Reinventing Shakey

Murray Shanahan

Department of Electrical and Electronic Engineering,
Imperial College,
Exhibition Road,
London SW7 2BT,
England.
m.shanahan@ic.ac.uk

Abstract

This paper describes the logical foundations of an implemented system which employs resolution-based theorem proving techniques for high-level robot control. The paper offers complementary logical characterisations of perception and planning, and shows how sensing, planning and acting are interleaved to control a real robot.

1 Introduction

In the late Sixties, when the Shakey project started [Nilsson, 1984], the vision of robot design based on logical representation seemed both attractive and attainable. Through the Seventies and early Eighties, however, the desire to build working robots led researchers away from logic to more practical but *ad hoc* approaches to representation. This movement away from logical representation reached an extreme in the late Eighties and early Nineties when Brooks jettisoned the whole idea of representation, along with the so-called sense-model-plan-act architecture epitomised by Shakey [Brooks, 1991].

However, the Shakey style of architecture, having an overtly logic-based deliberative component, seems to offer researchers a direct path to robots with high-level cognitive skills, such as planning, reasoning about other agents, and communication with other agents. Accordingly, a number of researchers have instigated a Shakey revival, and are aiming to achieve robots with these sorts of high-level cognitive skills by using logic as a representational medium [Lespérance, *et al.*, 1994].

This paper describes one such robot. The paper concentrates on logical foundations, but everything described has been implemented, deployed, and tested on a real robot, namely a Khepera, a miniature robot with two drive wheels and a suite of eight infra-red proximity sensors around its circumference. The robot inhabits a miniaturised office-like environment, depicted in Figure 1.

The robot has a simple repertoire of low-level actions, executed on-board, which includes wall-following, turning into doorways, and turning around corners. Using these, the high-level, off-board controller manoeuvres the robot around its environment.

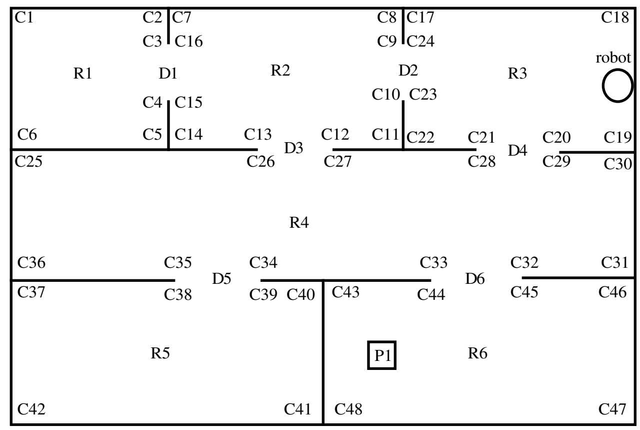

Figure 1: The Robot's Environment

1.1 A Motivating Example

To get a feel for its capabilities, let's take a look at an example of the way in which the high-level controller functions. Suppose the robot is initially between corners C18 and C19, as shown, and suppose it has the goal of retrieving package P1 from room R6. Informally, this is how the robot's high-level controller achieves the goal.

First, the robot plans a route. As soon as it finds a complete (though perhaps not fully decomposed) plan with an executable first action, the robot starts carrying that plan out. In this case, the first action is to go through door D4, so the robot sets out along the wall until it reaches corner C19. It then turns the corner, and heads off towards door D4.

Suppose someone now closes door D4. Unfortunately, because of its poor sensors, the robot cannot detect closed doors, which are indistinguishable from walls. So the robot continues wall-following until it reaches corner C22.

Up to this point, the assimilation of the robot's sensor data has been a trivial matter. The sensor events it receives are exactly what it would expect given what it has done and what it believes about its environment. So the explanations of those sensor events are trivial. But the encounter with a corner at this time requires a non-empty explanation.

Using abduction, the robot constructs an explanation of its encounter with the corner — door D4 must have been closed, and it must now be at corner C22. But this new piece of information conflicts with the assumptions underlying the plan it is executing. So the robot is forced to replan. It now finds a new route to room R6, via doors D2, D3, and D6, which it successfully executes and retrieves the package.

In the implemented system, the robot is controlled by a sense-plan-act cycle. Sensing and planning are both resolution-based abductive theorem proving processes, and in each iteration of the cycle, the sense phase and plan phase carry out a single step of resolution each. The chief aim of the rest of this paper is to set out the logical foundations of these planning and perception processes.

The paper presents,

- a short introduction to the event calculus, a formalism for reasoning about action,
- a logical account of abductive event calculus planning,
- a complementary logical account of abductive sensor data assimilation, also based on the event calculus, and,
- a brief outline of event calculus hierarchical planning, through which plans can be generated in progression order.

2 Representing Action

To supply the required logical accounts of perception and planning, a formalism for reasoning about action is needed. The formalism presented here is based on the circumscriptive event calculus [Shanahan, 1997a], [Shanahan, 1999]. The event calculus is a well-established logical formalism for reasoning about actions. It is capable of representing a wide variety of phenomena, including actions with indirect effects, non-deterministic actions, concurrent actions, and continuous change [Shanahan, 1999]. A robust solution to the frame problem exists for the event calculus, that works in the presence of all of these phenomena [Shanahan, 1997a].

