Implementing JoyLoL

Dr Stephen Gaito
PerceptiSys Ltd
April 30, 2019

Perish *then* Publish Press.

This document was prepared using ConTEXt and LuaTEX on April 30, 2019.

Contents

1 Introduction

1.1 Overview

1.1.1 Introduction

We will provide full formal descriptions of six distinctly different programming languages, JoyLoL, WhileLoL, WhileRecLoL, EagerLambdaLoL, LazyLambdaLoL and LogicLoL. The languages WhileLoL and WhileRecLoL will be in the same overall class of languages as most standard imperative programming languages such as, Lua, C, Pascal, Algol, Java, Python, PHP, Ruby, etc. Most programmers will recognize WhileRecLoL as a close, but simplified, cousin to the languages they are currently using. The languages EagerLambaLoL and LazyLambdaLoL represent the class of eager and lazy functional languages such as ML and Haskell respectively. The LogicLoL language represents the class of logic programming lanaguages such as Prolog.

The JoyLoL language, is in a new class of 'concatenative' languages originally explored by Manfred von Thun[1]. The primary importance of JoyLoL is that it *is* a fixed point of the formal semantics operator. As a fixed point of the semantics operator, JoyLoL provides a foundation for both computation and more importantly Mathematics[2].

Across all of these languages the constant similarity is the 'LoL' or List of Lists. In each language the *only* expressions are Lists of Lists. Depending upon the language, these lists of lists are potentially infinite expressions, which will, however, always have a finite description at any particular point in a computation.

As developed over the past 50 years, the formal semantics operator has three parts:

1. Denotational Semantics (roughly equivalent to Tarski's model semantics)
2. Operational Semantics (roughly equivalent to Gentzen's natural deduction)
3. Axiomatic Semantics (roughly equivalent to Type theory)

Good introductions to these three types of formal semantics can be found in [Win93] and [Gun92]. (TODO see [AV80] and [Bil90] for early reports of Prolog's semantics)

The collection of Lists of Lists *is* an "infinitely" "complex" "structure". In its complete incarnation, it is strictly *more* complex than the whole of any formal *set* theory such as ZFC[3]. This is, for finite beings, such as mere mortal mathematicians, a large and complex 'world' to explore. It is a world in which it is very easy to get lost. While we assert that JoyLoL provides a computational foundation for

[1] For example, see [Thu94b], [Thu94a] or [Thu94c] all of which can be found in [Thu11] or [Thu05].
[2] While we assert that JoyLoL provides a *computational* foundation for Mathematics, proving this assertion will be the work of many (future) papers. We will not even attempt a proof in this document.
[3] In this paper we will only consider the component which corresponds to classical, ω-computation.

Mathematics, to help us 'mere mortal mathematicians' orient ourselves, we will often make reference to classical mathematical concepts. It is important to realize that these classical mathematical concepts are simply aids to our mathematical intuition, and not formal statements.

The most important classical intuition is that of Algebra and CoAlgebra, or equivalently, for a Computer Scientist, that of Data and Process. Both classical Mathematics and Computational theory have, by and large, explicitly limited themselves to the well-founded and terminating, largely to avoid Poincaré's 'Vicious Circles'. We will see that the non-well-founded, non-terminating processes, Cantor's 'Absolute Infinite', have a surprisingly easily understood structure, essentially dual to classical set theory. However, the *computational* theory of these non-terminating processes, has a profound impact on Mathematics. By ignoring this computational theory, we, as mathematicians, make simple problems, hard.

Intuitively, a List of Lists, is a potentially infinite structure which records a potentially infinite *structured* collection of observations of a potentially non-terminating process of processes. As *finite* mathematicians, we can only ever manipulate finite structures, *finite records of observations* of a potentially infinte process. One such record of observations might be denoted by, for example:

```
(() ( ( )))
```

This is of course the denotation of Lists in John McCarthy's Lisp, see [MAE+65].

Where classically, formal semantics concentrated on *one* denotation, to provide a *formal* description of these potentially non-terminating process, it is critical that we carefully distinguish *two* distinct denotations: the classical *algebraic* denotation (corresponding to a least fixed point of the semantics operator) and the non-classical *coAlgebraic* denotation (corresponding to a greatest fixed point of the semantics operator). While for data, the data itself is its own denotation, for a potentially non-terminating process, *an answer* (a data object) is insufficient to *denote* that process. Instead the appropriate denotation of a non-terminating process is its trace of observations (or any finite record of this trace which is 'sufficient' for current purposes). Similarly, while the 'big-step' operational semantics might suffice for a data object or a terminating process, the 'one-step' operational semantics is the only definition of operational semantics appropriate for a potentially non-terminating process. Finally, to provide an Axiomatic semantics, with out recourse to classical first order set theory, we will make essential use of finite descriptions of the traces of potentially non-terminating processes.

Philosophically, it is important to know when two 'things' are the 'same'. For an algebraic list of lists, two lists are equal if there is a *finite*, structurally inductive, comparison of the two objects. This is the familiar concept of recursive equality. For sets this is *extensive* equality. For a coAlgebraic list of lists, two potentially non-terminating processes are equal if they respond in the same way to any collection of 'observations'. This is the concept, from Theoretical Computer Science and CoAlgebraic Category theory, of 'bisimulation'.

1.1.2 How certain can certainty be?

We intend to show that JoyLoL provides a foundation for Mathematics. Any foundation for Mathematics, must be 'certain', but what exactly does 'certainty' mean and how 'certain' can a finite computational artefact be?

(TODO: there are two aspects here: (1) implementation vs idea (logical-formalism vs intuitionism) and (2) current mathematical certainty expressed *using* logical-formalism vs other possible approaches)

sCurrently, certainty in mathematics, is generally identified with the *computation* of the 'Logical' 'Truth' of a 'formal' assertion, which represents *a logical implementation*, of an intuitive understanding, of an idea which we want to show is ***'certain'***.

Notice that, as with any *human* language, there are many different possible ways to express the 'same' intuitive thought.

Notice as well that there are multiple different levels of granularity with which to express and 'prove' a given statement to a 'sufficient' 'level' of detail. While most current mathematical 'proofs' are conducted in an informal but rigorous style, the generally acknowledged highest standard of proof is a natural deduction proof expressed using first, second or higher order logical notation. (TODO wrong words!) At the moment, the translation of a high-level mathematical argument into a 'completely rigorous' but impossibly detailed logical argument is very tedious and difficult, largely because the highly detailed, n-th order logical notation, is very far from the original intuitive idea to be proved.

There are many discussions of the Logical-Formalist 'schools' of the foundations of Mathematics, [Gia02], [Sha00] and [Hat82] each provide interesting accounts of the strengths and weaknesses of these approaches. Equally important in any of these expositions, are the accounts of the Intuitionist critique of the logical-formalist approach.

For a (software) Business Analyst or Systems Architect, the key words from the above discussion are, 'computation', 'implementation', 'specification', and 'algorithm'. For the Business Analyst, every specification is always just one of many possible *implementations* of the business problem to be solved. Each different possible phrasing of the specification carries different 'non-functional' (or extra-specificational) implications for the way a given business problem is 'solved'. Equally, for the Systems Architect, each proposed software implementation satisfying a given specification, has different non-functional implications in terms of, for example, speed, memory usage, and programmer or maintenance effort. The critical point here is "*an* implementation is just that *an* implementation", one of many possible solutions to a problem. Each of which provides different capabilities or penalties. As any Business Analyst or Systems Architect knows, business problems can only ever be solved by *an* implementation, but to keep business flexibility in a highly dynamical environment, it should be as easy as possible to *change* implementations

as and when those implementations begin to limit the business growth. Implementations are critical to a business, *but* implementations come and go, the goal is always to solve the business problem.

For our purposes, the classical Logical-Formalist approach using, for example, set theory expressed in a first order logic formalism, or type theory[4], ...

Instead of computing logical truth that a given structure exists, using Gentzen's natural deduction (algorithm), compute the structure itself. However, if we *compute* a structure, how do we know that we have computed the structure we specified?

TODO: We want to discuss deduction vs induction in the generation of knowledge. Mathematics is (almost) pure deduction. However any human subject 'to be done' must include aspects of scientific/engineering induction. That is the verification of any *deductive* mathematical proof, requires some computational system to behave 'correctly'. How do we know that any particular verificational computation of a given proof is correct?

Deductively we can be certain a given computation is, *in theory*, correct. However conducting the *actual* computation entails dependence on *inductively* determined *models* of 'reality' which may or may not apply in a given time or place.

Arthur C. Clarke's 'Nine Billion Names of God', or Anthony S. Haines' 'And on gloomy Sunday...', in two different ways, suggest how potentially extremely rare events, the coincidence of naming all the names of God, or a 'research agency' outside our existence, could both have profound *global* impacts on any given 'computation'. The point here is not that either of these stories are 'true' (though they *could be*), but rather that potentially rare events might break any given model of physically realized computation. There will never be any way a *finite* being can mitigate against these rare events. Given the rarity of these example events in that they will only occur *once* in a given existence, it is not, on the whole, rational, to worry about rare events such as these. So, a *finite* being, can *never* be 100% *certain* of any computation, but we can be fairly certain, or rather, for all rational purposes, a computation can be considered to be *certain enough*.

We can 'draw the line' between deductive certainty and inductively good enough models at various levels in the 'computational hierarchy'. We could take the quite considerable effort to deductively verify the whole computational infrastructure (compilers, operating system, peripherals, CPU, memory, transistor states, etc.) down to the Quantum-mechanical level. Or, we could simply assume a good-enough model of computation of a computational language and then deductively verify any given program in that language. However, no mater where we draw this line between deductive certainty and inductively good enough models, a line must be chosen. The result of any particular computation will only ever be 'good enough'.

As with any security issue, we have a risk / benefit analysis to conduct. We then have to make difficult choices as to how to minimize the costs of the risks, maximize

[4] See for example, [ACV13].

the value of the benefits, while simultaneously minimizing the costs of the effort all required to get the chosen low-level of risks and high-level of benefits.

For current Mathematical practice, an individual mathematician 'verifying' a proof, is likely to be *highly* error prone. However it is assumed that over all interested mathematicians, any mistakes in a given proof *will be found*. To help reduce the difficulty of understanding (i.e. verifying) any given proof statement, mathematicians spend a lot of effort identifying independently useful lemmas from which to build simpler proof statements to a wide range of similar mathematical problems.

Similarly our collective confidence in a given proof of correctness of a given program, will come from running the verification on multiple *independently different* platforms (the equivalent of multiple mathematicians). Our collective confidence will also be increased if the program is structured out of a 'library' of simpler and independently useful 'parts', each of which are verified and more importantly regularly used on a wide range of range of platforms and in a wide range of problems.

See [Mac01] and [Lak76] TODO: provide references to relevant royal society conference

Bootstrap and circularity

Provide box diagram of working parts

white/black box testing.... formal model testing (finite automata corresponding to any finite operational structure) alasthe use of pre/post conditions is insufficient to provide any meaningful input to a model tester as the 'real' complexity of a given operational transition is in the parts not checked in the pre/post-condition HOWEVER, the parts not checked SHOULD NOT effect the transition. So I guess this might be tested. NO unfortunately any computation by case analysis will break this ability. Since the cases will be hidden to the external specifications.... white vs black box...

1.1.3 Judgements

TODO: cover judgements as base case coAlgebraic 'sets'.

1.2 Overview

1.2.1 Implementation

1.2.1.1 Bridging the semantic gap

JoyLoL *is explicitly defined to be* a fixed point of the formal semantic functor, making JoyLoL its own formal semantic definition. However there is, currently, no existing computational device which *implements* the JoyLoL language. That is, there is no computational device which 'runs' JoyLoL code natively.

The objective of this document is to provide an implementation of JoyLoL in as transparently correct way as possible. As discussed in, [Gai17], the formal definition of any computational language has two distinct components: one *deductive* and the other *inductive*. While we can rigorously check any deductive proofs of correctness, we can only ever hope to falsify any inductive tests of correctness. Any formally correct implementation of a computational language needs to be explicitly clear where the line between the deductively provable and the inductively testable is located.

The desired goal of any rigorous implementation is to keep as much as possible of the code deductively provable. Conversely any rigorous implementation needs to keep any code which is only inductively testable as clear and simple as possible. However how and were we draw the line between the deductively provable and the merely inductively testable implementation, will have profound impact upon the *performance* of all resulting JoyLoL computations run using this implementation. Provable correctness *and* performance are *both* critically important.

To obtain the correct balance of correctness, (potential) performance, and simplicity, JoyLoL has been designed as a 'trampolining' interpreter, written in ANSI-C, but meta-compiled from Literate sources written in ConTEXt/LuaTEX which are transcribed into ANSI-C source before being compiled to an executable on a given platform by an appropriately chosen ANSI-C compiler.

Finally since JoyLoL is meant to form a foundation for Mathematics, and, as such, the basis of mathematical proof, we need to ensure the JoyLoL language is accessible within the most common tool, TEX, used by mathematicians to communicate their proofs. To do this we wrap JoyLoL in a simple Lua interface. By wrapping the ANSI-C JoyLoL libraries in a Lua interface, we allow the JoyLoL libraries to be used, in particular, inside LuaTEX and hence inside LATEX and ConTEXt documents. At the moment, LATEX does not make integral use of LuaTEX's Lua subsystem. Instead we make use of ConTEXt for most of our documentation and mathematical writing, since ConTEXt does make integral use of LuaTEX's Lua subsystem.

1.2.1.2 Literate Sources

The literate sources, provide human readable documentation and justifications for each JoyLoL Co-Algebraic extension, complete with formal semantic definitions of each axiomatic word in JoyLoL.

1.2.1.3 ANSI-C system code

We build the lowest level system code for JoyLoL using ANSI-C with a few "standard" POSIX extension libraries. We have chosen ANSI-C for its:

- **portability**: There are a large number of ANSI-C compatible C compilers which target almost *all* computers currently in existence.
- **inter-working**: There are a large number of code libraries implementing useful algorithms which can be 'loaded' into the runtime image of an ANSI-C compiled program.
- **performance**: *If* desired, the overall JoyLoL interpreter can be compiled using any of the modern ANSI-C compilers' optimization modes. Since ANSI-C is so heavily used, the optimizing modes of most compilers are realatively well 'understood', tested and stable.
- **transparency**: The semantic gap between ANSI-C and the 'assembler' / 'machine-code' of almost any computer is small enough that a *large number* of skilled programmers could, if needed, hand code any C code directly into a given machine-code. For our needs, this means that there is no obscure mapping between the short pieces of JoyLoL implementation code and a given CPU's machine-code. This ensures that what JoyLoL does when running is 'relatively' easy to understand for most programmers.
- **familiarity**: While programmers are only a small part of our target audience, given we are explicitly dealing with the mathematics of computation, the programming community is an important part of the audience. More importantly 'most' programmers have a 'working' familiarity with the subset of ANSI-C used to implement the lowest levels of JoyLoL.

1.2.1.4 Interpreter structure

Since all JoyLoL words explicitly manage the context's data and process stacks, there is, in theory, no need for the ANSI-C call stack. The typical C-like language uses the call stack to hold both local data, any call parameters, as well as the process location to which to 'return'. Because data and process information are mixed on the call stack, to keep the call stack from growing without bounds, the explicit

expectation of any C-like language is that calls 'return' in the *strict* reverse order in which they are called, and that, more importantly, there is a finite limit on the number of calls a process might make.

When using JoyLoL as a foundation of Mathematics, we will find that there are many processes which do not naturally follow this strict return in reverse order pattern. Keeping the data and process stacks separate ensures that JoyLoL does not need to be enforce this strict call pattern. Instead JoyLoL implements a 'continuation passing style' of programming, see, for example, [SW00], [Gor79, section 5.1), [Ten81, chapter 10) [FW08, chapters 5 and 6).

In typical programming languages, this continuation passing style is implemented using either explicit 'jumps'/'gotos', see [Ten81, chapter 10), or, alternatively, using 'tail calls', see [Pro01, chapter 2), or [FW08, section 6.2). The use of explicit computed gotos, which are implemented as non-ANSI-C standard *extensions*, requires the use of global variables to pass the data and process stacks. Unfortunately this use of global variables inhibits most standard C compiler optimizations[5].

In the best of all worlds, we could implement JoyLoL's lowest levels using a *systems programming language* with native 'tail calls'. Since the data and process stacks can now be passed as 'normal' procedure parameters, standard C compiler optimizations will not be inhibited. Unfortunately no widely used systems programming language currently implements tail calls. While the functional languages such as Haskell, and Lisp/Scheme have native tail calls, they do not map sufficiently cleanly onto the underlying machine-language of a given computer's CPU. All C-like languages, who do typically map reasonably cleanly onto a given CPU's architecture, do not have a tail call friendly call structure.

To solve this problem, following [FW08, Section 5.2), we use a 'trampolining' interpreter as the main 'eval loop' for JoyLoL. The use of trampolining, ensures that the C-call stack never grows very large. JoyLoL words are implemented as simple C procedures keeping the structure of the resulting C-code simple. Trampolining also allows the use of external libraries which expect C-call stacks. For each cycle around the JoyLoL eval loop, the top of the process stack is used to determine which C procedure (JoyLoL word) to call next.

Unfortunately, while providing simply structured C-code, trampolining of small C procedures, is not as performant as a system which makes use of native tail calls. Instead by using the ConTEXt/LuaTEX based meta-compiler we can pre-compile any complex JoyLoL word definitions as explicit C procedures which *can allow* a given C compiler's optimization mode to produce performant code. This means

[5] With considerably more effort, we *could* arrange to keep the data and process stacks in 'local' variables in 'simulated' 'C-call stacks'. While this *might* improve performance, the use of such simulated C-call stacks, being so non-standard, would seriously reduce the number of programmers who could *easily* understand the resulting C-code.

that the JoyLoL system itself can be written in JoyLoL, allowing it to be proven deductively correct, yet still be performant.

1.2.1.5 Call structure

JoyLoL is a Forth-like language which manipulates 'stacks'. Almost all existing general purpose computational devices are 'register based'. Cleanly and performantly implementing stack based languages on a register based computational device has been previously explored in Ertl's thesis, [Ert96].

1.2.1.6 Bootstrapping JoyLoL

Since we want your tool set to be as rigorous as possible, we ultimately need to use JoyLoL to deductively prove its own correctness. Unfortunately, at least initially, most users do not have a running version of JoyLoL. In order to obtain the *first* running version, we need to 'bootstrap' the tool set by building an initial version of JoyLoL which is not rigorously checked.

Since we assume that any serious user of JoyLoL will be using JoyLoL to develop mathematical arguments, and hence will be using ConTEXt, we will provide this 'bootstrapped' JoyLoL using Lua. To do this each JoyLoL CoAlgebra will contain a highly simplified version of itself as pure Lua using the `MinJoyLoL` environment.

1.2.2 Organization

While we assert that all of the CoAlgebraic extensions provided in this document are conservative extensions over JoyLoL provided with only Lists of Lists, it is useful, for 'bears of *very* little brains' such as myself, to work, at least initially, with the extra structure provided by these CoAlgebraic extensions. We will show in a subsequent paper that all of the CoAlgebraic extensions provided in this document, are conservative extensions over JoyLoL provided with only lists of lists.

1.3 The syntax and semantics of JoyLoL

1.3.1

The formal semantics of the *other* programming languages are certainly simpler. This is because, the semantics of the other languages, are defined in terms of the semantics of JoyLoL. This means that any deep subtleties are simply encapsulated in the semantics of JoyLoL. Any reader who is willing to take the existence of the formal semantics of JoyLoL as given, are welcome to skip to the other chapters until they are ready or willing to understand the full import of the semantics of JoyLoL.

The essential subtleties of our formal semantics for JoyLoL comes from three areas:

1. **Foundations**
   We are explicitly working *without* classical first order set theory. Yet formal semantics *is* defined as *mappings* between *collections* of 'things'.
   How do we define mappings and collections while we are defining semantics with which to define mappings and collections with which... ?[6]
2. **Metamathematics**
   A partial answer to our foundational question above
   Any formal semantics is a programming language *about* another 'object-level' programming language (which is itself a language *about* objects as 'values').
3. **Computation of *properties* of potentially non-terminating processes**

We begin by listing the syntax of JoyLol together with its associated semantics. We loosely follow the presentation in [Win93].

[6] I personally know of no completely rigorous exposition of the foundations of classical mathematics. I suspect the closest, mathematically, we might come close to this is in Gödel's proofs first and second incompleteness results. However, even here the results hinge upon an informal interpretation of 'truth' at the meta-level.

1.4 The syntax of WhileRecLoL

22 22

1.5 CPU Model

In [Gai18] we modelled the *simple* model of computation which we call `JoyLoL`. However there are no implementations of this `JoyLoL` model of computation in a commercially available CPU. In this chapter we define a model of an idealized *commercially* available CPU upon which we can rigorously implement pure `JoyLoL`.

In both our models of computation, pure `JoyLoL` and an idealized commercial CPU, the important distinction is the underlying memory models. The pure `JoyLoL` model of computation is essentially a *restricted access* Harvard model of memory. The pure `JoyLoL` model has a pair of memories one each for the data and the process. Each memory is structured as a stack for which only the 'top' of the stack can be directly accessed.

