STL Script Documentation

A domain-specifc programming language to specify and evaluate Signal Temporal Logic formulae.

Contributor(s)

Simon Chu
Eunsuk Kang

December 1, 2020

1 Motivation

Signal temporal logic (STL) is a temporal logic formalism for specifying properties of continuous signals. It is widely used for analyzing programs in cyber-physical systems (CPS) that interact with physical entities exhibiting continous dynamics.

However, neither the industry nor academia has agreed upon a common standard of STL, and the non-standardized technique has bring about the discrepancies in analyzing system properties, and resulting in the industry and academia spending a tremendous amount of effort implementing ad hoc/specialized APIs that are tailored to specific use cases, cannot communicate with each other, requires steep learning curve for new personnel that just got involved in the project, and causing inefficiencies when trying to apply the same reasoning techniques across projects.

A programming language is a bicycle for the mind. By standardizing STL using a programming language, it effectively support collaboration across projects and disciplines, effectively reduce the overhead for training new personnel, and developing ad hoc tools for each project, which allows developer to focus on reasoning about the STL specifications instead of its implementation and verification.

I will introduce the concrete syntax and semantics of the language in the following sections.

2 Hello, World! in STL Script

```
println "Hello, World!"
```

`println` is a built-in function in STL Script. It will print whatever expression is behind and append a `\n` character at the end of the expression to create a newline afterwards. `"Hello, World!"` is a string, which is one of the primitive types in the language.

3 Usage

3.1 Standalone Program

User can specify a standalone program in a file with suffix `.stl`. For example, the following program is specified in a file called `hello.stl`.

```
// hello.stl
println "Hello, World!"
```

To run the program, user can execute the following command on the command line (note that `$` is not part of the command, it indicates that the command must be executed on a shell).

```
$ stl hello.stl
```

The user will then obtain the following result from the command line.

```
Hello, World!
```

3.2 REPL

To start REPL (read-eval-print loop), execute `stl` command on the command line like follows.

```
$ stl
```

Once the command is executed, you will see the following interface.

```
STL Script v1.0.0
Copyright © 2020 Carnegie Mellon University. All rights reserved.
>>> |
```

From this point on, you are allowed to type and evaluate expressions and statements in the STL Script. Note that the vertical bar `"|"` indicates the location of the curser. We can print `"Hello, World!"` by typing `println "Hello, World"` in the REPL interface as follows. The result of execution will be displayed in the subsequent line, followed by >>>, where you are supply more STL expressions or statements.

```
>>> println "Hello, World!"
Hello, World!
>>> |
```

3.3 API (in progress)

```
interally: STL_Spec object: STL specification
-----------------
G[0, 10]($x > $y)(2, signal)
---------------------------
interally: STL_Eval_Expr object: STL evaluation expression


API/internally: evaluated to STL_Eval_Expr
                ---------------------------------------
stl_eval_expr = STL("G[0, 10]($x > $y)").eval(2, signal)
                                                ------
                                        signal in JSON object format

# two way of obtaining robustness -> Float ("+" for satisfy, "-" for violation of the STL proper
signal_robustness = stl_eval_expr.robustness()
signal_robustness = STL("G[0, 10]($x > $y)").robustness(2, signal)

# two way of obtaining satisfiability -> True/False
signal_satisfy = stl_eval_expr.satisfy()
signal_satisfy = STL("G[0, 10]($x > $y)").satisfy(2, signal)

# two way of obtaining probability of a signal satisfying the condition -> Float (between 0 - 1)
signal_prob_eval_expr = STL("P[0, 10]($x > $y)")
signal_robustness = stl_prob_eval_expr.probability()
signal_robustness = STL("P[0, 10]($x > $y)").probability(2, signal)
```

4 Preliminaries

4.1 Primitive Types

STL Script is a statically-typed language, it supports both implicit or explicit typing, and it has the following primitive types built into the language:

- Int
- Float
- String
- Boolean
- List
- Tuple
- STL
- STL_Expr: Specify STL Formulas
- Signal: (add time(stamp) as a signal entryHell)

Most of the primitive types are similar to those in programming languages like Scala or Java. Note that `List` is a parametric type. It can accept another primitive type as an argument. For example, `List<Int>` indicates a `List` consists of Integers (of `Int` Type). `STL` type is assigned to STL formulas.

4.2 Values, Expressions and Statements

4.2.1 Values

Values are language components that cannot be evaluated any further. They are the building block for Expressions. Instances of Values can be `3.1415926`, `42`, $\cdots$

4.2.2 Expressions

Expressions are all language components that can be evaluated to a value, and does not have any side effects (i.e. assignment to a variable, or write to standard output or file streams, etc). They are the building block for Statements.

Instances of Expressions can be `1+1`, `true||false`, $\cdots$

4.2.3 Statements

Statements are all language components that perform certain actions, and may exhibit side effects. They typically consist of Expressions, and they are followed by separators like `;` or `\n` (or both) characters.

Instances of Statements can be `val i = 3;  println "Hello, World!";`

4.3 Function Invocation

Function can be invoked on expressions. The invocation is initiated by `.` (dot). For example, in the following assignment statement of STL formula:

```
val property = G[0,1]($distance_to_boundary > 5.0)(0, signal)
```

Note that the signal in our case can be any arbitrary signal (we will discuss this later). We can evaluate the satisfaction of the STL formula by invoking the `eval()` function on the property variable like the following:

```
property.eval()
```

This will evaluate the STL formula to a `Boolean` value `true` or `false`. The invocation itself is a expression.

4.4 Signal

There are two way of specifying a signals in JSON, namely, signal with index, and signal with both index and timestamp.

The following is the signal specification with only index. The Signal API will automatically use the signal index (in the case below, "0" and "1", respectively) as the timestamp of each signal content. Note that the index of the signal must start with "0" instead of "1". The index must be of `string` type.

```
{
    "0" : {
        "content" : {
                "param" : 7
        }
    },

    "1" : {
        "content" : {
                "param" : 10
        }
    },

    ...
}
```

The following signal is equivalent to the signal above.

```
{
    "0" : {
        "content" : {
                "timestamp" : 0.0,
                "param" : 7
        }
    },

    "1" : {
        "content" : {
                "timestamp" : 1.0,
                "param" : 10
        }
    },

    ...
}
```

Note that when quantifiing signals using STL formulas, it will first look at if "timestamp" field exists in the signal (quantifiable), then using the index to quantify