Testing Mizar User Interactivity in a University-level Introductory Course on Foundations of Mathematics

Adam Naumowicz

Institute of Informatics
University of Białystok, Poland
adamn@mizar.org

FMM 2019 at CICM 2019, Prague, July 8, 2019

MIZAR interactivity

MIZAR interactivity

- MIZAR has been created to support developing students' mathematical reasoning skills by
  interacting with intelligent computer software capable of creating and proof checking formal mathematics

MIZAR interactivity

- MIZAR has been created to support developing students' mathematical reasoning skills by
  interacting with intelligent computer software capable of creating and proof checking formal mathematics
- MIZAR is not a typical interactive theorem prover
  - its mode of interaction resembles various source code compilers
  - processing a complete input file in a series of lexical and semantic passes
  - more adequate to batch processing multiple or massive data rather than real-time human-computer interplay

MIZAR interactivity

- MIZAR has been created to support developing students' mathematical reasoning skills by
  interacting with intelligent computer software capable of creating and proof checking formal mathematics
- MIZAR is not a typical interactive theorem prover
  - its mode of interaction resembles various source code compilers
  - processing a complete input file in a series of lexical and semantic passes
  - more adequate to batch processing multiple or massive data rather than real-time human-computer interplay
- tailored with a dedicated user interface (J. Urban's MIZAR Mode for Emacs) the system has a fairly interactive user experience

MIZAR interactivity

- MIZAR has been created to support developing students' mathematical reasoning skills by
  interacting with intelligent computer software capable of creating and proof checking formal mathematics
- MIZAR is not a typical interactive theorem prover
  - its mode of interaction resembles various source code compilers
  - processing a complete input file in a series of lexical and semantic passes
  - more adequate to batch processing multiple or massive data rather than real-time human-computer interplay
- tailored with a dedicated user interface (J. Urban's MIZAR Mode for Emacs) the system has a fairly interactive user experience
  - flexible gap-filling work on a plain text human-readable proof sketch with the system pointing out proof steps that require justification

MIZAR interactivity

- MIZAR has been created to support developing students' mathematical reasoning skills by
  interacting with intelligent computer software capable of creating and proof checking formal mathematics
- MIZAR is not a typical interactive theorem prover
  - its mode of interaction resembles various source code compilers
  - processing a complete input file in a series of lexical and semantic passes
  - more adequate to batch processing multiple or massive data rather than real-time human-computer interplay
- tailored with a dedicated user interface (J. Urban's MIZAR Mode for Emacs) the system has a fairly interactive user experience
  - flexible gap-filling work on a plain text human-readable proof sketch with the system pointing out proof steps that require justification
  - users may focus on any reported gap in the proof with no imposed order (as soon as the whole text has been parsed)

Long record of educational use

Long record of educational use

- various versions of MIZAR have been applied in many educational settings

Long record of educational use

- various versions of MIZAR have been applied in many educational settings
- most typically at the university level to support introductory as well as advanced courses

Long record of educational use

- various versions of MIZAR have been applied in many educational settings
- most typically at the university level to support introductory as well as advanced courses
- current context:
  - results based on the data collected during a one-semester university-level introductory mathematical course for computer science undergraduate students
  - the students hadn't had any previous contact with MIZAR

Long record of educational use

- various versions of MIZAR have been applied in many educational settings
- most typically at the university level to support introductory as well as advanced courses
- current context:
  - results based on the data collected during a one-semester university-level introductory mathematical course for computer science undergraduate students
  - the students hadn't had any previous contact with MIZAR
  - *their workings can approximate the problems typically encountered by* **all inexperienced users**

Course setting

Course setting

- course on foundations of mathematics for computer science undergraduates at the University of Bialystok in Poland

Course setting

- course on foundations of mathematics for computer science undergraduates at the University of Bialystok in Poland
- students provided with a restricted environment containing definitions of the needed notions only

Course setting

- course on foundations of mathematics for computer science undergraduates at the University of Bialystok in Poland
- students provided with a restricted environment containing definitions of the needed notions only
- the subject notions included elementary set operations, binary relations and natural numbers with induction

