Addressing the Qualification Problem in FLUX

Yves Martin and Michael Thielscher

Dresden University of Technology

Abstract. The Qualification Problem arises for planning agents in real-world environments, where unexpected circumstances may at any time prevent the successful performance of an action. We present a logic programming method to cope with the Qualification Problem in the action programming language FLUX, which builds on the Fluent Calculus as a solution to the fundamental Frame Problem. Our system allows to plan under the default assumption that actions succeed as they normally do, and to reason about these assumptions in order to recover from unexpected action failures.

1 Introduction

Intelligent agents in open environments inevitably face the Qualification Problem: The executability of an action can never be predicted with absolute certainty; at any time, actions in the real-world may surprisingly fail [13]. Yet it would be irrational, and even impossible in general, for a planning agent to foresee all conceivable reasons for an action to go wrong. Rather, a rational agent needs to devise plans under the assumption that the world will behave as expected. On the other hand, being aware of these assumptions helps an agent to explain and recover from unexpected failures encountered during the execution of a plan.

For a long time, the main theoretical result on the Qualification Problem had been a negative one: While a solution must involve the ability to assume away, by default, so-called abnormal qualifications of actions [14], straightforward minimization of abnormality yields anomalous models [9]. This problem being unsolved, previously developed action programming languages and planning systems, such as [8, 16, 12, 2], did not attempt to address the Qualification Problem. The problem of anomalous models has, however, recently been solved in a formal account of the Qualification Problem presented in [21]. This theory builds on the Fluent Calculus as a predicate logic formalism for reasoning about actions which is one of the standard solutions to the fundamental Frame Problem [18].

In this paper, we integrate the theoretical account of the Qualification Problem into the action programming language FLUX (the *Fluent Calculus Executor*) [20]. Based on constraint logic programming, FLUX allows to specify and reason about actions with incomplete states, and thus to solve planning problems under incomplete information. Its core consists of a logic programming account of the Fluent Calculus solutions to the Frame and Ramification Problem [19, 17]. Extending FLUX so as to cope with the Qualification Problem, our system allows

the user to specify default assumptions concerning the executability and effects of actions. Plans are then generated under these assumptions, and the system is able to reason about the assumptions made and to withdraw appropriate ones in order to explain and recover from unexpected action failures. The language allows to distinguish between strong qualifications (actions not being executable) and weak ones (actions producing unexpected effects). Furthermore, it supports the specification of preferences among the default assumptions, by which is aided the search for reasonable explanations in case of unexpected action failure.

The paper is organized as follows. In Section 2, we recapitulate the theoretical account of the Qualification Problem in the Fluent Calculus. In Section 3, we show how to extend FLUX to cope with the Qualification Problem. For a more detailed discussion of this section the reader is referred to [11]. In Section 4 an application is described and Section 5 gives a summary.

2 The Qualification Problem in the Fluent Calculus

2.1 Simple state update axioms

The simple Fluent Calculus [19] combines, in pure classical logic, the Situation Calculus with a STRIPS-like solution to the representational and inferential Frame Problem. The standard sorts ACTION and SIT (i.e., situations) are inherited from the Situation Calculus [7] along with the standard functions $S_0 : \text{SIT}$ and $Do : \text{ACTION} \times \text{SIT} \mapsto \text{SIT}$ denoting, resp., the initial situation and the successor situation after performing an action; furthermore, the standard predicate $Poss : \text{ACTION} \times \text{SIT}$ denotes whether an action is possible in a situation. To this the Fluent Calculus adds the sort STATE with sub-sort FLUENT along with the pre-defined functions $\emptyset : \text{STATE}$, $\circ : \text{STATE} \times \text{STATE} \mapsto \text{STATE}$, and $State : \text{SIT} \mapsto \text{STATE}$, denoting, resp., the empty state, the union of two states, and the state of the world in a situation. Based on this signature, the Fluent Calculus provides a rigorously logical account of the concept of a state being characterized by the set of fluents that are true in the state. To this end, the following foundational axioms stipulate that function $\circ$ behaves like set union with $\emptyset$ as the empty set:[1]

$$
\begin{array}{rl}
z_1 \circ (z_2 \circ z_3) = (z_1 \circ z_2) \circ z_3 & \neg Holds(f, \emptyset) \\
z_1 \circ z_2 = z_2 \circ z_1 & Holds(f_1, f) \supset f = f_1 \\
z \circ z = z & Holds(f, z_1 \circ z_2) \supset Holds(f, z_1) \vee Holds(f, z_2) \\
z \circ \emptyset = z & (\forall f)\,(Holds(f, z_1) \equiv Holds(f, z_2)) \supset z_1 = z_2 \\
 & (\forall \Phi)(\exists z)(\forall f)\,(Holds(f, z) \equiv \Phi(f))
\end{array}
$$

where Φ is a second-order predicate variable of sort FLUENT and the macro *Holds* means that a fluent is part of a state:

$$Holds(f, z) \stackrel{\text{def}}{=} (\exists z')\, z = f \circ z' \tag{1}$$

[1] Free variables in formulas are assumed universally quantified. Variables of sorts ACTION, SIT, FLUENT, and STATE shall be denoted by the letters a, s, f, and z, resp. The function $\circ$ is written in infix notation.

