An Argumentation Framework for Reasoning about Actions and Change

Antonis Kakas[1], **Rob Miller**[2] and **Francesca Toni**[3]

[1] Department of Computer Science,
University of Cyprus
antonis@turing.cs.ucy.ac.cy

[2] School of Library, Archive and Information Studies,
University College London, UK
rsm@ucl.ac.uk

[3] Department of Computing,
Imperial College of Science, Technology and Medicine, London, UK
ft@doc.ic.ac.uk

Abstract. We show how a class of domains written in the Language $\mathcal{E}$, a high level language for reasoning about actions, narratives and change, can be translated into the argumentation framework of Logic Programming without Negation as Failure (LPwNF). This translation enables us 1) to understand default persistence as captured by various temporal reasoning frameworks in a simple and natural way, by assigning higher priority to information about effects of later actions over effects of earlier actions; 2) to develop an argumentation-based computational model for this type of reasoning in logic programming goal-driven style.

1 Introduction

The idea of specialised *action description languages* was first introduced in [7, 8] (and here exemplified by the Language $\mathcal{A}$) with the aim that such languages could serve as specifications for theories of action and change written in different, general purpose or computation oriented formalisms. One such language is the Language $\mathcal{E}$ [10], designed to describe domains involving *narrative information*, i.e. information about actual occurrences of actions, using a basic ontology of actions, fluents and time-points inspired by the Event Calculus [12, 13].

This paper contributes to the research agenda suggested in [7, 8] by showing how a significant class of $\mathcal{E}$ domains may be translated into the argumentation framework of Logic Programming without Negation as Failure (LPwNF) [4] and thus be given an argumentation-based, goal-driven proof theory.

LPwNF uses a priority relation between potentially conflicting or incompatible pieces of information to measure the validity of arguments supporting conclusions of interest against potentially conflicting arguments. Like many formalisms for reasoning about action, the Language $\mathcal{E}$ incorporates a notion of *default persistence* over time to address the frame problem. In the LPwNF translation, the default nature of this persistence is reflected in the nature of the priority relation, which assigns higher priority to change brought about by later actions than to change brought about by earlier actions.

The full Language $\mathcal{E}$ uses a more-or-less arbitrary structure of time, so that in some ways it generalises both "branching time" formalisms (such as the Situation Calculus) and linear time formalisms (such as the Event Calculus). However, for the purposes of this paper we have made the simplifying assumption that the structure of time is isomorphic either to the the integer or to the real number line. For such structures of time, it is reasonable to assume that the number of action occurrences in a given domain is finite.

Various previous works [8, 5, 3, 2, 10] address the problem of translating various declarative specifications for reasoning about actions and change into a computation-oriented form. Most of these works translate Language $\mathcal{A}$ theories into (some extension of) Logic Programming. On the whole these translations are less direct than the one presented here, as they rely on a complex use of Negation as Failure to capture default persistence. In contrast, our argumentation-based translation provides an intuitive reformulation as well as a sound and complete computational model.

The paper is structured as follows. Section 2 reviews the *basic* Language $\mathcal{E}$ (without "ramification" statements) used in this paper. Section 3 reviews Argumentation for LPwNF. Section 4 gives the translation from $\mathcal{E}$ domains to argumentation and proves it sound and complete. Section 5 describes the argumentation-based proof theory for the result of the transformation. Proofs of all technical results can be found in an extended version of this paper at: `http://laotzu.doc.ic.ac.uk/UserPages/staff/ft/bibliography.html`.

2 A Review of the Basic Language $\mathcal{E}$

The Language $\mathcal{E}$ is really a collection of languages. The particular vocabulary of each language depends on the domain being represented, but always includes a set of *fluent constants*, a set of *action constants*, and a partially ordered set $\langle \Pi, \preceq \rangle$ of *time-points*. A *fluent literal* is either a fluent constant F or its negation $\neg F$. Given time-points T_1 and T_2, $T_1 \prec T_2$ stands for $T_1 \preceq T_2$ and $T_1 \neq T_2$.

Domain descriptions in the Language $\mathcal{E}$ are collections of statements of three kinds (where A is an action constant, T is a time-point, F is a fluent constant, L is a fluent literal and C is a set of fluent literals): **t-propositions** ("t" for "time-point"), of the form L `holds-at` T; **h-propositions** ("h" for "happens"), of the form A `happens-at` T; **c-propositions** ("c" for "causes"), of the form A `initiates` F `when` C or A `terminates` F `when` C. Any such c-proposition should be regarded as meaning "C is a minimally sufficient set of conditions for an occurrence of A to have an initiating or terminating effect on F". When C is empty, the c-propositions can be written as "A `initiates` F" and "A `terminates` F" respectively.

In this paper, unless explicitly stated otherwise, we will consider only domains without t-propositions.

The semantics of $\mathcal{E}$ is based on simple definitions of interpretations and models. Since we are primarily interested in inferences about what holds at particular time-points (i.e. the truth value of t-propositions that might or might not occur explicitly in the domain), it is sufficient to define an interpretation as a mapping of fluent/time-point pairs to truth values: an **interpretation** is a mapping $H : \Phi \times \Pi \mapsto \{true, false\}$, where Φ is the set of fluent constants and Π is set of time-points in $\mathcal{E}$. Given a set of fluent literals C and a time-point T, an interpretation H **satisfies** C *at* T iff for each fluent constant $F \in C$, $H(F,T) = true$, and for each fluent constant F' such that $\neg F' \in C$, $H(F',T) = false$.

