5.1 Types for Prolog

Example:

\[
\begin{align*}
\text{nat}(X) & \leftarrow X = 0 \\
\text{nat}(X) & \leftarrow X = s(Y), \text{nat}(Y) \\
\text{nat_list}(X) & \leftarrow X = [] \\
\text{nat_list}(X) & \leftarrow X = [H|T], \text{nat}(H), \text{nat_list}(T)
\end{align*}
\]

Discussion

- In Prolog, a type is a set of ground terms with a simple description.
- There is no common agreement what simple means.
- One possibility are (non-deterministic) finite tree automata or normal Horn clauses:

\[
\begin{align*}
\text{nat_list}([H|T]) & \leftarrow \text{nat}(H), \text{nat_list}(T) & \text{normal} \\
\text{bin}(\text{node}(T, T)) & \leftarrow \text{bin}(T) & \text{nicht normal} \\
\text{tree}(\text{node}(T_1, T_2)) & \leftarrow \text{tree}(T_1), \text{tree}(T_2) & \text{normal}
\end{align*}
\]
Comparison:

<table>
<thead>
<tr>
<th>Normal clauses</th>
<th>Tree automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>unary predicate</td>
<td>state</td>
</tr>
<tr>
<td>normal clause</td>
<td>transition</td>
</tr>
<tr>
<td>constructor in the head body</td>
<td>input symbol</td>
</tr>
<tr>
<td>body</td>
<td>pre-condition</td>
</tr>
</tbody>
</table>

General Form:

\[
p(a(X_1, \ldots, X_k)) \leftarrow p_1(X_1), \ldots, p_k(X_k)
\]

\[
p(X) \leftarrow
\]

\[
p(b) \leftarrow
\]

Properties:

- Types then are in fact regular tree languages :-)  
- Types are closed under intersection:
  \[
  (p \cdot q)(a(X_1, \ldots, X_k)) \leftarrow (p_1 \cdot q_1)(X_1), \ldots, (p_k \cdot q_k)(X_k)
  \]
  if 
  \[
  p(a(X_1, \ldots, X_k)) \leftarrow p_1(X_1), \ldots, p_k(X_k)
  \]
  \[
  q(a(X_1, \ldots, X_k)) \leftarrow q_1(X_1), \ldots, q_k(X_k)
  \]
- Types are also closed under union :-)
- Queries \(p(X)\) and \(p(i)\) can be decided in polynomial time but:
- ... only in presence of tabulation!
- Or the program is topdown deterministic ...

Example: Topdown vs. Bottom-up

\[
p(a(X_1, X_2)) \leftarrow p_1(X_1), p_2(X_2)
\]

\[
p(G(X_1, X_2)) \leftarrow p_2(X_1), p_1(X_2)
\]

\[
p(b) \leftarrow
\]

... is bottom-up, but not topdown deterministic.

There is no topdown deterministic program for this type!

\[\Rightarrow\]

Topdown deterministic types are closed under intersection, but not under union !!!
Example: Topdown vs. Bottom-up

\[ p(a(X_1, X_2)) \leftarrow p_1(X_1), p_2(X_2) \]
\[ p(a(X_1, X_2)) \leftarrow p_2(X_1), p_1(X_2) \]
\[ p_1(b) \leftarrow \]
\[ p_2(c) \leftarrow \]

... is bottom-up, but not topdown deterministic.
There is no topdown deterministic program for this type !

Topdown deterministic types are closed under intersection, but not under union !!!!

Vice versa from a set \( P \) of paths, a set \( \Pi^{-}(P) \) of terms can be recovered:

\[ \Pi^{-}(P) = \{ t | \Pi(t) \subseteq P \} \]

Example (Cont.):

\[ P = \{ a_1b, a_2c, a_1c, a_2b \} \]
\[ \Pi^{-}(P) = \{ a(b, b), a(b, c), a(c, b), a(c, c) \} \]

The set has become larger !!!

