

Formal Confluence for φ -Calculus in Lean 4


Short paper

Anatoliy Baskakov 

Lab of Programming Languages and Compilers, Innopolis University, Tatarstan Republic, Russia

Nikolai Kudasov 

Lab of Programming Languages and Compilers, Innopolis University, Tatarstan Republic, Russia

Violetta Sim 

Innopolis University, Tatarstan Republic, Russia

Abstract

Elegant Objects is a minimalistic programming paradigm that builds on the Decorator Pattern. EO is an experimental programming language implementing the paradigm in its pure form with φ -calculus acting as its formal model. In this paper, we use formalize the minimal φ -calculus using a computer proof assistant Lean 4, building up of the work of Kudasov and Sim. Our formalization focuses on the properties of φ -calculus as a rewriting system, but also provides a library of definitions and lemmas that we believe are useful for future development of the metatheory of φ -calculus. Our main result is the Lean 4 formalization of the proof of confluence for the minimal φ -calculus.

2012 ACM Subject Classification Theory of computation \rightarrow Computability; Theory of computation \rightarrow Operational semantics; Theory of computation \rightarrow Rewrite systems

Keywords and phrases proof assistant, abstract rewriting, confluence, φ -calculus, elegant objects

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements The research was supported by Huawei Technologies Co., Ltd. We thank Yegor Bugayenko, Maxim Trunnikov, and Vladimir Zakharov for fruitful technical discussions.

1 Introduction

Elegant Objects¹ is a minimalistic programming paradigm that builds on the Decorator pattern [15, §4]. Bugayenko's EOLANG² [4] is an experimental programming language implementing the paradigm in its pure form with φ -calculus acting as its formal model. Kudasov and Sim outline a minimal version of φ -calculus [18], complete syntax of which is presented in Figure 1. They also prove confluence and compare it to some alternative formal systems, such as Wand's λ -calculus with records [31].

In this paper, we develop basic metatheory for Abstract Rewriting Systems (ARS) in Lean 4, formalize the minimal φ -calculus and the main confluence result [18, Theorem 2.3]. The rest of the paper is organized as follows:

- In Section 1.1, we list some of related work on object-oriented calculi, confluence, and formalization of abstract rewriting.
- In Section 2, we briefly overview the definition of φ -calculus and the approach to the representation of object formations that we take in Lean 4.
- In Section 3, we establish a basic proof-relevant framework for abstract rewriting.
- In Section 4, we formalize the proof of confluence for the minimal φ -calculus, discovering and fixing minor mistakes in the statement of the Substitution Lemma in the original paper [18, Lemma 2.3].

¹ See <https://www.elegantobjects.org>.

² <https://www.eolang.org>



$t ::= \llbracket a_1 \mapsto \emptyset, \dots, a_n \mapsto \emptyset, b_1 \mapsto t_1, \dots, b_k \mapsto t_k \rrbracket$	(object formation)
$t.a$	(dispatch)
$t_1(a \mapsto t_2)$	(application)
ρ^n	(locator)

■ **Figure 1** Complete syntax of the minimal φ -calculus [18].

1.1 Related Work

Object-oriented calculi

There is a plethora of object-oriented calculi, which can roughly be classified into type-theoretic models with “purely functional” objects [31, 24, 6], λ -calculi with mutable objects [1, 5], Featherweight Java family [17, 23, 9], models for multiple dispatch [8, 2, 26]. The family of calculi most related to φ -calculus are those that model delegation-based inheritance, such as λObj and its variants [12, 10]. That said, φ -calculus relies on the Decorator pattern [15, §4], which involves a different delegation mechanism.

Formalizations of Abstract Rewriting Systems

A notable formalization library of rewriting systems is ISAFOR (Isabelle Formalization of Rewriting)³. It contains a general ARS framework, upon which numerous results about confluence have been built [21, 27, 20, 28].

