Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Error Classification in Action Descriptions: A Heuristic Approach

Thomas Eiter and **Michael Fink** and **Ján Senko**
Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria.
Email: {eiter, fink, senko}@kr.tuwien.ac.at

Abstract

Action languages allow to formally represent and reason about actions in a highly declarative manner. In recent work, revision and management of conflicts for domain descriptions in such languages wrt. semantic integrity constraints have been considered, in particular their reconciliation. However, merely ad hoc tests and methods have been presented to aid the user in analyzing and correcting a flawed description. We go beyond this and present a methodology on top of such tests for identifying a possible error, which works in several stages. The issue of such a methodology for action languages is novel and has not been addressed before, but is important for building tools and engineering action descriptions in practice.

Introduction

Action languages (Gelfond & Lifschitz 1998) are a tool to formally represent and reason about actions in a highly declarative manner. Action domain descriptions comprise statements about actions and their effects, and have a well-defined meaning in terms of transition diagrams, which are directed graphs whose nodes correspond to states and whose edges correspond to transitions caused by occurrences and non-occurrences of actions. This can be exploited to solve various reasoning problems, including planning, temporal projection, etc. and has been successfully applied in various areas, e.g. (Grell *et al.* 2006; Watson & Chintabathina 2003; Zepeda *et al.* 2005).

Table 1 shows a formalization of the well-known Yale Shooting (YS) in the action language $\mathcal{C}$; the corresponding transition diagram is shown in Figure 1. The agent can take the actions $load$ and $shoot$, which affect fluents $loaded$ and $alive$, where the latter is about the status of a turkey on the scene. The statements $i_1 - i_4$ say that the fluents are inertial, i.e., their values stay the same unless there is a reason for change. The statements d_1 – d_4 state the action effects.[1]

Besides the action description, a set of conditions (axioms or observations) represented in an action query language (Gelfond & Lifschitz 1998) might provide further knowledge. For example, the condition

$$\textbf{possibly } \neg alive \textbf{ after } shoot, \neg load \textbf{ if } loaded. \qquad (1)$$

i_1 :	**inertial** $alive.$	i_2 :	**inertial** $\neg alive.$
i_3 :	**inertial** $loaded.$	i_4 :	**inertial** $\neg loaded.$
d_1 :	**caused** $loaded$ **after** $load.$		
d_2 :	**caused** $\neg alive$ **after** $load, shoot.$		
d_3 :	**caused** $\neg loaded$ **after** $shoot.$		
d_4 :	**caused** $loaded$ **after** $\neg loaded, shoot.$		

Table 1: Formalization of the Yale Shooting.

expresses that, after simply shooting when the gun is loaded, the turkey may be dead. However, while desired, the formalization in Table 1 does not fulfill it; in fact, there is no such transition from the state $s_1 = \{loaded, alive\}$ by taking the action $shoot$; the formalization is flawed and must be fixed.

In (Eiter *et al.* 2006a), revision of an action description D with respect to a set of conditions Q has been studied, which can be used to reconcile D and Q by changing the former in case of a conflict as above. To this end, a *suite of test queries* has been described which the user may ask to get useful information about D, in terms of states and transitions involving violations, which might help her to get an idea about what is wrong with D; Eiter *et al.* (2006b) elaborate on this, describing an implementation. In our example, a test **T4** for underspecified fluents wrt. condition (1) would reveal that at the state s_1, the fluent value $\neg alive$ is not (as desired) caused by the action $shoot$; in fact, the dynamic law d_2 is flawed since $load$ should be actually $loaded$.

While the above works were important steps, they did not address the *crucial issue of how to use the tests* in order to single out possible errors in the description in a systematic way. That is, a *methodology to identify possible errors* was completely missing; without it, a non-expert user likely follows a time consuming trial and error strategy for repair, and the underlying reason of error may remain opaque.

