Approximation of Paths:

Every clause

$$p(t_1, \ldots, t_k) \;\leftarrow\; \alpha$$

is approximated by the clauses:

$$\begin{array}{lcll} p_j(w) & \leftarrow & \bigwedge \Pi(\alpha) & \text{where} \\ \Pi(g_1, \ldots, g_m) & = & \Pi(g_1) \cup \ldots \cup \Pi(g_m) & \\ \Pi(q(s_1, \ldots, s_n)) & = & \{q_i(w) \mid w \in \Pi(s_i)\} & \end{array}$$

$(j = 1, \ldots, k, w \in \Pi(t_j))$.

Example:

$$\begin{array}{lcl} \mathsf{add}(0, Y, Y) & \leftarrow & \mathsf{nat}(Y) \\ \mathsf{add}(s(X), Y, s(Z)) & \leftarrow & \mathsf{add}(X, Y, Z) \end{array}$$

yields:

$$
\begin{aligned}
\mathsf{add}_1(0) &\leftarrow \mathsf{nat}_1(Y) \\
\mathsf{add}_2(Y) &\leftarrow \mathsf{nat}_1(Y) \\
\mathsf{add}_3(Y) &\leftarrow \mathsf{nat}_1(Y) \\
\\
\mathsf{add}_1(s_1\, X) &\leftarrow \mathsf{add}_1(X), \mathsf{add}_2(Y), \\
&\quad\ \mathsf{add}_3(Z) \\
\mathsf{add}_2(Y) &\leftarrow \mathsf{add}_1(X), \mathsf{add}_2(Y), \\
&\quad\ \mathsf{add}_3(Z) \\
\mathsf{add}_3(s_1\, Z) &\leftarrow \mathsf{add}_1(X), \mathsf{add}_2(Y), \\
&\quad\ \mathsf{add}_3(Z)
\end{aligned}
$$

Discussion:

- Every literal has at most one occurrence of a variable.
- The literals $q_j(w_j Y)$ where the variable Y does not occur in the head, represent tests:

  If there is a w with $w_j w \in [\![q_j]\!]_{C^\sharp}$ for all such j, then we can cancel these literals.

  If there is no such w, then we can cancel the clause ...

... in the Example:

The literals:

$$\text{add}_1(X), \text{add}_2(Y), \text{add}_3(Z)$$

are all satisfiable :-)

We conclude:

$$
\begin{array}{lcl}
\mathsf{add}_1(0) & \leftarrow & \\
\mathsf{add}_2(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\mathsf{add}_3(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\\
\mathsf{add}_1(s_1\ X) & \leftarrow & \mathsf{add}_1(X) \\
\mathsf{add}_2(Y) & \leftarrow & \mathsf{add}_2(Y) \\
\mathsf{add}_3(s_1\ Z) & \leftarrow & \mathsf{add}_3(Z)
\end{array}
$$

We conclude:

$$
\begin{aligned}
\mathsf{add}_1(0) &\leftarrow \\
\mathsf{add}_2(Y) &\leftarrow \mathsf{nat}_1(Y) \\
\mathsf{add}_3(Y) &\leftarrow \mathsf{nat}_1(Y) \\
\mathsf{add}_1(s_1\, X) &\leftarrow \mathsf{add}_1(X) \\
\mathsf{add}_3(s_1\, Z) &\leftarrow \mathsf{add}_3(Z)
\end{aligned}
$$

We verify:

Theorem

Assume that $\mathcal{C}$ is a set of clauses.

Let $\mathcal{C}^\sharp$ denote the corresponding set of clauses for the paths.

Then for all predicates p/k:

$$\Pi([\![p]\!]_{\mathcal{C}}) \subseteq [\![p_1]\!]_{\mathcal{C}^\sharp} \cup \ldots \cup [\![p_k]\!]_{\mathcal{C}^\sharp}$$

Proof:

Induction on the approximations of the respective fixpoints :-)

A set of clauses with unary predicates and unary constructors is called Alternating Pushdown System (APS).