We tend to assume that our commercial CPUs have a von Neumann model of memory consisting of *one, large, randomly accessible* memory at the Instruction Set Architectural (ISA) interface[7]. As we will discuss below, for a mathematician, this assumption, of a *single, large, randomly accessible* memory, is misguided.

For a 64 bit CPU, the *maximal* addressable memory is 16 Exabytes. For a mathematician, this is *tiny* when compared to ω. We could model a 128 bit CPU, or, an x bit CPU for any particular value of x we might choose, however for any *finite* value of x, we will still have a *maximal* addressable memory which is *tiny* compared to ω. While we could consider an idealized ω bit CPU, no such *implementable* CPU exists. Our goal here is to model a *realizable* CPU upon which `JoyLoL` might be able to run efficiently.

1.5.1 Model details

1. Storage
    1. uint64, uint32, uint16, uint8
    2. int64, int32, int16, int8
    3. byte, utf8Char, char
    4. at the moment we do *not* model floats or doubles
    5. we also do *not* explicitly model pointers. (Should we?)

[7] Note that most modern CPUs actually have a modified Harvard architecture at level of the the underlying micro-architecture, since they implement a pair of data and instruction caches between the Random Access Memory (RAM) and the CPU itself. We will, for our purposes, ignore this underlying micro-architecture since we are only interested in modelling the ISA interface. At the level of the ISA, both data and instructions are 'loaded' from 'one' 'large' (randomly accessible) Memory (RAM).

6. arrays
7. structures

2. Actions
   1. array indexing
   2. addition, subtraction, mutiplication, division
   3. and, or, xor, ...
   4. assignments
   5. functions

1.5.2 Memory Model

In `JoyLoL` the memory model, we have unlimited memory but it is only directly accessible at the tops of the data and process stacks.

In `JoyLoLRM` the memory model, we have unlimited memory which is again only directly accessible at the tops of a small number of stacks which include `JoyLoL`'s data and process stacks. The non-data, non-process stacks are referred to as registers or local variables.

1.5.3 Call patterns

In `JoyLoL`, `JoyLoL` words pass their parameters on the data or process stack[8].

In `JoyLoLRM` there are two possible call patterns, as a `JoyLoL` word, or inline. When called as a `JoyLoL` word, parameters can only be passed on the `JoyLoL` data and process stacks. When 'called inline', the implementation of all callee `JoyLoL` words are embedded inline into the body of the implementation of the caller `JoyLoL` word.

TODO: What about more traditional C-like call patterns? Can we interpret them as implicitly pushing the arguments on to the data stack? *Most* words simply manipulate the top few items on the data stack, and return *one* result. This is essentially the standard C-like call pattern. Lua allows multiple return values:

```
x, y, z = aCall(a, b, c)
```

[8] Context changing words are allowed to return the new context via a third 'return' parameter. However this special parameter could be modelled as the top of the data stack.

TODO: This leaves one case: how do we denote the *infinite* change to the data stack. Effectively this can only be the removal of the whole stack, AKA clearData.

TODO: What about words that manipulate the process stack? I think most such words actually always manipulate the data stack as well. How are these represented in a C-like way? Since the process stack *is* the continuation structure of a method, the Lua multi-return is not appropriate for the process changes. However the multi-return does suggest a way forward: the multi-return is all about what will be done with the results on the data stack. In process terms this (might) translate into control structures. However a problem here is that the callee word has the arbitrary ability to *change* the contents of the process stack.. and so what happens once the callee returns. This is difficult if we equate the contents of the process stack as a control structure compiled *before* the callee word gets called. Is it this difficulty which precludes some words being compiled? (see below)

TODO: When does a word get too complex to be compiled (inline)?

2 Base CoAlgebras

2.1 Overview

In the next four parts we explore the base coAlgebras required to implement the `JoyLoL` compiler/interpreter.

3 Memory CoAlgebras

3.1 Memory

3.1.1 Overview

From a purely *mathematical* point of view there is no such thing as a (global) randomly accessible (computational) memory.

4 Base Type CoAlgebras

35 35

4.1 Base Types

4.1.1 Overview

This part collects a number of important base types.

5 Container CoAlgebras

39 39

5.1 Containers

5.1.1 Overview

This part collects a number of useful containers of other types.

5.2 avlNode

avlNode

5.2.1 Goals

Our primary goal is to implement a dictionary based upon a balanced binary AVL tree. This coAlgebra implements individual nodes in the tree.

5.2.2 JoyLoL Words

JoylolCode : default

```
1  // Some example JoyLoL code
2  //
3  // (Taken from
4  //   ExpositionGit/tools/conTeXt/joylol-c/base/dictNodes/buildDir/dictNodes.c
5  // )
6
7  /*
8   * This is a test
9   *
10  * This is another line
11  */
12
13 CoAlgebra AVLNode
14
15   Invariant
16   EndInvariant
17
18   Lexer
19   EndLexer
20
21   Parser
22
23   EndParser
24
25   Structure
26     super    : Object,
27     symbol   : Symbol,
```

```
28      preObs   : Object,
29      value    : Object,
30      postObs  : Object,
31      left     : DictNode,
32      right    : DictNode,
33      previous : DictNode,
34      next     : DictNode,
35      height   : SizeNum,
36      balance  : LongNum
37    EndStructure
38
39    Method findSymbolRecurse
40
41      PreDataStack
42        aDict     : Dictionary,
43        anAVLNode : DictNode,
44        aSymbol   : Symbol
45      EndPreDataStack
46
47      PreCondition
48        this is a test
49      EndPreCondition
50
51      RMCode
52        if (!anAVLNode) return NULL;
53        int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
54        if (aStrCmp < 0) {
55          // aSymbol < anAVLNode->symbol // search the LEFT subtree
56          return findSymbolRecurse(aDict, anAVLNode->left, aSymbol);
57        } else if (0 < aStrCmp) {
58          // aSymbol > anAVLNode->symbol // search the RIGHT subtree
59          return findSymbolRecurse(aDict, anAVLNode->right, aSymbol);
60        } else {
61          // aSymbol == anAVLNode->symbol // return this association pair
62          return anAVLNode;
63        }
64        return NULL;
65      EndRMCode
66
67      PostDataStack
68        anAVLNode : DictNode
69      EndPostDataStack
70
```

```
71      PostCondition
72        this is a test
73      EndPostCondition
74
75    EndMethod

76  DictNodeObj* findLUBSymbolRecurseImpl(
77    DictObj     *aDict,
78    DictNodeObj *anAVLNode,
79    Symbol      *aSymbol
80  ) {
81    assert(aDict);
82    if (!anAVLNode) return aDict->firstSymbol;
83
84    DEBUG(aDict->jInterp,
85          "findLUBSymbol %p {%s}[%s] %p %p\n",
86          anAVLNode, anAVLNode->symbol, aSymbol,
87          anAVLNode->left, anAVLNode->right);
88
89    int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
90    DEBUG(aDict->jInterp, "findLUBSymbol cmp: %d\n", aStrCmp);
91    if (aStrCmp < 0) {
92      // aSymbol < anAVLNode->symbol
93      // the current anAVLNode->symbol is an upper bound
94      // search the LEFT subtree for a smaller upper bound
95      if (anAVLNode->left) {
96        DictNodeObj* aNode = findLUBSymbolRecurse(aDict, anAVLNode->left,
97  aSymbol);
98        if (!aNode) {
99          // there is nothing in the LEFT subtree which is an upper bound
100         // so return this node.
101         return anAVLNode;
102       }
103       // we have found a smaller upper bound... so return it
104       return aNode;
105     }
106     // there is nothing less than this node so return this node
107     return anAVLNode;
108     //
109   } else if (0 < aStrCmp) {
110     // anAVLNode->symbol < symbol
111     // the current anAVLNode->symbol is a lower bound
112     // search the RIGHT subtree for any upper bounds
```

```
113     if (anAVLNode->right) {
114       return findLUBSymbolRecurse(aDict, anAVLNode->right, aSymbol);
115     }
116     // there is nothing greater than this node so return NULL to signal
117 failure
118     return NULL;
119     //
120   } else {
121     // aSymbol == anAVLNode->symbol
122     // the current anAVLNode->symbol is the lowest possible upper bound
123     // return it
124     return anAVLNode;
125     //
126   }
127   return aDict->firstSymbol;
128 }
129
130 DictNodeObj* insertSymbolRecurseImpl(
131   DictObj     *aDict,
132   DictNodeObj *anAVLNode,
133   Symbol      *aSymbol
134 ) {
135   assert(aDict);
136   JoyLoLInterp *jInterp = aDict->jInterp;
137   assert(jInterp);

138   if (!anAVLNode) return newDictNode(jInterp, aSymbol);
139
140   StringBufferObj *aStrBuf =
141     (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
142
143   DEBUG(jInterp, "\ninsertSymbolRecurse %p <%s>[%s] %ld:%zu\n",
144         anAVLNode, anAVLNode->symbol, aSymbol,
145         anAVLNode->balance, anAVLNode->height);
146
147   DEBUG(jInterp, "insertSymbolRecurse strncmp %d\n",
148         strcmp(aSymbol, anAVLNode->symbol));
149
150   int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
151   if (aStrCmp < 0) {
152     // aSymbol < anAVLNode->symbol // insert in LEFT subtree
153     if (jInterp->debug) {
154       printDicInto(aStrBuf, anAVLNode, 10);
```

```
155       DEBUG(jInterp, ">-insert LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
156             anAVLNode, aSymbol, anAVLNode->balance,
157             anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
158             getCString(aStrBuf));
159       strBufClose(aStrBuf);
160     }
161     DictNodeObj* leftResult =
162       insertSymbolRecurse(aDict, anAVLNode->left, aSymbol);
163     assert(leftResult);
164     if (!anAVLNode->left) {
165       // we have inserted a new node ...
166       // ... insert this new node into the doubly linked list
167       //
168       DictNodeObj* oldPrevious           = anAVLNode->previous;
169       assert(aDict->firstSymbol);
170       if (oldPrevious) oldPrevious->next = leftResult;
171       else aDict->firstSymbol            = leftResult;
172       leftResult->next                   = anAVLNode;
173       leftResult->previous               = oldPrevious;
174       anAVLNode->previous                = leftResult;
175       //
176     }
177     anAVLNode->left = leftResult;
178     reCalculateAVLNodeHeightBalance(anAVLNode);
179     if (jInterp->debug) {
180       printDicInto(aStrBuf, anAVLNode, 10);
181       DEBUG(jInterp, "<-insert LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
182           anAVLNode, aSymbol, anAVLNode->balance,
183           anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
184           getCString(aStrBuf));
185       strBufClose(aStrBuf);
186     }
187     //
188     if (2 < anAVLNode->balance) {
189       assert(anAVLNode->left);
190       if (strcmp(aSymbol, anAVLNode->left->symbol) < 0) {
191         anAVLNode = rotateLeftLeft(aDict, anAVLNode);
192       } else {
193         anAVLNode = rotateLeftRight(aDict, anAVLNode);
194       }
195     }
196   } else if (0 < aStrCmp) {
197     // aSymbol > anAVLNode->symbol // insert in RIGHT subtree
```

```
198     if (jInterp->debug) {
199       printDicInto(aStrBuf, anAVLNode, 10);
200       DEBUG(jInterp, ">-insert RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
201           anAVLNode, aSymbol, anAVLNode->balance,
202           anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
203           getCString(aStrBuf));
204       strBufClose(aStrBuf);
205     }
206     DictNodeObj* rightResult =
207       insertSymbolRecurse(aDict, anAVLNode->right, aSymbol);
208     if (!anAVLNode->right) {
209       // we have inserted a new node ...
210       // ... insert this new node into the doubly linked list
211       //
212       DictNodeObj* oldNext           = anAVLNode->next;
213       if (oldNext) oldNext->previous = rightResult;
214       rightResult->previous          = anAVLNode;
215       rightResult->next              = oldNext;
216       anAVLNode->next                = rightResult;
217       //
218     }
219     anAVLNode->right = rightResult;
220     reCalculateAVLNodeHeightBalance(anAVLNode);
221     if (jInterp->debug) {
222       printDicInto(aStrBuf, anAVLNode, 10);
223       DEBUG(jInterp, "<-insert RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
224           anAVLNode, aSymbol, anAVLNode->balance,
225           anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
226           getCString(aStrBuf));
227       strBufClose(aStrBuf);
228     }
229     //
230     if (anAVLNode->balance < -2) {
231       assert(anAVLNode->right);
232       if (strcmp(aSymbol, anAVLNode->right->symbol) > 0) {
233         anAVLNode = rotateRightRight(aDict, anAVLNode);
234       } else {
235         anAVLNode = rotateRightLeft(aDict, anAVLNode);
236       }
237     }
238   } else {
239     // aSymbol == anAVLNode->symbol // nothing to do...
240     DEBUG(jInterp,"symols equal <%s>[%s]\n",
```

```
241           anAVLNode->symbol, aSymbol);
242   }
243
244   reCalculateAVLNodeHeightBalance(anAVLNode);
245   return anAVLNode;
246 }
247 DictNodeObj* deleteSymbolRecurseImpl(
248   DictObj     *aDict,
249   DictNodeObj *anAVLNode,
250   Symbol      *aSymbol
251 ) {
252   assert(aDict);
253   JoyLoLInterp *jInterp = aDict->jInterp;
254   assert(jInterp);

255   if (!anAVLNode) return NULL;
256
257   StringBufferObj *aStrBuf =
258     (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
259
260   DEBUG(jInterp, "\ndeleteSymbol %p <%s>[%s] %ld:%zu\n",
261         anAVLNode, anAVLNode->symbol, aSymbol,
262         anAVLNode->balance, anAVLNode->height);
263
264   DEBUG(jInterp, "deleteSymbol strncmp %d\n",
265         strcmp(aSymbol, anAVLNode->symbol));
266
267   int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
268   if (aStrCmp < 0) {
269     // aSymbol < anAVLNode->symbol // delete from LEFT subtree
270     if (jInterp->debug) {
271       printDicInto(aStrBuf, anAVLNode, 10);
272       DEBUG(jInterp, ">-delete LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
273             anAVLNode, aSymbol, anAVLNode->balance,
274             anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
275             getCString(aStrBuf));
276       strBufClose(aStrBuf);
277     }
278
279     anAVLNode->left =
280       deleteSymbolRecurse(aDict, anAVLNode->left, aSymbol);
281     reCalculateAVLNodeHeightBalance(anAVLNode);
282     if (jInterp->debug) {
```

```
283       printDicInto(aStrBuf, anAVLNode, 10);
284       DEBUG(jInterp, "<-delete LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
285           anAVLNode, aSymbol, anAVLNode->balance,
286           anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
287           getCString(aStrBuf));
288       strBufClose(aStrBuf);
289     }
290     //
291     if (anAVLNode->balance < -2) {
292       if (
293         anAVLNode->right &&
294         strcmp(aSymbol, anAVLNode->right->symbol) < 0
295       ) {
296         anAVLNode = rotateRightRight(aDict, anAVLNode);
297       } else {
298         anAVLNode = rotateRightLeft(aDict, anAVLNode);
299       }
300     }
301   } else if (0 < aStrCmp) {
302     // aSymbol > anAVLNode->symbol // delete in RIGHT subtree
303     if (jInterp->debug) {
304       printDicInto(aStrBuf, anAVLNode, 10);
305       DEBUG(jInterp, ">-delete RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
306           anAVLNode, aSymbol, anAVLNode->balance,
307           anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
308           getCString(aStrBuf));
309       strBufClose(aStrBuf);
310     }
311    anAVLNode->right =
312       deleteSymbolRecurse(aDict, anAVLNode->right, aSymbol);
313    reCalculateAVLNodeHeightBalance(anAVLNode);
314    if (jInterp->debug) {
315       printDicInto(aStrBuf, anAVLNode, 10);
316       DEBUG(jInterp, "<-delete RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
317           anAVLNode, aSymbol, anAVLNode->balance,
318           anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
319           getCString(aStrBuf));
320       strBufClose(aStrBuf);
321     }
322     //
323     if (2 < anAVLNode->balance) {
324       if (
325         anAVLNode->left &&
```

```
326         strcmp(aSymbol, anAVLNode->left->symbol) > 0
327       ) {
328         anAVLNode = rotateLeftLeft(aDict, anAVLNode);
329       } else {
330         anAVLNode = rotateLeftRight(aDict, anAVLNode);
331       }
332     }
333   } else {
334     //
335     // aSymbol == anAVLNode->symbol
336     //
337     DEBUG(jInterp,"symols equal <%s>[%s]\n",
338           anAVLNode->symbol, aSymbol);
339     if (anAVLNode->right) {
340       //
341       // we need to find the next node greater than this one (gtNode)
342       // copy it(gtNode) to this node
343       // and then delete it(gtNode) from right branch of this node
344       //
345       // SINCE we are copying nodes, the doubly linked list is still
346       // correct... except that it has a duplicate entry...
347       // so long as we have a right node... this duplication propogates
348       // to the right...
349       //
350       DictNodeObj *gtNode = anAVLNode->right;
351       while ( gtNode->left ) {
352         gtNode = gtNode->left;
353       }
354       copyDictNodeFromTo(jInterp, gtNode, anAVLNode);
355       anAVLNode->right =
356         deleteSymbolRecurse(aDict, anAVLNode->right, anAVLNode->symbol);
357       reCalculateAVLNodeHeightBalance(anAVLNode);
358       //
359       if (jInterp->debug) {
360         printDicInto(aStrBuf, anAVLNode, 10);
361         DEBUG(jInterp, "<-delete RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
362             anAVLNode, aSymbol, anAVLNode->balance,
363             anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
364             getCString(aStrBuf));
365         strBufClose(aStrBuf);
366       }
367       //
368       if (2 < anAVLNode->balance) {
```

```
369         if (
370           anAVLNode->left &&
371           strcmp(aSymbol, anAVLNode->left->symbol) > 0
372           ) {
373           anAVLNode = rotateLeftLeft(aDict, anAVLNode);
374         } else {
375           anAVLNode = rotateLeftRight(aDict, anAVLNode);
376         }
377       }
378     } else {
379       //
380       // this node will become unlinked from the AVL tree
381       // SO unlink this node from the doubly linked list
382       // as well...
383       //
384       DictNodeObj* oldPrevious           = anAVLNode->previous;
385       if (oldPrevious) oldPrevious->next = anAVLNode->next;
386       else aDict->firstSymbol            = anAVLNode->next;
387       if (anAVLNode->next) {
388         anAVLNode->next->previous        = oldPrevious;
389       }
390       //
391       // unlink this node..
392       //
393       anAVLNode->next     = NULL;
394       anAVLNode->previous = NULL;
395       //
396       return anAVLNode->left;
397     }
398   }
399
400   reCalculateAVLNodeHeightBalance(anAVLNode);
401   return anAVLNode;
402 }
403
404 DictNodeObj* rotateLeft(
405   DictObj     *aDict,
406   DictNodeObj *anAVLNode
407 ) {
408   assert(aDict);
409   JoyLoLInterp *jInterp = aDict->jInterp;
410   assert(jInterp);
```

```
411   StringBufferObj *aStrBuf =
412     (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
413   if (jInterp->debug) {
414     printDicInto(aStrBuf, anAVLNode, 10);
415     DEBUG(jInterp, ">-rotateLeft %p %ld:%zu=%zu %s\n",
416           anAVLNode, anAVLNode->balance, anAVLNode->height,
417           deepCalculateAVLNodeHeight(anAVLNode),
418           getCString(aStrBuf));
419     strBufClose(aStrBuf);
420   }
421   assert(anAVLNode->right);
422
423   DictNodeObj* newRoot = anAVLNode->right;
424   anAVLNode->right = newRoot->left;
425   newRoot->left    = anAVLNode;
426
427   reCalculateAVLNodeHeightBalance(anAVLNode);
428   reCalculateAVLNodeHeightBalance(newRoot);
429
430   if (jInterp->debug) {
431     printDicInto(aStrBuf, anAVLNode, 10);
432     DEBUG(jInterp, "<o-rotateLeft %p %ld:%zu=%zu %s\n",
433           anAVLNode, anAVLNode->balance, anAVLNode->height,
434           deepCalculateAVLNodeHeight(anAVLNode),
435           getCString(aStrBuf));
436     strBufClose(aStrBuf);
437     printDicInto(aStrBuf, newRoot, 10);
438     DEBUG(jInterp, "<n-rotateLeft %p %ld:%zu=%zu %s\n",
439           newRoot, newRoot->balance, newRoot->height,
440           deepCalculateAVLNodeHeight(newRoot),
441           getCString(aStrBuf));
442     strBufClose(aStrBuf);
443   }
444   assert(anAVLNode->height == deepCalculateAVLNodeHeight(anAVLNode));
445   assert(newRoot->height == deepCalculateAVLNodeHeight(newRoot));
446   return newRoot;
447 }
448
449 DictNodeObj* rotateRight(
450   DictObj     *aDict,
451   DictNodeObj *anAVLNode
452 ) {
453   assert(aDict);
```

```
454   JoyLoLInterp *jInterp = aDict->jInterp;
455   assert(jInterp);

456   StringBufferObj *aStrBuf =
457     (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
458   if (jInterp->debug) {
459     printDicInto(aStrBuf, anAVLNode, 10);
460     DEBUG(jInterp, ">-rotateRight %p %ld:%zu=%zu %s\n",
461           anAVLNode, anAVLNode->balance, anAVLNode->height,
462           deepCalculateAVLNodeHeight(anAVLNode),
463           getCString(aStrBuf));
464     strBufClose(aStrBuf);
465   }
466   assert(anAVLNode->left);
467
468   DictNodeObj* newRoot = anAVLNode->left;
469   anAVLNode->left = newRoot->right;
470   newRoot->right  = anAVLNode;
471
472   reCalculateAVLNodeHeightBalance(anAVLNode);
473   reCalculateAVLNodeHeightBalance(newRoot);
474
475   if (jInterp->debug) {
476     printDicInto(aStrBuf, anAVLNode, 10);
477     DEBUG(jInterp, "<o-rotateRight %p %ld:%zu=%zu %s\n",
478           anAVLNode, anAVLNode->balance, anAVLNode->height,
479           deepCalculateAVLNodeHeight(anAVLNode),
480           getCString(aStrBuf));
481     strBufClose(aStrBuf);
482     printDicInto(aStrBuf, newRoot, 10);
483     DEBUG(jInterp, "<n-rotateRight %p %ld:%zu=%zu %s\n",
484           newRoot, newRoot->balance, newRoot->height,
485           deepCalculateAVLNodeHeight(newRoot),
486           getCString(aStrBuf));
487     strBufClose(aStrBuf);
488   }
489   assert(anAVLNode->height == deepCalculateAVLNodeHeight(anAVLNode));
490   assert(newRoot->height == deepCalculateAVLNodeHeight(newRoot));
491   return newRoot;
492 }
493 DictNodeObj* rotateLeftLeft(
494   DictObj     *aDict,
495   DictNodeObj *anAVLNode
```

```
496 ) {
497   assert(aDict);
498   JoyLoLInterp *jInterp = aDict->jInterp;
499   assert(jInterp);

500   if (jInterp->debug) {
501     StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
502     printDicInto(aStrBuf, anAVLNode, 10);
503     DEBUG(jInterp, "LL %p %s\n",
504       anAVLNode, getCString(aStrBuf));
505     strBufClose(aStrBuf);
506   }
507   return rotateRight(aDict, anAVLNode);
508 }
509
510 DictNodeObj* rotateLeftRight(
511   DictObj     *aDict,
512   DictNodeObj *anAVLNode
513 ) {
514   assert(aDict);
515   JoyLoLInterp *jInterp = aDict->jInterp;
516   assert(jInterp);

517   StringBufferObj *aStrBuf =
518     (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
519   if (jInterp->debug) {
520     printDicInto(aStrBuf, anAVLNode, 10);
521     DEBUG(jInterp, "0-LR %p %s\n",
522       anAVLNode, getCString(aStrBuf));
523     strBufClose(aStrBuf);
524   }
525   anAVLNode->left = rotateLeft(aDict, anAVLNode->left);
526   if (jInterp->debug) {
527     printDicInto(aStrBuf, anAVLNode, 10);
528     DEBUG(jInterp, "1-LR %p %s\n",
529       anAVLNode, getCString(aStrBuf));
530     strBufClose(aStrBuf);
531   }
532   return rotateRight(aDict, anAVLNode);
533 }
534 DictNodeObj* rotateRightLeft(
535   DictObj     *aDict,
536   DictNodeObj *anAVLNode
```

```
537 ) {
538   assert(aDict);
539   JoyLoLInterp *jInterp = aDict->jInterp;
540   assert(jInterp);

541   StringBufferObj *aStrBuf =
542     (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
543   if (jInterp->debug) {
544     printDicInto(aStrBuf, anAVLNode, 10);
545     DEBUG(jInterp, "0-RL %p %s\n",
546       anAVLNode, getCString(aStrBuf));
547     strBufClose(aStrBuf);
548   }
549   anAVLNode->right = rotateRight(aDict, anAVLNode->right);
550   if (jInterp->debug) {
551     printDicInto(aStrBuf, anAVLNode, 10);
552     DEBUG(jInterp, "1-RL %p %s\n",
553       anAVLNode, getCString(aStrBuf));
554     strBufClose(aStrBuf);
555   }
556   return rotateLeft(aDict, anAVLNode);
557 }
558
559 DictNodeObj* rotateRightRight(
560   DictObj     *aDict,
561   DictNodeObj *anAVLNode
562 ) {
563   assert(aDict);
564   JoyLoLInterp *jInterp = aDict->jInterp;
565   assert(jInterp);

566   if (jInterp->debug) {
567     StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
568     printDicInto(aStrBuf, anAVLNode, 10);
569     DEBUG(jInterp, "RR %p %s\n",
570       anAVLNode, getCString(aStrBuf));
571     strBufClose(aStrBuf);
572   }
573   return rotateLeft(aDict, anAVLNode);
574 }
575
576 void reCalculateAVLNodeHeightBalance(DictNodeObj* anAVLNode) {
577   if (!anAVLNode) return;
```

```
578
579   if (!anAVLNode->left && !anAVLNode->right) {
580     anAVLNode->height  = 1;
581     anAVLNode->balance = 0;
582   } else if (!anAVLNode->left) {
583     anAVLNode->height  =  1 + anAVLNode->right->height;
584     anAVLNode->balance = -1 - anAVLNode->right->height;
585   } else if (!anAVLNode->right) {
586     anAVLNode->height  = 1 + anAVLNode->left->height;
587     anAVLNode->balance = 1 + anAVLNode->left->height;
588   } else if (anAVLNode->left->height < anAVLNode->right->height) {
589     anAVLNode->height  = 1 + anAVLNode->right->height;
590     anAVLNode->balance = anAVLNode->left->height - anAVLNode->right->height;
591   } else {
592     anAVLNode->height  = 1 + anAVLNode->left->height;
593     anAVLNode->balance = anAVLNode->left->height - anAVLNode->right->height;
594   }
595 }
596
597 size_t deepCalculateAVLNodeHeight(DictNodeObj* anAVLNode) {
598   if (!anAVLNode) return 0;
599
600   size_t leftHeight = 1 + deepCalculateAVLNodeHeight(anAVLNode->left);
601   size_t rightHeight = 1 + deepCalculateAVLNodeHeight(anAVLNode->right);
602
603   if (leftHeight > rightHeight) return leftHeight;
604   return rightHeight;
605 }
606
607 Boolean checkAVLNode(
608   JoyLoLInterp *jInterp,
609   DictNodeObj  *anAVLNode
610 ) {
611   assert(jInterp);

612   if (!anAVLNode) return TRUE;
613   if (jInterp->debug) {
614     StringBufferObj *aStrBuf =
615       newStringBuffer(jInterp->rootCtx);
616     printDicInto(aStrBuf, anAVLNode, 10);
617     DEBUG(jInterp, "checkAVLNode %p %ld:%zu=%zu %s\n",
618           anAVLNode, anAVLNode->balance, anAVLNode->height,
619           deepCalculateAVLNodeHeight(anAVLNode),
```

```
620           getCString(aStrBuf));
621     strBufClose(aStrBuf);
622   }
623
624   if (anAVLNode->left) {
625       DEBUG(jInterp, "car>-checkAVLNode %p\n", anAVLNode);
626       checkAVLNode(jInterp, anAVLNode->left);
627       assert(0 < strcmp(anAVLNode->symbol,
628                         anAVLNode->left->symbol));
629     DEBUG(jInterp, "car<-checkAVLNode %p\n", anAVLNode);
630   }
631
632   if (anAVLNode->right) {
633     DEBUG(jInterp, "cdr>-checkAVLNode %p\n", anAVLNode);
634     checkAVLNode(jInterp, anAVLNode->right);
635     assert(strcmp(anAVLNode->symbol,
636                   anAVLNode->right->symbol) < 0);
637     DEBUG(jInterp, "cdr<-checkAVLNode %p\n", anAVLNode);
638   }
639
640   assert(anAVLNode->height == deepCalculateAVLNodeHeight(anAVLNode));
641
642   return TRUE;
643 }
644
645 EndCoAlgebra
```

