What robots can do: Robot programs and effective achievability

Fangzhen Lin
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
flin@cs.ust.hk

Hector Levesque
Department of Computer Science
University of Toronto
Toronto, Canada M5S 3H5
hector@ai.toronto.edu

May 15, 1997

Abstract

In this paper, we propose a definition of goal achievability: given a basic action theory describing an initial state of the world and some primitive actions available to a robot, including some actions which return binary sensing information, what goals can be achieved by the robot? The main technical result of the paper is a proof that a simple robot programming language is universal, in that any effectively achievable goal can be achieved by getting the robot to execute one of the robot programs. The significance of this result is at least two fold. First, it is in many ways similar to the equivalence theorem between Turing machines and recursive functions, but applied to robots whose actions are specified by an action theory. Secondly, it provides formal justifications for using the simple robot programming language as a foundation for our work on robotics.

1 Introduction

Imagine that in the not too distant future, you are given a robot of some sort, and that you want to figure out what it can do. Browsing through the manual that came with it, you discover that the robot is capable of performing under computer control any of a set of primitive actions a_1, ..., a_n. According to the manual, what each action a_i actually does depends on the state of the environment. First, to complete successfully, a precondition of the action must hold in the environment. Next, assuming the action is successful, its effect on the environment may also depend on certain other conditions. Finally, some of the actions are connected to sensors and can return a binary value indicating when a certain condition holds. The question is: assuming we are willing to do some programming, what do we expect to be able to achieve with the robot?

In this paper, we propose an answer to this question. Specifically, we propose an abstract characterization of what goals are *effectively achievable* as a function of a given logical theory describing the initial state of the world and the primitive actions available to the robot. The main contribution of the paper is a precise technical framework where questions of goal achievability can be posed and answered. The main technical result is a proof of the *universality* of the simple robot programming language introduced in [8]: it will turn out that a goal is effectively achievable according to logical theory T iff there is a robot program that achieves it according to T.

1.1 A motivating example

To make the problem more concrete, imagine that you are also given a solid steel box that contains a treasure. There is a small robot-sized door on the box, which is currently closed, and there are two buttons beside it, a green one and a red one. The primitive actions available to the robot are *pressGreen*, *pressRed*, and *fetch*. The manual says that if the robot happens to be beside the closed door, *pressGreen* causes the green button to be pressed, and *pressRed* similarly. The manual also says that the robot has a heat sensor so that a press action returns 1 when the button pressed was hot, and 0 otherwise. The *fetch* action causes the robot to trundle inside the box and retrieve what's inside, provided that the robot is beside the door and the door is open.[1] The goal we are interested in here, obviously, is getting the treasure, under assumptions like the following:

1. If we know nothing else about the environment, we want our account of achievability to predict that we cannot achieve the goal. Of course, we might end up eventually getting the treasure by forcing the door open with a crowbar, or by saying some magic

[1] In a more realistic setting, of course, there would be a large number of other preconditions for actions like these.

words, or even by getting the robot to press the buttons in some order. But there is no reason to believe *a priori* that any of these methods will work.

2. If we know that the red button opens the door of the box, we want our account of achievability to say that we can achieve the goal using the robot: we get it to do the sequence *pressRed*, then *fetch*. Of course, something might go wrong: the door might jam, lightning might strike the robot, a comet might hit the earth. But there is no reason to believe that the sequence will fail given what we have been told.

3. If we know that one of the buttons opens the door of the box, and the other button locks the door permanently, but we don't know which is which, our account should predict that we cannot achieve the goal using the robot. As in (2), we know that there is a sequence of actions that will work – press one of the buttons then *fetch* – but here we do not know what that sequence is.

4. But consider the following situation: we know that the door can be opened by first pressing the green button, and then pressing one more button, but we are not told which, and again, getting it wrong locks the door permanently. However, suppose that we know that the safe will lock forever iff the robot pushes a button that felt hot on the previous press. As in (3), we know that there is a sequence of actions that will work, and again we cannot say what that sequence is. This time, however, our account should predict that we can achieve the goal: we get the robot to *pressGreen*, and then *pressGreen* once more if the button was cold, but *pressRed* if it was hot.

5. Finally, suppose we know that after pressing the green button some unspecified number of times and at least once, pressing the red button will open the door and pressing the green one will lock it forever. With no other information, we clearly cannot obtain the treasure. However, if we also know as in (4) that the door will lock forever iff the robot presses a button that was just hot, then we can once again achieve the goal: we get the robot to repeatedly press the green button until it feels hot, then press the red one to open the door, and then fetch the treasure.

To the best of our knowledge, there is as yet no formal framework that would give the intuitively correct answers for examples like these.

1.2 Relation to other work

There are, however, three areas of research that come close to providing these answers.

Planning As the five examples above illustrate, the idea of a goal being achievable by a robot is clearly related to the concept of planning and especially, given the sensing, conditional planning, as in [2, 9, 13, 16]. In all of the variants above, we ended up saying that the treasure was obtainable precisely when we could formulate some sort of plan to obtain it. Why then not simply define goal achievability in terms of the existence of a plan?

The problem with this involves characterizing exactly what we mean by a plan. An obvious case is when a fixed sequence of actions is sufficient. But in some of the variants above, we needed to consider conditional and iterative plans, which suggests a structure more like that of a program [10]. Clearly these would not be programs in a traditional language like C or LISP. For one thing, the primitive statements of the program would have to involve the actions a_i, rather than the usual variable assignment or read/write statements. What would we use as the conditions in an if-then-else or a while-loop statement? How should the execution of programs containing the a_i be defined?

We believe that these questions can be resolved and that it is possible to characterize achievability in terms of such programs (see Section 4 below) . However, to avoid making design decisions that might initially appear to be arbitrary or restrictive, we prefer to first define achievability in a general program-independent way, and then *prove* that a programming language is adequate according to this definition.

Computability A second concept related to achievability is that of effective computability [12]. As will become clear, we will end up defining achievable goals as those where what to do next to achieve them, given what is known about the actions and the initial state of the world, can be "computed" as a function of what the sensors tell the robot.

However, we cannot simply use an existing account of computability for two reasons. First, we want to allow for incomplete information about the environment surrounding the robot. In contrast to typical accounts of computability, the initial state of the environment need only be partially specified by a collection of axioms. The second reason concerns the primitive actions. In typical computability models, the available actions are predefined and *internal* to the machine (or formalism). For instance, we might have actions to write and read a Turing machine tape, or to increment and decrement registers, or to assign values to variables, and so on. In our case, by contrast, the primitive actions for a robot are not predefined and are *external*, in that they have effects in the environment outside of the robot. These actions are also described by a collection of axioms, which specify the action preconditions and effects, and deal with the frame problem.

Thus our account of goal achievability depends crucially on what the given axioms say about the initial state and the available actions. In some of the examples above, we had two theories $T_1 \subseteq T_2$ describing the same initial state and set of actions. A goal was considered unachievable relative to the information provided by T_1, but achievable relative to T_2 where additional information was available. We would like to define a notion of goal achievability as a *relation* between a formal theory T and the goals we would like to achieve, and no existing account of computability does this.

Knowing how Finally, the concept of achievability is very closely related to the concept of an agent knowing how (or being able to) achieve a goal or execute a plan, as discussed for example, in [1, 17, 18]. One difference between the two concepts concerns the issue of effectiveness. As far as we know, no existing account of knowing how or ability considers whether or not the know how of an agent would be effective, in the sense of allowing the agent to "compute" what to do. But putting effectiveness aside, there is also a difference in point of view: who has to know what and when. There may be conditions that we would consider to be achievable, but that the agent does not know how to bring about. For example, if the red button opens the door, *we* know that goal of getting the treasure is achievable by the agent/robot; but if the agent does not know which button is the correct one, we would not say that *it* knew how to get the treasure. Conversely, we can imagine a situation where we do not consider the goal to be achievable (in the sense of being able to produce a plan) because we do not know which buttons to use, but where we know that the agent does. We can also imagine situations where the agent initially knows less than we do, but after obtaining information from its sensors, knows as much or more than we do.

When reasoning about what one agent knows about another, the concept of knowing-how or ability may be the more useful one; when attempting to analyze what we can get an agent or robot to do for us, our notion of goal achievability may be the more appropriate. Moreover, it ought to be the case that the two notions coincide when the agent knows exactly what we do about the environment and the actions. The precise relation between the two concepts is subtle, however, and we will not explore it further here (see [6]).

In sum, while the concept of goal achievability is clearly related to the areas of planning, computability, and agent ability, none of these can give us the answers we want, for example, in the five situations above.

The rest of the paper is organized as follows. In the next section, we review the situation calculus, a formal logical language in terms of which the state of the environment and the primitive actions can be described by a collection of axioms we call a basic action theory. In Section 3, we define precisely what we mean by effective achievability (and related notions) as a function of a given basic action theory. In Section 4, we review the syntax and semantics of a simple robot programming language first proposed in [8] as a language for plans. In Section 5, we present some results, including the main technical result of the paper: the universality of the robot programming language. This is a robot analogue of the classic universality result in computability theory: a function is computable iff there is a program/machine that computes it. Finally, in Section 6, we summarize the paper and suggest topics for further research.

