arXiv:2304.14901v1 [cs.PL] 28 Apr 2023

Caos：一个可复用的 Scala 操作语义 Web 动画工具

（附实操教程）

José Proença[1]和 Luc Edixhoven[2]

[1] CISTER，ISEP，波尔图理工学院，葡萄牙 pro@isep.ipp.pt[2] 开放大学（海尔伦）和 CWI（阿姆斯特丹），荷兰 led@ou.nl

摘要。本文介绍了Caos：一种用于形式模型结构操作语义的计算机辅助设计方法论和编程框架。该框架包含一组 Scala 库及一套工作流程，用于生成可视化、交互式图表，以动态展示并深入揭示具有操作规则的给定抽象模型的结构和语义。

Caos 采用一种将理论基础与实用工具共同构建的方法，以此替代先构建基础的设计（“工具证明理论的合理性”）或先构建工具的设计（“基础证明实践的合理性”）。Caos的优势在于，开发中的工具可以立即用于自动运行大量且规模可观的示例，从而尽早发现开发中的理论基础中细微的错误、意外结果和未曾预料到的局限。

我们分享了 Caos 的方法论和框架在我们自身教学与研究环境中的两个成功案例，其中分析了一种简单的 while 语言和一种编舞语言，包括它们的操作规则以及这些规则的并发组合。我们还进一步探讨了其他人如何将 Caos 用于自己的分析和 Scala 工具中。

演示视频： https://zenodo.org/record/7876060 和 https://youtu.be/Xcfn3zqpubw

动手教程： 附录 A

1 引言

设计形式化方法可能很困难。形式化方法相关研究的典型挑战包括识别并处理边界情况、发现缺失的假设、找到合适的抽象层次，当然还有证明定理（并将其恰当地分解为引理）。有趣的是，与其他科学学科不同，我们发现本领域绝大多数论文主要关注研究结果而非方法。与之相反，这篇工具论文致力于形式化方法的设计方法论，特别强调结构操作语义（SOS）：我们分享使用计算机辅助设计 SOS 进行形式化方法的经验，并展示由工具集*Computer-Aided Design of SOS for formal methods* 生成的一组示例Caos。源代码和示例汇编可在此处获取：https://github.com/arcalab/caos。我们希望它能够

激励同事们运用我们的方法和工具，并分享他们在方法应用方面的经验，让整个社区受益。

简而言之，在 Caos 中，理论基础和实用工具从一开始就并行构建，彼此协同，作为对更常见的 基础优先设计 （“工具证明理论正确”）或 工具优先设计（“基础证明实践合理”）的替代方案。 Caos 的主要优势在于，正在开发的工具可以立即用于自动运行大量且规模可观的示例，以尽早发现正在构建的理论基础中细微的错误、意外结果和/或未预见的局限性。从本质上说， Caos 中自动化示例的优先地位，类似于现代敏捷软件工程中自动化（单元）测试的优先地位。形式化方法对验证和配套工具的这种需求已得到认可，例如，Garavel 等人在近期一项关于关键系统形式化方法的综述中便指出了这一点 [12]。

该Caos 工具集基于 ReoLive，[3] ReoLive 最初开发为一套在线 Scala 和 JavaScript (JS) 工具，用于分析 Reo 连接器 [6]。目前，它还支持许多与 Reo 无关的扩展 [13,5], ，这些扩展中的通用代码块既可编译为 JS（客户端），也可编译为 Java 二进制文件（服务器端），从而能够将计算任务委托给远程服务器。因此，它逐渐成为一种**单体式**实现，其中为不同的独立扩展编写了许多**重复**代码块，且难以**复用**于不同项目。我们提出的替代方案Caos 工具集旨在解决这些问题，并满足以下要求：

– **R1：** Caos 应使用通用编程语言，以便于采用，并在需要时支持更复杂的后端；– **R2：** 由 Caos 支持的实现所生成的输出应易于执行和使用，无需特定平台或复杂的安装过程；– **R3：** Caos 应易于复用，且Caos 支持的实现应采用模块化设计，并易于通过新增分析进行扩展。

