(X, Y)) \Leftarrow q(b(X, Y))$$

is approximated by the set constraint:

$$\text{val}_p \supseteq a(\pi_{b,1} \text{val}_q, \pi_{b,2} \text{val}_q)$$

Precision

The \mathcal{H}_1 -clause:

$$p(a(X, Y)) \Leftarrow q(b(X, Y))$$

is approximated by the set constraint:

$$\text{val}_p \supseteq a(\pi_{b,1} \text{val}_q, \pi_{b,2} \text{val}_q)$$

... which for

$$\text{val}_q = \{b(1, 2), b(2, 1)\}$$

returns:

$$\text{val}_p = \{a(1, 1), a(1, 2), a(2, 1), a(2, 2)\}$$

Summary

- ▶ Horn clauses provide a convenient tool for describing analysis problems :-)

Summary

- ▶ Horn clauses provide a convenient tool for describing analysis problems :-)
- ▶ The approximation of arbitrary Horn clauses with \mathcal{H}_1 -clauses may result in depressing results :-(

Summary

- ▶ Horn clauses provide a convenient tool for describing analysis problems :-)
- ▶ The approximation of arbitrary Horn clauses with \mathcal{H}_1 -clauses may result in depressing results :-(
- ▶ Better results can be hoped for by taking the query into account via Magic Sets :-)

3. Dynamic Pushdown Networks

Rules

pa	\rightarrow	q_1bc	call	}	PD
pa	\rightarrow	q_1b	do		
pa	\rightarrow	q_1	return		

Rules

$pa \rightarrow q_1 bc$	call	} PD	} DPN
$pa \rightarrow q_1 b$	do		
$pa \rightarrow q_1$	return		
$pa \rightarrow q_1 b q_2 c$	spawn		

Rules

$pa \rightarrow q_1 bc$	call	}	PD	}	DPN
$pa \rightarrow q_1 b$	do				
$pa \rightarrow q_1$	return				
$pa \rightarrow q_1 b q_2 c$	spawn				

a, b, c, \dots program points + local data

Rules

$pa \rightarrow q_1 bc$	$call$	}	PD	}	DPN
$pa \rightarrow q_1 b$	do				
$pa \rightarrow q_1$	$return$				
$pa \rightarrow q_1 b q_2 c$	$spawn$				

a, b, c, \dots program points + local data
 p, q, q_1, q_2, \dots (thread-local) global data

Java-like Security Model

State q — set of currently held **privileges**

- ▶ Some operations **require** privileges.
- ▶ Others **acquire** privileges or **release** them.

Java-like Security Model

State q — set of currently held **privileges**

- ▶ Some operations **require** privileges.
- ▶ Others **acquire** privileges or **release** them.

Question:

Are always sufficiently many privileges held?

Java-like Security Model

State q — set of currently held **privileges**

- ▶ Some operations **require** privileges.
- ▶ Others **acquire** privileges or **release** them.

Question:

Are always sufficiently many privileges held?

Observation

The set of potentially dangerous configurations is **regular!**

The Analysis Problem

Given R regular set of configurations.

Question:

$$\{\text{conf} \mid q_0 Z \rightarrow^* \text{conf}\} \cap R \neq \emptyset ?$$

The Analysis Problem

Given R regular set of configurations.

Question:

$$\{\text{conf} \mid q_0 Z \rightarrow^* \text{conf}\} \cap R \neq \emptyset ?$$

Observation

- ▶ The set of reachable configurations of PDs is regular.

The Analysis Problem

Given R regular set of configurations.

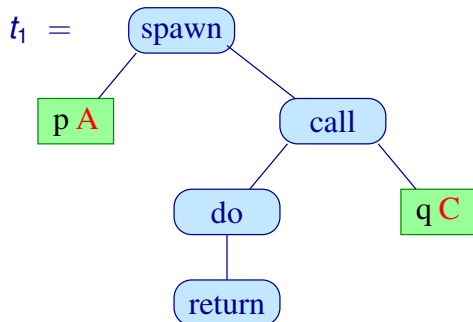
Question:

$$\{\text{conf} \mid q_0 Z \rightarrow^* \text{conf}\} \cap R \neq \emptyset ?$$

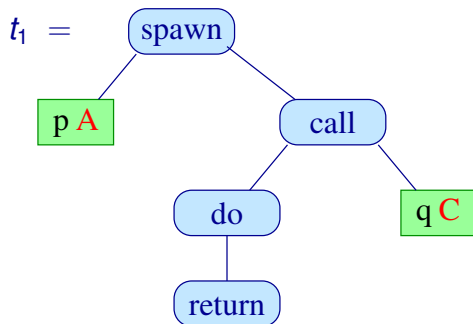
Observation

- ▶ The set of reachable configurations of PDs is regular.
- ▶ For DPNs, it is contextfree.

