Modelling Properties of Security Protocols

Henrik Pilegaard

LYNGBY 2002
EKSAMENSPROJEKT
NR. 67

IMM

Trykt af IMM, DTU

Preface

This report constitutes my Master of Science Thesis, written during the period from February 1st, 2002 to November 1st, 2002, in the Secure and Safe Systems group of the Computer Science & Engineering section at the Informatics and Mathematical Modeling department of the Technical University of Denmark.

My supervisors have been Associate Professor Michael R. Hansen, Associate Professor Robin Sharp, and to some extent PhD. Thomas M. Rasmussen who unfortunately left the department while this thesis was in the making. I would like to thank my supervisors for always finding the time for answering my questions, discuss problems, and otherwise help me in the making of this thesis.

I would also like to thank Mikael, Rene, Morten, Torben, Jan, Johan, Thomas, Mikkel, Michael, and Jesper for making every meal at the canteen a pleasure - no easy task.

Finally, thanks to my beloved - Anette Jensen, for making every other aspect of life easy during days of stress.

Lyngby, November 1st, 2002

Henrik Pilegaard

Abstract

Meadows has shown how *availability* issues regarding *security protocols* may be treated by analyzing the time-consuming *internal actions* of agents with respect to a *fail-stop* criteria that is cost dependent.

In this thesis we investigate a method of combining Paulson's *inductive approach* to protocol analysis with *Interval Logic* in order to create a verification tool that supports analysis of the kind Meadows proposes.

Based on a novel notion of packets and a redefined notion of *external events* we develop theoretical alterations and extensions that enables the inductive method to distinguish *active attacks*. To supplement the *global traces* of external events we define an inductive theory of (untimed) *local traces* of internal actions.

Based on notions of *timed global traces* and *timed local traces* we develop a method that allows natural modeling of *real-time* properties of security protocols via *Neighborhood Logic*, an adequate first order modal logic.

Proof support for the developed theory is achieved via encoding in Isabelle/LSILHOL an Isabelle/HOL environment for *labelled interval logics*. Several small examples of protocols are treated and properties are shown via interactive theorem proving.

Keywords

security protocols, availability, inductive approach, fail-stop, Interval Logic, neighborhood logic, labelled interval logics.

Resumé

Meadows har vist, hvordan spørgsmål om *tilgængelighed* vedrørende *sikkerhedsprotokoller* kan behandles ved at analysere agenters tidskrævende *interne aktioner* med hensyn til *fail-stop* kriterier, som er omkostningsafhængige.

I dette speciale undersøges en metode til at kombinere Paulsons *induktive metode* til verifikation af sikkerhedsprotokoller med *intervallogik* for at lave et verifikationsværktøj, der understøtter analyser af den slags, som Meadows foreslår.

Med udgangspunkt i *pakker* og *hændelser* ændrer og udvider vi den induktive metode til at kunne skelne mellem aktive og passive angreb. For at supplere de *globale spor* af eksterne hændelser definerer vi en induktiv teori for (ikke tidslige) *lokale spor* af interne aktioner.

På baggrund af *tidslige globale spor* og *tidslige lokale spor* udvikler vi en metode der tillader naturlig modellering af *sandtidsegenskaber ved sikkerhedsprotokoller* vha. *omegnslogik*, en tilstrækkelig første ordens intervallogik.

Assisteret bevisførelse for den udviklede teori er etableret ved hjælp af Isabelle/LSILHOL - et Isabelle/HOL miljø der understøtter *mærkede intervallogikker*. Et antal mindre eksempler på protokoller er behandlet og egenskaber ved dem er vist vha. assisteret bevisførelse.

Nøgleord

Sikkerhedsprotokoller, tilgængelighed, induktiv metode, fail-stop, intervallogik, omegnslogik, mærkede intervallogikker.

Notation

If not explicitly stated otherwise the following notational conventions will be used throughout this thesis.

Boldfaced italics will be used for all sets, e.g. $\boldsymbol{bool}$, except well-known ones, such as $\mathbb{N}$. In order to obtain a uniform presentation we do not distinguish between sets and types.

Sans serif will be used for all functions and predicates, e.g. $\mathsf{isFalse}$. In most cases we use the functional arrow $\rightarrow$ in order to describe the types of functions, but for injective functions we use $\hookrightarrow$.

We make the one element of singleton sets $\boldsymbol{1}$ anonymous $\{\star\}$.

Sometimes we explicitly name the members of a set via injective functions from e.g. $\mathbb{N}$. If the source is a singleton set, however, it is not mentioned, i.e. we don't write $\mathsf{True} : \{*\} \hookrightarrow \boldsymbol{bool}$ but rather $\mathsf{True} : \boldsymbol{bool}$

List can be formed by elements of any set $\boldsymbol{'a}$, e.g. $\boldsymbol{'a\ list}$. We write the empty list as [] and use # as the list construction operator. The first element of a non-empty list is retrieved by $\mathsf{hd} : \boldsymbol{'a\ list} \rightarrow \boldsymbol{'a}$, and the list of elements after the first is retrieved by $\mathsf{tl} : \boldsymbol{'a\ list} \rightarrow \boldsymbol{'a\ list}$. The set of elements of a list is produced by the function $\mathsf{set} : \boldsymbol{'a\ list} \rightarrow \mathcal{P}(\boldsymbol{'a})$

When defining functions or predicates we use the operator $\hat{=}$, e.g. $\mathsf{isBool}\ b \mathrel{\hat{=}} b \in \boldsymbol{bool}$. When defining syntactic abbreviations we use the operator $\equiv$, e.g. $x\#xs \equiv \mathsf{cons}(x, xs)$. Finally, when defining type abbreviations we use the operator $=$, e.g. $\boldsymbol{nat} = \mathbb{N}$.

Beware that when used *in* the defined theories $=$ denotes normal equality.

Contents

List of Figures

CHAPTER 1

Introduction

It is hard to imagine a time when communication was not essential to the everyday life of human beings. And as history readily testifies there are times when communication should be conducted securely. E.g. with strong guarantees that the *secret* contents is not disclosed to third-parties (*confidentiality*), the contents has not been altered by third-parties (*integrity*), or that a given entity, e.g. a service-provider, is always reachable (*availability*).

In this modern age computers are very handy tools of communication. *Protocols* are the descriptions of behavioral patterns that allow pieces of information to be transferred from one entity to another. Such protocols have certain *goals* and can only be considered *correct* if these are all achieved. Otherwise, entities relying on the protocol may fail, possibly causing disaster.

Entities may be of any type. They may be components of the same computer located adjacent to one another inside a box. They may also be two entirely different computers located at opposite ends of an open computer network, such as the Internet. There is, however, an important difference between these two situations.

If the entities are both inside the same protected environment, the protocol, like two persons talking face to face in an invisible and soundproof box, need not pursue *security goals* because no third-person entities are likely to *interfere* with the communication and thereby *compromising* it.

In open networks the situation is different. While messages are in transit across the network a malicious entity - a *spy* - may alter, steal, block, or interfere with them in other ways. When the prevention of any such

malicious interference is a *security goal* of the protocol we call it a *security protocol.*

While *goal verification* for simple non-security protocols has made great progress and is now a well known discipline the similar treatment of security protocols has shown to be difficult. There are three important goals that may be regarded as prerequisites of protocol security in a broader sense. *Confidentiality* concerns prevention of the release of secret contents from messages. *Integrity* concerns the prevention of unauthorized alteration of messages in transit. And finally *Availability* concerns guaranteeing that certain services will always be granted.

During the last two decades especially goals relying on confidentiality and integrity have been scrutinized thoroughly by researchers. This is not the case for goals related to availability.

Availability is a diverse notion. There are aspects of availability, such as physically damaged communication equipment, that are not directly related to protocol issues. Those that are, rely on the protocols ability to deal with requests in a timely fashion - even when the host system is being stressed by e.g. *Denial of Service* attacks aiming to exhaust its resources. In other words, availability may be considered a *real-time* property of protocols.

In general security protocols constitute an application area that demands a very high level of trust in correctness. Such high degrees of trust are almost impossible to obtain without *formal modeling* and subsequent *mechanical verification.*

1.1 Thesis Objectives

This thesis considers the analysis of real-time properties in connection with security protocols, e.g. to ensure that a server has a certain availability for "normal users" even when attacked by intruders with given capabilities.

The project has aimed to combine an existing logic, for reasoning about (untimed) security properties, with an interval logic, for reasoning about timing properties. It is our thesis that this combination gives a powerful tool for reasoning about real-time security properties.

The work carried out consists of the following parts:

- Development of a suitable combination of Paulson's Inductive method [Pau98] and a labelled variant [Ras02] of Neighborhood Logic [CH98].
- Mechanization of this combination in the interactive theorem prover Isabelle/HOL [NPW02].
- Testing the tool on small examples.

1.2 Theoretical Background

The earliest successful approach to formal goal verification was the *authentication* (BAN) logic by Burrows, Abadi, and Needham [BAN89], which was based on a notion of *belief*. Simply applied to abstract specifications after an idealization step the logic argues universally about authentication by derivation of guarantees from each message in the protocol. Though various authors, such as Abadi [AT91], Needham [AN96], Gong [GNY90], Syverson [Syv93], Diffie [DvOW92], van Oorschot [vO93], and Wiener [DvOW92], have worked on it, the logic has never been really successful when applied to goals other than authentication. Boyd and Mao [BM93] have shown that the idealization step is inherently ambiguous.

Newer research seems to be concentrated around more general approaches. Various researchers such as Lowe [Low97], Ryan [Rya96], Schneider [Sch97], Meadows [Mea01] have made great progress with state exploration methods based on the operational semantics of protocols, i.e. a concrete notion of system events and operational models for their use. These approaches are quite good at finding flaws in finite models, but cannot, however, verify universal properties of unbounded models.

With his Inductive Approach [Pau98] Paulson has shown that an approach to analysis of security protocols based on inductively defined traces is effective when the goals to be verified do not depend on time. Like state exploration this method is based on operational semantics but abandons the state enumeration in favor of mathematical induction and theorem proving. The method is based on Higher-Order Logic and mechanical verification is supported via the interactive theorem prover Isabelle/HOL.

Notably, the analysis of Denial of Service (DoS) is not feasible within Paulson's setup. A key observation in this respect is that attacks may be divided into *passive attacks*, where intruders read, but do not change or fabricate, messages sent over a network, and *active attacks*, where intruders change or

fabricate messages in an unauthorized manner, e.g. to impersonate somebody else [Sta99].

Gong and Syverson have introduced the notion of *fail-stop protocols* [GS95]. A protocol satisfies the fail-stop criteria if it automatically halts in response to any active attack. Thus, having proved that a protocol is fail-stop, one only has to consider passive attacks.

Meadows [Mea99] has suggested how the fail-stop mechanism may be extended into a framework, where the costs of detecting the various kinds of fake messages are taken into account. The idea is that messages are checked by computationally cheap procedures early in the protocol execution while the more expensive ones are used as late as possible in order to obtain a good use of resources. Availability issues may then be evaluated based on a concrete model of the internal actions involved in the verification procedure, their costs, and a predefined relationship between attackers intrusive strength and the point in the protocol execution where they *must* have been rejected.

1.3 Contribution

The goal of this work is proof assistance for the analysis of security protocols, focusing on active attacks and availability questions. The work is based on the approaches of Paulson and Meadows.

1.3.1 Starting Point

The starting point of our work is Paulson's setup that analyses protocols at an abstraction level corresponding to Alice-and-Bob specifications:

$$\begin{aligned} A_1 &\rightarrow B_1 : M_1 \\ &\vdots \\ A_n &\rightarrow B_n : M_n \end{aligned}$$

where the ith step reads: "agent A_i sends the message M_i to agent B_i."

This is adequate as long as one is not interested in availability issues in which case Meadows shows that we need to analyze at a level corresponding

to augmented Alice-and-Bob specifications:

$$\begin{aligned} A_1 \rightarrow B_1 &: \mathrm{T}_1 \| M_1 \| \mathrm{O}_1 \\ &\vdots \\ A_n \rightarrow B_n &: \mathrm{T}_n \| M_n \| \mathrm{O}_n \end{aligned}$$

where the ith step reads: "agent A_i performs the sequence T_i of message preparation events and then sends the message M_i to agent B_i. Agent B_i receives message M_i and performs the sequence O_i of message verification actions in order to validate the message."

Since the latter analysis is harder it should only be applied when necessary, i.e. when there is an active attack. Paulson's setup deals adequately with passive attacks and we do not treat them in this thesis.

1.3.2 Identifying Active Attacks

Dealing with active attacks, the problem is that things need not be what they seem to be, like when you receive a packet by mail. The packet has a content (e.g. a message) and it contains information about its origin and its destination. One cannot be sure of the correctness of this information since in active attacks any part of it may have been tampered with.

To allow a theoretical distinction between active and passive attacks we introduce the notion of *packets* having the form:

$$(a_s, l_s) \rightarrow (a_r, l_r) : M$$

where a_s is the alleged origin, l_s the true origin (location), a_r the desired destination, and l_r the actual destination (whoever manages to get the message).

Also we introduce a notion of *events*, where agents can *send*, *receive*, and *block* packets. Agents' interaction with the network is then modeled as inductively defined sets of *global traces* of such events.

When interacting with the network agents are ignorant to packet information besides that which is prescribed by the Alice-and-Bob notation, i.e. alleged sender, desired recipient, and message.

Packets and events seem to constitute a convenient modeling foundation considering the various ways in which packets may be tampered with in active attacks. The packets allow us to judge when events are *fake* and (even

though agents cannot distinguish between good events and fake events) we may use this to identify active attacks.

Assume a *realistic model* of the *global traces* of the protocol, which includes the malicious activities of a spy. Also assume an *ideal model* of the protocol, which only contains traces corresponding to faithful executions.

The set of active attacks are then: Those global traces for which a *projection*, with respect to some agent, A, contains a fake event, but the *idealized projection* (focusing on the Alice-and-Bob information), with respect to A, is indistinguishable from the similar projection of some trace from the ideal model.

The remaining set of global traces may be thought of as passive attacks.

1.3.3 Analyzing Active Attacks

Having identified active attacks we need to analyze them in an extended setup that also accounts for the *internal actions* of agents.

We identify a suitable set of internal activities that relate to message preparation and verification. Based on this notion af *actions* the internal behavior of agents can be modeled via a notion of *local traces*.

In order to analyze the timed properties of the protocols modeled by local and global traces we introduce a notion of *timed traces*. By *timed traces* we understand functions from time to untimed traces. We develop a theory of timed traces within an interval logic framework where the theory is expressed in Neighborhood Logic.

We then cope with meadows cost-based approach to fail-stop protocols by interpreting costs as the time it takes to perform given actions. The timed theory also describes the causal connections between events and actions, as e.g. an occurrence of a receive event in the global trace may initiate a sequence of local actions.

With this setup we can model real-time aspects of security protocols, where the timed global trace describes the global events occurring on the network and the timed local traces describe internal actions in the individual agents.

1.4 Thesis Organization

This thesis consists of eight chapters of which this introduction is the first. An overview of the remaining seven chapters is given in the following:

In chapter 2 we account for the view of the domain of security protocols that motivates the theoretical work carried out in this thesis. We introduce central notions, notations, and theoretical approaches.

In chapter 3 we present a theoretical framework derived from the work of Paulson. We show how his inductive approach may be extended in a way that allows us to make theoretical distinctions between active and passive attacks and model attackers in a more detailed manner.

In chapter 4 we exemplify our theoretical extensions of the Inductive Approach described in chapter 3. The Station-to-Station protocol is formalized and an active attack found by Lowe is formally shown to be recognized as such.

In chapter 5 we present an untimed modeling of Meadows' internal actions. We show that a smaller and different set of actions are more appropriate for our purposes. We explain the general form of inductively defined local traces and give a simple example.

In chapter 6 we introduce a labelled Neighborhood logic. We then present a modeling method, based on the notions of timed global traces and timed local traces, that is adequate for describing both timing issues and causal relationships between global events and local actions.

In chapter 7 we exemplify the outlined method for the modeling of real-time properties of protocols described in chapter 6. We formalize simple pseudo-protocols. In one instance we focus on causality of global events, and in another we capture causality between local actions and global events.

Appendix A contains the Isabelle/HOL encoding of the entire theoretical setup. **The structure of presentation in this appendix is completely faithful to the structure of the main document.**

Appendix B contains the axioms and definitions of Rasmussen's Isabelle/-HOL encodings of labelled first order Interval Logics.

Appendix C contains some reasonably premature preparations for a case study of the IPSEC-OAKLEY protocol.

Appendix D contains some experiments with mutually inductive definitions in Isabelle/HOL.

Appendix E contains the source of the a2ps style file that had to be used to create the material in appendix D.

CHAPTER 2

Setting the Scene

> *The subject of this thesis is security protocols - Specifications for the format and relative timing of messages exchanged securely between processes in open distributed systems.*

While this statement appears to be very concise it actually leaves a number of things to be defined. In order to justify our theoretical developments concerning security protocols we must further clarify our views.

There are a number of well-established notions that need to be presented because they are central to the domain. Of equal importance are the assumptions regarding the domain that we necessarily make in order to formalize it.

Both notions and assumptions will be informally described in the following as we aim to clarify our *view* of the domain:

- We say that protocols are *specifications*. We elaborate on this and introduce the *Alice-and-Bob notation* for specifying protocols (§2.1).
- By *open distributed systems* we in fact mean *peer networks* where *processes* are the *peers*. We describe such environments and their threats (§2.2).
- Due to the potentially hostile nature of open networks correctness of *open distributed systems* depends on *security protocols*. We introduce a number of *security aspects* in the form of *security goals* (§2.3).
- Obviously our theoretical efforts are somewhat inspired by the work of other people. Paulson [Pau98] and Meadows [Mea99] are our main sources of inspiration and their ideas are informally characterized (§2.4).

2.1 Security Protocols

We view *protocols* as specifications for the format and relative timing of messages exchanged between processes in distributed systems. In open systems where the specifications concern *secure* (in some sense) exchanges we call them *security protocols*. Security protocols usually employ *cryptography* in order to achieve their security goals and are, therefore, often referred to as *cryptographic protocols*.

Security protocols are often abstractly described in a standard notation referred to as *Alice-and-Bob notation*. A description in this notation contains one line for each step in the protocol. Each line states the step number, the sender, the intended receiver, and the format of the message.

$$
\begin{aligned}
&1.\ A \rightarrow B : M_1 \\
&2.\ B \rightarrow S : M_2 \\
&3.\ S \rightarrow B : M_3 \\
&4.\ B \rightarrow A : M_4 \\
&\quad\vdots
\end{aligned}
$$

A = Alice, B = Bob, S = Server

Figure 2.1: Alice and Bob notation for security protocols.

Every line is an instruction telling one process to send a message and another process to receive it. The line numbering indicates a causal and, thus, temporal ordering of these instructions. The first line in figure 2.1 instructs A to send message M_1 to B. If B receives message M_1 line 2 instructs him to send the message M_2 to S, and so forth.

While the term 'Alice-and-Bob notation' clearly reflects the fact that protocols enable communication between two entities, in our case processes, security protocols may sometimes involve one or more trusted third parties in order to meet security demands. As most security protocols are in fact cryptographic protocols the trusted third party is usually responsible for *session key distribution*.

A specification implicitly prescribes a certain assignment of roles. The *initiator* is the entity that requests communication (A in figure 2.1). The *responder* is the entity targeted for communication (B in figure 2.1). A

trusted third-party we call a *server* (S in figure 2.1). The roles are assigned when the first line of the protocol is executed and faithful execution require that they remain fixed throughout.

The message specifications M_i must state the type and ordering of contents. We adopt the notation proposed by Paulson [Pau98] which enclose messages in fat braces, but omits outermost braces.

1. $A \to B:\ \ \alpha^{X_A}$
2. $B \to A:\ \ \alpha^{X_B},\ \{\{\alpha^{X_B},\ \alpha^{X_A}\}_{S_B}\}_K$
3. $A \to B:\ \ \{\{\alpha^{X_A},\ \alpha^{X_B}\}_{S_A}\}_K$

Figure 2.2: The Station-to-Station protocol by Diffie, Wiener, and Van Oorschot

The Station-to-Station protocol [DvOW92] as shown in figure 2.2 is a nice example of the full protocol notation. The protocol is for authenticated key exchange. It enables two parties to agree on a shared secret (an integer valued key) by the exchange of public information. In the figure α is a publicly known integer, which is a primitive root of a publicly known prime q, and all arithmetic is assumed to take place *modulo* q. The exponents X_A and X_B are secret integers known only to A and B respectively. The exchange allows A and B to compute the same secret integer key, $K = \alpha^{X_A X_B}$, which is subsequently used for authentication.

The first line instructs A to send the integer α^{X_A} to B. The second line then instructs B to reply with the integer α^{X_B} (to complete the shared key) concatenated with the concatenation of the integers α^{X_B} and α^{X_A} signed with B's signing (private) key S_B and encrypted with the shared key K (to authenticate himself). The third line of the protocol instructs A to authenticate himself similarly and thereby complete the protocol.

2.2 Peer Networks

We consider modern computer networks, such as the Internet, the natural environments for security protocols. In order to achieve a useful abstraction we say that such a network consists of a transport layer and a number of *peer processes*. These peer processes are then the communicating entities of the Alice-and-Bob specifications.

2.2.1 The Transport Layer

The transport layer comprises some wiring that physically connects the peers, and one or more low level protocols that take care of basic packet distribution issues. This corresponds to the abstraction provided by layer 4 in the OSI 7-layer model [Bri00].

In order to say something meaningful about the capabilities of the network peers, it is necessary to make certain generalized assumptions about the facilities of the transport layer.

- We think of the transport layer communication as packets. Each protocol message is sent via the network in a packet.
- Each peer has a unique name in the network. This name is controlled by software and usually reflects who owns the peer process.
- Each peer has a unique location in the network at the time of execution. This location is controlled by hardware and usually reflects what machine the peer runs on.
- The transport layer somehow maintains information that binds the name of each peer to a unique location in the network[1].
- All packets send in the network are marked with the name and location of the sender and the name and location of the intended receiver.
- A packet exists in the network and the transport layer attempts to deliver it until someone confirms to have received it.

These assumptions naturally suggest a packet format and a very limited number of peer capabilities.

$$(\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$$

n_A	name of A
l_A	location of A

Figure 2.3: Format of a packet P

Peers are able to send, receive, and confirm the reception of packets in the network by interacting with the transport layer. When a peer enforces one of these capabilities an event is caused in the transport layer.

In fact the confirmation of message reception is normally taken care of by the transport layer. If all participants execute protocols faithfully recep-

[1] As we shall later see agents have no simple means of verifying this binding

$\mathsf{Snd}_A\ P$	A sends P
$\mathsf{Rcv}_A\ P$	A receives P
$\mathsf{Blk}_A\ P$	A blocks P

Figure 2.4: Events that a peer A may cause given a packet P.

tion and confirmation can be regarded as one unified event. Some attacks, however, rely on the attacker being able to cause each of the events individually. We therefore consider reception and blocking (confirming) two distinct events and make them both available to processes with the proviso that legal participants use them as a compound.

2.2.2 Peer Processes

As proposed by Abadi and Needham [AN96] we choose to think of the peers as processes running just above the transport layer. There are several reasons for making this choice.

Bella points out [Bel00] that the guarantees given by the protocol analysis are valid only for processes at this level because malicious break-ins may occur at any level of the workstation architecture.

Also we plan to consider availability issues via a notion of time. Availability has not been researched very much, but Meadows [Mea99] has shown how it may be fruitful to think of peers as processes and add detail regarding their execution to the model.

Friendly Agents

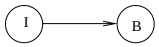
We use the term *friendly agent* about any peer on the network that faithfully (according to protocol) either initiates or responds to communication with another peer. We base our notion of friendly agents on a number of assumptions.

- Friendly agents call themselves by their proper name only.
- Friendly agents always honestly fill in the address information in the packages they send.
- Friendly agents only receive messages actually addressed to them.
- Friendly agents always confirm the reception of messages.

- Friendly agents execute protocols faithfully assuming the role of either initiator or responder depending on circumstances.
- Friendly agents may interleave any number of protocol executions.

When protocols rely on a trusted third party in order to achieve the security goals each friendly agent is usually required to share a long-term cryptographic secret with the third party. In this case we make an additional assumption.

- Friendly agents may be compromised by the loss of a long-term key.



$$\mathsf{Snd}_A\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$$
$$\mathsf{Rcv}_B\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$$
$$\mathsf{Blk}_B\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$$

Figure 2.5: Normal Flow

The Server

As mentioned there are sometimes peers that are assigned the roles of *servers*. When more than one peer has such a role we distinguish between different server types. The KerberosIV protocol for example has both an authentication server and a ticket granting server.

Most of the assumptions that apply to friendly agents also apply to servers. There are a couple of exceptions, however.

- A trusted third party will always assume the appropriate server role in protocol executions.
- A server cannot be compromised through the loss of a long-term secret.

Friendly agents rely on their long-term secrets in order to obtain safe session keys, while servers have no use of session keys. Therefore the loss of a long-term secret compromises the corresponding friendly agent but not the server.

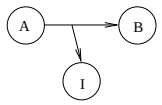
The Spy

The entire concept of security protocols is based on the idea that malicious peers may exist in the network. Such a malicious peer, a *spy*, may have a variety of intentions and may pursue them by many means.

It is easiest to think of the spy as a single entity whose intention it is to prevent protocol executions from achieving security goals. The spy will use any means available to him and we will make **no** behavioral assumptions about the spy. When Alice-and-Bob notation is used to describe an attack the spy is commonly referred to as the *intruder* and is denoted by an I.

Security Attacks It is natural to try and categorize and abstractly describe the different types of attack that may occur in our network model. Stallings [Sta99] makes one such categorization, which turns out to fit well in our setup.

Passive Attacks is a category of attacks that involves no sending or blocking events caused by the spy. The spy is thus a passive listener that may possibly *intercept* all of the network traffic in order to obtain useful information. In figure 2.6 interception occurs in the second line where the spy receives a packet meant for Bob.



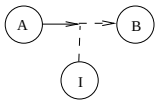
$$\mathsf{Snd}_A\ (\!|(n_A, l_A) \to (n_B, l_B) : M|\!)$$
$$\mathsf{Rcv}_I\ (\!|(n_A, l_A) \to (n_B, l_B) : M|\!)$$
$$\mathsf{Rcv}_B\ (\!|(n_A, l_A) \to (n_B, l_B) : M|\!)$$
$$\mathsf{Blk}_B\ (\!|(n_A, l_A) \to (n_B, l_B) : M|\!)$$

Figure 2.6: Message Interception

The release of secret message contents would be the most serious implication of a passive attack. Information revealed from traffic analysis, however, may also have dangerous applications as changes in traffic flow may reveal significant information to the trained eye.

Active Attacks is a category containing all attacks that involve sending or blocking events caused by the spy.

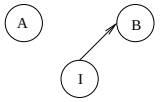
The types of possible active attacks range from very simple to quite sophisticated. The blocking ability alone allows the intruder to *interrupt* messages meant for other participants. If the intruder is clever enough to fill in the address information of packets with fake information he can create messages (*message fabrication*) and send them to other participants while pretending to be someone else (*participant spoofing*). Such insertions



$\mathsf{Snd}_A\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Blk}_I\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$

Figure 2.7: Message Interruption

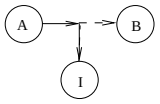
of fraudulent messages into the message stream is also known as *masquerades*.



$\mathsf{Snd}_I(\!|(n_A, l_I) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Rcv}_B(\!|(n_A, l_I) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Blk}_B(\!|(n_A, l_I) \rightarrow (n_B, l_B) : M|\!)$

Figure 2.8: Masquerades

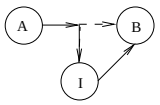
If combining the receiving and blocking abilities the intruder is able to simply *steal packets* intended for other participants.



$\mathsf{Snd}_A\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Rcv}_I\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Blk}_I\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$

Figure 2.9: Message Theft

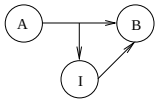
If combining all abilities the intruder may modify the messages of other agents. This is commonly referred to as *Man-in-the-middle* attacks.



$\mathsf{Snd}_A\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Rcv}_I\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Blk}_I\ (\!|(n_A, l_A) \rightarrow (n_B, l_B) : M|\!)$
$\mathsf{Snd}_I(\!|(n_A, l_I) \rightarrow (n_B, l_B) : M'|\!)$

Figure 2.10: Message Modification

A similar situation is *message reuse*, where the intruder makes a replay of earlier intercepted messages in order to have an unauthorized protocol run with another participant. This is referred to a *replay attacks*.



$$
\begin{aligned}
&\mathsf{Snd}_A\ (\!|(n_A, l_A) \to (n_B, l_B) : M|\!)\\
&\mathsf{Rcv}_B\ (\!|(n_A, l_A) \to (n_B, l_B) : M|\!)\\
&\mathsf{Blk}_B\ (\!|(n_A, l_A) \to (n_B, l_B) : M|\!)\\
&\vdots\\
&\mathsf{Snd}_I(\!|(n_A, l_I) \to (n_B, l_B) : M|\!)
\end{aligned}
$$

Figure 2.11: Message Reuse

2.3 Security Goals

In order to adequately describe the notion of security we must break it into a number of aspects or constituting notions. Each of these aspects constitutes a security goal of the protocol. If a protocol achieves all of its security goals we consider it secure.

As pointed out by Bella [Bel00] establishing a suitable list of relevant aspects for the domain is ongoing work. There are some, however, the importance of which is universally agreed on and that have already received a great deal of attention.

2.3.1 Goals Unrelated to Time

Thus, this section will deal with aspects that are largely not dependent on time. First we introduce a set of notions that are more or less classic and then a number of newer, less established, notions will be mentioned briefly.

Confidentiality

The *confidentiality* of the information exchanged between two peers is probably the most natural goal of any security protocol. The goal can be pursued at a number of levels. One extreme is to use strong encryption over the entire information stream. The other extreme is to protect only the critical fields inside the actual messages. Obviously confidentiality measures may be implemented at any level between the two extremes. We will, however, assume that encryption takes place on the message level and not the packet level.

As mentioned in section 2.2.2 about passive attacks, confidentiality is breached primarily by means of analysis of intercepted messages.

Certain attacks, i.e. Man-in-the-middle attacks, aim to gain access to confidential knowledge by impersonating a trusted participant. We do not consider this a breach of confidentiality but of *authenticity* and *integrity* as described in section 2.3.1.

Also, one could argue that, since some security protocols rely on timestamps in order to achieve confidentiality, the goal depends on time. This is only indirectly so. We consider the time dependent aspects of this in section 2.3.2 on *fail-stop protocols*.

Integrity/Authenticity

If the protocol guarantees that a message is not altered while in transit it provides *integrity* for this message. If the protocol guarantees that a message does in fact originate with the agent from which it appears to originate it provides *authenticity* for this message.

As an extension to a conversation with Dieter Gollmann [Gol99] Bella argues convincingly (by *inter-provability*) that these two notions are equivalent, at least in the inductive setting.

In [Sta99] Stallings argues that integrity can be enforced at any level in between and including complete stream protection and protection of individual fields inside messages. We take the view that integrity and authenticity deal with the individual messages and their fields.

Authenticity over the entire protocol execution then becomes *authentication* as described in section 2.3.1.

Authentication

Authentication is concerned with establishing the authenticity of an entire protocol execution. Therefore authentication entails both the initial verification of participant identities and the subsequent protection against interference in the form of masquerades and man-in-the-middle attacks.

There is some confusion regarding the actual meaning of the notion of authentication. In other words, when a protocol claims to achieve authentication it is unclear what is actually achieved. Bella [Bel00] points to a taxonomy due to Lowe [Low97]. If an initiator A completes a protocol with a responder B then one of a number of guarantees may be given by the protocol.

1. **Aliveness** of B guaranteeing that B has been running the protocol.
2. **Weak agreement** of B with A guaranteeing 1 and that B ran the protocol with A.
3. **Non-injective agreement** of B with A guaranteeing 2 and that A and B agreed on the set $\boldsymbol{H}$ of message components.
4. **Injective agreement** of B with A guaranteeing 3 and that B did not respond more than once on the session with A.

Non-Repudiation

If a protocol prevents either the sender or receiver from denying a transmitted message *non-repudiation* is achieved. It allows the sender to prove that the message was in fact received by the alleged receiver, the receiver to prove that the message was actually send by the alleged sender, or both.

Others

There are a number of relatively new notions that are becoming important, but have not yet received much attention from researchers.

Delegation The *delegation* of *rights* or *responsibilities* is achieved if the protocol guarantees that recipients of delegation in fact have received certain credentials, such as the obligation to fire an employee or the right to use the photocopier.

Anonymity In the field of E-commerce *anonymity* is useful. This is achieved if the protocol guarantees that agents can complete executions without their identities being disclosed to anyone.

Accountability Also financial transactions may benefit from *accountability*. The protocol achieves this if at the end of an execution it can be established that certain events have taken place.

2.3.2 Goals Related to Time

As opposed to the above described goals, that may or may not be vaguely related to time, there exist at least a couple of goals that rely heavily on time.

Availability

Probably the single-most important notion that can be seen as directly dependent on time is *availability*. Availability is achieved if the protocol initiator can be assured that certain services will be granted.

The loss of availability can be caused by a number of events. Some of these, such as regular breakdowns of service providers, power failures, line failures, even malicious break-ins, are outside the scope of protocol security.

When it comes to denial-of-service (DoS) attacks, however, the protocol is critical in preventing the loss of availability. This is why this aspect can also be referred to as *non-denial of service* [Bel00].

DoS attacks aim to exhaust the resources of the service provider thereby preventing other peers from receiving the service. The following account is due to Meadows [Mea99].

The simplest type of DoS attack is probably the SYN [2] attack, which is generally only effective against protocols that do nothing or very little to prevent DoS. The attack aims to exhaust the service providers capability of maintaining state information of protocol executions. The intruder simply initiates a large number of protocol executions, which he subsequently abandons. The service provider is not able to free up the resources tied to a particular protocol execution until a suitable time-out period has expired. This allows attackers to carry out DoS attacks with minimal effort.

In order to prevent DoS, security protocols force the initiator to authenticate himself before the service provider initiates protocol execution. This makes it substantially more difficult to mount a SYN attack. Unfortunately the cryptographic operations involved are usually computationally expensive. This makes it possible to mount another type of attack aiming to exhaust the computational resources of the service provider.

The complete prevention of DoS is probably impossible. The aim of DoS is to force the service provider into wasting all available computation time. Thus the natural aim of security protocols must be to minimize the computation time required to reject fake messages thereby making the cost of mounting a successful DoS attack prohibitively high.

Fail-Stop

A security protocol is *fail-stop* if it automatically halts in response to any active attack that interferes with protocol execution.

[2] The term stems from an attack on the TCP protocol that abuses the SYN (synchronize/start) message in order to make the server instantiate a protocol run, which is subsequently aborted by the attacker.

The notion was introduced by Gong and Syverson [GS95] as part of an attempt to establish sound design practices for protocols and simplify the protocol security analysis. They have the following explanation:

> Using Lamport's definition of causality [Lam78], we can organize the messages of a protocol into an acyclic directed graph where each arc represents a message and each directed path represents a sequence of messages. In a fail-stop protocol, if a message actually sent is in any way inconsistent with the protocol specification, then all the messages that are behind this altered message on some path in the graph (that is, they are causally after the altered message) will not be sent.

The authors argue convincingly that active attacks can cause no harm to fail-stop protocols other than cause early termination.

This claim rests on two assumptions:

- The integrity of messages is protected by some (cryptographic) mechanism.
- Active attacks are ones involving the insertion of fraudulent messages, obtained by fabrication or modification, into the message stream.

It is clear that an intruder will learn nothing from such an attack that could not be learned from passive eavesdropping. Thus, the benefit of fail-stop protocols is that analysis is reduced to that of passive attacks only, which includes reasoning about authentication and *secrecy* (confidentiality).

Gong and Syverson state that protocols are fail-stop if they conform to one of the known specifications of fail-stop protocols. Basically a number of points should be fulfilled.

1. The contents of each message suffices to reveal forgeries.
2. The integrity of each message is guaranteed.
3. An honest process follows the protocol and ignores all unexpected messages.
4. A process halts any protocol run in which an expected message does not arrive within a specified timeout period.

Item 1 and 2 are interpretations of the authors original text and consider the contents of the messages. Item 3 and 4 consider the behavior of peer processes and show how the notion is dependent on time.

2.4 Important Approaches

Various researchers have put an enormous amount of work into the development of formal approaches to the analysis of security protocols, especially during the last two decades. Obviously, it does not make sense to carry out new work within the field without drawing upon the results achieved by at least some of these contributors.

So, while a number of previous results have influenced the present work to some small degree, there are at least two theoretical approaches that have directly inspired and influenced the ideas described in this thesis. These will be described shortly in the following.

2.4.1 Inductive Analysis

Paulson has developed an inductive approach to protocol analysis presented thoroughly in [Pau98] and less so in [Pau99], which has later been extended and matured by Bella in [Bel00] in order to treat protocols that rely on timestamps. Specifically the Kerberos Authentication System has received a great deal of attention from Bella in [BP98b], [BP98a], and [BP97].

The inductive approach is based on *operational semantics* and is inspired by work of Schneider et al. The foundation is a concrete notion of *network events*. Protocols are inductively defined as the set of all possible *traces* of such events. The actions of *honest agents* are incorporated into the protocol model. The actions of the *intruder* are specified separately and are based on the Dolev-Yao assumption of a strong attacker and strong encryption [DY83], which entails:

- The attacker intercepts every message
- The attacker can cause a party to apply any of its operators at any time (by starting new sessions)
- The attacker can apply any operator except other's decryption

The resulting structure can be seen as a tree rooted in the empty trace and the inductive definition then defines a causal relationship between events similar to that mentioned by Gong and Syverson.

The formalization of the method is carried out in higher order logic and subsequently encoded in the interactive proof assistant Isabelle/HOL [3]. The

[3] In which the present work is also encoded

notions of *agents* (peers) and *messages* are formalized as *heterogeneous sets* through the use of *datatype* definitions as known from the functional programming language ML[Pau91]. The connection between agents, messages and the network is captured by the similarly defined heterogeneous set of *events*.

Secrecy, authentication, and other goals are then proved by induction on traces as guarantees are derived from each event in the trace. The necessary operators are defined inductively over sets, such as the growing pool of intruder knowledge. A number of derived laws for the operators and appropriate rewrite rules for symbolic evaluation are defined.

Capabilities

Due to the hostile nature of protocol environments no perfect method for analysis has yet been devised. The inductive approach gives quite strong (but not perfect [Pau99]) assurances but obtaining them is costly in terms of expert time needed to conduct the proofs.

The most notable shortcoming of inductive analysis is that it is no good at analyzing availability issues [Bel00]. There are lots of other properties that can be analyses, though. The following list of such properties is due to Bella:

- *Reliability* properties, e.g. *possibility*, which shows that the protocol can be completed by honest agents, can be proved in order to highlight any discrepancies between the model and the real system.
- *Regularity* results concern the implications of a particular item `X` occurring in the trace as a message component. If, for example, the shared key of an agent occurs in the traffic it may imply that the agent is compromised.
- *Integrity/authenticity* results can be shown.
- *Unicity* of messages may be shown to be an implication of the presence of certain message components, such as nonces, signatures etc.
- *Confidentiality* results are proven by induction and could, for example, show that the compromise of e.g. a session key does not jeopardize the content of other unrelated communications.
- *Authentication* guarantees up to weak agreement [4] of peers can be shown to exist by induction.

[4] Bella argues in [Bel00] that the approach can be modified up to but not beyond non-injective agreement.

- *Key distribution* requirements, i.e. when two peers completing a session gain evidence of sharing a session key with each other, are proven by an authentication argument.
- *Minimal trust* results concern the assumptions that are indispensable for the formal reasoning but cannot be verified by any agent besides the spy, e.g. that a particular agent is uncompromised,
- *Goal Availability* results concern the extent to which agents are able to verify the assumptions on which the guarantees given to them are based.

2.4.2 Cost Based Fail-Stop Analysis of Denial of Service

Meadows [Mea99] proposes an approach to the analysis of availability issues related to DoS resistance.

In her view, as in ours, the network is an open distributed system with a number of peer processes executing protocols as concrete programs. As in all distributed systems, the processes perform a number of *internal actions* that are not externally visible and cause *external events* (in the form of message exchanges) to synchronize their executions and exchange data.

To describe protocols Meadows employ an *annotated Alice-and-Bob* notation

$$i \quad A \rightarrow B: \quad \mathrm{T} \parallel M \parallel \mathrm{O}$$

where $\mathrm{T} \in \boldsymbol{T}^{\star}$ is a sequence of pre-message actions performed by the sender, M indicates the transmission of a message from the sender to the receiver, and $\mathrm{O} \in \boldsymbol{O}^{\star}$ is a sequence of post-message internal actions performed by the receiver. The M is considered as containing two external events: $send_A(M)$, the sending of the message M by A[5], and $rec_B(M)$, the reception of the message by B[6].

Internal actions take time, especially because of the cryptographic operations required to protect the integrity and authenticity of messages. Furthermore, it is costly to initialize protocol runs. A machine can only host a limited number of concurrent protocols, but once initialized a protocol stack exists until either the protocol finishes, a fraud is detected, or a timeout period expires.

[5] Equivalent to $\mathsf{Snd}_A\ P$ in our view.

[6] Equivalent to the compound event of $\mathsf{Rcv}_A\ P$ and $\mathsf{Blk}_A\ P$ in our view.

DoS attacks may abuse any of these weak points. In order to avoid exhaustion of the available protocol stacks the protocol must implement suitable checks that must be passed before a protocol stack is initialized. If, however, these checks are expensive, computational resources may be exhausted instead.

Meadows proposes that this may be solved by constructing the protocol to use relatively simple checks in the first steps, while proceeding to more complex and expensive checks as the protocol progresses successfully. The construction of such protocols should be guided by a predetermined tolerance relation that determines how large a cost one is wiling to accept in order to provide security against an intruder of a given strength.

The costs of actions and steps are defined in terms of *cost functions*, the concrete formulations of which depend on the type of resources spent. Intruders are characterized by an *intruder capability* $g \in \boldsymbol{G}$ expressing the intrusive power of the set of actions available to them.

