Modelling Properties of Security Protocols

Henrik Pilegaard

LYNGBY 2002
EKSAMENSPROJEKT
NR. 67

IMM
Trykt af IMM, DTU
Preface


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 sandtidsevngskaber 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. \texttt{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. \texttt{isFalse}. In most cases we use the functional arrow \(\rightarrow\) in order to describe the types of functions, but for injective functions we use \(\mapsto\).

We make the one element of singleton sets \texttt{1} anonymous \{\texttt{*}\}.

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 \texttt{True : \{*\} \mapsto bool} but rather \texttt{True : bool}

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

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

Beware that when used in the defined theories \(=\) denotes normal equality.
# Contents

1 Introduction .............................. 1  
   1.1 Thesis Objectives ...................... 2  
   1.2 Theoretical Background ................ 3  
   1.3 Contribution ........................... 4  
   1.3.1 Starting Point ...................... 4  
   1.3.2 Identifying Active Attacks ........... 5  
   1.3.3 Analyzing Active Attacks ............ 6  
   1.4 Thesis Organization .................. 7  
2 Setting the Scene ....................... 9  
   2.1 Security Protocols .................... 10  
   2.2 Peer Networks .......................... 11  
   2.2.1 The Transport Layer ................ 12  
   2.2.2 Peer Processes ...................... 13  
   2.3 Security Goals ....................... 18  
   2.3.1 Goals Unrelated to Time .......... 18  
   2.3.2 Goals Related to Time ............. 20  
   2.4 Important Approaches ................. 23  
   2.4.1 Inductive Analysis ................ 23  
   2.4.2 Cost Based Fail-Stop Analysis of Denial of Service  25
3 Unfimed Theory of Global Traces 29

3.1 Agents, Keys, and Messages 30
  3.1.1 Agents 30
  3.1.2 Cryptographic Keys 31
  3.1.3 Messages 31

3.2 The Network Packets 35
  3.2.1 Components of Packets 36
  3.2.2 Projections of Packets 37
  3.2.3 Judgments on Packets 37

3.3 Events, Knowledge, and Freshness 38
  3.3.1 The Events of the Network 38
  3.3.2 The Knowledge of Agents 39
  3.3.3 Freshness of Message Components 42

3.4 Traces 42
  3.4.1 Visible Packets 42
  3.4.2 Well-Formed Traces 43
  3.4.3 Projection of Traces 43

3.5 Ideal Protocol Definitions 44

3.6 Schematics of Traces and Active Attacks 45
  3.6.1 Schematic Projection of Traces 46
  3.6.2 Consistent Executions 47
  3.6.3 Identifying Active Attacks 47

4 Example: The Station-to-Station Protocol 49

4.1 Modeling StS 50

4.2 Consistency of Repeated Protocol Engagement 52

4.3 Active Attack on StS 53
5 Untimed Theory of Local Traces 57
  5.1 A Suitable Set of Actions .......................... 58
  5.2 The Local Model .................................... 60
    5.2.1 A Simple Example .............................. 61
6 Timed Properties of Security Protocols 65
  6.1 A Suitable Interval Logic ............................ 66
    6.1.1 Neighborhood Logic ............................ 67
    6.1.2 Rigidity and Chop-freeness ..................... 68
    6.1.3 Labelled Neighborhood Logic ................... 69
  6.2 Timed Lists ........................................ 70
    6.2.1 Causality and Prefix Operators ............... 70
    6.2.2 Rigid and Chop-free Terms and Formulas ...... 70
    6.2.3 Timed Operators ................................ 71
  6.3 Timed Global Traces ................................ 72
    6.3.1 The Flexible Constant Tr ...................... 72
    6.3.2 Predicates Concerning Tr ..................... 73
    6.3.3 Lemmas and Theorems Regarding Predicates .... 76
  6.4 Timing Local Traces ................................ 78
    6.4.1 The Flexible Constant ITr .................... 78
    6.4.2 Predicates Concerning ITr ..................... 79
    6.4.3 Lemmas and Theorems Regarding Predicates .... 80
    6.4.4 Concerning Verification Functions ............ 82
  6.5 Modeling Heuristics ................................. 83
    6.5.1 Dependency versus Obligation ................. 84
    6.5.2 Universal versus Specific Properties ......... 85
7 Examples: Simple Protocols 89
   7.1 Network Delays ................................. 90
      7.1.1 Creating a Model ......................... 90
      7.1.2 Proving Properties ....................... 91
   7.2 Rejecting Bogus Messages .................... 92
      7.2.1 Creating a Model ......................... 93
      7.2.2 Proving Properties ....................... 94

8 Conclusion 97
   8.1 Executive Summary ............................. 97
   8.2 Discussion ..................................... 99
      8.2.1 Technical Evaluation ...................... 99
      8.2.2 Conclusion ................................. 101
   8.3 Future Work ................................... 102

Isabelle Theories 107
   A.1 Theory Dependency Graph ....................... 108
   A.3 Untimed Theory of Global Traces ............... 109
      A.3.1 Agents, Keys, and Messages - Theory due to Paulson 110
      A.3.2 The Network Packets ...................... 113
      A.3.3 Events, Knowledge, and Freshness .......... 115
      A.3.4 Traces .................................... 118
      A.3.5 Ideal Protocol Definitions ................ 120
      A.3.6 Schematics of Traces and Active Attacks ... 122
   A.4 Example: The Station-to-Station Protocol .......... 124
   A.5 Untimed Theory of Local Traces ................. 134
      A.5.1 A Suitable Set of Actions ................ 135
      A.5.2 The Local Model ........................... 136
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.6</td>
<td>Timed Theory of Protocols</td>
<td>138</td>
</tr>
<tr>
<td>A.6.1</td>
<td>A Suitable Interval Logic</td>
<td>139</td>
</tr>
<tr>
<td>A.6.2</td>
<td>Timed Properties of Lists</td>
<td>144</td>
</tr>
<tr>
<td>A.6.3</td>
<td>Timed Properties of Global Traces</td>
<td>146</td>
</tr>
<tr>
<td>A.6.4</td>
<td>Timed Properties of Local Traces</td>
<td>151</td>
</tr>
<tr>
<td>A.6.5</td>
<td>Modeling Heuristics</td>
<td>156</td>
</tr>
<tr>
<td>A.7</td>
<td>Example: Simple Protocols</td>
<td>158</td>
</tr>
<tr>
<td>A.7.1</td>
<td>Network Delays</td>
<td>159</td>
</tr>
<tr>
<td>A.7.2</td>
<td>Rejecting Bogus Messages</td>
<td>166</td>
</tr>
<tr>
<td>NLDCHOL</td>
<td></td>
<td>169</td>
</tr>
<tr>
<td>B.1</td>
<td>LFOLHOL</td>
<td>169</td>
</tr>
<tr>
<td>B.2</td>
<td>LSILHOL</td>
<td>172</td>
</tr>
<tr>
<td>B.3</td>
<td>LSDCHOL</td>
<td>173</td>
</tr>
<tr>
<td>B.4</td>
<td>ILDCHOL</td>
<td>175</td>
</tr>
<tr>
<td>B.5</td>
<td>NLDCHOL</td>
<td>175</td>
</tr>
<tr>
<td>Case Study: IPSEC - OAKLEY</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>C.1</td>
<td>A Conservative Exchange</td>
<td>179</td>
</tr>
<tr>
<td>C.1.1</td>
<td>Exchange Description</td>
<td>179</td>
</tr>
<tr>
<td>C.1.2</td>
<td>Simplifying the Exchange</td>
<td>182</td>
</tr>
<tr>
<td>C.2</td>
<td>An Annotated Conservative Exchange</td>
<td>183</td>
</tr>
<tr>
<td>C.2.1</td>
<td>Simplifying the Annotations</td>
<td>185</td>
</tr>
<tr>
<td>C.3</td>
<td>Un-timed Inductive Model</td>
<td>187</td>
</tr>
<tr>
<td>C.4</td>
<td>Timed properties of OAKLEY</td>
<td>188</td>
</tr>
<tr>
<td>Mutually Inductive Definition Experiments</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>D.1</td>
<td>A Simple Experiment</td>
<td>190</td>
</tr>
<tr>
<td>D.2</td>
<td>Defining Local and Global Traces</td>
<td>192</td>
</tr>
<tr>
<td>An a2ps Pretty Printer for Theory and ML Files</td>
<td>195</td>
<td></td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Alice and Bob notation for security protocols</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>The Station-to-Station protocol by Diffie, Wiener, and Van Oorschot</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Format of a packet $P$</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Events that a peer $A$ may cause given a packet $P$</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Normal Flow</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>Message Interception</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>Message Interruption</td>
<td>16</td>
</tr>
<tr>
<td>2.8</td>
<td>Masquerades</td>
<td>16</td>
</tr>
<tr>
<td>2.9</td>
<td>Message Theft</td>
<td>16</td>
</tr>
<tr>
<td>2.10</td>
<td>Message Modification</td>
<td>16</td>
</tr>
<tr>
<td>2.11</td>
<td>Message Reuse</td>
<td>17</td>
</tr>
<tr>
<td>6.1</td>
<td>$T_r$ as flexible function of intervals</td>
<td>72</td>
</tr>
<tr>
<td>C.1</td>
<td>Explanation of symbols used in OAKLEY scheme</td>
<td>180</td>
</tr>
<tr>
<td>C.2</td>
<td>Internal actions and their annotations. EHA abbreviates encryption/hashing/authentication</td>
<td>184</td>
</tr>
<tr>
<td>C.3</td>
<td>Abbreviations of internal actions</td>
<td>186</td>
</tr>
</tbody>
</table>
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:

\[ A_1 \rightarrow B_1 : M_1 \]
\[ \vdots \]
\[ A_n \rightarrow B_n : M_n \]

where the \( i \)th 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:

\[ A_1 \rightarrow B_1 : T_1 \| M_1 \| O_1 \]
\[ \vdots \]
\[ A_n \rightarrow B_n : T_n \| M_n \| O_n \]

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 of 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.

1. $A \rightarrow B : M_1$
2. $B \rightarrow S : M_2$
3. $S \rightarrow B : M_3$
4. $B \rightarrow A : M_4$
   
   $\vdots$

$A = \text{Alice}, B = \text{Bob}, S = \text{Server}$

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. $A \rightarrow B : \alpha^x_A$
2. $B \rightarrow A : \alpha^x_B \cdot \{\alpha^x_A, \alpha^x_B\} S_B\}$
3. $A \rightarrow B : \{\alpha^x_A, \alpha^x_B\} S_A\}$

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 $\alpha$ 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, $K = \alpha^x_A x^x_B$, which is subsequently used for authentication.

The first line instructs $A$ to send the integer $\alpha^x_A$ to $B$. The second line then instructs $B$ to reply with the integer $\alpha^x_B$ (to complete the shared key) concatenated with the concatenation of the integers $\alpha^x_B$ and $\alpha^x_A$ 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 sent 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.

\[ [[n_A, l_A] \rightarrow (n_B, l_B) : M] \]

\[ n_A \quad \text{name of } A \]
\[ l_A \quad \text{location of } A \]

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
2.2 Peer Networks

\[
\begin{align*}
\text{Snd}_A P & \quad A \text{ sends } P \\
\text{Rcv}_A P & \quad A \text{ receives } P \\
\text{Blk}_A P & \quad A \text{ blocks } P
\end{align*}
\]

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 [Mead99] 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.

\[
\begin{align*}
\text{Snd}_A &: \{(n_A, t_A) \rightarrow (n_B, t_B) : M\} \\
\text{Rcv}_B &: \{(n_A, t_A) \rightarrow (n_B, t_B) : M\} \\
\text{Blk}_B &: \{(n_A, t_A) \rightarrow (n_B, t_B) : M\}
\end{align*}
\]

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.

$A \xrightarrow{\text{Snd}_A} (n_A, l_A) \rightarrow (n_B, l_B) : M$

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

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

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

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

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

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

\[
\begin{align*}
\text{Snd}_A &\mid ((n_A, l_A) \rightarrow (n_B, l_B) : M) \\
\text{Rcv}_B &\mid ((n_A, l_A) \rightarrow (n_B, l_B) : M) \\
\text{Blk}_B &\mid ((n_A, l_A) \rightarrow (n_B, l_B) : M) \\
&\vdots \\
\text{Snd}_I &\mid ((n_A, l_I) \rightarrow (n_B, l_B) : M)
\end{align*}
\]

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.
2.3 Security Goals

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 [Go99] 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 $H$ 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.
2.3 Security Goals

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-denyal 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. The

---

3In 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 analysed, 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 \( I \) 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/authentication** 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.
2.4 Important Approaches

- **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


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

\[
i \ A \rightarrow B : \ T \parallel M \parallel O
\]

where \( T \in T^\ast \) 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 \( O \in O^\ast \) 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 \( A \), and \( rec_B(M) \), the reception of the message by \( B \).

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 \( send_A \ P \) in our view.

\(^6\)Equivalent to the compound event of \( Rcv_A \ P \) and \( Blk_A \ P \) 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 willing 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 $g \in G$ expressing the intrusive power of the set of actions available to them.

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

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

where the relation desirably-cause-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 appears 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 re-
lected 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 no-
tion 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\footnote{The work on Kerberos [BP98a] is the only known case where this assumption is too weak.} and only one spy while there is an infinite number of friendly agents. Hence, three disjoint sets are given.

\begin{verbatim}
Friend  a set of friendly agents (§2.2.2)  
Server  a singleton set \{x\} (§2.2.2)  
Spy     a singleton set \{x\} (§2.2.2)
\end{verbatim}

\begin{verbatim}
3.1  Agents, Keys, and Messages

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

\[ \text{agent} = \text{Friend} \cup \text{Server} \cup \text{Spy} \]

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

\[ \text{Friend} : \mathbb{N} \leftrightarrow \text{agent} \]
\[ \text{Server} : \text{agent} \]
\[ \text{Spy} : \text{agent} \]

3.1.2  Cryptographic Keys

Paulson assumes an infinite set, \textit{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 \textit{automorphic} function:

\[ \text{invKey} : \text{key} \cong \text{key} \]

which is its own \textit{inverse function} and thus subject to

\[ (\text{invKey} \circ \text{invKey}) K = K \]

For a subset of \textit{symmetric keys} \text{invKey} is \textit{idempotent}:

\[ \text{symKeys} = \{ K : \text{key} | \text{invKey} K = K \} \]

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

\[ \text{key} = \mathbb{N} \]

3.1.3  Messages

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

\begin{align*}
\text{agent} & \quad \text{The described set of agent names} \\
\text{Number} & \quad \text{A set of guessable nonces} \\
\text{Nonce} & \quad \text{A set of unguessable nonces} \\
\text{key} & \quad \text{The described set of cryptographic keys} \\
\text{MPair} & \quad \text{A set of pairs of messages} \\
\text{Hash} & \quad \text{A set of message digests} \\
\text{Crypt} & \quad \text{A set of encrypted messages}
\end{align*}
A message is then either an agent, a number, a nonce, a key, a message pair, a message digest, or an encrypted message.

\[ \text{msg} = \text{agent} \cup \text{Number} \cup \text{Nonce} \cup \text{key} \cup \text{MPair} \cup \text{Hash} \cup \text{Crypt} \]

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 \( \text{agent} \) (§3.1.1).

\[ \text{Agent} : \text{agent} \mapsto \text{msg} \]

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

\[ \text{Number} : \mathbb{N} \mapsto \text{msg} \]

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

\[ \text{Nonce} : \mathbb{N} \mapsto \text{msg} \]

**The cryptographic keys** is a set of messages corresponding to the set \( \text{key} \). They are important for confidentiality.

\[ \text{Key} : \text{key} \mapsto \text{msg} \]

**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.

\[ \text{MPair} : \text{msg} \mapsto (\text{msg} \mapsto \text{msg}) \]

Nested applications of \( \text{MPair} \) are abbreviated as tuples.

\[
\begin{align*}
\{X, Y\} & \equiv \text{MPair } X \ Y \\
\{X, Y, Z\} & \equiv \{X, \{Y, Z\}\}
\end{align*}
\]
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.

\[
\text{Hash} : \text{msg} \mapsto \text{msg}
\]

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

\[
\text{HPair} \ X \ Y \doteq \{ \text{Hash}[:X, Y], Y \} : \text{msg} \mapsto (\text{msg} \mapsto \text{msg})
\]

In order to emphasize dependence on the seed an illustrative abbreviation of \( \text{HPair} \) is defined.

\[
\text{Hash}[X] \ Y \equiv \text{HPair} \ X \ Y
\]

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

\[
\text{Crypt} : \text{key} \mapsto (\text{msg} \mapsto \text{msg})
\]

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

\[
\text{Crypt} \ K \ X = \text{Crypt} \ K' \ X' \Rightarrow K = K' \land X = X'
\]

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.

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

\[
\begin{align*}
X & \in H \quad \text{Crypt } KX & \in \text{parts } H \\
\langle X, Y \rangle & \in \text{parts } H
\end{align*}
\]

Paulson has shown this operator to be monotonous

\[G \subseteq H \implies \text{parts } G \subseteq \text{parts } H\]

and idempotent

\[\text{parts}(\text{parts } H) = \text{parts } H\]

The operator \texttt{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.

\[\text{analz} : \mathcal{P}(\text{msg}) \rightarrow \mathcal{P}(\text{msg})\]

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

\[
\begin{align*}
X & \in H \quad \text{Crypt } KX & \in \text{analz } H \\
\langle X, Y \rangle & \in \text{analz } H
\end{align*}
\]

Paulson has shown this operator to be monotonous

\[G \subseteq H \implies \text{analz } G \subseteq \text{analz } H\]

and idempotent

\[\text{analz}(\text{analz } H) = \text{analz } H\]
The operator \textit{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:

\[
\text{synth}: \mathcal{P}(\text{msg}) \rightarrow \mathcal{P}(\text{msg})
\]

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

\[
\begin{align*}
\text{Agent } A &\in \text{synth} \ H \\
\text{Number } n &\in \text{synth} \ H \\
X &\in H \\
X &\in \text{synth} \ H \\
X &\in \text{synth} \ H \\
\text{Hash } X &\in \text{synth} \ H \\
X &\in \text{synth} \ H \quad Y &\in \text{synth} \ H \\
\{X,Y\} &\in \text{synth} \ H \\
X &\in \text{synth} \ H \quad K &\in H \\
\text{Crypt } KX &\in \text{synth} \ H
\end{align*}
\]

Paulson has shown this operator to be monotonous

\[
G \subseteq H \implies \text{synth} \ G \subseteq \text{synth} \ H
\]

and idempotent

\[
\text{synth} (\text{synth} \ H) = \text{synth} \ H
\]

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 \textit{events} directly on those of agents and messages, we introduce the novel notion of \textit{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 $Nm$ of names is isomorphic to the set of agents. We abuse this and simply define them to be identical.

$$Nm = Agent$$

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

$$Loc = Agent$$

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

$$Adr = Name \times Loc$$

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.

$$fakeAdr \ (n, l) \triangleq n \neq l : Adr \rightarrow bool$$

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

$$goodAdr \ a \triangleq \neg fakeAdr \ a : Adr \rightarrow bool$$
3.2 The Network Packets

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

\[ Pkt = \text{Adr} \times \text{Adr} \times \text{msg} \]

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

\[ \langle s \rightarrow r : m \rangle \equiv \langle s, r, m \rangle \]  \hspace{1cm} (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).

\[
\begin{align*}
\text{sAdrPkt} \; \langle s \rightarrow r : m \rangle & \doteq s : Pkt \rightarrow \text{Adr} \\
\text{rAdrPkt} \; \langle s \rightarrow r : m \rangle & \doteq r : Pkt \rightarrow \text{Adr} \\
\text{MsgPkt} \; \langle s \rightarrow r : m \rangle & \doteq m : Pkt \rightarrow \text{msg} \\
\text{sNmPkt} \; \langle (n, l) \rightarrow r : m \rangle & \doteq n : Pkt \rightarrow \text{Nm} \\
\text{sLocPkt} \; \langle (n, l) \rightarrow r : m \rangle & \doteq l : Pkt \rightarrow \text{Loc} \\
\text{rNmPkt} \; \langle s \rightarrow (n, l) : m \rangle & \doteq n : Pkt \rightarrow \text{Nm} \\
\text{rLocPkt} \; \langle s \rightarrow (n, l) : m \rangle & \doteq l : Pkt \rightarrow \text{Loc}
\end{align*}
\]

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*.

\[
\text{fakePkt} \; \langle s \rightarrow r : m \rangle \doteq \text{fakeAdr} \; s \lor \text{fakeAdr} \; r : Pkt \rightarrow \text{bool}
\]

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

\[
\text{goodPkt} \; p \doteq \neg \text{fakePkt} \; p : Pkt \rightarrow \text{bool}
\]
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:

\[
\begin{align*}
\text{Snd} & \quad \text{a set of sending events} \\
\text{Rcv} & \quad \text{a set of receiving events} \\
\text{Blk} & \quad \text{a set of blocking events}
\end{align*}
\]

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

\[
\text{event} = \text{Snd} \cup \text{Rcv} \cup \text{Blk}
\]

Thus, we have three named injections:

\[
\begin{align*}
\text{Snd} : \text{Nm} & \rightarrow (\text{Pkt} \rightarrow \text{event}) \\
\text{Rcv} : \text{Nm} & \rightarrow (\text{Pkt} \rightarrow \text{event}) \\
\text{Blk} : \text{Nm} & \rightarrow (\text{Pkt} \rightarrow \text{event})
\end{align*}
\]

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

\[
\text{fakeEvent : event \to bool}
\]

which is defined through the following three cases

\[
\begin{align*}
\text{fakeEvent (Snd n } & \{(sn, sl) \to (rn, rl) : m\}) \equiv \\
\text{fakePkt } & \{(sn, sl) \to (rn, rl) : m\} \lor n \neq sn \\
\text{fakeEvent (Rcv n } & \{(sn, sl) \to (rn, rl) : m\}) \equiv \\
\text{fakePkt } & \{(sn, sl) \to (rn, rl) : m\} \lor n \neq rn \\
\text{fakeEvent (Blk n } & \{(sn, sl) \to (rn, rl) : m\}) \equiv \\
\text{fakePkt } & \{(sn, sl) \to (rn, rl) : m\} \lor n \neq rn
\end{align*}
\]

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

\[
\text{goodEvent e } \equiv \neg\text{fakeEvent e}
\]

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:

\[
\text{initState} : \text{agent} \rightarrow \text{msg}
\]

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

\[
\text{bad} : \mathcal{P}(\text{agent})
\]

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:

\[
\text{Spy} \in \text{bad}
\]

\[
\text{Server} \notin \text{bad}
\]

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.

\[
\text{shrK} : \text{agent} \rightarrow \text{key}
\]

with the requirement that for all keys \(K\)

\[
K \in \text{symKeys}
\]

For these systems the initial state is captured by

\[
\begin{align*}
\text{initState Server} & \doteq \{ \text{Key}(\text{shrK } A) \mid A \in \text{agent} \} \\
\text{initState (Friend i)} & \doteq \{ \text{Key}(\text{shrK(Friend i)}) \} \\
\text{initState Spy} & \doteq \{ \text{Key}(\text{shrK } A) \mid A \in \text{bad} \}
\end{align*}
\]
3.3 Events, Knowledge, and Freshness

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

\[
\text{pubK : agent} \rightarrow \text{key} \\
\text{priK } x \doteq \text{invKey(pubK } x) : \text{agent} \rightarrow \text{key}
\]

with the proviso that all symmetric keys are excluded.

\[
\text{priK } x \neq \text{pubK } x
\]

For these systems the initial state would be:

\[
\begin{align*}
\text{initState Server} & \doteq \{ \text{Key(priK Server)} \} \cup \{ \text{Key(pubK } A) \mid A \in \text{agent} \} \\
\text{initState (Friend } i) & \doteq \{ \text{Key(priK(Friend } i)) \} \cup \{ \text{Key(pubK } A) \mid A \in \text{agent} \} \\
\text{initState Spy} & \doteq \{ \text{Key(priK } A) \mid A \in \text{bad} \} \\
& \cup \{ \text{Key(pubK } A) \mid A \in \text{agent} \}
\end{align*}
\]

**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:

\[
\text{knows : agent} \rightarrow (\text{event list} \rightarrow \mathcal{P}(msg))
\]

in the following way:

\[
\begin{align*}
\text{knows } A \ [] & \doteq \text{initState } A \\
\text{knows } A \ (\text{Blk } A' \ P)' \# evs & \doteq \text{knows } A \ evs \\
\text{knows } A \ (\text{Snd } A' \ P)' \# evs & \doteq \begin{cases} 
\{ \text{MsgPkt } P \} \cup \text{knows } A \ evs & \text{if } A' \in A \\
\text{knows } A \ evs & \text{otherwise}
\end{cases} \\
\text{knows } A \ (\text{Rcv } A' \ P)' \# evs & \doteq \begin{cases} 
\{ \text{MsgPkt } P \} \cup \text{knows } A \ evs & \text{if } A' \in A \\
\text{knows } A \ evs & \text{otherwise}
\end{cases}
\end{align*}
\]

**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

\[
\text{spies } \equiv \text{knows Spy}
\]
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:

\[ \text{used} : \text{event list} \rightarrow \mathcal{P}(\text{msg}) \]

as

\[
\begin{align*}
\text{used } [] & = \bigcup A.\text{parts}(\text{initState } A) \\
\text{used } ((\text{Snd } A B X) \# evs) & = \text{parts}\{X\} \cup \text{used } evs \\
\text{used } ((\text{Rcv } A B X) \# evs) & = \text{used } evs \\
\text{used } ((\text{Blk } A B X) \# evs) & = \text{used } evs
\end{align*}
\]

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:

\[ \text{visible} : \text{Pkt} \rightarrow (\text{event list} \rightarrow \text{bool}) \]
which is defined recursively by the following cases:

\[ P \text{ visible } [] \triangleq false \]

\[ P \text{ visible } (\text{Snd } n \ p)\#e\text{vs} \triangleq \begin{cases} 
  true & \text{if } P = p \\
  P \text{ visible } e\text{vs} & \text{otherwise}
\end{cases} \]

\[ P \text{ visible } (\text{Rcv } n \ p)\#e\text{vs} \triangleq P \text{ visible } e\text{vs} \]

\[ P \text{ visible } (\text{Blk } n \ p)\#e\text{vs} \triangleq \begin{cases} 
  false & \text{if } P = p \\
  P \text{ visible } e\text{vs} & \text{otherwise}
\end{cases} \]

### 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:

\[ \text{trace} : \mathcal{P}(\text{event list}) \]

as the least inductive closure of the following rules:

\[
\begin{align*}
[] & \in \text{trace} \\
(Snd \ n \ p)\#h & \in \text{trace} \\
(Rcv \ n \ p)\#h & \in \text{trace} \\
(Blk \ n \ p)\#h & \in \text{trace}
\end{align*}
\]

In general, realistic models should be restrictions of \text{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 e\text{vs} 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

