Learning and revising dynamic temporal theories in the full Discrete Event Calculus

Oliver Ray

Dept. of Computer Science, Univ. of Bristol, csxor@bristol.ac.uk

Abstract. This paper shows how a nonmonotonic ILP system XHAIL can perform general-purpose learning and revision of temporal theories in a full-fledged *Discrete Event Calculus* (DEC) framework with several features (now introduced into ILP for the first time) for dynamically releasing fluents from the commonsense law of inertia and subjecting them to mathematically-defined gradual change trajectories. First, we review key aspects in the development of the DEC and prior attempts aimed at learning within simple fragments of it. Then, we introduce a new formulation of DEC, called the eXploratory Event Calculus (XEC), which (a) results in significantly reduced grounding size and execution times with respect to state-of-the-art translations of DEC from circumscriptive logic into Answer Set Programming (ASP), and which (b) provides a unifying framework that supports different policies used (explicitly or implicitly) in prior work to resolve conflicts between concurrent events competing to determine the truth value and/or release status of fluents under a circumscriptive or answer set semantics. Finally, we give the first known demonstration of learning and revision of theories in the full DEC by using XHAIL and XEC in a proof-of-principle example showing how such theories can be revised under different conflict resolution policies.

Keywords: Theory Revision · Temporal Reasoning · Discrete Event Calculus

1 Introduction

This paper shows how a nonmonotonic ILP system XHAIL [29] can perform general-purpose learning and revision of expressive temporal theories in a full-fledged *Discrete Event Calculus* (DEC) framework [26] which has many core features (now being introduced into ILP for the first time) for dynamically releasing fluents from the commonsense law of inertia and subjecting them instead to mathematically-defined gradual change trajectories.

The present work supersedes previous applications of ILP to variants of the Simplified Event Calculus (SEC) [31] which include inertial fluents with positive/negative effects, fluent/action preconditions and trigger axioms, but which exclude static/dynamic fluents, concurrent/disjunctive events, cumulative/-cancelling effects, causal/effect constraints and trajectories/antitrajectories – all of which are essential for modelling realistic temporal domains [26].

To enable the possibility of running XHAIL on a wide spectrum of benchmarks encompassing both the nonmonotonic logic programming tradition and the circumscriptive classical logic tradition of Event Calculus (EC) development, the first main contribution of this paper is to introduce a novel formalisation of the DEC, called the eXploratory Event Calculus (XEC), which offers at least four major advantages over previous EC axiomatisations:

- it supports all the different policies used (explicitly or implicitly) in prior work to resolve various conflicts between concurrent events competing to set the truth value of fluents and/or their release from the law of inertia [19];
- it provides an efficient (native) non-monotonic logic programming encoding of the DEC which may be optionally supplemented with additional axioms to simulate a circumscriptive classical logic semantics, if the need arises;
- it uses a new new encoding of trajectories that scales much better in Answer Set Programming (ASP) and Boolean Satisfiability (SAT) by eliminating from prior work any axioms quantified over more than two time points;
- XEC significantly outperforms state-of-the-art translations of DEC from 2nd-order circumscriptive logic into ASP by F2LP [17] when tested on the same benchmarks previously used to show the superiority of F2LP over the prior state-of-the-art reduction of DEC into SAT by DEC-Reasoner [24].

To provide the first known demonstration of incremental learning and revision of fully-featured DEC domain theories, the second main contribution is to exploit the combination of XHAIL and XEC in a running example which shows how such theories can be automatically revised to accommodate new observations or goals using different conflict resolution and semantics. This work uses an open-source ASP instance of XHAIL [6] but could be adapted (modulo nondeterminism) to work with an Abductive Logic Programming (ALP) instance of XHAIL [29].

2 Background and Related Work

Since even a brief introduction to the full range of features supported by the DEC would require more space than available here, the reader is referred to the excellent chapter-by-chapter treatment in [26] which explains the features progressively added to the the Simple/Full/Extended-EC of [31], along with variants in [19] which inspired the Original/Simplified/Basic/Continuous/Discrete-EC in [25]. The aim of this paper is not to provide a technical introduction to these features, but rather to demonstrate how ILP is ideally poised to play a leading role in terms of their exploitation in practical reasoning tasks.

From a historical perspective, the EC evolved over four decades through four main paradigms: logic programming [16] (80s); circumscriptive logic [19] (90s); SAT [24] (00s); and ASP [17] (10s). The modern reliance on bottom-up SAT/ASP systems has shifted focus from infinite continuous timelines to finite discrete timelines which can be efficiently grounded and solved. This was enabled by the axiomatisation of DEC [24] with provided equivalent discrete linear encodings for 8 of 10 continuous EC axioms that were all originally quantified over (at least) three time-points – leaving just two cubic axioms for (anti)trajectories.

Although the DEC was originally used as the basis of a translation into SAT by a tool known as the DEC-Reasoner [23], the same axiomatisation is still used by current state-of-the-art reasoners that are now based on general translation from 2nd-order circumscriptive logic into ASP using and approach called F2LP [17]. The superiority of F2LP was demonstrated on a range of benchmark problems [17, Fig.5] covering the full spectrum of features (including durative events and compound actions which are not strictly included in the DEC, but are realised in a well-known variant of the continuous EC [26, App.C]).

