Automated Inference of Shape Specifications

L Q Loc, Gherghina (Google), Qin (Teesside), W-N Chin

Dept of Computer Science - National University of Singapore

December 16, 2014

Related Work

Shape analysis: discover invariants that describe the data structures in a program

Given pre-shapes, infer post-shapes

- TVLA: T. Reps et. al. [POPL'99].
- Xisa: Rival et. al. [POPL'08].

More Automatic

- Bi-abduction: Calcagno et.al. [POPL'09, J.ACM'11], Predator [CAV'11].
  - infer both pre- and post-shapes
  - bottom-up, verify Linux kernel 2.6.25.4 with 2.473MLOC in 1739.28 seconds $\surd$
- Forestor: Vojnar et.al. [CAV'13]. top-down
- Cycle proof: James et.al. [SAS'14].

Bi-Abduction [POPL'09,J.ACM'11]

Frame Inference

$$\Delta_{\texttt{ante}} \vdash \Delta_{\texttt{conseq}} * \underline{?\Delta_{\texttt{frame}}}$$

Example

$$\texttt{struct nnode}\{\ \texttt{struct nnode}^{*}\ \texttt{next}\}.$$

$$x{\mapsto}nnode(n_1) * y{\mapsto}nnode(n_1) \vdash y{\mapsto}nnode(n_2) * ?\Delta_{\texttt{frame}}$$

$$\Delta_{\texttt{frame}} = x{\mapsto}nnode(n_1) \wedge x{\neq}y \wedge y{\neq}\texttt{NULL} \wedge n_1{=}n_2$$

Bi-Abduction [POPL'09,J.ACM'11]

Abduction

$$\underline{?\Delta_{\texttt{pre}}} * \Delta_{\texttt{ante}} \vDash \Delta_{\texttt{conseq}}$$

Example:

$$?\Delta_{\texttt{pre}} \wedge \texttt{true} \vDash y \mapsto nnode(n_2)$$

$$\Delta_{\texttt{pre}} = y \mapsto nnode(n_2)$$

Do not infer trivial precondition, i.e. `false`

Bi-Abduction [POPL'09,J.ACM'11]

Abduction + Frame Inference = Bi-Abduction

$$\underline{?\Delta_{\texttt{pre}}} * \Delta_{\texttt{ante}} \vDash \Delta_{\texttt{conseq}} * \underline{?\Delta_{\texttt{frame}}}$$

Example:

$$?\Delta_{\texttt{pre}} * x \mapsto nnode(n_1) \wedge x \neq y \vDash y \mapsto nnode(n_2) * ?\Delta_{\texttt{frame}}$$

$$\Delta_{\texttt{pre}} = y \mapsto nnode(n_2) \qquad \Delta_{\texttt{frame}} = x \mapsto nnode(n_1) \wedge x \neq y \wedge y \neq \texttt{NULL}$$

Bi-Abduction [POPL'09,J.ACM'11]

Bi-Abduction (Calcagno et. al. [J.ACM'11])

```
1 void free_list(struct snode *x){
2   struct snode *t;
3   while(x!=0){
4     t=x;
5     x=x->next;
6     free(t);
7   }
8 }
```

UNSOUND!

Aims:

1. scalability $\surd$
2. expressive data structures $\{$lists,.. ? $\}$
3. soundness ?

Second-Order Bi-Abduction

- Unknown Predicates as Second-Order Variables
- $\mathcal{R} \wedge \Delta_{\texttt{ante}} \vDash \Delta_{\texttt{conseq}} * \underline{\Delta_{\texttt{frame}}}$

$\mathcal{R}$ is a set of relational assumptions

$$\mathcal{R} \;=\; \bigwedge_{i=1}^{n} (\Delta_i \circledcirc \Delta_g \Rightarrow \Phi_i)$$

Entailment syntax: $\Delta_{\texttt{ante}} \vdash \Delta_{\texttt{conseq}} \rightsquigarrow (\mathcal{R},\ \underline{\Delta_{\texttt{frame}}})$

Second-Order Bi-Abduction

Examples:

Abductive Unfold

$$H(y)*x\mapsto nnode(n_1) \vdash y\mapsto nnode(n_2) \rightsquigarrow (\mathcal{R},\ \underline{\Delta_{\texttt{frame}}})$$

$$\mathcal{R} \equiv H(y) \Rightarrow y\mapsto nnode(n_2)*U(n_2)$$
$$\Delta_{\texttt{frame}} = x\mapsto nnode(n_1)*\underline{U(n_2)}$$

