

Frame Inference for Inductive Entailment Proofs in Separation Logic

Quang Loc Le (TU) Jun Sun (SUTD) Shengchao Qin (TU)

TACAS 2018, Thessaloniki - Greece

April 16, 2018

Research Problem

Frame Inference:

- Input: Separation logic formulae A, C
- Question: Does exist a formula $?frame$, the "leftover" portions of heap, which makes the following entailment valid?

$$A \models C * ?frame$$

Research Problem

Frame Inference:

- Input: Separation logic formulae A, C
- Question: Does exist a formula $?frame$, the "leftover" portions of heap, which makes the following entailment valid?

$$A \models C * ?frame$$

Challenges:

- Unbounded heaps
- Infinite numerical domain

Research Problem

Frame Inference:

- Input: Separation logic formulae A, C
- Question: Does exist a formula $?frame$, the "leftover" portions of heap, which makes the following entailment valid?

$$A \models C * ?frame$$

Challenges:

- Unbounded heaps
- Infinite numerical domain

Our Contribution

S2ENT, an automatic procedure to infer frame for Inductive Entailment Proofs in Separation Logic.

A fragment of Separation Logic

Formula $\Phi ::= \Delta \mid \Phi_1 \vee \Phi_2$ $\Delta ::= \exists \bar{v}. (\kappa \wedge \pi)$
Spatial formula $\kappa ::= \text{emp} \mid x \mapsto c(v_i) \mid P(\bar{v}) \mid \kappa_1 * \kappa_2$
Pure formula $\pi ::= \pi_1 \wedge \pi_2 \mid \alpha \mid \phi$

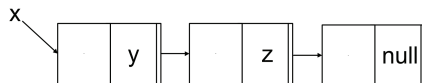
- α : Pointer (Dis)Equalities
- ϕ : Presburger arithmetic
- P : inductive predicate. Predicate Definition: $P(\bar{t}) \equiv \Phi$

A fragment of Separation Logic

Formula	$\Phi ::= \Delta \mid \Phi_1 \vee \Phi_2$	$\Delta ::= \exists \bar{v}. (\kappa \wedge \pi)$
Spatial formula	$\kappa ::= \text{emp} \mid x \mapsto c(v_i) \mid P(\bar{v}) \mid \kappa_1 * \kappa_2$	
Pure formula	$\pi ::= \pi_1 \wedge \pi_2 \mid \alpha \mid \phi$	

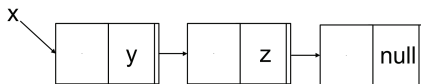
- α : Pointer (Dis)Equalities
- ϕ : Presburger arithmetic
- P : inductive predicate. Predicate Definition: $P(\bar{t}) \equiv \Phi$

Warning: no pointer arithmetic and no magic wand



- Singly-linked list

$$\begin{aligned} \llbracket(\text{root}) \equiv \text{emp} \wedge \text{root} = \text{null} \\ \vee \exists r. \text{root} \mapsto \text{node}(r) * \llbracket(r); \end{aligned}$$

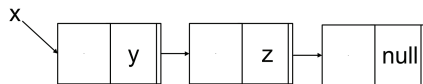


- Singly-linked list

$$\begin{aligned} \llbracket(\text{root}) \equiv \text{emp} \wedge \text{root} = \text{null} \\ \vee \exists r. \text{root} \mapsto \text{node}(r) * \llbracket(r); \end{aligned}$$

- Singly-linked list with **size property**

$$\begin{aligned} \llbracket n(\text{root}, n) \equiv \text{emp} \wedge \text{root} = \text{null} \wedge n = 0 \\ \vee \exists q, n_1. \text{root} \mapsto \text{node}(q) * \llbracket n(q, n_1) \wedge n = n_1 + 1; \end{aligned}$$



- Singly-linked list

$$\begin{aligned} \text{ll}(\text{root}) &\equiv \text{emp} \wedge \text{root} = \text{null} \\ &\vee \exists r. \text{root} \mapsto \text{node}(r) * \text{ll}(r); \end{aligned}$$

