语义等价与精化

1 定义与例子

表达式语义等价

```
Record eequiv (e1 e2: expr): Prop := {
  nrm_eequiv:
    ⟦ e1 ⟧.(nrm) == ⟦ e2 ⟧.(nrm);
  err_eequiv:
    ⟦ e1 ⟧.(err) == ⟦ e2 ⟧.(err);
}.
```

表达式精化关系

```
Record erefine (e1 e2: expr): Prop := {
  nrm_erefine:
    ⟦ e1 ⟧.(nrm) ⊆ ⟦ e2 ⟧.(nrm) ∪ (⟦ e2 ⟧.(err) × int64);
  err_erefine:
    ⟦ e1 ⟧.(err) ⊆ ⟦ e2 ⟧.(err);
}.
```

程序语句语义等价

```
Record cequiv (c1 c2: com): Prop := {
  nrm_cequiv: ⟦ c1 ⟧.(nrm) == ⟦ c2 ⟧.(nrm);
  err_cequiv: ⟦ c1 ⟧.(err) == ⟦ c2 ⟧.(err);
  inf_cequiv: ⟦ c1 ⟧.(inf) == ⟦ c2 ⟧.(inf);
}.
```

程序语句精化关系

```
Record crefine (c1 c2: com): Prop := {
  nrm_crefine:
    ⟦ c1 ⟧.(nrm) ⊆ ⟦ c2 ⟧.(nrm) ∪ (⟦ c2 ⟧.(err) × state);
  err_crefine:
    ⟦ c1 ⟧.(err) ⊆ ⟦ c2 ⟧.(err);
  inf_crefine:
    ⟦ c1 ⟧.(inf) ⊆ ⟦ c2 ⟧.(inf) ∪ ⟦ c2 ⟧.(err);
}.
```

精化的例子

```
Lemma const_plus_const_refine: forall n m: Z,
  EConst (n + m) <<= [[n + m]].
```

证明见 Coq 代码。

- 定理：$c_1;(c_2;c_3) \equiv (c_1;c_2);c_3$。

- 证明：程序正常终止的情况

$$\begin{aligned}
& \llbracket c_1;(c_2;c_3) \rrbracket .\mathrm{nrm} \\
= \quad & \llbracket c_1 \rrbracket .\mathrm{nrm} \circ (\llbracket c_2 \rrbracket .\mathrm{nrm} \circ \llbracket c_3 \rrbracket .\mathrm{nrm}) \\
= \quad & (\llbracket c_1 \rrbracket .\mathrm{nrm} \circ \llbracket c_2 \rrbracket .\mathrm{nrm}) \circ \llbracket c_3 \rrbracket .\mathrm{nrm} \\
= \quad & \llbracket (c_1;c_2);c_3 \rrbracket .\mathrm{nrm}
\end{aligned}$$

- 上面证明用到集合运算性质：$A \circ (B \circ C) = (A \circ B) \circ C$。

```
Theorem CSeq_assoc: forall (c1 c2 c3: com),
  [[c1; (c2; c3)]] ~=~ [[(c1; c2); c3]].
Proof.
  intros.
  split.
  + simpl.
    rewrite Rels_concat_assoc.
    reflexivity.
  + simpl.
    rewrite Rels_concat_union_distr_l.
    rewrite Sets_union_assoc.
    rewrite Rels_concat_assoc.
    reflexivity.
  + simpl.
    rewrite Rels_concat_union_distr_l.
    rewrite Sets_union_assoc.
    rewrite Rels_concat_assoc.
    reflexivity.
Qed.
```

- 定理：$\mathrm{if}\ (e)\ \mathrm{then}\{c_1\}\ \mathrm{else}\ \{c_2\};c_3 \equiv \mathrm{if}\ (e)\ \mathrm{then}\{c_1;c_3\}\ \mathrm{else}\ \{c_2;c_3\}$。

