Interactive Theorem Proving – Modelling the User in the Proof Process *

Bernhard Beckert and Sarah Grebing
{beckert, sarah.grebing}@kit.edu

Karlsruhe Institute of Technology (KIT)

Abstract. Proving complex problems requires user interaction during proof construction. A major prerequisite for user interaction is that the user is able to understand the proof state in order to guide the prover in finding a proof.
Previous evaluations using focus groups for two interactive theorem provers have shown that there exists a gap between the user's model of the proof and the actual proof performed by the provers' strategies.
In this paper, we sketch a process model of the interactive proof process that helps to analyze this gap. Additionally, we give insight into the results of a usability test of the interactive verification System KeY, which provides evidence that this model is consistent with the actual proof process.

1 Introduction

Motivation. The degree of automation of interactive theorem provers (ITPs) has increased to a point where complex theorems over large formalisations for real-world problems can be proven effectively. But even with a high degree of automation, user interaction is still required on different levels. On a global level, users have to find the right formalisation and have to decompose the proof task by finding useful lemmas. On a local level, when automatic proof search for a lemma fails, they have to either direct proof search or understand why no proof can be constructed and fix the lemma or the underlying formalisation. As the degree of automation increases, the number of interactions decreases. But the remaining interactions get more and more complex as ITPs are applied to more and more complex problems.

We report on work in progress using the method of usability testing for several goals: (a) to gain insight into the interactive proof process using an ITP, (b) insight into problems in the interactive proof process and (c) insights for possible improvements. We carried out an experiment performing usability testing of the interactive verification system KeY [7]. In this paper we will briefly introduce a model of the interactive proof process, introduce our experiment and briefly give insights into the first results of the experiment, which relate to the

* The work presented here is part of the project Usability of Software Verification Systems within the BMBF-funded programme Software Campus.

proof process. In earlier work [6] we identified a gap between the user's model of the proof process and the actual proof process in the system. For illustrating the gap we developed a first informal model of the interactive proof process which we will extend in this paper.

In Section 2 we present related work of usability evaluations of interactive theorem provers and attempts to find a suitable model of the proof process. A first abstract model of the proof process follows in Section 3 and the gap between the user's and the prover's state is described in Section 4. The experiment and insights into the results are given in Section 5; and Section 6 concludes our work and shows future work.

2 Related Work

The usability of interactive theorem provers has been evaluated using various evaluation methods. Related work is concerned with usability evaluations of interactive theorem provers based on models defined prior to the evaluations. In addition, related work is also concerned with the derivation of models of the interactive proof process from evaluation results.

Merriam and Harrison [13] have evaluated interfaces of three theorem provers: CADiZ, IMPS and PVS. In this work they have identified four key activities in the interactive proof process where the user needs support from the proof system: planning, reuse, reflection and articulation. The three theorem provers have been examined with respect to these activities. Based on these results, gaps in user support of the theorem provers have been identified as well as points in the systems' interfaces where the user can make errors that cost him or her a lot of time to recover from.

Merriam [14] developed two approaches for the description of user activities in the proof process. He formalized a generic formal model of the proof using Z as formal language. This model is used to enable to gain insight into which kind of information is necessary for the user to conduct a proof effectively. Merriam assumes in this model that the user forms an opinion during the proof process about the provability of a proof goal using heuristics. He remarks that to model this assumption, a suitable cognitive model of the user is necessary. Interactions the user performs in the system are outside this model and are modelled in a second model of Merriam on the basis of Newman's Action cycle. Both models together were used to evaluate the PVS proof system.

Norbert Voelker [15] published a discussion paper on requirements and design issues of user interfaces for provers. He presented difficulties in the design of user interfaces of theorem provers developed in academia. In addition, a requirement analysis based on the scenarios using the scenario method has been carried out and resulted in a high-level description of the interaction with the proof system.

Aitken and Melham evaluated the interactive proof systems Isabelle and HOL using recordings of user interactions with the systems in collaboration with HCI experts. During the proof process the users were asked to think aloud and afterwards the users were interviewed. The authors goal was to study the activities

performed by users of interactive provers during the proof process to obtain an interaction model of the users. They propose to use typical user errors as usability metric and they compared provers w.r.t. these errors [3,4,2]. Also, suggestions for improvements of the systems have been made by the authors based on the evaluation results, including improved search mechanisms and improved access to certain proof relevant components.

