ISAC
User Requirements Document
Software Requirements Document

The ISACTeam

November 20, 2002
Contents

1 User Requirements Document 2
  1.1 Kinds of ISAC users . . . . . . . . . . . . . . . . . . . . . . . . 2
  1.2 General requirements . . . . . . . . . . . . . . . . . . . . . . . 3
  1.3 Requirements of the visitor . . . . . . . . . . . . . . . . . . . . 4
  1.4 Requirements of the learner . . . . . . . . . . . . . . . . . . . . 5
  1.5 Requirements of the math author . . . . . . . . . . . . . . . . 7
  1.6 Requirements of the dialog author . . . . . . . . . . . . . . . . 8
  1.7 Requirements of the course admin . . . . . . . . . . . . . . . . 9
  1.8 Requirements of the course designer . . . . . . . . . . . . . . . 11
  1.9 Requirements of the administrator . . . . . . . . . . . . . . . . 13

2 Software Requirements Document 14

A ISAC’s tactics 16

A Abbreviations 18
Chapter 1

User Requirements Document

This document describes the user requirements for the ISAC-system. This version of the document captures at least those requirements in detail, which are a prerequisite for the first phase of development, i.e. the development of the kernel of the math knowledgebase, of the indispensable tools for interaction on the knowledge, and of the tools for authoring the example collection.

The design will try to meet these requirements while accepting the structure of the ISAC-math-engine, which is defined by mathematical reasons, and offers new functionality (calculations are done stepwise, the learner can input a formula or a tactic and receive feedback from the engine, the math-engine usually 'knows' the next step).

As new and different the functionality of the envisaged system is, as limited is the functionality w.r.t the expectations a user can demand from (educational) math software at the state of the art. Thus from the beginning ISAC is seen as part of a collection of cooperating modules, and some requirements are stated here with the purpose, to provide for an architecture ready for combining ISAC with other components (administration, graphing etc.), and not with the purpose, to implement everything within ISAC.

The User Requirements Document is structured along the different kinds of users envisaged. The Software Requirements Document, on the other hand, will be structured along the modules implementing the functionality. In order to establish comfortable tracing, the $m:n$ relation between user requirements and software requirements will be accurately recorded in a double-linked way.

1.1 Kinds of ISAC users

There are several kinds of ISAC users which set the respective requirements:
**visitor (Besucher):** occasionally drops into an ISAC-site and browses the respective math knowledge base and the example collection. May try to calculate some examples.

**learner (Lernender):** uses ISAC for learning and exercising, i.e. primarily calculates examples in the example collection by use of the math knowledge base. As a member of courses the learner is called a student.

**math author (Mathematik-Autor):** is an expert in computer mathematics who adapts and extends the mathematics knowledge base.

**dialog author (Dialog-Autor):** is an expert in learning theory who adapts and extends the dialog guide.

**course admin (Kurs-Administrator):** is a person administering the use of ISAC for learning within a group of learners.

**course designer (Kurs-Designer):** adapts and extends the example collection which can be solved by a given math knowledge base, and adds explanations to items in the knowledge base. These tasks do not require special knowledge in computer mathematics.

**administrator (Administrator):** this role combines the system administrator installing the software, and the person who implements the overall design of an ISAC-site (introductory pages etc.) on behalf of the site-owner.

### 1.2 General requirements

This section describes the primary decisions for underlying systems, and requirements common to all users.

**Decisions for underlying systems** have been done already. As this requirements document shows, ISAC is a large system: thus it cannot be done from scratch, rather is has to use as much as components already available. The theorem prover Isabelle provides for both, a comprehensive frontend for interactive deduction (i.e. for definitions, axioms and proving theorems), and a huge mathematics knowledge base. Both are under rapid development, which ISAC shall take advantage of. The kernel of Isabelle, however is relatively stable already, and thus the interfaces to ISAC’s math engine (ME) are sufficiently stable.

**UR 1.2.1 The deductive part is left to Isabelle.**

**UR 1.2.2 ISAC's math engine closely cooperates with Isabelle**

Thus the math engine is already implemented in the same program language as Isabelle, SML.
The steps of calculation to be done in interaction with the ME are already specified. The operations propagating a calculation are called tactics. The tactics specified so far are listed in appendix A.  

**UR 1.2.3 Tactics as listed in appendix A**

**UR 1.2.4 The users access ISAC via internet**

This raises specific issues, because handling of mathematics formulas on standard browsers is still under construction, and several other relevant standards are under construction.

