Received 6 February 2009
Revised 17 August 2009
Accepted 20 August 2009

Deductive verification of simple foraging robotic behaviours

Abdelkader Behdenna
Departement d'informatique de l'Ecole Normale Supérieure de Cachan, Cachan, France, and

Clare Dixon and Michael Fisher
Department of Computer Science, University of Liverpool, Liverpool, UK

Abstract

Purpose – The purpose of this paper is to consider the logical specification, and automated verification, of high-level robotic behaviours.

Design/methodology/approach – The paper uses temporal logic as a formal language for providing abstractions of foraging robot behaviour, and successively extends this to multiple robots, items of food for the robots to collect, and constraints on the real-time behaviour of robots. For each of these scenarios, proofs of relevant properties are carried out in a fully automated way. In addition to automated deductive proofs in propositional temporal logic, the possibility of having arbitrary numbers of robots involved is considered, thus allowing representations of robot swarms. This leads towards the use of first-order temporal logics (FOTLs).

Findings – The proofs of many properties are achieved using automatic deductive temporal provers for the propositional and FOTLs.

Research limitations/implications – Many details of the problem, such as location of the robots, avoidance, etc. are abstracted away.

Practical implications – Large robot swarms are beyond the current capability of propositional temporal provers. Whilst representing and proving properties of arbitrarily large swarms using FOTLs is feasible, the representation of infinite numbers of pieces of food is outside of the decidable fragment of FOTL targeted, and practically, the provers struggle with even small numbers of pieces of food.

Originality/value – The work described in this paper is novel in that it applies automatic temporal theorem provers to proving properties of robotic behaviour.

Keywords Modeling, Robotics, Control applications, Control technology

Paper type Technical paper

1. Introduction

Here, we are concerned with the formal analysis of the high-level behaviour of robots. Our target is to consider swarm behaviour within robotic systems, but from a logical point of view, automatically verifying the behaviour of robot swarms. However, this objective is extremely challenging for practical verification technology due to the large state spaces involved. To allow practical verification we abstract away from some details and begin with a small number of robots and consider their behaviour.

This work was partially supported by EPSRC (UK) Grant EP/D052548. Abdelkader Behdenna was partially supported by the Department of Computer Science at the University of Liverpool.

International Journal of Intelligent Computing and Cybernetics
Vol. 2 No. 4, 2009
pp. 604-643

1756-378X
DOI 10.1108/17563780911005818

These simple foraging robots explore a fixed space, searching for "food", then returning to their "nest" with the food, before embarking on further searching. We analyse such behaviour and then extend the approach to increasing numbers of robots, increasing items of food and real-time considerations. Previous work by some of the authors (Winfield *et al.*, 2005) concentrated on the specification of swarm algorithms, i.e. the formalisation of the steps each robot carries out relating to its movement and direction that should ensure that the group of robots remains clustered together. As mentioned above, the work described here focuses on the specification and verification of a swarm of foraging robots and concentrates on formalising the scenario of robots searching for food and applying temporal deduction methods to these formalisations.

To carry out the specification and verification of swarms of foraging robots we utilise propositional and then first-order linear-time temporal logics (LTLs). In particular, we assume a discrete linear model of time with finite past and infinite future. Thus, models can be viewed as a sequence of propositional logic domains (for propositional temporal logic) or a sequence of first-order domains (for first-order temporal logic (FOTL)). The computational complexity of satisfiability for propositional LTL is PSPACE (Sistla and Clarke, 1985). We target the monodic fragment of FOTL (essentially any subformula occurring in the scope of a temporal operator can have at most one free variable) which is decidable for underlying decidable fragments of first-order logic (Hodkinson *et al.*, 2000). The first-order formulae we obtain are within either the monodic monadic or monodic two variable fragments which both have complexity of satisfiability of EXSPACE (Hodkinson *et al.*, 2003). Having specified the foraging robots in the given temporal logic we try to prove a number of properties about the behaviour of the robots. We use a deductive resolution-based approach which is applicable both to propositional and FOTLs and has implemented provers, e.g. TRP++ and TSPASS, which carry out the proofs automatically. TRP++ (Hustadt and Konev, 2003) is a resolution prover for propositional LTL based on a sound, complete and terminating calculus (Fisher *et al.*, 2001). TSPASS (Ludwig and Hustadt, 2009/2010) is a resolution prover for first-order LTL based on a sound and complete calculus (Konev *et al.*, 2005). The contribution of the paper is the application and analysis of fully automatic temporal deductive provers to the behaviour of foraging robot swarms.

Before describing our formal model in more detail, we briefly review some necessary background.

1.1 Robots and robot swarms

A robot swarm is a collection of (usually simple, often identical) robots working together to carry out some task (Bonabeau *et al.*, 1999; Beni, 2005; Sahin and Winfield, 2008). Each robot has a simple set of behaviours and may be able to interact with each other and the environment. It is non-trivial for designers to formulate individual robot behaviours so that the emergent behaviour of the swarm as a whole achieves the task of the swarm, and that the swarm does not exhibit any undesirable behaviour (Spears *et al.*, 2004).

Rather than using, say, one or two highly complex robots, the use of robot swarms is appealing. This is because if a small number of robots fail then this should not affect the overall behaviour of the swarm, building in fault tolerance. However, it is often difficult to predict the overall behaviour of the swarm. Given the behaviour of

individual robots and their interaction with each other and the environment, we would like to be sure that the swarms do actually behave as the designer intended. This is often investigated by experimenting with real robot swarms or by simulating robot swarms. Both have the disadvantage that experimentation or simulation may help show that the desired behaviour is possible, but does not ensure that it occurs under all circumstances or rule out the possibility that undesirable behaviour may also occur in some situations. An alternative is to formally verify that robot control algorithms, for a swarm of robots, do exhibit the required behaviour. This is the approach we will take in this paper by considering the formal specification and verification of a robot swarm using temporal logics. While we do not model low-level control aspects, we model the behaviour of robots at a sufficiently high level to describe relevant scenarios.

This work is based on a robot foraging scenario and high level control systems described in Liu *et al.* (2007). There are a number of robots that must search for food in a finite arena. Robots may be searching for food (involving one of several modes such as random walking, scanning the arena), returning home, or resting (i.e. the robot is in the nest). Parameters relating to maximum searching time (K_{hs}), maximum resting time (K_{hr}) and maximum scanning time (K_{ht}) control how long the robots can remain in one particular phase. In Liu *et al.* (2007), the authors are interested in examining the number of robots foraging as compared to the number of robots resting, with respect to several parameters. These parameters are split into three groups: internal, i.e. whether the robot has retrieved food recently itself or not; environmental, i.e. whether it collided with other robots whilst searching; and social, whether other robots have been successful in finding food or not. These parameters can be used to adjust the maximum resting and searching times, dynamically affecting the numbers of robots foraging and resting. Additionally, the purpose of foraging for food is to provide energy for the robot swarm. However, searching for food, bringing food back to the nest and even resting all require energy so the overall energy of the swarm is considered.

Formalising robot control is complex, involving potentially large numbers of robots, each located at a particular point in the arena, numbers of pieces of food located in the arena, algorithms relating to robot movement, avoidance, searching for food and moving home, and parameters relating to the current and maximum times each robot can spend in each phase. To enable us to be able to formulate the problem in a logic that is decidable, and to allow automatic provers to carry out proofs within a reasonable time, we must abstract away from some of this detail. For example, we avoid representing low level control relating to explicit movement in the arena but do formalise the basic state structure shown in Figure 1 of Liu *et al.* (2007). This shows the robot's activity as one of the previously mentioned phases, e.g. scanning the arena, random walk, moving home, etc.

1.2 Propositional LTL

Propositional temporal logic (Emerson, 1990; Fisher, 2007) is an extension of classical propositional logic with operators that deal with time. Propositions are true or false at each moment in time. For example, some proposition might be true now, but false in every future moment in time. To define a temporal logic, we must be precise about this underlying temporal structure and so, for simplicity, we consider a discrete, linear model of time, isomorphic to the natural numbers (Gabbay *et al.*, 1980). Within this model, there is an initial moment in time, called "0", and, for any particular moment in

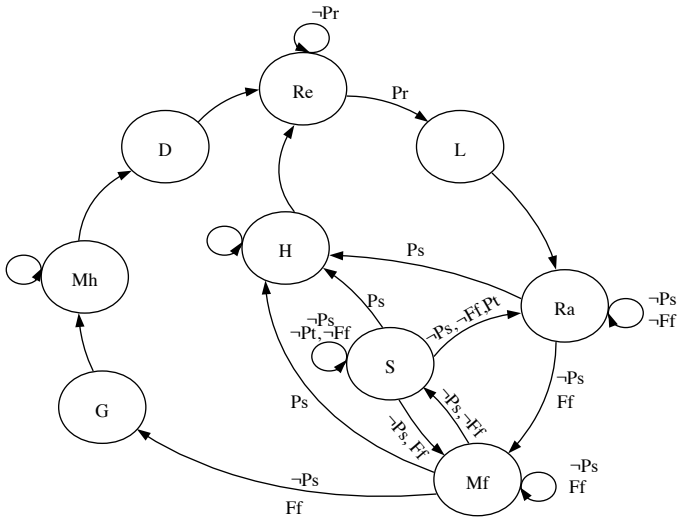


Figure 1.
Abstract behaviour of a robot

time, i, the successor moment (or next moment in the linear sequence) is termed "$i + 1$" (Gabbay *et al.*, 1980; Pnueli, 1981). Thus, we consider models as infinite sequences of states where propositions may or may not hold in each state.

The syntax of such a propositional LTL is based on a set *PROP* of proposition symbols. Let:

$$PROP = \{p_1, p_2, p_3, \ldots\}$$

be a countably infinite set of propositional variables. In addition to the usual Boolean operators, $\wedge$ ("and"), $\vee$ ("or"), and $\neg$ ("not"), we have the future-time temporal operators, $\bigcirc$ ("in the next moment"), $\Box$ ("always") and $\Diamond$ ("eventually"). Formulae in LTL are constructed recursively as follows (where ϕ and ψ are LTL formulae):

$$Formulae ::= p_i | \neg \phi | \phi \wedge \psi | \phi \vee \psi | \bigcirc \phi | \Box \phi | \Diamond \phi$$

Turning to the semantics for LTL, a structure (or model) for LTL is an infinite sequence of sets of propositions which hold at each moment in time. We define an interpretation function σ that takes some moment in time (a natural number $\mathbb{N}$) and returns the set of propositions which hold at that moment, $\sigma : \mathbb{N} \rightarrow \mathcal{P}(PROP)$ (where $\mathcal{P}$ denotes the powerset). The satisfaction relation, " $\models$ ", relates a model of the above form and a specific moment in time, say i, to formulae in LTL. For elements of PROP the semantics is straightforward:

$$\sigma, i \models p \quad iff \quad p \in \sigma(i).$$

Assuming the usual semantics of $\neg$, $\wedge$ and $\vee$, we now give the semantics for the three principal temporal operators:

$$\sigma, i \models \bigcirc \phi \quad iff \quad \sigma, i + 1 \models \phi$$

$$\sigma, i \models \Diamond \phi \quad iff \quad \exists j \geq i . \sigma, j \models \phi$$

$$\sigma, i \models \Box \phi \quad iff \quad \forall j \geq i . \sigma, j \models \phi$$

Typically we consider the satisfiability of a formula within a model, when assessed at the first moment in time, i.e. "0". Thus, we often write "$\sigma \vDash \phi$" instead of "$\sigma, 0 \vDash \phi$". A formula ϕ is LTL-satisfiable if, and only if, there exists a σ such that $\sigma \vDash \phi$. A formula ϕ is LTL-valid if, and only if, for all structures σ, $\sigma \vDash \phi$.

We also define a literal be either a proposition or its negation.

1.3 Automated deduction in LTL

Rather than experimentation with, or simulation of, robot swarms we propose to specify the swarms using temporal logic and verify that the swarms do satisfy the required properties. This has the advantage of being able to verify properties about the swarm algorithms before they are actually implemented in real robots or simulations. Further, logical verification captures all specified behaviours rather than just considering one path through the behaviours as the experimentation/simulation approach would do. Obviously trying to carry out such proofs by hand would be time consuming and prone to error so we use an automatic prover for the logic LTL.

In this paper, we will consider deductive verification in LTL, in particular, based on a temporal resolution calculus (Fisher *et al.*, 2001). The idea is as follows. Given a specification of a system in LTL, S, and a property, P, we wish to prove of that specification we will try to show that $S \Rightarrow P$ is a valid formula. Since the resolution calculus we use is a refutation process, we actually negate the formula we wish to show valid, obtaining $\neg (S \Rightarrow P)$ or equivalently $S \wedge \neg P$, and try to show the negated formula is unsatisfiable. That is, if we add the negation of the property we wish to prove to the specification, once we establish inconsistency, then we know that the non-negated property is a logical consequence of the original specification. In this paper, S corresponds to the logical representation of the foraging robots and P corresponds to something that we want to prove about this, for example, infinitely often a robot deposits food, or a piece of food will eventually be found.

As is usual with resolution calculi, given any LTL formula we must first translate it into a particular normal form. Thus, using the terminology from the above paragraph, we must translate $S \wedge \neg P$ into a normal form. The normal form we translate to is known as SNF (Fisher *et al.*, 2001). This provides formulae that hold in the initial moment of time (known as initial clauses), disjunctions of literals which hold everywhere (known as global clauses), formulae which define what must hold in the next moment given what holds now (step clauses) and formulae which define what must hold in the future given what hold now (sometime clauses). For the purposes of the normal form we introduce a symbol **start** such that **start** holds only at the initial moment in time, i.e.:

$$\sigma, i \vDash \mathbf{start} \quad \textit{iff} \quad i = 0.$$

Formulae in SNF[1] are of the general form $\Box \wedge_i C_i$ (recall $\Box$ means "always" and $\wedge$ denotes "and" or "conjunction") where each C_i is known as a clause and must be one of the following forms:

$$\mathbf{start} \Rightarrow \bigvee_{b=1}^{r} l_b \quad (\text{an } \textit{initial} \text{ clause})$$

$$\mathbf{true} \Rightarrow \bigvee_{b=1}^{r} l_b \quad (\text{a } \textit{global} \text{ clause})$$

$$\bigwedge_{a=1}^{g} k_a \Rightarrow \bigcirc \bigvee_{b=1}^{r} l_b \quad \text{(a } \textit{step} \text{ clause)}$$

$$\bigwedge_{a=1}^{g} k_a \Rightarrow \Diamond l \quad \text{(a } \textit{sometime} \text{ clause)}$$

Here, k_a, l_b, and l are literals. Any LTL formula, ϕ, can be translated into SNF, $\tau(\phi)$, such that ϕ is satisfiable if and only if $\tau(\phi)$ is satisfiable (Fisher *et al.*, 2001).

Next, we repeatedly apply a number of resolution rules which add additional initial, step and global clauses to the clause-set. If we can derive a contradiction, i.e. **start** $\Rightarrow$ **false** then this means that the set of clauses derived from the negation of the original formula is unsatisfiable and the original formula is valid.

