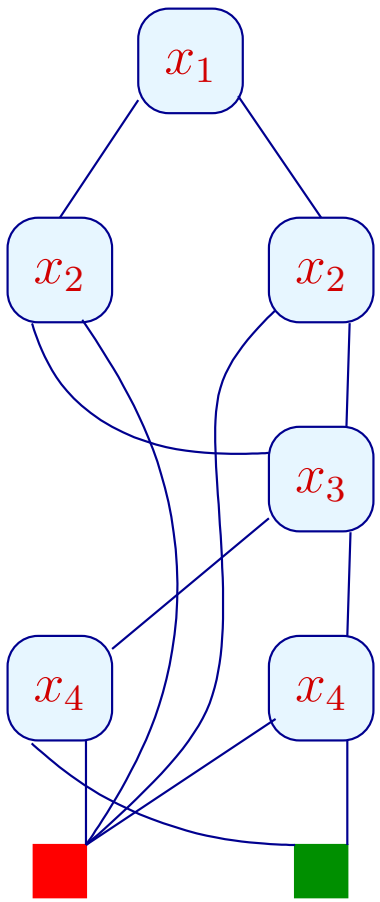
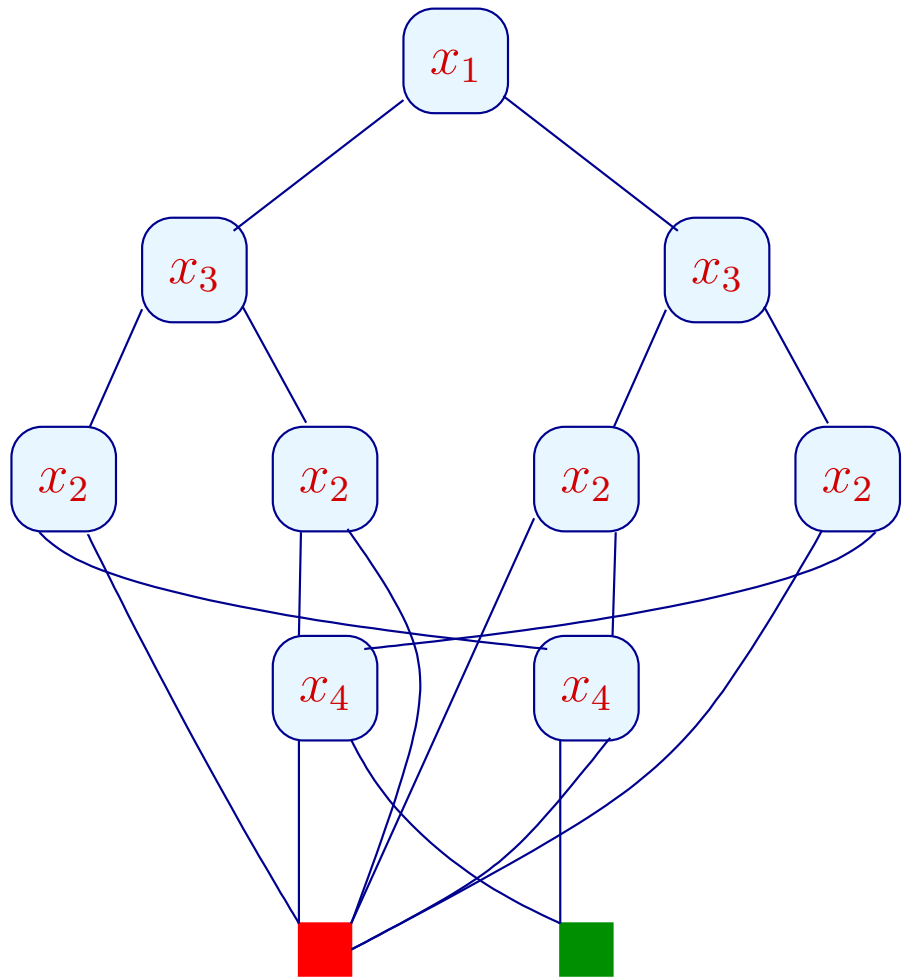
Discussion:

- 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 (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 :-(

Perspectives: Further Properties of Programs

Freeness: Is X_i possibly/always unbound ?

