A new algorithm for Higher-order model checking

Jérémy Ledent Martin Hofmann

For first order programs (M. Hofmann & W. Chen)

Let Σ be a set of *events* and $\mathcal{F}$ a set of procedure identifiers.

- Syntax of expressions:

$$e ::= \underline{a} \mid f \mid e_1; e_2 \mid e_1 + e_2 \qquad \text{where } a \in \Sigma \text{ and } f \in \mathcal{F}$$

For first order programs (M. Hofmann & W. Chen)

Let Σ be a set of *events* and $\mathcal{F}$ a set of procedure identifiers.

- Syntax of expressions:

$$e ::= \underline{a} \mid f \mid e_1; e_2 \mid e_1 + e_2 \qquad \text{where } a \in \Sigma \text{ and } f \in \mathcal{F}$$

- Program: an expression e_f for every $f \in \mathcal{F}$.

Examples:

$$\begin{aligned} f &= \underline{a}; \underline{b}; g \\ g &= \underline{d} + (\underline{c}; f) \end{aligned}$$

$$L(f) \quad = (abc)^* abd \ \cup \ \{(abc)^\omega\}$$

For first order programs (M. Hofmann & W. Chen)

Let Σ be a set of *events* and $\mathcal{F}$ a set of procedure identifiers.

- Syntax of expressions:

$$e ::= \underline{a} \mid f \mid e_1; e_2 \mid e_1 + e_2 \qquad \text{where } a \in \Sigma \text{ and } f \in \mathcal{F}$$

- Program: an expression e_f for every $f \in \mathcal{F}$.

Examples:

$$\begin{aligned} f &= \underline{a}; \underline{b}; g \\ g &= \underline{d} + (\underline{c}; f) \end{aligned} \qquad\qquad \begin{aligned} u &= \underline{a}; v \\ v &= v \end{aligned}$$

$$\begin{aligned} L(f) &= (abc)^* abd \ \cup \ \{(abc)^\omega\} \\ L(u) &= \{a\} \end{aligned}$$

For first order programs (M. Hofmann & W. Chen)

Let Σ be a set of *events* and $\mathcal{F}$ a set of procedure identifiers.

- Syntax of expressions:

$$e ::= \underline{a} \mid f \mid e_1; e_2 \mid e_1 + e_2 \qquad \text{where } a \in \Sigma \text{ and } f \in \mathcal{F}$$

- Program: an expression e_f for every $f \in \mathcal{F}$.

Examples:

$$\begin{aligned} f &= \underline{a}; \underline{b}; g \\ g &= \underline{d} + (\underline{c}; f) \end{aligned} \qquad\qquad \begin{aligned} u &= \underline{a}; v \\ v &= v \end{aligned}$$

$$\begin{aligned} L_*(f) &= (ab\checkmark\, c\checkmark\,)^* ab\checkmark\, d \\ L_*(u) &= \varnothing \end{aligned} \qquad\qquad \begin{aligned} L_\omega(f) &= \{(ab\checkmark\, c\checkmark\,)^\omega\} \\ L_\omega(u) &= \{a(\checkmark\,)^\omega\} \end{aligned}$$

Policy Automaton

```
#define TIMEOUT 65536
while (true) {
  int i,s; i = s = 0;
  while (i++ < TIMEOUT && s == 0) {
    s = auth();
  }
  work();
}
```

Policy Automaton

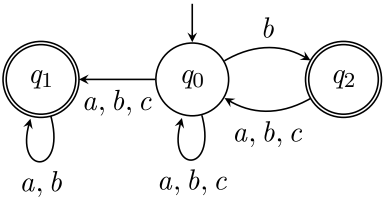
```
#define TIMEOUT 65536
while (true) {
  int i,s; i = s = 0;
  while (i++ < TIMEOUT && s == 0) {
    s = auth(); /* a */
  } /* c */
  work(); /* b */
}
```

$$
\begin{aligned}
f &= g; \underline{b}; f \\
g &= (\underline{a}; g) + \underline{c}
\end{aligned}
$$

Policy Automaton

```
#define TIMEOUT 65536
while (true) {
  int i,s; i = s = 0;
  while (i++ < TIMEOUT && s == 0) {
    s = auth(); /* a */
  } /* c */
  work(); /* b */
}
```

$$f = g; \underline{b}; f$$
$$g = (\underline{a}; g) + \underline{c}$$



"If c occurs infinitely often, then b occurs infinitely often."

Büchi type system

Let $GFb = (a^* b)^\omega$ be a type asserting "b occurs infinitely often".

Consider the procedure:

$$f = \underline{a}; f$$

Assuming $f : GFb$, we can derive $(\underline{a}; f) : aGFb$, and since $aGFb = GFb$, that means we have a derivation

$$f : GFb \;\vdash\; (\underline{a}; f) : GFb$$

Büchi type system

Let $GFb = (a^* b)^\omega$ be a type asserting "b occurs infinitely often".

Consider the procedure:

$$f = \underline{a}; f$$

Assuming $f : GFb$, we can derive $(\underline{a}; f) : aGFb$, and since $aGFb = GFb$, that means we have a derivation

$$f : GFb \;\vdash\; (\underline{a}; f) : GFb$$

Under "usual" typing rules, this would allow us to establish

$$\vdash\; f : GFb$$

which is clearly wrong.

Büchi type system

Idea:

$$\frac{f : X \;\vdash\; e_f : T(X)}{\vdash\; f : \mathrm{gfp}(\lambda X.\; T(X))}$$

Büchi type system

Idea:

$$\frac{f : X \ \vdash \ e_f : T(X)}{\vdash \ f : \mathrm{gfp}(\lambda X.\ T(X))}$$

$f = (\underline{a}; f) + \underline{b}$

Looks like a language equation $X = aX + b$
Smallest solution: $X = a^* b$
Greatest solution: $X = a^* b + a^\omega = L(f)$

Büchi type system

Idea:

$$\frac{f : X \ \vdash \ e_f : T(X)}{\vdash \ f : \mathrm{gfp}(\lambda X.\ T(X))}$$

$f = (\underline{a}; f) + \underline{b}$

Looks like a language equation $X = aX + b$
Smallest solution: $X = a^* b$
Greatest solution: $X = a^* b + a^\omega = L(f)$

For first-order programs:

$$T(X) = U \cdot X + V$$

$$\mathrm{gfp}(T) = U^* V + U^\omega$$

Büchi Abstraction

Let $\mathfrak{L}_* = \mathcal{P}(\Sigma^*)$ and $\mathfrak{L}_\omega = \mathcal{P}(\Sigma^\omega)$.
Given the policy automaton $\mathcal{A}$, we can construct complete lattices $\mathfrak{M}_*$ and $\mathfrak{M}_\omega$ such that:

- They are finite.

