Using the Theorem Prover SETHEO for Verifying the Development of a Communication Protocol in FOCUS*
— A Case Study —

Johann Schumann

Institut für Informatik
Technische Universität München
80290 München, Germany
email: schumann@informatik.tu-muenchen.de

Abstract. This paper describes experiments with the automated theorem prover SETHEO. The prover is applied to proof tasks which arise during formal design and specification in FOCUS.
These proof tasks originate from the formal development of a communication protocol (Stenning protocol). Its development and verification in FOCUS is described in "C. Dendorfer, R. Weber: *Development and Implementation of a Communication Protocol – An Exercise in* FOCUS" [DW92a]. A number of propositions of that paper deal with safety and liveness properties of the Stenning protocol on the level of traces. All given propositions and lemmata could be proven automatically using the theorem prover SETHEO.
This paper gives a short introduction into the proof tasks as provided in [DW92a]. All steps which were necessary to apply SETHEO to the given proof tasks (transformation of syntax, axiomatization) will be described in detail. The surprisingly good results obtained by SETHEO will be presented, and advantages and problems using an automated theorem prover for simple, but frequently occurring proof tasks during a formal development in FOCUS, as well as possibly ways for improvements for using SETHEO as a "back-end" for FOCUS will be discussed.

1 Introduction

During a formal development of a specification, the correctness of each step has to be proven. Many of these proofs are quite simple — when done by hand they require only few lines. Nevertheless, all these proofs have to be carried out on a highly formal level. This tends to be very time-consuming and error-prone, since a large number of proof tasks occur even in small specifications. This is especially the case, if during development parts of the specification are changed, and all the proofs have to be made again.

* This work has been carried out within the Sonderforschungsbereich SFB 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen" funded by the Deutsche Forschungsgemeinschaft.

For this kind of small proof tasks, the application of an automated theorem prover would be of great value.

In order to evaluate how the automated theorem prover SETHEO, an automated theorem prover based on the Model Elimination Calculus (for details see [LSBB92]), can be applied to such proof tasks, this case study has been made. As the basis for the experiments, the formal development of a communication protocol (Stenning protocol) carried out in FOCUS (on the level of *traces*) has been used. Its development and verification in the formal design method FOCUS is described in [DW92a] and [DW92b], from which all formulae and necessary information has been extracted.

Here again, we stress that this paper does not describe a case study on how FOCUS can be used to develop and refine specifications of a protocol. Rather, we take a given specification and study, how and if SETHEO can prove automatically the proof obligations present in the specification.

This paper proceeds as follows: first, we give a short introduction into the problem, the specification of the Stenning protocol (on the level of traces) and its refinements. We list all operators, liveness and safety properties which form the basis for the proof tasks to be tackled by SETHEO. Then, we describe all steps necessary to prepare the proof tasks for SETHEO (transformation into FOL, transformation of the notation, axiomatization, conversion into clausal form), and present the results of experiments carried out. We conclude with a summary of the experience made during this case study and focus on work to be done which will lead to the development of methods and heuristics for use of SETHEO during the development process carried out in FOCUS. The Appendix lists all axioms and formulae which have been used. The SETHEO representation of these formulae can be obtained via e-mail from the author.

We assume that the reader is familiar with the basic properties of SETHEO as well as some basic notions of FOCUS. For details on SETHEO, we refer to e.g., [LSBB92, GLMS94, LMG94], for an overview on FOCUS see e.g., [BDD+93, DDW93]. [Sch94b] contains a detailed description of this case study which includes all proofs found by SETHEO and all used formulae in SETHEO's input syntax.

2 The Proof Tasks

The given proof tasks originate from a case study about the specification of the *Stenning protocol* [Ste76]. The Stenning Protocol ensures reliable communication of upper layer data units (UDU)[2] using an unreliable transport medium ("lower layer"; see Figure 1). The medium can loose packages ("lower layer data units", LDU) or permute the sequence of packages. Reliable communication is accomplished by adding a unique sequence number to each UDU and by introducing acknowledge messages: each package is sent repeatedly, until an acknowledge message with the correct sequence number is received by the transmitter.

[2] These data units are seen as black boxes and are not further specified.

Although the application of SETHEO was carried out without detailed knowledge about the Stenning protocol or FOCUS (and how the protocol was specified), we give a short introduction on how the Stenning protocol was specified and this specification refined.

The development of the specification in FOCUS starts from an abstract description of the services provided by the upper layer and those provided by the lower layer (level of transport medium). These services are depicted in Figure 1 and will be described below. FOCUS covers all steps of the development from a non-constructive global specification down to an executable program. Several design steps are performed to accomplish the executable program.

Here, we focus on the first steps, namely the refinement of the *global* requirements specification (a trace specification) to a *modular* requirements specification (also a trace specification). This refinement process is necessary, since the (global) requirements of the upper layer cannot be separated directly into requirements of the transmitter, the receiver, and the transport medium. This process which will be also sketched in the following is described in detail in Chapter 3 of [DW92a] and the propositions shown there are the basis for the proof tasks of our case study.

Trace logic is used to specify the systems requirements. A trace specification states requirements for histories ("traces") of a distributed system as a sequence of *actions*.

The situation which we are dealing here is shown in Figure 1. The actions of that system are: $snd(d)$ and $rec(d)$ (send/receive an upper layer data unit, $d \in UDU$), $get_R(x)$ and $put_T(x)$ (send/receive a lower layer data unit, $x \in LDU$), and $get_T(ack)$ and $put_R(ack)$ to receive (and send respectively) an acknowledge message (over the transport medium)[3].

