Komponenten- und Service-orientierte Softwarekonstruktion

Vorlesung 5: Combinatory Logic Synthesis

Jakob Rehof
LS XIV – Software Engineering

TU Dortmund
Sommersemester 2015

SS 2015
Function composition in Combinatory Logic

\[ \Gamma \vdash F : \tau' \rightarrow \tau \quad \Gamma \vdash G : \tau' \]

\[ \Gamma \vdash (F \ G) : \tau \] (→E)

as logical model of applicative composition of named component interfaces \((F : \rho) \in \Gamma\) from a repository \(\Gamma\), satisfying goal \(\tau\)
Composition Synthesis

- Function composition in Combinatory Logic

\[
\frac{\Gamma \vdash F : \tau' \rightarrow \tau \quad \Gamma \vdash G : \tau'}{\Gamma \vdash (F \ G) : \tau} (\rightarrow E)
\]

as logical model of applicative composition of named component interfaces \((F : \rho) \in \Gamma\) from a repository \(\Gamma\), satisfying goal \(\tau\)

- Inhabitation problem as foundation for automatic synthesis:

\[\exists F. \, \Gamma \vdash F : \tau \]  Notation \(\Gamma \vdash ? : \tau\)
Composition Synthesis

- Function composition in Combinatory Logic

\[
\frac{\Gamma \vdash F : \tau' \to \tau \quad \Gamma \vdash G : \tau'}{\Gamma \vdash (F \ G) : \tau}(\to E)
\]

as logical model of applicative composition of named component interfaces \((F : \rho) \in \Gamma\) from a repository \(\Gamma\), satisfying goal \(\tau\)

- Inhabitation problem as foundation for automatic synthesis:

\[\exists F. \; \Gamma \vdash F : \tau\] Notation \(\Gamma \vdash ? : \tau\)

- Does there exist a program composition \(F\) from repository \(\Gamma\) with \(\Gamma \vdash F : \tau\)? Inhabitation algorithm is used to construct (synthesize) \(F\) from \(\Gamma\) and \(\tau\)
Composition Synthesis

- Function composition in Combinatory Logic

\[
\Gamma \vdash F : \tau' \rightarrow \tau \quad \Gamma \vdash G : \tau' \\
\therefore \Gamma \vdash (F \ G) : \tau
\]

as logical model of applicative composition of named component interfaces \((F : \rho) \in \Gamma\) from a repository \(\Gamma\), satisfying goal \(\tau\)

- Inhabitation problem as foundation for automatic synthesis:
  \(\exists F. \ \Gamma \vdash F : \tau\) ? Notation \(\Gamma \vdash ? : \tau\)
  
  Does there exist a program composition \(F\) from repository \(\Gamma\) with \(\Gamma \vdash F : \tau\) ?
  Inhabitation algorithm is used to construct (synthesize) \(F\) from \(\Gamma\) and \(\tau\)

- CLS is inherently component-oriented
Components are exposed as typed combinator symbols \((F : \tau)\), representing component names with types as interfaces. Types will be generalized later.

Component composition as applicative combinations \((FG)\). Composition will be generalized later.

However, we will first have to generalize the notion of combinatory logic from any particular \textit{fixed base} (like \(\mathcal{B} = \{S, K, I\}\)) to arbitrary finite sets of combinators.
CL vs $\lambda$-calculus

- Recall from Lecture 3 that the fixed base $\mathcal{B} = \{S, K, I\}$ (even $\mathcal{B} = \{S, K\}$) is equivalent to $\lambda$-calculus, both untyped and in simple types.
- We saw that inhabitation in $\lambda\to$ and simple typed SKI-calculus is $\mathbb{PSPACE}$-complete (Statman).
- Proof/term enumeration, Ben-Yelles, Hindley: See [Hindley, 2008].
Recall from Lecture 3 that the fixed base $\mathcal{B} = \{S, K, I\}$ (even $\mathcal{B} = \{S, K\}$) is equivalent to $\lambda$-calculus, both untyped and in simple types.

We saw that inhabitation in $\lambda \to$ and simple typed SKI-calculus is $\text{PSPACE}$-complete (Statman).

Proof/term enumeration, Ben-Yelles, Hindley: See [Hindley, 2008].

