Science of Computer Programming 240 (2025) 103222

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Original Software Publication

The CAOS framework for Scala: Computer-aided design of SOS

José Proença [a,*], Luc Edixhoven [b]

[a] CISTER and Faculty of Sciences of the University of Porto, Portugal
[b] Open University (Heerlen) and CWI (Amsterdam), Netherlands

ARTICLE INFO

Keywords:
Visualisation
Structural operational semantics
Formal methods
Web-frontend
Pedagogical tools

ABSTRACT

We present Caos: 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 where 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.
More concretely, Caos supports the quick creation of interactive websites that help the end-users better understand a new language, structure, or analysis. End-users can be research colleagues trying to understand a companion paper or students learning about a new simple language or operational semantics. We include a list of open-source projects with a web frontend supported by Caos that are used both in research and teaching contexts.

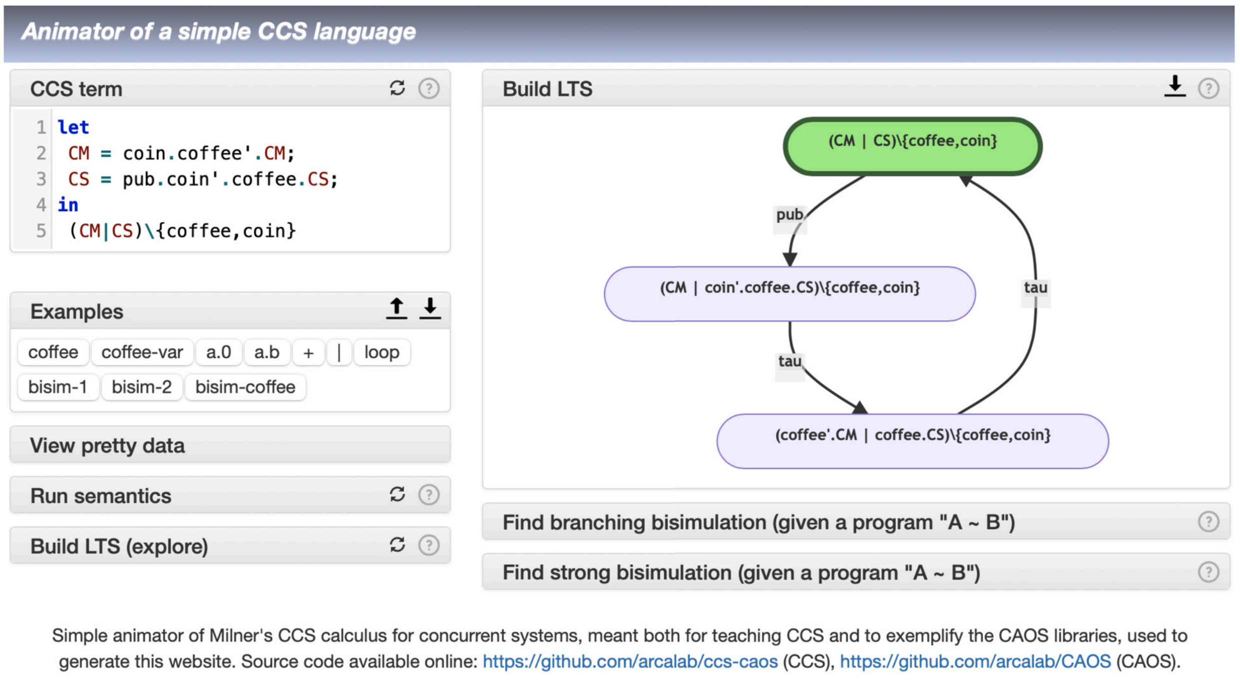
Code metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0.0
C2	Permanent link to code/repository used for this code version	https://github.com/arcalab/CAOS/releases/tag/v1.0.0
C3	Permanent link to Reproducible Capsule	https://github.com/arcalab/ccs-caos/releases/tag/v1.0.0
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Scala & JavaScript
C7	Compilation requirements, operating environments and dependencies	Scala building tools (sbt) & Java Runtime Environment (JRE) $\geq$ 1.8
C8	If available, link to developer documentation/manual	https://arxiv.org/abs/2304.14901 https://youtu.be/Xcfn3zqpubw
C9	Support email for questions	`jose@proenca.org`

