Agents and Roles: Refinement in Alternating-Time Temporal Logic

Pierre-Yves Schobbens
Institut d'Informatique
Facultés Universitaires de Namur
Rue Grandgagnage 21
5000 Namur, Belgium
www.info.fundp.ac.be/~pys
pys@info.fundp.ac.be

Mark Ryan
School of Computer Science
University of Birmingham
Edgbaston
Birmingham B15 2TT, UK
www.cs.bham.ac.uk/~mdr
M.D.Ryan@cs.bham.ac.uk

April 8, 2001

DRAFT

Abstract

We present a notion of refinement between agent-oriented systems defined using alternating-time temporal logic (ATL). The refinement relation provides a framework for defining *roles* in a society of interacting agents, and formalising a relation of *conformance* between agents and roles. The refinement relation also allows us to construct abstractions in order to make verification more tractable.

1 Introduction

In order to make descriptions of reactive systems more tractable, programming languages and specification languages usually have a modularisation construct which allows the decomposition of the system under consideration into objects (sometimes called agents, to emphasise their autonomy). This horizonal decomposition of a system should be integrated with a vertical decomposition, in which we refine abstract descriptions of objects or agents into more concrete ones during the course of the system development. The abstract descriptions can be thought of as roles, which are fulfilled by the concrete agents.

Verifying that a set of agents fulfills its role must be done in the context of the other agents. This idea has lead to the assume/guarantee paradigm: the specification of an agent guarantees its behaviour will be correct, assuming that its environment (formed in part by other agents) is correct. Our goal is to integrate this idea with a framework for agent refinement.

To formalise these intuitions, we propose to use Alternating-time Temporal Logic [3] (ATL), which allows us to describe precisely the properties of the different agents involved in a system, and the *strategies* they have for achieving their goals. ATL is a branching temporal logic based on game theory. It contains the usual temporal operators (next, always, until) plus cooperation modalities $\langle\langle A\rangle\rangle\phi$, where A is a set of players (also called agents). This modality quantifies over the set of behaviours and means that the agents in A have a collective strategy to enforce ϕ, whatever the choices of the other agents. 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.

We define a very general notion of refinement between alternating-time transition systems (ATS, the models of ATL), which allows splitting or coalescing sets of agents so that a role may be fulfilled by several agents, or more generally several roles fulfilled by several agents. As such, our refinement relation generalises that of [2]. As well as providing a framework for describing agents and the roles they conform to, the refinement relation can be used to construct abstractions that allow us to verify systems efficiently in the manner of Cadence SMV [6].

Outline. In this preliminary version we present our early results. In section 2 we recall the basic concepts of ATL and ATL* and their semantics on ATSs. Section 3 defines refinement and shows some of its properties, and some examples. Finally in section 4 we discuss some directions for future work.

2 Alternating-time temporal logic

Alternating-time temporal logic (ATL) is based on CTL. Let us first recall a few facts about CTL. CTL [4] is a branching-time temporal logic in which we can express properties of reactive systems. For example, properties of cache-coherence protocols [7], telephone systems [8], and communication protocols have been expressed in CTL. One problem with CTL is that it does not distinguish between different sources of non-determinism. In a telephone system, for example, the different sources include individual users, the environment, and internal non-determinism in the telephone exchange. CTL provides the A quantifier to talk about all paths, and the E quantifier to assert the existence of a path. $\mathrm{A}\psi$ means that, no matter how the non-determinism is resolved, ψ will be true of the resulting path. $\mathrm{E}\psi$ asserts that, for at least one way of resolving the non-determinism, ψ will hold. But because CTL does not distinguish between different types of non-determinism, the A quantifier is often too strong, and the E quantifier too weak. For example, if we want to say about a telephone system that *user i can converse with user j*, CTL allows us to write the formulas $A\Diamond talking(i,j)$, $E\Diamond talking(i,j)$ The first one says that in all paths, somewhere along the path there is a state in which i is talking to j, and is clearly much stronger than the intention. The second formula says that there is a path along which i is eventually talking j. This formula is weaker than the intention, because to obtain that path we may have to make choices on behalf of all the components of the system that behave non-deterministically. What we wanted to say is that users i and j can resolve their non-deterministic choices in such a way that, no matter how the other users or the system or the environment behaves, all the resulting paths will eventually have a state in which i is talking j. These subtle differences in expressing the properties we want to check can be captured accurately with ATL.

Alternating-time temporal logic (ATL) [3] generalises CTL by introducing *agents*, which represent different sources of non-determinism. In ATL the A and E path quantifiers are replaced by a path quantifier $\langle\langle A\rangle\rangle$, indexed by a subset A of the set of agents. The formula $\langle\langle A\rangle\rangle\psi$ means that the agents in A can resolve their non-deterministic choices such that, no matter how the other agents resolve their choices, the resulting paths satisfy ψ. We can express the property that user i has the power, or capability, of talking to j by the ATL formula[1]

$$\langle\langle i\rangle\rangle\Diamond\mathtt{talking}(i,j).$$

We read $\langle\langle A\rangle\rangle\psi$ as saying that the agents in A can, by cooperating together, force the system to execute a path satisfying ψ. If A is the empty set of agents, $\langle\langle A\rangle\rangle\psi$ says that the system will execute a path satisfying ψ without the cooperation of any agents at all; in other words, $\langle\langle\emptyset\rangle\rangle\psi$ is equivalent to $\mathrm{A}\psi$ in CTL. Dually, $\langle\langle\Sigma\rangle\rangle\psi$ (where Σ is the entire set of agents) is a weak assertion, saying that if all the agents conspire together they may enforce ψ, which is equivalent to $\mathrm{E}\psi$ in CTL.

