External Transaction Logic with Automatic Compensations

Ana Sofia Gomes and José Júlio Alferes*

CENTRIA - Dep. de Informática, Faculdade Ciências e Tecnologias
Universidade Nova de Lisboa

Abstract External Transaction Logic ($\mathcal{ETR}$) is an extension of logic programming useful to reason about the behavior of agents that have to operate in a two-fold environment in a transactional way: an internal knowledge base defining the agent's internal knowledge and rules of behavior, and an external world where it executes actions and interact with other entities. Actions performed by the agent in the external world may fail, e.g. because their preconditions are not met or because they violate some norm of the external environment. The failure to execute some action must lead, in the internal knowledge base, to its complete rollback, following the standard ACID transaction model. Since it is impossible to rollback external actions performed in the outside world, external consistency must be achieved by executing compensating operations (or repairs) that revert the effects of the initial executed actions.
In $\mathcal{ETR}$, repairs are stated explicitly in the program. With it, every performed external action is explicitly associated with its corresponding compensation or repair. Such user defined repairs provide no guarantee to revert the effects of the original action. In this paper we define how $\mathcal{ETR}$ can be extended to automatically calculate compensations in case of failure. For this, we start by explaining how the semantics of Action Languages can be used to model the external domain of $\mathcal{ETR}$, and how we can use it to reason about the reversals of actions.

1 Introduction and Motivation

Intelligent agents in a multi-agent setting must work and reason over a two-fold environment: an external environment, representing the outside world where the agent acts, and which may include other agents; and an internal environment comprising the information about the agent's rules of behavior, preferences about the outside world, its knowledge and beliefs, intentions, goals, etc. An agent may act on the external environment (external actions), but also on the internal environment (internal actions). Examples of the latter are insertions and deletions in the agent's own knowledge base, updates on its rules of behavior or preferences.

When performing actions, agents must take into account what to do upon an action failure. This is especially relevant inasmuch as the agent has no control over the behavior of the external world. External actions may fail because their preconditions are not met at the time of intended execution or, in norm regimentation, because the execution

* The first author was supported by FCT grant SFRH/BD/64038/2009. The work was partially supported by project ERRO (PTDC/EIA-CCO/121823/2010).

of the action would cause the violation of some norm (e.g. as allowed by 2OPL [7]), or even by some totally unknown reason to the agent.

The failure of an action should trigger some repair plan. This is especially important when the action is part of a plan, in which case it may be necessary to undo the effects of previous actions that have succeeded. When the action to undo is an internal action, the undo should be trivial. In fact, since the agent has full control over its own internal environment, actions and updates can be made to follow the standard ACID[1] properties of transactions in databases and, as such, the effects made by internal actions are completely discarded. However, since an agent has no control over the external environment, such transactional properties cannot, in general, be guaranteed when undoing external actions.

Example 1 (Medical Diagnosis). Consider an agent in a medical scenario dealing with a two-fold Knowledge Base (KB). An internal KB defining e.g. treatment specifications and history of successful treatments of patients, and an external world where the agent interacts with patients and executes actions. When a patient arrives with a series of symptoms, the agent needs to reason about what should be the treatment applicable to the given patient, but also execute this treatment by possibly giving some medication. In case a patient shows a negative reaction to the medication, thus failing the action of treating the patient, something must be done to counter the possible side-effects of the previous treatment. Moreover, actions and updates in the internal KB need to be executed transactionally, so as to guarantee that the history of successful treatments is not updated with the medication that showed negative effects, which thereby could lead the agent to apply the same treatment again.

In this example, "what to do to counter the side-effects of a previous unsuccessful treatment" is a typical case of a repair plan, something that can be found in agent languages such as 2APL [6] (plan-repair rules) and 3APL [14] (plan-revision rules). In these languages, it is possible to state for each plan, which alternative plan should be performed in case something fails. E.g., in the example, one could say that if some treatment fails, then one should give the patient some alternative medication to counter the effects of the first medication given in the failed treatment.

In this example it is reasonable to assume that the plan (treatment) can only be repaired if the agent's specification explicitly states what are the actions to execute for each failed treatment. In other words, it is reasonable to assume that whoever programmed the agent explicitly included in the program the repair plans for each possible failure. This is e.g. the case in 2APL, where plan-repair rules explicitly include the actions to execute when a given action or plan fails.

However, if one has some knowledge of the external environment, it should be possible for the agent to automatically infer the repair plan in a given failure situation, thus saving the programmer from that task, and from having to anticipate all possible relevant failures.

Example 2 (Supermarket Robot). Imagine a scenario of a robot in a supermarket that has the task to fill up the supermarket's shelves with products. In its internal KB, the

[1] where ACID, as usual, stands for Atomicity, Consistency, Isolation and Durability.

agent keeps information about the products' stocks and prices, but also rules on how products should be placed (e.g. "premium" products should be placed in the shelves with higher visibility). Externally, the agent needs to perform the task of putting products in a given shelf, something that can be encoded in a blocks-world manner. In this case, when some action fails in the context of a plan for e.g. arranging the products in some manner, the agent, knowing the effects of the actions in the outside world, should be able to infer what actions to perform in order to restore the external environment to some consistent configuration, upon which some other alternative plan can be started.

Several solutions exist in the literature addressing the problem of reversing actions. E.g. [8] introduces a solution based on Action Languages [9] that reasons about what actions may revert the effects of other actions. For that they define the notions of reverse action, reverse plan and conditional reversals that undo the effects of a given (set of) action(s). These notions may allow the automatic inference of plan repairs.

In this paper we propose a logic programming like language that tackles all the previously mentioned issues. In particular, the language operates over two-fold KBs, with both an internal and an internal environment; it allows for performing actions both in the internal and the external environment; it deals with failure of actions, having a transactional behavior in the actions performed in the internal environment, and executing repair plans in the external environment; it allows to automatically infer repair plans when there is knowledge about the effects of actions.

Our solution is based on External Transaction Logic ($\mathcal{ETR}$) [12,13], an extension of Transaction Logic ($\mathcal{TR}$) [3] for dealing with the execution of external actions. Here, if a transaction fails after external actions are executed in the environment, then external consistency is achieved by issuing compensating actions that revert the effects of the initial executed actions. $\mathcal{ETR}$, as its ancestor $\mathcal{TR}$, is a very general language, that relies on the existence of oracles for querying and updating an internal KB and, in the case of $\mathcal{ETR}$, also for dealing with the external environment. Besides recalling the preliminaries of $\mathcal{ETR}$ (Section 2) and [8] (Section 4), in this paper we:

1. formalize how the external oracle in $\mathcal{ETR}$ can be instantiated using action languages in general, and specifically, with action language $\mathcal{C}$ (Section 3);
2. extend $\mathcal{ETR}$ to deal with repair plans, rather than simply with compensating actions (Section 5);
3. formalize how to automatically infer repair plans when the external environment is expressed as an action language (Section 5);
4. elaborate on the properties of these repair plans (Section 5.3).

