Proving Trust in Systems of Second-Order Processes: Preliminary Results*

Mads Dam
Swedish Institute of Computer Science

Abstract

We consider the problem of proving correctness properties for concurrent systems with features such as higher-order communication and dynamic resource generation. Operational models of security and authentication protocols based on the higher-order π-calculus are considered as examples. In this setting key features such as nonces/time stamps, encryption/decryption, and key generation can be modelled in a simple and abstract fashion using channel name generation and second-order process communication. A temporal logic is proposed as an appropriate logic for expressing correctness properties such as secrecy and authenticity. The logic is based on the modal μ-calculus with only greatest fixed points and universal next-state quantification. Extensions include first-order features to deal with names, and second-order features including function space constructions to deal with process input and output. One difficulty is that formulas need recursion in both covariant and contravariant positions. We show how this problem can be overcome in two different, but equivalent, ways, first using a standard semantical account based on intervals, and then by an iterative construction. We propose a predicate of trust in a monotonically increasing set of channels as an example, and establish structural decomposition principles for this predicate for concurrent composition and local channel declaration. On this basis a type system for trust inference can be derived quite easily, and we use this type system to establish trust of a very simple protocol example.

1 Introduction

In this paper we consider the problem of proving correctness properties in semantically very rich models of concurrent systems with features for communication of second- and higher-order objects (eg.: code), and for dynamically generating and communicating resource names. A long list of recent programming languages

*Work partially supported by Esprit BRA 8130 LOMAPS. Work partly done while visiting CMI, Université d'Aix Marseille 1

and models, including Java, CML, Facile, Oz, Actors, Erlang, and the π-calculus, explore these sorts of features to various extents. Typical of many applications written in these kinds of languages is that they are *open*, designed and intended to operate in environments that are possibly hostile, and at any rate only partially known at compile time. An important task is therefore to protect information and resources against intrusion, intended or otherwise. Intruders have at their disposal the full armoury usually considered in the field of computer security: they can steal messages, tamper with messages, crack codes, synthesize messages, store and replay messages, and much more. In the presence of higher-order communication they can even generate programs (viruses) that will be activated dynamically by the receiving agent. The question we address is how, in spite of this, we can prove that a system nonetheless performs correctly.

Key management protocols Classical key management protocols such as the Needham-Schroeder protocol provide excellent examples of programs designed to work reliably in face of hostile intruders. A typical purpose of a key management protocol is for participants to agree on a secret session key given some initial amount of trusted information. In this paper we show how such protocols can fruitfully be viewed as higher-order communicating processes and show some initial ideas as to how, on this basis, they can be verified. The idea is best introduced through a simple example, a modified version of the corrected version of the Andrew remote RPC protocol as introduced by Needham, Abadi, and Burrows [7]. This protocol is extremely simple, yet it introduces all the features needed to account for a whole class of key management protocols. Initially the two participants, A and B, share a private key K_{ab}. The task is to agree on a new session key. Using standard notation the protocol can be described as the exchange between A and B of the following three message sequence:

1. $A \to B : \{N_a\}_{K_{ab}}$
2. $B \to A : \{N_a, K'_{ab}\}_{K_{ab}}$
3. $A \to B : \{N_a\}_{K'_{ab}}$

In step (1) A transmits the nonce N_a to B, signalling his intent to generate a new session key. B responds, in step (2), by returning the new session key K'_{ab}, along with N_a to authenticate the message, both encrypted using the old key K_{ab}. In step (3), A returns the nonce N_a to B, this time encrypted using the new key K'_{ab}, serving as an acknowledgement to B that the previous message was received and decrypted.

Modelling key management protocols as second-order processes The protocol involves the following features:

1. Value passing and data type operations: Given a (possibly composite) message m it is possible to communicate m from A to B. It is also possible, given messages m_1 and m_2, to form the pair (m_1, m_2).

2. Private key encryption and decryption: Given a message m and a private key K, we can form $\{m\}_K$, m encrypted using key K. Also, given $\{m\}_K$ and given K we can decrypt to extract the message m.

3. Key generation. B has the capability of generating a new private key K'_{ab}, by assumption distinct from any other key known to any other participant in the exchange, friendly or hostile.

4. Nonce generation. It is possible to generate a fresh, non-composite piece of information, by assumption distinct from any other such pieces of information possessed by other participants.

We propose accounting for these features in the following fashion:

- Nonces are names as in the π-calculus [13]. New names are declared by the binding $\nu a.A$ introducing a as a new name with scope initially extending over A but not further. In $\nu a.A$, a will by definition be distinct from any other name occurring freely in A. Furthermore it is possible dynamically to extend the scope of a by "scope extrusion", eg. $\nu a.b.[a]P$ which declares a new name a and immediately passes it to the outside world along the channel b.

- Keys are names too, introduced using the ν-operator. Observe that in the π-calculus names are channel identifiers. This we can exploit in accounting for encryption in the following way:

- Encryption is second-order process passing. A message m encrypted using the private key K is an object that can deliver m to anyone happening to now K. That is, it is a process with one output port K along which m is passed to whoever possesses K and is willing to listen.

This suggests using a second-order version of the π-calculus as a semantical framework for modelling key management protocols, and indeed this is what we propose to do. Of course such models will be highly idealised: For instance the bit lengths used to represent nonces and keys are bounded, opening up for attacks on the encryption/decryption algorithms, and information can often be extracted from encrypted messages with very limited knowledge of the keys, or no knowledge at all, by analysing their bit representations. Nonetheless we believe that an idealised modelling of key management aspects alone can be useful, leaving analysis of actual encryption algorithms to be addressed by other means, even while recognising that a really water-tight boundary between the two is not a reasonable hope (cf. [3]).

Nested encryption and firewalling One complication needs to be attended to, though. As encryption can be nested we need to consider process terms of the shape

$$P = a.[K_1.[K_2.[m]0].0]P',$$

modelling a process passing $\{\{m\}_{K_2}\}_{K_1}$ along a and then proceeding to act as P'. A process receiving such a packet and decrypting to extract m would have the shape

$$Q = a.\lambda X_1.(X_1 \mid (K_1.\lambda X_2.X_2 \mid (K_2.\lambda m.Q'(m)))).$$

That is, it receives the process X_1, activates it and tries to receive from it along K_1 another process which it proceeds to activate, to try and receive from it m along K_2. As it stands, however, the π-calculus has no good way of preventing a third party from stealing m using K_2 once Q has decrypted using K_1. That is, once Q has reached the configuration $(K_2.[m]0) \mid (K_2.\lambda m.Q'(m))$, if an external intruder is present that may know about K_2 it will have the capability of receiving the m without necessarily having to know K_1 first. Thus decryption is unsafe, contrary to most reasonable modelling assumptions (cf. [7]).

A good remedy of this is to use a firewalling, or blocking operator $A\backslash a$ preventing communication along the channel a between A and its environment. This operator is already well known: It is just the CCS restriction operator, extended to the π-calculus in the obvious fashion by allowing state transitions $P\backslash a \xrightarrow{b} A$ just in case A has the shape $A'\backslash a$, $P \xrightarrow{b} A'$, and b, the communication channel, is distinct from a. This operator was also considered in the context of higher order processes by Thomsen [19][1]. Using the blocking operator we can protect m from theft along K_2 by putting $(K_2.[m]0) \mid (K_2.\lambda m.Q'(m))$ inside a K_2 firewall. Observe that we regard a as free in $A\backslash a$. Thus communication of a across the a firewall itself will be perfectly legitimate.

First or second-order π-Calculus? The blocking operator helps to explain why we insist on using second-order process passing even while we already know [17] that higher-order processes are reducible to first-order ones in the π-calculus. The point is that in the presence of the blocking operator the higher-order reduction of [17] is no longer applicable. We are currently accumulating strong evidence to suggest that the reduction from the second-order calculus with blocking to the first-order π-calculus with or without blocking while feasible in principle is very complicated and definitely not suitable as a modelling tool.

Specification in second-order temporal logic Our aim is to use a second-order temporal logic to specify desired correctness properties like secrecy and

[1]The blocking operator is sometimes known as "dynamic restriction", as a contrast to the static binding discipline of the π-calculus restriction. We find this terminology confusing, however, since in another sense the blocking operator is really the more static one, as name scopes are not subject to dynamic change.

authenticity. Specifically we suggest using a fragment of the modal μ-calculus extended with first-order features for names and name generation, and two arrows to account for process input and output. The logic follows quite closely ideas put forward in [4], using a function arrow $\phi \to \psi$ for input dependency, and a second-order arrow $(\phi \to \psi) \to \gamma$ for contextual output dependency. The idea is the following: A process waiting to input a parameter x to continue acting as P is written as a lambda-abstraction $\lambda x.P$. Dually, a process wanting to output to some receiver the process Q_1 to continue acting as Q_2 is written as the term $[Q_1]Q_2$, called a *process concretion*. Sometimes Q_1 and Q_2 may share a vector of private channel names $\vec{a}$ wishing to maintain these connections after Q_1 has been passed to its receiver. Such a process concretion is written $\nu\vec{a}.[Q_1]Q_2$. Matching receiver and sender results in the term $\lambda x.P \mid \nu\vec{a}.[Q_1]Q_2$ which is identified with the term $\nu\vec{a}.\{Q_1/x\}P \mid Q_2$, alpha-converting the bound names $\vec{a}$ as needed to avoid collision with names free in P. The input arrow expresses the expected functional dependency: For $\lambda x.P$ to have the property $\phi \to \psi$ it must be the case that Q_1 has the property ϕ only if $\{Q_1/x\}P$ has the property ψ. The output arrow expresses dependency upon receiving context: The process concretion $\nu\vec{a}.[Q_1]Q_2$ will have the property $(\phi \to \psi) \to \gamma$ just in case $\lambda x.P$ has the property $\phi \to \psi$ only if $\nu\vec{a}.\{Q_1/x\}P \mid Q_2$ has the property γ. In [4] we showed how this setting could be used to achieve an appropriate level of discriminatory power when measured against a strong version of bisimulation equivalence (cf. [11]), and we began investigating proof principles for these connectives.

Handling contravariant recursion Attempting to write down a first suggestion for a predicate expressing monotonically increasing trust in a set of channels faces a basic difficulty in that contravariant recursion seems indispensable. We have so far found no way around this difficulty. The problem is that the Knaster-Tarski fixed point theorem usually appealed to for least and greatest fixed point semantics require monotonicity which fails in the presence of contravariant recursion. We address this issue by giving alternative semantics in two different, but equivalent ways. First we suggest a semantics based on intervals in which solutions to recursive equations are computed using standard techniques as pairs of lower and upper approximations which will in general not meet. One will expect upper approximations to be the solutions of interest, and to support this we devise an alternative model using an iterative construction. This iterative model exploits a continuity property which is very useful in providing induction principles for later use. These properties only hold, however, for a fragment of the modal μ-calculus which lacks least fixed points and diamonds (existential next-state quantifiers). This, however, is a limited loss in view of the nature of the properties in which we have primary interest: Matters like secrecy and authenticity would be expected to have formulations as invariants and not to use existential computation path quantification.

Proving trust We then arrive at a suggestion for a process predicate expressing trust in a monotonically increasing set of channels, using contravariant recursion and greatest fixed points only, and the rest of the paper is devoted to proof principles for this trustedness predicate, and the proof of trust for a very simple example protocol. The most involved proof principles concern, as one should expect, parallel composition and name scoping. Several crucial lemmas need to be proved, of which we highlight two. First we need to show that if P is a process which respects trust of $\vec{a}$, and b does not occur freely in P, then P will respect the trust of $\vec{a} \cup \{b\}$ (we usually write $\vec{a}, b$ as a shorthand). The proof of this is, as one should expect, a simple inductive argument, using the induction principle hinted to earlier. However, we also need a corresponding result for functions, that if $\lambda x.P$ has the property $\vec{a}$ `trusted` $\rightarrow \vec{a}$ `trusted`, and if b does not occur freely in $\lambda x.P$, then $\lambda x.P$ will also have the property $\vec{a}, b$ `trusted` $\rightarrow \vec{a}, b$ `trusted`. This property has a far more intensional character as it has to do with definability of functions, and the proof is also much more delicate.

Deriving a type system Having proved the crucial lemmas for decomposing trust for parallel compositions and ν declarations we show how we quite simply can derive a type system for proving assertions of the shape $\vdash^{\vec{a}} A : \vec{b}$ `trusted`, and how, using this type system, we can prove correctness of the Andrew protocol example discussed above.

A word of caution One should take this correctness proof with a pinch of salt, though, for the following reason: The trustedness predicate we consider assumes trusted sets of names to be monotonically increasing. That is, we can accomodate protocols that add to the set of trusted names, but not protocols that revoke trust. But this means that all attacks that rely on some earlier piece of trusted information, like an old session key, having been compromised, can not be accomodated, as users of the protocol for our trustedness predicate to be applicable, can not be allowed to divulge such information. This is the reason for the "preliminary results" subtitle of the paper, and the reason why our approach should not yet be considered applicable to security and authentication protocols in general, but rather be considered as a first attempt to find useful specification and reasoning principles for concurrent programs that mix higher-order process passing with dynamic name generation and communication features as in the π-calculus.

Related work The present paper can be viewed as part of an ongoing trend towards operationally based accounts of security and authentication protocol. The closest predecessor of this work is Abadi and Gordon's work on the spi calculus [2]. Even though our work was developed independently, it is the credit of Abadi and Gordon first to have observed the usefulness of the π-calculus name scoping discipline for modelling security protocol features like nonce and key generation. In the spi calculus extra operators for encryption and decryption are added to the

π-calculus. Properties such as secrecy and authenticity are accounted for in equational terms, for instance by reflecting insensitivity of environments to changes in trusted values. By contrast we represent such properties more directly, as a logical formula. Moreover, due to the explicit treatment of encryption and decryption a rather non-standard version of testing equivalence [15] has to be appealed to for the correctness proofs in [2]. This complication does not arise in our approach since we reduce encryption and decryption to more general computational features.

Recently a number of authors have attempted to use state exploration methods to analyse security protocols (cf. [12, 10]). In such approaches the main difficulty is to faithfully represent protocols and intruders as finite state automata. Instead of leaving intruders undetermined, as in our approach, it becomes necessary to state explicitly at every possible step whether an action is or is not possible, including history dependencies. Secondly it becomes difficult to deal with unbounded information, such as protocols runs that can cause an in principle unbounded number of nonces, time stamps, or keys to be generated. For this reason (and for sheer model size considerations, one suspects) work has so far focused on public key encryption, and on single session establishment runs.

In another related strand of work a large number of authors have used static analysis and type systems to analyse security of information flow, cf. [1, 6, 9, 14, 20, 16]. The scope of these analyses is roughly the same as ours: They analyse whether security levels are respected during program execution, sometimes stratifying the analysis by eg. distinguishing readers and writers. As in our work revocation of trust is not supported. Our contribution to this line of work is to show how a type system for secure information flow can quite easily be derived from the very general and sound semantical basis that we provide, using the account of programs as second order π-calculus processes, and types/properties as interpreted second-order temporal formulas.

A Road Map We start by presenting the version of the second-order π-calculus used in the present paper and show, as an example, how the Andrew protocol is represented in this calculus. We then proceed to introduce the syntax and intended semantics of our second-order modal μ-calculus. To illustrate our need for contravariant recursion we suggest, in section 5, a predicate of trust in a finite set of names. This prompts an investigation of ways to give semantics to our logic in face of this difficulty. This issue is addressed in sections 6, 7, and 8 where we present two alternative semantics, and show them equivalent. One of these semantics, the iterative one, is particularly useful in that it identifies an induction principle which is used throughout the remainder of the paper. Having established a reasonable semantics we can then proceed to complete the specifications of secrecy and authenticity, and to prove that these properties actually hold of our example. Only proof of secrecy is considered in this paper. As the correctness properties are stated as properties of open systems (senders, receivers, and unknown intruders) a useful proof strategy is to provide a sequence of lemmas

showing the ways in which the trust predicate composes over the process combinators. The difficult cases are restriction and parallel composition. Proofs are given in section 10 and in the appendix. Up to some minor details what the structural lemmas established in section 10 amounts to, is a type system for trust inference. This type system is exhibited and shown to be sound in section 11. Then, in section 12, we show as an example how the type system is used to establish secrecy of the Andrew protocol. Finally, in section 13, we summarise our results and discuss some outstanding matters including alternatives for representing cryptographic primitives, how to measure the correctness of our representations, and the need for the blocking operator.

2 Processes

In this section we give an informal presentation of the language used to model protocols., and as much as its operational semantics as is needed to understand the specification logic and the reasoning of the correctness proofs. A formalised semantics is given in appendix 1. Roughly, the process language is a merge of the π-calculus [13] with the second-order process-passing calculus presented in [4]. It uses the following primitive objects:

- Channel names a, b, along with the special label τ, used for invisible, or silent transitions.
- Agent variables x, y.
- Agent constants D. With each constant is associated a unique defining equation $D = A$ where A is an agent according to the definition below.

Agents come in three flavours: Processes which perform transitions; abstractions, responsible for name and agent input; and concretions, responsible for name and agent output. Process terms are ranged over by P, Q, abstractions by F, concretions by C, and agents in general by A and B. To each well-formed agent term is assigned an arity $+w$ or $-w$, $w \in \{\texttt{chan}, \texttt{agent}\}^*$ indicating, e.g. for an abstraction, the number and position of channel and agent arguments it requires to become a process term. The null arity is (), and by convention, $+() = () = -()$.

Processes Processes are agent terms of null arity: They neither require nor provide parameters to be able to perform (or refuse) transitions. Agent variables are (open) process terms; 0 is the terminated process; $P + Q$ is the process that can choose between transitions of P and of Q; $a.A$ is the prefix process that can perform an a-transition and evolve into A; $P \mid Q$ is the parallel composition of P and Q; $\nu a.P$ declares a new name a, local to P (but exportable to the outside world through subsequent communications); $\texttt{if } a = b \texttt{ then } P \texttt{ else } Q$ is the conditional, often generalised to arbitrary boolean combinations of name equalities

and inequalities; $P\backslash a$ is blocking; and if $D = F$ and F has shape $\lambda b_1.\cdots\lambda b_n.P$ then $D(a_1,\ldots,a_n)$ is a process term too.

Abstractions We operate with two abstraction constructors, one for free input and one for bound output, similar to the situation in [5]. The free input abstraction has the shape $\lambda a.A$ ($\lambda x.A$) and has arity $-\mathtt{chan}w$ ($-\mathtt{agent}w$) if A is an abstraction term of arity $-w$ or, if w is empty, A is a process term. The arity of a bound input abstraction, $\nu\lambda a.A$, is calculated similarly.