**Requirements for all users:** There are two different ways for users to approach ISAC's facilities for learning: (b) the user may browse the knowledge base, and eventually calculate an example (illustrating a definition, a problem, etc.) and (a) the user calculates examples from the example collection, and while asking for justifications enters knowledge browsers. The latter case implements the really innovative didactic approach of ISAC, 'learning by doing (calculations)'.

**UR 1.2.5 Access a: from example to knowledge, acces b: from knowledge to example to knowledge — are both possible.**

**UR 1.2.6 Consistent “look & feel” for all users.**

As long as the deductive part is left to Isabelle, the same is with the theory browser. ISAC will develop the “look & feel” of its own, and thus violate uniformity w.r.t. the theory browser.

### 1.3 Requirements of the visitor

Each user approaching an ISAC-site first time wants to know about the purpose of ISAC and about the contents of the site.

**Browsers: overview – detail:** The contents of the knowledgebase and of the example collection are expected to become very large. Thus there need to be facilities to switch from an overview to a detail and vice versa. These facilities should be handled similarly for all the browsers. Primarily, there is no direct interaction between the visitor and the ME. Informations for the visitor should be provided in statical HTML-Pages.

Visitors should get an overview over all the knowledge available at an ISAC-site, and all the explanations, and all the examples. Therefore, access to additional information should be gained if available.

---

1Eventually it may turn out, that some special tactics are missing and have to be added.
There are 4 kinds of data to be browsed: theories, problems, methods, examples.

All 4 kinds of data need a table of contents of variable detail.

Browse through statical HTML-Pages without e.g. matching a model with a problem.

Do some examples: Visitors should get an overview over all the features available at an ISAC-site, and the most exciting feature is stepwise interactive calculating and reasoning guided by the system. This should be demonstrated by some examples, which also the visitor can execute.

Call of a worksheet from a link in the knowledge for instance demonstrating an example for a definition, a problem, etc.

Execute some selected examples in the example collection.

1.4 Requirements of the learner

The typical access of the learner to ISAC is via the example collection (see UR1.2.5). Solving an example typically comprises three phases, modeling, specifying and solving.

Help in the model phase: In the model phase the learner generally has to input the items for a model: input and output items can be 'correct', 'incomplete', 'missing', 'superfluous' or have a 'syntax error'; preconditions can be 'correct' or 'false' — these properties should be clearly indicated.

Visualization of the feedback on input to a model.

Retrieve a matching problem: Given a model initiating a calculation, or as a subproblem within a calculation, the learner has to determine a problem matching this model. This involves information retrieval from a large problem-hierarchy. There the learner may get lost, and thus he should get help: make the math-engine find a matching problem; visualize the problem found within the hierarchy.

In case of an exam, the user is forced to find the correct problem with no or limited help. Therefore it is necessary to adjust the amount of help given by the browsers (e.g. only "match" or "nomatch" instead of a detailed listing of all conditions)

Can retrieve a matching problem.

Gets help by automated refinement of a problem.
Additional information in a calculation should be provided any time on request of the learner. This feature comprises a more detailed views onto the proofstate, as well as explanations. A command ‘detail’ will be available on the following items:

<table>
<thead>
<tr>
<th><code>detail</code> on</th>
<th>element</th>
<th>yields</th>
</tr>
</thead>
<tbody>
<tr>
<td>whole formula</td>
<td>//</td>
<td>generating tactic</td>
</tr>
<tr>
<td>whole formula</td>
<td>//</td>
<td>associated assumptions</td>
</tr>
<tr>
<td>formula</td>
<td>function-constant</td>
<td>definition in the theory</td>
</tr>
<tr>
<td>tactic</td>
<td>theorem</td>
<td>theorem instantiated</td>
</tr>
<tr>
<td>theorem instantiated</td>
<td>//</td>
<td>animation of matching</td>
</tr>
<tr>
<td>tactic</td>
<td>ruleset</td>
<td>intermediate steps</td>
</tr>
<tr>
<td>tactic</td>
<td>subproblem</td>
<td>intermediate steps</td>
</tr>
<tr>
<td>model context</td>
<td>theory</td>
<td>file of the respective theory</td>
</tr>
<tr>
<td>model context</td>
<td>problem</td>
<td>model instantiating this problem</td>
</tr>
<tr>
<td>model context</td>
<td>problem</td>
<td>inherited assumptions</td>
</tr>
<tr>
<td>model context</td>
<td>method</td>
<td>model instantiating the guard</td>
</tr>
<tr>
<td>model context</td>
<td>method</td>
<td>script of the respective method</td>
</tr>
</tbody>
</table>

UR 1.4.4 Details of a calculation as given above.

