A Type-Based Approach to Component-Oriented Synthesis

Lecture 0: Lecture Overview

Jakob Rehof
TU Dortmund University, Germany

Warsaw University Open Lectures for PhD Students in Computer Science (PhD open)

University of Warsaw, Poland, December 1-2 2017

Acknowledgements

- My students in Dortmund (former and present) including Jan Bessai, Boris Düdder (now Copenhagen), Andrej Dudenhefner, Moritz Martens (now in industry), Anna Vasileva
- Collaborators, including Paweł Urzyczyn (Warsaw), George Heineman (WPI Boston), Ugo de'Liguoro (Torino)
- Colleagues, including Mariangiola Dezani, Simona Ronchi Della Rocca, Mario Coppo and the Torino λ-calculus group, Roger Hindley (Swansey), Aleksy Schubert (Warsaw)

Program Synthesis Problems

In general, program synthesis problems are of the form:

- Given a specification ϕ, construct (if possible) a program P such that $P \models \phi$

Program Synthesis Problems

In general, program synthesis problems are of the form:

- Given a specification ϕ, construct (if possible) a program P such that $P \models \phi$

Here we will be concerned with *type-based* synthesis problems, where ϕ is a type in some type theory:

- Given a type τ and a type environment Γ, construct (if possible) an expression M such that $\Gamma \vdash M : \tau$

Program Synthesis Problems

In general, program synthesis problems are of the form:

- Given a specification ϕ, construct (if possible) a program P such that $P \models \phi$

Here we will be concerned with *type-based* synthesis problems, where ϕ is a type in some type theory:

- Given a type τ and a type environment Γ, construct (if possible) an expression M such that $\Gamma \vdash M : \tau$

Such problems have a theoretical foundation in the *inhabitation problem* for the type theory in question:

- Given a type τ and a type environment Γ, does there exist an expression M such that $\Gamma \vdash M : \tau$?

Program Synthesis Problems

In general, program synthesis problems are of the form:

- Given a specification ϕ, construct (if possible) a program P such that $P \models \phi$

Here we will be concerned with *type-based* synthesis problems, where ϕ is a type in some type theory:

- Given a type τ and a type environment Γ, construct (if possible) an expression M such that $\Gamma \vdash M : \tau$

Such problems have a theoretical foundation in the *inhabitation problem* for the type theory in question:

- Given a type τ and a type environment Γ, does there exist an expression M such that $\Gamma \vdash M : \tau$?

Such a problem can usually be understood as the problem of *provability* in a corresponding *constructive* logic.

Goals and Plan

- Introduction to the inhabitation problem, a fundamental problem in type theory with applications to proof search and synthesis.
- Applications in one particular framework for type-based synthesis (CLS).

Goals and Plan

- Introduction to the inhabitation problem, a fundamental problem in type theory with applications to proof search and synthesis.
- Applications in one particular framework for type-based synthesis (CLS).

- Day 1 (Lectures 0 – 2)
  - Introduction and overview
  - Introduction to simple typed λ-calculus and combinatory logic
  - The inhabitation problem in simple typed λ-calculus and combinatory logic
- Day 2 (Lectures 3 – 5)
  - Inhabitation in intersection types and bounded combinatory logic
  - Applications in CLS, Combinatory Logic Synthesis Framework
  - Fundamental research on inhabitation: Dimensional intersection type calculus

Combinatory Logic Synthesis (CLS)

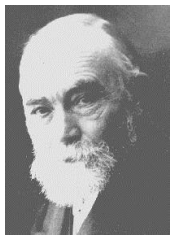
A type-theoretic approach to component-oriented synthesis

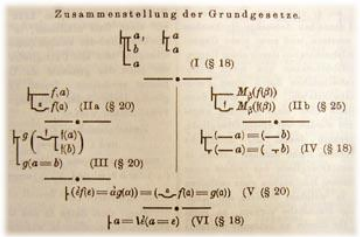
Component-oriented Synthesis

Synthesis *relative* to *library* (repository) of components

Combinatory Logic Synthesis (CLS)

Libraries need *classification systems* to enable *retrieval and composition*



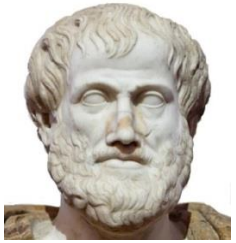


$$(\forall\, x; \forall\, y; \forall\, z; (p(x,y) \wedge p(y,z) \rightarrow p(x,z)) \wedge$$
$$\forall\, x; \forall\, y; (p(x,y) \rightarrow \neg p(y,x)) \wedge$$
$$\exists\, x; (p(a,x) \wedge p(x,b))$$
$$) \rightarrow \neg p(b,a)$$

Bottom-up specification

Hoare logic

CLS

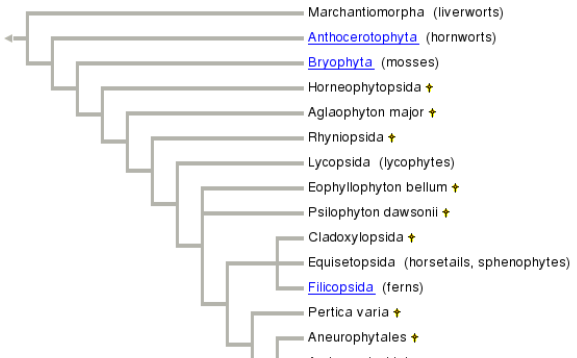


Classification

Taxonomy

…

Types



Combinatory Logic Synthesis (CLS)

A type-theoretic approach to component-oriented synthesis

Über die Bausteine der mathematischen Logik.

