

# Designing a FLUX Agent for the Dynamic Wumpus World

Michael Thielscher

Department of Computer Science  
Dresden University of Technology, Germany  
mit@inf.tu-dresden.de

## Abstract

Following an earlier proposal by Michael Genesereth, the Wumpus World has been described in (Russell & Norvig 2003) as an example for autonomous agents that need to reason logically about their actions and sensor information they acquire. The dynamic Wumpus World extends the original specification by several challenging aspects: a more complex environment, dynamic changes, and unexpected action failure. In (Thielscher 2005b) we have specified an agent for the simple Wumpus World using the high-level action programming language FLUX. In this paper we present the design of a FLUX agent for the extended problem with the help of advanced features such as dynamic state properties and a solution to the qualification problem.

## Introduction

Cognitive Robotics (McCarthy 1958; Lespérance *et al.* 1994) is concerned with the problem of endowing agents with the high-level cognitive capability of reasoning. Intelligent agents rely on this ability when drawing inferences from sensor data acquired over time, when acting under incomplete information, and in order to exhibit plan-oriented behavior. For this purpose, agents form a mental model of their environment, which they constantly update to reflect the changes they have effected and the sensor information they have acquired. A simulated environment, the Wumpus World as defined in (Russell & Norvig 2003) is a good example of a problem for controlling an agent that needs to choose its actions not only on the basis of the current status of its sensors but also on the basis of what it has previously observed or done. Moreover, some properties of the environment can be observed only indirectly, which requires the agent to logically combine observations made at different stages. The dynamic Wumpus World<sup>1</sup> extends the original problem by a variety of challenging features: a more complex environment with walls between cells, dynamic changes, and unexpected action failures which are not immediately recognizable.

In (Thielscher 2005b) we have specified an agent for the simple Wumpus World using the high-level action programming language FLUX (Thielscher 2005a), which supports

the design of intelligent agents that reason about their actions on the basis of the fluent calculus (Thielscher 1999). A constraint logic program, FLUX comprises a method for encoding incomplete states along with a technique of updating these states according to a declarative specification of the elementary actions and sensing capabilities of an agent. Incomplete states are represented by lists (of fluents) with variable tail, and negative and disjunctive state knowledge is encoded by constraints. In a series of experiments with a grid of fixed size (Sardina & Vassos 2005), the runtime behavior of the FLUX agent, with its efficient constraint solving mechanism, turned out to be superior to that of a similar GOLOG agent for the Wumpus World. Moreover, thanks to a solution to the computational frame problem under incomplete information, paired with the inference technique of progression, experiments described in (Thielscher 2005b) showed that the FLUX agent scaled up well.

In this paper we present the design of a FLUX agent for the dynamic Wumpus World, which requires to combine three extensions of the original framework:

- We use dynamic state properties in FLUX, which may change independent of the actions of the agent; hence, the agent can rely on them only at the time when they are explicitly sensed.
- We use the notion of implicational state constraints in FLUX (Thielscher 2005c) in order to represent conditional dependencies among fluents that arise from the fact that in the extended Wumpus World the presence of a pit can only be sensed if it is not obstructed by a wall.
- We address the Qualification Problem (McCarthy 1977), which arises whenever the successful execution of actions cannot be predicted with certainty. Agents for the dynamic Wumpus World rely on a solution to this problem in order to be able both to realize that some of their foregoing actions must have failed and to recover from this (Thielscher 2001).

The contribution of this paper is not to provide a new theoretical result, but rather to demonstrate how an existing knowledge representation language can be used to axiomatize a non-trivial domain, and how the action programming language and system FLUX can be used to design an intelligent agent that employs its ability to reason about actions to effectively act in this complex environment.

---

<sup>1</sup>[www.cl.inf.tu-dresden.de/~mit/LRAPP/](http://www.cl.inf.tu-dresden.de/~mit/LRAPP/)

