Online appendix for the paper ***An Event Calculus Production Rule System for Reasoning in Dynamic and Uncertain Domains*** *published in Theory and Practice of Logic Programming*

Theodore Patkos, Dimitris Plexousakis
Institute of Computer Science, FO.R.T.H.
E-mail: {patkos, dp}@ics.forth.gr

Abdelghani Chibani, Yacine Amirat
Lissi Laboratory, Paris-Est Créteil
Val-de-Marne University (UPEC)
E-mail: {chibani,amirat}@u-pec.fr

submitted 30 December 2014; revised 13 July 2015; accepted 14 December 2015

Appendix A Extracts from the Resources to Aid the Reader

A.1 Cerbere Reasoner

Cerbere as a standalone Event Calculus reasoner can be downloaded from:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/Cerbere.rar`

To run Cerbere, one needs to copy the jess.jar file in the lib folder of the reasoner. This file can be downloaded from the Jess home page. We recommend Jess version 7.1p2 and JDK 7.

Source code for examples mentioned in the paper (more are available in the */EC* files folder of the reasoner):
- Ex1. Shanahan's circuit: An example of epistemic reasoning

  - source:
    `http://www.csd.uoc.gr/~patkos/tplpAppendix/ShanahanCircuit.ec`
    sample output:
    `http://www.csd.uoc.gr/~patkos/tplpAppendix/ShanahanCircuitOutput.txt`

- Ex2. Multiple model generation:

  - source1:
    `http://www.csd.uoc.gr/~patkos/tplpAppendix/MultiModels1.ec`
    sample snapshot:

http://www.csd.uoc.gr/~patkos/tplpAppendix/MultiModels1Snapshot.JPG

- source2:
  http://www.csd.uoc.gr/~patkos/tplpAppendix/MultiModels2.ec
  sample snapshot:
  http://www.csd.uoc.gr/~patkos/tplpAppendix/MultiModels2Snapshot.JPG

A.2 Probabilistic Component

Sample XMLs implementing Recognition and Composition eBNs are given next:

- Recognition eBN for the Prepare Breakfast activity (XML):
  http://www.csd.uoc.gr/~patkos/tplpAppendix/PrepareBreakfastBNRecognition.xml
- Recognition eBN for the Take Shower activity (XML):
  http://www.csd.uoc.gr/~patkos/tplpAppendix/TakeShowerBNRecognition.xml
- Composition eBN for the Prepare Breakfast activity (XML):
  http://www.csd.uoc.gr/~patkos/tplpAppendix/PrepareBreakfastBNComposition.xml

A.3 Domain-Independent Evaluation

Execution codes to reproduce the results of Tables 2 and 3 can be found here (add/remove axioms to generate all cases shown in the tables):

- Experiment 1 with 1000 objects.
  http://www.csd.uoc.gr/~patkos/tplpAppendix/exp1_1k_objects.ec
- Experiment 2 with 1000 fluents.
  http://www.csd.uoc.gr/~patkos/tplpAppendix/exp1_1k_fluents.ec

A.4 The Hybrid Framework

A.4.1 Execution Instructions

The framework is designed to run over the infrastructure installed at the Lissi lab that connects the reasoner with the underlying components. Yet, one can bypass the initialization phase and run executions of the hybrid framework with simulated events manually (i.e., events entered by the user, rather than retrieved from the sensors).

One can download the version of Cerbere that automatically uploads the necessary domain axiomatizations and eBN files for activity recognition from here:
http://www.csd.uoc.gr/~patkos/tplpAppendix/CerbereHybrid.rar
Please run using Eclipse: import as a new project, press run and wait 5 seconds, while the program tries to connect with the different components (connection will be unsuccessful of course, as the server and the actual sensors will not be running).

As before, one needs to copy the jess.jar file in the lib folder of the reasoner, before running. This file can be downloaded from the Jess home page. We recommend Jess version 7.1p2:
`http://www.jessrules.com/jess/download.shtml`

To enter simulated events, please do not use the Add Event or Observation button; this will only send the events to the reasoner, but it will not trigger the 3-step reasoning cycle of the hybrid framework. Instead, click inside the Eclipse Console and press Enter; a pop up will appear where you can enter the event, as if obtained from some sensor.

A user's manual, with sample executions can be downloaded from the following location:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/User's%20Manual.pdf`

A.4.2 Source Codes

Our activity recognition framework relies on a set of Event Calculus axiomatizations that implement the rational behind our inference process. Below are some of the files uploaded to the Cerbere reasoner to recognitze activities, as well as to instruct proper actions based on them. Some of these axiomatizations are domain dependent, others are domain independent.

Domain Definition: The Basic Domain Axiomatization.ec defines the basic concepts that constitute the domain.
Definition of the basic concepts:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/basic%20domain%20axiomatization.ec`

Monitoring of ADLs: Different .ec files can be created to model how the reasoning system should recognize the different activities of daily living of an individual, e.g., take shower axiomatization.ec, brush teeth axiomatization.ec, etc. Additional axiomatization files can be added by developers to monitor other activities, along with the actions that should be executed by the system in order to assist people during these activities.
Axiomatization of the Take Shower activity:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/take%20shower%20axiomatization.ec`
Axiomatization of the Brushing Teeth activity:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/brush%20teeth%20axiomatization.ec`

Modeling of the reasoning system's behaviour: The system.ec file implement the reasoners functionality. This is an important domain-independent file that models the behavior for Possible and Recognized Activities, and also implements the main reasoning steps.
Definition of the reasoner's functionality:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/system.ec`

