Script generated by TTT

Title: Seidl: Programmoptimierung (06.02.2012)

Date: Mon Feb 06 12:34:05 CET 2012

Duration: 54:57 min

Pages: 29

Step 1: Removal of complicated heads:

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

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

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

Step 2: Splitting

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

$$head \leftarrow rest, \boxed{p_1(w_1\,X), \ldots, p_m(w_m\,X)}$$

$$(X \text{ does not occur in } head,\ rest)$$

is replaced with:

$$\cancel{head \leftarrow rest, q()}$$

$$q() \leftarrow p_1(w_1\,X), \ldots, p_m(w_m\,X)$$

for a new predicate $q/0$.

Step 3: Normalization

We add simpler derived clauses:

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

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

Step 3 (Cont.): Normalization

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

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

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

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:

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

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:

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

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 $\mathsf{add}_1(X)$ 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:

$$\begin{array}{ll} [A](b) \leftarrow & \text{whenever} \quad p_i(b) \leftarrow \quad \text{for all } i. \\ [A](a\,X) \leftarrow [B](X) & \text{whenever} \quad B = \{p_{ij} \mid i = 1, \ldots, m\} \quad \text{for} \\ & p_i(a\,X) \leftarrow p_{i1}(X), \ldots, p_{ir_i}(X) \end{array}$$

Last Step: Transformation into a Type

- First, the automaton is determinized ...

In the Example we find:

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

Discussion:

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

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

- 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

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

$\Longrightarrow$ 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

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

... results in:

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

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

Ignoring the query, we find via normalization:

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

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

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
-
$$\begin{array}{rcl} \mathsf{called}_p(\underline{t}) & \leftarrow & \text{for the query} \quad \leftarrow \mathsf{p}(\underline{t}) \\ \mathsf{called}_{p_i}(\underline{t_i}) & \leftarrow & \mathsf{called}_p(\underline{t}), \mathsf{p}_1(\underline{t_1}), \ldots, \mathsf{p}_{i-1}(\underline{t_{i-1}}) \\ p(\underline{t}) & \leftarrow & \mathsf{called}_p(\underline{t}), \mathsf{p}_1(\underline{t_1}), \ldots, \mathsf{p}_m(\underline{t_m}) \end{array}$$

for every clause:

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

Example: append (Cont.)

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

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

The Normalized APS-Approximation (Cont.)

app [] 1,2

$\mathsf{app}_1([\|]_1X)$	$\leftarrow$	$q_1(X)$	$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_3(X)$	$q_4([\|]_2X)$	$\leftarrow$	$q_0(X)$
$\mathsf{app}_1([\|]_1X)$	$\leftarrow$	$q_2(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_0(X)$	$q_5([\|]_1X)$	$\leftarrow$	$q_2(X)$
$\mathsf{app}_1([])$	$\leftarrow$		$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_4(X)$	$q_6([\|]_1X)$	$\leftarrow$	$q_3(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_4(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_6(X)$	$q_7([\|]_1X)$	$\leftarrow$	$q_1(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_0(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_7(X)$	$q_7([\|]_1X)$	$\leftarrow$	$q_2(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_5(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_8(X)$	$q_8([\|]_2X)$	$\leftarrow$	$q_4(X)$
$\mathsf{app}_2([\|]_1X)$	$\leftarrow$	$q_3(X)$	$q_0([])$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_7(X)$
$\mathsf{app}_2([\|]_2X)$	$\leftarrow$	$q_0(X)$	$q_1(1)$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_8(X)$
$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_1(X)$	$q_2(2)$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_6(X)$
$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_2(X)$	$q_3(3)$	$\leftarrow$				

The Normalized APS-Approximation (Cont.)

$\mathsf{app}_1([\|]_1X)$	$\leftarrow$	$q_1(X)$	$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_3(X)$	$q_4([\|]_2X)$	$\leftarrow$	$q_0(X)$
$\mathsf{app}_1([\|]_1X)$	$\leftarrow$	$q_2(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_0(X)$	$q_5([\|]_1X)$	$\leftarrow$	$q_2(X)$
$\mathsf{app}_1([])$	$\leftarrow$		$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_4(X)$	$q_6([\|]_1X)$	$\leftarrow$	$q_3(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_4(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_6(X)$	$q_7([\|]_1X)$	$\leftarrow$	$q_1(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_0(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_7(X)$	$q_7([\|]_1X)$	$\leftarrow$	$q_2(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_5(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_8(X)$	$q_8([\|]_2X)$	$\leftarrow$	$q_4(X)$
$\mathsf{app}_2([\|]_1X)$	$\leftarrow$	$q_3(X)$	$q_0([])$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_7(X)$
$\mathsf{app}_2([\|]_2X)$	$\leftarrow$	$q_0(X)$	$q_1(1)$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_8(X)$
$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_1(X)$	$q_2(2)$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_6(X)$
$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_2(X)$	$q_3(3)$	$\leftarrow$				

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 :-)

The Normalized APS-Approximation (Cont.)

$\mathsf{app}_1([\|]_1X)$	$\leftarrow$	$q_1(X)$	$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_3(X)$	$q_4([\|]_2X)$	$\leftarrow$	$q_0(X)$
$\mathsf{app}_1([\|]_1X)$	$\leftarrow$	$q_2(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_0(X)$	$q_5([\|]_1X)$	$\leftarrow$	$q_2(X)$
$\mathsf{app}_1([])$	$\leftarrow$		$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_4(X)$	$q_6([\|]_1X)$	$\leftarrow$	$q_3(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_4(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_6(X)$	$q_7([\|]_1X)$	$\leftarrow$	$q_1(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_0(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_7(X)$	$q_7([\|]_1X)$	$\leftarrow$	$q_2(X)$
$\mathsf{app}_1([\|]_2X)$	$\leftarrow$	$q_5(X)$	$\mathsf{app}_3([\|]_2X)$	$\leftarrow$	$q_8(X)$	$q_8([\|]_2X)$	$\leftarrow$	$q_4(X)$
$\mathsf{app}_2([\|]_1X)$	$\leftarrow$	$q_3(X)$	$q_0([])$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_7(X)$
$\mathsf{app}_2([\|]_2X)$	$\leftarrow$	$q_0(X)$	$q_1(1)$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_8(X)$
$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_1(X)$	$q_2(2)$	$\leftarrow$		$q_8([\|]_2X)$	$\leftarrow$	$q_6(X)$
$\mathsf{app}_3([\|]_1X)$	$\leftarrow$	$q_2(X)$	$q_3(3)$	$\leftarrow$				

Perspective: Normal Horn Clauses

- Prolog may no longer be the sexiest programming language :-)
- Horn clauses, though, are very well suited for the specification of analysis problems.
- It is a separate problem then to solve the stated analysis problem :-)
- If the least solution cannot be computed exactly, approximate solutions may at least yield approximative answers ...

Example: Cryptographic Protocols

Perspective: Normal Horn Clauses

- Prolog may no longer be the sexiest programming language :-)
- Horn clauses, though, are very well suited for the specification of analysis problems.
- It is a separate problem then to solve the stated analysis problem :-)
- If the least solution cannot be computed exactly, approximate solutions may at least yield approximative answers ...

Example: Cryptographic Protocols

945