Examples of tasks

```
Y c= X implies Y\/X=X
proof
        assume z4: Y c= X;
        thus Y\/X c= X
        proof
                let x;
                assume x in Y\/X;
                then x in Y or x in X by ENUMSET: def 6;
                then x in X or x in X by z4, ENUMSET: def 10;
                hence x in X;
        end;
        thus X c= Y\/X
        proof
                let x;
                assume x in X;
                then x in Y or x in X ;
                hence x in Y\/X by ENUMSET: def 6;
        end;
end;
```

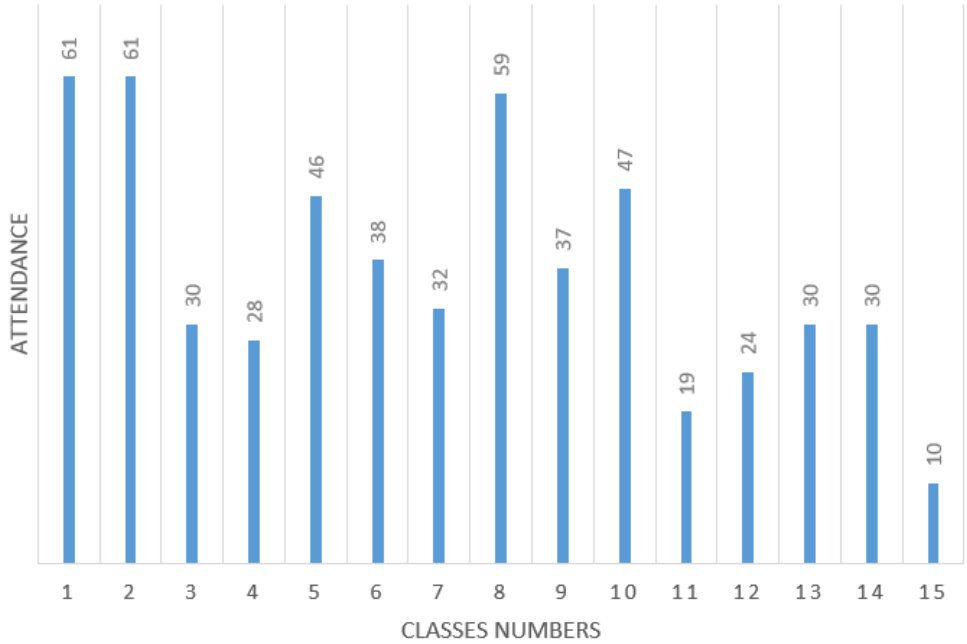
Examples of tasks

```
R is transitive implies R*R c=R
proof
        assume z2:R is transitive;
        thus  R*R c=R
        proof
                let x,y ;
                assume [x,y] in R*R;
                then ex z st ([x,z] in R & [z,y] in R) by RELATION: def 7;
                then consider z such that z1:([x,z] in R & [z,y] in R);
                ([x,z] in R & [z,y] in R) by z1;
                hence [x,y] in R by RELATION: def 12,z2;
        end;
end;
```