Von

M. Schönfinkel in Moskau[1]).

§ 1.

Es entspricht dem Wesen der axiomatischen Methode, wie sie heute vor allem durch die Arbeiten Hilberts zur Anerkennung gelangt ist, daß man nicht allein hinsichtlich der Zahl und des Gehalts der *Axiome* nach möglichster Beschränkung strebt, sondern auch die Anzahl der als undefiniert zugrunde zu legenden *Begriffe* so klein wie möglich zu machen sucht, indem man nach Begriffen fahndet, die vorzugsweise geeignet sind, um aus ihnen alle anderen Begriffe des fraglichen Wissenszweiges aufzubauen. Begreiflicherweise wird man sich im Sinne dieser Aufgabe bezüglich des Verlangens nach Einfachheit der an den Anfang zu stellenden Begriffe entsprechend bescheiden müssen.

Bekanntlich lassen sich die grundlegenden *Aussagenverknüpfungen* der mathematischen Logik, die ich hier in der von Hilbert in seinen Vorlesungen verwendeten Bezeichnungsweise wiedergebe:

$$\bar{a}, \quad a \vee b, \quad a \,\&\, b, \quad a \rightarrow b, \quad a \sim b$$

[1]) Die folgenden Gedanken wurden vom Verfasser am 7. Dez. 1920 vor der Mathematischen Gesellschaft in Göttingen vorgetragen. Ihre formale und stilistische Durcharbeitung für diese Veröffentlichung wurde von H. Behmann in Göttingen übernommen.

Combinatory Logic Synthesis (CLS)

A type-theoretic approach to component-oriented synthesis

Can we use *inhabitation in combinatory logic with intersection types* as a foundation for component-oriented, type-based synthesis?

- Typed combinators $X : \tau$ as named interfaces
- Automated composition synthesis via inhabitation
- Intersection types as *semantic types* (cf. also Haack,Wells,Yakobowski et al. [Haa+02; WY05]) for specification
- Beyond purely functional composition via meta-programming – compose a meta-program which, when executed, computes (say) a Java program

Example Repository

$$
\begin{aligned}
\Gamma = \{ \quad & \mathit{customerForm} : (\texttt{String} \rightarrow \texttt{java.net.URL} \rightarrow \texttt{OptionSelection} \rightarrow \texttt{Form}) \\
& \mathit{dropDownSelector} : (\texttt{java.net.URL} \rightarrow \texttt{OptionSelection}) \\
& \mathit{radioButtonSelector} : (\texttt{java.net.URL} \rightarrow \texttt{OptionSelection}) \\
& \mathit{companyTitle} : \texttt{String} \\
& \mathit{databaseLocation} : \texttt{java.net.URL} \\
& \mathit{logoLocation} : \texttt{java.net.URL} \\
& \mathit{alternateLogoLocation} : \texttt{java.net.URL} \quad \}
\end{aligned}
$$

From: Scala integrated framework for CLS by Bessai, Düdder, Dudenhefner, Heinemann