In this paper, we tackle this non-trivial problem and provide such a methodology, which comprises three subsequent stages: forming error assumptions by analyzing the emerging conflicts; confirming the assumptions based on plausibility; and localizing the error by narrowing down the part of D in which it presumably occurs.

The methodology uses a heuristics that is based on both analytic and empirical elements, and has been tested in a number of experiments on different domains. An important component is a *taxonomy of error types* referring to properties of actions, fluents, and domains in general, which is

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]This encoding disregards *False* in heads of causal laws.

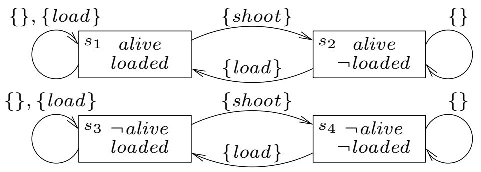

Figure 1: Transition diagram of the Yale Shooting.

of independent interest. The experiments showed that by using the methodology, a correct error assumption was reliably found and the error tracked down to a small part of a larger action description.

Our work contributes to tools and methods which are needed to bring action languages from theory to practice. To our knowledge, a methodology for finding errors in an action description has not been addressed before, and is novel.

Preliminaries

Action descriptions. We consider action descriptions in a fragment of the action description language $\mathcal{C}$ (Giunchiglia & Lifschitz 1998) that consists of expressions termed *causal laws* which are distinguished into *static laws*

$$\textbf{caused } l \textbf{ if } c, \tag{2}$$

where l is a fluent literal or $False$, and c is a propositional combination of fluent names; and *dynamic laws* of the form

$$\textbf{caused } l \textbf{ if } c \textbf{ after } p, \tag{3}$$

where l and c are as above, and p is a propositional combination of fluent names and action names (basic actions). In causal laws the **if** part can be dropped if c is $True$.[2] An *action description* is a set of causal laws.

For instance, the Yale Shooting described in the introduction can be represented by the causal laws in Table 1. It also includes laws of inertia **inertial** l, which we use as an abbreviation for the dynamic causal law **caused** l **if** l **after** l.

The meaning of such an action description, D, can be represented by a transition diagram, $T(D)$—a directed graph whose nodes correspond to the states of the world, $S(D)$, and the edges to the transitions, $R(D)$, describing action occurrences; Figure 1 shows an example.[3]

Intuitively, states are given by total interpretations of the fluent names that satisfy all static laws. Transitions are triples $\langle s, A, s'\rangle$, where s and s' are states and A is an action (i.e., a total interpretation of action names) such that s' is the only interpretation satisfying the heads l of all applicable causal laws (i.e., if $s \cup A \models p$ and/or $s' \models c$).

Conditions. We consider the problem of revising action descriptions in the presence of conflicts between the action description and a set of conditions (axioms or observations) represented in an action query language (Gelfond & Lifschitz 1998). For expressing these conditions, we consider *possibility* and *necessity queries* of the respective form

$$\textbf{possibly } \psi \textbf{ after } A \textbf{ if } \phi \tag{4}$$
$$\textbf{necessarily } \psi \textbf{ after } A \textbf{ if } \phi \tag{5}$$

where ϕ and ψ are propositional combinations of fluent names, and A is an action.

For instance, shooting the loaded gun may lead to a state where the turkey is dead is expressed by the possibility query (1), whereas the fact that one cannot load and shoot at the same time is expressed as:

$$\textbf{necessarily } False \textbf{ after } shoot, load. \tag{6}$$

Intuitively, possibility conditions are violated *at states* (that satisfy ϕ) by missing transitions (with action A to a state s' that satisfies ψ). Necessity queries are violated by existing *transitions* ($\langle s, A, s'\rangle$ such that $s \models \phi$ but $s' \not\models \psi$).

