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 ⇔ constant
structure ⇔ structure
variable ⇔ variable
variable ⇔ constant/structure
atom
integer
structure
unbound variable
reference

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

functor ./2 int 1 ref

functor ./2 int 2 atom []

functor ./2 int 1 ref

functor ./2 int 2 atom []

ref

ref
Tagged Pointer Representations

```
list ]---[ int 3 atom []
```

unbound variable as self reference

```
ref ]---[
```
simple recursive unification algorithm

occur check

infinite terms
depth first traversal of proof tree results in a stack

\[
\begin{align*}
\text{x}(X) & :\!- \ a(X). & \quad \text{b}(s(0)). \\
\text{a}(C) & :\!- \ b(C), \ c(C). & \quad \text{c}(s(0)). \\
\text{a}(D) & :\!- \ b(D), \ d(D). & \quad \text{d}(s(0)).
\end{align*}
\]
Prolog Data Areas

- copy stack
- trail
- environment stack
- code area
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

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

<table>
<thead>
<tr>
<th>skeleton</th>
<th>environment</th>
</tr>
</thead>
</table>

| undef    | int | 2 |

Andreas Krall  WAM & VAM  12/33
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
Last Call Optimization

Problems:
- dangling References
- copying overhead
- stack trimming

Andreas Krall
Last Call Optimization

Problems:
- dangling References
- copying overhead
- stack trimming

called clause
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
WAM Instruction Set

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>B’</th>
<th>previous choice point</th>
</tr>
</thead>
<tbody>
<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>...</td>
<td></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
VAM$_{2P}$ Instructions

unification instructions

const C
nil
list
struct F
void
fsttmp Xn
nxttmp Xn
fstvar Yn
nxtvar Yn

resolution instructions

goal P
nogoal
cut
builtin I
termination instructions

call
lastcall
append([], nil).

append(L, list.
append(H| L1, list.
append(H| L3) :-
append(L1, lastcall.

Andreas Krall
VAM₂P 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
  - choicepntptr
- environment stack
  - goalframeptr
  - headframeptr
  - stackptr
- code area
  - goalptr
  - headptr
### VAM₂P 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>variable₀</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>variableₙ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>local variables</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>copyyptr’</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>
The VAM$_{1P}$

- 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 $\Rightarrow$ completeness

finite domain $\Rightarrow$ termination

practical implementation by an abstract machine for AI $\Rightarrow$ VAM$\text{AI}$
The VAM$_A$I

based on VAM$_{2P}$ collecting information for VAM$_{1P}$

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