- Singly-linked list with **size property**

$$\begin{aligned} \text{lln}(\text{root}, n) &\equiv \text{emp} \wedge \text{root} = \text{null} \wedge n = 0 \\ &\vee \exists q, n_1. \text{root} \mapsto \text{node}(q) * \text{lln}(q, n_1) \wedge n = n_1 + 1; \end{aligned}$$

- Singly-linked list with **last pointer** and size property

$$\begin{aligned} \text{ll_last}(\text{root}, l, n) &\equiv \text{root} \mapsto \text{node}(\text{null}) \wedge l = \text{root} \wedge n = 1 \\ &\vee \exists q, n_1. \text{root} \mapsto \text{node}(q) * \text{ll_last}(q, l, n_1) \wedge n = n_1 + 1; \end{aligned}$$

Verification Problem: Memory Safety and Functional Correctness.

$$\{ \text{lln}(x, n) \wedge n > 0 \} \text{ res} = \text{last}(x) \{ \text{ll_last}(x, \text{res}, n) \wedge n > 0 \}$$

```
1  node append(node x, node y)
2    requires ll_n(x, i) * ll_n(y, j) ∧ i > 0
3    ensures ll_n(res, i + j);
4  {
5    node t = last(x);

6    t->next = y;

7    return x; }
```

Motivation - Hoare-style & Compositional Verification

VC1: $\llbracket \text{lln}(x,i) * \text{lln}(y,j) \wedge i > 0 \rrbracket \models \llbracket \text{lln}(x,i) \wedge i > 0 \rrbracket * ?\text{frame}$

$\{ \llbracket \text{lln}(x,i) \wedge i > 0 \rrbracket \} \quad t = \text{last}(x) \quad \{ \llbracket \text{ll_last}(x,t,i) \wedge i > 0 \rrbracket \}$

```
1  node append(node x, node y)
2    requires  $\llbracket \text{lln}(x,i) * \text{lln}(y,j) \wedge i > 0 \rrbracket$ 
3    ensures  $\llbracket \text{lln}(\text{res}, i+j) \rrbracket$ ;
4    {  $\llbracket \text{lln}(x,i) * \text{lln}(y,j) \wedge i > 0 \rrbracket$ 
5      node t = last(x);
6      t->next = y;
7      return x; }
```

VC1

Motivation - Hoare-style & Compositional Verification

VC1: $\llbracket n(x,i) * \llbracket n(y,j) \wedge i > 0 \models (\llbracket n(x,i) \wedge i > 0) * \llbracket n(y,j)$

$\{ \llbracket n(x,i) \wedge i > 0 \} \quad t = \text{last}(x) \quad \{ \llbracket n_{\text{last}}(x,t,i) \wedge i > 0 \}$

```
1 node append(node x, node y)
2   requires  $\llbracket n(x,i) * \llbracket n(y,j) \wedge i > 0$ 
3   ensures  $\llbracket n(\text{res}, i+j);$ 
4   {  $\llbracket n(x,i) * \llbracket n(y,j) \wedge i > 0$ 
5   node t = last(x);
6   t->next = y;
7   return x; }
```

VC1

Motivation - Compositional Verification

FRAME RULE

$$\frac{\{ \text{lln}(x,i) \wedge i > 0 \} \quad t = \text{last}(x) \quad \{ \text{ll_last}(x,t,i) \wedge i > 0 \}}{\{ \text{lln}(x,i) * \text{lln}(y,j) \wedge i > 0 \} \quad t = \text{last}(x) \quad \{ \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0 \}}$$

```
1  node append(node x, node y)
2    requires lln(x,i)*lln(y,j)∧i>0
3    ensures lln(res,i+j);
4    { //lln(x,i)*lln(y,j)∧i>0
5      node t=last(x);
6      //ll_last(x,t,i)*lln(y,j)∧i>0
7      t->next=y;
8
9      return x; }
```

Motivation - Compositional Verification

VC2:

$$\begin{aligned} & \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0 \\ & \models \exists q. t \mapsto \text{node}(q) * \text{lsegn}(x,t,i-1) * \text{lln}(y,j) \wedge i > 0 \end{aligned}$$

```
1 node append(node x, node y)
2   requires ll_n(x,i)*ll_n(y,j) ∧ i > 0
3   ensures ll_n(res,i+j);
4   { // ll_n(x,i)*ll_n(y,j) ∧ i > 0
5     node t = last(x);
6     // ll_last(x,t,i)*ll_n(y,j) ∧ i > 0
7     t->next = y;
8     // lsegn(x,t,i-1)*t ↦ node(y)* ll_n(y,j) ∧ i > 0
9     return x; }
```