Concretions Concretions have one of the form $[a]A$, $\nu a.[a]A$, or $\nu\vec{a}[P]A$. The first instance corresponds to the output of the free channel name a. The second to the output of a local name a, and the third to the output of a process term P with local names $\vec{a}$. If A is a concretion term of arity $+w$ then $[a]A$ and $\nu a.[a]A$ both have arity $+\mathtt{chan}w$ and $\nu\vec{a}[P]A$ has arity $+\mathtt{agent}w$. If w is empty A is again a process term.

The transition semantics A standard π-calculus style semantics can easily be given to the above language. We assume a transition relation $P \xrightarrow{\tau} Q$, and a family of transition relations $P \xrightarrow{a} A$. A few examples suffices to highlight the important points:

- Invisible transitions arise because of communication. For communication to take place arities of the resulting abstraction/concretion pair must match. Thus, e.g. if $P_1 \xrightarrow{a} \nu\lambda b_1.F$ and $P_2 \xrightarrow{a} \nu b_2.[b_2]C$, F has arity $-w$, and C has arity $+w$, communication can take place. Then, if $F \mid C = Q'$, the invisible transition $P_1 \mid P_2 \xrightarrow{\tau} Q = \nu b_2.\{b_2/b_1\}Q'$ is enabled, where we assume variables to have been alpha-converted such that confusion does not arise. Similarly, if $P_1 \xrightarrow{a} \lambda x.F$ and $P_2 \xrightarrow{a} \nu\vec{b_2}.[P]C$ we obtain $P_1 \mid P_2 \xrightarrow{\tau} Q = \nu\vec{b_2}.\{P/x\}Q'$ where $Q' = F \mid C$.
- Similarly, for $P_1 \mid P_2$, it is possible that no communication takes place. Thus, eg. if $P_1 \xrightarrow{a} \lambda b.F$ and $F \mid P_2 = Q'$ then $P_1 \mid P_2 \xrightarrow{a} Q = \lambda b.Q'$. Observe again that α-conversion is used to avoid capture of variables.
- The remaining connectives reflect the intuitions given above. Thus, for instance, $\nu a.A$ declares a local name a in A and does not permit a-transitions to take place. That is, $\nu a.P \xrightarrow{b} A$ if and only if $a \neq b$ and $P \xrightarrow{b} A'$ and $A = \nu a.A'$.

3 An Example: The Andrew Secure RPC Protocol

Consider the Andrew secure RPC protocol given in the introduction. We show how to represent this protocol in the second-order process calculus, using the ideas

```
Alice = λK_ab.in?d.xfer!{data,d}_{K_ab}.Alice K_ab + AliceSwitch K_ab

AliceSwitch = λK_ab.
    νN_a.xfer!{switch,N_a}_{K_ab}.
        xfer?x.x |
        (AliceSwitch K_ab + (K_ab?(t,N'_a,K'_ab).
        if t = next and N_a = N'_a
        then xfer!{ack,N_a}_{K'_ab}.(Alice K'_ab + AliceSwitch K_ab)
        else AliceSwitch K_ab))
```

Figure 1: Agent `Alice`

```
Bob = λK_ab.xfer?x.x | ((Bob K_ab) + (K_ab?(t,x).
      if t = data then out!x.(Bob K_ab) else
      if t = switch
      then
         νK'_ab.xfer!{next,x,K'_ab}_{k_ab}.
         xfer?x.x |((Bob K_ab) + (K'_ab?(t,N'_a).
         if t = ack and N'_a = x then (Bob K'_ab) else (Bob K_ab)))
      else (Bob K_ab)))
```

Figure 2: Agent `Bob`

outlined in the introduction. Thus nonces and encryption keys are represented as names, and encryption and decryption as second-order communication. As a first approximation of such a representation consider the agents `Alice` and `Bob` presented in figures 1 and 2. We use some abbreviations. First

$$\begin{aligned} c!(T_1,\ldots,T_n).A &\triangleq c.[T_1]\cdots[T_n]A \\ \{T_1,\ldots,T_n\}_K &\triangleq K!(T_1,\ldots,T_n).0 \end{aligned}$$

where $T_1,\ldots,T_n$ ranges over names and processes. Secondly we let $c?(v_1,\ldots,v_n).A$ abbreviate the sum of all terms of the shape $c.(\nu)\lambda v_1.\cdots.(\nu)\lambda v_n.A$ where the ν is optional, and requires the lambda to which it is applied to be a free name abstraction. Observe that this involves a non-deterministic commitment to a particular choice of input parameter types and thus may introduce deadlocks. This can be remedied, but as we are only interested in properties to hold for all possible computations the matter has little importance.

Compared to the "standard" account little has been changed except that the protocols have been augmented with message tags to handle control flow, and a data transfer phase, in which input data is received along a channel `in`, encrypted and passed from `Alice` to `Bob`, and then output along `out`. As our aim is to

specify and analyse properties in terms of external input-output behaviour some such modification is necessary, and in most parts it is completely uncontroversial. On three counts, however, some discussion is needed.

Free and bound input Our process language possesses the capability of detecting whether a given argument occurred freely or bound at the sender. On the face of it this is clearly an unreasonable assumption: What is received are bit strings and even if some tag of some sort states the nature of the argument how is this tag to be trusted? On the other hand we need this distinction in order to know, when a channel parameter is received along a trusted channel, whether to extend trust to this new channel or not. Our policy is simple: new channels communicated along trusted channels are themselves to be trusted. The argument of unreasonable expressiveness is countered by the examples always allowing for both free and bound input, as is the case above.

Looseness of specification The data transfer phases of `Alice` and `Bob` consist simply of inputting a piece of data, encrypting and then transferring it over the medium, respectively receiving the encrypted package, decrypting and then outputting. In this respect the model is overspecific: it states explicitly, for instance, that old session keys are *not* corrupted. But this is too strong an assumption as many attacks use replays with old and corrupted session keys. Rather one would want to replace Alice by an *open* specification of the shape

$$\texttt{Alice} = \lambda K_{ab}.(F\ K_{ab});\texttt{AliceSwitch}\ K_{ab}$$

where F is a free abstraction variable subject to assumptions such as

- F never reveals its first argument to the outside world,
- F never reveals secrets received along `in`, except when encrypted by K_{ab}.

Distinguishing name and process parameters One more remark needs to be made concerning the representation. In the systems we are modelling, data is shifted in terms of bit strings or voltages on a wire. Thus there is no way to tell whether a piece of datum is really a nonce or a piece of encrypted information. This does not hold in the representation, as name and process passing are treated distinctly. Observe, however, that the safety of our conclusions are not affected by this. Rather intruders are given slightly more discriminating power in the representation than in the modelled system. Our conjecture is that in practice this issue is negligible.

4 Process Properties

Our chief interest is in the properties of secrecy and authenticity. Our intention is to formulate these as functional and temporal properties expressing constraints

on the input-output behaviour of the system under consideration. In our example the system consists of the agents `Alice` and `Bob` running in an unknown (and potentially hostile) environment Z. Z should be assumed to have access only to channels and data open to outside intruders. Evidently this includes the channel `xfer` (but also the tags `data`, `switch` etc.). However, the initial value of K_{ab} should be regarded as trusted, as should the channels `in` and `out`. Supposing now that ϕ expresses an correctness property such as secrecy. The overall proof goal can then be formulated as a sequent of the shape

$$Z : \quad \psi \vdash (\nu K_{ab}.\texttt{Alice}\ K_{ab} \mid \texttt{Bob}\ K_{ab}) \mid Z : \quad \phi$$

where ψ are the assumptions made on Z (roughly: that Z does not know `in` and `out`).

Since the intruder Z is already considered "part of" the global system which is considered, the correctness property ϕ does not need to speak about process passing: If eg. secrecy is violated there will be a way for Z to reveal a secret along a name which is not `out`, resorting to encryption or other second-order communication only internally. More general properties which *do* talk about second-order communication will be needed once we arrive at the proofs, however.

Thus a suitable functional + temporal logic for our purpose will need to talk about names and their identities, properties of names and processes which are output, dependencies on names and properties of processes being received, in addition to usual safety properties. Observe, however, that to express the correctness properties we have in mind there is no use for liveness properties or existential path quantification. This fact will be quite useful once we come to consider the semantics. The logic will have the following primitives:

- $a = b$, $\neq b$, $\phi \wedge \psi$, $\phi \vee \psi$, $\forall a.\phi$, $\exists a.\phi$. This is just first-order logic with equality. We also need basic operations on finite sets $\vec{a}$: set membership and quantification over finite sets.

- $\vec{a}$ `fresh`, `new` $\vec{a}.\phi$. The first primitive expresses that no element of the set $\vec{a}$ occurs freely in the agent being predicated. The second primitive expresses of an agent A that it is identical to an agent of the shape $\nu\vec{b}.A'$ such that A' has the property $\{\vec{b}/\vec{a}\}\phi$. For now we can use the term "identical" as meaning, roughly, "bisimulation equivalent" (cf. [11]). We return to this issue shortly.

- $[a]\phi$, $[\tau]\phi$. These are the universal next-state quantifiers well-known from modal logic. So $[a]\phi$ will hold of an agent just in case it is a process, and whatever agent results from the performance of an a-transition must satisfy ϕ.

- $a \rightarrow \phi$, $a \rightarrow_\nu \phi$, $a \leftarrow \phi$. These primitives express name input-output properties. The first expresses of an agent A that it is an abstraction $\lambda a'.A'$, and

that $\{a/a'\}A'$ has the property ϕ. The second expresses that A is an abstraction $\nu\lambda a'.A'$, and that $\{a''/a'\}A'$ has the property $\{a''/a\}\phi$ whenever a'' does not occur freely in neither $\nu\lambda a'.A$ nor ϕ (minus a). The third expresses the property that A is a concretion of the shape $[a']A'$, that $a = a'$, and that A' has the property ϕ. A fourth connective $a \leftarrow_\nu \phi$ will be derivable, as `new` $a.a \leftarrow \phi$.

- $\phi \to \psi$, $(\phi \to \psi) \to \gamma$. These primitives are used for second-order communication. The function arrow $\phi \to \psi$ expresses of A that it is identical to a second-order abstraction $\lambda x.A'$, and that if P is a process satisfying ϕ, then $\{P/x\}A'$ will have property ψ. The second primitive is a contextual property. It holds of a second-order (process) concretion A of the shape $\nu\vec{a}.[P]A'$ provided that for any receiving context f with the property $\phi \to \psi$, the process $\nu\vec{a}.(fP) \mid A'$ will have the property γ. This idea of using a second-order implication to capture contextual properties of process output originates with the paper [4].

In addition to these primitives our intention is to allow properties to be defined by greatest fixed points in the style familiar from the modal μ-calculus (cf. [18]). This is quite straightforward if we can define the required properties using covariant recursion only. Unfortunately as yet we only have solutions that make use of contravariant recursion, and thus we need to address the foundational problem of making sense of this. This we do in the subsequent sections. First, however, some syntactical matters: Recursively defined properties take the shape $(\nu X(a_1, \ldots, a_n).\phi)(b_1, \ldots, b_n)$ (cf. [5]). Alternatively we use the sugared form $X(b_1, \ldots, b_n)$ in the context of a definition of the shape $X(a_1, \ldots, a_n) => \phi$. We require that recursive definitions are *guarded* in the sense that all occurrences of X in ϕ must be within the scope of either a modal operator, or one of the arrows. A formula ϕ is *propositionally closed* if ϕ does not have free occurrences of (parametrised) variables X.

5 Expressing Trust

Let us try to express, using the connectives introduced above along with greatest fixed points, the property $\vec{a}$ `trusted` that, intuitively, trusted information can appear along trusted channels only. With some experimentation one arrives at a definition like the following:

$\vec{a}$ `trusted =>`
  $[\tau]$ $\vec{a}$ `trusted` $\wedge$ $\forall b$.
  $[b]$`(/* Continuation:  process */` $\vec{a}$ `trusted` $\vee$
    `/* Output */` $\vec{a}$ `trusted_out_after` b $\vee$
    `/* Input */` $\vec{a}$ `trusted_in_after` b`)`

$\vec{a}$ `trusted_out_after` b `=>`

$$
\begin{array}{l}
\quad \texttt{/* process */}\ \vec{a}\ \texttt{trusted} \vee \\
\quad \texttt{/* free name output */} \\
\quad (\exists c.c \leftarrow (\ \vec{a}\ \texttt{trusted\_out\_after}\ b \wedge (c \in \vec{a} \supset b \in \vec{a}))) \vee \\
\quad \texttt{/* bound name output */} \\
\quad (\texttt{new}\ c.c \leftarrow ((b \in \vec{a} \supset \vec{a}, c\ \texttt{trusted\_out\_after}\ b) \wedge \\
\qquad\qquad (b \notin \vec{a} \supset \vec{a}\ \texttt{trusted\_out\_after}\ b))) \vee \\
\quad \texttt{/* process output */} \\
\quad (\forall \vec{d}.\vec{d}\ \texttt{fresh} \supset \texttt{new}\ \vec{c}. \\
\quad\ (\vec{a}, \vec{d}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}, \vec{c}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{d}, \vec{c}\ \texttt{trusted\_out\_after}\ b)
\end{array}
$$

$$
\begin{array}{l}
\vec{a}\ \texttt{trusted\_in\_after}\ b\ \texttt{=>} \\
\quad \texttt{/* process */}\ \vec{a}\ \texttt{trusted} \vee \\
\quad \texttt{/* free name input */} \\
\quad (\forall c.c \rightarrow \vec{a}\ \texttt{trusted\_in\_after}\ b) \vee \\
\quad \texttt{/* bound name input */} \\
\quad (c \rightarrow_\nu (b \in \vec{a} \supset \vec{a}, c\ \texttt{trusted\_in\_after}\ b) \wedge \\
\qquad (b \notin \vec{a} \supset \vec{a}\ \texttt{trusted\_in\_after}\ b)) \vee \\
\quad \texttt{/* process input */} \\
\quad (\forall \vec{c}.\vec{c}\ \texttt{fresh} \supset \vec{a}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{c}\ \texttt{trusted\_in\_after}\ b))
\end{array}
$$

The idea is quite simple: To show that the process being predicated respects the trustedness of names in $\vec{a}$ we need to consider the various transitions that may be possible from the initial state and the various types of continuation agents that may ensue. If the continuation is a free output of a name c then $\vec{a}$ must continue to be trusted, and if c is trusted then b had better be trusted too. If the continuation is a bound output of a name c then if b was trusted we can regard both $\vec{a}$ and c as trusted, and otherwise we must continue to respect the trustedness of $\vec{a}$. The most interesting cases are those for second-order input and output. Consider for instance second order input. The process being input must respect the trustedness of $\vec{a}$, evidently. But in addition we must permit that process to mention other trusted information of which we are not yet aware. That information will be "fresh" to us, and we had better ensure that after input of the process we respect the trustnedness of both $\vec{a}$ and $\vec{c}$ (as it were). Similarly for process output. The receiving context may contain information which should be trusted, but of which we have not yet been informed. Secondly we, as outputting agent, may through scope extrusion generate new information to be trusted. All three types of information needs to be trusted by the entire system.

The trustedness predicate above is given for polyadic communication. For monadic communication a simpler definition can be given for which the disjunct $\vec{a}$ `trusted` is removed from the input and output predicates, and for which the recursive calls of the input and output predicates are replaced by calls of the main trustedness predicate. In the proofs we use the monadic version only. However, the arguments extend to the polyadic case quite easily.

Observe the two contravariant occurrences of the trustedness predicate, for the cases of second order input and output. We see no possibility at present of

avoiding these. Freeness checks, for instance, are clearly much too inexpressive. On the other hand the semantics of the modal μ-calculus on which the logic is built rests on the fact that fixed points are required to be computed of monotone operations only, and in the presence of contravariant recursion monotonicity will fail. In the next section we go on to show a way of bypassing this problem.

Trust and monotonicity One should not view the above trustedness predicate as a serious candidate for proving trust in distributed systems: For this it has too many shortcomings. Essentially the trustedness predicate embodies a particular and very rigid protocol for handling mutual trust:

1. Whenever a new piece of information passes a piece of trusted information, trust is extended.

2. Trust increases monotonically, ie. there is no way of revoking trust.

In terms of modelling security protocol the second point is damaging, as very many attacks rely on replays using eg. old and possibly compromised session keys.

6 On Contravariant Recursion

A standard semantical account of our specification logic would appeal to a semantical mapping $\|\phi\|_s(\rho)$ where ρ is an environment giving for each formula identifier X a set of agents, possibly parametrised on a sequence of names or finite sets of names, and $\|\phi\|_s(\rho)$ is a set of agents. For fixed points, function arrows, and "boxes" we would expect a clauses like the following (for simplicity we consider only unparametrised recursive definitions):

$$
\begin{array}{ll}
(1) & \|X\|_s(\rho) = \rho(X) \\
(2) & \|\nu X.\phi\|_s(\rho) = \cup\{S \mid S \subseteq \|\phi\|_s(\{S/X\}\rho)\} \\
(3) & \|[a]\phi\|_s(\rho) = \{P \mid \forall A.P \xrightarrow{a} A \supset A \in \|\phi\|_s(\rho)\} \\
(4) & \|\phi \to \psi\|_s(\rho) = \{A \mid A = \lambda x.A', \forall P \in \|\phi\|_s(\rho), \{P/x\}A' \in \|\psi\|_s(\rho)\} \\
 & \qquad\qquad = \|\phi\|_s(\rho) \to \|\psi\|_s(\rho)
\end{array}
$$

We might hope that, despite the contravariant recursion, $\nu X.\phi$ would nonetheless be the greatest fixed point of $\nu(X)$. Unfortunately this is not the case. Consider for instance the formula

$$(5)\ \phi = \nu X.[a](([b](\exists c.c = d \wedge c \leftarrow \texttt{true}) \wedge X) \to X).$$

Consider the set

$$S_1 = \{P^n : n \in \omega, P = a.\lambda X.(P \mid X)\}$$

where $P^n = P \mid P \mid \cdots (n\ times) \cdots \mid P$ is the n'ary parallel composition of P. Observe that $S_1 \subseteq \|\phi\|_s(\{S_1/X\}\rho)$. For let $Q \in S_1$, ie. $Q = P^n$ for some $n \in \omega$. Whenever $Q \xrightarrow{a} Q'$ then $Q' = (\lambda X.P \mid X) \mid P^{n-1}$, considering $\mid$ as a commutative

monoid operation. If $R \in \|[b](\exists c.c = d \wedge c \leftarrow \mathtt{true})\|_s(\{S_1/X\})$ then $R = P^m$ for some $m \in \omega$. So $Q'R$ will have shape P^k for some $k \in \omega$. Thus S_1 is a prefixed point of $[a](([b](\exists c.c = d \wedge c \leftarrow \mathtt{true}) \wedge X) \rightarrow X)$. The set $S_2 = \{b.[d']0\}$ $(d' \neq d)$ is another such prefixed point, as no member of S_2 has an a-transition enabled. But no set containing both P and $b.[d']0$ can be a prefixed point. So greatest fixed points do not in general exist.