6 Compiler CoAlgebras

59 59

6.1 Compiler/Interpreter

6.1.1 Overview

The JoyLoL compiler/interpreter is a fairly standard multi-component compiler. However instead of compiling to machine code, it actually transpiles core functions to ANSI-C and interprets non-core functions by calling the core functions in order from the process stack.

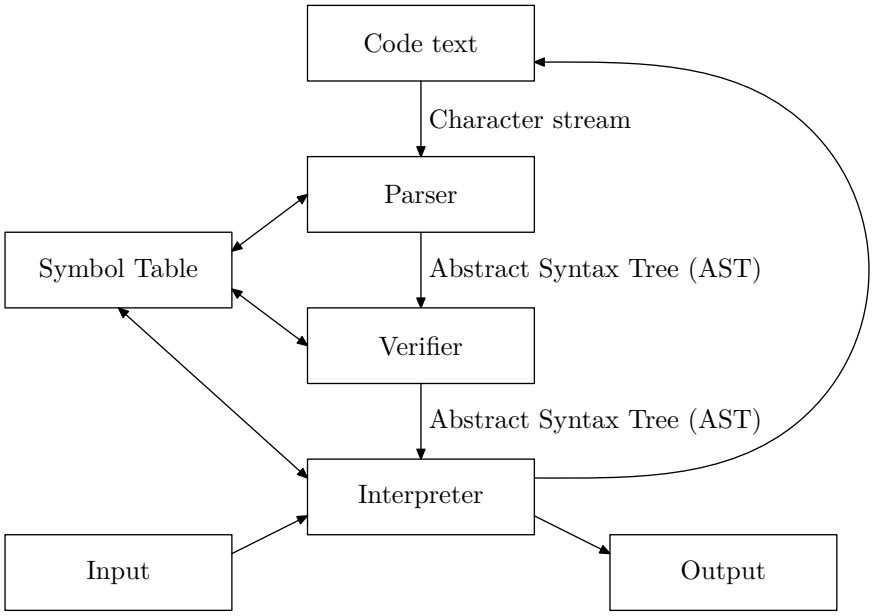

Figure 1.1 A high-level structure of the JoyLoL compiler/interpreter

As can be seen in figure 1.1, the compiler/interpreter consists of three main parts, a parser, a (correctness) verifier, and an interpreter, each with additional sub-structure.

- **Parser**
  As can be seen in figure 1.2, the parser is composed of a lexer, parser pair. Both the lexer and parser will use a push down automata composed by a JoyLoL Parsing Expression Grammar, jPeg. Using a push down automata means that, technically, the lexer is powerful enough to directly parse JoyLoL code. We use a standard lexer, parser pair to allow the lexer and parser's grammars to be kept simpler, both to understand as well as to be more performant.
  - **Lexer**

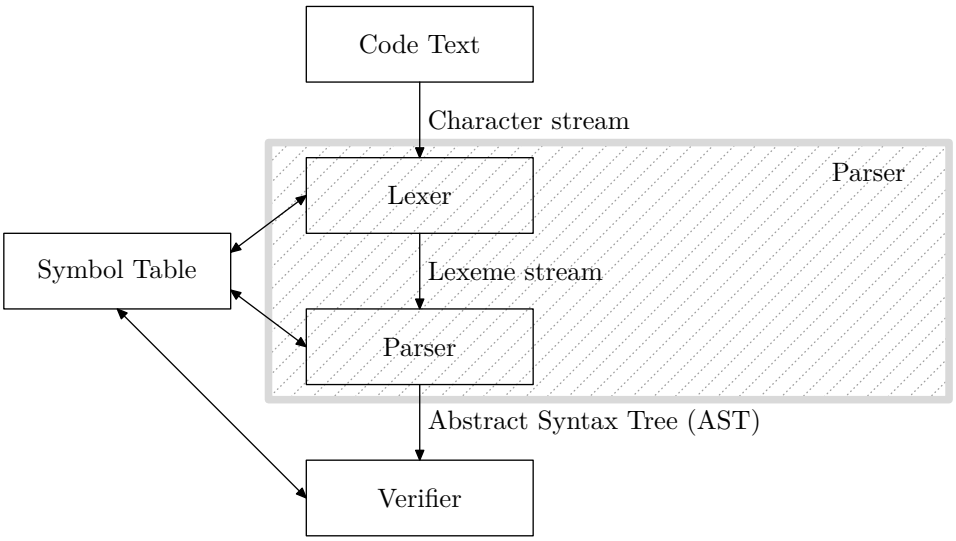

Figure 1.2 A high-level structure of the `JoyLoL` parser

The lexer extracts lexemes consisting of operators, identifiers, strings, comments, and numbers, from the character stream. Each lexeme is either categorized by or inserted into the symbol table. Any categorizations will be used by the parser.

- **Parser**
  The parser builds an abstract syntax tree (AST) using the categorized lexemes extracted by the lexer.

- **Verifier**
  As can be seen in figure 1.3, the verifier consists of a type inferencer and a correctness verifier.

  - **Type inferencer**
    The type inferencer ensures that all data on the stack as well as all variables in the register machine code has a definite type.

  - **Correctness verifier**
    The correctness verifier checks all implicit and explicit post conditions imply the corresponding pre conditions conditions in both the `JoyLoL` and register machine code. The invariants associated with each data type contribute implicit pre and post conditions for each `JoyLoL` statement.

- **Interpreter**
  As can be seen in figure 1.4, the interpreter consists of the code generator, ANSI-C compiler and the `JoyLoL` interpreter.

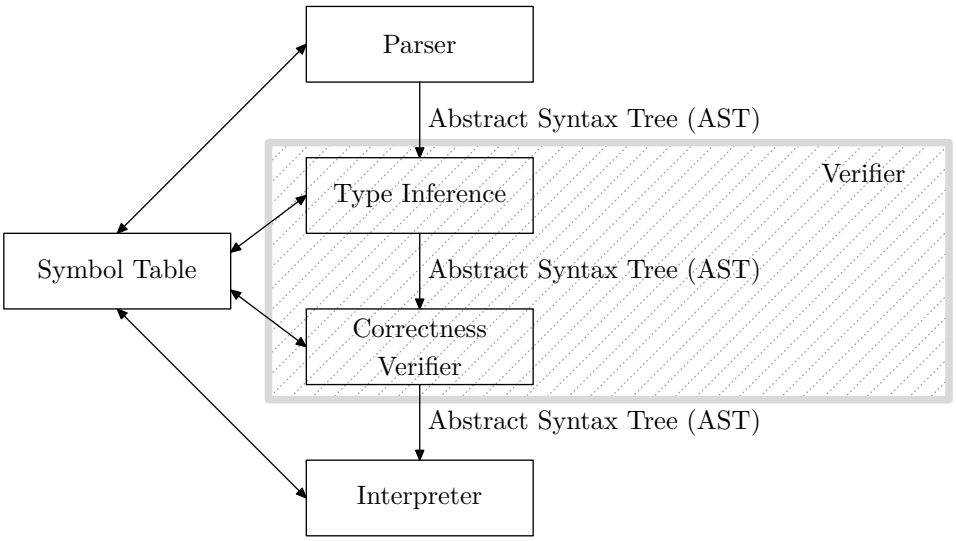

Figure 1.3 A high-level structure of the `JoyLoL` verifier

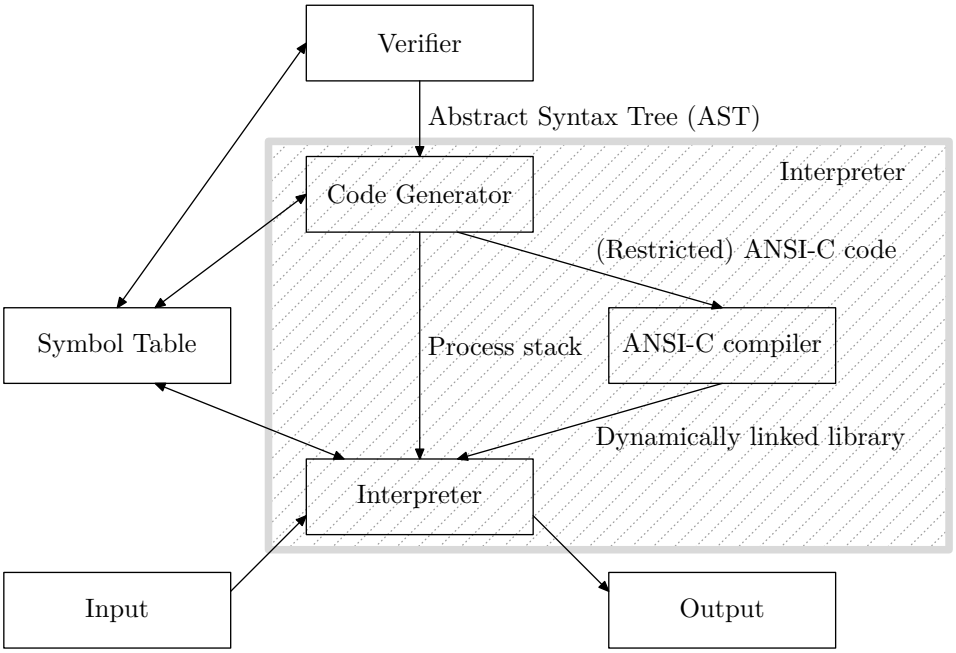

Figure 1.4 A high-level structure of the `JoyLoL` interpreter

– **Code generator**

The code generator determines whether any particular code fragment will be interpreted as a list of `JoyLoL` words, or as ANSI-C compiled code. If it decides that a code fragment can be compiled as ANSI-C code, the code generator assigns variables and optimizes the control flow graph to use ANSI-C if-then-else or for statements as appropriate.

Code which will be interpreted as a list of `JoyLoL` words are placed directly on the process stack using the entries in the symbol table. This ensures that the interpreter does not usually need to query the symbol table as it is interpreting.

- **ANSI-C compiler**
  The ANSI-C compiler compiles a restricted form of ANSI-C code generated by the code generator. These restricted forms ensure that only (relatively) well understood ANSI-C code is compiled, in order to maintain the correctness of the over all code. We ensure that 'risky' C code is not used in implementing `JoyLoL`.

  While there are a large number of ANSI-C compilers that could be used in any project, the typical compilers will be Gnu's gcc and LLVM's clang. If LLVM's clang is used, it is potentially possible to dynamically compile and load any resulting code. However, the whole system is designed to use a completely external compiler in a multi-phased compilation.

- **Interpreter**
  The interpreter calls the individual `JoyLoL` words taken from the process stack. Each call may alter either the data or process (or both) stacks as the result of its action.

6.1.2 Challenges

6.1.2.1 Sublanguages

Since our `JoyLoL` language is actually best structured as a pair of sublanguages, we need a way of 'switching' between these sublanguages *inside* a given `JoyLoL` text. Our overall strategy is to construct the compiler as a sequence of co-routines, which each consume, transform and the generate 'objects' along the sequence. To deal with multiple sublanguages we need the ability to deal with a 'sequence' of co-routines which does not have a *linear* topology. That is for some sections of the co-routine structure there will be multiple paths. To deal with this we need multiplexer co-routine which listens for a small collection of tokens and then chooses which sub-sequence along which to send all subsequence tokens. Conversely we need a reverse-multiplexer to fold tokens from multiple streams back into one stream.

The reverse-multiplexer will 'listen' on all of its 'input' streams waiting for the next token[9].

A subsequent question now becomes, where do we locate the multiplexer and reverse-multiplexer? For our purposes, locating the multiplexer depends upon whether or not the sublanguages (as well as the multiplexer) can share the 'lexer'[10]. If they can all share the same lexer then the multiplexer should be located between the single lexer and the multiple parsers[11].

6.1.2.2 Co-routines

Since co-routines will be an important way to structure complex mathematical computational proofs, we need to ensure switching between co-routines can be compiled into performant (ANSI-C) idioms.

6.1.3 Syntax/Semantics

For the initial version of the syntax for both the JoyLoL Configuratoion Language, and the JoyLoL Language itself, we make essential use of Bertrand Meyer's definition of the early version of Eiffel contained in [Mey92]. We will however make extensive chages to suit a new purpose. Eiffel's essentiall use of 'programming by contract' is however very close to one of our twin purposes for JoyLoL.

We structure the various parsers using ideas from Meyer's [Mey90].

6.1.3.1 The JoyLoL Configuration Language

Syntax

```
configuration = systemConfig
  defaults^0
  clusters^0
  externals^0
  generation
  end ("--" "system" systemName)^0
systemConfig = "system" systemName
systemName   = name
```

[9] This architecture seems to assume that we have an inherently sequential, non-parallel, computational system with only one co-routine 'running' at any one time.

[10] Note that, since the lexer will take care of rather large structures such as quoted strings and comments, sharing lexers *means* that the sublanguages *must* also share the syntax of these 'large' structures.

