Approximating ATL* in ATL
Extended Abstract

Aidan Harding[1], Mark Ryan[1], and Pierre-Yves Schobbens[2]

[1] School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK

[2] Institut d'Informatique, Facultés Universitaires de Namur, Rue Grandgagnage 21, 5000 Namur, Belgium

Abstract. The temporal logic ATL [2] has proved useful in specifying systems that can be viewed as the parallel composition of a set of *agents*. It has tool-support for model checking and simulation in the form of MOCHA [1]. ATL* is a more expressive form of ATL which provides a more natural way to write specifications. Whilst ATL can be model checked in linear time (relative to the size of the model), ATL* is 2EXPTIME-complete [2]. Here we present a method of "translating" an ATL* formula, into ATL so that model checking can then be performed. This method cannot, in general, be entirely exact but instead produces a strong and a weak bound. From these we may be able to infer whether the original formula was satisfied. To minimise the number of undecided cases, the bounds must be as close as possible to the original. Exact translations help to ensure that this is so, and we have identified a number of patterns which can be translated without loss. Case studies support the method by showing that most ATL* formulae attempted did yield conclusive results, even after approximation.

1 Introduction

The aim of this work is to provide a method of model checking ATL* specifications using a model checker for ATL. Model checking ATL* directly is infeasible, so we have taken the option of rewriting a given ATL* property φ, into a pair of ATL properties, φ_s and φ_w. By checking these two properties, we may be able to infer whether or not φ is satisfied. There is some uncertainty in the method, because φ_s and φ_w do not capture *all* of the information in φ. In some cases this abstraction of the formula may be too coarse, making it impossible to decide whether φ is true or not. To add to the accuracy and sophistication of our method, exact (i.e. information preserving) transformations on ATL* formulae are used, wherever possible.

1.1 Why ATL*?

ATL* [2] is a temporal logic for reasoning about systems composed of *agents*. It is desirable to write specifications in ATL* rather than CTL* or LTL because it allows us to distinguish between the possible choices of agents, which are the sources of non-determinism. By recognising the agents in a system, it is possible to separate out properties which would otherwise remain hidden.

A formal look at ATL and ATL* is deferred until Section 2, but first we look at a motivating example for their use. Consider a basic phone system. A natural question to ask about it is *"Can two users, i and j, cooperate such that in the future, they will be talking to one and other"*. The idea of their cooperation is that we wish to exclude paths such as those where i never dials j, or j goes off-hook every time the connection is being attempted. However, we allow for the rest of the system to be as awkward as possible e.g. another phone k may try to interfere by also dialling j and the exchange may solve the conflict by favouring k. In CTL, this cannot be expressed – it is only possible to write about all computation paths or the existence of at least one. However, in ATL we can write about the paths enforceable by the cooperation of i and j: $\langle\langle i, j\rangle\rangle \mathrm{F}(phone[i].talking \wedge phone[j].talking \wedge currentConnection[j] = i)$. Clearly there are many other systems where ATL* is beneficial, allowing us to reason about the capabilities of sets of agents in cooperation/opposition.

Just as CTL* generalises CTL by allowing temporal operators to be nested directly, ATL* generalises ATL. ATL* can be more useful than ATL due to this extra expressiveness. It provides all of the advantages of LTL whilst retaining the ability to reason about the capabilities of agents. LTL specifications are claimed to be easier to write in [8], and to be more useful for reasoning about concurrent systems in [6]. By using ATL*, we have the best of both worlds (in expressivity).

1.2 Approximating ATL* in ATL

Since ATL* is strictly more expressive than ATL, we cannot hope to translate all possible formulae exactly from ATL* into ATL. The complexity of model checking ATL is linear in the size of the model, whilst model checking ATL*

directly is doubly exponential [2]. Our method is a partial solution to the problem of model checking ATL* – it returns within a feasible time, but may lose some of the original information. In essence, this is achieved by approximating a single property φ, into two properties φ_s and φ_w which surround the original property with a strong and a weak bound such that:

$$\varphi_s \Rightarrow \varphi \Rightarrow \varphi_w \tag{1.1}$$

We can then model check the ATL formulae with MOCHA to deduce the satisfaction of φ. If we find φ_s to be true, then φ is true; If we find φ_w to be false, then φ is false; If φ_s is false and φ_w is true, we cannot decide whether φ is true or false.

It is essential to minimise the number of times our method may come back undecided. This means ensuring that the strong and weak bound are as close as possible to φ. To do this, we use exact equivalences, where possible. These equivalences are designed to make φ in some sense, better with each application i.e. they should make the property closer to ATL than it was before. When no more equivalences are applicable, approximation is used to copy path quantifiers over temporal operators e.g.

$$\langle\!\langle A\rangle\!\rangle \mathrm{F}\mathrm{G}\varphi \rightsquigarrow \begin{array}{ll} \langle\!\langle A\rangle\!\rangle \mathrm{F}\langle\!\langle A\rangle\!\rangle \mathrm{G}\varphi & \text{(strong)} \\ \langle\!\langle A\rangle\!\rangle \mathrm{F}\exists \mathrm{G}\varphi & \text{(weak)} \end{array}$$

After each approximation, equivalences are applied until either the formula is in ATL or more approximation is needed. With the complete set of approximations provided, any well-formed ATL* formula can be translated into into a pair of well-formed ATL formulae.