Spatial entities and their relations: The spatial reasoning.ec axiomatizes spatial relations for the different entities of the domain. The spatial relations con-

cerns the events and the space region in which they may occur.
Definition of parameters for spatial reasoning:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/spatial%20reasoning.ec`

A.4.3 How to reproduce the measurements of the experimental evaluation

The version of Cerbere in the following link has hard-coded the narrative of actions that were used to run our experimental evaulation (please, follow the instructions in A.4.1):
`http://www.csd.uoc.gr/~patkos/tplpAppendix/Cerbere%20for%20experimental%20evaluation.rar`

To see the results after running the reasoner, open the "Output" tab and click on the root of the tree of models in the left panel, titled "Models produced". Statistical data for each execution step are presented, as shown in the snapshot here:
`http://www.csd.uoc.gr/~patkos/tplpAppendix/statistics.JPG`
Recall that we aggregate the reasoning times for each timepoint within every round, in order to calculate the mean time of each action (each of the 6 actions causes the reasoner to progress 3 timepoints, due to the 3-step cycle; thus, the first round lasts from timepoint 1 to 18, the second from timepoint 19 to 36 etc.).

Alternatively, one can manually add the actions that constitute one round. These are:

```
Happens(DoorOpens(Ned,HallBedroom,0), -1)
Happens(DoorOpens(Ned,HallBathroom,100), -1)
Happens(TriggerAlert(NoActivity,340), -1)
Happens(TriggerAlert(NoActivity,580), -1)
Happens(DoorOpens(Ned,HallToilet,636), -1)
Happens(DoorOpens(Ned,HallBedroom,650), -1)
```

adding 15000 to the absolute time of each action after every round.

A.5 Event Calculus to Jess Parsing Methodology

For each object, fluent or event type, such as

```
sort: object(O1,O2).
fluent: F1(object,object).
event: E1(object).
```

one 'deffacts' Jess rule is created, with one tuple for each instance. For example:

```
(deffacts objects
(sort (name object) (instance O1))
(sort (name object) (instance O2)))

(deffacts fluentDEF_F1
(fluentDEF (name F1) (argSort object object)))
```

```
(deffacts eventDEF_E1
(eventDEF (name E1) (argSort object)))
```

These rules fire only once, during the initialization of the reasoner, producing all appropriate facts.

For each Event Calculus axiom, such as

```
Initiates(E1(?o2), F1(?o2, ?o1), ?t).
```

two 'defrule' Jess rules are created. The idea is not to instantiate all fluents or events, until they are needed. Recall that Cerbere assumes that all fluents that do not exist in the KB are false. As such, we can manage the size of the KB in favor of efficiency. For the above axiom, we create:

```
(defrule EFFECTAXIOMS::0_AssertEffect_Initiates_E1_F1_?o2_?o1
(Time (timepoint ?t))
?event <- (event (name E1) (arg ?o2))
(EC (predicate Happens) (event ?event) (time ?t))
(fluentDEF (name F1) (argSort ?sort0 ?sort1))
(sort (name ?sort0) (instance ?o2))
(sort (name ?sort1) (instance ?o1))
(not (fluent (name F1) (arg ?o2 ?o1)))
=>
(assert (fluent (name F1) (arg ?o2 ?o1) )))

(defrule EFFECTAXIOMS::1_Initiates_E1_F1
(Time (timepoint ?t))
?event <- (event (name E1) (arg ?o2))
(EC (predicate Happens) (event ?event) (time ?t))
?effect <- (fluent (name F1) (arg ?o2 ?o1))
=> (assert (EC (predicate Initiates)
(epistemic no)
(event ?event)
(posLtrs ?effect)
(time ?t))))
```

The first rule instantiates the fluent and fires only once. The second fires every time the preconditions of the axioms are true.

This pattern is followed for any type of axioms, such as event occurrence axioms, effect axioms, observations, trigger axioms or other. These Jess rules can become rather complicated, based on the preconditions of the axioms, such as whether negation is used, if value comparisons exist or if there are nested preconditions combining the above. For instance, the axiom

```
HoldsAt(F1(?o1, ?o2),?t) ^
{?o1 <> ?o2} ^
~Happens(E2(O1),?t)=>
Initiates(E1(?o2), F1(?o2, ?o1), ?t).
```

is parsed as the following Jess rule (only the one is shown here, the other is similar):

```
(defrule EFFECTAXIOMS::1_Initiates_posF1_negE2_E1_F1
(Time (timepoint ?t))
?event <- (event (name E1) (arg ?o2))
(EC (predicate Happens) (event ?event) (time ?t))
?effect <- (fluent (name F1) (arg ?o2 ?o1))
?f0 <- (fluent (name F1) (arg ?o1 ?o2))
(EC (predicate HoldsAt) (epistemic no) (posLtrs ?f0 )
    (time ?tf0&:(or (eq ?tf0 ?t) (eq ?tf0 -1)) ))
(test (neq ?o1 ?o2))
(not (and ?e2 <- (event (name E2) (arg O1))
(EC (predicate Happens) (event ?e2rxl ) (time ?t))
(test (eq ?e2 ?e2rxl))
))
=> (assert (EC (predicate Initiates)
(epistemic no)
(event ?event)
(posLtrs ?effect)
(time ?t))))
```

Notice how all variables are instantiated in the body of the Jess, how the condition between variables is translated into a 'test' Jess command and how the event occurrence precondition exists inside a 'not' statement.