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

given by

$$[\] \downarrow n \triangleq []$$

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

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

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

### 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 \textit{Ideal} as a wildcard for the descriptive name of a protocol then its definition would be typed:

$$\text{Ideal} : \mathcal{P}(\text{event list})$$
and defined as the least inductive closure of the rules:

\[ \boxed{\emptyset \in \text{Ideal}} \]

\[ h \in \text{Ideal} \quad \boxed{\langle (A, A) \rightarrow (B, B) : m \rangle \text{ visible } h} \]

\[ \boxed{\text{Blk} \ A \langle (A, A) \rightarrow (B, B) : m \rangle \# \text{Rcv} \ A \langle (A, A) \rightarrow (B, B) : m \rangle \# h \in \text{Ideal}} \]

\[ h \in \text{Ideal} \quad \boxed{\text{Snd} \ A \langle (A, A) \rightarrow (B, B) : M_1 \rangle \# h \in \text{Ideal}} \]

\[ h \in \text{Ideal} \quad \boxed{\text{hd} \ (h \downarrow A) = \text{Blk} \ A \langle (A, A) \rightarrow (B, B) : M_1 \rangle} \]

\[ \boxed{\text{Snd} \ A \langle (A, A) \rightarrow (B, B) : M_2 \rangle \# h \in \text{Ideal}} \]

\[ \vdots \]

\[ h \in \text{Ideal} \quad \boxed{\text{hd} \ (h \downarrow A) = \text{Blk} \ A \langle (A, A) \rightarrow (B, B) : M_{last-1} \rangle} \]

\[ \boxed{\text{Snd} \ A \langle (A, A) \rightarrow (B, B) : M_{last} \rangle \# h \in \text{Ideal}} \]

where \( m \) denotes any message of the protocol and the \( M_i \)'s denote messages of the kind that the protocols requires to be sent and received in its \( i \)th 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:

\[ pLn\text{Schema} = Nm \times Nm \times msg \]

which means that the full specifications belong to:

\[ p\text{Schema} = pLn\text{Schema list} \]

### 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

\[ \uparrow : event \text{ list} \to (Nm \to p\text{Schema}) \]

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

\[
\begin{align*}
\emptyset \uparrow n & \equiv \emptyset \\
\text{Snd }m \ (\langle n_1, l_1 \rangle \to (n_2, l_2) : M) \#evs \uparrow n & \equiv \\
& \begin{cases} \\
\langle n_1 \to n_2 : M \rangle \#(evs \uparrow n) & \text{if } n = m \land n = n_1 \\
evs \uparrow n & \text{otherwise} \\
\end{cases} \\
\text{Rcv }m \ (\langle n_1, l_1 \rangle \to (n_2, l_2) : M) \#evs \uparrow n & \equiv \\
& \begin{cases} \\
\langle n_1 \to n_2 : M \rangle \#(evs \uparrow n) & \text{if } n = m \land n = n_1 \\
evs \uparrow n & \text{otherwise} \\
\end{cases} \\
(Blk \ m \ p) \#evs \uparrow n & \equiv (evs \uparrow n)
\end{align*}
\]

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

\[
\text{Agents} : \text{event list} \rightarrow \mathcal{P} \,(\text{agent})
\]

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

\[
\triangle : \text{event list} \rightarrow (\mathcal{P}(\text{event list}) \rightarrow \text{bool})
\]

we formally define the notion of consistency:

\[
evs \triangle \text{ideal} \equiv \exists \text{trace} \land \\
\forall a \in (\text{Agents evs}) \cdot \exists \text{idl} \in \text{ideal} \cdot (\text{idl} \uparrow a = \text{evs} \uparrow a)
\]

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:

\[
\text{finish} : \text{event list} \rightarrow (\text{agent} \rightarrow \text{bool})
\]

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:

\[
\nabla : \text{event list} \rightarrow (\text{agent} \rightarrow \text{bool})
\]
as

\[ evs \downarrow a = (evs \triangle \text{Ideal}) \land (evs \text{ finish } a) \land \]
\[ \exists e \in \text{set} (evs \downarrow a) \cdot (\text{fakeEvent } e) \]
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 Oorschot [DvOW92]. The following is its specification in Alice-and-Bob notation (§2.1):

1. $A \rightarrow B : \alpha^{x_A}$
2. $B \rightarrow A : \alpha^{x_B}, \{\alpha^{x_B}, \alpha^{x_A}\}_{S_B}$
3. $A \rightarrow B : \{\alpha^{x_A}, \alpha^{x_B}\}_{S_A}$

The protocol enables two parties to agree on a shared secret (an integer valued key) by the exchange of public information. In the figure $\alpha$ 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

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

is given by the least inductive closure of the following rules

- \( \Box \in StS \)
- \( evs \in StS \) \( \langle (A,A) \rightarrow (B,B) : m \rangle \) visible \( evs \)
- \( \text{Blk} A \ \langle (A,A) \rightarrow (B,B) : m \rangle \) \( \text{Rev} A \ \langle (A,A) \rightarrow (B,B) : m \rangle \) \#\( evs \in StS \)
- \( evs \in StS \) \( \alpha^X \notin \) used \( evs \)
- \( \text{Send} A \ \langle (A,A) \rightarrow (B,B) : \alpha^X \rangle \) \#\( evs \in StS \)
- \( evs \in StS \) \( \text{hd}(evs \downarrow B) = \text{Blk} B \ \langle (A,A) \rightarrow (B,B) : \alpha^X \rangle \)
- \( \text{Send} B \ \langle (B,B) \rightarrow (A,A) : \alpha^X \rangle \) \( \\{(\alpha^X,\alpha^X) \in S_B \} \) \#\( evs \in StS \)
- \( evs \in StS \) \( \text{hd}(evs \downarrow A) = \text{Blk} A \ \langle (B,B) \rightarrow (A,A) : \alpha^X \rangle \) \( \\{(\alpha^X,\alpha^X) \in S_A \} \) \#\( evs \in StS \)

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 \( \alpha^X \) 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

$$\text{HKey } X = \text{Nonce } X : \mathbb{N} \rightarrow \text{msg}$$
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

$$combiK : \mathbb{N} \times \mathbb{N} \rightarrow key$$

with the proviso that

$$combiK(\ (A,B) \) \in 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 \implies.

\begin{align*}
\text{Nil} & \quad [] \in StS \\
\text{Rcv} & \quad [[ h \in StS; \ ((B,B) \rightarrow (A,A) : m) \visible h ]] \implies \\
& \quad \text{Blk } A \ ((B,B) \rightarrow (A,A) : m) \# \\
& \quad \text{Rcv } A \ ((B,B) \rightarrow (A,A) : m) \# h \in StS \\
\text{Msg1} & \quad [[ h \in StS; \ HKey XA \notin h ]] \implies \\
& \quad \text{Snd } A \ ((A,A) \rightarrow (B,B) : \text{HKey XA}) \# h \in StS \\
\text{Msg2} & \quad [[ h \in StS; \ HKey XB \notin h; \ \text{hd } (h \downarrow B) = \\
& \quad \text{Blk } B \ ((A,A) \rightarrow (B,B) : \text{HKey XA})] \implies \\
& \quad \text{Snd } B \ ((B,B) \rightarrow (A,A) : \{ \text{HKey XB, Crypt (combiK (X,A,XB)) (Sign[B] \{ \text{HKey XB, HKey XA} \})\}) \# h \in StS \\
\text{Msg3} & \quad [[ h \in StS; \ \text{hd } (h \downarrow A) = \\
& \quad \text{Blk } A \ ((B,B) \rightarrow (A,A) : \{ \text{HKey XB, Crypt (combiK (X,A,XB)) (Sign[B] \{ \text{HKey XB, HKey XA} \})\}) \implies \\
& \quad \text{Snd } A \ ((A,A) \rightarrow (B,B) : \text{Crypt (combiK (X,A,XB)) (Sign[A] \{ \text{HKey XA, HKey XB} \})}) \# h \in StS \\
\end{align*}

Except for the fortification this is identical to the actual Isabelle/HOL encoding presented in appendix A.A.1.

This model describes a restricted subset of the well-formed traces. This has been established formally. We have
Theorem 4.1.

\[ \text{StS} \subseteq \text{trace} \]

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

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:

\[
\text{repeat} = \begin{cases} 
\text{Snd } A & \{((A, A) \to (B, B) : \alpha^{X_A}) \} \\
\text{Rcv } B & \{((A, A) \to (B, B) : \alpha^{X_A}) \} \\
\text{Blk } B & \{((A, A) \to (B, B) : \alpha^{X_A}) \} \\
\text{Snd } B & \{((B, B) \to (A, A) : \alpha^{X_B}, \{\alpha^{X_B}, \alpha^{X_A}\}_{S_B})_K \} \\
\text{Rcv } A & \{((B, B) \to (A, A) : \alpha^{X_B}, \{\alpha^{X_B}, \alpha^{X_A}\}_{S_B})_K \} \\
\text{Blk } A & \{((B, B) \to (A, A) : \alpha^{X_B}, \{\alpha^{X_B}, \alpha^{X_A}\}_{S_B})_K \} \\
\text{Snd } A & \{((A, A) \to (B, B) : \{\alpha^{X_A}, \alpha^{X_B}\}_{S_A})_K \} \\
\text{Rcv } B & \{((A, A) \to (B, B) : \{\alpha^{X_A}, \alpha^{X_B}\}_{S_A})_K \} \\
\text{Blk } B & \{((A, A) \to (B, B) : \{\alpha^{X_A}, \alpha^{X_B}\}_{S_A})_K \} \\
\text{Snd } A & \{((A, A) \to (C, C) : \alpha^{X_A}) \} \\
\text{Rcv } C & \{((A, A) \to (C, C) : \alpha^{X_A}) \} \\
\text{Blk } C & \{((A, A) \to (C, C) : \alpha^{X_A}) \} \\
\text{Snd } C & \{((C, C) \to (A, A) : \alpha^{X_C}, \{\alpha^{X_C}, \alpha^{X_A}\}_{S_C})_K \} \\
\text{Rcv } A & \{((C, C) \to (A, A) : \alpha^{X_C}, \{\alpha^{X_C}, \alpha^{X_A}\}_{S_C})_K \} \\
\text{Blk } A & \{((C, C) \to (A, A) : \alpha^{X_C}, \{\alpha^{X_C}, \alpha^{X_A}\}_{S_C})_K \} \\
\text{Snd } A & \{((A, A) \to (C, C) : \{\alpha^{X_A}, \alpha^{X_C}\}_{S_A})_K \} \\
\text{Rcv } C & \{((A, A) \to (C, C) : \{\alpha^{X_A}, \alpha^{X_C}\}_{S_A})_K \} \\
\text{Blk } C & \{((A, A) \to (C, C) : \{\alpha^{X_A}, \alpha^{X_C}\}_{S_A})_K \}
\end{cases}
\]

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

\[ \text{repeat} \in \text{StS} \]

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 \( \text{repeat} \) is consistent with \( \text{StS} \). That is, if the participants choose fresh nonces - all nonces pairwise unequal - the following property holds:

\[ \text{repeat} \not\triangle \text{StS} \]

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

\[ \text{repeat} \not\triangle \text{StS} \equiv \text{repeat} \in \text{trace} \land \forall a \in (\text{Agents repeat}) \cdot \exists \text{sts} \in \text{StS} \cdot (\text{sts} \uparrow a = \text{repeat} \uparrow a) \]

First prove that \( \text{repeat} \) 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 \( \text{StS} \) 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.  

\[ \square \]

### 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:

\[
\begin{align*}
\text{Snd } A & \quad \{ (A, A) \rightarrow (B, B) : \alpha^A \} \\
\text{Rcv } \text{Spy} & \quad \{ (A, A) \rightarrow (B, B) : \alpha^A \} \\
\text{Blk } \text{Spy} & \quad \{ (A, A) \rightarrow (B, B) : \alpha^A \} \\
\text{Snd } \text{Spy} & \quad \{ (C, \text{Spy}) \rightarrow (B, B) : \alpha^A \} \\
\text{Rcv } B & \quad \{ (C, \text{Spy}) \rightarrow (B, B) : \alpha^A \} \\
\text{Blk } B & \quad \{ (C, \text{Spy}) \rightarrow (B, B) : \alpha^A \} \\
\text{Snd } B & \quad \{ (B, B) \rightarrow (C, \text{Spy}) : \alpha^B, \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Rcv } \text{Spy} & \quad \{ (B, B) \rightarrow (C, \text{Spy}) : \alpha^B, \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Blk } \text{Spy} & \quad \{ (B, B) \rightarrow (C, \text{Spy}) : \alpha^B, \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Snd } \text{Spy} & \quad \{ (B, \text{Spy}) \rightarrow (A, A) : \alpha^B, \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Rcv } A & \quad \{ (B, \text{Spy}) \rightarrow (A, A) : \alpha^B, \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Blk } A & \quad \{ (B, \text{Spy}) \rightarrow (A, A) : \alpha^B, \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Snd } A & \quad \{ (A, A) \rightarrow (B, \text{Spy}) : \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Rcv } \text{Spy} & \quad \{ (A, A) \rightarrow (B, \text{Spy}) : \{ \{ \alpha^B, \alpha^A \}_B \}_K \} \\
\text{Blk } \text{Spy} & \quad \{ (A, A) \rightarrow (B, \text{Spy}) : \{ \{ \alpha^B, \alpha^A \}_B \}_K \}
\end{align*}
\]

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.

\[ \text{attack} \in \text{trace} \]

*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.

\[ \text{attack} \text{ finish Friend a} \]

*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.

\[ \text{attack} \triangleq \text{StS} \]

*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

\[ \text{attack} \triangledown \text{Friend a} \]

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

\[ \text{evs} \triangledown a \triangleq (\text{evs} \triangle \text{Ideal}) \land (\text{evs} \text{ finish a}) \land \exists e \in \text{set (evs} \downarrow a) \cdot (\text{fakeEvent e}) \]

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 $i$th 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 \( i \)th step we assume the following six disjoint sets:

- **SState**  A set of actions that advance agents to a state where they are ready to send
- **RState**  A set of 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.

\[
\text{action} = \text{SState} \cup \text{RState} \cup \text{Ver} \cup \text{Abort} \cup \text{Accept} \cup \text{Ready}
\]

For this construction we have the injections:

\[
\begin{align*}
\text{SState} : \{i : \mathbb{N} \mid 0 < i \leq n\} & \mapsto \text{action} \\
\text{RState} : \{i : \mathbb{N} \mid 0 < i \leq n\} & \mapsto \text{action} \\
\text{Ver} : \{i : \mathbb{N} \mid 0 < i \leq n\} & \mapsto \{j : \mathbb{N} \mid 0 < j \leq k_i\} \mapsto (\text{bool} \mapsto \text{action}) \\
\text{Accept} : \text{action} \\
\text{Abort} : \text{action} \\
\text{Ready} : \text{action}
\end{align*}
\]
5.1 A Suitable Set of Actions

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

1. \( A \to B : T_{11}, T_{21}, \ldots, T_{k1} || M_1 || O_{k1}, O_{21}, \ldots, O_{k+1} \)
2. \( A \to B : T_{12}, T_{22}, \ldots, T_{k2} || M_2 || O_{12}, O_{22}, \ldots, O_{k2} + 1 \)

\[ \vdots \]

n. \( A \to B : T_{1n}, T_{2n}, \ldots, T_{kn} || M_n || O_{1n}, O_{2n}, \ldots, O_{kn + 1} \)

When the sender in the kth step performs the \( T_{ik} \) 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 \( O_{k_i + 1} \) has just been prepared, which commits the agent to the completion of the actions \( T_{ik} \). We do not need to distinguish between any of these actions and we gather the \( O_{k_i + 1} \) action and the \( T_{ik} \) actions into \textbf{SState} \( i \). The special case \textbf{SState} \( 1 \) does not contain an \( O \) action.

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

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

The remaining \( O_{kj} \) actions are verification actions \textbf{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 \textbf{Ver} action carries a \textbf{False}. When this is the case the next action performed is the execution abort action \textbf{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 \textbf{Ready}. When \( A \) is 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 \textbf{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:

\[
\begin{align*}
Nil & \quad [] \in ltrace A \\
Ready & \quad [\text{acs} \in ltrace A; \ \text{hd acs} = \text{hd } [] \lor \\
& \quad \text{hd acs} = \text{Accept} \lor \text{hd acs} = \text{Abort}] \implies \text{Ready} \neq \text{acs} \in ltrace A \\
Init & \quad \text{[acs} \in ltrace A; \ \text{hd acs} = \text{Ready} \\
& \quad \text{SState } 1 \neq \text{acs} \in ltrace A \\
Resp & \quad \text{[acs} \in ltrace A; \ \text{hd acs} = \text{Ready} \\
& \quad \text{RState } 1 \neq \text{acs} \in ltrace A \\
Rcv & \quad \text{[acs} \in ltrace A; \ \text{hd acs} = \text{RState } i] \implies \\
& \quad \text{Ver } i \ b \neq \text{acs} \in ltrace A \\
Snd & \quad \text{[acs} \in ltrace A; \ \text{hd acs} = \text{SState } i; \ i < n] \implies \\
& \quad \text{RState } (i+1) \neq \text{acs} \in ltrace A \\
Ver & \quad \text{[acs} \in ltrace A; \ \text{hd acs} = \text{Ver } i \ j \ True; \\
& \quad \text{j } < \text{ k} ] \implies \\
& \quad \text{Ver } i \ (j+1) \ b \neq \text{acs} \in ltrace A
\end{align*}
\]
5.2 The Local Model

\[\text{Ver} \quad \# \acs \in \lttrace A; \ \hd \acs = \text{Ver} \ i \ k \ True; \\
i < n \implies \text{SState} \ (i+1) \neq \acs \in \lttrace A\]

\[\text{Accept} \quad \# \acs \in \lttrace A; \ \hd \acs = \text{SState} \ n \vee \\
\hd \acs = \text{Ver} \ n \ k \ True \implies \text{Accept} \# \acs \in \lttrace A\]

\[\text{Abort} \quad \# \acs \in \lttrace A; \ \hd \acs = \text{Ver} \ i \ j \ False \implies \text{Abort} \# \acs \in \lttrace A\]

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:

\begin{align*}
Nil \quad & \mathbf{[]} \in \mathit{itrace} \ A \\
\mathit{Ready} \quad & \mathbf{[\text{acs} \in \mathit{itrace} \ A; \ \text{hd} \ \text{acs} = \ \text{hd} \ \mathbf{[]} \ \vee} \\
& \text{hd} \ \text{acs} = \ \text{Accept} \ \vee \ \text{hd} \ \text{acs} = \ \text{Abort}]} \implies \text{Ready} \neq \text{acs} \in \mathit{itrace} \ A \\
\mathit{Init} \quad & \mathbf{[\text{acs} \in \mathit{itrace} \ A; \ \text{hd} \ \text{acs} = \ \text{Ready}} \\
& \text{SState} \ 1 \neq \ \text{acs} \in \mathit{itrace} \ A \\
\mathit{Resp} \quad & \mathbf{[\text{acs} \in \mathit{itrace} \ A; \ \text{hd} \ \text{acs} = \ \text{Ready}} \\
& \text{RState} \ i \neq \ \text{acs} \in \mathit{itrace} \ A \\
\mathit{Rcv} \quad & \mathbf{[\text{acs} \in \mathit{itrace} \ A; \ \text{hd} \ \text{acs} = \ \text{RState} \ i]} \implies \\
& \text{Ver} \ i \ 1 \ \text{b} \neq \ \text{acs} \in \mathit{itrace} \ A \\
\mathit{Snd} \quad & \mathbf{[\text{acs} \in \mathit{itrace} \ A; \ \text{hd} \ \text{acs} = \ \text{SState} \ i; \ i < 2]} \implies \\
& \text{RState} \ (\text{suc} \ i) \neq \ \text{acs} \in \mathit{itrace} \ A
\end{align*}
5.2 The Local Model

\[ \text{Veri} \quad [\text{acs} \in \text{itrace } A; \text{hd } \text{acs} = \text{Ver } i \ 1; \ i < 2] \implies \text{SState (suc } i) \neq \text{acs} \in \text{itrace } A \]

Accept \[ [\text{acs} \in \text{itrace } A; \text{hd } \text{acs} = \text{Ver } 2 \ 1 \ True \ \lor \hd \text{acs} = \text{SState } 2] \implies \text{Accept } \# \text{acs} \in \text{itrace } A \]

Abort \[ [\text{acs} \in \text{itrace } A; \text{hd } \text{acs} = \text{Ver } i \ j \ False] \implies \text{Abort } \# \text{acs} \in \text{itrace } A \]

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.

\[ \text{chkAgent } A = A \notin \text{bad} : \text{agent} \rightarrow \text{bool} \]

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.

\[ \forall \text{acs} \in \text{itrace } A \cdot \text{acs} = a \#(\text{Ver } 1 \ 1 \ (\text{chkAgent } \text{Spy})) \]

\#(\text{acts} :: \text{action list}) \implies a = \text{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.

\[ \exists \text{acts} \in \text{itrace } A \cdot \text{Accept} \#(\text{acts} \in \text{itrace } A) \]

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.\footnote{Due to a bug in the batch processor of Isabelle this theory is pretty-printed with the a2ps pretty-printer presented in appendix E.}

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
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, ≤, 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 ≤ 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.
6.1 A Suitable Interval Logic

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 \( \Box_l \) and \( \Box_r \) 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:

\[
\text{intv} = \{ [b, e] \mid b, e \in \mathbb{R} \land b \leq e \}
\]

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:
\[ \phi \bowtie \psi \] : The interval can be partitioned so \( \phi \) holds on the first part and \( \psi \) on the last.
\[ \lozenge \phi \] : There is a subinterval where \( \phi \) holds.
\[ \lozenge_1 \phi \] : There is a left neighborhood where \( \phi \) holds.
\[ \lozenge_r \phi \] : There is a right neighborhood where \( \phi \) holds.
\[ \lozenge_l \phi \] : There is a converse left neighborhood where \( \phi \) holds.
\[ \lozenge_r \phi \] : There is a converse right neighborhood where \( \phi \) holds.
\[ \lozenge_0 \phi \] : There is an interval (somewhere) where \( \phi \) holds.
\[ \lozenge \phi \] : \( \phi \) holds on all subintervals.
\[ \lozenge_t \] : \( \phi \) holds on all left neighborhoods.
\[ \lozenge_r \] : \( \phi \) holds on all right neighborhoods.
\[ \lozenge_l \phi \] : \( \phi \) holds on all converse left neighborhoods.
\[ \lozenge_r \phi \] : \( \phi \) holds on all converse right neighborhoods.
\[ \lozenge_0 \phi \] : \( \phi \) holds on all proper subintervals.
\[ \lozenge_a \phi \] : \( \phi \) holds on all intervals (anywhere).

For example \( \lozenge_0 \phi \) holds on \( [b, e] \) if there is a time \( t \), where \( t \geq e \), and \( \phi \) 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 \( \lozenge_l \phi \) holds on \( [b, e] \) if there is a time \( t \), where \( t \leq e \), and \( \phi \) holds on \( [t, e] \). This is the converse of \( \lozenge_t \phi \) and \( [t, e] \) is a converse left neighborhood. Similarly \( [b, t] \) is a converse right neighborhood if \( b \leq t \).

The formulas \( \lozenge \phi \) and \( \lozenge \phi \) are closely related. If we have times \( t_1 \) and \( t_2 \) such that \( t_1 \leq t_2 \) and \( \phi \) holds on \( [t_1, t_2] \), then \( \lozenge \phi \) holds on \( [b, e] \) if \( b \leq t_1 \wedge t_2 \leq e \), while \( \lozenge \phi \) 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 \( \lozenge, \square, \lozenge_1, \lozenge_r, \square_l, \square_r \), 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.
6.1 A Suitable Interval Logic

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 $\ell$, which denotes the length of the current interval, is flexible. The relational symbol '$=$' and the functional symbols '$+$', '$-$' 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 even talk about a modal logic as the whole structure would collapse into ordinary FOL. So obviously, rigidity and chop-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 $\chi$ holds on the bounded and closed interval $[b, e]$". Here $\chi$ can be any formula of NL.

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

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:
\[
\frac{\text{\texttt{\textit{\textbf{\texttt{\textit{-label Ri}}}} \ P}}}{\text{\texttt{\textit{\textbf{\texttt{\textit{\textit{\textit{\textit{-label Ei}}}}}}}}}} \quad \frac{\text{\texttt{\textit{\textbf{\texttt{\textit{-label Ef}}}}}}}{\text{\texttt{\textit{\textbf{\texttt{\textit{\textit{\textit{\textit{-label Ef}}}}}}}}}}
\]
where \texttt{\textit{\textbf{\texttt{\textit{-label Ei}}}}} requires proof that \texttt{\textit{\textbf{\texttt{\textit{-label Ef}}}}} 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 \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-precedes-or-equals}}}}}}} operator in HOL. We define the infix operator:

\[
\preceq: \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list}}}}}}} \to (\texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list}}}}}}} \to \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-bool}}}}}}})
\]
which basically means: “is-a-prefix-of”. A suitable definition is:

\[
x s \preceq y s \equiv \exists z s : \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list}}}}}}} \cdot y s = z s \otimes x s
\]
where \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list concatenation}}}}}}} or \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-append}}}}}} as known from ML. In the context of inductively defined traces we say that \(x s \preceq y s\) means: “\(x s\) is \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-causally}}}}} before \texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-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:

\[
\frac{\text{\texttt{\textit{\textbf{\texttt{\textit{-list}}}}}}}{\text{\texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list}}}}}}}}}
\quad \frac{\text{\texttt{\textit{\textbf{\texttt{\textit{-list}}}}}} \text{\texttt{\textit{\textbf{\texttt{\textit{-list}}}}}}}{\text{\texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list}}}}}}}}}
\quad \frac{\text{\texttt{\textit{\textbf{\texttt{\textit{-list}}}}}}}{\text{\texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list}}}}}}}}}
\quad \frac{\text{\texttt{\textit{\textbf{\texttt{\textit{-list}}}}}}}{\text{\texttt{\textit{\textbf{\texttt{\textit{\texttt{\textit{-list}}}}}}}}}
\]

6.2 Timed Lists

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 NLDCOL.