$\Longrightarrow$

If X_i is always unbound, no indexing for X_i is required :-)

If X_i is never unbound, indexing for X_i is complete :-)

Pair Sharing: Are X_i, X_j possibly bound to terms t_i, t_j with

$$Vars(t_i) \cap Vars(t_j) \neq \emptyset \quad ?$$

$\Longrightarrow$

Literals without sharing can be executed in parallel :-)

Remark:

Both analyses may profit from Groundness !

5.2 Types for Prolog

Example:

$$
\begin{array}{lcl}
\mathsf{nat}(X) & \leftarrow & X = 0 \\
\mathsf{nat}(X) & \leftarrow & X = s(Y), \mathsf{nat}(Y) \\
\mathsf{nat\_list}(X) & \leftarrow & X = [] \\
\mathsf{nat\_list}(X) & \leftarrow & X = [H|T], \mathsf{nat}(H), \mathsf{nat\_list}(T)
\end{array}
$$

Discussion

- In Prolog, a type is a set of ground terms with a simple description.
- There is no common agreement what simple means :-)
- One possibility are (non-deterministic) finite tree automata or normal Horn clauses:

$\mathsf{nat\_list}([H\|T])$	$\leftarrow$	$\mathsf{nat}(H), \mathsf{nat\_list}(T)$	normal
$\mathsf{bin}(node(T,T))$	$\leftarrow$	$\mathsf{bin}(T)$	nicht normal
$\mathsf{tree}(node(T_1,T_2))$	$\leftarrow$	$\mathsf{tree}(T_1), \mathsf{tree}(T_2)$	normal

Comparison:

Normal clauses	Tree automaton
unary predicate	state
normal clause	transition
constructor in the head	input symbol
body	pre-condition

General Form:

$$
\begin{array}{lcl}
p(a(X_1, \ldots, X_k)) & \leftarrow & p_1(X_1), \ldots, p_k(X_k) \\
p(X) & \leftarrow & \\
p(b) & \leftarrow &
\end{array}
$$

Properties:

- Types then are in fact regular tree languages ;-)
- Types are closed under intersection:

$$
\begin{array}{lcll}
\langle p, q\rangle(a(X_1, \ldots, X_k)) & \leftarrow & \langle p_1, q_1\rangle(X_1), \ldots, \langle p_k, q_k\rangle(X_k) & \text{if} \\
p(a(X_1, \ldots, X_k)) & \leftarrow & p_1(X_1), \ldots, p_k(X_k) & \text{and} \\
q(a(X_1, \ldots, X_k)) & \leftarrow & q_1(X_1), \ldots, q_k(X_k) &
\end{array}
$$

- Types are also closed under union :-)
- Queries $p(X)$ and $p(t)$ can be decided in polynomial time but:
- ... only in presence of tabulation !

- Or the program is topdown deterministic ...

Example: Topdown vs. Bottom-up

$$
\begin{array}{lcl}
p(a(X_1, X_2)) & \leftarrow & p_1(X_1), p_2(X_2) \\
p(a(X_1, X_2)) & \leftarrow & p_2(X_1), p_1(X_2) \\
p_1(b) & \leftarrow & \\
p_2(c) & \leftarrow &
\end{array}
$$

... is bottom-up, but not topdown deterministic.

There is no topdown deterministic program for this type !

$\Longrightarrow$

Topdown deterministic types are closed under intersection, but not under union !!!

For a set T of terms, we define the set $\Pi(T)$ of paths in terms from T:

$$\begin{aligned}
\Pi(T) &= \bigcup\{\Pi(t) \mid t \in T\} \\
\Pi(b) &= \{b\} \\
\Pi(a(t_1, \ldots, t_k)) &= \{a_j w \mid w \in \Pi(t_j)\} \qquad (k > 0) \\
&\quad // \quad \text{for new unary constructors } a_j
\end{aligned}$$

Example

$$\begin{aligned}
T &= \{a(b, c), a(c, b)\} \\
\Pi(T) &= \{a_1 b, a_2 c, a_1 c, a_2 b\}
\end{aligned}$$

Vice versa from a set P of paths, a set $\Pi^-(P)$ of terms can be recovered:

$$\Pi^-(P) = \{t \mid \Pi(t) \subseteq P\}$$

Example (Cont.):

$$\begin{aligned} P &= \{a_1b, a_2c, a_1c, a_2b\} \\ \Pi^-(P) &= \{a(b,b), a(b,c), a(c,b), a(c,c)\} \end{aligned}$$

The set has become larger !!

