Results from Implementing the Event Calculus in CLP($\mathcal{R}$)

Imperial College Research Report No. DoC 95/17

K J Dryllerakis

Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London, SW7 2BZ, UK
Email: kd@doc.ic.ac.uk

October 1995

(Originally published May 1992)

Abstract

This report presents an implementation of the Event Calculus [KS86, Ser90, Sha90b, Sha90a] under the contraint logic programming language CLP($\mathcal{R}$). Constraint logic programing [JLM86, JL86, JL87] is then put into context; its advantages and limitations are judged and proposal for solutions are made. The description of the implementation is given in *phases* corresponding to levels of increasing difficulty. All phases are meant to follow a general scheme so that the examples under consideration should be solved under all phases.

Keywords

event calculus,constraint logic programming, CLP(R),constructive negation, continuous change

1 About the Event Calculus

The Event Calculus is a formalism for describing physical situations. It is based on the notion of an *event* as being the cause for initiation and termination of *properties*. Properties describe the state of the world. Events describe its history. Time is not explicitly introduced in the original formulation of the event calculus. Nevertheless a linear ordering of events is assumed so that in the more recent approaches events are always associated with a time point. So, reasoning about time reduces to reasoning about real numbers.

Many different implementations of the event calculus exist under the standard Prolog environment. The limitations of these systems are based on the inability of PROLOG to deal with arithmetic expressions. The system has to provide its own constraint solver which, in most of the times, is of limited power.

Recent implementations of event calculus try to tackle a more difficult problem; continuous change of properties (like the height of a falling ball) or simultaneous effects from events seem to be the heralds of the description of more complicated situations. The original formalisation tends to get out of hand when all these are taken into account.

2 About the CLP scheme

Constraint Logic Programming first appears in 1986 [JLM86] when a general scheme for constraint logic languages is specified by Jaffar. These languages use the same formalism as logic programs but work in a completely different way. Terms are now strongly typed; a particular type represents the chosen field of computations which is only an instance of the general scheme. Reasoning within a particular domain of computation reduces work to the intended interpretation. In our work, we will use CLP($\mathcal{R}$), the constraint logic programming language dealing with real numbers. Several researchers prefer an interval based approach to time. A constraint programming language on the domain of real number intervals may prove more suitable for their needs (no such implementation exists as far as we know).

3 Scope and Extent of Material

The main guidelines in developing an implementation of Event Calculus in CLP($\mathcal{R}$) will be *modularity* and *extensibility*. Modularity in this context refers to partitioning any given problem to a set of event calculus axioms and a set of domain specific knowledge. The aim is to achieve a general enough theory of event calculus that will allow many different types of problem to be described under the same set of axioms. The programs presented belong to a set of *phases* –starting from an easy and leading to a difficult example. Extensibility describes the idea presented in software design as backwards compatibility. Every new description of the axioms module should be able to handle the (less complicated) examples of the previous phases.

In general the Event Calculus axiom module will consist of a set of clauses describing when a property holds. The knoweldge base module will describe the history of events and the effect of these events in certain properties. The combination of both modules assisted by an inference engine will be able to supply answers on properties. Please note that our approach deals with time points as a neccessary factor of events which is not the case for general purpose event calculus. That in effect does not limit the applicability of our implementation which can be easily converted to deal with situation where the exact time ordering is not known.

4 Phase A – The light switch

4.1 Description of the Problem

We begin our implementation facing an easy *change problem*. Concider a light switch in a country house of the 50s, which turned the lights on and off by turning a knob. Each consecutive turn will alter the previous state of a lamp. The property we will be studying is the state of the lamp. We will allow two states lamp(on) and lamp(off).

4.2 The Event Calculus Axioms

First of all let us describe the event calculus axioms. The first axiom states:

Axiom 4.1 *A property P holds at a time T_1 if at a time T_2 before T_1 ($T_2 < T_1$) an event E happens which initiates the property P and there is nothing to prove that during the time interval T_2 and T_1 something happens to alter this fact.*

In CLP($\mathcal{R}$) this axiom will be written:

```
holds_at(Property,TimeA):-
        TimeB<TimeA,
        happens_at(Event,TimeB),
        does_initiate(Event@TimeB,Property),
        not property_invalid_in_period(TimeB,TimeA,Property).
```

where the operator @ was defined as:

```
?- op(900,xfx,'@').
```

The next axiom describes the interrupted intervals:

Axiom 4.2 *A property P is invalid in a time interval $[T_1, T_2)$ if at a time T_3 in this interval an event E happends which terminated the property.*