Given the tolerance relation and the cost function an attacker capability function, $\Gamma : \boldsymbol{action} \rightarrow \boldsymbol{G}$ from the set of actions, $\boldsymbol{action} = \boldsymbol{T} \cup \boldsymbol{O}$, to the set of intruder capabilities, $\boldsymbol{G}$, can be defined. A protocol is then *fail-stop with respect to* Γ if

> For each action $a \in \boldsymbol{action}$, if an intruder of capability $\Gamma(a)$ interferes with any message *desirably-causally-preceding* a, then neither a nor any action/event *desirably after* a will take place.

where the relation *desirably-causally-preceding* is a derivative of Gong and Syverson's application of Lamport's *causally-before*, that refers to the desired behavior of a correctly executed protocol, rather than the actual observed behavior. The idea that one event e_1 is the *cause* of another e_2 is expressed $e_1 \prec e_2$.

Capabilities

The proposal made by Meadows is informal and she argues that the outlined setup could be incorporated into a number of existing approaches.

Given such a setup and a suitably designed protocol the purpose of verification would be to show that the protocol is indeed fail-stop with respect to the given intruder capability function. We would then know that:

- The protocol is *fail-stop* as defined by Gong and Syverson and rules out active attacks.
- Certain guarantees with respect to *availability* in the face of DoS attacks are given, i.e. the upper bounds on the resources needed to cope with DoS attacks of different strengths are known.

CHAPTER 3

Untimed Theory of Global Traces

> *We need a foundation that allows us to model and examine the untimed aspects of security protocols. We achieve this by adopting Paulson's inductive approach to analysis and then extending it to be more suitable for a cost-based approach akin to Meadows'.*

We choose Paulson's inductive approach as the basis of our own work. There are a couple of reasons for this. Paulson's idea of deriving guarantees from each message that occurs on the network seems to be a good one. Modeling the protocol inductively as a set of possible traces also apears to be sensible for this purpose.

There are some things we want to do different, though, in order to make the foundation more suitable for the cost-based fail-stop analysis proposed by Meadows.

We need to be able to recognize active attacks, which means that we need more address information than Paulson does. This is reflected by the notion of *packets* (§2.2.1).

In order to adequately argue about Meadow's *attacker capability function* (§2.4.2) we need to explicitly specify attacker capabilities with a finer granularity than Paulson does (§2.4.1)[1].

These two demands imply that we need a theory of events different from Paulson's. In fact, our untimed theory of *global traces* turns out somewhat different from that of his inductive method.

This chapter describes the developed untimed theory of global traces:

[1] He simply assumes the strongest possible attacker.

- From the inductive approach [Pau98] we get a suitable theory of *agents*, *messages*, and *keys* (§3.1).
- The address information required for identifying active attacks is carried by *packets* (§3.2).
- The desired granularity in description of intruder capabilities is reflected in the choice of *events*. This choice determines the way *knowledge* is gained from a sequence of messages and how *freshness* of a message is checked (§3.3).
- Regardless of protocol and network type there is a minimal set of causal restriction that apply to all *traces* (§3.4).
- Faithful executions of actual protocols are object to much more severe causal restrictions than traces in general. This gives rise to ideal protocol descriptions that serve as references when identifying active attacks (§3.5).
- Finally, *schematic representations* of protocols help us define the notion of a *successful active attack* (§3.6).

The Isabelle/HOL encoding of the following theory is enclosed in appendix A.3.

3.1 Agents, Keys, and Messages

Paulson's basic theory of agents, keys, and messages is perfectly consistent with our view of the domain. So we adopt it and use it as the basis of our work. All theory described in this section is due to Paulson [Pau98][Pau99], so readers thoroughly familiar with his work may skip ahead to section 3.2.

3.1.1 Agents

As we describe in section 2.2 Paulson deals with systems consisting of a number of unique *agents*. He assumes that there is only one *server*[2] and only one *spy* while there is an infinite number of *friendly agents*. Hence, three disjoint sets are given.

Friend	a set of friendly agents (§2.2.2)
Server	a singleton set $\{\star\}$ (§2.2.2)
Spy	a singleton set $\{\star\}$ (§2.2.2)

[2]The work on Kerberos [BP98a] is the only known case where this assumption is too weak.

An agent is either a friendly agent, a server, or a spy.

$$\boldsymbol{agent} = \boldsymbol{Friend} \cup \boldsymbol{Server} \cup \boldsymbol{Spy}$$

For each of these sets there is a named injection identifying its members uniquely. The ranges of these are, of course, disjoint.

$$\begin{aligned} \mathsf{Friend} &: \mathbb{N} \hookrightarrow \boldsymbol{agent} \\ \mathsf{Server} &: \boldsymbol{agent} \\ \mathsf{Spy} &: \boldsymbol{agent} \end{aligned}$$

3.1.2 Cryptographic Keys

Paulson assumes an infinite set, ***key***, of *cryptographic keys*. These keys may be used to encrypt or decrypt messages (§3.1.3). Every key K has an inverse K^{-1} that reverses its cryptographic effect. This is captured by the *automorphic* function:

$$\mathsf{invKey} : \boldsymbol{key} \cong \boldsymbol{key}$$

which is its own *inverse function* and thus subject to

$$(\mathsf{invKey} \circ \mathsf{invKey})\, K = K$$

For a subset of *symmetric keys* invKey is *idempotent*:

$$\boldsymbol{symKeys} = \{K : \boldsymbol{key} \mid \mathsf{invKey}\, K = K\}$$

For reasons of convenience keys and natural numbers are considered identical:

$$\boldsymbol{key} = \mathbb{N}$$

3.1.3 Messages

In Paulson's setup [Pau98], as in ours (§2.1), the agents are transmitting and receiving *messages*. He defines seven disjoint sets:

agent	The described set of agent names
Number	A set of guessable nonces
Nonce	A set of unguessable nonces
key	The described set of cryptographic keys
MPair	A set of pairs of messages
Hash	A set of message digests
Crypt	A set of encrypted messages

A message is then either an agent, a number, a nonce, a key, a message pair, a message digest, or an encrypted message.

$$\boldsymbol{msg} = \boldsymbol{agent} \cup \boldsymbol{Number} \cup \boldsymbol{Nonce} \cup \boldsymbol{key} \cup \boldsymbol{MPair} \cup \boldsymbol{Hash} \cup \boldsymbol{Crypt}$$

Each kind of message will briefly be named and described in the following.

The agent names is a set of messages corresponding directly to the set ***agent*** (§3.1.1).

$$\mathsf{Agent} : \boldsymbol{agent} \hookrightarrow \boldsymbol{msg}$$

The guessable numbers are *numbers*, such as timestamps or sequence numbers, that are fairly predictable.

$$\mathsf{Number} : \mathbb{N} \hookrightarrow \boldsymbol{msg}$$

The nonces are (freshness) identifiers, such as forty byte random strings, that are NOT guessable. The *nonces* are important for *integrity* and *authenticity.*

$$\mathsf{Nonce} : \mathbb{N} \hookrightarrow \boldsymbol{msg}$$

The cryptographic keys is a set of messages corresponding to the set ***key***. They are important for *confidentiality.*

$$\mathsf{Key} : \boldsymbol{key} \hookrightarrow \boldsymbol{msg}$$

The message pairs is the set of *compound messages.* The theory of *message pairs* clarifies the relationship between the messages informally discussed in §2.1 and Paulson's formal construction discussed here.

$$\mathsf{MPair} : \boldsymbol{msg} \hookrightarrow (\boldsymbol{msg} \hookrightarrow \boldsymbol{msg})$$

Nested applications of MPair are abbreviated as tuples.

$$\{\!|X, Y|\!\} \equiv \mathsf{MPair}\ X\ Y$$
$$\{\!|X, Y, Z|\!\} \equiv \{\!|X, \{\!|Y, Z|\!\}|\!\}$$

The message digests are messages created by application of a suitable one-way function to a message computing a unique fingerprint for it. These *message hashes* are important for integrity and authenticity.

$$\mathsf{Hash} : \boldsymbol{msg} \hookrightarrow \boldsymbol{msg}$$

The actual application of the one-way function often depends on a *shared secret* $X : \boldsymbol{msg}$ which can be used to seed the pseudo-random one-way function.

$$\mathsf{HPair}\ X\ Y \mathrel{\hat{=}} \{\!|\mathsf{Hash}\{\!|X, Y|\!\}, Y|\!\} : \boldsymbol{msg} \hookrightarrow (\boldsymbol{msg} \hookrightarrow \boldsymbol{msg})$$

In order to emphasize dependence on the seed an illustrative abbreviation of HPair is defined.

$$\mathsf{Hash}[X]\ Y \equiv \mathsf{HPair}\ X\ Y$$

The encrypted messages are messages hidden under the application of a cryptographic key. We assume perfect encryption.

$$\mathsf{Crypt} : \boldsymbol{key} \hookrightarrow (\boldsymbol{msg} \hookrightarrow \boldsymbol{msg})$$

The assumption of perfect encryption gives the function a desirable feature, which justifies the double injection.

$$\mathsf{Crypt}\ K\ X = \mathsf{Crypt}\ K'\ X' \Longrightarrow K = K' \ \wedge\ X = X'$$

Alteration of encrypted messages requires decryption with the appropriate key K^{-1}. An important note is that the encrypting key is not considered a part of the resulting message.

Operators Dealing With Sets of Messages

Paulson [Pau98] states most of his verification goals with the help of three operators defined on messages. These will be explained in the following.

The operator parts names all constituting parts of all messages in a given set, excluding only keys used for encryption but not explicitly included in any message. Thus, it is a function from sets of messages to sets of messages.

$$\mathsf{parts} : \mathcal{P}(\boldsymbol{msg}) \to \mathcal{P}(\boldsymbol{msg})$$

If $\boldsymbol{H} \in \mathcal{P}(\boldsymbol{msg})$ then $\mathsf{parts}\ \boldsymbol{H}$ is formally defined as the least set closed under projection and decryption, i.e. the closure of the rules

$$\frac{X \in \boldsymbol{H}}{X \in \mathsf{parts}\ \boldsymbol{H}} \qquad \frac{\mathsf{Crypt}\ KX \in \mathsf{parts}\ \boldsymbol{H}}{X \in \mathsf{parts}\ \boldsymbol{H}}$$

$$\frac{\{\!|X,Y|\!\} \in \mathsf{parts}\ \boldsymbol{H}}{X \in \mathsf{parts}\ \boldsymbol{H}} \qquad \frac{\{\!|X,Y|\!\} \in \mathsf{parts}\ \boldsymbol{H}}{Y \in \mathsf{parts}\ \boldsymbol{H}}$$

Paulson has shown this operator to be monotonous

$$\boldsymbol{G} \subseteq \boldsymbol{H} \Longrightarrow \mathsf{parts}\ \boldsymbol{G} \subseteq \mathsf{parts}\ \boldsymbol{H}$$

and idempotent

$$\mathsf{parts}(\mathsf{parts}\ \boldsymbol{H}) = \mathsf{parts}\ \boldsymbol{H}$$

The operator analz names all visible parts of all messages in a given set, when decryption is only allowed for known keys. Thus, it is a function from sets of messages to sets of messages.

$$\mathsf{analz} : \mathcal{P}(\boldsymbol{msg}) \to \mathcal{P}(\boldsymbol{msg})$$

If $\boldsymbol{H} \in \mathcal{P}(\boldsymbol{msg})$ then $\mathsf{analz}\ \boldsymbol{H}$ is formally defined as the least set closed under projection and decryption with known keys, i.e. the closure of the rules

$$\frac{X \in \boldsymbol{H}}{X \in \mathsf{analz}\ \boldsymbol{H}} \qquad \frac{\mathsf{Crypt}\ KX \in \mathsf{analz}\ \boldsymbol{H} \quad K^{-1} \in \mathsf{analz}\ \boldsymbol{H}}{X \in \mathsf{analz}\ \boldsymbol{H}}$$

$$\frac{\{\!|X,Y|\!\} \in \mathsf{analz}\ \boldsymbol{H}}{X \in \mathsf{analz}\ \boldsymbol{H}} \qquad \frac{\{\!|X,Y|\!\} \in \mathsf{analz}\ \boldsymbol{H}}{Y \in \mathsf{analz}\ \boldsymbol{H}}$$

Paulson has shown this operator to be monotonous

$$\boldsymbol{G} \subseteq \boldsymbol{H} \Longrightarrow \mathsf{analz}\ \boldsymbol{G} \subseteq \mathsf{analz}\ \boldsymbol{H}$$

and idempotent

$$\mathsf{analz}(\mathsf{analz}\ \boldsymbol{H}) = \mathsf{analz}\ \boldsymbol{H}$$

The operator synth names all messages that can be constructed from the messages in a given set, when encryption is only possible under known keys. Thus, it is defined as a function from sets of messages to sets of messages:

$$\mathsf{synth} : \mathcal{P}(\boldsymbol{msg}) \to \mathcal{P}(\boldsymbol{msg})$$

If $\boldsymbol{H} \in \mathcal{P}(\boldsymbol{msg})$ then $\mathsf{synth}\ \boldsymbol{H}$ is formally defined as the least set that includes $\boldsymbol{H}$, agent names and guessable numbers, and is closed under pairing, hashing and encryption, i.e. the closure of the rules

$$\mathsf{Agent}\ A \in \mathsf{synth}\ \boldsymbol{H} \qquad \mathsf{Number}\ n \in \mathsf{synth}\ \boldsymbol{H}$$

$$\frac{X \in \boldsymbol{H}}{X \in \mathsf{synth}\ \boldsymbol{H}} \qquad \frac{X \in \mathsf{synth}\ \boldsymbol{H}}{\mathsf{Hash}\ X \in \mathsf{synth}\ \boldsymbol{H}}$$

$$\frac{X \in \mathsf{synth}\ \boldsymbol{H} \quad Y \in \mathsf{synth}\ \boldsymbol{H}}{\{\!|X, Y|\!\} \in \mathsf{synth}\ \boldsymbol{H}} \qquad \frac{X \in \mathsf{synth}\ \boldsymbol{H} \quad K \in \boldsymbol{H}}{\mathsf{Crypt}\ KX \in \mathsf{synth}\ \boldsymbol{H}}$$

Paulson has shown this operator to be monotonous

$$\boldsymbol{G} \subseteq \boldsymbol{H} \Longrightarrow \mathsf{synth}\ \boldsymbol{G} \subseteq \mathsf{synth}\ \boldsymbol{H}$$

and idempotent

$$\mathsf{synth}(\mathsf{synth}\ \boldsymbol{H}) = \mathsf{synth}\ \boldsymbol{H}$$

3.2 The Network Packets

We have now introduced Paulson's notion of messages, which we will use without change. As opposed to Paulson, who bases the notion of *events* directly on those of agents and messages, we introduce the novel notion of *packets*. This notion is a formalization of the packets described in 2.2.1 and we employ it to enable a theoretical distinction between active and passive attacks.

The idea behind this use is that network peers can lie about their name but not about their location. The extra information gained is primarily of theoretical interest as it in practice is very hard, perhaps infeasible, for network peers to verify whether the reported name and location of an alleged sender or receiver matches. Still, the information does have some practical significance as the name and location together may serve as a return address.

3.2.1 Components of Packets

Packets have a number of components, which shall be explained in the following.

Names of Agents We assume that every agent has a unique *name* in the network. This means that the set $\boldsymbol{Nm}$ of names is isomorphic to the set of agents. We abuse this and simply define them to be identical.

$$\boldsymbol{Nm} = \boldsymbol{Agent}$$

Locations of Agents Likewise we assume that every agent has a unique *location* in the network. We use the identity again.

$$\boldsymbol{Loc} = \boldsymbol{Agent}$$

Address of Agents Packets contain the *address* of the sender and that of the receiver. An address consists of a name and a location.

$$\boldsymbol{Adr} = \boldsymbol{Name} \times \boldsymbol{Loc}$$

Judgments on Addresses are necessary because not all addresses are meaningful on the network. As mentioned (§2.2.1) the network expects the name and the location of the address to match. Because of the fact, that both names and locations are in fact agents, simple equality of agents is the key to these judgments.

If these components of a given address do not match it is a *fake address*.

$$\mathsf{fakeAdr}\ (n, l) \mathrel{\hat{=}} n \neq l : \ \boldsymbol{Adr} \to \boldsymbol{bool}$$

If they do match, however, we say that it is a *good address*.

$$\mathsf{goodAdr}\ a \mathrel{\hat{=}} \neg \mathsf{fakeAdr}\ a \ : \ \boldsymbol{Adr} \to \boldsymbol{bool}$$

Formal Packets Now - the set of packets is formally defined as a Cartesian product:

$$\boldsymbol{Pkt} = \boldsymbol{Adr} \times \boldsymbol{Adr} \times \boldsymbol{msg}$$

We enforce a more convenient and descriptive notation for packets by abbreviation.

$$(\!|s \ \rightarrow \ r : \ m|\!) \equiv \langle s, r, m \rangle \tag{3.1}$$

Here s denotes the address of the sender, r that of the receiver, and m denotes a message.

3.2.2 Projections of Packets

In order to mention the contents of packets by name we define a number of projections. The names and types should speak for themselves. The prefixed lowercase letters correspond to the fields illustrated in eq. (3.1).

$$\begin{aligned}
\mathsf{sAdrPkt}\ (\!|s \rightarrow r : m|\!) &\mathrel{\hat{=}} s : \boldsymbol{Pkt} \rightarrow \boldsymbol{Adr} \\
\mathsf{rAdrPkt}\ (\!|s \rightarrow r : m|\!) &\mathrel{\hat{=}} r : \boldsymbol{Pkt} \rightarrow \boldsymbol{Adr} \\
\mathsf{MsgPkt}\ (\!|s \rightarrow r : m|\!) &\mathrel{\hat{=}} m : \boldsymbol{Pkt} \rightarrow \boldsymbol{msg} \\
\mathsf{sNmPkt}\ (\!|(n,l) \rightarrow r : m|\!) &\mathrel{\hat{=}} n : \boldsymbol{Pkt} \rightarrow \boldsymbol{Nm} \\
\mathsf{sLocPkt}\ (\!|(n,l) \rightarrow r : m|\!) &\mathrel{\hat{=}} l : \boldsymbol{Pkt} \rightarrow \boldsymbol{Loc} \\
\mathsf{rNmPkt}\ (\!|s \rightarrow (n,l) : m|\!) &\mathrel{\hat{=}} n : \boldsymbol{Pkt} \rightarrow \boldsymbol{Nm} \\
\mathsf{rLocPkt}\ (\!|s \rightarrow (n,l) : m|\!) &\mathrel{\hat{=}} l : \boldsymbol{Pkt} \rightarrow \boldsymbol{Loc}
\end{aligned}$$

3.2.3 Judgments on Packets

As not all addresses are meaningful, some packets (those containing fake addresses) are also problematic. We say that a packet containing a fake address is a *fake packet.*

$$\mathsf{fakePkt}\ (\!|s \rightarrow r : m|\!) \mathrel{\hat{=}} \mathsf{fakeAdr}\ s \ \vee \ \mathsf{fakeAdr}\ r : \boldsymbol{Pkt} \rightarrow \boldsymbol{bool}$$

A packet not containing a fake address we consider a *good packet.*

$$\mathsf{goodPkt}\ p \mathrel{\hat{=}} \neg \mathsf{fakePkt}\ p : \boldsymbol{Pkt} \rightarrow \boldsymbol{bool}$$

3.3 Events, Knowledge, and Freshness

Paulson [Pau98] formalizes communication as network events and allows his agents to derive their knowledge from the events that they see. We preserve these basic ideas in our setup, but we completely replace Paulson's set of events (Sends, Notes) with the set described in section 2.2.1.

We have two reasons for doing this. Firstly, we need a notion that is based on packets. Secondly, we want to pursue a method similar to that outlined by Meadows [Mea01], which requires the events to be able to model attackers of varying strengths - from the weakest (normal peers) to the strongest (Dolev-Yao §2.4.1).

3.3.1 The Events of the Network

Whenever agents do things with the packets they interact with the transport layer as described in section 2.2.1. Interactions are formalized as a set of events that agents may cause. We assume that three disjoint sets are given:

$\boldsymbol{Snd}$	a set of sending events
$\boldsymbol{Rcv}$	a set of receiving events
$\boldsymbol{Blk}$	a set of blocking events

An event is either a sending event, a receiving event, or a blocking event.

$$\boldsymbol{event} = \boldsymbol{Snd} \cup \boldsymbol{Rcv} \cup \boldsymbol{Blk}$$

Thus, we have three named injections:

$$\begin{aligned} \mathsf{Snd} &: \boldsymbol{Nm} \hookrightarrow (\boldsymbol{Pkt} \hookrightarrow \boldsymbol{event}) \\ \mathsf{Rcv} &: \boldsymbol{Nm} \hookrightarrow (\boldsymbol{Pkt} \hookrightarrow \boldsymbol{event}) \\ \mathsf{Blk} &: \boldsymbol{Nm} \hookrightarrow (\boldsymbol{Pkt} \hookrightarrow \boldsymbol{event}) \end{aligned}$$

Judgments on Events

As for packets some events are more acceptable than others. Clearly, if the agent causing the event has filled out the address information faithfully his name should be in the appropriate fields inside the packet.

If this is not the case it is a *fake event*. We capture the essence of this in the function

$$\mathsf{fakeEvent} : \boldsymbol{event} \rightarrow \boldsymbol{bool}$$

which is defined through the following three cases

$$\begin{aligned}
&\mathsf{fakeEvent}\ (\mathsf{Snd}\ n\ (\!|(sn, sl) \rightarrow (rn, rl) : m|\!)) \mathrel{\hat{=}} \\
&\qquad \mathsf{fakePkt}\ (\!|(sn, sl) \rightarrow (rn, rl) : m|\!) \vee n \neq sn \\
&\mathsf{fakeEvent}\ (\mathsf{Rcv}\ n\ (\!|(sn, sl) \rightarrow (rn, rl) : m|\!)) \mathrel{\hat{=}} \\
&\qquad \mathsf{fakePkt}\ (\!|(sn, sl) \rightarrow (rn, rl) : m|\!)\ \vee\ n \neq rn \\
&\mathsf{fakeEvent}\ (\mathsf{Blk}\ n\ (\!|(sn, sl) \rightarrow (rn, rl) : m|\!)) \mathrel{\hat{=}} \\
&\qquad \mathsf{fakePkt}\ (\!|(sn, sl) \rightarrow (rn, rl) : m|\!)\ \vee\ n \neq rn
\end{aligned}$$

If, on the other hand, the information in the packet is correct it is a *good event*.

$$\mathsf{goodEvent}\ e \mathrel{\hat{=}} \neg \mathsf{fakeEvent}\ e$$

These judgments are novel developments and their significance is purely theoretical since peers have no way of deciding these issues in practice.

3.3.2 The Knowledge of Agents

Paulson [Pau98] formalizes the notion of *knowledge* for an agent as the messages that the agent either knows initially or sees during the course of communication.

We adopt this view. Paulson's theory that considers the initial knowledge can then be used without change and the theory that considers the knowledge gained from lists of events can be redefined to fit into our setup.

The corresponding encodings are enclosed in appendix A.3.3.2, but the order is slightly different from the following. In the appendix the theories for shared and public key systems are included last in the section.

The Initial Knowledge of Agents

Agents see messages that they either send or receive and the full knowledge gained from these is captured by application of the analz operator.

Since the analz and synth operators depend on the knowledge of appropriate keys for decryption and encryption, Paulson assumes an initial knowledge of such keys for the protocol to work. He defines the function:

$$\mathsf{initState} : \boldsymbol{agent} \to \boldsymbol{msg}$$

that for each agent describes the corresponding initial knowledge.

This also clarifies what it means for an agent to be compromised. The meaning of this is captured by a set

$$\boldsymbol{bad} : \mathcal{P}(\boldsymbol{agent})$$

the elements of which are agents, whose long-term keys belong to the initial state of the spy. Paulson makes two assumptions about this set:

$$\mathsf{Spy} \in \boldsymbol{bad}$$
$$\mathsf{Server} \notin \boldsymbol{bad}$$

The formal definition of initState depends on the kind of cryptosystem that is modeled.

Shared Key Systems In these systems each agent shares a symmetric key with the server.

$$\mathsf{shrK} : \boldsymbol{agent} \hookrightarrow \boldsymbol{key}$$

with the requirement that for all keys K

$$K \in \boldsymbol{symKeys}$$

For these systems the initial state is captured by

$$\begin{aligned} \mathsf{initState\ Server} &\hat{=} \{\mathsf{Key}(\mathsf{shrK}\ A) \mid A \in agent\} \\ \mathsf{initState\ (Friend}\ i) &\hat{=} \{\mathsf{Key}(\mathsf{shrK}(\mathsf{Friend}\ i))\} \\ \mathsf{initState\ Spy} &\hat{=} \{\mathsf{Key}(\mathsf{shrK}\ A) \mid A \in \boldsymbol{bad}\} \end{aligned}$$

Public Keys Systems Here there is a public/private key-pair corresponding to each agent.

$$\begin{aligned} \mathsf{pubK} &: \boldsymbol{agent} \hookrightarrow \boldsymbol{key} \\ \mathsf{priK}\ x \mathrel{\hat{=}} \mathsf{invKey}(\mathsf{pubK}\ x) &: \boldsymbol{agent} \to \boldsymbol{key} \end{aligned}$$

with the proviso that all symmetric keys are excluded.

$$\mathsf{priK}\ x \neq \mathsf{pubK}\ x$$

For these systems the initial state would be:

$$\begin{aligned} \mathsf{initState\ Server} &\mathrel{\hat{=}} \{\mathsf{Key}(\mathsf{priK\ Server})\} \cup \{\mathsf{Key}(\mathsf{pubK}\ A) \mid A \in \boldsymbol{agent}\} \\ \mathsf{initState\ (Friend}\ i) &\mathrel{\hat{=}} \{\mathsf{Key}(\mathsf{priK}(\mathsf{Friend}\ i))\} \cup \{\mathsf{Key}(\mathsf{pubK}\ A) \mid A \in \boldsymbol{agent}\} \\ \mathsf{initState\ Spy} &\mathrel{\hat{=}} \{\mathsf{Key}(\mathsf{priK}\ A) \mid A \in \boldsymbol{bad}\} \\ &\quad \cup \{\mathsf{Key}(\mathsf{pubK}\ A) \mid A \in \boldsymbol{agent}\} \end{aligned}$$

Knowledge Gained From Lists of Events

Agents augment their initial knowledge by analyzing the traffic, or at least the relevant parts of the traffic. We capture this by redefining Paulson's recursively defined function:

$$\mathsf{knows} : \boldsymbol{agent} \to (\boldsymbol{event\ list} \to \mathcal{P}(\boldsymbol{msg}))$$

in the following way:

$$\begin{aligned} \mathsf{knows}\ A\ [] &\mathrel{\hat{=}} \mathsf{initState}\ A \\ \mathsf{knows}\ A\ (\mathsf{Blk}\ A'\ P)\#evs &\mathrel{\hat{=}} \mathsf{knows}\ A\ evs \\ \mathsf{knows}\ A\ (\mathsf{Snd}\ A'\ P)\#evs &\mathrel{\hat{=}} \begin{cases} \{\mathsf{MsgPkt}\ P\} \cup \mathsf{knows}\ A\ evs & \text{if } A' \in A \\ \mathsf{knows}\ A\ evs & \text{otherwise} \end{cases} \\ \mathsf{knows}\ A\ (\mathsf{Rcv}\ A'\ P)\#evs &\mathrel{\hat{=}} \begin{cases} \{\mathsf{MsgPkt}\ P\} \cup \mathsf{knows}\ A\ evs & \text{if } A' \in A \\ \mathsf{knows}\ A\ evs & \text{otherwise} \end{cases} \end{aligned}$$

Knowledge of The Spy The most interesting characteristic of this function is how it defines the knowledge of the Spy. Like Paulson, we define the abbreviation

$$\mathsf{spies} \equiv \mathsf{knows\ Spy}$$

3.3.3 Freshness of Message Components

Paulson defines a notion of freshness [Pau98] based on the idea that a message is fresh if it has not been used before.

We redefine this notion, which is captured by the function:

$$\textbf{\textit{used}} : \textbf{\textit{event list}} \rightarrow \mathcal{P}(\textbf{\textit{msg}})$$

as

$$\begin{aligned}
\mathsf{used}\ [] &\ \hat{=} \bigcup A.\mathsf{parts}(\mathsf{initState}\ A) \\
\mathsf{used}\ ((\mathsf{Snd}\ A\ B\ X)\#evs) &\ \hat{=} \mathsf{parts}\{X\} \cup \mathsf{used}\ evs \\
\mathsf{used}\ ((\mathsf{Rcv}\ A\ B\ X)\#evs) &\ \hat{=} \mathsf{used}\ evs \\
\mathsf{used}\ ((\mathsf{Blk}\ A\ B\ X)\#evs) &\ \hat{=} \mathsf{used}\ evs
\end{aligned}$$

And, thus, being fresh is the same as not being initially known by anyone and not having been sent in the past.

3.4 Traces

Paulson [Pau98] defines protocols as all possible sequences (represented by lists) of network events. He calls these sequences traces and defines the set of possible traces as the least inductive closure of a given set of rules.

We consider protocols to be both the events of the networks that occurs when agents communicate, which we consider in the following, and also the actions performed locally by the individual agents, which we consider in chapter 5. To reflect this we refer to the legal sequences of network events as *global traces*.

3.4.1 Visible Packets

Obviously it is impossible to receive packets unless they are visible in the network layer. Intuitively, a packet is visible from it is sent and until it is blocked. We express this by the function:

$$\mathsf{visible} : \textbf{\textit{Pkt}} \rightarrow (\textbf{\textit{event list}} \rightarrow \textbf{\textit{bool}})$$

which is defined recursively by the following cases:

$$
\begin{aligned}
P \text{ visible } [] &\;\hat{=}\; \mathit{false} \\
P \text{ visible } (\mathsf{Snd}\ n\ p)\#\mathit{evs} &\;\hat{=}\; \begin{cases} \mathit{true} & \text{if } P = p \\ P \text{ visible } \mathit{evs} & \text{otherwise} \end{cases} \\
P \text{ visible } (\mathsf{Rcv}\ n\ p)\#\mathit{evs} &\;\hat{=}\; P \text{ visible } \mathit{evs} \\
P \text{ visible } (\mathsf{Blk}\ n\ p)\#\mathit{evs} &\;\hat{=}\; \begin{cases} \mathit{false} & \text{if } P = p \\ P \text{ visible } \mathit{evs} & \text{otherwise} \end{cases}
\end{aligned}
$$

3.4.2 Well-Formed Traces

In other words there is a causal relationship between sending and reception of packets that we need to capture. We do this by inductively defining a *realistic* model of the global traces of the protocol. By a realistic model we mean one that, besides of allowing agents to execute the protocol, allows other activities, such as the intrusive activities of attackers.

The least restrictive such model is that of the well-formed subset of event lists:

$$\boldsymbol{trace} : \mathcal{P}(\boldsymbol{event\ list})$$

as the least inductive closure of the following rules:

$$
\frac{}{[] \in \boldsymbol{trace}} \qquad \frac{h \in \boldsymbol{trace}}{(\mathsf{Snd}\ n\ p)\#h \in \boldsymbol{trace}}
$$

$$
\frac{h \in \boldsymbol{trace} \quad p \text{ visible } h}{(\mathsf{Rcv}\ n\ p)\#h \in \boldsymbol{trace}} \qquad \frac{h \in \boldsymbol{trace} \quad p \text{ visible } h}{(\mathsf{Blk}\ n\ p)\#h \in \boldsymbol{trace}}
$$

In general, realistic models should be restrictions of ***trace***, but in the following we will just use trace.

3.4.3 Projection of Traces

For a given agent some parts of a trace are more important than others, namely those consisting of events caused by him. In order to talk about these parts we define the projection of an event list *evs* with respect to a

given agent n, denoted $evs \downarrow n$, as the list of events obtained from evs by deleting every event not caused by n.

The formal definition of this is a recursive function

$$\downarrow : \textbf{\textit{event list}} \to (\textbf{\textit{agent}} \to \textbf{\textit{event list}})$$

given by

$$[] \downarrow n \;\hat{=}\; []$$

$$(\mathsf{Snd}\ m\ p)\#evs \downarrow n \;\hat{=}\; \begin{cases} (\mathsf{Snd}\ m\ p)\#(evs \downarrow n) & \text{if } n = m \\ evs \downarrow n & \text{otherwise} \end{cases}$$

$$(\mathsf{Rcv}\ m\ p)\#evs \downarrow n \;\hat{=}\; \begin{cases} (\mathsf{Rcv}\ m\ p)\#(evs \downarrow n) & \text{if } n = m \\ evs \downarrow n & \text{otherwise} \end{cases}$$

$$(\mathsf{Blk}\ m\ p)\#evs \downarrow n \;\hat{=}\; \begin{cases} (\mathsf{Blk}\ m\ p)\#(evs \downarrow n) & \text{if } n = m \\ evs \downarrow n & \text{otherwise} \end{cases}$$

3.5 Ideal Protocol Definitions

In order to identify active attacks we need a faithful reference model of the protocol, which is being analyzed. The ideal definitions must be modeled separately for every actual protocol. We will describe the general form of such definitions.

If in the following we think of the word *Ideal* as a wildcard for the descriptive name of a protocol then its definition would be typed:

$$\textbf{\textit{Ideal}} : \mathcal{P}(\textbf{\textit{event list}})$$

and defined as the least inductive closure of the rules:

$$\frac{}{[] \in \textbf{\textit{Ideal}}}$$

$$\frac{h \in \textbf{\textit{Ideal}} \quad (\!|(A,A) \rightarrow (B,B) : m|\!) \text{ visible } h}{\mathsf{Blk}\ A\ (\!|(A,A) \rightarrow (B,B) : m|\!) \# \mathsf{Rcv}\ A\ (\!|(A,A) \rightarrow (B,B) : m|\!) \# h \in \textbf{\textit{Ideal}}}$$

$$\frac{h \in \textbf{\textit{Ideal}}}{\mathsf{Snd}\ A\ (\!|(A,A) \rightarrow (B,B) : M_1|\!) \# h \in \textbf{\textit{Ideal}}}$$

$$\frac{h \in \textbf{\textit{Ideal}} \quad \mathsf{hd}\ (h \downarrow A) = \mathsf{Blk}\ A\ (\!|(A,A) \rightarrow (B,B) : M_1|\!)}{\mathsf{Snd}\ A\ (\!|(A,A) \rightarrow (B,B) : M_2|\!) \# h \in \textbf{\textit{Ideal}}}$$

$$\vdots$$

$$\frac{h \in \textbf{\textit{Ideal}} \quad \mathsf{hd}\ (h \downarrow A) = \mathsf{Blk}\ A\ (\!|(A,A) \rightarrow (B,B) : M_{last-1}|\!)}{\mathsf{Snd}\ A\ (\!|(A,A) \rightarrow (B,B) : M_{last}|\!) \# h \in \textbf{\textit{Ideal}}}$$

where m denotes any message of the protocol and the M_is denote messages of the kind that the protocols requires to be sent and received in its ith step.

This model is completely restrictive and includes no rules for a spy. We may consider this a description of an ideal world where the spy is either non-existent or just as restricted as other agents.

One awkwardness resulting from modeling with our particular primitives is that the agents are required to block packets immediately after reception.

This does not change the fact that the model describes an environment where the protocol is always executed faithfully. It does, however, not describe all such environments unless the protocol specification specifically requires a behavior similar to that of the model.

3.6 Schematics of Traces and Active Attacks

Now that traces and ideal descriptions have been formalized we will aim to define a mechanism that can actually identify active attacks.

A key observation is that agents may not agree on their *view* of the protocol execution. As long as the protocol execution, as seen by each agent, is

somehow satisfactory with respect to the protocol specification they accept the execution.

We decide whether this is the case by comparing the protocol execution with the abstract specification given in Alice-and-Bob notation (§2.1). We observe that the schematic lines of Alice-and-Bob specifications belong to:

$$\boldsymbol{pLnSchema} = \boldsymbol{Nm} \times \boldsymbol{Nm} \times \boldsymbol{msg}$$

which means that the full specifications belong to:

$$\boldsymbol{pSchema} = \boldsymbol{pLnSchema\ list}$$

3.6.1 Schematic Projection of Traces

The protocol executions that are well-behaved with respect to this abstract specification are exactly those described by the ideal protocol model.

In order to take advantage of this we define a projection

$$\uparrow : \boldsymbol{event\ list} \to (\boldsymbol{Nm} \to \boldsymbol{pSchema})$$

which basically brings an agents view of the protocol execution onto schematic form by application of the recursively defined function:

$$[] \uparrow n \ \hat{=}\ []$$

$$\mathsf{Snd}\ m\ (\!|(n_1, l_1) \to (n_2, l_2) : M|\!)\#evs \uparrow n \ \hat{=}\ \begin{cases} (\!|n_1 \to n_2 : M|\!)\#(evs \uparrow n) \\ \quad \text{if } n = m \wedge n = n_1 \\ evs \uparrow n \\ \quad \text{otherwise} \end{cases}$$

$$\mathsf{Rcv}\ m\ (\!|(n_1, l_1) \to (n_2, l_2) : M|\!)\#evs \uparrow n \ \hat{=}\ \begin{cases} (\!|n_1 \to n_2 : M|\!)\#(evs \uparrow n) \\ \quad \text{if } n = m \wedge n = n1 \\ evs \uparrow n \\ \quad \text{otherwise} \end{cases}$$

$$(\mathsf{Blk}\ m\ p)\#evs \uparrow n \ \hat{=}\ (evs \uparrow n)$$

When this projection is applied to a single execution of the ideal protocol model the result is exactly the corresponding schematic Alice-and-Bob specification. When applied to multiple executions, interleaving or not, the resulting schematic trace is always consistent with the abstract specification.

3.6.2 Consistent Executions

We can now define what we mean by a trace being consistent with respect to the protocol. It means that for all (good) agents participating in the trace the corresponding schematic view must be equal to that of some ideal execution.

We first introduce the function

$$\mathsf{Agents} : \boldsymbol{event\ list} \to \mathcal{P}(\boldsymbol{agent})$$

which names the good participants in a list of events. The full Isabelle/HOL implementation is in appendix A.3.6.

And then, by a function

$$\triangle : \boldsymbol{event\ list} \to (\mathcal{P}(\boldsymbol{event\ list}) \to \boldsymbol{bool})$$

we formally define the notion of consistency:

$$\begin{aligned} evs \triangle \boldsymbol{ideal} \mathrel{\hat{=}} & evs \in \boldsymbol{trace} \wedge \\ & \forall a \in (\mathsf{Agents}\ evs) \cdot \exists idl \in \boldsymbol{ideal} \cdot (idl \uparrow a = evs \uparrow a) \end{aligned}$$

3.6.3 Identifying Active Attacks

For an attack on any agent to be successful the corresponding protocol execution must finish. That is, the agent under attack must execute every line of the protocol faithfully whether it is in the role of initiator or responder.

This is formalized in the function:

$$\mathsf{finish} : \boldsymbol{event\ list} \to (\boldsymbol{agent} \to \boldsymbol{bool})$$

which appears in appendix A.3.6.

Now, finally we define the successful active attacks as traces that are consistent, finishes for the attacked agent, and contains a fake event among those caused by the attacked agent.

Formally we define the function:

$$\nabla : \boldsymbol{event\ list} \to (\boldsymbol{agent} \to \boldsymbol{bool})$$

as

$$evs \nabla a \mathrel{\hat{=}} (evs \ \triangle \ \textbf{\textit{Ideal}}) \wedge (evs \ \mathsf{finish} \ a) \wedge \exists e \in \mathsf{set} \ (evs \downarrow a) \cdot (\mathsf{fakeEvent} \ e)$$

CHAPTER 4

Example: The Station-to-Station Protocol

In order for the theory make sense we need to apply it. We will formally model the Station-to-Station protocol and apply the untimed theory to prove that the protocol may be executed consistently and that a particular active attack is successful.

The Station-to-Station (StS) protocol for authenticated key exchange is due to Diffie, Wiener, and Van Oorshot [DvOW92]. The following is its specification in Alice-and-Bob notation (§2.1):

1. $A \to B:\ \ \alpha^{X_A}$
2. $B \to A:\ \ \alpha^{X_B},\ \{\{\alpha^{X_B},\ \alpha^{X_A}\}_{S_B}\}_K$
3. $A \to B:\ \ \{\{\alpha^{X_A},\ \alpha^{X_B}\}_{S_A}\}_K$

The protocol enables two parties to agree on a shared secret (an integer valued key) by the exchange of public information. In the figure α is a publicly known integer, which is a primitive root of a publicly known prime q, and all arithmetic is assumed to take place *modulo* q. The exponents X_A and X_B are secret integers known only to A and B, respectively. The exchange allows A and B to compute the same secret integer key, $K = \alpha^{X_A X_B}$, which is subsequently used for authentication.

The secret key is constructed by the exchange of nonces taking place in the first two lines. The authentication procedure is then carried out in the last two lines with the aid of the newly constructed shared key and the private (signing) keys of the two agents.

This chapter documents novel work and the corresponding Isabelle/HOL encoding is enclosed in appendix A.4.

4.1 Modeling StS

StS is clearly based on a public key cryptosystem in order to allow message signing. The key, however, that results from the key exchange is symmetric. We need to capture all of this in our model.

Informal Model First we model semi-formally with messages held in the format used in the Alice-and-Bob specification above.

Hence, the set

$$\boldsymbol{StS} : \mathcal{P}(\boldsymbol{event\ list})$$

is given by the least inductive closure of the following rules

$$\frac{}{[] \in \boldsymbol{StS}}$$

$$\frac{evs \in \boldsymbol{StS} \quad (\!|(A,A) \rightarrow (B,B) : m|\!)\ \mathsf{visible}\ evs}{\mathsf{Blk}\ A\ (\!|(A,A) \rightarrow (B,B) : m|\!)\#\mathsf{Rcv}\ A\ (\!|(A,A) \rightarrow (B,B) : m|\!)\#evs \in \boldsymbol{StS}}$$

$$\frac{evs \in \boldsymbol{StS} \quad \alpha^{X_A} \notin \mathsf{used}\ evs}{\mathsf{Snd}\ A\ (\!|(A,A) \rightarrow (B,B) : \alpha^{X_A}|\!)\#evs \in \boldsymbol{StS}}$$

$$\frac{evs \in \boldsymbol{StS} \quad \alpha^{X_B} \notin \mathsf{used}\ evs \quad \mathsf{hd}(evs \downarrow B) = \mathsf{Blk}\ B\ (\!|(A,A) \rightarrow (B,B) : \alpha^{X_A}|\!)}{\mathsf{Snd}\ B\ (\!|(B,B) \rightarrow (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!)\#evs \in \boldsymbol{StS}}$$

$$\frac{evs \in \boldsymbol{StS} \quad \mathsf{hd}(evs \downarrow A) = \mathsf{Blk}\ A\ (\!|(B,B) \rightarrow (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!)}{\mathsf{Snd}\ A\ (\!|(A,A) \rightarrow (B,B) : \{\{\alpha^{X_A}, \alpha^{X_B}\}_{S_A}\}_K|\!)\#evs \in \boldsymbol{StS}}$$

Even though this is not completely formalized within our framework it pretty much nails the intentions down. Because of the size of the constructs involved we will explain the examples in a notation similar to the above, and not in the fully formal notation illustrated below.

