Script **generated by TTT**

Title: Seidl: Programmoptimierung (03.02.2016)

Date: Wed Feb 03 10:21:20 CET 2016

Duration: 81:49 min

Pages: 41

Example

$$\text{from} \quad = \quad \mathbf{fun}\ n \ \to \quad n :: \text{from}\ (n+1)$$

$$\begin{array}{lcl} \text{take} & = & \mathbf{fun}\ k \to \mathbf{fun}\ s \to \ \mathbf{if}\ k \leq 0\ \mathbf{then}\ [\,] \\ & & \qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{match}\ s\ \mathbf{with}\ [\,] \ \to\ [\,] \\ & & \qquad\qquad\qquad\qquad\quad \mid\ x :: xs \ \to \ x :: \text{take}\ (k-1)\ xs \end{array}$$

$$\begin{array}{lcl} [\![\mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e]\!]^\sharp\ \rho & = & [\![e]\!]^\sharp\ (\rho \oplus \{x_1 \mapsto [\![e_1]\!]^\sharp\ \rho\}) \\ [\![\mathbf{let}\ \#x_1 = e_1\ \mathbf{in}\ e]\!]^\sharp\ \rho & = & ([\![e_1]\!]^\sharp\ \rho) \wedge ([\![e]\!]^\sharp\ (\rho \oplus \{x_1 \mapsto 1\})) \end{array}$$

System of Equations

$$[\![f_i]\!]^\sharp b_1\ \ldots\ b_k = [\![e_i]\!]^\sharp\ \{x_j \mapsto b_j \mid j = 1, \ldots, k\}, \qquad i = 1, \ldots, n, b_1, \ldots, b_k \in \mathbb{B}$$

- The unkowns of the system of equations are the functions $[\![f_i]\!]^\sharp$ or the individual entries $[\![f_i]\!]^\sharp b_1\ \ldots\ b_k$ in the value table.
- All right-hand sides are monotonic!
- Consequently, there is a least solution.
- The complete lattice $\mathbb{B} \to \ldots \to \mathbb{B}$ has height $\mathcal{O}(2^k)$.

Extension: Data Structures

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

$$\text{hd} \quad = \quad \mathbf{fun}\ l \ \to\ \mathbf{match}\ l\ \mathbf{with}\ x :: xs \ \to\ x$$

- 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: Data Structures

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

$$\text{hd} \quad = \quad \textbf{fun}\; l \;\to\; \textbf{match}\; l \;\textbf{with}\; x :: xs \;\to\; x$$

- 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:

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

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 $[\![e]\!]^\sharp\,\rho$ with rules for case-distinction ...

Idea (cont.)

- We determine the abstract semantics of all functions.
- For that, we put up a system of equations ...

Auxiliary Function

$$\begin{array}{lcl} [\![e]\!]^\sharp & : & (\mathit{Vars} \to \mathbb{B}) \to \mathbb{B} \\ [\![c]\!]^\sharp\,\rho & = & 1 \\ [\![x]\!]^\sharp\,\rho & = & \rho\, x \\ [\![\Box_1\, e]\!]^\sharp\,\rho & = & [\![e]\!]^\sharp\,\rho \\ [\![e_1\, \Box_2\, e_2]\!]^\sharp\,\rho & = & [\![e_1]\!]^\sharp\,\rho \wedge [\![e_2]\!]^\sharp\,\rho \\ [\![\textbf{if}\; e_0 \;\textbf{then}\; e_1 \;\textbf{else}\; e_2]\!]^\sharp\,\rho & = & [\![e_0]\!]^\sharp\,\rho \wedge ([\![e_1]\!]^\sharp\,\rho \vee [\![e_2]\!]^\sharp\,\rho) \\ [\![f\; e_1\; \ldots\; e_k]\!]^\sharp\,\rho & = & [\![f]\!]^\sharp\,([\![e_1]\!]^\sharp\,\rho)\; \ldots\; ([\![e_k]\!]^\sharp\,\rho) \\ \ldots & & \end{array}$$

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 $[\![e]\!]^\sharp\,\rho$ with rules for case-distinction ...

