Script generated by TTT

Title: Seidl: Programmoptimierung (30.01.2012)

Date: Mon Jan 30 12:29:50 CET 2012

Duration: 91:00 min

Pages: 33

Observation:

- In PuP, functions must be simulated through predicates.
- These then have designated input- and output parameters.
- Input parameters are those which are instantiated with a variable-free term whenever the predicate is called.

  These are also called ground.
- In the example, the first parameter of app is an input parameter.
- Unification with such a parameter can be implemented as pattern matching !
- Then we see that app in fact is deterministic !!!

5.1 Groundness Analysis

A variable X is called ground w.r.t. a program execution π starting program entry and entering a program point v, if X is bound to a variable-free term.

Goal:

- Find all variables which are ground whenever a particular program point is reached !
- Find all arguments of a predicate which are ground whenever the predicate is called !

Idea:

- Describe groundness by values from $\mathbb{B}$:

  $1 \quad == \quad$ variable-free term;

  $0 \quad == \quad$ term which contains variables.
- A set of variable assignments is described by Boolean functions :-)

  $X \leftrightarrow Y \quad == \quad X$ is ground iff Y is ground.

  $X \wedge Y \quad == \quad X$ and Y are ground.

Idea:

- Describe groundness by values from $\mathbb{B}$:

  $1 \quad == \quad$ variable-free term;

  $0 \quad == \quad$ term which contains variables.

- A set of variable assignments is described by Boolean functions :-)

  $X \leftrightarrow Y \quad == \quad X$ is ground iff Y is ground.

  $X \wedge Y \quad == \quad X$ and Y are ground.

Idea (cont.):

- The constant function 0 denotes an unreachable program point.
- Occurring sets of variable assignments are closed under substitution.

  This means that for every occurring function $\phi \neq 0$,

  $$\phi(1, \ldots, 1) = 1$$

  These functions are called positive.
- The set of all positive functions is called Pos.

  Ordering: $\phi_1 \sqsubseteq \phi_2$ if $\phi_1 \Rightarrow \phi_2$.
- In particular, the least element is 0 :-)

Idea:

- Describe groundness by values from $\mathbb{B}$:

  $1 \quad == \quad$ variable-free term;

  $0 \quad == \quad$ term which contains variables.

- A set of variable assignments is described by Boolean functions :-)

  $X \leftrightarrow Y \quad == \quad X$ is ground iff Y is ground.

  $X \wedge Y \quad == \quad X$ and Y are ground.

881

Idea (cont.):

- The constant function 0 denotes an unreachable program point.
- Occurring sets of variable assignments are closed under substitution.

  This means that for every occurring function $\phi \neq 0$,

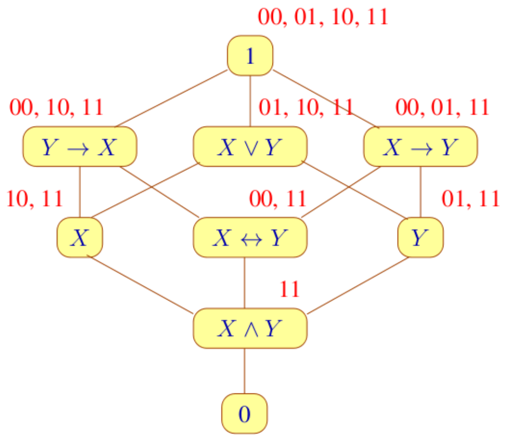
  $$\phi(1, \ldots, 1) = 1$$

  These functions are called positive.
- The set of all positive functions is called Pos.

  Ordering: $\phi_1 \sqsubseteq \phi_2$ if $\phi_1 \Rightarrow \phi_2$.
- In particular, the least element is 0 :-)

882

Example:



Remarks:

- Not all positive functions are monotonic !!!
- For k variables, there are $2^{2^k-1}+1$ many functions.
- The height of the complete lattice is 2^k.
- We construct an interprocedural analysis which for every predicate p determines a (monotonic) transformation

$$[\![p]\!]^\sharp \;:\; \mathsf{Pos} \to \mathsf{Pos}$$

- For every clause, $\quad p(X_1,\ldots,X_k) \Leftarrow g_1,\ldots,g_n \quad$ we obtain the constraint:

$$[\![p]\!]^\sharp\,\psi \quad \sqsupseteq \quad \exists\,X_{k+1},\ldots,X_m.\,[\![g_n]\!]^\sharp\,(\ldots([\![g_1]\!]^\sharp\,\psi)\ldots)$$

