A new algorithm for Higher-order model checking

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

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

- Syntax of expressions:

$$e ::= a \mid f \mid e_1; e_2 \mid e_1 + e_2$$

where $a \in \Sigma$ and $f \in \mathcal{F}$
For first order programs (M. Hofmann & W. Chen)

Let \( \Sigma \) be a set of events and \( \mathcal{F} \) a set of procedure identifiers.

- Syntax of expressions:
  \[
e ::= a \mid f \mid e_1; e_2 \mid e_1 + e_2 \quad \text{where} \ a \in \Sigma \ \text{and} \ f \in \mathcal{F}
\]

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

Examples:

\[
\begin{align*}
f & = a; b; g \\
g & = d + (c; f)
\end{align*}
\]

\[
L(f) = (abc)^* abd \cup \{ (abc)^\omega \}
\]
For first order programs (M. Hofmann & W. Chen)

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

- Syntax of expressions:

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

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

Examples:

$$f = a; b; g \quad u = a; v$$

$$g = d + (c; f) \quad v = v$$

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

$$L(u) = \{a\}$$
For first order programs (M. Hofmann & W. Chen)

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

- **Syntax of expressions:**
  
  $$e ::= a \mid f \mid e_1; e_2 \mid e_1 + e_2$$
  
  where $a \in \Sigma$ and $f \in \mathcal{F}$

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

**Examples:**

$$f = a; b; g$$

$$g = d + (c; f)$$

$$u = a; v$$

$$v = v$$

$L_*(f) = (ab\checkmark c\checkmark)^*ab\checkmark d$  
$L_*(u) = \emptyset$  
$L_\omega(f) = \{(ab\checkmark c\checkmark)^\omega\}$  
$L_\omega(u) = \{a(\checkmark)^\omega\}$
Policy Automaton