Theorem

- Every APS is equivalent to a simple APS of the form:

$$\begin{array}{lll} p(a\,X) & \leftarrow & p_1(X), \ldots, p_r(X) \\ p(X) & \leftarrow & \\ p(b) & \leftarrow & \end{array}$$

- Every APS is equivalent to a normal APS of the form:

$$\begin{array}{lll} p(a\,X) & \leftarrow & p_1(X) \\ p(X) & \leftarrow & \\ p(b) & \leftarrow & \end{array}$$

Step 1: Removal of complicated heads:

For $w = a^{(1)} \dots a^{(m)}$ $(m > 1)$ we replace

$$p(w\,X) \leftarrow rhs \qquad \text{with:}$$

$$\begin{array}{lcl}
p(a^{(1)}\,X) & \leftarrow & p_2(X) \\
p_2(a^{(2)}\,X) & \leftarrow & p_3(X) \\
& \dots & \\
p_{m-1}(a^{(m-1)}\,X) & \leftarrow & p_m(X) \\
p_m(a^{(m)}\,X) & \leftarrow & rhs \\
& & // \quad p_j \text{ all new}
\end{array}$$

Step 1 (Cont.): Removal of complicated heads:

For $w = a^{(1)} \dots a^{(m)} b \quad (m > 0)$ we replace

$$p(w) \leftarrow \mathit{rhs} \qquad \text{with:}$$

$$\begin{array}{lcl} p(a^{(1)}\, X) & \leftarrow & p_2(X) \\ p_2(a^{(2)}\, X) & \leftarrow & p_3(X) \\ & \dots & \\ p_{m-1}(a^{(m-1)}\, X) & \leftarrow & p_m(X) \\ p_m(a^{(m)}\, X) & \leftarrow & p_{m+1}(X) \\ p_{m+1}(b) & \leftarrow & \mathit{rhs} \end{array}$$

// p_j all new

Step 2: Splitting

We separate independent parts of pre-conditions into auxiliary predicates:

$$\begin{array}{lll} head & \leftarrow & rest,\ p_1(w_1\, X), \ldots, p_m(w_m\, X) \\ & & \qquad (X \text{ does not occur in } head,\ rest) \end{array}$$

is replaced with:

$$\begin{array}{lll} head & \leftarrow & rest, q() \\ q() & \leftarrow & p_1(w_1\, X), \ldots, p_m(w_m\, X) \end{array}$$

for a new predicate $q/0$.

Step 3: Normalization

We add simpler derived clauses:

$$
\begin{array}{lcl}
\mathit{head} & \leftarrow & p(a\,w), \mathit{rest} \\
p(a\,X) & \leftarrow & p_1(X), \ldots, p_r(X) \\
 & & \text{implies:} \\
\mathit{head} & \leftarrow & p_1(w), \ldots, p_r(w), \mathit{rest}
\end{array}
$$

$$
\begin{array}{lcl}
p(X) & \leftarrow & p_1(X), \ldots, p_m(X) \\
p_i(a\,X) & \leftarrow & p_{i1}(X), \ldots, p_{ir_i}(X) \\
 & & \text{implies:} \\
p(a\,X) & \leftarrow & p_{11}(X), \ldots, p_{mr_m}(X)
\end{array}
$$

Step 3 (Cont.): Normalization

$$\begin{array}{lcl} head & \leftarrow & p(w), rest \\ p(X) & \leftarrow & \qquad \text{implies:} \\ head & \leftarrow & rest \end{array}$$

$$\begin{array}{lcl} head & \leftarrow & p(b), rest \\ p(b) & \leftarrow & \qquad \text{implies:} \\ head & \leftarrow & rest \end{array}$$

$$\begin{array}{lcl} p() & \leftarrow & p_1(X), \ldots, p_m(X) \\ p_i(a\,X) & \leftarrow & p_{i1}(X), \ldots, p_{ir_i}(X) \\ & & \qquad \text{implies:} \\ p() & \leftarrow & p_{11}(X), \ldots, p_{mr_m}(X) \end{array}$$

Example:

$$\begin{array}{lcl}
\mathsf{add}_1(X) & \leftarrow & \mathsf{add}_0(X) \\
\mathsf{add}_0(0) & \leftarrow & \\
\mathsf{add}_1(X) & \leftarrow & \mathsf{add}_1(X) \\
\mathsf{add}_1(s_1\ X) & \leftarrow & \mathsf{add}_1(X)
\end{array}$$

... results in the new clause:

$$\mathsf{add}_1(0) \quad \leftarrow$$

Theorem

Assume that $\mathcal{C}$ is a finite set of clauses for which steps 1 and 2 have been executed and which then has been saturated according to step 3.

