$\mathcal{E}$-RES - A System for Reasoning about Actions, Events and Observations

Antonis Kakas
University of Cyprus,
Nicosia, CYPRUS
antonis@cs.ucy.ac.cy

Rob Miller
University College London,
London, U.K.
rsm@ucl.ac.uk

Francesca Toni
Imperial College,
London, U.K.
ft@doc.ic.ac.uk

Abstract

$\mathcal{E}$-RES is a system that implements the Language $\mathcal{E}$, a logic for reasoning about narratives of action occurrences and observations. $\mathcal{E}$'s semantics is model-theoretic, but this implementation is based on a sound and complete reformulation of $\mathcal{E}$ in terms of argumentation, and uses general computational techniques of argumentation frameworks. The system derives sceptical non-monotonic consequences of a given reformulated theory which exactly correspond to consequences entailed by $\mathcal{E}$'s model-theory. The computation relies on a complimentary ability of the system to derive credulous non-monotonic consequences together with a set of supporting assumptions which is sufficient for the (credulous) conclusion to hold. $\mathcal{E}$-RES allows theories to contain general action laws, statements about action occurrences, observations and statements of ramifications (or universal laws). It is able to derive consequences both forward and backward in time. This paper gives a short overview of the theoretical basis of $\mathcal{E}$-RES and illustrates its use on a variety of examples. Currently, $\mathcal{E}$-RES is being extended so that the system can be used for planning.

General Information

$\mathcal{E}$-RES is a system for modeling and reasoning about dynamic systems. Specifically, it implements the Language $\mathcal{E}$ (Kakas & Miller 1997b; 1997a), a specialist logic for reasoning about narratives of action occurrences and observations. $\mathcal{E}$-RES is implemented in SICStus Prolog and runs on any platform for which SICStus is supported (e.g. Windows, Linux, UNIX, Mac). The program is about 300 lines long (a URL is given at the end of the paper). The semantics of $\mathcal{E}$ is model-theoretic, but this implementation is based on a sound and complete reformulation of $\mathcal{E}$ in terms of argumentation, and uses general computational techniques of argumentation frameworks. To describe the operation and utility of $\mathcal{E}$-RES, it is necessary to first review the Language $\mathcal{E}$.

The Language $\mathcal{E}$

Like many logics, the Language $\mathcal{E}$ is really a collection of languages, since the particular vocabulary employed depends on the domain being modeled. The domain-dependent vocabulary always consists of 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 $\neg F$. In the current implementation of $\mathcal{E}$-RES the only time structure that is supported is that of the natural numbers, so we restrict our attention here to domains of this type, using the standard ordering relation $\leq$ in all examples.

Domain descriptions in the Language $\mathcal{E}$ are collections of four kinds of statements (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"), either of the form

  A `initiates` F `when` C

  or of the form

  A `terminates` F `when` C

- *r-propositions* ("r" for "ramification"), of the form

  L `whenever` C.

The precise semantics of $\mathcal{E}$ is described in (Kakas & Miller 1997b) and (Kakas & Miller 1997a). T-propositions are used to record observations that particular fluents hold or do not hold at particular time-points, and h-propositions are used to state that particular actions occur at particular time-points. C-propositions state general "action laws" – the intended meaning of "A `initiates` F `when` C" is "C is a minimally sufficient set of conditions for an occurrence of A to have an initiating effect on F". (When C is empty the proposition is stated simply as "A `initiates` F".) R-propositions serve a dual role in that they describe both static constraints between fluents and ways in which fluents may be indirectly affected by action occurrences. The intended meaning of "L `whenever` C" is "at every time-point that C holds, L holds, and hence every action occurrence that brings about C also brings about L".

$\mathcal{E}$'s semantics is perhaps best understood by examples, and so several are given in the next sub-section. The key features of the semantics are as follows.