Formal Model Of course the specification that we use in Isabelle/HOL must be fully formalized. In order to formalize the above model in this way we must make some choices. We simply assume that the half-keys α^X are special nonces that must be fresh and depend on an agents choice of secret integer X. The freshness of the nonces imply that agents must choose their secrets intelligently (unguessable and not previously used). We introduce the special functional abbreviation

$$\mathsf{HKey}\ X \triangleq \mathsf{Nonce}\ X : \mathbb{N} \rightarrow \boldsymbol{msg}$$

For the derived shared key K we simplify further. We say that it depends injectively on the two chosen secrets. Assuming that the secrets cannot be derived from the nonces this prevents the disclosure of the key to the spy. We have

$$\mathsf{combiK} : \mathbb{N} \times \mathbb{N} \hookrightarrow \boldsymbol{key}$$

with the proviso that

$$\mathsf{combiK}\ (A, B) \in \boldsymbol{symKeys}$$

We now go on to formalize the model. The result is somewhat sizeable and we use the pragmatic syntax of the Isabelle/HOL encoding to present it. In the following, the names of the rules are written to the left and the rules themselves to the right. The premises are written as lists enclosed by $[\![$ and $]\!]$. Elements of these lists are separated by semicolons. The conclusions are separated from their premises by $\Longrightarrow$.

`Nil` $[] \in \mathcal{sts}$

`Rcv` $[\![\ h \in \mathcal{sts};\ (\!|(B,B) \to (A,A) : m|\!)\ \mathsf{visible}\ h]\!] \Longrightarrow$
$\mathsf{Blk}\ A\ (\!|(B,B) \to (A,A) : m|\!)\#$
$\mathsf{Rcv}\ A\ (\!|(B,B) \to (A,A) : m|\!)\#h \in \mathcal{sts}$

`Msg1` $[\![\ h \in \mathcal{sts};\ \mathsf{HKey}\ XA \notin \mathsf{used}\ h]\!] \Longrightarrow$
$\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \mathsf{HKey}\ XA|\!)\#h \in \mathcal{sts}$

`Msg2` $[\![\ h \in \mathcal{sts};\ \mathsf{HKey}\ XB \notin \mathsf{used}\ h;\ \mathsf{hd}\ (h{\downarrow}B) = \mathsf{Blk}\ B\ (\!|(A,A) \to (B,B) : \mathsf{HKey}\ XA|\!)]\!] \Longrightarrow$
$\mathsf{Snd}\ B\ (\!|(B,B) \to (A,A) : \{\!|\mathsf{HKey}\ XB,\ \mathsf{Crypt}\ (\mathsf{combiK}\ (XA,XB))\ (\mathsf{Sign}[B]\ \{\!|\mathsf{HKey}\ XB,\ \mathsf{HKey}\ XA|\!\})|\!\}|\!)\#h \in \mathcal{sts}$

`Msg3` $[\![\ h \in \mathcal{sts};\ \mathsf{hd}\ (h{\downarrow}A) = \mathsf{Blk}\ A\ (\!|(B,B) \to (A,A) : \{\!|\mathsf{HKey}\ XB,\ \mathsf{Crypt}\ (\mathsf{combiK}\ (XA,XB))\ (\mathsf{Sign}[B]\ \{\!|\mathsf{HKey}\ XB,\ \mathsf{HKey}\ XA|\!\})|\!\}|\!)]\!] \Longrightarrow$
$\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \mathsf{Crypt}\ (\mathsf{combiK}(XA,XB))$
$(\mathsf{Sign}[A]\ \{\!|\mathsf{HKey}\ XA,\ \mathsf{HKey}\ XB|\!\})|\!)\#h \in \mathcal{sts}$

Except for the fontification this is identical to the actual Isabelle/HOL encoding presented in appendix A.4.1.

This model describes a restricted subset of the well-formed traces. This has been established formally. We have

Theorem 4.1.

$$\boldsymbol{StS} \subseteq \boldsymbol{trace}$$

Proof. This is proven by induction on the length of the elements of $\boldsymbol{StS}$. The formal proof-script can be found in appendix A.4.1. □

4.2 Consistency of Repeated Protocol Engagement

The idea of this section is to demonstrate a result, related to Paulson's possibility (§2.4.1) lemmas, for the encoded protocol. We will not show possibility directly, but it is a corollary of our main result.

Consider the following sequence of events:

$$repeat = \begin{cases}
\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \alpha^{X_A}|\!) \\
\mathsf{Rcv}\ B\ (\!|(A,A) \to (B,B) : \alpha^{X_A}|\!) \\
\mathsf{Blk}\ B\ (\!|(A,A) \to (B,B) : \alpha^{X_A}|\!) \\
\mathsf{Snd}\ B\ (\!|(B,B) \to (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\
\mathsf{Rcv}\ A\ (\!|(B,B) \to (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\
\mathsf{Blk}\ A\ (\!|(B,B) \to (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\
\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \{\{\alpha^{X_A}, \alpha^{X_B}\}_{S_A}\}_K|\!) \\
\mathsf{Rcv}\ B\ (\!|(A,A) \to (B,B) : \{\{\alpha^{X_A}, \alpha^{X_B}\}_{S_A}\}_K|\!) \\
\mathsf{Blk}\ B\ (\!|(A,A) \to (B,B) : \{\{\alpha^{X_A}, \alpha^{X_B}\}_{S_A}\}_K|\!) \\
\mathsf{Snd}\ A\ (\!|(A,A) \to (C,C) : \alpha^{X_{A_1}}|\!) \\
\mathsf{Rcv}\ C\ (\!|(A,A) \to (C,C) : \alpha^{X_{A_1}}|\!) \\
\mathsf{Blk}\ C\ (\!|(A,A) \to (C,C) : \alpha^{X_{A_1}}|\!) \\
\mathsf{Snd}\ C\ (\!|(C,C) \to (A,A) : \alpha^{X_C}, \{\{\alpha^{X_C}, \alpha^{X_{A_1}}\}_{S_C}\}_K|\!) \\
\mathsf{Rcv}\ A\ (\!|(C,C) \to (A,A) : \alpha^{X_C}, \{\{\alpha^{X_C}, \alpha^{X_{A_1}}\}_{S_C}\}_K|\!) \\
\mathsf{Blk}\ A\ (\!|(C,C) \to (A,A) : \alpha^{X_C}, \{\{\alpha^{X_C}, \alpha^{X_{A_1}}\}_{S_C}\}_K|\!) \\
\mathsf{Snd}\ A\ (\!|(A,A) \to (C,C) : \{\{\alpha^{X_{A_1}}, \alpha^{X_C}\}_{S_A}\}_K|\!) \\
\mathsf{Rcv}\ C\ (\!|(A,A) \to (C,C) : \{\{\alpha^{X_{A_1}}, \alpha^{X_C}\}_{S_A}\}_K|\!) \\
\mathsf{Blk}\ C\ (\!|(A,A) \to (C,C) : \{\{\alpha^{X_{A_1}}, \alpha^{X_C}\}_{S_A}\}_K|\!)
\end{cases}$$

where we consider A, B, and C friendly agents, i.e. Friend a, Friend b, and Friend c.

When we read this from top to bottom (the opposite way of formalization) it is clear that this is two consecutive executions of *StS*. We could easily prove that

$$repeat \in \boldsymbol{StS}$$

But, in order to get as much of the framework as possible into play, we prove the following property instead:

Theorem 4.2. *The sequence repeat is consistent with StS. That is, if the participants choose fresh nonces - all nonces pairwise unequal - the following property holds:*

$$repeat \ \triangle \ \boldsymbol{StS}$$

Proof. The proof follows the structure of the term in question. We remind that

$$\begin{aligned} &repeat \ \triangle \ \boldsymbol{StS} \mathrel{\hat{=}} repeat \in \boldsymbol{trace} \wedge \\ &\quad \forall a \in (\mathsf{Agents}\ repeat) \cdot \exists sts \in \boldsymbol{StS} \cdot (sts \uparrow a = repeat \uparrow a) \end{aligned}$$

First prove that *repeat* is a well-formed trace. Then reduce the quantifiers and apply the projections. We are then left with a number of instances of the general problem of showing that $\boldsymbol{StS}$ contains sequences that complete the protocol for pairs of agents. The full formal proof is named '*Repeat-consis*' and can be found in appendix A.4.2. A number of lemmas, applied in order to deal with the mentioned subgoals, are also found there. □

4.3 Active Attack on StS

Next we will consider an attack on the Station-to-Station protocol that was first showed by Lowe [Low96].

Consider the following sequence of events:

$$attack = \begin{cases} \mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \alpha^{X_A}|\!) \\ \mathsf{Rcv}\ \mathsf{Spy}\ (\!|(A,A) \to (B,B) : \alpha^{X_A}|\!) \\ \mathsf{Blk}\ \mathsf{Spy}\ (\!|(A,A) \to (B,B) : \alpha^{X_A}|\!) \\ \mathsf{Snd}\ \mathsf{Spy}\ (\!|(C,\mathsf{Spy}) \to (B,B) : \alpha^{X_A}|\!) \\ \mathsf{Rcv}\ B\ (\!|(C,\mathsf{Spy}) \to (B,B) : \alpha^{X_A}|\!) \\ \mathsf{Blk}\ B\ (\!|(C,\mathsf{Spy}) \to (B,B) : \alpha^{X_A}|\!) \\ \mathsf{Snd}\ B\ (\!|(B,B) \to (C,\mathsf{Spy}) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\ \mathsf{Rcv}\ \mathsf{Spy}\ (\!|(B,B) \to (C,\mathsf{Spy}) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\ \mathsf{Blk}\ \mathsf{Spy}\ (\!|(B,B) \to (C,\mathsf{Spy}) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\ \mathsf{Snd}\ \mathsf{Spy}\ (\!|(B,\mathsf{Spy}) \to (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\ \mathsf{Rcv}\ A\ (\!|(B,\mathsf{Spy}) \to (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\ \mathsf{Blk}\ A\ (\!|(B,\mathsf{Spy}) \to (A,A) : \alpha^{X_B}, \{\{\alpha^{X_B}, \alpha^{X_A}\}_{S_B}\}_K|\!) \\ \mathsf{Snd}\ A\ (\!|(A,A) \to (B,\mathsf{Spy}) : \{\{\alpha^{X_A}, \alpha^{X_B}\}_{S_A}\}_K|\!) \\ \mathsf{Rcv}\ \mathsf{Spy}\ (\!|(A,A) \to (B,\mathsf{Spy}) : \{\{\alpha^{X_A}, \alpha^{X_B}\}_{S_A}\}_K|\!) \\ \mathsf{Blk}\ \mathsf{Spy}\ (\!|(A,A) \to (B,\mathsf{Spy}) : \{\{\alpha^{X_A}, \alpha^{X_B}\}_{S_A}\}_K|\!) \end{cases}$$

where we consider A, B, and C friendly agents, i.e. Friend a, Friend b, and Friend c. And Spy is the spy.

In the terminology of section 2.2.2 we interpret this sequence as follows:

Friend a sends the first packet of a run to Friend b.
Spy performs a man-in-the-middle attack, stealing the packet and forwarding the enclosed message to Friend b while spoofing Friend c.
Friend b receives and blocks this packet believing to talk to Friend c.
Friend b responds to Spy (Friend c) with the second packet of the protocol run.
Spy forwards the enclosed message to Friend a while spoofing Friend b.
Friend a receives and blocks this packet believing to talk to Friend b.
Friend a responds to Spy (Friend b) with the last packet of the run, thus finishing his run.
Spy receives and blocks this last packet, thus finishing his attack.

This is an unfortunate sequence, because at the end of it Friend a believes to share a symmetric key with Friend b when, in reality, the key is shared with Spy.

We will show that our setup correctly classifies this sequence (or the formalization of it) as a successful active attack. First however we assert:

Theorem 4.3. *The sequence attack is a well-formed trace.*

$$attack \in \textbf{\textit{trace}}$$

Proof. The formal proof is conducted in Isabelle/HOL and the lemma, *Attack-is-trace*, with its proof-script can be found in appendix A.4.3. The proof simply reconstructs the shape of the term and then performs unification. □

And then we go on to present the main result:

Lemma 4.1. *The sequence attack finishes StS for Friend a.*

$$attack \ \mathsf{finish} \ \mathsf{Friend}\ a$$

Proof. The formal proof in Isabelle/HOL of the lemma, *Attack-finish-a*, which is trivial, is in appendix A.4.3. □

Lemma 4.2. *The sequence attack is consistent for* ***StS****.*

$$attack \ \triangle \ \textbf{\textit{StS}}$$

Proof. The formal proof in Isabelle/HOL of the lemma, *Attack-consis* is in appendix A.4.3. Proof is similar to that of theorem 4.1. □

Theorem 4.4. *The sequence attack is a successful attack on Friend a*

$$attack \ \nabla \ \mathsf{Friend}\ a$$

Proof. Formal proof of theorem, *Attack-a-successful*, is in appendix A.4.3. As we remember that

$$evs \ \nabla \ a \mathrel{\hat{=}} (evs \ \triangle \ \textbf{\textit{Ideal}}) \wedge (evs \ \mathsf{finish}\ a) \wedge \exists e \in \mathsf{set}\ (evs \downarrow a) \cdot (\mathsf{fakeEvent}\ e)$$

it is clear that the theorem follows directly from the lemmas and the fact that Friend a causes a fake event when accepting the last message. □

CHAPTER 5

Untimed Theory of Local Traces

Before any DoS relevant cost-based analysis of protocols can take place we must capture the costly internal actions of agents within our model. We will introduce the notion of local traces and explain their basic aspects.

In order to argue about *availability* in the sense of *non-denial of service* we have to realize that the availability of a service depends on the availability of certain resources to the service provider.

Meadows [Mea99] points this out and identifies the internal actions of message construction/verification, performed by the service provider as part of the protocol execution, as the primary resource spenders.

To capture the expenditure she proposes an augmented Alice-and-Bob notation that accounts for message construction/verification actions as well as the actual message exchange. Her ith protocol step would be:

$$i.\ A \rightarrow B : T_1, T_2, \ldots, T_j \| M \| O_1, O_2, \ldots, O_{k+1}$$

where A performs the internal actions $T_1, T_2, \ldots, T_j$ in order to construct the message. The message transfer is then described by the event(s) M, which contains both "A sends M to B" and "B receives M from A". Finally B performs the internal actions $O_1, O_2, \ldots, O_{k+1}$ in order to verify the message upon receipt. Meadows considers the last verification action of a step, O_{k+1}, a special accept action that progresses the protocol execution to the next step. An agent that performs this special action commits to performing all of the message preparation actions of the next protocol step and subsequently sending the corresponding message.

We will let these ideas inspire us in the following. We will view costs strictly in terms of (computational) time, but will leave this issue out of the development until the next chapter. Also we will ignore the causal relations between the local traces and the global trace until the next chapter.

For now we will focus on the modeling of internal actions and the order in which agents perform them when executing protocols.

The remainder of this chapter documents novel work and the corresponding Isabelle/HOL encoding is enclosed in appendix A.5.

5.1 A Suitable Set of Actions

For a protocol with n steps where the receiver performs k_i verification actions in the ith step we assume the following six disjoint sets:

SState	A set of $n$actions that advance agents to a state where they are ready to send
RState	A set of $n$actions that advance agents to A state where they are ready to receive
Ver	A set of $k_1 + \ldots + k_n$verification actions
Abort	A singleton set $\{*\}$
Accept	A singleton set $\{*\}$
Ready	A singleton set $\{*\}$

An action is either a sending state advancement action, a receiving state advancement action, a verification action, abort, accept, or ready.

$$\boldsymbol{action} = \boldsymbol{SState} \cup \boldsymbol{RState} \cup \boldsymbol{Ver} \cup \boldsymbol{Abort} \cup \boldsymbol{Accept} \cup \boldsymbol{Ready}$$

For this construction we have the injections:

$$\begin{aligned}
\mathsf{SState} &: \{i : \mathbb{N} \mid 0 < i \leq n\} \hookrightarrow \boldsymbol{action} \\
\mathsf{RState} &: \{i : \mathbb{N} \mid 0 < i \leq n\} \hookrightarrow \boldsymbol{action} \\
\mathsf{Ver} &: \{i : \mathbb{N} \mid 0 < i \leq n\} \hookrightarrow (\{j : \mathbb{N} \mid 0 < j \leq k_i\} \hookrightarrow (\boldsymbol{bool} \hookrightarrow \boldsymbol{action})) \\
\mathsf{Accept} &: \boldsymbol{action} \\
\mathsf{Abort} &: \boldsymbol{action} \\
\mathsf{Ready} &: \boldsymbol{action}
\end{aligned}$$

To understand where these actions come from we look at the annotated Alice-and-Bob description of such an n-step protocol:

$$
\begin{array}{ll}
1. & A \to B : T_{1_1}, T_{2_1}, \ldots, T_{j_1} \| M_1 \| O_{1_1}, O_{2_1}, \ldots, O_{k_1+1} \\
2. & A \to B : T_{1_2}, T_{2_2}, \ldots, T_{j_2} \| M_2 \| O_{1_2}, O_{2_2}, \ldots, O_{k_2+1} \\
 & \vdots \\
n. & A \to B : T_{1_n}, T_{2_n}, \ldots, T_{j_n} \| M_n \| O_{1_n}, O_{2_n}, \ldots, O_{k_n+1}
\end{array}
$$

When the sender in the ith step performs the $T_{i\_}$ actions in order to prepare message M_i the only visible effect is the expense of time. Furthermore, if the step is not the first one, the special accept action $O_{k_{i-1}+1}$ has just been performed, which commits the agent to the completion of the actions $T_{i\_}$. We do not need to distinguish between any of these actions and we gather the $O_{k_{i-1}+1}$ action and the $T_{i\_}$ actions into SState i. The special case SState 1 does not contain an O action.

While the SState i actions signifies that the agent is ready to send the ith message (cause a Snd event), the corresponding RState i action signifies that the agent is ready to receive the ith message (cause Rcv and Blk events).

The action O_{k_n+1} we regard as the execution accept action, Accept. Both the sender and the receiver performs this action after successfully completing their part in the nth step. After performing an Accept action agents finish the current protocol execution.

The remaining O_{i_j} actions are verification actions Ver i j b. The boolean values carried by these actions represent the results of applying suitable verification functions to the message just received.

If any such verification function fails the corresponding Ver action carries a False. When this is the case the next action performed is the execution abort action Abort. After performing this action agents finish the current protocol execution.

Initially, when an agent, A, is initialized the local trace is empty. The only sensible action for A to take is to prepare for the engagement in protocol executions. Therefore, A performs the abstract action Ready. When ready A may choose to participate in a protocol execution as either the initiator or the responder. Whenever A finishes a protocol execution A subsequently performs a Ready action.

5.2 The Local Model

So, when an agent executes a protocol a number of internal actions are performed. As we do it for events we will capture the sequences of such internal actions in lists, one for each agent.

Some of these lists are meaningful and some are not. Since no intruder can interfere with the sequencing of these actions, we can inductively define the set of all possible well-formed sequences. We call the well-formed sequences local traces and ***ltrace*** is the set containing all of them.

Now, since the number and sequencing of actions differ from protocol to protocol we have to define ***ltrace*** specifically for each of them. The form, however, can be generally described. In the following we will do this, using the Isabelle/HOL notation described in section 4.1.

Consider an n-step protocol in which the receiver in step i performs k_i verification actions. The following inductive definition describes the corresponding correct local traces for all agents:

Nil $\quad [] \in \mathit{ltrace}\ A$

$Ready$ $\quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{hd}\ [] \vee$
$\mathsf{hd}\ acs = \mathsf{Accept} \vee \mathsf{hd}\ acs = \mathsf{Abort}]\!] \Longrightarrow$
$\mathsf{Ready} \# acs \in \mathit{ltrace}\ A$

$Init$ $\quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{Ready}$
$\mathsf{SState}\ 1 \# acs \in \mathit{ltrace}\ A$

$Resp$ $\quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{Ready}$
$\mathsf{RState}\ 1 \# acs \in \mathit{ltrace}\ A$

$Rcvi$ $\quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{RState}\ i]\!] \Longrightarrow$
$\mathsf{Ver}\ i\ 1\ b \# acs \in \mathit{ltrace}\ A$

$Sndi$ $\quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{SState}\ i;\ i < n]\!] \Longrightarrow$
$\mathsf{RState}\ (i+1) \# acs \in \mathit{ltrace}\ A$

$Verij$ $\quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{Ver}\ i\ j\ True;$
$j < k]\!] \Longrightarrow$
$\mathsf{Ver}\ i\ (j+1)\ b \# acs \in \mathit{ltrace}\ A$

$$Verik \quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{Ver}\ i\ k\ True;\ i < n]\!] \Longrightarrow \mathsf{SState}\ (i+1)\ \#\ acs \in \mathit{ltrace}\ A$$

$$Accept \quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{SState}\ n \vee \mathsf{hd}\ acs = \mathsf{Ver}\ n\ k\ True]\!] \Longrightarrow \mathsf{Accept}\ \#\ acs \in \mathit{ltrace}\ A$$

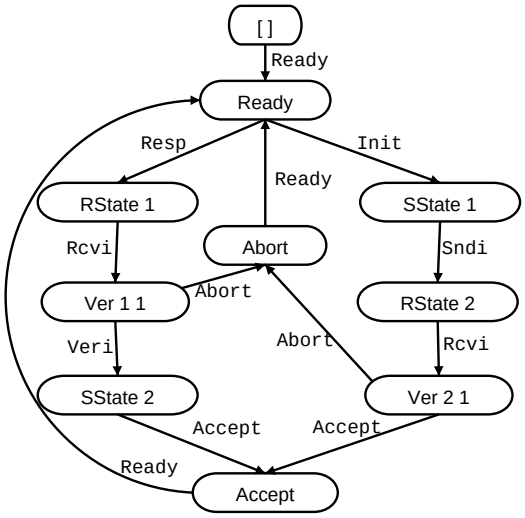
$$Abort \quad [\![\ acs \in \mathit{ltrace}\ A;\ \mathsf{hd}\ acs = \mathsf{Ver}\ i\ j\ False]\!] \Longrightarrow \mathsf{Abort}\ \#\ acs \in \mathit{ltrace}\ A$$

So, for a given protocol the inductive closure of an instantiation of the above rules define the set of well-formed local traces for all agents.

5.2.1 A Simple Example

In order to illustrate the outlined theory we will consider a protocol with two steps where the receiver in each step accepts or aborts according to the alleged identity of the sender.

Abstract Model In an abstract way the well-formedness of the corresponding traces is the same as being recognized by the following abstract machine:



The Well-Formed Set And the rules defining the corresponding set is actually the same as the transition rules for the machine:

Nil $[] \in \mathit{ltrace}\ A$

Ready $[\![\ \mathit{acs} \in \mathit{ltrace}\ A;\ \mathsf{hd}\ \mathit{acs} = \mathsf{hd}\ [] \vee \mathsf{hd}\ \mathit{acs} = \mathsf{Accept} \vee \mathsf{hd}\ \mathit{acs} = \mathsf{Abort}[\![\Longrightarrow \mathsf{Ready} \# \mathit{acs} \in \mathit{ltrace}\ A$

Init $[\![\ \mathit{acs} \in \mathit{ltrace}\ A;\ \mathsf{hd}\ \mathit{acs} = \mathsf{Ready} \mathsf{SState}\ 1 \# \mathit{acs} \in \mathit{ltrace}\ A$

Resp $[\![\ \mathit{acs} \in \mathit{ltrace}\ A;\ \mathsf{hd}\ \mathit{acs} = \mathsf{Ready} \mathsf{RState}\ 1 \# \mathit{acs} \in \mathit{ltrace}\ A$

Rcvi $[\![\ \mathit{acs} \in \mathit{ltrace}\ A;\ \mathsf{hd}\ \mathit{acs} = \mathsf{RState}\ i[\![\Longrightarrow \mathsf{Ver}\ i\ 1\ b \# \mathit{acs} \in \mathit{ltrace}\ A$

Sndi $[\![\ \mathit{acs} \in \mathit{ltrace}\ A;\ \mathsf{hd}\ \mathit{acs} = \mathsf{SState}\ i;\ i < 2[\![\Longrightarrow \mathsf{RState}\ (\mathit{suc}\ i) \# \mathit{acs} \in \mathit{ltrace}\ A$

$$Veri \quad [\![acs \in ltrace\ A;\ \mathsf{hd}\ acs = \mathsf{Ver}\ i\ 1;\ i < 2]\!] \Longrightarrow \mathsf{SState}\ (suc\ i)\ \#\ acs \in ltrace\ A$$

$$Accept \quad [\![acs \in ltrace\ A;\ \mathsf{hd}\ acs = Ver\ 2\ 1\ True \vee \mathsf{hd}\ acs = \mathsf{SState}\ 2]\!] \Longrightarrow \mathsf{Accept}\ \#\ acs \in ltrace\ A$$

$$Abort \quad [\![acs \in ltrace\ A;\ \mathsf{hd}\ acs = \mathsf{Ver}\ i\ j\ False]\!] \Longrightarrow \mathsf{Abort}\ \#\ acs \in ltrace\ A$$

Defining Verification Functions In order for verification to take place we must define appropriate verification functions.

In the outlined case we only need one - namely a function that can verify if the sender name belongs to a group that we consider ***bad***.

$$\mathsf{chkAgent}\ A \mathrel{\hat{=}} A \notin \boldsymbol{bad} : \boldsymbol{agent} \rightarrow \boldsymbol{bool}$$

Verification And we can now verify properties such as:

Lemma 5.1. *Whenever a local trace contains a* Ver *i j* Spy *action it is followed by an* Abort *action.*

$$\forall acs \in \boldsymbol{ltrace}\ A \cdot acs = a\#(\mathsf{Ver}\ 1\ 1\ (\mathsf{chkAgent\ Spy})) \#(acts :: \boldsymbol{action\ list}) \Longrightarrow a = \mathsf{Abort}$$

Proof. The full proof, which can be found in appendix A.5.2, proceeds by case-analysis and is quite simple. □

Lemma 5.2. *There are well-formed local traces that finish the protocol execution with an* Accept *action.*

$$\exists acts \in \boldsymbol{ltrace}\ A \cdot \mathsf{Accept}\#acts \in \boldsymbol{ltrace}\ A$$

Proof. The proof simply derives one such local trace. Appendix A.5.2. □

CHAPTER 6

Timed Properties of Security Protocols

> *Intuitively the causal orderings of traces are also temporal orderings. It is entirely possible that such orderings may be described adequately by temporal formalisms such as Interval Logics. We attempt such a description in Neighborhood Logic.*

The inductively defined traces of the previous chapter are fine tools for describing both global and local well-formed traces. Neither do they, however, capture the obvious causal relationship between the occurrence of certain actions in some local trace and the occurrence of corresponding events in the global trace, nor vice versa.

One solution to this problem would be to define local and global traces as mutually inductive sets. This is entirely feasible and some results of experiments of this kind can be found in appendix D[1].

On the other hand, it is not obvious how easily timed properties of protocols can be modeled in this way. Extra fields in actions and events may be sufficient when it is only necessary to accumulate the cost of performed actions. Other - more intricate, perhaps real-time - properties demand other solutions.

Intuitively the growth of traces and the passing of time are closely connected. Lamport highlights this relationship in [Lam78]. His observation

[1] Due to a bug in the batch processor of Isabelle this theory is pretty-printed with the a2ps pretty-printer presented in appendix E.

is that in distributed systems the ordering relation '*happened before*' (similar in nature to Paulson's inductive step, Gong and Syverson's causality, and Meadows' *desirably-causally-before* relation) is only a partial ordering of the events in the system because the only *observable ordering* is that of causality.

This allows us to interpret the inductively defined sets as *posets* with a well-defined *bottom*, namely the empty list []. The natural ordering relation of such sets, $\preceq$, we can interpret both as *is-a-prefix-of* and *precedes-or-equals* when thinking of the structures as lists and traces respectively.

In turn this motivates the idea that we may be able to characterize all traces if we can model them as functions of time. I.e. that we can use a timed formalism to capture exactly how the growth of traces depends on the passing of time.

This chapter describes the developed timed theory of protocols:

- The timed developments are based on a labelled variant of Zhou and Hansen´s Neighborhood Logic (NL) [CH98]. We motivate this choice and describe the logic (NLDCHOL) informally (§6.1).
- Meaningful use of the relational $\preceq$ operator and lists within NLDCHOL requires the operator and its rules to be defined. Also the basic *properties of lists* need to be formalized (§6.2).
- The modeling of properties of global traces requires formalization of certain properties of events. Furthermore, a suitable set of modeling primitives in the form of predicates may simplify such modeling substantially (§6.3).
- The modeling of local traces are subject to the same requirements as that of global traces (§6.4).
- With an appropriate framework in place *heuristics* for the modeling of protocols may be discussed (§6.5).

Except for section 6.1 this chapter presents novel work. The corresponding Isabelle/HOL encoding is enclosed in appendix A.6.

6.1 A Suitable Interval Logic

We want to model timed properties of security protocols. We want to do this using a temporal formalism where the notion of time is treated in as natural a way as possible.

From the outset we have desired to investigate whether Interval Logics could be successfully applied to the domain. Part of the problem is to formalize causal relationship between events, e.g. given the occurrence of a particular event we know that some other events must have occurred in the past or that some event must occur in the future.

Neighborhood Logic (NL) [CH98] with its two unary modalities $\Diamond_l$ and $\Diamond_r$ seems suitable for expressing such properties.

During his PhD. Rasmussen developed a labelled natural deduction proof system for first order Interval Logics [Ras01b]. He also provided automated proof support for such LND systems through an Isabelle (pure) encoding of his Labelled Signed Interval Logic (LSIL) [Ras01a]. Finally, in his thesis he outlined how this first order development could successfully be embedded in Isabelle/HOL [Ras02]. He has subsequently completed such a port resulting in Isabelle/LSILHOL.

We have formalized the present work in Isabelle/NLDCHOL which is a derivative of Isabelle/LSILHOL. To give a presentation which is reasonably similar to the actual encoding we use a labelled variant of NL to describe the timed theory.

In this section we shall informally introduce this logic. First unlabelled NL and then the distinguishing features of the labelled version. For this purpose we will in the following let $x, y, z, \ldots$ denote variables, $s, t, u, \ldots$ denote terms, and $\phi, \psi, \chi, \ldots$ denote formulas.

6.1.1 Neighborhood Logic

The logic is a modal interval logic where the possible worlds are bounded and closed intervals:

$$\boldsymbol{intv} = \{[b, e] \mid b, e \in \mathbb{R} \wedge b \leq e\}$$

Semantically all formulas are interpreted with respect to such given intervals. The intervals are not, however, part of the syntax of NL.

The syntax is basically that of First Order Logic with equality (FOL) with the addition of the following modal constructions:

$\phi^\frown\psi$	:	The interval can be partitioned so ϕ holds on the first part and ψ on the last.
$\Diamond\phi$	:	There is a subinterval where ϕ holds.
$\Diamond_l\phi$	:	There is a left neighborhood where ϕ holds.
$\Diamond_r\phi$	:	There is a right neighborhood where ϕ holds.
$\Diamond_l^c\phi$	:	There is a converse left neighborhood where ϕ holds.
$\Diamond_r^c\phi$	:	There is a converse right neighborhood where ϕ holds.
$\diamondsuit\,\phi$	:	There is a proper subinterval where ϕ holds.
$\Diamond_a\phi$	:	There is an interval (somewhere) where ϕ holds.
$\Box\phi$	:	ϕ holds on all subintervals.
$\Box_l$	:	ϕ holds on all left neighborhoods.
$\Box_r$	:	ϕ holds on all right neighborhoods.
$\Box_l^c\phi$	:	ϕ holds on all converse left neighborhoods.
$\Box_r^c\phi$	:	ϕ holds on all converse right neighborhoods.
$\boxdot\,\phi$	:	ϕ holds on all proper subintervals.
$\Box_a\phi$	:	ϕ holds on all intervals (anywhere).

For example $\Diamond_r\phi$ holds on $[b,e]$ if there is a time t, where $t \geq e$, and ϕ holds on $[e,t]$. $[e,t]$ is a *right neighborhood* of $[b,e]$. Similarly $[t,b]$ is a *left neighborhood* if $t \leq b$.

The formula $\Diamond_l^c\phi$ holds on $[b,e]$ if there is a time t, where $t \leq e$, and ϕ holds on $[t,e]$. This is the *converse* of $\Diamond_l\phi$ and $[t,e]$ is a *converse left neighborhood*. Similarly $[b,t]$ is a *converse right neighborhood* if $b \leq t$.

The formulas $\Diamond\phi$ and $\diamondsuit\,\phi$ are closely related. If we have times t_1 and t_2 such that $t_1 \leq t_2$ and ϕ holds on $[t_1,t_2]$, then $\Diamond\phi$ holds on $[b,e]$ if $b \leq t_1 \wedge t_2 \leq e$, while $\diamondsuit\,\phi$ holds on $[b,e]$ if $b < t_1 \wedge t_2 < e$. In the first case $[t_1,t_2]$ is a *subinterval* in the other case a *proper subinterval*.

Of the above modalities $\Diamond, \Box, \Diamond_l, \Diamond_r, \Box_l, \Box_r$ and their introduction/elimination rules are part of Rasmussen's work and can be found in appendix B.5. The remaining modalities and their intro-/elim-rules have been encoded in Isabelle-NLDCHOL and can be found in appendix A.6.1.

6.1.2 Rigidity and Chop-freeness

The logic distinguishes syntactically between different types of terms and formulas. Function/predicate symbols (and thus, terms by closure) are divided into *rigid* symbols and *flexible* symbols.

The interpretation of a *rigid* term is the same on every interval (possible world). The interpretation of a *flexible* term, however, depends on the interval context. The special nullary function symbol ℓ, which denotes the length of the current interval, is flexible. The relational symbol '=' and the functional symbols '0', '+', '-' are rigid. If a formula contains a flexible term we call it flexible, otherwise we call it rigid.

Furthermore, we distinguish between formulas that are *chop-free* (or *modality-free*) and formulas that are not. Basically, all formulas that do not contain modal symbols are chop-free.

Rigidity and chop-freeness are central to the logic. Without them we could not event talk about a modal logic as the whole structure would collapse into ordinary FOL. So obviously, rigidity and crop-freeness have profound influence on the way proofs are conducted. First of all rigid formulas have the property that if they hold in one possible world they hold in all possible worlds. Furthermore, either chop-freeness of formulas or rigidity of terms must be assumed in order for substitutions, specializations from universal quantifications, and generalizations from witnesses to existential quantifications to be sound.

6.1.3 Labelled Neighborhood Logic

The labelled logic of NLDCHOL extends the above in a number of ways. Instead of leaving the intervals in the semantics they are asserted explicitly, i.e. the logic deals with labelled formulas of the form:

$$(b, e) : \chi$$

which reads: "The formula χ holds on the bounded and closed interval $[b, e]$". Here χ can be any formula of NL.

Furthermore, the notions of rigidity and chop-freeness of terms s and formulas ϕ are made explicit in the syntax of the logic via the (unlabelled) judgments $\mathsf{ri}(s)$, $\mathsf{ri}(\phi)$ and $\mathsf{cf}(\phi)$.

Thanks to the labelled formulas the proof system is a proper (labelled) *natural deduction system* [BMV97]. Also, the labels allow the first order formulas of NL to exist side by side with formulas of Higher Order Logic. The following three rules due to Rasmussen allow the logic to benefit from

results obtained in HOL:

$$\frac{\mathsf{ri}(P) \quad P}{w : P}\ labelI\mathsf{ri} \qquad \frac{\mathsf{cf}(P) \quad P}{w : P}\ labelI\mathsf{cf} \qquad \frac{\forall w \cdot w : P}{P}\ labelE$$

where $labelE$ requires proof that P holds in all possible worlds.

As we shall see in the following these properties of the logic reduce a lot of the modeling issues to issues of rigidity and chop-freeness.

6.2 Timed Lists

As previously mentioned, we need to establish a notion of lists with an appropriate ordering relation in NLDCHOL before any modeling can take place. The Isabelle encoding of the constructs of this section can be found in appendix A.6.2.

6.2.1 Causality and Prefix Operators

We start out by defining the *precedes-or-equals* operator in HOL. We define the infix operator:

$$\preceq\text{: } \textit{'a list} \to (\textit{'a list} \to \textit{bool})$$

which basically means: "*is-a-prefix-of*". A suitable definition is:

$$xs \preceq ys \mathrel{\hat{=}} \exists zs : \textit{'a list} \cdot ys = zs @ xs$$

where @ is *list concatenation* or *append* as known from ML. In the context of inductively defined traces we say that $xs \preceq ys$ means: "xs is *causally before* ys or they are equal."

6.2.2 Rigid and Chop-free Terms and Formulas

With lists we have an entirely new class of terms. Intuitively, a list is rigid if all elements are and vice versa. We assert this axiomatically:

$$\frac{}{\mathsf{ri}([])}\ \mathsf{ri}[] \qquad \frac{\mathsf{ri}(x) \quad \mathsf{ri}(xs)}{\mathsf{ri}(x\#xs)}\ \mathsf{ricons}I$$

$$\frac{\mathsf{ri}(x\#xs)}{\mathsf{ri}(x)}\ \mathsf{ricons}E1 \qquad \frac{\mathsf{ri}(x\#xs)}{\mathsf{ri}(xs)}\ \mathsf{ricons}E2$$

Obviously, since lists are just a new kind of terms, the equality and prefix operators for lists have the same properties of rigidity and chop-freeness as other operators of NLDCHOL.

So, provided that s and t are lists we establish the following introduction and elimination rules for rigidity axiomatically:

$$\frac{\mathsf{ri}(s) \quad \mathsf{ri}(t)}{\mathsf{ri}(s \preceq t)}\ \mathsf{ri}{\preceq}I \qquad \frac{\mathsf{ri}(s \preceq t)}{\mathsf{ri}(s)}\ \mathsf{ri}{\preceq}E1 \qquad \frac{\mathsf{ri}(s \preceq t)}{\mathsf{ri}(t)}\ \mathsf{ri}{\preceq}E2$$

$$\frac{\mathsf{ri}(s) \quad \mathsf{ri}(t)}{\mathsf{ri}(s = t)}\ \mathsf{ri}{=}I \qquad \frac{\mathsf{ri}(s = t)}{\mathsf{ri}(s)}\ \mathsf{ri}{=}E1 \qquad \frac{\mathsf{ri}(s = t)}{\mathsf{ri}(t)}\ \mathsf{ri}{=}E2$$

We know that all atomic formulas are chop-free. The equality and prefix relations on lists are clearly atomic. So, provided that s and t are lists we assert:

$$\frac{}{\mathsf{cf}(s \preceq t)}\ \mathsf{cf}{\preceq} \qquad \frac{}{\mathsf{cf}(s = t)}\ \mathsf{cf}{=}$$

6.2.3 Timed Operators

Using the previously explained relationship between HOL and NLDCHOL (§6.1.3) we can prove a number of interesting properties for $=$ and $\preceq$ in HOL and subsequently import them into NLDCHOL.

Both relations are *reflexive* and *transitive*.

$$\frac{}{w : xs \preceq xs}\ \preceq refl \qquad \frac{}{w : xs = xs}\ = refl$$

$$\frac{w : xs \preceq ys \quad w : ys \preceq zs}{w : xs \preceq zs}\ \preceq trans \qquad \frac{w : xs = ys \quad w : ys = zs}{w : xs = zs}\ = trans$$

The relation $\preceq$ is *antisymmetric* while $=$ is *symmetric*.

$$\frac{w : xs \preceq ys \quad w : ys \preceq xs}{w : xs = ys}\ \preceq anti \qquad \frac{w : ys = xs}{w : xs = ys}\ = sym$$

Weakening applies such that equality of lists implies a prefix relationship either way:

$$\frac{w : xs = ys}{w : xs \preceq ys}\ = weaken1 \qquad \frac{w : xs = ys}{w : ys \preceq xs}\ = weaken2$$

6.3 Timed Global Traces

In the following we will formalize the relationship between time and global traces. The intuition is that these traces depend on time in a functional manner.

We deal with this issue in this section. Rigidity and chop-freeness of equality of events can be assumed in the following. The encodings of the structures presented in the following are enclosed in appendix A.6.3. When looking up rules in appendix substitute *pre* for $\preceq$ and *equ* for $=$ in names.

6.3.1 The Flexible Constant Tr

In NLDCHOL the time is explicit because of the labelled formulas. In some sense, however, the time is still implicit for the embedded formulas (χ in §6.1.3). Flexible terms, such as the special nullary function symbol ℓ, may be perceived as implicit functions of the interval endpoints. E.g., we could think of ℓ as:

$$\ell \mathrel{\hat{=}} \lambda b\, e \cdot e - b : \textbf{\textit{real}} \times \textbf{\textit{real}} \to \textbf{\textit{real}}$$

The logic itself operates at a higher level of abstraction. Therein lies its strength. Inspecting the logic at this lower level, however, inspires the idea that we may portray traces as such implicit functions of intervals. So, we introduce the trace as another flexible term or special nullary function symbol:

$$\mathsf{Tr} \in \textbf{\textit{event list}}$$

Unlike what was the case for ℓ it does not make sense to think of this term as a function of both endpoints. Assuming this is the case Tr must have the same value everywhere on the interval. We then face the problem of meaningfully separating $[b, e]$ in figure 6.1 into $[b, k]$ and $[k, e]$ without assuming $s = u = t$. Thus, either Tr is rigid, or we get nonsense.