The rest of the paper is organised as follows: Section 2 summarises the syntax and semantics of Alur and Henzinger's ATL; Section 3 lists the exact equivalences used in the translation process; Section 4 covers the approximations used in the translation process; Section 5 takes a look at how the rules are used and what properties they have; Section 6 has a model of a telephone system with ATL* specifications which have been translated and checked with MOCHA; Finally, Section 7 looks at some other methods of moving between temporal logics with differing expressivity.

2 Alternating-Time Temporal Logic

Alternating-Time Temporal Logic [2] (ATL) is a temporal logic for reasoning about *reactive systems* comprised of *agents*. It contains the usual temporal operators (next, always, until) plus cooperation modalities $\langle\!\langle A\rangle\!\rangle\varphi$, where A is a set of agents. This modality quantifies over the set of behaviours and means that A have a collective strategy to enforce φ, whatever the choices of the other players. ATL generalises CTL, and similarly ATL* generalises CTL*, μ-ATL generalises the μ-calculus. These logics can be model-checked by generalising the techniques of CTL, often with the same complexity.

This section contains a brief review of ATL, as we have used it in this paper. For a more detailed treatment, the interested reader is referred to [2].

2.1 Alternating Transition Systems

ATL is interpreted over Alternating Transition Systems (ATS) which are Kripke structures, extended to represent the choices of agents.

An ATS is a 5-tuple $\langle \Pi, \Sigma, Q, \pi, \delta \rangle$ where

- Π is a set of propositions
- Σ is a set of agents
- Q is a set of states
- $\pi : Q \to 2^{\Pi}$ maps each state to the propositions which are true in that state
- $\delta : Q \times \Sigma \to 2^{2^Q}$ is a transition function from a state, q, and an agent, a, to the set of a's choices. a's choices are sets of states, and one particular choice is taken, Q_a. The next state of the system is the intersection of the choices of all agents $\bigcap_{a \in \Sigma} Q_a$.
  The transition function is *non-blocking* and *unique* i.e. for every state, the intersection of all possible choices of all agents is singleton.

For two states q, q' and an agent a, q' is an *a-successor* of q if there exists some $Q' \in \delta(q, a)$ such that $q' \in Q'$. The set of a-successors of q is denoted $succ(q, a)$. For two states q and q', q' is a *successor* of q if $\forall a \in \Sigma \; q' \in succ(q, a)$. A computation, λ, is defined as an infinite sequence of states $q_0, q_1, q_2, \ldots$ such that for all $i \geq 0$, q_{i+1} is the successor of q_i.

Subsegments of a computation path $\lambda = q_1, q_2, \ldots$ are denoted by postfixing an interval in square brackets. For example, $\lambda[i, j] = q_i, \ldots, q_j$, $\lambda[i, \infty] = q_i, \ldots$ and $\lambda[i] = q_i$.

2.2 ATL Syntax

Let Π be a set of atomic propositions and Σ a set of agents. The syntax of ATL is given by

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\langle A \rangle\rangle(\varphi_1 \, \mathcal{U} \, \varphi_2) \mid \langle\langle A \rangle\rangle(\varphi_1 \, \mathcal{R} \, \varphi_2)$$

where $p \in \Pi$ and $A \subseteq \Sigma$[1]. We use the usual abbreviations for $\to$, $\wedge$ in terms of $\neg$, $\vee$. The operator $\langle\langle \; \rangle\rangle$ is a path quantifier, and $\mathcal{U}$ (*until*) and $\mathcal{R}$ (*release*) are temporal operators. As in CTL, we write $\mathrm{F}\varphi$ for $\top \, \mathcal{U} \, \varphi$ and $\mathrm{G}\varphi$ for $\top \, \mathcal{R} \, \varphi$.

While the formula $\langle\langle A \rangle\rangle\psi$ means that the agents in A can cooperate to make ψ true (they can "enforce" ψ), the dual formula $[[A]]\psi$ means that the agents in A cannot cooperate to make ψ false (they cannot "avoid" ψ) i.e. $[[A]]\psi \equiv \neg\langle\langle A \rangle\rangle\neg\varphi$

Since ATL is a generalisation of CTL, we can use CTL as shorthand for some cases of ATL i.e. write $\forall\psi$ for $\langle\langle \emptyset \rangle\rangle\psi$ and $\exists\psi$ for $\langle\langle \Sigma \rangle\rangle\psi$. The logic ATL* generalises ATL in the same way that CTL* generalises CTL, namely by allowing path quantifiers and temporal operators to be nested arbitrarily.