Then, a trace is a finite (or infinite) sequence of such actions. The requirements of the protocol are given as a set of *liveness* and *safety* properties using predicate logic formulae, specifying "allowed" traces.

Before we sketch the refinement of the specification and the resulting proof tasks, we give a list of operators working on traces (taken from [DW92a][4]; t, u, v are traces, a, b, c are actions):

- $t \cdot u$, $a \cdot t$ denotes concatenation of traces or actions with traces.
- $t \sqsubseteq u$ means "trace t is a prefix of u".
- $\#t$ denotes length of t. If the trace is infinite, $\#t = \infty$.
- $\langle a_0, \ldots, a_n \rangle$ denotes the trace consisting of actions $a_0, \ldots, a_n$.
- $t[k]$ denotes the k-th element of t (strong definition).
- $a \copyright t$ denotes the filtered trace t that contains only actions a. E.g., $Snd \copyright t$ results in a trace, containing *snd*-actions only.
- a *in* t holds exactly, if action a occurs in trace t.
- $\langle d, k \rangle$ denotes a pair, consisting of a piece of data d and an integer k.

[3] The set of all actions of a given kind is defined as $Snd := \{snd(d) | d \in UDU\}$. *Rec*, Get_R, Put_T are defined in a similar way. Furthermore, a *clock* action "$\surd$" is defined.

[4] This report also contains detailed definitions of these operators.

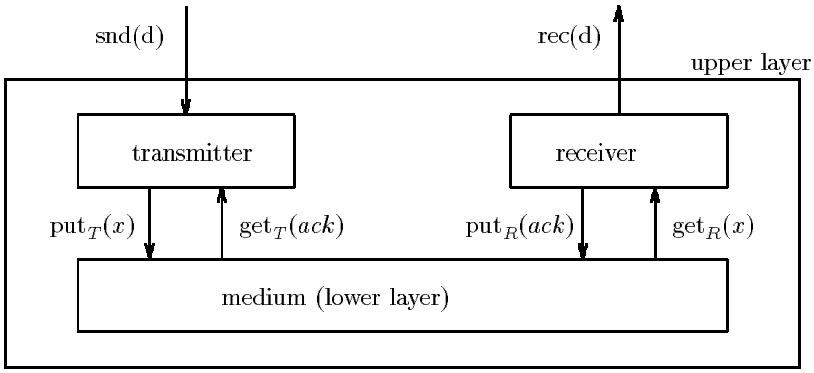

Fig. 1. The Stenning Protocol (Fig. 1 from [DW92a]), $d \in UDU, x \in LDU$

The FOCUS-specification of the Stenning protocol starts with services which are provided by the upper layer and the lower layer (see Fig. 1). Those provided on the upper layer ("reliable communication") are specified in the liveness property UL and the safety property US. All properties have been directly taken from [DW92a] and are shown for reference in Table 1 and Table 2. UL means that the number of packages (UDU's) sent is equal to the number of received packages, i.e., no package has been lost. US expresses that nothing "wrong" is received. The properties LL^1, LL^2 and LS^1, LS^2 specify the behavior of the transport medium (in both directions): $LS^{1,2}$ states that if a data package (LDU) is received on one side, it must have been put previously onto the medium on the other side. However, it does not say that any sent package must be also received, losses are possible. $LL^{1,2}$ specifies that the medium is not broken forever (if we have sent an infinite number of packages, then an infinite number of packages is received after an infinite number of clock-cycles).

The refinement of the specification with the goal of a modular requirements specification is performed in several steps. All requirements which are not local to the receiver or the transmitter, or which are not provided by the medium, have to be refined. In [DW92a] three steps are performed, namely

1. introduction of sequence numbers (Proposition 3.1)[5],
2. introduction of acknowledge messages (Proposition 3.2), and
3. complete localization (Proposition 3.3)

In $Step_1$, the introduction of sequence numbers (for each $d \in UDU$, we send a pair $\langle k, d \rangle$ where k is a natural number), two safety properties S_1, S_2 and three liveness properties are introduced: S_1 states that an UDU is given to the

[5] Each proposition (number in parenthesis from [DW92a]) shows that a given refinement is appropriate, i.e., that safety- and liveness properties are not violated.

$$
\begin{aligned}
UL(t) &\equiv \#(Rec©t) = \#(Snd©t)\\
LL^1(t) &\equiv (\#(\mathrm{put}_T(e)©t) = \infty \Rightarrow \#(\mathrm{get}_R(e)©t) = \infty) \wedge \#(\surd©t) = \infty\\
LL^2(t) &\equiv (\#(\mathrm{put}_R(e)©t) = \infty \Rightarrow \#(\mathrm{get}_T(e)©t) = \infty) \wedge \#(\surd©t) = \infty\\
L_1(t) &\equiv data(Snd©t) \sqsupseteq \langle d_0, \ldots, d_n\rangle \Rightarrow \forall k \leq n : \mathrm{put}_T(\langle k, d_k\rangle)\ in\ t\\
L_2(t) &\equiv \mathrm{put}_T(\langle k, d\rangle)\ in\ t \Rightarrow \mathrm{get}_R(\langle k, d\rangle)\ in\ t\\
L_3(t) &\equiv (\forall k \leq n : \mathrm{get}_R(\langle k, d_k\rangle)\ in\ t) \Rightarrow data(Rec©t) \sqsupseteq \langle d_0, \ldots, d_n\rangle\\
L_4(t) &\equiv \mathrm{put}_T(\langle k, d\rangle)\ in\ t \Rightarrow \mathrm{get}_T(ack(k))\ in\ t\\
L_5(t) &\equiv \#(Snd©t) > k \wedge \neg A(t,k) \wedge \#(\surd©t) = \infty \Rightarrow\\
&\qquad \exists j, d : \neg A(t,j) \wedge \#(\mathrm{put}_T(\langle j, d\rangle)©t) = \infty\\
&\qquad where A(t,k) \equiv \exists s : s \cdot \mathrm{get}_T(ack(k)) \sqsubseteq t \wedge \#(Snd©s) > k\\
L_6(t) &\equiv \#(\mathrm{get}_R(\langle k, d\rangle)©t) = \infty \Rightarrow \#(\mathrm{put}_R(ack(k))©t) = \infty
\end{aligned}
$$

