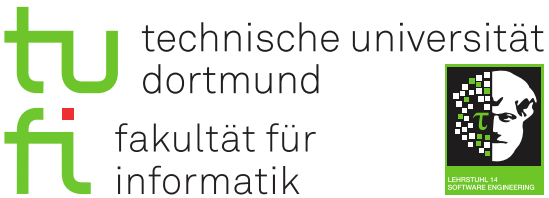
Synthesis

Jan Bessai

2017-06-21



Last Time: Pandoras Box

Real Programming Languages

- mostly informal specifications
- complex rule systems
  (ML: approx. 200 rules)
- lots and lots of variation
  (c.f. Pure Type Systems)
- sometimes "broken"
  (c.f. Java Unsoundness [1])

F.S.Church - Opening up Pandoras Box, source: wikimedia.org

Today: Equipment for Ghostbusters

source: wikimedia.org

- Language Independence with Algebra
  - mathematical abstraction of interfaces
  - implementation in any target language
- Synthesis
  - pick an inhabitation friendly target language
  - translate results
- Demo

Signatures – Formal Interfaces

Definition (Term Signatures)

Given a map $\mathbb{S} : \mathrm{Set} \to \mathrm{Set}$ and a finite set of variables $\mathbb{V}$, the term signature $\Sigma_{\mathbb{S},\mathbb{V}} = (\mathbb{O}, \mathsf{arity}, \mathsf{domain}, \mathsf{range})$ is defined by:

- A set $\mathbb{O}$ of operation symbols
- A function $\mathsf{arity} : \mathbb{O} \to \mathbb{N}^0$
- A family $\mathsf{domain} : (\prod_{i=1}^{\mathsf{arity}(op)} \mathbb{S}(\mathbb{V}))_{op \in \mathbb{O}}$
- A function $\mathsf{range} : \mathbb{O} \to \mathbb{S}(\mathbb{V})$

Signatures – Example

Informally: $\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \texttt{allUsers} : \mathrm{List}(\mathrm{String});$
$\texttt{newUser} : \mathrm{String};$
$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$
$\texttt{newUserId} : \mathrm{int};$
$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$
$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \,\}$

Or formally:

- $\mathbb{V} : \mathrm{Set}$
  $\mathbb{V} = \{\alpha\}$
- $\mathbb{S} : \mathrm{Set} \to \mathrm{Set}$
  $\mathbb{S}(X) = X \cup \{\mathrm{String}, \mathrm{int}\} \cup \{\mathrm{List}(x) \mid x \in \mathbb{S}(X)\}$

Signatures – Example

Informally: $\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \texttt{allUsers} : \mathrm{List}(\mathrm{String});$
$\texttt{newUser} : \mathrm{String};$
$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$
$\texttt{newUserId} : \mathrm{int};$
$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$
$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \,\}$

Or formally:

- $\mathbb{O} : \mathrm{Set}$
  $\mathbb{O} = \{\, \texttt{allUsers}, \texttt{newUser}, \texttt{allUserIds},$
  $\texttt{newUserId}, \texttt{append}, \texttt{first}\}$
- $\mathrm{arity} : \mathbb{O} \to \mathbb{N}^0$
  $\mathrm{arity} = \{\, \texttt{allUsers} \mapsto 0, \texttt{newUser} \mapsto 0, \texttt{allUserIds} \mapsto 0,$
  $\texttt{newUserId} \mapsto 0, \texttt{append} \mapsto 2, \texttt{first} \mapsto 1\}$

Signatures – Example

Informally: $\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \texttt{allUsers} : \mathrm{List}(\mathrm{String});$
$\texttt{newUser} : \mathrm{String};$
$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$
$\texttt{newUserId} : \mathrm{int};$
$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$
$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \,\}$

Or formally:

- domain : $(\prod_{i=1}^{\mathsf{arity}(op)} \mathbb{S}(\mathbb{V}))_{op \in \mathbb{O}}$
  domain $= \{\, \texttt{allUsers} \mapsto (), \texttt{newUser} \mapsto (),$
  $\texttt{allUserIds} \mapsto (), \texttt{newUserId} \mapsto (),$
  $\texttt{append} \mapsto (\alpha, \mathrm{List}(\alpha)), \texttt{first} \mapsto \mathrm{List}(\alpha)\}$
- range : $\mathbb{O} \to \mathbb{S}(\mathbb{V})$
  range $= \{\, \texttt{allUsers} \mapsto \mathrm{List}(\mathrm{String}), \texttt{newUser} \mapsto \mathrm{String},$
  $\texttt{allUserIds} \mapsto \mathrm{List}(\mathrm{int}), \texttt{newUserId} \mapsto \mathrm{int},$
  $\texttt{append} \mapsto \mathrm{List}(\alpha), \texttt{first} \mapsto \alpha\}$

