On Symbolic Heaps Modulo Permission Theories

Stéphane Demri (ENS Cachan), Etienne Lozes (Université Nice Sophia)

and Denis Lugiez (Université Marseille)

FSTTCS – Kanpur – december 2017

Outline

1. Background : separation logic and symbolic heaps
2. What are permissions ?
3. Contribution 1 : permissions constraints generation
4. Contribution 2 : permissions constraints solving

Separation Logic in a nutshell

Separation logic is an extension of Hoare-Floyd logic for both heap-manipulating and concurrent programs. It is based on the following two principles :

Ownership

A code fragment can access only those portions of state that it owns.

Separation

At any time, the state can be partitioned into that owned by each function or thread or “module”.

An example proof in separation logic

$$\{\mathsf{emp}\}$$

```
x = new()
```

$$\{x \mapsto -\}$$

```
y = new()
```

$$\{(x \mapsto -) * (y \mapsto -)\}$$

$\{x \mapsto -\}$	$\{y \mapsto -\}$
`free(x)`	`free(y)`
$\{\mathsf{emp}\}$	$\{\mathsf{emp}\}$

$$\{\mathsf{emp} * \mathsf{emp}\}$$

$$\{\mathsf{emp}\}$$

Symbolic Heaps : a fragment of separation logic

$$\Pi ::= \quad \top \mid x = y \mid x \neq y \mid \Pi \wedge \Pi$$

(pure formula)

$$\Sigma ::= \quad \mathsf{emp} \mid \top \mid x \mapsto y \mid \mathsf{lseg}(x, y) \mid \Sigma * \Sigma$$

(spatial formula)

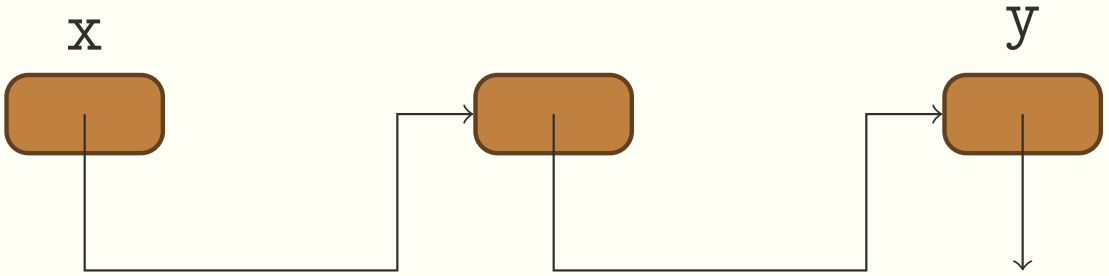
where $\mathrm{LVAR} = \{x, y, \ldots\}$ countably set of **location variables**.

A Simple Memory Model

a state $(s, h) \in \Sigma$ is a pair composed of a store s and heap h with

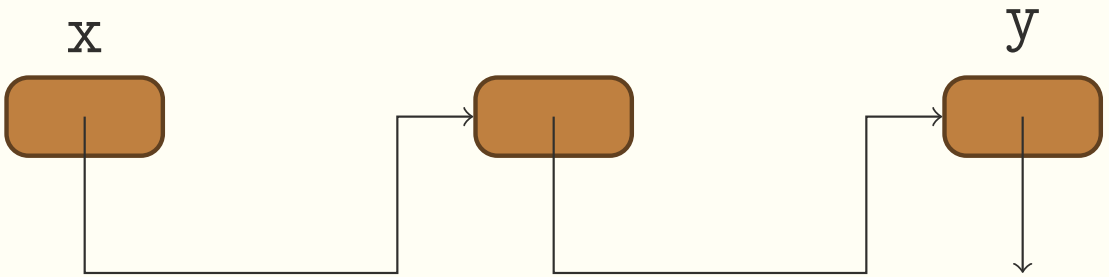
$$s : \mathsf{LVAR} \to \mathsf{Loc} \uplus \{\mathsf{null}\} \qquad h : \mathsf{Loc} \rightharpoonup_{\mathsf{fin}} \mathsf{Loc} \uplus \{\mathsf{null}\}$$

where $\rightharpoonup_{\mathsf{fin}}$ means partial functions with finite domain, and Loc is an infinite set (for instance $\mathsf{Loc} = \mathsf{N}$).