2 The situation calculus and basic action theories

Since the goal of this research is to make the specification of goal achievability depend on a given action theory T describing the initial state of the world and the available actions, we need to describe the representation language we use to formulate the theories, which is a dialect of the situation calculus [11].

The language of the situation calculus is many-sorted. Normally, there is a sort for situations, a sort for actions, and a sort for objects like blocks and people that are elements in the domain of interest. We assume that there is a special constant S_0 used to denote the *initial situation*, namely that situation in which no actions have yet occurred; there is a distinguished binary function symbol *do* where *do*(a, s) denotes the successor situation to s resulting from performing the action a; relations whose truth values vary from situation to situation, are called (relational) *fluents*, and are denoted by predicate symbols taking a situation term as their last argument; there is a special predicate *Poss*(a, s) used to state that action a is executable in situation s; and finally, there is a special predicate *SF*(a, s) used to state that the sensor associated with action a (if any) returns the value 1 in situation s.[2]

Within this language, we can formulate domain theories which describe the initial state of the world and the actions available to the robot. We specify the preconditions of actions, for example, by writing axioms that define *Poss*; we specify the condition measured by a sensor by writing axioms that define *SF*, and so on. Here, we use a theory which contains only the following axioms:

- Axioms describing the initial situation, S_0. Syntactically, these axioms cannot mention any other situation terms except S_0.

- Action precondition axioms, one for each primitive action A, characterizing *Poss*(A, s). Syntactically, these axioms all have the following form:

$$\mathit{Poss}(A, s) \equiv \Psi_A(s),$$

  where $\Psi_A(s)$ is a formula that does not mention any other situation terms except s, does not quantify over situation variables, and does not mention the special predicates like $Poss$, SF, or $<$ (introduced below).

- Sensed fluent axioms, one for each primitive action A, characterizing *SF*(A, s). Syntactically, these axioms have the form:

$$\mathit{SF}(A, s) \equiv \Pi_A(s),$$

  where $\Pi_A(s)$ satisfies the same conditions as those for $\Psi_A(s)$ above. For actions that have nothing to do with sensors, this should be $[\mathit{SF}(A, s) \equiv \mathit{True}]$.

- Successor state axioms, one for each fluent F, characterizing under what conditions $F(\vec{x}, \mathit{do}(a, s))$ holds as function of what holds in situation s. These take the place of

[2] In [8], the predicate *SF* was used to characterize what an agent knew in a situation in terms of a fluent K. In this paper, we will not be concerned with the knowledge of agents.

the so-called effect axioms, but also provide a solution to the frame problem [14]. Syntactically, successor state axioms have the form:

$$Poss(a, s) \supset [F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)],$$

where Φ_F satisfies the same conditions as those for $\Psi_A(s)$ above.

- Unique names axioms for the primitive actions: For any two different actions $A(\vec{x})$ and $A'(\vec{y})$, we have

  $$A(\vec{x}) \neq A'(\vec{y}),$$

  and for any action $A(x_1, ..., x_n)$, we have

  $$A(x_1, ..., x_n) = A(y_1, ..., y_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n.$$

- Foundational, domain independent axioms that characterize the structure of the space of situations, and define a predicate $\leq$ so that $s_1 \leq s_2$ holds iff there are actions $a_1, \ldots, a_n$, $0 \leq n$, such that

  $$s_2 = do([a_1, \ldots, a_n], s_1) \wedge Poss(a_1, s_1) \wedge \ldots \wedge Poss(a_n, do([a_1, \ldots, a_{n-1}], s_1))$$

  holds, where for any situation s, $do([], s) = s$, and inductively, $do([a|L], s) = do(L, do(a, s))$. These axioms are:

  $$S_0 \neq do(a, s),$$

  $$do(a_1, s_1) = do(a_2, s_2) \supset (a_1 = a_2 \wedge s_1 = s_2),$$

  $$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s),$$

  $$\neg s < S_0,$$

  $$s < do(a, s') \equiv (Poss(a, s') \wedge s \leq s').$$

  Notice the similarity between these axioms and Peano Arithmetic. The first two axioms are unique names assumptions; they eliminate finite cycles, and merging. The third axiom is second-order induction; it amounts to a domain closure axiom which says that every situation must be obtained by repeatedly applying do to S_0.[3] The last two axioms define $<$ inductively.

Following [4], we call a theory of this form a *basic action theory*.

[3]For a discussion of the use of induction in the situation calculus, see (Reiter [15]).

3 Effective achievability

To define in its most general form what a robot armed with primitive actions $a_1, \ldots a_n$, can achieve, it is useful to begin by looking at the problem from the point of view of a robot controller, for instance, an onboard computer.

What a robot controller needs to do at any given point in time is to select the primitive action to perform next (or to stop). We are willing to assume arbitrary amounts of computation and intelligence in making this decision, as well as full access to the given basic action theory. We do not want to assume, however, that the controller necessarily knows everything there is to know about the current state of the environment. For example, if it is part of the basic action theory that a door is open initially, the controller can use this fact; but if the action theory does not specify the state of the door, the robot may need to perform some (sensing) action to find out whether it is open, assuming such an action is available.

So what does a robot controller have access to beyond the given basic action theory? In its most general form, we might imagine that the robot controller remembers all of the actions it has selected until now, as well as the sensing results of all these actions. In general, these sensing results must be compatible with the given action theory, but will not be entailed by it, and so provide additional information to the controller.

Once we have specified what a robot controller is, we can then define the achievable goals. Roughly, a goal will be considered to be achievable if there exists a robot controller such that if we were to repeatedly do the primitive action it prescribes, then no matter how the sensing turns out, we would eventually terminate in a situation where, according to the action theory, the goal condition would hold. We now proceed to formally define the relevant notions.

3.1 Robot Controllers and Environments

We assume a finite set $\mathcal{A}$ of actions that are parameterless, and represented by constant symbols. At any point, the robot will be in some state determined by the actions it has performed so far and, in the event of sensing, the readings of its sensors. More precisely, we define:

Definition 1 (History) *A* history σ *is an element of the set* $\mathcal{R} = (\mathcal{A} \times \{0,1\})^*$.

Intuitively, the history $(\alpha_1, \beta_1) \circ \cdots \circ (\alpha_n, \beta_n)$ means that $\alpha_1, ..., \alpha_n$ is the sequence of actions performed so far, and $\beta_1, ..., \beta_n$ are the respective sensing results of the actions: For any i, if the sensing fluent SF holds for α_i in the situation where the action is performed, then $\beta_i = 1$, else $\beta_i = 0$. Notice that by the form of basic action theories (cf. section 2), if α_i is an action that has nothing to do with sensors, then $\beta_i = 1$. Notice also that the empty sequence ϵ is a history.

A robot controller is then a mapping from such a history to the next action to perform. In addition to the given primitive actions in $\mathcal{A}$, we assume some special symbols. In the following, let $\mathcal{A}^+ = \mathcal{A} \cup \{stop, abort, \perp\}$, where *stop*, *abort*, and $\perp$ are special symbols not in $\mathcal{A}$. Intuitively, *stop* will be used to denote termination, *abort* to signal exit before the end of the computation, and $\perp$ to denote an undefined computation.[4] Formally, we define:

Definition 2 (Robot controller) *A* (robot) controller $\mathcal{C}$ *is any function from histories to actions or special symbols,* $\mathcal{C} : \mathcal{R} \Rightarrow \mathcal{A}^+$.

Definition 3 (Effective Controller) *A controller is* effective *if the function* $\mathcal{C}$ *is recursive.*

It should be clear that the only feedback the robot gets from the environment is through its sensors. Just as a robot controller specifies the next action to perform, an environment specifies the sensing result of that action. More precisely, we define:

Definition 4 (Environment) *An* environment $\mathcal{E}$ *is any function from histories and actions to the set* $\{0, 1\}$, $\mathcal{E} : \mathcal{R} \times \mathcal{A} \Rightarrow \{0, 1\}$.

In other words, $\mathcal{E}(\sigma, \alpha)$ tells us what the sensor associated with action α will report given the history σ.

Intuitively, the picture is this. We start with the empty history ϵ; the robot controller $\mathcal{C}$ chooses an action to perform $\alpha_1 = \mathcal{C}(\epsilon)$; the environment $\mathcal{E}$ determines the value returned by the α_1 sensor: $\beta_1 = \mathcal{E}(\epsilon, \alpha_1)$; given this result, the robot then chooses another action to perform $\alpha_2 = \mathcal{C}((\alpha_1, \beta_1))$, and the environment determines the α_2 sensor value: $\beta_2 = \mathcal{E}((\alpha_1, \beta_1), \alpha_2)$; then $\alpha_3 = \mathcal{C}((\alpha_1, \beta_1) \circ (\alpha_2, \beta_2))$ and so on, until $\mathcal{C}$ says *stop*.