Motivation - Compositional Verification

VC3:

$$\begin{aligned} & \text{lsegn}(\text{res}, t, i-1) * t \mapsto \text{node}(y) * \text{lln}(y, j) \wedge i > 0 \\ & \models \text{lln}(\text{res}, i+j) * \text{emp} \end{aligned}$$

```
1 node append(node x, node y)
2   requires ll_n(x, i) * ll_n(y, j) ∧ i > 0
3   ensures ll_n(res, i+j);
4   { // ll_n(x, i) * ll_n(y, j) ∧ i > 0
5     node t = last(x);
6     // ll_last(x, t, i) * ll_n(y, j) ∧ i > 0
7     t->next = y;
8     // lsegn(x, t, i-1) * t ↦ node(y) * ll_n(y, j) ∧ i > 0
9     return x; }
10 // lsegn(res, t, i-1) * t ↦ node(y) * ll_n(y, j) ∧ i > 0
```

Motivation - Compositional Verification

$VC1 : \llbracket n(x,i) * \llbracket n(y,j) \wedge i > 0 \rrbracket \models \llbracket n(x,i) \wedge i > 0 \rrbracket * \llbracket n(y,j) \rrbracket$

$VC2 : \llbracket \text{ll_last}(x,t,i) * \llbracket n(y,j) \wedge i > 0 \rrbracket$

$\models \exists q. t \mapsto \text{node}(q) * \llbracket \text{lsegn}(x,t,i-1) * \llbracket n(y,j) \rrbracket$

$VC3 : \llbracket \text{lsegn}(res,t,i-1) * t \mapsto \text{node}(y) * \llbracket n(y,j) \wedge i > 0 \rrbracket$

$\models \llbracket n(res,i+j) * \text{emp} \rrbracket$

$\{ \llbracket n(x,n) \wedge n > 0 \rrbracket \} \text{ res} = \text{last}(x) \{ \llbracket \text{ll_last}(x,res,n) \wedge n > 0 \rrbracket \}$

```
1 node append(node x, node y)
2   requires  $\llbracket n(x,i) * \llbracket n(y,j) \wedge i > 0 \rrbracket$ 
3   ensures  $\llbracket n(res,i+j) \rrbracket$ ;
```

```
4 {  $\llbracket n(x,i) * \llbracket n(y,j) \wedge i > 0 \rrbracket$  VC1
```

```
5   node t = last(x);
```

```
    $\llbracket \text{ll\_last}(x,t,i) * \llbracket n(y,j) \wedge i > 0 \rrbracket$  VC2
```

```
6   t->next = y;
```

```
    $\llbracket \text{lsegn}(x,t,i-1) * t \mapsto \text{node}(y) * \llbracket n(y,j) \wedge i > 0 \rrbracket$ 
```

```
7   return x; }
```

```
    $\llbracket \text{lsegn}(res,t,i-1) * t \mapsto \text{node}(y) * \llbracket n(y,j) \wedge i > 0 \rrbracket$  VC3
```


$VC1 : \text{lln}(x,i) * \text{lln}(y,j) \wedge i > 0 \models \text{lln}(x,i) \wedge i > 0 * \text{lln}(y,j)$

$VC2 : \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0$
 $\models \exists q. t \mapsto \text{node}(q) * \text{lsegn}(x,t,i-1) * \text{lln}(y,j)$

$VC3 : \text{lsegn}(\text{res},t,i-1) * t \mapsto \text{node}(y) * \text{lln}(y,j) \wedge i > 0$
 $\models \text{lln}(\text{res},i+j) * \text{emp}$

How to deal with **infinite** domains?

Three Key Ideas:

- 1 Frame Inference: shape and then pure properties
 - Unknown predicates as place-holders

$$E_0 : \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0 \\ \models \exists q. t \mapsto \text{node}(q) * U(x,t,y,q)$$

- Generate relational constraints \mathcal{R} such that

E_0 is valid if \mathcal{R} is satisfied.

