利用集合与关系定义指称语义

1 布尔表达式的指称语义

对于任意布尔表达式 e，我们规定它的语义 $\llbracket e \rrbracket$ 是程序状态集合的子集，是所有使得 e 求值为真的程序状态构成的集合。

- $\llbracket \mathrm{TRUE} \rrbracket = \mathrm{state}$
- $\llbracket \mathrm{FALSE} \rrbracket = \varnothing$
- $s \in \llbracket e_1 < e_2 \rrbracket$ 当且仅当 $\llbracket e_1 \rrbracket (s) < \llbracket e_2 \rrbracket (s)$
- $\llbracket e_1 \&\& e_2 \rrbracket = \llbracket e_1 \rrbracket \cup \llbracket e_2 \rrbracket$
- $\llbracket !e_1 \rrbracket = \mathrm{state} \backslash \llbracket e_1 \rrbracket$

在 Coq 中可以如下定义：

```
Definition true_sem: state -> Prop := Sets.full.
```

```
Definition false_sem: state -> Prop := ∅.
```

```
Definition lt_sem (D1 D2: state -> Z):
  state -> Prop :=
  fun s => D1 s < D2 s.
```

```
Definition and_sem (D1 D2: state -> Prop):
  state -> Prop :=
  D1 ∩ D2.
```

```
Definition not_sem (D: state -> Prop):
  state -> Prop :=
  Sets.complement D.
```

```
Fixpoint eval_expr_bool (e: expr_bool): state -> Prop :=
  match e with
  | ETrue =>
      true_sem
  | EFalse =>
      false_sem
  | ELt e1 e2 =>
      lt_sem (eval_expr_int e1) (eval_expr_int e2)
  | EAnd e1 e2 =>
      and_sem (eval_expr_bool e1) (eval_expr_bool e2)
  | ENot e1 =>
      not_sem (eval_expr_bool e1)
  end.
```

与整数类型表达式的行为等价定义一样，我们也可以用函数相等定义布尔表达式行为等价。

```
Definition bequiv (e1 e2: expr_bool): Prop :=
  ⟦ e1 ⟧ == ⟦ e2 ⟧.
```

下面先证明三个语义算子 `lt_sem` 、 `and_sem` 与 `not_sem` 能保持函数相等，再利用函数相等的性质证明布尔表达式行为等价的性质。

```
#[export] Instance lt_sem_congr:
  Proper (func_equiv _ _ ==>
          func_equiv _ _ ==>
          Sets.equiv) lt_sem.
```

```
#[export] Instance and_sem_congr:
  Proper (Sets.equiv ==>
          Sets.equiv ==>
          Sets.equiv) and_sem.
```

```
#[export] Instance not_sem_congr:
  Proper (Sets.equiv ==> Sets.equiv) not_sem.
```

```
#[export] Instance bequiv_equiv: Equivalence bequiv.
```

```
#[export] Instance ELt_congr:
  Proper (iequiv ==> iequiv ==> bequiv) ELt.
```

```
#[export] Instance EAnd_congr:
  Proper (bequiv ==> bequiv ==> bequiv) EAnd.
```

```
#[export] Instance ENot_congr:
  Proper (bequiv ==> bequiv) ENot.
```

2 程序语句的指称语义定义

$(s_1, s_2) \in [\![c]\!]$ 当且仅当从 s_1 状态开始执行程序 c 会以程序状态 s_2 终止。

2.1 赋值语句

$$[\![x = e]\!] = \{(s_1, s_2) \mid s_2(x) = [\![e]\!](s_1), \text{for any } y \in \text{var\_name}, \text{ if } x \neq y, \ s_1(y) = s_2(y)\}$$

2.2 空语句

$$[\![\text{skip}]\!] = \{(s, s) \mid s \in \text{state}\}$$

2.3 顺序执行语句

$$[\![c_1; c_2]\!] = [\![c_1]\!] \circ [\![c_2]\!] = \{(s_1, s_3) \mid (s_1, s_2) \in [\![c_1]\!], (s_2, s_3) \in [\![c_2]\!]\}$$

2.4 条件分支语句

定义 1：

$$
\begin{aligned}
[\![\text{if } (e) \text{ then } \{c_1\} \text{ else } \{c_2\}]\!] = \quad & \big(\{(s_1, s_2) \mid s_1 \in [\![e]\!]\} \cap [\![c_1]\!]\big) \cup \\
& \big(\{(s_1, s_2) \mid s_1 \notin [\![e]\!]\} \cap [\![c_2]\!]\big)
\end{aligned}
$$

定义 2：

$$
[\![\text{if } (e) \text{ then } \{c_1\} \text{ else } \{c_2\}]\!] = \text{test\_true}([\![e]\!]) \circ [\![c_1]\!] \cup \text{test\_false}([\![e]\!]) \circ [\![c_2]\!]
$$

其中，

$$
\begin{aligned}
&\text{test\_true}([\![e]\!]) = \{(s_1, s_2) \mid s_1 \in [\![e]\!], s_1 = s_2\} \\
&\text{test\_false}([\![e]\!]) = \{(s_1, s_2) \mid s_1 \notin [\![e]\!], s_1 = s_2\}\text{。}
\end{aligned}
$$

2.5 循环语句语句

定义 1：

$$
\begin{aligned}
&\text{iterLB}_0([\![e]\!], [\![c]\!]) = \text{test\_false}([\![e]\!])\text{；} \\
&\text{iterLB}_{n+1}([\![e]\!], [\![c]\!]) = \text{test\_true}([\![e]\!]) \circ [\![c]\!] \circ \text{iterLB}_n([\![e]\!], [\![c]\!])\text{；} \\
&[\![\text{while } (e) \text{ do } \{c\}]\!] = \textstyle\bigcup_{n \in \mathbb{N}} \text{iterLB}_n([\![e]\!], [\![c]\!])\text{。}
\end{aligned}
$$

定义 2：

$$
\begin{aligned}
&\text{boundedLB}_0([\![e]\!], [\![c]\!]) = \varnothing \\
&\text{boundedLB}_{n+1}([\![e]\!], [\![c]\!]) = \text{test\_true}([\![e]\!]) \circ [\![c]\!] \circ \text{boundedLB}_n([\![e]\!], [\![c]\!]) \cup \text{test\_false}([\![e]\!]) \\
&[\![\text{while } (e) \text{ do } \{c\}]\!] = \textstyle\bigcup_{n \in \mathbb{N}} \text{boundedLB}_n([\![e]\!], [\![c]\!])
\end{aligned}
$$