2 External Transaction Logic

$\mathcal{ETR}$ [13] is an extension of Transaction Logic [3] to deal with actions performed in an external environment of which an agent has no control. The original Transaction Logic ($\mathcal{TR}$) is a logic to reason about changes in KBs, when these changes are performed as ACID transactions. In a nutshell[2], $\mathcal{TR}$ syntax extends that of first order logic with

[2] For lack of space, and since $\mathcal{ETR}$ is a proper extension of $\mathcal{TR}$ (cf. [13]), we do not include here a detailed overview of $\mathcal{TR}$ alone. For the complete details see e.g. [3].

a serial conjunction operator $\otimes$, where $\phi \otimes \psi$ represents the action composed by an execution of ϕ followed by an execution of ψ. Formulas are read as transactions, and they are evaluated over sequences of KB states (*paths*). A formula (or transaction) ϕ is true over a path π iff the transaction successfully executes over that sequence of states. In other words, in $\mathcal{TR}$ truth means successful execution of a transaction. The logic itself makes no particular assumption about the representation of states, or on how states change. For that, $\mathcal{TR}$ requires the existence of two oracles, one abstracting the representation of KB states and used to query them (data oracle $\mathcal{O}^d$), and another abstracting the way the states change (transition oracle $\mathcal{O}^t$).

Besides the concept of a model of a $\mathcal{TR}$ theory, which allows one to prove properties of the theory independently of the paths chosen, $\mathcal{TR}$ also defines the notion of executional entailment. A transaction is entailed by a theory given an initial state, if there is a path starting in that state on which the transaction succeeds. As such, given a transaction and an initial state, the executional entailment determines the path that the KB should follow in order to succeed the transaction in an atomic way. Nondeterministic transactions are possible, in which case several successful paths exist. Transaction Logic Programs [2] are a special class of $\mathcal{TR}$ theories that extend logic programs with serial conjunction. For them, a proof procedure and corresponding implementation exists, which takes into account the ACID execution of transactions.

To deal also with external actions, $\mathcal{ETR}$ operates over a KB including both an internal and an external component. For that, formally $\mathcal{ETR}$ works over two disjoint propositional languages: $\mathcal{L}_P$ (program language), and $\mathcal{L}_\mathcal{O}$ (oracles primitives language). Propositions in $\mathcal{L}_P$ denote actions and fluents that can be defined in the program. As usual, fluents are propositions that can be evaluated without changing the state and actions are propositions that cause evolution of states. Propositions in $\mathcal{L}_\mathcal{O}$ define the primitive actions and queries to deal with the internal and external KB. $\mathcal{L}_\mathcal{O}$ can still be partitioned into $\mathcal{L}_i$ and $\mathcal{L}_a$, where $\mathcal{L}_i$ denotes primitives that query and change the internal KB, while $\mathcal{L}_a$ defines the external actions primitives that can be executed externally. For convenience, it is assumed that $\mathcal{L}_a$ contains two distinct actions `failop` and `nop`, respectively defining trivial failure and trivial success in the external domain.

Further, it is also defined $\mathcal{L}_a^*$ as the result of augmenting $\mathcal{L}_a$ with expressions $\mathbf{ext}(a, b)$, called external actions, where $a, b \in \mathcal{L}_a$. Such an expression is used to denote the execution of action a, having action b as compensating action. If b is `nop`, then we simply write $\mathbf{ext}(a)$ or a. Note that there is no explicit relation between a and b and that it is possible to define different compensating actions for the same action a in the same program. It is thus the programmer's responsibility to determine which is the correct compensation for action a in a given moment.

To construct complex formulas, the language uses the standard connectives $\wedge, \neg$ and $\otimes$ denoting serial conjunction, where $\phi \otimes \psi$ represents the action composed by an execution of ϕ followed by an execution of ψ.

Definition 1 ($\mathcal{ETR}$ atoms, formulas and programs). *An $\mathcal{ETR}$* atom *is either a proposition in $\mathcal{L}_P$, $\mathcal{L}_i$ or $\mathcal{L}_a^*$ and an $\mathcal{ETR}$ literal is either ϕ or $\neg\phi$ where ϕ is an $\mathcal{ETR}$ atom. An $\mathcal{ETR}$* formula *is either a literal, or an expression, defined inductively, of the form $\phi \wedge \psi$, $\phi \vee \psi$ or $\phi \otimes \psi$, where ϕ and ψ are $\mathcal{ETR}$ formulas.*

An $\mathcal{ETR}$ program is a set of rules of the form $\phi \leftarrow \psi$ where ϕ is a proposition in $\mathcal{L}_P$ and ψ is an $\mathcal{ETR}$ formula.

Example 3. Recall Example 1 regarding a medical diagnosis. A possible (partial) encoding of it in $\mathcal{ETR}$ can be expressed by the following rules:

$$
\begin{aligned}
sick(X) &\leftarrow hasFlu(X)\\
hasFlu(X) &\leftarrow \mathbf{ext}(hasFever(X)) \otimes \mathbf{ext}(hasHeadache(X)) \otimes nonSerious(X)\\
nonSerious(X) &\leftarrow \mathbf{ext}(\neg vomiting(X)) \wedge \ldots \wedge \mathbf{ext}(\neg diarrhea(X))\\
treatment(X,Y) &\leftarrow hasFlu(X) \otimes treatFlu(X,Y) \otimes treatmentHistory(X,Y,Z).ins \otimes\\
&\qquad \mathbf{ext}(goodReaction(X,Y))\\
treatFlu(X,Y) &\leftarrow \mathbf{ext}(giveMeds(X,p_1), giveMeds(X,c_1))\\
treatFlu(X,Y) &\leftarrow \mathbf{ext}(giveMeds(X,p_2), giveMeds(X,c_2))
\end{aligned}
$$

In this example, predicate $treatment(X,Y)$ denotes a transaction for treating patient X with treatment Y. Then, one can say, e.g. in the 4th rule, that such a transaction succeeds if a patient X has flue and a medicine Y to treat the flue is given to X (i.e. transaction $treatFlu(X,Y)$ succeeds). Additionally, after a treatment is issued, the medical history of the patient should be updated and the agent needs to check if the patient shows a positive reaction to the treatment in question. In this sense, the formula $treatmentHistory(X,Y,Z).ins \otimes \mathbf{ext}(goodReaction(X,Y))$ denotes the action composed by updating the treatment history of patient X followed by externally asking if the patient X had a good reaction to treatment Y. Moreover, treating a patient with a flue is encoded by the nondeterministic transaction $treatFlu(X,Y)$ (5th and 6th rules) as the external action of giving patient X the medicine p_1 or the medicine p_2. While the action of asking about the reaction of a patient does not need to be repaired, the same is not true for the action of giving a medication. If a failure occurs, the agent has to compensate for it. This is, e.g. expressed by the external action $\mathbf{ext}(giveMeds(X,p_1), giveMeds(X,c_1))$ where c_1 cancels the effects of p_1.

A state in $\mathcal{ETR}$ is a pair (D,E), where D (resp. E) is the internal (resp. external) state identifier taken from a set $\mathcal{D}$ (resp. $\mathcal{E}$). The semantics of states is provided by 3 oracles, which come as a parameter to $\mathcal{ETR}$: a data oracle $\mathcal{O}^d$ that maps elements of $\mathcal{D}$ into transaction formulas; a transition oracle $\mathcal{O}^t$ that maps a pair of elements from $\mathcal{D}$ into transaction formulas; and an external oracle $\mathcal{O}^e$ that maps a pair of elements from $\mathcal{E}$ into transaction formulas. Intuitively $\mathcal{O}^d(D) \models \varphi$ means that, according to the oracle, φ is true in state D, and $\mathcal{O}^t(D_1,D_2) \models \varphi$ (resp. $\mathcal{O}^e(E_1,E_2) \models \varphi$) that φ is true in the transition of internal (resp. external) states from D_1 to D_2 (resp. E_1 to E_2).