The systems Isabelle and HOL have been evaluated by Aitken [1] using records of interactions. A semi-formal interaction model was extracted from the results, by identifying the actions that were performed during proof construction. Of the fifteen actions that have been identified, some relate to mental work of the users and some were direct actions in the system. All actions were modelled as activity diagram and it was distinguished between actions on the logical level and actions on the interaction level. In this work the relation between the problem class, the proof plan and the implementation is depicted.

In the work of Goguen [9] three user roles that can be represented by one single user have been identified: the prover, the reader and the specifier. Each of these roles has different requirements for the interactive proof system and some of the requirements can be conflicting. The authors claim that users of theorem provers need precise feedback on the failure of a proof attempt at the (sub)goal level. Further they argue that an unstructured proof tree is not easy to use as the users need to orient themselves in the proof tree. They present a proof approach where users should form the high-level proof plan and leave the "low-level computations" to the automatic prover. They implement their user interface for the proof assistance tool Kumo.

Similar to our findings in previous work, Archer and Heitmeyer [5] also realized the gap between the prover's and the user's model of the proof and have developed the TAME interface on top of the prover PVS to reduce the distance between manual proofs and proofs by automation. TAME is able to prove properties of timed automata using so called *human-style reasoning*. Proof steps in TAME are intended to be close to the large proof steps performed in manual proofs. The authors have developed strategies on top of the PVS strategies that correspond more to proof steps performed by humans. The goal is to provide evidence and comprehension of proofs for domain but not proof experts.

3 A Model of the System Consisting of User and Prover

In order to be able to describe the interactive proof process and to describe what influences the gap between the user and the prover states, a precise model of the proof process has to be developed. Our idea is to have an interactive proof system that consists of two main components that exchange information during the proof process: the user U and the prover P. We model both components as simple transition systems with three different transition functions: one that decides the next action for the user (f_{UDec}) resp. the next proof step for the prover (f_{PDec}), one that computes the next state of the user (f_{UCh}) resp. prover (f_{PCh}) according to the action/proof step and one function that computes the

next state of the user according to the prover's current state (f_{insp}) resp. the next state of the prover according to the action of the user ($f_{trigger}$).

Definition 1 (The Prover). *We model the prover as a transition system*

$$Prover = (P, PS, f_{trigger}, f_{PDec}, f_{PCh}, p_0, P_T) \ ,$$

where

- P *is a set of prover states*
- PS *is a set of actions which we call proof steps*
- $f_{trigger} : P \times A \to P$ *is a transition function*
- $f_{PCh} : P \times PS \to P$ *is a transition function*
- $f_{PDec} : P \to PS$ *is a choice function*
- $p_0 \in P$ *is the initial state*
- $P_T \subseteq P$ *is the set of terminating states*
- $P_{Proof} \subseteq P_T$ *is the set of terminating states in which a proof has been found*

Definition 2 (The User). *We model the user as a transition system*

$$User = (U, A, f_{insp}, f_{UDec}, f_{UCh}, u_0, U_T) \ ,$$

where

- U *is a set of user states*
- $A = (A_{proc} \cup A_{man})$ *is a set of actions, being the union of the proof manipulating and the process-oriented actions*
- $stopProcess \in A_{proc}$ *is the action to stop the proof process*
- $f_{insp} : U \times P \to U$ *is a transition function*
- $f_{UCh} : U \times A \to U$ *is a transition function*
- $f_{UDec} : U \to A$ *is a choice function*
- $u_0 \in U$ *is the initial state*
- $U_T \subseteq U$ *is the set of terminating states*

Definitions 1 and 2 depend on each other, as Definition 2 uses a component of Definition 1 (namely P) and vice versa (namely A). Both definitions could be combined to one system definition, but for simplicity we have two definitions, one for each component.