Assume that $\mathcal{C}_0 \subseteq \mathcal{C}$ is the subset of normal clauses of $\mathcal{C}$. Then for all occurring predicates p,

$$[\![p]\!]_{\mathcal{C}_0} = [\![p]\!]_{\mathcal{C}}$$

Proof:

Induction on the depth of terms in $[\![p]\!]_{\mathcal{C}}$:-)

... in the Example:

For $\quad \mathsf{add}_1(X) \quad$ we obtain the following clauses:

$$\begin{array}{lll} \mathsf{add}_1(0) & \leftarrow & \\ \mathsf{add}_1(s_1\; X) & \leftarrow & \mathsf{add}_1(X) \end{array}$$

These clauses are already normal :-)

Transforming into Normal Clauses:

Introduce new predicates for conjunctions of predicates.

Assume that $A = \{p_1, \ldots, p_m\}$. Then:

$[A](b) \leftarrow$ whenever $p_i(b) \leftarrow$ for all i.

$[A](a\,X) \leftarrow [B](X)$ whenever $B = \{p_{ij} \mid i = 1, \ldots, m\}$ for $p_i(a\,X) \leftarrow p_{i1}(X), \ldots, p_{ir_i}(X)$

Last Step: Transformation into a Type

- First, the automaton is determinized ...

Last Step: Transformation into a Type

- First, the automaton is determinized ...

- Then transitions for the components of constructors a:

$$p(a_j\, X) \leftarrow p^{(j)}(X)$$

  are joined into a transition for a:

$$p(a(X_1, \ldots, X_k)) \leftarrow p^{(1)}(X_1), \ldots, p^{(k)}(X_k)$$

- Finally, the predicates p_j for the components of the predicate p/k are joined to a transition:

$$p(X_1, \ldots, X_k) \leftarrow p_1(X_1), \ldots, p_k(X_k)$$

In the Example we find:

$$
\begin{array}{lcll}
\mathsf{add}(X, Y, Z) & \leftarrow & \mathsf{add}_1(X), \mathsf{nat}(Y), q'(Z) & \quad \text{where} \\
q'(0) & \leftarrow & & \\
q'(s\ X) & \leftarrow & q'(X) & \\
q' & = & \{\mathsf{nat}, \mathsf{add}_2\} &
\end{array}
$$

In the Example we find:

$$
\begin{array}{lcll}
\mathsf{add}(X, Y, Z) & \leftarrow & \mathsf{add}_1(X), \mathsf{nat}(Y), q'(Z) & \quad \text{where} \\
q'(0) & \leftarrow & & \\
q'(s\ X) & \leftarrow & q'(X) & \\
q' & = & \{\mathsf{nat}, \mathsf{add}_2\} &
\end{array}
$$

The types $\mathsf{add}_1, q', \mathsf{nat}$ are all equivalent :-)

Discussion:

- For type-checking, it suffices to check for every predicate p/k that

$$[\![p_i]\!]_{\mathcal{C}^\sharp} \subseteq \Pi(T_i)$$

- Since the T_i are topdown deterministic, we have a deterministic automaton for $\Pi(T_i)$:-)
- Therefore, we can easily construct a DFA for the complement $\overline{\Pi(T_i)}$!!
- Then we check whether

$$[\![p_i]\!]_{\mathcal{C}^\sharp} \cap \overline{\Pi(T_i)} = \emptyset$$

$\Longrightarrow$ this saves us determinization :-))

Warning:

- The emptiness problem for APS is DEXPTIME-complete !
- In many cases, though, our method terminates quickly ;-)

Warning:

- The emptiness problem for APS is DEXPTIME-complete !
- In many cases, though, our method terminates quickly ;-)

- Inferred types can also be used to understand legacy code.
- Then, however, they are only useful if they are not too complicated !
- Our type inference provides very precise information :-)
- In practical applications, further widenings are applied to accelerate the analysis, e.g., by reducing the number of occurring sets.

5.3 Goal-directed Type Inference

Prolog programs explore predicates only insofar as they contribute to answer a query.