UR 1.4.5 Explanations to each item in the knowledge base.
The explanations are filtered w.r.t. the current dialog state (explanations given several times are skipped, etc.) and filtered w.r.t. the knowledge profile.

The flow of interaction shall be adapted to the learner. The learner might be bored because the system offers too little challenge (by using less active dialog atoms – see UR1.6.1, or by proceeding in too little steps of calculation) – or, in contrary, the user might be frustrated by too high challenges (in activity and/or stepwidth).

UR 1.4.6 The ‘activity’ of the dialog atoms adapts to the learner.

UR 1.4.7 The learner can override the dialog atoms chosen by the system.

UR 1.4.8 Varying stepwidth with tactics, rule sets and subproblems.

UR 1.4.9 The stepwidth can be overridden by the learner. I.e. the stepwidt mechanically chosen by the dialog guide.

The dialog regards presets of the course admin (see Sect.1.7 below), when guiding the flow of interaction, and the dialog regards the ongoing interaction with the student.

UR 1.4.10 The dialog regards the performance in calculations done by the learner in the current session. The performance is measured by response
times, errors, difficulty of examples done, requests into the knowledge base, active-passive behaviour.  

**UR 1.4.11** The dialog regards the knowledge touched by the learner in the current session.

**UR 1.4.12** The dialog regards the history of performance and knowledge touched in previous sessions.

Guarantee of correct results is given by ISAC’s proofstate in the ME. This guarantee can only be given, when each editing on the worksheet is mirrored in the prooftree (eventually cutting certain branches, if an intermediate step is redone). Sometimes the learner might want to edit a calculation (shorten the calculation by cutting intermediate steps, etc.) without affecting the prooftree any more.

**UR 1.4.13** ISAC guarantees correct results.

In this case the worksheet reflects the proofstate, which shall be indicated on the screen and on the printout; in the latter case this indication should be certificate which cannot be manipulated.

**UR 1.4.14** A calculation can be edited arbitrarily. Once this has been done, the ME cannot resume user guidance. Worksheet and printout don’t show the certificate.

### 1.5 Requirements of the math author

In the current phase 1 of development (tutoring system only) all the authoring of math knowledge will be done on the SML toplevel, i.e. immediately on the datastructures holding the knowledge. This part of the task is described in the ‘interfaces for authors of math knowledge’ [1].

Another result of authoring math knowledge, however, will be tools for the visitor and the learner to view the knowledge generated. This part of the task is described here.

**UR 1.5.1** Automatic links to other occurrences of an item.

Automated linking is done between the following items:

<table>
<thead>
<tr>
<th>item</th>
<th>in . . .</th>
<th>linked to occurrence in</th>
</tr>
</thead>
<tbody>
<tr>
<td>predicate</td>
<td>theorem</td>
<td>definition in theory</td>
</tr>
<tr>
<td></td>
<td>pre/postcondition</td>
<td>definition in theory</td>
</tr>
<tr>
<td>theorem</td>
<td>tactic</td>
<td>definition in theory</td>
</tr>
<tr>
<td>TODO ⁴</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

³The completion of this list is up to a future phase of development, and the evaluation of these data as well.
**UR 1.5.2** Exchange data with other ISAC sites.
TODO

**UR 1.5.3** Exchange data with other knowledge bases.
TODO

Authoring theories
The generation of the theory browser is already implemented by Isabelle. Within phase 1 of development, ISAC will take this component without any change.

Authoring problems
TODO

Authoring methods
TODO

1.6 Requirements of the dialog author

We hope that it is possible to develop a language which allows to define dialog patterns (as combinations of dialog atoms already implemented) and dialog modes sequencing dialog patterns. By means of such a language learning strategies could be described, and this description could be interpreted in reaction to a dynamic dialog state and according to an knowledge profile.

To do such a ‘dialog programming’ is considered a comprehensive task, which in general exceeds the knowledge of a course designer or a course admin (both see below) — thus the statement of respective requirements is separated. A course designer should be able to associate courses with dialog profiles (combining dialog modes), and the course admin should be able to select dialog modes within process of time in a course.