A prover's state P includes a partial proof (tree). We assume that the user states U at least consist of a mental model of the provers' state. We do not characterize this model in full detail, as we believe it is different for every user and depends on the experience with the system and mathematical background knowledge. Determining this model in full detail goes beyond the scope of our work.

In our model we focus on the interaction between the user and the prover. We further assume that the user's model of the prover's state is more abstract than the actual prover's state. We believe the user has a proof plan, that is formed

when developing the proof obligation. We consider that plan to be encoded in the user's state. Furthermore, we assume that the user has an idea about the effect of performing actions on the prover's state. This knowledge is included in the function f_{UCh}, as the user calculates a successor state from the current state and the action that the user performs in the system. As the successor state of the user also includes an abstraction of the prover's state, the user updates this model according to the expectations of the effect of performing the action.

Actions of the user can be of two kinds: proof manipulating (A_{man}) (e.g., applying a single proof rule or invoking a specific automatic strategy in the prover) and process-oriented actions (A_{proc}) (e.g., inspecting the prover's state further or stop the proof process (*stopProcess*)).

A proof step in the prover is an application of a calculus rule onto the current proof state.

Terminating states in the prover's model can be of two kinds: either a state in which a proof is found, i.e., $t_{Proof} \in P_T$ or states in which the automatic strategies stops, i.e., $p_t \in P_T$. These terminating states mark the beginning of the user interacting with the prover.

Interaction between both, the user and the prover, involves an information exchange. This exchange happens through the functions $f_{trigger}$ (user to prover) and f_{insp} (prover to user). The function f_{insp} involves the user inspecting the proof state of the prover (which can be either $p_t \in P_T$ in case the automatic strategies stop or the initial state p_0 in case the proof process is at the beginning) and updating the user's model by changing the state. This update may involve changing the proof plan by concretizing proof states in the plan or changing the mental model of the proof state by refining states.

The function $f_{trigger}$ represents the state change in the prover when a user action involves input to the prover and corresponds to the user invoking an action. This action is then accepted by the prover and translated by the prover into the strategy that should be used. The strategy of the prover is responsible for choosing the next proof step that should be applied to the prover's state. We model this by a state change performed using the prover's decision function (f_{PDec}).

The user also has such a decision function, which we call f_{UDec}. This function decides which action the user will perform next, depending on the user's state.

In our model of the user the functions f_{insp} and f_{UDec} follow each other. After the user has made his or her decision, the corresponding action is performed and function $f_{trigger}$ is applied in case the user decides to invoke the prover's strategies. Function f_{PDec} follows $f_{trigger}$ and then a sequence of function applications of f_{PCh} apply until a terminating state is reached.

In the following, we will define the interaction between the user and the prover in the interactive proof system.

Definition 3 (The Interactive Proof Process in an Interactive Proof System).

We model the interactive proof system consisting of a user U and prover P as a transition system. The state space S of the interactive proof system is the

set of triples

$$S = U \times P \times \{automode, interactive, inspected, decided, fail, success\} \ .$$

The initial state is $s_0 = (u_0, p_0, interactive)$*, where* p_0 *and* u_0 *are the initial states of the user* U *resp. the prover* P*.*

For all states $s \in S$*, the successor state* s' *of* s *is defined as follows, where* $a = f_{UDec}(u)$*:*

(a)	*if* $s = (u, p, interactive)$	*then* $s' = (f_{insp}(u, p), p, inspected)$
(b)	*if* $s = (u, p, inspected)$	*then* $s' = (u, p, decided)$
(c)	*if* $s = (u, p, decided)$ *and* $a = stopProcess$	*then* $s' = (f_{UCh}(u, a), p, userStop)$
(d)	*if* $s = (u, p, decided)$ *and* $a \in A_{man}$	*then* $s' = (f_{UCh}(u, a), f_{trigger}(p, a), auto)$
(e)	*if* $s = (u, p, decided)$ *and* $a \in A_{proc} \setminus \{stopProcess\}$	*then* $s' = (f_{insp}(u, p), p, inspected)$
(f)	*if* $s = (u, p, auto)$ *and* $p \notin P_T$	*then* $s' = (u, f_{PCh}(p, f_{PDec}(p)), auto)$
(g)	*if* $s = (u, p, auto)$ *and* $p \in P_T$	*then* $s' = (u, p, interactive)$
(h)	*if* $s = (u, p, userStop)$ *and* $p \notin P_{Proof}$	*then* $s' = (u, p, fail)$
(i)	*if* $s = (u, p, userStop)$ *and* $p \in P_{Proof}$	*then* $s' = (u, p, success)$

