Integrated Logic Systems
Part 2: Prolog (II)

- Compiling L_0-programs
- Unification
- From L_0 to L_1: handling atoms
- Special data structures: constants and lists

Compiling a Program

The flattened equations need to be ordered

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

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

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

1. `get structure f/n, X`$_i$
2. `unify variable X`$_{i_1}$ or `unify value X`$_{i_1}$

⋮

n+1. `unify variable X`$_{i_n}$ or `unify value X`$_{i_n}$

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

`unify_value` X_i 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
```

WAM_0 Machine Instructions (II)

`get_structure f/n,X`$_i$ ≡

```
addr ← deref (X_i);
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 WAM_0

```
procedure bind(a1,a2: address);
    begin
        case ← STORE[a1] of
            ⟨REF,_⟩   : STORE[a1] ← STORE[a2];
            other     : STORE[a2] ← STORE[a1];
        endcase;
    end bind
```

WAM_0 Machine Instructions (III)

`unify_variable` X_i ≡ **case** *mode* **of**

read: $X_i \leftarrow HEAP[S]$;

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

$X_i \leftarrow HEAP[H]$;

$H \leftarrow H+1$;

endcase;

$S \leftarrow S+1$

`unify_value` X_i ≡ **case** *mode* **of**

read: *unify* (X_i ,S);

write: $HEAP[H] \leftarrow X_i$;

$H \leftarrow H+1$;

endcase;

$S \leftarrow S+1$

Unification (Part I)

```
procedure unify(a1,a2: address);
        begin
                push(a1,PDL); push(a2,PDL);
                fail ← false;
                while ¬(empty(PDL) ∨ fail) do
                        begin
                                d1 ← deref(pop(PDL)); d2 ← deref(pop(PDL));
                                if d1 ≠ d2 then
                                        begin
                                                ⟨t1,v1⟩ ← STORE[d1];
                                                ⟨t2,v2⟩ ← STORE[d2];
                                                if (t1=REF) ∨ (t2=REF)
                                                        then bind(d1,d2)
```

Unification (Part II)

