Finite Combinatory Logic with Predicates

Andrej Dudenhefner
TU Dortmund University, Germany

Christoph Stahl
TU Dortmund University, Germany

Constantin Chaumet
TU Dortmund University, Germany

Felix Laarmann
TU Dortmund University, Germany

Jakob Rehof
TU Dortmund University, Germany
Lamarr Institute for Machine Learning and Artificial Intelligence, Dortmund, Germany

Abstract

Type inhabitation in extensions of Finite Combinatory Logic (FCL) is the mechanism underlying various component-oriented synthesis frameworks. In FCL inhabitant sets correspond to regular tree languages and vice versa. Therefore, it is not possible to specify non-regular properties of inhabitants, such as (dis)equality of subterms. Additionally, the monomorphic nature of FCL oftentimes hinders concise specification of components.

We propose a conservative extension to FCL by quantifiers and predicates, introducing a restricted form of polymorphism. In the proposed type system (FCLP) inhabitant sets correspond to decidable term languages and vice versa. As a consequence, type inhabitation in FCLP is undecidable. Based on results in tree automata theory, we identify a fragment of FCLP with the following two properties. First, the fragment enjoys decidable type inhabitation. Second, it allows for specification of local (dis)equality constraints for subterms of inhabitants.

For empirical evaluation, we implement a semi-decision procedure for type inhabitation in FCLP. We compare specification capabilities, scalability, and performance of the implementation to existing FCL-based approaches. Finally, we evaluate practical applicability via a case study, synthesizing mechanically sound robotic arms.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases combinatory logic, inhabitation, intersection types, program synthesis

Digital Object Identifier 10.4230/LIPIcs.TYPES.2023.2

Supplementary Material
Software (Source Code): `https://github.com/tudo-seal/clsp-python` [21]
archived at `swh:1:dir:8cb1770ba578c4d7e9a562f6513d82792e49e726`

Funding *Constantin Chaumet*: Deutsche Forschungsgemeinschaft (DFG, German Research Foundation – Project Number: 276879186).

1 Introduction

Type inhabitation in a type assignment system is the following problem: Given a type environment Γ and a type τ, is there a term M which can be assigned the type τ in the type environment Γ? Type inhabitation can be understood as the search for a program (term M) which satisfies a given specification (type τ) under given assumptions (type environment Γ). For (polymorphic) λ-calculi type inhabitation corresponds to program synthesis from scratch [29, 30]. In comparison, for (variants of) combinatory logic type inhabitation corresponds to program synthesis from given domain-specific components [31].

29th International Conference on Types for Proofs and Programs (TYPES 2023).
Editors: Delia Kesner, Eduardo Hermo Reyes, and Benno van den Berg; Article No. 2; pp. 2:1–2:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Finite Combinatory Logic with Intersection Types FCL($\cap, \leq$) [32] is a monomorphic variant of combinatory logic with intersection types [17] relativized to arbitrary bases. Type inhabitation in FCL($\cap, \leq$) is ExpTime-complete [32, Theorem 12], which provides the basis for the Combinatory Logic Synthesizer (CLS) [8, 4]. CLS is a domain-agnostic program synthesis framework, and it has been applied in the following areas: Object-oriented program composition [7, 6], software product line design [25, 24], factory planning [37], motion planning [34], simulation model construction [28], and cyber-physical systems [14].

Previously, two extensions of FCL($\cap, \leq$) have been studied for which type inhabitation is decidable. First, Bounded Combinatory Logic [18] relaxes monomorphism to bounded schematism. Second, Finite Combinatory Logic with Boolean Queries [20] adds Boolean connectives $\wedge$, $\vee$, and $\neg$ atop the type language. A major limitation of FCL($\cap, \leq$) and both aforementioned extensions is that inhabitant sets correspond to regular tree languages and vice versa [32, Corollary 11]. Therefore, neither theory allows for specification of non-regular properties[1] of inhabitants, such as (dis)equality of subterms. Additionally, monomorphism (and also bounded schematism) oftentimes hinders concise specification of domain-knowledge.

In the present work, we propose *Finite Combinatory Logic with Predicates* (FCLP) as a conservative extension to FCL($\cap, \leq$), addressing the above shortcomings. The main distinguishing property of FCLP is that inhabitant sets correspond to decidable term languages and vice versa. The type language of FCLP encompasses intersection types with the following three additions:

Literals: Literals are nullary type constructors, which may also occur in argument position in combinatory terms (facilitating a restricted form of dependent types [1]).

Quantifiers and Variables: Quantifiers bind variables in types, which allows for a restricted form of polymorphism, improving conciseness of specification.

Predicates: Decidable predicates reference variables, and allow for specification of non-regular properties of inhabitants.

The inclusion of decidable predicates entails undecidability of type inhabitation. The main contribution of the present work is the identification of an expressive fragment of FCLP which strictly includes FCL($\cap, \leq$), and in which type inhabitation is decidable. We formally describe a decision procedure for type inhabitation in the identified fragment. The decision procedure is similar to the existing decision procedure for type inhabitation in FCL($\cap, \leq$) [4, Definition 29]. However, instead of representing an inhabitant set by a regular tree grammar (or, a regular tree automaton), we represent an inhabitant set by means of the minimal Herbrand model of a logic program[2] (or, a tree automaton with term constraints [33]).

For empirical evaluation, an algorithm for the construction of inhabitants is implemented in the Python programming language. The algorithm is evaluated on the basis of a case study, undertaken for both FCL($\cap, \leq$) and the identified fragment of FCLP.

Synopsis. The present work is structured as follows:

Section 2: Definition of FCLP (Definition 8), decidability of type checking (Theorem 18), and undecidability of type inhabitation (Theorem 20).

Section 3: Fragment of FCLP (Problem 31) with decidable type inhabitation (Theorem 42).

Section 4: Implementation of FCLP in the Python programming language.

Section 5: Empirical evaluation of FCLP at the basis of a case study.

Section 6: Conclusion and remarks on future work.

---

[1] Czajka et.al. [5] specify non-regular properties as external restrictions via term rewriting systems.

[2] Kallat et.al. [27] combine regular tree grammars with SMT constraints in a logic program.

2 FCLP

In this section we present the type assignment system *Finite Combinatory Logic with Predicates* (FCLP). The system FCLP adds three new constructs to the existing *Finite Combinatory Logic with Intersection Types* FCL($\cap, \leq$) [32, Figure 3]. First, *literals* are both types and term arguments, and facilitate a restricted form of dependent types. Second, *quantifiers* bind literal variables and term variables in types, and allow for polymorphic specification. Third, decidable *predicates* reference literal variables and term variables in types, and describe term properties, which are difficult to specify otherwise.

Intersection types with covariant constructors [7, Definition 15] extended with literals (Definition 1) constitute the core of the type language of FCLP.

▶ **Definition 1** (Intersection Types with Covariant Constructors and Literals).

$$\textsc{Intersection Types} \quad \mathbb{T} \ni \sigma, \tau, \rho \quad ::= \quad \omega \mid \sigma \to \tau \mid \sigma \cap \tau \mid c(\sigma) \mid l$$

where ω is the universal type, c ranges over an enumerable set of unary type constructors*, and l ranges over an enumerable set of* literals.

The intersection type constructor ($\cap$) is considered associative, commutative, and idempotent. Additionally, standard rules of intersection type subtyping [17, Definition 1.3] are extended to covariant constructors in the following Definition 2.

▶ **Definition 2** (Intersection Type Subtyping). *The relation ($\leq$) is the least preorder on intersection types closed under the following rules:*

$$\begin{array}{lll} \sigma \leq \omega & \omega \leq \omega \to \omega & \\ \sigma \cap \tau \leq \sigma & \sigma \cap \tau \leq \tau & \textit{if } \sigma \leq \tau_1 \textit{ and } \sigma \leq \tau_2 \textit{ then } \sigma \leq \tau_1 \cap \tau_2 \\ (\sigma \to \tau) \cap (\sigma \to \rho) \leq \sigma \to (\tau \cap \rho) & & \textit{if } \sigma \leq \sigma' \textit{ and } \tau \leq \tau' \textit{ then } \sigma' \to \tau \leq \sigma \to \tau' \\ c(\sigma) \cap c(\tau) \leq c(\sigma \cap \tau) & & \textit{if } \sigma \leq \sigma' \textit{ then } c(\sigma) \leq c(\sigma') \end{array}$$

If $\sigma \leq \tau$ and $\tau \leq \sigma$, then we identify σ and τ, writing $\sigma = \tau$.

▶ Remark 3. Some existing extensions of Finite Combinatory Logic [4, Chapter 3] contain the binary product type constructor $\sigma \times \tau$. Observing the equivalence $\sigma \times \tau = (\sigma \times \omega) \cap (\omega \times \tau)$ [4, Definition 5], we omit an explicit product type constructor and represent products as intersections $\pi_1(\sigma) \cap \pi_2(\tau)$, where π_1 and π_2 are unary type constructors. In general, an n-ary constructor applied to $\sigma_1, \ldots, \sigma_n$ is represented as the intersection $\bigcap_{i=1}^{n} c_i(\sigma_i)$ using unary type constructors $c_1, \ldots, c_n$.

In practice, literals are partitioned in collections such as integers, floating point numbers, or character strings. We write $l : t$ to signify that the literal l belongs to a collection with the collection identifier t.

We tacitly extend intersection types by literal variables, ranged over by α, β, γ, and define *parameterized types* (the type language of FCLP) as follows.

▶ **Definition 4** (Parameterized Types).

$$\textsc{Parameterized Types} \quad \varphi, \psi \quad ::= \quad \sigma \mid \langle \alpha : t \rangle \Rightarrow \varphi \mid \langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \varphi \mid P \Rightarrow \varphi$$

where t ranges over collection identifiers and P ranges over decidable predicates[3]*, possibly containing literal variables and term variables, ranged over by x, y, z. A literal variable α is* bound *in $\langle \alpha : t \rangle \Rightarrow \varphi$. A term variable x is* bound *in $\langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \varphi$.*

[3] A predicate P is decidable, if there exists an effective procedure deciding whether P holds for given arguments. Syntactically, a predicate can be consider a first-order logic formula.

Combinatory terms which may contain literals in argument position (Definition 5) constitute the term language of FCLP.

▶ **Definition 5** (Combinatory Terms and Arguments).

$$\begin{array}{llcl} \textsc{Combinatory Terms} & \mathbb{C} \ni M, N & ::= & A \mid M\,T \\ \textsc{Combinatory Arguments} & T & ::= & M \mid l \end{array}$$

where A, B, C range over an enumerable set of combinators*.*

▶ **Definition 6** (Closed and Open Types, Substitutions). *A parameterized type is* closed*, if every occurrence of a literal variable and every occurrence of a term variable is bound; otherwise the parameterized type is* open*.* Literal variable substitution *is denoted $\varphi[\alpha := l]$ and* term variable substitution *is denoted $\varphi[x := M]$.*

In addition to *type environments* (finite sets of typed combinators), we introduce *literal environments* which contain pairs $l : t$, signifying that the literal l is associated with the collection identifier t.