Büchi Abstraction

Let $\mathfrak{L}_* = \mathcal{P}(\Sigma^*)$ and $\mathfrak{L}_\omega = \mathcal{P}(\Sigma^\omega)$.
Given the policy automaton $\mathcal{A}$, we can construct complete lattices $\mathfrak{M}_*$ and $\mathfrak{M}_\omega$ such that:

- They are finite.
- They are related to $\mathfrak{L}_*$, $\mathfrak{L}_\omega$ by a *galois insertion*. There are $\alpha_{*/\omega} : \mathfrak{L}_{*/\omega} \rightarrow \mathfrak{M}_{*/\omega}$ and $\gamma_{*/\omega} : \mathfrak{M}_{*/\omega} \rightarrow \mathfrak{L}_{*/\omega}$ such that

$$\gamma_{*/\omega}(\alpha_{*/\omega}(L)) \supseteq L \qquad \text{and} \qquad \alpha_{*/\omega}(\gamma_{*/\omega}(U)) = U$$

Büchi Abstraction

Let $\mathfrak{L}_* = \mathcal{P}(\Sigma^*)$ and $\mathfrak{L}_\omega = \mathcal{P}(\Sigma^\omega)$.
Given the policy automaton $\mathcal{A}$, we can construct complete lattices $\mathfrak{M}_*$ and $\mathfrak{M}_\omega$ such that:

- They are finite.
- They are related to $\mathfrak{L}_*$, $\mathfrak{L}_\omega$ by a *galois insertion*. There are $\alpha_{*/\omega} : \mathfrak{L}_{*/\omega} \to \mathfrak{M}_{*/\omega}$ and $\gamma_{*/\omega} : \mathfrak{M}_{*/\omega} \to \mathfrak{L}_{*/\omega}$ such that

$$\gamma_{*/\omega}(\alpha_{*/\omega}(L)) \supseteq L \qquad \text{and} \qquad \alpha_{*/\omega}(\gamma_{*/\omega}(U)) = U$$

- $L \subseteq L(\mathcal{A}) \iff \alpha(L) \sqsubseteq \alpha(L(\mathcal{A}))$

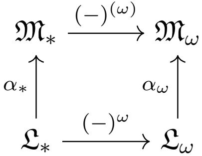
Büchi Abstraction

Let $\mathfrak{L}_* = \mathcal{P}(\Sigma^*)$ and $\mathfrak{L}_\omega = \mathcal{P}(\Sigma^\omega)$.
Given the policy automaton $\mathcal{A}$, we can construct complete lattices $\mathfrak{M}_*$ and $\mathfrak{M}_\omega$ such that:

- They are finite.
- They are related to $\mathfrak{L}_*$, $\mathfrak{L}_\omega$ by a *galois insertion*. There are $\alpha_{*/\omega} : \mathfrak{L}_{*/\omega} \to \mathfrak{M}_{*/\omega}$ and $\gamma_{*/\omega} : \mathfrak{M}_{*/\omega} \to \mathfrak{L}_{*/\omega}$ such that

$$\gamma_{*/\omega}(\alpha_{*/\omega}(L)) \supseteq L \qquad \text{and} \qquad \alpha_{*/\omega}(\gamma_{*/\omega}(U)) = U$$

- $L \subseteq L(\mathcal{A}) \iff \alpha(L) \sqsubseteq \alpha(L(\mathcal{A}))$
- The abstraction function α preserves unions, concatenation, least fixpoints and ω-iteration (but not greatest fixpoints !):



Büchi Abstraction

Define the equivalence relation $\sim_{\mathcal{A}}$ on Σ^+ as follows: $u \sim_{\mathcal{A}} v$ iff

$$\forall q, q'.\ (q \xrightarrow{u} q' \iff q \xrightarrow{v} q') \wedge (q \xrightarrow{u}_F q' \iff q \xrightarrow{v}_F q')$$

and extend it to Σ^* such that $[\varepsilon] = \{\varepsilon\}$.

Büchi Abstraction

Define the equivalence relation $\sim_{\mathcal{A}}$ on Σ^+ as follows: $u \sim_{\mathcal{A}} v$ iff

$$\forall q, q'. \ (q \xrightarrow{u} q' \iff q \xrightarrow{v} q') \wedge (q \xrightarrow{u}_F q' \iff q \xrightarrow{v}_F q')$$

and extend it to Σ^* such that $[\varepsilon] = \{\varepsilon\}$.

- Equivalence classes are regular languages.
- There's a finite number of classes.

Büchi Abstraction

Define the equivalence relation $\sim_{\mathcal{A}}$ on Σ^+ as follows: $u \sim_{\mathcal{A}} v$ iff

$$\forall q, q'. \, (q \xrightarrow{u} q' \iff q \xrightarrow{v} q') \wedge (q \xrightarrow{u}_F q' \iff q \xrightarrow{v}_F q')$$

and extend it to Σ^* such that $[\varepsilon] = \{\varepsilon\}$.

- Equivalence classes are regular languages.
- There's a finite number of classes.
- For every class C, either $C \cap L_*(\mathcal{A}) = \varnothing$ or $C \subseteq L_*(\mathcal{A})$.

Büchi Abstraction

Define the equivalence relation $\sim_{\mathcal{A}}$ on Σ^+ as follows: $u \sim_{\mathcal{A}} v$ iff

$$\forall q, q'.\ (q \xrightarrow{u} q' \iff q \xrightarrow{v} q') \wedge (q \xrightarrow{u}_F q' \iff q \xrightarrow{v}_F q')$$

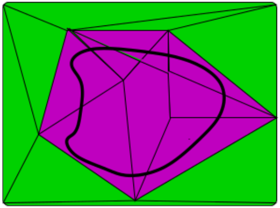
and extend it to Σ^* such that $[\varepsilon] = \{\varepsilon\}$.