860

$$[\![\text{match } e_0 \text{ with } [\,] \to e_1 \mid x :: xs \to e_2]\!]^\sharp\, \rho = [\![e_0]\!]^\sharp\, \rho \wedge ([\![e_1]\!]^\sharp\, \rho \vee [\![e_2]\!]^\sharp\, (\rho \oplus \{x, xs \mapsto 1\}))$$

$$[\![\text{match } e_0 \text{ with } (x_1, x_2) \to e_1]\!]^\sharp\, \rho = [\![e_0]\!]^\sharp\, \rho \wedge [\![e_1]\!]^\sharp\, (\rho \oplus \{x_1, x_2 \mapsto 1\})$$

$$[\![[\,]]\!]^\sharp\, \rho = [\![e_1 :: e_2]\!]^\sharp\, \rho = [\![(e_1, e_2)]\!]^\sharp\, \rho = 1$$

- 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

$$\begin{array}{lll} \text{app} & = & \text{fun } x \to \text{fun } y \to \text{match } x \text{ with } [\,] \to y \\ & & \mid x :: xs \to x :: \text{app } xs\ y \end{array}$$

Abstract interpretation yields the system of equations:

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

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 ...

$$[\![\text{match } e_0 \text{ with } [\,] \to e_1 \mid x, :: xs \to e_2]\!]^\sharp\, \rho = \text{let } b = [\![e_0]\!]^\sharp\, \rho \text{ in}$$
$$b \wedge [\![e_1]\!]^\sharp\, \rho \vee [\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto b, xs \mapsto 1\}) \vee [\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto 1, xs \mapsto b\})$$
$$[\![\text{match } e_0 \text{ with } (x_1, x_2) \to e_1]\!]^\sharp\, \rho = \text{let } b = [\![e_0]\!]^\sharp\, \rho \text{ in}$$
$$[\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto 1, x_2 \mapsto b\}) \vee [\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto b, x_2 \mapsto 1\})$$
$$[\![[\,]]\!]^\sharp\, \rho = 1$$
$$[\![e_1 :: e_2]\!]^\sharp\, \rho = [\![e_1]\!]^\sharp\, \rho \wedge [\![e_2]\!]^\sharp\, \rho$$
$$[\![(e_1, e_2)]\!]^\sharp\, \rho = [\![e_1]\!]^\sharp\, \rho \wedge [\![e_2]\!]^\sharp\, \rho$$

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 ...

$$[\![\text{match } e_0 \text{ with } [\,] \to e_1 \mid x, :: xs \to e_2]\!]^\sharp\, \rho = \text{let } b = [\![e_0]\!]^\sharp\, \rho \text{ in}$$
$$b \wedge [\![e_1]\!]^\sharp\, \rho \vee [\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto b, xs \mapsto 1\}) \vee [\![e_2]\!]^\sharp\, (\rho \oplus \{x \mapsto 1, xs \mapsto b\})$$
$$[\![\text{match } e_0 \text{ with } (x_1, x_2) \to e_1]\!]^\sharp\, \rho = \text{let } b = [\![e_0]\!]^\sharp\, \rho \text{ in}$$
$$[\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto 1, x_2 \mapsto b\}) \vee [\![e_1]\!]^\sharp\, (\rho \oplus \{x_1 \mapsto b, x_2 \mapsto 1\})$$
$$[\![[\,]]\!]^\sharp\, \rho = 1$$
$$[\![e_1 :: e_2]\!]^\sharp\, \rho = [\![e_1]\!]^\sharp\, \rho \wedge [\![e_2]\!]^\sharp\, \rho$$
$$[\![(e_1, e_2)]\!]^\sharp\, \rho = [\![e_1]\!]^\sharp\, \rho \wedge [\![e_2]\!]^\sharp\, \rho$$

863

$[\![\text{match } z \text{ with } (x_1, x_2) \to x_1]\!]^\sharp \rho$

$= 1$

$= 0 \vee 1$

$= 1$

$\rho z = 1$

$\rho ? = 0$

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 ...

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

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:

$$
\begin{array}{lll}
[\![\text{app}]\!]^\sharp\,b_1\,b_2 & = & b_1 \wedge b_2 \vee b_1 \wedge [\![\text{app}]\!]^\sharp\,1\,b_2 \vee \cancel{1 \wedge} [\![\text{app}]\!]^\sharp\,b_1\,b_2 \\
 & = & b_1 \wedge b_2 \vee b_1 \wedge [\![\text{app}]\!]^\sharp\,1\,b_2 \vee [\![\text{app}]\!]^\sharp\,b_1\,b_2
\end{array}
$$

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

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 ...

