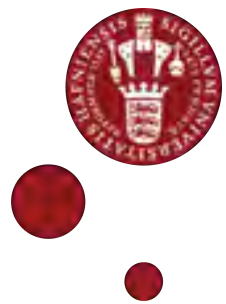
Type-based Program Synthesis of Controller Logic using the Combinatory Logic Synthesizer (CLS)

Synchron 2022, Fréjus, France

Boris Düdder

University of Copenhagen, Denmark

November 29th, 2022



Joint work with Jakob Rehof, Jan Bessai, George T. Heineman, Ugo de'Liguoro, Paweł Urzyczyn, Andrej Dudenhefner, and Moritz Martens

Main Motivation (1) - We cannot write simple programs today?

Software supply chains and sustainability

- Good news: software-reuse is accepted by software industry
- High fan-in of frameworks and libraries in software products
- Short life-time of frameworks and libraries (3 years)
- Rapid value deprecation of software investments
- Code navigation, comprehension, and composition are major activities in software development

Formal model of code composition and automatic programming

- Scalable model for software reuse
- Multiple programming languages and paradigms used in project
- Meta-programming of program construction for multiple languages and paradigms?

Main Motivation (2)

Program synthesis

- Program synthesis and generation (Polikarpova et al. [PS18])
- Synthesis of programming language expressions (usually functional or procedural), e.g, with focus on heap manipulation
- Rarely full OO languages (Java, C#, Python)

Composition-based synthesis

- Software reuse $\Rightarrow$ code templates with specifications
- Code templates usually as abstract syntax or binding tree
- Higher-order blueprints specifying how to transform and combine code templates
- Functional language with strong type system for blueprint program

Foundations in Combinatory Logic

$$\begin{array}{llll} \text{Types} & \tau & ::= & \alpha \mid \tau \to \tau' \\ \text{Terms} & e, e' & ::= & X \mid (e\ e') \end{array}$$

Rules

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

$$\frac{\Gamma \vdash e : \tau' \to \tau \qquad \Gamma \vdash e' : \tau'}{\Gamma \vdash (e\ e') : \tau}(\to \text{E})$$

Under Curry-Howard isomorphism, Hilbert-style presentation of minimal propositional logic (schematism S + modus ponens)

Decision problems

For a λ-term M and type τ:

- Type checking: $\Gamma \vdash M : \tau$?
- Type inference: $\exists \tau \,.\, \Gamma \vdash M : \tau$?
- Type inhabitation: $\exists M \,.\, \Gamma \vdash M : \tau$?
- Relativized type inhabitation: $\exists \Gamma, M \,.\, \Gamma \vdash M : \tau$?
- Complexity results on type inhabitation decidability

System	Complexity	Author(s)
$\mathrm{FCL}(\leq)$	PTIME	Rehof and PawełUrzyczyn [RP1
$\mathrm{FCL}(\leq, \cap)$	EXPTIME	Rehof and PawełUrzyczyn [RP1
$\mathrm{BCL}_k(\leq)$	EXPTIME	Düdder *et al.* [Düd+12]
$\mathrm{BCL}_k(\rightarrow, \cap)$k	(k+2)-EXPTIME	Düdder *et al.* [Düd+12]
$\mathrm{CL}(\mathbf{SK}) \rightarrow$	PSPACE	Statman [Sta79]
$\lambda(- \cap I)$	EXPSPACE	Rehof and Urzyczyn [RU12]
$\lambda^{r2}\cap$	EXPSPACE	Urzyczyn [Urz09]
$\lambda\cap$	∞	Urzyczyn [Urz99]

Curry-Howard isomorphism

Type system as deduction system

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}(\text{var}) \qquad \frac{\Gamma \vdash e : \tau \to \tau' \quad \Gamma \vdash e' : \tau}{\Gamma \vdash (e\ e') : \tau'}(\to\text{E})$$

$$\frac{\Gamma, X : \tau \vdash e' : \tau'}{\Gamma \vdash \lambda X.\ e' : \tau \to \tau'}(\text{Abs})$$

Programs-as-proofs correspondence

Programming side	Logic side
Types	Propositions
Type constructors	Logical connectives
Programs	Proofs

Writing expressions in typed programming languages = writing proof checked by type checker

Relativized Inhabitation

- We consider the *relativized inhabitation* problem:
  - **Given** Γ **and** τ, *does there exist* e *such that* $\Gamma \vdash e : \tau$*?*

Relativized Inhabitation

- We consider the *relativized inhabitation* problem:
  - **Given** Γ **and** τ, *does there exist* e *such that* $\Gamma \vdash e : \tau$*?*
- Relativized inhabitation is much harder
  - *Undecidable*: **Linial-Post theorems, 1948 ff.**

Relativized Inhabitation

- We consider the *relativized inhabitation* problem:
  - **Given** Γ **and** τ, *does there exist* e *such that* $\Gamma \vdash e : \tau$*?*
- Relativized inhabitation is much harder
  - *Undecidable*: **Linial-Post theorems, 1948 ff.**
- *The CLS view:* Already in simple types, relativized inhabitation defines a Turing-complete logic programming language for component composition
  - Γ is logic program, interfaces/combinators $(X : \tau') \in \Gamma$ are its rules, τ its input goal, search for inhabitants its execution semantics

