

# Computer Science Exercises in a Virtual University

Manfred Widera  
Department of CS  
FernUniversität in Hagen  
Germany

Barbara Messing  
Department of CS  
FernUniversität in Hagen  
Germany

Gabriele Kern-Isberner  
Department of CS  
Universität Dortmund  
Germany

Malte Isberner  
Department of CS  
FernUniversität in Hagen  
Germany

Christoph Beierle  
Department of CS  
FernUniversität in Hagen  
Germany

**Abstract**—In distance teaching, direct feedback to the students is difficult to give, but crucial for their learning success. The system ASTERIX was developed in order to support this direct feedback. It contains different interactive tasks from the area of the formal foundations of computer science. ASTERIX is available over an internet interface. The implementation of the interactive presentation and correction of the tasks is generated automatically from high-level XML specifications.

In this work we give a detailed overview of the system architecture and the supported task types, both, from the students' and the teacher's point of view and we show how ASTERIX is used in the context of distance teaching at the FernUniversität in Hagen. The presentation is driven by a series of real world examples.

**Index Terms**—distance learning, self-test tasks, formal foundations of computer science, XML-specifications

## I. INTRODUCTION

Providing direct feedback to the students is an important task in distance teaching. This task is, unfortunately, not easy to perform by human teachers because it is time consuming and demanded by the students at unusual times. Motivated by this situation, we developed the system ASTERIX that is used in the Bachelor program in computer science at Germany's distance teaching university FernUniversität in Hagen. ASTERIX provides interactive exercises for self testing with automatic correction of the answers given by the students. It is an additional offer to our students, complementing the regular course material. To clarify in which context this system is used, we will not only present the system ASTERIX, but we will also give a short description of the way teaching takes place at the FernUniversität.

This paper extends our previous work [WMKI+06] by a description of the system's architecture, a detailed presentation of the internal XML structure of the self tests and a series of example exercises, clarifying the correspondence between the XML specifications and the generated presentations of the tests. The paper is organized as follows: Section II analyses the requirements for teaching computer science in a virtual university, describes the typical situation in distance teaching in our University, and motivates the use of ASTERIX. In Section III the architecture of ASTERIX and its implementation are described. In Section IV a series of example exercises is shown, illustrating the essential features of the system. Section V introduces the

task types currently supported by ASTERIX. In Section VI we describe how high-level XML specifications can be used for defining tasks, and Section VII gives some conclusions and points out further work.

## II. MOTIVATION AND OVERVIEW

Essential parts of the way computer science is being taught at the FernUniversität in Hagen are course texts and home work assignments being sent out to the students. The course texts contain the material the students are expected to study at home. To support the understanding of the course material, so-called self-test tasks are integrated into the course text; students can control their understanding by looking up the solutions given at the end of the respective course unit.

In addition to these self-test tasks, the home work assignments must be completed by the students and their solutions must be sent in by particular due dates, motivating the students to work continuously. These home work assignments are corrected by the tutors of the course who send their corrections and comments back to the students. The correction of the home work assignments gives the students an individual feedback about their studying success. Due to the distance teaching situation, this feedback can only be given with a certain delay of time, while the course is going on. Especially for large classes like the obligatory first-year courses the correction takes some time. The individual feedback for the home work assignments has proved itself invaluable for the students, but especially for first-year students, further support is desirable. Newsgroups and internet fora are often used by the participants of the first-year courses to discuss difficulties and to clarify questions, and have proved to be very helpful.

In comparison with students at on-campus universities, the qualifications of the participants in courses of the FernUniversität are particularly heterogeneous. Most of them study and have a full-time job at the same time, thus often having severe time constraints. Some students have had few contacts with mathematical methods being used mandatorily in computer science. These gaps can only be closed by practicing again and again. If the personal contact is reduced, the direct studying control is reduced also. Thus, the questions "Have I done this right?" often remains open for too long.

ASTERIX complements the tutorial program with self-test tasks in the course texts. This system was developed as an accompanying support tool to the course *Formal Foundations of Computer Science*, a first-year course in the Bachelor of Science program in Computer Science. Among other things ASTERIX deals with elementary knowledge about set theory, relations and functions, and propositional and first-order logic. ASTERIX offers an additional way of doing exercises with automatic and direct feedback, enabled by its interactive access over the internet. Its tutoring component provides not just corrections to the students' answers, but also gives hints and tips to the students guiding their ways to find proper solutions.

