

ragni-cas - A Pure *Ruby* Automatic Differentiation Library for Fast Prototyping of Interfaces

Matteo Ragni^a

^a*Department of Industrial Engineering, University of Trento, 9, Sommarive, Povo di
Trento, Italy*

Abstract

Ca. 100 words

Keywords: CAS, code-generation, Ruby

1. Motivation and significance

Ruby is a purely object-oriented scripting language that allows to express several programming paradigms. It was designed in the mid-1990s by Yukihiro Matsumoto (also known as *Matz*), and it is internationally standardized since 2012 as ISO/IEC 30170.

With the advent of the *Internet of Things*, a written from scratch version of the *Ruby* interpreter called *mRuby* (*eMbedded Ruby*) has been published on *GitHub* by Matsumoto in 2014. The new interpreter is a lightweight implementation aimed at both low power devices and personal computer that complies with the standard. *mRuby* has a completely new API, and it is designed to be embedded in a complex project as a front-end interface — e.g. a numerical optimization suite may use *mRuby* to get problem input definitions.

The *Ruby* code-base exposes a a large set of utilities in core and standard library. This set of tool can be furthermore expanded through libraries, also

Email address: `matteo.ragni@unitn.it` (Matteo Ragni)

16 known as *gems*. Even the high number of gems deployed and available, there
17 is no complete library that implements a **symbolic automatic differenti-**
18 **ation** (AD) engine that also handles some basic computer algebra routines
19 that is cross compatible with all the different *Ruby* interpreter.

20 *Ruby* has matured its fame as a web oriented language because of its ca-
21 pabilities web template system and in general for processing complex content.
22 This characteristic allows to efficiently generate code in other languages, and
23 given such a gem, it is really easy to develop rapidly a specific code genera-
24 tor for well known software — e.g. IPOPT.

25 The library that is described in this work, is a gem implemented in pure
26 *Ruby* code and thus compatible with all interpreter that complies with the
27 standard, that is able to perform symbolic AD and some simple computer
28 algebra operations. In particular the library aims at:

- 29 • be an instrument for rapid development of prototype interface for nu-
30 merical algorithms — including the *mRuby* engine or exporting gener-
31 ated code — that can be in different languages;
- 32 • the library allows to rapidly test workaround for numerical issues par-
33 ticular to certain problems, by changing on request how the code is
34 exported, and how basic functions are formulated;
- 35 • long term impact, quite ambitious, of creating a complete open-source
36 CAS system for the *Ruby* language, that must be compatible with all
37 the interpreters that comply with the standard.

38 This is not the first gem that is able to perform AD. The available com-
39 puter algebra library for *Ruby* are:

40 ***Rucas*** gem at early stage and with discontinued developing status; it im-
41 plements basic simplification routines. There is no AD method, but it

42 is one of the milestones.

43 ***Symengine*** is a wrapper for the C++ library *symengine*. The back-end
44 library is very complete, but it is compatible only with the mainstream
45 *Ruby* interpreter. At the moment, the *SciRuby* project reports the gem
46 as broken, and removed it from its codebase. From a direct test, when
47 performing AD of a function, the engine returns always `nil`.

48 ***Datamelt*** probably the most complete choice, is a Java library and com-
49 patible with one particular *Ruby* virtual machine, called *JRuby*.

50 2. Software description

51 2.1. Software Architecture

52 *ragni-cas* is an object oriented AD gem that supports some simple com-
53 puter algebra routines such as *simplifications* and *substitutions*. When gem
54 is required, automatically overloads some methods of the `Fixnum` and `Flo-`
55 `at` classes, to make them compatible whit symbolic objects.