Theorem:

Assume that T is a regular set of terms. Then:

- $\Pi(T)$ is regular :-)
- $T \subseteq \Pi^{-}(\Pi(T))$:-)
- $T = \Pi^{-}(\Pi(T))$ iff T is topdown deterministic :-)
- $\Pi^{-}(\Pi(T))$ is the smallest superset of T which is topdown deterministic. :-)

Consequence:

If we are interested in topdown deterministic types, it suffices to determine the set of paths in terms !!!

Example (Cont.):

$$
\begin{array}{lcl}
\mathsf{add}(X, Y, Z) & \leftarrow & X = 0, \mathsf{nat}(Y), Y = Z \\
\mathsf{add}(X, Y, Z) & \leftarrow & \mathsf{nat}(X), X = s(X'), Z = s(Z'), \mathsf{add}(X', Y, Z') \\
\mathsf{mult}(X, Y, Z) & \leftarrow & X = 0, \mathsf{nat}(Y), Z = 0 \\
\mathsf{mult}(X, Y, Z) & \leftarrow & \mathsf{nat}(X), X = s(X'), \mathsf{mult}(X', Y, Z'), \mathsf{add}(Z', Y, Z)
\end{array}
$$

Question:

Which run-time checks are necessary?

Idea:

- Approximate the semantics of predicates by means of topdown-deterministic regular tree languages !
- Alternatively: Approximate the set of paths in the semantics of predicates by regular word languages !

Idea:

- All predicates $p/k, k > 0$, are split into predicates $p_1/1, \ldots, p_k/1$.

Semantics:

Let $\mathcal{C}$ denote a set of clauses.

The set $[\![p]\!]_{\mathcal{C}}$ is the set of tuples of ground terms $(s_1, \ldots, s_k)$, for which $p(s_1, \ldots, s_k)$ is provable :-)

$[\![p]\!]_{\mathcal{C}}$ (p predicate) thus is the smallest collection of sets of tuples for which:

$$\sigma(\underline{t}) \in [\![p]\!]_{\mathcal{C}} \quad \text{when ever} \quad \forall\, i.\ \sigma(\underline{t}_i) \in [\![p_i]\!]_{\mathcal{C}}$$

for clauses $p(\underline{t}) \leftarrow p_1(\underline{t}_1), \ldots, p_n(\underline{t}_n) \in \mathcal{C}$ and ground substitutions σ.

Approximation of Paths:

Every clause

$$p(t_1, \ldots, t_k) \leftarrow \alpha$$

is approximated by the clauses:

$$\begin{array}{lcll} p_j(w) & \leftarrow & \bigwedge \Pi(\alpha) & \text{where} \\ \Pi(g_1, \ldots, g_m) & = & \Pi(g_1) \cup \ldots \cup \Pi(g_m) & \\ \Pi(q(s_1, \ldots, s_n)) & = & \{q_i(w) \mid w \in \Pi(s_i)\} & \end{array}$$

$(j = 1, \ldots, k, w \in \Pi(t_j))$.

Example:

$$\begin{array}{lcl} \mathsf{add}(0, Y, Y) & \leftarrow & \mathsf{nat}(Y) \\ \mathsf{add}(s(X), Y, s(Z)) & \leftarrow & \mathsf{add}(X, Y, Z) \end{array}$$

yields:

$$
\begin{array}{lcl}
\mathsf{add}_1(0) & \leftarrow & \mathsf{nat}_1(Y) \\
\mathsf{add}_2(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\mathsf{add}_3(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\\
\mathsf{add}_1(s_1\, X) & \leftarrow & \mathsf{add}_1(X), \mathsf{add}_2(Y), \\
 & & \mathsf{add}_3(Z) \\
\mathsf{add}_2(Y) & \leftarrow & \mathsf{add}_1(X), \mathsf{add}_2(Y), \\
 & & \mathsf{add}_3(Z) \\
\mathsf{add}_3(s_1\, Z) & \leftarrow & \mathsf{add}_1(X), \mathsf{add}_2(Y), \\
 & & \mathsf{add}_3(Z)
\end{array}
$$

Discussion:

- Every literal has at most one occurrence of a variable.
- The literals $q_j(w_j Y)$ where the variable Y does not occur in the head, represent tests:

  If there is a w with $w_j w \in [\![q_j]\!]_{\mathcal{C}^\sharp}$ for all such j, then we can cancel these literals.

