Composite Abstract Domains for Shape Analysis

Antoine Toubhans

ECOLE NORMALE SUPERIEURE

Inria — INVENTEURS DU MONDE NUMÉRIQUE

Séminaire Gallium

October 14th, 2014

The Problem

Programs from real world often manipulate **many data structures**

- They may be **heterogeneous**
  e.g. **lists**, **trees**, **arrays**, **strings**...
- They may be **more or less complex**
  e.g. **trees**, **BST**, **B-trees**, **Red/Black trees** ...
- They may have **complex interactions**
  e.g. be **nested**, be **overlaid**, **share values** ...

The Problem

Programs from real world often manipulate many data structures

- They may be heterogeneous
  e.g. lists, trees, arrays, strings...
- They may be more or less complex
  e.g. trees, BST, B-trees, Red/Black trees ...
- They may have complex interactions
  e.g. be nested, be overlaid, share values ...

There exist analyses for most data structures
$\Longrightarrow$ How to *combine* these into a new more expressive analysis

Outline

Outline

The MemCAD analyzer (Memory Compositional Abstract Domains)

- targets C programs manipulating complex memory states
    - **complex nested/overlaid/heterogeneous data structures**

The **MemCAD** analyzer (**Mem**ory **C**ompositional **A**bstract **D**omains)

- targets C programs manipulating **complex memory states**
  - **complex nested/overlaid/heterogeneous data structures**
- proves **safety (numeric) properties**, e.g.
  - the **absence** of **division by zero**
  - the **absence** of **out of bound** array access
  - the **absence** of **arithmetic overflow**
  - ...

The MemCAD analyzer (Memory Compositional Abstract Domains)

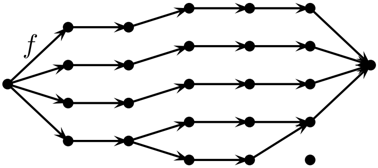
- targets C programs manipulating **complex memory states**
  - **complex nested/overlaid/heterogeneous data structures**
- proves **safety (numeric) properties**, e.g.
  - the **absence** of **division by zero**
  - the **absence** of **out of bound** array access
  - the **absence** of **arithmetic overflow**
  - ...
- proves **safety (memory) properties**, e.g.
  - the **absence** of **null pointer dereference**
  - the **absence** of **memory leak**
  - the **absence** of **incorrect** memory freeing
  - ...

The **MemCAD** analyzer (**Mem**ory **C**ompositional **A**bstract **D**omains)

- targets C programs manipulating **complex memory states**
  - **complex nested/overlaid/heterogeneous data structures**
- proves **safety (numeric) properties**, e.g.
  - the **absence** of **division by zero**
  - the **absence** of **out of bound** array access
  - the **absence** of **arithmetic overflow**
  - ...
- proves **safety (memory) properties**, e.g.
  - the **absence** of **null pointer dereference**
  - the **absence** of **memory leak**
  - the **absence** of **incorrect** memory freeing
  - ...
- **automatically** infers **shape/numeric invariant**, e.g.
  - "`i` is an **even** integer"
  - "`l` points to a **linked list**"
  - "`l` points to a **linked list** of **even** integers"

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.



Concrete Memory States

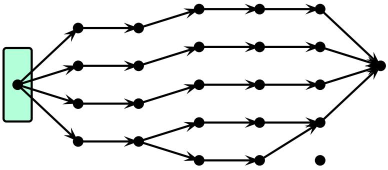
Concrete memory state $s \in \mathbb{S}$

Concrete Transfer Functions

$f : \mathbb{S} \rightarrow \mathcal{P}(\mathbb{S})$

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.



Abstract Memory States

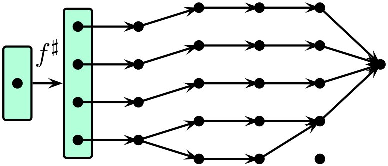
Abstract memory state $s^\sharp \in \mathrm{S}^\sharp$

Concretization Function

$$\gamma : \mathrm{S}^\sharp \to \mathcal{P}(\mathrm{S})$$

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.

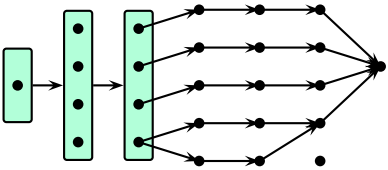


Abstract Transfer Functions

$$f^\sharp : \mathrm{S}^\sharp \rightarrow \mathrm{S}^\sharp$$

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.

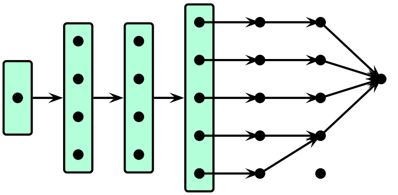


That are Sound...

$$\forall s \in \gamma(s^\sharp),\ f(s) \subseteq \gamma \circ f^\sharp(s^\sharp)$$

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.

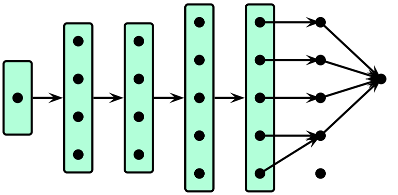


That are Sound...

$$\forall s \in \gamma(s^\sharp),\ f(s) \subseteq \gamma \circ f^\sharp(s^\sharp)$$

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.

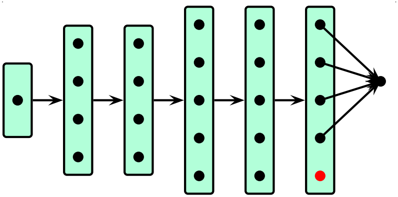


That are Sound...

$$\forall s \in \gamma(s^\sharp),\ f(s) \subseteq \gamma \circ f^\sharp(s^\sharp)$$

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.

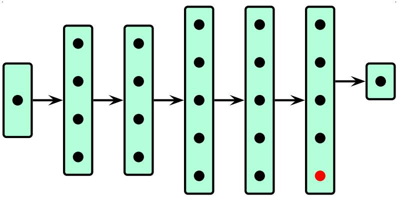


But not (necessarily) Complete...

$$\cup\{f(s) \mid s \in \gamma(s^\sharp)\} \not\supseteq \gamma \circ f^\sharp(s^\sharp)$$

The Abstract Interpretation Framework

A **Theory** for computing an over-approximation of semantics of programs.



But not (necessarily) Complete...

$$\cup\{f(s) \mid s \in \gamma(s^\sharp)\} \not\supseteq \gamma \circ f^\sharp(s^\sharp)$$

Concrete Memory States

- **Memory State** : environment + memory, i.e.

$$\mathrm{S} = \mathrm{E} \times \mathrm{M} \qquad\qquad \mathrm{S} \ni s = (e, m)$$

- **Values** : $\mathrm{V} \supset \mathrm{V}_{\mathrm{addr}}$
- **Environments** : maps program variables to addresses, i.e.