Discussion

$b_1 \wedge b_2$

- 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:

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

0

This results in the following fixpoint iteration:

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

We deduce that both arguments are definitely totally required if the result is totally required.

Caveat

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:

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

$= b_1 \wedge b_2 \vee (b_1 \wedge 1 \wedge b_2) \vee (b_1 \wedge b_2)$

This results in the following fixpoint iteration:

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

We deduce that both arguments are definitely totally required if the result is totally required.

Caveat

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:

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

This results in the following fixpoint iteration:

0	$\mathbf{fun}\; x \to \mathbf{fun}\; y \to 0$
1	$\mathbf{fun}\; x \to \mathbf{fun}\; y \to x \wedge y$
2	$\mathbf{fun}\; x \to \mathbf{fun}\; y \to x \wedge y$

We deduce that both arguments are definitely totally required if the result is totally required.

Caveat

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 ...

$$
\begin{aligned}
\text{app\#} \;=\; \mathbf{fun}\; x \to \mathbf{fun}\; y \to\; & \mathbf{let}\; \#x' = x\; \mathbf{and}\; \#y' = y\; \mathbf{in} \\
& \mathbf{match}\; x'\; \mathbf{with}\; [\,] \to y' \\
& \;\mid\; x :: xs \to \;\; \mathbf{let}\; \# r = x :: \text{app\#}\; xs\; y \\
& \qquad\qquad\qquad\; \mathbf{in}\; r
\end{aligned}
$$

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 ...

$[\![\text{match}\; z\; \text{with}\; (x_1, x_2) \to x_1]\!]^\sharp\, \rho$

$= 1$

$= 0 \vee 1$

$= 1$

$\rho\, z = 1$

$\rho\, \top = 0$

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

$[1;\ 2;\ \perp]$

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 ...

$[\![\textbf{match } z \textbf{ with } (x_1, x_2) \to x_1]\!]\rho$

Combined Strictness Analysis

- We use the complete lattice:

$= 1$

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

$\rho z = 1$

- The description relation is given by:

$= 0 \lor 1$

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

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

$= 1$

- We require the auxiliary functions:

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

Example

For our beloved function app, we obtain:

$[1;\ 2;\ \perp]$

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

this results in the fixpoint computation:

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

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:

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

- For some known higher-order functions such as map, foldl, loop, ... only unary or binary functional arguments are required — of which there are sufficiently few.

$[1; 2; \bot]$ $1 \mid 0$

$[1; \bot]$ $1 \mid 0$

$\bot$ $0 \mid 0$

$[\![\text{map}]\!]^\# : (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B}$

5 Optimization of Logic Programs

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

$[\![\text{foldl}]\!]^\# : (\mathbb{B} \to \mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B}$

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

3   2   4   6   2

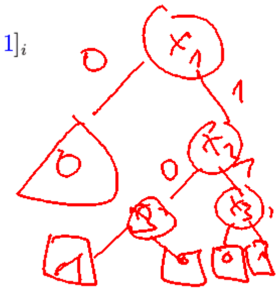
Background 6: Binary Decision Diagrams

Idea (1)

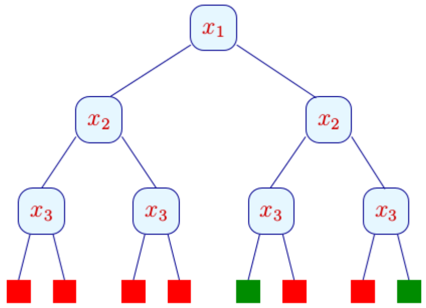
- Choose an ordering $x_1, \ldots, x_k$ on the arguments ...
- Represent the function $f : \mathbb{B} \to \ldots \to \mathbb{B}$ by $[f]_0$ where:

$$\begin{array}{lcl} [b]_k & = & b \\ [f]_{i-1} & = & \mathbf{fun}\ x_i\ \to\ \mathbf{if}\ x_i\ \mathbf{then}\ [f\ 1]_i \\ & & \qquad\qquad \mathbf{else}\ [f\ 0]_i \end{array}$$

Example $\quad f\ x_1\ x_2\ x_3 \;=\; x_1 \wedge (x_2 \leftrightarrow x_3)$

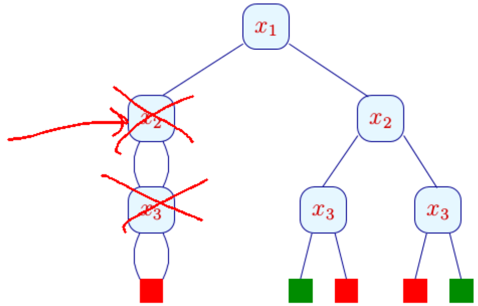


... yields the tree:



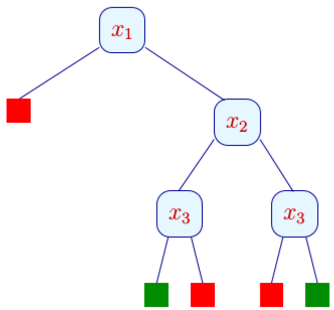
Idea (2)

- 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 ...