The transcription of this axiom in CLP($\mathcal{R}$) is

```
property_invalid_in_period(TimeA,TimeB,Property):-
        TimeA <= TimeC,
        TimeC < TimeB,
        happens_at(Event,TimeC),
        does_terminate(Event@TimeC,Property).
```

To complete the Event Calculus theory we specify the meaning of does_initiate and does_terminate predicates: lemma storage is ment to be used.

```
does_initiate(Event@Time,Property):-
        already_proved(initiates(Event@Time,Property)),
        !.
does_initiate(Event@Time,Property):-
        initiates(Event@Time,Property),
        assertz(already_proved(initiates(Event@Time,Property))).
```

```
does_terminate(Event@Time,Property):-
        already_proved(terminates(Event@Time,Property)),
        !.
does_terminate(Event@Time,Property):-
        terminates(Event@Time,Property),
        assertz(already_proved(terminates(Event@Time,Property))).
```

4.3 Stating the Problem in the EC

Having described the axioms of our theory we come to describe the specific problem. As mentioned before this description constitutes of a set of events, and a set of rules for initiation and termination of properties.

4.3.1 Static Description

Let us imagine the light bulb being ON at time 0. Every two minutes somebody turns the switch until his watch shows 6 minutes from the begining of time. This situation is described by a set of events:

```
happens_at(turn_switch,0).
happens_at(turn_switch,2).
happens_at(turn_switch,4).
happens_at(turn_switch,6).
```

4.3.2 Dynamic Behavior

Next the dynamics of the problem is described in the form of initiation and termination of properties:

```
initiates(turn_switch@TimeA,lamp(NewState)):-
        holds_at(lamp(OldState),TimeA),
        opposite(OldState,NewState).
terminates(turn_switch@TimeA,lamp(OldState)):-
        holds_at(lamp(OldState),TimeA).
opposite(on,off).
opposite(off,on).
```

4.3.3 Initial Conditions

Finally the initial state of the system is described:

```
?-assert(holds_at(lamp(on),0)).
```

4.4 Using CLP($\mathcal{R}$) to interprent our program

After loading all this information on the CLP($\mathcal{R}$) machine (and also defining the dynamic predicates explicitly) we can ask questions on the properties with respect to time. Using negation as failure in the first axiom limits the questions to be asked at a given time point or a given property. This

means that if both Property and Time are unkown, negation will become unsafe therefore resulting to loss of correctness. Here are some sample queries:

```
1 ?- holds_at(X,1).
X = lamp(off)
2 ?- holds_at(X,3).
X = lamp(on)
3 ?- holds_at(X,20).
X = lamp(on)
4 ?- holds_at(X,1.999).
X = lamp(off)
```

Note that if we are to ask

```
?- holds_at(lamp(on),X).
```

we will get the incomplete answer

```
6 < X
*** Retry? y
X = 0
```

because of unsafe negation.

Suppose we substitute negation with constructibe negation for the real numbers. Then the answers we will get from the same question will be:

```
X <= 4
2 < X
*** Retry? y
6 < X
*** Retry? y
X = 0
```

Furthermore we can ask more global queries like:

```
?- holds_at(X,Y).
```

which gibes the answer(s):

```
X = lamp(off)
Y <= 2
0 < Y
*** Retry? y
X = lamp(on)
Y <= 4
2 < Y
*** Retry? y
X = lamp(off)
```

```
Y <= 6
4 < Y
*** Retry? y
X = lamp(on)
6 < Y
*** Retry? y
X = lamp(on)
Y = 0
```

5 Phase B – The leak problem

Having solved the descrete valued property problem we move on to a more complex problem were properties can take up continuous values. Continuous properties are very common in physics and are mostly time dependent. This field is usually defined as dynamic change in the event calculus bibliography.

5.1 Description of the problem

Let us a create an imaginary system comprising of two identical tanks placed on the same level and orientation and being connected with a leaking pipe at the middle of their height. In order to work with a more descrete example let us further imagine that the height of each tank is 20 units. Furthermore let us start from a state where tank A is filled with water, tank B is completely empty and the leak rate from the pipe is 2 units. What is going to happen after time 0?

5.2 The EC Axioms

As before we first describe the axioms of the event calculus. A new operator is introduced to describe the value of a property being denoted by ==>. The first axiom is like the axiom in the previous phase.