Combinatory Logic with $\cap$-types and subtyping

- Subtyping rules

$$\sigma \leq \omega, \quad \omega \leq \omega \to \omega, \quad \sigma \cap \tau \leq \sigma, \quad \sigma \cap \tau \leq \tau,$$
$$\sigma \leq \sigma \cap \sigma; (\sigma \to \tau) \cap (\sigma \to \rho) \leq \sigma \to \tau \cap \rho;$$

If $\sigma \leq \sigma'$ and $\tau \leq \tau'$ then $\sigma \cap \tau \leq \sigma' \cap \tau'$ and $\sigma' \to \tau \leq \sigma \to \tau'$.

- Type rules

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}(\text{var}) \qquad \frac{\Gamma \vdash e : \tau \to \tau' \ \Gamma \vdash e' : \tau}{\Gamma \vdash (e\ e') : \tau'}(\to\text{E})$$

$$\frac{\Gamma \vdash e : \tau_1 \ \ \Gamma \vdash e : \tau_2}{\Gamma \vdash e : \tau_1 \cap \tau_2}(\cap\text{I}) \qquad \frac{\Gamma \vdash e : \tau \ \ \tau \leq \tau'}{\Gamma \vdash e : \tau'}(\leq)$$

- Characterizes exactly strong-normalizing λ-terms
- `TemperatureConv` : $\texttt{int} \cap \texttt{Fahrenheit} \to \texttt{int} \cap \texttt{Celsius}$
- Refinement types (finite universe of variable substitutions S (polymorphic))

Composition Synthesis

- Typed function composition (modus ponens)

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

  as logical model of applicative composition of *named component interfaces* $(X : \rho) \in \Gamma$ from a *repository* Γ, satisfying *goal* τ
- *Inhabitation problem* as foundation for *automatic synthesis:* $\exists e.\ \Gamma \vdash e : \tau$? Notation $\Gamma \vdash ? : \tau$
  - Does there exist a program composition e from repository Γ with $\Gamma \vdash e : \tau$? Inhabitation algorithm is used to *construct* (synthesize) e from Γ and τ
- CLS is inherently *component-oriented*

Type System

Intersection Types $\lambda^{\to,\cap,\leq}$ (Dezani et al. [BCD83])

- Intersection $\cap$ combines two types of a single λ-term
- Allows combining types, implementation and semantic types
- Combined types $\Leftrightarrow$ feature vector
- Refinement of (type) specification
- Combinatory Logic (CL) with $\cap$ types [RP11] (FCL) & [Düd+12] (BCL)

Modal Intersection Types $\lambda^{\to,\cap,\leq,\Box}$ (**D**, Martens, Rehof [DMR14])

- $\Box\tau$ modal type representing code of type τ
- $\texttt{eval} : \Box\tau \to \tau$ (e.g., $\Box\texttt{int} \to \texttt{int}$) corresponds to **T** axiom in modal logic S4
- Introducing $\to_\Box$ then **K** axiom $\Box(\tau \to_\Box \sigma) \to (\Box\tau \to \Box\sigma)$ (under isomorphism=function injection) (**D** HLGA15/KPS21)

Types as Logic Programs for Composition

Under Curry-Howard isomorphism (proofs-as-programs):

- 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
- CLS 3.0 Scala-tool with DSL for domain and code manipulation [Düd14; Bes19] and core verified in Coq[1]