Signatures – Substitutions

Definition (Substitution)

Given a map $\mathbb{S} : \mathrm{Set} \to \mathrm{Set}$ and a set of variables $\mathbb{V}$, we lift substitutions $S : \mathbb{V} \to \mathbb{S}(\emptyset)$ to $S^+ : \mathbb{S}(\mathbb{V}) \to \mathbb{S}(\emptyset)$ by applying a substitution map $\_^+ : (\mathbb{V} \to \mathbb{S}(\emptyset)) \to \mathbb{S}(\mathbb{V}) \to \mathbb{S}(\emptyset)$.

- We call $s \in \mathbb{S}(\emptyset)$ a closed sort and require closed sorts to be countable.
- Open sorts $s \in \mathbb{S}(\mathbb{V})$ are used to provide parametric polymorphism and closed sorts can be created from open sorts by applying substitutions.
- Example:
  - $\mathbb{V} : \mathrm{Set} \quad \mathbb{V} = \{\alpha\}$
  - $\mathbb{S} : \mathrm{Set} \to \mathrm{Set}$
    $\mathbb{S}(X) = X \cup \{\mathrm{String}, \mathrm{int}\} \cup \{\mathrm{List}(x) \mid x \in \mathbb{S}(X)\}$
  - $\_^+ : (\mathbb{V} \to \mathbb{S}(\emptyset)) \to \mathbb{S}(\mathbb{V}) \to \mathbb{S}(\emptyset)$
    $S^+ = \{\alpha \mapsto S(\alpha), \mathrm{List}(x) \mapsto \mathrm{List}(S^+(x)),$
    $\mathrm{String} \mapsto \mathrm{String}, \mathrm{int} \mapsto \mathrm{int}\}$
  - $S : \mathbb{V} \to \mathbb{S}(\emptyset) \quad S(\alpha) = \mathrm{int}$
  - $S^+(\mathrm{List}(\mathrm{List}(\alpha)) = \mathrm{List}(\mathrm{List}(int)))$

Signatures – Extensions

- subsorting: define a preorder $\leq_{\mathbb{S}} \subseteq \mathbb{S}(\emptyset) \times \mathbb{S}(\emptyset)$ on closed sorts
  - reflexivity: $s_1 \leq_{\mathbb{S}} s_1$
  - transitivity: $s_1 \leq_{\mathbb{S}} s_2$ and $s_2 \leq_{\mathbb{S}} s_3$ imply $s_1 \leq_{\mathbb{S}} s_3$
- well-formed substitution spaces: restrict the space of valid substitions, s.t. $S \in \mathrm{WF} \subseteq \mathbb{V} \to \mathbb{S}(\emptyset)$

Example:

$$\mathbb{S}(X) = X \cup \{\mathrm{Odd}, \mathrm{Even}, \mathrm{Nat}\} \quad \mathbb{V} = \{\alpha, \beta\}$$

$$\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \mathtt{one} : \mathrm{Odd};$$
$$\mathtt{succ} : \alpha \twoheadrightarrow \beta \,\}$$

- $\mathrm{Even} \leq_{\mathbb{S}} \mathrm{Nat}$ and $\mathrm{Odd} \leq_{\mathbb{S}} \mathrm{Nat}$
- $\mathrm{WF} \subseteq \mathbb{V} \to \mathbb{S}(\emptyset) \quad \mathrm{WF} = \{\, \{\alpha \mapsto \mathrm{Odd}, \beta \mapsto \mathrm{Even}\},$
  $\{\alpha \mapsto \mathrm{Even}, \beta \mapsto \mathrm{Odd}\}$
  $\{\alpha \mapsto \mathrm{Nat}, \beta \mapsto \mathrm{Nat}\}\}$

Connecting Interface and Implementation

Definition (Subsorted $\Sigma_{\mathbb{S},\mathbb{V}}$-Algebra)