Axiom 5.1 (Two-valued Properties) *A property P holds at a time T_1 if at a time T_2 before T_1 ($T_2 < T_1$) an event E happens which initiates the property P and there is nothing to prove that during the time interval T_2 and T_1 something happens to alter this fact.*

or in CLP($\mathcal{R}$) :

```
holds_at(Property,TimeA):-
        TimeB<TimeA,
        happens_at(Event,TimeB),
        does_initiate(Event@TimeB,Property),
        not property_invalid_in_period(TimeB,TimeA,Property).
```

Next we need an axiom for many valued properties:

Axiom 5.2 (Multi-Valued Properties) *A property P had the value V at a time point T_1 if at a time T_2 before T_1 an event E happens which initiates another property P' and this property can cause P to take the value V at time T_1 and nothing contradicts that the property P' is valid during this time period.*

The (optimised) CLP($\mathcal{R}$) code is:

```
holds(Property==>Value,@Time1):-
        Time2<Time1,
        happens_at(Event,Time2),
        causes(AnotherProperty,Time2,Property==>Value,Time1),
        does_initiate(Event@Time2,AnotherProperty),
        not property_invalid_in_period(Time2,Time1,AnotherProperty).
```

We also allow properties to hold if they are initiated by a trigerring condition :

Axiom 5.3 (Conditional Triggering) *A property P had the value V at time T_1 if a condition C can initiate the property and this Condition holds at a time T_2 before T_1 and there is nothing to contradict that the property P holds at that interval.*

Again the oprtimised CLP($\mathcal{R}$) code is:

```
holds_at(Property,Time1):-
        Time2<Time1,
        initiates_by_condition(Property,Condition),
        holds_at(Condition,Time2),
        not property_invalid_in_period(Time2,Time1,Property).
```

An axiom is needed to assure future persistance for continuous properties terminated by the extinction of their cause :

Axiom 5.4 (future persistance) *A property P holds at time T_3 if at some previous time T_1 and event E happens which initiates a property P' that can cause property to hold at some time T_2 after T_1 and before T_3; furthermore the property P' is terminated by a condition C which holds at the time (T_2) while our property P holds instanteously at T_2 and there is nothing to contradict that P' holds between the times T_2 and T_1.*

The transcription of this somewhat complicated axiom in CLP($\mathcal{R}$) is

```
holds_at(Property,Time3):-
        Time1<Time2,
        Time2<Time3,
        happens_at(Event,Time1),
        causes(AnotherProperty,Time1,Property,Time2),
        does_initiate(Event@Time1,AnotherProperty),
        terminates_by_condition(AnotherProperty,Condition),
        holds_at(Condition,Time2),
        holds(Property,@Time2),
        not property_invalid_in_period(Time2,Time1,AnotherProperty).
```

Finally an axiom for interrupted intervals is given to handle interruption by terminating condition:

Axiom 5.5 (interrupted intervals) *A property P is invalid in a the time period $[T_1, T_2)$ if either an event E happens at some time $T_3 \in [T_1, T_2)$ which terminates the property P or if terminating condition C holds at some time $T_3 \in [T_1, T_2)$*

This final axiom is written in CLP($\mathcal{R}$) as:

```
property_invalid_in_period(TimeA,TimeB,Property):-
```

```
        happens_at(Event,TimeC),
        TimeA <= TimeC,
        TimeC < TimeB,
        does_terminate(Event@TimeC,Property).
property_invalid_in_period(TimeA,TimeB,Property):-
        terminates_by_condition(Property,Condition),
        TimeA <= TimeC,
        TimeC < TimeB,
        holds(Condition,@TimeC).
```

To complete the event calculus machine we also have to specify the lemma initiation and termination of properties exactly as in Phase A.

5.3 Stating the Problem in the EC

Certain new features have to be noted in this description of the event calculus. Two different kind of properties exist and two different ways that a property can hold are defined. A property can hold instantly at a time point and have a specific value; the value of the property is not bounded and can be from any domain describable by the interpreting language.