TRP++ (Hustadt and Konev, 2003; TRP++, 2002) is a theorem prover for LTL which is implemented in C++ and based on the temporal resolution calculus (Fisher *et al.*, 2001; Konev *et al.*, 2005). Input again uses the SNF normal form but has a different syntax to that given above which is more amenable to mechanisation. The input syntax of TRP++ is given below.

TRP++ input syntax:

FORMULA ::= and (LIST_OF_SNFCLAUSES).

LIST_OF_SNFCLAUSES ::= []|[SNFCLAUSE, ..., SNFCLAUSE]

SNFCLAUSE ::= always(TEMPORALCLAUSE)
|PROPOSITIONALCLAUSE

TEMPORALCLAUSE ::= PROPOSITIONALCLAUSE|GLOBALCLAUSE
|SOMETIMECLAUSE

PROPOSITIONALCLAUSE ::= or ([LITERAL, ..., LITERAL])

GLOBALCLAUSE ::= or ([LITERAL, ..., LITERAL,
next(LITERAL), ..., next(LITERAL)])

SOMETIMECLAUSE ::= or([LITERAL, ..., LITERAL, sometime(LITERAL)])

LITERAL ::= not(IDENTIFIER)|IDENTIFIER

IDENTIFIER ::= [a − z0 − 0]+

Examples of translating initial, global, step and sometime clauses (on the left) into the syntax for TRP++ input (on the right) are given below:

start $\Rightarrow p_1 \vee p_2$	or([p1, p2])
true $\Rightarrow (p_1 \vee p_2 \vee p_3)$	always(or([p1, p2, p3]))
$p_1 \wedge p_2 \Rightarrow \bigcirc(p_3 \vee p_4)$	always(or([not(p1), not(p2), next(p3), next(p4)]))
$p_1 \wedge p_2 \Rightarrow \Diamond p_3$	always(or([not(p1), not(p2), sometime(p3)]))

Finally, note that we could have chosen other methods for carrying out verification within LTL. For example, rather than using a resolution-based approach we could have used a tableau-based system for LTL (Wolper, 1985; Schwendimann, 1998; Janssen, 1999). Implementations based on tableau for LTL are available from The Logics WorkBench (Balsiger *et al.*, 1998) and the Tableau Workbench (Abate and Goré, 2003). Alternatively model checking (Clarke *et al.*, 2000) is a popular technique for showing the satisfiability of a temporal formula given model of the system it should be checked on. These areas are discussed further in Section 7.

2. A simple model of robotic behaviour

We base our model of foraging robotic behaviour on that given by Liu *et al.* (2007). There a swarm of foraging robots is considered, with all robots having identical behaviour. The behaviour is quite simple, consisting of a series of phases as the robot attempts to find food and return to its "nest". The transition system in Figure 1 shows the high level behaviour of each robot. This is adapted from the transition system in Liu *et al.* (2007), explicitly representing transitions from a state to itself where necessary and representing the conditions on transitions using propositions as discussed below. To explain the elements within Figure 1, we consider first the states/phases, then the transitions.

Phases. Each robot moves through a series of phases. Within the model, each state represents such a phase and the different possible phases are described as follows:

- Re (Resting) the robot stays in the nest without moving, and rests within the resting time.
- L (Leaving the nest) the robot makes a movement and leaves the nest.
- Ra (Random walk/move) the robot makes successive random moves.
- Mf (Move to food) when the robot has found some food, it moves forward until it can access it.
- S (Scanning) if the food is lost, the robot will stop moving and will scan the arena in order to find food.
- H (Homing) when the searching time is over, and if the robot has not found food, then it comes back to the nest.
- G (Grab) the robot has reached food and grabs it.
- Mh (Move to home) the robot brings the food home (the "nest").
- D (Deposit) the robot deposits the food in the nest.

Transitions. The robot takes transitions to move between phases. These transitions correspond to edges within Figure 1 and, while many transitions are straightforward, some have conditions, such as Pr or Ff. To explain such conditions, we must first mention some of the key constants within the underlying robotic model (Liu *et al.*, 2007):

- K_{hr} (Resting time) the maximum time a robot can stay in the nest – this relates to K_r, the variable representing the actual resting time.

K_{hs} (Searching time) the maximum time a robot can look for food – this relates to K_s, the variable representing the actual searching time.

K_{ht} (Scanning time) the maximum time a robot can scan the arena without moving – this relates to K_t, the variable representing the actual scanning time.

We define the propositions Pr, Ps and Pt relating to each of these timing constraints and a proposition Ff which together provide the conditions on transitions, namely:

Pr $K_r \geq K_{hr}$, meaning that the resting time is over.

Ps $K_s \geq K_{hs}$, meaning that the searching time is over.

Pt $K_t \geq K_{ht}$, meaning that the scanning time is over.

Ff (Foundfood), meaning that the robot has located some food.

Consider the transition from Mf to S labelled by $\neg$ Ps and $\neg$ Ff. This transition represents a change of phase/state from moving towards food to scanning the arena when the searching time (Ps) is not yet over but the robot has, for some reason, lost the food it was moving towards. Maybe this robot has lost sight of the food or another robot has collected the food, etc. Similarly, from the state Mf when $\neg$ Ps and Ff hold, i.e. the searching time is not yet over and the robot has found food, the robot may either stay in the state Mf by following the transition to itself or may move to the state where it can grab the food, G. This allows us to model the robot remaining differing lengths of time in the move to food state before being able to grab food. Something similar happens with the transitions from move to home, Mh, and itself and move to home and deposit food, D. There are no additional conditions on these transitions allowing us to represent different numbers of cycles round the move to home state before the robot takes the transition into the deposit state. The number of states in the transition system could perhaps be reduced, for example the removal of the state L (leaving the nest). However, we chose to leave the transition system as close to the original presentation in Liu *et al.* (2007) as possible.

In Sections 2.1 and 2.2 we provide temporal formulae to represent this behaviour. Section 2.1 defines logical formulae representing the transitions from Figure 1. Section 2.2 presents logical formulae required to remove paths through the transition that allow undesirable behaviour, for example, robots staying in the nest forever. Also they are used to enforce conditions on the underlying structure, for example, that a robot can be in exactly one state at at any moment. The formulae in Section 2.2 we term constraints as the representation from Section 2.1 is further constrained by these formulae. Both the sets of formulae from these two sections are part of the specification of the system (termed S in Section 1.3).

2.1 Temporal specification of the transitions

Given the above definitions of the phases and conditions on transitions, we can represent the phases as propositions and the transitions themselves as sets of LTL formulae, as follows. We assume each of the following implications is within the scope of a $\Box$ operator. This is so that they must hold at every moment in time:

$$(\mathrm{Re} \wedge \mathrm{Pr}) \Rightarrow \bigcirc \mathrm{L} \tag{1}$$

$$(\mathrm{Re} \wedge \neg \mathrm{Pr}) \Rightarrow \bigcirc \mathrm{Re} \quad (2)$$

$$\mathrm{L} \Rightarrow \bigcirc \mathrm{Ra} \quad (3)$$

$$(\mathrm{Ra} \wedge \neg \mathrm{Ps} \wedge \neg \mathrm{Ff}) \Rightarrow \bigcirc \mathrm{Ra} \quad (4)$$

$$(\mathrm{Ra} \wedge \mathrm{Ps}) \Rightarrow \bigcirc \mathrm{H} \quad (5)$$

$$(\mathrm{Ra} \wedge \neg \mathrm{Ps} \wedge \mathrm{Ff}) \Rightarrow \bigcirc \mathrm{Mf} \quad (6)$$

$$(\mathrm{Mf} \wedge \neg \mathrm{Ps} \wedge \mathrm{Ff}) \Rightarrow \bigcirc (\mathrm{Mf} \vee \mathrm{G}) \quad (7)$$

$$(\mathrm{Mf} \wedge \neg \mathrm{Ps} \wedge \neg \mathrm{Ff}) \Rightarrow \bigcirc \mathrm{S} \quad (8)$$

$$(\mathrm{Mf} \wedge \mathrm{Ps}) \Rightarrow \bigcirc \mathrm{H} \quad (9)$$

$$(\mathrm{S} \wedge \mathrm{Ps}) \Rightarrow \bigcirc \mathrm{H} \quad (10)$$

$$(\mathrm{S} \wedge \neg \mathrm{Ps} \wedge \mathrm{Ff}) \Rightarrow \bigcirc \mathrm{Mf} \quad (11)$$

$$(\mathrm{S} \wedge \neg \mathrm{Ps} \wedge \neg \mathrm{Pt} \wedge \neg \mathrm{Ff}) \Rightarrow \bigcirc \mathrm{S} \quad (12)$$

$$(\mathrm{S} \wedge \neg \mathrm{Ps} \wedge \mathrm{Pt} \wedge \neg \mathrm{Ff}) \Rightarrow \bigcirc \mathrm{Ra} \quad (13)$$

$$\mathrm{H} \Rightarrow \bigcirc (\mathrm{H} \vee \mathrm{Re}) \quad (14)$$

$$\mathrm{G} \Rightarrow \bigcirc \mathrm{Mh} \quad (15)$$

$$\mathrm{Mh} \Rightarrow \bigcirc (\mathrm{Mh} \vee \mathrm{D}) \quad (16)$$

$$\mathrm{D} \Rightarrow \bigcirc \mathrm{Re} \quad (17)$$

This set of temporal formulae describes the basic structure as in Figure 1. Each clause defines one or more transitions where clauses (1) and (2) represent transitions from the state resting (Re), clause (3) represents the transition from the state leaving nest (L), clauses (4)-(6) represent transitions from the state random move (Ra), clauses (7)-(9) represent transitions from the state move to food (Mf), clauses (10)-(13) represent transitions from the state scanning (S), clause (14) represents the transitions from the state homing (H), clause (15) represents the transition from the state grab (G), clause (16) represents the transitions from the state move to home (Mh), and clause (17) represents the transition from the state deposit (D).

For example, clause (1) (relating to resting) states that it is always the case that if the robot is resting and the resting time is over then in the next moment the robot is leaving the nest. This corresponds to the edge labelled with Pr between the states Re and L in Figure 1. However, we need to provide additional constraints, primarily to avoid unwanted infinite behaviour (e.g. looping forever in one phase).

2.2 Constraints on robot behaviours

In this section, we provide the definitions and the descriptions for the additional constraints we have imposed. In order to be able to prove some useful properties of potentially infinite behaviours, particularly the absence of unwanted infinite loops or the certainty of finding food, we must add further constraints. We note that these are still part of the system specification (termed *S* in Section 1.3) but they constrain the

robot behaviours to what we would usually expect, hence the terminology constraints. These correspond to acceptance conditions which take the state-machine towards more complex *Büchi Automata* (Sistla *et al.*, 1987).

We can define several types of constraint: those related to the structure of the transition system, those modelling the numeric values, and others which are used to avoid unwanted infinite loops. We will describe these below.

2.2.1 Structural constraints. The first pair of constraints are "construction" constraints specifying that, at each moment, just one state is occupied while the others are empty (i.e. a robot is in exactly one current phase):

$$\Box(\mathrm{Re} \vee \mathrm{L} \vee \mathrm{Ra} \vee \mathrm{Mf} \vee \mathrm{S} \vee \mathrm{H} \vee \mathrm{G} \vee \mathrm{Mh} \vee \mathrm{D}) \tag{18}$$

$$\forall X, Y, X \neq Y, \in \{\mathrm{Re}, \mathrm{L}, \mathrm{Ra}, \mathrm{Mf}, \mathrm{S}, \mathrm{H}, \mathrm{G}, \mathrm{Mh}, \mathrm{D}\} . \Box(\neg X \vee \neg Y) \tag{19}$$

The formula (18) ensures that one or more of the propositions representing states hold at every moment and equation (19) represents that for every pair of (state) propositions at least one must be false, i.e. at most one of these propositions must be true.

2.2.2 Modelling numeric constraints. The next constraints are the ones relating to the numeric variables and parameters such as searching time, resting time and scanning time. To reduce complexity and remain decidable, we try to simplify our formalisation as much as possible. One way to do this is to find an abstract way to represent the numeric values present in the original system (Liu *et al.*, 2007). As mentioned earlier, rather than allowing explicit numeric values we use propositions to represent the facts that the searching, resting and scanning times are over (Ps, Pr, Pt, respectively). Related to this we impose the next six constraints (20)-(25).

Assume we begin with initial values of K_s, K_r and K_t, and that those values will increase as the time progresses, and may eventually exceed the constants K_{hs}, K_{hr} and K_{ht}. Abstracting away from the numeric values we add constraints ensuring that the searching, resting and scanning times are over infinitely often (i.e. there cannot be some execution where at some point in time, in the future, the maximum searching, resting and scanning times are never reached):

$$\Box\Diamond\mathrm{Pr} \tag{20}$$

$$\Box\Diamond\mathrm{Ps} \tag{21}$$

$$\Box\Diamond\mathrm{Pt} \tag{22}$$

From constraint (20), for example, if we have Pr initially false, then it will become true in the future, until the instant when we reach the state L, where it becomes false equation (23). Similarly for equation (21) and, by analogy, for equation (22). This means that we initially have $K_r < K_{hr}$, then $K_r \geq K_{hr}$ in the future (the resting time is over; the robot has to go out of the nest and begin searching) until we reach the state L. The following three constraints force Pr, Ps and Pt to become false in states L, L and Ra, respectively, and disallow infinite paths though these states where Pr, Ps and Pt remain true forever:

$$\Box(\mathrm{L} \Rightarrow \neg\, \mathrm{Pr}) \tag{23}$$

$$\Box(\mathrm{L} \Rightarrow \neg\, \mathrm{Ps}) \tag{24}$$

$$\Box(\mathrm{Ra} \Rightarrow \neg\, \mathrm{Pt}) \tag{25}$$

These formulae effectively represent a reset of the underlying numerical counters.

2.2.3 Avoiding infinite looping. We now consider other constraints that will be useful in proving properties of our transition system and stop a robot from remaining forever in one state (e.g. H, Mh), for instance. Here, is the list of those constraints:

$$\mathrm{H} \Rightarrow \Diamond\mathrm{Re} \quad \textit{If we reach the state } \mathrm{H}, \textit{ then at sometime in the future we will be resting} \tag{26}$$

$$\mathrm{Mh} \Rightarrow \Diamond\mathrm{D} \quad \textit{If we reach the state } \mathrm{Mh}, \textit{ then at sometime in the future, we will deposit some food in the nest} \tag{27}$$

$$\mathrm{Ra} \Rightarrow \Diamond(\mathrm{Ra} \wedge \mathrm{Ff} \wedge \neg\, \mathrm{Ps}) \quad \textit{If a robot begins a random move, then at sometime in the future, it will find food before the end of the searching time} \tag{28}$$

$$\mathrm{Mf} \Rightarrow \Diamond\mathrm{G} \quad \textit{If we begin to move to food, then at sometime in the future, we will grab it} \tag{29}$$

Constraints (26) and (27) avoid infinite looping on the corresponding states. Constraint (28) models the fact that if we are in the state Ra at some point now or in the future we will again reach state Ra and be able to take the transition to the state Mf, i.e. now or in the future the robot will reach the state Ra before the end of its searching time (so $K_s < K_{hs}$ and $\neg$ Ps) and it will find food (Ff becomes true) allowing the transition to Mf in the next moment. This avoids infinite looping on the state Ra itself and an infinite loop around the states Re, L, Ra, and H. Finally, equation (29) guarantees that, if at a particular moment a robot reaches the state move to food (Mf), then it will grab food at some point in the future.

