Notes on Functions

Dan Beatty

April 13, 2006

Functions in procedural languages have the following properties at definition

- Location of the code
- Return value
- Parameters (number and types of parameters).
- Activation Records have
  - Pass by reference or value (pass by value in this case)
  - Pass by value requires memory allocation at function activation time.
  - the activation record
  - the call by value information
  - return location get placed on the stack
  - All local variable to the calling scope
- Any calls to the function must include the activation record and information contained with in.
- Activation records are activated (pushed) and deactivated (popped).

Functions cause the quad table to have 2 more entries.

- where does it begin
- Active status

Assumptions on functions

1. One function active at a time (disallowing a function calling another)
2. Nesting of function is not allowed either.
3. All function variable references will be local to the function. (no globals)
4. No local variables other than the parameters themselves.

Table 1: The production for a function will need to have the following

FUNC $\rightarrow$ function id (params) { S }
PARAMS $\rightarrow$ id P1
P1 $\rightarrow$, id P1 — ϵ

Table 2: In fix production handling function

F $\rightarrow$ id \| cons \| (E) \| func (args)
S $\rightarrow$ \| func := E \|

Semantic Actions
push transfer args compare return address jump to quads to and from

- Header for active and activation record
- which function ST

Active

- ST entry for the active function
- Header to the AR stack

```
A | function := id
--+------------------------------------------------------------------------------
B | i := 2 ;
  | param := pop;
  | while param ≠ # do
  | {
  |     ST [param, context ] := function ;
  |     i := i +1 ;
  |     param := pop
  | }
  | ST [ function , # params] := i - 2; // Number of parameters
  | ST [next , name] := function.ret ;
  | next ++ ;
  | i := 2; // offset to be kept
  | n := function + function [ function # params ] ;
  | for j := function + 1 to n do
  | {
  |     ST [j, loc] := i;
  |     i ++ ;
  | }
  | ST [function , Loc ] := NQ;
--+------------------------------------------------------------------------------
C | push (function, ret).                          return := active_ar^.ar[1] ;
  | genquads ( pop, function, _ , _ )              manage ar stack
  |                                                and if ar stack
  |                                                is empty assign zero - active function
```

```
A   genquad ( := , push (id) , _ , active )          | active function , active ar (2 items)
    n := st [id, # of params ] ;                     |
    i := 2                                           |
    for j := n downto i do                           |
    {                                                |
                          active_ar^. ar[j] := pop ; |
    }                                                |
    empty := pop ;                                   |
    genquad (jmp, _, _ , st [id, loc] ) ;            |
    active ar^. ar[1] := NQ;                         |
```