2.1 ATL and ATL*

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

$$\phi ::= p \mid \top \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle\langle A\rangle\rangle[\phi_1 \ \mathrm{U}\ \phi_2] \mid \langle\langle A\rangle\rangle\Box\phi_1 \mid \langle\langle A\rangle\rangle\bigcirc\phi_1$$

[1] We write $\langle\langle i\rangle\rangle\psi$ instead of $\langle\langle\{i\}\rangle\rangle\psi$.

where $p \in P$ and $A \subseteq \Sigma$. We use the usual abbreviations for $\rightarrow$, $\wedge$ in terms of $\neg$, $\vee$. The operator $\langle\!\langle\ \rangle\!\rangle$ is a path quantifier, and $\bigcirc$ (*next*), $\Box$ (*always*) and U (*until*) are temporal operators. The logic ATL is similar to the branching-time logic CTL, except that path quantifiers are parameterised by sets of agents. As in CTL, we write $\langle\!\langle A \rangle\!\rangle \Diamond \phi$ for $\langle\!\langle A \rangle\!\rangle [\top \ \mathrm{U}\ \phi]$, and we define the weak-until: $\langle\!\langle A \rangle\!\rangle [\phi \ \mathrm{W}\ \psi] = \neg [[A]] [\neg \psi \ \mathrm{U}\ (\neg \phi \wedge \neg \psi)]$.

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). The formulas $[[A]]\Diamond\phi$, $[[A]]\Box\phi$, and $[[A]]\bigcirc\phi$ stand for $\neg\langle\!\langle A \rangle\!\rangle \Box \neg \phi$, $\neg\langle\!\langle A \rangle\!\rangle \Diamond \neg \phi$, and $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \neg \phi$.

The CTL path quantifiers A (all paths) and E (some path) can be recovered in ATL as $\langle\!\langle \emptyset \rangle\!\rangle$ and $\langle\!\langle \Sigma \rangle\!\rangle$. 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.

For a subset $A \subseteq \Sigma$ of agents, the fragment $\langle\!\langle A \rangle\!\rangle$-ATL of ATL consists of ATL formulas whose only modality is $\langle\!\langle A \rangle\!\rangle$, and that does not occur within the scope of a negation. The $\langle\!\langle A \rangle\!\rangle$-ATL* fragment of ATL* is defined similarly.

2.2 Alternating transitions systems

Whereas the semantics of CTL is given in terms of transition systems, the semantics of ATL is given in terms of *alternating transition systems* (ATSs). An ATS S over a set of atomic propositions P and a set of agents Σ is composed of (Q, π, δ, I, o), where Q is a set of states and $\pi : Q \rightarrow 2^P$ maps each state to the set of propositions that are true in it, and

$$\delta : Q \times \Sigma \rightarrow 2^{2^Q}$$

is a transition function which maps a state and an agent to a non-empty set of *choices*, where each choice is a non-empty set of possible next states. If the system is in a state q, each agent a chooses a set $Q_a \in \delta(q, a)$; the system will move to a state which is in $\bigcap_{a \in \Sigma} Q_a$. We require that the system is non-blocking and that the agents together choose a unique next state; that is, for every q and every tuple $(Q_a)_{a \in \Sigma}$ of choices $Q_a \in \delta(q, a)$, we require that $\bigcap_{a \in \Sigma} Q_a$ is a singleton. Similarly, the initial state is specified by $I : \Sigma \rightarrow 2^{2^Q}$. I maps each agent to a set of choices. The agents together choose a single initial state: for each tuple $(Q_a)_{a \in \Sigma}$ of choices $Q_a \in I(a)$, we require that $\bigcap_{a \in \Sigma} Q_a$ is a singleton.

In the sequel, we will consider ATS of a particular form: the choices of the agents are expressed by choosing the value of their variables. This is also the choice made in the language Mocha [1]. We therefore stipulate a function $o : P \rightarrow \Sigma$ that, for each propositional variable, gives its owner. We define $P_a = \{p \in P \mid o(p) = a\}$ to designate the propositional variables belonging to agent a, and $P_A = \{p \in P \mid o(p) \in A\}$ to a group A. The choices of an agent thus determine the value of its variables, and let the other vary freely:

$$\forall a \in \Sigma, q \in Q, Q_a \in \delta(q, a), \forall X \subseteq P_{\neq a}, \exists q \in Q_a : \pi(q) \cap P_{\neq a} = X$$

where $P_{\neq a} = \{p \in P | o(p) \neq a\}$. A signature is defined as this function o, together with its domain P and codomain Σ.

For two states q and q', we say that q' is a *successor* of q if, for each $a \in \Sigma$, there exists $Q' \in \delta(q, a)$ such that $q' \in Q'$. We write $\delta(q)$ for the set of successors of q; thus,

$$\delta(q) = \bigcap_{a \in \Sigma} \ \bigcup_{Q \in \delta(q,a)} Q$$

A computation of S is an infinite sequence $\lambda = q_0, q_1, q_2 \ldots$ of states such that (for each i) q_{i+1} is a successor of q_i. We write $\lambda[0, i]$ for the finite prefix $q_0, q_1, q_2, \ldots, q_i$.