But a fixed base is not the right model for composition synthesis, since repository ($\Gamma$) varies.

And $\lambda$-calculus (SKI-calculus) as model is not component-oriented as is CL.
Recall combinatory logic SKI (Lecture 3)

\[ \Gamma, x : \tau \vdash_{\text{SKI}} x : \tau \] \hspace{2cm} (\text{var})

\[ \Gamma \vdash_{\text{SKI}} I : \tau \rightarrow \tau \] \hspace{2cm} (I)

\[ \Gamma \vdash_{\text{SKI}} K : \tau \rightarrow \sigma \rightarrow \tau \] \hspace{2cm} (K)

\[ \Gamma \vdash_{\text{SKI}} S : (\tau \rightarrow \sigma \rightarrow \rho) \rightarrow (\tau \rightarrow \sigma) \rightarrow \tau \rightarrow \rho \] \hspace{2cm} (S)

\[ \Gamma \vdash_{\text{SKI}} F : \tau \rightarrow \sigma \quad \Gamma \vdash_{\text{SKI}} G : \tau \] \hspace{2cm} (\rightarrow E)

\[ \Gamma \vdash_{\text{SKI}} (FG) : \sigma \]

Notice that variables \( x \) have fixed, monomorphic types, whereas combinators \( S, K, I \) have infinitely many types (their types are schematic or polymorphic). We shall return to this important point in Lecture 5.
Fix a typed base $\mathcal{B}$, for example SKI:

$$
\begin{align*}
S & : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma \\
K & : \alpha \to \beta \to \alpha \\
I & : \alpha \to \alpha 
\end{align*}
$$

with the rules, for any given base $\mathcal{B}$:

$$
\begin{align*}
\Gamma \vdash _{\mathcal{B}} X : \tau & \quad \frac{(X : \tau) \in \mathcal{B}, \; S : \forall \to \exists}{\Gamma \vdash _{\mathcal{B}} X : S(\tau)} & \text{(comb)} \\
\frac{\Gamma, x : \tau \vdash _{\mathcal{B}} x : \tau}{\Gamma, x : \tau \vdash _{\mathcal{B}} x : \tau} & \text{(var)} \\
\frac{\Gamma \vdash _{\mathcal{B}} F : \tau \to \sigma \quad \Gamma \vdash _{\mathcal{B}} G : \tau}{\Gamma \vdash _{\mathcal{B}} (FG) : \sigma} & \text{($\to$E)}
\end{align*}
$$
Combinatory logic \( \text{CL} \)

Assuming that variables \( x \) are considered special combinator symbols with constant types, we can assume that \( \Gamma \) is an arbitrary set of typed combinator symbols and simplify the presentation to:

\[
\begin{align*}
\Gamma, X : \tau \vdash_{\text{CL}} X : S(\tau) & \quad (\text{var}) \\
\Gamma \vdash_{\text{CL}} F : \tau \rightarrow \sigma & \quad \Gamma \vdash_{\text{CL}} G : \tau \\
\Gamma \vdash_{\text{CL}} (FG) : \sigma & \quad (\rightarrow \text{E})
\end{align*}
\]
We consider the relativized inhabitation problem:

- **Given** $\Gamma$ and $\tau$, does there exist $F$ such that $\Gamma \vdash_{\text{CL}} F : \tau$?
We consider the relativized inhabitation problem:

- **Given** $\Gamma$ and $\tau$, does there exist $F$ such that $\Gamma \vdash_{\text{CL}} F : \tau$?

Relativized inhabitation in simple types is much harder than inhabitation in the fixed theory of $\lambda \rightarrow$ (SKI)

- **Undecidable**: Linial-Post theorems, 1948 ff.
We consider the *relativized inhabitation* problem:

- **Given** $\Gamma$ and $\tau$, does there exist $F$ such that $\Gamma \vdash_{cl} F : \tau$?

Relativized inhabitation in simple types is much harder than inhabitation in the fixed theory of $\lambda \to$ (SKI)

- **Undecidable**: Linial-Post theorems, 1948 ff.

Reason: instead of considering a fixed theory ($\lambda \to$, IPC) we consider an arbitrary input theory
We consider the *relativized inhabitation* problem:

- Given $Γ$ and $τ$, does there exist $F$ such that $Γ ⊢^{cl} F : τ$?