▶ **Definition 7** (Type and Literal Environments).

$$\begin{array}{llcl} \textsc{Type Environment} & \Gamma & ::= & \{A_1 : \varphi_1, \ldots, A_n : \varphi_n\} \textit{ where } \varphi_1, \ldots, \varphi_n \textit{ are closed} \\ \textsc{Literal Environment} & \Delta & ::= & \{l_1 : t_1, \ldots, l_n : t_n\} \end{array}$$

$$\begin{array}{lrcl} \textsc{Domain} & \mathrm{dom}(\{A_1 : \varphi_1, \ldots, A_n : \varphi_n\}) & = & \{A_1, \ldots, A_n\} \\ & \mathrm{dom}(\{l_1 : t_1, \ldots, l_n : t_n\}) & = & \{l_1, \ldots, l_n\} \\ \textsc{Range} & \mathrm{ran}(\{A_1 : \varphi_1, \ldots, A_n : \varphi_n\}) & = & \{\varphi_1, \ldots, \varphi_n\} \\ & \mathrm{ran}(\{l_1 : t_1, \ldots, l_n : t_n\}) & = & \{t_1, \ldots, t_n\} \end{array}$$

Finally, we give the rules of Finite Combinatory Logic with Predicates (FCLP), deriving *judgments* $\Gamma; \Delta \vdash M : \varphi$, where φ is closed.

▶ **Definition 8** (Finite Combinatory Logic with Predicates (FCLP)).

$$\frac{(A : \varphi) \in \Gamma}{\Gamma; \Delta \vdash A : \varphi}\ (\mathrm{Var}) \qquad \frac{\Gamma; \Delta \vdash M : P \Rightarrow \varphi \qquad P \textit{ holds}}{\Gamma; \Delta \vdash M : \varphi}\ (P\mathrm{E})$$

$$\frac{\Gamma; \Delta \vdash M : \langle \alpha : t \rangle \Rightarrow \varphi \qquad (l : t) \in \Delta}{\Gamma; \Delta \vdash M\,l : \varphi[\alpha := l]}\ (\langle\rangle\mathrm{E}) \qquad \frac{\Gamma; \Delta \vdash M : \langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \varphi \qquad \Gamma; \Delta \vdash N : \sigma}{\Gamma; \Delta \vdash M\,N : \varphi[x := N]}\ (\langle\!\langle\rangle\!\rangle\mathrm{E})$$

$$\frac{\Gamma; \Delta \vdash M : \sigma \qquad \sigma \leq \tau}{\Gamma; \Delta \vdash M : \tau}\ (\leq) \qquad \frac{\Gamma; \Delta \vdash M : \sigma \to \tau \qquad \Gamma; \Delta \vdash N : \sigma}{\Gamma; \Delta \vdash M\,N : \tau}\ (\to\mathrm{E})$$

▶ Remark 9. The above rules (Var), ($\leq$), ($\to$E), together with the derivable intersection introduction rule (Lemma 14) constitute the original FCL($\cap$,$\leq$) type system [32, Figure 3]. The additional rules $\langle\rangle$E, $\langle\!\langle\rangle\!\rangle$E, and PE mimic the pure type system application rule [1], where proofs that a predicate holds are irrelevant.

The notion of *paths* [18] is an essential ingredient in the algorithmic treatment of intersection type subtyping. Algebraically, paths are *prime factors* [4, Definition 10] into which each intersection type factorizes uniquely. In the following Definition 10 and Lemma 12 we recollect the notion and corresponding key property of paths.

▶ **Definition 10** (Path Decomposition [18, Lemma 1]).

$$\begin{array}{rcl} \mathbb{P}(\omega) & = & \emptyset \\ \mathbb{P}(\sigma \to \tau) & = & \{\sigma \to \tau' \mid \tau' \in \mathbb{P}(\tau)\} \\ \mathbb{P}(\sigma \cap \tau) & = & \mathbb{P}(\sigma) \cup \mathbb{P}(\tau) \\ \mathbb{P}(c(\sigma)) & = & \{c(\tau) \mid \tau \in \mathbb{P}(\sigma)\} \cup \{c(\omega)\} \\ \mathbb{P}(l) & = & \{l\} \end{array}$$

▶ Remark 11. If $\sigma \leq \tau$, then for all $\tau' \in \mathbb{P}(\tau)$ there exists $\sigma' \in \mathbb{P}(\sigma)$ such that $\sigma' \leq \tau'$ (easily shown for each rule in Definition 2). Therefore, $\sigma = \tau$ implies $\mathbb{P}(\sigma) = \mathbb{P}(\tau)$.

▶ **Lemma 12.** *We have $\rho \leq \sigma_1 \to \cdots \to \sigma_k \to \tau$ iff there exists a (possibly empty) set $\{\sigma_1^1 \to \cdots \to \sigma_k^1 \to \tau^1, \ldots, \sigma_1^m \to \cdots \to \sigma_k^m \to \tau^m\} \subseteq \mathbb{P}(\rho)$ such that*

1. $\sigma_j \leq \bigcap_{i=1}^m \sigma_j^i$ *for* $j = 1 \ldots k$

2. $\bigcap_{i=1}^m \tau^i \leq \tau$

where the empty intersection denotes the universal type ω.

Proof. Immediate consequence of beta-soundness [2, Lemma 2.4] (shown inductively using the definition of intersection type subtyping). ◀

The following Lemma 13 (cf. [32, Lemma 4]) characterizes derivable judgments in FCLP.

▶ **Lemma 13** (Path Lemma). *We have $\Gamma; \Delta \vdash A\, T_1 \ldots T_n : \tau$ iff for some typed combinator $(A : e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \rho) \in \Gamma$ there exists a literal substitution θ such that*

1. *There exists a term substitution ξ such that for $i = 1, \ldots, m$ we have*
   - **a.** *if $e_i = \langle \alpha : t \rangle$, then $\theta(\alpha) = T_i$ is a literal and $(T_i : t) \in \Delta$*
   - **b.** *if $e_i = \langle\!\langle x : \sigma \rangle\!\rangle$, then $\xi(x) = T_i \in \mathbb{C}$ and $\Gamma; \Delta \vdash T_i : \theta(\sigma)$*
   - **c.** *if $e_i = P$, then $\xi(\theta(P))$ holds*

2. *Let $k = n - m + p$ where p is the number of predicates in $\{e_1, \ldots, e_m\}$, there exists a (possibly empty) set $\{\sigma_1^1 \to \cdots \to \sigma_k^1 \to \tau^1, \ldots, \sigma_1^q \to \cdots \to \sigma_k^q \to \tau^q\} \subseteq \mathbb{P}(\theta(\rho))$ such that*
   - **a.** $\Gamma; \Delta \vdash T_{m-p+j} : \bigcap_{i=1}^q \sigma_j^i$ *for* $j = 1 \ldots k$
   - **b.** $\bigcap_{i=1}^q \tau^i \leq \tau$

Proof. The direction from right to left is obvious, observing that by Lemma 12 we have $\theta(\rho) \leq (\bigcap_{i=1}^q \sigma_1^i) \to \cdots \to (\bigcap_{i=1}^q \sigma_k^i) \to \tau$. For the converse, we assume $\Gamma; \Delta \vdash A\, T_1 \ldots T_n : \tau$. Necessarily, there is some $(A : e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \rho) \in \Gamma$ such that each e_i is addressed by either ($\langle\rangle$E), ($\langle\!\langle\rangle\!\rangle$E), or ($P$E). We collect the according literal and term instances in substitutions θ and ξ and obtain properties (1.a)–(1.c). Let $k = n - m + p$ where p is the number of predicates in $\{e_1, \ldots, e_m\}$. We have that $\Gamma; \Delta \vdash A\, T_1 \ldots T_{n-k} : \theta(\rho)$ such that the remaining k arguments are addressed by the rules ($\to$E) and ($\leq$) where (shown by reordering) rule ($\leq$) never follows rule ($\to$E). Finally, by Lemma 12 we obtain properties (2.a)–(2.b). ◀

At first glance, the intersection introduction rule ($\cap$I) of FCL($\cap, \leq$) is missing from FCLP. However, using the above Lemma 13 the rule ($\cap$I) is derivable (cf. [4, Lemma 11]).

▶ **Lemma 14.** *The following rule is derivable:* $\dfrac{\Gamma; \Delta \vdash M : \sigma \qquad \Gamma; \Delta \vdash M : \tau}{\Gamma; \Delta \vdash M : \sigma \cap \tau}\ (\cap\mathrm{I})$.

Proof. We assume $\Gamma; \Delta \vdash M : \sigma$ and $\Gamma; \Delta \vdash M : \tau$ and proceed by induction on the term $M = A\, T_1 \ldots T_n$. W.l.o.g. $\sigma \neq \omega \neq \tau$. We have $(A : e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \rho) \in \Gamma$ and substitutions $\theta_1, \theta_2, \xi_1, \xi_2$ which satisfy Lemma 13.1 and agree on the free variables in ρ. Let $k = n - m + p$ where p is the number of predicates in $\{e_1, \ldots, e_m\}$. We have subsets $S_1, S_2 \subseteq \mathbb{P}(\theta_1(\rho)) = \mathbb{P}(\theta_2(\rho))$ such that $S_1 \cup S_2 = \{\sigma_1^1 \to \cdots \to \sigma_k^1 \to \tau^1, \ldots, \sigma_1^q \to \cdots \to \sigma_k^q \to \tau^q\}$ which satisfy Lemma 13.2. By the induction hypothesis, we have $\Gamma; \Delta \vdash T_{m-p+j} : \bigcap_{i=1}^q \sigma_j^i$ for $j = 1 \ldots k$, and we have $\bigcap_{i=1}^q \tau^i \leq \sigma \cap \tau$. By Lemma 13 we obtain $\Gamma; \Delta \vdash M : \sigma \cap \tau$. ◀

Since intersection introduction is derivable and the additional rules ($\langle\rangle$E),($\langle\!\langle\rangle\!\rangle$E), and (PE) refer to new type constructors, FCLP is a conservative extension of FCL($\cap, \leq$).

▶ **Corollary 15.** *Let Γ be a type environment such that $\mathrm{ran}(\Gamma) \subseteq \mathbb{T}$, let M be a combinatory term which does not contain literals, and let τ be an intersection type. We have $\Gamma; \emptyset \vdash M : \tau$ iff $\Gamma \vdash M : \tau$ is a derivable judgment in the type assignment system* FCL($\cap, \leq$).

Finally, we state the key decision problems: *Intersection type checking* (Problem 16) and *intersection type inhabitation* (Problem 17).

▶ **Problem 16** (Intersection Type Checking)**.** Given a type environment Γ, a literal environment Δ, a combinatory term M, and an intersection type τ, does $\Gamma; \Delta \vdash M : \tau$ hold?

▶ **Problem 17** (Intersection Type Inhabitation)**.** Given a type environment Γ, a literal environment Δ, and an intersection type τ, is there a combinatory term M such that $\Gamma; \Delta \vdash M : \tau$ holds?

Viewing the judgment $\Gamma; \Delta \vdash M : \tau$ in the context of component-oriented program synthesis, the type environment Γ contains domain-specific components specified by corresponding parameterized types, the literal environment Δ contains possible parameters, and the intersection type τ specifies desired programs.

While decidability of intersection type checking (Theorem 18) follows from Lemma 13, intersection type inhabitation is undecidable (Theorem 20).