From its inception, the DEC has been formulated as a *"a version of the event calculus in which initiating and terminating an fluent at the same time causes inconsistency"* [24, Sec.2]. And, following its widespread adoption, most current work has followed suit. But this contrasts with earlier work where *"simultaneously initiating and terminating a fluent simply gives rise to two sets of models (one in which the fluent is true immediately afterwards and one in which it is false), rather than resulting in an inconsistent theory"* [19, Sec.2.4].

The fact that all benchmarks to date have been carefully hand-crafted to avoid any problematic conflicts on their intended EC variant has led to a total lack of research into the following topics – which all become important when domain theories can be machine learnt: there is no study of other conflicts types (such as when concurrent events compete to bind and free a dynamic fluent to and from the inertial frame); there is no study of other policy options (such as holding or flipping the status quo or preferring one outcome over another); and there is no study of which policies might be preferable in specific applications.

There is also a notable lack of research on inductive inference in the EC literature, which is focused almost exclusively on deduction and abduction over hand-crafted domain theories [26]. ILP is one of very few fields with a history of temporal induction, with roots going back more than 15 years to the learning of domain theories in various action languages [15,18,9,27] including several variants of the SEC on which the systems Clint [30], Progol [22], Alecto [21] and (an ALP instance of) XHAIL [28] were used in theory completion tasks.

The deployment of the first ASP instance of XHAIL [29] paved the way for two more realistic software engineering applications of SEC [3,7] which developed into two successful lines of work in temporal learning [1,2] and revision [8,5] based on two techniques first proposed in [29][1]: one (p.334) for reducing theory induction to abduction using meta-predicates `try` and `use` and one (p.338) for reducing theory revision to abduction using meta-predicates `try`, `del` and `ab`.

XHAIL has been credited with inspiring development of many subsequent systems specifically optimised for various aspects of practical EC reasoning: e.g ILED [13], OLED [11], INSPIRE [14], and I2XHAIL [20], Several enhancements have also been made to these systems [12,10] and the underlying EC formalism. [4,32]. Recently [33] used belief revision on a doxastic extension of the SEC. But, no work has yet been done on induction over theories with dynamic fluents or gradual change (or almost any other non-trivial DEC feature); and no work has yet been done on comparing different conflict policies or semantics.

[1] Note the published manuscript [29] was submitted in 2007 but only printed 2009.

3 The eXploratory Event Calculus (XEC)

It is unsurprising EC conflict policies have been given little attention when all DEC benchmarks have been carefully hand-crafted to avoid any possibility of simultaneously stop/start-ing a fluent or bind/free-ing it to/from the inertial frame. But, prior work has shown conflicts can be useful (e.g. to implement nondeterminism) and it is not unreasonable to imagine that certain policies may naturally apply to certain fluents under certain conditions in certain domains. Clearly, if such choices are left implicit and arbitrary (as they often are), then domain theories may not work across DEC instances or reasoners; and machine-learnt extensions or revisions may not respect prior assumptions.

To tackle these issues, XEC offers a set of conflict resolution policies which can be explicitly enabled and customised; and it also offers a set of optional clauses which can be asserted, if need be, to effect a simulation of classical semantics within its native logic programming semantics. This is achieved by the ASP encoding of the full DEC in Fig. 1, which has four parts parts: (i) axioms for *intertial* fluents and their optional Stop-Start policies; (ii) axioms for *dynamic* fluents and their optional Bind-Free policies; (iii) the new axioms for *trajectories* that scale quadratically with time; and (iv) optional clauses which can be used to simulate a classical semantics.

XEC recognises several *types* of fluent: *inertial* and *noninertial* denote frame (aka. primitive) fluents `F` and non-frame (aka. derived) fluents `N`, respectively, and their extents should be disjoint; *dynamic* fluents `D` identify inertial fluents that may be dynamically released from the frame; *monitored* fluents `M` are inertial fluents that may be used to sustain (anti)trajectories; and *controlled* fluents `C` are dynamic or noninertial fluents that may be controlled by (anti-)trajectories. Intuitively, *noninertial* fluents are not subject to the commonsense law of inertia and their truth will be determined by state constraints and trajectories; while *inertial* fluents are subject to inertia unless they are *dynamic* and released

The core *axioms* of XEC are superficially similar to existing translations of DEC into ASP such as those obtained from F2LP or given in Fig.15 of [26], but there are three key differences: First, XEC is written in a logic programming style which only requires to formalise when `holdsAt` and `releasedAt` are true. This means we can avoid several choice literals and integrity constraints shown at the end of the listing if it is not required to strictly enforce inertial integrity and/or compute unsupported classical models. While these can be added back to obtain a classical variant of XEC, if need be, they are arguably inefficient, unnecessary and undesirable from a logic programming point of view