Because this material is presented in considerable detail elsewhere, the description here will be kept fairly brief. A many sorted language is assumed, with variables for *fluents*, *actions* (or *events*), and *time points*. We have the following axioms, whose conjunction will be denoted EC. Their main purpose is to constrain the predicate HoldsAt. HoldsAt(f,t) represents that fluent f holds at time t. Throughout the paper, all variables are universally quantified with maximum scope, unless otherwise indicated.

$$\mathrm{HoldsAt}(f,t) \leftarrow \mathrm{Initially_P}(f) \wedge \neg\, \mathrm{Clipped}(0,f,t) \qquad \text{(EC1)}$$

$$\mathrm{HoldsAt}(f,t3) \leftarrow \mathrm{Happens}(a,t1,t2) \wedge \mathrm{Initiates}(a,f,t1) \wedge t2 < t3 \wedge \neg\, \mathrm{Clipped}(t1,f,t3) \qquad \text{(EC2)}$$

$$\mathrm{Clipped}(t1,f,t4) \leftrightarrow \exists\, a,t2,t3\, [\mathrm{Happens}(a,t2,t3) \wedge t1 < t3 \wedge t2 < t4 \wedge [\mathrm{Terminates}(a,f,t2) \vee \mathrm{Releases}(a,f,t2)]] \qquad \text{(EC3)}$$

$$\neg\, \mathrm{HoldsAt}(f,t) \leftarrow \mathrm{Initially_N}(f) \wedge \neg\, \mathrm{Declipped}(0,f,t) \qquad \text{(EC4)}$$

$$\neg\, \mathrm{HoldsAt}(f,t3) \leftarrow \mathrm{Happens}(a,t1,t2) \wedge \mathrm{Terminates}(a,f,t1) \wedge t2 < t3 \wedge \neg\, \mathrm{Declipped}(t1,f,t3) \qquad \text{(EC5)}$$

$$\mathrm{Declipped}(t1,f,t4) \leftrightarrow \exists\, a,t2,t3\, [\mathrm{Happens}(a,t2,t3) \wedge t1 < t3 \wedge t2 < t4 \wedge [\mathrm{Initiates}(a,f,t2) \vee \mathrm{Releases}(a,f,t2)]] \qquad \text{(EC6)}$$

$$\mathrm{Happens}(a,t1,t2) \rightarrow t1 \leq t2 \qquad \text{(EC7)}$$

A particular *domain* is described in terms of Initiates, Terminates, and Releases formulae. Initiates(a,f,t) represents that fluent f starts to hold after action a at time t. Conversely, Terminates(a,f,t) represents that f ceases to hold after action a at t. Releases(a,f,t) represents that fluent f is no longer subject to the common sense law of inertia after action a at t.

A particular *narrative* of events is described in terms of Happens and Initially formulae. The formulae $\mathrm{Initially_P}(f)$ and $\mathrm{Initially_N}(f)$ respectively represent that fluent f holds at time 0 and does not hold at time 0. Happens(a,t1,t2) represents that action or event a occurs, starting at time t1 and ending at time t2. Table 1 summarises the predicates of the calculus.

Formula	Meaning
$\mathrm{Initiates}(\alpha,\beta,\tau)$	Fluent β starts to hold after action α at time τ
$\mathrm{Terminates}(\alpha,\beta,\tau)$	Fluent β ceases to hold after action α at time τ
$\mathrm{Initially_P}(\beta)$	Fluent β holds from time 0
$\mathrm{Initially_N}(\beta)$	Fluent β does not hold from time 0
$\tau1 < \tau2$	Time point $\tau1$ is before time point $\tau2$
$\mathrm{Happens}(\alpha,\tau)$	Action α occurs at time τ
$\mathrm{Happens}(\alpha,\tau1,\tau2)$	Action α starts at time $\tau1$ and ends at time $\tau2$
$\mathrm{HoldsAt}(\beta,\tau)$	Fluent β holds at time τ
$\mathrm{Clipped}(\tau1,\beta,\tau2)$	Fluent β is terminated between times $\tau1$ and $\tau2$
$\mathrm{Declipped}(\tau1,\beta,\tau2)$	Fluent β is initiated between times $\tau1$ and $\tau2$

Table 1: Event Calculus Predicates

A two-argument version of Happens is defined as follows.

$$\mathrm{Happens}(a,t) \equiv_{def} \mathrm{Happens}(a,t,t)$$

Formulae describing triggered events are allowed, and will generally have the form,

$$\mathrm{Happens}(\alpha,\tau) \leftarrow \Pi.$$

As we'll see in Section 5, similar formulae can be used to define compound actions, which are used for hierarchical planning.

The frame problem is overcome through circumscription. Given a conjunction Σ of Initiates, Terminates, and Releases formulae describing the effects of actions, a conjunction Δ of Initially, Happens and temporal ordering formulae describing a narrative of actions and events, and a conjunction Ω of uniqueness-of-names axioms for actions and fluents, we're interested in,

$$\text{CIRC}[\Sigma\ ;\ \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta\ ;\ \text{Happens}] \wedge \text{EC} \wedge \Omega.$$

By minimising Initiates, Terminates and Releases we assume that actions have no unexpected effects, and by minimising Happens we assume that there are no unexpected event occurrences. In all the cases we're interested in, Σ and Δ will be conjunctions of Horn clauses, and the circumscriptions will reduce to predicate completions.

Care must be taken when domain constraints and triggered events are included. Domain constraints (or "state constraints") are formulae that constrain the combination of fluents that can hold simultaneously, and are used to define actions with indirect effects. Triggered events are events that occur when some specified combination of fluents holds. The former must be conjoined to EC, while the latter are conjoined to Δ.

3 A Logical Account of Planning

Planning can be thought of as the inverse operation to temporal projection, that is to say reasoning forwards in time from causes to effects. Temporal projection in the event calculus is naturally cast as a deductive task. Given Σ, Ω and Δ as above, we're interested in HoldsAt formulae Γ such that,

$$\text{CIRC}[\Sigma\ ;\ \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta\ ;\ \text{Happens}] \wedge \text{EC} \wedge \Omega \vDash \Gamma.$$

Conversely, planning in the event calculus can be considered as an abductive task, since it is a form of reasoning from effects to causes. Given a domain description Σ, a conjunction Γ of goals (HoldsAt formulae), and a conjunction Δ_N of Initially_P and Initially_N formulae describing the initial situation, a *plan* is a consistent conjunction Δ_P of Happens and temporal ordering formulae such that,

$$\text{CIRC}[\Sigma\ ;\ \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta_N \wedge \Delta_P\ ;\ \text{Happens}] \wedge \text{EC} \wedge \Omega \vDash \Gamma.$$

The following formulae capture the connectivity of the rooms, as shown in Figure 1.