Definition 5 (System) *A* system *is a pair* $(\mathcal{C}, \mathcal{E})$*, where* $\mathcal{C}$ *is a controller, and* $\mathcal{E}$ *an environment.*

Frequently, we shall refer to the system $(\mathcal{C}, \mathcal{E})$ as the controller $\mathcal{C}$ under the environment $\mathcal{E}$.

Definition 6 ((Terminating) run) *A history* σ *is a* run of a system $(\mathcal{C}, \mathcal{E})$ *if, inductively, either* σ *is the empty sequence* ϵ *or* $\sigma = \sigma' \circ (\alpha, \beta)$ *such that* σ' *is a run of the system,* $\mathcal{C}(\sigma') = \alpha \in \mathcal{A}$*, and* $\mathcal{E}(\sigma', \alpha) = \beta$*. A history* σ *is a* terminating run *of* $(\mathcal{C}, \mathcal{E})$ *if it is a run of* $(\mathcal{C}, \mathcal{E})$ *and* $\mathcal{C}(\sigma) = stop$.

Clearly, a system can have at most one terminating run.

[4]The reason we need *abort* and $\perp$ will be made clear later in the context of robot programs.

3.2 Achievability and Effective Achievability

Note that neither controllers nor environments are part of the situation calculus; they are simply abstract functions over the domain of histories. To make a connection with the situation calculus, we first relate histories to situations:

Definition 7 (Run and situation) *Given any history σ, and any situation term s, we define another situation term, the* end situation *of σ on s, written end(σ, s), as: end$(\epsilon, s) = s$; and inductively, if $\sigma = \sigma' \circ (\alpha, \beta)$, then end$(\sigma, s) =$ do$(\alpha,$ end$(\sigma', s))$.*

Next, we relate environments to logical interpretations of a basic action theory.

Definition 8 (Environment and interpretation) *Given an interpretation I and a ground situation term S, an environment $\mathcal{E}$ is said to be* determined by I at S *iff for any history σ, and any action α, $\mathcal{E}(\sigma, \alpha) = 1$ iff $I \models$ SF$(\alpha,$ end$(\sigma, S))$.*

It is clear that there is exactly one such environment for any given I and S. In other words, once we specify an interpretation of a basic action theory (and a starting situation), the interpretation of the *SF* predicate completely determines how the sensing will turn out, and hence the environment.

In general, we expect a basic action theory to be satisfied by many interpretations, corresponding to the various ways the environment could turn out. Goal achievability requires a controller to work in all such interpretations:

Definition 9 ((Effective) achievability) *Given an action theory T, a goal $G(s)$ which is a formula with a single free situation variable s, and a ground situation term S, we say that G is* (effectively) achievable *in S according to T iff there is an (effective) controller $\mathcal{C}$ such that for any model I of T, there is a terminating run σ of $\mathcal{C}$ under the environment determined by I at S such that $I \models S \leq$ end$(\sigma, S) \wedge G($end$(\sigma, S))$.*

In general, there will be goals that are achievable but not effectively achievable. However, as we are going to show below, for *context-free* action theories, if a goal is achievable, then it is also effectively achievable.

3.3 Achievability in Context-Free Action Theories

By a context-free action theory we mean a theory in which all actions are context free in the sense that their effects are independent of the state in which they are executed. For example, in the blocks world, the action *stack*(x, y), that picks up block x on the table and puts in on top of block y, is context free – as long as it is executable, it will always cause x to be on y. On the other hand, in the extended blocks world in which there may be more than one block on top of another block, the action *unstack*(x, y), that removes x from y, is not context free

– whether block y will be clear afterwards, for example, depends on whether x was the only block on top of y.

Now an action theory is context free if, according to the theory, all actions are context free. Formally, following [5], we call an action theory T context free if every successor state axiom in it has the following form:

$$\mathit{Poss}(a,s) \supset [F(\vec{x}, \mathit{do}(a,s)) \equiv \gamma_F^+(\vec{x},a) \vee (F(\vec{x},s) \wedge \neg\gamma_F^-(\vec{x},a))], \tag{1}$$

where $\gamma_F^+(\vec{x},a)$ and $\gamma_F^-(\vec{x},a)$ are situation independent formulas whose free variables are among those in $\vec{x}, a$. Under the following *consistency condition* [14]:

$$\Sigma \models (\forall a, \vec{x}).\neg(\gamma_F^+(\vec{x},s) \wedge \gamma_F^-(\vec{x},s)), \tag{2}$$

the axiom (1) implies that for any action a, after the action is performed, F will be true (added) for tuples in $\{\vec{x} \mid \gamma_F^+(\vec{x},a)\}$, false (deleted) for tuples in $\{\vec{x} \mid \gamma_F^-(\vec{x},a)\}$, and persist for tuples in $\{\vec{x} \mid \neg\gamma_F^+(\vec{x},a) \wedge \neg\gamma_F^-(\vec{x},a)\}$. The action a is context-free because the conditions γ_F^- and γ_F^+ are situation independent. Note that in the usual formulation of STRIPS, with add and delete lists, every action is considered to be context-free.

Theorem 1 *Let T be a context-free action theory, and the consistency condition (2) holds for every fluent F.*[5] *If a goal G is achievable in S, then it is also effectively achievable.*

Proof: See Appendix A. ■

Informally, the theorem holds because a context free action theory can only have finite number of possible legal states. One can read the theorem in two ways. On the one hand, it points to some potential computational advantages of working with context-free action theories. On the other hand, it also points out their expressive limitations. For example, this theorem implies that it is impossible to simulate an arbitrary context-sensitive action with a *finite* set of context free actions.

4 A robot program language

In [8], the following question was considered: what should the output of a planning procedure be? In the absence of sensing, the answer is reasonably clear and dates back to Green [3]: a plan is a legally executable sequence of actions that results in a final situation where the goal condition holds. In the presence of sensing, however, a planner cannot simply return a sequence of actions since the actions to execute to satisfy the goal could depend on the runtime result of earlier sensing operations (as in the examples in the introduction).

[5] Technically, this condition is not necessary. However, it simplifies our proof, and is a reasonable condition to impose on action theories.

Clearly, what is needed is something more like a program, with branches and loops. On the other hand, it would need to be a program that is not only legally executable (in the sense that the preconditions of the primitive actions at each step are satisfied), and leads to a goal state (in the sense that the program terminates and the goal condition holds in the terminating situation), but also a program that does not require more information to execute than what we expect the robot to have. For example, if all we know is that the door to the steel box opens by pushing either the red or the green button, then the program which says something like "if the red button is the one that opens the door then push it, else push the other one" might satisfy the first two conditions, but not the last one.

There are various ways to ensure this last requirement. The approach taken in [8] is to invent a simple language that contains branches and loops, but that does not mention any conditions involving fluents. The resulting programs are then trivial to execute since without such conditions, there is nothing for the robot executing the programs to know.

Consider the following simple programming language, defined as the least set of terms satisfying the following:

1. *nil* and *exit* are programs;
2. If a is an action and r_1 and r_2 are programs, then *branch*(a, r_1, r_2) is a program;
3. If r_1 and r_2 are programs, then *loop*(r_1, r_2) is a program.

We will call such terms *robot programs* and the resulting set of terms $\mathcal{R}$, the robot programming language.

Informally, these programs are executed by an agent as follows: to execute *nil* the agent does nothing; to execute *exit* it must be executing a *loop*, in which case see below; to execute *branch*(a, r_1, r_2) it first executes primitive action a, and then it executes r_1 if the sensor associated with a returns 1, and r_2 otherwise; to execute *loop*(r_1, r_2), it executes the body r_1, and if it ends with *nil*, it repeats r_1 again, and continues doing so until it ends with *exit*, in which case it finishes by executing r_2.

Note that many actions will not have an associated sensor and will always return 1. We thus use the abbreviation *seq*(a, r) for *branch*(a, r, r) for those cases where the returned value is ignored.

Here are some robot programs for the examples in the introduction. (Recall that the *pressGreen* action returns 1 if the button is hot).

1. Sequence of actions:
   seq(*pressRed*, *seq*(*fetch*, *nil*)).
2. Conditional plan:
   branch(*pressGreen*,
       seq(*pressRed*, *seq*(*fetch*, *nil*)),
       seq(*pressGreen*, *seq*(*fetch*, *nil*))).

3. Iterative plan:
   $\underline{loop}$(
   $\underline{branch}$(*pressGreen*, $\underline{exit}$, $\underline{nil}$),
   $\underline{seq}$(*pressRed*, $\underline{seq}$(*fetch*, $\underline{nil}$))).

Intuitively at least, the following should be clear:

- An agent can always be assumed to know how to execute a robot program. These programs are completely deterministic, and do not mention any fluents. Assuming the binary sensing actions return a single bit of information to the agent, there is nothing else it should need to know.
- The example robot programs above, when executed, result in final situations where the goal conditions from the introduction are satisfied.