[1] Following Lamport's warning that the X operator leads to over-specification [7] and for simplicity, we differ from [2] by omitting X.

2.3 ATL* Semantics

In ATL*, there are two types of formulae: *state formulae* are evaluated over states, and denoted here as φ; *path formulae* are evaluated over computation paths, and denoted ψ. To define the semantics of ATL*, the notion of *strategies* is used. A strategy for an agent a is a mapping $f_a : Q^+ \to 2^Q$ such that for all $\lambda \in Q^*$ and all $q \in Q$, we have $f_a(\lambda \cdot q) \in \delta(q, a)$. The strategies map finite prefixes of λ-computations to a choice in $\delta(q, a)$ as suggested by the strategy.

The *outcome* of a strategy must also be defined. For a state q, a set of agents A, and a family of strategies $F_A = \{f_a | a \in A\}$ the outcomes of F_A from q are denoted $out(q, F_A)$. They are the q-computations that the agents in A can enforce by following their strategies. $\lambda = q_0, q_1, q_2 \ldots$ is in $out(q, F_A)$ if $q = q_0$ and for all positions $i \geq 0$ q_{i+1} is the successor of q_i satisfying $q_{i+1} \in \bigcap_{a \in A} f_a(\lambda[0, i])$.

The semantics of ATL* are defined inductively:

- $\lambda \vDash p$ iff $p \in \pi(\lambda[0])$
- $\lambda \vDash \neg\varphi$ iff $\lambda \nvDash \varphi$
- $\lambda \vDash \varphi_1 \vee \varphi_2$ iff $\lambda \vDash \varphi_1$ or $\lambda \vDash \varphi_2$
- $\lambda \vDash \varphi$ iff $\lambda[0] \vDash \varphi$, if φ is a state formula
- $\lambda \vDash \langle\langle A \rangle\rangle \psi$ iff there exists a set of strategies, F_A one for each agent in A, such that $\forall \lambda \in out(q, F_A)$, we have $\lambda \vDash \psi$
- $\lambda \vDash \psi_1 \,\mathcal{U}\, \psi_2$ iff $\exists i \geq 0 . \lambda[i, \infty] \vDash \psi_2$ and $\forall 0 \leq j < i \lambda[j, \infty] \vDash \psi_1$.
- $\lambda \vDash \psi_1 \,\mathcal{R}\, \psi_2$ iff $\forall i \geq 0$, we have $\lambda[i, \infty] \vDash \psi_2$ unless there exists a position $0 \leq j < i$ such that $\lambda[j, \infty] \vDash \psi_1$.

3 Equivalences

These exact transformations are applied at the first stage of re-writing, to eliminate redundancy. In some cases, it is possible to perform the entire translation at this exact level. Discussion of how we may sensibly apply these rules is deferred until Section 5, when all of the necessary rules have been introduced.

We shall consider both $\wedge$ and $\vee$ as part of the basic language for our rule-set. The temporal operators we shall use are Until $\mathcal{U}$, Release $\mathcal{R}$. MOCHA accepts $\mathcal{U}$ but not $\mathcal{R}$. However, it does accept Weak Until (While). $\mathcal{W}$ and Release can be related as follows:

$$\psi_1 \,\mathcal{R}\, \psi_2 \equiv \psi_2 \,\mathcal{W}\, (\psi_1 \wedge \psi_2) \qquad \psi_1 \,\mathcal{W}\, \psi_2 \equiv \psi_2 \,\mathcal{R}\, (\psi_2 \vee \psi_1) \tag{3.1}$$

Release is used because it is more natural to use the dual of Until and it can still be translated into acceptable input for MOCHA.

We assume that the input formula is in negation normal form, and this be easily achieved with known LTL and ATL identities.

3.1 LTL Equivalences

LTL equivalences can be used to replace parts of ATL* sub-formulae and also serve as inspiration for some native ATL* rules. Each rule is applied left to right and reduces the number of nested temporal operators. Some of the equivalences below are from [9], others extend or generalise them. Where a rule requires knowing that $\varphi_1 \Rightarrow \varphi_2$, this is established using the heuristic method described in [9].

Future and Global Equations 3.2 to 3.8 are generalised by 3.9 to 3.15, below. These F and G abbreviations are included because their readability aids the intuition behind 3.9 to 3.15. The duals are used in practice, but omitted here.