[11] Alternatively the composite parser includes the multiplexer and sub-language parsers.

```
defaults = "default" optionClause (";" optionClause)^0
options  = "option" optionClause (";" optionClause)^0
optionClause = optionTag optionMark^0 targetList^0
optionTag    = classTag + systemTag + freeTag
classTag     = "assertion" + "debug" + "optimize" + "trace"
systemTag    = "collect"
freeTag      = name
optionMark   = "(" optionValue")"
optionValue  = standardValue + classValue + freeValue
standardValue = "yes" + "no" + "all"
classValue    = "require" + "ensure" + "invariant" + "loop" + "check"
freeValue     = fileName + directoryName + name

clusters = "cluster" clusterClause (";" clusterClause)^0
clusterClause = clusterTag^0 directoryName clusterProperties^0
clusterTag    = clusterName ":"
clusterName   = name
clusterProperties = use^0
  include^0
  exclude^0
  nameAdaptation^0
  defaults^0
  options^0
  visibility^0
use = "use" fileName
include = "include" fileList
exclude = "exclude" fileList
fileList = fileName (";" fileName)^0
fileName = name

nameAdaptation = "adapt" clusterAdaptationList
clusterAdaptationList = clusterAdaptation (";" clusterAdaptation)^0
clusterAdaptation = clusterIgnore + clusterRenameClause
clusterIgnore = clusterName ":" "ignore"
clusterRenameClause = clusterName ":" "rename" classRenameList
classRenameList = classRenamePair ("," classRenamePair)^0
classRenamePair = className "as" className
className = name

visiblity = "visible" classVisibility (";" classVisibility )^0
classVisiblity = className visibilityAdaptation
visibilityAdaptation = externalClassRename^0
  creationRestriction^0
```

```
  exportRestriction^0
  externalFeatureRename^0
  "end"
externalClassRename = "as" name
creationRestriction = "creation" featureName ("," featureName)^0
exportRestriction = "export" featureName ("," featureName)^0
externalFeatureRename = "rename" featureRenameList
featureRenameList = featureRenamePair ("," featureRenamePair)^0
featureRenamePair = featureName "as" name
featureName = name

externals = "external" languageContribution (";" languageContributions)^0
languageContribution = language ":" fileList
language = "JoyLoL" +
  "C" + "C++" +
  "Executable" + "SharedLibrary" +
  "Object" +
  "Make" +
  "ConTeXt" +
  name

generation = "generate" languageGeneration (";" languageGeneration)^0
languageGeneration = language generateOption^0 ":" target
generateOption = "(" generateOptionValue ")"
generateOptionValue = "yes" + "no"
target = directoryPath + fileName

name = identifier + manifestString
```

6.1.3.2 The JoyLoL Language

```
classDeclaration =
  indexing^0
  classHeader
  formalGenerics^0
  obsolete^0
  inheritance^0
  creators^0
  features^0
  invariants^0
  "end" ( "--" "class" className)^0
```

```
indexing = "indexing" indexList
indexList = indexClause (";" indexClause)^0
indexClause = indexID^0 indexTerms
indexID = identifier ":"
indexTerms = indexValue ("," indexValue)^0
indexValue = identifier + manifestConstant

classHeader = headerMark^0 "class" className
headerMark = "deferred" + "expanded"
className = identifier

formalGenerics = "[" formalGenericList "]"
formalGenericList = formalGeneric ("," formalGeneric )^0
formalGeneric     = formalGenericName constraint^0
formalGenericName = identifier
constraint = "->" classType
classType = className actualGenerics^0
actualGenerics =
```

6.2 jPeg

jPeg

6.2.1 Overview

We develop a JoyLoL Parsing Expression Grammar (jPeg) modelled upon the virtual machine implementation of the Lua Parsing Expression Grammar (LPeg). The Lua version upon which we base our implementation can be found in [Ier08] and [MI08]. Particularly important for our work are the extensive correctness proofs provided by [MI08].

An important difference is that while LPeg's implementation is based upon an interpreter specific to the LPeg virtual machine code, we will use JoyLoL's interpreter directly. The backtracking stack used by LPeg is replaced by a careful use of JoyLoL's process stack. By implementing jPeg directly in JoyLoL, jPeg programs will benefit from any optimizations provided by the JoyLoL compiler.

We begin by looking at the PEG description of Peg's grammar, taken from [For04].

```
# Hierarchical syntax
Grammar    <- Spacing Definition+ EndOfFile
Definition <- Identifier LEFTARROW Expression
Expression <- Sequence (SLASH Sequence)*
Sequence   <- Prefix*
Prefix     <- (AND / NOT)? Suffix
Suffix     <- Primary (QUESTION / STAR / PLUS)?
Primary    <- Identifier !LEFTARROW
            / OPEN Expression CLOSE
            / Literal / Class / DOT

# Lexical syntax
Identifier <- IdentStart IdentCont* Spacing
IdentStart <- [a-zA-Z_]
IdentCont  <- IdentStart / [0-9]

Literal    <- ['] (!['] Char)* ['] Spacing
            / ["] (!["] Char)* ["] Spacing
Class      <- '[' (!']' Range)* ']' Spacing
Range      <- Char '-' Char / Char
Char       <- '\\' [nrt'"\[\]\\]
            / '\\' [0-2][0-7][0-7]
            / '\\' [0-7][0-7]?
```

```
            / !'\\' .

LEFTARROW  <- '<-' Spacing
SLASH      <- '/' Spacing
AND        <- '&' Spacing
NOT        <- '!' Spacing
QUESTION   <- '?' Spacing
STAR       <- '*' Spacing
PLUS       <- '+' Spacing
OPEN       <- '(' Spacing
CLOSE      <- ')' Spacing
DOT        <- '.' Spacing

Spacing    <- (Space / Comment)*
Comment    <- '#' (!EndOfLine .)* EndOfLine
Space      <- ' ' / '\t' / EndOfLine
EndOfLine  <- '\r\n' / '\n' / '\r'
EndOfFile  <- !.
```

We now look at the same PEG description of Peg's grammar, taken from [Ier08].

```
pattern     <- grammar / simplepatt
grammar     <- (nonterminal '<-' sp simplepatt)+
simplepatt  <- alternative ('/' sp alternative)*
alternative <- ([!&]? sp suffix)+
suffix      <- primary ([*+?] sp)*
primary     <-
  '(' sp pattern ')' sp / '.' sp / literal /
  charclass / nonterminal !'<-'
literal     <- ['] (!['] .)* ['] sp
charclass   <- '[' (!']' (. '-' . / .))* ']' sp
nonterminal <- [a-zA-Z]+ sp
sp          <- [ \t\n]*
```

Translated into LPeg:

```
local lp = require 'lpeg';
local P, S, R, V = lp.P, lp.S, lp.R, lp.V;

local lpeg = P {
  pattern     = V"grammar" + V"simplepatt" ;
  grammar     = ( V"nonterminal" * S'<-' * V"sp" * V"simplepatt" )^1 ;
  simplepatt  = V"alternative" * ( S'/' * V"sp" * V"alternative" )^0 ;
  alternative = ( (S'!' + S"&")^-1 * V"sp" * V"suffix" )^1 ;
```

```
  suffix      = V"primary" * ([*+?] * V"sp")^0 ;
  primary     =
    S'(' * V"sp" * V"pattern" * S')' * V"sp" +
    S'.' * V"sp" +
    V"literal" +
    V"charclass" +
    V"nonterminal" * -S'<-' ;
  literal     = S"'" * (-S"'" * .)^0 * S"'" * V"sp" ;
  charclass   = S'[' * (-S']' * (. S'-' . + .))^0 * S']' * V"sp" ;
  nonterminal = R("az", "AZ")^1 * V"sp" ;
  sp          = (S" " + S"\t" + S"\n")^0 ;
};
```

We need to implement the following `JoyLoL` words:

- `Fail`
- `Commit`
- `Choose`
- `RepeatAtLeast`
- `RepeatAtMost`
- `Char`
- `CharSet`
- `Any`

TODO: We have not covered `not`, `and`, CharSet difference, captures, or actions on captures.

As suggested in [Ier08] we might implement captures using a `Capture` call with three distinct arguments:

1. `begin n` where the actual capture begins n characters before the current character. This capture method should be called *as soon as* it is known that the current path will succeed.
2. `end` this marks the end of a capture
3. `full n` this is the same as a `begin n` immediately followed by an `end`.

For all of the above `JoyLoL` words, the data stack must include both the current text structure (which must include a indication of the current character) as well as the current collection of captures.

A compiler is a pipeline of co-routines. The parser might itself be a pipeline of co-routines, one for the lexer, one for the ultimate parser, but there could be numerous intermediary co-routines parsing more complex syntactic structures.

6.2.2 Code

1 Test Suite: newBoolean

CHeader : public

```
typedef JObj* (NewBoolean)(
  JoyLoLInterp*,
  Boolean
);

#define newBoolean(jInterp, aBool)      \
  (                                     \
    assert(getBooleansClass(jInterp)    \
      ->newBooleanFunc),                \
    (getBooleansClass(jInterp)          \
      ->newBooleanFunc(jInterp, aBool)) \
  )
#define BOOLEAN_FLAG_MASK 0x8L
#define asBoolean(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
extern JObj* newBooleanImpl(
  JoyLoLInterp* jInterp,
  Boolean aBoolean
);
```

CCode : default

```
JObj* newBooleanImpl(
  JoyLoLInterp* jInterp,
  Boolean aBoolean
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  JObj* result = newObject(jInterp, BooleansTag);
  assert(result);
```

```
  result->type  = jInterp->coAlgs[BooleansTag];
  if (aBoolean) {
    result->flags |= BOOLEAN_FLAG_MASK;
  } else {
    result->flags &= ~BOOLEAN_FLAG_MASK;
  }
  return result;
}
```

Test case

should create a new boolean

```
  AssertPtrNotNull(jInterp);

  JObj* aNewBoolean = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aNewBoolean);
  AssertPtrNotNull(asType(aNewBoolean));
  AssertIntEquals(asTag(aNewBoolean), BooleansTag);
  AssertIntTrue(asFlags(aNewBoolean));
  AssertIntTrue(isAtom(aNewBoolean));
  AssertIntTrue(isBoolean(aNewBoolean));
  AssertIntFalse(isPair(aNewBoolean));
```

Test case

print Boolean

```
  AssertPtrNotNull(jInterp);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* aLoL = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, aLoL);
  AssertStrEquals(getCString(aStrBuf), "true ");
  strBufClose(aStrBuf);

  aLoL = newBoolean(jInterp, FALSE);
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, aLoL);
  AssertStrEquals(getCString(aStrBuf), "false ");
```

```
  strBufClose(aStrBuf);
```

2 Test Suite: isBoolean

CHeader : public

```
#define isBoolean(aLoL)                \
  (                                    \
    (                                  \
      (aLoL) &&                        \
      asType(aLoL) &&                  \
      (asTag(aLoL) == BooleansTag)     \
    ) ?                                \
      TRUE :                           \
      FALSE                            \
  )
```

3 Test Suite: isTrue and isFalse

CHeader : public

```
#define isTrue(aLoL)                      \
  (                                       \
    (                                     \
      (aLoL) &&                           \
      asType(aLoL) &&                     \
      (asTag(aLoL) == BooleansTag) &&     \
      asBoolean(aLoL)                     \
    ) ?                                   \
      TRUE :                              \
      FALSE                               \
  )
#define isFalse(aLoL)                     \
  (                                       \
    (                                     \
      (!aLoL) ||                          \
      (!asType(aLoL)) ||                  \
      (asTag(aLoL) != BooleansTag) ||     \
      !asBoolean(aLoL)                    \
    ) ?                                   \
      TRUE :                              \
      FALSE                               \
```

```
22    )
```

Test case

should return appropriate boolean values

```
  JObj *aBool = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aBool);
  AssertPtrNotNull(asType(aBool));
  AssertIntEquals(asTag(aBool), BooleansTag);
  AssertIntTrue(asBoolean(aBool));
  AssertIntTrue(isTrue(aBool));
  AssertIntFalse(isFalse(aBool));
  aBool = newBoolean(jInterp, FALSE);
  AssertPtrNotNull(aBool);
  AssertPtrNotNull(asType(aBool));
  AssertIntEquals(asTag(aBool), BooleansTag);
  AssertIntFalse(asBoolean(aBool));
  AssertIntFalse(asBoolean(aBool));
  AssertIntFalse(isTrue(aBool));
  AssertIntTrue(isFalse(aBool));
```

CHeader : private

```
1  extern Boolean equalityBoolCoAlg(
2    JoyLoLInterp *jInterp,
3    JObj         *lolA,
4    JObj         *lolB,
5    size_t        timeToLive
6  );
```

CCode : default

```
1  Boolean equalityBoolCoAlg(
2    JoyLoLInterp *jInterp,
3    JObj         *lolA,
4    JObj         *lolB,
5    size_t        timeToLive
6  ) {
7    DEBUG(jInterp, "boolCoAlg-equal a:%p b:%p\n", lolA, lolB);
8    if (!lolA && !lolB) return TRUE;
9    if (!lolA && lolB)  return FALSE;
10   if (lolA  && !lolB) return FALSE;
11   if (asType(lolA) != asType(lolB)) return FALSE;
```

```
12    if (!asType(lolA)) return FALSE;
13    if (asTag(lolA)  != BooleansTag) return FALSE;
14    if (asBoolean(lolA) != asBoolean(lolB)) return FALSE;
15    return TRUE;
16  }
```

4 Test Suite: printing booleans

CHeader : private

```
1   extern Boolean printBoolCoAlg(
2     StringBufferObj *aStrBuf,
3     JObj            *aLoL,
4     size_t           timeToLive
5   );
```

CCode : default

```
1   Boolean printBoolCoAlg(
2     StringBufferObj *aStrBuf,
3     JObj            *aLoL,
4     size_t           timeToLive
5   ) {
6     assert(aLoL);
7     assert(asTag(aLoL) == BooleansTag);
8
9     if (asBoolean(aLoL)) strBufPrintf(aStrBuf, "true ");
10    else strBufPrintf(aStrBuf, "false ");
11    return TRUE;
12  }
```

Test case

should print booleans

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[BooleansTag]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* aNewBoolean = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aNewBoolean);
```

```
  printLoL(aStrBuf, aNewBoolean);
  AssertStrEquals(getCString(aStrBuf), "true ");
  strBufClose(aStrBuf);

  aNewBoolean = newBoolean(jInterp, FALSE);
  AssertPtrNotNull(aNewBoolean);
  printLoL(aStrBuf, aNewBoolean);
  AssertStrEquals(getCString(aStrBuf), "false ");
  strBufClose(aStrBuf);
```

5 Test Suite: registerBooleans

CHeader : public

```
1 typedef struct booleans_class_struct {
2   JClass       super;
3   NewBoolean      *newBooleanFunc;
4 } BooleansClass;
```

CCode : default

```
1 static Boolean initializeBooleans(
2   JoyLoLInterp *jInterp,
3   JClass   *aJClass
4 ) {
5   assert(jInterp);
6   assert(aJClass);
7   return TRUE;
8 }
```

CHeader : private

```
1 extern Boolean registerBooleans(JoyLoLInterp *jInterp);
```

CCode : default

```
1 Boolean registerBooleans(JoyLoLInterp *jInterp) {
2   assert(jInterp);
3   assert(jInterp->coAlgs);

4   BooleansClass* theCoAlg
5     = joyLoLCalloc(1, BooleansClass);
6   assert(theCoAlg);
```

```
7   theCoAlg->super.name           = BooleansName;
8   theCoAlg->super.objectSize     = sizeof(JObj);
9   theCoAlg->super.initializeFunc = initializeBooleans;
10  theCoAlg->super.registerFunc   = registerBooleanWords;
11  theCoAlg->super.equalityFunc   = equalityBoolCoAlg;
12  theCoAlg->super.printFunc      = printBoolCoAlg;
13  theCoAlg->newBooleanFunc       = newBooleanImpl;
14  size_t tag =
15    registerJClass(jInterp, (JClass*)theCoAlg);

16  // do a sanity check...
17  assert(tag == BooleansTag);
18  assert(jInterp->coAlgs[tag]);

19  return TRUE;
20 }
```

Test case

should register the Booleans coAlg

```
// CTestsSetup has already created a jInterp
// and run registerBooleans
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getBooleansClass(jInterp));
BooleansClass *coAlg = getBooleansClass(jInterp);
registerBooleans(jInterp);
AssertPtrNotNull(getBooleansClass(jInterp));
AssertPtrEquals(getBooleansClass(jInterp), coAlg);
AssertIntEquals(
  getBooleansClass(jInterp)->super.objectSize,
  sizeof(JObj)
)
```

6.2.3 Words

true true (dataStack) (TRUE dataStack)
ansic ((pushData true))
false false (dataStack) (false dataStack)
ansic ((pushData false))

isTrue isTrue-true (true dataStack) (true dataStack)
isTrue-else ((top) dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData true) (pushData false)))
ifFalse
ifFalse-false (false dataStack) (true dataStack)
ifFalse-else ((top) dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData false) (pushData true)))
not not-true (false dataStack) (true dataStack)
not-false (true dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData false) (pushData true)))
or or-true1 (true (top2) dataStack) (true dataStack)
or-true2 (false true dataStack) (true dataStack)
or-false (false false dataStack) (false dataStack)
ansic ((popData aBool1) (popData aBool2) (doIfte aBool1 (pushData true) (doIfte aBool2 (pushData true) (pushData false))))
and
and-true (true true dataStack) (true dataStack)
and-false1 (false (top2) dataStack) (false dataStack)
and-false2 (true false dataStack) (false dataStack)
ansic ((popData aBool1) (popData aBool2) (doIfte aBool1 (doIfte aBool2 (pushData true) (pushData false)) (pushData false)))
isBoolean
isBoolean-true ((top aType) dataStack) (processStack) dup isTrue or isFalse << WRONG! ((TRUE Boolean) dataStack) (processStack)
isBoolean-false ((top aType) dataStack) (processStack) (top isBoolean not) ((FALSE Boolean) dataStack) (processStack)

CCode : default

```
1  static void isBooleanAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isBoolean(top))
8      result = newBoolean(jInterp, TRUE);
9    else
10     result = newBoolean(jInterp, FALSE);
11   pushCtxData(aCtx, result);
12 }
```

isTrue

CCode : default

```
static void isTrueAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isTrue(top)) {
    result = newBoolean(jInterp, TRUE);
  } else {
    result = newBoolean(jInterp, FALSE);
  }
  pushCtxData(aCtx, result);
}
```

isFalse

CCode : default

```
static void isFalseAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isTrue(top)) {
    result = newBoolean(jInterp, FALSE);
  } else {
    result = newBoolean(jInterp, TRUE);
  }
  pushCtxData(aCtx, result);
}
```

CHeader : private

```
extern Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
```

```
  ContextObj *rootCtx = jInterp->rootCtx;
  assert(rootCtx);
  DictObj *dict = rootCtx->dict;
  assert(dict);

  DictNodeObj* true  = getSymbolEntry(dict, "true");
  true->value = newBoolean(jInterp, TRUE);

  DictNodeObj* false = getSymbolEntry(dict, "false");
  false->value = newBoolean(jInterp, FALSE);

  extendJoyLoLInRoot(jInterp, "isBoolean", "", isBooleanAP, "");
  extendJoyLoLInRoot(jInterp, "isTrue",    "", isTrueAP,    "");
  extendJoyLoLInRoot(jInterp, "isFalse",   "", isFalseAP,   "");

  return TRUE;
}
```

6.2.4 Lua functions

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_booleans_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
```

```
}

static const struct luaL_Reg lua_booleans [] = {
  {"gitVersion", lua_booleans_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_booleans (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerBooleans(jInterp);
  luaL_newlib(lstate, lua_booleans);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedBooleans` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_booleans(lstate);
  lua_setfield(lstate, -2, "joylol.booleans");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

6.2.5 Specifying the lmsfile

CHeader : public

CHeader : private

```
1  extern size_t joylol_register_booleans(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <joylol/jInterps.h>
5  #include <joylol/stringBuffers.h>
6  #include <joylol/dictNodes.h>
7  #include <joylol/dictionaries.h>
8  #include <joylol/texts.h>
9  #include <joylol/cFunctions.h>
10 #include <joylol/assertions.h>
11 #include <joylol/contexts.h>
12 #include <joylol/booleans.h>
13 #include <joylol/booleans-private.h>
14 // dictionary
15 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.contexts");
requireStaticallyLinkedBooleans(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

6.3 Lexer

lexer

6.3.1 Overview

The lexer extracts lexemes consisting of white space, operators, identifiers, strings, comments, and numbers, from the character stream. Each lexeme is either categorized by or inserted into the symbol table. Any categorizations will be used by the parser. White space which is not inside a string and comments are ignored.

An LPeg grammar for our lexer is:

```
local lpeg = require 'lpeg';
local P, S, V = lpeg.P, lpeg.S, lpeg.V;

local lexer = P {
  lexeme     = V"naturalNum" +
               V"identifier" +
               V"whiteSpace" ;
  naturalNum = V"hexNumber" + V"decimalNum" ;
  hexNumber  = S'0' * (S'x' + S'X') * R('09', 'af', 'AF', ',')^1 ;
  decimalNum = R('19') * R('09')^0 ;
  identifier = R('az', 'AZ') * ( R('az', 'AZ', '09', '_') )^0 ;
  whiteSpace = (S' ' + S'\t' + S'\n')^0 ;
};
```

TODO: Rework this to be UTF8 friendly.