Table 1. Liveness Properties

$$
\begin{aligned}
US(t) &\equiv \forall s \sqsubseteq t : data(Rec©s) \sqsubseteq data(Snd©s)\\
LS^1(t) &\equiv \forall s \sqsubseteq t : \mathrm{get}_R(e)\ in\ s \Rightarrow \mathrm{put}_T(e)\ in\ s\\
LS^2(t) &\equiv \forall s \sqsubseteq t : \mathrm{get}_T(e)\ in\ s \Rightarrow \mathrm{put}_R(e)\ in\ s\\
S_1(t) &\equiv \forall s \sqsubseteq t : data(Rec©s) = \langle d_0, \ldots, d_n\rangle \Rightarrow \forall k \leq n : \mathrm{get}_R(\langle k, d_k\rangle)\ in\ s\\
S_2(t) &\equiv \forall s \sqsubseteq t : \mathrm{put}_T(\langle k, d\rangle)\ in\ s \Rightarrow (Snd©s)[k] = snd(d)\\
S_3(t) &\equiv \forall s \sqsubseteq t : \mathrm{put}_R(ack(k))\ in\ s \Rightarrow \exists d : \mathrm{get}_R(\langle k, d\rangle)\ in\ s
\end{aligned}
$$

Table 2. Safety Properties

upper layer only (on the receiver side), if all messages with a smaller sequence number have already been received (get_R). S_1 is local to the receiver. S_2 (local to the transmitter) requires the transmitter to put packages onto the medium only (Put_T), if they contain the correct sequence number and have already been sent by the upper layer.

L_1 (local to transmitter) expresses that when an UDU has been sent, it will eventually be put onto the medium; whereas L_3 (local to receiver) describes that all data packages which are delivered by the medium (Get_R) are eventually received on the upper layer. L_2 states the liveness of the medium. i.e., all packages put onto the medium are eventually received. Although this requirement is local (to the medium), the medium "does not support it" [DW92a]. Therefore, L_2 must be further refined. Before doing so, it must be ensured that the properties just described plus the properties of the medium (LS^i, LL^i) allow for a reliable communication. Given

$$Step_1(t) \equiv LS^1(t) \wedge LS^2(t) \wedge LL^1(t) \wedge LL^2(t) \wedge S_1(t) \wedge S_2(t) \wedge L_1(t) \wedge L_2(t) \wedge L_3(t)$$

we must show: $Step_1(t) \Rightarrow US(t) \wedge UL(t)$. This comprises our first proof obliga-

tion (Proposition 3.1).

In the next refinement step ($Step_2$) acknowledge messages are introduced, replacing L_2 by S_3 and L_4: S_3 (local to the receiver) states that a message (with sequence number k) must have been received (Get_R), before the corresponding acknowledge message can be put on the medium. L_4 (not local, will be refined in $Step_3$) expresses that every piece of information put onto the medium by the transmitter will eventually get its corresponding acknowledge message.

Our second proof task (Prop. 3.2 in [DW92a]) is to show $Step_2 \Rightarrow Step_1$. Since only L_2 was replaced, it is sufficient to prove $Step_2(t) \Rightarrow L_2(t)$. In general, each refinement step is characterized by a conjunction of certain liveness and safety properties:

$$Step_i(t) \equiv \bigwedge_j L_j(t) \wedge \bigwedge_k S_k(t) \wedge LL^1(t) \wedge LL^2(t) \wedge LS^1(t) \wedge LS^2(t).$$

Then, the proof tasks have the form: $Step_i(t) \Rightarrow Step_{i-1}(t)$. For reference, Table 3 shows the definition of all steps.

$Step_3$ replaces L_4 by two additional liveness properties L_5 (local to transmitter), and L_6 (local to receiver)[6]. Then, all properties of $Step_3$ are either local to the transmitter, to the receiver, or to the medium. Furthermore, the requirements for the medium is supported by it. $Step_3$ comprises the goal of the refinement, a *modular* requirements specification.

