Extension of the Syntax:

We additionally consider expression of the form:

$$\begin{array}{lll} e & ::= & \ldots \mid [\,] \mid e_1 :: e_2 \mid \textbf{match } e_0 \textbf{ with } [\,] \;\to\; e_1 \;\mid\; x :: xs \;\to\; e_2 \\ & & \mid (e_1, e_2) \mid \textbf{match } e_0 \textbf{ with } (x_1, x_2) \;\to\; e_1 \end{array}$$

Top Strictness

- We assume that the program is well-typed.
- We are only interested in top constructors.
- Again, we model this property with (monotonic) Boolean functions.
- For **int**-values, this coincides with strictness :-)
- We extend the abstract evaluation $[\![e]\!]^\sharp\,\rho$ with rules for case-distinction ...

$$
\begin{array}{ll}
[\![\mathbf{match}\ e_0\ \mathbf{with}\ [\,]\ \to\ e_1\ \mid\ x :: xs\ \to\ e_2]\!]^\sharp\,\rho & = \\
\qquad [\![e_0]\!]^\sharp\,\rho \wedge ([\![e_1]\!]^\sharp\,\rho \vee [\![e_2]\!]^\sharp\,(\rho \oplus \{x, xs \mapsto 1\})) & \\
[\![\mathbf{match}\ e_0\ \mathbf{with}\ (x_1, x_2)\ \to\ e_1]\!]^\sharp\,\rho & = \\
\qquad [\![e_0]\!]^\sharp\,\rho \wedge [\![e_1]\!]^\sharp\,(\rho \oplus \{x_1, x_2 \mapsto 1\}) & \\
[\![[\,]]\!]^\sharp\,\rho\ =\ [\![e_1 :: e_2]\!]^\sharp\,\rho\ =\ [\![(e_1, e_2)]\!]^\sharp\,\rho & = \ 1
\end{array}
$$

- The rules for **match** are analogous to those for **if**.
- In case of ::, we know nothing about the values beneath the constructor; therefore $\{x, xs \mapsto 1\}$.
- We check our analysis on the function app ...

Example:

$$\mathsf{app} \;=\; \textbf{fun}\; x \;\to\; \textbf{fun}\; y \;\to\; \textbf{match}\; x \;\textbf{with}\; [\,] \;\to\; y$$
$$\mid\; x :: xs \;\to\; x :: \mathsf{app}\; xs\; y$$

Abstract interpretation yields the system of equations:

$$[\![\mathsf{app}]\!]^\sharp\; b_1\; b_2 \;=\; b_1 \wedge (b_2 \vee 1)$$
$$=\; b_1$$

We conclude that we may conclude for sure only for the first argument that its top constructor is required :-)

Total Strictness

Assume that the result of the function application is totally required.
Which arguments then are also totally required ?

We again refer to Boolean functions ...

$$
\begin{array}{lll}
[\![\textbf{match } e_0 \textbf{ with } [\,] \;\to\; e_1 \;\mid\; x, :: xs \;\to\; e_2]\!]^\sharp\, \rho & = & \textbf{let } b = [\![e_0]\!]^\sharp\, \rho \textbf{ in} \\
\multicolumn{3}{l}{b \wedge [\![e_1]\!]^\sharp\, \rho \vee [\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto b, xs \mapsto 1\}) \vee [\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto 1, xs \mapsto b\})} \\
[\![\textbf{match } e_0 \textbf{ with } (x_1, x_2) \;\to\; e_1]\!]^\sharp\, \rho & = & \textbf{let } b = [\![e_0]\!]^\sharp\, \rho \textbf{ in} \\
\multicolumn{3}{r}{[\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto 1, x_2 \mapsto b\}) \vee [\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto b, x_2 \mapsto 1\})} \\
[\![[\,]]\!]^\sharp\, \rho & = & 1 \\
[\![e_1 :: e_2]\!]^\sharp\, \rho & = & [\![e_1]\!]^\sharp\, \rho \wedge [\![e_2]\!]^\sharp\, \rho \\
[\![(e_1, e_2)]\!]^\sharp\, \rho & = & [\![e_1]\!]^\sharp\, \rho \wedge [\![e_2]\!]^\sharp\, \rho
\end{array}
$$

Discussion:

- The rules for constructor applications have changed.
- Also the treatment of **match** now involves the components z and x_1, x_2.
- Again, we check the approach for the function app.

Example:

Abstract interpretation yields the system of equations:

$$\begin{aligned} [\![\mathsf{app}]\!]^\sharp\; b_1\; b_2 &= b_1 \wedge b_2 \vee b_1 \wedge [\![\mathsf{app}]\!]^\sharp\; 1\; b_2 \vee 1 \wedge [\![\mathsf{app}]\!]^\sharp\; b_1\; b_2 \\ &= b_1 \wedge b_2 \vee b_1 \wedge [\![\mathsf{app}]\!]^\sharp\; 1\; b_2 \vee [\![\mathsf{app}]\!]^\sharp\; b_1\; b_2 \end{aligned}$$

This results in the following fixpoint iteration:

0	$\textbf{fun}\, x \to \textbf{fun}\, y \to 0$
1	$\textbf{fun}\, x \to \textbf{fun}\, y \to x \wedge y$
2	$\textbf{fun}\, x \to \textbf{fun}\, y \to x \wedge y$

We deduce that both arguments are definitely totally required if the result is totally required :-)

Warning:

Whether or not the result is totally required, depends on the context of the function call!