基于这些要求，我们的 Caos 工具集使用 Scala 实现（R1），可编译为 JavaScript，以生成直观且交互式的网站（R2），并包含一个易于扩展的 API，便于其他开发者使用和复用（R3）。使用 Caos 工具集，可以生成类似 图 1 所示的网页。该网页包含一个输入文本框，以及一组小部件，用于展示或动画演示针对输入程序的不同分析，并在适用时利用可能的操作语义。第2节将进一步详细介绍这个示例，其中分析了一种简单的 while 语言（带契约）。第2节，其中分析了一种简单的 while 语言（带契约）。

Caos 专门支持结构操作语义（SOS），可对实现 `next` 和 `accepting` 方法的项进行动画演示、描绘或比较。它还支持为交互组件网络构建结构操作语义，如第3节所述。还有一些类似的方法可支持语言语义的开发，例如以下方法，但它们并未涵盖上述 3 项要求。

Maude 语言和工具集 [3] 侧重于如何指定（1）使用字符序列表示的配置（状态），以及（2）一组可能的重写规则，用于描述配置可以如何修改。它还提供了

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

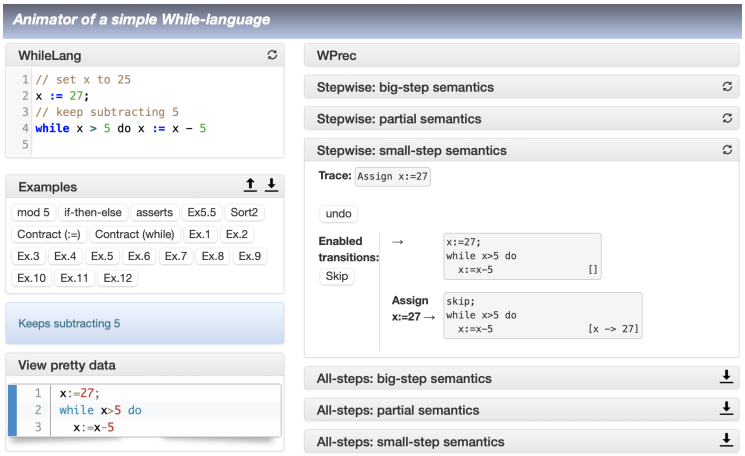


图 1. 用于分析简单 while 语言的网页界面截图，网址为
https://cister-labs.github.io/whilelang-scala/

一组用于促进创建新语法表示法的构造，例如将运算符标记为满足结合律并具有给定的单位元。Maude 包含完善的模型检查器和其他分析工具；其他模型检查器（例如，mCRL2 [1], UPPAAL [7]）也具有带有操作语义的规约语言，这些语言经过有意限制，以提供更好的模型检查支持。这些方法提供了类似的功能，但并不以满足我们的要求为目标。

Racket （及其 DrRacket 图形前端） [11]是一种面向语言的编程语言，即一种旨在用于创建语言的语言。它得到了广泛采用，拥有大量库，还包含一组便于创建新语法表示的构造；这些表示被打包为新语言，从而允许在一个程序中同时存在多种语言，并可在运行时创建新语言。PLT Redex [10] 嵌入于 Racket 中，是一种用于描述和调试操作语义的领域专用语言，它接收语法和归约规则，并支持交互式探索各个项。可以说，Racket 是一种通用语言（R1），尽管其普及程度不如 Java 或 Scala，但它具备支持复用的扩展机制（R3），而且我们认为用它部署产品更困难（R2）。一些**教学语言**，例如

如 Pyret [16], 旨在编译为 JavaScript，并用于教授入门计算机知识，在表现力和性能之间取得平衡。它包含一个强大的运行时，用于向用户隐藏 JavaScript 的一些复杂性和限制。这些语言通常包含可视化库和引-擎，以吸引其目标受众：学习编程概念的学生。这些语言并不具有完全相同的函数式目标，也不使用通用编程语言（R1），但通常可以生成易于运行的代码（R2），并且可扩展（R3）。

