The Epistemics of Encryption

Annette Bleeker Jan van Eijck

Abstract

Message passing by means of public key encryption is described in terms of doxastic dynamic logic. A secret message from a to b can have as effect that b learns something new from a, but it can also cause a change in the real world, when the contents of the message forces b to cease trusting a. As a tool for analysing secret message passing (and much else besides) we develop a framework that allows changes of states conditioned by beliefs about those states.

1 Introduction

We present a model for describing how changes in the world get reflected in multi-agent doxastic settings. One of the applications is the description of the results of message passing, where messages can be public announcements, limited access announcements, or secret communications. Message exchanges can result in belief updates, but also in changes of attitude towards the sender. Our model can also deal with adjustments of agents to a changing reality, collaborations between agents to achieve common goals, and much more besides.

In Section 2, we start with a short introduction to different applications of cryptographic functions, followed by a discussion on how to interpret the reception of a message. In Section 3 we introduce a language of message passing, which we can use to describe the cryptographic communication between agents that we are interested in. We also discuss the role of trust between agents in the different interpretations of a message. In Sections 4 until 8 we give the formal language of epistemic actions and change (which is an extension of the language of message passing) and its semantics, together with examples. We also analyze several forms of equivalences of actions, which may give a better understanding of actions, and which allow us to abstract from a particular description of an action, as we are ultimately interested in its result. In Section 9, we return to the case of message passing, and give our interpretation of the several possible forms of messages, described in the introduced formal language. Finally in Section 10 we state several sound reasoning principles in our framework.

We assume familiarity with a Kripke-style semantics for epistemic logic, and we will apply an analogous semantics for action expressions, building on the work of Baltag et al. [1, 2]. A general setting for our research is given in [6]. Our proposal is in the tradition of dynamic epistemic logic; see Gerbrandy and Groeneveld [7] for an epistemic logic of information updates in terms of non-wellfounded set theory. In Moss 1999 it was pointed out that there is no need for phrasing logics of announcements in terms of hypersets, but that Kripke models will do nicely [11]. Baltag, Moss, and Solecki [2] show how to do epistemic updates with standard Kripke models. Epistemic perspectives on change in the real world are still very much the province of philosophy: see, e.g., Von Wright [15], but also Shoham [14].

2 Using Encryption for Secrecy and Authentication

In this section we discuss what cryptographic methods can be used in communication, and in what way they ensure secrecy or authentication.

Encryption Methods To start with, here is an outline of *private key encryption.* In this method, key K is usually shared by (i.e. only known to) two (or more) agents. The encryption function

$$\mathsf{encr} :: \mathsf{Key} \times \mathsf{Message} \longrightarrow \mathsf{Message}$$

and the decryption function

$$\mathsf{decr} :: \mathsf{Key} \times \mathsf{Message} \longrightarrow \mathsf{Message}$$

are commonly known, and relate as expected: for each message the decryption of an encryption with a certain key of a message, gives back the original message again, that is, $\mathsf{decr}K(\mathsf{encr}KM) = M$. The idea is that without the key K, it is impossible — or more commonly: infeasible — to recover message M from the encrypted $\mathsf{encr}KM$.

In *public key encryption*, on which we will focus here, key K is public, but its inverse K^{-1} is kept private. The idea is that it is computationally infeasible to compute K^{-1} from K, and that an encrypted message cannot be deciphered without possessing the inverse key. When a key K is viewed as a function, it has the type

$$K :: \mathsf{Message} \longrightarrow \mathsf{Message},$$

and its inverse has the type

$$K^{-1} :: \mathsf{Message} \longrightarrow \mathsf{Message}.$$

If a uses a key K_b that is made public by b to encode a message M, then b, who is the sole owner of K_b^{-1}, will be able to decipher it with $K_b^{-1}(K_bM) = M$.

Furthermore, if we assume that the private key K^{-1} is not only the left-inverse, but also the right-inverse of K, it can be used for *signing* messages. When the owner b of public key K_b applies her private key K_b^{-1} to a message M, the resulting $K_b^{-1}M$ can in principle be decoded by everyone, with the publicly known key K_b, since $K_b(K_b^{-1}M) = M$, but *it can only have been created by someone possessing the secret key* K_b^{-1}, in this case assumedly b herself. In this way the public-private key pair can be used for ensuring the destined receiver to the sender, and ensuring the sender to the receiver. In practice, one would consider a tuple $(M, b, K_b^{-1}M)$ as a complete signed message, which consists of the plain message, the sender and the signed message itself, the latter treated as a proof of the first two, which can be checked by everyone. For simplicity reasons, we will denote a signed message here only by $K_b^{-1}M$.

Linking Agents to Keys If we want to model message passing by public key encryption with dynamic doxastic logic, we may assume that it is public knowledge that a certain key K_b belongs to b. We will not get into details here as to how such situation can be established; in practice, the use of a *trusted* key server (or authentication server) often solves this problem. The idea is that the key server is trusted on statements which link identities to keys, for example by handing out keys to agents which are (in another way) authenticated to it and by (one time) creating (sending around) a signed message (*certificate*) stating this. Nevertheless, considering that keys may get lost or get stolen, the problem remains that one needs external (non-cryptographic) means for relating the possession of a key to identification of agents. In this paper, we assume the setting as already given.

So the answer to the question 'Can you tell who wrote a given signed message?' seems to be: you can tell which key was used to encode it, and you can conclude that the agent that signed it must possess the key. Only to the extent that key is securely linked to an agent, you can securely identify the author of a signed message.

In principle, the question of knowing the key is separate from the question of knowing the possessors of it, but for our analysis we simplify by keeping the possessors of a key fixed, and assuming the identity of the possessor(s) to be common knowledge.

Linking Messages to Senders Suppose b receives a secret message signed by a. The message is encoded with b's public key, and it contains a message signed by a (using a's secret key), so it has form $K_b(K_a^{-1}M)$. Nothing would prevent b from resending the signed message of a to, say, c. All b has to do is decode $K_b(K_a^{-1}M)$ by means of

$$K_b^{-1}K_b(K_a^{-1}M) = K_a^{-1}M$$

and then form and send a message $K_c(K_a^{-1}M)$ to c. This suggests to c that a sent her a message, as it contains no trace of the resending by b. Suppose the message says

> Please meet me tomorrow at 8 p.m. in the lobby of Hotel de l'Europe.
> Signed: a.

Then b could fool c into a surprise meeting with a. This can be prevented by means of mentioning the recipient in an encoded message: a should be careful to word his message as:

> Dear b,
> Please meet me tomorrow at 8 p.m. in the lobby of Hotel de l'Europe.
> Signed: a.

In that case, c will understand that the message is not meant for her.

Even in the very simple set-up that we will assume below this distinction is relevant. Suppose c thinks that the message M is nonsense (we will make this more precise later), then receiving $K_c(K_a^{-1}M)$ — even if sent by b — will make c start to distrust a, not b.

In other words, signing messages gives security about authors of messages (in a broad sense), and not necessarily about senders. In what sense one is responsible for messages that are passed on by others remains a part of discussion and of interpretation. In our approach we maintain responsibility for the original authors. In the above example: c justly starts to distrust a.

3 A Language for Passing Secrets

In this section we introduce a language for messages and the reasoning about the broadcasting of them. We also discuss the representation of trust and trustworthiness in that language.

Building Messages The simplest situation to analyze seems to be the case where there is a finite number of agents $a, b, c, \ldots$, and it is common knowledge that they have public keys $K_a, K_b, K_c, \ldots$ Let us assume further that the agents exchange messages consisting of formulas, which in fact are invitations to *learn* something, and to accept the formula as true. We will make this more specific as we go along. For now, assume that sending a message can be represented with an expression of category $\mathsf{Message}$.

As a starting point, we will first look at the *event* of a message passing by, without worrying about who actually performed this action, i.e. who triggered this event. Also, we assume a broadcasting network, so that all agents receive the (total) message, and an addressee need not be specified.

All messages may contain signatures, but messages need not be signed. All messages may be encrypted with someone's public key, so that other agents cannot read it. As we assume that possessors of keys are fixed, and we only describe public key cryptography, we can identify the notion of "encrypting with key K" with "encrypting for agent a", for a the (sole) possessor of K. Hence we do not need a special type of keys in our language here, and we can use *names of agents* instead of keys, to specify the encryption or signing function. Under these assumptions we can define messages recursively either as a formula (F_{Message}), or a signed or encrypted message, as follows:

$$
\begin{array}{rcl}
\mathsf{Agent} & ::= & a \mid b \mid c \mid \cdots \\
F_{\mathsf{Message}} & ::= & p \mid q \mid \cdots \\
 & & \mid F_{\mathsf{Message}} \wedge F_{\mathsf{Message}} \mid \neg F_{\mathsf{Message}} \\
\mathsf{Message} & ::= & F_{\mathsf{Message}} \\
 & & \mid \mathsf{Encr}_{\mathsf{Agent}}\mathsf{Message} \\
 & & \mid \mathsf{Sign}_{\mathsf{Agent}}\mathsf{Message} \\
 & & \mid \mathsf{Sign}^{\mathsf{Agent}}_{\mathsf{Agent}}\mathsf{Message}
\end{array}
$$

Here $\mathsf{Encr}_a m$ can be read as message m encrypted for a, i.e. with a's public key. Similarly, $\mathsf{Sign}_b m$ is to be read as message m signed by b, that is, b's private key is applied to message m to sign it. $\mathsf{Sign}^a_b m$ means: message m, addressed to a, and signed by b, that is: message m identifies a as the intended recipient, and b's private key is used to sign it.

Meaning and Representation of Trust Trust between agents can be modeled in a number of ways. One of the simplest is to take trustworthiness of an agent a as a basic property Ta. We could then model 'b has complete trust in a' as $\Box_b Ta$ (i.e. 'b believes a is trustworthy'), and 'b has some trust in a' as $\Diamond_b Ta$ ('b considers it possible that a is trustworthy'). However, trustworthiness is not necessarily an intrinsic property, but rather something that could be used in a calculated way. The main difficulty of this analysis is that from an operational viewpoint, where trust is something which an agent a deserves in view of his behaviour vis-a-vis b, trust is more plausibly viewed as a relation. After all, it is very well possible that I have every reason to trust someone whom you have every reason to distrust. My bridge partner is trustworthy for me, but will try to help me in misleading our opponents.

Alternatively, we could consider not trustworthiness, but *trustfulness* as a basic property of an agent. In that case, agents could be either trustful or suspicious to all other agents. Again, this does not seem to be a subtle enough description of reality. If one agent shows untrustworthy behaviour, it does not mean that all others also turned into cheaters. At the same time when I fully trust someone, I could fully distrust someone else.

We therefore consider trust as a relation between agents, and we analyze 'a has complete trust in b' as: $\Box_a \mathsf{Trust}ab$, to be paraphrased as 'a knows that b has always in the past behaved in a trustworthy way towards a', or 'a knows that b has never lied to him'.

It is difficult to think about a distinction between $\mathsf{Trust}ab$ and $\Box_a \mathsf{Trust}ab$. Is it desirable that agents trust someone but (falsely) believe they do not? Or the other way around: could agents believe the trust, but in fact they do not? Exactly because trust is a property of their own internal structure or behaviour, one would expect agents not to be uncertain about their trust and distrust — let alone to be misled. Some *knowledge* rather than belief about trust seems more appropriate. But replacing in our approach beliefs by knowledge for all facts of the world is not desirable at all: when an encrypted message is being passed, agents who cannot read it could, in our simplified system, treat it as if nothing has happened. So, in fact, when considering actions as action structures, as we will do in this paper, the accessibility relations within those action structures are not always reflexive: in some cases agents have *wrong* beliefs about the action that is happening, and they may, as a result, acquire wrong beliefs about the world, even if all their beliefs were right beforehand.

In this paper we will therefore assume a special property only for trust, which works as a knowledge axiom for trust and distrust, but *only for the agent whose trust is at stake.* In fact, the axiom gives a stronger property than knowledge: it guarantees *certainty* about an agent's own trust or distrust. An agent either knows he trusts another agent, or he knows he distrusts, but cannot be uncertain. In other words, when formulated as an axiom, we have for all agents a and b:

$$\vdash (\mathsf{Trust}ab \leftrightarrow \Box_a \mathsf{Trust}ab) \wedge (\neg\mathsf{Trust}ab \leftrightarrow \Box_a \neg\mathsf{Trust}ab)$$

Together with the belief assumptions in general, which we will get back to in Section 9, this should give us an intuitive property of trust: the only proposition which truth value the agent is sure of. We will prove that after the sending of messages (and their processing), as modelled in this paper, this property still holds.

Note that agents do not need to know or be sure of other agents' trust. Question: does the fact that a ceases to trust b entail that b should be aware of this fact? Answer: not necessarily. Think of cases where my telling you a silly story causes you to not take me seriously anymore, without me being aware that anything is amiss.

