Implementation Techniques for Prolog

Andreas Krall

Compilers and Languages
TU Wien
nrev([],[]).
nrev([Head|Tail],Rev) :-
    nrev(Tail,TRev),
    append(TRev,[Head],Rev).

append([],L,L).
append([H|L1],L2,[H|L3]) :-
    append(L1,L2,L3).
Basic Execution Model

Resolution: search clauses top-down
evaluate goals left to right

Unification: constant $\Leftrightarrow$ constant
structure $\Leftrightarrow$ structure
variable $\Leftrightarrow$ variable
variable $\Leftrightarrow$ constant/structure
atom
integer
structure
unbound variable
reference

<table>
<thead>
<tr>
<th>tag</th>
<th>value</th>
</tr>
</thead>
</table>

Tag Representations

whole word ⇔ part of a word

fixed size ⇔ variable size

most significant bits ⇔ least significant bits

minimize tag manipulation
The Representation of Structures

(ref

functor ./2 int 1 ref

functor ./2 int 2 atom []

functor ./2 int 1 functor ./2 int 2 atom []

functor ./2 int 1 ref

functor ./2 int 2 atom []

functor ./2 int 1 functor ./2 int 2 atom []

Andreas Krall

WAM & VAM
Tagged Pointer Representations

unbound variable as self reference

Andreas Krall
WAM & VAM
simple recursive unification algorithm

occur check

infinite terms
Prolog Data Areas

Depth first traversal of proof tree results in a stack

\[
\begin{align*}
x(X) & : - \ a(A). & \ b(s(0)). \\
a(C) & : - \ b(C), \ c(C). & \ c(s(0)). \\
a(D) & : - \ b(D), \ d(D). & \ d(s(0)).
\end{align*}
\]

<table>
<thead>
<tr>
<th>x(X) : - a(A)</th>
<th>x(X) : - a(A)</th>
<th>x(X) : - a(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(C) : - b(C), c(C)</td>
<td></td>
<td>a(C) : - b(C), d(C)</td>
</tr>
<tr>
<td>b(s(0))</td>
<td></td>
<td>b(s(0))</td>
</tr>
<tr>
<td>c(s(0))</td>
<td></td>
<td>d(s(0))</td>
</tr>
</tbody>
</table>

Andreas Krall

WAM & VAM 9/33
Prolog Data Areas

- copy stack
- trail
- environment stack
- code area

Andreas Krall
WAM & VAM
deterministic stack frame

<table>
<thead>
<tr>
<th>callers goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>callers frame</td>
</tr>
<tr>
<td>variable(_0)</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>variable(_n)</td>
</tr>
</tbody>
</table>

choice point

<table>
<thead>
<tr>
<th>alternative clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy of top of trail</td>
</tr>
<tr>
<td>copy of top of copy stack</td>
</tr>
<tr>
<td>previous choice point</td>
</tr>
</tbody>
</table>
Representation of Terms

structure copying

structure sharing

\[ f(1, X, 3, Y) \]

\[ \text{skeleton} \quad \text{environment} \]

\[ \begin{array}{c|c} 
\text{undef} \\
\text{int} \quad 2 
\end{array} \]
representation for programs

term representation

break up unification into its atomic parts

abstract machine
Variable Classification: void variables
temporary variables
local variables
first and further occurrences

Clause Indexing: first argument indexing
hash table or search tree
Problems:
- dangling References
- copying overhead
- stack trimming

Andreas Krall
Problems:
- dangling References
- copying overhead
- stack trimming
The Warren Abstract Machine (WAM) resembles implementation of imperative languages parameter passing argument registers stack, if out of registers one instruction pointer one frame pointer
WAM Data Areas and Machine Registers

- **TR**: top of trail
- **A**: top of stack
- **E**: current environment pointer
- **B**: most recent choice point
- **H**: top of heap
- **S**: structure pointer
- **P**: program pointer
- **CP**: continuation program pointer
put instructions (put_value Vn,Ri)

get instructions (get_variable Vn,Ri)

unify instructions (unify_constant C) in read or write mode

procedural instructions (call P,N)

indexing instructions (switch_on_constant N,T)
try_me_else
<group_1>
retry_me_else
<group_2>
...
<group_n>
trust_me_else_fail

switch_on_term Variable,Constant,List,Structure
switch_on_constant Size,Table
switch_on_structure Size,Table
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B’</td>
<td>previous choice point</td>
</tr>
<tr>
<td>H’</td>
<td>top of heap</td>
</tr>
<tr>
<td>T’</td>
<td>top of trail</td>
</tr>
<tr>
<td>BP’</td>
<td>retry program pointer</td>
</tr>
<tr>
<td>CP’</td>
<td>continuation program pointer</td>
</tr>
<tr>
<td>E’</td>
<td>environment pointer</td>
</tr>
<tr>
<td>A1’</td>
<td>... argument registers</td>
</tr>
<tr>
<td>An’</td>
<td>...</td>
</tr>
</tbody>
</table>

saving of argument registers enables last-call optimization for indeterministic procedures
more temporary variables
unsafe variables
Optimizing the WAM

compilation of read/write mode

flag for each structure

propagate write mode down and read mode up

concept of uninitialized variable

must be initialized after the subgoal it was created is proven
nrev([],[]).