As in $\mathcal{TR}$, $\mathcal{ETR}$ formulas are evaluated in paths (sequence of states). For convenience, as it is necessary in the sequel, paths also include the explicit annotation of the action executed in each transition of states. So $\langle S_1,^{\varphi} S_2\rangle$ means that action φ occurred in the transition of state S_1 into S_2. Then, interpretations map paths to a Herbrand structures. If $\phi \in M(\pi)$ then, in the interpretation M, path π is a valid execution for the formula ϕ. Moreover, we only consider as interpretations the mappings that comply with the specified oracles:

Definition 2 (Interpretations). *An interpretation is a mapping M assigning a classical Herbrand structure to every path. This mapping is subject to the following restrictions, for all states D_i,E_j and every formula φ:*

1. $\varphi \in M(\langle (D, E) \rangle)$ *iff* $\mathcal{O}^d(D) \models \varphi$ *for any external state* E
2. $\varphi \in M(\langle (D_1, E), ^{\varphi} (D_2, E) \rangle)$ *iff* $\mathcal{O}^t(D_1, D_2) \models \varphi$ *for any external state* E
3. $\varphi \in M(\langle (D, E_1), ^{\varphi} (D, E_2) \rangle)$ *iff* $\mathcal{O}^e(E_1, E_2) \models \varphi$ *for any internal state* D

Satisfaction of $\mathcal{ETR}$ formulas over paths, requires the prior definition of operations on paths. For example, the formula $\phi \otimes \psi$ is true (i.e. successfully executes) in a path that executes ϕ up to some point in the middle, and executes ψ from then onwards. To deal with this:

Definition 3 (Path Splits). *A* split *of a path* $\pi = \langle S_1, ^{A_1} \ldots, ^{A_{i-1}} S_i, ^{A_i} \ldots, ^{A_{k-1}} S_k \rangle$ *of size* k *(k-path) is any pair of subpaths,* π_1 *and* π_2*, such that* $\pi_1 = \langle S_1, ^{A_1} \ldots, ^{A_{i-1}} S_i \rangle$ *and* $\pi_2 = \langle S_i, ^{A_i} \ldots, ^{A_{k-1}} S_k \rangle$ *for some* i $(1 \leq i \leq k)$*. In this case, we write* $\pi = \pi_1 \circ \pi_2$*.*

Before we are able to define general satisfaction of formulas, we need two auxiliary relations for constructing compensations. Classical satisfaction is similar to satisfaction in the original $\mathcal{TR}$, and a transaction formula is said to be classically satisfied by an interpretation given a path iff the transaction succeeds in the path without failing any action. A transaction is partially (or partly) satisfied by an interpretation given a path, iff the transaction succeeds in the path up to some point where an action may fail.

Definition 4 (Classical Satisfaction). *Let* M *be an interpretation,* π *a path and* ϕ *a formula.*

1. ***Base Case:*** $M, \pi \models_c \phi$ *iff* $\phi \in M(\pi)$ *for any atom* ϕ
2. ***Negation:*** $M, \pi \models_c \neg\phi$ *iff it is not the case that* $M, \pi \models_c \phi$
3. ***"Classical" Conjunction:*** $M, \pi \models_c \phi \wedge \psi$ *iff* $M, \pi \models_c \phi$ *and* $M, \pi \models_c \psi$*.*
4. ***Serial Conjunction:*** $M, \pi \models_c \phi \otimes \psi$ *iff* $M, \pi_1 \models_c \phi$ *and* $M, \pi_2 \models_c \psi$ *for some split* $\pi_1 \circ \pi_2$ *of path* π*.*

Definition 5 (Partial Satisfaction). *Let* M *be an interpretation,* π *a path and* ϕ *a formula.*

1. ***Base Case:*** $M, \pi \models_p \phi$ *iff* ϕ *is an atom and one of the following holds:*
   (a) $M, \pi \models_c \phi$
   (b) $M, \pi \not\models_c \phi$*,* $\phi \in \mathcal{L}_i$*,* $\pi = \langle (D, E) \rangle$*,* $\neg\exists D_i$ *s.t.* $M, \langle (D, E), ^{\phi} (D_i, E) \rangle \models_c \phi$
   (c) $M, \pi \not\models_c \phi$*,* $\phi \in \mathcal{L}_a^*$*,* $\pi = \langle (D, E) \rangle$*,* $\neg\exists E_i$ *s.t.* $M, \langle (D, E), ^{\phi} (D, E_i) \rangle \models_c \phi$
2. ***Negation:*** $M, \pi \models_p \neg\phi$ *iff it is not the case that* $M, \pi \models_p \phi$
3. ***"Classical" Conjunction:*** $M, \pi \models_p \phi \wedge \psi$ *iff* $M, \pi \models_p \phi$ *and* $M, \pi \models_p \psi$
4. ***Serial Conjunction:*** $M, \pi \models_p \phi \otimes \psi$ *iff one of the following holds:*
   (a) $M, \pi \models_p \phi$ *and* $M, \pi \not\models_c \phi$
   (b) $\exists$ *split* $\pi_1 \circ \pi_2$ *of path* π *s.t.* $M, \pi_1 \models_c \phi$ *and* $M, \pi_2 \models_p \psi$

With this, we say that a transaction ϕ fails and can be compensated only if $M, \pi \models_p \phi$ but $M, \pi \not\models_c \phi$ where the last state of π stands for the exact point where ϕ fails.

Example 4. Consider and internal KB where a state D is a set of ground atoms and $\mathcal{O}^d(D) = D$. Moreover, for every atom p in D, the transition oracle defines the actions $p.ins$ and $p.del$ respectively denoting insertion and deletion of atom p, and where

$p.ins \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 \cup \{p\}$ and $p.del \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 - \{p\}$. Furthermore, consider that the external oracle includes $\mathcal{O}^e(E_1, E_2) \models a$, (i.e. the external execution of a in state E_1 succeeds, and makes the external world evolve into E_2), $\mathcal{O}^e(E_1, E_4) \models c$, and that for every state E, $\mathcal{O}^e(E_2, E) \not\models b$ (i.e. the execution of b in state E_2 fails).

Besides these oracles, consider the following rules defining a transaction t:

$$t \leftarrow p.ins \otimes \mathbf{ext}(a, d) \otimes \mathbf{ext}(b, e)$$
$$t \leftarrow q.ins \otimes \mathbf{ext}(c)$$

In this example, the formula $p.ins \otimes \mathbf{ext}(a, d)$ is classically satisfied by all interpretations in the path $\langle(\{\}, E_1),^{p.ins}(\{p\}, E_1),^{\mathbf{ext}(a,d)}(\{p\}, E_2)\rangle$ while $q.ins \otimes \mathbf{ext}(c)$ is classically satisfied in the path $\langle(\{\}, E_1),^{q.ins}(\{q\}, E_1),^{\mathbf{ext}(c)}(\{q\}, E_4)\rangle$. Moreover, it is easy to check that $\mathbf{ext}(b, e)$ cannot succeed in any path starting in state E_2 (given the external oracle definition). The idea of partial satisfaction is to identify the path $\langle(\{\}, E_1),^{p.ins}(\{p\}, E_1),^{\mathbf{ext}(a,d)}(\{p\}, E_2)\rangle$ as one that partly satisfies the complex formula $p.ins \otimes \mathbf{ext}(a, d) \otimes \mathbf{ext}(b, e)$ up to some point, though it eventually fails since the external action $\mathbf{ext}(b, e)$ fails.