So we either abandon the enterprise here or else we try to make sense of contravariant recursion by other means. What we are really after is an *iterative* understanding of recursive definitions where iteration would be in number of transitions. Indeed this is what we do. Directly formalising this would be less than transparent, however, as we would have no understanding of iteration as a limiting construction. Instead we resort to an interval based semantics (cf. [4]).

7 Intervals

An *interval* is a pair (S_1, S_2) for which $S_1 \subseteq S_2$. One should regard the interval (S_1, S_2) as determining a pair of approximations, S_1 a lower approximation giving agents that must be included, and S_2 an upper approximation giving agents that may be included, or, better, have not yet been ruled out. Our intention is to compute greatest fixed points as the limit of upper approximations. Intervals are ordered pointwise by $(S_1, S_2) \sqsubseteq (S_1', S_2')$ iff $S_1 \subseteq S_1'$ and $S_2' \subseteq S_2$. There is a natural function space construction on intervals,

$$(S_1, S_2) \rightarrow (S_1', S_2') = S_2 \rightarrow S_1', S_1 \rightarrow S_2')$$

which is covariant in both arguments:

Proposition 1 *Let I, I', J be intervals. If $I \sqsubseteq I'$ then $I \rightarrow J \sqsubseteq I' \rightarrow J$ and $J \rightarrow I \sqsubseteq J \rightarrow I'$.* □

For the interval-based semantics we introduce some constants and operations on sets of agents:

$$\begin{array}{rcl}
\vec{a}\ \mathit{fresh} & = & (\{A \mid \vec{a} \cap fn(A) = \emptyset\}, \{A \mid \vec{a} \cap fn(A) = \emptyset\}) \\
\mathit{new}\ \vec{a}.(S_1, S_2) & = & (\{A \mid \nu\vec{a}.A' \preceq A, A' \in S_1\}, \{A \mid \nu\vec{a}.A' \preceq A, A' \in S_2\}) \\
(S_1, S_2) \wedge (S_1', S_2') & = & (S_1 \cap S_1', S_2 \cap S_2') \\
(S_1, S_2) \vee (S_1', S_2') & = & (S_1 \cup S_1', S_2 \cup S_2') \\
[\alpha](S_1, S_2) & = & (\{P \mid P \stackrel{\alpha}{\rightarrow} A \supset A \in S_1\}, \{P \mid P \stackrel{\alpha}{\rightarrow} A \supset A \in S_2\}) \\
a \rightarrow (S_1, S_2) & = & (\{\lambda b.A \mid \{a/b\}A \in S_1\}, \{\lambda b.A \mid \{a/b\}A \in S_2\}) \\
a \leftarrow (S_1, S_2) & = & (\{[a]A \mid A \in S_1\}, \{[a]A \mid A \in S_2\}) \\
(S_1, S_2) \Rightarrow (S_1', S_2') & = & (\{\nu\vec{a}.[P]A \mid \forall(\lambda x.A') \in S_2.\nu\vec{a}.\{P/x\}A' \mid A \in S_1'\}, \\
 & & \{\nu\vec{a}.[P]A \mid \forall(\lambda x.A') \in S_1.\nu\vec{a}.\{P/x\}A' \mid A \in S_2'\})
\end{array}$$

Observe that the definition is parametric on the relation $\preceq$. We return to the definition of this relation later.

Now, let ρ be a *formula environment*, a mapping of formula identifiers to agent intervals. We define an interval-based semantics, $\|\phi\|(\rho)$. For simplicity we continue to consider only unparametrised recursive definitions, and leave out the semantics for name equality, inequality, finite set equality, membership, inequality, and quantification. These features are easily dealt with, and adds only complexity to the exposition.

$$
\begin{array}{rcl}
\|X\|(\rho) & = & \rho(X) \\
\|\nu X.\phi\|(\rho) & = & \sqcap\{(S_1,S_2) \mid (S_1,S_2) \text{ interval}, \\
 & & \|\phi\|(\{(S_1,S_2)/X\}\rho) \sqsubseteq (S_1,S_2)\} \\
\|\phi \wedge \psi\|(\rho) & = & \|\phi\|(\rho) \wedge \|\psi\|(\rho) \\
\|\phi \vee \psi\|(\rho) & = & \|\phi\|(\rho) \vee \|\psi\|(\rho) \\
\|\vec{a}\ \texttt{fresh}\|(\rho) & = & \vec{a}\ \mathit{fresh} \\
\|\texttt{new}\ \vec{a}.\phi\|(\rho) & = & \mathit{new}\ \vec{a}.S \\
\|[\alpha]\phi\|(\rho) & = & [\alpha]\|\phi\|(\rho) \\
\|a \rightarrow \phi\|(\rho) & = & a \rightarrow \|\phi\|(\rho) \\
\|a \rightarrow_\nu \phi\|(\rho) & = & (\{\nu\lambda b.A \mid \{a'/b\}A \in \pi_1(\|\{a'/a\}\phi\|(\rho))\}, \\
 & & \{\nu\lambda b.A \mid \{a'/b\}A \in \pi_2(\|\{a'/a\}\phi\|(\rho))\}) \\
 & & (\text{where } a' \notin \mathit{fn}(\lambda b.A) \cup \mathit{fn}(\phi) - \{a\}) \\
\|a \leftarrow \phi\|(\rho) & = & a \leftarrow \|\phi\|(\rho) \\
\|\phi \rightarrow \psi\|(\rho) & = & \|\phi\|(\rho) \rightarrow \|\psi\|(\rho) \\
\|(\phi \rightarrow \psi) \rightarrow \gamma\|(\rho) & = & (\|\phi\|(\rho) \rightarrow \|\psi\|(\rho)) \Rightarrow \|\gamma\|(\rho)
\end{array}
$$

For propositionally closed ϕ we abbreviate $\|\phi\|(\rho)$ by $\|\phi\|$. We first prove that the semantics is well-defined:

Lemma 2 *For all formulas ϕ and formula environments ρ, $\|\phi\|(\rho)$ is an interval.*

Proof Structural induction. □

The crucial point of the interval-based semantics is that definable operators become monotone:

Lemma 3 *For all formulas ϕ and formula environments ρ, the interval operator*

$$\lambda(S_1,S_2).\|\phi\|(\{(S_1,S_2)/X\}\rho)$$

is monotone.

Proof Use prop. 1. □

By means of lemma 3, using the familiar Knaster-Tarski Fixed Point Theorem we can conclude that $\|\nu X.\phi\|(\rho)$ is indeed the least fixed point (under $\sqsubseteq$,

mind) of the interval operator $\lambda(S_1, S_2).\|\phi\|(\{(S_1, S_2)/X\}\rho)$. Moreover, if f is any monotone interval operator, define

$$\begin{aligned} f^0 &= (Dead, \{A \mid A \text{ agent}\}) \\ f^{n+1} &= f(f^n) \\ f^\lambda &= \sqcup_{\kappa<\lambda} f^\kappa. \end{aligned}$$

where

$$Dead = [\tau]false \wedge \forall a.[a]false.$$

We start iterating from $Dead$ to make induction work out slightly simpler. This is a technical convenience only.

Corollary 4 $\|\nu X.\phi\|(\rho) = \sqcup_\kappa(\lambda(S_1, S_2).\|\phi\|(\{(S_1, S_2)/X\}\rho))^\kappa$

Proof By the Knaster-Tarski Fixed Point Theorem. □

8 An Iterative Semantics

Intuitively we would want to think of an agent A satisfying the property ϕ, if (for ρ given) $\|\phi\|(\rho) = (S_1, S_2)$ and $A \in S_2$ (as S_2 is in some intuitive meaning the *largest* set consistent with the satisfaction clauses). In this section we give an alternative, iterative, semantics which is in a sense the semantics we are looking for, as it explains fixed points as an iteration limit. Our intention is to compute the semantics of a formula ϕ as the limit of an increasing chain of sets of agents $\|\phi\|^n(\sigma)$, $n \in \omega$. At each iteration step, $\|\phi\|^n(\sigma)$ will be a set of agents which is permitted to depend on the behavior of agents only down to a global transition depth n. To get at this notion we introduce a version of the simulation preorder.

Definition 5 (Simulation preorder)

1. Define the preorders $\preceq_n$ inductively by the following clauses (where we use f to range over functions from names to abstractions or processes to abstraction, as appropriate given the context):
   (a) $P \preceq_0 Q$ holds always.
   (b) $P \preceq_{n+1} Q$ iff $fn(P) = fn(Q)$ and $Q \xrightarrow{\alpha} B$ implies $P \xrightarrow{\alpha} A$ such that $A \preceq_n B$.
   (c) $\lambda x.A_1 \preceq_{n+1} \lambda y.A_2$ iff for all a (P), $\{a/x\}A_1 \preceq_n \{a/y\}A_2$ $(\{P/x\}A_1 \preceq_n \{P/y\}A_2)$.
   (d) $[a]A_1 \preceq_{n+1} [b]A_2$ iff $a = b$ and $A_1 \preceq_n A_2$, $\nu a.[a]A_1 \preceq_{n+1} \nu b.[b]A_2$ iff for all fresh c, $\{c/a\}A_1 \preceq_n \{c/b\}A_2$, $\nu\vec{a}.[P]A \preceq_{n+1} \nu\vec{b}.[Q]B$ iff for all process abstractions $\lambda x.A'$ for which $\vec{a}$ and $\vec{b}$ are fresh, $\nu\vec{a}.\{P/x\}A' \mid A \preceq_n \nu\vec{b}.\{Q/x\}A' \mid B$.

2. $A \preceq B$ iff for all $n \in \omega$, $A \preceq_n B$. $A \approx B$ iff $A \preceq B$ and $B \preceq A$.

3. Let S be a set of agents. Then $\uparrow_n S = \{B \mid \exists A \in S. A \preceq_n B\}$.

Observe that $\approx$, being the intersection of a simulation order and its converse, is strictly coarser than bisimulation equivalence [11]. Def. 5 determines the preorder used in the interval semantics of the *new* operator. In fact we could have used any preorder there which is at least as strong as $\preceq$ (such as bisimulation equivalence).

We can now introduce the iterative semantics $\|\phi\|(\sigma)$, where σ is an environment assigning sets of agents to formula identifiers:

$$
\begin{array}{rcl}
\|\phi\|^0 & = & \{A \mid A \text{ an agent}\} \\
\|X\|^{n+1}(\sigma) & = & \uparrow_{n+1} \sigma(X) \\
\|\nu X.\phi\|^{n+1}(\sigma) & = & \|\phi\|^{n+1}(\{\|\nu X.\phi\|^n(\sigma)/X\}\sigma) \\
\|\phi \wedge \psi\|^{n+1}(\sigma) & = & \|\phi\|^{n+1}(\sigma) \cap \|\psi\|^{n+1}(\sigma) \\
\|\phi \vee \psi\|^{n+1}(\sigma) & = & \|\phi\|^{n+1}(\sigma) \cup \|\psi\|^{n+1}(\sigma) \\
\|\vec{a}\ \texttt{fresh}\|^{n+1}(\sigma) & = & \{A \mid \vec{a} \cap fn(A) = \emptyset\} \\
\|\texttt{new}\ \phi\|^{n+1}(\sigma) & = & \{A \mid \nu\vec{a}.A' \preceq_{n+1} A, A' \in \|\phi\|^{n+1}(\sigma)\} \\
\|[\alpha]\phi\|^{n+1}(\sigma) & = & \{P \mid P \xrightarrow{\alpha} A \supset A \in \|\phi\|^n(\sigma)\} \\
\|a \to \phi\|^{n+1}(\sigma) & = & \{\lambda b.A \mid \{a/b\}A \in \|\phi\|^n(\sigma)\} \\
\|a \to_\nu \phi\|^{n+1}(\sigma) & = & \{\lambda b.A \mid \{a'/b\}A \in \|\{a'/a\}\phi\|^n(\sigma), \\
 & & a \notin fn(\lambda b.A) \cup (fn(\phi) - \{a\})\} \\
\|a \leftarrow \phi\|^{n+1}(\sigma) & = & \{[a]A \mid A \in \|\phi\|^n(\sigma)\} \\
\|\phi \to \psi\|^{n+1}(\sigma) & = & \|\phi\|^n(\sigma) \to \|\psi\|^n(\sigma) \\
\|(\phi \to \psi) \to \gamma\|^{n+1}(\sigma) & = & \{\nu\vec{a}.[P]A \mid \forall(\lambda x.A') \in \|\phi\|^n(\sigma) \to \|\psi\|^n(\sigma). \\
 & & \nu\vec{a}.\{P/x\}A' \mid A \in \|\gamma\|^n(\sigma)\} \\
 & = & (\|\phi\|^n(\sigma) \to \|\psi\|^n(\sigma)) \Rightarrow \|\gamma\|^n(\sigma)
\end{array}
$$

Observe that guardedness is important for this definition to make sense. As above abbreviate $\|\phi\|^n(\sigma)$ by $\|\phi\|^n$ when ϕ is propositionally closed. We want to show the following theorem relating the interval-based and the iterative semantics:

Theorem 6 *Let ϕ be a propositionally closed formula, and let $\|\phi\| = (S_1, S_2)$. Then $A \in S_2$ iff for all $n \in \omega$, $A \in \|\phi\|^n$.*

We prove this theorem in the appendix. Here we just give some intuition for why we might expect the theorem to hold.

The interval semantics constructs the semantics of greatest fixed points by simultaneously approaching the limit from below and from above. The lower approximation is used for contravariant argument places, and the upper one for covariant ones. The two limits, the lower and the upper, will not in general meet. However, by disallowing diamonds and least fixed points we obtain a continuity property

of our logic, in the sense that finite approximations are sufficient to determine whether a property holds of a given process. Here "finite approximations" are approximants in a temporal rather than functional sense. The important ingredient is a *truncation operator* $\texttt{trunc}(n, A)$ which ensures termination at a given transition depth. Operationally the truncation operator is very simple: If $P \xrightarrow{\alpha} A$ then $\texttt{trunc}(n+1, P) \xrightarrow{\alpha} \texttt{trunc}(n, P)$, and no transition from $\texttt{trunc}(0, P)$ is possible. Secondly the truncation operator commutes with λ, ν, and concretion formation while decreasing the truncation index, ie. $\texttt{trunc}(n+1, \lambda a.A) = \lambda a.\texttt{trunc}(n, A)$ etc.

Notice that while our proof of theorem 6 may depend on the truncation operator, the conclusion (the theorem itself) does not.

Using the truncation operator the proof of theorem 6 can be outlined as follows: Assume $A \in S_2$ where $(S_1, S_2) = \|\phi\|$. We conclude:

$$\begin{array}{lll} A \in S_2 & \text{iff} \quad \texttt{trunc}(n, A) \in S_2 & \text{(lemma 20)} \\ & \text{iff} \quad \texttt{trunc}(n, A) \in S_1 & \text{(lemma 21)} \\ & \text{iff} \quad \texttt{trunc}(n, A) \in \|\phi\|^n & \text{(lemma 23)} \\ & \text{iff} \quad A \in \|\phi\|^n & \text{(lemma 22)} \end{array}$$

The proofs of these four lemmas are given in the appendix.

9 The Andrew Protocol: Specification

Our chief interest is in the properties of secrecy and authenticity. These concern the agents `Alice` and `Bob` running in an unknown (and potentially hostile) environment Z. Z should be assumed to have access only to channels and data open to outside intruders. Evidently this includes the channel `xfer` (but also the tags `data`, `switch` etc.). However, the initial value of K_{ab} should clearly be regarded as secure, as should the channels `in` and `out`. We adopt the following intuitive account of secrecy and authenticity:

- *Secrecy:* A fresh piece of datum (ie. a secret) received along `in` can only be output along a secret channel.
- *Authenticity:* Only pieces of data previously received along `in` can be output along `out`.

Our aim is to formalise these properties as formulas ϕ for which the following kind of sequent should be established

(6) $Z : \{\texttt{in}, \texttt{out}\}\ \texttt{fresh} \vdash (\nu K_{ab}.\texttt{Alice}\ K_{ab} \mid \texttt{Bob}\ K_{ab}) \mid Z : \phi.$

This is intended to mean that if Z is any agent for which `in` and `out` is not free, the agent obtained by

- putting together `Alice` and `Bob` using K_{ab},

- protecting K_{ab} by a local scope declaration, and
- letting the resulting system run in parallel with Z,

will satisfy the desired property ϕ. By theorem 6 we can for "satisfaction" either use $A \in \pi_2(\|\phi\|)$, or $A \in \|\phi\|^n$ for all $n \in \omega$. The latter gives directly an induction principle which we rely on quite heavily in the proofs that follow.

Formalising secrecy and authenticity in terms of a ϕ in a context such as (6) is not that difficult. For secrecy:

$\vec{a}$ `secret =>`
$[\tau](\vec{a}$ `secret)` $\wedge$
$[$`in`$]($`bound_input` $\supset\ b \rightarrow_\nu\ \vec{a}, b$ `secret)`
$\forall c.[c]($`free_output` $\supset\ \exists d.d \leftarrow ((d \in \vec{a} \supset c \in \vec{a}) \wedge \vec{a}$ `secret))`

Here we use the following two ancillary predicates:

`bound_input` $= a \rightarrow_\nu$ `true`
`free_output` $= \exists a.a \leftarrow$ `true`

For secrecy the property ϕ of (6) becomes $\{$`in`, `out`$\}$ `secret`. The specification of secrecy reflects the intuition very closely. Secrets are either members of the initial value of $\vec{a}$, or they have sometime been input along `in` as a fresh name. Observe that only traces of τ-transitions, name inputs along `in`, or free outputs are considered. This is admissible as correctness is stated of an *open* system: If we accidentally choose a Z which violates secrecy by, say, passing secret-revealing processes to the outside world through an unsafe channel, then there will be another Z which decodes these secret-revealing processes to pass out the (first-order) secrets in a manner that will violate the proof goal.

Authenticity is specified in very similar terms:

$\vec{a}$ `authentic =>`
$[\tau](\vec{a}$ `authentic)` $\wedge$
$[$`in`$]($`bound_input` $\supset\ b \rightarrow_\nu\ \vec{a}, b$ `authentic)`
$[$`out`$]($`free_output` $\supset\ \forall d.d \leftarrow (d \in \vec{a} \wedge \vec{a}$ `authentic))`

and the desired property ϕ of (6) becomes $\{$`in`, `out`$\}$ `authentic`. In this paper we consider the proof of secrecy only.

10 Proving Trust

Secrecy is proved using the `trusted` predicate introduced earlier. For the proofs we consider only the monadic version.