Example: append

$$
\begin{array}{lll}
\mathsf{app}([\,], Y, Y) & \leftarrow & \\
\mathsf{app}([H|T], Y, [H|Z]) & \leftarrow & \mathsf{app}(T, Y, Z) \\
 & \leftarrow & \mathsf{app}([1,2], [3], Z)
\end{array}
$$

... results in:

The APS-Approximation

$$
\begin{array}{lcl}
\mathsf{app}_1([|]_1(H)) & \leftarrow & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z). \\
\mathsf{app}_1([|]_2(T)) & \leftarrow & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z). \\
\mathsf{app}_2(Y) & \leftarrow & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z). \\
\mathsf{app}_3([|]_1(H)) & \leftarrow & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z). \\
\mathsf{app}_3([|]_2(Z)) & \leftarrow & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z). \\
\mathsf{app}_1([]) & \leftarrow & \\
\mathsf{app}_2(X) & \leftarrow & \\
\mathsf{app}_3(X)) & \leftarrow & \\
 & \leftarrow & \mathsf{app}_1([|]_1(1)), \mathsf{app}_1([|]_2([|]_1(2))), \mathsf{app}_1([|]_2([|]_2([]))), \\
 & & \mathsf{app}_2([|]_1(3)), \mathsf{app}_2([|]_2([])), \mathsf{app}_3(X)
\end{array}
$$

Ignoring the query, we find via normalization:

$$
\begin{array}{lcl}
\mathsf{app}_2(X) & \leftarrow & \\
\mathsf{app}_3(X) & \leftarrow & \\
\mathsf{app}_1([]) & \leftarrow & \\
\mathsf{app}_1([|]_2 X) & \leftarrow & q_0(X) \\
\mathsf{app}_1([|]_2 X) & \leftarrow & q_1(X) \\
\mathsf{app}_1([|]_2 X) & \leftarrow & q_2(X) \\
\mathsf{app}_1([|]_1 X) & \leftarrow & \\
q_0([]) & \leftarrow & \\
q_1([|]_2 X) & \leftarrow & q_0(X) \\
q_1([|]_2 X) & \leftarrow & q_1(X) \\
q_1([|]_2 X) & \leftarrow & q_2(X) \\
q_2([|]_1 X) & \leftarrow &
\end{array}
$$

Discussion

- The second and third argument can be arbitrary.
- The first argument is a list where nothing is known about the elements :-)
- Ignoring the query, this result is the best we can hope for :-(
- Better results can be obtained if additionally call patterns are tracked !

$\Longrightarrow$ Magic Set Transformation

Magic Sets

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

$$\mathsf{called}_p(\underline{t}) \leftarrow \quad \text{for the query} \quad \leftarrow \mathsf{p}(\underline{t})$$

-

$$\mathsf{called}_{p_i}(\underline{t_i}) \leftarrow \mathsf{called}_p(\underline{t}), \mathsf{p}_1(\underline{t_1}), \ldots, \mathsf{p}_{i-1}(\underline{t_{i-1}})$$

$$p_i(\underline{t}) \leftarrow \mathsf{called}_p(\underline{t}), \mathsf{p}_1(\underline{t_1}), \ldots, \mathsf{p}_m(\underline{t_m})$$

for every clause:

$$\mathsf{p}(\underline{t}) \leftarrow \mathsf{p}_1(\underline{t_1}), \ldots, \mathsf{p}_m(\underline{t_m})$$

Example: append (Cont.)

$$
\begin{array}{lcl}
\mathsf{app}([\,], Y, Y) & \leftarrow & \mathsf{called}([\,], Y, Y) \\
\mathsf{app}([H|T], Y, [H|Z]) & \leftarrow & \mathsf{called}([H|T], Y, [H|Z]), \\
 & & \mathsf{app}(T, Y, Z) \\
\mathsf{called}(T, Y, Z) & \leftarrow & \mathsf{called}([H|T], Y, [H|Z]) \\
\mathsf{called}([1, 2], [3], Z) & \leftarrow &
\end{array}
$$