Caos 对于熟悉 Scala/Java 的用户尤其有用；对于有使用 Racket 或 Pyret 等语言经验的用户而言，帮助则较小。 **论文结构：** 本文首先介绍我们在教学（Caos ， 第2节）和研究（第3节）环境中使用Caos 的经验，重点介绍使用该工具集可以实现的成果。 第4节 介绍Caos 工具集的结构及其供他人使用的方式；第5节 则对本文作总结。 附录 A提供了如何使用Caos 工具集的分步教程。

2 用例：用于教学的 while 语言

在一门大学课程中，学生学习了自然语义和操作语义，以及如何推导最弱前置条件。作为教师，我们使用一种包含整数的简单 while 语言来介绍这些概念。我们在几天内创建了一个简单的网站，使用 Caos，如 图 1所示，并在一周内不断完善核心小部件。请注意，我们熟悉相关工具，也有一些使用 Scala 编写解析器的经验。学生们使用这个网站进行实验，从而更深入地理解这些概念。

图 1 展示了 Caos编译后的输出：一组小部件，其中始终包含一个输入小部件（此处称为 `WhileLang`）和一组输入程序示例。其他小部件均为定制开发，包括：(1) 对程序生成的字符串进行可视化，可呈现为纯文本、代码或 Mermaid 图（一种流行的类 Markdown 图表描述语言）； [4] 以及 (2) 执行给定的 `next` 函数，该函数会演进程序，可逐步（交互式）呈现，也可作为包含所有可达状态的单个状态图呈现。 Caos还提供了以下小部件：(3)使用双模拟或迹等价比较两个程序行为；以及 (4) 检查程序中的错误或警告。

图 1 展示了源代码（左下方）的可视化，以及使用小步语义进行的逐步演化及其文本表示（右侧），其余小部件则处于折叠状态。这些折叠的小部件使用不同的语义、提供所有步骤的视图，或计算最弱前置条件；折叠时不会进行处理。学生可以借此更好地理解每个时刻可以应用哪些规则，并浏览状态空间。

3 用例：研究中的编排分析

Caos 可用于借助原型工具阐释研究概念。例如，在研究编舞语言时，我们就使用了它。编舞语言描述代理之间可能的交互序列，例如，

`ctr →wrk1:工作 ; ctr →wrk2:工作 ; (wrk1 →ctr:完成 || wrk2 →ctr:完成)`

描绘了这样一个场景：控制器 `ctr` 将一些工作 分派给两名工作者，他们完成后会回复。我们与 Guillermina Cledou 和 Sung-Shik Jongmans 合作，发表了多项由以下工具支持的编排分析：

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

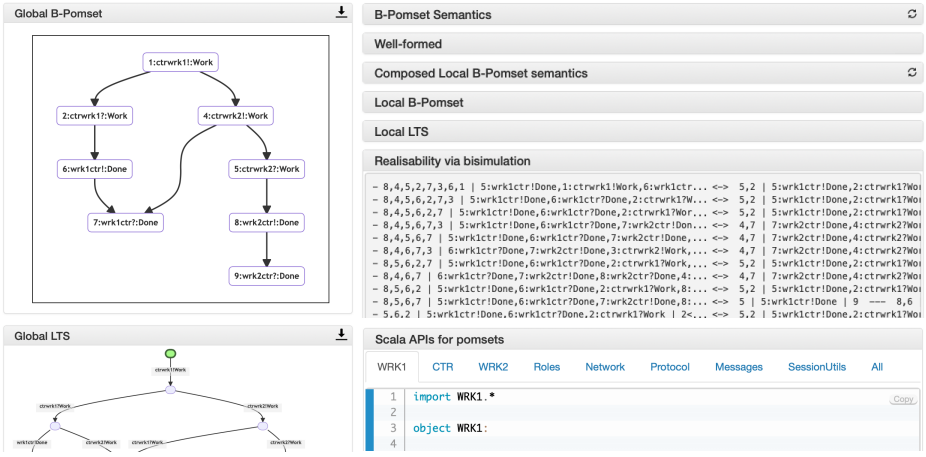


图 2. 对由 Caos 从编排语言生成的分支 pomset 进行分析

基于 Caos的原型，研究如何利用一种新颖的底层数学结构，检测局部代理的行为是否会导出全局行为（即所谓的可实现性） [9,8] (`https://lmf.di.uminho.pt/b-pomset`, 以及如何生成应用程序接口，从而以静态方式保证局部代理遵循其交互协议 [4,14] (`https://lmf.di.uminho.pt/{pompset,st4mp}`)。