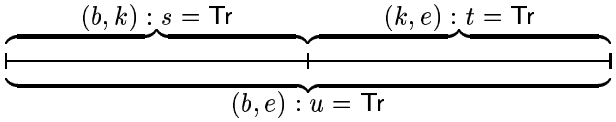


Figure 6.1: Tr as flexible function of intervals..

Instead we associate the value of Tr with only the endpoint of the interval. Thus, in figure 6.1 we know that $t = u$ but nothing about how s relates to t and u.

Our intuition is that Tr grows monotonously with time and therefore $s \preceq u$. The following axiomatic rules capture this:

$$\frac{w : \mathsf{Tr} = evs \quad \mathsf{ri}(evs)}{w : \Box_l^c(\mathsf{Tr} = evs)}\ Tr\_mono1 \qquad \frac{w : \mathsf{Tr} = evs \quad \mathsf{ri}(evs)}{w : \Box_r(evs \preceq \mathsf{Tr})}\ Tr\_mono2$$

$$\frac{w : \mathsf{Tr} = evs \quad \mathsf{ri}(evs)}{w : \Box_l^c(\Box_l(\mathsf{Tr} \preceq evs))}\ Tr\_mono3$$

These rules are in fact sufficient to completely describe the monotony and all other cases, such as the following, are theorems of the logic:

$$\frac{w : \mathsf{Tr} = evs \quad \mathsf{ri}(evs)}{w : \boxdot(\mathsf{Tr} \preceq evs)}\ Tr\_mono5 \qquad \frac{w : \mathsf{Tr} = evs \quad \mathsf{ri}(evs)}{w : \Box_r(\Diamond_l^c(evs \preceq \mathsf{Tr}))}\ Tr\_mono6$$

The formal proofs of these theorems are enclosed in appendix A. 6.3.1. The proofs of the two major assist lemmas regarding Tr and $\preceq$ (shown below) are also included in this appendix.

$$\frac{\mathsf{ri}(evs)}{w : evs \preceq \mathsf{Tr} \iff (\exists x \cdot \mathsf{Tr} = x \wedge evs \preceq x)}\ Tr\_\preceq Tr\_expand$$

$$\frac{\mathsf{ri}(evs)}{w : \mathsf{Tr} \preceq evs \iff (\exists x \cdot \mathsf{Tr} = x \wedge x \preceq evs)}\ Tr\_Tr\preceq\_expand$$

6.3.2 Predicates Concerning Tr

In order to simplify the modeling as much as possible we construct a toolbox of useful predicates. In the following we consider occurrences of events in the traces instantaneous - they are always confined to point intervals. Also, when describing stable properties that do not change throughout an interval of some non-zero duration we specifically exclude the endpoints. Without these guidelines we could only avoid inconsistency if no events ever occurred in the trace as any such occurrence would introduce a contradiction in the point of change.

We use the nullary predicate symbol

$$\mathsf{Tr\_empty} \mathrel{\hat{=}} \mathsf{Tr} = [] : \textbf{\textit{bool}}$$

to assert that the trace of the current interval is *empty.*

The binary predicate symbol

$$\begin{aligned}&\mathsf{Tr\_throughout} : \textbf{\textit{agent}} \to (\textbf{\textit{event list}} \to \textbf{\textit{bool}})\\ &\mathsf{Tr\_throughout}\ A\ evs \mathrel{\hat{=}} 0 < \ell \wedge \boxdot(\mathsf{Tr} \downarrow A = evs)\end{aligned}$$

expresses that agent A's *fragment* of the global trace, $(\mathsf{Tr} \downarrow A)$, has a particular value, evs, *throughout* the interval, endpoints excluded.

The existence of any such value, evs, implies *stability* for A. That is, A causes no new events within the interval, endpoints excluded. This is expressed by the unary predicate

$$\begin{aligned}&\mathsf{Tr\_stable} : \textbf{\textit{agent}} \to \textbf{\textit{bool}}\\ &\mathsf{Tr\_stable}\ A \mathrel{\hat{=}} \exists evs : \textbf{\textit{event list}} \cdot \mathsf{Tr\_throughout}\ A\ evs\end{aligned}$$

If for a certain interval stability holds for all agents we say that the trace is *constant* for the interval and model it with the nullary predicate:

$$\mathsf{Tr\_const} \mathrel{\hat{=}} \forall A : \textbf{\textit{agent}} \cdot \mathsf{Tr\_stable}\ A : \textbf{\textit{bool}}$$

When an agent A *does* cause an event e it marks the end of a non-point interval throughout which A's fragment has been static. At this point the previously static fragment is extended with the event e. We express this with the binary predicate symbol

$$\begin{aligned}&\mathsf{Tr\_causes} : \textbf{\textit{agent}} \to (\textbf{\textit{event}} \to \textbf{\textit{bool}})\\ &\mathsf{Tr\_causes}\ A\ e \mathrel{\hat{=}} \ell = 0 \wedge (\exists evs : \textbf{\textit{event list}} \cdot\\ &\qquad\qquad \mathsf{Tr} \downarrow A = e \# evs \wedge \Diamond_l \mathsf{Tr\_throughout}\ A\ evs)\end{aligned}$$

The existence of any agent, A, causing the event e implies that e *occurs* in the trace.

$$\begin{aligned}&\mathsf{Tr\_occurs} : \textbf{\textit{event}} \to \textbf{\textit{bool}}\\ &\mathsf{Tr\_occurs}\ e \mathrel{\hat{=}} \exists A \cdot \mathsf{Tr\_causes}\ A\ e\end{aligned}$$

On the other hand an agent, A, may have a quiet moment, i.e. in a particular point A explicitly causes no event.

$$\mathsf{Tr\_quiet} : \textbf{\textit{agent}} \to \textbf{\textit{bool}}$$
$$\mathsf{Tr\_quiet}\ A \mathrel{\hat{=}} \ell = 0 \wedge \neg(\exists e \cdot \mathsf{Tr\_causes}\ A\ e)$$

Axiomatic Rules Concerning Predicates

Given these predicates it is natural to specify some further properties of timed traces in terms of them. The rules described in the following are encoded as axioms.

First of all we assert *finite variability* for the trace fragment of an agent A. Any interval is preceded and succeeded by a non-point interval where A is stable.

$$\frac{}{(i,j) : \Diamond_l(\mathsf{Tr\_stable}\ A)}\ Tr\_lconst \qquad \frac{}{(i,j) : \Diamond_r(\mathsf{Tr\_stable}\ A)}\ Tr\_rconst$$

When an agent, A, is quiet in a particular point the trace must have the same value in non-point intervals before and after the point.

$$\frac{(i,j) : \mathsf{Tr\_quiet}\ A}{(i,j) : \ell = 0 \wedge \left(\begin{array}{c} \exists evs \cdot \mathsf{Tr} \downarrow A = evs \wedge \\ \Diamond_l(\mathsf{Tr\_throughout}\ A\ evs) \wedge \\ \Diamond_r(\mathsf{Tr\_throughout}\ A\ evs) \end{array}\right)}\ Tr\_silence$$

When an event occurs in the global trace the identity of the agent causing the event is accessible. In other words, occurs implies causes.

$$\frac{(i,j) : \mathsf{Tr\_occurs}(\mathsf{Snd}\ A\ P)}{(i,j) : \mathsf{Tr\_causes}\ A\ (\mathsf{Snd}\ A\ P)}\ Tr\_O\_Snd\_imp\_C\_Snd$$

$$\frac{(i,j) : \mathsf{Tr\_occurs}(\mathsf{Rcv}\ A\ P)}{(i,j) : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ P)}\ Tr\_O\_Rcv\_imp\_C\_Rcv$$

$$\frac{(i,j) : \mathsf{Tr\_occurs}(\mathsf{Blk}\ A\ P)}{(i,j) : \mathsf{Tr\_causes}\ A\ (\mathsf{Blk}\ A\ P)}\ Tr\_O\_Blk\_imp\_C\_Blk$$

In the above (and below) the capital letter O abbreviates occurs and C abbreviates causes. This naming schema based on abbreviations of the predicate names is used throughout the following.

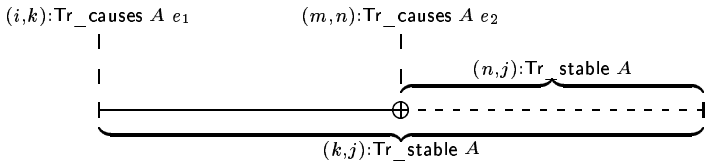
6.3.3 Lemmas and Theorems Regarding Predicates

There are a number of properties that are useful when proving theorems of the logic. Our intuition about them is that they are all provable theorems of the logic. We have not yet succeeded with most of the proofs, though. So, most of the lemmas presented in the following have actually been encoded into the logic as axioms. In the following we shall give rather informal arguments as to why they are sound.

Lemma 6.1. *"Tr_ CS_ CS_ equ"*

$$\frac{(i,k):\mathit{Tr\_causes}\ A\ e_1 \quad (m,n):\mathit{Tr\_causes}\ A\ e_2 \quad (k,j):\mathit{Tr\_stable}\ A \quad (n,j):\mathit{Tr\_stable}\ A}{(n,k):\ell{=}0}$$

Proof. Assume that the two points (i,k) and (m,n) are not the same, i.e. $(n,k) : \ell \neq 0$:



When A causes the event e_2 the stability of (k,j) is compromised and, thus, the assumption leads to contradiction. □

Lemma 6.2. *"Tr_ E_ O_ fwd"*

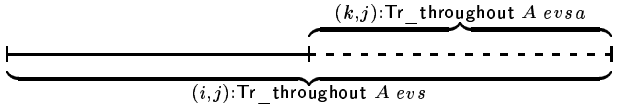
$$\frac{(i,k) : \mathit{Tr\_empty} \quad (l,j) : \mathit{Tr\_occurs}\ e}{(k,l) : \ell \geq 0}$$

Proof. Assuming (l,j) to be temporally before (k,k) contradicts our assumption about monotony for Tr. □

Lemma 6.3. *"Tr_ overlap1"*

$$\frac{(i,j):\mathit{Tr\_throughout}\ A\ evs \quad (k,j):\mathit{Tr\_throughout}\ A\ evsa \quad \mathit{ri}(evs) \quad \mathit{ri}(evsa)}{(i,j):evs{=}evsa}$$

Proof. Assume that $evs \neq evsa$:



Since both of the intervals (i,j) and (k,j) have the property $\ell > 0$ there will be an interval of overlap. On this interval we have a contradiction. □

Lemma 6.4. *"Tr_overlap2"*

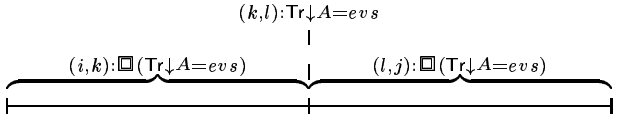
$$\frac{(i,j)\text{:}\mathit{Tr\_throughout}\ A\ evs \quad (i,k)\text{:}\mathit{Tr\_throughout}\ A\ evsa \quad \mathit{ri}(evs) \quad \mathit{ri}(evsa)}{(i,j)\text{:}evs{=}evsa}$$

Proof. Similar to that of "Tr_overlap1" (6.3). □

Lemma 6.5. *"Tr_connected"*

$$\frac{(i,k)\text{:}\Box(\mathsf{Tr}{\downarrow}A{=}evs) \quad (k,l)\text{:}\ell{=}0 \quad (k,l)\text{:}\mathsf{Tr}{\downarrow}A{=}evs \quad (l,j)\text{:}\Box(\mathsf{Tr}{\downarrow}A{=}evs)}{(i,j)\text{:}\Box(\mathsf{Tr}{\downarrow}A{=}evs)}$$

Proof.



When keeping in mind that Tr is associated with the endpoint only it is intuitively clear from the drawing that there is no choice of proper subinterval of (i,j) such that $\mathsf{Tr} \neq evs$. □

Lemma 6.6. *"Tr_silent_imp_S"*

$$\frac{w : 0 < \ell \quad w : \mathit{Tr\_stable}\ A \frown \mathit{Tr\_quiet}\ A \frown \mathit{Tr\_stable}\ A}{w : \mathit{Tr\_stable}\ A}$$

Proof. This proof has been conducted formally in Isabelle. It relies on "Tr_connected". See appendix A.6.3.3. □

6.4 Timing Local Traces

The relationship between time and the local traces is formalized much in the same way. The corresponding encodings are enclosed in appendix A.6.4. When looking up rules in appendix substitute *pre* for $\preceq$ and *equ* for $=$ in names.

Rigidity and chop-freeness of equality of actions is asserted axiomatically (see appendix) and can be assumed in the following.

6.4.1 The Flexible Constant lTr

We introduce the local trace as a flexible term or special nullary function symbol:

$$\mathsf{lTr} \in \textbf{\textit{action list}}$$

As for Tr we associate the value of lTr with only the endpoint of the interval. We then capture monotony by the following axiomatic rules:

$$\frac{w : \mathsf{lTr} = acs \quad \mathsf{ri}(acs)}{w : \Box_l^c(\mathsf{lTr} = acs)}\ lTr\_mono1 \qquad \frac{w : \mathsf{lTr} = acs \quad \mathsf{ri}(acs)}{w : \Box_r(acs \preceq \mathsf{lTr})}\ lTr\_mono2$$

$$\frac{w : \mathsf{lTr} = acs \quad \mathsf{ri}(acs)}{w : \Box_l^c(\Box_l(\mathsf{lTr} \preceq acs))}\ lTr\_mono3$$

Again, these rules are sufficient to completely describe the monotony and all other cases, such as the following, are theorems of the logic:

$$\frac{w : \mathsf{lTr} = acs \quad \mathsf{ri}(acs)}{w : \boxdot(\mathsf{lTr} \preceq acs)}\ lTr\_mono5 \qquad \frac{w : \mathsf{lTr} = acs \quad \mathsf{ri}(acs)}{w : \Box_r(\Diamond_l^c(acs \preceq \mathsf{lTr}))}\ lTr\_mono6$$

The formal proofs of these theorems are enclosed in appendix A.6.4.1. The proofs of the two major assist lemmas regarding lTr and $\preceq$ shown below are also included in this appendix.

$$\frac{\mathsf{ri}(acs)}{w : acs \preceq \mathsf{lTr} \iff (\exists x \cdot \mathsf{lTr} = x \wedge acs \preceq x)}\ lTr\_\preceq lTr\_expand$$

$$\frac{\mathsf{ri}(acs)}{w : \mathsf{lTr} \preceq acs \iff (\exists x \cdot \mathsf{lTr} = x \wedge x \preceq acs)}\ lTr\_lTr\preceq\_expand$$

6.4.2 Predicates Concerning lTr

We also define useful predicates for lTr. These predicates are similar to the ones for the global trace but we must keep in mind that there is one local trace for every participating agent.

Occurrences of actions in the local traces we consider instantaneous and, thus, confined to point intervals. Stability properties can only be expressed for non-point intervals ($\ell > 0$) and exclude the interval endpoints.

The local trace of an agent, A, may be *empty*:

$$\begin{aligned} &\mathsf{lTr\_empty\ A} : \boldsymbol{agent} \rightarrow \boldsymbol{bool} \\ &\mathsf{lTr\_empty}\ A \mathrel{\hat{=}} \mathsf{lTr} = [] : \boldsymbol{bool} \end{aligned}$$

The local trace of an agent, A, may have a particular value, acs, *throughout* an interval:

$$\begin{aligned} &\mathsf{lTr\_throughout} : \boldsymbol{agent} \rightarrow (\boldsymbol{action\ list} \rightarrow \boldsymbol{bool}) \\ &\mathsf{lTr\_throughout}\ A\ acs \mathrel{\hat{=}} 0 < \ell \wedge \Box(\mathsf{lTr}\ A = acs) \end{aligned}$$

The existence of any such value, acs, implies local *stability* for A:

$$\begin{aligned} &\mathsf{lTr\_stable} : \boldsymbol{agent} \rightarrow \boldsymbol{bool} \\ &\mathsf{lTr\_stable}\ A \mathrel{\hat{=}} \exists acs : \boldsymbol{action\ list} \cdot \mathsf{lTr\_throughout}\ A\ acs \end{aligned}$$

When an agent A *performs* an internal action, a, it is after a period of stability. At this point the previously stable local trace is extended with the action a:

$$\begin{aligned} &\mathsf{lTr\_performs} : \boldsymbol{agent} \rightarrow (\boldsymbol{action} \rightarrow \boldsymbol{bool}) \\ &\mathsf{lTr\_performs}\ A\ a \mathrel{\hat{=}} \ell = 0 \wedge (\exists acs : \boldsymbol{action\ list} \cdot \\ &\qquad\qquad \mathsf{lTr}\ A = a \# acs \wedge \Diamond_l \mathsf{lTr\_throughout}\ A\ acs) \end{aligned}$$

The contrary may also be the case. At a particular moment, after a stable period, the agent, A, may remain *idle* and do nothing:

$$\begin{aligned} &\mathsf{lTr\_idles} : \boldsymbol{agent} \rightarrow \boldsymbol{bool} \\ &\mathsf{lTr\_idles}\ A \mathrel{\hat{=}} \ell = 0 \wedge \neg(\exists a \cdot \mathsf{lTr\_performs}\ A\ a) \end{aligned}$$

Axiomatic Rules Concerning Predicates

In the following we present a number of rules, regarding the use of these predicates, that are defined as axioms.

The notion of *finite variability* applies to the local traces, i.e. every interval is surrounded by stable periods:

$$\frac{}{(i,j) : \Diamond_l(\mathsf{lTr\_stable}\ A)}\ lTr\_lconst \qquad \frac{}{(i,j) : \Diamond_r(\mathsf{lTr\_stable}\ A)}\ lTr\_rconst$$

When an agent, A, idles in a particular point the local trace must have the same value in non-point intervals before and after the point.

$$\frac{(i,j) : \mathsf{lTr\_idles}\ A}{(i,j) : \ell = 0 \wedge \left(\begin{array}{c} \exists acs \cdot \mathsf{lTr} = acs \wedge \\ \Diamond_l(\mathsf{lTr\_throughout}\ A\ acs) \wedge \\ \Diamond_r(\mathsf{lTr\_throughout}\ A\ acs) \end{array}\right)}\ lTr\_idling$$

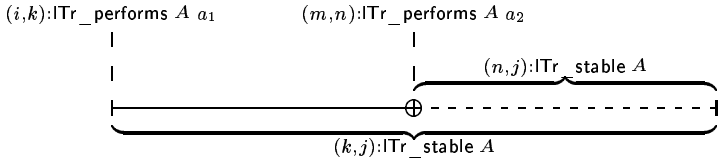
6.4.3 Lemmas and Theorems Regarding Predicates

Most of the rules presented in the following have been axiomatically asserted in the logic. We shall give informal arguments as to why they are sound.

Lemma 6.7. *"lTr_PS_PS_zlen"*

$$\frac{(i,k):\mathit{lTr\_performs}\ A\ a_1 \quad (m,n):\mathit{lTr\_performs}\ A\ a_2 \quad (k,j):\mathit{lTr\_stable}\ A \quad (n,j):\mathit{lTr\_stable}\ A}{(n,k):\ell=0}$$

Proof. Assume that the two points (i,k) and (m,n) are not the same, i.e. $(n,k) : \ell \neq 0$:

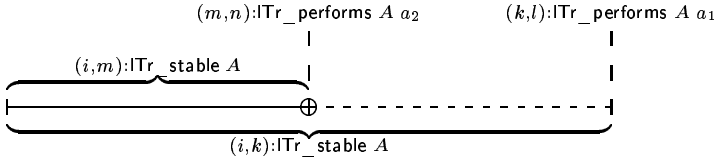


When A performs the action a_2 the stability of (k,j) is compromised and, thus, the assumption leads to contradiction. □

Lemma 6.8. *"lTr_SP_SP_zlen"*

$$\frac{(k,l):\mathit{lTr\_performs}\ A\ a_1 \quad (m,n):\mathit{lTr\_performs}\ A\ a_2 \quad (i,k):\mathit{lTr\_stable}\ A \quad (i,m):\mathit{lTr\_stable}\ A}{(n,k):\ell{=}0}$$

Proof. Assume that the two points (k,l) and (m,n) are not the same, i.e. $(n,k):\ell \neq 0$:



When A performs the action a_2 the stability of (i,k) is compromised and, thus, the assumption leads to contradiction. □

Lemma 6.9. *"lTr_E_P_fwd"*

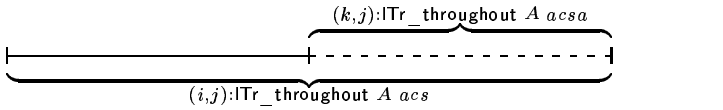
$$\frac{(i,k):\mathit{lTr\_empty}\ A \quad (l,j):\mathit{lTr\_performs}\ A\ e}{(k,l):\ell \geq 0}$$

Proof. Assuming (l,j) to be temporally before (k,k) contradicts our assumption about monotonicity of lTr. □

Lemma 6.10. *"lTr_overlap1"*

$$\frac{(i,j):\mathit{lTr\_throughout}\ A\ acs \quad (k,j):\mathit{lTr\_throughout}\ A\ acsa \quad \mathit{ri}(acs) \quad \mathit{ri}(acsa)}{(i,j):acs{=}acsa}$$

Proof. Assume that $acs \neq acsa$:



Since both of the intervals (i,j) and (k,j) have the property $\ell > 0$ there will be an interval of overlap. On this interval we have a contradiction. □

Lemma 6.11. *"lTr_overlap2"*

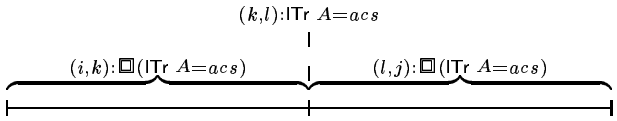
$$\frac{(i,j):\mathit{lTr\_throughout}\ A\ acs \quad (i,k):\mathit{lTr\_throughout}\ A\ acsa \quad \mathit{ri}(acs) \quad \mathit{ri}(acsa)}{(i,j):acs{=}acsa}$$

Proof. Similar to that of "lTr_overlap1" (6.3). □

Lemma 6.12. *"lTr_connected"*

$$\frac{(i,k){:}\boxdot(\mathit{lTr}\ A{=}acs) \quad (k,l){:}\ell{=}0 \quad (k,l){:}\mathit{lTr}\ A{=}acs \quad (l,j){:}\boxdot(\mathit{lTr}\ A{=}acs)}{(i,j){:}\boxdot(\mathit{lTr}\ A{=}acs)}$$

Proof.



When keeping in mind that lTr is associated with the endpoint only, it is intuitively clear from the drawing that there is no choice of proper subinterval of (i,j) such that $\mathsf{lTr} \neq acs$. □

Lemma 6.13. *"lTr_idling_imp_S"*

$$\frac{w : 0 < \ell \quad w : \mathit{lTr\_stable}\ A \frown \mathit{lTr\_idles}\ A \frown \mathit{lTr\_stable}\ A}{w : \mathit{lTr\_stable}\ A}$$

Proof. This proof has been conducted formally in Isabelle. It relies on "lTr_connected". See appendix A.6.4.3. □

6.4.4 Concerning Verification Functions

As already pointed out in chapter 5 we need to be able to somehow reason about the outcome of verification actions. We take advantage of the close connection to HOL and establish two important lemmas.

Lemma 6.14. *"lTr_P_Ver_True"*

$$\frac{w : \mathit{lTr\_performs}\ A\ (\mathit{Ver}\ k\ l\ P) \quad P}{w : \mathit{lTr\_performs}\ A\ (\mathit{Ver}\ k\ l\ True)}$$

Proof. The proof follows the structure of the formula. Notice that the unlabelled assumption is a HOL formula. See appendix A.6.4.4. □

Lemma 6.15. *"lTr_P_Ver_True"*

$$\frac{w : lTr\_performs\ A\ (Ver\ k\ l\ P) \quad \neg P}{w : lTr\_performs\ A\ (Ver\ k\ l\ False)}$$

Proof. The proof follows the structure of the formula. The unlabelled assumption is a HOL formula. See appendix A.6.4.4. □

6.5 Modeling Heuristics

We have now established a theoretical basis that allows us to express certain timed properties of traces in NLDCHOL. This theory describes a set of modeling primitives and an, admittedly limited, number of inference rules that may be used to derive consequences from models. In this section we will give some suggestions as to how these modeling primitives might be used to describe properties of protocols.

The aim of modeling is to relate the timings of events and local actions in a manner that faithfully represents their causal ordering. To establish such models we need a number of auxiliary functions, the existence of which we will assume throughout the following.

The message verification action cost function:

$$\delta : \mathbb{N} \to (\mathbb{N} \to \mathbb{R})$$

associates a cost in terms of time represented by a real number with a given verification action, e.g. the jth, of a given step, e.g. the ith.

The message preparation cost function:

$$\Delta : \mathbb{N} \to \mathbb{R}$$

associates a cost in terms of time represented by a real number with the preparation of the message to be send in a given step, e.g. the ith.

A number of verification functions:

$$\mathsf{A}_{i_j} : \textit{'a} \to \textit{bool}$$

return True if the message item to be verified is acceptable and return False otherwise.

6.5.1 Dependency versus Obligation

What faithfulness means for individual formulas depends on the properties we want to express. Not all properties can be expressed using the same kind of formulas.

Right-to-Left Modeling

In some situations certain events *may* occur if certain events or actions have been caused or performed in the past. We refer to this as *dependency* or *right-to-left* modeling.

For example, it is rather impossible to receive a message that was not send in the past:

$$\frac{w : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ (\!|(B,B) \rightarrow (A,A) : m_i|\!))}{w : \Diamond_l(\mathsf{Tr\_causes}\ B\ (\mathsf{Snd}\ B\ (\!|(B,B) \rightarrow (A,A) : m_i|\!)) \frown \mathsf{True})}$$

Also, the message can not be received until we are ready to do so:

$$\frac{w : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ (\!|(B,B) \rightarrow (A,A) : m_i|\!))}{w : \Diamond_l(\mathsf{lTr\_performs}\ A\ (\mathsf{RState}\ i) \frown \mathsf{lTr\_stable}\ A)}$$

Left-to-Right Modeling

In other situations, however, certain events *must* occur as a future result if certain events (or actions) take place now. We refer to this as *Obligation* or *left-to-right* modeling.

Again, the reception of an event is a good example. If a packet is received it must be processed:

$$\frac{w : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ (\!|(B,B) \rightarrow (A,A) : m_i|\!))}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \delta\ i\ 1) \frown \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ i\ 1\ \mathsf{A}_{i_1}(m_i)))}$$

Here $\delta\ i\ 1$ is the cost of the first verification action of the ith step and A_{i_1} is an appropriately formulated verification function.

6.5.2 Universal versus Specific Properties

Dependency and Obligation modeling seem to be the basic building blocks from which we build our protocols models. The question of *what to model* still remains, though, and in the following we try to give some guidelines.

Universal Properties

There seem to be a set of properties that are universal to all protocols. We will formalize some of these. The first two of the following are theorems of the established logic. The rest we assert as axioms.

Internal actions take time and agents can only perform one at a time:

Theorem 6.1. *"lTr_P_uniq"*

$$\frac{w : \mathit{lTr\_performs}\ A\ a_1 \quad w : \mathit{lTr\_performs}\ A\ a_2}{w : a_1 = a_2}$$

Proof. See appendix A.6.5. □

Causing events take time and agents can only cause one at a time:

Theorem 6.2. *"Tr_C_uniq"*

$$\frac{w : \mathit{Tr\_causes}\ A\ e_1 \quad w : \mathit{Tr\_causes}\ A\ e_2}{w : e_1 = e_2}$$

Proof. See appendix A.6.5. □

At some point in time the system is initialized and the global trace is empty:

$$\frac{}{w : \Diamond_a(\mathsf{Tr\_empty})}\ {}^{Tr\_init}$$

For each agent there is a point in time where he is initialized and the local trace is empty:

$$\frac{}{w : \Diamond_a(\mathsf{lTr\_empty}\ A)}\ {}^{lTr\_init}$$

All agents become ready to either send or receive at some point after initialization:

$$\frac{w : \mathsf{lTr\_empty}\ A}{w : \Diamond_r(\mathsf{lTr\_stable}\ A \frown \mathsf{lTr\_performs}\ A\ \mathsf{Ready})}\ lTr\_Ready$$

If an agent assumes the role of *initiator* during an execution he must have been ready to do so:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ 1)}{w : \Diamond_l(\mathsf{lTr\_performs}\ A\ \mathsf{Ready} \frown \mathsf{lTr\_stable}\ A)}\ Initiate$$

If an agent assumes the role of *responder* during an execution he must have been ready to do so:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{RState}\ 1)}{w : \Diamond_l(\mathsf{lTr\_performs}\ A\ \mathsf{Ready} \frown \mathsf{lTr\_stable}\ A)}\ Respond$$

After aborting a protocol execution agents become ready for a new execution within a maximum time frame of ϵ:

$$\frac{\mathsf{lTr\_performs}\ A\ \mathsf{Abort}}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell \leq \epsilon) \frown \mathsf{lTr\_performs}\ A\ \mathsf{Ready})}\ Abort$$

After accepting a protocol execution agents become ready for a new execution within a maximum time frame of ϵ:

$$\frac{\mathsf{lTr\_performs}\ A\ \mathsf{Accept}}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell \leq \epsilon) \frown \mathsf{lTr\_performs}\ A\ \mathsf{Ready})}\ Accept$$

Specific Properties

In the following we establish some guidelines as to what aspects of given protocols should be formalized - and how. The following rules can be seen as abstract schemas of which concrete instances are to be modeled for actual protocols.

Senders If an agent, A, is the sender in the ith step of the protocol then A must send the corresponding packet immediately after preparing it:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ i)}{\begin{array}{l} w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \Delta\ i) \frown \\ \mathsf{Tr\_causes}\ A\ (\mathsf{Snd}\ A\ (\!|(A,A) \rightarrow (B,B) : m_i|\!))) \end{array}}\ Senderi$$

At the same time A must prepare to receive the message of the following step:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ i)}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \Delta\ i) \frown \mathsf{lTr\_performs}\ A\ (\mathsf{RState}\ (i+1)))}\ Senderi$$

If, however, i is the last protocol step, n, the agent should just accept the run:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ n)}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \Delta\ n) \frown \mathsf{lTr\_performs}\ A\ \mathsf{Accept})}\ Sendern$$

Receivers If, on the other hand, the agent, A, is the receiver in the ith step, message reception requires that the message has actually been sent:

$$\frac{w : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ (\!|(B,B) \rightarrow (A,A) : m_i|\!))}{w : \Diamond_l(\mathsf{Tr\_causes}\ B\ (\mathsf{Snd}\ B\ (\!|(B,B) \rightarrow (A,A) : m_i|\!)) \frown \mathsf{True})}\ receiveri$$

The message can not be received until A is ready to do so:

$$\frac{w : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ (\!|(B,B) \rightarrow (A,A) : m_i|\!))}{w : \Diamond_l(\mathsf{lTr\_performs}\ A\ (\mathsf{RState}\ i) \frown \mathsf{lTr\_stable}\ A)}\ receiveri$$

When the message is received A must start validating it immediately:

$$\frac{w : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ (\!|(B,B) \rightarrow (A,A) : m_i|\!))}{\begin{array}{l} w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \delta\ i\ 1) \frown \\ \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ i\ 1\ \mathsf{A}_{i_1}(m_i))) \end{array}}\ receiveri$$

If the jth verification function of the ith step is completed and the outcome is True then the next verification must commence:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ i\ j\ \mathsf{True})}{\begin{array}{l} w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \delta\ i\ (j+1)) \frown \\ \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ i\ (j+1)\ \mathsf{A}_{i_{j+1}}(m_i))) \end{array}}\ Verij$$

If the jth is the last verification function, the k_ith, of the ith step and the outcome is True then the message of step $i+1$ must be prepared:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ i\ k_i\ \mathsf{True})}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell \leq \epsilon) \frown \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ (i+1)))}\ SuccVeri$$

If, furthermore, the ith step is the last of the protocol, n, then A must accept the execution:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ n\ k_n\ \mathsf{True})}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell \leq \epsilon) \frown \mathsf{lTr\_performs}\ A\ (\mathsf{Accept}))}\ SuccVern$$

If the agent, A, completes any verification function with the result False then A must abort the execution:

$$\frac{w : \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ i\ j\ \mathsf{False})}{w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell \leq \epsilon) \frown \mathsf{lTr\_performs}\ A\ \mathsf{Abort})}\ FailVer$$

CHAPTER 7

Examples: Simple Protocols

In order to evaluate the established theory it must be used. We apply the timed modeling to two small examples in order to gain the necessary insights.

As for the untimed theory a couple of examples have been developed. The examples of this chapter are going to be substantially simpler than those of chapter 4. Due to space constraints we will use a notation similar to that of section 4.1 to present the formal models.

The examples consider the simple pseudo-protocol shown at the beginning of section 2.1 but only with two or three message exchanges[1]. The Alice-and-Bob specification is as follows:

1. $A \rightarrow B : M_1$
2. $B \rightarrow A : M_2$
3. $A \rightarrow B : M_3$

The initiator sends the message M_1 to the responder who in turn responds with the message M_2 etc.

In spite of the apparent simplicity of this 'protocol' the modeling of it and subsequent reasoning about properties still reveal useful information about our proposed method.

The work presented here is new and the corresponding Isabelle/HOL encoding is anclosed in appendix A.7.

[1] The experiments with mutual inductive definitions mentioned in chapter 6 have also been conducted on this setup.

7.1 Network Delays

The first example is the simplest one. The intention is to argue about the delays that are caused by the network.

7.1.1 Creating a Model

For this purpose we develop a model that capture the causal dependencies between events. Furthermore, we assume two types of delay that must also be modeled[2]. When an agent receives a message he confirms this by blocking it after a delay no longer than:

$$\epsilon : \mathbb{R}$$

When an agent sends a message via the network the intended receiver will be able to receive it after a delay no shorter than

$$\delta : \mathbb{R}$$

We ignore the internal actions and develop the following purely right-to-left model:

$$\begin{array}{ll}
\texttt{Rcv} & [\![w : \mathsf{Tr_occurs}\ (\mathsf{Rcv}\ B\ (\!|(A,A) \to (B,B) : m|\!));\ A \neq B]\!] \Longrightarrow \\
& \quad w : \Diamond_l((\mathsf{Tr_occurs}\ (\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : m|\!)) \wedge \\
& \qquad \mathsf{Tr_quiet}\ B)^\frown(\mathsf{Tr_stable}\ A\ \wedge\ \mathsf{Tr_stable}\ B))
\end{array}$$

$$\begin{array}{ll}
\texttt{Blk} & [\![w : \mathsf{Tr_occurs}\ (\mathsf{Blk}\ B\ (\!|(A,A) \to (B,B) : m|\!));\ A \neq B]\!] \Longrightarrow \\
& \quad w : \Diamond_l((\mathsf{Tr_occurs}\ (\mathsf{Rcv}\ B\ (\!|(A,A) \to (B,B) : m|\!)) \wedge \\
& \qquad \mathsf{Tr_quiet}\ A)^\frown(\ell \leq \varepsilon\ \wedge\ \mathsf{Tr_stable}\ A\ \wedge\ \mathsf{Tr_stable}\ B))
\end{array}$$

$$\begin{array}{ll}
\texttt{Snd1} & [\![w : \mathsf{Tr_occurs}\ (\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \mathsf{M1}|\!));\ A \neq B]\!] \Longrightarrow \\
& \quad w : \Diamond_l(\mathsf{Tr_empty}) \vee \\
& \qquad \Diamond_l((\mathsf{Tr_occurs}\ (\mathsf{Snd}\ A\ (\!|(A,A) \to (C,C) : \mathsf{M3}|\!)) \wedge \\
& \qquad\quad \mathsf{Tr_quiet}\ C)^\frown(\mathsf{Tr_stable}\ A\ \wedge\ \mathsf{Tr_stable}\ C)) \vee \\
& \qquad \Diamond_l((\mathsf{Tr_occurs}\ (\mathsf{Blk}\ A\ (\!|(C,C) \to (A,A) : \mathsf{M3}|\!)) \wedge \\
& \qquad\quad \mathsf{Tr_quiet}\ C)^\frown(\mathsf{Tr_stable}\ A\ \wedge\ \mathsf{Tr_stable}\ C))
\end{array}$$

[2]This use of δ and ϵ is non-standard with respect to the method described in section 6.5

$$\texttt{Snd2}\ [\![w : \mathsf{Tr\_occurs}\ (\mathsf{Snd}\ B\ (\!|(B,B) \to (A,A) : \mathsf{M2}|\!));\ A \neq B]\!] \Longrightarrow$$
$$w : \Diamond_l((\mathsf{Tr\_occurs}\ (\mathsf{Blk}\ B\ (\!|(A,A) \to (B,B) : \mathsf{M1}|\!)) \wedge$$
$$\mathsf{Tr\_quiet}\ A)^\frown(\mathsf{Tr\_stable}\ B\ \wedge\ \mathsf{Tr\_stable}\ A))$$

$$\texttt{Snd3}\ [\![w : \mathsf{Tr\_occurs}\ (\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \mathsf{M3}|\!));\ A \neq B]\!] \Longrightarrow$$
$$w : \Diamond_l((\mathsf{Tr\_occurs}\ (\mathsf{Blk}\ A\ (\!|(B,B) \to (A,A) : \mathsf{M2}|\!)) \wedge$$
$$\mathsf{Tr\_quiet}\ B)^\frown(\mathsf{Tr\_stable}\ A\ \wedge\ \mathsf{Tr\_stable}\ B))$$

$$\texttt{delay}\ [\![w : (\mathsf{Tr\_occurs}\ (\mathsf{Snd}\ A\ P)^\frown \mathsf{True}^\frown \mathsf{Tr\_occurs}\ (\mathsf{Rcv}\ B\ P));$$
$$A \neq B]\!] \Longrightarrow w : \delta \leq \ell$$

Besides the rules shown the rules for universal properties are also valid for this protocol. Most of those rules are not used, however, as we focus solely on the externally visible events.

7.1.2 Proving Properties

Given the above model we would like to prove some simple properties regarding the protocol. We state the properties in order of the difficulty of proof.

Lemma 7.1. *"M2_imp_M1"*
If the message M2 is sent by any participant B then the message M1 must have been sent from a different participant, A, to B at some point in the past.

$$\frac{A \neq B \quad w : \mathit{Tr\_occurs}\ (\mathit{SndA}(\!|(A,A) \to (B,B) : \mathit{M2}|\!))}{w : \Diamond_l(\mathit{Tr\_occurs}(\mathit{SndB}(\!|(B,B) \to (A,A) : \mathit{M1}|\!))^\frown \mathit{True})}$$

Proof. This is fairly easy to show. The proof follows the structure of the formula but most reasoning is forward, i.e. uses forward or destruction rules to derive consequences from the premises. See formal proof-script in appendix A.7.1.2. □

Lemma 7.2. *"E_Rcv_delay"*
From the time when the trace is empty until a message is received by some agent B a time period of at least δ must have gone by:

$$\frac{A{\neq}B \quad \mathit{ri}(\delta) \quad w{:}\mathit{Tr\_empty}^\frown \mathit{True}^\frown \mathit{Tr\_occurs}(\mathit{Rcv}\ B\ (\!|(A,A){\to}(B,B){:}\mathit{M1}|\!))}{w{:}\delta{\leq}\ell}$$

Proof. This proof is also reasonably easy. By forward reasoning from the assumption we derive the trace right-to-left. When we reach the point where the message $M1$ is sent we have a minimal derivation and enough information to derive $\ell \geq \delta$. We stop and use the rule "Tr_E_O_fwd" to conclude that this point must be inside w. The delay follows. See formal proof-script in appendix A.7.1.2. □

Lemma 7.3. *"snd1_snd2_imp_delay"*
Assume that two agents A and B, with rigid identities, are executing the protocol. Between the occurence of "A sends M1 to B" and the occurence of "B sends M2 to A" a time period of at least δ passes by.

$$\frac{A \neq B \quad ri(A) \quad ri(B) \quad \begin{array}{l} w{:}\mathsf{Tr\_causes}\ A\ (\mathsf{Snd}\ A\ (\!|(A,A)\rightarrow(B,B){:}M1|\!))\ ^\frown \\ \mathsf{Tr\_stable}\ A\ ^\frown\ \mathsf{Tr\_causes}\ B\ (\mathsf{Snd}\ B\ (\!|(B,B)\rightarrow(A,A){:}M2|\!)) \end{array}}{w{:}\delta \leq \ell}$$

Proof. From the formulation one would think that the proof of this lemma is also easy. This is not the case. The difficulty arises from the fact that we do not only have to reason right-to-left in order to find the $\mathsf{Snd}\ A\ (\!|(A,A) \rightarrow (B,B) : M1|\!)$ event but we also have to prove that it occurs inside the interval w.

The proof relies on the lemmas "Tr_CS_CS_equ" and "Tr_silent_imp_S". Furthermore, the model needs to be extremely explicit about the fact that when A and B do not explicitly cause events they explicitly do not[3]. The proof can then be carried out by right-to-left constructing a trace, then decomposing the trace completely, and finally reassembling it in chunks that allow delay rule application *and* matching of endpoints. A rather lengthy and complicated affair - see appendix A.7.1.2. □

7.2 Rejecting Bogus Messages

In this section we present a slightly more comprehensive example. The intention of this example is to argue about the rejection of bogus messages.

[3]This is how the need for the Tr_quiet predicate arose

7.2.1 Creating a Model

For this purpose we need a model that is more complete with respect to the guidelines of section 6.5. The modeled protocol is the same as that in the last example but with only two message exchanges.

As proposed we assume a function:

$$\Delta : \mathbb{N} \rightarrow \mathbb{R}$$

that associates every message preparation event with a cost. These costs are predefined and therefore rigid:

$$\frac{}{\mathsf{ri}(\Delta\ i)}\ RIDelta$$

A similar function associates the individual verification actions with a predetermined fixed cost:

$$\delta : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R})$$

Costs defined in this way are also rigid:

$$\frac{}{\mathsf{ri}(\delta\ i\ j)}\ RIdelta$$

The minimum delay, which occurs when the internal state changes without involving larger computations, is:

$$\epsilon : \mathbb{R}$$

Which is also rigid:

$$\frac{}{\mathsf{ri}(\epsilon)}\ RIepsilon$$

We must also define an appropriate verification function for each verification action that the modeled protocol contains. We assume that there is only one verification associated with each step and that the corresponding verification function is the chkAgent function defined in section 5.2.1.