Note, the above makes the assumption that food can be found infinitely often in the arena. The constraints imposed on the transition system mean that we cannot loop infinitely on the state Re (constraint (20)), Ra (constraint (21)), Mf (constraint (21) or (29)), S (constraint (21) or (22)), H (constraint (26)), or Mh (constraint (27)). Thus, from any point in the transition system we can reach state Ra. The constraint (28) ensures that we must be able to find food (before searching time runs out) in the future.

2.2.4 Alternative constraints. Above we have defined several additional constraints to be added to the basic transition system. However, there are a wide variety of such constraints available. The set that is chosen most likely corresponds to the abstract view of the system behaviour that we wish to adopt. Below we will consider one particular alternative view and show that such views can also be captured as temporal logic formulae.

Let us again examine Figure 1. As we have seen, there are several states with self-loops. In the above section we added temporal constraints to avoid perpetually looping on one of these states. But these self-loops essentially correspond to one of three things:

(1) abstractions of real-time characteristics, for example "we can stay in this state, but for at most N time steps";

(2) abstractions of probabilistic characteristics, for example "we have a $P\%$ chance of finding food"; and

(3) a combinations of (1) and (2).

Thus, real-time constraints can be characterised by simple "$\Diamond$" formulae saying that the looping state will be left at some point in the future (thus abstracting away from the particular timing concerns):

$$\mathrm{H} \Rightarrow \Diamond \neg \mathrm{H} \tag{30}$$

$$\mathrm{Mh} \Rightarrow \Diamond \neg \mathrm{Mh} \tag{31}$$

$$\mathrm{Re} \Rightarrow \Diamond \neg \mathrm{Re} \tag{32}$$

$$\mathrm{S} \Rightarrow \Diamond \neg \mathrm{S} \tag{33}$$

Now, as we do not have any purely probabilistic characteristics, we must consider the option (3) above, namely states with combined probabilistic and real-time characteristics. In particular, let us examine states Ra and Mf. Both have a real-time aspect, in that we can only loop on these states for a bounded length of time, and so must move off eventually. Thus, we could happily add:

$$\mathrm{Ra} \Rightarrow \Diamond \neg \mathrm{Ra} \tag{34}$$

$$\mathrm{Mf} \Rightarrow \Diamond \neg \mathrm{Mf} \tag{35}$$

However, there is the probabilistic aspect to consider. When we leave either of these states there are at least two possible next states. One option corresponds to having achieved something (e.g. found food) while the others correspond to failure, e.g. the searching time is over or the food has been lost. We could, given all the constraints added so far in this section, happily navigate through the state machine without ever finding or depositing any food. However, we really wish to add a constraint corresponding to the non-zero probability of achieving what we want. Thus, if we keep on searching/trying then this non-zero probability means that we will eventually achieve what we require. Thus, we instead add:

$$\mathrm{Ra} \Rightarrow \Diamond \mathrm{Mf} \tag{36}$$

$$\mathrm{Mf} \Rightarrow \Diamond \mathrm{G} \tag{37}$$

Thus, for equation (36) if we keep on randomly searching, then eventually we will move to some food, and for equation (37) if we move to food then we will eventually be able to grab it. Note that due to the constraint (19) these imply equations (34) and (35), respectively.

Now, given the temporal constraints in this section plus equations (18) and (19) from the previous section we can, prove the constraints (20), (26)-(29), from the previous section.

We could also remove some of the constraints and see what properties hold. For example, if we remove the "probabilistic" constraints (36) and (37) above, we can establish that:

$$\Box \Diamond \mathrm{Re} \wedge \Box \neg \mathrm{D}$$

is satisfiable, i.e. we can return to the nest infinitely often, but never with food.

The choice of constraints is really a matter for the specifier, as the constraints should correspond with the abstractions required. An alternative choice to the constraints (20)-(29) would be the single constraint such as $\Box\Diamond G$. This would mean that any of the infinite loops, previously discussed, could not occur and replaces constraints (20)-(22) and (26)-(29) by just one. However, we chose not to do this in the propositional case as we felt the seven constraints were more easily justified and closer to the description of the foraging robots. In the next section, we will consider automated verification and, for simplicity, will use the constraints from the previous section. Later in this paper we will abstract away from these details within one robot's state machine and simply use $\Box\Diamond G$.

3. Verifying properties

Since we have defined our transition systems (1)-(17) and a complete set of constraints (18)-(29), together providing the system specification, we are now able to prove some properties. These are things we want to show holding (i.e. are satisfiable or valid) given the specification from Section 2 and represent *P* mentioned in Section 1.3. We will begin by examining a selection of properties that will be first justified informally and will then move on to the use of a prover for LTL to carry out automated proof. Using the constraints we specified we tried to remove the possibility of the robot remaining in an unwanted infinite loop (for example forever resting). Thus, one of the properties we aim to prove is that the robot deposits food infinitely often $\Box\Diamond D$.

3.1 Looping behaviours

Property 1. There is no infinite path in the transition system which does not visit the state D infinitely often.

Proof (sketch). We will prove this by examining all infinite loops in the transition system. Below is a summary of each of the infinite loops, characterised by a particular temporal formula, together with a sketch of how a contradiction is reached:

- An infinite loop remaining in the state Re: characterised by $\Diamond\Box Re$. This implies $\Diamond\Box \neg Pr$, i.e. that at some point in the future we will always have $\neg Pr$, which contradicts equation (20).
- An infinite loop remaining in the state Ra: characterised by $\Diamond\Box Ra$. This implies $\Diamond\Box(\neg Ps \wedge \neg Ff)$, contradicting equation (21).
- An infinite loop remaining in the state S: characterised by $\Diamond\Box S$. This implies $\Diamond\Box(\neg Ps \wedge \neg Pt \wedge \neg Ff)$, i.e. that at some point in the future we will always have $\neg Ps$, which contradicts equation (21) and sometime in the future we will always have $\neg Pt$, which contradicts equation (22).
- An infinite loop remaining in the state H: characterised by $\Diamond\Box H$. This contradicts equations (19) and (26).
- An infinite loop remaining in the state Mf: characterised by $\Diamond\Box Mf$. This implies $\Diamond\Box(\neg Ps \wedge Ff)$, which contradicts equation (21).
- An infinite loop remaining in the state Mh: characterised by $\Diamond\Box Mh$, contradicting equations (19) and (27).
- An infinite loop through the states Re, L, Ra and H. This implies $\Diamond(Ra \wedge (\neg(\Diamond(Ra \wedge Ff \wedge \neg Ps))))$ which leads to a contradiction with equation (28) since we will never reach the state Mf from Ra.

- Finally, consider all the loops involving the states {Re, L, Ra, Mf, S, H} with at least one occurrence of Mf. The infinite loops in the states Re, Ra, Mf, S or H have been dealt with above. As this category of loops visits Mf at least once, the constraint (29) means at some point in the future we must move to G which contradicts our assumption of infinite looping in this set of state. For example, consider infinite sequences of states (constructed from the following state names where Re^n means n repetitions of state Re where $n \geq 0$) $(Re^n \cdot L \cdot Ra^m \cdot Mf^k \cdot H^l)$ repeated infinitely. These loops do not involve any occurrence of D. Sequences that do not include the state Mf have already been considered above, the remaining loops/paths contradict (29).

Thus, we have proved the following property: $\Box\Diamond D$. This means that, from every state we will always reach the state D at some point in the future. So as a conclusion, if there is food in the arena, it will be always found and deposited in the nest. □

Property 2. $\Box\Diamond Ra$.

Proof. Since we have shown $\Box\Diamond D$, and $\neg\Diamond\Box Re$, from the temporal representation of the transition system (i.e. from equations (17), (1), (3), (19) and (20)) it follows that $\Box\Diamond Ra$.

We have already shown that there is no infinite loop that does not contain the state D. So, we can be sure that a path will always lead to the state Re:

If there is food, we can be sure that a robot will find it and bring it to the nest within a finite time.

And so on. We can continue developing proofs in this way. However, manually carrying out such proofs is tedious and error prone. So, we next move towards automating this process of proof. □

3.2 An automatic proof for our properties

Now we carry out some of the proofs using a resolution-based theorem prover for LTL, called TRP++ (Hustadt and Konev, 2003). We will formalise our transition system in the syntax of TRP++ in order to be able to use it to confirm the proofs of the previous part. First we should ensure that a robot is in exactly one state at each moment using equations (18) and (19). For example, the clause below formalises that we cannot be in the states Rest and Deposit at the same time (with the corresponding temporal formula on the right):

$$\text{always(or([not(re), not(d)]))} \quad \Box(\neg Re \vee \neg D)$$

The following clause ensures that we are in one of the prescribed states at every moment:

$$\text{always(or([re, l, ra, mf, s, h, g, mh, d]))} \quad \Box(Re \vee L \vee Ra \vee Mf \vee S \vee H \vee G \vee Mh \vee D)$$

Then we just translate the transition systems (1)-(17) before adding the constraints (19)-(29). Below are three examples:

(1) The clause, (1), that formalises the transition between the states Re and L using the condition Pr:

$$\text{always(or([not(re), not(pr), next(l)]))} \quad \Box((Re \wedge Pr) \Rightarrow \bigcirc L)$$

(2) How the constraints, (20) and (23), relating to Pr are formalised:

$$\text{always(or([sometime(pr)])),} \quad \Box(\Diamond Pr)$$

$$\text{always(or([not(l), not(pr)]))} \quad \Box(L \Rightarrow \neg Pr)$$

(3) If we reach the state H then, at sometime in the future, we will get out of this state, (26):

$$\text{always(or([not(h), sometime(re)]))} \qquad \Box(\text{H} \Rightarrow \Diamond\text{Re})$$

And so on for the translation of the system into the syntax of TRP++. Finally, we add a formula representing the negation of the property to be proved. For example, if we wish to prove ◇D, then we add its negation □ ¬ D stating that "the state D is never reached", i.e. "always(or([not(D)])". Once executed, TRP++ answers that this set of formulae is unsatisfiable, so we can conclude that the state D will be eventually reached.

The tests have been performed on a PC with a 2.13 GHz Intel Core 2 Duo E6400 processor, 3GB main memory, and 5GB virtual memory running Fedora release 9 with a 32-bit Linux kernel.

3.3 Some results

Next, we will consider what we can prove with TRP++, and what time it takes to do this automatically.

We leave consideration of multiple robots, and multiple pieces of food, until Section 4 and here just address the basic case discussed above. Here, we assume that there is always at least one piece of food in the arena. The aim is to prove, as we did by hand in Section 3.1, that the robot will deposit food infinitely often. We write the formalisation for TRP + + for one robot, as we have just explained, and then add a variety of formulae, in order to see if they are satisfiable or not.

In the following if we state a formula X is satisfiable it means that $S \wedge X$ is satisfiable (i.e. it has a model) where S is the conjunction of formulae representing the specification of the robot (here the formulae representing the transitions system for one robot (equations (1)-(17)) plus the constraints we introduced (equations (20)-(29)). If it is stated that a formula X is unsatisfiable it means that $S \wedge X$ is unsatisfiable (i.e. it does not have a model) and that $(S \Rightarrow \neg X)$ is a valid formula (i.e. it is true for all models).

We have considered the results and timings for several properties:

- The formula $\Box \neg X$ (i.e. "we never reach the state X") is satisfiable if, and only if, $X = \text{S}$ or $X = \text{H}$. This means that we are sure that during an infinite path any robot will reach every state, except S and H, which are "optional".
- We show that, given the constraints, it is not possible to remain in some state for ever by proving that $\Box X$ is not satisfiable for any state X. Thus, $(S \Rightarrow \Diamond \neg X)$ is valid.
- For every state X, the formula $\Box\Diamond X$ is satisfiable. Thus, for each state it is possible, reach that state in the system infinitely often.
- For every state X except $X = \text{S}$ or $X = \text{H}$, the formula $\Diamond\Box \neg X$ is unsatisfiable. Thus, for the states X other than S or H, $(S \Rightarrow \Box\Diamond X)$ is valid. In particular, $\vdash (S \Rightarrow \Box\Diamond D)$, i.e. the system reaches the deposit state infinitely often.

Thus, we have carried out automated analysis of the above, for only one robot and ignoring the number of pieces of food. Sample results are given in Table I. As can be seen, deciding on these properties is quite quick.

4. More robots and more food

In this section, we expand upon the above robotic model. While we have automatically proved simple properties of our previous model, using the temporal resolution prover TRP + + we have, up until now, made several simplifying assumptions implicit in the model. In particular, we have assumed that there is always food in the arena without taking care of how it is grabbed and any potential conflict with other robots over this food. We will now try to show how we can have a specific number of robots and items of food and that, in our formalisation, those pieces of food will be always grabbed. Since we have more than one piece of food and more than one robot, we will use the same propositions as in the previous model, but we will add an integer index (over a finite range) to distinguish each robot or piece of food. Thus, for example, Re_i denotes that the ith robot is resting. In addition, we assume that all pieces of food have to be in the nest before any are re-generated in the arena. However, later we relax this assumption.

The underlying representation of each robot and each piece of food is a transition system which will then be represented by temporal logic formulae. The combination of these is synchronous, i.e. every robot or piece of food will make a move (follow a transition) at the same time.

4.1 The formalisation: a transition system for each piece of food

To identify pieces of food we will define a transition system which deals with the state of a piece of food, and will later allow this to interact with transition systems for other robots/food. Within this transition system there are three types of states:

(1) IM_i: (In Arena) the food item i is in the arena.

(2) Gb_{ki}: (Grabbed) the food item i is grabbed by robot k.

(3) IN_i: (In Nest) the food item i has been deposited in the nest.

To illustrate the food transition systems, we will just begin with the case of two pieces of food and two robots in the system, as shown in Figure 2. As we can see, both are similar and contain conditions (D_1 and D_2) on some transitions relating to states in the robot transition systems. The robots have no way to communicate or to "know" the situation of the whole system.

Similar to our formalisation of the transition system relating to robot phases, we must formalise that each piece of food can only be in one of the four states at any moment. In particular, this disallows a piece of food being grabbed by more than one

Property	Meaning	Satisfiable?	Time (s)
$\Box \neg D$	We never reach the state Deposit	No	0.022
$\Box \neg S$	We never reach the state Scan	Yes	0.015
$\Box D$	We remain forever in the state Deposit	No	0.001
$\Box\Diamond D$	We can infinitely often reach the state Deposit	Yes	0.023
$\Box\Diamond Re$	We can infinitely often reach the state Rest	Yes	0.021
$\Diamond\Box \neg D$	Sometime we do not visit the state Deposit again	No	0.084
$\Diamond\Box \neg S$	Sometime we do not visit the state Scan again	Yes	0.022
$\Diamond\Box \neg Ff$	Sometime Foundfood is always false	No	0.066

Table I.
Sample tests from applying TRP++ to the "one robot" model

robot. Additionally, we must formalise that a robot can only grab one piece of food at a time as follows. This adds conditions between the food transition systems:

$$\text{always(or([not(gb11), not(gb12)]))} \quad \Box(\neg\, Gb_{11} \vee \neg\, Gb_{12})$$

$$\text{always(or([not(gb21), not(gb22)]))} \quad \Box(\neg\, Gb_{21} \vee \neg\, Gb_{22})$$

Further, we must link the transition systems for robots to the transitions systems for food. In particular, robot 1 can be in the states G_1 or Mh_1 or D_1 if, and only if, some piece of food has been grabbed by robot 1, i.e.:

$$\Box((G_1 \vee Mh_1 \vee D_1) \Leftrightarrow (\, Gb_{11} \vee Gb_{12})) \tag{38}$$

This can be represented by the following TRP + + clauses:

always(or([not(g1), gb11, gb12])),

always(or([not(mh1), gb11, gb12])),

always(or([not(d1), gb11, gb12])),

always(or([g1, mh1, d1, not(gb11)])),

always(or([g1, mh1, d1, not(gb12)])).