▶ **Theorem 18.** *Intersection type checking* $\Gamma; \Delta \vdash M : \tau$ *is decidable.*

Proof. By induction on M, and as immediate consequence of Lemma 13 and decidability of each predicate occurring in parameterized types in $\mathrm{ran}(\Gamma)$. ◀

▶ **Theorem 19.** *Intersection type inhabitation (Problem 17) is semi-decidable.*

Proof. Since intersection type checking is decidable (Theorem 18), a semi-decision procedure may enumerate and type check each combinatory term as a potential inhabitant. ◀

▶ **Theorem 20.** *Intersection type inhabitation (Problem 17) is undecidable.*

Proof. We reduce the halting problem to intersection type inhabitation. Let $\mathcal{T}$ be a Turing machine and let $P(x)$ be a predicate on combinatory terms stating that $\mathcal{T}$ halts on the empty word after exactly $\mathrm{size}(x)$ steps (where $\mathrm{size}(M)$ is the number of combinator occurrences in M). For any combinatory term M we have that $P(M)$ is decidable. Let

$$\Gamma = \{A : c(\omega) \cap (c(\omega) \to c(\omega)), B : \langle\!\langle x : c(\omega) \rangle\!\rangle \Rightarrow P(x) \Rightarrow d(\omega)\}$$

If $\Gamma; \emptyset \vdash M : d(\omega)$ for some combinatory term M, then M is of shape $B\,N$ and $\mathcal{T}$ halts on the empty word after $\mathrm{size}(N)$ steps. Complementarily, if $\mathcal{T}$ halts on the empty word after exactly k steps, then we have $\Gamma; \emptyset \vdash B\,N : d(\omega)$ where $N = A(\cdots(A\,A)\cdots)$ is of size k. Therefore, $\mathcal{T}$ halts on the empty word iff there exists a term M such that $\Gamma; \emptyset \vdash M : d(\omega)$. ◀

▶ Remark 21. Emptiness and finiteness of the set $\{M \mid \Gamma; \Delta \vdash M : \tau\}$ of inhabitants are orthogonal problems. The proof of the above Theorem 20 gives a finite set of inhabitants with an undecidable emptiness question. Complementarily, let us consider the predicate $P'(x)$ on combinatory terms stating that $x = A$ or a given Turing machine $\mathcal{T}$ halts on the empty word after at most $\mathrm{size}(x)$ steps. The corresponding set of inhabitants is non-empty, and infinite iff $\mathcal{T}$ halts on the empty word.

Since intersection type checking in FCLP is decidable (Theorem 18), the set of inhabitants $\{M \mid \Gamma; \Delta \vdash M : \tau\}$ is decidable. Complementarily, the following Theorem 22 shows that each decidable set of combinatory terms can be described by some set of inhabitants.

▶ **Theorem 22.** *Let* $\mathcal{M}$ *be a decidable set of combinatory terms containing combinators drawn from a finite set* $\mathcal{A}$*. There exists a combinator* B *and a type environment* Γ *such that* $\mathcal{M} = \{M \mid \Gamma; \emptyset \vdash B\,M : \omega\}$.

Proof. Let $\Gamma' = \{A : \omega \mid A \in \mathcal{A}\}$ and $\Gamma = \Gamma' \cup \{B : \langle\!\langle x : \omega \rangle\!\rangle \Rightarrow (x \in \mathcal{M}) \Rightarrow \omega\}$ for a fresh combinator $B \notin \mathcal{A}$. Since $\mathcal{M} \subseteq \{M \mid \Gamma'; \emptyset \vdash M : \omega\}$, we have $\Gamma; \emptyset \vdash B\,M : \omega$ iff $M \in \mathcal{M}$. ◀

3 Decidable Inhabitation Fragment

While intersection type inhabitation (Problem 17) is undecidable in general (Theorem 20), FCLP contains fragments which enjoy decidable inhabitation. By Corollary 15 one such fragment is FCL($\cap, \leq$), for which inhabitation is ExpTime-complete [32, Theorem 12]. In the remainder of this section we present a fragment of FCLP which strictly includes FCL($\cap, \leq$) and enjoys decidable inhabitation. The key idea is a reduction from intersection type inhabitation (under certain restrictions on predicates) to emptiness of bottom-up tree automata with term constraints [33].

The following Definition 23 specifies the *arity* of a parameterized type. If a combinator is applied to a number of arguments exceeding the arity of its type, then the only type assigned to such an application is the universal type ω.

▶ **Definition 23** (Arity).

$$\begin{aligned}
\mathrm{ar}(\langle \alpha : t\rangle \Rightarrow \varphi) = \mathrm{ar}(\langle x : \sigma\rangle \Rightarrow \varphi) &= 1 + \mathrm{ar}(\varphi) \\
\mathrm{ar}(P \Rightarrow \varphi) &= \mathrm{ar}(\varphi) \\
\mathrm{ar}(\omega) = \mathrm{ar}(c(\sigma)) = \mathrm{ar}(l) = \mathrm{ar}(\alpha) &= 0 \\
\mathrm{ar}(\sigma \to \tau) &= 1 + \mathrm{ar}(\tau) && \textit{where } \tau \neq \omega \\
\mathrm{ar}(\sigma \cap \tau) &= \max\{\mathrm{ar}(\sigma), \mathrm{ar}(\tau)\}
\end{aligned}$$

▶ **Lemma 24** (Maximal Arity). *Given environments Γ, Δ, for a typed combinator $(A : \varphi) \in \Gamma$, an $n > \mathrm{ar}(\varphi)$, combinatory arguments $T_1, \ldots, T_n$, and an intersection type τ, we have that if $\Gamma; \Delta \vdash A\, T_1 \ldots T_n : \tau$, then $\tau = \omega$.*

Proof. For Γ, Δ, and $(A : e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \rho) \in \Gamma$ let $n > \mathrm{ar}(e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \rho)$ and let $n' = n - \mathrm{ar}(e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \omega)$. By induction on ρ for any literal substitution θ we have that $\sigma'_1 \to \cdots \to \sigma'_{n'} \to \tau' \notin \mathbb{P}(\theta(\rho))$ for any types $\sigma'_1, \ldots, \sigma'_{n'}, \tau'$. Assuming $\Gamma; \Delta \vdash A\, T_1 \ldots T_n : \tau$, by Lemma 13.2.b we obtain $\omega \leq \tau$, showing the claim. ◀

Combinatory terms can be naively represented as binary trees, having combinators and literals as leaves and binary term application as inner nodes. However, the naive representation is inappropriate for certain tree constraints. For example, brother equality and disequality constraints [10] compare the terms M and N in the application $M\, N$ for the naive representation, which is of little interest in practice. In the following, we represent a combinatory term $A\, T_1 \ldots T_n$ as the tree with root $A_{(n)}$ with n children $T_1, \ldots, T_n$.

▶ **Definition 25** (Tree Representation).

$$\begin{aligned}
\mathrm{tree}(l) &= l \\
\mathrm{tree}(A\, T_1 \ldots T_n) &= A_{(n)}(\mathrm{tree}(T_1), \ldots, \mathrm{tree}(T_n))
\end{aligned}$$

In general, for a combinator A we have infinitely many symbols $A_{(0)}, A_{(1)}, A_{(2)}, \ldots$, which is unsatisfactory for tree languages over a *finite* signature. Fortunately, relying on Lemma 24 for a typed combinator $A : \varphi$ we can reasonably bound the arity by $\mathrm{ar}(\varphi)$ in the following Definition 26.

▶ **Definition 26** (Arity Respect). *We say that a combinatory term $A\, T_1 \ldots T_n$* respects arities *in Γ, if $(A : \varphi) \in \Gamma$, $n \leq \mathrm{ar}(\varphi)$, and each T_i which is not a literal respects arities in Γ.*

The set of combinatory terms M which contain literals from $\mathrm{dom}(\Delta)$ and respect arities in Γ is denoted $\mathbb{C}(\Gamma, \Delta)$.

The following Lemma 27 shows that for intersection type inhabitation in the fragment corresponding to FCL($\cap, \leq$) (cf. Corollary 15) it suffices to consider inhabitants which respect arities in Γ.

▶ **Lemma 27.** *Let Γ be a type environment such that* $\text{ran}(\Gamma) \subseteq \mathbb{T}$*, let M be a combinatory term which does not contain literals, and let τ be an intersection type. If $\Gamma; \emptyset \vdash M : \tau$, then there exists a combinatory term $N \in \mathbb{C}(\Gamma, \emptyset)$ such that $\Gamma; \emptyset \vdash N : \tau$.*

Proof. Assume $\Gamma; \emptyset \vdash M : \tau$ where $\text{ran}(\Gamma) \subseteq \mathbb{T}$; we proceed by induction on the number of combinator occurrences in M. If $M \in \mathbb{C}(\Gamma, \emptyset)$ we obtain the claim, otherwise the derivation of $\Gamma; \emptyset \vdash M : \tau$ contains a subderivation $\Gamma; \emptyset \vdash A\, M_1 \dots M_{\text{ar}(\sigma)}\, M' : \rho$ for some $(A : \sigma) \in \Gamma$ and an intersection type ρ. By Lemma 24 we have $\rho = \omega$. Therefore, we can derive $\Gamma; \emptyset \vdash A\, M_1 \dots M_{\text{ar}(\sigma)} : \rho$, replace the original subderivation, and obtain the claim by the induction hypothesis. ◀

In general, for intersection type inhabitation the following Remark 28 shows that inhabitants which respect arities in Γ do not suffice.

▶ Remark 28. For the type environment $\Gamma = \{A : \omega, B : \langle\!\langle x : \omega \rangle\!\rangle \Rightarrow (x \neq A) \Rightarrow l\}$ we have $\Gamma; \emptyset \vdash B\,(A\,A) : l$. Since $\text{ar}(\omega) = 0$ we have $B\,(A\,A) \notin \mathbb{C}(\Gamma, \emptyset)$. The only term $N \in \mathbb{C}(\Gamma, \emptyset)$ such that $\Gamma; \emptyset \vdash N : \omega$ is the term $N = A$. Therefore, there is no combinatory term $M \in \mathbb{C}(\Gamma, \emptyset)$ such that $\Gamma; \emptyset \vdash M : l$.

We consider a fragment of FCLP for which any predicate occurring in parameterized types in $\text{ran}(\Gamma)$ is of specific shape (Definition 29 and Definition 30), motivated by tree automata with term constraints [33].

▶ **Definition 29** (Term Constraint). *For a type environment Γ, a predicate occurring in a parameterized type in* $\text{ran}(\Gamma)$ *is a* term constraint*, if it is either $x = M$ or $x \neq M$ where M is (abusing notation) an open term (possibly containing variables) which respects arities in Γ.*

▶ **Definition 30** (Literal Constraint). *A predicate P is a* literal constraint *if every free variable occurring in P is a literal variable.*

Restricting predicates to literal and term constraints we define the following inhabitation problem.

▶ **Problem 31** (Intersection Type Inhabitation with Literal and Term Constraints). Given a type environment Γ such that each predicate occurring in a parameterized type in $\text{ran}(\Gamma)$ is a literal constraint or a term constraint, a literal environment Δ, and an intersection type τ, is there a combinatory term $M \in \mathbb{C}(\Gamma, \Delta)$ such that $\Gamma; \Delta \vdash M : \tau$ holds?

We follow the approach initiated by Frühwirth [23] (an overview is given by Jacquemard [26, Section I.2]) and describe tree languages via finite sets of Horn clauses over certain first-order signatures. The following Definition 32 establishes suitable first-order signatures, based on environments and a set of intersection types.

▶ **Definition 32** (Signature). *For environments Γ, Δ and a finite set of intersection types Ξ the signature $\Sigma(\Gamma, \Delta, \Xi)$ contains the following:*
- *nullary function symbols l for each $l \in \text{dom}(\Delta)$*
- *n-ary function symbols $A_{(n)}$ for each $(A : \varphi) \in \Gamma$ and $n \leq \text{ar}(\varphi)$*
- *unary predicates Q_τ for each $\tau \in \Xi$*
- *unary predicates Q_t for each $t \in \text{ran}(\Delta)$*
- *a binary equality $(=)$ predicate and a binary disequality $(\neq)$ predicate*

The Herbrand universe over the signature $\Sigma(\Gamma, \Delta, \Xi)$ is $\{\mathrm{tree}(T) \mid T \in \mathbb{C}(\Gamma, \Delta) \cup \mathrm{dom}(\Delta)\}$. Horn clauses over the signature $\Sigma(\Gamma, \Delta, \Xi)$ are of shape $H \leftarrow H_1, \dots, H_m$, where H is the *head* of the clause and $H_1, \dots, H_m$ are *antecedents* of the clause. Specifically, we consider Horn clauses with heads of shape either $Q_t(l)$ such that $(l : t) \in \Delta$, or $Q_\tau(A_{(n)}(X_1, \dots, X_n))$ where $X_1, \dots, X_n$ are free first-order variables. Let us recall in the following Definition 33 the standard Herbrand semantics [36] for signatures $\Sigma(\Gamma, \Delta, \Xi)$.

