Automating Separation Logic using SMT

Ruzica Piskac   Thomas Wies   *Damien Zufferey*

MPI-SWS   NYU   IST Austria

MSR, June 24 2013, Redmond

Motivation

Separation logic (SL) succinctly express invariants of heap configurations.

Good features:

- Spatial conjunction (*),
- Inductive spatial predicates (list, tree, etc.),
- Frame rule.

Not so good features:
Specialized provers for decidable fragments means that extension and combination with other solvers/theories is not straightforward.

Example

```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null) {
    return b;
  } else {
    var curr: Node;
    curr := a;
    while (curr.next != null)
      invariant curr != null * lseg(a, curr) * lseg(curr, null);
    {
      curr := curr.next;
    }
    curr.next := b;
    return a;
  }
}
```

Example

Specification

```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null) {
    return b;
  } else {
    var curr: Node;
    curr := a;
    while (curr.next != null)
      invariant curr != null * lseg(a, curr) * lseg(curr, null);
    {
      curr := curr.next;
    }
    curr.next := b;
    return a;
  }
}
```

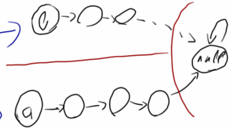
← pre/post

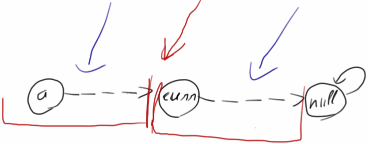
loop invariant

Example

* and inductive predicates

```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null) {
    return b;
  } else {
    var curr: Node;
    curr := a;
    while (curr.next != null)
      invariant curr != null * lseg(a, curr) * lseg(curr, null);
    {
      curr := curr.next;
    }
    curr.next := b;
    return a;
  }
}
```





Example

Frame inference

```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null) {
    return b;
  } else {
    var curr: Node;
    curr := a;
    while (curr.next != null)
      invariant curr != null * lseg(a, curr) * lseg(curr, null);
    {
      curr := curr.next;
    }
    curr.next := b;
    return a;
  }
}
```

$\bullet \vdash \bullet * F$

Our work

- Reduce a decidable fragment of SL to a decidable FO theory.
- Fits into the SMT framework.
- Satisfiability, entailment, frame inference, and abduction problems for SL using SMT solvers.
- Combining SL with other theories.
- Implemented in the GRASShopper tool.

Outline

1. Theoretical results
   - SLL$\mathbb{B}$ to GRASS
   - And back
2. Implementation

Decidable SL fragment: SLL$\mathbb{B}$

SLL (separation logic formulas for linked lists) introduced in [Berdine et al., 2004].

SLL

$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid \Sigma * \Sigma$$

With extend SLL to SLL by adding boolean connective on top:

$$H ::= \Sigma \mid \neg H \mid H \wedge H$$

Semantics of SLL$\mathbb{B}$ (1)

$\mathcal{A}, X \models_{\mathsf{SL}} H$
A: heap interpretation (total)
X: subset of A over which the formula is interpreted (footprint)

$$
\begin{array}{ll}
\mathcal{A}, X \models_{\mathsf{SL}} x = y & \text{iff } x^{\mathcal{A}} = y^{\mathcal{A}} \text{ and } X^{\mathcal{A}} = \emptyset \\
\mathcal{A}, X \models_{\mathsf{SL}} x \neq y & \text{iff } x^{\mathcal{A}} \neq y^{\mathcal{A}} \text{ and } X^{\mathcal{A}} = \emptyset \\
\mathcal{A}, X \models_{\mathsf{SL}} x \mapsto y & \text{iff } h^{\mathcal{A}}(x^{\mathcal{A}}) = y^{\mathcal{A}} \text{ and } X^{\mathcal{A}} = \{x^{\mathcal{A}}\}
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{A}, X \models_{\mathsf{SL}} H_1 * H_2 & \text{iff } \exists U_1, U_2.\, U_1 \cup U_2 = X^{\mathcal{A}} \text{ and } U_1 \cap U_2 = \emptyset \text{ and} \\
& \quad \mathcal{A}[X \mapsto U_1], X \models_{\mathsf{SL}} H_1 \text{ and } \mathcal{A}[X \mapsto U_2], X \models_{\mathsf{SL}} H_2
\end{array}
$$

