1 Readme

This document is to help you get started with your practical. *Note*, you should fully read the practical specification before referring to this document. The task is to implement a model checker that,

- Accepts a transition system TS, an asCTL constraint c and an asCTL formula f and
- Returns whether or not the transition system TS satisfies the formula f under the constraint c - that is, whether or not TS satisfies f only taking into account paths that satisfy c.

In order that you may spend more time on the implementation of the model checker, an *asCTL* parser has been provided. The following two sections explain the expected input format of the *asCTL* parser and the equivalent Java representations that input formulae are transformed to.

2 Input Formula

The formula parser reads in formulae in a JSON format. The JSON file must contain a single JSON object with one mandatory member of type string: `"formula"`. Other fields are described later.

```
{
    "formula":"..."
}
```

As the name suggests, the `"formula"` field contains the *asCTL* formula. The practical specification already describes the grammar of an asCTL formula. The table below gives the mapping from symbols to their encoding as expected by the parser:

Symbol	Description	Input Encoding
$\neg$	negation	`!`
$\wedge$	and	`&&`
$\vee$	or	`||`
$\exists$	there exists	`E`
$\forall$	for all	`A`
$\mathcal{U}$	until	`U`
X	next	`X`
F	eventually	`F`
G	always	`G`
true	true	`TRUE`
false	false	`FALSE`
ap	atomic proposition	`[a-z0-9]+`

Note, the parser is case-sensitive. Where letters are used to denote symbols such as $\forall$ `TRUE` or $\exists$, these symbols must always be upper case. Identifiers used for atomic propositions must consist of one or more *lower case* letters or numbers.

Brackets Are Important

Operators such as $\mathcal{U}$, `&&` and `||` require bracketing. Below are some example formulae. All brackets used in the examples are mandatory.

```
(p && q)
(fox || dog)
AG p
EF critical
AF (you && me)
A(s U t)
E(TRUE U (EF (EG p)))
```

Specifying action constraints

In order to add action constraints, one or more sets of actions must be added to the JSON object as JSON String arrays. Each member of type string array is treated as an action set which can be referred to in the formula string by using the array member key. For example:

```
{
    "formula":"EF (s aUb t)",
    "a":["action1","action2"],
    "b":["action3","action4"]
}
```

Here the until constraint has two action sets a and b associated with it.

3 Java Representation

The Java representation is close to that of the grammar tree. For example, each possible type of state formula (such as $\neg\Phi \mid \Phi \wedge \Phi \mid \exists\phi \mid \ldots$) are represented with a Java class (`Not.class, And.class, ThereExists.class, ...`) that extend a `StateFormula` class. The same is done for the path formulae. These objects are built up into a tree representation of the parsed formula allowing formulae to be explored recursively - see the method `writeToBuffer()` as an example (this method is used to recurse through and print formulae). Figure 1 shows an example of how a formula may be represented as a tree of Java objects.

Note, you can verify that the correct formula has been parsed simply by calling print on the formula objects returned by the parser.

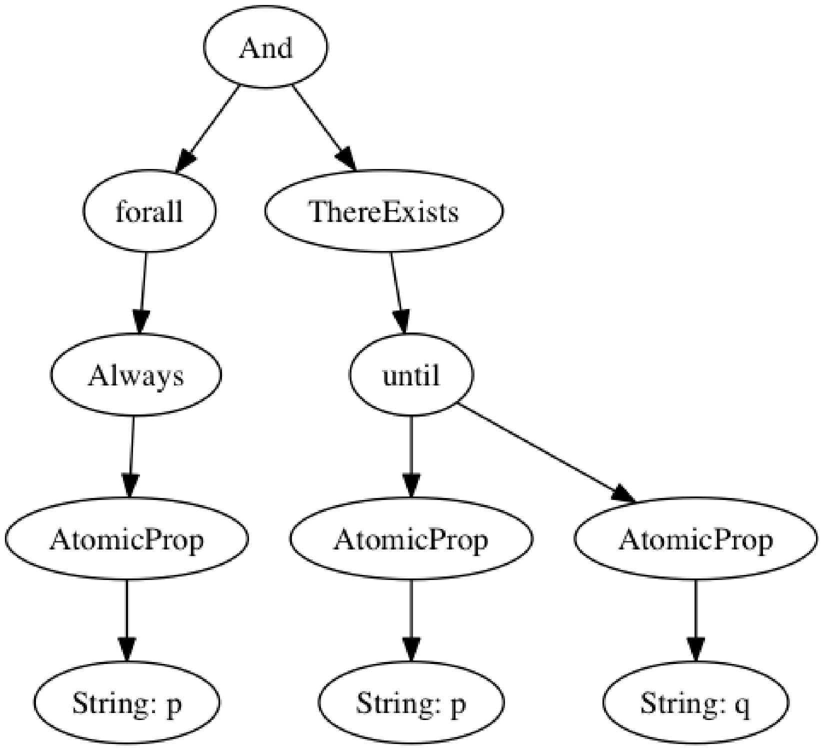
4 Parsing Transition Systems

The transition systems (the models) are also specified in the JSON format. The JSON representation is a very literal description of a transition system (a set of state objects composed with a set of transition objects). As there are many good ways in which a transition system may be represented in your Java model checker, a specific representation has not been given. Rather, the state and transition JSON objects are parsed into exact Java equivalent objects. It is very likely that you may want to transform this into a model that is more easily handled by model checking algorithms.

5 Running

You are to construct a set of JUnit test cases that demonstrate your model checker. Example models and formulae are provided in `src/test/resources`.

Figure 1: ((AG p) && E(p U q))



An example JUnit test case can be found in in `src/test/java/modelChecker` that demonstrates how to parse and test a model against a formula.

The code must be compiled and tested using gradle. To build the project using gradle, the syntax is as follows:

```
$ cd <project_root>
$ ./gradlew clean build test coverage
```

- clean is a gradle task to clean the build directory
- build is a gradle task to build the project (e.g compileJava)
- test is a gradle task to run the unit testing
- coverage is a gradle task to generate the test coverage report