A Language for Message Passing We need belief operators for every agent, and action modalities for every action, so we define the more extensive logical language (where L_{MP} stands for the language of message passing) as follows:

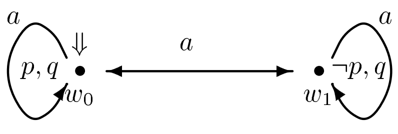
$$\begin{array}{rcl}
\mathsf{Prop} & ::= & p \mid q \mid \cdots \mid \mathsf{TrustAgentAgent} \\
L_{\mathsf{MP}} & ::= & \mathsf{Prop} \mid \neg L_{\mathsf{MP}} \mid L_{\mathsf{MP}} \wedge L_{\mathsf{MP}} \\
& & \mid \Box_{\mathsf{Agent}} L_{\mathsf{MP}} \\
& & \mid \Box^*_{\mathcal{P}(\mathsf{Agent})} L_{\mathsf{MP}} \\
& & \mid [\mathsf{Message}] L_{\mathsf{MP}}
\end{array}$$

We treat assertions about which agent trusts which other agent as basic propositions, as we discussed above. Also, we write instances of the knowledge modalities as usual: $\Box_a \varphi, \Box^*_{\{a,c,d\}} \psi$. Furthermore, the expression $[m]\varphi$ is read as: after broadcasting message m, φ holds. We will extend the language L_{MP} to the general language of dynamic doxastic logic (which includes more general actions) in Section 5.

4 Examples

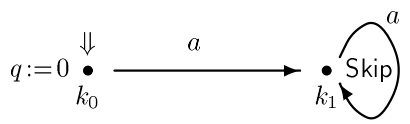
Before we introduce our formal apparatus, we will introduce our approach by means of some, hopefully illuminating, examples.

Suppose there is a single agent a who rightly believes that q is true, but does not know whether p is the case or not, while in fact p is true. We can picture this as the following Kripke-style doxastic state (with a '$\Downarrow$' pointing at the actual world):



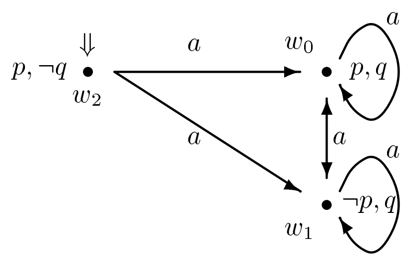
That is, a cannot distinguish between worlds w_0 and w_1: in either one a considers both w_0 and w_1 possible.

Consider now the change of the world where q changes to false, but the agent is unaware of the change. We will picture this action as a Kripke model, in the same style as the doxastic model above: with a $\Downarrow$ pointing at the happening that actually takes place, and $\xrightarrow{a}$ arrows indicating what agent a *thinks* (or: considers possible) that takes place. This yields the following picture:



So the actual action is that q is set to false, while a considers it only possible that nothing (Skip) is happening.

The result of updating the doxastic model with this epistemic action is a model where q has been set to false in the actual world (resulting in a new actual world w_2), while the only two worlds a considers possible (and still cannot distinguish) are the two 'old' worlds of the earlier model, where nothing has changed:



Indeed, from a's point of view nothing has happened, while in fact a change of facts took place.

Below, we will formally introduce a language where action expressions are built from basic commands by means of $+$ (choice), $\circ$ (sequencing), ^{a} (a-suspicion), and μ recursion. An action expression in this language that denotes the action structure of the example, is $q := 0 + (\mu x.(\mathsf{Skip} + x^a))^a$. The μ operator represents here the self reference in the a-loop at the Skip-action, and will be explained below.

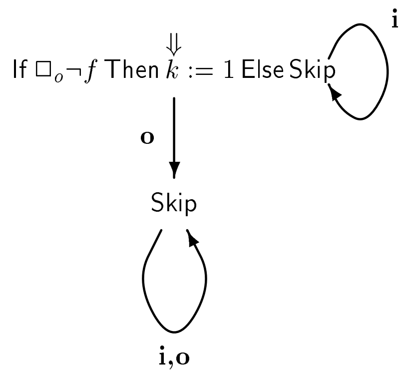
Our framework is rich enough to model dispositions to act in different ways depending on what one believes. Here is Othello's disposition (with o for Othello, $\neg f$ for 'Desdemona is unfaithful' and k for 'Othello kills Desdemona'):

$$\mathsf{If}\ \Box_o \neg f\ \mathsf{Then}\ k := 1\ \mathsf{Else}\ \mathsf{Skip}.$$

If Othello believes Desdemona is unfaithful to him he will kill her, otherwise he will do nothing. This is not yet truly dramatic. The following action expression captures Shakespeare's plot more fully:

$$\mu x.(\mathsf{If}\ \Box_o \neg f\ \mathsf{Then}\ k := 1\ \mathsf{Else}\ \mathsf{Skip}) + x^i + (\mu y.y^o + y^i)^o.$$

Or, in a picture:



Iago is aware of Othello's disposition, but Othello himself is not. Note that Iago knows that Othello is not aware of that, but Othello does not suspect this. Even Iago's knowledge that Othello is mistaken is covered by this action picture. We leave it to the reader to extend the action expression with Othello's trust in Iago.

5 Epistemic Actions and Actions on the Real World

Doxastic models Like in the semantics of [1], we define doxastic models as tuples $\mathbf{W} = (W, (\longrightarrow_a^W)_{a \in \mathsf{Agent}}, V^W)$ where W denotes the (finite) set of worlds, $\longrightarrow_a^W$ (usually we leave out the superscript W) denotes the accessibility relations for each agent and $V^W :: W \longrightarrow \mathcal{P}(\mathsf{Prop})$ is a valuation function. Recall our assumption that for any pair of agents a, b the proposition $\mathsf{Trust}ab$ is in Prop. In the coming sections, we will not require any additional properties on models and actions, but we will introduce them in our analysis of the sending of messages in Section 9.

Doxastic states An doxastic state (or a pointed model) is now a pair $s = (\mathbf{W}, w)$ of an doxastic model and a designated world (the "actual world"). We read $p \in V^W(w)$ as "p is true in world w". If

$$s = ((W, (\longrightarrow_a^W)_{a \in \mathsf{Agent}}, V^W), w)$$

is a state, we use V_s for $V^W(w)$, i.e., V_s is the valuation of the actual world of the state. We use State to refer to the set of states.

Language of Dynamic Doxastic Logic It is convenient to broaden our scope first, before focusing on encoded message passing as example actions. We extend the language L_{MP} (from Section 3) to the general action language L, as follows:

$$L \quad ::= \quad \mathsf{Prop} \mid \neg L \mid L_1 \wedge L_2 \mid \Box_{\mathsf{Agent}} L \mid \Box^*_{\mathcal{P}(\mathsf{Agent})} L \mid [\mathsf{Action}] L$$

The only difference with the language L_{MP} is the last line, where we allow expressions of the form $[\alpha]\varphi$, where α is an action (not just a message). We will describe in Section 9 how the messages relate to these general actions; they are more easily understood in terms of general actions. The general action type Action will be defined below.

Here are some standard abbreviations:

$\bot$ is shorthand for $p \wedge \neg p$.

$\top$ is shorthand for $\neg\bot$.

A Language of Basic Commands For actions, we will allow not only doxastic actions (which influence the beliefs of agents), but also actions which change the truth values of facts (proposition letters) of the actual world. We need that, as we have modelled trust by the propositional construct $\mathsf{Trust}ab$, and we want to be able to change trust into distrust, if desired. (In Section 3 we discussed why we take a propositional construct (as "fact" in every world of the model) for the binary trust relation.) To specify actions on the real world, we use the following command language:

$$\begin{array}{rcl} \mathsf{Bool} & ::= & 0 \mid 1 \\ \mathsf{Assign} & ::= & \mathsf{Prop} := \mathsf{Bool} \\ \mathsf{Assigns} & ::= & \{\mathsf{Assign}, \ldots, \mathsf{Assign}\} \\ \mathsf{Com} & ::= & \mathsf{If}\ L\ \mathsf{Then}\ \mathsf{Com}\ \mathsf{Else}\ \mathsf{Com} \mid \mathsf{Assigns} \end{array}$$

Assignments are expressions of the form $p := 0$ or $p := 1$, and sets of assignments ($\mathsf{Assigns}$) are interpreted as simultaneous assignments. Intuitively, $\mathsf{If}\ \varphi\ \mathsf{Then}\ \{p := 1, q := 1, r := 0\}\ \mathsf{Else}\ \{\}$ means that in states where φ is true, the facts about p, q, r, in the real world are simultaneously changed into true, true, false; in states where φ is not true, nothing will be changed. Note that setting p to the value determined by φ is done by means of the command

$$\mathsf{If}\ \varphi\ \mathsf{Then}\ p := 1\ \mathsf{Else}\ p := 0.$$

A set of assignments $\{A_1, \ldots, A_n\}$ is *consistent* if it does not contain any contradictory pairs $p := 0, p := 1$. We define for every list of assignments A the sets A^+ and A^- of proposition letters as follows:

$$\begin{array}{rcl} A^+ & := & \{p \mid p := 1 \in A\} \\ A^- & := & \{p \mid p := 0 \in A\} \end{array}$$

In other words, A is inconsistent if $A^+ \cap A^- \neq \emptyset$. Inconsistent assignment sets are interpreted to fail, so we treat them all equal. We will also write (A^+, A^-) to denote A.

Here are some useful abbreviations:

Skip is shorthand for the command $\{\}$.

Fail is shorthand for the command $\{p := 1, p := 0\}$ (or in fact, any other inconsistent assignment list).

$\varphi?$ is shorthand for the command $\mathsf{If}\ \varphi\ \mathsf{Then}\ \mathsf{Skip}\ \mathsf{Else}\ \mathsf{Fail}$.

This command language allows us to change propositional valuations and the Trust relation, either unconditionally or subject to certain conditions, and to fail, subject to certain conditions.

A Lattice of Assignment Sets Notice that, if we identify all inconsistent assignment lists with each other (which we did in the definition of Fail), $\mathsf{Assigns}$ is a complete lattice under the ordering $\sqsubseteq$, defined as $A \sqsubseteq B$ iff either $B = \mathsf{Fail}$ or both $A^+ \subseteq B^+$ and $A^- \subseteq B^-$ (see Davey and Priestley [5]). Let $\sqcup$ be the join operation in this lattice. Then we have: $A \sqcup B = (A^+ \cup B^+, A^- \cup B^-)$, and indeed Fail is the top of this lattice and Skip the bottom: $\mathsf{Skip} \sqsubseteq A \sqsubseteq \mathsf{Fail}$ for all A. Hence the following laws also hold: $A \sqcup \mathsf{Fail} = \mathsf{Fail} \sqcup A = \mathsf{Fail}$ and $A \sqcup \mathsf{Skip} = \mathsf{Skip} \sqcup A = A$.

A Language of Action Expressions Changes in the real world, as described by the above command language, can be viewed from different perspectives. Not everyone may be aware of a change, e.g., or some agents may even think that a different change is occurring than the actual change. The notion of an action (expression) takes all this into account, for in action expressions actions can appear under 'suspicion relations' (to use a phrase coined by Alexandru Baltag). The action expressions of our language are given by:

$$\mathsf{Action} \ ::= \ \mathsf{Com} \mid x^a \mid \mathsf{Action}^a \mid \mathsf{Action}_1 + \mathsf{Action}_2 \mid \mathsf{Action}_1 \circ \mathsf{Action}_2 \mid \mu x \cdot \mathsf{Action}$$

In words, an action expression in the first place be a bare *command*, as defined above. It can also be a variable x (referring to an action) or an action expression itself under *suspicion* of an agent a, which means that a considers x or the action as possibly happening. This suspicion of a is an action in itself that is happening. An action can also be a *parallel composition* of two actions α_1 and α_2, denoted by $\alpha_1 + \alpha_2$. Although '+' as a symbol for parallellism may be confusing (traditionally it is used for choice), we chose to keep the notation as introduced by Baltag et al.; the intuition is that the parallel composition of two suspicions by the same agent behaves like a choice: $\alpha^a + \beta^a$ means that agent a considers either α or β possibly happening. Furthermore, an action can consist of a *sequential composition* $\alpha_1 \circ \alpha_2$ of two actions α_1 and α_2 (with the traditional interpretation). Finally, an action can have the form $\mu x.\alpha$, where α is again an action expression. This expression is interpreted as α itself, in which occurring variables x refer back to the full expression.

Note that the only operator that can be applied directly on a variable is the suspicion operator (in x^a); which means that the only circularity we are interested in is the circular *awareness* of actions happening. This will correspond to circular arrows in the action models we will introduce in Section 6 — pictures of which we have seen in Section 4. We will see that it is intuitive to think of the μx operator as a *place holder*: μx gives name x to the current action (sub) expression, and if within that expression the name x is referred to (like in x^a), it means that there will be suspicion (by a) about it. We have seen in Section 4 how this corresponds to a picture view of actions.