Semantics of SLL$\mathbb{B}$ (2)

$$
\begin{array}{ll}
\mathcal{A}, X \models_{\mathsf{SL}} \mathsf{ls}(x,y) & \text{iff } \exists n \geq 0.\ \mathcal{A}, X \models_{\mathsf{SL}} \mathsf{ls}^n(x,y) \\
\mathcal{A}, X \models_{\mathsf{SL}} \mathsf{ls}^0(x,y) & \text{iff } x^{\mathcal{A}} = y^{\mathcal{A}} \text{ and } X^{\mathcal{A}} = \emptyset \\
\mathcal{A}, X \models_{\mathsf{SL}} \mathsf{ls}^{n+1}(x,y) & \text{iff } \exists u \in \mathsf{node}^{\mathcal{A}}.\ \mathcal{A}[z \mapsto u], X \models_{\mathsf{SL}} x \mapsto z * \mathsf{ls}^n(z,y) \\
& \quad \text{and } x^{\mathcal{A}} \neq y^{\mathcal{A}} \text{ and } z \neq x \text{ and } z \neq y
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{A}, X \models_{\mathsf{SL}} H_1 \wedge H_2 & \text{iff } \mathcal{A}, X \models_{\mathsf{SL}} H_1 \text{ and } \mathcal{A}, X \models_{\mathsf{SL}} H_2 \\
\mathcal{A}, X \models_{\mathsf{SL}} \neg H & \text{iff not } \mathcal{A}, X \models_{\mathsf{SL}} H
\end{array}
$$

GRASS: graph reachability and stratified sets

$$
\begin{array}{rcll}
 & & \text{graph reachability} & \\
T & ::= & x \mid h(T) & \\
A & ::= & T = T \mid T \xrightarrow{h \setminus T} T & \\
R & ::= & A \mid \neg R \mid R \wedge R \mid R \vee R & \\
 & & & \\
 & & \text{stratified sets} & \\
S & ::= & X \mid \emptyset \mid S \setminus S \mid S \cap S \mid S \cup S \mid \{x.\, R\} & x \text{ not below } h \text{ in } R \\
B & ::= & S = S \mid T \in S & \\
 & & & \\
 & & \text{top level boolean combination} & \\
F & ::= & A \mid B \mid \neg F \mid F \wedge F \mid F \vee F &
\end{array}
$$

GRASS

The theory $\mathcal{T}_{\mathsf{GS}}$ is the disjoint combination of:

- a theory of reachability in function graphs $\mathcal{T}_{\mathsf{G}}$
  - types: $\{\mathsf{node}\}$
  - function symbols $\{h\}$
  - predicate symbols $\{\xrightarrow{h\backslash}\}$
- a theory of stratified sets $\mathcal{T}_{\mathsf{S}}$ [Zarba, 2004]
  - types: $\{\mathsf{node}, \mathsf{set}\}$
  - function symbols $\{\emptyset, \cap, \cup, \setminus\}$
  - predicate symbols $\{\in\}$

$\mathcal{T}_{\mathsf{G}}$: theory of function graphs

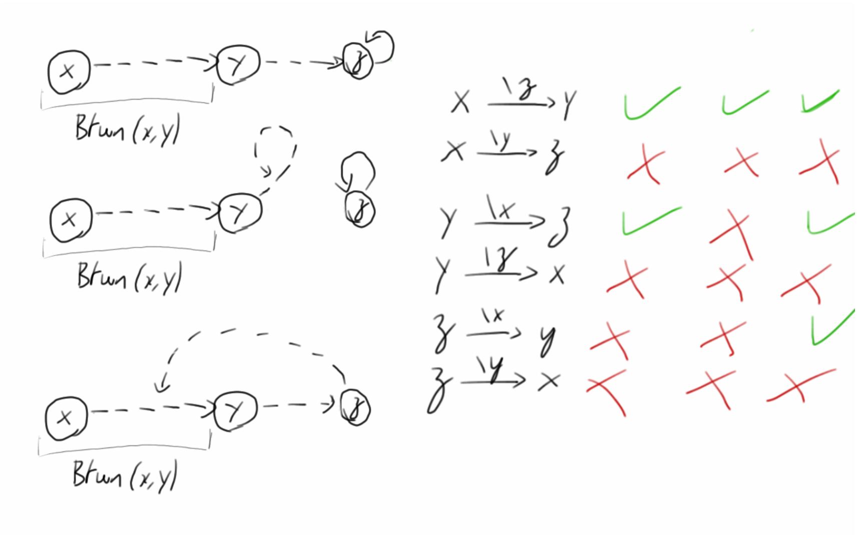
What is a function graph ?
A graph where each node has one outgoing edge (per function).