Likewise, for robot 2 we have:

$$\Box((G_2 \vee Mh_2 \vee D_2) \Leftrightarrow (\, Gb_{21} \vee Gb_{22})) \tag{39}$$

and their TRP + + representation.

Turning to the general case, with r robots and f pieces of food, we get a more general transition system, as shown in Figure 3. Here we consider just the transition system for one piece of food $k \leq f$. Note to take the transition from IN_k to IM_k requires that all the other pieces of food are back in the nest. So a piece of food can only be returned to the arena if all the other pieces of food have been returned to the nest too.

Now we examine the translation of the transition system for food item k into temporal formulae:

$$IM_k \Rightarrow \bigcirc\left(IM_k \vee \left(\bigvee_{1 \leq j \leq r} Gb_{jk}\right)\right) \tag{40}$$

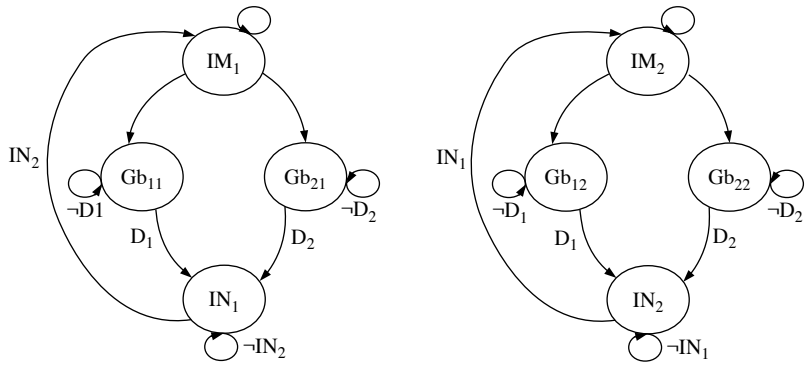
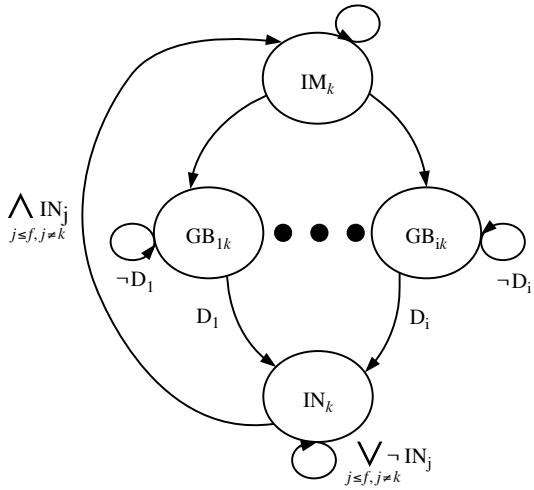


Figure 2.
Transition systems for two pieces of food

Note: In the context of two robots



Piece of food k, total of f pieces of food and r robots

Figure 3.
Transition systems for piece of food k with r robots and f pieces of food

$$(\mathrm{Gb}_{jk} \wedge \neg\, \mathrm{D}_j) \Rightarrow \bigcirc \mathrm{Gb}_{jk} \quad \forall j \in [|1;\mathrm{r}|] \tag{41}$$

$$(\mathrm{Gb}_{jk} \wedge \mathrm{D}_j) \Rightarrow \bigcirc \mathrm{IN}_k \quad \forall j \in [|1;\mathrm{r}|] \tag{42}$$

$$\mathrm{IN}_k \wedge \left(\bigvee_{1 \le j \le \mathrm{f}, j \ne k} \neg\, \mathrm{IN}_j \right) \Rightarrow \bigcirc \mathrm{IN}_k \tag{43}$$

$$\bigwedge_{1 \le j \le \mathrm{f}} \mathrm{IN}_j \Rightarrow \bigcirc \mathrm{IM}_k \tag{44}$$

In the above equation (40) represents the transitions out of the state in arena $\mathrm{IM_k}$, equations (41) and (42) represent transitions out of each state grabbed $\mathrm{Gb_{jk}}$ and equations (43) and (44) represent the transition out of the state in nest $\mathrm{IN_k}$.

The following formulae ensure that each piece of food can only be in one state at any moment. The first formula (45) ensures that the food must be in at least one of the states and the other equations (46)-(49) show that for any pair of states both cannot be true together, i.e. at least one must false:

$$\Box \left(\mathrm{IM}_k \vee \mathrm{IN}_k \vee \bigvee_{i=1}^{r} \mathrm{Gb}_{ik} \right) \tag{45}$$

$$\Box(\neg\, \mathrm{Gb}_{jk} \vee \neg\, \mathrm{Gb}_{ik}) \quad \forall j, i \in [|1;\mathrm{r}|], j \ne i \tag{46}$$

$$\Box(\neg\, \mathrm{IM}_k \vee \neg\, \mathrm{Gb}_{ik}) \quad \forall i \in [|1;\mathrm{r}|] \tag{47}$$

$$\Box(\neg\, \mathrm{IN}_k \vee \neg\, \mathrm{Gb}_{ik}) \quad \forall i \in [|1;\mathrm{r}|] \tag{48}$$

$$\Box(\neg\, \mathrm{IM}_k \vee \neg\, \mathrm{IN}_k) \tag{49}$$

Finally, we add further formulae connecting the different transition systems:

$$\Box(\neg\,(\mathrm{Gb}_{ji} \wedge \mathrm{Gb}_{js})) \quad \forall j \in [|1;\mathrm{r}|], \forall i, s \in [|1;\mathrm{f}|], i \neq s \tag{50}$$

$$\Box((\mathrm{G}_j \vee \mathrm{D}_j \vee \mathrm{Mf}_j) \Leftrightarrow (\mathrm{Gb}_{j1} \vee \ldots \vee \mathrm{Gb}_{jf}) \quad \forall j \in [|1;\mathrm{r}|] \tag{51}$$

The first of these, equation (50), disallows a robot being able to grab more than one piece of food at any moment. The second, equation (51), links the robot transition systems with the transition systems for food. In particular, robot j can be in one of the states Grab, Movetohome or Deposit if and only if some item of food is being grabbed by robot j.

4.2 Analysis using TRP++

Next we provide results of what we can prove with TRP++, and how long the proofs will take. We would like to see whether our specification is detailed enough to deal (in principle) with an arbitrary number of robots, and so we will see how TRP ++ deals with increasing numbers of robots and food. The following corresponds to different cases: first, varying the number of robots and pieces of food in Sections 4.2.1-4.2.3 and, in Section 4.2.4, regenerating any piece of food as soon as it has been brought to the nest (without waiting for the other pieces of food to arrive).

4.2.1 One robot, two pieces of food – Table II. First we consider one robot and two pieces of food. In this case, we want to know if the robot will grab every piece of food. As we can see in Table II, with the help of TRP++, the formula $\Box \neg \mathrm{D}_1$ is not satisfiable, so the robot will at least grab one piece of food in order to be able to reach the state D_1 (since we need to go through the state G_1, and it implies the grabbing of one piece of food). Then the tests for In Nest and In Arena show that each piece of food cannot have an infinite path in its transition system without reaching those states (for example the formula $\Box \neg \mathrm{IN}_1$ is not satisfiable). Finally, we show that infinitely often the robot grabs the first piece of food, infinitely often the robot grabs the second piece of food, (by showing the negation of infinitely often Gb_{11} and infinitely often Gb_{12} is unsatisfiable) and infinitely often the robot grabs each piece of food.

4.2.2 Two robots, two pieces of food – Table III. In this case we add a further robot (Table III). The conclusions for the food items are the same: each piece of food will be grabbed infinitely often. Furthermore, the results for the robots are also interesting. Since we have already proved that any robot is forced to reach the state G infinitely often, then we can easily conclude that the robots will grab a piece of food. The tests $\Box(\neg\,\mathrm{Gb}_{11} \wedge \neg\,\mathrm{Gb}_{12})$ ("it is always the case that robot 1 doesn't grab food 1 and doesn't grab food2") and $\Box(\neg\,\mathrm{Gb}_{21} \wedge \neg\,\mathrm{Gb}_{22})$ and their unsatisfiability show this. Then the tests $\Box(\neg\,\mathrm{Gb}_{11} \wedge \neg\,\mathrm{Gb}_{21})$ ("it is always the case that food 1 is not grabbed by any robot") and $\Box(\neg\,\mathrm{Gb}_{12} \wedge \neg\,\mathrm{Gb}_{22})$ and their unsatisfiability prove that each piece of food is grabbed at some point. We also show that both robot 1 grabs food infinitely often and robot 2 grabs food infinitely often.

Comparing these results with those in Table II, we see an increase in the time required for the automated proofs. Thus, the complexity of checking such properties is very dependent on the number of robots and pieces of food.

4.2.3 Two robots, three pieces of food – Table IV. Finally, we now add a third piece of food. The results of analysing properties are the same as in the previous section:

Property	Meaning	Satisfiable?	Time (s)
$\Box \neg D_1$	The robot never deposits food	No	0.039
$\Box\Diamond D_1$	The robot can infinitely often deposit food	Yes	0.056
$\Box\Diamond IN_1$	Food 1 can be infinitely often in the nest	Yes	0.082
$\Box\Diamond IN_2$	Food 2 can be infinitely often in the nest	Yes	0.133
$\Box \neg IN_1$	Food 1 is never in the nest	No	0.170
$\Box \neg IN_2$	Food 2 is never in the nest	No	0.179
$\Box\Diamond IM_1$	Food 1 can be infinitely often in the arena	Yes	0.143
$\Box\Diamond IM_2$	Food 2 can be infinitely often in the arena	Yes	0.132
$\Box \neg IM_1$	Food 1 is never in the arena	No	0.98
$\Box \neg IM_2$	Food 2 is never in the arena	No	0.103
$\Box\Diamond Gb_{11}$	Food 1 can be infinitely often grabbed	Yes	0.130
$\Box\Diamond Gb_{12}$	Food 2 can be infinitely often grabbed	Yes	0.110
$\Box \neg Gb_{11}$	Food 1 is never grabbed	No	0.368
$\Box \neg Gb_{12}$	Food 2 is never grabbed	No	0.385
$\neg \Box\Diamond Gb_{11}$	It is not the case that food 1 is grabbed infinitely often	No	0.757
$\neg \Box\Diamond Gb_{12}$	It is not the case that food 2 is grabbed infinitely often	No	0.853
$\neg (\Box\Diamond Gb_{11} \wedge \Box\Diamond Gb_{12})$	Its not the case that the robot grabs each piece of food infinitely often	No	1.754

Table II.
Applying TRP++ to the "one robot, two pieces of food" model

each piece of food will be grabbed infinitely often and each robot will grab food infinitely often.

In Table IV, different tests were made to model various situations. Within the results, the "$\varnothing$" sign means that the analysis takes more than 10 min. As can be seen, in certain cases the prover takes more than 10 min to finish the proof. This is because increasing the number or robots and food items means that the underlying transition system representing the combination (product) of the robot and food transition systems increases rapidly in size. In Table V, the number of propositions required for each of the problems discussed and the maximum number of states in the product of the transition system. However, it should be noted that not all these states are legal because of the conditions imposed between transition systems.

4.2.4 One robot, two pieces of food, food is directly regenerated – Table VI. We conclude this section with a final interesting case – the case where food is automatically regenerated in the arena as soon as it has been brought in the nest (i.e. the system does not wait until all the food is found and brought to the nest before regenerating a piece of food) (Table VI). As we could foresee, the scenario where only one piece of food is brought in the nest (infinitely many times) becomes possible, and so one piece of food can stay infinitely in the arena without ever being grabbed.

4.2.5 Comment. As we can see from Table IV, it takes a long time to prove some of these properties, beyond the 10 min threshold is certain cases.

In Section 6, we consider using a more powerful approach to tackling larger numbers of robots or items of food. Before we move on, there are several points to note about this:

Property	Meaning	Satisfiable?	Time (s)
$\Box \neg D_1$	Robot 1 never deposits food	No	0.325
$\Box \neg D_2$	Robot 2 never deposits food	No	5.336
$\Box\Diamond D_1$	Robot 1 infinitely often deposits food	Yes	1.944
$\Box\Diamond D_2$	Robot 2 infinitely often deposits food	Yes	1.985
$\Box \neg IN_1$	Food 1 is never in the nest	No	0.990
$\Box \neg IN_2$	Food 2 is never in the nest	No	1.058
$\Box\Diamond IN_1$	Food 1 is infinitely often in the nest	Yes	6.158
$\Box\Diamond IN_2$	Food 2 is infinitely often in the nest	Yes	15.016
$\Box \neg IM_1$	Food 1 is never in the arena	No	5.342
$\Box \neg IM_2$	Food 2 is never in the arena	No	7.103
$\Box\Diamond IM_1$	Food 1 is infinitely often in the arena	Yes	21.337
$\Box\Diamond IM_2$	Food 2 is infinitely often in the arena	Yes	10.208
$\Box \neg Gb_{11}$	Robot 1 never grabs food 1	Yes	1.495
$\Box \neg Gb_{12}$	Robot 1 never grabs food 2	Yes	2.714
$\Box \neg Gb_{21}$	Robot 2 never grabs food 1	Yes	0.882
$\Box \neg Gb_{22}$	Robot 2 never grabs food 2	Yes	2.668
$\Box(\neg Gb_{11} \wedge \neg Gb_{12})$	Robot 1 never grabs anything	No	0.045
$\Box(\neg Gb_{21} \wedge \neg Gb_{22})$	Robot 2 never grabs anything	No	0.288
$\Box(\neg Gb_{11} \wedge \neg Gb_{21})$	Food 1 is never grabbed	No	1.760
$\Box(\neg Gb_{12} \wedge \neg Gb_{22})$	Food 2 is never grabbed	No	1.796
$\neg \Box\Diamond(Gb_{11} \vee Gb_{21})$	It is not the case that food 1 is grabbed infinitely often	No	12.891
$\neg \Box\Diamond(Gb_{12} \vee Gb_{22})$	It is not the case that food 2 is grabbed infinitely often	No	13.514
$\neg \Box\Diamond(Gb_{11} \vee Gb_{12})$	It is not the case that robot 1 grabs food infinitely often	No	4.026
$\neg \Box\Diamond(Gb_{21} \vee Gb_{22})$	It is not the case that robot 2 grabs food infinitely often	No	10.773
$\neg (\Box\Diamond(Gb_{11} \vee Gb_{21}) \wedge \Box\Diamond(Gb_{12} \vee Gb_{22}))$	It is not the case that both food 1 is grabbed infinitely often and food 2 is grabbed infinitely often	No	24.640

