Fault-Tolerant Context-Based Interpretation of Mathematical Formulas

Helmut Horacek and Magdalena Wolska
Fachrichtung Informatik Fachrichtung Computerlinguistik
Universität des Saarlandes, Postfach 15 11 50, D-66041 Saarbrücken, Germany
{horacek@ags,magda@coli}.uni-sb.de

Abstract

We present a fault-tolerant formula interpreter that aims at finding plausibly intended, formally correct and contextually meaningful specifications from user statements containing formal inaccuracies.

1 Motivation

Strong efforts are invested in supporting scientific publication environments in formal areas such as mathematics, with powerful tools, including semantically-informed editors and proof-checkers. Similarly, several research activities aim at teaching reasoning and problem-solving skills in tutorial systems. In both settings, effective communication is frequently hindered by sloppiness and low-level errors in specifying formulas on behalf of the user, resp. student. Hence, not only diagnosing errors in formulas, but also building hypotheses for correcting errors is an important issue, a difficult subproblem in mathematical knowledge management according to [2].

2 Our Working Context

Our interest in fault-tolerant formula interpretation originates from work on a tutorial system which teaches proving mathematical theorems [1]. In this context, a corpus of written interactions with a simulated computer-based tutor in the domain of naive set theory has been collected [5]. In this corpus, lengthy clarification subdialogs are often carried out to resolve low-level issues, which one wants to avoid since they distract the student from the higher-level tutorial goal.

In Table 1, we present examples of flawed formulas from the corpus. In (1), a segmentation error is shown: not only a space between the operator symbol P and identifier C, but also parentheses are missing. (2) is an example of a typing error, where an operator symbol p has been used in place of the identifier b. In (3), the types of arguments of the main operator are invalid. (4) shows a well-formed formula but it is not relevant in the context of the task: a stronger assertion about an intersection rather than union of the sets on the right-hand side of the equation is required[1]. In (5), similarly to (4), a weaker operator of set inclusion ($\subseteq$) rather than equality is correct. Finally, (6) and (7) are examples of commonly confused relations of *subset* and *membership*.

[1]This is the only kind of error which we do not address here.

Table 1: Examples of flawed formulas from the corpus

	Example Formula	*Error Category*
(1)	$P((A \cup C) \cap (B \cup C)) = PC \cup (A \cap B)$	3
(2)	$(p \cap a) \in P(a \cap b)$	2
(3)	$(x \in b) \notin A \qquad x \subseteq K(A)$	2
(4)	$P((A \cup C) \cap (B \cup C)) = P(A \cup C) \cup P(B \cup C)$	1
(5)	$P((A \cap B) \cup C) = P(A \cap B) \cup P(C)$	1
(6)	$(A \cap B) \subseteq P(A \cap B)$	1
(7)	If $A \subseteq K(B)$ then $A \notin B$	2

In the context of the tutorial system mentioned above, the proof development environment ΩMEGA [3] is used to check validity of (possibly ambiguous) proof-step interpretations and consistency with the proof context. Communication with ΩMEGA is mediated by a *proof manager* whose task is to build and maintain a representation of the constructed proof. The proof manager also maintains a discourse memory of identifiers, operators, and their properties (e.g., type, arity).

3 Error Categories and Correction Attempts

Finding purposeful changes in a formula that aim at building a corrected and possibly intended version of that formula is substantially guided by its *correctness state*. We distinguish three categories of errors: 1) *logical*, 2) *type*, and 3) *structural* errors. A formula analyzer may find it impossible to build a formula analysis tree on the basis of the defined constructors (category 3), or to resolve a type mismatch in an analysis tree built successfully (category 2). Only for correctly analyzed formulas, consulting the proof manager yields a distinction between a formula expressing a *wrong* (category 1) or a correct statement. In case of an error, attempts are undertaken to remedy the error by applying local and contextually justified modifications to the formula. In order to obtain meaningful changes, we associate a set of replacement rules with each error category, aiming to achieve an improvement of at least one category level. Each of these rules is justified by evidence about the plausibility of the given error occurring in the domain, the nature of the task, and user capabilities. Some rules developed on the basis of errors observed in the corpus, their associated error categories, and examples are illustrated in Table 2. Testing applications of these rules is distributed between formula analysis and formula modifying algorithms.