Another term rewriting library is built in Prototype Verification System (PVS) as part of NASALib⁴. The library includes confluence of orthogonal systems [25], Knuth-Bendix critical pair theorem [13], and Newman’s and Yokouchi’s [14] lemmas for higher-order systems.

Other formalized results include confluence of Rewriting Type Theory (RTT) [11], the Agda type system extended with the rewrite rules. In the proof, the authors use the Takahashi’s technique [29] to show confluence of parallel reduction. The confluence check was implemented in Agda and formally verified in MetaCoq.

Another proof of confluence that uses Takahashi’s technique is confluence of Lean’s Type Theory [7]. There is also an ongoing effort in formalizing metatheory of Lean⁵.

Lean’s `mathlib` [19] has a module `Logic.Relation` for working with relations, including the ones that are subject to abstract rewriting. Among other things, the module contains definitions of reflexive-transitive closure (`Relation.ReflTransGen`), `join` (`Relation.Join`), and a proof that strong confluence implies confluence (`Relation.church_rosser`). In these definitions, the reduction relation is `Prop`-valued which prevents these results from being used in a proof-relevant setting.

Confluence checkers

Other category of related work are confluence checkers. In confluence checkers, decisions about the representation and the supported features of the abstract rewriting systems are

³ <http://cl-informatik.uibk.ac.at/isafor/>

⁴ <https://github.com/nasa/pvslib/tree/master/TRS>

⁵ <https://github.com/digama0/lean4lean/>

made, and then the input term rewriting system is checked against some confluence criterion on which the tool is based. For a meaningful comparison of such tools an annual competition CoCo and the dataset of the confluence problems[16] have been created. There are numerous instruments aimed for first-order, higher-order, and conditional rewriting systems.

Confluence for λ -calculus

The original proof of confluence of λ -calculus is due to Church and Rosser and uses “finiteness of developments” method [3]. Tait and Martin-Löf prove confluence of λ -calculus by introducing a notion of parallel reduction [3]. Takahashi[29] further simplifies Tait and Martin-Löf’s proof with the notion of complete development and triangle property of parallel reduction. Formalization of λ -calculus and this proof of confluence has been done at least in Agda [30] and Isabelle/HOL [22].

2 Overview of the Minimal φ -Calculus

Syntax of φ -calculus and its informal semantics was originally introduced by Bugaenko [4] as an attempt to provide mathematical foundation for his ideas of object-oriented programming, following the Elegant Objects⁶ paradigm and an experimental programming language EOLANG⁷. Kudasov and Sim [18] extracted the core calculus with formal reduction semantics, which we briefly overview in this section.

The syntax of φ -terms is presented in Figure 1 and consists of:

1. *object formation* that represents an object with some attributes; each attribute can be *attached* to an object, or be *void* (attached to nothing);
2. *dispatch* (or *attribute access*) that extracts the term attached to an attribute of an object;
3. *application* that assigns a term to a void attribute of an object;
4. *locators* that reference one of the outer object formations.

Kudasov and Sim [18] define object formations as mappings from a set of labels \mathcal{L} to an extended set of terms $\mathcal{T} \cup \{\emptyset, \perp\}$. However, implementing formations as functions in Lean 4 does not work out well, since functions are too opaque compared to data structures like lists. For example, it would be difficult to easily print objects represented as functions, without keeping some additional structure (such as sets of void and attached attributes).

Instead, we take a more computationally pragmatic approach⁸. We define object formations (**Bindings**) as lists of triplets (l, t, u) , where $l \in \mathcal{L}$ is a label, $t \in (\mathcal{T} \cup \{\emptyset\})$ is a void or attached term and u is a proof that l does not appear in the list’s tail. Presence of u in each triplet serves as a proof that all labels in a given object are unique.

To work comfortably with such representation, we provide a collection of utility functions and theorems: insertion and its properties (e.g. **Insert.insertTwice**), proofs of presence of certain terms, and more.

⁶ See <https://www.elegantobjects.org>.