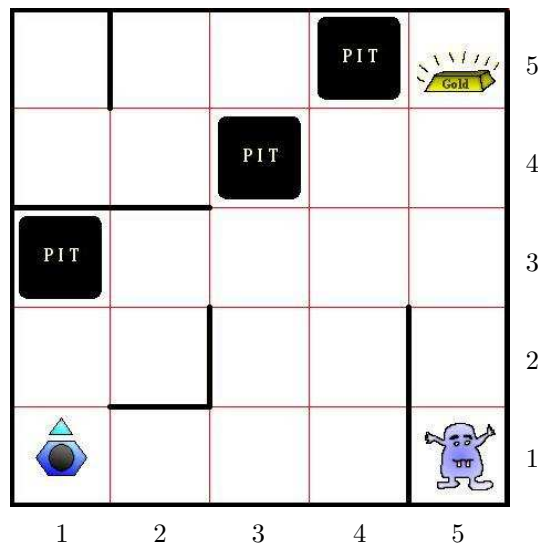


Figure 1: An example scenario in a dynamic Wumpus World where the  $5 \times 5$ -cave features three pits, gold in cell  $(5, 5)$ , and the Wumpus currently in cell  $(5, 1)$ . The agent is in its home square  $(1, 1)$  currently facing north.

The paper is organized as follows. The next section contains an informal description of the dynamic Wumpus World. Thereafter we axiomatize this environment as a fluent calculus theory, which provides the necessary background knowledge for an agent to reason about the effects of its actions in this world. We then describe a strategy for a FLUX agent for the dynamic Wumpus World, and in the final section we report on some initial experiments with our program.

### The Dynamic Wumpus World

Just like in the simple Wumpus World, the agent moves in a regular grid of cells, some of which contain bottomless pits, which are to be avoided, and somewhere in the grid there is a hostile creature called Wumpus and a heap of gold, which the agent should find and bring home. The agent can indirectly sense the presence of a pit and the Wumpus by noticing a breeze and a stench, respectively, next to these cells, and it can sense the gold once it enters the right location. The agent has exactly one arrow which it can shoot at any time in any direction and which kills the Wumpus if the latter happens to be somewhere on the trajectory.

The structure of the environment in the dynamic Wumpus World is, however, a more complex one, where cells may be separated by walls; see Figure 1 for an example. A wall can be sensed only when the agent stands next to it and faces it. Furthermore, the Wumpus may make arbitrary moves (synchronously with any physical action of the agent). The most challenging feature of the extended problem is that the action of going forward to an adjacent cell may fail with a small likelihood, and the agent has no direct means to notice this failure. Suppose, for example, the agent has acquired complete knowledge of the wall structure in Figure 1 and happens to be in cell  $(5, 3)$ , from where it attempts to take a

step northward. Even if the agent afterwards turns to check the walls around its current location, it is impossible to decide whether it has actually reached cell  $(5, 4)$ , for the wall structure in the two cells are identical. Only by making a further step northward to  $(5, 5)$  (and verifying that it now faces a wall) can the agent be sure that its actions were indeed successful. Therefore, an agent that maintains an internal model of the environment has to take into account that this model may be erroneous.

### The Background Theory

#### Fluents and states

In the fluent calculus, states are axiomatized on the basis of atomic components which are modeled as functions into the pre-defined sort `FLUENT`. The FLUX agent for the original Wumpus World described in (Thielscher 2005b) employs a systematic exploration strategy based on auxiliary parameters encoding the cells that it has visited and the path it has taken in order to backtrack from regions that have been completely explored. These parameters were kept outside the internal world model of the agent. In the dynamic Wumpus World, however, the agent's world model may be erroneous due to unnoticed failures when moving around. A successful recovery from action failure then requires not only to re-adjust its own positions, but also to revise its belief regarding the cells it has visited and the current path. These parameters must therefore be part of the internal state representation and hence be modeled as additional fluents. Because the Wumpus moves around freely, a further difference to the state representation in (Thielscher 2005b) is that the location of the creature can no longer be modeled as a fluent (to which the frame assumption applies); rather it has to be treated as a dynamic, situation-dependent property. The

Name	Type	Meaning
<i>At</i>	$\mathbb{N} \times \mathbb{N} \mapsto \text{FLUENT}$	position of the agent
<i>Facing</i>	$\{1, 2, 3, 4\} \mapsto \text{FLUENT}$	orientation of the agent
<i>Has</i>	$\{\text{Arrow}, \text{Gold}\} \mapsto \text{FLUENT}$	possessions of the agent
<i>Gold</i>	$\mathbb{N} \times \mathbb{N} \mapsto \text{FLUENT}$	location of the gold
<i>Pit</i>	$\mathbb{N} \times \mathbb{N} \mapsto \text{FLUENT}$	cells containing a pit
<i>Dead</i>	FLUENT	Wumpus is dead
<i>Wall</i>	$\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mapsto \text{FLUENT}$	walls between two cells
<i>Visited</i>	$\mathbb{N} \times \mathbb{N} \mapsto \text{FLUENT}$	visited cells
<i>Backtrack</i>	$\mathbb{N} \times \mathbb{N} \times \{1, 2, 3, 4\} \mapsto \text{FLUENT}$	element of the current path

Figure 2: The fluents for the dynamic Wumpus World. The numbers 1 . . . 4 encode the four directions north, east, south, and west. An instance  $Backtrack(x, y, d)$  is true if the agent entered square  $(x, y)$  by going into direction  $d$ .

table in Figure 2 lists the fluents that we use to model the dynamic Wumpus World, the first six of which are adopted from the original axiomatization in (Thielscher 2005b).