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

\[
\frac{\text{ri}(s) \land \text{ri}(t)}{\text{ri}(s \leq t)} \quad \frac{\text{ri}(s \leq t) \land \text{ri}(s)}{\text{ri}(t)} \quad \frac{\text{ri}(s \leq t) \land \text{ri}(s)}{\text{ri}(t)} \quad \frac{\text{ri}(s = t) \land \text{ri}(s)}{\text{ri}(t)} \quad \frac{\text{ri}(s = t) \land \text{ri}(s)}{\text{ri}(t)} \quad \frac{\text{ri}(s = t) \land \text{ri}(s)}{\text{ri}(t)}
\]

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:

\[
\frac{\text{cf}(s \leq t)}{\text{cf}(s = t)} \quad \frac{\text{cf}(s \leq t)}{\text{cf}(s = t)}
\]

6.2.3 Timed Operators

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

Both relations are reflexive and transitive.

\[
\frac{w : xs \leq xs}{\text{refl}} \quad \frac{w : xs = xs}{\text{refl}}
\]

\[
\frac{w : xs \leq ys}{w : xs \leq zs} \quad \frac{w : ys \leq zs}{\text{trans}} \quad \frac{w : xs = ys}{w : ys = zs} \quad \frac{w : ys = zs}{\text{trans}}
\]

The relation $\leq$ is antisymmetric while $=$ is symmetric.

\[
\frac{w : xs \leq ys}{w : xs = ys} \quad \frac{w : ys \leq xs}{\text{anti}} \quad \frac{w : ys = xs}{w : xs = ys} \quad \frac{w : ys = zs}{\text{sym}}
\]

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

\[
\frac{w : xs = ys}{w : xs \leq ys} = \text{weaken1} \quad \frac{w : xs = ys}{w : ys \leq xs} = \text{weaken2}
\]
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 ≤ and equ for = in names.

6.3.1 The Flexible Constant Tr

In NLDCOL 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:

\[ ℓ = λ b · e − b : \text{real} × \text{real} → \text{real} \]

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:

\[ \text{Tr} ∈ \text{event list} \]

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.](image_url)
6.3 Timed Global Traces

Instead we associate the value of $\text{Tr}$ 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 $\text{Tr}$ grows monotonously with time and therefore $s \leq u$. The following axiomatic rules capture this:

\[
\frac{w : \text{Tr} = evs \quad ri(evs)}{w : \square_i(\text{Tr} = evs)} \quad \text{Tr}_{\text{mon}1} \quad \frac{w : \text{Tr} = evs \quad ri(evs)}{w : \square_r(evts \leq \text{Tr})} \quad \text{Tr}_{\text{mon}2}
\]

\[
\frac{w : \text{Tr} = evs \quad ri(evs)}{w : \square_r(\square_i(\text{Tr} \leq evs))} \quad \text{Tr}_{\text{mon}3}
\]

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

\[
\frac{w : \text{Tr} = evs \quad ri(evs)}{w : \square_r(\text{evts} \leq \text{Tr})} \quad \text{Tr}_{\text{mon}5} \quad \frac{w : \text{Tr} = evs \quad ri(evs)}{w : \square_r(\square_i(evs \leq \text{Tr}))} \quad \text{Tr}_{\text{mon}6}
\]

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

\[
\frac{ri(evs)}{w : evts \leq \text{Tr} \iff (\exists x \cdot \text{Tr} = x \land evts \leq x)} \quad \text{Tr}_{\leq} \text{Tr}_{\text{expand}}
\]

\[
\frac{ri(evs)}{w : \text{Tr} \leq evts \iff (\exists x \cdot \text{Tr} = x \land x \leq evts)} \quad \text{Tr}_{\text{Tr} \leq} \text{Tr}_{\text{expand}}
\]

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

\[ \text{Tr\_empty} \equiv \text{Tr} = [] : \text{bool} \]

to assert that the trace of the current interval is \textit{empty}.

The binary predicate symbol

\[
\text{Tr\_throughout : agent} \rightarrow (\text{event list} \rightarrow \text{bool}) \\
\text{Tr\_throughout } A \text{ evs} \equiv 0 < \ell \land \square(\text{Tr} \downarrow A = \text{evs})
\]

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

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

\[
\text{Tr\_stable : agent} \rightarrow \text{bool} \\
\text{Tr\_stable } A \equiv \exists \text{evs} : \text{event list} \cdot \text{Tr\_throughout } A \text{ evs}
\]

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

\[
\text{Tr\_const} \equiv \forall A : \text{agent} \cdot \text{Tr\_stable } A : \text{bool}
\]

When an agent A \textit{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

\[
\text{Tr\_causes : agent} \rightarrow (\text{event} \rightarrow \text{bool}) \\
\text{Tr\_causes } A \text{ e} \equiv \ell = 0 \land (\exists \text{evs} : \text{event list} : \\
\text{Tr} \downarrow A = \text{e#evs} \land \square, \text{Tr\_throughout } A \text{ evs})
\]

The existence of any agent, A, causing the event e implies that e \textit{occurs} in the trace.

\[
\text{Tr\_occurs : event} \rightarrow \text{bool} \\
\text{Tr\_occurs } e \equiv \exists A : \text{Tr\_causes } A \text{ e}
\]
6.3 Timed Global Traces

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

\[
\text{Tr}_{\text{quiet}} : \text{agent} \rightarrow \text{bool} \\
\text{Tr}_{\text{quiet}} A \equiv \ell = 0 \land \lnot (\exists e \cdot \text{Tr}_{\text{causes}} A \ e)
\]

**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.

\[
(i, j) : \Diamond_i (\text{Tr}_{\text{stable}} A) \rightarrow \text{Tr}_{\text{const}} (i, j) : \Diamond_i (\text{Tr}_{\text{stable}} A) \\
\text{Tr}_{\text{const}} (i, j) : \Diamond_i (\text{Tr}_{\text{stable}} A)
\]

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.

\[
(i, j) : \text{Tr}_{\text{quiet}} A \\
(i, j) : \ell = 0 \land \left( \exists evs \cdot \text{Tr} \downarrow A = evs \land \Diamond_i (\text{Tr}_{\text{throughout}} A \ evs) \land \Diamond_i (\text{Tr}_{\text{throughout}} A \ evs) \right) \\
\text{Tr}_{\text{silence}}
\]

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

\[
(i, j) : \Diamond_i (\text{Tr}_{\text{occurs}} \text{Snd} A P) \\
(i, j) : \text{Tr}_{\text{causes}} A (\text{Snd} A P) \\
\text{Tr}_{\text{O-Snd-imp}} C \text{Snd}
\]

\[
(i, j) : \Diamond_i (\text{Tr}_{\text{occurs}} \text{Rcv} A P) \\
(i, j) : \text{Tr}_{\text{causes}} A (\text{Rcv} A P) \\
\text{Tr}_{\text{O-Rcv-imp}} C \text{Rcv}
\]

\[
(i, j) : \Diamond_i (\text{Tr}_{\text{occurs}} \text{Blk} A P) \\
(i, j) : \text{Tr}_{\text{causes}} A (\text{Blk} A P) \\
\text{Tr}_{\text{O-Blk-imp}} C \text{Blk}
\]

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”

\[
\begin{array}{l}
(i,k): Tr_{causes} A e_1 \quad (m,n): Tr_{causes} A e_2 \quad (k,j): Tr_{stable} A \\
(n,k): \ell = 0
\end{array}
\]

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

\[
\begin{array}{c}
(i,k): Tr_{causes} A e_1 \\
\hline
(l, j) \quad (n,j): Tr_{stable} A
\end{array}
\]

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

Lemma 6.2. “Tr_E_O_fud”

\[
\begin{array}{l}
(i,k): Tr_{empty} \quad (l,j): Tr_{occurs} e \\
(k,l) : \ell \geq 0
\end{array}
\]

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

Lemma 6.3. “Tr_overlap1”

\[
\begin{array}{l}
(i,j): Tr_{throughout} A e_u s \quad (k,j): Tr_{throughout} A e_v s a \\
\hline
\forall (e,v) \quad \forall (e,v,s)
\end{array}
\]

Proof. Assume that \(e_u s \neq e_v s a:\)
6.3 Timed Global Traces

\[
\begin{align*}
(k,j) &\in \text{Tr}_{\text{throughout}} A \\
(i,j) &\in \text{Tr}_{\text{throughout}} A \\
(i,k) &\in \text{Tr}_{\text{throughout}} A \\
(i,j) &\in \text{Tr}_{\text{throughout}} A
\end{align*}
\]

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. \(\Box\)

Lemma 6.4. "\text{Tr\_overlap2}"

\[
\begin{align*}
(i,j) &\in \text{Tr}_{\text{throughout}} A \\
(i,k) &\in \text{Tr}_{\text{throughout}} A \\
(i,j) &\in \text{Tr}_{\text{throughout}} A \\
(i,j) &\in \text{Tr}_{\text{throughout}} A
\end{align*}
\]

Proof. Similar to that of "\text{Tr\_overlap1}" (6.3). \(\Box\)

Lemma 6.5. "\text{Tr\_connected}"

\[
\begin{align*}
(k,j) &\in \text{Tr}_{\text{throughout}} A \\
(k,j) &\in \text{Tr}_{\text{throughout}} A \\
(k,j) &\in \text{Tr}_{\text{throughout}} A \\
(k,j) &\in \text{Tr}_{\text{throughout}} A
\end{align*}
\]

Proof.

\[
\begin{align*}
(k,j) &\in \text{Tr}_{\text{throughout}} A \\
(k,j) &\in \text{Tr}_{\text{throughout}} A \\
(k,j) &\in \text{Tr}_{\text{throughout}} A \\
(k,j) &\in \text{Tr}_{\text{throughout}} A
\end{align*}
\]

When keeping in mind that \(\text{Tr}\) 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 \(\text{Tr} \neq \text{evs}\). \(\Box\)

Lemma 6.6. "\text{Tr\_silent\_imp\_S}"

\[
\begin{align*}
w : 0 < \ell &\quad w : \text{Tr\_stable} A \vdash \text{Tr\_quiet} A \vdash \text{Tr\_stable} A \\
w &\vdash \text{Tr\_stable} A
\end{align*}
\]

Proof. This proof has been conducted formally in Isabelle. It relies on "\text{Tr\_connected}". See appendix A.6.3.3. \(\Box\)
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 ≤ 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:

\[ lTr \in \text{action list} \]

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:

\[
\frac{w : lTr = acs \quad r(acs)}{w : \sqcap t(lTr = acs)} \quad \text{(lTr\textsubscript{\text{-}mon1})} \quad \frac{w : lTr = acs \quad r(acs)}{w : \square_r(acs \leq lTr)} \quad \text{(lTr\textsubscript{\text{-}mon2})}
\]

\[
\frac{w : lTr = acs \quad r(acs)}{w : \sqcap t(\sqcap_t(lTr \leq acs))} \quad \text{(lTr\textsubscript{\text{-}mon3})}
\]

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

\[
\frac{w : lTr = acs \quad r(acs)}{w : \sqcap (lTr \leq acs)} \quad \text{(lTr\textsubscript{\text{-}mono5})} \quad \frac{w : lTr = acs \quad r(acs)}{w : \square_r(\sqcap_t(acs \leq lTr))} \quad \text{(lTr\textsubscript{\text{-}mono6})}
\]

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 ≤ shown below are also included in this appendix.

\[
\frac{r(acs)}{w : acs \leq lTr \iff (\exists x : lTr = x \land acs \leq x)} \quad \text{lTr\textsubscript{\text{-}\leq lTr\textsubscript{\text{-}expand}}}
\]

\[
\frac{r(acs)}{w : lTr \leq acs \iff (\exists x : lTr = x \land x \leq acs)} \quad \text{lTr\textsubscript{\text{-}lTr\textsubscript{\text{-}expand}}}
\]
6.4.2 Predicates Concerning \( \text{ITr} \)

We also define useful predicates for \( \text{ITr} \). 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:

\[
\text{ITr}_\text{empty} A : \text{agent} \to \text{bool} \\
\text{ITr}_\text{empty} A = \text{ITr} = [] : \text{bool}
\]

The local trace of an agent, \( A \), may have a particular value, \( \text{acs} \), throughout an interval:

\[
\text{ITr}_\text{throughout} : \text{agent} \to (\text{action list} \to \text{bool}) \\
\text{ITr}_\text{throughout} A \text{acs} = 0 < \ell \land \Box(\text{ITr} A = \text{acs})
\]

The existence of any such value, \( \text{acs} \), implies local stability for \( A \):

\[
\text{ITr}_\text{stable} : \text{agent} \to \text{bool} \\
\text{ITr}_\text{stable} A = \exists \text{acs} : \text{action list}. \text{ITr}_\text{throughout} A \text{acs}
\]

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 \( a \):

\[
\text{ITr}_\text{performs} : \text{agent} \to (\text{action} \to \text{bool}) \\
\text{ITr}_\text{performs} A a = 0 \land (\exists \text{acs} : \text{action list}. \\
\text{ITr} A = a \# \text{acs} \land \Diamond \text{ITr}_\text{throughout} A \text{acs})
\]

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

\[
\text{ITr}_\text{idles} : \text{agent} \to \text{bool} \\
\text{ITr}_\text{idles} A = 0 \land \neg (\exists a : \text{ITr}_\text{performs} A a)
\]
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:

\[
(i,j) : \Diamond_t (\triangleleft \text{Tr}_\text{stable} A) \quad \text{\(\triangleleft \text{Tr}_\text{const} \)} \\
(i,j) : \Diamond_r (\triangleleft \text{Tr}_\text{stable} A) \quad \text{\(\triangleleft \text{Tr}_\text{rcst} \)}
\]

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.

\[
(i,j) : \triangleleft \text{Tr}_\text{idles} A \\
(i,j) : \ell = 0 \land \left( \exists a \in S \cdot \triangledown \text{Tr} = a \land \right.
\left. \Diamond_t (\triangledown \text{Tr}_\text{throughout} A a \ell) \land \Diamond_r (\triangledown \text{Tr}_\text{throughout} A a \ell) \right)
\]

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.** \(\triangledown \text{Tr}_\text{PS}_\text{PS}_\text{iden}\)

\[
\begin{align*}
(i,k) : & \triangledown \text{Tr}_\text{performs} A a_1 \\
(m,n) : & \triangledown \text{Tr}_\text{performs} A a_2 \\
(k,j) : & \triangledown \text{Tr}_\text{stable} A \\
(n,k) : & \ell = 0 \\
\end{align*}
\]

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

\[
\begin{align*}
(i,k) : & \triangledown \text{Tr}_\text{performs} A a_1 \\
(m,n) : & \triangledown \text{Tr}_\text{performs} A a_2 \\
(k,j) : & \triangledown \text{Tr}_\text{stable} A \\
\end{align*}
\]

When \(A\) performs the action \(a_2\) the stability of \((k,j)\) is compromised and, thus, the assumption leads to contradiction. \(\square\)
Lemma 6.8. “\( \text{ITr}_{\_SP\_\_SP\_\_} \)\n
\[
(k, l) : \text{ITr\_performs} A a_1 \quad (m, n) : \text{ITr\_performs} A a_2 \quad (i, k) : \text{ITr\_stable} A \quad (i, m) : \text{ITr\_stable} A \\
(n, k) : \ell = 0
\]

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. \(\square\)

Lemma 6.9. “\( \text{ITr\_E\_P\_fwd} \)\n
\[
(i, k) : \text{ITr\_empty} A \quad (l, j) : \text{ITr\_performs} A e \\
(k, l) : \ell \geq 0
\]

Proof. Assuming \((l, j)\) to be temporally before \((k, k)\) contradicts our assumption about monotonicity of \(\text{ITr}\). \(\square\)

Lemma 6.10. “\( \text{ITr\_overlap1} \)\n
\[
(i, j) : \text{ITr\_throughout} A acs \quad (k, j) : \text{ITr\_throughout} A acsa \quad \ell(acs) \quad \ell(acsa) \\
(i, j) : acsa = acsa
\]

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. \(\square\)

Lemma 6.11. “\( \text{ITr\_overlap2} \)\n
\[
(i, j) : \text{ITr\_throughout} A acs \quad (k, j) : \text{ITr\_throughout} A acsa \quad \ell(acs) \quad \ell(acsa) \\
(i, j) : acsa = acsa
\]

\[
(i,k);(i'k) \models (i\mathcal{R}A=aCs \land (k,j) \models (k\mathcal{R}A=aCs \land (i,j) \models (i\mathcal{R}A=aCs))
\]

Proof.

\[
(i,k);(i'k) \models (i\mathcal{R}A=aCs \land (k,j) \models (k\mathcal{R}A=aCs \land (i,j) \models (i\mathcal{R}A=aCs))
\]

When keeping in mind that \(\mathcal{R}A\) 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 \(\mathcal{R}A \neq aCs\).

Lemma 6.13. “Tr_idling_imp_S”

\[
\begin{align*}
\mathcal{W} & : 0 < \ell \\
\mathcal{W} : \mathcal{R}\text{-stable} & \implies \mathcal{R}\text{-idles} \\
\mathcal{W} : \mathcal{R}\text{-idles} & \implies \mathcal{R}\text{-stable}
\end{align*}
\]

Proof. This proof has been conducted formally in Isabelle. It relies on “Tr_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.


\[
\begin{align*}
\mathcal{W} & : \mathcal{R}\text{-performs} \ (\mathcal{V} \ k \ P) \ P \\
\mathcal{W} & : \mathcal{R}\text{-performs} \ (\mathcal{V} \ k \ True)
\end{align*}
\]

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. "\( \text{Tr} \_ P \_ \text{Ver} \_ \text{True} \)"

\[
\begin{align*}
\text{Tr} \_ P \_ \text{Ver} \_ \text{True} & \\
\neg P & \\
\text{Tr} \_ P \_ \text{Ver} \_ \text{False} & \\
\end{align*}
\]

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 \( j \)-th, of a given step, e.g. the \( i \)-th.

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 \( i \)-th.

A number of verification functions:
\[ A_{ij} : 'a \to \text{bool} \]

return \text{True} if the message item to be verified is acceptable and return \text{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:

\[
\frac{w : \text{Tr\_causes } A (\text{Rcv } A ((B, B) \rightarrow (A, A) : m_i)))}{w : \Diamond_l (\text{Tr\_causes } B (\text{Send } B (((B, B) \rightarrow (A, A) : m_i))) \rightarrow \text{True})}
\]

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

\[
\frac{w : \text{Tr\_causes } A (\text{Rcv } A ((B, B) \rightarrow (A, A) : m_i)))}{w : \Diamond_l (\neg \text{Tr\_performs } A (\text{RState } i) \rightarrow \neg \text{Tr\_stable } A)}
\]

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:

\[
\frac{w : \text{Tr\_causes } A (\text{Rcv } A ((B, B) \rightarrow (A, A) : m_i)))}{w : \Diamond_r (\neg \text{Tr\_stable } A \land l = \delta i 1) \rightarrow \neg \text{Tr\_performs } A (\text{Ver } i 1 A_{i_1} (m_i)))}
\]

Here \( \delta i 1 \) is the cost of the first verification action of the \( i \)th 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.** \( \text{"} iTr \_ P \_ uniq \text{"} \)

\[
\frac{w : iTr \_ performs A a_1 \quad w : iTr \_ performs A a_2}{w : a_1 = a_2}
\]

*Proof.* See appendix A.6.5. \( \Box \)

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

**Theorem 6.2.** \( \text{"} Tr \_ C \_ uniq \text{"} \)

\[
\frac{w : Tr \_ causes A e_1 \quad w : Tr \_ causes A e_2}{w : e_1 = e_2}
\]

*Proof.* See appendix A.6.5. \( \Box \)

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

\[
\frac{}{w : \Diamond_a (Tr \_ empty) \quad Tr \_ init}
\]

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

\[
\frac{}{w : \Diamond_a (iTr \_ empty A) \quad iTr \_ init}
\]
All agents become ready to either send or receive at some point after initialization:

\[
\begin{align*}
  w : \text{Tr}_{-\text{empty}} A & \quad w : \Diamond_r (\text{Tr}_{-\text{stable}} A \land \text{Tr}_{-\text{performs}} A \text{ Ready}) & \text{Tr}_{-\text{Ready}} \\
  \end{align*}
\]

If an agent assumes the role of \textit{initiator} during an execution he must have been ready to do so:

\[
\begin{align*}
  w : \text{Tr}_{-\text{performs}} A (\text{SState } 1) & \quad w : \Diamond_l (\text{Tr}_{-\text{performs}} A \text{ Ready} \land \text{Tr}_{-\text{stable}} A) & \text{Initiate} \\
  \end{align*}
\]

If an agent assumes the role of \textit{responder} during an execution he must have been ready to do so:

\[
\begin{align*}
  w : \text{Tr}_{-\text{performs}} A (\text{RState } 1) & \quad w : \Diamond_l (\text{Tr}_{-\text{performs}} A \text{ Ready} \land \text{Tr}_{-\text{stable}} A) & \text{Respond} \\
  \end{align*}
\]

After aborting a protocol execution agents become ready for a new execution within a maximum time frame of \(\varepsilon\):

\[
\begin{align*}
  \text{Tr}_{-\text{performs}} A \text{ Abort} & \quad w : \Diamond_r ((\text{Tr}_{-\text{stable}} A \land \ell \leq \varepsilon) \land \text{Tr}_{-\text{performs}} A \text{ Ready}) & \text{Abort} \\
  \end{align*}
\]

After accepting a protocol execution agents become ready for a new execution within a maximum time frame of \(\varepsilon\):

\[
\begin{align*}
  \text{Tr}_{-\text{performs}} A \text{ Accept} & \quad w : \Diamond_r ((\text{Tr}_{-\text{stable}} A \land \ell \leq \varepsilon) \land \text{Tr}_{-\text{performs}} A \text{ Ready}) & \text{Accept} \\
  \end{align*}
\]

**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 $i$th step of the protocol then $A$ must send the corresponding packet immediately after preparing it:

$$w : \text{Tr\_performs } A (\text{SState } i) \quad \Rightarrow \quad w : \Diamond_r ((\text{Tr\_stable } A \land \ell = \Delta i) \rightarrow \text{Tr\_causes } A (\text{Snd } A ((A, A) \rightarrow (B, B) : m_i)))$$

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

$$w : \Diamond_r ((\text{Tr\_stable } A \land \ell = \Delta i) \rightarrow \text{Tr\_performs } A (\text{RState } (i + 1)))$$

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

$$w : \Diamond_r ((\text{Tr\_stable } A \land \ell = \Delta n) \rightarrow \text{Tr\_performs } A \text{ Accept})$$

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

$$w : \text{Tr\_causes } A (\text{Rcv } A ((B, B) \rightarrow (A, A) : m_i)) \quad \Rightarrow \quad w : \Diamond_l ((\text{Tr\_causes } B (\text{Snd } B ((B, B) \rightarrow (A, A) : m_i)) \rightarrow \text{True}))$$

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

$$w : \text{Tr\_causes } A (\text{Rcv } A ((B, B) \rightarrow (A, A) : m_i)) \quad \Rightarrow \quad w : \Diamond_l ((\text{Tr\_performs } A (\text{RState } i) \rightarrow \text{Tr\_stable } A)$$

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

$$w : \text{Tr\_causes } A (\text{Rcv } A ((B, B) \rightarrow (A, A) : m_i)) \quad \Rightarrow \quad w : \Diamond_r ((\text{Tr\_stable } A \land \ell = \Delta i 1) \rightarrow \text{Tr\_performs } A (\text{Ver } i 1 A_i (m_i)))$$

If the $j$th verification function of the $i$th step is completed and the outcome is True then the next verification must commence:

$$w : \Diamond_r ((\text{Tr\_stable } A \land \ell = \delta i (j + 1)) \rightarrow \text{Tr\_performs } A (\text{Ver } i (j + 1) A_{i+1} (m_i)))$$
If the $j$th is the last verification function, the $k_i$th, of the $i$th step and the outcome is True then the message of step $i + 1$ must be prepared:

$$w : ITr \text{ performs } A \ (Ver \ i \ k_i \ True)$$

$$w : \Diamond_r ((\text{ITr stable} \ A \land \ell \leq e) \land \neg \text{ITr performs } A \ (\text{SState} \ (i + 1))) \quad \text{Succ}_{Ver \ i}$$

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

$$w : ITr \text{ performs } A \ (Ver \ n \ k_n \ True)$$

$$w : \Diamond_r ((\text{ITr stable} \ A \land \ell \leq e) \land \neg \text{ITr performs } A \ (Accept)) \quad \text{Succ}_{Vern}$$

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

$$w : ITr \text{ performs } A \ (Ver \ i \ j \ False)$$

$$w : \Diamond_r ((\text{ITr stable} \ A \land \ell \leq e) \land \neg \text{ITr performs } A \ (Abort)) \quad \text{Fail}_{Ver}$$
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\(^\text{1}\). 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 captures the causal dependencies between events. Furthermore, we assume two types of delay that must also be modeled. 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:

\[
\text{Rcv} \ [w : \text{Tr\_occurs (Rcv B \ (\langle A, A \rangle \rightarrow (B, B) : m\rangle); A \neq B}] \implies \\
w : \Diamond_{l}(\langle \text{Tr\_occurs (Snd A \ (\langle A, A \rangle \rightarrow (B, B) : m\rangle); A \neq B]} \wedge \\
\text{Tr\_quiet B} \wedge (\text{Tr\_stable A} \wedge \text{Tr\_stable B}))
\]

\[
\text{Blk} \ [w : \text{Tr\_occurs (Blk B \ (\langle A, A \rangle \rightarrow (B, B) : m\rangle); A \neq B}] \implies \\
w : \Diamond_{l}(\langle \text{Tr\_occurs (Rcv B \ (\langle A, A \rangle \rightarrow (B, B) : m\rangle); A \neq B]} \wedge \\
\text{Tr\_quiet A} \wedge (l \leq \epsilon \wedge \text{Tr\_stable A} \wedge \text{Tr\_stable B}))
\]

\[
\text{Snd1} \ [w : \text{Tr\_occurs (Snd A \ (\langle A, A \rangle \rightarrow (B, B) : M1\rangle); A \neq B}] \implies \\
w : \Diamond_{l}(\text{Tr\_empty}) \vee \\
\Diamond_{l}(\langle \text{Tr\_occurs (Snd A \ (\langle A, A \rangle \rightarrow (C, C) : M3\rangle); A \neq B]} \wedge \\
\text{Tr\_quiet C} \wedge (\text{Tr\_stable A} \wedge \text{Tr\_stable C})) \vee \\
\Diamond_{l}(\langle \text{Tr\_occurs (Blk A \ (\langle C, C \rangle \rightarrow (A, A) : M3\rangle); A \neq B]} \wedge \\
\text{Tr\_quiet C} \wedge (\text{Tr\_stable A} \wedge \text{Tr\_stable C}))
\]

\[ ^2 \text{This use of } \delta \text{ and } \epsilon \text{ is non-standard with respect to the method described in section 6.5} \]


7.1 Network Delays

\[\text{Snd}2 \quad [w : \text{Tr\_occurs}(\text{Snd}\ B \ [(B, B) \rightarrow (A, A) : M2]); A \neq B] \implies \\
\quad w : \Diamond\left(\left(\text{Tr\_occurs}(\text{Blk}\ B \ [(A, A) \rightarrow (B, B) : M1]) \wedge \\
\text{Tr\_quiet}\ A \right) \rightarrow (\text{Tr\_stable}\ B \wedge \text{Tr\_stable}\ A)\right)\]

\[\text{Snd}3 \quad [w : \text{Tr\_occurs}(\text{Snd}\ A \ [(A, A) \rightarrow (B, B) : M3)]; A \neq B] \implies \\
\quad w : \Diamond\left(\left(\text{Tr\_occurs}(\text{Blk}\ A \ [(B, B) \rightarrow (A, A) : M2]) \wedge \\
\text{Tr\_quiet}\ B \right) \rightarrow (\text{Tr\_stable}\ A \wedge \text{Tr\_stable}\ B)\right)\]

\[\text{delay} \quad [w : (\text{Tr\_occurs}(\text{Snd}\ A \ P) \leftarrow \text{True} \rightarrow \text{Tr\_occurs}(\text{Rcv}\ B \ P)); A \neq B] \implies w : \delta \leq \ell\]

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.

\[A \neq B \quad w : \text{Tr\_occurs}(\text{Snd}\ A [(A, A) \rightarrow (B, B) : M2])\]

\[w : \Diamond\left(\left(\text{Tr\_occurs}(\text{Snd}\ B [(B, B) \rightarrow (A, A) : M1]) \leftarrow \text{True}\right)\right)\]

**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.\[\square\]

**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 \(\delta\) must have gone by:

\[A \neq B \quad \nu(\delta) \quad w : \text{Tr\_empty} \leftarrow \text{True} \leftarrow \text{Tr\_occurs}(\text{Rcv}\ B [(|A, A) \rightarrow (B, B) : M1])\]

\[w : \delta \leq \ell\]
\textbf{Chapter 7. Examples: Simple Protocols}

\textit{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 \( M_1 \) is sent we have a minimal derivation and enough information to derive \( \ell \geq \delta \). We stop and use the rule \text{“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. \qed

\textbf{Lemma 7.3. “snd1\_snd2\_imp\_delay”}
\textit{Assume that two agents} \( A \) \textit{and} \( B \), \textit{with rigid identities, are executing the protocol. Between the occurrence of “} \textit{A sends} \( M_1 \) \textit{to} \( B \) \textit{” and the occurrence of “} \textit{B sends} \( M_2 \) \textit{to} \( A \) \textit{” a time period of at least} \( \delta \) \textit{passes by.}

\[
\begin{align*}
A \neq B & \quad \ell(A) \quad r(B) & \quad w:\text{Tr\_causes} \ A \ (\text{Snd} \ A \ \ll (A, A) \rightarrow (B, B): M_1) \ \sim \\
& \quad \text{Tr\_stable} \ A \ \sim \text{Tr\_causes} \ B \ (\text{Snd} \ B \ \ll (B, B) \rightarrow (A, A): M_2) \quad w: \delta \leq \ell
\end{align*}
\]

\textit{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 \( \text{Snd} \ A \ \ll (A, A) \rightarrow (B, B): M_1 \) \ event but we also have to prove that it occurs inside the interval \( w \).