Idea: Execution Trees



Idea: Execution Trees



$$\text{yield}(t_1) = pA qC$$

Idea: Execution Trees

Predicates:

- $[pa, q]$ — trees for terminated executions
- $[pa]$ — trees for not yet terminated executions

Idea: Execution Trees

Predicates:

- $[pa, q]$ — trees for terminated executions
- $[pa]$ — trees for not yet terminated executions

Clauses for $pa \rightarrow q_1 bc$:

Idea: Execution Trees

Predicates:

- $[pa, q]$ — trees for terminated executions
- $[pa]$ — trees for not yet terminated executions

Clauses for $pa \rightarrow q_1 bc$:

$$[pa](\text{enter}(X, c)) \Leftarrow [q_1 b](X)$$

$$[pa](\text{call}(X, Y)) \Leftarrow [q_1 b, q_2](X), [q_2 c](Y)$$

$$[pa, q](\text{call}(X, Y)) \Leftarrow [q_1 b, q_2](X), [q_2 c, q](Y)$$

... more Clauses

Clauses for $pa \rightarrow q_1 b q_2 c$:

... more Clauses

Clauses for $pa \rightarrow q_1 b q_2 c$:

$$[pa](\text{spawn}(X, Y)) \Leftarrow [q_1 b](X), [q_2 c](Y)$$

$$[pa, q](\text{spawn}(X, Y)) \Leftarrow [q_1 b](X), [q_2 c, q](Y)$$

... more Clauses

Clauses for $pa \rightarrow q_1 b q_2 c$:

$$[pa](\text{spawn}(X, Y)) \Leftarrow [q_1 b](X), [q_2 c](Y)$$

$$[pa, q](\text{spawn}(X, Y)) \Leftarrow [q_1 b](X), [q_2 c, q](Y)$$

Clauses for $pa \rightarrow q_1 b$:

... more Clauses

Clauses for $pa \rightarrow q_1 b q_2 c$:

$$[pa](\text{spawn}(X, Y)) \Leftarrow [q_1 b](X), [q_2 c](Y)$$

$$[pa, q](\text{spawn}(X, Y)) \Leftarrow [q_1 b](X), [q_2 c, q](Y)$$

Clauses for $pa \rightarrow q_1 b$:

$$[pa](\text{do}(X)) \Leftarrow [q_1 b](X)$$

$$[pa, q](\text{do}(X)) \Leftarrow [q_1 b, q](X)$$

... more Clauses (finish)

Clauses for $pa \rightarrow q$:

... more Clauses (finish)

Clauses for $pa \rightarrow q$:

$$[pa, q](\text{pop}(X)) \Leftarrow$$

... more Clauses (finish)

Clauses for $pa \rightarrow q$:

$$[pa, q](\text{pop}(X)) \Leftarrow$$

Stopping Rule :

$$pa \Leftarrow$$

Theorem

$$pa \rightarrow^* \text{conf} \quad \text{iff} \quad \text{conf} = \text{yield}(t) \\ \text{for some } t \text{ with } [pa](t)$$

The set of reachable configurations of a DPN P with n states can be computed in time $\mathcal{O}(|P| \cdot n^2)$.

Theorem

$$pa \rightarrow^* \text{conf} \quad \text{iff} \quad \text{conf} = \text{yield}(t) \\ \text{for some } t \text{ with } [pa](t)$$

The set of reachable configurations of a DPN P with n states can be computed in time $\mathcal{O}(|P| \cdot n^2)$.

yield is a (single-state) tree-to-string transducer.



$\text{yield}^{-1}(R)$ regular — whenever R regular.

The Analysis

Idea 1:

- ▶ Compile P into automaton;
- ▶ Compile $\text{yield}^{-1}(R)$ into automaton;
- ▶ intersect!

The Analysis

Idea 1:

- ▶ Compile P into automaton;
- ▶ Compile $\text{yield}^{-1}(R)$ into automaton;
- ▶ intersect!

Idea 2:

- ▶ Represent analysis by **generic** Horn clauses;
- ▶ Represent transition relation of DPN and property by ground facts queried by the analysis;
- ▶ Solve!

Generic Clauses for Calls

$$\text{nont}(P, A, \text{enter}(X, C)) \quad \Leftarrow \quad \text{nont}(Q_1, B, X)$$
$$\text{nont}(P, A, \text{call}(X, Y)) \quad \Leftarrow \quad \begin{array}{l} \text{term}(Q_1, B, Q_2, X), \\ \text{nont}(Q_2, C, X) \end{array}$$
$$\text{term}(P, A, Q, \text{call}(X, Y)) \quad \Leftarrow \quad \begin{array}{l} \text{term}(Q_1, B, Q_2, X), \\ \text{term}(Q_2, C, Q, X) \end{array}$$

