

C*: 面向开发与证明一体化的 C 语言增强

曹奕远 北京大学 · 程序设计语言研究室

CCF 程序设计语言会议 · 2026 年 8 月 5 日

背景

大型证明实践：可行、有益，但代价不菲

seL4

微内核的功能正确性证明

Klein et al., SOSP'09

CertiKOS

并发操作系统内核的分层精化证明

Gu et al., OSDI'16

μ C/OS-II

实时操作系统内核的关键组件证明

Xu et al., CAV'16

背景

大型证明实践：可行、有益，但代价不菲

seL4

微内核的功能正确性证明

Klein et al., SOSP'09

CertiKOS

并发操作系统内核的分层精化证明

Gu et al., OSDI'16

μC/OS-II

实时操作系统内核的关键组件证明

Xu et al., CAV'16

seL4 的代价 Klein et al., CACM'10; Heiser, LCA'15

≈ 2.5 人年
抽象行为规约与具体 C 实现

≈ 20 人年
Isabelle 框架构建 + 证明开发

开发成本 **400\$** 每行代码

背景

大型证明实践：可行、有益，但代价不菲

seL4

微内核的功能正确性证明

Klein et al., SOSP'09

CertiKOS

并发操作系统内核的分层精化证明

Gu et al., OSDI'16

μC/OS-II

实时操作系统内核的关键组件证明

Xu et al., CAV'16

seL4 的代价 Klein et al., CACM'10; Heiser, LCA'15

≈ 2.5 人年
抽象行为规约与具体 C 实现

≈ 20 人年
Isabelle 框架构建 + 证明开发

开发成本 **400\$** 每行代码

如何进一步降低**程序形式化证明**的代价与门槛？

大型证明实践：可行、有益，但代价不菲

seL4

微内核的功能正确性证明

Klein et al., SOSP'09

CertiKOS

并发操作系统内核的分层精化证明

Gu et al., OSDI'16

μC/OS-II

实时操作系统内核的关键组件证明

Xu et al., CAV'16

seL4 的代价 Klein et al., CACM'10; Heiser, LCA'15

≈ 2.5 人年
抽象行为规约与具体 C 实现

≈ 20 人年
Isabelle 框架构建 + 证明开发

开发成本 **400\$** 每行代码

如何进一步降低**程序形式化证明**的代价与门槛？

我们以 VST 为代表，展示现有范式如何验证链表原地反转算法的正确性。¹

1. 证明来自 Software Foundations · Verifiable C · [Verif_reverse](#)

验证状态实时展示

The screenshot shows the Verif reverse.v editor with the following components:

- Left Panel (Code):** Displays the Verif reverse.v code. A blue box with the text "进入函数体" (Enter function body) points to the `start_function` line (line 70).
- Right Panel (Proof):** Displays the proof state. A blue box with the text "初始符号状态" (Initial symbolic state) points to the `Goal 1` section, which shows the initial state of the proof.
- Bottom Panel (Messages):** Displays the messages generated during the proof process.

The code in the left panel includes a `Lemma body_reverse` and a `Proof` block. The proof block starts with `start_function` and contains a `forward_while` loop. The right panel shows the proof state with `Goal 1` and `Messages`.

```
struct list *reverse(struct list *p) {
    struct list *w = NULL;
    struct list *v = p;
    while (v) {
        struct list *t = v->tail;
        v->tail = w;
        w = v;
        v = t;
    }
    return w;
}
```

```
WITH sigma : list val, p: val
PRE [ tptr t_list ]
| PROP () PARAMS (p) SEP (listrep sigma p)
POST [ (tptr t_list) ]
| EX q:val,
| PROP () RETURN (q) SEP (listrep(rev sigma) q)
```

验证状态实时展示

Verif_reverse.v — vc

reverse.c Verif_reverse.v

Verif_reverse.v > ...

```
68 Lemma body_reverse: semax_body Vprog Gprog f_reverse reverse_spec.
69 Proof.
70 start_function.
71 forward. (* w = NULL; *)
72 forward. (* v = p; *)
73 forward_while
74   (EX s1: list val, EX s2 : list val,
75    EX w: val, EX v: val,
76    PROP (sigma = rev s1 ++ s2)
77    LOCAL (temp_w w; temp_v v)
78    SEP (listrep s1 w; listrep s2 v)).
79 - (* Prove that (current) precondition implies the loop invariant *)
80   Exists (a nil val) sigma nullval p.
81   entailer!.
82   unfold listrep.
83   entailer!.
84 - (* Prove that loop invariant implies typechecking of loop condition *)
85   entailer!.
86 - (* Prove that loop body preserves invariant *)
87   destruct s2 as [ | h r].
88   + (* s2=nil *)
89     unfold listrep at 2.
90     Intros.
91     subst. contradiction.
92   + (* s2=h::r *)
93     unfold listrep at 2; fold listrep.
94     Intros y.
95     forward. (* t = v->tail; *)
96     forward. (* v->tail = w; *)
97     forward. (* w = v; *)
98     forward. (* v = t; *)
99     entailer!.
100    Exists (h::s1,r,v,y).
101    entailer!.
102    * simpl. rewrite app_ass. auto.
```

← 单步符号执行

Proof

Main 1 Shelved 0 Given up 0

Goal 1

```
Espece : OracleKind
sigma : list val
p : val
Delta_specs : Maps.PTree.t funspec
Delta := abbreviate : tycontext
POSTCONDITION := abbreviate : ret_assert
MORE_COMMANDS := abbreviate : statement

(1/1)
semex Delta
(PROP (
  LOCAL (temp_w (Vlong (Int64.repr (Int.signed (Int.repr 0)))); temp_p p
  SEP (listrep sigma p)) (_v = _p;
  MORE_COMMANDS) POSTCONDITION
```