Following [Ier08] this can be translated into `JoyLoL` as:

```
; lexeme (right associative)
(
  (
    naturalNum
    identifier
    whiteSpace
  )
  Choose
)
'lexeme'
define

; naturalNum
```

```
(
  (
    hexNumber
    decimalNum
  )
  Choose
)
'naturalNum'
define

; hexNumber
(
  '0'
  Char
  ('x' 'X')
  CharSet
  (
    ('09' 'af' 'AF' ',')
    CharSet
  )
  1
  RepeatAtLeast
)
'hexNumber'
define

; decimalNum
(
  ('19')
  CharSet
  (
    ('09')
    CharSet
  )
  0
  RepeatAtLeast
)
'hexNumber'
define

; identifier
(
  ('az' 'AZ' '_')
```

```
  CharSet
  (
    ('09' 'az' 'AZ' '_')
    CharSet
  )
  0
  RepeatAtLeast
)
'identifier'
define

; whitespace
(
  (
    ( ' ' '\t' '\n')
    CharSet
  )
  0
  RepeatAtLeast
)
'whiteSpace'
define
```

6.3.2 Code

1 Test Suite: newBoolean

CHeader : public

```
typedef JObj* (NewBoolean)(
  JoyLoLInterp*,
  Boolean
);

#define newBoolean(jInterp, aBool)      \
  (                                     \
    assert(getBooleansClass(jInterp)    \
      ->newBooleanFunc),                \
    (getBooleansClass(jInterp)          \
      ->newBooleanFunc(jInterp, aBool)) \
  )
#define BOOLEAN_FLAG_MASK 0x8L
```

```
14 #define asBoolean(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
1 extern JObj* newBooleanImpl(
2   JoyLoLInterp* jInterp,
3   Boolean aBoolean
4 );
```

CCode : default

```
1  JObj* newBooleanImpl(
2    JoyLoLInterp* jInterp,
3    Boolean aBoolean
4  ) {
5    assert(jInterp);
6    assert(jInterp->coAlgs);

7    JObj* result = newObject(jInterp, BooleansTag);
8    assert(result);

9    result->type  = jInterp->coAlgs[BooleansTag];
10   if (aBoolean) {
11     result->flags |= BOOLEAN_FLAG_MASK;
12   } else {
13     result->flags &= ~BOOLEAN_FLAG_MASK;
14   }
15   return result;
16 }
```

Test case

should create a new boolean

```
  AssertPtrNotNull(jInterp);

  JObj* aNewBoolean = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aNewBoolean);
  AssertPtrNotNull(asType(aNewBoolean));
  AssertIntEquals(asTag(aNewBoolean), BooleansTag);
  AssertIntTrue(asFlags(aNewBoolean));
  AssertIntTrue(isAtom(aNewBoolean));
  AssertIntTrue(isBoolean(aNewBoolean));
  AssertIntFalse(isPair(aNewBoolean));
```

Test case

print Boolean

```
AssertPtrNotNull(jInterp);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

JObj* aLoL = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, aLoL);
AssertStrEquals(getCString(aStrBuf), "true ");
strBufClose(aStrBuf);

aLoL = newBoolean(jInterp, FALSE);
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, aLoL);
AssertStrEquals(getCString(aStrBuf), "false ");
strBufClose(aStrBuf);
```

2 Test Suite: isBoolean

CHeader : public

```
#define isBoolean(aLoL)                \
  (                                    \
    (                                  \
      (aLoL) &&                        \
      asType(aLoL) &&                  \
      (asTag(aLoL) == BooleansTag)     \
    ) ?                                \
      TRUE :                           \
      FALSE                            \
  )
```

3 Test Suite: isTrue and isFalse

CHeader : public

```
#define isTrue(aLoL)                    \
```

Code 6.3.2

```
  (                                    \
    (                                  \
      (aLoL) &&                        \
      asType(aLoL) &&                  \
      (asTag(aLoL) == BooleansTag) &&  \
      asBoolean(aLoL)                  \
    ) ?                                \
      TRUE :                           \
      FALSE                            \
  )
#define isFalse(aLoL)                  \
  (                                    \
    (                                  \
      (!aLoL) ||                       \
      (!asType(aLoL)) ||               \
      (asTag(aLoL) != BooleansTag) ||  \
      !asBoolean(aLoL)                 \
    ) ?                                \
      TRUE :                           \
      FALSE                            \
  )
```

Test case

should return appropriate boolean values

```
  JObj *aBool = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aBool);
  AssertPtrNotNull(asType(aBool));
  AssertIntEquals(asTag(aBool), BooleansTag);
  AssertIntTrue(asBoolean(aBool));
  AssertIntTrue(isTrue(aBool));
  AssertIntFalse(isFalse(aBool));
  aBool = newBoolean(jInterp, FALSE);
  AssertPtrNotNull(aBool);
  AssertPtrNotNull(asType(aBool));
  AssertIntEquals(asTag(aBool), BooleansTag);
  AssertIntFalse(asBoolean(aBool));
  AssertIntFalse(asBoolean(aBool));
  AssertIntFalse(isTrue(aBool));
  AssertIntTrue(isFalse(aBool));
```

CHeader : private

```
extern Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "boolCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != BooleansTag) return FALSE;
  if (asBoolean(lolA) != asBoolean(lolB)) return FALSE;
  return TRUE;
}
```

4 Test Suite: printing booleans

CHeader : private

```
extern Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
```

Code 6.3.2

```
6   assert(aLoL);
7   assert(asTag(aLoL) == BooleansTag);
8
9   if (asBoolean(aLoL)) strBufPrintf(aStrBuf, "true ");
10  else strBufPrintf(aStrBuf, "false ");
11  return TRUE;
12 }
```

Test case

should print booleans

```
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs[BooleansTag]);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

JObj* aNewBoolean = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aNewBoolean);
printLoL(aStrBuf, aNewBoolean);
AssertStrEquals(getCString(aStrBuf), "true ");
strBufClose(aStrBuf);

aNewBoolean = newBoolean(jInterp, FALSE);
AssertPtrNotNull(aNewBoolean);
printLoL(aStrBuf, aNewBoolean);
AssertStrEquals(getCString(aStrBuf), "false ");
strBufClose(aStrBuf);
```

5 Test Suite: registerBooleans

CHeader : public

```
1  typedef struct booleans_class_struct {
2    JClass       super;
3    NewBoolean      *newBooleanFunc;
4  } BooleansClass;
```

CCode : default

```
1  static Boolean initializeBooleans(
2    JoyLoLInterp *jInterp,
```

```
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerBooleans(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerBooleans(JoyLoLInterp *jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  BooleansClass* theCoAlg
    = joyLoLCalloc(1, BooleansClass);
  assert(theCoAlg);

  theCoAlg->super.name           = BooleansName;
  theCoAlg->super.objectSize     = sizeof(JObj);
  theCoAlg->super.initializeFunc = initializeBooleans;
  theCoAlg->super.registerFunc   = registerBooleanWords;
  theCoAlg->super.equalityFunc   = equalityBoolCoAlg;
  theCoAlg->super.printFunc      = printBoolCoAlg;
  theCoAlg->newBooleanFunc       = newBooleanImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == BooleansTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Booleans coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerBooleans
```

```
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getBooleansClass(jInterp));
BooleansClass *coAlg = getBooleansClass(jInterp);
registerBooleans(jInterp);
AssertPtrNotNull(getBooleansClass(jInterp));
AssertPtrEquals(getBooleansClass(jInterp), coAlg);
AssertIntEquals(
  getBooleansClass(jInterp)->super.objectSize,
  sizeof(JObj)
)
```

6.3.3 Words

true true (dataStack) (TRUE dataStack)
ansic ((pushData true))
false false (dataStack) (false dataStack)
ansic ((pushData false))
isTrue isTrue-true (true dataStack) (true dataStack)
isTrue-else ((top) dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData true) (pushData false)))
ifFalse
ifFalse-false (false dataStack) (true dataStack)
ifFalse-else ((top) dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData false) (pushData true)))
not not-true (false dataStack) (true dataStack)
not-false (true dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData false) (pushData true)))
or or-true1 (true (top2) dataStack) (true dataStack)
or-true2 (false true dataStack) (true dataStack)
or-false (false false dataStack) (false dataStack)
ansic ((popData aBool1) (popData aBool2) (doIfte aBool1 (pushData true) (doIfte aBool2 (pushData true) (pushData false))))
and
and-true (true true dataStack) (true dataStack)
and-false1 (false (top2) dataStack) (false dataStack)
and-false2 (true false dataStack) (false dataStack)
ansic ((popData aBool1) (popData aBool2) (doIfte aBool1 (doIfte aBool2 (pushData true) (pushData false)) (pushData false)))
isBoolean
isBoolean-true ((top aType) dataStack) (processStack) dup isTrue or isFalse << WRONG! ((TRUE Boolean) dataStack) (processStack)

isBoolean-false ((top aType) dataStack) (processStack) (top isBoolean not) ((FALSE Boolean) dataStack) (processStack)

CCode : default

```
1  static void isBooleanAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isBoolean(top))
8      result = newBoolean(jInterp, TRUE);
9    else
10     result = newBoolean(jInterp, FALSE);
11   pushCtxData(aCtx, result);
12 }
```

isTrue

CCode : default

```
1  static void isTrueAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isTrue(top)) {
8      result = newBoolean(jInterp, TRUE);
9    } else {
10     result = newBoolean(jInterp, FALSE);
11   }
12   pushCtxData(aCtx, result);
13 }
```

isFalse

CCode : default

```
1  static void isFalseAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isTrue(top)) {
8      result = newBoolean(jInterp, FALSE);
```

```
  } else {
    result = newBoolean(jInterp, TRUE);
  }
  pushCtxData(aCtx, result);
}
```

CHeader : private

```
extern Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  ContextObj *rootCtx = jInterp->rootCtx;
  assert(rootCtx);
  DictObj *dict = rootCtx->dict;
  assert(dict);

  DictNodeObj* true  = getSymbolEntry(dict, "true");
  true->value = newBoolean(jInterp, TRUE);

  DictNodeObj* false = getSymbolEntry(dict, "false");
  false->value = newBoolean(jInterp, FALSE);

  extendJoyLoLInRoot(jInterp, "isBoolean", "", isBooleanAP, "");
  extendJoyLoLInRoot(jInterp, "isTrue",    "", isTrueAP,    "");
  extendJoyLoLInRoot(jInterp, "isFalse",   "", isFalseAP,   "");

  return TRUE;
}
```

6.3.4 Lua functions

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
```

```
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_booleans_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_booleans [] = {
  {"gitVersion", lua_booleans_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_booleans (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerBooleans(jInterp);
  luaL_newlib(lstate, lua_booleans);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedBooleans` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_booleans(lstate);
  lua_setfield(lstate, -2, "joylol.booleans");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

6.3.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_booleans(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
#include <joylol/dictNodes.h>
#include <joylol/dictionaries.h>
#include <joylol/texts.h>
#include <joylol/cFunctions.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/booleans.h>
#include <joylol/booleans-private.h>
// dictionary
// printer
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
  requireLuaModule(lstate, "joylol.assertions");
```

```
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.contexts");
requireStaticallyLinkedBooleans(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

6.4 Parser

parser

6.4.1 Overview

The parser builds an abstract syntax tree (AST) using the categorized lexemes extracted by the lexer.

6.4.2 Code

1 Test Suite: newBoolean

CHeader : public

```
typedef JObj* (NewBoolean)(
  JoyLoLInterp*,
  Boolean
);

#define newBoolean(jInterp, aBool)      \
  (                                     \
    assert(getBooleansClass(jInterp)    \
      ->newBooleanFunc),                \
    (getBooleansClass(jInterp)          \
      ->newBooleanFunc(jInterp, aBool)) \
  )
#define BOOLEAN_FLAG_MASK 0x8L
#define asBoolean(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
extern JObj* newBooleanImpl(
  JoyLoLInterp* jInterp,
  Boolean aBoolean
);
```

CCode : default

```
JObj* newBooleanImpl(
  JoyLoLInterp* jInterp,
  Boolean aBoolean
) {
```

Code 6.4.2

```
  assert(jInterp);
  assert(jInterp->coAlgs);

  JObj* result = newObject(jInterp, BooleansTag);
  assert(result);

  result->type  = jInterp->coAlgs[BooleansTag];
  if (aBoolean) {
    result->flags |= BOOLEAN_FLAG_MASK;
  } else {
    result->flags &= ~BOOLEAN_FLAG_MASK;
  }
  return result;
}
```

Test case

should create a new boolean

```
  AssertPtrNotNull(jInterp);

  JObj* aNewBoolean = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aNewBoolean);
  AssertPtrNotNull(asType(aNewBoolean));
  AssertIntEquals(asTag(aNewBoolean), BooleansTag);
  AssertIntTrue(asFlags(aNewBoolean));
  AssertIntTrue(isAtom(aNewBoolean));
  AssertIntTrue(isBoolean(aNewBoolean));
  AssertIntFalse(isPair(aNewBoolean));
```

Test case

print Boolean

```
  AssertPtrNotNull(jInterp);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* aLoL = newBoolean(jInterp, TRUE);
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, aLoL);
  AssertStrEquals(getCString(aStrBuf), "true ");
```

```
strBufClose(aStrBuf);

aLoL = newBoolean(jInterp, FALSE);
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, aLoL);
AssertStrEquals(getCString(aStrBuf), "false ");
strBufClose(aStrBuf);
```

2 Test Suite: isBoolean

CHeader : public

```
#define isBoolean(aLoL)                \
  (                                    \
    (                                  \
      (aLoL) &&                        \
      asType(aLoL) &&                  \
      (asTag(aLoL) == BooleansTag)     \
    ) ?                                \
      TRUE :                           \
      FALSE                            \
  )
```

3 Test Suite: isTrue and isFalse

CHeader : public

```
#define isTrue(aLoL)                     \
  (                                      \
    (                                    \
      (aLoL) &&                          \
      asType(aLoL) &&                    \
      (asTag(aLoL) == BooleansTag) &&    \
      asBoolean(aLoL)                    \
    ) ?                                  \
      TRUE :                             \
      FALSE                              \
  )
#define isFalse(aLoL)                    \
  (                                      \
    (                                    \
      (!aLoL) ||                         \
```

```
    (!asType(aLoL)) ||              \
    (asTag(aLoL) != BooleansTag) || \
    !asBoolean(aLoL)                \
  ) ?                               \
    TRUE :                          \
    FALSE                           \
)
```

Test case

should return appropriate boolean values

```
JObj *aBool = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aBool);
AssertPtrNotNull(asType(aBool));
AssertIntEquals(asTag(aBool), BooleansTag);
AssertIntTrue(asBoolean(aBool));
AssertIntTrue(isTrue(aBool));
AssertIntFalse(isFalse(aBool));
aBool = newBoolean(jInterp, FALSE);
AssertPtrNotNull(aBool);
AssertPtrNotNull(asType(aBool));
AssertIntEquals(asTag(aBool), BooleansTag);
AssertIntFalse(asBoolean(aBool));
AssertIntFalse(asBoolean(aBool));
AssertIntFalse(isTrue(aBool));
AssertIntTrue(isFalse(aBool));
```

CHeader : private

```
extern Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
```

```
) {
  DEBUG(jInterp, "boolCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != BooleansTag) return FALSE;
  if (asBoolean(lolA) != asBoolean(lolB)) return FALSE;
  return TRUE;
}
```

4 Test Suite: printing booleans

CHeader : private

```
extern Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == BooleansTag);

  if (asBoolean(aLoL)) strBufPrintf(aStrBuf, "true ");
  else strBufPrintf(aStrBuf, "false ");
  return TRUE;
}
```

Test case

should print booleans

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[BooleansTag]);
```

Code 6.4.2

```
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

JObj* aNewBoolean = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aNewBoolean);
printLoL(aStrBuf, aNewBoolean);
AssertStrEquals(getCString(aStrBuf), "true ");
strBufClose(aStrBuf);

aNewBoolean = newBoolean(jInterp, FALSE);
AssertPtrNotNull(aNewBoolean);
printLoL(aStrBuf, aNewBoolean);
AssertStrEquals(getCString(aStrBuf), "false ");
strBufClose(aStrBuf);
```

5 Test Suite: registerBooleans

CHeader : public

```
typedef struct booleans_class_struct {
  JClass       super;
  NewBoolean      *newBooleanFunc;
} BooleansClass;
```

CCode : default

```
static Boolean initializeBooleans(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerBooleans(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerBooleans(JoyLoLInterp *jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);
```

```
4   BooleansClass* theCoAlg
5     = joyLoLCalloc(1, BooleansClass);
6   assert(theCoAlg);

7   theCoAlg->super.name           = BooleansName;
8   theCoAlg->super.objectSize     = sizeof(JObj);
9   theCoAlg->super.initializeFunc = initializeBooleans;
10  theCoAlg->super.registerFunc   = registerBooleanWords;
11  theCoAlg->super.equalityFunc   = equalityBoolCoAlg;
12  theCoAlg->super.printFunc      = printBoolCoAlg;
13  theCoAlg->newBooleanFunc       = newBooleanImpl;
14  size_t tag =
15    registerJClass(jInterp, (JClass*)theCoAlg);

16  // do a sanity check...
17  assert(tag == BooleansTag);
18  assert(jInterp->coAlgs[tag]);

19  return TRUE;
20 }
```

Test case

should register the Booleans coAlg

```
// CTestsSetup has already created a jInterp
// and run registerBooleans
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getBooleansClass(jInterp));
BooleansClass *coAlg = getBooleansClass(jInterp);
registerBooleans(jInterp);
AssertPtrNotNull(getBooleansClass(jInterp));
AssertPtrEquals(getBooleansClass(jInterp), coAlg);
AssertIntEquals(
  getBooleansClass(jInterp)->super.objectSize,
  sizeof(JObj)
)
```

6.4.3 Words

true true (dataStack) (TRUE dataStack)
ansic ((pushData true))
false false (dataStack) (false dataStack)
ansic ((pushData false))
isTrue isTrue-true (true dataStack) (true dataStack)
isTrue-else ((top) dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData true) (pushData false)))
ifFalse
ifFalse-false (false dataStack) (true dataStack)
ifFalse-else ((top) dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData false) (pushData true)))
not not-true (false dataStack) (true dataStack)
not-false (true dataStack) (false dataStack)
ansic ((popData aBool) (doIfte aBool (pushData false) (pushData true)))
or or-true1 (true (top2) dataStack) (true dataStack)
or-true2 (false true dataStack) (true dataStack)
or-false (false false dataStack) (false dataStack)
ansic ((popData aBool1) (popData aBool2) (doIfte aBool1 (pushData true) (doIfte aBool2 (pushData true) (pushData false))))
and
and-true (true true dataStack) (true dataStack)
and-false1 (false (top2) dataStack) (false dataStack)
and-false2 (true false dataStack) (false dataStack)
ansic ((popData aBool1) (popData aBool2) (doIfte aBool1 (doIfte aBool2 (pushData true) (pushData false)) (pushData false)))
isBoolean
isBoolean-true ((top aType) dataStack) (processStack) dup isTrue or isFalse << WRONG! ((TRUE Boolean) dataStack) (processStack)
isBoolean-false ((top aType) dataStack) (processStack) (top isBoolean not) ((FALSE Boolean) dataStack) (processStack)

CCode : default

```
1  static void isBooleanAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isBoolean(top))
8      result = newBoolean(jInterp, TRUE);
9    else
10     result = newBoolean(jInterp, FALSE);
```

```
  pushCtxData(aCtx, result);
}
```

isTrue

CCode : default

```
static void isTrueAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isTrue(top)) {
    result = newBoolean(jInterp, TRUE);
  } else {
    result = newBoolean(jInterp, FALSE);
  }
  pushCtxData(aCtx, result);
}
```

isFalse

CCode : default

```
static void isFalseAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isTrue(top)) {
    result = newBoolean(jInterp, FALSE);
  } else {
    result = newBoolean(jInterp, TRUE);
  }
  pushCtxData(aCtx, result);
}
```

CHeader : private

```
extern Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  ContextObj *rootCtx = jInterp->rootCtx;
  assert(rootCtx);
  DictObj *dict = rootCtx->dict;
  assert(dict);

  DictNodeObj* true  = getSymbolEntry(dict, "true");
  true->value = newBoolean(jInterp, TRUE);

  DictNodeObj* false = getSymbolEntry(dict, "false");
  false->value = newBoolean(jInterp, FALSE);

  extendJoyLoLInRoot(jInterp, "isBoolean", "", isBooleanAP, "");
  extendJoyLoLInRoot(jInterp, "isTrue",    "", isTrueAP,    "");
  extendJoyLoLInRoot(jInterp, "isFalse",   "", isFalseAP,   "");

  return TRUE;
}
```

6.4.4 Lua functions

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_booleans_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
```

```
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_booleans [] = {
  {"gitVersion", lua_booleans_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_booleans (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerBooleans(jInterp);
  luaL_newlib(lstate, lua_booleans);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedBooleans` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_booleans(lstate);
  lua_setfield(lstate, -2, "joylol.booleans");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

6.4.5 Conclusions

CHeader : public

CHeader : private

```
1  extern size_t joylol_register_booleans(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <joylol/jInterps.h>
5  #include <joylol/stringBuffers.h>
6  #include <joylol/dictNodes.h>
7  #include <joylol/dictionaries.h>
8  #include <joylol/texts.h>
9  #include <joylol/cFunctions.h>
10 #include <joylol/assertions.h>
11 #include <joylol/contexts.h>
12 #include <joylol/booleans.h>
13 #include <joylol/booleans-private.h>
14 // dictionary
15 // printer
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
  requireLuaModule(lstate, "joylol.assertions");
  requireLuaModule(lstate, "joylol.pairs");
  requireLuaModule(lstate, "joylol.cFunctions");
  requireLuaModule(lstate, "joylol.stringBuffers");
  requireLuaModule(lstate, "joylol.texts");
  requireLuaModule(lstate, "joylol.dictionaries");
  requireLuaModule(lstate, "joylol.dictNodes");
  requireLuaModule(lstate, "joylol.contexts");
  requireStaticallyLinkedBooleans(lstate);
  getJoyLoLInterpInto(lstate, jInterp);
  initializeAllLoaded(lstate, jInterp);
  registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

7 Language CoAlgebras

1 Overview

In this chapter we look at the various sub languages which together define the `JoyLoL` Language.