- Equivalence classes are regular languages.
- There's a finite number of classes.
- For every class C, either $C \cap L_*(\mathcal{A}) = \varnothing$ or $C \subseteq L_*(\mathcal{A})$.
- For every C, D, either $CD^\omega \cap L_\omega(\mathcal{A}) = \varnothing$ or $CD^\omega \subseteq L_\omega(\mathcal{A})$.
- For every $w \in \Sigma^\omega$, there are C, D such that $w \in CD^\omega$.

The sets CD^ω behave almost like classes, but they may overlap !

Büchi Abstraction

Define $\mathfrak{M}_* = \mathcal{P}(\Sigma^*/\sim_{\mathcal{A}})$

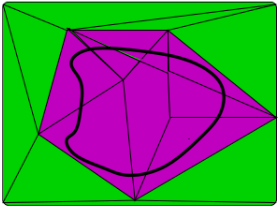


$$\gamma_*(\mathcal{V}) = \bigcup_{C \in \mathcal{V}} C$$

$$\alpha_*(L) = \{C \mid C \cap L \neq \varnothing\}$$

Büchi Abstraction

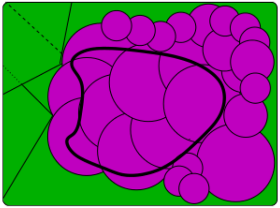
Define $\mathfrak{M}_* = \mathcal{P}(\Sigma^*/\sim_{\mathcal{A}})$



$$\gamma_*(\mathcal{V}) = \bigcup_{C \in \mathcal{V}} C$$

$$\alpha_*(L) = \{C \mid C \cap L \neq \varnothing\}$$

and $\mathfrak{M}_\omega = \{\mathcal{V} \subseteq (\Sigma^*/\sim_{\mathcal{A}}) \times (\Sigma^*/\sim_{\mathcal{A}}) \mid \mathcal{V} \text{ is closed}\}$



$$\gamma_\omega(\mathcal{V}) = \bigcup_{(C,D) \in \mathcal{V}} CD^\omega$$

$$\alpha_\omega(L) = \mathsf{cl}\ \{(C, D) \mid CD^\omega \cap L \neq \varnothing\}$$

Extending to Higher-order

Terms:

$$e ::= x \mid \underline{a} \mid e_1; e_2 \mid e_1 + e_2 \mid \text{fix } e \mid \lambda x.\, e \mid e_1 \ e_2$$

Extending to Higher-order

Terms:

$$e ::= x \mid \underline{a} \mid e_1; e_2 \mid e_1 + e_2 \mid \text{fix}\ e \mid \lambda x.\ e \mid e_1\ e_2$$

Types:

$$\tau ::= o \mid \tau_1 \rightarrow \tau_2$$

Typing rules:

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma \vdash e : \tau \rightarrow \tau}{\Gamma \vdash \text{fix}\ e : \tau} \qquad \frac{}{\Gamma \vdash \underline{a} : o} \qquad \frac{\Gamma \vdash e_1 : o \quad \Gamma \vdash e_2 : o}{\Gamma \vdash e_1 + e_2 : o} \qquad \frac{\Gamma \vdash e_1 : o \quad \Gamma \vdash e_2 : o}{\Gamma \vdash e_1; e_2 : o}$$

Extending to Higher-order

Terms:

$$e ::= x \mid \underline{a} \mid e_1; e_2 \mid e_1 + e_2 \mid \mathrm{fix}\ e \mid \lambda x.\, e \mid e_1\ e_2$$

Types:

$$\tau ::= o \mid \tau_1 \to \tau_2$$

Typing rules:

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash e : \tau \to \tau}{\Gamma \vdash \mathrm{fix}\ e : \tau} \qquad \frac{}{\Gamma \vdash \underline{a} : o} \qquad \frac{\Gamma \vdash e_1 : o \quad \Gamma \vdash e_2 : o}{\Gamma \vdash e_1 + e_2 : o} \qquad \frac{\Gamma \vdash e_1 : o \quad \Gamma \vdash e_2 : o}{\Gamma \vdash e_1; e_2 : o}$$

Program: closed term of type o.

Examples

First order: only use $\mathrm{fix} : (o \to o) \to o$.

- $\mathrm{fix}(\lambda f.\, (\underline{a}; f) + \underline{b})$
- $\mathrm{fix}(\lambda f.\, \underline{a}; \underline{b}; \mathrm{fix}(\lambda g.\, \underline{d} + (\underline{c}; f)))$

Examples

First order: only use $\text{fix} : (o \to o) \to o$.

- $\text{fix}(\lambda f.\, (\underline{a}; f) + \underline{b})$
- $\text{fix}(\lambda f.\, \underline{a}; \underline{b}; \text{fix}(\lambda g.\, \underline{d} + (\underline{c}; f)))$

Call-by-value versus call-by-name:

- $e = (\lambda x.\, \underline{a}; x)\ \underline{b} \quad \longrightarrow \quad L_*(e) = \{ab\}$

Examples

First order: only use $\mathrm{fix} : (o \to o) \to o$.

- $\mathrm{fix}(\lambda f.\, (\underline{a}; f) + \underline{b})$
- $\mathrm{fix}(\lambda f.\, \underline{a}; \underline{b}; \mathrm{fix}(\lambda g.\, \underline{d} + (\underline{c}; f)))$

Call-by-value versus call-by-name:

- $e = (\lambda x.\, \underline{a}; x)\ \underline{b} \quad \longrightarrow \quad L_*(e) = \{ab\}$

Non context-free examples:

- $e' = \mathrm{fix}(\lambda f. \lambda x.\, (\underline{a}; f(\underline{b}; x; \underline{c})) + x)$

$$L_*(e'\ \underline{d}) = \{a^n b^n d c^n \mid n \geq 0\} \qquad L_\omega(e'\ \underline{d}) = \{a^\omega\}$$

Examples

First order: only use $\mathrm{fix} : (o \to o) \to o$.