* Corresponding author.
E-mail addresses: jose.proenca@fc.up.pt (J. Proença), led@ou.nl (L. Edixhoven).

https://doi.org/10.1016/j.scico.2024.103222
Received 18 December 2023; Received in revised form 19 August 2024; Accepted 14 October 2024

Available online 18 October 2024


Animator of a simple CCS language
CCS term
Examples
coffee
coffee-var
a.0
a.b
+
|
loop
bisim-1
bisim-2
bisim-coffee
View pretty data
Run semantics
Build LTS (explore)
Build LTS
(CM | CS)\{coffee,coin}
pub
(CM | coin'.coffee.CS)\{coffee,coin}
tau
tau
(coffee'.CM | coffee.CS)\{coffee,coin}
Find branching bisimulation (given a program "A ~ B")
Find strong bisimulation (given a program "A ~ B")

```
let
 CM = coin.coffee'.CM;
 CS = pub.coin'.coffee.CS;
in
 (CM|CS)\{coffee,coin}
```

Simple animator of Milner's CCS calculus for concurrent systems, meant both for teaching CCS and to exemplify the CAOS libraries, used to generate this website. Source code available online: https://github.com/arcalab/ccs-caos (CCS), https://github.com/arcalab/CAOS (CAOS).

Fig. 1. Screenshot of a generated web-frontend by CAOS for a concurrent calculus.

1. Motivation and significance

Designing and explaining 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 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*. By contrast, this tool – Caos (*computer-aided design of SOS for formal methods*) – supports *the methodology* of designing and explaining formal methods, with special emphasis on Structural Operational Semantics (SOS). Caos combines, simplifies, extends, and bundles in a single software artefact a collection of functions that we have been using in the context of several research projects; therefore allowing other researchers and teachers to re-use and adapt functionalities that we have found useful in our own work. Source code, links to documentation, and a compilation of examples can be found at `https://github.com/arcalab/caos`.

Caos has been introduced in a companion conference paper [1], allowing users to produce interactive websites by providing an implementation in Scala of (1) a data structure with an associated parser, and (2) a set of analyses, some parameterised by a *next* function that describes state reductions. It further provides support for the Mermaid language (a popular Markdown-like language for diagrams)[1] to generate visual diagrams, and extra functions to compute equivalences of labelled transition systems. Related tools include Maude [2], Racket [3], and Pyret [4], as explained in the tutorial [5] and companion paper [1]. To the best of our knowledge, Caos is the first framework that provides support to design models and operational semantics using a general programming language such as Scala.

This paper includes a new example of a web frontend produced by Caos to analyse a simple variation of Milner's CCS [6] in Section 2. Other motivating examples of Caos used in the context of teaching (for a simple while-language) and research (to provide insights over the semantics of a choreographic language) have been described in the companion paper [1], and a step-by-step tutorial using a lambda-calculus has been included in the extended technical report [5]. The improvements of the toolset since the companion paper are described in Section 4.

2. Illustrative examples

Fig. 1 presents a screenshot of a `ccs` project that analyses a simple variant of Milner's CCS language [6], accessible at `https://lmf.di.uminho.pt/ccs-caos`, developed for illustrative purposes. This screenshot includes an input widget in the upper-left corner, a widget with examples immediately below, and six other widgets customised for CCS, most of them collapsed except for "Build LTS":

[1] See `https://mermaid.js.org`.