$$\begin{aligned} s(x) &= \ell_1 \\ s(y) &= \ell_3 \\ \mathsf{dom}(h) &= \{\ell_1, \ell_2, \ell_3\} \\ h(\ell_1) &= \ell_2 \\ h(\ell_2) &= \ell_3 \\ h(\ell_3) &= \ell_4 \end{aligned}$$

Heap Composition



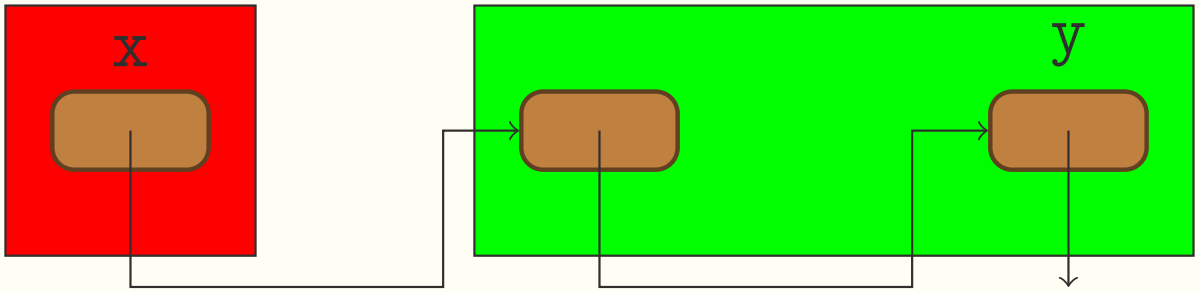
$$
\begin{aligned}
s(x) &= \ell_1 \\
s(y) &= \ell_3 \\
\mathrm{dom}(h) &= \{\ell_1, \ell_2, \ell_3\} \\
h(\ell_1) &= \ell_2 \\
h(\ell_2) &= \ell_3 \\
h(\ell_3) &= \ell_4
\end{aligned}
$$

Definition

$h = h_1 \bullet h_2$ if

1. $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2)$
2. $\mathrm{dom}(h) = \mathrm{dom}(\mathrm{h}_1) \cup \mathrm{dom}(\mathrm{h}_2)$
3. for all $i \in \{1, 2\}$ and $\ell \in \mathrm{dom}(h_i)$, $h(\ell) = h_i(\ell)$

Heap Composition



$$
\begin{aligned}
s(x) &= \ell_1 \\
s(y) &= \ell_3 \\
\mathrm{dom}(h) &= \{\ell_1, \ell_2, \ell_3\} \\
h(\ell_1) &= \ell_2 \\
h(\ell_2) &= \ell_3 \\
h(\ell_3) &= \ell_4
\end{aligned}
$$