Often, we are interested in the cooperation of a subset $A \subseteq \Sigma$ of agents. Given A, we define $\delta(q, A) = \{\bigcap_{a \in A} Q_a \mid Q_a \in \delta(q, a)\}$. Intuitively, when the system is in state q, the agents in A can choose a set $T \in \delta(q, A)$ such that, no matter what the other agents do, the next state of the system is in T. Note that $\delta(q, \{a\})$ is just $\delta(q, a)$, and $\delta(q, \Sigma)$ is the set of singleton successors of q.

Example 1 ([3]) Consider a system with two agents "user" u and "telephone exchange" e. The user may lift the handset, represented as assigning value true to the boolean variable "offhook". The exchange may then send a tone, represented by assigning value true to the boolean variable "tone". Initially, both variables are false. Clearly, obtaining a tone requires collaboration of both agents.

We model this as an ATS $S = (Q, \pi, \delta, I)$ over the agents $\Sigma = \{u, e\}$ and propositions $P = \{\text{offhook,tone}\}$. Let $Q = \{00, 01, 10, 11\}$. 00 is the state in which both are false, 01 the state in which "offhook" is false and "tone" is true, etc. (thus, $\pi(00) = \emptyset$, $\pi(01) = \{tone\}$, etc.). The transition function δ and initial states I are as indicated in the figure.

$\delta(q,a)$	u	e
00	$\{\{00,01\},\{10,11\}\}$	$\{\{00,10\}\}$
10	$\{\{10,11\}\}$	$\{\{00,10\},\{01,11\}\}$
01	$\{\{00,01\},\{10,11\}\}$	$\{\{01,11\}\}$
11	$\{\{10,11\}\}$	$\{\{01,11\}\}$
I	$\{\{00,01\}\}$	$\{\{00,10\}\}$

Figure 1: The transition function of the ATS.

2.3 Semantics

The semantics of ATL uses the notion of *strategy.* A *strategy* for an agent $a \in \Sigma$ is a mapping $f_a : Q^+ \to 2^Q$ such that $f_a(\lambda \cdot q) \in \delta(q, a)$ with $\lambda \in Q^*$. In other words, the strategy is a recipe for a to make its choices. Given a state q, a set A of agents, and a family $F_A = \{f_a \mid a \in A\}$ of strategies, the *outcomes* of F_A from q are the set $out(q, F_A)$ of all computations from q where agents in A follow their strategies, that is,

$$out(q_0, F_A) = \{\lambda = q_0, q_1, q_2, \cdots \mid \forall i,\ q_{i+1} \in \delta(q_i) \cap \big(\bigcap_{a\in A} f_a(\lambda[0,i])\big)\}.$$

If $A = \emptyset$, then $out(q, F_A)$ is the set of all computations, while if $A = \Sigma$ then it consists of precisely one computation.

The semantics of ATL* is as CTL*, with the addition of:

- $q \models \langle\langle A\rangle\rangle\psi$ if there exists a set F_A of strategies, one for each agent in A, such that for all computations $\lambda \in out(q, F_A)$ we have $\lambda \models \psi$.

Remark 2 To help understand the ideas of ATL, we state below some validities, and more surprising non-validities.

1. If $A \subseteq B$, then $\langle\langle A\rangle\rangle\psi \to \langle\langle B\rangle\rangle\psi$, and $[[B]]\psi \to [[A]]\psi$. Intuitively, anything that A can enforce can also be enforced by a superset B; and if anything that B is powerless to prevent cannot be prevented by a subset of B.

2. In CTL*, A distributes over $\wedge$. But in general in ATL*, $\langle\langle A\rangle\rangle(\psi_1 \wedge \psi_2)$ only implies $(\langle\langle A\rangle\rangle\psi_1) \wedge (\langle\langle A\rangle\rangle\psi_2)$. The first formula asserts that A can enforce $\psi_1 \wedge \psi_2$, while the second is weaker, asserting that A has a way to enforce ψ_1 and another, possibly incompatible, way to enforce ψ_2. Similarly, $\langle\langle A\rangle\rangle(\psi_1 \vee \psi_2)$ and $\langle\langle A\rangle\rangle\psi_1 \vee \langle\langle A\rangle\rangle\psi_2$ are different (for $A \neq \Sigma$). The first one asserts that A can enforce $\psi_1 \vee \psi_2$, but which of the two is true might be chosen by others. This is weaker than the second formula, which asserts that A can guarantee ψ_1, or A can guarantee ψ_2, but nobody can choose which. The strongest variant where A can choose, is expressed as: $\langle\langle A\rangle\rangle(\psi_1 \wedge \neg\psi_2) \wedge \langle\langle A\rangle\rangle(\psi_2 \wedge \neg\psi_1)$.

3. The cooperation operator can often be repeated inside a temporal operator, changing slightly the meaning of the formula, for instance: $\langle\langle A\rangle\rangle\Box\Diamond\phi \to \langle\langle A\rangle\rangle\Box\langle\langle A\rangle\rangle\Diamond\phi$. Here, this is because

the strategies F_A that A use in the outer modality may be adapted for the inner modality, by shifting its time: each $f'_a(x)$ is simply $f_a(\lambda \cdot x)$, where λ is the path linking the points of evaluation of the two modalities. (Note the CTL* validities $\mathrm{E}\Box\Diamond\phi \rightarrow \mathrm{E}\Box\mathrm{E}\Diamond\phi$ and $\mathrm{A}\Box\Diamond\phi \leftrightarrow \mathrm{A}\Box\mathrm{A}\Diamond\phi$.)

