Prolog Implementation

- The Generic Idea: Everything except unification.

- An Introduction to the WAM: Flat unification only.
What needs to be kept track of

- The current substitution
- The current goal
- The clause currently in use
- Backtrack Information
A Stack based “Implementation”
Adapted from the ISO standard

This execution model is based on a stack. *ES.*

St-index – A value defined by the current number of components of *S.*

sglstk – A stack of subgoals which defines a sequence of activators that might be activated during execution.

subst – A substitution which defines the state of the instantiations of the variables.

BI – Backtrack information: a value which defines how to re-execute a goal.

*currstate* is the top of the stack *ES.*
The current subgoal is the given by the top of sglstk. It contains:

1. The current activator, \textit{curract},

2. Its \textit{cutparent}.
$BI$ has value:

nil – Its initial value, or

ctrl – The procedure is a control construct, or

bip – The activated procedure is a built-in predicate, or

up($CL$) – $CL$ is a list of the clauses of a user-defined procedure whose predicate is identical to $curract$, and which are still to be executed, and $DM$ is the module in whose body these clauses appear.
The *choicepoint* for the execution state $ES_{i+1}$ is $ES_i$.

<table>
<thead>
<tr>
<th>St-index</th>
<th>Subgoal Stack</th>
<th>Substitution</th>
<th>BI</th>
</tr>
</thead>
</table>
A subgoal $DS$ on the subgoal stack is a structured data type with components:

activator – A predication $P$ which must be executed successfully in order to satisfy the goal.

cutparent – A pointer to a deeper execution state that indicates where control is resumed should a cut be re-executed.
**Initialization**

<table>
<thead>
<tr>
<th>St-index</th>
<th>Decorated Subgoal Stack,</th>
<th>Substitution</th>
<th>BI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((goal, 0), \text{newstack}_{DS}), {}</td>
<td>\text{nil}</td>
<td></td>
</tr>
</tbody>
</table>

\(\text{newstack}_{ES}\)
Success and Failure

• A goal succeeds when the subgoal stack of *currstate* is empty

• A goal fails when the execution stack is empty
Backtracking

1. Examine the value of $BI$ for the new $currstate$.

2. If $BI$ is $up(\text{CL})$ then $p$ is a user defined procedure remove the head of $\text{CL}$ and continue executing a user defined procedure.

3. If $BI$ is $bip$ then $p$ is a built-in predicate, continue by executing the builtin.

4. If $BI$ is $ctrl$ the effect of re-executing depends on the control construct

5. If $BI$ is $nil$ then the new $curract$ has not been executed, continue execution by selecting a clause.
Executing a user defined procedure

1. If there are no more clauses for $p$ then pop $currstate$ from $ES$.

2. Else select the clause $c$ from $up(c|CL)$.

3. Unify the head of $c$ and $curract$ producing a most general unifier $MGU$.

4. Apply the substitution $MGU$ to the body of $c$.

5. Make a copy $C$ of $currstate$.

6. Apply the substitution $MGU$ to the current goal in $C$. 

11
7. Replace the current activator of the current goal in \( C \) by the \( MGU \) modified body of \( c \).

8. Create a new entry for \( ES \) whose elements are \( C, BI = nil \) and with substitution the composition of the substitution in \( currstate \) and \( MGU \).

9. Set \( cutparent \) of the new first subgoal of the subgoal stack of this new entry to be the current choice point.

10. Push this new entry on to \( S \). It becomes the new \( currstate \) and the previous \( currstate \) becomes its \( choicepoint \).

11. Continue execution by selecting a clause.
Selecting a clause

1. Search the database for a procedure \( p \) corresponding to the identifier and arity of \textit{curract}.

2. If \( p \) is a control construct (true, fail, call, cut, conjunction, disjunction, if-then, if-then-else, catch, throw) then \( BI \) is set to \textit{ctrl} and execution continues according to the rules for control constructs.

3. If \( p \) is a built-in predicate \( BP \) then \( BI \) is set to \textit{bip} and continue execution for a bip.

4. If \( p \) is a user-defined procedure \( BI \) is set to \( up(CL) \), where \( CL \) is a list of the current clauses of \( p \) of the procedure; continue by executing the user-defined procedure.
The Warren Abstract Machine

Most implementations of Prolog now use some version of the WAM.

This introduction follows that of Hassan Aït-Kaci in his tutorial reconstruction of the WAM. We cover a simple version of unification.
Unification in $L_0$: Term Representation

Terms are represented on a \textit{HEAP} which is an array. The term's reference is it's index on the \textit{HEAP}. To represent

$$f(X, Y, a)$$

we need to tag the \textit{HEAP} items to distinguish structures such as

$$f(@_1, \ldots, @_n)$$

from the references @$i$. Cells that are references to variables will be of the form

$$REF, k$$

where $k$ is the cell referred to.

Cells that reference structures will be of the form

$$STR, k$$

Structures themselves are denoted by \textit{functor/arity}
Heap representation of $p(Z, h(Z, W, f(W)))$.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STR</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>h/2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>REF</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>REF</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>STR</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>f/1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>REF</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>STR</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>p/3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>REF</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>STR</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>STR</td>
<td>5</td>
</tr>
</tbody>
</table>

Note that (the first occurrence of) a variable is a self reference. So 2 is $Z$ and 3 is $W$. Note also that the arity tells you how many references to take.
Unification in $\mathcal{L}_0$ : Compiling Queries

To compile a query $q$ we have to translate it into a representation of its heap format. Thus while processing part of a term we need to be able to save temporarily pieces of terms yet to be processed, or a variable that may appear later in a term. For this (variable) registers $X1$, $X2$ etc. are used. are used.

For our example we would allocate as follows.

$X1 = p(X2, X3, X4)$
$X2 = Z$
$X3 = h(X2, X5)$
$X4 = f(X5)$
$X5 = W$
Unification in $\mathcal{L}_0$: Compiling Queries 2

There are three kinds of tokens that must be processed:

1. A register associated with a structure functor;

2. A register argument not previously encountered;

3. A register argument previously seen.
Leading to three instructions for the machine $M_0$

1. \texttt{put\_structure} $f/n, Xi$

2. \texttt{set\_variable} $Xi$

3. \texttt{set\_value} $Xi$

The heap is therefore used as a stack for building terms.
Unification $\mathcal{L}_0$: Compiling Queries 3

put_structure h/2,X3  % X3 = h
set_variable X2       % (Z,
set_variable X5       %  W)
put_structure f/1,X4   % X4 = f
set_value X5          % (W),
put_structure p/3, X1  % X1 = p
set_value X2          % (Z,
set_value X3          %  X3,
set_value X4          %  X4).

Note that the address of the query is $X_1$. 
Unification in $\mathcal{L}_0$ : Compiling Programs

Unifying a query $q$ to $p$ (a program term) can proceed by following the term structure in $X1$ as long as it matches the structure of $p$. There is a complication with unbound $REF$ terms.

So if we have a program term.
$p(f(X), h(Y, f(a)), Y)$ the matching we should get is:

\begin{align*}
X1 & = p(X2, X3, X4) \\
X2 & = f(X5) \\
X3 & = h(X4, X6) \\
X4 & = Y \\
X5 & = X \\
X6 & = f(X7) \\
X7 & = a
\end{align*}
Flattening the program.

1. A register associated with a structure functor;

2. a first seen register argument;

3. an already seen register argument.

Instructions:

1. `get_structure f/n, X_i`

2. `unify_variable X_i`

3. `unify_value X_i`