Abductive Fold

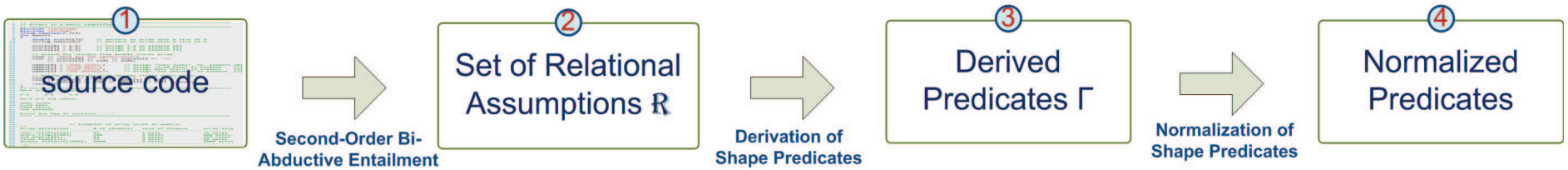
$$x\mapsto nnode(\texttt{NULL})*y\mapsto nnode(\texttt{NULL}) \vdash \texttt{G(x)}$$
$$\rightsquigarrow (x\mapsto nnode(\texttt{NULL})\Rightarrow G(x),\ \texttt{y}\mapsto\texttt{nnode(NULL)})$$

from Verification to Inference

Research Problem: Given a program, find pre-shape and post-shape such that the program is absence of memory-errors (null dereference)?
Solution:

1. Assume pre-shape is P(..), post-shape is Q(..)
2. Transform requirement to relational assumptions on P,Q.
   Denote proof obligations to be met
3. Solve

Our framework:



Shape Inference: get_next Example

Method Specification:

$$\texttt{requires}\ \ \texttt{ptr}{\mapsto}\texttt{node}\langle\texttt{d},\texttt{p}\rangle$$
$$\texttt{ensures}\ \ \ \texttt{ptr}{\mapsto}\texttt{node}\langle\texttt{d},\texttt{p}\rangle{\wedge}\texttt{res}{=}\texttt{p};$$

Verification

```
1  struct node* get_next(struct node* ptr){
   //α1 : ptr↦node⟨d,p⟩
   //(binding)E1 : α1 ⊢ ∃d1,p1 · ptr↦node⟨d1,p1⟩
2    return ptr->next;
   //α2 : ∃d1,p1 · ptr↦node⟨d1,p1⟩∧d1=d∧p1=p∧res=p1
   //(post)E2 : α2 ⊢ ptr↦node⟨d,p⟩∧res=p
3  }
```

Shape Inference: get_next Example

Method Specification:

```
requires H(ptr)    ensures G(ptr,res);
```

Verification to Inference

```
1  struct node* get_next(struct node* ptr){
   //α'1 : H(ptr)
   //(binding)E'1 : α'1 ⊢ ...
2    return ptr->next;
   //α'2 : ...
   //(post)E'2 : α'2 ⊢ G(ptr,res)
```

Shape Inference: get_next Example

Method Specification:

```
requires H(ptr)    ensures G(ptr,res);
```

Verification to Inference

```
1  struct node* get_next(struct node* ptr){
   //α'1 : H(ptr)
   //(binding)E'1 : α'1 ⊢ ∃d1,p1 · ptr↦node⟨d1,p1⟩
2    return ptr->next;
   //α'2 : ...
   //(post)E'2 : α'2 ⊢ G(ptr,res)
3  }
```

Shape Inference: get_next Example

Method Specification:

```
requires H(ptr)    ensures G(ptr,res);
```

Verification to Inference

```
1  struct node* get_next(struct node* ptr){
   //α'1 : H(ptr)
   //(binding)E'1 : α'1 ⊢ ∃d1,p1 · ptr↦node⟨d1,p1⟩
2    return ptr->next;
   //α'2 : ...
   //(post)E'2 : α'2 ⊢ G(ptr,res)
3  }
```

$$E_1' \rightsquigarrow (\mathcal{R}_1, \Delta^1_{frame})$$

$$\mathcal{R}_1 \equiv H(ptr) \Rightarrow ptr \mapsto node(\_, p_1) * U(p_1) \quad \Delta^1_{\texttt{frame}} = ptr \mapsto node(p_1) * U(p_1)$$

Shape Inference: get_next Example

Method Specification:

```
requires H(ptr)     ensures G(ptr,res);
```

Verification to Inference $\mathcal{R}_1 \equiv H(ptr) \Rightarrow ptr \mapsto node(\_, p_1) * U(p_1)$

```
1  struct node*(struct node* ptr){
   //α'1 : H(ptr)
   //(binding)E'1 : α'1 ⊢ ∃d1,p1 · ptr↦node⟨d1,p1⟩
2    return ptr->next;
   //α'2 : ∃d1,p1 · ptr↦node⟨d1,p1⟩*U(p1)∧res=p1
   //(post)E'2 : α'2 ⊢ G(ptr,res)
3  }
```

Shape Inference: get_next Example

Method Specification:

$$\texttt{requires H(ptr)} \quad \texttt{ensures G(ptr,res);}$$

Verification to Inference

```
1  struct node*(struct node* ptr){
   //α'1 : H(ptr)
   //(binding)E'1 : α'1 ⊢ ∃d1,p1 · ptr↦node⟨d1,p1⟩
2    return ptr->next;
   //α'2 : ∃d1,p1 · ptr↦node⟨d1,p1⟩*U(p1)∧res=p1
   //(post)E'2 : α'2 ⊢ G(ptr,res)
3  }
```

$$\texttt{E}_2' \rightsquigarrow (\mathcal{R}_2, \Delta^2_{frame})$$

$$\mathcal{R}_2 \equiv ptr \mapsto node(\_, p_1) * U(p_1) \wedge \texttt{res} = p_1 \Rightarrow G(ptr, \texttt{res}) \quad \Delta^2_{\texttt{frame}} = U(p_1)$$