56 Each symbolic function is an object modeled by a class. The class in-
57 herits from a common virtual ancestor: `CAS::Op(operation)`. An operation
58 encapsulates sub-operations recursively, building a linked list, that can be
59 considered as the mathematical equivalent of function composition:

$$(f \circ g) \tag{1}$$

60 When a new operation is created, it is appended to the linked list, creat-
61 ing a graph that can have an arbitrary number of branches. The number of
62 branches are determined by the parent container class of the current symbolic
63 function. There are three possible containers. Single argument functions
64 — e.g. `sin(·)` — have as closest parent the `CAS::Op` class, that links to one

65 sub-graph. Functions with two arguments — e.g. difference or exponential
 66 function — inherit from `CAS::BinaryOp`, that links to two subgraphs. Func-
 67 tions with arbitrary number of arguments — e.g. sum and product — have
 68 as parent the `CAS::NaryOp`¹, that links to an arbitrary number of subgraph.
 69 The different kind of containers allows to introduce some properties like *as-*
 70 *sociativity* and *commutativity*. Each container allows to access the subgraphs
 through instance properties. Containers structure is shown in figure 2.1

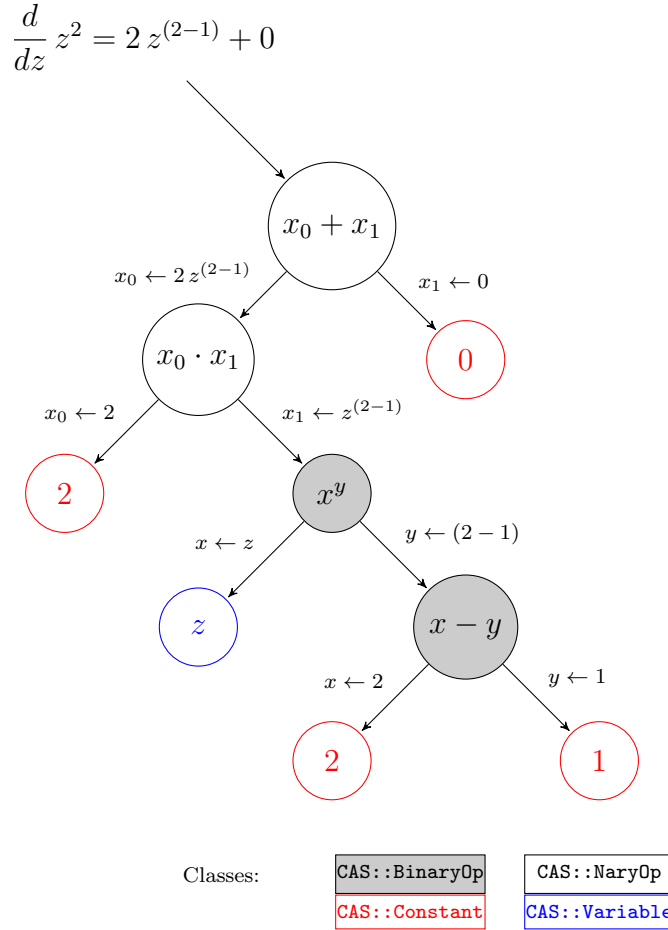


Figure 1: Example graph from the function expressed in script ??

71

¹Please note that this container is still at experimental stage

Terminal leaf of the graph are the two classes `CAS::Constant` and `CAS::Variable`. The first is a node for a simple numerical value, while the latter represents an independent variable, that can be used to perform derivatives and evaluations. As for now, those nodes are only scalar numbers, with the plan to define also the vector and matrix extensions in the next major release.

Automatic differentiation (`CAS::Op#diff`) crosses the graph until reaches the ending node. The terminal node is the starting point for derivatives accumulation, the mathematical equivalent of the chain rule:

$$(f \circ g)' = (f' \circ g) g' \quad (2)$$

The recursiveness is used also for simplifications (`CAS::Op#simplify`), substitutions (`CAS::Op#subs`) and evaluations (`CAS::Op#call`).