The dialog atoms are the following, ordered by descending ‘activity’ of the learner: All atoms concern a step from the current formula \( f \) applying a tactic \( tac \) which yields the resulting formula \( f' \) (the derivation of \( f \) ), i.e. \( f \longrightarrow_{tac} f' \).

1. given \( f \), input the next formula \( f' \)
2. given a partial \( f \) (supplied by ISAC), complete \( f \) such that it is a derivation of \( f \)
3. given \( f \), input a tactic \( tac \) to be applied to \( f \)
4. given \( f \), select \( tac \) from a list (supplied by ISAC) to be applied to \( f \)
5. given \( f \) and a partial \( tac \), complete the \( tac \) (i.e. a theorem, a substitution, etc.) such that it can be applied to \( f \)
6. given \( f \), \( tac \), and a partial \( f' \), complete \( f' \) such that it is the result of applying \( tac \) to \( f \)
7. given \( f \) and \( f' \), input \( tac \) such that \( f' \) is the result of \( f \) applying \( tac \)

8. given \( f \) and \( f' \), select \( tac \) from a list (supplied by \( ISAC \)) such that \( f' \) is the result of \( f \) applying \( tac \)

9. given \( f, f' \), and a partial \( tac \), complete \( tac \) such that \( f' \) is the result of \( f \) applying \( tac \)

**UR 1.6.1** Dialog atoms according to the list above.

The 'activity' requested from the learner for doing such a dialog atom can be varied by the appropriateness of the lists of tactics supplied by \( ISAC \) and the structure of the formula to be completed. The appearance of the dialog atoms is very different depending on the phase, modeling or solving.

The requirements analysis for a dialog state and a learner model is postponed to a future phase of development; the 'dialog programming' will prepare \( ISAC \) to meet the user-requirements UR1.4.10, UR1.4.11, UR1.4.12, and UR??.

### 1.7 Requirements of the course admin

Authoring in \( ISAC \) comprises various tasks: authoring mathematics knowledge and authoring the dialog have been described in Sect.1.5 and Sect.1.6; these both tasks require more special knowledge than the others. Sect.1.8 describes a kind of knowledge ('explanations') which may change from one course to the other, and which does not require special knowledge on computer mathematics or dialog design.

The latter part of authoring is done by the course designer preparing as course in advance (see Sect.1.8). Some of the knowledge prepared in advance underlies time constraints, which are managed by the course admin. The requirements of the course admin are separated in this section (despite the fact that the course designer and the course admin are one and the same person, the lecturer or teacher of the course).

**There are groups of learners** in order to support the administration of courses. The membership w.r.t. these groups determines the selections of examples in the example collection (see UR1.8.23), the selection of explanations in the knowledge (see UR1.8.21) and the initial setting of the dialog as captured in UR1.4.4 and UR1.4.8.

**UR 1.7.1** There are groups of learners.

**UR 1.7.2** One learner may belong to different groups but only to one group within a session.
Time limits for delivering course material are a general requirement for educational systems, and it applies for example collections as well. This requirement provides the course admin to distribute workload over time and to focus the attention of the learners. Sometimes it may be desirable to have some examples finished within a certain time limit. Also examples (for instance prepared for an examination) may be invisible for learners.

UR 1.7.3 Groups of examples may be locked for groups of learners for certain durations. (SR: attention with links from the KB !)

UR 1.7.4 Groups of examples may be invisible for all users except the example author for certain durations. (SR: examples have an author !)

UR 1.7.5 Restricted help for the learners during a exam i.e. particular links into the knowledge base are restricted for a particular group for a limited time.

Survey on the progress of the learners i.e. of the students in a course. Such surveys are indispensable requirements for a course admin. Thus all the surveys below are related to a specific course.

UR 1.7.6 Which examples have been done by whom with which performance.

UR 1.7.7 Sorts w.r.t. the examples: frequency of touched, not touched, solved, unsolved (, evaluation of performance, as soon as evaluation functions have been designed) over all students of a course.

UR 1.7.8 Sort w.r.t. the students: likewise over all examples in specified groups.

UR 1.7.9 Which lookups into the knowledge have been done.

UR 1.7.10 Sorts w.r.t. the destination: likewise over students.

UR 1.7.11 Sorts w.r.t. the students: likewise over the knowledge.

Special requirements may be raised by field studies on learning mathematics. ISAC shall be open for such special research.

UR 1.7.12 Special queries for field studies. I.e. queries over examples, lookups, students and courses.