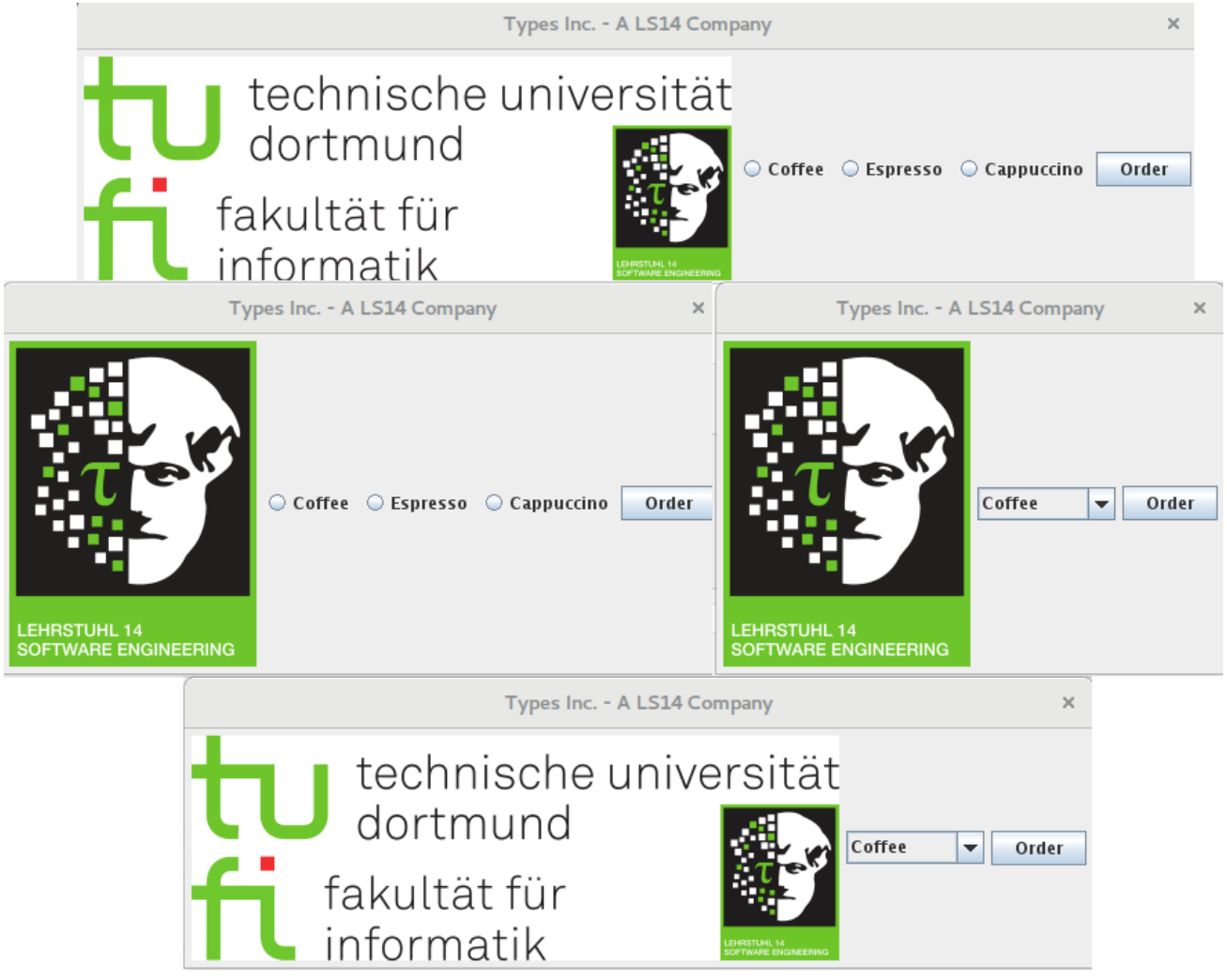
Example Repository with Semantic (Intersection) Types

$$
\begin{aligned}
\Gamma = \{\ & \mathit{customerForm} : (\texttt{String} \to \texttt{java.net.URL} \to \texttt{OptionSelection} \to \texttt{Form}) \cap \\
& \quad (\mathit{Title} \to \mathit{Location}(\mathit{Logo}) \to \mathit{ChoiceDialog}(\alpha) \to \mathit{OrderMenu}(\alpha)) \\
& \mathit{dropDownSelector} : (\texttt{java.net.URL} \to \texttt{OptionSelection}) \cap \\
& \quad (\mathit{Location}(\mathit{Database}) \to \mathit{ChoiceDialog}(\mathit{DropDown})) \\
& \mathit{radioButtonSelector} : (\texttt{java.net.URL} \to \texttt{OptionSelection}) \cap \\
& \quad (\mathit{Location}(\mathit{Database}) \to \mathit{ChoiceDialog}(\mathit{RadioButtons})) \\
& \mathit{companyTitle} : \texttt{String} \cap \mathit{Title} \\
& \mathit{databaseLocation} : \texttt{java.net.URL} \cap \mathit{Location}(\mathit{Database}) \\
& \mathit{logoLocation} : \texttt{java.net.URL} \cap \mathit{Location}(\mathit{Logo}) \\
& \mathit{alternateLogoLocation} : \texttt{java.net.URL} \cap \mathit{Location}(\mathit{Logo}) \quad \}
\end{aligned}
$$