⁷ <https://www.eolang.org>

⁸ The encoding took several attempts, and with the help from the Lean Zulip chat, we were able to finalize it. See the discussion at <https://leanprover.zulipchat.com/#narrow/stream/113489-new-members/topic/.E2.9C.94.20unique.20record.20fields.20.28and.2C.20probably.2C.20a.20bug.29>.

$$\begin{array}{c}
\frac{t_i \rightsquigarrow t'_i}{\llbracket \dots, b_i \mapsto t_i, \dots \rrbracket \rightsquigarrow \llbracket \dots, b_i \mapsto t'_i, \dots \rrbracket} \text{cong}_{\text{OBJ}} \quad \frac{t \rightsquigarrow t'}{t.a \rightsquigarrow t'.a} \text{cong}_{\text{DOT}} \\
\\
\frac{t \rightsquigarrow t'}{t(a \mapsto u) \rightsquigarrow t'(a \mapsto u)} \text{cong}_{\text{APPL}} \quad \frac{u \rightsquigarrow u'}{t(a \mapsto u) \rightsquigarrow t(a \mapsto u')} \text{cong}_{\text{APPR}} \\
\\
\frac{t \equiv \llbracket \dots, c \mapsto t_c, \dots \rrbracket}{t.c \rightsquigarrow t_c [\rho^0 \mapsto t]} \text{DOT}_c \quad \frac{t \equiv \llbracket \dots \rrbracket \quad c \notin \text{attr}(t) \quad \varphi \in \text{attr}(t)}{t.c \rightsquigarrow t.\varphi.c} \text{DOT}_c^\varphi \\
\\
\frac{t \equiv \llbracket a_1 \mapsto \emptyset, \dots, a_k \mapsto \emptyset, c \mapsto \emptyset, b_1 \mapsto t_1, \dots, b_n \mapsto t_n \rrbracket}{t(c \mapsto u) \rightsquigarrow \llbracket a_1 \mapsto \emptyset, \dots, a_k \mapsto \emptyset, c \mapsto u \uparrow, b_1 \mapsto t_1, \dots, b_n \mapsto t_n \rrbracket} \text{APP}_c
\end{array}$$

■ **Figure 2** Reduction semantics (**Reduce**) of the minimal φ -calculus [18, Figure 1].

3 Metatheory for Rewriting in Lean 4

In this section, we describe formalizations that are related to Abstract Rewriting Systems, but not depend on φ -calculus. Our formalization follows the approach taken in `mathlib`'s `Logic.Relation` module, except we work in a proof-relevant setting, taking `Type`-valued reduction relations. So far, it contains the bare minimum to prove confluence results in our specific case, but we are willing to develop it further in the future.

First, we define reflexive-transitive closure (`ReflTransGen`), and prove that it is in fact transitive (`ReflTransGen.trans`). Then, we define the join of two terms, the diamond property, and confluence. Let $R : \alpha \rightarrow \alpha \rightarrow \text{Type } u$ be a reduction relation in the following definitions.

► **Definition 1** (`Join`). The join of $a, b : \alpha$ is an element $c : \alpha$, such that $R(a, c)$ and $R(b, c)$.

► **Definition 2** (`DiamondProperty`). Relation R possesses the diamond property if for any $a, b, c : \alpha$, such that $R(a, b)$ and $R(a, c)$, b and c can be joined using R .

► **Definition 3** (`Confluence`). Relation R is confluent if for any $a, b, c : \alpha$, such that $R(a, b)$ and $R(a, c)$, b and c can be joined using the reflexive transitive closure of R .

We show that diamond property implies confluence by induction on reflexive-transitive closure of the reductions (`diamond_implies_confluence`). This concludes the ARS module that we developed for proving confluence of φ -calculus.

In the proof of confluence for φ -calculus in Section 4, we use Takahashi's technique [29] and introduce parallel reduction and complete development, show the triangle property of parallel reduction and equivalence of regular and parallel reductions. Many of these definitions and proofs can be generalized, using Lean's metaprogramming. However, we leave this for future work, and the formalization that follows is specialized to φ -calculus.