To be precise about this, we need to first define what situation is the final one resulting from executing a robot program r in an initial situation s. Because a robot program could conceivably loop forever (e.g. $\underline{loop}(\underline{nil},\underline{nil})$), we will use a formula $Rdo(r, s, s')$ to mean that r terminates legally when started in s, and s' is the final situation. Formally, *Rdo* is an abbreviation for the following second-order formula:

$$Rdo(r, s_1, s_2) \stackrel{def}{=} \forall P[\ldots \supset P(r, s_1, s_2, 1)] \tag{3}$$

where the ellipsis is (the conjunction of the universal closure of) the following:

1. Termination, normal case: $P(\underline{nil}, s, s, 1)$;
2. Termination, loop body: $P(\underline{exit}, s, s, 0)$;
3. Primitive actions returning 1:
   $Poss(a, s) \wedge SF(a, s) \wedge P(r', do(a, s), s', x) \supset P(\underline{branch}(a, r', r''), s, s', x)$;
4. Primitive actions returning 0:
   $Poss(a, s) \wedge \neg SF(a, s) \wedge P(r'', do(a, s), s', x) \supset P(\underline{branch}(a, r', r''), s, s', x)$;
5. Loops, exit case:
   $P(r', s, s'', 0) \wedge P(r'', s'', s', x) \supset P(\underline{loop}(r', r''), s, s', x)$;
6. Loops, repeat case:
   $P(r', s, s'', 1) \wedge P(\underline{loop}(r', r''), s'', s', x) \supset P(\underline{loop}(r', r''), s, s', x)$.

By using second-order quantification in this way, we are defining *Rdo* recursively as the *least* predicate P satisfying the constraints in the ellipsis. Second-order logic is necessary here since there is no way to characterize the transitive closure implicit in unbounded iteration in first-order terms.

The relation $P(r, s, s', 0)$ in this definition is intended to hold when executing r starting in s terminates legally at s' with *exit*; $P(r, s, s', 1)$ is the same but terminating with *nil*. The difference shows up when executing *loop*(r, r'): in the former case, we exit the loop and continue with r'; in the latter, we continue the iteration by repeating *loop*(r, r') once more.

With this definition in place, we can now characterize precisely the goals that are achievable using robot programs:

> Given an action theory T, a robot program r, a goal condition $G(s)$ and a ground situation term S, we say that r *achieves* G in S according to T iff
>
> $$T \models \exists s'.Rdo(r, S, s') \wedge G(s')$$

We now relate this definition to effective achievability.

5 Robot programs are universal

Our main technical result in this paper is that a goal is achievable by an (augmented) robot program iff it is achievable by an effective controller. We shall prove this in two parts. First, we show that for any robot program, there is a corresponding effective controller for it. We then show that if 5 special "Turing machine actions" are included, then any effective controller can be simulated by a robot program.

5.1 From robot programs to effective controllers

Theorem 2 *For any robot program r and any ground situation term S, there is an effective controller $\mathcal{C}$ such that for any interpretation I and any ground situation term S':*

1. *If $I \models Rdo(r, S, S')$, then there is a terminating run σ of the system $(\mathcal{C}, \mathcal{E})$ such that $S' = end(\sigma, S)$, where $\mathcal{E}$ is the environment determined by I at S.*

2. *If there is a terminating run σ of the system $(\mathcal{C}, \mathcal{E})$ such that $end(\sigma, S) = S'$, then $I \models S \leq S' \supset Rdo(r, S, S')$. Here $\mathcal{E}$ is the environment determined by I at S.*

Proof: See Appendix B. ■

5.2 From effective controllers to robot programs

Given an effective controller, there may not always be a robot program that simulates it.[6] The easiest way to remedy this is to add some special Turing machine actions as in [8].

Formally, we assume that in addition to the actions in $\mathcal{A}$, we have 5 special actions, *left*, *right*, *mark*, *erase*, *read_mark*, and two special fluents *Marked*, *loc*, characterized by the following axioms:

1. Precondition: the 5 actions are always possible
   $Poss(left, s) \wedge Poss(right, s) \wedge Poss(mark, s) \wedge Poss(erase, s) \wedge Poss(read\_mark, s)$;

2. Successor state: only *erase* and *mark* change the *Marked* fluent
   $Poss(a, s) \supset \{Marked(n, do(a, s)) \equiv$
   $\quad a = mark \wedge loc(s) = n \quad \vee$
   $\quad Marked(n, s) \wedge \neg[a = erase \wedge loc(s) = n]\}$;

3. Successor state: only *left* and *right* change the *loc* fluent
   $Poss(a, s) \supset \{loc(do(a, s)) = n \equiv$
   $\quad a = left \wedge loc(s) = n + 1 \quad \vee$
   $\quad a = right \wedge loc(s) = n - 1 \quad \vee$
   $\quad loc(s) = n \wedge a \neq left \wedge a \neq right\}$;

4. Sensed fluent: *read_mark* tells the agent whether the current location is marked
   $SF(left, s) \wedge SF(right, s) \wedge SF(erase, s) \wedge SF(mark, s) \wedge$
   $\quad [SF(read\_mark, s) \equiv Marked(loc(s), s)]$.

These axioms ensure that the 5 special actions provide the robot with what amounts to a Turing machine tape.

In the following, we will be using what we will call a *TM basic action theory.* This is a basic action theory as before, but where $\mathcal{A}$ includes the 5 special actions, T contains the above axioms, and where the successor state axioms in T for fluents other than *loc* and *Marked* are such they are unaffected by the 5 actions. More precisely, for any fluent F that is different from *Marked* and *loc*, and when A is any of the 5 special actions, the theory T entails:

$$Poss(A, s) \supset [F(do(A, s)) \equiv F(s)].$$

In the following, for any ground situation term S, we define $clean(S)$ to be the situation term obtained from S by deleting all the special five actions: $clean(S_0) = S_0$, and

[6] The proof of this is somewhat laborious. Observe that despite the presence of loops, a robot program has no internal memory and so no way of counting. So consider an action theory that encodes a string problem: decide if a string of `0`s and `1`s has more `0`s than `1`s. A Turing machine can do this, but a finite-state automaton cannot. Similarly, an effective controller can achieve the corresponding goal, but a robot program in general cannot. We omit the details.

clean$(do(\alpha, S)) = do(\alpha, clean(S))$ if $\alpha \in \mathcal{A}$ is not a Turing action, and *clean*$(do(\alpha, S)) = clean(S)$ otherwise.

Theorem 3 *For any TM basic action theory T, any effective controller $\mathcal{C}$, and any ground situation term S, there is a robot program r such that for any model I of T (as above) and any ground situation S', we have*

1. *If $I \models Rdo(r, S, S')$, then there is a terminating run σ of the system $(\mathcal{C}, \mathcal{E})$ such that $clean(S') = end(\sigma, S)$, where $\mathcal{E}$ is the environment determined by I at S.*

2. *If there is a terminating run σ of the system $(\mathcal{C}, \mathcal{E})$ such that $S' = end(\sigma, S)$, then there is a situation S'' such that $S' = clean(S'')$, and $I \models S \leq end(\sigma, S) \supset Rdo(r, S, S'')$, where $\mathcal{E}$ is the environment determined by I at S.*

Proof: See Appendix C. ■

5.3 The main theorem

By Theorem 2 and 3, we have the following result:

Theorem 4 *Let T be any TM basic action theory and G be any goal that does not mention the special fluents loc and Marked. Then G is effectively achievable in S according to T iff there is a robot program r such that r achieves G in S according to T.*

Proof: Suppose G is achieved by the effective controller $\mathcal{C}$. We show that the robot program r for $\mathcal{C}$ as in Theorem 3 achieves G. Suppose I is any model of T. Then, by the definition of achievability, there is a terminating run σ of $(\mathcal{C}, \mathcal{E})$ such that $I \models S \leq end(\sigma, S) \wedge G(end(\sigma, S))$, where $\mathcal{E}$ is the environment determined by I and S. According to Theorem 3, there is a situation S'' such that $clean(S'') = end(\sigma, S)$ and $I \models S \leq end(\sigma, S) \supset Rdo(r, S, S'')$. Thus $I \models Rdo(r, S, S'')$. By $I \models G(end(\sigma, S))$ and $clean(S'') = end(\sigma, S)$, we have $I \models G(S'')$ (by a property about *clean*). So $I \models (\exists s')Rdo(r, S, s') \wedge G(s')$. Therefore $T \models (\exists s')Rdo(r, S, s') \wedge G(s')$, and so r achieves G.