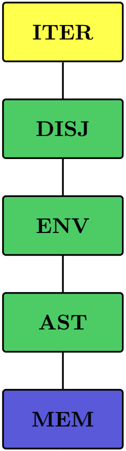
$$\mathrm{E} = \mathrm{X} \rightarrow \mathrm{V}_{\mathrm{addr}}$$

- **Memories** : maps addresses to values

$$\begin{array}{rcl} \textbf{read} & : & \mathrm{V}_{\mathrm{addr}} \times \mathsf{Size} \times \mathrm{M} \rightarrow \mathrm{V} \\ \textbf{write} & : & \mathrm{V}_{\mathrm{addr}} \times \mathsf{Size} \times \mathrm{V} \times \mathrm{M} \rightarrow \mathrm{M} \\ \textbf{alloc} & : & \mathsf{Size} \times \mathrm{M} \rightarrow \mathrm{V}_{\mathrm{addr}} \times \mathrm{M} \\ \ldots & & \end{array}$$

MemCAD is modular

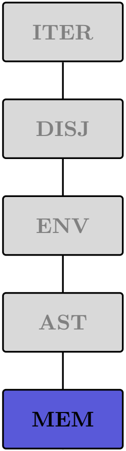
Layers of Abstract Domains :



- Each box is an **abstract domain** with
  - its concretization function
  - its abstract transfer functions
  - implemented as OCaml modules
- Edges are **Functors**
  - implemented as OCaml functors
  - offers **modularity**

MemCAD is modular

Layers of Abstract Domains :

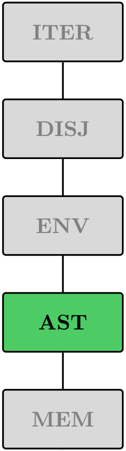


Memory Abstract Domain $\mathrm{M}^\sharp$

- abstract memories $m^\sharp \in \mathrm{M}^\sharp$
- consists of predicates **quantified** on **symbolic variables**
- **symbolic variables** denoted by Greek letters $\alpha, \beta, \ldots \in \mathrm{V}^\sharp$, **represents concrete values**
- valuations $\nu \in \mathrm{Val} = \mathrm{V}^\sharp \rightarrow \mathrm{V}$
- concretization $\gamma_{\mathrm{M}^\sharp} : \mathrm{M}^\sharp \rightarrow \mathcal{P}(\mathrm{Val} \times \mathrm{M})$
- **simple** abstract transfer functions

MemCAD is modular

Layers of Abstract Domains :

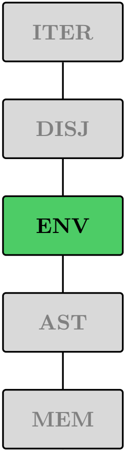


Program Expressions Evaluation $\mathrm{M}^{\sharp}_{\mathrm{ast}}$

- same abstract memories
- more complex abstract transfer functions that involves expressions with memory operation e.g. $(\star\alpha) \cdot \texttt{next}$

MemCAD is modular

Layers of Abstract Domains :



Abstract States with Environment $\mathrm{S}^\sharp$

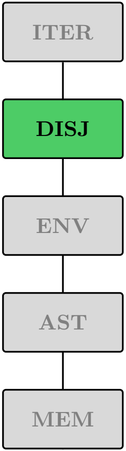
- abstract states are abstract memories with **abstract environment**, i.e. $\mathrm{S}^\sharp = \mathrm{E}^\sharp \times \mathrm{M}^\sharp$
- abstract environments $\mathrm{E}^\sharp = \mathrm{X} \rightarrow \mathrm{V}^\sharp$ map program variables to symbolic variables **representing their addresses**
- concretization $\gamma_{\mathrm{S}^\sharp} : \mathrm{S}^\sharp \rightarrow \mathcal{P}(\mathrm{S})$

$$\gamma_{\mathrm{S}^\sharp}(e^\sharp, m^\sharp) \stackrel{\text{def}}{=} \{(\nu \circ e^\sharp, m) \mid (\nu, m) \in \gamma_{\mathrm{M}^\sharp}(m^\sharp)\}$$

- abstract transfer functions that involves **program expressions** e.g. $(\star\mathrm{x}) \cdot \mathtt{next}$

MemCAD is modular

Layers of Abstract Domains :



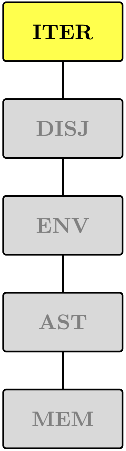
Disjunctive Abstract Domain $\mathrm{S}^{\sharp}_{\vee}$

- $\mathrm{S}^{\sharp}_{\vee} = \mathcal{P}_{\mathrm{fin}}(\mathrm{S}^{\sharp})$
- concretization $\gamma_{\vee} : \mathrm{S}^{\sharp}_{\vee} \rightarrow \mathcal{P}(\mathrm{S})$

$$\gamma_{\vee}(s^{\sharp}_{\vee}) \stackrel{\mathsf{def}}{=} \cup\{\gamma_{\mathrm{S}^{\sharp}}(s^{\sharp}) \mid s^{\sharp} \in s^{\sharp}_{\vee}\}$$

MemCAD is modular

Layers of Abstract Domains :



Fixed-Point engine

- Iterates over the control flow graph

What is (so far) implemented in MemCAD ?

Basic Memory Abstract Domains :

- the **Bounded Memory Abstract Domain** $\mathrm{M}_{\mathrm{b}}^{\sharp}$
  - handles set of **spatially-bounded** memories
- the **List Memory Abstract Domain** $\mathrm{M}_{\mathrm{lst}}^{\sharp}$
  - handles **linear, linked-list-like** data structures
- the **Separating Shape Graphs Domain** $\mathrm{M}_{\mathrm{ssg}}^{\sharp}$
  - handles **more complex data structures**
  - relies on **user-provided inductive definitions**

What is (so far) implemented in MemCAD ?

Combination of Memory Abstract Domains :

- a functor that add **numerical constraints** to memory abstractions
  - constraints hold on symbolic variables
  - APRON library (Intervals/Octogons/Polyhedra + Disequalities)
- the **Separating Product** of two memory abstract domains
- the **Reduced Product** of two memory abstract domains

Outline

The bounded memory abstract domain : $\mathrm{M}^\sharp_\mathrm{b}$

- **Points-to** predicates $\alpha \boxed{\beta}$ representing cells
- **Combination** with a **numerical abstract domain**
- **No summarization**

- **An abstract state** $(e^\sharp, m^\sharp_\mathrm{b}) \in \mathrm{E}^\sharp \times \mathrm{M}^\sharp_\mathrm{b}$:

$$\left(\begin{array}{rcl} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array} , \quad \begin{array}{l} \alpha_0 \boxed{\beta_0} \\ \alpha_1 \boxed{\beta_1} \\ \alpha_2 \boxed{\beta_2} \\ \alpha_3 \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \mathtt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

The bounded memory abstract domain : $\mathrm{M}^\sharp_{\mathrm{b}}$

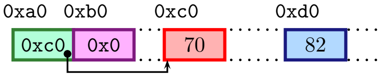
- **Points-to** predicates $\alpha \; \boxed{\beta}$ representing cells
- **Combination** with a **numerical abstract domain**
- **No summarization**

- **An abstract state** $(e^\sharp, m^\sharp_{\mathrm{b}}) \in \mathrm{E}^\sharp \times \mathrm{M}^\sharp_{\mathrm{b}}$:

$$\left(\begin{array}{rcl} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array} \;,\; \begin{array}{l} \alpha_0 \; \boxed{\beta_0} \\ \alpha_1 \; \boxed{\beta_1} \\ \alpha_2 \; \boxed{\beta_2} \\ \alpha_3 \; \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \mathtt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

- **Abstracting** the **concrete state** $(e, m) \in \gamma(e^\sharp, m^\sharp_{\mathrm{b}})$:

$$\left(\begin{array}{rcl} \mathtt{p0} & \mapsto & \mathtt{0xa0} \\ \mathtt{p1} & \mapsto & \mathtt{0xb0} \\ \mathtt{i} & \mapsto & \mathtt{0xd0} \end{array} \;, \right)$$

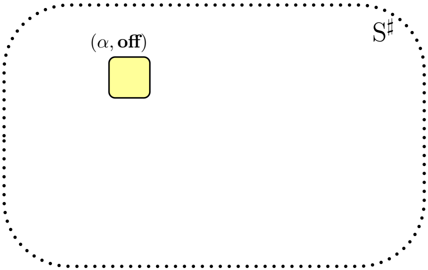


A bit of static analysis

To compute a post-condition of an assignment, the analysis :

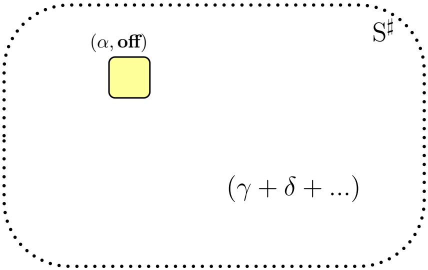
To compute a post-condition of an assignment, the analysis :