▶ **Definition 33** (Model). *Let Γ, Δ be environments, let $M \in \mathbb{C}(\Gamma, \Delta)$ be a combinatory term, let Ξ be a finite set of intersection types, let $\mathcal{H}$ be a finite set of Horn clauses over the signature $\Sigma(\Gamma, \Delta, \Xi)$, and let $\tau \in \Xi$ be a type. We write $\mathcal{H} \Vdash Q_\tau(\mathrm{tree}(M))$ if $Q_\tau(\mathrm{tree}(M))$ is true in the smallest Herbrand model in which every Horn clause from $\mathcal{H}$ is true.*

Next, given environments Γ (restricted to literal and term constraints) and Δ, and an intersection type τ, we present a terminating algorithm INH which computes a set $\mathcal{H}$ of Horn clauses such that for any $M \in \mathbb{C}(\Gamma, \Delta)$ we have $\Gamma; \Delta \vdash M : \tau$ iff $\mathcal{H} \Vdash Q_\tau(\mathrm{tree}(M))$.

▶ **Definition 34** (Algorithm $\mathsf{INH}_{\Gamma,\Delta}(\tau, \Xi)$). *Let Γ be a type environment such that each predicate occurring in a parameterized type in $\mathrm{ran}(\Gamma)$ is a literal constraint or a term constraint and let Δ be a literal environment. For an intersection type τ and a set Ξ of intersection types let*

$$\mathsf{INH}_{\Gamma,\Delta}(\tau, \Xi) = \begin{cases} \emptyset & \text{if } \tau \in \Xi \\ \bigcup_{(A:\varphi) \in \Gamma} \mathsf{REC}^A_{\Gamma,\Delta,\tau,(\Xi \cup \tau)}((), \varphi, (), \emptyset) & \text{otherwise} \end{cases}$$

where Algorithm $\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}$ is defined as follows. The arguments of $\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}$ are
- *a list $\vec{X}$ of distinct first-order variables*
- *a parameterized type φ*
- *a list $\vec{H}$ of antecedents*
- *a finite set $\mathcal{H}$ of Horn clauses*

The result of $\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}(\vec{X}, \varphi, \vec{H}, \mathcal{H})$ is a set of Horn clauses computed as follows. Consider the shape of φ:

Case φ is $\langle \alpha : t \rangle \Rightarrow \psi$: *let Y be a fresh first-order variable and return*

$$\bigcup_{(l:t) \in \Delta} \mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}((\vec{X}, Y), \psi[\alpha := l], (\vec{H}, Q_t(Y), Y = l), \mathcal{H} \cup \{Q_t(l) \leftarrow\})$$

Case φ is $\langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \psi$: *let Y be a fresh first-order variable and return*

$$\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}((\vec{X}, Y), \psi[x := Y], (\vec{H}, Q_\sigma(Y)), \mathcal{H} \cup \mathsf{INH}_{\Gamma,\Delta}(\sigma, \Xi))$$

Case φ is $P \Rightarrow \psi$ such that P is closed: *if P does not hold return $\emptyset$, otherwise return*

$$\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}(\vec{X}, \psi, \vec{H}, \mathcal{H})$$

Case φ is $(X = M) \Rightarrow \psi$ where M may contain free first-order variables: *return*

$$\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}(\vec{X}, \psi, (\vec{H}, X = \mathrm{tree}(M)), \mathcal{H})$$

Case φ is $(X \neq M) \Rightarrow \psi$ where M may contain free first-order variables: *return*

$$\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}(\vec{X}, \psi, (\vec{H}, X \neq \mathrm{tree}(M)), \mathcal{H})$$

Case φ is ρ for some intersection type ρ: *return*

$$\mathcal{H} \cup \bigcup_{k=0}^{\mathrm{ar}(\rho)} \bigcup_{S \subseteq \mathbb{P}(\rho)} \mathcal{H}_{k,S} \text{ where } \mathcal{H}_{k,S} \text{ is defined as follows.}$$

If $S = \{\sigma_1^1 \to \cdots \to \sigma_k^1 \to \tau^1, \ldots, \sigma_1^q \to \cdots \to \sigma_k^q \to \tau^q\}$ and $\bigcap_{i=1}^q \tau^i \leq \tau$, then let $Y_1, \ldots, Y_k$ be fresh first-order variables, let $\sigma_j = \bigcap_{i=1}^q \sigma_j^i$ for $j = 1 \ldots k$, and let n be the length of the list $(\vec{X}, Y_1, \ldots, Y_k)$ in

$$\mathcal{H}_{k,S} = \{Q_\tau(A_{(n)}(\vec{X}, Y_1, \ldots, Y_k)) \leftarrow \vec{H}, Q_{\sigma_1}(Y_1), \ldots, Q_{\sigma_k}(Y_k)\} \cup \bigcup_{j=1}^{k} \mathsf{INH}_{\Gamma,\Delta}(\sigma_j, \Xi)$$

Otherwise, $\mathcal{H}_{k,S} = \emptyset$.

The following Example 35 illustrates an invocation of Algorithm INH for the type environment from Remark 28.

▶ **Example 35.** Consider the type environment $\Gamma = \{A : \omega, B : \langle\!\langle x : \omega \rangle\!\rangle \Rightarrow (x \neq A) \Rightarrow l\}$ from Remark 28 where l is some literal. We have

$$\begin{aligned}
&\mathsf{INH}_{\Gamma,\emptyset}(l, \emptyset)\\
&= \mathsf{REC}^A_{\Gamma,\emptyset,l,\{l\}}\big((), \omega, (), \emptyset\big) \cup \mathsf{REC}^B_{\Gamma,\emptyset,l,\{l\}}\big((), \langle\!\langle x : \omega \rangle\!\rangle \Rightarrow (x \neq A) \Rightarrow l, (), \emptyset\big)\\
&= \emptyset \cup \mathsf{REC}^B_{\Gamma,\emptyset,l,\{l\}}\big((Y), (Y \neq A) \Rightarrow l, (Q_\omega(Y)), \emptyset \cup \mathsf{INH}_{\Gamma,\emptyset}(\omega, \{l\})\big)\\
&= \mathsf{REC}^B_{\Gamma,\emptyset,l,\{l\}}\big((Y), l, (Q_\omega(Y), (Y \neq A_{(0)})), \mathsf{INH}_{\Gamma,\emptyset}(\omega, \{l\})\big)\\
&= \mathsf{INH}_{\Gamma,\emptyset}(\omega, \{l\}) \cup \{Q_l(B_{(1)}(Y)) \leftarrow Q_\omega(Y), (Y \neq A_{(0)})\}\\
&= \ldots\\
&= \{Q_\omega(A_{(0)}) \leftarrow,\\
&\qquad Q_\omega(B_{(1)}(Z)) \leftarrow Q_\omega(Z), (Z \neq A_{(0)}),\\
&\qquad Q_l(B_{(1)}(Y)) \leftarrow Q_\omega(Y), (Y \neq A_{(0)})\}
\end{aligned}$$

In accordance with Remark 28, in the smallest Herbrand model in which every Horn clause from the above set $\mathsf{INH}_{\Gamma,\emptyset}(l, \emptyset)$ is true we have

- $Q_\omega = \{A_{(0)}\} = \{\mathrm{tree}(M) \mid M \in \mathbb{C}(\Gamma, \emptyset) \text{ such that } \Gamma; \emptyset \vdash M : \omega\}$
- $Q_l = \emptyset = \{\mathrm{tree}(M) \mid M \in \mathbb{C}(\Gamma, \emptyset) \text{ such that } \Gamma; \emptyset \vdash M : l\}$

The following Example 36 illustrates the result of Algorithm INH in the presence of literal constraints (cf. Section 5).

▶ **Example 36.** Consider the literal environment $\Delta = \{0 : \mathrm{int}, 1 : \mathrm{int}, 2 : \mathrm{int}, 3 : \mathrm{int}\}$ and the type environment $\Gamma = \{A : 0, B : \langle \alpha : \mathrm{int} \rangle \Rightarrow \langle \beta : \mathrm{int} \rangle \Rightarrow (\beta = \alpha + 1) \Rightarrow \alpha \to \beta\}$. We have

$$\begin{aligned}
&\mathsf{INH}_{\Gamma,\Delta}(2, \emptyset)\\
&= \{Q_0(A_{(0)}) \leftarrow,\\
&\qquad Q_1(B_{(3)}(X_1, X_2, X_3)) \leftarrow Q_{\mathrm{int}}(X_1), (X_1 = 0), Q_{\mathrm{int}}(X_2), (X_2 = 1), Q_0(X_3),\\
&\qquad Q_2(B_{(3)}(Y_1, Y_2, Y_3)) \leftarrow Q_{\mathrm{int}}(Y_1), (Y_1 = 1), Q_{\mathrm{int}}(Y_2), (Y_2 = 2), Q_1(Y_3),\\
&\qquad Q_{\mathrm{int}}(0) \leftarrow, Q_{\mathrm{int}}(1) \leftarrow, Q_{\mathrm{int}}(2) \leftarrow\}
\end{aligned}$$

In the smallest Herbrand model in which every Horn clause from the above set $\mathsf{INH}_{\Gamma,\Delta}(2, \emptyset)$ is true we have $Q_0 = \{A_{(0)}\}$, $Q_1 = \{B_{(3)}(0, 1, A_{(0)})\}$, $Q_2 = \{B_{(3)}\big(1, 2, B_{(3)}(0, 1, A_{(0)})\big)\}$, and $Q_{\mathrm{int}} = \{0, 1, 2\}$. Specifically, we have $Q_i = \{\mathrm{tree}(M) \mid M \in \mathbb{C}(\Gamma, \Delta) \text{ such that } \Gamma; \Delta \vdash M : i\}$ for $i \in \{0, 1, 2\}$. The literal $3 \in \mathrm{dom}(\Delta)$ does not occur in Horn clauses in $\mathsf{INH}_{\Gamma,\Delta}(2, \emptyset)$.

Termination of $\mathsf{INH}_{\Gamma,\Delta}(\tau, \Xi)$ is shown using an upper bound on the set Ξ of types and the fact that Ξ strictly increases in recursive invocations of $\mathsf{INH}_{\Gamma,\Delta}$.

▶ **Lemma 37.** *For any type environment* Γ*, literal environment* Δ*, intersection type* τ*, and set* Ξ *of intersection types we have that Algorithm* $\mathsf{INH}_{\Gamma,\Delta}(\tau,\Xi)$ *terminates.*

Proof. Recursive invocations of $\mathsf{INH}_{\Gamma,\Delta}$ increase the set Ξ by the considered type τ, such that for some literal substitution θ with range $\mathrm{dom}(\Delta)$ one of the following conditions holds:

- $\tau = \theta(\sigma)$ such that $\langle\!\langle x : \sigma \rangle\!\rangle$ is a binder occurring in a parameterized type in $\mathrm{ran}(\Gamma)$
- $\tau = \bigcap_{i=1}^{q} \sigma_j^i$ such that $\{\sigma_1^1 \to \cdots \to \sigma_k^1 \to \tau^1, \ldots, \sigma_1^q \to \cdots \to \sigma_k^q \to \tau^q\} \subseteq \mathbb{P}(\theta(\rho))$ for some ρ occurring in a parameterized type in $\mathrm{ran}(\Gamma)$ and $k \leq \mathrm{ar}(\theta(\rho))$

Since Γ, Δ, the number of literal substitution θ with range $\mathrm{dom}(\Delta)$, and the number of distinct subsets of $\mathbb{P}(\theta(\rho))$ are finite, the number of types τ obeying the above restriction is finite. Therefore, the number of recursive invocations of $\mathsf{INH}_{\Gamma,\Delta}$ is finite. ◀

The following Theorem 38 shows that $\mathsf{INH}_{\Gamma,\Delta}(\tau,\emptyset)$ computes Horn clauses which characterize inhabitants (respecting arities in Γ) of type τ.