- $\mathrm{fix}(\lambda f.\, (\underline{a}; f) + \underline{b})$
- $\mathrm{fix}(\lambda f.\, \underline{a}; \underline{b}; \mathrm{fix}(\lambda g.\, \underline{d} + (\underline{c}; f)))$

Call-by-value versus call-by-name:

- $e = (\lambda x.\, \underline{a}; x)\ \underline{b} \quad \longrightarrow \quad L_*(e) = \{ab\}$

Non context-free examples:

- $e' = \mathrm{fix}(\lambda f.\lambda x.\, (\underline{a}; f(\underline{b}; x; \underline{c})) + x)$

$$L_*(e'\ \underline{d}) = \{a^n b^n d c^n \mid n \geq 0\} \qquad L_\omega(e'\ \underline{d}) = \{a^\omega\}$$

- $e'' = \mathrm{fix}(\lambda x.\, (e'\ \underline{d}); x)$

$$L_*(e'') = \emptyset \qquad L_\omega(e'') = (L_*(e'\ \underline{d}))^\omega \cup \{a^\omega\}$$

Related Work

Higher-order model checking (Ong & Kobayashi, Walukiewicz & Salvati, Melliès & Grellois).

- $\lambda\mathbf{Y}$, higher-order recursion schemes, higher-order pushdown automata with collapse.
- Model-checking of temporal logic, μ-calculus formulas.
- Relies heavily on tree properties, even if we are only interested in traces.

Related Work

Higher-order model checking (Ong & Kobayashi, Walukiewicz & Salvati, Melliès & Grellois).

- $\lambda\mathbf{Y}$, higher-order recursion schemes, higher-order pushdown automata with collapse.
- Model-checking of temporal logic, μ-calculus formulas.
- Relies heavily on tree properties, even if we are only interested in traces.

Example: $\lambda\mathbf{Y}$.

Choose first-order constants

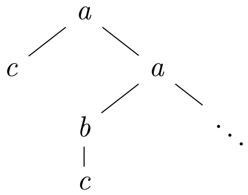
$a : o \to o \to o$

$b : o \to o$

$c : o$

$M = \mathbf{Y}(\lambda f.\, \lambda x.\, a\ x\ (f\ (b\ x)))$

Böhm-tree of $(M\ c)$:



GFP semantics

We define the category **GFP**

- Its objects A are pairs (A_*, A_ω) of complete lattices.
- A morphism $f : A \to B$ is a pair (f_*, f_ω) where
  - $f_* : A_* \to B_*$
  - $f_\omega : A_* \times A_\omega \to B_\omega$

GFP semantics

We define the category **GFP**

- Its objects A are pairs (A_*, A_ω) of complete lattices.
- A morphism $f : A \to B$ is a pair (f_*, f_ω) where
  - $f_* : A_* \to B_*$
  - $f_\omega : A_* \times A_\omega \to B_\omega$

Composition $h = g \circ f$ is given by

- $h_*(a_*) = g_*(f_*(a_*))$
- $h_\omega(a_*, a_\omega) = g_\omega(f_*(a_*), f_\omega(a_*, a_\omega))$

GFP semantics

We define the category **GFP**

- Its objects A are pairs (A_*, A_ω) of complete lattices.
- A morphism $f : A \to B$ is a pair (f_*, f_ω) where
  - $f_* : A_* \to B_*$
  - $f_\omega : A_* \times A_\omega \to B_\omega$

Composition $h = g \circ f$ is given by

- $h_*(a_*) = g_*(f_*(a_*))$
- $h_\omega(a_*, a_\omega) = g_\omega(f_*(a_*), f_\omega(a_*, a_\omega))$

Proposition

***GFP** is cartesian-closed.*

Cartesian products

- $(A \times B)_* = A_* \times B_*$
- $(A \times B)_\omega = A_\omega \times B_\omega$

Function spaces

- $(A \Rightarrow B)_* = B_*^{A_*}$
- $(A \Rightarrow B)_\omega = B_\omega^{A_* \times A_\omega}$

GFP semantics

GFP has the following fixpoint combinator for every A:

$$\mathrm{fix}_A : (A \Rightarrow A) \to A$$

where

- $(\mathrm{fix}_A)_*(f_*) = \mathrm{lfp}(f_*)$
- $(\mathrm{fix}_A)_\omega(f_*, f_\omega) = \mathrm{gfp}(\lambda a_\omega.\, f_\omega(\mathrm{lfp}(f_*), a_\omega))$

Proposition

*This is indeed a fixpoint: $f(\mathrm{fix}_A(f)) = \mathrm{fix}_A(f)$ holds in the internal language of **GFP***

$$\mathrm{app} \circ \langle \mathrm{id}_{A \Rightarrow A}, \mathrm{fix}_A \rangle = \mathrm{fix}_A$$

GFP semantics

Interpretation of types:

To every type τ, associate an object $[\![\tau]\!]$ of **GFP**

$$[\![o]\!] = (\mathfrak{L}_*, \mathfrak{L}_\omega) \qquad \text{and} \qquad [\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$$

GFP semantics

Interpretation of types:

To every type τ, associate an object $[\![\tau]\!]$ of **GFP**

$$[\![o]\!] = (\mathfrak{L}_*, \mathfrak{L}_\omega) \qquad \text{and} \qquad [\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$$

Interpretation of contexts:

To a context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, associate the object

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!]$$

GFP semantics

Interpretation of types:

To every type τ, associate an object $[\![\tau]\!]$ of **GFP**

$$[\![o]\!] = (\mathfrak{L}_*, \mathfrak{L}_\omega) \qquad \text{and} \qquad [\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$$

Interpretation of contexts:

To a context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, associate the object

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!]$$

Interpretation of terms:

To a derivation $\Gamma \vdash e : \tau$, associate a morphism $[\![e]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$

GFP semantics

Interpretation of types:

To every type τ, associate an object $[\![\tau]\!]$ of **GFP**

$$[\![o]\!] = (\mathfrak{L}_*, \mathfrak{L}_\omega) \qquad \text{and} \qquad [\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$$

Interpretation of contexts:

To a context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, associate the object

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!]$$

Interpretation of terms:

To a derivation $\Gamma \vdash e : \tau$, associate a morphism $[\![e]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$