Shape Inference: get_next Example

Method Specification:

```
requires H(ptr)    ensures G(ptr,res);
```

Relational Assumptions derived:

$$\mathcal{R}_1 \equiv H(ptr) \Rightarrow ptr \mapsto node(\_, p_1) * U(p_1)$$

$$\mathcal{R}_2 \equiv ptr \mapsto node(\_, p_1) * U(p_1) \wedge \texttt{res} = p_1 \Rightarrow G(ptr, \texttt{res})$$

Dangling Predicates, such as $U(p_1)$:

- Uninstantiated predicates
- Can link pre/post specification

Shape Inference: get_next Example

Method Specification:

$$\texttt{requires}\ \texttt{H(ptr)} \qquad \texttt{ensures}\ \texttt{G(ptr,res);}$$

Weakest Pre-Predicate:

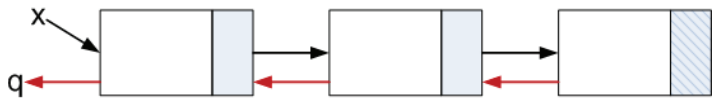
$$H(ptr) \equiv ptr \mapsto node(\_, DP)$$

Strongest Post-Predicate:

$$G(ptr, \texttt{res}) \equiv ptr \mapsto node(\_, DP) \wedge \texttt{res} = DP$$

Shape Inference: sll2dll Example

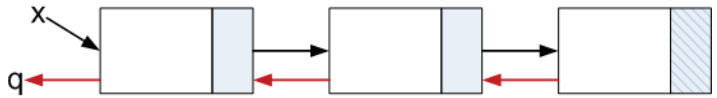
```
1 struct node{struct node* prev;struct node* next;}
2 void node* sll2dll (struct node *x, struct node *q)
        {
3   if(x==NULL) return;
4   else{
5     x->prev=q;
6     sll2dll(x->next,x);
7   }
8 }
```



requires sll(x) ensures dll(x,q)

Shape Inference: sll2dll Example

```
1 struct node{struct node* prev;struct node* next;}
2 void node* sll2dll (struct node *x, struct node *q)
        {
3   if(x==NULL) return;
4   else{
5     x->prev=q;
6     sll2dll(x->next,x);
7   }
8 }
```



Unknown Predicates as Second-Order Variables:

`requires` $H(x, q_{\#})$ `ensures` $G(x, q)$

1. Infer Relational Assumptions via Second-Order Bi-abduction
2. Derive Predicate Definitions
3. Normalize Predicate Definitions

Shape Inference: sll2dll Example. Step 1

```
void sll2dll(struct node* x, struct node* q) {
 (α1) H(x, q#)
 if (x==NULL)
  (α2) H(x, q#)∧x=NULL
  return;
 x->prev=q;
 sll2dll(x->next, x);
}
```

Post Proving at α_2

$$\mathtt{H(x,q\#)} \wedge \mathtt{x=NULL} \vdash G(x,q)$$
$$\leadsto ((\mathtt{A1}) : \ \mathtt{H(x,q\#)} \wedge \mathtt{x=NULL} \Rightarrow \mathtt{G(x,q)},\ \mathtt{emp} \wedge \mathtt{x=NULL})$$

Shape Inference: sll2dll Example. Step 1

```
void sll2dll(struct node* x, struct node* q) {
 (α1) H(x, q#)
 if (x==NULL)
  return;
 (α3) H(x, q#)∧x≠NULL
 x->prev=q;
 sll2dll(x->next,x);
}
```

Field Access after α_3:

$$\alpha_3 \vdash x{\mapsto}node(x_p,x_n) \rightsquigarrow (\mathtt{A2},\ H_p(x_p,q_\#){*}H_n(x_n,q_\#) \wedge \mathtt{x}{\neq}\mathtt{NULL})$$

$$(\mathsf{A2}).\ \ H(x,q_\#) \wedge \mathtt{x}{\neq}\mathtt{NULL} \Rightarrow x{\mapsto}node(x_p,x_n){*}H_p(x_p,q_\#){*}H_n(x_n,q_\#)$$

Shape Inference: sll2dll Example. Step 1

```
void sll2dll(struct node* x, struct node* q) {
 (α1) H(x, q#)
 if (x==NULL)
  return;
 (α3) H(x, q#)∧x≠NULL
 (α'3) x↦node(xp,xn)*Hp(xp,q#)*Hn(xn,q#) ∧x≠NULL
 x->prev=q;
 sll2dll(x->next,x);
}
```