↑ 执行赋值后的符号状态

Messages

vc 0 0 0 Cursor Tab Screen Reader Optimized Ln 71, Col 28 Spaces: 2 UTF-8 LF {} Rocq -> Tab Stats: 0/2 (0%) Agent Stats: 0/0 (0%) -> No commit scored

forward 前推一条 C 基本语句

符号状态的改变，反映这条指令执行程序状态的局部变化。

交互式证明

Verif_reverse.v > ...

```
68 Lemma body_reverse: semax_body Vprog Gprog f_reverse reverse_spec.
69 Proof.
70 start_function.
71 forward. (* w = NULL; *)
72 forward. (* v = p; *)
73 forward_while
74 (EX s1: list val, EX s2 : list val,
75  EX w: val, EX v: val,
76   PROP (sigma = rev s1 ++ s2)
77   LOCAL (temp_w w; temp_v v)
78   SEP (listrep s1 w; listrep s2 v)).
79 (* Prove that (current) precondition implies the loop invariant *)
80 Exists (anil val) sigma nullval p.
81 entailer!.
82 unfold listrep.
83 entailer!.
84 (* Prove that loop invariant implies typechecking of loop condition *)
85 entailer!.
86 (* Prove that loop body preserves invariant *)
87 destruct s2 as [ h r ].
88 + (* s2=nil *)
89   unfold listrep at 2.
90   Intros.
91   subst. contradiction.
92 + (* s2=h::r *)
93   unfold listrep at 2; fold listrep.
94   Intros y.
95   forward. (* t = v->tail; *)
96   forward. (* v->tail = w; *)
97   forward. (* w = v; *)
98   forward. (* v = t; *)
99   entailer!.
100   Exists (h::s1,r,v,y).
101   entailer!.
102 * simpl. rewrite app_ass. auto.
```

← 给出循环不变式

Click on the window and keep Alt pressed in to enable term eliding/expanding.

Proof

Main 1 Shelled 0 Given up 0

Goal 1

Espec : OracleKind
sigma : list val
p : val
Delta_specs : Maps.PTree.t funspec
Delta := abbreviate : tycontext

(1/1)

ENTAIL Delta,
PROP ()
LOCAL (temp_v p; temp_w (Vlong (Int64.repr (Int.signed (Int.repr 0))));
temp_p p) SEP (listrep sigma p)
├ EX (s1 s2 : list val) (w v : val),
PROP (sigma = rev s1 ++ s2)
LOCAL (temp_w w; temp_v v) SEP (listrep s1 w; listrep s2 v)

↑ 状态转换生成的证明义务

Messages

证明义务

循环前的符号状态与循环不变式不一致，产生一条待证的分离逻辑蕴含式。

交互式证明

The screenshot shows the Verif reverse.v editor with a proof script on the left and a proof state on the right.

Proof Script (Left Panel):

```

68 Lemma body_reverse: semax_body Vprog Gprog f_reverse reverse_spec.
69 Proof.
70 start_function.
71 forward. (* w = NULL; *)
72 forward. (* v = p; *)
73 forward_while
74   (EX s1: list val, EX s2 : list val,
75    EX w: val, EX v: val,
76    PROP (sigma = rev s1 ++ s2)
77    LOCAL (temp_w w; temp_v v)
78    SEP (listrep s1 w; listrep s2 v)).
79 - (* Prove that (current) precondition implies the loop invariant *)
80   Exists (anil val) sigma nullval p.
81   entailer!.
82   unfold listrep.
83   entailer!.
84 - (* Prove that loop invariant implies typechecking of loop condition *)
85   entailer!.
86 - (* Prove that loop body preserves invariant *)
87   destruct s2 as [ | h r ].
88   + (* s2=nil *)
89     unfold listrep at 2.
90     Intros.
91     subst. contradiction.
92   + (* s2=h::r *)
93     unfold listrep at 2; fold listrep.
94     Intros y.
95     forward. (* t = v→tail; *)
96     forward. (* v→tail = w; *)
97     forward. (* w = v; *)
98     forward. (* v = t; *)
99     entailer!.
100    Exists (h::s1,r,v,y).
101    entailer!.
102    * simpl. rewrite app_ass. auto.

```

Proof State (Right Panel):

Proof

Main 1 Shelled 0 Given up 0

Goal 1

Espece : OracleKind
sigma : list val
p : val
Delta_specs : Maps.PTree.t funspec
Delta := abbreviate : tycontext

(1/1)

ENTAIL Delta,
PROP ()
LOCAL (temp_w p; temp_w (Vlong (Int64.repr (Int.signed (Int.repr 0))));
temp_p p) SEP (listrep sigma p)
├─ PROP (sigma = rev [] ++ sigma)
LOCAL (temp_w nullval; temp_v p) SEP (listrep [] nullval;
listrep sigma p)

