Implementation Techniques for Prolog

Andreas Krall

Compilers and Languages
TU Wien

Example Prolog Program

```
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

Representation of Data

atom
integer
structure
unbound variable
reference

tag	value

Tag Representations

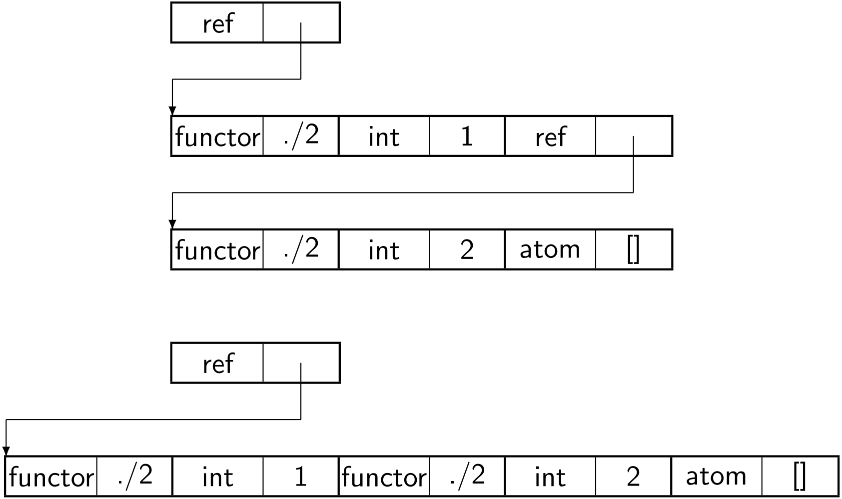
whole word ⇔ part of a word

fixed size ⇔ variable size

most significant bits ⇔ least significant bits

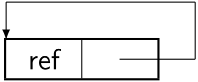
mininize tag manipulation

The Representation of Structures



Tagged Pointer Representations

unbound variable as self reference



Unification Algorithm

simple recursive unification algorithm

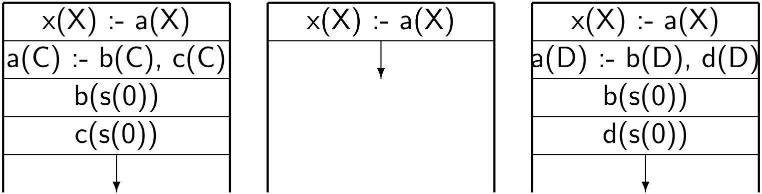
occur check

infinite terms

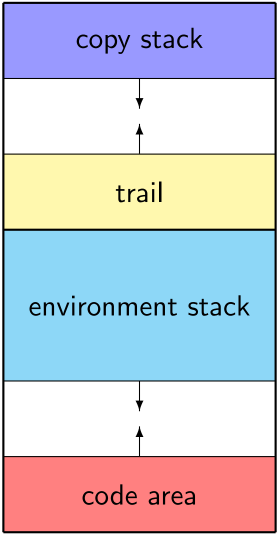
Prolog Data Areas

depth first traversal of proof tree results in a stack

```
x(X) :- a(X).           b(s(0)).
a(C) :- b(C), c(C).     c(s(0)).
a(D) :- b(D), d(D).     d(s(0)).
```



Prolog Data Areas



Stack Frame

deterministic stack frame

callers goal
callers frame
variable_0 ... variable_n

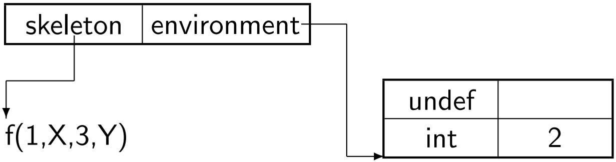
choice point

alternative clauses
copy of top of trail
copy of top of copy stack
previous choice point

Representation of Terms

structure copying

structure sharing



Interpreters and Compilers

representation for programs

term representation

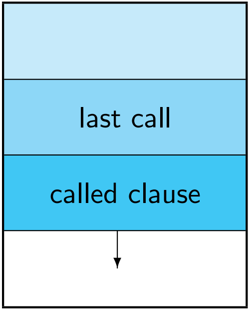
break up unification into its atomic parts

abstract machine

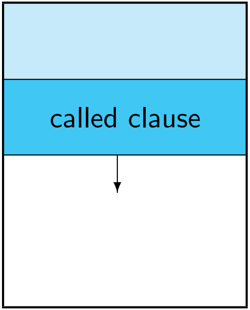
Abstract Machine

Variable Classification:	void variables temporary variables local variables first and further occurences
Clause Indexing:	first argument indexing hash table or search tree

Last Call Optimzation



Last Call Optimzation



Problems:

- dangling References
- copying overhead
- stack trimming

The Warren Abstract Machine (WAM)

resembles implementation of imperative languages

parameter passing

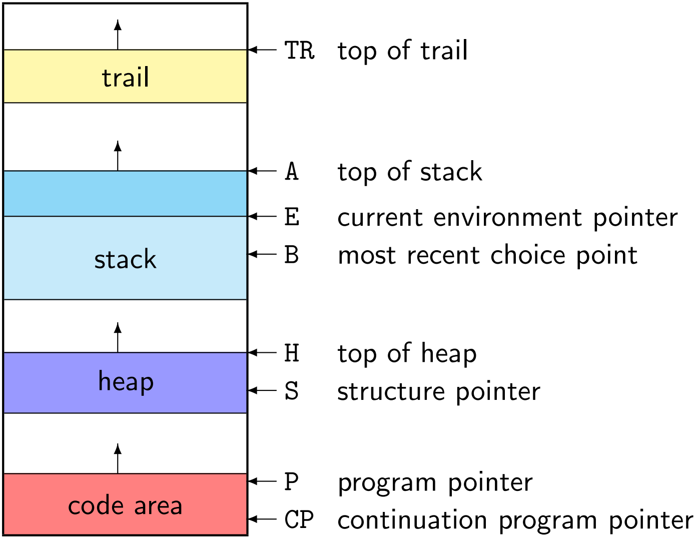
argument registers

stack, if out of registers

one instruction pointer

one frame pointer

WAM Data Areas and Machine Registers



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`)

WAM Indexing Instructions

