Logische Methoden des Software Engineerings
Vertiefungsmodul 2

Combinatory Logic Synthesis (Simple Types)

Jakob Rehof & Andrej Dudenhefner
LS XIV – Software Engineering

TU Dortmund

WS 2018/19

WS 2018/19

Composition Synthesis

- Function composition in Combinatory Logic

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

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

Composition Synthesis

- Function composition in Combinatory Logic

$$\frac{\Gamma \vdash F : \tau' \to \tau \qquad \Gamma \vdash G : \tau'}{\Gamma \vdash (F\ G) : \tau}(\to\mathsf{E})$$

  as logical model of applicative composition of *named component interfaces* $(F : \rho) \in \Gamma$ from a *repository* Γ, satisfying *goal* τ
- *Inhabitation problem* as foundation for *automatic synthesis:* $\exists F.\ \Gamma \vdash F : \tau$? Notation $\Gamma \vdash \textcolor{red}{?} : \tau$

Composition Synthesis

- Function composition in Combinatory Logic

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

  as logical model of applicative composition of *named component interfaces* $(F : \rho) \in \Gamma$ from a *repository* Γ, satisfying *goal* τ
- *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 Γ with $\Gamma \vdash F : \tau$? Inhabitation algorithm is used to *construct* (synthesize) F from Γ and τ

Composition Synthesis

- Function composition in Combinatory Logic

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

  as logical model of applicative composition of *named component interfaces* $(F : \rho) \in \Gamma$ from a *repository* Γ, satisfying *goal* τ
- *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 Γ with $\Gamma \vdash F : \tau$? Inhabitation algorithm is used to *construct* (synthesize) F from Γ and τ
- CLS is inherently *component-oriented*

Foundations in Combinatory Logic

- 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 *fixed base* (like $\mathfrak{B} = \{\mathbf{S}, \mathbf{K}, \mathbf{I}\}$) to arbitrary finite sets of combinators.

CL vs λ-calculus

- Recall that the fixed *base* $\mathfrak{B} = \{\mathbf{S}, \mathbf{K}, \mathbf{I}\}$ (even $\mathfrak{B} = \{\mathbf{S}, \mathbf{K}\}$) is equivalent to λ-calculus, both untyped and in simple types.
- We saw that inhabitation in $\lambda^{\rightarrow}$ and simple typed SKI-calculus is $\textsc{Pspace}$-complete (Statman).
- Proof/term enumeration, Ben-Yelles, Hindley: See [Hin08].

CL vs λ-calculus

- Recall that the fixed *base* $\mathfrak{B} = \{\mathbf{S}, \mathbf{K}, \mathbf{I}\}$ (even $\mathfrak{B} = \{\mathbf{S}, \mathbf{K}\}$) is equivalent to λ-calculus, both untyped and in simple types.
- We saw that inhabitation in $\lambda^{\rightarrow}$ and simple typed SKI-calculus is $\textsc{Pspace}$-complete (Statman).
- Proof/term enumeration, Ben-Yelles, Hindley: See [Hin08].
- But a *fixed base* is not the right model for composition synthesis, since repository (Γ) *varies*
- And λ-calculus (SKI-calculus) as model is not *component-oriented* as is CL

Recall combinatory logic SKI

$$\frac{}{\Gamma, x : \tau \vdash_{\text{SKI}} x : \tau}\,(\mathsf{var})$$

$$\frac{}{\Gamma \vdash_{\text{SKI}} \mathbf{I} : \tau \to \tau}\,(\mathbf{I})$$

$$\frac{}{\Gamma \vdash_{\text{SKI}} \mathbf{K} : \tau \to \sigma \to \tau}\,(\mathbf{K})$$

$$\frac{}{\Gamma \vdash_{\text{SKI}} \mathbf{S} : (\tau \to \sigma \to \rho) \to (\tau \to \sigma) \to \tau \to \rho}\,(\mathbf{S})$$

$$\frac{\Gamma \vdash_{\text{SKI}} F : \tau \to \sigma \qquad \Gamma \vdash_{\text{SKI}} G : \tau}{\Gamma \vdash_{\text{SKI}} (FG) : \sigma}\,(\to\mathsf{E})$$

Notice that variables x have fixed, *monomorphic types*, whereas combinators $\mathbf{S}, \mathbf{K}, \mathbf{I}$ have infinitely many types (their types are *schematic* or *polymorphic*).