- $[\![\underline{a}]\!] = (\{a\}, \varnothing)$

GFP semantics

Interpretation of types:

To every type τ, associate an object $[\![\tau]\!]$ of **GFP**

$$[\![o]\!] = (\mathfrak{L}_*, \mathfrak{L}_\omega) \qquad \text{and} \qquad [\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$$

Interpretation of contexts:

To a context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, associate the object

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!]$$

Interpretation of terms:

To a derivation $\Gamma \vdash e : \tau$, associate a morphism $[\![e]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$

- $[\![\underline{a}]\!] = (\{a\}, \varnothing)$
- $[\![+]\!]_*(X_*, Y_*) = X_* \cup Y_*$
  $[\![+]\!]_\omega(X_*, Y_*, X_\omega, Y_\omega) = X_\omega \cup Y_\omega$

GFP semantics

Interpretation of types:

To every type τ, associate an object $[\![\tau]\!]$ of **GFP**

$$[\![o]\!] = (\mathfrak{L}_*, \mathfrak{L}_\omega) \qquad \text{and} \qquad [\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$$

Interpretation of contexts:

To a context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, associate the object

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!]$$

Interpretation of terms:

To a derivation $\Gamma \vdash e : \tau$, associate a morphism $[\![e]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$

- $[\![\underline{a}]\!] = (\{a\}, \varnothing)$
- $[\![+]\!]_*(X_*, Y_*) = X_* \cup Y_*$
  $[\![+]\!]_\omega(X_*, Y_*, X_\omega, Y_\omega) = X_\omega \cup Y_\omega$
- $[\![\,;]\!]_*(X_*, Y_*) = X_* Y_*$
  $[\![\,;]\!]_\omega(X_*, Y_*, X_\omega, Y_\omega) = X_\omega \cup X_* Y_\omega$

GFP semantics

Reminder: a *program* is a closed term of type o.

Let e be a program, then $[\![e]\!] : 1 \to [\![o]\!]$ is (isomorphic to) an element of $\mathfrak{L}_* \times \mathfrak{L}_\omega$.

Theorem

*Let e be a program, write $(L_*, L_\omega) = [\![e]\!]$ its interpretation in **GFP**. Then we have $L_*(e) = L_*$ and $L_\omega(e) = L_\omega$.*

GFP semantics

Reminder: a *program* is a closed term of type o.

Let e be a program, then $[\![e]\!] : 1 \to [\![o]\!]$ is (isomorphic to) an element of $\mathfrak{L}_* \times \mathfrak{L}_\omega$.

Theorem

*Let e be a program, write $(L_*, L_\omega) = [\![e]\!]$ its interpretation in **GFP**. Then we have $L_*(e) = L_*$ and $L_\omega(e) = L_\omega$.*

If we choose $[\![o]\!] = (\mathfrak{M}_*, \mathfrak{M}_\omega)$ instead, everything is computable.

But α doesn't commute with greatest fixpoints :-(

Affine Functions

For first-order fixpoints:

The denotation of $f : o \to o$ has two components:

- $[\![f]\!]_* : \mathfrak{L}_* \to \mathfrak{L}_*$
- $[\![f]\!]_\omega : \mathfrak{L}_* \times \mathfrak{L}_\omega \to \mathfrak{L}_\omega$

$[\![\mathrm{fix}\ f]\!]$ involves some gfp of $[\![f]\!]_\omega$.

Affine Functions

For first-order fixpoints:
The denotation of $f : o \to o$ has two components:

- $[\![f]\!]_* : \mathfrak{L}_* \to \mathfrak{L}_*$
- $[\![f]\!]_\omega : \mathfrak{L}_* \times \mathfrak{L}_\omega \to \mathfrak{L}_\omega$

$[\![\mathrm{fix}\ f]\!]$ involves some gfp of $[\![f]\!]_\omega$.

But every function $F : \mathfrak{L}_* \times \mathfrak{L}_\omega \to \mathfrak{L}_\omega$ that actually occurs as the interpretation of a term is *affine*: there exists $A : \mathfrak{L}_* \to \mathfrak{L}_*$ and $B : \mathfrak{L}_* \to \mathfrak{L}_\omega$ such that

$$F(x, X) = A(x) \cdot X \cup B(x)$$

Then $\mathrm{gfp}(F(x, -)) = A(x)^* B(x) \ \cup \ A(x)^\omega$ commutes with α.

Affine Functions

For higher-order fixpoints:

Consider $f : (\tau \to o) \to (\tau \to o)$, then

$$[\![f]\!]_\omega : [\![\tau \to o]\!]_* \times ([\![\tau]\!]_* \times [\![\tau]\!]_\omega \Rightarrow \mathfrak{L}_\omega) \to ([\![\tau]\!]_* \times [\![\tau]\!]_\omega \Rightarrow \mathfrak{L}_\omega)$$

Affine Functions

For higher-order fixpoints:

Consider $f : (\tau \to o) \to (\tau \to o)$, then

$$[\![f]\!]_\omega : [\![\tau \to o]\!]_* \times ([\![\tau]\!]_* \times [\![\tau]\!]_\omega \Rightarrow \mathfrak{L}_\omega) \to ([\![\tau]\!]_* \times [\![\tau]\!]_\omega \Rightarrow \mathfrak{L}_\omega)$$

A function $F : S \times (T \Rightarrow \mathfrak{L}_\omega) \to (T \Rightarrow \mathfrak{L}_\omega)$ that occurs as the interpretation of a term will have the form:

$$F(s, X) = \lambda t.\ A(s,t) \cup \bigcup_{t' \in T} B(s,t,t') \cdot X(t')$$

Affine Functions

For higher-order fixpoints:

Consider $f : (\tau \to o) \to (\tau \to o)$, then

$$\llbracket f \rrbracket_\omega : \llbracket \tau \to o \rrbracket_* \times (\llbracket \tau \rrbracket_* \times \llbracket \tau \rrbracket_\omega \Rightarrow \mathfrak{L}_\omega) \to (\llbracket \tau \rrbracket_* \times \llbracket \tau \rrbracket_\omega \Rightarrow \mathfrak{L}_\omega)$$