Second, XEC is written so that the core axioms only fire in the absence of *conflicts*. This results in a conservative default policy of stopping/binding a fluent when a conflict occurs. In turn, this allows the possibility of overriding those policies by uncommenting the starred conflict detection clause corresponding to the default option and uncommenting a conflict resolution clause corresponding to one of five given substitutes: apply the converse of the default policy (Start/Free); make a non-deterministic choice (Pick); maintain the status quo (Hold); change the status quo (Flip); or declare a logical inconsistency (Fail).

```
%%%%%%%%%%%%% i. INERTIAL (& NONINERTIAL) FLUENTS %%%%%%%%%%%%%%%

 #domain inertial(F). #domain noninertial(N). #domain event(E).
 #domain time(T;T1;T2). #domain succ(R,S). succ(T1,T2):-T2=T1+1.

 % STOP-START CONFLICT (SSC) RESOLUTION POLICY OPTIONS          %
 % (uncomment default* and another option to override):         %
 %-------------------------------------------------------------%
 % ssc(F,T) :- stops(F,T), starts(F,T).                        %     STOP*
 % holdsAt(F,S) :- ssc(F,R), not releasedAt(F,S).              %     START
 % {holdsAt(F,S)} :- ssc(F,R), not releasedAt(F,S).            %     PICK
 % holdsAt(F,S) :- ssc(F,R), holdsAt(F,R), not releasedAt(F,S). % HOLD
 % holdsAt(F,S):-ssc(F,R),not holdsAt(F,R),not releasedAt(F,S). %       FLIP
 % :- ssc(F,T).                                                %     FAIL

 holdsAt(F,S)  :-  starts(F,R), not stops(F,R), not releasedAt(F,S).
 holdsAt(F,S)  :-  holdsAt(F,R), not stops(F,R), not releasedAt(F,S).
 starts(F,T)  :-  happens(E,T),  initiates(E,F,T).
 stops(F,T)  :-  happens(E,T),  terminates(E,F,T).

%%%%%%%%%%%%%%%%%%%% ii.DYNAMIC FLUENTS %%%%%%%%%%%%%%%%%%%%%%%%

 #domain dynamic(D).

 % BIND-FREE CONFLICT (BFC) RESOLUTION POLICY OPTIONS          %
 % (uncomment default* and another option to override):        %
 %------------------------------------------------------------%
 % bfc(D,T) :- binds(D,T), frees(D,T).                        %     BIND*
 % releasedAt(D,S) :- bfc(D,R).                               %     FREE
 % {releasedAt(D,S)} :- bfc(D,R).                             %     PICK
 % releasedAt(D,S) :- bfc(D,R), releasedAt(D,R).              % HOLD
 % releasedAt(D,S) :- bfc(D,R), not releasedAt(D,R),          %       FLIP
 % :- bfc(D,T).                                               %     FAIL

 releasedAt(D,S)  :-  frees(D,R),  not binds(D,R).
 releasedAt(D,S)  :-  releasedAt(D,R),  not binds(D,R).
 frees(D,T)  :-  happens(E,T),  releases(E,D,T).
 binds(D,T)  :-  starts(D,T).
 binds(D,T)  :-  stops(D,T).

 %%%%%%%%%%% iii. TRAJECTORY (& ANTITRAJECTORY) AXIOMS %%%%%%%%%%

  #domain monitored(M). #domain controlled(C).

  holdsAt(C,T1+T2) :- followT(M,T1,T2), trajectory(M,T1-1,C,T2+1).
  followT(M,S,0) :- starts(M,R), holdsAt(M,S), not releasedAt(M,S).
  followT(M,T,S) :- followT(M,T,R), not stops(M,T+R), time(T+S).

  holdsAt(C,T1+T2) :- followA(M,T1,T2), antiTrajectory(M,T1-1,C,T2+1).
  followA(M,S,0) :- stops(M,R), not holdsAt(M,S), not releasedAt(M,S).
  followA(M,T,S) :- followA(M,T,R), not starts(M,T+R), time(T+S).

%%%%%%%%%%%%%%%%%%%% iv. CLASSICAL SEMANTICS %%%%%%%%%%%%%%%%%%%%%


 % {holdsAt(F,T)}. {holdsAt(N,T)}.
 % :-holdsAt(F,S), not holdsAt(F,R), {starts(F,R), releasedAt(F,S)}0.
 % :-not holdsAt(F,S), holdsAt(F,R), {stops(F,R), releasedAt(F,S)}0.
 % :-holdsAt(F,S), holdsAt(F,R), stops(F,R), {starts(F,R), releasedAt(F,S)}0.
 % :-not holdsAt(F,S), not holdsAt(F,R), starts(F,R), {stops(F,R), releasedAt(F,S)}0.

 % {releasedAt(D,T)}.
 % :-releasedAt(F,S), not releasedAt(F,R), {frees(F,R)}0.
 % :-not releasedAt(F,S), releasedAt(F,R), {binds(F,R)}0.
 % :-releasedAt(F,S), releasedAt(F,R), binds(F,R), {frees(F,R)}0.
 % :-not releasedAt(F,S), not releasedAt(F,R), frees(F,R), {binds(F,R)}0.