$$\mathrm{FF}\varphi \equiv \mathrm{F}\varphi \tag{3.2}$$
$$\mathrm{FGF}\varphi \equiv \mathrm{GF}\varphi \tag{3.3}$$
$$\mathrm{F}(\varphi_1 \vee \mathrm{F}\varphi_2) \equiv \mathrm{F}(\varphi_1 \vee \varphi_2) \tag{3.4}$$
$$\mathrm{F}(\varphi_1 \vee \mathrm{GF}\varphi_2) \equiv \mathrm{F}\varphi_1 \vee \mathrm{GF}\varphi_2 \tag{3.5}$$
$$\mathrm{F}(\varphi_1 \wedge \mathrm{FG}\varphi_2) \equiv \mathrm{F}\varphi_1 \wedge \mathrm{FG}\varphi_2 \tag{3.6}$$
$$\mathrm{F}(\varphi_1 \wedge \mathrm{GF}\varphi_2) \equiv \mathrm{F}\varphi_1 \wedge \mathrm{GF}\varphi_2 \tag{3.7}$$
$$\mathrm{FG}(\varphi_1 \wedge \mathrm{F}\varphi_2) \equiv \mathrm{FG}(\varphi_1 \wedge \varphi_2) \tag{3.8}$$

Until and Release

$$\varphi_1 \Rightarrow \varphi_2 \vdash \varphi_1 \,\mathcal{U}\, (\varphi_2 \,\mathcal{U}\, \varphi_3) \equiv \varphi_2 \,\mathcal{U}\, \varphi_3 \tag{3.9}$$
$$\varphi_1 \,\mathcal{U}\, (\varphi_2 \,\mathcal{R}\, (\varphi_1 \,\mathcal{U}\, \varphi_3)) \equiv \varphi_2 \,\mathcal{R}\, (\varphi_1 \,\mathcal{U}\, \varphi_3) \tag{3.10}$$
$$\varphi_1 \,\mathcal{U}\, (\varphi_2 \vee \varphi_1 \,\mathcal{U}\, \varphi_3) \equiv \varphi_1 \,\mathcal{U}\, (\varphi_2 \vee \varphi_3) \tag{3.11}$$
$$\varphi_1 \,\mathcal{U}\, (\varphi_2 \vee \varphi_3 \,\mathcal{R}\, (\varphi_1 \,\mathcal{U}\, \varphi_4)) \equiv \varphi_1 \,\mathcal{U}\, \varphi_2 \vee \varphi_3 \,\mathcal{R}\, (\varphi_1 \,\mathcal{U}\, \varphi_4) \tag{3.12}$$
$$\varphi_1 \Rightarrow \neg\varphi_3 \vdash \varphi_1 \,\mathcal{U}\, (\varphi_2 \wedge (\varphi_1 \,\mathcal{U}\, (\varphi_3 \,\mathcal{R}\, \varphi_4))) \equiv (\varphi_1 \,\mathcal{U}\, \varphi_2) \wedge (\varphi_1 \,\mathcal{U}\, (\varphi_3 \,\mathcal{R}\, \varphi_4)) \tag{3.13}$$
$$\begin{gathered}\varphi_1 \Rightarrow (\varphi_4 \vee \varphi_5), \varphi_1 \Rightarrow \neg\varphi_3 \vdash \\ \varphi_1 \,\mathcal{U}\, (\varphi_2 \wedge \varphi_3 \,\mathcal{R}\, (\varphi_4 \,\mathcal{U}\, \varphi_5)) \equiv (\varphi_1 \,\mathcal{U}\, \varphi_2) \wedge (\varphi_3 \,\mathcal{R}\, (\varphi_4 \,\mathcal{U}\, \varphi_5))\end{gathered} \tag{3.14}$$
$$\varphi_1 \Rightarrow \neg\varphi_2 \vdash \varphi_1 \,\mathcal{U}\, (\varphi_2 \,\mathcal{R}\, (\varphi_3 \wedge \varphi_1 \,\mathcal{U}\, \varphi_4)) \equiv \varphi_1 \,\mathcal{U}\, (\varphi_2 \,\mathcal{R}\, (\varphi_3 \wedge \varphi_1 \,\mathcal{U}\, \varphi_4)) \tag{3.15}$$
$$(\varphi_1 \,\mathcal{U}\, \psi) \wedge (\varphi_2 \,\mathcal{U}\, \psi) \equiv (\varphi_1 \wedge \varphi_2) \,\mathcal{U}\, \psi \tag{3.16}$$
$$\varphi \Rightarrow \psi \vdash \varphi \,\mathcal{U}\, \psi \equiv \psi \tag{3.17}$$

3.2 ATL* Equivalences

As in Section 3.1, these rules are applied left to right and reduce the number of temporal operators which do not have a matching path quantifier. However, they are specific to ATL* – rather than removing redundant temporal operators, they match them to path quantifiers.

Conjunctions and Disjunctions If φ_1 and φ_2 are ATL path formulae, then both $\langle\langle A\rangle\rangle(\varphi_1 \wedge \varphi_2)$ and $\langle\langle A\rangle\rangle(\varphi_1 \vee \varphi_2)$ are ATL* formulae but neither are well-formed ATL formulae. In Table 1, we consider cases where they have exact equivalences in ATL that can be reached by rewriting[2]. One proof is given below, in order to show the general form of how the others proceed, others are available in the full paper.