Lemma 7

$$X : \vec{a}, \texttt{in trusted} \vdash X : \vec{a}, \texttt{in secret}$$

PROOF We use a goal driven proof strategy, exploiting the induction principle given by theorem 6. We need to show that whenever $X \in \|\vec{a}, \texttt{in trusted}\|^n$ for all $n \in \omega$ then $X \in \|\vec{a}, \texttt{in secret}\|^m$ for all $m \in \omega$. We use induction in m. The details are straightforward. □

We refer to the m of the above proof as the "approximation index".

The problem of proving secrecy is thus "reduced" to the problem of proving trust. The point of the trustedness predicate is that it lends itself to a structural analysis. The verification takes the shape of series of lemmas intended to support this structural analysis. The most difficult issue is how to deal with parallel composition. In this case we need to be careful about the creation of new internal resources. We extend the sequent notation slightly, following the suggestion of [5], by writing, eg.,

$$X : \phi, Y : \psi \vdash^{\vec{b}} X \mid Y : \gamma$$

to express that whenever X satisfies ϕ and Y satisfies ψ, then $\nu\vec{b}.(X \mid Y)$ satisfies γ, where the scope of $\vec{b}$ includes both ϕ and ψ (but not γ). The delicate part of the trustedness predicate is to deal with the situations in which the "coverage" of the trust predicate needs to be modified because trusted channels are given local scopes, or because trust needs to be extended to channels that are currently unknown to the agent being predicated.

We first consider expanding the set of trusted name to include fresh ones in the case of processes:

Lemma 8

$$X : \vec{a}\ \texttt{trusted}, X : \vec{b}\ \texttt{fresh} \vdash X : \vec{a}, \vec{b}\ \texttt{trusted}$$

PROOF This is a direct consequence of lemma 10 below. But as the result is needed for lemma (9) it needs a separate proof here. The proof is a straightforward induction using approximation indices and is left out. □

As a consequence of lemma 8, if we can show $P : \emptyset\ \texttt{trusted}$ (lemma 13) then we can show $P : \vec{a}\ \texttt{trusted}$ for any set $\vec{a}$ of names that do not occur in P.

We next need a series of results concerning locally scoped names. The first lemma shows that a trusted name can be made local without trust being violated.

Lemma 9

$$X : \vec{a}, \vec{b}\ \texttt{trusted} \vdash^{\vec{b}} X : \vec{a}\ \texttt{trusted}$$

PROOF Assume $\vdash P : \vec{a}, \vec{b}\ \texttt{trusted}$. We show $\nu\vec{b}.P : \vec{a}\ \texttt{trusted}$. So assume $\nu\vec{b}.P \xrightarrow{\alpha} P'$. P' can be assumed to have the shape $\nu\vec{b}.Q$ such that $P \xrightarrow{\alpha} Q$. We proceed by induction in approximation index and cases in Q. The case for $\alpha = \tau$ is trivial so assume that $\alpha = c$.

- Q a process: Trivial, use the induction hypothesis.

- $Q = [d]Q'$, $d \notin \vec{b}$. We get $\nu\vec{b}.Q' : \vec{a}$ `trusted` by the induction hypothesis. Also $d \in \vec{a}$ implies $d \in \vec{a}, \vec{b}$ so $c \in \vec{a}, \vec{b}$ too, by the assumption. But $c \notin \vec{b}$ so $c \notin \vec{a}$ as desired.

- $Q = [d]Q'$ and $d \in \vec{b}$. By the assumption we get $\vdash Q' : \vec{a}, \vec{b}$ `trusted` and, since $d \in \vec{b}$, $c \in \vec{a}, \vec{b}$. But $c \notin \vec{b}$ so $c \in \vec{a}$. Also, by the induction hypothesis we get $\vdash \nu\vec{b} - d.Q' : \vec{a}, d$ `trusted` and we are done.

- $Q = \nu d.[d]Q'$. Use the induction hypothesis.

- $Q = \nu\vec{c}.[Q_1]Q_2$. We need to show

  $$(7) \vdash \nu\vec{b}.\nu\vec{c}.[Q_1]Q_2 : \forall\vec{d}.\vec{d}\ \texttt{fresh} \supset \texttt{new}\ \vec{e}.\\ (\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}) \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}.$$

  Let $\vec{d}$ be fresh and we need to show

  $$(8) \vdash \nu\vec{b}.\nu\vec{c}.[Q_1]Q_2 : \texttt{new}\ \vec{e}.\\ (\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}) \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}.$$

  We choose $\vec{e}$ to have the shape $\vec{b}, \vec{c_1}$ for some $\vec{c_1}$, and (8) is thus reduced to

  $$(9) \vdash \nu\vec{c} - \vec{c_1}.[Q_1]Q_2 : (\vec{a}, \vec{b}, \vec{c_1}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{b}, \vec{c_1}, \vec{d}\ \texttt{trusted})\\ \to \vec{a}, \vec{b}, \vec{c_1}, \vec{d}\ \texttt{trusted}.$$

  By the assumption,

  $$(10) \vdash \nu\vec{c}.[Q_1]Q_2 : \forall\vec{d}.\vec{d}\ \texttt{fresh} \supset \texttt{new}\ \vec{e}.\\ (\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \to \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$$

  so

  $$(11) \vdash \nu\vec{c} - \vec{c_1}.[Q_1]Q_2 : (\vec{a}, \vec{b}, \vec{c_1}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{b}, \vec{c_1}, \vec{d}\ \texttt{trusted})\\ \to \vec{a}, \vec{b}, \vec{c_1}, \vec{d}\ \texttt{trusted},$$

  settling (9) and the case.

- $Q = \lambda d.Q'$. We need to show

  $$(12) \vdash \nu\vec{b}.\lambda d.Q' : \forall d.d \to \vec{a}\ \texttt{trusted}.$$

  which is resolved be the induction hypothesis.

- $Q = \nu\lambda d.Q'$. We need to show

(13) $\vdash \nu\vec{b}.\nu\lambda d.Q' : d \rightarrow_\nu \vec{a}, d$ `trusted`.

This is reduced to

(14) $\vdash \nu\vec{b}.Q' : \vec{a}, d$ `trusted`

where we can choose d to be not free in P. For such a P we can conclude that $\vdash P : \vec{a}, \vec{b}, c$ `trusted`, and hence (14) is obtained.

- $Q = \nu\vec{b}.\lambda x.Q'$. We need to show

(15) $\vdash \nu\vec{b}.\lambda x.Q' : \forall\vec{d}.\vec{d}$ `fresh` $\supset \vec{a}, \vec{d}$ `trusted` $\rightarrow \vec{a}, \vec{d}$ `trusted`.

So let $\vec{d}$ be fresh for $\nu\vec{b}.\lambda x.Q'$, and let $\vdash Q'' : \vec{a}, \vec{d}$ `trusted` and we must show $\vdash \nu\vec{b}.\{Q''/x\}Q' : \vec{a}, \vec{d}$ `trusted`. Now, since $\vec{b}$ is alpha-converted such as not to collide with names free in Q'' we obtain by lemma 8 that also $\vdash Q'' : \vec{a}, \vec{b}, \vec{d}$ `fresh`. By the assumption,

(16) $\vdash \lambda x.Q' : \vec{a}, \vec{b}, \vec{d}$ `trusted` $\rightarrow \vec{a}, \vec{b}, \vec{d}$ `trusted`

so we see that $\vdash \{Q''/x\}Q' : \vec{a}, \vec{b}, \vec{d}$ `trusted`, and then the result follows by the induction hypothesis, concluding the proof.

□

We also need to consider the case of expanding the set of trusted names to fresh ones for functions:

Lemma 10 *Assume that* $X : \vec{a}$ `trusted` $\vdash A : \vec{a}$ `trusted` *and* $\vec{b} \cap \mathrm{fn}(A) = \emptyset$. *Assume also that* $\vdash B : \vec{a}, \vec{b}$ `trusted`. *Then* $\vdash \{B/X\}A : \vec{a}, \vec{b}$ `trusted`. □

The proof of this lemma turns out to be surprisingly delicate, and requires techniques that are somewhat different from the quite elementary techniques used elsewhere in this section. Essentially lemma 10 states a property which is much more "intensional" than the corresponding property 8, concerning, as it does, function definability: All functions in $\vec{a}$ `trusted` $\rightarrow \vec{a}$ `trusted` that do not "mention" $\vec{b}$ can be extended to functions in $\vec{a}, \vec{b}$ `trusted` $\rightarrow \vec{a}, \vec{b}$ `trusted`. The proof of this lemma is deferred to appendix 3.

One further result is needed, to show that local scoping does not affect information which is already trusted.

Lemma 11

$$X : \vec{a} \text{ `trusted`} \vdash^{\vec{b}} X : \vec{a} \text{ `trusted`}$$

PROOF The proof is quite simple, following the inductive proof strategy already introduced with lemma 8. □

We now proceed to the first main result, proving that the trustedness predicate is preserved by parallel composition:

Lemma 12

$$X : \vec{a}, \vec{b}\ \texttt{trusted}, Y : \vec{a}, \vec{b}\ \texttt{trusted} \vdash^{\vec{b}} X \mid Y : \vec{a}\ \texttt{trusted}$$

PROOF See appendix 4. □

To start off proofs of trust in the case of open terms we need to show that all terms respect trustedness of the empty set of names.

Lemma 13 *For all P,* $\vdash P : \emptyset\ \texttt{trusted}$

PROOF See appendix 5. □

11 Deriving a Type System

In this section we show how a type system for inferring judgments of the form $? \vdash^{\vec{a}} A : \vec{b}\ \texttt{trusted}$ can be derived from the results achieved so far. Here $?$ is a set of hypotheses which are either boolean combinations of name equations or inequations, or of one of the forms $x : \vec{a}\ \texttt{trusted}$, or $f : \vec{a}\ \texttt{trusted} \rightarrow \vec{a}\ \texttt{trusted}$. The interpretation of judgments is self-evident: A judgment is true if any substitution of names for names and agents for agent variables that makes the hypotheses true, also makes the conclusion true. We write $? \models F$ where F is an equational or inequational assertion (or a derived form such as $a \in \vec{b}$), if F is a consequence of $?$. The proof system uses an ancillary relation $? \vdash^{\vec{b}} A : c\ \texttt{fresh}$ to hold if c does not occur freely in $\nu\vec{b}.A$, and whenever x (f) is a process (function) variable occurring in $?$ then $? \vdash x : c\ \texttt{fresh}$.

Structural rules:

$$\textsc{Local1}\quad \frac{? \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b},c} P : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Local2}\quad \frac{? \vdash^{\vec{b}} P : \vec{a}, \vec{c}\ \texttt{trusted}}{? \vdash^{\vec{b},\vec{c}} P : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Forget1}\quad \frac{? \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted} \quad ? \vdash^{\vec{b}} P : \vec{c}\ \texttt{fresh}}{? \vdash^{\vec{b}} P : \vec{a}, \vec{c}\ \texttt{trusted}}$$

$$\textsc{Forget2}\quad \frac{? \vdash A : \vec{a}\ \texttt{trusted} \rightarrow \vec{a}\ \texttt{trusted} \quad ? \vdash A : \vec{c}\ \texttt{fresh}}{? \vdash A : \vec{a}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{c}\ \texttt{trusted}}$$

$$\textsc{Cons}\quad \frac{?' \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted} \quad ? \models ?'}{? \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Cases}\quad \frac{?, c = d \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted} \quad ?, c \neq d \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted}}$$

$$\textsc{App}\quad \frac{? \vdash f : \vec{a}\ \texttt{trusted} \to \vec{a}\ \texttt{trusted} \quad ? \vdash P : \vec{a}\ \texttt{trusted}}{? \vdash fP : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Empty}\quad \frac{-}{? \vdash^{\vec{b}} P : \emptyset\ \texttt{trusted}}$$

Term rules:

$$\textsc{Nil}\quad \frac{-}{? \vdash^{\vec{b}} 0 : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Sum}\quad \frac{? \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted} \quad ? \vdash^{\vec{b}} Q : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} P + Q : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Prefix1}\quad \frac{? \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} c.P : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Prefix2}\quad \frac{c \in \vec{b}}{? \vdash^{\vec{b}} c.P : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Prefix3}\quad \frac{? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_out\_after}\ c}{? \vdash^{\vec{b}} c.A : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Prefix4}\quad \frac{? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_in\_after}\ c}{? \vdash^{\vec{b}} c.A : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Par}\quad \frac{? \vdash P : \vec{a}\ \texttt{trusted} \quad ? \vdash Q : \vec{a}\ \texttt{trusted}}{? \vdash P \mid Q : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Nu}\quad \frac{?' \vdash^{\vec{b},\vec{c}} A : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} \nu\vec{c}.A : \vec{a}\ \texttt{trusted}}\quad \text{(See below)}$$

$$\textsc{Cond}\quad \frac{?, c = d \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted} \quad ?, c \neq d \vdash^{\vec{b}} Q : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} \texttt{if}\ c = d\ \texttt{then}\ P\ \texttt{else}\ Q : \vec{a}\ \texttt{trusted}}$$

$$\textsc{Block}\quad \frac{? \vdash^{\vec{b}} P : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} P \backslash c : \vec{a}\ \texttt{trusted}}$$

$$\textsc{In1}\quad \frac{? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_in\_after}\ c}$$

$$\textsc{In2}\quad \frac{? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_in\_after}\ d \quad c\ \mathit{fresh}}{? \vdash^{\vec{b}} \lambda c.A : \vec{a}\ \texttt{trusted\_in\_after}\ d}$$

$$\textsc{In3}\quad \frac{? \models d \in \vec{a} \quad c\ \mathit{fresh} \quad ? \vdash^{\vec{b}} A : \vec{a}, c\ \texttt{trusted\_in\_after}\ d}{? \vdash^{\vec{b}} \nu\lambda c.A : \vec{a}\ \texttt{trusted\_in\_after}\ d}$$

$$\textsc{In4}\quad \frac{? \models d \notin \vec{a} \quad c\ \mathit{fresh} \quad ? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_in\_after}\ d}{? \vdash^{\vec{b}} \nu\lambda c.A : \vec{a}\ \texttt{trusted\_in\_after}\ d}$$

$$\textsc{In5}\quad \frac{?, x : \vec{a}, \vec{c}\ \texttt{trusted} \vdash^{\vec{b}} A : \vec{a}, \vec{c}\ \texttt{trusted\_in\_after}\ d \quad \vec{c}\ \mathit{fresh}}{? \vdash^{\vec{b}} \lambda x.A : \vec{a}\ \texttt{trusted\_in\_after}\ d}$$

$$\textsc{Out1}\quad \frac{? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted}}{? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_out\_after}\ c}$$

$$\textsc{Out2}\quad \frac{c \notin \vec{b} \quad ? \models c \in \vec{a} \supset d \in \vec{a} \quad ? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_out\_after}\ d}{? \vdash^{\vec{b}} [c]A : \vec{a}\ \texttt{trusted\_out\_after}\ d}$$

$$\textsc{Out3}\quad \frac{c \in \vec{b} \quad ? \models d \in \vec{a} \quad ? \vdash^{\vec{b}} A : \vec{a}, c\ \texttt{trusted\_out\_after}\ d}{? \vdash^{\vec{b}} [c]A : \vec{a}\ \texttt{trusted\_out\_after}\ d}$$

$$\textsc{Out4}\quad \frac{c \in \vec{b} \quad ? \models d \notin \vec{a} \quad ? \vdash^{\vec{b}} A : \vec{a}\ \texttt{trusted\_out\_after}\ d}{? \vdash^{\vec{b}} [c]A : \vec{a}\ \texttt{trusted\_out\_after}\ d}$$

$$\textsc{Out5}\quad \frac{\begin{array}{c}?, f : \vec{a}, \vec{c}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{c}, \vec{e}\ \texttt{trusted} \\ \vdash^{\vec{b}-\vec{c}} (fP \mid A) : \vec{a}, \vec{c}, \vec{e}\ \texttt{trusted\_out\_after}\ d \\ \vec{c} \subseteq \vec{b} \quad \vec{e}\ \mathit{fresh}\end{array}}{? \vdash^{\vec{b}} [P]A : \vec{a}\ \texttt{trusted\_out\_after}\ d}$$

The set $?'$ in rule Nu is computed in the following way:

$$?' = ? \cup \{x : \vec{c}\ \texttt{fresh} \mid x \text{ mentioned in } ?\} \cup \{f : \vec{c}\ \texttt{fresh} \mid f \text{ mentioned in } ?\}$$

To terminate proof construction we have the following rule of unfolding and discharge:

$$\frac{\begin{array}{c}[?' \vdash^{\vec{b'}} D(d_1, \ldots, d_n) : \vec{a'}\ \texttt{trusted}] \\ \vdots \\ ? \vdash^{\vec{b}} F(c_1, \ldots, c_n) : \vec{a}\ \texttt{trusted}\end{array}}{? \vdash^{\vec{b}} D(c_1, \ldots, c_n) : \vec{a}\ \texttt{trusted}}$$

The rule is subject to the sidecondition that $?' \vdash^{\vec{b'}} D(d_1, \ldots, d_n) : \vec{a'}\ \texttt{trusted}$ is a substitution instance of $? \vdash^{\vec{b}} D(c_1, \ldots, c_n) : \vec{a}\ \texttt{trusted}$, and that the assumed deduction

$$\begin{array}{c}?' \vdash^{\vec{b'}} D(d_1, \ldots, d_n) : \vec{a'}\ \texttt{trusted} \\ \vdots \\ ? \vdash^{\vec{b}} F(c_1, \ldots, c_n) : \vec{a}\ \texttt{trusted}\end{array}$$

is non-trivial in the sense that it includes the application of a term rule (cf. similar side conditions in [8]).

Theorem 14 (Soundness) *If* $? \vdash^{\vec{b}} A : \vec{a}\ \mathtt{trusted}$ *is provable in the above inference system then it is true.*

PROOF The difficult cases (LOCAL1, LOCAL2, FORGET1, FORGET2, PAR, OUT5) are already dealt with. The remaining rules are quite straightforward. □

Our conjecture is that for the blocking-free fragment of the calculus the type system is complete and decidable.