States in the fluent calculus are formalized using the pre-defined sort STATE. Every FLUENT is of this sort, representing a singleton state, and the special function  $\circ : \text{STATE} \times \text{STATE} \mapsto \text{STATE}$  is used to compose sub-states. The special constant  $\emptyset : \text{STATE}$  denotes the empty state. A fluent is then defined to hold in a state just in case the latter contains it:<sup>2</sup>

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

This definition is accompanied by the foundational axioms of the fluent calculus, which ensure that a state can be identified with the fluents that hold in it; see, e.g., (Thielscher 2005a). Let  $Z_0$  denote the initial state of the agent, then the agent may know the following:

$$\begin{aligned} \text{Holds}(\text{At}(x, y), Z_0) &\equiv x = 1 \wedge y = 1 \\ \text{Holds}(\text{Facing}(d), Z_0) &\equiv d = 1 \\ \text{Holds}(\text{Has}(\text{Arrow}), Z_0) \wedge \neg \text{Holds}(\text{Has}(\text{Gold}), Z_0) \\ \neg \text{Holds}(\text{Dead}, Z_0) \wedge \neg \text{Holds}(\text{Pit}(1, 1), Z_0) \\ \text{Holds}(\text{Visited}(x, y), Z_0) &\equiv x = 1 \wedge y = 1 \\ (\exists x, y) \text{Holds}(\text{Gold}(x, y), Z_0) \end{aligned} \quad (1)$$

### Actions and situations

Adopted from the situation calculus (McCarthy 1963; Reiter 2001), actions are modeled in the fluent calculus by terms into the pre-defined sort ACTION. Sequences of actions are represented by situations (sort SIT) with the help of a constant  $S_0 : \text{SIT}$ , denoting the initial situation, and the function  $Do : \text{ACTION} \times \text{SIT} \mapsto \text{SIT}$ , denoting the successor situation reached after performing an action in a situation. Situations and states are related by the pre-defined function  $State : \text{SIT} \mapsto \text{STATE}$ , accompanied by the definition<sup>3</sup>

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

<sup>2</sup>Throughout the paper, variables of sort FLUENT and STATE will be denoted by the variables  $f$  and  $z$ , respectively, possibly with sub- or superscript.

<sup>3</sup>Throughout the paper, variables of sort ACTION and SIT will be denoted by the variables  $a$  and  $s$ , respectively, possibly with sub- or superscript.

The agent in the dynamic Wumpus World can perform the same actions as in the original problem; see Figure 3. As in the situation calculus, preconditions of actions are axiomatized with the help of the predicate  $Poss : \text{ACTION} \times \text{SIT}$ . In the Wumpus World, all action can always be executed with the exception of *Go*, *Shoot*, and *Exit*, whose preconditions are as follows:

$$\begin{aligned} \text{Poss}(\text{Go}, s) &\equiv (\exists d, x, y, x', y') (\text{Holds}(\text{At}(x, y), s) \wedge \\ &\quad \text{Holds}(\text{Facing}(d), s) \wedge \\ &\quad \text{Adjacent}(x, y, d, x', y') \wedge \\ &\quad \neg \text{Holds}(\text{Wall}(x, y, x', y'), s)) \end{aligned}$$

$$\text{Poss}(\text{Shoot}, s) \equiv \text{Holds}(\text{Has}(\text{Arrow}), s)$$

$$\text{Poss}(\text{Exit}, s) \equiv \text{Holds}(\text{At}(1, 1), s)$$

The auxiliary predicate  $\text{Adjacent}(x, y, d, x', y')$  means that  $(x', y')$  is adjacent to  $(x, y)$  in direction  $d$ .