The very last one of the axioms above stipulates the existence of a state for all possible combinations of fluents. A second macro, which reduces to (1), is used for fluents holding in situations:

$$Holds(f, s) \stackrel{\text{def}}{=} Holds(f, State(s))$$

As an example, consider a blocks world axiomatization using the FLUENT functions $On(x, y)$, $GluedToTable(x)$, and $Has(r, x)$ denoting, resp., whether block x is on y (which could either be another block or the constant $Table$), whether block x is glued to the table, and whether robot r is in possession of x. Suppose, that in the initial state it is known that blocks A and C are on the table and B is on C; that no block y is on top of block A or B; and that robot $Robbie$ is in possession of glue:

$$\begin{array}{l} Holds(On(A, Table), S_0) \wedge Holds(On(C, Table), S_0) \wedge Holds(On(B, C), S_0) \\ \wedge\ (\forall y)\,(\neg Holds(On(y, A), S_0) \wedge \neg Holds(On(y, B), S_0)) \\ \wedge\ Holds(Has(Robbie, Glue), S_0) \end{array} \tag{2}$$

Assuming uniqueness of names for all functions with range FLUENT, the macro definitions and the foundational axioms imply that (2) is equivalent to,

$$\begin{array}{l} (\exists z)\,(\,State(S_0) = On(A, Table) \circ On(C, Table) \\ \qquad\qquad \circ On(B, C) \circ Has(Robbie, Glue) \circ z \\ \qquad \wedge\ (\forall y)\,(\neg Holds(On(y, A), z) \wedge \neg Holds(On(y, B), z)) \\ \qquad \wedge\ \neg Holds(On(A, Table), z) \wedge \neg Holds(On(C, Table), z) \\ \qquad \wedge\ \neg Holds(On(B, C), z) \wedge \neg Holds(Has(Robbie, Glue), z)\,) \end{array} \tag{3}$$

The reader may notice that the constraints on sub-STATE z not only reflect the negated statements in (2) but also the fact that the fluents $On(A, Table)$ etc. do not recur. This will allow to quickly infer the result of removing any of these fluents from $State(S_0)$ as a negative effect.

The Frame Problem is solved in the Fluent Calculus using so-called state update axioms, which specify the difference between the states before and after an action. The axiomatic characterization of negative effects, i.e., facts that become false, is given by this inductive abbreviation, which generalizes STRIPS-style update [3] to incomplete states:

$$z' = z - f \stackrel{\text{def}}{=} [z' \circ f = z \vee z' = z] \wedge \neg Holds(f, z')$$

$$\begin{array}{l} z' = z - (f_1 \circ \ldots \circ f_n \circ f_{n+1}) \stackrel{\text{def}}{=} \\ \quad (\exists z'')\,(z'' = z - (f_1 \circ \ldots \circ f_n) \wedge z' = z'' - f_{n+1}) \end{array}$$

On this basis, the following is the general form of a state update axiom for a (possibly nondeterministic) action $A(\vec{x})$ with a bounded number of (possibly conditional) effects:

$$\begin{array}{l} Poss(A(\vec{x}), s) \supset (\exists \vec{y}_1)\,(\Delta_1 \wedge State(Do(A(\vec{x}), s)) = (State(s) \circ \vartheta_1^+) - \vartheta_1^-) \\ \qquad\qquad \vee \ldots \vee \\ \qquad\qquad (\exists \vec{y}_n)\,(\Delta_n \wedge State(Do(A(\vec{x}), s)) = (State(s) \circ \vartheta_n^+) - \vartheta_n^-) \end{array} \tag{4}$$

where the sub-formulas $\Delta_i(\vec{x}, \vec{y}_i, State(s))$ specify the conditions on $State(s)$ under which $A(\vec{x})$ has the positive and negative effects ϑ_i^+ and ϑ_i^-, resp. Both ϑ_i^+ and ϑ_i^- are STATE terms composed of fluents with variables among $\vec{x}, \vec{y}_i$.[2]

Consider, e.g., the ACTION terms $Move(r, u, v, w)$ and $GlueToTable(r, x)$, denoting the action of robot r moving block u away from v onto w, and gluing block x to the table, resp. The direct effects of these two actions can be defined by these state update axioms:

$$\begin{array}{l} Poss(Move(r, u, v, w), s) \supset \\ \quad State(Do(Move(r, u, v, w), s)) = (State(s) \circ On(u, w)) - On(u, v) \\ Poss(GlueToTable(r, x), s) \supset \\ \quad State(Do(GlueToTable(r, x), s)) = State(s) \circ GluedToTable(x) \end{array} \tag{5}$$

Put in words, after moving u it is on w and no longer on v, and after gluing x this block is glued to the table. Recall specification (2), and suppose, for the sake of argument, that $Poss(Move(Robbie, A, Table, B), S_0)$. Let $S_1 = Do(Move(Robbie, A, Table, B), S_0)$. Then the state update axiom for $Move$ in (5) implies

$$State(S_1) = (State(S_0) \circ On(A, B)) - On(A, Table)$$

Replacing $State(S_0)$ by an equal term according to (3) yields, after applying the macro for negative effects and performing simplification,

$$(\exists z)\, State(S_1) = On(C, Table) \circ On(B, C) \circ Has(Robbie, Glue) \circ z \circ On(A, B)$$

We have now obtained from an incomplete initial specification a still partial description of the successor state, which in particular includes the unaffected fluents $On(C, Table)$, $On(B, C)$, and $Has(Robbie, Glue)$. These fluents have thus survived the computation of the effect of the action and so need not be carried over by separate axioms now. Moreover, knowledge specified in (3) as to which fluents do not hold in z applies to the new state, which includes z, just as well. Thus, all unchanged fluent values have been concluded to persist without applying extra inference steps.