- 证明：程序正常终止的情况

$$\begin{aligned}
& \llbracket \mathrm{if}\ (e)\ \mathrm{then}\{c_1\}\ \mathrm{else}\ \{c_2\};c_3 \rrbracket .\mathrm{nrm} \\
= \quad & (\mathrm{test\_true}(\llbracket e \rrbracket) \circ \llbracket c_1 \rrbracket .\mathrm{nrm} \cup \mathrm{test\_false}(\llbracket e \rrbracket) \circ \llbracket c_2 \rrbracket .\mathrm{nrm}) \circ \\
& \llbracket c_3 \rrbracket .\mathrm{nrm} \\
= \quad & \mathrm{test\_true}(\llbracket e \rrbracket) \circ \llbracket c_1 \rrbracket .\mathrm{nrm} \circ \llbracket c_3 \rrbracket .\mathrm{nrm} \cup \\
& \mathrm{test\_false}(\llbracket e \rrbracket) \circ \llbracket c_2 \rrbracket .\mathrm{nrm} \circ \llbracket c_3 \rrbracket .\mathrm{nrm} \\
= \quad & \llbracket \mathrm{if}\ (e)\ \mathrm{then}\{c_1;c_3\}\ \mathrm{else}\ \{c_2;c_3\} \rrbracket .\mathrm{nrm}
\end{aligned}$$

- 上面证明用到集合运算性质：$(A \cup B) \circ C = (A \circ C) \cup (B \circ C)$。

```
Theorem CIf_CSeq: forall e c1 c2 c3,
  [[ if e then { c1 } else { c2 }; c3 ]] ~=~
  [[ if e then { c1; c3 } else { c2; c3 } ]].
Proof.
  intros.
  split.
  + simpl.
    rewrite <- ! Rels_concat_assoc.
    apply Rels_concat_union_distr_r.
  + simpl.
    rewrite ! Rels_concat_union_distr_r.
    rewrite ! Rels_concat_union_distr_l.
    rewrite <- ! Rels_concat_assoc.
    sets_unfold; intros s; tauto.
  + simpl.
    rewrite ! Rels_concat_union_distr_r.
    rewrite ! Rels_concat_union_distr_l.
    rewrite <- ! Rels_concat_assoc.
    sets_unfold; intros s; tauto.
Qed.
```

2 语义等价与精化的性质

接下去，我们介绍语义等价的两条重要性质。其一：语义等价是一种等价关系。

- 对于任意表达式 E, $E \equiv E$。
- 证明：
  - 求值成功的情况：$\llbracket E \rrbracket.\mathrm{nrm} = \llbracket E \rrbracket.\mathrm{nrm}$ （集合相等的自反性）；
  - 求值失败的情况：$\llbracket E \rrbracket.\mathrm{err} = \llbracket E \rrbracket.\mathrm{err}$ （集合相等的自反性）；
- 对于任意表达式 E_1 与 E_2，如果 $E_1 \equiv E_2$，那么 $E_2 \equiv E_1$。
- 证明：
  - 求值成功的情况：由 $E_1 \equiv E_2$ 这一假设可知 $\llbracket E_1 \rrbracket.\mathrm{nrm} = \llbracket E_2 \rrbracket.\mathrm{nrm}$，故 $\llbracket E_2 \rrbracket.\mathrm{nrm} = \llbracket E_1 \rrbracket.\mathrm{nrm}$ （集合相等的对称性）。
  - 求值失败的情况：由 $E_1 \equiv E_2$ 这一假设可知 $\llbracket E_1 \rrbracket.\mathrm{err} = \llbracket E_2 \rrbracket.\mathrm{err}$，故 $\llbracket E_2 \rrbracket.\mathrm{err} = \llbracket E_1 \rrbracket.\mathrm{err}$ （集合相等的对称性）。
- 对于任意表达式 E_1，E_2 与 E_3, 如果 $E_1 \equiv E_2$ 且 $E_2 \equiv E_3$，那么 $E_1 \equiv E_3$。
- 证明：
  - 求值成功的情况：由 $E_1 \equiv E_2$ 与 $E_2 \equiv E_3$ 这两条假设可知 $\llbracket E_1 \rrbracket.\mathrm{nrm} = \llbracket E_2 \rrbracket.\mathrm{nrm}$ 并且 $\llbracket E_2 \rrbracket.\mathrm{nrm} = \llbracket E_3 \rrbracket.\mathrm{nrm}$，故 $\llbracket E_1 \rrbracket.\mathrm{nrm} = \llbracket E_3 \rrbracket.\mathrm{nrm}$（集合相等的传递性）。
  - 求值失败的情况：由 $E_1 \equiv E_2$ 与 $E_2 \equiv E_3$ 这两条假设可知 $\llbracket E_1 \rrbracket.\mathrm{err} = \llbracket E_2 \rrbracket.\mathrm{err}$ 并且 $\llbracket E_2 \rrbracket.\mathrm{err} = \llbracket E_3 \rrbracket.\mathrm{err}$，故 $\llbracket E_1 \rrbracket.\mathrm{err} = \llbracket E_3 \rrbracket.\mathrm{err}$（集合相等的传递性）。