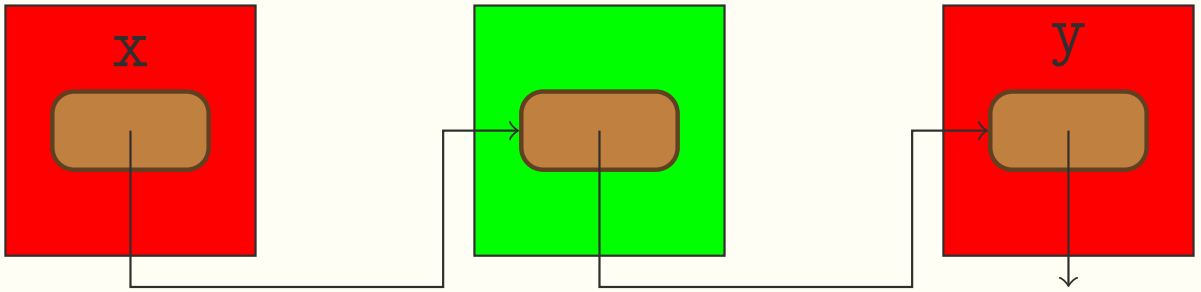
$$h = h_1 \bullet h_2$$

Definition

$h = h_1 \bullet h_2$ if

1. $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2)$
2. $\mathrm{dom}(h) = \mathrm{dom}(\mathrm{h}_1) \cup \mathrm{dom}(\mathrm{h}_2)$
3. for all $i \in \{1, 2\}$ and $\ell \in \mathrm{dom}(h_i)$, $h(\ell) = h_i(\ell)$

Heap Composition



$$s(x) = \ell_1$$
$$s(y) = \ell_3$$
$$\mathrm{dom}(h) = \{\ell_1, \ell_2, \ell_3\}$$
$$h(\ell_1) = \ell_2$$
$$h(\ell_2) = \ell_3$$
$$h(\ell_3) = \ell_4$$

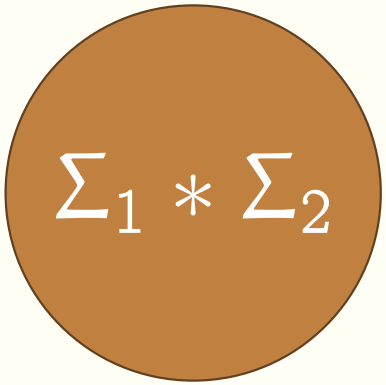
$$h = h_1 \bullet h_2$$

Definition

$h = h_1 \bullet h_2$ if

1. $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2)$
2. $\mathrm{dom}(h) = \mathrm{dom}(h_1) \cup \mathrm{dom}(h_2)$
3. for all $i \in \{1, 2\}$ and $\ell \in \mathrm{dom}(h_i)$, $h(\ell) = h_i(\ell)$

Separating Conjunction

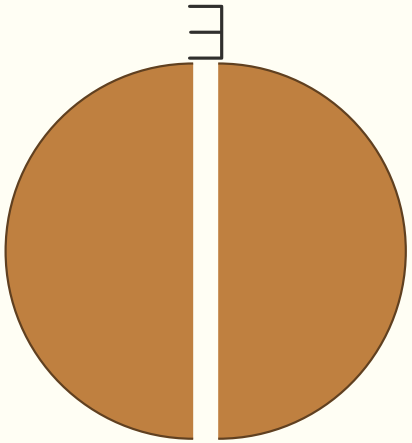


$s, h \models \Sigma_1 * \Sigma_2$

if there is h_1, h_2 s.t.

- $h = h_1 \bullet h_2$, and
- $s, h_1 \models \Sigma_1$, and
- $s, h_2 \models \Sigma_2$
- $s, h \models \text{emp}$

Separating Conjunction

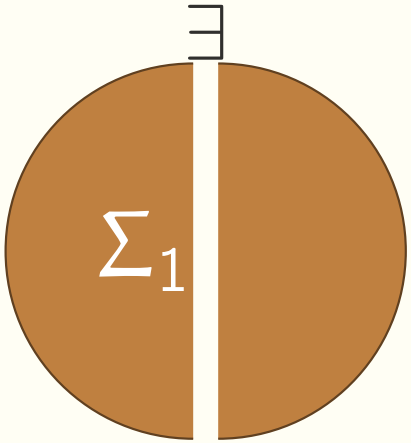


$s, h \models \Sigma_1 * \Sigma_2$

if there is h_1, h_2 s.t.

