Designing Teleo-Reactive Programs

Krysia Broda and Christopher J. Hogger

Department of Computing, Imperial College London
South Kensington Campus, London SW7 2AZ UK
{kb,cjh}@doc.ic.ac.uk

Abstract. A teleo-reactive program is a condition action sequence used to control a reactive robot in response to stimuli in its environment, but in such a way as to enable the robot to achieve a pre-defined goal under normal conditions. This paper presents a systematic procedure for constructing teleo-reactive programs. It extends the authors' previous work, in particular by considering the multiple-robot case and inter-robot communication.

1 Introduction

This paper presents a procedure for constructing programs for *teleo-reactive robots*. Such robots are reactive in that they act in direct response to the stimuli they receive from their environment. Additionally, however, their behaviour is predisposed towards achieving particular goals. This is because their internal programs – which relate actions to stimuli – have been constructed using a goal-sensitive procedure. They therefore occupy a middle ground between wholly reactive robots [1] whose actions have no overall motivation, and wholly deliberative robots [2] whose actions – including on-the-fly planning – flow from an explicit stored goal.

A significant advantage of a teleo-reactive robot is the relatively low resources it needs for its internal logic, which consists of little more than a fixed rule-set requiring minimal hardware for its execution. Unlike a deliberative robot, it does not need on-board computational facilities capable of executing arbitrarily complicated software. Nevertheless, provided that the rule-set has been constructed with appropriate regard to the desired goal, the robot is likely to achieve it.

The basic principles of our method for constructing teleo-reactive programs were set out in [4] but were restricted to single-robot worlds. The new contribution of this present paper is to extend the framework to deal with multiple-robot worlds, which afford opportunities for exploiting communication and cooperation to solve problems more effectively.

The material is organized as follows. The basic framework is presented in Sections 2, 3 and 4. Extensions, including hierarchical programs, exogenous actions and multi-robot environments are presented in Sections 5 to 8. Issues of scalability, construction tools and open questions are discussed in Sections 9 to 11.

2 Basic Aspects of Formulation

2.1 World Representation

Any world in which our robots operate is one capable of assuming various *objective states*. There is an assumed (first order) language in which such states can be represented unambiguously. A state of a block-world might, for example, be represented by the conjunction

$$\langle table,\ towers(size(1), 2),\ towers(size(2), 3)\rangle$$

signifying that the state comprises a table (on which towers may stand) together with two towers each of size (height) 1 and three towers each of size 2. The state's description excludes the nature and locations of any robots operating upon the world. The presentation of our framework does not actually require detailed descriptors of the world – it is sufficient for different states to be distinguished by simple atomic identifiers. We will denote by $\mathcal{O}$ the set of all objective states for a particular application.

2.2 Robot Representation

Any robot in our framework has three main features: a set $\mathcal{P}$ of truth-valued *perceptions* it may have, a set $\mathcal{A}$ of possible *actions* it may take and a *program* relating actions to perceptions. In this paper we restrict the language of objective states, perceptions and actions to be propositional. We will use the following conventions. A *perception* is denoted by a conjunction of none or more atoms or negated atoms; the empty conjunction is denoted by $\top$ (true). $\mathcal{P}$ is the set of all perceptions. A *(basic) action* is denoted by an atom. A perception is a correct observation made by the robot of some aspect of the world's state or of itself or of other robots. It is possible that an action may not be basic, in which case it is expressed as a TR-program. The physical implementation of the perceiving mechanism employs suitable hardware sensors whose details are here immaterial. Note that a perception does not, in general, capture the entire state of the world. On the contrary, our low-resource assumption entails that the robot normally perceives only a very limited amount of information about that state.

Example 1. Returning again to block-world, a robot might have the perception $H \wedge s2$ signifying that it is holding a block and also seeing a tower of size 2. The set $\mathcal{A}$ might contain the member L (`place`), signifying the action whereby the robot places a block it is holding. Suppose that, if it is currently seeing the table then it can place the block upon that, but if it is currently seeing a tower then it can place the block upon the top of that tower. Its program might contain the condition-action rule $H, s2 \longrightarrow L$[1] signifying that if the robot has the perception on the left of the arrow then it may take the action on the right. If it takes this

[1] In TR-program rules, the conjunction operator $\wedge$ is represented by ",".

action then the world undergoes a change of state, and the perception of the robot (and potentially of other robots) must be updated accordingly.

A *TR-application* is described by the four sets $\mathcal{O}$, $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{R}$, respectively, the objective states, the perceptions, the possible actions and the robots. Associated with each kind of robot $R0$ in $\mathcal{R}$ are subsets of $\mathcal{P}$ and $\mathcal{A}$, called $R0$-*admissible* perceptions and actions, which represent its own particular capabilities. Note that it is not necessary for all robots in an application to have identical capabilities, but, unless otherwise stated, it is assumed that all robots are identical, so that $\mathcal{R}$ is a singleton set $= \{R0\}$.

Definition 1. *Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$ be a TR-application. A* situation *is a pair (o, p) where $o \in \mathcal{O}$ and $p \in \mathcal{P}$ and p is a perception that some robot $R0 \in \mathcal{R}$ may have of o.*

Note that situations are limited by the capabilities of the robots and an R-admissible perception is one that is satisfiable in its intended interpretation. If r be a rule in a TR-program for a robot type $R0$, then r is called $R0$-admissible if, and only if, its condition is an $R0$-admissible perception.

Example 1 continued. If $\mathcal{P}$ contains the descriptor $\neg H$, signifying that the robot is not holding a block, then no rule can have the perception $H, \neg H$ since no robot can be both holding and not holding. Likewise, if the robot can see no more than one tower at a time then we would disallow the perception $s2, s3$.

2.3 Program Execution

The robot's internal program – which our framework aims to construct – consists of an ordered set of condition-action rules of the kind seen above. The program controls the behaviour of the robot in that the robot commits to the action of the (textually) earliest rule whose perception is satisfied in the current state. Ordering of rules enables their conditions to be simplified and also enables the expression of preferred priorities between them.

In order that the robot can remain always active, the program includes a default rule whose condition is simply the atom $\top$. This is the program's last rule and its action is often, though not always, the `wander` action W, introduced next.