In general, if there is a conflict between an action description and a condition, then it is difficult to formalize the process of arriving at appealing "repairs", which often depends on additional knowledge or intuitions of the designer. We aim at supporting a designer in conflict analysis by employing AD-Query[4] (Eiter *et al.* 2006b), a tool that allows a user to issue a number of relevant tests on an action description in the fragment of $\mathcal{C}$ we consider and its associated transition diagram in the presence of conditions. The tool computes answers to these tests by answer-set programming, in particular using disjunctive logic programming (DLP)—disjunction is actually needed due to Σ_2^p-completeness of most of the tests (Eiter *et al.* 2006a).

Tests. The tests a designer can issue by this tool, resemble questions about a set of conditions (queries), Q, and D, respectively $T(D)$, as in (Eiter *et al.* 2006a). They focus on dynamic aspects, i.e., $S(D)$ is assumed to be correct and hence static laws need not be considered. The tool allows for *partial actions* (i.e, a partial interpretation of action names) in queries. Such a query represents the set of queries obtained by replacement with any compatible action (completion to a total interpretation). We assume that Q is not contradictory and give an informal summary of the tests that we use in this paper. For further details, see (Eiter *et al.* 2006a).

To better understand the reasons for conflicts, a designer may want to extract information from $T(D)$, e.g., information about states, respectively transitions, violating a query q in Q, or information about candidates for transitions, that do not constitute transitions due to *under-specification* (i.e., not every fluent is causally explained):

T1: Which states of $T(D)$ that satisfy a given formula ϕ, violate q?

T2: Given formulas ψ and ϕ, which transitions $\langle s, A, s'\rangle$ of $T(D)$ such that s satisfies ϕ and s' satisfies ψ, violate q?

T3: Given a literal L, for every state s of $T(D)$ such that s satisfies ϕ, is there some under-specified transition candidate $tc = \langle s, A, s'\rangle$ for D such that s' satisfies $\psi \wedge L$ and L is under-specified relative to tc?

[2] $True$ ($False$) is the empty conjunction (resp. disjunction).

[3] Potential transitions may be missing because of inconsistency, e.g., any transition with $A = \{load, shoot\}$ due to d_1 and d_3.

[4] www.kr.tuwien.ac.at/research/ad-query

D	**Domain Properties**	X	**Action Executability**
D.1	Concurrency	X.1	Concurrency
D.2	Event-driven	X.2	Precondition
D.3	Time-driven	X.3	Unconditional NC
		X.4	Conditional NC
		X.5	General
F	**Fluent Properties**	E	**Action Effects**
F.1	Inertia	E.1	Deterministic
F.2	Persistency	E.2	Non-deterministic

Table 2: Classification of errors.

T4: Which transition candidates $tc = \langle s, A, s' \rangle$ for D such that s satisfies ϕ and s' satisfies ψ are under-specified?

Example 1 Test **T1** reveals for the Yale Shooting in Table 1, that the condition (1) is violated at state s_1. A further test **T4** exhibits that the transition $\langle s_1, \{shoot, \neg load\}, s_4 \rangle$ is under-specified with $\neg alive$ as underspecified fluent. □

Our aim is to provide a methodology of applying these tests in order to reveal possible causes of conflicts, i.e., to classify an error in order to assist the designer localizing it.

Error Heuristics

We propose a heuristic methodology that supports the designer of an action description in conflict analysis and error classification. It comprises three tasks at subsequent stages:

Error Assessment: Conflicts between an action description and corresponding conditions that are not satisfied (observations) are analyzed, yielding an assumption about the type of the present error.

Assumption Confirmation: An assumption on the error type is checked for plausibility and confirmed or rejected.

Error Localization: The set of causal laws that is responsible for an error of particular type is narrowed down.

Note that these tasks might be carried out iteratively. Starting with the most likely error assumption obtained during error assessment, one might first check whether it is confirmed. If this is not the case, one may return to error assessment in order to obtain the next plausible error assumption and proceed with its confirmation. A designer may also skip the error assessment step, if she already suspects that a particular type of error is present due to her knowledge about the semantic meaning of the violated conditions.