1. evaluates l.h.s. to a symbolic variable (plus an offset)



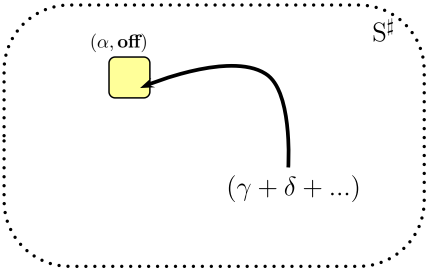
To compute a post-condition of an assignment, the analysis :

1. evaluates l.h.s. to a symbolic variable (plus an offset)
2. evaluates r.h.s. to a numerical expression of symbolic variables

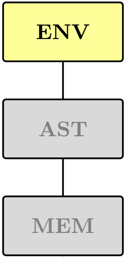


To compute a post-condition of an assignment, the analysis :

1. **evaluates** l.h.s. to a symbolic variable (plus an offset)
2. **evaluates** r.h.s. to a numerical expression of symbolic variables
3. **writes** the cell at the abstract level



Assignment : $\texttt{i} = \texttt{i} + \star\texttt{p0};$



pre

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} \; , \; \begin{array}{c} \alpha_0 \; \boxed{\beta_0} \\ \alpha_1 \; \boxed{\beta_1} \\ \alpha_2 \; \boxed{\beta_2} \\ \alpha_3 \; \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\, \beta_0 = \alpha_2 \;\wedge\; \beta_1 = \texttt{0x0} \\ \wedge\, \beta_2 \in [0;100] \;\wedge\; \beta_3 > \beta_2 \end{array} \right)$$

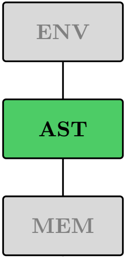
post

$$\left(\quad , \quad \right)$$

Status

Replace program variables by symbolic variables

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{lcl} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array}, \begin{array}{l} \alpha_0\ \boxed{\beta_0} \\ \alpha_1\ \boxed{\beta_1} \\ \alpha_2\ \boxed{\beta_2} \\ \alpha_3\ \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2\ \wedge\ \beta_1 = \mathtt{0x0} \\ \wedge\ \beta_2 \in [0; 100]\ \wedge\ \beta_3 > \beta_2 \end{array}\right)$$

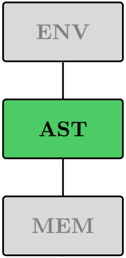
post

$$\left(\begin{array}{lcl} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array}, \qquad \right)$$

Status

Evaluating left hand side α_3

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{lcl} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} ,\ \begin{array}{ll} \alpha_0 & \boxed{\beta_0} \\ \alpha_1 & \boxed{\beta_1} \\ \alpha_2 & \boxed{\beta_2} \\ \alpha_3 & \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2\ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0; 100]\ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

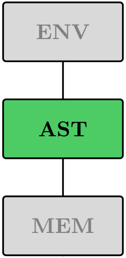
post

$$\left(\begin{array}{lcl} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \qquad \right)$$

Status

Left hand side evaluated to α_3

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{cc} \alpha_0 & \boxed{\beta_0} \\ \alpha_1 & \boxed{\beta_1} \\ \alpha_2 & \boxed{\beta_2} \\ \alpha_3 & \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0;100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

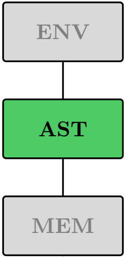
post

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \right)$$

Status

Evaluating right hand side $\alpha_3 + \star\alpha_0$

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{ccc} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array}, \begin{array}{c} \alpha_0 \boxed{\beta_0} \\ \alpha_1 \boxed{\beta_1} \\ \alpha_2 \boxed{\beta_2} \\ \alpha_3 \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \mathtt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3 > \beta_2 \end{array}\right)$$

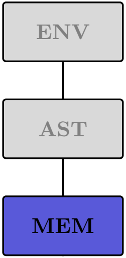
post

$$\left(\begin{array}{ccc} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array}, \qquad \right)$$

Status

Evaluating right hand side α_3

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



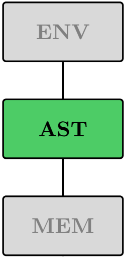
pre
$$\left(\begin{array}{lcl} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array}\right., \quad \begin{array}{l} \alpha_0 \boxed{\beta_0} \\ \alpha_1 \boxed{\beta_1} \\ \alpha_2 \boxed{\beta_2} \\ \alpha_3 \boxed{\beta_3} \end{array} \quad \left.\begin{array}{l} \wedge\ \beta_0 = \alpha_2\ \wedge\ \beta_1 = \mathtt{0x0} \\ \wedge\ \beta_2 \in [0;100]\ \wedge\ \beta_3 > \beta_2 \end{array}\right)$$

post
$$\left(\begin{array}{lcl} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array}\quad , \qquad\qquad \right)$$

Status

Evaluating right hand side α_3

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



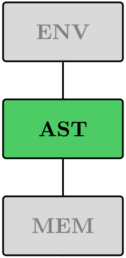
pre $\left(\begin{array}{lcl} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{l} \alpha_0 \boxed{\beta_0} \\ \alpha_1 \boxed{\beta_1} \\ \alpha_2 \boxed{\beta_2} \\ \alpha_3 \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2\ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0; 100]\ \wedge\ \beta_3 > \beta_2 \end{array} \right)$

post $\left(\begin{array}{lcl} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \right)$

Status

Right hand side α_3 evaluated to β_3

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{ccc} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{c} \alpha_0 \; \boxed{\beta_0} \\ \alpha_1 \; \boxed{\beta_1} \\ \alpha_2 \; \boxed{\beta_2} \\ \alpha_3 \; \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \mathtt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

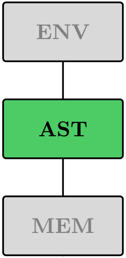
post

$$\left(\begin{array}{ccc} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array} , \qquad \right)$$

Status

Evaluating right hand side $\star\alpha_0$

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



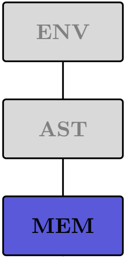
pre
$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{cc} \alpha_0 & \boxed{\beta_0} \\ \alpha_1 & \boxed{\beta_1} \\ \alpha_2 & \boxed{\beta_2} \\ \alpha_3 & \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0;100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

post
$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \qquad \right)$$

Status

Evaluating right hand side α_0

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{rcl} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array},\ \begin{array}{l} \alpha_0\ \boxed{\beta_0} \\ \alpha_1\ \boxed{\beta_1} \\ \alpha_2\ \boxed{\beta_2} \\ \alpha_3\ \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2\ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0; 100]\ \wedge\ \beta_3 > \beta_2 \end{array}\right)$$

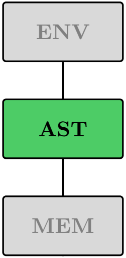
post

$$\left(\begin{array}{rcl} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array},\ \right)$$

Status

Evaluating right hand side α_0

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



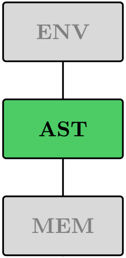
pre
$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{c} \alpha_0 \boxed{\beta_0} \\ \alpha_1 \boxed{\beta_1} \\ \alpha_2 \boxed{\beta_2} \\ \alpha_3 \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

post
$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \qquad \qquad \right)$$

Status

Right hand side α_0 evaluated to β_0

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{c} \alpha_0 \boxed{\beta_0} \\ \alpha_1 \boxed{\beta_1} \\ \alpha_2 \boxed{\beta_2} \\ \alpha_3 \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0;100] \ \wedge\ \beta_3 > \beta_2 \end{array}\right)$$

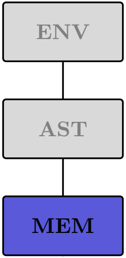
post

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \qquad \right)$$

Status

Evaluating right hand side β_0

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{ccc} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{c} \alpha_0 \ \boxed{\beta_0} \\ \alpha_1 \ \boxed{\beta_1} \\ \alpha_2 \ \boxed{\beta_2} \\ \alpha_3 \ \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \mathtt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