The definition of model is parametric on the definitions of initiation- and termination-points, i.e. time-points where a c-proposition and an h-proposition combine to describe a direct effect: a time-point T is an **initiation-point** (**termination-point** resp.) **for** a fluent constant F **in** an interpretation H **relative to** a domain description D iff there is an action constant A such that (i) there is both an h-proposition in D of the form "A `happens-at` T" and a c-proposition in D of the form "A `initiates` (`terminates`, resp.) F `when` C" and (ii) H satisfies C at T. Then, an interpretation H is a **model** of a given domain description D *without* t-propositions iff, for every fluent constant F and time-points $T_1 \prec T_3$:

1. If there is no initiation- or termination-point T_2 for F in H relative to D such that $T_1 \preceq T_2 \prec T_3$, then $H(F,T_1) = H(F,T_3)$, i.e. fluents change their truth values only via occurrences of initiating or terminating actions;
2. If T_1 is an initiation-point for F in H relative to D, and there is no termination-point T_2 for F in H relative to D such that $T_1 \prec T_2 \prec T_3$, then $H(F,T_3) = true$, i.e. initiating a fluent establishes its truth value as $true$;
3. If T_1 is a termination-point for F in H relative to D, and there is no initiation-point T_2 for F in H relative to D such that $T_1 \prec T_2 \prec T_3$, then $H(F,T_3) = false$, i.e. terminating a fluent forces its truth value to $false$.

A domain description is **consistent** iff it has a model. A domain description D **entails** the t-proposition "F `holds-at` T" ("$\neg F$ `holds-at` T", resp.), iff for every model H of D, $H(F,T) = true$ ($H(F,T) = false$, resp.).

As an example we formulate a car engine domain, with action constants *TurnOn* and *Empty* and fluents *Running* and *Petrol*. The following domain D_c expresses that, in general, turning on the engine is only effective if the tank has not been emptied of petrol, and, in particular, that the tank has been emptied at time 3 and a *TurnOn* action has been performed at time 5:

TurnOn `initiates` *Running* `when` $\{Petrol\}$	(D_c1)
Empty `terminates` *Petrol*	(D_c2)
Empty `terminates` *Running*	(D_c3)
Empty `happens-at` 3	(D_c4)
TurnOn `happens-at` 5	(D_c5)

It is easy to see, for example, that D_c entails $\neg$*Running* `holds-at` 7.

For domain descriptions *with* t-propositions, the definition of model is extended by the condition:

4. For all t-propositions in D of the form "F `holds-at` T", $H(F,T) = true$, and for all t-propositions of the form "$\neg F$ `holds-at` T'", $H(F,T') = false$.

Thus, in effect, the t-propositions are like "static" constraints that interpretations must satisfy in order to be deemed models. For example, consider the domain description D'_c obtained from D_c by replacing (D_c3) and (D_c4) with the single t-proposition $\neg Running$ `holds-at` 7. Then, for an interpretation H to be a model of D'_c, necessarily $H(Running, 7) = false$. More interestingly, $H(Petrol, 5) = false$, as otherwise $H(Running, 7) = true$ (by condition 2. in the definition of model).

In the sequel we will assume that the ordered set of time-points $\langle \Pi, \preceq \rangle$ is isomormphic either to the integer or to the real numbers. This ensures that the relation $\preceq$ is a total order and that time extends infinitely forwards into the future and backwards into the past. This assumption simplifies considerably the various results, which would otherwise need to be qualified to avoid anomalous (looping, backwardly branching, fragmented, disconnected, etc.) time structures.

3 A Review of Argumentation

Argumentation has recently proved to be a unifying mechanism for most existing non-monotonic formalisms [1, 6]. In this section we review (a version of) the *LPwNF* ([4]) argumentation framework.

Let a **monotonic logic** be a pair $(\mathcal{L}, \vdash)$ consisting of a formal language $\mathcal{L}$ and a monotonic derivability notion $\vdash$ between sentences of the formal language. In the sequel we will assume

- $\mathcal{L}$ consisting of all sentences (called **rules**) $\lambda_0 \leftarrow \lambda_1, \ldots, \lambda_n$ $(n \geq 0)$, with each λ_i a classical literal, and all variables in the λ_i implicitly universally quantified from the outside; λ_0 is called the **head** and $\lambda_1, \ldots, \lambda_n$ the **body** of any such rule;
- $\vdash$ obtained by repeatedly applying the classical modus ponens inference rule $\dfrac{X \leftarrow Y, \quad Y}{X}$ with $X \leftarrow Y$ any ground instance of a sentence in $\mathcal{L}$.

Then, an **argumentation program** (relative to $(\mathcal{L}, \vdash)$) is a tuple $(B, \mathcal{A}, \mathcal{A}', <)$ consisting of a **background theory** B, i.e. a (possibly empty) set of sentences in $\mathcal{L}$, an **argumentation theory** $\mathcal{A}$, i.e. a set of sentences in $\mathcal{L}$ (the **argument rules**), an **argument base** $\mathcal{A}' \subseteq \mathcal{A}$, and a **priority relation**, i.e. an irreflexive and antisymmetric relation $<$ on the ground instances of the argument rules, where $\phi < \psi$ means that ϕ has lower priority than ψ. Intuitively, any subset of the argument base can be used to extend non-monotonically the background theory in the underlying monotonic logic, if the extension satisfies some requirements. The sentences in the background theory can be seen as non-defeasible argument rules which must belong to any extension.