Written examinations can be done due to requirements stated elsewhere: hidden examples UR1.7.4, restricted access to knowledge UR1.7.5, adapted dialog UR1.6.1 and evaluation UR1.7.8 (evaluation of performance as soon as evaluations function have been designed).

UR 1.7.13 Written examinations can be done with ISAC following the traditional conventions.
1.8 Requirements of the course designer

A course designer prepares the learning materials and exercises for a course according to the goals of that course, or according to the learners’ level. If ISAC is being used for a course, specific explanations may be added to the knowledge and examples will be prepared within the example collection.

UR 1.8.1 Explanations from other ISAC-sites can be imported.

UR 1.8.2 Examples from other ISAC-sites can be imported.

Appearance of example collections: One application of ISAC will be to mechanize tutoring on existing example collections, as found for instance in traditional textbooks. Thus the example collection must copy the structure already given (the enumerations, letters etc.) in order to allow the learner to find a particular example (e.g. given as homework). Traditional textbooks use arbitrary labels for their chapters and sections, the levels are nested arbitrarily deep, and there are arbitrary labels of the examples.

For copyright reasons it also may happen, that the example itself (i.e. text, formulas, figures) is not displayed, only its label. In this case the label should be located exactly at the same position on a virtual ‘page’ on the screen as the original position in the page of the textbook.

UR 1.8.3 Each example has a unique label.

UR 1.8.4 The labels are defined by the author

UR 1.8.5 The graphical layout is equivalent to textbooks.

The structure of example collections is a hierarchy of groups of examples. A group of examples consists of the part visible to the learner (which may be copied from a textbook) and additional data for ISAC to suggest examples of a level appropriate to an individual learner. There are already requirements concerning examples, UR1.7.3 and UR1.7.4, which apply to groups of examples for convenience.

UR 1.8.6 There are groups of examples with common properties, UR1.7.3 and UR1.7.4.

UR 1.8.7 The hierarchy of groups of examples can be zoomed in/out.

UR 1.8.8 A group and/or an example are weighted w.r.t. properties to be defined in a later phase of development (at least the properties ‘difficulty’ and ‘length’).

5The contents of a math knowledgebase, however, can only be edited by a mathematics author!
UR 1.8.9 *A limit of the number of solved examples* may be defined for a group; if the limit is touched, this group has been mastered successfully by a student within a certain course. An evaluation-function for the performance will be advantageous here in the future.

UR 1.8.10 *There might be obligatory examples* in a group; i.e. such an example must be solved by a student in order to have the group mastered successfully.

*Edit examples in the example collection* requires several editors. An example can be described by verbal text, by formulas, and by figures (and eventual by movies). Additionally, each example contains data hidden from the visitor and the learner.

UR 1.8.11 *Integrated editing of text, formulas and figures*

UR 1.8.12 *A formalizations is a list of formulas.*

UR 1.8.13 *A specification is a triple* of three pointers to a theory, a problem and a method respectively.

UR 1.8.14 *Each example is combined with a list of pairs of* a formalization and a specification respectively.

UR 1.8.15 *Formalizations and specifications are hidden* from the user.

UR 1.8.16 *An example may have error schemes* for particular input items, paired with an explanation (see UR1.8.19...UR1.8.22 below).

*Check of solvability* for each example or for groups of examples is necessary before delivery to the learners.

UR 1.8.17 *The course designer can execute an example.*

UR 1.8.18 *The course designer can execute groups of examples* in a batch mode.

*Edit explanations in the knowledge base* requires the same range of editors as in UR1.8 plus lookup other sources of knowledge on the web, which could be linked into the knowledge base for use in a particular course.

UR 1.8.19 *An explanation may consist of* text, formulas, figures (movies, and links into the KB, into the web, to an example in the resident site.

UR 1.8.20 *An explanation is one combination* of arbitrary elements from UR1.8.19; i.e. there is one link which, however, may have nested links.

UR 1.8.21 *Explanations are course-specific.* Each course might have different explanations according to the contents of the course, according to the specific example collection, and according to the learners' level.

UR 1.8.22 *Each item in a knowledge base can have one explanation.* per course.
A knowledge profile for each course results from the explanations following the requirements from above, plus from the detail knowledge is used in, from error schemes and from fill-in patterns for theorems.

UR 1.8.23 An example group and/or an example specifies the details as defined in UR1.4.4.

UR 1.8.24 There are error schemes, eventually several for one theorem in a particular method, or for specific tactics in a particular method. An error scheme is paired with an explanation (UR1.8.19…UR1.8.22).