在 Coq 标准库中，`Reflexive`、`Symmetric`、`Transitive` 以及 `Equivalence` 定义了自反性、对称性、传递性以及等价关系。下面证明中，我们统一使用了 `Instance` 关键字，而非之前证明中常用的 `Theorem` 与 `Lemma`，我们将稍后再解释 `Instance` 关键字的特殊作用。

```
Instance eequiv_refl: Reflexive eequiv.
Proof.
  unfold Reflexive; intros.
  split.
  + reflexivity.
  + reflexivity.
Qed.
```

```
Instance eequiv_sym: Symmetric eequiv.
Proof.
  unfold Symmetric; intros.
  split.
  + rewrite H.(nrm_eequiv).
    reflexivity.
  + rewrite H.(err_eequiv).
    reflexivity.
Qed.
```

```
Instance eequiv_trans: Transitive eequiv.
Proof.
  unfold Transitive; intros.
  split.
  + rewrite H.(nrm_eequiv), H0.(nrm_eequiv).
    reflexivity.
  + rewrite H.(err_eequiv), H0.(err_eequiv).
    reflexivity.
Qed.
```

```
Instance eequiv_equiv: Equivalence eequiv.
Proof.
  split.
  + apply eequiv_refl.
  + apply eequiv_sym.
  + apply eequiv_trans.
Qed.
```

下面还可以证明精化关系也具有自反性和传递性。

```
Instance erefine_refl: Reflexive erefine.
Proof.
  unfold Reflexive; intros.
  split.
  + apply Sets_included_union1.
  + reflexivity.
Qed.
```

```
Instance erefine_trans: Transitive erefine.
Proof.
  unfold Transitive; intros.
  split.
  + rewrite H.(nrm_erefine).
    rewrite H0.(nrm_erefine).
    rewrite H0.(err_erefine).
    sets_unfold; intros s1 s2; tauto.
  + rewrite H.(err_erefine).
    rewrite H0.(err_erefine).
    reflexivity.
Qed.
```

并且精化关系在语义等价变换下不变。

```
Instance erefine_well_defined:
  Proper (eequiv ==> eequiv ==> iff) erefine.
Proof.
  unfold Proper, respectful; intros.
  split; intros.
  + split.
    - rewrite <- H.(nrm_eequiv).
      rewrite <- H0.(nrm_eequiv).
      rewrite <- H0.(err_eequiv).
      apply H1.(nrm_erefine).
    - rewrite <- H.(err_eequiv).
      rewrite <- H0.(err_eequiv).
      apply H1.(err_erefine).
  + split.
    - rewrite H.(nrm_eequiv).
      rewrite H0.(nrm_eequiv).
      rewrite H0.(err_eequiv).
      apply H1.(nrm_erefine).
    - rewrite H.(err_eequiv).
      rewrite H0.(err_eequiv).
      apply H1.(err_erefine).
Qed.
```

程序语句间的语义等价关系也是等价关系，程序语句间的精化关系也具有自反性与传递性。

```
Instance cequiv_refl: Reflexive cequiv.
(* 证明详见Coq源代码。 *)
Instance cequiv_sym: Symmetric cequiv.
(* 证明详见Coq源代码。 *)
Instance cequiv_trans: Transitive cequiv.
(* 证明详见Coq源代码。 *)
Instance cequiv_equiv: Equivalence cequiv.
(* 证明详见Coq源代码。 *)
Instance crefine_refl: Reflexive crefine.
(* 证明详见Coq源代码。 *)
Instance crefine_trans: Transitive crefine.
(* 证明详见Coq源代码。 *)
Instance crefine_well_defined:
  Proper (cequiv ==> cequiv ==> iff) crefine.
(* 证明详见Coq源代码。 *)
```