When a formula fails in a path after the execution of some external action, we have to say how these actions can be compensated. To define this, we first need to define some auxiliary operations on paths. To start, one has to collect all actions that have been executed in a path and need to be compensated; and to rollback the internal state:

Definition 6 (Rollback Path, and Sequence of External Actions). *Let π be a k-path of the form $\langle(D_1, E_1),^{A_1}(D_2, E_2),^{A_2} \ldots,^{A_{k-1}}(D_k, E_k)\rangle$. The* rollback *path of π is the path obtained from π by: (1) Replacing all D_is by the initial state D_1; (2) Keeping just the transitions where $A_i \in \mathcal{L}_a^*$.*
The sequence of external actions of π, denoted $\mathtt{Seq}(\pi)$, is the sequence of actions of the form $\mathbf{ext}(a, b)$ that appear in the transitions of the rollback path of π.

$\mathtt{Seq}(\pi)$ only collects the external actions that have the form $\mathbf{ext}(a, b)$. Since this operation aims to compensate the executed actions, then actions without compensations are skipped. With this, a recovery path is obtained from executing each compensation operation defined in $\mathtt{Seq}(\pi)$ in the inverse order.

Definition 7 (Inversion, and Recovery Path). *Let S be a sequence of actions from $\mathcal{L}_a^*$ of the form $\langle \mathbf{ext}(A_1, A_1^{-1}), \ldots \mathbf{ext}(A_n, A_n^{-1})\rangle$. Then, the inversion of S is the transaction formula $\mathtt{Inv}(S) = A_n^{-1} \otimes \ldots \otimes A_1^{-1}$.*

π_r is a recovery path *of $\mathtt{Seq}(\pi)$ w.r.t. M iff $M, \pi_r \models_c \mathtt{Inv}(\mathtt{Seq}(\pi))$.*

We can now say which paths compensate a formula and define satisfaction.

Definition 8 (Compensating Path for a Transaction). *Let M be an interpretation, π a path and ϕ a formula. $M, \pi \rightsquigarrow \phi$ iff* all *the following hold:*

1. *$\exists \pi_1$ such that $M, \pi_1 \models_p \phi$ and $M, \pi_1 \not\models_c \phi$*
2. *$\exists \pi_0$ such that π_0 is the rollback path of π_1*
3. *$\mathtt{Seq}(\pi_1) \neq \emptyset$ and $\exists \pi_r$ such that π_r is a recovery path of $\mathtt{Seq}(\pi_1)$ w.r.t. M*
4. *π_0 and π_r are a split of π, i.e. $\pi = \pi_0 \circ \pi_r$*

Definition 9 (General Satisfaction). *Let M be an interpretation, π a path and ϕ a formula.*

1. ***Base Case:*** *$M, \pi \models \phi$ if $\phi \in M(\pi)$ for any atom ϕ*
2. ***Negation:*** *$M, \pi \models \neg\phi$ if it is not the case that $M, \pi \models \phi$*
3. ***"Classical" Conjunction:*** *$M, \pi \models \phi \wedge \psi$ if $M, \pi \models \phi$ and $M, \pi \models \psi$.*
4. ***Serial Conjunction:*** *$M, \pi \models \phi \otimes \psi$ if $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$ for some split $\pi_1 \circ \pi_2$ of π.*
5. ***Compensating Case:*** *$M, \pi \models \phi$ if $M, \pi_1 \rightsquigarrow \phi$ and $M, \pi_2 \models \phi$ for some split $\pi_1 \circ \pi_2$ of π*
6. *For no other M, π and ϕ, $M, \pi \models \phi$.*

With this notion of satisfaction, a formula ϕ succeeds if it succeeds classically or, if although an external action failed to be executed, the system can recover from the failure and ϕ can still succeed in an alternative path (point 5). Obviously, recovery only makes sense when external actions are performed before the failure. Otherwise we can just rollback to the initial state and try to satisfy the formula in an alternative branching.

Example 5. Recall example 4 and assume that $\mathcal{O}^e(E_3, E_4) \models c$ and $\mathcal{O}^e(E_2, E_3) \models d$. Then, the rollback path of $\pi = \langle(\{\}, E_1),^{p.ins}(\{p\}, E_1),^{\mathbf{ext}(a,d)}(\{p\}, E_2)\rangle$ is the path $\langle(\{\}, E_1),^{\mathbf{ext}(a,d)}(\{\}, E_2)\rangle$ and $\mathtt{Seq}(\pi) = \langle\mathbf{ext}(a, d)\rangle$. Furthermore, the path $\langle(\{\}, E_2),^{a^{-1}}(\{\}, E_3)\rangle$ is a recovery path of $\mathtt{Seq}(\pi)$ w.r.t. any interpretation M.

Based on these, the complex formula $(p.ins \otimes \mathbf{ext}(a, d) \otimes \mathbf{ext}(b, e)) \vee (q.ins \otimes \mathbf{ext}(c))$ is satisfied both in the path $\langle(\{\}, E_1),^{q.ins}(\{q\}, E_1),^{\mathbf{ext}(c)}(\{q\}, E_4)\rangle$ – without compensations – but also in the path: $\langle(\{\}, E_1),^{\mathbf{ext}(a,d)}(\{\}, E_2),^{d}(\{\}, E_3),^{q.ins}(\{q\}, E_3),^{\mathbf{ext}(c)}(\{q\}, E_4)\rangle$ – using point 5 above, in this case.

Definition 10 (Models, Logical and Executional Entailment). *Let ϕ and ψ be two $\mathcal{ETR}$ formulas and M be an interpretation. M is a model of ϕ (denoted $M \models \phi$) iff $M, \pi \models \phi$ for every path π. M is a model of a program P iff for every rule $\phi \leftarrow \psi$ in P, if M is a model of ψ then it is also a model of ψ.*
Then, ϕ logically entails ψ ($\phi \models \psi$) if every model of ϕ is also a model of ψ.

$P, \langle S_1,^{A_1} \ldots,^{A_{n-1}} S_n\rangle \models \phi$ $(\star)$ iff $M, \langle S_1,^{A_1} \ldots,^{A_{n-1}} S_n\rangle \models \phi$ for every model M of P. We also define $P, S_1{-} \models \phi$ to be true (and say that ϕ succeeds in P from the state S_1), if there exists a path $S_1,^{A_1} \ldots,^{A_{n-1}} S_n$ that makes $(\star)$ true.

3 Action Languages in $\mathcal{ETR}$

The general $\mathcal{ETR}$ is parametrized by a set of oracles defining the elementary primitives to query and update the internal and external KB. However, to deal with specific problems, these oracles must be defined. For example, to deal with simple internal KBs, one can define a so-called relational oracle, in which: states are defined by sets of atoms; the data oracle simply returns all these formulas, i.e., $\mathcal{O}^d(D) = D$; the transition oracle defines, for each predicate p, two internal actions, $p.ins$ and $p.del$, respectively stating the insertion and deletion of p as $p.ins \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 \cup \{p\}$, and $p.del \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 - \{p\}$.

If the agent knows nothing about the external environment, the external oracle $\mathcal{O}^e$ can be left open, and whenever the evaluation of an action is required of that oracle, the oracle is called returning either failure or a subsequent successful state (which can be the same state, if the external action is simply a query). However, the agent may have some knowledge about the behavior of the external world. Here we consider the case where the agent's knowledge about the external world can be formalized by Action Languages [9], and show how to define an external oracle for that. Moreover, below we use the external oracle defined in this section to automatically infer repair plans.