114

114
114

115

8 JoyLoL Configuration Language CoAlgebras

1 Overview

Our 'JoyLoL Configuration Language', JoyLoLCF, is loosely based upon the 'Language for assembling classes in Eiffel' (Lace) ([Mey92] Appendix D).

Since a specific JoyLoL artefact is 'built' out of a large collection of inter-dependent individual JoyLoL CoAlgebras, the primary concern of the JoyLoLCF is to specify which particular implementations of these CoAlgebras, are to be used in a given JoyLoL artefact.

A system is composed from clusters of CoAlgebras.

2 CoAlgebras

JoylolCode : default

```
1  CoAlgebra System
2
3  creation
4    parseSystem ;
5
6  feature -- Abstract Syntax
7    systemName : Name;
8    defaults   : Defaults;
9    clusters   : Clusters;
10   externals  : Externals
11   generation : Generation;

12 feature -- Concrete Syntax
13   parseSystem(aCtx : Context) : System is
14     beginSExp
15       "system" P
16       ( parseName ) capture asName
17       ( parseDefaults ) capture asDefaults
18       ( parseClusters ) capture asClusters
19       ( parseExternals ) capture asExternals
20       ( parseGeneration ) capture asGeneration
21       "endSystem" P "--" P ( parseName ) capture isName
22     endSExp -- parseSystem ;

23 endCoAlgebra -- System
```

JoylolCode : default

115
115

JoylolCode : default

```
1   CoAlgebra Defaults
2
3   creation
4     parseDefaults ;
5
6   feature -- Abstract Syntax
7     optionTag  : OptionTag ;
8     optionMark : OptionMark ;
9     targetList : Targets ;

10  feature -- Concrete Syntax
11    parseDefaults(aCtx : Context) : Defaults is
12      beginSExp
13        "default" P
14        ( ( parseOptionClause ";" P ) + )
15      endSExp -- parseDefaults ;
16
17  endCoAlgebra -- Defaults
18
19  CoAlgebra OptionTag
20
21  creation
22    parseOptionTag ;

23  feature -- Abstract Syntax
24
25  feature -- Concrete Syntax
26    parseOptionTag(aCtx : Context ) : OptionTag is
27      beginSExp

28      endSExp -- parseOptionTag
29
30  endCoAlgebra -- OptionTag
31
32  CoAlgebra StandardValue
33
34  inherit
35    OptionTag ;
36
37  creation
38    parseStandardValue ;
```

```
39  feature -- Abstract Syntax
40
41  feature -- Concrete Syntax
42
43  endCoAlgebra -- StandardValue
44
45  CoAlgebra StandardValueYes
46
47  inherit
48    StandardValue ;
49
50  creation
51    parseStandardValueYes ;

52  feature -- Abstract Syntax
53
54  feature -- Concrete Syntax
55    parseStandardValueYes(aCtx : Context) : StandardValueYes is
56      beginSExp
57        "yes" P
58      endSExp -- parseStandardValueYes

59  endCoAlgebra -- StandardValueYes
60
61  CoAlgebra StandardValueNo
62
63  inherit
64    StandardValue ;
65
66  creation
67    parseStandardValueNo ;

68  feature -- Abstract Syntax
69
70  feature -- Concrete Syntax
71    parseStandardValueNo(aCtx : Context) : StandardValueNo is
72      beginSExp
73        "no" P
74      endSExp -- parseStandardValueNo

75  endCoAlgebra -- StandardValueNo
76
77  CoAlgebra StandardValueAll
```

```

inherit
  StandardValue ;

creation
  parseStandardValueAll ;

feature -- Abstract Syntax

feature -- Concrete Syntax
  parseStandardValueAll(aCtx : Context) : StandardValueAll is
    beginSExp
      "all" P
    endSExp -- parseStandardValueAll

endCoAlgebra -- StandardValueAll
```

JoylolCode : default

Lmsfile : default

Lmsfile : default

Lmsfile : default

```
require 'lms.contextDoc'
contextDoc.targets(lpTargets)
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

```
require 'lms.contextDoc'
contextDoc.targets(lpTargets)
recurseTargets{}
```

9 ConTeXt

119 119

120 120

9.1 JoyLoL CoAlgebraic Extensions ConTeXt module

9.1.1 Overview

The joyLoLCoAlg ConTeXt module is an extension of the Literate Programming ConTeXt module specifically tailored for use with `JoyLoL` code.

9.1.2 Code Manipulation

In this chapter we define the ConTeXt tools we will use to define the JoyLoL language.

The JoyLoL CoAlgebra ConTeXt module provides the tools required to fully describe the formal semantics of a particular JoyLoL CoAlgebraic extension including any defined JoyLoL words. It consists of literate documentation of the actual source code produced to implement the JoyLoL CoAlgebraic extension.

9.1.2.1 Implementation

In this section we load the Syntax Highlighter modules used by the code display commands (below). We also load the ConTests module used to test the JoyLoL CoAlgebra module itself. We then load the lua code associated with the `t-joylol` module.

MkIVCode : default

```
1 \writestatus{loading}{ConTeXt User Module / JoyLoL CoAlgebra Extensions}
2
3 \usemodule[t-literate-progs]
4 \usemodule[t-high-lisp]
5 \usemodule[t-contests]
6 \usemodule[t-joylol]
7
8 \ctxloadluafile{t-joylol-coalg}
```

9.1.2.1.1 Test Suite: JoyLoLCoAlg environment

The JoyLoLCoAlg environment provides a highly structured environment in which to describe the formal semantics and implementation of a particular JoyLoL CoAlgebraic extension.

A typical JoyLoLCoAlg environment consists of a collection of JoyLoL words. This includes a 'global' word which defines any global code required by the CoAlgegraic extension as a whole.

9.1.2.2 Examples

```
\startJoyLoLCoAlg[title=List of Lists][lists]
```

The first argument provides the arguments to an embedded `\startchapter` command.

The second argument provides the arguments to an embedded `\startcomponent` command. It also provides the base file name of all of the automatically generated code fragments.

The second argument also determines the name of any JoyLoL, ANSI-C, or Lua source file artefacts produced by this literate code documentation.

MkIVCode : default

```
\def\declareJoyLoLCoAlg[#1]{
  \directlua{thirddata.joylolCoAlgs.newCoAlg('#1')}
}

\let\startJoyLoLCoAlg=\declareJoyLoLCoAlg

\def\stopJoyLoLCoAlg{\relax}
```

LuaCode : default

```
--local function newCoAlg(coAlgName)
--  local lCoAlg        = setDefs(theCoAlg, coAlgName)
--  lCoAlg.name         = coAlgName
--  lCoAlg.words        = lCoAlg.words or {}
--  lCoAlg.words.global = {}
--end

local function newCoAlg(coAlgName)
  texio.write_nl('newCoAlg: ['..coAlgName..']')
  theCoAlg               = {}
  theCoAlg.name          = coAlgName
  theCoAlg.ctx           = nil --joylol.newContext()
  theCoAlg.hasJoyLoLCode = false;
  theCoAlg.hasLuaCode    = false;
  theCoAlg.hasCHeader    = false;
  theCoAlg.hasCCode      = false;
  build.coAlgsToBuild = build.coAlgsToBuild or {}
```

```
18    tInsert(build.coAlgsToBuild, coAlgName)
19    build.coAlgDependencies = build.coAlgDependencies or {}
20  end
21
22  coAlgs.newCoAlg = newCoAlg
```

9.1.2.2.1 Implementation: Start: Tests

Test case

should do something

```
\mockContextMacro{startcomponent}{1}
\mockContextMacro{startchapter}{1}
\startJoyLoLCoAlg[title=List of Lists][lists]
\assertMacroNthArgumentOnMthExpansionMatches%
  {startcomponent}{1}{1}{lists}{}
\assertMacroNthArgumentOnMthExpansionMatches%
  {startchapter}{1}{1}{title=List of Lists}{}
\startLuaConTest
  local theCoAlg = thirddata.joylolCoAlgs.theCoAlg
  assert.isTable(theCoAlg)
  assert.hasKey(theCoAlg, 'lists')
  local lists = theCoAlg.lists
  assert.isTable(lists)
  assert.hasKey(lists, 'name')
  assert.matches(lists.name, 'lists')
  assert.hasKey(lists, 'words')
  local words = lists.words
  assert.hasKey(words, 'global')
\stopLuaConTest
```

ConTest FAILED:
Expected [startcomponent] to have been expanded
in file: /home/stg/ExpositionGit/tools/conTeXt/joyLoL/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118 ConTest FAILED:
Expected [startchapter] to have been expanded
in file: /home/stg/ExpositionGit/tools/conTeXt/joyLoL/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118 LuaTest FAILED:
Could not execute the LuaTest.

Expected nil to be a table.
in file: /home/stg/ExpositionGit/tools/conTeXt/joyLoL/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118 ConTest FAILED:
LuaConTest failed
expected LuaConTest [local theCoAlg = thirddata.joylolCoAlgs.theCoAlgassert.isTable(theCoAlg)assert.hasKey(theCoAlg, 'lists')local lists = theCoAlg.listsassert.isTable(lists)assert.hasKey(lists, 'name')assert.matches(lists.name, 'lists')assert.hasKey(lists, 'words')local words = lists.wordsassert.hasKey(words, 'global')] to succeed
in file: /home/stg/ExpositionGit/tools/conTeXt/joyLoL/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118

9.1.2.3 Implementation: Stop

MkIVCode : default

```
\def\stopJoyLoLCoAlg{
  \directlua{thirddata.joylolCoAlgs.createCoAlg()}
  \stopchapter
  \stopcomponent
}
```

LuaCode : default

```
local function createCoAlg()
end

coAlgs.createCoAlg = createCoAlg
```

9.1.3 Source licenses

9.1.3.1 Examples

9.1.3.2 Implementation

MkIVCode : default

```
\unexpanded\def\srcCopyrightCCBYSA{}
```

9.1.4 Target licenses

9.1.4.1 Examples

9.1.4.2 Implementation

MkIVCode : default

```
\unexpanded\def\targetCopyrightMIT{}
```

9.1.5 Describing CoAlgebraic dependencies

9.1.5.1 Examples

9.1.5.2 Implementation

MkIVCode : default

```
\def\dependsOn[#1]{
  \directlua{thirddata.joylolCoAlgs.addDependency('#1')}
}
```

LuaCode : default

```
local function addDependency(dependencyName)
  build.coAlgDependencies = build.coAlgDependencies or {}
  tInsert(build.coAlgDependencies, dependencyName)
end

coAlgs.addDependency = addDependency
```

9.1.6 JoyLoL stack action: In

A JoyLoL stack action (either in or out) contains one or more sections of *implementation* code, either ANSI-C or Lua, together with a collection of descriptors of the JoyLoL {pre, post} {data, process} stacks. These stack actions provide the only allowed interface between an implementation language's 'local' variables and the JoyLoL stack context. 'In' actions take a data structure in a local variable and place it on either the data or process stacks.

9.1.6.1 Examples

9.1.6.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLStackActionIn[#1]{
  \directlua{thirddata.joylolCoAlgs.newStackActionIn('#1')}
}
```

LuaCode : default

```
local function newStackActionIn(aWord)
end

coAlgs.newStackActionIn = newStackActionIn
```

9.1.6.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLStackActionIn{
  \directlua{thirddata.joyloCoAlgs.endStackActionIn()}
}
```

LuaCode : default

```
local function endStackActionIn()
end

coAlgs.endStackActionIn = endStackActionIn
```

9.1.7 JoyLoL stack action: Out

A JoyLoL stack action (either in or out) contains one or more sections of *implementation* code, either ANSI-C or Lua, together with a collection of descriptors of the JoyLoL {pre, post} {data, process} stacks. These stack actions provide the only allowed interface between an implementation language's 'local' variables and the JoyLoL stack context. 'Out' actions take an item on either the data or process stack and place it into a data structure in a local variable and *possibly* 'removing' it from the appropriate stack.

9.1.7.1 Examples

9.1.7.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLStackActionOut[#1]{
  \directlua{thirddata.joylolCoAlgs.newStackActionOut('#1')}
}
```

LuaCode : default

```
local function newStackActionOut(aWord)
end

coAlgs.newStackActionOut = newStackActionOut
```

9.1.7.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLStackActionOut{
  \directlua{thirddata.joylolCoAlgs.endStackActionOut()}
}
```

LuaCode : default

```
local function endStackActionOut()
end

coAlgs.endStackActionOut = endStackActionOut
```

9.1.8 Describing the data stack

9.1.8.1 Examples

9.1.8.2 Implementation

MkIVCode : default

```
\def\preDataStack[#1][#2]{
  \directlua{thirddata.joylolCoAlgs.addPreDataStackDescription('#1', '#2')}
}

\def\postDataStack[#1]{
```

```
  \directlua{thirddata.joylolCoAlgs.addPostDataStackDescription('#1')}
}
```

LuaCode : default

```
local function addPreDataStackDescription(arg1, arg2)
end

coAlgs.addPreDataStackDescription = addPreDataStackDescription

local function addPostDataStackDescription(arg1, arg2)
end

coAlgs.addPostDataStackDescription = addPostDataStackDescription
```

9.1.9 Describing the process stack

9.1.9.1 Examples

9.1.9.2 Implementation

MkIVCode : default

```
\def\preProcessStack[#1][#2]{
  \directlua{thirddata.joylolCoAlgs.addPreProcessStackDescription('#1',
'#2')}
}

\def\postProcessStack[#1]{
  \directlua{thirddata.joylolCoAlgs.addPostProcessStackDescription('#1')}
}
```

LuaCode : default

```
local function addPreProcessStackDescription(arg1, arg2)
end

coAlgs.addPreProcessStackDescription = addPreProcessStackDescription

local function addPostProcessStackDescription(arg1, arg2)
end

coAlgs.addPostProcessStackDescription = addPostProcessStackDescription
```

9.2 JoyLoL

QUESTION: How do we load a *.joy file? Where do we put this command?

9.2.1 JoyLoL code environment

9.2.1.1 Examples

9.2.1.2 Implementation

We begin by registering the `JoylolCode` code type with the build srcTypes system. This will ensure the `\createJoylolCodeFile` macro (and corresponding lua code) knows how to deal with files of `JoylolCode`.

LuaCode : default

```
build.srcTypes = build.srcTypes or { }
build.srcTypes['JoylolCode'] = 'joylolCode'
```

MkIVCode : default

```
\defineLitProgs
  [JoylolCode]
  [ option=lisp, numbering=line,
    before={\noindent\startLitProgFrame}, after=\stopLitProgFrame
  ]

\setLitProgsOriginMarker[JoylolCode][markJoylolCodeOrigin]
```

LuaCode : default

```
local function markJoylolCodeOrigin()
  local codeType       = setDefs(code, 'JoylolCode')
  local codeStream     = setDefs(codeType, 'curCodeStream', 'default')
  codeStream           = setDefs(codeType, codeStream)
  return sFmt(';; from file: %s after line: %s',
    codeStream.fileName,
    toStr(
      mFloor(
        codeStream.startLine/code.lineModulus
      )*code.lineModulus
    )
  )
```

```
end

litProgs.markJoylolCodeOrigin = markJoylolCodeOrigin
```

9.2.2 Lua Make System files

In this section we add the code required to produce Lua Make System files which know how to compile JoyLoL CoAlgebraic extensions as shared libraries which can be loaded into a Lua implementation.

MkIVCode : default

```
\def\addJoyLoLTargets#1{
  \directlua{
    thirddata.joylolCoAlgs.addJoyLoLTargets('#1')
  }
}
```

LuaCode : default

```
local function addJoyLoLTargets(aCodeStream)
  litProgs.setCodeStream('Lmsfile', aCodeStream)
  litProgs.markCodeOrigin('Lmsfile')
  local lmsfile = {}
  tInsert(lmsfile, "require 'lms.joyLoL'\n")
  tInsert(lmsfile, "joylol.targets(lpTargets, {")
  tInsert(lmsfile, "  coAlgs = {")
  for i, aCoAlg in ipairs(build.coAlgsToBuild) do
    tInsert(lmsfile, "    '"..aCoAlg.."',")
  end
  tInsert(lmsfile, "  },")

  build.srcTargets = build.srcTargets or { }
  local srcTargets = build.srcTargets

  srcTargets.cHeader = srcTargets.cHeader or { }
  local cHeader      = srcTargets.cHeader
  tInsert(lmsfile, "  cHeaderFiles = {")
  for i, aSrcFile in ipairs(cHeader) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  srcTargets.cCode = srcTargets.cCode or { }
```

```
  local cCode      = srcTargets.cCode
  tInsert(lmsfile, "  cCodeFiles = {")
  for i, aSrcFile in ipairs(cCode) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  srcTargets.joylolCode = srcTargets.joylolCode or { }
  local joylolCode      = srcTargets.joylolCode
  tInsert(lmsfile, "  joylolCodeFiles = {")
  for i, aSrcFile in ipairs(joylolCode) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  if build.cCodeLibDirs then
    tInsert(lmsfile, "  cCodeLibDirs = {")
    for i, aLibDir in ipairs(build.cCodeLibDirs) do
      tInsert(lmsfile, "    '"..aLibDir.."',")
    end
    tInsert(lmsfile, "  },")
  end
  if build.cCodeLibs then
    tInsert(lmsfile, "  cCodeLibs = {")
    for i, aLib in ipairs(build.cCodeLibs) do
      tInsert(lmsfile, "    '"..aLib.."',")
    end
    tInsert(lmsfile, "  },")
  end

  tInsert(lmsfile, "  coAlgLibs = {")
  for i, aCoAlgDependency in ipairs(build.coAlgDependencies) do
    tInsert(lmsfile, "    '"..aCoAlgDependency.."',")
  end
  tInsert(lmsfile, "  },")
  tInsert(lmsfile, "})")
  litProgs.setPrepend('Lmsfile', aCodeStream, true)
  litProgs.addCode.default('Lmsfile', tConcat(lmsfile, '\n'))
end

coAlgs.addJoyLoLTargets = addJoyLoLTargets
```

MkIVCode : default

```
\def\addCTestJoyLoLCallbacks#1{
  \directlua{
    thirddata.joylolCoAlgs.addCTestJoyLoLCallbacks('#1')
  }
}
```

LuaCode : default

```
local function addCTestJoyLoLCallbacks(aCodeStream)
  local contests      = setDefs(thirddata, 'contests')
  local tests         = setDefs(contests, 'tests')
  local methods       = setDefs(tests, 'methods')
  local setup         = setDefs(methods, 'setup')
  local cTests        = setDefs(setup, 'cTests')
  aCodeStream         = aCodeStream         or 'default'
  cTests[aCodeStream] = cTests[aCodeStream] or { }
  tInsert(cTests[aCodeStream], [=[void ctestsWriteStdOut(
  JoyLoLInterp *jInterp,
  Symbol       *aMessage
) {
  fprintf(stdout, "%s", aMessage);
}

void ctestsWriteStdErr(
  JoyLoLInterp *jInterp,
  Symbol       *aMessage
) {
  fprintf(stderr, "%s", aMessage);
}
void *ctestsCallback(
  lua_State *lstate,
  size_t resourceId
) {
  if (resourceId == JoyLoLCallback_StdOutMethod) {
    return (void*)ctestsWriteStdOut;
  } else if (resourceId == JoyLoLCallback_StdErrMethod) {
    return (void*)ctestsWriteStdErr;
  } else if (resourceId == JoyLoLCallback_Verbose) {
    return (void*)FALSE;
  } else if (resourceId == JoyLoLCallback_Debug) {
    return (void*)FALSE;
  }
  return NULL;
}
```

```
]=])
  setup               = setDefs(tests, 'setup')
  cTests              = setDefs(setup, 'cTests')
  cTests[aCodeStream] = cTests[aCodeStream] or { }
  tInsert(cTests[aCodeStream], [=[setJoyLoLCallbackFrom(lstate, ctestsCallback);
]=])
end

coAlgs.addCTestJoyLoLCallbacks = addCTestJoyLoLCallbacks
```

MkIVCode : default

```
\def\setJoylolVerboseOn{
  \directlua{thirddata.joylol.setVerbose(true)}
}

\def\setJoylolVerboseOff{
  \directlua{thirddata.joylol.setVerbose(false)}
}

\def\setJoylolDebuggingOn{
  \directlua{thirddata.joylol.setDebugging(true)}
}

\def\setJoylolDebuggingOff{
  \directlua{thirddata.joylol.setVDebugging(false)}
}

