34 Last Call Optimization

Consider the app predicate from the beginning:

\[ \text{app}(X, Y, Z) \leftarrow X = [], Y = Z \]
\[ \text{app}(X, Y, Z) \leftarrow X = [H|X'], Z = [H|Z'], \text{app}(X', Y, Z') \]

We observe:
- The recursive call occurs in the last goal of the clause.
- Such a goal is called last call.

\[ \rightarrow \text{ we try to evaluate it in the current stack frame } \]
\[ \rightarrow \text{ after (successful) completion, we will not return to } \]
\[ \rightarrow \text{ the current caller } \]

---

34 Last Call Optimization

Consider the app predicate from the beginning:

\[ \text{app}(X, Y, Z) \leftarrow X = [], Y = Z \]
\[ \text{app}(X, Y, Z) \leftarrow X = [H|X'], Z = [H|Z'], \text{app}(X', Y, Z') \]

We observe:
- The recursive call occurs in the last goal of the clause.
- Such a goal is called last call.

\[ \rightarrow \text{ we try to evaluate it in the current stack frame } \]
\[ \rightarrow \text{ after (successful) completion, we will not return to } \]
\[ \rightarrow \text{ the current caller } \]
Consider a clause: 
\[ p(X_1, \ldots, X_n) \leftarrow g_1, \ldots, g_k \]
with \( m \) locals where 
\[ g_k \equiv q(t_1, \ldots, t_k) \].

The interplay between `codeC` and `codeQ`: 
\[ \text{codeC} \rightarrow \text{pushenv m} \]
\[ \text{codeC} \rightarrow g_1 \rho \]
\[ \ldots \]
\[ \text{codeC} \rightarrow g_{k-1} \rho \]

Mark B
\[ \text{codeA} \rightarrow t_1 \rho \]
\[ \ldots \]
\[ \text{codeA} \rightarrow t_k \rho \]

Call q/h

Replacement: 
\[ \text{mark B} \quad \text{lastmark} \]
\[ \text{call q/h; popenv} \quad \text{lastcall q/h m} \]

If the current clause is not last or the \( g_1, \ldots, g_{k-1} \) have created backtrack points, then \( \text{FP} \leq \text{BP} \) (-)

Then lastmark creates a new frame but stores a reference to the predecessor:

If \( \text{FP} \leq \text{BP} \), then lastcall q/h m behaves like a normal call q/h.

Otherwise, the current stack frame is re-used. This means that:
- the cells \( S[\text{FP}+1], S[\text{FP}+2], \ldots, S[\text{FP}+h] \) receive the new values and
- q/h can be jumped to (-)

If \( \text{FP} > \text{BP} \) then lastmark does nothing (-)

The difference between the old and the new addresses of the parameters m just equals the number of the local variables of the current clause (-)
Example:

Consider the clause:

\[ a(X, Y) \leftarrow f(X, X_1), a(X_1, \bar{Y}) \]

The last-call optimization for \( \text{code}_C \) yields:

\[
\begin{array}{ll}
\text{pushenv } 3 & \text{mark } A \\
p \text{putref } 1 & \text{A: } \text{lastmark} \\
p \text{putvar } 3 & \text{putref } 3 \\
c \text{call } f/2 & \text{putref } 2 \\
\end{array}
\]

Note:

If the clause is \text{last} and the last literal is the only one, we can skip \text{lastmark} and can replace lastcall \( q/h \) \( m \) with the sequence \text{move } m \ n; \text{jump } p/n \Rightarrow \).
Consider a clause \( r(X_1, \ldots, X_n) \leftarrow q(t_1, \ldots, t_k) \). The interplay between code\( g \) and

code\( r \) = pushenv \( m \)
code\( g \) = \( g \)
\[ \ldots \]
code\( g_n \) = \( g \)
mark \( B \)
code\( p \) = \( t_1 \)
\[ \ldots \]
code\( p \) = \( t_k \)
call \( q/h \)

B : popenv

Replacement:
mark \( B \) \( \Longrightarrow \) lastmark
call \( q/h \); popenv \( \Longrightarrow \) lastcall \( q/h \) \( m \)

35 Trimming of Stack Frames

Idea:
- Order local variables according to their life times;
- Pop the dead variables — if possible  \(-:)\)

Example:
Consider the clause:
\[ z(X, Z) \leftarrow p_1(X, X_1), p_2(X, X_2), p_3(X, X_3), p_4(X_4, Z) \]
36 Clause Indexing

Observation:

Often, predicates are implemented by case distinction on the first argument.

- Inspecting the first argument, many alternatives can be excluded  
- Failure is earlier detected  
- Backtrack points are earlier removed.  
- Stack frames are earlier popped  

Example (continued):

\[ a(X, Z) \leftarrow p_1(\bar{X}, X_1), p_2(\bar{X}_2, X_2), p_3(\bar{X}_3, X_3), p_4(\bar{X}_4, Z) \]

Ordering of the variables:

\[ \rho = \{X \mapsto 1, Z \mapsto 2, X_3 \mapsto 3, X_2 \mapsto 4, X_1 \mapsto 5\} \]

The resulting code:

- pushenv 5 A: mark B mark C lastmark
- mark A putref 5 putvar 4 putvar 3 putref 2
- putvar 5 call p_2/2 call p_1/2 lastcall p_1/3
- call p_1/2 B: trim 4 C: trim 3
Idea:
- Introduce separate try chains for every possible constructor.
- Inspect the root node of the first argument.
- Depending on the result, perform an indexed jump to the appropriate try chain.