Table 1. Equivalences for Conjunctions/Disjunctions of Path Formulae

ATL*	ATL (Assuming all φ are ATL state formulae)	
$\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2 \wedge \varphi_3 \,\mathcal{U}\, \varphi_4)$	$\langle\langle A\rangle\rangle((\varphi_1 \wedge \varphi_3) \,\mathcal{U}\, [(\varphi_2 \wedge \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4)) \vee (\varphi_4 \wedge \langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2))])$	(3.18)
$\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2 \vee \varphi_3 \,\mathcal{U}\, \varphi_4)$	$\langle\langle A\rangle\rangle[(\varphi_1 \wedge \varphi_3) \,\mathcal{U}\, (\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4))]$	(3.19)
$\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{R}\, \varphi_2 \wedge \varphi_3 \,\mathcal{R}\, \varphi_4)$	$\langle\langle A\rangle\rangle[((\varphi_1 \wedge \langle\langle A\rangle\rangle\varphi_3 \,\mathcal{R}\, \varphi_4) \vee (\varphi_3 \wedge \langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{R}\, \varphi_2))) \,\mathcal{R}\, (\varphi_2 \wedge \varphi_4)]$	(3.20)
$\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{R}\, \varphi_2 \vee \varphi_3 \,\mathcal{R}\, \varphi_4)$	$\langle\langle A\rangle\rangle[(\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{R}\, \varphi_2) \vee \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{R}\, \varphi_4)) \,\mathcal{R}\, (\varphi_2 \wedge \varphi_4)]$	(3.21)
$\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2 \wedge \varphi_4 \,\mathcal{R}\, \varphi_3)$	$\langle\langle A\rangle\rangle[(\varphi_1 \wedge \varphi_3) \,\mathcal{U}\, ((\varphi_2 \wedge \langle\langle A\rangle\rangle(\varphi_4 \,\mathcal{R}\, \varphi_3)) \vee (\varphi_4 \wedge \varphi_3 \wedge \langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2)))]$	(3.22)
$\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2 \vee \varphi_4 \,\mathcal{R}\, \varphi_3)$	$\langle\langle A\rangle\rangle[(\varphi_1 \wedge \varphi_3) \,\mathcal{W}\, (\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_3) \vee \langle\langle A\rangle\rangle(\varphi_4 \,\mathcal{R}\, \varphi_3)))]$	(3.23)

Proof. (Equation 3.19) Suppose $q \models \langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2 \vee \varphi_3 \,\mathcal{U}\, \varphi_4)$. Then there exists a set of strategies, F_A one for each agent in A, such that $\forall\lambda \in out(q, F_A) \quad \lambda \models \varphi_1 \,\mathcal{U}\, \varphi_2$ or $\lambda \models \varphi_3 \,\mathcal{U}\, \varphi_4$. Take a path $\lambda \in out(q, F_A)$, let i be the first point such that $\lambda[i] \models \varphi_2 \vee \varphi_4$ or $\lambda[i] \not\models \varphi_1 \wedge \varphi_3$ (from the hypothesis, this point *must* exist). There are four cases to consider:

- $\lambda[i] \models \varphi_2$ – we know that $\forall 0 \leq j < i \lambda[j] \models \varphi_1 \wedge \varphi_3$. So $\lambda \models (\varphi_1 \wedge \varphi_3) \,\mathcal{U}\, (\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4))$
- $\lambda[i] \models \varphi_4$ is similar to $\lambda[i] \models \varphi_2$
- $\lambda[i] \not\models \varphi_1 \vee \varphi_2$ – We know that $\forall 0 \leq j < i \lambda[j] \not\models \varphi_2$ and $\lambda[j] \models \varphi_1 \wedge \varphi_3$. Since $\lambda \not\models \varphi_1 \,\mathcal{U}\, \varphi_2$, it follows from our assumption that $\lambda \models \varphi_3 \,\mathcal{U}\, \varphi_4$. Thus, we can construct a strategy G_A such that $\lambda[i] \models \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4)$. $G_A = \{g_a \mid a \in \Sigma\}$ where $g_a(\mu) = f_a(\lambda[0, i-1].\mu)$. So $\lambda \models (\varphi_1 \wedge \varphi_3) \,\mathcal{U}\, (\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4))$
- $\lambda[i] \not\models \varphi_3 \vee \varphi_4$ is similar to $\lambda[i] \not\models \varphi_1 \vee \varphi_2$

Suppose $q \models \langle\langle A\rangle\rangle[(\varphi_1 \wedge \varphi_3) \,\mathcal{U}\, (\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4))]$. Then there exists a set of strategies, F_A one for each agent in A, such that $\forall\lambda \in out(q, F_A)$ $\lambda \models (\varphi_1 \wedge \varphi_3) \,\mathcal{U}\, (\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4))$. We show that a strategy H_A