For states of the form $s = (u, p, success)$ *and* $s = (u, p, fail)$*, the successor state* s' *is undefined. They do not have a successor state.*

In the following, we will give a brief description of the Definition 3. In addition we have depicted the interactive proof process in Figure 1.

(a) If the system is in the state *interactive*, the user inspects the proof state and updates the own state using the information gained from the inspection $f_{insp}(u, p)$ (e.g., at the beginning of the proof process after the user has formulated the proof obligation and the system has translated it into the prover's representation, i.e., $s = (u_0, p_0, interactive)$ or after the prover has reached a terminating state, i.e., $s = (u_0, p_0, interactive)$)

(b) If the system is in the state *inspected*, the user makes a decision about the next action in the process according to the updated own model of the proof state. The action is decided by the internal choice function $f_{UDec}(u)$, (e.g., when the user has inspected the formula in the proof obligation and now determines what to do next in the proof process)

(c) If the system is in the state *decided* and the user has chosen the process-oriented action *stopProcess*, the user has decided to stop the proof process (e.g., when the user discovers a mistake in the specification or the program)

(d) If the system is in the state *decided*, and the user has chosen a proof manipulating action the user interacts with the prover (e.g., the user has inspected the formula in the proof obligation and encounters that a quantifier instantiation has to be performed or the induction rule has to be applied). The function $f_{trigger}$ translates the user's actions into the prover's strategies.

(e) If the system is in the state *decided*, and the user has chosen a process-oriented action (except *stopProcess*) the user inspects the proof state further (e.g., the user wants to inspect the proof tree in more detail).

(f) If the system is in the state *auto*, the prover applies a proof step according to the provers internal choice function resp. strategies until the automatic strategies can not apply more rules and therefore a terminating state is reached (e.g., the user has invoked the automatic strategies of the prover and the prover applies consecutive proof rules. Each rule application corresponds to one state transition.)

(g) If the system is in the state *auto* and the prover has reached a terminating state after proof step application, the user now can interact with the prover (e.g., the prover's strategies cannot apply proof rules anymore and presents the remaining proof obligation to the user)

(h) + (i) If the system is in the state *userStop*, the user decides whether the proof process was successful respec. failed (e.g., the user has either found a proof or discovered a mistake in the formalization and decides to terminate the proof process)

We model that while the prover's strategies apply proof steps (so state transitions in the prover's model according to f_{PCh} are made), the user can not interact with the prover until it reaches a terminating state p_t.

In our model the user's state also consists of a proof plan that the user formed when formulating the proof obligation. The *(partial) proof plan* of the user is a sequence of abstract proof states, denoted by $abs(P)$, related to each other by actions. These actions can be identical to the actions defined in the user's model and abstractions of the proof steps of the prover. We assume that this proof plan of the user consists of abstract proof states – they may either be identical to some prover states, possibly with intermediate prover states in between the abstract states of the user's proof plan. Some abstract states in the plan may also correspond to a sequence of prover states and, which are summarized as one abstract state.

The user might not always have a clear proof plan, e.g., at the beginning of the proof process. In this case, the user may consider several actions that he or