2.4 The Wander Action

It is important to the framework that the action set $\mathcal{A}$ for every robot shall include a special action which we call `wander` and denote by W. It enables the robot to change its perception without altering the world state. In the block-world context, for instance, it corresponds to enabling the robot literally to wander around on the table, so bringing different items into its range of vision. It likewise enables the robot to have its perception updated (if appropriate) when some other robot acts so as to change the state.

2.5 A Simple Example

Example 2. Here the world has a table and three blocks, and there is just one robot. The table is regarded as a tower of size 0, and a single block as a tower of size 1. The robot can perceive whether or not it is holding a block and whether or not it is seeing a tower of any particular size. The actions of which it is physically capable are `pick`, `place` and `wander`, denoted by K, L and W. A `pick` action consists of taking the top block from a tower of size > 0 and holding it. A `place` action consists of releasing a held block by placing it upon the top of a tower of size ≥ 0. A program for this robot might be

$$\neg H, s1 \longrightarrow K, \quad H, s1 \longrightarrow L, \quad H, s2 \longrightarrow L, \quad \top \longrightarrow W$$

In general, the result of any robot's actions may vary according to the initial state of the world presented to it. However, in this example the result is inevitably a 3-tower (that is, a tower of size 3) whatever the initial state, provided that W is implemented in a fair manner – that is, allows all locations in the world to be visited in the long run. Once this tower has been built the robot can only wander indefinitely unless terminated by some extraneous mechanism.

The state having a 3-tower is, in fact, the intended goal for this program. This goal is not explicit in the program, nor is it made known to the robot by any other means. Instead, the program has been constructed by a procedure that takes account of the intended goal (or goals). Note that the program above does not allow the robot to deconstruct a tower of size > 1 even though the robot is physically capable of doing so.

Even for a simple world and goal, a suitable program can be very difficult to compose using intuition alone. Consider, for instance, a modest extension to this example whereby the world has four blocks and the goal remains a 3-tower. The program shown above can still achieve the goal from most initial states. However, if the initial state has all four blocks arranged as a 4-tower or as two 2-towers then the goal is unreachable. It is not immediately obvious which alternative program would cope with those possibilities whilst remaining effective for the other initial states.

Our approach to the systematic construction of a robot's program employs a procedure which exposes for analysis the full range of possibilities determined by the assumed world together with the possible perceptions and actions of the robot. Our procedure employs for this purpose a structure which we refer to as an OP-graph. Analysis of this graph enables us to extract programs appropriate to particular goals.

3 OP-Graphs

An OP-graph OPG is a structure showing the situations $R0$ may be in and the possible actions it may take. Each directed arc in OPG signifies, and is labelled by, some such action. When $R0$ is in a situation (o, p) its possible actions depend only upon p and form a set denoted by $A(p)$.

Definition 2. *Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \{R0\}\rangle$ be a TR-application. OPG is a directed graph whose nodes are all the acceptable situations admitted by the given application and the characteristics of R0. The arcs emanating from (o, p) are precisely those corresponding to $A(p)$ and each one is directed to a situation that R0 could be in if that action were taken.*

Thus OPG can be viewed as an elaboration of a conventional state-transition graph, dealing with situations rather than with objective states alone.

Insofar as all possible actions are shown in it, this sort of graph is called a complete OP-graph. Rather than showing what will happen, it shows what could happen. A key feature of our framework is the process of pruning selected arcs from such a graph to leave a reduced OP-graph OPR which commits $R0$ to take, in any situation, just one of the possible actions. OPR then shows what will actually happen.

From any situation (o, p) in OPG, each action a in $A(p)$ yields, in principle, multiple arcs directed to all situations (o', p') for which o' is the unique objective state resulting from $R0$ taking action a and p' is a perception which $R0$ may have of o'. However, for the sake of economy we restrict the graph so that any such p' has to be the most 'natural' perception which $R0$ would have of o' after taking action a. For example, if $R0$ is seeing a 1-tower and places a held block upon it, the most natural perception for $R0$ to have of the resulting objective state is to be seeing the 2-tower it has just built, rather than to be looking elsewhere. Using this more economical representation of OPG is equivalent to avoiding composite actions which would combine acting upon the world with wandering.[2]

The possible objective states for Example 2, named 1, 2, ... etc., are shown in Figure 1, in which H denotes a block being held. Figure 1 shows, for each perception p, the set $O(p)$ of associated objective states and the set $A(p)$ of associated actions. In the perceptions, which are named $a, b, \ldots$ etc., H denotes holding, $\neg H$ denotes not holding and sn denotes seeing a tower of size n.

	$\mathcal{O}$
	Objective States
1	s1, s1, s1
2	s1, s2
3	s3
4	s1, s1, H
5	s2, H

	p	O(p)	A(p)
a	H, s0	4, 5	L, W
b	H, s1	4	L, W
c	H, s2	5	L, W
d	¬H, s0	1, 2, 3	W
e	¬H, s1	1, 2	K, W
f	¬H, s2	2	K, W
g	¬H, s3	3	K, W

Fig. 1. States, Situations and Actions (Example 2)

[2] However, it may require some perceptions to be explicitly formalised, that would otherwise not be necessary. For example, one might have to include the perception "looking at a tower of size 0", as the state immediately after picking up the single block in a tower of size 1.

Figure 2 shows the resulting graph OPG. For the sake of compactness, a situation such as $(2, f)$ is shown there simply as $2f$.

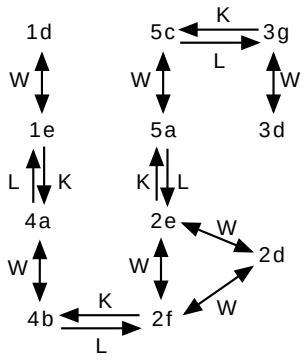

Fig. 2. Complete graph OPG (Example 2)

This graph implements the economy mentioned earlier. For example, the `pick` (K) action applied at situation $(1, e)$ changes the objective state to 4, and the natural perception for the robot then to acquire is a – that is, to be looking at the table. It is not necessary to show the possibility that K could instead take $(1, e)$ directly to $(4, b)$ because we assume that in picking the block from the table the robot would not simultaneously shift its perception to looking instead at another block. Notice that, if the `pick` action had shown the robot moving directly between $(1, e)$ and $(4, a)$, then the perception a would be redundant. It is not difficult to transform the OPG given here to one in which the perception a is eliminated.