Field Access:

$$\alpha_3 \vdash x{\mapsto}node(x_p,x_n) \rightsquigarrow (\mathtt{A2},\ H_p(x_p,q_\#){*}H_n(x_n,q_\#) \wedge \mathtt{x}{\neq}\mathtt{NULL})$$

$$(\mathsf{A2}).\ \ H(x,q_\#) \wedge \mathtt{x}{\neq}\mathtt{NULL} \Rightarrow x{\mapsto}node(x_p,x_n){*}H_p(x_p,q_\#){*}H_n(x_n,q_\#)$$

Shape Inference: sll2dll Example

```
void sll2dll(struct node* x, struct node* q) {
 (α1) H(x, q#)
 if (x==NULL)
  return;
 (α'3) x↦node(xp,xn)*Hp(xp,q#)*Hn(xn,q#) ∧x≠NULL
 x->prev=q;
 (α4) x↦node(q,xn)*Hp(xp,q#)*Hn(xn,q#) ∧x≠NULL
 sll2dll(x->next,x);
}
```

Pre- Proving for recursive call:

$$(\alpha_4) \vdash H(x_n, x_\#) \rightsquigarrow (\mathtt{A3},\ x{\mapsto}node(q,x_n){*}H_p(x_p,q_\#){\wedge}x{\neq}\mathtt{NULL})$$

$$(\mathsf{A3}).\ \ H_n(x_n, q_\#)\ @\ x{\mapsto}node(q, x_n) \Rightarrow H(x_n, x_\#)$$

Shape Inference: sll2dll Example. Step 1

Relational Assumptions:

(A1). $H(x,q) \wedge \texttt{x=NULL} \Rightarrow G(x,q)$

(A2). $H(x,q) \wedge \texttt{x}\neq\texttt{NULL} \Rightarrow x\mapsto node(x_p,x_n) * H_p(x_p,q) * H_n(x_n,q)$

(A3). $H_n(x_n,q)$ @ $x\mapsto node(q,x_n) \Rightarrow H(x_n,x)$

(A4). $x\mapsto node(q,x_n) * G(x_n,x) \Rightarrow G(x,q)$

After this step, we can confirm that the `sll2dll` is memory-safety

- `requires` `H(x,q)` `ensures` `G(x,q)`, and $\{A1 \wedge A2 \wedge A3 \wedge A4\}$
- However, we wish to go beyond the memory-safety

We aim to infer a understandable and reusable specs, like

`requires sll(x)` `ensures dll(x,q)`

Derivation

Key idea: Split relational assumptions of pre- and post-predicates.

- Pre-predicates: strengthened
- Post-predicates: weakened

Steps

1. Base Split (with Guard)
2. Sort
3. Pre-Predicates Synthesis Rules
4. Post-Predicates Synthesis Rules

Shape Inference: sll2dll Example. Step 2

Step 2.1: Base Split:

(A1). $H(x,q) \wedge \texttt{x=NULL} \Rightarrow G(x,q)$

(A2). $H(x,q) \wedge \texttt{x}\neq\texttt{NULL} \Rightarrow x{\mapsto}node(x_p,x_n){*}H_p(x_p,q){*}H_n(x_n,q)$

(A3). $H_n(x_n,q) \ @ \ x{\mapsto}node(q,x_n) \Rightarrow H(x_n,x)$

(A4). $x{\mapsto}node(q,x_n){*}G(x_n,x) \Rightarrow G(x,q)$

$\Longrightarrow$

(A1a). $H(x,q) \wedge \texttt{x=NULL} \Rightarrow \texttt{emp}\wedge\texttt{true}$

(A1b). $\texttt{emp} \wedge \texttt{x=NULL} \Rightarrow G(x,q)$

(A2). $H(x,q) \wedge \texttt{x}\neq\texttt{NULL} \Rightarrow x{\mapsto}node(x_p,x_n){*}H_p(x_p,q){*}H_n(x_n,q)$

(A3). $H_n(x_n,q) \ @ \ x{\mapsto}node(q,x_n) \Rightarrow H(x_n,x)$

(A4). $x{\mapsto}node(q,x_n){*}G(x_n,x) \Rightarrow G(x,q)$

Shape Inference: sll2dll Example. Step 2

Step 2.2: Sort

(A1a). $H(x,q) \wedge \texttt{x=NULL} \Rightarrow \texttt{emp}\wedge\texttt{true}$
(A1b). $\texttt{emp} \wedge \texttt{x=NULL} \Rightarrow G(x,q)$
(A2). $H(x,q) \wedge \texttt{x}\neq\texttt{NULL} \Rightarrow x\mapsto node(x_p,x_n)*H_p(x_p,q)*H_n(x_n,q)$
(A3). $H_n(x_n,q)\ @\ x\mapsto node(q,x_n) \Rightarrow H(x_n,x)$
(A4). $x\mapsto node(q,x_n)*G(x_n,x) \Rightarrow G(x,q)$

$\Longrightarrow$

Pre-Preds Assumptions

(A3). $H_n(x_n,q)\ @\ x\mapsto node(q,x_n) \Rightarrow H(x_n,x)$
(A1a). $H(x,q) \wedge \texttt{x=NULL} \Rightarrow \texttt{emp}\wedge\texttt{true}$
(A2). $H(x,q) \wedge \texttt{x}\neq\texttt{NULL} \Rightarrow x\mapsto node(x_p,x_n)*H_p(x_p,q)*H_n(x_n,q)$

