Online appendix for the paper

Modeling and Reasoning in Event Calculus using Goal-Directed Constraint Answer Set Programming ∗

published in Theory and Practice of Logic Programming

Joaquín Arias[1]

Manuel Carro[2,3]

Zhuo Chen[4]

Gopal Gupta[4]

[1] *CETINIA, Universidad Rey Juan Carlos,* [2] *IMDEA Software Institute*
[3] *Universidad Politécnica de Madrid,* [4] *University of Texas at Dallas*
`joaquin.arias@urjc.es, manuel.carro@{imdea.org,upm.es}, {zhuo.chen,gupta}@utdallas.edu`

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Appendix A F2LP Encoding of the *Light* Scenario

The code below corresponds to what F2LP generates for the light scenario described in Section 4 using *discrete* Event Calculus. Since the directive `#domain` is not available in *clingo 5.1.1*, we had to adapt the translation of F2LP adding `timestep(1..10)` and `timestep/1` to make the clauses safe.

```
timestep(0..10).

% If a light is turned on, it will be on:
initiates(turn_on,on,T) :- timestep(T).

% If a light is turned on, whether it is red or green will be
% released from the commonsense law of inertia:
releases(turn_on,red,T) :- timestep(T).
releases(turn_on,green,T) :- timestep(T).

% If a light is turned off, it will not be on
terminates(turn_off,on,T) :- timestep(T).

```

∗ Work partially supported by EIT Digital, MICINN projects RTI2018-095390-B-C33 InEDGEMobility (MCIU/AEI/FEDER, UE), PID2019-108528RB-C21 ProCode, Comunidad de Madrid project S2018/TCS-4339 BLOQUES-CM co-funded by EIE Funds of the European Union, US NSF Grants IIS 1718945, IIS 1910131, IIP 1916206.

```
% After a light is turned on, it will emit red for up to 1 second
% and green after at least 1 second
trajectory(on, T1, red, T2) :-
    timestep(T1),
    timestep(T2),
    T1 < T2, T2 < T1 + 1.
trajectory(on, T1, green, T2) :-
    timestep(T1),
    timestep(T2),
    T2 >= T1 + 1.

initiallyN(on).

%% Actions
happens(turn_on,2).
happens(turn_off,4).
happens(turn_on,5).

%% Query
:- not query.
query :- holdsAt(red,_).
```

Appendix B Adapted F2LP Translation of the *Light* Scenario with Increased Precision

The code next shows an F2LP program for the light scenario described in Section 4, where the new predicate `precision/1` makes it possible to have a finer grain for the possible values of `timestep` by increasing the value of `P`.

```
timestep(0..10*P) :- precision(P).

% If a light is turned on, it will be on:
initiates(turn_on,on,T) :- timestep(T).

% If a light is turned on, whether it is red or green will be
% released from the commonsense law of inertia:
releases(turn_on,red,T) :-
    timestep(T).
releases(turn_on,green,T) :-
    timestep(T).

% If a light is turned off, it will not be on
terminates(turn_off,on,T) :- timestep(T).

% After a light is turned on, it will emit red for up to 1 second
% and green after at least 1 second
trajectory(on, T1, red, T2) :-
    timestep(T1),
    timestep(T2),
    precision(P),
    T1 < T2, T2 < T1 + (1*P).
```

```
trajectory(on, T1, green, T2) :-
    timestep(T1),
    timestep(T2),
    precision(P),
    T2 >= T1  +  (1*P).

initiallyN(on).

%% Actions
happens(turn_on,2*P) :- precision(P).
happens(turn_off,4*P) :- precision(P).
happens(turn_on,5*P) :- precision(P).

%% Query
:- not query.

precision(10).
query :- holdsAt(red,59).
```

Appendix C Adapted F2LP Translation of the *Water Level* Scenario with Increased Precision

The next figure shows an F2LP program for the water level scenario described in Section 4, where the new predicate `precision/1` makes it possible to have a finer grain for the possible values of `timestep` and for the level of water by increasing the value of `P`.

```
timestep(0..30*P) :- precision(P).

% Two possible worlds - vessel size = 10 or size = 16
max_level(10*P) :-
    precision(P),
    not max_level(16*P).
max_level(16*P) :-
    precision(P),
    not max_level(10*P).

initiates(tapOn,filling,T) :- timestep(T).
terminates(tapOff,filling,T) :- timestep(T).

initiates(overflow,spilling,T) :-
    timestep(T),
    max_level(Max),
    holdsAt(level(Max), T).

% Note that (S1.3) has to be a Releases formula instead of a
% Terminates formula, so that the Level fluent is immune from the
% common sense law of inertia after the tap is turned on.
releases(tapOn,level(0),T) :-
    timestep(T),
    happens(tapOn, T).
```

```

% Now we have the Trajectory formula, which describes the
% continuous variation in the Level fluent while the Filling
% fluent holds.
trajectory(filling,T1,level(X2),T2) :-
    timestep(T1),
    timestep(T2),
    T1 < T2,
    X2 = X  +  (4 * (T2 - T1)/3),
    max_level(Max),
    X2 <= Max,
    holdsAt(level(X),T1).

trajectory(filling,T1,overlimit,T2) :-
    timestep(T1),
    timestep(T2),
    T1 < T2, X2 = X  +  (4 * (T2 - T1)/3),
    max_level(Max),
    X2 > Max,
    holdsAt(level(X),T1).

% Now we have the Trajectory formula, which describes the
% continuous variation in the Leaf fluent while the Spilling
% fluent holds.
trajectory(spilling,T1,leak(X),T2) :-
    timestep(T1),
    timestep(T2),
    holdsAt(filling, T2),
    T1 < T2, X = 4 * (T2 - T1) /3.

initiallyP(level(0)).

%% Actions
% The next formulae ensures the Overflow event is triggered when
% it should be.
happens(overflow,T) :- timestep(T).

% Here is a simple narrative. The level is initially 0, and the
% tap is turned on at time 5.
happens(tapOn,5*P) :- precision(P).

%% Query
:- not query.

precision(100).
query :- holdsAt(level(1100),_).
```