4 Confluence for the Minimal φ -Calculus

In this section, we prove confluence for the minimal φ -calculus. The relevant formalization is available in the `Minimal/Calculus` module. Figure 2 shows the reduction rules. Our proof follows Takahashi's technique [29]. We start by introducing the parallel reduction (`PReduce`), as defined in [18, Figure 2]. To accommodate for the parallel congruence of object formations (`congOBJ`), we introduce a mutually defined helper construction (`Premise`).

We show that reflexive-transitive closure of regular reduction (**RedMany**) is equivalent to reflexive-transitive closure of parallel reduction (**ParMany**). It enables us to prove that confluence of regular reduction follows from confluence of parallel reduction (**confluence_reduce**).

► **Theorem 4.** *Parallel reduction (\Rightarrow) is equivalent to regular reduction (\rightsquigarrow).*

1. $t \rightsquigarrow t'$ implies $t \Rightarrow t'$ (**reg_to_par**)
2. $t \rightsquigarrow^* t'$ implies $t \Rightarrow^* t'$ (**redMany_to_parMany**)
3. $t \Rightarrow t'$ implies $t \rightsquigarrow^* t'$ (**par_to_redMany**)
4. $t \Rightarrow^* t'$ implies $t \rightsquigarrow^* t'$ (**parMany_to_redMany**)

The first step to prove the diamond property of parallel reduction is *substitution lemma* used in the following proofs.

► **Lemma 5** (**substitution_lemma**). *Let t, t', u, u' be φ -terms with $t \Rightarrow t'$ and $u \Rightarrow u'$. If $\min_\rho^0(u') \geq i$, then $t[\rho^i \mapsto u] \Rightarrow t'[\rho^i \mapsto u']$.*

The lemma requires a condition on the locators pointing outside the outermost formation:

► **Definition 6** (**min_free_loc**). *The minimal free locator $\min_\rho^i : \mathcal{T} \rightarrow (\mathbb{N} \cup \{\top\})$ is defined inductively on φ -terms:*

$$\begin{aligned} \min_\rho^i(\rho^n) &:= \begin{cases} n - i, & \text{if } n \geq i \\ \top & \text{otherwise} \end{cases} \\ \min_\rho^i(t.a) &:= \min_\rho^i(t) \\ \min_\rho^i(t(a \mapsto u)) &:= \min(\min_\rho^i(t), \min_\rho^i(u)) \\ \min_\rho^i(\llbracket a_1 \mapsto \emptyset, \dots, a_k \mapsto \emptyset, b_1 \mapsto t_1, \dots, b_n \mapsto t_n \rrbracket) &:= \min(\min_\rho^{(i+1)}(t_1), \dots, \min_\rho^{(i+1)}(t_n)) \end{aligned}$$

We assume that for all $n \in \mathbb{N}$, $\top \geq n$.

The condition on minimal free locators is used in the $\text{DOT}_c^{\Rightarrow}$ case of $t \Rightarrow t'$ of the substitution lemma, which relies on *Substitution Swap* lemma [18, Lemma A.7]:

► **Lemma 7** (**subst_swap**). *Let i, j be natural numbers, $j \leq i$ and t, u, v be φ -terms. If $\min_\rho^0(v) \geq i$, then $t[\rho^j \mapsto u][\rho^i \mapsto v] = t[\rho^{(i+1)} \mapsto v \uparrow][\rho^j \mapsto u[\rho^i \mapsto v]]$.*

► **Remark 8.** The definition of Lemma 5 and Lemma 7 differ from their counterparts in original paper [18], having an extra condition on the minimal free locators (\min_ρ^i) in u' . In fact, without this condition the lemmas are not true. In particular, without the condition Lemma 7 does not hold when $t = \rho^{(i+1)}$. Originally, the lemmas were used only in a safe context, hiding the problem and keeping the main result valid. The process of formalization with a computer proof assistant made this mistake obvious and allowed us to fix the statement of Lemma 5.