\def\setJoylolTracingOn{
  \directlua{thirddata.joylol.setTracing(true)}
}

\def\setJoylolTracingOff{
  \directlua{thirddata.joylol.setTracing(false)}
}

\def\setJoylolShowStackOn{
  \directlua{thirddata.joylol.setShowStack(true)}
}

\def\setJoylolShowStackOff{
  \directlua{thirddata.joylol.setShowStack(false)}
}

```

```
\def\setJoylolShowSpecificationsOn{
  \directlua{thirddata.joylol.setShowSpecifications(true)}
}

\def\setJoylolShowSpecificationsOff{
  \directlua{thirddata.joylol.setShowSpecifications(false)}
}

\def\setJoylolCheckingOn{
  \directlua{thirddata.joylol.setChecking(true)}
}

\def\setJoylolCheckingOff{
  \directlua{thirddata.joylol.setChecking(false)}
}
```

LuaCode : default

```
function showStack(aMessage)
  texio.write_nl('-----------------------------------------------')
  if aMessage and type(aMessage) == 'string' and 0 < #aMessage then
    texio.write_nl(aMessage)
  end
  dataStack    = joylol.showData()
  processStack = joylol.showProcess()
  texio.write_nl("Data:")
  texio.write_nl(dataStack)
  texio.write_nl("Process:")
  texio.write_nl(processStack)
  texio.write_nl('AT: '..status.filename..'::'..status.linenumber)
  texio.write_nl('-----------------------------------------------')

end

contests.showStack = showStack
```

9.3 JoyLoL Tests

QUESTION: How do we load a *.joy file? Where do we put this command?

1 JoylolTests

see ConTests LuaTests.tex file

To integrate into ConTests inside ConTeXt runner we need to create something like:

local function runCurLuaTestCase(suite, case) runALuaTest(case.lua, suite, case) end

contests.testRunners.runCurLuaTestCase = runCurLuaTestCase

Anything in the testRunners table must be a function taking two arguments as above.

MkIVCode : default

```
\definetyping[JoylolTest]
\setuptyping[JoylolTest][option=lisp]

\let\oldStopJoylolTest=\stopJoylolTest
\def\stopJoylolTest{
  \oldStopJoylolTest%
  \directlua{thirddata.contests.addJoylolTest('_typing_')}
}

\def\showJoylolTest{
  \directlua{thirddata.contests.showJoylolTest()}
}

\def\setJoylolTestStage#1#2{
  \directlua{
    thirddata.contests.setJoylolTestStage('#1', '#2')
  }
}

\def\JoylolTestsMethodSetup{
  \setJoylolTestStage{Methods}{Setup}
}

\def\JoylolTestsMethodTeardown{
  \setJoylolTestStage{Methods}{Teardown}
}
```

```
27
28  \def\JoylolTestsSetup{
29    \setJoylolTestStage{Global}{Setup}
30  }
31
32  \def\JoylolTestsTeardown{
33    \setJoylolTestStage{Global}{Teardown}
34  }
35
36  \def\JoylolTestSuiteSetup{
37    \setJoylolTestStage{TestSuite}{Setup}
38  }
39
40  \def\JoylolTestSuiteTeardown{
41    \setJoylolTestStage{TestSuite}{Teardown}
42  }
43
44  \def\setJoylolTestStream#1{
45    \directlua{
46      thirddata.contests.setJoylolTestStream('#1')
47    }
48  }
49
50  \def\addJoylolTestInclude#1{
51    \directlua{
52      thirddata.contests.addJoylolTestInclude('#1')
53    }
54  }
55
56  \def\addJoylolTestLibDir#1{
57    \directlua{
58      thirddata.contests.addJoylolTestLibDir('#1')
59    }
60  }
61
62  \def\addJoylolTestLib#1{
63    \directlua{
64      thirddata.contests.addJoylolTestLib('#1')
65    }
66  }
67
68  \def\createJoylolTestFile#1#2#3{
69    \directlua{
```

```
    thirddata.contests.createJoylolTestFile('#1', '#2', '#3')
  }
}

\def\addJoylolTestTargets#1{
  \directlua{
    thirddata.contests.addJoylolTestTargets('#1')
  }
}
```

LuaCode : default

```
local function addJoylolTest(bufferName)
  local bufferContents = buffers.getcontent(bufferName):gsub("\13", "\n")
  local methods        = setDefs(tests, 'methods')
  local suite          = setDefs(tests, 'curSuite')
  local case           = setDefs(suite, 'curCase')
  local joylolTests    = setDefs(case,  'joylolTests')
  local curStage       = tests.stage:lower()
  if curStage:find('global') then
    if curStage:find('up') then
      local setup      = setDefs(tests,    'setup')
      joylolTests      = setDefs(setup,    'joylolTests')
    elseif curStage:find('down') then
      local teardown   = setDefs(tests,    'teardown')
      joylolTests      = setDefs(teardown, 'joylolTests')
    end
  elseif curStage:find('suite') then
    if curStage:find('up') then
      local setup      = setDefs(suite,    'setup')
      joylolTests      = setDefs(setup,    'joylolTests')
    elseif curStage:find('down') then
      local teardown   = setDefs(suite,    'teardown')
      joylolTests      = setDefs(teardown, 'joylolTests')
    end
  elseif curStage:find('method') then
    if curStage:find('up') then
      local setup      = setDefs(methods,  'setup')
      joylolTests      = setDefs(setup,    'joylolTests')
    elseif curStage:find('down') then
      local teardown   = setDefs(methods,  'teardown')
      joylolTests      = setDefs(teardown, 'joylolTests')
    end
  end
```

```
33    tests.stage            = ''
34    local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
35    joylolTestStream       = setDefs(joylolTests, joylolTestStream)
36    tInsert(joylolTestStream, bufferContents)
37  end
38
39  contests.addJoylolTest = addJoylolTest
40
41  local function setJoylolTestStage(suiteCase, setupTeardown)
42    tests.stage = suiteCase..'-'..setupTeardown
43  end
44
45  contests.setJoylolTestStage = setJoylolTestStage
46
47  local function setJoylolTestStream(aCodeStream)
48    if type(aCodeStream) ~= 'string'
49      or #aCodeStream < 1 then
50      aCodeStream = 'default'
51    end
52    tests.curJoylolTestStream = aCodeStream
53  end
54
55  contests.setJoylolTestStream = setJoylolTestStream
56
57  local function addJoylolTestInclude(anInclude)
58    local joylolIncludes   = setDefs(tests, 'joylolIncludes')
59    local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
60    joylolTestStream       = setDefs(joylolIncludes, joylolTestStream)
61    tInsert(joylolTestStream, anInclude)
62  end
63
64  contests.addJoylolTestInclude = addJoylolTestInclude
65
66  local function addJoylolTestLibDir(aLibDir)
67    local joylolLibDirs    = setDefs(tests, 'joylolLibDirs')
68    local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
69    joylolTestStream       = setDefs(joylolLibDirs, joylolTestStream)
70    tInsert(joylolTestStream, aLibDir)
71  end
72
73  contests.addJoylolTestLibDir = addJoylolTestLibDir
74
75  local function addJoylolTestLib(aLib)
```

```
  local joylolLibs       = setDefs(tests, 'joylolLibs')
  local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
  joylolTestStream       = setDefs(joylolLibs, joylolTestStream)
  tInsert(joylolTestStream, aLib)
end

contests.addJoylolTestLib = addJoylolTestLib
```

LuaCode : default

```
local function buildJoylolChunk(joylolChunk, curSuite, curCase)
  if type(joylolChunk) == 'table' then
    joylolChunk = tConcat(joylolChunk, '\n')
  end

  if type(joylolChunk) ~= 'string' then
    return nil
  end

  if joylolChunk:match('^%s*$') then
    return nil
  end

  return [=[(
]=]..joylolChunk..[=[
)
(
 "]=]..curCase.desc..[=["
  ]=]..curCase.fileName..[=[
  ]=]..curCase.startLine..[=[
  ]=]..status.linenumber..[=[
)
runTestCase
showStack
true
]=]
end

contests.buildJoylolChunk = buildJoylolChunk

local function showJoylolTest()
  local curSuite = setDefs(tests, 'curSuite')
  local curCase  = setDefs(curSuite, 'curCase')
  texio.write_nl('================================================')
```

```
  local joylolChunk =
    buildJoylolChunk(curCase.joylol, curSuite, curCase)
  if joylolChunk then
    texio.write_nl('Joylol Test: ')
    texio.write_nl('-----------------------------------------------')
    texio.write_nl(joylolChunk)
    texio.write_nl('-----------------------------------------------')
  else
    texio.write_nl('NO Joylol Test could be built')
  end
  texio.write_nl('AT: '..status.filename..'::'..status.linenumber)
  texio.write_nl('===============================================')
end

contests.showJoylolTest = showJoylolTest
```

LuaCode : default

```
local function runAJoylolTest(joylolTest, suite, case)
  case.passed = case.passed or true
  local joylolChunk = buildJoylolChunk(joylolTest, suite, case)
  if not joylolChunk then
    -- nothing to test
    return true
  end

  local caseStats = tests.stats.joylol.cases
  caseStats.attempted = caseStats.attempted + 1
  tex.print("\\starttyping")
  joylol.evalString(joylolChunk)
  tex.print("\\stoptyping")
  local testResult = joylol.popData()
  if not testResult then
    local errObj = joylol.popData()
    local failure = logFailure(
      "LuaTest FAILED",
      suite.desc,
      case.desc,
      errObj.message,
      toStr(errObj[1]),
      sFmt("in file: %s between lines %s and %s",
        case.fileName, toStr(case.startLine), toStr(case.lastLine))
      )
    reportFailure(failure, false)
```

```
    tInsert(tests.failures, failure)
    return false
  end

  -- all tests passed
  caseStats.passed = caseStats.passed + 1
  tex.print("\\noindent{\\green PASSED}")
  return true
end

contests.runAJoylolTest = runAJoylolTest

local function runCurJoylolTestCase(suite, case)
  runAJoylolTest(case.joylol, suite, case)
end

contests.testRunners.runCurJoylolTestCase = runCurJoylolTestCase
```

MkIVCode : default

```
\def\createJoylolTestFile#1#2#3{
  \directlua{
    thirddata.contests.createJoylolTestFile('#1', '#2', '#3')
  }
}
```

LuaCode : default

```
local function createJoylolTestFile(
  aCodeStream, aFilePath, aFileHeader
)
  texio.write("\n-------------------------------\n")
  texio.write("aCodeStream = ".. aCodeStream.."\n")
  texio.write("aFilePath   = ".. aFilePath.."\n")
  texio.write("\n-------------------------------\n")

  if not build.buildDir then
    texio.write('\nERROR: document directory NOT yet defined\n')
    texio.write('       NOT creating code file ['..aFilePath..']\n\n')
    return
  end

  if type(aFilePath) ~= 'string'
    or #aFilePath < 1 then
```

```
17    texio.write('\nERROR: no file name provided for joylolTests\n\n')
18    return
19  end
20
21  build.joylolTestTargets = build.joylolTestTargets or { }
22  local aTestExec = aFilePath:gsub('%..+$','')
23  tInsert(build.joylolTestTargets, aTestExec)
24
25  aFilePath = build.buildDir .. '/buildDir/' .. aFilePath
26  local outFile = io.open(aFilePath, 'w')
27  if not outFile then
28    return
29  end
30  texio.write('creating JoylolTest file: ['..aFilePath..']\n')
31  if type(aFileHeader) == 'string'
32    and 0 < #aFileHeader then
33    outFile:write(aFileHeader)
34    outFile:write('\n\n')
35  end
36
37  tests.suites = tests.suites or { }
38
39  if type(aCodeStream) ~= 'string'
40    or #aCodeStream < 1 then
41    aCodeStream = 'default'
42  end
43
44  outFile:write(';; A JoylolTest file\n\n')
45  outFile:write(';;-------------------------------------------------------\n')
46  outFile:write(';; global setup\n')
47  outFile:write(';;-------------------------------------------------------\n\n')
48  local joylolIncludes = setDefs(tests, 'joylolIncludes')
49
50  joylolIncludes[aCodeStream] = joylolIncludes[aCodeStream] or { }
51
52  for i, anInclude in ipairs(joylolIncludes[aCodeStream]) do
53    outFile:write(anInclude..'\n')
54    outFile:write('load \n\n')
55  end
56  outFile:write('\n')
```

```
57
58    tests.methods = tests.methods or { }
59    local methods = tests.methods
60    methods.setup = methods.setup or { }
61    local mSetup  = methods.setup
62    mSetup.joylolTests = mSetup.joylolTests or { }
63    msJoylolTests      = mSetup.joylolTests
64
65    --msJoylolTests[aCodeStream] = msJoylolTests[aCodeStream] or { }

66    if msJoylolTests and
67      msJoylolTests[aCodeStream] then
68      local setupCode = tConcat(msJoylolTests[aCodeStream],'\n')
69      setupCode       = litProgs.splitString(setupCode)
70      outFile:write(tConcat(setupCode, '\n'))
71      outFile:write('\n')
72    end
73    outFile:write('\n')
74
75    outFile:write(';;-------------------------------------------------------\n')
76    outFile:write(';; all tests\n')
77    outFile:write(';;-------------------------------------------------------\n')
78
79    outFile:write('(\n')
80    outFile:write('  (\n')
81    tests.setup = tests.setup or { }
82    if tests.setup.joylolTests and
83      tests.setup.joylolTests[aCodeStream] then
84      local setupCode = tConcat(tests.setup.joylolTests[aCodeStream],'\n')
85      setupCode       = litProgs.splitString(setupCode)
86      outFile:write('  '..tConcat(setupCode, '\n  '))
87      outFile:write('\n')
88    end
89    outFile:write('  ) ;; JoylolTests setup\n')
90    outFile:write('  tests.defineTestsSetup\n\n')
91
92    outFile:write('  (\n')
93    tests.teardown = tests.teardown or { }
94    if tests.teardown.joylolTests and
95      tests.teardown.joylolTests[aCodeStream] then
96      local teardownCode =tConcat(tests.teardown.joylolTests[aCodeStream],'\n
97  ')
98      teardownCode = litProgs.splitString(teardownCode, '\n')
```

```
99      outFile:write('  '..tConcat(teardownCode, '\n  '))
100   end
101   outFile:write('  ) ;; JoylolTests teardown\n')
102   outFile:write('  tests.defineTestsTeardown\n\n')
103
104   for i, aTestSuite in ipairs(tests.suites) do
105     aTestSuite.cases = aTestSuite.cases or { }
106     local suiteCaseBuf = { }
107
108     for j, aTestCase in ipairs(aTestSuite.cases) do
109       local joylolTests     = setDefs(aTestCase, 'joylolTests')
110       if aTestCase.desc and
111         aTestCase.fileName and
112         aTestCase.startLine and
113         aTestCase.lastLine and
114         joylolTests[aCodeStream] then
115         tInsert(suiteCaseBuf, '    ;;-------------------------------------------------------
116         tInsert(suiteCaseBuf, '    ;; jTC: '..aTestCase.desc..'\n')
117         tInsert(suiteCaseBuf, '    ;;-------------------------------------------------------
118         tInsert(suiteCaseBuf, '    (\n')
119         tInsert(suiteCaseBuf, '      (\n')
120         tInsert(suiteCaseBuf, '        "'..aTestCase.desc..'"\n')
121         tInsert(suiteCaseBuf, '        "'..aTestCase.fileName..'"\n')
122         tInsert(suiteCaseBuf, '        '..toStr(aTestCase.startLine)..'\n')
123         tInsert(suiteCaseBuf, '        '..toStr(aTestCase.lastLine)..'\n')
124         tInsert(suiteCaseBuf, '      ) ;; test case details\n')
125         tInsert(suiteCaseBuf, '      tests.recordTestCaseDetails\n\n')
126         local joylolTestsCode = tConcat(joylolTests[aCodeStream], '\n')
127         joylolTestsCode       = litProgs.splitString(joylolTestsCode)
128         tInsert(suiteCaseBuf, '    '..tConcat(joylolTestsCode, '\n    '))
129         tInsert(suiteCaseBuf, '\n    ) ;; test case\n')
130         tInsert(suiteCaseBuf, '    tests.runTestCase\n\n')
131       elseif (not aTestCase.desc or
132         not aTestCase.fileName or
133         not aTestCase.startLine or
134         not aTestCase.lastLine) and
135         joylolTests[aCodeStream] then
136         texio.write("\nERROR missing \\startTestCase\n")
137         texio.write("near:\n")
138         texio.write(tConcat(joylolTests[aCodeStream], '\n'))
139         texio.write('\n')
140       end
141     end
```

```
142
143     if aTestSuite.desc and (0 < #suiteCaseBuf) then
144       outFile:write('  ;;-------------------------------------------------------\n')
145       outFile:write('  ;; jTS:'..aTestSuite.desc..'\n')
146       outFile:write('  ;;-------------------------------------------------------\n')
147       outFile:write('  (\n')
148       outFile:write('    (\n')
149       outFile:write('      "'..aTestSuite.desc..'"\n')
150       outFile:write('    ) ;; test suite details\n')
151       outFile:write('    tests.recordTestSuiteDetails\n\n')
152
153       outFile:write('    (\n')
154       aTestSuite.setup = aTestSuite.setup or { }
155       if aTestSuite.setup.joylolTests and
156         aTestSuite.setup.joylolTests[aCodeStream] then
157         local setupCode = tConcat(aTestSuite.setup.joylolTests[aCodeStream],'\n
158 ')
159         setupCode = litProgs.splitString(setupCode, '\n')
160         outFile:write('    '..tConcat(setupCode, '\n    '))
161       end
162       outFile:write('    ) ;; test suite setup\n')
163       outFile:write('    tests.defineTestSuiteSetup\n\n')
164
165       outFile:write('    (\n')
166       aTestSuite.teardown = aTestSuite.teardown or { }
167       if aTestSuite.teardown.joylolTests and
168         aTestSuite.teardown.joylolTests[aCodeStream] then
169         local teardownCode = tConcat(aTestSuite.teardown.joylolTests[aCodeStream],'\n
170 ')
171         teardownCode = litProgs.splitString(teardownCode, '\n')
172         outFile:write('    '..tConcat(teardownCode, '\n    '))
173       end
174       outFile:write('    ) ;; test suite teardown\n')
175       outFile:write('    tests.defineTestSuiteTeardown\n\n')
176
177       outFile:write(tConcat(suiteCaseBuf))
178
179       outFile:write('  )\n')
180       outFile:write('  tests.runTestSuite\n\n')

181     elseif not aTestSuite.desc and (0 < #suiteCaseBuf) then
182       texio.write("\nERROR missing \\startTestSuite\n")
183       texio.write("near:\n")
```

```
184       texio.write(tConcat(suiteCaseBuf, '\n'))
185       texio.write('\n')
186     end
187   end

188   outFile:write(')\n')
189   outFile:write('tests.runAllTests\n\n')
190
191   outFile:write(';;-------------------------------------------------------\n')
192   outFile:write(';; global teardown\n')
193   outFile:write(';;-------------------------------------------------------\n\n')

194   methods.teardown      = methods.teardown or { }
195   local mTeardown       = methods.teardown
196   mTeardown.joylolTests = mTeardown.joylolTests or { }
197   mtJoylolTests         = mTeardown.joylolTests
198
199   --mtJoylolTests[aCodeStream] = mtJoylolTests[aCodeStream] or { }
200
201   if mtJoylolTests and
202     mtJoylolTests[aCodeStream] then
203     local teardownCode = tConcat(mtJoylolTests[aCodeStream],'\n')
204     teardownCode       = litProgs.splitString(teardownCode)
205     outFile:write('  '..tConcat(teardownCode, '\n  '))
206     outFile:write('\n')
207   end
208   outFile:write('\n')
209   outFile:write(';;-------------------------------------------------------\n')
210
211   outFile:close()
212 end
213
214 contests.createJoylolTestFile = createJoylolTestFile
```

9.3.1 Lua Make System files

In this section we add the code required to produce Lua Make System files which know how to compile JoyLoL Tests.

MkIVCode : default

```
1 \def\addJoyLoLTestTargets#1{
2   \directlua{
```

```
    thirddata.joylolCoAlgs.addJoylolTestTargets('#1')
  }
}
```

LuaCode : default

```
local function addJoylolTestTargets(aCodeStream)
  litProgs.setCodeStream('Lmsfile', aCodeStream)
  litProgs.markCodeOrigin('Lmsfile')
  local lmsfile = {}
  tInsert(lmsfile, "require 'lms.joylolTests'\n")
  tInsert(lmsfile, "joylolTests.targets(lpTargets, {")
  tInsert(lmsfile, "  testExecs = {")
  for i, aTestExec in ipairs(build.joylolTestTargets) do
    tInsert(lmsfile, "    '"..aTestExec.."',")
  end
  tInsert(lmsfile, "  },")

  build.srcTargets = build.srcTargets or { }
  local srcTargets = build.srcTargets

  srcTargets.cHeader = srcTargets.cHeader or { }
  local cHeader      = srcTargets.cHeader
  tInsert(lmsfile, "  cHeaderFiles = {")
  for i, aSrcFile in ipairs(cHeader) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  srcTargets.cCode = srcTargets.cCode or { }
  local cCode      = srcTargets.cCode
  tInsert(lmsfile, "  cCodeFiles = {")
  for i, aSrcFile in ipairs(cCode) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  if build.cCodeLibDirs then
    tInsert(lmsfile, "  cCodeLibDirs = {")
    for i, aLibDir in ipairs(build.cCodeLibDirs) do
      tInsert(lmsfile, "    '"..aLibDir.."',")
    end
    tInsert(lmsfile, "  },")
  end
```