Messages

↑ 调用证明策略

↑ 策略执行后的证明义务

交互式证明

观察证明状态，调用合适的证明策略，逐步推进。

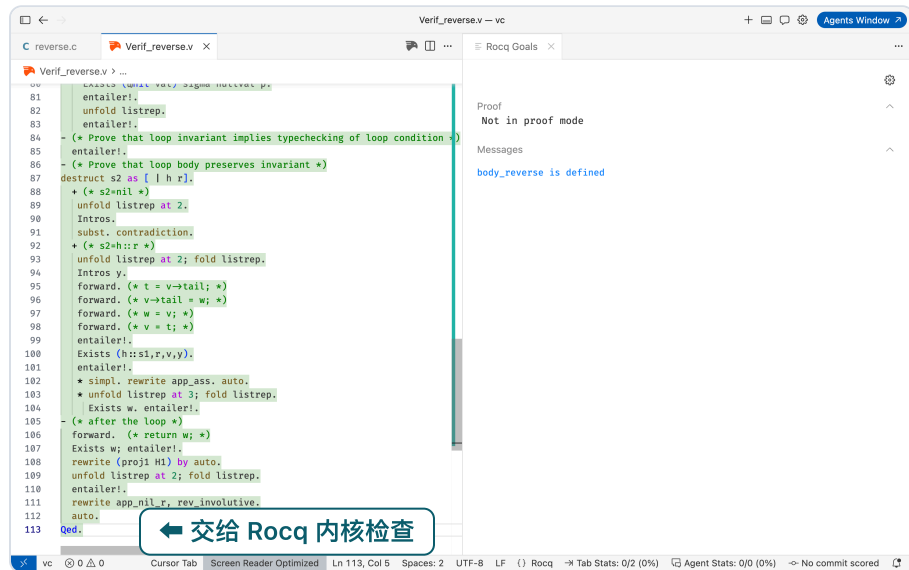
现有范式 · VST/Rocq 演示

可扩展的证明能力

```
68 Lemma body_reverse: semax_body Vprog Gprog f_reverse reverse_spec.
69 Proof.
70 start_function.
71 forward. (* w = NULL; *)
72 forward. (* v = p; *)
73 forward_while
74 (EX s1: list val, EX s2: list val,
75  EX w: val, EX v: val,
76  PROP (sigma = rev s1 ++ s2)
77  LOCAL (temp_w w; temp_v v))
78 SEP (listrep s1 w; listrep s2 v)).
79 - (* Prove that (current) precondition implies the loop invariant *)
80 Exists (snil val)
81 entailer!.
82 unfold listrep.
83 entailer!.
84 - (* Prove that loop invariant implies typechecking of loop condition *)
85 entailer!.
86 - (* Prove that loop body preserves invariant *)
87 destruct s2 as [ l h r ].
88 + (* s2=nil *)
89 unfold listrep at 2.
90 Intros.
91 subst. contradiction.
92 + (* s2=h::r *)
93 unfold listrep at 2; fold listrep.
94 Intros y.
95 forward. (* t = v->tail; *)
96 forward. (* v->tail = w; *)
97 forward. (* w = v; *)
98 forward. (* v = t; *)
99 entailer!.
100 Exists (h::s1,r,v,y).
101 entailer!.
102 * simpl; rewrite app_assoc; auto.
```

```
577 Proof.
578 intros. apply andp_right; auto. apply prop_right; auto.
579 Qed.
580
581 Definition prop_and_same_derives_mprd :=
582   @prop_and_same_derives_mprd _ .
583
584 Ltac entangler :=
585   try match goal with POSTCONDITION => @abbreviate ret_assert _ _ =>
586     clear POSTCONDITION
587   end;
588   try match goal with MORE_COMMANDS => @abbreviate statement _ _ =>
589     clear MORE_COMMANDS
590   end;
591   lazy match goal with
592   | ?T ?P ?_ =>
593     lazy match type of P with
594     | ?T ?_ => try if unify T (environ->mprd)
595       then (clear_up_stackframe; go_lower)
596       else try if unify T mprd
597         then (clear_Delta; pull_out_props)
598         else fail "Unexpected type of entailment, neither mprd nor environ->mprd"
599     end
600   | _ => fail "The entailer tactic works only on entailments _ _ => _"
601   end;
602   try solve [simple apply prop_right; my_auto];
603   try solve [simple apply prop_and_same_derives_mprd; my_auto];
604   saturate_local;
605   entailer!;
606   rewrite <- ?sepcon_assoc.
607
608 Ltac entbang :=
609   intros;
610   lazy match goal with POSTCONDITION => @abbreviate ret_assert _ _ =>
```

总结：交互式程序证明框架



三个不可或缺的元素

1. 验证状态可观测——符号状态与证明目标在边栏实时更新。
2. 证明能力可扩展——自动化求解策略可由用户定制扩展。
3. 验证结论高可信——由可信证明内核检查整个证明项。

关键问题

开发与证明的割裂

开发

程序文本 **reverse.c**
编程语言 **C**
开发环境 **C 集成开发环境**

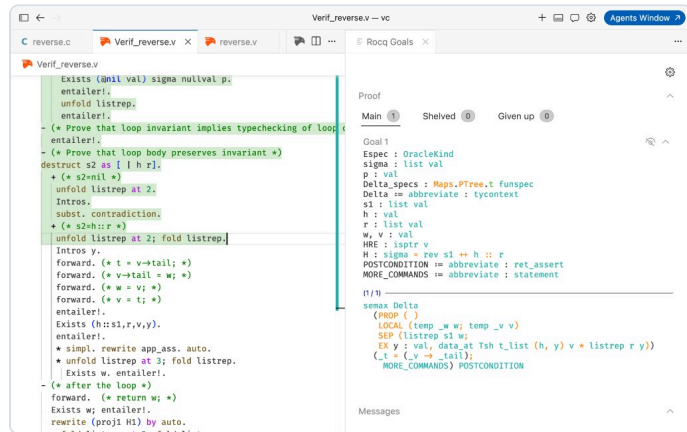
```
struct list *reverse(struct list *p) {  
    struct list *w = NULL, *v = p;  
    while (v) {  
        struct list *t = v->tail;  
        v->tail = w;  
        w = v;  
        v = t;  
    }  
    return w;  
}
```

切换

切换

证明

程序文本 **Verif_reverse.v**
编程语言 **Ltac**
开发环境 **Rocq**



解决方案

C*: 开发与证明的一体化



C* 中的链表反转

```

struct list *reverse(struct list *p)
  PARAM(`l:(int)list`)
  REQUIRE(`sll p l`)
  ENSURE(`sll __return (REVERSE l)`)
{
  struct list *w = NULL, *v = p;
  PROOF fold_sll_null();
  PROOF prove_fact_by(`l == (REVERSE []) ++ l`, simp_conv, ...);
  while (v)
    INV(`exists l1 l2 w_v v_v. fact(l == (REVERSE l1) ++ l2) **
      sll w_v l1 ** sll v_v l2 ** data_at v__addr Tptr v_v **
      data_at w__addr Tptr w_v ** data_at p__addr Tptr p__pre`)
    {
      PROOF unfold_sll_not_null(`v_v`);
      struct list *t = v->tail;
      v->tail = w; w = v; v = t;
      PROOF {
        fold_sll_not_null(`v_v`);
        prove_fact_by(`l == (REVERSE (hd :: l1)) ++ t1`, simp_conv, ...);
      }
    }
  PROOF { /* ... 出口: 展开空链表, 化简出 l1 == REVERSE l ... */ }
  return w;
}

```

C* 中的链表反转

```
struct list *reverse(struct list *p)
  PARAM(`l:({int})list`)
  REQUIRE(`sll p l`)
  ENSURE(`sll __return (REVERSE l)`)
{
  struct list *w = NULL, *v = p;
  PROOF fold_sll_null();
  PROOF prove_fact_by(`l == (REVERSE []) ++ l`, simp_conv, ...);
  while (v)
    INV(`exists l1 l2 w_v v_v. fact(l == (REVERSE l1) ++ l2) **
      sll w_v l1 ** sll v_v l2 ** data_at v__addr Tptr v_v **
      data_at w__addr Tptr w_v ** data_at p__addr Tptr p__pre`)
    {
      PROOF unfold_sll_not_null(`v_v`);
      struct list *t = v->tail;
      v->tail = w; w = v; v = t;
      PROOF {
        fold_sll_not_null(`v_v`);
        prove_fact_by(`l == (REVERSE (hd :: l1)) ++ t1`, simp_conv, ...);
      }
    }
  PROOF { /* ... 出口: 展开空链表, 化简出 l1 == REVERSE l ... */ }
  return w;
}
```