Three Key Ideas:

- 1 Frame Inference: shape and then pure properties
 - Unknown predicates as place-holders

$$E_0 : \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0 \\ \models \exists q. t \mapsto \text{node}(q) * U(x,t,y,q)$$

- Generate relational constraints \mathcal{R} such that

E_0 is valid if \mathcal{R} is satisfied.

- 2 New cyclic proof system with Cyclic Cut rule

Three Key Ideas:

- 1 Frame Inference: shape and then pure properties
 - Unknown predicates as place-holders

$$E_0 : \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0 \\ \models \exists q. t \mapsto \text{node}(q) * U(x,t,y,q)$$

- Generate relational constraints \mathcal{R} such that

E_0 is valid if \mathcal{R} is satisfied.

- 2 New cyclic proof system with Cyclic Cut rule
- 3 Predicate Normalization using Lemma Synthesis

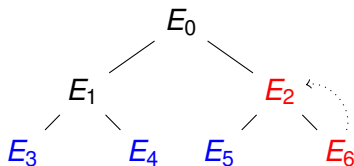
Proving Property P by induction on natural number n

- Base case: prove that P holds when $n = 0$
- Inductive case:
 - induction hypothesis (IH):
 P holds when $n = k$
 - prove that P holds when $n = k+1$

Proving Property P by induction on natural number n

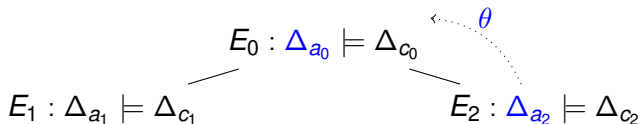
- Base case: prove that P holds when $n = 0$
- Inductive case:
 - induction hypothesis (IH):
 P holds when $n = k$
 - prove that P holds when $n = k+1$

Cyclic Proof (J. Brotherston - UCL)

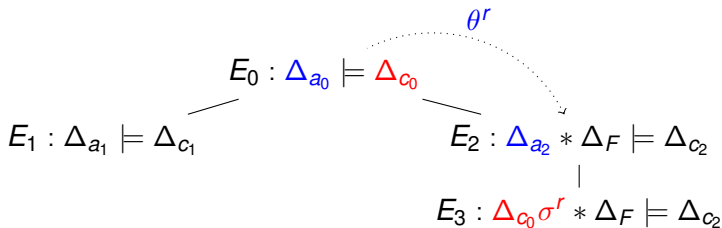


Automatically detect IH E_2

- Weaken E_6 to E'_6
- Find θ s.t. $E'_6\theta \equiv E_2$
- Global soundness condition



(a) `CyclicSL`- J. Brotherston: Full back-link



(b) Our Proposal: Partial back-link

Proposed Cyclic Cut Rule

$$\text{CUT} \frac{A \models B \quad B \models C}{A \models C}$$

$$\text{CUT} \frac{\Delta_{a_0} \theta^r * \Delta_{a_F} \models \Delta_{c_0} \theta^r * \Delta_{a_F} \quad E_3 : \Delta_{c_0} \theta^r * \Delta_{a_F} \models \Delta_{c_2}}{E_2 : \Delta_{a_2} * \Delta_{a_F} \models \Delta_{c_2}}$$

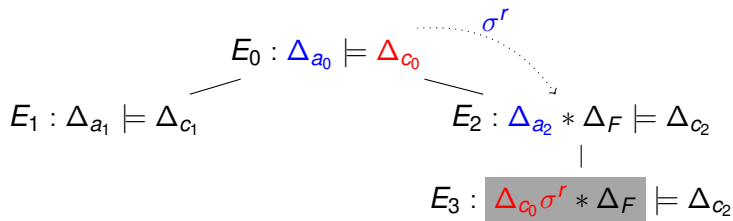


Figure : Our Proposal: Partial back-link for Cut Inference.