Why a graph and not just functions ?
Rather than just the successors we are interested of in paths (transitive closure of the functions).

$t_1 \xrightarrow{h \setminus t_3} t_2$ is true if there exists a path in the graph of h that connects t_1 and t_2 without going through t_3.

$\mathrm{ls}(x, y)$ is a shortcut for $x \xrightarrow{h \setminus y} y$
$Btwn(x, y)$ is a shortcut for $\{z.x \xrightarrow{h \setminus y} z \wedge z \neq y\}$

$\mathcal{T}_G$: examples



SLL$\mathbb{B}$ $\rightarrow$ GRASS (1)

Usual way of translating SL to FO:

- structure: uses $\mathcal{T}_\mathsf{G}$ to encode the shape of the heap (pointers)
- footprint: uses $\mathcal{T}_\mathsf{S}$ to encode the part of the heap used by a formula

Negation $\Rightarrow$ things get more complicated

- structure: uses $\mathcal{T}_\mathsf{G}$ and $\mathcal{T}_\mathsf{S}$ to encode the shape of the heap (pointers) and disjointness
- set definition: uses $\mathcal{T}_\mathsf{S}$ for keep track of the sets that will make the footprint

SLL$\mathbb{B}$ $\rightarrow$ GRASS: $*$ or below

$$
\begin{aligned}
str_Y(x = y) &= (x = y,\ \ Y = \emptyset)\\
str_Y(x \neq y) &= (x \neq y,\ \ Y = \emptyset)\\
str_Y(x \mapsto y) &= (h(x) = y,\ \ Y = \{x\})\\
str_Y(\mathrm{ls}(x, y)) &= (x \xrightarrow{h} y,\ \ Y = Btwn(x, y))\\
str_Y(\Sigma_1 * \Sigma_2) &= \text{let } \ Y_1, Y_2 \in \mathcal{X} \text{ fresh}\\
&\quad\ \text{and } (F_1, G_1) = tr_{Y_1}(\Sigma_1)\\
&\quad\ \text{and } (F_2, G_2) = tr_{Y_2}(\Sigma_2)\\
&\quad\ \text{in } (F_1 \wedge F_2 \wedge Y_1 \cap Y_2 = \emptyset,\ \ Y = Y_1 \cup Y_2 \wedge G_1 \wedge G_2)
\end{aligned}
$$

SLL$\mathbb{B}$ $\rightarrow$ GRASS: boolean structure

$$
\begin{aligned}
tr_X(\Sigma) &= \text{let } Y \in \mathcal{X} \text{ fresh and } (F, G) = str_Y(\Sigma) \\
&\qquad \text{in } (F \wedge X = Y,\ G) \\
tr_X(\neg H) &= \text{let } (F, G) = tr_X(H) \text{ in } (\neg F,\ G) \\
tr_X(H_1 \wedge H_2) &= \text{let } (F_1, G_1) = tr_X(H_1) \text{ and } (F_2, G_2) = tr_X(H_2) \\
&\qquad \text{in } (F_1 \wedge F_2,\ G_1 \wedge G_2) \\
Tr_X(H) &= \text{let } (F, G) = tr_X(H) \text{ in } F \wedge G
\end{aligned}
$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathrm{ls}(y, z)$$

translate to

$$x \neq z \wedge h(x) = y \wedge y \xrightarrow{h} z \wedge Y_2 \cap Y_3 = \emptyset \wedge Y_4 \cap Y_5 = \emptyset \wedge X = Y_1 \wedge Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathrm{ls}(y, z)$$

translate to

$$x \neq z \wedge h(x) = y \wedge y \xrightarrow{h} z \wedge Y_2 \cap Y_3 = \emptyset \wedge Y_4 \cap Y_5 = \emptyset \wedge X = Y_1 \wedge Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathrm{ls}(y, z)$$