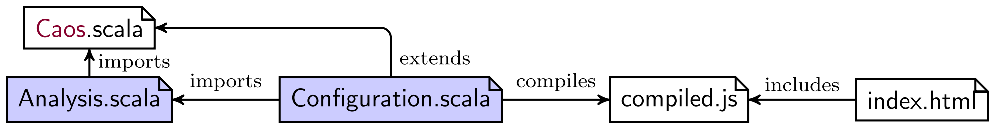


Fig. 2. Architecture of a Scala project that uses Caos.

- "View pretty data" produces a re-indented version of the formula;
- "Run semantics" interactively asks which SOS rule should be applied;
- "Build LTS (explore)" builds the same LTS as "Build LTS", now interactively, initially presenting only the initial state and its successors – clicking any of these successors adds its own successors; and
- the last two widgets search for bisimulations whenever the input term is written as a pair of terms separated by ~, providing an explanation whenever the search fails.

The source code of this and many other examples can be found at CAOS' website `https://github.com/arcalab/CAOS`. These examples include:

- projects used to produce didactic content to teach university students, by analysing and animating semantics for languages such as CCS [6], a simple while-language, a lambda-calculus with integers [5], and a pseudo-assembly language Apoo [7].
- projects used as tool-companions of research papers, including analysis of *multi-party session types* [8], *choreographic languages* [9], *variations of pomsets* [10–12], and *team automata* [13].

3. Software framework

This section describes the Caos architecture and its functionality.

3.1. Software architecture

Fig. 2 depicts the structure of a typical Scala project that uses Caos. Projects using Caos require a Java Virtual Machine and SBT (Scala Builder Tool) to compile, and a web browser to execute. The user provides the highlighted documents: data structures for the input language with functions to parse this language and to compute analyses (Analysis.scala), and a description of the desired widgets that use these functions using special constructors (Configuration.scala). More specifically, 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 [5]. This configuration file is a plain Scala file that can import any function in its scope, such as a parsing function defined within the Scala project. Compiling it yields a JavaScript (JS) file used by an HTML file included in the Caos library.

3.2. Widgets provided by CAOS

The web frontend produced by Caos always includes an input widget, where the user can provide programs to be analysed; and a widget with a list of examples, each as a button that triggers the analysis of a pre-defined program. The rest of the frontend consists of a list of custom widgets, defined using the provided constructors, which can be categorised as follows. The full list of widgets and constructors can be found online [5].

- **Visualisation** of a string produced from the program, representing plain text, code, or a mermaid diagram;
- **Execution** of a model of the program, given a `next` function that evolves the program, presented to the user either in a step-wise manner (interactive) or as a single state diagram with all reachable states;
- **Equivalence** of two programs using bisimilarity or trace equivalence;
- **Check** for errors or warnings in a program, aborting all analysis when errors are found.

Every widget can be either *collapsed* or *expanded*; clicking on the widget title alternates between these states. A widget reloads its analysis when it is expanded, and re-evaluating the input reloads all expanded widgets.