These assumptions, however, follow from more basic ones concerning the durations of actions and the possibility of exogenous impacts upon the world by other agents (such as other robots or environmental factors). Later, when we consider multiple-robot scenarios and actions which themselves are composed of TR-programs, these assumptions will be reviewed.

The graph discloses how any situation can (or cannot) be reached from another, in particular whether a given goal situation can be reached from a given initial situation. It may also reveal subgraphs from which a given goal could never be reached. Such considerations drive the next stage of the process, namely the pruning of OPG to leave a reduced graph OPR in which each situation offers exactly one action. The pruning operation amounts to the application of a plan function. Different plan functions generally yield different reduced graphs.

4 Plan Functions

We illustrate the use of a plan function in Example 3, which extends Example 2. We have the same robot as before but in a world having four blocks, and the goal

now is to build a 4-tower. The objective states and the associated perceptions and actions are shown in Figure 3 and the complete graph is shown in Figure 4.

	$\mathcal{O}$
1	s2, s2
2	s1, s1, s1, s1
3	s1, s1, s2
4	s1, s3
5	s4
6	s1, s1, s1, H
7	s1, s2, H
8	s3, H

	p	O(p)	A(p)
a	H, s1	6, 7	L, W
b	H, s2	7	L, W
c	H, s3	8	L, W
d	¬H, s1	2, 3, 4	K, W
e	¬H, s2	1, 3	K, W
f	¬H, s3	4	K, W
g	¬H, s4	5	K, W
h	H, s0	6, 7, 8	L, W
i	¬H, s0	1, 2, 3, 4, 5	W

Fig. 3. States, Situations and Actions (Example 3)

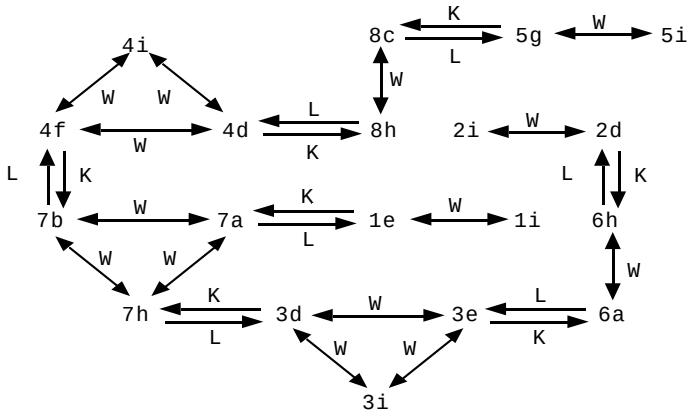

Fig. 4. Complete graph OPG (Example 3)

A plan function (also called a *policy*) is any total function F from perceptions to actions. Some such function must be chosen in order to yield a program which makes the robot's action-selection deterministic. One possible function is indicated by the reduced graph in Figure 5.

The corresponding program contains a rule $p \longrightarrow a$ in each case that $F(p) = a$ and a is not W, together with the default rule to cover all cases where $F(p) = W$:

$$\neg H, s1 \longrightarrow K, \quad H, s1 \longrightarrow L, \quad H, s2 \longrightarrow L, \quad H, s3 \longrightarrow L, \quad \top \longrightarrow W$$

Since in this example all perceptions (other than $\top$) are pairwise jointly impossible, any rule for which the action is W may correctly be relegated to the default rule. More generally, this may not be the case if the policy has been prioritised and/or optimised by taking into account the orderedness of TR-programs.

The reduced graph shows the behaviour produced by this program. Its key feature is the node set $\{(1,e),\ (1,i),\ (7,a)\}$ which forms a *trough*, that is, a subgraph containing no node from which the goal is reachable. The trough is circled for emphasis in Figure 5. From all initial situations outside the trough, the robot will achieve the goal unless it subsequently enters the trough. From $(7,h)$ it may wander into the trough, but if it wanders instead to $(7,b)$ then the goal will be achieved.

The essential problem in this example is in deciding upon the best actions for perceptions a and e. Since each one has two possible actions there are four possibilities to consider. It turns out that none of them can avoid a trough somewhere. For instance, if we modify the original policy by choosing K for perception e and W for perception a, we obtain a different OPR and this program

$$\neg H, s1 \longrightarrow K, \quad \neg H, s2 \longrightarrow K, \quad H, s2 \longrightarrow L, \quad H, s3 \longrightarrow L, \quad \top \longrightarrow W$$

In this case the trough is formed by $\{(2,d),(2,i),(3,e),(6,a),(6,h)\}$. Each of perceptions a and e has the property of being associated with several objective states having different best actions. The limited perceptions of the robot render it unable to know which objective state the world is in and hence which of the possible best actions to take.

Fundamentally, given the particular block-world and goal stipulated, the robot in this example is not perceptive enough to cope with all situations. It needs to know more in order to achieve more. In [4] we showed that one way of elegantly improving the robot is to equip it with a single-register memory capable of recording whether it has ever seen a tower of size at least 2, and to treat the reading of the register's state as another perception. For this modified robot one can find a much better (though still imperfect) policy.

A procedure for programming these robots must therefore include a method of choosing a plan function. This method should determine $F(p)$ as the overall best action in $A(p)$ taking account of all objective states associated with p. Ideally, it should at least have the property that for any non-goal state there will be some perception p for which $F(p)$ is not W, for otherwise there would be non-goal situations in which the robot was unable to change the world in order to make progress.

The notion of a best action has to be defined primarily in terms of topological features of the OP-graph, such as the reachability and proximity of goal states and the propensity of paths to enter loops or troughs. However, it may also be the case that certain actions are deemed to have a higher cost (in some defined sense) than others, allowing richer kinds of merit ordering.

