Caos: A Reusable Scala Web Animator of Operational Semantics

José Proença[1][0000−0003−0971−8919] and Luc Edixhoven[2][0000−0002−6011−9535]

[1] CISTER, ISEP, Polytechnic Institute of Porto, Portugal pro@isep.ipp.pt
[2] Open University (Heerlen) and CWI (Amsterdam), Netherlands led@ou.nl

Abstract. This tool paper presents Caos: a methodology and a programming framework for *computer-aided design of structural operational semantics for formal models*. This framework includes a set of Scala libraries and a workflow to produce visual and interactive diagrams that animate and provide insights over the structure and the semantics of a given abstract model with operational rules.
Caos follows an approach in which theoretical foundations and a practical tool are built together, as an alternative to foundations-first design ("tool justifies theory") or tool-first design ("foundations justify practice"). The advantage of Caos is that the tool-under-development can immediately be used to automatically run numerous and sizeable examples in order to identify subtle mistakes, unexpected outcomes, and unforeseen limitations in the foundations-under-development, as early as possible.
We share two success stories of Caos' methodology and framework in our own teaching and research context, where we analyse a simple while-language and a choreographic language, including their operational rules and the concurrent composition of such rules. We further discuss how others can include Caos in their own analysis and Scala tools.

Demo video: https://zenodo.org/record/7876060 & https://youtu.be/Xcfn3zqpubw

Hands-on tutorial: In a companion report [17, Appendix A].

1 Introduction

Designing formal methods can be hard. Typical challenges of formal-methods-related research include identifying and dealing with corner cases, discovering missing assumptions, finding the right abstraction level, and—of course—proving theorems (and adequately decomposing them into lemmas). Curiously, and unlike other scientific disciplines, we find that a large majority of papers written in our community primarily focuses on research *results* instead of *methods*. In contrast, this tool paper contributes to *the methodology* of designing formal methods, with special emphasis on Structural Operational Semantics (SOS): we share our experiences with *computer-aided design of SOS for formal methods* with a set of examples produced by our toolset Caos. Source code and a compilation of examples can be found at https://github.com/arcalab/caos. We hope that it may inspire colleagues both to apply our methodology and tools, and to share their own methodology-related experiences to our community's benefit.

In a nutshell, in Caos, theoretical foundations and a practical tool are built together side-by-side, from the start, as an alternative to the more typical *foundations-first design* ("tool justifies theory") or *tool-first design* ("foundations justify practice"). The main advantage of Caos is that the tool-under-development can immediately be used to automatically run numerous and sizeable examples in order to identify subtle mistakes, unexpected outcomes, and/or unforeseen limitations in the foundations-under-development, as early as possible. This need for validation and supporting tools in formal methods has been acknowledged, e.g., by Garavel et al. in a recent survey over formal methods in critical systems [12].

The Caos toolset is based on ReoLive,[3] which was developed as an online set of Scala & JavaScript (JS) tools to analyse Reo connectors [6]. Currently it also hosts many extensions unrelated to Reo [13,5], where common code blocks can be compiled both to JS (client) and to Java binaries (server), allowing computations to be delegated to a remote server. Consequently, it became a **monolithic** implementation with many **replicated** blocks of code for different independent extensions, and it is non-trivial to **reuse** it for different projects. Our alternative Caos toolset aims at addressing these issues, targeting the following requirements:

– **R1:** Caos should use a general programming language, facilitating adoption and supporting more complex back-ends when desired;
– **R2:** The output from Caos-supported implementation should be easy to execute and use, without requiring specific platforms or complex installations;
– **R3:** Caos should be easily reused, and Caos-supported implementations should be modular and easily extended with new analyses.