Nota bene: An action is *closed* if every variable occurrence x in it is in the scope of a μx operator. In this paper, we mean by 'action expression' a closed action expression.

Action Models and Action Structures Using the bare commands as building blocks for our semantics, we now define action models as tuples $\mathbf{K} = (K, (\longrightarrow_a^K)_{a \in \mathsf{Agent}}, C^K)$, where K is a set of action tokens, $\longrightarrow_a^K$ (again, we usually leave out superscript K) denotes the accessibility relation for each agent, and $C^K :: K \longrightarrow \mathsf{Com}$ is a function which maps every action token to a command. We will call such a function a *change function*, as it represents the different atomic changes that may be happening or suspected to happen during the action.

Note the similarities between doxastic models and action models: both have a graph structure, where edges are labelled by agents, and the nodes also have a 'value' given by the valuation function (doxastic models) or the change function (action models).

Analogous to a pointed model, a pointed action model, or action structure, is now a pair $\alpha = (\mathbf{K}, k_0)$ of an action model and a designated action token (the "actual action"). We use $\mathsf{ActionStructure}$ to refer to the set of action structures. We will use $\alpha, \beta, \ldots$ to refer both to action expressions and to action structures; we will see that there is a correspondence between the two types, and from the context it will be clear which of the two is used.

Lifting the Inner Accessibility Relations and Functions We define the relation $\longrightarrow_a$ between pointed models (or action structures) as meaning that the underlying models coincide and the tops (designated worlds or action tokens) are related according to the accessibility relation within the underlying model (or action structure). Hence, for $s = (\mathbf{W}, w)$ and $s' = (\mathbf{W}', w')$, $s \longrightarrow_a s'$ iff $\mathbf{W} = \mathbf{W}'$ and $w \longrightarrow_a w'$; the relation $\alpha \longrightarrow_a \alpha'$ is defined analogously. Similarly, we write V_s, or sometimes $V(s)$, for the valuation in the top world of state s, and C_α or $C(\alpha)$ for the command in the top action of pointed action structure α.

6 Semantics

We will first give a general semantics of the doxastic action logic, together with a definition of the update function for the actions as defined above. In Section 9 we show how the sending of a message m can be captured in such a pointed action structure.

Truth in a Doxastic State The truth relation $\models$ between an doxastic state $s = ((W, (\longrightarrow_a^W)_{a \in \mathsf{Agent}}, V^W), w)$ and a formula $\varphi \in L$ is defined as follows (recursively over the structure of φ):

$$
\begin{array}{lcl}
s \models p & := & p \in V_s \\
s \models \neg\varphi & := & s \not\models \varphi \\
s \models \varphi \wedge \psi & := & s \models \varphi \text{ and } s \models \psi \\
s \models \Box_a \varphi & := & s' \models \varphi \text{ for all } s' \text{ with } s \longrightarrow_a s' \\
s \models \Box_A^* \varphi & := & s' \models \varphi \text{ for all } s' \text{ with } s \longrightarrow_A^* s' \\
s \models [\alpha]\varphi & := & s' \models \varphi \text{ for all } s' \text{ with } s' = s.[\![\alpha]\!]
\end{array}
$$

Note that formulas of the form $\mathsf{Trust}ab$ are treated as propositions, as we assume that $\{\mathsf{Trust}ab \mid a, b \in \mathsf{Agent}\} \subseteq \mathsf{Prop}$.

The function '$[\![.]\!]$' (as in '$[\![\alpha]\!]$') that maps action expressions to actions structures and the update function '.' (as in '$s.[\![\alpha]\!]$') that updates states with action structures will be given below. We will also define a function Expr which does the opposite of '$[\![.]\!]$': it gives for each given (finite) action model a corresponding action expression.

Applying a Command to a State For commands C, we define $\langle\!\langle C \rangle\!\rangle(s)$ as the new valuation in the top world of s after command C has been executed. As valuations are sets of proposition letters, the function $\langle\!\langle . \rangle\!\rangle$ is of type $\langle\!\langle . \rangle\!\rangle :: \mathsf{Com} \longrightarrow (\mathsf{State} \longrightarrow \mathcal{P}(\mathsf{Prop}))$, taking a command and a state, and returning a valuation. We define this function only for states where the command does not fail. We state the definition recursively to commands (A refers to an assignment list and C_1, C_2 to any command):

$$
\begin{array}{rcl}
\langle\!\langle \mathsf{If}\ \varphi\ \mathsf{Then}\ C_1\ \mathsf{Else}\ C_2 \rangle\!\rangle(s) & := & \begin{cases} \langle\!\langle C_1 \rangle\!\rangle(s) & \text{if } s \models \varphi \\ \langle\!\langle C_2 \rangle\!\rangle(s) & \text{otherwise} \end{cases} \\
\langle\!\langle A \rangle\!\rangle(s) & := & (V_s \cup A^+) \setminus A^- \text{ if } A \text{ consistent}
\end{array}
$$

Note that $\langle\!\langle \mathsf{Fail} \rangle\!\rangle(s)$ is undefined for every state s.

Operations on Action Structures Next, we associate an action structure with every closed action. This definition is an extension of the definition in [1]. First, we define the semantic operations $^{\mathbf{a}}$, $\oplus$ and $\odot$ on action structures which correspond with the syntactic operations ^{a}, $+$ and $\circ$ that construct action expressions. In order to define them, we also need a function Expr which gives an action expression for each finite action structure.

Operations on Action Structures: Suspicion The operation $^{\mathbf{a}}$ (with $a \in \mathsf{Agent}$) gives for a given action structure $(\mathbf{K}, k_0)$ a new action structure $(\mathbf{K}, k_0)^{\mathbf{a}}$, which consists of a new top node from which only one arrow leaves: an a-arrow pointing to the top k_0 the initial structure $(\mathbf{K}, k_0)$. The idea is that it reflects that a suspects the given structure. It is defined as follows:

$$((K, (\rightarrow_b)_{b \in \mathsf{Agent}}, C), k_0)^{\mathbf{a}} := ((K', (\rightarrow'_b)_{b \in \mathsf{Agent}}, C'), k'_0),$$

where

- $K' = K \cup \{n\}$, with n new,
- $(\rightarrow'_b)_{b \in \mathsf{Agent}} = (\rightarrow_b)_{b \in \mathsf{Agent}} \cup \{n \rightarrow_a k_0\}$,
- C' given by: $C'(k) = C(k)$ for $k \in K$, $C(n) = \mathsf{Skip}$,
- $k'_0 = n$.

Operations on Action Structures: Parallel Composition The operation $\oplus$ returns for two action structures their parallel composition: an action structure which top gives a parallel "merge" of the two given top tokens, and which unites the arrows of the given tops. It is defined as follows:

$$((K,(\rightarrow_b)_{b\in\mathsf{Agent}},C),k_0)\oplus((K',(\rightarrow'_b)_{b\in\mathsf{Agent}},C'),k'_0) := ((K'',(\rightarrow''_b)_{b\in\mathsf{Agent}},C''),k''_0),$$

where

- K'' is the direct (disjoint) sum of K, K' and $\{n\}$, with n new,
- $(\rightarrow''_b)_{b\in\mathsf{Agent}} = (\rightarrow_b)_{b\in\mathsf{Agent}} \cup (\rightarrow'_b)_{b\in\mathsf{Agent}} \cup \{n \rightarrow''_b k \mid k_0 \rightarrow_b k \text{ or } k'_0 \rightarrow'_b k\}$.
- C'' is given by:
  - $C''(k) = C(k)$ for $k \in K$,
  - $C''(k) = C'(k)$ for $k \in K'$,
  - $C''(n) = C(k_0) \parallel C'(k'_0)$, with $\parallel$ as given below.
- $k''_0 = n$.

In the definition we use a parallel composition on commands, which we define as follows:

$$\begin{array}{rcl}
\parallel & :: & \mathsf{Com}\times\mathsf{Com} \longrightarrow \mathsf{Com} \\
A_1 \parallel A_2 & := & A_1 \sqcup A_2 \\
(\mathsf{If}\,\varphi\,\mathsf{Then}\,C_1\,\mathsf{Else}\,C_2) \parallel C_3 & := & \mathsf{If}\,\varphi\,\mathsf{Then}\,(C_1 \parallel C_3)\,\mathsf{Else}\,(C_2 \parallel C_3) \\
C_1 \parallel C_2 & := & C_2 \parallel C_1
\end{array}$$

Recall the definition of $\sqcup$ as we have defined above: $A_1 \sqcup A_2 = (A_1^+ \sqcup A_2^+, A_1^- \sqcup A_2^-)$. Note that this indeed defines the function, as evaluation order does not matter. It is easy to see that $\parallel$ is commutative, associative and idempotent.

Operations on Action Structures: Sequential Composition The operation $\odot$ returns the sequential composition of two given action structures. The action tokens consist of sequential compositions of pairs of action tokens from the two given structures, and the idea is that a sequential composition of suspicions by an agent is a suspicion by that agent of the sequential composition ("if b first thinks that α may be happening, and afterwards b thinks that β may be happening, then in fact b thinks that α followed by β may be happening"). The operation is defined as follows:

$$((K,(\rightarrow_b)_{b\in\mathsf{Agent}},C),k_0)\odot((K',(\rightarrow'_b)_{b\in\mathsf{Agent}},C'),k'_0) := ((K'',(\rightarrow''_b)_{b\in\mathsf{Agent}},C''),k''_0),$$

where

- $K'' = \{(k,k') \mid k \in K, k' \in K'\}$,
- $(\rightarrow''_b)_{b\in\mathsf{Agent}}$ given by $(k_1,k'_1) \rightarrow''_b (k_2,k'_2)$ iff $k_1 \rightarrow_b k_2$ and $k'_1 \rightarrow'_b k'_2$.
- C'' is given by: $C''(k,k') = \mathsf{Expr}(\mathbf{K},k); C'(k')$, with the functions Expr and ';' as given below.
- $k''_0 = (k_0,k'_0)$.

In the definition we use a kind of sequential composition ';' on pairs of action structures and commands, which we define as follows:

$$\begin{array}{rcl} ; & :: & \mathsf{Action} \times \mathsf{Com} \longrightarrow \mathsf{Com} \\ \alpha; (\mathsf{If}\ \varphi\ \mathsf{Then}\ C_1\ \mathsf{Else}\ C_2) & := & \mathsf{If}\ [\alpha]\varphi\ \mathsf{Then}\ (\alpha; C_1)\ \mathsf{Else}\ (\alpha; C_2) \\ A_1; A_2 & := & (A_2^+ \cup (A_1^+ \setminus A_2^-), A_2^- \cup (A_1^- \setminus A_2^+)) \\ (\mathsf{If}\ \varphi\ \mathsf{Then}\ C_1\ \mathsf{Else}\ C_2); A & := & \mathsf{If}\ \varphi\ \mathsf{Then}\ (C_1; A)\ \mathsf{Else}\ (C_2; A) \\ \alpha; A & := & \mathsf{topcomm}(\alpha); A\ \text{(for } \alpha \notin \mathsf{Com}) \end{array}$$

Here, topcomm is a function which gives the command which is applied to the actual world, i.e. the command of the top action token, It is defined mutually with the above ';', as follows:

$$\begin{array}{rcl} \mathsf{topcomm} & :: & \mathsf{Action} \longrightarrow \mathsf{Com} \\ \mathsf{topcomm}(C) & := & C \\ \mathsf{topcomm}(\alpha^a) & := & \mathsf{Skip} \\ \mathsf{topcomm}(\alpha + \beta) & := & \mathsf{topcomm}(\alpha) \parallel \mathsf{topcomm}(\beta) \\ \mathsf{topcomm}(\alpha \circ \beta) & := & \alpha; \mathsf{topcomm}(\beta) \\ \mathsf{topcomm}(\mu x.\alpha(x)) & := & \mathsf{topcomm}(\alpha(\mu x.\alpha(x))) \end{array}$$

For the topcomm of a μ expression $\mu x.\alpha(x)$ we recall that a variable x can only occur in $\alpha(x)$ under suspicion of some agent a. As topcomm of a suspicion is Skip, it is clear that the definition is not circular.