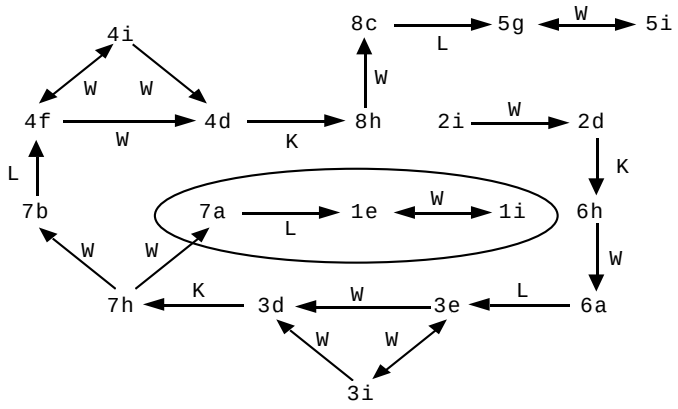

Fig. 5. Reduced graph OPR (Example 3)

4.1 Computing the Value of a Policy

In [4] it was suggested that the value of a policy (plan function) might be based upon the lengths of paths from each node in the OPR to the other nodes. More precisely, to any path from a node s to another node s' one can assign a utility which combines a reward for the robot being at s' with the cost of reaching it from s via that path. If several paths emanate from s then their separate utilities can be combined, taking into account their various probabilities, to form an expected utility which effectively measures the benefit of the robot being at node s. The main practical difficulty with this approach is that it does not afford any simple algorithm for computing the expected utilities, since their inter-relationships are non-linear.

A different way of evaluating a policy is the so-called method of discounted rewards [3], defined as follows.

Definition 3. *Let F be a policy for a TR-application $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \{R0\}\rangle$ and let $s = (o, p)$ be a situation in the OPR determined by F. The* discounted reward *$V(s, F)$, effectively measuring the benefit of $R0$ being at s, is given by the expectation*

$$V(s, F) = E[r(s, F) + \gamma(F(p)) \times V(s', F)]$$

In the above, $r(s, F)$ is the immediate reward for taking the action $F(p)$ at s. The factor $\gamma(F(p))$ discounts the benefit of reaching any s' via that action. Normally, we choose $0 < \gamma(F(p)) < 1$ to reflect the cost to the robot – in time or other resources – of performing successive actions.

Unless one has good reason to distinguish the rewards for actions from different situations, the fine tuning provided for in the above definition can be dispensed with. It then suffices to use a fixed discount factor $0 < \gamma < 1$ and two fixed values r, R such that

$r(s,F) = R$ if the action $F(p)$ taken at s leads immediately to a goal
$r(s,F) = r$ otherwise

in the simpler definition

$$V(s,F) = E[r(s,F) + \gamma \times V(s',F)]$$

The immediate reward for reaching a goal should always outweigh that for reaching a non-goal. We can arrange this by making R positive and large and making r negative. The overall value of F is the average value of $V(s,F)$ taken over all nodes s in OPR. This is equivalent to adding a virtual initial node i to OPR, connecting it by equiprobable arcs to all other nodes and computing $V(i,F)$ as the overall value of F.

Example 4. We consider an OPR having 4 situations named 1, 2, 3 and G. G is a goal and the others are not. The policy transitions – that is, the arcs in OPR – are (1, 2), (1, 3), (2, 1) and (2, G), all having probability 0.5. The discounted rewards (in which the identity of the policy F is left implicit) are:

$V(1) = 0.5(r + \gamma \times V(2)) + 0.5(r + \gamma \times V(3))$
$V(2) = 0.5(r + \gamma \times V(1)) + 0.5(R + \gamma \times V(G))$
$V(3) = 0$
$V(G) = 0$

and the policy's overall value is their average, which is $(3r + R)/(4(2 - \gamma))$. More generally, the overall value is always some linear combination of the immediate reward parameters, here r and R.

The key practical advantage of the method is that, for an OPR having n situations, it yields n linear equations relating their discounted rewards, which can be computed easily using standard algorithms. An alternative to the exhaustive brute-force computation of policies is offered in [5], which describes a learning-based approach. Returning to Example 3, the $A(p)$ sets determine that there are 256 possible policies. There are 19 situations, and varying the policy merely varies the arcs connecting them. Identifying the best policy requires solving 256 sets of linear equations in 19 unknowns. A modern desktop computer can achieve this in less than a minute. A test run was made for the case r=-1, R=100, γ=0.9 and identified two optimal policies

$$\{(a,W),(b,L),(c,L),(d,K),(e,K),(f,W),(g,W),(h,W),(i,W)\}$$

$$\{(a,W),(b,L),(c,L),(d,K),(e,K),(f,W),(g,K),(h,W),(i,W)\}$$

each having an overall value of 241.22. The policy corresponding to the OPR in Figure 5 for this example, namely

$$\{(a,L),(b,L),(c,L),(d,K),(e,W),(f,W),(g,W),(h,W),(i,W)\}$$

turns out to be the third-best one, having overall value 215.19. Its OPR has a trough, containing 3 situations, which can be entered from 8 exterior ones. By

contrast, for each of the two optimal policies the OPR has a trough, containing 5 situations, which can be entered from just 1 exterior one. The policy values therefore correctly reflect the propensity, on average, of the robot failing to reach the goal by entering a trough.

The ranking of policies is largely insensitive to the choice of values for r, R and γ provided that R strongly dominates r in magnitude. The smaller γ is, the less important is the dominance of R over r. Thus the method does not require domain-oriented intuitions about these parameters beyond giving prominence to goal situations.

4.2 Summary of Procedure for Single Robots

The procedure for programming a robot $R0$ intended to operate singly in the world can be summarized as follows:

Step 1 construct OPG from the characteristics of $R0$ and the world;
Step 2 identify the intended goal situation(s);
Step 3 generate possible plan functions and choose the best one F;
Step 4 assemble $R0$'s program from F and, if possible, simplify its rules.

5 Hierarchical Actions

So far, it has been assumed that all actions are atomic. In practice it is likely that actions could be more complex, themselves requiring a TR-program for their execution. Recall that a TR-program executes by selecting the first rule with a true condition and carrying out the prescribed action. We make two assumptions about this execution strategy. First, an action a may have a physical pre-condition such that it can be initiated only if this pre-condition is true. The physical pre-condition would normally be implied by the condition of the rule specifying that a be carried out, but the contrary case is possible too. For example, if at some point the action was "switch on motor" and for some reason the TR-program selected the same action again when the motor was already running (e.g. due to an error) the motor would not be switched on a second time and the action initiation command would be ignored. The physical pre-condition in this case is that the motor is not already switched on. Second, we assume an action a taken according to a rule $p \longrightarrow a$ does not normally cause its explicit enabling condition p to be violated until the action finishes.