- Models are simply mappings of fluent/time-point pairs to $\{true, false\}$ which satisfy various properties relating to the propositions in the domain.
- The semantics describes *entailment* ($\models$) of extra t-propositions (but not h-, c- or r-propositions) from domain descriptions.
- $\mathcal{E}$ is monotonic as regards addition of t-propositions to domain descriptions, but non-monotonic (in order to eliminate the frame problem) as regards addition of h-, c- and r-propositions. The semantics encapsulates the assumptions that (i) no actions occur other than those explicitly represented by h-propositions, (ii) actions have no direct effects other than those explicitly described by c-propositions, and (iii) actions have no indirect effects other than those that can be explained by "chains" of r-propositions in the domain description. (Technically, these "chains" are defined using the notion of a least fixed point.)
- The semantics ensures that fluents have a *default persistence*. In each model, fluents change truth values only at time-points (called *initiation points* and *termination points*) where an h-proposition and a c-proposition (whose preconditions are satisfied in the model) combine to cause a change, or where an h-proposition, a c-proposition and a "chain" of r-propositions all combine to give an indirect or knock-on effect. All effects (direct and indirect) of an action occurrence are instantaneous, i.e. all changes are apparent immediately after the occurrence.
- As well as indicating how the effects of action occurrences instantaneously propagate, r-propositions place constraints on which combinations of t-propositions referring to the same time-point are allowable. In this latter respect they behave as ordinary classical implications.

Example Language $\mathcal{E}$ Domain Descriptions

Each of the following domain descriptions illustrates how $\mathcal{E}$ supports particular modes of reasoning about the effects of actions. These domain descriptions are used in subsequent sections of the paper to illustrate the functionality of the $\mathcal{E}$-RES system.

Example 1 *(Vaccinations)*
This example concerns vaccinations against a particular disease. Vaccine A only provides protection for people with blood type O, and vaccine B only works on people with blood type other than O. Fred's blood type is not known, so he is injected with vaccine A at 2 o'clock and vaccine B at 3 o'clock. To describe this scenario we need a vocabulary of two action constants *InjectA* and *InjectB*, and two fluent constants *Protected* and *TypeO*. The domain description D_v consists of two c-propositions and two h-propositions:

$$\begin{array}{lr} \mathit{InjectA}\ \texttt{initiates}\ \mathit{Protected}\ \texttt{when}\ \{\mathit{TypeO}\} & (D_v1) \\ \mathit{InjectB}\ \texttt{initiates}\ \mathit{Protected}\ \texttt{when}\ \{\neg\mathit{TypeO}\} & (D_v2) \\ \mathit{InjectA}\ \texttt{happens-at}\ 2 & (D_v3) \\ \mathit{InjectB}\ \texttt{happens-at}\ 3 & (D_v4) \end{array}$$

If we now consider some time later than 3 o'clock, say 6 o'clock, we can see intuitively that Fred should be protected, and indeed it is the case that

$$D_v \models \mathit{Protected}\ \texttt{holds-at}\ 6.$$

This is because there are two classes of models for this domain. In models of the first type, *TypeO* holds for all time-points, so that (D_v1) and (D_v3) combine to form an initiation point for *Protected* at 2. In models of the second type, *TypeO* does not hold for any time-point, and so (D_v2) and (D_v4) combine to form an initiation point for *Protected* at 3. In either type of model, *Protected* then persists from its initiation point up to time-point 6, since the fluent has no intervening termination points to override its initiation. Note, however, that there are no default assumptions directly attached to t-propositions, so that for any time $T \leq 3$ it is neither the case that D_v entails *Protected* `holds-at` T nor the case that D_v entails $\neg$*Protected* `holds-at` T. □

Example 2 *(Photographs)*
This example shows that the Language $\mathcal{E}$ can be used to infer information about what conditions hold at the time of an action occurrence, given other information about what held at times before and afterwards. It concerns taking a photograph. There is a single action *Take*, and two fluents *Picture* (representing that a photograph has been successfully taken) and *Loaded* (representing that the camera is loaded with film). Suppose that the domain description D_p consists of a single c-proposition, a single h-proposition and two t-propositions:

$$\begin{array}{lr} \mathit{Take}\ \texttt{initiates}\ \mathit{Picture}\ \texttt{when}\ \{\mathit{Loaded}\} & (D_p1) \\ \mathit{Take}\ \texttt{happens-at}\ 2 & (D_p2) \\ \neg\mathit{Picture}\ \texttt{holds-at}\ 1 & (D_p3) \\ \mathit{Picture}\ \texttt{holds-at}\ 3 & (D_p4) \end{array}$$