translate to

$$x \neq z \wedge h(x) = y \wedge y \xrightarrow{h} z \wedge Y_2 \cap Y_3 = \emptyset \wedge Y_4 \cap Y_5 = \emptyset \wedge X = Y_1 \wedge Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathrm{ls}(y, z)$$

translate to

$$x \neq z \wedge h(x) = y \wedge y \xrightarrow{h} z \wedge Y_2 \cap Y_3 = \emptyset \wedge Y_4 \cap Y_5 = \emptyset \wedge X = Y_1 \wedge Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathrm{ls}(y, z)$$

translate to

$$x \neq z \wedge h(x) = y \wedge y \xrightarrow{h} z \wedge Y_2 \cap Y_3 = \emptyset \wedge Y_4 \cap Y_5 = \emptyset \wedge X = Y_1 \wedge Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathsf{ls}(y, z)$$

translate to

$$x \neq z \land h(x) = y \land y \xrightarrow{h} z \land Y_2 \cap Y_3 = \emptyset \land Y_4 \cap Y_5 = \emptyset \land X = Y_1 \land Y_1 = Y_2 \cup Y_3 \land Y_2 = \emptyset \land Y_3 = Y_4 \cup Y_5 \land Y_4 = \{x\} \land Y_5 = Btwn(y, z)$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathrm{ls}(y, z)$$

translate to

$$x \neq z \wedge h(x) = y \wedge y \xrightarrow{h} z \wedge Y_2 \cap Y_3 = \emptyset \wedge Y_4 \cap Y_5 = \emptyset \wedge X = Y_1 \wedge$$
$$Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Example: with negation

a non-empty acyclic list segment from x to z

$$\neg(x \neq z * x \mapsto y * \mathrm{ls}(y, z))$$

ignoring the negation (same as before):

structure

$$x \neq z \wedge h(x) = y \wedge y \xrightarrow{h} z \wedge Y_2 \cap Y_3 = \emptyset \wedge Y_4 \cap Y_5 = \emptyset \wedge X = Y_1$$

set definitions

$$Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Example: with negation

a non-empty acyclic list segment from x to z

$$\neg(x \neq z * x \mapsto y * \mathrm{ls}(y, z))$$

with negation (only the structure part is changed)

structure

$$x = z \vee h(x) \neq y \vee \neg y \xrightarrow{h} z \vee Y_2 \cap Y_3 \neq \emptyset \vee Y_4 \cap Y_5 \neq \emptyset \vee X \neq Y_1$$

set definitions

$$Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Why is that correct ?

Translation: $Tr_X(H) = \text{ let } (F, G) = tr_X(H) \text{ in } F \wedge G$

the auxiliary variables Y_i (in G) are existentially quantified

below negation, the existential quantifiers should become universal

Why is that correct ?

Translation: $Tr_X(H) = \text{ let } (F, G) = tr_X(H) \text{ in } F \wedge G$

the auxiliary variables Y_i (in G) are existentially quantified

below negation, the existential quantifiers should become universal

the Y_i are defined as finite unions of set comprehensions
$\rightarrow$ satisfiable in any given heap interpretation $\mathcal{A}$

Why is that correct ?

Translation: $Tr_X(H) = \text{ let } (F, G) = tr_X(H) \text{ in } F \wedge G$

the auxiliary variables Y_i (in G) are existentially quantified

below negation, the existential quantifiers should become universal

the Y_i are defined as finite unions of set comprehensions
$\rightarrow$ satisfiable in any given heap interpretation $\mathcal{A}$

Due to the precise semantics of SLL$\mathbb{B}$
$\rightarrow$ exists exactly one assignment of the Y_i that makes G true in $\mathcal{A}$

Why is that correct ?

Translation: $Tr_X(H) = \text{ let } (F, G) = tr_X(H) \text{ in } F \wedge G$

the auxiliary variables Y_i (in G) are existentially quantified

below negation, the existential quantifiers should become universal

the Y_i are defined as finite unions of set comprehensions
$\rightarrow$ satisfiable in any given heap interpretation $\mathcal{A}$

Due to the precise semantics of SLL$\mathbb{B}$
$\rightarrow$ exists exactly one assignment of the Y_i that makes G true in $\mathcal{A}$

$\exists Y_1, \ldots, Y_n.\, F \wedge G$ and
$\forall Y_1, \ldots, Y_n.\, G \Rightarrow F$ are equivalent.