We also use a function Expr, which gives a corresponding action expression for every finite action structure. It is defined as follows:

$$\begin{array}{rcl} \mathsf{Expr} & :: & \mathsf{ActionStructure} \longrightarrow \mathsf{Action} \\ \mathsf{Expr}(\mathbf{K}, k_0) & := & \mathsf{buildexpr}\ k_0\ \emptyset \\ \text{where } \mathsf{buildexpr}\ k\ V & := & \begin{cases} x_k & \text{if } k \in V \\ \mu x_k.C^K(k) + \sum_{a \in \mathsf{Agent}} \sum_{k' \in K, k \longrightarrow_a k'} (\mathsf{buildexpr}\ k'\ (V \cup \{k\}))^a & \text{otherwise} \end{cases} \end{array}$$

This definition can be read as follows: for every action token k we introduce a subexpression preceded by a μ operator (μx_k). We introduce an a-suspicion for every $a \in$ Agent and $k \longrightarrow_a k'$, either to $x^a_{k'}$ (if we are already in the scope of $\mu x_{k'}$), or to a more complicated subexpression which introduces $x_{k'}$ and describes the token $((\mu x_{k'}.C^K(k) + \ldots)^a)$. For an example we refer to the Othello-example of Section 4, in which, if we call the two action tokens k_0 and k_1, x should be read as x_{k_0} and y as x_{k_1}. In fact, the above definition of buildexpr introduces unnecessary μ operators (also for action tokens which do not have a self-reference in their subexpression), but we use this definition for readability and simplicity reasons.

Interpretation of Action Expressions The action structure associated with an action expression α is denoted by $[\![\alpha]\!]$, and is defined as follows:

$$\begin{array}{rcl} [\![\ ]\!] & :: & \mathsf{Action} \longrightarrow \mathsf{ActionStructure} \\ [\![\mathsf{Com}]\!] & := & ((\{n\}, \emptyset, \{(n, \mathsf{Com})\}), n) \\ [\![\mathsf{Action}^a]\!] & := & [\![\mathsf{Action}]\!]^{\mathbf{a}} \end{array}$$

$$
\begin{aligned}
[\![\mathsf{Action}_1 + \mathsf{Action}_2]\!] &:= [\![\mathsf{Action}_1]\!] \bigoplus [\![\mathsf{Action}_2]\!] \\
[\![\mathsf{Action}_1 \circ \mathsf{Action}_2]\!] &:= [\![\mathsf{Action}_1]\!] \bigodot [\![\mathsf{Action}_2]\!] \\
[\![\mu x.\mathsf{Action}(x)]\!] &:= [\![\mathsf{Action}(\mu x.\mathsf{Action}(x))]\!].
\end{aligned}
$$

Note that in the definition of $[\![\mathsf{Com}]\!]$ we use n for a *new* name of an action token. The action structure becomes then one node with that name n, an empty accessibility relation, and a function (here represented as a tuple) which maps n to the command it represents. The top node is the only node n. The operators for suspicion, parallel and sequential composition we have discussed above. The action structure for a μ expression is defined as the action structure of the once "unfolded" μ expression. So, the smallest action structure satisfying the expression $\mu x.x^a$ is, as expected, the structure with one action token that has Skip as a command, and one a-arrow from the token to itself. (See [10] for more on the μ-calculus.)

Updating Doxastic States With Action Structures Finally, we define the update function, which maps a state and an action structure to the state "after the action". Given a state $s = (\mathbf{W}, w_0)$, with $\mathbf{W} = (W, (\longrightarrow_a^W)_{a \in \mathsf{Agent}}, V^W)$, and given an action model $\alpha = (\mathbf{K}, k_0)$ with $\mathbf{K} = (K, (\longrightarrow_a^K)_{a \in \mathsf{Agent}}, C^K)$, we say that s *survives* α, if $C^K(k_0)(s)$ is defined. If s survives α, then we define the *update* $s.\alpha$ of s by α as the state

$$s.\alpha := ((W.K, (\longrightarrow_a^{W.K})_{a \in \mathsf{Agent}}, V^{W.K}), w_0.k_0),$$

where

- $W.K := \{(w,k) \mid w \in W, k \in K, \langle\!\langle C^K(k) \rangle\!\rangle(\mathbf{W}, w) \text{ is defined}\}$
- $w_0.k_0 := (w_0, k_0)$
- $(w,k) \longrightarrow_a^{W.K} (w',k')$ iff both $w \longrightarrow_a^W w'$ and $k \longrightarrow_a^K k'$ for $(w,k), (w',k') \in W.K$
- $V^{W.K}(w,k) := \langle\!\langle C^K(k) \rangle\!\rangle(\mathbf{W}, w)$.

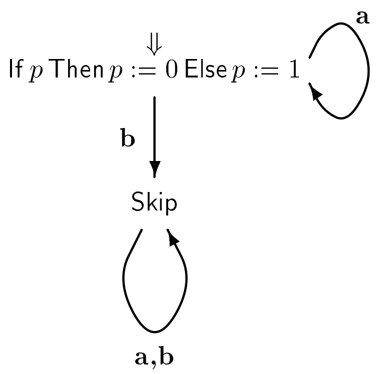
For s and α such that s does not survive α, the update $s.\alpha$ remains *undefined*.

7 Further Examples

For further illustration, here are some examples of actions involving tests and conditionals. Suppose we want to express the following conditional action: if p then p is reset to 0, and otherwise p is reset to 1, while a is aware of this conditional action happening, but b thinks that nothing is happening. Moreover, a realizes what b believes. This can now be represented by the expression:

$$\mu x.(\mathsf{If}\, p\, \mathsf{Then}\, p := 0\, \mathsf{Else}\, p := 1) + x^a + (\mu y.\mathsf{Skip} + y^a + y^b)^b$$

This expression can be interpreted as follows: the first μ operator introduces a placeholder (x) for the outermost (top) action, in which the if-then-else takes place, while at the same time (connected with '+') a is aware of that (x^a) and b suspects something else taking place: $\mu y.\mathsf{Skip} + y^a + y^b$. This latter subexpression indicates an action with placeholder y (via μy) where nothing happens (Skip), while a and b are simultaneously aware of that ($y^a + y^b$). In other words, pictorially we can represent this action as follows:

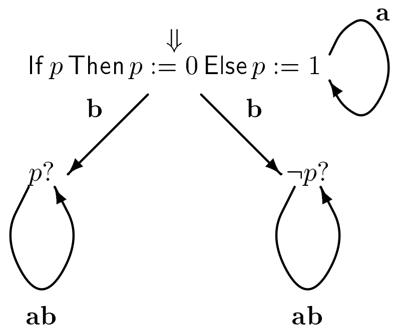


Note that we cannot write the whole expression as a kind of if-then-else, as our if-then-else operator (used in the subexpression) is only defined on *commands*, not on general action expressions. Also, the maybe familiar construction $p? \circ \alpha + \neg p? \circ \beta$ from other formalisms does *not* act as an if-then-else in our language: '+' is not a choice operator, but a parallel composition. Hence $p?$ failing in states where $\neg p$ holds, while $\neg p?$ fails in states where p holds, results in an action that always fails ($p? + \neg p? = \mathsf{Fail}$).

Another interesting expression is the following:

$$\mu x.\mathsf{If}\; p\; \mathsf{Then}\; p := 0\; \mathsf{Else}\; p := 1 + x^a + (\mu y.(p? + y^a + y^b))^b + (\mu y.(\neg p? + y^a + y^b))^b$$

In a picture, this is:



Although exactly one of b's suspected actions (either $p?$ or $\neg p?$) will succeed in each of b's possible worlds, the a, b-labelled arrows affect the worlds accessible from them. So, b learns p or $\neg p$, that is, afterwards she will know that she (together with a) knows either p or $\neg p$, which she did not necessarily know beforehand. That means that either learning p or learning $\neg p$ is, without extra conditions, not the same as learning $(p \vee \neg p)$.

8 State and Action Bisimulations

Expressions of L and action expression are obviously used to describe different things, states and changes of state, but they also differ in another important way. Where the formulas of L that hold in a state 'define' that state *up to bisimilarity*, an action expression describes much more precisely the shape of the corresponding action structure. Furthermore, one action expression "fully" describes the action, while one single L-formula that holds in a state does not necessarily (often possibly) describe all formulas holding in that state. E.g. "$p \vee q$" does not specify if only p, q or $p \wedge q$ hold — and it does not mention if r or $\Box_a p$ holds.

In this section we define a bisimulation relation on states, and, analogously to that, a bisimulation relation on actions. We will see that the bisimilarity of two actions is a sufficient, but not necessary condition on them having the same effects on states. Furthermore, we define another bisimulation relation on pairs of actions and states which does correspond to the effect of an action.

Bisimulation is a key notion in both modal logic and process theory [9, 13]. Bisimulations were introduced in modal logic under the name 'p-relations' in Van Benthem's dissertation, dating from 1976 [3]. The bisimulation notion was introduced in computer science around 1980, where the classical reference is Park 1981 [12]. For a catalogue of classical results about bisimulation, the reader is referred to the work of Hennesy and Milner [8].

Definition 1 *Two states s_1, s_2 are* bisimilar *(notation: $s_1 \sim s_2$) iff*

- $V_{s_1} = V_{s_2}$;
- *for all t_1 such that $s_1 \longrightarrow_a t_1$ for some a, there is a t_2 such that $s_2 \longrightarrow_a t_2$ and $t_1 \sim t_2$;*
- *and vice versa.*

The definition of states that two states s_1, s_2 are bisimilar if they (i.e. their top worlds) have the same valuation ($V_{s_1} = V_{s_2}$), and for every a-arrow $s_1 \longrightarrow_a t_1$ from the top to some world in the underlying model of s_1 there is an a-arrow $s_2 \longrightarrow_a t_2$ from the top to a world in s_2 for which those two worlds t_1, t_2 are, together with their underlying models, again bisimilar states ($t_1 \sim t_2$); and vice versa: for every $s_2 \longrightarrow_a t_2$ we can also find such a $s_1 \longrightarrow_a t_1$.

We will define the notion of bisimilarity for actions completely analogous to the one for states, where the change functions take the role of the valuations. However, rather than requiring equality of commands, we will use a notion of equivalence of commands, defined as follows, using the definition of applying a command to a state from Section 6:

Definition 2 *Two commands C, C' are* equivalent *(notation: $C \approx C'$) if for all states s, $\langle\!\langle C \rangle\!\rangle(s) = \langle\!\langle C' \rangle\!\rangle(s)$.*

In fact, we say that two commands are equivalent if they act the same, i.e. return the same valuation, on every state s. For example, $\mathsf{If}\ p\ \mathsf{Then}\ q := 0\ \mathsf{Else}\ q := 0$ is equivalent to $q := 0$, and $\mathsf{If\ True\ Then\ Skip\ Else\ Fail}$ is equivalent to Skip. (Note that it is sufficient to require this for all bisimilation classes $[s]$.)

Now we define bisimilarity of action structures analogous to bisimilarity of states.

Definition 3 *Two action structures α_1, α_2 are* bisimilar *(notation: $\alpha_1 \sim \alpha_2$) iff*

- $C_{\alpha_1} \approx C_{\alpha_2}$;
- *for all β_1 such that $\alpha_1 \longrightarrow_a \beta_1$ for some a, there is a β_2 such that $\alpha_2 \longrightarrow_a \beta_2$ and $\beta_1 \sim \beta_2$;*
- *and vice versa.*

Note that, unlike the process algebra notion of bisimulation, the action bisimulation does not refer to a *process*, but to a *single action* that has an internal knowledge structure. Following of the arrows according to this definition represents, like in the case of state bisimulations, the comparing of possibilities (actions, worlds), rather than looking ahead in the process. In process algebra, $s \longrightarrow_a t$ would refer to a process step (a, which takes s to t), whereas in our framework $s \longrightarrow_a t$ or $\alpha \longrightarrow_a \beta$ refer to possibility and suspicion relations.

Lemma 4 *For states s, s' and action structures α, α' we have: if $s \sim s'$ and $\alpha \sim \alpha'$, and if $s.\alpha$ is defined, then $s'.\alpha'$ is also defined, and $s.\alpha \sim s'.\alpha'$.*

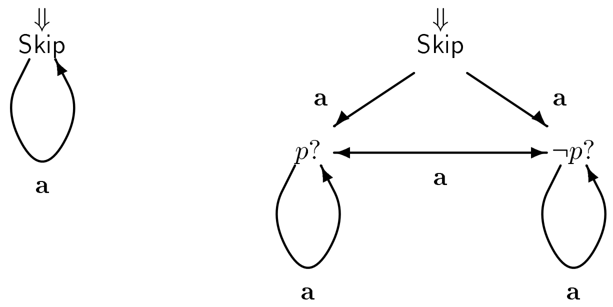
Proof Let $s_1 \sim s_2$ and $\alpha_1 \sim \alpha_2$ be given, with $s_i := (\mathbf{W}_\mathbf{i}, w_i)$ and $\alpha_i := (\mathbf{K}_\mathbf{i}, k_i)$. Suppose that $s_1.\alpha_1$ is defined. Because of the given bisimilarities, we know that $C_{\alpha_1} \approx C_{\alpha_2}$, which are commands that respect bisimilarity. So if they are defined on a state, then they are defined on a bisimilar state as well, so $s'.\alpha'$ is defined. Also, the changes they return for each actual world are equal. In other words: the valuation $\langle\!\langle C_{\alpha_1} \rangle\!\rangle(s_1)$ equals $\langle\!\langle C_{\alpha_2} \rangle\!\rangle(s_1)$, this equals again $\langle\!\langle C_{\alpha_2} \rangle\!\rangle(s_2)$. So (see definition update), $V_{(s_1.\alpha_1)} = V_{(s_2.\alpha_2)}$. This proves the first condition of bisimilarity of $s_1.\alpha_1$ and $s_2.\alpha_2$.