```

Listing 1. eXploratory Event Calculus (XEC) with optional policies.

Note that, if more than one option is chosen, then the label positions depict a partial order in which any lower option will take priority over any other options vertically above them; and where the combination of Hold and Flip is equivalent to FAIL. Note also that it is possible to customise any of these policies and/or add new ones which might be appropriate in a specific modelling task. Note additionally that the conflict detection rules for `ssc` and `bsc` can be treated as definitional abbreviations which can be trivially unfolded from the program. They are included here only for intellectual clarity and to make the optional policy definitions slightly more compact.

Third, XEC introduces a new encoding of *trajectories* whose rules (in contrast to prior work) are quantified over at most two timepoints[2]. This is done by introducing the predicates `followT` and `followA` to recast (anti)trajectory definitions into a successor state form in the same way `holdsAt` and `releasedAt` were first recast in the DEC by eliminating `clipped` and `declipped` from the continuous EC. Intuitively `followT(f,t,k)` means that, at time `t+k`, we are `k` time steps into a trajectory we have been following since the fluent `f` was turned on at time `t`. Thus (anti)trajectory axioms in XEC are quadratic in time, as opposed to cubic, and all other (non-optional) axioms remain linear.

Note this code emulates the canonical DEC behaviour by only allowing trajectories to be cancelled by an explicit terminating event on the monitored fluent. Note also, since the Start-Stop and Bind-Free conflict policies may both override the effect an *initiates* event may have in the next state, the trajectory axioms include checks to ensure the monitored fluent did actually hold at the timepoint after the initiating event (so the initiation was not overruled by a competing termination) and the monitored fluent is not released either (so the initiation was not overruled by a competing release - which would then have meant state constraints must have activated the fluent instead of the initiating event)!

Since durative events can also be efficiently encoded in a similar way to what has just been described for trajectories, a generalised XEC was developed to support the `happens/3` predicate needed to run the F2LP Commuter benchmark, which includes a compound action. The result is a discrete projection of existing axiomatisations of durative events in the (continuous) EC [26, App.C] but the details are beyond the scope of this paper.

To validate the correctness of XEC and evaluate its efficiency when run on XHAIL, it was tested against the state-of-the-art translation of DEC into ASP by F2LP [17]. Tests were run on nine benchmarks previously used to show the superiority of F2LP over the prior state-of-the-art DEC-Reasoner [24]. All benchmarks were tested on the same timeline 0..50. The durative variant of XEC was used in one task; a causal extension was added to both XEC and DEC in one task; and the classical extension was added to XEC in three tasks. The results in Table 1 show that XEC leads to much reduced grounding size and run time.

[2] Although the last rules for `followT` and `followA` do syntactically mention three timepoints `R`, `S` and `T`, domain declarations mean that `S` is merely an abbreviation for `R+1`, and so the rule is only actually quantified over two times: `R` and `T`.

Task [maxstep=50]	F2LP(DEC)	XHAIL(XEC)
Bus Ride (disjunctive event)	**144** ms (gnd:86±11; slv:58±4) **47** kb (rul:2,618; atm:157)	**112** ms (gnd:56±4; slv:56±2) **12** kb (rul:603; atm:123)
Commuter [a] (compound event)	**1,736** ms (gnd:946±4; slv:790±7) **8,590** kb (rul:436,497; atm:7,237)	**140** ms (gnd:74±4; slv:66±2) **28** kb (rul:1,470; atm:203)
Kitchen Sink (trajectory+trigger)	**1,694** ms (gnd:940±16; slv:754±11) **8,713** kb (rul:404,507; atm:970)	**242** ms (gnd:136±9; slv:106±4) **383** kb (rul:19,664; atm:728)
Thielscher Circuit [b] (causal constraint)	**374** ms (gnd:210±9; slv:164±7) **1,084** kb (rul:66,901; atm:409)	**140** ms (gnd:78±13; slv:62±7) **58** kb (rul:2,688; atm:394)
Walking Turkey (effect constraint)	**148** ms (gnd:82±7; slv:66±7) **29** kb (rul:1,576; atm:154)	**140** ms (gnd:70±9; slv:70±7) **11** kb (rul:424; atm:155)
Falling w/AntiTraj [c] (traj.+antitraj.)	**346** ms (gnd:218±7; slv:128±2) **799** kb (rul:43,919; atm:664)	**258** ms (gnd:152±2; slv:106±2) **491** kb (rul:26,165; atm:664)
Falling w/Events [c] (traj.+rebind.)	**1,898** ms (gnd:1,056±13; slv:812±7) **9,907** kb (rul:454,657; atm:1,072)	**332** ms (gnd:190±7; slv:142±2) **884** kb (rul:44,726; atm:1,072)
Hot Air Balloon [c] (traj.+antitraj.)	**158** ms (gnd:90±0; slv:68±2) **104** kb (rul:5,790; atm:410)	**160** ms (gnd:86±2; slv:74±7) **115** kb (rul:6,602; atm:410)
Telephone1 (direct effects)	**264** ms (gnd:158±2; slv:106±2) **350** kb (rul:17,950; atm:819)	**158** ms (gnd:88±4; slv:70±0) **68** kb (rul:3,309; atm:275)

a { F2LP run on variant of (continuous) EC with durative events [26, Apndx.C];
XHAIL run on novel extension of (discrete) XEC with durative events.

b { F2LP/XHAIL both run on identical causal extension of DEC/XEC.

c { XHAIL run on classical extension of XEC.