实现

与原来一样的 C 实现代码

C* 中的链表反转

```
struct list *reverse(struct list *p)
  PARAM(`l:(int)list`)
  REQUIRE(`sll p l`)
  ENSURE(`sll __return (REVERSE l)`)
{
  struct list *w = NULL, *v = p;
  PROOF fold_sll_null();
  PROOF prove_fact_by(`l == (REVERSE []) ++ l`, simp_conv, ...);
  while (v)
    INV(`exists l1 l2 w_v v_v. fact(l == (REVERSE l1) ++ l2) **
      sll w_v l1 ** sll v_v l2 ** data_at v__addr Tptr v_v **
      data_at w__addr Tptr w_v ** data_at p__addr Tptr p__pre`)
    {
      PROOF unfold_sll_not_null(`v_v`);
      struct list *t = v->tail;
      v->tail = w; w = v; v = t;
      PROOF {
        fold_sll_not_null(`v_v`);
        prove_fact_by(`l == (REVERSE (hd :: l1)) ++ t1`, simp_conv, ...);
      }
    }
  PROOF { /* ... 出口: 展开空链表, 化简出 l1 == REVERSE l ... */ }
  return w;
}
```

实现

与原来一样的 C 实现代码

规约

断言写在反引号里, 用高阶逻辑与分离逻辑联结词表达

C* 中的链表反转

```
struct list *reverse(struct list *p)
  PARAM(`l: (int)list`)
  REQUIRE(`sll p l`)
  ENSURE(`sll __return (REVERSE l)`)
{
  struct list *w = NULL, *v = p;
  PROOF fold_sll_null();
  PROOF prove_fact_by(`l == (REVERSE []) ++ l`, simp_conv, ...);
  while (v)
    INV(`exists l1 l2 w_v v_v. fact(l == (REVERSE l1) ++ l2) **
        sll w_v l1 ** sll v_v l2 ** data_at v__addr Tptr v_v **
        data_at w__addr Tptr w_v ** data_at p__addr Tptr p__pre`)
    {
      PROOF unfold_sll_not_null(`v_v`);
      struct list *t = v->tail;
      v->tail = w; w = v; v = t;
      PROOF {
        fold_sll_not_null(`v_v`);
        prove_fact_by(`l == (REVERSE (hd :: l1)) ++ t1`, simp_conv, ...);
      }
    }
  PROOF { /* ... 出口: 展开空链表, 化简出 l1 == REVERSE l ... */ }
  return w;
}
```

实现

与原来一样的 C 实现代码

规约

断言写在反引号里, 用高阶逻辑与分离逻辑联结词表达

证明

PROOF 标注的 C 语句: 计算产生定理值并更新验证状态

展示光标处的验证状态

8_reverse.c — cstar_monorepo-proof-lib-reimpl — Index Modified

C 8_reverse.c M x C 10_reverse_op.c M

cstar_examples > tutorial > C 8_reverse.c > reverse

124 struct list *reverse(struct list *p)

125 PARAM("l:(int)list")

126 REQUIRE("sll p l")

127 ENSURE("sll __return (REVERSE l)")

128 {

129 struct list *w, *v;

130 w = (void *)0;

131 v = p;

132

133 PROOF apply_operation_st(fold_sll_null, term_list_n(0));

134 PROOF assert_fact_st("l = (REVERSE []) ++ l:(int)list", simp_conv,

135 THM_LIST(REVERSE_def, APPEND_def)

136);

137

138 while (v)

139 INV("exists l1 l2 w_v v_v.

140 fact(l = (REVERSE l1) ++ l2) **

141 sll w_v l1 ** sll v_v l2 **

142 data_at v__addr Tptr v_v **

143 data_at w__addr Tptr w_v **

144 data_at p__addr Tptr p_pre

145 ")

146 {

147 struct list *t = v->tail;

148 v->tail = w;

149 w = v;

150 v = t;

151

152 PROOF apply_operation_st(fold_sll_not_null, TERM_LIST("v_v:int"));

153 PROOF assert_fact_st(

154 "l = (REVERSE (hd :: l1)) ++ tl:(int)list",

155 simp_conv,

156 THM_LIST(REVERSE_def, APPEND_def, gsym_rule(APPEND_ASSOC))

157);

158 }

Function : reverse

▼ Symbolic State

Existentials (none) 0

Facts (none) 0

Locals 3

undef_data_at v__addr Tptr

undef_data_at w__addr Tptr

data_at p__addr Tptr p_pre

Heap 1

sll p_pre l

▼ Scope: Global

REVERSE_REVERSE : ⊢ forall l. REVERSE (REVERSE l) = l

APPEND_NIL : ⊢ forall l. l ++ [] = l

APPEND_ASSOC : ⊢ forall l m n. l ++ m ++ n = (l ++ m) ++ n

APPEND_def : ⊢ (forall l. [] ++ l = l) && (forall h t l. h :: t ++ l = h :: (t ++ l))

REVERSE_def : ⊢ REVERSE [] = [] && REVERSE (x :: l) = REVERSE l ++ [x]

sll_def :

⊢ (sll pt [] ⊢ fact (pt = 0i)) &&

(sll pt (h :: t) ⊢

fact (¬(pt = 0i)) **

← 当前程序点

↓ 符号状态面板

↓ 证明状态面板

开发过程中实时同步展示验证状态，无需切换环境。

符号状态

Existentials 存在量词 · Facts 已知事实
Locals 局部变量 · Heap 堆资源

证明状态

可使用的定理、待证义务等

写下实现语句，更新验证状态

8_reverse.c — cstar_monorepo-proof-lib-reimpl — Index Modified

C 8_reverse.c M x

C 10_reverse_op.c M

Symbolic State (Line 130) x

Agents Window

cstar_examples > tutorial > C 8_reverse.c > reverse

```

124 struct list *reverse(struct list *p)
125   PARAM("l:(int)list")
126   REQUIRE("sll p l")
127   ENSURE("sll __return (REVERSE l)")
128 {
129   struct list *w, *v;
130   w = (void *)0;
131   v = p;
132
133   PROOF apply_operation_st(fold_sll_null, term_list_n(0));
134   PROOF assert_fact_st("l = (REVERSE []) ++ l:(int)list", simp_conv,
135     THM_LIST(REVERSE_def, APPEND_def)
136   );
137
138   while (v)
139     INV("exists l1 l2 w_v v_v.
140       fact(l = (REVERSE l1) ++ l2) **
141       sll w_v l1 ** sll v_v l2 **
142       data_at v__addr Tptr v_v **
143       data_at w__addr Tptr w_v **
144       data_at p__addr Tptr p_pre
145     ")
146   {
147     struct list *t = v->tail;
148     v->tail = w;
149     w = v;
150     v = t;
151
152     PROOF apply_operation_st(fold_sll_not_null, TERM_LIST("v_v:int"));
153     PROOF assert_fact_st(
154       "l = (REVERSE (hd :: l1)) ++ tl:(int)list",
155       simp_conv,
156       THM_LIST(REVERSE_def, APPEND_def, gsym_rule(APPEND_ASSOC))
157     );
158   }