2.2 State Update Axioms with Ramifications

In the Fluent Calculus for ramifications, indirect effects are inferred by the successive application of so-called causal relationships, which state under what conditions an effect triggers another one [17]. A causal relationship is formally specified with the help of the expression $Causes(\varepsilon, \varrho, z, s)$ where ε (the *triggering effect*) and ϱ (the *ramification*, i.e., indirect effect) are possibly negated atomic fluent formulas and z is a state and s a situation. The intuitive meaning is that the change to ε *causes* the change to ϱ in state z and situation s.

For example, let $\mathsf{Ab}(Movable(x), Glued)$ be a new FLUENT, denoting an abnormality wrt. block x being movable due to the fact that x is glued to the

[2] If the conditions Δ_i are not mutually exclusive, then the action is nondeterministic.

table.[3] The following *state constraint* relates this fluent in the obvious way to the fluent $GluedToTable(x)$:

$$Holds(\mathsf{Ab}(Movable(x), Glued), s) \equiv Holds(GluedToTable(x), s) \tag{6}$$

Two accompanying causal relationships specify the causal dependence that a block x will become immovable if it gets glued to the table, and that this abnormal qualification will disappear if the block gets freed somehow:

$$\begin{array}{l} Causes(GluedToTable(x), \mathsf{Ab}(Movable(x), Glued), z, s) \\ Causes(\neg GluedToTable(x), \neg\mathsf{Ab}(Movable(x), Glued), z, s) \end{array} \tag{7}$$

On the basis of causal relationships, the Ramification Problem is solved by causally propagating indirect effects: Starting from the direct effects of an action, causal relationships are applied successively. The overall result of performing the action is then a fixpoint of such a chain of indirect effects. Formally, in state update axioms for ramifications the simple equations $State(Do(A(\vec{x}), s)) = (State(s) \circ \vartheta_i^+) - \vartheta_i^-$ as in (4) are replaced by sub-formulas of this form:

$$z = (State(s) \circ \vartheta_i^+) - \vartheta_i^- \supset Ramify(z, \vartheta_i^+, \vartheta_i^-, Do(A(\vec{x}), s))$$

where $Ramify(z, e^+, e^-, s)$ means that $State(s)$ is a fixpoint of iteratively applying causal relationships to state z and effects e^+, e^- in situation s. (We refer to [21] for the formal definition of *Ramify* by a second-order axiom.)

2.3 Qualifications in the Fluent Calculus

The theoretical account of the Qualification Problem uses the binary function $\mathsf{Ab}(x, y)$ whose range is the sort FLUENT. The first argument, x, denotes properties like $Movable(u)$ or $Functioning(Gripper\text{-}of(r))$. The second argument, y, indicates the cause for the abnormality. For convenience, we use the macros $\mathsf{Ab}(x, z)$ and $\mathsf{Ab}(x, s)$ to represent that for some y, $\mathsf{Ab}(x, y)$ holds in state z and situation s, respectively:

$$\mathsf{Ab}(x, z) \stackrel{\text{def}}{=} (\exists y)\, Holds(\mathsf{Ab}(x, y), z) \qquad \mathsf{Ab}(x, s) \stackrel{\text{def}}{=} \mathsf{Ab}(x, State(s))$$

Instances of the generic 'abnormality' fluent are used to summarize the abnormal qualifications of actions, that is, obstacles which are *a priori* unlikely to happen and therefore need to be assumed away by default in order to jump to the conclusion that the action is possible under normal circumstances. E.g., in the light of abnormal qualifications, the preconditions for our example actions are specified by,

$$\begin{array}{l} Poss(Move(r, u, v, w), s) \equiv \\ \quad u \neq w \wedge v \neq w \wedge Holds(On(u, v), s) \\ \quad \wedge\ (\forall y)\,(\neg Holds(On(y, u), s) \wedge \neg Holds(On(y, w), s)) \\ \quad \wedge \neg\mathsf{Ab}(Movable(u), s) \wedge \neg\mathsf{Ab}(Functioning(Gripper\text{-}of(r)), s) \end{array} \tag{8}$$

[3] The special fluent Ab will play a key role in our account for the Qualification Problem later in this paper.

$$\begin{aligned} &Poss(GlueToTable(r,x),s) \equiv \\ &\quad Holds(Has(r,Glue),s) \wedge Holds(On(x,Table),s) \wedge (\forall y)\,\neg Holds(On(y,x),s) \\ &\quad \wedge \neg \mathsf{Ab}(Movable(x),s) \wedge \neg \mathsf{Ab}(Functioning(Gripper\text{-}of(r)),s) \end{aligned} \tag{9}$$

As illustrated in the previous section, abnormal qualifications that have been caused by the agent himself are accounted for by suitable causal relationships, which is how the general problem of anomalous models is overcome; see [21]. To account for abnormal qualifications other than those caused by the agent, instances of Ab are allowed to become true during any situation transition as a side effect of the mere fact that the very transition takes place. So doing requires additional causal relationships, which, as opposed to those shown in (7), describe exogenously caused abnormalities. These are modeled using the predicates $ExogCaused(f,s)$ and $ExogUncaused(f,s)$, indicating that in situation s fluent f arises (resp. vanishes) due to an exogenous cause. The effect of exogenous causes is specified by corresponding causal relationships.