Every action language defines a series of *laws* describing actions in the world and their effects. Which laws are possible as well as the syntax and semantics of each law depends on the action language in question. Several solutions like STRIPS, languages $\mathcal{A}, \mathcal{B}, \mathcal{C}$ or $PDDL$, have been proposed in the literature, each with different applications in mind. A set of laws of each language is called an action program description. The semantic of each language is determined by a transition system which depends on the action program description.

Let $\langle \{\texttt{true}, \texttt{false}\}, \mathcal{F}, \mathcal{A} \rangle$ be the signature of an action language, where $\mathcal{F}$ is the set of fluent names and $\mathcal{A}$ is the set of action names in the language. Let $\langle S, V, R \rangle$ be a transition system where S is the set of all possible states, V is the evaluation function from $\mathcal{F} \times S$ into $\{\texttt{true}, \texttt{false}\}$, and finally R is the set of possible relations in the system defined as a subset of $S \times \mathcal{A} \times S$. We assume a function $\mathcal{T}(E)$ that from action program E defines the transition system $\langle S, V, R \rangle$ associated with E, and the previously defined signature. We also define $\mathcal{L}^a = \mathcal{F} \cup \mathcal{A}$.

Equipped with such a function, an $\mathcal{ETR}$ external state is a pair, with the program E describing the external domain and a state of the transition system, and the general external oracle $\mathcal{O}^e$ is (where $\mathcal{T}(E) = \langle S, V, R \rangle$):

1. $\mathcal{O}^e((E, s), (E, s')) \models action$ iff $action \in \mathcal{A} \wedge \langle s, action, s' \rangle \in R$
2. $\mathcal{O}^e((E, s), (E, s)) \models fluent$ iff $fluent \in \mathcal{F} \wedge V(fluent, s) = \texttt{true}$

To be more concrete, let us show one instantiation of this, with action language $\mathcal{C}$ [11]. This language and its extensions like $\mathcal{C}^+$ [10], are known for being traditionally used to represent norms and protocols (e.g. auction, contract formation, negotiation, rules of procedure, communication, etc.) [16,1]

A *state formula* is a propositional combination of fluent names while a *formula* is a propositional combination of fluent names and elementary action names. An external description E is a set of static and dynamic laws. A static law is a law of the form "**caused** F **if** G", where F and G are state formulas. A dynamic law is of the form "**caused** F **if** G **after** U", where F and G are state formulas and U is a formula

An important notion is that actions can be done concurrently. So, in a transition $\langle s_1, A, s_2 \rangle$, A is a subset of $\mathcal{A}$. Intuitively, to execute A from s_1 to s_2 means to execute concurrently the "elementary actions" represented by the action symbols in A changing the state s_1 into s_2. A state is an interpretation of the set of fluents $\mathcal{F}$ that is closed under the static laws. I.e. for every static law "**caused** F **if** G" and every state s, s satisfies F if s satisfies G. Then, the interpretation function V for a state is simply defined as $V(P, s) = s(P)$. To define the set of valid relations R we first need the notion of reduct. For any description E and any transition $\langle s_0, A, s_1 \rangle$ we can define the $E^{\langle s_0, A, s_1 \rangle}$, the reduct of E relative to $\langle s_0, A, s_1 \rangle$, which stands for the set consisting of:

- F for all static laws from E s.t. s_1 satisfies G
- F for all dynamic laws from E s.t. s_1 satisfies G and $s_0 \cup A$ satisfies H

We say that $\langle s_0, A, s_1 \rangle$ is *causally explained* if s_1 is the only state that satisfies the reduct $E^{\langle s_0, A, s_1 \rangle}$. Since the external oracle is defined for elementary actions rather than for sets of actions, we can define the relation R of $\mathcal{T}(E)$ as follows: $\langle s_0, a, s_1 \rangle \in R$ iff $\langle s_0, A, s_1 \rangle$ is causally explained by E and $a \in A$.

4 Reverse Actions in Action Languages

Before defining how to automatically infer repair plans in $\mathcal{ETR}$ plus an external oracle of an action language, we briefly overview [8]'s action reverses, adapting it for the action languages framework defined above.

To start, we need the notion of trajectory of a sequence of actions. Intuitively, we say that a state s_f is the trajectory of a sequence of actions applied to state s_i if there exists a trace from s_i to s_f by executing the given sequence of actions.

Definition 11 (Trajectory of a Sequence of Actions). *We say that s_f is the trajectory of $a_0 \otimes \ldots \otimes a_{m-1}$ when applied to s_0 iff: $\exists s'_1, \ldots, s'_m$ s.t. $\langle s_0, a_0, s'_1 \rangle \in R$ and $\langle s'_i, a_i, s'_{i+1} \rangle \in R$ then $s'_m = s_f$ where $(1 \leq i \leq m-1)$. In this case we write* $\texttt{traj}(s_0; [a_0 \otimes \ldots \otimes a_{m-1}]) = s_f$.

With this we can define the notion of reverse action. An action a^{-1} is a reverse action of a if whenever we execute a^{-1} after we execute a, we always obtain the (initial) state before the execution of a. This is encoded as follows.

Definition 12 (Reverse Action). *Let a, a^{-1} be actions in $\mathcal{A}$. We say that an action a^{-1} reverses a iff $\forall s_1, s_2$ if $\langle s_1, a, s_2 \rangle \in R$ then $\exists s. \langle s_2, a^{-1}, s \rangle \in R$ and $\forall s. \langle s_2, a^{-1}, s \rangle \in R, s = s_1$. In this case we write* $\texttt{revAct}(a; a^{-1})$.

Besides the notion of reverse action, the authors of [8] also introduce the notion of reverse plan. Since a single action may not be enough to reverse the effects of another action, the notion of reverse is generalized into a sequence of actions, or *plan*. A reverse plan defines what sequences of actions are able to reverse the effects of one action.

Definition 13 (Reverse Plan). *Let $a, a_0, \ldots, a_{m-1}$ be actions in $\mathcal{A}$. We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a plan that reverses action a iff $\forall s_1, s_2$ s.t. $\langle s_1, a, s_2 \rangle \in R$ then $\exists s'$ s.t.* $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s'$ *and $\forall s'$ s.t.* $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s'$ *then $s' = s_1$. In this case we write* $\texttt{revPlan}(a; [a_0 \otimes \ldots \otimes a_{m-1}])$.

Intuitively, a reverse plan is a generalization of a reverse action, as every reverse action $\texttt{revAct}(a, a')$ is a reverse plan of size one: $\texttt{revPlan}(a, [a'])$.

The previous definitions denote a strong relation between an action and a sequence of actions which holds for *any* state in the set of states defined in the framework. I.e., a sequence of actions is a reverse plan of a given action, if the sequence can always be applied after the execution of a and, in all the transitions defined in the set R, the application of this sequence always leads to the state before the execution of a.

However, some states may prevent the existence of a reverse plan. I.e., an action may have a reverse plan under some conditions, that do not necessarily hold at every reachable state. Thus, we need a weaker notion of reverse that takes into account the information of the states, e.g. values of some fluents obtained by sensing. By restraining the states where the reverse plan is applied, we might get reverse plans that were not applicable before. This is the idea of conditional reversal plan formalized as follows.