Not only the large number of first-year students being able to profit from using ASTERIX justified the efforts of developing this system. Also the subject area we are currently using ASTERIX for—i.e. basics of formal foundations of computer science—seems to be well-suited for automated self-testing. For instance, when computing the intersection of two sets or negating a logic formula, there is little room for subjective interpretation. Especially for beginners, already a few carefully chosen questions can reveal understanding gaps. Moreover, there is a rich fund of examples and tasks, and from former teaching experiences, many points posing special difficulties for the students as well as frequently made mistakes are well-known.

ASTERIX' range of exercises is also useful because many first-year students of computer science tend to think that the mathematical and formal base knowledge taught in this course seems to be far from computer science applications. Not just for them it is nice to see that part of learning and practicing this takes place not only on paper. Therefore, ASTERIX is an additional incentive for the students to engage themselves in the basic formal foundations, and only students having a good grasp of these foundations are able to learn advanced techniques.

Besides simple multiple choice tasks ASTERIX offers a range of different task types, e.g., matching tasks, tasks involving truth tables for the evaluation of logic formulas, or tasks where the elements of a set are to be determined. In order to guarantee portability and extendibility as much as possible, the internal task specifications are done system-independently in XML. A parser generates code for the interactive representation, processing and automated correction of the tasks from the XML specifications. The access via the internet is achieved by using the *Learning Platform Virtual University (Lernraum Virtuelle Universität, LVU)* of the FernUniversität [LVU05].

### III. ARCHITECTURE AND IMPLEMENTATION

ASTERIX consists of several modules (cf. Fig. 1). A parser produces the XML task code for the interactive processing which is controlled by the task module. A correction module checks the input of the user. Depending on whether the task was solved correctly or not, the tutoring module generates remarks, error messages, hints or help information so that the user can check his solution directly and gets relevant information on the subject of the

task. Figure 1 also indicates the internet connection of the system that uses the learning platform LVU [LVU05] and the interface of the Virtual Computer Science Laboratory VILAB [LGH02].

ASTERIX is implemented in C++ and uses the *Qt* library of the company *Trolltech* for several standard tasks (e.g. for the representation of the user interface, parsing the XML-files). As implementation and execution platform Linux is used, but ASTERIX can be executed on every platform where the *Qt* library is available. The number of lines of the (not automatically generated) source code is approximately 6600.

ASTERIX was designed and implemented using the object-oriented paradigm. For every task type there is a class that itself is derived from a base class of task types. These classes receive the XML data from the XML parser, build a user interface, and evaluate the input of the user. The construction of the task class from the base class is carried out dynamically depending on the task type provided by the parser. With small modifications it is possible to add new task types. Furthermore, it is easily possible to implement new task types using dynamically loaded libraries.

### IV. SELF-TESTING WITH ASTERIX

After registering with the system, the student can choose among the different self-testing tasks being offered. For the task chosen, ASTERIX presents a task description and indicates how the solution has to be filled in. The input fields and formats depend on the type of the task. A task may consist of several sub-tasks that can be selected from a menu.

The evaluation of the input is started by clicking the corresponding button. The task processing is an iterative process, since also partial inputs can be evaluated. If ASTERIX detects errors in the input given, the student has the possibility to correct them in the next step and to go on with further inputs.

For each task, help information, tips and remarks are available:

- *Help* information can be requested by pushing the respective button.
- *Tip* is shown when a (sub-)task is solved wrongly. Unlike an error indication it is a task specific text pointing out *why* the student's solution is wrong.
- *Remark* is shown when a task has been solved correctly, providing further information on the subject of the task.

In the following we describe some tasks from various topic areas and give examples of different task types supported by ASTERIX.