Combinatory logic $\mathrm{CL}(\mathfrak{B})$

Fix a typed base $\mathfrak{B}$, for example SKI:

$$\begin{array}{lcl} \mathbf{S} & : & (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma \\ \mathbf{K} & : & \alpha \to \beta \to \alpha \\ \mathbf{I} & : & \alpha \to \alpha \end{array}$$

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

$$\frac{(X : \tau) \in \mathfrak{B}, \quad S : \mathbb{V} \to \mathbb{T}}{\Gamma \vdash_{\mathfrak{B}} X : S(\tau)}\,(\text{comb})$$

$$\frac{}{\Gamma, x : \tau \vdash_{\mathfrak{B}} x : \tau}\,(\text{var})$$

$$\frac{\Gamma \vdash_{\mathfrak{B}} F : \tau \to \sigma \qquad \Gamma \vdash_{\mathfrak{B}} G : \tau}{\Gamma \vdash_{\mathfrak{B}} (FG) : \sigma}\,(\to\text{E})$$

Combinatory logic CL

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

$$\frac{[S : \mathbb{V} \to \mathbb{T}]}{\Gamma, X : \tau \vdash_{\text{CL}} X : S(\tau)}(\mathsf{var})$$

$$\frac{\Gamma \vdash_{\text{CL}} F : \tau \to \sigma \qquad \Gamma \vdash_{\text{CL}} G : \tau}{\Gamma \vdash_{\text{CL}} (FG) : \sigma}(\to\mathsf{E})$$

Relativized Inhabitation

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

Relativized Inhabitation

- We consider the *relativized inhabitation* problem:
  - **Given** Γ **and** τ, *does there exist* F *such that* $\Gamma \vdash_{\mathrm{CL}} F : \tau$*?*
- Relativized inhabitation in simple types is much harder than inhabitation in the fixed theory of $\lambda^{\rightarrow}$ (SKI)
  - *Undecidable*: **Linial-Post theorems, 1948 ff.**

Relativized Inhabitation

- We consider the *relativized inhabitation* problem:
  - **Given** Γ **and** τ, *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 $\lambda^{\to}$ (SKI)
  - *Undecidable*: **Linial-Post theorems, 1948 ff.**
- Reason: instead of considering a fixed theory ($\lambda^{\to}$, IPC) we consider an arbitrary input theory

Relativized Inhabitation

- We consider the *relativized inhabitation* problem:
  - **Given** Γ **and** τ, *does there exist* F *such that* $\Gamma \vdash_{\mathrm{CL}} F : \tau$*?*
- Relativized inhabitation in simple types is much harder than inhabitation in the fixed theory of $\lambda^{\rightarrow}$ (SKI)
  - *Undecidable*: **Linial-Post theorems, 1948 ff.**
- Reason: instead of considering a fixed theory ($\lambda^{\rightarrow}$, 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

Turing-Completeness of Simple Types!

Two-counter automaton acceptance is undecidable. Two counter automaton $\mathcal{A} = \langle Q, q_0, q_F, \delta \rangle$, control states Q, inital state q_0, final state q_F, counters $c_1, c_2 \in \mathbb{N}$, transition relation δ given by $(i = 1, 2)$:

- $q : c_i := c_i + 1; \text{goto } p$
- $q : c_i := c_i - 1; \text{goto } p$
- $q : if\ (c_i = 0)$ then goto p 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 $[\mathcal{C}] = q \to s^n(0) \to s^m(0)$ will represent configurations $\mathcal{C} = (q, n, m)$

Encoding of $\mathcal{A}$ into $\Gamma_{\mathcal{A}}$

- $\texttt{Fin} : q_F \to \alpha \to \beta$
- $q : c_1 := c_1 + 1; \text{goto } p$:

$$\texttt{Add}_1[\mathsf{q}, \mathsf{p}] : (p \to s(\alpha) \to \beta) \to (q \to \alpha \to \beta).$$

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

$$\texttt{Sub}_1[\mathsf{q}, \mathsf{p}] : (p \to \alpha \to \beta) \to (q \to s(\alpha) \to \beta).$$

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

Consider the two-counter automaton

$$\mathcal{A} = \begin{array}{l} q_0 : c_1 := c_1 - 1; \text{goto } q_1 \\ q_1 : if\ (c_1 = 0) \text{ then goto } q_F \text{ else goto } q_0 \end{array}$$