// $\quad m$ number of clause variables

Remarks:

- Not all positive functions are monotonic !!!
- For k variables, there are $2^{2^k-1}+1$ many functions.
- The height of the complete lattice is 2^k.
- We construct an interprocedural analysis which for every predicate p determines a (monotonic) transformation

$$[\![p]\!]^\sharp \;:\; \mathsf{Pos} \to \mathsf{Pos}$$

- For every clause, $\quad p(X_1,\ldots,X_k) \Leftarrow g_1,\ldots,g_n \quad$ we obtain the constraint:

$$[\![p]\!]^\sharp\,\psi \quad \sqsupseteq \quad \exists\,X_{k+1},\ldots,X_m.\,[\![g_n]\!]^\sharp\,(\ldots\boxed{([\![g_1]\!]^\sharp\,\psi}\ldots)$$

// $\quad m$ number of clause variables

884

Abstract Unification:

$$\begin{array}{rcl} [\![X=t]\!]^\sharp\psi & = & \psi \wedge (X \leftrightarrow X_1 \wedge \ldots \wedge X_r) \\ & & \text{if} \quad \mathit{Vars}(t) = \{X_1,\ldots,X_r\}. \end{array}$$

Abstract Literal:

$$[\![q(s_1,\ldots,s_k)]\!]^\sharp\psi \quad = \quad \mathsf{combine}^\sharp_{s_1,\ldots,s_k}(\psi, [\![q]\!]^\sharp\,(\mathsf{enter}^\sharp_{s_1,\ldots,s_k}\,\psi))$$

// $\quad$ analogous to procedure call !!

Thereby:

$$\text{enter}^\sharp_{s_1,\ldots,s_k}\,\psi \;=\; \text{ren}\,(\exists\, X_1,\ldots,X_m.\, [\![\bar{X}_1 = s_1,\ldots,\bar{X}_k = s_k]\!]^\sharp\psi)$$

$$\text{combine}^\sharp_{s_1,\ldots,s_k}(\psi,\psi_1) \;=\; \exists\, \bar{X}_1,\ldots,\bar{X}_r.\, \psi \wedge [\![\bar{X}_1 = s_1,\ldots,\bar{X}_k = s_k]\!]^\sharp(\overline{\text{ren}}\,\psi_1)$$

where

$$\begin{aligned} \exists\, X.\,\phi &= \phi[0/X] \vee \phi[1/X] \\ \text{ren}\,\phi &= \phi[X_1/\bar{X}_1,\ldots,X_k/\bar{X}_k] \\ \overline{\text{ren}}\,\phi &= \phi[\bar{X}_1/X_1,\ldots,\bar{X}_r/X_r] \end{aligned}$$

Abstract Unification:

$$\begin{aligned} [\![X = t]\!]^\sharp\psi &= \psi \wedge (X \leftrightarrow X_1 \wedge \ldots \wedge X_r) \\ &\quad \text{if} \quad \mathit{Vars}(t) = \{X_1,\ldots,X_r\}. \end{aligned}$$

Abstract Literal:

$$[q(s_1,\ldots,s_k)]^\sharp\psi \;=\; \text{combine}^\sharp_{s_1,\ldots,s_k}(\psi, [\![q]\!]^\sharp(\text{enter}^\sharp_{s_1,\ldots,s_k}\,\psi))$$

// analogous to procedure call !!

$\exists\, \bar{X}_1, \bar{X}_2 .\; \bar{X}_1 \leftrightarrow \bar{X}_2 \wedge$
$\bar{X}_1 \leftrightarrow (X_1 \wedge X_2) \wedge \ldots$

Thereby:

$$\text{enter}^\sharp_{s_1,\ldots,s_k}\,\psi \;=\; \text{ren}\,(\exists\, X_1,\ldots,X_m.\, [\![\bar{X}_1 = s_1,\ldots,\bar{X}_k = s_k]\!]^\sharp\psi)$$

$$\text{combine}^\sharp_{s_1,\ldots,s_k}(\psi,\psi_1) \;=\; \exists\, \bar{X}_1,\ldots,\bar{X}_r.\, \psi \wedge [\![\bar{X}_1 = s_1,\ldots,\bar{X}_k = s_k]\!]^\sharp(\overline{\text{ren}}\,\psi_1)$$

where

$$\begin{aligned} \exists\, X.\,\phi &= \phi[0/X] \vee \phi[1/X] \\ \text{ren}\,\phi &= \phi[X_1/\bar{X}_1,\ldots,X_k/\bar{X}_k] \\ \overline{\text{ren}}\,\phi &= \phi[\bar{X}_1/X_1,\ldots,\bar{X}_r/X_r] \end{aligned}$$