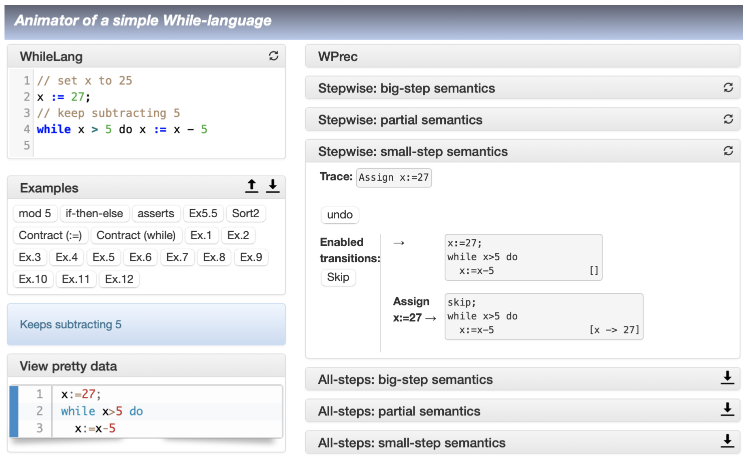
Guided by these requirements, our Caos toolset is implemented in Scala (R1), compiles to JS that generates intuitive and interactive websites (R2), and includes a simple-to-extend API that facilitates its usage and reuse by other developers (R3). By using the Caos toolset, one can produce a webpage such as the one in Fig. 1. This webpage has an input text box and a collection of widgets that depict or animate different analyses over the input program, exploiting possible operational semantics when applicable. This example will be further detailed in Section 2, which analyses a simple while-language (with contracts).

Caos includes dedicated support for SOS, by animating, depicting, or comparing terms that implement a `next` and an `accepting` method. It further supports building SOS for networks of interacting components, mentioned in Section 3.

Similar approaches to support the development of language semantics exist, such as the ones below, which do not address all of the 3 requirements above.

The **Maude language** and toolset [3] focus on how to specify (1) a configuration (a state) using a sequence of characters, and (2) a set of possible rewrite rules capturing how configurations can be modified. It further provides a set of constructs to facilitate the creation of new syntactical notations, such as marking operators as being associative and with a given identity. Maude includes well polished model checkers and other analysis tools; other model checkers (e.g., mCRL2 [1], UPPAAL [7]) also have specification languages with an operational

[3] https://github.com/ReoLanguage/ReoLive


Animator of a simple While-language
WhileLang
Examples
mod 5
if-then-else
asserts
Ex5.5
Sort2
Contract (:=)
Contract (while)
Ex.1
Ex.2
Ex.3
Ex.4
Ex.5
Ex.6
Ex.7
Ex.8
Ex.9
Ex.10
Ex.11
Ex.12
Keeps subtracting 5
View pretty data
WPrec
Stepwise: big-step semantics
Stepwise: partial semantics
Stepwise: small-step semantics
Trace:
undo
Enabled transitions:
Skip
All-steps: big-step semantics
All-steps: partial semantics
All-steps: small-step semantics

```
// set x to 25
x := 27;
// keep subtracting 5
while x > 5 do x := x - 5
```

```
x:=27;
while x>5 do
  x:=x-5
```

Trace: `Assign x:=27`

```
→     x:=27;
      while x>5 do
        x:=x-5              []

Assign
x:=27 →  skip;
         while x>5 do
           x:=x-5           [x -> 27]
```

Fig. 1. Screenshot of the web interface to analyse a simple while-language, available at https://cister-labs.github.io/whilelang-scala/

semantics, restricted by design to provide better model-checking support. These approaches provide a similar functionality but do not target our requirements.

Racket (and its DrRacket graphical frontend) [11] is a *Language-Oriented Programming Language*, i.e., a language meant for making languages. It is widely adopted and comes with a large collection of libraries, and includes a set of constructs that facilitate the creation of new syntactical notations, bundled as new languages, allowing multiple languages in a program to exist and to be created on the fly. Embedded in Racket, PLT Redex [10] is a domain specific language for specifying and debugging operational semantics, which receives a grammar and reduction rules and supports an interactive exploration of the terms. Arguably, Racket is a general purpose language (R1), although less adopted than Java or Scala, with extension mechanisms to support reusability (R3), and which we believe to be harder to deploy products (R2).