Assume that the predicate \( p/k \) is defined by the sequence \( rr \) of clauses \( r_1 \ldots r_n \). Let \( \text{tchains} \ rr \) denote the sequence of try chains as built up for the root constructors occurring in unifications \( X_1 = 1 \).

Example:
Consider again the \( \text{app} \)-predicate, and assume that the code for the two clauses start at addresses \( A_1 \) and \( A_2 \), respectively.
Then we obtain the following four try chains:

\[
\begin{align*}
\text{VAR:} & \quad \text{setbp} \quad \text{// variables} \quad \text{NIL:} \quad \text{jump } A_1 \quad \text{// atom } [ ] \\
& \text{try } A_1 \\
& \text{delbtp} \quad \text{CONS:} \quad \text{jump } A_2 \quad \text{// constructor } [ ] \\
& \text{jump } A_2 \\
& \text{ELSE:} \quad \text{fail} \quad \text{// default}
\end{align*}
\]

Then we generate for a predicate \( p/k \):

\[
\text{code} \ rr = \text{putref 1} \\
\quad \text{getNode} \quad \text{// extracts the root label} \\
\quad \text{index } p/k \quad \text{// jumps to the try block} \\
\quad \text{tchains } rr \\
\| A_1 : \text{code} \ r_1 \\
\vdots \\
\| A_n : \text{code} \ r_n
\]

The new instruction \text{fail} takes care of any constructor besides \([ ]\) and \([ ]\)...

\[
\text{fail} = \text{backtrack()}
\]

It directly triggers \text{backtracking} :-)}
The instruction `getNode` returns "R" if the pointer on top of the stack points to an unbound variable. Otherwise, it returns the content of the heap object:

```
switch (H[SP]) {
    case (S, fn): S[SP] = fn; break;
    case (A, a): S[SP] = a; break;
    case (R, _): S[SP] = R;
}
```

The instruction `index p/k` performs an indexed jump to the appropriate try chain:

```
PC = map (p/k, S[SP]);
SP--;```

The function `map()` returns, for a given predicate and node content, the start address of the appropriate try chain. It typically is defined through some hash table.
37 Extension: The Cut Operator

Realistic Prolog additionally provides an operator "!" (cut) which explicitly allows to prune the search space of backtracking.

Example:

\[
\begin{align*}
\text{branch}(X, Y) & \leftarrow p(X), !, q_1(X, Y) \\
\text{branch}(X, Y) & \leftarrow q_2(X, Y)
\end{align*}
\]

Once the queries before the cut have succeeded, the choice is committed. Backtracking will return only to backtrack points preceding the call to the left-hand side ...

324

The Basic Idea:

- We restore the oldBP from our current stack frame;
- We pop all stack frames on top of the local variables.

Accordingly, we translate the cut into the sequence:

\[
\begin{align*}
\text{prune} \\
\text{pushenv } m
\end{align*}
\]

where \( m \) is the number of (still used) local variables of the clause.

325

Example:

Consider our example:

\[
\begin{align*}
\text{branch}(X, Y) & \leftarrow p(X), /, q_1(X, Y) \\
\text{branch}(X, Y) & \leftarrow q_2(X, Y)
\end{align*}
\]

We obtain:

\[
\begin{align*}
\text{setbp} & A: \text{pushenv 2} \\
\text{try A} & C: \text{prune} \\
\text{delbp} & \text{putref 1} \\
\text{jump B} & \text{call p/1}
\end{align*}
\]

326

Example:

Consider our example:

\[
\begin{align*}
\text{branch}(X, Y) & \leftarrow p(X), /, q_1(X, Y) \\
\text{branch}(X, Y) & \leftarrow q_2(X, Y)
\end{align*}
\]

We obtain:

\[
\begin{align*}
\text{setbp} & A: \text{pushenv 2} \\
\text{try A} & C: \text{prune} \\
\text{delbp} & \text{putref 1} \\
\text{jump B} & \text{call p/1}
\end{align*}
\]
The new instruction `prune` simply restores the backtrack pointer:

\[ BP = BP_{old}; \]

Problem:

If a clause is `single`, then (at least so far ;-) we have not stored the old `BP` inside the stack frame :-(

\[ \rightarrow \rightarrow \rightarrow \]

For the cut to work also with `single-clause` predicates or try chains of length 1, we insert an extra instruction `setcut` before the clausal code (or the jump):

\[ BP_{old} = BP; \]
The Final Example: Negation by Failure

The predicate $\text{notP}$ should succeed whenever $p$ fails (and vice versa). $\triangleright$

$$\text{notP}(X) \leftarrow p(X), \text{fail}$$
$$\text{notP}(X) \leftarrow$$

where the goal fail never succeeds. Then we obtain for notP:

setbtp A: pushenv 1 C: prune B: pushenv 1
try A mark C
delbtp putref 1
jump B call p/1
popenv

The Final Example: Negation by Failure

The predicate $\text{notP}$ should succeed whenever $p$ fails (and vice versa). $\triangleright$

$$\text{notP}(X) \leftarrow p(X), \text{fail}$$
$$\text{notP}(X) \leftarrow$$

where the goal fail never succeeds. Then we obtain for notP:

setbtp A: pushenv 1 C: prune B: pushenv 1
try A mark C pushenv 1 popenv
delbtp putref 1 fail
jump B call p/1 popenv