Example:

\[
\text{app} = \text{fun } x \to \text{fun } y \to \text{match } x \text{ with } [] \to y \\
| x::xs \to x :: \text{app } xs \ y
\]

Abstract interpretation yields the system of equations:

\[
[\text{app}]^2 b_1 b_2 = b_1 \land (b_2 \lor 1) \\
= b_1
\]

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

Example:

\[
\text{app} = \text{fun } x \to \text{fun } y \to \text{match } x \text{ with } [] \to y \\
| x::xs \to x :: \text{app } xs \ y
\]

Abstract interpretation yields the system of equations:

\[
[\text{app}]^2 b_1 b_2 = b_1 \land (b_2 \lor 1) \\
= b_1
\]

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

- 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:

\[
\text{app} = \text{fun } x \rightarrow \text{fun } y \rightarrow \text{match } x \text{ with } [] \rightarrow y \\
| \ x :: xs \rightarrow x :: \text{app} \ xs \ y
\]

Abstract interpretation yields the system of equations:

\[
[\text{app}]^2 b_1 b_2 = b_1 \land (b_2 \lor \perp) = b_1
\]

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 \text{totally required}.
Which arguments then are also totally required?

We again refer to Boolean functions...

\[
[\text{match } v_0 \text{ with } [] \rightarrow e_1 | x :: xs \rightarrow e_2]^2 \rho = \begin{cases} \\
\begin{array}{l}
\text{let } b = [e_0]^2 \rho \text{ in } b \land[e_1]^2 \rho \lor [e_2]^2 \rho \left(\rho \oplus \{x \mapsto b, x :: s \mapsto \perp\}\right) \\
\text{let } b = [e_0]^2 \rho \text{ in } [e_1]^2 \rho \left(\rho \oplus \{x_1 \mapsto b, x_2 \mapsto \perp\}\right) \\
\text{let } b = [e_0]^2 \rho \text{ in } [e_1]^2 \rho \left(\rho \oplus \{x_1 \mapsto 1, x_2 \mapsto b\}\right) \lor [e_2]^2 \rho \left(\rho \oplus \{x_1 \mapsto b, x_2 \mapsto 1\}\right) \\
[\perp]^2 \rho \\
[e_1 :: e_2]^2 \rho \\
[(e_1, e_2)]^2 \rho
\end{array}
\end{cases}
\]

\[
\begin{array}{c}
\text{let } n c \text{ f } x = f(\text{f } x + 1) \\
[f]^n b = [f]^n (b \land \perp)
\end{array}
\]
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 ...

\[
\text{match } e_0 \text{ with } [ ] \rightarrow e_1 | x :: xs \rightarrow e_2 \] \[\rho \quad \Rightarrow \quad \text{let } b = \mathbf{[e_0]_\rho} \text{ in }
\]
\[
\beta \cdot \mathbf{[e_1]_\rho} \lor \mathbf{[e_2]_\rho} (\rho \triangleright \{x \mapsto b, xs \mapsto \text{true}\}) \lor \mathbf{[e_2]_\rho} (\rho \triangleright \{x \mapsto \text{false}, xs \mapsto b\})
\]
\[
\text{match } e_0 \text{ with } (x_1, x_2) \rightarrow e_1 \] \[\rho \quad \Rightarrow \quad \text{let } b = \mathbf{[e_0]_\rho} \text{ in }
\]
\[
\mathbf{[e_1]_\rho} (\rho \triangleright \{x_1 \mapsto \text{false}, x_2 \mapsto b\}) \lor \mathbf{[e_1]_\rho} (\rho \triangleright \{x_1 \mapsto \text{true}, x_2 \mapsto b\})
\]
\[
\mathbf{[f]_\rho} \quad \Rightarrow \quad \mathbf{[e_1]_\rho} \wedge \mathbf{[e_2]_\rho}
\]
\[
\mathbf{[f_1; f_2]_\rho} \quad \Rightarrow \quad \mathbf{[e_1]_\rho} \wedge \mathbf{[e_2]_\rho}
\]

Discussion:

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

Example:

Abstract interpretation yields the system of equations:

\[
\mathbf{[\text{app}]_\rho} \cdot b_1 \cdot b_2 = b_1 \land b_2 \lor \mathbf{[\text{app}]_\rho} \cdot 1 \cdot b_2 \lor 1 \land \mathbf{[\text{app}]_\rho} \cdot b_1 \cdot b_2
\]
\[
= b_1 \land b_2 \lor b_1 \land \mathbf{[\text{app}]_\rho} \cdot 1 \cdot b_2 \lor \mathbf{[\text{app}]_\rho} \cdot b_1 \cdot b_2
\]
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:

\[
\text{[app]}^2 \ b_1 \ b_2 &= b_1 \land b_2 \lor b_1 \land \text{[app]}^2 1 \ b_2 \lor \text{[app]}^2 \ b_1 \ b_2 \\
&= b_1 \land b_2 \lor b_1 \land \text{[app]}^2 1 \ b_2 \lor \text{[app]}^2 \ b_1 \ b_2
\]

This results in the following fixpoint iteration:

<table>
<thead>
<tr>
<th>Step</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \text{fun} \ x \to \text{fun} \ y \to 0 )</td>
</tr>
<tr>
<td>1</td>
<td>( \text{fun} \ x \to \text{fun} \ y \to x \land y )</td>
</tr>
<tr>
<td>2</td>
<td>( \text{fun} \ x \to \text{fun} \ y \to x \land y )</td>
</tr>
</tbody>
</table>

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 ...

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:

\[
\text{[app]}^2 \ b_1 \ b_2 &= b_1 \land b_2 \lor b_1 \land \text{[app]}^2 1 \ b_2 \lor 1 \land \text{[app]}^2 \ b_1 \ b_2 \\
&= b_1 \land b_2 \lor b_1 \land \text{[app]}^2 1 \ b_2 \lor \text{[app]}^2 \ b_1 \ b_2 \\
&= b_1 \land b_2 \lor b_1 \land \text{[app]}^2 1 \ b_2 \lor \text{[app]}^2 \ b_1 \ b_2 \\
&= b_1 \land b_2 \lor b_1 \land 0 \lor 0
\]
This results in the following fixpoint iteration:

\[
\begin{array}{c|c}
0 & \text{fun } x \to \text{fun } y \to 0 \\
1 & \text{fun } x \to \text{fun } y \to x \land y \\
2 & \text{fun } x \to \text{fun } y \to x \land y \\
\end{array}
\]

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 ...

853

\[
\begin{array}{c}
\text{app} \text{#} \text{fun } x \to \text{fun } y \to (\text{let } \# x' = x \land y \text{ in } \text{app} \text{#} x' y) \\
\text{match } x \text{ with } [ ] \to y \\
| x :: x s \to \text{let } \# r = x :: \text{app} \text{#} x s y \\
\end{array}
\]

Discussion:

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

Combined Strictness Analysis

- We use the complete lattice:
  \[ T = \{ 0 \sqsubseteq 1 \sqsubseteq 2 \} \]
- The description relation is given by:
  \[ \bot \sqsubseteq 0 \quad z \sqsubseteq 1 \quad (z \text{ contains } \bot) \quad z \sqsubseteq 2 \quad (z \text{ value}) \]
- The lattice is more informative, the functions, though, are no longer as efficiently representable, e.g., through Boolean expressions. :-(
- We require the auxiliary functions:
  \[(i \sqsubseteq x) \cdot y = \begin{cases} y & \text{if } i \sqsubseteq x \\ 0 & \text{otherwise} \end{cases} \]
The Combined Evaluation Function:

\[
\text{match } c_0 \text{ with } \begin{cases} 
| \rightarrow c_1 \mid x::xs & \rightarrow c_2 \end{cases} \rho = \begin{cases} 
\text{let } b = [c_0]^{\rho} \text{ in } \\
(2 \sqcup b); [c_1]^{\rho} \sqcup \\\n(1 \sqcap b); ([c_2]^{\rho} (\rho \sqcup \{x \mapsto 2, xs \mapsto b\}) \\
\sqcup [c_2]^{\rho} (\rho \sqcup \{x \mapsto b, xs \mapsto 2\}) \\
(1 \sqcap b); ([c_1]^{\rho} (\rho \sqcup \{x_1 \mapsto 2, x_2 \mapsto b\}) \\
\sqcup [c_1]^{\rho} (\rho \sqcup \{x_1 \mapsto b, x_2 \mapsto 2\})) \\
\end{cases}
\]

\[
\llbracket \llbracket \rho \rrbracket^\rho = 2 \\
[e_1; e_2]^{\rho} = 1 \sqcup ([e_1]^{\rho} \cap [e_2]^{\rho})
\]

Example:

For our beloved function \texttt{app}, we obtain:

\[
\texttt{app}^{\rho} d_1 d_2 = (2 \sqcup d_1); d_2 \sqcup (1 \sqcap d_1); (1 \sqcup \texttt{app}^\rho d_1 d_2 \sqcup d_1 \sqcap \texttt{app}^\rho d_2)
\]

\[
= (2 \sqcup d_1); d_2 \sqcup (1 \sqcap d_1); 1 \sqcup (1 \sqcap d_1); \texttt{app}^\rho d_1 d_2 \sqcup d_1 \sqcap \texttt{app}^\rho d_2
\]

this results in the fixpoint computation:

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:

  \[
  \text{fun } x_1 \rightarrow \ldots \text{fun } x_n \rightarrow \top
  \]

- For some known higher-order functions such as \texttt{map}, \texttt{foldl}, \texttt{loop}, ...
  this approach then should be improved

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:
  \[
  \text{fun } x_1 \to \ldots \text{fun } x_r \to \top
  \]

- 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:

\[
\begin{align*}
\text{bigger}(X, Y) & \leftarrow X = \text{elephant}, Y = \text{horse} \\
\text{bigger}(X, Y) & \leftarrow X = \text{horse}, Y = \text{donkey} \\
\text{bigger}(X, Y) & \leftarrow X = \text{donkey}, Y = \text{dog} \\
\text{bigger}(X, Y) & \leftarrow X = \text{donkey}, Y = \text{monkey} \\
\text{is_bigger}(X, Y) & \leftarrow \text{bigger}(X, Y) \\
\text{is_bigger}(X, Y) & \leftarrow \text{bigger}(X, Z), \text{is_bigger}(Z, Y) \\
& \quad \leftarrow \text{is_bigger}(\text{elephant}, \text{dog})
\end{align*}
\]

... yields the tree:
... yields the tree:

Idea (2):
- Decision trees are exponentially large :-(
- Often, however, many sub-trees are isomorphic :-) 
- Isomorphic sub-trees need to be represented only once ...
Idea (3):

- Nodes whose test is irrelevant, can also be abandoned …

\[ \text{ROBDD} \]

Discussion:

- This representation of the Boolean function \( f \) is unique !
  \[ \implies \] Equality of functions is efficiently decidable !!
- For the representation to be useful, it should support the basic operations: \( \land, \lor, \neg, \Rightarrow, \exists x \) …

\[
\begin{align*}
[b_1 \land b_2]_k &= b_1 \land b_2 \\
[f \land g]_{i-1} &= \text{fun } x_i \rightarrow \begin{cases} [f 1 \land g 1]_i, & \text{if } x_i \\ [f 0 \land g 0]_i, & \text{else} \end{cases}
\end{align*}
\]

// analogous for the remaining operators

Background 6: Binary Decision Diagrams

Idea (1):

- Choose an ordering \( x_1, \ldots, x_k \) on the arguments …
- Represent the function \( f : \mathbb{B} \rightarrow \ldots \rightarrow \mathbb{B} \) by \( [f]_0 \) where:
  \[
  \begin{align*}
  [b]_k &= b \\
  [f]_{i-1} &= \text{fun } x_i \rightarrow \begin{cases} [f 1]_i, & \text{if } x_i \\ [f 0], & \text{else} \end{cases}
  \end{align*}
  \]

Example: \( f \ x_1 \ x_2 \ x_3 = x_1 \land (x_2 \leftrightarrow x_3) \)
\[ \square x_j f |_{i-1} = \text{fun } x_i \rightarrow \text{if } x_i \text{ then } \exists x_j f |_i \text{ else } \exists x_j f |_i \text{ if } i < j \]

\[ \exists x_j f |_{j-1} = \neg f 0 \lor f 1 |_j \]

- Operations are executed bottom-up.
- Root nodes of already constructed sub-graphs are stored in a unique-table
  
  \[ \implies \text{Isomorphy can be tested in constant time} \]
- The operations thus are polynomial in the size of the input BDDs :-)

Discussion:

- Originally, BDDs have been developed for circuit verification.
- Today, they are also applied to the verification of software ...
- A system state is encoded by a sequence of bits.
- A BDD then describes the set of all reachable system states.
- Warning: Repeated application of Boolean operations may increase the size dramatically!
- The variable ordering may have a dramatic impact ...

Example: 
\[(x_1 \leftrightarrow x_2) \land (x_3 \leftrightarrow x_4)\]