Effects of actions are specified with the help of an axiomatic definition for removal and addition of fluents:

$$\begin{aligned} z_1 - f = z_2 &\stackrel{\text{def}}{=} (z_2 = z_1 \vee z_2 \circ f = z_1) \\ &\quad \wedge \neg \text{Holds}(f, z_2) \\ z_1 + f = z_2 &\stackrel{\text{def}}{=} z_2 = z_1 \circ f \end{aligned}$$

Based on this definition, the frame problem is solved in the fluent calculus by so-called state update axioms, which define the effects of an action  $a$  in a situation  $s$  as the difference between the current  $State(s)$  and its successor  $State(Do(a, s))$ ; for example,

$$\begin{aligned} \text{Poss}(\text{Grab}, s) &\supset \\ &(\exists x, y) (\text{Holds}(\text{At}(x, y), s) \wedge \text{Holds}(\text{Gold}(x, y), s) \wedge \\ &\quad \text{State}(Do(\text{Grab}, s)) = \text{State}(s) + \text{Has}(\text{Gold})) \\ &\vee \\ &\neg (\exists x, y) (\text{Holds}(\text{At}(x, y), s) \wedge \text{Holds}(\text{Gold}(x, y), s)) \wedge \\ &\quad \text{State}(Do(\text{Grab}, s)) = \text{State}(s) \end{aligned}$$

$$\text{Poss}(\text{Exit}, s) \supset \text{State}(Do(\text{Exit}, s)) = \text{State}(s) - \text{At}(1, 1)$$

Sensor information may provide the agent with additional knowledge of a successor state. We represent the result of sensing in the dynamic Wumpus World with the help of additional situation-dependent predicates:

$$\text{Stench}, \text{Breeze}, \text{Glitter}, \text{Bump}, \text{Scream} : \text{SIT}$$

Name	Type	Meaning
<i>Go</i>	ACTION	go forward to the adjacent cell
<i>TurnRight</i>	ACTION	make a quarter turn clockwise
<i>TurnLeft</i>	ACTION	make a quarter turn counterclockwise
<i>Grab</i>	ACTION	grab the gold
<i>Shoot</i>	ACTION	shoot the arrow
<i>Exit</i>	ACTION	exit the cave

Figure 3: The actions for the dynamic Wumpus World.

Because the Wumpus moves around freely and its position is not represented as a fluent (which would persist by default), the predicate  $Stench(s)$  will only be used for the definition of the agent’s strategy. Predicate  $Breeze(s)$  provides knowledge as to whether one of the four cells next to the agent’s location contains a pit, provided there is no wall in between:

$$Breeze(s) \equiv (\exists x, y, x', y', d) (Holds(At(x, y), s) \wedge Holds(Pit(x', y'), s) \wedge Adjacent(x, y, d, x', y') \wedge \neg Holds(Wall(x, y, x', y'), s))$$

A glitter indicates that gold is at the current location:

$$Glitter(s) \equiv (\exists x, y) (Holds(At(x, y), s) \wedge Holds(Gold(x, y), s))$$

A bump is perceived in case the agent faces a wall:

$$Bump(s) \equiv (\exists x, y, d, x', y') (Holds(At(x, y), s) \wedge Holds(Facing(d), s) \wedge Adjacent(x, y, d, x', y') \wedge Holds(Wall(x, y, x', y'), s))$$

For the sake of simplicity, we assume that the knowledge of the five sensing predicates is given to the agent whenever it performs one of the actions  $TurnRight$ ,  $TurnLeft$ ,  $Shoot$ , or  $Go$ .<sup>4</sup> The state update axioms for the first three of these actions are as follows:

$$Poss(TurnRight, s) \supset (\exists d) (Holds(Facing(d), s) \wedge State(Do(TurnRight, s)) = State(s) - Facing(d) + Facing(d \bmod 4 + 1))$$

$$Poss(TurnLeft, s) \supset (\exists d) (Holds(Facing(d), s) \wedge State(Do(TurnLeft, s)) = State(s) - Facing(d) + Facing((d + 2) \bmod 4 + 1))$$

$$Poss(Shoot, s) \supset \begin{aligned} &Scream(Do(Shoot, s)) \wedge \\ &State(Do(Shoot, s)) = State(s) - Has(Arrow) + Dead \end{aligned} \vee \neg Scream(Do(Shoot, s)) \wedge State(Do(Shoot, s)) = State(s) - Has(Arrow)$$

<sup>4</sup>This is conceptually simpler than the use of an extension of the fluent calculus which is based on an explicit model of the knowledge of an agent and which allows to explicitly reason about the effects of sensing actions (Thielscher 2000).