12 Secrecy of the Andrew Protocol

In this section we use the type system of section 11 to prove the secrecy of the Andrew protocol as stated in section 9. In particular we obtain

Theorem 15

$$Z : \{\mathtt{in}, \mathtt{out}\}\ \mathtt{fresh} \vdash (\nu K_{ab}.\mathtt{Alice}\ K_{ab} \mid \mathtt{Bob}\ K_{ab}) \mid Z : \{\mathtt{in}, \mathtt{out}\}\ \mathtt{secret}$$

PROOF First use lemma 7 to reduce the problem to one of trust instead of secrecy. Now the proof is a straightforward application of the given proof system. We illustrate just the beginning steps of the proof. First the proof goal is reduced to the following three subgoals:

(17) $Z : \{\mathtt{in}, \mathtt{out}\}\ \mathtt{fresh} \vdash Z : \{\mathtt{in}, \mathtt{out}\}\ \mathtt{trusted}$
(18) $\vdash \mathtt{Alice}\ K_{ab} : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$
(19) $\vdash \mathtt{Bob}\ K_{ab} : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$

Of these, (17) is resolved by FORGET1. For (18) we use the term rules (unfolding and sum) to resolve to the following 3 subgoals:

(20) $\vdash \mathtt{in}.\lambda d.\mathtt{xfer!}(K_{ab}![\mathtt{data}][d]0).\mathtt{Alice}K_{ab} : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$
(21) $\vdash \mathtt{in}.\nu\lambda d.\cdots : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$
(22) $\vdash \mathtt{AliceSwitch}K_{ab} : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$

We consider just subgoal (20). Using a few obvious term rules we reduce to:

(23) $\vdash \mathtt{xfer!}(K_{ab}![\mathtt{data}][d]0).\mathtt{Alice}K_{ab} : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$

and then to

(24) $f : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted} \to \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$
$\vdash f(K_{ab}![\mathtt{data}][d]0) \mid \mathtt{Alice}K_{ab} : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$

which is in a few more steps reduced to the following two:

(25) $\vdash K_{ab}![\mathtt{data}][d]0 : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$
(26) $\vdash \mathtt{Alice}K_{ab} : \{\mathtt{in}, \mathtt{out}, K_{ab}\}\ \mathtt{trusted}$

of which (25) is easily dealt with, and (26) is discharged by the unfolding and discharge rule. The proof of subgoal (20) is thus complete. The remaining parts of the proof are completed in similar fashions and left out. □

13 Conclusion

In this paper we have addressed the problem of proving behavioral properties of computationally rather rich higher-order communicating processes in terms of examples drawn from the field of computer security. We have introduced a second-order process calculus based on the π-calculus, and shown, by means of a very simple example, how the important features of security and authentication protocols—viz. nonce generation, key generation, communication, encryption and decryption—can be reduced to features of this process calculus. We have shown also how to account for a simple predicate of trust in a monotonically increasing set of channels using a general second-order temporal logic based on a safety fragment of the modal μ-calculus, and we have shown how to give semantics to this logic in the presence of contravariant recursion in two different, but equivalent, ways. One, iterative, account is useful for deriving induction principles used in the subsequent correctness proofs. We then showed trust of our simple protocol example through a series of lemmas, identifying ways of decomposing the trust predicate according to process structure. Here parallel composition and local name declaration are the difficult connectives. Having obtained these decomposition principles it was a fairly simple matter to devise a sound type system with which correctness of our protocol example could be proved.

Revoking Trust A serious shortcoming of our approach is that revocation of trust is not supported. We have already commented on this issue. The problem is that our trustedness predicate attempts to use general computational features for deciding when to extend and when to revoke trust where in reality these are protocol-specific features. In future work we will have to investigate more refined versions of the trust predicate to address more realistic protocols and properties. Also we have not yet considered proofs of authenticity.

Encryption Primitives We give a direct representation of private key encryption where keys are primitive. We have not yet resolved how to handle variations such as public-key encryption or computed keys in our setting. Public-key encryption is handled using structured channels in CSP/FDR by Lowe [12]—it is quite possible that a similar approach would be useful here. The extension to the basic calculus needed would be quite modest.

Alternative Representations An important issue is to which extent higher-order features and blocking are really needed to adequately represent encryption and decryption. Mainly the choice of representation depends on the level of indirection one is willing to suffer. For instance we are currently building formal evidence to show that our second-order process calculus with blocking can in fact be reduced to first-order π-calculus without blocking. However, the reduction is extremely indirect and not usable as a modelling tool. In the full version of [2]

several alternative representations of encryption and decryption in the π-calculus are discussed. The most interesting idea is to use structured channels as in [12] to represent the encoding of A by the key K as an abstraction

$$\lambda l.(l, K).\lambda l'.A(l').$$

Here A is a unary π-calculus abstraction representing the data to be encoded. Synchronisation is along the structured channel (l, K) to ensure that both the "location" l of $\{A\}_K$ and K itself are present simultaneously. In this approach decryption is made safe as the location l' of $\{A\}_{K_1}$ is not passed to $\{\{A\}_{K_1}\}_{K_2}$ along a publicly known channel. Compared to our approach the π-calculus representation introduces a level of indirection by passing pointers instead of the resources themselves. Thus the model is moved, in our view, one step further away from the physical realities being modelled with the ensuing risk of introducing discrepancies. On the other hand the computational primitives involved are also simpler and computationally more tractable. It would be of interest to relate our second-order representation with a first-order one such as that of Abadi and Gordon, to gain better faith in the correctness of our representations as well as in our accounts of correctness properties. Observe that due to the communication of pointers rather than the objects themselves, a logical account of security and authenticity properties in a first-order setting would be likely to be very different from the account suggested in the present paper.

Acknowledgements

Thanks are due to Martin Abadi, Jose-Luis Vivas and Alan Mycroft for comments and discussions on several topics treated here. It is the credit of Jose-Luis to have observed the need for firewalling using the blocking operator. Also thanks are due to one anonymous referee in particular for some very insightful comments.

References

[1] M. Abadi. Secrecy by typing in security protocols (draft). Manuscript, Available at http://www.research.digital.com/SRC/personal/Martin_Abadi/home.html, 1997.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997. Full version available as tech. rep. 414, Univ. Cambridge Computer Lab.

[3] M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22:2–15, 1996.

[4] R. Amadio and M. Dam. Reasoning about higher-order processes. In *Proc. CAAP'95,* Lecture Notes in Computer Science, 915:202–217, 1995.

[5] R. Amadio and M. Dam. A modal theory of types for the π-calculus. In *Proc. FTRTFT'96,* Lecture Notes in Computer Science, 1135:347–365, 1996.

[6] J.-P. Banâtre, C. Bryce, and D. Le Metayer. Compile time detection of information flow in sequential programs. In *Proc. European Symp. on Research in Computer Security,* LNCS 875, pages 55–73, 1994.

[7] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proc. Royal Society of London A*, 1989.

[8] M. Dam. Model checking mobile processes. *Information and Computation*, 129:35–51, 1996.

[9] D. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20:504–513, 1977.

[10] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow properties. To appear in *IEEE Transactions on Software Engineering.*

[11] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, **32**:137–162, 1985.

[12] G. Lowe. Breaking and fixing the needham-schroeder public-key authentication protocol. *Proc. TACAS,* Lecture Notes in Computer Science, **1055**:147–166, 1996.

[13] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, 1992.

[14] M. Mizuno and D. Schmidt. A security flow control algorithm and its denotational semantics correctness proof. *Formal Aspects of Computing*, 4(1):727–754, 1992.

[15] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[16] P. Ørbæk. Can you trust your data? *Proc. TAPSOFT'95,* Lecture Notes in Computer Science, 915:575–589, 1995.

[17] D. Sangiorgi. From π-calculus to higher-order π-calculus—and back. To appear in Proc. TAPSOFT'93, 1993.

[18] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89:161–177, 1991.

[19] B. Thomsen. A calculus of higher order communicating systems. In *Proc. POPL'89*, pages 143–154, 1989.

[20] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:1–21, 1996.

Appendix 1: Operational Semantics

Agent Syntax

$$\begin{array}{rcl} A & ::= & 0 \mid A + A \mid a.A \mid A \mid A \mid \nu a.A \mid \texttt{if } a = b \texttt{ then } A \texttt{ else } A \mid \\ & & A\backslash a \mid D(a_1, \ldots, a_n) \mid \lambda a.A \mid \lambda x.A \mid \nu\lambda a.A \mid [a]A \mid [P]A \end{array}$$

The Arity Calculus The purpose of the arity calculus is to perform a basic sanity check to ensure eg. that agent expressions do not get bound to channel names as a result of communication, and that, in a communication, the right number of arguments is transferred.

An *arity* is an expression θ of the shape $+w$ or $-w$ where $w \in \{\texttt{chan}, \texttt{agent}\}^*$. The null arity is $()$, and by convention, $+() = () = -()$. Moreover $\overline{+w} = -w$ and $\overline{-w} = +w$. We use the notation $A : \theta$ for arity assignment, disambiguating by context.

$$\frac{\cdot}{O : ()} \quad \frac{A : () \quad B : ()}{A + B : ()} \quad \frac{A : \theta}{a.A : ()} \quad \frac{A : \theta \quad B : ()}{A \mid B : \theta} \quad \frac{A : () \quad B : \theta}{A \mid B : \theta}$$

$$\frac{A : \theta \quad B : \overline{\theta}}{A \mid B : ()} \quad \frac{A : \theta}{\nu a.A : \theta} \quad \frac{A : () \quad B : ()}{\texttt{if } a = b \texttt{ then } A \texttt{ else } B : ()}$$

$$\frac{D = F \quad F : -\texttt{chan}^n}{D(a_1, \ldots, a : n) : ()} \quad \frac{A : -w}{(\nu)\lambda a.A : -\texttt{chan}w} \quad \frac{A : -w}{\lambda x.A : -\texttt{agent}w}$$

$$\frac{A : ()}{A\backslash a : ()} \quad \frac{A : +w}{[a]A : +\texttt{chan}w} \quad \frac{A : () \quad B : +w}{[A]B : +\texttt{agent}w}$$

The Transition Semantics The transition relation is derived from a pair of relations $A \twoheadrightarrow B$ and $A \overset{\alpha}{\twoheadrightarrow} B$ where α is a name or the special symbol τ. The relation $\twoheadrightarrow$ is used for rewriting of agent terms into a normal form by passing and instantiating communicated values, by resolving conditionals, and by commuting lambda's etc. with restrictions. The relation $\overset{\alpha}{\twoheadrightarrow}$ is used to account for the basic rendez-vous mechanism in CCS style. Both the normalisation and the transition relation applies to well-formed agents only.

Normalisation Let a *communication prefix* be an expression *pre* of one of the forms $[a]$, $\nu a.[a]$, $\nu\vec{a}.[P]$, $\lambda a.$, $\nu\lambda a.$, or $\lambda x.$. The normalisation relation is defined by the following clauses:

$$(par)\quad \frac{A \twoheadrightarrow A'}{A \mid B \twoheadrightarrow A' \mid B}$$

$$(merge1)\quad \frac{B : ()}{pre(A) \mid B \twoheadrightarrow pre(A \mid B)}$$

$$(comm1)\quad \frac{A \twoheadrightarrow \lambda a.A' \quad A' : \theta \quad B \twoheadrightarrow [b]B' \quad B' : -\theta}{A \mid B \twoheadrightarrow (\{b/a\}A') \mid B'}$$

$$(comm2)\quad \frac{A \twoheadrightarrow \nu\lambda a.A' \quad A' : \theta \quad B \twoheadrightarrow \nu b.[b]B' \quad B' : -\theta}{A \mid B \twoheadrightarrow \nu b.((\{b/a\}A') \mid B')}$$

$$(comm3)\quad \frac{A \twoheadrightarrow \lambda x.A' \quad A' : \theta \quad B \twoheadrightarrow \nu\vec{a}.[P]B' \quad B' : -\theta}{A \mid B \twoheadrightarrow \nu\vec{a}.(\{P/x\}A') \mid B'}$$

$$(nu1)\quad \frac{\cdot}{\nu a.(\nu)\lambda b.A \twoheadrightarrow (\nu)\lambda b.\nu a.A} \quad (a \neq b)$$

$$(nu2)\quad \frac{\cdot}{\nu a.\lambda x.A \twoheadrightarrow \lambda x.\nu a.A}$$

$$(nu3)\quad \frac{\cdot}{\nu a.[b]A \twoheadrightarrow [b]\nu a.A} \quad (a \neq b)$$

$$(nu4)\quad \frac{\cdot}{\nu a.\nu b.[b]A \twoheadrightarrow \nu b.[b]\nu a.A} \quad (a \neq b)$$

$$(nu5)\quad \frac{A \twoheadrightarrow B}{\nu a.A \twoheadrightarrow \nu a.B}$$

$$(if1)\quad \frac{\cdot}{\texttt{if}\ \ a = a\ \texttt{then}\ \ A\ \texttt{else}\ \ B\ \twoheadrightarrow A}$$

$$(if2)\quad \frac{\cdot}{\texttt{if}\ \ a = b\ \texttt{then}\ \ A\ \texttt{else}\ \ B\ \twoheadrightarrow B}$$

$$(block1)\quad \frac{A \twoheadrightarrow A'}{A\backslash a \twoheadrightarrow A'\backslash a}$$

$$(block2)\quad \frac{\cdot}{(pre(A))\backslash a \twoheadrightarrow pre(A\backslash a)}$$

$$(id)\quad \frac{D = \lambda b_1.\cdots\lambda b_n.P \quad P : ()}{D(a_1,\ldots,a_n) \twoheadrightarrow \{a_1/b_1,\ldots,a_n/b_n\}P}$$

Define the *normal form* of A, $nf(A)$, as the unique B, if it exists, such that $A \twoheadrightarrow^* B$ and $B \not\twoheadrightarrow B'$ for any B'.

The Transition Relation The transition relation is determined by the following rules where we require for $A \stackrel{\alpha}{\to\!\!\to} B$ to be provable that $A : ()$:

$$(norm) \quad \frac{nf(A) \stackrel{\alpha}{\to\!\!\to} B}{A \stackrel{\alpha}{\to\!\!\to} B}$$

$$(sum) \quad \frac{A \stackrel{\alpha}{\to\!\!\to} A'}{A + B \stackrel{\alpha}{\to\!\!\to} A'}$$

$$(alpha) \quad \frac{\cdot}{a.A \stackrel{a}{\to\!\!\to} A}$$

$$(merge2) \quad \frac{A \stackrel{\alpha}{\to\!\!\to} A'}{A \mid B \stackrel{\alpha}{\to\!\!\to} A' \mid B}$$

$$(comm4) \quad \frac{A \stackrel{a}{\to\!\!\to} A' \quad B \stackrel{\bar{a}}{\to\!\!\to} B'}{A \mid B \stackrel{\tau}{\to\!\!\to} A' \mid B'}$$

$$(nu6) \quad \frac{A \stackrel{\alpha}{\to\!\!\to} A'}{\nu a.A \stackrel{\alpha}{\to\!\!\to} \nu a.A'} \quad (\alpha \neq a)$$

$$(block3) \quad \frac{A \stackrel{\alpha}{\to\!\!\to} A'}{A\backslash a \stackrel{\alpha}{\to\!\!\to} A'\backslash a} \quad \alpha \neq a$$

Observe that in the definition of both $\to\!\!\to$ and $\stackrel{\alpha}{\to\!\!\to}$ symmetric cases (for the rules *(par)*, *(merge)*, *(comm)* and *(sum)*) are omitted, and that we in general assume alpha-conversion to be applied whenever necessary to avoid capture of variables. This applies, in particular, for the *(merge)* and *(comm)* rules. Completing the appendix we can now define the transition relation proper.

Definition 16 (The Transition Relation) *Let $A \stackrel{\alpha}{\to} B$ iff for some B', $A \stackrel{\alpha}{\to\!\!\to} B'$ and $\mathrm{nf}(B')$ is defined and equal to B.*

Proposition 17 *If $A \stackrel{a}{\to} A'$ then A' is well-formed and either a process, an abstraction, or a concretion. If $A \stackrel{\tau}{\to} A'$ then A' is well-formed and a process.*

Proof Induction in size of derivations. □

Appendix 2: Proof of Theorem 6

In this appendix we give proofs of lemma's 20, 21, 23, and 22. These lemma's are all proved by induction in formula structure.

We need a little preliminary work. First we observe that both semantics give rise to upper-closed sets. The proof of these two lemmas are routine and left out.

Lemma 18 *Assume that for all X, if $\rho(X) = (S_1, S_2)$ then $S_2 =\uparrow S_2$. Let $\|\phi\|(\rho) = (S_1', S_2')$. Then $S_2' =\uparrow S_2'$.* □

Lemma 19 *Let $\sigma(X) = \uparrow_n (X)$ for all X. Then $\|\phi\|^n(\sigma) = \uparrow_n \|\phi\|^n(\sigma)$.* □

Now we proceed to the four main lemmas. Say that a formula identifier X *occurs in ϕ at depth n* iff X occurs freely in ϕ in the scope of n occurrences of a modal operator or one of the input/output operators $\leftarrow$, $\rightarrow$ etc.

Lemma 20 *Let ϕ and $n \in \omega$ be given. Suppose ρ has the property $(*)$ that whenever a formula identifier X occurs in ϕ at depth n' then for all agents A, $A \in \pi_2(\rho(X))$ iff $\mathtt{trunc}(n - n', A) \in \pi_2(\rho(X))$. Then for all A, $A \in \pi_2(\|\phi\|(\rho))$ iff $\mathtt{trunc}(n, A) \in \pi_2(\|\phi\|(\rho))$.*

Proof Structural induction.

$\phi = \nu X.\phi'$. Suppose first that $A \in \pi_2(\|\phi\|(\rho))$. Then, by corollary 4, for all κ,

(27) $A \in \pi_2(\lambda(S_1, S_2).\|\phi'\|(\{(S_1, S_2)/X\}\rho)^\kappa)$.

We must show that also

(28) $\mathtt{trunc}(n, A) \in \pi_2(\lambda(S_1, S_2).\|\phi'\|(\{(S_1, S_2)/X\}\rho)^\kappa)$.

We proceed by well-founded induction in κ:

- $\kappa = 0$: Trivial.
- $\kappa = \kappa' + 1$: We obtain

  $$A \in \pi_2(\lambda(S_1, S_2).\|\phi'\|(\rho')$$

  where $\rho' = \{\lambda(S_1, S_2).\|\phi'\|(\{(S_1, S_2)/X\}\rho)^{\kappa'}/X\}\rho$. Observe that, by the induction hypothesis, and since X can only occur in guarded positions in ϕ', ρ' has the property $(*)$. Thus we find that (27) is satisfied.
- $\kappa = \sqcup_{\kappa' < \kappa} \kappa'$ is a limit ordinal: This case follows trivially, as $\sqcup$ is set intersection in its second component.