▶ **Theorem 38** (Correctness)**.** *Let* Γ, Δ *be environments such that each predicate occurring in a parameterized type in* $\mathrm{ran}(\Gamma)$ *is a literal constraint or a term constraint, let* Ξ *be a set of intersection types, let* $\tau \in \Xi$*, and let* $\mathcal{H}$ *be the set* $\mathsf{INH}_{\Gamma,\Delta}(\tau,\emptyset)$ *of Horn clauses over the signature* $\Sigma(\Gamma,\Delta,\Xi)$*. We have* $\Gamma;\Delta \vdash M : \tau$ *iff* $\mathcal{H} \Vdash Q_\tau(\mathrm{tree}(M))$ *for any* $M \in \mathbb{C}(\Gamma,\Delta)$*.*

Proof. W.l.o.g. we assume that distinct bound variables have distinct names and there is a bijection μ between term variables and first-order variables such that in case $\langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \psi$ of Algorithm REC the chosen fresh first-order variable is $\mu(x)$.

For the implication from left to right, we assume $\Gamma;\Delta \vdash M : \tau$ and proceed by induction on M. We have $M = A\,T_1 \ldots T_n$ such that $(A : \varphi) \in \Gamma$, $n \leq \mathrm{ar}(\varphi)$, and there exists a literal substitution θ and a term substitution ξ satisfying properties of Lemma 13.

We have $\mathsf{REC}^A_{\Gamma,\Delta,\tau,\{\tau\}}((),\varphi,(),\emptyset) \subseteq \mathcal{H}$ containing the clause $Q_\tau(A_{(n)}(X_1,\ldots,X_n)) \leftarrow \vec{H}$ such that for $1 \leq i \leq n$ the following properties hold.

- If T_i is a literal, then $Q_t(X_i), (X_i = T_i) \in \vec{H}$, introduced by case $\langle \alpha : t \rangle \Rightarrow \psi$ such that $\theta(\alpha) = T_i$ and $(T_i : t) \in \Delta$. Additionally, $\mathcal{H}$ contains the clause $Q_t(T_i) \leftarrow$.
- If T_i is not a literal, then $Q_\sigma(X_i) \in \vec{H}$ for some type σ such that $\Gamma;\Delta \vdash T_i : \sigma$ by either Lemma 13.1.b or Lemma 13.2.a, and by the induction hypothesis $\mathcal{H} \Vdash Q_\sigma(\mathrm{tree}(T_i))$.
- If a literal constraint P occurs in φ, then $\theta(P)$ holds by Lemma 13.1.c.
- If a term constraint P occurs in φ, then $\mu(\theta(P))$ occurs in $\vec{H}$ and $\xi(\theta(P))$ holds by Lemma 13.1.c.

Using the substitution which maps X_i to $\mathrm{tree}(T_i)$ for $i = 1,\ldots,n$ each antecedent in $\vec{H}$ is true in the considered smallest Herbrand model, and we obtain $\mathcal{H} \Vdash Q_\tau(\mathrm{tree}(A\,T_1 \ldots T_n))$.

For the implication from right to left, we assume $\mathcal{H} \Vdash Q_\tau(\mathrm{tree}(M))$ and proceed by induction on M. We have $M = A\,T_1 \ldots T_n$ and $\mathcal{H}$ contains the clause $Q_\tau(A_{(n)}(X_1,\ldots,X_n)) \leftarrow \vec{H}$, constructed by $\mathsf{REC}^A_{\Gamma,\Delta,\tau,\Xi}$ for some set Ξ. Additionally, for some literal substitution θ and the substitution which maps X_i to $\mathrm{tree}(T_i)$ for $i = 1,\ldots,n$ each antecedent in $\vec{H}$ is true in the considered smallest Herbrand model, and the following properties hold.

- If T_i is a literal, then $(T_i : t) \in \Delta$ for some t. Additionally, $(X_i = T_i) \in \vec{H}$, introduced by case $\langle \alpha : t \rangle \Rightarrow \psi$ such that $\theta(\alpha) = T_i$.
- If T_i is not a literal, then $Q_\sigma(X_i) \in \vec{H}$ for some type σ and $\mathcal{H} \Vdash Q_\sigma(\mathrm{tree}(T_i))$. By the induction hypothesis we have $\Gamma;\Delta \vdash T_i : \sigma$.
- For any literal constraint P occurring in φ we have that $\theta(P)$ holds.
- Let ξ be a term substitution such that $\xi(x) = T_i$ if $\mu(x) = X_i$. For any term constraint P occurring in φ, we have $\mu(\theta(P)) \in \vec{H}$ and $\xi(\theta(P))$ holds.

By Lemma 13 we obtain $\Gamma;\Delta \vdash A\,T_1 \ldots T_n : \tau$. ◀

The following Definition 39 gives the tree language over a signature $\Sigma(\Gamma, \Delta, \Xi)$ for an intersection type $\tau \in \Xi$ described by a set of Horn clauses constructed in Algorithm $\mathsf{INH}_{\Gamma,\Delta}(\tau, \emptyset)$.

▶ **Definition 39.** *Let $\mathcal{H}$ be a set of Horn clauses over the signature $\Sigma(\Gamma, \Delta, \Xi)$ and let $\tau \in \Xi$, we call $\mathcal{L}_{\mathcal{H}}(\tau) = \{\mathrm{tree}(T) \mid T \in \mathbb{C}(\Gamma, \Delta) \cup \mathrm{dom}(\Delta)$ such that $\mathcal{H} \Vdash Q_\tau(\mathrm{tree}(T))\}$ the* tree language *of τ in $\mathcal{H}$.*

We recall the shape of *automata clauses* by Reuß and Seidl in the following Definition 40, for which emptiness of the corresponding tree language is decidable (Theorem 41).

▶ **Definition 40** (Automata Clauses [33, Section 2])**.** *An* automata clause *over the signature $\Sigma(\Gamma, \Delta, \Xi)$ is a Horn clause of the form*

$$Q_0(A_{(n)}(X_1, \ldots, X_n)) \leftarrow Q_1(X_1), \ldots, Q_n(X_n), X_{i_1} = u_1, \ldots, X_{i_k} = u_k, X_{j_1} \neq v_1, \ldots, X_{j_m} \neq v_m$$

where $A_{(n)} \in \Sigma(\Gamma, \Delta, \Xi)$ is an n-ary function symbol, $Q_0, \ldots, Q_n \in \Sigma(\Gamma, \Delta, \Xi)$ are unary predicates, $X_1, \ldots, X_n$ are distinct first-order variables, $u_1, \ldots, u_k, v_1, \ldots, v_m$ are trees over $\Sigma(\Gamma, \Delta, \Xi)$ containing variables from $\{X_1, \ldots, X_n\}$, and $i_1, \ldots, i_k, j_1, \ldots, j_m \in \{1, \ldots, n\}$.

The tree language $\mathcal{L}_{\mathcal{H}}(\tau)$ corresponds to the language of a bottom-up tree automaton with term constraints [33] described by automata clauses $\mathcal{H}$ and having the accepting states $\{Q_\tau\}$. Therefore, emptiness of $\mathcal{L}_{\mathcal{H}}(\tau)$ is decidable.

▶ **Theorem 41** ([33, Theorem 14])**.** *Given a set $\mathcal{H}$ of automata clauses over the signature $\Sigma(\Gamma, \Delta, \Xi)$ and $\tau \in \Xi$, emptiness of the tree language $\mathcal{L}_{\mathcal{H}}(\tau)$ is decidable.*

Finally, we show decidability of intersection type inhabitation with literal and term constraints by reduction to emptiness of bottom-up tree automata with term constraints.

▶ **Theorem 42.** *Intersection type inhabitation with literal and term constraints (Problem 31) is decidable.*

Proof. Due to Theorem 38 and Theorem 41, it suffices to show that the set of Horn clauses $\mathcal{H} = \mathsf{INH}_{\Gamma,\Delta}(\tau, \emptyset)$ over the signature $\Sigma(\Gamma, \Delta, \Xi)$ for some set Ξ of intersection types contains only automata clauses.

Heads of clauses in $\mathcal{H}$ are either $Q_t(l)$ for some $(l : t) \in \Delta$ (constructed in case $\langle \alpha : t \rangle \Rightarrow \psi$) or $Q_\tau(A_{(n)}(\vec{X}, Y_1, \ldots, Y_k))$ where n is the length of the list $(\vec{X}, Y_1, \ldots, Y_k)$ (constructed in the intersection type case), which are both of proper shape. It remains to show that any antecedent in clauses in $\mathcal{H}$ is of proper shape. We consider the individual cases in which antecedents are constructed in Algorithm REC.

Case $\langle \alpha : t \rangle \Rightarrow \psi$: The constructed antecedents are $Q_t(Y)$ and $Y = l$ for some fresh first-order variable Y and $(l : t) \in \Delta$.

Case $\langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \psi$: The constructed antecedent is $Q_\sigma(Y)$ for some fresh first-order variable Y and $\sigma \in \Xi$.

Case $P \Rightarrow \psi$ such that P is closed: No antecedents are constructed.

Case $(X = M) \Rightarrow \psi$ where M may contain free variables: The constructed antecedent is $(X = \mathrm{tree}(M))$. Since Γ contains only closed parameterized types, any literal variable in M is substituted by some literal in $\mathrm{dom}(\Delta)$ and any term variable in M is substituted by some first-order variable. Therefore, $(X = \mathrm{tree}(M))$ is of proper shape.

Case $(X \neq M) \Rightarrow \psi$ where M may contain free variables: The constructed antecedent is $(X \neq \mathrm{tree}(M))$, which analogously to the above case is of proper shape.

Case ρ: The constructed antecedents are $Q_{\sigma_1}(Y_1), \ldots, Q_{\sigma_k}(Y_k)$ for some fresh first-order variables $Y_1, \ldots, Y_k$ and $\sigma_1, \ldots, \sigma_k \in \Xi$.

Finally, we need to ensure that each first-order variable Z occurring in the head of the constructed clause $Q_\tau(A_{(n)}(\vec{X}, Y_1, \dots, Y_k)) \leftarrow \vec{H}, Q_{\sigma_1}(Y_1), \dots, Q_{\sigma_k}(Y_k)$ in the last case occurs in exactly one antecedent $Q_\sigma(Z)$ for some $\sigma \in \Xi$ or $Q_t(Z)$ for some $t \in \mathrm{ran}(\Delta)$. This trivially holds for the above fresh first-order variables $Y_1, \dots, Y_k$. The remaining first-order variables $Y \in \vec{X}$ are introduced in case $\langle \alpha : t \rangle \Rightarrow \psi$ (with the corresponding antecedent $Q_t(Y)$) and in case $\langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \psi$ (with the corresponding antecedent $Q_\sigma(Y)$). ◀

Concluding the presentation of the decidable fragment of FCLP, we give remarks on its complexity bounds (Remark 43), extensions (Remark 44), and alternatives (Remark 45).

