Extension: Data Structures

- Functions may vary in the parts which they require from a data structure ...

$$\begin{array}{lllll} \mathsf{hd} & = & \textbf{fun}\ l \ \rightarrow & \textbf{match}\ l\ \textbf{with}\ ::\ z & \rightarrow \\ & & & \textbf{match}\ z\ \textbf{with}\ (x, xs) & \rightarrow\ \ x \end{array}$$

- hd only accesses the first element of a list.
- length only accesses the backbone of its argument.
- rev forces the evaluation of the complete argument — given that the result is required completely ...

Extension of the Syntax:

We additionally consider expression of the form:

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

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 ...

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

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

Example:

$$\begin{array}{lllllllll} \mathsf{app} & = & \textbf{fun}\ x & \to & \textbf{fun}\ y & \to & \textbf{match}\ x\ \textbf{with}\ [\,] & \to & y \\ & & & & & | & ::z & \to & \textbf{match}\ z\ \textbf{with}\ (x, xs) \ \to\ ::\ (x, \mathsf{app}\ xs\ y) \end{array}$$

Abstract interpretation yields the system of equations:

$$\begin{array}{lll} [\![\mathsf{app}]\!]^\sharp\ b_1\ b_2 & = & b_1 \wedge (b_2 \vee 1) \\ & = & b_1 \end{array}$$

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 ...

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

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:

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

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 ...

$$
\begin{array}{rcl}
\mathsf{app\#} & = & \textbf{fun } x \rightarrow \textbf{fun } y \rightarrow \quad \textbf{let } \#x' = x \textbf{ and } \#y' = y \textbf{ in} \\
& & \qquad\qquad\qquad\qquad \textbf{match } {'x} \textbf{ with } [\,] \rightarrow y' \\
& & \qquad\qquad\qquad\qquad\quad \mid \; :: z \rightarrow \quad \textbf{match } z \textbf{ with } (x, xs) \rightarrow \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{let } \# r = \, :: \, (x, \mathsf{app\#}\; xs\; y) \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{in } r
\end{array}
$$

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 : $\perp \,\Delta\, 0$

  Total Strictness : $z \,\Delta\, 0$ if $\perp$ 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:

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

- 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:

$$
\begin{aligned}
[\![\textbf{match}\, e_0\, \textbf{with}\, [\,] \;\to\; e_1 \;|\; ::z \;\to\; e_2]\!]^\sharp\, \rho \quad &= \\
&\hspace{-8em}(2 \sqsubseteq [\![e_0]\!]^\sharp\, \rho) \,;\, [\![e_1]\!]^\sharp\, \rho \sqcup (1 \sqsubseteq [\![e_0]\!]^\sharp\, \rho) \,;\, [\![e_2]\!]^\sharp\, (\rho \oplus \{z \mapsto [\![e_0]\!]^\sharp\, \rho\}) \\
[\![\textbf{match}\, e_0\, \textbf{with}\, (x_1, x_2) \;\to\; e_1]\!]^\sharp\, \rho \quad &= \quad \textbf{let}\ b = [\![e_0]\!]^\sharp\, \rho \\
&\hspace{-8em}\textbf{in}\ (1 \sqsubseteq [\![e_0]\!]^\sharp\, \rho) \,;\, ([\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto 2, x_2 \mapsto b\}) \\
&\hspace{-4em}\sqcup\ [\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto b, x_2 \mapsto 2\})) \\
[\![[\,]]\!]^\sharp\, \rho \quad &= \quad 2 \\
[\![::\ e]\!]^\sharp\, \rho \quad &= \quad 1 \sqcup [\![e]\!]^\sharp\, \rho \\
[\![(e_1, e_2)]\!]^\sharp\, \rho \quad &= \quad 1 \sqcup ([\![e_1]\!]^\sharp\, \rho \sqcap [\![e_2]\!]^\sharp\, \rho)
\end{aligned}
$$

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

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:

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

A more realistic Example:

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