Proposed Cyclic Cut Rule

$$\text{Conseq} \frac{\Delta_{a_2} \equiv \Delta_{a_0} \theta^r}{\text{* -Intro} \frac{\text{Subst} \frac{IH : \Delta_{a_0} \models \Delta_{c_0}}{\Delta_{a_0} \theta^r \models \Delta_{c_0} \theta^r}}{\Delta_{a_2} \models \Delta_{a_0} \theta^r}}{\Delta_{a_2} * \Delta_F \models \Delta_{a_0} \theta^r * \Delta_F}}$$

$$\text{CUT} \frac{\Delta_{a_0} \theta^r * \Delta_{a_F} \models \Delta_{c_0} \theta^r * \Delta_{a_F} \quad E_3 : \Delta_{c_0} \theta^r * \Delta_{a_F} \models \Delta_{c_2}}{E_2 : \Delta_{a_2} * \Delta_{a_F} \models \Delta_{c_2}}$$

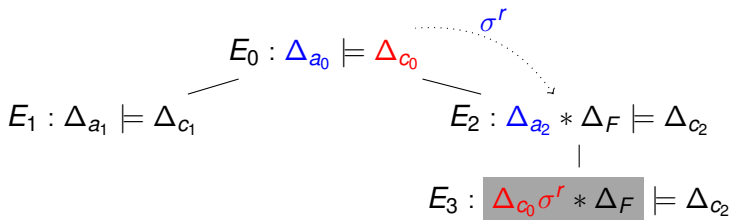
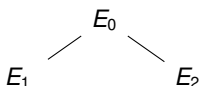


Figure : Our Proposal: Partial back-link for Cut Inference.

$$E_0$$
$$E_0 : \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0 \models \exists q. t \mapsto \text{node}(q) * \mathcal{U}(x,t,q,y)$$

Example - VC2



$$E_0 : \text{ll_last}(x,t,i) * \text{lln}(y,j) \wedge i > 0 \models \exists q. t \mapsto \text{node}(q) * U(x,t,q,y)$$

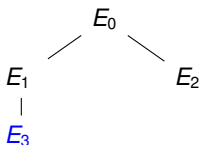
Unfold ll_last in LHS

$$\begin{aligned} \text{ll_last}(\text{root}, \text{lst}, n) &\equiv \text{root} \mapsto \text{node}(\text{null}) \wedge \text{root} = \text{lst} \wedge n = 1 \\ &\vee \exists q. \text{root} \mapsto \text{node}(q) * \text{ll_last}(q, \text{lst}, n-1); \end{aligned}$$

$$\begin{aligned} E_1 : t \mapsto \text{node}(\text{null}) * \text{lln}(y,j) \wedge i > 0 \wedge x = t \wedge i = 1 \\ \models \exists q. t \mapsto \text{node}(v,q) * U(x,t,q,y) \end{aligned}$$

$$\begin{aligned} E_2 : \exists q_1. x \mapsto \text{node}(q_1) * \text{ll_last}(q_1, t, i-1) * \text{lln}(y,j) \wedge i > 0 \\ \models \exists q. t \mapsto \text{node}(v,q) * U(x,t,q,y) \end{aligned}$$

Example - VC2



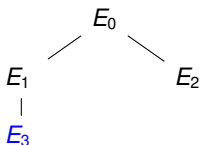
$$E_1 : t \mapsto \text{node}(\text{null}) * \text{lln}(y, j) \wedge i > 0 \wedge x = t \wedge i = 1 \\ \models \exists q. t \mapsto \text{node}(q) * \text{U}(x, t, q, y)$$

Matching: unify $t \mapsto \text{node}(\dots)$ both sides

$$E_3 : \text{lln}(y, j) \wedge i > 0 \wedge x = t \wedge i = 1 \wedge q = \text{null} \\ \models \text{U}(x, t, q, y)$$

$$E_2 : \exists q_1. x \mapsto \text{node}(q_1) * \text{ll_last}(q_1, t, i-1) * \text{lln}(y, j) \wedge i > 0 \\ \models \exists q. t \mapsto \text{node}(q) * \text{U}(x, t, q, y)$$

Example - VC2



$E_3 : \text{emp} \wedge x = t \wedge i = 1 * \text{lln}(y, j) \wedge i > 0 \wedge q = \text{null} \models U(x, t, q, y)$

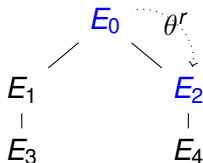
Abduction: synthesize constraint σ_1

$\sigma_1 : \text{lln}(y, -) \wedge t = x \wedge q = \text{null} \Rightarrow U(x, t, q, y)$