Definition 14 (Conditional Reversal Plan). *Let $a, a_0, \ldots, a_{m-1}$ be actions in $\mathcal{A}$. We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a ϕ;ψ-reverse plan that reverses action a back iff: $\forall s_1, s_2$ where $V(s_2, \phi) = V(s_1, \psi) = \texttt{true}$, if $\langle s_1, a, s_2 \rangle \in R$ then $\exists s'$ s.t. $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s'$ and $\forall s'$ s.t. $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s'$ then $s' = s_1$.*

5 $\mathcal{ETR}$ with automatic compensations

After defining the reversals of actions for action languages, we can show how $\mathcal{ETR}$'s external oracle can be instantiated to use these definitions and automatic infer what is the correct repair plan for each action.

However, we do not need such a strong and generic notion of reverse action as the one defined in [8]. In fact, both reverse actions and reverse plans are defined disregarding the initial state where they are being applied. When defining compensations or repairs of actions in $\mathcal{ETR}$, we already have information about the specific states where the repairs will be applied. This demands for a weaker notion of reverse action and reverse plan, defined for a pair of states rather than for a given action.

Definition 15 (Situated Reverse Action). *We say that an action a^{-1} reverses s_2 into s_1 iff $\exists s.\langle s_2, a^{-1}, s\rangle \in R$ and $\forall s.\langle s_2, a^{-1}, s\rangle \in R, s = s_1$. In this case we write $\texttt{revAct}(s_1, s_2; a^{-1})$.*

Intuitively, we say that action a is a reverse action for states s_1 and s_2 iff a can be executed in state s_2 and *all* the transitions that exist in the set of relations R w.r.t. action a applied to state s_2 end in state s_1.

As in [8], instead of only considering singleton actions, we also define the notion of situated reverse plan to specify sequences of actions that are able to reverse the effects of one action. Then, $\texttt{revPlan}(s_1, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$ states that the sequence of actions $a_0 \otimes \ldots \otimes a_{m-1}$ always restores s_1 when executed in state s_2. For that, the KB may pass through m arbitrary states necessarily ending in s_2.

Definition 16 (Situated Reverse Plan). *We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a plan that reverses s_2 back to s_1 iff: $\exists s_f$ s.t. $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s_f$ and $\forall s_f$ s.t. $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s_f$ then $s_f = s_1$. In this case we write $\texttt{revPlan}(s_1, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$*

Clearly, several reverse plans may exist restoring s_1 from state s_2. Moreover, there are better reverse plans than others. E.g., imagine that in a state s_i there exists an action a_i that always leads us to the same state s_i, i.e. $\langle s_i, a_i, s_i\rangle \in R$. If a plan exists to restore the system back from s_2 to s_1 passing into state s_i, then there are several plans where the only difference is the amount of times we execute the "dummy" action a_i.

Since recovery is a sensitive operation, in order to minimize the amount of operations to be executed, we define the notion of shorter reverse plans. A shorter reverse plan $\texttt{revPlan}_s(s_1, s_2; [a_1 \otimes \ldots \otimes a_m])$ is a reverse plan where the number of actions to be executed is minimal (i.e. there is no other $\texttt{revPlan}(s_1, s_2; [a_1 \otimes \ldots \otimes a_n])$ with $n < m$).

5.1 Goal Reverse Plans

The previous notions define a reverse action or a reverse plan for a pair of states s_1 and s_2, reverting the system from state s_2 back to state s_1, and imposing that the final state obtained is *exactly* s_1. However, it may happen that, for some pair of states, a reverse plan does not exist. Furthermore, if some information is provided (e.g. by the programmer) about the state that we intend to reach, then we might still achieve a state where this condition holds. This is useful for cases where the agent has to find repairs to deal with norm violations. For instance, it may not be possible to return to the exact state before the violation, but it may be possible to reach a consistent state where the agent complies with all the norms.

This corresponds to the notion of goal reverse plans that we introduce here. Based on a state formula ϕ characterizing the state that we want to reach, then $\texttt{goalRev}(\phi, s_2 ; [a_0 \otimes \ldots \otimes a_{m-1}])$ says that the sequence $a_0 \otimes \ldots \otimes a_{m-1}$ reverses the system from s_2 into a consistent state s where the state formula ϕ holds.

Definition 17 (Goal Reverse Plan). *We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a goal plan that reverses s_2 to a state where ϕ holds iff $\exists s'$ s.t. $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s'$ and $\forall s'$ s.t. $\texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) = s'$ then $V(\phi, s') = \texttt{true}$. In this case we write $\texttt{goalRev}(\phi, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$.*

As before, to preserve efficiency of plans, we define the notion of shorter goal reverse plan. $\texttt{goalPlan}_s(\phi, s_2; [a_1 \otimes \ldots \otimes a_m])$ holds, if the sequence $a_1 \otimes \ldots \otimes a_m$ is a sequence with minimal length that takes s_2 into a state where ϕ is true.

5.2 External Oracle for Action Languages with Automatic Compensations

We can now make precise how and when repairs are calculated in $\mathcal{ETR}$'s semantics, and what changes of $\mathcal{ETR}$'s language are needed to deal with these automatic repairs.

Besides defining automatically inferred repairs, we want to keep the option of explicitly defining compensations for external actions. The latter are useful in external environments where the agent is not able to automatically infer the repair (see e.g. the repairs in Example 1). However, since more than one action may be required to repair the effects of one external action (e.g., in Example 1 it may be necessary to give the patient a series of medications in order to repair the side-effects of the previously given one), we also extend these explicitly defined compensations to plans.

Consequently, the language of $\mathcal{ETR}$ is extended so that external actions can appear in a program in three different ways: 1) without any kind of compensation associated, i.e. $\mathbf{ext}(a, \texttt{nop})$, and in this case we write $\mathbf{ext}(a)$ or simply a, where $a \in \mathcal{L}_a$; 2) with a user defined repair plan, written $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ where $a, b_i \in \mathcal{L}_a$; 3) with an automatic repair plan, denoted $\mathbf{extA}(a[\phi])$, where $a \in \mathcal{L}_a$, ϕ is an external state formula, and an external state formula is a conjunction of external fluents. Formally:

Definition 18. *An $\mathcal{ETR}$* atom *is either a proposition in $\mathcal{L}_P$, $\mathcal{L}_i$ or an external atom. An external atom is either a proposition in $\mathcal{L}_a$ (where $\mathcal{L}_a = \mathcal{F} \cup \mathcal{A}$), $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ or $\mathbf{extA}(a[\phi])$ where $a, b_i \in \mathcal{L}_a$ and ϕ is an external state formula. An $\mathcal{ETR}$ literal is either ϕ or $\neg\phi$ where ϕ is an $\mathcal{ETR}$ atom. An external state formula is a either a literal from $\mathcal{F}$ or an expression $\phi \wedge \psi$ where ϕ and ψ are external state formulas. An $\mathcal{ETR}$* formula *is either a literal, or an expression, defined inductively, of the form $\phi \wedge \psi$, $\phi \vee \psi$ or $\phi \otimes \psi$, where ϕ and ψ are $\mathcal{ETR}$ formulas. An $\mathcal{ETR}$* program is a set of rules of the form $\phi \leftarrow \psi$ where ϕ is a proposition in $\mathcal{L}_P$ and ψ is an $\mathcal{ETR}$ formula.

Intuitively, $\mathbf{extA}(a[\phi])$ stands for *"execute the external action a, and if something fails automatically repair the action's effects either leading to the state just before a was executed, or to a state where ϕ holds"*. When one wants the repair to restore the system to the very state just before a was executed, one may simply write $\mathbf{extA}(a)$ (equivalent to $\mathbf{extA}(a[\bot])$).