The last axiom uses the sensor predicate  $Scream(s)$ , which is true if the agent hears a scream, that is, in case the arrow hit the Wumpus.

### Abnormalities

The most challenging aspect of the dynamic Wumpus World is the possibility of unexpected action failure. Specifically, when the agent attempts to go forward, it may actually remain where it was. The main difficulty stems from the fact that this is usually not immediately recognizable. This is an instance of the general Qualification Problem (McCarthy 1977), which arises whenever unexpected circumstances, albeit unlikely, may prevent an autonomous agent from performing the intended actions. Planning and acting under this proviso requires the agent to rigorously assume away, by default, all of the possible but unlikely *abnormal qualifications* of its actions, lest the agent is unable to make decisions which are perfectly rational although they cannot guarantee success. We adopt here a solution to this problem where a fluent calculus axiomatization is embedded in a so-called default theory (Reiter 1980; Thielscher 2001). To this end, let  $\mathbf{Ab}(s)$  be a situation-dependent predicate denoting the “abnormality” that a  $Go$ -action fails in situation  $s$ . The state update axiom for this action then comprises both the regular and the abnormal effect:

$$Poss(Go, s) \supset \begin{aligned} &\neg \mathbf{Ab}(s) \wedge (\exists d, x, y, x', y') (Holds(At(x, y), s) \wedge Holds(Facing(d), s) \wedge Adjacent(x, y, d, x', y') \wedge \\ &[\neg Holds(Visited(x', y'), s) \wedge State(Do(Go, s)) = State(s) - At(x, y) + At(x', y') + Visited(x', y') \\ &\quad + Backtrack(x', y', d) \\ &\vee Holds(Visited(x', y'), s) \wedge State(Do(Go, s)) = State(s) - At(x, y) + At(x', y')])]) \end{aligned} \vee \mathbf{Ab}(s) \wedge State(Do(Go, s)) = State(s) \quad (2)$$

The entire fluent calculus axiomatization  $\Sigma$  is then accompanied by a single default rule:

$$\frac{}{\neg \mathbf{Ab}(s)} \quad (3)$$

Reasoning with a default theory is based on the notion of *extensions*, which—thanks to the simple structure of our default (Reiter 1980)—can be defined as maximally consistent

sets

$$\Sigma \cup \{\neg \mathbf{Ab}(\sigma) : \sigma \text{ ground SIT term}\}$$

This allows agents to predict the success of a *Go*-action by default, but also to find explanations for (and to recover from) unexpected observations. For example, suppose for the sake of argument that the agent is already aware of the wall between cells (2,1) and (2,2) (cf. Figure 1). Let  $S$  denote the current situation where the agent is at (1,1) and faces north. Suppose further that the agent turns right, then attempts to go to square (1,2), and finally turns left to double-check the wall:

$$S' = Do(TurnLeft, Do(Go, Do(TurnRight, S)))$$

Our default theory then contains a unique extension, where all instances of  $\mathbf{Ab}$  are false and which thus entails

$$Holds(At(2,1), S') \wedge Holds(Facing(1), S')$$

and hence

$$Bump(S')$$

due to the wall between (2,1) and (2,2). If, however, the observation

$$\neg Bump(S')$$

is added to the axiomatization, the default theory thus augmented admits a unique extension again, in which all instances of  $\mathbf{Ab}$  are false except for  $\mathbf{Ab}(Do(TurnRight, S))$ . This together with the state update axioms, in particular the second disjunct in (2), entails

$$Holds(At(1,1), S')$$

This shows how an agent can explain an observation which contradicts its expectations, and how it can automatically recover by finding a minimal set of positive  $\mathbf{Ab}$ -instances which are consistent with the observations.

## FLUX

FLUX (Thielscher 2005a; 2005d) is a high-level programming method for the design of intelligent agents that reason about their actions on the basis of the fluent calculus (Thielscher 1999). A constraint logic program, FLUX comprises a method for encoding incomplete states along with a technique of updating these states according to a declarative specification of the elementary actions and sensing capabilities of an agent. Incomplete states are represented by lists (of fluents) with variable tail, and negative and disjunctive state knowledge is encoded by constraints. The incomplete initial state knowledge of the agent in the dynamic Wumpus World (cf. (1)), for example, is encoded in FLUX by the following clause:

```
init(Z0) :-
  Z0 = [at(1,1),facing(1),has(arrow),
        visited(1,1),gold(X,Y) | Z],
  not_holds_all(at(_,_),Z),
  not_holds_all(facing(_,)Z),
  not_holds_all(visited(_,_)Z),
  not_holds(has(gold),Z),
  not_holds(dead,Z),
  not_holds(pit(1,1),Z).
```

Constraint `not_holds(f,z)` encodes the fluent calculus formula  $\neg Holds(f,z)$ , and `not_holds_all(f,z)` stands for  $(\forall \vec{x}) \neg Holds(f,z)$ , where  $\vec{x}$  are the variables in  $f$ .