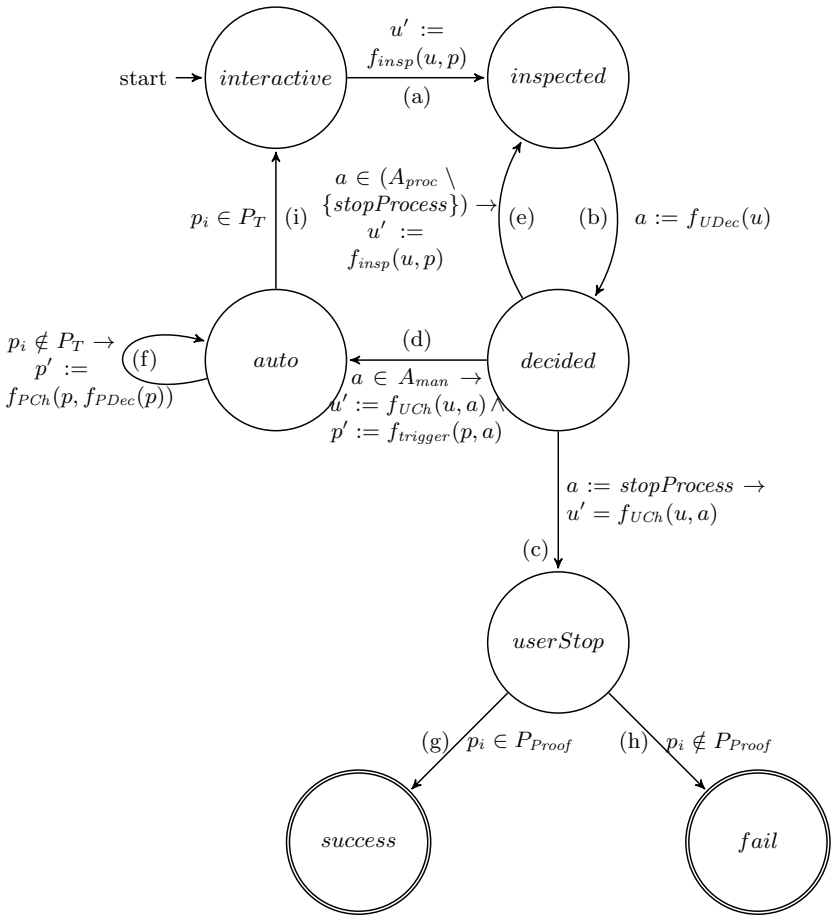

Fig. 1. Model of the interactive proof process ((a)-(i) are references to Definition 3)

she deems worthwhile to pursue, and for each of these actions he or she likely only has a rough idea of the resulting proof state. Of course, in certain situations, this set of possible actions to continue a proof is empty, as the user is unable to come up with a proof plan.

4 The Gap in the Proof Process

In former evaluations we have already identified a gap between the user's abstract states of the proof process and the concrete state of the prover. Based on this gap we have identified three major challenges an interactive theorem prover has

to meet in order to be more usable: (a) keeping the gap small, (b) bridging the gap and (c) allowing for effective interactions [6].

Here, we will now give a more precise description of the gap between the users abstract states and the provers states using a proof system with an explicit proof object, a proof tree as proof state and a sequent calculus as underlying proof calculus as an example. The following two problems can occur in such a system:

Provers Strategies applied too many Proof Rules To determine which next action to apply (modelled by the function f_{UDec}) the user has to inspect the proof state and update the own state according to the information gained by inspecting the prover's state (f_{insp}).

In this situation a gap between the users model of the proof state and the prover's state can occur when the automatic strategies applied too many proof steps or the proof steps were too "complicated". The user has to inspect the provers state and update the own model according to the information gained in the inspection. However, for this update the user has to find a correspondence for the current prover's state p_t in the own model too. As described above, p_t is a terminating state after the application of several proof steps decided by the strategy. If the user often iterates between the states *decided* and *inspected*, this may be a sign of a gap caused by the prover's strategies. Here the user needs a lot of time to find a correspondence between the own model of the prover's state and the current prover's state.

User Expectations Not Met. Another possibility for a gap is that the user performed the proof according to the proof plan he or she has made before the proof process and at a terminating prover's state p_t the prover's state does not correspond to the expectations of the user (it does not correspond to the state in the user's plan). The user has to inspect the prover's state further in order to determine whether he or she has made a mistake in the proof plan or the proof steps in the proof plan have been to abstract and have to be concretized by the inspection process.

If the user only has a partial proof plan, a gap can occur during the proof process when the difference between the prover's current state and the last state in the user's proof plan is large and the user is not able to relate the states to each other anymore. In this case the user has to inspect the proof state in order to retrace the proof steps the prover's strategies have applied and update the own state according to the gathered information.