$E_2 : \exists q_1. x \mapsto \text{node}(q_1) * \text{ll_last}(q_1, t, i-1) * \text{lln}(y, j) \wedge i > 0$
 $\models \exists q. t \mapsto \text{node}(q) * U(x, t, q, y)$

Example - VC2

$$\sigma_1: \text{lln}(y, -) \wedge t = x \wedge q = \text{null} \\ \Rightarrow U(x, t, q, y)$$



$$E_2: \exists q_1. \mathbf{x} \mapsto \mathbf{node}(q_1) * \text{ll_last}(q_1, t, i-1) * \text{lln}(y, j) \wedge i > 0 \\ \models \exists q. t \mapsto \mathbf{node}(q) * U(x, t, q, y)$$

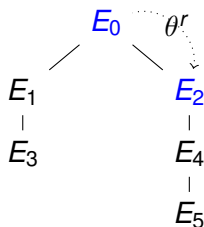
Induction: generate back-link

$$E_0: \text{ll_last}(x, t, i) * \text{lln}(y, j) \wedge i > 0 \models \exists q. t \mapsto \mathbf{node}(q) * U(x, t, q, y) \\ \theta^r \equiv [q_1/x; i-1/i]$$

$$E_4: \exists q_1. \exists q. t \mapsto \mathbf{node}(q) * U(q_1, t, q, y) * \mathbf{x} \mapsto \mathbf{node}(q_1) \wedge i-1 > 0 \\ \models \exists q. t \mapsto \mathbf{node}(q) * U(x, t, q, y)$$

Example - VC2

$$\sigma_1: \llbracket n(y, -) \wedge t = x \wedge q = \text{null} \rrbracket \\ \Rightarrow U(x, t, q, y)$$



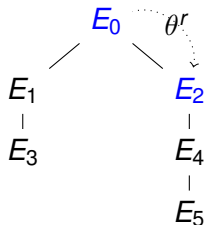
$$E_4 : \exists q_1. \exists q. t \mapsto \text{node}(q) * U(q_1, t, q, y) * x \mapsto \text{node}(q_1) \wedge i - 1 > 0 \\ \models \exists q. t \mapsto \text{node}(q) * U(x, t, q, y)$$

Matching: Unify $t \mapsto \text{node}(\cdot)$ in both sides

$$E_5 : \exists q_1. U(q_1, t, q, y) * x \mapsto \text{node}(q_1) \wedge i - 1 > 0 \\ \models U(x, t, q, y)$$

Example - VC2

$$\sigma_1: \text{lln}(y, -) \wedge t = x \wedge q = \text{null} \\ \Rightarrow \text{U}(x, t, q, y)$$



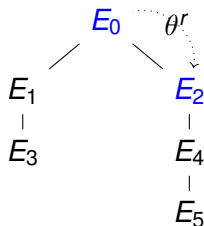
$$E_5 : \exists q_1. \text{U}(q_1, t, q, y) * x \mapsto \text{node}(q_1) \wedge i - 1 > 0 \models \text{U}(x, t, q, y)$$

Abduction: synthesize constraint σ_2

$$\sigma_2: x \mapsto \text{node}(x) * \text{U}(x, t, q, y) \Rightarrow \text{U}(x, t, q, y)$$

Example - VC2

$$\begin{aligned}\sigma_1: & \text{lln}(y, -) \wedge t = x \wedge q = \text{null} \\ & \Rightarrow U(x, t, q, y) \\ \sigma_2: & x \mapsto \text{node}(x) * U(x, t, q, y) \\ & \Rightarrow U(x, t, q, y)\end{aligned}$$



E_0 is valid if $\sigma_1 \wedge \sigma_2$ is satisfied.