To prove the second part, we use the bisimulation game, and show that Duplicator has a winning strategy. Suppose Spoiler chooses u_1 with $s_1.\alpha_1 \longrightarrow_a u_1$ for some a. First note that the underlying models of $s_1.\alpha_1$ and u_1 are the same, they only may differ in the top world. Denote the top world of u_1 by (w_1, k_1). Now write t_1 for the state with the same underlying model as s_1 and top world w_1, and β_1 for α_1 with top k_1. From the definition of update, we know that $t_1.\beta_1$ is defined (since $w_1.k_1$ is) and $u_1 \sim t_1.\beta_1$. Furthermore, we have $s_1 \longrightarrow_a t_1$ and $\alpha_1 \longrightarrow_a \beta_1$. Now, because of the given bisimilarities, there exist t_2 and β_2 with $s_2 \longrightarrow_a t_2$ and $\alpha_2 \longrightarrow_a \beta_2$ such that $t_1 \sim t_2$ and $\beta_1 \sim \beta_2$. As $t_1.\beta_1$ is defined, $t_2.\beta_2$ is as well, and so Duplicator takes as her move the state u_2 which has the same underlying model as $s_2.\alpha_2$ and top given by the top of $t_2.\beta_2$. Because of the definition of update we have $u_2 \sim t_2.\beta_2$. Now for the bisimilarity of u_1 and u_2, we can play the same game with $t_1.\beta_1$ and $t_2.\beta_2$. As the game will not finish in this way, Duplicator wins.
□

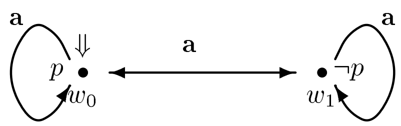
Now, for an example of two actions which have the same effect, but are *not* bisimilar according to the above definition, consider the following two action expressions:

$$\alpha = (\mu x.x^a) \text{ and } \beta = (\mu x.\mu y.(p? + x^a + y^a)^a + (\neg p? + x^a + y^a)^a)$$

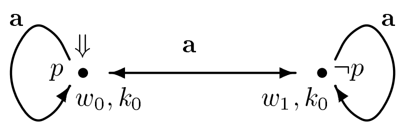
Action α describes a skip action, of which agent a is aware. Action β is a skip action, while a suspects a test for p or for $\neg p$, but cannot distinguish between the two. In a picture these look as follows:



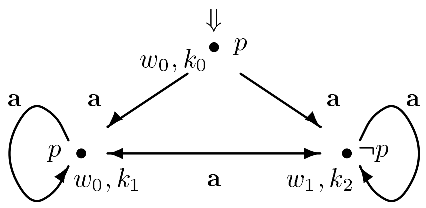
Clearly, the two action structures are not bisimilar, as the change function of a test for a proposition letter or its negation is strictly partial, while the change function of Skip is not, so they are not equal. We will now show for a particular s that the updates with the respective actions nevertheless do give a bisimilar result. The proof that this holds for any state we will not give here. Take for s the following state in which p is true, but agent a cannot distinguish between p being true or not:



The result of updating s with α is now straightforward; roughly, as α contains only one action token, the product with the set of worlds in s is equal to set the worlds in s. As there is an arrow from the single action token to itself, this allows all the arrows in s to be copied in the new world. Hence, as expected, nothing changes, so s equals $s.\alpha$ (we have only changed the labels at the worlds as described in the definition of update, using k_0 here as the name of the action token):



For the update of s by β we first observe that the action token $\neg p?$ fails in w_0, as does $p?$ in w_1, so the set of new worlds after the update $s.\beta$ is: $\{(w_0, k_0), (w_1, k_0), (w_0, k_1), (w_1, k_2)\}$ (here we use k_0 for the top token of β, k_1 for $p?$ and k_2 for $\neg p?$). Furthermore, when calculating the accessibility relation, we see that there is no arrow to the world (w_1, k_0) (because there is no arrow to the token k_0 in β), and, because that is also not the top token of the new state, it is totally disconnected, and we may leave it out in the picture. This leads to the following picture of $s.\beta$:



To see that $s.\alpha$ and $s.\beta$ are bisimilar, observe that both tops are p worlds, and that from each of the worlds in both states there is an arrow to a p world and to a $\neg p$ world. This ensures that for a step (or a move, in a game proof) in one of the two states, there is an equivalent step in the other.

As we can prove this for arbitrary s, we may conclude: there are actions α and β, which are not bisimilar ($\alpha \not\sim \beta$), such that for all bisimilar states s, s' they give bisimilar results (i.e. $s \sim s'$ implies $s.\alpha \sim s'.\beta$).

Fact 5 *If two actions α and α' give bisimilar results (updates) on bisimilar states, then that does not guarantee that they are bisimilar: if for all s, s', $s \sim s'$ implies $s.\alpha \sim s'.\alpha'$, then not necessarily $\alpha \sim \alpha'$.*

As we are ultimately interested in what actions actually change in given states, we define another relation between action structures, which does have the property that two action structures are related if and only if the updates on bisimilar states is bisimilar again. We are able to define this relation as a bisimulation relation, using the definition of twin-bisimilarity, which relates *pairs* of states and actions.

Definition 6 *A pair (s, α) of a state and an action structure is* twin-bisimilar *with another such pair (t, β) iff:*

- *$\langle\!\langle C_\alpha \rangle\!\rangle(s) = \langle\!\langle C_\beta \rangle\!\rangle(t)$, or both $\langle\!\langle C_\alpha \rangle\!\rangle(s)$ and $\langle\!\langle C_\beta \rangle\!\rangle(t)$ are undefined;*
- *for each $s \longrightarrow_a s'$ and $\alpha \longrightarrow_a \alpha'$ there is $t \longrightarrow_a t'$ and $\beta \longrightarrow_a \beta'$ s.t. (s', α') is twin-bisimilar to (t', β');*
- *and vice versa.*

From this we can naturally define the notion of *t-bisimulation* relations on actions:

Definition 7 *Two actions α and α' are* t-bisimilar *iff the pairs (s, α) and (s', α') are twin-bisimilar for all bisimilar states s, s'. Notation: ($\alpha \sim_t \alpha'$.)*

Lemma 8 *Comparing the "normal" bisimilarity with t-bisimilarity, we have the following observations:*

- *if $\alpha \sim \alpha'$ then $\alpha \sim_t \alpha'$;*
- *if $s \sim s'$ and $\alpha \sim_t \alpha'$ then $s.\alpha \sim s'.\alpha'$;*
- *if for all $s \sim s'$ we have that $s.\alpha \sim s'.\alpha'$, then $\alpha \sim_t \alpha'$.*

Note that the earlier examples α and β that were not action bisimilar but had the same update effect are indeed twin bisimilar.

Finally, we are interested in the relation between states that behave the same under update with a given action α. This relation is given by the following definition:

Definition 9 *Let s, t be states, and let α be a (pointed) action. s and t are α-*update bisimilar *(notation $s \sim_\alpha t$) iff*

- *either $C_{\alpha_1}(V_{s_1})$, $C_{\alpha_1}(V_{t_1})$ both defined, and $\langle\!\langle C_{\alpha_1} \rangle\!\rangle(s) = \langle\!\langle C_{\alpha_1} \rangle\!\rangle(t)$, or $\langle\!\langle C_{\alpha_1} \rangle\!\rangle(s)$ and $\langle\!\langle C_{\alpha_1} \rangle\!\rangle(t)$ both undefined,*
- *if $s \to_a s'$ and $\alpha \to_a \alpha'$ then there is a t' with $t \to_a t'$ and $s' \sim_{\alpha'} t'$, or $s'.\alpha'$ is undefined;*
- *and vice versa.*

Lemma 10 *For all states s, t, all pointed actions α: $s \sim_\alpha t$ iff $s.\alpha \sim t.\alpha$.*

Proof We will prove for the base case and the coinduction step of both definitions that they are equivalent. For the base case, note that $V_{s.\alpha} = \langle\!\langle C_\alpha \rangle\!\rangle(s)$. Hence $\langle\!\langle C_\alpha \rangle\!\rangle(s) = \langle\!\langle C_\beta \rangle\!\rangle(t)$ iff $V_{s.\alpha} = V_{t.\alpha}$. For the step, note that $s.\alpha \longrightarrow_a s'.\alpha'$ iff $s \longrightarrow_a s'$ and $\alpha \longrightarrow_a \alpha'$ and $s.\alpha, s'.\alpha'$ defined. Now suppose $s.\alpha \longrightarrow_a s'.\alpha'$ but there is no $t'.\alpha'$ s.t. $t.\alpha \longrightarrow_a t'.\alpha'$. That means, there is no $t \longrightarrow_a t'$ s.t. $t'.\alpha' \sim s'.\alpha'$. Hence no $t \longrightarrow_a t'$ s.t. $s' \sim_\alpha t'$, and vice versa.
$\square$

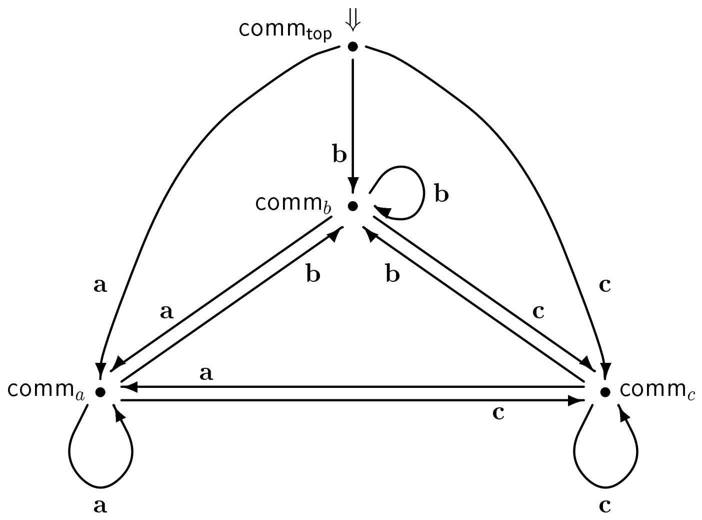
9 The Sending of a Message as an Action

It is time to focus again on the action of sending messages, possibly encrypted and signed. We will define for examples of such actions, that is, for messages $m \in \mathsf{Message}$, a translation as an action expression. It will turn out that there are several translation options available. This is a familiar feature about formalisation of informal notions: the formal rigour imposed by the description medium acts as an incentive to make up one's mind about the informal notions.

9.1 Awareness Actions

In the example of Othello, we have seen that Othello is not aware of what action Iago really suspects (knows) to happen. Othello thinks that Iago — just like himself — suspects nothing, i.e. Skip. In the picture this is indicated by the i-arrow from Skip to itself, rather than to the top node. In order to be able to define fully mutually transparant actions, we now first define an action form where all agents *are* aware of each other's suspected actions.

We envision the situation where each agent a suspects a command comm_a to be happening, possibly different from what is really happening, and from what other agents suspect to be happening. The command $\mathsf{comm}_{\mathsf{top}}$ denotes what is actually happening, that is, what is applied to the top world of the model. In general, for a group of agents to be aware of each other's suspicions, we would need arrows from and to each of the actions of the agents. The picture should look like (or be bisimilar to) a fully connected graph, where the nodes are the commands comm_a each agent a suspects, and arrows to the command of a are labelled a. On top of that there is a top node, with the "real" command $\mathsf{comm}_{\mathsf{top}}$ that is happening in the top world; this node would only have arrows to all other nodes. For three agents, a, b, c, with each one command that they suspect, that would look like the action below.



In this picture, we can walk from the top node over an a-arrow and a c-arrow to comm_c, which should be interpreted as "a believes that c believes that comm_c is happening". Similarly, one can "see" in the graph statements like "b believes that a believes that b believes that comm_b is happening". Note that every node has outgoing arrows to all other nodes (apart from the top node), labelled by the agent to whose command the arrow leads.