In such a context, a specialized function may be called ...

$$
\begin{array}{lllll}
\mathsf{app}\# & = & \textbf{fun } x \to \textbf{fun } y \to & \textbf{let } \#x' = x \textbf{ and } \#y' = y \textbf{ in} & \\
& & & \textbf{match } 'x \textbf{ with } [\,] \to y' & \\
& & & \mid \; x :: xs \to & \textbf{let } \# r = x :: \mathsf{app}\# \; xs \; y \\
& & & & \textbf{in } r
\end{array}
$$

Discussion:

- Both strictness analyses employ the same complete lattice.
- Results and application, though, are quite different :-)
- Thereby, we use the following description relations:

  Top Strictness : $\bot \, \Delta \, 0$

  Total Strictness : $z \, \Delta \, 0$ if $\bot$ occurs in z.

- Both analyses can also be combined to an a joint analysis ...

Combined Strictness Analysis

- We use the complete lattice:

$$\mathbb{T} = \{0 \sqsubset 1 \sqsubset 2\}$$

- The description relation is given by:

$$\bot \,\Delta\, 0 \qquad z \,\Delta\, 1 \; (z \text{ contains } \bot) \qquad z \,\Delta\, 2 \; (z \text{ value})$$

- The lattice is more informative, the functions, though, are no longer as efficiently representable, e.g., through Boolean expressions :-(
- We require the auxiliary functions:

$$(i \sqsubseteq x);\; y = \begin{cases} y & \text{if } i \sqsubseteq x \\ 0 & \text{otherwise} \end{cases}$$

The Combined Evaluation Function:

$$
\begin{array}{lcl}
[\![\textbf{match}\, e_0\, \textbf{with}\, [\,] \;\to\; e_1 \;\mid\; x :: xs \;\to\; e_2]\!]^\sharp\, \rho & = & \textbf{let}\; b = [\![e_0]\!]^\sharp\, \rho\; \textbf{in} \\
 & & (2 \sqsubseteq b)\; ;\; [\![e_1]\!]^\sharp\, \rho \;\sqcup \\
 & & (1 \sqsubseteq b)\; ;\; ([\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto 2, xs \mapsto b\}) \\
 & & \sqcup\; [\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto b, xs \mapsto 2\})) \\
[\![\textbf{match}\, e_0\, \textbf{with}\, (x_1, x_2) \;\to\; e_1]\!]^\sharp\, \rho & = & \textbf{let}\; b = [\![e_0]\!]^\sharp\, \rho\; \textbf{in} \\
 & & (1 \sqsubseteq b)\; ;\; ([\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto 2, x_2 \mapsto b\}) \\
 & & \sqcup\; [\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto b, x_2 \mapsto 2\})) \\
[\![[\,]]\!]^\sharp\, \rho & = & 2 \\
[\![e_1 :: e_2]\!]^\sharp\, \rho & = & \\
[\![(e_1, e_2)]\!]^\sharp\, \rho & = & 1 \sqcup ([\![e_1]\!]^\sharp\, \rho \sqcap [\![e_2]\!]^\sharp\, \rho)
\end{array}
$$

Example:

For our beloved function app, we obtain:

$$
\begin{array}{lll}
[\![\mathsf{app}]\!]^\sharp\; d_1\; d_2 & = & (2 \sqsubseteq d_1)\;;\; d_2 \sqcup \\
 & & (1 \sqsubseteq d_1)\;;\; (1 \sqcup [\![\mathsf{app}]\!]^\sharp\; d_1\; d_2 \sqcup d_1 \sqcap [\![\mathsf{app}]\!]^\sharp\; 2\; d_2) \\
 & = & (2 \sqsubseteq d_1)\;;\; d_2 \sqcup \\
 & & (1 \sqsubseteq d_1)\;;\; 1 \sqcup \\
 & & (1 \sqsubseteq d_1)\;;\; [\![\mathsf{app}]\!]^\sharp\; d_1\; d_2 \sqcup \\
 & & d_1 \sqcap [\![\mathsf{app}]\!]^\sharp\; 2\; d_2
\end{array}
$$

this results in the fixpoint computation:

0	$\mathbf{fun}\,x \to \mathbf{fun}\,y \to$	0
1	$\mathbf{fun}\,x \to \mathbf{fun}\,y \to$	$(2 \sqsubseteq x);\ y \sqcup (1 \sqsubseteq x);\ 1$
2	$\mathbf{fun}\,x \to \mathbf{fun}\,y \to$	$(2 \sqsubseteq x);\ y \sqcup (1 \sqsubseteq x);\ 1$