Table III.
Applying TRP++ to the "two robots, two pieces of food" model

- TRP++ is just a prototype and more sophisticated and efficient temporal resolution provers will appear in the future – thus larger numbers of robots/items will be able to be practically verified.
- We are currently investigating deductive calculi which allow the input of constraints (such as the robot is in exactly one of states/phases at any moment) which provide improved complexity results (Dixon *et al.*, 2007a, b) and would be applicable here.
- The most expensive part of the temporal resolution calculus is the part which deals with formulae of the form $\Diamond\varphi$. We have introduced many of these formulae to represent some of the constraints about the system, namely equations (20)-(22) and (26)-(29). We may be able to complete the proofs in less time and even allow proofs of greater numbers of robots and pieces food by using a smaller number of alternative constraints for example some of those discussed in Section 2.2.4.
- We have only intended this automated proof analysis as a way to explore whether the models we have are realistic and to verify interesting properties of

Property	Meaning	Satisfiable?	Time (s)
$\Box \neg D_1$	Robot 1 never deposits food	No	0.835
$\Box \neg D_2$	Robot 2 never deposits food	No	83.834
$\Box\Diamond D_1$	Robot 1 deposits food infinitely often	Yes	13.793
$\Box\Diamond D_2$	Robot 2 deposits food infinitely often	Yes	15.136
$\Box \neg IN_1$	Food 1 is never in the nest	No	11.932
$\Box \neg IN_2$	Food 2 is never in the nest	No	12.530
$\Box \neg IN_3$	Food 3 is never in the nest	No	11.440
$\Box\Diamond IN_1$	Food 1 is infinitely often in the nest	Yes	23.988
$\Box\Diamond IN_2$	Food 2 is infinitely often in the nest	Yes	79.434
$\Box\Diamond IN_3$	Food 3 is infinitely often in the nest	Yes	72.742
$\Box \neg IM_1$	Food 1 is never in the arena	No	286.533
$\Box \neg IM_2$	Food 2 is never in the arena	No	393.277
$\Box \neg IM_3$	Food 3 is never in the arena	No	358.187
$\Box\Diamond IM_1$	Food 1 is infinitely often in the arena	Yes	$\varnothing$
$\Box\Diamond IM_2$	Food 2 is infinitely often in the arena	Yes	230.038
$\Box\Diamond IM_3$	Food 3 is infinitely often in the arena	Yes	256.435
$\Box \neg Gb_{11}$	Robot 1 never grabs food 1	Yes	11.440
$\Box \neg Gb_{12}$	Robot 1 never grabs food 2	Yes	25.364
$\Box \neg Gb_{13}$	Robot 1 never grabs food 3	Yes	23.089
$\Box \neg Gb_{21}$	Robot 2 never grabs food 1	Yes	13.084
$\Box \neg Gb_{22}$	Robot 2 never grabs food 2	Yes	40.984
$\Box \neg Gb_{23}$	Robot 2 never grabs food 3	Yes	34.564
$\Box(\neg Gb_{11} \wedge \neg Gb_{12} \wedge \neg Gb_{13})$	Robot 1 never grabs anything	No	0.057
$\Box(\neg Gb_{21} \wedge \neg Gb_{22} \wedge \neg Gb_{23})$	Robot 2 never grabs anything	No	0.398
$\Box(\neg Gb_{11} \wedge \neg Gb_{21})$	Food 1 is never grabbed	No	39.090
$\Box(\neg Gb_{12} \wedge \neg Gb_{22})$	Food 2 is never grabbed	No	42.452
$\Box(\neg Gb_{13} \wedge \neg Gb_{23})$	Food 3 is never grabbed	No	42.160
$\neg \Box\Diamond(Gb_{11} \vee Gb_{21})$	It is not the case that food 1 is grabbed infinitely often	No	198.809
$\neg \Box\Diamond(Gb_{12} \vee Gb_{22})$	It is not the case that food 2 is grabbed infinitely often	No	201.716
$\neg \Box\Diamond(Gb_{13} \vee Gb_{23})$	It is not the case that food 3 is grabbed infinitely often	No	205.435
$\neg \Box\Diamond(Gb_{11} \vee Gb_{12} \vee Gb_{13})$	It is not the case that robot 1 grabs food infinitely often	No	23.421
$\neg \Box\Diamond(Gb_{21} \vee Gb_{22} \vee Gb_{23})$	It is not the case that robot 2 grabs food infinitely often	No	69.784
$\neg (\Box\Diamond(Gb_{11} \vee Gb_{21}) \wedge \Box\Diamond(Gb_{12} \vee Gb_{22}) \wedge \Box\Diamond(Gb_{13} \vee Gb_{23}))$	It is not the case that each piece of food is grabbed infinitely often	No	$\varnothing$

Table IV.
Applying TRP++ to the "two robots, three pieces of food" model

relatively small numbers of robots/items – to tackle arbitrarily sized swarms, we must turn to first-order temporal representation (Section 6).

In the next section, however, we increase the sophistication of the basic model.

5. Introduction of time

The next natural step in our formalisation is to introduce the notion of explicit time. Indeed, it would be very useful to be able to represent some kind of counter which would bound the number of movements used by a robot to search, grab and then to bring a piece of food to the nest. For that, we will proceed as we have done for introducing the pieces of food – we will create two transition systems and we will link them to the transition system for the robot.

5.1 The transition systems

First we have to define a strategy to represent our movement/time counters. We chose two transition systems: the first one being a general counter which counts all the movements of the robot; the other one being a more specific counter, that counts (in a loop) the movement done for searching and then (in a second loop linked to the first one) the movements carried out in order to bring the found food to the nest. The corresponding graphs are shown in Figure 4.

Here are some explanations about the transition systems in Figure 4. First, there are four types of states:

- "dead" which is a "shaft-state", i.e. a robot in this state is lost – it has no more energy to move in the arena;
- en_i means that there remains i movements to the robot before it has no more energy – it bounds the total number of movements by en_{max} (here, en_{max} is 20); and
- in the same way, st_i and ht_i represent the searching and the homing time, i.e. they, respectively, bound the time a robot has to search and to bring back a piece

Table V.
Size of combined transition system and number of propositions

Number of robots	Number of pieces of food	Number of propositions	Maximum states
1	0	13	9
1	2	19	81
2	2	34	1,296
2	3	38	5,184

Table VI.
Applying TRP++ to the "one robot, two pieces of food", regeneration model

Property	Meaning	Satisfiable?	Time (s)
$\Box(\neg Gb_1 \wedge \neg Gb_2)$	No food is grabbed	No	0.019
$\Box(\neg Gb_1)$	Food 1 is never grabbed	Yes	0.035
$\Box(\neg Gb_2)$	Food 2 is never grabbed	Yes	0.031
$\Box(\Diamond Gb_1 \wedge \Diamond Gb_2)$	Foods 1 and 2 are infinitely grabbed	Yes	0.098

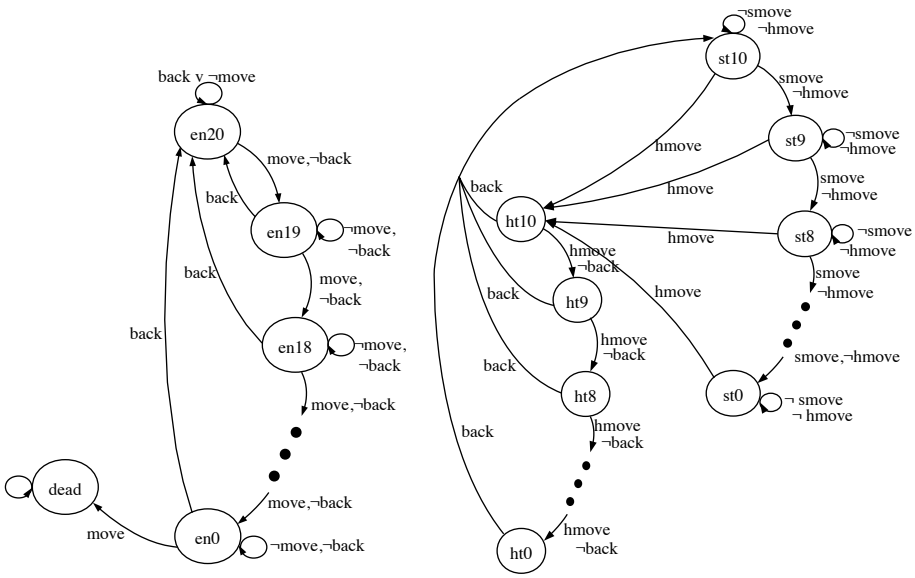


Figure 4.
Transitions systems relating to energy and time

of food to the nest. As for the total energy, the maximum searching time is st_{max} (here 10) and the maximum homing time is ht_{max} (also 10).

Similarly, there are two types of transitions:

(1) The transitions "move", "smove" and "hmove" correspond to a movement carried out by the robot in the arena. This can be a searching movement (corresponding to the states L, Ra, Mf) for "smove", or homing movement (corresponding to the states H, G, Mh) for "hmove". Finally, "move" refers to either a homing move or a searching move. Remaining in the state S does not require movement to another place in the arena so we assume that in this state the robot consumes no energy. However, alternatively, if we require that scanning the arena does consume energy it is easy to make scanning the arena a searching movement smove similar to L, Ra and Mf.

(2) The transition "back" corresponds to the states Re and D and means that the robot is either going back to the nest, or is already in it.

The first transition system in Figure 4. This represents general movement. At the beginning, the energy of the robot is maximal and so the robot is in state en_{max}. Every time the robot moves, the energy decreases and it moves to the successive energy state. If the robot is in the state S (scanning) then there is no move – represented by a transition back to the same energy state. Finally, if the robot reaches the state Re or D, then it is back at the nest, and so its energy goes back up to en_{max}. If it reaches the state "dead" (i.e. it has no more energy and is not yet back at the nest) then the robot is lost.

The second transition system in Figure 4. This represents specific movement (concerning searching and homing) with two counters. At the beginning, the robot is in the state st_{max}. If it makes a searching move (to the states L, Ra and Mf) then it takes an

"smove" transition (i.e. the searching time decreases). If it is in state S, then it makes no movement. If it reaches the states H, G or Mh, then it takes the "hmove" transition and begins to deal with the homing process, beginning in ht_{max}. From any one of those states, it can move back to the nest and so would revert to the initial state st_{max}. The state "dead" has exactly the same function as in the first transition system, except for the fact that it is now reached from searching or homing.

Now that we have defined two counters for the time, we only need to link this with the transition system for the robot, in order to assess its movements.

5.2 Linking robot transition systems with the new transition systems

The best way to link the robot and the counters is to define, as we explained (intuitively) above which proposition is true in each state. For example, in state Ra, the proposition smove is true (since the robot moves to search for a piece of food) but the proposition hmove is false (it is not homing) and vice versa. These links can be given directly as follows, where each formula is in the scope of a $\Box$ operator (as they hold everywhere):

$$L \Rightarrow (\text{smove} \wedge \neg \text{back}) \quad \text{(The robot is searching)} \tag{52}$$

$$Ra \Rightarrow (\text{smove} \wedge \neg \text{back}) \quad \text{(The robot is searching)} \tag{53}$$

$$Mf \Rightarrow (\text{smove} \wedge \neg \text{back}) \quad \text{(The robot is searching)} \tag{54}$$

$$S \Rightarrow (\neg \text{smove} \wedge \neg \text{hmove} \wedge \neg \text{back}) \quad \text{(The robot is scanning)} \tag{55}$$

$$G \Rightarrow (\text{hmove} \wedge \neg \text{back}) \quad \text{(The robot is homing)} \tag{56}$$

$$H \Rightarrow (\text{hmove} \wedge \neg \text{back}) \quad \text{(The robot is homing)} \tag{57}$$

$$Mh \Rightarrow (\text{hmove} \wedge \neg \text{back}) \quad \text{(The robot is homing)} \tag{58}$$

$$D \Rightarrow (\text{back} \wedge \neg \text{move}) \quad \text{(The robot is in the nest)} \tag{59}$$

$$Re \Rightarrow (\text{back} \wedge \neg \text{move}) \quad \text{(The robot is in the nest)} \tag{60}$$

We complete the specification by writing some simple logical laws capturing movement:

$$\Box \neg (\text{smove} \wedge \text{hmove}) \tag{61}$$

$$\Box((\text{smove} \vee \text{hmove}) \Leftrightarrow \text{move}) \tag{62}$$

Now each state of the robot defines a behaviour in the new graphs, since it covers every transition.

5.3 Simplifying the system

As we have explicitly encoded the time spent searching and homing within the transition systems in Figure 4 we do not now require the proposition Ps, which was used in the original specification to capture the fact that the searching time is over. Further, we remove the propositions Pr and Pt and provide alternative constraints to avoid the infinite loops discussed in Section 2.2. Here, is the new set of formulae representing the transition system, where each formula is in the scope of a $\Box$ operator (as they hold everywhere):

$$\mathrm{Re} \Rightarrow \bigcirc(\mathrm{Re} \vee \mathrm{L}) \qquad \mathrm{H} \Rightarrow \bigcirc(\mathrm{H} \vee \mathrm{Re})$$

$$\mathrm{L} \Rightarrow \bigcirc\mathrm{Ra} \qquad \mathrm{G} \Rightarrow \bigcirc\mathrm{Mh}$$

$$\mathrm{Ra} \Rightarrow \bigcirc(\mathrm{Ra} \vee \mathrm{Mf} \vee \mathrm{H}) \qquad \mathrm{Mh} \Rightarrow \bigcirc(\mathrm{Mh} \vee \mathrm{D})$$

$$\mathrm{Mf} \Rightarrow \bigcirc(\mathrm{Mf} \vee \mathrm{S} \vee \mathrm{G} \vee \mathrm{H}) \qquad \mathrm{D} \Rightarrow \bigcirc\mathrm{Re}$$

$$\mathrm{S} \Rightarrow \bigcirc(\mathrm{S} \vee \mathrm{Mf} \vee \mathrm{H} \vee \mathrm{Ra})$$

As we can see, these formulae represent a simple transition system with no labels on the transitions. We again need the structural constraints (18) and (19). It is clear that we again need some additional constraints in order to avoid undesirable infinite loops discussed in Section 2.2. These constraints are:

- $\mathrm{Re} \Rightarrow \Diamond \neg \mathrm{Re}$ – this constraint replaces equation (20) $\Box\Diamond\mathrm{Pr}$ ensuring that a robot cannot stay infinitely in the nest.
- $\mathrm{S} \Rightarrow \Diamond \neg \mathrm{S}$ – this constraint replaces equation (22) $\Box\Diamond\mathrm{Pt}$ ensuring that a robot cannot scan the arena infinitely.
- $\mathrm{Ra} \Rightarrow \Diamond\mathrm{Mf}$ – this constraint replaces the constraint (28) ensuring that, if a robot makes an infinite number of random moves then it will find food sometime in the future and so it will reach the state Mf.
- $\mathrm{Mf} \Rightarrow \Diamond\mathrm{G}$ – a repeat of constraint (29).