- $h = h_1 \bullet h_2$, and
- $s, h_1 \models \Sigma_1$, and
- $s, h_2 \models \Sigma_2$
- $s, h \models \mathsf{emp}$

Separating Conjunction

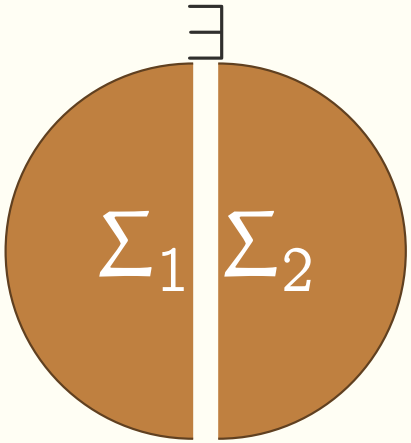


$s, h \models \Sigma_1 * \Sigma_2$

if there is h_1, h_2 s.t.

- $h = h_1 \bullet h_2$, and
- $s, h_1 \models \Sigma_1$, and
- $s, h_2 \models \Sigma_2$
- $s, h \models \mathsf{emp}$

Separating Conjunction

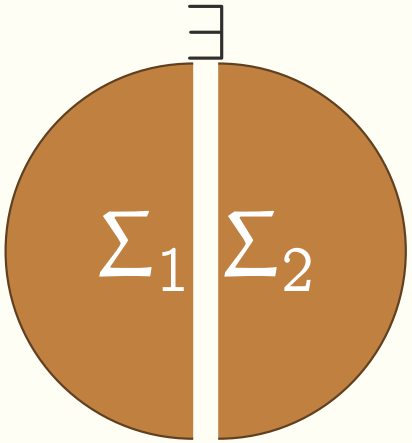


$s, h \models \Sigma_1 * \Sigma_2$

if there is h_1, h_2 s.t.

- $h = h_1 \bullet h_2$, and
- $s, h_1 \models \Sigma_1$, and
- $s, h_2 \models \Sigma_2$
- $s, h \models \mathsf{emp}$

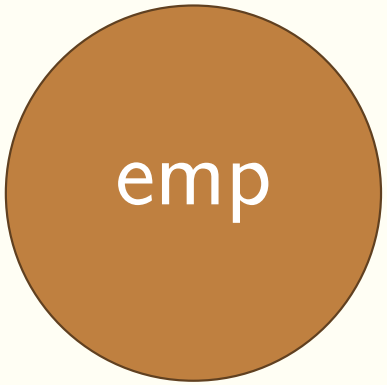
Separating Conjunction



$s, h \models \Sigma_1 * \Sigma_2$

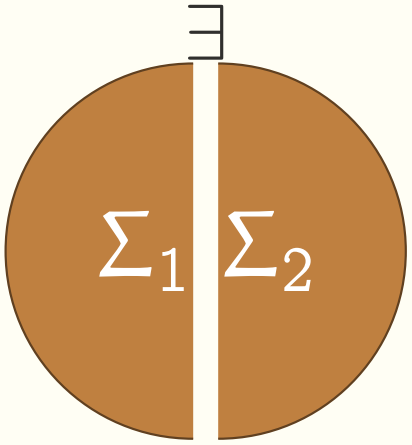
if there is h_1, h_2 s.t.

- $h = h_1 \bullet h_2$, and
- $s, h_1 \models \Sigma_1$, and
- $s, h_2 \models \Sigma_2$
- $s, h \models \text{emp}$



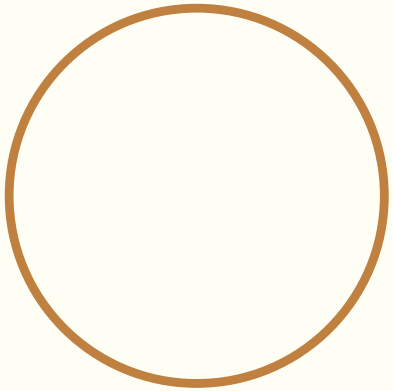
$s, h \models \text{emp}$

Separating Conjunction



$s, h \models \Sigma_1 * \Sigma_2$
if there is h_1, h_2 s.t.

