



软件理论基础与实践

# AUTO: More Automation

熊英飞  
北京大学



# 复习

- IMP包括哪几条指令？
- IMP的操作语义是什么？



# ceval是确定的吗?

```
Theorem ceval_deterministic: forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```



# ceval是确定的吗？

Proof.

```
  intros c st st1 st2 E1 E2.
  generalize dependent st2.
(** [Coq Proof View]
 * 1 subgoal
 *
 *   c : com
 *   st, st1 : state
 *   E1 : st =[ c ]=> st1
 *   =====
 *   forall st2 : state, st =[ c ]=> st2 -> st1 = st2
 *)
```

此处有E1和E2两个归纳定义的relation，我们采用induction证明。为了确保归纳假设的正确性，我们先把st2放回到goal中，并对E1做induction。



# ceval是确定的吗？

```
induction E1; intros st2 E2; inversion E2; subst.
(** [Coq Proof View]
 * 11 subgoals
 *
 *   st2 : state
 *   E2 : st2 =[ skip ]=> st2
 *   =====
 *   st2 = st2
 *
 * subgoal 2 is:
 * (x !-> aeval st a; st) = (x !-> aeval st a; st)
 * subgoal 3 is:
 * ...)
```

inversion之后接subst可以有效减少等式数量。



# ceval是确定的吗?

```
- (* E_Skip *) (** [Coq Proof View]
* 1 subgoal
*
*   st2 : state
*   E2 : st2 =[ skip ]=> st2
*   =====
*   st2 = st2
*)
reflexivity.
```



# ceval是确定的吗?

```
- (* E_Ass *) (** [Coq Proof View]
* 1 subgoal
*
*   st : state
*   a  : aexp
*   x  : string
*   E2 : st =[ x := a ]=> (x !-> aeval st a; st)
*   =====
*   (x !-> aeval st a; st) = (x !-> aeval st a; st)
*)
reflexivity.
```



# ceval是确定的吗?

```
- (* E_Seq *)
*   E1_1 : st =[ c1 ]=> st'
*   E1_2 : st' =[ c2 ]=> st''
*   IHE1_1 : forall st2 : state, st =[ c1 ]=> st2 -> st' = st2
*   IHE1_2 : forall st2 : state, st' =[ c2 ]=> st2 -> st'' = st2
*   H1 : st =[ c1 ]=> st'0
*   H4 : st'0 =[ c2 ]=> st2
*   =====
*   st'' = st2
*)
rewrite (IHE1_1 st'0 H1) in *.
apply IHE1_2. assumption.
```





# ceval是确定的吗?

```
- (* E_IfTrue, b evaluates to true *)
*   H : beval st b = true
*   E1 : st =[ c1 ]=> st'
*   IHE1 : forall st2 : state, st =[ c1 ]=> st2 -> st' = st2
*   E2 : st =[ if b then c1 else c2 end ]=> st2
*   H5 : beval st b = true
*   H6 : st =[ c1 ]=> st2
*   =====
*   st' = st2
*)
apply IHE1. assumption.
```



# ceval是确定的吗?

```
- (* E_IfTrue, b evaluates to false (contradiction) *)
*   H : beval st b = true
*   E1 : st =[ c1 ]=> st'
*   IHE1 : forall st2 : state, st =[ c1 ]=> st2 -> st' = st2
*   E2 : st =[ if b then c1 else c2 end ]=> st2
*   H5 : beval st b = false
*   H6 : st =[ c2 ]=> st2
*   =====
*   st' = st2
*)
rewrite H in H5. discriminate.
```



# ceval是确定的吗?

```
- (* E_IfFalse, b evaluates to true (contradiction) *)
*   H : beval st b = false
*   E1 : st =[ c2 ]=> st'
*   IHE1 : forall st2 : state, st =[ c2 ]=> st2 -> st' = st2
*   E2 : st =[ if b then c1 else c2 end ]=> st2
*   H5 : beval st b = true
*   H6 : st =[ c1 ]=> st2
*   =====
*   st' = st2
*)
rewrite H in H5. discriminate.
```



# ceval是确定的吗?

```
- (* E_IfFalse, b evaluates to false *)
*   H : beval st b = false
*   E1 : st =[ c2 ]=> st'
*   IHE1 : forall st2 : state, st =[ c2 ]=> st2 -> st' = st2
*   E2 : st =[ if b then c1 else c2 end ]=> st2
*   H5 : beval st b = false
*   H6 : st =[ c2 ]=> st2
*   =====
*   st' = st2
*)
apply IHE1. assumption.
```



# ceval是确定的吗?