Relativized inhabitation in simple types is much harder than inhabitation in the fixed theory of $λ →$ (SKI)


Reason: instead of considering a fixed theory ($λ →$, IPC) we consider an arbitrary input theory

*The CLS view*: Already in simple types, relativized inhabitation defines a Turing-complete logic programming language for component composition
Two-counter automaton acceptance is undecidable. Two counter automaton $A = \langle Q, q_0, q_F, \delta \rangle$, control states $Q$, initial state $q_0$, final state $q_F$, counters $c_1, c_2 \in \mathbb{N}$, transition relation $\delta$ given by ($i = 1, 2$):

- $q : c_i := c_i + 1; \text{goto } p$
- $q : c_i := c_i - 1; \text{goto } p$
- $q : \text{if } (c_i = 0) \text{ then goto } p \text{ else goto } r$

Configurations $\mathcal{C} = (q, n, m)$, $q \in Q$, $n$ and $m$ contents of counters $c_1$ resp. $c_2$.

Types of the form $[\mathcal{C}] = q \rightarrow s^n(0) \rightarrow s^m(0)$ will represent configurations $\mathcal{C} = (q, n, m)$.
Encoding of $\mathcal{A}$ into $\Gamma_\mathcal{A}$

- **Fin**: $q_F \rightarrow \alpha \rightarrow \beta$

- $q : c_1 := c_1 + 1; \text{ goto } p$:

  $$\text{Add}_1[q, p] : (p \rightarrow s(\alpha) \rightarrow \beta) \rightarrow (q \rightarrow \alpha \rightarrow \beta).$$

- $q : c_1 := c_1 - 1; \text{ goto } p$:

  $$\text{Sub}_1[q, p] : (p \rightarrow \alpha \rightarrow \beta) \rightarrow (q \rightarrow s(\alpha) \rightarrow \beta).$$

- $q : \text{if } (c_1 = 0) \text{ then goto } p \text{ else goto } r$:
  - $\text{Tst}_1^Z[q, p] : (p \rightarrow 0 \rightarrow \beta) \rightarrow (q \rightarrow 0 \rightarrow \beta)$ and
  - $\text{Tst}_1^{NZ}[q, r] : (r \rightarrow s(\alpha) \rightarrow \beta) \rightarrow (q \rightarrow s(\alpha) \rightarrow \beta)$. 
Reduction

Consider the two-counter automaton

\[ \mathcal{A} = \begin{align*}
q_0 : & \quad c_1 := c_1 - 1; \text{goto } q_1 \\
q_1 : & \quad \text{if } (c_1 = 0) \text{ then goto } q_F \text{ else goto } q_0
\end{align*} \]

from initial state \((q_0, 1, 0)\). Since

\[ \begin{align*}
\text{Fin} : & \quad q_F \rightarrow 0 \rightarrow 0 \\
\text{Tst}_1^Z[q_1, q_F] : & \quad (q_F \rightarrow 0 \rightarrow 0) \rightarrow (q_1 \rightarrow 0 \rightarrow 0) \\
\text{Sub}_1[q_0, q_1] : & \quad (q_1 \rightarrow 0 \rightarrow 0) \rightarrow (q_0 \rightarrow s(0) \rightarrow 0)
\end{align*} \]

we get

\[ \Gamma_\mathcal{A} \vdash \text{Sub}_1[q_0, q_1] \ (\text{Tst}_1^Z[q_1, q_F] \ \text{Fin}) : q_0 \rightarrow s(0) \rightarrow 0 \]
Reduction

Theorem 1

Let \( A \) be a two-counter automaton with initial configuration \((q_0, n_0, m_0)\). \( A \) accepts if and only if there exists a term \( e \) with \( \Gamma_A \vdash e : q_0 \rightarrow s^{n_0}(0) \rightarrow s^{m_0}(0) \).

Lemma 2

Let \( C \) and \( C' \) be configurations in \( A \). We have \( C \rightarrow C' \) if and only if there is a term \( e \) with \( \Gamma_A \vdash e : [C'] \rightarrow [C] \).