Up to this point the treatment of qualifications did not affect the monotonicity of the solution to the Frame and Ramification Problem. A nonmonotonic component, however, is required to *minimize* abnormal qualifications whenever they are not caused by an action that has been performed. This is achieved by adding appropriate default rules in the sense of [15], by which the Fluent Calculus gets embedded into a default theory. Formally, exogenous influence on abnormalities is minimized by default rules of the following form:

$$\frac{:\ \neg ExogCaused(\mathsf{Ab}(x,Exog),s)}{\neg ExogCaused(\mathsf{Ab}(x,Exog),s)} \qquad \frac{:\ \neg ExogUncaused(\mathsf{Ab}(x,Exog),s)}{\neg ExogUncaused(\mathsf{Ab}(x,Exog),s)} \tag{10}$$

An accompanying default assumption concerns abnormalities of any kind in the *initial* situation. Their minimization is carried out by defaults of the following form:

$$\frac{:\ \neg Holds(\mathsf{Ab}(x,y),S_0)}{\neg Holds(\mathsf{Ab}(x,y),S_0)} \tag{11}$$

If, e.g., the observations suggest no abnormalities initially, then the underlying default theory has a unique extension (in the sense of [15]), which includes $(\forall x)\,\neg\mathsf{Ab}(x,S_0)$. (Recall that $\neg\mathsf{Ab}(x,s)$ means $\neg Holds(\mathsf{Ab}(x,y),s)$ for any y.) Hence, (8) implies that $Move(Robbie,A,Table,B)$ is possible in S_0 given initial state (2). If this action nonetheless fails, that is, if the observation $\neg Poss(Move(Robbie,A,Table,B),S_0)$ is added, then the default theory admits different extensions. These are obtained by applying all defaults except for one instance of (11) with either $\{x/Movable(A)\}$ or $\{x/Functioning(Gripper\text{-}of(Robbie))\}$. As will be shown in Section 3.5 below, this approach can also be used to account for so-called weak qualifications, that is, unexpected effects of actions. Furthermore, by appealing to *prioritized* default logic [1], one can specify qualitative knowledge of the relative likelihood of the various explanations for abnormal qualifications (cf. Section 3.4 below). The accompanying concept of *preferred* extensions helps selecting the most reasonable explanations in case of unexpected action.

3 Addressing the Qualification Problem in FLUX

3.1 Basic FLUX

Our system is implemented in the Eclipse-Prolog system with constraints. It is an extension of the programming language Fluent Calculus Executor (FLUX), a recent implementation of the Fluent Calculus based on Constraint Logic Programming [20]. The distinguishing feature of FLUX is to support incomplete states, which are modeled by open lists of the form

```
Z0 = [F1,...,Fm | Z]
```

(encoding the state description $\texttt{Z0} = \texttt{F1} \circ \ldots \circ \texttt{Fm} \circ \texttt{Z}$), along with constraints

```
not_holds(F, Z)
not_holds_all([X1,...,Xk], F, Z)
```

encoding, resp., the negative statements $(\exists \vec{y})\, \neg \mathit{Holds}(\texttt{F}, \texttt{Z})$ (where $\vec{y}$ are the variables occurring in F) and $(\exists \vec{y})(\forall \texttt{X1}, \ldots, \texttt{Xk})\, \neg \mathit{Holds}(\texttt{F}, \texttt{Z})$ (where $\vec{y}$ are the variables occurring in F except $\texttt{X1}, \ldots, \texttt{Xk}$). These two constraints are used to bypass the problem of "negation as failure" for incomplete states. In order to process these constraints, so-called declarative Constraint Handling Rules [4] have been defined and proved correct under the foundational axioms of the Fluent Calculus. In addition, the core of FLUX contains definitions for `holds(F,Z)` , by which is encoded macro (1), and `update(Z1,ThetaP,ThetaN,Z2)` , which encodes the state equation $\texttt{Z2} = (\texttt{Z1} \circ \texttt{ThetaP}) - \texttt{ThetaN}$. The following is an encoding in FLUX of the precondition (8) and state update axiom (5) with ramifications of the action *Move* ,

```
poss(move(R, U, V, W), Z) :-
   is_robot(R), is_block(U), is_block(V), is_block(W), U\=W, V\=W,
   not_holds_all(Y, on(Y, U), Z), holds(on(U, V), Z),
   not_holds_all(Y, on(Y, W), Z),
   not_holds_all(Y, ab(mov(U), Y), Z),
   not_holds_all(Y, ab(func(grip(R)), Y), Z).

state_update(Z1, move(R,U,V,W), Z2, S, H) :-
   update(Z1, [on(U,W)], [on(U,V)], Z3),
   ramify(Z3, [on(U,W)], [on(U,V)], Z2, S, H)).
```

where the variable H stands for the history list as defined below.