Figure 2 shows a task on the topic “sets and first-order logic”. A list of set expressions must be matched against a list of first-order formulas. Two matching elements in the lists must be connected by drawing a line between them which is achieved by clicking the respective buttons on the left hand side and on the right hand side. In Fig. 2, three correct assignments have been made, along with a

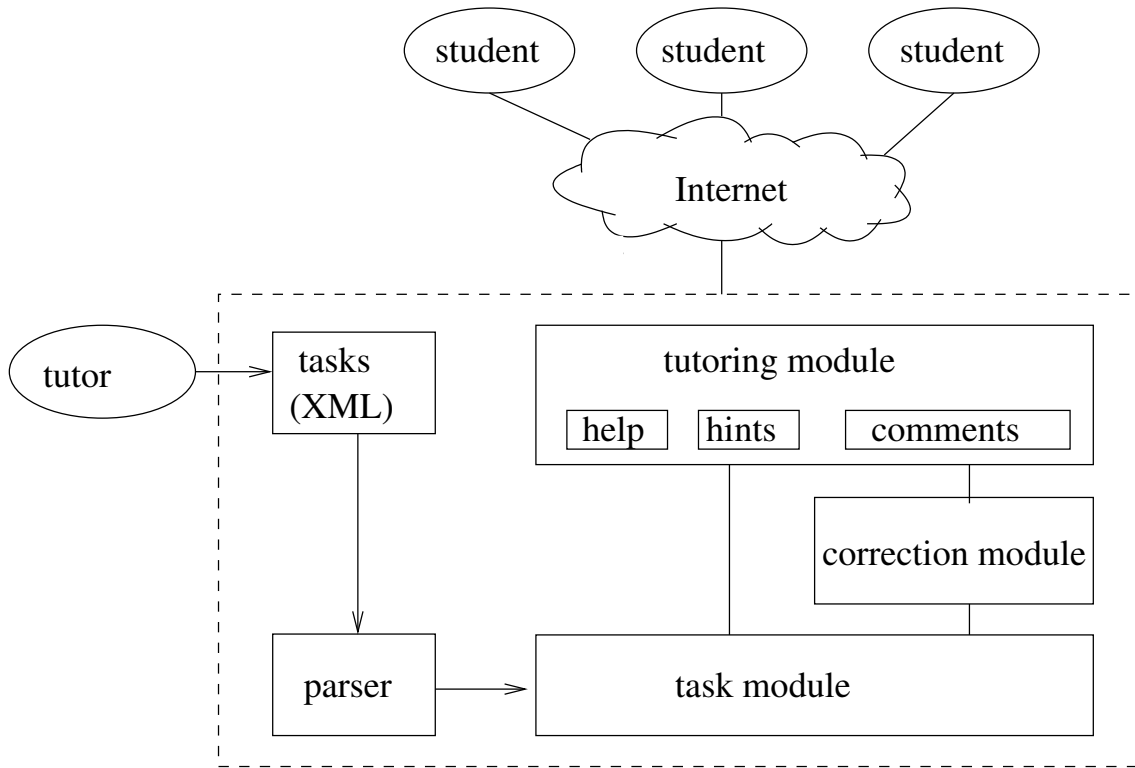


Fig. 1. The ASTERIX system

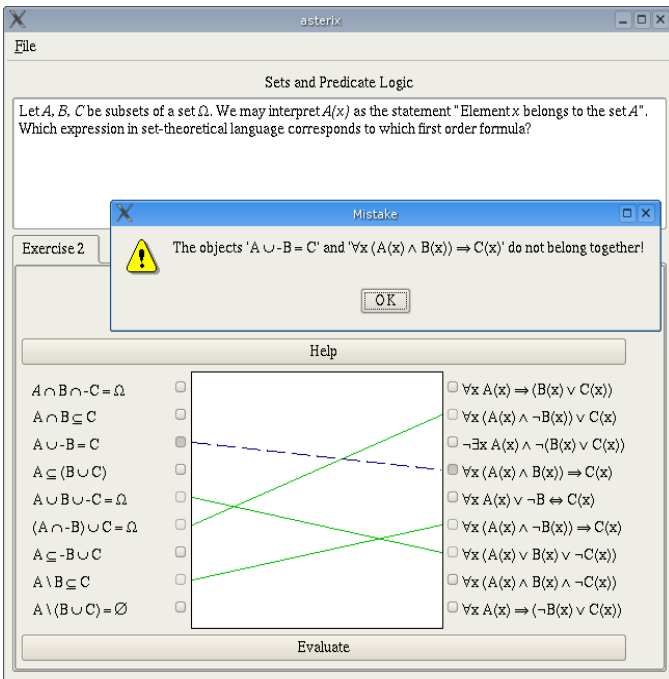


Fig. 2. Match task in the area of sets and first-order logic

wrong assignment for which a corresponding error message appears.