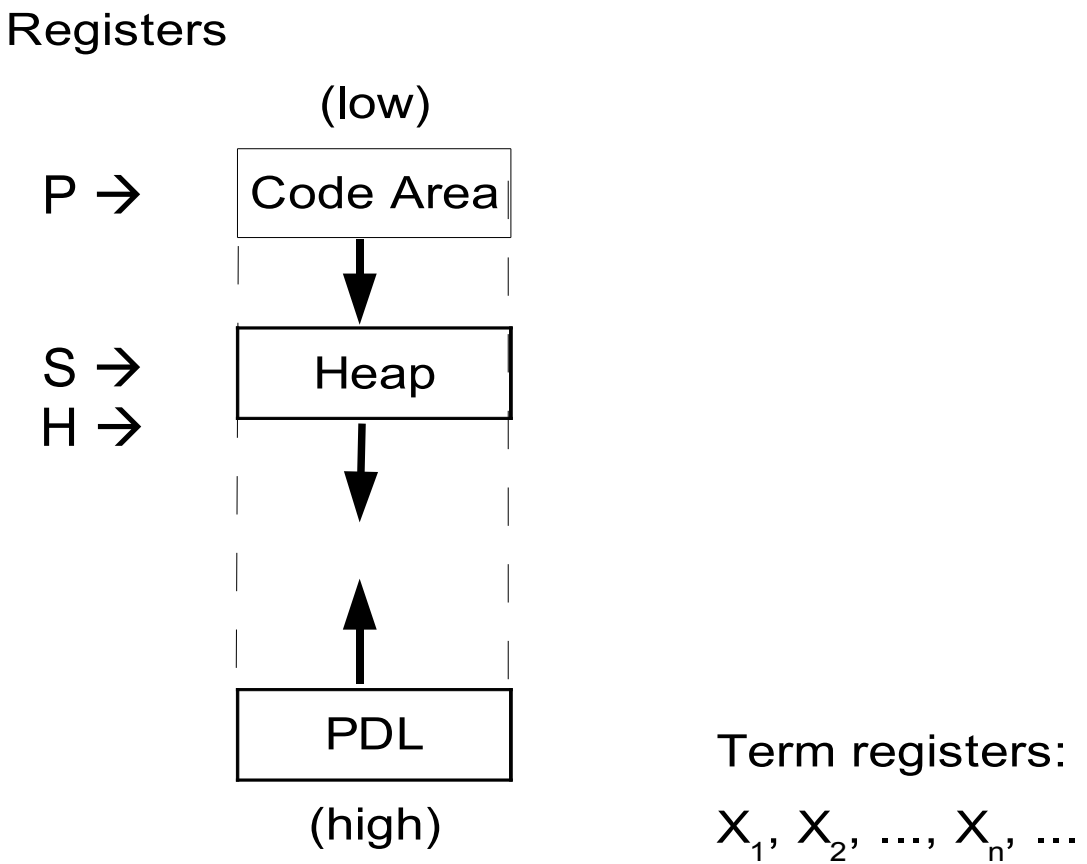
```
                else
                    begin
                        f1 / n1 ← STORE[v1]; f2 / n2 ← STORE[v2];
                        if (f1=f2) ∧ (n1=n2)
                            then
                                for i ← 1 to n1 do
                                    begin
                                        push(v1 + i, PDL);
                                        push(v2 + i, PDL)
                                    end
                            else fail ← 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.

WAM_1 Memory Layout



Term registers:

$X_1, X_2, ..., X_n, ...$

Control Instructions

`call p/n` ≡ P ← @(p/n);

where @(*p*/*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` ≡ no-op; (for the moment)

Argument Registers

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

To emphazise 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 occured 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

WAM_1 Machine Instructions

`put_variable` X_n, A_i ≡ HEAP[H] ← ⟨REF,H⟩;

X_n ← HEAP[H];

A_i ← HEAP[H];

H ← H+1;

`put_value` X_n, A_i ≡ A_i ← X_n;

`get_variable` X_n, A_i ≡ X_n ← A_i;

`get_value` X_n, A_i ≡ *unify*(X_n, A_i);

Special Datastructures: Heap Representation with Constants

7	STR	8
8	g/1	
9	CON	a
10	STR	11
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 ≡ $A_i \leftarrow \langle \text{CON}, c \rangle$;

`get_constant` c, A_i ≡ $addr \leftarrow deref(A_i)$;

case STORE[*addr*] **of**

⟨REF,_⟩ : STORE[*addr*] ← ⟨CON,*c*⟩;

⟨CON,*c*'⟩ : *fail* ← (*c* ≠ *c*');

other : *fail* ← **true**;

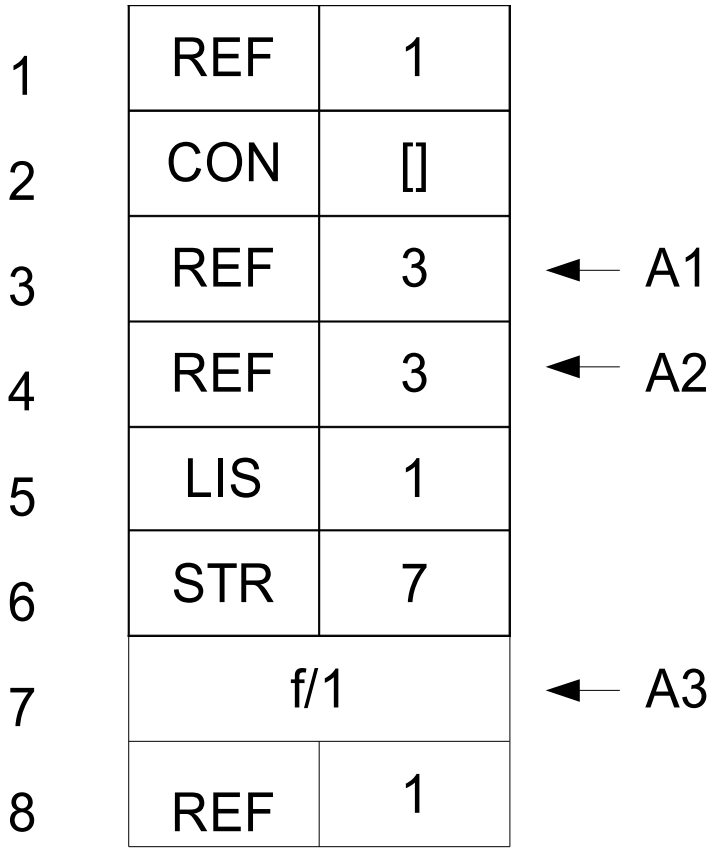
endcase;

Instructions for Constants (II): Constants Inside a Term

```
set_constant c     ≡  HEAP[H] ← ⟨CON,c⟩;
                      H ← H+1;

unify_constant c   ≡  case mode of
                         read  :  addr ← deref(S);
                                  case STORE[addr] of
                                    ⟨REF,_⟩   :  STORE[addr] ← ⟨CON,c⟩;
                                    ⟨CON,c'⟩  :  fail ← (c ≠ c');
                                    other     :  fail ← true;
                                  endcase;
                         write :  HEAP[H] ← ⟨CON,c⟩;
                                  H ← H+1;
                      endcase;
```

Heap Representation with Lists



Heap representation of

p(Z,[Z,W],f(W))

Instructions for Lists

`put_list` X_i ≡ $X_i \leftarrow \langle$LIS,H$\rangle$;

`get_list` X_i ≡ *addr* ← *deref*(X_i);
case STORE[*addr*] **of**
⟨REF,_⟩ : HEAP[H] ← ⟨LIS,H+1⟩;
bind(*addr*,H);
H ← H+1;
mode ← write;
⟨LIS,a⟩ : S ← a;
mode ← read;
other : *fail* ← **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,A1
unify_variable X4
get_list A2
unify_variable X5
unify_variable X6
get_value X5,A3
get_list X6
unify_variable X7
unify_structure f/1,X7
unify_constant a
proceed
```

Objectives

- Compiling L_0-programs
- Unification
- From L_0 to L_1: handling atoms
- Special data structures: constants and lists