▶ Remark 43. Complexity bounds for emptiness of bottom-up tree automata with term constraints are not known [33, Section 6]. Therefore, we cannot give complexity bounds for intersection type inhabitation with literal and term constraints. Additionally, we do not impose complexity bounds on predicate evaluation (besides decidability).

▶ Remark 44. The class of bottom-up tree automata with term constraints is closed under Boolean operations [33, Proposition 6]. Therefore, existing techniques extending FCL$(\cap, \leq)$ by a Boolean query language [20] are applicable.

▶ Remark 45. There are other classes of constrained tree automata [26] which could be used to obtain a decidable fragment of FCLP. One example are generalized encompassment automata [11, Definition 1], which are more expressive than bottom-up tree automata with term constraints. However, the presentation of such automata as sets of Horn clauses resulting from an inhabitation algorithm appears challenging.

4 Implementation

In practical applications one is rarely interested in solving just the inhabitation *decision problem*, but rather in computing one, several, or all inhabitants. Therefore, we consider the following *synthesis problem*, which is a slightly modified version of Problem 17.

▶ **Problem 46** (Synthesis). Given a type environment Γ, a literal environment Δ, and an intersection type τ, enumerate combinatory terms M for which $\Gamma; \Delta \vdash M : \tau$ holds.

Furthermore, when solving particular problems via synthesis, we want to *interpret* the resulting combinatory terms as solutions for those problems.

A framework [21] addressing Problem 46 was implemented on the basis of the existing Combinatory Logic Synthesizer (CLS)[4] using the Python programming language.

In this section we discuss the implementation, and evaluate how the added features of FCLP help modeling, and improve performance compared to FCL$(\cap, \leq)$, using maze solving (Problem 52) as a benchmark example.

▶ Remark 47. The presented framework does not make use of Python's built-in type system for synthesis, and implements types as Python classes.

4.1 Usage

The framework is implemented as a Python library, requiring Python version 3.10 or later. It does not rely on any additional libraries. For the sake of brevity, an embedded domain-specific language (eDSL), shown in Figure 1, was created for writing parameterized types.

[4] `https://github.com/cls-python/cls-python`

Constructor	Python
ω	`Omega()`
$\sigma \to \tau$	σ `**` τ
$\sigma \cap \tau$	σ `&` τ
$c(\sigma)$	`'c'@`σ
Literal $l : t$	`Literal(l,t)`
Variable α	`LVar('`α`')`

Constructor	Python
$\langle x : t \rangle$	`Use('x', 't')`
$\langle\!\langle \alpha : \tau \rangle\!\rangle$	`Use('`α`',` τ`)`
Predicate P using variables $v_0, \dots, v_n$	`With(lambda` $\mathtt{v}_0, \dots, \mathtt{v}_n$`:`$P$`)`
$\dots \Rightarrow \tau$	`DSL(). ... .In(`τ`)`

Figure 1 Embedded DSL for parameterized and intersection types in Python.

▶ **Example 48** (eDSL). The parameterized type $\langle \alpha : \mathsf{int} \rangle \Rightarrow \langle\!\langle x : \sigma \rangle\!\rangle \Rightarrow \langle\!\langle y : \sigma \rangle\!\rangle \Rightarrow (x = y) \Rightarrow \langle \beta : \mathsf{int} \rangle \Rightarrow (\beta = \alpha + 1) \Rightarrow c(\alpha) \to c(\beta) \to (c(4) \cap c(\omega))$ corresponds to the following eDSL term:

```
DSL().Use('α', 'int').Use('x', σ).Use('y', σ)
  .With(lambda x y: x == y).Use(β, 'int').With(lambda α β: β = α + 1)
  .In(('c'@LVar(α)) ** ('c'@LVar(β)) **
    ('c'@Literal(4, 'int') & 'c'@(Omega())))
```

▶ Remark 49. The operator `**` was chosen to represent the arrow type constructor, since it is the only right associative operator available in Python.

In order to synthesize inhabitants, we need to define a type environment, a literal environment, and an intersection type as a *query*. A type environment is a `dict`, mapping combinators to their types, where combinators can be any `Hashable` Python `object`. Types can be formed via the eDSL or by instantiating appropriate subclasses of the class `Type`. A literal environment is a `dict`, mapping collection identifiers (represented as `str`) to literals, which can be any Python `objects`. The three main operations of the framework are:

- `FiniteCombinatoryLogic(...).inhabit(...)` to initialize the synthesis procedure and compute an intermediate result representation,
- `enumerate_terms` to extract combinatory terms from the intermediate representation,
- `interpret_term` to interpret a combinatory term as a solution in the problem domain.

Given a type environment Γ, literal environment Δ, and a type τ, we can use the above operations to enumerate elements of the set $\{M \mid \Gamma, \Delta \vdash M : \tau\}$ by the following steps. First, we generate an intermediate representation of the synthesis results.

```
results = FiniteCombinatoryLogic(repository = Γ, literals = Δ).inhabit(τ)
```

Second, we enumerate up to n distinct terms.

```
terms = enumerate_terms(τ, results, n)
```

A term is represented as a tuple, such that its first projection is the associated combinator, and the following projections are representations of the arguments. Finally, we interpret these terms to obtain solutions in the problem domain.

```
solutions = [interpret_term(term) for term in terms]
```

Each combinator can be equipped with a computational component, realized by implementing the `Callable` protocol. In this step each callable combinator is interpreted by calling it on its interpreted arguments.

4.2 Synthesis Procedure

In contrast to Algorithm INH (Definition 34), the implemented synthesis procedure is not limited to the decidable fragment. While this makes inhabitation undecidable, in most practical applications the advantages of unrestricted predicates outweigh potential lock-ups.

Given a type environment Γ, literal environment Δ, and intersection type τ, the synthesis process is structured as follows.

Preprocessing: For each typed combinator $(C : e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \rho) \in \Gamma$, we first generate the set of substitutions determined by Δ and the literal quantifiers in $e_1, \ldots, e_m$. Next, we use these substitutions to evaluate all literal constraints in $e_1, \ldots, e_m$, discarding substitutions which violate at least one constraint. The remaining substitutions are stored alongside the combinator. Afterward, we remove all literal quantifiers and literal constraints from $e_1, \ldots, e_m$. Finally, for each arity k up to $\mathrm{ar}(\rho)$, we decompose ρ into possible pairs of k *argument* types and a *return* type.

Generating Horn clauses: For each typed combinator $(C : e_1 \Rightarrow \cdots \Rightarrow e_m \Rightarrow \rho) \in \Gamma$, each literal substitution θ for C, and each arity k, if the intersection of the return types is a subtype of τ, a Horn clause is created analogous to the last case of Algorithm INH. Further Horn clauses are generated by recursion on each argument type, as well as on each type occurring in term quantifiers in $e_1, \ldots, e_m$.

Enumeration: Given the above set of Horn clauses and a number n, we enumerate up to n inhabitants in a *bottom-up* manner. In this step we resolve term quantifiers by enumerating inhabitants of the type quantified over, placing them at the respective argument position in a given combinator and substituting the respective term variables in the remaining constraints by those inhabitants. If this leads to violated constraints, we discard those terms. Similarly, literals are placed at positions corresponding to their quantifier position.

4.3 Solutions in a Maze

The additions of FCLP compared to FCL$(\cap, \leq)$ improve upon the expressiveness of specification, and it was observed that more concise modeling can lead to performance improvements. Consider the following example of finding solutions in a maze.

▶ **Definition 50** (Maze). *Let $n \in \mathbb{N}$, an $n \times n$-*maze *is a function $\mathcal{M} : \{0, \ldots, n-1\}^2 \to \mathbb{B}$ indicating whether a position is* free *or* blocked.

▶ **Definition 51** (Maze Solution). *A* solution *to an $n \times n$-maze $\mathcal{M}$ is a finite sequence $\big((x_0, y_0), \ldots, (x_l, y_l)\big)$ such that:*

- $(x_0, y_0) = (0, 0)$ *and* $(x_l, y_l) = (n-1, n-1)$,
- *for each $i \in \{0, \ldots, l\}$ we have $(x_i, y_i) \in \mathrm{dom}(\mathcal{M})$ and $\mathcal{M}(x_i, y_i) =$ true,*
- *for each $i \in \{0, \ldots, l-1\}$ we have $|x_i - x_{i+1}| + |y_i - y_{i+1}| = 1$.*

▶ **Problem 52** (Maze Solving). Given an $n \times n$-maze $\mathcal{M}$, enumerate solutions to $\mathcal{M}$.

Variants of maze solving are common, miniature benchmark examples[5] for component-oriented synthesis [9, 4, 19]. Domain-specific components for maze solving encompass movement directions and the maze layout. Specification capabilities, scalability, and performance of the framework in the case of maze solving translate well to software product line design [25], factory planning [37], and cyber-physical system design [14].

[5] Of course, if one is solely interested in maze solving, a domain-specific algorithm using dynamic programming is recommended instead of domain-agnostic component-oriented synthesis.

Let us explore an approach to maze solving in FCL($\cap, \leq$). Given an $n \times n$-maze $\mathcal{M}$ we construct the following type environment $\Gamma^{\mathcal{M}}_{\text{FCL}}$.

$$\begin{aligned}
\Gamma^{\mathcal{M}}_{\text{FCL}} = \{&\textsc{Free}_{x,y} : \text{isfree}(x,y) \mid (x,y) \in \text{dom}(\mathcal{M}) \text{ such that } \mathcal{M}(x,y) = \textit{true}\} \cup \\
&\{\textsc{Start} : \text{pos}(0(\omega), 0(\omega)), \\
&\quad \textsc{Up} : \bigcap_{(x,y)\in\text{dom}(\mathcal{M})} (\text{isfree}(x,y) \to \text{pos}(x,y+1) \to \text{pos}(x,y)), \\
&\quad \textsc{Down} : \bigcap_{(x,y)\in\text{dom}(\mathcal{M})} (\text{isfree}(x,y) \to \text{pos}(x,y-1) \to \text{pos}(x,y)), \\
&\quad \textsc{Left} : \bigcap_{(x,y)\in\text{dom}(\mathcal{M})} (\text{isfree}(x,y) \to \text{pos}(x+1,y) \to \text{pos}(x,y)), \\
&\quad \textsc{Right} : \bigcap_{(x,y)\in\text{dom}(\mathcal{M})} (\text{isfree}(x,y) \to \text{pos}(x-1,y) \to \text{pos}(x,y))\}
\end{aligned}$$

For better legibility, we allow for binary constructors as described in Remark 3. Combinators $\textsc{Free}_{x,y}$ denote witnesses that the space at coordinates x and y is free. Combinators $\textsc{Up}$, $\textsc{Down}$, $\textsc{Left}$ and $\textsc{Right}$ denote movement in the corresponding direction. Since FCL($\cap, \leq$) does not allow for literals, numbers need to be encoded by constructors $(0(\omega), 1(\omega), \ldots, n(\omega))$ and the position shifts need to be computed beforehand.

▶ **Example 53.** Consider a 2×2 maze $\mathcal{M}$ in which exactly the position $(1, 0)$ is blocked. In order to synthesize (not necessarily loop-free) solutions in $\mathcal{M}$, we enumerate elements of the set $\{M \mid \Gamma^{\mathcal{M}}_{\text{FCL}} \vdash M : \text{pos}(1(\omega), 1(\omega))\}$, resulting in combinatory terms such as: $\big(\textsc{Right}\ \textsc{Free}_{1,1}\ (\textsc{Down}\ \textsc{Free}_{0,1}\ \textsc{Start})\big)$. Given the appropriate interpretation for the movement combinators, we can interpret the term as the solution $\big((0,0),(0,1),(1,1)\big)$, shown in the below Figure 2.