Examples of tasks

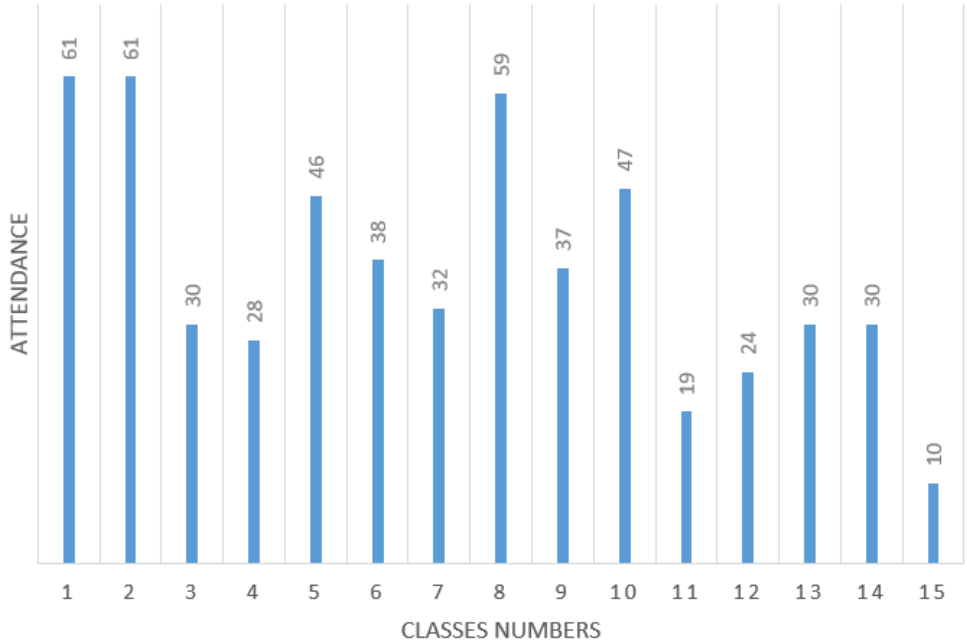
```
9 divides 10|^n-1
proof
        defpred P[Nat] means 9 divides 10|^$1-1;
        a:P[0]
        proof
                10|^0-1=1-1 by NEWTON: 9
                .=0
                .=9*0;
                hence thesis by INT_1: def 9;
        end;
        b:for n holds P[n] implies P[n+1]
        proof
                let n;
                assume P[n]; :: 9 divides 10|^n-1
                then consider j be Integer such that a1:10|^n-1 = 9*j by INT_1: def 9;
                10|^(n+1)-1 = 10|^n * 10 - 1 by NEWTON: 11
                .= (10|^n-1)*10 +10 -1
                .= 9*j*10  +10-1 by a1
                .= 9*j*10 + 9
                .=9*(j*10+1);
                hence thesis by INT_1: def 9;
        end;
        for j holds P[j] from NAT_1: sch 2(a,b);
        hence thesis;
end;
```

Class attendance statistics during the semester



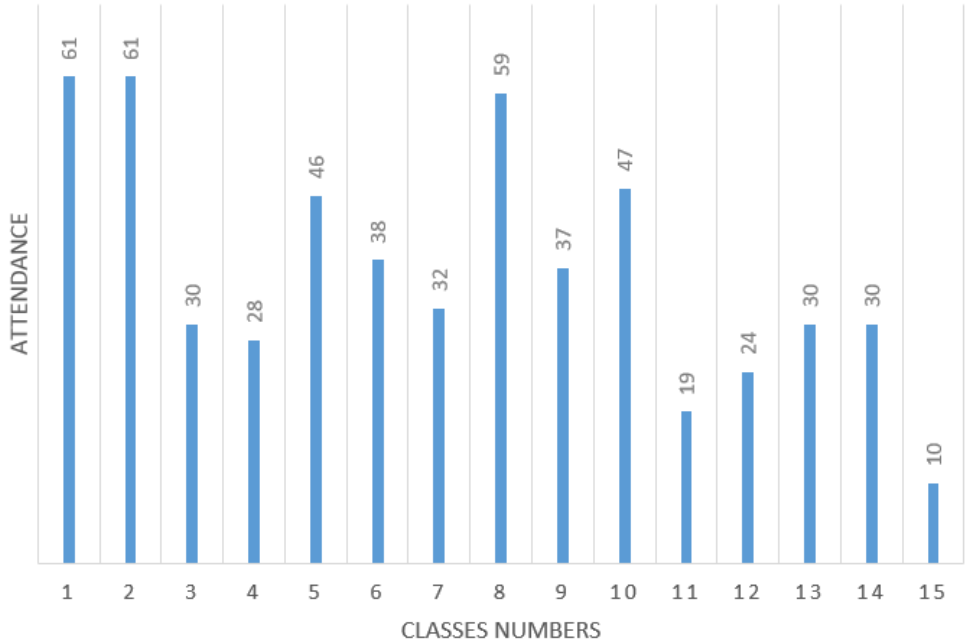
Class attendance statistics during the semester

- 70 students enrolled to this course for their first winter semester (15 weeks with one 90-minute class each week)



Class attendance statistics during the semester

- 70 students enrolled to this course for their first winter semester (15 weeks with one 90-minute class each week)
- the course had a rather low overall attendance rate (52.7%), with a typical peak at the beginning, and later in the middle of the semester (classes 5-10), which can be attributed to mid-semester activities (tests)



Recording the interactions

Recording the interactions

- students worked on local computers with the necessary database already installed

Recording the interactions

- students worked on local computers with the necessary database already installed
- task sets were provided together in a ready-made file with a complete environment, so the students' job was just to complete the missing proofs

