CAOS
A Reusable Scala Web Animator of Operational Semantics

José Proença & Luc Edixhoven
(Polytechnic Institute of Porto, Portugal) *(Open University and CWI, the Netherlands)*

Tool paper – 20 June – COORDINATION 2023

CISTER
Research Center in
Real-Time & Embedded
Computing Systems

isep Instituto Superior de Engenharia do Porto

Open
Universiteit

CWI

CAOS

A Reusable Scala Web Animator of Operational Semantics

Computer-**A**ided design of **O**perational **S**emantics

José Proença & Luc Edixhoven

(Polytechnic Institute of Porto, Portugal) *(Open University and CWI, the Netherlands)*

Tool paper – 20 June – COORDINATION 2023

CISTER Research Center in Real-Time & Embedded Computing Systems

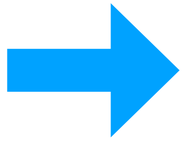
isep Instituto Superior de Engenharia do Porto

Open Universiteit

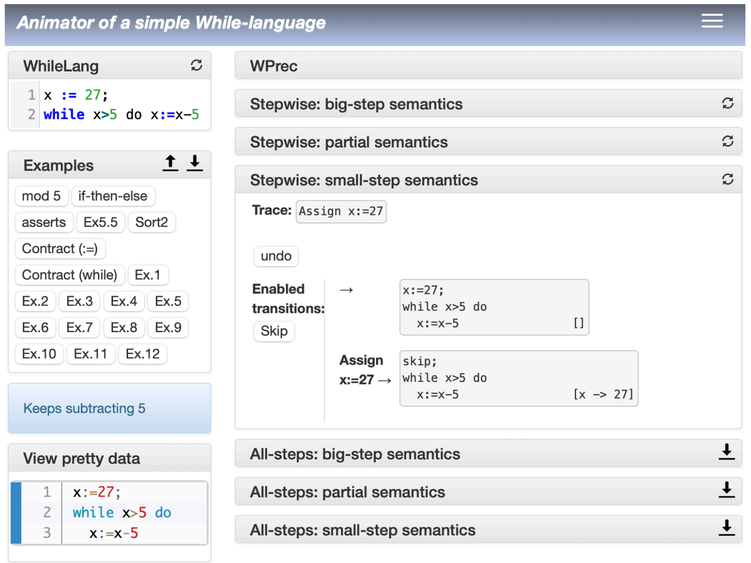
CWI

CAOS framework

My program structure
+
Analysis of my program
+
How to evolve my program



Interactive
Web
Frontend



Scala

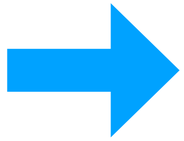
CAOS framework

AST + parser

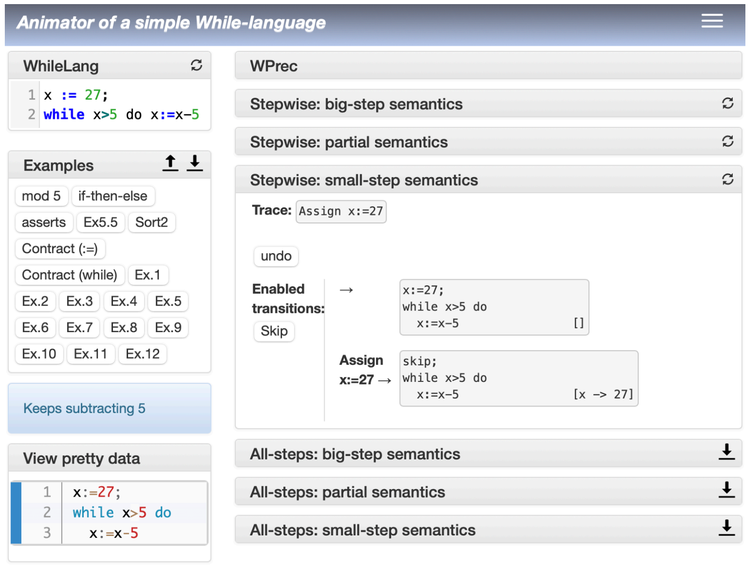
AST => Text/Diagram

State => Next[State]

My program structure
+
Analysis of my program
+
How to evolve my program



Interactive
Web
Frontend



Is CAOS for me?