```
- (* E_WhileFalse, b evaluates to false *)
*   b : bexp
*   c : com
*   st2 : state
*   E2 : st2 =[ while b do c end ]=> st2
*   H, H4 : beval st2 b = false
*   =====
*   st2 = st2
*)
reflexivity.
```



# ceval是确定的吗?

```
- (* E_WhileFalse, b evaluates to true (contradiction) *)
*   H : beval st b = false
*   E2 : st =[ while b do c end ]=> st2
*   H2 : beval st b = true
*   H3 : st =[ c ]=> st'
*   H6 : st' =[ while b do c end ]=> st2
*   =====
*   st = st2
*)
rewrite H in H2. discriminate.
```



# ceval是确定的吗?

```
- (* E_WhileTrue, b evaluates to false (contradiction) *)
*   E2 : st2 =[ while b do c end ]=> st2
*   E1_1 : st2 =[ c ]=> st'
*   H : beval st2 b = true
*   E1_2 : st' =[ while b do c end ]=> st''
*   IHE1_1 : forall st3 : state, st2 =[ c ]=> st3 -> st' = st3
*   IHE1_2 : forall st2 : state, st' =[ while b do c end ]=> st2
-> st'' = st2
*   H4 : beval st2 b = false
*   =====
*   st'' = st2
*)
rewrite H in H4. discriminate.
```



# ceval是确定的吗?

```
- (* E_WhileTrue, b evaluates to true *)
*   H : beval st b = true
*   E1_1 : st =[ c ]=> st'
*   E1_2 : st' =[ while b do c end ]=> st''
*   IHE1_1 : forall st2 : state, st =[ c ]=> st2 -> st' = st2
*   IHE1_2 : forall st2 : state, st' =[ while b do c end ]=> st2
-> st'' = st2
*   E2 : st =[ while b do c end ]=> st2
*   H2 : beval st b = true
*   H3 : st =[ c ]=> st'0
*   H6 : st'0 =[ while b do c end ]=> st2
*   =====
=====
*   st'' = st2
*)
rewrite (IHE1_1 st'0 H3) in *.
apply IHE1_2. assumption. Qed.
```





# ceval是确定的吗?

```
Theorem ceval_deterministic: forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```

Proof.

```
intros c st st1 st2 E1 E2.  
generalize dependent st2.  
induction E1; intros st2 E2; inversion E2; subst.  
- (* E_Skip *) reflexivity.  
- (* E_Ass *) reflexivity.  
- (* E_Seq *)  
  rewrite (IHE1_1 st'0 H1) in *.  
  apply IHE1_2. assumption.
```



# ceval是确定的吗?

- (\* E\_IfTrue, b evaluates to true \*)  
  apply IHE1. assumption.
- (\* E\_IfTrue, b evaluates to false (contradiction) \*)  
  rewrite H in H5. discriminate.
- (\* E\_IfFalse, b evaluates to true (contradiction) \*)  
  rewrite H in H5. discriminate.
- (\* E\_IfFalse, b evaluates to false \*)  
  apply IHE1. assumption.
- (\* E\_WhileFalse, b evaluates to false \*)  
  reflexivity.
- (\* E\_WhileFalse, b evaluates to true (contradiction) \*)  
  rewrite H in H2. discriminate.
- (\* E\_WhileTrue, b evaluates to false (contradiction) \*)  
  rewrite H in H4. discriminate.
- (\* E\_WhileTrue, b evaluates to true \*)  
  rewrite (IHE1\_1 st'0 H3) in \*.  
  apply IHE1\_2. assumption. Qed.



# auto策略

```
Example auto_example_1 : forall (P Q R: Prop),  
  (P -> Q) -> (Q -> R) -> P -> R.
```

Proof.

```
  intros P Q R H1 H2 H3.
```

```
  apply H2. apply H1. assumption.
```

Qed.

```
Example auto_example_1' : forall (P Q R: Prop),  
  (P -> Q) -> (Q -> R) -> P -> R.
```

Proof.

```
  auto.
```

Qed.

auto策略自动搜索可以由一系列intro和apply组成的证明



# auto策略

```
Example auto_example_3 : forall (P Q R S T U: Prop),
```

```
(P -> Q) ->
```

```
(Q -> R) ->
```

```
(R -> S) ->
```

```
(S -> T) ->
```

```
(T -> U) ->
```

```
P ->
```

```
U.
```

Proof.