When complex actions are allowed the execution sequence of a TR-program is extended. First we give an example to illustrate the procedure.

Example 5. Let $TR1$, $TR2$ and $TR3$ be the schematic programs below, where $TR1$ is the *main* program, and $TR2$ and $TR3$ are programs for actions $a2$ and $a3$ respectively. Other actions are basic actions.

TR1	$\{c1 \longrightarrow d1,\ \ c2 \longrightarrow a2,\ \ \top \longrightarrow d2\}$
TR2	$\{c3 \longrightarrow d3,\ \ c4 \longrightarrow a3,\ \ c5 \longrightarrow a3,\ \ \top \longrightarrow d4\}$
TR3	$\{c6 \longrightarrow d5,\ \ \top \longrightarrow d6\}$

Initially, suppose that $TR1$ is executed and that a state arises such that $c2$ is true. Then action $a2$ (i.e. program $TR2$) is initiated. In accordance with the principle that while $c2$ is true action $a2$ is carried out, $TR2$ should continue to execute while $c2$ is true. To ensure this the implicit enabling condition $c2$ is conjoined with each explicit enabling condition of $TR2$. The physical pre-condition is that $TR2$ is not already operating. But also in accordance with the principle that an action initiation is ignored if the physical pre-condition is false, a second instance of program $TR2$ is not initiated when action $a2$ is again required. Now both programs $TR1$ and $TR2$ are operating; if $c2$ should become false (say as a consequence of $TR2$) then some other action in $TR1$ would be selected and $TR2$ should be stopped. As the enabling condition of $TR2$ is false, that program is not carried out and indeed can be terminated. The second of our assumptions ensures that $c2$ must have become false by some other activity and so $TR2$ should no more be carried out. In addition, it may be that $TR2$ causes action $a3$ (program $TR3$) to be started. Its implicit enabling condition is $(c2 \wedge (c4 \vee c5))$ and its physical pre-condition is that $TR3$ is not operating. If either $c2$ or both of $c4$ and $c5$ should become false then the program $TR3$ should terminate. The effect of the termination of a program is analogous to its being interrupted by a higher priority program.

The execution procedure for a hierarchy of TR-programs can now be given. A program (or action) may be initiated if its physical pre-condition is true. Otherwise it is ignored. The enabling conditions of rules in a complex action are determined from the conditions of the sequences of rules that can result in their being initiated. This is defined formally in Definition 4, in which it is assumed that the hierarchy of program calls does not contain any loops. That is, no TR-program T causes, through its actions, another program to be called that calls T.

Definition 4. *Let T be a TR-program that is initiated by the action a. The enabling condition E of a rule $p \longrightarrow b$ in T is $E = p \wedge (e_1 \vee \ldots \vee e_n)$, where each e_i is the enabling condition of a rule $c \longrightarrow a$ in any program A.*

Although actions may be described by TR-programs, this paper does not yet consider how hierarchies of programs might be engineered. For instance, we assume that an action is basic when constructing the OPG, even though it may be known that it is not. It is assumed that the program for a non-basic action is already known – it is not to be synthesised from an OPG containing only basic actions. Hierarchical construction is an issue for further work.

6 Exogenous Actions

The world may be acted upon by agents other than the robot $R0$ under consideration. Such actions are called exogenous. They are not necessarily benign and are not necessarily predictable. In a block-world, for instance, we may imagine that while $R0$ is following its own agenda, some other agent is randomly knocking the top blocks off towers and leaving them scattered around singly upon the table. This may or not be helpful to $R0$, depending upon the circumstances.

Exogenous activity can be represented in the graph OPG by introducing a special action named `x`. An arc labelled `x` may be drawn from any situation $s1$ to any other $s2$ when we wish to entertain the possibility that another agent effects the transition from $s1$ to $s2$. `x` is included in $R0$'s action set. Then, if $R0$ selects `x` in situation $s1$ the effect is to make $R0$ wait for the other agent to achieve $s2$. Whether $R0$ does select `x` in situation $s1$ depends, of course, upon its program and hence upon the underlying plan function.

If the exogenous agent behaves unpredictably then, in general, it is hard to obtain a significant benefit from this new provision. By contrast, if this other agent is predictable – in particular, is itself a robot with a known plan function – then it can be exploited to advantage in circumstances where $R0$ acting alone would be inefficient in achieving the goal or be unable to reach the goal at all. Our consideration of exogenous actions from now on will therefore concentrate upon multiple-robot scenarios. Actions that arise from serendipitous actions of the environment are not considered when finding or evaluating policies.

7 Multiple Robots

In any TR-application there can be one or more robot types with one or more robots of each type. Robots of the same type are called *clones*. The description of OPG has so far assumed just one robot. As explained in Section 6, the presence of other robots can be benign, or not. The changes required to OPG to accommodate several robots are of two kinds, but in any event an OPG is always constructed from the point of view of one robot, say $R0$.

- The additional robots may affect the set of objective states $\mathcal{O}$, and possibly the perceptions $\mathcal{P}$ of $R0$; however, since these are dependent only upon the capabilities of a single robot, it is less likely they will be affected, unless, for example, the robots are themselves upgraded to be able to perceive other robots.
- From each situation there may be additional arcs for exogenous actions by other robots that will affect the situation of $R0$. These additional arcs are labelled by `x` and are interpreted from $R0$'s point of view as a `wait` action.

The change in $\mathcal{O}$ could be large. However, unless $R0$ needs to identify the other robots by name, states can be described using anonymous references to those robots. So in block world, for example, instead of requiring states that describe $R0$ and 3 other robots holding a block, it may be sufficient to record that $R0$ is holding a block and that *some* other robot is also doing so. Which one, or how many, may not be important.

When constructing an OPR from some OPG when the `x`-arcs are due to robot clones, care must be taken to obtain a *consistent* policy. To see what this means, suppose the chosen action for $R0$ in situation $s_1 = (o_1, p_1)$ is `x`, using the rule $p_1 \longrightarrow$ `x`, and leading to situation $s_2 = (o_2, p_2)$. Then the clone that causes the transition represented by `x` could not be perceiving p_1 else it too would have to obey the rule $p_1 \longrightarrow x$. In that case neither robot would take any action as

each would be waiting for the other. Thus an x-arc is allowed from a perception p (for $R0$) only if it does not require some other robot also to be perceiving p. This requirement is necessary only if the other robot is a clone of $R0$.