Table 1. Comparison of execution time and grounding size of XHAIL with XEC (right) and F2LP with DEC (centre) on 9 standard benchmarks from [17, Fig.5] (left). Grounding size (number of rules; number of atoms) computed by Gringo 3.0.5 with timeline 0..50 in all tasks. Execution time (grounding time; solving time of first model) computed by Clasp 3.1.0 (averaged over 5 trials) on a Windows 10 i64 command shell running on a Dell Lattitude 5480 with Intel Core i7 2.6 GHz CPU and 16 Gb RAM.

4 XEC Theory Learning and Revision with XHAIL

XHAIL [29,6] is a nonmonotonic ILP system which operates under the credulous answer set semantics. It allows answer set programs to be annotated with *examples* (which are positive/negative ground literals representing goals or properties that a user desires to be true/false in some model of the program) and *mode declarations* (which specify syntactic constraints on the heads/bodies of clauses that may be potentially added to program in order to achieve those goals). Integer weights and priorities may be optionally attached to examples and/or mode declarations in order to modulate the built-in compression heuristic and semantic bias that seeks to pragmatically identify extensions of a given program having models with the required properties.

The following seemingly simple running example shows how XHAIL can accomplish theory learning and revision in the full-featured XEC under different conflict resolution policies. We start with an input file consisting of the code in Listing 1 – which implements the default Stop/Bind resolution policies. For convenience, we assume the conflict detection clauses for Stop-Start conflicts `ssc` and for Binf-Free conflicts `bfc` are initially included with the core axioms. On top of these core axioms, fragments of code (red labels) will be cumulatively added to the input file and and summaries of XHAIL's output will be given at key points (blue labels).

The first fragment of code (01) brings into play eleven timepoints `0,1,..,10`, a frame fluent (`f`) that is known to hold in the initial state, and an event `e` that is known to happen on all eleven of the timepoints defined so far. The `#example` directive (which can also be called a goal) tells XHAIL to try and find or constuct models where `f` holds at time `2`. The `#display` directive tells XHAIL to print out true instance of `hold/2` (when called with the -f option for displaying full output). Given this input file XHAIL will compute the one and only model of this program.

```
%%% Start with default policies: SSC=Stop; BFC=Bind
01: time(0..10). inertial(f). holdsAt(f,0). event(e). happens(e,0..10).
    #example holdsAt(f,2). #display holds/2.
```

```
>>> model: holdsAt(f,0) holdsAt(f,1) holdsAt(f,2)...holdsAt(f,9) holdsAt(f,10)
```

If another goal is added stating that `f` should be false at time `1`, then XHAIL will have to infer a hypothesis in order to make both existing goals true (as the only model currently has `f` true always). The head declarations allow XHAIL to infer atoms of the form `initiates(e,f,T)` and `terminates(e,f,T)` where `e` is a specific event, `f` is a specific fluent, and `T` is a variable representing a timepoint.

```
#example not holdsAt(f,1).
02: #modeh initiates($event, $inertial, +time).
#modeh terminates($event, $inertial, +time).
```

If the variable placemarker denoted "+" had been replaced by the constant placemarker denoted "$", then XHAIL would have returned a ground abductive

hypothesis consisting of the two facts `terminates(e,f,0)` and `initiates(e,f,1)`. But as a variable must appear in the third argument, there is no solution to this problem under a Stop SSC. This is because event e would end up simultaneously initiating and terminating f on every time point. And since the termination would always take priority, there would be no way for f to become true again at 2 once it was false at 1.

```
>>> no meaningful answers, ...
```

But if we now switch to a nondeterministic SSC policy by asserting the choice rule associated with the Pick option, then XHAIL is able to return a hypothesis. This is because the competition between the initiating and terminating effect of e on f will allow a nondeterministic choice on every time point, so the goals can now be correctly satisfied according to the following (seemingly overgeneral) hypothesis:

```
03: %%% Switch to secondary SSC policy: SSC=Pick; BFC=Bind
    {holdsAt(F,S)} :- ssc(F,R), not releasedAt(F,S).
```

```
>>> hypothesis: initiates(e,f,V1). terminates(e,f,V1).
```

On the other hand, if we override the SSC policy yet again, using the Fail option, then previous hypothesis is no longer a correct solution.

```
04: %%% Switch to tertiary SSC policy: SSC=Fail; BFC=Bind
    :- ssc(F,T).
```

```
>>> no meaningful answers, ...
```

But if some body declarations are added, then XHAIL can qualify the previous hypothesis by restricting the conditions under which initiation and termination may occur so that an SSC may be avoided.

```
05: #modeb holdsAt($inertial,+time). #modeb not holdsAt($inertial,+time).
```

```
>>> hypothesis:
      terminates(e,f,V1) :- holdsAt(f,V1), time(V1).
      initiates(e,f,V1) :- not holdsAt(f,V1), time(V1).
```

The above hypothesis dictates that f will constantly alternate being on for one even timepoints and then off for odd ones. But if we are given an extra observation that f does not actually hold at 4 (although it does at 6), then it is not possible to correctly specify *terminates* solely on the basis of whether f holds or not.

```
06: #example not holdsAt(f,4). #example holdsAt(f,6).
```

```
>>> no meaningful answers, ...
```

But suppose we supply some additional background knowledge stating when one integer is a multiple of another.

