

Integrated Logic Systems

Part 1: Deduction Systems

This part is concerned with the design
and use of various deduction systems:

Prolog

Tableaux-Prover

Answer Set Programming

Roadmap

Prolog

- Efficient implementation of terms, unification
- Derivations
- Search (Backtracking)

Tableaux-Prover

- Normal form transformation
- Tableau Calculus
- Application: Description Logics

Answer Set Programming

- Stable model semantics for logic programs
- Implementation techniques
- Applications

Chapter Overview

Introducing the Warren Abstract Machine (WAM)

Language L_0 : handling terms

Compiling L_0 -queries and L_0 -programs

Unification

From L_0 to L_1 : handling atoms

Special data structures: constants and lists

Warren Abstract Machine

The **WAM** is the standard for all existing Prolog implementations.

It provides an abstract method for compiling Prolog programs and queries into basic, “assembler-like” commands.

Knowing the WAM enables you to
understand specific behaviors of a Prolog system
design and implement basic functionalities of deduction systems
- term unification
- derivations (= sequences of deduction steps)
- search (via backtracking)

The Basic Picture

```
member(X, [X|_]) .  
member(X, [_|Y]) :- member(X, Y) .  
subset([X|Y], Z) :- member(X, Z) ,  
                 ...  
disjoint(X, Y) :- ...  
...
```

```
?- subset([2|X], [1, Y, 3]) .
```

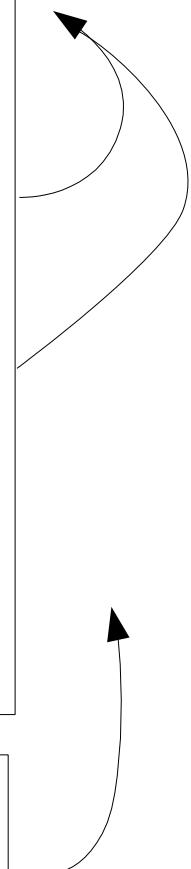


WAM commands

```
member/2: ...  
m2: ...  
call member/2  
subset/2: ...  
call member/2  
...  
disjoint/2: ...  
...
```



```
...  
call subset/2
```



The Language L_0

Consider a term alphabet.

A **program** is a term.

A **query** is a term.

Intended meaning:

“query” t fails wrt “program” s if s and t are not unifiable

otherwise t succeeds with a binding of the variables in t obtained by unifying it with s

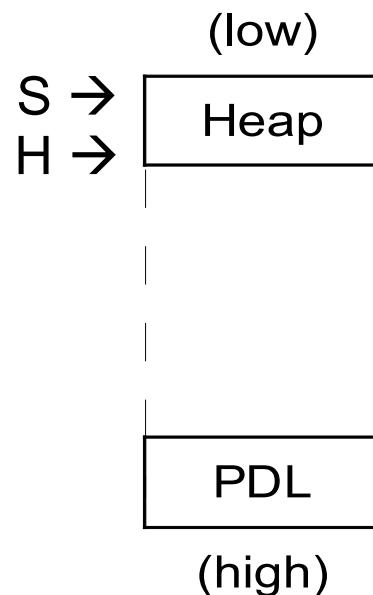
Example:

`p(f(X), h(Y, f(a)), Y).`

`?- p(Z, h(Z, W), f(W)).`

WAM₀ Memory Layout

Registers



S – special register,
pointing to “sub-term”

H – pointer to top of heap

Push-down-list

Term registers:
 X_1, X_2, \dots

Heap Representation of Terms

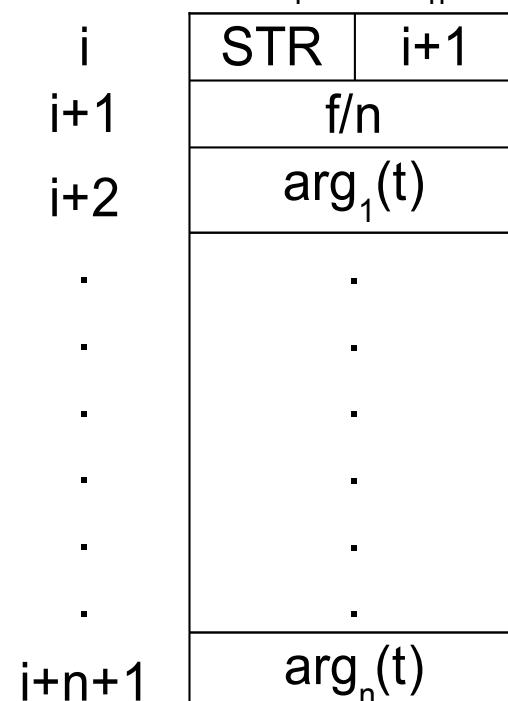
Unbound variable:



Bound variable:



Non-variable term $t = f(t_1, \dots, t_n)$:



Example

0	STR	1
1		h/2
2	REF	2
3	REF	3
4	STR	5
5		f/1
6	REF	3
7	STR	8
8		p/3
9	REF	2
10	STR	1
11	STR	5

Heap representation of
 $p(Z, h(Z,W), f(W))$

Compiling a Query

For processing a query term, there is a sufficient number of registers X_1, X_2, \dots

Register X_1 is allocated to the outermost term.