One possible requirement that extensions of the background theory must satisfy is that the extension be *admissible*, namely *non-self-attacking* and able to *counterattack* any (set of) argument rules *attacking* it (see below for the formal definitions). Whereas the admissible set consists only of argument rules in the argument base, *attacks* against an admissible set of rules are allowed to be subsets of the larger argument theory.

The argumentation-theoretic notion of *attack* is formally defined as follows: a set of argument rules S' **attacks** another non-empty such set S iff there are a literal λ and sets $S_1 \subseteq S'$, $S_2 \subseteq S$ such that

(i) $B \cup S_1 \vdash_{min} \lambda$ and $B \cup S_2 \vdash_{min} \neg\lambda$, and

(ii) if $\exists r' \in S_1, r \in S_2$ s.t. $r' < r$ then $\exists r' \in S_1, r \in S_2$ s.t. $r < r'$

where $B \cup X \vdash_{min} \alpha$ iff $B \cup X \vdash \alpha$ and $\nexists X' \subset X$ s.t. $B \cup X' \vdash \alpha$. Note that, by minimality, S_2 contains a unique rule r with head $\neg\lambda$. We say that S' attack S **on the rule** r **on** $\neg\lambda$. Intuitively, a set of argument rules attacks another such set if the two sets are in conflict, by deriving in the underlying logic complimentary literals λ and $\neg\lambda$, respectively, and the subset of the attacking set (minimally) responsible for the derivation of λ is not overall lower in priority than the subset of the attacked set (minimally) responsible for the derivation of $\neg\lambda$.

Note that the notion of attack is monotonic, namely any superset of a set of argument rules attacking another still attacks the second set, because of the minimality requirement. Finally, note that a set S is **contradictory**, i.e. it derives together with the background theory B a literal λ and its complement $\neg\lambda$, iff S attacks itself.

The notion of *admissible* extension is formally defined as follows. Let a set S of argument rules be **closed** if it contains no rule whose body is not derived (via $\vdash$) from the background theory extended by S. Then, a closed subset S of $\mathcal{A}'$ is **admissible** iff $B \cup S$ is non-contradictory and for any $S' \subseteq \mathcal{A}$ if S' attacks S then S attacks S'. For any maximal (wrt set inclusion) admissible subset S, $\Delta = B \cup S$ is called an **admissible extension** of $(B, \mathcal{A}, \mathcal{A}', <)$.

The admissible extensions of an argumentation program P determine the (non-monotonic) consequences of P: a ground literal λ is a **skeptical (**resp. **credulous) consequence** of P iff $\Delta \vdash \lambda$ for every (resp. some) admissible extension Δ of P.

4 Translating $\mathcal{E}$ into Argumentation

In translating a given domain description D into an argumentation program, all individual h- and c-proposition translations are included in the background theory.

Definition 1. *The* **background theory for** *D is the theory $B(D)$ such that*

- *for all time-points T and T' and action constants A:*
  $B(D) \vdash T \preceq T'$ *iff* $T \preceq T'$, $B(D) \vdash T \prec T'$ *iff* $T \prec T'$, *and*
  $B(D) \vdash HappensAt(A, T)$ *iff "A* `happens-at` *T" is in D;*
- *for each c-proposition "A* `initiates` *F* `when` *$\{L_1, \ldots, L_n\}$" in D, $B(D)$ contains the rule*

$$Initiation(F, t) \leftarrow HappensAt(A, t), \Lambda(L_1), \ldots, \Lambda(L_n),\ and$$

– *for each c-proposition "A* `terminates` *F* `when` *$\{L_1, \ldots, L_n\}$" in D, $B(D)$ contains the rule*

$$Termination(F, t) \leftarrow HappensAt(A, t), \Lambda(L_1), \ldots, \Lambda(L_n)$$

where $\Lambda(L_i) = HoldsAt(F_i, t)$ if $L_i = F_i$, and $\Lambda(L_i) = \neg HoldsAt(F_i, t)$ if $L_i = \neg F_i$, for some fluent constant F_i, and finally

– *$B(D)$ contains no other rules.*

As an example, consider the domain description D_c in section 2. Then,

$$B(D_c) \vdash HappensAt(Empty, 3), HappensAt(TurnOn, 5)$$

and $B(D_c)$ contains the rules

$$\begin{aligned}
&Initiation(Running, t) \leftarrow HappensAt(TurnOn, t), HoldsAt(Petrol, t)\\
&Termination(Petrol, t) \leftarrow HappensAt(Empty, t)\\
&Termination(Running, t) \leftarrow HappensAt(Empty, t).
\end{aligned}$$

The remaining components of the argumentation program corresponding to D are domain independent.