Connects(D1,R1,R2) (W2.1)

Connects(D2,R2,R3) (W2.2)

Connects(D3,R2,R4) (W2.3)

Connects(D4,R3,R4) (W2.4)

Connects(D5,R4,R5) (W2.5)

Conects(D6,R4,R6) (W2.6)

Connects(d,r1,r2) $\leftarrow$ Connects(d,r2,r1) (W2.7)

Let's suppose the robot can perform only one action, which is to go through a specified door d, denoted by the term GoThrough(d). (In the implemented system, the GoThrough action is broken down by hierarchical planning into a sequence of FollowWall and Turn actions.) We'll assume, for now, that doors are always open. The only fluent in the domain is InRoom(r) representing that the robot is in room r. We have the following Initiates and Terminates formulae.

Initiates(GoThrough(d),InRoom(r1),t) $\leftarrow$ Connects(d,r2,r1) $\wedge$ HoldsAt(InRoom(r2),t) (R2.1)

Terminates(GoThrough(d),InRoom(r),t) $\leftarrow$ HoldsAt(InRoom(r),t) (R2.2)

Since there is only one action and only one fluent, this example doesn't require any uniqueness-of-names axioms.

Suppose the robot is initially in room R3.

Initially_P(InRoom(R3)) (N2.1)

The goal is to get the robot to room R6.

HoldsAt(InRoom(R6),T) (G2.1)

Clearly one plan for achieving the goal is to go through D4 then go through D6.

Happens(GoThrough(D4),T1) (P2.1)

Happens(GoThrough(D6),T2) (P2.2)

T1 < T2 (P2.3)

T2 < T (P2.4)

The fact that this is a plan according to the abductive definition is expressed in the following proposition. Let,

- Σ be the conjunction of (R2.1) and (R2.2)
- Δ_N be formula (N2.1),
- Δ_P be the conjunction of (P2.1) to (P2.4),
- Φ be the conjunction of (W2.1) to (W2.7), and
- Γ be formula (G2.1).

Proposition 3.1.

$$\text{CIRC}[\Sigma\ ;\ \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta_N \wedge \Delta_P\ ;\ \text{Happens}] \wedge \text{EC} \wedge \Phi \vDash \Gamma.$$ □

3.1 Implementation

This account of planning can be implemented through abductive logic programming, as described in [Shanahan, 1997c] and [Shanahan, 2000]. The implementation is an abductive meta-interpreter, which has been tailored for the event calculus. HoldsAt goals are treated in a special way, and the sub-goals they generate are processed in a particular order to prevent looping. In addition, a dedicated constraint solver is applied to temporal ordering constraints, whose efficiency is improved by the maintenance of a cache of temporal ordering lemmas.

The computation carried out by the resulting system strongly resembles that of a hand-coded partial-order planning algorithm. (The present example of route planning can be reduced to graph search, but for general purpose planning, we need a more generic technique.) In particular, the implementation has to record negated Clipped formulae that it has proved, and these correspond to *protected links* in partial-order planning terminology. A protected link records the fact that the value of a fluent initiated or terminated by a given action must be preserved until the occurrence of another subsequent action whose preconditions depend that fluent.

The efficiency of this planner is in line with that of a partial order planner. A more recent event calculus planner, based on the planning as satisfiability work of Kautz and Selman [1996], is more efficient, but has not yet been applied to robotics [Shanahan & Witkowski, 2000]. However, as we'll see later, this is only part of the story for planning. In order to be able to interleave planning, sensing and acting in a respectable way, we need to carry out *hierarchical* planning. Moreover, by using hierarchical planning wherever possible, we minimise the use of search-heavy planning from first principles.

Now, what exactly is the relationship between the logical specification of planning and its implementation by means of abductive logic programming? This question is addressed more fully in [Shanahan, 2000]. But, in outline, here is the answer.

In [Shanahan, 2000], a class of event calculus domain descriptions is defined which, among other restrictions, confines Initiates and Terminates formulae to the Horn clause subset. This means that the theorems of Lifschitz [1994] can be applied to reduce the circumscriptions of these formulae to predicate completion. Moreover, since these restrictions rule out recursion, predicate completion and SLDNF coincide for the class of theories in question. Thus the prospect of a straightforward logic programming implementation is brought closer.

However, the use of temporal ordering constraints whose completions we can't assume complicates the issue. In effect, the abductive meta-interpreter treats temporal constraints in a special way, using a dedicated constraint solver.

For the restricted class of event calculus theories defined, the abductive meta-interpreter of [Shanahan, 2000] is both sound and complete with respect to the abductive characterisation of planning. This meta-interpreter forms the basis of both the planning and sensor data assimilation components in the present work.

However, many useful domain descriptions fall outside the scope of these theorems. Domain descriptions including formulae describing compound actions are an example of particular relevance to the present paper, since they form the basis of hierarchical planning. In the presence of such formulae, the soundness and completeness of the meta-interpreter are conjectural. So there is a theoretical gap that needs to be filled here, and this is the subject of ongoing work.

Next, we'll look into the topic of perception.

4 A Logical Account of Perception

This section offers a logical account of sensor data assimilation (perception) which mirrors the logical account of planning in Section 3. The need for such an account arises from the fact that sensors do not deliver facts directly into the robot's model of the world. Rather they provide raw data from which facts can be inferred.