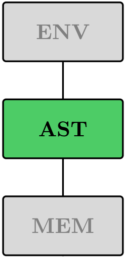
post

$$\left(\begin{array}{ccc} \mathtt{p0} & \mapsto & \alpha_0 \\ \mathtt{p1} & \mapsto & \alpha_1 \\ \mathtt{i} & \mapsto & \alpha_3 \end{array} , \qquad \right)$$

Status

Evaluating right hand side β_0

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



pre

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} \;,\; \begin{array}{l} \alpha_0 \boxed{\beta_0} \\ \alpha_1 \boxed{\beta_1} \\ \alpha_2 \boxed{\beta_2} \\ \alpha_3 \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0;100] \ \wedge\ \beta_3 > \beta_2 \end{array}\right)$$

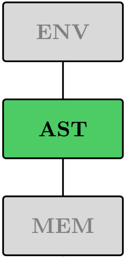
post

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} \;,\; \right)$$

Status

Right hand side β_0 evaluated to β_2

Assignment : $\alpha_3 = \alpha_3 + \star\alpha_0;$



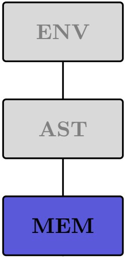
pre
$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} ,\ \begin{array}{cc} \alpha_0 & \boxed{\beta_0} \\ \alpha_1 & \boxed{\beta_1} \\ \alpha_2 & \boxed{\beta_2} \\ \alpha_3 & \boxed{\beta_3} \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2\ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0;100]\ \wedge\ \beta_3 > \beta_2 \end{array}\right)$$

post
$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} ,\ \right)$$

Status

Right hand side $\alpha_3 + \star\alpha_0$ evaluated to $\beta_3 + \beta_2$

Assignment : $\alpha_3 \leftarrow \beta_3 + \beta_2;$



pre

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{cc} \alpha_0 & \beta_0 \\ \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3 > \beta_2 \end{array} \right)$$

post

$$\left(\begin{array}{ccc} \texttt{p0} & \mapsto & \alpha_0 \\ \texttt{p1} & \mapsto & \alpha_1 \\ \texttt{i} & \mapsto & \alpha_3 \end{array} , \begin{array}{cc} \alpha_0 & \beta_0 \\ \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3' \end{array} \quad \begin{array}{l} \wedge\ \beta_0 = \alpha_2 \ \wedge\ \beta_1 = \texttt{0x0} \\ \wedge\ \beta_2 \in [0; 100] \ \wedge\ \beta_3' > 2 * \beta_2 \end{array} \right)$$

Status

Perform the assignment

Static Analysis

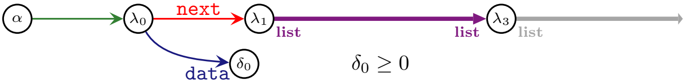
Abstract domains are also provided with :

- **abstract transfer functions** for **creating/removing** new memory blocks
- **guard operation** for **branching**
- **inclusion test/join** for **loop invariant**
- **widening** for **ensuring termination**

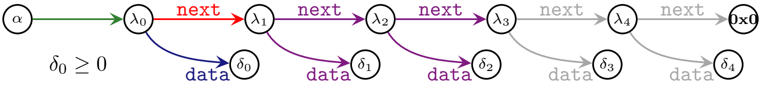
Separating shape graphs with inductive definitions : $\mathbb{M}^\sharp_{\mathrm{ssg}}$

- **Shape graphs** with **points-to** edges, and **inductive** edges
- **Nodes** denote **concrete values**, **edges** denote **memory regions**
- **Summarization**, using **inductive** definitions

- **A separating shape graph** $m^\sharp_{\mathrm{s}} \in \mathbb{M}^\sharp_{\mathrm{ssg}}$:



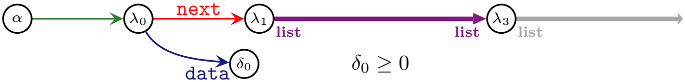
- **An unfolded graph** (partial concretization) :



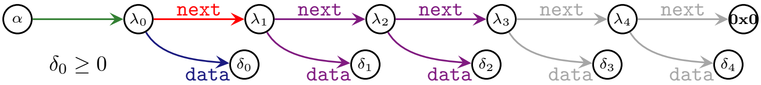
Separating shape graphs with inductive definitions : $\mathrm{M}^{\sharp}_{\mathrm{ssg}}$

- **Shape graphs** with **points-to** edges, and **inductive** edges
- **Nodes** denote **concrete values**, **edges** denote **memory regions**
- **Summarization**, using **inductive** definitions

- **A separating shape graph** $m^{\sharp}_{\mathrm{s}} \in \mathrm{M}^{\sharp}_{\mathrm{ssg}}$:

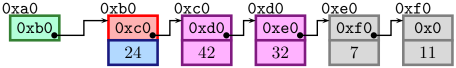


- **An unfolded graph** (partial concretization) :



- **Abstracting** the **concrete memory** $(\nu, m) \in \gamma(m^{\sharp}_{\mathrm{s}})$:

$$\left(\begin{array}{ccc} \alpha & \mapsto & \mathtt{0xa0} \\ \lambda_0 & \mapsto & \mathtt{0xb0} \\ \ldots & & \end{array} \right. ,$$



$\Big)$

Outline

Separating product : Insight

1. In many cases, programs manipulate memories with **completely different** data structures in **disjoint** memory regions.
2. **There exist memory abstractions** handling each of these data structures
3. **There is no memory abstraction** handling all of them

Separating product : Insight

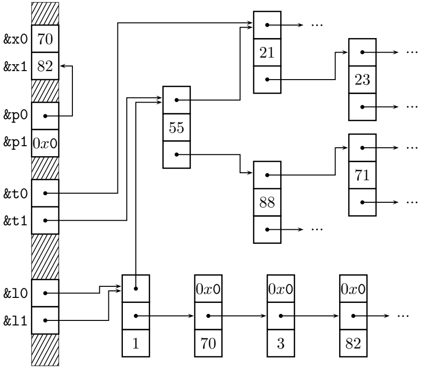
1. In many cases, programs manipulate memories with **completely different** data structures in **disjoint** memory regions.
2. **There exist memory abstractions** handling each of these data structures
3. **There is no memory abstraction** handling all of them

> Apply existing abstractions to **disjoint** part of the memory
> and **glue** them together

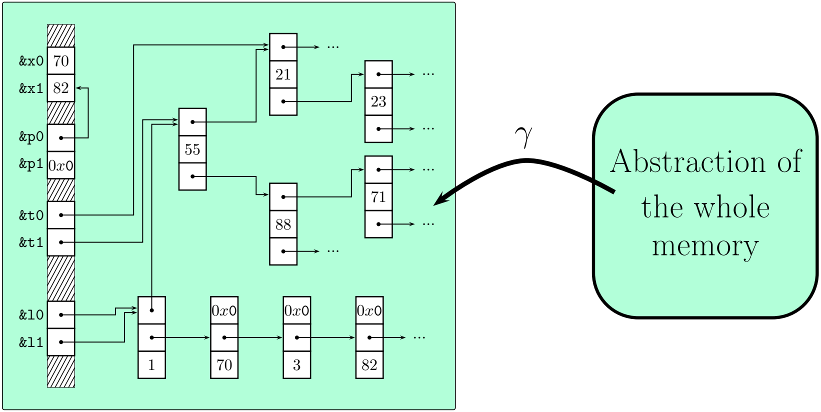
A concrete memory with a list and a tree

