# Generically Automating Separation Logic by Functors, Homomorphisms, and Modules

Qiyuan Xu[1], David Sanán[2], Zhé Hóu[3], Xiaokun Luan[4], Conrad Watt[1], Yang Liu[1]

1: Nanyang Technological University
2: Singapore Institute of Technology
3: Griffith University, Australia
4: Peking University, China

# Motivation: Scalability Issue in Separation Logic (SL)

- We use predicates to represent data structures.
- Abstract predicates hide complexities from internal implementations.
- *But...* complexities are merely invisible 😒, not eliminated.

*But...* complexities are merely invisible 😒, not eliminated.

- Current reasoning mechanisms have to unfold them if necessary.
- If you unfold predicates, all the hidden complexities come out. Expression explosion! 🌋
- Unfolding is bad 👎. Compositional reasoning is good 👍.

Compositional reasoning relies on reasoning rules of predicates.

Where do we get the reasoning rules?

- Manually proven? People are lazy. 🥴
- Automatically generated? Hey! Look at our work! 🤩

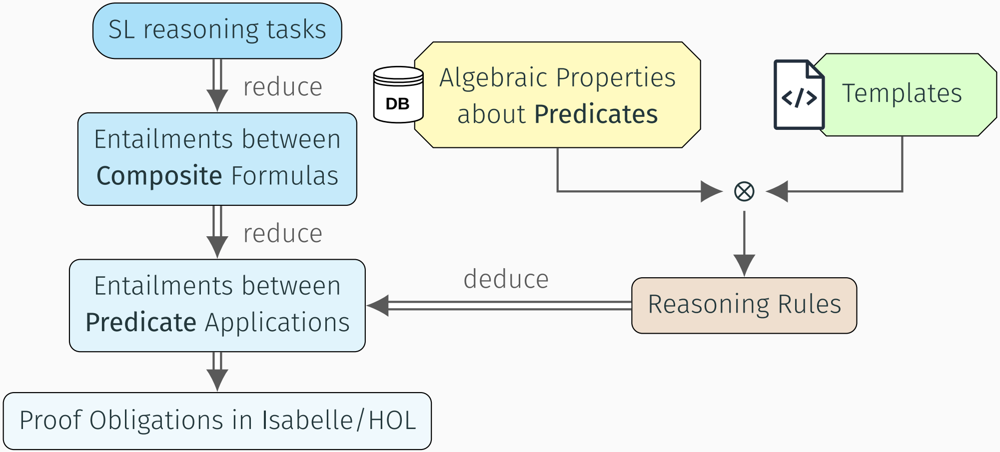
# Generating Rules Generically

Data structures are diverse. How do we generically generate their reasoning rules?

Algebraic approach:

1. Identify **fundamental** algebraic properties about **predicates**.
2. Provide reasoning rule **templates** parameterized over the algebra axioms.
3. Derive the properties of predicates compositionally.

# Reasoning Scheme



# Preliminary

An SL predicate implies a Data Refinement relation.

Predicate $\text{Array}(addr, l) \Rightarrow \begin{cases} \text{Refinement relation:} & \text{Array}_{addr} \\ \text{Abstract object:} & l \\ \text{Concrete object:} & \text{the memory heaps} \end{cases}$

Entailment $\text{Ref}(addr, v) \longrightarrow \text{Array}(addr, [v])$

$\Rightarrow$ Transformation of abstraction from one refinement relation to another

# Notations

**Notation**

$$l \fatsemi \text{Array}_{addr} \equiv \text{Array}(addr, l),$$ to distinguish $\begin{cases} \text{abstraction and} \\ \text{refinement relation.} \end{cases}$

**Notation**

$$\left(T \xrightarrow{f} U\right) \quad \equiv \quad \forall x \in \text{dom}(f).\ x \fatsemi T \longrightarrow f(x) \fatsemi U$$

$$\text{Ref}\,(addr, v) \longrightarrow \text{Array}\,(addr, [v]) \quad \equiv \quad \left(\text{Ref}_{addr} \xrightarrow{\lambda x.\ [x]} \text{Array}_{addr}\right)$$