两条重要性质之二是：所有语法连接词能保持语义等价关系（congruence），也能保持精化关系（monotonicity）。下面先证明加法、减法、乘法的情况。

```
Lemma arith_sem1_nrm_congr: forall Zfun (e11 e12 e21 e22: expr),
  e11 ~=~ e12 ->
  e21 ~=~ e22 ->
  arith_sem1_nrm Zfun ⟦ e11 ⟧.(nrm) ⟦ e21 ⟧.(nrm) ==
  arith_sem1_nrm Zfun ⟦ e12 ⟧.(nrm) ⟦ e22 ⟧.(nrm).
Proof.
  sets_unfold.
  intros ? ? ? ? ? ? ? s i.
  unfold arith_sem1_nrm.
  apply ex_iff_morphism; intros i1.
  apply ex_iff_morphism; intros i2.
  apply and_iff_morphism; [apply H.(nrm_eequiv) |].
  apply and_iff_morphism; [apply H0.(nrm_eequiv) |].
  reflexivity.
Qed.
```

```
Lemma arith_sem1_err_congr: forall Zfun (e11 e12 e21 e22: expr),
  e11 ~=~ e12 ->
  e21 ~=~ e22 ->
  ⟦ e11 ⟧.(err) ∪ ⟦ e21 ⟧.(err) ∪
  arith_sem1_err Zfun ⟦ e11 ⟧.(nrm) ⟦ e21 ⟧.(nrm) ==
  ⟦ e12 ⟧.(err) ∪ ⟦ e22 ⟧.(err) ∪
  arith_sem1_err Zfun ⟦ e12 ⟧.(nrm) ⟦ e22 ⟧.(nrm).
Proof.
  sets_unfold.
  intros ? ? ? ? ? ? ? s.
  unfold arith_sem1_err.
  apply or_iff_morphism.
  + apply or_iff_morphism.
    - apply H.(err_eequiv).
    - apply H0.(err_eequiv).
  + apply ex_iff_morphism; intros i1.
    apply ex_iff_morphism; intros i2.
    apply and_iff_morphism; [apply H.(nrm_eequiv) |].
    apply and_iff_morphism; [apply H0.(nrm_eequiv) |].
    reflexivity.
Qed.
```

```
Lemma arith_sem1_nrm_mono: forall Zfun (e11 e12 e21 e22: expr),
  e11 <<= e12 ->
  e21 <<= e22 ->
  arith_sem1_nrm Zfun ⟦ e11 ⟧.(nrm) ⟦ e21 ⟧.(nrm) ⊆
  arith_sem1_nrm Zfun ⟦ e12 ⟧.(nrm) ⟦ e22 ⟧.(nrm) ∪
  ((⟦ e12 ⟧.(err) ∪ ⟦ e22 ⟧.(err) ∪
    arith_sem1_err Zfun ⟦ e12 ⟧.(nrm) ⟦ e22 ⟧.(nrm)) × int64).
Proof.
  intros.
  sets_unfold.
  intros s i.
  unfold arith_sem1_nrm, arith_sem1_err.
  intros [i1 [i2 [? [? ?] ] ] ].
  apply H.(nrm_erefine) in H1.
  apply H0.(nrm_erefine) in H2.
  sets_unfold in H1.
  sets_unfold in H2.
  destruct H1; [| tauto].
  destruct H2; [| tauto].
  left.
  exists i1, i2.
  tauto.
Qed.
```

```
Lemma arith_sem1_err_mono: forall Zfun (e11 e12 e21 e22: expr),
  e11 <<= e12 ->
  e21 <<= e22 ->
  ⟦ e11 ⟧.(err) ∪ ⟦ e21 ⟧.(err) ∪
  arith_sem1_err Zfun ⟦ e11 ⟧.(nrm) ⟦ e21 ⟧.(nrm) ⊆
  ⟦ e12 ⟧.(err) ∪ ⟦ e22 ⟧.(err) ∪
  arith_sem1_err Zfun ⟦ e12 ⟧.(nrm) ⟦ e22 ⟧.(nrm).
Proof.
  intros.
  sets_unfold.
  intros s.
  unfold arith_sem1_err.
  intros [ [? | ?] | [i1 [i2 [? [? ?] ] ] ] ].
  + apply H.(err_erefine) in H1.
    tauto.
  + apply H0.(err_erefine) in H1.
    tauto.
  + apply H.(nrm_erefine) in H1.
    apply H0.(nrm_erefine) in H2.
    sets_unfold in H1.
    sets_unfold in H2.
    destruct H1; [| tauto].
    destruct H2; [| tauto].
    right.
    exists i1, i2.
    tauto.
Qed.
```