A function $F : S \times (T \Rightarrow \mathfrak{L}_\omega) \to (T \Rightarrow \mathfrak{L}_\omega)$ that occurs as the interpretation of a term will have the form:

$$F(s, X) = \lambda t.\ A(s,t) \cup \bigcup_{t' \in T} B(s, t, t') \cdot X(t')$$

Then

$$\mathrm{gfp}(F(s, -))(t) = \bigcup_{\substack{(t_k) \in T^{\mathbb{N}} \\ t_0 = t}} \prod_{i=0}^{\infty} B(s, t_i, t_{i+1})$$

$$\cup \quad \bigcup_{t_1, \dots, t_n \in T} B(s, t, t_1) \cdot B(s, t_1, t_2) \cdots B(s, t_{n-1}, t_n) \cdot A(s, t_n)$$

ω-semigroups (Perrin, Pin)

An *ω-semigroup* is a pair of sets $\mathcal{S} = (\mathcal{S}_+, \mathcal{S}_\omega)$ equipped with:

- a mapping $\mathcal{S}_+ \times \mathcal{S}_+ \to \mathcal{S}_+$ called *binary product*
- a mapping $\mathcal{S}_+ \times \mathcal{S}_\omega \to \mathcal{S}_\omega$ called *mixed product*
- a mapping $\pi : \mathcal{S}_+^{\mathbb{N}} \to \mathcal{S}_\omega$ called *infinite product*

such that

- $\mathcal{S}_+$ with the binary product is a semigroup
- for each $s, t \in \mathcal{S}_+$ and $u \in \mathcal{S}_\omega$, $s(tu) = (st)u$
- for every increasing sequence $(k_n)_n \in \mathbb{N}^{\mathbb{N}}$ and $(s_n)_n \in \mathcal{S}_+^{\mathbb{N}}$, one has $\pi((s_n)_n) = \pi((t_n)_n)$ where $t_0 = s_0 s_1 \ldots s_{k_0}$ and $t_{n+1} = s_{k_n+1} \ldots s_{k_{n+1}}$
- $s \cdot \pi(s_0, s_1, s_2, \ldots) = \pi(s, s_0, s_1, s_2, \ldots)$

ω-semigroups (Perrin, Pin)

An *ω-semigroup* is a pair of sets $\mathcal{S} = (\mathcal{S}_+, \mathcal{S}_\omega)$ equipped with:

- a mapping $\mathcal{S}_+ \times \mathcal{S}_+ \to \mathcal{S}_+$ called *binary product*
- a mapping $\mathcal{S}_+ \times \mathcal{S}_\omega \to \mathcal{S}_\omega$ called *mixed product*
- a mapping $\pi : \mathcal{S}_+^{\mathbb{N}} \to \mathcal{S}_\omega$ called *infinite product*

such that

- $\mathcal{S}_+$ with the binary product is a semigroup
- for each $s, t \in \mathcal{S}_+$ and $u \in \mathcal{S}_\omega$, $s(tu) = (st)u$
- for every increasing sequence $(k_n)_n \in \mathbb{N}^{\mathbb{N}}$ and $(s_n)_n \in \mathcal{S}_+^{\mathbb{N}}$, one has $\pi((s_n)_n) = \pi((t_n)_n)$ where $t_0 = s_0 s_1 \dots s_{k_0}$ and $t_{n+1} = s_{k_n+1} \dots s_{k_{n+1}}$
- $s \cdot \pi(s_0, s_1, s_2, \dots) = \pi(s, s_0, s_1, s_2, \dots)$

Remark: An ω-semigroup is in particular a *Wilke algebra*.

$\mathfrak{M}$ is an ω-semigroup

Examples of ω-semigroups:

- $(\Sigma^+, \Sigma^\omega)$ with the usual products

$\mathfrak{M}$ is an ω-semigroup

Examples of ω-semigroups:

- $(\Sigma^+, \Sigma^\omega)$ with the usual products
- $(\mathfrak{L}_+, \mathfrak{L}_\omega)$ with the usual products

$\mathfrak{M}$ is an ω-semigroup

Examples of ω-semigroups:

- $(\Sigma^+, \Sigma^\omega)$ with the usual products
- $(\mathfrak{L}_+, \mathfrak{L}_\omega)$ with the usual products
- $(\mathfrak{M}_+, \mathfrak{M}_\omega)$: the infinitary product is defined as follows.

  Given $(s_n) \in \mathfrak{M}_+^{\mathbb{N}}$, define

$$\pi((s_n)_n) = \alpha_\omega(\prod_{n=0}^{\infty} \gamma_*(s_n))$$

$\mathfrak{M}$ is an ω-semigroup

Examples of ω-semigroups:

- $(\Sigma^+, \Sigma^\omega)$ with the usual products
- $(\mathfrak{L}_+, \mathfrak{L}_\omega)$ with the usual products
- $(\mathfrak{M}_+, \mathfrak{M}_\omega)$: the infinitary product is defined as follows.

  Given $(s_n) \in \mathfrak{M}_+^{\mathbb{N}}$, define

$$\pi((s_n)_n) = \alpha_\omega(\prod_{n=0}^{\infty} \gamma_*(s_n))$$

Proposition

The abstraction function $\alpha : \mathfrak{L} \to \mathfrak{M}$ is a morphism of ω-semigroups. In particular, for $(L_n)_{n\in\mathbb{N}}$ a family of languages,

$$\alpha_\omega(\prod_{i=0}^{\infty} L_n) = \pi((\alpha_*(L_n))_n)$$

Back to affine functions

Idea:

Restrict to the sub-category of **GFP**

- whose objects are of the form $(X_*, \mathfrak{L}_\omega^{X_{\mathrm{arg}}})$
- whose morphisms $f : X \to Y$ have an infinitary component $f_\omega : X_* \times \mathfrak{L}_\omega^{X_{\mathrm{arg}}} \to \mathfrak{L}_\omega^{Y_{\mathrm{arg}}}$ which is affine w.r.t. its second argument.