(\* 如果证明不出来，就什么都不做 \*)

```
auto.
```

(\* 也可以指定搜索的深度 (apply的数量)，默认5 \*)

```
auto 6.
```

Qed.



# 打印auto的内容

```
Example auto_example_4 : forall P Q R : Prop,  
  Q ->  
  (Q -> R) ->  
  P \/\ (Q /\ R).  
Proof. info_auto. Qed.
```

info\_auto可以打印auto的内容

auto的搜索范围除了当前的假设，也包括常见的逻辑命题

auto的搜索范围也包括eq\_refl，即等价于reflexivity

```
(* info auto: *)  
intro.  
intro.  
intro.  
intro.  
intro.  
simple apply or_intror (in core).  
simple apply conj (in core).  
assumption.  
simple apply H0.  
assumption.
```



# 扩展auto的搜索范围

```
Example auto_example_6 : forall n m p : nat,  
  (n <= p -> (n <= m /\ m <= n)) ->  
  n <= p ->  
  n = m.
```

Proof.

```
  info_auto using le_antisym.
```

Qed.

```
(* info auto: *)  
intro.  
intro.  
intro.  
intro.  
intro.  
simple apply le_antisym.  
  simple apply H.  
  assumption.
```

用using扩展单个apply可用的定理



# 扩展auto的搜索范围

```
Hint Resolve le_antisym : core.
```

```
Example auto_example_6 : forall n m p : nat,  
  (n <= p -> (n <= m /\ m <= n)) (* info auto: *)  
  n <= p ->  
  n = m.
```

```
Proof.  
  info_auto.  
Qed.
```

```
intro.  
intro.  
intro.  
intro.  
intro.  
simple apply le_antisym.  
simple apply H.  
assumption.
```

也可以用hint resolve全局扩展auto的范围



# 扩展auto的搜索范围

```
Definition is_fortytwo x := (x = 42).  
  
Hint Unfold is_fortytwo : core.  
  
Example auto_example_7' : forall x,  
  (x <= 42 /\ 42 <= x) -> is_fortytwo x.  
Proof.  
  info_auto. (* try also: info_auto. *)  
Qed.
```

用hint unfold允许auto展开定义

还可以用Hint Constructors c : core.  
允许auto使用归纳定义c的所有  
constructor

```
(* info auto: *)  
intro.  
intro.  
unfold is_fortytwo (in core).  
simple apply le_antisym (in core).  
assumption.
```





# 采用auto简化证明

```
Theorem ceval_deterministic': forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```

Proof.

```
intros c st st1 st2 E1 E2.  
generalize dependent st2;  
  induction E1; intros st2 E2; inversion E2; subst; auto.  
- (* E_Seq *)  
  rewrite (IHE1_1 st'0 H1) in *.  
  auto.  
- (* E_IfTrue *)  
  .....
```



# 采用auto简化证明

```
- (* E_IfTrue *)
+ (* b evaluates to false (contradiction) *)
  rewrite H in H5. discriminate.
- (* E_IfFalse *)
+ (* b evaluates to true (contradiction) *)
  rewrite H in H5. discriminate.
- (* E_WhileFalse *)
+ (* b evaluates to true (contradiction) *)
  rewrite H in H2. discriminate.
(* E_WhileTrue *)
- (* b evaluates to false (contradiction) *)
  rewrite H in H4. discriminate.
- (* b evaluates to true *)
  rewrite (IHE1_1 st'0 H3) in *.
  auto.
```

Qed.



# Proof with

采用proof with t允许我们写 “t1...” 来代表 “t1;t”

```
Theorem ceval_deterministic'_alt: forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```

Proof with auto.

```
intros c st st1 st2 E1 E2;  
generalize dependent st2;  
induction E1;  
  intros st2 E2; inversion E2; subst...  
- (* E_Seq *)  
  rewrite (IHE1_1 st'0 H1) in *...
```



# 分析剩下的证明

```
- (* E_IfTrue *)
+ (* b evaluates to false (contradiction) *)
  rewrite H in H5. discriminate.
- (* E_IfFalse *)
+ (* b evaluates to true (contradiction) *)
  rewrite H in H5. discriminate.
- (* E_WhileFalse *)
+ (* b evaluates to true (contradiction) *)
  rewrite H in H2. discriminate.
(* E_WhileTrue *)
- (* b evaluates to false (contradiction) *)
  rewrite H in H4. discriminate.
- (* b evaluates to true *)
  rewrite (IHE1_1 st'0 H3) in *.
  auto.
```

Qed.