3 Refinement of agents

In this section we introduce refinement between ATSs. Our aim is to present a framework which provides

- A method for constructing systems by assembling components. The system is specified as a collection of roles, and the component agents are shown to conform to the roles.
- A method for verifying agents efficiently, by abstracting their environment (similar to [6], but generalising from LTL to ATL). An agent is verified under the guarantees made by the roles of its environment. This addresses the state explosion problem, since it is simpler to verify against the roles of the environment than against the agents implementing the roles.

Example 3 The sliding-window protocol (SWP) is a communications protocol designed to guarantee communication over an unreliable channel. The channel may loose, duplicate, and re-order messages. We assume, however, that the channel is not completely unreasonable; thus,

- Although it might loose some messages, it does not loose all of them. More precisely, if a given message is sent often enough, it will eventually get through.
- Although it might re-order messages, the degree to which it re-orders is bounded. The channel will not swap a given message with other messages more than N times.

The agents involved in the abstract specification of the SWP are depicted in Figure 2. Here is a

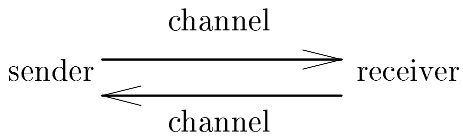

Figure 2: Roles for the sliding-window protocol

very abstract account of the SWP. The message to be sent is split into packets, and the packets are passed in order to the sender. In order to cope with the deficiencies of the channels, the sender may send individual packets multiple times, until an acknowledgment is received from the receiver. Since the acknowledgments are also sent along an unreliable channel, they might need to be sent several times too. In order to cope with re-ordering, the sender labels messages with a number in the set $\{0, \ldots, M\}$, where $M > N$. When all the numbers are used up, the sender starts again from 0. The sender maintains an interval $[i, i+w]$, known as its window, of size $w < N$. Initially $i = 0$. The sender non-deterministically picks a packet whose sequence number is in the window, and sends it. It records any acknowledgments it receives; when it has received an acknowledgment for the ith message, it may advance its window by incrementing i. The receiver, whose job is to assemble the incoming message and send out acknowledgments, may be described in similar abstract terms.

This account is highly non-deterministic; it does not specify what order the sender should send the packets in its window, or what order the receiver should send the acknowledgments; this is inefficient. It allows the sender to resend messages which have already been acknowledged, which is wasteful. A more concrete specification of the SWP is given in [5], where the sender is split into three processes: the *trans*, which sends the packets initially and in order; the *ack-recv*, which records which messages have been acknowledged, and the *re-trans*, which re-transmits packets for which an acknowledgment has not been received before a given timeout. The reciever is also split into several components.

Trans, ack-rcvr, and re-trans jointly play the role of sender, and accept and ack-send jointly play the role of receiver (Figure 3).

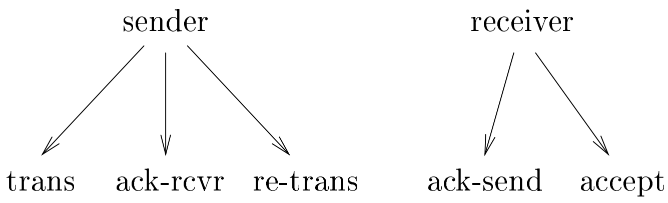

Figure 3: Agents and Roles for the sliding-window protocol

When proving properties of the concrete agents, we assume only the guarantees made by the roles of the other agents. This is a weaker assumption than the guarantees actually provided by the other agents, but it is leads to more efficient verification. Indeed, in a multi-layered example, while proving the properties of one agent we would assume the guarantees of the other agents in the most abstract layer possible.

This example helps us motivate the definition of ATS refinement: a set of agents is refined, while the complement set of agents (forming the refinement) may be abstracted.

The formal definition of refinement is split into two parts. Signature refinement deals with the symbols in the vocabulary. Next, ATS refinement extends a signature refinement, and deals with behaviours.

Definition 4 Let Σ and Σ' be disjoint sets of agents with variables P and P' respectively. A *signature refinement from* $A \subseteq \Sigma$ *to* $A' \subseteq \Sigma'$ is: a relation $f \subseteq \Sigma \times \Sigma'$ between agents in both societies, and a relation $g \subseteq P \times P'$, such that:

- f links agents in A to agents in A', and agents outside A to agents outside A'. To define this formally, we extend f to sets of agents in the usual way: $f(B) = \{a' \mid \exists a \in B f(a, a')\}$ and $f^{-1}(B') = \{a \mid \exists a' \in B' f(a, a')\}$. A set of agents $B \in \Sigma$ *corresponds to* $B' \in \Sigma'$ iff $B' = f(B)$ and $f^{-1}(B') = B$. Note that some sets of agents will not have a correspondent. We require that A corresponds to A' and $\bar{A}$ to $\bar{A}'$ (where $\bar{A} = \Sigma \setminus A$ and $\bar{A}' = \Sigma' \setminus A'$).
- g respects ownership: $g(p, p')$ implies $f(o(p), o(p'))$. Additionally, the more detailed description of the component should at least include the variables required in its role, thus g is a function from $P_A = \{p \mid o(p) \in A\}$, and conversely, g^{-1} is a function on $P_{\bar{A}'}$.