# Functor

| | |
|---|---|
| Inspiration | $l ⨾ \mathrm{Array}_{addr}(T)$, predicate $T$ for the refinement of elements. |
| Yes | $\mathrm{Array}_{addr} : \mathrm{Predicate} \to \mathrm{Predicate}$. |
| Moreover | Transformation $\left(T \xrightarrow{f} U\right)$ is our reasoner's core. |
| Inspiration | A subtyping rule that transforms $T$ ? |

**Inspiration**
Subtyping Rule

$$\frac{\mathbb{Z} \times \mathbb{Z} \xrightarrow{\lambda(n,d).\ \frac{n}{d}} \mathbb{Q}}{\mathrm{Array}_{addr}(\mathbb{Z} \times \mathbb{Z}) \xrightarrow{\color{red}{\mathrm{map}\,(}\lambda(n,d).\ \frac{n}{d}\color{red}{)}} \mathrm{Array}_{addr}(\mathbb{Q})}$$

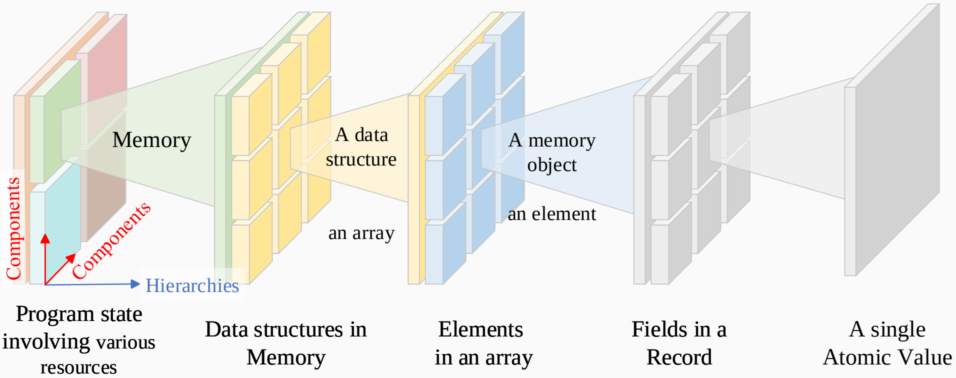
**Definition (Transformation Functor)**

$\mathrm{Functor}(F, m) \triangleq$ For any predicates $T, U$ and any $x, f$,

$$\frac{T \xrightarrow{f} U}{F(T) \xrightarrow{m(f)} F(U)}$$

Our SL specify not only memory heaps but also any concrete objects like a pair of integers

# Functor - Benefit



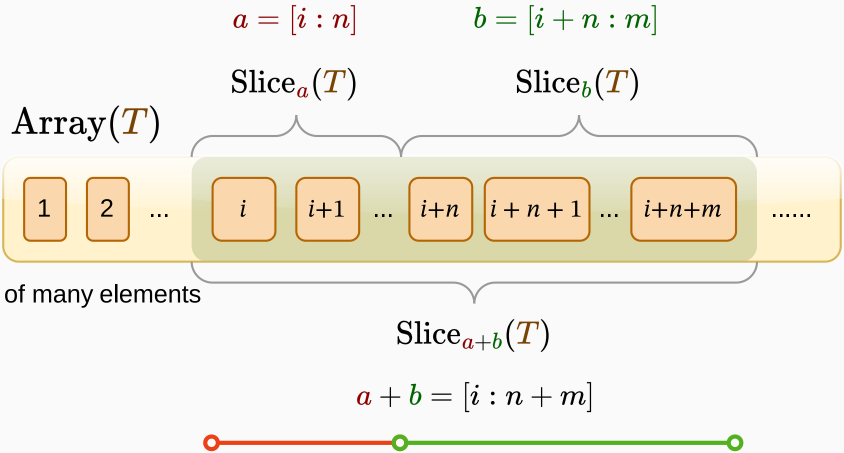

$$\frac{T \xrightarrow{f} U}{F(T) \xrightarrow{m(f)} F(U)}$$

Reduces the reasoning
from containers to their elements.