[2] Simpler special cases for F and G exists, but are omitted from this extended abstract

exists such that $\langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2 \vee \varphi_3 \,\mathcal{U}\, \varphi_4)$. It is constructed from a set of functions h_a as follows,

```
if $\exists i < n \, \forall j < i \quad q_j \models \varphi_1 \wedge \varphi_3 \wedge \neg\varphi_4 \wedge \neg\varphi_4$ and $q_i \not\models \varphi_1 \wedge \varphi_3$
  then We know that $q_i \models \langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2) \vee \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4)$. If $q_i \models \langle\langle A\rangle\rangle(\varphi_1 \,\mathcal{U}\, \varphi_2)$, then let $G_A$ be the set of strategies such that $\forall\lambda \in out(q_i, G_A)\, \lambda \models \varphi_1 \,\mathcal{U}\, \varphi_2$. Define $h_a$ as $h_a(q_0, \ldots, q_i, \ldots, q_n) = g_a(q_i, \ldots, q_n)$. Similarly, if $q_i \models \langle\langle A\rangle\rangle(\varphi_3 \,\mathcal{U}\, \varphi_4)$ call the resulting set of strategies $G'_A$ and define $h_a$ with $h_a(q_0, \ldots, q_i, \ldots, q_n) = g'_a(q_i, \ldots, q_n)$
else $h(q_0, \ldots, q_n) = f_a(q_0, \ldots, q_n)$
```
□

State Formulae In Section 3.2 above, we considered conjunctions and disjunctions of path formulae under an $\langle\langle A\rangle\rangle$. Often, these can be pulled apart into well-formed ATL. However, when state formulae are mixed in, none of the rules apply directly. For example, $\langle\langle A\rangle\rangle(p \wedge \mathrm{F}q \wedge \mathrm{G}r)$ does not match any rule because of the p. Obviously, a state formula under a path quantifier is equivalent to the same formula outside the path quantifier.

$$stateFormula(\varphi) \vdash \langle\langle A\rangle\rangle\varphi \Leftrightarrow \varphi \tag{3.24}$$

More generally, we can pull state formulae out from any boolean combination of path and state formulae by rewriting to disjunctive normal form and applying the following rule:

$$\langle\langle A\rangle\rangle((\varphi_1 \wedge \psi_1) \vee \ldots \vee (\varphi_n \wedge \psi_n)) \equiv \bigvee_{i=1}^{n}\bigvee_{j=1}^{n}\left(\bigwedge_{k=j}^{i}\varphi_k \wedge \langle\langle A\rangle\rangle \bigvee_{k=j}^{i}\psi_k\right) \tag{3.25}$$

4 Approximations

These approximations are applied when no more equivalences can be used on a formula. Again, they are applied left to right and match temporal operators with path quantifiers. The ψs in each rule represent ATL* path formulae. Each approximation produces a strong or a weak bound, which is closer to being in ATL than the original (one nested temporal operator is paired with a path quantifier). Details on how these are used follow in Section 5.

$$\langle\langle A\rangle\rangle\mathrm{F}\psi \Rightarrow \langle\langle A\rangle\rangle\mathrm{F}\exists\psi \qquad \langle\langle A\rangle\rangle\mathrm{G}\psi \Leftarrow \langle\langle A\rangle\rangle\mathrm{G}\forall\psi \tag{4.1}$$

$$\langle\langle A\rangle\rangle\mathrm{F}\psi \Leftarrow \langle\langle A\rangle\rangle\mathrm{F}\langle\langle A\rangle\rangle\psi \qquad \langle\langle A\rangle\rangle\mathrm{G}\psi \Rightarrow \langle\langle A\rangle\rangle\mathrm{G}\langle\langle A\rangle\rangle\psi \tag{4.2}$$

$$\langle\langle A\rangle\rangle\mathrm{FG}\psi \Rightarrow \exists\mathrm{F}\langle\langle A\rangle\rangle\mathrm{G}\psi \qquad \langle\langle A\rangle\rangle\mathrm{GF}\psi \Leftarrow \forall\mathrm{G}\langle\langle A\rangle\rangle\mathrm{F}\psi \tag{4.3}$$

$$\langle\langle A\rangle\rangle(\psi_1 \,\mathcal{U}\, \psi_2) \Rightarrow \langle\langle A\rangle\rangle(\langle\langle A\rangle\rangle\psi_1 \,\mathcal{U}\, \exists\psi_2) \qquad \langle\langle A\rangle\rangle(\psi_1 \,\mathcal{R}\, \psi_2) \Leftarrow \langle\langle A\rangle\rangle(\langle\langle A\rangle\rangle\psi_1 \,\mathcal{R}\, \forall\psi_2) \tag{4.4}$$

$$\langle\langle A\rangle\rangle(\psi_1 \,\mathcal{U}\, \psi_2) \Leftarrow \langle\langle A\rangle\rangle(\forall\psi_1 \,\mathcal{U}\, \langle\langle A\rangle\rangle\psi_2) \qquad \langle\langle A\rangle\rangle(\psi_1 \,\mathcal{R}\, \psi_2) \Rightarrow \langle\langle A\rangle\rangle(\exists\psi_1 \,\mathcal{R}\, \langle\langle A\rangle\rangle\psi_2) \tag{4.5}$$