Lemma 3

Let \( C \) be a configuration of \( A \). \( C \) leads to acceptance in \( A \) if and only if there is a term \( e \) with \( \Gamma_A \vdash e : [C] \).

Exercise 1

Prove Theorem 1.
The input repository $\Gamma$ is a logic program at the level of types
Each combinator type is a rule in the program
The inhabitation goal is the input goal to the program
Search for inhabitants is the execution of the program
Inhabitants are programs synthesized as solution space to the program

Broadly related (proof search as semantics of generalized logic programming):

"Linial-Post Spectrum"

The Linial-Post Spectrum is a graphical representation of various complexity classes and their relationships. It illustrates the hierarchy of computational complexity classes, starting from the simplest class at the bottom to the most complex at the top. The classes are:

- **Ptime** (Polynomial time)
- **Co-NP**
- **PSPACE** (Polynomial space)
- **EXPTIME** (Exponential time)
- **2EXPTIME**
- **∞**

These classes are arranged on a line from Ptime at 1 to ∞, with intermediate classes such as Co-NP and PSPACE. The Linial-Post Spectrum also highlights various logics, including:

- **CPL** (Classical Propositional Logic)
- **IPL (S4)**
- **T ➔ R**
- **KSOS**

The diagram uses arrows (e.g., R ➔) to indicate relationships or inclusions between the complexity classes. For example, R includes IPL (S4), which in turn includes CPL. The question mark (?) may represent an open question or an unknown relationship within the spectrum.
Simple types are not sufficient to specify composition (even though they are Turing-complete under relativized inhabitation).
Intersection types

Definition 4 (Intersection types)

Let $\mathbb{V}$ denote a denumerable set of *type variables*, ranged over by metavariables $\alpha, \beta, \gamma, \ldots$, and let $b$ range over a set $\mathbb{B}$ of *type constants*. The set $\mathbb{T}_\cap$ of *intersection types*, ranged over by $\tau, \sigma, \rho, \ldots$, is defined inductively by:

- $\alpha \in \mathbb{V} \Rightarrow \alpha \in \mathbb{T}_\cap$
- $b \in \mathbb{B} \Rightarrow b \in \mathbb{T}_\cap$
- $\tau \in \mathbb{T}_\cap, \sigma \in \mathbb{T}_\cap \Rightarrow \tau \rightarrow \sigma \in \mathbb{T}_\cap$
- $\tau \in \mathbb{T}_\cap, \sigma \in \mathbb{T}_\cap \Rightarrow \tau \cap \sigma \in \mathbb{T}_\cap$

Intersection types are considered modulo associativity, commutativity and idempotence of intersection: $\tau \cap (\sigma \cap \rho) = (\tau \cap \sigma) \cap \rho$, $\tau \cap \sigma = \sigma \cap \tau$, $\tau \cap \tau = \tau$. 
Intersection type system $\lambda^\cap$

\[
\frac{}{\Gamma, x : \tau \vdash x : \tau} \text{(var)}
\]

\[
\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x. M : \tau \rightarrow \sigma} \text{ (→I)}
\]

\[
\frac{\Gamma \vdash M : \tau \rightarrow \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma} \text{ (→E)}
\]

\[
\frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash M : \tau_2}{\Gamma \vdash M : \tau_1 \cap \tau_2} \text{ (∩I)}
\]

\[
\frac{\Gamma \vdash M : \tau_i}{\Gamma \vdash M : \tau_i} \text{ (∩E)}
\]

Major reference for this system (a.k.a. "BCD", Barendregt-Coppo-Dezani):

[Barendregt et al., 1983].
A good exposition of the following fundamental result (which goes back to around 1980) can be found in [Ghilezan, 1996].

**Lemma 5 (Subject expansion)**

Suppose $M \rightarrow^\beta N$ by contracting the redex occurrence $(\lambda x.P)Q$ in $M$. If $\Gamma \vdash M : \sigma$ and $Q$ is typable in the same context $\Gamma$, then $\Gamma \vdash N : \sigma$.

**Theorem 6 (Fundamental theorem for $\lambda^\cap$)**

A term $M$ is typable in system $\lambda^\cap$, if and only if, $M$ is strongly normalizing.

**Corollary 7 (Undecidability)**

Typability in $\lambda^\cap$ is undecidable.