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

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

f = g; b; f

f = (a; g) + c
```
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; b; f

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:

$$f = a; f$$

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

$$f : GFb \vdash (a; f) : GFb$$
Let $GFb = (a^* b)^\omega$ be a type asserting “$b$ occurs infinitely often”.

Consider the procedure:

$$f = a; f$$

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

$$f : GFb \vdash (a; f) : GFb$$

Under “usual” typing rules, this would allow us to establish

$$\vdash f : GFb$$

which is clearly wrong.
Büchi type system

Idea:

\[
\begin{align*}
  f : X & \vdash e_f : T(X) \\
  \vdash f : \text{gfp}(\lambda X. T(X))
\end{align*}
\]
Büchi type system

Idea:

\[
\begin{align*}
f : X & \vdash e_f : T(X) \\
\vdash f : \text{gfp}(\lambda X. T(X))
\end{align*}
\]

\[f = (a; f) + b\]

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:

\[
\begin{align*}
&f : X \vdash e_f : T(X) \\
&\vdash f : \text{gfp}(\lambda X. T(X))
\end{align*}
\]

\[
f = (a; f) + b
\]

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:

\[
T(X) = U \cdot X + V
\]

\[
\text{gfp}(T) = U^* V + U^\omega
\]
Büchi Abstraction

Let $\mathcal{L}_* = \mathcal{P}(\Sigma^*)$ and $\mathcal{L}_\omega = \mathcal{P}(\Sigma^\omega)$.

Given the policy automaton $\mathcal{A}$, we can construct complete lattices $\mathcal{M}_*$ and $\mathcal{M}_\omega$ such that:

- They are finite.

The abstraction function $\alpha$ preserves unions, concatenation, least fixpoints and $\omega$-iteration (but not greatest fixpoints!).
Büchi Abstraction

Let \( \mathcal{L}_* = \mathcal{P}(\Sigma^*) \) and \( \mathcal{L}_\omega = \mathcal{P}(\Sigma^\omega) \).

Given the policy automaton \( \mathcal{A} \), we can construct complete lattices \( \mathcal{M}_* \) and \( \mathcal{M}_\omega \) such that:

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

\[
\gamma_*/\omega(\alpha_*/\omega(L)) \supseteq L \quad \text{and} \quad \alpha_*/\omega(\gamma_*/\omega(U)) = U
\]

The abstraction function \( \alpha \) preserves unions, concatenation, least fixpoints and \( \omega \)-iteration (but not greatest fixpoints!).
Büchi Abstraction

Let $\mathcal{L}_* = \mathcal{P}(\Sigma^*)$ and $\mathcal{L}_\omega = \mathcal{P}(\Sigma^\omega)$.

Given the policy automaton $A$, we can construct complete lattices $\mathcal{M}_*$ and $\mathcal{M}_\omega$ such that:

- They are finite.
- They are related to $\mathcal{L}_*$, $\mathcal{L}_\omega$ by a galois insertion. There are $\alpha_{*/\omega} : \mathcal{L}_{*/\omega} \rightarrow \mathcal{M}_{*/\omega}$ and $\gamma_{*/\omega} : \mathcal{M}_{*/\omega} \rightarrow \mathcal{L}_{*/\omega}$ such that
  \[
  \gamma_{*/\omega}(\alpha_{*/\omega}(L)) \supseteq L \quad \text{and} \quad \alpha_{*/\omega}(\gamma_{*/\omega}(U)) = U
  \]
- $L \subseteq L(A) \iff \alpha(L) \sqsubseteq \alpha(L(A))$
Let $\mathcal{L}_* = \mathcal{P}(\Sigma^*)$ and $\mathcal{L}_\omega = \mathcal{P}(\Sigma^\omega)$. Given the policy automaton $\mathcal{A}$, we can construct complete lattices $\mathcal{M}_*$ and $\mathcal{M}_\omega$ such that:

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

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

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

- The abstraction function $\alpha$ preserves unions, concatenation, least fixpoints and $\omega$-iteration (but not greatest fixpoints !):

$$
\begin{align*}
\mathcal{L}_* & \xrightarrow{(-)^\omega} \mathcal{L}_\omega \\
\mathcal{M}_* & \xrightarrow{(-)^\omega} \mathcal{M}_\omega
\end{align*}
$$

\[\begin{bmatrix}
\alpha_* \\
\gamma_* \\
\alpha_*/\omega \\
\gamma_*/\omega
\end{bmatrix}
\]
Define the equivalence relation $\sim_A$ on $\Sigma^+$ as follows: $u \sim_A v$ iff

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