Some **teaching languages**, such as Pyret [16], are designed to be compiled to JavaScript and to be used when teaching introductory computing, balancing expressiveness and performance. It includes a powerful runtime to hide from the user some of the intricacies and limitations of JS, and this and similar languages include visualisation libraries to better engage students. These languages do not share the same functional goal, and do not use a general programming language (R1), but can often produce easy-to-run code (R2) and be extendable (R3).

Caos is particularly useful for users familiarised with Scala/Java, and less to users with some experience in languages such as Maude, Racket, or Pyret.

Paper structure: This paper starts by describing our experience with Caos both in a teaching (Section 2) and a research (Section 3) context, focused on what can be produced using the toolset. Section 4 describes how the Caos toolset is structured and how it can be used by others, and Section 5 concludes this paper.

2 Use-case: a While-language for Teaching

In the context of a university course, students were taught about natural and operational semantics, and how to infer weakest preconditions. We, as teachers, used a simple while-language with integers to describe these concepts. We created a simple website **in a couple of days** using Caos, depicted in Fig. 1, improving core widgets over the period of one week. Note that we were familiarised with the tools and had some experience with writing parsers in Scala. This website was used by the students to experiment and gain better insights over the concepts.

Fig. 1 illustrates the compiled output of Caos: a collection of widgets that always includes an input widget (here called `WhileLang`) and a list of example input programs. The other widgets are custom-made, and include: (1) *visualisation* of a string produced from the program, representing plain text, code, or a mermaid diagram (a popular Markdown-like language for diagrams); [4] and (2) *execution* given a `next` function that evolves the program, which can be presented either step-wise (interactive) or as a single state diagram with all reachable states. Caos also provides widgets for (3) *comparing* two program behaviours using bisimilarity or trace equivalence; and (4) *checking* for errors or warnings in a program.

Figure 1 depicts a visualisation of the source code (bottom left) and a step-wise evolution using a small-step semantics with a textual representation (right), and the remaining widgets are collapsed. These collapsed widgets use different semantics, provide a view of all steps, or calculate the weakest preconditions, and are not processed while collapsed. Students could use better understand which rules could be applied at each moment, and navigate through the state space.

3 Use-case: Analysing Choreographies in Research

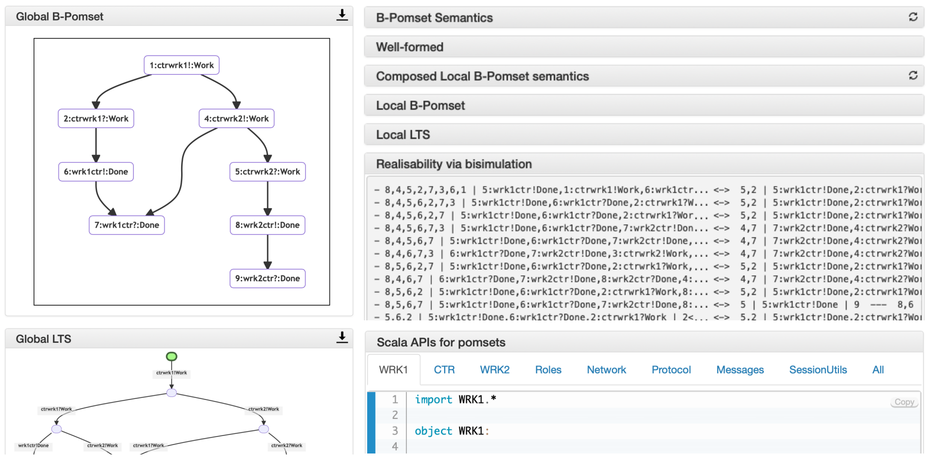
Caos can be used to illustrate research concepts using prototyping tools. We used it, for example, when investigating choreographic languages. A choreographic language describes possible sequences of interactions between agents, e.g.,