Example 6. We previously considered in Example 2 the behaviour of a single robot in a block-world comprised of a table and three blocks. Here we elaborate this case by allowing multiple clones. The objective states, perceptions and actions are shown in Figure 6. As before, the objective state descriptor H denotes

	$\mathcal{O}$
1	s1, s1,s1
2	s1, s2
3	s3
4	s1, s1, H
5	s2, H
6	s1, s1, H*
7	s2, H*
8	s1, H, H*

	p	O(p)	A(p)
a	H, s0	4, 5, 8	L, W, x
b	H, s1	4, 8	L, W, x
c	H, s2	5	L, W, x
d	¬H, s0	1, 2, 3, 6, 7	W, x
e	¬H, s1	1, 2, 6	K, W, x
f	¬H, s2	2, 7	K, W, x
g	¬H, s3	3	K, W, x

Fig. 6. States, Situations and Actions (Example 6)

that the robot under consideration is holding a block. The new descriptor H^* denotes that some other robot is holding one. Introducing H^* increases the number of states to 8 whereas, in the single-robot case, there were just 5. Some of the $O(p)$ sets become correspondingly enlarged, and each $A(p)$ set now contains the exogenous action x. The OPG has 17 situations and there are 1458 possible plan functions, of which just 88 are clone-consistent in the sense outlined earlier.

Choosing again the parameter values r=-1, R=100, γ=0.9, the best policy among those catering for an exogenous action x is

$$\{(a, W), (b, L), (c, L), (d, \mathtt{x}), (e, K), (f, K), (g, K)\}$$

whose overall value is 102.88. Figure 7 shows a very small fragment of the OPR corresponding to this policy. Consider any clone $R0$ governed by this policy. In

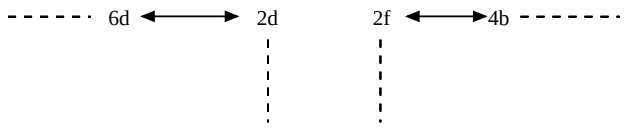

Fig. 7. Fragment of an OPR (Example 6)

the situation (2, d) there is a 1-tower and a 2-tower, and $R0$ is looking at the

table and not holding. For this perception the policy requires $R0$ to wait for another clone, say $R1$, to effect the transition from (2, d) to (6, d), that is, to pick (K) a block from the 2-tower. ¿From $R1$'s point of view this is a transition from (2, f) to (4, b). The only action that can alter the objective state from 2 to 4 is K, so taking that action requires $R1$'s policy to contain (f, K). Since clones share the same policy, it follows that $R0$'s policy must also contain (f, K) – which it does. This precisely illustrates what we mean by the policy being consistent for the clones. Put succinctly it requires that, under a shared policy F, if F enables one clone to wait for a transition to occur then F must enable another clone to effect that transition.

For this example the best policies happen to be ones that do not cater for exogenous actions, namely

$$\{(a, W), (b, L), (c, L), (d, W), (e, K), (f, W), (g, W)\}$$

$$\{(a, W), (b, L), (c, L), (d, W), (e, K), (f, W), (g, K)\}$$

each of which has overall value 239.74. These are also the best policies for the single-robot case of Example 2. This shows that using several clones to pursue a goal is not necessarily always better than using just one.

Instead of cloning, two robots $R0$ and $R1$ can be assigned distinct policies to pursue the common goal. This requires the property of *cooperativeness* whereby if either policy enables its robot to wait for a transition to occur then the other policy must enable its robot to effect it. The consistency requirement for clones is just a special case of this, namely self-cooperativeness. We do not possess a way of computing, or even estimating, the overall value of a set of several policies. Their individual values do not provide a reliable guide to the value of co-implementing them among several robots. The policy value for a particular robot reflects only how it is expected to fare in the context of possible actions by other robots but, unless it can perceive what those others are actually doing, its policy cannot take any account of their joint influence, for instance whether they are cooperating or merely getting in each other's way. In cases where this kind of joint planning is important it is necessary that robots should be able to communicate, which is the subject of the next Section.

8 Communicating Robots

The extension of the framework to allow several robots has so far excluded any communication between them. If non-communicating robots appear to co-operate in achieving a task then this is merely a fortituitous manifestation of emergent behaviour. In [4] we called this "as-if" co-operation.

In this Section we show how deliberate (planned) co-operation can be obtained by enabling robots to communicate. We avoid the need to devise special languages and protocols for this by restricting the communicable elements to be perceptions of the kind already employed. If some robot $R1$ has an atomic perception p then we can allow another robot $R0$ to have the atomic perception

p^* whose meaning is that $R1$ is perceiving, and communicating to $R0$, that p is true. We assume that the content of p is instantly transmissible from $R1$ to $R0$ by some suitable broadcasting mechanism. With this provision in place, policy formation for $R0$ can then take account of what it receives from $R1$ in addition to its own direct perceptions of the world.

Example 7. Consider the 4-blocks problem of Example 3 but with a physical restriction applied that a robot may pick a block only from a 1-tower, that is, not from any higher tower. Whether there be one robot or several, the possibility then arises of creating a state having a pair of 2-towers. In Example 3 this was unproblematic because either tower could be deconstructed to supply blocks with which to extend the other in the quest to build a 4-tower. With the new restriction, however, such a state is guaranteed to produce a trough.