Post-Preds Assumptions

(A1b). $\texttt{emp} \wedge \texttt{x=NULL} \Rightarrow G(x,q)$
(A4). $x\mapsto node(q,x_n)*G(x_n,x) \Rightarrow G(x,q)$

Shape Inference: sll2dll Example. Step 2

Step 2.3: Pre-Predicates Synthesis

Pre-Preds Assumptions

(A3). $H_n(x_n, q)$ @ $x \mapsto node(q, x_n) \Rightarrow H(x_n, x)$
(A1a). $H(x, q) \wedge \texttt{x=NULL} \Rightarrow \texttt{emp} \wedge \texttt{true}$
(A2). $H(x, q) \wedge \texttt{x} \neq \texttt{NULL} \Rightarrow x \mapsto node(x_p, x_n) * H_p(x_p, q) * H_n(x_n, q)$

$\Longrightarrow$

Synthesize $\texttt{H}_\texttt{n}$ from A3

$$H_n(x_n, q) \equiv H(x_n, x) \text{ @ } x \mapsto node(q, x_n)$$

Pre-Preds Assumptions

(A1a). $H(x, q) \wedge \texttt{x=NULL} \Rightarrow \texttt{emp} \wedge \texttt{true}$
(A2). $H(x, q) \wedge \texttt{x} \neq \texttt{NULL} \Rightarrow x \mapsto node(x_p, x_n) * H_p(x_p, q) * H_n(x_n, q)$

Shape Inference: sll2dll Example. Step 2

Step 2.3: Pre-Predicates Synthesis
Inline $\texttt{H}_\texttt{n}$ into $\texttt{A2}$

Synthesized Predicate Definition:

$$H_n(x_n, q) \equiv H(x_n, x) \,@\, x{\mapsto}node(q, x_n)$$

Pre-Preds Assumptions

(A1a). $H(x, q) \wedge \texttt{x=NULL} \Rightarrow \texttt{emp}\wedge\texttt{true}$
(A2). $H(x, q) \wedge \texttt{x}{\neq}\texttt{NULL} \Rightarrow x{\mapsto}node(x_p, x_n){*}H_p(x_p, q){*}H_n(x_n, q)$

$\Longrightarrow$

Pre-Preds Assumptions

(A1a). $H(x, q) \wedge \texttt{x=NULL} \Rightarrow \texttt{emp}\wedge\texttt{true}$
(A2). $H(x, q) \wedge \texttt{x}{\neq}\texttt{NULL} \Rightarrow x{\mapsto}node(x_p, x_n){*}H_p(x_p, q){*}H(x_n, q)$

Shape Inference: sll2dll Example. Step 2

Step 2.3: Pre-Predicates Synthesis
Synthesize H.

Synthesized Predicate Definition:

$$H_n(x_n, q) \equiv H(x_n, x) \,@\, x{\mapsto}node(q, x_n)$$

Pre-Preds Assumptions

$$H(x, q) \Rightarrow (\mathtt{emp}\wedge \mathtt{x}{=}\mathtt{NULL})\vee (x{\mapsto}node(x_p, x_n){*}H_p(x_p, q){*}H(x_n, q)\wedge \mathtt{x}{\neq}\mathtt{NULL})$$

$\Longrightarrow$

Synthesized Predicate Definition:

$$\begin{array}{ll} H_n(x_n, q) & \equiv H(x_n, x) \,@\, x{\mapsto}node(q, x_n) \\ H(x, q) & \equiv (\mathtt{emp}\wedge \mathtt{x}{=}\mathtt{NULL}) \vee \\ & \quad (x{\mapsto}node(x_p, x_n){*}H_p(x_p, q){*}H(x_n, q)\wedge \mathtt{x}{\neq}\mathtt{NULL}) \end{array}$$

Shape Inference: sll2dll Example. Step 2

Step 2.4: Post-Predicates Synthesis
Synthesize G. soundly weaken conjunction into disjunction
Post-Preds Assumptions

$$\text{(A1b).}\ \ \texttt{emp} \wedge \texttt{x=NULL} \Rightarrow G(x,q)$$
$$\text{(A4).}\ \ x{\mapsto}node(q,x_n){*}G(x_n,x) \Rightarrow G(x,q)$$

$\Longrightarrow$

$$\texttt{emp} \wedge \texttt{x=NULL} \vee x{\mapsto}node(q,x_n){*}G(x_n,x) \Rightarrow G(x,q)$$

$\Longrightarrow$

$$\begin{array}{ll} H_n(x_n,q) & \equiv H(x_n,x)\ @\ x{\mapsto}node(q,x_n) \\ H(x,q) & \equiv (\texttt{emp}{\wedge}\texttt{x=NULL}) \vee \\ & \quad (x{\mapsto}node(x_p,x_n){*}H_p(x_p,q){*}H(x_n,q){\wedge}\texttt{x}{\neq}\texttt{NULL}) \\ G(x,q) & \equiv (\texttt{emp} \wedge \texttt{x=NULL}) \vee (x{\mapsto}node(q,x_n){*}G(x_n,x)) \end{array}$$