很多其它情况的证明是类似的。具体证明详见 Coq 代码
下面把二元运算的情况汇总起来。

```
Instance EBinop_congr: forall op,
  Proper (eequiv ==> eequiv ==> eequiv) (EBinop op).
Proof.
  unfold Proper, respectful.
  intros.
  destruct op.
```

布尔二元运算的情况

```
  + split.
    - apply or_sem_nrm_congr; tauto.
    - apply or_sem_err_congr; tauto.
  + split.
    - apply and_sem_nrm_congr; tauto.
    - apply and_sem_err_congr; tauto.
```

大小比较的情况

```
  + split.
    - apply cmp_sem_nrm_congr; tauto.
    - apply cmp_sem_err_congr; tauto.
  + split.
    - apply cmp_sem_nrm_congr; tauto.
    - apply cmp_sem_err_congr; tauto.
  + split.
    - apply cmp_sem_nrm_congr; tauto.
    - apply cmp_sem_err_congr; tauto.
  + split.
    - apply cmp_sem_nrm_congr; tauto.
    - apply cmp_sem_err_congr; tauto.
  + split.
    - apply cmp_sem_nrm_congr; tauto.
    - apply cmp_sem_err_congr; tauto.
  + split.
    - apply cmp_sem_nrm_congr; tauto.
    - apply cmp_sem_err_congr; tauto.
```

加减乘运算的情况

```
  + split.
    - apply arith_sem1_nrm_congr; tauto.
    - apply arith_sem1_err_congr; tauto.
  + split.
    - apply arith_sem1_nrm_congr; tauto.
    - apply arith_sem1_err_congr; tauto.
  + split.
    - apply arith_sem1_nrm_congr; tauto.
    - apply arith_sem1_err_congr; tauto.
```

除法与取余的情况

```
  + split.
    - apply arith_sem2_nrm_congr; tauto.
    - apply arith_sem2_err_congr; tauto.
  + split.
    - apply arith_sem2_nrm_congr; tauto.
    - apply arith_sem2_err_congr; tauto.
Qed.
```

```
Instance EBinop_mono: forall op,
  Proper (erefine ==> erefine ==> erefine) (EBinop op).
Proof.
  unfold Proper, respectful.
  intros.
  destruct op.
```

布尔二元运算的情况

```
  + split.
    - simpl.
    Locate "×".
    Set Printing All.

      apply (or_sem_nrm_mono x y x0 y0); tauto.
    - apply or_sem_err_mono; tauto.
  + split.
    - apply and_sem_nrm_mono; tauto.
    - apply and_sem_err_mono; tauto.
```

大小比较的情况

```
  + split.
    - apply cmp_sem_nrm_mono; tauto.
    - apply cmp_sem_err_mono; tauto.
  + split.
    - apply cmp_sem_nrm_mono; tauto.
    - apply cmp_sem_err_mono; tauto.
  + split.
    - apply cmp_sem_nrm_mono; tauto.
    - apply cmp_sem_err_mono; tauto.
  + split.
    - apply cmp_sem_nrm_mono; tauto.
    - apply cmp_sem_err_mono; tauto.
  + split.
    - apply cmp_sem_nrm_mono; tauto.
    - apply cmp_sem_err_mono; tauto.
  + split.
    - apply cmp_sem_nrm_mono; tauto.
    - apply cmp_sem_err_mono; tauto.
```