Example 5 $A = \{\text{sender}\} \subseteq \Sigma = \{\text{sender}, \text{receiver}\}$. $A' = \{\text{trans, ack-rcvr, re-trans}\} \subseteq \Sigma' = \{\text{trans, ack-rcvr, re-trans, receiver}\}$. This refines the sender. In this case, f is shown as the left three arrows of figure 3, together with the pair (receiver, receiver).

Signature refinements have the following properties:

Theorem 6 1. *The identity is a signature refinement;*

2. *Signature refinements compose: if* (f_1, g_1) *is a signature refinement from* $A_1 \subseteq \Sigma_1$ *to* $A_2 \subseteq \Sigma_2$*, and* (f_2, g_2) *is a signature refinement from* $A_2 \subseteq \Sigma_2$ *to* $A_3 \subseteq \Sigma_3$*, then* $(f_1 \circ f_2, g_1 \circ g_2)$ *is a signature refinement from* $A_1 \subseteq \Sigma_1$ *to* $A_3 \subseteq \Sigma_3$*.*

Given such a signature refinement, we can now consider how to translate formulae. Not all formulae can be translated: they may involve agents or variables that have no correspondent. The translation T along f, g of a formula is thus: $T(p) = g(p)$ assuming g is functional on p;

$T(\langle\!\langle A\rangle\!\rangle\phi) = \langle\!\langle f(A)\rangle\!\rangle T(\phi)$ assuming $f(A)$ corresponds to A; and the other logical operators translate unchanged. If the assumptions above are not fulfilled, the formula cannot be translated.

A refinement between signatures can sometimes be extended to a refinement between ATS $S = (\Sigma, Q, \pi, \delta, I)$ and $S' = (\Sigma', Q', \pi', \delta', I')$. We then say that the agents A' are conformant to the role A.

Definition 7 An *ATS refinement* r extending a signature refinement $s = (f, g)$ from $A \subseteq \Sigma$ to $A' \subseteq \Sigma'$is a triple (H, C, D), where:

1. $H \subset Q \times Q'$ is a relation between states such that corresponding variables have the same value, formally: if $H(q, q'), g(v, v')$, then $v \in \pi(q)$ iff $v' \in \pi'(q')$),
2. $C : H \to \delta'(q', A') \to \delta(q, A)$: C gives for any pair (q, q') in the relation H and for any choice possible for the component A' in q', a corresponding choice for the role A in q
3. conversely $D : H \to \delta(q, \bar{A}) \to \delta'(q', \bar{A}')$,

such that $\forall(q, q') \in H, \forall T \in \delta(q, \bar{A}), \forall R' \in \delta'(q', A') : (T \cap C(q, q')(R')) \times (D(q, q')(T) \cap R') \subseteq H$.

H relates states in the abstract system with those of the refined system. The intuition for C and D is this: any choices A' makes in the refined system must correspond to choices A makes in the original system. C maps A''s choices to A's choices. The environment goes the other way; in S' it is more abstract, so D maps $\bar{A}$'s choices to $\bar{A}'$'s choices. The 'such that' part of the definition guarantees that, after these choices have been resolved, the two systems are again in H-related states. Note that since T and $C(q, q')(R)$ together define the choices of all agents, their intersection is a singleton, and similarly for $D(q, q')(T) \cap R$.

If the agents A of the abstract system S are viewed as a role, then this definition ensures that the agents A' of S' conform to that role. Note that again the notion of conformance goes in opposite directions for the system and for its environment.

Note that this definition is similar to the definition of alternating refinement in [2]. Our definition is more general than theirs, since it allows the agents and the variables to be different in the two systems. Our definition implies their definition:

Remark 8 If $r = (f, g, H, C, D)$ is an ATS refinement from $S = (\Sigma, Q, \pi, \delta, I)$ to $S' = (\Sigma', Q', \pi', \delta', I')$ and $\Sigma = \Sigma'$, $P = P'$, and f, g are identity, then $S \leq_{\bar{A}} S'$ in the sense of [2].

Proof The definition of $S \leq_{\bar{A}} S'$ is that there exists $H \subseteq Q \times Q'$ such that $H(q, q')$ implies $\pi(q) = \pi'(q')$ and $\forall T \in \delta(q, \bar{A})\, \exists T' \in \delta'(q', \bar{A})\, \forall R' \in \delta'(q', A)\, \exists R \in \delta(q, A)\, (T \cap R) \times (T' \cap R') \subseteq H$. As R may depend on T as well as R' in this definition, while $C(q, q')(R')$ depends on R' but not on T in ours, their definition is weaker.

Now ATS refinements have the properties:

Theorem 9 *1. The identity is an ATS refinement;*

2. ATS refinements compose vertically: if $r_1 = (f_1, g_1, H_1, C_1, D_1)$ is a ATS $S_1 \to S_2$ refinement from $A_1 \subseteq \Sigma_1$ to $A_2 \subseteq \Sigma_2$, and $r_2 = (f_2, g_2, H_2, C_2, D_2)$ is a $S_2 \to S_3$ refinement from $A_2 \subseteq \Sigma_2$ to $A_3 \subseteq \Sigma_3$, then there is a $S_1 \to S_3$ refinement r_3 from A_1 to A_3, which may also be written $r_1; r_2$, given by $(f_1 \circ f_2, g_1 \circ g_2, H_1 \circ H_2, C_1 \circ C_2, D_1 \circ D_2)$, where $\circ$ is the relation composition: $H_1 \circ H_2 = \{(q_1, q_3) \mid \exists q_2 (q_1, q_2) \in H_1 \wedge (q_2, q_3) \in H_2\}$.