我们使用一种基础数学结构为编排赋予语义：分支 pomset [9]（它与事件结构相似 [15,2]）。如图 2所示，借助 Caos 可以实现：**(1)** 可视化偏序多重集结构（左上）；**(2)** 执行一个 pomset（B-Pomset语义）以及将其投影组合到每个相关代理上（组合局部B-Pomset语义）；**(3)** 检查良构性(良构），这是一种新颖的语法检查方法（可靠但不完备），用于检查可实现性；**(4)** 检查可实现性，方法是搜索全局行为与各投影组合行为之间的双模拟（该方法完备但更复杂）（通过双模拟检查可实现性）；以及 **(5)** 生成 *Scala* 代码，并借助相关库在编译时保证局部代理遵循预期协议（右下）。Caos 提供了用于组合局部代理行为以及搜索双模拟的构造器。

搭建每个网站**耗费了一人约半周的工作时间**。在我们的研究过程中，Caos 支持的实现是至关重要的机制，使我们能够尝试编排语言语义、偏序多重集结构以及可实现性分析的多种变体，最终逐步收敛到当前版本。

4 Caos框架

本节介绍 Caos 提供的功能，以及如何使用它来生成动画器，例如 第 2 节和第 3 节中的动画器。有关如何使用Caos 的分步指南是

包含在附录 A中，并使用基于 λ 演算的语言来进行说明。

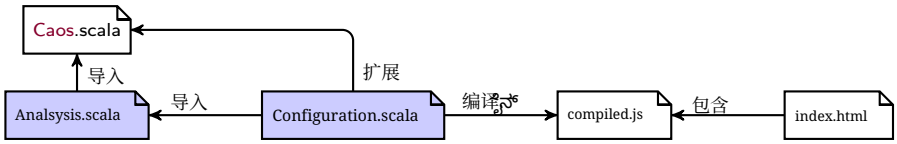


图 3. 使用 Caos 的 Scala 项目架构

图 3 展示了使用 Caos 的典型 Scala 项目的结构。用户需要为输入语言提供数据结构，以及用于解析该语言并执行分析的函数（Analysis.scala），并使用所提供的构造器（Configuration.scala）编译一组调用这些函数的小部件。编译此配置后会生成一个 JS 文件，该文件由所提供的 HTML 文件调用。Configuration 是一个对象，它扩展了 Caos 中与之关联的类，并包含：语言名称和网站；该语言的解析器；示例列表，每个示例都是一个三元组（名称、程序、描述）；以及使用附录 A.4 中所述的所提供构造器的小部件列表。编译配置在专用的 build.sbt 文件中完成，该文件选择 ScalaJS 编译器、要编译的主文件以及最终 JS 文件的位置。用户可以通过扩展一个 `SOS[Act,State]`类来指定操作规则，该类提供一个函数 `next(s:State): Set[(Act,State)]`：给定一个 `State`s，返回一组带有某个 `Act`作标记的新状态。这些实例可以使用所提供的小部件构造器进行动画演示、比较或组合。例如，`lts(makeBPomset,BPomsetSOS,bp=>"")` 构建了图 2 中的 `Global LTS`，其中`makeBPomset` 创建偏序多重集结构，`BPomsetSOS`扩展了 `SOS` 类，而`bp=>""` 隐藏了状态名称。

5 结论与经验教训

本文采用形式化方法的计算机辅助设计方法，借助Caos介绍相关工具集，并分享将其应用于开发不同系统的操作语义的经验。在开发新结构和操作语义的过程中，Caos 工具集为快速查看、模拟和比较不同设计选择提供了支持。

我们只需投入少量时间开发工具，就能高效地找出可能的问题及其解决方案。我们进一步认为，Caos工具集可重复使用，输出直观，并且通过使用通用编程语言而具有良好的表达能力。借助标准的 HTML 和 CSS，生成的网站也可以轻松自定义。