$\phi = \phi_1 \wedge \phi_2$. We obtain:

$A \in \pi_2(\|\phi_1 \wedge \phi_2\|(\rho))$
iff $A \in \pi_2(\|\phi_1\|(\rho))$ and $A \in \pi_2(\|\phi_2\|(\rho))$
iff $\mathtt{trunc}(n, A) \in \pi_2(\|\phi_1\|(\rho))$ and $\mathtt{trunc}(n, A) \in \pi_2(\|\phi_2\|(\rho))$
iff $\mathtt{trunc}(n, A) \in \pi_2(\|\phi_1 \wedge \phi_2\|(\rho))$

$\phi = [a]\phi'$. We obtain:

$A \in \pi_2(\|[a]\phi'\|(\rho))$
iff for all B, if $A \xrightarrow{a} B$ then $B \in \pi_2(\|\phi'\|(\rho))$
iff $n = 0$ or $n = n' + 1$ and for all B, if $A \xrightarrow{a} B$ then
$\mathtt{trunc}(n', B) \in \pi_2(\|\phi'\|(\rho))$
iff $\mathtt{trunc}(n, A) \in \pi_2(\|\phi\|(\rho))$

$\phi = \phi_1 \to \phi_2$. We obtain

$(\lambda x.A) \in \pi_2(\|\phi\|(\rho))$
iff for all B, if $B \in \pi_1(\|\phi_1\|(\rho))$ then $\{B/x\}A \in \pi_2(\|\phi_2\|(\rho))$
iff $n = 0$ or $n = n' + 1$ and for all B, if $B \in \pi_1(\|\phi_1\|(\rho))$ then $\mathtt{trunc}(n', \{B/x\}A) \in \pi_2(\|\phi_2\|(\rho))$
iff $n = 0$ or $n = n' + 1$ and for all B, if $B \in \pi_1(\|\phi_1\|(\rho))$ then $\mathtt{trunc}(n', \{B/x\}\mathtt{trunc}(n', A)) \in \pi_2(\|\phi_2\|(\rho))$
(as $\mathtt{trunc}(n', \{B/x\}A) \approx \mathtt{trunc}(n', \{B/x\}\mathtt{trunc}(n', A))$, and by proposition 18)
iff $\mathtt{trunc}(n, \lambda x.A) \in \pi_2(\|\phi\|(\rho))$

The remaining cases are similar to the above and left to the reader. □

Observe that we take the equivalence

(29) $\mathtt{trunc}(n, \{B/x\}A) \approx \mathtt{trunc}(n, \{B/x\}\mathtt{trunc}(n, A))$

as evident. Now we proceed to the second main lemma, which is proved in a rather similar fashion:

Lemma 21 *Let ϕ and $n \in \omega$ be given. Suppose ρ has the property $(*^2)$ that whenever a formula identifier X occurs in ϕ at depth n' then for all agents A, $\mathtt{trunc}(n - n', A) \in \pi_1(\rho(X))$ iff $\mathtt{trunc}(n - n', A) \in \pi_2(\rho(X))$. Then for all A, $\mathtt{trunc}(n, A) \in \pi_1(\|\phi\|(\rho))$ iff $\mathtt{trunc}(n, A) \in \pi_2(\|\phi\|(\rho))$.*

Proof The proof follows the pattern of the proof of lemma 20. The most difficult case is that of fixed points:

$\phi = \nu X.\phi'$. Assume first that $\mathtt{trunc}(n, A) \in \pi_1(\|\nu X.\phi'\|(\rho))$. Using corollary 4 it follows that we find a κ such that

(30) $\mathtt{trunc}(n, A) \in \pi_1(\lambda(S_1, S_2).\|\phi'\|(\{S_1, S_2)/X\}\rho)^\kappa$.

Observe that, by monotonicity, this same property will hold for all $\kappa' \geq \kappa$. We show, using well-founded induction, that

(31) $\mathtt{trunc}(n, A) \in \pi_2(\lambda(S_1, S_2).\|\phi'\|(\{S_1, S_2)/X\}\rho)^\kappa$.

This proof is quite straightforward and left out. It follows, again by monotonicity, that this same property holds for all $\kappa' < \kappa$. We can thus conclude that

(32) $\mathtt{trunc}(n, A) \in \pi_2(\|\nu X.\phi'\|(\rho))$.

For the converse direction we assume (32) and need to establish a κ for which (30) holds. This is straightforward.

The remaining cases are left for the reader. □

The proof of the following lemma is very similar to the proof of lemma 20 and therefore omitted.

Lemma 22 *Let ϕ and $n \in \omega$ be given. Assume that σ has the property that whenever a formula identifier X occurs in ϕ at depth n' then for all agents A, $\mathtt{trunc}(n - n', A) \in \sigma(X)$ iff $A \in \sigma(X)$. Then for all ϕ, $\mathtt{trunc}(n, A) \in \|\phi\|^n(\sigma)$ iff $A \in \|\phi\|^n(\sigma)$.* □

Finally we arrive at the lemma which ties up the two semantics for truncated agent.

Lemma 23 *Let ϕ and $n \in \omega$ be given. Assume that ρ and σ have the property that whenever a formula identifier X occurs in ϕ at depth n' then for all agents A, $\mathtt{trunc}(n - n', A) \in \pi_1(\rho(X))$ iff $\mathtt{trunc}(n - n', A) \in \sigma(X)$. Then for all ϕ, $\mathtt{trunc}(n, A) \in \pi_1(\|\phi\|(\rho))$ iff $\mathtt{trunc}(n, A) \in \|\phi\|^n(\sigma)$*

Proof The most delicate cases are for the connectives that introduce contravariance. We consider here just the second-order arrow:

$\phi = \phi_1 \to \phi_2$. We obtain:

$\mathtt{trunc}(n+1, \lambda x.A) \in \pi_1(\|\phi\|(\rho))$
iff for all B, if $B \in \pi_2(\|\phi_1\|(\rho))$ then $\mathtt{trunc}(n, \{B/x\}A) \in \pi_1(\|\phi_2\|(\rho))$
iff for all B, if $\mathtt{trunc}(n, B) \in \pi_2(\|\phi_1\|(\rho))$ then
$\mathtt{trunc}(n, \{\mathtt{trunc}(n, B)/x\}A) \in \pi_1(\|\phi_2\|(\rho))$ (lemma 20
iff for all B, if $\mathtt{trunc}(n, B) \in \pi_1(\|\phi_1\|(\rho))$ then
$\mathtt{trunc}(n, \{\mathtt{trunc}(n, B)/x\}A) \in \pi_1(\|\phi_2\|(\rho))$ (lemma 21)
iff for all B, if $\mathtt{trunc}(n, B) \in \|\phi_1\|^n(\sigma)$ then
$\mathtt{trunc}(n, \{\mathtt{trunc}(n, B)/x\}A) \in \|\phi_2\|^n(\sigma)$ (ind. hyp.)
iff for all B, if $B \in \|\phi_1\|^n(\sigma)$ then
$\mathtt{trunc}(n, \{B/x\}A) \in \|\phi_2\|^n(\sigma)$ (lemma 22)
iff $\mathtt{trunc}(n+1, \lambda x.A) \in \|\phi\|^n(\sigma)$

□

Appendix 3: Proof of lemma 10

In this appendix we prove

Lemma 10 *Assume that $x : \vec{a}$ $\mathtt{trusted} \vdash A : \vec{a}$ $\mathtt{trusted}$ and $\vec{b} \cap \mathrm{fn}(A) = \emptyset$. Assume also that $\vdash B : \vec{a}, \vec{b}$ $\mathtt{trusted}$. Then $\vdash \{B/x\}A : \vec{a}, \vec{b}$ $\mathtt{trusted}$.*

Proof Theorem 6 along with lemmas 24 and 25 below. □

The proof of lemma 10 is by induction using theorem 6. We explore the possible computations of the agent $\{B/x\}A$ under the assumption stated in the lemma. In fact we generalise the statement somewhat, to prove the following lemma instead:

Lemma 24 *If*

$x_1 : \vec{a_1}$ `trusted`, $x_1 : \vec{a_1'}$ `fresh`, $\ldots, x_n : \vec{a_n}$ `trusted`, $x_n : \vec{a_n}$ `fresh`
$\vdash \nu\vec{c}.A : \vec{a}$ `trusted`,

and if the following conditions are satisfied:

1. *the sets* $\vec{a}$, $\vec{a_i}$, $\vec{a_i'}$, $\vec{b}$, $\vec{c}$ *are mutually disjoint,*
2. $\vec{b} \cap \mathrm{fn}(A) = \emptyset$,
3. $\vec{c'} \subseteq \vec{c}$,
4. *the variables* $x_1, \ldots, x_n$ *occur linearly in* A,
5. $\vec{a_i} \cup \vec{a_i'} = \vec{a} \cup \vec{c'}$ *for all* $i : 1 \leq i \leq n$,
6. $\vdash B_i : \vec{a}, \vec{b}, \vec{c'}$ `trusted` *for all* $i : 1 \leq i \leq n$,

then $\vdash \nu\vec{c}.\{B_1/x_1\} \cdots \{B_n/x_n\}A : \vec{a}, \vec{b}$ `trusted`.

Most of the remainder of this appendix is devoted to the proof of lemma 24. First, however, we discuss the uses of this lemma. Its significance is, of course, to make the induction needed for 10 work. Concerning linearity the generalisation is in need of justification, however:

Lemma 25 *If* $?, x_1 : \phi, x_2 : \phi \vdash A(x_1, x_2) : \psi$ *then* $?, x : \phi \vdash A(x, x) : \psi$.

Proof Induction in structure of ψ. □

Lemma 25 fails in the presence of the diamond modality. The sets $\vec{c}$, $\vec{c'}$ of (1), (3), (5), (6) are needed to permit substitution to be performed in the context of the set $\vec{c}$ of local channel names, of which some, the members of $\vec{c'}$, are trusted.

We can now proceed to the proof of lemma 24. Let θ be the substitution $\{B_1/x_1\} \cdots \{B_n/x_n\}$. The proof explores the computations of $\nu\vec{c}.\theta A$. For this purpose we need to capture the ways the B_i and the A may interact in performing computation steps. This is done by the following lemma. In the statement of the lemma we tacitly assume that the transition relation $\xrightarrow{\alpha}$ applies to general terms, not only, as is common, terms without free occurrences of process variables.

Lemma 26 *Assume that the variables* $x_1, \ldots x_n$ *occur linearly in* A. *Assume that* $\theta A \xrightarrow{\alpha} C$. *One of the following 4 cases apply:*

1. $A \xrightarrow{\alpha} A'$ *and* $C = \theta A'$.
2. *Some* $B_i \xrightarrow{\alpha} B_i'$ *and* $C = \{B_i'/x_i\}\theta A$.
3. $\alpha = \tau$, *for some* b, B_i, B_j, $B_i \xrightarrow{b} B_i'$, $B_j \xrightarrow{\bar{b}} B_j'$, $i \neq j$, $B_i' \mid B_j'$ *is defined, and* $C = \{B_i'/x_i\}\{B_j'/x_j\}\theta A$.

4. $\alpha = \tau$, *for some* B_i, $B_i \xrightarrow{b} B_i'$, $A \xrightarrow{b} A'$, $B_i' \mid A'$ *is defined, and* $C = \{B_i'/x_i\}\theta A'$

PROOF The proof is by structural induction in A. We consider here only the case of $A = A_1 \mid A_2$. The remaining cases are easier and left to the reader. So assume that $\theta A \xrightarrow{\alpha} C$. One of two cases apply. Either

(i) $\theta A_1 \xrightarrow{\alpha} C_1$ and $C = C_1 \mid \theta A_2$, or

(ii) the symmetrical case applies, or

(iii) $\alpha = \tau$, $\theta A_1 \xrightarrow{b} C_1$, $\theta A_2 \xrightarrow{b} C_2$, $C_1 \mid C_2$ is defined, and $C_1 \mid C_2 = C$.

So assume that (i) holds. By the induction hypothesis one of the cases (1)–(4) holds of A_1.

1. $A_1 \xrightarrow{\alpha} A_1'$ and $C_1 = \theta A_1'$. Then $A_1 \mid A_2 \xrightarrow{\alpha} A_1' \mid A_2$ and $C = \theta A_1' \mid A_2$.

2. $B_i \xrightarrow{\alpha} B_i'$ and $C_1 = \{B_i'/x_i\}\theta A_1$. Then, due to the assumption of linearity, $C = \{B_i'/x_i\}\theta A$ as required.

3. $\alpha = \tau$, $B_i \xrightarrow{b} B_i'$, $B_j \xrightarrow{b} B_j'$, $B_i' \mid B_j'$ defined, $i \neq j$, and

$$C_1 = \{B_i'/x_i\}\{B_j'/x_j\}\theta A_1.$$

   Then we can use linearity to conclude that $C = \{B_i'/x_i\}\{B_j'/x_j\}\theta(A_1 \mid A_2)$.

4. $\alpha = \tau$, $B_i \xrightarrow{b} B_i'$, $A_1 \xrightarrow{b} A_1'$, and $C_1 = \{B_i'/x_i\}\theta A_1'$. Then, using linearity, $A_1 \mid A_2 \xrightarrow{b} A_1' \mid A_2$, and $C_1 \mid C_2$ has the right form.

So assume instead (iii). Again we use the induction hypothesis to conclude that one of (1)–(4) must hold for A_1 and A_2 respectively. Actually, (3) and (4) are both rules out, as these apply to τ-transitions only.

- (1)–(1): Assume $A_1 \xrightarrow{b} A_1'$, $C_1 = \theta A_1'$, $A_2 \xrightarrow{b} A_2'$, and $C_2 = \theta A_2'$. Then $A_1 \mid A_2 \xrightarrow{\tau} A_1' \mid A_2'$ and $C_1 \mid C_2$ has the desired shape.

- (1)–(2): Assume $A_1 \xrightarrow{b} A_1'$, $C_1 = \theta A_1'$, $B_i \xrightarrow{b} B_i'$, and $C_2 = \{B_i'/x_i\}\theta A_2$. Then we find that $A_1 \mid A_2 \xrightarrow{b} A_1' \mid A_2$ and $C = \{B_i'/x_i\}\theta(A_1' \mid A_2)$ so that (4) holds of A.

- (2)–(2): Assume $B_i \xrightarrow{b} B_i'$, $C_1 = \{B_i'/x_i\}\theta A_1$, $B_j \xrightarrow{b} B_j'$, $C_2 = \{B_j'/x_j\}\theta A_2$. Due to linearity, $i \neq j$. Also $B_i' \mid B_j'$ is defined. Moreover, $C_1 \mid C_2 = \{B_i'/x_i\}\{B_j'/x_j\}\theta(A_1 \mid A_2)$ so that (3) holds of A.

The case (2)–(1) is symmetrical to the (1)–(2) case and left out. The proof is thus complete. □

We need another lemma, mainly to show that if a process concretion of the shape $[P]A$ has the property $(\vec{a}\ \texttt{trusted} \rightarrow \vec{a}\ \texttt{trusted}) \rightarrow \vec{a}\ \texttt{trusted}$ then $\vdash P : \vec{a}\ \texttt{trusted}$ too. In fact a more general property holds:

Lemma 27 *If* $\vdash \nu\vec{c}.(P \mid Q) : \vec{a}\ \texttt{trusted}$ *then for some* $c' \subseteq c$, $\vdash P : \vec{a}, \vec{c'}\ \texttt{trusted}$ *and* $\vdash Q : \vec{a}, \vec{c'}\ \texttt{trusted}$, *and* $x : \vec{a}, \vec{c'}\ \texttt{trusted}, y : \vec{a}, \vec{c'}\ \texttt{trusted} \vdash \nu\vec{c}.(x \mid y) : \vec{a}\ \texttt{trusted}$.

Proof We show that we can find a $\vec{c'}$ such that $\vdash P : \vec{a}, \vec{c'}\ \texttt{trusted}$ by induction in the approximation index as usual. So assume that $P \xrightarrow{\alpha} A$. There are a number of cases to consider for which we consider the most difficult one only:

Assume that $A = \nu d.[A_1]A_2$. Then $\alpha = b$ for some b and

$$\nu\vec{c}.(P \mid Q) \xrightarrow{b} \nu\vec{c}.\nu\vec{d}.[A_1](A_2 \mid Q),$$

assuming bound names are alpha-converted as appropriate. As $\vdash \nu\vec{c}.(P \mid Q) : \vec{a}\ \texttt{trusted}$ we must also have that

(34) $\nu\vec{c}.\nu\vec{d}.[A_1](A_2 \mid Q) : \forall\vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{f}.$
$(\vec{a}, \vec{e}, \vec{f}\ \texttt{trusted} \rightarrow \vec{a}, \vec{e}, \vec{f}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{e}, \vec{f}\ \texttt{trusted}.$

Let $\vec{e}$ be fresh for $\nu\vec{d}.[A_1]A_2$. By choosing names appropriately we can be sure that $\vec{e}$ is also fresh for $\nu\vec{c}.\nu\vec{d}.[A_1](A_2 \mid Q)$, so that

(35) $\nu\vec{c}.\nu\vec{d}.[A_1](A_2 \mid Q) : \texttt{new}\ \vec{f}.$
$(\vec{a}, \vec{e}, \vec{f}\ \texttt{trusted} \rightarrow \vec{a}, \vec{e}, \vec{f}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{e}, \vec{f}\ \texttt{trusted}.$

We find $\vec{f_1} \subseteq \vec{c}$ and $\vec{f_2} \subseteq d$ such that

(36) $\nu\vec{c} - \vec{f_1}.\nu\vec{d} - \vec{f_2}.[A_1](A_2 \mid Q) :$
$(\vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted}.$

Now, let $\vdash F : \vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted}$. We get that

(37) $\nu\vec{c} - \vec{f_1}.\nu\vec{d} - \vec{f_2}.(FA_1) \mid A_2 \mid Q : \vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted}.$

By the induction hypothesis we see that we can find $\vec{f_1'} \subseteq \vec{c} - \vec{f_1}$, and $\vec{f_2'} \subseteq \vec{d} - \vec{f_2}$ such that

(38) $FA_1 : \vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted}$
(39) $A_2 : \vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted}$
(40) $Q : \vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted}.$

Choose $\vec{c'} = \vec{f_1} \cup \vec{f_1'}$ and $\vec{f} = \vec{f_2} \cup \vec{f_2'}$. Observe that neither $\vec{c'}$ nor $\vec{f}$ have elements occurring freely in F, so $\vec{c'}$ and $\vec{f}$ can be chosen independently of F. Let us verify that

$$(41)\ \nu\vec{d}[A_1]A_2 : (\vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted}$$

So assume that

$$(42) \vdash F : \vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{f_1}, \vec{f_1'}, \vec{f_2}, \vec{f_2'}, \vec{e}\ \texttt{trusted}$$

Since neither $\vec{f_1'}$ nor $\vec{f_2'}$ have elements in common with F we obtain that also

$$(43) \vdash F : \vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{f_1}, \vec{f_2}, \vec{e}\ \texttt{trusted},$$

and so the desired conclusion can be drawn, appealing to the observation above of independence of choice of F, c', and f.