Generic Clauses for Calls

$$\text{nont}(P, A, \text{enter}(X, C)) \quad \Leftarrow \quad \text{dpn}(P, A, \text{call}(Q_1, B, C)), \\ \text{nont}(Q_1, B, X)$$

$$\text{nont}(P, A, \text{call}(X, Y)) \quad \Leftarrow \quad \text{dpn}(P, A, \text{call}(Q_1, B, C)), \\ \text{term}(Q_1, B, Q_2, X), \\ \text{nont}(Q_2, C, X)$$

$$\text{term}(P, A, Q, \text{call}(X, Y)) \quad \Leftarrow \quad \text{dpn}(P, A, \text{call}(Q_1, B, C)), \\ \text{term}(Q_1, B, Q_2, X), \\ \text{term}(Q_2, C, Q, X)$$

Generic Clauses for Properties

- // `check`(T_1, X, T_2) holds for all execution trees X whose
- // `yield` goes from T_1 to T_2 of the property

$$\text{check}(T_1, \text{enter}(X, C), T_2) \Leftarrow \text{check}(T_1, X, T), \text{property}(T, C, T_2)$$

$$\text{check}(T_1, \text{call}(X, Y), T_2) \Leftarrow \text{check}(T_1, X, T), \text{check}(T, Y, T_2)$$

Discussion

- ▶ Generic clauses only fall into class \mathcal{H}_1 after instantiation :-)

Discussion

- ▶ Generic clauses only fall into class \mathcal{H}_1 after instantiation :-)
- ▶ This simple approach also works for:
 - DPNs with tests;
 - DPNs with well-nested locking ...

Well-Nested Locking

- ▶ Finite set \mathcal{L} of locks
- ▶ Later locks are released earlier

Well-Nested Locking

- ▶ Finite set \mathcal{L} of locks
- ▶ Later locks are released earlier

Execution trees now are no longer necessarily **realizable**

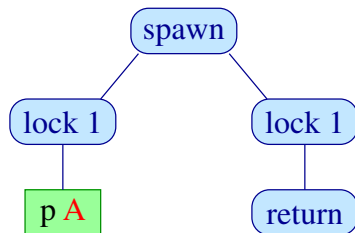
...

Well-Nested Locking

- ▶ Finite set \mathcal{L} of locks
- ▶ Later locks are released earlier

Execution trees now are no longer necessarily **realizable**

...



Lammich, MMO, Wenner, 2009

The set of realizable execution trees of a DPN with well-nested locking is **regular**.

Lammich, MMO, Wenner, 2009

The set of realizable execution trees of a DPN with well-nested locking is **regular**.

Idea:

Maintain the **acquisition history**, i.e.,

- ▶ the set of initially released locks;
- ▶ the set of used locks;
- ▶ the set of finally acquired locks;

Lammich, MMO, Wenner, 2009

The set of realizable execution trees of a DPN with well-nested locking is **regular**.

Idea:

Maintain the **acquisition history**, i.e.,

- ▶ the set of initially released locks;
- ▶ the set of used locks;
- ▶ the set of finally acquired locks;
- ▶ the acyclic acquisition and release graphs recording dependencies between these.

Challenge

- ▶ The set of global/local states is too large to be spelled out **explicitly** before-hand :-)

Challenge

- ▶ The set of global/local states is too large to be spelled out **explicitly** before-hand :-(
▶ The lock automaton is too large to be spelled out **explicitly** before-hand :-(
▶ Construct a tractable algorithm which computes the automata and their intersection **on the fly ...**

Challenge

- ▶ The set of global/local states is too large to be spelled out **explicitly** before-hand :-)
- ▶ The lock automaton is too large to be spelled out **explicitly** before-hand :-)
- ▶ Construct a tractable algorithm which computes the automata and their intersection **on the fly ...**
- ▶ One idea is to enhance \mathcal{H}_1 -normalization with **goal-directed instantiation** :-)

Conclusion

- ▶ Horn clauses are a decent formalism for the specification of program analyses.
- ▶ **Normalization** for particular classes allows to compute automata.

Conclusion

- ▶ Horn clauses are a decent formalism for the specification of program analyses.
- ▶ **Normalization** for particular classes allows to compute automata.
- ▶ **Instantiation** allows for program independent specifications of analyses for functional programs.

Conclusion

- ▶ Horn clauses are a decent formalism for the specification of program analyses.
- ▶ **Normalization** for particular classes allows to compute automata.
- ▶ **Instantiation** allows for program independent specifications of analyses for functional programs.
- ▶ **Magic sets** enable goal-directed computations and thus give decent results for Prolog.

Conclusion

- ▶ Horn clauses are a decent formalism for the specification of program analyses.
- ▶ **Normalization** for particular classes allows to compute automata.
- ▶ **Instantiation** allows for program independent specifications of analyses for functional programs.
- ▶ **Magic sets** enable goal-directed computations and thus give decent results for Prolog.
- ▶ All these ideas together are required for DPNs with nested locks :-)

Thank You!