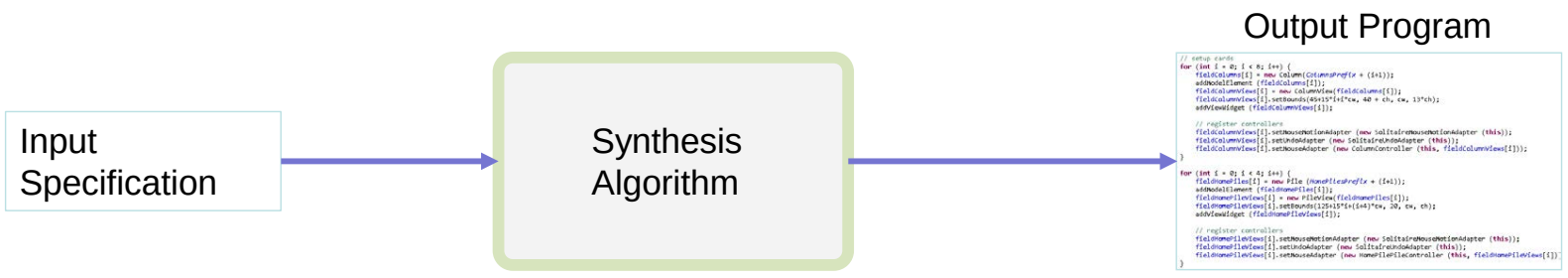
Combinator Implementation in Scala

```scala
class SwingRepository extends Repository {
  type Form = CompilationUnit
  type OptionSelection = CompilationUnit => CompilationUnit

  @combinator object customerForm extends CustomerForm {
    override def apply(title: String, logoLocation: URL, optionSelector: OptionSelection): CompilationUnit = {
      val file = scala.io.Source.fromInputStream(getClass.getResourceAsStream("CustomerForm.java")).mkString
      val form = Java(file).compilationUnit()
      val cls = form.getClassByName("CustomerForm").get
      val initMethod = cls.getMethodsByName("initComponents").get(0)

      optionSelector(form)

      form.addImport("java.net.URL")
      initMethod
        .getBody.get()
        .addStatement(0, Java(
          s"""
             |try {
             |  this.add(new JLabel(new ImageIcon(new URL("$logoLocation"))));
             |} catch (Exception e) {
             |
             |}""".stripMargin).statement())

      initMethod
        .getBody.get()
        .addStatement(0, Java(s"""this.setTitle("$title");""").statement())

      form
    }
  }
```

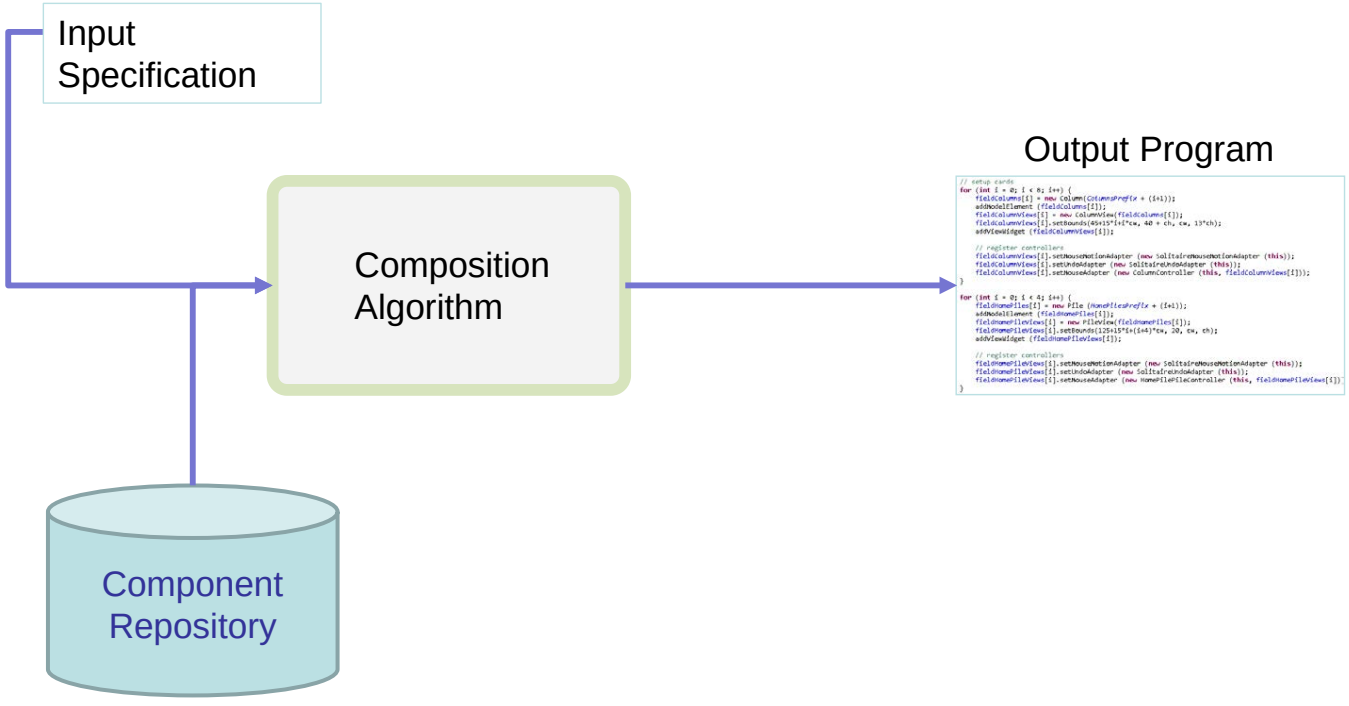
Variation, Configuration, and Product Lines



