RTEC-UnitTester
Manual

1 Introduction

This document provides a short manual on the RTEC-unit-tester and guidance on creating a simple set of unit tests. RTEC-UnitTester provides a way of testing the recognition results of RTEC using a set of narratives, rules, and test cases provided by the user (a human).

2 Tests file

Testing RTEC with the UnitTester requires a set of compiled rules and their declarations, one or more narratives and a declaration file of the tests. In this section, we will describe the needed test-cases declarations for a "tests-file", and provide guidance for the creation of a simple "tests-file", testing the correctness of the recognised intervals using the following rule:

```
initiatedAt(runningTest(X)=true,T):-
  happensAt(starts_test(X),T).
terminatedAt(runningTest(X)=true,T):-
  happensAt(ends_test(X),T).
```

2.1 Narratives

A narrative must be declared in one of the following three forms:

```
%Type A
updateSDE(exampleScenario) :-
  assert(happensAtIE(starts_test(unit_tester),1)),
  assert(happensAtIE(ends_test(unit_tester),10)).
%Type B
updateSDE(exampleScenario,5) :-
  assert(happensAtIE(starts_test(unit_tester),1)).
updateSDE(exampleScenario,10) :-
  assert(happensAtIE(ends_test(unit_tester),10)).
%Type C
updateSDE(exampleScenario,0,5) :-
  assert(happensAtIE(starts_test(unit_tester),1)).
updateSDE(exampleScenario,5,10) :-
  assert(happensAtIE(ends_test(unit_tester),10)).
```

Type A uses only a scenario ID value, type B uses additionally an extra argument expressing the end time of the narrative and type C uses two extra arguments expressing the start time and the end time of the narrative. Narratives of type B and type C can be useful when testing unsorted narratives or running tests with windows. When the narratives are of type B or C, care must be taken when choosing the values of step and windowsize so that the start time in type C narratives is equal to the $CurrentTime - WM$ value of each recognition query and end time is equal to the $CurrentTime$. If the test case declaration uses a type A narrative then all input events are asserted before event recognition starts, if the test case declaration uses a type B or C then the appropriate narrative is loaded in each recognition step. To start creating your first tests with the RTEC-UnitTester create a narratives file with the above prolog code and additionally create the appropriate rules and declarations file in the RTEC form.

2.2 Test Cases

There are three ways of declaring a test case depending on the type of the narrative. $testCase[E/SE](ScenarioID, TestName, TestID, ExpectedResuts, (Step, WM, StartTime, EndTime))$ is a fact holding information for identifying the test case, the narrative, and the values required for each recognition step. $ScenarioID$ is the name of the narrative, $TestName$ is the name of the test (e.g., cycles), $TestID$ is a number used for the identification of each test, $ExpectedResults$ is a list containing the expected result for each recognition query, and the tuple argument contains the necessary values for event recognition. $Step$ is used for the calculation of query times, WM is the window size, $StartTime$ is the time from which we calculate the first query time by adding $Step$, and $EndTime$ is the time of the last query time. For example the tuple (5,6,0,20) would create the following event recognition (ER) periods (-1,5],(4,10],(9,15],(14,20]. However, as you probably have noticed in the $testCase[E/SE]$ declaration there isn't information regarding the testing fluent, to this end for each $testCase$ a $check(TestName, TestID, Interval)$ rule describing the check must be defined, $TestName$ and $TestID$ are the test name and the test id of the corresponding test case and $Interval$ is the variable in which the recognised intervals must be assigned. For example consider the following testcases:

```
%uses Type A narrative
testcase(exampleScenario,example,1, [[(2,11)]],(10,10,0,10)).

%uses Type B narrative
testcaseE(exampleScenario,example,2, [[(2,inf)],[(2,11)]],(5,5,0,10)).

%uses Type C narrative
%expected results of the first query is wrong
testcaseSE(exampleScenario,example,3, [[(2,3)],[(2,11)]],(5,5,0,10)).

%defining a check rule for the above test cases
check(example,N,I):-
    member(N,[1,2,3]),
    holdsFor(runningTest(unit_tester)=true,I).
```

To continue creating your first UnitTester tests create a prolog file loading the UnitTester, the compiled rules, the declarations, the narratives and then copy the above prolog code. This file, is what we call the "tests-file".

3 Running the tests

3.1 Inside prolog

Running the unit tests, simply requires loading a tests definition file (e.g., 'example_test.prolog') and then type "runtests('example_test.prolog').". An execution of the set of unit tests described in this manual is the following:

```
?-['example_test.prolog'].
?-runtests('example_test.prolog').
Case name: example        Case number: 3   Status: failed
--ER number: 0
---TP: [(2,3)]
---FP: [(3,inf)]
---FN: []

Passed (3 tests)
Failed (1 tests)
yes
```

As you can see, tests 1,2 passed and test 3 failed as expected. ER number is the number of the Event Recognition period starting from zero.

3.2 From command line

Running the unit tests from the command line can be performed with the following command:

```
yap -quiet -l example_test.prolog -g runtests -- example_test.prolog
```

The 'quiet' flag is used to set the Prolog flag verbose to silent, the 'l' flag is used to load the prolog file, the 'g' flag to set the goal, and the argument after '--', is the name of the tests definition file.

3.3 Running all tests inside the tests folder

In order to perform all the provided RTEC tests, navigate to the 'unit-tests/tests/' folder and run the 'runalltests.sh' bash script.