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

▶ Equivalence classes are regular languages.
▶ There's a finite number of classes.
▶ For every class $C$, either $C \cap L(A) = \emptyset$ or $C \subseteq L(A)$.
▶ For every $C, D$, either $C \omega \cap D \omega = \emptyset$ or $C \omega \subseteq D \omega$.
▶ For every $w \in \Sigma^\omega$, there are $C, D$ such that $w \in C \omega$.
Define the equivalence relation $\sim_A$ on $\Sigma^+$ as follows: $u \sim_A v$ iff

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

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

- Equivalence classes are regular languages.
- There’s a finite number of classes.
Büchi Abstraction

Define the equivalence relation $\sim_A$ on $\Sigma^+$ as follows: $u \sim_A v$ iff
\[
\forall q, q'. (q \xrightarrow{u} q' \iff q \xrightarrow{v} q') \land (q \xrightarrow{u} F q' \iff q \xrightarrow{v} F q')
\]
and extend it to $\Sigma^*$ such that $[\varepsilon] = \{\varepsilon\}$.

- Equivalence classes are regular languages.
- There’s a finite number of classes.
- For every class $C$, either $C \cap L_*(A) = \emptyset$ or $C \subseteq L_*(A)$.
Büchi Abstraction

Define the equivalence relation $\sim_A$ on $\Sigma^+$ as follows: $u \sim_A v$ iff

\[
\forall q, q'. (q \xrightarrow{u} q' \iff q \xrightarrow{v} q') \land (q \xrightarrow{u} F q' \iff q \xrightarrow{v} F q')
\]

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

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

The sets $CD^\omega$ behave almost like classes, but they may overlap!
Büchi Abstraction

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

\[
\begin{align*}
\gamma_*(\mathcal{V}) &= \bigcup_{C \in \mathcal{V}} C \\
\alpha_*(L) &= \{ C \mid C \cap L \neq \emptyset \}
\end{align*}
\]
Büchi Abstraction

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

\[
\gamma_*(\mathcal{V}) = \bigcup_{C \in \mathcal{V}} C
\]
\[
\alpha_*(L) = \{ C \mid C \cap L \neq \emptyset \}
\]

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

\[
\gamma_\omega(\mathcal{V}) = \bigcup_{(C,D) \in \mathcal{V}} CD_\omega
\]
\[
\alpha_\omega(L) = \text{cl} \{(C, D) \mid CD_\omega \cap L \neq \emptyset\}
Extending to Higher-order

Terms:

\[ e ::= x | a | e_1; e_2 | e_1 + e_2 | \text{fix } e | \lambda x. e | e_1 \ e_2 \]
Extending to Higher-order

Terms:

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

Types:

\[ \tau ::= o \mid \tau_1 \rightarrow \tau_2 \]

Typing rules:

\[
\begin{array}{c}
\Gamma \vdash x : \Gamma(x) \\
\hline
\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1 \\
\Gamma \vdash e_1 \; e_2 : \tau_2 \\
\Gamma, x : \tau_1 \vdash e : \tau_2 \\
\Gamma \vdash \lambda x. e : \tau_1 \rightarrow \tau_2 \\
\Gamma \vdash e : \tau \rightarrow \tau \\
\Gamma \vdash \text{fix } e : \tau \\
\Gamma \vdash a : o \\
\Gamma \vdash e_1 + e_2 : o \\
\Gamma \vdash e_1; e_2 : o \\
\Gamma \vdash e_1 : o \quad \Gamma \vdash e_2 : o \\
\Gamma \vdash e_1 : o \quad \Gamma \vdash e_2 : o \\
\Gamma \vdash e_1; e_2 : o
\end{array}
\]
Extending to Higher-order

Terms:

\[ e ::= x \mid a \mid e_1 \cdot e_2 \mid e_1 + e_2 \mid \text{fix } e \mid \lambda x. \ e \mid e_1 \ e_2 \]

Types:

\[ \tau ::= o \mid \tau_1 \to \tau_2 \]

Typing rules:

\[
\begin{array}{c}
\frac{\Gamma \vdash x : \Gamma(x)}{\Gamma \vdash x : \tau} \\
\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 \ e_2 : \tau_2} \\
\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x. e : \tau_1 \to \tau_2} \\
\frac{\Gamma \vdash e : \tau \to \tau}{\Gamma \vdash \text{fix } e : \tau} \\
\frac{\Gamma \vdash e_1 : o \quad \Gamma \vdash e_2 : o}{\Gamma \vdash e_1 + e_2 : o} \\
\frac{\Gamma \vdash a : o}{\Gamma \vdash \text{fix } e : \tau} \\
\end{array}
\]

Program: closed term of type \( o \).
Examples

First order: only use fix : (o → o) → o.

- fix(λf. (a; f) + b)
- fix(λf. a; b; fix(λg. d + (c; f)))