The APS-Approximation:

$$
\begin{array}{lcl}
\mathsf{app}_1([\,]) & \leftarrow & \mathsf{called}_1([\,]), \mathsf{called}_2(X), \mathsf{called}_3(X) \\
\mathsf{app}_2(X) & \leftarrow & \mathsf{called}_1([\,]), \mathsf{called}_2(X), \mathsf{called}_3(X) \\
\mathsf{app}_3(X) & \leftarrow & \mathsf{called}_1([\,]), \mathsf{called}_2(X), \mathsf{called}_3(X) \\
\mathsf{app}_1([|]_1 H) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z), \\
 & & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z) \\
\mathsf{app}_1([|]_2 T) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z), \\
 & & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z) \\
\mathsf{app}_2(Y) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z), \\
 & & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z) \\
\mathsf{app}_3([|]_1 H) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z), \\
 & & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z) \\
\mathsf{app}_3([|]_2 Z) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z), \\
 & & \mathsf{app}_1(T), \mathsf{app}_2(Y), \mathsf{app}_3(Z)
\end{array}
$$

$$
\begin{array}{lcl}
 & \cdots & \\
\mathsf{called}_1(T) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z) \\
\mathsf{called}_2(Y) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z) \\
\mathsf{called}_3(Z) & \leftarrow & \mathsf{called}_1([|]_1 H), \mathsf{called}_1([|]_2 T), \mathsf{called}_2(Y), \mathsf{called}_3([|]_1 H), \mathsf{called}_3([|]_2 Z) \\
\mathsf{called}_1([|]_1 1) & \leftarrow & \\
\mathsf{called}_1([|]_2 ([|]_1 2) & \leftarrow & \\
\mathsf{called}_1([|]_2 [|]_2 []) & \leftarrow & \\
\mathsf{called}_2([|]_1 3) & \leftarrow & \\
\mathsf{called}_2([|]_2 []) & \leftarrow & \\
\mathsf{called}_3(X) & \leftarrow &
\end{array}
$$

The Normalized APS-Approximation (Cont.)

$$
\begin{array}{lcl}
\mathsf{app}_1([|]_1 X) & \leftarrow & q_1(X) \\
\mathsf{app}_1([|]_1 X) & \leftarrow & q_2(X) \\
\mathsf{app}_1([]) & \leftarrow & \\
\mathsf{app}_1([|]_2 X) & \leftarrow & q_4(X) \\
\mathsf{app}_1([|]_2 X) & \leftarrow & q_0(X) \\
\mathsf{app}_1([|]_2 X) & \leftarrow & q_5(X) \\
\mathsf{app}_2([|]_1 X) & \leftarrow & q_3(X) \\
\mathsf{app}_2([|]_2 X) & \leftarrow & q_0(X) \\
\mathsf{app}_3([|]_1 X) & \leftarrow & q_1(X) \\
\mathsf{app}_3([|]_1 X) & \leftarrow & q_2(X)
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{app}_3([|]_1 X) & \leftarrow & q_3(X) \\
\mathsf{app}_3([|]_2 X) & \leftarrow & q_0(X) \\
\mathsf{app}_3([|]_2 X) & \leftarrow & q_4(X) \\
\mathsf{app}_3([|]_2 X) & \leftarrow & q_6(X) \\
\mathsf{app}_3([|]_2 X) & \leftarrow & q_7(X) \\
\mathsf{app}_3([|]_2 X) & \leftarrow & q_8(X) \\
q_0([]) & \leftarrow & \\
q_1(1) & \leftarrow & \\
q_2(2) & \leftarrow & \\
q_3(3) & \leftarrow &
\end{array}
$$

$$
\begin{array}{lcl}
q_4([|]_2 X) & \leftarrow & q_0(X) \\
q_5([|]_1 X) & \leftarrow & q_2(X) \\
q_6([|]_1 X) & \leftarrow & q_3(X) \\
q_7([|]_1 X) & \leftarrow & q_1(X) \\
q_7([|]_1 X) & \leftarrow & q_2(X) \\
q_8([|]_2 X) & \leftarrow & q_4(X) \\
q_8([|]_2 X) & \leftarrow & q_7(X) \\
q_8([|]_2 X) & \leftarrow & q_8(X) \\
q_8([|]_2 X) & \leftarrow & q_6(X)
\end{array}
$$

Discussion

- The result now is amazingly precise !!
- The correct values for the second parameter is inferred.
- For the result parameter, a list containing 1,2 and 3 is inferred.

- It only fails to infer that this list is finite and of length 3 :-)