5.3.1 Static Description

A single event initiates the phyical system: at time 0 the valve of the interconnecting pipe is opened and liquid starts leaking from one vessel to the other. This situation is described by:

```
happens_at(open_pipe,0).
```

5.3.2 Dynamic Behavior

The dynamic behavior of the Phase-B system is not only described but the set of initiating and terminating events. We can also supply initating and terminating condition as well as causal relation between properties. For the problem in hand termination and initiation is given by:

```
initiates(open_pipe@Time1,leaking).
terminates_by_condition(leaking,level(a)==>10).
```

Causal relation as specified with the **causes/4** predicate which can be read as: given that the property P is valid at time T_1, the value of property P' is v at time T_2.

```
causes(leaking,T0,level(a)==>H,T2):-
        holds_at(level(a)==>H0,T0),
        holds_at(leak_height==>Hf,T0),
        H0>=Hf,
        holds_at(leak_rate==>K,T0),
        H=H0-K*(T2-T0).
causes(leaking,T0,level(b)==>H,T2):-
        holds_at(level(b)==>H0,T0),
        holds_at(leak_height==>Hf,T0),
        H0<=Hf,
        holds_at(leak_rate==>K,T0),
        H=H0+K*(T2-T0).
```

5.3.3 Initial Conditions

Finally the initial conditions of the problem are:

```
asserta(holds_at(level(a)==>20,0)).
asserta(holds_at(level(b)==>0,0)).
asserta(holds_at(leak_height==>10,_)).
asserta(holds_at(leak_rate==>2,_)).
```

5.4 CLP($\mathcal{R}$) interpretation of the program

After loading the program and data on the CLP($\mathcal{R}$) engine we can query the system on the properties that hold at some point. The more difficult queries are always the indefinite ones i.e. the ones that come with free variables.

5.4.1 Property-value queries

To find the value of a property at a specific point we ask:

```
holds_at(level(b)==>X,1).
```

to get the answer

```
X = 2
```

A more general query can be the time dependence of the level for the vessel b:

```
holds_at(level(b)==>X,T).
```

which gets the answer:

```
T = 0
X = 0
*** Retry? y
X = 2*T
T <= 5
0 < T
*** Retry? y
X = 10
5 < T
```

5.5 Complicated Queries

Suppose we want to find the time when both vessels will have the same amount of liquid. The answer can be easily obtained by posing the query:

```
holds_at(level(a)==>A,T1),holds_at(level(b)==>A,T1).
```

which gives the answer

```
T1 = 5
A = 10
*** Retry? y
A = 10
5 < T1
```

5.6 Quering for all knowledge

The most general query one can ask is

```
holds_at(X,Y).
```

Here is the answer from the CLP($\mathcal{R}$) system:

```
X = leak_rate ==> 2
*** Retry? y
X = leak_height ==> 10
*** Retry? y
X = level(b) ==> 0
Y = 0
*** Retry? y
X = level(a) ==> 20
Y = 0
*** Retry? y
X = leaking
Y <= 5
0 < Y
*** Retry? y
X = level(a) ==> -2*Y + 20
Y <= 5
0 < Y
*** Retry? y
X = level(b) ==> 2*Y
Y <= 5
0 < Y
*** Retry? y
X = level(a) ==> 10
5 < Y
*** Retry? y
X = level(b) ==> 10
5 < Y
```

6 Conclusions

As it can be easily seen, CLP($\mathcal{R}$) offers a useful framework for developing the Event Calculus system for problems that are more mathematically oriented. I guess the next step (i.e. Phase-C) would be simultaneous processes affecting a single property. This is the well known example of the leaking and filling tank. Two processes (that of leaking and that of filling) affect a single property (the heigh of the liquid in the vessel). A way to takle this requirement would be to find all possible processes that can affect the property find their contribution and add all contributions up to find the final value of the property. This involves the usage of all solutions predicates which are alréady implemented in CLP($\mathcal{R}$).

Furthermore, the existance of a logic programming language with the ability to deal with mathematics will facilitate all the "physical" style situations described in the event calculus.

A Source Code – Phase-A

This is the source code for the Phase A development. The system and negation libraries need to be loaded at execution time.

A.1 Axioms (ec)

```
%
%       Declaration of operators and system dependent stuff
%
?- op(900,xfx,'@').                        % To be read as 'At'
?- ibmclpr,dynamic(holds_at,2),            % CLP(R) IBM needs this in order
        dynamic(constructive_negation,1),%
        dynamic(already_proved,1).         % to use assert and retract
%
?-printf("\n Loading Axioms for Descrere-Property (Ph-A)
                                            Event Calculus...",[]).
%
%
holds_at(Property,TimeA):-
        clpr,                                       % Working with CLPR ?
        TimeB<TimeA,
        happens_at(Event,TimeB),
        does_initiate(Event@TimeB,Property),    % Lemma storage as well
        ('  ##negation loaded' ->
        do_not(property_invalid_in_period(TimeB,TimeA,Property));
        not property_invalid_in_period(TimeB,TimeA,Property)).
%
%                       Interrupted Intervals
%  (B1)
%
property_invalid_in_period(TimeA,TimeB,Property):-
        TimeA <= TimeC,
        TimeC < TimeB,
        happens_at(Event,TimeC),
        does_terminate(Event@TimeC,Property).
%
%
%       Lemma Storing of initiates and terminates knoweledge
%
does_initiate(Event@Time,Property):-
        already_proved(initiates(Event@Time,Property)),
        !.
does_initiate(Event@Time,Property):-
        initiates(Event@Time,Property),
        assertz(already_proved(initiates(Event@Time,Property))).
%
does_terminate(Event@Time,Property):-
        already_proved(terminates(Event@Time,Property)),
        !.
does_terminate(Event@Time,Property):-
        terminates(Event@Time,Property),
        assertz(already_proved(terminates(Event@Time,Property))).
```

```
%
%
?-printf("done\n",[]).
```