Examples

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

$\text{fix}(\lambda f. (a; f) + b)$

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

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

$\text{fix}(\lambda x. a; x) b \quad \rightarrow \quad L_\ast(e) = \{ab\}$
Examples

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

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

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

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

Non context-free examples:

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

\[
L_*(e'\ d) = \{a^n b^n d c^n \mid n \geq 0\} \quad L_\omega(e'\ d) = \{a^\omega\}\
\]
Examples

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

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

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

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

Non context-free examples:

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

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

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

$$L_*(e'') = \emptyset \quad L_\omega(e'') = (L_*(e' d))^{\omega} \cup \{a^{\omega}\}$$
Related Work

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

- $\lambda Y$, higher-order recursion schemes, higher-order pushdown automata with collapse.
- Model-checking of temporal logic, $\mu$-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).

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

**Example:** $\lambda Y$.

Choose first-order constants

- $a : o \rightarrow o \rightarrow o$
- $b : o \rightarrow o$
- $c : o$

$M = Y(\lambda f. \lambda x. a \ x \ (f \ (b \ x)))$

Böhm-tree of $(M \ c)$:
GFP semantics

We define the category GFP

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

We define the category **GFP**

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

$$\text{fix}_A : (A \Rightarrow A) \rightarrow A$$

where

- $(\text{fix}_A)^*(f^*) = \text{lfp}(f^*)$
- $(\text{fix}_A)^\omega(f^*, f^\omega) = \text{gfp}(\lambda a^\omega. f^\omega(\text{lfp}(f^*), a^\omega))$

**Proposition**

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

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

Interpretation of types:
To every type $\tau$, associate an object $[\tau]$ of GFP

$$[o] = (\mathcal{L}_*, \mathcal{L}_\omega) \quad \text{and} \quad [\sigma \to \tau] = [\sigma] \Rightarrow [\tau]$$
GFP semantics

Interpretation of types:
To every type $\tau$, associate an object $[\tau]$ of GFP

$$[\circ] = (\mathcal{L}_*, \mathcal{L}_\omega) \quad \text{and} \quad [\sigma \to \tau] = [\sigma] \Rightarrow [\tau]$$

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 $\tau$, associate an object $[\tau]$ of GFP

$$[o] = (L_*, L_\omega) \quad \text{and} \quad [\sigma \rightarrow \tau] = [\sigma] \Rightarrow [\tau]$$

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 $[e] : [\Gamma] \rightarrow [\tau]$
GFP semantics

Interpretation of types:
To every type $\tau$, associate an object $[\tau]$ of GFP

$$[o] = (L_*, L_\omega) \quad \text{and} \quad [\sigma \rightarrow \tau] = [\sigma] \Rightarrow [\tau]$$

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 $[e] : [\Gamma] \rightarrow [\tau]$

$$[a] = (\{a\}, \emptyset)$$
**GFP semantics**

**Interpretation of types:**
To every type \( \tau \), associate an object \([\tau]\) of \( \text{GFP} \)

\[
[\nu] = (\mathcal{L}_*, \mathcal{L}_\omega) \quad \text{and} \quad [\sigma \to \tau] = [\sigma] \Rightarrow [\tau]
\]

**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 \([e] : [\Gamma] \to [\tau] \)

- \([a] = (\{a\}, \emptyset)\)
- \([\oplus]_*(X_*, Y_*) = X_* \cup Y_*\)
- \([\oplus]_\omega(X_*, Y_*, X_\omega, Y_\omega) = X_\omega \cup Y_\omega\)
GFP semantics

Interpretation of types:
To every type \( \tau \), associate an object \([\tau]\) of GFP

\[