Handling truth tables plays a crucial role in propositional logic. Figure 3 shows how a propositional formula

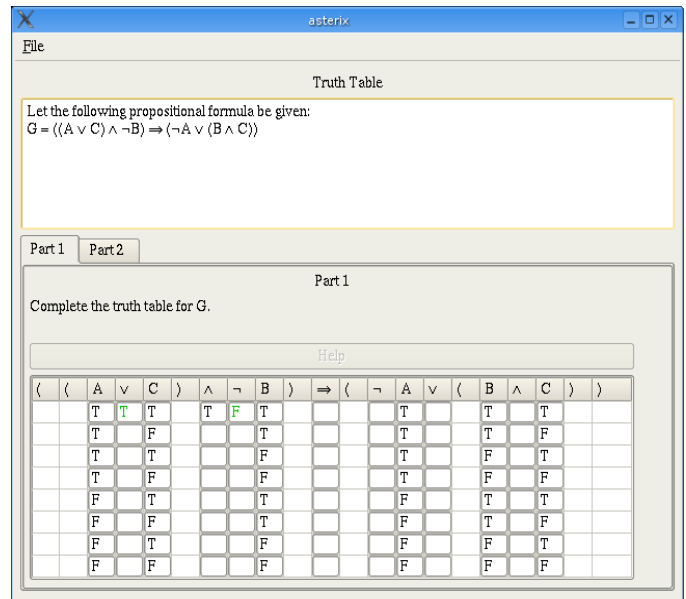


Fig. 3. Interactive truth table for a propositional formula

can be evaluated by means of an interactive truth table.

In exercises dealing with sets, the training and learning effect can be improved by not just presenting a list of candidate solutions the student can choose from, but by enabling the direct input of the elements of a set and of set expressions. In the task shown in Fig. 4, different set expressions must be evaluated. The answer must be given

## V. SUPPORTED TASK TYPES

ASTERIX supports several different task types, thereby going well beyond the functionality of multiple-choice tasks:

In a *1 of N choice* task, N solution candidates are given, out of which exactly one is correct. Thus, the student chooses exactly one solution. If it is wrong, it will not be offered again as a possible choice to the student when he is correcting his answer.

In an *X of N choice* task, N solution candidates are given, out of which a to the student unknown number of solutions is correct. The student may mark as many solutions as he wants. After evaluation, the wrongly selected solutions can not be selected again, and the correctly selected solutions can not be deselected.

For a *truth table* task, the tutor provides a propositional formula to be evaluated by the student. ASTERIX presents a truth table to the student containing all possible truth value assignments to the variables occurring in that formula. For every logical subformula there is a field where the truth value of the subformula under the respective variable assignment must be entered (cf. Fig. 3).

In a *1 of N table* task, several *1 of N choice* tasks having an identical set of candidate solutions are grouped into a single table. The rows of the table correspond to the different candidate solutions, and the columns of the table specify the questions to be answered. In every column, there are selection buttons out of which only one can be selected. After evaluation of the answers, ASTERIX states for every column whether the selection was correct or not.

An *X of N table* task is similar to a *1 of N table*, except that in a column, several answers may be selected, analogously to an *X of N choice* (cf. Fig. 5).

In a *text input* task, the student answers the given questions by typing in textual or numerical input in the respective fields. For convenience, this could also be numerical expressions; for example, instead of typing “256”, the student could also enter the expression “ $2^8$ ” as an alternative. ASTERIX automatically compares the given answers with the correct solutions which have been specified by the tutor.

A *choice* task is similar to a *1 of N table* task; however, the presentation is not given in a table, but in the form of drop down list attached to the corresponding question.

A *set input* task allows for a rather complex input format for defining sets. For defining the elements of a set, the student may use numbers, characters, tuples and subsets. Furthermore, the task specifications may specify sets by using set expressions containing operations like union, intersection or Cartesian products. The correct solution sets are computed automatically from these set expressions and compared to the sets entered by the students (cf. Fig. 4).