- $h = h_1 \bullet h_2$, and
- $s, h_1 \models \Sigma_1$, and
- $s, h_2 \models \Sigma_2$
- $s, h \models \text{emp}$



$s, h \models \text{emp}$
if $h = \emptyset$

List Predicate

- Inductive definition

$$\begin{array}{rcll} \mathsf{lseg}(x,y) & \Leftrightarrow & & x = y \wedge \mathsf{emp} \\ & & \vee & x \neq y \wedge \exists z.x \mapsto z * \mathsf{lseg}(z,y) \end{array}$$

- Explicit definition : $(s,h) \models \mathsf{lseg}(x,y)$ if
  - either $s(x) = s(y)$
  - or $s(x) = l_1, s(y) = l_n$, $\mathsf{dom}(h) = \{\ell_1, \ldots, \ell_{n-1}\}$ all ℓ_i distinct, and $h(\ell_i) = \ell_{i+1}$

- in particular : no cyclic loop, and $x \mapsto x \not\models \mathsf{lseg}(x,x)$

Two problems of concerns for verification

$$\Pi ::= \quad \top \mid x = y \mid x \neq y \mid \Pi \wedge \Pi$$
(pure formula)

$$\Sigma ::= \quad \mathsf{emp} \mid \top \mid x \mapsto y \mid \mathsf{lseg}(x, y) \mid \Sigma * \Sigma$$
(spatial formula)

- ➡ Satisfiability (Π, Σ) has a model ?
- ➡ Entailment $(\Pi, \Sigma) \models (\Pi', \Sigma')$

Both problems are in PTIME
(Haase et al, CONCUR'11).

Permissions

Idea :

- ➡ full permission 1 = write access granted
  $(x \overset{1}{\mapsto} -)$: owns x exclusively, can read modify, and deallocate the cell x
- ➡ partial permission π = read shared access
  $(x \overset{\pi}{\mapsto} -)$: owns a "fraction of" cell x, can read it, but cannot modify or deallocate it
- ➡ $\oplus$ adds permissions
  gather all read permissions to recover write access :
  $(x \overset{1/2}{\mapsto} y) * (x \overset{1/2}{\mapsto} y) = (x \overset{1}{\mapsto} -)$

A proof with permissions

$$\{\mathsf{emp}\}$$

```
x = new()
```

$$\{x \mapsto -\}$$

```
x->n = null;
```

$$\{x \mapsto \mathsf{null}\}$$

$$\begin{array}{c|c} \{x \overset{1/2}{\mapsto} \mathsf{null}\} & \{x \overset{1/2}{\mapsto} \mathsf{null}\} \\ \texttt{z = x->n} & \texttt{t = x->n)} \\ \{(x \overset{1/2}{\mapsto} z) \wedge z = \mathsf{null}\} & \{(x \overset{1/2}{\mapsto} t) \wedge t = \mathsf{null}\} \end{array}$$

$$\{(x \overset{1/2}{\mapsto} z) \wedge z = \mathsf{null} * (x \overset{1/2}{\mapsto} t) \wedge t = \mathsf{null}\}$$

$$\{(x \mapsto \mathsf{null})\}$$

```
free(x);
```

$$\{\mathsf{emp}\}$$

Syntax extension

Predicate $x \overset{p}{\mapsto} y$ where p is a permission term

$$
\begin{array}{lll}
p ::= & 1 \mid \alpha \mid p \oplus p & \\
& \text{(permission term)} & \\
A ::= & \top \mid p = p \mid p \leq p \mid \mathsf{defined}(p) \mid A \wedge A & \\
& \text{(permission formula)} & \\
\Pi ::= & \top \mid x = y \mid x \neq y \mid A \mid \Pi \wedge \Pi & \\
& \text{(pure formula)} &
\end{array}
$$

$$
\begin{array}{lll}
\Sigma ::= & \mathsf{emp} \mid \top \mid x \overset{p}{\mapsto} y \mid \mathsf{lseg}_p(x, y) \mid \Sigma * \Sigma & \\
& \text{(spatial formula)} &
\end{array}
$$