Since a change occurs in the truth value of *Picture* between 1 and 3, in all models an action must occur at some time-point between 1 and 3 whose initiating conditions for the property *Picture* are satisfied at that point. The only candidate is the *Take* occurrence at 2, whose condition for initiating *Picture* is *Loaded*. Hence

$$D_p \models \mathit{Loaded}\ \texttt{holds-at}\ 2.$$

Indeed, by the persistence of *Loaded* (in the absence of possible initiation or termination points for this fluent), for any time T, $D_p \models$ *Loaded* `holds-at` T. □

Example 3 *(Cars)*
This example illustrates the use of r-propositions, and shows how the effects of later action occurrences override the effects of earlier action occurrences. It concerns a car engine. The fluent *Running* represents that

the engine is running, the fluent *Petrol* represents that there is petrol (gas) in the tank, the action *TurnOn* represents the action of turning on the engine, the action *TurnOff* represents the action of turning off the engine, and the action *Empty* represents the event of the tank becoming empty (or the action of someone emptying the tank). We describe a narrative where the engine is initially running, is turned off at time 2, is turned back on at time 5, and runs out of petrol at time 8. We also want to state the general constraint that the engine cannot run without petrol. The domain description D_c consists of:

TurnOn `initiates` *Running* `when` $\{Petrol\}$	(D_c1)
TurnOff `terminates` *Running*	(D_c2)
Empty `terminates` *Petrol*	(D_c3)
$\neg$*Running* `whenever` $\{\neg Petrol\}$	(D_c4)
Running `holds-at` 1	(D_c5)
TurnOff `happens-at` 2	(D_c6)
TurnOn `happens-at` 5	(D_c7)
Empty `happens-at` 8	(D_c8)

The Language $\mathcal{E}$ supports the following conclusions concerning the fluents *Running* and *Petrol*:
(i) For $T \leq 2$, $D_c \models$ *Running* `holds-at` T. This is because of (D_c5), and because there are no relevant action occurrences before time 2 to override *Running*'s default persistence.
(ii) For $T \leq 8$, $D_c \models$ *Petrol* `holds-at` T. This is because we obtain *Petrol* `holds-at` 1 directly from (D_c4) and (D_c5), and because there are no relevant action occurrences before time 8 to override *Petrol*'s default persistence. Note that in this case (D_c4) has been used (in the contrapositive) in its capacity as a static constraint at time 1.
(iii) For $2 < T \leq 5$, $D_c \models \neg$*Running* `holds-at` T. This is because (D_c2) and (D_c6) combine to form a termination point for *Running* at 2 (in all models).
(iv) For $5 < T \leq 8$, $D_c \models$ *Running* `holds-at` T. This is because (D_c1) and (D_c7) combine to form an initiation point for *Running* at 5, and this overrides the earlier termination point (for all times greater than 5).
(v) For $T > 8$, $D_c \models \neg$*Petrol* `holds-at` T. This is because (D_c3) and (D_c8) combine to form a termination point for *Petrol* at 8.
(vi) For $T > 8$, $D_c \models \neg$*Running* `holds-at` T. This is because (D_c3), (D_c4) and (D_c8) combine to form a termination point for *Running* at 8 which overrides the earlier initiation point.

Note that $\mathcal{E}$ does not allow r-propositions to be used in the contrapositive to generate extra initiation or termination points. For example, if we were to add the two propositions

JumpStart `initiates` *Running*	(D_c9)
JumpStart `happens-at` 11	(D_c10)

to the domain description, we would have inconsistency. The combination of (D_c9) and (D_c10) would give an initiation point for *Running* at time 11, so that at subsequent times *Running* would be true. However, (v) above shows that for such times *Petrol* is false, and this contradicts (D_c4) in its capacity as a static constraint. (D_c4) cannot be used in the contrapositive to "fix" this by generating a termination point for *Petrol* from the termination point for *Running*. $\square$

Description of the System

The system relies upon a reformulation of the Language $\mathcal{E}$ into argumentation as described in (Kakas, Miller, & Toni 1999).