Back to affine functions

Idea:

Restrict to the sub-category of **GFP**

- whose objects are of the form $(X_*, \mathfrak{L}_\omega^{X_{\mathrm{arg}}})$
- whose morphisms $f : X \to Y$ have an infinitary component $f_\omega : X_* \times \mathfrak{L}_\omega^{X_{\mathrm{arg}}} \to \mathfrak{L}_\omega^{Y_{\mathrm{arg}}}$ which is affine w.r.t. its second argument.

What is an affine function ?

Back to affine functions

Idea:

Restrict to the sub-category of **GFP**

- whose objects are of the form $(X_*, \mathfrak{L}_\omega^{X_{\mathrm{arg}}})$
- whose morphisms $f : X \to Y$ have an infinitary component $f_\omega : X_* \times \mathfrak{L}_\omega^{X_{\mathrm{arg}}} \to \mathfrak{L}_\omega^{Y_{\mathrm{arg}}}$ which is affine w.r.t. its second argument.

What is an affine function ?

$\longrightarrow$ a function of the form $f(x) = ax + b$.

Back to affine functions

Idea:

Restrict to the sub-category of **GFP**

- whose objects are of the form $(X_*, \mathfrak{L}_\omega^{X_{\mathrm{arg}}})$
- whose morphisms $f : X \to Y$ have an infinitary component $f_\omega : X_* \times \mathfrak{L}_\omega^{X_{\mathrm{arg}}} \to \mathfrak{L}_\omega^{Y_{\mathrm{arg}}}$ which is affine w.r.t. its second argument.

What is an affine function ?

$\longrightarrow$ a function of the form $f(x) = ax + b$.

$\longrightarrow$ a pair (a, b).

The category $\mathbf{AFF}_{\mathcal{S}}$

Let $\mathcal{S} = (\mathcal{S}_+, \mathcal{S}_\omega)$ be an ω-semigroup.

- Objects are pairs (X_*, X_{arg})
- A morphism $f : X \to Y$ is given by
  - $f_* : X_* \to Y_*$
  - $f_{\mathrm{arg}} : X_* \times Y_{\mathrm{arg}} \to \mathcal{S}_\omega \times \mathcal{S}_*^{X_{\mathrm{arg}}^{\mathrm{op}}}$

The category $\mathbf{AFF}_{\mathcal{S}}$

Let $\mathcal{S} = (\mathcal{S}_+, \mathcal{S}_\omega)$ be an ω-semigroup.

- Objects are pairs (X_*, X_{arg})
- A morphism $f : X \to Y$ is given by
  - $f_* : X_* \to Y_*$
  - $f_{\mathrm{arg}} : X_* \times Y_{\mathrm{arg}} \to \mathcal{S}_\omega \times \mathcal{S}_*^{X_{\mathrm{arg}}^{\mathrm{op}}}$

Notation: we decompose f_{arg} in two components
$f_c : X_* \times Y_{\mathrm{arg}} \to \mathcal{S}_\omega$ and $f_p : X_* \times Y_{\mathrm{arg}} \times X_{\mathrm{arg}}^{\mathrm{op}} \to \mathcal{S}_*$

The category $\mathbf{AFF}_{\mathcal{S}}$

Let $\mathcal{S} = (\mathcal{S}_+, \mathcal{S}_\omega)$ be an ω-semigroup.

- Objects are pairs (X_*, X_{arg})
- A morphism $f : X \to Y$ is given by
  - $f_* : X_* \to Y_*$
  - $f_{\mathrm{arg}} : X_* \times Y_{\mathrm{arg}} \to \mathcal{S}_\omega \times \mathcal{S}_*^{X_{\mathrm{arg}}^{\mathrm{op}}}$

Notation: we decompose f_{arg} in two components $f_c : X_* \times Y_{\mathrm{arg}} \to \mathcal{S}_\omega$ and $f_p : X_* \times Y_{\mathrm{arg}} \times X_{\mathrm{arg}}^{\mathrm{op}} \to \mathcal{S}_*$

There is a functor $\mathrm{Ext} : \mathbf{AFF}_{\mathcal{S}} \to \mathbf{GFP}$ defined as:

- $\mathrm{Ext}(X_*, X_{\mathrm{arg}}) = (X_*, \mathcal{S}_\omega^{X_{\mathrm{arg}}})$
- $\mathrm{Ext}(f_*, f_{\mathrm{arg}}) = (f_*, f_\omega)$ where $f_\omega : X_* \times \mathcal{S}_\omega^{X_{\mathrm{arg}}} \to \mathcal{S}_\omega^{Y_{\mathrm{arg}}}$ is defined as

$$f_\omega(x, X, \eta) = f_c(x, \eta) \cup \bigcup_{\xi \in X_{\mathrm{arg}}} f_p(x, \eta, \xi) \cdot X(\xi)$$

The category $\mathbf{AFF}_{\mathcal{S}}$

Composition is defined so that $\mathrm{Ext}(g \circ f) = \mathrm{Ext}(g) \circ \mathrm{Ext}(f)$.

The category $\mathbf{AFF}_{\mathcal{S}}$

Composition is defined so that $\mathsf{Ext}(g \circ f) = \mathsf{Ext}(g) \circ \mathsf{Ext}(f)$.

The cartesian product $(X \times Y)$ is given by:

- $(X \times Y)_* = X_* \times Y_*$
- $(X \times Y)_{\mathrm{arg}} = X_{\mathrm{arg}} + Y_{\mathrm{arg}}$

The category $\mathbf{AFF}_{\mathcal{S}}$

Composition is defined so that $\mathsf{Ext}(g \circ f) = \mathsf{Ext}(g) \circ \mathsf{Ext}(f)$.

The cartesian product $(X \times Y)$ is given by:

- $(X \times Y)_* = X_* \times Y_*$
- $(X \times Y)_{\mathrm{arg}} = X_{\mathrm{arg}} + Y_{\mathrm{arg}}$

The function space $(X \Rightarrow Y)$ is given by:

- $(X \Rightarrow Y)_* = X_* \Rightarrow (Y_* \times \mathcal{S}_*^{Y_{\mathrm{arg}} \times X_{\mathrm{arg}}^{\mathrm{op}}})$
- $(X \Rightarrow Y)_{\mathrm{arg}} = X_* \times Y_{\mathrm{arg}}$

The category $\textbf{AFF}_{\mathcal{S}}$

Composition is defined so that $\text{Ext}(g \circ f) = \text{Ext}(g) \circ \text{Ext}(f)$.