# 分析剩下的证明

- 大量的证明通过矛盾排除不可能的情况
- 这些证明都首先找到如下形式的两个假设
  - H1: XXXX = false
  - H2: XXXX = true
- 然后通过如下策略完成证明
  - `rewrite H1 in H2. discriminate.`
- 定义策略替换如上指令
  - `Ltac rwd H1 H2 :=  
rewrite H1 in H2; discriminate.`



# 采用rwd简化证明

```
- (* E_IfTrue *)
+ (* b evaluates to false (contradiction) *)
  rwd H H5.
- (* E_IfFalse *)
+ (* b evaluates to true (contradiction) *)
  rwd H H5.
- (* E_WhileFalse *)
+ (* b evaluates to true (contradiction) *)
  rwd H H2.
(* E_WhileTrue *)
- (* b evaluates to false (contradiction) *)
  rwd H H4.
- (* b evaluates to true *)
  rewrite (IHE1_1 st'0 H3) in *.
  auto.
```

Qed.



# 采用match goal简化证明

```
Ltac find_rwd :=  
  match goal with  
    H1: ?E = true,  
    H2: ?E = false  
  |- _  
  => rwd H1 H2  
end.
```

H1: ... 匹配一条假设

|- ... 匹配目标

?E 元变量，多次出现需匹配上同样的表达式

\_ 匹配任意

=> ... 匹配成功后执行的策略。

如果有多个匹配，则任选一个不会导致策略失败的匹配执行。

如果没有不导致策略失败的匹配，则整个策略失败。

} 没查到文献，我实验出来的



# 采用match goal简化证明

```
Theorem ceval_deterministic''': forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```

Proof.

```
intros c st st1 st2 E1 E2.  
generalize dependent st2;  
induction E1; intros st2 E2; inversion E2; subst;  
try find_rwd; auto.  
- (* E_Seq *)  
  rewrite (IHE1_1 st'0 H1) in *.  
  auto.  
- (* E_WhileTrue *)  
  + (* b evaluates to true *)  
    rewrite (IHE1_1 st'0 H3) in *.  
    auto. Qed.
```





# 分析剩下的证明

```
Theorem ceval_deterministic''': forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```

Proof.

```
intros c st st1 st2 E1 E2.  
generalize dependent st2;  
induction E1; intros st2 E2; inversion E2; subst;  
try find_rwd; auto.  
- (* E_Seq *)  
  rewrite (IHE1_1 st'0 H1) in *.  
  auto.  
- (* E_WhileTrue *)  
  + (* b evaluates to true *)  
    rewrite (IHE1_1 st'0 H3) in *.  
    auto.
```

Qed.



# 分析剩下的证明

- 寻找合适归纳假设并应用
- 归纳假设：
  - $E1\_1: st = [ c1 ] \Rightarrow st'$
  - $IHE1\_1 : \text{forall } st2 : \text{state}, st = [ c1 ] \Rightarrow st2 \rightarrow st' = st2$
  - $H1: st = [ c1 ] \Rightarrow st' \theta$
- 归纳假设的模式：
  - $H0: ?P ?E,$
  - $H1: \text{forall } x, ?P x \rightarrow ?E = ?R,$
  - $H2: ?P ?X$



# 采用match goal简化证明

```
Ltac find_eqn :=  
  match goal with  
    H0: ?P ?E,  
    H1: forall x, ?P x -> ?E = ?R,  
    H2: ?P ?X  
  |- _ => rewrite (H1 X H2) in *  
end.
```

H0并不会被用在证明中，可以省略



# 采用match goal简化证明

```
Ltac find_eqn :=  
  match goal with  
    H1: forall x, ?P x -> ?L = ?R,  
    H2: ?P ?X  
  |- _ => rewrite (H1 X H2) in *  
end.
```

虽然Ltac中的H2有可能匹配上E1\_1，但这会导致rewrite报错，所以会被忽略。

```
E1_1: st =[ c1 ]=> st'
```

```
IHE1_1 : forall st2 : state, st =[ c1 ]=> st2 -> st' = st2
```

```
H1: st =[ c1 ]=> st'0
```



# 采用match goal简化证明

```
Theorem ceval_deterministic''''': forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```

Proof.