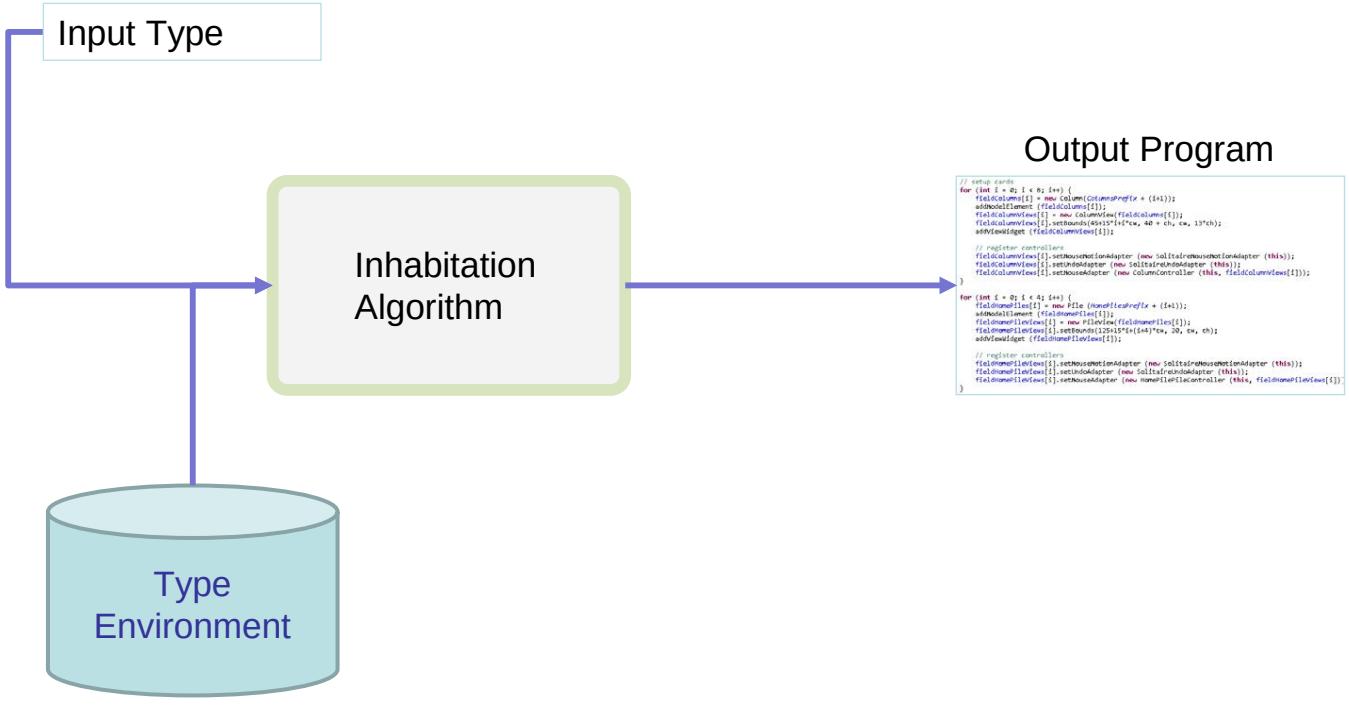
General Synthesis



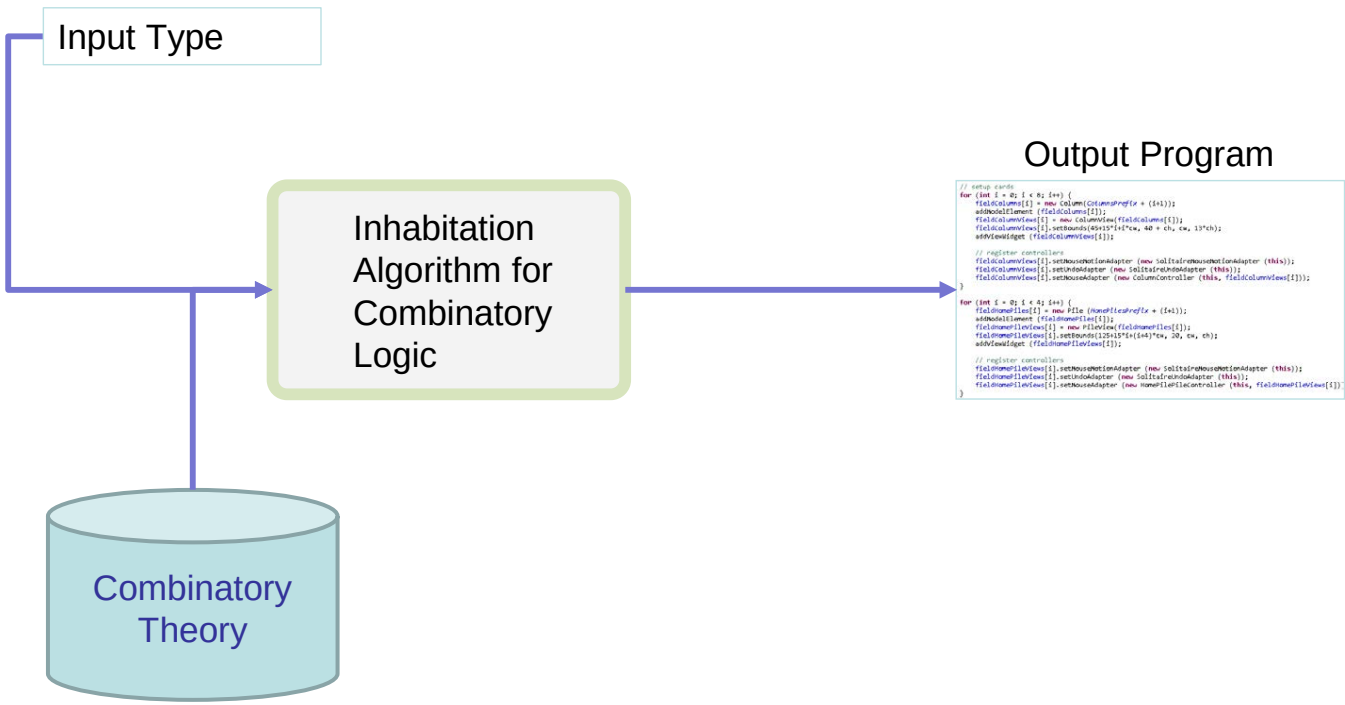
Component-Oriented Synthesis



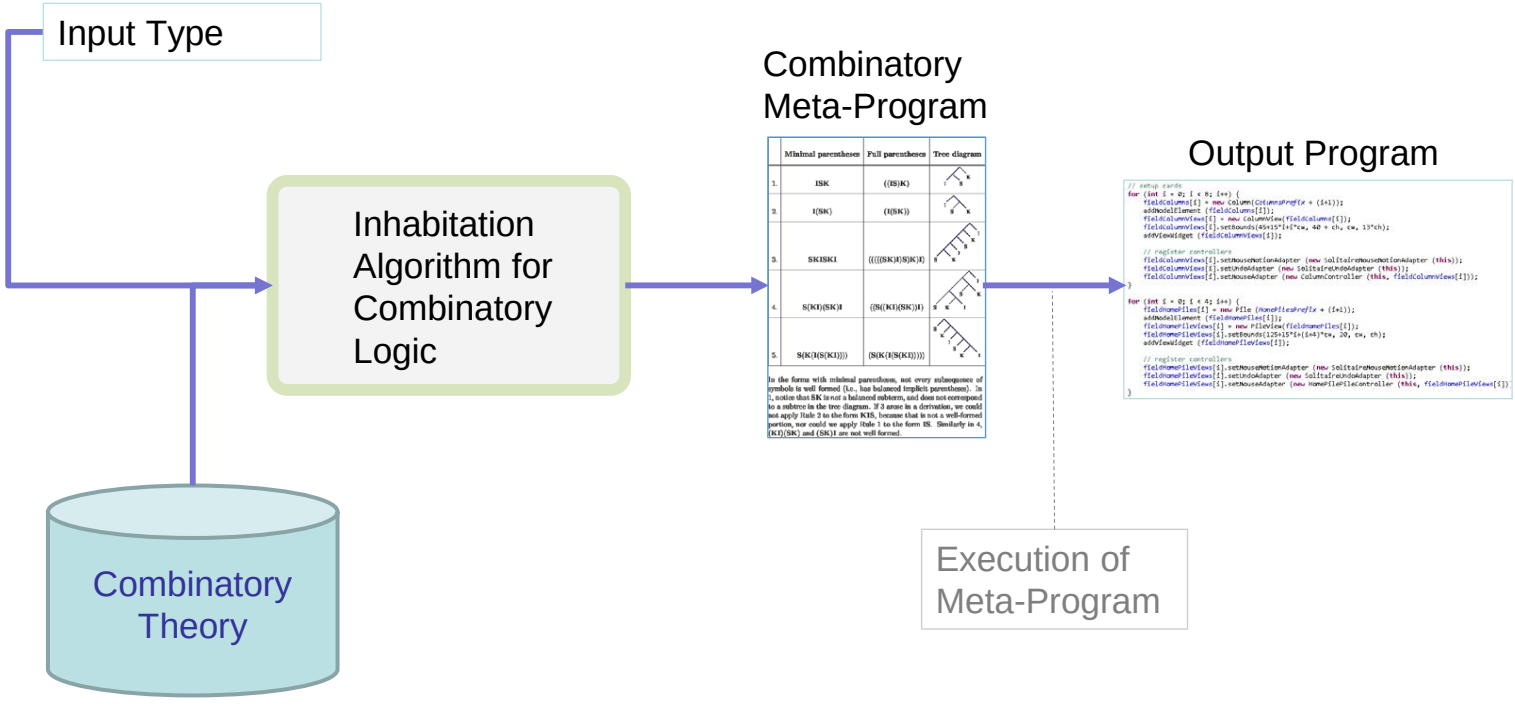
Type-Based Synthesis



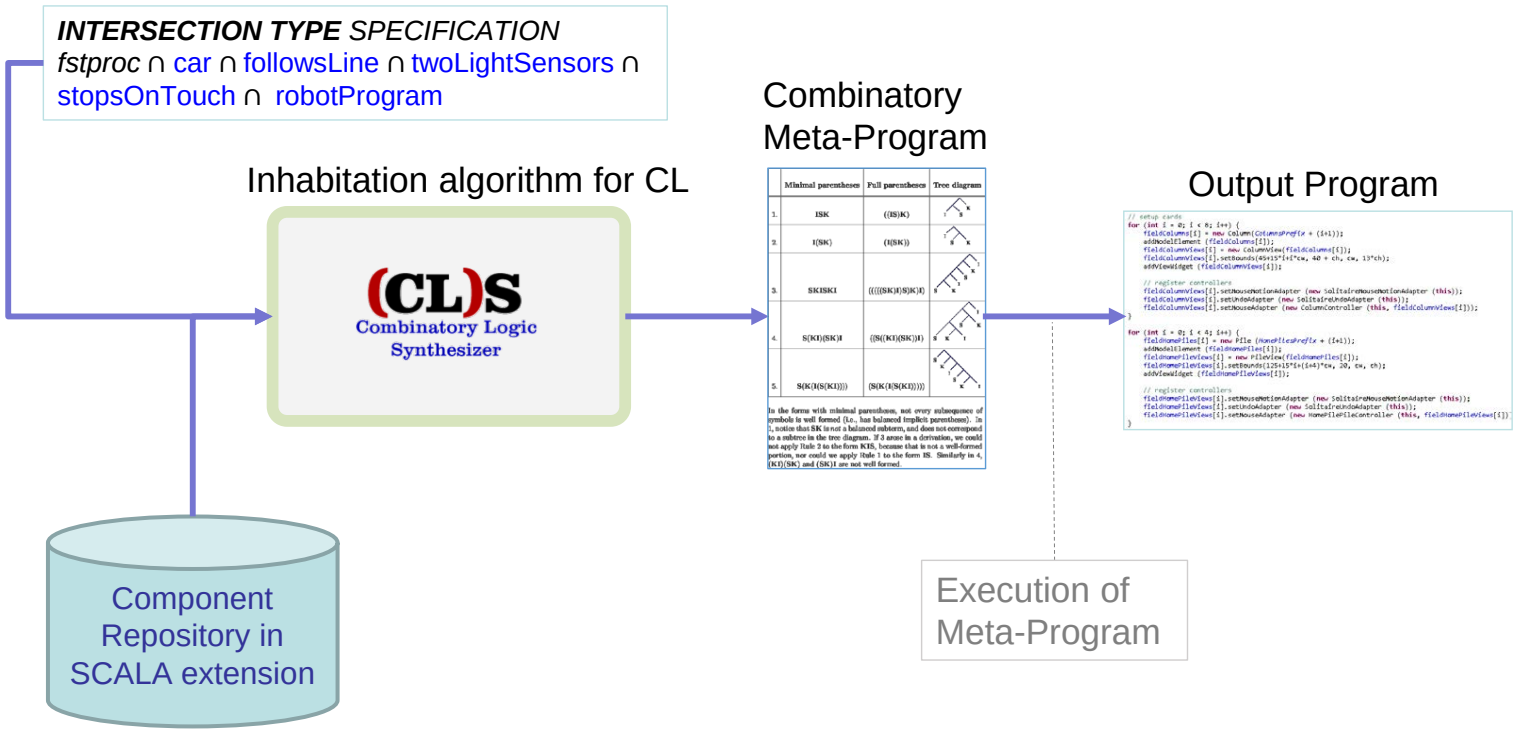
Component-Oriented Synthesis based on Combinatory Logic (CL)



Combinatory Logic Synthesis (CLS)



CLS Framework



Combinatory Logic Synthesis (CLS)

This idea was developed in a series of papers, including:

- *Finite Combinatory Logic with Intersection Types*, TLCA 2011 [RU11]
  - First mention of CL with intersection types as semantic specifications for synthesis via inhabitation
- *Bounded Combinatory Logic*, CSL 2012 [Düd+12]
  - Complexity hierarchy for *relativized inhabitation* with CL
- *Using Inhabitation in Bounded Combinatory Logic with Intersection Types for Composition Synthesis*, ITRS 2012 [Düd+12]
  - First short experiments
- *Towards Combinatory Logic Synthesis*, BEAT 2013 [Reh13]
  - Outline of research program
- *Design and Synthesis from Components (Dagstuhl Seminar 14232)*, 2014 (organized by Rehof and Vardi) [RV14]
  - The idea of synthesis from components
- *Staged Composition Synthesis*, ESOP 2014 [DMR14]
  - Moving to the meta-level via modal types
- *Mixin Composition Synthesis Based on Intersection Types*, TLCA 2015 [Bes+15]
  - First study of object-oriented features
- The idea of using intersection types as foundation for type-based synthesis also taken up for λ-calculus inhabitation, see Frankle, Osera, Walker, Zdancewic, POPL 2016 [Fra+16]

Combinatory Logic Synthesis (CLS)

It is currently being developed into a meta-programming framework based on a combinatory extension to Scala led by Jan Bessai (TU Dortmund), Boris Düdder (U Copenhagen), George Heineman (WPI Boston)