Theorem 10 *If there is an ATS refinement from $A \subseteq \Sigma$ to $A' \subseteq \Sigma'$, then any translatable $\langle\!\langle A'\rangle\!\rangle, [[\bar{A}']]$-ATL formula valid on the ATS S' is also valid on S when translated; conversely, any translatable $\langle\!\langle \bar{A}\rangle\!\rangle, [[A]]$-ATL formula valid on S is also valid on S' when translated.*

Proof Use remark 8 and theorem 6 of [2].

Horizontal composition of refinements We use the refinement from $A \subseteq \Sigma$ to $A' \subseteq \Sigma'$ when we want to prove that the agents in A' satisfy the role A. The agents in A' are more *concrete* than those in A. As already noted, the definition of refinement means that the agents in $\bar{A}'$ may be more *abstract* than those in $\bar{A}$.

Now suppose we have two refinements, $r_1 = (f_1, g_1, H_1, C_1, D_1)$ and $r_2 = (f_2, g_2, H_2, C_2, D_2)$, such that r_1 is an $S \to S_1$ refinement from $A \subseteq \Sigma$ to $A_1 \subseteq \Sigma_1$, and a $S \to S_2$ refinement from $B \subseteq \Sigma$ to $B_2 \subseteq \Sigma_2$. Refinement r_1 refines the agents in A, and r_2 refines those in B. We would like to be able to put these refinements together, to form a refinement on $A \cup B$. This would allow us to implement the roles A and B independently, and verify the implementations independently. Putting the refinements together means that we obtain a correct refinement of $A \cup B$ with no additional verification. Recall that r_1 refines A, but may abstract B; and r_2, which refines B, may abstract A. Therefore, the composition of r_1 and r_2, noted $r_1 \| r_2$ in Figure 4, has to combine the refinement of A by r_1 and the refinement of B by r_2. We stipulate that $A \cap B$ is empty, in order to avoid the possibility that these refinements are incompatible on their common part.

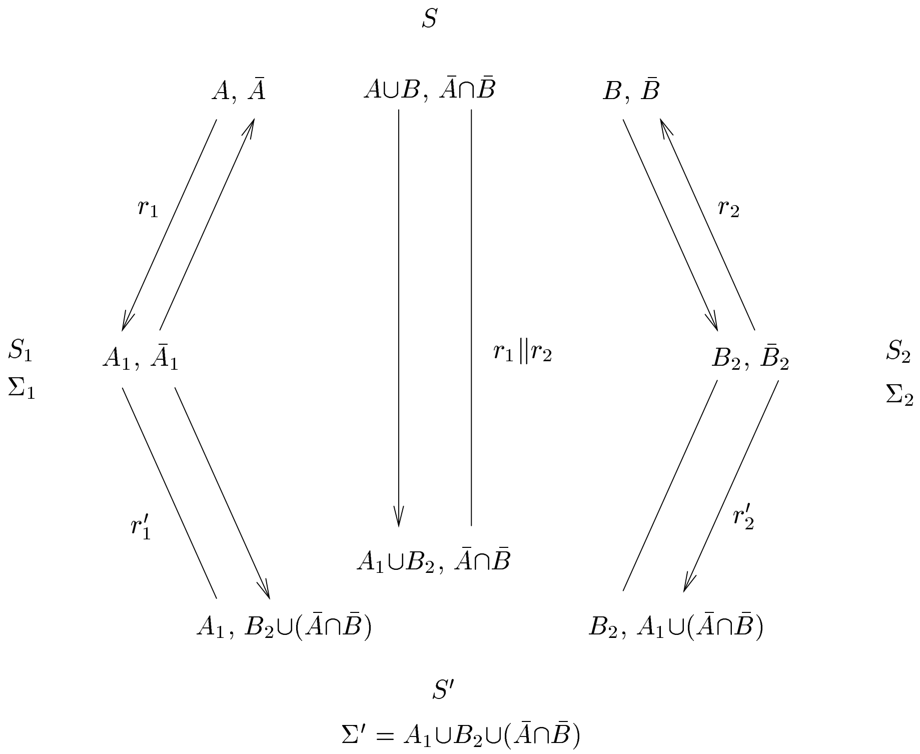

Figure 4: Horizontal composition of refinements.

Example 11 (continued) Let S be the abstract description of the sliding-window protocol (SWP), having agents $\Sigma = \{$sender, receiver, channel1, channel2$\}$. We decide to implement the sender and the receiver separately (Figure 5). The set $A = \{$sender$\}$ is implemented as $A_1 = \{$trans, ack-rcvr, re-trans$\}$. In this example, $\bar{A}_1 = \bar{A} = \{$channel1, channel2, receiver$\}$. We check that this implementation, S_1, is a refinement of S from A to A_1.

The receiver is also implemented. $B_2 = \{$receiver$\}$ is refined into $B_2' = \{$accept, ack-send$\}$.

In the general framework of Figure 4 and Theorem 12 below, $\bar{A}_1$ is allowed to be an abstraction of $\bar{A}$. This is useful for model checing, because S_1 will be more abstract and so checking the refinement will be computationally cheaper.

