过程调用

1 含有局部变量的程序语言 WhileDL 与语义

下面在程序语句中增加局部变量声明。

```
C :: = skip
       V = E | * E = E |
       C; C |
       if (E) then { C } else { C } |
       while (E) do { C } |
       var V; C
```

```
Inductive com : Type :=
  | CSkip: com
  | CAsgnVar (x: var_name) (e: expr): com
  | CAsgnDeref (e1 e2: expr): com
  | CSeq (c1 c2: com): com
  | CIf (e: expr) (c1 c2: com): com
  | CWhile (e: expr) (c: com): com
  | CLocalVar (x: var_name) (c: com): com.
```

```
Definition set_addr
             (x: var_name)
             (l1 l2: Z):
  state -> state -> Prop :=
  fun s1 s2 =>
    (s1.(env) x = l1 /\ s2.(env) x = l2) /\
    (forall y, x <> y -> s1.(env) y = s2.(env) y).
```

```
Definition alloc_mem (l: Z) (v: val):
  state -> state -> Prop :=
  fun s1 s2 =>
    (s1.(mem) l = None /\ s2.(mem) l = Some v) /\
    (forall l', l <> l' -> s1.(mem) l' = s2.(mem) l').
```

```
Definition dealloc_mem (l: Z):
  state -> state -> Prop :=
  fun s1 s2 =>
    (s1.(mem) l <> None /\ s2.(mem) l = None) /\
    (forall l', l <> l' -> s1.(mem) l' = s2.(mem) l').
```

```
Definition alloc_mem_err: state -> Prop :=
  fun s => forall l, s.(mem) l <> None.
```

```
Definition dealloc_mem_err (l: Z): state -> Prop :=
  fun s => s.(mem) l = None.
```

```
Definition local_var_sem (x: var_name) (D: CDenote): CDenote :=
  {|
    nrm := ⋃ (fun l1 =>
                ⋃ (fun l2 =>
                     (set_addr x l1 l2 ∩ alloc_mem l2 Vuninit) ∘
                     D.(nrm) ∘
                     (set_addr x l2 l1 ∩ dealloc_mem l2)));
    err := ⋃ (fun l1 =>
                ⋃ (fun l2 =>
                     (set_addr x l1 l2 ∩ alloc_mem l2 Vuninit) ∘
                     D.(err))) ∪
           ⋃ (fun l1 =>
                ⋃ (fun l2 =>
                     (set_addr x l1 l2 ∩ alloc_mem l2 Vuninit) ∘
                     D.(nrm) ∘
                     dealloc_mem_err l2)) ∪
           alloc_mem_err;
    inf := ⋃ (fun l1 =>
                ⋃ (fun l2 =>
                     (set_addr x l1 l2 ∩ alloc_mem l2 Vuninit) ∘
                     D.(inf)))
  |}.
```

```
Fixpoint eval_com (c: com): CDenote :=
  match c with
  | CSkip =>
      skip_sem
  | CLocalVar x c =>
      local_var_sem x (eval_com c)
  | CAsgnVar X e =>
      asgn_var_sem X (eval_r e)
  | CAsgnDeref e1 e2 =>
      asgn_deref_sem (eval_r e1) (eval_r e2)
  | CSeq c1 c2 =>
      seq_sem (eval_com c1) (eval_com c2)
  | CIf e c1 c2 =>
      if_sem (eval_r e) (eval_com c1) (eval_com c2)
  | CWhile e c1 =>
      while_sem (eval_r e) (eval_com c1)
  end.
```

2 含过程调用的程序语言 WhileProc

```
C :: = skip |
       V = E | * E = E |
       P(E, E, ..., E) |
       C; C |
       if (E) then { C } else { C } |
       while (E) do { C } |
       var V; C
```

```
Definition proc_name: Type := string.
```

```
Inductive com : Type :=
  | CSkip: com
  | CAsgnVar (x: var_name) (e: expr): com
  | CAsgnDeref (e1 e2: expr): com
  | CSeq (c1 c2: com): com
  | CIf (e: expr) (c1 c2: com): com
  | CWhile (e: expr) (c: com): com
  | CLocalVar (x: var_name) (c: com): com
  | CProcCall (f: proc_name) (es: list expr).
```