Shape Inference: sll2dll Example. Step 2

Synthesized Predicate Definition:

$$
\begin{array}{lll}
H_n(x_n, q) & \equiv & H(x_n, x) \circledast x{\mapsto}node(q, x_n) \\
H(x, q) & \equiv & (\mathtt{emp} \wedge \mathtt{x{=}NULL}) \vee \\
 & & (x{\mapsto}node(x_p, x_n) * H_p(x_p, q) * H(x_n, q) \wedge \mathtt{x{\neq}NULL}) \\
G(x,q) & \equiv & (\mathtt{emp} \wedge \mathtt{x{=}NULL}) \vee (x{\mapsto}node(q,x_n) * G(x_n,x))
\end{array}
$$

Other issues:

1. Base Split with Guard
2. Inline with Guarded Assumptions
3. Soundness

Shape Normalization. Step 3

$$
\begin{array}{ll}
H_n(x_n, q) & \equiv H(x_n, x) \,@\, x{\mapsto}node(q, x_n) \\
H(x, q) & \equiv (\texttt{emp}{\wedge}\texttt{x=NULL}) \vee \\
 & \quad (x{\mapsto}node(x_p, x_n){*}H_p(x_p, q){*}H(x_n, q){\wedge}\texttt{x}{\neq}\texttt{NULL}) \\
G(x,q) & \equiv (\texttt{emp} \wedge \texttt{x=NULL}) \vee (x{\mapsto}node(q,x_n){*}G(x_n,x))
\end{array}
$$

1. Detect Dangling Predicates, e.g. $H_p(x_p, q)$
2. Eliminate Useless Parameters, e.g. parameter q of `H(x,q)`
3. Resue Predicates, e.g relate H with `sll` and G with with `dll`

```
requires sll(x)      ensures dll(x)
```

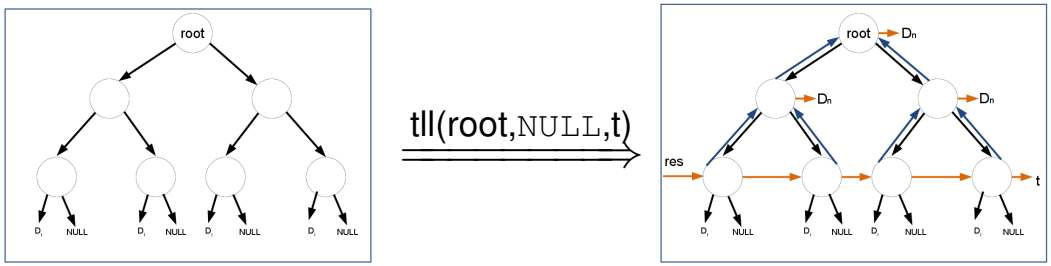
Other Issues:

1. Split Predicates
2. Lemma Synthesis Mechanism

Key steps

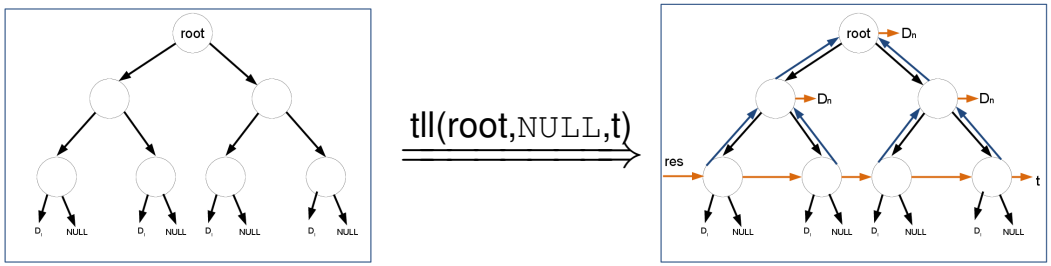
1. Forward verification
   - Rcollects the constraints over “unknown shape predicates” required for memory safety
2. Sound entailment via second-order bi-abduction
   - Rgenerates the constraints over “unknown shape predicates” required for the entailment to hold
3. Derivation of predicate definitions from the constraints
4. Normalization for “concise” and “reusable” predicate definitions

tll Example




```
1 // Iosif et. al. [CADE'13]
2 struct tree{ struct tree* parent; struct tree* left;
3   struct tree* right; struct tree* next;
4 };
5 struct tree* set_right (struct tree* x, struct tree* parent, struct tree* t){
6   x->parent=parent;
7   if (x->right==NULL){
8       x->next = t;
9       return x;
10  } else{
11    struct node* r_most = set_right(x->right, x, t);
12    return set_right(x->left, x, r_most);
13  }
14 }
```

tll Example



- We introduce two unknown predicates $H(x)$ and $G(x,p,\texttt{res},t)$

```
struct tree* tll(struct tree* x, struct tree* p, struct tree* t)
requires H(x)     ensures G(x,p,res,t)
```