Given a term signature $\Sigma_{\mathbb{S},\mathbb{V}} = (\mathbb{O}, \text{arity}, \text{domain}, \text{range})$, a sort preorder $\leq_{\mathbb{S}}$ and a function space $\mathrm{WF} \subseteq \mathbb{V} \to \mathbb{S}(\emptyset)$ of well-formed substitutions, a subsorted $\Sigma_{\mathbb{S},\mathbb{V}}$-Algebra is

- A carrier $\mathbb{C}$, which is a family of sets indexed by closed sorts $s \in \mathbb{S}(\emptyset)$ and
- A $\mathbb{S}(\emptyset)$ indexed family of morphisms $m_s : \mathbb{F}_s(\mathbb{C}) \to \mathbb{C}_s$ where

$$\mathbb{F}_s(\mathbb{C}) = \sum_{S \in \mathrm{WF}} \sum_{o \in \text{ranged}(s,S)} \prod_{i=1}^{\text{arity}(o)} \mathbb{C}_{S^+(\pi_i(\text{domain}_o))}$$

with $\text{ranged} : \mathbb{S}(\emptyset) \times \mathrm{WF} \to \mathrm{Set}$ defined by $\text{ranged}(s, S) = \{o \in \mathbb{O} \mid S^+(\text{range}(o)) \leq_{\mathbb{S}} s\}$.

Algebra – Example (Carrier)

$$\Sigma_{\mathbb{S},\mathbb{V}} = \{\texttt{allUsers} : \mathrm{List}(\mathrm{String});$$
$$\texttt{newUser} : \mathrm{String};$$
$$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$$
$$\texttt{newUserId} : \mathrm{int};$$
$$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$$
$$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \ \}$$

$\mathbb{C}_s =$

$\{$`exp` | `exp` is a Java expression s.t. program

```
import java.util.List;
public class C { private translate(s) x = exp; }
```

compiles for

$\mathrm{translate} = \{\ \mathrm{int} \mapsto \texttt{int}, \mathrm{String} \mapsto \texttt{String},$
$\mathrm{List}(x) \mapsto \texttt{List<}\mathrm{translate}(x)\texttt{>}\}\}$

Algebra – Example (morphisms)

$$\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \texttt{allUsers} : \mathrm{List}(\mathrm{String});$$
$$\texttt{newUser} : \mathrm{String};$$
$$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$$
$$\texttt{newUserId} : \mathrm{int};$$
$$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$$
$$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \,\}$$

$m_s : \mathbb{F}_s(\mathbb{C}) \to \mathbb{C}_s$ where
$\mathbb{F}_s(\mathbb{C}) = \sum_{S \in \mathrm{WF}} \sum_{o \in \mathsf{ranged}(s,S)} \prod_{i=1}^{\mathsf{arity}(o)} \mathbb{C}_{S^+(\pi_i(\mathsf{domain}_o))}$
with $\mathsf{ranged}(s,S) = \{o \in \mathbb{O} \mid S^+(\mathsf{range}(o)) \leq_{\mathbb{S}} s\}$

- $m_{\mathrm{List(String)}}(S, \texttt{allUsers}, ()) =$
  `new java.util.Arrays.asList("bob", "jack")`
- $m_{\mathrm{List(String)}}(\{\alpha \mapsto \mathrm{String}\}, \texttt{append}, (x, l)) =$ `l.add(x)`

Algebra – Example (morphisms)

$$\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \texttt{allUsers} : \mathrm{List}(\mathrm{String});$$
$$\texttt{newUser} : \mathrm{String};$$
$$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$$
$$\texttt{newUserId} : \mathrm{int};$$
$$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$$
$$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \,\}$$

$m_s : \mathbb{F}_s(\mathbb{C}) \to \mathbb{C}_s$ where
$\mathbb{F}_s(\mathbb{C}) = \sum_{S \in \mathrm{WF}} \sum_{o \in \mathsf{ranged}(s,S)} \prod_{i=1}^{\mathsf{arity}(o)} \mathbb{C}_{S^+(\pi_i(\mathsf{domain}_o))}$
with $\mathsf{ranged}(s,S) = \{o \in \mathbb{O} \mid S^+(\mathsf{range}(o)) \leq_{\mathbb{S}} s\}$

- $m_{\mathrm{List(int)}}(S, \texttt{allUserIds}, ()) =$
  `new java.util.Arrays.asList(1, 2)`
- $m_{\mathrm{List(int)}}(\{\alpha \mapsto \mathrm{int}\}, \texttt{append}, (x, l)) =$ `l.add(x)`