[\sigma \rightarrow \tau] = [\sigma] \Rightarrow [\tau]
\]

Interpretation of contexts:
To a context \( \Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n \), associate the object

\[
[\Gamma] = [\tau_1] \times \cdots \times [\tau_n]
\]

Interpretation of terms:
To a derivation \( \Gamma \vdash e : \tau \), associate a morphism \([e] : [\Gamma] \rightarrow [\tau] \)

- \([a] = (\{a\}, \emptyset)\)
- \([+]* (X_*, Y_*) = X_* \cup Y_*\)
  \([+]* (X_*, Y_*, X_\omega, Y_\omega) = X_\omega \cup Y_\omega\)
- \([;]* (X_*, Y_*) = X_* Y_*\)
  \([;]* (X_*, Y_*, X_\omega, Y_\omega) = X_\omega \cup X_* Y_\omega\)
Reminder: a program is a closed term of type $o$.

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

**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 $\llbracket e \rrbracket : 1 \to \llbracket o \rrbracket$ is (isomorphic to) an element of $\mathcal{L}_* \times \mathcal{L}_\omega$.

**Theorem**

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

If we choose $\llbracket o \rrbracket = (M_*, M_\omega)$ instead, everything is computable.

But $\alpha$ doesn’t commute with greatest fixpoints :-(
Affine Functions

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

- $[f]_* : \mathcal{L}_* \rightarrow \mathcal{L}_*$
- $[f]_\omega : \mathcal{L}_* \times \mathcal{L}_\omega \rightarrow \mathcal{L}_\omega$

$[\text{fix } f]$ involves some gfp of $[f]_\omega$. 
Affine Functions

For first-order fixpoints:
The denotation of \( f : o \rightarrow o \) has two components:

\[ [f]_* : \mathcal{L}_* \rightarrow \mathcal{L}_* \]
\[ [f]_\omega : \mathcal{L}_* \times \mathcal{L}_\omega \rightarrow \mathcal{L}_\omega \]

\([\text{fix } f]\) involves some gfp of \([f]_\omega\).

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

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

Then \( \text{gfp}(F(x, -)) = A(x)^* B(x) \cup A(x)^\omega \) commutes with \( \alpha \).
Affine Functions

For higher-order fixpoints:

Consider \( f : (\tau \rightarrow o) \rightarrow (\tau \rightarrow o) \), then

\[
[f]_\omega : \mathcal{T}_\omega \times (\mathcal{T}_\omega \times \mathcal{T}_\omega \Rightarrow \mathcal{L}_\omega) \rightarrow (\mathcal{T}_\omega \times \mathcal{T}_\omega \Rightarrow \mathcal{L}_\omega)
\]
Affine Functions

For higher-order fixpoints:

Consider \( f : (\tau \rightarrow o) \rightarrow (\tau \rightarrow o) \), then

\[
[f]_\omega : [\tau \rightarrow o]_* \times ([\tau]_* \times [\tau]_\omega \Rightarrow \mathcal{L}_\omega) \rightarrow ([\tau]_* \times [\tau]_\omega \Rightarrow \mathcal{L}_\omega)
\]

A function \( F : S \times (T \Rightarrow \mathcal{L}_\omega) \rightarrow (T \Rightarrow \mathcal{L}_\omega) \) 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

\[
[f]_\omega : [\tau \to o]_* \times ([\tau]_* \times [\tau]_\omega \Rightarrow L_\omega) \to ([\tau]_* \times [\tau]_\omega \Rightarrow L_\omega)
\]

A function \( F : S \times (T \Rightarrow L_\omega) \to (T \Rightarrow L_\omega) \) 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

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

\[
\quad \quad \cup \bigcup_{t_1, \ldots, t_n \in T} B(s, t, t_1) \cdot B(s, t_1, t_2) \cdots B(s, t_{n-1}, t_n) \cdot A(s, t_n)
\]
$\omega$-semigroups (Perrin, Pin)

An $\omega$-semigroup is a pair of sets $S = (S_+, S_\omega)$ equipped with:

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

such that

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

Remark: An $\omega$-semigroup is in particular a Wilke algebra.
ω-semigroups (Perrin, Pin)

An ω-semigroup is a pair of sets $S = (S_+, S_\omega)$ equipped with:

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

such that

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

Remark: An ω-semigroup is in particular a Wilke algebra.
\( M \) is an \( \omega \)-semigroup

Examples of \( \omega \)-semigroups:

\begin{itemize}