# Example: Cutting and Merging Slices

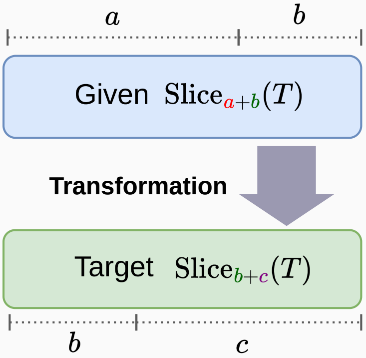
$$\mathrm{Slice}_{a+b}(T) \xrightarrow{?} \mathrm{Slice}_{b+c}(T)$$

# Example: Cutting and Merging Slices



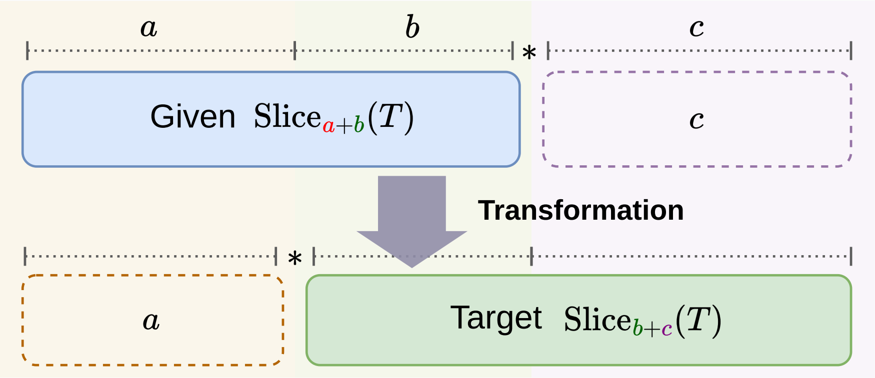
# Example: Cutting and Merging Slices

$$\mathrm{Slice}_{a+b}(T) \xrightarrow{?} \mathrm{Slice}_{b+c}(T)$$



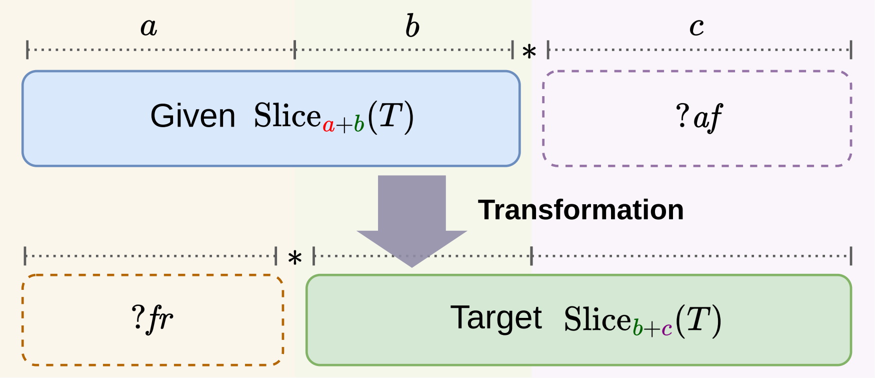
# Example: Cutting and Merging Slices

$$\text{Slice}_{a+b}(T) \xrightarrow{?} \text{Slice}_{b+c}(T)$$



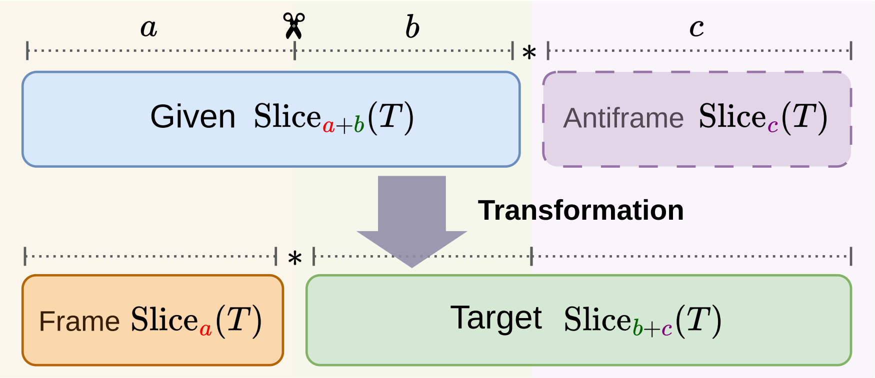
# Example: Cutting and Merging Slices

$$\text{Slice}_{a+b}(T) * ?af \xrightarrow{?} \text{Slice}_{b+c} * ?fr(T)$$



# Example: Cutting and Merging Slices

$$\text{Slice}_{a+b}(T) * \text{Slice}_c(T) \xrightarrow[]{\lambda(x_{ab}, x_c).\ \text{let } (x_a, x_b) = cut_{a,b}(x_{ab}) \text{ in } (x_a, \text{cat}_{b,c}(x_b, x_c))} \text{Slice}_{b+c}(T) * \text{Slice}_a(T)$$



# Generalized Cutting and Concatenation

Formalization

$$\mathrm{Slice}_{a+b} \underset{\mathrm{cut}_{a,b}}{\overset{\mathrm{cat}_{a,b}}{\rightleftarrows}} \mathrm{Slice}_a * \mathrm{Slice}_b$$

$$P * Q \triangleq \lambda(x, y).\ P(x) * Q(y)$$

Generalization