Algebra – Example (morphisms)

$$\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \texttt{allUsers} : \mathrm{List}(\mathrm{String});$$
$$\texttt{newUser} : \mathrm{String};$$
$$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$$
$$\texttt{newUserId} : \mathrm{int};$$
$$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$$
$$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \,\}$$

$m_s : \mathbb{F}_s(\mathbb{C}) \to \mathbb{C}_s$ where
$\mathbb{F}_s(\mathbb{C}) = \sum_{S \in \mathrm{WF}} \sum_{o \in \mathsf{ranged}(s,S)} \prod_{i=1}^{\mathsf{arity}(o)} \mathbb{C}_{S^+(\pi_i(\mathsf{domain}_o))}$
with $\mathsf{ranged}(s,S) = \{o \in \mathbb{O} \mid S^+(\mathsf{range}(o)) \leq_{\mathbb{S}} s\}$

- $m_{\mathrm{String}}(S, \texttt{newUser}, ()) = \texttt{"trudy"}$
- $m_{\mathrm{String}}(\{\alpha \mapsto \mathrm{String}\}, \texttt{first}, l) = \texttt{l.get(0)}$

Algebra – Example (morphisms)

$$\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \texttt{allUsers} : \mathrm{List}(\mathrm{String});$$
$$\texttt{newUser} : \mathrm{String};$$
$$\texttt{allUserIds} : \mathrm{List}(\mathrm{int});$$
$$\texttt{newUserId} : \mathrm{int};$$
$$\texttt{append} : (\alpha, \mathrm{List}(\alpha)) \twoheadrightarrow \mathrm{List}(\alpha);$$
$$\texttt{first} : \mathrm{List}(\alpha) \twoheadrightarrow \alpha \,\}$$

$m_s : \mathbb{F}_s(\mathbb{C}) \to \mathbb{C}_s$ where
$\mathbb{F}_s(\mathbb{C}) = \sum_{S \in \mathrm{WF}} \sum_{o \in \mathsf{ranged}(s,S)} \prod_{i=1}^{\mathsf{arity}(o)} \mathbb{C}_{S^+(\pi_i(\mathsf{domain}_o))}$
with $\mathsf{ranged}(s,S) = \{o \in \mathbb{O} \mid S^+(\mathsf{range}(o)) \leq_{\mathbb{S}} s\}$

- $m_{\mathrm{int}}(S, \texttt{newUserId}, ()) = \texttt{3}$
- $m_{\mathrm{int}}(\{\alpha \mapsto \mathrm{int}\}, \texttt{first}, l) = \texttt{l.get(0)}$

Algebra – Language Independence

By exchanging the carrier $\mathbb{C}$ we could have compiled $\Sigma_{\mathbb{S},\mathbb{V}}$ to any other language. Even untyped languages are no problem, e.g. unary natural numbers:

$$\Sigma_{\mathbb{S},\mathbb{V}} = \{\, \mathtt{one} : \mathrm{Odd};\ \mathtt{succ} : \alpha \twoheadrightarrow \beta \,\}$$

$\mathbb{C}_{\mathrm{Odd}} \ni o ::= \mathtt{I} \mid \mathtt{I}e;\ \mathbb{C}_{\mathrm{Even}} \ni e ::= \mathtt{I}o;\ \mathbb{C}_{\mathrm{Nat}} \ni n ::= e \mid o$

$m_{\mathrm{Even}}(\{\alpha \mapsto \mathrm{Odd}, \beta \mapsto \mathrm{Even}\}, \mathtt{succ}, o) = \mathtt{I}o$

$m_{\mathrm{Odd}}(S, \mathtt{one}, ()) = \mathtt{I}$

$m_{\mathrm{Odd}}(\{\alpha \mapsto \mathrm{Even}, \beta \mapsto \mathrm{Odd}\}, \mathtt{succ}, e) = \mathtt{I}e$

$m_{\mathrm{Nat}} = m_{\mathrm{Even}} \cup m_{\mathrm{Odd}}$

Synthesis – Goal

Given a signature $\Sigma_{(\mathbb{V},\mathbb{S})}$ *(an interface)*, a well-formedness restriction WF, a subsort relation $\leq_{\mathbb{S}}$, an algebra $(\mathbb{D}, h)$ *(an implementation)*, and a sort s *(a goal)*, find all elements of $\mathbb{D}_s$ *(programs)*, which can be generated from the signature only using h.