In a *match* task, the student must establish a correspondence between the elements of two different lists. The matching connections are represented graphically as illustrated in Figure 2. The order of the elements in the list on the left hand side are taken from the internal XML task

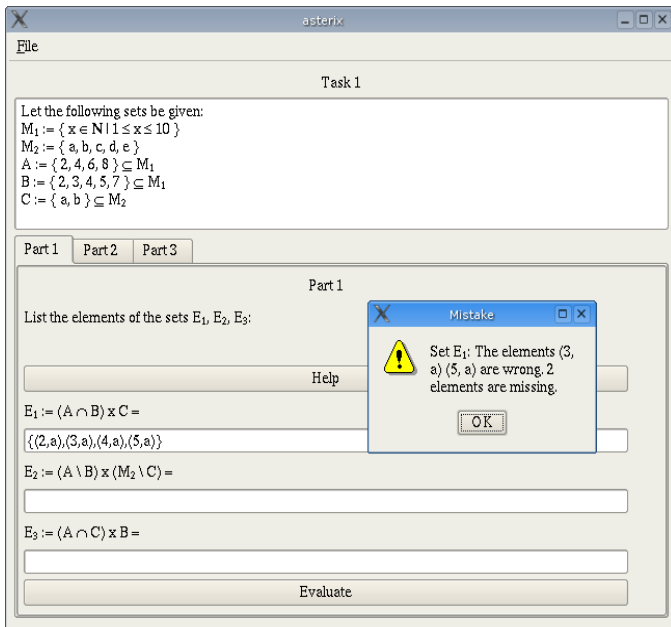


Fig. 4. Specification of sets with direct input in set notation

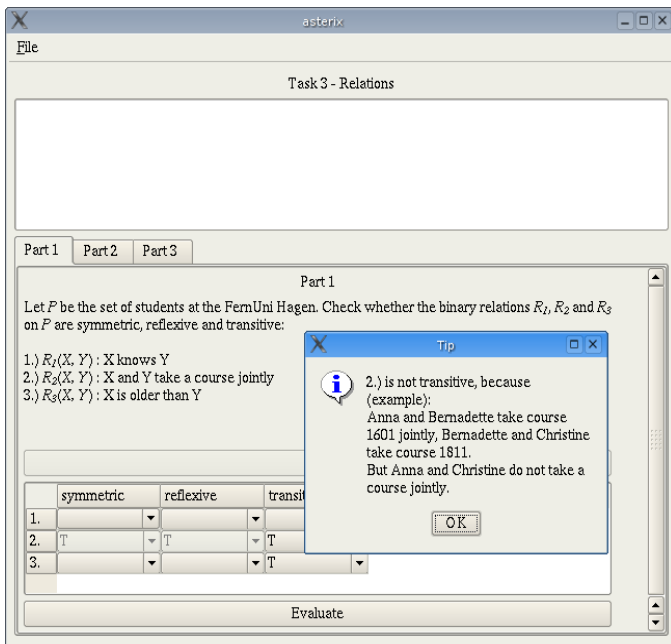


Fig. 5. *X of N table* to practice elementary properties of relations

and typed in in regular mathematical set notation.

In Fig. 5 a task is shown for practicing basic properties of relations. The task is to determine properties like symmetry and transitivity of given relations like *X knows Y*. The answer is given by selection buttons where more than one answer might be correct. The hint in Figure 5 presents an illustrative counter-example why the second relation (*X and Y take a course jointly*) is not—as the student suggested—transitive.

```

<?xml version="1.0" ?>
<tasksheet>
<title>Sets and Predicate Logic</title>
<text>Let  $[i]A, B, C[/i]$  be subsets of a set  $\{\omega\}$ . We may interpret  $[i]A(x)[/i]$  as
the statement "Element  $[i]x[/i]$  belongs to the set  $[i]A[/i]". Which expression in
set-theoretical language corresponds to which first order formula?</text>
<task type="match">
<title>Exercise 2</title>
<help>For example, the expression  $A \cap \neg B$  denotes all those elements of  $\{\omega\}$ 
for which  $A$  is true, but  $B$  is not. Please note that on the left hand side there are
statements about the relationships between sets (like subset relationship, equality).
Try to first formulate these relationships in natural language, then in first-order
predicate logic. E.g.,  $A \cap \neg B = A$  could be read as:  $A$  intersected with the
complement of  $B$  yields again  $A$ , that is, the elements of  $A \cap \neg B$  and  $A$  are the
same. In FOL we get:  $\forall x (A(x) \wedge \neg B(x)) \implies A(x)$ .</help>
<answer left="A  $\cap B \cap \neg C = \{\omega\}$ "
right="  $\forall x (A(x) \wedge B(x) \wedge \neg C(x))$ " />
<answer left="A  $\cap B \subseteq C$ "
right="  $\forall x (A(x) \wedge B(x)) \implies C(x)$ " />
<answer left="A  $\cup \neg B = C$ "
right="  $\forall x A(x) \vee \neg B(x) \implies C(x)$ " />
...
</task>
</tasksheet>$ 
```