$$\mathtt{ctr} \to \mathtt{wrk1}{:}\mathsf{Work}\ ;\ \mathtt{ctr} \to \mathtt{wrk2}{:}\mathsf{Work}\ ;\ (\mathtt{wrk1} \to \mathtt{ctr}{:}\mathsf{Done} \parallel \mathtt{wrk2} \to \mathtt{ctr}{:}\mathsf{Done})$$

captures a scenario where a controller `ctr` delegates some Work to two workers, and they reply once they are Done. Together with Guillermina Cledou and Sung-Shik Jongmans we published several choreography analyses supported by Caos-based prototypes, investigating how to detect that the behaviour of the local agents induce the global behaviour (known as realisability) using a novel underlying mathematical structure [9,8] (`https://lmf.di.uminho.pt/b-pomset`) and how to generate APIs that statically guarantee that the local agents follow their interaction protocol [4,14] (`https://lmf.di.uminho.pt/{pompset,st4mp}`).

An underlying mathematical structure was used to give semantics to choreographies: branching pomsets [9] (which are similar to event structures [15,2]). As shown in Fig. 2, using Caos it was possible to: (1) *visualize* the pomset structure (top left); (2) *execute* a pomset (B-Pomset Semantics) and the composition

[4] `https://mermaid-js.github.io/mermaid`



```
- 8,4,5,2,7,3,6,1 | 5:wrk1ctr!Done,1:ctrwrk1!Work,6:wrk1ctr... <->  5,2 | 5:wrk1ctr!Done,2:ctrwrk1?Wo
- 8,4,5,6,2,7,3 | 5:wrk1ctr!Done,6:wrk1ctr?Done,2:ctrwrk1?W... <->  5,2 | 5:wrk1ctr!Done,2:ctrwrk1?Wo
- 8,4,5,6,2,7 | 5:wrk1ctr!Done,6:wrk1ctr?Done,2:ctrwrk1?Wor... <->  5,2 | 5:wrk1ctr!Done,2:ctrwrk1?Wo
- 8,4,5,6,7,3 | 5:wrk1ctr!Done,6:wrk1ctr?Done,7:wrk2ctr!Don... <->  4,7 | 7:wrk2ctr!Done,4:ctrwrk2?Wo
- 8,4,5,6,7 | 5:wrk1ctr!Done,6:wrk1ctr?Done,7:wrk2ctr!Done,... <->  4,7 | 7:wrk2ctr!Done,4:ctrwrk2?Wo
- 8,4,6,7,3 | 6:wrk1ctr?Done,7:wrk2ctr!Done,3:ctrwrk2!Work,... <->  4,7 | 7:wrk2ctr!Done,4:ctrwrk2?Wo
- 8,5,6,2,7 | 5:wrk1ctr!Done,6:wrk1ctr?Done,2:ctrwrk1?Work,... <->  5,2 | 5:wrk1ctr!Done,2:ctrwrk1?Wo
- 8,4,6,7 | 6:wrk1ctr?Done,7:wrk2ctr!Done,8:wrk2ctr?Done,4:... <->  4,7 | 7:wrk2ctr!Done,4:ctrwrk2?Wo
- 8,5,6,2 | 5:wrk1ctr!Done,6:wrk1ctr?Done,2:ctrwrk1?Work,8:... <->  5,2 | 5:wrk1ctr!Done,2:ctrwrk1?Wo
- 8,5,6,7 | 5:wrk1ctr!Done,6:wrk1ctr?Done,7:wrk2ctr!Done,8:... <->  5 | 5:wrk1ctr!Done | 9  ---  8,6
- 5,6,2 | 5:wrk1ctr!Done,6:wrk1ctr?Done,2:ctrwrk1?Work | 2<... <->  5,2 | 5:wrk1ctr!Done,2:ctrwrk1?Wo
```

```scala
import WRK1.*

object WRK1:

```

Fig. 2. Analysis of branching pomsets produced by Caos from a choreography language

of its projections to each agent involved (Composed Local B-Pomset Semantics); (3) *check well-formedness* (Well-formed), a novel syntactic (sound but incomplete) realisability check; (4) *check realisability* using a (complete but more complex) search for a bisimulation between the global behaviour and the composed behaviour of the projections (Realisability via bisimulation); and (5) *generate Scala code* with libraries that can guarantee at compile time that local agents obey the expected protocol (bottom right). Caos provides constructors for the composition of the behaviour of the local agents and for the search for bisimulations. Setting up each of these websites **took around half a week** of work by one person. During our investigation, the Caos-supported implementation was a *crucial* mechanism to experiment with many variations of the semantics and projections of the choreography language, of the pomset structure, and of the realisability analysis, ultimately converging to the current version.

4 Caos framework

This section describes what Caos provides and how to use it to produce animators such as the ones in Sections 2 and 3. Figure 3 depicts the structure of a typical Scala project that uses Caos. The user provides data structures for the input language with functions to parse this language and to compute analysis

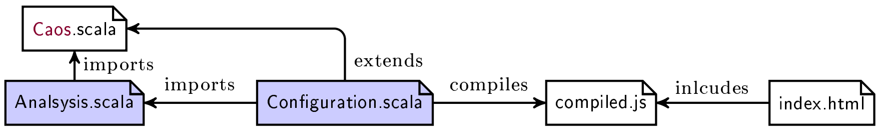


Fig. 3. Architecture of a Scala project that uses Caos

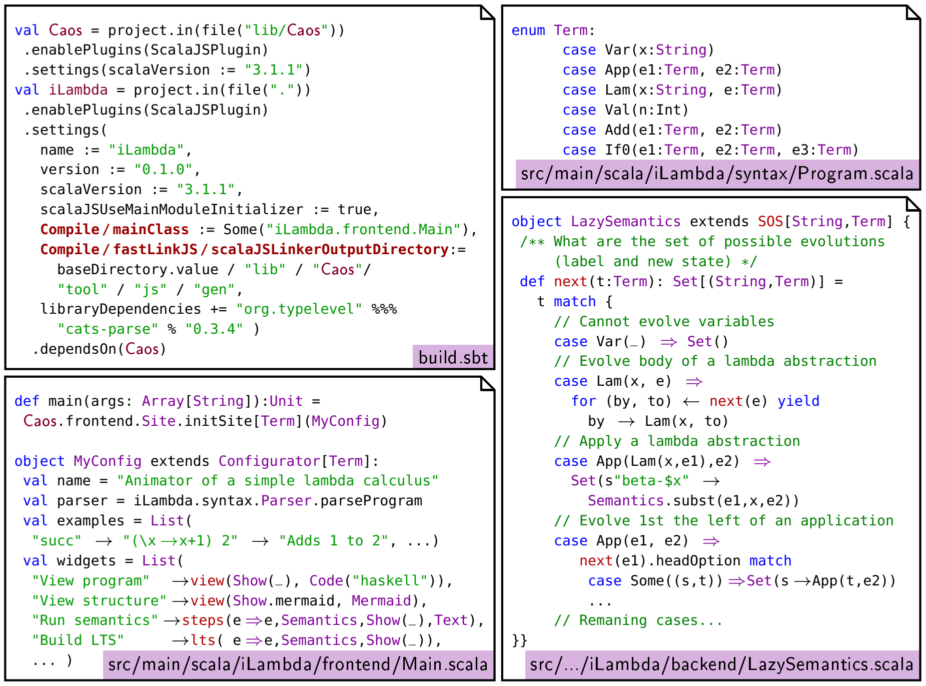
(Analysis.scala), and compiles a collection of widgets that use these functions using special constructors (Configuration.scala). Compiling this configuration yields a JS file used by a provided HTML file. The Configuration is an object that extends an associated class in Caos and holds: the **name** of the language and the website; the **parser** for the language; a list of **examples**, each as a triple (name, program, description); and a list of **widgets** using the provided constructors [17].