Argumentation Formulation of $\mathcal{E}$

A domain description D without t-propositions and without r-propositions is translated into an *argumentation program* $P_{\mathcal{E}}(D) = (B(D), \mathcal{A}_{\mathcal{E}}, \mathcal{A}'_{\mathcal{E}}, <_{\mathcal{E}})$, where $B(D)$ is the *background theory*, $\mathcal{A}_{\mathcal{E}}$ is the *argumentation theory*, i.e. a set of *argument rules*, $\mathcal{A}'_{\mathcal{E}} \subseteq \mathcal{A}_{\mathcal{E}}$ is the *argument base*, and $<_{\mathcal{E}}$ is a *priority relation* over the (ground instances of the) argument rules. Intuitively, the sentences in the monotonic background theory can be seen as non-defeasible argument rules which must belong to any non-monotonic extension of the theory. These extensions are given by the *admissible* subsets of $\mathcal{A}'_{\mathcal{E}}$, namely subsets that are both *non-self-attacking* and *(counter)attack* any set of argument rules *attacking* them. Whereas an admissible set can consist only of argument rules in the argument base, *attacks* against an admissible set are allowed to be subsets of the larger argument theory. The exact definition of an attack, which is dependent on the priority relation $<_{\mathcal{E}}$ and the derivation of complimentary literals, is given in (Kakas, Miller, & Toni 1999).

Both $B(D)$ and $\mathcal{A}_{\mathcal{E}}$ use the predicates *HappensAt*, *HoldsAt*, *Initiation* and *Termination*. $B(D)$ is a set of Horn clauses corresponding to the h- and c-propositions in D defining the above predicates expect *HoldsAt*. $\mathcal{A}_{\mathcal{E}}$ is a domain independent set of *generation*, *persistence* and *assumption* rules for *HoldsAt*. For example, a generation rule is given by $HoldsAt(f, t_2) \leftarrow Initiation(f, t_1)$, $t_1 \prec t_2$, where f is any fluent and t_1, t_2 are any two time points. The relation $<_{\mathcal{E}}$ is such that the effects of later events take priority over the effects of earlier ones (see (Kakas, Miller, & Toni 1999)). Given the translation, results in (Kakas, Miller, & Toni 1999) show that there is a one-to-one correspondence between (i) models of D and maximal admissible sets of arguments of $P_{\mathcal{E}}(D)$, and (ii) t-propositions entailed by D and sceptical non-monotonic consequences of the form $(\neg)HoldsAt(F, T)$ of $P_{\mathcal{E}}(D)$, where a given literal σ is a *sceptical* (resp. *credulous*) non-monotonic consequence of an argumentation program iff $B(D) \cup \Delta \vdash \sigma$ for *all* (resp. *some*) maximal admissible extension(s) Δ of the program.

This method can be applied directly for conjunctions of literals rather than individual literals. Hence the above techniques can be straightforwardly applied to domains with t-propositions simply by adding all t-propositions in the domain to the conjunctions of literals whose entailment we want to check. Similarly, the above techniques can be directly adapted for domains with r-propositions by conjoining to the given literals the conclusion of ramification statements that are "fired".

Proof theory

Given the translation of an $\mathcal{E}$ domain description D into $P_{\mathcal{E}}(D)$, a proof theory can be developed directly (Kakas, Miller, & Toni 1999), in terms of *derivations of trees*, whose nodes are sets of arguments in $\mathcal{A}_{\mathcal{E}}$ attacking the arguments in their parent nodes. Suppose that we wish to demonstrate that a t-proposition $(\neg)F$ `holds-at` T is entailed by D. Let S_0 be a (non-self-attacking) set of arguments in $\mathcal{A}'_{\mathcal{E}}$ such that $B(D) \cup S_0 \vdash (\neg)HoldsAt(F,T)$ (S_0 can be easily built by backward reasoning). Two kinds of derivations are defined:

- *successful derivations*, building, from a tree consisting only of the root S_0, a tree whose root S is an admissible subset of $\mathcal{A}'_{\mathcal{E}}$ such that $S \supseteq S_0$, and
- *finitely failed derivations*, guaranteeing the absence of any admissible set of arguments containing S_0.

Hence the given t-proposition is entailed by D if there exists a successful derivation with initial tree consisting only of the root S_0 and, for every set S'_0 of argument rules in $\mathcal{A}'_{\mathcal{E}}$ such that $B(D) \cup S'_0$ derives (via $\vdash$) the complement of the (literal translation of the) t-proposition, every derivation for S'_0 is finitely failed.