The methodology for supplying the required logical account is as follows [Shanahan, 1997b]. First, using a suitable formalism for reasoning about actions, construct a theory Σ of the effects of the robot's actions on the world and the impact of the world on the robot's sensors. Second, consider sensor data assimilation as abduction with this theory. Roughly speaking, given a narrative Δ of the robot's actions, and a description Γ of the robot's sensor data, the robot needs to find some Ψ such that,

$$\Sigma \wedge \Delta \wedge \Psi \models \Gamma.$$

In event calculus terms, Γ might comprise Happens and/or HoldsAt formulae describing sensor events or values, and Ψ might comprise Initially_N and Initially_P formulae describing the environment's initial configuration and/or Happens formulae describing the intervening actions of other agents which have modified that configuration.

To illustrate this, we'll stay with the office delivery domain. But we must begin with a look at the sensory capabilities of the Khepera robots which are being used to test the ideas presented in this paper.

The Khepera can be straightforwardly programmed to navigate around the environment of Figure 1. Using its proximity sensors, it can follow walls and detect inner and outer corners. If all the doors are open, the GoThrough action of Section 3 can be executed, assuming the robot's initial location is known, by counting inner and outer corners until the robot reaches the required door, then passing through it.

If any of the doors is closed, however, this approach to executing the GoThrough action will fail, because the infra-red proximity sensors cannot detect a closed door, which looks to them just like a continuation of the wall.

This, in an extreme form, is the predicament facing any perceptual system. Inference must be carried out on raw sensor data in order to produce knowledge. In this case, the robot can abduce the fact that a door is closed as the only possible explanation of its unexpected arrival at an inner corner instead of the outer corner of the doorway.

Our aim here is to give a formal account of this sort of inference that gels with the formal account of planning already supplied. Indeed, in the implemented system, the same knowledge, expressed using the same formalism, is used for both planning and sensor data assimilation. Furthermore, as already emphasised, both planning and sensor data assimilation are viewed as abductive tasks with a very similar character. This means that the same abductive logic programming technology, indeed the very same code, can be used to implement both processes.

4.1 The Robot's Environment

Returning to the example at hand, the representation of the robot's environment, as depicted in Figure 1, now needs to include corners, which were neglected in the planning example. The formula NextCorner(r,c1,c2) represents that corner c2 is the next inner or outer corner in room r after corner c1, in a clockwise direction. For room R1 alone, we have the following formulae.

NextCorner(R1,C1,C2)	NextCorner(R1,C2,C3)
NextCorner(R1,C3,C4)	NextCorner(R1,C4,C5)
NextCorner(R1,C5,C6)	NextCorner(R1,C6,C1)

In addition, the formula Door(d,c1,c2) represents that there is a doorway between the two corners c1 and c2. For each door, there will be a pair of such formulae. Here they are for door D1.

Door(D1,C3,C4)	Door(D1,C15,C16)

Finally, the formulae Inner(c) and Outer(c) represent respectively that c is an inner corner and c is an outer corner. Again confining our attention to room R1, we have the following.

Inner(C1)	Inner(C2)
Outer(C3)	Outer(C4)
Inner(C5)	Inner(C6)

Each of these predicates will need to be minimised using circumscription, so that their completions are formed.

4.2 The Robot's Effect on the World

Now we can formalise the effects of the robot's actions on the world. To simplify the example, the following formulae assume the robot always hugs the left wall, although parameters are provided which allow for it to hug the right wall as well.

Again, a finer grain of detail is required than for the planning example. Instead of a single GoThrough action, the robot's repertoire now comprises three actions: FollowWall, Turn(s), and GoStraight, where s is either Left or Right. These actions affect three fluents. The fluent AtCorner(c,s) holds if the robot is at (inner or outer) corner c, with c in direction s, where s is Left or Right. The fluent BesideWall(w,s) holds if the robot is adjacent to wall w in direction s, where s is Left or Right. The fluent InDoorway(d,r) holds if the robot is in doorway d, with its back to room r. (By convention, the three fluents are mutually exclusive.)

Let's formalise the effects of the three actions in turn. Each action is assumed to be instantaneous, an assumption which has no practical implications in the present example. The term Wall(c1,c2) denotes the wall between corners c1 and c2. First, if the robot follows a wall, it ends up at the next visible corner.

$$\begin{aligned}&\text{Initiates(FollowWall,AtCorner(c3,Left),t)} \leftarrow \\ &\quad \text{HoldsAt(BesideWall(Wall(c1,c2),Left),t)} \wedge \\ &\qquad \text{NextVisibleCorner(c1,c3,Left,t)}\end{aligned} \tag{K4.1}$$

$$\text{Terminates(FollowWall,BesideWall(w,s),t)} \tag{K4.2}$$

The formulae NextVisibleCorner(c1,c2,s,t) means that, at time t, c2 is the next visible corner after c1, where the wall in question is in direction s. The corner of a doorway whose door is closed is invisible.

$$\begin{aligned}&\text{NextVisibleCorner(c1,c2,Left,t)} \leftarrow \\ &\quad \text{NextCorner(r,c1,c2)} \wedge \neg\, \text{InvisibleCorner(c2,t)}\end{aligned} \tag{K4.3}$$

$$\begin{aligned}&\text{NextVisibleCorner(c1,c3,Left,t)} \leftarrow \\ &\quad \text{NextCorner(r,c1,c2)} \wedge \text{InvisibleCorner(c2,t)} \wedge \\ &\qquad \text{NextVisibleCorner(c2,c3,Left,t)}\end{aligned} \tag{K4.4}$$

$$\begin{aligned}&\text{NextVisibleCorner(c1,c2,s,t)} \wedge \\ &\quad \text{NextVisibleCorner(c1,c3,s,t)} \rightarrow \text{c2} = \text{c3}\end{aligned} \tag{K4.5}$$

$$\begin{aligned}&\text{InvisibleCorner(c1,t)} \leftrightarrow \\ &\quad \exists\, \text{d,c2 [[Door(d,c1,c2)} \vee \text{Door(d,c2,c1)]} \wedge \\ &\qquad \neg\, \text{HoldsAt(DoorOpen(d),t)]}\end{aligned} \tag{K4.6}$$