加减乘运算的情况

```
  + split.
    - apply arith_sem1_nrm_mono; tauto.
    - apply arith_sem1_err_mono; tauto.
  + split.
    - apply arith_sem1_nrm_mono; tauto.
    - apply arith_sem1_err_mono; tauto.
  + split.
    - apply arith_sem1_nrm_mono; tauto.
    - apply arith_sem1_err_mono; tauto.
```

除法与取余的情况

```
  + split.
    - apply arith_sem2_nrm_mono; tauto.
    - apply arith_sem2_err_mono; tauto.
  + split.
    - apply arith_sem2_nrm_mono; tauto.
    - apply arith_sem2_err_mono; tauto.
Qed.
```

一元运算的情况是类似的。具体证明详见 Coq 代码，这里只展示结论。

```
Instance EUnop_congr: forall op,
  Proper (eequiv ==> eequiv) (EUnop op).

Instance EUnop_mono: forall op,
  Proper (erefine ==> erefine) (EUnop op).
```

下面证明程序语句中的语法连接词也能保持语义等价性和精化关系。顺序执行保持等价性是比较显然的。

```
Instance CSeq_congr:
  Proper (cequiv ==> cequiv ==> cequiv) CSeq.
Proof.
  unfold Proper, respectful.
  intros c11 c12 ? c21 c22 ?.
  split; simpl.
  + rewrite H.(nrm_cequiv).
    rewrite H0.(nrm_cequiv).
    reflexivity.
  + rewrite H.(nrm_cequiv).
    rewrite H.(err_cequiv).
    rewrite H0.(err_cequiv).
    reflexivity.
  + rewrite H.(nrm_cequiv).
    rewrite H.(inf_cequiv).
    rewrite H0.(inf_cequiv).
    reflexivity.
Qed.
```

为了证明顺序执行能保持精化关系，先证明两条引理。

```
Lemma Rels_times_full_concat2:
  forall {A B C: Type} (X: A -> Prop) (Y: B -> C -> Prop),
    (X × B) ∘ Y ⊆ X × C.
Proof.
  intros.
  sets_unfold.
  intros a c.
  intros [b [? ?] ].
  tauto.
Qed.
```

```
Lemma Rels_times_full_concat1:
  forall {A B: Type} (X: A -> Prop) (Y: B -> Prop),
    (X × B) ∘ Y ⊆ X.
Proof.
  intros.
  sets_unfold.
  intros a.
  intros [b [? ?] ].
  tauto.
Qed.
```

下面证明顺序执行能保持精化关系。

```
Instance CSeq_mono:
  Proper (crefine ==> crefine ==> crefine) CSeq.
Proof.
  unfold Proper, respectful.
  intros c11 c12 ? c21 c22 ?.
  split; simpl.
  + rewrite H.(nrm_crefine).
    rewrite H0.(nrm_crefine).
    rewrite Rels_concat_union_distr_l.
    rewrite ! Rels_concat_union_distr_r.
    rewrite ! Rels_times_full_concat2.
    sets_unfold; intros s1 s2; tauto.
  + rewrite H.(err_crefine).
    rewrite H.(nrm_crefine).
    rewrite H0.(err_crefine).
    rewrite Rels_concat_union_distr_r.
    rewrite Rels_times_full_concat1.
    sets_unfold; intros s; tauto.
  + rewrite H.(inf_crefine).
    rewrite H0.(inf_crefine).
    rewrite H.(nrm_crefine).
    rewrite Rels_concat_union_distr_l.
    rewrite ! Rels_concat_union_distr_r.
    rewrite ! Rels_times_full_concat1.
    sets_unfold; intros s; tauto.
Qed.
```