2.2. Software Functionalities

The main functionality of the library is the **AD**, that can be performed against an independent variable or a symbolic expression. The function that performs the differentiation is a method of the `CAS::Op` container. Argument of the function is again a `CAS::Op` or a `CAS::Variable`.

```
\label{code:example1}
x = CAS::vars 'x'           # creates a variable
f = x ** 2 + 1              # define a symbolic expression
f.diff(x)                   # derivative w.r.t. x
# => 2 * x ^ (2 - 1) + 0
```

Resulting graph still contains a zero, since **simplifications** are not executed automatically. Each node of the graph contains some rules for simplify itself, that means simplification engine can see only one node ahead in the graph. Simplification are called recursively inside the graph, exactly like AD, bringing the strong limitation of not handling simplifications that come

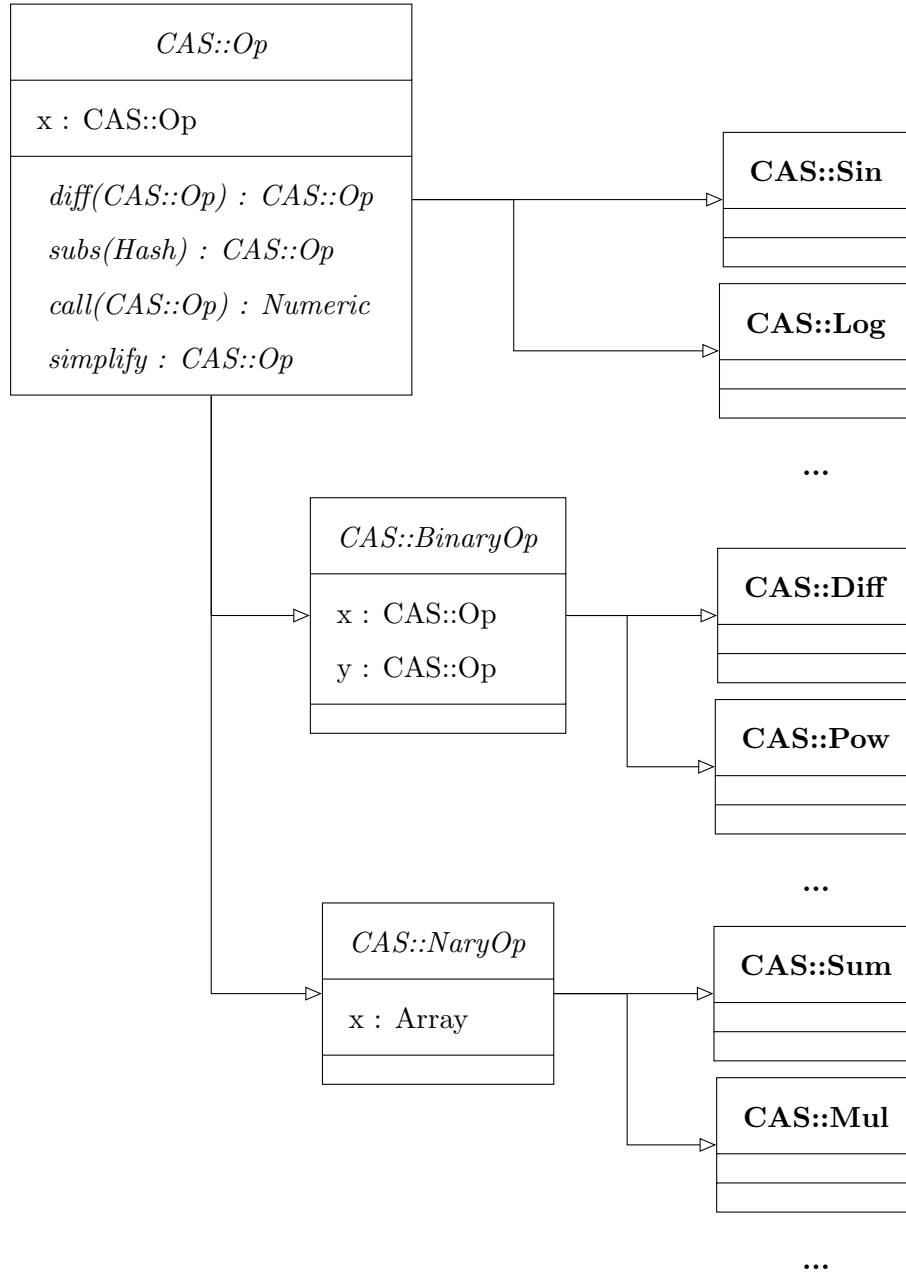


Figure 2: Simplified version of classes interface and inheritance

97 from *heuristic expansion* of sub-graphs — e.g. simplification through the use
 98 of trigonometric identities. Those simplification must be achieved manually
 99 using **substitutions**.

```

100 x, y = CAS::vars 'x', 'y' # creates two variables
101 f = CAS::log( CAS::sin( y ) ) # symbolic expression
102 f.subs({ y=> CAS::asin(CAS::exp(x)) }) # perform substitution
103 f.simplify # simplify expression
104 # => x

```

105 The graph can be **evaluated** substituting defining some values for the
106 independent variable in a feed dictionary. The graph is recursively reduced
107 to a single numeric value:

```

108 x = CAS::vars 'x' # creates a variable
109 f = x ** 2 + 1 # define a symbolic expression
110 f.call({ x => 2 }) # evaluate for x = 2
111 # => 5

```

112 Symbolic functions can be used also to create expressions — e.g. $f(\cdot) \geq$
113 $g(\cdot)$ — or piecewise functions — e.g. $\max(f(\cdot), g(\cdot))$:

```

114 x = CAS::vars 'x'
115 f = x ** 2
116 g = 2 * x + 1
117 f.greater_equal g
118 # => ((x)^(3) ((2 * x) + 1))
119 CAS::max f, g
120 # => (((x)^(3) ((2 * x) + 1)) ? (x)^(3) : ((2 * x) + 1))

```