07:
```
int(0..9). multiple(T1,T2) :- T2>1, T1 #mod T2==0.
#modeb multiple(+time, $int).
```

Now XHAIL will once again find a solution.

>>>
```
hypothesis:
  terminates(e,f,V1):-multiple(V1,3),time(V1).
  initiates(e,f,V1):-not holdsAt(f,V1),time(V1).
```

The above hypothesis has e potentially initiating f whenever it becomes false; and it has e potentially terminating f on any time point that is a multiple of 3. This means f will false for one time before going true for two timepoints and then repeating this cycle.

But if we make f into a dynamic fluent and a another event f that releases f at time at 5, then XHAIL will have to infer an additional clause under the curent default BFC policy Stop. This is because, if left uncontested, the release at 5 would push f to be false at 6, which would violate a goal (in code fragment 06). But XHAIL can avoid this by effectively generating a conflict at 5 whereby the the initiation of d will take priority and keep f high at 6:

08:
```
dynamic(f). event(d). releases(d,f,T). happens(d,5).
```

>>>
```
hypothesis:
  terminates(e,f,V1):-multiple(V1,3),time(V1).
  initiates(e,f,V1):-not holdsAt(f,V1),time(V1).
  initiates(d,f,V1).
```

But if we now change to the opposite BFC policy Free, which favours releasing fluents, then this escape will no longer be possible:

09:
```
%%% Switch to secondary BFC policy: SSC=Fail; BFC=Free
releasedAt(D,S) :- bfc(D,R).
```

>>>
```
no meaningful answers, ...
```

But, now it is released, we can allow XHAIL to infer a state law to justify the activation of f at 6. But, since we don't want to trivialise the whole learning task by letting XHAIL directly abduce each example, we attach a cost of 10 (or any other high number) on this mode declaration in order to make its use ten times less preferable than other head and body declarations - which all inherit a default cost of 1.

10:
```
#modeh holdsAt($dynamic,$time)=10.
```

```
>>> hypothesis:
      terminates(e,f,V1):-multiple(V1,3),time(V1).
      initiates(e,f,V1):-not holdsAt(f,V1),time(V1).
      holdsAt(f,6).
```

At this point we may decide to accept the clauses learnt so far into the knowledge base. But as we are not yet sure if they are trustworthy, we can use the following method to add the clauses in a way that allows them to be subsequently revised. As explained in [29, p.338]: (i) we wrap each (revisable) body literal b in the clause in an atom of the form $try(n, m, b)$ - where each (revisable) clause is assigned a unique identifier m and each literal within that clause is assigned a unique identifier n; and (ii) we add a literal $not\ exception(m, h)$ - where h is the atom in the head of the clause. Then two rules for each try atom are added: one representing the possibility of deleting the literal form the clause, and one representing the possibility of keeping it. The language bias is then set to allow literals to be deleted or exceptions to be inserted.

```
11: holdsAt(f,6) :- not exception(0,holdsAt(f,6)).

    terminates(e,f,T) :-
       try(1,1,multiple(T,3)), not exception(1,terminates(e,f,T)).
    try(1,1,multiple(T,3)) :- del(1,1).
    try(1,1,multiple(T,3)) :- not del(1,1), multiple(T,3).

    initiates(e,f,T) :-
       try(1,2,not_holdsAt(f,T)), not exception(2,initiates(e,f,T)).
    try(1,2,not_holdsAt(f,T)) :- del(1,2).
    try(1,2,not_holdsAt(f,T)) :- not del(1,2), not holdsAt(f,T).

    literal(1..1). clause(0..2).

    #modeh del($literal,$clause).
    #modeh exception($clause,holdsAt(f,six)). six(6).
    #modeh exception($clause,terminates(e,f,+time)).
    #modeh exception($clause,initiates(e,f,+time)).
```

Now, any hypotheses containing rules for *del* or *exception* are treated as instructions for revising these clauses through a post-processing step. So, adding the following information which states d happens at 8 but f is false at 9 will introduce an inconsistency that can only be removed by tightening the definition of terminates. This is because, according to the original rules, once d causes f to be released at 9, then the simultaneous happening of e will both initiate f (because f is false at 9) and terminate f (because 9 is a multiple of 3) - which is not allowed under the current Fail-Free conflict policies.

```
11: happens(d,8). :- holdsAt(f,9).
```