For a set \( T \) of terms, we define the set \( \Pi(T) \) of paths in terms from \( T \):

\[ \Pi(T) = \bigcup \{ \Pi(t) | t \in T \} \]
\[ \Pi(b) = \{ b \} \]
\[ \Pi(a(t_1, \ldots, t_k)) = \{ a_jw | w \in \Pi(t_j) \} \quad (k > 0) \]

// for new unary constructors \( a_j \)

Example

\[ T = \{ a(b, c), a(c, b) \} \]
\[ \Pi(T) = \{ a_1b, a_2c, a_1c, a_2b \} \]

For a set \( T \) of terms, we define the set \( \Pi(T) \) of paths in terms from \( T \):

\[ \Pi(T) = \bigcup \{ \Pi(t) | t \in T \} \]
\[ \Pi(b) = \{ b \} \]
\[ \Pi(a(t_1, \ldots, t_k)) = \{ a_jw | w \in \Pi(t_j) \} \quad (k > 0) \]

// for new unary constructors \( a_j \)

Example

\[ T = \{ a(b, c), a(c, b) \} \]
\[ \Pi(T) = \{ a_1b, a_2c, a_1c, a_2b \} \]
Vice versa from a set \( P \) of paths, a set \( \Pi^-(P) \) of terms can be recovered:

\[
\Pi^-(P) = \{ t \mid \Pi(t) \subseteq P \}
\]

**Example (Cont.):**

\[
P = \{ a_1b, a_2c, a_1c, a_2b \}
\]

\[
\Pi^-(P) = \{ a(b, b), a(b, c), a(c, b), a(c, c) \}
\]

The set has become larger !!

---

Vice versa from a set \( P \) of paths, a set \( \Pi^-(P) \) of terms can be recovered:

\[
\Pi^-(P) = \{ t \mid \Pi(t) \subseteq P \}
\]

**Example (Cont.):**

\[
P = \{ a_1b, a_2c, a_1c, a_2b \}
\]

\[
\Pi^-(P) = \{ a(b, b), a(b, c), a(c, b), a(c, c) \}
\]

The set has become larger !!

---

**Theorem:**

Assume that \( T \) is a regular set of terms. Then:

- \( \Pi(T) \) is regular \( :- \)
- \( T \subseteq \Pi^-(\Pi(T)) \) \( :- \)
- \( T = \Pi^-(\Pi(T)) \) iff \( T \) is todown deterministic \( :- \)
- \( \Pi^-(\Pi(T)) \) is the smallest superset of \( T \) which is todown deterministic. \( :- \)

**Consequence:**

If we are interested in todown deterministic types, it suffices to determine the set of paths in terms !!!

---

For a set \( T \) of terms, we define the set \( \Pi(T) \) of paths in terms from \( T \):

\[
\Pi(T) = \bigcup \{ \Pi(t) \mid t \in T \}
\]

\[
\Pi(b) = \{ b \}
\]

\[
\Pi(a(t_1, \ldots, t_k)) = \{ a_jw \mid w \in \Pi(t_j) \} \quad (k > 0)
\]

// for new unary constructors \( a_j \)

**Example**

\[
T = \{ a(b, c), a(c, b) \}
\]

\[
\Pi(T) = \{ a_1b, a_2c, a_1c, a_2b \}
\]

---

---

---
Theorem:

Assume that $T$ is a regular set of terms. Then:

- $\Pi(T)$ is regular $\implies$
- $T \subseteq \Pi^-(\Pi(T))$ $\implies$
- $T = \Pi^-(\Pi(T))$ if $T$ is topos down deterministic $\iff$
- $\Pi^-(\Pi(T))$ is the smallest superset of $T$ which is topos down deterministic. $\implies$

Consequence:

If we are interested in topos down deterministic types, it suffices to determine the set of paths in terms $!!!$

Example (Cont.):

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{add}(X, Y, Z)$</td>
<td>$X = 0, \text{nat}(Y), Y = Z$</td>
</tr>
<tr>
<td>$\text{add}(X, Y, Z)$</td>
<td>$\text{nat}(X), X = s(X'), Z = s(Z'), \text{add}(X', Y, Z')$</td>
</tr>
<tr>
<td>$\text{mult}(X, Y, Z)$</td>
<td>$X = 0, \text{nat}(Y), Z = 0$</td>
</tr>
<tr>
<td>$\text{mult}(X, Y, Z)$</td>
<td>$\text{nat}(X), X = s(X'), \text{mult}(X', Y, Z'), \text{add}(Z', Y, Z)$</td>
</tr>
</tbody>
</table>

Question:

Which run-time checks are necessary?

Semantics:

Let $C$ denote a set of clauses.

The set $[p]_C$ is the set of tuples of ground terms $(s_1, \ldots, s_k)$, for which $p(s_1, \ldots, s_k)$ is provable $\implies$

$[p]_C$ (p predicate) thus is the smallest collection of sets of tuples for which:

$\sigma(t) \in [p]_C$ when ever $\forall i. \sigma(t_i) \in [p_i]_C$

for clauses $p(t) \leftarrow p_1(t_1), \ldots, p_n(t_n) \in C$ and ground substitutions $\sigma$. 