Fig. 6. XML specification of the task given in Figure 2

```

<?xml version="1.0" ?>
<tasksheet>
<title>Truth Table</title>
<text>Let the following propositional formula be given:  $G = ((A \vee C) \wedge \neg B) \implies (\neg A \vee (B \wedge C))$ </text>
<task type="truthtable"> <title>Part 1</title>
<text>Complete the truth table for  $G$ .</text>
<var name="A" />
<var name="B" />
<var name="C" />
<expr string="((A or C) and not B) => (not A or (B and C))" />
</task>
... (specification for Part 2)
</tasksheet>

```

Fig. 7. XML specification of the task given in Figure 3

specification given by the tutor (cf. Section VI), whereas the order of the elements on the right hand side is generated randomly each time the task is invoked by the student.

## VI. XML SPECIFICATIONS

In order to obtain a maximal degree of portability and flexibility, the internal representation of the tasks is given system-independently in XML. For the representation of teaching contents, XML is a well-established standard and it is being used in many projects in this area (e.g. [WWR05]). The argumentation for XML can be trans-

ferred directly to ASTERIX. For the ASTERIX tasks, the XML version 1.0 [BPSM<sup>+</sup>04] is being used.

### A. General Information

The top level element of every task file is the tag **tasksheet**. All subtasks belonging to this task are contained in this tag, as well as a task title and a text briefly describing the task.

A **task** tag starts a new subtask and is structured similarly to a **tasksheet** tag. The attribute **type** specifies the type of the task. Further general tags are, for instance,

```

<?xml version="1.0"?>
<tasksheet>
<title>Task 1</title>
<text> Let the following sets be given:
      M1 := { x ∈ ℕ | 1 ≤ x ≤ 10 }
      M2 := { a, b, c, d, e }
      A := { 2, 4, 6, 8 } ⊆ M1
      B := { 2, 3, 4, 5, 7 } ⊆ M1
      C := { a, b } ⊆ M2 </text>
<task type="setinput">
<title>Part 1</title>
<text> List the elements of the sets E1, E2, E3: </text>
<help> A x B consists of ordered pairs. Thus, (a, b) is different from (b,
      a). On the other hand, within set brackets, the particular order or even "multiple
      occurrences" are not significant: {a, b} = {b, a} = {a, a, b}. </help>
<set name="A" contents="{2, 4, 6, 8}" />
<set name="B" contents="{2, 3, 4, 5, 7}" />
<set name="C" contents="{a, b}" />
<set name="M2" contents="{a, b, c, d, e}" />
<input name="E1" defn="(A intersect B) x C" defnString="(A ∩ B) x C" />
<input name="E2" defn="(A B) x (M2 C)" defnString="(A B) x (M2 C)" />
<input name="E3" defn="(A intersect C) x B" defnString="(A ∩ C) x B" />
</task>
...
</tasksheet>

```

Fig. 8. XML specification of the task given in Figure 4

help or hint indicating the respective texts. Additionally, there are specific tags and attributes for every task type. In the following, we will illustrate the XML specifications by looking again at the examples presented in Section IV.

### B. Task Type Match

The value of the `type` tag for a match task is `match`. The type specific tag `answer` determines a solution pair. The attributes `left` and `right` specify the matching elements of the lists on the left hand side and on the right hand side, respectively. When generating the graphical task representation where the matching elements have to be connected by drawing a line between them, the order of the elements on the right hand side is generated randomly each time the task is invoked. In Fig. 6, the XML specification of the task presented in Fig. 2 is given.