For expressing this general action structure as an action expression (for any set of agents Agent and given $\mathsf{comm}_a, \mathsf{comm}_b$, etc.), we first recall the intuitive meaning of the μ-operator as the introduction of a placeholder. In the recursive definition below we introduce a placeholder x_a for every node comm_a, by keeping track of the agents whose commands do not have a placeholder yet. The idea s that every action subexpression representing a node in the picture has a suspicion x_a^a for all agents a in the range of "their" μx_a operator. For the others, this must still be introduced. Using a recursive definition, which starts from the set of all agents Agent, we obtain the definition of what we call *aware suspicion*. (We use the notation $\sum_{a \in A}$ for the "+" operator on actions.)

Definition 11 *The action* aware suspicion *of a group of agents a that each have one command* comm_a *they suspect to happen, while the command* $\mathsf{comm}_{\mathsf{top}}$ *takes place in the top node, is defined as* $\mathsf{aware\_suspicion}(\mathsf{comm}_{\mathsf{top}}, (a, \mathsf{comm}_a)_{a \in \mathsf{Agent}}) := \mathsf{act}_{\mathsf{top}}$ *where*

$$\mathsf{act}_{\mathsf{top}} := \mathsf{comm}_{\mathsf{top}} + \sum_{a \in \mathsf{Agent}} (\mathsf{act}_a(\mathsf{Agent}))^a, \text{ and, for any set } A \subseteq \mathsf{Agent} :$$
$$\mathsf{act}_a(A) := \mu x_a.\mathsf{comm}_a + \sum_{b \notin A \setminus \{a\}} (x_b)^b + \sum_{b \in A \setminus \{a\}} (\mathsf{act}_b(A \setminus \{a\}))^b$$

To improve readability and comprehension of the action expressions, we will restrict ourselves to the situation where the number of agents is three. The formulas can be extended to larger numbers of agents.

9.2 Assumptions on Belief and Trust

For the setting of cryptographic communication, we use a type of doxastic models, that is, models that are suited to reason about belief, rather than knowledge (see [4] on doxastic logic). Exactly in the case of secret communication, agents may have wrong beliefs about what is being communicated or about the effects the communication has.

We assume the usual belief axioms D45:

$$\vdash \Diamond_a \mathsf{True}$$
$$\vdash \Diamond_a \Diamond_a \varphi \rightarrow \Diamond_a \varphi$$
$$\vdash \Diamond_a \varphi \rightarrow \Box_a \Diamond_a \varphi$$

These axioms correspond to properties of models and their underlying frames:

Lemma 12 *For a given frame* F*,*

- *all models on* F *satisfy axiom D iff* F *is serial;*
- *all models on* F *satisfy axiom 4 iff* F *is transitive;*
- *all models on* F *satisfy axiom 5 iff* F *is euclidean.*

In general, models that satisfy one or more of these axioms need not have an underlying frame with the corresponding properties, but one can always find a bisimilar model that does have such a frame.

Furthermore, for every pair of agents a, b we have the following axiom for the trust relation, which we have discussed in Section 3:

$$\vdash (\mathsf{Trust}ab \leftrightarrow \Box_a \mathsf{Trust}ab) \wedge (\neg\mathsf{Trust}ab \leftrightarrow \Box_a \neg\mathsf{Trust}ab)$$

This axiom states that every agent has right beliefs about her own trust or distrust in other agents.

Unfortunately, it is not possible to formulate a corresponding *frame* property for the trust axiom, but we can formulate the requirements on the models by help of the following definition. The idea is that if $\mathsf{Trust}ab$ is true in some world in a model, it should also be true in all worlds that are a-related to that world, that is, the worlds that a considers possible. And reversely, if $\mathsf{Trust}ab$ is false there, it should be false in all of a's possible worlds. We say that a *controls* her own trust relations. We define this property for any proposition letter (recall that $\mathsf{Trust}ab$ is among them: $\mathsf{Trust}ab \in \mathsf{Prop}$).

Definition 13 *For a state s and a proposition letter $p \in \mathsf{Prop}$, we say that an agent $a \in \mathsf{Agent}$* controls *p in s iff for all states t reachable from s, we have that*

- *$p \in V(t)$ iff for all t' such that $t \longrightarrow_a t'$, $p \in V(t')$;*
- *$p \notin V(t)$ iff for all t' such that $t \longrightarrow_a t'$, $p \notin V(t')$.*

For a model W, we say that a controls p in W if a controls p in all *states $s = (\mathsf{W}, w)$ in the model.*

Note that this must hold also for states that are unreachable by a itself. If you view a model as a group of (disconnected) "a-clusters" of worlds internally connected by a-arrows, then the value of p must be the same within each a-cluster, but may differ in different a-clusters.

Now we define the models we need for our analysis of crypto actions:

Definition 14 *A model W is a* doxastic trust model *iff*

- *there exists a bisimilar model that is serial, transitive and euclidean;*
- *for all pairs of agents $a, b \in \mathsf{Agent}$, a controls $\mathsf{Trust}ab$ in W.*

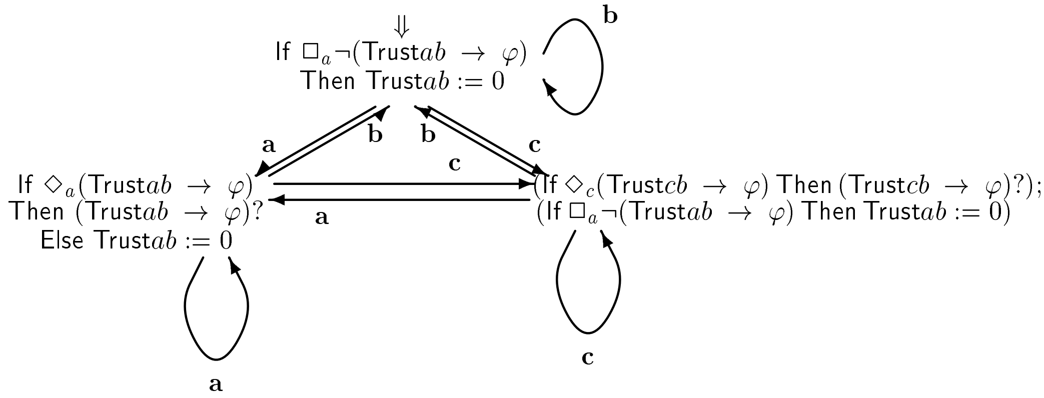
For such a W, we call a state (pointed model) $s = (\mathsf{W}, w)$ a pointed doxastic trust model or a doxastic trust state.

For the actions we define in the following subsections, we will prove that they preserve the properties as defined here, i.e. the update applied to a (pointed) doxastic trust model will also return a (pointed) doxastic trust model.

9.3 Sending Signed Formulas with Addressee

As a first case, let us discuss a possible translation of $\mathsf{Sign}^a_b\varphi$, where φ is a formula. This message is addressed to a, and signed by b, but every agent (also c) can see the contents. The sending of such a message can be seen as an announcement, and we model it here as an action during which the following events are happening. Agent a learns φ in the worlds where she trusts b, that is, she learns the implication ($\mathsf{Trust}ab \rightarrow \varphi$), unless that would lead to inconsistency (if in all her possible worlds $\mathsf{Trust}ab$ *and* $\neg\varphi$ are true), in which case she changes her trust in b into distrust. In the mean time, both b and the third agent c change the $\mathsf{Trust}ab$ value according to their own beliefs of a's beliefs. At the same time, c (not b) learns the formula if he can and if he trusts b (so agent c learns the implication $\mathsf{Trust}cb \rightarrow \varphi$ if that does not lead to inconsistencies), but c does not change his trust in b if he cannot learn it (concluding that b may only be untrustworthy to a). All three of the agents are aware of each other's actions.

This is in fact an aware suspicion action as defined in definition 11. Because b's suspicion is exactly what happens in the top-world, we can simplify the picture of Section 9.1 to the following bisimilar variant:



Note that the condition $\Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi)$ is exactly the negation of the condition $\Diamond_a(\mathsf{Trust}ab \rightarrow \varphi)$, so all if-then-else commands in the above action set $\mathsf{Trust}ab := 0$ in all worlds when the former condition is true and the latter is false. The condition on when the value of $\mathsf{Trust}ab$ is changed in the top world should be related to a's belief on the condition of changing it, because we aim to preserve doxastic trust models (in which a controls $\mathsf{Trust}ab$). We will dicuss this below. Note also that the command in the top token (If $\Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi)$ Then $\mathsf{Trust}ab := 0$) is defined for all states, so for every state, the update with the total action is defined. The other two tokens contain tests of the form $(\mathsf{Trust}xb \rightarrow \varphi)$? of which the updates are undefined in states where their test condition is false, hence these tokens may "delete" some worlds in the update. The condition of the if-statement, however, prevents that *all* possible worlds of either a or c would be eliminated: it tests exactly for the presence of at least one world that would survive this test. We will use this observation below, in the proof of the well-formedness of the updated model.

In a formula, we can describe the above complex action as follows (for readability, we added abbreviations for each of the commands in the three different action tokens):

$$
\begin{aligned}
&(\mu x.\mathsf{comm}_b + x^b + \\
&\quad (\mu y.\mathsf{comm}_a + x^b + y^a + (\mu z.\mathsf{comm}_c + x^b + y^a + z^c)^c)^a + \\
&\quad (\mu z.\mathsf{comm}_c + x^b + z^c + (\mu y.\mathsf{comm}_a + x^b + z^c + y^a)^a)^c
\end{aligned}
$$

where

$$
\begin{aligned}
\mathsf{comm}_a &:= \text{If } \Diamond_a(\mathsf{Trust}ab \rightarrow \varphi) \text{ Then } (\mathsf{Trust}ab \rightarrow \varphi)? \text{ Else } \mathsf{Trust}ab := 0 \\
\mathsf{comm}_b &:= \text{If } \Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi) \text{ Then } \mathsf{Trust}ab := 0 \\
\mathsf{comm}_c &:= (\text{If } \Diamond_c(\mathsf{Trust}ac \rightarrow \varphi) \text{ Then } (\mathsf{Trust}ac \rightarrow \varphi)?); (\text{If } \Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi) \text{ Then } \mathsf{Trust}ab := 0)
\end{aligned}
$$

This action expression is equivalent to $\mathsf{aware\_suspicion}(\mathsf{comm}_b, (a, \mathsf{comm}_a), (b, \mathsf{comm}_b), (c, \mathsf{comm}_c))$.

Lemma 15 *The action of sending a signed formula with addressee, as described above, preserves doxastic trust models.*

Proof We give the proof here for the case of three agents, but this can be easily extended to the same result for the extension of the action to a larger number of agents. Let α be the above defined action of sending a signed formula with addressee. Let $s = (\mathsf{W}, w_0)$ be a doxastic trust model, without loss of generality we assume that W is serial, transitive and euclidean and that for all pairs of agents d, d', $\mathsf{Trust}dd'$ is controlled by d in W. We will prove that $s.\alpha$ is a doxastic trust model as well. Note that the action structure of α is serial, transitive and euclidean itself (for all three relations $\longrightarrow_a$, $\longrightarrow_b$ and $\longrightarrow_c$). Observe that a given $u \in s.\alpha$ (i.e. a world in the underlying model of the update) must be of the form $w.k$ for a world $w \in \mathsf{W}$ and action token k.

For seriality of $s.\alpha$, let such a $w.k$ be given. As W is serial and α is, we can find for every agent d a w'_d and k'_d s.t. $w \longrightarrow_d w'_d$ and $k \longrightarrow_d k'_d$. For $d = b$, k_b is the top-action in α (comm_b in the definition), as that is the only action token where all b-arrows point to. As we observed, it can never fail (its update is defined for every possible state), so $w'_b.k'_b$ is defined, and (see definition update) $w.k \longrightarrow_b w'_b.k'_b$. For $d = a$, we first note that k_a must be comm_a in the above definition. That action can only fail in state (W, w'_a) if the if-then-else condition ($\Diamond_a(\mathsf{Trust}ab \rightarrow \varphi)$) holds in w'_a, but the test doesn't (so $\neg(\mathsf{Trust}ab \rightarrow \varphi)$ holds). So if $w'_a.k'_a$ is undefined, there is another world $w''_a \in \mathsf{W}$ with $w'_a \longrightarrow_a w''_a$ which does satisfy the test ($\mathsf{Trust}ab \rightarrow \varphi$). In other words, $w''_a.k'_a$ is defined. Because W is transitive, we have $w \longrightarrow_a w''_a$ as well. So, by definition of the update, we have $w.k \longrightarrow_a w''_a.k'_a$. A similar argument works for the case of c.

Transitivity and euclidicity of $s.\alpha$ follows directly from transitivity and euclidicity of s and α themselves.