Idea:

- Approximate the semantics of predicates by means of topos down-deterministic regular tree languages $!$
- Alternatively: Approximate the set of paths in the semantics of predicates by regular word languages $!$

Idea:

- All predicates $p/k, k > 0$, are split into predicates $p_1/1, \ldots, p_n/1$. 

907

908

909

910
Semantics:

Let \( \mathcal{C} \) denote a set of clauses.

The set \([\mathcal{P}]_\mathcal{C}\) is the set of tuples of ground terms \((s_1, \ldots, s_k)\), for which \(\mathcal{P}(s_1, \ldots, s_k)\) is provable \(\therefore\)

\([\mathcal{P}]_\mathcal{C}\) (\(\mathcal{P}\) predicate) thus is the smallest collection of sets of tuples for which:

\[\sigma(t) \in [\mathcal{P}]_\mathcal{C} \quad \text{when ever} \quad \forall i, \sigma(t_i) \in [\mathcal{P}]_\mathcal{C}\]

for clauses \(\mathcal{P}(t) \leftarrow \mathcal{P}_1(t_1) \ldots \mathcal{P}_n(t_n) \in \mathcal{C}\) and ground substitutions \(\sigma\).

Approximation of Paths:

Every clause

\(\mathcal{P}(t_1, \ldots, t_k) \leftarrow \alpha\)

is approximated by the clauses:

\(\mathcal{P}_j(w) \leftarrow \bigwedge \Pi(\alpha)\) where

\(\Pi(g_1, \ldots, g_m) = \Pi(g_1) \cup \ldots \cup \Pi(g_m)\)

\(\Pi(q(s_1, \ldots, s_n)) = \{q_l(w) \mid w \in \Pi(s_i)\}\)

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

Example:

\[
\begin{align*}
\text{add}(0, Y, Y) & \leftarrow \text{nat}(Y) \\
\text{add}(s(X), Y, s(Z)) & \leftarrow \text{add}(X, Y, Z)
\end{align*}
\]

Approximation of Paths:

Every clause

\(\mathcal{P}(t_1, \ldots, t_k) \leftarrow \alpha\)

is approximated by the clauses:

\(\mathcal{P}_j(w) \leftarrow \bigwedge \Pi(\alpha)\) where

\(\Pi(g_1, \ldots, g_m) = \Pi(g_1) \cup \ldots \cup \Pi(g_m)\)

\(\Pi(q(s_1, \ldots, s_n)) = \{q_l(w) \mid w \in \Pi(s_i)\}\)

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

Example:

\[
\begin{align*}
\text{add}(0, Y, Y) & \leftarrow \text{nat}(Y) \\
\text{add}(s(X), Y, s(Z)) & \leftarrow \text{add}(X, Y, Z)
\end{align*}
\]
yields:

\[ \text{add}_1(0) \leftarrow \text{nat}_1(Y) \]
\[ \text{add}_2(Y) \leftarrow \text{nat}_1(Y) \]
\[ \text{add}_3(Y) \leftarrow \text{nat}_1(Y) \]
\[ \text{add}_1(s_1 X) \leftarrow \text{add}_1(X), \text{add}_2(Y), \text{add}_3(Z) \]
\[ \text{add}_2(Y) \leftarrow \text{add}_1(X), \text{add}_2(Y), \text{add}_3(Z) \]
\[ \text{add}_3(s_1 Z) \leftarrow \text{add}_1(X), \text{add}_2(Y), \text{add}_3(Z) \]

Discussion:

- Every literal has at most one occurrence of a variable.
- The literals \( q_j(w, Y) \) where the variable \( Y \) does not occur in the head, represent tests:
  - If there is a \( w \) with \( w_j w \in \{q_j\}_{C} \) 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 \( \Rightarrow \)
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 \( w_j \in \{q_j\} \) 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 \( \checkmark \)
We conclude:

\[
\begin{align*}
\text{add}_1(1) & \leftarrow \\
\text{add}_2(Y) & \leftarrow \text{nat}_1(Y) \\
\text{add}_3(Y) & \leftarrow \text{nat}_1(Y) \\
\text{add}_1(s_1, X) & \leftarrow \text{add}_1(X) \\
\text{add}_2(Y) & \leftarrow \text{add}_2(Y) \\
\text{add}_3(s_1, Z) & \leftarrow \text{add}_3(Z)
\end{align*}
\]