Semantics

Heap model with permissions

$h : \mathsf{Loc} \rightharpoonup_{\mathsf{fin}} P_{\mathfrak{P}} \times \mathsf{Loc}$

Heap composition

$h = h_1 \bullet h_2$ if

- $\mathsf{dom}(h) = \mathsf{dom}(h_1) \cup \mathsf{dom}(h_2)$
- $h(\ell) = h_i(\ell)$ for all $\ell \in \mathsf{dom}(()h_i) \setminus \mathsf{dom}(h_{2-i})$
- for all $\ell \in \mathsf{dom}(h_1) \cap \mathsf{dom}(h_2)$, $h(\ell) = (\pi_1 \oplus_{\mathfrak{P}} \pi_2, l')$ if

$$\begin{cases} \mathit{defined}(\pi_1 \oplus_{\mathfrak{P}} \pi_2) \\ h_i(l) = (\pi_i, l') \\ l \in \mathit{dom}(h_1) \cap \mathit{dom}(h_2) \end{cases}$$

- Satisfaction relation $(s, h, \iota) \models (\Pi, \Sigma)$

Some permission models

1. Trivial model (only 1).
2. Boyland's fractional model $\mathfrak{P}_{\mathsf{Boy}} = (]0,1], 1, +)$ with $+$ defined if the sum is less or equal to 1,
3. Bornat *et al* token counting model $\mathfrak{P}_{\mathsf{Tok}} = (\mathcal{P}_{\mathit{fin}}(\mathsf{N}) \cup \mathcal{P}_{\mathit{cofin}}(\mathsf{N}), \mathsf{N}, \uplus)$
4. Hobor *et al* tree share model (binary trees)
5. combinations ...

Axiom and permission-model dependent formulas

Axiom (true for all permission models)

$$(x \overset{p_1}{\mapsto} y_1) * (x \overset{p_2}{\mapsto} y_2) \quad \Leftrightarrow \quad (x \overset{p}{\mapsto} y_1) \wedge y_1 = y_2 \wedge p = p_1 \oplus p_2$$

Disjointness axiom (false in the fractionnal model)

$$(x \overset{p}{\mapsto} -) * (x \overset{p}{\mapsto} -) \quad \Rightarrow \quad \bot$$

Atomicity axiom (only true in the trivial model)

$$(x \overset{p_1}{\mapsto} -) * (x \overset{p_2}{\mapsto} -) \quad \Rightarrow \quad \bot$$

Automating sat and entailment : state of the art

Fractional Model

1. tool support (VCC,Verifast,Chalice,...) but no formalization of the decision procedure implemented
2. common belief : in PTIME, "simple" reduction to linear programming

Tree share model

1. a decision procedure for permission constraints
   Bach Le, Gherghina, Hobor. APLAS'12
2. complexity results for a permission logic
   Bach Le, Hobor, Widjaja Lin. FSTTCS'16
3. certified decision procedure. Bach Le *et al*, ICFEM'17

Our contribution : overview

1. first formal decision procedure for symbolic heaps
2. its exact complexity
3. parametricity in the permission model
4. exact complexities of constraints in several permission models
5. support for the list predicate

Contribution 1 : symbolic heaps without lists

Fractionnal permissions

SAT and ENT are PTIME-complete

- ➡ as common belief, but now proved,
- ➡ only without lists

Token and tree permissions

SAT is NP complete, ENT is co-NP complete.