C struct

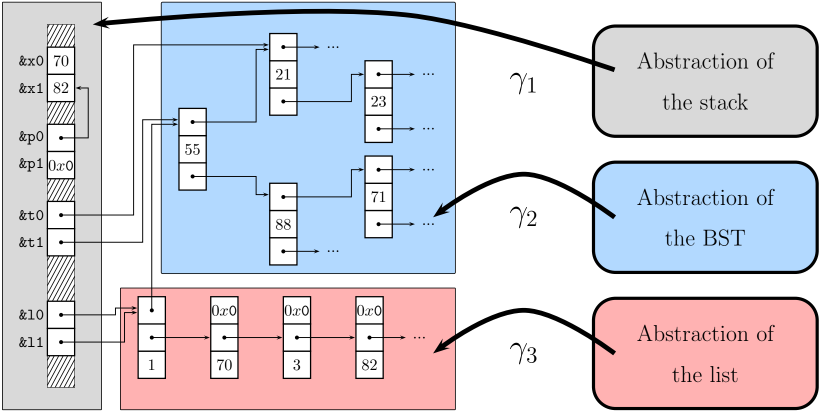
```
struct Tree {
    struct Tree *lft, *rgt;
    int data;
};
struct List {
    struct Tree *tree;
    struct List *next;
    int data;
};
```



A concrete memory with a list and a tree



A concrete memory with a list and a tree



Discussion

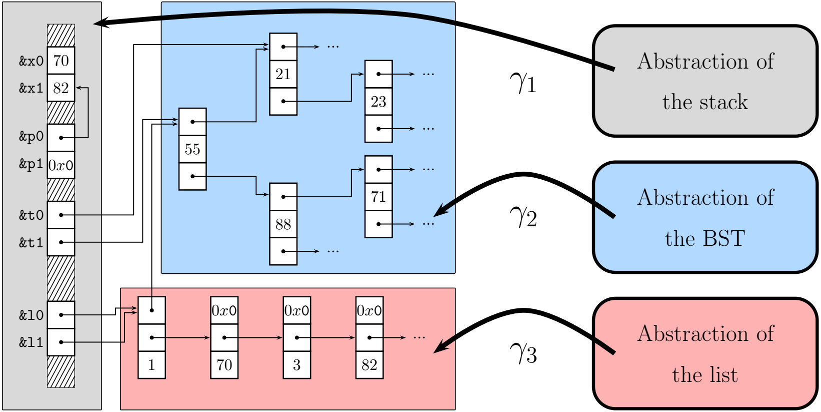
Pros

- ✓ Use **data-structure-specific** abstract domains (and the most efficient algorithms that come with them)
- ✓ Better **modularity** and Abstract domain re-uses
- ✓ Pay the cost of complex algorithms only **where** it is required

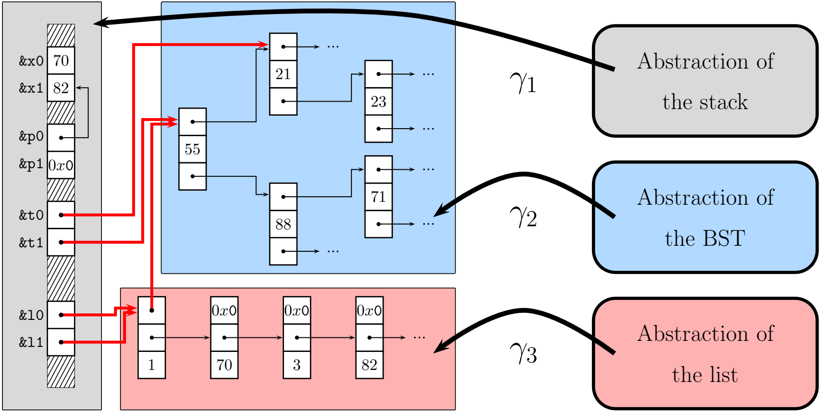
Challenges

- ✗ Set up **abstract transfer functions** for the combination
- ✗ Carefully describe the **interface** between memory regions (e.g. value sharing)

Abstracting the interface between memory regions



Abstracting the interface between memory regions



The **product analysis** **must** abstract **crossing pointers**

▷ **Article** at SAS'14 [1]

▷ **Contributions of the paper :**

- **Formalization** of the separating product of memory abstract domains
- An abstract domain for the **interface** between memory regions
- **Abstract transfer functions** for the separating product
- A **heuristic** to decide which abstract domain should handle which memory chunk
- **Practical validation** (integration into the MemCAD analyzer)

1. An Abstract Domain Combinator for Separately Conjoining Memory Abstraction

Abstraction

Separating product of memory abstract domains : $\mathrm{M}^\sharp_\mathrm{b} \circledast \mathrm{M}^\sharp_\mathrm{ssg}$

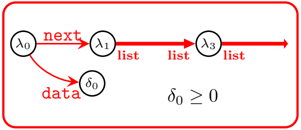
Abstract memories are triples $(m^\sharp_\mathsf{b}, m^\sharp_\mathsf{s}, i^\sharp) \in \mathrm{M}^\sharp_\mathrm{b} \circledast \mathrm{M}^\sharp_\mathrm{ssg}$:

- **Two abstract sub-memories** abstracting **disjoint** part of the memory
- An **abstract interface** $i^\sharp \in \mathrm{I}^\sharp$ representing equalities

- **An abstract memory** $(m^\sharp_\mathsf{b}, m^\sharp_\mathsf{s}, i^\sharp) \in \mathrm{M}^\sharp_\mathrm{b} \circledast \mathrm{M}^\sharp_\mathrm{ssg}$:

&p0 β_0 &x0 β_2 &l0 β_4
&p1 β_1 &x1 β_3 &l1 β_5
$\wedge\ \beta_0 = \&\mathtt{x0}\ \wedge\ \beta_1 = \mathtt{0x0}$
$\wedge\ \beta_2 \in [0; 100]\ \wedge\ \beta_3 > \beta_2$

,



,

$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

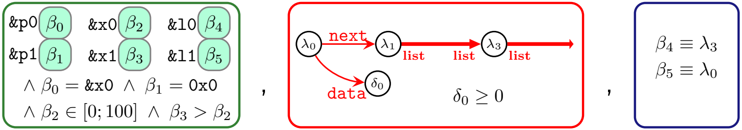
- **Abstracting** the **concrete memory** $m \in \gamma_\circledast(m^\sharp_\mathsf{b}, m^\sharp_\mathsf{s}, i^\sharp)$:

Separating product of memory abstract domains : $\mathrm{M}^\sharp_{\mathrm{b}} \circledast \mathrm{M}^\sharp_{\mathrm{ssg}}$

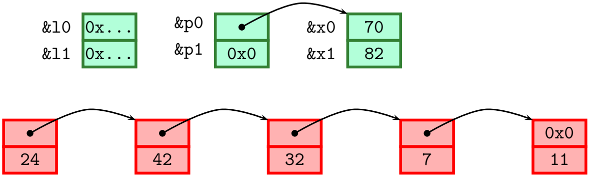
Abstract memories are triples $(m^\sharp_{\mathsf{b}}, m^\sharp_{\mathsf{s}}, i^\sharp) \in \mathrm{M}^\sharp_{\mathrm{b}} \circledast \mathrm{M}^\sharp_{\mathrm{ssg}}$:

- **Two abstract sub-memories** abstracting **disjoint** part of the memory
- An **abstract interface** $i^\sharp \in \mathrm{I}^\sharp$ representing equalities

- **An abstract memory** $(m^\sharp_{\mathsf{b}}, m^\sharp_{\mathsf{s}}, i^\sharp) \in \mathrm{M}^\sharp_{\mathrm{b}} \circledast \mathrm{M}^\sharp_{\mathrm{ssg}}$:



- **Abstracting** the **concrete memory** $m \in \gamma_{\circledast}(m^\sharp_{\mathsf{b}}, m^\sharp_{\mathsf{s}}, i^\sharp)$:



$\in \gamma_{\mathsf{b}}(m^\sharp_{\mathsf{b}})$

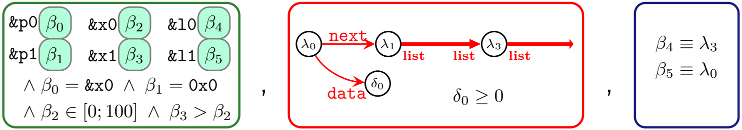
$\in \gamma_{\mathsf{s}}(m^\sharp_{\mathsf{s}})$