Discussion

- This representation of the Boolean function f is unique !

  $\Longrightarrow$

  Equality of functions is efficiently decidable !!
- For the representation to be useful, it should support the basic operations: $\wedge, \vee, \neg, \Rightarrow, \exists\, x_j$ ...

$$
\begin{array}{lll}
[b_1 \wedge b_2]_k & = & b_1 \wedge b_2 \\
[f \wedge g]_{i-1} & = & \textbf{fun } x_i \;\to\; \textbf{if } x_i \textbf{ then } [f\,1 \wedge g\,1]_i \\
 & & \qquad\qquad \textbf{else } [f\,0 \wedge g\,0]_i
\end{array}
$$

// analogous for the remaining operators

$$
\begin{aligned}
[\exists x_j .\, f]_{i-1} &= \textbf{fun } x_i \to \textbf{if } x_i \textbf{ then } [\exists x_j .\, f\, 1]_i \\
&\qquad\qquad \textbf{else } [\exists x_j .\, f\, 0]_i \qquad\qquad \text{if } i < j \\
[\exists x_j .\, f]_{j-1} &= [f\, 0 \vee f\, 1]_j
\end{aligned}
$$

- Operations are executed bottom-up.
- Root nodes of already constructed sub-graphs are stored in a unique-table

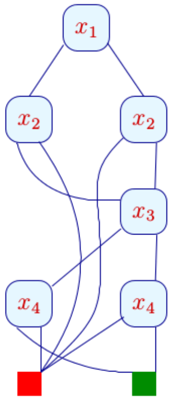
  $\Longrightarrow$

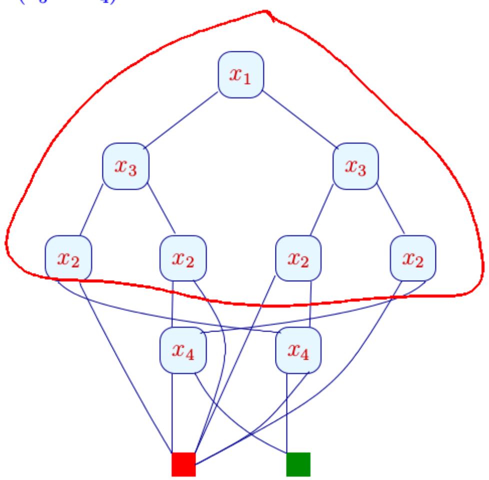
  Isomorphy can be tested in constant time !
- The operations thus are polynomial in the size of the input BDDs.

Discussion

- Originally, BDDs have been developped 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.
- Caveat: Repeated application of Boolean operations may increase the size dramatically !
- The variable ordering may have a dramatic impact ...

Example: $(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)$





Discussion (2)

- In general, consider the function:

  $$(x_1 \leftrightarrow x_2) \wedge \ldots \wedge (x_{2n-1} \leftrightarrow x_{2n})$$

  W.r.t. the variable ordering:

  $$x_1 < x_2 < \ldots < x_{2n}$$

  the BDD has $3n$ internal nodes.

  W.r.t. the variable ordering:

  $$x_1 < x_3 < \ldots < x_{2n-1} < x_2 < x_4 < \ldots < x_{2n}$$

  the BDD has more than 2^n internal nodes !!
- A similar result holds for the implementation of Addition through BDDs.

Discussion (3)

- Not all Boolean functions have small BDDs ...
- Difficult functions:
  - multiplication;
  - indirect addressing ...

$\Longrightarrow$ data-intensive programs cannot be analyzed in this way !

Discussion (3)

- Not all Boolean functions have small BDDs ...
- Difficult functions:
  - multiplication;
  - indirect addressing ...

$\Longrightarrow$ data-intensive programs cannot be analyzed in this way !

900