- ➡ improves on $\mathrm{NP}^{\mathrm{NP}}$ conjecture for SAT in APLAS'12

Normalization rules

$$(\Pi, \Sigma) \quad \Rightarrow \quad (\Pi, \Sigma[y/x]) \quad \text{if } \Pi \models x = y \text{ and } \{x, y\} \subseteq \mathsf{LVAR}(\Sigma)$$

$$(\Pi, \Sigma) \quad \Rightarrow \quad (\Pi, \Sigma[y/x]) \quad \text{if } \Pi \models x = y \text{ and } \{x, y\} \subseteq \mathsf{LVAR}(\Sigma)$$

$$(\Pi, \Sigma * x \overset{p}{\mapsto} y * x \overset{p'}{\mapsto} z) \quad \Rightarrow \quad (\Pi \wedge y = z, \Sigma * x \overset{p \oplus p'}{\mapsto} y)$$

$$(\Pi, \Sigma) \quad \Rightarrow \quad \bot \quad \text{if } \Pi_{pv} \models x \neq y,\ \Pi_{pv} \models x = y$$

$$(\Pi, \Sigma * \mathsf{emp}) \quad \Rightarrow \quad (\Pi, \Sigma) \quad \text{if non-empty } \Sigma$$

$$(\Pi, \Sigma * \top * \top) \quad \Rightarrow \quad (\Pi, \Sigma * \top)$$

Theorem

1. *Normalization terminates in PTIME and preserves semantics*
2. (Π, Σ) *in normal form is satisfiable iff* $\Pi_{pe} \wedge \mathit{defined}(\Sigma)$ *satisfiable (*Π_{pe} *permission part of* Π*).*

A short word about entailment

- ➡ extend rewrite rules to entailment problems
- ➡ again PTIME with an oracle on entailments of permission constraints
- ➡ more complex : may need several calls to the oracle

Contribution 2 : a decision procedure for permissions and lists

Main result

For all considered permission models (but the trivial one)

- ➡ SAT is NP complete
- ➡ ENT is co-NP complete

Remarks

- ➡ Even fractionnal permissions are (co)NP-hard
- ➡ contrasts with PTIME without permissions

Lower bound

Theorem

$\mathrm{SATSH}(\mathfrak{P})$ *is* NP*-hard*

(under the mild hypothesis on the permission model that unboundedly many permissions can be summed.)

1. 3-colorability instance $G = (V = \{x_1, \ldots, x_n\}, E)$.
2. Formula

$$(\bigwedge_{\{x_i,x_j\}\in E}(x_i \neq x_j)) \wedge y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_2 \neq y_3$$
$$y_1 \overset{\alpha_0}{\mapsto} y_2 * y_2 \overset{\alpha_0'}{\mapsto} y_3 * y_3 \overset{1}{\mapsto} y_1 *_{x_i \in V} (\mathsf{lseg}_{\alpha_i}(y_1, x_i) * \mathsf{lseg}_{\alpha_i'}(x_i, y_3))$$

G 3-colorable iff (Π_G, Σ_G) satisfiable.

General scheme for decidability

$$(\Pi, \Sigma) \models (\Pi', \Sigma')?$$

SL graph : graphical representation of a formula

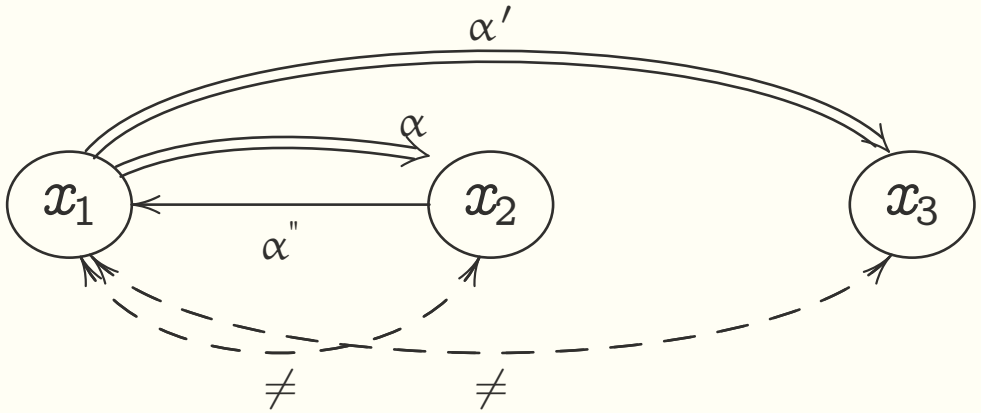
SL-graph G $\leftrightarrow$ formula $(pure(G), spatial(G))$