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

$$\begin{array}{lcl} \mathtt{Fin} & : & q_F \to 0 \to 0 \\ \mathtt{Tst}_1^{\mathtt{Z}}[\mathtt{q}_1, \mathtt{q}_F] & : & (q_F \to 0 \to 0) \to (q_1 \to 0 \to 0) \\ \mathtt{Sub}_1[\mathtt{q}_0, \mathtt{q}_1] & : & (q_1 \to 0 \to 0) \to (q_0 \to s(0) \to 0) \end{array}$$

we get

$$\Gamma_{\mathcal{A}} \vdash \mathtt{Sub}_1[\mathtt{q}_0, \mathtt{q}_1]\ (\mathtt{Tst}_1^{\mathtt{Z}}[\mathtt{q}_1, \mathtt{q}_F]\ \mathtt{Fin}) : q_0 \to s(0) \to 0$$

Reduction

Theorem 1

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

Lemma 2

Let $\mathcal{C}$ and $\mathcal{C}'$ be configurations in $\mathcal{A}$. We have $\mathcal{C} \to \mathcal{C}'$ if and only if there is a term e with $\Gamma_{\mathcal{A}} \vdash e : [\mathcal{C}'] \to [\mathcal{C}]$.

Lemma 3

Let $\mathcal{C}$ be a configuration of $\mathcal{A}$. $\mathcal{C}$ leads to acceptance in $\mathcal{A}$ if and only if there is a term e with $\Gamma_{\mathcal{A}} \vdash e : [\mathcal{C}]$.

Exercise 1

Prove Theorem 1.

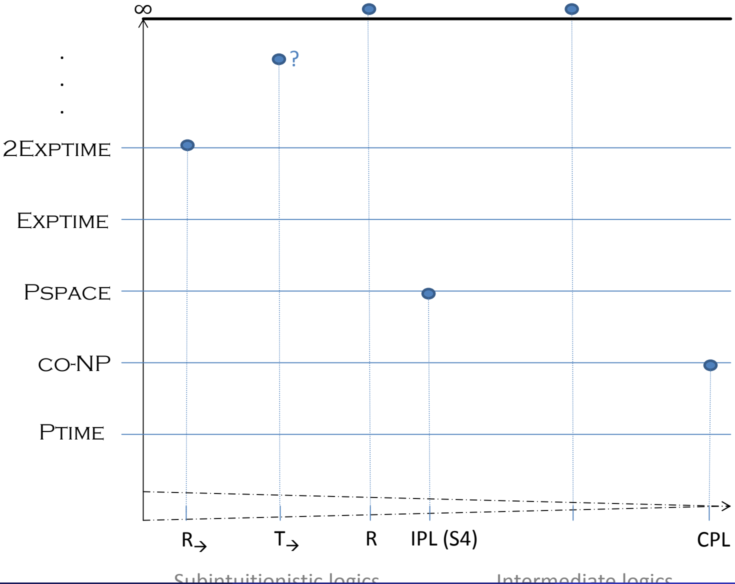
Types as Logic Programs for Composition

- The input repository Γ 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):

D. Miller, G. Nadathur, F. Pfenning, A. Scedrov: *Uniform Proofs as a Foundation for Logic Programming*, Ann. Pure App. Logic, 1991

"Linial-Post Spectrum"



Semantic specification

Simple types are not sufficient to specify composition (even though they are Turing-complete under relativized inhabitation).

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:

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

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 $\lambda^\cap$

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

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma}(\to\text{I})$$

$$\frac{\Gamma \vdash M : \tau \to \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}(\to\text{E})$$

$$\frac{\Gamma \vdash M : \tau_1 \qquad \Gamma \vdash M : \tau_2}{\Gamma \vdash M : \tau_1 \cap \tau_2}(\cap\text{I}) \qquad \frac{\Gamma \vdash M : \tau_1 \cap \tau_2}{\Gamma \vdash M : \tau_i}(\cap\text{E})$$

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

[BCDC83].

A good exposition of the following fundamental result (which goes back to around 1980) can be found in [Ghi96].

Lemma 5 (Subject expansion)

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

H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini.
A Filter Lambda Model and the Completeness of Type Assignment.
Journal of Symbolic Logic, 48(4):931–940, 1983.

S. Ghilezan.
Strong Normalization and Typability with Intersection Types.
Notre Dame Journal of Formal Logic, 37(1):44–52, 1996.

J. Roger Hindley.
Basic Simple Type Theory.
Cambridge Tracts in Theoretical Computer Science, vol. 42,
Cambridge University Press, 2008.