```

← 当前程序点

Function : reverse

▼ Symbolic State

Previous line 129 · branch 1	Current line 130 · branch 1
Existentials (none)	Existentials (none)
Facts (none)	Facts (none)
Locals undef_data_at v__addr Tptr undef_data_at w__addr Tptr data_at p__addr Tptr p_pre	Locals undef_data_at v__addr Tptr data_at w__addr Tptr 0i data_at p__addr Tptr p_pre
Heap sll p_pre l	Heap sll p_pre l

▼ Scope: Global

```

REVERSE_REVERSE : ⊢ forall l. REVERSE (REVERSE l) = l
APPEND_NIL : ⊢ forall l. l ++ [] = l
APPEND_ASSOC : ⊢ forall l m n. l ++ m ++ n = (l ++ m) ++ n
APPEND_def : ⊢ (forall l. [] ++ l = l) && (forall h t l. h :: t ++ l = h :: (t ++ l))
REVERSE_def : ⊢ REVERSE [] = [] && REVERSE (x :: l) = REVERSE l ++ [x]
sll_def :
  ⊢ (sll ot l1 ⊢ fact (ot == 0i)) &&

```

程序实现指令自动推进符号状态。

面板展示前后对比

左半是执行前的状态，右半是执行后的状态。

符号状态与语句要求不一致时就地报错

The screenshot displays the CStar IDE interface. The left pane shows the source code for `reverse` in `8_reverse.c`. The code defines a `list` structure and a `reverse` function that iteratively reverses a linked list. The right pane shows the **Symbolic State** for the function `reverse`.

Symbolic State:

- Existentials:** `v_v w_v l1 l2` (4)
- Facts:**
 - `fact ~(v_v == 0i)` (2)
 - `fact (l == REVERSE l1 ++ l2)`
- Locals:**
 - `data_at v__addr Tptr v_v` (3)
 - `data_at w__addr Tptr w_v`
 - `data_at p__addr Tptr p_pre`
- Heap:**
 - `sll w_v l1` (2)
 - `sll v_v l2`

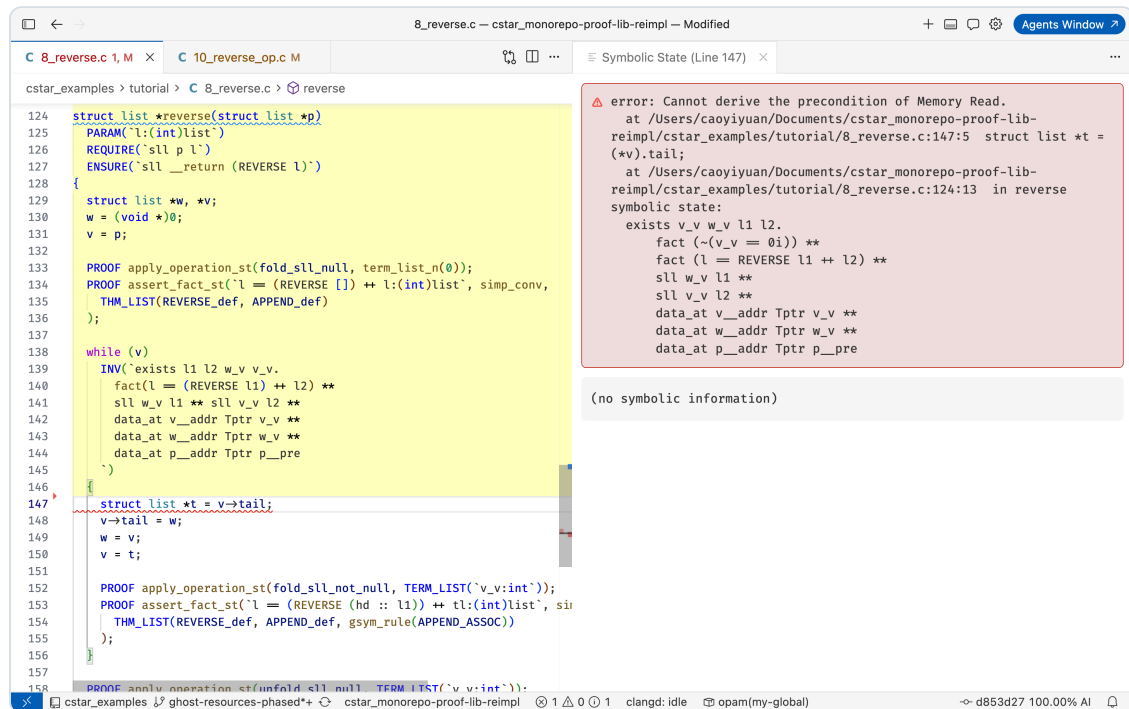
Scope: Global

- `REVERSE_REVERSE :  $\vdash \text{forall } l. \text{ REVERSE } (\text{REVERSE } l) = l$`
- `APPEND_NIL :  $\vdash \text{forall } l. l ++ [] = l$`
- `APPEND_ASSOC :  $\vdash \text{forall } l\ m\ n. l ++ m ++ n = (l ++ m) ++ n$`
- `APPEND_def :  $\vdash (\text{forall } l. [] ++ l = l) \ \&\& \ (\text{forall } h\ t\ l. h :: t ++ l = h :: (t ++ l))$`
- `REVERSE_def :  $\vdash \text{REVERSE } [] = [] \ \&\& \ \text{REVERSE } (x :: l) = \text{REVERSE } l ++ [x]$`
- `sll_def :`