The user has expectations about the effect of his or her actions on the proof state. If such an expectation is not met by the prover's strategies, a gap may occur as well. The user now has to try to understand what the effect of the performed action was by closely inspecting the proof state.

To summarize, we assume that the gap occurs at the point in the proof process where the prover reaches a terminating state p_t and the user applies function f_{insp} in order to apply function f_{UDec}. A hint that a gap has occurred can be, when the user needs a lot of time for the inspection process. In this case the loop between the states *inspected* and *decided* is traversed several times.

5 Insights into the Usability Test of the KeY system

To gain insights into the interactive proof process and to find evidence that our model is consistent with the proof process we conducted a usability test. Based on earlier results of two focus group discussions we conducted a formative, explorative usability test for the KeY system as the target of evaluation. Usability tests are structured interviews guided by a moderator following a script, which consists of all tasks and questions in the order they should be posed. While the participants perform the tasks, they should use the "thinking-aloud" technique. In addition their actions on the screen are recorded. The recorded data is then transcribed and anonymized. Later on, a qualitative content analysis [12,11,10] is performed to evaluate the test results.

In the following, we will first briefly describe the target of evaluation, give details about the usability test sessions and give insights into first observations and first analysis results.

The Target of Evaluation: KeY. The KeY system is an interactive verification system for programs written in Java and specified using the Java Modeling Language (JML). As such it is mostly used for the verification of Java programs w.r.t. a formal specification (usually a functional specification but also, e.g., information-flow properties). KeY has an explicit proof object, i.e., all intermediate proof states can be inspected by the user. The underlying calculus is a sequent calculus for Java Dynamic Logic [8]. Its user interface represents proofs as a tree. The nodes of the tree are intermediate proof goals (i.e., sequents). Each node is annotated with the rule that was applied to some formula in its direct parent node that lead to the current node.

The Participants. Nine KeY users took part in our usability tests, either intermediate or expert users. We excluded novice users, as our hypothesis was that advanced users perform more complex or larger proofs than novice users and therefore suffer more from efficiency problems in the proof process.

The Usability Test. Our goal of performing the usability test was (a) to gain insight into the proof process using the KeY system and (b) to determine whether a new mechanism, prototypically introduced into KeY, helps the user in bridging the gap between the concrete proof state and the model of the proof. We also wanted to gain information about further room for improvement of the target of evaluation. We planned a session time of approximately 70 minutes.

We structured the usability test into different phases[1]: introduction, warm-up, task and cool-down phase. In the *introduction-phase* the users were interviewed by the moderator about their experiences using the KeY system. The *warm-up phase* started with an interview about the proof process of the participants using the KeY system. Then the participants were asked to specify and

[1] The testing script can be found at `http://formal.iti.kit.edu/~grebing/SWC/` in German.

verify a Java method within the time frame of 10-15 minutes. We did not restrict the usage of system features in the warm-up phase. Our intention for this phase was to get insight into how the user uses the system to find a proof.

Based on earlier focus group discussions we prototypically implemented a mechanism to support the display of the history of a formula in the KeY system: It allowed the user to select a formula in the open goal and retrieve the path from the open goal to the original proof obligation in which the formula was affected by rule applications, in the following also called *history of a formula*. This mechanism should help to bridge the gap between the user's model of the proof and the current proof, as the user is able to trace back the history of a selected formula and see the changes during the proof process.

For the *task phase* we developed tasks that should help to evaluate the mechanism. We divided this phase into two parts with two different tasks each, one with and one without the new mechanism. One of the two different task types involved showing the user a partial proof for a proof obligation in first-order logic, obfuscating the predicate and function symbol names. The second task type involved a partial proof for the correctness of a method contract of a Java method.

For both types of tasks and both parts of the task phase the questions were identical: the user should describe the proof situation, they should name the history of two formulas of the open goal and name the next step to continue the proof process. At the end of the task phase the users were asked about their expectations about parent formulas of a given formula and proof.

In the *cool-down* phase participants were interviewed again about the new mechanism and generally about room for improvement in the system.