Definition 2. *The* **argumentation program of** *D is $(B(D), \mathcal{A}_\mathcal{E}, \mathcal{A}'_\mathcal{E}, <_\mathcal{E})$, referred to in short as $P_\mathcal{E}(D)$, where*
• *$\mathcal{A}_\mathcal{E}$ consists of*

Generation rules: $HoldsAt(f, t_2) \leftarrow Initiation(f, t_1), t_1 \prec t_2$ $\quad (PG[f, t_2; t_1])$
$\neg HoldsAt(f, t_2) \leftarrow Termination(f, t_1), t_1 \prec t_2$ $\quad (NG[f, t_2; t_1])$

Persistence rules: $HoldsAt(f, t_2) \leftarrow HoldsAt(f, t_1), t_1 \prec t_2$ $\quad (PP[f, t_2; t_1])$
$\neg HoldsAt(f, t_2) \leftarrow \neg HoldsAt(f, t_1), t_1 \prec t_2$ $\quad (NP[f, t_2; t_1])$

Assumptions: $HoldsAt(f, t)$ $\quad (PA[f, t])$
$\neg HoldsAt(f, t)$ $\quad (NA[f, t])$

• *$\mathcal{A}'_\mathcal{E}$ consists of all the generation rules and assumptions only.*
• *$<_\mathcal{E}$ is given by (for any fluent constant F and time-points T, T', T_1, T_2):*

(i) if $T_1 \preceq T_2$ then $NP[F, T; T_1] <_\mathcal{E} PG[F, T; T_2]$ and
$PP[F, T; T_1] <_\mathcal{E} NG[F, T; T_2]$;
(ii) if $T_1 \prec T_2$ then $NG[F, T, T_1] <_\mathcal{E} PG[F, T; T_2]$ and
$PG[F, T; T_1] <_\mathcal{E} NG[F, T; T_2]$;
(iii) $PA[F, T] <_\mathcal{E} NG[F, T; T']$, $PA[F, T] <_\mathcal{E} NP[F, T; T']$,
$NA[F, T] <_\mathcal{E} PG[F, T; T']$ and $NA[F, T] <_\mathcal{E} PP[F, T; T']$.

In other words, the effects of later events take priority over the effects of earlier ones and thus (i) persistence rules have lower priority than "conflicting" and "later" generation rules, (ii) "earlier" generation rules have lower priority than "conflicting" "later" generation rules. Moreover, (iii) assumptions have lower priority than "conflicting" generation rules. For example, given D_c as in section 2,

$$PA[Running, 5] <_{\mathcal{E}} NG[Running, 5; 3]$$
$$PG[Running, 7; 5] <_{\mathcal{E}} NG[Running, 7; 3].$$

We will show that the above translation is sound and complete, with respect to the semantics for the Language $\mathcal{E}$, for *fluent-independent* domain descriptions (see the definition below) with a *finite number of h-propositions*. We will need the following two definitions, which provide the Language $\mathcal{E}$ with a notion analogous to that of e-consistency introduced in [3] for the Language $\mathcal{A}$.

Definition 3. *The action constants A_1 and A_2* **conflict in** *D iff D contains c-propositions of the form "A_1* `initiates` *F* `when` *C_1" and "A_2* `terminates` *F* `when` *C_2" and there is no fluent symbol F' in $\mathcal{E}$ such that both $F' \in C_1 \cup C_2$ and $\neg F' \in C_1 \cup C_2$. When $A_1 = A_2 = A$ we say that the action constant A* **self-conflicts** *in D.*

Definition 4. *D is* **fluent-independent** *iff (i) there are no time-point T and h-propositions in D of the form "A_1* `happens-at` *T" and "A_2* `happens-at` *T" such that A_1 and A_2 conflict in D, and (ii) there is no h-proposition in D of the form "A* `happens-at` *T" such that A self-conflicts in D.*

The sound and completeness results are given as follows: for any fluent-independent domain description D with only a finite number of h-propositions, the corresponding argumentation program $P_{\mathcal{E}}(D)$ is semantically equivalent to D, since there is a one to one correspondence between models of D and admissible extensions of $P_{\mathcal{E}}(D)$, such that true t-propositions in the models match *HoldsAt* literals derivable from the corresponding admissible extensions.

Clearly, a fluent-independent domain description with a finite number of h-propositions (for totally ordered time) is consistent. Then:

Theorem 1 (Soundness). *Let D be a fluent-independent domain description with a finite number of h-propositions, and let M be a model of D. Then there exists an admissible extension Δ of $P_{\mathcal{E}}(D)$ such that for any time-point T and fluent constant F:*

$$M(F,T) = true \quad iff\ \Delta \vdash HoldsAt(F,T)$$
$$M(F,T) = false\ iff\ \Delta \vdash \neg HoldsAt(F,T).$$

Theorem 2 (Completeness). *Let D be a fluent-independent domain description with a finite number of h-propositions. and let Δ be an admissible extension of $P_{\mathcal{E}}(D)$. Then, the interpretation H defined as follows is a model of D. For any time-point T and fluent constant F:*

$$H(F,T) = true \quad iff\ \Delta \vdash HoldsAt(F,T)$$
$$H(F,T) = false\ iff\ \Delta \vdash \neg HoldsAt(F,T).$$

Note that these results imply that any admissible extension Δ is *total*, i.e. is such that, for any fluent F and time-point T, $\Delta \vdash HoldsAt(F,T)$ or $\Delta \vdash \neg HoldsAt(F,T)$. Indeed, we use this property to prove the results above.