Next we have the GoStraight action, which the robot executes to bypass a doorway, travelling in a straight line from the near corner of the doorway and coming to rest when it detects the far corner.

$$\begin{aligned}&\text{Initiates(GoStraight,} \\ &\quad \text{BesideWall(Wall(c2,c3),Left),t)} \leftarrow \\ &\qquad \text{HoldsAt(AtCorner(c1,Left),t)} \wedge \\ &\qquad\quad \text{Door(d,c1,c2)} \wedge \text{NextCorner(r,c2,c3)}\end{aligned} \tag{K4.7}$$

$$\text{Terminates(GoStraight,AtCorner(c,s),t)} \tag{K4.8}$$

Finally we have the Turn action. Since the robot has to hug the left wall, it always turns left (or goes straight) at outer corners, and always turns right at inner corners. If it turns left at the corner of a doorway, it ends up in the doorway.

$$\begin{aligned}&\text{Initiates(Turn(Left),InDoorway(d,r),t)} \leftarrow \\ &\quad \text{HoldsAt(AtCorner(c1,Left),t)} \wedge \text{Door(d,c1,c2)} \wedge \\ &\qquad \text{HoldsAt(DoorOpen(d),t)} \wedge \text{NextCorner(r,c1,c2)}\end{aligned} \tag{K4.9}$$

If the robot turns left when in a doorway, it ends up alongside a wall in the next room.

$$\begin{aligned}&\text{Initiates(Turn(Left),} \\ &\quad \text{BesideWall(Wall(c2,c3),Left),t)} \leftarrow \\ &\qquad \text{HoldsAt(InDoorway(d,r1),t)} \wedge \text{Connects(d,r1,r2)} \wedge \\ &\qquad\quad \text{Door(d,c1,c2)} \wedge \text{NextCorner(r2,c2,c3)}\end{aligned} \tag{K4.10}$$

If the robot turns right at an inner corner, it ends up next to a new wall.

$$\begin{aligned}&\text{Initiates(Turn(Right),} \\ &\quad \text{BesideWall(Wall(c1,c2),Left),t)} \leftarrow \\ &\qquad \text{HoldsAt(AtCorner(c1,Left),t)} \wedge \\ &\qquad\quad \text{Inner(c1)} \wedge \text{NextCorner(r,c1,c2)}\end{aligned} \tag{K4.11}$$

The mutual exclusivity of the AtCorner, InDoorway and BesideWall fluents is preserved by the following formulae.

$$\text{Terminates(Turn(s1),AtCorner(c,s2),t)} \tag{K4.12}$$

$$\begin{aligned}&\text{Terminates(Turn(s),InDoorway(d,r),t)} \leftarrow \\ &\quad \text{HoldsAt(InDoorway(d,r),t)}\end{aligned} \tag{K4.13}$$

4.3 The Effect of the World on the Robot

Having axiomatised the effects of the robot's actions on the world, now we need to formalise the impact the world has on the robot's sensors. For this purpose, we introduce two new types of event. The event GoesHigh(s) occurs if the average value of the two sensors in direction s exceeds a threshold $\delta 1$, where s is Left, Right or Front. Similarly the event GoesLow(s) occurs if the average value of the two sensors in direction s goes below a threshold $\delta 1$. (By making $\delta 1 > \delta 2$, we avoid a chatter of GoesHigh and GoesLow events when the robot approaches an obstacle.)

$$\text{Happens(GoesHigh(Front),t)} \leftarrow \quad \text{(S4.1)}$$
$$\text{Happens(FollowWall,t)} \wedge$$
$$\text{Initiates(FollowWall,AtCorner(c,s),t)} \wedge \text{Inner(c)}$$

$$\text{Happens(GoesLow(Front),t)} \leftarrow \quad \text{(S4.2)}$$
$$\text{HoldsAt(AtCorner(c,Left),t)} \wedge$$
$$\text{Inner(c)} \wedge \text{Happens(Turn(Right),t)}$$

$$\text{Happens(GoesHigh(s),t)} \leftarrow \quad \text{(S4.3)}$$
$$\text{HoldsAt(AtCorner(c,s),t)} \wedge \text{Outer(c)} \wedge$$
$$[\text{Happens(GoStraight,t)} \vee \text{Happens(Turn(s),t)}]$$

$$\text{Happens(GoesLow(s),t)} \leftarrow \quad \text{(S4.4)}$$
$$\text{Happens(FollowWall,t)} \wedge$$
$$\text{Initiates(FollowWall,AtCorner(c,s),t)} \wedge \text{Outer(c)}$$

Our overall aim, of course, is to use abduction to explain the occurrence of GoesHigh and GoesLow events. In the present example, if the doors are all initially open and never subsequently closed, every sensor event is predicted by the theory as it stands, so no explanation is required. The interesting case is where there are sensor events which can only be explained by a closed door. Accordingly, we need to introduce the events OpenDoor(d) and CloseDoor(d), with the obvious meanings.

$$\text{Initiates(OpenDoor(d),DoorOpen(d),t)} \quad \text{(K4.14)}$$

$$\text{Terminates(CloseDoor(d),DoorOpen(d),t)} \quad \text{(K4.15)}$$

Finally, we need some uniqueness-of-names axioms.

$$\text{UNA[FollowWall, GoStraight, Turn,} \quad \text{(U4.1)}$$
$$\text{GoesHigh, GoesLow, OpenDoor, CloseDoor]}$$

$$\text{UNA[BesideWall, AtCorner, InDoorway,} \quad \text{(U4.2)}$$
$$\text{DoorOpen]}$$

The UNA notation is defined as follows. Let f_1 to f_k be function symbols. Then UNA[$f_1, f_2, \ldots, f_k$] abbreviates the conjunction of the formulae,

$$f_i(x_1, x_2, \ldots x_m) \neq f_j(y_1, y_2, \ldots, y_n)$$

for all i < j < k, and,

$$f_i(x_1, x_2, \ldots x_n) = f_i(y_1, y_2, \ldots, y_n) \rightarrow$$
$$[x_1 = y_1 \wedge x_2 = y_2 \wedge \ldots \wedge x_n = y_n]$$

for all i < k.