So the following hypothesis is returned (among some others), which is viewed as an instruction to modify revisable clause 1 for the termination of f by adding

the complements of the literals in the body of the exception clause into the body of original clause.

```
>>> hypothesis: exception(1,terminates(e,f,V1)):-not holdsAt(f,V1).
```

After a renaming variables to reflect their types, we are left with the following revised version of the previously learnt clauses, which we now decide to assert as given into our final knowledge base:

```
terminates(e,f,T) :- multiple(T,3), holdsAt(f,T).
12: initiates(e,f,T) :- not holdsAt(f,T).
holdsAt(f,6).
```

The next and final part of the example demonstrates how trajectories can be learnt. The mechanism proposed below works by fitting a polynomial of some chosen degree to a set of observations in order to define a domain specific trajectory. To keep things simple we will stick to a linear regression using a polynomial of degree 1. First the timeline is extended by 5 more points 11..15 and new types are introduced to represent equation coefficients and values. Atoms linear(x0,x1,t,v) facilitates linear regression by precomputing mappings from a timepoint t to a value v using coefficients $x0$ and $x1$ to define a linear equation v=x1.t+x0. We define *beginning* and *elapsed* as synonyms for *time* and specify the monitored fluent f and the controlled fluents $g(V)$. There is also an integrity constraint stating that g can only have at most one value at any given time. The mode declarations allow us to learn the definition of a trajectory in terms of a linear equation.

```
#domain time(T). time(11..15).
#domain coeff(X0;X1;X2). coeff(0..2).
#domain value(V;V1;V2). value(0..15).

linear(X0,X1,T,V) :- V==X1*T+X0.
beginning(T). elapsed(T).
13: noninertial(g(V)). controlled(g(V)). monitored(f).
:- holdsAt(g(V1),T), holdsAt(g(V2),T), V1!=V2.

#example holdsAt(g(5),11).
#example holdsAt(g(11),14).

#modeh trajectory(f, +beginning, g(+value), +elapsed).
#modeb linear($coeff, $coeff, +time, +value).
```

When run on this task, XHAIL produces the following hypothesis:

```
hypothesis:
>>>  trajectory(f,V1,g(V2),V3) :-
        multiple(V1,3),linear(1,2,V3,V2),
        beginning(V1),elapsed(V3),value(V2).
```

This can be more clearly written as simply

```
trajectory(f,T1,g(2*T1+1)) :- multiple(T1,3)
```

In theory, this method can be generalised to arbitrary polynomials for example using the following rule to implement quadratic regression.

```
quadratic(X0,X1,X2,T,V) :- V==X2*T*T+X1*T+X0.
```

But, in practice, this method does not scale very in ASP. In contrast to the previous learning tasks which were all solved in an instant, this task (which remember is still conjoined with the 12 previous code snippets) took about 20 seconds to solve. Given the general-purpose nature of the revisions being performed by XHAIL it should be clear this example can easily be extended to learn antitrajectories, triggers, causal constants, or any other standard (or even non-standard) types of DEC axiom.

5 Conclusion

This paper introduced the eXploratory Event Calculus (XEC) as a pragmatic framework for reasoning with the Discrete Event Calculus (DEC) under an extended set of conflict policies and semantic choices. This has begun to allow the systematic comparison of existing works in terms of the explicit or implicit choices they have made; and it has begun to allow the investigation of how these choices impact upon theory learning and revision.

In contrast to the prevailing trend of translating DEC theories from 2nd order circumscriptive classic logic into ASP, the work presented here suggests that the use of compact native ASP programs may have significant conceptual and practical advantages.

This paper also presented the first axiomatisation of trajectories that does not rely upon any rules quantified over more than two timepoints; and this technique is likely to realise significant efficiency benefits for reasoning with trajectories using ASP systems on fully materialised timelines.

And the paper showed how XHAIL can be applied to the tasks of theory completion and theory revision in a fully featured XEC under different reasoning policies.

This work provides the first known demonstration of learning and revision of temporal theories with dynamic fluents and trajectories.

References

1. Alrajeh, D., Kramer, J., Russo, A., Uchitel, S.: Learning operational requirements from goal models. In: Proceedings of the 31st international conference on software engineering. pp. 265—-275 (2009)
2. Alrajeh, D., Kramer, J., Russo, A., Uchitel, S.: An inductive approach for modal transition system refinement. In: Technical communications of the international conference of logic programming ICLP. p. 106–116 (2011)
3. Alrajeh, D., Ray, O., Russo, A., Uchitel, S.: Using abduction and induction for operational requirements elaboration. Journal of Applied Logic 7(3), 275–288 (2009)
4. Artikis, A., Sergot, M., Paliouras, G.: An event calculus for event recognition. IEEE Transactions on Knowledge and Data Engineering 27(4), 895–908 (2015)