Separating product of memory abstract domains : $M^\sharp_b \circledast M^\sharp_{ssg}$

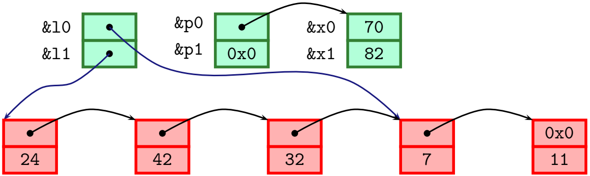
Abstract memories are triples $(m^\sharp_b, m^\sharp_s, i^\sharp) \in M^\sharp_b \circledast M^\sharp_{ssg}$:

- **Two abstract sub-memories** abstracting **disjoint** part of the memory
- An **abstract interface** $i^\sharp \in I^\sharp$ representing equalities

- **An abstract memory** $(m^\sharp_b, m^\sharp_s, i^\sharp) \in M^\sharp_b \circledast M^\sharp_{ssg}$:



- **Abstracting** the **concrete memory** $m \in \gamma_\circledast(m^\sharp_b, m^\sharp_s, i^\sharp)$:



$\in \gamma_b(m^\sharp_b)$

$\in \gamma_s(m^\sharp_s)$

Analysis

Memory allocation

Creation of **new memory cells** occurs in programs.

▷ **Basic Memory abstract domains** handle them.

▷ In separating product $\mathrm{M}^\sharp_\mathrm{b} \circledast \mathrm{M}^\sharp_\mathrm{ssg}$:

- Sub-domains $\mathrm{M}^\sharp_\mathrm{b}$, $\mathrm{M}^\sharp_\mathrm{ssg}$ could both handle them
- The analysis **must** decide which one will do

C types are examined, to **guide** the choice

C struct declaration

```
struct List {
    struct List ⋆ next;
    int data;
};
```

```
0:  int i;
1:  struct List ⋆ l;
2:  l = malloc( sizeof( List ) );
```

Assignment in a separating product : simple case

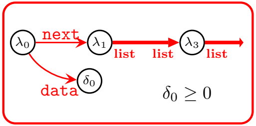
Pre

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \texttt{\&x0}\ \wedge\ \beta_1 = \texttt{0x0}$

$\wedge \ldots$



$\beta_4 \equiv \lambda_3$

$\beta_5 \equiv \lambda_0$

$[\![\star\ \texttt{p0} = \texttt{x1} - \texttt{x0};]\!]^\sharp$

Post

Status - 1

Evaluating l.h.s, in $\mathrm{M}_\mathrm{b}^\sharp$: content of cell &p0 is β_0

Assignment in a separating product : simple case

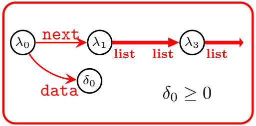
Pre | **Post**

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \&x0\ \wedge\ \beta_1 = 0x0$

$\wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

$[\![\star\ p0\ =\ x1 - x0;]\!]^\sharp$ →

Status - 2

Evaluating l.h.s, in $M_b^\sharp$: as $\beta_0 = \&x0$, l.h.s is cell at address $(\&x0, 0)$

Assignment in a separating product : simple case

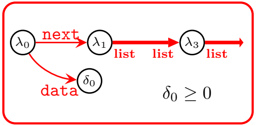
Pre

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \texttt{\&x0}\ \wedge\ \beta_1 = \texttt{0x0}$

$\wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

$[\![\star\, \texttt{p0} \ = \ \texttt{x1} - \texttt{x0;}]\!]^\sharp$

Post

Status - 3

Evaluating r.h.s, in $\mathrm{M}_\mathrm{b}^\sharp$: r.h.s is expression $\beta_3 - \beta_2$

Assignment in a separating product : simple case

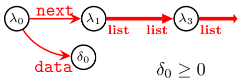
Pre

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \&x0\ \wedge\ \beta_1 = 0x0$

$\wedge \ldots$



$\beta_4 \equiv \lambda_3$

$\beta_5 \equiv \lambda_0$

$[\![\star\ p0\ =\ x1 - x0;]\!]^\sharp$

Post

&p0 β_0 &x0 β_2' &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

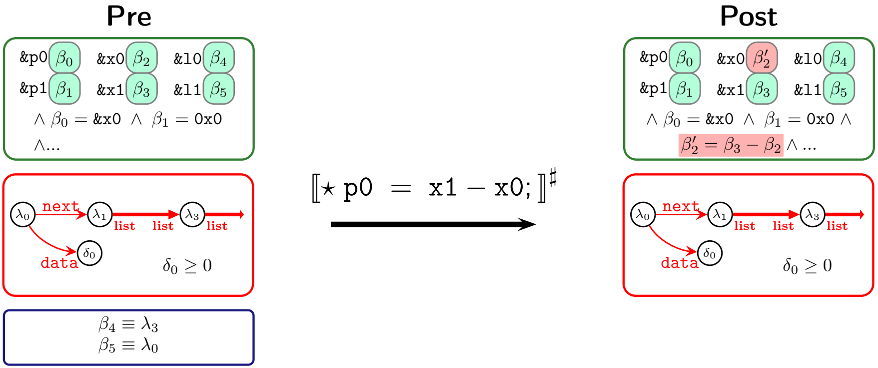
$\wedge\ \beta_0 = \&x0\ \wedge\ \beta_1 = 0x0\ \wedge$

$\beta_2' = \beta_3 - \beta_2 \wedge \ldots$

Status - 4

Writing the cell, in $\mathrm{M}_\mathrm{b}^\sharp$: write $\beta_3 - \beta_2$ into cell at address $(\&x0, 0)$

Assignment in a separating product : simple case



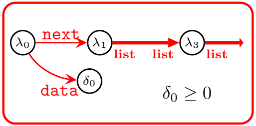
Status - 5

Writing the cell, in $\mathrm{M}^\sharp_{\mathrm{ssg}}$: nothing to do

Assignment in a separating product : simple case

Pre

&p0 β_0 &x0 β_2 &l0 β_4
&p1 β_1 &x1 β_3 &l1 β_5
$\wedge\ \beta_0 =$ &x0 $\wedge\ \beta_1 =$ 0x0
$\wedge \ldots$

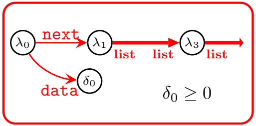


$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

$[\![\star\ \mathtt{p0}\ =\ \mathtt{x1} - \mathtt{x0};]\!]^\sharp$

Post

&p0 β_0 &x0 β_2' &l0 β_4
&p1 β_1 &x1 β_3 &l1 β_5
$\wedge\ \beta_0 =$ &x0 $\wedge\ \beta_1 =$ 0x0 $\wedge$
$\beta_2' = \beta_3 - \beta_2 \wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

Status - 6

Writing the cell, in $\mathrm{I}^\sharp$: nothing to do

Assignment in a separating product : a more complex case

Pre

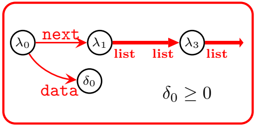
Post

$\&\mathtt{p0}\ \beta_0 \quad \&\mathtt{x0}\ \beta_2 \quad \&\mathtt{l0}\ \beta_4$

$\&\mathtt{p1}\ \beta_1 \quad \&\mathtt{x1}\ \beta_3 \quad \&\mathtt{l1}\ \beta_5$

$\wedge\ \beta_0 = \&\mathtt{x0}\ \wedge\ \beta_1 = \mathtt{0x0}$

$\wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

$$[\![\mathtt{x1}\ =\ (\star\,\mathtt{l1})\cdot\mathtt{data};]\!]^\sharp$$

Status - 1

Evaluating l.h.s, in $\mathrm{M}_\mathrm{b}^\sharp$: l.h.s is cell at address $(\&\mathtt{x1}, 0)$