3.2 Overview

Our system implements the defaults of the underlying default theory without the need of any special theorem prover. Rather, a modified version of the planning algorithm together with the internal Eclipse-Prolog inference mechanisms is used to construct the extensions of the underlying default theory, which entail the

possible explanations for unexpected action failures. This also means that if a plan can be executed without any exceptions to the normal execution of actions then no additional computations are needed. In this case, the implementation will behave like any other system without an approach to the Qualification Problem.

In this work, we have used a search process with an associated level in order to find the most likely explanations first. In terms of the underlying default theory (with priorities among defaults) this means to search for the least preferred default that does not apply. The system also replaces previously considered explanations in case a default is no longer preferred in context of a formerly established explanation. Of course, the replacement is only performed if it is consistent with the executed action sequence.

In the following we sketch the course of the program and give references to the next subsections:

1. **Definition of the Task**
   - 1.1 **Define the initial state.** This operation includes the verification that the state is consistent wrt. the state constraints.
   - 1.2 **Define the goal state.**
2. **The Planning Algorithm**
   - 2.1 **Find a plan.** State updates are computed with ramifications (cf. Section 2.2). An agent has no influence on exogenously caused abnormalities and cannot yet have caused any abnormal qualification in the initial situation. Therefore, all defaults on the absence of exogenous causes are assumed to apply during the planning process. Iterative deepening is applied as search strategy. Furthermore, the program uses some heuristics to cut down the search space.
   - 2.2 **Double-check the computed plan.** Planning with incomplete state information requires to verify an established plan against both, not achieving the goal and not being executable.
3. **Plan Execution**
   - 3.1 **Execute the computed plan step by step.** The execution of each action is monitored. If no action fails and all the actions achieve their intended effects then the goal state will be reached and the program terminates. Otherwise, the program proceeds with step 4.
4. **Explanation of and Recovery from Action Failures**
   - 4.1 **Search for an explanation for the unexpected observation.** If an action surprisingly fails or does not produce all the intended effects then an abnormal qualification must have occurred. This means that at least one default of the underlying default theory can no longer be applied. For the intended applications of the program it is in most cases sufficient to search for atomic explanations, i.e., where the application of exactly one default is blocked during the construction of each of the possible extensions. Furthermore, we assume that there is always at most one explanation at each level. Using the built-in inference mechanisms of the Eclipse-Prolog system, each extension of the underlying default theory is considered where one instance of a default rule is blocked. If

the non-application of such a default rule entails the observation then an explanation has been found and the search process stops (cf. Section 3.3).

4.2 **Find the explanation with the highest priority first.** Using a search process with levels, the search will find only explanations with a priority higher or equal to the current level of the search algorithm (cf. Section 3.4). Only if there is no such very likely explanation that accounts for the observation then the program searches with the next lower level and thus considers less likely explanations.

4.3 **Determine the current state and replan.** The search process of the program can deliver a strong or a weak qualification (cf. Section 3.5) as an explanation. In both cases, this new information is integrated into the current state, which becomes the initial state of the new planning problem. Then the planning algorithm is used to find a new plan despite the encountered abnormality. Hence, the program proceeds with step 2.

An important concept in the approach to the Qualification Problem in FLUX is the notion of a *history list*. Such a list contains all the abnormalities that have occurred so far during the execution of the program. Each entry in the list has three parts. The first part describes the abnormality predicate with the property and the cause. In the second part the situation, in which the abnormality occurred, is stated. The third part denotes the possible observation which lead to the occurrence of the abnormality. In addition to these entries, the history list contains the current level of the search process. The history list is modeled by an open list with a tail variable.

3.3 Constructions of Extensions

In this section we show how extensions of the underlying default theory are constructed for strong qualifications of actions. The inference process is similar for weak qualifications.

Effects with an exogenous cause are implemented by causal relationships of the following form:

```
causes(_, ab(mov(X), exog), Z, S, H) :- block(X),
 X\= table, exogcaused(ab(mov(X), exog), S, H).

causes(_, -(ab(mov(X), exog)), Z, S, H) :- block(X),
 X\=table, exoguncaused(-(ab(mov(X), exog)), S, H).
```

Please note that the indirect effects $\mathsf{Ab}(\mathit{Movable}(x), \mathit{Exog})$ in these clauses are not conditioned on any direct effect. Consequently, these positive or negative indirect effects occur as a side effect of every transition whenever the predicates $\mathit{ExogCaused}(\mathsf{Ab}(x, \mathit{Exog}), s, h)$ or $\mathit{ExogUncaused}(\mathsf{Ab}(x, \mathit{Exog}), s, h)$ hold, where the variables s and h stand for the considered situation and history list, respectively. Indirect effects wrt. the abnormality $\mathsf{Ab}(\mathit{Functioning}(\mathit{Gripper\text{-}of}(r)), \mathit{Exog})$ are similarly encoded.