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

**Theorem**

- Every APS is equivalent to a simple APS of the form:
  \[
  \begin{align*}
  p(a, X) & \leftarrow p_1(X), \ldots, p_s(X) \\
p(X) & \leftarrow \\
p(b) & \leftarrow
  \end{align*}
  \]
  
- Every APS is equivalent to a normal APS of the form:
  \[
  \begin{align*}
  p(a, X) & \leftarrow p_1(X) \\
p(X) & \leftarrow \\
p(b) & \leftarrow
  \end{align*}
  \]

We verify:

**Theorem**

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

Let \( \mathcal{C}^* \) denote the corresponding set of clauses for the paths.

Then for all predicates \( p/k \):

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

**Proof:**

Induction on the approximations of the respective fixpoints.

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

**Theorem**

- Every APS is equivalent to a simple APS of the form:
  \[
  \begin{align*}
  p(a, X) & \leftarrow p_1(X), \ldots, p_s(X) \\
p(X) & \leftarrow \\
p(b) & \leftarrow
  \end{align*}
  \]

- Every APS is equivalent to a normal APS of the form:
  \[
  \begin{align*}
  p(a, X) & \leftarrow p_1(X) \\
p(X) & \leftarrow \\
p(b) & \leftarrow
  \end{align*}
  \]
Step 1: Removal of complicated heads:

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

\[
\begin{align*}
p(w X) & \leftarrow \text{rhs} \\
p(a^{(1)} X) & \leftarrow p_2(X) \\
p_2(a^{(2)} X) & \leftarrow p_3(X) \\
& \quad \ldots \\
p_{m-1}(a^{(m-1)} X) & \leftarrow p_m(X) \\
p_m(a^{(m)} X) & \leftarrow \text{rhs} \\
& \quad // \ p_j \text{ all new}
\end{align*}
\]

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:
  \[
  p(o X) \leftarrow p_1(X), \ldots, p_r(X) \\
p(X) \leftarrow \\
p(b) \leftarrow
  \]
- Every APS is equivalent to a normal APS of the form:
  \[
  p(o X) \leftarrow p_1(X) \\
p(X) \leftarrow \\
p(b) \leftarrow
  \]

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

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

\[
\begin{align*}
p(w) & \leftarrow \text{rhs} \\
p(a^{(1)} X) & \leftarrow p_2(X) \\
p_2(a^{(2)} X) & \leftarrow p_3(X) \\
& \quad \ldots \\
p_{m-1}(a^{(m-1)} X) & \leftarrow p_m(X) \\
p_m(a^{(m)} X) & \leftarrow \text{rhs} \\
p_{m+1}(b) & \leftarrow \text{rhs} \\
& \quad // \ p_j \text{ all new}
\end{align*}
\]
Step 2: Splitting

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

\[
\text{head} \leftarrow \text{rest, } p_1(w_1 X), \ldots, p_m(w_m X) \\
(X \text{ does not occur in head, rest})
\]

is replaced with:

\[
\text{head} \leftarrow \text{rest, } q() \\
q() \leftarrow p_1(w_1 X), \ldots, p_m(w_m X)
\]

for a new predicate \(q/0\).

Step 3: Normalization

We add simpler derived clauses:

\[
\text{head} \leftarrow p(a w), \text{rest} \\
p(a X) \leftarrow p_1(X), \ldots, p_v(X) \\
\text{implies:} \\
\text{head} \leftarrow p_1(w), \ldots, p_v(w), \text{rest} \\
p(X) \leftarrow p_1(X), \ldots, p_m(X) \\
p_1(a X) \leftarrow p_1(X), \ldots, p_v(X) \\
\text{implies:} \\
p(X) \leftarrow p_1(X), \ldots, p_m(X)
\]
Step 3 (Cont.): Normalization

\[
\begin{align*}
\text{head} & \leftarrow p(w), \text{rest} \\
p(X) & \leftarrow \text{implies:} \\
\text{head} & \leftarrow \text{rest} \\
\text{head} & \leftarrow p(b), \text{rest} \\
p(b) & \leftarrow \text{implies:} \\
\text{head} & \leftarrow \text{rest} \\
p() & \leftarrow p_1(X), \ldots, p_m(X) \\
p_a(aX) & \leftarrow p_{ni}(X), \ldots, p_{nr}(X) \\
& \text{implies:} \\
p() & \leftarrow p_{ni}(X), \ldots, p_{mnr}(X)
\end{align*}
\]