- *Combinatory Logic Synthesizer*, ISOLA 2014 [Bes+14]
- *Synthesizing Type-safe Compositions in Feature Oriented Software Designs using Staged Composition*, ModSyn-PL 2015 [DRH15]
- *Towards Migrating Object-Oriented Frameworks to Enable Synthesis of Product Line Members*, SPLC 2015 [Hei+15]
- *Combinatory Synthesis of Classes using Feature Grammars*, FACS 2015 [Bes+16b]
- *A Long and Winding Road Towards Modular Synthesis*, ISOLA 2016 [Hei+16]
- *Combinatory Process Synthesis*, ISOLA 2016 [Bes+16a]
- *Component-Oriented Synthesis via Algebras and Combinatory Logic*, in preparation

Foundational Study of Decision Problems in λ-Calculus with Intersection Types

- Understanding the borderline between decidability and undecidability for the classical decision problems (inhabitation and typability)
- Dimensional theory of inhabitation and typability for the intersection type system in λ-calculus
  - A. Dudenhefner and J.R.:
    Intersection Type Calculi of Bounded Dimension, POPL 2017
  - A. Dudenhefner and J.R.:
    Typablity in Bounded Dimension, LICS 2017

Bibliography I

Jan Bessai et al. “Combinatory Logic Synthesizer.” In: *ISoLA’14*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 8802. 2014, pp. 26–40.

Jan Bessai et al. “Mixin Composition synthesis Based on Intersection Types.” In: *LIPIcs-Leibniz International Proceedings in Informatics*. Vol. 38. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2015.

Jan Bessai et al. “Combinatory Process Synthesis.” In: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*. 2016, pp. 266–281. DOI: 10.1007/978-3-319-47166-2_19.

Bibliography II

Jan Bessai et al. “Combinatory Synthesis of Classes using Feature Grammars.” In: *Proceedings of the 12th International Conference on Formal Aspects of Component Software (FACS'15)*. Ed. by C. Braga and P.C. Olveczky. Vol. 9539. Springer, 2016, pp. 1–18. DOI: 10.1007/978-3-319-28934-2_7.

Boris Düdder, Moritz Martens, and Jakob Rehof. “Staged Composition Synthesis.” In: *ESOP 2014, Proceedings of European Symposium on Programming, Grenoble, France 2014*. Vol. 8410. LNCS. Springer, 2014, pp. 67–86.

Bibliography III

Boris Düdder, Jakob Rehof, and George T. Heineman. “Synthesizing Type-Safe Compositions in Feature Oriented Software Designs Using Staged Composition.” In: *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*. 2015, pp. 398–401. DOI: 10.1145/2791060.2793677. URL: http://doi.acm.org/10.1145/2791060.2793677.

Boris Düdder et al. “Bounded Combinatory Logic.” In: *CSL 2012, Proceedings of Computer Science Logic*. Vol. 16. LIPIcs. Schloss Dagstuhl, 2012, pp. 243–258.

Bibliography IV

Boris Düdder et al. “Using Inhabitation in Bounded Combinatory Logic with Intersection Types for Composition Synthesis.” In: *Proceedings Sixth Workshop on Intersection Types and Related Systems, ITRS 2012, Dubrovnik, Croatia, 29th June 2012*. Ed. by Stéphane Graham-Lengrand and Luca Paolini. Vol. 121. EPTCS. 2012, pp. 18–34. DOI: 10.4204/EPTCS.121.2. URL: http://dx.doi.org/10.4204/EPTCS.121.2.

Jonathan Frankle et al. “Example-Directed Synthesis: A Type-Theoretic Interpretation.” In: *POPL'16*. ACM. 2016, pp. 802–815.

Christian Haack et al. “Fully Automatic Adaptation of Software Components Based on Semantic Specifications.” In: *AMAST*. Vol. 2422. LNCS. Springer, 2002, pp. 83–98.

Bibliography V

George T. Heineman et al. “Towards Migrating Object-Oriented Frameworks to Enable Synthesis of Product Line Members.” In: *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*. 2015, pp. 56–60. DOI: 10.1145/2791060.2791076. URL: http://doi.acm.org/10.1145/2791060.2791076.

George T. Heineman et al. “A Long and Winding Road Towards Modular Synthesis.” In: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*. 2016, pp. 303–317. DOI: 10.1007/978-3-319-47166-2_21.

Bibliography VI

Jakob Rehof. “Towards Combinatory Logic Synthesis.” In: *BEAT 2013, 1st International Workshop on Behavioural Types*. ACM, 2013.

Jakob Rehof and Paweł Urzyczyn. “Finite Combinatory Logic with Intersection Types.” In: *TLCA 2011, Proceedings of Typed Lambda Calculus and Applications*. Vol. 6690. LNCS. Springer, 2011, pp. 169–183.

Jakob Rehof and Moshe Y. Vardi. “Design and Synthesis from Components. Dagstuhl Seminar 14232.” In: vol. 7941. Dagstuhl Reports. http://dx.doi.org/10.4230/DagRep.4.6.29. 2014.

Joe B. Wells and Boris Yakobowski. “Graph-Based Proof Counting and Enumeration with Applications for Program Fragment Synthesis.” In: *LOPSTR 2004*. Vol. 3573. LNCS. Springer, 2005, pp. 262–277.