Apart for the structural constraints (18) and (19) we now only have four additional constraints instead of the ten constraints (20)-(29) provided earlier in this paper. Further, we do not need the propositions Ps, Pt and Pr representing the fact that the searching, scanning and resting times are over.

5.4 Some results with TRP++

Adding the notion of time, via the new formulae, makes analysis of the overall set of formulae quite complex. Consequently, we have carried out an analysis of just the simplest variety, i.e. with only one robot and ignoring the number and position of pieces of food (Table VII).

An interesting result concerns the minimum time necessary to make a loop, i.e. to go on the arena, grab some food and bring it back to the nest. In our formalisation (where $\mathrm{st}_{max} = 5$, $\mathrm{ht}_{max} = 4$ and $\mathrm{en}_{max} = 10$), the minimal number of steps to get a piece of food is 3 since $\Box \neg st2$ is unsatisfiable whereas $\Box \neg st1$ is satisfiable. The minimal number of steps to bring food to the nest is 2 since $\Box \neg ht3$ is unsatisfiable whereas $\Box \neg ht2$ is satisfiable (we have to add 1 to the result because of the transition between the searching part of the second graph and its homing part is not recorded as a movement though it is one). Testing the first graph leads us to conclude that the general minimal number of steps to get and bring the piece of food is 5 since $\Box \neg en5$ is unsatisfiable whereas $\Box \neg en4$ is satisfiable. Calculating the sum: "$2 + 3 = 5$" we get the required result.

These results can be related to the transition system. A robot needs a minimum of three steps between the states Re and Ra (so three steps to get a piece of food) and it needs to take two transitions from the state G to the state D (so two steps to bring it to the nest). Therefore, these results are as expected. In conclusion, the inequality:

Property	Meaning	Satisfiable?	Time (s)
$\Box \neg$ D	Robot never deposits food	No	4.632
$\Box\Diamond$D	Robot can infinitely deposit food	Yes	0.233
$\Box\Diamond st0$	Robot can grab food using st_{max} steps	Yes	1.392
$\Box \neg st0$	Robot always grabs food using $< st_{max}$ steps	Yes	0.175
$\Box \neg st1$	Robot always grabs food using $< st_{max} - 1$ step	Yes	0.196
$\Box \neg st2$	Robot always grabs food using $< st_{max} - 2$ steps	No	0.843
$\Box \neg ht0$	Robot always grabs food using $< ht_{max}$ steps	Yes	0.196
$\Box \neg ht1$	Robot always brings food using $< ht_{max} - 1$ step	Yes	0.199
$\Box \neg ht2$	Robot always brings food using $< ht_{max} - 2$ steps	Yes	0.194
$\Box \neg ht3$	Robot always brings food using $< ht_{max} - 3$ steps	No	0.798
$\Box \neg en4$	Robot always grabs and brings food using $< en_{max} - 4$ steps	Yes	0.305
$\Box \neg en5$	Robot always grabs and brings food using $< enmax - 5$ steps	No	4.810
Addition of two pieces of food			
$\Box \neg$ D	Robot never deposits food	No	0.397
$\Box \neg Gb_1$	Food 1 is never grabbed	No	6.329
$\Box \neg Gb_2$	Food 2 is never grabbed	No	7.821
$\Box \neg en4$	Robot always grabs and brings food using $< en_{max} - 4$ steps	Yes	1.501
$\Box \neg en5$	Robot always grabs and brings food using $< en_{max} - 5$ steps	No	37.883

Table VII.
Adding time and movement – some TRP++ results

$$ht_{max} + st_{max} \leq en_{max}$$

which intuitively captures the energy requirements in order to find and retrieve food, is now confirmed via TRP++.

6. Arbitrary sized swarms and FOTL

As we have seen, fixed numbers of finite-state transitions systems are a convenient way to represent small, fixed numbers of robots and/or items of food. We have verified the properties of such swarms by formalising the transition systems as propositional temporal formulae and utilising the TRP + + clausal resolution prover. However, we have also seen that this approach is not feasible for larger swarms. So, what are we to do if we have 100 robots? Or 1,000? Or we have arbitrarily many?

In the latter case, this problem corresponds to that of formalising infinite-state systems, particularly where there are arbitrarily many finite-state components, each with similar behaviour. While such problems are difficult to verify either using propositional methods or using model checking, it has been shown that FOTL is a strong and viable formalism for describing and analysing such systems (Fisher *et al.*, 2006). The idea here is to have a single transition system to represent robot behaviour (as earlier), but then to parameterise this with the particular robot under consideration. Thus, while we might have states such as D and Re, these are translated into unary predicates within our temporal representation. Similarly Robot(X) denotes that X is a robot. The single argument to each such predicate represents the particular robot. Thus, D($R1$) is true if robot $R1$ is in state D, D($R2$) is true if robot $R2$ is in state D, and so on. This addition of a parameter then allows us to reason about full, unbounded sets.

For example, we might wish to say that "at least one robot is in state D". This would be represented by $\exists X.\text{Robot}(X) \wedge \text{D}(X)$.

We note from the above that we ideally would like a first-order temporal language, which at least has unary predicates. Fortunately, this is exactly the language utilised in Fisher *et al.* (2006) and is also a variety extensively studied and developed over the last decade (Hodkinson *et al.*, 2000). In particular, this logic is a fragment of FOTL termed monodic FOTL which is expressive yet in many cases decidable (Wolter and Zakharyaschev, 2002). We will define the temporal basis below.

6.1 First-order temporal logic

Syntax. Let:

- $VAR = \{\text{x}, \text{y}, \ldots\}$ be an infinite countable set of variables.
- $CONS = \{\text{c}_0, \text{c}_1, \ldots\}$ be a countable supply of constants symbols, possibly empty.
- $\forall n \in \mathbb{N}, PRED^n = \{P_0^n, P_1^n, P_2^n, \ldots\}$ be an infinite supply of n-ary predicates symbols also called relation symbols of arity n.
- $\forall n \in \mathbb{N}, FUN^n = \{f_0^n, f_1^n, f_2^n, \ldots\}$ be a countable set of n-ary functions symbols, possibly empty.

The set *TERM* of terms is the smallest set defined by the following rules:

- $\forall x \in VAR, x \in TERM$.
- $\forall c \in CONS, c \in TERM$.
- $\forall t_1, t_2, \ldots, t_n \in TERM, \forall f \in FUN^n, f(t_1, \ldots, t_n) \in TERM$.

Now we can give the syntax of FOTL. The set *FORM* of the formulae of FOTL is the smallest set defined by the following rules:

- If $P \in PROP^n$, and $\forall t_1, t_2, \ldots, t_n \in TERM, P(t_1, \ldots, t_n) \in FORM$.
- If $A, B \in FORM$, then $\neg A, A \vee B, A \wedge B, \bigcirc A, \Box A, \Diamond A, \exists xA, \forall xA \in FORM$.

Semantics. We now consider interpretations for statements in the above logic. For our particular purposes we will not use any function symbols. However, we will use a restricted variety of the full FOTL called the monodic fragment.

Definition 1. Monodicity (Wolter and Zakharyaschev, 2002). A formula ϕ in a FOTL language without equality and function symbols (constants are allowed) is called monodic if any subformula of ϕ of the form $\bigcirc\psi, \Box\psi, \Diamond\psi$ contains at most one free variable.

To ascribe meaning to all sentences of a first-order language, the following information (called a realisation) is needed:

- A domain of discourse D, usually required to be non-empty.
- For every constant symbol an element of D as its interpretation.
- For every n-ary predicate symbol an n-ary relation on D, i.e. a subset of D^n, as its interpretation, which is equivalent to a function $D^n \rightarrow \{True, False\}$.

Let M be a realisation of L a FOTL language. An interpretation I for M and L is a function $VAR \rightarrow D$.

Now we can define the semantics of the language in a way similar to that given to LTL earlier. Thus, the semantics relates a realisation, M, a temporal sequence, σ and a moment in time, i, to each formula in the logic. It does this through the satisfaction relation "$\models$" and the specific satisfaction relation corresponding to an interpretation, "$\models_I$":

$$M, \sigma, i \models_I P(t_1, \ldots, t_n) \quad iff \quad (t_1, \ldots, t_n) \in P$$
$$\text{the usual semantics for } \neg, \wedge \text{ and } \vee \ldots$$
$$M, \sigma, i \models_I \exists x\phi \quad iff \quad \exists c \in D.M, \sigma, j \models_I \phi[x/c]$$
$$M, \sigma, i \models_I \forall x\phi \quad iff \quad \forall c \in D.M, \sigma, j \models_I \phi[x/c]$$
$$M, \sigma, i \models_I \bigcirc\phi \quad iff \quad M, \sigma, i+1 \models_I \phi$$
$$M, \sigma, i \models_I \Diamond\phi \quad iff \quad \exists j \geq i.M, \sigma, j \models_I \phi$$
$$M, \sigma, i \models_I \Box\phi \quad iff \quad \forall j \geq i.M, \sigma, j \models_I \phi$$

The notion of satisfiability is the same as for LTL.

6.2 Robotic swarms and FOTL

We can now extend the work described in Sections 2 and 3, essentially using the same model but translating to FOTL, rather than LTL. Using the same model will avoid us again having to describe behaviours relating to robots, etc.

The principle of this extension is simple. Every state of each transition system is translated into FOTL as a unary (or monadic) predicate. We also define the additional predicate Robot which will be used to declare any robot. We will simply re-write the logical representation of the transition system for LTL, parameterising it with respect to the robot considered.

Thus, we begin with the FOTL description of robot behaviours where the formulae below are in the scope of a $\Box$ operator:

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Re}(X) \wedge \text{Pr}(X)) \Rightarrow \bigcirc \text{L}(X) \tag{63}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Re}(X) \wedge \neg \text{Pr}(X)) \Rightarrow \bigcirc \text{Re}(X) \tag{64}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{L}(X)) \Rightarrow \bigcirc \text{Ra}(X) \tag{65}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Ra}(X) \wedge \neg \text{Ps}(X) \wedge \neg \text{Ff}(X)) \Rightarrow \bigcirc \text{Ra}(X) \tag{66}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Ra}(X) \wedge \text{Ps}(X)) \Rightarrow \bigcirc \text{H}(X) \tag{67}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Ra}(X) \wedge \neg \text{Ps}(X) \wedge \text{Ff}(X)) \Rightarrow \bigcirc \text{Mf}(X) \tag{68}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Mf}(X) \wedge \neg \text{Ps}(X) \wedge \text{Ff}(X)) \Rightarrow \bigcirc (\text{Mf}(X) \vee \text{G}(X)) \tag{69}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Mf}(X) \wedge \neg \text{Ps}(X) \wedge \neg \text{Ff}(X)) \Rightarrow \bigcirc \text{S}(X) \tag{70}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Mf}(X) \wedge \text{Ps}(X)) \Rightarrow \bigcirc \text{H}(X) \tag{71}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{S}(X) \wedge \text{Ps}(X)) \Rightarrow \bigcirc \text{H}(X) \tag{72}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{S}(X) \wedge \neg \text{Ps}(X) \wedge \text{Ff}(X)) \Rightarrow \bigcirc \text{Mf}(X) \tag{73}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{S}(X) \wedge \neg \text{Ps}(X) \wedge \neg \text{Pt}((X) \wedge \neg \text{Ff}(X)) \Rightarrow \bigcirc \text{S}(X) \tag{74}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{S}(X) \wedge \neg \text{Ps}(X) \wedge \text{Pt}((X) \wedge \neg \text{Ff}(X)) \Rightarrow \bigcirc \text{Ra}(X) \tag{75}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{H}(X)) \Rightarrow \bigcirc(\text{H}(X) \vee \text{Re}(X)) \tag{76}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{G}(X)) \Rightarrow \bigcirc\text{Mh}(X) \tag{77}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{Mh}(X)) \Rightarrow \bigcirc(\text{Mh}(X) \vee \text{D}(X)) \tag{78}$$

$$\forall X \cdot (\text{Robot}(X) \wedge \text{D}(X)) \Rightarrow \bigcirc\text{Re}(X) \tag{79}$$

As previously we need to specify that a robot can be in only one state at a time, e.g.:

$$\begin{aligned} \forall X.\text{Robot}(X) \Rightarrow (&\text{Re}(X) \vee \text{L}(X) \vee \text{Ra}(X) \vee \text{Mf}(X) \vee \text{S}(X) \vee \text{H}(X) \\ &\vee \text{G}(X) \vee \text{Mh}(X) \vee \text{D}(X)) \end{aligned} \tag{80}$$

and for each pair of states (e.g. for Re and L):

$$\forall X.\text{Robot}(X) \Rightarrow (\neg\, \text{Re}(X) \vee \neg\, \text{L}(X)) \tag{81}$$

Further we can translate the constraints from Section 2.2 in a similar way. We note that equations (63)-(81) and the first-order translations of equations (20)-(29) only involve unary or monadic predicates so these formulae fall within the decidable, monodic monadic fragment of FOTL.

If we try to follow the same route for the food transition systems using a unary predicate Food(X) representing X is a piece of food and a binary predicate "Gb(Y, Z)" representing robot Y grabs food Z we obtain the following:

$$\Box\forall X \cdot (\text{Food}(X) \wedge \text{IM}(X)) \Rightarrow \bigcirc(\text{IM}(X) \vee (\exists Z \cdot (\text{Robot}(Z) \wedge \text{Gb}(Z, X)))) \tag{82}$$

$$\Box\forall X, \forall Y \cdot (\text{Food}(X) \wedge \text{Robot}(Y) \wedge \text{Gb}(Y, X) \wedge \neg\, \text{D}(Y)) \Rightarrow \bigcirc\text{Gb}(Y, X) \tag{83}$$

$$\Box\forall X, \forall Y \cdot (\text{Food}(X) \wedge \text{Robot}(Y) \wedge \text{Gb}(Y, X) \wedge \text{D}(Y)) \Rightarrow \bigcirc\text{IN}(X) \tag{84}$$

$$\Box\forall X \cdot (\text{Food}(X) \wedge \text{IN}(X) \wedge (\exists Y \cdot (\text{Food}(Y) \wedge \neg\, \text{IN}(Y)))) \Rightarrow \bigcirc\text{IN}(X) \tag{85}$$

$$\Box\forall X \cdot (\text{Food}(X) \wedge (\forall Y \cdot (\text{Food}(Y) \wedge \text{IN}(Y))) \Rightarrow \bigcirc\text{IM}(X) \tag{86}$$

In the above equation (82) represents the transitions out of the state In Arena, equations (83) and (84) represent transitions out of each state Grabbed and equations (85) and (86) represent the transition out of the state In Nest.