Tool demonstration with the iLambda language

We provide a short demonstration on how to use Caos; an expanded version can be found in the companion report [17] and the video tutorial [18]. In this demonstration we implement a lambda-calculus language with integers (`iLambda`); the full source-code can be found in `https://github.com/arcalab/lambda-caos`.

This project requires JVM (>=1.8) and SBT (Scala Builder Tool) to compile, and a web-browser to execute. The top folder should contain the following files:

– `build.sbt` – is the main configuration file of the project (top-left of Fig. 4);
– `project/plugins.sbt` – describes the plug-in to compile to JS, in our case with `addSbtPlugin("org.scala-js" % "sbt-scalajs" % "1.7.1")`;
– `lib/`Caos – includes all Caos files, as-is in its git repository; and
– `src/main/scala/iLambda` – includes all the source-code of our project.



```
val Caos = project.in(file("lib/Caos"))
 .enablePlugins(ScalaJSPlugin)
 .settings(scalaVersion := "3.1.1")
val iLambda = project.in(file("."))
 .enablePlugins(ScalaJSPlugin)
 .settings(
   name := "iLambda",
   version := "0.1.0",
   scalaVersion := "3.1.1",
   scalaJSUseMainModuleInitializer := true,
   Compile / mainClass := Some("iLambda.frontend.Main"),
   Compile / fastLinkJS / scalaJSLinkerOutputDirectory :=
     baseDirectory.value / "lib" / "Caos"/
     "tool" / "js" / "gen",
   libraryDependencies += "org.typelevel" %%%
     "cats-parse" % "0.3.4" )
  .dependsOn(Caos)
```
build.sbt

```
enum Term:
      case Var(x:String)
      case App(e1:Term, e2:Term)
      case Lam(x:String, e:Term)
      case Val(n:Int)
      case Add(e1:Term, e2:Term)
      case If0(e1:Term, e2:Term, e3:Term)
```
src/main/scala/iLambda/syntax/Program.scala

```
def main(args: Array[String]):Unit =
 Caos.frontend.Site.initSite[Term](MyConfig)

object MyConfig extends Configurator[Term]:
 val name = "Animator of a simple lambda calculus"
 val parser = iLambda.syntax.Parser.parseProgram
 val examples = List(
  "succ" → "(\x →x+1) 2" → "Adds 1 to 2", ...)
 val widgets = List(
  "View program"   →view(Show(_), Code("haskell")),
  "View structure" →view(Show.mermaid, Mermaid),
  "Run semantics" →steps(e ⇒e,Semantics,Show(_),Text),
  "Build LTS"      →lts( e ⇒e,Semantics,Show(_)),
  ... )
```
src/main/scala/iLambda/frontend/Main.scala

```
object LazySemantics extends SOS[String,Term] {
 /** What are the set of possible evolutions
     (label and new state) */
 def next(t:Term): Set[(String,Term)] =
   t match {
     // Cannot evolve variables
     case Var(_) ⇒ Set()
     // Evolve body of a lambda abstraction
     case Lam(x, e) ⇒
       for (by, to) ← next(e) yield
         by → Lam(x, to)
     // Apply a lambda abstraction
     case App(Lam(x,e1),e2) ⇒
       Set(s"beta-$x" →
         Semantics.subst(e1,x,e2))
     // Evolve 1st the left of an application
     case App(e1, e2) ⇒
        next(e1).headOption match
         case Some((s,t)) ⇒Set(s →App(t,e2))
         ...
     // Remaning cases...
}}
```
src/.../iLambda/backend/LazySemantics.scala