```
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
```

WAM Choicepoint

B'	previous choice point
H'	top of heap
T'	top of trail
BP'	retry program pointer
CP'	continuation program pointer
E'	environment pointer
A1'	argument registers
...	
An'	

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

Binary Prolog

```
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
```

VAM_{2P} Translation

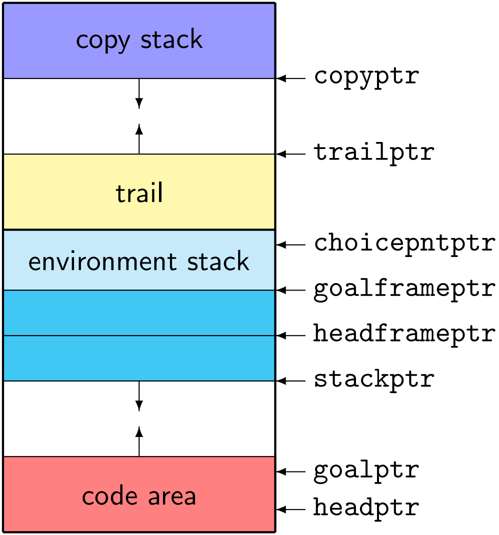
`append([],`	`nil`
`L,`	`fsttmp L`
`L`	`nxttmp L`
`).`	`nogoal`
`append([`	`list`
`H|`	`fsttmp H`
`L1],`	`fstvar L1`
`L2,`	`fstvar L2`
`[`	`list`
`H|`	`nxttmp H`
`L3]) :-`	`fstvar L3`
`append(`	`goal 3,append`
`L1,`	`nxtvar L1`
`L2,`	`nxtvar L2`
`L3`	`nxtvar L3`
`).`	`lastcall`

VAM_{2P} Instruction Fetch and Decode

	VAM	WAM
subgoal	`fstvar X`	`put_variable X,A0`
head	`const 3`	`get_constant 3,A0`

```
switch (*goalptr++ + *headptr++)
```

VAM Data Areas



VAM_{2P} Stack Frame

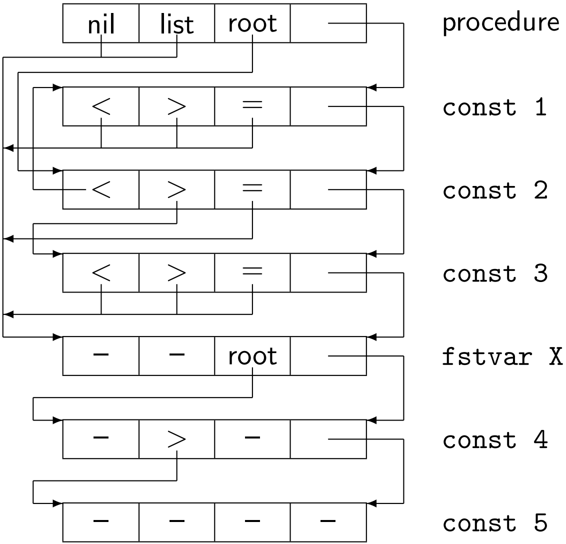
stack frame

goalptr'	continuation code pointer
goalframeptr'	continuation frame pointer
$variable_0$ ... $variable_n$	local variables

choice point

trailptr'	copy of top of trail
copyptr'	copy of top of copy stack
headptr'	alternative clauses
goalptr'	restart code pointer
goalframeptr'	restart frame pointer
choicepntptr'	previous choice point

Clause Indexing in the VAM_{2P}



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_{AI}

The VAM_{AI}

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