Example 6. With this extended language one can write, e.g. for the situation described in Example 2, rules like the ones below, plus a specification in $\mathcal{C}$ of the external environment which must include the definition of blocks-world-like actions (omitted here for brevity). Intuitively the rules say that: to place a product one should decrease the stock and then place the product; one can place a product in a better shelf, or in a normal shelf in case the product is not premium. Moreover, moving a product to a given shelf is an external action that can be automatically repaired based on the existing information about the external world. Consequently $\mathbf{extA}(move(X, warehouse, betterShelf))$ means that, if something fails after the agent has moved X from the warehouse into a better shelf, then a repair plan will be automatically defined for this action by the semantics.

$$
\begin{aligned}
placeProduct(X) &\leftarrow decreaseStock(X) \otimes X > 0 \otimes placeOne(X) \\
decreaseStock(X) &\leftarrow stock(X, S) \otimes stock(X, S).del \otimes stock(X, S-1).ins \\
placeOne(X) &\leftarrow \mathbf{extA}(move(X, warehouse, betterShelf)) \\
placeOne(X) &\leftarrow \neg premium(X) \otimes \mathbf{extA}(move(X, warehouse, normalShelf))
\end{aligned}
$$

Note that, the semantics must ensure that the external world is always left consistent by the agent in this example. Particularly, whenever it is not possible to place a non-premium product in the better shelf, a repair plan is executed to put the product back in the warehouse, where after one can try to put the product in the normal shelf; if it is not possible to put the product in either shelf (or to put a premium product in the better shelf), then a repair plan is executed to put the product back in the warehouse, and the stock is rolled back to its previous value (and the transaction fails).

Contrary to the semantics of the original $\mathcal{ETR}$ which is independent of the defined oracles, the semantics of this new language can only be defined given specific oracles that allow the inference of repair plans. For example, for external environments described by action languages, an external state is a pair, with the action program E describing the external domain and a state of the transition system, and the external oracle $\mathcal{O}^e$ is (where $\mathcal{T}(E) = \langle S, V, R \rangle$):

Definition 19 (Action Language Oracle). *Let f, a be atoms in $\mathcal{L}_a$ s.t. f is a fluent in $\mathcal{F}$ and a is an action in $\mathcal{A}$.*

1. $\mathcal{O}^e((E, s_1), (E, s_1)) \models f$ *iff* $V(f, s_1) = \texttt{true}$

2. $\mathcal{O}^e((E,s_1),(E,s_2)) \models a$ *iff* $\langle s_1, a, s_2\rangle \in R$
3. $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a, b_1 \otimes \ldots \otimes b_n)$ *iff* $\langle s_1, a, s_2\rangle \in R$
4. $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a[\phi], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ *iff one holds:*
   (a) $\langle s_1, a, s_2\rangle \in R \wedge \texttt{revPlan}_s(s_1, s_2; [a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}])$*; or*
   (b) $\langle s_1, a, s_2\rangle \in R \wedge (\neg\exists a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ *s.t.* $\texttt{revPlan}_s(s_1, s_2; [a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}])) \wedge \texttt{goalRev}_s(\phi, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$

Points 3 and 4 above define how the oracle satisfies external actions with compensations. If the agent wants to explicitly define $b_1 \otimes \ldots \otimes b_n$ as the reverse plan for action a, then $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_n)$ is evaluated solely by what the oracle knows about a, holding in a transition iff a holds in that transition of states.

When the agent wants to infer a repair plan for a, then these repairs are calculated based on the notions of reverse plan and goal reverse defined previously. Namely, the formula $\mathbf{ext}(a[\phi], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ holds, iff a holds in the transition s_1 into s_2, and $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is a shorter reverse plan to repair s_2 back to s_1 or, if $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is a shorter goal plan to repair s_2 into a state where the state formula ϕ holds.

Note that the order of points 4a and 4b is not arbitrary. Goal reverse plans provide an elegant solution to relax the necessary conditions to obtain repairs plans and are specially useful in scenarios where it is not possible to return to the initial state before executing the external action, as e.g. in norms or contracts violations. However, care must be taken when defining the external state formula ϕ of an external action $\mathbf{extA}(a[\phi])$. In fact, if ϕ provides a very incomplete description of the state that we want to achieve, then we might achieve a state substantially different from the intended one. Particularly, although we constrain the applicability of goal reverse plans to the ones that are shorter, there is no guarantee that the changes of these plans are minimal (w.r.t. the amount of fluents that are different from the previous state). To guarantee such property represents a belief revision problem and is, at this moment, out of scope of this paper.

Finally, compensations can be instantiated by changing the definition of interpretation (Def. 2) which now determines how to deal with automatic repairs.

Definition 20 (Interpretations). *An interpretation is a mapping M assigning a classical Herbrand structure (or $\top$) to every path. This mapping is subject to the following restrictions, for all states D_i,E_j and every formula φ, every external atom a and every state formula ψ:*

1. $\varphi \in M(\langle(D,E)\rangle)$ *iff* $\mathcal{O}^d(D) \models \varphi$ *for any external state E*
2. $\varphi \in M(\langle(D_1,E),^{\varphi}(D_2,E)\rangle)$ *iff* $\mathcal{O}^t(D_1,D_2) \models \varphi$ *for any external state E*
3. $\varphi \in M(\langle(D,E_1),^{\varphi}(D,E_2)\rangle)$ *iff* $\mathcal{O}^e(E_1,E_2) \models \varphi$ *for any internal state D*
4. $\mathbf{extA}(a[\psi]) \in M(\langle(D,E_1),^{\mathbf{ext}(a[\psi],a_0^{-1}\otimes\ldots\otimes a_{m-1}^{-1})}(D,E_2)\rangle)$ *iff* $\mathcal{O}^e(E_1,E_2) \models \mathbf{ext}(a[\psi], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ *for any internal state D*

Note that, an external action with automatic repair plans only appears in the program in the form $\mathbf{extA}(a[\phi])$. With this previous definition, it is the interpretation's responsibility to ask the oracle to instantiate it with the correct repair plan $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$.

5.3 Properties of Repair Plans

We start by making precise the relation between the concepts presented here, and the definitions from [8]. Specifically, if a goal reverse plan is not considered, then $a_0^{-1} \otimes$

$\ldots \otimes a_{m-1}^{-1}$ is a valid repair plan iff it is a $\phi;\psi$-conditional plan in [8] where the ψ (respectively ϕ) represents the state formula of the initial (resp. final) state s_1 (resp. s_2).

Theorem 1 (Relation with [8]). *Let F_1 and F_2 be formulas that respectively represent completely the states s_1 and s_2. Then, $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a[\bot], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ iff $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is a $F_2;F_1$-reversal for a*

Then, we can apply the result on the sufficient condition for the existence of repairs plans from [8] which is based on the notion of involutory actions. An action is said to be involutory if executing the action twice from any state where the action is executable, always results in the starting state, i.e. iff for every s_1, s_2 s.t. $\mathtt{traj}(s_1; [a \otimes a]) = s_2$ then $s_2 = s_1$. An example of an involutory action is a toggle action, as toggling a simple switch twice will always lead the system into the initial state.

Lemma 1. *Let a be an involutory action. For every pair of states s_1, s_2 s.t. $\langle s_1, a, s_2\rangle \in R$ it holds that $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a[\phi], a)$ for every state formula ϕ.*

Further, we can talk about safety of repairs w.r.t. programs. We say that a program is *repair safe* iff all its external actions have a repair that is guaranteed to succeed.