目前我们正在考虑两种可能的改进。一方面，支持一个轻量级服务器（灵感来自 ReoLive [6]，但使用例如 `https:`

//http4s.org），以便将模型检查器等较重的任务委托给它。另一方面，为了更好地支持解析器开发，可以借助同时支持 JavaScript 和 Java 后端的工具 https://antlr.org ，而不是使用解析组合器。所有工具均以开源形式提供，我们欢迎任何反馈、贡献或经验分享。

参考文献

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.: 用于分析并发系统的 mCRL2 工具集。载于：Vojnar, T., Z hang, L.（编）TACAS。《计算机科学讲义》，第 11428 卷，第 21 39 2019 10 1007 978 3 030 17465 1 _2 – 页。Springer 2（）。https://doi.org/./----. Castellani, I., Dezani-Ciancaglini, M., Giannini, P.: 多方会话的事件结构语义。载于：Boreale, M., Corradini, F., Loreti, M., Pugliese, R.（编）《并发与分布式编程的模型、语言和工具——纪念 Rocco De Nicola 65 岁诞辰文集》。《计算机科学讲义》，第 11665 卷，第 340–363 页。Springer（2019）。https://doi.org/10.1007/978-3-030-21485-2_19, doi.org/10.1007/978-3-030-21485-2_19 https://3 . Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: Maude 2.0 系统。载于：Nieuwenhuius, R.（编）《重写技术与应用》，第14届国际会议，RTA 2003，西班牙瓦伦西亚，2003年6月9日至11日，会议论文集。《计算机科学讲义》，第 2706 卷，第 76–87 页。Springer（2003）。https://doi.org/10.1007/3-540-44881-0_7, https://doi.org/10.1007/3-540-44881-0_74. Cledou, G., Edixhoven, L., Jongmans, S.S., Proença, J.: 多方会话类型的 API 生成：基于 Scala 3 的重新审视与修订。载于：Ali, K., Vitek, J.（编）第36届欧洲面向对象编程会议，ECOOP 2022，2022年6月6日至10日，德国柏林。LIPIcs，第 222 卷，第 27:1–27:28 页。Schloss Dagstuhl - Leibniz-Zentrum für Informatik（2022）。https://doi.org/10.4230/LIPIcs.ECOOP.2022.27, https://doi.org/10.4230/LIPIcs.ECOOP.2022.275. Cledou, G., Proença, J., Sputh, B.H.C., Verhulst, E.: virtuosonext 枢纽：实时协调器的在线验证。计算机编程科学 **203**, 102566 2021 10 1016 2020 102566 https://doi.org/10. （）。1016/j.scico.2020.102566 https://doi.org/./j.scico.., 6 . Cruz, R., Proença, J.: ReoLive：在浏览器中分析连接器。载于：Mazzara, M., Ober, I., Salaün, G.（编）《软件技术：应用与基础——STAF 2018》第29届会议，2018年，25 11176，专题研讨会，法国图卢兹，6月，修订版精选论文。《计算机科学讲义》，第 pp. 336–350 页。Springer（2018）。https://doi.org/10.1007/978-3-030-04771-9_25, https://doi.org/10.1007/978-3-030-04771-9_257 . David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: UPPAAL SMC 教程。STTT**17**(4), 397–415（2015年8月）。https://doi.org/10.1007/s10009-014-0361 https://doi.org/10.1007/s10009-014-0361-y-y, . Edixhoven, L., Jongmans, S.S.: 分支 pomset 的可实现性。载于：Tapia Tarifa, S.L., Proença, J.（编）《组件软件的形式化方面——第18届国际会议，FACS 2022，线上会议，2022年11月10日至11日，修订版精选论文》。《计算机科学讲义》，Springer（2022），即将出版