Table 2: Replacement rules attempting to remedy errors

Replacement Rules	*Error Categories*	*Examples*
dual operator	1	$\cap \Leftrightarrow \cup$, $\subset\Leftrightarrow\supset$, $\subseteq\Leftrightarrow\supseteq$
stronger/weaker operator	1	$\supset\Leftrightarrow\supseteq$, $\subseteq\Leftrightarrow=$, $\supseteq\Leftrightarrow=$
confused operator	1,2	$\subset\Leftrightarrow\in$, $K \Leftrightarrow P$
confused identifi er	1,2	$a \Leftrightarrow b$, $p \Leftrightarrow b$, $p \Leftrightarrow a$
delete character	2	$PC \Rightarrow P$, $PC \Rightarrow C$
insert blank or brackets	3	$PC \Rightarrow P\ C$, $PC \Rightarrow P(C)$

4 The Enhanced Formula Analysis

Formula analysis is part of an input interpretation component whose task is to produce a representation that can be further analysed by the proof manager. The formula modification procedure presented here extends the method of parsing mathematical expressions embedded within natural language text [4]. Extended formula analysis consists of: (1) identification of mathematical expressions within the word-tokenized text; (2) verification of the identified sequence as to syntactic validity and correction, in case of a parentheses mismatch or further tokenization needed; [2] (3) parsing of the expression.

1. The mathematical expression tagger uses a list of operation and identifier symbols. Identification of mathematical expressions is based on: single character tokens (including parenthesis), multiple-character tokens consisting only of known relevant characters, mathematical symbol unicodes, and new-line characters. Multiple-character candidate tokens are further segmented into operators and identifiers by inserting the missing spaces.
2. Once a candidate string is identified, "chains" of formulas are separated into individual segments, and parentheses match is verified. Missing parentheses are inserted while obeying to the following preferences: (i) provide parentheses for operators that require bracketed arguments, (ii) avoid redundant parentheses (i.e. double parentheses around the same substring).
3. Syntactically correct candidate strings are parsed by a tree-building algorithm accessing standard requisite information (e.g. list of operators and their precedence).

The output is a set of formula trees with nodes marked as to type compatibility and bracketing where applicable.

5 The Formula Modifying Algorithm

The algorithm incrementally generates formula modifications in a best-first fashion and tests their impact on resolving the original error, within specified resource limitations, that is, a maximum number of modified formulas and a time limit:

1. If the formula analyzer has produced a set of interpretations by applying rules addressing syntactic errors, these modifications are taken as successors of the original formula. Otherwise, successors are generated for the formula considered best by applying replacement rules in the category associated with the error reported for the original formula, unless the error is already resolved.
2. If resources are not exceeded, the most promising successor of a formula generated so far is promoted into the new best one, and its correctness state is assessed by the proof manager, continuing at step 1. Otherwise, an ordered list of modified formulas examined is returned.

Preferred orderings among created formulas are established by the error-related category and a similarity-assessing function, the former dominating the latter. The assessment function combines the number of replacement rules applied and the differences to the structurally most similar formula in the context, which comprises the set of formulas consisting of the goal expression, the previous proof step and possible follow-up steps. This function is approximated by simply adding the number of rule applications to the number of operators and variables appearing in both formulas compared, multiple occurrences counted according to their frequency.

6 Preliminary Results

Finally, we give analyses of some examples from Table 1. For utterance (1), we get two interpretations, depending on whether the formula analyzer separates PC and inserts parentheses (2 alternatives), or whether it reports a type error, flagging PC. In the latter case, replacing PC by any type compatible identifier yields an error of category 1. The same holds for the parenthesis insertion with narrower scope, $P(C)$, but the other alternative, $P(C \cup (A \cap B))$ yields no error and wins. Example (2) is relatively simple, since only replacing the occurrence of p, flagged as a type clash, is to be changed. Only replacements by a and b yield no error, b winning over a since it gets a better context agreement score. In utterance (5), many rules are applicable. Changing variables gives lower agreement scores than changing an operator to its dual counterpart, but all these choices remain within error category 1. Only replacing "$=$" by "$\supseteq$" resolves the error.

In the future, we intend to look into other subdomains of mathematics and to incorporate results of formula modifications in tutorial strategies, e.g., asking for clarification, offering a small set of modified formulas. Similar interactions could be also tailored for scientific publication environments.

[2]This implements replacement rules of category 3. Parentheses correction may yield multiple plausible interpretations. From this stage on, further processing is applied to all the interpretations.

References

[1] C. Benzm¨uller et al. Tutorial Dialogs on Mathematical Proofs. In Proc. of *IJCAI Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, pages 12–22, Acapulco, Mexico, 2003.

[2] B. Buchberger and K. Nakagawa. Mathematical Knowledge Editor: A Research Plan. SFB Report 2004-40, Johannes Kepler University Linz, 2004.

[3] J. Siekmann et al. Proof Development with ΩMEGA. In Proc. of *CADE-02*, pages 144–149, Copenhagen, Denmark, 2002.

[4] M. Wolska and I. Kruijff-Korbayová. Analysis of Mixed Natural and Symbolic Language Input in Mathematical Dialogs. In Proc. of *ACL-04*, pages 24–32, 2004.

[5] M. Wolska et al. An Annotated Corpus of Tutorial Dialogs on Mathematical Theorem Proving. In Proc. of *LREC-04*, pages 1007–1010, Lisbon, Portugal, 2004.