ScaRTEC Manual

Ioannis Kontopoulos

June 8, 2017

1 Installation and Software Requirements

To use this version of RTEC, first we need to install Java and Scala. It is recommended to install Java 8 and Scala version 2.11.7, although theoretically it should work with Java 7 and versions of Scala from 2.10.4 or higher.

To install Java version 8 open a terminal and type the following commands :

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

To confirm the installation type :

```
$ java -version
```

To install Scala version 2.11.7 open a terminal and type :

```
$ wget http://www.scala-lang.org/files/archive/scala-2.11.7.tgz
$ tar zxf scala-2.11.7.tgz
$ sudo mv scala-2.10.4 /usr/share/scala
$ sudo ln -s /usr/share/scala/bin/scala /usr/bin/scala
$ sudo ln -s /usr/share/scala/bin/scalac /usr/bin/scalac
$ sudo ln -s /usr/share/scala/bin/fsc /usr/bin/fsc
$ sudo ln -s /usr/share/scala/bin/sbaz /usr/bin/sbaz
$ sudo ln -s /usr/share/scala/bin/sbaz-setup /usr/bin/sbaz-setup
$ sudo ln -s /usr/share/scala/bin/scaladoc /usr/bin/scaladoc
$ sudo ln -s /usr/share/scala/bin/scalap /usr/bin/scalap
```

To confirm the installation type :

```
$ scala -version
```

2 A simple example

Let‘s start with a simple example that illustrates the functionality of the application as well as the input format.

Suppose that Chris is having an all but ordinary day. He goes to work in the morning and in the afternoon he finds out that he has won the lottery. In the evening he goes to the pub, but loses his wallet. Finally, he goes home at night. We want to know whether Chris is happy or not, as these actions take place. Our story has three events, “`got_to`”, “`lose_wallet`” and “`win_lottery`”, and three properties, “happy”, “location” and “rich”.

We would like to specify the conditions that make Chris happy. Being rich is such a condition. Another condition could be being at the pub. Therefore, the ‘union’ of these two conditions can meet the needs of a happy man in our scenario. Winning the lottery causes someone to be rich. For the sake of the example, let‘s assume that losing your wallet causes you to stop being rich.

Now that we have specified the rules for our example we are ready to give them as input to our application. Create a new text file, “definitions.txt” and type the following:

```
> InitiatedAt [rich X = true] T
     HappensAt [win_lottery X] T

> TerminatedAt [rich X = true] T
     HappensAt [lose_wallet X] T

> InitiatedAt [loc X = Y] T
     HappensAt [go_to X Y] T
```

```
> HoldsFor [happy X = true] I
      HoldsFor [rich X = true] I1
      HoldsFor [loc X = pub] I2
      Union_All [I1 I2] I
```

Now that we have defined our rules we need a companion file that will describe all of the events, fluents and entity identifiers of our scenario. Create a new file, "declarations.txt" and type the following:

```
InstantEvents {
      Input: [go_to 2]
      Input: [lose_wallet 1]
      Input: [win_lottery 1]
}

Fluents {
      Simple: [loc 1 = home]
      Simple: [loc 1 = pub]
      Simple: [loc 1 = work]
      Simple: [rich 1 = true]
      OutputSD: [happy 1 = true]
}

InputEntities {
      InputPerson 1:
            [lose_wallet]
            [win_lottery]
      InputGoTo 2:
            [go_to]
}
```

```
BuiltEntities {
        Person 1:
                [InputPerson()]
                [InputGoTo(0,1)]
}

CachingOrder {
        [loc 1 = _]          -> Person
        [rich 1 = _]         -> Person
        [happy 1 = true]     -> Person
}
```

In this example there is a section called "InputEntities". In this section, we tell the application which entity identifiers it will use in the recognition form the input dataset. Specifically, we use as identifier the name "InputPerson". This identifier will be the parameters extracted from `lose_wallet` and `win_lottery` events. Similarly, "InputGoTo" will be the parameters extracted from `go_to` events. Then, there is a second section called "BuiltEntities". In this section, we tell the application how to build the entities from the input ones. In this example, the identifier "Person" will use the parameters from "InputPerson" identifier as well as the parameters from the "InputGoTo" identifier. Note that in the latter we use numbers inside the parenthesis. This means that it will take only the first parameter from the identifier. The numbers are indexes, where the first number is the "from" statement and the last number is the "until" statement. Finally, we need to give in the caching order the identifiers that each output event will use. For this example, every output event will use the Person identifier.

Finally, to put the application to the test we need to provide the events or dataset. Create a new file, "dataset.txt" and type the following:

```
HappensAt [go_to chris work] 9
HappensAt [win_lottery chris] 13
HappensAt [go_to chris pub] 17
HappensAt [lose_wallet chris] 19
HappensAt [go_to chris home] 21
```

The events above describe the scenario mentioned earlier. Specifically, at 9:00 Chris goes to work. At 13:00 he finds out that he has won the lottery. Then, at 17:00 he goes to the pub. Afterwards, at 19:00 he loses his wallet and finally at 21:00 he returns home.

In order for our application to read these files we need to put them in a folder called "input". The folder is in the same directory with our jar file or in the home directory of the project if we run it from the IDE. When we finally run the application it will output a file called "recognition.txt" that contains the following:

```
loc(chris)=pub,[(18,22)]
rich(chris)=true,[(14,20)]
loc(chris)=work,[(10,18)]
loc(chris)=home,[(22,inf)]
happy(chris)=true,[(14,22)]
```

The above output suggests that Chris is at work from 10:00 to 18:00, at the pub from 18:00 to 22:00 and at home from 22:00 to infinity.

For more details on the Event Calculus, see: `https://github.com/aartikis/RTEC`