8

9. Edixhoven, L., Jongmans, S.S., Proença, J., Cledou, G.: 编排的分支 pomset。载于：Aubert, C., Giusto, C.D., Safina, L., Scalas, A.（编）第15届交互与并发体验会议论文集，ICE 2022，意大利卢卡，2022年6月17日。EPTCS，第 365 卷，第 37–52 页（2022）。https://doi.org/103 104204365./EPTCS.. Felleisen, M., Findler, R.B., Flatt, M.: 使用 PLT Redex 进行语义工程。麻省理工学院出版社（2009），http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=1188511. Flatt, M.: 在 Racket 中创建语言。《ACM 通讯》**55**（1），48–56（2012）。https://doi.org/10.1145/2063176.2063195， https://doi.org/10.1145/2063176.206319512. Garavel, H., ter Beek, M.H., van de Pol, J.: 2020 年形式化方法专家调查。载于：ter Beek, M.H., Nickovic, D.（编）《工业关键系统形式化方法——第25届国际会议，FMICS 2020，奥地利维也纳，2020年9月2日至3日，会议论文集》。《计算机科学讲义》，第 12327 卷，第 3–69 页。Springer（2020）。https://doi.org/10.1007/978-3-030-58298-2_1， https://doi.org/10.1007/978-3-030-58298-2_113. Goncharov, S., Neves, R., Proença, J.: 实现混合语义：从函数式到命令式。载于：Pun, K.I., da Silva Simão, A., Stolz, V.（编）《计算理论方面——ICTAC 2020——第17届国际研讨会，中国澳门特别行政区，2020年10月31日至11月30日，会议论文集》。《计算机科学讲义》，第 12545 卷。Springer（2020）14. Jongmans, S.S., Proença, J.: St4mp：面向多语言编程的多方会话类型蓝图。载于：Margaria, T., Steffen, B.（编）《利用形式化方法、验证与确认——第10届利用形式化方法国际研讨会，ISoLA 2022，希腊罗得岛，2022年10月24日至28日，会议论文集》。《计算机科学讲义》，Springer（2022），即将出版 15. Nielsen, M., Plotkin, G.D., Winskel, G.: 佩特里网、事件结构与域，第一部分。《理论计算机科学》**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.: 作为一等值的事件循环：教学型语言设计案例研究。《艺术、科学与工程编程》**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

工具演示：为 iLambdalanguage 制作动画

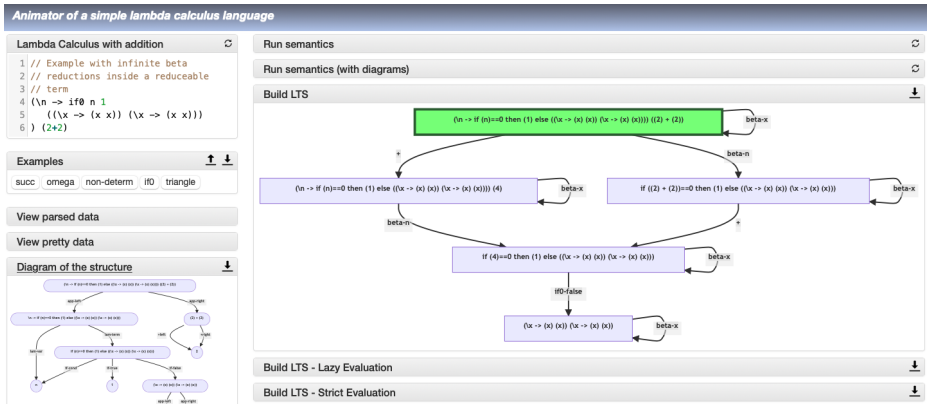


图 4. `iLambda` 语言的生成网站

本附录演示如何在带整数的 lambda 演算语言中集成和使用 Caos ， 我们将这种语言称为 `iLambda`，并采用不同的求值策略。此示例的演示视频可在以下位置找到：`https://zenodo.org/record/7876060`。生成的网页如 图 4所示， 也可在线访问：

– **完整源代码：** `https://github.com/arcalab/lambda-caos`– **生成的网站：** `http://lmf.di.uminho.pt/lambda-caos/`