The proof relies on the lemmas \text{“Tr\_CS\_CS\_equ”} and \text{“Tr\_silent\_imp\_S”}. Furthermore, the model needs to be extremely explicit about the fact that when \( A \) \textit{and} \( B \) do not explicitly cause events they explicitly do not.\footnote{This is how the need for the \text{Tr\_quiet} predicate arose} 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 \textit{and} matching of endpoints. A rather lengthy and complicated affair - see appendix A.7.1.2. \quad \qed

\subsection*{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.
7.2 Rejecting Bogus Messages

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:

$${\overline{\text{r}(\Delta i)}}^{\text{R} \text{Delta}}$$

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:

$${\overline{\text{r}(\delta i j)}}^{\text{R} \text{delta}}$$

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

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

Which is also rigid:

$${\overline{\text{r}(\epsilon)}}^{\text{R} \text{epsilon}}$$

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:

$$M : \mathbb{N} \rightarrow \text{msg}$$

We take the universal properties for granted and extend the model with the following rules:
SState1  \[ w : \text{ITr} \text{ performs } A \text{ (SState 1); } A \neq B \] \implies \\
\begin{align*}
w & : \Diamond_r ((\text{ITr} \text{ stable } A \land \ell = \Delta(1)) \land \\
& \quad \Tr \text{ causes } A (\text{Snd } A (\{(A, A) \rightarrow (B, B) : M 1\})) \land \\
& \quad \Diamond_r ((\text{ITr} \text{ stable } A \land \ell = \Delta(1)) \land \\
& \quad \text{ITr} \text{ performs } A \text{ (RState 2)})
\end{align*}

SState2  \[ w : \text{ITr} \text{ performs } A \text{ (SState 2); } A \neq B \] \implies \\
\begin{align*}
w & : \Diamond_r ((\text{ITr} \text{ stable } A \land \ell = \Delta(2)) \land \\
& \quad \Tr \text{ occurs} (\text{Snd } A (\{(A, A) \rightarrow (B, B) : M 2\})) \land \\
& \quad \Diamond_r ((\text{ITr} \text{ stable } A \land \ell = \Delta(2)) \land \\
& \quad \text{ITr} \text{ performs } A \text{ Accept})
\end{align*}

SuccVer1  \[ w : \text{ITr} \text{ performs } A \text{ (Ver 1 \& True) } ] \implies \\
\begin{align*}
w & : \Diamond_r (\text{ITr} \text{ stable } A \land \text{ITr} \text{ performs } A \text{ (SState 2)})
\end{align*}

SuccVer2  \[ w : \text{ITr} \text{ performs } A \text{ (Ver 2 \& True) } ] \implies \\
\begin{align*}
w & : \Diamond_r (\text{ITr} \text{ stable } A \land \text{ITr} \text{ performs } A \text{ Accept})
\end{align*}

FailVer  \[ w : \text{ITr} \text{ performs } A \text{ (Ver k \& False)} ] \implies \\
\begin{align*}
w & : \Diamond_r (\text{ITr} \text{ stable } A \land \text{ITr} \text{ performs } A \text{ Abort})
\end{align*}

Rcv  \[ w : \Tr \text{ causes } A \text{ (Rcv } A (\{(B, B) \rightarrow (A, A) : M k\}; A \neq B) \] \implies \\
\begin{align*}
w & : \Diamond_r (\text{ITr} \text{ performs } A \text{ (RState k) \land } \text{ITr} \text{ stable } A) \land \\
& \quad \Diamond_l (\text{Tr} \text{ causes } B (\text{Snd } B (\{(B, B) \rightarrow (A, A) : M k\}) \land \\
& \quad \text{Tr} \text{ quiet } A) \land (\text{Tr} \text{ stable } A \land \text{Tr} \text{ stable } B) \land \\
& \quad \Diamond_r (\text{ITr} \text{ stable } A \land \ell = \Delta(k)(1)) \land \\
& \quad (\text{ITr} \text{ performs } A \text{ (Ver k \& (chkAgent B))))
\end{align*}

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.
7.2 Rejecting Bogus Messages

**Theorem 7.1.** “\texttt{Rcv\_bogus\_imp\_delay\_Abort}”

\[
\begin{align*}
    w : \text{Tr\_causes} A (\text{Rcv} A \{ (Spy, Spy) \rightarrow (A, A) : M 1 \}) & \quad A \neq Spy \\
    w : \diamond_r (\ell = (\delta 1 1) + e \rightarrow \text{Tr\_performs} A \text{ Abort})
\end{align*}
\]

*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. \( \Box \)

**Theorem 7.2.** “\texttt{Rcv\_bogus\_imp\_no\_Accept}”

\[
\begin{align*}
    w : \text{Tr\_causes} A (\text{Rcv} A \{ (Spy, Spy) \rightarrow (A, A) : M 1 \}) & \quad A \neq Spy \\
    \text{Tr\_stable} A \rightarrow \text{Tr\_performs} A (\text{Ver} 1 1 \text{ True}) & \quad w : \text{False}
\end{align*}
\]

*Proof.* This proof is more difficult. Our only option is to reason left-to-right from the premises until a \( \text{Tr\_performs} A (\text{Ver} 1 1 \text{ False}) \) action is found. Once this happens we have to prove that it necessarily occurs in the same point as the \( \text{Tr\_performs} A (\text{Ver} 1 1 \text{ True}) \) action. As for events this requires us to model non-activity explicitly and then rely on “\( \text{Tr\_SP\_SP\_zlen} \)” in order force a match. Formal proof-script can be found in appendix A.7.2.2. \( \Box \)
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, \((a_x, l_x) \rightarrow (a_r, s_r) : \mathcal{M}\), as the primitives of communication and a particular set of events, \(\{\text{Snd, Blk, Rcv}\}\), 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", \(\Delta\), and "contains
a successful active attack on $\nabla$, between traces and agents. Notably, $\nabla$, 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 $\textsf{StS}$ is indeed a restriction of the well-formed network traces $\textsf{trace}$.
2. Repetitions of the $\textsf{StS}$ protocol are consistent, $\Delta$, with respect to the ideal model.
3. An active attack, first shown by Lowe, is indeed recognized as a successful active attack, $\nabla$, 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, \{\texttt{Ready, Ver, SState, RState, Abort, 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, $\textsf{Tr}$, and timed local traces, $\textsf{Tr}$, 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, $\mathcal{V}$, 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 save 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.
8.2 Discussion

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 'short cuts' 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 protocols¹ 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.

¹The Station-to-Station protocol mentioned in this thesis is one such protocol.
Bibliography


kerberos by the inductive method. In Alan J. Hu and Moshe Y.
Vardi, editors, Computer-Aided Verification: CAV ’98, number

[Bri00] Neil Briscoe. Understanding the osi 7-layer model. PC Network

[CH98] Zhou Chaochen and Michael R. Hansen. An adequate first
order interval logic. In Compositionality: The Significant Dif-
ference (COMPOS’97), volume 1536 of Lecture Notes in Com-

Authentication and authenticated key exchanges. In Designs,
Codes and Cryptography, volume 2, pages 107–125. Kluwer

[DY83] D. Dolev and A. C. Yao. On the security of public key pro-
208, 1983.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Resoning
about belief in cryptographic protocols. In Proceedings 1990


[GS95] Li Gong and Paul Syverson. Fail-stop protocols: An approach
to designing secure protocols. In Proceedings of the 5th In-
ternational Working Conference on Dependable Computing for

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a
distributed system. Communications of the ACM, 21(7):558–

[Low96] Gavin Lowe. Some new attacks upon security protocols. In Pro-
cedings of the 9th 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


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 \sqcup (B \sqcup A) = B \sqcup 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
       \mid Number nat
       \midNonce nat
       \mid Key key
       \mid Hash msg
       \mid MPair msg msg
       \mid Crypt key msg

syntax (@MTuple :: \{ a, arg\} => \texttt{\textasciitilde{a} * 'b} \ ((\exists{\cdot / \cdot})))

syntax (xsymbols)
  @MTuple :: \{ a, arg\} => \texttt{\textasciitilde{a} * 'b} \ ((\exists{\cdot / \cdot}))

translations
\{lx, y, z\} == \{lx, \{ly, z\}\}
\{lx, y\} == MPair x y

defines
HPair :: [msg, msg] => msg
\((\downarrow \text{Hash}[-] /-) [0, 1000]\)
Hash[X] Y == \{[X, Y], \{Y\}\}

keysFor :: msg set => key set
keysFor H == invKey \{ K. \exists X. Crypt K X \in H\}

Operators Dealing With Sets of Messages
consts parts :: msg set => msg set
inductive parts H
  intros
  Inj [HOL intro]: X \in H ==> X \in parts H
  Fst: \{X, Y\} \in parts H ==> X \in parts H
  Snd: \{X, Y\} \in parts H ==> Y \in parts H
  Body: Crypt K X \in parts H ==> X \in parts H

lemma parts-mono: G \subseteq H ==> parts(G) \subseteq parts(H)
consts analz :: msg set => msg set
inductive analz H
  intros
  Inj [HOL intro, HOL simp]: X \in H ==> X \in analz H
  Fst: \{X, Y\} \in analz H ==> X \in analz H
  Snd: \{X, Y\} \in analz H ==> Y \in analz H
  Decrypt [HOL dest]:\{Crypt K X \in analz H; Key(invKey K); analz H\} ==>
  \{X \in analz H\}

lemma analz-mono: G \subseteq H ==> analz(G) \subseteq analz(H)
consts synth :: msg set => msg set
inductive synth H
  intros
  Inj [HOL intro]: X \in H ==> X \in synth H
  Agent [HOL intro]: Agent agt \in synth H
  Number [HOL intro]: Number n \in synth H
  Hash [HOL intro]: X \in synth H ==> Hash X \in synth H
  MPair [HOL intro]: \{X \in synth H; Y \in synth H\} ==>
  \{\{X, Y\}\} \in synth H
  Crypt [HOL intro]: \{X \in synth H; Key(K) \in H\} ==>
  Crypt K X \in synth H
lemma synth-mono: \( G \leq H \implies \text{synth}(G) \leq \text{synth}(H) \)
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 == λp. fst p ≠ snd p

goodAdr :: Adr => bool
goodAdr == λp. ¬fakeAdr p

Projections of Packets

constdefs
sAdrPkt :: Pkt => Adr
sAdrPkt == λp. fst p

rAdrPkt :: Pkt => Adr
rAdrPkt == λp. fst (snd p)

msgPkt :: Pkt => msg
msgPkt == λp. snd (snd p)

sNmPkt :: Pkt => Nm
sNmPkt == λp. fst (sAdrPkt p)
sLocPkt :: Pkt ⇒ Loc
sLocPkt ≡ λp. snd (sAdrPkt p)

rNmPkt :: Pkt ⇒ Nm
rNmPkt ≡ λp. fst (rAdrPkt p)

lemma sNmPkt: sNmPkt {x, y} (u, v): m = x
lemma sLocPkt: sLocPkt {x, y} (u, v): m = y
lemma rNmPkt: rNmPkt {x, y} (u, v): m = u
lemma rLocPkt: rLocPkt {x, y} (u, v): m = v
lemma msgPkt: msgPkt {s → r: m} = m

Judgments on Packets

constdefs
fakePkt :: Pkt ⇒ bool
fakePkt ≡ λp. fakeAdr (fst p) ∨ fakeAdr (fst (snd p))

goodPkt :: Pkt ⇒ bool
goodPkt ≡ λp. ¬fakePkt p

lemma fakeAdrE [rule-format (no-asn)]: fakeAdr (x, y) → x ≠ y
lemma fakeAdrI [rule-format (no-asn)]: x ≠ y → fakeAdr(x, y)
lemma fakePkt1s [rule-format (no-asn)]: (x ≠ y) → fakePkt (x, y) → r: m
lemma fakePkt1r [rule-format (no-asn)]: (u ≠ v) → fakePkt (u, v): m
lemma goodPktEs [rule-format (no-asn)]: goodPkt (x, y) → r: m → (x = y)
lemma goodPktEr [rule-format (no-asn)]: goodPkt {s → (u, v): m} → (u = v)
lemma fakePktEs [rule-format (no-asn)]: fakePkt (x, y) → (u, v): m → (x ≠ y) ∨ (u ≠ v)
lemma fakePkt11 [rule-format (no-asn)]: (x = y) ∧ (u = v) → goodPkt (x, y) → (u, v): m
lemma goodPkt: goodPkt {x, y} → (y, y): m
end
A.3.3 Events, Knowledge, and Freshness

**theory** Event = Packet:

**The Events of The Network**

datatype
event = Sn d Nm Pkt
| Rcv Nm Pkt
| Blk Nm Pkt

axioms

hdnil-not-event [iff]: ((e::event) = hd []) = False

constdefs

fake-event :: event ⇒ bool
fake-event ≡ λe. case e of Sn d n p ⇒ sNmpkt p ≠ n ∨ fakePkt p
| Rcv n p ⇒ rNmpkt p ≠ n ∨ fakePkt p
| Blk n p ⇒ rNmpkt p ≠ n ∨ fakePkt p

goed-event :: event ⇒ bool

goed-event e ≡ ¬ fake-event e

**The knowledge of Agents**

const

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 Sn d 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 \( A' \ P \Rightarrow \) knows Spy \( xs \)
else
(case \( x \)
of Snd \( A' \ P \Rightarrow \) if \( A' = A \)
then
insert (msgPkt \( P \)) (knows \( A \) \( xs \))
else
knows \( A \) \( xs \)
| Rcv \( A' \ P \Rightarrow \) if \( A' = A \)
then
insert (msgPkt \( P \)) (knows \( A \) \( xs \))
else
knows \( A \) \( xs \)
| Blk \( A' \ P \Rightarrow \) knows \( A \) \( xs \))

**syntax**

spies :: event list \( \Rightarrow \) msg set

**translations**

spies == knows Spy

**axioms**

Spy-in-bad: Spy \( \in \) bad
Server-not-bad: Server \( \notin \) bad

**Freshness of Message Components**

**consts** used :: event list \( \Rightarrow \) msg set

**primrec**

used-Nil: used [] = (\( \bigcup \ B \). parts (initState \( B \)))
used-Cons: used \(( x \neq xs )\) = (case \( x \)
of Snd \( A \ P \Rightarrow \) parts \{msgPkt \( P \}\} \( \cup \) (used \( xs \))
| Rcv \( A \ P \Rightarrow \) used \( xs \)
| Blk \( A \ P \Rightarrow \) used \( xs \))

end
A.3 Untimed Theory of Global Traces

Shared Key Systems
t{theory} Shared = Event :
const
shrK :: agent => key

axioms
issym-keys: K є symKeys
inj-shrK: inj shrK

prime:
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
t{theory} Public = Trace :
const
pubK :: agent => key

syntax
priK :: agent => key

translations
priK x == invKey(pubK x)

prime:
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 є symKeys => invKey K = K end
A.3.4 Traces

theory Trace = Event:

Visible 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

evproj-Nil:  [] ↓ n = []
evproj-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


A.3 Untimed Theory of Global Traces

\[
(\text{Rev} \ m \ p) \# (xs \downarrow n)
\]
\[\text{else} \]
\[ (xs \downarrow n)) \]
\[ | \ \text{Blk} \ m \ p = \geq (\text{if} \ n = m \]
\[ \text{then} \]
\[ (\text{Blk} \ m \ p) \# (xs \downarrow n) \]
\[ \text{else} \]
\[ (xs \downarrow n)) \]

end

Public Key Systems
A.3.5 Ideal Protocol Definitions

theory StSdef = Public:

consts
  Sign :: [agent, msg] ⇒ msg  \((\{\text{Sign}[\cdot]\} - [0, 1000])\)
  Sign[A] Y ≡ Crypt (priK A) Y

  HKey :: nat ⇒ msg
  HKey X ≡ Nonce X

axioms
  inj-combiK: inj combiK
  isSym-combiK: combiK (A, B) ∈ symKeys

consts
  StS :: event kstr set

inductive StS
  intros
    Nil [HOL.intro]: [] ∈ StS
    Rev [HOL.intro]: [h : StS; \{(B, B) → (A, A): m\} visible h] ⇒
    Bk A \{(B, B) → (A, A): m\} # Rev 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) = Bk 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) =
    Bk 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
\( \uparrow : [\text{event list}, \text{Nm}] \Rightarrow \text{pschema} \)  
(infix 60)

primrec
projs-Nil: \([], \uparrow \) = []
projs-Cons: \( x \# xs \uparrow n = (\text{case } x \) of \( \text{Snd } m \{ (n1, l1) \rightarrow (n2, l2); m1 \}\) ⇒ \( (\text{if } n = m \land n = n1 \) then \( (n1, n2, m1) \# (xs \uparrow n) \) else \( (xs \uparrow n) \) | \( \text{Rec } m \{ (n1, l1) \rightarrow (n2, l2); m1 \}\) ⇒ \( (\text{if } n = m \land n = n2 \) then \( (n1, n2, m1) \# (xs \uparrow n) \) else \( (xs \uparrow n) \) | \( \text{Blk } m \ p \Rightarrow xs \uparrow n \)

Consistent Executions

consts
Agents :: event list ⇒ agent set

primrec
Agents-Nil: \( \text{Agents } [] = \{ \}
Agents-Cons: \( \text{Agents } (e \# evs) = (\text{case } e \text{ of } \text{Snd } X \ p \Rightarrow \text{if } X \neq \text{Spy then insert } X (\text{Agents evs}) \text{ else } (\text{Agents evs}) \) | \( \text{Rec } X \ p \Rightarrow \text{if } X \neq \text{Spy then insert } X (\text{Agents evs}) \text{ else } (\text{Agents evs}) \) | \( \text{Blk } X \ p \Rightarrow (\text{Agents evs}) \)

constsdefs
consis :: [event list, event list set] ⇒ bool  
(- Δ -)
evs Δ ideal ≡ \( \forall a \in (\text{Agents evs}). \exists s \in \text{ideal. evs} \in \text{trace} \land (s \uparrow a = \text{evs} \uparrow a) \)
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] [lHKey aXA, HKey aXB])) ∈ set (evs ↓ a)
  ∨
  Rcv a ⇧((b, c) → (a, a): Crypt (combiK(aXA, aXB)) (Sign[b] [lHKey aXB, HKey aXA])) ∈ set (evs ↓ a)

constdefs
  sattack :: [event list, agent] ⇒ bool  (- \n -)
  evs \n a ≡ ∃ e ∈ set (evs ↓ a). (evs \n \n SIS) ∧ fake-event(e) ∧ (evs finish a)

end
A.4 Example: The Station-to-Station Protocol

theory StSAttack = Schematic:

Modeling StS

See section: Ideal Protocol Definitions

Consistency of Repeated Protocol Engagement

lemma one-app: \[ a \neq b; Xa1 \neq Xa2; Xb1 \neq Xa1; Xb1 \neq Xb2; Xb2 \neq Xa1; Xb2 \neq Xa2; Xb1 \neq Xa2 \] \( \implies (\exists s \in StS. s \uparrow \text{Friend } b = \) [(Friend a, Friend b, Crypt (combiK (Xa2, Xb2)) Sign[Friend a] \{Key Xa2, Key Xb2\}],

(Friend b, Friend a, \{Key Xb2, Key Xa2\}) Sign[Friend b] \{Key Xb2, Key Xa2\}],

(Friend a, Friend b, Key Xa2),

(Friend a, Friend b, Crypt (combiK (Xa1, Xb1)) Sign[Friend a] \{Key Xa1, Key Xb1\}],

(Friend b, Friend a, \{Key Xb1, Key Xa1\}) Sign[Friend b] \{Key Xb1, Key Xa1\}],

(Friend a, Friend b, Key Xa1))

lemma two-app: \[ a \neq b; a \neq c; Xa1 \neq Xa2; Xb \neq Xa1; Xb \neq Xc; Xc \neq Xa1; Xc \neq Xa2; Xb \neq Xa2 \] \( \implies (\exists s \in StS. s \uparrow \text{Friend } a = \) [(\{} Friend a \rightarrow Friend b: Crypt (combiK (Xa2, Xb)) Sign[Friend a] \{Key Xa2, Key Xb\} \},

\{Key Xb, Key Xa2\}],

\{Friend a \rightarrow Friend b: Key Xa2\}],

\{Friend a \rightarrow Friend c: Crypt (combiK (Xa1, Xc)) Sign[Friend a] \{Key Xa1, Key Xc\} \},

\{Key Xc, Key Xa1\}],

\{Friend a \rightarrow Friend c: Key Xa1\})]

lemma app-b: \[ a \neq b; Xa \neq Xb \] \( \implies (\exists s \in StS. s \uparrow \text{Friend } b = \) [(\{} Friend a \rightarrow Friend b: Crypt (combiK (Xa, Xb)) Sign[Friend a] \{Key Xa, Key Xb\} \}],

\{Key Xb, Key Xa\}],

\{Friend b \rightarrow Friend a: \{Key Xb, Key Xa\} \} Sign[Friend b] \{Key Xb, Key Xa\}].
\{\text{Friend } a \rightarrow \text{Friend } b: \text{HKey } X_a[\]}\}

**lemma** app-a: \[[a \neq b; X_a \neq X_b] \implies (\exists s \in \text{St}S. s \uparrow \text{Friend } a = \)
\[
\{[\text{Friend } a \rightarrow \text{Friend } b: \text{Crypt (combiK (X_a, X_b)) Sign[Friend a] [HKey } X_a, HKey X_b]\}]
\]
\[
\{\text{Friend } b \rightarrow \text{Friend } a: \{\text{HKey } X_b, \text{Crypt (combiK (X_a, X_b)) Sign[ Friend b] [HKey } X_b, HKey X_a]\}]
\]
\[
\{\text{Friend } a \rightarrow \text{Friend } b: \text{HKey } X_a[\]}
\]

**theorem** Repeat-consis: \[[a \neq b; a \neq c; X_{a1} \neq X_{a2}; X_b \neq X_{a1}; X_b \neq X_c; X_{c} \neq X_{a1}; X_{c} \neq X_{a2}; X_b \neq X_{a2}] \implies \)
\[
(\text{Blk (Friend c) ((Friend a, Friend a) \rightarrow (Friend c, Friend c): Crypt (combiK(X_{a2}, X_c)) (Sign[Friend c] [{HKey } X_{a2}, HKey X_c [})]}) \} \} \}
\]
\[
(\text{Rev (Friend c) ((Friend a, Friend a) \rightarrow (Friend c, Friend c): Crypt (combiK(X_{a2}, X_c)) (Sign[Friend c] [{HKey } X_{a2}, HKey X_c [})]}) \} \} \}
\]
\[
(\text{Snd (Friend a) ((Friend a, Friend a) \rightarrow (Friend c, Friend c): Crypt (combiK(X_{a2}, X_c)) (Sign[Friend c] [{HKey } X_{a2}, HKey X_c [})]}) \} \} \}
\]
\[
(\text{Blk (Friend b) ((Friend a, Friend a) \rightarrow (Friend b, Friend b): Crypt (combiK(X_{a1}, X_b)) (Sign[Friend a] [{HKey } X_{a1}, HKey X_b [})]}) \} \} \}
\]
\[
(\text{Rev (Friend b) ((Friend a, Friend a) \rightarrow (Friend b, Friend b): Crypt (combiK(X_{a1}, X_b)) (Sign[Friend a] [{HKey } X_{a1}, HKey X_b [})]}) \} \} \}
\]
\[
(\text{Snd (Friend a) ((Friend a, Friend a) \rightarrow (Friend b, Friend b): Crypt (combiK(X_{a1}, X_b)) (Sign[Friend a] [{HKey } X_{a1}, HKey X_b [})]}) \} \} \}
\]
\[
(\text{Blk (Friend a) ((Friend b, Friend b) \rightarrow (Friend a, Friend a): [{HKey } X_{b}, Crypt (combiK(X_{a1}, X_b)) (Sign[Friend b] [{HKey } X_{b}, HKey X_{a1} [})]}) \} \} \}
\]
\[
(\text{Rev (Friend a) ((Friend b, Friend b) \rightarrow (Friend a, Friend a): [{HKey } X_{b}, Crypt (combiK(X_{a1}, X_b)) (Sign[Friend b] [{HKey } X_{b}, HKey X_{a1} [})]}) \} \} \}
\]
\[
(\text{Snd (Friend b) ((Friend b, Friend b) \rightarrow (Friend a, Friend a): [{HKey } X_{b}, Crypt (combiK(X_{a1}, X_b)) (Sign[Friend b] [{HKey } X_{b}, HKey X_{a1} [})]}) \} \} \}
\]
\[
(\text{Blk (Friend b) ((Friend a, Friend a) \rightarrow (Friend b, Friend b): HKey X_{a1} [})]}) \} \}
\]
\[
(\text{Rev (Friend b) ((Friend a, Friend a) \rightarrow (Friend b, Friend b): HKey X_{a1} [})]}) \} \}
\]
\[
(\text{Snd (Friend a) ((Friend a, Friend a) \rightarrow (Friend b, Friend b): HKey X_{a1} [})]}) \} \} \]
\( \triangle {\text{Sts}} \)