Know:

Scala (or Java)

Investigating:

a **Program** or a **Data Structure**

Want to:

- experiment with **new analysis**
- have quick/intuitive **feedback**
- **Explain/teach** ideas to others (e.g., build companion prototypes)

Appreciate help to:

- build **visual representations** (UML-like)
- animate **reduction rules** (also of interacting systems)
- **compare** program behaviours

Is CAOS for me?

Know:

Scala (or Java)

Investigating:

a **Program** or a **Data Structure**

Want to:

- experiment with **new analysis**
- have quick/intuitive **feedback**
- **Explain/teach** ideas to others (e.g., build companion prototypes)

Appreciate help to:

- build **visual representations** (UML-like)
- animate **reduction rules** (also of interacting systems)
- **compare** program behaviours

Examples

https://github.com/arcalab/CAOS

How do I use CAOS?

1. Check examples : https://github.com/arcalab/CAOS
2. Get CAOS source code (e.g., git submodule)
3. Set SBT to use a compiler to JS
4. Create a **configuration** object for CAOS
5. Compile to JS
6. Open a provided CAOS/tools/index.html

A glimpse at the code

Animator of a simple lambda calculus language

Lambda Calculus with addition

```
1 // Example with infinite beta
2 // reductions inside a reduceable
3 // term
4 (\n -> if0 n 1
5    ((\x -> (x x)) (\x -> (x x)))
6 ) (2+2)
```