Before completing the proof one slightly delicate matter needs to be attended to. Observe that our choice of $\vec{f_1}$ and $\vec{f_2}$ depends on the choice of A. However, it is only the size of the sets $\vec{f_1}$ and $\vec{f_2}$ that matter, as bound names can be alpha-converted. As the underlying transition system is finitely-branching maximal sizes of sets $\vec{f_1}$ and $\vec{f_2}$ can be determined. Notice now that it is possible to "pad" these sets, if necessary, using dummy names that are never used, to obtain sets of equal size, independent of the choice of A.

Having made this observation it is now a simple matter to realise that the second part of the lemma is also true. □

Corollary 28 *If*

$$\vdash \nu\vec{c}.[P]Q : \nu\vec{d}.(\vec{a}, \vec{d}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{d}\ \texttt{trusted}$$

then for some $\vec{c'} \subseteq \vec{c}$, $\vdash P : \vec{a}, \vec{c'}\ \texttt{trusted}$ *and* $\vdash Q : \vec{a}, \vec{c'}\ \texttt{trusted}$, *and* $x : \vec{a}, \vec{c'}\ \texttt{trusted}, y : \vec{a}, \vec{c'}\ \texttt{trusted} \vdash \nu\vec{c}.[x]y : \nu\vec{d}.(\vec{a}, \vec{d}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{d}\ \texttt{trusted}$.

Proof Use lemma 27. □

Now we can proceed to the proof of lemma 24. Assume that $\nu\vec{c}.\theta A \xrightarrow{\alpha} C$. By lemma 26 one of the cases (1)–(4) of the statement of the lemma applies (where $\alpha \notin \vec{c}$). We proceed accordingly.

1. We have that $A \xrightarrow{\alpha} A'$ and $C = \nu\vec{c}.\theta A'$. We use the definition of $\vec{a}, b\ \texttt{trusted}$.
   - $\alpha = \tau$ and C is a process. The induction hypothesis applies.
   - $\alpha = d$ and C is a process. The induction hypothesis applies.
   - $\alpha = d$ and $C = [e]C'$ (or, more generally, $C = \nu\vec{c}.[e]C'$ and $e \notin \vec{c}$). We have that $A' = [e]A''$ for some A'' as the x_i are process variables, and

(44) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}\ \texttt{fresh}, \ldots, x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}\ \texttt{fresh}$
$\vdash \nu\vec{c}.A' : \exists e.e \leftarrow (\vec{a}\ \texttt{trusted} \wedge (e \in \vec{a} \supset d \in \vec{a}))$

thus

(45) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}\ \texttt{fresh}, \ldots, x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}\ \texttt{fresh}$
$\vdash \nu\vec{c}.A'' : \vec{a}\ \texttt{trusted}$

and $e \in \vec{a} \supset d \in \vec{a}$. By the induction hypothesis we find that $\vdash \nu\vec{c}.\theta A'' : \vec{a}, \vec{b}\ \texttt{trusted}$. Assume also that $e \in \vec{a}, \vec{b}$. Then $e \in \vec{a}$ as $d \notin fn(A)$. So $e \in \vec{a}, \vec{b}$ too, and $\vdash \nu\vec{c}.\theta A : \vec{a}, \vec{b}\ \texttt{trusted}$.

- $\alpha = d$ and C is a bound output agent. This case is similar and left out.
- $\alpha = d$ and $C = \nu\vec{c}.[C']C''$ where we can assume that $A' = [A_1]A_2$, $\theta A_1 = C'$ and $\theta A_2 = C''$. We get that

(46) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}\ \texttt{fresh}, \ldots, x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}\ \texttt{fresh}$
$\vdash \nu\vec{c}.A' : \forall\vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{d}.$
$(\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

Let $\vec{e}$ be fresh for $\theta A'$. We need to show that

(47) $\vdash \nu\vec{c}.[\theta A_1]\theta A_2 : \texttt{new}\ \vec{d}.$
$(\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}$,

where $\vec{c}$ is chosen to satisfy the conditions of the lemma. So we need to find $\vec{d}$ such that

(48) $\vdash \nu\vec{c} - \vec{d}.[\theta A_1]\theta A_2 :$
$(\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}$.

Assume that

(49) $\vdash \lambda y.D : \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}$.

We need to show

(50) $\vdash \nu\vec{c} - \vec{d}.\{\theta A_1/y\}D \mid \theta A_2 :$
$\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}$.

To show this using the induction hypothesis we need to show the following two subgoals:

(51) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{d}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{d}, \vec{e}\ \texttt{fresh},$
$x : \vec{a}, \vec{c'}, \vec{d}, \vec{e}\ \texttt{trusted}$
$\vdash \nu\vec{d} - \vec{d}.x \mid A_2 : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

(52) $\vdash \{\theta A_1/y\}D : \vec{a}, \vec{b}, \vec{c'}, \vec{d}, \vec{e}\ \texttt{trusted}$

where $\vec{c'} \subseteq \vec{c} - \vec{d}$. To prove first (51), from (46) we see that

(53) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{e}\ \texttt{fresh},$
$\vdash \nu\vec{c}.A' : \texttt{new}\ \vec{d}.(\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted})$
$\to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

where $\vec{e}$ is appropriately chosen. So we find that

(54) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{d}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{d}, \vec{e}\ \texttt{fresh},$
$\vdash \nu\vec{d} - \vec{d}.[A_1]A_2 : (\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted})$
$\to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

(choosing in the process the $\vec{d}$), and then

(55) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{d}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{d}, \vec{e}\ \texttt{fresh},$
$F : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$
$\vdash \nu\vec{c} - \vec{d}.(FA_1) \mid A_2 : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}.$

Now $\vdash \lambda x.x : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \to \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$ so we find that

(56) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{d}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{d}, \vec{e}\ \texttt{fresh}$
$\vdash \nu\vec{c} - \vec{d}.A_1 \mid A_2 : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}.$

By lemma 27 we can find $\vec{c'} \subseteq \vec{c} - \vec{d}$ such that

(57) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{d}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{d}, \vec{e}\ \texttt{fresh}$
$\vdash A_1 : \vec{a}, \vec{c'}, \vec{d}, \vec{e}\ \texttt{trusted},$

(58) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{d}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{d}, \vec{e}\ \texttt{fresh}$
$\vdash A_2 : \vec{a}, \vec{c'}, \vec{d}, \vec{e}\ \texttt{trusted},$

(59) $x : \vec{a}, \vec{c'}, \vec{d}, \vec{e}\ \texttt{trusted}, y : \vec{a}, \vec{c'}, \vec{d}, \vec{e}\ \texttt{trusted}$
$\vdash \nu\vec{c} - \vec{d}.x \mid y : \vec{a}, \vec{de}\ \texttt{trusted}$

Observe that $\vec{c'}$ can be chosen uniformly in x and y. We thus find that (51) holds. To show (52) we use (57) along with the induction hypothesis to show that

(60) $\theta A_1 : \vec{a}, \vec{b}, \vec{c'}, \vec{d}, \vec{e}\ \texttt{trusted}$

and another application of the induction hypothesis to the sequent

(61) $x : \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \vdash F(x) : \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}$

establishes (52) and the subcase is complete.

- $\alpha = d$ and $C = \nu\vec{c}.\lambda d.C'' = \lambda d.\nu\vec{c}.C''$, or $C = \nu\vec{c}.\nu\lambda d.C''$. The proof is left out.
- $\alpha = d$ and $C = \nu\vec{c}.\lambda x.C'' = \lambda x.\nu\vec{c}.C''$, so that A' has the shape $\lambda x.A''$ and $C'' = \nu\vec{c}.\theta A''$ up to naming of x. We get that

  (62) $x_1 : \vec{a_1}$ `trusted`$, x_1 : \vec{a_1'}$ `fresh`$, \ldots,$
  $x_n : \vec{a_n}$ `trusted`$, x_n : \vec{a_n'}$ `fresh`
  $\vdash \nu\vec{c}.A' : \forall\vec{e}.\vec{e}$ `fresh` $\supset \vec{a}, \vec{e}$ `trusted` $\rightarrow \vec{a}, \vec{e}$ `trusted`.

  We conclude

  (63) $x_1 : \vec{a_1}$ `trusted`$, x_1 : \vec{a_1'}, \vec{e}$ `fresh`$, \ldots,$
  $x_n : \vec{a_n}$ `trusted`$, x_n : \vec{a_n'}, \vec{e}$ `fresh`
  $x : \vec{a}, \vec{e}$ `trusted`
  $\vdash \nu\vec{c}.A'' : \vec{a}, \vec{e}$ `trusted`.

  Now the result follows directly from the induction hypothesis.

2. We have that $B_i \xrightarrow{\alpha} B_i'$ and $C = \nu\vec{c}.\{B_1/x_1\}\cdots\{B_i'/x_i\}\cdots\{B_n/x_n\}A$. Again we follow the def. of $\vec{a}, \vec{b}$ `trusted`.

   - $\alpha = \tau$ or $\alpha = d$ and C is a process. The induction hypothesis applies.
   - $\alpha = d$ and $C = \nu\vec{c}.[e]C'$, $e \notin \vec{c}$. In this case B_i' will have the shape $[e]B_i''$ and

     (64) $[e]B_i'' : \exists e.e \leftarrow (\vec{a}, \vec{b}, \vec{c'}$ `trusted` $\wedge\ e \in \vec{a}, \vec{b}, \vec{c'} \supset d \in \vec{a}, \vec{b}, \vec{c'})$

     thus

     (65) $B_i'' : \vec{a}, \vec{b}, \vec{c'}$ `trusted`

     so by the induction hypothesis, $\vdash \nu\vec{c}.C' : \vec{a}, \vec{b}$ `trusted`. Also $e \in \vec{a}, \vec{b}$ implies $d \in \vec{a}, \vec{b}$ as $d \notin \vec{c'}$. So indeed

     (66) $\vdash \nu\vec{c}.\{B_1/x_1\}\cdots\{B_i'/x_i\}\cdots\{B_n/x_n\}A : \vec{a}, \vec{b}$ `trusted`.

   - $\alpha = d$ and $C = \nu\vec{c}.[e]C''$ and $e \in \vec{c}$. This case is left for the reader.
   - $\alpha = d$ and C has the shape $\nu\vec{c}.\nu\vec{f}.[C']C''$ where $B_i' = \nu\vec{f}.[B_{i,1}]B_{i,2}$, $C' = B_{i,1}$ and $C'' = \{B_1/x_1\}\cdots\{B_{i,2}/x_i\}\cdots\{B_n/x_n\}A$. We get that

     (67) $B_i' : \forall\vec{e}.\vec{e}$ `fresh` $\supset$ `new` $\vec{f}.$
     $(\vec{a}, \vec{b}, \vec{c'}, \vec{e}, \vec{f}$ `trusted` $\rightarrow \vec{a}, \vec{b}, \vec{c'}, \vec{e}, \vec{f}$ `trusted`$)$
     $\rightarrow \vec{a}, \vec{b}, \vec{c'}, \vec{e}, \vec{f}$ `trusted`.

     We need to show that

     (68) $\nu\vec{c}.\nu\vec{f}.[B_{i,1}]C'' : \forall\vec{e}.\vec{e}$ `fresh` $\supset$ `new` $\vec{f}.$
     $(\vec{a}, \vec{b}, \vec{e}, \vec{f}$ `trusted` $\rightarrow \vec{a}, \vec{b}, \vec{e}, \vec{f}$ `trusted`$)$
     $\rightarrow \vec{a}, \vec{b}, \vec{e}, \vec{f}$ `trusted`.

So let $\vec{e}$ be fresh for C. We can assume that $\vec{e}$, $\vec{c}$ and $\vec{f}$ are disjoint. (68) is then reduced first to

(69) $\nu\vec{c}.\nu\vec{f}.[B_{i,1}]C'' : \texttt{new}\ \vec{f}.(\vec{a},\vec{b},\vec{e},\vec{f}\ \texttt{trusted} \to \vec{a},\vec{b},\vec{e},\vec{f}\ \texttt{trusted}) \to \vec{a},\vec{b},\vec{e},\vec{f}\ \texttt{trusted}.$

and then we need to find $\vec{c_1}$ and $\vec{f_1}$ such that

(70) $\nu\vec{c}-\vec{c_1}.\nu\vec{f}-\vec{f_1}.[B_{i,1}]C'' :$
$(\vec{a},\vec{b},\vec{c_1},\vec{e},\vec{f_1}\ \texttt{trusted} \to \vec{a},\vec{b},\vec{c_1},\vec{e},\vec{f_1}\ \texttt{trusted})$
$\to \vec{a},\vec{b},\vec{c_1},\vec{e},\vec{f_1}\ \texttt{trusted}.$

So assume that

(71) $\vdash \lambda y.D : \vec{a},\vec{b},\vec{c_1},\vec{e},\vec{f_1}\ \texttt{trusted} \to \vec{a},\vec{b},\vec{c_1},\vec{e},\vec{f_1}\ \texttt{trusted}$

and we need to show

(72) $\nu\vec{c}-\vec{c_1}.\nu\vec{f}-\vec{f_1}.\{B_{i,1}/y\}D \mid C'' : \vec{a},\vec{b},\vec{c_1},\vec{e},\vec{f_1}\ \texttt{trusted}.$

To prove this by the induction hypothesis we need to find sets $\vec{c_2} \subseteq \vec{c}-\vec{c_1}$ and $\vec{f_2} \subseteq \vec{f}-\vec{f_1}$ such that

(73) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'},\vec{e},\vec{f_1},\vec{f_2}\ \texttt{fresh},\ldots,$
$x_i : \vec{a_i},\vec{e},\vec{f_1},\vec{f_2}\ \texttt{trusted}, x_i : \vec{a_i'}\ \texttt{fresh},\ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'},\vec{e},\vec{f_1},\vec{f_2}\ \texttt{fresh},$
$y : \vec{a_i},\vec{e},\vec{f_1},\vec{f_2}\ \texttt{trusted}, y : \vec{a_i'}\ \texttt{fresh}$
$\vdash \nu\vec{c}-\vec{c_1}.\nu\vec{f}-\vec{f_1}.y \mid A : \vec{a},\vec{c_1},\vec{e},\vec{f_1}\ \texttt{trusted}$

(74) $y : \vec{a},\vec{c_1},\vec{c_2},\vec{e},\vec{f_1},\vec{f_2}\ \texttt{trusted} \vdash D : \vec{a},\vec{c_1},\vec{c_2},\vec{e},\vec{f_1},\vec{f_2}\ \texttt{trusted}$

(75) $\vdash B_{i,j} : \vec{a_i},\vec{b},\vec{e},\vec{f_1},\vec{f_2}\ \texttt{trusted},\ j \in \{1,2\}.$

To prove (73), by assumption we have

(76) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}\ \texttt{fresh},\ldots,$
$x_i : \vec{a_i}\ \texttt{trusted}, x_1 : \vec{a_i'}\ \texttt{fresh},\ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_1 : \vec{a_n'}\ \texttt{fresh}$
$\vdash \nu\vec{c}.A : \vec{a}\ \texttt{trusted}$

from which we can deduce

(77) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}\ \texttt{fresh},\ldots,$
$x_i : \forall\vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{f}.(\vec{a_i},\vec{e},\vec{f}\ \texttt{trusted} \to \vec{a_i},\vec{e},\vec{f}\ \texttt{trusted}) \to \vec{a_i},\vec{e},\vec{f}\ \texttt{trusted}, x_i : \vec{a_i'}\ \texttt{fresh},\ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}\ \texttt{fresh}$
$\vdash \nu\vec{c}.A : \forall\vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{f}.(\vec{a},\vec{e},\vec{f}\ \texttt{trusted} \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}) \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}$

and

(78) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_i : \texttt{new}\ \vec{f}.(\vec{a_i}, \vec{e}, \vec{f}\ \texttt{trusted} \to \vec{a_i}, \vec{e}, \vec{f}\ \texttt{trusted})$
$\to \vec{a_i}, \vec{e}, \vec{f}\ \texttt{trusted}, x_i : \vec{a_i'}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{e}\ \texttt{fresh}$
$\vdash \nu\vec{c}.A : \texttt{new}\ \vec{f}.(\vec{a}, \vec{e}, \vec{f}\ \texttt{trusted} \to \vec{a}, \vec{e}, \vec{f}\ \texttt{trusted})$
$\to \vec{a}, \vec{e}, \vec{f}\ \texttt{trusted}$

and then

(79) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{e}\ \texttt{fresh}, \ldots,$
$x_i : \texttt{new}\ \vec{f}.(\vec{a_i}, \vec{e}, \vec{f}\ \texttt{trusted} \to \vec{a_i}, \vec{e}, \vec{f}\ \texttt{trusted})$
$\to \vec{a_i}, \vec{e}, \vec{f}\ \texttt{trusted}, x_i : \vec{a_i'}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{e}\ \texttt{fresh}$
$\vdash \nu\vec{c} - \vec{c_1}.A : \texttt{new}\ \vec{f}.(\vec{a}, \vec{c_1}, \vec{e}, \vec{f}\ \texttt{trusted} \to \vec{a}, \vec{c_1}, \vec{e}, \vec{f}\ \texttt{trusted})$
$\to \vec{a}, \vec{c_1}, \vec{e}, \vec{f}\ \texttt{trusted}.$

Now we use the assumption on x_i to instantiate the local names $\vec{f}$ to the left of the turnstile as two parts, $\vec{f_1}$ which is named by $\vec{f}$ to the right of the turnstile, and a part $\vec{f_2}$ which becomes bound by $\vec{f} - \vec{f_1}$:

(80) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{e}, \vec{f_1}\ \texttt{fresh}, \ldots,$
$x_i : \texttt{new}\ \vec{f_2}.(\vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted} \to \vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted})$
$\to \vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted}, x_i : \vec{a_i'}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{e}, \vec{f_1}\ \texttt{fresh}$
$\vdash \nu\vec{c} - \vec{c_1}.A : (\vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted} \to \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted})$
$\to \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted},$

from where we get

(81) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{e}, \vec{f_1}\ \texttt{fresh}, \ldots,$
$x_i : \texttt{new}\ \vec{f_2}.(\vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted} \to \vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted})$
$\to \vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted}, x_i : \vec{a_i'}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{e}, \vec{f_1}\ \texttt{fresh},$
$f : \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted} \to \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted}$
$\vdash \nu\vec{c} - \vec{c_1}.(f(x_i) \mid A) : \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted}.$

Now, using the assumption on x_i we see:

(82) $x_1 : \vec{a_1}\ \texttt{trusted}, x_1 : \vec{a_1'}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{fresh}, \ldots,$
$x_i : (\vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted} \to \vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted})$
$\to \vec{a_i}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{trusted}, x_i : \vec{a_i'}\ \texttt{fresh}, \ldots,$
$x_n : \vec{a_n}\ \texttt{trusted}, x_n : \vec{a_n'}, \vec{e}, \vec{f_1}, \vec{f_2}\ \texttt{fresh},$
$f : \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted} \to \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted},\ f : \vec{f_2}\ \texttt{fresh}$
$\vdash \nu\vec{c} - \vec{c_1}.\nu\vec{f} - \vec{f_1}.(f(x_i) \mid A) : \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}\ \texttt{trusted}$

where $\vec{f}$ includes $\vec{f_2}$. Now we use the induction hypothesis to conclude

(83) $f : \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}$ `trusted` $\rightarrow \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}$ `trusted`, $f : \vec{f_2}$ `fresh`
$\vdash f : \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}, \vec{f_2}$ `trusted` $\rightarrow \vec{a}, \vec{c_1}, \vec{e}, \vec{f_1}, \vec{f_2}$ `trusted`

which suffices to conclude (73). Subgoal (74) follows by the induction hypothesis applied to (71). Finally (75) is easily proved by the methods already introduced.