  \item (\( \Sigma^+, \Sigma^\omega \)) with the usual products
\end{itemize}
\( M \) is an \( \omega \)-semigroup

Examples of \( \omega \)-semigroups:

- \((\Sigma^+, \Sigma^\omega)\) with the usual products
- \((\mathcal{L}_+, \mathcal{L}_\omega)\) with the usual products
\( \mathcal{M} \) is an \( \omega \)-semigroup

**Examples of \( \omega \)-semigroups:**

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

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

  \[
  \pi((s_n)_n) = \alpha_{\omega}(\prod_{n=0}^{\infty} \gamma_*(s_n))
  \]
\( \mathcal{M} \) is an \( \omega \)-semigroup

**Examples of \( \omega \)-semigroups:**

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

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

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

**Proposition**

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

\[
\alpha_\omega(\prod_{i=0}^{\infty} L_n) = \pi((\alpha_*(L_n))_n)
\]
Idea:
Restrict to the sub-category of \( \text{GFP} \)

- whose objects are of the form \((X_\star, \mathcal{L}_\omega^{X_{\text{arg}}})\)
- whose morphisms \(f : X \rightarrow Y\) have an infinitary component \(f_\omega : X_\star \times \mathcal{L}_\omega^{X_{\text{arg}}} \rightarrow \mathcal{L}_\omega^{Y_{\text{arg}}}\) which is affine w.r.t. its second argument.
Idea:
Restrict to the sub-category of \textbf{GFP}

- whose objects are of the form \((X_*, \mathcal{L}_X^{\arg})\)
- whose morphisms \(f : X \to Y\) have an infinitary component
\[ f_\omega : X_\ast \times \mathcal{L}_X^{\arg} \to \mathcal{L}_Y^{\arg}\]
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 \((X_*, \mathcal{L}_X^{X_{\text{arg}}})\)
- whose morphisms \(f : X \to Y\) have an infinitary component \(f_\omega : X_\ast \times \mathcal{L}_X^{X_{\text{arg}}} \to \mathcal{L}_Y^{Y_{\text{arg}}}\) which is affine w.r.t. its second argument.

What is an affine function?

\[ f(x) = ax + b. \]
Back to affine functions

Idea:
Restrict to the sub-category of GFP

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

What is an affine function?

\(\rightarrow\) a function of the form \(f(x) = ax + b\).
\(\rightarrow\) a pair \((a, b)\).
The category $\textbf{AFF}_S$

Let $S = (S_+, S_\omega)$ be an $\omega$-semigroup.

- Objects are pairs $(X_*, X_{\text{arg}})$
- A morphism $f : X \to Y$ is given by
  
  $\begin{align*}
  f_* &: X_* \to Y_* \\
  f_{\text{arg}} &: X_* \times Y_{\text{arg}} \to S_\omega \times S_{*\text{arg}}^{X_{\text{op}}}
  \end{align*}$
The category $\textbf{AFF}_S$

Let $S = (S_+, S_\omega)$ be an $\omega$-semigroup.

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

**Notation:** we decompose $f_{\text{arg}}$ in two components

$f_c : X_* \times Y_{\text{arg}} \to S_\omega$ and $f_p : X_* \times Y_{\text{arg}} \times X_{\text{arg}}^{\text{op}} \to S_*$
The category $\text{AFF}_S$

Let $S = (S_+, S_\omega)$ be an $\omega$-semigroup.

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

**Notation:** we decompose $f_{\text{arg}}$ in two components

$f_c : X_* \times Y_{\text{arg}} \to S_\omega$ and $f_p : X_* \times Y_{\text{arg}} \times X_{\text{arg}}^{\text{op}} \to S_*$

There is a functor $\text{Ext} : \text{AFF}_S \to \text{GFP}$ defined as:

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

$$f_\omega(x, X, \eta) = f_c(x, \eta) \cup \bigcup_{\xi \in X_{\text{arg}}} f_p(x, \eta, \xi) \cdot X(\xi)$$
The category $\mathbf{AFF}_S$

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

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 category $\text{AFF}_S$

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:

- $(X \Rightarrow Y)_* = X_* \Rightarrow (Y_* \times S_{Y_{\text{arg}}}^{X_{\text{arg}} \times X^\text{op}_{\text{arg}}})$
- $(X \Rightarrow Y)_{\text{arg}} = X_* \times Y_{\text{arg}}$
The category $\text{AFF}_S$

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:

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

**Proposition**

The category $\text{AFF}_S$ is cartesian-closed.
Affine Semantics

Base type:  $[o] = (S_*, \{\star\})$
Affine Semantics

Base type: \([o] = (S_*, \{\star\})\)

Terms:

- \([a]_* (\star) = a\)
- \([a]_{\text{arg}} (\star) = (\emptyset, \emptyset)\)
- \([+]_* (s_1, s_2) = s_1 \cup s_2\)
- \([+]_{\text{arg}} (s_1, s_2, \star) = (\emptyset, \lambda \eta. \varepsilon)\)
- \([;]_* (s_1, s_2) = s_1 s_2\)
- \([;]_{\text{arg}} (s_1, s_2, \star) = (\emptyset, \lambda \eta. \text{case}(\eta) \begin{cases} \text{inl} \star \mapsto \varepsilon \\ \text{inr} \star \mapsto s_1 \end{cases})\)

Remarks:
- One needs an element \(a \in S_*\): pick \(\{a\}\) for \([L]_*\) and \([L]_\emptyset\).
- The fixpoint operator can be defined accordingly.
Affine Semantics

Base type: \( \llbracket o \rrbracket = (\mathcal{S}_*, \{\ast\}) \)

Terms:

\[ \begin{align*}
\text{▶ } \llbracket a \rrbracket_\ast(\ast) & = a \\
\llbracket a \rrbracket_{\text{arg}}(\ast) & = (\emptyset, \emptyset) \\
\text{▶ } \llbracket s_1 + s_2 \rrbracket_\ast & = s_1 \cup s_2 \\
\llbracket s_1 + s_2, \ast \rrbracket_{\text{arg}} & = (\emptyset, \lambda \eta. \varepsilon) \\
\text{▶ } \llbracket s_1 \cdot s_2 \rrbracket_\ast & = s_1 \cdot s_2 \\
\llbracket s_1 \cdot s_2, \ast \rrbracket_{\text{arg}} & = (\emptyset, \lambda \eta. \text{case}(\eta) \begin{cases} \inl \ast & \mapsto \varepsilon \\
\inr \ast & \mapsto s_1 \end{cases})
\end{align*} \]

Remarks:

\[ \text{▶ } \text{One needs an element } a \in \mathcal{S}_*: \text{ pick } \{a\} \text{ for } \mathcal{L}_* \text{ and } \llbracket a \rrbracket \text{ for } \mathcal{M}_*. \]
Affine Semantics

Base type: \[ [o] = (S_*, \{\star\}) \]

Terms:

- \([a]_\star(\star) = a\)
- \([a]_\text{arg}(\star) = (\emptyset, \emptyset)\)
- \([+\mathbb{\_}]* (s_1, s_2) = s_1 \cup s_2\)
- \([+\mathbb{\_}]* \text{arg}(s_1, s_2, \star) = (\emptyset, \lambda \eta. \varepsilon)\)
- \([\_; \_]* (s_1, s_2) = s_1 s_2\)
- \([\_; \_]* \text{arg}(s_1, s_2, \star) = (\emptyset, \lambda \eta. \text{case}(\eta) \left\{ \begin{array}{ll} \text{inl} \star & \mapsto \varepsilon \\ \text{inr} \star & \mapsto s_1 \end{array} \right. )\)

Remarks:

- One needs an element \(a \in S_*\): pick \(\{a\}\) for \(L_*\) and \([a]\) for \(M_*\).
- The fixpoint operator can be defined accordingly.
Putting it all together

**Theorem**

For every program $e$, we have $[e]^GFP = \text{Ext}([e]^E)$.

**Corollary**

Let $e$ be a program, and write $[e]^K_M = (X^*, X^\omega)$. Then $L^*/\omega(e) \subseteq L^*/\omega(A) \iff X^*/\omega \sqsubseteq \alpha^*/\omega(L^*/\omega(A))$. Moreover, $[e]^K_M$ is effectively computable.
### Theorem

**For every program** $e$, we have $[e]^{GFP} = \text{Ext}([e]^L)$.

### Corollary

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

Moreover, $J_{eK}M$ is effectively computable.
Putting it all together

**Theorem**

For every program $e$, we have $\llbracket e \rrbracket^{GFP} = \text{Ext}(\llbracket e \rrbracket^L)$.

**Corollary**

For every program $e$, $\llbracket e \rrbracket^L = (L_*(e), L_\omega(e))$.

**Theorem**

For every program $e$, $\alpha(\llbracket e \rrbracket^L) = \llbracket e \rrbracket^M$.
Putting it all together

Theorem

For every program $e$, we have $\llbracket e \rrbracket^{\text{GFP}} = \text{Ext}(\llbracket e \rrbracket^{\gg})$.

Corollary

For every program $e$, $\llbracket e \rrbracket^{\gg} = (L_*(e), L_\omega(e))$.

Theorem

For every program $e$, $\alpha(\llbracket e \rrbracket^{\gg}) = \llbracket e \rrbracket^{\text{M}}$.

Corollary

Let $e$ be a program, and write $\llbracket e \rrbracket^{\text{M}} = (X_*, X_\omega)$. Then $L_{\ast/\omega}(e) \subseteq L_{\ast/\omega}(A) \iff X_{\ast/\omega} \subseteq \alpha_{\ast/\omega}(L_{\ast/\omega}(A))$. Moreover, $\llbracket e \rrbracket^{\text{M}}$ is effectively computable.
Thanks !