Conversely, suppose r achieves G in S. We show that the effective controller $\mathcal{C}$ for r as in Theorem 2 achieves G in S, i.e. for any model I of T, there is a terminating run σ of the system $(\mathcal{C}, \mathcal{E})$ such that $I \models S \leq end(\sigma, S) \wedge G(end(\sigma, S))$, where $\mathcal{E}$ is the environment determined by I and S. Now suppose I is a model of T, by the assumption that r achieves G, i.e. $T \models (\exists s')Rdo(r, S, s') \wedge G(s')$, there is a ground situation term S' such that $I \models Rdo(r, S, S') \wedge G(S')$. By Theorem 2, there is a terminating run σ of $(\mathcal{C}, \mathcal{E})$ such that $S' = end(\sigma, S)$. So $I \models Rdo(r, S, end(\sigma, S)) \wedge G(end(\sigma, S))$. But by the definition of *Rdo*, we have $\models (\forall s_1, s_2)Rdo(r, s_1, s_2) \supset s_1 \leq s_2$. Thus we have $I \models S \leq end(\sigma, S)$. This shows that G is achieved by the controller $\mathcal{C}$. ■

6 Conclusion

We have provided a definition of what it might mean for a condition to be achievable by a robot relative to a given action theory which describes the initial state of the world and the primitive actions available to the robot. Our main technical contribution is in showing that this notion of effective achievability coincides with a notion of achievability by a simple class of robot programs independently introduced in [8]. The significance of this result is at least two fold. First, it is in many ways similar to the equivalence theorem between Turing machines and recursive functions, but applied to robots whose actions are specified by an action theory. Secondly, it provides formal justifications for using the simple class of robot programs as a foundation for our work on robotics. For instance, [8] uses this class of robot programs as a basis for robot planning. We are also beginning work on compiling high-level GOLOG programs [7] into this class of robot programs.

There are some limitations with our current model that are worth mentioning here. First of all, we have assumed that there are only a finite number of parameterless actions. We have also assumed that the sensing actions are binary, characterized by the special *SF* predicate. Furthermore, we have assumed that the only feedback from the environment is the result of these sensing actions. In particular, we have not concerned ourselves here with possible action failure or exogenous actions. Some of these assumptions, such as the binary nature of sensing, are easy to relax; others will require more effort.

In concluding, we want to mention that we are working on relating this work to our other work on agent ability and knowing-how [6]. Another direction worth pursuing is investigating the "finite automaton" version of achievability, *i.e.* the power of robot programs without the special Turing machine actions.

Acknowledgments

This work was done as part of the Cognitive Robotics project at the University of Toronto, and we greatly benefitted from discussions with Yves Lespérance, and Ray Reiter. Financial support for the second author was gratefully received from the Natural Sciences and Engineering Research Council of Canada, the Information Technology Research Centre of Ontario, and the Institute for Robotics and Intelligent Systems of Canada. The first author was also supported in part by grant DAG96/97.EG34 from the Hong Kong University of Science and Technology.

References

[1] E. Davis. Knowledge Preconditions for Plans. *Journal of Logic and Computation*, vol. 4, no. 5, pp. 721–766, Oct. 1994.

[2] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pp. 115–125, Cambridge, MA, 1992. Morgan Kaufmann Publishing.

[3] C. Green. Theorem proving by resolution as a basis for question-answering systems. In *Machine Intelligence 4*, pp. 183–205. Edinburgh University Press, 1969.

[4] F. Lin and R. Reiter. State constraints revisited. In *Journal of Logic and Computation*, Vol. 4, No. 5, pp. 655–678, 1994.

[5] F. Lin and R. Reiter. How to progress a database. *Artificial Intelligence*, 1997. To appear.

[6] Y. Lespérance, H. Levesque, and F. Lin. Ability and knowing how in the situation calculus. In preparation, 1997.

[7] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. To appear in the *Journal of Logic Programming*, 1996.

[8] H. Levesque. What is planning in the presence of sensing? In *Proc. of AAAI-96*, Portland, OR, to appear, Aug. 1996.

[9] K. Krebsbach, D. Olawsky, and M. Gini. An empirical study of sensing and defaulting in planning. In *Proc. of 1st Conference on AI Planning Systems*, pp. 136–144, San Mateo CA, 1992.

[10] Z. Manna and R. Waldinger. How to clear a block: A theory of plans. *Journal of Automated Reasoning*, 3:343–377, 1987.

[11] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press, 1969.

[12] E. Mendelson *An Introduction to Mathematical Logic*. Van Rostrand Reinhold Company, New York, 1964.

[13] M. Peot and D. Smith. Conditional nonlinear planning. In *Proc. of 1st Conference on AI Planning Systems*, pp. 189–197, San Mateo CA, 1992.

[14] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press, San Diego, CA, 1991.

[15] R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351, 1993.

[16] M. Schoppers. Building plans to monitor and exploit open-loop and closed-loop dynamics. In *Proc. of 1st Conference on AI Planning Systems*, pp. 204–213, San Mateo CA, 1992.

[17] S. Thomas. *PLACA, An Agent Oriented Programming Language.* Ph.D. Thesis. Department of Computer Science, Stanford University, 1993.

[18] W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. A logic of capabilities. In A. Nerode and Y. Matiyasevitch editors, *Proc. of LFCS-94, the 3rd International Symposium on the Logical Foundations of Computer Science*, pp. 366–378, Springer Verlag, 1994.

Appendix A Proof of Theorem 1

Theorem 1 *Let T be a context-free action theory, and the consistency condition (2) holds for every fluent F. If a goal G is achievable in S, then it is also effectively achievable.*

We first prove a lemma which says that the set of *legal states* in a context-free action theory is finite. To formulate the lemma, we first introduce a few notations. Let $F_1(\vec{x_1}, s), \cdots, F_n(\vec{x_n}, s)$ be all the fluents in the language. We define *SameState*(s, s'), meaning that s and s' yield the same state, as follows:

$$\begin{aligned}&\mathit{SameState}(s, s') \stackrel{def}{=} \\ &\qquad (\forall \vec{x}_1)(F_1(\vec{x}_1, s) \equiv F_1(\vec{x}_1, s')) \wedge \cdots \wedge (\forall \vec{x}_n)(F_n(\vec{x}_n, s) \equiv F_n(\vec{x}_n, s')).\end{aligned}$$

The following are some simple properties about *SameState*:

Lemma 6.1 *For any basic action theory T, we have:*

$$\begin{aligned} T &\models (\forall a, s, s').\mathit{SameState}(s, s') \supset [\mathit{Poss}(a, s) \equiv \mathit{Poss}(a, s')], \\ T &\models (\forall a, s, s').\mathit{SameState}(s, s') \supset [\mathit{SF}(a, s) \equiv \mathit{SF}(a, s')], \\ T &\models (\forall a, s, s').\mathit{Poss}(a, s) \wedge \mathit{SameState}(s, s') \supset \mathit{SameState}(\mathit{do}(a, s), \mathit{do}(a, s')). \end{aligned}$$

Proof: Trivially from the definition of basic action theories. ■

Lemma 6.2 *Let T be a context-free action theory with finite number of parameterless actions. Under the consistency condition (2), there is a natural number N such that the following set*

$$\{\| \mathit{do}(\xi, S_0) \| \ \mid \xi \textit{ is a list of actions}\}$$

contains less than N elements, where for any list of actions ξ,

$$\begin{aligned}\| \mathit{do}(\xi, S_0) \| \ = \ \{\mathit{do}(\xi', S_0) \mid T \models S_0 \leq \mathit{do}(\xi, S_0) \wedge S_0 \leq \mathit{do}(\xi', S_0) \supset \\ \mathit{SameState}(\mathit{do}(\xi, S_0), \mathit{do}(\xi', S_0))\}\end{aligned}$$

In other words, the number of possibly different legal states is bounded by N.