All properties just described have been taken directly from [DW92a] and form the basis of the proof tasks, tackled with SETHEO. These tasks (in some cases, more than one proof task arises from one proposition, e.g., by splitting "$\Leftrightarrow$" into the "$\Leftarrow$" and the "$\Rightarrow$" case) are listed in Table 4. We furthermore processed two additional lemmata which have been defined and used in [DW92a].

$$\begin{aligned}
Step_1(t) &\equiv LS^{1,2}(t) \wedge LL^{1,2}(t) \wedge S_1(t) \wedge S_2(t) \wedge L_1(t) \wedge L_2(t) \wedge L_3(t) \\
Step_2(t) &\equiv LS^{1,2}(t) \wedge LL^{1,2}(t) \wedge S_1(t) \wedge S_2(t) \wedge S_3(t) \wedge L_1(t) \wedge L_3(t) \wedge L_4(t) \\
Step_3(t) &\equiv LS^{1,2}(t) \wedge LL^{1,2}(t) \wedge S_1(t) \wedge S_2(t) \wedge S_3(t) \wedge L_3(t) \wedge L_5(t) \wedge L_6(t)
\end{aligned}$$

Table 3. Definition of the refinement steps

[6] Description of L_5 and L_6 from [DW92a]: "Requirement L_5 is a little bit intricate. It might be the case that some acknowledge $ack(k)$ arrives at the transmitter before the k-th message has actually been sent. The formula $A(t,k)$ says that there has been a "proper" acknowledgement for the k-th message, i.e., one that did not occur before at least k messages have been sent to the transmitter. So L_5 says that if at least k UDU's are sent and the k-th UDU has not got a "proper" acknowledgement yet, then the transmitter will send some not "properly" acknowledged UDUs (not necessarily the k-th one) infinitely often. [...] Property L_6 expresses that if the receiver gets an information infinitely often, it will also send the corresponding acknowledgement infinitely often."

task 1	Prop. 3.1	$Step_1(t) \Rightarrow US(t)$
task 2	Prop. 3.1	$Step_1(t) \Rightarrow UL(t)$ (strong)
task 3	Prop. 3.2	$Step_2(t) \Rightarrow Step_1(t)$ (i.e. $L_2(t)$)
task 4	Prop. 3.2	task 3 without Lemma 3.4
task 5,6,7	Prop. 3.3	$Step_3(t) \Rightarrow Step_2(t)$ (i.e. $L_1(t) \wedge L_4(t)$)
task 8	Lemma 3.4	Lemma for Prop. 3.2
task 9,10	Lemma 3.5	Lemma for Prop. 3.3

Table 4. Proof tasks and their definitions

3 Formalization for SETHEO

A transformation of the given proof tasks into a representation suitable for SETHEO is accomplished in 4 steps:

1. Transformation into First Order Predicate Logic (FOL): all axioms, theorems, and lemmata must be represented in first order predicate logic.
2. Transformation of the notation: the entire formula must be represented in a syntax which is readable by SETHEO.
3. Axiomatization of the underlying theory: for all operators axioms describing all properties of that operator must be added.
4. Transformation into clause normal form (CNF): the formula must be transformed into a set of clauses and the quantifiers must be removed by Skolemization. For this task, we used a standard algorithm (see e.g., [Lov78]).

In the following, we will describe each of the steps in detail. Since all formulae in this case study already were available in First Order Logic, the first step is skipped.

3.1 Transformation of the Notation

The aim of this step is to transform all formulae (in the original notation) into a syntactical form which is readable by SETHEO.

The major part of this step involves the decision which operator (or function symbol) is represented as a predicate symbol, and which symbols are written as (syntactic) function symbols. Because of better readability, we have selected equality ($=$), and all relational binary operators (i.e., $\leq, in, >, \sqsubseteq$) to be represented as predicate symbols of arity 2 ("equational representation"). This means that an expression $a = b$ is written as `equal(a,b)`. All other symbols occurring in the formulae are transformed into pre-fix function symbols or syntactic constants. Table 5 gives the relation between the operators as used in [DW92a], and their representation for SETHEO as predicate and function symbols. For example, Property L_2 is written in SETHEO syntax as follows:

```
forall T forall K forall D
( in(putt(pair(K,D)),T) -> in(getr(pair(K,D)),T))
```

predicate symbols		function symbols	
in [DW92a]	SETHEO input	in [DW92a]	SETHEO input
$A \leq B$	`less(A,B)`	A @ B	`filt(A,B)`
A *in* B	`in(A,B)`	$get_R(A), get_T(A)$	`getr(A), gett(A)`
$A > B$	`gt(A,B)`	$put_R(A), put_T(A)$	`putr(A), putt(A)`
$A = B$	`equal(A,B)`	$\langle d_0, \ldots, d_n \rangle$	`data_sequence(T,N)`
$A \sqsubseteq B$	`ispre(A,B)`	$A \cdot B$	`cons(A,B)`
		$data(A)$	`data(A)`
		$\langle k, d \rangle$	`pair(K,D)`
		$snd(d)$	`snd_data(D)`
		Snd	`snd`
		Rec	`rec`
		$T[n]$	`nth(T,N)`
		∞	`inf`

Table 5. Transformation of notation: list of predicate and function symbols

3.2 Axiomatization

Finding the appropriate set of axioms for all operations and relations is an extremely difficult task for any application of a theorem prover. A number of important decisions must be made here: which *kind* of axiomatization is to be used for which *subset* of operations and relations, and *which* axioms (and lemmata) are to be added to the formula.

The aim of this study has been to *find* proofs for the given proof tasks, using SETHEO. Therefore, we started with a small set of "common" axioms (for equality) and added more axioms and primitive lemmata "by need"; lemmata of which it was thought that they might be helpful for the current proof task. Since these lemmata are quite obvious and directly follow from the definition of the operators, we will call them *high-level axioms*. However, these axioms needed for our proof tasks are often rather weak (e.g. $\forall t, a : \#(a \copyright t) = \infty \Rightarrow a$ *in* t, i.e., if an action occurs infinitely often in a trace, it occurs at least once in that trace). Those axioms are too weak to be useful in a general context.