4.4 An Example Narrative

Now let's examine the narrative of robot actions and sensor events for the example of Section 1.1. The following formulae describe the initial situation.

$$\text{Initially}_P\text{(DoorOpen(d))} \quad \text{(N4.1)}$$

$$\text{Initially}_P\text{(BesideWall(w,s))} \leftrightarrow \quad \text{(N4.2)}$$
$$\text{w = Wall(C18,C19)} \wedge \text{s = Left}$$

$$\text{Initially}_N\text{(AtCorner(c,s))} \quad \text{(N4.3)}$$

$$\text{Initially}_N\text{(InDoorway(d,r))} \quad \text{(N4.4)}$$

The robot follows the wall to its left until it arrives at corner C19, where it turns right and follows the wall to its left again.

$$\text{Happens(FollowWall,T1)} \quad \text{(N4.5)}$$

$$\text{Happens(Turn(Right),T2)} \quad \text{(N4.6)}$$

$$\text{Happens(FollowWall,T3)} \quad \text{(N4.7)}$$

$$\text{T1} < \text{T2} \quad \text{(N4.8)}$$

$$\text{T2} < \text{T3} \quad \text{(N4.9)}$$

Now let's suppose someone closes door D4 shortly after the robot sets out, and consider the incoming sensor events. The robot's front sensors go high at time T1, when it arrives at corner C19. (Recall that the FollowWall action is considered instantaneous.) They go low when it turns, then (unexpectedly) go high again, when it arrives at corner C22, having bypassed door D4.

$$\text{Happens(GoesHigh(Front),T1)} \quad \text{(D4.1)}$$

$$\text{Happens(GoesLow(Front),T2)} \quad \text{(D4.2)}$$

$$\text{Happens(GoesHigh(Front),T3)} \quad \text{(D4.3)}$$

The above formulae only describe the sensor events that *do* occur. But in general, we want explanations of sensor data to exclude those sensor events that have *not* occurred. Hence we have the following definition, which captures the completion of the Happens predicate for sensor events.

Definition 4.4.1.

$$\text{COMP}[\Psi] \equiv_{def}$$
$$[\text{Happens(a,t)} \wedge$$
$$[\text{a = GoesHigh(s)} \vee \text{a = GoesLow(s)}]] \rightarrow$$
$$\bigvee_{\langle\alpha,\tau\rangle \in \Pi} [\text{a} = \alpha \wedge \text{t} = \tau]$$

where $\Pi = \{\langle\alpha,\tau\rangle \mid \text{Happens}(\alpha,\tau) \in \Psi\}$ □

The following formula is one possible explanation of the above sensor events.

$$\text{Happens(CloseDoor(D4),t)} \wedge 0 \leq \text{t} < \text{T30} \quad \text{(E4.1)}$$

This is expressed by the following proposition. Let,

- Σ be the conjunction of (K4.1) to (K4.15),
- Δ_N be the conjunction of (N4.1) to (N4.9),
- Δ_T be the conjunction of (S4.1) to (S4.4),
- Δ_E be formula (E4.1),
- Φ be the conjunction of the formulae representing the robot's environment, as described in Section 4.1,
- Ω be the conjunction of (U4.1) and (U4.2), and
- Γ be the conjunction of (D4.1) to (D4.3).

Proposition 4.4.1.

$$\text{CIRC}[\Sigma \text{ ; Initiates, Terminates, Releases}] \wedge$$
$$\text{CIRC}[\Delta_N \wedge \Delta_T \wedge \Delta_E \text{ ; Happens}] \wedge$$
$$\text{EC} \wedge \Omega \wedge \Phi \vDash \text{COMP}[\Gamma].$$ □

In general, a collection of sensor data can have many explanations. Explanations can be ordered using a preference criterion, such as one which favours explanations with few events. But there can still be many mimimal explanations. In these circumstances, the robot can simply proceed on the assumption that the first explanation it finds is the true explanation. It's reasonable to expect that, if the explanation is indeed false, the processing of subsequent sensor data will reveal this. But obviously this topic merits further investigation.

In the present experiment, the policy of adopting the first explanation can lead to faulty explanations if there is more than one door on the same wall. This is because there would then be no way to distinguish between two competing explanations — one involving the first door being closed, and one involving the second door being closed. This problem can be remedied by the use of distance information to select between the competing explanations.

The next section spells out how the planning and perception processes are embedded in the overall sense-plan-act cycle that controls the robot.

5 Interleaving Sensing, Planning, and Acting

In the implemented system, the planning and perception processes each carry out a single resolution step before suspending and going around the cycle again. The perception task can be thought of as a producer of explanations in the form of Happens formulae, which are consumed by the planning process. The planner treats them in exactly the same way as it treats new steps in the plan it's generating, which are also Happens formulae. Therefore, these incoming explanations can violate the plan's "protected links" (previously proved ¬ Clipped formulae). When this occurs, the system replans from scratch.

Consider the example of Section 1.1. In response to the GoesHigh(Front) sensor event the robot receives when it encounters corner C22, the perception process generates a formula of the form,

$$\text{Happens}(\text{CloseDoor}(\text{D4}),\tau)$$

as described in the previous section. When this is assimilated by the planning process, it violates a protected link of the form,

$$\neg\,\text{Clipped}(\tau 1,\text{DoorOpen}(\text{D4}),\tau 2)$$

where $\tau 1 < \tau < \tau 2$. This precipitates replanning, whereupon the planner finds the alternative route via room R2.

As it stands, the planner of Section 3 produces actions in regression order. That is to say, the last action to be carried out is generated first. This means that, if interrupted, the planner's partial results are useless. What we require instead is a progression planner — one that generates the earliest action of a plan first. If a progression planner is interrupted, its partially constructed plan will contain actions that can be executed immediately.

