Step 1: Removal of complicated heads:

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

\[
\begin{align*}
  p(w, X) & \leftarrow \text{rhs} & \text{with:} \\
  p(a^{(1)} X) & \leftarrow p_2(X) \\
  p_2(a^{(2)} X) & \leftarrow p_3(X) \\
  \vdots \\
  p_{m-1}(a^{(m-1)} X) & \leftarrow p_m(X) \\
  p_m(a^{(m)} X) & \leftarrow \text{rhs} \\
  \end{align*}
\]

// \( p_j \) all new

Step 2: Splitting

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

\[
\begin{align*}
  \text{head} & \leftarrow \text{rest} \quad p_1(w_1 X, \ldots, p_m(w_m X) \\
            \quad \text{(X does not occur in } \text{head}, \text{rest)}
\end{align*}
\]

is replaced with:

\[
\begin{align*}
  \text{head} & \leftarrow \text{rest} \quad q() \\
  q() & \leftarrow p_1(w_1 X, \ldots, p_m(w_m X)
\end{align*}
\]

for a new predicate \( q / 0 \).
Step 3 (Cont.): Normalization

\[ \text{head} \leftarrow p(w), \text{rest} \]
\[ p(X) \leftarrow \text{implies:} \]
\[ \text{head} \leftarrow \text{rest} \]
\[ p(b), \text{rest} \]
\[ p(b) \leftarrow \text{implies:} \]
\[ \text{head} \leftarrow \text{rest} \]
\[ p() \leftarrow p_1(X), \ldots, p_m(X) \]
\[ p_b(a, X) \leftarrow p_{11}(X), \ldots, p_{r}(X) \]
\[ \text{implies:} \]
\[ p() \leftarrow p_{11}(X), \ldots, p_{n+r}(X) \]

Example:

\[ \text{add}_1(X) \leftarrow \text{add}_0(X) \]
\[ \text{add}_0(0) \leftarrow \text{add}_1(X) \]
\[ \text{add}_1(X) \leftarrow \text{add}_1(X) \]
\[ \text{add}_1(s_1, X) \leftarrow \text{add}_1(X) \]

... results in the new clause:

\[ \text{add}_1(0) \leftarrow \]

Theorem

Assume that \( C \) 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 \( C_0 \subseteq C \) is the subset of normal clauses of \( C \). Then for all occurring predicates \( p \),

\[ [p]_{C_0} = [p]_C \]

Proof:
Induction on the depth of terms in \( [p]_C \)
\[ :-) \]
... in the Example:

For \texttt{add}_1(X) \text{ we obtain the following clauses:}

\begin{align*}
\texttt{add}_1(0) & \leftarrow \\
\texttt{add}_1(s_1, X) & \leftarrow \texttt{add}_1(X)
\end{align*}

These clauses are already normal. 

Transforming into Normal Clauses:

Introduce new predicates for \texttt{conjunctions} of predicates.

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

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

Last Step: Transformation into a Type

- First, the automaton is determinized ...

In the Example we find:

\begin{align*}
\texttt{add}(X, Y, Z) & \leftarrow \texttt{add}_1(X), \texttt{nat}(X), \texttt{add}_2(Z) \\
\texttt{add}'(0) & \leftarrow \\
\texttt{add}'(a X) & \leftarrow \texttt{add}'(X) \\
\texttt{add}' & = \{\texttt{nat, add}_2\}
\end{align*}

where
Discussion:

- For type-checking, it suffices to check for every predicate \( p_i / k \) that
  \[
  [p_i] \subseteq \Pi(T_i)
  \]

- Since the \( T_i \) are topdown deterministic, we have a deterministic
  automaton for \( \Pi(T_i) \)  \( \vdash \)

- Therefore, we can easily construct a DFA for the complement
  \( \Pi(T_i) \)  \( \Leftrightarrow \)

- Then we check whether
  \[
  ([p_i] \cap \Pi(T_i)) = \emptyset
  \]
  \( \implies \) this saves us determinization  \( \vdash \)

Warning:

- The emptiness problem for APS is \textit{DEXPTIME-complete} !
- In many cases, though, our method terminates quickly  \( \vdash \)