We first consider the case where the task is given to one or more clones having no communication. The objective states, perceptions and actions are shown in Figure 8. As before, the state descriptor H denotes that the robot

	$\mathcal{O}$
1	s2, s2
2	s1, s1, s1, s1
3	s1, s1, s2
4	s1, s3
5	s4
6	s1, s1, s1, H
7	s1, s2, H
8	s3, H
9	s1, s1, s1, H*
10	s1, s2, H*
11	s3, H*
12	s1, s1, H, H*
13	s2, H, H*

	p	O(p)	A(p)
a	H, s1	6, 7, 12	L, W, x
b	H, s2	7, 13	L, W, x
c	H, s3	8	L, W, x
d	¬H, s1	2, 3, 4, 9, 10	K, W, x
e	¬H, s2	1, 3, 10	W, x
f	¬H, s3	4, 11	W, x
g	¬H, s4	5	W, x
h	H, s0	6, 7, 8, 12, 13	L, W, x
i	¬H, s0	1, 2, 3, 4, 5, 9, 10, 11	W, x

Fig. 8. States, Situations and Actions (Example 7)

under consideration is holding a block whilst H^* denotes that some other robot is holding one. There are 30 situations and 3888 possible plan functions, of which just 64 are clone-consistent. Choosing again the parameter values r=-1, R=100, γ=0.9, the optimal clone-consistent policy is

$$\{(a, L), (b, L), (c, L), (d, K), (e, W), (f, \mathrm{x}), (g, W), (h, W), (i, W)\}$$

whose value is 282.86. Figure 9 outlines the OPR corresponding to this policy. ¿From all nodes in the ringed subgraph at the top there is a path to (7, h). In this situation there is a 1-tower and a 2-tower, and a robot is holding a block and looking at the table. Its action is to wander, either to (7, b) or to (7, a). (7,

b) is in the ringed subgraph marked 'succeeds', all of whose nodes lead to the goal. (7, a) is in the ringed subgraph on the right, which is a trough: in this case the fatal situation (1, e) having a pair of 2-towers is created. Assuming that the emergent arcs from any node are equiprobable, there is a probability of 1/2 at (7, h) of the robot entering this trough.

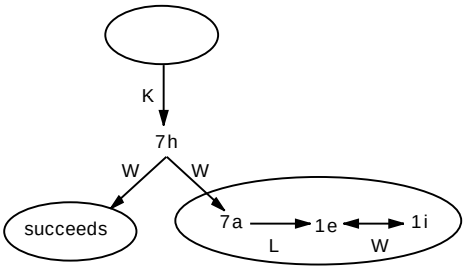

Fig. 9. Outline of optimal OPR (Example 7)

We can reduce the probability of entering a trough in this example by enabling our robot to receive communication from another clone as to whether or not the latter is currently seeing a 2-tower. If it knows that a 2-tower is being seen then in state 7 it can desist from placing its block on the 1-tower and instead wait for another clone to alter the state exogenously and so remove the jeopardy. The problem formulation is therefore adjusted slightly by replacing perception a by two new alternates

$$a1 : H, s1, s2^* \qquad \text{and} \qquad a2 : H, s1, \neg s2^*$$

Here $s2^*$ denotes that another robot is communicating that it is perceiving a 2-tower, whilst $\neg s2^*$ denotes that it is not communicating this. This results in 31 situations and 11664 possible plan functions, of which just 182 are clone-consistent. With r=-1, R=100, γ=0.9, the optimal clone-consistent policy is

$$\{(a1, \mathtt{x}), (a2, L), (b, L), (c, L), (d, K), (e, W), (f, \mathtt{x}), (g, W), (h, W), (i, W)\}$$

whose value is 295.33. Figure 10 outlines the OPR corresponding to this policy. The 'succeeds' subgraph now contains two situations to which the robot may wander from (7, h). These are (7, b), as before, and now also (7, a1), the situation in which it is holding a block, seeing a 1-tower and knowing that another robot is seeing a 2-tower. The rule ($a1$, x) in the above policy ensures that the robot will then, as we intended, wait for another robot to remove the jeopardy. On the right of the figure the trough remains because the robot may wander instead to (7, a2) and then, not knowing whether a 2-tower exists, create the fatal situation (1, e). We do not, of course, have to choose a policy which assigns action L to perception $a2$, but any other action for $a2$ yields a policy that is significantly sub-optimal. In the OPR for the optimal policy the probability of wandering

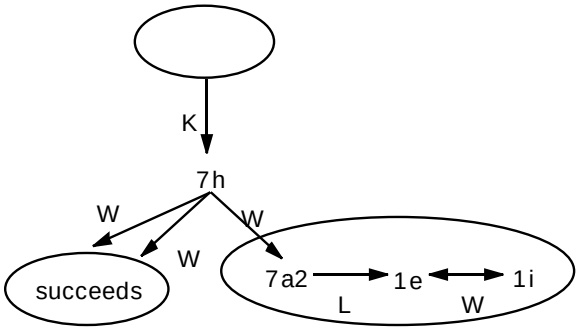

Fig. 10. Outline of optimal OPR (Example 7)

from (7, h) into the trough is now just 1/3, a modest but definite improvement over the case where there was no communication.

The use of communication in the manner illustrated is semantically equivalent to the alternative of simply increasing the ability of a robot to perceive by its own means more of the objective state. Viewed in this way, we could have cast the new perceptions as

$$a1 : H, s1, s2 \qquad \text{and} \qquad a2 : H, s1, \neg s2$$

on the assumption that a robot was physically equipped to see, simultaneously, both a 1-tower and a 2-tower, however far apart they might be. In engineering terms, however, it is more practical to broadcast perceptions through one uniform technology than to equip robots with a diverse range of perceptual sensors. So although the two views are equivalent, we prefer the one we interpret as communication.

9 Scalability

A key concern for a framework of this kind is that it shall adequately scale up to the treatment of realistic scenarios. For a scenario having perceptions $p_1, \ldots, p_n$ the number of distinct policies is the product $|A(p_1)| \times \ldots \times |A(p_n)|$. We have already emphasized that the framework is intended for simple robots in which each action set $A(p_i)$ is small. Given this, the number of policies depends chiefly upon the number n of distinct perceptions and, correspondingly, upon the diversity of the objective world as modeled for any particular problem.

The key to scalability therefore lies, in our view, in the use of appropriate representation. In the case of blockworld, the tower-building examples examined in this paper fall into a general class in which the basic perceptions (other than holding and not holding) need only be seeing a tower of the desired height, or one of lesser height or one of greater height. It is therefore not necessary to represent the seeing and acting-upon of all specific heights (1, 2, 3, etc.), unless the robot's physical capabilities happen to be predicated upon specific heights.

Although we must not claim here that scalability is an issue of little concern, we do believe that, with appropriate world-modeling, the framework enables the feasible design of simple robots operating in moderately complicated worlds. But, only further case studies could properly test that belief.