- The cases for name and process input are left to the reader.

3. Of the numerous subcases consider the following: $B_i \xrightarrow{d} \nu\vec{e}.[B_{i,1}]B_{i,2}$ and $B_j \xrightarrow{d} \lambda y.B'_j$, and our task is then to prove that

(84) $\vdash \nu\vec{c}.\nu\vec{e}.\{B_1/x_1\} \cdots \{B_{i,2}/x_i\} \cdots \{\{B_{i,1}/y\}B'_j/x_j\} \cdots \{B_n/x_n\}A :$
$\vec{a}$ `trusted`

where $\vec{e}$ is chosen to not be confused with other variables. The proof is quite an easy exercise given the previous steps, and left for the reader.

4. Similar comments apply to the final case.

This completes the proof. □

Appendix 4: Proof of lemma 12

In this appendix we prove

Lemma 12

$$X : \vec{a}, \vec{b} \text{ `trusted`}, Y : \vec{a}, \vec{b} \text{ `trusted`} \vdash^{\vec{b}} X \mid Y : \vec{a} \text{ `trusted`}$$

Proof Reducing the proof goal we obtain two subgoals of the shapes:

(85) $X : \vec{a}, \vec{b}$ `trusted`, $Y : \vec{a}, \vec{b}$ `trusted` $\vdash^{\vec{b}} X \mid Y : [\tau]\vec{a}$ `trusted`
(86) $X : \vec{a}, \vec{b}$ `trusted`, $Y : \vec{a}, \vec{b}$ `trusted` $\vdash^{\vec{b}} X \mid Y : \forall c.[c](\vec{a}$ `trusted` $\vee \cdots)$

Considering first (85) observe that a τ transition of $\nu\vec{b}.X \mid Y$ can be due to either (1) a τ-transition of X, (2) a τ-transition of Y (symmetrical to (1)), or (3) a communication of X and Y resulting in agents of matching arity (ie. a name abstraction matching a name concretion, etc.). Thus we see that we can reduce (85) the following list of subgoals:

(87) $X : \vec{a}, \vec{b}\ \texttt{trusted}, Y : \vec{a}, \vec{b}\ \texttt{trusted} \vdash^{\vec{b}} X \mid Y : \vec{a}\ \texttt{trusted}$

(88) $X : \exists c.c \leftarrow (\vec{a}, \vec{b}\ \texttt{trusted} \wedge (c \in \vec{a} \supset d \in \vec{a})),$
$Y : \forall c.c \rightarrow \vec{a}, \vec{b}\ \texttt{trusted})$
$\vdash^{\vec{b}} X \mid Y : \vec{a}\ \texttt{trusted}$

(89) $X : \texttt{new}\ c.c \leftarrow ((d \in \vec{a}, \vec{b} \supset (\vec{a}, \vec{b}, c\ \texttt{trusted})) \wedge (d \notin \vec{a}, \vec{b} \supset \vec{a}, \vec{b}\ \texttt{trusted}))$
$Y : c \rightarrow_{\nu} (d \in \vec{a}, \vec{b} \supset \vec{a}, \vec{b}, c\ \texttt{trusted}) \wedge (d \notin \vec{a}, \vec{b} \supset \vec{a}, \vec{b}\ \texttt{trusted})$
$\vdash^{\vec{b}} X \mid Y : \vec{a}\ \texttt{trusted}$

(90) $X : \forall \vec{d}.\vec{d}\ \texttt{fresh} \supset \texttt{new}\ \vec{c}.$
$(\vec{a}, \vec{b}, \vec{d}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{c}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{c}\ \texttt{trusted}$
$Y : \forall \vec{c}.\vec{c}\ \texttt{fresh} \supset \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \vec{a}\ \texttt{trusted}$

Subgoal (87) arises in several ways: Because a τ-transition of X, because of a τ-transition of Y, or because of a synchronisation where both X and Y evolve into processes. In any case the subgoal is immediately resolved by the induction hypothesis. Subgoal (88) is resolved by a little elementary reasoning to a sequent of the shape (87). Subgoal (89) is equally easy to resolve. Finally we address subgoal (90). Letting $\vec{d} = \emptyset$ we obtain:

(91) $X : \texttt{new}\ \vec{c}.(\vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$
$Y : \forall \vec{c}.\vec{c}\ \texttt{fresh} \supset \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \vec{a}\ \texttt{trusted}$

which is reduced to

(92) $X : (\vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$
$\vec{c}\ \texttt{fresh}$
$Y : \vec{c}\ \texttt{fresh} \supset \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$
$\vdash^{\vec{b},\vec{c}} X \mid Y : \vec{a}\ \texttt{trusted}$

which can be decomposed into two subgoals:

(93) $X : (\vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$
$Y : \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$
$\vdash X \mid Y : \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted}$

(94) $X : \vec{a}, \vec{b}, \vec{c}\ \texttt{trusted} \vdash^{\vec{b},\vec{c}} \vec{a}\ \texttt{trusted}$

of which (93) is immediate, and (94) is an application of lemma 9. This completes the proof of (85). We need also to consider (86). Exploiting the symmetry of $\mid$ this goal can be resolved to the following (long) list of subgoals:

(95) $X : \vec{a}, \vec{b}\ \texttt{trusted}, Y : \vec{a}, \vec{b}\ \texttt{trusted} \vdash^{\vec{b}} X \mid Y : \vec{a}\ \texttt{trusted}$

(96) $X : d \notin \vec{b}, \quad X : c \notin \vec{b}$
$X : d \leftarrow (\vec{a}, \vec{b}\ \texttt{trusted} \wedge (d \in \vec{a}, \vec{b} \supset c \in \vec{a}, \vec{b})),$

$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \exists d.d \leftarrow (\vec{a}\ \texttt{trusted} \wedge (d \in \vec{a} \supset c \in \vec{a}))$

(97) $X : d \in \vec{b}, \quad X : c \notin \vec{b}$
$X : d \leftarrow (\vec{a}, \vec{b}\ \texttt{trusted} \wedge (d \in \vec{a}, \vec{b} \supset c \in \vec{a}, \vec{b}))$,
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \texttt{new}\ d.d \leftarrow ((c \in \vec{a} \supset \vec{a}, d\ \texttt{trusted}) \wedge (c \notin \vec{a} \supset \vec{a}\ \texttt{trusted}))$

(98) $X : c \notin \vec{b}$
$X : \texttt{new}\ d.d \leftarrow ((c \in \vec{a}, \vec{b} \supset \vec{a}, \vec{b}, d\ \texttt{trusted}) \wedge (c \notin \vec{a}, \vec{b} \supset \vec{a}, \vec{b}\ \texttt{trusted}))$
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \texttt{new}\ d.d \leftarrow ((c \in \vec{a} \supset \vec{a}, d\ \texttt{trusted}) \wedge (c \notin \vec{a} \supset \vec{a}\ \texttt{trusted}))$

(99) $X : c \notin \vec{b}$
$X : \forall \vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{d}.$
$(\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}$
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \forall \vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{d}.$
$(\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

(100) $X : c \notin \vec{b}$
$X : \forall d.d \rightarrow \vec{a}, \vec{b}\ \texttt{trusted})$
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \forall d.d \rightarrow \vec{a}\ \texttt{trusted})$

(101) $X : c \notin \vec{b}$
$X : d \rightarrow_\nu (c \in \vec{a}, \vec{b} \supset \vec{a}, \vec{b}, d\ \texttt{trusted}) \wedge (c \notin \vec{a}, \vec{b} \supset \vec{a}, \vec{b}\ \texttt{trusted})$
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : d \rightarrow_\nu \vec{a}\ \texttt{trusted})$

(102) $X : c \notin \vec{b}$
$X : \forall \vec{d}, \vec{d}\ \texttt{fresh} \supset \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \forall \vec{d}, \vec{d}\ \texttt{fresh} \supset \vec{a}, \vec{d}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}\ \texttt{trusted}$

Subgoal (95) is as usual resolved immediately by the induction hypothesis. This applies also to (96) given a minute amount of boolean reasoning. Subgoal (96) and (97) arise because of the assumption of a free output for X followed by existential elimination to the left and a case analysis on d. In the latter case the free output of X becomes a bound output of the entire process, due to the local scope of $\vec{b}$. Subgoal (97) is reduced to

(103) $X : d \in \vec{b}, \quad X : c \notin \vec{b}$
$X : d \leftarrow (\vec{a}, \vec{b}\ \texttt{trusted} \wedge (d \in \vec{a}, \vec{b} \supset c \in \vec{a}, \vec{b}))$,
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$

$$\vdash^{\vec{b}-d} X \mid Y : d \leftarrow ((c \in \vec{a} \supset \vec{a}, d\ \texttt{trusted}) \wedge (c \notin \vec{a} \supset \vec{a}\ \texttt{trusted}))$$

and further, using a little boolean reasoning, to

(104) $X : d \in \vec{b}, \quad X : c \notin \vec{b}$
$X : \vec{a}, \vec{b}\ \texttt{trusted}$
$c \in \vec{a},$
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}-d} X \mid Y : d \leftarrow \vec{a}, d\ \texttt{trusted}$

which follows by the induction hypothesis. Subgoal (98) is straightforward. Now, for (99) we reduce this to

(105) $X : d \notin \vec{b},$
$X : (\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$
$Y : \vec{e}\ \texttt{fresh}, \quad Y : d\ \texttt{fresh}$
$Y : \vec{a}, \vec{b}\ \texttt{trusted},$
$\vdash^{\vec{b}} X \mid Y : (\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

and then, using lemma 9, to

(106) $X : d \notin \vec{b},$
$X : (\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$
$Y : \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$
$\vdash^{\vec{b}} X \mid Y : (\vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

Now we use right arrow introduction:

(107) $X : d \notin \vec{b},$
$X : (\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$
$Y : \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$
$Z : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$
$Z : \vec{b}\ \texttt{fresh}$
$\vdash^{\vec{b}} X \mid Y \mid Z : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

We can now appeal to lemma 10 to reduce to

(108) $X : d \notin \vec{b},$
$X : (\vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}) \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$
$Y : \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted},$
$Z : \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted} \rightarrow \vec{a}, \vec{b}, \vec{d}, \vec{e}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y \mid Z : \vec{a}, \vec{d}, \vec{e}\ \texttt{trusted}$

which is resolved by the induction hypothesis. We leave out the proofs of subgoal (100) and (101). Finally we need to consider (102). We first reduce this to

(109) $X : c \notin \vec{b} \quad X : \vec{d}\ \texttt{fresh}$
$X : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$Y : \vec{d}\ \texttt{fresh}$,
$Y : \vec{a}, \vec{b}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \vec{a}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{d}\ \texttt{trusted}$

and then, using lemma 8 to

(110) $X : c \notin \vec{b} \quad X : \vec{d}\ \texttt{fresh}$
$X : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$Y : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$\vdash^{\vec{b}} X \mid Y : \vec{a}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{d}\ \texttt{trusted}$

Now we introduce $\to$ to the right:

(111) $X : c \notin \vec{b} \quad X : \vec{d}\ \texttt{fresh}$
$X : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$Y : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$Z : \vec{a}, \vec{d}\ \texttt{trusted}$
$Z : b\ \texttt{fresh}$
$\vdash^{\vec{b}} (X \mid Y)(Z) : \vec{a}, \vec{d}\ \texttt{trusted}$

Observe that we here use am applicative notation which strictly speaking is not in the syntax of the process calculus. Using 8 once more we obtain

(112) $X : c \notin \vec{b} \quad X : \vec{d}\ \texttt{fresh}$
$X : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted} \to \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$Y : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$Z : \vec{a}, \vec{b}, \vec{d}\ \texttt{trusted}$
$\vdash^{\vec{b}} (X \mid Y)(Z) : \vec{a}, \vec{d}\ \texttt{trusted}$

which is provable by the induction hypothesis. □

Appendix 5: Proof of lemma 13

In this appendix we prove

Lemma 13 *For all P, $\vdash P : \emptyset\ \texttt{trusted}$.*

For the proof we need the following lemma, permitting us to make free names private while preserving trustedness of names.

Lemma 29 *Suppose $\vec{a} \cap \vec{b} = \emptyset$. Then*

$$X : \vec{a}\ \texttt{trusted} \vdash \nu\vec{b}.X : \vec{a}\ \texttt{trusted}$$

PROOF Suppose $\vdash P : \vec{a}\ \texttt{trusted}$ and $\nu\vec{b}.P \xrightarrow{\alpha} P'$. Then P' has the shape $\nu\vec{b}.Q$ such that $P \xrightarrow{\alpha} Q$. We proceed by induction in approximation index and cases in Q as usual. The case for $\alpha = \tau$ is trivial, so assume $\alpha = c$.

- $Q = [d]Q''$. Then $\vdash Q' : \vec{a}\ \texttt{trusted}$, and $d \in \vec{a}$ implies $c \in \vec{a}$. Whether or not $d \in \vec{b}$ we get that $\nu\vec{b}.Q'$ has the desired property.
- $Q = \nu\vec{d}.[Q_1]Q_2$. We need to show

  $$(113) \vdash \nu\vec{b}.\nu\vec{d}.[Q_1]Q_2 : \forall\vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{f}.\\ (\vec{a},\vec{e},\vec{f}\ \texttt{trusted} \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}) \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}.$$

  Reduce this to

  $$(114) \vdash \nu\vec{b}.\nu\vec{d}.[Q_1]Q_2 : \texttt{new}\ \vec{f}.(\vec{a},\vec{e},\vec{f}\ \texttt{trusted} \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}) \\ \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}$$

  where $\vec{e}$ is fresh, and then to

  $$(115) \vdash \nu\vec{b}.\nu\vec{d} - \vec{d_1}.[Q_1]Q_2 : (\vec{a},\vec{e},\vec{d_1}\ \texttt{trusted} \to \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}) \\ \to \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}.$$

  So let $\vdash \lambda y.D : \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted} \to \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}$ and we need to show

  $$(116) \vdash \nu\vec{b}.\nu\vec{d} - \vec{d_1}.\{Q_1/y\}D \mid Q_2 : \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}.$$

  By assumption we have

  $$(117) \vdash \nu\vec{d}.[Q_1]Q_2 : \forall\vec{e}.\vec{e}\ \texttt{fresh} \supset \texttt{new}\ \vec{f}.\\ (\vec{a},\vec{e},\vec{f}\ \texttt{trusted} \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}) \to \vec{a},\vec{e},\vec{f}\ \texttt{trusted}.$$

  So for some $\vec{c'} \subseteq \vec{c}$,

  $$(118) \vdash \nu\vec{d} - \vec{d_1}.[Q_1]Q_2 : (\vec{a},\vec{e},\vec{d_1}\ \texttt{trusted} \to \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}) \\ \to \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}.$$

  So we have

  $$(119) \vdash \nu\vec{d} - \vec{d_1}.\{Q_1/y\}D \mid Q_2 : \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}.$$

  But then $\vdash \nu\vec{b}.\nu\vec{d} - \vec{d_1}.\{Q_1/y\}D \mid Q_2 : \vec{a},\vec{e},\vec{d_1}\ \texttt{trusted}$ by the induction hypothesis.
- The remaining cases are straightforward.

□

We can now proceed to the proof of lemma 13 which is by induction in approximation index as usual.

So assume that $P \xrightarrow{\alpha} P'$. Assume $\alpha = a$. We proceed by cases in P' as usual, observing that the case for P' as process is trivial:

- P' free output of shape $[b]P''$. By the induction hypothesis $\vdash P'' : \emptyset\ \texttt{trusted}$ and $b \in \emptyset \supset a \in \emptyset$ completing the case.
- P' bound output of shape $\nu b.[b]P''$. As $a \notin \emptyset$ and $\vdash P'' : \emptyset\ \texttt{trusted}$ by the induction hypothesis we get

  $$(120) \vdash \nu b.[b]P'' : \texttt{new}\ c.c \leftarrow ((a \in \emptyset \supset c\ \texttt{trusted}) \wedge (a \notin \emptyset \supset \emptyset\ \texttt{trusted}))$$

  as desired.
- P' process concretion of shape $\nu\vec{b}.[P_1]P_2$. Let $\vec{c}$ be fresh. We must show

  $$(121) \vdash \nu\vec{b}.[P_1]P_2 : \texttt{new}\ \vec{d}.(\vec{c}, \vec{d}\ \texttt{trusted} \rightarrow \vec{c}, \vec{d}\ \texttt{trusted}) \rightarrow \vec{c}, \vec{d}\ \texttt{trusted}.$$

  We reduce this to showing

  $$(122) \vdash \nu\vec{b}.[P_1]P_2 : (\vec{c}\ \texttt{trusted} \rightarrow \vec{c}\ \texttt{trusted}) \rightarrow \vec{c}\ \texttt{trusted}.$$

  So let $\vdash \lambda y.D : \vec{c}\ \texttt{trusted} \rightarrow \vec{c}\ \texttt{trusted}$ and we must show

  $$(123) \vdash \nu\vec{b}.\{P_1/y\}D \mid P_2 : \vec{c}\ \texttt{trusted}.$$

  Now $\vdash P_1 : \vec{c}\ \texttt{fresh}$ so $\vdash P_1 : \vec{c}\ \texttt{trusted}$ by lemma 8, and thus

  $$(124) \vdash \{P_1/y\}D : \vec{c}\ \texttt{trusted}.$$

  Also $\vdash P_2 : \vec{c}\ \texttt{fresh}$ so also $\vdash P_2 : \vec{c}\ \texttt{trusted}$. But then

  $$(125) \vdash \{P_1/y\}D \mid P_2 : \vec{c}\ \texttt{trusted}$$

  by lemma 12, and so by lemma 29 we conclude (123).
- The remaining cases are straightforward.

The proof of lemma 13 is thus complete. □