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

[1] `https://www.combinators.org` or `https://github.com/combinators`

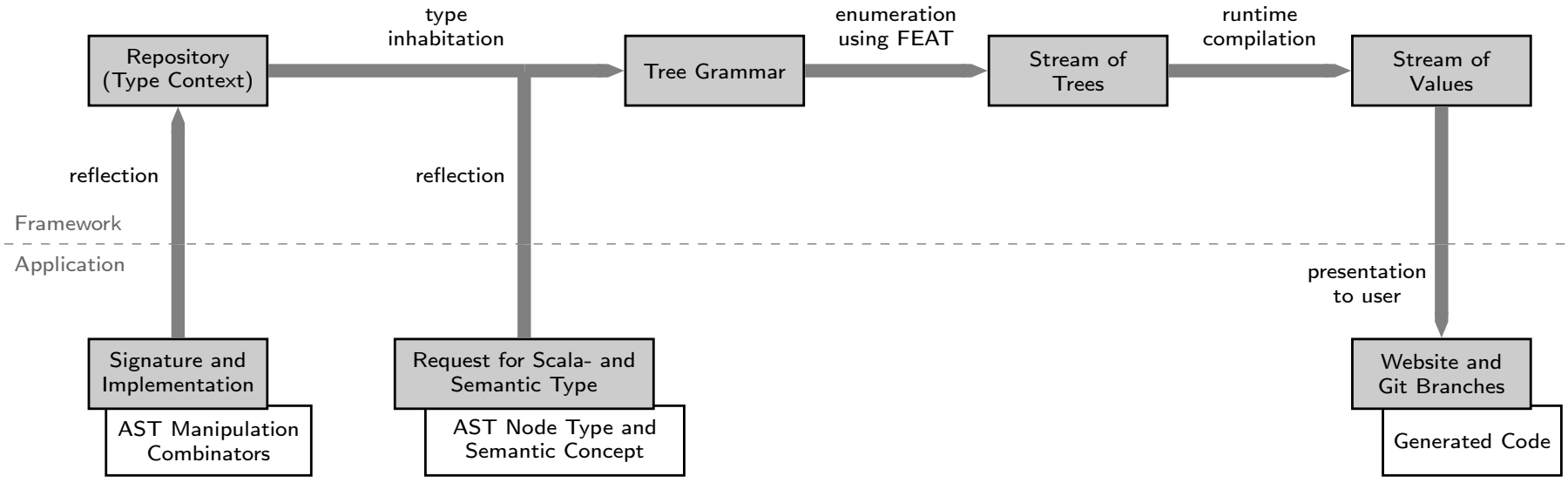
(CL)S

Combinatory Logic Synthesizer

- Scala framework to do synthesis
- Continuation of prior work in F# [Düd14]
- Deeply integrated into the host language:

```
@combinator object addLyrics {
  def apply(
  music: LilyPond[Music],
  lyrics: String): LilyPond[Song] =
  music.setLyrics(lyrics)
  val semanticType: Type =
  (springsteen =>: springsteen =>: springsteen) :&:
  (dion =>: dion =>: dion)
}
```

(CL)S Scala


Repository
(Type Context)
type
inhabitation
Tree Grammar
enumeration
using FEAT
Stream of
Trees
runtime
compilation
Stream of
Values
reflection
reflection
Framework
Application
presentation
to user
Signature and
Implementation
AST Manipulation
Combinators
Request for Scala- and
Semantic Type
AST Node Type and
Semantic Concept
Website and
Git Branches
Generated Code

Results

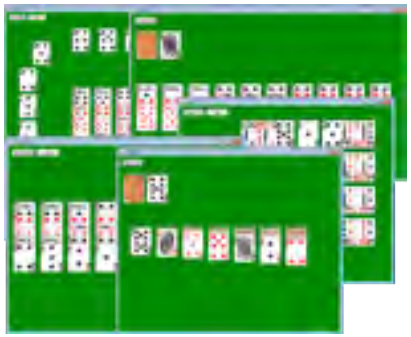
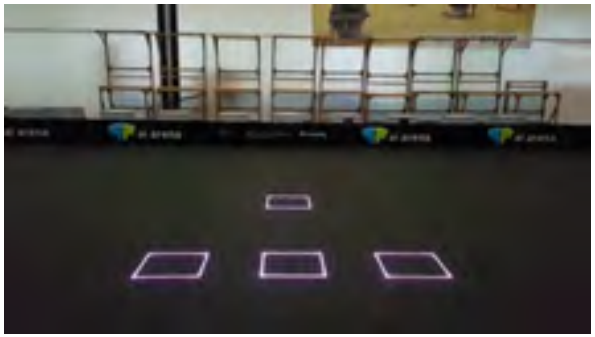
- Battle-tested and easy to use
- Open-Source, high test coverage ($\approx$ 90%), CI-enabled
- Tight integration into Scala means:
  - Full JVM ecosystem available – including tools
  - Combinator implementations can use OO-modularity, e.g.,

```
@combinator object Foo extends Bar(42)
```

  - Programmable model-based repositories/signatures:

```
val repository = if (model.hasProperty) {
 staticRepository.addCombinator(
 new MyComb(model.someValue))
} else {
 staticRepository
}
val results =
repository.inhabit(model.computeTargetType)
```

(CL)S is and has been used for synthesis of...


Music, Docker-Containers, Network Service Configurations, Card Games, Robot Control Programs, Path-Planning Algorithms, Intelligent Plant Pots, Clinical Pathways, Vulnerability Benchmark Code, Web Applications, Dependency-Injected Software, Factory Construction Plans, Object Oriented Mixin Code, Image Processing Pipelines, Unit Tests, Product Lines, Continuous-Integration Pipelines, BPMN Workflows, Numeric Data Processing Tools, Expression Problem Approaches, Graph Libraries, ...

Record calculus for OOP (OOP≠FP)

- Type system for mixin-based synthesis [Bes+15; Bes+18]

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}\,(Ax) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\,(\to I) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\,(\to E)$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \cap \tau}\,(\cap) \qquad \frac{}{\Gamma \vdash M : \omega}\,(\omega) \qquad \frac{\Gamma \vdash M : \sigma \qquad \sigma \leq \tau}{\Gamma \vdash M : \tau}\,(\leq)$$

$$\frac{}{\Gamma \vdash \langle l_i = M_i \mid i \in I \rangle : \langle\rangle}\,(\langle\rangle) \qquad \frac{\Gamma \vdash M_k : \sigma \quad k \in I}{\Gamma \vdash \langle l_i = M_i \mid i \in I \rangle : \langle l_k : \sigma \rangle}\,(rec) \qquad \frac{\Gamma \vdash M : \langle l : \sigma \rangle}{\Gamma \vdash M.l : \sigma}\,(sel)$$

$$\frac{\Gamma \vdash M : \rho_1 \qquad \Gamma \vdash R : \rho_2 \quad (*)}{\Gamma \vdash M \oplus R : \rho_1 * \rho_2}\,(+)$$