State update axioms in the fluent calculus can be directly translated into FLUX as clauses which define the predicate `state_update(Z1,A,Z2,Y)` with the intended meaning that performing action  $a$  in state  $z_1$  and sensing  $\vec{y}$  yields updated state  $z_2$ . As an example, we give the specification of the regular effects of action *Go* with sensor input for breeze, glitter, and wall—the other state update axioms are encoded in a similar fashion:

```
state_update(Z1,go,Z2,[Br,Gl,Bm]) :-
  holds(at(X,Y),Z1),
  holds(facing(D),Z1),
  adjacent(X,Y,D,X1,Y1),
  ( not_holds(visited(X1,Y1),Z1),
    update(Z1,[at(X1,Y1),
                visited(X1,Y1),
                backtrack(X1,Y1,D)],
          [at(X,Y)],Z2)
  ;
  holds(visited(X1,Y1),Z1),
  update(Z1,[at(X1,Y1)],
        [at(X,Y)],Z2) ),
  breeze(X1,Y1,Br,Z2),
  glitter(X1,Y1,Gl,Z2),
  bump(X1,Y1,D,Bm,Z2).
```

Standard FLUX predicate `update(Z1,P,N,Z2)` means that the update of incomplete state  $z_1$  by positive and negative effects  $p$  and  $n$ , respectively, results in state  $z_2$ .

Due to the possible existence of walls, the evaluation of the sensor input in the dynamic Wumpus World requires to use a recent extension of FLUX which allows to express conditional state knowledge in form of a state constraint `if_then_holds(f,g,z)`, encoding the fluent calculus formula  $Holds(f,z) \supset Holds(g,z)$  (Thielscher 2005c):

```
breeze(X,Y,Percept,Z) :-
  XE#=X+1, XW#=X-1, YN#=Y+1, YS#=Y-1,
  ( Percept=false,
    if_then_holds(pit(XE,Y),
                  wall(X,Y,XE,Y),Z),
    if_then_holds(pit(XW,Y),
                  wall(X,Y,XW,Y),Z),
    if_then_holds(pit(X,YN),
                  wall(X,Y,X,YN),Z),
    if_then_holds(pit(X,YS),
                  wall(X,Y,X,YS),Z)
  ; Percept=true ).
```

The current expressiveness of FLUX does not allow to give a complete account of the conditional knowledge of pits that follows from sensing a breeze. Fortunately, this is not necessary in order that the agent functions effectively, because the agent will always enter a square only if this is *known* to be safe. Hence, there is no need to distinguish between not knowing whether a cell is safe and knowing for sure that there is a pit!

The other two sensors are encoded in a straightforward manner:

```
glitter(X,Y,Percept,Z) :-
    Percept=false,
    not_holds(gold(X,Y),Z)
;
Percept=true,
holds(gold(X,Y),Z).

bump(X,Y,D,Percept,Z) :-
    adjacent(X,Y,D,X1,Y1),
    ( Percept=false,
      not_holds(wall(X,Y,X1,Y1),Z),
      not_holds(wall(X1,Y1,X,Y),Z)
    ;
      Percept=true,
      holds(wall(X,Y,X1,Y1),Z),
      holds(wall(X1,Y1,X,Y),Z) ).
```

The state update axiom given above merely encodes the normal effect of the *Go*-action. The solution to the Qualification Problem in FLUX (Thielscher 2005d) requires to encode the abnormal effects of actions by an additional clause; in our case (cf. (2)),

```
ab_state_update(Z1,go,Z2,
                [Br,G1,Bm]) :-
    Z2 = Z1,
    holds(at(X,Y),Z2),
    holds(facing(D),Z2),
    breeze(X,Y,Br,Z2),
    glitter(X,Y,G1,Z2),
    bump(X,Y,D,Bm,Z2).
```

The application of default rule (3) is modeled in FLUX as follows: Predicate *state\_update* is used by default whenever the world model gets updated after an action has been performed (via predicate *execute*, see below). Only when the agent, upon executing an action, makes an observation which is inconsistent with the world model thus updated, predicate *ab\_state\_update* can be additionally used in order to find a sequence of updates which is consistent with the sequence of observations that have been made; for details we refer to (Thielscher 2005d).