Implementation

The system is an implementation of the proof theory presented in (Kakas, Miller, & Toni 1999), but it does not rely explicitly on tree-derivations. Instead, it implicitly manipulates trees via their frontiers, in a way similar to the proof procedure for computing partial stable models of logic programs in (Eshghi & Kowalski 1989; Kakas & Mancarella 1990). (See also (Kakas & Toni 1999) for a general discussion of this technique.)

$\mathcal{E}$-RES defines the Prolog predicates `sceptical/1` and `credulous/1`. For some given `Goal` which is a list of literals, with each literal either of the form `holds(f,t)` or `neg(holds(f,t))` (where the Prolog constant symbols `f` and `t` represent a ground fluent constant F and time point T respectively),

- if `sceptical(Goal)` succeeds then each literal in `Goal` is a sceptical non-monotonic consequence of the domain
- if `sceptical(Goal)` finitely fails then some literal in `Goal` is not a sceptical non-monotonic consequence of the domain
- if `credulous(Goal)` succeeds then each literal in `Goal` is a credulous non-monotonic consequence of the domain
- if `credulous(Goal)` finitely fails then some literal in `Goal` is not a credulous non-monotonic consequence of the domain.

The implementation also defines the Prolog predicate `credulous/2`. This is such that for some given `Goal`,

- if `credulous(Goal,X)` succeeds then each literal in `Goal` is a credulous non-monotonic consequence of the domain, and the set of arguments in `X` provides the corresponding admissible extension of the argumentation program translation of the domain.

Hence `credulous(Goal,X)` can be used to provide an *explanation* `X` for the goal `Goal`.

Domain descriptions in $\mathcal{E}$ may sometimes be described using meta-level quantification, and $\mathcal{E}$-RES can support a restricted form of non-propositional programs where all c-propositions are "strongly range-restricted", i.e. only of the form $A(\overline{Y})$ `initiates` $F(\overline{X})$ `when` $C(\overline{Z})$ where $\overline{Z} \subseteq \overline{X} \cup \overline{Y}$. (We assume the usual convention of universal quantification over the whole proposition.) However, all h-propositions and queries must be ground. Ramifications could also be specified with variables provided that they all have the general form $L(\overline{Z})$ `whenever` $C(\overline{Z_1})$ where $\overline{Z_1} \subseteq \overline{Z}$. However, in the present implementation such statements need to be ground before they can be handled by the system.

Applying the System

Methodology

The system relies upon the formulation of problems as domains in the Language $\mathcal{E}$, and a simple and straightforward translation of these $\mathcal{E}$-domains into their logic-programming based counterparts, which are directly manipulated by the system. At the time of writing this report, the translation needs to be performed by hand by the user. However, the problem of automating this translation presents no conceptual difficulties, and is scheduled to be implemented in the near future. As an illustration, consider Example 3. Its translation is:

```
initiation(running,T):-
   happens(turnOn,T), holds(petrol,T), true.
termination(running,T):-
   happens(turnOff,T), true.
termination(petrol,T):-
   happens(empty,T), true.
ram(neg(holds(running,T))):-
   neg(holds(petrol,T)).
tprop(holds(running,1)).
happens(turnOff,2).
happens(turnOn,5).
happens(empty,8).
```

Specifics

The system relies upon a logic-based representation of concrete domains. The system has been developed systematically from its specification given by the model-theoretic semantics, and this guarantees its correctness.

The system performs the kind of reasoning which forms the basis of a number of applications in computer science and artificial intelligence, such as simulation, fault diagnosis, planning and cognitive robotics. We are currently studying extensions of the system that can be used directly to perform planning in domains that are partially unknown (Kakas, Miller, & Toni 2000).

Users and Usability

The use of $\mathcal{E}$-RES requires knowledge of the Language $\mathcal{E}$, which (like the Language $\mathcal{A}$ (Gelfond & Lifschitz 1993)) has been designed as a high-level specification tool, in $\mathcal{E}$'s case for modeling dynamic systems as narratives of action occurrences and observations, where actions can have both direct and indirect effects. As mentioned above, $\mathcal{E}$-RES is at an early stage of development, but we aim to soon have a user interface that will allow domain descriptions to be described directly in $\mathcal{E}$'s syntax.