Now we prove that for all pairs of agents d, d', $\mathsf{Trust}dd'$ is controlled by d in $s.\alpha$. Suppose it doesn't hold. Let agents d, d' and worlds $w.k, w'.k'$ reachable from $s.\alpha$ be given such that $w.k \longrightarrow_a w'.k'$ and the value of $\mathsf{Trust}dd'$ in $w.k$ differs from the value in $w'.k'$. Note that for all action tokens in α it holds that the value of $\mathsf{Trust}dd'$ only (and exactly) changes if $d = a, d' = b$ and if $\Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi)$ holds in the original state, in which case the value is set to 0. So for d, d' different than that, the values of $\mathsf{Trust}dd'$ are already different before the action update, in w and w'. But this contradicts the assumption that s is a doxastic trust model. For $d = a, d' = b$, we consider the case that in w it holds that $\Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi)$ and in w' it doesn't (or vice versa). But this contradicts transitivity and euclideanity of s, which together guarantee that for every world w'', $w \longrightarrow_a w'$ iff $w' \longrightarrow_a w$. The value of $\mathsf{Trust}ab$ can also not already have been different before the update, in s, as that contradicts the assumption that s is a doxastic trust model. All options lead to contradiction, hence for all pairs of agents d, d', $\mathsf{Trust}dd'$ is controlled by d in $s.\alpha$.

$\Box$

9.4 Sending Signed Formulas Without Addressee

In our language we also allow for b sending a signed formula, but without specifying an addressee. The message then looks like $\mathsf{Sign}_b\varphi$. In the modelling of the sending of such message, we will interpret it here as a message to *all* agents, so both agents a and c will learn the implication $\mathsf{Trust}ab \rightarrow \varphi$ and $\mathsf{Trust}cb \rightarrow \varphi$ respectively, if they can, or otherwise give up their trust in b. Again, all agents are aware of what the others do, and update the trust relations of the other two agents according to their own beliefs about the others' beliefs.

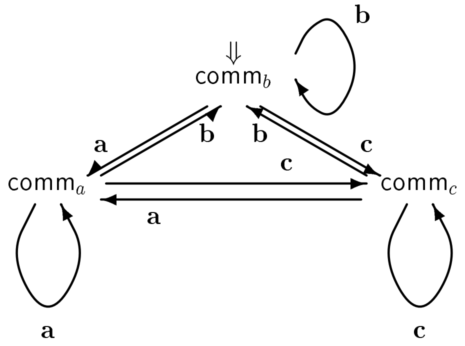
In structure, the action for this is the same as the one for $\mathsf{Sign}^a_b\varphi$ but with different commands in the three action tokens:

$$
\begin{aligned}
&(\mu x.\mathsf{comm}_b + x^b + \\
&\quad (\mu y.\mathsf{comm}_a + x^b + y^a + (\mu z.\mathsf{comm}_c + x^b + y^a + z^c)^c)^a + \\
&\quad (\mu z.\mathsf{comm}_c + x^b + z^c + (\mu y.\mathsf{comm}_a + x^b + z^c + y^a)^a)^c
\end{aligned}
$$

where

$$
\begin{aligned}
\mathsf{comm}_a &:= (\mathsf{If}\ \Diamond_a(\mathsf{Trust}ab \rightarrow \varphi)\ \mathsf{Then}\ (\mathsf{Trust}ab \rightarrow \varphi)?\ \mathsf{Else}\ \mathsf{Trust}ab := 0);\\
&\quad\ (\mathsf{If}\ \Box_c\neg(\mathsf{Trust}cb \rightarrow \varphi)\ \mathsf{Then}\ \mathsf{Trust}cb := 0)\\
\mathsf{comm}_b &:= (\mathsf{If}\ \Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi)\ \mathsf{Then}\ \mathsf{Trust}ab := 0); (\mathsf{If}\ \Box_c\neg(\mathsf{Trust}cb \rightarrow \varphi)\ \mathsf{Then}\ \mathsf{Trust}cb := 0)\\
\mathsf{comm}_c &:= (\mathsf{If}\ \Diamond_c(\mathsf{Trust}ac \rightarrow \varphi)\ \mathsf{Then}\ (\mathsf{Trust}ac \rightarrow \varphi)?\ \mathsf{Else}\ \mathsf{Trust}cb := 0);\\
&\quad\ (\mathsf{If}\ \Box_a\neg(\mathsf{Trust}ab \rightarrow \varphi)\ \mathsf{Then}\ \mathsf{Trust}ab := 0)
\end{aligned}
$$

In a picture this has the same structure seen before (we now use comm_x for abbreviating the commands in the picture).



Again, this action is bisimilar to the action defined by $\mathsf{aware\_suspicion}(\mathsf{comm}_b, (a, \mathsf{comm}_a), (b, \mathsf{comm}_b), (c, \mathsf{comm}_c))$.

Lemma 16 *The action of sending a signed formula without addressee, as described above, preserves doxastic trust models.*

Proof The proof goes analogous to the proof of Lemma 15. For seriality we observe again that not all possible worlds of an agent will be deleted, as the test is only performed if there is at least one world surviving it. The pattern for changing the value of the Trust relation is the same as in the action mentioned in Lemma 15, and the argument for preserving the control is likewise. The action is transitive and euclidean, hence those properties are also preserved in the model.
□

9.5 Sending Encrypted Formulas

Another example of the sending of a message that we can interpret by an action expression is the sending of an encrypted formula $\mathsf{Encr}_a\varphi$. An example of an interpretation is the action during which agents who cannot decrypt the message do not know that a message is passing (one could think of them hearing some background noise which they cannot distinguish from the passing of an encrypted message), while at the same time the agent a who can decipher the encrypted formula, cannot see who sent the message, but may learn that formula if she trusts all others and has at least one possible world in which the formula is true (which would survive the learning). She is aware of the fact that other agents do not know she receives something at all.

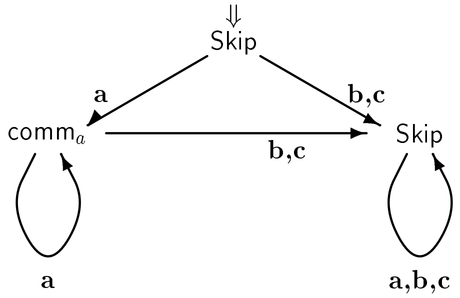
As other agents do not know that a may learn something, this action is clearly not an aware suspicion as defined in Definition 11. Note that in the top world nothing changes, only a's beliefs possibly change. The action expression representing this action for three agents looks as follows:

$$
\begin{aligned}
& (\mu x.\mathsf{comm}_a + x^a + (\mu y.y^a + y^b + y^c)^b + (\mu y.y^a + y^b + y^c)^c)^a + \\
& \quad (\mu y.y^a + y^b + y^c)^b + \\
& \quad (\mu y.y^a + y^b + y^c)^c
\end{aligned}
$$

where

$$\mathsf{comm}_a \quad := \quad \mathsf{If}\ \Diamond_a((\mathsf{Trust}ab \wedge \mathsf{Trust}ac) \ \rightarrow\ \varphi)\ \mathsf{Then}\ ((\mathsf{Trust}ab \wedge \mathsf{Trust}ac) \ \rightarrow\ \varphi)?$$

In a picture, we can represent this as follows:



Again, analogously to Lemma 15 we can state that this action preserves the doxastic models we are using.

Lemma 17 *The action of sending an encrypted formula, as described above, preserves doxastic trust models.*

As we mentioned, this is only one interpretation of what happens when an encrypted message is sent. If the number of possible messages and the number of agents are both finite, one could also choose an interpretation where agents suspect that someone got some formula, and therefore consider all combinations of destined agents and formulas possible.

9.6 Sending Encrypted Signed Formulas

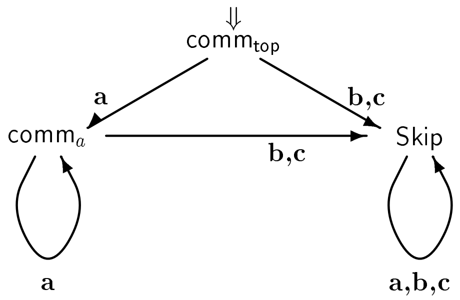
As a last case we consider the sending of an encrypted signed formula $\mathsf{Encr}_a\mathsf{Sign}_b\varphi$. The difference with the anonymous encryption is that the recipient now treats signing as she would if it were just a signed message, but she knows that no-one else is aware of her learning or trust changing actions.

$$
\begin{aligned}
&\mathsf{comm_{top}}+\\
&\quad(\mu x.\mathsf{comm}_a + x^a + (\mu y.y^a + y^b + y^c)^b + (\mu y.y^a + y^b + y^c)^c)^a+\\
&\quad(\mu y.y^a + y^b + y^c)^b+\\
&\quad(\mu y.y^a + y^b + y^c)^c
\end{aligned}
$$

where

$$
\begin{aligned}
\mathsf{comm}_a &:= \mathsf{If}\ \Diamond_a(\mathsf{Trust}ab \rightarrow \varphi)\ \mathsf{Then}\ (\mathsf{Trust}ab \rightarrow \varphi)?\ \mathsf{Else}\ \mathsf{Trust}ab := 0\\
\mathsf{comm_{top}} &:= \mathsf{If}\ \Box_a\neg((\mathsf{Trust}ab \wedge \mathsf{Trust}ac) \rightarrow \varphi)\ \mathsf{Then}\ \mathsf{Trust}ab := 0
\end{aligned}
$$

In a picture, we can represent this as follows:



Again, this action preserves the doxastic models we are using.

Lemma 18 *The action of sending an encrypted signed formula, as described above, preserves doxastic trust models.*

Similarly we could treat the case of encrypted signatures with addressee as follows: $\mathsf{Encr}_a\mathsf{Sign}_b^a\varphi$ we could treat like $\mathsf{Encr}_a\mathsf{Sign}_b\varphi$ (described above), and $\mathsf{Encr}_a\mathsf{Sign}_b^c\varphi$ could be Skip for all agents.

10 Reasoning Principles

In this section, we state a number of sound reasoning principles, which use some new functions on action expressions.

The Precondition of an Action The function PRE is a function from action expressions to formulas, such that $\mathsf{PRE}(\alpha)$ represents the precondition of an action α (describing the states on which it is defined). PRE is defined recursively for action expressions:

$$
\begin{array}{rcl}
\mathsf{PRE} & :: & \mathsf{Action} \longrightarrow L \\
\mathsf{PRE}(A) & := & \begin{cases} \mathsf{True} & \text{if } A \text{ consistent} \\ \mathsf{False} & \text{otherwise.} \end{cases} \\
\mathsf{PRE}(\mathsf{If}\,\varphi\,\mathsf{Then}\,C_1\,\mathsf{Else}\,C_2) & := & (\varphi \rightarrow \mathsf{PRE}(C_1)) \wedge (\neg\varphi \rightarrow \mathsf{PRE}(C_2)) \\
\mathsf{PRE}(\alpha + \beta) & := & \mathsf{PRE}\alpha \wedge \mathsf{PRE}\beta \\
\mathsf{PRE}(\alpha^a) & := & \mathsf{True} \\
\mathsf{PRE}(\alpha \circ \beta) & := & \mathsf{PRE}\alpha \wedge [\alpha]\mathsf{PRE}\beta \\
\mathsf{PRE}(\mu x.\alpha(x)) & := & \mathsf{PRE}(\alpha(\mu x.\alpha(x)))
\end{array}
$$

In other words, the precondition of an assignment list is its being consistent (it can update any state if it is consistent, otherwise it fails on any state), and the precondition of an if-then-else command can be read as "either φ and the precondition of C_1, or $\neg\varphi$ and the precondition of C_2". Recall that our '+'-operator is in fact a sort of parallel composition, so the precondition of $\alpha+\beta$ is the conjunction of the preconditions of α and β. For suspicion we do not need any precondition (as it refers to what an agent suspects happening in her own possible worlds, not in the current one). Unsurprisingly, the precondition of the sequential composition of α and β is defined as the conjunction of the precondition of α and the formula denoting that "after α, the precondition of β holds". The precondition of a μ-expression may look as a circular definition, but it is not: recall from the definition of action expressions that the μ variable x is only to be used in the form x^a, that is, under a suspicion operator. As $\mathsf{PRE}(x^a)$ is defined as True, the recursion stops before reaching the x again.

The Change After an Action We define a function CH which gives for every proposition letter p and action α a formula which represents the value of p after α.

$$
\begin{array}{rcl}
\mathsf{CH} & :: & \mathsf{Prop} \longrightarrow (\mathsf{Action} \longrightarrow L) \\
\mathsf{CH}_p(A) & := & \begin{cases} \mathsf{True} & \text{if } p \in A^+ \setminus A^- \\ \mathsf{False} & \text{if } p \in A^- \setminus A^+ \\ p & \text{otherwise.} \end{cases} \\
\mathsf{CH}_p(\mathsf{If}\,\varphi\,\mathsf{Then}\,C_1\,\mathsf{Else}\,C_2) & := & (\varphi \rightarrow \mathsf{CH}_p(C_1)) \wedge (\neg\varphi \rightarrow \mathsf{CH}_p(C_2)) \\
\mathsf{CH}_p(\alpha) & := & \mathsf{CH}_p(\mathsf{topcomm}(\alpha)) \text{ for other actions } \alpha
\end{array}
$$

So, the changed value of p after assignment list A is True if A sets p to True (and not to False as well); and it is False if A sets it to False (and not to True). Otherwise it keeps the current value p. In fact, in case $p \in A^+ \cap A^-$, A is inconsistent, and the update with A is undefined for any state. In particular, the value of p after the action is undefined. Nevertheless, we choose to define the change to p here, in order to always have a defined formula. In our principles we will take care of the undefined case by using the PRE function.