$$\mathrm{Distributivity}(F, cut, cat) \triangleq \left( F_{a+b}(T) \underset{cat_{a,b}}{\overset{cut_{a,b}}{\rightleftarrows}} F_a(T) * F_b(T) \right)$$

for any $a, b$ denoting domains

Why do we call it "Distributivity"?

# Modules over Rings

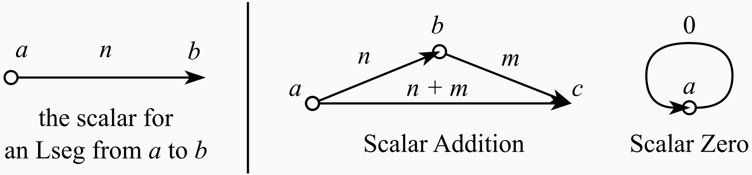
| | Usual Notation | Semimodule of Predicates |
|---|---|---|
| Partial Semiring: | $\{a, b, \cdots\}$ | The domain of $a, b$, with $+$, $\cdot$ |
| Commutative monoid: | $\{x, y, \cdots\}$ | SL predicates, with $*$ |
| Scalar multiplication: | Juxtaposition | $F : \{a, b, \cdots\} \times \text{Predicate} \to \text{Predicate}$ |

Laws

| | | |
|---|---|---|
| Distributivity: | $(a+b)x = ax + bx$ | $F_{a+b}(T) \rightleftarrows F_a(T) * F_b(T)$ |
| | $a(x \cdot y) = ax \cdot ay$ | $F_a(T * U) \rightleftarrows F_a(T) * F_a(U)$ |
| Associativity: | $(ab)m = a(bm)$ | $F_a(F_b(T)) \rightleftarrows F_{a \cdot b}(T)$ |
| Identity: | $1m = m$ | $F_1(T) \rightleftarrows T$ |
| Zero: | $0m = \varepsilon$ | $F_0(T) \rightleftarrows \text{Empty}$ |

# Example: Linked List Segment

- $l : \mathrm{Lseg}_{a \overset{n}{\circ\!\!\longrightarrow} b}$ for head address $a$, tail address $b$ and length $n$.
- Scalar defined as follows,