One way to generate plans in progression order is via *hierarchical planning*. This is the approach adopted here. The foregoing logical treatment of partial order planning can be straightforwardly extended to planning via hierarchical decomposition. Compound action definitions are introduced, and the abductive definition of planning can be retained as is.

As an example, here's the definition of a GoToRoom action in terms of the GoThrough action from Section 3. The term GoToRoom(r1,r2) denotes the action of going from room r1 to room r2. (In the implemented system, the GoThrough action itself is broken down into FollowWall and Turn actions in a similar way.)

$$\text{Happens}(\text{GoToRoom}(r,r),t,t) \qquad \text{(H5.1)}$$

$$\begin{aligned}&\text{Happens}(\text{GoToRoom}(r1,r3),t1,t3) \leftarrow \qquad \text{(H5.2)}\\&\quad \text{Connects}(d,r1,r2) \wedge \text{Happens}(\text{GoThrough}(d),t1) \wedge\\&\qquad \text{Happens}(\text{GoToRoom}(r2,r3),t2,t3) \wedge\\&\qquad\quad t1 < t2 \wedge \neg\,\text{Clipped}(t1,\text{InRoom}(r2),t2)\end{aligned}$$

$$\begin{aligned}&\text{Initiates}(\text{GoToRoom}(r1,r2),\text{InRoom}(r2),t) \leftarrow \qquad \text{(R5.1)}\\&\quad \text{HoldsAt}(\text{InRoom}(r1),t)\end{aligned}$$

In effect, when implemented via abductive logic programming, these clauses carry out a forward search, in contrast to the backward search effected by the clauses in Section 3. The clauses used in the implemented system incorporate a heuristic to give more direction to the search.

In general, if the effects of a compound action follow from the effects of its sub-actions, it adds little to the formalisation, logically. But, if they are defined in the right way, the presence of compound actions will adjust the computation so that it generates actions in progression order. Specifically, the earliest component action in a compound action definition must be the closer to a low level action than its successors. In (H5.2), for example, the GoThrough action is at a lower level than the GoToRoom action.

Moreover, the ability of hierarchical decomposition to quickly generate a first action in response to a situation justifies the use of a replan-from-scratch strategy rather than a more sophisticated replanning technique.

Formulae (H5.1) and (H5.2) illustrate both conditional decomposition and recursive decomposition: a compound action can decompose into different sequences of sub-actions depending on what conditions hold, and a compound action can be decomposed into a sequence of sub-actions that includes a compound action of the same type as itself. A consequence of this is that the event calculus with compound actions could be used to implement a universal Turing machine, and is therefore formally as powerful as any programming language. In this respect, it can be used in the same way as GOLOG [Levesque, *et al.*, 1997]. Note, however, that we can freely mix direct programming with planning from first principles.

6 Related Work

The title of this paper, "Reinventing Shakey", alludes to the fact that the logic-based approach to robotics was first seriously attempted in the Shakey project in the late Sixties [Nillson, 1984]. One the successes for which the Shakey project is well-known was the STRIPS approach to planning [Fikes & Nilsson, 1971]. The STRIPS planner arose out of dissatisfaction with Green's earlier attempts to use resolution-based theorem proving for planning [Green, 1969]. In many ways, Green's work is more akin to modern cognitive robotics than STRIPS. We'll return to STRIPS shortly. But first, let's consider how contemporary work in cognitive robotics is an advance on Green's efforts. Why is it not subject to the same pitfalls?

There are two main differences bewteen contemporary cognitive robotics and Green's work. First, Green's approach to planning was beset by the frame problem. But we now have a number of satisfactory solutions to the frame problem that can be deployed in cognitive robotics [Shanahan, 1997a]. Second, the Shakey project relied on full search-based planning from first principles, which is computationally very expensive. Like other recent work in cognitive robotics [Levesque, *et al.*, 1997], the approach presented here uses chiefly pre-compiled plans – programs, effectively. In the present approach, this also facilitates reactivity, another feature notably lacking in Shakey.

Let's return briefly to STRIPS. The STRIPS planning algorithm is now of purely historical interest, as it has long been superseded by more efficient techniques. But the STRIPS language — the language in which planning problems are described — has had lasting influence on the planning community. One aim of contemporary cognitive robotics is to return to logic as a planning language. This move can be justified in many ways — logic is a more expressive language, logic is a *lingua franca* used in other areas of AI, logic has a clear semantics with well-understood mathematical properties, and so on.

However, the logic-based approach to robotics faces a number of challenges. First, with respect to the present work, a major question is how to scale up. The Khepera robots have very poor sensors, and can only carry out wall-following, or similar basic navigational operations. When we move up to richer sensors, such as sonar or vision, how will the techniques of this paper be adapted? Likewise, the Kheperas in the experiments reported here inhabit a simple, static, uniform environment, of just the sort that Brooks criticises [Brooks, 1991]. How will logic fare when confronted with a complex, dynamic, messy environment?

One approach is to employ an architecture that cleanly separates low- and high-level issues, both on the control front and on the sensing front. This is the approach taken by the Toronto group [Lespérance, *et al.*, 1994]. Then, the question of how to deal with complex sensor data and motor control can be pushed into the lower level, where off-the-shelf techniques can be employed. But ideally, the line between low-level and high-level, between the level of perception and control and the logical level, should be drawn as low as possible, to allow logical reasoning as much influence as possible. So more work is required on topics such as how to move directly from raw sensor data to logical representations of low-level detail of the environment, such as the shapes of obstacles.