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
A.4 Example: The Station-to-Station Protocol

| rule app-b),
(assumption | (rule not-sym, assumption))+) +
apply (rule cong)
apply force
apply (rule two-app)
by (assumption | (rule not-sym, assumption)) +

Active Attack on StS

lemma Attack-is-Trace: Bk Spy \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy})\): Crypt
(\text{combiK}(X_a, X_b)) \((\text{Sign} \text{[Friend a]} \{ \{ \text{HKey} X_a, \text{HKey} X_b \} \}) \) \# #
Rcv Spy \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy})\): Crypt (\text{combiK}(X_a, X_b))
(\text{Sign} \text{[Friend a]} \{ \{ \text{HKey} X_a, \text{HKey} X_b \} \}) \# #
Snd (Friend a) \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy})\): Crypt (\text{combiK}(X_a, X_b))
(\text{Sign} \text{[Friend a]} \{ \{ \text{HKey} X_a, \text{HKey} X_b \} \}) \# #
Bk (Friend a) \((\text{Friend b, Spy}) \rightarrow (\text{Friend a, Friend a})\): \{ \{ \text{HKey} X_b, Crypt
(\text{combiK}(X_a, X_b)) \((\text{Sign} \text{[Friend b]} \{ \{ \text{HKey} X_b, \text{HKey} X_a \} \}) \) \# #
Rcv (Friend a) \((\text{Friend b, Spy}) \rightarrow (\text{Friend a, Friend a})\): \{ \{ \text{HKey} X_b, Crypt
(\text{combiK}(X_a, X_b)) \((\text{Sign} \text{[Friend b]} \{ \{ \text{HKey} X_b, \text{HKey} X_a \} \}) \) \# #
Snd Spy \((\text{Friend b, Spy}) \rightarrow (\text{Friend a, Spy})\): Crypt (\text{combiK}(X_a, X_b))
(\text{Sign} \text{[Friend b]} \{ \{ \text{HKey} X_b, \text{HKey} X_a \} \}) \# #
Bk (Friend b) \((\text{Friend c, Spy}) \rightarrow (\text{Friend b, Friend b})\): \text{HKey} X_a \# #
Rcv (Friend b) \((\text{Friend c, Spy}) \rightarrow (\text{Friend b, Friend b})\): \text{HKey} X_a \# #
Snd Spy \((\text{Friend c, Spy}) \rightarrow (\text{Friend b, Friend b})\): \text{HKey} X_a \# #
Bk Spy \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b})\): \text{HKey} X_a \# #
Rcv Spy \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b})\): \text{HKey} X_a \} \# #
Snd (Friend a) \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b})\): \text{HKey} X_a \} \# #]
: trace
by force

lemma Attack-a-proj-StS: \[a \# b; X_a \# X_b \] \implies \exists \text{evs} \in \text{StS. (evs \uparrow \text{Friend a}) =}

((
Bk Spy \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy})\): Crypt (\text{combiK}(X_a, X_b))
(\text{Sign} \text{[Friend a]} \{ \{ \text{HKey} X_a, \text{HKey} X_b \} \}) \# #
Rcv Spy \((\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy})\): Crypt (\text{combiK}(X_a, X_b))
(\text{Sign} \text{[Friend a]} \{ \{ \text{HKey} X_a, \text{HKey} X_b \} \}) \# #

(\text{Sign} \text{[Friend a]} \{ \{ \text{HKey} X_a, \text{HKey} X_b \} \}) \# #)
Snd (Friend a) \( \langle \text{Friend a}, \text{Friend a} \rangle \rightarrow (\text{Friend b}, \text{Spy}) \): Crypt (combiK(Xa, Xb)) (Sign[Friend a] [HKey Xa, HKey Xb]) \# 

Blk (Friend a) \( \langle \text{Friend b}, \text{Spy} \rangle \rightarrow (\text{Friend a}, \text{Friend a}) \): [HKey Xb, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xb, HKey Xa])]) \# 

Rev (Friend a) \( \langle \text{Friend b}, \text{Spy} \rangle \rightarrow (\text{Friend a}, \text{Friend a}) \): [HKey Xb, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xb, HKey Xa])]) \# 

Snd Spy \( \langle \text{Friend a}, \text{Friend a} \rangle \rightarrow (\text{Friend b}, \text{Spy}) \): [HKey Xb, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xb, HKey Xa])]) \# 

Blk Spy \( \langle \text{Friend b}, \text{Friend b} \rangle \rightarrow (\text{Friend c}, \text{Spy}) \): [HKey Xa, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xa, HKey Xa])]) \# 

Rev Spy \( \langle \text{Friend b}, \text{Friend b} \rangle \rightarrow (\text{Friend c}, \text{Spy}) \): [HKey Xa, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xa, HKey Xa])]) \# 

Snd (Friend b) \( \langle \text{Friend b}, \text{Friend b} \rangle \rightarrow (\text{Friend c}, \text{Spy}) \): [HKey Xa, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xa, HKey Xa])]) \# 

lemma Attack-finish-a: \( a \neq b \implies \)

Blk Spy \( \langle \text{Friend a}, \text{Friend a} \rangle \rightarrow (\text{Friend b}, \text{Spy}) \): Crypt (combiK(Xa, Xb)) (Sign[Friend a] [HKey Xa, HKey Xb]) \# 

Rev Spy \( \langle \text{Friend a}, \text{Friend a} \rangle \rightarrow (\text{Friend b}, \text{Spy}) \): Crypt (combiK(Xa, Xb)) (Sign[Friend a] [HKey Xa, HKey Xb]) \# 

Snd (Friend a) \( \langle \text{Friend a}, \text{Friend a} \rangle \rightarrow (\text{Friend b}, \text{Spy}) \): Crypt (combiK(Xa, Xb)) (Sign[Friend a] [HKey Xa, HKey Xb]) \# 

Blk (Friend a) \( \langle \text{Friend b}, \text{Spy} \rangle \rightarrow (\text{Friend a}, \text{Friend a}) \): [HKey Xb, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xb, HKey Xa])]) \# 

Rev (Friend b) \( \langle \text{Friend b}, \text{Friend b} \rangle \rightarrow (\text{Friend c}, \text{Spy}) \): [HKey Xa, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xa, HKey Xa])]) \# 

Snd (Friend b) \( \langle \text{Friend b}, \text{Friend b} \rangle \rightarrow (\text{Friend c}, \text{Spy}) \): [HKey Xa, Crypt (combiK(Xa, Xb)) (Sign[Friend b] [HKey Xa, HKey Xa])]) \#
\[\text{Snd Spy} \{(\text{Friend c, Spy}) \rightarrow (\text{Friend b, Friend a}) : H\text{Key Xa} \} \] #
\[\text{Bk Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \} \] #
\[\text{Rcv Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \} \] #
\[\text{Snd} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \} \] # #
**finish Friend a**
b y (simp add: finish-def)

**lemma** Attack-a-Agents: \(a \neq b \implies \text{Friend a} \in \text{Agents} \)

\[
\text{Bk Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy}) : Crypt (\text{combiK(Xa, Xb)})
\]
\[
\text{Sign}[\text{Friend a}][\{ \text{HKey Xa, HKey Xb} \}]) \] #
\[\text{Rcv Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy}) : Crypt (\text{combiK(Xa, Xb)})
\]
\[
\text{Sign}[\text{Friend a}][\{ \text{HKey Xa, HKey Xb} \}]) \] #
\[\text{Snd} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy}) : Crypt (\text{combiK(Xa, Xb)})
\]
\[
\text{Sign}[\text{Friend a}][\{ \text{HKey Xa, HKey Xb} \}]) \] #
\[\text{Bk Spy} \{(\text{Friend b, Spy}) \rightarrow (\text{Friend a, Friend a}) : \{ \text{HKey Xb, Crypt}
\]
\[
\text{combiK(Xa, Xb)}) (\text{Sign}[\text{Friend b}][\{ \text{HKey Xb, HKey Xa} \}]) \] #
\[\text{Rcv Spy} \{(\text{Friend b, Spy}) \rightarrow (\text{Friend a, Friend a}) : \{ \text{HKey Xb, Crypt}
\]
\[
\text{combiK(Xa, Xb)}) (\text{Sign}[\text{Friend b}][\{ \text{HKey Xb, HKey Xa} \}]) \] #
\[\text{Snd} \{(\text{Friend b, Spy}) \rightarrow (\text{Friend a, Friend a}) : \{ \text{HKey Xb, Crypt}
\]
\[
\text{combiK(Xa, Xb)}) (\text{Sign}[\text{Friend b}][\{ \text{HKey Xb, HKey Xa} \}]) \] #
\[\text{Bk Spy} \{(\text{Friend c, Spy}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \} \] #
\[\text{Rcv Spy} \{(\text{Friend c, Spy}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \} \] #
\[\text{Snd Spy} \{(\text{Friend c, Spy}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \] #
\[\text{Bk Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \} \] #
\[\text{Rcv Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \] #
\[\text{Snd} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Friend b}) : H\text{Key Xa} \] # #]

**lemma** Attack-b-Agents: \(a \neq b \implies \text{Friend b} \in \text{Agents} \)

\[
\text{Bk Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy}) : Crypt (\text{combiK(Xa, Xb)})
\]
\[
\text{Sign}[\text{Friend a}][\{ \text{HKey Xa, HKey Xb} \}]) \] #
\[\text{Rcv Spy} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy}) : Crypt (\text{combiK(Xa, Xb)})
\]
\[
\text{Sign}[\text{Friend a}][\{ \text{HKey Xa, HKey Xb} \}]) \] #
\[\text{Snd} \{(\text{Friend a, Friend a}) \rightarrow (\text{Friend b, Spy}) : Crypt (\text{combiK(Xa, Xb)})
\]
\[\text{lemma Attack-consis: } [a \neq b; b \neq c; X_a \neq X_b] \implies\]
\[
\begin{align*}
\text{Blk (Friend a)} \ (\text{Friend b}) &\rightarrow (\text{Friend a}) \ (\text{Friend b}): \text{Crypt (combiK(Xa, Xb))} \\
\text{Sign(Friend a) } [\{ \text{HKey Xa, HKey Xb} \}] &\rightarrow # \\
\text{Rec (Friend b)} \ (\text{Friend a}) &\rightarrow (\text{Friend b}, \text{Spy}): \text{Crypt (combiK(Xa, Xb))} \\
\text{Sign(Friend b) } [\{ \text{HKey Xa, HKey Xb} \}] &\rightarrow # \\
\text{Blk (Friend b)} &\rightarrow (\text{Friend c, Spy}): \text{Crypt (combiK(Xa, Xb))} \\
\text{Sign(Friend b) } [\{ \text{HKey Xa, HKey Xb} \}] &\rightarrow # \\
\text{Rec (Friend b)} &\rightarrow (\text{Friend a, Friend b}, \text{Spy}): \text{Crypt (combiK(Xa, Xb))} \\
\text{Sign(Friend b) } [\{ \text{HKey Xa, HKey Xb} \}] &\rightarrow # \\
\text{Blk (Friend b)} &\rightarrow (\text{Friend a, Friend b}): \text{HKey Xa} \rightarrow # \\
\text{Rec (Friend b)} &\rightarrow (\text{Friend c, Spy}): \text{HKey Xa} \rightarrow # \\
\text{Sign(Friend b) } [\{ \text{HKey Xa, HKey Xb} \}] &\rightarrow # \\
\end{align*}
\]
\[ Snd \ Spy \{(\text{Friend } a, \ Spy) \rightarrow (\text{Friend } b, \text{Friend } b): \text{HKey } Xa \} \] #
\[ Bk \ Spy \{(\text{Friend } a, \text{Friend } a) \rightarrow (\text{Friend } b, \text{Friend } b): \text{HKey } Xa \} \] #
\[ Rcv \ Spy \{(\text{Friend } a, \text{Friend } a) \rightarrow (\text{Friend } b, \text{Friend } b): \text{HKey } Xa \} \] #
\[ Snd \{(\text{Friend } a, \text{Friend } a) \rightarrow (\text{Friend } b, \text{Friend } b): \text{HKey } Xa \} \] # #
\[ \triangle \text{StS} \]
apply (simp add: consis-def)
apply auto
apply force
apply (rule bezI)
prefer 2
apply (rule StS.Msg3)
apply (rule-tac \( A = \text{Friend } a \) \text{ and } B = \text{Friend } b \) in StS.Rcv)
apply (rule StS.Msg2)
apply (rule-tac \( A = \text{Friend } b \) \text{ and } B = \text{Friend } a \) in StS.Rcv)
apply (rule StS.Msg1)
apply (rule StS.Nil)
apply force
apply force
deferr 1
apply force
apply force
apply force
apply force
apply force
apply force
prefer 2
apply (rule bezI)
prefer 2
apply (rule StS.Msg3)
apply (rule-tac \( A = \text{Friend } a \) \text{ and } B = \text{Friend } b \) in StS.Rcv)
apply (rule StS.Msg2)
apply (rule-tac \( A = \text{Friend } b \) \text{ and } B = \text{Friend } a \) in StS.Rcv)
apply (rule StS.Msg1)
apply (rule StS.Nil)
deferr 1
apply force
deferr 1
apply force
apply force
apply force
apply force
prefer 2
apply force
apply (rule bezI)
prefer 2
apply (rule Sts_Msg2)
apply (rule-tac A = Friend b and B = Friend c in Sts.Rev)
apply (rule Sts_Msg1)
apply (rule Sts.Nil)
def er 1
apply force
def er 1
by force+

theorem Attack-a-successful: \([ a \neq b \land b \neq c \land Xa \neq Xb] \implies\)

( Bk Spy ((Friend a, Friend a) \rightarrow (Friend b, Spy)) : Crypt (combiK(Xa, Xb))
(Sign[Friend a] [\{ HKey Xa, HKey Xb \}] ) \# #
Rev Spy ((Friend a, Friend a) \rightarrow (Friend b, Spy)) : Crypt (combiK(Xa, Xb))
(Sign[Friend a] [\{ HKey Xa, HKey Xb \}] ) \# #
Snd (Friend a) ((Friend a, Friend a) \rightarrow (Friend b, Spy)) : Crypt (combiK(Xa, Xb))
(Sign[Friend a] [\{ HKey Xa, HKey Xb \}] ) \# #
Bk (Friend a) ((Friend b, Spy) \rightarrow (Friend a, Friend a) : \{ HKey Xb, Crypt (combiK(Xa, Xb))
(Sign[Friend b] [\{ HKey Xb, HKey Xa \}] )\# #
Rev (Friend a) ((Friend b, Spy) \rightarrow (Friend a, Friend a) : \{ HKey Xb, Crypt (combiK(Xa, Xb))
(Sign[Friend b] [\{ HKey Xb, HKey Xa \}] )\# #
Snd Spy (Friend b) ((Friend b, Spy) \rightarrow (Friend a, Friend a) : \{ HKey Xb, Crypt (combiK(Xa, Xb))
(Sign[Friend b] [\{ HKey Xb, HKey Xa \}] )\# #
Bk Spy (Friend c, Spy) \rightarrow (Friend b, Friend b) : HKey Xa \# #
Rev Spy (Friend c, Spy) \rightarrow (Friend b, Friend b) : HKey Xa \# #
Snd (Friend a) (Friend a, Friend a) \rightarrow (Friend b, Friend b) : HKey Xa \# #
) \n (Friend a

apply (simp add: attack-def del: HKey-def Sign-def)
apply (rule com\#)
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
d | SState nat
 | RState nat
 | Abort
 | Accept
 | Ver nat nat bool
 | Ready

axioms

\[ \text{hdnil-not-action [iff]}: ((x::\text{action}) = \text{hd \{\}}) = \text{False} \]

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
    Revi [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

constsdefs
    chkAgent :: agent ⇒ bool
    chkAgent ≡ λa. a ν bad

lemma chkServer: chkAgent Server
lemma chkSpy: ~chkAgent Spy
lemma chkFriend [simp]: \( \text{chkAgent} \ (\text{Friend} \ a) \equiv (\neg (\text{Friend} \ a) \in \text{bad}) \)
lemma Ver-True: \( P \implies \text{Ver} \ n \ m \ P = \text{Ver} \ n \ m \ True \)
lemma Ver-false: \( \neg P \implies \text{Ver} \ n \ m \ P = \text{Ver} \ n \ m \ False \)
lemma notbad-vertrue: \( A \not \in \text{bad} \implies \text{Ver} \ n \ m \ (\text{chkAgent} \ A) = \text{Ver} \ n \ m \ True \)

lemma \( \forall \text{acs} \in \text{lttrace} \ A. \ \text{acs} = \text{a}\#(\text{Ver} \ 1 \ 1 \ (\text{chkAgent} \ Spy))\#(\text{acts}\cdot\text{action list}) \)
\( \Rightarrow a = \text{Abort} \)
apply (rule ballE)
apply (rule lttrace.cases)
by auto

lemma \( \exists \text{acts} \in \text{lttrace} \ A. \ \text{Accept}\#\text{acts} \in \text{lttrace} \ A \)
apply (rule bexI)
prefer 2
apply (rule-tac b = \text{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**

```latex
constdefs
  arnconv :: bool => bool (is (Re) [50] 50)
  [Re]P = ~(<Re>(~P))
  lncnv :: bool => bool (is (Le) [50] 50)
  [Le]P = ~(<Le>(~P))

theorem shcnv [intro]: [i,j]:fwd; <k,j>:fwd; <k,i>:P] \implies <i,j>:<Le>P

theorem srnconv [intro]: [i,j]:fwd; <i,k>:fwd; <i,i>:P] \implies <i,j>:<Re>P

theorem shcnvE [elim]:
  assumes 1: [i,j]: [Le]P
  and 2: [i,j]:fwd
  and 3: [k,j]:fwd; <k,j>:P] \implies <m,n>:R
  shows <m,n>:R

theorem srnconvE [elim]:
  assumes 1: [i,j]: [Re]P
  and 2: [i,j]:fwd
  and 3: [k,i]:fwd; <k,i>:P] \implies <m,n>:R
  shows <m,n>:R

theorem lncnvI [intro]:
  assumes 1: [k,j]:fwd] \implies [k,j]:P
  and 2: [i,j]:fwd
  shows [i,j]:[Le]P

theorem arnconvI [intro]:
  assumes 1: [k,j]:fwd] \implies [i,k]:P
  and 2: [i,j]:fwd
  shows [i,j]:[Re]P

theorem lncnvD [dest]: [i,j]:[Le]P; <k,j]:fwd; <i,j]:fwd] \implies <k,j]:P
```
theorem arnconvD [dest]: \[[\langle i,j \rangle : \text{Rec}\rangle P; \langle i,k \rangle : \text{fwd}; \langle i, j \rangle : \text{fwd}\] \implies \langle i, k \rangle : P\]

theorem arnconvE [elim];
  assumes 1: \langle i,j \rangle : \text{Rec}\rangle P
  and  2: \langle k,j \rangle : \langle i,n \rangle : R
  and  3: \langle k,j \rangle : \text{fwd}
  and  4: \langle i,j \rangle : \text{fwd}
  shows \langle i,n \rangle : R

theorem arnconvE [elim];
  assumes 1: \langle i,j \rangle : \text{Rec}\rangle P
  and  2: \langle i,k \rangle : \langle i,n \rangle : R
  and  3: \langle i,k \rangle : \text{fwd}
  and  4: \langle i,j \rangle : \text{fwd}
  shows \langle i,n \rangle : R
end

theory Propersub = ConverseSub:

Proper Subintervals

constdefs
  spsub :: bool \Rightarrow bool  (\langle PS \rangle : [50] 50)
  \langle PS \rangle P \equiv \langle PS \rangle \langle 0 \rangle \langle P \rangle \langle 0 \rangle \langle len \rangle
  apsub :: bool \Rightarrow bool  (\langle PS \rangle : [50] 50)
  \langle PS \rangle P \equiv \neg (\langle PS \rangle \langle \neg P \rangle)
  srsub :: bool \Rightarrow bool  (\langle RPS \rangle : [50] 50)
  \langle RPS \rangle P \equiv \langle RPS \rangle \langle P \rangle \langle 0 \rangle \langle len \rangle
  arsub :: bool \Rightarrow bool  (\langle RPS \rangle : [50] 50)
  \langle RPS \rangle P \equiv \neg (\langle RPS \rangle \langle \neg P \rangle)
  slsub :: bool \Rightarrow bool  (\langle LPS \rangle : [50] 50)
  \langle LPS \rangle P \equiv \langle LPS \rangle \langle P \rangle \langle 0 \rangle \langle len \rangle
  alsub :: bool \Rightarrow bool  (\langle LPS \rangle : [50] 50)
  \langle LPS \rangle P \equiv \neg (\langle LPS \rangle \langle \neg P \rangle)

lemma zless-imp-fwd: \langle i,j \rangle : 0 \langle len \rangle \Rightarrow \langle i,j \rangle : \text{fwd}
lemma zlessen-zelen-imp-zelen: \langle i,j \rangle : 0 \langle len \rangle \Rightarrow \langle i,j \rangle : 0 \langle len \rangle \Rightarrow \langle i,j \rangle : 0 \langle len \rangle
lemma zelen-zlessen-imp-zelen: \langle k,j \rangle : 0 \langle len \rangle \Rightarrow \langle i,j \rangle : 0 \langle len \rangle \Rightarrow \langle i,j \rangle : 0 \langle len \rangle
lemma ssub-zlessen-imp-zelen: \langle i,j \rangle : \text{fwd}; \langle i,j \rangle : \langle S \rangle \langle 0 \rangle \langle len \rangle \Rightarrow \langle i,j \rangle : \langle S \rangle \langle 0 \rangle \langle len \rangle
\[ j > 0 < \text{len} \]

**lemma** \(_{\text{fwd-zlessen-imp-zlessen}}\) \([<i, k>: \text{fwd}; <k, j>: 0 < \text{len}] \implies <i, j>: 0 < \text{len}\)

**lemma** \(_{\text{zlessen-fwd-imp-zlessen}}\) \([<k, j>: \text{fwd}; <i, k>: 0 < \text{len}] \implies <i, j>: 0 < \text{len}\)

**theorem** \(_{\text{ulspsubI \ [(intro)]}}\) \([<i, k>: 0 < \text{len}; <k, l>: P; <k, l>: \text{fwd}; <l, j>: 0 < \text{len}] \implies <i, j>: <PS>P\)

**theorem** \(_{\text{ulspsubE \ [(elim)]}}\)
**assumes** 1: \(<i, j>: <PS>P\)
and 2: \(!! k. [<i, k>: 0 < \text{len}; <k, l>: P; <k, l>: \text{fwd}; <l, j>: 0 < \text{len}] \implies <m, n>: R\)
**shows** \(<m, n>: R\)

**theorem** \(_{\text{ulspsubI \ [(intro)]}}\)
**assumes** 1: \(!! k. l. [<i, k>: 0 < \text{len}; <k, l>: \text{fwd}; <l, j>: 0 < \text{len}] \implies <k, l>: P\)
**shows** \(<i, j>: <PS>P\)

**theorem** \(_{\text{ulspsubD \ [(dest)]}}\)
**assumes** \(<i, j>: <PS>P; <i, k>: 0 < \text{len}; <k, l>: \text{fwd}; <l, j>: 0 < \text{len}] \implies <k, l>: P\)

**theorem** \(_{\text{ulspsubE \ [(elim)]}}\)
**assumes** 1: \(<i, j>: <PS>P\)
and 2: \(<k, l>: P \implies <m, n>: R\)
and 3: \(<i, k>: 0 < \text{len}\)
and 4: \(<k, l>: \text{fwd}\)
and 5: \(<L, j>: 0 < \text{len}\)
**shows** \(<m, n>: R\)

**theorem** \(_{\text{ulspsubI \ [(intro)]}}\)
**assumes** \(<i, k>: \text{fwd}; <k, l>: P; <k, l>: \text{fwd}; <l, j>: 0 < \text{len}] \implies <i, j>: <RPS>P\)

**theorem** \(_{\text{ulspsubE \ [(elim)]}}\)
**assumes** 1: \(<i, j>: <RPS>P\)
and 2: \(!! k. l. [<i, k>: \text{fwd}; <k, l>: P; <k, l>: \text{fwd}; <l, j>: 0 < \text{len}] \implies <m, n>: R\)
**shows** \(<m, n>: R\)