Insights into the results of the Usability Test. As the analysis is still work in progress we only give insights into the results of the warm-up phase and not a full analysis[2].

Almost all testing sessions have taken longer than we planned beforehand. Solving a task or answering the interview questions took longer than anticipated, as we didn't want to interrupt the participants.

In the warm-up phase we wanted to see how the participants use the KeY system to solve the task in order to gain insights into the proof process. Before the task, we wanted to know detailed information about the expectations of the users in a certain proof situation and about the proof process in general.

The interview questions have been:

1. Please imagine you are sitting in front of the KeY system and the automatic strategies stop with a lot of open goals/proof branches and quite a large sequent formula. What could have happened? What could have been reasons that KeY opened a lot of proof branches and was not able to close them? (In addition a screenshot of a proof with open goals in the KeY system has been used as stimulus)

[2] The sessions have been conducted in German, as it was the native language of the users. We translate the tasks and answers to the best of our knowledge.

2. How do you solve the problem of determining what has happened and what the next steps are?
3. Which possibilities do you have for that? Please arrange them in the order relatively to each other how often you use the possibilities.
4. Are there other alternatives in this situation, or are you missing a mechanism which is better suitable than the ones implemented in the system?
5. If you could wish for a functionality that could support you in proving using the KeY system, which one would it be?

The practical task for the users in the warm-up consisted of proving a method that removes the k-th element of a given array and returns the rest of the array, given the following task description:

> Please verify that the method fulfills its contract. Please conduct the proof like you are used to do it. Please complete or add something to the specification or the program if necessary. Please think-aloud what you are searching for and please explain before you click why you would like to click on that element on the screen.

The first specification of the method which we provided did not formalize the requirements we described to the user but was already a partial contract.

Our Intentions for the Tasks and Questions. Our intention behind these questions and tasks in the warm-up phase was to gain insight into how users use the KeY system to conduct a proof. In the practical task we intentionally did not show the method and its specification from the beginning on. We wanted to see which user directly proceeds to use the system to find out whether the method meets its specification and which user requests for the specification from the beginning on. In the first case, if the user found a proof, and if task time was not too far advanced, we asked the user whether the specification is an adequate rendition of the requirements.

First Results of the Usability Test. In the following, we will briefly mention those observations from the warm-up phase that contribute to our model. Our observation was that the users try to abstract from the concrete proof tree to gain an overview over the proof by using a feature of the KeY system that hides all intermediate proof states after using the automatic strategy. Almost all participants either used this feature in the practical tasks or mentioned the usage of this feature for proof inspection in the answer to the second question. This relates to the user's having or trying to build an abstract model of the prover's state.

When determining whether a proof is closeable, some users first tried the provers strategies again, as they assume the amount of user defined proof steps is not sufficiently high, before inspecting the proof tree in detail. In this case, we assume the users to have an expectation about the prover's strategies.

There were also users who noted that they would prune the proof tree when the strategies "went too far". They would prune the tree at a proof node from which they know its meaning, e.g., after finishing the symbolic execution and would then "apply rules in a controlled way." Here we have a hint for the gap, when the strategies of the prover apply too many rules and open too many goals without closing them again, the user goes back to a state from which he has a model.

At least one participant noted that he or she always switches from local to global proving, i.e. the participant first has an idea on the global, more abstract level how a proof should be performed, and during the proof process, when the automatic strategies are not able to close all goals he or she switches to inspecting the proof in detail on the local level and therefore inspecting the sequent in the proof node more closely. When a proof branch is closed, the user switches back to the global level and tries to close the next open goals. This indicates that the user has different abstraction layers of the proof.

6 Discussion and Future Work

We have presented a model of the components involved in an interactive proof process and briefly described a usability test of the KeY system to gain insights into the interactive proof process and to find evidence that the proposed model is consistent with the actual proof process. We described the point in the proof process where a gap between the user's model and the prover's actual proof state can occur using our model. We are aware that it is not possible to find evidence for all parts of the model, as some parts, such as how the precise user state can be characterized cannot be assessed by the "thinking-aloud" technique. Participants do not always verbalize everything they are thinking.