10 Framework Tools

As the examples in this paper demonstrate, even quite simple problems can determine large and complex situation graphs whose manual characterization would be highly tedious and error-prone. To reduce the scope for erroneous formulation we employ, for blockworld (and, potentially, other scenarios) an *OPG-generator* program which, for any designated set of blocks, robots, actions and goal, autonomously constructs the OPG graph and further checks it against robust integrity constraints.

We further employ a *policy-generator* program for building policies capable of executing, in a multi-robot case, with or without the constraint of clone-consistency. Testing for clone-consistency is an intricate and delicate task, owing to the need to transpose one robot's view of any world transition into the view of that same transition from the perspective of some other robot. The policy-generator employs, when testing for clone-consistency, tables of assuredly-equivalent transition views constructed in advance by the OPG-generator.

A third program, the *policy-evaluator*, computes the overall value of each policy. With potentially thousands of policies to consider, this process needs to be as efficient as possible. A key optimization here is to reuse reward values for individual nodes. As each policy (after the first) is evaluated, the program compares this policy with the preceding one to identify those nodes whose rewards remain invariant under this policy change. Under any policy each node has an associated subgraph (in the corresponding OPR) containing the nodes reachable from it. If this subgraph is topologically invariant upon switching to a new policy then the latter inherits the earlier rewards for that node and for all others in the subgraph. The policy evaluator performs appropriate graph traversals to test this property efficiently. In practice this tactic reduces enormously the time spent on computing node rewards.

We do not yet possess a principle for ordering policies so as to optimize the time required to evaluate them. However, there is an intuitive argument that ordering policies so as to minimize their pairwise differences will favour the reusability of node rewards as just indicated. Therefore, the policy-generator employs the analogue of a "gray-tones" algorithm which ensures (a) that each policy differs from its predecessor by precisely one (p, a) pair and (b) that all policies will be generated. Using these principles, the policy-generator assembles for each policy the corresponding equations relating the node rewards, solves them using a Gaussian procedure and then computes the overall policy value.

All these programs are currently written in LPA-Prolog[3], enabling the reshaping of examples to be undertaken through transparent and easily-adapted

[3] Logic Programming Associates Ltd. London, UK, (http://www.lpa.co.uk)

representations. Despite the relative tardiness of an interpretive formalism such as Prolog, it takes only a minute or so to evaluate 10,000 policies on an Apple G4 platform.

11 Discussion

TR-robots, their applications and many extended treatments of them have been investigated extensively by Nilsson [6, 7]. His formulation of the basic nature of any such robot shares some similarities with our own in focusing upon perception-action rules and their impact upon the objective world. However, it differs significantly from ours in imposing upon the rules a stricter structure than we do. In particular, he imposes the so-called *regression property*. This requires that the effect of any rule $p \rightarrow a$ shall be to cause the condition of some earlier rule to become satisfied and that p shall be weaker (i.e. implied by) the condition of any other rule having that same effect. Additionally, these arrangements must lead ultimately to the achievement of the goal.

This regression property combines considerations that are actually unrelated to one another – the capacity of rules to enable the conditions of other rules, and the significance of rule ordering. In our framework the disposition of rules to become enabled and of their actions to promote progress towards the goal is not enforced *a priori*, but is instead only expected to emerge as a natural outcome of the design process. Moreover, our treatment attaches no goal-related significance to rule ordering. Indeed, the ordering of rules is immaterial if perceptions are pairwise mutually exclusive. For us, the only benefit of rule ordering is that it caters for the suppression within any rule of those perceptual conjuncts known, by default, to be satisfied by virtue of some earlier rule having been enabled with their logical complements satisfied. It is, therefore, merely the analogue of the default 'negation-by-failure' rule employed to confer programming economy in formalisms like Prolog. Altogether, our view is that the regression property is overly restrictive as an *a priori* constraint upon robot design.

The main contributions of our framework to the design of TR-robots are the following. First, our method of accommodating exogenous behaviour, including multi-robot interaction, solely through the special `x`-action and its associated clone-consistency principle. Second, our method of supporting inter-robot communication solely through suitable enrichment of perceptions. Both of these provisions extend substantially the power of the framework without requiring any extension of the robots' basic structure or any additional mechanisms for their implementation. Third, our precise analytical procedure for evaluating their policies. In all of these we have sought to maintain maximum simplicity in both concept and formulation.

There are many other issues of interest that we intend to investigate in future work, including the following:

- the use of memory to enlarge a robot's perceptive capability. Accommodating and broadcasting perceptions of the form $m = v$, where m names a memory

store and v names its value, effectively enables robots to access and exploit shared memory;
- the enhancement of the rule language to definite-clause logic, enabling the formulation of recursive hierarchical TR-programs more expressive than flat propositional programs;
- the construction of better algorithms for policy evaluation and optimization in order to favour scalability;
- the empirical testing of policies by simulation, in particular to assess how well the "best" policy for an individually-designed clone behaves when implemented in a multi-clone scenario;
- the establishing of principles for evaluating the efficacy of co-implemented non-cloned robots possessing distinct, individually-designed policies. This is perhaps the hardest task of those listed here but would offer major benefits, since it appears from our own studies that inter-robot cooperation is most useful when the robots have distinct but complementary capabilities.

References

1. R.A. Brooks, *Intelligence without Reason*, in: Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, August 1991, pp. 569-595.
2. R.A. Kowalski and F. Sadri, *From logic programming to multi-agent systems*, in: Annals of Mathematics and Artificial Intelligence, 25, 1999, pp. 391-419.
3. L.P. Kaelbling, M.L. Littman and A.R. Cassandra, *Planning and acting in partially observable stochastic domains*, Artificial Intelligence, 101, 1998, pp. 99-134.
4. K. Broda, C.J. Hogger and S. Watson, *Constructing Teleo-reactive Robot Programs*, in: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000), Berlin, August 2000, pp. 653-657.
5. T. Mitchell, *Reinforcement Learning*, in: Machine Learning (author T. Mitchell), McGraw Hill, 1997, pp. 367-390.
6. N.J. Nilsson, *Teleo-Reactive Programs for Agent Control*, Journal of Artificial Intelligence Research, 1, January 1994, pp. 139-158.
7. N.J. Nilsson, *Teleo-Reactive Programs and the Triple-Tower Architecture*, Electronic Transactions on Artificial Intelligence 5:99-110 (2001)