These results continue to hold when D contains t-propositions by simply considering only the admissible sets that confirm all the t-propositions. Thus, the semantic role of the t-propositions in D is like that of integrity constraints i.e. meta-level properties that must hold in any extension of the program. Note that, by expressing the requirement imposed by the t-propositions in this (meta-level) manner, the translation of the domain description D into an argumentation program does not need to take into account the t-propositions in D. Alternatively, the semantic role of the t-propositions can also be captured by extending the translation of D so that the background theory $B(D)$ includes a sentence $given(F,T)$ (resp. $given(neg(F),T)$ for every t-proposition F `holds-at` T (resp. $\neg F$ `holds-at` T) in D, and adding the following generation rules to the argument theory $\mathcal{A}_\mathcal{E}$ but not to the argument base $\mathcal{A}'_\mathcal{E}$ in $P_\mathcal{E}(D)$:

$$HoldsAt(f,t_2) \leftarrow given(f,t_1),\ t_1 \preceq t_2$$

$$\neg HoldsAt(f,t_2) \leftarrow given(neg(f),t_1),\ t_1 \preceq t_2$$

and by extending the priority ordering accordingly (as for the basic generation rules). Note that, since these additional rules are not in the argument base $\mathcal{A}'_\mathcal{E}$, they cannot be used in building an admissible extension. Indeed, allowing them as part of admissible extensions would trivially allow to confirm all the t-propositions of the domain, and this would be incorrect in cases where there is no other independent way of confirming the t-propositions.

5 Proof Theory

The translation of domain descriptions into argumentation programs allows us to employ existing computational model(s) for argumentation, such as [11] or more directly for LPwNF [4], to compute entailed t-propositions by computing the sceptical consequences ot the corresponding argumentation programs.

In this section we adapt the abstract computational framework in [11] for the Language $\mathcal{E}$. As an example, consider the domain description D_c in section 2. We can use the corresponding argumentation program $P_\mathcal{E}(D)$ to show (argue) that $\neg HoldsAt(R,7)$ (here and below R stands for *Running* and P for *Petrol*). The background theory $B(D)$ together with any set S such that

$$S \supseteq \{NG[R,7;3]\}$$

can (monotonically) derive this conclusion. S is attacked by

$$A = \{\ PG[R,7;5],\ PA[P,5]\}.$$

S can attack back A on the rule $PA[P,5]$ by bringing in the argument $NG[P,5;3]$, giving the extended set

$$S_1 = S \cup \{NG[P,5;3]\}.$$

S_1 still derives the required conclusion $\neg HoldsAt(R,7)$. Moreover, S_1 is admissible and hence $\neg HoldsAt(R,7)$ is a credulous consequence of $P_\mathcal{E}(D)$. In order to show that it is also a sceptical consequence we can show that we can not build an admissible set for its negation $HoldsAt(R,7)$. This can be done by showing that for whatever set S' deriving $HoldsAt(R, 7)$, there is at least one attack against S' (in this case given by the set S_1) that can not be attacked back by any non-contradictory extension of S'. In other words, we can show that there is no admissible argument set S' deriving $HoldsAt(R, 7)$.

The general argumentation-based computational framework developed in [11] is expressed in abstract terms via derivations of *trees*, whose nodes are sets of arguments attacking the arguments in their parent nodes, and the construction of "acceptable" trees. We develop a proof theory for $\mathcal{E}$ in a similar way in terms of derivations of trees which construct *admissible trees.*

Definition 5. *Given an argumentation program* $(B, \mathcal{A}, \mathcal{A}', <\ )$*, an* **admissible tree** *for a set of arguments* $S \subseteq \mathcal{A}'$ *is an and-tree of depth 2 where*

- *each node is a closed set of arguments;*
- *the root (node of level 0) is* S*;*
- *the root has as children (nodes of level 1) all sets of arguments* $A \subseteq \mathcal{A}$ *such that* A *is a minimal attack against* S*;*
- *every node* A *of level 1 has (exactly) one child* $D \subseteq \mathcal{A}'$ *(node of level 2) and* D *is a minimal attack against* A*;*
- *no node of level 1 is a subset of the root* S*;*
- *every node of level 2 is a subset of the root* S*.*

Nodes of level 1 play the role of *attacks* against the given set S. These attacks are counter attacked by the nodes of level 2, that play the role of *defences* for S. The last two conditions correspond to the two properties of an admissible set S that it is non-contradictory and that it defends itself under any attack. Figure 1(a) shows the structure of a generic admissible tree.

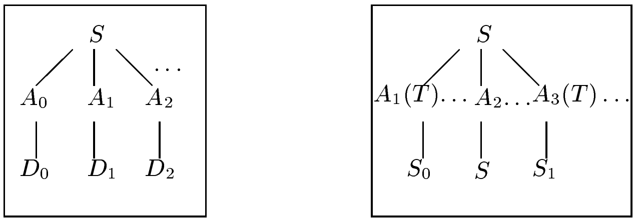

Fig. 1. (a) Generic admissible tree, and (b) Admissible tree for the D_c domain

Theorem 3. *A set of argument rules is admissible iff it is the root of an admissible tree.*

The proof theory detects whether a given literal $(\neg)HoldsAt(F, T)$ holds in some admissible extension Δ, and hence that it is a credulous consequence of the argumentation program, by constructing

1. a non-contradictory (and closed) S_0 such that $B(D) \cup S_0 \vdash (\neg)HoldsAt(F, T)$

2. extending S_0 to $S \supseteq S_0$ such that S is the root of an admissible tree.