As a convenience we also define the message as a function of the step:

$$\mathsf{M} : \mathbb{N} \rightarrow \boldsymbol{msg}$$

We take the universal properties for granted and extend the model with the following rules:

$$\texttt{SState1}\quad [\![w : \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ 1);\ A \neq B]\!] \Longrightarrow$$
$$w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \Delta(1))^\frown$$
$$\mathsf{Tr\_causes}\ A\ (\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \mathsf{M}\ 1|\!))) \wedge$$
$$\Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \Delta(1))^\frown$$
$$\mathsf{lTr\_performs}\ A\ (\mathsf{RState}\ 2))$$

$$\texttt{SState2}\quad [\![w : \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ 2);\ A \neq B]\!] \Longrightarrow$$
$$w : \Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \Delta(2))^\frown$$
$$\mathsf{Tr\_occurs}(\mathsf{Snd}\ A\ (\!|(A,A) \to (B,B) : \mathsf{M}\ 2|\!))) \wedge$$
$$\Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \Delta(2))^\frown$$
$$\mathsf{lTr\_performs}\ A\ \mathsf{Accept})$$

$$\texttt{SuccVer1}\quad [\![w : \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ 1\ 1\ \mathsf{True})\ ]\!] \Longrightarrow$$
$$w : \Diamond_r(\mathsf{lTr\_stable}\ A^\frown \mathsf{lTr\_performs}\ A\ (\mathsf{SState}\ 2))$$

$$\texttt{SuccVer2}\quad [\![w : \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ 2\ 1\ \mathsf{True})\ ]\!] \Longrightarrow$$
$$w : \Diamond_r(\mathsf{lTr\_stable}\ A^\frown \mathsf{lTr\_performs}\ A\ \mathsf{Accept})$$

$$\texttt{FailVer}\quad [\![w : \mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ k\ l\ \mathsf{False})]\!] \Longrightarrow$$
$$w : \Diamond_r(\mathsf{lTr\_stable}\ A^\frown \mathsf{lTr\_performs}\ A\ \mathsf{Abort})$$

$$\texttt{Rcv}\quad [\![w : \mathsf{Tr\_causes}\ A\ (\mathsf{Rcv}\ A\ (\!|(B,B) \to (A,A) : \mathsf{M}\ k|\!));\ A \neq B]\!] \Longrightarrow$$
$$w : \Diamond_l(\mathsf{lTr\_performs}\ A\ (\mathsf{RState}\ k)^\frown \mathsf{lTr\_stable}\ A) \wedge$$
$$\Diamond_l((\mathsf{Tr\_causes}\ B\ (\mathsf{Snd}\ B\ (\!|(B,B) \to (A,A) : \mathsf{M}\ k|\!)) \wedge$$
$$\mathsf{Tr\_quiet}\ A)^\frown(\mathsf{Tr\_stable}\ A \wedge \mathsf{Tr\_stable}\ B)) \wedge$$
$$\Diamond_r((\mathsf{lTr\_stable}\ A \wedge \ell = \delta(k)(1))^\frown$$
$$(\mathsf{lTr\_performs}\ A\ (\mathsf{Ver}\ k\ 1\ (\mathsf{chkAgent}\ B))))$$

The rules have been constructed according to the guidelines of section 6.5. Not all of the prescribed rules have been necessary to define and some have been combined in order to keep the model size small. As an example the `Rcv` rule as defined above can be used for both right-to-left and left-to-right reasoning.

7.2.2 Proving Properties

We shall now present some results regarding bogus messages that may be derived from the model. The following properties are presented in order of difficulty of proof.

Theorem 7.1. *"Rcv_bogus_imp_delay_Abort"*

$$\frac{w : \mathit{Tr\_causes}\ A\ (\mathit{Rcv}\ A\ (\!|(\mathit{Spy}, \mathit{Spy}) \to (A, A) : \mathit{M}\ 1|\!)) \quad A \neq \mathit{Spy}}{w : \Diamond_r(\ell = (\delta\ 1\ 1) + \epsilon \frown \mathit{lTr\_performs}\ A\ \mathit{Abort})}$$

Proof. This result is easy to show since the reasoning follows the structure of the formula. By forward reasoning from the premises we can construct a sequence of events similar to that of the conclusion. The result then follows more or less directly by unification. See formal proof-script in appendix A.7.2.2. □

Theorem 7.2. *"Rcv_bogus_imp_no_Accept"*

$$\frac{\begin{array}{r} w : \mathit{Tr\_causes}\ A\ (\mathit{Rcv}\ A\ (\!|(\mathit{Spy}, \mathit{Spy}) \to (A, A) : \mathit{M}\ 1|\!)) \frown \\ \mathit{lTr\_stable}\ A \frown \mathit{lTr\_performs}\ A\ (\mathit{Ver}\ 1\ 1\ \mathit{True}) \end{array} \quad A \neq \mathit{Spy}}{w : \mathit{False}}$$

Proof. This proof is more difficult. Our only option is to reason left-to-right from the premises until a lTr_performs A (Ver 1 1 False) action is found. Once this happens we have to prove that it necessarily occurs in the same point as the lTr_performs A (Ver 1 1 True) action. As for events this requires us to model non-activity explicitly and then rely on "lTr_SP_SP_zlen" in order force a match. Formal proof-script can be found in appendix A.7.2.2. □

CHAPTER 8

Conclusion

8.1 Executive Summary

Most of this thesis has concerned the development of a combination of Paulson's Inductive Approach to security protocol verification and a Labelled Neighborhood Logic derived from Rasmussen's Labelled Signed Interval Logic. This combination has been conducted in a way as to support Meadows' ideas about a cost-based framework for evaluation of Denial of Service issues.

We began by investigating the domain of protocol security. Taking the view that protocols are programs executed by peers in peer networks we motivated and justified the choice of *packets*, $(a_s, l_s) \rightarrow (a_r, s_r) : M$, as the primitives of communication and a particular set of *events*, $\{\mathsf{Snd}, \mathsf{Blk}, \mathsf{Rcv}\}$, as the primitives of network activity. In the face of certain threats from intruders we characterized a number of important security goals. We noted that Paulson's inductive approach is strong when regarding untimed properties of protocols but unable to analyze timed properties such as availability or fail-stop criteria. We then considered Meadows' idea of an availability analysis based on a cost-dependent fail-stop criteria and models that include internal actions.

Rooted in Paulson's untimed theories of agents, keys, and messages we then went on to (re)build an inductive framework based on the novel notion of packets and the redefined notion of events. When doing this we eliminated Paulson's pervasive assumption of a strong intruder (Dolev-Yao) and allowed for more differentiated intruder modeling. The novel notions of ideal protocols and realistic protocols with simple projections $\downarrow$ and ideal projections $\uparrow$ allowed us to define relations, "is consistent for", $\triangle$, and "contains

a successful active attack on", ∇, between traces and agents. Notably, ∇, is the key to distinguishing active attacks.

After defining this extended untimed theory we formalized the Station-to-Station protocol in order to test and illustrate the novel theoretic developments. We showed three small results:

1. The ideal model of ***StS*** is indeed a restriction of the well-formed network traces ***trace***.
2. Repetitions of the StS protocol are consistent, $\triangle$, with respect to the ideal model.
3. An active attack, first shown by Lowe, is indeed recognized as a successful active attack, ∇, by the theory.

From these examples we moved on to formalize Meadows' internal actions in an untimed setting. Here we first showed how a smaller and different set of actions, $\{\mathsf{Ready}, \mathsf{Ver}, \mathsf{SState}, \mathsf{RState}, \mathsf{Abort}, \mathsf{Accept}\}$, are sufficient for our purposes. We then described the parameterized inductive definitions that define the set of all well-formed local traces for given agents. In order to illustrate this development we defined a concrete model of a simple two-message protocol with a simple verification function. We showed two simple properties:

1. Verification actions work - in the sense that only an abort action can follow the verification of a 'bad' message.
2. Possibility - there are sequences that end with an accept action.

We then identified the need of an adequate timed formalism for describing both timing issues and causal relationships between global events and local actions. In response to this we introduced NLDCHOL, a labelled Neighborhood Logic that extends HOL, also within the Isabelle/HOL environment. We extended NLDCHOL with partial ordering (causality), $\preceq$, and equality, $=$, for lists. These relations allowed us to introduce the notions of timed global traces, Tr, and timed local traces, lTr, as implicit monotonous functions of time. A number of important and convenient modeling primitives were presented and some basic properties of them shown. Finally an approach for the succinct modeling of real-time properties of security protocols was presented.

The timed theory was tested in practice. We first treated a dependency-driven model of a small pseudo-protocol. It was easy to show that one event was causally before another. It was reasonably easy to verify the existence of a minimal delay between system initialization the sending of the second

message by any agent. It was substantially harder to show a minimal delay between the occurrences of two particular events on the network. This proof required the model to be very explicit.

Subsequently, we treated a more complete model of a similar small pseudo-protocol. Here we modeled both dependency and obligation, and included both global events and local actions. It was easy to show that when an agent received a message from a known intruder he would abort the protocol run after a given delay. It was somewhat harder to show that a message from a known intruder would never be accepted by an agent.

8.2 Discussion

8.2.1 Technical Evaluation

On Global Traces

Like Paulson's original notion of inductive traces our derived one is based on the notion of events. We introduce, however, a complication in the form of packets. This is not a problem as the information carried by packets is a superset of that carried by Paulson's events.

The real change is introduced with our notion of events. The introduced set of events seems sensible, given the fact that we would like to model attackers of widely varying capabilities. However, since the information in packets are fully sufficient to identify the source of a given event the notion of events can be simplified.

Paulson uses a special event, Notes, to model situations in which the intruder gains possession of a session key by accident. Since no tests have been conducted regarding untimed verification of resistance against passive attacks it is unclear whether our setup would also benefit from such an event.

Motivated by the desire of complete modeling control over the spy behavior we completely removed the assumption of the spy seeing everything on the network when redefining the notion of knowledge. However, this is unnecessary as we are only concerned with explicit modeling control of active attacks.

The notion of ideal traces seem to be overly complicated. The ideal traces are meant to be used as references for comparing an actual run to the protocol schema as defined by the Alice-and-Bob specification. This motivates the thought that they could possibly be defined directly in this form. Well-formed traces, however, depend on a notion of visibility, which ensures that messages are only available on the network until blocked. This can only be defined on the event level and it is unlikely that a more abstract definition could capture this.

The defined relation, ∇, for identifying active attacks is adequate in the sense that all active attacks are indeed classified as such. It does, however, also seem to be a bit too general. A sequence where an intruder simply forwards a message, like a router, would be classified as an active attack.

On local traces

In chapter 5 we conjectured the adequacy of the actions the we use to build our local traces. Essentially, we chose to ignore the actual form of the individual actions performed to prepare a given message. The justification of this was that these actions have no effect on any protocol execution except for incurring a cost in terms of computation time. This is correct but if modeling an intruder, that leaves some of these actions out in order to safe time when mounting a DoS attack, explicitly modeling these actions could be useful.

On Timed Modeling

As the examples of chapter 7 shows the modeling approach is very reliant on explicitness. In fact, none of the examples really show how explicit a full model would need to be, but they still reveal that modeling of e.g. protocol interleaving is not straightforward.

Basing the timed developments directly on Neighborhood Logic and not a logic extended with Duration Calculus is a crucial choice in several ways. With this choice we have effectively given up the ability to express scheduling issues. This means that the description of e.g. multi-threaded servers are out of the question. However, defining the traces as flexible implicit (nullary) function symbols that depend monotonously on the intervals has the advantage of being conceptually simple.

Importantly, due to the close relationship between HOL and NLDCHOL described in section 6.1.3 it is possible, under suitable assumptions of rigidity and chop-freeness, to switch the traces between a timed and the untimed context (NLDCHOL and HOL). This might very well be a strong-point of the entire theoretical framework and we can only regret that it is probably the least explored part of the setup.

Further exploration of this may also be the key to establishing a timed notion of *visibility*. In the present work, the absence of such a notion is a distinct and undesirable difference between the timed and untimed theory.

In the existing theory we have only considered rigidity and chop-freeness issues when absolutely necessary. It is likely that we, in order to treat realistic protocols or for any other reason take advantage of the connection to the HOL based theory, will need to define these notions recursively over all of the defined structures.

Taking 'shortcuts' and defining alleged lemmas as axioms the way it is done in sections 6.3.3 and 6.4.3 is clearly not satisfactory. Introducing advanced ruled like these into the axiomatic foundation of any logic significantly increases the risk of creating an inconsistent logic. This should be remedied as soon as possible.

8.2.2 Conclusion

The initial goal of this thesis was to combine an existing untimed approach to protocol verification with an interval logic in order to facilitate analysis of availability issues. Mechanical proof support was also a goal.

As described we have identified concepts that seem to be central to availability issues and created a theoretical framework that focuses on these concepts. Proof support for the developed theory has been implemented in Isabelle/NLDCHOL, the Isabelle/HOL based proof tool for labeled neighborhood logic.

We have not shown strong evidence that the achieved combination of theories constitutes a 'strong' tool for reasoning about real-time security properties. And, clearly, the developed theory has some issues. But the approach does seem interesting and potentially useful. We do not hesitate to recommend further investigation.

Overall, the project has been successful.

8.3 Future Work

In this final section of the thesis we will shortly consider possible directions for future work.

What seems to have a high priority is to consider some case studies in order to get a better understanding of the usefulness of the outlined approach, also from a more pragmatic point of view. In [MI99] Matsuura and Imai proposes DoS protected variants of authenticated key agreement protocols[1] that could be a starting point for such a study. A later more advanced study could treat the deployed IPSEC - OAKLEY protocol [NWGH98], of which an unfinished description can be found in appendix C.

As mentioned the theory as presented here has a few inadequacies that should be resolved.

- The theory of events should be simplified.
- The proof-theoretical basis should be brought in order and all proposed lemmas either proved or dismissed.
- It should be clarified whether the notion of ideal traces can be simplified or not.
- The "is an active attack on" relation ∇ should be investigated a bit further in order to clarify its usefulness.

In a more long-term perspective, the following topics would be interesting to consider:

- A more advanced timed theory based on Duration Calculus.
- More groundwork in order to improve proof support.
- How to take full advantage of the relationship between the timed and untimed theories.

[1] The Station-to-Station protocol mentioned in this thesis is one such protocol.

Bibliography

[AN96] M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[AT91] Martín Abadi and Mark R. Tuttle. A semantics for a logic of authentication (extended abstract). In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 201–216, 1991.

[BAN89] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. In *Procedings of the Royal Society of London*, volume 426, pages 233–271, 1989.

[Bel00] Giampaolo Bella. *Inductive Verification of Cryptographic Protocols*. PhD thesis, Clare College - University of Cambridge, March 2000.

[BM93] Colin Boyd and Wenbo Mao. On a limitation of ban logic. In *Advances in Cryptology - Proceedings of EUROCRYPT 93*, Lecture Notes in Computer Science, pages 240–247. Springer-Verlag, 1993.

[BMV97] David Basin, Seán Matthews, and Luca Viganó. Labelled propositional modal logics: Theory and practice. *Journal of Logic and Computation*, 7(6):685–717, 1997.

[BP97] Giampaolo Bella and Lawrence C. Paulson. Using isabelle to prove properties of the kerberos authentication system. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 3-5 1997.

[BP98a] Giampaolo Bella and Lawrence C. Paulson. Kerberos version iv: inductive analysis of the secrecy goals. In Jean-Jacques Quisquater et al., editor, *Computer Security – ES-*

ORICS 98, number 1485 in LNCS, pages 361–375. Springer, 1998.

[BP98b] Giampaolo Bella and Lawrence C. Paulson. Mechanising ban kerberos by the inductive method. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer-Aided Verification: CAV '98*, number 1427 in LNCS, pages 416–427, 1998.

[Bri00] Neil Briscoe. Understanding the osi 7-layer model. *PC Network Advisor*, (120):13, July 2000.

[CH98] Zhou Chaochen and Michael R. Hansen. An adequate first order interval logic. In *Compositionality: The Significant Difference (COMPOS'97)*, volume 1536 of *Lecture Notes in Computer Science*, pages 581–608. Springer-Verlag, 1998.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentification and authenticated key exchanges. In *Designs, Codes and Cryptography*, volume 2, pages 107–125. Kluwer Academic Publishers, 1992.

[DY83] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Resoning about belief in cryptographic protocols. In *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, 1990.

[Gol99] Dieter Gollmann. *Computer Security*. Wiley, February 1999.

[GS95] Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 44–55, 1995.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Low96] Gavin Lowe. Some new attacks upon security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society, June 1996.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997.

[Mea99] Catherine Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the 12th Computer Security Foundations Workshop*. IEEE, 1999.

[Mea01] Catherine Meadows. A cost-based framework for analysis of denial of service networks. *Journal of Computer Security*, 9(1/2):143–164, 2001.

[MI99] K. Matsuura and H. Imai. Protection of authenticated key-agreement protocol against a denial-of-service attack. *Cientifica*, 2(11):15–19, 1999.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A proof Assistant for Higher-Order Logic*. Springer-Verlag, 2002.

[NWGH98] University of Arizona Network Working Group: H.Orman, Department of Computer Science. Request for comments 2412: The oakley key determination protocol. Internet RFC, November 1998.

[Pau91] Lawrence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[Pau99] Lawrence C. Paulson. Proving security protocols correct. *IEEE Symposium on Logic in Computer Science*, 1999.

[Ras01a] Thomas M. Rasmussen. Automated proof support for interval logics. In *LPAR 2001*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 317–326. Springer-Verlag, 2001.

[Ras01b] Thomas M. Rasmussen. Labelled natural deduction for interval logics. In *Computer Science Logic CSL'01*, volume 2142 of *Lecture Notes in Computer Science*, pages 308–323. Springer-Verlag, 2001.

[Ras02] Thomas Marthedal Rasmussen. *Interval Logic - Proof Theory and Theorem Prooving*. PhD thesis, IMM, Technical University of Denmark, 2002.

[Rya96] Peter Y. A. Ryan. The design and verification of security protocols. Technical report, Defence Research Agency, 1996.

[Sch97] Stece Schneider. Verifying authentication protocols with csp. In *10th Computer Security Foundations Workshop*, 1997.

[Sta99] William Stallings. *Cryptography and Network Security, Principles and Practice*. Prentice Hall Inc., second edition, 1999.

[Syv93] Paul F. Syverson. Adding time to a logic of authentication. In *Proceedings of th First ACM Conference on Computer and Communications Security*, pages 97–101, 1993.

[vO93] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the First ACM Conference on Computers and Communications Security*, pages 232–243, 1993.

Appendix A

Isabelle Theories

This appendix contains the Isabelle/NLDCHOL encoding of the theory developed in conjunction with this project. The theory of NLDCHOL itself is included in appendix B.

This appendix contains a session graph that shows the interdependencies of the developed theories. A few of the theories in this graph have not been developed as part of this project. We will briefly account for their sources:

NLDCHOL is a proof support environment for labelled interval logics that is embedded in Isabelle/HOL-Real. The development of this environment is due to Rasmussen, who finished the port to Isabelle/HOL after handing in his PhD thesis [Ras02].

Message is a theory of agents, keys, and message. This theory is part of Paulson's inductive method.

Public is a theory that regards the initial knowledge of agents in public key cryptosystems. This theory is part of Paulson's inductive method.

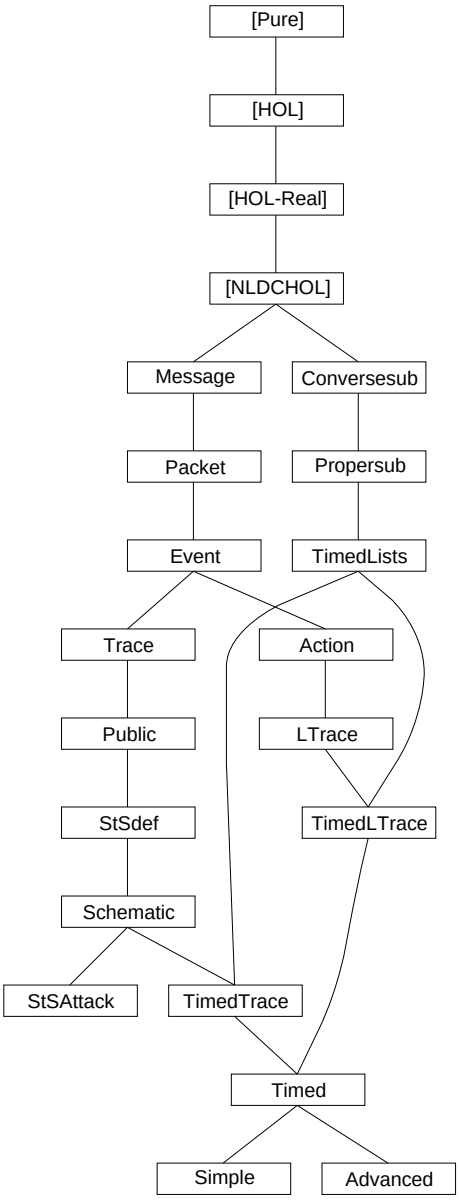
Event is novel work but inspired by the similar theory of Paulson's inductive method.

Remaining theories that descend from NLDCHOL are novel developments. All theories descending from NLDCHOL are documented in the main text.

On Presentation The less interesting parts of the setup, e.g. unused lemmas, have been left out.

The resulting presentation is quite accessible, but for one thing. Proofs are simply presented as proof-scripts and nothing else. It has been impossible to make Isabelle report on intermediate proof-states and, thus, the presentation is somewhat lacking.

A.1 Theory Dependency Graph



A.3 Untimed Theory of Global Traces

A.3.1 Agents, Keys, and Messages - Theory due to Paulson

theory *Message = NLDCHOL*
lemma [*HOL.simp*] : *A Un (B Un A) = B Un A*

Agents

datatype
agent = Server | Friend nat | Spy

Cryptographic Keys

types
key = nat

consts
invKey :: key=>key

axioms
invKey [*HOL.simp*] : *invKey (invKey K) = K*

constdefs
symKeys :: key set
symKeys == {K. invKey K = K}

Messages

datatype
msg = Agent agent
| Number nat
| Nonce nat
| Key key
| Hash msg
| MPair msg msg
| Crypt key msg

syntax
*@MTuple :: ['a, args] => 'a * 'b ((2{|-,/ -|}))*

syntax (*xsymbols*)
*@MTuple :: ['a, args] => 'a * 'b ((2{|-,/ -|}))*

translations

```
  {|x, y, z|}   == {|x, {|y, z|}|}
  {|x, y|}      == MPair x y

constdefs
  HPair :: [msg,msg] => msg                    ((4Hash[-] /-) [0, 1000])
    Hash[X] Y == {| Hash{|X,Y|}, Y|}

  keysFor :: msg set => key set
  keysFor H == invKey ` {K. ∃X. Crypt K X ∈ H}
```

Operators Dealing With Sets of Messages

```
consts  parts   :: msg set => msg set
inductive parts H
 intros
   Inj [HOL.intro]:          X ∈ H ==> X ∈ parts H
   Fst:        {|X,Y|}   ∈ parts H ==> X ∈ parts H
   Snd:        {|X,Y|}   ∈ parts H ==> Y ∈ parts H
   Body:       Crypt K X ∈ parts H ==> X ∈ parts H

lemma parts-mono: G<=H ==> parts(G) <= parts(H)
consts  analz   :: msg set => msg set
inductive analz H
 intros
 Inj [HOL.intro,HOL.simp] :
                X ∈ H ==> X ∈ analz H
 Fst:           {|X,Y|} ∈ analz H ==> X ∈ analz H
 Snd:           {|X,Y|} ∈ analz H ==> Y ∈ analz H
 Decrypt [HOL.dest]:[|Crypt K X ∈ analz H; Key(invKey K): analz H|] ==>
X ∈ analz H

lemma analz-mono: G<=H ==> analz(G) <= analz(H)
consts  synth   :: msg set => msg set
inductive synth H
 intros
   Inj   [HOL.intro]:   X ∈ H ==> X ∈ synth H
   Agent [HOL.intro]:   Agent agt ∈ synth H
   Number [HOL.intro]:   Number n  ∈ synth H
   Hash   [HOL.intro]:   X ∈ synth H ==> Hash X ∈ synth H
   MPair [HOL.intro]:   [|X ∈ synth H;  Y ∈ synth H|] ==> {|X,Y|} ∈ synth
H
    Crypt  [HOL.intro]:   [|X ∈ synth H;  Key(K) ∈ H|] ==> Crypt K X ∈
synth H
```

lemma *synth-mono*: $G <= H \Longrightarrow synth(G) <= synth(H)$

A.3.2 The Network Packets

theory *Packet* = *Message* :

Components of Packets

types
Loc = *agent*
Nm = *agent*
Adr = *Nm* × *Loc*
Pkt = *Adr* × *Adr* × *msg*

syntax
@Pkt :: [*Adr*, *Adr*, *msg*] => *Pkt* ('(- –> -: -'))

syntax (*xsymbols*)
@Pkt :: [*Adr*, *Adr*, *msg*] => *Pkt* ('(⦇- → -: -⦈))

translations
(*x* –> *y*: *k*) == (*x*, *y*, *k*)

constdefs
fakeAdr :: *Adr* ⇒ *bool*
fakeAdr ≡ λ*p*. *fst p* ≠ *snd p*

goodAdr :: *Adr* ⇒ *bool*
goodAdr ≡ λ*p*. ¬*fakeAdr p*

Projections of Packets

constdefs
sAdrPkt :: *Pkt* ⇒ *Adr*
sAdrPkt ≡ λ*p*. *fst p*

rAdrPkt :: *Pkt* ⇒ *Adr*
rAdrPkt ≡ λ*p*. *fst* (*snd p*)

msgPkt :: *Pkt* ⇒ *msg*
msgPkt ≡ λ*p*. *snd* (*snd p*)

sNmPkt :: *Pkt* ⇒ *Nm*
sNmPkt ≡ λ*p*. *fst* (*sAdrPkt p*)

$sLocPkt :: Pkt \Rightarrow Loc$
$sLocPkt \equiv \lambda p.\ snd\ (sAdrPkt\ p)$

$rNmPkt :: Pkt \Rightarrow Nm$
$rNmPkt \equiv \lambda p.\ fst\ (rAdrPkt\ p)$

$rLocPkt :: Pkt \Rightarrow Loc$
$rLocPkt \equiv \lambda p.\ snd\ (rAdrPkt\ p)$

lemma *sNmPkt*: $sNmPkt\ (\!|(x, y) \rightarrow (u, v): m|\!) = x$
lemma *sLocPkt*: $sLocPkt\ (\!|(x, y) \rightarrow (u, v): m|\!) = y$
lemma *rNmPkt*: $rNmPkt\ (\!|(x, y) \rightarrow (u, v): m|\!) = u$
lemma *rLocPkt*: $rLocPkt\ (\!|(x, y) \rightarrow (u, v): m|\!) = v$
lemma *msgPkt*: $msgPkt\ (\!|s \rightarrow r: m|\!) = m$

Judgments on Packets

constdefs
$fakePkt :: Pkt \Rightarrow bool$
$fakePkt \equiv \lambda p.\ fakeAdr\ (fst\ p) \vee fakeAdr\ (fst\ (snd\ p))$

$goodPkt :: Pkt \Rightarrow bool$
$goodPkt \equiv \lambda p.\ \neg fakePkt\ p$

lemma *fakeAdrE* [*rule-format* (*no-asm*)]: $fakeAdr\ (x, y) \longrightarrow x \neq y$
lemma *fakeAdrI* [*rule-format* (*no-asm*)]: $x \neq y \longrightarrow fakeAdr(x, y)$
lemma *fakePktIs* [*rule-format* (*no-asm*)]: $(x \neq y) \longrightarrow fakePkt\ (\!|(x, y) \rightarrow r: m|\!)$
lemma *fakePktIr* [*rule-format* (*no-asm*)]: $(u \neq v) \longrightarrow fakePkt\ (\!|s \rightarrow (u, v): m|\!)$
lemma *goodPktEs* [*rule-format* (*no-asm*)]: $goodPkt\ (\!|(x, y) \rightarrow r: m|\!) \longrightarrow (x = y)$
lemma *goodPktEr* [*rule-format* (*no-asm*)]: $goodPkt\ (\!|s \rightarrow (u, v): m|\!) \longrightarrow (u = v)$
lemma *fakePktE* [*rule-format* (*no-asm*)]: $fakePkt\ (\!|(x, y) \rightarrow (u, v): m|\!) \longrightarrow (x \neq y) \vee (u \neq v)$
lemma *goodPktI* [*rule-format* (*no-asm*)]: $(x = y) \wedge (u = v) \longrightarrow goodPkt\ (\!|(x, y) \rightarrow (u, v): m|\!)$
lemma *goodPkt*: $goodPkt\ (\!|(x, x) \rightarrow (y, y): m|\!)$
end

A.3.3 Events, Knowledge, and Freshness

theory *Event* = *Packet*:

The Events of The Network

datatype
event = *Snd Nm Pkt*
| *Rcv Nm Pkt*
| *Blk Nm Pkt*

axioms
hdnil-not-event [*iff*]: ((*e*::*event*) = *hd* []) = *False*

constdefs
fake-event :: *event* ⇒ *bool*
fake-event ≡ λ*e*. *case e of Snd n p* ⇒ *sNmPkt p* ≠ *n* ∨ *fakePkt p*
| *Rcv n p* ⇒ *rNmPkt p* ≠ *n* ∨ *fakePkt p*
| *Blk n p* ⇒ *rNmPkt p* ≠ *n* ∨ *fakePkt p*

good-event :: *event* ⇒ *bool*
good-event e ≡ ¬*fake-event e*

The knowledge of Agents

consts
initState :: *agent* ⇒ *msg set*
bad :: *agent set*
knows :: *agent* ⇒ *event list* ⇒ *msg set*
primrec
knows-Nil: *knows A* [] = *initState A*
knows-Cons: *knows A* (*x* # *xs*) =
(*if A* = *Spy then*
(*case x*
of Snd A' P ⇒ *if A'* ∈ *bad* ∨ *A'* = *Spy*
then
insert (*msgPkt P*) (*knows Spy xs*)
else
knows Spy xs
| *Rcv A' P* ⇒ *if A'* ∈ *bad* ∨ *A'* = *Spy*
then
insert (*msgPkt P*) (*knows Spy xs*)
else
knows Spy xs

```
  | Blk A' P ⇒ knows Spy xs)
 else
 (case x
  of Snd A' P ⇒ if A' = A
                then
                  insert (msgPkt P) (knows A xs)
                else
                  knows A xs
  | Rcv A' P ⇒ if A' = A
                then
                  insert (msgPkt P) (knows A xs)
                else
                  knows A xs
  | Blk A' P ⇒ knows A xs))
```

syntax
spies :: *event list* ⇒ *msg set*

translations
spies == *knows Spy*

axioms
Spy-in-bad: *Spy* ∈ *bad*
Server-not-bad: *Server* ∉ *bad*

Freshness of Message Components

consts *used* :: *event list* => *msg set*
primrec
used-Nil: *used* [] = (⋃ *B*. *parts* (*initState B*))
used-Cons: *used* (*x* # *xs*) = (*case x*
of Snd A P => *parts* {*msgPkt P*} ∪ (*used xs*)
| *Rcv A P* => *used xs*
| *Blk A P* => *used xs*)

end

Shared Key Systems

theory *Shared = Event :*
consts
shrK :: agent => key

axioms
isSym-keys: K ∈ symKeys
inj-shrK: inj shrK

primrec
initState-Server: initState Server = Key ‘ range shrK
initState-Friend: initState (Friend i) = {Key (shrK (Friend i))}
initState-Spy: initState Spy = Key‘shrK‘bad

axioms
Key-supply-ax: finite KK ==> EX K. K ~: KK & Key K ~: used evs
end

Public Key Systems

theory *Public = Trace :*
consts
pubK :: agent => key

syntax
priK :: agent => key

translations
priK x == invKey(pubK x)

primrec
initState-Server: initState Server =
insert (Key (priK Server)) (Key ‘ range pubK)
initState-Friend: initState (Friend i) =
insert (Key (priK (Friend i))) (Key ‘ range pubK)
initState-Spy: initState Spy =
(Key‘invKey‘pubK‘bad) Un (Key ‘ range pubK)

axioms
inj-pubK: inj pubK
priK-neq-pubK: priK A ~= pubK B

lemma [*simp*]: *K ∈ symKeys ==> invKey K = K***end**

A.3.4 Traces

theory *Trace = Event*:

Visibible Packets

```
consts visible :: [Pkt, event list] => bool  (infixr 60)
primrec
  visible-Nil:  p visible [] = False
  visible-Cons: p visible (x#xs) = (case x of Snd n q => if q = p
                                              then
                                                True
                                              else
                                                p visible xs
                                    | Rcv n q => p visible xs
                                    | Blk n q => if q = p
                                              then
                                                False
                                              else
                                                p visible xs)
```

Well-formed Traces

```
consts trace :: event list set
inductive trace
intros
  Nil [HOL.intro]: [] ∈ trace
  Snd [HOL.intro]: h ∈ trace ==> ((Snd n p) # h) ∈ trace
  Blk [HOL.intro]: [| h ∈ trace; p visible h |] ==> ((Blk n p) # h) ∈ trace
  Rcv [HOL.intro]: [| h ∈ trace; p visible h |] ==> ((Rcv n p) # h) ∈ trace
```

Projection of Traces

```
consts ↓ :: [event list, Nm] => event list  (infixr 60)
primrec
  evlproj-Nil:  [] ↓ n = []
  evlproj-Cons: x#xs ↓ n = (case x of Snd m p => (if n = m
                                                  then
                                                    (Snd m p) # (xs ↓ n)
                                                  else
                                                    (xs ↓ n))
                            | Rcv m p => (if n = m
                                          then
```

```
                    (Rcv m p) # (xs ↓ n)
                  else
                    (xs ↓ n))
  | Blk m p => (if n = m
                  then
                    (Blk m p) # (xs ↓ n)
                  else
                    (xs ↓ n)))


end
```

Public Key Systems

A.3.5 Ideal Protocol Definitions

theory *StSdef* = *Public*:

constdefs
Sign :: [*agent*, *msg*] ⇒ *msg* ((*4Sign*[-] -) [*0*, *1000*])
Sign[*A*] *Y* ≡ *Crypt* (*priK A*) *Y*

HKey :: *nat* ⇒ *msg*
HKey X ≡ *Nonce X*

consts
combiK :: *nat* × *nat* ⇒ *key*

axioms
inj-combiK: *inj combiK*
isSym-combiK: *combiK* (*A*, *B*) ∈ *symKeys*

consts
StS :: *event list set*
inductive *StS*
intros
Nil [*HOL.intro*]: [] ∈ *StS*

Rcv [*HOL.intro*]: ⟦*h* : *StS*; ⦇(*B*, *B*) → (*A*, *A*): *m*⦈ *visible h*⟧ ⟹
Blk A ⦇(*B*, *B*) → (*A*, *A*): *m*⦈ # *Rcv A* ⦇(*B*, *B*) → (*A*, *A*): *m*⦈ # *h* : *StS*

Msg1 [*HOL.intro*]: ⟦*h* ∈ *StS*; *HKey aXA* ∉ *used h*⟧ ⟹
Snd A ⦇(*A*, *A*) → (*B*, *B*): *HKey aXA*⦈ # *h* ∈ *StS*

Msg2 [*HOL.intro*]: ⟦*h* ∈ *StS*; *HKey aXB* ∉ *used h*; *hd* (*h* ↓ *B*) = *Blk B* ⦇(*A*, *A*) → (*B*, *B*): *HKey aXA*⦈⟧ ⟹
Snd B ⦇(*B*, *B*) → (*A*, *A*): {|*HKey aXB*, *Crypt* (*combiK*(*aXA*, *aXB*)) (*Sign*[*B*] {|*HKey aXB*, *HKey aXA*|})|}⦈ # *h* : *StS*

Msg3 [*HOL.intro*]: ⟦*h* ∈ *StS*; *hd* (*h* ↓ *A*) =
Blk A ⦇(*B*, *B*) → (*A*, *A*): {|*HKey aXB*, *Crypt* (*combiK*(*aXA*, *aXB*)) (*Sign*[*B*] {|*HKey aXB*, *HKey aXA*|})|}⦈⟧
⟹ *Snd A* ⦇(*A*, *A*) → (*B*, *B*): *Crypt* (*combiK*(*aXA*,*aXB*)) (*Sign*[*A*] {|*HKey aXA*, *HKey aXB*|})⦈ # *h* : *StS*

theorem *StS-subset-trace*: *StS* <= *trace*
theorems *traceI* = *StS-subset-trace* [*THEN subsetD*]

end

A.3.6 Schematics of Traces and Active Attacks

theory *Schematic* = *StSdef*:

types
pline-schema = *Nm* × *Nm* × *msg*
pschema = *pline-schema list*

Schematic Projection of Traces

consts
↑ :: [*event list*, *Nm*] ⇒ *pschema* (**infixr** *60*)
primrec
projs-Nil: [] ↑ *n* = []
projs-Cons: *x*#*xs* ↑ *n* = (*case x*
of Snd m (|(*n1*, *l1*) → (*n2*, *l2*): *m1*|) ⇒
(*if n* = *m* ∧ *n* = *n1*
then
(*n1*,*n2*, *m1*) # (*xs* ↑ *n*)
else
(*xs* ↑ *n*))
| *Rcv m* (|(*n1*, *l1*) → (*n2*, *l2*): *m1*|) ⇒
(*if n* = *m* ∧ *n* = *n2*
then
(*n1*, *n2*, *m1*) # (*xs* ↑ *n*)
else
(*xs* ↑ *n*))
| *Blk m p* ⇒ *xs* ↑ *n*)

Consistent Executions

consts
Agents :: *event list* ⇒ *agent set*
primrec
Agents-Nil: *Agents* [] = {}
Agents-Cons: *Agents* (*e*#*evs*) = (*case e of Snd X P* ⇒ *if X* ≠ *Spy then insert X* (*Agents evs*) *else* (*Agents evs*)
| *Rcv X P* ⇒ *if X* ≠ *Spy then insert X* (*Agents evs*) *else* (*Agents evs*)
| *Blk X P* ⇒ (*Agents evs*))

constdefs
consis :: [*event list*, *event list set*] ⇒ *bool* (- △ -)
evs △ *ideal* ≡ ∀ *a* ∈ (*Agents evs*). ∃ *s* ∈ *ideal*. *evs* ∈ *trace* ∧ (*s* ↑ *a* = *evs* ↑ *a*)

Identifying Active Attacks

consts
finish :: [*event list*, *agent*] ⇒ *bool* (**infixr** *60*)
defs
finish-def:
evs finish a ≡
∃ *b aXA aXB c*.
Snd a (|(*a*, *a*) → (*b*, *c*): *Crypt* (*combiK*(*aXA*,*aXB*)) (*Sign*[*a*] {|*HKey aXA*, *HKey aXB*|})|) ∈ *set* (*evs* ↓ *a*)
∨
Rcv a (|(*b*, *c*) → (*a*, *a*): *Crypt* (*combiK*(*aXA*,*aXB*)) (*Sign*[*b*] {|*HKey aXB*, *HKey aXA*|})|) ∈ *set* (*evs* ↓ *a*)

constdefs
sattack :: [*event list*, *agent*] ⇒ *bool* (- ∇ -)
evs ∇ *a* ≡ ∃ *e* ∈ *set* (*evs* ↓ *a*). (*evs* △ *StS*) ∧ *fake-event*(*e*) ∧ (*evs finish a*)

end

A.4 Example: The Station-to-Station Protocol

theory *StSAttack* = *Schematic*:

Modelling StS

See section: Ideal Protocol Definitions

Consistency of Repeated Protocol Engagement

lemma *one-app*: $[\![a \neq b;\ Xa1 \neq Xa2;\ Xb1 \neq Xa1;\ Xb1 \neq Xb2;\ Xb2 \neq Xa1;\ Xb2 \neq Xa2;\ Xb1 \neq Xa2]\!] \Longrightarrow (\exists s{\in}StS.\ s \uparrow Friend\ b =$
$[(Friend\ a,\ Friend\ b,\ Crypt\ (combiK\ (Xa2,\ Xb2))\ Sign[Friend\ a]\ \{\!|HKey\ Xa2,\ HKey\ Xb2|\!\}),$
$(Friend\ b,\ Friend\ a,\ \{\!|HKey\ Xb2,\ Crypt\ (combiK\ (Xa2,\ Xb2))\ Sign[Friend\ b]\ \{\!|HKey\ Xb2,\ HKey\ Xa2|\!\}|\!\}),$
$(Friend\ a,\ Friend\ b,\ HKey\ Xa2),$
$(Friend\ a,\ Friend\ b,\ Crypt\ (combiK\ (Xa1,\ Xb1))\ Sign[Friend\ a]\ \{\!|HKey\ Xa1,\ HKey\ Xb1|\!\}),$
$(Friend\ b,\ Friend\ a,\ \{\!|HKey\ Xb1,\ Crypt\ (combiK\ (Xa1,\ Xb1))\ Sign[Friend\ b]\ \{\!|HKey\ Xb1,\ HKey\ Xa1|\!\}|\!\}),$
$(Friend\ a,\ Friend\ b,\ HKey\ Xa1)])$

lemma *two-app*: $[\![a \neq b;\ a \neq c;\ Xa1 \neq Xa2;\ Xb \neq Xa1;\ Xb \neq Xc;\ Xc \neq Xa1;\ Xc \neq Xa2;\ Xb \neq Xa2]\!] \Longrightarrow (\exists s{\in}StS.\ s \uparrow Friend\ a =$
$[(|\ Friend\ a \rightarrow Friend\ b:\ Crypt\ (combiK\ (Xa2,\ Xb))\ Sign[Friend\ a]\ \{\!|HKey\ Xa2,\ HKey\ Xb|\!\}|),$
$(|\ Friend\ b \rightarrow Friend\ a:\ \{\!|HKey\ Xb,\ Crypt\ (combiK\ (Xa2,\ Xb))\ Sign[Friend\ b]\ \{\!|HKey\ Xb,\ HKey\ Xa2|\!\}|\!\}|),$
$(|\ Friend\ a \rightarrow Friend\ b:\ HKey\ Xa2\ |),$
$(|\ Friend\ a \rightarrow Friend\ c:\ Crypt\ (combiK\ (Xa1,\ Xc))\ Sign[Friend\ a]\ \{\!|HKey\ Xa1,\ HKey\ Xc|\!\}|),$
$(|\ Friend\ c \rightarrow Friend\ a:\ \{\!|HKey\ Xc,\ Crypt\ (combiK\ (Xa1,\ Xc))\ Sign[Friend\ c]\ \{\!|HKey\ Xc,\ HKey\ Xa1|\!\}|\!\}|),$
$(|\ Friend\ a \rightarrow Friend\ c:\ HKey\ Xa1\ |)])$

lemma *app-b*: $[\![a \neq b;\ Xa{\neq}Xb]\!] \Longrightarrow (\exists s{\in}StS.\ s \uparrow Friend\ b =$
$[(|Friend\ a \rightarrow Friend\ b:\ Crypt\ (combiK\ (Xa,\ Xb))\ Sign[Friend\ a]\ \{\!|HKey\ Xa,\ HKey\ Xb|\!\}|),$
$(|Friend\ b \rightarrow Friend\ a:\ \{\!|HKey\ Xb,\ Crypt\ (combiK\ (Xa,\ Xb))\ Sign[Friend\ b]\ \{\!|HKey\ Xb,\ HKey\ Xa|\!\}|\!\}|),$