As a result, we obtained a set of axioms (listed in Appendix A) which allows to prove the *given* set of proof tasks, but we do not have a full-fledged set of axioms, defining all properties of the operations and relations.

4 Experiments and Results

In this section, we will describe results of the SETHEO experiments made with the proof tasks `task 1` to `task 10`, after all steps, described in the previous section have been performed.

Although the SETHEO system allows to set a large variety of parameters (e.g., different ways of performing iterative deepening), default parameters have

been used for this case study[7]. This parameter setting results in an iterative deepening over the depth of the tableau (A-literal depth). Only in those cases, where SETHEO did not find the proof within a few seconds, two additional techniques have been used to tackle the problem: enabling the additional "fold-up" inference rule (see [LMG94]), and using the preprocessor DELTA. DELTA [Sch94a] generates selected unit clauses in a bottom-up way during the preprocessing phase. These are added to the formulae before the top-down search with SETHEO starts.

Table 6 gives an overview over the results of the experiments. Run-times are given in seconds and have been obtained on a sun sparc10. For each proof task, we give the number of clauses after transformation into clausal form, the run-time for SETHEO (V3.2) (and DELTA) in seconds, the necessary resources (fold-up, DELTA, or "–" for standard options), and the number of inferences (Model Elimination extension and reduction steps) of the proof.

This table, however, does *not* reflect the actual overall time (needed to find the axioms, to debug the formula, and to make preliminary experiments). The figures in this table just show that, given an appropriate set of axioms, SETHEO can find the requested (non-trivial) proofs automatically within short run-times.

An estimation of the time needed for the entire case study is difficult to give because the author did not log the sessions for this case study. In total, this work was completed within a few days. Major time-consuming tasks were (in decreasing order): setting up an environment ("infra-structure") to keep all formulae and results, keeping all files and formulae consistent, debugging formulae (find out why things went wrong and correct the formulae (mostly mis-prints)), finding and formulating the appropriate axioms, and checking the proofs for correctness (in order to detect possible flaws in the axioms or theorems). Below, these issues will be discussed in detail.

5 Experiences & Future Work

The entire case study could be carried out *without* a detailed knowledge of FOCUS and the problem domain (communication protocols in our case). For the axiomatization (see below), only a knowledge about the operators used in the proof tasks had been necessary.

In the following, we will summarize the experiences made during each step of formalization and execution of the proof tasks by SETHEO. Furthermore, we try to give hints how to approach such proof tasks in a methodical way. Many of these items are well known (see e.g. [ORSvH93]) (and solved) within interactive theorem proving environments. Automatic theorem provers, like SETHEO, however do not have facilities needed for a case study like this. Rather, ATPs

[7] The default parameters of SETHEO are: `inwasm -cons`, `wasm -opt`, and `sam -cons -dr`. They are used automatically, when the command `setheo` is called. For further details see e.g., [LSBB92, LMG94, GLMS94] or the manual pages of SETHEO.

proof task	# clauses	run-time [s]	resources	length of proof		
				total	ME-Ext	ME-Red
task 1	23	0.2	–	10	9	1
task 2	22	0.2	–	6	5	1
task 3	24	0.6	fold-up	18	18	0
task 4	23	7.7	DELTA	20	20	0
task 5	25	17.0	DELTA	20	20	0
task 6	40	8.0	–	20	20	0
task 7	37	0.4	–	6	6	0
task 8	24	0.2	–	11	11	0
task 9	26	0.1	–	3	3	0
task 10	24	0.1	–	10	10	0

Table 6. Experimental results (run-times of SETHEO on a sun sparc 10)

in general only provide an efficient search algorithm and assume that the enire (and error-free) formula is given to the prover.

Transformation into FOL. All formulae in this case study had already been given in First Order Predicate Logic. This has been extremely helpful, because in many cases, properties on the trace level cannot (or can not be easily) represented in pure FOL. Then, these properties are specified using Higher Order constructs. Typically for that are quantifiers ranging over predicate symbols (i.e., operations and relations), or induction.

Although we believe that many such constructs can be transformed (automatically or manually) to FOL formulae, much work will have to be done in that area.

Transformation into SETHEO Syntax. Essentially, this step is a straightforward syntactic transformation of symbols for operations and relations into predicate symbols and syntactic function symbols (e.g., $A©B$ into `filt(A,B)`). The selection of which operations are to be represented by predicate symbols and which are to be represented by function symbols have been made in a very simple, straight-forward way.

Nevertheless, this transformation step turned out to be very time-consuming and error prone. A lot of typing errors have been made (e.g., typing `file` instead of `filt`), errors which the SETHEO system does not detect (except by not finding a proof): mistyped symbols are just interpreted as different symbols. Since axioms are used in many proof tasks, it has been extremely difficult to remove such bugs and keep all formulae in a consistent state.

This problem, however, could be solved in the course of this case study (in a very primitive way) by keeping all axioms, theorems, lemmata, etc., in separate files and by using standard UNIX-tools to maintain syntactic consistency. In our case, we used the C-language preprocessor **cpp** to assemble the formula, **sccs**

to keep different versions of the axioms and theorems, and **make** to control the processing of the proof tasks. Further applications now can use this primitive, but rather helpful environment.