By monotonicity of $\vdash$, $B(D) \cup S \vdash (\neg)\mathit{HoldsAt}(F,T)$. Moreover, the following lemma guarantees that $S \subseteq \Delta$ for some admissible extension $\Delta = B(D) \cup S^*$ for some $S^* \supseteq S$. Thus, again by monotonicity of $\vdash$, $\Delta \vdash (\neg)\mathit{HoldsAt}(F,T)$, and thus $(\neg)\mathit{HoldsAt}(F,T)$ is a credulous consequence of the argumentation program.

Lemma 1. *For every admissible set of argument rules S there exists an admissible extension containing S.*

In the sequel we assume that S_0 is given. The proof theory constructs an admissible tree with root $S \supseteq S_0$ incrementally, via derivations of *partial trees.* A partial tree $\mathcal{T}$ is a subtree of an admissible tree $\mathcal{T}'$ such that both the root and all nodes of level 2 of $\mathcal{T}$ are subsets of the root of $\mathcal{T}'$. In a derivation, which starts with the partial tree consisting only of the root S_0, two operations can be performed on the current partial tree: the tree can be expanded vertically by lengthening its branches, by adding defences against a group of attack nodes; or the tree can be expanded horizontally by first extending the root with new argument rules occurring in some group of defence nodes and then by adding new children of the root corresponding to the new attacks against the extended root.

In the definition below we assume a *selection strategy* identifying the "group" of nodes of the current partial tree to be handled next, and we *mark nodes* that should not be selected further. We assume that the selection strategy satisfies the following conditions: (i) all nodes in the selected group are at the same level, and (ii) if the level is 1 then there exists a single set of argument rules attacking each node in the group, whereas (iii) if the level is 2 then the sets of argument rules at all nodes of the group coincide.

Definition 6. *A* **derivation** *for a set of argument rules S_0 is a sequence of partial trees $\mathcal{T}_1, \ldots, \mathcal{T}_n, \ldots$ such that $\mathcal{T}_1$ consists only of the (unmarked) root S_0 and, given $\mathcal{T}_i$ ($i \geq 1$), if $\mathcal{N}$ is the selected group of (unmarked) nodes in $\mathcal{T}_i$ and $R(\mathcal{T}_i)$ is the root of $\mathcal{T}_i$, then $\mathcal{T}_{i+1}$ is obtained as follows:*

(α) *if the nodes in $\mathcal{N}$ are of level 1, choose a (minimal) attack D against $N - R(\mathcal{T}_i)$ for all nodes N in $\mathcal{N}$. Then, $\mathcal{T}_{i+1}$ is $\mathcal{T}_i$ where*
   1. *all the nodes in $\mathcal{N}$ are marked, and*
   2. *D is added as the (unmarked) node child of each node N in $\mathcal{N}$.*

(δ) *if $\mathcal{N}$ is a group of nodes of level 0 or 2, then $\mathcal{T}_{i+1}$ is $\mathcal{T}_i$ where*
   1. *all nodes in $\mathcal{N}$ are marked,*
   2. *the root is extended by $N - R(\mathcal{T}_i)$ where N is the set of argument rules at each node in $\mathcal{N}$, and*
   3. *if $A_1, \ldots, A_m, \ldots$ ($m \geq 0$) are all the (new) minimal attacks against $N \cup R(\mathcal{T}_i)$ such that each A_j does not attack $R(\mathcal{T}_i)$ alone then $A_1, \ldots, A_m, \ldots$ are added as additional (unmarked) nodes children of the root.*

Note that in a step of type δ if the set of arguments in the selected defence node(s) is a subset of the current root, then there will be no new attacks against the extended root and hence no extra attack nodes children of the root.

As an example, consider again the domain D_c, and suppose we want to construct a derivation for $S_0 = \{NG[R,7;3]\}$ (deriving, together with $B(D_c)$, the conclusion $\neg HoldsAt(R,7)$). One possible such derivation is as follows: $\mathcal{T}_1$ is the tree consisting uniquely of the root S_0. There are two groups of (minimal) attacks against S_0 that it is necessary to consider:

$A_1(T) = \{PP[R,7;T], PA[R,T]\}$ for any time-point T with $3 \prec T \prec 7$,
$A_2 = \{PG[R,7;5], PA[P,5]\}$.

Note that there are additional groups of (minimal) attacks, e.g.

$A_1^{'}(T,T') = \{PP[R,7;T], PP[R,T;T'], PA[R,T']\}$
for any time-points T, T' with $3 \prec T' \prec T \prec 7$,
$A_1^{''}(T,T',T'') = \{PP[R,7;T], PP[R,T;T'], PP[R,T';T''], PA[R,T'']\}$
for any time-points T, T', T'' with $3 \prec T'' \prec T' \prec T \prec 7$,
etc.
$A_2^{'}(T) = \{PG[R,7;5], PP[P,5;T], PA[P,T]\}$
for any time-point T such that $T \prec 5$,
$A_2^{''}(T,T') = \{PG[R,7;5], PP[P,5;T], PP[P,T;T'], PA[P,T']\}$
for any time-point T, T' such that $T' \prec T \prec 5$,
etc.