Proof: To simplify our proof, without loss of generality, we assume that the action theory has only two actions A and B. Consider an arbitrary fluent $F(\vec{x}, s)$. Suppose its successor state axiom is as (1). By the consistency condition (2), we have

$$T \models F(\vec{x}, \mathit{do}([], S_0)) \equiv F(\vec{x}, S_0),$$

$$T \models S_0 \leq do([A], S_0) \supset F(\vec{x}, do([A], S_0)) \equiv \begin{cases} \mathit{True} & \text{if } \gamma_F^+(\vec{x}, A) \\ \mathit{False} & \text{if } \gamma_F^-(\vec{x}, A) \\ F(\vec{x}, S_0) & \text{otherwise,} \end{cases}$$

$$T \models S_0 \leq do([B], S_0) \supset F(\vec{x}, do([B], S_0)) \equiv \begin{cases} \mathit{True} & \text{if } \gamma_F^+(\vec{x}, B) \\ \mathit{False} & \text{if } \gamma_F^-(\vec{x}, B) \\ F(\vec{x}, S_0) & \text{otherwise,} \end{cases}$$

$$T \models S_0 \leq do([A, B], S_0) \supset F(\vec{x}, do([A, B], S_0)) \equiv \begin{cases} \mathit{True} & \text{if } \gamma_F^+(\vec{x}, A) \wedge \neg\gamma_F^-(\vec{x}, B) \\ \mathit{True} & \text{if } \gamma_F^+(\vec{x}, B) \\ \mathit{False} & \text{if } \gamma_F^-(\vec{x}, A) \wedge \neg\gamma_F^+(\vec{x}, B) \\ \mathit{False} & \text{if } \gamma_F^-(\vec{x}, B) \\ F(\vec{x}, S_0) & \text{otherwise,} \end{cases}$$

$$T \models S_0 \leq do([B, A], S_0) \supset F(\vec{x}, do([B, A], S_0)) \equiv \begin{cases} \mathit{True} & \text{if } \gamma_F^+(\vec{x}, B) \wedge \neg\gamma_F^-(\vec{x}, A) \\ \mathit{True} & \text{if } \gamma_F^+(\vec{x}, A) \\ \mathit{False} & \text{if } \gamma_F^-(\vec{x}, B) \wedge \neg\gamma_F^+(\vec{x}, A) \\ \mathit{False} & \text{if } \gamma_F^-(\vec{x}, A) \\ F(\vec{x}, S_0) & \text{otherwise.} \end{cases}$$

Furthermore

$$T \models S_0 \leq do([A, A], S_0) \supset F(\vec{x}, do([A, A], S_0)) \equiv F(\vec{x}, do([A], S_0)),$$

$$T \models S_0 \leq do([B, B], S_0) \supset F(\vec{x}, do([B, B], S_0)) \equiv F(\vec{x}, do([B], S_0)),$$

$$T \models S_0 \leq do([A, B, A], S_0) \wedge S_0 \leq do([B, A], S_0) \supset$$
$$F(\vec{x}, do([A, B, A], S_0)) \equiv F(\vec{x}, do([B, A], S_0)),$$

$$T \models S_0 \leq do([A, B, B], S_0) \supset F(\vec{x}, do([A, B, B], S_0)) \equiv F(\vec{x}, do([A, B], S_0)),$$

$$T \models S_0 \leq do([B, A, A], S_0) \supset F(\vec{x}, do([B, A, A], S_0)) \equiv F(\vec{x}, do([B, A], S_0)),$$

$$T \models S_0 \leq do([B, A, B], S_0) \wedge S_0 \leq do([A, B], S_0) \supset$$
$$F(\vec{x}, do([B, A, B], S_0)) \equiv F(\vec{x}, do([A, B], S_0)).$$

Since the fluent F is arbitrary, the finiteness of the set in question follows. ■

Proof of Theorem 1 Suppose G is achievable in S according to T. We need to show that there is a recursive controller that achieves G in S according to T. In fact, we can do better. We'll show that there is a *finite* controller that achieves G. Let $\mathcal{C}$ be a robot controller that achieves G according to T, and let

$$\mathcal{S} = \{\sigma \mid \sigma \text{ is a terminating run of } \mathcal{C} \text{ under the environment determined by a model of } T \text{ at } S\}$$

Then the following controller

$$\mathcal{C}'(\sigma) = \begin{cases} \bot & \text{If } \sigma \text{ is not a prefix of any history in } \mathcal{S} \\ \textit{stop} & \text{If } \sigma \in \mathcal{S} \\ \alpha & \text{If for some } \beta,\ \sigma \circ (\alpha, \beta) \text{ is a prefix of a history in } \mathcal{S} \end{cases} \tag{4}$$

is clearly well (uniquely) defined, and achieves G in S as well. Furthermore, if $\mathcal{S}$ is finite, then $\mathcal{C}'$ is finite, thus recursive. We now show that $\mathcal{S}$ is indeed finite.

Observe, however, that even though there are only finitely many states, we cannot bound the length of a run by removing "loops" starting and ending in the same state (and guarantee the finiteness of $\mathcal{S}$ this way). This is because a controller may be using pure sensing actions which do not change the state to obtain information. So we need a slightly more complex approach.

Given a set of histories $\mathcal{H}$, and a history $\sigma \in \mathcal{H}$, a segment τ in σ: $\sigma = \sigma_1 \circ \tau \circ \sigma_2$ for some σ_1 and σ_2, is said to be *determinate* if for any σ' in $\mathcal{H}$, whenever σ_1 is a proper prefix of σ', then $\sigma_1 \circ \tau$ is a prefix of σ'. In other words, the underlying controller determined by $\mathcal{H}$ according to (4) with $\mathcal{S}$ replaced by $\mathcal{H}$, if any, does not need to consider the alternative outcome of the actions in τ. Notice that the empty sequence is trivially a determinate segment of any history σ with respect to any σ_1 and σ_2 above.

Given any run $\sigma \in \mathcal{S}$, we can decompose it into

$$\sigma = \sigma_1 \circ (\alpha_1, \beta_1) \circ \cdots \circ \sigma_k \circ (\alpha_k, \beta_k) \circ \sigma_{k+1} \tag{5}$$

such that

1. $\sigma_1, \ldots, \sigma_{k+1}$ are determinate segments of σ in $\mathcal{S}$.
2. $\sigma_1 \circ (\alpha_1, \beta_1), \ldots, \sigma_k \circ (\alpha_k, \beta_k)$ are not determinate segments of σ in $\mathcal{S}$.

Clearly, this decomposition is unique. Furthermore, it has the following properties:

(I) For any $\sigma' \in \mathcal{S}$, if

$$\sigma' = \sigma'_1 \circ (\alpha'_1, \beta'_1) \circ \cdots \circ \sigma'_m \circ (\alpha'_m, \beta'_m) \circ \sigma'_{m+1}$$

is a similar decomposition of σ', and

$$(\alpha_1, \beta_1) \circ \cdots \circ (\alpha_k, \beta_k) = (\alpha'_1, \beta'_1) \circ \cdots \circ (\alpha'_m, \beta'_m),$$

then $\sigma = \sigma'$. Suppose the above equation holds, then $m = k$, $\alpha_i = \alpha'_i$, and $\beta_i = \beta'_i$, $1 \leq i \leq k$. We show by induction on $0 \leq i \leq k$ that

$$\sigma_1 \circ (\alpha_1, \beta_1) \circ \cdots \circ \sigma_i \circ (\alpha_i, \beta_i) = \sigma'_1 \circ (\alpha'_1, \beta'_1) \circ \cdots \circ \sigma'_i \circ (\alpha'_i, \beta'_i).$$

The base case is trivial because both of them are empty sequence. Suppose

$$\sigma_1 \circ (\alpha_1, \beta_1) \circ \cdots \circ \sigma_i \circ (\alpha_i, \beta_i) = \sigma'_1 \circ (\alpha'_1, \beta'_1) \circ \cdots \circ \sigma'_i \circ (\alpha'_i, \beta'_i),$$

we show that

$$\sigma_1 \circ (\alpha_1, \beta_1) \circ \cdots \circ \sigma_{i+1} \circ (\alpha_{i+1}, \beta_{i+1}) = \sigma'_1 \circ (\alpha'_1, \beta'_1) \circ \cdots \circ \sigma'_{i+1} \circ (\alpha'_{i+1}, \beta'_{i+1}),$$

that is, $\sigma_{i+1} = \sigma'_{i+1}$. Because σ_{i+1} is a determinate segment, by definition, we have that $\sigma_1 \circ (\alpha_1, \beta_1) \circ \cdots \circ \sigma_{i+1}$ must be a prefix of σ'. Similarly, $\sigma'_1 \circ (\alpha'_1, \beta'_1) \circ \cdots \circ \sigma'_{i+1}$ is a prefix of σ. So either $\sigma_{i+1} = \sigma'_{i+1}$ or one of them is a proper prefix of the other. The latter is impossible because otherwise, say σ_{i+1} is a proper prefix of σ'_{i+1}, then σ'_{i+1} cannot be a determinate segment of σ, which violates our assumptions. This proves that

$$\sigma_1 \circ (\alpha_1, \beta_1) \circ \cdots \circ \sigma_k \circ (\alpha_k, \beta_k) = \sigma'_1 \circ (\alpha'_1, \beta'_1) \circ \cdots \circ \sigma'_k \circ (\alpha'_k, \beta'_k).$$

Finally, $\sigma = \sigma'$. For otherwise, one of them would be a proper prefix of the other, which is impossible because they are both runs of $\mathcal{C}$ under some environment.