From a methodological point of view, the present work falls somewhere between the robotics work carried out at the University of Texas [Baral & Tran, 1998] and the early work carried out at the University of Toronto [Lespérance, *et al.*, 1994]. In the Toronto work, there is no planning, except as a last resort. Instead, the robot directly executes a program written in Golog [Levesque, *et al.*, 1997], a language based on the situation calculus. A planner is invoked only if the plan monitor detects a failure, in which case planning is used to get the world into a state from which execution of the program can resume [De Giacomo, *et al.*, 1998]. Therefore, the course of actions the robot is to take is worked out almost entirely in advance, and is fixed in the program the robot executes.

More recently, the Toronto group have developed a variant of Golog called ConGolog [De Giacomo, *et al.*, 1997]. ConGolog incorporates facilities for concurrent execution, interrupt handling, and dealing with exogenous actions. Using ConGolog, the precise course of actions taken by the robot depends on conditions and events at run-time, resulting in more reactivity. However, in neither Golog nor ConGolog is there any notion of a goal with respect to which a program can be proved correct. In the Texas work [Baral & Tran, 1998], by contrast, the effects of actions are specified using logic, and the resulting theory is used to generate, off-line, a set of provably correct condition-action control rules which are then used to direct the robot. The course of actions carried out by the robot is, therefore, largely determined at run-time, and the resulting behaviour is highly reactive.

The present work differs from each of these approaches. In contrast to both the Toronto and Texas approaches, the work reported here uses on-line planning, and there is a clear notion of a goal, which must be entailed by the plan. However, the heavy use of hierarchical planning means that most of the time the robot is, in effect, executing a program. On the other hand, the constant checking of protected links (negated Clipped literals) during execution, which can precipitate rapid replanning, means that the robot is reactive to unexpected changes, like a ConGolog program or the Texas robot. The aim is to combine the advantages of planning, direct programming, and reactive control rules in a uniform, logic-based architecture.

Concluding Remarks

To summarise, the aim of the ongoing work reported here is to design and build theoretically well-founded, general purpose systems for high-level robot control, in which each computational step is also a step of logical inference, and each computational state has declarative meaning. Needless to say, the ideas presented merit a good deal of further study, and although preliminary results are promising, it remains to be seen whether they will scale up to robots with richer sensors in more realistic environments.

Acknowledgments

Thanks to Jack Minker, Mark Witkowski, and two anonymous referees. This work was carried out as part of EPSRC project GR/L20023 "Cognitive Robotics".

References

[Baral & Tran, 1998] C.Baral and S.C.Tran, Relating Theories of Actions and Reactive Control, *Linköping Electronic Articles in Computer and Information Science*, vol. 3 (1998), no. 9.

[Brooks, 1991] R.A.Brooks, Intelligence Without Reason, *Proceedings 1991 International Joint Conference on Artificial Intelligence (IJCAI 91)*, Morgan Kaufmann, pages 569-595.

[De Giacomo, *et al.*, 1997] G. De Giacomo, Y.Lespérance, H.Levesque, Reasoning about Concurrent Execution, Prioritized Interrupts, and Exogenous Actions in the Situation Calculus, *Proceedings 1997 International Joint Conference on Artificial Intelligence (IJCAI 97)*, Morgan Kaufmann, pp. 1221–1226.

[De Giacomo, *et al.*, 1998] G. De Giacomo, R.Reiter and M.Soutchanski, Execution Monitoring of High-Level Robot Programs, *Proceedings 1998 Knowledge Representation Conference (KR 98)*, Morgan Kaufmann, pp. 453–464.

[Fikes & Nilsson, 1971] R.E.Fikes and N.J.Nilsson, STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence*, vol. 2 (1971), pp. 189–208.

[Green, 1969] C.Green, Applications of Theorem Proving to Problem Solving, *Proceedings 1969 International Joint Conference on Artificial Intelligence (IJCAI 69)*, pp. 219–240.

[Kautz & Selman, 1996] H.Kautz and B.Selman, Pushing the Envelope: Planning, Propositional Logic and Stochastic Search, *Proceedings 1996 American Association for Artificial Intelligence Conference (AAAI 96)*, MIT Press, pp. 1194–1201.

[Lespérance, *et al.*, 1994] Y.Lespérance, H.J.Levesque, F.Lin, D.Marcu, R.Reiter, and R.B.Scherl, A Logical Approach to High-Level Robot Programming: A Progress Report, in *Control of the Physical World by Intelligent Systems: Papers from the 1994 American Association for Artificial Intelligence Fall Symposium*, ed. B.Kuipers, New Orleans (1994), pp. 79–85.

[Levesque, *et al.*, 1997] H.Levesque, R.Reiter, Y.Lespérance, F.Lin and R.B.Scherl, GOLOG: A Logic Programming Language for Dynamic Domains, *The Journal of Logic Programming*, vol. 31 (1997), pp. 59–83.

[Nilsson, 1984] N.J.Nilsson, ed., *Shakey the Robot*, SRI Technical Note no. 323 (1984), SRI, Menlo Park, California.

[Shanahan, 1997a] M.P.Shanahan, *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*, MIT Press (1997).

[Shanahan, 1997b] M.P.Shanahan, Noise, Non-Determinism and Spatial Uncertainty, *Proceedings 1997 American Association for Artificial Intelligence Conference (AAAI 97)*, MIT Press, pp. 153–158.

[Shanahan, 1997c] M.P.Shanahan, Event Calculus Planning Revisited, *Proceedings 4th European Conference on Planning (ECP 97)*, Springer Lecture Notes in Artificial Intelligence no. 1348 (1997), pp. 390–402.

[Shanahan, 1999] M.P.Shanahan, The Event Calculus Explained, in *Artificial Intelligence Today*, eds. M.J.Wooldridge & M.Veloso, Springer-Verlag Lecture Notes in Artificial Intelligence no. 1600, Springer-Verlag (1999), pages 409-430.

[Shanahan, 2000] M.P.Shanahan, An Abductive Event Calculus Planner, *The Journal of Logic Programming*, to appear.

[Shanahan & Witkowski, 2000] M.P.Shanahan and M.Witkowski, Event Calculus Planning Through Satisfiability, submitted.