Recording the interactions

- students worked on local computers with the necessary database already installed
- task sets were provided together in a ready-made file with a complete environment, so the students' job was just to complete the missing proofs
- whenever a student typed some input and called the MIZAR verifier from within the Emacs editor, a customized MIZAR mode recorded relevant data in a special log file

Recording the interactions

Recording the interactions

- customized `mizar.el` file

```
(defun mizar-it (&optional util noqr compil silent forceacc noacc options)
"Run mizar verifier on the text in the current .miz buffer.
Show the result in buffer *mizar-output*.
In interactive use, a prefix argument directs this command
to read verifier options from the minibuffer.

...
          (setq antime (car (current-time)))
          (setq antimf (car (cdr (current-time))))
          (write-region (concat "\n\n::: " (format-time-string "%D %T " nil)
            (number-to-string antime) " " (number-to-string antimf) "\n") nil
                    (concat (file-name-sans-extension(buffer-file-name)) ".abc") t )
          (write-region nil nil
                    (concat (file-name-sans-extension(buffer-file-name)) ".abc") t )
                  (setq last-verification-date (file-mtime (buffer-file-name)))
                  (mizar-handle-output)
                  (mizar-show-errors)))
                ))))))
```

Recording the interactions

- customized `mizar.el` file

```
(defun mizar-it (&optional util noqr compil silent forceacc noacc options)
"Run mizar verifier on the text in the current .miz buffer.
Show the result in buffer *mizar-output*.
In interactive use, a prefix argument directs this command
to read verifier options from the minibuffer.

...
          (setq antime (car (current-time)))
          (setq antimf (car (cdr (current-time))))
          (write-region (concat "\n\n::: " (format-time-string "%D %T " nil)
            (number-to-string antime) " " (number-to-string antimf) "\n") nil
                    (concat (file-name-sans-extension(buffer-file-name)) ".abc") t )
          (write-region nil nil
                    (concat (file-name-sans-extension(buffer-file-name)) ".abc") t )
                  (setq last-verification-date (file-mtime (buffer-file-name)))
                  (mizar-handle-output)
                  (mizar-show-errors)))
                ))))))
```

- the time when the verifier was called and the complete input files were recorded including any error messages reported by the checker

Recording the interactions

- customized `mizar.el` file

```
(defun mizar-it (&optional util noqr compil silent forceacc noacc options)
"Run mizar verifier on the text in the current .miz buffer.
Show the result in buffer *mizar-output*.
In interactive use, a prefix argument directs this command
to read verifier options from the minibuffer.

...
          (setq antime (car (current-time)))
          (setq antimf (car (cdr (current-time))))
          (write-region (concat "\n\n::: " (format-time-string "%D %T " nil)
            (number-to-string antime) " " (number-to-string antimf) "\n") nil
                    (concat (file-name-sans-extension(buffer-file-name)) ".abc") t )
          (write-region nil nil
                    (concat (file-name-sans-extension(buffer-file-name)) ".abc") t )
                  (setq last-verification-date (file-mtime (buffer-file-name)))
                  (mizar-handle-output)
                  (mizar-show-errors)))
                ))))))
```

- the time when the verifier was called and the complete input files were recorded including any error messages reported by the checker
- at the end of each session students uploaded their log files to their individual accounts on the teachers' server

Recording the interactions

Recording the interactions

- during the experiment, the solutions of 553 task sets were recorded

Recording the interactions

- during the experiment, the solutions of 553 task sets were recorded
- the data revealed that overall the group made 24326 calls to the verifier, which accounts for c.a. 43.99 calls per user per class session

Recording the interactions

- during the experiment, the solutions of 553 task sets were recorded
- the data revealed that overall the group made 24326 calls to the verifier, which accounts for c.a. 43.99 calls per user per class session
- with such simple tasks, the time needed for the verifier to check the input file is negligible (usually less than 1s.)