Predicate *ExogCaused* and predicate *ExogUncaused* are specified in a similar way in our program. For brevity we present only the important details of the definition for *ExogCaused* .

```
exogcaused(AB, S, H) :-  ...
                         top(H, H2),
                         length1(H2,0),
                         holds(h(AB, S), H)))).
```

The clause uses the auxiliary predicates $Top(h_1, h_2)$ and $Length1(h, n)$. The predicate *Top* takes the present history list and yields the currently considered abnormality predicate, if any. The predicate *Length1* delivers the length of a list. The definition of the clause ensures that exactly one abnormality is considered as an explanation at any stage of the search process.

Extensions are constructed during state updates with ramifications. For each action all possible extensions of the underlying default theory are tried until the established extension accounts for the observation. This is achieved by blocking the application of each default one after the other. The *ExogCaused* or the *ExogUncaused* predicate is assumed to hold and the specific default is blocked by means of adding the corresponding abnormality as indirect effect to the current state. The addition is performed by the clauses for causal relationships together with the clauses for state update axioms with ramifications. The predicates *ExogCaused* and *ExogUncaused* ensure that only one default is blocked at a time, and the Eclipse-Prolog SLDNF-resolution with backtracking yields all the extensions. The possible defaults for the initial situation are computed in the same way. To this end, the special constant " ϵ " (read "no-op") is introduced. It denotes the empty action without any positive or negative direct effects. This action is always possible and is only executed as the very first action.

As illustration, consider the initial state as specified in (2) together with the following definitions for blocks and robots:

```
is_robot(robbie).          is_block(table). is_block(a).
                           is_block(b).     is_block(c).
```

The query $Init(z_0, h), Res(\epsilon, z_0, S_0, z_1, s_1, h)$, $\backslash{+}Poss(Move(Robbie, A, Table, B), z_1)$ admits two computed answer substitutions, where the predicate *Init* denotes the initial state and the predicate *Res* denotes the execution of exactly one action:

$$\{z_1/[\mathsf{Ab}(Movable(A), Exog), On(A, Table), On(B, C), On(C, Table), \\ Has(Robbie, Glue) \mid z],\ h/[H(\mathsf{Ab}(Movable(A), Exog), S_0) \mid h_1]\}$$

$$\{z_1/[\mathsf{Ab}(Functioning(Gripper\text{-}of(Robbie)), Exog), On(A, Table), On(B, C), \\ On(C, Table), Has(Robbie, Glue) \mid z], \\ h/[H(\mathsf{Ab}(Functioning(Gripper\text{-}of(Robbie)), Exog), S_0) \mid h_1]\}$$

These substitutions are computed by applying all defaults except for one instance of (11) with $\{x/Movable(A)\}$ for the first substitution and $\{x/Functioning($

Gripper-of(*Robbie*))} for the second one. This way, our program determines the two extensions in this example as described at the end of Section 2.3.

Extensions are constructed using the inference mechanism of the Eclipse-Prolog system, i.e., an implementation of the SLDNF-resolution. This resolution scheme is sound. For the soundness proof it should be referred to Lloyd [10].

In our implementation extensions are constructed using the 9-ary predicate *RunDefault* . We present a schematic version of its encoding:

```
rundefault([], Z, Z, S, S, SW, H1, H2, R) :-
 \+ poss(R, Z), ...
rundefault([F|L], Z0, Z, S, SF, SW, H1, H2, R) :-
 append([A], [E], F), !, ...
 res(A, Z0, S, Z1, S1, H1), ...
 current(S1, H1, HH), (HH=[]; HH=[h(_,_,o(M,_))], \+ poss(M, Z1)),
 ...
 rundefault(L, Z2, Z, S1, SF, SW, H1, H2, R).
```

The predicate *RunDefault* has a recursive definition. The recursion is performed on its first argument. This argument represents the executed sequence of actions as a list. The last argument of *RunDefault* stores the current observation. It is the action that failed unexpectedly for the reason of a strong abnormal qualification. Thus, for the predicate to terminate successfully the computed explanation must account for the observation after having performed the complete sequence of actions. Of course, all other observations recorded in the history list must also be taken into account during the search. To this end, the auxiliary predicate $Current(s, h_1, h_2)$ is used. It checks for the current situation of the search process, whether an unexpected action failure has occurred in this situation during the execution of the plan. If this is the case then the predicate *Current* delivers the corresponding action. Otherwise, the empty list is returned.

3.4 Selection of the Preferred Extension

Extensions are constructed in accordance with the underlying set-prioritized default logic, which is an extension of the prioritized default logic and can be used to define preferences between defaults [1, 21]. The preference relations in the program are defined with or without context-dependency. The first case defines a set-preference ordering among defaults so that any two instances of a default concerning the same object are compared to an instance of another default concerning the object. If there is no context of a previous explanation then the second case gives a definition of general preference between defaults. As illustration, consider the implementation for the running example together with the following preference relations:

```
level([12,11,10,9]).

preference(ab(mov(_), _), nc,  9).
preference(ab(func(grip(_)), _), nc,  10).
preference(ab(mov(_), _), ab(func(grip(_)), _), 11).
```

That is, without context the explanation $\mathsf{Ab}(\mathit{Functioning}(\mathit{Gripper\text{-}of}(x)), y)$ is more likely than the explanation $\mathsf{Ab}(\mathit{Movable}(x), y)$. In contrast, the explanation $\mathsf{Ab}(\mathit{Movable}(x), y)$ is preferred over the assumption of two explanations of the form $\mathsf{Ab}(\mathit{Functioning}(\mathit{Gripper\text{-}of}(x)), y)$.