Fig. 4. Snippets of code and configurations used in the `iLambda` project

Figure 4 presents 4 snippets from the `iLambda` project. The `build.sbt` configures the compilation process, including the main class to be compiled and the target folder to place the compiled JS, marked in bold. The project is compiled by the command line instruction "`sbt fastLinkJS`". The `Program.scala` defines the internal data structure, which represents our lambda terms produced by our parser in `src/main/scala/iLambda/syntax/Parser`. The `Main.scala` provides the `Configuration` object mentioned above, and the `LazySemantics` exemplifies the definition of an SOS semantics. SOS semantics are specified by extending a `SOS[Act,State]` class providing a function `next(s:State): Set[(Act,State)]` that, given a `State s`, returns a set of new states labelled by an `Act`ion. These instances can be animated, compared, or combined using provided widget constructors. For example, `lts(e=>e,LazySemantics,Show(_))` builds the `LTS`, where `e=>e` states that the initial state is the original lambda term, `LazySemantics` defines the semantics, and `Show(_)` defines how to visualise states (which are lambda terms).

5 Conclusions and lessons learned

This paper follows a *computer-aided design approach for formal methods* by means of Caos, introducing its toolset and sharing experiences of its application to develop operational semantics of different systems. During the development of new structures and operational semantics, the Caos toolset provided support to quickly view, simulate, and compare different design choices. We were able to identify problems and solutions with a small investment of time in tool development. We further claim that the Caos toolset is reusable, provides intuitive outputs, and is expressive by using a general programming language. By using standard HTML and CSS, the resulting websites can be easily customisable.

Currently we consider two possible improvements. On one hand, to support a lightweight server (inspired in ReoLive [6] but using, e.g., https://http4s.org) that could be used to delegate heavier tasks, such as the usage of a model-checker. On the other hand, to support the parser development with tools such as https://antlr.org instead of using parsing combinators. All tools are available as open-source, and we welcome any feedback, contribution, or sharing of experiences.

Acknowledgments This work was supported by the CISTER Research Unit (UIDP/UIDB/04234/2020), financed by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology) and by project PTDC/CCI-COM/4280/2021 financed by national funds through FCT. It is also a result of the work developed under projects and Route 25 (ref. TRB/2022/00061 - C645463824-00000063) funded by the EU/Next Generation, within the Recovery and Resilience Plan (RRP); and project VALU3S (ECSEL/0016/2019 - JU grant nr. 876852) financed by national funds through FCT and European funds through the EU ECSEL JU. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey - Disclaimer: This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