### C. Truth Table

For a *truth table* task, the `type` tag value is `truthtable`. The type specific tag `var` determines a variable that is used in the propositional formula (given under `expr`) and its attribute `name` gives the name of the variable. The tag `expr` specifies under the attribute `string` a formula in propositional logic over the variables that have been introduced before. For this formula, ASTERIX automatically generates an interactive truth table with a field for every subformula and for every variable in all possible truth value assignments (cf. Fig. 3). In Fig. 7, the XML specification of the

task presented in Fig. 3 is given.

### D. Set Input

For a task requiring the determination of set elements the `type` tag value is `setinput` (cf. Fig. 8). The type specific tag `set` with its attributes `name` and `contents` defines the name of a set and the elements it contains. The tag `input` specifies the set whose elements have to be determined by the student; its attribute `name` defines a name for the set, and its attribute `defn` provides a definition for it using the sets introduced before and set operations like Cartesian product, union, intersection or set difference. The attribute `defnString` specifies a string that will be presented to the student; based on this representation, the student has to determine the elements of the set and to enter them in the respective input field. ASTERIX compares the student's input with the set expression given under `defn` that is automatically evaluated. In Fig. 8, the XML specification of the task presented in Fig. 4 is given.

### E. X of N Table

For an *X of N table* task, the `type` tag value is `XofNtable`. The type specific tag `value` determines a value that can be assigned to a variable. The attribute `id` contains an identification and the attribute `name` contains a name for the value. The tag `var` introduces a variable the student assigns values to; the attribute `name` contains a short text identifying it, and the attribute `correct` speci-

fies the list of correct answers.

From this, ASTERIX generates a table-structured representation; an example is given in Fig. 5. The names of the variables occur as identifiers for the columns, while the names of the corresponding values identify the rows of the table. For instance, the XML specification of the task given in Fig. 5 contains:

```
<task type="XofNtable">
  ...
  <value id="symm" name="symmetric" />
  <value id="refl" name="reflexive" />
  <value id="trans" name="transitive" />
  <var correct="refl" name="1." />
  <var correct="symm;refl" name="2." />
  <var correct="trans" name="3." />
</task>
```

## VII. CONCLUSIONS AND OUTLOOK

Especially in distance teaching, the process of supervising one's own learning progress by self-testing tasks can be supported by automated tools. The system ASTERIX was developed as an automatic learning support and self-testing tool for the course *Formal Foundations of Computer Science*, a first-year course in the B.Sc. program in Computer Science at the FernUniversität in Hagen. ASTERIX provides a range of different tasks in a variety of presentation formats. The program code for the interactive presentation and correction is generated automatically from high-level XML specifications.

The course *Formal Foundations of Computer Science* is taken regularly each year by about 1200 first-year students. During the current semester, ASTERIX is being offered to them for the first time. We will evaluate the students' feedback on the system at the end of the semester; the first statements we have received so far are quite promising.

A possible extension of ASTERIX is the connection to an intelligent tutor system (ITS) [Lel99]. With the internal availability of a tutor model e.g. the tips provided by ASTERIX can be adapted to the individual students and situations.

## REFERENCES

- [BPSM<sup>+</sup>04] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C, February 2004. <http://www.w3.org/TR/2004/REC-xml-20040204/>.
- [Lel99] R. Lelouche. Intelligent tutoring systems from birth to now. *Künstliche Intelligenz*, 13(4):5–11, November 1999.
- [LGH02] R. Lütticke, C. Gnörlich, and H. Helbig. VILAB - a virtual electronic laboratory for applied computer science. In *Proceedings of the Conference Networked Learning in a Global Environment*. ICSC Academic Press, Canada/The Netherlands, 2002.
- [LVU05] Lernraum Virtuelle Universität, Fernuniversität in Hagen, <http://www.fernuni-hagen.de/LVU/>. 2005.
- [WMKI<sup>+</sup>06] M. Widera, B. Messing, G. Kern-Isberner, M. Isberner, and C. Beierle. An extendable system for the specification and generation of interactive self-tests. In Z. Pan, R. Aylett, H. Diener, and X. Jin, editors, *Edutainment: Technology and Application*, Lecture Notes in Computer Science. Springer-Verlag, 2006. (to appear).