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

Schedule

Lectures		Tutorials
April, 8th, 2009	Prolog (I)	April, 17th, 2009
April, 24th, 2009	Prolog (II)	
May, 6th, 2009	Prolog (III)	May, 15th, 2009
May, 20th, 2009	Tableaux-Prover	May, 29th, 2009
June, 10th, 2009	Answer Set Programming (I)	June, 19th, 2009
June, 24th, 2009	Answer Set Programming (II)	July, 3rd, 2009
July, 8th, 2009	Wrap Up	July, 10th, 2009

Today's Lecture

- Introducing the Warren Abstract Machine (WAM)
- Language L_0
- Encoding terms
- Compiling L_0-queries

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.

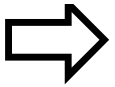
Knowing the WAM enables you to

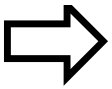
- understand specific behaviors of a Prolog system
- design and implement basic functionalities of deduction systems
  - term unification
  - derivations (= sequences of deduction steps)
  - search (via backtracking)

The Basic Picture

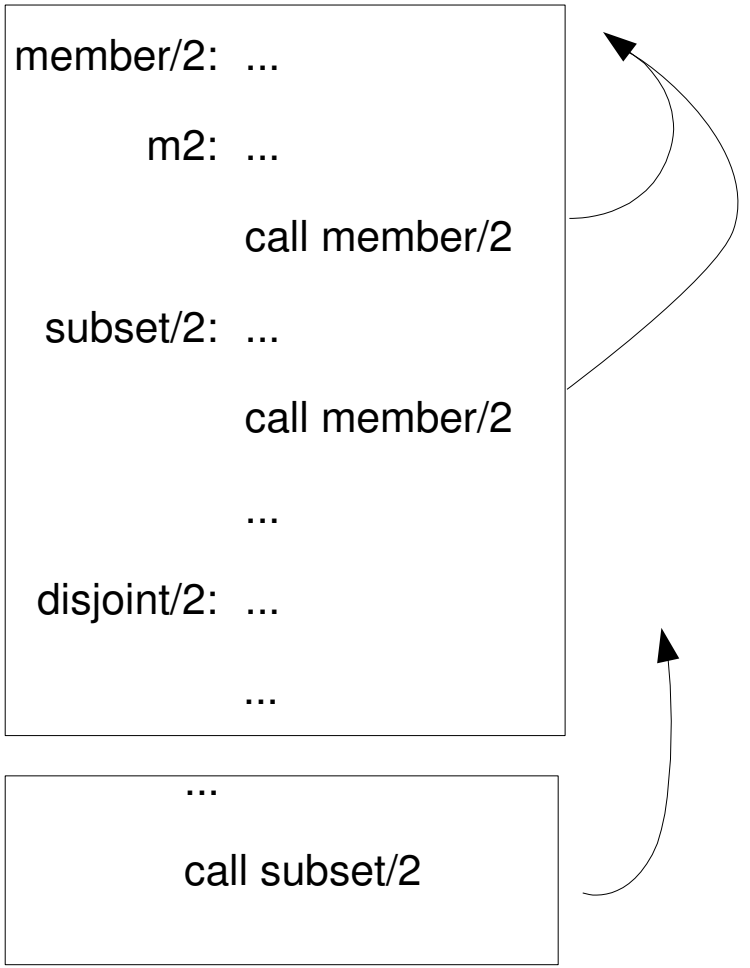
```
member(X,[X|_]).
member(X,[_|Y]):- member(X,Y).
subset([X|Y],Z):- member(X,Z),
                  ...
disjoint(X,Y):- ...
...
```

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





WAM commands



The Language L_0

Consider a term alphabet.

A program is a term.

A query is a term.

Intended meaning:

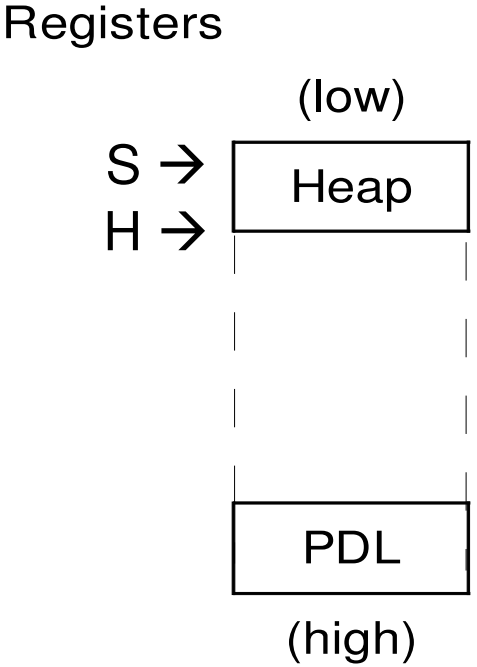
- "query" *t* fails wrt "program" *s* if *s* and *t* are not unifiable
- otherwise *t* succeeds with a binding of the variables in *t* obtained by unifying it with *s*

Example:

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

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

WAM_0 Memory Layout



S – special register,

pointing to “sub-term”

H – pointer to top of heap

Push-down-list

Term registers:
X_1, X_2, ...

Heap Representation of Terms

Unbound variable:

i	REF	i

Bound variable:

i	REF	j	$j \neq i$

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

i	STR | i+1
i+1	f/n
i+2	$arg_1(t)$
.	.
.	.
.	.
.	.
.	.
.	.
i+n+1	$arg_n(t)$

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, ...

- Register X_1 is allocated to the outermost term.
- The same register is allocated to all occurences of a given variable.

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

Processing Flattened Equations

The 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 $\vec{X}_1, ..., \vec{X}_{i-1}$

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

1. `put_structure f/n, X`$_i$
2. `set_variable X`$_{i_1}$ or `set_value X`$_{i_1}$

⋮

n+1. `set_variable X`$_{i_n}$ or `set_value X`$_{i_n}$

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

`set_value` X_i used if X_i has been processed before

WAM_0 Machine Instructions (I)

`put structure f/n,`X_i $\equiv$ HEAP[H] $\leftarrow$ ⟨STR,H+1⟩;

HEAP[H+1] $\leftarrow$ `f/n`;

X_i $\leftarrow$HEAP[H];

H $\leftarrow$ H+2;

`set variable` X_i $\equiv$ HEAP[H] $\leftarrow$ ⟨REF,H⟩;

X_i $\leftarrow$HEAP[H];

H $\leftarrow$ H+1;

`set value` X_i $\equiv$ HEAP[H] $\leftarrow$ X_i;

H $\leftarrow$ H+1;

Objectives

- Introducing the Warren Abstract Machine (WAM)
- Language L_0
- Encoding terms
- Compiling L_0-queries