In this paper, we focus on settings with a single error in an action description. We do this for both ground and non-ground action descriptions; for the latter, this amounts to multiple correlated errors at the ground level.

Error Types

By an *error*, we understand *a causal law in the action description that does not fulfill its intended meaning which exhibits itself in the violation of some condition(s).* According to a law's purpose, this may affect different aspects of the domain description serving as top-level error categories. They are further divided into the error types in Table 2.

Some errors may lead to a violation of general properties of the represented action domain, like whether basic actions may be executed *concurrently* or not (D.1). In some domains, state transitions may occur only if some basic action takes place, i.e., transitions with the 'empty action' $A = \emptyset$ (all action names are assigned false) are not allowed. We call such domains *event-driven* (D.2) as opposed to *time-driven* domains (D.3), which allow for transitions by $\emptyset$.

Sometimes an error causes a dynamic fluent property to be broken. A well-known important such property is *inertia* (F.1). In addition, we consider truth-value *persistency*, i.e., a fluent does not change its value anymore.

A next error category concerns action executability (X.5). It is refined into cases where, while concurrency is generally allowed, a particular action is not concurrently executable with another one (X.1); and where the preconditions for execution are violated (X.2). They are further specialized into the case of a non-concurrent (NC) basic action that should be unconditionally executable (X.3), respectively a conditionally executable non-concurrent basic action (X.4).

Finally, the (direct or indirect) effects of certain actions may be unintended. Special cases are if the action effect is deterministic (E.1) or non-deterministic (E.2).

Stage 1: Error Assessment

In this section, our goal is to identify an assumption on the type of error present in a given action description A that violates a given set of conditions Q (axioms expressed as queries). We first focus on the case of a single violated condition and later extend the heuristics towards determining prioritized error assumptions in case of multiple violations.

The *main problem is to match an arbitrary violated condition to a type of error.* In other words, we assume an uninformed user, who has no (or not enough) knowledge about the semantic properties expressed by the conditions (suppose she is not the designer of the action description) in order to come up with an assumption of the type of error.

We proceed by an analysis of the broken conditions. To this end, we identify conditions of four types:

C1: Conditions **necessarily** $False$ **after** ...

C2: Conditions **possibly** $True$ **after** ...

C3: Conditions **necessarily** ψ **after** ...

C4: Conditions **possibly** ψ **after** ...

In a first step, we need to know *which conditions are violated.* Hence, the user runs the test **T1** for each possibility query in Q and test **T2** for each necessity query in Q.

Violations of C4 conditions may be trivial (if no successor state exists) or non-trivial. To distinguish, we consider in addition the violations of the corresponding C2 condition, i.e., the condition where the head ψ is replaced with $True$. Thus, only the non-trivial violations of the original conditions will be taken into account as violations of a C4 condition.

The output of these tests is a list of violated conditions and the list of states and transitions violating these queries. This information is used to pinpoint a most likely error assumption as follows. We start with a single violated condition.

Violation of a condition by a transition $\langle s, \emptyset, s' \rangle$**,** i.e., with an empty action, that is legal. In this case, the error is usually w.r.t. some general fluent property (F). In case of a violated possibility query, the user may run test **T4** to find out whether the query failed due to underspecification. In this case, the error is most likely wrt. inertia (F.1) concerning the underspecified fluent reported by test **T4**. In case of no or multiple underspecified fluents, further domain knowledge is needed for a more detailed assumption.

Justification: In a time-driven domain, we can first focus on transitions $\langle s, \emptyset, s' \rangle$. If such a transition violates a condition, we can eliminate action executability and action effects as possible culprits, and what remains are errors with respect to dynamic fluent properties such as, e.g., inertia.

Violation of a C1 condition. There usually is an error with respect to action executability (X). More specifically, it either concerns concurrency (X.1) if there is just an action condition in the after-part of the condition, otherwise it concerns (non-) executability preconditions (X.2).