**theorem** \(_{\text{ulspsubI \ [(intro)]}}\)
\begin{enumerate}
\item \textbf{assumes} 1: \texttt{!! k l. \langle i, k \rangle: \textit{fwd}; \langle k, l \rangle: \textit{fwd}; \langle l, j \rangle: \textit{0 < len} \rangle \implies \langle k, l \rangle: \textit{P}}
\item \textbf{shows} \langle i, j \rangle: [\textit{RPS}] P
\end{enumerate}

\begin{enumerate}
\item \textbf{theorem} \texttt{darpsubD [dest]}: \[ \langle i, j \rangle: [\textit{RPS}] P; \langle i, k \rangle: \textit{fwd}; \langle k, l \rangle: \textit{fwd}; \langle l, j \rangle: \textit{0 < len} \]\implies \langle k, l \rangle: \textit{P}
\end{enumerate}

\begin{enumerate}
\item \textbf{theorem} \texttt{darpsubE [elim]}:
\item \textbf{assumes} 1: \langle i, j \rangle: [\textit{RPS}] P
\item \textbf{and} 2: \texttt{\langle k, l \rangle: \textit{P} \implies \langle m, n \rangle: \textit{R}}
\item \textbf{and} 3: \langle i, k \rangle: \textit{fwd}
\item \textbf{and} 4: \langle k, l \rangle: \textit{fwd}
\item \textbf{and} 5: \langle l, j \rangle: \textit{0 < len}
\item \textbf{shows} \langle m, n \rangle: \textit{R}
\end{enumerate}

\begin{enumerate}
\item \textbf{theorem} \texttt{lsrpsub1 [intro]}: \[ \langle i, k \rangle: \textit{0 < len}; \langle k, l \rangle: \textit{P}; \langle k, l \rangle: \textit{fwd}; \langle l, j \rangle: \textit{fwd} \]\implies \langle i, j \rangle: \textit{LPS} P
\end{enumerate}

\begin{enumerate}
\item \textbf{theorem} \texttt{lsrpsubE [elim]}:
\item \textbf{assumes} 1: \langle i, j \rangle: \textit{LPS} P
\item \textbf{and} 2: \texttt{\langle i, k \rangle: 0 < len; \langle k, l \rangle: P; \langle k, l \rangle: \textit{fwd}; \langle l, j \rangle: \textit{fwd} \rangle \implies \langle m, n \rangle: \textit{R}}
\item \textbf{shows} \langle m, n \rangle: \textit{R}
\end{enumerate}

\begin{enumerate}
\item \textbf{theorem} \texttt{lsrpsubD [dest]}: \[ \langle i, j \rangle: [\textit{LPS}] P; \langle i, k \rangle: \textit{0 < len}; \langle k, l \rangle: \textit{fwd}; \langle l, j \rangle: \textit{fwd} \]\implies \langle k, l \rangle: \textit{P}
\end{enumerate}

\begin{enumerate}
\item \textbf{theorem} \texttt{lsrpsubE [elim]}:
\item \textbf{assumes} 1: \langle i, j \rangle: \textit{[LPS]} P
\item \textbf{and} 2: \texttt{\langle i, k \rangle: \textit{0 < len}; \langle k, l \rangle: P \implies \langle m, n \rangle: \textit{R}}
\item \textbf{and} 3: \langle i, k \rangle: \textit{0 < len}
\item \textbf{and} 4: \langle k, l \rangle: \textit{fwd}
\item \textbf{and} 5: \langle l, j \rangle: \textit{fwd}
\item \textbf{shows} \langle m, n \rangle: \textit{R}
\end{enumerate}

\begin{enumerate}
\item \textbf{lemma} \texttt{lssl-lessl-add}: \[ \langle i, j \rangle: \textit{RI s}; \textit{RI t}; \langle i, k \rangle: \textit{s \leq len}; \langle k, j \rangle: \textit{t < len} \]\implies \langle i, j \rangle: \textit{s + t < len}
\end{enumerate}
lemma lesslen-lelen-add: \[\begin{align*} & | \ \text{RI} \ s; \ \text{RI} \ t; \ \langle i,k \rangle : s \leq \text{len}; \ \langle k,j \rangle : t \leq \text{len} \ | \ \Rightarrow \\
& \langle i,j \rangle : s + t \leq \text{len} \end{align*}\]

lemma zelen-zlessen-imp-zlesslen: \[\begin{align*} & | \ \langle i,k \rangle : 0 \leq \text{len}; \ \langle k,j \rangle : 0 \leq \text{len} \ | \ \Rightarrow \\
& \langle i,j \rangle : 0 \leq \text{len} \end{align*}\]

lemma zlessen-zelen-imp-zlesslen: \[\begin{align*} & | \ \langle i,k \rangle : 0 \leq \text{len}; \ \langle k,j \rangle : 0 \leq \text{len} \ | \ \Rightarrow \\
& \langle i,j \rangle : 0 \leq \text{len} \end{align*}\]

lemma lelen-zlessen-imp-lesslen: \[\begin{align*} & | \ \langle i,k \rangle : y \leq \text{len}; \ \langle k,j \rangle : 0 \leq \text{len}; \ \text{RI} \ y \ | \ \Rightarrow \\
& \langle i,j \rangle : y \leq \text{len} \end{align*}\]

lemma lesslen-zelen-imp-lesslen: \[\begin{align*} & | \ \langle i,k \rangle : y \leq \text{len}; \ \langle k,j \rangle : 0 \leq \text{len}; \ \text{RI} \ y \ | \ \Rightarrow \\
& \langle i,j \rangle : y \leq \text{len} \end{align*}\]

lemma zelen-lessen-imp-lesslen: \[\begin{align*} & | \ \langle i,k \rangle : 0 \leq \text{len}; \ \langle k,j \rangle : y \leq \text{len}; \ \text{RI} \ y \ | \ \Rightarrow \\
& \langle i,j \rangle : y \leq \text{len} \end{align*}\]

lemma zlessen-lelen-imp-lesslen: \[\begin{align*} & | \ \langle i,k \rangle : 0 \leq \text{len}; \ \langle k,j \rangle : y \leq \text{len}; \ \text{RI} \ y \ | \ \Rightarrow \\
& \langle i,j \rangle : y \leq \text{len} \end{align*}\]

lemma lesslen-lelen-imp-lesslen: \[\begin{align*} & | \ \langle i,k \rangle : 0 \leq \text{len}; \ \langle k,j \rangle : y \leq \text{len}; \ \text{RI} \ y \ | \ \Rightarrow \\
& \langle i,j \rangle : y \leq \text{len} \end{align*}\]

lemma ssu-subless-imp-lesslen: \[\begin{align*} & | \ \text{RI} \ z; \ \langle i,j \rangle : < S > z \leq \text{len} \ | \ \Rightarrow \\
& \langle i,j \rangle : x \leq \text{len} \end{align*}\]

lemma ssu-sublen-imp-lesslen: \[\begin{align*} & | \ \text{RI} \ z; \ \langle i,j \rangle : < S > x \leq \text{len} \ | \ \Rightarrow \\
& \langle i,j \rangle : x \leq \text{len} \end{align*}\]

end
A.6.2 Timed Properties of Lists

theory TimedLists = Propersub:

axioms
RLemp-list [HOL intro!]: RI []
RLconsI [HOL intro!]: []RI x; RI xs] " RI (x#xs)
RLconsI1 [HOL dest!]: (RI (x#xs)) " RI x
RLconsE2 [HOL dest!]: (RI (x#xs)) " RI x

Equality for Lists

axioms
RLegal [HOL intro!]: [RI x; RI y] " RI x = (y::'a list)
RLegalI: (RI x = (y::'a list)) " RI x
RLegalE2: (RI x = (y::'a list)) " RI y
CFequ [HOL intro!]: CF (x = (y::'a list))

theorem leqtrans: [<i,j>:xs=ys; <i,j>:ys=zs] " <i,j>:zs=(xs::'a list)

theorem leqrefl: <i,j>: evs = (evs::'a list)
theorem leqsub: [] CF (P x); <i,j>:s=t; <i,j>:(P s) [] " <i,j>:(P (t::'a list))
theorem leqsubRI: [] RI s; RI t; <i,j>:s=t; <i,j>:(P s) [] " <i,j>:(P (t::'a list))

Precedes-or-Equals

consts
defs
prefix :: ['a list, 'a list] => bool (- ≤ - [60,60] 60)
x ≤ y = (∃z. x@z = y)

theorem presub: evs ≤ evs
theorem presubtrans: [evs ≤ evs`; evs` ≤ evs`] " evs ≤ evs``
theorem presubeq: [evs ≤ evs`; evs` ≤ evs] " evs = evs`
theorem presubeq1: evs = evs` " evs` ≤ evs
theorem presubeq2: evs = evs` " evs` ≤ evs
lemma pre-equ-pre: [evs ≤ evs`; evs = evs"] " evs`` ≤ evs`
lemma pre-equ-pre2: [evs ≤ evs`; evs` = evs"] " evs` ≤ evs``

axioms
RlpredI [HOL intro!]: [RI x; RI y] " RI x ≤ (y::'a list)
RlpredE1: (RI x ≤ (y::'a list)) " RI x
RlpredE2: (RI x ≤ (y::'a list)) " RI y
A.6 Timed Theory of Protocols

\[ CF \text{pre } [HOL \text{ intro}]: \quad CF \ (x \leq (y::'a \text{ list})) \]

\textbf{Theorems}

\textbf{theorem} \textit{lpref}: \( <i,j>: evs \leq evs' \)

\textbf{theorem} \textit{lpref\_trans}: \( [<i,j>: evs \leq evs'; \quad <i,j>: evs' \leq evs''] \Rightarrow <i,j>: evs \leq evs'' \)

\textbf{theorem} \textit{lpref\_anti}: \( [<i,j>: evs \leq evs'; \quad <i,j>: evs' \leq evs] \Rightarrow <i,j>: evs = evs' \)

\textbf{theorem} \textit{lpref\_weaken1}: \( <i,j>: evs = evs' \Rightarrow <i,j>: evs \leq evs' \)

\textbf{theorem} \textit{lpref\_weaken2}: \( <i,j>: evs = evs' \Rightarrow <i,j>: evs' \leq evs \)

\textbf{theorem} \textit{lpref\_sym}: \( <i,j>: evs = evs' \Rightarrow <i,j>: evs' = (evs::'a \text{ list}) \)

\textbf{Miscellaneous Lemmas}

\textbf{lemma} \textit{lsubstr}: \( [<i,j>: s=t; \quad CF \ P(x); \quad <i,j>: P(s)] \Rightarrow <i,j>: P(t::'a \text{ list}) \)

\textbf{lemma} \textit{lsubst}: \( [\langle i,j \rangle : t=s; \quad CF \ P(x); \quad <i,j>: P(s)] \Rightarrow <i,j>: P(t::'a \text{ list}) \)

\textbf{lemma} \textit{lpref\_equ\_pre1}: \( [<i,j>: evs \leq evs'; \quad <i,j>: evs = evs''] \Rightarrow <i,j>: evs' \leq evs'' \)

\textbf{lemma} \textit{lpref\_equ\_pre2}: \( [<i,j>: evs \leq evs'; \quad <i,j>: evs' = evs''] \Rightarrow <i,j>: evs \leq evs'' \)

\textbf{lemma} \textit{RI\ a; RI\ b]} \Rightarrow RI \ [a, b]

\textbf{lemma} \textit{RI\ [a, b]} \Rightarrow RI \ b

end
A.6.3 Timed Properties of Global Traces

theory TimedTrace = Schematic + TimedLists:

axioms

R1eq-evt1 [intro]: || RI s; RI t || ---> (RI (s = (t::event)))
R1eq-evtE1: (RI (s = (t::event))) ---> RI s
R1eq-evtE2: (RI (s = (t::event))) ---> RI t

CFeq-evt [intro]: CF (e1 = (e2::event))

Flexible Constant Tr

consts

Tr :: event list

axioms

Tr-monot1: ||<i,j>: Tr = evs; <i,j>:fwd; RI evs || ---> <i,j>:[Le](Tr = evs)
Tr-monot2: ||<i,j>: Tr = evs; <i,j>:fwd; RI evs || ---> <i,j>:[R](evs ≤ Tr)
Tr-monot3: ||<i,j>: Tr = evs; <i,j>:fwd; RI evs || ---> <i,j>:[Le][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 tiffI)
apply (rule Tr-ex)
apply (erule lexE)
apply (rule lexIRI)
prefer 2
apply (rule konji)
apply assumption
apply (erule lsubstr)
prefer 4
apply (erule lexE)
apply (rule konji)
apply (rule lsubsl)
by (assumption | rule Cpre)+

theorem Tr-Trpre-expand: [RI evs] ---> <i,j>:Tr ≤ evs ---> (∃ x. Tr = x ∧ x ≤ evs)
apply (rule lefl)
apply (rule Tr-ex)
apply (rule lexE)
apply (rule lexIRI)
prefer 2
apply (rule conjI)
apply assumption
apply (rotate-tac 3)
apply (rule lsubstr)
apply (rule CFpre)
apply assumption+
apply (rule lexE)
apply (rule lexRJE)
apply (rule lsubsl)
by (assumption | rule CFpre)+

theorem Tr-mono4: \([i,j]: Tr = evs; <i,j>:fwd; RI evs] \implies <i,j>: [S](Tr \leq evs)
apply (rule Tr-mono3)
apply assumption+
apply (rule lasubI)
apply (rule anconE)
apply (rule anE)
by assumption+

theorem Tr-mono5: \([i,j]: Tr = evs; <i,j>:fwd; RI evs] \implies <i,j>: [PS]Tr \leq evs
apply (rule Tr-mono3)
apply assumption+
apply (rule lasubI)
apply (rule anconE)
apply (rule anE)
by (drule lelessen-imp-fwd | assumption)+

lemma Tr-mono-pre1: \([RI evs; <i,j>: evs \leq Tr; <i,j>:fwd] \implies <i,j>: [Lc]evs \leq Tr

theorem Tr-mono6: \([<i,j>: Tr = evs; <i,j>:fwd; RI evs] \implies <i,j>: [R][Lc]evs \leq Tr
apply (rule Tr-mono2)
apply assumption+
apply (rule anI)
apply assumption
apply (rule allocv)
apply (drule arnE)
prefer 3
defer 1
apply assumption+
apply (drule Tr-mono-pre1)
apply assumption+
apply (rule alncomE)
by assumption+

Predicates Concerning Tr

consts
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-IC: <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)
A.6 Timed Theory of Protocols

\[ Tr-O-Rcv-imp-C-Rcv: \left[<i,j>:Tr-occurs\ (Rcv\ A\ P)\right] \Rightarrow <i,j>:Tr-causes\ A\ (Rcv\ A\ P) \]
\[ Tr-O-Blk-imp-C-Blk: \left[<i,j>:Tr-occurs\ (Blk\ A\ P)\right] \Rightarrow <i,j>:Tr-causes\ A\ (Blk\ A\ P) \]

Lemma and Theorems Regarding Predicates

Axioms

\[ Tr-CS-CS-eqv: \left[<i,k>:Tr-causes\ A\ e; <m,n>: Tr-causes\ A\ e; <k,j>: Tr-stable\ A; <n,j>: Tr-stable\ A\right] \Rightarrow <n,k>: len = 0 \]
\[ Tr-E-O-fwd: \left[<i,k>: Tr-empty; <l,j>: Tr-occurs\ e\right] \Rightarrow <k,l>: fwd \]
\[ Tr-overlap1: \left[<i,j>: Tr-throughout\ A\ evs; <k,j>: Tr-throughout\ A\ evs; RI\ evs; RI\ evs\right] \Rightarrow <i,j>: evs = evs \]
\[ Tr-overlap2: \left[<i,j>: Tr-throughout\ A\ evs; <i,k>: Tr-throughout\ A\ evs; RI\ evs; RI\ evs\right] \Rightarrow <i,j>: evs = evs \]
\[ Tr-connected: \left[<i,k>: [PS](Tr\ A = evs); <k,l>: len = 0; <k,l>: Tr\ A = evs; <l,j>: [PS](Tr\ A = evs)\right] \Rightarrow <i,j>: [PS](Tr\ A = evs) \]

Lemma \[ Tr-C-imp-O: \left[<i,j>:Tr-causes\ A\ P; RI\ A\right] \Rightarrow <i,j>: Tr-occurs\ P \]

Lemma \[ Tr-S-imp-lesseq: <i,j>: Tr-stable\ A \Rightarrow <i,j>: 0 < len \]

Lemma \[ pos-pos-inv-false: \left[0 < (x::real); 0 < y; x = -y\right] \Rightarrow False \]

Lemma \[ bwdless-fwdless-false: \left[<i,j>: 0 < len; <j,i>: 0 < len\right] \Rightarrow <i,j>: False \]

Lemma \[ Tr-silent-imp-S: \left[<i,j>: 0 < len; <i,j>: Tr-stable\ A\ |; Tr-quiet\ A\ |; Tr-stable\ A\right] \Rightarrow <i,j>: Tr-stable\ A \]

apply (crude utchopsubE)+
apply (simp only: Tr-stable-def)
apply (frude Tr-silence)
apply ((crude lexE)+, (crude |E|)+)+
apply (crude shrE)
apply assumption
apply (crude smnE)
apply assumption
apply (frude Tr-overlap1)
apply (rotate-tac -3)
apply assumption+
apply (frude lsubRI)
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 (rule Tr-overlap2)
prefer 3
apply (rotate-tac 1)
apply assumption+
apply (rule lsubstRI)
prefer 2
apply assumption+
apply (unfold Tr-throughout-def)
apply (rule konjE)+
apply (rule lexIRI)
prefer 2
apply (rule konjI, assumption)
apply (rule Tr-connected)
defer 1
apply assumption+
done

done
A.6.4 Timed Properties of Local Traces

theory TimedLTrace = LTrace + TimedLists:

axioms
RIeq-actI [intro]: || RI s; RI t || ==> (RI (s::(t::action)))
RIeq-actE1: (RI (s=(t::action))) ==> RI s
RIeq-actE2: (RI (s=(t::action))) ==> RI t
CFequ-act [intro]: CF (e1 = (e2::action))

RIsimr [intro]: [RI P] ==> RI (Ver P)

Flexible Constant ITr

consts
ITr :: agent => action list

axioms
ITr-mono1: [<i,j>: ITr A = acs; <i,j>:fwd; RI acs] ==> <i,j>: [Le](ITr A = acs)
ITr-mono2: [<i,j>: ITr A = acs; <i,j>:fwd; RI acs] ==> <i,j>: [R](acs <= ITr A)
ITr-mono3: [<i,j>: ITr A = acs; <i,j>:fwd; RI acs] ==> <i,j>: [Le][L](ITr A <= acs)

theorem ITr-ex:
assumes 1: <i,j>:EX acs. ITr A = acs ==> <i,j>:P
shows <i,j>:P

theorem ITr-preITr-expand: [RI acs] ==> <i,j>:acs <= ITr A <-> (exists x. ITr A = x & acs <= x)
apply (rule iffI)
apply (rule ITr-ex)
apply (erule lexE)
apply (rule lexIRI)
prefer 2
apply (rule leconfl)
apply assumption
apply (erule lsubtr)
prefer 4
apply (erule lexE)
apply (erule leconf)
apply (rule lsubst)
by (assumption | rule CFpre)+

**Theorem** \( ITr-ITrpre\)-expand: \([RI\ acs] \implies <i,j>:ITr\ A \preceq acs \iff (\exists z. ITr\ A = z \land z \preceq acs)\)

apply (rule \( \text{iff} \))
apply (rule \( ITr-ex \))
apply (rule \( \text{le} x E \))
apply (rule \( \text{le} x RI \))
pref 2
apply (rule \( \text{konjI} \))
apply assumption
apply (rotate-tac 3)
apply (rule \( \text{lsubstr} \))
apply (rule \( \text{CFpre} \))
apply assumption+
apply (rule \( \text{le} x E \))
apply (rule \( \text{konjE} \))
apply (rule \( \text{lsubst} \))
by (assumption | rule \( \text{CFpre} \))+

**Theorem** \( ITr-monod\): \(<i,j>:ITr\ A = acs; <i,j>:fwd; RI\ acs] \implies <i,j>:\ [SI](ITr\ A \preceq acs)\)

apply (rule \( ITr-monod3 \))
apply assumption+
apply (rule \( \text{dasubI} \))
apply (rule \( \text{ancomwE} \))
apply (rule \( \text{anE} \))
by assumption+

**Theorem** \( ITr-monos\): \(<i,j>:ITr\ A = acs; <i,j>:fwd; RI\ acs] \implies <i,j>:\ [PS]ITr\ A \preceq acs\)

apply (rule \( ITr-monod3 \))
apply assumption+
apply (rule \( \text{dasubI} \))
apply (rule \( \text{ancomwE} \))
apply (rule \( \text{anE} \))
by (drule \( \text{lesslen-imp-fwd} \ | \text{assumption} \)) +

**Lemma** \( ITr-monopret\): \([RI\ acs; <i,j>: acs \preceq ITr\ A; <i,j>:fwd]\] \implies <i,j>:\ [Le]acs \preceq ITr\ A\)

**Theorem** \( ITr-monos\): \(<i,j>:ITr\ A = acs; <i,j>:fwd; RI\ acs] \implies <i,j>:\ [R][Le]acs \preceq ITr\ A\)
apply (rule ITr-mono2)
apply assumption+
apply (rule arn1)
apply assumption
apply (rule alnconvl)
apply (rule arnE)
prefer 3
defer 1
apply assumption+
apply (rule ITr-mono-pre1)
apply assumption+
apply (rule alnconvE)
by assumption+

Predicates Concerning ITr

constdefs
ITr-empty :: agent ⇒ bool
ITr-empty A ≡ ITr A = []

ITr-throughout :: agent ⇒ (action list ⇒ bool)
ITr-throughout A acs ≡ 0 < len ∧ (∃PS)(ITr A = acs)

ITr-performs :: agent ⇒ (action ⇒ bool)
ITr-performs A a ≡ len = 0 ∧ (∃acs. ITr A = a#acs ∧ <L>(ITr-throughout A acs))

ITr-idles :: agent ⇒ bool
ITr-idles A ≡ len = 0 ∧ (∃a. ITr-performs A a)

ITr-stable :: agent ⇒ bool
ITr-stable A ≡ ∃acs. ITr-throughout A acs

axioms

ITr-IC:  <i,j>:<L>ITr-const(A)
ITr-rC:  <i,j>:<R>ITr-const(A)
ITr-idling: <i,j>:ITr-idles A ⇔ <i,j>: len = 0 ∧ (∃acs. ITr A = acs ∧ <L>(ITr-throughout A acs) ∧ <R>(ITr-throughout A acs))
Lemma and Theorems Regarding Predicates

Axioms

\[ \text{ITr-PS-PS-den: } [<i,k>: \text{ITr-performs A a}; <m,n>: \text{ITr-performs A a}]; <k,j>: \text{ITr-stable A}; <n,j>: \text{ITr-stable A}] \implies <n,k>: \text{len}=0 \]

\[ \text{ITr-SP-SP-den: } [<k,l>: \text{ITr-performs A a}; <m,n>: \text{ITr-performs A a}]; <i,k>: \text{ITr-stable A}; <i,m>: \text{ITr-stable A}] \implies <n,k>: \text{len}=0 \]

\[ \text{ITr-P-O-fud: } [<i,k>: \text{ITr-empty A}; <l,j>: \text{ITr-performs A a}] \implies <k,l>: \text{fud} \]

\[ \text{ITr-overlap1: } [<i,j>: \text{ITr-throughout A acs}; <k,j>: \text{ITr-throughout A acs}]; RI acs; RI acs]\]

\[ \text{ITr-overlap2: } [<i,j>: \text{ITr-throughout A acs}; <i,k>: \text{ITr-throughout A acs}]; RI acs; RI acs]\]

\[ \text{ITr-connected: } [<i,k>: [PS](\text{ITr A = acs}); <k,l>: \text{len} = 0]; <k,l>: \text{ITr A} = \text{acs}; <l,j>: [PS](\text{ITr A = acs})] \implies <i,j>: [PS](\text{ITr A = acs}) \]

Lemma \text{ITr-S-imp-dlessen: } <i,j>: \text{ITr-stable A} \implies <i,j>: 0 < \text{len}

Lemma \text{pos-pos-imp-true: } [0 < (z::real); 0 < y; x = -y] \implies \text{True}

Lemma \text{bwdless-bwdless-true: } [i,j]: 0 < \text{len}; <j,i>: 0 < \text{len}] \implies <i,j>: \text{False}

Lemma \text{ITr-idling-imp-S: } [<i,j>: 0 < \text{len}; <i,j>: \text{ITr-stable A} ]; 0 < \text{len} \implies <i,j>: \text{ITr-stable A}

apply \text{(crude inchopsubE)+}
apply \text{(simp only: IT} \text{r-stable-def })
apply \text{(frude IT} \text{r-idling })
apply \text{((crude lexE)+, (crude iconjE)+)+}
apply \text{(crude shE })
apply \text{assumption }
apply \text{(crude srnE })
apply \text{assumption }
apply \text{(frude IT} \text{r-overlap1 })
apply \text{(rotate-tac -3)}
apply \text{assumption+}
apply \text{(frude lsubsRF )}
apply \text{(rotate-tac 9)}
apply \text{assumption }
apply \text{assumption }
apply \text{(rotate-tac 6)}
apply assumption
apply (thin-tac <i,k>:acs=acs b)
apply (rotate-tac 8)
apply (frule ITr-overlap2)
prefer 3
apply (rotate-tac 1)
apply assumption+
apply (frule Isubsti)
prefer 2
apply assumption+
apply (unfold ITr-throughout-def)
apply (erule conjE)+
apply (rule kexIRI)
prefer 2
apply (rule conjI, assumption)
apply (rule ITr-connected)
defer 1
apply assumption+
done

Modeling Verification Functions

lemma [Tr.P.Ver-True: [<i,j>:ITr-performs A (Ver k l P); P] \Rightarrow <i,j>:ITr-performs A (Ver k l True)]
lemma [Tr.P.Ver-False: [<i,j>:ITr-performs A (Ver k l P); \neg P] \Rightarrow <i,j>:ITr-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:** \(<i,j>:<A> \text{Tr-empty}\)
- **ITr-init:** \(<i,j>:<A>(\text{ITr-empty } A)\)
- **ITr-Ready:** \(<i,j>: \text{ITr-empty } A \implies <i,j>:<R>(\text{ITr-stable } A | \text{ITr-performs } A \text{ Ready})\)

**Initiate:**
\[
<i,j>: \text{ITr-performs } A (\text{SSate } 1) \implies <i,j>:<L>(\text{ITr-performs } A \text{ Ready } | \text{ITr-stable } A)
\]

**Respond:**
\[
<i,j>: \text{ITr-performs } A (\text{RSate } 1) \implies <i,j>:<L>(\text{ITr-performs } A \text{ Ready } | \text{ITr-stable } A)
\]

**Abort:**
\(<i,j>: \text{ITr-performs } A \text{ Abort} \implies <i,j>:<R>\text{ITr-stable } A | \text{ITr-performs } A \text{ Ready}\)

**Accept:**
\(<i,j>: \text{ITr-performs } A \text{ Accept} \implies <i,j>:<R>\text{ITr-stable } A | \text{ITr-performs } A \text{ Ready}\)

**theorem** **Tr-C-uniq:** \([<i,j>: \text{Tr-causes } A \ e1; <i,j>:\text{Tr-causes } A \ e2] \implies <i,j>:e1 = e2\)
- apply (unfold Tr-causes-def)
- apply (erule iconjE)+
- apply (erule lexE)+
- apply (erule iconjE)+
- apply (subgoal-tac <i,j> : Tr ↓ A = e1 # evs ∧ Tr ↓ A = e2 # evsa → e1 = e2)
- apply (tactic LFast-tac 1)
- apply (rule labelICF)
- apply auto
- apply rule +
- apply (rule TimedLists.CFequ)+
- by (rule CFequ-eut)

**theorem** **ITr-P-uniq:** \([<i,j>: \text{ITr-performs } A \ a1; <i,j>:\text{ITr-performs } A \ a2] \implies <i,j>:a1 = a2\)
- apply (unfold ITr-performs-def)
- apply (erule iconjE)+
apply (erule lexE)+
apply (erule conjE)+
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 Example: Simple Protocols

A.7.1 Network Delays

theory Simple = Timed:

Creating a Model

consts
\[ \delta \] :: real
\[ \epsilon \] :: real

syntax
@\[ \delta \] :: 'a \Rightarrow real \quad (\delta)
@\[ \epsilon \] :: 'a \Rightarrow real \quad (\epsilon)

translations
\[ \delta \equiv \delta \]
\[ \epsilon \equiv \epsilon \]

axioms
RI\[ \delta \] [intro]: RI \[ \delta \]
RI\[ \epsilon \] [intro]: RI \[ \epsilon \]

consts
\[ M_1 \] :: msg
\[ M_2 \] :: msg
\[ M_3 \] :: msg

axioms
\[
\text{Rev}: \quad [i,j]: \text{Tr-occurs}(\text{Rev} \ B \ [(A,A) \rightarrow (B,B):m]); \ A \neq B \quad \Longrightarrow \quad [i,j]: \text{<L}((\text{Tr-occurs}(\text{Snd} \ A \ [(A,A) \rightarrow (B,B):m]) \land \text{Tr-quiet } B) \\
\text{<L}((\text{Tr-stable } A \land \text{Tr-stable } B))
\]

\[
\text{Bk}: \quad [i,j]: \text{Tr-occurs}(\text{Bk} \ B \ [(A,A) \rightarrow (B,B):m]); \ A \neq B \quad \Longrightarrow \quad [i,j]: \text{<L}((\text{Tr-occurs}(\text{Rev} \ B \ [(A,A) \rightarrow (B,B):m]) \land \text{Tr-quiet } A) \\
\text{<L}((\text{Tr-stable } A \land \text{Tr-stable } B))
\]

\[
\text{Snd}: \quad [i,j]: \text{Tr-occurs}(\text{Snd} \ A \ [(A,A) \rightarrow (B,B):M_t]); \ A \neq B \quad \Longrightarrow \quad [i,j]: \text{<L}((\text{Tr-empty}) \lor \\
\text{<L}((\text{Tr-occurs}(\text{Snd} \ A \ [(A,A) \rightarrow (C,C):M_S])); \ A \neq B) \land \text{Tr-quiet } C) \\
\text{<L}((\text{Tr-stable } A \land \text{Tr-stable } C)); \ A \neq B) \land \text{Tr-quiet } C) \\
\text{<L}((\text{Tr-occurs}(\text{Bk} \ A \ [(C,C) \rightarrow (A,A):M_S])); \ A \neq B) \land \text{Tr-quiet } C) \\
\text{<L}((\text{Tr-stable } A \land \text{Tr-stable } C))
\]
\[<i,j>:\text{Tr-occurs}(\text{Snd } B \mod ((B, B) \rightarrow (A, A) : \mathcal{M} 2)) ; A \neq B] \implies \neg <i,j>: \text{<L> ((\text{Tr-occurs}(\text{Blk } B \mod ((A, A) \rightarrow (B, B) : \mathcal{M} 1)) \land \text{Tr-quiet } A) \implies (\text{Tr-stable } B \land \text{Tr-stable } A))} \]

\[<i,j>: \text{Tr-occurs}(\text{Snd } A \mod ((A, A) \rightarrow (B, B) : \mathcal{M} 3)) ; A \neq B] \implies \neg <i,j>: \text{<L> ((\text{Tr-occurs}(\text{Blk } A \mod ((B, B) \rightarrow (A, A) : \mathcal{M} 2)) \land \text{Tr-quiet } B) \implies (\text{Tr-stable } A \land \text{Tr-stable } B))} \]

axioms
delay: \[<i,j>: (\text{Tr-occurs}(\text{Snd } A \ P) \mod \text{True} \mod \text{Tr-occurs}(\text{Rev } B \ P)); A \neq B] \implies <i,j>: \delta \leq \text{len} \]

Proving Properties

lemma snd-1snd2-imp-delay:
\[
[[A \neq B ; RI A ; RI B] \implies <i,j>: \text{Tr-causes } A \mod ((A, A) \rightarrow (B, B) : \mathcal{M} 1)) \mod \text{Tr-stable } A \mod \text{Tr-causes } B \mod ((B, B) \rightarrow (A, A) : \mathcal{M} 2)) \implies \delta \leq \text{len}
\]

apply (rule impI)
apply (crude \text{IchopsubE}+)
apply (drule Tr-C-imp-O [THEN Snd2], assumption+)
apply (elim \text{SndE}, assumption, crude \text{IchopsubE}, (crude conjE)+)
apply (drule Blk, assumption+)
apply (elim \text{SndE}, assumption, crude \text{IchopsubE}, (crude conjE)+)
apply (frule Rev, assumption+)
apply (elim \text{SndE}, assumption, crude \text{IchopsubE}, (crude conjE)+)
apply (subgoal_tac \text{<ke,ka>:Tr-stable } A)
prefer 2
apply (rule Tr-silent-imp-S)
prefer 2
apply (rule \text{IchopsubI})
prefer 3
apply assumption+
apply (rule \text{IchopsubI})
prefer 3
apply assumption+
defer 1
apply (subgoal_tac \text{<kg,ka>:Tr-stable } A)
prefer 2
apply (rule Tr-silent-imp-S)
prefer 2
apply (rule `dchopsub1`)  
prefer 3  
apply assumption+  
defer 1  
apply (rule `dchopsub1`)  
prefer 3  
apply assumption+  
prefer 2  
apply assumption  
prefer 3  
apply (rule `Tr-CS-CS-equ`)  
apply (rule `Tr-O-Snd-imp-C-Snd`)+  
apply (rotate-tac -1)  
apply assumption+  
apply (rule `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 `zlessen-fwd-imp-zlessen`)  
prefer 2  
apply (rule `Tr-S-imp-zlessen`)  
apply assumption+  
apply (rule `zlessen-fwd-imp-zlessen`)  
apply (rule `fwd-fwd`)  
prefer 2  
apply (rotate-tac 9)  
apply assumption+  
apply (rule `Tr-S-imp-zlessen`)  
apply assumption  
apply (rule `ssub-lelen-imp-lelen`)  
apply rule  
apply (rule `dssubI`)  
prefer 2  
apply (rule `delay`)  
apply (rule `dchopsubl`)  
prefer 3  
apply assumption
prefer 3
apply (rule \textit{schopsubI})
prefer 4
apply assumption+
apply (rule \textit{ITrueI})
apply assumption+
apply (rule \textit{fwd-fwd})
apply assumption+
deref 1
apply (rule \textit{fwd-fwd})
apply assumption+
apply (rule \textit{fwd-fwd [THEN fwd-fwd]})
apply assumption+
apply (rotate-tac -8)
apply (simp only: \textit{Tr-occurs-def Tr-causes-def})
apply (erule \textit{lexE})
apply (rotate-tac -1)
apply (erule \textit{iconyE})
apply (drule \textit{zeropoint})
apply (rotate-tac -3)
apply (drule \textit{zeropoint})
apply (drule \textit{point-fwd}+)
apply (rule \textit{fwd-fwd [THEN fwd-fwd]})
prefer 3
apply assumption+
done

\textbf{lemma} $M2\text{-imp-M1}$:
\begin{align*}
[[A \neq B; <i,j>: \textit{fwd}]] & \rightarrow \\
<i,j>: \textit{Tr-occurs}(\textit{Snd} A \{[(A,A)\rightarrow (B,B); M2]\}) & \rightarrow \\
<L>(\textit{Tr-occurs}(\textit{Snd} B \{[(B,B)\rightarrow (A,A); M1]\}) \mid \lnot \textit{True}) & \rightarrow \\
\end{align*}
apply (rule \textit{tmpI})
apply (drule \textit{Snd2})
apply (rule \textit{not-sym})
apply assumption
apply (erule \textit{shyE}, assumption)
apply (erule \textit{IchopsubE})
apply (erule \textit{iconyE})+
apply (drule \textit{Blk})
apply (rule \textit{not-sym})
apply assumption
apply (erule \textit{shyE}, assumption)
apply (rule ichopsubE)
apply (rule ichopsubE)
apply (rule ichopsubE)
apply (rule ichopsubE+)
apply (rule lconjE)
apply (rule lconjE+)
apply (rule Rcv)
apply (rule not-sym)
apply assumption
apply (rule snI, assumption)
apply (rule snE, assumption)
apply (rule snI, assumption)
apply (rule snE, assumption)
apply (rule snI, assumption)
apply (rule snE, assumption)
apply (rule snI, assumption)
apply (rule snE, assumption)
by assumption+

lemma ichop-chop-sub:
[[<i, k> :: P | l] Q; <i, k> :: fud; <k, j> :: fud; <k, j> :: R] \(\Rightarrow\)
<i, j> :: P | l] Q | l] R

lemma fud-grow:
[[<k, j> :: l <= len; <k, j> :: fud; <i, k> :: fud; RI l]] \(\Rightarrow\)
<i, j> :: l <= len

lemma E-Rcv-delay:
[[<i, j> :: fud; A \not= B; Ri \delta] \(\Rightarrow\)
<i, j> :: Tr-empty | l] Tr-\emptyset (Tr-occurs (Rcv B (~(A,B) \rightarrow (B,B):M1)) | l] Tr-\emptyset (Tr-occurs (Rcv B (~(A,B) \rightarrow (B,B):M1)) | l] Tr-\emptyset (Tr-occurs (Rcv B (~(A,B) \rightarrow (B,B):M1)) | l] Tr-\emptyset (Tr-occurs (Rcv B (~(A,B) \rightarrow (B,B):M1)) | l]
<= len

apply (rule lamp1)
apply (rule ichopsubE)
apply (rule ichopsubE)
apply (rule ichopsubE+)
apply (rotate-tac \delta)
apply (frule Rcv)
apply assumption+
apply (rule snE)
apply assumption
apply (rotate-tac 9)
apply (erule ichopsubE) +
apply (erule lconjE) +
apply (rotate-tac − 6)
apply (frule ichopsubI)
prefer 2
apply (rotate-tac 1)
apply assumption
apply assumption
apply (subgoal-tac <\text{k}_c,\text{k}_a>:\text{True})
apply (rotate-tac − 1)
apply assumption
apply (rule ITrueI)
apply (rotate-tac − 11)
apply (erule \text{ichop-chop-sub})
apply assumption
apply (rotate-tac − 1)
apply assumption
apply (erule delay, assumption)
apply (erule Tr-E-O-fud)
apply assumption
apply (erule fud-grow)
apply (rule fud-fud)
apply assumption +
apply rule
apply (erule fud-grow)
apply (rule fud-fud)
apply assumption +
apply rule
by assumption

lemma asub-Rcv:
[[\text{l},\text{j}]>:\text{fud}; \ A \neq \ B] \implies
<\text{l},\text{j}>[[S](\text{Tr-occurs}(\text{Rcv} \ B \ \{(A,A)\to(B,B):m\})] \implies
<\text{l}>((\text{Tr-occurs}(\text{Snd} \ A \ \{(A,A)\to(B,B):m\}) \land \text{Tr-quiet} \ B) \land \text{Tr-stable} \ A) \land \text{Tr-stable} \ B))]

lemma asub-Snd2:
[[\text{l},\text{j}]>:\text{fud}; \ A \neq \ B] \implies
<\text{l},\text{j}>[[S](\text{Tr-occurs}(\text{Snd} \ B \ \{(B,B)\to(A,A):M2\})] \implies
<\text{l}>((\text{Tr-occurs}(\text{Blk} \ B \ \{(A,A)\to(B,B):M1\}) \land \text{Tr-quiet} \ A) \land \text{Tr-stable} \ B) \land \text{Tr-stable} \ A))]}
lemma asub-Snd3:
\[
\langle i, j \rangle; \text{fwd}; A \neq B \implies \\
\langle i, j \rangle; [S](T\text{-occurs}(Snd A \triangleright (A, A) \rightarrow (B, B); MS)) \implies \\
\langle L \rangle((T\text{-occurs}(Blk A \triangleright (A, B) \rightarrow (A, A); M2))) \wedge (T\text{-quiet } B)) \uparrow ((T\text{-stable } A \wedge (T\text{-stable } B)))
\]
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 )
  Repsilon [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)) | ~ lTr-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)) | ~ lTr-occurs(Snd A
  ((A,A)→(B,B):M 2))) ∧
  <R>((lTr-stable A ∧ len = Δ(2)) | ~ lTr-performs A Accept)

axioms
  SuccVert: [[i,j]: lTr-performs A (Ver 1 1 True)] ⇒
\[<i,j>: <R>((\text{ITr-stable } A \land \text{len } = \varepsilon) \lor \neg \text{ITr-performs } A (\text{SState } 2))\]

**SuccVer2:** \[<i,j>: \text{ITr-performs } A (\text{Ver } 2 \land \text{True}) \Rightarrow<\]
\[<i,j>: <R>((\text{ITr-stable } A \land \text{len } = \varepsilon) \lor \neg \text{ITr-performs } A \text{ Accept})\]

**FailVer:** \[<i,j>: \text{ITr-performs } A (\text{Ver } k \land \text{False}) \Rightarrow<\]
\[<i,j>: <R>((\text{ITr-stable } A \land \text{len } = \varepsilon) \lor \neg \text{ITr-performs } A \text{ Abort})\]

**Axioms**

**Rec:** \[<i,j>: \text{Tr-causes } A (\text{Rec } A (\text{Tstate } k)); A \neq B \Rightarrow<\]
\[<i,j>: <L>(\neg \text{ITr-performs } A (\text{RState } 1) \lor \neg \text{ITr-stable } A) \land<\]
\[<L>((\text{Tr-causes } B (\text{Sm} (B, B) \rightarrow (A, A): M k); A \neq B) \land \text{Tr-quiet A}) \lor (\text{Tr-stable } A \land \text{Tr-stable } B)) \land \]
\[<R>((\text{ITr-stable } A \land \text{len } = \delta(k)(1)) \lor \neg (\text{ITr-performs } A (\text{Ver } k 1 (\text{chkAgent } B)))) )\]

**Proving Properties**

**Lemma** \(\text{Tr-C-imp-zlen}: <i,j>: \text{Tr-causes } e \Rightarrow<\]
\(<i,j>: \text{len } = 0\)

**Lemma** \(\text{ITr-P-imp-zlen}: <i,j>: \text{ITr-performs } e \Rightarrow<\]
\(<i,j>: \text{len } = 0\)

**Lemma** \(\text{zero-zero-same} : [<i,j>: P; <i,j>: \text{len } = 0; <k,l>: \text{len } = 0; <j,l>: \text{len } = 0] \Rightarrow<\]
\(<k,l>: P\)

**Theorem** \(\text{Rec-bogus-imp-delay-Abort}: [<i,j>: \text{Tr-causes } A (\text{Rec } A (\text{Spy } k, \text{Spy } k) \rightarrow (A, A): M 1));<i,j>: \text{fwd}; A \neq \text{Spy}] \Rightarrow<\]
\(<i,j>: <R>(\text{len } = (\delta(1)+1) + \varepsilon) \lor \neg \text{ITr-performs } A \text{ Abort})\)

**Apply** (drule Recv, assumption)

**Apply** (erule leqjE)

**Apply** (erule smNE, assumption)

**Apply** (erule ichtsubE)

**Apply** simp

**Apply** (erule ITr-P-imp-zlen)

**Apply** (drule FailVer)

**Apply** (erule smNE, assumption)

**Apply** (erule ichtsubE)

**Apply** (rule smI, assumption)

**Prefer** 2

**Apply** (rule ichtsub1)

**Prefer** 4

**Apply** assumption

**Apply** simp-all

**Apply** (rule fwd-fwd, assumption+)

**Prefer** 2
apply (rule fwd-fwd, assumption+)
apply (rule `conjE)+
apply (rule plusI)
apply assumption
apply (drule plusI, assumption)
by auto

**Theorem** `Recv-bogus-imp-no-Accept`: `<i,j> : Tr-causes A (Recv A \{(Spy,Spy)\} \rightarrow (A,A); M \{i\} | \{ITr-stable A | \{ITr-performs A (Ver 1 1 True); <i,j>:fwd; A \neq Spy\}
\Rightarrow <i,j> : False`

apply (drule `chopsubE)+
apply (frule Recv)
apply assumption
apply (drule `conjE)+
apply (drule `conjE)
apply (rotate-tac \{-5\})
apply (frule ITr-SP-SP-den)
apply (rotate-tac \{1\})
apply assumption+
apply (rotate-tac \{-4\})
apply (frule ITr-P-imp-den)
apply (rotate-tac \{-4\})
apply (frule ITr-P-imp-den)
apply (frule zero-zero-same)
apply assumption
apply assumption
prefer 2
apply simp
apply (rotate-tac \{-1\})
apply (drule ITr-P-uniq)
apply simp
apply (rule rigid)
apply simp
apply rule
apply (rule zeroE1)
by assumption+
end
The following is a listing of the complete axiomatic foundation of NLDCHOL as it is encoded in Isabelle/HOL by Rasmussen.

B.1 LFOLHOL

\[ LFOLHOL = \text{Real} + \]

\textit{global}

\textit{types}
\[ T \]

\textit{arities}
\[ T :: \text{logic} \]

\textit{consts}
\[ LF :: [T, T, \text{bool}] \Rightarrow \text{prop} \quad ((\langle -, - \rangle : (-)) [6,6,5] 4) \]
\[ RI :: 'a::logic \Rightarrow \text{prop} \quad ((RI :)) \]
\[ CF :: \text{bool} \Rightarrow \text{prop} \quad ((CF :)) \]
\[ <\rightarrow :: [\text{bool}, \text{bool}] \Rightarrow \text{bool} \quad (\text{infix 25}) \]

\textit{syntax (xsymbols)}
\[ \textit{op} \langle\rightarrow \rangle :: [\text{bool}, \text{bool}] \Rightarrow \text{bool} \quad (\text{infix \leftarrow 25}) \]

\textit{local}

\textit{defs}
\[ \text{iff-def} \quad P\langle\rightarrow\rangle Q \quad \equiv \quad (P\rightarrow Q) \& \quad (Q\rightarrow P) \]
rules

\texttt{lconj1} \quad [[ \langle i, j \rangle : P; \langle i, j \rangle : Q ]] \rightarrow \langle i, j \rangle : P \& Q
\texttt{lconjE1} \quad \langle i, j \rangle : P \& Q \rightarrow \langle i, j \rangle : P
\texttt{lconjE2} \quad \langle i, j \rangle : P \& Q \rightarrow \langle i, j \rangle : Q

\texttt{idisj1} \quad \langle i, j \rangle : P \rightarrow \langle i, j \rangle : P \or Q
\texttt{idisj2} \quad \langle i, j \rangle : Q \rightarrow \langle i, j \rangle : P \or Q
\texttt{idisjE} \quad \[[ \langle i, j \rangle : P \or Q; \langle i, j \rangle : P \rightarrow \langle k, l \rangle : R; \langle i, j \rangle : Q \rightarrow \langle k, l \rangle : R ]] \rightarrow \langle k, l \rangle : R

\texttt{limpI} \quad (\langle i, j \rangle : P \rightarrow \langle i, j \rangle : Q) \rightarrow \langle i, j \rangle : P \rightarrow Q
\texttt{bmp} \quad [\langle i, j \rangle : P \rightarrow Q; \langle i, j \rangle : P] \rightarrow \langle i, j \rangle : Q

\texttt{ITrue} \quad \langle i, j \rangle : True
\texttt{IFalseE} \quad (\langle i, j \rangle : P \rightarrow False) \rightarrow \langle i, j \rangle : P

\texttt{all} \quad (\forall x: 'a. (R x) \rightarrow \langle i, j \rangle : (P x)) \rightarrow \langle i, j \rangle : (\forall x. (P x))
\texttt{allERI} \quad [[ R1 s; \langle i, j \rangle : (\forall x. (P x)) ]] \rightarrow \langle i, j \rangle : (P s)
\texttt{allECF} \quad [[ CF (P x); \langle i, j \rangle : (\forall x. (P x)) ]] \rightarrow \langle i, j \rangle : (P s)

\texttt{lex1RI} \quad [[ R1 s; \langle i, j \rangle : (P s) ]] \rightarrow \langle i, j \rangle : (\exists x: 'a. (P x))
\texttt{lex1CF} \quad [[ CF (P x); \langle i, j \rangle : (P s) ]] \rightarrow \langle i, j \rangle : (\exists x: 'a. (P x))
\texttt{lexE} \quad [[ \langle i, j \rangle : \exists x: 'a. (P x); \forall x. (R1 x; \langle i, j \rangle : (P x)) ]] \rightarrow \langle k, l \rangle : R \rightarrow \langle k, l \rangle : R

\texttt{lrefl} \quad \langle i, j \rangle : s = (s: 'a)
\texttt{lsubst} \quad [\langle i, j \rangle : CF (P x); \langle i, j \rangle : s = t; \langle i, j \rangle : (P s) ] \rightarrow \langle i, j \rangle : (P (t: 'a))
\texttt{lsubstrI} \quad [\langle i, j \rangle : RI s; RI t; \langle i, j \rangle : s = t; \langle i, j \rangle : (P s) ] \rightarrow \langle i, j \rangle : (P (t: 'a))

(* Order axioms *)

\texttt{orefl} \quad \langle i, j \rangle : s = s
\texttt{otran} \quad [\langle i, j \rangle : s = t; \langle i, j \rangle : t = u ] \rightarrow \langle i, j \rangle : s = u
\texttt{oanti} \quad [\langle i, j \rangle : s = t; \langle i, j \rangle : t = s ] \rightarrow \langle i, j \rangle : s = t
\texttt{oless} \quad \langle i, j \rangle : s < t \rightarrow s < t & s < t

(* Labels *)

\texttt{labelRI} \quad [\langle i, j \rangle : P; P ] \rightarrow \langle i, j \rangle : P
\texttt{labelCF} \quad [\langle i, j \rangle : CF; P ] \rightarrow \langle i, j \rangle : P
\texttt{labelE} \quad (\forall i, j. \langle i, j \rangle : P) \rightarrow P
(* Rigidity and Chopfreeness *)

\[
\begin{align*}
\text{rigid} & \quad || <k,l>:P; \ RI \ P \ || \implies <i,j>:P \\
RItrue & \quad RI \ True \\
RIfalse & \quad RI \ False \\
RIzero & \quad RI \ 0 \\
RIone & \quad RI \ 1 \\
RIconji & \quad || RI \ P; \ RI \ Q \ || \implies (RI \ P \& \ Q) \\
RIconji1 & \quad (RI \ P \& \ Q) \implies RI \ P \\
RIconji2 & \quad (RI \ P \& \ Q) \implies RI \ Q \\
RIdisj & \quad || RI \ P; \ RI \ Q \ || \implies (RI \ P \| \ Q) \\
RIdisj1 & \quad (RI \ P \| \ Q) \implies RI \ P \\
RIdisj2 & \quad (RI \ P \| \ Q) \implies RI \ Q \\
RIimp1 & \quad || RI \ P; \ RI \ Q \ || \implies (RI \ P \implies Q) \\
RIimp1 & \quad (RI \ P \implies Q) \implies RI \ P \\
RIimp2 & \quad (RI \ P \implies Q) \implies RI \ Q \\
RIall1 & \quad ((RI \ s) \implies RI (P(s))) \implies RI (ALL \; x::'a. \; P(x)) \\
RIall1 & \quad || RI \ s; \ RI \ (ALL \; x. \; P(x)) \ || \implies RI \ P(s::'a) \\
RIex1 & \quad ((RI \ s) \implies RI (P(s))) \implies RI (EX \; x::'a. \; P(x)) \\
RIex1 & \quad || RI \ s; \ RI \ (EX \; x. \; P(x)) \ || \implies RI \ P(s::'a) \\
RIequ1 & \quad || RI \ s; \ RI \ t \ || \implies (RI \ (s=(t::real))) \\
RIequ1 & \quad (RI \ (s=(t::real))) \implies RI \ s \\
RIequ1 & \quad (RI \ (s=(t::real))) \implies RI \ t \\
RIadd1 & \quad || RI \ s; \ RI \ t \ || \implies (RI \ (s+t)) \\
RIadd1 & \quad (RI \ (s+t)) \implies RI \ s \\
RIadd2 & \quad (RI \ (s+t)) \implies RI \ t \\
RIneg1 & \quad (RI \ s) \implies RI \ (-s) \\
RIneg1 & \quad (RI \ (-s)) \implies RI \ s \\
RImul1 & \quad || RI \ s; \ RI \ t \ || \implies (RI \ (s*t)) \\
RImul1 & \quad (RI \ (s*t)) \implies RI \ s \\
RImul2 & \quad (RI \ (s*t)) \implies RI \ t \\
RIdiv1 & \quad || RI \ s; \ RI \ t \ || \implies (RI \ (s/t)) \\
RIdiv1 & \quad (RI \ (s/t)) \implies RI \ s \\
RIdiv2 & \quad (RI \ (s/t)) \implies RI \ t \\
RIleq1 & \quad || RI \ s; \ RI \ t \ || \implies (RI \ (s<=t)) \\
RIleq1 & \quad (RI \ (s<=t)) \implies RI \ s \\
RIleq2 & \quad (RI \ (s<=t)) \implies RI \ t \\
CFtrue & \quad CF \ True
\end{align*}
\]
\texttt{CF}\texttt{false} \texttt{CF False}
\texttt{CF}\texttt{conj1} \texttt{|| CF P, CF Q \implies CF (P\&Q)}
\texttt{CF}\texttt{conjE1} \texttt{(CF P\&Q) \implies CF P}
\texttt{CF}\texttt{conjE2} \texttt{(CF P\&Q) \implies CF Q}
\texttt{CF}\texttt{disj1} \texttt{|| CF P, CF Q \implies CF (P|Q)}
\texttt{CF}\texttt{disjE1} \texttt{(CF P|Q) \implies CF P}
\texttt{CF}\texttt{disjE2} \texttt{(CF P|Q) \implies CF Q}
\texttt{CF}\texttt{imp1} \texttt{|| CF P, CF Q \implies CF (P\implies Q)}
\texttt{CF}\texttt{impE1} \texttt{(CF P\implies Q) \implies CF P}
\texttt{CF}\texttt{impE2} \texttt{(CF P\implies Q) \implies CF Q}
\texttt{CF}\texttt{all} \texttt{CF \ P(s) \implies CF (ALL x::'a. \ P(x))}
\texttt{CF}\texttt{allE} \texttt{CF (ALL x. \ P(x)) \implies CF \ P(s::'a)}
\texttt{CF}\texttt{ex} \texttt{CF \ P(s) \implies CF (EX x::'a. \ P(x))}
\texttt{CF}\texttt{exE} \texttt{CF (EX x. \ P(x)) \implies CF \ P(s::'a)}
\texttt{CF}\texttt{equ} \texttt{CF \ (s=t::real) \implies CF \ P(s::'a)}
\texttt{CF}\texttt{less} \texttt{CF \ (s<t) \implies CF \ P(s::'a)}
\texttt{CF}\texttt{le} \texttt{CF \ (s\leq t) \implies CF \ P(s::'a)}