The cartesian product $(X \times Y)$ is given by:

- $(X \times Y)_* = X_* \times Y_*$
- $(X \times Y)_{\text{arg}} = X_{\text{arg}} + Y_{\text{arg}}$

The function space $(X \Rightarrow Y)$ is given by:

- $(X \Rightarrow Y)_* = X_* \Rightarrow (Y_* \times \mathcal{S}_*^{Y_{\text{arg}} \times X_{\text{arg}}^{\text{op}}})$
- $(X \Rightarrow Y)_{\text{arg}} = X_* \times Y_{\text{arg}}$

Proposition

The category $\textbf{AFF}_{\mathcal{S}}$ is cartesian-closed.

Affine Semantics

Base type: $[\![o]\!] = (\mathcal{S}_*, \{\star\})$

Affine Semantics

Base type: $[\![o]\!] = (\mathcal{S}_*, \{\star\})$

Terms:

- $[\![\underline{a}]\!]_*(\star) = a$

  $[\![\underline{a}]\!]_{\mathrm{arg}}(\star) = (\varnothing, \varnothing)$
- $[\![+]\!]_*(s_1, s_2) = s_1 \cup s_2$

  $[\![+]\!]_{\mathrm{arg}}(s_1, s_2, \star) = (\varnothing, \lambda\eta.\, \varepsilon)$
- $[\![\,;\,]\!]_*(s_1, s_2) = s_1 s_2$

  $[\![\,;\,]\!]_{\mathrm{arg}}(s_1, s_2, \star) = \left(\varnothing, \lambda\eta.\, \mathrm{case}(\eta) \begin{cases} \mathrm{inl}\, \star \mapsto \varepsilon \\ \mathrm{inr}\, \star \mapsto s_1 \end{cases}\right)$

Affine Semantics

Base type: $[\![o]\!] = (\mathcal{S}_*, \{\star\})$

Terms:

- $[\![\underline{a}]\!]_*(\star) = a$
  $[\![\underline{a}]\!]_{\mathrm{arg}}(\star) = (\varnothing, \varnothing)$
- $[\![+]\!]_*(s_1, s_2) = s_1 \cup s_2$
  $[\![+]\!]_{\mathrm{arg}}(s_1, s_2, \star) = (\varnothing, \lambda\eta.\, \varepsilon)$
- $[\![\,;\,]\!]_*(s_1, s_2) = s_1 s_2$
  $[\![\,;\,]\!]_{\mathrm{arg}}(s_1, s_2, \star) = \left(\varnothing, \lambda\eta.\, \mathrm{case}(\eta) \begin{cases} \mathrm{inl}\, \star \mapsto \varepsilon \\ \mathrm{inr}\, \star \mapsto s_1 \end{cases}\right)$

Remarks:

- One needs an element $a \in \mathcal{S}_*$: pick $\{a\}$ for $\mathfrak{L}_*$ and $[a]$ for $\mathfrak{M}_*$.

Affine Semantics

Base type: $[\![o]\!] = (\mathcal{S}_*, \{\star\})$

Terms:

- $[\![\underline{a}]\!]_*(\star) = a$
  $[\![\underline{a}]\!]_{\mathrm{arg}}(\star) = (\varnothing, \varnothing)$
- $[\![+]\!]_*(s_1, s_2) = s_1 \cup s_2$
  $[\![+]\!]_{\mathrm{arg}}(s_1, s_2, \star) = (\varnothing, \lambda\eta.\, \varepsilon)$
- $[\![\,;\,]\!]_*(s_1, s_2) = s_1\, s_2$

$$[\![\,;\,]\!]_{\mathrm{arg}}(s_1, s_2, \star) = \left(\varnothing, \lambda\eta.\, \mathrm{case}(\eta) \begin{cases} \mathrm{inl}\,\star \mapsto \varepsilon \\ \mathrm{inr}\,\star \mapsto s_1 \end{cases}\right)$$

Remarks:

- One needs an element $a \in \mathcal{S}_*$: pick $\{a\}$ for $\mathfrak{L}_*$ and $[a]$ for $\mathfrak{M}_*$.
- The fixpoint operator can be defined accordingly.

Putting it all together

Theorem

For every program e*, we have* $[\![e]\!]^{\textbf{\textit{GFP}}} = \mathsf{Ext}([\![e]\!]^{\mathfrak{L}})$.

Putting it all together

Theorem

For every program e*, we have* $[\![e]\!]^{\textit{\textbf{GFP}}} = \mathsf{Ext}([\![e]\!]^{\mathfrak{L}})$.

Corollary

For every program e*,* $[\![e]\!]^{\mathfrak{L}} = (L_*(e), L_\omega(e))$.

Putting it all together

Theorem

For every program e*, we have* $[\![e]\!]^{\textbf{\textit{GFP}}} = \mathsf{Ext}([\![e]\!]^{\mathfrak{L}})$*.*

Corollary

For every program e*,* $[\![e]\!]^{\mathfrak{L}} = (L_*(e), L_\omega(e))$*.*

Theorem

For every program e*,* $\alpha([\![e]\!]^{\mathfrak{L}}) = [\![e]\!]^{\mathfrak{M}}$*.*

Putting it all together

Theorem

For every program e*, we have* $[\![e]\!]^{\textbf{\textit{GFP}}} = \mathsf{Ext}([\![e]\!]^{\mathfrak{L}})$.

Corollary

For every program e*,* $[\![e]\!]^{\mathfrak{L}} = (L_*(e), L_\omega(e))$.

Theorem

For every program e*,* $\alpha([\![e]\!]^{\mathfrak{L}}) = [\![e]\!]^{\mathfrak{M}}$.

Corollary

Let e *be a program, and write* $[\![e]\!]^{\mathfrak{M}} = (X_*, X_\omega)$.
Then $L_{*/\omega}(e) \subseteq L_{*/\omega}(\mathcal{A}) \iff X_{*/\omega} \sqsubseteq \alpha_{*/\omega}(L_{*/\omega}(\mathcal{A}))$.
Moreover, $[\![e]\!]^{\mathfrak{M}}$ *is effectively computable.*

Thanks !