Justification: A condition of this type specifies that for particular states, there should not be a transition by certain actions. If this is violated, an action is executable while the condition requires its non-executability.

Violation of a C2 condition. The error is most likely wrt. action executability (X). If the condition is on a basic action with no fluents in the after-part, then it concerns an unconditional non-concurrent action (X.3). In case of fluents in the after-part, it is wrt. a conditional non-concurrent action (X.4); in the general case, the action is not basic (X.5).

Justification: Dual to a C1 condition, a C2 condition requests that in each state satisfying the precondition, a transition by the respective action exists. Hence, in case of failure certain actions are not executable, while they should be so.

Violation of a C3 or C4 condition. The error is with respect to action effects (E). If the violated condition is a necessity query, then the error concerns a deterministic action effect (E.1), otherwise the effect need not be deterministic and is classified as non-deterministic action effect (E.2).

Justification: A violation of a C3 respectively C4 condition occurs if a transition exists that satisfies the given preconditions, but leads to a state where ψ is false, respectively if no transition satisfying the preconditions to a state exists where ψ is true. Hence, C3 and C4 encode an expected action effect which is not observed in case of violation.

Example 2 In our running example, to determine whether the violation of (1) is trivial, we check for violations of the query with $\neg alive$ replaced with $True$. This query does not fail, however, since there is a compatible transition to s_2. Hence, we observe a C4 violation and assume an error with a (potentially) non-deterministic action effect (E.2). □

Heuristics for multiple violations. In reality, even a single error in the action description usually leads to violation of multiple conditions. In this case, we suggest a *heuristic ranking* of the error assumptions to identify the most likely culprit, which has been singled out in extensive experiments.

1. *Errors assumptions due to a violation of any condition w.r.t. the empty action.* They are most frequently correct, and ranked top since violations involving $A = \emptyset$ can not be fixed by repairing violations by non-empty actions.
2. *Error assumptions due to violations of C3 and C4 conditions.* They correspond to action effect errors (E). If a single error manifests itself by violations of both action effects (E) and action executability (X), an error assumption wrt. the former is ranked higher, for the following reason. An error concerning executability may mean that at some states, there is no compatible successor state. This cannot cause action effect violations as described by necessity queries, since they deal with existing transitions only. In case of an error wrt. action effects, however, some transitions may need to be removed since they cause inconsistency (which in turn may cause executability violations).
3. *Error assumptions due to violations of C2 conditions.* They indicate an error wrt. action executability, which is the most likely in absence of assumptions at rank 1 and 2.
4. *Error assumptions due to violations of C1 conditions.* These conditions encode non-executability for some actions. They are ranked last due to empirical evidence. The respective error assumptions were seldom appropriate.

Example 3 Consider another representation of the Yale Shooting, where the causal laws d_2 and d_3 in Table 1 are replaced by the respective laws

$$d'_2: \ \textbf{caused}\ \neg alive\ \textbf{after}\ shoot, loaded$$
$$d'_3: \ \textbf{caused}\ \neg loaded\ \textbf{after}\ \neg shoot$$

Then, not only the C1 condition (6) fails (e.g., $\langle s_1, \{load, shoot\}, s_3 \rangle$ is a valid transition), but also the query

$$\textbf{possibly}\ loaded\ \textbf{after}\ \neg shoot, \neg load\ \textbf{if}\ loaded. \quad (7)$$

The latter is a failure with respect to the empty action and is ranked higher. Therefore, we assume an error wrt. a general fluent property (F). Although the query does not fail due to underspecification, by inspection of the condition, even if we are not a domain expert, we might suspect an error wrt. inertia (F.1) concerning the fluent $loaded$. □

Stage 2: Assumption Confirmation

Once the designer has an assumption about the type of the error present in the action description—obtained by her knowledge about the domain, or by the heuristics from the previous section, etc—she may next *confirm the plausibility of this assumption*. For each type of error identified above (see Table 2), we provide a heuristics for that in terms of a recipe, given by certain tests on the action description together with corresponding input. The heuristics is summarized in Table 3 with the following notation for inputs: L is a fluent literal, A and B (partial) actions, A_i a basic action, and c, c_i a conjunction of fluent literals.

