拓展阅读

1 在 Coq 中定义布尔表达式的语法与语义

在 Coq 中可以通过严格定义区分语与语义。下面定义的`var_name`表示变量名。

```
Definition var_name: Type := string.
```

下面的`expr`类型定义了所有布尔表达式的语法树。`EVar`表示布尔变元的情况。`EAnd`表示逻辑连接词：且；`EOr`表示逻辑连接词：或；`ENot`表示逻辑连接词：非。

```
Inductive expr: Type :=
| EVar (x: var_name): expr
| EAnd (e1 e2: expr): expr
| EOr (e1 e2: expr): expr
| ENot (e1: expr): expr.
```

在定义布尔表达式之前先定义真值指派。下面定义表明：每个真值指派是一个从变量名到真值（`bool`）的函数。

```
Definition asgn: Type := var_name -> bool.
```

对于任意布尔表达式`e`与真值指派`J`，`eval e J`表示`e`在真值指派`J`上的真值。

```
Fixpoint eval (e: expr) (J: asgn): bool :=
  match e with
  | EVar x => J x
  | EAnd e1 e2 => (eval e1 J && eval e2 J)%bool
  | EOr e1 e2 => (eval e1 J || eval e2 J)%bool
  | ENot e1 => negb (eval e1 J)
  end.
```

Coq 中还可以通过`Notation`指令引入易读的符号。

```
Definition EVar': string -> expr := EVar.
Declare Custom Entry expr_entry.
Coercion EVar: var_name >-> expr.
Coercion EVar': string >-> expr.
Notation "x" := x
  (in custom expr_entry at level 0, x constr at level 0).
Notation "⌜ e ⌝" := (e: expr)
  (at level 0, e custom expr_entry at level 99).
Notation "( x )" := x
  (in custom expr_entry, x custom expr_entry at level 99).
Notation "f x" := (f x)
                    (in custom expr_entry at level 1,
                     f custom expr_entry,
                     x custom expr_entry at level 0).
Notation "! x" := (ENot x)
  (in custom expr_entry at level 10, no associativity).
Notation "x && y" := (EAnd x y)
  (in custom expr_entry at level 11, left associativity).
Notation "x || y" := (EOr x y)
  (in custom expr_entry at level 12, left associativity).
Notation "⟦ e ⟧ ( J )" := (eval e J)
                            (at level 0,
                             e custom expr_entry at level 99).
```

Coq 中可以证明德摩根律：`! (e1 && e2)` 与 `! e1 || ! e2` 在任何真值指派上的真值都相等。

```
Lemma demorgan_not_and: forall e1 e2 J,
  ⟦ ! (e1 && e2) ⟧ (J) = ⟦ ! e1 || ! e2 ⟧ (J).
(* 证明详见Coq源代码。 *)
```

下面还可以证明许多语义等价的性质。

```
Lemma demorgan_not_or: forall e1 e2 J,
  ⟦ ! (e1 || e2) ⟧ (J) = ⟦ ! e1 && ! e2 ⟧ (J).
(* 证明详见Coq源代码。 *)

Lemma not_involutive: forall e J,
  ⟦ ! (! e) ⟧ (J) = ⟦ e ⟧ (J).
(* 证明详见Coq源代码。 *)
```

但是语义相等与语法相等是不一样的，下面是其中一个例子。

```
Lemma demorgan_not_and_syntax_unequal: forall e1 e2,
  ⌜ ! (e1 && e2) ⌝ <> ⌜ ! e1 || ! e2 ⌝.
(* 证明详见Coq源代码。 *)
```

通过精确的定义布尔表达式的语法与语义，我们还可以在 Coq 中证明，一些语法变换是保持语义不变的。下面 Coq 代码证明了：否定范式 `nnf` 与原表达式是语义等价的。

```
Fixpoint negate (e: expr): expr :=
  match e with
  | EVar x => ENot (EVar x)
  | EAnd e1 e2 => EOr (negate e1) (negate e2)
  | EOr e1 e2 => EAnd (negate e1) (negate e2)
  | ENot e1 => e1
  end.
```

```
Fixpoint nnf (e: expr): expr :=
  match e with
  | EVar x => EVar x
  | EAnd e1 e2 => EAnd (nnf e1) (nnf e2)
  | EOr e1 e2 => EOr (nnf e1) (nnf e2)
  | ENot e1 => negate e1
  end.
```

```
Lemma negate_sound: forall e J,
  ⟦ negate e ⟧ (J) = ⟦ ! e ⟧ (J).
(* 证明详见Coq源代码。 *)

Lemma nnf_sound: forall e J,
  ⟦ nnf e ⟧ (J) = ⟦ e ⟧ (J).
(* 证明详见Coq源代码。 *)
```