References

1. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 Toolset for Analysing Concurrent Systems. In: Vojnar, T., Zhang, L. (eds.) TACAS. LNCS, vol. 11428, pp. 21–39. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_2
2. Castellani, I., Dezani-Ciancaglini, M., Giannini, P.: Event structure semantics for multiparty sessions. In: Boreale, M., Corradini, F., Loreti, M., Pugliese, R. (eds.) Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday. Lecture Notes in Computer Science, vol. 11665, pp. 340–363. Springer (2019). https://doi.org/10.1007/978-3-030-21485-2_19, https://doi.org/10.1007/978-3-030-21485-2_19
3. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: The maude 2.0 system. In: Nieuwenhuis, R. (ed.) Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2706, pp. 76–87. Springer (2003). https://doi.org/10.1007/3-540-44881-0_7, https://doi.org/10.1007/3-540-44881-0_7
4. Cledou, G., Edixhoven, L., Jongmans, S.S., Proença, J.: Api generation for multiparty session types, revisited and revised using scala 3. In: Ali, K., Vitek, J. (eds.) 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany. LIPIcs, vol. 222, pp. 27:1–27:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ECOOP.2022.27, https://doi.org/10.4230/LIPIcs.ECOOP.2022.27
5. Cledou, G., Proença, J., Sputh, B.H.C., Verhulst, E.: Hubs for virtuosonext: Online verification of real-time coordinators. Science of Computer Programming **203**, 102566 (2021). https://doi.org/10.1016/j.scico.2020.102566, https://doi.org/10.1016/j.scico.2020.102566
6. Cruz, R., Proença, J.: Reolive: Analysing connectors in your browser. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11176, pp. 336–350. Springer (2018). https://doi.org/10.1007/978-3-030-04771-9_25, https://doi.org/10.1007/978-3-030-04771-9_25
7. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal SMC tutorial. STTT **17**(4), 397–415 (Aug 2015). https://doi.org/10.1007/s10009-014-0361-y, https://doi.org/10.1007/s10009-014-0361-y
8. Edixhoven, L., Jongmans, S.S.: Realisability of branching pomsets. In: Tapia Tarifa, S.L., Proença, J. (eds.) Formal Aspects of Component Software - 18th International Conference, FACS 2022, Virtual Event, November 10-11, 2022, Revised Selected Papers. Lecture Notes in Computer Science, Springer (2022), to appear
9. Edixhoven, L., Jongmans, S.S., Proença, J., Cledou, G.: Branching pomsets for choreographies. In: Aubert, C., Giusto, C.D., Safina, L., Scalas, A. (eds.) Proceedings 15th Interaction and Concurrency Experience, ICE 2022, Lucca, Italy, 17th June 2022. EPTCS, vol. 365, pp. 37–52 (2022). https://doi.org/10.4204/EPTCS.365.3
10. Felleisen, M., Findler, R.B., Flatt, M.: Semantics Engineering with PLT Redex. MIT Press (2009), http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11885

11. Flatt, M.: Creating languages in racket. Commun. ACM **55**(1), 48–56 (2012). https://doi.org/10.1145/2063176.2063195, https://doi.org/10.1145/2063176.2063195
12. Garavel, H., ter Beek, M.H., van de Pol, J.: The 2020 expert survey on formal methods. In: ter Beek, M.H., Nickovic, D. (eds.) Formal Methods for Industrial Critical Systems - 25th International Conference, FMICS 2020, Vienna, Austria, September 2-3, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12327, pp. 3–69. Springer (2020). https://doi.org/10.1007/978-3-030-58298-2_1, https://doi.org/10.1007/978-3-030-58298-2_1
13. Goncharov, S., Neves, R., Proença, J.: Implementing hybrid semantics: From functional to imperative. In: Pun, K.I., da Silva Simão, A., Stolz, V. (eds.) Theoretical Aspects of Computing - ICTAC 2020 - 17th International Colloquium, Macau S.A.R., China, October 31 - November 30, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12545. Springer (2020)
14. Jongmans, S.S., Proença, J.: St4mp: A blueprint of multiparty session typing for multilingual programming. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation - 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2022, Rhodes, Greece, October 24-28, 2022, Proceedings. Lecture Notes in Computer Science, Springer (2022), to appear
15. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theor. Comput. Sci. **13**, 85–108 (1981). https://doi.org/10.1016/0304-3975(81)90112-2, https://doi.org/10.1016/0304-3975(81)90112-2
16. Politz, J.G., Lerner, B.S., Porncharoenwase, S., Krishnamurthi, S.: Event loops as first-class values: A case study in pedagogic language design. Art Sci. Eng. Program. **3**(3), 11 (2019). https://doi.org/10.22152/programming-journal.org/2019/3/11, https://doi.org/10.22152/programming-journal.org/2019/3/11
17. Proença, J., Edixhoven, L.: Caos: A reusable scala web animator of operational semantics (extended with hands-on tutorial). CoRR **abs/2304.14901** (2023). https://doi.org/10.48550/arXiv.2304.14901, https://arxiv.org/abs/2304.14901
18. Proença, J., Edixhoven, L.: Demonstration video of Caos: A reusable scala web animator of operational semantics. CoRR (April 2023). https://doi.org/10.5281/zenodo.7876059, https://zenodo.org/record/7876059