- Our analysis generates:

$$H(x) \equiv x{\mapsto}tree(\_,\_,r,\_) \wedge \texttt{r=NULL} \\ \vee\ x{\mapsto}tree(\_,l,r,\_){*}H(l){*}H(r) \wedge \texttt{r} \neq \texttt{NULL}$$

$$G(x,p,\texttt{res},t) \equiv x{\mapsto}tree(p,\_,r,t) \wedge \texttt{res=x}\wedge\texttt{r=NULL} \\ \vee\ x{\mapsto}tree(p,l,r,\_){*}G(l,x,\texttt{res},lm){*}G(r,x,lm,t)\wedge\texttt{r}\neq\texttt{NULL}$$

Second-Order Bi-Abduction

For the tll example the set of constraints $\mathcal{R}$ is :

```
struct tree* tll
  (struct tree* x,
   struct tree* p,
   struct tree* t) {

 x->parent = p; (1)
 if (x->r = null) {
  x->next = t;
  return x; (2)
 } else {
  struct tree* lm;
  lm = tll(x->r, x, t); (3)
  return tll(x->l, x, lm); (4)
 }(5)
}
```

(1). $H(x,q,t) \Rightarrow x \mapsto tree(p,l,r,n) * H_l(l,q,t) * H_r(r,q,t) * H_n(n,q,t) * H_p(p,q,t)$

(2). $x \mapsto tree(p,l,r,t) * H_l(l,q,t) * H_p(p,q,t) * H_r(r,q,t) \wedge \mathtt{r}{=}\mathtt{NULL} \wedge \mathtt{res}{=}\mathtt{x} \Rightarrow G(x,p,\mathtt{res},t)$

(3). $H_r(r,q,t) \wedge \mathtt{r} \neq \mathtt{NULL} \Rightarrow H(r,x,t)$

(4). $H_l(l,q,t) \Rightarrow H(l,q,lm)$

(5). $x \mapsto tree(p,l,r,n) * H_n(n,q,t) * G(r,x,lm,t) * G(l,x,\mathtt{res},lm) \wedge \mathtt{r} \neq \mathtt{NULL} \Rightarrow G(x,p,\mathtt{res},t)$

Predicate Derivation

Starting from the set of constraints $\mathcal{R}$

- separate pre-preds constraints from post-preds constraints
- soundly derive definitions for the unknown predicates

Base Split

Separate pre-preds constraints from post-preds constraints

$$(4).\ x{\mapsto}tree(p,l,r,t){*}H_l(l,q,t){*}H_p(p,q,t){*}H_r(r,q,t){\wedge}\texttt{r=NULL}{\wedge}\texttt{res=x} \Rightarrow G(x,p,\texttt{res},t)$$

Is split into:

$$(4a).\ H_r(r,q,t){\wedge}\texttt{r=NULL} \Rightarrow \texttt{emp}$$

$$(4b).\ H_l(l,q,t) \Rightarrow \top$$

$$(4c).\ H_p(p,q,t) \Rightarrow \top$$

$$(4d).\ x{\mapsto}tree(p,l,r,t) \wedge \texttt{r=NULL} \wedge \texttt{res=x} \Rightarrow G(x,p,\texttt{res},t)$$

This split enables separate refinements for pre and post predicates.

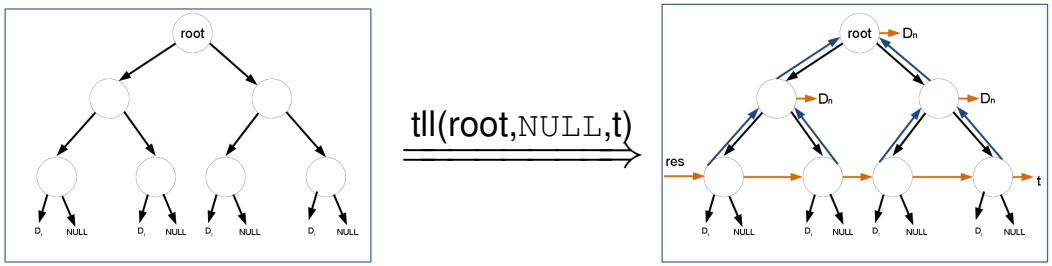
Predicate synthesis

After the application of the refinement rules (see the paper):

$$H_l(l,q,t) \equiv H(l,x,lm)$$
$$H_r(r,q,t) \equiv \texttt{emp}\wedge\texttt{r=NULL} \vee H(r,x,t)\wedge\texttt{r}\neq\texttt{NULL}$$
$$H_p(p,q,t) \equiv \top$$
$$H_n(n,q,t) \equiv \top$$
$$H(x,q,t) \equiv x\mapsto tree(p,l,r,n)*H_l(l,q,t)*H_r(r,q,t)*H_n(n,q,t)*H_p(p,q,t)$$

$$G(x,p,\texttt{res},t) \equiv x\mapsto tree(p,\_,r,t) \wedge \texttt{res=x}\wedge\texttt{r=NULL}$$
$$\vee\ x\mapsto tree(p,l,r,\_)*G(l,x,\texttt{res},lm)*G(r,x,lm,t)\wedge\texttt{r}\neq\texttt{NULL}$$