Assignment in a separating product : a more complex case

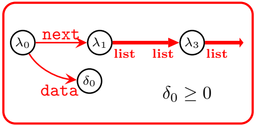
Pre

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \&x0\ \wedge\ \beta_1 = 0x0$

$\wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

$[\![\texttt{x1} = (\star\ \texttt{l1}) \cdot \texttt{data};]\!]^\sharp$

Post

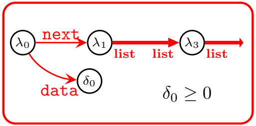
Status - 2

Evaluating r.h.s, in $\mathrm{M}_\mathrm{b}^\sharp$: content of cell &l1 is β_5

Assignment in a separating product : a more complex case

Pre

&p0 β_0 &x0 β_2 &l0 β_4
&p1 β_1 &x1 β_3 &l1 β_5
$\wedge\ \beta_0 =$ &x0 $\wedge\ \beta_1 =$ 0x0
$\wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

Post

$$[\![\texttt{x1} \ = \ (\star\ \texttt{l1}) \cdot \texttt{data};]\!]^\sharp$$

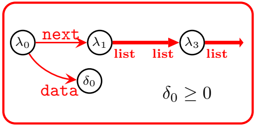
Status - 3

Evaluating r.h.s, in $\mathrm{M}_\mathrm{b}^\sharp$: there is no cell at address $(\beta_5, \texttt{data})$

Assignment in a separating product : a more complex case

Pre

&p0 β_0 &x0 β_2 &l0 β_4
&p1 β_1 &x1 β_3 &l1 β_5
$\wedge\ \beta_0 = \&\mathtt{x0}\ \wedge\ \beta_1 = \mathtt{0x0}$
$\wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

Post

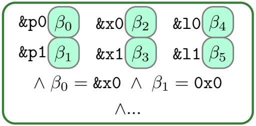
$$[\![\mathtt{x1}\ =\ (\star\ \mathtt{l1}) \cdot \mathtt{data};]\!]^\sharp$$

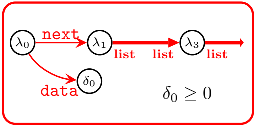
Status - 4

Evaluating r.h.s, in $\mathrm{I}^\sharp$: **retrieve another symbolic name** $\beta_5 = \lambda_0$

Assignment in a separating product : a more complex case

Pre





$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

Post

$[\![\mathtt{x1} \;=\; (\star\,\mathtt{l1})\cdot\mathtt{data};]\!]^\sharp$ ⟶

Status - 5

Evaluating r.h.s, in $\mathrm{M}^\sharp_{\mathrm{ssg}}$: content of cell $(\lambda_0, \mathtt{data})$ is δ_0

Assignment in a separating product : a more complex case

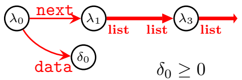
Pre

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \&\mathtt{x0}\ \wedge\ \beta_1 = \mathtt{0x0}$

$\wedge \ldots$



$\delta_0 \geq 0$

$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

$$[\![\mathtt{x1}\ =\ (\star\, \mathtt{l1}) \cdot \mathtt{data};\,]\!]^\sharp \longrightarrow$$

Post

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3' &l1 β_5

$\wedge\ \beta_0 = \&\mathtt{x0}\ \wedge\ \beta_1 = \mathtt{0x0}$

$\wedge \ldots$

Status - 6

Writing the cell, in $\mathrm{M}_{\mathrm{b}}^\sharp$: write **fresh** β_3' in cell at address $(\&\mathtt{x1}, 0)$

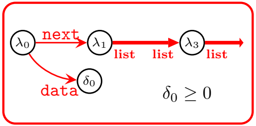
Assignment in a separating product : a more complex case

Pre

&p0 β_0 &x0 β_2 &l0 β_4
&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \&x0 \ \wedge\ \beta_1 = 0x0$

$\wedge \ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

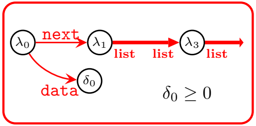
$$[\![\texttt{x1} \ = \ (\star\, \texttt{l1}) \cdot \texttt{data};]\!]^\sharp \longrightarrow$$

Post

&p0 β_0 &x0 β_2 &l0 β_4
&p1 β_1 &x1 β'_3 &l1 β_5

$\wedge\ \beta_0 = \&x0 \ \wedge\ \beta_1 = 0x0$

$\wedge \ldots$



Status - 7

Writing the cell, in $M^\sharp_{ssg}$: nothing to do

Assignment in a separating product : a more complex case

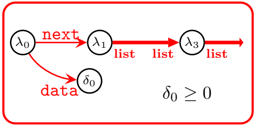
Pre

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3 &l1 β_5

$\wedge\ \beta_0 = \&\text{x0}\ \wedge\ \beta_1 = \text{0x0}$

$\wedge\ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$

$$[\![\text{x1} = (\star\, \text{l1}) \cdot \text{data};]\!]^\sharp$$

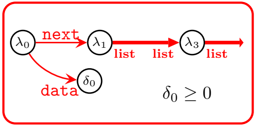
Post

&p0 β_0 &x0 β_2 &l0 β_4

&p1 β_1 &x1 β_3' &l1 β_5

$\wedge\ \beta_0 = \&\text{x0}\ \wedge\ \beta_1 = \text{0x0}$

$\wedge\ldots$



$\beta_4 \equiv \lambda_3$
$\beta_5 \equiv \lambda_0$
$\beta_3' \equiv \delta_0$

Status - 8

Writing the cell, in $\mathrm{I}^\sharp$: write equality $\beta_3' = \delta_0$

Integration into the MemCAD analyzer

- A ML functor : `MEM_DOM -> MEM_DOM -> MEM_DOM`
- Can be iteratively applied, to cope with more than **2 sub-domains**

Target of the analysis :

- **C Programs** ∼100LOC
- **Manipulation** of list and Tree data structures, e.g.
  - insertion/removal routines
  - search in trees
  - ...
- **Interactions** between data structures, e.g.
  - search **in trees** data from **lists**
  - insert **in lists** data from **trees**
  - sort a list using a BST

Goals of the analysis :

- Detect (potential) **null pointer dereferences**
- Data structure **invariant**

Results

Mem. Abstract Domain	t(s)	tSP(s)	#R	#RA
I<list,tree>	0.330	-	172	-
I<list> ⊛ I<tree>	0.364	0.031	172	48
B ⊛ I<list,tree>	0.194	0.035	172	70
B ⊛ I<list> ⊛ I<tree>	0.231	0.071	172	70

- **Memory abstract domains :**
  - B : Bounded Memory Abstract domain
  - I$<\iota_1, ..., \iota_k>$: Separating Shape graphs with inductive definitions
- **t(s)** : Analysis time (in sec.)
- **tSP(s)** : Time spent in the **separating product functor** (in sec.)
- **#R** : Number of **abstract read** operations
- **#RA** : Number of **abstract read** operations **crossing** sub-domains

Results

Mem. Abstract Domain	t(s)	tSP(s)	#R	#RA
I<list,tree>	0.330	-	172	-
I<list> ⊛ I<tree>	0.364	0.031	172	48
B ⊛ I<list,tree>	0.194	0.035	172	70
B ⊛ I<list> ⊛ I<tree>	0.231	0.071	172	70

- **Memory abstract domains :**
  - B : Bounded Memory Abstract domain
  - I$<\iota_1, ..., \iota_k>$: Separating Shape graphs with inductive definitions
- **t(s)** : Analysis time (in sec.)
- **tSP(s)** : Time spent in the **separating product functor** (in sec.)
- **#R** : Number of **abstract read** operations
- **#RA** : Number of **abstract read** operations **crossing** sub-domains