Synthesis – Battle Plan

1. Pick a well-understood small type system
   - Combinatory Logic with Intersection Types, Distributing Covariant Constructors and Finite Substitution Spaces
2. For the desired $\mathbb{S}$ find injective maps $[\![\_]\!] : (\mathbb{S}(V) \to \mathbb{T}_V)_{V \in \{\mathbb{V}, \emptyset\}}$, from open and closed sorts to types $\mathbb{T}_\emptyset$ and type schemas $\mathbb{T}_\mathbb{V}$, such that
   - the maps respect subsorting/subtyping
   - commute with substitutions on sorts/type schemas
   - generated type (schemas) use $\cap$ only in argument positioins of arrows
   - top-level arrows of generated type (schemas) are protected by type-constructors
3. Generate Γ with entries
   op : $[\![s_1]\!]_\mathbb{V} \to [\![s_2]\!]_\mathbb{V} \to \ldots \to [\![s]\!]_\mathbb{V}$ for each
   op : $(s_1, s_2, \ldots, s_n) \twoheadrightarrow s$ of $\Sigma_{(\mathbb{S},\mathbb{V})}$
4. Use inhabitation $\Gamma \vdash ? : [\![s]\!]_\emptyset$ to produce terms of carrier $\mathbb{C}_s = \{M \mid \Gamma \vdash M : [\![s]\!]_\emptyset\}$
5. Translate results to desired target carrier $\mathbb{D}_s$

Combinatory Logic with Intersection Types

$$\frac{\Gamma(x) = \tau \qquad S \in \mathrm{WF}}{\Gamma \vdash x : S^*(\tau)}\ (\textsc{Var})$$

$$\frac{\Gamma \vdash M : \mathtt{a} \to \mathtt{b} \qquad \Gamma \vdash N : \mathtt{a}}{\Gamma \vdash (M\ N) : \mathtt{b}}\ (\to \mathrm{E})$$

$$\frac{\Gamma \vdash M : \mathtt{a} \qquad \mathtt{a} \leq \mathtt{b}}{\Gamma \vdash M : \mathtt{b}}\ (\leq) \qquad \frac{\Gamma \vdash M : \mathtt{a} \qquad \Gamma \vdash M : \mathtt{b}}{\Gamma \vdash M : \mathtt{a} \cap \mathtt{b}}\ (\cap \mathrm{I})$$

- $\_^* : (\mathbb{V} \to \mathbb{T}_\emptyset) \to (\mathbb{T}_\mathbb{V} \to \mathbb{T}_\emptyset)$ lifts substitutions to type schemas, just like $\_^+ : (\mathbb{V} \to \mathbb{S}_\emptyset) \to (\mathbb{S}_\mathbb{V} \to \mathbb{S}_\emptyset)$ on sorts.
- WF is a translation of the well-formed space we specify with our signature

Intersection Types and Distributing Covariant Constructors

$$\frac{C_1 \leq_{\mathbb{C}} C_2 \qquad |C_1| = |C_2| \qquad \forall n : \mathtt{a}_n \leq \mathtt{b}_n}{C_1(\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_{|C_1|}) \leq C_2(\mathtt{b}_1, \mathtt{b}_2, \ldots, \mathtt{b}_{|C_2|})}\ (\textsc{CAx})$$

$$\frac{}{\mathtt{a} \leq \omega}\ (\leq \omega) \qquad \frac{}{\omega \leq \omega \to \omega}\ (\to \omega)$$

$$\frac{}{C_1(\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_{|C_1|}) \cap C_1(\mathtt{b}_1, \mathtt{b}_2, \ldots, \mathtt{b}_{|C_1|}) \leq C_1(\mathtt{a}_1 \cap \mathtt{b}_1, \mathtt{a}_2 \cap \mathtt{b}_2, \ldots, \mathtt{a}_{|C_1|} \cap \mathtt{b}_{|C_1|})}\ (\textsc{CDist})$$

$$\frac{\mathtt{a}_2 \leq \mathtt{a}_1 \qquad \mathtt{b}_1 \leq \mathtt{b}_2}{\mathtt{a}_1 \to \mathtt{b}_1 \leq \mathtt{a}_2 \to \mathtt{b}_2}\ (\textsc{Sub})$$