UR 1.8.25 There are fill-in patterns for theorems, eventually several for one theorem in a particular method. Such patterns are used for the dialog atoms UR1.6.1, e.g. 2 or 5.

A dialog profile can be preset according to the students’ level. These predefined setting can be overridden by the students in most cases, but not in all cases. The dialog profile will be more elaborated as soon the ‘dialog programming’ and the user model habe been clarified in a future development phase.

UR 1.8.26 A dialog mode restricts the dialog atoms (UR1.6.1) to be used by the learner within certain time limits (e.g. during the time of a written examination).

1.9 Requirements of the administrator

The administrator has to set up the system and implement the overall design of an ISAC-site. This includes introductory pages as well as an basic overall design (Corporate Design, links to “home”…)

UR 1.9.1 Basic parameters to adapt the HTML-putput to an corporate design (e.g. colors, font, …)
Chapter 2

Software Requirements Document

This Software Requirements Document is structured along the modules abstracted from the functionality defined in the User Requirements Document. The User Requirements Document, however, is structured along the different kinds of users envisaged. In order to establish comfortable tracing, the \( m : n \) relation between user requirements and software requirements will be accurately recorded in a double-linked way.
Bibliography

Appendix A

\textit{ISAC}s tactics

Init\_Proof\_Hid (dialogmode, formalization, specification) transfers the arguments to the math engine, the latter two in order to solve the example automatically. The tactic is not intended to be used by the student; it generates a proof tree with an empty model.

Init\_Proof generates a proof tree with an empty model.

Model\_Problem problem determines a problemtype (eventually found in the hierarchy) to be used for modeling.

Add\_Given, Add\_Find, Add\_Relation formula inputs a formula to the respective field in a model (necessary as long as there is no facility for the user to input formula directly, and not only select the respective tactic plus formula from a list).

Specify\_Theory theory, Specify\_Problem problem, Specify\_Method method specifies the respective element of the knowledgebase.

Refine\_Problem problem searches for a matching problem in the hierarchy below ‘problem’.

Apply\_Method method finishes the model and specification phase and starts the solve phase.

Free\_Solve initiates the solve phase without guidance by a method.

Rewrite theorem applies ‘theorem’ to the current formula and transforms it accordingly (if possible – otherwise error).

Rewrite\_Asm theorem is the same tactic as ‘Rewrite’, but stores an eventual assumption of the theorem (instead of evaluating the assumption, i.e. the condition)

Rewrite\_Set ruleset similar to ‘Rewrite’, but applies a whole set of theorems (‘ruleset’).
Rewrite.Inst (substitution, theorem), Rewrite.Set.Inst (substitution, ruleset) similar to the respective tactics, but substitute a constant (e.g. a bound variable) in ‘theorem’ before application.

Calculate operation calculates the result of numerals w.r.t. ’operation’ (plus, minus, times, cancel, pow, sqrt) within the current formula.

Substitute substitution applies ’substitution’ to the current formula and transforms it accordingly.

Take formula starts a new sequence of calculations on ’formula’ within an already ongoing calculation.

Subproblem (theory, problem) initiates a subproblem within a calculation.

Function formula calls a function, where ’formula’ contains the function name, e.g. ’Function (solve 1 + 2x + 3x^2 = 0 x)’. In this case the modelling and specification phases are suppressed by default, i.e. the solving phase of this subproblem starts immediately.

Split_And, Conclude_And, Split_Or, Conclude_Or, Begin_Trans, End_Trans, Begin_Sequ, End_Sequ, Split_Intersect, End_Intersect concern the construction of particular branches of the prooftree; usually suppressed by the dialog guide.

Check_elementwise assumptions w.r.t. the current formula which comprises elements in a list.

Or_to_List transforms a conjunction of equations to a list of equations (a questionable tactic in equation solving).

Check_postcond: check the current formula w.r.t. the postcondition on finishing the respective (sub)problem.

End_Proof finishes a proof and delivers a result only if ’Check_postcond’ has been successful before.
Appendix A

Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG</td>
<td>browser generator</td>
</tr>
<tr>
<td>BR</td>
<td>browser, for theories, problems, methods or examples</td>
</tr>
<tr>
<td>DG</td>
<td>dialog guide</td>
</tr>
<tr>
<td>KB</td>
<td>mathematics knowledge base</td>
</tr>
<tr>
<td>ME</td>
<td>mathematics engine</td>
</tr>
</tbody>
</table>