However, the clause (83) is not monodic. To address this we use multiple copies of the food transition system, similar to what we did for the propositional case, where $\text{Gb}_i(Z)$ is a unary predicate representing that piece of food i is grabbed by robot X. The following represent formulae for the ith piece of food:

$$\Box\text{IM}_i \Rightarrow \bigcirc(\text{IM}_i \vee (\exists Z \cdot (\text{Robot}(Z) \wedge (\text{Gb}_i(Z))))) \tag{87}$$

$$\Box\forall Y \cdot (\text{Robot}(Y) \wedge \text{Gb}_i(Y) \wedge \neg\, \text{D}(Y)) \Rightarrow \bigcirc\text{Gb}_i(Y) \tag{88}$$

$$\Box\forall Y \cdot (\text{Robot}(Y) \wedge \text{Gb}_i(Y) \wedge \text{D}(Y)) \Rightarrow \bigcirc\text{IN}_i \tag{89}$$

$$\Box\text{IN}_i \wedge \left(\bigvee_j \neg\, \text{IN}_j\right) \Rightarrow \bigcirc\text{IN}_i \tag{90}$$

$$\Box\left(\bigwedge_j \mathrm{IN}_j\right) \Rightarrow \bigcirc \mathrm{IM}_i \tag{91}$$

Again we need to specify that a piece of food can be in exactly one of the food states at any moment. As we do not allow equality we must introduce a binary predicate $NE(X, Y)$ to denote that X and Y are not the same and explicitly state for every pair of robots introduced that they are not the same using this. Now to represent that a piece of food i can only be in one of the food states we have the following:

$$\Box\left(\forall X.\mathrm{Robot}(X) \Rightarrow \left(\mathrm{IN}_i \vee \mathrm{IM}_i \vee \bigvee_i \mathrm{Gb}_i(X)\right)\right) \tag{92}$$

$$\Box(\forall X, Y \cdot (\mathrm{Robot}(X) \wedge \mathrm{Robot}(Y) \wedge NE(X, Y)) \Rightarrow (\neg\, \mathrm{Gb}_i(X) \vee \neg\, \mathrm{Gb}_i(Y))) \tag{93}$$

$$\Box\forall X.\mathrm{Robot}(X) \Rightarrow (\neg\, \mathrm{IN}_i \vee \neg\, \mathrm{Gb}_i(X)) \tag{94}$$

$$\Box\forall X.\mathrm{Robot}(X) \Rightarrow (\neg\, \mathrm{IM}_i \vee \neg\, \mathrm{Gb}_i(X)) \tag{95}$$

$$\Box(\neg\, \mathrm{IN}_i \vee \neg\, \mathrm{IM}_i) \tag{96}$$

Finally, to link the robot transition system and the pieces of food and the food transition systems with each other we have:

$$\Box\left(\forall X \cdot \left(\mathrm{Robot}(X) \Rightarrow \left((\mathrm{G}(X) \vee \mathrm{D}(X) \vee \mathrm{Mf}(X)) \Leftrightarrow \left(\bigvee_j \mathrm{Gb}_j(X)\right)\right)\right)\right) \tag{97}$$

$$\Box(\forall X.\forall i, j, i \neq j(\mathrm{Robot}(X) \Rightarrow (\neg\, \mathrm{Gb}_i(X) \vee \neg\, \mathrm{Gb}_j(X)))) \tag{98}$$

We have "translated" the different transition systems we have developed in the previous parts. Now when we prove formulae if we explicitly introduce more than one robot we must remember to state that they are different (with the predicate "*NE*") and we have a model of a system with an arbitrary number of robots. For example, for two robots we would need to declare:

$$\mathrm{Robot}(R1), \mathrm{Robot}(R2), NE(R1, R2)$$

We now have a model of our system in monodic FOTL. For each robot declared, we can consider a "copy" of the transition system in LTL; whereas for each piece of food we must explicitly represent each transition system. The FOTL formulae we have constructed fall within the decidable, monodic two-variable fragment of FOTL.

6.3 TSPASS: a temporal monodic prover

Next we will consider automated proof over this temporal model. Just as a clausal resolution prover was developed for LTL, there are also clausal resolution provers for monodic FOTL (Konev *et al.*, 2005), called TeMP (Hustadt *et al.*, 2004) and TSPASS (Ludwig and Hustadt, 2009/2010). TeMP is the first automatic theorem prover for the monodic fragment of FOTL and TSPASS a new implementation with a different underlying architecture. As previously discussed, our model of a system (with an

arbitrary number of robots and a fixed number of pieces of food) is monodic, and it is obviously in FOTL. So we can translate it as input for TeMP or TSPASS and prove some properties of it.

The syntax of both TeMP and TSPASS is very simple:

- ![X] corresponds to $\forall X$;
- ?[X] corresponds to $\exists X$;
- $\sim$ corresponds to $\neg$;
- | corresponds to $\vee$;
- & corresponds to $\wedge$; and
- "next", "always" and "sometime" correspond to $\bigcirc$, $\Box$ and $\Diamond$, respectively.

We have carried out automated analysis of the above in TSPASS, for an infinite number of robots while ignoring the number of pieces of food. Sample results are given in Table VIII. As can be seen, deciding on these properties is quite quick.

Next in Table IX we provide results for an infinite number of robots and two pieces of food which are regenerated when all pieces of food are back in the nest. To reduce the number of constraints we replace the first-order version of the constraints (20)-(27) from Section 2.2 with:

$$\text{Robot}(X) \Rightarrow \Box\Diamond \text{G}(X) \tag{99}$$

In Table IX Ø in the "Time" column denotes the prover did not solve the problem within ten min and "??". In the column "satisfiable" denotes that we have not shown the problem to be satisfiable or otherwise.

Here, the problems that are unsatisfiable finish quite quickly but the satisfiable problems do not finish within 10 min.

6.4 Comment

Using FOTLs we can model the robot transition system representing infinite numbers of robots without having to explicitly describe each robot as we did in the propositional case. The formulae (63)-(81), the first-order translations of equations (20)-(29), and

Property	Meaning	Satisfiable?	Time (s)
$\exists X(\text{robot}(X) \wedge \Box \neg \text{D}(X))$	There is a robot that never reaches the state Deposit	No	0.340
$\exists X(\text{robot}(X) \wedge \Box \neg \text{S}(X))$	There is a robot that never reaches the state Scan	Yes	0.424
$\exists X(\text{robot}(X) \wedge \Box\text{D}(X))$	There is a robot always in the state Deposit	No	0.064
$\exists X(\text{robot}(X) \wedge \Box\Diamond\text{D}(X))$	There is a robot that can infinitely often reach the state Deposit	Yes	0.544
$\exists X(\text{robot}(X) \wedge \Box\Diamond\text{Re}(X))$	There is a robot that can infinitely often reach the state Rest	Yes	0.436
$\exists X(\text{robot}(X) \wedge \Diamond\Box \neg \text{D}(X))$	There is a robot that sometime does not visit the state Deposit again	No	0.856
$\exists X(\text{robot}(X) \wedge \Diamond\Box \neg \text{S}(X))$	There is a robot that sometime does not visit the state Scan again	Yes	0.448
$\exists X(\text{robot}(X) \wedge \Diamond\Box \neg \text{Ff}(X))$	There is a robot where sometime Foundfood is always false	No	0.580

Table VIII. Sample tests from use of TSPASS on the infinitely many robots model

Property	Meaning	Satisfiable?	Time (s)
$\exists X(\text{robot}(X) \wedge \Box \neg \text{D}(X)))$	There exists a robot that never deposits food	No	0.461
$\exists X(\text{robot}(X) \wedge \Diamond\Box \neg \text{D}(X)))$	There exists a robot that sometime in the future never deposits food again	No	0.573
$\exists X\text{robot}(X) \wedge \Diamond(\text{D} \wedge \bigcirc\Box \neg \text{D}(X)))$	There is a robot and at some point in the future it deposits food and from the next moment onwards does not deposit food again	No	1.125
$\exists X(\text{robot}(X) \wedge \Box\Diamond\text{D}(X))$	There exists a robot that can infinitely often deposit food	??	$\varnothing$
$\forall X(\text{robot}(X) \Rightarrow \Box\Diamond\text{D}(X))$	All robots infinitely often deposit food	??	$\varnothing$
$\exists X\text{robot}(X) \wedge \Box\Diamond\text{IN}_1$	There is a robot and food 1 can be infinitely often in the nest	??	$\varnothing$
$\exists X\text{robot}(X) \wedge \Box\Diamond\text{IN}_2$	There is a robot and food 2 can be infinitely often in the nest	??	$\varnothing$
$\exists X\text{robot}(X) \wedge \Box \neg \text{IN}_1$	There is a robot and food 1 is never in the nest	No	1.148
$\exists X\text{robot}(X) \wedge \Box \neg \text{IN}_2$	There is a robot and food 2 is never in the nest	No	1.119
$\exists X\text{robot}(X) \wedge \Box\Diamond\text{IM}_1$	Food 1 can be infinitely often in the arena	??	$\varnothing$
$\exists X\text{robot}(X) \wedge \Box\Diamond\text{IM}_2$	Food 2 can be infinitely often in the arena	??	$\varnothing$
$\exists X\text{robot}(X) \wedge \Box \neg \text{IM}_1$	There is a robot and food 1 is never in the arena	No	1.033
$\exists X\text{robot}(X) \wedge \Box \neg \text{IM}_2$	There is a robot and food 2 is never in the arena	No	1.024
$\exists X(\text{robot}(X) \wedge \Box \neg \text{Gb}_1(X))$	There is a robot that never grabs food 1	??	$\varnothing$
$\exists X\text{robot}(X) \wedge \Box \neg \text{Gb}_2(X)$	There is a robot that never grabs food 2	??	$\varnothing$
$\exists X\text{robot}(X) \wedge \Box(\neg \text{Gb}_1(X) \wedge \neg \text{Gb}_2(X))$	There is a robot that does not grab any food	No	0.164
$(\exists X\text{robot}(X) \wedge \Box(\forall X\text{robot}(X) \Rightarrow \neg \text{Gb}_1(X)))$	There is a robot and food 1 is never grabbed by any robot	No	0.227
$(\exists X\text{robot}(X) \wedge \Box(\forall X\text{robot}(X) \Rightarrow \neg \text{Gb}_2(X)))$	There is a robot and food 2 is never grabbed by any robot	No	0.272
$(\exists X\text{robot}(X) \wedge \Diamond\Box(\neg \text{Gb}_1(X) \wedge \neg \text{Gb}_2(X))$	There is a robot that at some point in the future does not Grab Food 1 or 2 any more pieces of food infinitely often	No	0.180

Table IX.
Sample tests from use of TSPASS on the "at least one robot, two pieces of food" model

properties proved all fall into the monodic monadic fragment of FOTL which is decidable (Hodkinson *et al.*, 2000). The properties we tried to prove with TSPASS finished fairly quickly. When trying to add the representation for food things become more difficult. Introducing a predicate Food(X) to allow infinite pieces of food, as we did with robots, causes us to move out of the monodic fragment. To avoid this we have

to explicitly represent each piece of food as we did in the propositional case. The resulting formulae lie within the decidable monodic two variable fragment of FOTL. We can see from the results from Table IX that now TSPASS does not finish within 10 min for several cases even though we simplified the number of constraints using just constraint (99) (plus equations (80) and (81)). We note that the cases that do not finish in the time limit we expect to be satisfiable. Essentially we have a conjunction of formulae as input to the prover. In the case of an unsatisfiable set of formulae, the prover may be able to find a contradiction, and declare the formulae as unsatisfiable fairly quickly without having to explore the full search space. With satisfiable sets of formulae, however, we need to explore the whole search space before being able to declare the formulae are satisfiable. The improvement of techniques to try and reduce the search space without affecting the satisfiability of formulae may help with this.

7. Related work

Previously, we have formally specified the alpha algorithm for a wireless connected swarm (Nembrini *et al.*, 2002) using temporal logics. Each robot has range limited wireless communication and can only receive and broadcast messages to robots within range. A robot moves forward, periodically broadcasting "I am here" messages to its neighbours. If, at some point, the number of robots within range falls below some threshold (i.e. the robot is assumed to be moving out of the swarm) it turns 180° and moves in the new direction. When the number of neighbours rises above the threshold (i.e. the robot has regained the swarm) the robot executes a random turn to avoid the swarm collapsing in on itself. In Winfield *et al.* (2005) we specified this swarm algorithm using propositional LTL. In Chen (2005) this temporal specification of swarm algorithms was used to explore ways to generate implementations from a formal specification.

In Kloetzer and Belta (2007) a model checking approach is adopted to considering the motion of robot swarms. A hierarchical framework is suggested to abstract away from the many details of the problem. First, a continuous abstraction is used to capture the main features of the swarm's position and size. An example considered is using the centroid and variance of robot positions. Next the continuous abstraction is discretised, to which model checking can be applied. That approach differs in a number of ways to this paper. First, it concentrates on robot motion and behaviour such as obstacle avoidance, swarm cohesion, and collision avoidance. Our work has abstracted away from considerations of robot movement. Second, we are interested in the emergent behaviour of the swarm so model individual robots whereas in that paper the descriptions of individual robots are abstracted away from. Third, that paper assumes a centralised communication architecture, which is not assumed in our work. A related paper is (Fainekos *et al.*, 2005) which again considers robot motion but does not discuss robot swarms. That again produces a discrete representation of the continuous space of movement producing a finite state transition system. A model checker is used to produce traces that satisfy particular properties (e.g. visiting regions in a particular order, eventually visiting a region but avoiding other regions on the way). These are then used to produce a continuous movement plan whilst maintaining the required property. The differences between that work and ours are that they focus on motion and do not mention robot swarms.

In this paper, we chose to carry out the automatic proof of properties of the temporal specifications in our foraging robots scenario using the propositional and first-order linear-time temporal resolution provers TRP + + and TSPASS. An alternative would have been to use a tableau-based prover. To show a property P follows from a specification S, i.e. $S \Rightarrow P$ is a valid formula, tableau algorithms, similar to resolution, would try to show that $S \wedge \neg P$ is unsatisfiable. A graph is constructed with nodes containing sets of temporal formula. The graph is extended by applying a series of tableau rules to the main operator of some formula. These deconstruct temporal formulae into formulae that must hold now and conditions on future moments. Deletions are carried out to remove parts of the graph, which cannot be used to construct a model. If applying the tableau algorithm to a formula φ results in the underlying structure constructed being empty then this means that φ is unsatisfiable. Alternatively if applying the tableau algorithm to φ results in the underlying structure being non-empty then this can be used to construct a model for φ, so φ must be satisfiable. Tableau calculi are described in Wolper (1985), Schwendimann (1998), Janssen (1999) and implementations of tableau algorithms for LTL can be found at The Logics WorkBench (Balsiger *et al.*, 1998) and the Tableau Workbench (Abate and Goré, 2003). Calculi for tableau for monodic FOTL were given in Kontchakov *et al.* (2004).

We chose to use a resolution rather than tableau-based approach as the representation of the underlying transition systems are close to the normal form our resolution systems are based on. Also implemented resolution-based reasoners for both propositional temporal logics and monodic FOTLs were available whereas the only implemented temporal tableau-based reasoners we know of are propositional.

Model checking (Clarke *et al.*, 2000) is a popular technique for showing the satisfiability of temporal formulae given a model of the system over which the formulae are to be checked. Implemented model checkers that allow properties specified in LTL include NuSMV (Cimatti *et al.*, 2002) and Spin (Holzmann, 1997). We chose to consider a deductive, resolution-based approach due to the fact that there is a chance of extending this to infinite state systems by using FOTL whereas standard model checking must be finite state. As mentioned previously, model checkers have been applied to the verification of the motion robot swarms in Kloetzer and Belta (2007), however many details are abstracted away.