3.3. Sample code snippets analysis

Fig. 3 presents three snippets from our `ccs` project that analyses a simple variant of Milner's CCS language [6]. `Program.scala` defines the internal data structure, which represents the CCS terms produced by our parser in `ccs/syntax/Parser`, and `Semantics.scala` exemplifies the definition of an SOS semantics. `Config.scala` provides the Configuration object with the concrete layout of the webpage, cf. Section 3.1. SOS semantics are specified by extending a `SOS[Act,State]` class. In this case the type variable `Act` is unified with `Action` and `State` with `System`, which are our concrete data types for actions and states of our semantics. These instances can be animated, compared, or combined using provided widget constructors. For example,

```
case class System(
    defs: Map[String,Term],
    main:Term, ...)
enum Term:
  case End
  case Proc(p:String)
  case Prefix(act:Action,t:Term)
  ...
```
src/main/scala/ccs/syntax/Program.scala

```
object Semantics extends SOS[Action,System] {
 /** What are the set of possible evolutions
     (label and new state) */
 def next(st:System): Set[(Action,System)] =
   st.main match {
    case End           ⇒ Set()
    case Prefix(act,t) ⇒ Set(act → st(t))
    case Proc(p)    ⇒ next(...st.defs(p)...)
    case Choice(t1,t2) ⇒ next(st(t1)) ++
                         next(st(t2))
    case Par(t1, t2)   ⇒ //...
    case Restr(t,r)    ⇒ //..
```
src/.../ccs/backend/Semantics.scala

```
object Config extends Configurator[System]:
 val name =
  "Animator of a simple CCS calculus"
 val parser = ccs.syntax.Parser.parseProgram
 val examples = List(
  "coffee"  → "let\n CM = coin.coff...",...
 val widgets = List(
  "View pretty data" →
    view(Show(_),Code("haskell")),
  "Run semantics" →
    steps(e⇒e,Semantics,...,Text),
  "Build LTS" →
    lts(e⇒e,Semantics,...),
  "Find branching bisimulation (...)" →
    compareBranchBisim(
      e ⇒ ...,  // left  term
      e ⇒ ...,  // right term
      ...),
  ...)
 val documentation: Documentation = List(
    "Build LTS" →
      "(helper)" → "(html explanation)",
    ...)
```
src/main/scala/ccs/frontend/Config.scala

Fig. 3. Snippets of code and configurations used in the `ccs` project.

`lts(e⇒e,Semantics,Show.justTerm,Show(_))` builds the Labelled Transition System (LTS), where `e=>e` states that the initial state is the original CCS term, `Semantics` defines the semantics, and `Show.justTerm` and `Show(_)` convert states (CCS terms) and actions to a string representation, respectively.

4. Impact

Caos has been used to produce several companion tools of scientific publications and to produce didactic content to teach university students, as explained in the previous section. Since the publication of the companion conference paper, Caos has been extended to support the inclusion of documentation in the generated frontends. This includes new buttons ? and explanatory messages that guide the final user of the generated websites. Furthermore, we introduced a new widget to iteratively unfold the state-space of a given set of SOS rules (as suggested at the companion conference), created the new example based on Milner's CCS [6] used in this paper, provided a new approach based on giter8 templates[2] to bootstrap a project that uses Caos, thus reducing the effort for new users, and extended the online documentation in Caos' GitHub repository.

We are currently collaborating with researchers involved in the MSc supervision of projects that plan to use Caos, and have been contacted by university teachers who have used instances of Caos to illustrate concepts in their lessons. Furthermore, all tools are available as open-source, and we welcome any feedback, contribution, or sharing of experiences.

5. Conclusions

This paper presents Caos, a Scala framework that supports a *computer-aided design approach for formal methods*, introducing its toolset and sharing its applications in the context of companion prototype research tools and of teaching in higher education. During the development of new structures and operational semantics, the Caos toolset provided us support to quickly view, simulate, and compare different design choices. More recently, we made an explicit effort to open and facilitate the usage of Caos by others, both researchers and teachers, receiving positive and constructive feedback. New additions include the preparation of richer documentation, and support to attach explanations to the generated frontends. We further claim that the Caos toolset is reusable, provides intuitive outputs, and is expressive by using Scala – a general programming language. By using standard HTML and CSS, the resulting websites can be easily customised.

6. Future plans

Currently, we consider two possible improvements. On one hand, to support a lightweight server (inspired in ReoLive [14] 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 parser combinators.

[2] See https://www.foundweekends.org/giter8/.

CRediT authorship contribution statement

José Proença: Writing – review & editing, Writing – original draft, Visualization, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Luc Edixhoven:** Writing – review & editing, Validation, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Jose Proenca reports financial support was provided by Portuguese Foundation for Science and Technology. Jose Proenca reports a relationship with Portuguese Foundation for Science and Technology that includes: funding grants and travel reimbursement. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

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 Ibex (ref. PTDC/CCI-COM/4280/2021) financed by national funds through FCT. It is also a result of the work developed under project Route 25 (ref. TRB/2022/00061 - C645463824-00000063) funded by the Next Generation EU, within the Recovery and Resilience Plan (RRP).