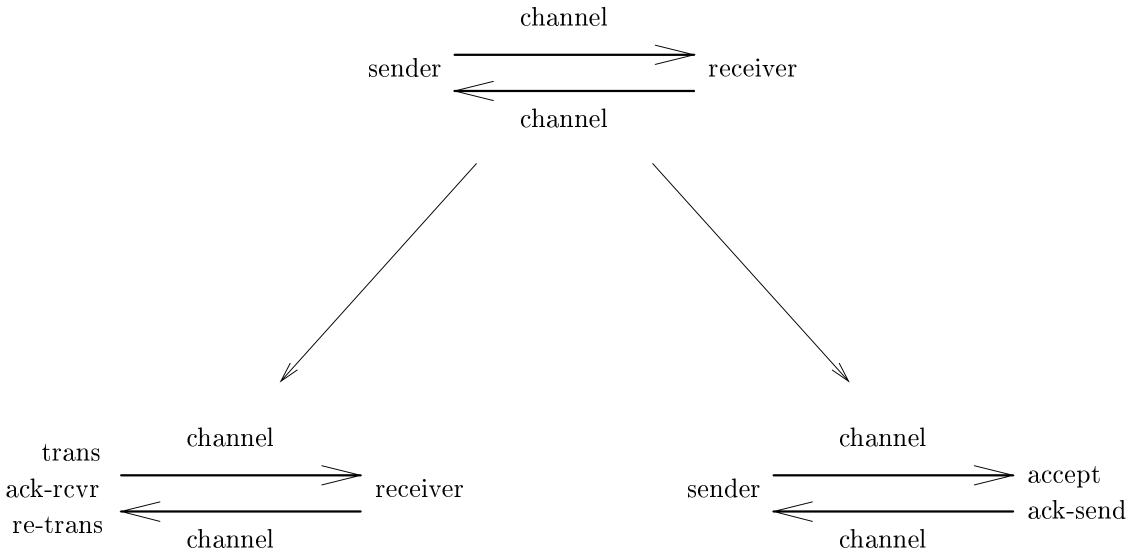

Figure 5: Independently implementing the sender and the receiver.

Theorem 12 *ATS refinements compose horizontally: if $r_1 = (f_1, g_1, H_1, C_1, D_1)$ is an $S \to S_1$ refinement from $A \subseteq \Sigma$ to $A_1 \subseteq \Sigma_1$, and $r_2 = (f_2, g_2, H_2, C_2, D_2)$ is an $S \to S_2$ refinement from $B \subseteq \Sigma$ to $B_2 \subseteq \Sigma_2$ such that $A \cap B = \emptyset$, then we can build r, also noted $r_1 \| r_2$, an $S \to S'$ refinement from $A \cup B \subseteq \Sigma$ to $A_1 \cup B_2 \subseteq \Sigma'$, such that there are refinements:*

1. *r_1', an $S_1 \to S'$ refinement from $\bar{A}_1 \subseteq \Sigma_1$ to $B_2 \cup (\bar{A} \cap \bar{B}) \subseteq \Sigma'$, and*
2. *r_2', an $S_2 \to S'$ refinement from $\bar{B}_2 \subseteq \Sigma_2$ to $A_1 \cup (\bar{A} \cap \bar{B}) \subseteq \Sigma'$,*

and any other such $S \to S'$ refinement r' from $A \cup B$ to C can be decomposed through r: there is r'' such that $r' = r; r''$. The situation is shown in figure 4.

Example 13 (continued) This shows that the two implementations can be put together, and that the result refines the original specification. Moreover, we do not have to check this (which would be computationally expensive, because of the size of S'); we simply check the refinements r_1 and r_2, and the theorem guarantees the refinement $r_1 \| r_2$.

Proof We construct $S' = (\Sigma', Q', \pi', \delta', I')$ and the $S \to S'$ refinement $r = (f, g, H, C, D)$:

- $\Sigma' = A_1 \cup B_2 \cup (\bar{A} \cap \bar{B})$
- $Q' = \{(q, q_1, q_2) \mid H_1(q_1, q) \wedge H_2(q_2, q)\}$
- $v \in \pi'(q, q_1, q_2)$ if $o(v) \in A_1$ and $v \in \pi_1(q_1)$; or $o(v) \in B_2$ and $v \in \pi_2(q_2)$; or $o(v) \in \bar{A} \cap \bar{B}$ and $v \in \pi(q)$.
- $\delta'(a, (q, q_1, q_2)) =$
$$\begin{cases} \{C_1(q_1, q)(T_1) \times T_1 \times D_2(q, q_2)(C_1(q_1, q)(T_1)) \cap Q' \mid T_1 \in \delta_1(a, q_1)\} & \text{if } a \in A_1 \\ \ldots \text{(similar)} & \text{if } a \in B_2 \\ \{T \times D_1(q, q_1)(T) \times D_2(q, q_2)(T) \cap Q' \mid T \in \delta(q, a)\} & \text{if } a \in \bar{A} \cap \bar{B} \end{cases}$$
- $f(a, a')$ if $a \in A$ and $f_1(a, a')$; or $a \in B$ and $f_2(a, a')$; or $a \in \bar{A} \cap \bar{B}$ and $a = a'$.