循环体入口的符号状态

循环不变式加上循环条件带来的路径条件。

符号状态与语句要求不一致时就地报错



```
124 struct list *reverse(struct list *p)
125   PARAM("l:(int)list")
126   REQUIRE("sll p l")
127   ENSURE("sll __return (REVERSE l)")
128 {
129   struct list *w, *v;
130   w = (void *)0;
131   v = p;
132
133   PROOF apply_operation_st(fold_sll_null, term_list_n(0));
134   PROOF assert_fact_st("l = (REVERSE []) ++ l:(int)list", simp_conv,
135     THM_LIST(REVERSE_def, APPEND_def)
136 );
137
138   while (v)
139   INV("exists l1 l2 w_v v_v.
140     fact(l = (REVERSE l1) ++ l2) **
141     sll w_v l1 ** sll v_v l2 **
142     data_at v__addr Tptr v_v **
143     data_at w__addr Tptr w_v **
144     data_at p__addr Tptr p_pre
145   ")
146
147   struct list *t = v->tail;
148   v->tail = w;
149   w = v;
150   v = t;
151
152   PROOF apply_operation_st(fold_sll_not_null, TERM_LIST("v_v:int"));
153   PROOF assert_fact_st("l = (REVERSE (hd :: li)) ++ tl:(int)list", si
154     THM_LIST(REVERSE_def, APPEND_def, gsym_rule(APPEND_ASSOC))
155 );
156
157   PROOF apply_operation_st(unfold_sll_null, TERM_LIST("v_v:int"));
```

error: Cannot derive the precondition of Memory Read.
at /Users/caoyiyuan/Documents/cstar_monorepo-proof-lib-reimpl/cstar_examples/tutorial/8_reverse.c:147:5 struct list *t = (v->tail;
at /Users/caoyiyuan/Documents/cstar_monorepo-proof-lib-reimpl/cstar_examples/tutorial/8_reverse.c:124:13 in reverse
symbolic state:
exists v_v w_v l1 l2.
fact (~(v_v = 0i)) **
fact (l = REVERSE l1 ++ l2) **
sll w_v l1 **
sll v_v l2 **
data_at v__addr Tptr v_v **
data_at w__addr Tptr w_v **
data_at p__addr Tptr p_pre

(no symbolic information)

持有的是整条链表的所有权 `sll v_v l2`，而读字段需要的是字段所有权 `data_at`。

通过证明变换符号状态的内存视图

消耗 `sll v_v 12`，分离出头结点的两个字段所有权
`data_at` 与剩余段 `sll q t1`。

消耗 `sll v_v 12`，分离出头结点的两个字段所有权
`data_at` 与剩余段 `sll q t1`。

通过证明转换符号状态的内存视图后继续

8_reverse.c -- cstar_monorepo-proof-lib-reimpl -- 1 problem in this file · Modified

C_8_reverse.c M ×C_10_reverse_op.c M

Symbolic State (Line 151) ×

Agents Window

cstar_examples > tutorial > C_8_reverse.c > reverse

124 struct list *reverse(struct list *p)

125 PARAM('l:(int)list')

126 REQUIRE('sll p l')

127 ENSURE('sll __return (REVERSE l)')

128 {

129 struct list *w, *v;

130 w = (void *)0;

131 v = p;

132

133 PROOF apply_operation_st(fold_sll_null, term_list_n(0));

134 PROOF assert_fact_st('l = (REVERSE []) ++ l:(int)list', simp_conv,

135 THM_LIST(REVERSE_def, APPEND_def)

136);

137

138 while (v)

139 INV('exists l1 l2 w_v v_v.

140 fact(l = (REVERSE l1) ++ l2) **

141 sll w_v l1 ** sll v_v l2 **

142 data_at v__addr Tptr v_v **

143 data_at w__addr Tptr w_v **

144 data_at p__addr Tptr p_pre

145 `')

146

147 PROOF apply_operation_named_st(

148 unfold_sll_not_null,

149 TERM_LIST('v_v:int'),

150 TERM_LIST('hd:int', 'tl:(int)list', 'q:int'));

151 struct list *t = v->tail;

152 v->tail = w;

153 w = v;

154 v = t;

155

156 PROOF apply_operation_st(fold_sll_not_null, TERM_LIST('v_v:int'));

157 PROOF assert_fact_st('l = (REVERSE (hd :: l1)) ++ tl:(int)list', si

158 THM_LIST(REVERSE_def, APPEND_def, decm_rule(APPEND_ASSOC))

Function : reverse

▼ Symbolic State

Consumed Produced

Previous line 150 · branch 1		Current line 151 · branch 1	
Existentials	7	Existentials	7
q hd tl w_v l1 l2 v_v		q hd tl w_v l1 l2 v_v	
Facts	3	Facts	3
fact ~(v_v = 0i))		fact ~(v_v = 0i))	
fact (l = REVERSE l1 ++ l2)		fact (l = REVERSE l1 ++ l2)	
fact (l2 = hd :: tl)		fact (l2 = hd :: tl)	
Locals	3	Locals	4
data_at v__addr Tptr v_v		data_at t__addr Tptr q	
data_at w__addr Tptr w_v		data_at v__addr Tptr v_v	
data_at p__addr Tptr p_pre		data_at w__addr Tptr w_v	
		data_at p__addr Tptr p_pre	
Heap	4	Heap	4
sll w_v l1		sll w_v l1	
data_at (field_addr v_v Tlist		data_at (field_addr v_v Tlist	
Fhead) Tint hd		Fhead) Tint hd	
data_at (field_addr v_v Tlist		data_at (field_addr v_v Tlist	
Ftail) Tptr q		Ftail) Tptr q	
sll q tl		sll q tl	

▼ Scope: Global

REVERSE_REVERSE : ⊢ forall l. REVERSE (REVERSE l) = l

cstar_examplesghost-resources-phased*cstar_monorepo-proof-lib-reimplclangd: idleopam(my-global)

d853d27 100.00% AI

读字段的前置条件可以推出，开发继续

重新建立循环不变式

8_reverse.c — cstar_monorepo-proof-lib-reimpl — Modified

Symbolic State (Line 157)

```

128 {
129   //
130   while (v)
131     INV( exists l1 l2 w_v v_v.
132         fact(l == (REVERSE l1) ++ l2) **
133         sll w_v l1 ** sll v_v l2 **
134         data_at v_addr Tptr v_v **
135         data_at w_addr Tptr w_v **
136         data_at p_addr Tptr p_pre
137       );
138   PROOF apply_operation_named_st(
139     unfold_sll_not_null,
140     TERM_LIST('v_v:int'),
141     TERM_LIST('hd:int', 'tl:(int)list', 'q:int'));
142   struct list *t = v->tail;
143   v->tail = w;
144   w = v;
145   v = t;
146   PROOF apply_operation_st(fold_sll_not_null, TERM_LIST('v_v:int'));
147   PROOF assert_fact_st(
148     'l == (REVERSE (hd :: l1)) ++ tl:(int)list',
149     simp_conv,
150     THM_LIST(REVERSE_def, APPEND_def, gsym_rule(APPEND_ASSOC))
151   );
152   PROOF apply_operation_st(unfold_sll_not_null, TERM_LIST('v_v:int'));
153   PROOF assert_fact_st('l1 == (REVERSE l):(int)list', simp_conv,
154     THM_LIST(APPEND_NIL, REVERSE_REVERSE)
155   );
156   PROOF rewrite_st(THM_LIST(assume_rule('l1 == (REVERSE l):(int)list')));
157 }