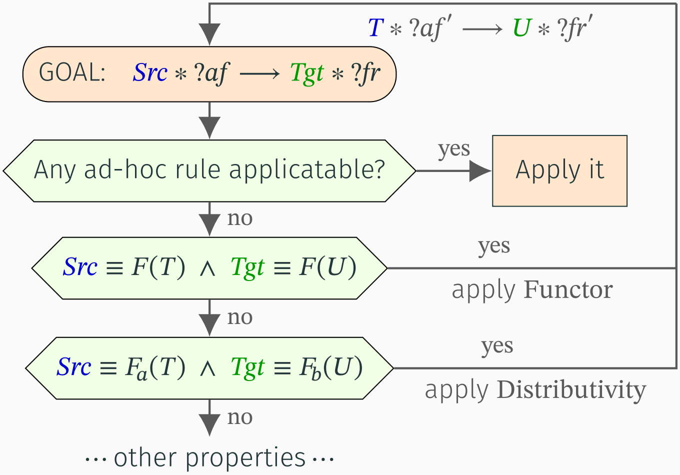

Laws

Distributivity
$$\left(\begin{array}{rl} \mathrm{Lseg}_{a \overset{n}{\circ\!\!\longrightarrow} b} * \mathrm{Lseg}_{b \overset{m}{\circ\!\!\longrightarrow} c} & \xrightarrow{\text{cat}} \mathrm{Lseg}_{a \overset{n+m}{\circ\!\!\longrightarrow} c} \\ \exists b.\ \mathrm{Lseg}_{a \overset{n}{\circ\!\!\longrightarrow} b} * \mathrm{Lseg}_{b \overset{m}{\circ\!\!\longrightarrow} c} & \xleftarrow{\text{cut}} \mathrm{Lseg}_{a \overset{n+m}{\circ\!\!\longrightarrow} c} \end{array}\right)$$

Identity $\quad \mathrm{Lseg}_{a \overset{1}{\circ\!\!\longrightarrow} b} \rightleftarrows \mathrm{Node}_{a,b}$

Zero $\quad \mathrm{Lseg}_{a \overset{0}{\circ\!\!\longrightarrow} a} \rightleftarrows \mathrm{Empty}$

# Sketch of Algebra-driven Reasoner



# Contents

1. Identify **fundamental** algebraic properties.
2. Provide reasoning rule **templates** parameterized by the properties.
3. Derive the properties of predicates compositionally.

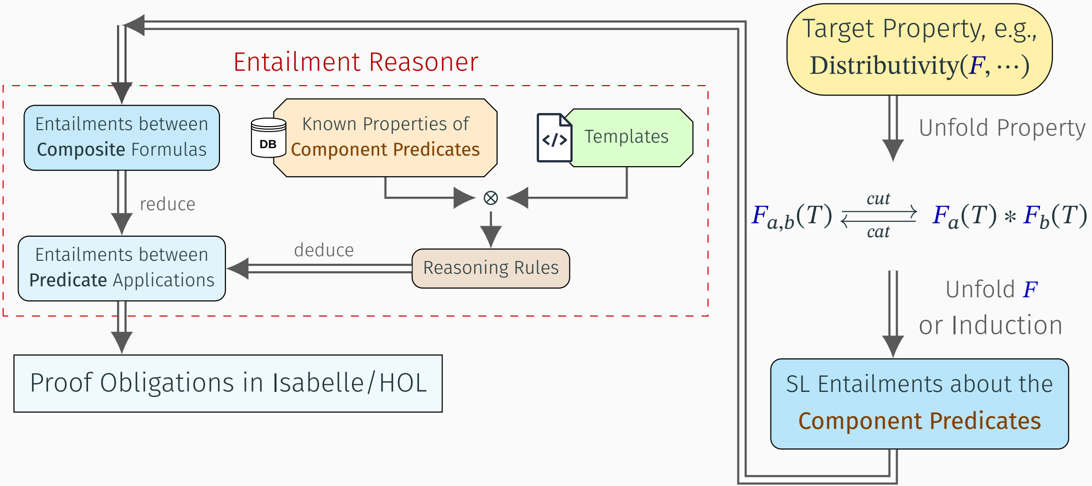
# Compositionally Deriving Properties

- Predicates are built compositionally

$$T(x) \triangleq \cdots \; y_1 \mathbin{⨟} U_1 \; * \; y_2 \mathbin{⨟} U_2 \; \cdots \; y_3 \mathbin{⨟} U_3 \; \vee \; y_4 \mathbin{⨟} U_4 \; \cdots$$

- Derive properties from known properties of their components
  - without unfolding the components