$$\frac{}{(\mathtt{a} \to \mathtt{b}_1) \cap (\mathtt{a} \to \mathtt{b}_2) \leq \mathtt{a} \to \mathtt{b}_1 \cap \mathtt{b}_2}\ (\textsc{Dist})$$

$$\frac{}{\mathtt{a} \leq \mathtt{a} \cap \mathtt{a}}\ (\textsc{Idem}) \qquad \frac{\mathtt{a}_1 \leq \mathtt{a}_2 \qquad \mathtt{a}_2 \leq \mathtt{a}_3}{\mathtt{a}_1 \leq \mathtt{a}_3}\ (\textsc{Trans})$$

$$\frac{\mathtt{a} \leq \mathtt{b}_1 \qquad \mathtt{a} \leq \mathtt{b}_2}{\mathtt{a} \leq \mathtt{b}_1 \cap \mathtt{b}_2}\ (\text{GLB})$$

$$\frac{}{\mathtt{a} \cap \mathtt{b} \leq \mathtt{a}}\ (\text{LUB}_1) \qquad \frac{}{\mathtt{a} \cap \mathtt{b} \leq \mathtt{b}}\ (\text{LUB}_2)$$

Constructors – Examples

$$\frac{}{\mathrm{Pair}(\mathrm{URL},\mathrm{Protocol}) \cap \mathrm{Pair}(\mathrm{DatabaseLocation},\mathrm{Network}) \leq \mathrm{Pair}(\mathrm{URL} \cap \mathrm{DatabaseLocation}, \mathrm{Protocol} \cap \mathrm{Network})}\ (\textsc{CDist})$$

$$\frac{\mathrm{List} \leq_{\mathbb{C}} \mathrm{List} \qquad |\mathrm{List}| = 1 \qquad \dfrac{\mathrm{Even} \leq_{\mathbb{C}} \mathrm{Nat} \qquad |\mathrm{Even}| = 0 = |\mathrm{Nat}|}{\mathrm{Even} \leq \mathrm{Nat}}\ (\textsc{CAx})}{\mathrm{List}(\mathrm{Even}) \leq \mathrm{List}(\mathrm{Nat})}\ (\textsc{CAx})$$

Exercise:

Show $\omega \leq \mathtt{a} \to \omega$ and

$(\mathtt{a}_1 \to \mathtt{b}_1) \cap (\mathtt{a}_2 \to \mathtt{b}_2) \leq \mathtt{a}_1 \cap \mathtt{a}_2 \to \mathtt{b}_1 \cap \mathtt{b}2$

Sort Translations

We can find the sort translation $[\![\_]\!]$ for all sorts isomorphic to regular trees:

$$\mathfrak{T}_{\mathbb{V}} \ni t_1, t_2 \ldots, t_n ::= l(t_1, \ldots, t_{\mathrm{arity}(l)}) \mid \alpha$$

respecting variance labeled subtyping:

$$\frac{l_1 \leq_{\mathbb{L}} l_2 \qquad \mathrm{arity}(l_1) = \mathrm{arity}(l_2) \qquad t_n \leq_{\mathfrak{T}} t'_n \wedge t'_n \leq_{\mathfrak{T}} t_n \text{ if } \mathrm{v}_{l_1}(n) = \mathsf{in}}{l_1(t_1, \ldots, t_n) \leq_{\mathfrak{T}} l_2(t'_1, \ldots, t'_n)}$$

$$\forall n : \mathrm{v}_{l_1}(n) = \mathrm{v}_{l_2}(n)$$

$$t_n \leq_{\mathfrak{T}} t'_n \text{ if } \mathrm{v}_{l_1}(n) = \mathsf{co}$$

$$t'_n \leq_{\mathfrak{T}} t_n \text{ if } \mathrm{v}_{l_1}(n) = \mathsf{contra}$$

Subtyping example:

$\mathrm{arity}(\mathrm{Arrow}) = 2, \mathrm{v}_{\mathrm{Arrow}}(0) = \mathsf{contra}, \mathrm{v}_{\mathrm{Arrow}}(1) = \mathsf{co}$

$\mathrm{Arrow}(\mathrm{Nat}, \mathrm{Even}) \leq_{\mathfrak{T}} \mathrm{Arrow}(\mathrm{Even}, \mathrm{Nat})$

Translation idea:

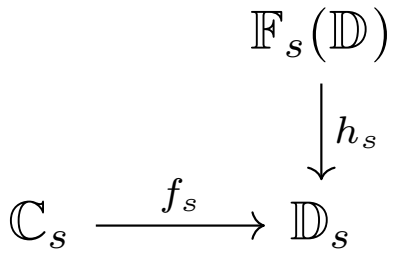
- $[\![l(t_1, t_2, \ldots t_n)]\!]_{\mathbb{V}} = \blacksquare((l([\![t_1]\!]^{\mathrm{v}_l}, [\![t_2]\!]^{\mathrm{v}_l}, \ldots, [\![t_n]\!]^{\mathrm{v}_l}) \to \bullet) \to \bullet)$
- $[\![t]\!]^{\mathsf{co}} = ([\![t]\!]_{\mathbb{V}} \to \bullet) \to \bullet \quad [\![t]\!]^{\mathsf{contra}} = (\bullet \to [\![t]\!]_{\mathbb{V}}) \to \bullet$
  $[\![t]\!]^{\mathsf{in}} = ([\![t]\!]_{\mathbb{V}} \to [\![t]\!]_{\mathbb{V}}) \to \bullet$

Inhabitation – Combinatory Logic with Intersection Types

For finite well-formed substitution spaces we have

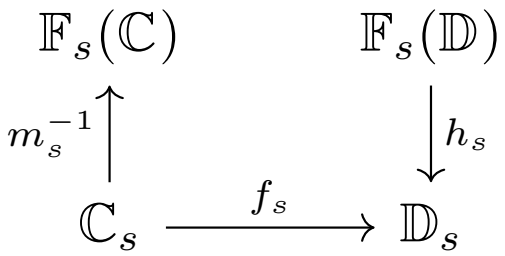
- Type checking ($\Gamma \vdash M : \mathsf{a}?$) is decidable
- Type inhabitation ($\Gamma \vdash ? : \mathsf{a}$) is decidable
  - the system is a slight extension of [3]
  - we also get a grammar describing all inhabitants
- When we have specifications $\Gamma_1, \mathrm{WF}_1, \mathsf{a}$ and $\Gamma_2, \mathrm{WF}_2, \mathsf{b}$ using completely unrelated constructor symbols we can use $\Gamma(x) = \Gamma_1(x) \cap \Gamma_2(x), \mathrm{WF} = \mathrm{WF}_1 \uplus \mathrm{WF}_2$ and get $\Gamma \vdash M : \mathsf{a} \cap \mathsf{b} \Leftrightarrow \Gamma_1 \vdash M : \mathsf{a}$ and $\Gamma_2 \vdash M : \mathsf{b}$
  - we can refine the search for inhabitants using multiple specifications in addition to signatures
  - *warning*: this only holds for unrelated constructor symbols!

Target Translation



- We want a function f_s to translate inhabitation results $\mathbb{C}_s = \{M \mid \Gamma \vdash M : [\![s]\!]_\emptyset\}$ to some target language carrier $\mathbb{D}_s$, for which we have an algebra h_s

Target Translation



- We want a function f_s to translate inhabitation results $\mathbb{C}_s = \{M \mid \Gamma \vdash M : [\![s]\!]_\emptyset\}$ to some target language carrier $\mathbb{D}_s$, for which we have an algebra h_s
- We can proof that there is a coalgebra $m_s^{-1} : \mathbb{C}_s \to \mathbb{F}_s(\mathbb{C})$:
  - for a term $M = \mathsf{op}\ M_1\, M_2 \ldots M_{\mathsf{arity}(op)}$ with type $[\![s]\!]_\emptyset$ we get $m_s^{-1}(M) = (S, \mathsf{op}, (M_1, M_2, \ldots, M_{\mathsf{arity}(op)}))$

Target Translation

$$\begin{array}{ccc} \mathbb{F}_s(\mathbb{C}) & \xrightarrow{\mathbb{F}(f)_s} & \mathbb{F}_s(\mathbb{D}) \\ {\scriptstyle m_s^{-1}}\uparrow & & \downarrow{\scriptstyle h_s} \\ \mathbb{C}_s & \xrightarrow{f_s} & \mathbb{D}_s \end{array}$$