Axiomatization. The step of finding the right axioms and "high-level axioms" (small lemmata) for the actual proof task is probably the most difficult problem in applying an automated theorem prover to this kind of application. In our case study, the most naive approach has been taken: we used an equational axiomatization which represents $a = b$ as `equal(a,b)`. Besides standard axioms for equality (reflexivity, symmetry, transitivity, substitution axioms), axioms and lemmata ("high-level axioms") have been added where necessary. Here, emphasis was put on a set of axioms which was *sufficient* to find the proofs, not one which completely defines the operations and relations[8]. These high-level axioms describe obvious features of the operators (e.g., getting the n-th value of a trace) without the necessity to use the low-level definition of the operators. In FOCUS, operators are normally defined using recursive equations. Using these definitions would require induction for all (even the most trivial) proof obligations. Therefore, high-level axioms help to avoid simple cases of induction. This is also reflected in the original manual proofs in [DW92a] which do not require induction.

In general, however, the maintenance of a complete and consistent set of axioms is a hard piece of work which requires careful planning and development. Such a set of axioms is presumably much larger than that used in our case study. Its usage leads to a huge search space which can be handled only by (a) selecting an "appropriate" subset of axioms, and/or (b) running SETHEO with a fine-tuned set of parameters and search heuristics. The development of such techniques can only come from experience gathered from further case studies.

Running SETHEO. A main goal of this case study was to show that the *automated* theorem prover SETHEO is capable of tackling the given proof tasks. Therefore, only standard parameters for SETHEO have been used. If SETHEO could not find a proof easily, first the folding-up inference rule was activated, and, if that was not successful, the DELTA preprocessor was activated with a rule-of-thumb set of parameters. Although this has been done by hand in this case study, further experiments and case studies will certainly lead to hints (and possibly even heuristics) how the parameters of SETHEO must be set reasonably.

6 Conclusions

We have presented the results of experiments which applied the automated theorem prover SETHEO to given proof tasks, arising from the refinements of a specification carried through in FOCUS. This specification and its refinements

[8] Of course, care was taken that all axioms are consistent.

have been taken from a Technical Report [DW92a], dealing with the development of a (Stenning) communication protocol. We have taken all propositions and lemmata in Chapter 3 of that paper (a specification of the level of traces), and have been able to prove all of them fully automatic. All of the 10 proof tasks could be solved by SETHEO within several seconds on a sun sparc 10. The obtained results are surprisingly good and show that these proof tasks are of a size and complexity which can be handled automatically (given a proper set of axioms). These results can be seen as a first step of successfully using automated theorem provers for FOCUS.

A central goal of using an automated theorem prover like SETHEO during the development of specifications in FOCUS will be the *automatic* processing of *simple*, but frequently occurring proof tasks. Such proof tasks tend to be tedious when they are to be proven manually; nevertheless it is necessary to prove them for full verification. Complicated proof tasks (requiring elaborate proof strategies and ideas), on the other hand, will certainly have to be proven by hand in the near future. Although the proof tasks in our case study could be solved easily, several major (theoretical and practical) problems will have to be solved before that goal can be accomplished.

The setup of a complete and consistent set of axioms and lemmata, together with ways of preselecting axioms to reduce the search space, is a task which has to be tackled as a first step. Handling of proof tasks which are still quite simple from the human point of view, but reveal a more complex structure than the current one (e.g., proofs containing easy types of induction) will have to be studied next. We believe that in many cases of induction a small manual preprocessing (e.g., saying "induction on length of trace") is sufficient to prepare the problem in such a way that SETHEO can solve it.

From a more practical point of view, an environment ("infra structure") must be set up to handle proof tasks. Although default in most interactive provers, the following points should be looked at: a *formula editor* (together with a maintained data-base for formulae) would extremely facilitate the usage of SETHEO on such proof tasks. Furthermore, our *interactive* version of SETHEO will be of help in cases, where a proof cannot be found automatically (e.g., due to a missing axiom). Also, the generation of *counter examples* (e.g., by a model generator) for satisfiable theorems or during debugging of formulae would be of great interest. A transformation of SETHEO's proofs into a readable form (e.g., into Natural Deduction) is almost indispensable.

However, it should be emphasized that the time to build a *tool* for using Automated Theorem Proving for FOCUS has not yet come. The next steps will certainly be further case studies on the level of trace specifications and with other specification formalisms (relational, functional) which FOCUS provides. A careful evaluation of the results will lead to the development of methods and heuristics for the use of SETHEO for simple, but often occurring proof tasks during the development process carried out in FOCUS.

Acknowledgements. I would like to thank the members of subproject SFB 342-A6 for providing the proof tasks and their great interest in results of this

case study, Ketil Stølen for interesting discussions, and the anonymous referees for many helpful comments.

References

[BDD+93] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The design of distributed systems — an introduction to Focus (revised version). Technical Report SFB 342/2/92 A, Technische Universität München, 1993.

[DDW93] F. Dederichs, C. Dendorfer, and R. Weber. FOCUS: A Formal Design Method for Distributed Systems. In A. Bode and M. Dal Cin, editors, *Parallel Computer Architectures*, pages 190–202. Springer, 1993.

[DW92a] C. Dendorfer and R. Weber. Development and Implementation of a Communication Protocol – An exercise in FOCUS. SFB-bericht nr. 342/4/92 a, Technische Universität München, Institut für Informatik, 1992.