Decision procedure for GRASS: $\mathcal{T}_S$

1. Transform F in nnf and eliminate all $S_1 \neq S_2$:

$$S_1 \neq S_2 \;\rightsquigarrow\; x \in S_1 \setminus S_2 \cup S_2 \setminus S_1 \qquad \text{where } x \in \mathcal{X} \text{ fresh}$$

2. Eliminate all set comprehensions by applying:

$$C[\{x.\, R\}] \;\rightsquigarrow\; C[X] \wedge (\forall x.\, x \in X \Leftrightarrow R) \qquad \text{where } X \in \mathcal{X} \text{ fresh}$$

3. Instantiate all universal quantifiers as follows. Let $t_1, \ldots, t_n$ be the terms of sort node that do not contain quantified variables. Then apply:

$$(\forall x.\, x \in X \Leftrightarrow R) \;\rightsquigarrow\; (t_1 \in X \Leftrightarrow R[t_1/x]) \wedge \ldots \wedge (t_n \in X \Leftrightarrow R[t_n/x])$$

This result is a quantifier-free Σ_{GS}-formula.

Decision procedure for GRASS: set reduction example (1)

Consider the GRASS formula (unsat):

$$F \equiv \{x.\, x \xrightarrow{h} y\} = \mathcal{U} \wedge y \xrightarrow{h} z \wedge \neg(w \xrightarrow{h} z)$$

After rewriting set operation:

$$F_2 \equiv S = U \wedge y \xrightarrow{h} z \wedge \neg(w \xrightarrow{h} z) \wedge (\forall x.\, x \in S \Leftrightarrow x \xrightarrow{h} y) \wedge (\forall x.\, x \in U \Leftrightarrow x = x)$$

After instantiating the quantifiers:

$$\begin{aligned} G \equiv\ & S = U \wedge y \xrightarrow{h} z \wedge \neg(w \xrightarrow{h} z) \wedge \\ & (y \in S \Leftrightarrow y \xrightarrow{h} y) \wedge (z \in S \Leftrightarrow z \xrightarrow{h} y) \wedge (w \in S \Leftrightarrow w \xrightarrow{h} y) \wedge \\ & (y \in U \Leftrightarrow y = y) \wedge (z \in U \Leftrightarrow z = z) \wedge (w \in U \Leftrightarrow w = w) \end{aligned}$$

Decision procedure for GRASS: set reduction example (2)

After instantiating the quantifiers:

$$
\begin{aligned}
G \equiv S = U \wedge y \xrightarrow{h} z \wedge \neg(w \xrightarrow{h} z) \wedge \\
(y \in S \Leftrightarrow y \xrightarrow{h} y) \wedge (z \in S \Leftrightarrow z \xrightarrow{h} y) \wedge (w \in S \Leftrightarrow w \xrightarrow{h} y) \wedge \\
(y \in U \Leftrightarrow y = y) \wedge (z \in U \Leftrightarrow z = z) \wedge (w \in U \Leftrightarrow w = w)
\end{aligned}
$$

To see that this formula is unsatisfiable in $\mathcal{T}_{\mathrm{GS}}$, we simplify G to the equivalent formula:

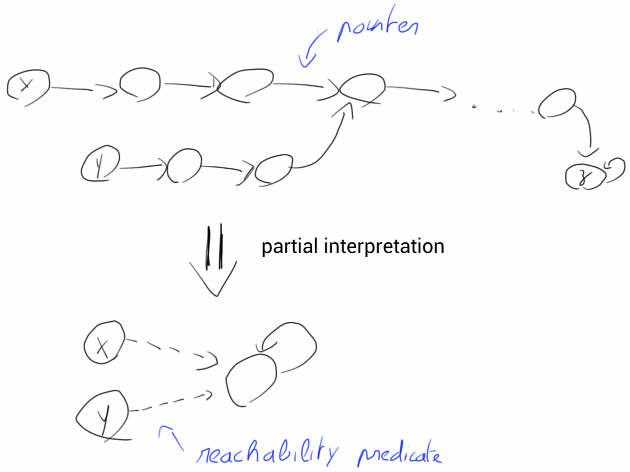
$$
\begin{aligned}
G' \equiv S = U \wedge y \xrightarrow{h} z \wedge \neg(w \xrightarrow{h} z) \wedge y \in U \wedge z \in U \wedge w \in U \wedge \\
(y \in S \Leftrightarrow y \xrightarrow{h} y) \wedge (z \in S \Leftrightarrow z \xrightarrow{h} y) \wedge (w \in S \Leftrightarrow w \xrightarrow{h} y)
\end{aligned}
$$