We conclude

- that both arguments are totally required if the result is totally required; and
- that the root of the first argument is required if the root of the result is required :-)

Remark:

The analysis can be easily generalized such that it guarantees evaluation up to a depth d ;-)

Further Directions:

- Our Approach is also applicable to other data structures.
- In principle, also higher-order (monomorphic) functions can be analyzed in this way :-)
- Then, however, we require higher-order abstract functions — of which there are many :-(
- Such functions therefore are approximated by:

$$\textbf{fun}\ x_1\ \to\ \ldots\ \textbf{fun}\ x_r\ \to\ \top$$

:-)

- For some known higher-order functions such as map, foldl, loop, ... this approach then should be improved :-))

5 Optimization of Logic Programs

We only consider the mini language PuP ("Pure Prolog"). In particular, we do not consider:

- arithmetic;
- the cut-operator.
- Self-modification by means of assert and retract.

Example:

$$
\begin{array}{lcl}
\mathsf{bigger}(X,Y) & \leftarrow & X = elephant, Y = horse \\
\mathsf{bigger}(X,Y) & \leftarrow & X = horse, Y = donkey \\
\mathsf{bigger}(X,Y) & \leftarrow & X = donkey, Y = dog \\
\mathsf{bigger}(X,Y) & \leftarrow & X = donkey, Y = monkey \\
\mathsf{is\_bigger}(X,Y) & \leftarrow & \mathsf{bigger}(X,Y) \\
\mathsf{is\_bigger}(X,Y) & \leftarrow & \mathsf{bigger}(X,Z), \mathsf{is\_bigger}(Z,Y) \\
 & \leftarrow & \mathsf{is\_bigger}(elephant, dog)
\end{array}
$$

A more realistic Example:

$$\begin{array}{rcl} \mathsf{app}(X,Y,Z) & \leftarrow & X = [\,],\ Y = Z \\ \mathsf{app}(X,Y,Z) & \leftarrow & X = [H|X'],\ Z = [H|Z'],\ \mathsf{app}(X',Y,Z') \\ & \leftarrow & \mathsf{app}(X,[Y,c],[a,b,Z]) \end{array}$$

A more realistic Example:

$$\begin{array}{rcl} \mathsf{app}(X,Y,Z) & \leftarrow & X = [\,],\ Y = Z \\ \mathsf{app}(X,Y,Z) & \leftarrow & X = [H|X'],\ Z = [H|Z'],\ \mathsf{app}(X',Y,Z') \\ & \leftarrow & \mathsf{app}(X,[Y,c],[a,b,Z]) \end{array}$$

Remark:

$[\,]$	$==$	the atom empty list
$[H\|Z]$	$==$	binary constructor application
$[a,b,Z]$	$==$	Abbreviation for: $[a\|[b\|[Z\|[\,]]]]$

Accordingly, a program p is constructed as follows:

$$
\begin{array}{lcl}
t & ::= & a \mid X \mid \_ \mid f(t_1, \ldots, t_n) \\
g & ::= & p(t_1, \ldots, t_k) \mid X = t \\
c & ::= & p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_r \\
q & ::= & \leftarrow g_1, \ldots, g_r \\
p & ::= & c_1 \ldots c_m q
\end{array}
$$

- A term t either is an atom, a (possibly anonymous) variable or a constructor application.
- A goal g either is a literal, i.e., a predicate call, or a unification.
- A clause c consists of a head $p(X_1, \ldots, X_k)$ together with body consisting of a sequence of goals.
- A program consists of a sequence of clauses together with a sequence of goals as query.

Procedural View of PuP-Programs:

literal	$\hat{=}$	procedure call
predicate	$\hat{=}$	procedure
definition	$\hat{=}$	body
term	$\hat{=}$	value
unification	$\hat{=}$	basic computation step
binding of variables	$\hat{=}$	side effect

Warning: Predicate calls ...

- do not return results!
- modify the caller solely through side effects :-)
- may fail. Then, the following definition is tried $\Longrightarrow$ backtracking

Inefficiencies:

Backtracking:

- The matching alternative must be searched for

  $\Longrightarrow$ Indexing

- Since a successful call may still fail later, the stack can only be cleared if there are no pending alternatives.

Unification:

- The translation possibly must switch between build and check several times.

- In case of unification with a variable, an Occur Check must be performed.

Type Checking:

- Since Prolog is untyped, it must be checked at run-time whether or not a term is of the desired form.

- Otherwise, ugly errors could show up.

Some Optimizations:

- Replacing last calls with jumps;
- Compile-time type inference;
- Identification of deterministic predicates ...

Example:

$$
\begin{array}{rcl}
\mathsf{app}(X, Y, Z) & \leftarrow & X = [\,],\ Y = Z \\
\mathsf{app}(X, Y, Z) & \leftarrow & X = [H|X'],\ Z = [H|Z'],\ \mathsf{app}(X', Y, Z') \\
 & \leftarrow & \mathsf{app}([a, b], [Y, c], Z)
\end{array}
$$