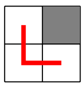

Figure 2 A 2×2 maze with a solution in red. Position $(0, 0)$ is in the top-left corner.

While the above shows that FCL($\cap, \leq$) can model Problem 52, we identify three improvements that can be made using FCLP.

- Quantifiers can be used to avoid intersections spanning all positions in the maze.
- Since we expect each combinator to be fully applied (have exactly as many arguments as the arity of its type), we model positions as parameters.
- The combinators $\textsc{Free}_{x,y}$ act as predicates for the movement combinators. Using FCLP, we can use predicates directly, removing those combinators.

Applying the above improvements, we construct the type environment $\Gamma^{\mathcal{M}}_{\text{FCLP}}$ together with the literal environment $\Delta = \{0 : \text{int}, \ldots, n-1 : \text{int}\}$.

$$\begin{aligned}
&\Gamma^{\mathcal{M}}_{\text{FCLP}} = \{\textsc{Start} : \text{pos}(0,0), \\
&\textsc{Up} : \langle\alpha : \text{int}\rangle \Rightarrow \langle\beta : \text{int}\rangle \Rightarrow \langle\gamma : \text{int}\rangle \Rightarrow (\gamma = \beta{+}1) \Rightarrow \mathcal{M}(\alpha,\beta) \Rightarrow \langle\!\langle \text{p} : \text{pos}(\alpha,\gamma)\rangle\!\rangle \Rightarrow \text{pos}(\alpha,\beta), \\
&\textsc{Down} : \langle\alpha : \text{int}\rangle \Rightarrow \langle\beta : \text{int}\rangle \Rightarrow \langle\gamma : \text{int}\rangle \Rightarrow (\gamma = \beta{-}1) \Rightarrow \mathcal{M}(\alpha,\beta) \Rightarrow \langle\!\langle \text{p} : \text{pos}(\alpha,\gamma)\rangle\!\rangle \Rightarrow \text{pos}(\alpha,\beta), \\
&\textsc{Left} : \langle\alpha : \text{int}\rangle \Rightarrow \langle\beta : \text{int}\rangle \Rightarrow \langle\gamma : \text{int}\rangle \Rightarrow (\gamma = \alpha{+}1) \Rightarrow \mathcal{M}(\alpha,\beta) \Rightarrow \langle\!\langle \text{p} : \text{pos}(\gamma,\beta)\rangle\!\rangle \Rightarrow \text{pos}(\alpha,\beta), \\
&\textsc{Right} : \langle\alpha : \text{int}\rangle \Rightarrow \langle\beta : \text{int}\rangle \Rightarrow \langle\gamma : \text{int}\rangle \Rightarrow (\gamma = \alpha{-}1) \Rightarrow \mathcal{M}(\alpha,\beta) \Rightarrow \langle\!\langle \text{p} : \text{pos}(\gamma,\beta)\rangle\!\rangle \Rightarrow \text{pos}(\alpha,\beta)\}
\end{aligned}$$

Clearly, the above type environment $\Gamma^{\mathcal{M}}_{\text{FCLP}}$ is more concise compared to the type environment $\Gamma^{\mathcal{M}}_{\text{FCL}}$. Additionally, we do not need to manually compute the position shifts to construct $\Gamma^{\mathcal{M}}_{\text{FCLP}}$ beforehand, as was needed for $\Gamma^{\mathcal{M}}_{\text{FCL}}$.

Performance Evaluation

Using the modeling techniques introduced with FCLP, we observe improved performance. Figure 3 shows synthesis execution time to find all solutions using different type environments, up to a maze size of 70×70. All benchmarks were performed on the same machine[6] using the implementation at hand. Environments $\Gamma^{\mathcal{M}}_{\mathrm{FCLP(lit)}}$, $\Gamma^{\mathcal{M}}_{\mathrm{FCLP(pos)}}$, $\Gamma^{\mathcal{M}}_{\mathrm{FCLP(pred)}}$ each correspond to an improvement identified above, namely using literals for coordinates, using term quantifiers for the position, and using a predicate for free positions respectively.

Size	$\Gamma^{\mathcal{M}}_{\mathrm{FCL}}$	$\Gamma^{\mathcal{M}}_{\mathrm{FCLP(lit)}}$	$\Gamma^{\mathcal{M}}_{\mathrm{FCLP(pos)}}$	$\Gamma^{\mathcal{M}}_{\mathrm{FCLP(pred)}}$	$\Gamma^{\mathcal{M}}_{\mathrm{FCLP}}$
10×10	$1.3\,s$	$0.5\,s$	$0.3\,s$	$0.1\,s$	$0.1\,s$
20×20	$21.9\,s$	$8.0\,s$	$6.4\,s$	$2.4\,s$	$1.9\,s$
30×30	$125.2\,s$	$41.0\,s$	$30.7\,s$	$12.8\,s$	$9.8\,s$
40×40	$464.7\,s$	$130.2\,s$	$97.9\,s$	$42.3\,s$	$32.4\,s$
50×50	$1279.8\,s$	$322.2\,s$	$239.5\,s$	$103.2\,s$	$78.5\,s$
60×60	$3038.5\,s$	$645.4\,s$	$486.3\,s$	$214.2\,s$	$160.2\,s$
70×70	> 5000	$1195.6\,s$	$893.5\,s$	$384.9\,s$	$299.4\,s$

Figure 3 Benchmarks for different maze sizes and different type environments.

While the data shows that all approaches scale at an exponential rate given the size, using $\Gamma^{\mathcal{M}}_{\mathrm{FCLP}}$ leads to a performance increase of one order of magnitude compared to $\Gamma^{\mathcal{M}}_{\mathrm{FCL}}$, with each modeling technique contributing to the speed-up. The performance increase of $\Gamma^{\mathcal{M}}_{\mathrm{FCLP}}$ compared to $\Gamma^{\mathcal{M}}_{\mathrm{FCL}}$ can be attributed to the following three factors:

1. $\Gamma^{\mathcal{M}}_{\mathrm{FCLP(lit)}}$ reduces the size of the type of each movement combinator.
2. $\Gamma^{\mathcal{M}}_{\mathrm{FCLP(pos)}}$ induces fewer subtype checks due to restricted term shape.
3. $\Gamma^{\mathcal{M}}_{\mathrm{FCLP(pred)}}$ reduces the number of combinators.

A benchmark using $\Gamma^{\mathcal{M}}_{\mathrm{FCL}}$ and the latest version of the prior implementation of CLS in Python was conducted, leading to a time of $276\,s$ for a 10×10 maze. The difference to the prior implementation stems from the fact that the implementation at hand focuses on performance, while the previous focuses on formal verification [4].

For performance evaluation, frameworks based on Bounded Combinatory Logic [18] or based on Finite Combinatory Logic with Boolean Queries [20] are of no consequence. Neither bounded polymorphism nor Boolean connectives are suited to model maze solving. Therefore, the resulting performance is close to the prior implementation of CLS.

Loop-free solutions

As observed above, utilizing quantifiers and predicates can lead to significant speed-ups in certain use-cases. Interestingly, predicates also model specifications, for which an effective model in $\mathrm{FCL}(\cap, \leq)$ is unclear.

Consider the maze presented in Figure 4 and solutions, that do not visit any position more than once (*loop-free*). In each movement combinator a predicate can require each visited position to be unique in a given term, thereby only allowing for loop-free solutions. During enumeration, terms containing at least one loop are discarded and the procedure will return

[6] AMD Ryzen 7 5800X (3.8 GHz), 16 GB DDR4 RAM

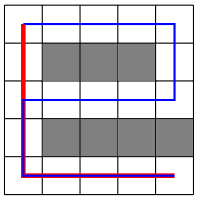

Figure 4 A 5×5 maze with exactly two loop-free solutions (in red and blue).

exactly two solutions, after which it halts. In contrast, for FCL($\cap, \leq$) we need to rely on a generate-and-test approach. Such an approach would enumerate infinitely many candidates (including those, which contain arbitrary many loops) and filter out those containing loops. In particular, the procedure as a whole would search indefinitely for a third loop-free solution. Furthermore, adjusting the size of the maze in Figure 4, there are arbitrary many looping solutions whose length lie between the two loop-free solutions. The FCLP approach discards solutions containing at least one loop early, and thereby never considers solutions with multiple loops. In comparison, the FCL($\cap, \leq$) approach has no such mechanism, leading to exponentially more candidates to be checked and therefor an arbitrary long time between the two loop-free solutions.

It is possible to model loop-free solutions in the synthesis framework based on Finite Combinatory Logic with Boolean Queries [20]. In particular, negation is suitable to express that a position is not yet visited. However, a performance evaluation has shown that such an approach is infeasible for mazes beyond size 5×5.

5 Case Study: Robotic Arms

We evaluate practical applicability of FCLP by means of a case study in which robotic arms are synthesized from a set of 28 modular components [15]. The individual components are modeled as typed combinators such that inhabitants of specific types can be interpreted as assembly instructions for robotic arms. The assembly instructions are executed in CAD software, assembling 3D models of robotic arms. Analysis tools, which are part of the CAD software, confirm that the assembled robotic arms are mechanically sound, individual components do not interfere which each other, and mechanical joints kinematically work as intended. These properties hold for all robotic arms up to six degrees of freedom (six moving joints) synthesized in the case study, a total of 364 arms containing on average 140 components each. Due to the chain-like nature of robotic arms we argue that higher degrees of freedom also exhibit these properties.

In previous work [14] conducting the same case study utilizing FCL($\cap, \leq$), numerical constraints necessitate an exponential number of combinators. There are a number of common numerical constraints of high importance, such as the degrees of freedom, the total drawn current, total weight of the assembly, or the torque of motors. In previous work such constraints are modeled as families of the following typed combinators, which specify individual cases.

$$\begin{aligned} C_{i_1,i_2,i_3,i_4} : \ & \mathrm{Assembly}(i_1(\omega)) \to \mathrm{Assembly}(i_2(\omega)) \to \mathrm{Assembly}(i_3(\omega)) \to \mathrm{Assembly}(i_4(\omega)) \\ & \text{such that } 0, ..., n \text{ are unary type constructors,} \\ & i_1, i_2, i_3, i_4 \in \{0, ..., n\}, \\ & \text{and } i_4 = i_1 + i_2 + i_3 + k \end{aligned}$$

Each of the above typed combinators C_{i_i,i_2,i_3,i_4} refers to individual numbers i_i, i_2, i_3 of specific parts in each connected assembly, and the accumulated number i_4 of specific parts increased by a constant k. The number of such combinators is exponential in the number of connected assemblies (the arity of the type) multiplied by the number of distinct constraints of interest. The number of connected assemblies depends on the granularity of the model, but even for a case study of this scale a typical number is five, and requests usually employ at least three constraints. The value range n for the constraints is usually no more than ten. This leads to repositories of enormous size, containing tens of thousands of combinators for typical requests. In practice, this large number of essentially redundant combinators impairs debugging and deteriorates performance of the inhabitation algorithm.