- Self-fixpoint approximation (subtyping ≠ inheritance) [Bes+18][Bes+20]

Software product line (cf. [PBD05])

- family of similar software systems from shared set of software assets
- using a common means of production
- Management of product variants challenging (software evolution increases challenges)

Composition-based approach

- generate software product line members from complex compositions of code fragments [BHD21]
- based on a selection of features constraint by predefined selection logic, cf. [DHR15a; Bes+16; DHR15b]

Solitaire Production Line

- Solitaire card game of George T. Heineman[a]
- Variation in:
  - layouts and stacks
  - selection logic and winning rules
- 32 variations in Java[b]
- Generated code size of member 3k–9k LoC
- Based on tool Combinatory Logic Synthesizer CLS 3.0[c]

---

[a] `https://github.com/combinators/nextgen-solitaire`
[b] `https://github.com/combinators/solitaire-downloads`
[c] `https://combinators.org`

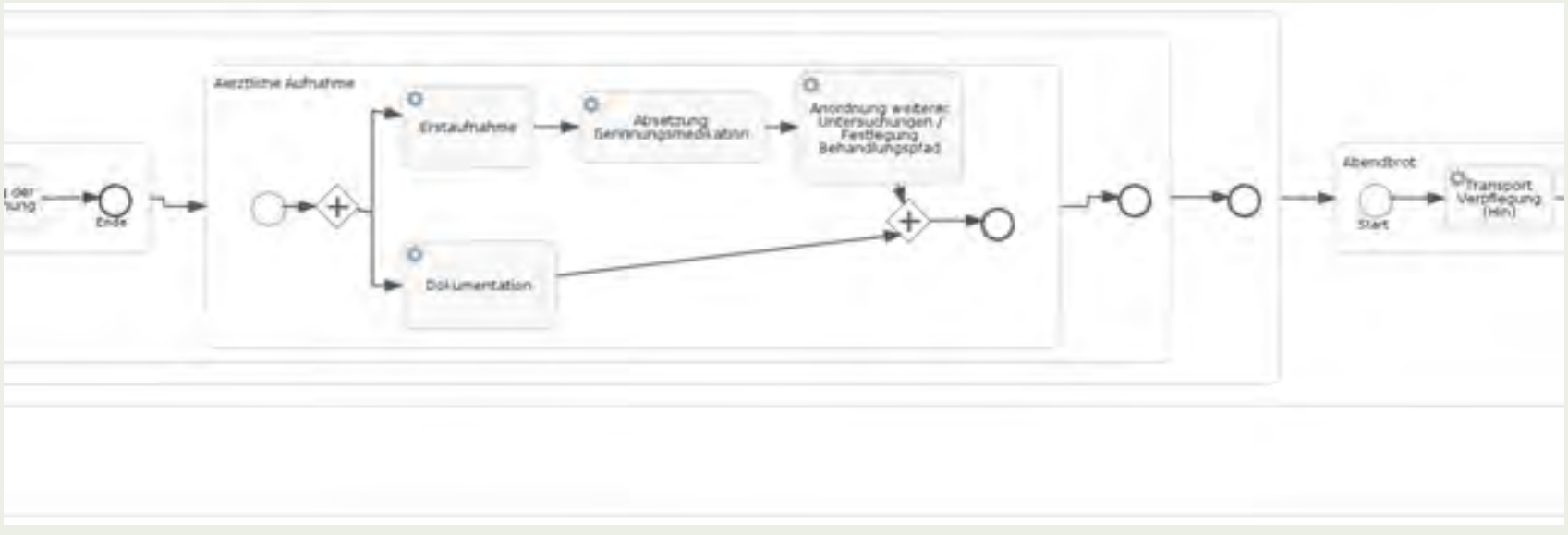
Experimental Results

Running Synthesized Control Flows on NXT Robots

Transfer: Synthesis of eHealth Workflows

Generated Individual Clinical Path for a Cardiovascular Examination (ICD-10-CM I00-I99)

Section of Examination Process Variant



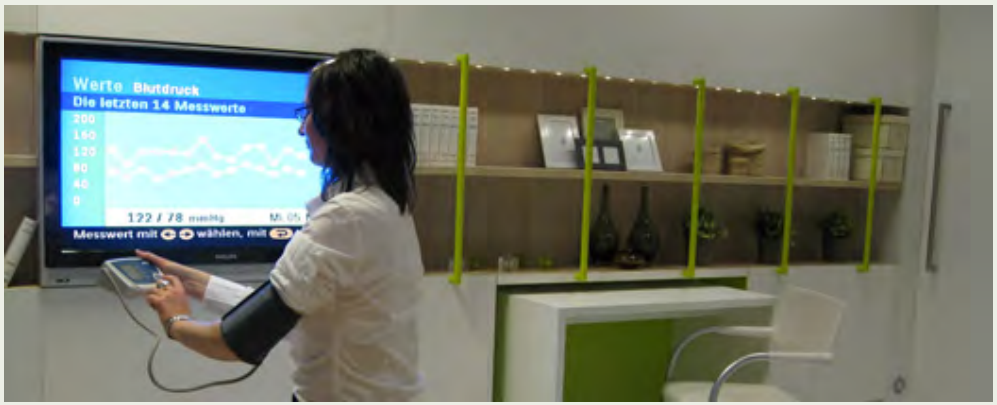
Full Examination Process Variant (70 activities)