为了证明 if 语句能保持语义等价关系与精化关系，先证明`test_true`与 `test_false`的性质。

```
Lemma test_true_mono: forall (e1 e2: expr),
  e1 <<= e2 ->
  test_true ⟦ e1 ⟧ ⊆
  test_true ⟦ e2 ⟧ ∪ (⟦ e2 ⟧.(err) × state).
(* 证明详见Coq源代码。 *)

Lemma test_false_mono: forall (e1 e2: expr),
  e1 <<= e2 ->
  test_false ⟦ e1 ⟧ ⊆
  test_false ⟦ e2 ⟧ ∪ (⟦ e2 ⟧.(err) × state).
(* 证明详见Coq源代码。 *)

Lemma test_true_congr: forall (e1 e2: expr),
  e1 ~=~ e2 ->
  test_true ⟦ e1 ⟧ == test_true ⟦ e2 ⟧.
(* 证明详见Coq源代码。 *)

Lemma test_false_congr: forall (e1 e2: expr),
  e1 ~=~ e2 ->
  test_false ⟦ e1 ⟧ == test_false ⟦ e2 ⟧.
(* 证明详见Coq源代码。 *)
```

基于此就可以证明 if 语句能保持语义等价关系与精化关系。

```
Instance CIf_congr:
  Proper (eequiv ==> cequiv ==> cequiv ==> cequiv) CIf.
(* 证明详见Coq源代码。 *)

Instance CIf_mono:
  Proper (erefine ==> crefine ==> crefine ==> crefine) CIf.
(* 证明详见Coq源代码。 *)
```

要证明 while 语句保持语义等价关系，就需要运用集合并集的有关性质，还需要对迭代的次数进行归纳。

```
Instance CWhile_congr:
  Proper (eequiv ==> cequiv ==> cequiv) CWhile.
Proof.
  unfold Proper, respectful.
  intros e1 e2 ? c1 c2 ?.
  split; simpl.
```

正常运行终止的情况。

```
  + apply Sets_indexed_union_congr; intros n.
    induction n; simpl.
    - reflexivity.
    - rewrite IHn.
      rewrite (test_true_congr _ _ H).
      rewrite (test_false_congr _ _ H).
      rewrite H0.(nrm_cequiv).
      reflexivity.
```

运行出错的情况。

```
  + apply Sets_indexed_union_congr; intros n.
    induction n; simpl.
    - reflexivity.
    - rewrite IHn.
      rewrite (test_true_congr _ _ H).
      rewrite H.(err_eequiv).
      rewrite H0.(nrm_cequiv).
      rewrite H0.(err_cequiv).
      reflexivity.
```

运行不终止的情况。

```
  + apply Sets_general_union_congr.
    intros X.
    unfold_CL_defs.
    rewrite H0.(nrm_cequiv).
    rewrite H0.(inf_cequiv).
    rewrite (test_true_congr _ _ H).
    reflexivity.
Qed.
```

要证明 while 语句保持精化关系就更复杂一些了。

```
Definition boundedLB_nrm
             (D0: EDenote)
             (D1: CDenote)
             (n: nat):
  state -> state -> Prop :=
  Nat.iter
    n
    (fun X => test_true D0 ∘ D1.(nrm) ∘ X ∪ test_false D0)
    ∅.

Definition boundedLB_err
             (D0: EDenote)
             (D1: CDenote)
             (n: nat):
  state -> Prop :=
  Nat.iter
    n
    (fun X => test_true D0 ∘ D1.(nrm) ∘ X ∪ test_true D0 ∘ D1.(err) ∪ D0.(err))
    ∅.

Definition is_inf
             (D0: EDenote)
             (D1: CDenote)
             (X: state -> Prop): Prop :=
  X ⊆ test_true D0 ∘ D1.(nrm) ∘ X ∪
      test_true D0 ∘ D1.(inf).


Instance CWhile_mono:
  Proper (erefine ==> crefine ==> crefine) CWhile.
(* 证明详见Coq源代码。 *)
```

下面是一个证明中使用了语义等价的重要性质：语义等价是等价关系、语法连接词保持等价性。

```
Example cequiv_sample: forall (e: expr) (c1 c2 c3 c4: com),
  [[ if (e) then { c1 } else { c2 }; c3; c4 ]] ~=~
  [[ if (e) then { c1; c3 } else { c2; c3 }; c4 ]].
Proof.
  intros.
  rewrite CSeq_assoc.
  rewrite CIf_CSeq.
  reflexivity.
Qed.
```