886

Example:

$$\begin{aligned} \text{app}(X,Y,Z) &\leftarrow X = [\,],\; Y = Z \\ \text{app}(X,Y,Z) &\leftarrow X = [H|X'],\; Z = [H|Z'],\; \text{app}(X',Y,Z') \end{aligned}$$

Then

$$\begin{aligned} [\![\text{app}]\!]^\sharp(X) &\sqsupseteq X \wedge (Y \leftrightarrow Z) \\ [\![\text{app}]\!]^\sharp(X) &\sqsupseteq \textbf{let } \psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z') \\ &\qquad \textbf{in } \exists\, H, X', Z'.\, \text{combine}^\sharp_{\ldots}(\psi, [\![\text{app}]\!]^\sharp(\text{enter}^\sharp_{\ldots}(\psi))) \end{aligned}$$

where for $\psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z')$:

$$\begin{aligned} \text{enter}^\sharp_{\ldots}(\psi) &= X \\ \text{combine}^\sharp_{\ldots}(\psi, X \wedge (Y \leftrightarrow Z)) &= (X \wedge H \wedge X' \wedge (Z \leftrightarrow Z') \wedge (Y \leftrightarrow Z') \end{aligned}$$

Remarks:

- Not all positive functions are monotonic !!!
- For k variables, there are $2^{2^k-1}+1$ many functions.
- The height of the complete lattice is 2^k.
- We construct an interprocedural analysis which for every predicate p determines a (monotonic) transformation

$$[\![p]\!]^\sharp \;:\; \mathsf{Pos} \to \mathsf{Pos}$$

- For every clause, $\quad p(X_1,\ldots,X_k) \Leftarrow g_1,\ldots,g_n \quad$ we obtain the constraint:

$$[\![p]\!]^\sharp\,\psi \quad \sqsupseteq \quad \exists\, X_{k+1},\ldots,X_m.\; [\![g_n]\!]^\sharp\,(\ldots([\![g_1]\!]^\sharp\,\psi)\ldots)$$

// $\quad m$ number of clause variables

Example:

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

$X \wedge (Y \leftrightarrow Z)$

Then

$$\begin{array}{lll}
[\![\mathsf{app}]\!]^\sharp(X) & \sqsupseteq & X \wedge (Y \leftrightarrow Z) \\
[\![\mathsf{app}]\!]^\sharp(X) & \sqsupseteq & \textbf{let } \psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z') \\
 & & \textbf{in } \exists\, H, X', Z'.\; \mathsf{combine}^\sharp_{\ldots}\,(\psi, [\![\mathsf{app}]\!]^\sharp(\mathsf{enter}^\sharp_{\ldots}\,(\psi)))
\end{array}$$

where for $\quad \psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z')$:

$$\begin{array}{lll}
\mathsf{enter}^\sharp_{\ldots}\,(\psi) & = & X \\
\mathsf{combine}^\sharp_{\ldots}\,(\psi, X \wedge (Y \leftrightarrow Z)) & = & (X \wedge \cancel{H \wedge X'} \wedge (Z \leftrightarrow \cancel{Z'}) \wedge (Y \leftrightarrow \cancel{Z'})
\end{array}$$

Example (Cont.):

Furthermore,

$$\begin{array}{lll}
[\![\mathsf{app}]\!]^\sharp(Z) & \sqsupseteq & X \wedge Y \wedge Z \\
[\![\mathsf{app}]\!]^\sharp(Z) & \sqsupseteq & \textbf{let } \psi = X \wedge H \wedge X' \wedge Z \wedge Z' \\
 & & \textbf{in } \exists\, H, X', Z'.\; \mathsf{combine}^\sharp_{\ldots}\,(\psi, [\![\mathsf{app}]\!]^\sharp(\mathsf{enter}^\sharp_{\ldots}\,(\psi)))
\end{array}$$

where for $\quad \psi = Z \wedge H \wedge Z' \wedge (X \leftrightarrow X')$:

$$\begin{array}{lll}
\mathsf{enter}^\sharp_{\ldots}\,(\psi) & = & Z \\
\mathsf{combine}^\sharp_{\ldots}\,(\psi, X \wedge Y \wedge Z) & = & X \wedge H \wedge X' \wedge Y \wedge Z \wedge Z'
\end{array}$$