ICD-10-CM I00-I99 process has 227 variants without constraints

Transfer: Synthesis of eHealth Workflows

Generated Individual Clinical Path for a Cardiovascular Examination (ICD-10-CM I00-I99)

Experiments in inHaus2 project Fraunhofer, Duisburg



MSc. Sebastian Kurth in cooperation with eHealth division of Fraunhofer ISST

Ongoing work related to Reactive Systems

- Secure and Fair Crowdsourcing [Wu+21; Jia+22]
- Event-based Reactive Systems (specification, actors and monitor synthesis) with Yongluan Zhou, Tilman Zuckmantel [Zuc+22]
- DCR Graphs (BPM) with Thomas Hildebrandt, Tijs Slaats, Yibin Xu) [Sla+13]
- Automatic Compliance Models and Checking for Gaia-X with Jakob Rehof
- Automatic Acceptance Checking for Industrial Internet Services with Ma Yufeng

Conclusion

- Types support developers by restriction (avoiding errors)
- Type-based programming and automatic program generation is coming
  - Significant progress in last years
- CLS 3.0[2] generates non-trivial programs >1k LoC
- Programming language and logic is seed for complex software systems
- Automatic program synthesis allows for exploratory generation of program solutions

[2] `https://combinators.org`

Backup slides

Definition (Subsorted Signature Families)

I-indexed $\leq_{\mathbb{S}}$-subsorted signature families Σ are defined by $\Sigma_{i \in I} = (\mathbb{S}, \mathbb{O}, \mathrm{arity}_i, dom_i, \mathrm{range}_i)$ where

- I is a set of indexes
- $\mathbb{S}$ is a set of sorts ordered by a preorder (reflexive, transitive) relation $\leq_{\mathbb{S}} \subseteq \mathbb{S} \times \mathbb{S}$
- $\mathbb{O}$ is a finite set of operations
- $\mathrm{arity} : (\mathbb{O} \to \mathbb{N})_{i \in I}$ is a family of functions, assigning an arity to each operation
- $dom : \left(\left(\prod_{n=1}^{\mathrm{arity}_i(o)} \mathbb{S} \right)_{o \in \mathbb{O}} \right)_{i \in I}$ is a family of families of vectors, assigning a domain, which is a vector of $\mathrm{arity}_i(o)$ sorts, to each operation
- $\mathrm{range} : \left((\mathbb{S})_{o \in \mathbb{O}} \right)_{i \in I}$ is a family of families of sorts, assigning a range to each operation.

Example

$$\mathbb{S} = \{\text{Exp}, \text{LitExp}, \text{AddExp},$$
$$\text{PrintVisitor},$$
$$\text{String}, \text{Double}\}$$

$$\text{LitExp} \leq_{\mathbb{S}} \text{Exp} \qquad \text{AddExp} \leq_{\mathbb{S}} \text{Exp}$$

$$\mathbb{O} = \{\texttt{dispatch}, \texttt{one}, \texttt{add}, \texttt{printer}\}$$

$$I = \{(\mathit{visitor}, \mathit{print}), (\mathit{oo}, \mathit{eval})\}$$

$$\begin{aligned}\Sigma = \{&\texttt{one}_{i \in I} : () \to \text{Lit};\\ &\texttt{add}_{i \in I} : (\text{Exp}, \text{Exp}) \to \text{Add};\\ &\texttt{printer}_{i \in I} : () \to \text{PrintVisitor};\\ &\texttt{dispatch}_{(\mathit{visitor}, \mathit{print})} : (\text{Exp}, \text{PrintVisitor}) \to \text{String}\\ &\texttt{dispatch}_{(\mathit{oo}, \mathit{eval})} : (\text{Exp}) \to \text{Double}\}\end{aligned}$$

Definition (Subsorted Signature Family Functor)