$$\langle\langle A\rangle\rangle(\psi_1 \vee \psi_2) \Rightarrow \exists\psi_1 \vee \exists\psi_2 \qquad \langle\langle A\rangle\rangle(\psi_1 \wedge \psi_2) \Leftarrow \forall\psi_1 \wedge \forall\psi_2 \tag{4.6}$$

$$\langle\langle A\rangle\rangle(\psi_1 \vee \psi_2) \Leftarrow \langle\langle A\rangle\rangle\psi_1 \vee \langle\langle A\rangle\rangle\psi_2 \qquad \langle\langle A\rangle\rangle(\psi_1 \wedge \psi_2) \Rightarrow \langle\langle A\rangle\rangle\psi_1 \wedge \langle\langle A\rangle\rangle\psi_2 \tag{4.7}$$

5 Re-Writing

The rewrite rules given above provide a framework for translating formulae from ATL* into ATL. The general pattern is to use equivalences as far as possible. Then approximate and repeat by applying equivalences to the strong and weak bounds. Stop when both bounds are well-formed ATL formulae. The rule-set has some important properties:

- Using these rules, any ATL* formula φ may be re-written into a pair of ATL formulae φ_s, φ_w such that $\varphi_s \Rightarrow \varphi \Rightarrow \varphi_w$. This can be proved using Equations 4.1 to 4.7, and structural induction on the syntax of ATL* formulae.
- The rewrites will always terminate. Every rule has a single direction and all but two reduce the number of temporal operators which do not have a matching path quantifier. The exceptions, are the ones concerned with state formulae (Section 3.2). However, they do not increase the number of unmatched temporal operators, and clearly terminate in themselves. The number of nested temporal operators is finite, so the rules must terminate.
- It does not matter in which order the equivalence rewrites are applied. When approximating, it does not matter which approximation is applied. It is only necessary that equivalences are used in preference to approximations.
- The end result is at most two formulae. Although a formula may be split into strong and weak bounds many times, after the first split one of these can be thrown away. If you are already working on a strong bound and have to approximate, then the weak bound of the resulting pair is discarded because it can provide no further information. Similarly, strong bounds are discarded when already working on a weak one.

6 Examples

To adequately measure our technique, it is not enough to just translate some formulae and look at the results. The real use or lack thereof comes from the result of model-checking translated properties against models.

An existing project[3] [5] has tried to ease the difficulty of writing temporal logic specifications. They identify a number of common patterns drawn from a range of application domains and provide these as templates. For example, the property *"p becomes true between q and r"* can be written in LTL as $\mathrm{G}(q \wedge \neg r \rightarrow (\neg r \; \mathcal{W} \; (p \wedge \neg r)))$. These patterns provide a level of complexity which is as deep as hand-written specifications are likely to be, thus provide a realistic setting to test our technique.

[3] `http://www.cis.ksu.edu/santos/spec-patterns/`

6.1 Feature Interaction in a Telephone System

The model for this case study is one developed for a paper on proving Feature Non-Interaction in ATL [4] and as such, had a pre-written MOCHA model. Some of the specifications given in the paper were in ATL*, so they could not be checked at the time. Here, we translate the properties with our method and comment on the results.

The basic system was the Plain Old Telephone System (POTS) – Four phones and an exchange can interact to make calls in the familiar way. Then features were added with a construct described in the paper. For POTS itself, there are some basic properties to check; for the featured system, we examine the Call Forward on Busy feature. The results are summarised in Table 2.

To illustrate the translation process, the derivation of one property is given below. "*The user cannot change the callee without replacing the handset.*" Although the original property was successfully checked with our method, a variant given below gives a better illustration of how the translation works. Instead of using a $\mathcal{W}$ operator, we follow a specification pattern from [5] "Existence between p and r".

$p \equiv$ `i.callee=j`	Renaming
$q \equiv$ `i.trying`	
$r \equiv$ `i.idle`	
$[[i]]\mathrm{G}(p \wedge q \rightarrow (p\,\mathcal{W}\,r))$	Original property from [4]
$[[i]]\mathrm{G}(p \wedge q \wedge \mathrm{F}r \rightarrow (p\,\mathcal{U}\,r))$	Same property, expressed using pattern from [5]
$[[i]]\mathrm{G}(\neg p \vee \neg q \vee \mathrm{G}\neg r \vee (p\,\mathcal{U}\,r))$	Negation Normal Form
$[[i]]\mathrm{G}\forall(\neg p \vee \neg q \vee \mathrm{G}\neg r \vee (p\,\mathcal{U}\,r))$	Approximation using Eq 4.1
$[[i]]\mathrm{G}(\neg p \vee \neg q \vee \forall(\mathrm{G}\neg r \vee (p\,\mathcal{U}\,r)))$	Equivalence using Eq 3.25
$[[i]]\mathrm{G}(\neg p \vee \neg q \vee \forall((p \wedge \neg r)\,\mathcal{W}\,((\forall(p\,\mathcal{U}\,r)) \vee (\forall \mathrm{G}\neg r))))$	Equivalence using Eq 3.23