\textbf{B.2 LSILHOL}

\textit{LSILHOL = LFOLHOL +
\global
\consts
\chop :: [bool, bool] \Rightarrow bool \ (\text{infixr } \sim)
\len :: real
\conv :: bool \Rightarrow bool
\fud :: bool
\bwd :: bool
\syntax \textbf{(asymbols)}
\op :: [bool, bool] \Rightarrow bool \ (\text{infixr } \sim)
\local
\defs
B.3 LSDCHOL

\[ \text{conv-def} \quad \text{conv}(P) == (EX \ z. \ (\text{len}=z) \ & \ (\text{len}=0 \ & \ (\text{len}=z) \ ^\lor \text{True}) \ ^\land \text{True}) \]

\[ \text{bud-def} \quad \text{bud} == \text{len} <= 0 \]

\[ \text{fwd-def} \quad \text{fwd} == 0 <= \text{len} \]

\[ \text{rules} \]

\[ \text{chopI} \quad || <i, k>:P; <k, j>:Q || ==\Rightarrow (<i, j>:P ^\lor (Q :: \text{bool})) \]

\[ \text{chopE} \quad || <i, j>:P ^\lor (Q :: \text{bool}); \ !k. \ || <i, k>:P; <k, j>:Q || ==\Rightarrow <l, m>:R \]

\[ \text{uniq1} \quad || RI \ s; <i, k>:\text{len}=s; <i, l>:\text{len}=s; <l, j>:P || ==\Rightarrow <k, j>:P \]

\[ \text{uniq2} \quad || RI \ s; <k, j>:\text{len}=s; <l, j>:\text{len}=s; <i, l>:P || ==\Rightarrow <i, k>:P \]

\[ \text{zero} \quad <i, i>:\text{len}=0 \]

\[ \text{plusI} \quad || <i, k>:\text{len}=s; <k, j>:\text{len}=t; RI \ s; RI \ t || ==\Rightarrow <i, j>:\text{len}=s+t \]

\[ \text{plusE} \quad || <i, j>:\text{len}=s+t; RI \ s; RI \ t; \ !k. \ || <i, k>:\text{len}=s; <k, j>:\text{len}=t || ==\Rightarrow <l, m>:R || ==\Rightarrow <l, m>:R \]

(* Rigidity *)

\[ \text{RchopI} \quad || RI \ P; RI \ Q || ==\Rightarrow RI \ (P ^\lor (Q :: \text{bool})) \]

\[ \text{RchopE1} \quad (RI \ (P ^\lor (Q :: \text{bool}))) ==\Rightarrow RI \ P \]

\[ \text{RchopE2} \quad (RI \ (P ^\lor (Q :: \text{bool}))) ==\Rightarrow RI \ Q \]

setup LCla.setu
setup Lelasetup

end

B.3 LSDCHOL

\[ LSDCHOL = LSILHOL + \]

\[ \text{global} \]

\[ \text{types} \]

\[ s \]

\[ \text{arities} \]

\[ s :: \text{type} \]
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  ((\'))
| 'I' :: [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
DA#b  <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

IRI  || <i,j>:len=0 --> P; <i,j>:\'S'\|^\'P --> P; <i,j>:\'NOT S'\|^\'P --> P

--> P ||
B.4 ILDCHOL

\[
\Rightarrow \langle i,j \rangle : P
\]

end

B.4 ILDCHOL

\(ILDCHOL = LSDCHOL +\)

global

\[
\begin{align*}
\text{const} & \quad \text{ssub} :: \text{bool} \to \text{bool} \quad (\langle S \rangle : [50] 50) \\
& \quad \text{asub} :: \text{bool} \to \text{bool} \quad ([S] : [50] 50) \\
& \quad \text{spref} :: \text{bool} \to \text{bool} \quad (\langle P \rangle : [50] 50) \\
& \quad \text{apref} :: \text{bool} \to \text{bool} \quad ([P] : [50] 50)
\end{align*}
\]

local

\[
\begin{align*}
\text{def} & \quad \text{ssub-def} \quad \langle S \rangle P \Leftarrow \text{True} | \neg [P] \neg \text{True} \\
& \quad \text{asub-def} \quad [S] P \Leftarrow \neg (\langle S \rangle \neg [P]) \\
& \quad \text{spref-def} \quad \langle P \rangle P \Leftarrow P \neg \text{True} \\
& \quad \text{apref-def} \quad [P] P \Leftarrow \neg (\langle P \rangle \neg [P])
\end{align*}
\]

end

B.5 NLDCHOL

\(NLDCHOL = ILDCHOL +\)

global

\[
\begin{align*}
\text{const} & \quad \text{srn} :: \text{bool} \to \text{bool} \quad (\langle R \rangle : [50] 50) \\
& \quad \text{arn} :: \text{bool} \to \text{bool} \quad ([R] : [50] 50) \\
& \quad \text{sln} :: \text{bool} \to \text{bool} \quad (\langle L \rangle : [50] 50) \\
& \quad \text{alm} :: \text{bool} \to \text{bool} \quad ([L] : [50] 50) \\
& \quad \text{srnmv} :: \text{bool} \to \text{bool} \quad (\langle R \neg \rangle : [50] 50)
\end{align*}
\]
arnrev :: bool => bool ((R[1..][50] 50)
shrrev :: bool => bool ((L[1..]- [50] 50)
almrev :: bool => bool ((L[1..]- [50] 50)
srnconv :: bool => bool ((Re[1..]- [50] 50)
slnconv :: bool => bool ((Le[1..]- [50] 50)

sfwd :: bool => bool ((A[1..]- [50] 50)
afwd :: bool => bool ((A[1..]- [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
shrrev-def <L->P === <R>P
alm-def [R]P === ^ (<R>("P))
alm-def [L]P === ^ (<L>("P))
srnrev-def [R->]P === [L->]("P))
slnrev-def [L->]P === [R->]("P))
srnconv-def <Re>P === <Re-><R>P
slnconv-def <Le>P === <Le-><L>P
sfwd-def <A>P === True ^ (P & fwd) ^ True
afwd-def [A]P === ^ (<A>("P))
APPENDIX C

Case Study: IPSEC - OAKLEY

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 [Mee99] 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 $\text{KEYID} \equiv \text{COOKIE-I} \mid \text{COOKIE-R}$.

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.

\[
OAKS = \begin{cases} 
I \rightarrow R : & 0, 0, \kappa x \quad \text{(1)} \\
R \rightarrow I : & 0, C_R, \kappa x \quad \text{(2)} \\
I \rightarrow R : & C_I, C_R, \kappa x, G, g^{\alpha_I}, \text{Opt} \quad \text{(3)} \\
R \rightarrow I : & C_R, C_I, \kappa x, G, g^{\alpha_R}, \text{Sel} \quad \text{(4)} \\
I \rightarrow R : & C_I, C_R, \kappa x, G, g^{\alpha_I}, \text{idP}, \\
& ID_I, ID_R, E\{N_I\}_{K_R} \quad \text{(5)} \\
R \rightarrow I : & C_R, C_I, \kappa x, G, 0, 0, \text{idP}, \\
& E\{N_R, N_I\}_{K_I}, ID_R, ID_I, \\
& prf(K_{IR}, ID_R | ID_I | G | g^{\alpha_R} | g^{\alpha_I} | \text{Sel}) \quad \text{(6)} \\
I \rightarrow R : & C_I, C_R, \kappa x, G, 0, 0, \text{idP}, \\
& E\{N_I, N_R\}_{K_R}, ID_I, ID_R, \\
& prf(K_{IR}, ID_I | ID_R | G | g^{\alpha_I} | g^{\alpha_R} | \text{Sel}) \quad \text{(7)} 
\end{cases}
\]

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 \(\kappa x\), 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,
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
</table>
| $| \text{Concatenation operator}$
| $I$ | Initiator |
| $R$ | Responder |
| $K$ | Cryptographic or signing key |
| $K_I, K_R$ | Public keys of initiator/responder |
| $K_{IR}$ | Shared key for hashing ($\equiv prf(0, N_I \mid N_R)$) |
| $K_{prf}$ | 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 $C_I | C_R$. 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

$$ I \rightarrow R: \ C_I, C_R, \ kx, \ G, \ g^{x_I}, \ wp, \ ID_I, \ ID_R, \ E[N_I]_{k_R} \tag{5} $$

implicitly means

$$ I \rightarrow R: \ C_I, C_R, \ kx, \ G, \ g^{x_I}, \ wp, \ E[ID_I, \ ID_R, \ E[N_I]_{k_R}]_{k_{ph}} \tag{5} $$

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 $K_s \equiv prf(N_I \mid N_R, g^{x_R} \mid C_I \mid C_R)$ 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, \(K_X\), 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, \(ID_P\), 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.

\[
OAKS = \begin{cases} 
  I \rightarrow R: & 0, 0 \\
  R \rightarrow I: & 0, C_R \\
  I \rightarrow R: & C_I, C_R, G, g^{z_I}, \text{Opt} \\
  R \rightarrow I: & C_R, C_I, G, g^{z_R}, \text{Sel} \\
  I \rightarrow R: & C_I, C_R, G, g^{z_I}, \\
  & E\{ID_I, ID_R, E\{N_I\}K_R\}K_{R,S} \\
  R \rightarrow I: & C_R, C_I, G, 0, 0, \\
  & E\{E\{N_R, N_I\}K_I, ID_R, ID_I, \\
  & prf(K_{IR}, ID_R | ID_I | G | g^{z_R} | g^{z_I} \mid \text{Sel})\}K_{R,S} \\
  I \rightarrow R: & C_I, C_R, G, 0, 0, \\
  & E\{E\{N_I, N_R\}K_R, ID_I, ID_R, \\
  & prf(K_{IR}, ID_I | ID_R | G | g^{z_I} | g^{z_R} \mid \text{Sel})\}K_{R,S} 
\end{cases}
\]

The resulting Alice and Bob specification is the basis of the formal modeling I will carry out.

Man kan diskutere \(ID_P\), idet det 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
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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>cr-cky</td>
<td>Create cookie</td>
<td>1</td>
</tr>
<tr>
<td>cr-sta</td>
<td>Create state</td>
<td>1</td>
</tr>
<tr>
<td>st-cky</td>
<td>Store cookie</td>
<td>1</td>
</tr>
<tr>
<td>accept</td>
<td>Accept received message</td>
<td>0</td>
</tr>
<tr>
<td>se-grp</td>
<td>Select Diffie-Hellman group</td>
<td>1</td>
</tr>
<tr>
<td>rse-exp</td>
<td>Randomly select exponent in group</td>
<td>1</td>
</tr>
<tr>
<td>se-opt</td>
<td>State options for EHA</td>
<td>1</td>
</tr>
<tr>
<td>cr-hkey</td>
<td>Create Diffie-Hellman half-key</td>
<td>4</td>
</tr>
<tr>
<td>ch-cky</td>
<td>Check cookie</td>
<td>1</td>
</tr>
<tr>
<td>st-cky</td>
<td>Store cookie in state</td>
<td>1</td>
</tr>
<tr>
<td>st-hkey</td>
<td>Store half-key in state</td>
<td>1</td>
</tr>
<tr>
<td>cr-keypfs</td>
<td>Create key for perfect forwarding secrecy</td>
<td>4</td>
</tr>
<tr>
<td>se-sel</td>
<td>select algorithmic options for EHA</td>
<td>1</td>
</tr>
<tr>
<td>st-sel</td>
<td>Store selected options for EHA</td>
<td>1</td>
</tr>
<tr>
<td>cr-nce</td>
<td>Create nonce</td>
<td>4</td>
</tr>
<tr>
<td>re-pkey</td>
<td>Retrieve public key of participant</td>
<td>3</td>
</tr>
<tr>
<td>en-pkey</td>
<td>Encrypt using public key</td>
<td>4</td>
</tr>
<tr>
<td>en-keypfs</td>
<td>Encrypt using Diffie-Hellman key</td>
<td>3</td>
</tr>
<tr>
<td>de-keypfs</td>
<td>Decrypt using Diffie-Hellman key</td>
<td>3</td>
</tr>
<tr>
<td>ch-ID</td>
<td>Check ID of participant</td>
<td>1</td>
</tr>
<tr>
<td>st-ID</td>
<td>Store ID of participant</td>
<td>1</td>
</tr>
<tr>
<td>do-pkey</td>
<td>Decrypt using private key</td>
<td>3</td>
</tr>
<tr>
<td>st-nce</td>
<td>Store nonce</td>
<td>1</td>
</tr>
<tr>
<td>cr-keyr</td>
<td>Create shared signing key from nonces</td>
<td>4</td>
</tr>
<tr>
<td>cr-sgn</td>
<td>Create signature over items</td>
<td>3</td>
</tr>
<tr>
<td>ch-sgn</td>
<td>Check signature over items</td>
<td>3</td>
</tr>
</tbody>
</table>

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.
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 muligst åt aktion 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 incoming messages, may be postponed and considered message preparation steps instead. In the following we employ the abbreviations shown in figure C.3.
<table>
<thead>
<tr>
<th>Symb</th>
<th>Definition</th>
<th>Cont.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>cr-cryy</td>
<td>0</td>
</tr>
<tr>
<td>$T_2$</td>
<td>cr-cryy, cr-cky, se-gp, rse-ex, se-opt, cr-hkey</td>
<td>1</td>
</tr>
<tr>
<td>$T_3$</td>
<td>cr-sta, cr-cky, se-gp, rse-ex, se-opt, cr-hkey</td>
<td>10</td>
</tr>
<tr>
<td>$T_4$</td>
<td>cr-sta, cr-cky, se-gp, rse-ex, cr-hkey, cr-keypfs, se-sel</td>
<td>14</td>
</tr>
<tr>
<td>$T_5$</td>
<td>st-hkey, st-sel, cr-keypfs, cr-cke, re-pkey, en-pkey, en-keypfs</td>
<td>20</td>
</tr>
<tr>
<td>$T_6$</td>
<td>st-nce, cr-cke, re-pkey, en-pkey, cr-cke, cr-sgn, en-keypfs</td>
<td>22</td>
</tr>
<tr>
<td>$T_7$</td>
<td>en-pkey, cr-sgn, en-keypfs</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure C.3: Abbreviations 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.

\[
\text{OAKS} = \left\{\begin{array}{l}
I \rightarrow R: \quad T_1 || \\
| \quad 0, 0 \\
R \rightarrow I: \quad T_2 || \\
| \quad 0, C_R \\
I \rightarrow R: \quad T_3 || \\
| \quad C_I, C_R, G, g^{z_t}, \text{Opt} \\
R \rightarrow I: \quad T_4 || \\
| \quad C_R, C_I, G, g^{z_n}, \text{Sel} \\
I \rightarrow R: \quad T_5 || \\
| \quad C_I, C_R, G, g^{z_t}, \\
| \quad E\{ID_I, ID_R, E\{N_I\}K_n\}K_{s_p} \text{, ch-cky, de-keypfs, de-pkey} \\
R \rightarrow I: \quad T_6 || \\
| \quad C_R, C_I, G, 0, 0, \\
| \quad E\{E\{N_R, N_I\}K_n, ID_I, ID_R, ID_I, \\
| \quad \text{prf}(K_{IR}, ID_I | ID_R | G | g^{z_n} | g^{z_t} | \text{Sel})\}K_{s_p} \text{, ch-cky, de-keypfs, de-pkey, cr-cke, cr-sgn} \\
I \rightarrow R: \quad T_7 || \\
| \quad C_I, C_R, G, 0, 0, \\
| \quad E\{E\{N_I, N_R\}K_n, ID_I, ID_R, \\
| \quad \text{prf}(K_{IR}, ID_I | ID_R | G | g^{z_n} | g^{z_t} | \text{Sel})\}K_{s_p} \text{, ch-cky, de-keypfs, de-pkey, cr-cke, cr-sgn} \\
\end{array}\right\}
\]
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

\[
e \in \text{oakley}
\]

Reception

\[
\begin{aligned}
& h \in \text{oakley} \quad S_n((A, l(A)) \rightarrow (B, l(B)) : X) \in \text{visible}(h) \\
& h \cdot R_B((A, l(A)) \rightarrow (B, l(B)) : X) \in \text{oakley}
\end{aligned}
\]

\[
\begin{aligned}
& h \in \text{oakley} \quad S_n((A, l(A)) \rightarrow (B, l(B)) : X) \in \text{visible}(h) \\
& h \cdot B_B((A, l(A)) \rightarrow (B, l(B)) : X) \in \text{oakley}
\end{aligned}
\]

\[
\begin{aligned}
& h \in \text{oakley} \quad S_n((A, l(A)) \rightarrow (B, l(B)) : X) \in \text{visible}(h) \\
& h \cdot R_B((A, l(A)) \rightarrow (B, l(B)) : X) \cdot B_B((A, l(A)) \rightarrow (B, l(B)) : X) \in \text{oakley}
\end{aligned}
\]

Message 1

\[
\begin{aligned}
& h \in \text{oakley} \quad A \neq B \\
& h \cdot S_A((A, l(A)) \rightarrow (B, l(B)) : 0, 0) \in \text{oakley}
\end{aligned}
\]

Message 2

\[
\begin{aligned}
& h \in \text{oakley} \quad R_R((n, l) \rightarrow (R, l(R)) : 0, 0) = \text{last}((h \downarrow n) \downarrow R) \quad C_R \notin \text{used}(h) \\
& h \cdot S_R((B, l(B)) \rightarrow (n, l) : 0, C_R) \in \text{oakley}
\end{aligned}
\]

Message 3

\[
\begin{aligned}
& h \in \text{oakley} \quad R_I((I, l) \rightarrow (I, l(I)) : 0, C_R) = \text{last}((h \downarrow n) \downarrow R) \quad C_I \notin \text{used}(h) \\
& h \cdot S_I(I, l(I)) \rightarrow (n, l) : C_I, C_R, G, g^R, \text{Opt}) \in \text{oakley}
\end{aligned}
\]

Message 4

\[
\begin{aligned}
& h \in \text{oakley} \quad R_R((l, n) \rightarrow (R, l(R)) : C_I, C_R, G, g^R, \text{Opt}) = \text{last}((h \downarrow n) \downarrow R) \\
& h \cdot S_R((R, l(R)) \rightarrow (n, l) : C_R, C_I, G, g^{\text{sen}}, \text{Sel}) \in \text{oakley}
\end{aligned}
\]
Message 5
\[ h \in \text{oakley} \quad R_I((l, n) \rightarrow (I, l(I)) : C_R, C_I, G, g^{x_n}, Sel) = last((h \downarrow n) \downarrow R) \quad N_I \notin used(h) \]
\[ h \cdot S_I((I, l(I)) \rightarrow (n, I) : C_I, C_R, G, g^{2^l}, E\{ID_I, ID_R, E\{N_I\}_{K_{l(n)}}\}_{K_{l(n)}}) \in \text{oakley} \]

Message 6
\[ h \in \text{oakley} \quad R_R((l, n) \rightarrow (R, l(R)) : C_I, C_R, G, g^{2^l}, E\{ID_I, ID_R, E\{N_I\}_{K_{l(n)}}\}_{K_{l(n)}}) = last((h \downarrow n) \downarrow R) \quad N_R \notin used(h) \]
\[ h \cdot S_R((R, l(R)) \rightarrow (n, I) : C_R, C_I, G, 0, 0, E\{E\{N_R, N_I\}_{K_{l(n)}}\}_{K_{l(n)}}, ID_R, ID_I, prf(K_{l(n)}, ID_R | ID_I | g^{x_n} | g^{2^l} | Sel)_{K_{l(n)}}) \in \text{oakley} \]

Message 7
\[ h \in \text{oakley} \quad R_I((n, l) \rightarrow (I, l(I)) : C_R, C_I, G, 0, 0, E\{E\{N_R, N_I\}_{K_{l(n)}}\}_{K_{l(n)}}, ID_R, ID_I, prf(K_{l(n)}, ID_R | ID_I | g^{x_n} | g^{2^l} | Sel)_{K_{l(n)}}) = last((h \downarrow n) \downarrow R) \]
\[ h \cdot S_I((I, l(I)) \rightarrow (n, I) : C_I, C_R, G, 0, 0, E\{E\{N_I, N_R\}_{K_{l(n)}}\}_{K_{l(n)}}, ID_I, ID_R, prf(K_{l(n)}, ID_I | ID_R | g^{2^l} | g^{x_n} | Sel)_{K_{l(n)}}) \in \text{oakley} \]

C.4 Timed properties of OAKLEY
APPENDIX D

Mutually Inductive Definition
Experiments
D.1 A Simple Experiment

```isar
tmp = Real+

datatype num = Nul | One | Two

datatype alp = A | B

c consts numseq :: "num list set"
alpseq :: "nat ⇒ alp list set"

inductive "numseq" "alpseq Num"

<table>
<thead>
<tr>
<th>intra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Niln &quot;[]} ∈ numseq&quot;</td>
</tr>
<tr>
<td>Nila &quot;[]} ∈ alpseq Num&quot;</td>
</tr>
<tr>
<td>Nul &quot;[n : numseq; hd n = hd []]&quot; ⇒ Nul#n : numseq&quot;</td>
</tr>
<tr>
<td>One &quot;[n : numseq; a : alpseq Num; hd a = A]&quot; ⇒ One#n : numseq&quot;</td>
</tr>
<tr>
<td>A &quot;[n : numseq; a : alpseq Num; hd n = Nul]&quot; ⇒ A#a : (alpseq Num)&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>headn &quot;(x::num) = hd []&quot; = False*</td>
</tr>
<tr>
<td>heada &quot;(x::alp) = hd []&quot; = False*</td>
</tr>
<tr>
<td>headn1 &quot;hd [] = (x::num)&quot; = False*</td>
</tr>
<tr>
<td>heada1 &quot;hd [] = (x::alp)&quot; = False*</td>
</tr>
</tbody>
</table>

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 "\exists 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 "numseq0ha";
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"
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) # acs ∈ 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 bex1) prefer 2
  apply (rule Accept) prefer 5
  apply (rule_tac B = Server and 1 = 2 in Veri1) prefer 6
  apply simp prefer 5
  apply simp prefer 3
  apply (rule_tac B = Server and 1 = 2 in Rcvi) prefer 5
  apply simp prefer 4
  apply (rule_tac A = a and B = Server and i = 2 in Snd1) prefer 4
  apply simp prefer 3
  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) prefer 3
  apply (rule_tac A = a and B = Server in Snd1) prefer 3
  apply simp prefer 3
  apply simp
  apply (rule Initl) prefer 3
  apply (rule Ready)
apply (rule Nill)
apply force
apply (rule Resp1)
apply (rule Ready)
apply (rule Nill)
apply simp
apply (rule_tac i = 1 in Ver1)
prefer 3
apply simp
prefer 3
apply simp
apply (rule_tac B = a and i = 1 in Ver11)
prefer 5
apply simp
apply (rule_tac i = 1 and B = a and A = Server in Rcv1)
prefer 5
apply simp
apply (rule_tac A = a and B = Server in Sndl)
prefer 4
apply simp
apply (rule Init1)
apply (rule Ready)
apply (rule Nill)
apply simp
apply (rule Nilg)
apply (rule Resp1)
apply (rule Ready)
apply (rule Nill)
apply simp
apply (rule Resp1)
apply (rule Ready)
apply (rule Nill)
apply simp
apply (rule_tac B = Server in Sndl)
apply (rule Init1)
apply (rule Ready)
apply (rule Nill)
apply simp
apply (rule_tac A = a and B = Server in Sndl)
apply (rule Init1)
apply (rule Ready)
apply (rule Nill)
apply simp
apply (rule Nilg)
apply (rule tac B = Server in Sndl)
apply (rule Init1)
apply (rule Ready)
apply (rule Nill)
apply simp
apply (rule_tac A = a and B = Server in Sndl)
apply (rule Init1)
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 © 2001 Henrik Pilegaard
# $Id: thyssh1.epa,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, defl, constdefs, subsection, axioms, setup, text, setup, global, local, defaultsort, typedefci, inductive, datatype, primrec, "::;", typedef, lemma, theorem, lemmas, done, method_setup, default, rules, "end", proof, next, qed, use, ML, theory, intra, fun, val,
Goal, Goalw, goal, qed_spec_mp sig, br, ba, 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-20-9\+tac/

end keywords

keywords in Keyword are pop_proof, push_proof, structure, signature, intros, infir, infil, infix, binder, files, assume, OFCLASS, PROP, unfold, bal, bs, addss, deess, map, impliesOfSubs, auto, split, rule_format, iff, intro, simp, dest, elim,
blast,
simp_all,
clarify,
add,
force,
assumption,
[/a-zA-Z]*rule/,
[/a-zA-Z-0-9]+simps/,
[/a-zA-Z-0-9]+is/,
[/a-zA-Z-0-9]+Es/,
[/a-zA-Z-0-9]+Ds/
end

optional operators are
\preceq "prefix of",
\alpha α
\sum Σ
\int \int
\epsilon ε
\ge ≥
\delta δ
\triangle Δ
\nabla ∇
\leq ≤
\in ∈
\and ∧
\or ∨
\lbrace |,
\rbrace |
\equiv ≡
\lambda λ
\parallel "|,
\\parallel "|
\not ∈
\down ↓
\Longrightarrow ⇒
\lbrakk "|,
\rbrakk |
\rightarrow ⇒,
\\rightarrow ⇒
\bracket "|,
\\bracket "|

Index

Δ, 83
Δ, 47, 53
↑, 46
↓, 44
δ, 83
∇, 47, 48, 55
\texttt{ITr\_empty A}, 79

\textit{I}, vii

A, 83
\textit{Abort}, 58
\textit{Abort}, 58
\textit{Accept}, 58
\textit{Accept}, 58
\textit{action}, 58
\textit{Adr}, 36
Agent, 32
\textit{agent}, 31
Agents, 47
analz, 34

\textit{bad}, 40
\textit{Blk}, 38
Blk, 13, 38

cf, 69
\textit{chkAgent}, 63
combi\texttt{K}, 51
\textit{Crypt}, 31
Crypt, 33

\textit{event}, 38
fakeAddr, 36
fakeEvent, 39
fakePkt, 37
finish, 47
\textit{Friend}, 30
Friend, 31
goodAddr, 36
goodEvent, 39
goodPkt, 37

\textit{Hash}, 31
Hash, 33
HKey, 50
HPair, 33

\textit{Ideal}, 44
initState, 40, 41
\textit{intv}, 67
invKey, 31

Key, 32
\textit{key}, 31
knows, 41

\textit{Loc}, 36
\texttt{ITr}, 78
\texttt{ITr\_empty}, 79
\texttt{ITr\_Idles}, 79
INDEX

Tr_performs, 79
Tr_stable, 79
Tr_throughout, 79
ltrace, 60

M, 93
MPair, 31
MPair, 32
msg, 32
MsgPkt, 37
Nm, 36
Nonce, 31
Nonce, 32
Number, 31
Number, 32

parts, 33
Pkt, 37
pLNSchema, 46
priK, 41
pSchema, 46
pubK, 41

rAdrPkt, 37
Rcv, 38
Rcv, 13, 38
Ready, 58
Ready, 58
ri, 69
rLocPkt, 37
rNmPkt, 37
RState, 58
RState, 58

sAdrPkt, 37
Server, 30
Server, 31
shrK, 40
sLocPkt, 37

Snd, 38
Snd, 13, 38
sNmPkt, 37
spies, 41
Spy, 30
Spy, 31
SState, 58
SState, 58
StS, 50
symKeys, 31
synth, 35

Tr, 72
Tr Causes, 74
Tr Const, 74
Tr Empty, 74
Tr Occurs, 74
Tr Quiet, 75
Tr Stable, 74
Tr Throughout, 74
trace, 43

used, 42

Ver, 58

Ver, 58
visible, 43