Formula (Π, Σ) $\leftrightarrow$ SL graph

$(\Pi, \Sigma) \not\models (\Pi', \Sigma')$ iff

- $\exists G_D$ deterministic SL-graph
- G_D represents a satisfiable formula
- G_D is *small*
- $h_{G,G_D} : G \to G_D$ precise homomorphism
- $h_{G',G_D} : G' \to G_D$ homomorphism not strongly precise

SL-graphs



the formula $(pure(G), spatial(G))$ associated to G is

$$\left(x_1 \neq x_2 \wedge x_2 \neq x_3 \ , \ \mathsf{lseg}_{\alpha}(x_1, x_2) * \mathsf{lseg}_{\alpha'}(x_1, x_3) * x_2 \overset{\alpha''}{\mapsto} x_1\right)$$

G is deterministic iff there is at most one edge $\overset{p}{\rightarrow}$ or $\overset{p}{\Rightarrow}$ issued from a node.

Precise Graph homomorphism

$G_1 = (A_1, V_1, \rightarrow_1, \Rightarrow_1, \overset{\neq}{\leftrightarrow}_1, L_1)$,
$G_2 = (A_2, V_2, \rightarrow_2, \Rightarrow_2, \overset{\neq}{\leftrightarrow}_2, L_2)$ deterministic
$h : G_1 \rightarrow G_2$ precise homomorphism iff :

1. $h : V_1 \rightarrow V_2$ surjective
2. $A_2' = A_2 \wedge \bigwedge_{p \in G_2} \mathit{defined}(p) \models A_1$
3. $V_1 \ni \mathit{var}(v) \subseteq \mathit{var}(h(v))$
4. $G_1 \ni v \xrightarrow{p} v'$ implies $\exists p'\ h(v) \xrightarrow{p'} h(v') \in G_2$,
5. $G_1 \ni v \overset{p}{\Rightarrow} v'$ implies $h(v) = h(v')$ and $A_2' \models \mathit{defined}(p)$ or there is a path $v_0 \overset{p_0}{\rightsquigarrow}_2 v_1 \overset{p_1}{\rightsquigarrow}_2 v_2 \cdots \overset{p_{n-1}}{\rightsquigarrow}_2 v_n = h(v')$
6. Each edge of G_2 contributes to at least one edge of G_1.

Strongly precise homomorphism

Modifications :

- **1'** $A_2 = A_1 \wedge \bigwedge_{p \in G_2} \mathit{defined}(p)$
- **5'** no condition $A_2' \models \mathit{defined}(p)$ when $h(v) = h(v')$
- **6'** Each edge $\overset{p'}{\rightsquigarrow}$ of G_2 contributes to at least one edge $p' = \bigoplus\{\{p \mid v \overset{p'}{\rightsquigarrow} v' \text{ contributes to } u \overset{p}{\rightsquigarrow} u'\}\}$

Syntactic conditions replace semantics conditions.

Main Theorem

Theorem

- *Satisfiability is* NP*-complete and Entailment is* coNP*-complete for* $\mathfrak{P}_{\mathsf{Boy}}$
- *Satisfiability is* NP*-complete and Entailment is* coNP*-complete for any model* $\mathfrak{P}$ *of width* ω *with an entailment problem in* coNP*.*

Precise bound.

CONCLUSION

- ➡ Optimal complexity results for separation logic with lists and permissions.
- ➡ Parametrized by the permission model
- ➡ Complexity results on permission models.

PERSPECTIVES

- ➡ A proof system ?
- ➡ Richer permission constraints
- ➡ More recursive structures (trees,...)