To complete the proof, two new lemmas (not present in [18]) are required:

► **Lemma 9** (**min_free_loc_inc**). *If $\min_\rho^k(v) \geq i$, then $\min_\rho^k(v \uparrow^k) \geq i + 1$.*

► **Lemma 10** (**subst_inc_cancel**). *Let v, u be φ -terms and j, k, i, z natural numbers, such that $j \leq i + z$, $k \leq i + z$, $z \leq j$, $z \leq k$. If $\min_\rho^z(v) \geq i$, then $v = v \uparrow^k[\rho^j \mapsto u]$.*

Lemma 9 states that *locator increment* \uparrow^k increments the minimal free locator if it acts only on free locators. Lemma 10 states that *locator substitution* and *locator increment* cancel effect of each other if they act *only* on free locators.

We now follow the rest of the Takahashi's technique [29] and introduce the *complete development* of a term:

175 ► **Definition 11** (`complete_development`). Let t be a φ -term. Then a term t^+ denotes the
 176 complete development of t , which is defined à la Kudasov and Sim [18, Definition 2.7].

177 The complete development possesses the property that all parallel reductions of a term
 178 reduce to complete development. We use this property to show that all divergent one-step
 179 parallel reductions converge in one step to the complete development:

180 ► **Proposition 12** (`half_diamond`). Let t, t' be φ -terms with $t \Rightarrow t'$. Then $t' \Rightarrow t^+$.

181 ► **Corollary 13** (`diamond_preduce`). Let t, u, v be φ -terms with $t \Rightarrow u$ and $t \Rightarrow v$. Then
 182 there exists a φ -term w such that $u \Rightarrow w$ and $v \Rightarrow w$.

183 **Proof.** Let $w \equiv t^+$. From Proposition 12, $u \Rightarrow w$ and $v \Rightarrow w$. ◀

184 ► **Corollary 14** (`confluence_preduce`). Let t, u, v be φ -terms with $t \xRightarrow{*} u$ and $t \xRightarrow{*} v$. Then
 185 there exists a φ -term w such that $u \xRightarrow{*} w$ and $v \xRightarrow{*} w$.

186 **Proof.** Follows from Corollary 13 and the fact that diamond property implies confluence
 187 (`diamond_implies_confluence`). ◀

188 ► **Corollary 15** (`confluence_reduce`). Let t, u, v be φ -terms with $t \rightsquigarrow^* u$ and $t \rightsquigarrow^* v$. Then
 189 there exists a φ -term w such that $u \rightsquigarrow^* w$ and $v \rightsquigarrow^* w$.

190 In the process of formalization, we were stuck with Lean’s pattern matching behavior a
 191 few times. For instance, the proof of Lemma 5 makes use of the fact that the only possible
 192 reduction of an object formation is congruence. When pattern matching on it in Lean,
 193 however, we were asked for other (impossible) cases. Simply adding the cases did not type
 194 check. Moving the match into a separate function (`get_premise`) resolved the issue. We
 195 have yet to construct a minimal working example to submit an issue to the Lean GitHub
 196 repository.

197 Although we have the proof of confluence, at the moment it is incomplete because a few
 198 lemmas lack a termination proof. Adding an explicit termination proof requires a significant
 199 complication in some places, also negating code reuse. It is possible that Lean can be taught
 200 to understand termination in the case (mutual) structural induction better, but so far we
 201 have not found a suitable solution (except writing explicit proofs).

202 5 Conclusion and Future Work

203 We formalize the minimal φ -calculus in Lean 4, providing basic definitions for proof-relevant
 204 abstract syntax rewriting and proving confluence for φ -calculus in a proof assistant. Work-
 205 ing with Lean exposed a mistake in an earlier statement of the Substitution Lemma [18,
 206 Lemma 2.3], which lacked an extra condition. Our formalization fixes the mistake and shows
 207 that confluence still holds.