```
intros c st st1 st2 E1 E2.  
generalize dependent st2;  
induction E1; intros st2 E2; inversion E2; subst;  
try find_rwd; try find_eqn; auto.
```

Qed.



# 可维护性

- 可维护性是基于证明的高可信软件开发的主要障碍之一
- 普通软件修改
  - 改需求->改代码->改测试
  - 改测试通常比较容易，且常被（不靠谱的工程师）省略
- 高可信软件修改
  - 改需求->改代码->改证明
  - 证明所需的代码量通常是功能代码的数倍
    - 并行CertiKOS: 6500行功能代码，约10万行证明代码 [OSDI16]
- 采用智能化证明脚本可能显著降低改证明工作量



# 改需求： 加入REPEAT命令

```
Inductive com : Type :=  
  | CSkip  
  | CAss (x : string) (a : aexp)  
  | CSeq (c1 c2 : com)  
  | CIf (b : bexp) (c1 c2 : com)  
  | CWhile (b : bexp) (c : com)  
  | CRepeat (c : com) (b : bexp).
```



# 改代码

```
Notation "'repeat' x 'until' y 'end'" :=
  (CRepeat x y)
  (in custom com at level 0,
   x at level 99, y at level 99).
```

```
Inductive ceval : com -> state -> state -> Prop :=
```

.....

```
| E_RepeatEnd : forall st st' b c,
  st =[ c ]=> st' ->
  beval st' b = true ->
  st =[ repeat c until b end ]=> st'
| E_RepeatLoop : forall st st' st'' b c,
  st =[ c ]=> st' ->
  beval st' b = false ->
  st' =[ repeat c until b end ]=> st'' ->
  st =[ repeat c until b end ]=> st''
```





# 改证明

```
Theorem ceval_deterministic': forall c st st1 st2,  
  st =[ c ]=> st1 ->  
  st =[ c ]=> st2 ->  
  st1 = st2.
```

Proof.

```
intros c st st1 st2 E1 E2.  
generalize dependent st2;  
induction E1; intros st2 E2; inversion E2; subst;  
try find_eqn; try find_rwd; auto.
```

Qed.

需要交换顺序是因为repart需要先应用find\_eqn找到等价状态之后才能推出矛盾



# 复习：程序运行的例子

```
Example ceval_example1: empty_st = [  
  X := 2;  
  if (X <= 1) then Y := 3  
  else Z := 4 end  
]=> (Z !-> 4 ; X !-> 2).
```

能否让Coq自动推出这个参数？

Proof.

```
apply E_Seq with (X !-> 2).  
- (* empty_st = [ X := 2 ] => (X !-> 2) *)  
  apply E_Ass. reflexivity.  
- (* (X !-> 2)  
    = [ if X <= 1 then Y := 3 else Z := 4 end ] =>  
      (Z !-> 4; X !-> 2) *)  
  apply E_IfFalse.  
  reflexivity.  
  apply E_Ass. reflexivity.
```

Qed.



# 采用eapply

Existential  
variable  
存在变量

Proof.

```
eapply E_Seq.  
(* (1/2) empty_st =[ X := 2 ]=> ?st'  
   (2/2) ?st' =[ if X <= 1 then Y := 3 else Z := 4 end ]=>  
           (Z !-> 4; X !-> 2) *)  
- (* empty_st =[ X := 2 ]=> ?st' *)  
  apply E_Ass.  
  (* aeval empty_st 2 = ?n *)  
  reflexivity.  
- (* (X !-> aeval empty_st 2)  
   =[ if X <= 1 then Y := 3 else Z := 4 end ]=>  
   (Z !-> 4; X !-> 2) *)  
  apply E_IfFalse. reflexivity.  
  apply E_Ass. reflexivity.
```

Qed.

存在变量会在后续规则应用中被实例化。

其他生成apply的策略也有e-版本，比如exists, constructor, auto, assumption



# 采用eapply

Proof.

```
eapply E_Seq.
```

```
(* (1/2) empty_st =[ X := 2 ]=> ?st'  
   (2/2) ?st' =[ if X <= 1 then Y := 3 else Z := 4 end ]=>  
           (Z !-> 4; X !-> 2) *)
```

- Show Proof.