```
36    if build.cCodeLibs then
37      tInsert(lmsfile, "  cCodeLibs = {")
38      for i, aLib in ipairs(build.cCodeLibs) do
39        tInsert(lmsfile, "    '"..aLib.."',")
40      end
41      tInsert(lmsfile, "  },")
42    end
43
44    tInsert(lmsfile, "  coAlgLibs = {")
45    for i, aCoAlgDependency in ipairs(build.coAlgDependencies) do
46      tInsert(lmsfile, "    '"..aCoAlgDependency.."',")
47    end
48    tInsert(lmsfile, "  },")
49    tInsert(lmsfile, "})")
50    litProgs.setPrepend('Lmsfile', aCodeStream, true)
51    litProgs.addCode.default('Lmsfile', tConcat(lmsfile, '\n'))
52  end
53
54  coAlgs.addJoylolTestTargets = addJoylolTestTargets
```

9.4 Rules

1 Test Suite: rule environment

MkIVCode : default

```
\let\stopRule\relax

\def\stopRuleDone{
  \directlua{thirddata.joylolCoAlgs.stopRule()}
}

\def\startRule[#1]{
  \directlua{thirddata.joylolCoAlgs.startRule('#1')}
  \buff_pickup{_rules_buffer_}%
    {startRule}{stopRule}%
    {\relax}{\stopRuleDone}\plusone%
}
```

LuaCode : default

```
local function startRule(ruleName)
  texio.write_nl("starting rule: ["..ruleName.."]")
end

coAlgs.startRule = startRule

local sectionHeaders = tConcat({
  'arguments',
  'returns',
  'preDataStack',
  'preProcessStack',
  'preConditions',
  'postDataStack',
  'postProcessStack',
  'postConditions'
}, '|'):lower()

local function stopRule()
  local rulesBody  = buffers.getcontent('_rules_buffer_'):gsub("\13",
"\n")
  local rules      = { }
  local lines      = { }
  local curSection = 'ignore'
```

```
  for aLine in rulesBody:gmatch("[^\r\n]+") do
    local aMatch = aLine:match("^%s*\\(%a+)%s*$")
    if aMatch and
      sectionHeaders:find(aMatch:lower(), 1, true)
    then
      rules[curSection] = lines
      lines             = { }
      curSection        = aMatch
    else
      tInsert(lines, aLine)
    end
  end
  rules[curSection] = lines

  texio.write_nl('---------rules-buffer-------------')
  texio.write_nl(lpPP(rules))
  texio.write_nl('---------rules-buffer-------------')
end

coAlgs.stopRule = stopRule
```

Test case

should manipulate buffers

```
\startRule[testRule]
  \arguments
    some argument content
  \returns
    some returns content
  \preDataStack
    some preDataStack content
  \preProcessStack
    some preProcessStack content
  \preConditions
    some preConditions content
  \postDataStack
    some postDataStack content
  \postProcessStack
    some postProcessStack content
  \postConditions
    some postConditions content
\stopRule
```

SKIPPED

152

152
152

9.5 JoyLoL code fragments

9.5.1 JoyLoL implementation fragment

9.5.1.1 Examples

9.5.1.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLFragment[#1]{
  \directlua{thirddata.joylolCoAlgs.newFragment('#1')}
}
```

LuaCode : default

```
local function newFragment(fragmentName)
  local curFragment = setDefs(theCoAlg, 'curFragment')
  curFragment.name  = fragmentName
  setDefs(curFragment, 'code')
end

coAlgs.newFragment = newFragment
```

9.5.1.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLFragment{
  \directlua{thirddata.joylolCoAlgs.endFragment()}
}
```

LuaCode : default

```
local function endFragment()
  local curFragment =
    shouldExist(theCoAlg, 'curFragment', {
      '\\stopJoyLoLFragment used outside of ',
      '\\startJoyLoLFragment environment'
    })

  texio.write_nl('---------joylol-fragment-------------')
```

```
  texio.write_nl(lpPP(curFragment))
  texio.write_nl('---------joylol-fragment-------------')

  local wordName =
    shouldExist(curFragment, 'name',
      'joylol fragment not named'
    )
  local codeVersions =
    shouldExist(curFragment, 'code',
      'incorrectly setup joylol fragment'
    )

  local numCodeVersions = 0
  for fragmentType, fragmentBody in pairs(codeVersions) do
--    joylol.crossCompilers.addFragment(
--      fragmentType,
--      wordName,
--      fragmentBody
--    )
    numCodeVersions = numCodeVersions + 1
  end
  if numCodeVersions < 1 then
    error(tConcat({
      'no \\startFragment environment used ',
      'inside a \\startJoyLoLFragment environment'
    }))
  end
end

coAlgs.endFragment = endFragment
```

9.5.2 fragment definition environment

MkIVCode : default

```
\let\stopFragment\relax

\def\stopFragmentDone{
  \directlua{thirddata.joylolCoAlgs.stopFragment()}
}

\def\startFragment[#1]{
  \directlua{thirddata.joylolCoAlgs.startFragment('#1')}
```

```
  \buff_pickup{_fragment_buffer_}%
    {startFragment}{stopFragment}%
    {\relax}{\stopFragmentDone}\plusone%
}
```

LuaCode : default

```
local function startFragment(fragmentType)
  local curFragment =
    shouldExist(theCoAlg, 'curFragment', {
      '\\startFragment used outside of ',
      '\\startJoyLoLFragment environment'
    })
  curFragment.curType  = fragmentType
end

coAlgs.startFragment = startFragment

local function stopFragment()
  local curFragment  =
    shouldExist(theCoAlg, 'curFragment', {
      '\\stopFragment used outside of ',
      '\\startJoyLoLFragment environment'
    })
  local codeVersions =
    shouldExist(curFragment, 'code',
      'incorrectly setup joylol fragment - missing code'
    )
  local curType =
    shouldExist(curFragment, 'curType',
      'incorrectly setup fragment - missing curType'
    )
  local fragmentBody =
    buffers.getcontent('_fragment_buffer_'):gsub("\13", "\n")
  codeVersions[curType] = fragmentBody

  tex.sprint("\\starttyping")
  tex.print(fragmentBody)
  tex.sprint("\\stoptyping")
end

coAlgs.stopFragment = stopFragment
```

9.6 JoyLoL words

9.6.1 JoyLoL word environment

A JoyLoL word contains one or more sections of code, either JoyLoL, ANSI-C or Lua, together with a collection of descriptors of the JoyLoL {pre, post} {data, process} stacks.

9.6.1.1 Examples

9.6.1.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLWord[#1]{
  \directlua{thirddata.joylolCoAlgs.newWord('#1')}
}
```

LuaCode : default

```
local function newWord(wordName)
  local curWord = setDefs(theCoAlg, 'curWord')
  curWord.name = wordName
  setDefs(curWord, 'code')
end

coAlgs.newWord = newWord
```

9.6.1.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLWord{
  \directlua{thirddata.joylolCoAlgs.endWord()}
}
```

LuaCode : default

```
local function endWord()
  local curWord =
    shouldExist(theCoAlg, 'curWord', {
      '\\stopJoyLoLWord used outside of ',
      '\\startJoyLoLWord environment'
```

```
    })

  texio.write_nl('---------joylol-word-------------')
  texio.write_nl(lpPP(curWord))
  texio.write_nl('---------joylol-word-------------')

  local wordName =
    shouldExist(curWord, 'name',
      'joylol word not named'
    )
  local codeVersions =
    shouldExist(curWord, 'code',
      'incorrectly setup joylol word'
    )

  local numCodeVersions = 0
  for implType, implBody in pairs(codeVersions) do
--    joylol.crossCompilers.addImplementation(
--      implType,
--      wordName,
--      implBody
--    )
    numCodeVersions = numCodeVersions + 1
  end
  if numCodeVersions < 1 then
    error(tConcat({
      'no \\startImplementation environment used ',
      'inside a \\startJoyLoLWord environment'
    }))
  end
end

coAlgs.endWord = endWord
```

9.6.2 JoyLoL word implementation

MkIVCode : default

```
\let\stopImplementation\relax

\def\stopImplementationDone{
  \directlua{thirddata.joylolCoAlgs.stopImplementation()}
}
```

```

\def\startImplementation[#1]{
  \directlua{thirddata.joylolCoAlgs.startImplementation('#1')}
  \buff_pickup{_implementation_buffer_}%
    {startImplementation}{stopImplementation}%
    {\relax}{\stopImplementationDone}\plusone%
}
```

LuaCode : default

```
local function startImplementation(implType)
  local curWord =
    shouldExist(theCoAlg, 'curWord', {
      '\\startImplementation used outside of ',
      '\\startJoyLoLWord environment'
    })
  curWord.curType = implType
end

coAlgs.startImplementation = startImplementation

local function stopImplementation()
  local curWord =
    shouldExist(theCoAlg, 'curWord', {
      '\\stopImplementation used outside of ',
      '\\startJoyLoLWord environment'
    })
  local codeVersions =
    shouldExist(curWord, 'code',
      'incorrectly setup joylol word - missing code'
    )
  local curType =
    shouldExist(curWord, 'curType',
      'incorrectly setup joylol word - missing curType'
    )
  local implBody =
    buffers.getcontent('_implementation_buffer_'):gsub("\13", "\n")
  codeVersions[curType] = implBody

  tex.sprint("\\starttyping")
  tex.print(implBody)
  tex.sprint("\\stoptyping")
end

```

JoyLoL word implementation 9.6.2

```
35 coAlgs.stopImplementation = stopImplementation
```

9.7 Preamble

MkIVCode : default

```
%D \module
%D   [     file=t-joylol-coalg,
%D      version=2017.05.10,
%D        title=\CONTEXT\ User module,
%D     subtitle=The JoyLoL CoAlgebraic Extensions \ConTeXt\ module,
%D       author=Stephen Gaito,
%D         date=\currentdate,
%D    copyright=PerceptiSys Ltd (Stephen Gaito),
%D        email=stephen@perceptisys.co.uk,
%D      license=MIT License]

%C Copyright (C) 2017 PerceptiSys Ltd (Stephen Gaito)
%C
%C Permission is hereby granted, free of charge, to any person obtaining a
%C copy of this software and associated documentation files (the
%C "Software"), to deal in the Software without restriction, including
%C without limitation the rights to use, copy, modify, merge, publish,
%C distribute, sublicense, and/or sell copies of the Software, and to
%C permit persons to whom the Software is furnished to do so, subject to
%C the following conditions:
%C
%C The above copyright notice and this permission notice shall be included
%C in all copies or substantial portions of the Software.
%C
%C THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
%C OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
%C MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
%C IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
%C CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
%C TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
%C SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

% begin info
%
% title   : JoyLoL CoAlgebra definitions
% comment : Provides structured document and code generation
% status  : under development, mkiv only
%
% end info
```

```

\unprotect
```

MkIVCode : default

```
\protect \endinput
```

LuaCode : default

```
-- This is the lua code associated with t-joylol-coalg.mkiv

if not modules then modules = { } end modules ['t-joylol-coalg'] = {
    version   = 1.000,
    comment   = "joylol coalgegraic extensions - lua",
    author    = "PerceptiSys Ltd (Stephen Gaito)",
    copyright = "PerceptiSys Ltd (Stephen Gaito)",
    license   = "MIT License"
}

thirddata         = thirddata        or {}
thirddata.joylol  = thirddata.joylol or {}

local joylol      = thirddata.joylol

thirddata.joylolCoAlgs = thirddata.joylolCoAlgs or {}
local coAlgs      = thirddata.joylolCoAlgs
coAlgs.theCoAlg   = {}
local theCoAlg    = coAlgs.theCoAlg

thirddata.literateProgs = thirddata.literateProgs or {}
local litProgs    = thirddata.literateProgs
litProgs.code     = litProgs.code or {}
local code        = litProgs.code
local setDefs     = litProgs.setDefs
local shouldExist = litProgs.shouldExist
local build       = setDefs(litProgs, 'build')

local contests    = setDefs(thirddata, 'contests')
local initStats   = contests.initStats
local tests       = setDefs(contests, 'tests')
                    setDefs(tests, 'suites')
                    setDefs(tests, 'failures')
local assert      = setDefs(contests, 'assert')
                    setDefs(contests, 'testRunners')
```

```
local expInfo     = setDefs(contests, 'expInfo')

                        setDefs(tests, 'stats')
tests.stats.joylol     = initStats()
local joylolStats      = tests.stats.joylol
local joylolAssertions = joylolStats.assertions

local tInsert = table.insert
local tConcat = table.concat
local tRemove = table.remove
local tSort   = table.sort
local sFmt    = string.format
local sMatch  = string.match
local toStr   = tostring
local mFloor  = math.floor
local lpPP    = litProgs.prettyPrint

--local pushData, pushProcess = joylol.pushData, joylol.pushProcess
--local pushProcessQuoted = joylol.pushProcessQuoted
--local popData, popProcess   = joylol.popData, joylol.popProcess
--local newList, newDictionary = joylol.newList, joylol.newDictionary
--local jEval = joylol.eval

if joylol.core then
  interfaces.writestatus(
    "joyLoL",
    joylol.core.context.gitVersion('commitDate')
  )
else
  interfaces.writestatus(
    "joyLoL",
    "partially loaded"
  )
end

interfaces.writestatus('joyLoLCoAlg', "loaded JoyLoL CoAlgs")
```

LuaTemplate : default

```
if not modules then modules = { } end modules ['t-joylol-coalg-templates']
= {
    version   = 1.000,
    comment   = "JoyLoL CoAlgebraic extensions module - templates",
    author    = "PerceptiSys Ltd (Stephen Gaito)",
```

```
6       copyright = "PerceptiSys Ltd (Stephen Gaito)",
7       license   = "MIT License"
8   }
9
10  thirddata              = thirddata              or {}
11  thirddata.joylolCoAlgs = thirddata.joylolCoAlgs or {}
12
13  local coAlgs     = thirddata.joylolCoAlgs
14
15  local templates  = { }
16
17  templates.cHeader = [=[This is the start of a cHeader template
18  {{ lookupInDict 'coAlgName }}
19  This is the end of a cHeader template
20  ]=]
21
22  templates.cCode = [=[This is the start of a cCode template
23  {{ lookupInDict 'coAlgName }}
24  This is the end of the cCode template
25  ]=]
26
27  templates.joyLoLCode = [=[This is the start of a joyLoLCode template
28  {{ lookupInDict 'coAlgName }}
29  This is the end of the joyLoLCode template
30  ]=]
31
32  templates.luaCode = [=[-- A Lua file (automatically generated)
33  {{ lookupInDict 'coAlgName }}
34  This is the end of the luaCode template
35  ]=]
36
37  local joyLoL = coAlgs.joyLoL
38  local pushData, pushProcess = joyLoL.pushData, joyLoL.pushProcess
39  local pushProcessQuoted = joyLoL.pushProcessQuoted
40  local popData, popProcess   = joyLoL.popData, joyLoL.popProcess
41  local newList, newDictionary = joyLoL.newList, joyLoL.newDictionary
42  local jEval = joyLoL.eval
43
44  ------------------------------------------------------------------------
45  -- NOTE the following uses raw JoyLoL code to load the templates into the
46  -- context provided.
47
48  -- To understand this code.... **think categorically**
```

```
49
50  -- In JoyLoL a particular object in the category *is* the structure of the
51  -- data stack, while a particular arrow in the category *is* the process
52  -- stack.
53
54  -- To understand what these arrows are doing... you read the JoyLoL code
55  -- in reverse order (from a 'jEval' up).
56  ------------------------------------------------------------------------
57
58  function coAlgs.loadTemplates(aCtx)
59    pushProcess(aCtx, 'addToDict')
60    for aKey, aValue in pairs(templates) do
61      pushProcess(aCtx, 'addToDict')
62      pushProcessQuoted(aCtx, aValue)
63      pushProcessQuoted(aCtx, aKey)
64    end
65    newDictionary(aCtx)
66    pushProcessQuoted(aCtx, 'templates')
67    jEval(aCtx)
68  end
69
70  interfaces.writestatus('joyLoLCoAlg', 'loaded JoyLoL CoAlg templates')
```

166

166
166

9.8 Conclusion

Lmsfile : default

168

168
168

Bibliography

AV80 report: [author: Apt, Krzysztof R and Van Emden, M H] [institution: Department of Computer Science, University of Waterloo] [number: CS-80-12] [pagetotal: 54] [title: Contributions to the theory of Logic Programming] [type: Research report] [url: https://cs.uwaterloo.ca/research/tr/1980/CS-80-12.pdf] [year: 1980]

ACV13 Awodey, S., Coquand, T., & Voevodsky, V. (Eds.) (2013). *Homotopy Type Theory: Univalent Foundation of Mathematics*. The Univalent Foundations Program, Institute for Advanced Study. Retrieved from http://homotopytypetheory.org/book/ (first-edition-974-g068cbba)

Bil90 Billaud, M. (1990). Simple operational and denotational semantics for Prolog with cut. doi:10.1016/0304-3975(90)90197-P

Ert96 Ertl, M. A. (1996). *Implementation of Stack-Based Languages on Register Machines*. (Dissertation). Retrieved from http://www.complang.tuwien.ac.at/papers/ertl96diss.ps.gz

For04 Ford, B. (2004). Parsing expression grammars: a recognition-based syntactic foundation. In Parsing expression grammars: a recognition-based syntactic foundation., *POPL '04 Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM. doi:10.1145/982962.964011

FW08 Friedman, D. P. & Wand, M. (2008). *Essentials of Programming Languages* (, Ed.). (3rd ed.). MIT Press. Retrieved from http://www.eopl3.com/

Gai17 report: [author: Gaito, Stephen T] [institution: PerceptiSys Ltd] [title: JoyLoL] [type: Technical Report] [year: 2017]

Gai18 _____ (2018). *Hilbert's Program Revisited: Volume I: Base case: omega-computation*. (Vol. I). Perish then Publish Press.

Gia02 Giaquinto, M. (2002). *The search for certainty: A Philosophical Account of Foundations of Mathematics*. Oxford University Press.

Gor79 Gordon, M. J. C. (1979). *The Denotational description of Programming Languages: An introduction* (, Ed.). Springer-Verlag.

Gun92 Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. MIT Press. Retrieved from https://mitpress.mit.edu/books/semantics-programming-languages

Hat82 Hatcher, W. S. (1982). *The Logical Foundations of Mathematics*. Pergamon Press.

Ier08 Ierusalimschy, R. (2008). A text pattern-matching tool based on Parsing Expression Grammars. doi:10.1002/spe.892

Lak76 Lakatos, I. (1976). *Proofs and Refutations* (J. Worrall & E. Zahar, Eds.). Cambridge University Press.

Mac01 MacKenzie, D. (2001). *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press.

MAE+65 McCarthy, J., Abrahams, P. W., Edwards, D. J., Hart, T. P., & Levin, M. I. (1965). *LISP 1.5 Programmer's Manual*. (2nd Ed ed.). MIT Press.

MI08 Medeiros, S. & Ierusalimschy, R. (2008). A parsing machine for PEGs. In A parsing machine for PEGs., *DLS '08 Proceedings of the 2008 symposium on Dynamic languages*. ACM. doi:10.1145/1408681.1408683

Mey90 Meyer, B. (1990). *Introduction to the theory of programming languages*. Prentice-Hall.

Mey92 ——— (1992). *Eiffel: The Language*. Prentice Hall.

Pro01 Probst, M. (2001). *Proper Tail Recursion in C*. (Diplomarbeit). Retrieved from https://www.complang.tuwien.ac.at/schani/diplarb.ps

Sha00 Shapiro, S. (2000). *Thinking about Mathematics: The philosophy of mathematics*. Oxford University Press.

SW00 Strachey, C. & Wadsworth, C. P. (2000). Continuations: A Mathematical Semantics for handling full jumps. Retrieved from http://link.springer.com/article/10.1023/A (originally published in 1974, as technical report, PRG-11, Programming Research Group, University of Oxford)

Ten81 Tennent, R. D. (1981). *Principles of Programming Languages* (, Ed.). Prentice-Hall.

Thu94 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [pagetotal: 14] [title: Mathematical Foundations of Joy] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/j02maf.html] [year: 1994]

Thu94 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [title: Overview of the language JOY] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/j00ovr.html] [year: 1994]

Thu94 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [title: Rationale for Joy, a functional language] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/j00rat.html] [year: 1994]

Thu05 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [title: Joy website] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/joy.html] [year: 2005]

Thu11 online: [author: von Thun, Manfred] [title: Joy Programming Language] [url: http://www.latrobe.edu.au/humanities/research/research-projects/past-projects/joy-programming-language] [year: 2011]

Win93 Winskel, G. (1993). *The Formal Semantics of Programming Languages: An Introduction*. Massachusetts Institute of Technology.

172

172
172