# Compositionally Deriving Properties



# Experiments

| Case | Manual.R | Anot | Fold | Othr | Ovh | LoC | Time | … |
|---|---|---|---|---|---|---|---|---|
| Link-List | 0 | 0.19 | 0.19 | 0 | 0.24 | 67 | 0.3s + 0.7min + 8s | |
| Quicksort | 0 | 0.39 | 0 | 0 | 0.72 | 18 | 0.5s + 3.4min + 50s | |
| Dynamic Array | 0 | 0.19 | 0.18 | 0 | 0.24 | 62 | 2.5s + 3.1min + 53s | |
| Strassen Matrix | 2 | 0.30 | 0.11 | 0.11 | 0.62 | 104 | 3.0s + 4.2min + 67s | … |
| AVL Tree | 0 | 0.31 | 0.31 | 0 | 1.30 | 152 | 7.5s + 9.8min + 223s | |
| Bucket Hash | 0 | 0.31 | 0.09 | 0.04 | 0.51 | 113 | 3.7s + 6.3min + 23s | |
| … | | | | … … | | | | |

Unfolding is still necessary for revealing internal data representations.

# Summary

- Functor ✅
- Separating Homomorphism
- Modules over Rings
  - Distributivity ✅
  - Associativity
  - Identity
  - Zero

https://github.com/xqyww123/phi-system

xu@qiyuan.me

# Future Works

- More Automation
  - The automation of FOL proof obligations
  - Neural Theorem Proving?
- Stronger Expressiveness
  - The natural definitions of predicates do not allow SepHom
  - Fictional Separation
  - A fictional modality $\approx$ Iris, the higher-order ghost SL.

# Thanks for Your Attention

https://github.com/xqyww123/phi-system

# Separating Homomorphism (SepHom)

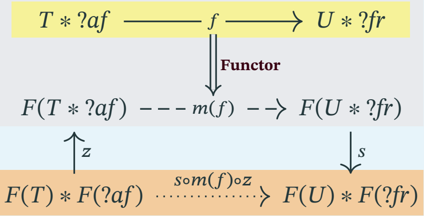
Functor + SepHom reduces *bi-abductive* reasoning from containers to their elements.

**Definition (Separating Homomorphism)**

$$\begin{aligned} \mathrm{SepHom}(F,s,z) \triangleq \forall T,U.\quad & \forall x.\ x \mathbin{⨾} F(T*U) \longrightarrow s(x) \mathbin{⨾} F(T)*F(U) \\ \wedge\ & \forall x.\ x \mathbin{⨾} F(T)*F(U) \longrightarrow z(x) \mathbin{⨾} F(T*U) \end{aligned}$$

where $(T*U)(a,b) \triangleq T(a)*U(b)$ is predicate separating conjunction.

**bi-Abductive Reasoning over Functors**




If $\mathrm{Functor}(F,m,d)$ and $\mathrm{SepHom}(F,s,z)$ hold,

$$\frac{\forall a \in d(z(x)).\ a \mathbin{⨾} T * ?af \longrightarrow f(a) \mathbin{⨾} U * ?fr}{x \mathbin{⨾} F(T) * F(?af) \longrightarrow g(a) \mathbin{⨾} F(U) * F(?fr)}$$

where $g = (s \circ m(f) \circ z)$

# Separating Homomorphism (SepHom)

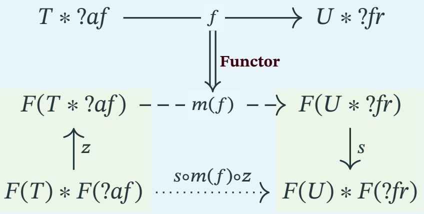
Functor + SepHom reduces *bi-abductive* reasoning from containers to their elements.

**Definition (Separating Homomorphism)**

$$\mathrm{SepHom}(F, s, z) \triangleq \forall T, U. \quad \forall x.\ x \mathbin{⨾} F(T * U) \longrightarrow s(x) \mathbin{⨾} F(T) * F(U) \\ \wedge \forall x.\ x \mathbin{⨾} F(T) * F(U) \longrightarrow z(x) \mathbin{⨾} F(T * U)$$

where $(T * U)(a, b) \triangleq T(a) * U(b)$ is predicate separating conjunction.

**bi-Abductive Reasoning over Functors**




**Correspondence in Category Theory**

Functor + SepHom ⇒ lax monoidal functor taking the predicate separating conjunction as the tensor operator.