(II) For any $1 \leq i < j \leq k$, if $\alpha_i = \alpha_j$, then

$$T \not\models S \leq \mathit{end}(\xi_{j-1} \circ \sigma_j, S) \supset \mathit{SameState}(\mathit{end}(\xi_{i-1} \circ \sigma_i, S), \mathit{end}(\xi_{j-1} \circ \sigma_j, S)),$$

where for any $0 \leq m \leq k$, ξ_m is the history:

$$\sigma_1 \circ (\alpha_1, \beta_1) \circ \cdots \circ \sigma_m \circ (\alpha_m, \beta_m).$$

For otherwise, suppose

$$T \models S \leq \mathit{end}(\xi_{j-1} \circ \sigma_j, S) \supset \mathit{SameState}(\mathit{end}(\xi_{i-1} \circ \sigma_i, S), \mathit{end}(\xi_{j-1} \circ \sigma_j, S)). \quad (6)$$

We claim that $\sigma_j \circ (\alpha_j, \beta_j)$ would have to be a determinate segment of σ, a contradiction with the assumptions that we made about equation (5). To show that $\sigma_j \circ (\alpha_j, \beta_j)$ is a determinate segment of σ, suppose $\sigma' \in \mathcal{S}$, and ξ_{j-1} is a proper prefix of σ'. Then $\xi_{j-1} \circ \sigma_j$ must be a prefix of σ' for σ_j is a determinate segment of σ. Because both σ' and σ are terminating runs of the controller $\mathcal{C}$, for some β, $\xi_{j-1} \circ \sigma_j \circ (\alpha_j, \beta)$ must be a prefix of σ'. Now let I be the model of T under which σ' is the terminating run of the controller $\mathcal{C}$. Since $\sigma' \in \mathcal{S}$, by Definition 9, $I \models S \leq \mathit{end}(\sigma', S)$. Thus by (6),

$$I \models \mathit{SameState}(\mathit{end}(\xi_{i-1} \circ \sigma_i, S), \mathit{end}(\xi_{j-1} \circ \sigma_j, S)).$$

Since $\alpha_i = \alpha_j$, by Lemma 6.1 $\beta = \beta_i$. Similarly, $\beta_j = \beta_i$. So $\beta = \beta_j$. Thus ξ_j is a prefix of σ'. This shows that $\sigma_j \circ (\alpha_j, \beta_j)$ is a determinate segment of σ.

Thus by Lemma 6.2 and the above property (II), for any run σ in $\mathcal{S}$, if it is decomposed as (5), then $k \leq M \times N$, where M is the number of actions, and N is the upper bound of the number of possible equivalent classes given in Lemma 6.2. By property (I), this implies that $\mathcal{S}$ must be finite.

Appendix B Proof of Theorem 2

We shall prove a more general result than Theorem 2. To that end, we introduce a relation *Rexit*(r, s, s') meaning that when started in s, the program r will exit (*abort*) in s'. It is defined in the way similar to *Rdo* in Section 4, as an abbreviation of the following second-order formula:

$$Rexit(r, s, s') \stackrel{def}{=} (\forall P)[\cdots \supset P(s, s', 0)].$$

where the ellipsis is exactly the same as in (3), i.e. the conjunction of six conditions given in Section 4. From this definition, it is easy to verify that the following are consequences of any basic action theory:

$$\neg Rexit(\underline{nil}, s, s'),$$
$$Rexit(\underline{exit}, s, s),$$
$$Rexit(\underline{branch}(a, r_1, r_2), s, s') \equiv$$
$$Poss(a, s) \wedge [SF(a, s) \supset Rexit(r_1, do(a, s), s')] \wedge [\neg SF(a, s) \supset Rexit(r_2, do(a, s), s')].$$

The purpose of introducing *Rexit* is for the following lemma:

Lemma 6.3 *Let T be an action theory, and I a model of T. For any ground situation terms S and S', $I \models Rdo(\underline{loop}(r_1, r_2), S, S')$ iff there are some ground situation terms $S_1, ..., S_n$, $n \geq 3$, such that*

1. $S = S_1$ *and* $S' = S_n$.
2. *For any* $1 \leq i \leq n-3$, $I \models Rdo(r_1, S_i, S_{i+1})$.
3. $I \models Rexit(r_1, S_{n-2}, S_{n-1})$, *and* $I \models Rdo(r_2, S_{n-1}, S_n)$.

Similarly, $I \models Rexit(\underline{loop}(r_1, r_2), S, S')$ *iff there are some* $S_1, ..., S_n$, $n \geq 3$, *such that*

1. $S = S_1$ *and* $S' = S_n$.
2. *For any* $1 \leq i \leq n-3$, $I \models Rdo(r_1, S_i, S_{i+1})$.
3. $I \models Rexit(r_1, S_{n-2}, S_{n-1})$, *and* $I \models Rexit(r_2, S_{n-1}, S_n)$.

Proof: By induction on the structure of r_1. ■

In the following, a history σ is called an *exiting run* of a system $(\mathcal{C}, \mathcal{E})$ iff it is a run of the system, and $\mathcal{C}(\sigma) = abort$. The following theorem includes Theorem 2 as a special case.

Theorem 5 *For any robot program r and any ground situation term S, there is an effective robot controller $\mathcal{C}$ such that for any interpretation I and any ground situation term S':*

1. *(a) If $I \models$ Rdo(r, S, S'), then there is a terminating run σ of $(\mathcal{C}, \mathcal{E})$ such that $S' =$ end(σ, S); (b) if $I \models$ Rexit(r, S, S'), then there is an exiting run σ of $(\mathcal{C}, \mathcal{E})$ such that $S' =$ end(σ, S), where $\mathcal{E}$ is the environment determined by I at S.*

2. *(a) If there is a terminating run σ of $(\mathcal{C}, \mathcal{E})$ such that end$(\sigma, S) = S'$, then $I \models S \leq S' \supset$ Rdo(r, S, S'); (b) If there is an exiting run σ of $(\mathcal{C}, \mathcal{E})$ such that end$(\sigma, S) = S'$, then $I \models S \leq S' \supset$ Rexit(r, S, S'), where $\mathcal{E}$ is the environment determined by I at S.*

Proof: We shall construct a recursive function $\Gamma : \mathcal{P} \times \mathcal{R} \to \mathcal{A}^+$, where $\mathcal{P}$ is the set of robot programs, such that for any robot program r, $\lambda\sigma\Gamma(r, \sigma)$ is a robot controller that satisfies the two conditions in the theorem with respect to r. We define Γ inductively on the structure of robot programs:

$$\Gamma(\underline{\mathit{nil}}, \sigma) = \begin{cases} \mathit{stop} & \text{if } \sigma = \epsilon \\ \perp & \text{otherwise} \end{cases}$$

$$\Gamma(\underline{\mathit{exit}}, \sigma) = \begin{cases} \mathit{abort} & \text{if } \sigma = \epsilon \\ \perp & \text{otherwise} \end{cases}$$

$$\Gamma(\underline{\mathit{branch}}(a, r_1, r_2), \sigma) = \begin{cases} a & \text{if } \sigma = \epsilon \\ \Gamma(r_1, \sigma') & \text{if } \sigma = (a, 1) \circ \sigma' \\ \Gamma(r_2, \sigma') & \text{if } \sigma = (a, 0) \circ \sigma' \\ \perp & \text{otherwise} \end{cases}$$

$$\Gamma(\underline{\mathit{loop}}(r_1, r_2), \sigma) = \begin{cases} \Gamma(r_2, \sigma'') & \text{if for some } n \geq 0, \sigma = \sigma_1 \circ \cdots \circ \sigma_n \circ \sigma' \circ \sigma'' \text{ such that} \\ & \Gamma(r_1, \sigma_i) = \mathit{stop} \text{ for } 1 \leq i \leq n, \text{ and } \Gamma(r_1, \sigma') = \mathit{abort}. \\ \Gamma(r_1, \sigma') & \text{otherwise, where } \sigma' \text{ is a history} \\ & \text{such that for some } n \geq 0, \sigma = \sigma_1 \circ \cdots \circ \sigma_n \circ \sigma', \\ & \Gamma(r_1, \sigma_i) = \mathit{stop} \text{ for } 1 \leq i \leq n, \text{ and there is no} \\ & \text{proper prefix } \sigma'' \text{ of } \sigma' \text{ such that } \Gamma(r_1, \sigma'') = \mathit{stop} \end{cases}$$

Clearly, for any program r, $\lambda\sigma\Gamma(r, \sigma)$ is a recursive function. We now show that $\lambda\sigma\Gamma(r, \sigma)$ satisfies the two conditions in the Theorem. We do so by induction over the structure of programs.

- r is *nil*. For any ground situation term S, and any interpretation I:

  $$I \models (\forall s)(\mathit{Rdo}(\underline{\mathit{nil}}, S, s) \equiv s = S) \wedge \neg(\exists s)\mathit{Rexit}(\underline{\mathit{nil}}, S, s).$$

  From this, and the definition of $\Gamma(\underline{\mathit{nil}}, \sigma)$, the two conditions in the theorem are trivially satisfied.

- r is *exit*. This case is analogous to the case of *nil*: For any S, and any I,

  $$I \models (\forall s)(\mathit{Rexit}(\underline{\mathit{exit}}, S, s) \equiv s = S) \wedge \neg(\exists s)\mathit{Rdo}(\underline{\mathit{exit}}, S, s).$$

- r is $\underline{branch}(a, r_1, r_2)$. Inductively, we assume that the two conditions are satisfied for $\lambda\sigma\Gamma(r_1, \sigma)$ and $\lambda\sigma\Gamma(r_2, \sigma)$. Let S be an arbitrary ground situation term, and I an arbitrary interpretation.

