Chapter Overview

- From $L_1$ to $L_2$: SLD-derivations
- From $L_2$ to Prolog: Backtracking
- The cut in Prolog
A program is a set of clauses - no two of which define the same predicate.

A query is a finite sequence of atoms.
WAM₂ Memory Layout

Registers

<table>
<thead>
<tr>
<th>P</th>
<th>CP</th>
<th>S</th>
<th>H</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>P ↦</td>
<td>CP ↦</td>
<td>S ↦</td>
<td>H ↦</td>
<td>E ↦</td>
</tr>
</tbody>
</table>

(low)

Argument and
term registers:
X₁, X₂, ...

Code Area

Heap

Stack

environment

PDL

(high)
The Basic Picture

```
first(X, [X|_]).
second(X, [_,X|_]).
sequence(Y,Z) :- first(X,Z), second(X,Y).
```

```
?- sequence([a,b],[b,c]), sequence([b,c],[c,d]).
```

```
first/2: program_code_for_first
         proceed
second/2: program_code_for_second
         proceed
sequence/2: allocate
            program_code_for_sequence
            query_code_for_first
            call first/2
            query_code_for_second
            call second/2
            deallocate
```

```
allocate
query_code_for_sequence
call sequence/2
...
```
Handling Facts

\[
\text{call } p/n \equiv \ CP \leftarrow P + \text{instruction}_\text{size}(P);
\]
\[
P \leftarrow @(p/n);
\]

\[
\text{proceed} \equiv \ P \leftarrow CP;
\]

\text{instruction}_\text{size}(P): \text{ number of memory units occupied by the instruction at address } P
Handling Clauses

A *permanent variable* in a clause $H : \neg B_1, \ldots, B_m$:

:ô

A occurs in more than one of these $m$ sets:

$\text{Var}(H) \cup \text{Var}(B_1), \text{Var}(B_2), \ldots, \text{Var}(B_m)$

To emphasize that a variable is permanent, we write $Y_i$.

Permanent variables are cells in the stack area.
Environmental Frame

The environment frame is used to save local data upon calling a procedure.

- \( E \) (continuation environment)
- \( E + 1 \) (continuation point)
- \( E + 2 \) (number of permanent variables)
- \( E + 3 \) (permanent variable 1)
- \( \vdots \)
- \( E + n + 2 \) (permanent variable \( n \))
WAM$_2$ Machine Instructions

\[ \text{allocate } N \equiv \] 
\[ \text{newE } \leftarrow \ E + \text{STACK}[E+2] + 3; \] 
\[ \text{STACK}[\text{newE}] \leftarrow \ E; \] 
\[ \text{STACK}[\text{newE}+1] \leftarrow \text{CP}; \] 
\[ \text{STACK}[\text{newE}+2] \leftarrow \text{N}; \] 
\[ \text{E } \leftarrow \text{newE}; \] 
\[ \text{P } \leftarrow \text{P } + \text{instruction\_size}(\text{P}); \]

\[ \text{deallocation } \equiv \] 
\[ \text{P } \leftarrow \text{STACK}[E+1]; \] 
\[ \text{E } \leftarrow \text{STACK}[E]; \]
Example

\[
p/2: \text{ allocate } 2 \quad \% \quad p
\]
\[
\text{get_variable } X_3,A_1 \quad \% \quad (X,
\]
\[
\text{get_variable } Y_1,A_2 \quad \% \quad Y) :-
\]
\[
\text{put_value } X_3,A_1 \quad \% \quad q(X,
\]
\[
\text{put_variable } Y_2,A_2 \quad \% \quad Z
\]
\[
\text{call } q/2 \quad \% \quad ),
\]
\[
\text{put_value } Y_2,A_1 \quad \% \quad r(Z,
\]
\[
\text{put_value } Y_1,A_2 \quad \% \quad Y
\]
\[
\text{call } r/2 \quad \% \quad )
\]
\[
\text{deallocate} \quad \% \quad .
\]

\[\text{WAM}_2 \text{ Machine Code for } p(X, Y) :- q(X, Z), r(Z, Y).\]
Handling the Query

The query is encoded like a clause but without the instructions for the missing head.