- $g(v, v')$ if $o(v) \in A$ and $g_1(v, v')$; or $o(v) \in B$ and $g_2(v, v')$; or $o(v) \in \bar{A} \cap \bar{B}$ and $v = v'$.
- $H(q', (q, q_1, q_2)) \Leftrightarrow q = q'$
- $C(q, (q, q_1, q_2))(T \times T_1 \times T_2) = T$
- $D(q, (q, q_1, q_2))(T) = T \times D_1(q, q_1)(T) \times D_2(q, q_2)(T)$

It is straightforward to prove that S' is an ATS (the main part of which is to show that $\delta'(\Sigma', q)$ is a singleton). We prove that $r : S \to S'$ is a refinement:

1. f links $A \cup B$ to some $C' \in \Sigma'$; in fact $C' = A_1 \cup B_2$.
2. g respects ownership: if $g(v, v')$ then, for instance, $o(v) \in A$ and $g_1(v, v')$; the second conjunct implies $f_1(o(v), o(v'))$, which together with $o(v) \in A$ implies $f(o(v), o(v'))$
3. g is functional on $P_{A \cup B}$, since g_1 is functional on P_A and g_2 functional on P_B.
4. We show $H(q', (q, q_1, q_2))$ and $g(v, v')$ imply $v \in \pi(q')$ iff $v' \in \pi'(q, q_1, q_2)$. Since $H(q', (q, q_1, q_2))$, $q = q'$. Now we distinguish between the usual three cases: $o(v)$ may be in A, B, or $\bar{A} \cap \bar{B}$. For example, if $o(v) \in A$ then $g(v, v')$ implies $g_1(v, v')$, therefore $o'(v') \in A_1$. Since $(q, q_1, q_2) \in Q'$, $H_1(q, q_1)$. Therefore, $v \in \pi(q')$ iff $v' \in \pi_1(q_1)$. The other cases are similar.
5. Finally, we need to show that $H(q, (q, q_1, q_2))$, $T \in \delta(q, A \cup B)$, $R \in \delta'((q, q_1, q_2), \bar{A} \cap \bar{B})$: $(T \cap C(q, (q, q_1, q_2))(R)) \times (D(q, (q, q_1, q_2))(T) \cap R) \subseteq H$. Note that $T = \bigcap_{a \in A \cup B} Q_a$ for some choices $Q_a \in \delta(q, a)$, $a \in A \cup B$, and $R = \bigcap_{a \in \bar{A} \cap \bar{B}} (Q_a \times D_1(q, q_1)(Q_a) \times D_2(q, q_2)(Q_a)) \cap Q'$, for again some choices $Q_a \in \delta(q, a)$, $a \in \bar{A} \cap \bar{B}$. Let us fix these choices of Q_a, $a \in \Sigma'$. Suppose $(s, (s', s_1, s_2) \in (T \cap C(q, (q, q_1, q_2))(R)) \times (D(q, (q, q_1, q_2))(T) \cap R)$. Then $\{s\} = \bigcap_{a \in \Sigma'} Q_a$ and $\{(s', s_1, s_2)\} = \bigcap_{a \in \Sigma'} (Q_a \times D_1(q, q_1)(Q_a) \times D_2(q, q_2)(Q_a)) \cap Q'$. Therefore, $s = s'$.

The refinement $r'_1 : S_1 \to S'$ is given by

- $f'_1 \subseteq \Sigma' \times \Sigma_1$ given by: $f'_1(a', a_1)$ iff $a_1 = a' \in A_1$; or $a_1 \notin A_1$ and $a' \in B_2$ and $(a_1, a') \in f_2 \circ f_1$; or $a_1 \notin A_1$ and $a' \in \bar{A} \cap \bar{B}$ and $(a_1, a') \in f_1$.
- $H'_1(q'_1, (q, q_1, q_2)) \Leftrightarrow q_1 = q'_1$
- $C'_1(q_1, (q, q_1, q_2))(T_1) = C_1(q_1, q)(T_1) \times T_1 \times D_2(q, q_2)(C_1(q_1, q)(T_1))$
- $D'_1(q_1, (q, q_1, q_2))(T \times T_1 \times T_2) = T_1$.

The refinement $r'_2 : S' \to S_2$ is defined similarly, and we must again check that r_1 and r_2 are a refinement (omitted).

Finally, we show that any other such $S \to S'$ refinement r' from $A \cup B$ to C can be decomposed through r: there is r'' such that $r' = r; r''$: . . .

4 Conclusions

Our refinement framework allows the system developer to implement various components of the system independently. The results can be put together automatically to form an implementation of the whole. This guaranteed interoperability: in the example, any implementation of the sender is compatible with any implementation of the receiver.

This paper is preliminary. Showing that the roles A and B may be refined independently is an important step in building a refinement calculus suitable for making abstractions to address the state explosion problem in ATL model checking. We intend to continue this work by efficiently computing refinements and developing examples.

References

[1] R. Alur, H. Anand, R. Grosu, F. Ivancic, M. Kang, M. McDougall, B.-Y. Wang, L. de Alfaro, T. Henzinger, B. Horowitz, R. Majumdar, F. Mang, C. Meyer, M. Minea, S. Qadeer, S. Rajamani, and J.-F. Raskin. *Mocha User Manual.* University of California, Berkeley. `www.eecs.berkeley.edu/~mocha`.

[2] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In D. Sangiorgi and R. de Simone, editors, *CONCUR 98: Concurrency Theory*, Lecture Notes in Computer Science 1466, pages 163–178. Springer-Verlag, 1998.

[3] 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.

[4] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, editor, *Logic of Programs Workshop*, number 131 in LNCS. Springer Verlag, 1981.

[5] G. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1990.

[6] K. McMillan. The SMV language. Available from www-cad.eecs.berkeley.edu/~kenmcmil, June 1998.

[7] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[8] M. C. Plath and M. D. Ryan. Feature integration using a feature construct. *Science of Computer Programming*, 2001. In print.