# Separating Homomorphism - Example

### Example - Records

Assume $\text{Field}_a(T)$ denote the type of a field named $a$ and its value has type $T$.

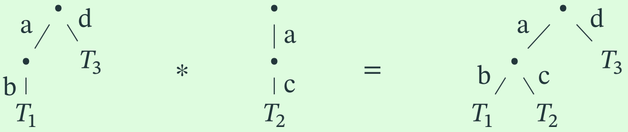
**Notation**. $\{\mathrm{a}\colon T\} \equiv \text{Field}_a(T)$

Use separation $(*)$ to conjunct record fields.

**Notation**. $\{\mathrm{a}\colon T,\ \mathrm{b}\colon U\} \equiv \{\mathrm{a}\colon T\} * \{\mathrm{b}\colon U\} \equiv \text{Field}_a(T) * \text{Field}_b(U)$.

We know a nested record forms a tree labeled by field names.

$$\{\mathrm{a}\colon \{\mathrm{b}\colon T_1\},\ \mathrm{d}\colon T_3\} \quad * \quad \{\mathrm{a}\colon \{\mathrm{c}\colon T_2\}\} \quad = \quad \{\mathrm{a}\colon \{\mathrm{b}\colon T_1,\ \mathrm{c}\colon T_2\},\ \mathrm{d}\colon T_3\}$$




SepHom: $\{\mathrm{a}\colon \{\mathrm{b}\colon T_1\}\} * \{\mathrm{a}\colon \{\mathrm{c}\colon T_2\}\} = \{\mathrm{a}\colon \{\mathrm{b}\colon T_1\} * \{\mathrm{c}\colon T_2\}\} \equiv \{\mathrm{a}\colon \{\mathrm{b}\colon T_1, \mathrm{c}\colon T_2\}\}$.

# Separating Homomorphism - More Examples

**Examples**

- References to records.
  Assume $\text{Ref}_{addr}(T)$ is the type for a reference to a memory object at address *addr* and this memory object has type $T$. On certain memory model,
  $$\text{Ref}_{addr}(T * U) \longleftrightarrow \text{Ref}_{addr}(T) * \text{Ref}_{addr}(U)$$
- Arrays as a sequence of memory objects,
  $$\text{Array}(T * U) \longleftrightarrow \text{Array}(T) * \text{Array}(U)$$
- For more advanced data structures, sadly, fictional separation is required.

* Symbol ($\longleftrightarrow$) denotes existing a forward and a backward transformation.

# Modules over Rings - More Examples

- $\{\mathrm{a}: \{\mathrm{b}: T\}\} \rightleftarrows \{\mathrm{a.b}: T\}$ allows one rule

$$\{x \,⨾\, \mathrm{Ref}_{addr}\{\mathit{field}: T\}\}\ \mathtt{load}(\,addr.\mathit{field}\,)\,\{\cdots\}$$

  to access any field at any deep level,e.g., to access $addr.\mathrm{a.b}$ in $addr \mapsto \{\mathrm{a}: \{\mathrm{b}: T_1, \mathrm{c}: T_2\}, \mathrm{d}: T_3\}$, where we instantiate $\mathit{field}$ to $\mathrm{a.b}$.

- Permission modality $(s \oplus T)$

$$s \oplus (t \oplus T) \rightleftharpoons (s \cdot t) \oplus T \qquad (s+t) \oplus T \rightleftharpoons (s \oplus T) * (t \oplus T) \qquad 1 \oplus T \rightleftharpoons T \qquad 0 \oplus T \rightleftharpoons \mathrm{Empty}$$

- Separating quantifier $\mathop{\Huge *}_{i \in \{1,\cdots,n\}} T_i \triangleq T_1 * \cdots * T_n$ whose scalar is its domain.

$$\mathop{\Huge *}_{i \in s \uplus t} T_i \rightleftharpoons \mathop{\Huge *}_{i \in s} T_i * \mathop{\Huge *}_{i \in t} T_i \qquad \mathop{\Huge *}_{i \in s} \mathop{\Huge *}_{j \in t} T_{i,j} \rightleftharpoons \mathop{\Huge *}_{(i,j) \in s \times t} T_{i,j}$$