The preference relations are taken into account when using the predicates *ExogCaused* and *ExogUncaused*. The following shows the part of the clause without context information for the predicate *ExogCaused*:

```
exogcaused(AB, S, H) :-  ...
                         preference(AB, nc, P),
                         member(l(D), H), !, P>=D,
                         top(H, H2), ...
```

The priority of the currently considered explanation is obtained. Afterwards, the current level of the search is obtained from the history list. Further on, the computed priority of the considered explanation is compared to this level. If the priority is at least as high as the current level then the procedure continues as described in Section 3.3.

The change to the next lower level in the search is encoded as:

```
..., level(LEVEL), member(LE, LEVEL), changed(LE, H, H3),
rundefault(L2, ZN, ZF, s0, SV, SW, H3, HF, W), ...
```

The auxiliary predicate *ChangeD* sets the current level in the history list. If the predicate *RunDefault* fails then the Eclipse-Prolog system backtracks and the predicate *Member* chooses the next lower level for the search process.

For example, consider the query $\mathit{Init}(z_0, h)$, $\mathit{Level}(\mathit{level})$, $\mathit{Member}(\mathit{le}, \mathit{level})$, $\mathit{ChangeD}(\mathit{le}, h, h_1)$, $\mathit{Res}(\epsilon, z_0, S_0, z_1, s_1, h_1)$, $\backslash{+}\mathit{Poss}(\mathit{Move}(\mathit{Robbie}, A, \mathit{Table}, B), z_1)$, where the initial state is specified as in (2) and the preferences are defined as above. All preferred extensions of the underlying default theory entail $\mathsf{Ab}(\mathit{Functioning}(\mathit{Gripper\text{-}of}(\mathit{Robbie})), S_0)$. In accordance with the preferred extension the query yields the computed answer substitution:

$$\begin{array}{l}\{z_1/[\mathsf{Ab}(\mathit{Functioning}(\mathit{Gripper\text{-}of}(\mathit{Robbie})), \mathit{Exog}), \mathit{On}(A, \mathit{Table}),\\ \quad \mathit{On}(B, C), \mathit{On}(C, \mathit{Table}), \mathit{Has}(\mathit{Robbie}, \mathit{Glue}) \mid z],\\ \; h/[\mathit{le}(10), H(\mathsf{Ab}(\mathit{Functioning}(\mathit{Gripper\text{-}of}(\mathit{Robbie})), \mathit{Exog}), S_0) \mid h_1]\}\end{array}$$

3.5 Weak Qualifications

Weak qualifications, that is, failure to produce expected effects, are denoted and minimized in the same way as strong qualifications. Causal relationships and preference relations for weak qualifications are implemented as illustrated by the following clauses:

```
causes(_, ab(trans(X), exog), Z, S, H) :- block(X),
 X\= table, exogcaused(ab(trans(X), exog), S, H).

preference(ab(trans(_), _), nc, 11).
```

The weak qualification $\mathsf{Ab}(\mathit{Transportable}(x), y)$ means that block x is slippery and will slip out of the gripper when transported over long distances. The construction of extension for default theories, which contain defaults with abnormality predicates regarding weak qualifications of actions, is performed in a similar fashion as described in Section 3.3. The information given in Section 3.4 regarding the preferred extension also holds for weak qualifications.

Fluents denoting weak qualifications strengthen the antecedents of state update axioms. This is in contrast to strong qualifications where abnormality predicates occur in action precondition axioms. Thus, a suitable state update axiom with a possible weak qualification for the action $\mathit{Move}(r, u, v, w)$ is encoded as:

```
state_update(Z1, move(R,U,V,W), Z2, S, H) :-
 (not_holds_all(Y, ab(trans(U), Y), Z1), !,
   update(Z1, [on(U,W)], [on(U,V)], Z3),
   ramify(Z3, [on(U,W)], [on(U,V)], Z2, S, H));
 (holds(ab(trans(U), Y), Z1),
  (V\=table,
     update(Z1, [on(U,table)], [on(U,V)], Z3),
     ramify(Z3, [on(U,table)], [on(U,V)], Z2, S, H);
   V==table, ramify(Z1, [], [], Z2, S, H))).
```

For this state update axiom let us consider the query $\mathit{Init}(z_0, h), \mathit{Res}(\epsilon, z_0, S_0, z_1, s_1, h), \mathit{Res}(\mathit{Move}(\mathit{Robbie}, B, C, A), z_1, s_1, z_2, s_2, h), \mathit{NotHolds}(\mathit{On}(B, A), z_2)$, where the initial state is specified as in (2). This query yields the computed answer substitution:

$$\{z_2/[\mathit{On}(B, \mathit{Table}), \mathsf{Ab}(\mathit{Transportable}(B), \mathit{Exog}), \mathit{On}(A, \mathit{Table}), \mathit{On}(C, \mathit{Table}), \mathit{Has}(\mathit{Robbie}, \mathit{Glue}) \,|z],\ h/[H(\mathsf{Ab}(\mathit{Transportable}(B), \mathit{Exog}), S_0) \,|\, h_1]\}$$

Thus, the program concluded that the weak qualification $\mathsf{Ab}(\mathit{Transportable}(B), \mathit{Exog})$ occurred, and that block B can be found somewhere on the table.