Fixpoint iteration therefore yields:

$$[\![\mathsf{app}]\!]^\sharp\,(X) \;=\; X \wedge (Y \leftrightarrow Z) \qquad [\![\mathsf{app}]\!]^\sharp\,(Z) \;=\; X \wedge Y \wedge Z$$

Example (Cont.):

Furthermore,

$$\begin{array}{lll}
[\![\mathsf{app}]\!]^\sharp(Z) & \sqsupseteq & X \wedge Y \wedge Z \\
[\![\mathsf{app}]\!]^\sharp(Z) & \sqsupseteq & \textbf{let } \psi = X \wedge H \wedge X' \wedge Z \wedge Z' \\
 & & \textbf{in } \exists\, H, X', Z'.\; \mathsf{combine}^\sharp_{\ldots}\,(\psi, [\![\mathsf{app}]\!]^\sharp(\mathsf{enter}^\sharp_{\ldots}\,(\psi)))
\end{array}$$

where for $\quad \psi = Z \wedge H \wedge Z' \wedge (X \leftrightarrow X')$:

$$\begin{array}{lll}
\mathsf{enter}^\sharp_{\ldots}\,(\psi) & = & Z \\
\mathsf{combine}^\sharp_{\ldots}\,(\psi, X \wedge Y \wedge Z) & = & X \wedge H \wedge X' \wedge Y \wedge Z \wedge Z'
\end{array}$$

Fixpoint iteration therefore yields:

$$[\![\mathsf{app}]\!]^\sharp\,(X) \;=\; X \wedge (Y \leftrightarrow Z) \qquad [\![\mathsf{app}]\!]^\sharp\,(Z) \;=\; X \wedge Y \wedge Z$$

888

Example:

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

Then

$$\begin{array}{lll} [\![\mathsf{app}]\!]^\sharp(X) & \sqsupseteq & X \wedge (Y \leftrightarrow Z) \\ [\![\mathsf{app}]\!]^\sharp(X) & \sqsupseteq & \textbf{let } \psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z') \\ & & \textbf{in } \exists\, H, X', Z'.\; \mathsf{combine}^\sharp_{\dots}(\psi, [\![\mathsf{app}]\!]^\sharp(\mathsf{enter}^\sharp_{\dots}(\psi))) \end{array}$$

where for $\psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z')$:

$$\begin{array}{lll} \mathsf{enter}^\sharp_{\dots}(\psi) & = & X \\ \mathsf{combine}^\sharp_{\dots}(\psi, X \wedge (Y \leftrightarrow Z)) & = & (X \wedge H \wedge X' \wedge (Z \leftrightarrow Z') \wedge (Y \leftrightarrow Z') \end{array}$$

Example:

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

Then

$$\begin{array}{lll} [\![\mathsf{app}]\!]^\sharp(X) & \sqsupseteq & X \wedge (Y \leftrightarrow Z) \\ [\![\mathsf{app}]\!]^\sharp(X) & \sqsupseteq & \textbf{let } \psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z') \\ & & \textbf{in } \exists\, H, X', Z'.\; \mathsf{combine}^\sharp_{\dots}(\psi, [\![\mathsf{app}]\!]^\sharp(\mathsf{enter}^\sharp_{\dots}(\psi))) \end{array}$$

where for $\psi = X \wedge H \wedge X' \wedge (Z \leftrightarrow Z')$:

$$\begin{array}{lll} \mathsf{enter}^\sharp_{\dots}(\psi) & = & X \\ \mathsf{combine}^\sharp_{\dots}(\psi, X \wedge (Y \leftrightarrow Z)) & = & (X \wedge H \wedge X' \wedge (Z \leftrightarrow Z') \wedge (Y \leftrightarrow Z') \end{array}$$

887

Example (Cont.):

Furthermore,

$$\begin{array}{lll} [\![\mathsf{app}]\!]^\sharp(Z) & \sqsupseteq & X \wedge Y \wedge Z \\ [\![\mathsf{app}]\!]^\sharp(Z) & \sqsupseteq & \textbf{let } \psi = X \wedge H \wedge X' \wedge Z \wedge Z' \\ & & \textbf{in } \exists\, H, X', Z'.\; \mathsf{combine}^\sharp_{\dots}(\psi, [\![\mathsf{app}]\!]^\sharp(\mathsf{enter}^\sharp_{\dots}(\psi))) \end{array}$$

where for $\psi = Z \wedge H \wedge Z' \wedge (X \leftrightarrow X')$:

$$\begin{array}{lll} \mathsf{enter}^\sharp_{\dots}(\psi) & = & Z \\ \mathsf{combine}^\sharp_{\dots}(\psi, X \wedge Y \wedge Z) & = & X \wedge H \wedge X' \wedge Y \wedge Z \wedge Z' \end{array}$$