  If there is no such w, then we can cancel the clause ...

... in the Example:

The literals:

$$\mathsf{add}_1(X), \mathsf{add}_2(Y), \mathsf{add}_3(Z)$$

are all satisfiable :-)

We conclude:

$$
\begin{array}{lcl}
\mathsf{add}_1(0) & \leftarrow & \\
\mathsf{add}_2(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\mathsf{add}_3(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\\
\mathsf{add}_1(s_1\, X) & \leftarrow & \mathsf{add}_1(X) \\
\mathsf{add}_2(Y) & \leftarrow & \mathsf{add}_2(Y) \\
\mathsf{add}_3(s_1\, Z) & \leftarrow & \mathsf{add}_3(Z)
\end{array}
$$

We conclude:

$$
\begin{array}{lcl}
\mathsf{add}_1(0) & \leftarrow & \\
\mathsf{add}_2(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\mathsf{add}_3(Y) & \leftarrow & \mathsf{nat}_1(Y) \\
\\
\mathsf{add}_1(s_1\, X) & \leftarrow & \mathsf{add}_1(X) \\
\\
\mathsf{add}_3(s_1\, Z) & \leftarrow & \mathsf{add}_3(Z)
\end{array}
$$

We verify:

Theorem

Assume that $\mathcal{C}$ is a set of clauses.

Let $\mathcal{C}^\sharp$ denote the corresponding set of clauses for the paths.

Then for all predicates p/k:

$$\Pi([\![p]\!]_{\mathcal{C}}) \;\subseteq\; [\![p_1]\!]_{\mathcal{C}^\sharp} \cup \ldots \cup [\![p_k]\!]_{\mathcal{C}^\sharp}$$

Proof:

Induction on the approximations of the respective fixpoints :-)

A set of clauses with unary predicates and unary constructors is called Alternating Pushdown System (APS).

Theorem

- Every APS is equivalent to a simple APS of the form:

$$\begin{array}{lcl} p(a\,X) & \leftarrow & p_1(X), \ldots, p_r(X) \\ p(X) & \leftarrow & \\ p(b) & \leftarrow & \end{array}$$

- Every APS is equivalent to a normal APS of the form:

$$\begin{array}{lcl} p(a\,X) & \leftarrow & p_1(X) \\ p(X) & \leftarrow & \\ p(b) & \leftarrow & \end{array}$$

Step 1: Removal of complicated heads:

For $w = a^{(1)} \dots a^{(m)}$ $(m > 1)$ we replace

$$p(w\,X) \leftarrow rhs \qquad \text{with:}$$

$$\begin{aligned}
p(a^{(1)}\,X) &\leftarrow p_2(X) \\
p_2(a^{(2)}\,X) &\leftarrow p_3(X) \\
&\dots \\
p_{m-1}(a^{(m-1)}\,X) &\leftarrow p_m(X) \\
p_m(a^{(m)}\,X) &\leftarrow rhs
\end{aligned}$$

// p_j all new

Step 1 (Cont.): Removal of complicated heads:

For $w = a^{(1)} \ldots a^{(m)} b \quad (m > 0)$ we replace

$$
\begin{array}{lcll}
p(w) & \leftarrow & rhs & \text{with:} \\
p(a^{(1)}\, X) & \leftarrow & p_2(X) & \\
p_2(a^{(2)}\, X) & \leftarrow & p_3(X) & \\
 & \ldots & & \\
p_{m-1}(a^{(m-1)}\, X) & \leftarrow & p_m(X) & \\
p_m(a^{(m)}\, X) & \leftarrow & p_{m+1}(X) & \\
p_{m+1}(b) & \leftarrow & rhs & \\
 & & // \quad p_j \text{ all new} &
\end{array}
$$

Step 2: Splitting

We separate independent parts of pre-conditions into auxiliary predicates:

$$head \leftarrow rest,\ p_1(w_1\, X), \ldots, p_m(w_m\, X)$$

(X does not occur in $head,\ rest$)

is replaced with:

$$head \leftarrow rest, q()$$

$$q() \leftarrow p_1(w_1\, X), \ldots, p_m(w_m\, X)$$

for a new predicate $q/0$.