Prior to calling the code for the query, an initial environment is created:

\[
\begin{align*}
\text{STACK}[E] & \leftarrow E; \\
\text{STACK}[E+1] & \leftarrow \text{CP}; \\
\text{STACK}[E+2] & \leftarrow 0;
\end{align*}
\]

Following the (successful) call to the query, the register bindings are printed as the CAS.
From $L_2$ to Pure Prolog

A program is an arbitrary set of clauses.
Choice Points vs. Environment Frames

Example:

\[
\begin{align*}
  a & : - b(X), c(X). \\
  b(X) & : - e(X). \\
  c(1) & . \\
  e(X) & : - f(X). \\
  e(X) & : - g(X). \\
  f(2) & . \\
  g(1) & . \\
  ?- a.
\end{align*}
\]

This example shows that a choice point must protect each previously stored environment frame from deallocation in order that the environment is still available upon backtracking.
WAM Memory Layout

Registers

- P → Code Area
- CP → Heap
- S, HB → Stack
- H → choice point environment
- B → nth argument
- E → cont. environment
- TR → CP cont. point
- TR → Trail
- TR → previous choice pt.
- TR → trail pointer
- TR → heap pointer

Argument and term registers:

- X₁, X₂, ...

n arity
A₁ 1st argument
:  
Aₙ nth argument
CE cont. environment
CP cont. point
B previous choice pt.
BP next clause
TR trail pointer
H heap pointer
Backtracking requires to “undo” substitutions made after the choicepoint to which we jump back.

The trail contains the addresses of all variables which need to be unbound upon backtracking.

Register HB contains the value of H at the time of the latest choice point. Only variables below HB need to be reset.
Recording the Bindings

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

- trail (addr: address)
  
  pushes \(addr\) onto the trail, provided that
  \[addr < \text{HB}\]

- unwind_trail (addr\_1, addr\_2: address)
  
  resets all variables stored in \(addr\_1, addr\_1 + 1, \ldots, addr\_2 - 1\)
The Trail and Unwind Operations

procedure trail(a: address);
begin
  if (a < HB)
    then
      begin
        TRAIL[TR] ← a;
        TR ← TR+1
      end
  end trail

procedure unwind_trail(a₁,a₂: address);
begin
  for i ← a₁ to a₂ – 1 do
    STORE[TRAIL[i]] ← ⟨REF,TRAIL[i]⟩
end unwind_trail
Choice Point Frame

The choice point frame is used to save local data needed upon backtracking.

<table>
<thead>
<tr>
<th>B</th>
<th>n     (number of arguments)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B+1</td>
<td>A1   (argument register 1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>B+n</td>
<td>An   (argument register n)</td>
</tr>
<tr>
<td>B+n+1</td>
<td>CE   (continuation environment)</td>
</tr>
<tr>
<td>B+n+2</td>
<td>CP   (continuation point)</td>
</tr>
<tr>
<td>B+n+3</td>
<td>B    (previous choice point)</td>
</tr>
<tr>
<td>B+n+4</td>
<td>BP   (next clause)</td>
</tr>
<tr>
<td>B+n+5</td>
<td>TR   (trail pointer)</td>
</tr>
<tr>
<td>B+n+6</td>
<td>H    (heap pointer)</td>
</tr>
</tbody>
</table>
allocate N ≡ if E > B

    then newE ← E+STACK[E+2] + 3

    else newE ← B+STACK[B] + 7;

STACK[newE] ← E;
STACK[newE+1] ← CP;
STACK[newE+2] ← N;
E ← newE;
P ← P + instruction_size(P);
Coding Scheme for Backtracking

\[
\begin{align*}
 p(t_{11}, \ldots, t_{1n}) & : B_1 \\
p(t_{21}, \ldots, t_{2n}) & : B_2 \\
p(t_{31}, \ldots, t_{3n}) & : B_3 \\
 & \quad \vdots \\
p(t_{k1}, \ldots, t_{kn}) & : B_k
\end{align*}
\]

\[
\begin{align*}
p / n & : \text{try\_me\_else } L_1 \\
& \quad \text{code for first clause} \\
L_1 & : \text{retry\_me\_else } L_2 \\
& \quad \text{code for second clause} \\
& \quad \cdots \\
L_{k-2} & : \text{retry\_me\_else } L_{k-1} \\
& \quad \text{code for third clause} \\
L_{k-1} & : \text{trust\_me} \\
& \quad \text{code for k-th clause}
\end{align*}
\]
try_me_else L ≡ if E > B
  then newB ← E+STACK[E+2] + 3
  else newB ← B+STACK[B] + 7;
STACK[newB] ← num_of_args;
n ← STACK[newB];
for i ← 1 to n do STACK[newB+i] ← A_i;
STACK[newB+n+1] ← E;
STACK[newB+n+2] ← CP;
STACK[newB+n+3] ← B;
STACK[newB+n+4] ← L;
STACK[newB+n+5] ← TR;
STACK[newB+n+6] ← H;
B ← newB;
HB ← H;
P ← P + instruction_size(P);
WAM Machine Instructions (III)