Other formalisms have been considered to specify and verify aspects of robot swarms. In Rouff *et al.* (2007) four formal methods were selected to specify part of the Autonomous Nano Technology Swarm mission which will send small swarms of small spacecraft to study the asteroid belt. The chosen formal methods are communicating sequential processes, weighted synchronous calculus of communicating systems, X-machines and unity logic. These are being used alongside techniques from Agent Oriented Software Engineering. The authors conclude the need to develop new formal techniques alongside specialised sets of models and software processes based on a number of formal methods and other areas software engineering.

In Lerman *et al.* (2005) the authors use distributed stochastic processes to model swarm robots. The modelling is macroscopic, i.e. it directly describes the collective behaviour of the swarm. These models are compared with simulations and experiments on real robots with good agreement between the models and simulations or experiments. The paper (Martinoli *et al.*, 2004) also develops a stochastic approach.

Robot swarms have been studied using Lyapunov stabilty (Harper and Winfield, 2006). The underlying behaviour is represented by state vectors and the change of state is modelled as a state trajectory. Different categories of behaviour are specified which, if proven, show the systems will converge on some equilibrium condition, is sufficiently close to this, or otherwise. This differs from the approach we take in this paper as we focus on a logical rather than numerical/dynamic representation. Also there are many other approaches for analysing and modelling complex swarm architectures (Gordon-Spears and Kiriakidis, 2004).

8. Conclusions and future work

8.1 Concluding remarks

We have taken the case study of a swarm of robots foraging for food and specified this in both propositional and monodic first-order LTLs. We have abstracted away from details about robot movement in the physical world, i.e. we have not recorded its location co-ordinates, but focused on representing its phases of activity, e.g. scanning the arena or moving homewards. The different phases of the robots were originally represented by a state transition system and we suggested a number of constraints to avoid unwanted infinite loops in the systems. We then introduced a transition system representing food and have proved a number of properties with varying (small) numbers of robots and pieces of food. As the original paper (Liu *et al.*, 2007) this case study was based on was concerned with energy levels of the robots we introduced another transition system to represent this and again proved several properties.

The proofs have been carried out using TRP++ a temporal resolution theorem prover for propositional temporal logics. TRP++ has completed almost all of the proofs in the scenarios we have tried with up to two robots and three pieces of food. The formulae taking the longest times (or not completing) concerned properties of the form infinitely often. Table V gives an indication of why the times taken by TRP++ increase by considering the number of propositions required. For example, the case of two robots and three pieces of food requires 38 propositions which involves 2^{38} different interpretations with the product of the transition systems having at most 5, 184 states.

The most expensive part for these type of temporal resolution provers is dealing with sometime clauses (whose right hand sides are of the form $\Diamond l$). The way we have formalised the additional constraints in Sections 2.2 have introduced several formulae of this type. We may get improved results and be able to deal automatically with larger number of robots and food by reducing such formulae to a minimum. Further, when representing the transition system we must explicitly state that the system can only be in exactly one state or phase at any moment in time. This produces many temporal clauses and we believe slows down the system. An alternative would be to use a temporal prover that deals with such constraints as part of the input. We have been investigating such logics and associated calculi in Dixon *et al.* (2007a, b, 2008).

One disadvantage of this approach is that to increase the number of robots we must add the formulae representing the robot transition system to the input file and update the formulae linking the robot and food transition systems. Seeking for a better way to proceed we utilised FOTLs where variables could represent robots and pieces of food. We saw that we could use this approach to represent robots thus potentially modelling swarms of arbitrary size. However, this did not extend also to items of food because

representing information that some robot had grabbed some food in the next moment in time was outside the decidable fragment we targeted.

The prover TSPASS managed to prove all the properties of the robot transition system but struggled with the satisfiable properties when combined with two pieces of food. Nevertheless, this work shows that propositional deductive temporal verification is viable for small numbers of robots, while first-order deductive temporal verification is possible for arbitrary sized swarms.

8.2 Future work

Our aims for future work are as follows:

- as temporal prover technology develops, we will address more complex and sophisticated variations of the verification problems tackled here;
- while we have initially considered the problem of representing real-time issues on the robotic behaviour, this needs to be explored further, perhaps using a more powerful, specifically real-time, temporal framework;
- similarly, the behaviour of a robot is essentially probabilistic, with actions and sensor reading being fuzzy, and so extension to probabilistic temporal logics will be explored;
- while we have considered some aspects of arbitrarily sized swarms, we would like to develop this much further, particularly considering properties of numbers of robots; and
- to address the problem of specifying and verifying fault tolerance within such arbitrarily sized swarms of robots (Winfield and Nembrini, 2006; Fisher *et al.*, 2009).

Note

1. The original version of SNF did not include global clauses but, as global clauses can be re-written as an initial clause and a step clause, this version is equivalent.

References

Abate, P. and Goré, R. (2003), "The tableaux work bench", *Proceedings of International Workshop on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux), Aix-en-Proxence*, Lecture Notes in Artificial Intelligence, Vol. 2796, pp. 230-6.

Balsiger, P., Heuerding, A. and Schwendimann, S. (1998), "Logics workbench 1.0", *Proceedings of International Workshop on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux)*, Lecture Notes in Computer Science, Springer, Oisterwijk, Vol. 1397, p. 35.

Beni, G. (2005), "From swarm intelligence to swarm robotics", *Proceedings of International Workshop on Swarm Robotics (SAB), Revised Selected Papers*, Lecture Notes in Computer Science, Vol. 3342, Springer, Berlin, pp. 1-9.

Bonabeau, E., Dorigo, M. and Theraulaz, G. (1999), *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NY.

Chen, D. (2005), "A simulation environment for swarm robotic system based on temporal logic specifications", Master's thesis, University of the West of England, Bristol.

Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A. (2002), "NuSMV 2: an opensource tool for symbolic model checking",

Proceedings of 14th International Conference on Computer-aided Verification (CAV), Lecture Notes in Computer Science, Vol. 2404, Springer, Berlin, pp. 359-64.

Clarke, E., Grumberg, O. and Peled, D.A. (2000), *Model Checking*, MIT Press, Cambridge, MA.

Dixon, C., Fisher, M. and Konev, B. (2007a), "Temporal logic with capacity constraints", *Proceedings of 6th International Symposium on Frontiers of Combining Systems (FroCos)*, Lecture Notes in Computer Science, Vol. 4720, Springer, Berlin, pp. 163-77.

Dixon, C., Fisher, M. and Konev, B. (2007b), "Tractable temporal reasoning", *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI)*, AAAI Press, Portland, OR, pp. 318-23.

Dixon, C., Fisher, M., Konev, B. and Lisitsa, A. (2008), "Practical first-order temporal reasoning", *Proceedings of 15th International Symposium on Temporal Representation and Reasoning (TIME)*, IEEE Press, Los Alamitos, CA.

Emerson, E.A. (1990), "Temporal and modal logic", in van Leeuwen, J. (Ed.), *Handbook of Theoretical Computer Science*, Elsevier, Amsterdam, pp. 996-1072.

Fainekos, G., Kress-Gazit, H. and Pappas, G. (2005), "Temporal logic motion planning for mobile robots", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Barcelona, Spain, pp. 2020-5.

Fisher, M. (2007), "Temporal representation and reasoning", in van Harmelen, F., Porter, B. and Lifschitz, V. (Eds), *Handbook of Knowledge Representation, Foundations of Artificial Intelligence*, Vol. 2, Elsevier, Amsterdam.

Fisher, M., Dixon, C. and Peim, M. (2001), "Clausal temporal resolution", *ACM Transactions on Computational Logic*, Vol. 2 No. 1, pp. 12-56.

Fisher, M., Konev, B. and Lisitsa, A. (2006), "Practical infinite-state verification with temporal reasoning", *Verification of Infinite State Systems and Security, NATO Security through Science Series: Information and Communication*, Vol. 1, IOS Press, Amsterdam.

Fisher, M., Konev, B. and Lisitsa, A. (2009), "Temporal verification of fault-tolerant protocols", *Methods, Models and Tools for Fault Tolerance*, Lecture Notes in Computer Science, Vol. 5454, Springer, Berlin, pp. 44-56.

Gabbay, D., Pnueli, A., Shelah, S. and Stavi, J. (1980), "The temporal analysis of fairness", *Proceedings of 7th ACM Symposium on the Principles of Programming Languages, Las Vegas, NV*, ACM, New York, NY, pp. 163-73.

Gordon-Spears, D. and Kiriakidis, K. (2004), "Reconfigurable robot teams: modeling and supervisory control", *IEEE Transactions on Control Systems Technology*, Vol. 12 No. 5, pp. 763-9.

Harper, C. and Winfield, A. (2006), "A methodology for provably stable behaviour-based intelligent control", *Robotics and Autonomous Systems*, Vol. 54 No. 1, pp. 52-73.

Hodkinson, I., Wolter, F. and Zakharyashev, M. (2000), "Decidable fragments of first-order temporal logics", *Annals of Pure and Applied Logic*, Vol. 106 Nos 1-3, pp. 85-134.

Hodkinson, I., Kontchakov, R., Kurucz, A., Wolter, F. and Zakharyaschev, M. (2003), "On the computational complexity of decidable fragments of first-order linear temporal logics", *Proceedings of TIME-ICTL 2003*, IEEE Press, Los Alamitos, CA.

Holzmann, G.J. (1997), "The model checker spin", *IEEE Transactions on Software Engineering*, Vol. 23 No. 5, pp. 279-95, (special issue on Formal Methods in Software Practice).

Hustadt, U. and Konev, B. (2003), "TRP++2.0: a temporal resolution prover", *Proceedings of 19th International Conference on Automated Deduction (CADE)*, Lecture Notes in Artificial Intelligence, Vol. 2741, Springer, Berlin, pp. 274-8.

Hustadt, U., Konev, B., Riazanov, A. and Voronkov, A. (2004), "TeMP: a temporal monodic prover", in Basin, D.A. and Rusinowitch, M. (Eds), *Proceedings of the Second International Joint Conference on Automated Reasoning (IJCAR 2004)*, Lecture Notes in Artificial Intelligence, Vol. 3097, Springer, Berlin, pp. 326-30.

Janssen, G. (1999), "Logics for digital circuit verification: theory, algorithms, and applications", PhD thesis, Eindhoven University of Technology, Eindhoven.

Kloetzer, M. and Belta, C. (2007), "Temporal logic planning and control of robotic swarms by hierarchical abstractions", *IEEE Transactions on Robotics*, Vol. 23, pp. 320-30.

Konev, B., Degtyarev, A., Dixon, C., Fisher, M. and Hustadt, U. (2005), "Mechanising first-order temporal resolution", *Information and Computation*, Vol. 199 Nos 1-2, pp. 55-86.

Kontchakov, R., Lutz, C., Wolter, F. and Zakharyaschev, M. (2004), "Temporalizing tableaux", *Studia Logica*, Vol. 76, pp. 91-134.

Lerman, K., Martinoli, A. and Galstyan, A. (2005), "A review of probabilistic macroscopic models for swarm robotic systems", *Swarm Robotics*, Lecture Notes in Computer Science, Vol. 3342, Springer, Berlin, pp. 143-52.

Liu, W., Winfield, A.F.T., Sa, J., Chen, J. and Dou, L. (2007), "Strategies for energy optimisation in a swarm of foraging robots", *Proceedings of 2nd International Workshop on Swarm Robotics (SAB)*, Lecture Notes in Computer Science, Vol. 4433, Springer, Berlin, pp. 14-26.

Ludwig, M. and Hustadt, U. (2009/2010), "Implementing a fair monodic temporal prover", *AI Communications* (in press).

Martinoli, A., Easton, K. and Agassounon, W. (2004), "Modeling swarm robotic systems: a case study in collaborative distributed manipulation", *International Journal of Robotics Research*, Vol. 23 No. 4, pp. 415-36.

Nembrini, J., Winfield, A.F.T. and Melhuish, C. (2002), "Minimalist coherent swarming of wireless connected autonomous mobile robots", *Proceedings of 7th International Conference on Simulation of Adaptive Behavior (ICSAB)*, MIT Press, Cambridge, MA, pp. 373-82.

Pnueli, A. (1981), "The temporal semantics of concurrent programs", *Theoretical Computer Science*, Vol. 13, pp. 45-60.

Rouff, C.A., Hinchey, M.G., Pena, J. and Ruiz-Cortes, A. (2007), "Using formal methods and agent-oriented software engineering for modeling NASA swarm-based systems", *Proceedings of International Swarm Intelligence Symposium (SIS)*, IEEE Press, Los Alamitos, CA, pp. 348-55.

Sahin, E. and Winfield, A.F.T. (2008), "Special issue on swarm robotics", *Swarm Intelligence*, Vol. 2 Nos 2-4, pp. 69-72.

Schwendimann, S. (1998), "A new one-pass tableau calculus for PLTL", *Proceedings of International Workshop on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, Lecture Notes in Artificial Intelligence, Vol. 1397, Springer, Berlin, pp. 277-91.

Sistla, A.P. and Clarke, E.M. (1985), "Complexity of propositional linear temporal logics", *Journal of the ACM*, Vol. 32 No. 3, pp. 733-49.

Sistla, A.P., Vardi, M. and Wolper, P. (1987), "The complementation problem for Büchi automata with applications to temporal logic", *Theoretical Computer Science*, Vol. 49, pp. 217-37.

Spears, W.M., Spears, D.F., Hamann, J.C. and Heil, R. (2004), "Distributed, physics-based control of swarms of vehicles", *Autonomous Robots*, Vol. 17 Nos 2-3, pp. 137-62.

TRP++ (2002), "Temporal resolution prover", available at: www.csc.liv.ac.uk/~konev/software/trp++

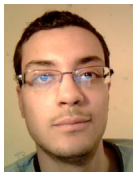
Winfield, A. and Nembrini, J. (2006), "Safety in numbers: fault tolerance in robot swarms", *International Journal of Modelling Identification and Control*, Vol. 1 No. 1, pp. 30-7.

Winfield, A., Sa, J., Fernández Gago, M.-C., Dixon, C. and Fisher, M. (2005), "On formal specification of emergent behaviours in swarm robotic systems", *International Journal of Advanced Robotic Systems*, Vol. 2 No. 4, pp. 363-70.

Wolper, P. (1985), "The tableau method for temporal logic: an overview", *Logique et Analyse*, Vol. 110-111, pp. 119-36.

Wolter, F. and Zakharyaschev, M. (2002), "Axiomatizing the monodic fragment of first-order temporal logic", *Annals of Pure and Applied Logic*, Vol. 118 Nos 1-2, pp. 133-45.

About the authors



Abdelkader Behdenna is a Research Student at the Departement d'informatique de l'Ecole Normale Supérieure de Cachan. This work is the result of a extended visit to Liverpool in 2008.

Clare Dixon is a Senior Lecturer in the Department of Computer Science, University of Liverpool. She has previously held research positions at the University of Manchester and Manchester Metropolitan University. She is interested in specification and verification of systems using non-classical logics, temporal reasoning, automated reasoning and theorem proving. Clare Dixon is the corresponding author and can be contacted at: cldixon@liverpool.ac.uk

Michael Fisher is a Professor in the Department of Computer Science, University of Liverpool. He has been active in research for over 20 years and his research interests involve logic in computer science and artificial intelligence, particularly temporal reasoning, theorem-proving, programming languages, formal verification and autonomous and agent-based systems.