[DW92b] C. Dendorfer and R. Weber. Form service specification to protocol entity implementation – an exercise in formal protocol development. In R.J. Linn and M. Ü. Uyar, editors, *Protocol, Specification, Testing and Verification XII*, volume C-8 of *IFIP Transactions*, pages 163–177, 1992.

[GLMS94] Chr. Goller, R. Letz, K. Mayr, and J. Schumann. SETHEO V3.2: Recent Developments (System Abstract) . In *Proc. CADE 12*, pages 778–782, June 1994.

[LMG94] R. Letz, K. Mayr, and C. Goller. Controlled Integration of the Cut Rule into Connection T ableau Calculi. *Journal Automated Reasoning (JAR)*, (13):297–337, 1994.

[Lov78] D. W. Loveland. *Automated Theorem Proving: a Logical Basis*. North–Holland, 1978.

[LSBB92] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[ORSvH93] S. Owre, J. Rushby, N. Shankar, and N. von Henke. Formal Verification for Fault-tolerant Architectures: Some Lessons Learned. In *Proc. FME '93*, volume 710 of *LNCS*, pages 482–500. Springer, 1993.

[Sch94a] J. Schumann. DELTA — A Bottom-up Preprocessor for Top-Down Theorem Provers, System Abstract. In *CADE 12*, 1994.

[Sch94b] J. Schumann. Using SETHEO for verifying the development of a Communication Pro tocol in FOCUS – a case study –. SFB Bericht SFB342/20/94A, Technische Universität München, 1994. long version.

[Ste76] V. Stenning. A data transfer protocol. *Computer Networks*, 1:98–110, 1976.

A Axioms and Lemmata

$Ax_{1.1}$	$\equiv \forall x : x = x$	Reflexivity of "$=$"
$Ax_{1.2}$	$\equiv \forall x, y : x = y \Rightarrow y = x$	Symmetry of "$=$"
$Ax_{1.3}$	$\equiv \forall x, y, z : x = y \wedge y = z \Rightarrow x = z$	Transitivity of "$=$"
$Ax_{2.1}$	$\equiv \forall x, y, z : x \sqsubseteq y \wedge y = z \Rightarrow x \sqsubseteq z$	Subst. of "$\sqsubseteq$"
$Ax_{2.2}$	$\equiv \forall x, y : x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$	Symmetry of "$\sqsubseteq$"
$Ax_{3.1}$	$\equiv \forall x, y : snd(x) = snd(y) \Rightarrow x = y$	Monotony of snd
$Ax_{3.2}$	$\equiv \forall x, y : x = y \Rightarrow \mathrm{get}_R(x) = \mathrm{get}_R(y)$	Subst. for get_R
$Ax_{3.3}$	$\equiv \forall x, y, z : x = y \Rightarrow \langle z, x\rangle = \langle z, y\rangle$	Subst. for $\langle\rangle$
$Ax_{4.1}$	$\equiv \forall x, y, z : x \ in \ y \wedge y = z \Rightarrow x \ in \ z$	Subst. for in
$Ax_{5.1}$	$\equiv \forall x, y : x = y \Rightarrow \mathrm{put}_T(x) = \mathrm{put}_T(y)$	Subst. for put_T
$Ax_{6.1}$	$\equiv \forall x, k : data(x[k]) = (data(x))[k]$	
$Ax_{6.2}$	$\equiv \forall t, n, k, x, s :$ $x \sqsubseteq s \wedge k \leq n \Rightarrow \langle d_0, \ldots, d_n\rangle[k] = s[k]$	strong def. of "[]"
$Ax_{6.3}$	$\equiv \forall x, y : x = snd(y) \Rightarrow data(x) = y$	$data(snd(d)) = d$
$Ax_{7.1}$	$\equiv \forall s, k, d : (Snd@s)[k] = snd(d) \Rightarrow \#(Snd@s) > k$	strong def. of "[]"
$Ax_{8.1}$	$\equiv \forall s, x, t : s \cdot x \sqsubseteq t \Rightarrow x \ in \ t$	action in trace
$Ax_{9.1}$	$\equiv \forall t, a : \#(a@t) = \infty \Rightarrow a \ in \ t$	action in trace
$Ax_{10.1}$	$\equiv \forall t : t \sqsubseteq t$	reflexivity of "$\sqsubseteq$"
$Ax_{11.1}$	$\equiv \forall t, n : \langle d_0, \ldots, d_n\rangle \sqsubseteq data(Snd@t) \Rightarrow$ $\#(Snd@t) > n$	definition of trace
$Ax_{12.1}$	$\equiv \forall s, n, d, k : (k \leq n \Rightarrow (Snd@s)[k] = snd(d))$ $\Rightarrow \langle d_0, \ldots, d_n\rangle \sqsubseteq data(Snd@s)$	
$Ax_{13.1}$	$\equiv \forall x, y : x > y \Rightarrow y \leq x$	combine $>$ with $\leq$

B Lemmata from [DW92]

$Lemma_{3.4} \equiv \forall t, k, d, d' : \mathrm{put}_T(\langle k, d\rangle) \ in \ t \wedge \mathrm{get}_R(\langle k, d'\rangle) \ in \ t \Rightarrow d = d'$
$Lemma_{3.5} \equiv \forall t, k : A(t, k) \Leftrightarrow \mathrm{get}_T(ack(k)) \ in \ t$

C Theorems

C.1 Task 1: Proposition 3.1

$S_1 \wedge S_2 \wedge LS^1 \wedge LS^2 \wedge L_1 \wedge L_2 \wedge L_3 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge Ax_{10} \wedge Ax_{12}$
$\Rightarrow \forall s, t : s \sqsubseteq t \Rightarrow (data(Rec@s) = \langle d_0, \ldots, d_n\rangle \Rightarrow \langle d_0, \ldots, d_n\rangle \sqsubseteq data(Snd@s))$
Note: Here, we use the same notation of US as in [DW92a].