208 There is a semantic gap between the minimal φ -calculus and EOLANG. First, EO treats
 209 its global object as an anchor for the (runtime) object graph, while the minimal φ -calculus
 210 does not have such a concept. Second, EO supports primitive data and operators (atoms),
 211 which are also not handled by the minimal φ -calculus. Finally, in EOLANG, the closest
 212 enclosing object formation is accessible through “this” locator $\$$, while outer formations are
 213 available through the special “parent” attribute (\wedge) , not a family of locators ρ^n . Thus, one
 214 direction of future work involves formalization of the semantics of EOLANG and extended
 215 version of φ -calculus.

216 Another direction of future work is to develop a proper library for proof-relevant treatment
 217 of abstract rewriting systems in Lean, extending the work presented in Section 3.

References

- 1 Martín Abadi and Luca Cardelli. A theory of primitive objects: Untyped and first-order systems. *Inf. Comput.*, 125(2):78–102, 1996. URL: <https://www.sciencedirect.com/science/article/pii/S0890540196900243>, doi:10.1006/inco.1996.0024.
- 2 Eric Allen, Joseph J. Hallett, Victor Luchangco, Sukyoung Ryu, and Guy L. Steele. Modular multiple dispatch with multiple inheritance. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, page 1117–1121, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1244002.1244245.
- 3 Henk Barendregt and Erik Barendsen. Introduction to lambda calculus. *Nieuw archief voor wetenschap*, 4:337–372, 01 1984.
- 4 Yegor Bugayenko. EOLANG and φ -calculus, 2024. [arXiv:2111.13384](https://arxiv.org/abs/2111.13384).
- 5 L. Cardelli, S. Martini, J.C. Mitchell, and A. Scedrov. An extension of system f with subtyping. *Information and Computation*, 109(1):4–56, 1994. URL: <https://www.sciencedirect.com/science/article/pii/S0890540184710133>, doi:10.1006/inco.1994.1013.
- 6 Luca Cardelli. *Extensible Records in a Pure Calculus of Subtyping*, page 373–425. MIT Press, Cambridge, MA, USA, 1994.
- 7 Mario Carneiro. Lean’s type theory. Master’s thesis, Carnegie Mellon University, 2019. URL: <https://github.com/digama0/lean-type-theory/releases/download/v1.0/main.pdf>.
- 8 Giuseppe Castagna, Giorgio Ghelli, and Giuseppe Longo. A calculus for overloaded functions with subtyping. In *Proceedings of the 1992 ACM Conference on LISP and Functional Programming*, LFP '92, page 182–192, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/141471.141537.
- 9 Elias Castegren and Tobias Wrigstad. Oolong: A concurrent object calculus for extensibility and reuse. *SIGAPP Appl. Comput. Rev.*, 18(4):47–60, Jan. 2019. doi:10.1145/3307624.3307629.
- 10 Alberto Ciaffaglione, Pietro Di Gianantonio, Furio Honsell, and Luigi Liquori. A prototype-based approach to object evolution. *J. Object Technol.*, 20(2):1–24, 2021. doi:10.5381/jot.2021.20.2.a4.
- 11 Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. The taming of the rew: a type theory with computational assumptions. *Proc. ACM Program. Lang.*, 5(POPL), jan 2021. doi:10.1145/3434341.
- 12 Kathleen Fisher, Furio Honsell, and John C. Mitchell. A lambda calculus of objects and method specialization. *Nordic J. of Computing*, 1(1):3–37, Mar. 1994.
- 13 André L Galdino and Mauricio Ayala-Rincón. A formalization of the knuth–bendix (–huet) critical pair theorem. *Journal of Automated Reasoning*, 45:301–325, 2010.
- 14 André Luiz Galdino and Mauricio Ayala-Rincón. A formalization of newman’s and yokouchi’s lemmas in a higher-order language. *Journal of Formalized Reasoning*, 1(1):39–50, 2008.
- 15 Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995.
- 16 Nao Hirokawa, Julian Nagele, and Aart Middeldorp. Cops and cocoweb: Infrastructure for confluence tools. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning*, pages 346–353, Cham, 2018. Springer International Publishing.
- 17 Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight java: A minimal core calculus for java and gj. *ACM Trans. Program. Lang. Syst.*, 23(3):396–450, May 2001. doi:10.1145/503502.503505.
- 18 Nikolai Kudasov and Violetta Sim. Formalizing φ -calculus: A purely object-oriented calculus of decorated objects. In *Proceedings of the 24th ACM International Workshop on Formal Techniques for Java-like Programs*, FTfJP '22, page 29–36, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3611096.3611103.
- 19 The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page

