HolloRay

A domain-specific programming language for 3D computer-aided design

Kwok Lam

Lee

101030588

Informal language specification

The language HolloRay can perform 3D rendering of simple shapes and call back end JavaFX 3D functions.

HolloRay supports 32 bit integers and reals. Additionally, string literals can be printed.

Data of the above types, as well as 3D shapes can be assigned to variables, and dereferenced later.

HolloRay supports 9 common predicates, and 6 common arithmetic operators.

HolloRay has common flow controls such as if-else, while-loop, for-loop, and procedure.

HolloRay has the capability to support flexible number, and type-mixed function arguments.

HolloRay has the following domain specific types representing common shapes: box, cube, sphere, cylinder, cone, torus, tetrahedron, pyramid.

HolloRay provides the following DSL-type statements for manipulating solids: translate, rotate, scale. They operate on one solid at a time and accept 3 reals as arguments x, y, z.

For full detailed description please see `informalAndInternal.txt`.

Internal syntax signatures

For full detailed description please see `informalAndInternal.txt`.

eSOS rules

The eSOS rules are extensively expanded to ensure all internal constructors can take all the type literals that make sense, as well as any expressions that eventually resolve to those literals.

A set of grammar rules for `println(...)` which allow an arbitrary number of printable expressions to be passed, separated by commas. The high level idea is to catch the cases

of single, and current print elements plus the rest which has a recursive nature. The single case could mean there is only a print element, or it is the last one. Either case, printing a newline character will be appropriate. For the recursive case, printing a space character after each print element is appropriate.

Limited support for conversion between integers and reals.

There is much more to do. However, eSOS is simply not scalable. For most rules, there need to be variants for handling other variable types, resolving expression, and different arities.

External syntax parser generating internal syntax trees

A set of grammar rules for `println(...)` which allow an arbitrary number of printable expressions to be passed, separated by commas. The high level idea is to make use of the hat operator such that the statements will expand to a sequence of `seq(seq(...), …)`.

Rules are set up such that flexibility is provided. For instance, `if` statements can have optional `else` statements.

It is expressions-everywhere. The parser is kept as general as possible, accepting expressions that will eventually resolve to the appropriate literals.

3D solids can be instantiated by invoking their respective “constructor”s. They can be stored in variables. Behind the scenes, it is just an integer ID to keep track of these solids. The assigned variables can be dereferenced like any other variables.

Dot notation is used to provide an intuitive way of invoking CAD operations on a particular solid. For example, `mySolid.rotate(x, y, z);`.

Common comparators and logical operators are implemented.

Nested brackets evaluation is supported, like `(!(-(8-9)**2))`.

Compound assignment, like `x += 1;`.

For-loop in theory can take a valid HolloRay program as its first and third argument.

Attribute evaluation based interpreter

This interpreter includes the features mentioned in the above parser, except dot notation for CAD solids because of a technical difficulty between the interpreter and invoking JavaFX methods.

Additionally:

Compound assignment with default value 0 for undefined variables. There are also guards implemented to ensure no null references.

Optional body for for-loop and while-loop, useful for minimal predicates that have side effects.

Optional 1st and 3rd argument for for-loop. It would be a walk in the park to make the 2nd argument optional as well. However, I decided against it because otherwise it defeats the advantage of for-loop, specifying a terminating condition.

Procedure definition and invoking.

The order of arithmetic operations is explicit and ensured.

There is a huge room of potential improvement. However, Kwok Lam ran out of time.

Example domain specific programs and tests

They can be found under the `programs/` directory. These are targeted towards the external-to-internal parser. They include classical procedural programs like fibonacci sequence and the 3x+1 conjecture. There is also a `shape` program for spawning and manipulating 3D solids.

For the interpreter specifically, the `AttrAct.str` at root directory is a long program with very verbose calls to test HolloRay against expected behaviours.

A substantial amount of time was spent designing a solution for unifying boolean and integers in a systematic way such that relational, arithmetic, and logical operators would work on both boolean and integers. That was too big of an ambition. There are traces of related test cases to be found in `AttrAct.str`.

Special characters need to be double-escaped, like `System.out.println("Hello World!\\n")`. This is most likely because this string is first transformed into a Java interpreter, during which one level of escape is lost.

Plugin

The plugin is engineered with great care in mind. It is extremely modular and readable, and thus future engineers can easily expand this in a boilerplate manner.

The plugin also provides public access, so invoking its methods from the interpreter by `valueUserPugin.<method>()` is possible.

User-specified JavaFX screen dimension.

Graphical rendering is by default deferred till the end of the program for performance enhancement, but can still be invoked manually at any point.

Summary of achievements

- C like syntax which allows a big target audience.

- Conservative and explicit in syntax design with the aim of avoiding confusion and improving readability.
- Optional statements as arguments/body for control flow statements such as for-loop, while-loop, if-else.
- Emphasis on generality, that is to accept expressions as high-up in the abstraction level as possible and allow them to resolve to literals eventually.
- Covers majority of mathematical expressions, even nested, a strongly desired aspect for CAD purpose.
- Allows 3D solids to be manipulated as variables, and be dereferenced and worked on by CAD operations.
- Some support for type conversion between integers and reals.
- High Plugin code quality.
- A painful but fruitful boost in knowledge.