Decision procedure for GRASS: $\mathcal{T}_\mathsf{G}$

By [Totla and Wies, 2013] we know $\mathcal{T}_\mathsf{G}$ is a local theory extensions [Sofronie-Stokkermans, 2005]. We just need to instantiate a set of axioms on the ground terms in the formula.

$$
\begin{array}{rl}
\text{Reflexive} & x \xrightarrow{h\backslash u} x \\
\text{Step} & x \xrightarrow{h\backslash u} h(x) \lor x = u \\
\text{SelfLoop} & h(x) = x \land x \xrightarrow{h} y \Rightarrow x = y \\
\text{Sandwich} & x \xrightarrow{h\backslash x} y \Rightarrow x = y \\
\text{Reach} & x \xrightarrow{h\backslash u} y \Rightarrow x \xrightarrow{h} y \\
\text{Linear1} & x \xrightarrow{h} y \Rightarrow x \xrightarrow{h\backslash y} u \lor x \xrightarrow{h\backslash u} y \\
\text{Linear2} & x \xrightarrow{h\backslash u} y \land x \xrightarrow{h\backslash v} z \Rightarrow x \xrightarrow{h\backslash u} z \land z \xrightarrow{h\backslash u} y \lor x \xrightarrow{h\backslash v} y \land y \xrightarrow{h\backslash v} z \\
\text{Transitive1} & x \xrightarrow{h\backslash u} y \land y \xrightarrow{h\backslash u} z \Rightarrow x \xrightarrow{h\backslash u} z \\
\text{Transitive2} & x \xrightarrow{h\backslash z} y \land y \xrightarrow{h\backslash z} u \land y \xrightarrow{h} z \Rightarrow x \xrightarrow{h\backslash u} y
\end{array}
$$

Decision procedure for GRASS: $\mathcal{T}_G$



Where are we now ?

With the SLL$\mathbb{B}$ to GRASS translation we can

- Check for satisfiability
- Check entailment (reduces to satisfiability of $H_1 \wedge \neg H_2$)

For the (anti-)frame inference:
finding F in $A \models_{\mathsf{SL}} B * F$ (frame) or $A * F \models_{\mathsf{SL}} B$(antiframe)
we need the inverse translation

GRASS $\rightarrow$ SLL$\mathbb{B}$

Requirements:

- a GRASS formula F obtained from a SLL$\mathbb{B}$ formula (for the sake of simplicity)
- a model generating SMT solver (e.g. Z3),

Steps:

- get for all the partial interpretations that satisfy F
- for all a partial interpretation:
  - construct *succ* : node $\rightharpoonup$ node
  - extract the *pure* part from the interpretation
  - lift the interpretation to SL using h and *succ*.

where *succ* is the closest successor node in the partial interpretation

GRASS $\rightarrow$ SLL$\mathbb{B}$: example

$\ldots$

assume(ls(x, z));

if ($x \neq z$)

  free_head(x); //frame with precondition $x \mapsto y$

$\ldots$

GRASS:

$x \neq z \wedge x \xrightarrow{h} z \wedge h(x) = y \wedge X = Btwn(x, z) \wedge Y = \{x\} \wedge Z = X \setminus Y$

GRASS $\rightarrow$ SLL$\mathbb{B}$: example

. . .
assume(ls(x, z));
if ($x \neq z$)
    free_head(x); //frame with precondition $x \mapsto y$
. . .

GRASS:

$$x \neq z \wedge x \xrightarrow{h} z \wedge h(x) = y \wedge X = Btwn(x, z) \wedge Y = \{x\} \wedge Z = X \setminus Y$$

Partial interpretations:

$$\mathcal{B}_1 : \begin{tikzcd} x \arrow[r] & y, z \end{tikzcd}, \quad Z = \emptyset$$

$$\mathcal{B}_2 : \begin{tikzcd} x \arrow[r] & y \arrow[r, dashed] & z \end{tikzcd}, \quad Z = \{y\}$$

GRASS $\to$ SLL$\mathbb{B}$: example

...
assume(ls(x, z));
if ($x \neq z$)
    free_head(x); //frame with precondition $x \mapsto y$
...