An assumption is confirmed, if the corresponding tests produce some output (witnesses, i.e, states or transitions, providing further details) for the given inputs.

Example 4 In order to confirm the error assumption in Example 2, we provide the action $A = \{shoot, \neg load\}$, the precondition $c_1 = loaded$, and the effect $c_2 = \neg alive$ under suspicion as inputs and run test **T1** with the query for error type E.2 from Table 3. Since the output is non-empty (cf. Example 1), the assumption is confirmed.

In Example 3, we get confirmation (two violating transitions) with the receipe for F.1 and input $L = loaded$. □

Type	Inputs	Recipe
D.1	None	**T2: necessarily** $False$ **after** A_i, A_j[*]
D.2	None	**T2: necessarily** $False$ **after** $A_\emptyset$[†]
D.3	None	**T1: possibly** $True$ **after** $A_\emptyset$
F.1	L	**T2: necessarily** L **after** $A_\emptyset$ **if** L
F.2	L	**T2: necessarily** L **if** L
X.1	A, B	**T2: necessarily** $False$ **after** A, B
X.2	A, c	**T2: necessarily** $False$ **after** A **if** c
X.3	A_i	**T1: possibly** $True$ **after** A_{A_i}[‡]
X.4	$A_i, c_1 \vee \cdots \vee c_k$	**T1: possibly** $True$ **after** A_{A_i} **if** c_j[⋆]
X.5	A, c	**T1: possibly** $True$ **after** A **if** c
E.1	A, c_1, c_2	**T2: necessarily** c_2 **after** A **if** c_1
E.2	A, c_1, c_2	**T1: possibly** c_2 **after** A **if** c_1

[*] $1 \leq i < j \leq n$ [†] $A_\emptyset = \neg A_1, \dots, \neg A_n$ [⋆] $1 \leq j \leq k$
[‡] $A_{A_i} = \neg A_1, \dots, \neg A_{i-1}, \neg A_{i+1}, \dots, \neg A_n$

Table 3: Heuristics for error confirmation.

Stage 3: Error Localization

There are several possibilities for how to proceed with localization, and in general it will not be possible to pinpoint an error to a specific causal law, let alone suggesting a particular repair; cf. (Eiter *et al.* 2006a) for an automatic repair method and discussion. We developed a heuristics for pruning the search space which is based on empirical studies. For space reasons, we just summarize the general ideas.

We assume that the error is in the head or body of a causal law, and maintain two sets H and B of causal laws whose heads, respectively bodies, might contain the error which causes a condition to fail. Initially, both sets contain all causal laws of the action description.

We then consider the output of the violated condition (of highest rank), i.e., violating transitions in case of a necessity query and violating states together with the action condition of the query in case of a possibility query: (a) We keep laws that fire wrt. all these situations and in B additionally laws that fire in none of them. (b) In case of a possibility query, we check for underspecification (test **T4**) to further reduce B. (c) Finally, we take the error type into account to shrink B further in some cases; e.g., respecting that deterministic action effects (E.1) are mostly encoded as direct effects, etc.

Example 5 Continuing Example 2, in Step (a) all laws that fire wrt. s_1 and the after part of condition (1) constitute $H = \{i_1, i_3, d_3\}$, while B contains all laws. Step (b) reduces B to laws incompatible with the underspecified transition having $\neg alive$ in the head, i.e., $B = \{i_2, d_2\}$. Realizing that the inertia laws are as intended, one ends up with the dynamic laws d_3, d_2; one of them (d_2) contains indeed the error.