C.2 Task 2: Proposition 3.1

$S_1 \wedge S_2 \wedge LS^1 \wedge LS^2 \wedge L_1 \wedge L_2 \wedge L_3 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge Ax_{10} \wedge Ax_{12} \Rightarrow$
$\forall t, n : \langle d_0, \ldots, d_n\rangle \sqsubseteq data(Snd@t) \Rightarrow \langle d_0, \ldots, d_n\rangle \sqsubseteq data(Rec@t)$
Note: For this proof task, a stronger version of UL is shown.

C.3 Task 3: Proposition 3.2

$S_1 \wedge S_2 \wedge S_3 \wedge LS^1 \wedge LS^2 \wedge L_1 \wedge L_3 \wedge L_4 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge$
$Ax_{10} \wedge Ax_{12} \wedge Lemma_{3.4} \Rightarrow L_2$
Note: Task 4 uses the same axioms as Task 3, except that $Lemma_{3.4}$ is missing.

C.4 Task 5: Proposition 3.3

$S_1 \wedge S_2 \wedge S_3 \wedge LS^1 \wedge LS^2 \wedge L_3 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge Ax_5 \wedge Ax_6 \wedge$
$Ax_{10} \wedge Ax_{12} \wedge \forall t, n : \langle d_0, \ldots, d_n \rangle \sqsubseteq data(Snd@t)^\dagger$
$\Rightarrow \forall t, k, n : k \leq n \Rightarrow \mathrm{get}_T(ack(k))\ in\ t \Rightarrow \mathrm{put}_T(\langle\langle d_0, \ldots, d_n \rangle[k], k\rangle\ in\ t)$
Note: The theorem to be shown is part one of proposition 3.3. Assumption † corresponds to Assumption (*) in [DW92a].

C.5 Task 6: Proposition 3.3

$S_1 \wedge S_2 \wedge S_3 \wedge LS^1 \wedge LS^2 \wedge LL^1 \wedge LL^2 \wedge L_3 \wedge L_5 \wedge L_6 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge$
$Ax_4 \wedge Ax_5 \wedge Ax_7 \wedge Ax_9 \wedge Ax_{10} \wedge Ax_{11} \wedge Ax_{12} \wedge A \wedge Lemma_{3.5}$
$\Rightarrow \forall t, k, n : (\langle d_0, \ldots, d_n \rangle \sqsubseteq data(Snd@t)^\dagger \wedge$
$\neg k > n \wedge \neg \mathrm{get}_T(ack(k))\ in\ t \wedge \#(\surd\copyright t) = \infty) \Rightarrow \texttt{false}$
Note: This proof task comprises part two of proposition 3.3. It is shown, as in [DW92a] via contradiction. Assumption † corresponds to Assumption (*) in [DW92a].

C.6 Task 7: Proposition 3.3

$S_1 \wedge S_2 \wedge S_3 \wedge LS^1 \wedge LS^2 \wedge LL^1 \wedge LL^2 \wedge L_3 \wedge L_5 \wedge L_6 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge$
$Ax_4 \wedge Ax_5 \wedge Ax_7 \wedge Ax_9 \wedge Ax_{10} \wedge Ax_{11} \wedge Ax_{12} \wedge Ax_{13} \wedge A \wedge$
$(\forall t, k, n : k \leq n \Rightarrow \mathrm{get}_T(ack(k))\ in\ t)^\dagger \Rightarrow L_4$
Note: Formula † corresponds to the second part of proposition 3.3 (as shown in the previous proof task)

C.7 Task 8: Lemma 3.4

$S_1 \wedge S_2 \wedge S_3 \wedge LS^1 \wedge LS^2 \wedge L_1 \wedge L_3 \wedge L_4 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge$
$Ax_{10} \wedge Ax_{12} \Rightarrow Lemma_{3.4}$

C.8 Task 9: Lemma 3.5

$S_1 \wedge S_2 \wedge S_3 \wedge LS^1 \wedge LS^2 \wedge L_3 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge Ax_5 \wedge Ax_7 \wedge$
$Ax_8 \wedge Ax_{10} \wedge Ax_{12} \Rightarrow Lemma_{3.5}(\Rightarrow)$
Note: The proof task concerns the (easy) "$\Rightarrow$" direction of Lemma 3.5.

C.9 Task 10: Lemma 3.5

$S_1 \wedge S_2 \wedge S_3 \wedge LS^1 \wedge LS^2 \wedge L_3 \wedge Ax_1 \wedge Ax_2 \wedge Ax_3 \wedge Ax_4 \wedge$
$Ax_5 \wedge Ax_7 \wedge Ax_{10} \wedge Ax_{12} \Rightarrow Lemma_{3.5}(\Leftarrow)$
Note: The proof task concerns the (more difficult) "$\Leftarrow$" direction of Lemma 3.5. The predicate A is expanded.

This article was processed using the LaTeX macro package with LLNCS style