The same register is allocated to all occurrences of a given variable.

A term is seen as a set of “flattened” equations $X_i = f(X_{i_1}, \dots, X_{i_n})$

Processing Flattened Equations

The equations need to be ordered

$$X_1 = f_1(\vec{X}_1), \dots, X_k = f_k(\vec{X}_k)$$

such that no X_i occurs in $\vec{X}_1, \dots, \vec{X}_{i-1}$

An equation $X_i = f(X_{i_1}, \dots, X_{i_n})$ is processed as:

1. `put_structure f/n, Xi`
2. `set_variable Xi1 or set_value Xi1`

⋮

- n+1. `set_variable Xin or set_value Xin`

`set_variable Xi` used if X_i has not been processed before

`set_value Xi` used if X_i has been processed before

WAM₀ Machine Instructions (I)

put_structure $f/n, X_i \equiv \text{HEAP}[H] \leftarrow \langle \text{STR}, H+1 \rangle;$

$\text{HEAP}[H+1] \leftarrow f/n;$

$X_i \leftarrow \text{HEAP}[H];$

$H \leftarrow H+2;$

set_variable $X_i \equiv \text{HEAP}[H] \leftarrow \langle \text{REF}, H \rangle;$

$X_i \leftarrow \text{HEAP}[H];$

$H \leftarrow H+1;$

set_value $X_i \equiv \text{HEAP}[H] \leftarrow X_i;$

$H \leftarrow H+1;$

Compiling a Program

The flattened equations need to be ordered

$$X_1 = f_1(\vec{X}_1), \dots, X_k = f_k(\vec{X}_k)$$

such that no X_i occurs in $\vec{X}_{i+1}, \dots, \vec{X}_k$

An equation $X_i = f(X_{i_1}, \dots, X_{i_n})$ is processed as:

1. `get_structure f/n, Xi`
2. `unify_variable Xi1 or unify_value Xi1`

⋮

- n+1. `unify_variable Xin or unify_value Xin`

`unify_variable Xi` used if X_i has not been processed before

`unify_value Xi` used if X_i has been processed before

Dereferencing (Through a Chain of Variable References)

```
function deref(a: address): address;  
begin  
    ⟨tag,value⟩ ← STORE[a];  
    if (tag=REF) ∧ (value ≠ a)  
        then return deref (value)  
    else return a  
end deref
```

WAM₀ Machine Instructions (II)

```
get_structure f/n, Xi ≡ addr ← deref(Xi);  
      case STORE[addr] of  
        <REF,_> : HEAP[H] ← <STR,H+1>;  
                    HEAP[H+1] ← f/n;  
                    bind(addr,H);  
                    H ← H+2;  
                    mode ← write;  
        <STR,a> : if HEAP[a] = f/n  
                    then  
                      begin  
                        S ← a+1;  
                        mode ← read  
                      end  
                    else fail ← true;  
        other: fail ← true;  
      endcase
```

Binding in WAM₀

```
procedure bind( $a_1, a_2$ : address);  
begin  
    case  $\leftarrow \text{STORE}[a_1]$  of  
        <REF,_> :  $\text{STORE}[a_1] \leftarrow \text{STORE}[a_2]$ ;  
        other      :  $\text{STORE}[a_2] \leftarrow \text{STORE}[a_1]$ ;  
    endcase;  
end bind
```

WAM₀ Machine Instructions (III)

unify_variable $X_i \equiv \text{case mode of}$

read: $X_i \leftarrow \text{HEAP}[S];$

write: $\text{HEAP}[H] \leftarrow \langle \text{REF}, H \rangle;$

$X_i \leftarrow \text{HEAP}[H];$

$H \leftarrow H+1;$

endcase;

$S \leftarrow S+1$

unify_value $X_i \equiv \text{case mode of}$

read: $\text{unify}(X_i, S);$

write: $\text{HEAP}[H] \leftarrow X_i;$

$H \leftarrow H+1;$

endcase;