5.3 Goal-directed Type Inference

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

Example: \texttt{append}

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

... results in:

\[
[1, 2, 3, Z]
\]
The \textit{APS}-Approximation

\begin{align*}
\text{app}_1([[], (H)]) & \leftarrow \text{app}_1(T), \text{app}_2(Y), \text{app}_3(Z).
\text{app}_1([[], (T)]) & \leftarrow \text{app}_1(T), \text{app}_2(Y), \text{app}_3(Z).
\text{app}_2(Y) & \leftarrow \text{app}_1(T), \text{app}_2(Y), \text{app}_3(Z).
\text{app}_3([[], (H)]) & \leftarrow \text{app}_1(T), \text{app}_2(Y), \text{app}_3(Z).
\text{app}_3([[], (Z)]) & \leftarrow \text{app}_1(T), \text{app}_2(Y), \text{app}_3(Z).
\text{app}_1([]) & \leftarrow \\
\text{app}_2(X) & \leftarrow \\
\text{app}_3(X) & \leftarrow \\
& \leftarrow \text{app}_1([], (1)), \text{app}_1([], (1), (2)), \text{app}_1([], (1), ([], [])), \\
& \phantom{\leftarrow} \text{app}_2([], (3)), \text{app}_2([], (3)), \text{app}_3(X)
\end{align*}

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!

\[ \text{Magic Set Transformation} \]

5.3 Goal-directed Type Inference

Ignoring the query, we find via normalization:

\begin{align*}
\text{app}_1(X) & \leftarrow \\
\text{app}_2(X) & \leftarrow \\
\text{app}_1([]) & \leftarrow \\
\text{app}_1([], X) & \leftarrow q_0(X)
\end{align*}

Example: \texttt{append}

\begin{align*}
\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)
\end{align*}

... results in:
Discussion

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

\[ \Rightarrow \text{Magic Set Transformation} \]

---

**Example: append (Cont.)**

\[
\begin{align*}
\text{app}([], Y, Y) & \leftarrow \text{called}([], Y, Y) \\
\text{app}([H | T], Y, [H | Z]) & \leftarrow \text{called}([H | T], Y, [H | Z]), \\
& \quad \text{app}(T, Y, Z) \\
\text{called}(T, Y, Z) & \leftarrow \text{called}([H | T], Y, [H | Z]) \\
\text{called}([1, 2], [3], Z) & \leftarrow \\
\end{align*}
\]

---

**Magic Sets**

- For every predicate \( p / k \), we introduce a new predicate \( \text{called}_{p/k} \) with the clauses:

\[
\begin{align*}
\text{called}_{p/k}(t) & \leftarrow \text{for the query} \leftarrow p(t) \\
\text{called}_{p/k}(t_1, \ldots, t_{n-1}) & \leftarrow \text{called}_{p/k}(t, p_1(t_1), \ldots, p_m(t_m)) \\
\end{align*}
\]

for every clause:

\[
p(t) \leftarrow p_1(t_1), \ldots, p_m(t_m)
\]

---

\[
\begin{align*}
\text{called}_1(T) & \leftarrow \text{called}_1([], H), \text{called}_1([], 2), \text{called}_2(Y), \text{called}_3(T), \text{called}_4([], H), \text{called}_5([], Z) \\
\text{called}_2(Y) & \leftarrow \text{called}_1([], H), \text{called}_1([], 2), \text{called}_2(Y), \text{called}_3(T), \text{called}_4([], H), \text{called}_5([], Z) \\
\text{called}_3(Z) & \leftarrow \text{called}_1([], H), \text{called}_1([], 2), \text{called}_2(Y), \text{called}_3(T), \text{called}_4([], H), \text{called}_5([], Z) \\
\text{called}_4([], 1) & \leftarrow \text{called}_1([], H), \text{called}_1([], 2), \text{called}_2(Y), \text{called}_3(T), \text{called}_4([], H), \text{called}_5([], Z) \\
\text{called}_5([], 1) & \leftarrow \text{called}_1([], H), \text{called}_1([], 2), \text{called}_2(Y), \text{called}_3(T), \text{called}_4([], H), \text{called}_5([], Z) \\
\end{align*}
\]
The Normalized $APS$-Approximation (Cont.)

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 :-(
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