A.2 Light Switch Data - (lights)

```
?-printf("\n Loading light switch (Ph-A) example...",[]).
%
%       The description of the light switching problem
%       In event Calculus.
%
% Property That holds at time 0
%
?-assert(holds_at(switch(on),0)).
%
%       B. Events that describe the situation in hand
%       ^^^^^^^^^
happens_at(turn_switch,0).
happens_at(turn_switch,2).
happens_at(turn_switch,4).
happens_at(turn_switch,6).
%
%       C. Dynamics of properties and events
%       ^^^^^^^^^^^
initiates(turn_switch@TimeA,switch(NewState)):-
        holds_at(switch(OldState),TimeA),
        opposite(OldState,NewState).

terminates(turn_switch@TimeA,switch(OldState)):-
        holds_at(switch(OldState),TimeA).
%
%       Auxiliary predicates on value space
%
opposite(on,off).
opposite(off,on).
?-printf("done\n",[]).
```

B Source Code – Phase-B

This is the source listing for the phase B implementation. The information for the lights example is exactly the same as in phase-A.

B.1 EC Axioms (ec)

```
%
%       Declaration of operators and system dependent stuff
%
?- op(900,xfx,'@').                    % To be read as 'At'
```

```
?- op(900,fx,'@').                      % again 'At'
?- op(900,xfx,'==>').                   % To be read as value of
?- ibmclpr,dynamic(holds_at,2),         % CLP(R) IBM needs this in order
        dynamic(constructive_negation,1),%
        dynamic(already_proved,1).      % to use assert and retract
%
?-printf("\n Loading Axioms for Non-Concurent Event Calculus (Ph-B)...",[]).
%
%                       Properties with discrete values
%  (A1)
%
holds_at(Property,TimeA):-
        clpr,                                   % Working with CLPR ?
        happens_at(Event,TimeB),
        TimeB<TimeA,
        does_initiate(Event@TimeB,Property),    % Lemma storage as well
        ('  ##negation loaded' ->
        do_not(property_invalid_in_period(TimeB,TimeA,Property));
        not property_invalid_in_period(TimeB,TimeA,Property)).
%
%
%               B. Properties with continuous values
%  (A2)
%
holds_at(Property,Time1):-
        holds(Property,@Time1).

holds(Property==>Value,@Time1):-
        Time2<Time1,
        happens_at(Event,Time2),
        causes(AnotherProperty,Time2,Property==>Value,Time1),
        does_initiate(Event@Time2,AnotherProperty),
        ('  ##negation loaded' ->
        do_not(property_invalid_in_period(Time2,Time1,AnotherProperty));
        not property_invalid_in_period(Time2,Time1,AnotherProperty)).
%
%  (A3)
%
holds_at(Property,Time1):-
        clpr,
        Time2<Time1,
        initiates_by_condition(Property,Condition),
        holds_at(Condition,Time2),
        ('  ##negation loaded' ->
        do_not(property_invalid_in_period(Time2,Time1,Property));
        not property_invalid_in_period(Time2,Time1,Property)).
%
%  (A4)
%
%       Persistance of continuous properties terninated by condition of the
%       causing discrete property
%
holds_at(Property,Time3):-
        clpr,
```

```
        Time1<Time2,
        Time2<Time3,
        happens_at(Event,Time1),
        causes(AnotherProperty,Time1,Property,Time2),
        does_initiate(Event@Time1,AnotherProperty),
        terminates_by_condition(AnotherProperty,Condition),
        holds_at(Condition,Time2),
        holds(Property,@Time2),
        ('  ##negation loaded' ->
        do_not(property_invalid_in_period(Time2,Time1,AnotherProperty));
        not property_invalid_in_period(Time2,Time1,AnotherProperty)).
%
% ============================================================================
%
%                        Interrupted Intervals
%  (B1)
%
property_invalid_in_period(TimeA,TimeB,Property):-
        happens_at(Event,TimeC),
        TimeA <= TimeC,
        TimeC < TimeB,
        does_terminate(Event@TimeC,Property).
%
%  (B2)
%
property_invalid_in_period(TimeA,TimeB,Property):-
        terminates_by_condition(Property,Condition),
        TimeA <= TimeC,
        TimeC < TimeB,
        holds(Condition,@TimeC).
%
% ============================================================================
%
%
%       Lemma Storing of initiates and terminates knoweledge
%
does_initiate(Event@Time,Property):-
        already_proved(initiates(Event@Time,Property)),
        !.
does_initiate(Event@Time,Property):-
        initiates(Event@Time,Property),
        assertz(already_proved(initiates(Event@Time,Property))).
%
%
does_terminate(Event@Time,Property):-
        already_proved(terminates(Event@Time,Property)),
        !.
does_terminate(Event@Time,Property):-
        terminates(Event@Time,Property),
        assertz(already_proved(terminates(Event@Time,Property))).
%
% ============================================================================

?-printf("done\n",[]).
```