```
(* (E_Seq <{ X := 2 }>  
   <{ if X <= 1 then Y := 3 else Z := 4 end }> empty_st  
   ?st' (Z !-> 4; X !-> 2) ?Goal ?Goal0)  
*)
```

proof object中同样包含存在变量



# eauto例子

```
Hint Constructors ceval : core.  
Hint Transparent state total_map : core.
```

```
Example ceval'_example1':  
  empty_st = [  
    X := 2;  
    if (X <= 1)  
      then Y := 3  
      else Z := 4  
    end  
  ]=> (Z !-> 4 ; X !-> 2).  
Proof.  
  info_eauto.  
Qed.
```

使得一些定义对  
auto策略透明（自  
动被展开）\*

```
(* info eauto: *)  
simple eapply E_Seq.  
simple apply E_Ass.  
  simple apply @eq_refl.  
simple apply E_IfFalse.  
  simple apply @eq_refl.  
simple apply E_Ass.  
  simple apply @eq_refl.
```

\* Transparent和unfold有时可以互换，具体区别不是很清楚（手册中写得很模糊），不过用Transparent的时候不会生成显式的unfold命令。



# 存在变量使用约束

```
Lemma silly1 : forall (P : nat -> nat -> Prop) (Q : nat -> Prop),  
  (forall x y : nat, P x y) ->  
  (forall x y : nat, P x y -> Q x) ->  
  Q 42.
```

Proof.

```
intros P Q HP HQ. eapply HQ.  
(* P 42 ?y *)  
apply HP.  
(* There are unfocused goals. *)
```

Fail Qed.

```
(* Some unresolved existential variables remain *)
```

存在变量在Qed之前必须被赋值

如何解决这个问题？



# 存在变量使用约束

```
Lemma silly1 : forall (P : nat -> nat -  
> Prop) (Q : nat -> Prop),  
  (forall x y : nat, P x y) ->  
  (forall x y : nat, P x y -> Q x) ->  
  Q 42.
```

Proof.

```
intros P Q HP HQ. eapply HQ.  
(* P 42 ?y *)  
apply (HP _ 1).
```

Qed.

实际传入值（当然，eapply的意义也就没了）。



# 存在变量使用约束

```
Lemma silly2 :  
  forall (P : nat -> nat -> Prop) (Q : nat -> Prop),  
  (exists y, P 42 y) ->  
  (forall x y : nat, P x y -> Q x) ->  
  Q 42.
```

Proof.

```
intros P Q HP HQ. eapply HQ.
```

```
(* P 42 ?y *)
```

```
destruct HP as [y HP'].
```

```
(*
```

```
  y: nat
```

```
  HP': P 42 y
```

```
*)
```

```
Fail apply HP'.
```

```
(* "y" is not in its scope *)
```

存在变量实例化之后不能包含引入时还没被定义的变量





# 存在变量使用约束

Show Proof.

```
(* (fun (P : nat -> nat -> Prop) (Q : nat -> Prop)
      (HP : exists y : nat, P 42 y)
      (HQ : forall x y : nat, P x y -> Q x) =>
      HQ 42 ?y
      match HP as e return (P 42 ?y@{HP:=e}) with
      | ex_intro _ x p =>
        (fun (y : nat) (HP' : P 42 y) => ?Goal) x p
      end) *)
```

因为 $y$ 是作用域不包括 $?y$

如何解决?



# 存在变量使用约束

```
Lemma silly2_fixed :  
  forall (P : nat -> nat -> Prop) (Q : nat -> Prop),  
  (exists y, P 42 y) ->  
  (forall x y : nat, P x y -> Q x) ->  
  Q 42.
```

Proof.

```
intros P Q HP HQ. destruct HP as [y HP'].  
eapply HQ. apply HP'.
```

Qed.

通过交换destruct和eapply的顺序可简单解决



# 存在变量使用约束

```
Lemma silly2_fixed :  
  forall (P : nat -> nat -> Prop) (Q : nat -> Prop),  
  (exists y, P 42 y) ->  
  (forall x y : nat, P x y -> Q x) ->  
  Q 42.
```

Proof.

```
intros P Q HP HQ. destruct HP as [y HP'].  
eapply HQ. eassumption.
```

Qed.

最后一步也可以换成eassumption（注意因为涉及存在变量，assumption不工作）



# 作业

- 无（毕竟是让电脑干活的一章）