As for Example 3, Step (a), considering the violations of (7), yields $H = \{i_3, d_3'\}$ and $B = \{i_3, i_4, d_1, d_2', d_3', d_4\}$. Step (b) will reduce B to $\{i_3, d_3'\}$ since the condition does not fail due to underspecification. Hence, we end up with two culprits, one of which is easily identified as correct. □

Experimental Results

We used the following four action domains for experiments:

Error assumption	Yale	Gofish	Blocks	Shell
Total	6 (7)	7 (9)	6 (7)	7
Expected & correct	4	6 (7)	4 (5)	5
Unexpected & correct	2	1	1	1
Incorrect	0 (1)	0 (1)	0	0
No error	0	0	1	1

Table 4: Classification results.

Yale: The well-known Yale Shooting Problem, as a deterministic, non-concurrent domain.

Gofish: A postal service domain including quality control features (mail tracking, customer notifications etc), with non-concurrent but sometimes non-deterministic actions.

Blocks: The well-known Blocks World with three blocks and concurrent but deterministic move actions.

Shell: A representation of the so-called Shell Game (three shells and a pea, aka Thimblerig, the old army game) with an action for swapping the position of two shells and a non-deterministic concurrent action $cheat$.

For all but Yale we considered also non-ground (parameterized) versions; the domain sizes are listed in Table 5. Note that all domains are time-driven; however, detecting the violation of event-drivenness is simple and easily fixed, so we did not consider it relevant for experiments.

In total, we made 53 experiments, of which 30 were on ground domains and 23 on non-ground domains. In each experiment, one causal law was blindly (by third party) modified in the domain to introduce a (not random but typical) error, and then we used our methodology to classify errors. To check for robustness, also some cases (7) were included that violated the general assumptions, e.g., an error in a static law or deletion of a law; in four cases, the change turned out to be immaterial, as the action diagram did not change.

A successful run through all stages of the process required, depending on the specific localization procedure, 5 tests on average, if the initial error assumption was confirmed, and one further test otherwise.

Error Classification. In most cases, the error assumption according to the heuristics was correct, however in some examples the result was unexpected. By unexpected we mean an error assumption different from the error assumption that had been expected (by third party) when the error was introduced, but was also correct. For instance, considering the error introduced in Example 3, cf. law d_3', without taking the failing conditions into account, one would rather expect an error wrt. action executability (E). Hence the confirmed error assumption F.1 is unexpected, although correct since, as we have seen, the inertia of $loaded$ is broken.

Taking only those examples into account, where none of the general assumptions were broken and the modification was not immaterial, we got correct and expected error assumptions in 34 out of 42 examples, approx. 81%. In the remaining examples, the determined error assumptions were also correct yet unexpected ($\sim$19%), see Table 4. The rate of success depends of course on the domain representation,

	# causal laws	# fluents+actions	$\lvert B \cup H \rvert$
Yale	8	2+2	3.666 (46%)
Gofish	86	6+7	5.143 (6.0%)
ground	184	46+35	6.714 (3.6%)
Blocks	9	3+2	5.6 (62%)
ground	236	25+13	55 (23%)
Shell	19	3+2	5.333 (28%)
ground	521	6+17	20.5 (3.9%)

Table 5: Localization results.

i.e., the domain description and conditions at hand. Error detection and classification is more effective if functionalities (e.g., action effect, executability, etc.) are clearly separated and expressed by dedicated dynamic laws and conditions. In small and/or simple domains, such as the Yale Shooting, most laws serve multiple purposes, and unexpected error assumptions were encountered more often.

Also note that we did not observe differences between the ground and non-ground versions, except for one case where, however, the general assumptions were broken as well. An error was introduced in the AD-query specific typing information for variables in the background knowledge, which is needed for grounding, and could not be equally introduced to the non-ground domain description. Eventually, we remark that even in the seven cases where general assumptions were broken we just got two wrong classifications, while five cases yielded an unexpected but correct error assumption

Localization. The results are summarized in Table 5. It shows the absolute and relative numbers of laws (wrt. the total number of laws in the domain) that remained as possible culprits in the sets B and H. The figures are averaged over all test domains which did not violate the general assumptions on the error.