Evaluating the System

The $\mathcal{E}$-RES system is an initial prototype. The prototype has been evaluated in two distinct ways. First, theoretical results have been developed (documented in (Kakas, Miller, & Toni 1999)) which verify that the system meets its specification, i.e. that it faithfully captures the entailment relation of the Language $\mathcal{E}$. Second, the system has been evaluated by testing it on a suite of examples that involve different modes of reasoning about actions and change. These examples, which include those given above, provide "proof-of-principle" evidence for the Language $\mathcal{E}$ (and the argumentation approach taken in providing a computational counterpart to it) as a suitable framework for reasoning about actions and change.

$\mathcal{E}$-RES correctly computes all the t-propositions entailed by Examples 1, 2 and 3. This involves reasoning with incomplete information, reasoning from effects to causes, reasoning backwards and forwards in time, reasoning about alternative causes of effects, reasoning about indirect effects, and combining these forms of reasoning. In the remainder of this section we consider in detail how $\mathcal{E}$-RES processes a small selection of the queries that can be associated with these example domains.

Testing with Example 1

Example 1 can be used to test how $\mathcal{E}$-RES deals with incomplete information about fluents, and how it is able to reason with alternatives. As explained previously, up until time 3 the truth value of *Protected* is unknown. In other words, for times less than or equal to 3, the literals *Protected* and $\neg$*Protected* both hold credulously, but neither holds sceptically. Reflecting this, for all $t \leq 3$, $\mathcal{E}$-RES succeeds on

```
credulous([holds(protected,t)])
credulous([neg(holds(protected,t))])
```

but fails on

```
sceptical([holds(protected,t)])
sceptical([neg(holds(protected,t))])
```

After time 3, however, the fluent *Protected* holds sceptically and so for all $t < 3$ the system correctly succeeds on

```
sceptical([holds(protected,t)])
```

and fails on

```
sceptical([neg(holds(protected,t))]).
```

Testing with Example 2

Example 2 can be used to test how $\mathcal{E}$-RES can reason from effects to causes, and how it is able to reason both forwards and backwards in time. The observed value of *Picture* at time 3 is explained by the fact that at the time 2, when a *Take* action occurred, *Loaded* held (there is no alternative way to explain this in the given domain). Hence the system reasons both backwards and forwards in time so that

```
sceptical([holds(loaded,2)])
```

succeeds. Furthermore, by persistence,

```
sceptical([holds(loaded,t)])
```

also succeeds for any time `t`.

Note however that if we remove (D_p3) from the domain, then *Loaded* holds only credulously at any time `t`, and hence the system fails on

```
sceptical([holds(loaded,t)])
```

but succeeds on

```
credulous([holds(loaded,t)]).
```

If D_p is augmented with the c-proposition

Take `initiates` *Picture* `when` $\{Digital\}$ (D_p5)

then *Loaded* no longer holds sceptically at any time, since there is now an alternative assumption to explain the observation given by (D_p4), namely that *Digital* holds at 2. Thus, for any time `t` the system fails on

```
sceptical([holds(loaded,t)])
sceptical([holds(digital,t)])
```

but succeeds on

```
credulous([holds(loaded,t)])
credulous([holds(digital,t)]).
```

If we pose the query

```
credulous([holds(picture,3)],X).
```

then we will get the two explanations for this observation in terms of the possible generation rules for *Picture* and assumptions on corresponding fluents in their preconditions. These will be given by the system as:

```
X = [rule(gen,picture,3,2),rule(ass,loaded,2)],
X = [rule(gen,picture,3,2),rule(ass,digital,2)].
```

Testing with Example 3

Example 3 can be used to test how $\mathcal{E}$-RES can reason with ramification statements (r-propositions) and how it can reason with a series of action occurrences where later occurrences override the effects of earlier ones. As required, $\mathcal{E}$-RES succeeds on each of the following queries:

```
sceptical([holds(running,0)])
sceptical([neg(holds(running,3))])
sceptical([holds(running,6)])
sceptical([neg(holds(running,10))])
sceptical([holds(petrol,0)])
sceptical([holds(petrol,3)])
sceptical([holds(petrol,6)])
sceptical([neg(holds(petrol,10))])
```

and fails on each converse query.