本演示由逐步指南构成，说明如何复现此 `iLambda` Scala 项目的创建过程。

A.1 要求

你将使用 Scala 编写程序，并将代码编译为 JavaScript（JS）， 但无需安装 Scala 或 JavaScript 引擎。你需要准备：

– JVM（1.8+）： Java 虚拟机（无需安装 Java 编译器）； – SBT：用于 Scala 项目的构建工具，类似于 Java 的 Maven； – 支持 JavaScript 的网络浏览器，例如 Chrome 或 Firefox； – 你喜欢的 Scala 编辑器（文本编辑器即可）。

A.2 引导与编译

你将创建一个 SBT 可识别的文件结构， 下载Caos，并编辑两个 SBT 配置文件。我们将演示项目命名为 `iLambda`，文件结构如下：

– `build.sbt` – 项目的主配置文件；– `project/plugins.sbt` – 在此引入用于编译为 JavaScript 的 ScalaJS 插件；– `lib/Caos` – 在此按原样包含 Git 仓库中的所有Caos 文件；– `src/main/scala/iLambda` – 在此包含项目的所有源代码。– `src/main/scala/iLambda/frontend/Main.scala` – 在此包含 `main`方法，并将其配置为 JavaScript 编译的起始点。

如果你使用 Git，可以通过将 Caos 作为 Git 子模块导入来跟踪其新版本，例如使用以下命令 `git submodule add <link-to-Caos-local-git-repo> lib/Caos`。假设前面提到的 `Main.scala` 文件包含一个 `Main` 对象，其中有一个 `main` 方法，现在需要按照 `.sbt` 图中的方式编辑这两个图 5所示的文件。

```
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"
~ )
  .dependsOn(Caos)
```

build.sbt

```
addSbtPlugin(
   "org.scala-js"
   "sbt-scalajs"
   "1.7.1"
)
```

project/plugin.sbt

图 5。 用于配置将项目编译为 JavaScript 的 SBT 构建文件

要编译前端，只需在命令行中运行 `sbtfastLinkJS` ，并确保当前目录是项目根目录，与 `build.sbt` 位于同一文件夹。随后， `src/main/scala/iLambda/frontend/Main.scala` 中的 Main 方法将被编译到文件`lib/Caos/tool/js/gen/main.js`中。

A.3 开发 iLambda

在向 `Main.scala` 文件填入网站信息之前，我们需要定义一组核心函数和数据类型。本例中，我们开发了以下类：

– `iLambda/syntax/Program.scala` – 在此我们定义了抽象语法树的结构，在本例中：

```
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)
```

– `iLambda/syntax/Parser.scala` – 在此我们定义解析器，即一个函数 `Parser.parseProgram`，它将字符串转换为上述定义的 `Program.Term`。

– `iLambda/syntax/Show.scala`– 我们在此定义用于将项可视化为字符串的函数，包括表示 Mermaid 图的字符串。

– `iLambda/backend/Semantics.scala` – 在这里，我们通过使用不同方式重写 `Term`（例如，将 λ 抽象应用于另一项（称为 β 归约））来实现 `next` 函数，从而扩展Caos `.sos.SOS` 类。

– `iLambda/backend/{LazySemantics,StrictSemantics}.scala` – 我们在此实现了先前语义的一种变体，仅限于具有特定求值顺序的重写规则；惰性语义优先进行 β 归约，而严格语义则先求值参数。

A.4 配置Caos

作为最后一步，我们现在为 `Main.scala` 填入配置，以生成图 4 中描绘的用于分析 λ 项的小部件。下面给出了这个类的一种可能定义，随后将介绍核心小部件构造器。

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

object MyConfig extends Configurator[Term]:
  val name = "Animator of a simple lambda calculus language"
  override val languageName: String = "Lambda Calculus with addition"

  val parser = iLambda.syntax.Parser.parseProgram

  val examples = List(
  "succ" → "(\x → x + 1) 2" → "Adds 1 to number 2",
  ...)

  val widgets = List(
  "View parsed data"              → view(_.toString, Text),
  "View pretty data"              → view(Show(_), Code("haskell")),
  "Diagram of the structure"      → view(Show.mermaid, Mermaid),
  "Run semantics" → steps(e⇒e, Semantics, Show(_), Text),
  "Run semantics (with diagrams)"→
  steps(e⇒e, Semantics, Show.mermaid, Mermaid),
```