In general, localization reduced the search space significantly and never missed the erroneous law. For ground domain descriptions it was significantly more effective unless the domain was very small (Yale). In most cases, it reduced to a set of laws which can be easily checked manually for the error. For space reasons, we omit a detailed discussion on differences in the non-ground setting. Intuitively, the criterion of a law firing wrt. all violating (ground) situations is less applicable (and would have to be considered wrt. parametric representations of relevant situations). In cases where the general assumptions were broken, localization failed.

We observed a remarkable difference in the success rate of localization depending on the type of condition used (according to Table 5). In case of possibility queries, we often ended up with just one or two laws. This was mainly due to localization steps which are based on underspecified transition candidates, and thus only apply to possibility queries. For necessity queries, localization was less effective; for C1 conditions, sometimes no law was pruned. This intuitively explains the deviation for the Blocks domain, which was the only domain with C1 violations ranked highest.

Related Work and Conclusion

We have presented a methodology, based on heuristics, to form and test assumptions about possible errors in an action description according to an error taxonomy. Different from this is a model-based approach, which incorporates error assumptions formally into reasoning about the action domain description. Here, Balduccini and Gelfond (2003) and Baral *et al.* (2000) considered diagnostic reasoning for domains in an action language. Their concern, however, was to explain unexpected behavior rather than to debug a domain description itself, relying on a model of abnormality in it (which is not needed in our approach, and moreover, error-prone itself). Simply suspending causal laws that are assumed to be wrong for restoring consistency does not work, as action descriptions are non-monotonic theories. On the other hand, a naive approach to consider all possible repairs for an error quickly becomes infeasible due to a huge search space (as in parametric domains like Gofish, Blocks, and Shell).

As for future work, (Eiter *et al.* 2006a) described further tests that refer to the necessity or possibility of fixing an error by modifying or keeping specific dynamic laws. Such tests can be incorporated into our methodology. Another issue is extending our methodology to multiple uncorrelated errors, which is not obvious, and respecting static laws.

An interesting issue concerns testing of parametric action domains, in which causal laws are stated generically for a sort of objects, and the actual set of objects is a parameter (e.g., the blocks in a blocks world). A small set of objects is usually sufficient to assess soundness of an action description wrt. a given set of queries; techniques to determine such small test sets would be useful.

Acknowledgments This work was supported by the Austrian Science Fund (FWF) under grant P16536-N04.

References

Balduccini, M., and Gelfond, M. 2003. Diagnostic reasoning with A-prolog. *TPLP* 3(4-5):425–461.

Baral, C.; McIlraith, S. A.; and Son, T. C. 2000. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Proc. KR '00*, 311–322.

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2006a. Resolving conflicts in action descriptions. In *Proc. ECAI 2006*, 367–371. IOS Press.

Eiter, T.; Fink, M.; and Senko, J. 2006b. A tool for answering queries on action descriptions. In *Proc. JELIA 2006*, LNCS 4160, 473–476. Springer.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electronic Trans. Artificial Intelligence* 3(16):195–210.

Giunchiglia, E., and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, 623–630. AAAI Press.

Grell, S.; Schaub, T.; and Selbig, J. 2006. Modelling biological networks by action languages via answer set programming. In *Proc. ICLP 2006*, LNCS 4079, 285–299.

Watson, R., and Chintabathina, S. 2003. Modeling hybrid systems in action languages. In *Proc. ASP 2003*, 356-370.

Zepeda, C.; Osorio, M.; and Sol, D. 2005. Modeling evacuation planning using A-prolog. In *CONIELECOMP*, 292–297. IEEE Computer Society.