However, any such additional attacks, involving chains of persistence rules, can be counterattacked in the same way as the corresponding simpler attacks involving a single persistence rule, and so may be disregarded.

Then, by a step of type (δ), $\mathcal{T}_2$ is $\mathcal{T}_1$ with the root marked and all attacks $A_1(T)$ and A_2 added as children of the root.

All attacks $A_1(T)$ are counter attacked by S_0 itself, since their argument rules are non-comparable under the priority ordering. Then, selecting all attacks $A_1(T)$, a step of type (α) gives $\mathcal{T}_3$ amounting to $\mathcal{T}_2$ with S_0 added as the child of each attack node $A_1(T)$ and each $A_1(T)$ marked.

All newly added defence nodes (with S_0) can be selected next and marked, by a step of kind (δ), thus giving $\mathcal{T}_4$.

S_0 alone does not counter attack the attack node A_2 or any of the attack nodes in the group of nodes $A_2^{'}(T)$. The set $S = S_0 \cup \{NG[P,5;3]\}$ can attack each one of these nodes. Then, by a step of type (α) where A_2 and each $A_2^{'}(T)$ node are selected together, $\mathcal{T}_5$ is $\mathcal{T}_4$ with A_2 and all nodes $A_2^{'}(T)$ marked and S added as the child of A_2 and of each attack node $A_2^{'}(T)$.

Further, since $S - S_0$ is attacked by $A_3(T) = \{PP[P,T,5], PA[P,T]\}$, for any time-point T such that $3 \prec T \prec 5$, by a step of type (δ) selecting the defence node S, $\mathcal{T}_6$ is $\mathcal{T}_5$ with S marked, $S - S_0$ added to the root and all attacks $A_3(T)$ added as additional children of the root [1].

Every attack $A_3(T)$ is counter attacked by $S_1 = \{NG[P, 5; 3]\}$ (as again their rules are non-comparable) and thus by the current root. Then, by a step of type (α) selecting all attack nodes $A_3(T)$, $\mathcal{T}_7$ is $\mathcal{T}_6$ with every $A_3(T)$ marked and S_1 added as a child of every $A_3(T)$.

Finally, a step of type (δ) selecting the group of these new defence nodes gives $\mathcal{T}_8$ amounting to $\mathcal{T}_7$ with all these defence nodes marked. No more steps can be applied to $\mathcal{T}_8$ as all its nodes are marked. The computed tree obtained from the final tree $\mathcal{T}_8$ by ignoring all marks is illustrated in figure 1(b). Note that this tree is an admissible tree for S.

Definition 7. *A* **successful derivation** *from S_0 to S is a (possibly infinite) derivation $\mathcal{T}_1, \ldots, \mathcal{T}$ for S_0 such that all nodes in $\mathcal{T}$ are marked, all leaves in $\mathcal{T}$ are of even level and S is the root of $\mathcal{T}$.*

Theorem 4 (Soundness and Completeness). *If there exists a successful derivation from S_0 to S then $S_0 \subseteq S$ and S is admissible. Conversely, if there exists an admissible set S containing S_0, then there exists a successful derivation from S_0 to S', for some admissible set S' such that $S_0 \subseteq S' \subseteq S$.*

Definition 8. *A* **finitely failed derivation** *from S_0 to S is a finite derivation $\mathcal{T}_1, \ldots, \mathcal{T}_k$ for S_0 such that S is the root of $\mathcal{T}_k$, all nodes in $\mathcal{T}_k$ are marked and some leaves in $\mathcal{T}_k$ are of odd level.*

Corollary 1 (Soundness of Finite Failure). *If every derivation from S_0 is finitely failed, then there exists no set S such that $S_0 \subseteq S$ and S is admissible.*

The given proof theory can be used to compute sceptical consequences as follows:

Theorem 5. *Let D be a domain description. Let $HoldsAt(F, T)$ ($\neg HoldsAt(F, T)$ resp.) be given and S_0 be a set of argument rules such that $B(D) \cup S_0 \vdash HoldsAt(F, T)$ (resp. $B(D) \cup S_0 \vdash \neg HoldsAt(F, T)$). If there exists a successful derivation for S_0 and, for every set S'_0 of argument rules such that $B(D) \cup S'_0 \vdash \neg HoldsAt(F, T)$ (resp. $B(D) \cup S'_0 \vdash HoldsAt(F, T)$), every derivation for S'_0 finitely fails, then $HoldsAt(F, T)$ (resp. $\neg HoldsAt(F, T)$) is a sceptical consequence of $P_{\mathcal{E}}(D)$.*

The abstract proof theory forms the basis for developing concrete top-down proof procedures for query evaluation, such as the one in [4]. The proof procedures are obtained by adopting specific selection strategies and ways of computing attacks in the concrete argumentation framework. For example, [4] adopts a resolution-based computation of attacks. The selection of a group of attack or defence

[1] Again, there are additional groups of attacks against $S - S_0$, involving chains of persistence rules accounting for $Petrol$ to hold at 5. And again, these additional groups of attacks are defended against in the same way as the simpler (single persistence rule) attacks in $A_3(T)$.

nodes, in the particular argumentation framework for $\mathcal{E}$, can be guided by time constraints. In this way the computation essentially reduces to satisfaction of appropriate sets of time constraints.