## The Strategy

FLUX allows to define complex behaviors by means of strategy programs. These strategies are independent of the background specification, which allows to devise and compare different strategies. In the following, we describe a specific strategy for the dynamic Wumpus World, which is similar to the systematic exploration defined in (Thielscher 2005b) for the simple environment.

After initializing the world model, the agent executes a loop where it either grabs the gold, if possible, and goes home, or it explores a new cell, if possible, and otherwise it backtracks. If the latter is not possible, too, then the agent believes it is at its home square and exits (without having found the gold). The latter may fail, because of an erroneous world model, in which case the agent remains somewhere in the environment and continues with its revised model:

```
main_loop(Z) :-
    knows_val([X,Y],at(X,Y),Z),
    ( knows(gold(X,Y),Z) ->
      execute(grab,Z,Z1), go_home(Z1)
    ;
      explore(X,Y,Z,Z1)
      -> main_loop(Z1)
    ;
      backtrack(X,Y,Z,Z1)
      -> main_loop(Z1)
    ;
      execute(exit,Z,Z1),
      ( knows_val([X1,Y1],at(X1,Y1),Z1)
        -> main_loop(Z1)
      ; true ) ).
```

Here, the standard FLUX predicate *knows\_val*( $X,F,Z$ ) means that the agent knows arguments  $\vec{x}$  of fluent  $f$  so that  $f$  holds in state  $z$ ; predicate *knows*( $F,Z$ ) is true if fluent  $f$  is known to hold in state  $z$ ; and predicate *execute*( $A,Z1,Z2$ ) means that the actual execution of action  $a$  in state  $z_1$  leads to state  $z_2$ .

The agent can take an exploration step if it is adjacent to a cell which has not been visited, which is not blocked by a wall, and which is known not to house a pit. If these conditions are satisfied, then the agent enters this cell and makes three turns in order to gain information about the wall structure. The agent needs this information to decide whether it can continue with the exploration. Knowledge of the wall structure also helps with locating the pits in case a breeze is being sensed. But even when the agent knows the complete wall structure, double-checking the presence of walls helps with recognizing a failed *Go*-action as early as possible:

```
explore(X,Y,Z1,Z7) :-
    wumpus_alert(X,Y,Z1,Z2),
    adjacent(X,Y,D,X1,Y1),
    knows_not(visited(X1,Y1),Z2),
    knows_not(wall(X,Y,X1,Y1),Z2),
    knows_not(pit(X1,Y1),Z2)
->
    turn_to(D,Z2,Z3),
    execute(go,Z3,Z4),
    execute(turn_left,Z4,Z5),
    execute(turn_left,Z5,Z6),
    execute(turn_left,Z6,Z7).
```

Here, the standard FLUX predicate *knows\_not*( $F,Z$ ) means that the agent knows that fluent  $f$  is false in state  $z$ . The auxiliary predicate *adjacent*( $X,Y,D,X1,Y1$ ) defines cell  $(x_1,y_1)$  to be adjacent to cell  $(x,y)$  in direction  $d$ . Auxiliary predicate *wumpus\_alert* defines the reaction in case the agent senses a stench while it knows that the Wumpus is still alive. Whenever this happens, our agent either turns into a direction with no wall and shoots the arrow, in the hope of hitting the Wumpus. In case the agent has already used its arrow unsuccessfully, then upon sensing a stench it backtracks to the previous location (we omit the details). Finally, auxiliary predicate *turn\_to*( $D,Z1,Z2$ ) means to turn into direction  $d$ .

If no exploration step is possible, the agent backtracks according to the following definition:

```
backtrack(X,Y,Z1,Z3) :-
    knows_val([D],backtrack(X,Y,D),Z1),
    R is (D+1) mod 4 + 1,
    turn_to(R,Z1,Z2),
    execute(go,Z2,Z3).
```

(Note that  $R$  is the reverse of direction  $D$ ). According to its definition, predicate  $\text{backtrack}(X,Y,Z1,Z3)$  is false if state  $z_1$  does not contain a fluent  $\text{Backtrack}(x,y,d)$ , which means that the agent is in cell (1,1) (or rather that it believes it is).