  Suppose $I \models Rdo(\underline{branch}(a, r_1, r_2), S, S')$. By the definition of *Rdo*, there are two cases:

  1. $I \models SF(a, S) \wedge Rdo(r_1, do(a, S), S')$. By inductive assumption, there is a terminating run τ of $\lambda\sigma\Gamma(r_1, \sigma)$ under the environment determined by I at $do(a, S)$ such that $S' = end(\tau, do(a, S))$. By our construction of $\Gamma(\underline{branch}(a, r_1, r_2), \sigma)$, $(a, 1) \circ \tau$ is a terminating run of $\lambda\sigma\Gamma(\underline{branch}(a, r_1, r_2), \sigma)$ under the environment determined by I at S.[7] Furthermore,
  $$end((a, 1) \circ \tau, S) = end(\tau, do(a, S)) = S'.$$
  2. $I \models \neg SF(a, S) \wedge Rdo(r_2, do(a, S), S')$. analogous to the previous case.

  This proves condition (1)(a) of the theorem. The proof of (1)(b) is analogous.

  To prove (2)(a), suppose τ is a terminating run of $\lambda\sigma\Gamma(\underline{branch}(a, r_1, r_2), \sigma)$ under the environment $\mathcal{E}$ determined by I at S. Again, there are two cases:

  1. $I \models SF(a, S)$. By our construction, $\tau = (a, 1) \circ \tau'$, and τ' is a terminating run of $\lambda\sigma\Gamma(r_1, \sigma)$ under the environment determined by I at $do(a, S)$. By inductive assumption,
  $$I \models do(a, S) \leq S' \supset Rdo(r_1, do(a, S), S'),$$
  where $S' = end(\tau', do(a, S)) = end(\tau, S)$. By
  $$I \models Poss(a, S) \wedge SF(a, S) \wedge Rdo(r_1, do(a, S), S') \supset Rdo(\underline{branch}(a, r_1, r_2), S, S'),$$
  we have
  $$I \models S \leq S' \supset Rdo(\underline{branch}(a, r_1, r_2), S, S').$$
  2. $I \models \neg SF(a, S)$. analogous to the previous case.

  This proves (2)(a). The proof of (2)(b) is again analogous.

- r is $\underline{loop}(r_1, r_2)$. The proof for this case is exactly like that for the branch case, but using Lemma 6.3.

■

[7] This makes use of easy lemma that if τ is a terminating run of $\mathcal{C}$ under the environment determined by I at $do(a, S)$, and $I \models SF(a, S)$, then $(a, 1) \circ \tau$ is a terminating run of $\mathcal{C}'$ under the environment determined by I at S, where $\mathcal{C}'$ is a controller such that $\mathcal{C}'(\epsilon) = a$ and $\mathcal{C}'((a, 1) \circ \sigma) = \mathcal{C}(\sigma)$.

Appendix C Proof of Theorem 3

We restate Theorem 3:

Theorem 3 *For any TM basic action theory T, any effective controller $\mathcal{C}$, and any ground situation term S, there is a robot program r such that for any model I of T and any ground situation S', we have*

1. *If $I \models Rdo(r, S, S')$, then there is a terminating run σ of the system $(\mathcal{C}, \mathcal{E})$ such that $clean(S') = end(\sigma, S)$, where $\mathcal{E}$ is the environment determined by I at S.*

2. *If there is a terminating run σ of the system $(\mathcal{C}, \mathcal{E})$ such that $S' = end(\sigma, S)$, then there is a situation S'' such that $S' = clean(S'')$, and $I \models S \leq end(\sigma, S) \supset Rdo(r, S, S'')$, where $\mathcal{E}$ is the environment determined by I at S.*

Proof: The proof involves using the fluents *Marked* and *loc* to emulate a Turing machine tape. If $Marked(n, s)$ holds, then the nth position of the tape will be `1`, and otherwise, `0`. To indicate the state of some segment of the tape, we will show a sequence of `0`s and `1`s, with the current location (*i.e.* $loc(s)$) taken to be the position of the first digit of the underlined pair of `0`s and `1`s. Thus if the initial segment of the tape in S_0 is $\underline{00}1011$, then $loc(S_0) = 1$, $\neg Marked(1, S_0)$, $\neg Marked(2, S_0)$, $Marked(3, S_0)$, etc.

We assume without loss of generality that there are only two action A and B. We can encode a history as follows:

$$\underline{\texttt{00}}\alpha_1\beta_1 \cdots \alpha_n\beta_n\texttt{11},$$

where α_i is `01` or `10` for action A or action B respectively, and where β_i is `01` or `10` for outcome 0 or 1 respectively. Since $\mathcal{C}$ is a recursive function, there is a Turing machine M that computes it. Given a history encoded as above, it will terminate with the tape looking like

$$\texttt{00}\alpha_1\beta_1 \cdots \alpha_n\beta_n\underline{\alpha},$$

where α is either `00` (for *abort* or $\perp$), `11` (for *stop*) or `01` or `10` as above for A or B. Since the five special actions have the same power as a Turing machine, there is a robot program r_M that consists of only Turing actions that senses and marks the tape in exactly this way.

We can now describe the robot program that achieves the same goal as the controller $\mathcal{C}$.

- For any fixed string z of `0`s and `1`s, let $write(z)$ be the robot program that writes z on the tape and sets *loc* to the position just right of z. It is defined inductively by:

$$write(\epsilon) = \underline{nil};$$
$$write(\texttt{0}z) = \underline{seq}(erase, \underline{seq}(right, write(z)));$$
$$write(\texttt{1}z) = \underline{seq}(mark, \underline{seq}(right, write(z))).$$

- The robot program *home* is such that whenever the tape encodes some history like:

$$\texttt{00}\alpha_1\beta_1 \cdots \alpha_n\beta_n\underline{\alpha},$$

  home will reset *loc* to 1:

$$\begin{array}{l} \underline{loop}(\underline{seq}(left,\ \underline{branch}(read\_mark, \\ \qquad\qquad \underline{seq}(left,\ \underline{nil}), \\ \qquad\qquad \underline{seq}(left,\ \underline{branch}(read\_mark,\ \underline{nil},\ \underline{exit})))), \\ \qquad \underline{nil}). \end{array}$$

- The robot program *ini* is $[write(\texttt{0011})\,;\ home]$.[8]

- The robot program r_A performs the action A, and writes either $\texttt{0111}$ or $\texttt{1011}$ on the tape depending on the sensing result returned:

$$\underline{branch}(A, [write(\texttt{1011})\,;\ home], [write(\texttt{0111})\,;\ home]).$$

  The program r_B is analogous.

- The robot program *interpret* is

$$\begin{array}{l} \underline{branch}(read\_mark, \\ \quad \underline{seq}(right,\ \underline{branch}(read\_mark, \\ \qquad \underline{seq}(right,\ \underline{exit}), \\ \qquad \underline{seq}(right,\ r_B))), \\ \quad \underline{seq}(right,\ \underline{branch}(read\_mark, \\ \qquad \underline{seq}(right,\ r_A), \\ \qquad \underline{seq}(right,\ \underline{loop}(\underline{nil},\underline{nil}))))) \end{array}$$

- Finally, the desired robot program r is $[ini\,;\ \underline{loop}([r_M\,;\ interpret],\ \underline{nil})]$ where r_M is the robot program associated with a single step of the effective controller $\mathcal{C}$.

To paraphrase, the robot program r, starting at some home position on the tape, first writes $\texttt{0011}$ encoding the empty history, and then returns to the home position. Next, within a loop, it repeatedly uses r_M to place a suitable α at the end of the history, and then interprets

[8] For any robot programs r and r', $r\,;\,r'$ means executing r followed by r'. Formally, the ";" notation is defined inductively:

$$\begin{array}{l} [\underline{nil}; r] = r; \\ [\underline{exit}; r] = \underline{exit}; \\ [\underline{branch}(a, r_1, r_2); r] = \underline{branch}(a, [r_1; r], [r_2; r]); \\ [\underline{loop}(r_1, r_2); r] = \underline{loop}(r_1, [r_2; r]). \end{array}$$

this α: if it is `00` it goes into an infinite loop; if it is `11` it exits the loop (and so terminates); if it is `10`, it performs action A, writes either `0111` or `1011` on the tape depending on the sensing result returned, and then returns to the home position; if it is `01`, it does the same as above but for action B. Note that when α is `01` or `10`, the effect of writing `0111` or `1011` on the tape depending on the result of action A or B ensures that the tape now encodes an extended history

$$\underline{\mathtt{00}}\alpha_1\beta_1\cdots\alpha_n\beta_n\alpha_{n+1}\beta_{n+1}\mathtt{11},$$

which is then ready for the next iteration.

It can be verified that from this construction, the conditions in the theorem follow. ■