$S \leftarrow S+1$

Unification (Part I)

```
procedure unify( $a_1, a_2$ : address);  
  begin  
    push( $a_1$ , PDL); push( $a_2$ , PDL);  
    fail  $\leftarrow$  false;  
    while  $\neg(\text{empty}(\text{PDL}) \vee \text{fail})$  do  
      begin  
         $d_1 \leftarrow \text{deref}(\text{pop}(\text{PDL}))$ ;  $d_2 \leftarrow \text{deref}(\text{pop}(\text{PDL}))$ ;  
        if  $d_1 \neq d_2$  then  
          begin  
             $\langle t_1, v_1 \rangle \leftarrow \text{STORE}[d_1]$ ;  
             $\langle t_2, v_2 \rangle \leftarrow \text{STORE}[d_2]$ ;  
            if ( $t_1 = \text{REF}$ )  $\vee$  ( $t_2 = \text{REF}$ )  
              then bind( $d_1, d_2$ )
```

Unification (Part II)

```
else
begin
     $f_1 / n_1 \leftarrow \text{STORE}[v_1]; f_2 / n_2 \leftarrow \text{STORE}[v_2];$ 
    if ( $f_1 = f_2 \wedge n_1 = n_2$ )
        then
            for  $i \leftarrow 1$  to  $n_1$  do
                begin
                    push( $v_1 + i$ , PDL);
                    push( $v_2 + i$ , PDL)
                end
            else fail  $\leftarrow$  true
        end
    end
end
end unify
```

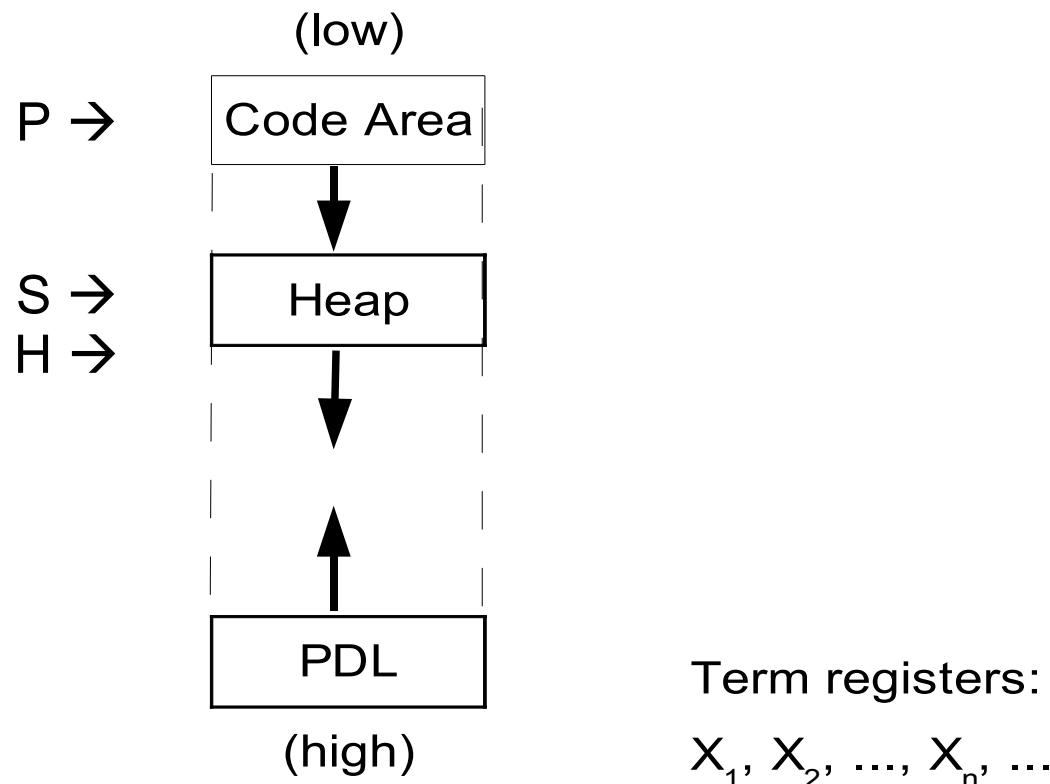
From L_0 to L_1

A program is a set of **atoms**, with at most one for each predicate name.

A query is an **atom**.

WAM₁ Memory Layout

Registers



Control Instructions

call $p/n \equiv P \leftarrow @(\bar{p}/\bar{n});$

where $@(\bar{p}/\bar{n})$ stands for the address in the code area of the instructions for the clause defining predicate p/n .

If p/n has not been defined, failure occurs.

proceed \equiv no-op; (for the moment)

Argument Registers

The first n registers X_1, \dots, X_n are systematically allocated to the arguments of an n -ary predicate.

To emphasize this, we write A_i if X_i is used as argument register.