Normalization of predicate definitions



- User-Defined Predicate for binary tree

$$\texttt{pred}\ \texttt{bt}\langle\texttt{root}\rangle \equiv (\texttt{root}\mapsto\texttt{tree}\langle\_,\_,\texttt{NULL},\_\rangle) \vee (\texttt{root}\mapsto\texttt{tree}\langle\_,\texttt{l},\texttt{r},\_\rangle * \texttt{bt}\langle\texttt{l}\rangle * \texttt{bt}\langle\texttt{r}\rangle \wedge \texttt{r}\neq\texttt{NULL};$$

- User-Defined Predicate for tree with linked-leaves and parents

$$\texttt{pred}\ \texttt{tll}\langle\texttt{root},\texttt{p},\texttt{ll},\texttt{lr}\rangle \equiv (\texttt{root}\mapsto\texttt{tree}\langle\texttt{p},\_,\texttt{NULL}\rangle \wedge \texttt{root}=\texttt{ll}) \vee (\texttt{root}\mapsto\texttt{tree}\langle\texttt{p},\texttt{l},\texttt{r},\_\rangle * \texttt{tll}\langle\texttt{l},\texttt{root},\texttt{ll},\texttt{z}\rangle * \texttt{tll}\langle\texttt{r},\texttt{root},\texttt{z},\texttt{lr}\rangle \wedge \texttt{r}\neq\texttt{NULL};$$

- Inferred Specification

$$\texttt{requires}\ \texttt{bt}\langle\texttt{x}\rangle \quad \texttt{ensures}\ \texttt{tll}\langle\texttt{x},\texttt{p},\texttt{res},\texttt{t}\rangle;$$

Shape Inference: Implementation and Experiments

Example	Syn.	Veri.	Example	Syn.	Veri.
SLL (reverse)	0.21	0.2	SLL2DLL	0.19	0.18
SLL (insert)	0.2	0.2	DLL (check)	0.21	0.19
SLL (setTail)	0.16	0.16	DLL (append)	0.2	0.2
SLL (get-last)	0.7	0.21	CDLL (c)	0.22	0.21
SLL-sorted (c)	0.26	0.22	CDLL of 5CSLLs (c)	0.39	1.3
SLL (bubblesort)	0.28	0.26	CDLL of $CSLLs_2$ (c)	0.33	0.29
SLL (insertsort)	0.3	0.27	btree (search)	0.23	0.23
SLL (zip)	0.27	0.24	btree-parent (t)	0.23	0.24
SLL + tail (c)	0.19	0.18	rose-tree (c)	0.28	0.23
$skip\text{-}list_3$ (c)	0.36	0.3	swl (t)	0.23	22
SLL of 0/1 SLLs	0.25	0.23	mcf (c)	0.26	0.26
CSLL (t)	0.22	0.24	tll (t)	0.23	0.21
CSLL of CSLLs (c)	0.24	0.22	tll (set-parent)	0.24	0.24

c for *check* and **t** for *traverse*

- S2: `http://loris-7.ddns.comp.nus.edu.sg/~project/s2/beta/`
- synthesis $<$ 1s for all examples

Shape Inference: Implementation and Experiments

S2: `http://loris-7.ddns.comp.nus.edu.sg/~project/s2/beta/`

	LOC	#Proc	#Loop	#$\surd$	Syn. (second)
gslist.c	698	33	18	47	11.73
glist.c	784	35	19	49	7.43
gtree.c	1204	36	14	44	3.69
gnode.c	1128	37	27	52	16.34

- GLIB open source
- gtree: balanced binary tree. gnode: N-ary tree.
- shape analyse successfully for 89% of procedures/loops (192/216)
- limitation: overlaid data structures, array pointers, combined shape + pure (e.g. get_data_nth)

Normalization Experiment

Example	w/o norm.		w/ norm.	
	size	Syn.	size	Syn.
SLL (get-last)	20	0.24	17	0.24
SLL-sorted (c)	8	0.26	2	0.27
SLL (zip)	12	0.31	2	0.31
skip-list$_3$ (c)	17	0.45	1	0.46
CSLL (t)	8	0.23	5	0.23
CSLL of CSLLs (c)	18	0.24	4	0.23
SLL2DLL	18	0.19	2	0.2
DLL (append)	11	0.2	8	0.2
CDLL (c)	23	0.22	8	0.26
CDLL of 5CSLLs	28	0.39	4	0.66
CDLL of CSLLs$_2$	29	0.33	4	0.44
tree-parent (t)	11	0.23	2	0.29
rose-tree (c)	14	0.28	14	0.3
mcf (c)	19	0.26	17	0.28
tll (t)	21	0.23	2	0.25

reduce by 68% (169/533) the number of conjuncts with a time overhead of 26% (8.37s/10.62.s)

Remark

Shape Inference

1. scalability: potential
2. expressive data structures: $\surd$
3. soundness: $\surd$

Questions?

Thank you!