Given an I-indexed $\leq_{\mathbb{S}}$-subsorted signature family $\Sigma_{i \in I} = (\mathbb{S}, \mathbb{O}, \mathrm{arity}_i, dom_i, \mathrm{range}_i)$, its functor $\mathbf{F}^{\Sigma}$ is defined by:

$$\mathbf{F}^{\Sigma} : \mathbf{Set}^{\mathbb{S}} \to \mathbf{Set}^{\mathbb{S}}$$

$$\mathbf{F}^{\Sigma}(\mathscr{C}) = \left(\bigoplus_{i \in I} \bigoplus_{o \in \mathrm{ranged}(i,s)} \prod_{n=1}^{\mathrm{arity}_i(o)} \mathscr{C}_{\pi_n(dom_i(o))} \right)_{s \in \mathbb{S}}$$

$$\mathbf{F}^{\Sigma}(f)_s(i, o, (x_1, x_2, \ldots, x_{\mathrm{arity}_i(o)})) =$$
$$(i, o,$$
$$(f_{\pi_1(dom_i(o))}(x_1), f_{\pi_2(dom_i(o))}(x_2), \ldots, f_{\pi_{\mathrm{arity}_i(o)}(dom_i(o))}(x_{\mathrm{arity}_i(o)})))$$

where

- $\mathscr{C}$ is a $\mathbb{S}$-indexed family of sets
- $\mathrm{ranged}(i, s) = \{o \in \mathbb{O} \mid \mathrm{range}_i(o) \leq_{\mathbb{S}} s\}$ collects all operations that have ranges compatible with s
- $f : (\mathscr{C}_s \to \mathcal{D}_s)_{s \in \mathbb{S}}$ is a morphism in $\mathbf{Set}^{\mathbb{S}}$

Definition ($\mathbf{F}^{\Sigma}$-Algebra)

Given an indexed $\leq_{\mathbb{S}}$-subsorted signature family Σ, an $\mathbf{F}^{\Sigma}$-Algebra $\mathcal{A}^{\Sigma} = (\mathscr{C}, h)$ is defined by

- A carrier $\mathscr{C}$, which is a $\mathbb{S}$-indexed family of sets, and
- An action $h : \left(\mathbf{F}^{\Sigma}(\mathscr{C})_s \to \mathscr{C}_s\right)_{s \in \mathbb{S}}$, which is a $\mathbb{S}$-indexed family of functions

Example (Algebra $\mathcal{J}$)

- Σ as before
- $\mathscr{C}_s$ = set of all Java expressions of type s
- Action defined by:

$$h_{\mathsf{Lit}}(i \in I, \mathtt{one}, ()) = h_{\mathsf{Exp}}(i \in I, \mathtt{one}, ()) = \texttt{new Lit(1)}$$

$$h_{\mathsf{Add}}(i \in I, \mathtt{add}, (\mathtt{l}, \mathtt{r})) = h_{\mathsf{Exp}}(i \in I, \mathtt{add}, (\mathtt{l}, \mathtt{r})) = \texttt{new Add(l, r)}$$

$$h_{\mathsf{PrintVisitor}}(i \in I, \mathtt{printer}, ()) = \texttt{new PrintVisitor(StandardCharsets.UTF\_8)}$$

$$h_{\mathsf{Double}}((oo, eval), \mathtt{dispatch}, (\mathtt{toEval})) = \texttt{toEval.eval()}$$

$$h_{\mathsf{String}}((visitor, print), \mathtt{dispatch}, (\mathtt{toVisit}, \mathtt{visitedBy})) = \texttt{toVisit.accept(visitedBy)}$$

Definition (Algebraically Generated Objects)

Let Σ be an indexed $\leq_{\mathbb{S}}$-subsorted signature family, and $\mathcal{A}^{\Sigma} = (\mathscr{C}, h)$ be an $\mathbf{F}^{\Sigma}$-algebra. An object is algebraically generated iff it is contained in any set in the family $\mathbf{AlgGen}^{\mathcal{A}^{\Sigma}}$. For any sort s, $\mathbf{AlgGen}_s^{\mathcal{A}^{\Sigma}} \subseteq \mathscr{C}_s$ is the least set closed under the rule:

If $(i, o, (x_1, x_2, \ldots, x_{\mathrm{arity}_i(o)})) \in \mathbf{F}^{\Sigma}(\mathscr{C})_s$

and for all $i \in \{1, 2, \ldots, \mathrm{arity}_i(o)\} : x_i \in \mathbf{AlgGen}^{\mathcal{A}^{\Sigma}}_{\pi_i(dom_i(o))}$

then $h_s(i, o, (x_1, x_2, \ldots, x_{\mathrm{arity}_i(o)})) \in \mathbf{AlgGen}_s^{\mathcal{A}^{\Sigma}}$

Example (Objects of sort String generated by $\mathcal{J}$)

$\mathbf{AlgGen}^{\mathcal{J}^{\Sigma}}_{\mathsf{String}} =$

$\{h_{\mathsf{String}}((visitor, print), \texttt{dispatch}, (\texttt{toVisit}, \texttt{visitedBy})) \mid$

$\texttt{toVisit} \in \mathbf{AlgGen}^{\mathcal{J}^{\Sigma}}_{\mathsf{Expr}}$ and $\texttt{visitedBy} \in \mathbf{AlgGen}^{\mathcal{J}^{\Sigma}}_{\mathsf{PrintVisitor}} \quad \} =$

{`new Lit(1).accept(new PrintVisitor(StandardCharsets.UTF_8))`, ...}

Synthesis – Algebraically

Definition (Synthesis)

Given an indexed $\leq_{\mathbb{S}}$-subsorted signature family Σ, an $\mathbf{F}^{\Sigma}$-algebra $\mathcal{A}^{\Sigma} = (\mathscr{C}, h)$, and a sort $s \in \mathbb{S}$:
Enumerate $\mathbf{AlgGen}_s^{\mathcal{A}^{\Sigma}}$ or determine that it is empty.