✓ **No loss in precision** with a separating product

✓ **Faster analysis** when sub-domains are efficient

Outline

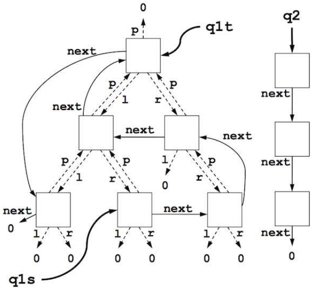
Reduced Product : Insight

Programs sometime manipulate memories with **complex** data structures

- that **can not** be described **using a single inductive definition**
- that **can** be easily described as a **conjunction** of properties

For instance :

- a **doubly linked list**, that is **sorted**, and whose elements have a **static pointer** to the head of the list ;
- **linked list** and **tree** data structures **overlaid** ;



Reduced Product : Insight

Programs sometime manipulate memories with **complex** data structures

- that **can not** be described **using a single inductive definition**
- that **can** be easily described as a **conjunction** of properties

For instance :

- a **doubly linked list**, that is **sorted**, and whose elements have a **static pointer** to the head of the list ;
- **linked list** and **tree** data structures **overlaid** ;

Apply **several time** existing abstractions to whole memory
and take the **conjunction** of them

Discussion

Pros

- ✓ Properties about data structures could be understood separately by programmers/analyzers
- ✓ **Increased** expressiveness
- ✓ Better **modularity**

Potential issues

- ✗ It could be less efficient as it runs several analysis **simultaneously**
- ✗ To remain precise, Memory Abstract Domains **must** be able to exchange **information**

Reduced Product of abstract domains [CC, POPL'79]

Cartesian product : $D_1^\sharp \times D_2^\sharp$

- **conjunction** of properties : $\gamma(x_1^\sharp, x_2^\sharp) \ := \ \gamma_1(x_1^\sharp) \cap \gamma_2(x_2^\sharp)$

Loss of precision during the analysis !

$\mathtt{t} \in [0,3] \quad \boxed{\wedge} \quad \mathtt{t} = 1 \ \mathbf{mod}\ 2$

The information "$t > 0$" : $\begin{cases} \text{is not verified in any component} \\ \text{is expressed by the conjunction} \end{cases}$

The information "$t \neq 0$" is verified by second component

Reduced Product of abstract domains [CC, POPL'79]

Cartesian product : $D_1^\sharp \times D_2^\sharp$

- **conjunction** of properties : $\gamma(x_1^\sharp, x_2^\sharp) \;:=\; \gamma_1(x_1^\sharp) \cap \gamma_2(x_2^\sharp)$

Loss of precision during the analysis !

$$\mathtt{t} \in [0, 3] \quad \boxed{\wedge} \quad \mathtt{t} = 1 \textbf{ mod } 2$$

The information "$t > 0$" : $\begin{cases} \text{is not } \textbf{verified} \text{ in any component} \\ \text{is } \textbf{expressed} \text{ by the } \textbf{conjunction} \end{cases}$

The information "$t \neq 0$" is verified by **second component**

Reduction

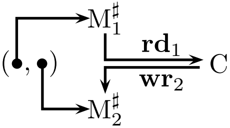
$$\mathtt{t} \in [1, 3] \quad \boxed{\wedge} \quad \mathtt{t} = 1 \textbf{ mod } 2$$

A generic reduction operator construction

Communication between two memory abstract domains $\mathrm{M}_1^\sharp$, $\mathrm{M}_2^\sharp$

- a universal language of constraints C
- a concretisation function : γ_{C}
- two operators handling communications with abstract domains :

$$\begin{cases} \mathbf{rd}_i \quad : \quad \mathrm{M}_i^\sharp \to \mathrm{C} & \textit{reads constrains} \\ \quad \gamma_i(m_i^\sharp) \subseteq \gamma_{\mathrm{C}}(\mathbf{rd}(m_i^\sharp)) & \\ \mathbf{wr}_i \quad : \quad \mathrm{C} \times \mathrm{M}_i^\sharp \to \mathrm{M}_i^\sharp & \textit{writes constraints} \\ \quad \gamma_i(m_i^\sharp) \cap \gamma_{\mathrm{C}}(c) \subseteq \gamma_i(\mathbf{wr}(c, m_i^\sharp)) & \end{cases}$$



Reduction functions

$$\rho_{1\to 2}(m_1^\sharp, m_2^\sharp) := \langle m_1^\sharp, \mathbf{wr}_2(\mathbf{rd}_1(m_1^\sharp), m_2^\sharp)\rangle$$
$$\rho_{2\to 1}(m_1^\sharp, m_2^\sharp) := \langle \mathbf{wr}_1(\mathbf{rd}_2(m_2^\sharp), m_1^\sharp), m_2^\sharp\rangle$$

▷ **Article** at VMCAI'13[2]

▷ **Contributions of the paper :**

- **Formalization** of a reduced product of memory abstract domains
- **Abstract transfer functions** for the reduced product
- **Formalization** of a **universal language of constraints** for communication between memory abstract domains
- **Static (pre)analysis** of **inductive definition** for constraints extraction
- **Practical validation** (integration into the MemCAD analyzer)

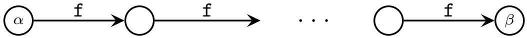
2. Reduced Product Combination of Abstract Domains for Shapes

Universal Language of Constraints

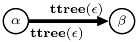
Path predicate

$$\alpha \cdot p \triangleright \beta \quad \begin{cases} \alpha \text{ and } \beta \text{ denote symbolic variables} \\ p \text{ is a regular expression of fields} \end{cases}$$

- Predicate $\alpha \cdot (\mathtt{f})^* \triangleright \beta$ means :



- **Read** operator :

rd(

) $= \{\alpha \cdot (\mathtt{lft} + \mathtt{rgt})^* \triangleright \beta\}$

- **Sound** unbounded path predicates for **inductive** predicate are **automatically inferred**

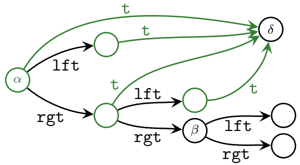
Universal Language of Constraints

Path quantification

$\mathcal{S}_\forall[p, a[X]](\alpha, S)$ $\begin{cases} \alpha \text{ denotes a symbolic variable, } S \text{ denotes a set of s.v.} \\ p \text{ is a regular expression on fields} \\ a[X] \text{ is a path predicates with a free variable } X \end{cases}$

In the right figure, green nodes χ :

- are characterized by :
  - ➢ they **can** be reached from α, following **path expression** $(\texttt{lft} + \texttt{rgt})^*$;
  - ➢ they **cannot** be reached from β, following **path expression** $(\texttt{lft} + \texttt{rgt})^*$;
- satisfy the property $\chi \cdot \texttt{t} \triangleright \delta$;



$\mathcal{S}_\forall[(\texttt{lft}+\texttt{rgt})^*, X\cdot\texttt{t}\triangleright\delta](\alpha, \{\beta\})$

Implementation : reduction strategies

When do we trigger reduction ?

- Only when the analysis is about **to run out all the information** :

  Minimal strategy

- At each computed abstract states :

  Maximal strategy

- When the location of a cell is about **to be lost** :

  On-read strategy

- ...

Empirical notion of strategies

Practical verification

Integration into the MemCAD analyzer

- A ML functor : `MEM_DOM -> MEM_DOM -> MEM_DOM`
- Can be iteratively applied, to cope with more than **2 sub-domains**

Program :
C programs manipulating overlaid data structures.
Random traversal + routines

Strategy	Time	Red.calls
minimal	0.120	4
maximal	0.095	32
on-read	0.086	9

on-read strategy is a **good balance**

Between 2X and 3X slower than the analysis with a monolithic memory abstract domain (when it is possible).

Conclusion

MemCAD analyzer

- **Great Modularity** in the choice of the Memory Abstraction

Generic framework for Separately Conjoining Memory Abstractions

- **Modular Spatial combination** of memory abstract domains
- **Abstraction** of the interface between memory regions

Generic framework for Reduced Product of Memory Abstractions

- **Modular combination** of memory abstract domains
- **Mechanism** for extracting **constraints** from inductive definitions

The End.