$(|Friend\ a \to Friend\ b\colon HKey\ Xa|)])$

lemma *app-a*: $[\![a \neq b;\ Xa \neq Xb]\!] \Longrightarrow (\exists\, s{\in}StS.\ s \uparrow Friend\ a =$
$[(|Friend\ a \to Friend\ b\colon Crypt\ (combiK\ (Xa,\ Xb))\ Sign[Friend\ a]\ \{\!|HKey\ Xa,$
$HKey\ Xb|\!\}|),$
$(|Friend\ b \to Friend\ a\colon \{\!|HKey\ Xb,\ Crypt\ (combiK\ (Xa,\ Xb))\ Sign[\ Friend\ b]$
$\{\!|HKey\ Xb,\ HKey\ Xa|\!\}\!\}|),$
$(|Friend\ a \to Friend\ b\colon HKey\ Xa|)])$

theorem *Repeat-consis*: $[\![a \neq b;\ a \neq c;\ \ Xa1 \neq Xa2;\ Xb \neq Xa1;\ Xb \neq Xc;\ Xc$
$\neq Xa1;\ Xc \neq Xa2;\ Xb \neq Xa2]\!] \Longrightarrow$
(
Blk (*Friend c*) (|(*Friend a*, *Friend a*) → (*Friend c*, *Friend c*)\colon *Crypt* (*combiK*(*Xa2*,
Xc)) (*Sign*[*Friend a*] {|*HKey Xa2*, *HKey Xc*|}) |) #
Rcv (*Friend c*) (|(*Friend a*, *Friend a*) → (*Friend c*, *Friend c*)\colon *Crypt* (*combiK*(*Xa2*,
Xc)) (*Sign*[*Friend a*] {|*HKey Xa2*, *HKey Xc*|}) |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend c*, *Friend c*)\colon *Crypt* (*combiK*(*Xa2*,
Xc)) (*Sign*[*Friend a*] {|*HKey Xa2*, *HKey Xc*|}) |) #
Blk (*Friend a*) (|(*Friend c*, *Friend c*) → (*Friend a*, *Friend a*)\colon {|*HKey Xc*,
Crypt (*combiK*(*Xa2*, *Xc*)) (*Sign*[*Friend c*] {|*HKey Xc*, *HKey Xa2*|})|} |) #
Rcv (*Friend a*) (|(*Friend c*, *Friend c*) → (*Friend a*, *Friend a*)\colon {|*HKey Xc*,
Crypt (*combiK*(*Xa2*, *Xc*)) (*Sign*[*Friend c*] {|*HKey Xc*, *HKey Xa2*|})|} |) #
Snd (*Friend c*) (|(*Friend c*, *Friend c*) → (*Friend a*, *Friend a*)\colon {|*HKey Xc*,
Crypt (*combiK*(*Xa2*, *Xc*)) (*Sign*[*Friend c*] {|*HKey Xc*, *HKey Xa2*|})|} |) #
Blk (*Friend c*) (|(*Friend a*, *Friend a*) → (*Friend c*, *Friend c*)\colon *HKey Xa2* |) #
Rcv (*Friend c*) (|(*Friend a*, *Friend a*) → (*Friend c*, *Friend c*)\colon *HKey Xa2* |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend c*, *Friend c*)\colon *HKey Xa2* |) #

Blk (*Friend b*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*)\colon *Crypt* (*combiK*(*Xa1*,
Xb)) (*Sign*[*Friend a*] {|*HKey Xa1*, *HKey Xb*|}) |) #
Rcv (*Friend b*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*)\colon *Crypt* (*combiK*(*Xa1*,
Xb)) (*Sign*[*Friend a*] {|*HKey Xa1*, *HKey Xb*|}) |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*)\colon *Crypt* (*combiK*(*Xa1*,
Xb)) (*Sign*[*Friend a*] {|*HKey Xa1*, *HKey Xb*|}) |) #
Blk (*Friend a*) (|(*Friend b*, *Friend b*) → (*Friend a*, *Friend a*)\colon {|*HKey Xb*,
Crypt (*combiK*(*Xa1*, *Xb*)) (*Sign*[*Friend b*] {|*HKey Xb*, *HKey Xa1*|})|} |) #
Rcv (*Friend a*) (|(*Friend b*, *Friend b*) → (*Friend a*, *Friend a*)\colon {|*HKey Xb*,
Crypt (*combiK*(*Xa1*, *Xb*)) (*Sign*[*Friend b*] {|*HKey Xb*, *HKey Xa1*|})|} |) #
Snd (*Friend b*) (|(*Friend b*, *Friend b*) → (*Friend a*, *Friend a*)\colon {|*HKey Xb*,
Crypt (*combiK*(*Xa1*, *Xb*)) (*Sign*[*Friend b*] {|*HKey Xb*, *HKey Xa1*|})|} |) #
Blk (*Friend b*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*)\colon *HKey Xa1* |) #
Rcv (*Friend b*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*)\colon *HKey Xa1* |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*)\colon *HKey Xa1* |) #

```
[]
 ) △ StS
 apply (simp add: consis-def del: HKey-def Sign-def)
 apply (rule conjI)
 apply (force)
 apply (rule impI)
 apply (rule conjI)
 apply (rule impI)
 apply (rule conjI)
 apply (rule impI)
 apply (simp del: HKey-def Sign-def)
 apply (rule impI)
 apply (rule conjI)+
 apply force
 apply (rule conjI)
 apply (rule one-app)
 apply assumption
 apply (simp, simp, simp, simp, simp)
 apply simp
 apply (rule conjI)
 apply force
 apply (rule conjI, (rule one-app | rule two-app | rule app-a | rule app-b),
   (assumption | (rule not-sym, assumption))+,
   (rule conjI),
     force)+
 apply (rule two-app)
 apply (assumption | (rule not-sym, assumption))+

 apply (rule impI)
 apply (rule conjI)
 apply (rule impI)
 apply (simp del: HKey-def Sign-def)
 apply (rule impI)
 apply (rule conjI)
 apply (rule impI)


 apply (rule conjI)
 apply force
 apply (rule conjI)
 apply (simp del: HKey-def Sign-def)+
 apply (rule impI)
 apply (rule conjI, force, rule conjI, (rule one-app | rule two-app | rule app-a
```

| *rule app-b*),
(*assumption* | (*rule not-sym*, *assumption*))+)+
apply (*rule conjI*)
apply *force*
apply (*rule two-app*)
by (*assumption* | (*rule not-sym*, *assumption*))+

Active Attack on StS

lemma *Attack-is-Trace*: *Blk Spy* (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Blk (*Friend a*) (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Rcv (*Friend a*) (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Snd Spy (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Blk Spy (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Rcv Spy (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Snd (*Friend b*) (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Blk (*Friend b*) (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv (*Friend b*) (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd Spy (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) # [] : *trace*
by *force*

lemma *Attack-a-proj-StS*: ⟦*a*≠*b*; *Xa*≠*Xb*⟧ ⟹ ∃ *evs* ∈ *StS*. (*evs* ↑ *Friend a*) =

((
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #

Snd (*Friend a*) (‖(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) ‖ #
Blk (*Friend a*) (‖(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Rcv (*Friend a*) (‖(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Snd Spy (‖(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Blk Spy (‖(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Rcv Spy (‖(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Snd (*Friend b*) (‖(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Blk (*Friend b*) (‖(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ #
Rcv (*Friend b*) (‖(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ #
Snd Spy (‖(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ #
Blk Spy (‖(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ #
Rcv Spy (‖(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ #
Snd (*Friend a*) (‖(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ # []) ↑ *Friend a*)

lemma *Attack-finish-a*: $a \neq b \Longrightarrow$
Blk Spy (‖(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) ‖ #
Rcv Spy (‖(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) ‖ #
Snd (*Friend a*) (‖(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) ‖ #
Blk (*Friend a*) (‖(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Rcv (*Friend a*) (‖(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Snd Spy (‖(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Blk Spy (‖(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Rcv Spy (‖(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Snd (*Friend b*) (‖(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} ‖ #
Blk (*Friend b*) (‖(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ #
Rcv (*Friend b*) (‖(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* ‖ #

Snd Spy (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) # []
finish Friend a
by (*simp add*: *finish-def*)

lemma *Attack-a-Agents*: $a \neq b \Longrightarrow$ *Friend a* ∈ *Agents*
(
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Blk (*Friend a*) (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Rcv (*Friend a*) (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Snd Spy (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Blk Spy (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Rcv Spy (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Snd (*Friend b*) (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Blk (*Friend b*) (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv (*Friend b*) (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd Spy (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) # []
)

lemma *Attack-b-Agents*: $a \neq b \Longrightarrow$ *Friend b* ∈ *Agents*
(
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*,

Xb)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |})]) #
Blk (*Friend a*) ([|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Rcv (*Friend a*) ([|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Snd Spy ([|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Blk Spy ([|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Rcv Spy ([|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Snd (*Friend b*) ([|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Blk (*Friend b*) ([|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |]) #
Rcv (*Friend b*) ([|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |]) #
Snd Spy ([|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |]) #
Blk Spy ([|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |]) #
Rcv Spy ([|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |]) #
Snd (*Friend a*) ([|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |]) # []
)

lemma *Attack-consis*: ⟦*a* ≠ *b*; *b* ≠ *c*; *Xa*≠*Xb*⟧ ⟹
(
Blk Spy ([|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |]) #
Rcv Spy ([|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |]) #
Snd (*Friend a*) ([|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |]) #
Blk (*Friend a*) ([|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Rcv (*Friend a*) ([|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Snd Spy ([|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Blk Spy ([|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Rcv Spy ([|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Snd (*Friend b*) ([|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |]) #
Blk (*Friend b*) ([|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |]) #
Rcv (*Friend b*) ([|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |]) #

Snd Spy (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) # []
) △ *StS*
apply (*simp add*: *consis-def*)
apply *auto*
apply *force*
apply (*rule bexI*)
prefer *2*
apply (*rule StS.Msg3*)
apply (*rule-tac A* = *Friend a* **and** *B* = *Friend b* **in** *StS.Rcv*)
apply (*rule StS.Msg2*)
apply (*rule-tac A* = *Friend b* **and** *B* = *Friend a* **in** *StS.Rcv*)
apply (*rule StS.Msg1*)
apply (*rule StS.Nil*)
apply *force*
apply *force*
defer *1*
apply *force*
apply *force*
apply *force*
apply *force*
apply *force*
prefer *3*
apply (*rule bexI*)
prefer *2*
apply (*rule StS.Msg3*)
apply (*rule-tac A* = *Friend a* **and** *B* = *Friend b* **in** *StS.Rcv*)
apply (*rule StS.Msg2*)
apply (*rule-tac A* = *Friend b* **and** *B* = *Friend a* **in** *StS.Rcv*)
apply (*rule StS.Msg1*)
apply (*rule StS.Nil*)
defer *1*
apply *force*
defer *1*
apply *force*
apply *force*
apply *force*
apply *force*
prefer *2*
apply *force*
apply (*rule bexI*)

prefer *2*
apply (*rule StS.Msg2*)
apply (*rule-tac A = Friend b* **and** *B = Friend c* **in** *StS.Rcv*)
apply (*rule StS.Msg1*)
apply (*rule StS.Nil*)
defer *1*
apply *force*
defer *1*
by *force+*

theorem *Attack-a-successful*: ⟦*a ≠ b*; *b ≠ c*; *Xa≠Xb*⟧ ⟹
(
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*))
(*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*, *Xb*))
(*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Spy*): *Crypt* (*combiK*(*Xa*,
Xb)) (*Sign*[*Friend a*] {| *HKey Xa*, *HKey Xb* |}) |) #
Blk (*Friend a*) (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt*
(*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Rcv (*Friend a*) (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt*
(*combiK*(*Xa*, *Xb*))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Snd Spy (|(*Friend b*, *Spy*) → (*Friend a*, *Friend a*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*,
Xb))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Blk Spy (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*,
Xb))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Rcv Spy (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt* (*combiK*(*Xa*,
Xb))(*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Snd (*Friend b*) (|(*Friend b*, *Friend b*) → (*Friend c*, *Spy*): {| *HKey Xb*, *Crypt*
(*combiK*(*Xa*, *Xb*)) (*Sign*[*Friend b*] {| *HKey Xb*, *HKey Xa* |})|} |) #
Blk (*Friend b*) (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv (*Friend b*) (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd Spy (|(*Friend c*, *Spy*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Blk Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Rcv Spy (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) #
Snd (*Friend a*) (|(*Friend a*, *Friend a*) → (*Friend b*, *Friend b*): *HKey Xa* |) # []
) ▽ *Friend a*
apply (*simp add*: *sattack-def del*: *HKey-def Sign-def*)
apply (*rule conjI*)
apply (*rule Attack-consis*)
apply (*auto simp del*: *HKey-def Sign-def*)
apply (*rule Attack-finish-a*)
by *assumption*

end

A.5 Untimed Theory of Local Traces

A.5.1 A Suitable Set of Actions

theory *Action* = *Event*:

datatype
action = *SState nat*
| *RState nat*
| *Abort*
| *Accept*
| *Ver nat nat bool*
| *Ready*

axioms
hdnil-not-action [*iff*]: ((*x*::*action*) = *hd* []) = *False*

end

A.5.2 The Local Model

theory *LTrace* = *Action*:

A Simple Example

consts *ltrace* :: *agent* ⇒ *action list set*
inductive *ltrace A*
intros
Nil [*HOL.intro*!]: [] ∈ *ltrace A*

Ready [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *hd* [] ∨ *hd acs* = *Accept* ∨ *hd acs* = *Abort*⟧ ⟹ *Ready* # *acs* ∈ *ltrace A*

Init [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *Ready*⟧ ⟹ *SState* (*Suc 0*) # *acs* ∈ *ltrace A*

Resp [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *Ready*⟧ ⟹ *RState* (*Suc 0*) # *acs* ∈ *ltrace A*

Rcvi [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *RState i*⟧ ⟹ *Ver i* (*Suc 0*) *b* # *acs* ∈ *ltrace A*

Sndi [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *SState i*; *i* < *2*⟧ ⟹ *RState* (*Suc i*) # *acs* ∈ *ltrace A*

Veri [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *Ver i 1 True*; *i* < *2*⟧ ⟹ *SState* (*Suc i*) # *acs* ∈ *ltrace A*

Accept [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *Ver* (*Suc* (*Suc 0*)) (*Suc 0*) *True* ∨ *hd acs* = *SState 2*⟧ ⟹ *Accept* # *acs* ∈ *ltrace A*

Abort [*HOL.intro*!]: ⟦*acs* ∈ *ltrace A*; *hd acs* = *Ver i j False*⟧ ⟹ *Abort* # *acs* ∈ *ltrace A*

Defining Verification Actions

constdefs
chkAgent :: *agent* ⇒ *bool*
chkAgent ≡ λ*a*. *a* ∉ *bad*

lemma *chkServer*: *chkAgent Server*
lemma *chkSpy*: ¬*chkAgent Spy*

lemma *chkFriend* [*simp*]: *chkAgent* (*Friend a*) ≡ (¬(*Friend a*) ∈ *bad*)
lemma *Ver-True*: *P* ⟹ *Ver n m P* = *Ver n m True*
lemma *Ver-false*: ¬*P* ⟹ *Ver n m P* = *Ver n m False*
lemma *notbad-vertrue*: *A* ∉ *bad* ⟹ *Ver n m* (*chkAgent A*) = *Ver n m True*

lemma ∀ *acs* ∈ *ltrace A*. *acs* = *a*#(*Ver 1 1* (*chkAgent Spy*))#(*acts*::*action list*)
⟹ *a* = *Abort*
apply (*erule ballE*)
apply (*rule ltrace.cases*)
by *auto*

lemma ∃ *acts* ∈ *ltrace A*. *Accept*#*acts* ∈ *ltrace A*
apply (*rule bexI*)
prefer *2*
apply (*rule-tac b* = *chkAgent Server* **in** *Rcvi*)
apply (*rule Sndi*)
apply (*rule Init*)
apply (*rule Ready*)
apply (*rule Nil*)
apply *simp*
apply *simp*
apply *auto*
done

end

A.6 Timed Theory of Protocols

A.6.1 A Suitable Interval Logic

theory *Conversesub = NLDCHOL*:

The basis of this encoding is Rasmussen's NLDCHOL, which can be found in appendix B.

Converse Modalities

constdefs
arnconv :: *bool* => *bool* ([*Rc*]- [*50*] *50*)
[*Rc*]*P* ≡ ¬(<*Rc*>(¬*P*))
alnconv :: *bool* => *bool* ([*Lc*]- [*50*] *50*)
[*Lc*]*P* ≡ ¬(<*Lc*>(¬*P*))

theorem *slnconvI* [*intro*]: ⟦<*i*,*j*>:*fwd*; <*k*,*j*>:*fwd*; <*k*,*j*>:*P*⟧ ⟹ <*i*,*j*>:<*Lc*>*P*
theorem *srnconvI* [*intro*]: ⟦<*i*,*j*>:*fwd*; <*i*,*k*>: *fwd*; <*i*,*k*>:*P*⟧ ⟹ <*i*,*j*>:<*Rc*>*P*

theorem *slnconvE* [*elim*]:
assumes *1*: <*i*,*j*>: <*Lc*>*P*
and *2*: <*i*,*j*>:*fwd*
and *3*: !!*k*. ⟦<*k*,*j*>:*fwd*; <*k*,*j*>:*P*⟧ ⟹ <*m*,*n*>:*R*
shows <*m*,*n*>:*R*

theorem *srnconvE* [*elim*]:
assumes *1*: <*i*,*j*>: <*Rc*>*P*
and *2*: <*i*,*j*>:*fwd*
and *3*: !!*k*. ⟦<*i*,*k*>:*fwd*; <*i*,*k*>:*P*⟧ ⟹ <*m*,*n*>:*R*
shows <*m*,*n*>:*R*

theorem *alnconvI* [*intro*]:
assumes *1*: !! *k*. ⟦<*k*, *j*>: *fwd*⟧ ⟹ <*k*, *j*>: *P*
and *2*: <*i*,*j*>:*fwd*
shows <*i*, *j*>: [*Lc*]*P*

theorem *arnconvI* [*intro*]:
assumes *1*: !! *k*. ⟦<*i*, *k*>: *fwd*⟧ ⟹ <*i*, *k*>: *P*
and *2*: <*i*,*j*>:*fwd*
shows <*i*, *j*>: [*Rc*]*P*

theorem *alnconvD* [*dest*]: ⟦<*i*,*j*>:[*Lc*]*P*; <*k*,*j*>: *fwd*; <*i*, *j*>: *fwd*⟧ ⟹ <*k*, *j*>: *P*

theorem *arnconvD* [*dest*]: ⟦<*i*,*j*>:[*Rc*]*P*; <*i*,*k*>: *fwd*; <*i*, *j*>: *fwd*⟧ ⟹ <*i*, *k*>: *P*

theorem *alnconvE* [*elim*]:
assumes *1*: <*i*,*j*>: [*Lc*]*P*
and *2*: <*k*,*j*>:*P* ⟹ <*m*,*n*>:*R*
and *3*: <*k*,*j*>:*fwd*
and *4*: <*i*,*j*>:*fwd*
shows <*m*,*n*>:*R*

theorem *arnconvE* [*elim*]:
assumes *1*: <*i*,*j*>: [*Rc*]*P*
and *2*: <*i*,*k*>:*P* ⟹ <*m*,*n*>:*R*
and *3*: <*i*,*k*>:*fwd*
and *4*: <*i*,*j*>:*fwd*
shows <*m*,*n*>:*R*

end
theory *Propersub* = *Conversesub*:

Proper Subintervals

constdefs
spsub :: *bool* ⇒ *bool* (<*PS*>- [*50*] *50*)
<*PS*>*P* ≡ *0* < *len* |^| *P* |^| *0* < *len*
apsub :: *bool* ⇒ *bool* ([*PS*]- [*50*] *50*)
[*PS*]*P* ≡ ¬(<*PS*>(¬*P*))
srpsub :: *bool* ⇒ *bool* (<*RPS*>- [*50*] *50*)
<*RPS*>*P* ≡ *True* |^| *P* |^| *0* < *len*
arpsub :: *bool* ⇒ *bool* ([*RPS*]- [*50*] *50*)
[*RPS*]*P* ≡ ¬(<*RPS*>(¬*P*))
slpsub :: *bool* ⇒ *bool* (<*LPS*>- [*50*] *50*)
<*LPS*>*P* ≡ *0* < *len* |^| *P* |^| *True*
alpsub :: *bool* ⇒ *bool* ([*LPS*]- [*50*] *50*)
[*LPS*]*P* ≡ ¬(<*LPS*>(¬*P*))

lemma *zlesslen-imp-fwd*: <*i*, *j*>: *0* < *len* ⟹ <*i*, *j*>: *fwd*
lemma *zlesslen-zlelen-imp-zlelen*: ⟦<*i*, *k*>: *0* <= *len*; <*k*, *j*>: *0* < *len*⟧ ⟹ <*i*, *j*>: *0* < *len*
lemma *zlelen-zlesslen-imp-zlelen*: ⟦<*k*, *j*>: *0* <= *len*; <*i*, *k*>: *0* < *len*⟧ ⟹ <*i*, *j*>: *0* < *len*
lemma *ssub-zlesslen-imp-zlesslen*: ⟦<*i*, *j*>: *fwd*; <*i*, *j*>: <*S*> *0* < *len*⟧ ⟹ <*i*,

$j>$: $0 < len$
lemma *fwd-zlesslen-imp-zlesslen*: $[\![<i, k>: fwd; <k, j>: 0 < len]\!] \Longrightarrow <i, j>: 0 < len$
lemma *zlesslen-fwd-imp-zlesslen*: $[\![<k, j>: fwd; <i, k>: 0 < len]\!] \Longrightarrow <i, j>: 0 < len$
theorem *ilspsubI* [*intro*]: $[\![<i, k>: 0 < len; <k, l>: P; <k, l>: fwd; <l, j>: 0 < len]\!] \Longrightarrow <i, j>: <PS>P$

theorem *ilspsubE* [*elim*]:
assumes *1*: $<i, j>:<PS>P$
and *2*: $!!k\ l.\ [\![<i, k>: 0 < len; <k, l>: P; <k, l>: fwd; <l, j>: 0 < len]\!] \Longrightarrow <m, n>: R$
shows $<m, n>: R$

theorem *ilapsubI* [*intro*]:
assumes *1*: $!!\ k\ l.\ [\![<i, k>: 0 < len; <k, l>: fwd; <l, j>: 0 < len]\!] \Longrightarrow <k, l>: P$
shows $<i, j>: [PS]P$

theorem *ilapsubD* [*dest*]: $[\![<i,j>:[PS]P; <i,k>: 0 < len; <k, l>: fwd; <l,j>:0<len]\!] \Longrightarrow <k, l>: P$

theorem *ilapsubE* [*elim*]:
assumes *1*: $<i,j>: [PS]P$
and *2*: $<k,l>:P \Longrightarrow <m,n>:R$
and *3*: $<i,k>: 0 < len$
and *4*: $<k,l>: fwd$
and *5*: $<l,j>: 0 < len$
shows $<m,n>:R$

theorem *ilsrpsubI* [*intro*]: $[\![<i, k>: fwd; <k, l>: P; <k, l>: fwd; <l, j>: 0 < len]\!] \Longrightarrow <i, j>: <RPS>P$

theorem *ilsrpsubE* [*elim*]:
assumes *1*: $<i,j>:<RPS>P$
and *2*: $!!k\ l.\ [\![<i, k>: fwd; <k, l>: P; <k, l>: fwd; <l, j>: 0 < len]\!] \Longrightarrow <m, n>: R$
shows $<m, n>: R$

theorem *ilarpsubI* [*intro*]:

assumes *1*: !! *k l*. ⟦<*i*, *k*>: *fwd*; <*k*, *l*>: *fwd*; <*l*, *j*>: *0* < *len*⟧ ⟹ <*k*, *l*>: *P*
shows <*i*, *j*>: [*RPS*]*P*

theorem *ilarpsubD* [*dest*]: ⟦<*i*,*j*>:[*RPS*]*P*; <*i*,*k*>: *fwd*; <*k*, *l*>: *fwd*; <*l*,*j*>:*0*<*len*⟧
⟹ <*k*, *l*>: *P*

theorem *ilarpsubE* [*elim*]:
assumes *1*: <*i*,*j*>: [*RPS*]*P*
and *2*: <*k*,*l*>:*P* ⟹ <*m*,*n*>:*R*
and *3*: <*i*,*k*>: *fwd*
and *4*: <*k*,*l*>: *fwd*
and *5*: <*l*,*j*>: *0* < *len*
shows <*m*,*n*>:*R*

theorem *ilslpsubI* [*intro*]: ⟦<*i*,*k*>:*0* < *len*; <*k*,*l*>:*P*; <*k*,*l*>:*fwd*; <*l*,*j*>:*fwd*⟧
⟹ <*i*, *j*>: <*LPS*>*P*

theorem *ilslpsubE* [*elim*]:
assumes *1*: <*i*,*j*>:<*LPS*>*P*
and *2*: !!*k l*. ⟦<*i*, *k*>: *0* < *len*; <*k*, *l*>: *P*; <*k*, *l*>: *fwd*; <*l*, *j*>: *fwd*⟧ ⟹
<*m*, *n*>: *R*
shows <*m*, *n*>: *R*

theorem *ilalpsubI* [*intro*]:
assumes *1*: !! *k l*. ⟦<*i*, *k*>: *0* < *len*; <*k*, *l*>: *fwd*; <*l*, *j*>: *fwd*⟧ ⟹ <*k*, *l*>: *P*
shows <*i*, *j*>: [*LPS*]*P*

theorem *ilalpsubD* [*dest*]: ⟦<*i*,*j*>:[*LPS*]*P*; <*i*,*k*>: *0* < *len*; <*k*, *l*>: *fwd*; <*l*,*j*>:*fwd*⟧
⟹ <*k*,*l*>: *P*

theorem *ilalpsubE* [*elim*]:
assumes *1*: <*i*,*j*>: [*LPS*]*P*
and *2*: <*k*,*l*>:*P* ⟹ <*m*,*n*>:*R*
and *3*: <*i*,*k*>: *0* < *len*
and *4*: <*k*,*l*>: *fwd*
and *5*: <*l*,*j*>: *fwd*
shows <*m*,*n*>:*R*

lemma *lelen-lesslen-add*: [| *RI s*; *RI t*; <*i*,*k*>:*s*<=*len*; <*k*,*j*>:*t*<*len* |] ==>
<*i*,*j*>:*s*+*t*<=*len*

lemma *lesslen-lelen-add*: [| *RI s*; *RI t*; <*i*,*k*>:*s*<*len*; <*k*,*j*>:*t*<=*len* |] ==> <*i*,*j*>:*s*+*t*<=*len*
lemma *zlelen-zlesslen-imp-zlesslen*: [| <*i*,*k*>:*0*<=*len*; <*k*,*j*>:*0*<*len* |] ==> <*i*,*j*>:*0*<*len*
lemma *zlesslen-zlelen-imp-zlesslen*: [| <*i*,*k*>:*0*<*len*; <*k*,*j*>:*0*<=*len* |] ==> <*i*,*j*>:*0*<*len*
lemma *lelen-zlesslen-imp-lesslen*: [| <*i*,*k*>:*y*<=*len*; <*k*,*j*>:*0*<*len*; *RI y* |] ==> <*i*,*j*>:*y*<*len*
lemma *lelen-zlelen-imp-lelen*: [| <*i*,*k*>:*y*<=*len*; <*k*,*j*>:*0*<=*len*; *RI y* |] ==> <*i*,*j*>:*y*<=*len*
lemma *lesslen-zlelen-imp-lesslen*: [| <*i*,*k*>:*y*<*len*; <*k*,*j*>:*0*<=*len*; *RI y* |] ==> <*i*,*j*>:*y*<*len*
lemma *zlelen-lesslen-imp-lesslen*: [| <*i*,*k*>:*0*<=*len*; <*k*,*j*>:*y*<*len*; *RI y*|] ==> <*i*,*j*>:*y*<*len*
lemma *zlesslen-lelen-imp-lesslen*: [| <*i*,*k*>:*0*<*len*; <*k*,*j*>:*y*<=*len*; *RI y*|] ==> <*i*,*j*>:*y*<*len*
lemma *zlelen-lelen-imp-lelen*: [|<*i*,*k*>:*0*<=*len*; <*k*,*j*>:*y*<=*len*; *RI y*|] ==> <*i*,*j*>:*y*<=*len*
lemma *ssub-lesslen-imp-lesslen*: ⟦*RI x*; <*i*,*j*>:<*S*>*x* < *len*⟧ ⟹ <*i*,*j*>: *x* < *len*
lemma *ssub-lelen-imp-lelen*: ⟦*RI x*; <*i*,*j*>:<*S*>*x* <= *len*⟧ ⟹ <*i*,*j*>: *x* <= *len*
end

A.6.2 Timed Properties of Lists

theory *TimedLists* = *Propersub*:

axioms
RIemp-list [*HOL.intro*!]: *RI* []
RIconsI [*HOL.intro*!]: $[\![RI\ x;\ RI\ xs]\!] \Longrightarrow RI\ (x\#xs)$
RIconsE1 [*HOL.dest*]: $(RI\ (x\#xs)) \Longrightarrow RI\ x$
RIconsE2 [*HOL.dest*]: $(RI\ (x\#xs)) \Longrightarrow RI\ xs$

Equality for Lists

axioms
RIequI [*HOL.intro*!]: $[\![RI\ x;\ RI\ y]\!] \Longrightarrow RI\ x = (y::'a\ list)$
RIequE1: $(RI\ x = (y::'a\ list)) \Longrightarrow RI\ x$
RIequE2: $(RI\ x = (y::'a\ list)) \Longrightarrow RI\ y$
CFequ [*HOL.intro*!]: $CF\ (x = (y::'a\ list))$

theorem *lequtrans*: $[\![<i,j>:xs=ys;\ <i,j>:ys=zs]\!] \Longrightarrow <i,j>:xs=(zs::'a\ list)$

theorem *lequrefl*: $<i,j>:\ evs = (evs::'a\ list)$
theorem *lsubst*: [| $CF\ (P\ x)$; $<i,j>:s=t$; $<i,j>:(P\ s)$ |] ==> $<i,j>:(P\ (t::'a\ list))$
theorem *lsubstRI*: [| $RI\ s$; $RI\ t$; $<i,j>:s=t$; $<i,j>:(P\ s)$ |] ==> $<i,j>:(P\ (t::'a\ list))$

Precedes-or-Equals

constdefs
$prefix :: ['a\ list,\ 'a\ list] \Rightarrow bool$ $(-\preceq -\ [60,60]\ 60)$
$x \preceq y \equiv (\exists z.\ z@x = y)$

theorem *prerefl*: $evs \preceq evs$
theorem *pretrans*: $[\![evs \preceq evs';\ evs' \preceq evs'']\!] \Longrightarrow evs \preceq evs''$
theorem *preanti*: $[\![evs \preceq evs';\ evs' \preceq evs]\!] \Longrightarrow evs = evs'$
theorem *preweaken1*: $evs = evs' \Longrightarrow evs \preceq evs'$
theorem *preweaken2*: $evs = evs' \Longrightarrow evs' \preceq evs$
lemma *pre-equ-pre1*: $[\![evs \preceq evs';\ evs = evs'']\!] \Longrightarrow evs'' \preceq evs'$
lemma *pre-equ-pre2*: $[\![evs \preceq evs';\ evs' = evs'']\!] \Longrightarrow evs \preceq evs''$

axioms
RIpreI [*HOL.intro*!]: $[\![RI\ x;\ RI\ y]\!] \Longrightarrow RI\ x \preceq (y::'a\ list)$
RIpreE1: $(RI\ x \preceq (y::'a\ list)) \Longrightarrow RI\ x$
RIpreE2: $(RI\ x \preceq (y::'a\ list)) \Longrightarrow RI\ y$

CFpre [*HOL.intro*!]: $CF\ (x \preceq (y::'a\ list))$

theorem *lprerefl*: $<i,j>:\ evs \preceq evs$
theorem *lpretrans*: $[\![<i,j>:evs \preceq evs';\ <i,j>:evs' \preceq evs'']\!] \Longrightarrow <i,j>:\ evs \preceq evs''$
theorem *lpreanti*: $[\![<i,j>:\ evs \preceq evs';\ <i,j>:\ evs' \preceq evs]\!] \Longrightarrow <i,j>:\ evs = evs'$
theorem *lpreweaken1*: $<i,j>:\ evs = evs' \Longrightarrow <i,j>:\ evs \preceq evs'$
theorem *lpreweaken2*: $<i,j>:\ evs = evs' \Longrightarrow <i,j>:\ evs' \preceq evs$
theorem *lpresym*: $<i,j>:evs = evs' \Longrightarrow <i,j>:evs' = (evs::'a\ list)$

Miscellaneous Lemmas

lemma *lsubstr*: $[\![<i,j>:s=t;\ CF\ P(x);\ <i,j>:P(s)]\!] \Longrightarrow <i,j>:P(t::'a\ list)$
lemma *lsubstl*: $[\![<i,j>:t=s;\ CF\ P(x);\ <i,j>:P(s)]\!] \Longrightarrow <i,j>:P(t::'a\ list)$
lemma *lpre-equ-pre1*: $[\![<i,j>:evs \preceq evs';\ <i,j>:evs = evs'']\!] \Longrightarrow <i,j>:evs'' \preceq evs'$
lemma *lpre-equ-pre2*: $[\![<i,j>:evs \preceq evs';\ <i,j>:evs' = evs'']\!] \Longrightarrow <i,j>:evs \preceq evs''$
lemma $[\![RI\ a;\ RI\ b]\!] \Longrightarrow RI\ [a,\ b]$
lemma $RI\ [a,\ b] \Longrightarrow RI\ b$

end

A.6.3 Timed Properties of Global Traces

theory *TimedTrace* = *Schematic* + *TimedLists*:
axioms
RIequ-evtI [*intro*!]: [| *RI s*; *RI t* |] ==> (*RI* (*s*=(*t*::*event*)))
RIequ-evtE1: (*RI* (*s*=(*t*::*event*))) ==> *RI s*
RIequ-evtE2: (*RI* (*s*=(*t*::*event*))) ==> *RI t*

CFequ-evt [*intro*!]: *CF* (*e1* = (*e2*::*event*))

Flexible Constant Tr

consts
Tr :: *event list*

axioms
Tr-mono1: ⟦<*i*,*j*>: *Tr* = *evs*; <*i*,*j*>:*fwd*; *RI evs*⟧ ⟹ <*i*,*j*>:[*Lc*](*Tr* = *evs*)
Tr-mono2: ⟦<*i*,*j*>: *Tr* = *evs*; <*i*,*j*>:*fwd*; *RI evs*⟧ ⟹ <*i*,*j*>:[*R*](*evs* ≼ *Tr*)
Tr-mono3: ⟦<*i*,*j*>: *Tr* = *evs*; <*i*,*j*>:*fwd*; *RI evs*⟧ ⟹ <*i*,*j*>:[*Lc*][*L*](*Tr* ≼ *evs*)

theorem *Tr-ex*:
assumes *1*: <*i*,*j*>:*EX evs*. *Tr* = *evs* ⟹ <*i*,*j*>:*P*
shows <*i*,*j*>:*P*

theorem *Tr-preTr-expand*: ⟦*RI evs*⟧ ⟹ <*i*,*j*>:*evs* ≼ *Tr* ⟷ (∃*x*. *Tr* = *x* ∧ *evs* ≼ *x*)
apply (*rule liffI*)
apply (*rule Tr-ex*)
apply (*erule lexE*)
apply (*rule lexIRI*)
prefer *2*
apply (*rule lconjI*)
apply *assumption*
apply (*erule lsubstr*)
prefer *4*
apply (*erule lexE*)
apply (*erule lconjE*)
apply (*rule lsubstl*)
by (*assumption* | *rule CFpre*)+

theorem *Tr-Trpre-expand*: ⟦*RI evs*⟧ ⟹ <*i*,*j*>:*Tr* ≼ *evs* ⟷ (∃*x*. *Tr* = *x* ∧ *x* ≼ *evs*)

apply (*rule liffI*)
apply (*rule Tr-ex*)
apply (*erule lexE*)
apply (*rule lexIRI*)
prefer *2*
apply (*rule lconjI*)
apply *assumption*
apply (*rotate-tac 3*)
apply (*erule lsubstr*)
apply (*rule CFpre*)
apply *assumption*+
apply (*erule lexE*)
apply (*erule lconjE*)
apply (*erule lsubstl*)
by (*assumption* | *rule CFpre*)+

theorem *Tr-mono4*: $[\![<i,j>\text{: } Tr = evs;\ <i,j>\text{:}fwd;\ RI\ evs]\!] \Longrightarrow <i,j>\text{: } [S](Tr \preceq evs)$
apply (*frule Tr-mono3*)
apply *assumption*+
apply (*rule ilasubI*)
apply (*erule alnconvE*)
apply (*erule alnE*)
by *assumption*+

theorem *Tr-mono5*: $[\![<i,j>\text{: } Tr = evs;\ <i,j>\text{:}fwd;\ RI\ evs]\!] \Longrightarrow <i,j>\text{:}[PS]Tr \preceq evs$
apply (*frule Tr-mono3*)
apply *assumption*+
apply (*rule ilapsubI*)
apply (*erule alnconvE*)
apply (*erule alnE*)
by (*drule zlesslen-imp-fwd* | *assumption*)+

lemma *Tr-mono-pre1*: $[\![RI\ evs;\ <i,j>\text{: } evs \preceq Tr;\ <i,j>\text{:}fwd]\!] \Longrightarrow <i,j>\text{:}[Lc]evs \preceq Tr$

theorem *Tr-mono6*: $[\![<i,j>\text{: } Tr = evs;\ <i,j>\text{: } fwd;\ RI\ evs]\!] \Longrightarrow <i,j>\text{:}[R][Lc]evs \preceq Tr$
apply (*frule Tr-mono2*)
apply *assumption*+
apply (*rule arnI*)
apply *assumption*

apply (*rule alnconvI*)
apply (*drule arnE*)
prefer *3*
defer *1*
apply *assumption+*
apply (*drule Tr-mono-pre1*)
apply *assumption+*
apply (*erule alnconvE*)
by *assumption+*

Predicates Concerning Tr

constdefs
Tr-empty :: *bool*
Tr-empty ≡ *Tr* = []

Tr-throughout :: [*agent*, *event list*] ⇒ *bool*
Tr-throughout A evs ≡ *0* < *len* ∧ [*PS*](*Tr*↓*A*=*evs*)

Tr-causes :: [*agent*, *event*] ⇒ *bool*
Tr-causes A e ≡ *len* = *0* ∧ (∃ *evs*. *Tr*↓*A* = *e*#*evs* ∧ <*L*>*Tr-throughout A evs*)

Tr-quiet :: *agent* ⇒ *bool*
Tr-quiet A ≡ *len* = *0* ∧ ¬(∃ *e*. *Tr-causes A e*)

Tr-stable :: *agent* ⇒ *bool*
Tr-stable A ≡ ∃ *evs*. *Tr-throughout A evs*

Tr-occurs :: *event* ⇒ *bool*
Tr-occurs(*e*) ≡ ∃ *A*. *Tr-causes*(*A*)(*e*)

Tr-const :: *bool*
Tr-const ≡ ∀ *A*. *Tr-stable A*

axioms
Tr-lC: <*i*,*j*>:<*L*>*Tr-stable A*
Tr-rC: <*i*,*j*>:<*R*>*Tr-stable A*
Tr-silence: <*i*,*j*>: *Tr-quiet A* ⟹ <*i*,*j*>: *len* = *0* ∧ (∃ *evs*. *Tr*↓*A* = *evs* ∧ <*L*>*Tr-throughout A evs* ∧ <*R*>*Tr-throughout A evs*)

Tr-O-Snd-imp-C-Snd: ⟦<*i*,*j*>:*Tr-occurs* (*Snd A P*)⟧ ⟹ <*i*,*j*>:*Tr-causes A* (*Snd A P*)

Tr-O-Rcv-imp-C-Rcv: $[\![<i,j>:Tr\text{-}occurs\ (Rcv\ A\ P)]\!] \Longrightarrow <i,j>:Tr\text{-}causes\ A\ (Rcv\ A\ P)$
Tr-O-Blk-imp-C-Blk: $[\![<i,j>:Tr\text{-}occurs\ (Blk\ A\ P)]\!] \Longrightarrow <i,j>:Tr\text{-}causes\ A\ (Blk\ A\ P)$