Theorem 2 (Repair Safety). *Let P be a $\mathcal{ETR}$ program without user defined repair plans of the form $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$. If for every $\mathbf{extA}(a[\phi])$ defined in P there exists a reverse plan $a_1 \otimes \ldots \otimes a_k$ s.t. $\mathtt{revPlan}(a, [a_1 \otimes \ldots \otimes a_k])$ then P is a repair safe program.*

Note that, although the conditions for a repair safe program are considerably strong, they allow us to reason about the safeness of a program *before* execution. Obviously, we do not want to restrict only to repair safe programs. However, if an agent is defined by a repair safe program, we know that, whatever happens, the agent will always leave the external world in a consistent state.

We can also define a safe property regarding a particular execution of a transaction.

Theorem 3 (Repair Safe Execution). *Let P be a program without user defined repairs, ϕ be a formula, π be a path and M an interpretation where $M \models P$. If $M, \pi \models_p \phi$, $M, \pi \not\models_c \phi$ and $\mathtt{Seq}(\pi) \neq \emptyset$ then $\exists \pi_0, \pi_r$ where π_0 is a rollback path of π, and π_r is a recovery path of π_0 s.t. $\pi' = \pi_0 \circ \pi_r$ and $M, \pi' \rightsquigarrow \phi$*

This result talks about the existence of compensating paths for a given transaction ϕ being executed in a path π. Intuitively, if P does not contain user defined transactions, and π is an execution of ϕ that fails (i.e. $M, \pi \models_p \phi$ but $M, \pi \not\models_c \phi$) after executing some external actions (i.e. if $\mathtt{Seq}(\pi) \neq \emptyset$), then there always exists a path where the execution of ϕ is repaired, i.e. there exists a path π' where $M, \pi' \rightsquigarrow \phi$ holds.

Note that these theorems only provide guarantees for programs where explicit user defined repairs are not presented. The problem with the user defined repairs is that it is impossible to predict, before execution, what will be the resulting state of the external world after their execution, or to guarantee any properties about this resulting state. As

such, it may be the case that the existence of user defined repairs jeopardizes the applicability of automatic repair plans. This is as expected: since the user may arbitrarily change the repair of some actions, it may certainly be the case that the specification of the external domain cannot infer any repair plan for other actions in the same sequence. To prevent this, we could preclude the possibility to define user defined repairs. However, this would make $\mathcal{ETR}$ less expressible, making it impossible to use whenever the agent does not possess enough information about the external world.

6 Conclusions and Related Work

We have extended the $\mathcal{ETR}$ language to deal with external environments described by an action language, and to deal with automatically inferred repair plans when some external action fails. The obtained language is able to reason and act in a two-fold environment in a transactional way. By defining a semantics that automatically infers what should be the repairs when something fails in the external world, we ease the programmer's task to anticipate for all the possible failures and write the corresponding correct repair for it. Thus, when enough information is available regarding the external world, $\mathcal{ETR}$ can be used to automatically infer plans to deal with failures. Contrarily, when the agent has no information about the external environment on which she performs actions, then repair plans can be defined explicitly in the agent's program. Though not presented here for lack of space, we have devised a proof procedure for $\mathcal{ETR}$ [13], that readily provides a means for an implementation that is underway.

For dealing with the inference of repair plans, we assumed that the environment is described using the action language $\mathcal{C}$ and based the representation of reversals on the work of [8]. An alternative would be to chose another language for representing changes in the external environment like [17]. [17] defines an action language to represent and reason about commitments in multi-agent domains. In it, it is possible to encode directly in the language which actions are reversible and how. Using this language to represent the external world in $\mathcal{ETR}$ could also be done by changing the external oracle definition, similarly to what we have done here. However, we chose the reversals representation from [8] since its generality makes it applicable to a wider family of action aanguages, like, e.g. the action language $\mathcal{C}$. Since this latter language has several extensions that are already used for norms and protocol representation in multi-agent systems [16,1], by defining an external oracle using this action language $\mathcal{C}$ we provide means to employ such representations together with $\mathcal{ETR}$, extending them with the possibility to describe an agent's behavior in a transactional way. Furthermore, our version of goal reverse plans can be seen as a contribution to the work of [8]'s as it provides means to relax the conditions for the existence of plans, increasing the possibility of achieving a state with some desirable consistent properties.

Several languages to describe an agent's behavior partitioned over an internal and external KB have been proposed in the literature. Jason[4], 2APL[6] and 3APL[14] are successful examples of agent programming languages that deal with environments with both internal updates and external actions. All these language have some way to deal with action failures, and to execute repair plans of some form. However, none of them consider the automatic inference of the repair plans based on the external information

available. Moreover, none of them guarantees transactional properties, in particular for actions performed in the internal environment.

Other agent's logic programming based languages exist (e.g. [5,15]) but, to our knowledge, none of them deals with transactions nor with repair plans. The closest might be [15], where the authors mention as future work the definition of transactions. However, the model theory of [15] does not consider the possibility of failure and thus neither the possibility of repairing plans. Contrarily, its operational semantics may react to external events defining failure of actions performed externally, but since no tools are provided to model the external environment, the decision about what to do with the failure is based only on internal knowledge (but which has information about external events). Moreover, since there is no strict distinction between action performed externally and internally, it not clear to see how the semantics would deal with the different levels of atomicity that the combination between internal and external actions demands.

References

1. A. Artikis, M. J. Sergot, and J. V. Pitt. Specifying norm-governed computational societies. *ACM Trans. Comput. Log.*, 10(1), 2009.
2. A. J. Bonner and M. Kifer. Transaction logic programming. In *ICLP*, pages 257–279, 1993.
3. A. J. Bonner and M. Kifer. Transaction logic programming (or a logic of declarative and procedural knowledge). Technical Report CSRI-323, University of Toronto, 1995.
4. R. H. Bordini, M. Wooldridge, and J. F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.
5. S. Costantini and A. Tocchio. About declarative semantics of logic-based agent languages. In *DALT*, pages 106–123, 2005.
6. M. Dastani. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
7. M. Dastani, J.-J. C. Meyer, and D. Grossi. A logic for normative multi-agent programs. *J. Log. Comput.*, 23(2):335–354, 2013.
8. T. Eiter, E. Erdem, and W. Faber. Undoing the effects of action sequences. *J. Applied Logic*, 6(3):380–415, 2008.
9. M. Gelfond and V. Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.
10. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artif. Intell.*, 153(1-2):49–104, 2004.
11. E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *AAAI/IAAI*, pages 623–630. AAAI / The MIT Press, 1998.
12. A. S. Gomes and J. J. Alferes. Transaction logic with external actions. In *LPNMR*, pages 272–277, 2011.
13. A. S. Gomes and J. J. Alferes. Extending transaction logic with external actions. *Theory and Practice of Logic Programming, On-line Supplement*, To appear, 2013.
14. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
15. R. A. Kowalski and F. Sadri. Abductive logic programming agents with destructive databases. *Ann. Math. Artif. Intell.*, 62(1-2):129–158, 2011.
16. M. J. Sergot and R. Craven. The deontic component of action language nC+. In *DEON*, pages 222–237. Springer, 2006.
17. T. C. Son, E. Pontelli, and C. Sakama. Formalizing commitments using action languages. In *DALT*, pages 67–83, 2011.