```

Function: reverse

Previous line 155 · branch 1		Current line 157 · branch 1	
Existentials	7	Existentials	6
q hd tl w_v l1 l2 v_v		q hd tl l1 l2 v_v	
Facts	3	Facts	3
fact (-(v_v == 0i))		fact (-(v_v == 0i))	
fact (l == REVERSE l1 ++ l2)		fact (l == REVERSE l1 ++ l2)	
fact (l2 == hd :: tl)		fact (l2 == hd :: tl)	
Locals	4	Locals	4
data_at t_addr Tptr q		data_at t_addr Tptr q	
data_at v_addr Tptr q		data_at v_addr Tptr q	
data_at w_addr Tptr v_v		data_at w_addr Tptr v_v	
data_at p_addr Tptr p_pre		data_at p_addr Tptr p_pre	
Heap	4	Heap	2
sll w_v l1		sll q tl	
data_at (field_addr v_v Tlist		sll v_v (hd :: l1)	
Fhead) Tint hd			
data_at (field_addr v_v Tlist			
Ptail) Tptr w_v			
sll q tl			

Scope: Global

REVERSE_REVERSE : $\vdash \text{forall } l. \text{REVERSE (REVERSE } l) = l$

左：把新链接的结点合入已反转的链表

8_reverse.c — cstar_monorepo-proof-lib-reimpl — Modified

Symbolic State (Line 162)

```

128 {
129   //
130   while (v)
131     INV( exists l1 l2 w_v v_v.
132         fact(l == (REVERSE l1) ++ l2) **
133         sll w_v l1 ** sll v_v l2 **
134         data_at v_addr Tptr v_v **
135         data_at w_addr Tptr w_v **
136         data_at p_addr Tptr p_pre
137       );
138   PROOF apply_operation_named_st(
139     unfold_sll_not_null,
140     TERM_LIST('v_v:int'),
141     TERM_LIST('hd:int', 'tl:(int)list', 'q:int'));
142   struct list *t = v->tail;
143   v->tail = w;
144   w = v;
145   v = t;
146   PROOF apply_operation_st(fold_sll_not_null, TERM_LIST('v_v:int'));
147   PROOF assert_fact_st(
148     'l == (REVERSE (hd :: l1)) ++ tl:(int)list',
149     simp_conv,
150     THM_LIST(REVERSE_def, APPEND_def, gsym_rule(APPEND_ASSOC))
151   );
152   PROOF apply_operation_st(unfold_sll_not_null, TERM_LIST('v_v:int'));
153   PROOF assert_fact_st('l1 == (REVERSE l):(int)list', simp_conv,
154     THM_LIST(APPEND_NIL, REVERSE_REVERSE)
155   );
156   PROOF rewrite_st(THM_LIST(assume_rule('l1 == (REVERSE l):(int)list')));
157 }

```

Function: reverse

Previous line 157 · branch 1		Current line 162 · branch 1	
Existentials	6	Existentials	6
q hd tl l1 l2 v_v		q hd tl l1 l2 v_v	
Facts	3	Facts	4
fact (-(v_v == 0i))		fact (-(v_v == 0i))	
fact (l == REVERSE l1 ++ l2)		fact (l == REVERSE l1 ++ l2)	
fact (l2 == hd :: tl)		fact (l2 == hd :: tl)	
		fact (l == REVERSE (hd :: l1) ++ tl)	
Locals	4	Locals	4
data_at t_addr Tptr q		data_at t_addr Tptr q	
data_at v_addr Tptr q		data_at v_addr Tptr q	
data_at w_addr Tptr v_v		data_at w_addr Tptr v_v	
data_at p_addr Tptr p_pre		data_at p_addr Tptr p_pre	
Heap	2	Heap	2
sll q tl		sll q tl	
sll v_v (hd :: l1)		sll v_v (hd :: l1)	

Scope: Global

REVERSE_REVERSE : $\vdash \text{forall } l. \text{REVERSE (REVERSE } l) = l$

APPEND_NIL : $\vdash \text{forall } l. l ++ [] = l$

右：补充证明循环不变式所需的逻辑事实

证明代码复用 C 语言与编辑器支持

8_reverse.c — cstar_monorepo-proof-lib-reimpl — Index Modified

Function: reverse

Symbolic State

Previous line 157 · branch 1	Current line 162 · branch 1
Existentials q hd tl l1 l2 v_v	Existentials q hd tl l1 l2 v_v
0i)) RSE l1 ++ l2 :: tl	Facts fact (-(v_v = 0i)) fact (l = REVERSE l1 ++ l2) fact (l2 = hd :: tl) fact (l = REVERSE (hd :: l1) ++ tl)
Tptr q Tptr q Tptr v_v data_at p_addr Tptr p_pre	Locals data_at t_addr Tptr q data_at v_addr Tptr q data_at w_addr Tptr v_v data_at p_addr Tptr p_pre
Heap sll q tl sll v_v (hd :: l1)	Heap sll q tl sll v_v (hd :: l1)

Scope: Global

REVERSE_REVERSE : $\vdash$ forall l. REVERSE (REVERSE l) = l

APPEND_NIL : $\vdash$ forall l. l ++ [] = l

operational.c — cstar_monorepo-proof-lib-reimpl

Function: reverse

Symbolic State

Previous line 157 · branch 1	Current line 162 · branch 1
Existentials q hd tl l1 l2 v_v	Existentials q hd tl l1 l2 v_v
Facts fact (-(v_v = 0i)) fact (l = REVERSE l1 ++ l2) fact (l2 = hd :: tl)	Facts fact (-(v_v = 0i)) fact (l = REVERSE l1 ++ l2) fact (l2 = hd :: tl) fact (l = REVERSE (hd :: l1) ++ tl)
Locals data_at t_addr Tptr q data_at v_addr Tptr q data_at w_addr Tptr v_v data_at p_addr Tptr p_pre	Locals data_at t_addr Tptr q data_at v_addr Tptr q data_at w_addr Tptr v_v data_at p_addr Tptr p_pre
Heap sll q tl sll v_v (hd :: l1)	Heap sll q tl sll v_v (hd :: l1)