Lemmas and Thorems Regarding Predicates

axioms
Tr-CS-CS-equ: $[\![<i,k>:Tr\text{-}causes\ A\ e;\ <m,n>:\ Tr\text{-}causes\ A\ e1;\ <k,j>:\ Tr\text{-}stable\ A;\ <n,j>:Tr\text{-}stable\ A]\!] \Longrightarrow <n,k>:len=0$
Tr-E-O-fwd: $[\![<i,k>:\ Tr\text{-}empty;\ <l,j>:\ Tr\text{-}occurs\ e]\!] \Longrightarrow <k,l>:\ fwd$
Tr-overlap1: $[\![<i,j>:\ Tr\text{-}throughout\ A\ evs;\ <k,j>:\ Tr\text{-}throughout\ A\ evsa;\ RI\ evs;\ RI\ evsa]\!] \Longrightarrow <i,j>:evs = evsa$
Tr-overlap2: $[\![<i,j>:\ Tr\text{-}throughout\ A\ evs;\ <i,k>:\ Tr\text{-}throughout\ A\ evsa;\ RI\ evs;\ RI\ evsa]\!] \Longrightarrow <i,j>:evs = evsa$
Tr-connected: $[\![<i,k>:\ [PS](Tr{\downarrow}A = evs);\ <k,l>:\ len = 0;\ <k,l>:Tr{\downarrow}A = evs;\ <l,j>:\ [PS](Tr{\downarrow}A = evs)]\!] \Longrightarrow <i,j>:[PS](Tr{\downarrow}A = evs)$

lemma *Tr-C-imp-O*: $[\![<i,j>:Tr\text{-}causes\ A\ P;\ RI\ A]\!] \Longrightarrow <i,j>:\ Tr\text{-}occurs\ P$
lemma *Tr-S-imp-zlesslen*: $<i,j>:\ Tr\text{-}stable\ A \Longrightarrow <i,j>:\ 0 < len$
lemma *pos-pos-inv-false*: $[\![0 < (x::real);\ 0 < y;\ x = -y]\!] \Longrightarrow False$

lemma *bwdless-fwdless-false*: $[\![<i,j>:\ 0 < len;\ <j,i>:\ 0 < len]\!] \Longrightarrow <i,j>:False$

lemma *Tr-silent-imp-S*: $[\![<i,j>:0 < len;\ <i,j>:Tr\text{-}stable\ A\ |\hat{}|\ Tr\text{-}quiet\ A\ |\hat{}|\ Tr\text{-}stable\ A]\!] \Longrightarrow <i,j>:\ Tr\text{-}stable\ A$
apply (*erule ilchopsubE*)+
apply (*simp only*: *Tr-stable-def*)
apply (*frule Tr-silence*)
apply ((*erule lexE*)+, (*erule lconjE*)+)+
apply (*erule slnE*)
apply *assumption*
apply (*erule srnE*)
apply *assumption*
apply (*frule Tr-overlap1*)
apply (*rotate-tac* -3)
apply *assumption*+
apply (*frule lsubstRI*)
apply (*rotate-tac 9*)
apply *assumption*
apply *assumption*
apply (*rotate-tac 6*)

```
 apply assumption
 apply (thin-tac <i,k>:evs=evsb)
 apply (rotate-tac 8)
 apply (frule Tr-overlap2)
 prefer 3
 apply (rotate-tac 1)
 apply assumption+
 apply (frule lsubstRI)
 prefer 2
 apply assumption+
 apply (unfold Tr-throughout-def)
 apply (erule lconjE)+
 apply (rule lexIRI)
 prefer 2
 apply (rule lconjI, assumption)
 apply (rule Tr-connected)
 defer 1
 apply assumption+
 done


end
```

A.6.4 Timed Properties of Local Traces

theory *TimedLTrace = LTrace + TimedLists*:

axioms
RIequ-actI [*intro*!]: [| *RI s*; *RI t* |] ==> (*RI* (*s*=(*t*::*action*)))
RIequ-actE1: (*RI* (*s*=(*t*::*action*))) ==> *RI s*
RIequ-actE2: (*RI* (*s*=(*t*::*action*))) ==> *RI t*
CFequ-act [*intro*!]: *CF* (*e1* = (*e2*::*action*))

RIver [*intro*!]: ⟦*RI P*⟧ ⟹ *RI* (*Ver P*)

Flexible Constant lTr

consts
lTr :: *agent* ⇒ *action list*

axioms
lTr-mono1: ⟦<*i*,*j*>: *lTr A* = *acs*; <*i*,*j*>:*fwd*; *RI acs*⟧ ⟹ <*i*,*j*>: [*Lc*](*lTr A* = *acs*)
lTr-mono2: ⟦<*i*,*j*>: *lTr A* = *acs*; <*i*,*j*>:*fwd*; *RI acs*⟧ ⟹ <*i*,*j*>: [*R*](*acs* ≼ *lTr A*)
lTr-mono3: ⟦<*i*,*j*>: *lTr A* = *acs*; <*i*,*j*>:*fwd*; *RI acs*⟧ ⟹ <*i*,*j*>: [*Lc*][*L*](*lTr A* ≼ *acs*)

theorem *lTr-ex*:
assumes *1*: <*i*,*j*>:*EX acs*. *lTr A* = *acs* ⟹ <*i*,*j*>:*P*
shows <*i*,*j*>:*P*

theorem *lTr-prelTr-expand*: ⟦*RI acs*⟧ ⟹ <*i*,*j*>:*acs* ≼ *lTr A* ⟷ (∃ *x*. *lTr A* = *x* ∧ *acs* ≼ *x*)
apply (*rule liffI*)
apply (*rule lTr-ex*)
apply (*erule lexE*)
apply (*rule lexIRI*)
prefer *2*
apply (*rule lconjI*)
apply *assumption*
apply (*erule lsubstr*)
prefer *4*
apply (*erule lexE*)
apply (*erule lconjE*)
apply (*rule lsubstl*)

by (*assumption* | *rule CFpre*)+

theorem *lTr-lTrpre-expand*: ⟦*RI acs*⟧ ⟹ <*i*,*j*>:*lTr A* ≼ *acs* ⟷ (∃ *x*. *lTr A* = *x* ∧ *x* ≼ *acs*)
apply (*rule liffI*)
apply (*rule lTr-ex*)
apply (*erule lexE*)
apply (*rule lexIRI*)
prefer *2*
apply (*rule lconjI*)
apply *assumption*
apply (*rotate-tac 3*)
apply (*erule lsubstr*)
apply (*rule CFpre*)
apply *assumption*+
apply (*erule lexE*)
apply (*erule lconjE*)
apply (*erule lsubstl*)
by (*assumption* | *rule CFpre*)+

theorem *lTr-mono4*: ⟦<*i*,*j*>: *lTr A* = *acs*; <*i*,*j*>:*fwd*; *RI acs*⟧ ⟹ <*i*,*j*>: [*S*](*lTr A* ≼ *acs*)
apply (*frule lTr-mono3*)
apply *assumption*+
apply (*rule ilasubI*)
apply (*erule alnconvE*)
apply (*erule alnE*)
by *assumption*+

theorem *lTr-mono5*: ⟦<*i*,*j*>: *lTr A* = *acs*; <*i*,*j*>:*fwd*; *RI acs*⟧ ⟹ <*i*,*j*>:[*PS*]*lTr A* ≼ *acs*
apply (*frule lTr-mono3*)
apply *assumption*+
apply (*rule ilapsubI*)
apply (*erule alnconvE*)
apply (*erule alnE*)
by (*drule zlesslen-imp-fwd* | *assumption*)+

lemma *lTr-mono-pre1*: ⟦*RI acs*; <*i*,*j*>: *acs* ≼ *lTr A*; <*i*,*j*>:*fwd*⟧ ⟹ <*i*,*j*>:[*Lc*]*acs* ≼ *lTr A*

theorem *lTr-mono6*: ⟦<*i*,*j*>: *lTr A* = *acs*; <*i*,*j*>: *fwd*; *RI acs*⟧ ⟹ <*i*,*j*>:[*R*][*Lc*]*acs* ≼ *lTr A*

apply (*frule lTr-mono2*)
apply *assumption+*
apply (*rule arnI*)
apply *assumption*
apply (*rule alnconvI*)
apply (*drule arnE*)
prefer *3*
defer *1*
apply *assumption+*
apply (*drule lTr-mono-pre1*)
apply *assumption+*
apply (*erule alnconvE*)
by *assumption+*

Predicates Concerning lTr

constdefs
lTr-empty :: *agent* ⇒ *bool*
lTr-empty A ≡ *lTr A* = []

lTr-throughout :: *agent* ⇒ (*action list* ⇒ *bool*)
lTr-throughout A acs ≡ *0* < *len* ∧ [*PS*](*lTr A* = *acs*)

lTr-performs :: *agent* ⇒ (*action* ⇒ *bool*)
lTr-performs A a ≡ *len* = *0* ∧ (∃ *acs*. *lTr A* = *a*#*acs* ∧ <*L*>(*lTr-throughout A acs*))

lTr-idles :: *agent* ⇒ *bool*
lTr-idles A ≡ *0* = *len* ∧ ¬(∃ *a*. *lTr-performs A a*)

lTr-stable :: *agent* ⇒ *bool*
lTr-stable A ≡ ∃ *acs*. *lTr-throughout A acs*

axioms

lTr-lC: <*i*,*j*>:<*L*>*lTr-const*(*A*)
lTr-rC: <*i*,*j*>:<*R*>*lTr-const*(*A*)
lTr-idling: <*i*,*j*>:*lTr-idles A* ⟹ <*i*,*j*>: *len* = *0* ∧ (∃ *acs*. *lTr A* = *acs* ∧ <*L*>(*lTr-throughout A acs*) ∧ <*R*>(*lTr-throughout A acs*))

Lemmas and Thorems Regarding Predicates

axioms
lTr-PS-PS-zlen:⟦<*i*,*k*>:*lTr-performs A a*; <*m*,*n*>: *lTr-performs A a1*; <*k*,*j*>: *lTr-stable A*; <*n*,*j*>:*lTr-stable A*⟧ ⟹ <*n*,*k*>:*len=0*

lTr-SP-SP-zlen:⟦<*k*,*l*>:*lTr-performs A a*; <*m*,*n*>: *lTr-performs A a1*; <*i*,*k*>: *lTr-stable A*; <*i*,*m*>:*lTr-stable A*⟧ ⟹ <*n*,*k*>:*len=0*

lTr-P-O-fwd: ⟦<*i*,*k*>: *lTr-empty A*; <*l*,*j*>: *lTr-performs A e*⟧ ⟹ <*k*,*l*>:*fwd*

lTr-overlap1: ⟦<*i*,*j*>: *lTr-throughout A acs*; <*k*,*j*>: *Tr-throughout A acs1*; *RI acs*; *RI acs1*⟧ ⟹ <*i*,*j*>:*acs* = *acs1*

lTr-overlap2: ⟦<*i*,*j*>: *lTr-throughout A acs*; <*i*,*k*>: *lTr-throughout A acs1*; *RI acs*; *RI acs1*⟧ ⟹ <*i*,*j*>:*acs* = *acs1*

lTr-connected: ⟦<*i*,*k*>: [*PS*](*lTr A* = *acs*); <*k*,*l*>: *len* = *0*; <*k*,*l*>:*lTr A* = *acs*; <*l*,*j*>: [*PS*](*lTr A* = *acs*)⟧ ⟹ <*i*,*j*>:[*PS*](*lTr A* = *acs*)

lemma *lTr-S-imp-zlesslen*: <*i*,*j*>: *lTr-stable A* ⟹ <*i*,*j*>: *0* < *len*
lemma *pos-pos-inv-false*: ⟦*0* < (*x*::*real*); *0* < *y*; *x* = −*y*⟧ ⟹ *False*

lemma *bwdless-fwdless-false*: ⟦<*i*,*j*>: *0* < *len*; <*j*,*i*>: *0* < *len*⟧ ⟹ <*i*,*j*>:*False*

lemma *lTr-idling-imp-S*: ⟦<*i*,*j*>:*0* < *len*; <*i*,*j*>:*lTr-stable A* |^| *lTr-idles A* |^| *lTr-stable A*⟧ ⟹ <*i*,*j*>: *lTr-stable A*
apply (*erule ilchopsubE*)+
apply (*simp only*: *lTr-stable-def*)
apply (*frule lTr-idling*)
apply ((*erule lexE*)+, (*erule lconjE*)+)+
apply (*erule slnE*)
apply *assumption*
apply (*erule srnE*)
apply *assumption*
apply (*frule lTr-overlap1*)
apply (*rotate-tac* −*3*)
apply *assumption*+
apply (*frule lsubstRI*)
apply (*rotate-tac 9*)
apply *assumption*
apply *assumption*
apply (*rotate-tac 6*)

apply *assumption*
apply (*thin-tac* <*i*,*k*>:*acs*=*acsb*)
apply (*rotate-tac 8*)
apply (*frule lTr-overlap2*)
prefer *3*
apply (*rotate-tac 1*)
apply *assumption*+
apply (*frule lsubstRI*)
prefer *2*
apply *assumption*+
apply (*unfold lTr-throughout-def*)
apply (*erule lconjE*)+
apply (*rule lexIRI*)
prefer *2*
apply (*rule lconjI*, *assumption*)
apply (*rule lTr-connected*)
defer *1*
apply *assumption*+
done

Modelling Verification Functions

lemma *lTr-P-Ver-True*: ⟦<*i*,*j*>:*lTr-performs A* (*Ver k l P*); *P*⟧ ⟹ <*i*,*j*>:*lTr-performs A* (*Ver k l True*)
lemma *lTr-P-Ver-False*: ⟦<*i*,*j*>:*lTr-performs A* (*Ver k l P*); ¬*P*⟧ ⟹ <*i*,*j*>:*lTr-performs A* (*Ver k l False*)
end

A.6.5 Modeling Heuristics

theory *Timed = TimedTrace + TimedLTrace*:

Dependency versus Obligation

Universal versus Specific Properties

axioms

Tr-init: $<i,j>: <A>Tr\text{-}empty$

lTr-init: $<i,j>: <A>(lTr\text{-}empty\ A)$

lTr-Ready: $<i,j>: lTr\text{-}empty\ A \Longrightarrow <i,j>:<R>(lTr\text{-}stable\ A\ |\hat{}|\ lTr\text{-}performs\ A\ Ready)$

Initiate: $[\![<i,j>: lTr\text{-}performs\ A\ (SState\ 1)]\!] \Longrightarrow <i,j>: <L>(lTr\text{-}performs\ A\ Ready\ |\hat{}|\ lTr\text{-}stable\ A)$

Respond: $[\![<i,j>: lTr\text{-}performs\ A\ (RState\ 1)]\!] \Longrightarrow <i,j>: <L>(lTr\text{-}performs\ A\ Ready\ |\hat{}|\ lTr\text{-}stable\ A)$

Abort: $<i,j>: lTr\text{-}performs\ A\ Abort \Longrightarrow <i,j>: <R>lTr\text{-}stable\ A\ |\hat{}|\ lTr\text{-}performs\ A\ Ready$

Accept: $<i,j>: lTr\text{-}performs\ A\ Accept \Longrightarrow <i,j>: <R>lTr\text{-}stable\ A\ |\hat{}|\ lTr\text{-}performs\ A\ Ready$

theorem *Tr-C-uniq*: $[\![<i,j>: Tr\text{-}causes\ A\ e1; <i,j>:Tr\text{-}causes\ A\ e2]\!] \Longrightarrow <i,j>: e1 = e2$

apply (*unfold Tr-causes-def*)
apply (*erule lconjE*)+
apply (*erule lexE*)+
apply (*erule lconjE*)+
apply (*subgoal-tac* $<i,j> : Tr \downarrow A = e1\ \#\ evs \wedge Tr \downarrow A = e2\ \#\ evsa \longrightarrow e1=e2$)
apply (*tactic LFast-tac 1*)
apply (*rule labelICF*)
apply *auto*
apply *rule*+
apply (*rule TimedLists.CFequ*)+
by (*rule CFequ-evt*)

theorem *lTr-P-uniq*: $[\![<i,j>: lTr\text{-}performs\ A\ a1; <i,j>:lTr\text{-}performs\ A\ a2]\!] \Longrightarrow <i,j>: a1 = a2$

apply (*unfold lTr-performs-def*)
apply (*erule lconjE*)+

apply (*erule lexE*)+
apply (*erule lconjE*)+
apply (*tactic LSimp-tac 1*)
apply (*subgoal-tac* <*i*,*j*> : *lTr A* = *a1* # *acs* ∧ *lTr A* = *a2* # *acsa* ⟶ *a1*=*a2*)
apply (*tactic LFast-tac 1*)
apply (*rule labelICF*)
apply *auto*
apply *rule*+
apply (*rule TimedLists.CFequ*)+
by (*rule CFequ-act*)

end

A.7 Example: Simple Protocols

A.7.1 Network Delays

theory *Simple* = *Timed*:

Creating a Model

consts
delta :: *real*
epsilon :: *real*

syntax
@*delta* :: *'a* ⇒ *real* (δ)
@*epsilon* :: *'a* ⇒ *real* (ε)

translations
δ == *delta*
ε == *epsilon*

axioms
RIdelta [*intro*!]: *RI delta*
RIepsilon [*intro*!]: *RI epsilon*

consts
M1 :: *msg*
M2 :: *msg*
M3 :: *msg*

axioms

Rcv: ⟦<*i*,*j*>: *Tr-occurs*(*Rcv B* (|(*A*,*A*)→(*B*,*B*):*m*|)); *A* ≠ *B*⟧ ⟹
<*i*,*j*>: <*L*>((*Tr-occurs*(*Snd A* (|(*A*,*A*)→(*B*,*B*):*m*|)) ∧ *Tr-quiet B*)
|^| (*Tr-stable A* ∧ *Tr-stable B*))

Blk: ⟦<*i*,*j*>: *Tr-occurs*(*Blk B* (|(*A*,*A*)→(*B*,*B*):*m*|)); *A* ≠ *B*⟧ ⟹
<*i*,*j*>: <*L*>((*Tr-occurs*(*Rcv B* (|(*A*,*A*)→(*B*,*B*):*m*|)) ∧ *Tr-quiet A*)
|^| (*len* ≤ ε ∧ *Tr-stable A* ∧ *Tr-stable B*))

Snd1: ⟦<*i*,*j*>: *Tr-occurs*(*Snd A* (|(*A*,*A*)→(*B*,*B*):*M1*|)); *A* ≠ *B*⟧ ⟹
<*i*,*j*>: <*L*>(*Tr-empty*) ∨
<*L*>((*Tr-occurs*(*Snd A* (|(*A*,*A*)→(*C*,*C*):*M3*|)) ∧ *Tr-quiet C*)
|^| (*Tr-stable A* ∧ *Tr-stable C*)) ∨
<*L*>((*Tr-occurs*(*Blk A* (|(*C*,*C*)→(*A*,*A*):*M3*|)) ∧ *Tr-quiet C*)
|^| (*Tr-stable A* ∧ *Tr-stable C*))

Snd2: $[\![<i,j>: Tr\text{-}occurs(Snd\ B\ (\!|(B,B)\rightarrow(A,A):M2|\!));\ A \neq B]\!] \Longrightarrow$
$<i,j>: <L>((Tr\text{-}occurs(Blk\ B\ (\!|(A,A)\rightarrow(B,B):M1|\!)) \wedge Tr\text{-}quiet\ A)$
$|\hat{}|\ (Tr\text{-}stable\ B \wedge Tr\text{-}stable\ A))$

Snd3: $[\![<i,j>: Tr\text{-}occurs(Snd\ A\ (\!|(A,A)\rightarrow(B,B):M3|\!));\ A \neq B]\!] \Longrightarrow$
$<i,j>: <L>((Tr\text{-}occurs(Blk\ A\ (\!|(B,B)\rightarrow(A,A):M2|\!)) \wedge Tr\text{-}quiet\ B)$
$|\hat{}|\ (Tr\text{-}stable\ A \wedge Tr\text{-}stable\ B))$

axioms

delay: $[\![<i,j>: (Tr\text{-}occurs(Snd\ A\ P)\ |\hat{}|\ True\ |\hat{}|\ Tr\text{-}occurs(Rcv\ B\ P));\ A \neq B]\!] \Longrightarrow <i,j>: \delta <= len$

Proving Properties

lemma *snd1-snd2-imp-delay*:
$[\![A \neq B;\ RI\ A;\ RI\ B]\!] \Longrightarrow$
$<i,j>: Tr\text{-}causes\ A\ (Snd\ A\ (\!|(A,A)\rightarrow(B,B):M1|\!))|\hat{}|\ Tr\text{-}stable\ A\ |\hat{}|\ Tr\text{-}causes\ B\ (Snd\ B\ (\!|(B,B)\rightarrow(A,A):M2|\!)) \longrightarrow$
$\delta <= len$

apply (*rule limpI*)
apply (*erule ilchopsubE*)+
apply (*drule Tr-C-imp-O [THEN Snd2]*, *assumption*+)
apply (*elim slnE*, *assumption*, *erule ilchopsubE*, (*erule lconjE*)+)
apply (*drule Blk*, *assumption*+)
apply (*elim slnE*, *assumption*, *erule ilchopsubE*, (*erule lconjE*)+)
apply (*frule Rcv*, *assumption*+)
apply (*elim slnE*, *assumption*, *erule ilchopsubE*, (*erule lconjE*)+)
apply (*subgoal-tac* $<ke,ka>: Tr\text{-}stable\ A$)
prefer *2*
apply (*rule Tr-silent-imp-S*)
prefer *2*
apply (*rule ilchopsubI*)
prefer *3*
apply *assumption*+
apply (*rule ilchopsubI*)
prefer *3*
apply *assumption*+
defer *1*
apply (*subgoal-tac* $<kg,ka>: Tr\text{-}stable\ A$)
prefer *2*
apply (*rule Tr-silent-imp-S*)
prefer *2*

apply (*rule ilchopsubI*)
prefer *3*
apply *assumption*+
defer *1*
apply (*rule ilchopsubI*)
prefer *3*
apply *assumption*+
prefer *2*
apply *assumption*
prefer *3*
apply (*frule Tr-CS-CS-equ*)
apply (*drule Tr-O-Snd-imp-C-Snd*)+
apply (*rotate-tac −1*)
apply *assumption*+
apply (*drule Tr-C-imp-O*, *assumption*)+

defer *1*
apply (*rule fwd-fwd*)
apply *assumption*+
prefer *3*
apply (*rule fwd-fwd*)
apply *assumption*+
prefer *2*
apply (*rule zlesslen-fwd-imp-zlesslen*)
prefer *2*
apply (*rule Tr-S-imp-zlesslen*)
apply *assumption*+
apply (*rule zlesslen-fwd-imp-zlesslen*)
apply (*rule fwd-fwd*)
prefer *2*
apply (*rotate-tac 9*)
apply *assumption*+
apply (*rule Tr-S-imp-zlesslen*)
apply *assumption*

apply (*rule ssub-lelen-imp-lelen*)
apply *rule*
apply (*rule ilssubI*)
prefer *2*
apply (*rule delay*)
apply (*rule ilchopsubI*)
prefer *3*
apply *assumption*

```
  prefer 3
  apply (rule ilchopsubI)
  prefer 4
  apply assumption+
  apply (rule lTrueI)
  apply assumption+
  apply (rule fwd-fwd)
  apply assumption+
  defer 1
  apply (rule fwd-fwd)
  apply assumption+
  apply (rule fwd-fwd [THEN fwd-fwd])
  apply assumption+
  apply (rotate-tac -8)
  apply (simp only: Tr-occurs-def Tr-causes-def)
  apply (erule lexE)
  apply (rotate-tac -1)
  apply (erule lconjE)
  apply (drule zeropoint)
  apply (rotate-tac -3)
  apply (drule zeropoint)
  apply (drule point-fwd)+
  apply (rule fwd-fwd [THEN fwd-fwd])
  prefer 3
  apply assumption+
  done


lemma M2-imp-M1:
  [A≠B; <i,j>: fwd] ⟹
     <i,j>: Tr-occurs(Snd A (|(A,A)→(B,B):M2|)) ⟶
          <L>(Tr-occurs(Snd B (|(B,B)→(A,A):M1|)) |^| True)
  apply (rule limpI)
  apply (drule Snd2)
  apply (rule not-sym)
  apply assumption
  apply (erule slnE, assumption)
  apply (erule ilchopsubE)
  apply (erule lconjE)+
  apply (drule Blk)
  apply (rule not-sym)
  apply assumption
  apply (erule slnE, assumption)
```

apply (*erule ilchopsubE*)
apply (*erule lconjE*)+
apply (*drule Rcv*)
apply (*rule not-sym*)
apply *assumption*
apply (*erule slnE*, *assumption*)
apply (*erule ilchopsubE*)
apply (*erule lconjE*)
apply (*rule slnI*, *assumption*)
prefer *2*
apply (*rule ilchopsubI*)
prefer *3*
apply *assumption*+
apply (*rule fwd-fwd*, *assumption*)
apply (*rule fwd-fwd*)
prefer *2*
apply *assumption*+
apply (*rule lTrueI*)
apply (*rule fwd-fwd* [*THEN fwd-fwd*])
prefer *3*
apply *assumption*
prefer *2*
by *assumption*+

lemma *ilchop-chop-sub*:
⟦<*i*, *k*>: *P* |^| *Q*; <*i*, *k*>:*fwd*; <*k*, *j*>: *fwd*; <*k*, *j*>: *R*⟧ ⟹
<*i*, *j*>: *P* |^| *Qă*|^| *R*
lemma *fwd-grow*:
⟦<*k*, *j*>: *l* <= *len*; <*k*, *j*>: *fwd*; <*i*, *k*>: *fwd*; *RI l*⟧ ⟹
<*i*, *j*>: *l* <= *len*
lemma *E-Rcv-delay*:
⟦<*i*,*j*>: *fwd*; *A*≠*B*; *Ri δ*⟧ ⟹
<*i*,*j*>: *Tr-empty* |^| *True* |^| *Tr-occurs*(*Rcv B* (|(*A*,*A*)→(*B*,*B*):*M1*|)) ⟶ *δ* <= *len*
apply (*rule limpI*)
apply (*erule ilchopsubE*)
apply (*erule ilchopsubE*)
apply (*rotate-tac 8*)
apply (*frule Rcv*)
apply *assumption*+
apply (*erule slnE*)
apply *assumption*

apply (*rotate-tac 9*)
apply (*erule ilchopsubE*)+
apply (*erule lconjE*)+
apply (*rotate-tac −6*)
apply (*frule ilchopsubI*)
prefer *2*
apply (*rotate-tac 1*)
apply *assumption*
apply *assumption*
apply (*subgoal-tac <kc,ka>:True*)
apply (*rotate-tac −1*)
apply *assumption*
apply (*rule lTrueI*)
apply (*rotate-tac −11*)
apply (*drule ilchop-chop-sub*)
apply *assumption*
apply *assumption*
apply (*rotate-tac −1*)
apply *assumption*
apply (*drule delay, assumption*)
apply (*drule Tr-E-O-fwd*)
apply *assumption*
apply (*drule fwd-grow*)
apply (*rule fwd-fwd*)
apply *assumption*+
apply *rule*
apply (*drule fwd-grow*)
apply (*rule fwd-fwd*)
apply *assumption*+
apply *rule*
by *assumption*

lemma *asub-Rcv*:
⟦<*i*,*j*>:*fwd*; *A* ≠ *B*⟧ ⟹
<*i*,*j*>:[*S*](*Tr-occurs*(*Rcv B* (|(*A*,*A*)→(*B*,*B*):*m*|)) ⟶
<*L*>((*Tr-occurs*(*Snd A* (|(*A*,*A*)→(*B*,*B*):*m*|)) ∧ *Tr-quiet B*) | ^ | (*Tr-stable A* ∧ *Tr-stable B*)))

lemma *asub-Snd2*:
⟦<*i*,*j*>:*fwd*; *A* ≠ *B*⟧ ⟹
<*i*,*j*>:[*S*](*Tr-occurs*(*Snd B* (|(*B*,*B*)→(*A*,*A*):*M2*|)) ⟶
<*L*>((*Tr-occurs*(*Blk B* (|(*A*,*A*)→(*B*,*B*):*M1*|)) ∧ *Tr-quiet A*) | ^ | (*Tr-stable B* ∧ *Tr-stable A*)))

lemma *asub-Snd3*:
$[\![<i, j>: fwd;\ A \neq B]\!] \Longrightarrow$
$<i, j>: [S](Tr\text{-}occurs(Snd\ A\ (\!|(A,A) \rightarrow (B,B):M3|\!)) \longrightarrow$
$<L>((Tr\text{-}occurs(Blk\ A\ (\!|(B,B) \rightarrow (A,A):M2|\!)) \wedge Tr\text{-}quiet\ B)\ |\ ^\wedge |\ (Tr\text{-}stable\ A \wedge Tr\text{-}stable\ B)))$
end

A.7.2 Rejecting Bogus Messages

theory *Advanced = Timed:*

Creating a Model

consts
Delta :: *nat* ⇒ *real*
delta :: *nat* ⇒ (*nat* ⇒ *real*)
epsilon :: *real*

syntax
@*delta* :: *'a* ⇒ *real* (δ)
@*Delta* :: *'a* ⇒ *real* (Δ)
@*epsilon* :: *'a* ⇒ *real* (ε)

translations
δ == *delta*
ε == *epsilon*
Δ == *Delta*

axioms
RIDelta [*HOL.intro*!]: *RI* (Δ *i*)
RIdelta [*HOL.intro*!]: *RI* (δ *i j*)
RIepsilon [*HOL.intro*!]: *RI* ε

consts
M :: *nat* ⇒ *msg*

axioms
SState1: ⟦<*i*,*j*>: *lTr-performs A* (*SState 1*); *A* ≠ *B*⟧ ⟹
<*i*,*j*>: <*R*>((*lTr-stable A* ∧ *len* = Δ(*1*)) |^| *Tr-causes A* (*Snd A* (|(*A*,*A*)→(*B*,*B*):*M 1*|))) ∧
<*R*>((*lTr-stable A* ∧ *len* = Δ(*1*)) |^| *lTr-performs A* (*RState 2*))

SState2: ⟦<*i*,*j*>: *lTr-performs A* (*SState 2*); *A* ≠ *B*⟧ ⟹
<*i*,*j*>: <*R*>((*lTr-stable A* ∧ *len* = Δ(*2*)) |^| *Tr-occurs*(*Snd A* (|(*A*,*A*)→(*B*,*B*):*M 2*|))) ∧
<*R*>((*lTr-stable A* ∧ *len* = Δ(*2*)) |^| *lTr-performs A Accept*)

axioms
SuccVer1:⟦<*i*,*j*>: *lTr-performs A* (*Ver 1 1 True*) ⟧ ⟹

$<i,j>$: $<R>((lTr\text{-}stable\ A \wedge len = \varepsilon)\ |\hat{}|\ lTr\text{-}performs\ A\ (SState\ 2))$

SuccVer2: $[\![<i,j>$: $lTr\text{-}performs\ A\ (Ver\ 2\ 1\ True)\ ]\!] \Longrightarrow$
$<i,j>$: $<R>((lTr\text{-}stable\ A \wedge len = \varepsilon)\ |\hat{}|\ lTr\text{-}performs\ A\ Accept)$

FailVer: $[\![<i,j>$: $lTr\text{-}performs\ A\ (Ver\ k\ l\ False)]\!] \Longrightarrow$
$<i,j>$: $<R>((lTr\text{-}stable\ A \wedge len = \varepsilon)\ |\hat{}|\ lTr\text{-}performs\ A\ Abort)$

axioms

Rcv: $[\![<i,j>$: $Tr\text{-}causes\ A\ (Rcv\ A\ (\!|(B,B)\rightarrow(A,A)\text{: } M\ k|\!));\ A \neq B]\!] \Longrightarrow$
$<i,j>$: $<L>(lTr\text{-}performs\ A\ (RState\ k)\ |\hat{}|\ lTr\text{-}stable\ A) \wedge$
$<L>((Tr\text{-}causes\ B\ (Snd\ B\ (\!|(B,B)\rightarrow(A,A)\text{: } M\ k|\!)) \wedge Tr\text{-}quiet\ A)\ |\hat{}|\ (Tr\text{-}stable\ A \wedge Tr\text{-}stable\ B)) \wedge$
$<R>((lTr\text{-}stable\ A \wedge len = \delta(k)(1))\ |\hat{}|\ (lTr\text{-}performs\ A\ (Ver\ k\ 1\ (chkAgent\ B))))$

Proving Properties

lemma *Tr-C-imp-zlen*: $<i,j>$: $Tr\text{-}causes\ A\ e \Longrightarrow <i,j>$: $len = 0$
lemma *lTr-P-imp-zlen*: $<i,j>$: $lTr\text{-}performs\ A\ e \Longrightarrow <i,j>$: $len = 0$
lemma *zero-zero-same*: $[\![<i,j>$:P; $<i,j>$:$len = 0$; $<k,l>$:$len = 0$; $<j,l>$:$len = 0]\!] \Longrightarrow <k,l>$:$P$

theorem *Rcv-bogus-imp-delay-Abort*: $[\![<i,j>$: $Tr\text{-}causes\ A\ (Rcv\ A\ (\!|(Spy,Spy)\rightarrow(A,A)\text{: } M\ 1|\!));\ <i,j>$:$fwd$; $A \neq Spy]\!] \Longrightarrow <i,j>$: $<R>(len = (\delta\ 1\ 1)+\varepsilon\ |\hat{}|\ lTr\text{-}performs\ A\ Abort)$
apply (*drule Rcv*, *assumption*)
apply (*erule lconjE*)+
apply (*erule srnE*, *assumption*)
apply (*erule ilchopsubE*)+
apply *simp*
apply (*frule lTr-P-imp-zlen*)
apply (*drule FailVer*)
apply (*erule srnE*, *assumption*)
apply (*erule ilchopsubE*)
apply (*rule srnI*, *assumption*)
prefer *2*
apply (*rule ilchopsubI*)
prefer *4*
apply *assumption*
apply *simp-all*
apply (*rule fwd-fwd*, *assumption*+)
prefer *2*

apply (*rule fwd-fwd*, *assumption*+)
apply (*erule lconjE*)+
apply (*rule plusI*)
apply *assumption*
apply (*drule plusI*, *assumption*)
by *auto*

theorem *Rcv-bogus-imp-no-Accept*: ⟦<*i*,*j*>: *Tr-causes A* (*Rcv A* (|(*Spy*,*Spy*)→(*A*,*A*): *M 1*|)) |^| *lTr-stable A* |^| *lTr-performs A* (*Ver 1 1 True*); <*i*,*j*>:*fwd*; *A* ≠ *Spy*⟧
⟹ <*i*,*j*>: *False*
apply (*erule ilchopsubE*)+
apply (*frule Rcv*)
apply *assumption*
apply (*erule lconjE*)+
apply (*erule srnE*)
apply *assumption*
apply (*erule ilchopsubE*)+
apply (*erule lconjE*)
apply (*rotate-tac* −*5*)
apply (*frule lTr-SP-SP-zlen*)
apply (*rotate-tac 1*)
apply *assumption*+
apply (*rotate-tac* −*4*)
apply (*frule lTr-P-imp-zlen*)
apply (*rotate-tac* −*4*)
apply (*frule lTr-P-imp-zlen*)
apply (*frule zero-zero-same*)
apply *assumption*
apply *assumption*
prefer *2*
apply *simp*
apply (*rotate-tac* −*1*)
apply (*drule lTr-P-uniq*)
apply *simp*
apply (*rule rigid*)
apply *simp*
apply *rule*
apply (*rule zeroE1*)
by *assumption*+

end

Appendix B

NLDCHOL

The following is a listing of the complete axiomatic foundation of NLDCHOL as it is encoded in Isabelle/HOL by Rasmussen.

B.1 LFOLHOL

LFOLHOL = Real +

global

types
  T
arities
  T :: logic

consts
  LF :: [*T*, *T*, *bool*] => *prop* ((<-,-> : (-)) [6,6,5] 4)

  RI :: 'a::*logic* => *prop* ((*RI* -))
  CF :: *bool* => *prop* ((*CF* -))

  <-> :: [*bool*, *bool*] => *bool* (*infixr* 25)

syntax (*xsymbols*)
  op <-> :: [*bool*, *bool*] => *bool* (*infixr* $\longleftrightarrow$ 25)

local

defs
  iff-def $P <-> Q == (P --> Q) \,\&\, (Q --> P)$

```
rules
  lconjI      [| <i,j>:P;  <i,j>:Q |] ==> <i,j>:P&Q
  lconjE1      <i,j>:P&Q ==> <i,j>:P
  lconjE2      <i,j>:P&Q ==> <i,j>:Q

  ldisjI1     <i,j>:P ==> <i,j>:P|Q
  ldisjI2     <i,j>:Q ==> <i,j>:P|Q
  ldisjE      [| <i,j>:P|Q; <i,j>:P ==> <k,l>:R; <i,j>:Q ==> <k,l>:R |]
                 ==> <k,l>:R

  limpI        (<i,j>:P ==> <i,j>:Q) ==> <i,j>:P-->Q
  lmp          [| <i,j>:P-->Q;  <i,j>:P |] ==> <i,j>:Q

  lTrueI       <i,j>:True
  lFalseE      (<i,j>:P-->False ==> <k,l>:False) ==> <i,j>:P

  lallI       (!!x::'a. (RI x) ==> <i,j>:(P x)) ==> <i,j>:(ALL x. (P x))
  lallERI      [| RI s; <i,j>:(ALL x::'a. (P x)) |] ==> <i,j>:(P s)
  lallECF      [| CF (P x); <i,j>:(ALL x::'a. (P x)) |] ==> <i,j>:(P s)

  lexIRI       [| RI s; <i,j>:(P s) |] ==> <i,j>:(EX x::'a. (P x))
  lexICF       [| CF (P x); <i,j>:(P s) |] ==> <i,j>:(EX x::'a. (P x))
  lexE         [| <i,j>:EX x::'a. (P x); !!x. [| RI x; <i,j>:(P x) |]
                 ==> <k,l>:R |] ==> <k,l>:R

  lrefl       <i,j>:s=(s::'a)
  lsubst       [| CF (P x); <i,j>:s=t; <i,j>:(P s) |] ==> <i,j>:(P (t::'a))
  lsubstRI     [| RI s; RI t; <i,j>:s=t; <i,j>:(P s) |] ==> <i,j>:(P (t::'a))

  (* Order axioms *)

  orefl        <i,j>:s<=s
  otran        [| <i,j>:s<=t; <i,j>:t<=u |] ==> <i,j>:s<=u
  oanti        [| <i,j>:s<=t; <i,j>:t<=s |] ==> <i,j>:s=t
  oless        <i,j>:s<t <-> s<=t & s~=t

  (* Labels *)

  labelIRI   [| RI P; P |] ==> <i,j>:P
  labelICF    [| CF P; P |] ==> <i,j>:P
  labelE     (!!i j. <i,j>:P) ==> P
```

(* *Rigidity and Chopfreeness* *)

rigid [| <*k*,*l*>:*P*; *RI P* |] ==> <*i*,*j*>:*P*

RItrue *RI True*
RIfalse *RI False*
RIzero *RI 0*
RIone *RI 1*
RIconjI [| *RI P*; *RI Q* |] ==> (*RI P*&*Q*)
RIconjE1 (*RI P*&*Q*) ==> *RI P*
RIconjE2 (*RI P*&*Q*) ==> *RI Q*
RIdisjI [| *RI P*; *RI Q* |] ==> (*RI P*|*Q*)
RIdisjE1 (*RI P*|*Q*) ==> *RI P*
RIdisjE2 (*RI P*|*Q*) ==> *RI Q*
RIimpI [| *RI P*; *RI Q* |] ==> (*RI P*−−>*Q*)
RIimpE1 (*RI P*−−>*Q*) ==> *RI P*
RIimpE2 (*RI P*−−>*Q*) ==> *RI Q*
RIallI ((*RI s*) ==> *RI P*(*s*)) ==> *RI* (*ALL x*::'*a*. *P*(*x*))
RIallE [| *RI s*; *RI* (*ALL x*. *P*(*x*)) |] ==> *RI P*(*s*::'*a*)
RIexI ((*RI s*) ==> *RI P*(*s*)) ==> *RI* (*EX x*::'*a*. *P*(*x*))
RIexE [| *RI s*; *RI* (*EX x*. *P*(*x*)) |] ==> *RI P*(*s*::'*a*)
RIequI [| *RI s*; *RI t* |] ==> (*RI* (*s*=(*t*::*real*)))
RIequE1 (*RI* (*s*=(*t*::*real*))) ==> *RI s*
RIequE2 (*RI* (*s*=(*t*::*real*))) ==> *RI t*
RIaddI [| *RI s*; *RI t* |] ==> (*RI* (*s*+*t*))
RIaddE1 (*RI* (*s*+*t*)) ==> *RI s*
RIaddE2 (*RI* (*s*+*t*)) ==> *RI t*
RInegI (*RI s*) ==> *RI* (−*s*)
RInegE (*RI* (−*s*)) ==> *RI s*
RImulI [| *RI s*; *RI t* |] ==> (*RI* (*s*∗*t*))
RImulE1 (*RI* (*s*∗*t*)) ==> *RI s*
RImulE2 (*RI* (*s*∗*t*)) ==> *RI t*
RIdivI [| *RI s*; *RI t* |] ==> (*RI* (*s*/*t*))
RIdivE1 (*RI* (*s*/*t*)) ==> *RI s*
RIdivE2 (*RI* (*s*/*t*)) ==> *RI t*
RIlessI [| *RI s*; *RI t* |] ==> (*RI* (*s*<*t*))
RIlessE1 (*RI* (*s*<*t*)) ==> *RI s*
RIlessE2 (*RI* (*s*<*t*)) ==> *RI t*
RIleI [| *RI s*; *RI t* |] ==> (*RI* (*s*<=*t*))
RIleE1 (*RI* (*s*<=*t*)) ==> *RI s*
RIleE2 (*RI* (*s*<=*t*)) ==> *RI t*

CFtrue *CF True*

```
  CFfalse    CF False
  CFconjI    [| CF P; CF Q |] ==> CF (P&Q)
  CFconjE1   (CF P&Q) ==> CF P
  CFconjE2   (CF P&Q) ==> CF Q
  CFdisjI    [| CF P; CF Q |] ==> CF (P|Q)
  CFdisjE1   (CF P|Q) ==> CF P
  CFdisjE2   (CF P|Q) ==> CF Q
  CFimpI     [| CF P; CF Q |] ==> CF (P-->Q)
  CFimpE1    (CF P-->Q) ==> CF P
  CFimpE2    (CF P-->Q) ==> CF Q
  CFallI     CF P(s) ==> CF (ALL x::'a. P(x))
  CFallE     CF (ALL x. P(x)) ==> CF P(s::'a)
  CFexI      CF P(s) ==> CF (EX x::'a. P(x))
  CFexE      CF (EX x. P(x)) ==> CF P(s::'a)
  CFequ      CF (s=(t::real))
  CFless     CF (s<t)
  CFle       CF (s<=t)

end
```