Suspicions of Agents during an Action The function SUSP gives for an agent and a given action expression all subexpressions which that agent considers possible, in other words: all suspicions of that agent.

$$
\begin{array}{rcl}
\mathsf{SUSP} & :: & \mathsf{Agent} \longrightarrow (\mathsf{Action} \longrightarrow \mathcal{P}(\mathsf{Action})) \\
\mathsf{SUSP}_a(C) & := & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{SUSP}_a(\alpha^a) & := & \{\alpha\} \\
\mathsf{SUSP}_a(\alpha^b) & := & \emptyset \text{ for } a \neq b \\
\mathsf{SUSP}_a(\alpha + \beta) & := & \mathsf{SUSP}_a\alpha \cup \mathsf{SUSP}_a\beta \\
\mathsf{SUSP}_a(\alpha \circ \beta) & := & \{\alpha' \circ \beta' \mid \alpha' \in \mathsf{SUSP}_a\alpha, \beta' \in \mathsf{SUSP}_a\beta\} \\
\mathsf{SUSP}_a(\mu x.\alpha(x)) & := & \mathsf{SUSP}_a\alpha(\mu x.\alpha(x))
\end{array}
$$

Again, note that the definition on μ-expressions is not circular, as variables x can only occur under suspicion operators, and the suspicion case is a base case (non-recursive case) in our definition. Hence, the μ expression may occur in the subexpressions of the final function result, but the SUSP function is not called again on them.

Note also that the result of the function is always a set of *subexpressions* of the given action expression, and this set is (therefore) always *finite*.

Axioms for Actions Using the above (syntactical) functions, we now state axioms about actions for our logic, which can be treated as sound reasoning principles. In fact, apart from the first and the last one, they give equivalences of formulas of the form $[\alpha]\varphi$, by induction on φ.

$$
\begin{array}{l}
\vdash [\alpha](\varphi \rightarrow \psi) \rightarrow ([\alpha]\varphi \rightarrow [\alpha]\psi) \\
\vdash [\alpha]\mathsf{False} \leftrightarrow \neg\mathsf{PRE}(\alpha) \\
\vdash [\alpha]p \leftrightarrow (\mathsf{PRE}(\alpha) \rightarrow \mathsf{CH}_p(\alpha)) \\
\vdash [\alpha]\neg\varphi \leftrightarrow (\mathsf{PRE}(\alpha) \rightarrow \neg[\alpha]\varphi) \\
\vdash [\alpha](\varphi \wedge \psi) \leftrightarrow ([\alpha]\varphi \wedge [\alpha]\psi) \\
\vdash [\alpha](\varphi \rightarrow \psi) \leftrightarrow (\mathsf{PRE}(\alpha) \rightarrow ([\alpha]\varphi \rightarrow [\alpha]\psi) \\
\vdash [\alpha]\Box_a\varphi \leftrightarrow (\mathsf{PRE}(\alpha) \rightarrow \bigwedge_{\beta \in \mathsf{SUSP}_a(\alpha)} \Box_a[\beta]\varphi) \\
\vdash [\alpha][\beta]\varphi \leftrightarrow [\alpha \circ \beta]\varphi \\
\vdash \neg\mathsf{PRE}(\alpha) \rightarrow [\alpha]\varphi
\end{array}
$$

The first principle expresses the well-known normality rule (K-axiom) for all modal logics: the distribution of the $[\alpha]$-modality over implication. The second principle expresses the *partial functionality* of the $[\alpha]$-modality: if the precondition of an action does not hold, then (and exactly then) "anything" holds after that action. So, updating with an action is a partial function. In almost all other principles, the $\mathsf{PRE}(\alpha)$ plays a similar role: for example $[\alpha]p$ is equivalent to the changed value of p after α — if $\mathsf{PRE}(\alpha)$ holds. We call this the *atomic change principle*. In conjunctions after actions, the condition on $\mathsf{PRE}(\alpha)$ is not needed, as both $[\alpha]\varphi \wedge [\alpha]\psi$ and $[\alpha](\varphi \wedge \psi)$ follow from $\neg\mathsf{PRE}(\alpha)$. The knowledge rule deserves special attention: it says that a will be convinced of φ after α has taken place ($[\alpha]\Box_a\varphi$) exactly when a now already knows that φ would hold after all its suspected actions (the (finite) conjunction of $\Box_a[\beta]\varphi$, where β ranges over suspicions of a) — or $\mathsf{PRE}(\alpha)$ does not hold at all. Here it is important that the SUSP function indeed returns a finite set of action expressions. When using this as a rewriting system for a completeness proof (which we will not discuss in this paper), it is also important that the β's are subexpressions, so that the process stops after finitely many steps. Expressions of the form $[\alpha][\beta]\varphi$ ("after α it holds that φ holds after β") are simply equivalent to $[\alpha \circ \beta]\varphi$ ("φ holds after $\alpha \circ \beta$"). Finally, the last principle gives another formulation of partial functionality.

Axioms for Belief and Model Properties In our applications, we are typically interested in beliefs of agents with standard properties. For our doxastic logic, we wish to use the usual belief axioms D45, which express consistency in belief (no belief in False) and positive and negative introspection:

$$
\begin{array}{l}
\vdash \Diamond_a\mathsf{True} \\
\vdash \Diamond_a\Diamond_a\varphi \rightarrow \Diamond_a\varphi \\
\vdash \Diamond_a\varphi \rightarrow \Box_a\Diamond_a\varphi
\end{array}
$$

As is well-known from modal theory, these axioms correspond to properties of models and their underlying frames (a frame being the model structure without the particular valuations yet):

Fact 19 *For a given frame* F*,*

- *all models on* F *satisfy axiom D iff* F *is serial;*
- *all models on* F *satisfy axiom 4 iff* F *is transitive;*
- *all models on* F *satisfy axiom 5 iff* F *is euclidean.*

In general, models that satisfy one or more of these axioms need not have an underlying frame with the corresponding properties, but one can always find a bisimilar model that does have such a frame. We refer to such models as D45 models.

It is not sufficient to adopt these axioms in our dynamic doxastic logic, as we could construct actions such that *after the action* the axioms do not hold anymore, so those actions do not preserve D45-models. For example, $[\alpha]\Box_a\mathsf{False}$ can be achieved by choosing an α which does not contain any suspicions for a. It is beyond the scope of this paper to find sufficient and necessary conditions on action expressions for them to preserve D45 models, but the actions discussed in the examples do preserve them.

Principles for Crypto Communications We now sum up some reasoning principles which could apply to the particular case of actions that describe the passing of messages. As for the actions we have defined in Section 9, and for their extensions to a larger number of agents, we could think of the following rules:

$$
\begin{array}{l}
(\mathsf{Trust}ab \wedge \Diamond_a\varphi) \;\rightarrow\; ([\mathsf{Sign}_b^a\varphi]\Box_a\varphi) \\
\mathsf{Trust}ab \;\rightarrow\; (\Diamond_a\varphi \;\rightarrow\; [\mathsf{Sign}_b\varphi]\Box_a\varphi) \\
[\mathsf{Sign}_b^a\varphi]\neg\mathsf{Trust}ab \;\leftrightarrow\; (\Box_a\neg\varphi \vee \neg\mathsf{Trust}ab) \\
[\mathsf{Sign}_b\varphi]\neg\mathsf{Trust}ab \;\leftrightarrow\; (\Box_a\neg\varphi \vee \neg\mathsf{Trust}ab) \\
(\Box_a\neg\varphi \vee \neg\mathsf{Trust}ab) \;\rightarrow\; \Box_a p \;\leftrightarrow\; \Box_a[\mathsf{Sign}_b\varphi]p \text{ (for } p \neq \mathsf{Trust}ab) \\
[m]\mathsf{Trust}ab \;\rightarrow\; \mathsf{Trust}ab \\
\neg\mathsf{Trust}ab \;\rightarrow\; [m]\neg\mathsf{Trust}ab \\
\neg\mathsf{Trust}ab \;\rightarrow\; ([\mathsf{Encr}_a\mathsf{Sign}_b\varphi]\psi \;\leftrightarrow\; \psi) \\
[\mathsf{Encr}_b m]\Box_a p \;\leftrightarrow\; \Box_a p \text{ (for } a \neq b) \\
[\mathsf{Encr}_a\mathsf{Sign}_b\varphi]\Box_a p \;\leftrightarrow\; \Box_a[\mathsf{Sign}_b^a\varphi]p
\end{array}
$$

These principles reflect the way we have defined effects of messages, e.g. of a trusted agent one will learn statements if possible; if there is no trust after a message from an agent, then either there was not trust before, or trust was lost because the message was contradicting all possible worlds; if there is no trust or the statement is contradictory, then afterwards no facts of the world have changed (apart from the trust fact itself); trust can only be lost, not gained; agents who cannot decrypt a message do not learn anything from it; agents who can decrypt a signed message will learn from it exactly like they would learn from it if it were unencrypted and addressed to them.

These principles are just examples of rules that one may want to reason with. The principles can be proven with the semantical framework for action expressions that we have given. Analysing the properties of such actions is a way of analysing the security of communicated messages.

11 Conclusion

In this paper, we have introduced a language for epistemic action and change together with its formal semantics. The actions are considered as structures which, analogous to Kripke structures for static knowledge, represent beliefs about actions happening. The actions can be epistemic actions (in which case they show something about the world) or change actions, in which case they change the world, that is, the value of some of the proposition letters. With this framework it is possible to describe many examples of actions and the change of knowledge they involve. We have developed notions of bisimulation that show equivalences between the results of actions.

In general, we focused on the actions of sending signed or encrypted messages. For this, we introduced the notion of a doxastic trust model, which guarantees certain properties regarding trust between agents. We gave definitions of possible interpretations of the learning that is triggered by the sending of such messages, and showed that they preserved the properties of a doxastic trust model. Finally, we have given some sound principles of reasoning, and some conclusions on the particular interpretations of encrypted and signed messages earlier.

What still remains to be done is the development of a reasoning system in which the principles of Section 10 hold, together with a proof of completeness. Furthermore it will be interesting to explore other cases and settings of cryptographic communication. For that, it may be necessary to distinguish more between representation of the actual receiving of a message and that of its interpretation by the receiving agent.

Another line of future work lies in the direction of probability analysis: both state models and action models could be expanded in a straightforward way to probabilistic models. They could be used as models for a language in which one could express e.g. chances of breaking a cryptographic key, and the epistemics that would induce.

Acknowledgements

We wish to thank Alexandru Baltag, Johan van Benthem and Yde Venema for stimulating discussions of the issues raised in this paper.

References

[1] A. Baltag. A logic of epistemic actions. Manuscript, CWI, Fall 1999.

[2] A. Baltag, L.S. Moss, and S. Solecki. The logic of public announcements, common knowledge, and private suspicions. Technical report, CWI, Amsterdam, 1999.

[3] J. van Benthem. *Modal Correspondence Theory.* PhD thesis, University of Amsterdam, 1976.

[4] B.F. Chellas. *Modal Logic: An Introduction.* Cambridge University Press, 1980.

[5] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, 1990.

[6] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge.* MIT Press, 1995.

[7] J. Gerbrandy and W. Groeneveld. Reasoning about information change. *Journal of Logic, Language and Information*, 6:147–169, 1997.

[8] M. Hennesy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32, 1985.

[9] M. Hollenberg. *Logic and Bisimulation.* PhD thesis, Utrecht University, 1998.

[10] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[11] L.S. Moss. From hypersets to kripke models in logics of announcements. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, editors, *JFAK — Essays dedicated to Johan van Benthem on the Occasion of his 50th Birthday.* ILLC, Amsterdam, 1999.

[12] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. 5th GI Conference.* Springer-Verlag, 1981.

[13] A. Ponse, M. de Rijke, and Y. Venema, editors. *Modal Logic and Process Algebra.* Number 53 in CSLI Lecture Notes. CSLI, Stanford, 1996.

[14] Y. Shoham. *Reasoning about Change.* MIT Press, Cambridge, Mass, 1988.

[15] G.H. von Wright. *Practical Reason.* Blackwell, Oxford, 1983.