Programming Assignment

CS6770: Knowledge Representation and Reasoning

Nipam Basumatary (**CS16B111**)

Raghav Nauhria (**CS16B113**)

Problem Statement

Implement a system that encodes a Discrete Event Calculus problem to FOL representation so that one could do deductive reasoning on a given event calculus problem using a theorem prover.

Input

Input to the problem are the Discrete Event Calculus Axioms and the Domain Description.

Domain description consists of the following:

- Domain dependent fluents, events and declaration of other sorts.
- Domain dependent axioms: action descriptions and state constraints in terms of Initiates ( ) , Terminates ( ) , Releases ( ) and HoldsAt ( ) .
- Initial conditions in terms of HoldsAt ( ) .
- A sequence of events in terms of Happens ( ) .

In the sentence given below, [xyz] means forall xyz and {xyz} means exists xyz (where x, y and z are variables).

EC Axioms:

[fluent, time]

(HoldsAt (fluent, time) &

! ReleasedAt (fluent, time+1) &

```
   ! ( {event} Happens ( event, time ) &
Terminates ( event, fluent, time ) ) ) ->
HoldsAt ( fluent, time+1 ) .

[fluent, time]
( ! HoldsAt ( fluent, time) &
   ! ReleasedAt ( fluent, time+1 ) &
   ! ( {event} Happens ( event, time) &
Initiates ( event, fluent, time ) ) ) ->
! HoldsAt ( fluent, time+1 ) .

[fluent, time]
( ! ReleasedAt ( fluent, time) &
   ! ( {event} Happens ( event, time ) &
Releases ( event, fluent, time ) ) ) ->
! ReleasedAt ( fluent, time+1 ) .

[fluent, time]
( ReleasedAt ( fluent, time ) &
   ! ( {event} Happens ( event, time ) &
      ( Initiates ( event, fluent, time) |
      Terminates ( event, fluent, time ) ) ) ) ->
ReleasedAt ( fluent, time+1 ) .

[event, fluent, time]
( Happens ( event, time ) & Initiates ( event, fluent, time ) ) ->
( HoldsAt ( fluent, time+1 ) & ! ReleasedAt ( fluent, time+1 ) ) .

[event, fluent, time]
( Happens ( event, time ) & Terminates ( event, fluent, time ) ) ->
```

(! HoldsAt (fluent, time+1) & ! ReleasedAt (fluent, time+1)) .

[event, fluent, time]
(Happens (event, time) & Releases (event, fluent, time)) ->
ReleasedAt (fluent, time+1) .

Sample Domain Description:

// declaring sorts
event Load ( ) .
event Shoot ( ) .
event Sneeze ( ) .
fluent Loaded ( ) .
fluent Alive ( ) .
fluent Dead( ) .
time 0 .
time 1 .
time 2 .
noninertial Dead ( ) .

// action descriptions
[time] Initiates (Load ( ), Loaded ( ), time) .
[time] HoldsAt (Loaded ( ), time) -> Terminates (Shoot ( ), Alive ( ), time) .
[time] Terminates (Shoot ( ), Loaded ( ), time) .

// state constraint(s)
[time] HoldsAt (Dead ( ), time) <-> ! HoldsAt (Alive ( ), time) .

// initial state
HoldsAt (Alive ( ) ,0) .
! HoldsAt (Loaded ( ) ,0) .

// sequence of events
Happens (Load ( ), 0) .
Happens (Sneeze ( ), 1) .
Happens (Shoot ( ), 2) .

Steps followed to encode/translate the axioms and sentences in EC to FOL representation

Step1: Reification

We use reification to convert event and fluent atoms into first-order terms. Reification is the technique that makes a formula of a first-order language (FOL) into a term of another first-order language (EC encoded as FOL).