\[
\text{retry\_me\_else } L \equiv \quad n \leftarrow \text{STACK}[B]; \\
\text{for } i \leftarrow 1 \text{ to } n \text{ do } A_i \leftarrow \text{STACK}[B+i]; \\
E \leftarrow \text{STACK}[B+n+1]; \\
\text{CP} \leftarrow \text{STACK}[B+n+2]; \\
\text{STACK}[B+n+4] \leftarrow L; \\
unwind\_trail(\text{STACK}[B+n+5], \text{TR}); \\
\text{TR} \leftarrow \text{STACK}[B+n+5]; \\
H \leftarrow \text{STACK}[B+n+6]; \\
\text{HB} \leftarrow H; \\
P \leftarrow P + \text{instruction\_size}(P);
\]
trust_me ≡
\[
\begin{align*}
n &\leftarrow \text{STACK}[B]; \\
\text{for } i &\leftarrow 1 \text{ to } n \text{ do } A_i \leftarrow \text{STACK}[B+i]; \\
E &\leftarrow \text{STACK}[B+n+1]; \\
\text{CP} &\leftarrow \text{STACK}[B+n+2]; \\
\text{unwind\_trail}(\text{STACK}[B+n+5], \text{TR}); \\
\text{TR} &\leftarrow \text{STACK}[B+n+5]; \\
H &\leftarrow \text{STACK}[B+n+6]; \\
B &\leftarrow \text{STACK}[B+n+3]; \\
\text{HB} &\leftarrow \text{STACK}[B+n+6]; \\
P &\leftarrow P + \text{instruction\_size}(P);
\end{align*}
\]
WAM Machine Instructions (V)

\[
\text{backtrack} \equiv P \leftarrow \text{STACK}[B + \text{STACK}[B]+4];
\]
(whenever failure occurs)

\[
\text{call p/n} \equiv \text{CP} \leftarrow P + \text{instruction_size}(P);
\]
\[
\text{num_of_arg} \leftarrow n;
\]
\[
P \leftarrow @ (p/n);
\]
The Cut

The effect of the cut “!” is to forget any other alternative for the clause in which it appears and any other alternative arising from preceding body atoms in this clause.

This means to discard all choice points (CPs) created after the CP that was the current CP at the time the clause with the cut was called.

An additional cut register $B_0$ is used to store the appropriate CP where to return upon backtracking over a cut.
Call Revisited

call p/n  ≡  CP ← P + \textit{instruction\_size}(P); \
num\_of\_arg ← n; 
B0 ← B; 
P ← @ (p/n);

The value of B0 is saved in every choice point frame.
The Cut as First Body Atom

\[ a/0: \text{allocate} \, 0 \quad \% \quad a \]
\[ \text{neck\_cut} \quad \% \quad :{-}!, \]
\[ \text{call} \, b/0 \quad \% \quad b \]
\[ \text{deallocate} \quad \% \quad . \]

WAM Machine Code for \( a :{-}!, b \).
Machine Instruction

\[
\text{neck_cut} \equiv \text{ if } B > B0 \text{ then begin }
\]
\[
\quad B \leftarrow B0; \\
\quad H_B \leftarrow \text{STACK}[B+\text{STACK}[B]+6]; \\
\quad \text{tidy_trail}
\]
\[
\quad \text{end}; \\
\quad P \leftarrow P + \text{instruction_size}(P);
\]

The value of $B0$ is saved in every choice point frame.
Auxiliary Function

procedure tidy_trail;
  begin
    i ← STACK[B+STACK[B]+5];
    while i < TR do
      if TRAIL[i] < HB then
        i ← i+1
      else begin
        TRAIL[i] ← TRAIL[TR-1];
        TR ← TR-1;
      end
  end tidy_trail;
A Deep Cut

\[ a/0: \text{allocate 1} \quad \% \quad a \]
\[ \text{get_level } Y_1 \]
\[ \text{call } b/0 \quad \% \quad :- b, \]
\[ \text{cut } Y_1 \quad \% \quad !, \]
\[ \text{call } c/0 \quad \% \quad c \]
\[ \text{deallocate} \quad \% \quad . \]

WAM Machine Code for \( a : - b, !, c. \)
Machine Instructions

\[
\text{get\_level } Y_n \equiv \text{STACK}[E+2+n] \leftarrow B0;
\]
\[
P \leftarrow P + \text{instruction\_size}(P);
\]

\[
\text{cut } Y_n \equiv \text{if } B > \text{STACK}[E+2+n] \text{ then begin}
\]
\[
B \leftarrow \text{STACK}[E+2+n];
\]
\[
HB \leftarrow \text{STACK}[B+\text{STACK}[B]+6];
\]
\[
tidy\_trail
\]
\[
\text{end;}
\]
\[
P \leftarrow P + \text{instruction\_size}(P);
\]
Objectives

- From $L_1$ to $L_2$: SLD-derivations
- From $L_2$ to Prolog: Backtracking
- The cut in Prolog