Example: $A_1, A_2, A_3, X_4, X_5, X_6, X_7$ when processing a predicate
with 3 arguments

Now also a variable can occur on the right hand side of a flattened equation.

Instructions for Variables as Roots

`put_variable X_n, A_i`

handles a root variable which occurs for the first time in a query

`put_value X_n, A_i`

handles a root variable which previously occurred in a query

`get_variable X_n, A_i`

handles a root variable which occurs for the first time in a program atom

`get_value X_n, A_i`

handles a root variable which previously occurred in a program atom

WAM₁ Machine Instructions

put_variable $X_n, A_i \equiv \text{HEAP}[H] \leftarrow \langle \text{REF}, H \rangle;$

$X_n \leftarrow \text{HEAP}[H];$

$A_i \leftarrow \text{HEAP}[H];$

$H \leftarrow H+1;$

put_value $X_n, A_i \equiv A_i \leftarrow X_n;$

get_variable $X_n, A_i \equiv X_n \leftarrow A_i;$

get_value $X_n, A_i \equiv \text{unify}(X_n, A_i);$

Special Datastructures: Heap Representation with Constants

7	STR	8
g/1		
9	CON	a
10	STR	11
f/2		
12	CON	b
13	STR	8

Heap representation of
 $f(b, g(a))$

Instructions for Constants (I): Constants as Arguments

put_constant c , $A_i \equiv A_i \leftarrow \langle \text{CON}, c \rangle;$

get_constant c , $A_i \equiv \text{addr} \leftarrow \text{deref}(A_i);$

case STORE[addr] **of**

$\langle \text{REF}, _ \rangle$: STORE[addr] $\leftarrow \langle \text{CON}, c \rangle;$

$\text{trail}(addr);$

$\langle \text{CON}, c' \rangle$: $\text{fail} \leftarrow (c \neq c');$

other : $\text{fail} \leftarrow \text{true};$

endcase;

$\text{trail}(addr)$ \equiv no-op (for the moment)

Instructions for Constants (II): Constants Inside a Term

set_constant $c \equiv \text{HEAP}[H] \leftarrow \langle \text{CON}, c \rangle;$
 $H \leftarrow H+1;$

unify_constant $c \equiv \text{case } mode \text{ of}$
 read : $addr \leftarrow \text{deref}(S);$
 case STORE[$addr$] **of**
 $\langle \text{REF}, _ \rangle$: $\text{STORE}[addr] \leftarrow \langle \text{CON}, c \rangle;$
 $\text{trail}(addr);$
 $\langle \text{CON}, c' \rangle$: $fail \leftarrow (c \neq c');$
 other : $fail \leftarrow \text{true};$
 endcase;
 write : $\text{HEAP}[H] \leftarrow \langle \text{CON}, c \rangle;$
 $H \leftarrow H+1;$
 endcase;
 $S \leftarrow S+1$

Heap Representation with Lists

1	REF	1
2	CON	[]
3	REF	3
4	REF	3
5	LIS	1
6	STR	7
7	f/1	
8	REF	1

Heap representation of
 $p(Z,[Z,W],f(W))$

A1: $\langle \text{REF } 3 \rangle$

A2: $\langle \text{LIS } 4 \rangle$

A3: $\langle \text{STR } 7 \rangle$

Instructions for Lists

put_list X_i	$\equiv X_i \leftarrow \langle \text{LIS}, H \rangle;$
get_list X_i	$\equiv addr \leftarrow \text{deref}(X_i);$ case STORE[addr] of $\langle \text{REF}, _ \rangle$: HEAP[H] $\leftarrow \langle \text{LIS}, H+1 \rangle;$ bind(addr, H); H $\leftarrow H+1;$ mode \leftarrow write; $\langle \text{LIS}, a \rangle$: S $\leftarrow a;$ mode \leftarrow read; other : fail \leftarrow true; endcase;

Example: Query $p(Z,[Z,W],f(W))$

```
put_list X5
set_variable X6
set_constant []
put_variable X4,A1
put_list A2
set_value X4
set_value X5
put_structure f/1,A3
set_value X6
call p/3
```

Example: Fact $p(f(X), [Y, f(a)], Y)$

get_structure $f/1, A_1$

unify_variable X_4

get_list A_2

unify_variable X_5

unify_variable X_6

get_value X_5, A_3

get_list X_6

unify_variable X_7

unify_constant []

get_structure $f/1, X_7$

unify_constant a

proceed

Objectives

Introducing the Warren Abstract Machine (WAM)

Language L_0 : handling terms

Compiling L_0 -queries and L_0 -programs

Unification

From L_0 to L_1 : handling atoms

Special data structures: constants and lists