Method

1. Events and fluents and with no arguments such as 'Load ( )' and 'Alive ( )' will be reified as 'load' and 'alive' respectively.
2. Events and fluents with agents as arguments are reified as follows:
    - For example, we have an event as Load (agent1, agent2)
    - We have two agent1: a11 and a12
    - We have two agent2: a21 and a22
    - We generate all the combinations of agent1 and agent2 and then 'Load (agent1, agent2)' is reified as 'load_a11_a21', 'load_a11_a22', 'load_a12_a21', and 'load_a12_a22'.

Step2: Unique-name axioms

We generate uniqueness-of-names axioms to ensure uniqueness of events and fluents terms. We specify all the reified events and fluents as distinct terms. For example,

%% Uniqueness-of-names axioms for events
load != sneeze.
sneeze != shoot.
shoot != load.

%% Uniqueness-of-names axioms for fluents
alive != loaded.
dead != alive.
loaded != dead.

Step3: Circumscription

We use predicate completion to convert second-order circumscriptions into first-order formulae. The idea behind this is to express whatever is not given in the domain as false. The following examples from the sample domain description explains the circumscribed Initiates (event, fluent, time) , Terminates (event, fluent, time) , Releases (event, fluent, time) , and Happens (event, time) axioms.

1. **Initiates (event, fluent, time) axioms**

Initiates (event, fluent, time) axiom from the sample domain description:
[time] Initiates (Load ( ), Loaded ( ), time) .

Circumscribed Initiates (event, fluent, time) axioms:
[event, fluent, time] (Initiates (event, fluent, time) ⇔
(event = load ∧ fluent = loaded)) .

2. **Terminates (event, fluent, time) axioms**

Terminates (event, fluent, time) axiom from the sample domain description:
[time] Terminates (Shoot ( ), Loaded ( ), time) .

Circumscribed Terminates (event, fluent, time) axioms:

[event, fluent, time] (Terminates (event, fluent, time) ⇔
(event = shoot ∧ fluent = loaded)) .

3. **Happens (event, time) axioms**

Happens (event, time) axioms from the sample domain description:
Happens (Load ( ), 0) .
Happens (Sneeze ( ), 1) .
Happens (Shoot ( ), 2) .

Circumscribed Happens (event, time) axioms:
[event, time] (Happens (event, time) ⇔
((event = load ∧ time = 0) ∨ (event = sneeze ∧ time = 1) ∨ (event = shoot ∧ time = 2))).

4. **Releases (event, fluent, time) axioms**

It is circumscribed in the similar way but it is possible that the domain description doesn't have a Releases (event, fluent, time) axiom, which means it's never the case that an event releases a fluent from the law of inertia. In such case, it is circumscribed as follows:

[event, fluent, time] ! Releases (event, fluent, time) .

And add the following statement to reified initial conditions:

[fluent, time] ! ReleasedAt (fluent, time) .

- **If we have a non-inertial fluent in the domain description, then we circumscribe Releases (event, fluent, time) axiom as follows:**

Non-inertial fluent declared in the sample domain description:

fluent Loaded ( ) .
fluent Alive ( ) .
fluent Dead ( ) .
noninertial Dead ( ) .

Circumscribed Releases (event, fluent, time) axioms:
[event, fluent, time] (! Releases (event, fluent, time) ⇔
(fluent = alive ∨ fluent = loaded)) .
[event, fluent, time] (Releases (event, fluent, time) ⇔
(fluent = dead)) .

And add the following statement to reified initial conditions:

[fluent, time] (! Releases (fluent, time) ⇔
(fluent = alive ∨ fluent = loaded)) .

Team Member Contributions

Code design: Nipam and Raghav
Implementation:

- Reification and uniqueness-of-names: Nipam
- Circumscription: Raghav

References

1. Mueller, Erik T., and Geoff Sutcliffe. "Discrete event calculus deduction using first-order automated theorem proving." In *Fifth Workshop on the Implementation of Logics*, p. 43. 2005.
2. McCarthy, John. *First Order Theories of Individual Concepts and Propositions*. STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE, 1979.
3. Shanahan, Murray. "The event calculus explained." In *Artificial intelligence today*, pp. 409-430. Springer, Berlin, Heidelberg, 1999.