Note that there are several redundancies in the attacks considered by the proof theory. For example, attack nodes whose rules are non-comparable with those in the root need not be considered as they can always be counter attacked by the root itself. Hence, in a resolution-based computation of attacks, we need only consider attacks whose top-most rule is of higher priority than that of the corresponding conflicting rule in the root. This means that in $\mathcal{E}$ case we can restrict our attention to (i) attacks via generation rules only against the root on a generation rule, and (ii) attacks via generation or persistence rules against the root on a assumption rule. In the earlier example this would mean that the groups of attacks $A_1(T)$ and $A_3(T)$ would not need to be considered.

Consider now the case where domain descriptions contain t-propositions. A simple (but naive) way to extend the proof theory is to add all t-propositions to every query. If the extended query succeeds (resp. fails) then the original query can (resp. cannot) be satisfied simultaneously with the t-propositions in the domain and hence the proof theory is still correct. To illustrate this consider a simple domain description D which contains only the t-proposition F `holds-at` 5. If we pose the query $\neg HoldsAt(F, 2)$, then the proof theory would succeed from $S_0 = \{NA[F, 2]\}$ returning $S = S_0 \cup \{NA[F, T] | T \prec 2\}$. However, when the query is extended with the t-proposition $HoldsAt(F, 5)$, the proof theory would fail. Indeed, in order to derive the t-proposition the argument set S_0 would need to contain the rule $PA[F, 5]$. This gives a new attack $A = \{NA[F, 2], NP[F, 5; 2]\}$ which can only be defended by $PA[F, 2]$. If we attempt to add this, the root would become contradictory.

We are currently investigating more efficient proof theories for domains with t-propositions which identify which t-propositions are relevant to a given query by some limited form of forward reasoning from the query.

6 Conclusions

We have shown how reasoning about actions and change as formalized by the Language $\mathcal{E}$ can be understood in argumentation terms within the framework of $LPwNF$. This re-formulation rests on the fact that, for default persistence, later information has priority over earlier conflicting information. Although in this paper we have restricted our attention to totally ordered time, we believe that analogous results hold for more general time structures, such as the branching time structure employed by the Situation Calculus and the Language $\mathcal{A}$.

An important consequence of the argumentation formulation of $\mathcal{E}$ is that it provides a *computational view* on which we can build an automated proof theory for reasoning about actions and change. The proof theory is a Logic Programming style query-oriented proof procedure. Various previous works [8, 5, 3, 2, 10] address the problem of translating various declarative specifications for reasoning about actions and change into a computation-oriented form. Whereas these translations rely on a complex use of Negation as Failure to capture default persistence, our translation is more direct and, arguably, more intuitive.

We are currently investigating ways to improve the effeciency of the proof theory proposed in this paper. In particular, we are examining the possibility of further restricting the type of attacks that need to be considered, as well as introducing a limited form of forward reasoning to allow us to consider only t-propositions relevant to the query at hand. The latter feature is particularly desirable when we want to consider $\mathcal{E}$ domains with "ramification" statements [9].

Acknowledgements

This research has been partially supported by the EC Keep-In-Touch Project "Computational Logic for Flexible Solutions to Applications". The third author has been supported by the UK EPSRC Project "Logic-based multi-agent systems".

References

1. A. Bondarenko, P. M. Dung, R. A. Kowalski, F. Toni. *An abstract, argumentation-theoretic framework for default reasoning.* AI 93(1-2) 63–101 (1997).
2. S.-E. Bornscheuer and M. Thielscher. *Explicit and Implicit Indeterminism: Reasoning about uncertain and contradictory specifications of dynamic systems.* JLP 31(1-3) 119–155, 1997.
3. M. Denecker and D. De Schreye. *Representing Incomplete Knowledge in Abductive Logic Programming.* ILPS'93, MIT Press.
4. Y. Dimopoulos and A.C. Kakas. Logic Programming without Negation as Failure. ILPS'95, 369–383, MIT Press.
5. P. M. Dung. *Representing Actions in Logic Programming and its Applications in Database Updates.* ICLP'93, 222–238, MIT Press.
6. P. M. Dung. *The acceptability of arguments and its fundamental role in non-monotonic reasoning and logic programming and n-person game.* AI 77 321–357, 1995.
7. M. Gelfond and V. Lifschitz. *Representing Actions in Extended Logic Programming.* JICSLP'92, 560, MIT Press.
8. M. Gelfond and V. Lifschitz. *Representing Action and Change by Logic Programs.* JLP, 17 (2,3,4) 301–322, 1993.
9. A.C. Kakas and R.S. Miller. *Reasoning about Actions, Narratives and Ramifications.* Journal of Electronic Transactions on Artificial Intelligence 1(4), 1997.
10. A.C. Kakas and R.S. Miller. *A Simple Declarative Language for Describing Narratives with Actions.* JLP 31(1–3) (Special Issue on Reasoning about Action and Change) 157–200, 1997.
11. A. C. Kakas and F. Toni. *Computing argumentation in logic programming.* To appear in JLC, Oxford University Press, 1999.
12. R.A. Kowalski and M. Sergot. *A Logic-Based Calculus of Events.* New Generation Computing 4, 267, 1986.
13. M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Commonsense Law of Inertia*, MIT Press, 1997.