B.2 Two Tanks - (twotanks)

```
?-printf("\n Loading the 2 tank (Ph-B) example...",[]).
%
%               DYNAMIC CHANGE IN THE EVENT CALCULUS
%
%       The description of the 2 tank leak problem
%       In event Calculus.
%
%       The problem:
%
%        Tank A          Tank B
%       |       |       |       |
% Ha=20 |~~~~~~~|       |       |
%       |       |       |       |
% Hl=10 |       =========       |
%       |       |       |       |
%       |       |       |       |
%       |       |       |       |
%       ---------       --------- Hb=0
%
%       A. Properties
%       ^^^^^^^^^^^^^   Name            Type            Value Space
%                       ----            ----            ------------
%                       level(A or B)   continuous      0 to h(A or B)
%                       leak_height     Constant        (0 to 30,k)
%                       leak_rate       Constant        k
%                       leaking         Process         true false
%
%       B. Initial Conditions
%       ^^^^^^^^^^^^^^^^^^^^^
?-asserta(holds_at(level(a)==>20,0)),    % Initial state of the system
  asserta(holds_at(level(b)==>0,0)),     % i.e. liquid height in each
  asserta(holds_at(leak_height==>10,_)), % container
  asserta(holds_at(leak_rate==>2,_)).
%
%       Events that initiate a change to the system
%
happens_at(open_pipe,0).
%
%       C. Dynamics of properties and events
%
initiates(open_pipe@Time1,leaking).
terminates_by_condition(leaking,level(a)==>10).
%
% Reads: given that the process of leaking is active at T0
%        what is the value of property level(a,H) at T2 ?
%
causes(leaking,T0,level(a)==>H,T2):-
        holds_at(level(a)==>H0,T0),
```

```
        holds_at(leak_height==>Hf,T0),
        H0>=Hf,
        holds_at(leak_rate==>K,T0),
        H=H0-K*(T2-T0).

causes(leaking,T0,level(b)==>H,T2):-
        holds_at(level(b)==>H0,T0),
        holds_at(leak_height==>Hf,T0),
        H0<=Hf,
        holds_at(leak_rate==>K,T0),
        H=H0+K*(T2-T0).
%
?-printf("done\n",[]).
```

References

[JL86] J Jaffar and J-L Lassez. Constraint logic programming. Technical Report 74, Monash University, Department of Computer Sience, Clayton,Victoria 3168, Australia, June 1986.

[JL87] J Jaffar and J-L Lassez. Constraint logic programing. In *Proc 14th ACM POPL Conf, Munich*, January 1987.

[JLM86] J Jaffar, J-L Lassez, and M. J. Maher. A logic programming language scheme. In D DeGroot and G Lindstrom, editors, *Logic Programming: Relations, Functions and Equations*. Prentice-Hall, 1986.

[KS86] Robert Kowalski and Marek Sergot. A logic based calculus of events. *New Generation Computing*, (4):67–95, 1986.

[Ser90] Marek Sergot. Lectures on event calculus, 8 1990.

[Sha90a] Murray Shanahan. Representing continuous change in the event calculus. In *Proceedings of ECAI 90*, 1990.

[Sha90b] Murray Shanahan. Towards a calculus for temporal and qualitative reasoning. Technical report, Department of Computing Imperial College, 2 1990.