The satisfiability problem is solved by S2SAT (Le et. al - CAV16,17)

Example - VC2

$$\sigma_1: \text{lln}(y, -) \wedge t = x \wedge q = \text{null} \Rightarrow U(x, t, q, y)$$

$$\sigma_2: x \mapsto \text{node}(x) * U(x, t, q, y) \Rightarrow U(x, t, q, y)$$

1 Predicate Normalization with Lemma Synthesis

$$U(x, t, q, y) \leftrightarrow \text{lseg}(x, t) * \text{lln}(y, -) \wedge q = \text{null}$$

VC2:

$$\begin{aligned} & \text{ll_last}(x, t, i) * \text{lln}(y, j) \wedge i > 0 \\ & \models t \mapsto \text{node}(\text{null}) * \text{lseg}(x, t) * \text{lln}(y, -) \end{aligned}$$

Example - VC2

$$\sigma_1: \text{lln}(y, -) \wedge t = x \wedge q = \text{null} \Rightarrow U(x, t, q, y)$$

$$\sigma_2: x \mapsto \text{node}(x) * U(x, t, q, y) \Rightarrow U(x, t, q, y)$$

1 Predicate Normalization with Lemma Synthesis

$$U(x, t, q, y) \leftrightarrow \text{lseg}(x, t) * \text{lln}(y, -) \wedge q = \text{null}$$

VC2:

$$\begin{aligned} & \text{ll_last}(x, t, i) * \text{lln}(y, j) \wedge i > 0 \\ & \models t \mapsto \text{node}(\text{null}) * \text{lseg}(x, t) * \text{lln}(y, -) \end{aligned}$$

2 Extend size property

$$\begin{aligned} & \text{ll_last}(x, t, i) * \text{lln}(y, j) \wedge i > 0 \\ & \models t \mapsto \text{node}(\text{null}) * \text{lsegn}(x, t, m) * \text{lln}(y, n) \wedge p(i, j, m, n) \end{aligned}$$

Example - VC2

$$\sigma_1: \text{lln}(y, -) \wedge t = x \wedge q = \text{null} \Rightarrow U(x, t, q, y)$$

$$\sigma_2: x \mapsto \text{node}(x) * U(x, t, q, y) \Rightarrow U(x, t, q, y)$$

1 Predicate Normalization with Lemma Synthesis

$$U(x, t, q, y) \leftrightarrow \text{lseg}(x, t) * \text{lln}(y, -) \wedge q = \text{null}$$

VC2:

$$\begin{aligned} & \text{ll_last}(x, t, i) * \text{lln}(y, j) \wedge i > 0 \\ & \models t \mapsto \text{node}(\text{null}) * \text{lseg}(x, t) * \text{lln}(y, -) \end{aligned}$$

2 Extend size property

$$\begin{aligned} & \text{ll_last}(x, t, i) * \text{lln}(y, j) \wedge i > 0 \\ & \models t \mapsto \text{node}(\text{null}) * \text{lsegn}(x, t, m) * \text{lln}(y, n) \wedge p(i, j, m, n) \end{aligned}$$

3 Infer constraints over size property

$$\begin{aligned} & \text{ll_last}(x, t, i) * \text{lln}(y, j) \wedge i > 0 \\ & \models t \mapsto \text{node}(\text{null}) * \text{lsegn}(x, t, i-1) * \text{lln}(y, j) \end{aligned}$$

- Soundness of Cyclic Proofs
- Predicate Normalization and Synthesis over Pure Properties
- Implementation: Based on SLEEK (Chin *et. al.*)
- Evaluation on 36 Frame Inference Problems
 - `CyclicSL`: solves 18 problems (shape only and frame is `emp`).
 - `songbird`: solves 25 problems (frame is `emp`).
 - `S2ENT`: solves 36 problems.
- Integrated into S2
 - Compositional Verification of Heap-Manipulating Programs in C library Glib open source (`glist.c`, `glist.c`, `gtree.c`, `gnode.c`).

Frame Inference for Cyclic Entailment Proofs

- Infer shape and then pure properties
- New cyclic proof system with Cyclic Cut Rule

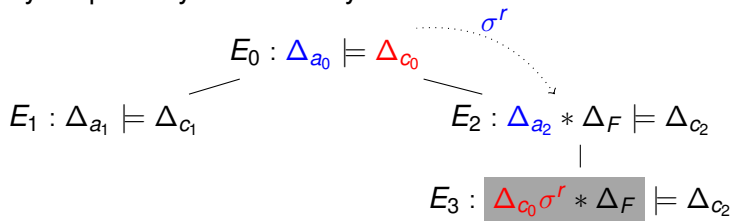


Figure : Our Proposal: Partial back-link for Cut Inference.

- Predicate Normalization using Lemma Synthesis