Once the agent has found the gold, it uses the backtracking information to find its way to the home square. As above, if the agent believes it is at its home square then it exits, which may fail because of an erroneous world model, in which case the agent remains somewhere in the environment and continues with backtracking:

```
go_home(Z) :-
    knows_val([X,Y],at(X,Y),Z),
    ( backtrack(X,Y,Z,Z1)
      -> go_home(Z1)
    );
    execute(exit,Z,Z1),
    ( knows_val([X1,Y1],at(X1,Y1),Z1)
      -> go_home(Z1)
    ); true).
```

## Discussion

In order to test the agent program, we ran a series of experiments with the scenario depicted in Figure 1. Of a total of 30 runs, 47% were successful in that the agent managed to locate and grab the gold and to exit safely. In 30% of the test cases, the agent stayed alive but did not find the gold, because the Wumpus came in the way and the agent did not guess its location correctly when shooting the arrow. In 23% of the runs the Wumpus killed the agent by moving into its location.

The agent was always able to recover from action failure. In some cases, where one or more *Go* actions failed at early stages, the agent built up a largely erroneous model of the environment and it took until the very end, when the agent intended to exit, to detect the abnormalities. Generally, the later an abnormality was detected, the longer it took to infer the correct position because the space of possible explanations is exponential in the total number of *Go* actions. One way to improve the computational behavior of this recovery process would be to begin with searching for explanations in which just  $k = 1$  failure occurs, and to iteratively increase  $k$  until a consistent sequence of updates is found. Under the assumption that failures are rare, this would avoid computing a large portion of the search space. Generally speaking, abnormalities should only be used for the specification of truly exceptional effects of actions. If the *Go* action in the Wumpus World did not have a high chance of succeeding, then it had better be specified by a nondeterministic state update axiom, lest the agent has to frequently recover from an erroneous world model.

## References

- Lespérance, Y.; Levesque, H.; Lin, F.; Marcu, D.; Reiter, R.; and Scherl, R. 1994. A logical approach to high-level robot programming—a progress report. In Kuipers, B., ed., *Control of the Physical World by Intelligent Agents, Papers from the AAAI Fall Symposium*, 109–119.
- McCarthy, J. 1958. Programs with Common Sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes*, volume 1, 77–84. (Reprinted in: (McCarthy 1990)).
- McCarthy, J. 1963. *Situations and Actions and Causal Laws*. Stanford University, CA: Stanford Artificial Intelligence Project, Memo 2.
- McCarthy, J. 1977. Epistemological problems of artificial intelligence. In Reddy, R., ed., *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1038–1044. Cambridge, MA: MIT Press.
- McCarthy, J. 1990. *Formalizing Common Sense*. Norwood, New Jersey: Ablex. (Edited by V. Lifschitz).
- Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.
- Reiter, R. 2001. *Knowledge in Action*. MIT Press.
- Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice-Hall.
- Sardina, S., and Vassos, S. 2005. The wumpus world in IndiGolog: A preliminary report. In Morgenstern, L., and Pagnucco, M., eds., *Proceedings of the Workshop on Nonmonotonic Reasoning, Action and Change at IJCAI*, 90–95.
- Thielscher, M. 1999. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence* 111(1–2):277–299.
- Thielscher, M. 2000. Representing the knowledge of a robot. In Cohn, A.; Giunchiglia, F.; and Selman, B., eds., *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 109–120. Breckenridge, CO: Morgan Kaufmann.
- Thielscher, M. 2001. The qualification problem: A solution to the problem of anomalous models. *Artificial Intelligence* 131(1–2):1–37.
- Thielscher, M. 2005a. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming* 5(4–5):533–565.
- Thielscher, M. 2005b. A FLUX agent for the Wumpus World. In Morgenstern, L., and Pagnucco, M., eds., *Proceedings of the Workshop on Nonmonotonic Reasoning, Action and Change at IJCAI*, 104–108.
- Thielscher, M. 2005c. Handling implicational and universal quantification constraints in flux. In van Beek, ed., *Proceedings of the International Conference on Principle and Practice of Constraint Programming (CP)*, volume 3709 of *LNCS*, 667–681. Sitges, Spain: Springer.
- Thielscher, M. 2005d. *Reasoning Robots: The Art and Science of Programming Robotic Agents*, volume 33 of *Applied Logic Series*. Kluwer.