Results from [Bes19]

- Emptiness of $\mathbf{AlgGen}_s^{\mathcal{A}^\Sigma}$ is undecidable
  - Reduction of 2-Counter Automaton halting-problem
- Restriction to finite I in $\Sigma_{i \in I}$ is decidable
  - Possible to obtain $\mathbf{AlgGen}_s^{\mathcal{A}^\Sigma}$ via type inhabitation in Finite Combinatory Logic with Intersection Types (FCL)

$$\exists M.\ \Gamma \vdash M : \tau$$

$$\Leftrightarrow$$

$$\exists T.\ T \in \mathbf{AlgGen}_s^{\mathcal{A}^\Sigma}$$

Finite Combinatory Logic with Intersection Types

Given finite sets $\mathbf{B}$, $\mathbf{C}$, preorder $\leq_{\mathbf{C}} \subseteq \mathbf{C} \times \mathbf{C}$, and $\Gamma : \mathbf{B} \to \mathbb{T}$:

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \,(\to \mathrm{E})$$

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \,(\textsc{Var}) \qquad \frac{\Gamma \vdash M : A \qquad A \leq B}{\Gamma \vdash M : B} \,(\leq)$$

where:
$\mathbb{A} \ni M, N ::= x \mid MN$ with $x \in \mathbf{B}$
$\mathbb{T} \ni A, B ::= \omega \mid c(A) \mid (A \star B) \mid (A \to B) \mid (A \cap B)$ with $c \in \mathbf{C}$
$\leq_{\mathbf{C}}$ is extended to $\leq \subset \mathbb{T} \times \mathbb{T}$ in a canonical way.

pictures: wikimedia.org, di.unito.it, mimuw.edu.pl, tu-dortmund.de

Results from [Bes19]

- Algorithm to decide $A \leq B$ in $\mathcal{O}(n^2)$
- Algorithm for minimal prime factor sets of types
  - C is prime if $A \cap B \leq C$ implies $A \leq C$ or $B \leq C$
- Algorithm for type checking $\Gamma \vdash M : A$
- Algorithm for *enumerative* type inhabitation
  - Given Γ and A, compute regular tree grammar G, s.t. $\Gamma \vdash M : A$ iff $M \in \mathcal{L}(G)$
  - Regular tree grammar $\cong$ non-deterministic tree automaton, avoids alternation removal blowup [RP11]
- Technique to combine different domains of discourse:
  - $\Gamma_{1\cap 2} \vdash M : A_{1\cap 2}$ if *and only if* $\Gamma_1 \vdash M : A_1$ and $\Gamma_2 \vdash M : A_2$
  - Example: $\Gamma_1 \vdash \mathrm{srt} : \mathbb{N}$ and $\Gamma_2 \vdash \mathrm{srt} : \mathrm{GoedelNumber}(\mathrm{QuickSort})$

Algebra and Inhabitation – Hip to be Square

Given any finitely indexed $\leq_{\mathbb{S}}$-subsorted signature family Σ and $\mathbf{F}^{\Sigma}$-algebra $\mathcal{A}^{\Sigma} = (g, \mathcal{D})$, we can construct

- A sort-embedding embed : $\mathbb{S} \to \mathbb{T}$
- A context Γ_{Σ}
- An algebra-coalgebra pair h and h^{-1} with $\mathscr{C}_s = \{M \mid \Gamma_{\Sigma} \vdash M : \mathsf{embed}(s)\}$

resulting in an algebra morphism family interpret that is:

- *Unique*: if $m_s \circ h_s = g_s \circ \mathbf{F}^{\Sigma}(m)_s$ then $m = \mathsf{interpret}$
- *Sound*: for all $M_s \in \mathscr{C}_s$: $\mathsf{interpret}(M_s) \in \mathbf{AlgGen}_s^{\mathcal{A}^{\Sigma}}$
- *Complete*: for all $T_s \in \mathbf{AlgGen}_s^{\mathcal{A}^{\Sigma}}$ exists $N_s \in \mathscr{C}_s$, s.t. $\mathsf{interpret}_s(N_s) = T_s$

References I

[1] N. Polikarpova and I. Sergey, "Structuring the Synthesis of Heap-Manipulating Programs," *CoRR*, vol. abs/1807.07022, 2018. arXiv: 1807.07022. [Online]. Available: http://arxiv.org/abs/1807.07022.

[2] J. Rehof and PawełUrzyczyn, "Finite Combinatory Logic with Intersection Types," in *Proceedings of TLCA'11*, ser. LNCS, vol. 6690, Springer, 2011, pp. 169–183.

[3] B. Düdder, M. Martens, J. Rehof, and PawełUrzyczyn, "Bounded Combinatory Logic," in *CSL 2012*, ser. LIPIcs, vol. 16, Schloss Dagstuhl, 2012, pp. 243–258.