5. Athakravi, D., Corapi, D., Russo, A., De Vos, M., Padget, J., Satoh, K.: Handling change in normative specifications. In: Baldoni, M., Dennis, L., Mascardi, V., Vasconcelos, W. (eds.) Declarative Agent Languages and Technologies X, Proceeding of the 10th International Workshop (DALT 2012), Revised Selected Papers. Lecture Notes in Artificial Intelligence, vol. 7784, pp. 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
6. Bragaglia, S., Ray, O.: Nonmonotonic learning in large biological networks. In: Inductive Logic Programming. pp. 33–48 (2015)
7. Corapi, D., Ray, O., Russo, A., Bandara, A., Lupu, E.: Learning rules from user behaviour. In: in Proceedings of the 5th International Conference on Artificial Intelligence Applications and Innovations (AIAI 2009). pp. 459–468 (2009)
8. Corapi, D., Russo, A., De Vos, M., Padget, J., Satoh, K.: Normative design using inductive learning. Theory and Practice of Logic Programming 11(4-5), 783–799 (2011)
9. Fern, A., Givan, R., Siskind, J.M.: Specific-to-general learning for temporal events with application to learning event definitions from video. Journal Of AI Research 17, 379–449 (2002)
10. Katzouris, N., Artikis, A.: WOLED: A tool for online learning weighted answer set rules for temporal reasoning under uncertainty. In: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020. pp. 790–799 (2020)
11. Katzouris, N., Artikis, A., Paliouras, G.: Online learning of event definitions. Theory and Practice of Logic Programming 16(5-6), 817–833 (2016)
12. Katzouris, N., Artikis, A., Paliouras, G.: Parallel online event calculus learning for complex event recognition. Future Gener. Comput. Syst. 94, 468–478 (2019)
13. Katzouris, N., Paliouras, G., Artikis, A.: Incremental learning of event definitions with inductive logic programming. Machine Learning 100, 555–585 (2015)
14. Kazmi, M., Schuller, P., Saygin, Y.: Improving scalability of inductive logic programming via pruning and best-effort optimisation. Expert Systems with Applications 87, 291–303 (2017)
15. Klingspor, V., Morik, K.J., Rieger, A.D.: Learning concepts from sensor data of a mobile robot. Machine Learning 23(2), 305–332 (1996)
16. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing 4(1), 67–95 (1986)
17. Lee, J., Palla, R.: Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. Journal of Artificial Intelligence Research 43, 571–620 (2012)
18. Lorenzo, D., Otero, R.P.: Learning to reason about actions. In: Proc. 14th European Conf. on AI (ECAI'00). pp. 316–320 (2000)
19. Miller, R., Shanahan, M.: Some alternative formulations of the event calculus. In: Kakas, A., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II. pp. 452–490. Springer-Verlag (2002)
20. Mitra, A., Baral, C.: Incremental and iterative learning of answer set programs from mutually distinct examples. Theory and Practice of Logic Programming 18(3-4), 623–637 (2018)
21. Moyle, S.: Using theory completion to learn a robot navigation control program. In: 12th International Workshop on Inductive Logic Programming. LNAI, vol. 2583, pp. 182–197. Springer (2002)
22. Moyle, S., Muggleton, S.: Learning programs in the event calculus. In: 7th Int. Workshop on ILP. LNAI, vol. 1297, pp. 205–212. Springer (1997)

23. Mueller, E.: A tool for satisfiability-based commonsense reasoning in the event calculus. In: Proc. 7th Int. Florida AI Research Soc. Conf. pp. 147–152 (2004)
24. Mueller, E.T.: Event calculus reasoning through satisfiability. Journal of Logic and Computation 14(5), 703–730 (2004)
25. Mueller, E.T.: Event calculus. In: Handbook of Knowledge Representation, Foundations of Artificial Intelligence, vol. 3, pp. 671–708 (2008)
26. Mueller, E.T.: Commonsense Reasoning: An Event Calculus Based Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edn. (2014)
27. Needham, C.J., Santos, P.E., Magee, D.R., Devin, V., Hogg, D.C., Cohn, A.G.: Protocols from perceptual observations. Artificial Intelligence 167(1), 103–136 (2005)
28. Ray, O.: Using abduction for induction of normal logic programs. In: Proceedings of the ECAI'06 Workshop on Abduction and Induction in Artificial Intelligence and Scientific Modelling (AIAI'07). pp. 28–31 (2006), http://people.cs.bris.ac.uk/~oray/AIAI06/AIAI06.pdf
29. Ray, O.: Nonmonotonic abductive inductive learning. Journal of Applied Logic 7(3), 329–340 (2009)
30. Sablon, G., Bruynooghe, M.: Using the event calculus to integrate planning and learning in an intelligent autonomous agent. In: Proc. 2nd Europ. Workshop on Planning (EWS'93). pp. 254–265 (1994)
31. Shanahan, M.: The Event Calculus Explained, p. 409–430 (1999)
32. Skarlatidis, A., Paliouras, G., Artikis, A., Vouros, G.A.: Probabilistic event calculus for event recognition. ACM Transactions on Computational Logic (TOCL) 16(02, Article 11), 1–37 (2015)
33. Tsampanaki, N., Patkos, T., Flouris, G., Plexousakis, D.: Revising event calculus theories to recover from unexpected observations. Annals of Mathematics and Artificial Intelligence 89, 209–236 (2021)