Our model does not yet include how users form a proof plan. This is a research field of its own and it remains for future work to include results of this research field into our model. We did not consider different user types and their special requirements on the prover yet, but we are confident that it is possible to include this in our model as well. The model of the proof process has to be enhanced, as it does not capture yet that the user and the prover are parallelized. It is not yet captured that the user can make decisions while the prover searches for a proof, as well as the user is able to interrupt the proof process. The role of the user interface is not yet captured by the model. We assume it can be modelled as a filter function for the prover's state p, which only shows parts of the prover's state to the user and the user only inspects this filtered state in the function f_{insp}.

As the evaluation of the usability tests is work in progress a full analysis and evaluation of the results is ongoing work.

Acknowledgements. We thank the participants in our focus group discussions on the usability of KeY and of Isabelle and our participants of the usability tests of the KeY system. In particular, we also thank the three moderators for their

great work. In addition, we thank our project partners from DATEV eG for sharing their expertise in how to prepare and analyse focus group discussions.

References

1. J. S. Aitken. Problem solving in interactive proof: A knowledge-modelling approach. In *Proceedings of the European Conference on Artificial Intelligence 1996 (ECAI96): 335-339, Edited by W. Wahlster*, pages 335–339, 1996.
2. J. S. Aitken, P. Gray, T. Melham, and M. Thomas. Interactive theorem proving: An empirical study of user activity. *J. of Symbolic Comp.*, 25(2):263–284, 1998.
3. J. S. Aitken and T. F. Melham. An analysis of errors in interactive proof attempts. *Interacting with Computers*, 12(6):565–586, 2000.
4. S. Aitken, P. Gray, T. Melham, and M. Thomas. A study of user activity in interactive theorem proving. In *Task Centred Approaches To Interface Design*, pages 195–218. Dept. of Computing Science, 1995. GIST Technical Report G95.2.
5. M. Archer and C. Heitmeyer. Human-style theorem proving using PVS. In *Theorem Proving in Higher Order Logics*, LNCS 1275. Springer, 1997.
6. B. Beckert, S. Grebing, and F. Böhl. A usability evaluation of interactive theorem provers using focus groups. In *Software Engineering and Formal Methods – SEFM 2014 Collocated Workshops*, Lecture Notes in Computer Science. Springer, 2014.
7. B. Beckert, R. Hähnle, and P. H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach.* LNCS 4334. Springer-Verlag, 2007.
8. B. Beckert, V. Klebanov, and S. Schlager. Dynamic logic. In Beckert et al. [7], chapter 3, pages 69–175.
9. J. Goguen. Social and semiotic analyses for theorem prover user interface design. *Formal Aspects of Computing*, 11:11–272, 1999.
10. U. Kuckartz. *Qualitative Inhaltsanalyse. Methoden, Praxis, Computerunterstützung.* Weinheim und Basel: Beltz Juventa, 2014.
11. P. Mayring. *Einführung in die qualitative Sozialforschung – Eine Anleitung zu qualitativem Denken (Introduction to qualitative social research).* Weinheim: Psychologie Verlags Union, 1996.
12. P. Mayring. Qualitative content analysis. *Forum : Qualitative Social Research*, 1(2), June 2000. Online Journal, 1(2). Available at: `http://qualitative-research.net/fqs/fqs-e/2-00inhalt-e.htm` [Date of access: 04, 2014].
13. N. Merriam and M. Harrison. Evaluating the interfaces of three theorem proving assistants. In F. Bodart and J. Vanderdonckt, editors, *Design, Specification and Verification of Interactive Systems '96*, Eurographics, pages 330–346. Springer Vienna, 1996.
14. N. A. Merriam. Two modelling approaches applied to user interfaces to theorem proving assistants. In *Proceedings of the 2nd International Workshop on User Interface Design for Theorem Proving Systems.*, pages 75–82. Department of Computer Science, University of York, 1996.
15. N. Völker. Thoughts on requirements and design issues of user interfaces for proof assistants. *Electron. Notes Theor. Comput. Sci.*, 103:139–159, Nov. 2004.