- 367–381, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3372885.3373824.
- 20 Julian Nagele, Bertram Felgenhauer, and Harald Zankl. Certifying confluence proofs via relative termination and rule labeling. *Logical Methods in Computer Science*, 13(2:4):1–27, 2017. doi:10.23638/LMCS-13(2:4)2017.
- 21 Julian Nagele and Aart Middeldorp. Certification of classical confluence results for left-linear term rewrite systems. In Jasmin Christian Blanchette and Stephan Merz, editors, *Proceedings of the 7th International Conference on Interactive Theorem Proving*, volume 9807 of *Lecture Notes in Computer Science*, pages 290–306, 2016. doi:10.1007/978-3-319-43144-4_18.
- 22 Tobias Nipkow. More church-rosser proofs (in isabelle/hol). In *International Conference on Automated Deduction*, pages 733–747. Springer, 1996.
- 23 Johan Östlund and Tobias Wrigstad. Welterweight java. In *Proceedings of the 48th International Conference on Objects, Models, Components, Patterns*, TOOLS’10, page 97–116, Berlin, Heidelberg, 2010. Springer-Verlag.
- 24 Benjamin C Pierce and David N Turner. Simple type-theoretic foundations for object-oriented programming. *J. Funct. Program.*, 4(2):207–247, 1994.
- 25 Ana Cristina Rocha-Oliveira, André Luiz Galdino, and Mauricio Ayala-Rincón. Confluence of orthogonal term rewriting systems in the prototype verification system. *Journal of Automated Reasoning*, 58:231–251, 2017.
- 26 Lee Salzman and Jonathan Aldrich. Prototypes with multiple dispatch: An expressive and dynamic object model. In *Proceedings of the 19th European Conference on Object-Oriented Programming*, ECOOP’05, page 312–336, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11531142_14.
- 27 Christian Sternagel and Thomas Sternagel. Certifying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.FSCD.2016.29>, doi:10.4230/LIPIcs.FSCD.2016.29.
- 28 Christian Sternagel and René Thiemann. Formalizing Knuth-Bendix Orders and Knuth-Bendix Completion. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 287–302, Dagstuhl, Germany, 2013. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.RTA.2013.287>, doi:10.4230/LIPIcs.RTA.2013.287.
- 29 Masako Takahashi. Parallel reductions in λ -calculus. *Inf. Comput.*, 118(1):120–127, 1995. URL: <https://www.sciencedirect.com/science/article/pii/S0890540185710577>, doi:10.1006/inco.1995.1057.
- 30 Philip Wadler. Programming language foundations in agda. In *Formal Methods: Foundations and Applications: 21st Brazilian Symposium, SBMF 2018, Salvador, Brazil, November 26–30, 2018, Proceedings 21*, pages 56–73. Springer, 2018.
- 31 Mitchell Wand. Type inference for record concatenation and multiple inheritance. *Inf. Comput.*, 93(1):1–15, 1991. URL: <https://www.sciencedirect.com/science/article/pii/S089054019190050C>, doi:10.1016/0890-5401(91)90050-C.