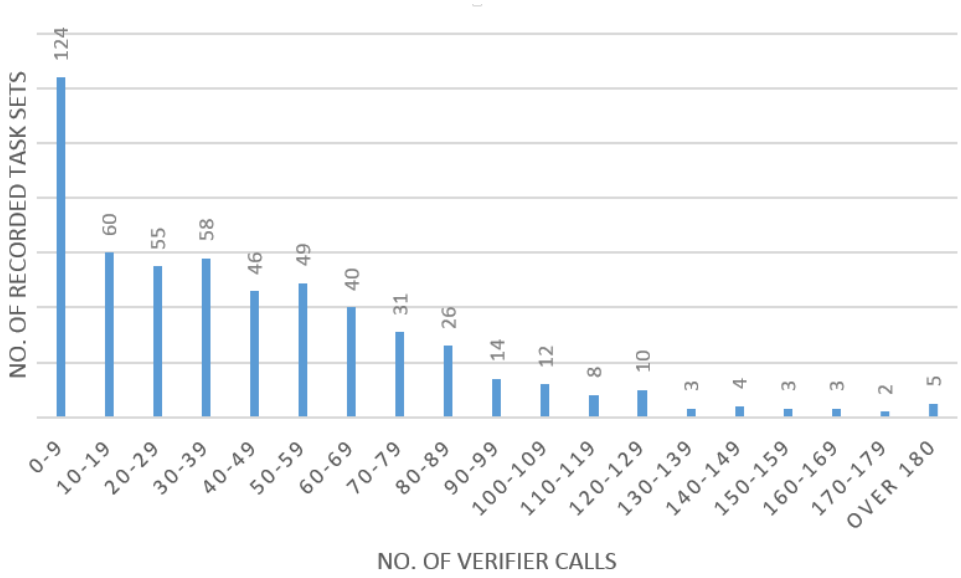
Recording the interactions

- during the experiment, the solutions of 553 task sets were recorded
- the data revealed that overall the group made 24326 calls to the verifier, which accounts for c.a. 43.99 calls per user per class session
- with such simple tasks, the time needed for the verifier to check the input file is negligible (usually less than 1s.)
- a typical call being made more or less every two minutes after some thinking and typing

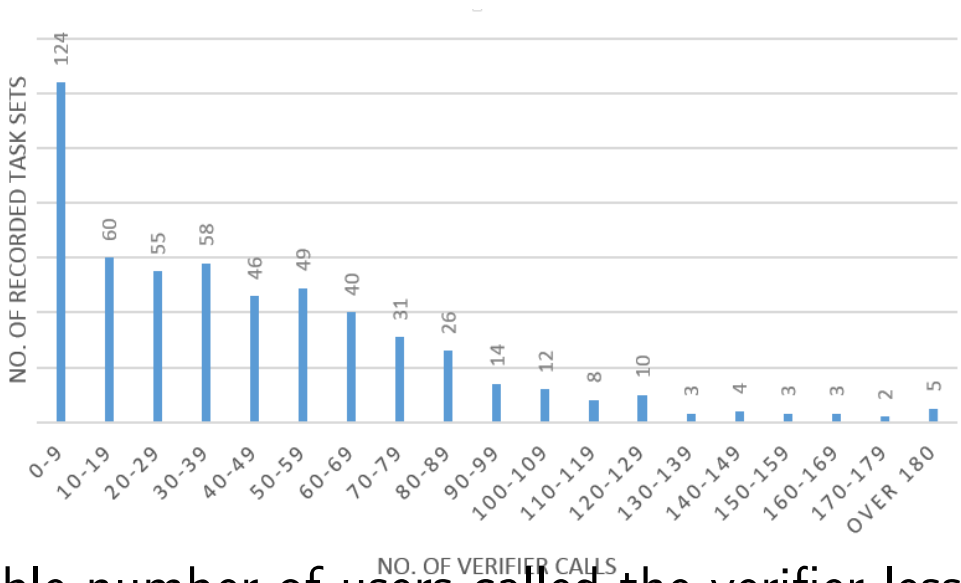
Recording the interactions

- during the experiment, the solutions of 553 task sets were recorded
- the data revealed that overall the group made 24326 calls to the verifier, which accounts for c.a. 43.99 calls per user per class session
- with such simple tasks, the time needed for the verifier to check the input file is negligible (usually less than 1s.)
- a typical call being made more or less every two minutes after some thinking and typing
- the exact data shows that individual interaction strategies varied among the students