4 Experiments

Our program has been tested with a LEGO® MINDSTORM^{TM} robot in a delivery scenario. The main component of such a robot is a programmable brick. It is referred to as RCX (Robotic Command Explorer) and has as its core a Hitachi H8 microcontroller. Our robot has a light sensor and a pushbutton sensor and two motors attached to the input ports and output ports of the RCX, respectively. An infrared port is used for the communication between the computer and the RCX while a user program is running on the RCX. Such programs only realize simple behaviours like following a line.

All high level control is performed by the Eclipse-Prolog system. This includes the planning process and the monitoring of the executions of actions by means of exogenous events. In case of an unexpected action failure, the system searches for explanations. The robot only executes primitive actions. Additionally, it observes the occurrence of exogenous events and reports them to the

Eclipse-Prolog system. The communication between the system and the robot is achieved through message exchange using a module from the Legolog system [6].

The robot is supposed to deliver objects from one office to another, where a cardboard with bright markers as offices and black lines as tracks denotes the floor plan. The robot has solved the task in our example scenario if there are no more requests in the current state and the robot has returned to its initial position. Two kinds of abnormalities with an appropriate preference relations between them have been defined for this scenario.

For this example domain our program was able to find explanations for unexpected action failures and to recover from them. In the scenario the system concluded that the robot had missed a marker long before the observance of an exception. In the search process all previously executed actions and related observations were taken into consideration to generate the most likely explanation first. With the established explanations the program was able to infer the robot's current position and to find a new plan to solve the task.

5 Summary

We have presented an extension of the action programming language FLUX which copes with the Qualification Problem. Our approach builds on the theoretical work of [21], where the Fluent Calculus has been embedded into a default theory to account for abnormal qualifications of actions. Our system allows to generate plans under the assumption that actions succeed as they normally do, and to reason about these assumptions in order to recover from unexpected action failures. Furthermore, it supports the specification of preferences among the default assumptions, by which is aided the search for reasonable explanations in case of unexpected action failure. While action programming languages have been extended by execution monitoring in the past, e.g., [5], our system is the first which is based on a formal approach to the Qualification Problem. It thus provides a declarative approach to troubleshooting. The crucial advantage of our approach is that explaining unexpected action failures is carried out on the basis of the same action specifications and reasoning techniques which are used for planning.

References

1. Gerhard Brewka. Adding priorities and specificity to default logic. In C. MacNish, D. Pearce, and L. M. Pereira, editors, *Proc. of the European Workshop on Logics in AI (JELIA)*, volume 838 of *LNAI*, pages 247–260, York, UK, September 1994. Springer.
2. Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. Temporal action logics (TAL): Language specification and tutorial. *Electronic Transactions on Artificial Intelligence*, 2(3–4):273–306, 1998. `http://www.ep.liu.se/ea/cis/1998/015/`.
3. Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

4. Thom Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1–3):95–138, 1998.
5. Giuseppe De Giacomo, Ray Reiter, and Mikhail Soutchanski. Execution monitoring of high-level robot programs. In Cohn, Schubert, and Shapiro, editors, *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 453–464, Trento, Italy, June 1998.
6. Hector Levesque and Maurice Pagnucco. Legolog: Inexpensive experiments in cognitive robotics. In *Cognitive Robotics Workshop at ECAI*, pages 104–109, Berlin, Germany, August 2000.
7. Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for a calculus of situations. *Electronic Transactions on Artificial Intelligence*, 3(1–2):159–178, 1998. `http://www.ep.liu.se/ea/cis/1998/018/`.
8. Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.
9. Vladimir Lifschitz. Formal theories of action (preliminary report). In J. McDermott, editor, *Proc. of IJCAI*, pages 966–972, Milan, Italy, August 1987. Morgan Kaufmann.
10. John W. Lloyd. *Foundations of Logic Programming*. Series Symbolic Computation. Springer, second, extended edition, 1987.
11. Yves Martin. Solving the Qualification Problem in FLUX. Master's thesis, TU Dresden, Germany, March 2001. `http://www.cl.inf.tu-dresden.de/~yves`.
12. Norman McCain and Hudson Turner. Satisfiability planning with causal theories. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 212–223, Trento, Italy, June 1998. Morgan Kaufmann.
13. John McCarthy. Epistemological problems of artificial intelligence. In *Proc. of IJCAI*, pages 1038–1044, Cambridge, MA, 1977. MIT Press.
14. John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.
15. Ray Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
16. Murray Shanahan. Event calculus planning revisited. In *Proc. of the European Conference on Planning (ECP)*, volume 1348 of *LNAI*, pages 390–402. Springer, 1997.
17. Michael Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1–2):317–364, 1997.
18. Michael Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):179–192, 1998. `http://www.ep.liu.se/ea/cis/1998/014/`.
19. Michael Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299, 1999.
20. Michael Thielscher. The fluent calculus: A specification language for robots with sensors in nondeterministic, concurrent, and ramifying environments. Technical Report CL-2000-01, Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology, 2000.
21. Michael Thielscher. The qualification problem: A solution to the problem of anomalous models. *Artificial Intelligence*, 2001.