Scope: Global

REVERSE_REVERSE : $\vdash$ forall l. REVERSE (REVERSE l) = l

APPEND_NIL : $\vdash$ forall l. l ++ [] = l

证明函数如何改变验证状态

```
PROOF void prove_fact_by(term goal, conv_builder prover, thm_list extra) {  
    term symst = get_symbolic_state();           // 取: 读出当前符号状态 S  
    term_list facts = symst_facts(symst);  
    gnode root = gnode_new(facts, goal);  
    CONV_TAC(root, prover, extra);  
    thm vshift = local_apply(symst, gnode_prove(root)); // 证: 构造视图变换定理 S |-- S'  
    set_symbolic_state(vshift);                  // 交: 以定理为可信凭据更新状态为 S'  
}
```



1. 当前默认的符号执行器是 [QCP \(Wu et al., TASE 2026\)](#)。感谢上海交通大学曹钦翔团队的长期合作与支持!
2. C 中 LCF 风格的可信证明接口实现架构基于我们的前序工作 [Cao et al., TASE 2026](#)。
3. 当前默认的证明内核是 HOL Light。

证明函数如何改变验证状态

```
PROOF void prove_fact_by(term goal, conv_builder prover, thm_list extra) {  
    term symst = get_symbolic_state();           // 取: 读出当前符号状态 S  
    term_list facts = symst_facts(symst);  
    gnode root = gnode_new(facts, goal);  
    CONV_TAC(root, prover, extra);  
    thm vshift = local_apply(symst, gnode_prove(root)); // 证: 构造视图变换定理  $S \dashv\vdash S'$   
    set_symbolic_state(vshift);                  // 交: 以定理为可信凭据更新状态为  $S'$   
}
```



1. 当前默认的符号执行器是 [QCP \(Wu et al., TASE 2026\)](#)。感谢上海交通大学曹钦翔团队的长期合作与支持!
2. C 中 LCF 风格的可信证明接口实现架构基于我们的前序工作 [Cao et al., TASE 2026](#)。
3. 当前默认的证明内核是 HOL Light。

证明函数如何改变验证状态

```
PROOF void prove_fact_by(term goal, conv_builder prover, thm_list extra) {  
    term symst = get_symbolic_state();           // 取: 读出当前符号状态 S  
    term_list facts = symst_facts(symst);  
    gnode root = gnode_new(facts, goal);  
    CONV_TAC(root, prover, extra);  
    thm vshift = local_apply(symst, gnode_prove(root)); // 证: 构造视图变换定理 S |-- S'  
    set_symbolic_state(vshift);                 // 交: 以定理为可信凭据更新状态为 S'  
}
```



1. 当前默认的符号执行器是 [QCP \(Wu et al., TASE 2026\)](#)。感谢上海交通大学曹钦翔团队的长期合作与支持!
2. C 中 LCF 风格的可信证明接口实现架构基于我们的前序工作 [Cao et al., TASE 2026](#)。
3. 当前默认的证明内核是 HOL Light。

证明函数如何改变验证状态

```
PROOF void prove_fact_by(term goal, conv_builder prover, thm_list extra) {  
    term symst = get_symbolic_state();           // 取: 读出当前符号状态 S  
    term_list facts = symst_facts(symst);  
    gnode root = gnode_new(facts, goal);  
    CONV_TAC(root, prover, extra);  
    thm vshift = local_apply(symst, gnode_prove(root)); // 证: 构造视图变换定理  $S \dashv\vdash S'$   
    set_symbolic_state(vshift);                  // 交: 以定理为可信凭据更新状态为  $S'$   
}
```



1. 当前默认的符号执行器是 [QCP \(Wu et al., TASE 2026\)](#)。感谢上海交通大学曹钦翔团队的长期合作与支持!
2. C 中 LCF 风格的可信证明接口实现架构基于我们的前序工作 [Cao et al., TASE 2026](#)。
3. 当前默认的证明内核是 HOL Light。

证明函数如何改变验证状态

```
PROOF void prove_fact_by(term goal, conv_builder prover, thm_list extra) {  
    term symst = get_symbolic_state();           // 取: 读出当前符号状态 S  
    term_list facts = symst_facts(symst);  
    gnode root = gnode_new(facts, goal);  
    CONV_TAC(root, prover, extra);  
    thm vshift = local_apply(symst, gnode_prove(root)); // 证: 构造视图变换定理 S |-- S'  
    set_symbolic_state(vshift);                  // 交: 以定理为可信凭据更新状态为 S'  
}
```



证明函数不在可信计算基内。

1. 当前默认的符号执行器是 QCP (Wu et al., TASE 2026)。感谢上海交通大学曹钦翔团队的长期合作与支持!
2. C 中 LCF 风格的可信证明接口实现架构基于我们的前序工作 Cao et al., TASE 2026。
3. 当前默认的证明内核是 HOL Light。

总结

程序文本的一体化

证明代码就地嵌入在普通 C 实现代码之中

编程语言的一体化

用 C 书写证明，以 C 代码库的形式扩展证明支持

开发环境的一体化

在 C 的编译器与语言服务器之上构建可实时观测验证状态的 IDE

证明库

标准证明库约 6.5 千行（分离逻辑推理规则、目标树与后向证明策略），用户端通用扩展库约 2 千行（操作式推理函数等），以及列表、树、整数算术等领域理论库。

示例

约 40 个经验证程序：链表与数组算法（查找、排序、`memset`）、二叉搜索树、内存分配器（Google pKVM `hyp_allocator`、`buddy_allocator`），以及从 VST、VeriFast、LiveVerif 迁移的案例。

教程

安装指引与渐进式教程：从函数契约与循环不变式，到在 C 中进行定理证明与证明库扩充。

未来工作

- 扩展并发推理支持，目标是验证真实世界的系统软件。
- 定量评估真实任务上 C* 与其他非一体化的程序验证工具的效率差异。

C* 官方网站

cstarlang.org

[安装指引](#) · [示例教程](#) · [证明库文档](#)

一体化对大模型编写可信代码同样有益？

上下文局部性

程序与证明处于同一位置，完成任务无需大量跨文件跳转，上下文更局部。

反馈信号的质量

基于 SMT 求解的工具在验证失败时反馈往往不精确，且执行时间不稳定。C* 对证明代码和普通 C 语句都维护稳定、逐行的符号状态，可以更精确地指导模型修改代码。

安全性由小内核保证

模型生成的证明代码不在可信计算基中，出错的后果是证明不通过，而不是产生伪证。

初步实验

约 150 行的内存分配器 C 实现。使用 Codex 自动完成所有证明开发。等额计算 API 调用开销约 150 美元。

程序与规约取自 [Appel & Naumann. Verified Sequential Malloc/Free. ISMM 2020](#) · 实现代码 [VST/progs/memmgr/malloc.c](#)