Fixpoint iteration therefore yields:

$$[\![\mathsf{app}]\!]^\sharp\,(X) \;=\; X \wedge (Y \leftrightarrow Z) \qquad [\![\mathsf{app}]\!]^\sharp\,(Z) \;=\; X \wedge Y \wedge Z$$

Discussion:

- Exhaustive tabulation of the transformation $[\![\mathsf{app}]\!]^\sharp$ is not feasible.
- Therefore, we rely on demand-driven fixpoint iteration !
- The evaluation starts with the evaluation of the query g, i.e., with the evaluation of $[\![g]\!]^\sharp\,1$.
- The set of inspected fixpoint variables $[\![p]\!]^\sharp\,\psi$ yields a description of all possible calls :-))
- For an efficient representation of functions $\psi \in \mathsf{Pos}$ we rely on binary decision diagrams (BDDs).

Background 6: Binary Decision Diagrams

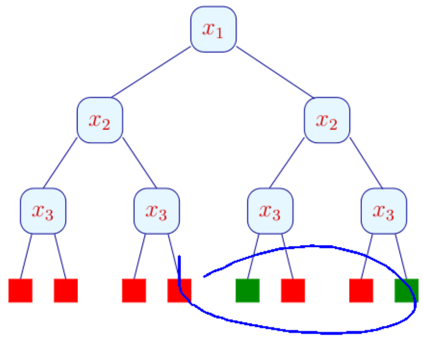
Idea (1):

- 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:

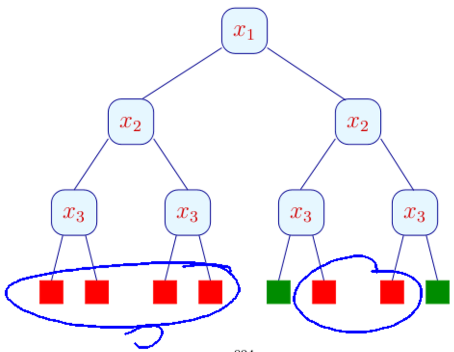
$$
\begin{aligned}
[b]_k &= b \\
[f]_{i-1} &= \textbf{fun } x_i \to \textbf{if } x_i \textbf{ then } [f\ 1]_i \\
&\qquad\qquad\qquad \textbf{else } [f\ 0]_i
\end{aligned}
$$

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

... yields the tree:

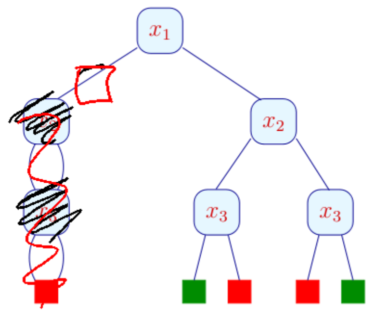


... yields the tree:



Idea (2):

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



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: $\wedge, \vee, \neg, \Rightarrow, \exists\, x_j$ ...

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

// analogous for the remaining operators

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

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

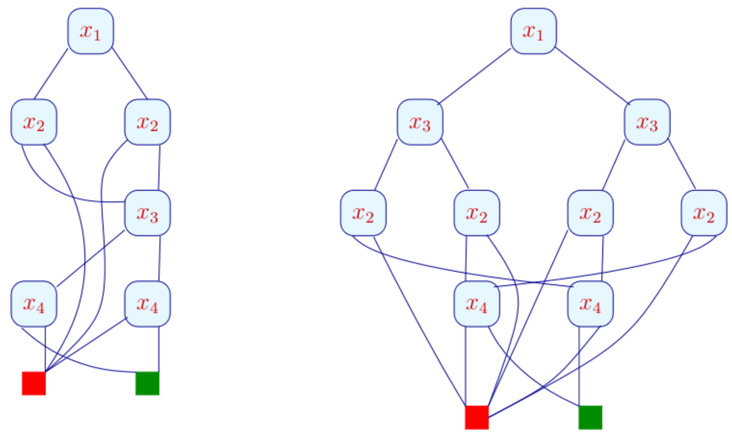
  $\Longrightarrow$

  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.
- Warning: Repeated application of Boolean operations may increase the size dramatically !
- The variable ordering may have a dramatic impact ...

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



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 :-(

899