In the later case study [15] the described issues are tackled using FCLP. The above family of typed combinators is condensed to the following single typed combinator:

$$\begin{aligned}&C : \langle\alpha_1 : \text{int}\rangle \Rightarrow \langle\alpha_2 : \text{int}\rangle \Rightarrow \langle\alpha_3 : \text{int}\rangle \Rightarrow \langle\alpha_4 : \text{int}\rangle \Rightarrow (\alpha_4 = \alpha_1 + \alpha_2 + \alpha_3 + k) \Rightarrow\\ &\quad \langle\!\langle x_1 : \text{Assembly}(\alpha_1)\rangle\!\rangle \Rightarrow \langle\!\langle x_2 : \text{Assembly}(\alpha_2)\rangle\!\rangle \Rightarrow \langle\!\langle x_3 : \text{Assembly}(\alpha_3)\rangle\!\rangle \Rightarrow \text{Assembly}(\alpha_4)\\ &\text{with } \Delta \supseteq \{0 : \text{int}, \ldots, n : \text{int}\}\end{aligned}$$

The above combinator C concisely expresses the described numeric constraint for the particular assembly. Literal variables $\alpha_1, \alpha_2, \alpha_3$ refer to individual numbers of specific parts in each connected assembly. The literal variable α_4 refers to the accumulated number of specific parts increased by a constant k, which is described by the literal constraint $\alpha_4 = \alpha_1 + \alpha_2 + \alpha_3 + k$. As a result, such combinators closely represent the individual component in the actual use-case [15]. The number of required combinators per component is constant, and independent from specified constraints. Since each predicate is a literal constraint (Definition 30), the corresponding inhabitation problem is decidable (Theorem 42).

The ability to cleanly handle constraints allows leveraging combinatory logic synthesis to explore the robotic arm design space efficiently and glean information about it.

6 Conclusion

The present work conservatively extends the type system FCL($\cap, \leq$) [32] by literals, quantifiers, and predicates. While the inhabitation problem in the resulting type system FCLP (Definition 8) is undecidable (Theorem 20), we give an expressive fragment of FCLP for which inhabitation is decidable (Theorem 42). The particular fragment is based on results for tree automata with term constraints by Reuß and Seidl [33], and allows for specification of certain local (dis)equality constraints (Definition 29) for subterms of inhabitants. The main contribution of the present work is a terminating algorithm INH which given a type environment, a literal environment, and an intersection type computes a logic program (set of Horn clauses) which represents all inhabitants.

For empirical evaluation, an algorithm for inhabitant enumeration (Problem 46) is implemented in the programming language Python. The implementation, as part of a larger synthesis framework [21], is shown superior to an existing FCL($\cap, \leq$)-based framework CLS [4] with respect to specification capabilities, scalability, and performance. Finally, practical applicability is demonstrated via a case study in the area of cyber-physical systems.

There are several directions for further research.

First, it is worth investigating type inhabitation for more expressive type languages in the setting of combinatory logic. Polymorphic set-theoretic types [12, 13] constitute a promising candidate for type-based component-oriented program synthesis.

Second, there is room for exploration of more expressive fragments of FCLP with decidable type inhabitation. A promising candidate could be obtained by using generalized encompassment automata [11, Definition 1] instead of (the less expressive) bottom-up tree automata with term constraints. Another candidate could rely on automata with disequality constraints [16, Definition 1]. Such automata are used by Czajka et.al. [5] to externally restrict sets of inhabitants via term rewriting systems. It appears appealing to internalize such restrictions as part of the specification language.

Third, the present work focuses on inhabitants which respect arities in the given type environment. This restriction is based on the observation that in practice not every domain-specific component is a function. However, the subtyping rule $\omega \leq \omega \to \omega$ is in conflict with this observation. It is intriguing to explore semantics of combinatory logic with intersection types [17] without the subtyping rule $\omega \leq \omega \to \omega$.

Fourth, satisfiability of literal constraints, such as $\beta = \alpha + 1$, could be addressed in algorithm INH by a principled approach, for example based on SMT. Besides potential performance improvements, such an approach may allow for a countably infinite parameter space (literal environment).

Fifth, efficient enumeration procedures [35, 22] for tree languages focus on regular structures. Besides the naive generate-and-test approach in the present work, there is no practical enumeration procedure for trees accepted by bottom-up tree automata with term constraints. It is unclear whether methods known from logic programming, such as sideways information passing [3], are applicable.

References

1 Henk Barendregt. Introduction to generalized type systems. *J. Funct. Program.*, 1(2):125–154, 1991. `doi:10.1017/S0956796800020025`.

2 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983. URL: `http://www.jstor.org/stable/2273659`.

3 Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *J. Log. Program.*, 10(3&4):255–299, 1991. `doi:10.1016/0743-1066(91)90038-Q`.

4 Jan Bessai. *A type-theoretic framework for software component synthesis*. PhD thesis, TU Dortmund University, 2019.

5 Jan Bessai, Lukasz Czajka, Felix Laarmann, and Jakob Rehof. Restricting tree grammars with term rewriting. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*, volume 228 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.FSCD.2022.14`.

6 Jan Bessai, Boris Düdder, George Heineman, and Jakob Rehof. Combinatory synthesis of classes using feature grammars. In *Formal Aspects of Component Software: 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers 12*, pages 123–140. Springer, 2016.

7 Jan Bessai, Andrej Dudenhefner, Boris Düdder, Tzu-Chun Chen, Ugo de'Liguoro, and Jakob Rehof. Mixin composition synthesis based on intersection types. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 76–91, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.TLCA.2015.76`.

8 Jan Bessai, Andrej Dudenhefner, Boris Düdder, Moritz Martens, and Jakob Rehof. Combinatory Logic Synthesizer. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part I*, volume 8802 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2014. `doi:10.1007/978-3-662-45234-9_3`.

9. Jan Bessai and Anna Vasileva. User support for the combinator logic synthesizer framework. In Paolo Masci, Rosemary Monahan, and Virgile Prevosto, editors, *Proceedings 4th Workshop on Formal Integrated Development Environment, F-IDE@FLoC 2018, Oxford, England, 14 July 2018*, volume 284 of *EPTCS*, pages 16–25, 2018. `doi:10.4204/EPTCS.284.2`.
10. Bruno Bogaert, Franck Seynhaeve, and Sophie Tison. The recognizability problem for tree automata with comparisons between brothers. In Wolfgang Thomas, editor, *Foundations of Software Science and Computation Structure, Second International Conference, FoSSaCS'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, volume 1578 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 1999. `doi:10.1007/3-540-49019-1_11`.
11. Anne-Cécile Caron, Hubert Comon, Jean-Luc Coquidé, Max Dauchet, and Florent Jacquemard. Pumping, cleaning and symbolic constraints solving. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, volume 820 of *Lecture Notes in Computer Science*, pages 436–449. Springer, 1994. `doi:10.1007/3-540-58201-0_88`.
12. Giuseppe Castagna. Programming with union, intersection, and negation types. *CoRR*, abs/2111.03354, 2021. `arXiv:2111.03354`.
13. Giuseppe Castagna, Mickaël Laurent, and Kim Nguyen. Polymorphic type inference for dynamic languages. *Proc. ACM Program. Lang.*, 8(POPL):1179–1210, 2024. `doi:10.1145/3632882`.
14. Constantin Chaumet and Jakob Rehof. CLS-CAD: Synthesizing CAD Assemblies in Fusion 360, 2023. `arXiv:2311.18492`.
15. Constantin Chaumet, Jakob Rehof, and Thomas Schuster. A knowledge-driven framework for synthesizing designs from modular components, 2023. `arXiv:2311.18533`.
16. Hubert Comon and Florent Jacquemard. Ground reducibility is EXPTIME-complete. *Inf. Comput.*, 187(1):123–153, 2003. `doi:10.1016/S0890-5401(03)00134-2`.
17. Mariangiola Dezani-Ciancaglini and J. Roger Hindley. Intersection types for combinatory logic. *Theor. Comput. Sci.*, 100(2):303–324, 1992. `doi:10.1016/0304-3975(92)90306-Z`.
18. Boris Düdder, Moritz Martens, Jakob Rehof, and Paweł Urzyczyn. Bounded combinatory logic. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 243–258. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. `doi:10.4230/LIPICS.CSL.2012.243`.
19. Andrej Dudenhefner. *Algorithmic aspects of type-based program synthesis.* PhD thesis, TU Dortmund University, 2019.
20. Andrej Dudenhefner, Felix Laarmann, Jakob Rehof, and Christoph Stahl. Finite combinatory logic extended by a Boolean query language for composition synthesis. In *29th International Conference on Types for Proofs and Programs TYPES 2023–Abstracts*, page 105, 2023.
21. Andrej Dudenhefner, Christoph Stahl, and Jan Bessai. Combinatory Logic Synthesizer with Predicates. URL: `https://github.com/tudo-seal/clsp-python`.
22. Jonas Duregård, Patrik Jansson, and Meng Wang. Feat: functional enumeration of algebraic types. In Janis Voigtländer, editor, *Proceedings of the 5th ACM SIGPLAN Symposium on Haskell, Haskell 2012, Copenhagen, Denmark, 13 September 2012*, pages 61–72. ACM, 2012. `doi:10.1145/2364506.2364515`.
23. Thom W. Frühwirth, Ehud Shapiro, Moshe Y. Vardi, and Eyal Yardeni. Logic programs as types for logic programs. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 300–309. IEEE Computer Society, 1991. `doi:10.1109/LICS.1991.151654`.
24. George Heineman, Jan Bessai, Boris Düdder, and Jakob Rehof. 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 7*, pages 303–317. Springer, 2016.

25 George Heineman, Armend Hoxha, Boris Düdder, and Jakob Rehof. Towards migrating object-oriented frameworks to enable synthesis of product line members. In *Proceedings of the 19th International Conference on Software Product Line*, pages 56–60, 2015.

26 Florent Jacquemard. *Extended Tree Automata Models for the Verification of Infinite State Systems*. PhD thesis, École normale supérieure de Cachan-ENS Cachan, 2011.

27 Fadil Kallat, Tristan Schäfer, and Anna Vasileva. CLS-SMT: bringing together combinatory logic synthesis and satisfiability modulo theories. In Giselle Reis and Haniel Barbosa, editors, *Proceedings Sixth Workshop on Proof eXchange for Theorem Proving, PxTP 2019, Natal, Brazil, August 26, 2019*, volume 301 of *EPTCS*, pages 51–65, 2019. doi:10.4204/EPTCS.301.7.

28 Alexander Mages, Carina Mieth, Jens Hetzler, Fadil Kallat, Jakob Rehof, Christian Riest, and Tristan Schäfer. Automatic component-based synthesis of user-configured manufacturing simulation models. In *2022 Winter Simulation Conference (WSC)*, pages 1841–1852. IEEE, 2022.

29 Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. In David Grove and Stephen M. Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 619–630. ACM, 2015. doi:10.1145/2737924.2738007.

30 Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. In Chandra Krintz and Emery D. Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 522–538. ACM, 2016. doi:10.1145/2908080.2908093.

31 Jakob Rehof. Towards combinatory logic synthesis. In *1st International Workshop on Behavioural Types, BEAT*, 2013.

32 Jakob Rehof and Paweł Urzyczyn. Finite combinatory logic with intersection types. In C.-H. Luke Ong, editor, *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2011. doi:10.1007/978-3-642-21691-6_15.

33 Andreas Reuß and Helmut Seidl. Bottom-up tree automata with term constraints. In Christian G. Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*, volume 6397 of *Lecture Notes in Computer Science*, pages 581–593. Springer, 2010. doi:10.1007/978-3-642-16242-8_41.

34 Tristan Schäfer, Jim Bergmann, Rafael Garcia Carballo, Jakob Rehof, and Petra Wiederkehr. A synthesis-based tool path planning approach for machining operations. *Procedia CIRP*, 104:918–923, 2021.

35 Cas van der Rest and Wouter Swierstra. A completely unique account of enumeration. *Proc. ACM Program. Lang.*, 6(ICFP):411–437, 2022. doi:10.1145/3547636.

36 Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976. doi:10.1145/321978.321991.

37 Jan Winkels, Julian Graefenstein, Tristan Schäfer, David Scholz, Jakob Rehof, and Michael Henke. Automatic composition of rough solution possibilities in the target planning of factory planning projects by means of combinatory logic. In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV 8*, pages 487–503. Springer, 2018.