The unknown result for the third property is a little disappointing, but this is actually an inaccurate specification. It doesn't allow for j putting the phone down whilst the call-forwarding is being resolved. If we add this to the formula, and check a new strong bound:

```
⟨⟨i⟩⟩F⟨⟨i⟩⟩G(j.trying & j.callee=i & !i.idle
        -> A (j.trying U ((j.trying & j.callee=k) | !j.offhook))
```

We find that the property is **true** – Call Forward on Busy has been implemented correctly. The translation method did not help in coming to this conclusion, other than by forcing consideration on why the original strong bound was false.

Table 2. Results of Translating and Model Checking for POTS and POTS+CFB

Any phone may call any other phone (POTS)

Original	$\langle\langle i,j\rangle\rangle$ G F (i.talking & i.callee=j)	**n/a**
Strong	$\forall$ G $\langle\langle i,j\rangle\rangle$ F (i.talking & i.callee=j)	**T**
Conclusion	**Original is true**	

The user cannot change the callee without replacing the handset (POTS)

Original	$[[i]]$ G (i.callee=j & i.trying & F i.idle -> (i.callee=j U i.idle))	**n/a**
Strong	$[[i]]$ G (!i.callee=j | !i.trying | A ((i.callee=j & !i.idle) W ((A (i.callee=j U i.idle)) | (A G !i.idle)))	**T**
Conclusion	**Original is true**	

If user[i] is busy, they can force a call from j to be forwarded to k (POTS+CFB)

Original	$\langle\langle i\rangle\rangle$F G (j.trying & j.callee=i & !i.idle -> j.trying U (j.trying & j.callee=k))	**n/a**
Strong	$\langle\langle i\rangle\rangle$F $\langle\langle i\rangle\rangle$ G (j.trying & j.callee=i & !i.idle -> $\forall$ (j.trying U (j.trying & j.callee=k)))	**F**
Weak	$\langle\langle i\rangle\rangle$F E G (j.trying & j.callee=i & !i.idle -> $\langle\langle i\rangle\rangle$ (j.trying U (j.trying & j.callee=k)))	**T**
Conclusion	**No result**	

7 Conclusions and Related Work

We have demonstrated that Alur and Henzinger's ATL* is a good logic for writing specifications of reactive systems. Given a specification in ATL*, our method produces bounds in ATL which are guaranteed to be correct (i.e. the strong bound implies the original and the weak bound is implied by it). Alternative, ad hoc. simplification may produce formulae which look close to the original but due to the subtleties of ATL are not as close as they appear. To the best of our knowledge, there is no existing work to do the same thing with ATL* but there is much written about the expressivity of linear and branching time logics.

In [3], Clarke et al. show that for any CTL* formula which distinguishes two finite Kripke models, the models may also be distinguished by a CTL formula. Since ATL is a generalisation of CTL, this seems like exactly what we were looking for, but their precondition is actually very strong. The CTL* formula must "sufficiently detailed" i.e. it must characterise exactly one class of equivalent finite Kripke structures. The first notion of equivalence used in [3] is similar to bisimulation and then this is extended to equivalence modulo stuttering. Even with this extension, the resulting CTL* specification must be nearly as detailed as the implementation so it would be just as likely to contain errors and thus defeat the point of model checking it.

In Cadence SMV [10], specifications are written in LTL and then translated to CTL in order to perform symbolic model checking. Direct conversions are used where possible, otherwise new variables are introduced into the model to characterise the parts which cannot be translated. We intend to investigate this idea in the context of ATL.

References

1. R. Alur, T. A. Henzinger, S. C. Krishnan, et al. *Mocha User Manual.* Computer and Information Science Department, University of Pennsylvania and Electrical Engineering and Computer Sciences Department, University of California, Nov. 1999.
2. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.
3. M. Browne, E. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1-2), July 1988.
4. F. Cassez, M. D. Ryan, and P.-Y. Schobbens. Proving feature non-interaction with alternating-time temporal logic. In S. Gilmore and M. D. Ryan, editors, *Language Constructs for Describing Features.* Springer-Verlag, 2000.
5. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering*, May 1999.
6. L. Lamport. “Sometimes” is sometimes “not never” - on the temporal logic of programs. In *Proc. 7th ACM Symposium on Principles of Programming Languages*, pages 174–185, Jan. 1980.
7. L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Proceedings of the IFIP Congress on Information Processing*, pages 657–667. North-Holland, 1983.
8. T. Laureys. From event based semantics to linear temporal logic. Master's thesis, School of Cognitive Science - University of Edinburgh, 2 Buccleuch Place, Edinburgh, UK, 1999.
9. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of 10th International Conference on Computer Aided Verification*, pages 248–263. Springer-Verlag, 2000.
10. SMV. `http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/`.