The number of verifier runs invoked by the students

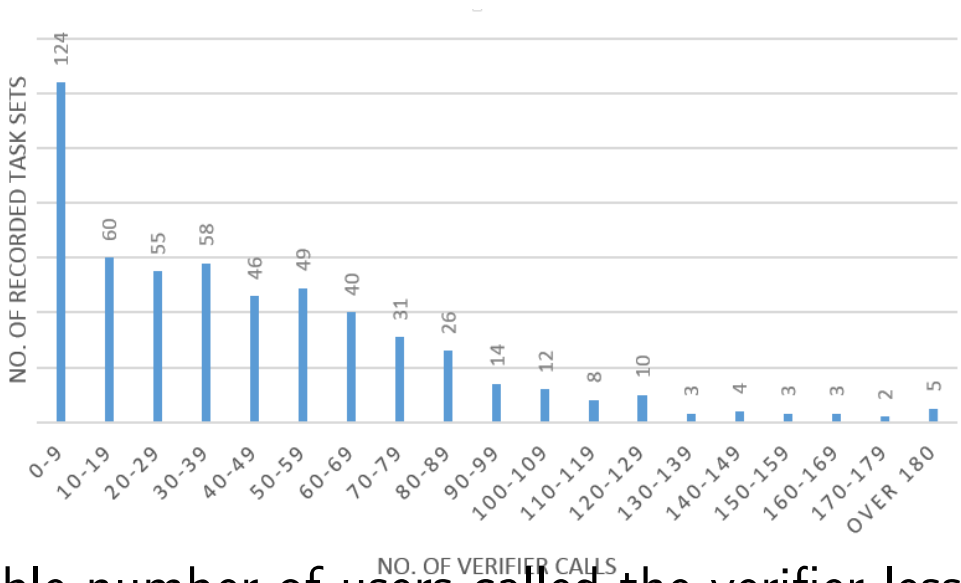


The number of verifier runs invoked by the students



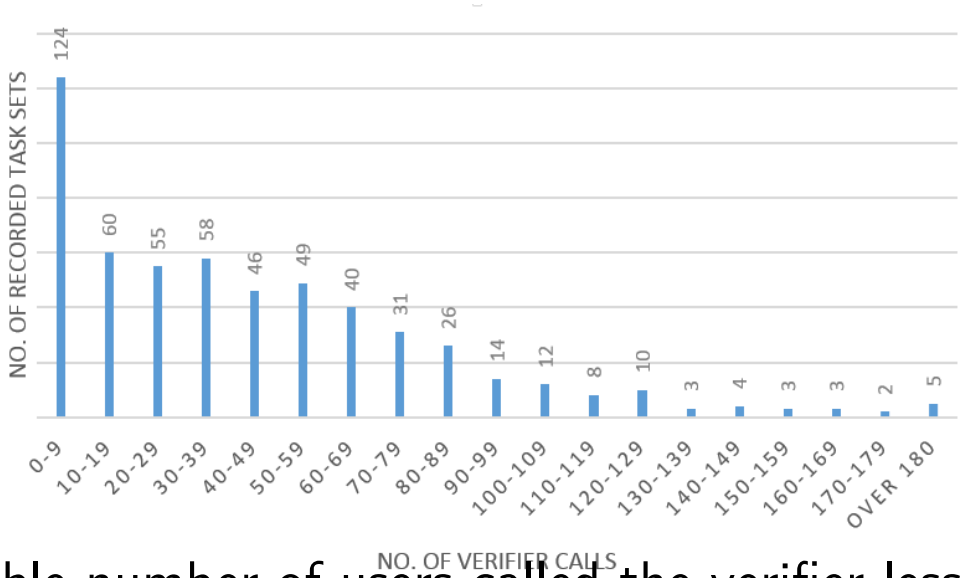
- a considerable number of users called the verifier less than 10 times during a session, clearly in preference of typing longer chunks of text rather than checking the text frequently

The number of verifier runs invoked by the students



- a considerable number of users called the verifier less than 10 times during a session, clearly in preference of typing longer chunks of text rather than checking the text frequently
- there were also task set recordings showing as many as 264 calls to the verifier during one session!