B.2 LSILHOL

```
LSILHOL = LFOLHOL +

global

consts
  chop          :: [bool, bool] => bool   (infixr ^ 38)

  len           :: real

  conv          :: bool => bool
  fwd           :: bool
  bwd           :: bool

syntax (xsymbols)
  op ^          :: [bool, bool] => bool   (infixr ⌢ 38)

local

defs
```

```
  conv-def     conv(P) == (EX x. (len=x) & (len=0 & (len=x) ^P) ^True)
  bwd-def      bwd == len <= 0
  fwd-def      fwd == 0 <= len

rules
  chopI        [| <i,k>:P; <k,j>:Q |] ==> (<i,j>:P^(Q::bool))
  chopE        [| <i,j>:P^(Q::bool); !!k. [| <i,k>:P; <k,j>:Q |] ==> <l,m>:R
|]
                 ==> <l,m>:R

  uniq1        [| RI s; <i,k>:len=s; <i,l>:len=s; <l,j>:P |] ==> <k,j>:P
  uniq2        [| RI s; <k,j>:len=s; <l,j>:len=s; <i,l>:P |] ==> <i,k>:P
  zero         <i,i>:len=0
  plusI        [|<i,k>:len=s; <k,j>:len=t; RI s; RI t |] ==> <i,j>:len=s+t
  plusE        [| <i,j>:len=s+t; RI s; RI t;
                 !!k. [| <i,k>:len=s; <k,j>:len=t |]
                 ==> <l,m>:R |] ==> <l,m>:R

  (* Rigidity *)

  RIchopI    [| RI P; RI Q |] ==> RI (P^(Q::bool))
  RIchopE1   (RI (P^(Q::bool))) ==> RI P
  RIchopE2   (RI (P^(Q::bool))) ==> RI Q


setup LCla.setup
setup Lclasetup

end
```

B.3 LSDCHOL

```
LSDCHOL = LSILHOL +

global

types
  s
arities
  s :: type
```

```
consts
  ST          :: s => bool                ((ST -))

  TOP         :: s
  BOT         :: s
  NOT         :: s => s                   (NOT - [40] 40)
  AND         :: [s, s] => s              (infixr 35)
  OR          :: [s, s] => s              (infixr 30)
  IMP         :: [s, s] => s              (infixr 25)
  IFF         :: [s, s] => s              (infixr 25)

  dur         :: s => real
  high        :: s => bool                (('-'))
  |^|         :: [bool, bool] => bool     (infixr 38)

local

defs
  high-def    'S' == dur(S) = len & len ~= 0
  chopsub-def P|^|Q == (P & fwd)^(Q & fwd)

rules
  STTOP   ST TOP == True
  STBOT   ST BOT == False
  STNOT   ST (NOT P) == ~(ST P)
  STAND   ST (P AND Q) == (ST P) & (ST Q)
  STOR    ST (P OR Q) == (ST P) | (ST Q)
  STIMP   ST (P IMP Q) == (ST P) --> (ST Q)
  STIFF   ST (P IFF Q) == (ST P) = (ST Q)

  DA1   <i,j>:dur(BOT) = 0
  DA2   <i,j>:dur(TOP) = len
  DA3a  <i,j>:len <= 0 ==> <i,j>:dur(S) <= 0
  DA3b  <i,j>:0 <= len ==> <i,j>:0 <= dur(S)
  DA4   <i,j>:dur(S1) + dur(S2) = dur(S1 OR S2) + dur(S1 AND S2)
  DA5   [| <i,k>:dur(S)=s; <k,j>:dur(S)=t; RI s; RI t |] ==> <i,j>:dur(S)=s+t
  DA6   ST (S1 IFF S2) ==> <i,j>:dur(S1) = dur(S2)

  IRr   [| <i,j>:len=0 --> P; <i,j>:P|^|'S' --> P; <i,j>:P|^|'NOT S'
--> P |]
        ==> <i,j>:P
  IRl   [| <i,j>:len=0 --> P; <i,j>:'S'|^|P --> P; <i,j>:'NOT S'|^|P
--> P |]
```

$==> <i,j>:P$

end

B.4 ILDCHOL

$ILDCHOL = LSDCHOL +$

global

consts

ssub	:: *bool* => *bool*	(<*S*>- [*50*] *50*)
asub	:: *bool* => *bool*	([*S*]- [*50*] *50*)
spref	:: *bool* => *bool*	(<*P*>- [*50*] *50*)
apref	:: *bool* => *bool*	([*P*]- [*50*] *50*)

local

defs

ssub-def	$<S>P == True \mid \hat{} \mid P \mid \hat{} \mid True$
asub-def	$[S]P == \tilde{}(<S>(\tilde{}P))$
spref-def	$<P>P == P \mid \hat{} \mid True$
apref-def	$[P]P == \tilde{}(<P>(\tilde{}P))$

end

B.5 NLDCHOL

$NLDCHOL = ILDCHOL +$

global

consts

srn	:: *bool* => *bool*	(<*R*>- [*50*] *50*)
arn	:: *bool* => *bool*	([*R*]- [*50*] *50*)
sln	:: *bool* => *bool*	(<*L*>- [*50*] *50*)
aln	:: *bool* => *bool*	([*L*]- [*50*] *50*)
srnrev	:: *bool* => *bool*	(<*R*−>- [*50*] *50*)

```
  arnrev    :: bool => bool    ([R-]- [50] 50)
  slnrev    :: bool => bool    (<L->- [50] 50)
  alnrev    :: bool => bool    ([L-]- [50] 50)
  srnconv   :: bool => bool    (<Rc>- [50] 50)
  slnconv   :: bool => bool    (<Lc>- [50] 50)

  sfwd      :: bool => bool    (<A>- [50] 50)
  afwd      :: bool => bool    ([A]- [50] 50)


defs
  srn-def       <R>P == True^(len=0 & (P^bwd))
  sln-def       <L>P == (len=0 & (bwd^P))^True
  srnrev-def    <R->P == <L>P
  slnrev-def    <L->P == <R>P
  arn-def       [R]P == ~(<R>(~P))
  aln-def       [L]P == ~(<L>(~P))
  arnrev-def    [R-]P == ~([R](~P))
  alnrev-def    [L-]P == ~([L](~P))
  srnconv-def   <Rc>P == <R-><R>P
  slnconv-def   <Lc>P == <L-><L>P
  sfwd-def      <A>P == True^(P & fwd)^True
  afwd-def      [A]P == ~(<A>(~P))

end
```

Appendix C

Case Study: IPSEC - OAKLEY

[NWGH98] The OAKLEY key determination protocol is the key exchange mechanism used with IP-SEC. It is specifically intended to be used with the Internet Security Association and Key Management Protocol (ISAKMP) over UDP. ISAKMP provides a framework for Internet key management and protocol support (including format) for security attribute negotiation.

The use of the protocol is to establish a shared key with an assigned identifier and associated authenticated identities to be used between two parties. The key such established is associated with appropriate algorithms for authentication and privacy and one-way message digest operations such as hashing that are established as part of the negotiation.

OAKLEY is versatile and offers a number of features that are grouped into three main components:

- Cookie exchange with the option of being stateless, delaying the initialization of a state until the initiators address is verified, in order to provide maximal resistance to simple DoS attacks.
- An optional Diffie-Hellman half-key exchange essential for the use of perfect forwarding secrecy.
- Authentication with a number of possible options:
  - Privacy for identities.
  - Privacy for identities with perfect forwarding secrecy.
  - Non-reputability.

The resulting protocol is pretty much a derivative of the Station-to-Station protocol treated by Meadows [Mea99] but it differs in a number of ways:

- A weak address authentication mechanism (cookies) is introduced in order to counter DoS.

- The protocol allows two parties to select mutually agreeable support mechanisms.
- Authentication is performed by validation of the bindings of exponentials to identities.
- Authentication may be performed before expensive calculations of exponentials are carried out.
- The key derivation depends both on the Diffie-Hellman mechanism and the specific authentication mechanism used.
- The protocol determines how group representations and operations are selected.

The anti-clogging (DoS resistance) tokens (Cookies) initially exchanged are used both for anti-clogging and key naming. For each session the two newly exchanged and fresh cookies become the key-identifier such that KEYID $\equiv$ COOKIE-I $\mid$ COOKIE-R.

Of course the versatility of OAKLEY allows it to be used in a number of ways depending on the individual needs of the clients. An environment that is less demanding in terms of security allows for very aggressive exchanges where the entire protocol finishes in only three messages. In other environments more conservative approaches are necessary in order to provide maximal security. As DoS is a central topic for us, we are going to study a conservative exchange that takes measures against this type of attack.

C.1 A Conservative Exchange

The conservative protocol version exhibited here is minimally aggressive. The protocol progresses through a weak stateless address verification stage to prevent DoS, a Diffie-Hellman half-key exchange to establish perfect forwarding secrecy, and then goes on to perform authentication using public-key encryption.

In order to create a sufficiently detailed model of the protocol we start out from a standard Alice and Bob style specification. Each symbol used in the schema is explained in figure C.1.

$$
OAKS = \left\{
\begin{array}{lll}
I \rightarrow R: & 0,\ 0,\ \text{KX} & (1)\\
R \rightarrow I: & 0,\ C_R,\ \text{KX} & (2)\\
I \rightarrow R: & C_I,\ C_R,\ \text{KX},\ G,\ g^{x_I},\ Opt & (3)\\
R \rightarrow I: & C_R,\ C_I,\ \text{KX},\ G,\ g^{x_R},\ Sel & (4)\\
I \rightarrow R: & C_I,\ C_R,\ \text{KX},\ G,\ g^{x_I},\ \text{IDP}, & \\
 & ID_I,\ ID_R,\ E\{N_I\}_{K_R} & (5)\\
R \rightarrow I: & C_R,\ C_I,\ \text{KX},\ G,\ 0,\ 0,\ \text{IDP}, & \\
 & E\{N_R,\ N_I\}_{K_I},\ ID_R,\ ID_I, & \\
 & prf(K_{IR},\ ID_R \mid ID_I \mid G \mid g^{x_R} \mid g^{x_I} \mid Sel) & (6)\\
I \rightarrow R: & C_I,\ C_R,\ \text{KX},\ G,\ 0,\ 0,\ \text{IDP}, & \\
 & E\{N_I,\ N_R\}_{K_R},\ ID_I,\ ID_R, & \\
 & prf(K_{IR},\ ID_I \mid ID_R \mid G \mid g^{x_I} \mid g^{x_R} \mid Sel) & (7)
\end{array}
\right.
$$

As mentioned the protocol is executed in three quite distinct communication stages followed by a key construction stage. The different stage will be discussed individually and detailed in the following.

C.1.1 Exchange Description

Address Verification Stage

The first two messages are being exchanged only serve to satisfy the anti-clogging scheme. By sending a message containing only the conversation type flag KX, the initiator notifies the responder that a key exchange session is desired.

The responder returns nothing but a cookie C_R and a conversation type flag. Optimally this cookie is created using a one-way function that depends on a periodically changed secret value, the local and remote IP address,

Symbol	Description
$\mid$	Concatenation operator
I	Initiator
R	Responder
K	Cryptographic or signing key
K_I, K_R	Public keys of initiator/responder
K_{IR}	Shared key for hashing ($\equiv\ prf(0,\ N_I \mid N_R)$)
K_{pfs}	Key for perfect forwarding secrecy ($\equiv\ prf(0, g^{x_I x_R})$)
ID_{K_s}	Identity of final shared keying material ($\equiv\ C_I \mid C_R$)
K_s	Final shared keying material ($\equiv\ prf(N_I \mid N_R,\ g^{x_I x_R} \mid C_I \mid C_R)$))
KX	OAKLEY key exchange msg type
NG	OAKLEY new group msg type
C_I, C_R	64bit pseudo-random number (cookie)
ID_K	Session key identifier ($\equiv\ C_I \mid C_R$)
IDP	Material after encryption boundary is encrypted
NIDP	Material after encryption boundary is not encrypted
G	32bit value identifying group and relevant parameters
g^x	Encoding of group element g by raising to x'th power
Opt	Offered options for encryption/hashing/authentication
Sel	Selected options for encryption/hashing/authentication
N_I, N_R	Nonces
ID_I, ID_R	Identities to be used in authentication
$E(x)_K$	Encryption of message x with key K
$S(x)_K$	Signing of message x with key K
$prf(a,\ b)$	Application of pseudo-random function a to b
$prf(0,\ b)$	One-way function applied to b

Figure C.1: Explanation of symbols used in OAKLEY scheme

and the local and remote UDP port. This allows the responder to validate a returned cookie without having to store it locally. Thus the address

verification may be completed without establishing an associated state.

Diffie-Hellman State Establishment Stage

In case the address of the initiator is genuine the Diffie-Hellman half-key exchange and corresponding feature negotiation may be initiated. The initiator also creates a cookie C_I and establishes a state uniquely identified by the compound value $C_I \mid C_R$. A Diffie-Hellman group G and a pseudo-randomly selected exponent x_I are selected and associated with the state. An encoded group member g^{x_I} is calculated and stored and a set of encryption, hashing, and authentication options *Opt* are selected and stored. The key is marked as unauthenticated and a time-out limit is stored. Finally a message containing cookies, conversation type identifier, group identifier, encoded group member, and selected encryption/hashing/authentication options is sent to the responder.

Upon receipt the responder verifies that two appropriate cookies are included. If so, a persistent state similar to that of the initiator is created and associated with the conversation. If any of the options proposed by the initiator are appropriate a selection is made. This selection is then communicated to the initiator in a similarly structured message where the options are replaced by an actual selection.

Authenticated Key Construction Stage

Once an acceptable set of features are established the authentication phase may be initiated. The initiator associates with his state a flag IDP signifying that perfect forwarding secrecy (*pfs*) is used. This means that the authentication payload is encrypted with a shared key constructed by the application of a one-way function to the two Diffie-Hellman half-keys, such that i.e. the line

$$\begin{aligned} I \rightarrow R: \quad & C_I,\ C_R,\ \text{KX},\ G,\ g^{x_I},\ \text{IDP}, \\ & ID_I,\ ID_R,\ E\{N_I\}_{K_R} \end{aligned} \qquad (5)$$

implicitly means

$$\begin{aligned} I \rightarrow R: \quad & C_I,\ C_R,\ \text{KX},\ G,\ g^{x_I},\ \text{IDP}, \\ & E\{ID_I,\ ID_R,\ E\{N_I\}_{K_R}\}_{K_{pfs}} \end{aligned} \qquad (5)$$

Furthermore a cryptographic nonce N_I is created and associated with the state. Finally a message is constructed and sent to the responder. This

message contains cookies, conversation type, group, half-key, *pfs*-flag, and the authentication payload encrypted with K_{pfs}. The authentication payload includes the identities of the two correspondents ID_I and ID_R as well as the nonce of the initiator encrypted with the public key of the responder K_R.

Following the receipt of this message the responder updates the associated state. The *pfs*-flag is set, the nonce created by the initiator is stored, and a new nonce N_R is created and stored as well. A new message consisting of cookies, conversation type, group, *pfs*-flag, and encrypted authentication payload is then constructed and send. The authentication payload of this message contains the identities of both parties, both of the generated nonces encrypted with the public key of the initiator K_I, and a dependent hash of the identity-, group-, half-key-, and selected option-parts of the current state, where the dependency is on the generated nonces.

Finally, the responder acknowledges the authentication of the exchanged keying material by returning a similar authentication message. The only difference between this and the last message is that all items associated with identities are swapped in order to produce a different plain-text.

Key Construction

At this stage the two parties have exchanged the information necessary in order to build the shared key. The shared key material named $ID_{K_s} \equiv C_I \mid C_R$ is given by $K_s \equiv prf(N_I \mid N_R,\ g^{x_I x_R} \mid C_I \mid C_R)$ and such depends on all of the employed security mechanisms.

The use of the Diffie-Hellman half-key encryption guarantees that the identities of the participants are hidden throughout the exchange. Because of the dependent hashes of the state values exchanged the participants obtain mutual assurances that their keys are derived from the same information. And, finally, the use of public key encryption allows the participants to trust that the identities associated with the derived keys are genuine.

C.1.2 Simplifying the Exchange

The conservative exchange, as presented above, contains information, which is only necessary in order to distinguish the message types in a real world

implementation where the protocol serves a variety of purposes. The message type flag, KX, included in every single message in order to determine its purpose is strictly not necessary in the isolated context in which we are going to study the exchange.

The use of the authentication type flag, IDP, to distinguish messages using a particular forwarding option is not necessary either, as the present context does not allow for other options. In fact, the use of the flag obscures the clarity of what actually happens with regards to encryption in the authentication payload.

In order to make the protocol description as explicit and succinct as possible, before I begin the formal modeling, I alter the above description to exclude the two superfluous flags and explicitly describe the encryption of the authentication header.

$$
OAKS = \begin{cases}
I \rightarrow R: & 0,\ 0 & (1)\\
R \rightarrow I: & 0,\ C_R & (2)\\
I \rightarrow R: & C_I,\ C_R,\ G,\ g^{x_I},\ Opt & (3)\\
R \rightarrow I: & C_R,\ C_I,\ G,\ g^{x_R},\ Sel & (4)\\
I \rightarrow R: & C_I,\ C_R,\ G,\ g^{x_I},\ E\{ID_I,\ ID_R,\ E\{N_I\}_{K_R}\}_{K_{pfs}} & (5)\\
R \rightarrow I: & C_R,\ C_I,\ G,\ 0,\ 0,\ E\{E\{N_R,\ N_I\}_{K_I},\ ID_R,\ ID_I,\ prf(K_{IR},\ ID_R \mid ID_I \mid G \mid g^{x_R} \mid g^{x_I} \mid Sel)\}_{K_{pfs}} & (6)\\
I \rightarrow R: & C_I,\ C_R,\ G,\ 0,\ 0,\ E\{E\{N_I,\ N_R\}_{K_R},\ ID_I,\ ID_R,\ prf(K_{IR},\ ID_I \mid ID_R \mid G \mid g^{x_I} \mid g^{x_R} \mid Sel)\}_{K_{pfs}} & (7)
\end{cases}
$$

The resulting Alice and Bob specifiaction is the basis of the formal modelling I will carry out.

Man kan diskutere IDP, idet det jo her er initiatoren, som bestemmer

C.2 An Annotated Conservative Exchange

The next logical step toward a formal model of the protocol, that fits into our proposed framework, is to add detail to the given Alice and Bob specification. The extra detail is added as annotations, in the style of Meadows

[Mea99], that describe the internal actions that logically precede or follow the sending and receiving of messages respectively. The actions that precede sending are carried out by the sender, and the actions that follow reception are carried out by the receiver.

Symbol	Description	Cost
cr-cky	Create cookie	1
cr-sta	Create state	1
st-cky	Store cookie	1
accept	Accept recieved message	0
se-grp	Select Diffie-Hellman group	1
rse-exp	Randomly select exponent in group	1
se-opt	State options for EHA	1
cr-hkey	Create Diffie-Hellman half-key	4
ch-cky	Check cookie	1
st-cky	Store cookie in state	1
st-hkey	Store half-key in state	1
cr-keypfs	Create key for perfect forwarding secrecy	4
se-sel	select algorithmic options for EHA	1
st-sel	Store selected options for EHA	1
cr-nce	Create nonce	4
re-pkey	Retrieve public key of participant	3
en-pkey	Encrypt using public key	4
en-keypfs	Encrypt using Diffie-Hellman key	3
de-keypfs	Decrypt using Diffie-Hellman key	3
ch-ID	Check ID of participant	1
st-ID	Store ID of participant	1
de-pkey	Decrypt using private key	3
st-nce	Store nonce	1
cr-keyir	Create shared signing key from nonces	4
cr-sgn	Create signature over items	3
ch-sgn	Check signature over items	3

Figure C.2: Internal actions and their annotations. EHA abbreviates encryption/hashing/authentication.

The main purpose of the annotation is to associate a cost in terms of processing time with the execution of the individual protocol steps. Figure C.2 lists and explains the different actions. For now the costs are guessed.

$$
OAKS = \left\{
\begin{array}{lll}
I \rightarrow R: & \| & \\
 & 0,\ 0 & (1) \\
 & \| \ accept_1 & \\
R \rightarrow I: & cr\text{-}cky \ \| & \\
 & 0,\ C_R & (2) \\
 & \| \ cr\text{-}state,\ st\text{-}cky,\ accept & \\
I \rightarrow R: & cr\text{-}cky,\ se\text{-}grp,\ rse\text{-}exp,\ se\text{-}opt,\ cr\text{-}hkey \ \| & \\
 & C_I,\ C_R,\ G,\ g^{x_I},\ Opt & (3) \\
 & \| \ ch\text{-}cky,\ cr\text{-}state,\ st\text{-}cky,\ st\text{-}grp,\ st\text{-}hkey,\ accept & \\
R \rightarrow I: & rse\text{-}exp,\ cr\text{-}hkey,\ cr\text{-}keypfs\ se\text{-}sel \ \| & \\
 & C_R,\ C_I,\ G,\ g^{x_R},\ Sel & (4) \\
 & \| \ ch\text{-}cky,\ st\text{-}hkey,\ st\text{-}sel,\ accept & \\
I \rightarrow R: & cr\text{-}keypfs,\ cr\text{-}nce,\ re\text{-}pkey,\ en\text{-}pkey,\ en\text{-}keypfs \ \| & \\
 & C_I,\ C_R,\ G,\ g^{x_I}, & \\
 & E\{ID_I,\ ID_R,\ E\{N_I\}_{K_R}\}_{K_{pfs}} & (5) \\
 & \| \ ch\text{-}cky,\ de\text{-}keypfs,\ de\text{-}pkey,\ st\text{-}nce,\ accept & \\
R \rightarrow I: & cr\text{-}nce,\ re\text{-}pkey,\ en\text{-}pkey,\ cr\text{-}keyir,\ cr\text{-}sgn,\ en\text{-}keypfs \ \| & \\
 & C_R,\ C_I,\ G,\ 0,\ 0, & \\
 & E\{E\{N_R,\ N_I\}_{K_I},\ ID_R,\ ID_I, & \\
 & prf(K_{IR},\ ID_R \mid ID_I \mid G \mid g^{x_R} \mid g^{x_I} \mid Sel)\}_{K_{pfs}} & (6) \\
 & \| \ ch\text{-}cky,\ de\text{-}keypfs,\ de\text{-}pkey,\ cr\text{-}keyir,\ ch\text{-}sgn,\ accept & \\
I \rightarrow R: & en\text{-}pkey,\ cr\text{-}sgn,\ en\text{-}keypfs \ \| & \\
 & C_I,\ C_R,\ G,\ 0,\ 0, & \\
 & E\{E\{N_I,\ N_R\}_{K_R},\ ID_I,\ ID_R, & \\
 & prf(K_{IR},\ ID_I \mid ID_R \mid G \mid g^{x_I} \mid g^{x_R} \mid Sel)\}_{K_{pfs}} & (7) \\
 & \| \ ch\text{-}cky,\ de\text{-}keypfs,\ de\text{-}pkey,\ cr\text{-}keyir,\ ch\text{-}sgn,\ accept &
\end{array}
\right.
$$

This annotated protocol is derived directly from the description of the conservative exchange in section C.1.1. This description is, therefore, descriptive of this annotated version as well.

C.2.1 Simplifying the Annotations

As previously mentioned, we really want to keep the detail level to the minimum required to achieve an adequate model. There seem to be little or no reason in modelling the sender actions carried out in order to prepare messages as the outcome of these actions do not influence the execution of the protocol. Det er muligt at aktioner som re-pkey alligevel kan influere.

Also, it seems natural that most of those reception actions that are strictly administrative and not employed in the actual validation of the incomming messages, may be postponed and considered message preparation steps instead. In the following we employ the abbreviations shown in figure C.3.

Symb	Definition	Cost
T_1		0
T_2	*cr-cky*	1
T_3	*cr-sta, st-cky, cr-cky, se-grp, rse-exp, se-opt, cr-hkey*	10
T_4	*cr-sta, st-cky, st-grp, st-hkey, rse-exp, cr-hkey, cr-keypfs, se-sel*	14
T_5	*st-hkey, st-sel, cr-keypfs, cr-nce, re-pkey, en-pkey, en-keypfs*	20
T_6	*st-nce, cr-nce, re-pkey, en-pkey, cr-keyir, cr-sgn, en-keypfs*	22
T_7	*en-pkey, cr-sgn, en-keypfs*	10

Figure C.3: Abreviations of internal actions

We employ these abbreviations to obtain an annotated protocol schema, which is by far simpler than the above. We also leave the accept actions out of the annotations and consider them an implicit part of the T-action.

$$
OAKS = \left\{
\begin{array}{lll}
I \to R: & T_1 \| & \\
 & 0,\ 0 & (1)\\
 & \| & \\
R \to I: & T_2 \| & \\
 & 0,\ C_R & (2)\\
 & \| & \\
I \to R: & T_3 \| & \\
 & C_I,\ C_R,\ G,\ g^{x_I},\ Opt & (3)\\
 & \|\ \textit{ch-cky} & \\
R \to I: & T_4 \| & \\
 & C_R,\ C_I,\ G,\ g^{x_R},\ Sel & (4)\\
 & \|\ \textit{ch-cky} & \\
I \to R: & T_5 \| & \\
 & C_I,\ C_R,\ G,\ g^{x_I}, & \\
 & E\{ID_I,\ ID_R,\ E\{N_I\}_{K_R}\}_{K_{pfs}} & (5)\\
 & \|\ \textit{ch-cky},\ \textit{de-keypfs},\ \textit{de-pkey} & \\
R \to I: & T_6 \| & \\
 & C_R,\ C_I,\ G,\ 0,\ 0, & \\
 & E\{E\{N_R,\ N_I\}_{K_I},\ ID_R,\ ID_I, & \\
 & prf(K_{IR},\ ID_R \,|\, ID_I \,|\, G \,|\, g^{x_R} \,|\, g^{x_I} \,|\, Sel)\}_{K_{pfs}} & (6)\\
 & \|\ \textit{ch-cky},\ \textit{de-keypfs},\ \textit{de-pkey},\ \textit{cr-keyir},\ \textit{ch-sgn} & \\
I \to R: & T_7 \| & \\
 & C_I,\ C_R,\ G,\ 0,\ 0, & \\
 & E\{E\{N_I,\ N_R\}_{K_R},\ ID_I,\ ID_R, & \\
 & prf(K_{IR},\ ID_I \,|\, ID_R \,|\, G \,|\, g^{x_I} \,|\, g^{x_R} \,|\, Sel)\}_{K_{pfs}} & (7)\\
 & \|\ \textit{ch-cky},\ \textit{de-keypfs},\ \textit{de-pkey},\ \textit{cr-keyir},\ \textit{ch-sgn} & \\
\end{array}
\right.
$$

C.3 Un-timed Inductive Model

In the following an un-timed inductive model of the OAKLEY protocol is constructed. This model is constructed according to Paulson's inductive approach to formal verification of security protocols.

Empty Trace

$$\frac{}{\epsilon \in oakley}$$

Reception

$$\frac{h \in oakley \quad S_n((A, l(A)) \rightarrow (B, l(B)) : X) \in visible(h)}{h \cdot R_B((A, l(A)) \rightarrow (B, l(B)) : X) \in oakley}$$

$$\frac{h \in oakley \quad S_n((A, l(A)) \rightarrow (B, l(B)) : X) \in visible(h)}{h \cdot B_B((A, l(A)) \rightarrow (B, l(B)) : X) \in oakley}$$

$$\frac{h \in oakley \quad S_n((A, l(A)) \rightarrow (B, l(B)) : X) \in visible(h)}{h \cdot R_B((A, l(A)) \rightarrow (B, l(B)) : X) \cdot B_B((A, l(A)) \rightarrow (B, l(B)) : X) \in oakley}$$

Message 1

$$\frac{h \in oakley \quad A \neq B}{h \cdot S_A((A, l(A)) \rightarrow (B, l(B)) : 0, 0) \in oakley}$$

Message 2

$$\frac{h \in oakley \quad R_R((n, l) \rightarrow (R, l(R)) : 0, 0) = last((h \downarrow n) \downarrow R) \quad C_R \notin used(h)}{h \cdot S_R((B, l(B)) \rightarrow (n, l) : 0, C_R) \in oakley}$$

Message 3

$$\frac{h \in oakley \quad R_I((l, n) \rightarrow (I, l(I)) : 0, C_R) = last((h \downarrow n) \downarrow R) \quad C_I \notin used(h)}{h \cdot S_I((I, l(I)) \rightarrow (n, l) : C_I, C_R, G, g^{x_I}, Opt) \in oakley}$$

Message 4

$$\frac{h \in oakley \quad R_R((l, n) \rightarrow (R, l(R)) : C_I, C_R, G, g^{x_I}, Opt) = last((h \downarrow n) \downarrow R)}{h \cdot S_R((R, l(R)) \rightarrow (n, l) : C_R, C_I, G, g^{x_R}, Sel) \in oakley}$$

Message 5

$$\frac{h \in oakley \quad R_I((l,n) \rightarrow (I,l(I)) : C_R, C_I, G, g^{x_R}, Sel) = last((h \downarrow n) \downarrow R) \quad N_I \notin used(h)}{h \cdot S_I((I,l(I)) \rightarrow (n,l) : C_I, C_R, G, g^{x_I}, E\{ID_I, ID_R, E\{N_I\}_{K_R}\}_{K_{pfs}}) \in oakley}$$

Message 6

$$\frac{h \in oakley \quad \begin{array}{c} R_R((l,n) \rightarrow (R,l(R)) : C_I, C_R, G, g^{x_I}, \\ E\{ID_I, ID_R, E\{N_I\}_{K_R}\}_{K_{pfs}}) = last((h \downarrow n) \downarrow R) \end{array} \quad N_R \notin used(h)}{\begin{array}{c} h \cdot S_R((R,l(R)) \rightarrow (n,l) : C_R, C_I, G, 0, 0, E\{E\{N_R, N_I\}_{K_I}, ID_R, ID_I, \\ prf(K_{IR}, ID_R|ID_I|G|g^{x_R}|g^{x_I}|Sel\}_{K_{pfs}}) \in oakley \end{array}}$$

Message 7

$$\frac{h \in oakley \quad \begin{array}{c} R_I((n,l) \rightarrow (I,l(I)) : C_R, C_I, G, 0, 0, E\{E\{N_R, N_I\}_{K_I}, ID_R, ID_I, \\ prf(K_{IR}, ID_R|ID_I|G|g^{x_R}|g^{x_I}|Sel\}_{K_{pfs}}) = last((h \downarrow n) \downarrow R) \end{array}}{\begin{array}{c} h \cdot S_I((I,l(I)) \rightarrow (n,l) : C_I, C_R, G, 0, 0, E\{E\{N_I, N_R\}_{K_R}, ID_I, ID_R, \\ prf(K_{IR}, ID_I|ID_R|G|g^{x_I}|g^{x_R}|Sel\}_{K_{pfs}}) \in oakley \end{array}}$$

C.4 Timed properties of OAKLEY

Appendix D

Mutually Inductive Definition Experiments

D.1 A Simple Experiment

```
tmp = Real+

datatype num = Nul | One | Two
datatype alp = A | B

consts
  numseq :: "num list set"
  alpseq :: "nat ⇒ alp list set"

inductive "numseq" "alpseq Num"
  intrs
  Niln  "[] ∈ numseq"
  Nila  "[] ∈ alpseq Num"
  Nul   "[|n : numseq; hd n = hd []|] ⇒ Nul#n : numseq"
  One   "[|n : numseq; a : alpseq Num; hd a = A|] ⇒ One#n : numseq"
  A     "[|n : numseq; a : alpseq Num; hd n = Nul|] ⇒ A#a : (alpseq Num)"


rules
  headn  "(x::num) = hd [] ≡ False"
  heada  "(x::alp) = hd [] ≡ False"
  headn1 "hd [] = (x::num) ≡ False"
  heada1 "hd [] = (x::alp) ≡ False"


end
```

```
val One = numseq_alpseq.One;
val Nul = numseq_alpseq.Nul;
val Niln = numseq_alpseq.Niln;
val Nila = numseq_alpseq.Nila;
val A = numseq_alpseq.A;

Goal "∃seq ∈ numseq. hd seq = One";
  br bexI 1;
  by (rtac One 2);
  by (rtac Nul 2);
  by (rtac A 4);
  by (rtac Nul 4);
  by (rtac Niln 4);
  by (rtac Nila 5);
  by (rtac Niln 2);
  by (Auto_tac);
qed "numseqOne";
```

D.2 Defining Local and Global Traces

```
theory test = Trace + LTrace:

consts
  M :: "nat ⇒ msg"

consts
  "gtrace" :: "event list set"
  "ltrace"  :: "agent ⇒ action list set"
inductive "gtrace" "ltrace A"
intros
  Nilg:  "[] ∈ gtrace"
  Nill:  "[] ∈ ltrace A"

  Ready: "[|acs ∈ ltrace A; hd acs = hd [] ∨ hd acs = Accept ∨ hd acs = Abort|]
 ⇒ Ready # acs ∈ ltrace A"

  Initl: "[|acs ∈ ltrace A; hd acs = Ready|] ⇒ SState (Suc 0) # acs ∈ ltrace A
"
  Respl: "[|acs ∈ ltrace A; hd acs = Ready|] ⇒ RState (Suc 0) # acs ∈ ltrace A
"

  Rcvi:  "[|evs : gtrace; (|(B, B) → (A, A): M i|) visible evs; acs ∈ ltrace A;
 hd acs = RState i|] ⇒
            Blk A (|(B, B) → (A, A): M i|) # Rcv A (|(B, B) → (A, A): M i|) #
evs : gtrace"
  Veri1: "[|h : gtrace; hd h = Blk A (|(B, B) → (A, A): M i|); acs ∈ ltrace A;
hd acs = RState i|] ⇒
            Ver i (Suc 0) (chkAgent B) # acs ∈ ltrace A"
  Veri:  "[|acs ∈ ltrace A; hd acs = Ver i 1 True; i ≤ 2|] ⇒ SState (Suc i) # a
cs ∈ ltrace A"

  Snd1:  "[|acs ∈ ltrace A; hd acs = SState 1; evs ∈ gtrace|] ⇒ Snd A (|(A,A)→
(B,B):M 1|)#evs ∈ gtrace"
  Snd1l: "[|acs ∈ ltrace A; hd acs = SState 1; evs ∈ gtrace; hd evs = Snd A (|(
A,A)→(B,B):M 1|)|] ⇒ RState 2 # acs ∈ ltrace A"

  Snd2:  "[|acs ∈ ltrace A; hd acs = SState 2; evs ∈ gtrace|] ⇒ Snd A (|(A,A)→
(B,B):M 2|)#evs ∈ gtrace"
  Snd2l: "[|acs ∈ ltrace A; hd acs = SState 1; evs ∈ gtrace; hd evs = Snd A (|(
A,A)→(B,B):M 1|)|] ⇒ Accept # acs ∈ ltrace A"

  Accept:"[|acs ∈ ltrace A; hd acs = Ver (Suc (Suc 0)) (Suc 0) True|] ⇒ Accept
# acs ∈ ltrace A"
  Abort: "[|acs ∈ ltrace A; hd acs = Ver i j False|] ⇒ Abort # acs ∈ ltrace A"

lemma "a ∉ bad ⇒ ∃acs ∈ ltrace a. hd acs = Accept"
  apply (rule bexI)
  prefer 2
  apply (rule Accept)
  apply (rule_tac B = Server and i = 2 in Veri1)
  prefer 6
  apply simp
  prefer 5
  apply simp
  apply (rule_tac A = a and B = Server and i = 2 in Rcvi)
  prefer 5
  apply simp
  apply (rule_tac A = Server and B = a in Snd2)
  prefer 4
  apply simp
  prefer 3
  apply (rule_tac A = Server and B = a and i = 1 in Rcvi)
  apply (rule_tac A = a and B = Server in Snd1)
  prefer 3
  apply (rule Nilg)
  prefer 3
  apply simp

  apply (rule Initl)
  apply (rule Ready)
```

```
  apply (rule Nill)
  apply force+
  apply (rule Respl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+
  apply (rule_tac i = 1 in Veri)
  prefer 3
  apply simp
  prefer 3
  apply simp
  apply (rule_tac B = a and i = 1 in Veri1)
  prefer 5
  apply simp
  apply (rule_tac i = 1 and B = a and A = Server in Rcvi)
  prefer 5
  apply simp
  apply (rule_tac A = a and B = Server in Snd1)
  prefer 4
  apply simp
  apply (rule Initl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+
  apply (rule Nilg)
  apply (rule Respl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+
  apply (rule Respl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+

  apply (rule_tac B = Server in Snd1l)
  apply (rule Initl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+
  apply (rule_tac A = a and B = Server in Snd1)
  apply (rule Initl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+
  apply (rule Nilg)
  apply simp+
  apply (rule_tac B = Server in Snd1l)
  apply (rule Initl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+
  apply (rule_tac A = a and B = Server in Snd1)
  apply (rule Initl)
  apply (rule Ready)
  apply (rule Nill)
  apply simp+
  apply (rule Nilg)
  by simp+

end
```

Appendix E

An a2ps Pretty Printer for Theory and ML Files

```
# Style sheet for Isabelle theories and proof scripts
# Copyright (c) 2001 Henrik Pilegaard
# $Id: thyssh1.eps,v 1.2 2002/11/02 20:24:24 hp Exp $
#


style "thy" is

written by "Henrik Pilegaard <henrik.pilegaard@acm.org>"
version is 1.0
requires a2ps version 4.9.7

documentation is
  "This is the style sheet for Isabelle Logic theories"
end documentation

alphabets are
   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_#$%"
case sensitive

keywords in Keyword_strong are
        header,
        section,
        subsection,
        subsubsection,
        bind_thm,
        let,
        local,
        "|",

        thus,
        classes,
        types,
        arities,
        consts,
        syntax,
        translations,
        defs,
        constdefs,
        subsection,
        axioms,
        setup,
        text,
        setup,
        global,
        local,
        defaultsort,
        typedecl,
        inductive,
        datatype,
        primrec,
        "::",
        typedef,
        lemma,
        theorem,
        lemmas,
        done,
        method_setup,
        default,
        rules,
        "end",
        proof,
        next,
        qed,
        use,
        ML,
        theory,
        intrs,
        fun,
        val,
```

```
        Goal,
        Goalw,
        goal,
        qed_spec_mp
        sig,
        br,
        be,
        ba,
        bd,
        struct
end keywords

keywords in Label are
        and,
        "in",
        "by",
        declare,
        apply,
        show,
        from,
        have,
        with,
        shows,
        assumes,
        prefer,
        defer,
        inductive_cases,
        REPEAT,
        ALLGOALS,
        SELECT_GOAL,
        RS,
        MRS,
        EVERY,
        CHANGED,
        FIRSTGOAL,
        DEPTH_SOLVE,
        THEN,
        IF_UNSOLVED,
        /[a-zA-Z0-9_]+tac/
end keywords

keywords in Keyword are
        pop_proof,
        push_proof,
        structure,
        signature,
        intros,
        infixr,
        infixl,
        infix,
        binder,
        files,
        assume,
        OFCLASS,
        PROP,
        unfold,
        bal,
        bs,
        addss,
        delss,
        map,
        impOfSubs,
        auto,
        split,
        rule_format,
        iff,
        intro,
        simp,
        dest,
        elim,
```

```
        blast,
        simp_all,
        clarify,
        add,
        force,
        assumption,
        /[a-z]*rule/,
        /[a-zA-Z0-9]+simps/,
        /[a-zA-Z0-9]+Is/,
        /[a-zA-Z0-9]+Es/,
        /[a-zA-Z0-9]+Ds/'
end keywords

optional operators are
    \<preceq> "prefix of",
    \\<alpha> α,
    \<alpha> α,
    \<Sum> ∑,
    \\<Sum> ∑,
    \\<integral> ∫,
    \<integral> ∫,
    \<epsilon> ε,
    \\<epsilon> ε,
    \<ge> ≥,
    \\<ge> ≥,
    \<delta> δ,
    \\<delta> δ,
    \\<triangle> Δ,
    \<triangle> Δ,
    \\<Delta> Δ,
    \<Delta> Δ,
    \<nabla> ∇,
    \\<frown> ^,
    \\<le> ≤,
    \<le> ≤,
    Un ∪,
    \\<subseteq> ⊆,
    \<subseteq> ⊆,
    <= ≤,
    \\<in> ∈,
    \<in> ∈,
    \\<and> ∧,
    \<and> ∧,
    \<And> Λ,
    \\<or> ∨,
    \<or> ∨,
    \\<lbrace> {|,
    \<lbrace> {|,
    \\<rbrace> |},
    \<rbrace> |},
    \\<times> ×,
    \<times> ×,
    \\<equiv> ≡,
    \<equiv> ≡,
    \\<lambda> λ,
    \<lambda> λ,
    \\<lparr> "(|",
    \<lparr> "(|",
    \\<rparr> "|)",
    \<rparr> "|)",
    \\<notin> ∉,
    \<notin> ∉,
    \\<down> ↓,
    \<down> ↓,
    \\<Longrightarrow> ⇒,
    \<Longrightarrow> ⇒,
    \\<lbrakk> "[|",
    \<lbrakk> "[|",
    \\<rbrakk> "|]",
    \<rbrakk> "|]",
```

```
    \\<Union> ∪,
    \<Union> ∪,
    \\<union> ∪,
    \<union> ∪,
    \\<up> ↑,
    \<up> ↑,
    \\<Up> ⇑,
    \<Up> ⇑,
    \\<triangleleft> ∠,
    \<triangleleft> ∠,
    ==> ⇒,
    \\<Rightarrow> ⇒,
    \<Rightarrow> ⇒,
    => ⇒,
    == ≡,
    --> →,
    <-> ↔,
    \\<forall> ∀,
    \<forall> ∀,
    \\<exists> ∃,
    \<exists> ∃,
    EX ∃,
    !! Λ,
    ALL ∀,
    \\<leftarrow>\\<rightarrow> ↔,
    \<leftarrow>\<rightarrow> ↔,
    \\<rightarrow> →,
    \<rightarrow> →,
    \\<leftarrow> ←,
    \<leftarrow> ←,
    \\<midarrow>\\<rightarrow> →,
    \<midarrow>\<rightarrow> →,
    \\<longleftarrow> ←,
    \<longleftarrow> ←,
    \\<longrightarrow> →,
    \<longrightarrow> →,
    \\<longleftrightarrow> ↔,
    \<longleftrightarrow> ↔,
    \\<not> ¬,
    \<not> ¬,
    ~= ≠,
    \\<noteq> ≠,
    \<noteq> ≠,
    ~ ¬
end operators

sequences are
    "(*" Comment Comment "*)" Comment
end sequences

end style
```

Index