References

[1] J. Proença, L. Edixhoven, Caos: a reusable scala web animator of operational semantics, in: S. Jongmans, A. Lopes (Eds.), Coordination Models and Languages - 25th IFIP WG 6.1 International Conference, COORDINATION 2023, Held as Part of the 18th International Federated Conference on Distributed Computing Techniques, DisCoTec 2023, Lisbon, Portugal, June 19-23, 2023, Proceedings, in: Lecture Notes in Computer Science, vol. 13908, Springer, 2023, pp. 163–171.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C.L. Talcott, The maude 2.0 system, in: R. Nieuwenhuis (Ed.), Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings, in: Lecture Notes in Computer Science, vol. 2706, Springer, 2003, pp. 76–87.

[3] M. Flatt, Creating languages in racket, Commun. ACM 55 (1) (2012) 48–56, https://doi.org/10.1145/2063176.2063195.

[4] J.G. Politz, B.S. Lerner, S. Porncharoenwase, S. Krishnamurthi, Event loops as first-class values: a case study in pedagogic language design, Art Sci. Eng. Program. 3 (3) (2019) 11, https://doi.org/10.22152/programming-journal.org/2019/3/11.

[5] J. Proença, L. Edixhoven, Caos: a reusable scala web animator of operational semantics (extended with hands-on tutorial), CoRR, arXiv:2304.14901, 2023, https://doi.org/10.48550/arXiv.2304.14901.

[6] R. Milner, A Calculus of Communicating Systems, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[7] R. Reis, N. Moreira, Apoo: an environment for a first course in assembly language programming, ACM SIGCSE Bull. 33 (4) (2001) 43–47, https://doi.org/10.1145/572139.572168.

[8] S. Jongmans, J. Proença, ST4MP: a blueprint of multiparty session typing for multilingual programming, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part I, in: Lecture Notes in Computer Science, vol. 13701, Springer, 2022, pp. 460–478.

[9] G. Cledou, L. Edixhoven, S. Jongmans, J. Proença, API generation for multiparty session types, revisited and revised using scala 3, in: K. Ali, J. Vitek (Eds.), 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany, in: LIPIcs, vol. 222, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 27:1–27:28.

[10] L. Edixhoven, S.-S. Jongmans, J. Proença, G. Cledou, Branching pomsets for choreographies, in: C. Aubert, C.D. Giusto, L. Safina, A. Scalas (Eds.), Proceedings 15th Interaction and Concurrency Experience, ICE 2022, Lucca, Italy, 17th June 2022, in: EPTCS, vol. 365, 2022, pp. 37–52.

[11] L. Edixhoven, S. Jongmans, Realisability of branching pomsets, in: S.L.T. Tarifa, J. Proença (Eds.), Formal Aspects of Component Software - 18th International Conference, FACS 2022, Virtual Event, November 10-11, 2022, Proceedings, in: Lecture Notes in Computer Science, vol. 13712, Springer, 2022, pp. 185–204.

[12] L. Edixhoven, S.-S. Jongmans, J. Proença, I. Castellani, Branching pomsets: design, expressiveness and applications to choreographies, J. Log. Algebraic Methods Program. 136 (2024) 100919, https://doi.org/10.1016/j.jlamp.2023.100919.

[13] M.H. ter Beek, R. Hennicker, J. Proença, Realisability of global models of interaction, in: E. Ábrahám, C. Dubslaff, S.L.T. Tarifa (Eds.), Theoretical Aspects of Computing - ICTAC 2023 - 20th International Colloquium, Lima, Peru, December 4-8, 2023, Proceedings, in: Lecture Notes in Computer Science, vol. 14446, Springer, 2023, pp. 236–255.

[14] R. Cruz, J. Proença, Reolive: analysing connectors in your browser, in: M. Mazzara, I. Ober, G. Salaün (Eds.), Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 11176, Springer, 2018, pp. 336–350.