过程 p 的组成部分

- 过程名：`p.(name_of_proc)`
- 过程形参列表：`p.(args_of_proc)`
- 过程体：`p.(body_of_proc)`

```
Record proc: Type := {
  name_of_proc: proc_name;
  body_of_proc: com;
  args_of_proc: list var_name;
}.
```

```
Definition prog: Type := list proc.
```

3 WhileProc 的指称语义

程序状态的定义

- 全局状态：state_G := `Z -> val`
- 局部状态：state
  - state.(env) : `var_name -> int64`
  - state.(mem) : `Z -> val`

假设 $\chi : \text{proc\_name} \to \mathcal{P}(\mathbb{Z}^* \times \text{state}_G \times \text{state}_G)$ 描述被调用过程的行为
那么

- 表达式的语义 $[\![e]\!].(\text{nrm}) \subseteq \text{state} \times \mathbb{Z}$
- 语句的语义 $[\![c]\!](\chi).(\text{nrm}) \subseteq \text{state} \times \text{state}$
- 过程的语义 $[\![p]\!](\chi).(\text{nrm}) \subseteq \mathbb{Z}^* \times \text{state}_G \times \text{state}_G$

$(s, s') \in [\![P(e_1, e_2, ..., e_k)]\!](\chi).(\text{nrm})$ 当且仅当
存在 n_1, n_2, ..., n_k 使得：

- $(s, n_1) \in [\![e_1]\!].(\text{nrm})$
- $(s, n_2) \in [\![e_2]\!].(\text{nrm})$
- ...
- $(s, n_k) \in [\![e_k]\!].(\text{nrm})$

- $([n_1, n_2, ..., n_k], s.(\text{mem}), s'.(\text{mem})) \in \chi(P)$
- $s.(\text{env}) = s'.(\text{env})$

程序语句的指称语义

- $[\![c_1; c_2]\!](\chi).(\text{nrm}) = [\![c_1]\!](\chi).(\text{nrm}) \circ [\![c_2]\!](\chi).(\text{nrm})$
- $[\![\text{if } (e) \text{ then } \{c_1\} \text{ else } \{c_2\}]\!](\chi).(\text{nrm}) =$

  $\text{test\_true}([\![e]\!]) \circ [\![c_1]\!](\chi).(\text{nrm}) \cup$

  $\text{test\_false}([\![e]\!]) \circ [\![c_2]\!](\chi).(\text{nrm})$
- $[\![\text{while } (e) \text{ do } \{c\}]\!](\chi).(\text{nrm}) =$

  the smallest X :

  $$X = \text{test\_true}([\![e]\!]) \circ [\![c]\!](\chi).(\text{nrm}) \circ X \ \cup \ \text{test\_false}([\![e]\!])$$

过程的指称语义 $([n_1, n_2, ..., n_k], s_1^G, s_2^G) \in [\![p]\!](\chi).(\text{nrm})$ 当且仅当存在 s'_1,s'_2 使得

- $(s_1^G \uplus s'_1, s_2^G \uplus s'_2) \in [\![p.(\text{body\_of\_proc})]\!](\chi).(\text{nrm})$
- 对于任意地址 p,

  $s'_1(p) \neq \epsilon$ 当且仅当

  $s'_2(p) \neq \epsilon$ 当且仅当

  $p = s'_1.(\text{env})(x)$ 是某个参数 x 的地址（$x \in p.(\text{args\_of\_proc})$）
- 对于 p 的第 i 个参数 $x_i \in p.(\text{args\_of\_proc})$, $s'_1.(\text{mem})(s'_1.(\text{env})(x_i)) = n_i$

程序 $p_1, p_2, ..., p_k$ 的语义（安全运行终止部分）是最小的 χ 使得

$$[\![p_i]\!](\chi).(\text{nrm}) = \chi(p_i.(\text{name\_of\_proc}))$$