nrev([Head|Tail],Rev) :-
    nrev(Tail, TRev),
    append(TRev, [Head], Rev).

nrev([],[],Cont) :- call(Cont).

nrev([Head|Tail],Rev,Cont) :-
    nrev(Tail, TRev,
    append(TRev, [Head], Rev, Cont)).

simplified BinWAM
The Vienna Abstract Machine (VAM)

eliminates the parameter passing bottleneck

partial evaluation of a call

$VAM_{2P}$ (2 instruction pointers) aimed for interpreters

$VAM_{1P}$ (1 instruction pointer) aimed for compilers

$VAM_{AI}$ (2 instruction pointers) aimed for abstract interpretation
<table>
<thead>
<tr>
<th>unification instructions</th>
<th>resolution instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const C</code></td>
<td><code>goal P</code></td>
</tr>
<tr>
<td><code>nil</code></td>
<td><code>nogoal</code></td>
</tr>
<tr>
<td><code>list</code></td>
<td><code>cut</code></td>
</tr>
<tr>
<td><code>struct F</code></td>
<td><code>builtin I</code></td>
</tr>
<tr>
<td><code>void</code></td>
<td><code>termination instructions</code></td>
</tr>
<tr>
<td><code>fsttmp Xn</code></td>
<td><code>call</code></td>
</tr>
<tr>
<td><code>nxttmp Xn</code></td>
<td><code>lastcall</code></td>
</tr>
<tr>
<td><code>fstvar Yn</code></td>
<td></td>
</tr>
<tr>
<td><code>nxtvar Yn</code></td>
<td></td>
</tr>
</tbody>
</table>
append([], nil)
  L, fsttmp L
  L, nxttmp L
).
nogoal

append([H| L1],
  lst
  H| L2,
  lst
  [H| L3]) :-
  fstvar L2
  nxtttmp H
  lst
  nxtttmp H
  lst
  goal 3, append
  L1, nxtvar L1
  L2, nxtvar L2
  L3, nxtvar L3
).
lastcall
VAM\textsubscript{2P} Instruction Fetch and Decode

<table>
<thead>
<tr>
<th>VAM</th>
<th>WAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>subgoal</td>
<td>put_variable X,A0</td>
</tr>
<tr>
<td>head</td>
<td>get_constant 3,A0</td>
</tr>
</tbody>
</table>

switch (*goalptr++ + *headptr)
VAM Data Areas

- copy stack
  - copyptr
- trail
  - trailptr
- environment stack
  - choicepntptr
  - goalframeptr
  - headframeptr
  - stackptr
- code area
  - goalptr
  - headptr
### VAM$_2$P Stack Frame

**Stack Frame**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>goalptr’</td>
<td>continuation code pointer</td>
</tr>
<tr>
<td>goalframeptr’</td>
<td>continuation frame pointer</td>
</tr>
<tr>
<td>variable$_0$</td>
<td>local variables</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>variable$_n$</td>
<td></td>
</tr>
</tbody>
</table>

**Choice Point**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trailptr’</td>
<td>copy of top of trail</td>
</tr>
<tr>
<td>copypntptr’</td>
<td>copy of top of copy stack</td>
</tr>
<tr>
<td>headptr’</td>
<td>alternative clauses</td>
</tr>
<tr>
<td>goalptr’</td>
<td>restart code pointer</td>
</tr>
<tr>
<td>goalframeptr’</td>
<td>restart frame pointer</td>
</tr>
<tr>
<td>choicepntptr’</td>
<td>previous choice point</td>
</tr>
</tbody>
</table>
Clause Indexing in the VAM$_{2P}$

procedure

const 1

const 2

const 3

fstvar X

const 4

const 5
instruction combination at compile time

one instruction pointer

instruction elimination

no temporary variables

compile time clause indexing

different forms of last call optimization

preventing code explosion by dummy calls
Abstract Interpretation

collects information about types modes, trailing, reference chain length and aliasing of variables

AI executes a program over an abstract domain

computes fixpoint

iterates until information does not change ⇒ completeness

finite domain ⇒ termination

practical implementation by an abstract machine for AI ⇒ VAM_{AI}
The VAM\textsubscript{AI} based on VAM\textsubscript{2P} collecting information for VAM\textsubscript{1P}

two instruction pointers

clause heads are duplicated for a more precise analysis

reference chain lengths 0, 1, >1

types: nil, list, structure, atom, integer, unbound

alias sets for variables

local variables have additional fields for the collected information

incremental abstract interpretation
What is important for a fast Prolog System

- selection of an abstract machine and its optimizations
  - efficient tag representation
  - variable classification
  - clause indexing
  - last call optimization
  - abstract interpretation
  - machine code generation