A more complex example (a variation of Example 1) that reasons with alternatives from observations and ramifications is given as follows. Consider the domain description D_i given by:

Expose `initiates` *Infected* `when` $\{TypeA\}$	(D_i1)
Expose `happens-at` 3	(D_i2)
$\neg$*Infected* `holds-at` 1	(D_i3)
Infected `holds-at` 6	(D_i4)
Expose `initiates` *Infected* `when` $\{TypeB\}$	(D_i5)
InjectA `initiates` *Danger* `when` $\{TypeA\}$	(D_i6)
InjectA `initiates` *Danger* `when` $\{TypeB\}$	(D_i7)
InjectA `happens-at` 4	(D_i8)
Allergic `whenever` $\{TypeA, Infected\}$	(D_i9)
Allergic `whenever` $\{TypeB\}$	(D_i10)

From the observation at time 6 the system is able to reason (both backwards and forwards in time) to prove that under any of the two possible alternatives *Danger* holds after time 4. Thus, for any time `t` after 4 the system succeeds on

`sceptical([holds(danger,t)])`.

Similarly, the system succeeds on

`sceptical([holds(allergic,t)])`,

for any time `t` after 3.

Conclusions and Future Work

We have described $\mathcal{E}$-RES, a Language $\mathcal{E}$ based system for reasoning about narratives involving actions, change, observations and indirect effects via ramifications. The functionality of $\mathcal{E}$-RES has been demonstrated both via theoretical results and by testing with benchmark problems. We have shown that the system is versatile enough to handle a variety of reasoning tasks in simple domains. In particular, $\mathcal{E}$-RES can correctly reason with incomplete information, both forwards and backwards in time, from causes to effects and from effects to causes, and about the indirect effects of action occurrences.

The system still needs to be tested with very large problems, and possibly developed further to cope with the challenges that these pose. In particular, the current handling of t-propositions and ramification statements will probably be unsatisfactory for very large domains, and techniques will need to be devised to select and reason with only the t- and r-propositions that are *relevant* the the goal being asked.

Work is currently underway to extend the $\mathcal{E}$-RES system so that it can carry out *planning*. The implementation will correspond to the $\mathcal{E}$-Planner described in (Kakas, Miller, & Toni 2000). In our setting, planning amounts to finding a suitable set of h-propositions which, when added to the given domain description, allow the entailment of a desired goal. The $\mathcal{E}$-Planner is especially suitable for planning under incomplete information, e.g. when we do not have full knowledge of the initial state of the problem, and the missing information cannot be "filled in" by performing additional actions, (either because no actions exist which can affect the missing information, or because there is no time to perform such actions). The planner needs to be able to reason correctly despite this incompleteness, and construct plans (when such plans exist) in cases where missing information is not necessary for achieving the desired goal. For instance, in Example 1, if (D_v3) and (D_v4) are missing, and the goal to achieve is *Protected* `holds-at` 4, then the $\mathcal{E}$-Planner generates *InjectA* `happens-at` T_1 and *InjectB* `happens-at` T_2, with $T_1, T_2 < 4$.

Obtaining the System

Both $\mathcal{E}$-RES and codings of example test domains are available from the Language $\mathcal{E}$ and $\mathcal{E}$-RES website at `http://www.ucl.ac.uk/~uczcrsm/LanguageE/`.

References

Eshghi, K., and Kowalski, R. 1989. Abduction compared with negation as failure. In *ICLP'89, MIT Press.*

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. In *JLP, 17 (2,3,4) 301–322.*

Kakas, A., and Mancarella, P. 1990. On the relation between truth maintenance and abduction. In *Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence.*

Kakas, A., and Miller, R. 1997a. Reasoning about actions, narratives and ramifications. In *J. of Electronic Transactions on A.I. 1(4), Linkoping University E. Press, http://www.ep.liu.se/ea/cis/1997/012/.*

Kakas, A., and Miller, R. 1997b. A simple declarative language for describing narratives with actions. In *JLP 31(1–3), 157–200.*

Kakas, A., and Toni, F. 1999. Computing argumentation in logic programming. In *JLC 9(4), 515–562, O.U.P.*

Kakas, A.; Miller, R.; and Toni, F. 1999. An argumentation framework for reasoning about actions and change. In *LPNMR'99, 78–91, Springer Verlag.*

Kakas, A.; Miller, R.; and Toni, F. 2000. Planning with incomplete information. In *NMR'00, Session on Representing Actions and Planning.*