- We want a function f_s to translate inhabitation results $\mathbb{C}_s = \{M \mid \Gamma \vdash M : [\![s]\!]_\emptyset\}$ to some target language carrier $\mathbb{D}_s$, for which we have an algebra h_s
- We can proof that there is a coalgebra $m_s^{-1} : \mathbb{C}_s \to \mathbb{F}_s(\mathbb{C})$:
  - for a term $M = \text{op}\ M_1\, M_2 \ldots M_{\text{arity}(op)}$ with type $[\![s]\!]_\emptyset$ we get $m_s^{-1}(M) = (S, \text{op}, (M_1, M_2, \ldots, M_{\text{arity}(op)}))$
- We construct f_s by solving the fixpoint equation: $f = \left(h_s \circ \mathbb{F}(f)_s \circ m_s^{-1}\right)_{s \in \mathbb{S}(\emptyset)}$
  - base case: the size of M is 1, hence $M = \text{op}$, $\text{arity}(\text{op}) = 0$ and $m_s^{-1}(M) = (S, \text{op}, ())$: solve by h_s.
  - recursion: the size is $n + 1$ we have $m_s^{-1}(M) = (S, \text{op}, (M_1, M_2, \ldots, M_{\text{arity}(\text{op})}))$ with smaller M_i: solve by h_s and f_s on smaller parts.

Target Translation – Properties

$$\begin{array}{ccc} \mathbb{F}_s(\mathbb{C}) & \xrightarrow{\mathbb{F}(f)_s} & \mathbb{F}_s(\mathbb{D}) \\ m_s^{-1} \uparrow & & \downarrow h_s \\ \mathbb{C}_s & \xrightarrow{f_s} & \mathbb{D}_s \end{array}$$

The construction is (up to choice of well-formed sort substitutions):

- Sound: all generated $\mathbb{D}_s$ terms are h_s interpretations of signature operations
- Complete: by listing all $\mathbb{C}_s$ we get all $\mathbb{D}_s$ which are h_s interpretations of signature operations
- Unique: all possible ways of constructing a translation f_s yield identical results to our construction

Related Approaches

- Traditionally, multi-sorted signatures don't use sub-typing and variables
  - if we leave them out, the signature specifications are compatible e.g. with those in lectures by Prof. Padawitz [4]
  - the fixpoint constrution of f_s is related to Lambek's Lemma ([6], Lemma 1)
- Without additions one can also use SMT Solving [5]
- An ad-hoc solving approach was used in [2]

Demo (Scala)

Conclusion

- Signatures are generic interface descriptions and algebras implementations
- Algebraic bureaucracy is powerful enough to cut through the programming language chaos
- We can connect (almost arbitrary) signatures with combinatory logic
- Synthesis can be done outside the "simple" world of $\lambda^{\rightarrow}$

References

[1] Nada Amin and Ross Tate. "Java and Scala's Type Systems are Unsound: The Existential Crisis of Null Pointers". In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM. 2016, pp. 838–848.

[2] Luís Eduardo de Souza Amorim, Sebastian Erdweg, Guido Wachsmuth, and Eelco Visser. "Principled Syntactic Code Completion Using Placeholders". In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2016. Amsterdam, Netherlands: ACM, 2016, pp. 163–175. ISBN: 978-1-4503-4447-0. DOI: 10.1145/2997364.2997374. URL: http://doi.acm.org/10.1145/2997364.2997374.

[3] Boris Düdder, Moritz Martens, Jakob Rehof, and Pawel Urzyczyn. "Bounded Combinatory Logic". In: *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*. 2012, pp. 243–258. DOI: 10.4230/LIPIcs.CSL.2012.243. URL: http://dx.doi.org/10.4230/LIPIcs.CSL.2012.243.

[4] Peter Padawitz. "From Grammars and Automata to Algebras and Coalgebras". In: *Algebraic Informatics - 4th International Conference, CAI 2011, Linz, Austria, June 21-24, 2011. Proceedings*. 2011, pp. 21–43. DOI: 10.1007/978-3-642-21493-6_2. URL: http://dx.doi.org/10.1007/978-3-642-21493-6_2.

[5] Andrew Reynolds, Viktor Kuncak, Cesare Tinelli, Clark Barrett, and Morgan Deters. "Refutation-based synthesis in SMT". In: *Formal Methods in System Design* (Feb. 16, 2017), p. 1. ISSN: 1572-8102. DOI: 10.1007/s10703-017-0270-2. URL: http://dx.doi.org/10.1007/s10703-017-0270-2.

[6] Michael B. Smyth and Gordon D. Plotkin. "The Category-Theoretic Solution of Recursive Domain Equations". In: *SIAM J. Comput.* 11.4 (1982), pp. 761–783. DOI: 10.1137/0211062.