GRASS:

$$x \neq z \wedge x \xrightarrow{h} z \wedge h(x) = y \wedge X = Btwn(x, z) \wedge Y = \{x\} \wedge Z = X \setminus Y$$

Partial interpretations:

$$\mathcal{B}_1 : x \longrightarrow y,z \;,\; Z = \emptyset$$

$$\mathcal{B}_2 : x \longrightarrow y \dashrightarrow z \;,\; Z = \{y\}$$

$$tr_Z^{-1}(\mathcal{B}_1) = x \neq z * x \neq y * y = z$$
$$tr_Z^{-1}(\mathcal{B}_2) = x \neq z * x \neq y * y \neq z * \mathrm{ls}(y, z)$$
$$Tr_Z^{-1}(F) = tr_Z^{-1}(\mathcal{B}_1) \vee tr_Z^{-1}(\mathcal{B}_2) \equiv x \neq z * x \neq y * \mathrm{ls}(y, z).$$

Combination with other theories and extensions

- The theories $\mathcal{T}_G$ and $\mathcal{T}_S$ are stably infinite with respect to sort node. (Nelson-Oppen)
- More pointers: we can extend the signature with field and uses $\bullet \xrightarrow{\bullet \setminus \bullet} \bullet$ with different fields. We can the also do read and write on the fields (array theory).
- Data: we can add data and constraints if it is local.
  $str_Y(\mathrm{sls}(x, y)) = (x \xrightarrow{h} y \wedge \forall z, w \in Y.\, z \xrightarrow{h} w \Rightarrow d(z) \leq d(w),\ Y = Btwn(x, y))$
- More complex data structures, e.g. doubly linked lists
  $$\begin{aligned} str_Y(\mathrm{dlls}(x, a, y, b)) = (\ & x \xrightarrow{n} y \wedge (x = y \wedge a = b \vee p(x) = a \wedge n(b) = y \wedge b \in Y) \wedge \\ & \forall z \in Y.\, n(z) \in Y \Rightarrow p(n(z)) = z,\ Y = Btwn(x, y)\ ) \end{aligned}$$

We are also considering implementing a decision procedure for trees.

Outline

1. Theoretical results

2. Implementation
   - GRASSHOPPER
   - Implicit frame inference
   - Experimental results

Reduction steps

We have implemented the translation is GRASSHOPPER.

Takes as input a program with SLL$\mathbb{B}$ specification and reduces it to a program with FO specification (Boogie-like)

The reduction is as follows:

1. if as choose + assume
2. replace loops by tail-recursive method
3. SLL$\mathbb{B}$ $\rightarrow$ GRASS, adding the heap (frame, memory accesses)
4. SSA, add assert/assume at call site

Let's look at a concrete example: merge sort.

Frame inference

Reconstructing the frame from the partial interpretations does not work (exponential in the works case).

Can we avoid the explicit computation of the frame ?
(e.g. have an axiomatic definition of the frame rule)

In previous example we had:

```
assume Frame(Alloc_1, Alloc_2, next, next_1);
```

assume Frame(Alloc_1, Alloc_2, next, next_1);

Meaning: a path which doesn't go through the frame is unchanged.

For this we need the entry point of x in the set X by following h, denoted by $ep_{X,h}(x)$

$Frame(X, A, h, h') =$

$$\begin{aligned}
&\forall x.\, x \in A \setminus X \Rightarrow \mathrm{sel}(h', x) = \mathrm{sel}(h, x) \wedge \\
&\forall x\, y\, z.\, x \xrightarrow{h \setminus ep_{X,h}(x)} y \Rightarrow (x \xrightarrow{h \setminus z} y \Leftrightarrow x \xrightarrow{h' \setminus z} y) \wedge \\
&\forall x\, y\, z.\, x \in A \setminus X \wedge ep_{X,h}(x) = x \Rightarrow (x \xrightarrow{h \setminus z} y \Leftrightarrow x \xrightarrow{h' \setminus z} y)
\end{aligned}$$

$ep_{X,h}(x)$

Axioms defining the entry point function:

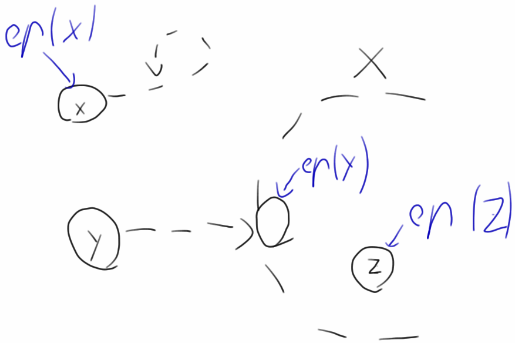
$$\forall x.\, x \xrightarrow{h} ep_{X,h}(x)$$

$$\forall x.\, ep_{X,h}(x) \in X \vee ep_{X,h}(x) = x$$

$$\forall x\, y.\, x \xrightarrow{h} y \wedge y \in X \Rightarrow ep_{X,h}(x) \in X \wedge x \xrightarrow{h \backslash y} ep_{X,h}(x)$$

epX, h is local (idempotent), we can use the same approach as $\mathcal{T}_{\mathsf{G}}$.

ep



experiments

program	sl		dl		rec sl		sls	
	#	t	#	t	#	t	#	t
concat	4	0.1	5	1.3	6	0.6	5	0.2
copy	4	0.2	4	3.9	6	0.8	7	3.5
filter	7	0.6	5	1.1	8	0.4	5	1.1
free	5	0.1	5	0.3	4	0.1	5	0.1
insertion sort							10	0.7
merge sort							25	24

program	sl		dl		rec sl		sls	
	#	t	#	t	#	t	#	t
insert	6	0.2	5	1.5	5	0.2	6	0.4
reverse	4	0.1	4	0.5	6	0.2	4	0.2
remove	8	0.2	8	0.8	7	0.2	7	0.5
traverse	4	0.1	5	0.3	3	0.1	4	0.2
double all							7	2.2
pairwise sum							10	20

sl singly-linked list (loop or recursion)

dl doubly-linked list

sls sorted lists

number of VCs

t total time in second

Related work

- Most prominent decidable fragments of SL: linked lists [Berdine et al., 2004], decidable in polynomial time [Cook et al., 2011] (graph-based).
- SL $\rightarrow$ FO: [Calcagno and Hague, 2005] (no inductive predicate) and [Bobot and Filliâtre, 2012] (not a decidable fragment).
- Alternatives to SL: (implicit) dynamic frames [Kassios, 2011] and region logic [Banerjee et al., 2008, Rosenberg et al., 2012].
- The connection between SL and implicit dynamic frames has been studied in [Parkinson and Summers, 2012].
- SMT-based decision procedures for theories of reachability in graphs [Lahiri and Qadeer, 2008, Wies et al., 2011, Totla and Wies, 2013] , decision procedures for theories of stratified sets [Zarba, 2004].

Work in progress, future work

- dealing with the frame (still work in progress)
- more example using other theories (arrays, integers, ...)
- inferring GRASS predicate definition from SLL$\mathbb{B}$ definition
- decision procedure for trees
- abstraction/modularity (generic list)
- etc.

Questions ?

Banerjee, A., Naumann, D. A. and Rosenberg, S. (2008).
Regional Logic for Local Reasoning about Global Invariants.
In ECOOP vol. 5142, of LNCS pp. 387–411,.

Berdine, J., Calcagno, C. and O'Hearn, P. (2004).
A Decidable Fragment of Separation Logic.
In FSTTCS.

Bobot, F. and Filliâtre, J.-C. (2012).
Separation Predicates: a Taste of Separation Logic in First-Order Logic.
In ICFEM.

Calcagno, C. and Hague, M. (2005).
From separation logic to first-order logic.
In FoSSaCs'05 pp. 395–409, Springer.

Cook, B., Haase, C., Ouaknine, J., Parkinson, M. and Worrell, J. (2011).
Tractable Reasoning in a Fragment of Separation Logic.
In CONCUR. Springer.

Kassios, I. T. (2011).
The dynamic frames theory.
Formal Asp. Comput. *23*, 267–288.

Lahiri, S. K. and Qadeer, S. (2008).
Back to the future: revisiting precise program verification using SMT solvers.