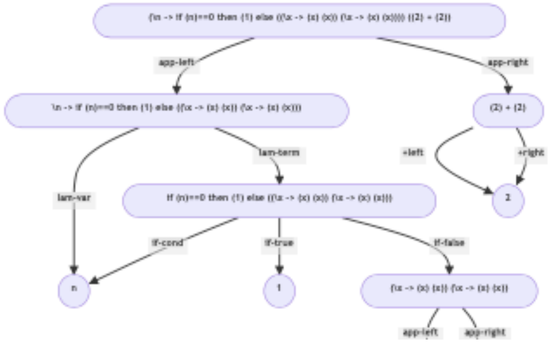
Examples

succ | omega | non-determ | if0 | triangle

View parsed data

View pretty data

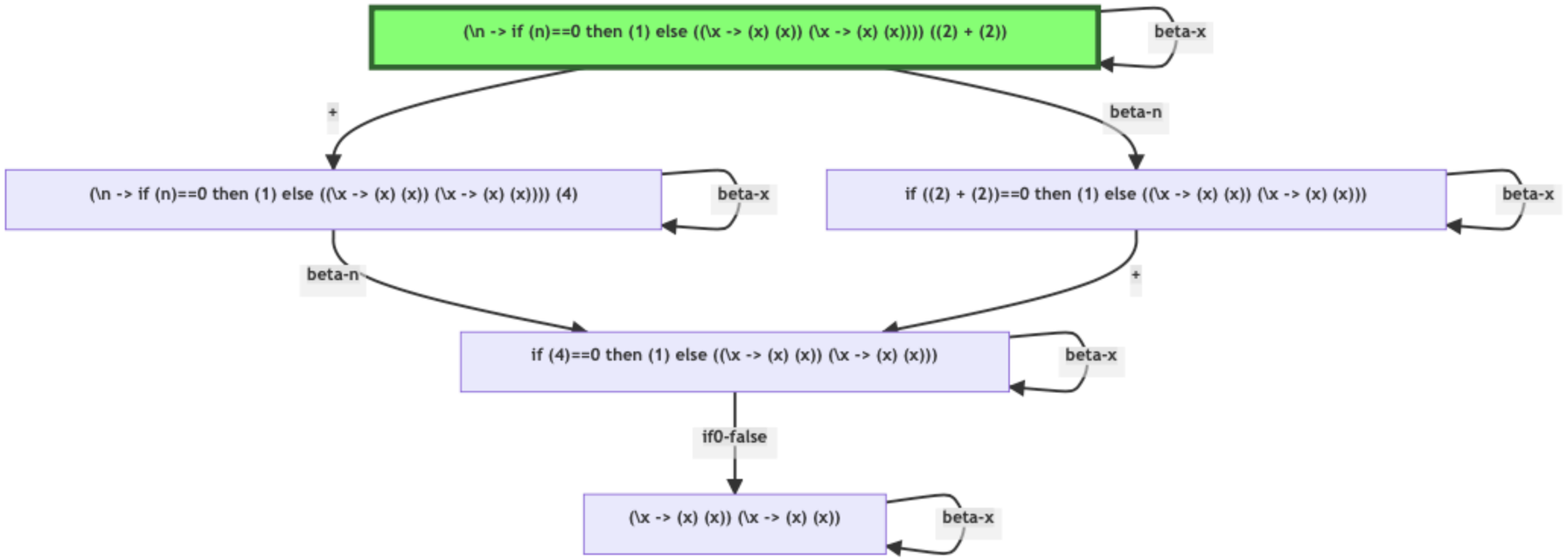
Diagram of the structure



Run semantics

Run semantics (with diagrams)

Build LTS



Build LTS - Lazy Evaluation

Build LTS - Strict Evaluation

1. Configure SBT (build tool)

```
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

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

project/plugin.sbt

1. Configure SBT (build tool)

Compile to JS

Where:
Main class

Where:
to compiled JS

```
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

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

project/plugin.sbt

2. Internal representation (AST)

```
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)
```

3. Main file (set up widgets)

```
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 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 nu
    ...)
```

```
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(_)),
)
```

4. Define SOS semantics

```
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...
}}
```

Non-det. Labelled Trans. System

4. Define SOS semantics

```
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) ⇒
```

```
val networkSOS = Network.sos(sync, relabel, localSOS)

where
  sync:     (List[Set[LocalAct]], NetSt) ⇒ Set[(List[Option[LocalAct]], NetSt)]
  relabel:  List[Option[LocalAct]]       ⇒ NetAct
  localSOS: SOS[LocalAct, LocalSt]
```

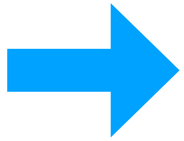
CAOS framework

AST + parser

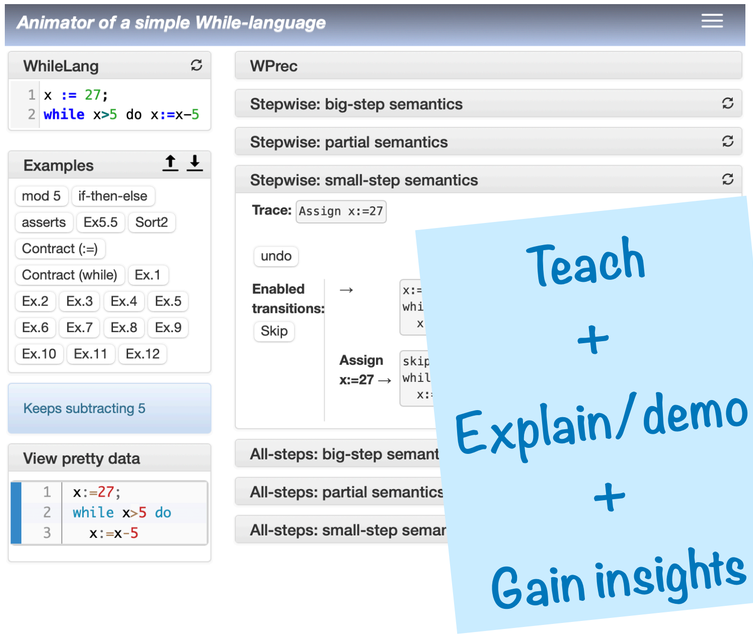
AST => Text/Diagram

State => Next[State]

My program structure
+
Analysis of my program
+
How to evolve my program



Interactive
Web
Frontend



Teach
+
Explain/demo
+
Gain insights

Scala