121 Expression are stored in a special container class, called `CAS::Condition`.

122 The library is developed to be used for **code generation**, and in some
123 case also **metaprogramming**. Expressions, once manipulated, can be easily
124 exported as source code (in a defined language —i.e. the following example
125 in standard *Ruby* code):

```

126 x = CAS::vars 'x' # creates a variable
127 f = CAS::log(CAS::sin(x)) # define a symbolic function
128 # => Math::log(Math::sin(x))

```

129 the same function can be also used to create directly a callable *lambda* already
130 parsed by the interpreter:

```
131 proc = f.as_proc          # exports callable lambda
132 proc.call({"x" => Math::PI/2})
133 # => 0.0
```

134 Must be noted that parsing the graph creates a snapshot, and any further
135 modification to the expression will not update the callable object. This draw-
136 back is balanced by faster execution time of the *lambda*: when a graph needs
137 only to be evaluated in a iterative algorithm, and not to be manipulated,
138 transforming it in a *lambda* reduces the execution time per loop.

139 Other functionalities — e.g. displaying drivers — are not reported in the
140 current work for a sake of brevity. Please refer to gem documentation for
141 more insight.

142 **3. Illustrative Examples**

143 **4. Impact**

144 **5. Conclusions**

145 **Acknowledgements**

146 This research did not receive any specific grant from funding agencies in
147 the public, commercial, or not-for-profit sectors.

148 **Current code version**

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.0.0
C2	Permanent link to code/repository used for this code version	github.com/MatteoRagni/cas-rb & rubygems.org/gems/ragni-cas
C3	Legal Code License	MIT
C4	Code versioning system used	<i>git</i> (GitHub)
C5	Software code languages, tools, and services used	<i>Ruby</i>
C6	Compilation requirements, operating environments	<i>Ruby</i> $\geq 2.x$, <i>pry</i> for testing console (optional)
C7	If available Link to developer documentation/manual	rubydoc.info/gems/ragni-cas
C8	Support email for questions	info@ragni.me

Table 1: Code metadata (mandatory)