[4] R. Statman, "Intuitionistic Propositional Logic Is Polynomial-space Complete," *Theoretical Computer Science*, vol. 9, pp. 67–72, 1979.

References II

[5] J. Rehof and P. Urzyczyn, "The Complexity of Inhabitation with Explicit Intersection," in *Kozen Festschrift*, ser. Lecture Notes in Computer Science, vol. 7230, Springer, 2012, pp. 256–270.

[6] P. Urzyczyn, "Inhabitation of Low-Rank Intersection Types," in *Proceedings of TLCA'09*, ser. Lecture Notes in Computer Science, vol. 5608, Springer, 2009, pp. 356–370.

[7] P. Urzyczyn, "The Emptiness Problem for Intersection Types," *Journal of Symbolic Logic*, vol. 64, no. 3, pp. 1195–1215, 1999.

References III

[8] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini, "A Filter Lambda Model and the Completeness of Type Assignment," *Journal of Symbolic Logic*, vol. 48, no. 4, pp. 931–940, 1983.

[9] B. Düdder, M. Martens, and J. Rehof, "Staged Composition Synthesis," in *Proceedings of ESOP'14*, ser. LNCS, vol. 8410, Springer, 2014, pp. 67–86.

[10] B. Düdder, "Automatic synthesis of component & connector software architectures with bounded combinatory logic," Ph.D. dissertation, Technical University Dortmund, Germany, 2014.

References IV

[11] J. Bessai, "A type-theoretic framework for software component synthesis.," Ph.D. dissertation, Technical University of Dortmund, Germany, 2019.

[12] J. Bessai, A. Dudenhefner, B. Duedder, U. De'Liguoro, T.-C. Chen, and J. Rehof, "Mixin Composition synthesis Based on Intersection Types," in *TLCA 2015*, To appear, 2015.

[13] J. Bessai, T.-C. Chen, A. Dudenhefner, B. Düdder, U. de'Liguoro, and J. Rehof, "Mixin Composition Synthesis based on Intersection Types," *Logical Methods in Computer Science (LMCS)*, vol. Volume 14, Issue 1, Feb. 2018. DOI: 10.23638/LMCS-14(1:18)2018. [Online]. Available: https://lmcs.episciences.org/4319.

References V

[14] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software Product Line Engineering – Foundations, Principles, and Techniques*. Springer, 2005.

[15] J. Bessai, G. T. Heineman, and B. Düdder, "Covariant Conversions (CoCo): A Design Pattern for Type-Safe Modular Software Evolution in Object-Oriented Systems," in *Proceedings of the 35th European Conference on Object-Oriented Programming (ECOOP)*, M. Sridharan, Ed., vol. 194, Schloss Dagstuhl, 2021. DOI: 10.4230/LIPIcs.ECOOP.2021.5.

References VI

[16] B. Düdder, G. T. Heineman, and J. Rehof, "Synthesizing Type-safe Compositions in Feature Oriented Software Designs using Staged Composition," in *Workshop on Modular Synthesis of Product Lines (ModSyn-PL) in Proceedings of the 19th International Software Product Line Conference (SPLC'15)*, Nashville, Tennessee: ACM, 2015, pp. 398–401, ISBN: 978-1-4503-3613-0.

[17] J. Bessai, A. Dudenhefner, B. Düdder, M. Martens, and J. Rehof, "Combinatory process synthesis," in *International Symposium on Leveraging Applications of Formal Methods*, Springer, 2016, pp. 266–281.

References VII

[18] B. Düdder, G. T. Heineman, and J. Rehof, "Towards Migrating Object-Oriented Frameworks to Enable Synthesis of Product Line Members," in *Proceedings of the 19th International Software Product Line Conference (SPLC)*, ser. SPLC '15, Nashville, Tennessee: ACM, 2015, pp. 56–60. DOI: 10.1145/2791060.2791076. [Online]. Available: http://doi.acm.org/10.1145/2791060.2791076.

[19] H. Wu, B. Düdder, S. Jiang, and L. Wang, "Blockchain-based Reliable and Privacy-aware Crowdsourcing with Truth and Fairness Assurance," *IEEE Internet of Things Journal*, Jul. 2021. DOI: 10.1109/JIOT.2021.3097950.

References VIII

[20] X. Jiang, C. Ying, Y. Luo, and B. Düdder, "Incentive Mechanism Design for Uncertain Tasks in Mobile Crowd Sensing Systems Utilizing Smart Contract in Blockchain," in *Proceedings of 18th International Conference on Collaborative Computing (CollabCom 2022)*, Springer, 2022.

[21] T. Zuckmantel, B. Düdder, Y. Zhou, and T. Hildebrandt, "Event-Based Data-Centric Semantics for Consistent Data Management in Microservices," in *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems (DEBS 2022)*, Jun. 2022, pp. 97–102. DOI: 10.1145/3524860.3539807.

References IX

[22] T. Slaats, R. R. Mukkamala, T. Hildebrandt, and M. Marquard, "Exformatics declarative case management workflows as dcr graphs," in *Business process management*, Springer, 2013, pp. 339–354.