```
    "Build LTS"                    → lts(e⇒e, Semantics, Show(_)),
  "Build LTS - Lazy Evaluation"   → lts(e⇒e, LazySemantics, Show(_)),
  "Build LTS - Strict Evaluation"→ lts(e⇒e, StrictSemantics, Show(_))
  "Find bisimulation: given 'A B',  check if 'A ~ B'" →
  compareBranchBisim(Semantics, Semantics, getApp(_).e1,  getApp(_).e2,
   Show(_),  Show(_)), )


  private def getApp(t:Term): Term.App = t match
    case a:Term.App ⇒ a
    case _ ⇒ sys.error("Input must be an application \"A B\" to compare
    \"A\" and \"B\".")
```

下面总结了一些最常用的构件构造器，相关文档详见 Caos （开放）源代码。

– `view[Pr](v: Pr⇒String, t:ViewType)`根据 `t`，以文本、代码或 Mermaid 图的形式可视化由 `v(p)` 表示的输入程序 `p`；– `steps[Pr,A,St](initSt:Pr⇒St, sos:SOS[A,St], v:St⇒String, t:ViewType)`使用 `sos` 语义描述状态 `initSt(p)` 的演化，并像之前一样使用 `v(p)` 和 `t` 可视化状态；– `lts[Pr,A,St](initSt:Pr⇒St, sos:SOS[A,St], vs:St⇒String,va:A⇒String)`与之前类似，但会尽可能多次展开 `sos` 语义，生成包含可达状态空间的图；– `compareBranchBisim[Pr,A,S1,S2](init1:Pr⇒S1,init2:Pr⇒S2sos1:SOS[A,S1], sos2:SOS[A,S2]v1:S1⇒String,v2:S2⇒String)`返回 `init1(p)`（使用 `sos1`）与 `init2(p)`（使用 `sos2`）之间的分支双模拟、未能找到双模拟的原因说明，或超时消息；使用 `v1` 和 `v2` 可视化状态；– `check[Pr](a: Pr⇒Seq[String])` 生成一组警告消息 `check(a)`，也可能抛出由前端捕获的异常；如果未生成任何消息，则不会显示。

A.5 定义结构操作语义

请记住，有几个小部件接收一个 `SOS[A,St]` 其动作类型为 `A`且状态类型为 `St`的对象。这些接口由Caos定义；例如，下面实现了 iLambda 的部分惰性语义，且未利用接受这一概念。

```
object LazySemantics extends SOS[String,Term] {
  /** What are the set of possible evolutions (label and new state) */
  def next[A>:String](t: Term): Set[(A, 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))
    // Try to evolve the left of an application first
    case App(e1, e2) ⇒
      next(e1).headOption match
        case Some(head) ⇒ Set(head._1 → App(head._2,e2))
        case None ⇒       for (by,to) ← next(e2) yield by → App(e1,to)
    // Remaning cases...
}}
```

为帮助分析**并发系统**，Caos 进一步提供了一个构造器，用于构建一个 `SOS` 由局部状态组成的网络。请注意，我们的 iLambda 示例并未使用此构造器。更具体地说，可以使用函数`caos.sos.Network.sos(sync, relabel, localSOS)` 来构建一个新的 `SOS` ，给定：

– `sync` – 一个用于检查各参与者的标签组合能否在同一步中执行的函数；– `relabel` – 一个将有效的标签组合重命名为新的（全局）标签的函数；– `localSOS` – 一个用于描述每个局部参与者行为的 `SOS` 对象。

这些函数使用了一种状态概念，该状态由局部状态列表以及一个捕获网络状态的对象组成。例如，这曾用于分析分支 pomset 的 [9,8]可实现性。

A.6 总结

这是 `iLambda` 一个简单的示例，可在线获取，仅用于说明如何启动一个使用 Caos的项目。该项目的大部分内容是在**大约 1 天内**开发完成的，部分原因是开发人员熟悉这些库以及解析组合子的用法。这里我们仅展示了可用小部件中的一小部分，重点介绍最常用的小部件；更多文档和示例可在线获取。