The number of verifier runs invoked by the students



- a considerable number of users called the verifier less than 10 times during a session, clearly in preference of typing longer chunks of text rather than checking the text frequently
- there were also task set recordings showing as many as 264 calls to the verifier during one session!
- that way of interaction indicates more of a guessing approach into finding the proof and using the proving capabilities of MIZAR to construct the proof semi-automatically

Kinds of errors reported

Kinds of errors reported

- The source codes of developed tasks contained 107 different errors reported (out of total 496 documented error messages issued by the MIZAR verifier)

Kinds of errors reported

- The source codes of developed tasks contained 107 different errors reported (out of total 496 documented error messages issued by the MIZAR verifier)
  - frequency

Kinds of errors reported

- The source codes of developed tasks contained 107 different errors reported (out of total 496 documented error messages issued by the MIZAR verifier)
  - frequency
  - error messages' adequacy

Top 20 most frequent error codes and messages

Occurrences	Code	Message
17017	4	This inference is not accepted
14320	70	Something remains to be proved
4519	395	Justification expected
3719	396	Formula expected
3137	51	Invalid conclusion
2578	330	Unexpected end of an item (perhaps ";" missing)
2342	321	Predicate symbol or "is" expected
1969	391	Incorrect beginning of a text item
1919	215	No pairing "end" for this word
1905	214	"end" missing
1712	52	Invalid assumption
1662	143	No implicit qualification
1582	55	Invalid generalization
1436	144	Unknown label
1064	131	No reserved type for a variable, free in the default type
942	164	Nothing to link
733	60	Something remains to be proved in this case
633	165	Unknown functor format
620	216	Unexpected "end"
603	153	Unknown predicate format

Quick observations

Quick observations

- the list is opened with the most common error *4 meaning that a given step needs to be justified with more information to be accepted by the checker as an obvious consequence of available references

Quick observations

- the list is opened with the most common error *4 meaning that a given step needs to be justified with more information to be accepted by the checker as an obvious consequence of available references
- there are some errors reported by the scanner, parser and analyzer modules

Quick observations

- the list is opened with the most common error *4 meaning that a given step needs to be justified with more information to be accepted by the checker as an obvious consequence of available references
- there are some errors reported by the scanner, parser and analyzer modules
- their frequent occurrences in the students' tasks indicate e.g. that the description might need less cryptic forms

Quick observations

- the list is opened with the most common error *4 meaning that a given step needs to be justified with more information to be accepted by the checker as an obvious consequence of available references
- there are some errors reported by the scanner, parser and analyzer modules
- their frequent occurrences in the students' tasks indicate e.g. that the description might need less cryptic forms
- this sort of feedback is essential in showing which MIZAR constructs are less intuitive from the perspective of a new user and therefore require more explanation when they are being introduced to the students

Quick observations

- the list is opened with the most common error *4 meaning that a given step needs to be justified with more information to be accepted by the checker as an obvious consequence of available references
- there are some errors reported by the scanner, parser and analyzer modules
- their frequent occurrences in the students' tasks indicate e.g. that the description might need less cryptic forms
- this sort of feedback is essential in showing which MIZAR constructs are less intuitive from the perspective of a new user and therefore require more explanation when they are being introduced to the students
- anticipating potential problems based on their frequency may help minimizing the number of common errors encountered by students if they have been forewarned about them

More observations

[1] http://mizar.uwb.edu.pl/~softadm/linking/

More observations

- some of the issues can only be resolved by extending/changing a bit of the language's grammar

[1] http://mizar.uwb.edu.pl/~softadm/linking/

More observations

- some of the issues can only be resolved by extending/changing a bit of the language's grammar
- e.g. a typical error results from the restriction of straightforward linking to a statements in a previous line using the keyword `then`; inn consequence, the error *164 (position 16 in Table) is reported whenever making such a link has been tried in the context of compound conditions

[1] http://mizar.uwb.edu.pl/~softadm/linking/

More observations

- some of the issues can only be resolved by extending/changing a bit of the language's grammar
- e.g. a typical error results from the restriction of straightforward linking to a statements in a previous line using the keyword `then`; inn consequence, the error *164 (position 16 in Table) is reported whenever making such a link has been tried in the context of compound conditions
- an experimental version of the MIZAR software reflecting the relaxed syntax exists now and can be used for evaluating the change e.g. in some student classes[1]

[1] http://mizar.uwb.edu.pl/~softadm/linking/

More observations

- some of the issues can only be resolved by extending/changing a bit of the language's grammar
- e.g. a typical error results from the restriction of straightforward linking to a statements in a previous line using the keyword `then`; inn consequence, the error *164 (position 16 in Table) is reported whenever making such a link has been tried in the context of compound conditions
- an experimental version of the MIZAR software reflecting the relaxed syntax exists now and can be used for evaluating the change e.g. in some student classes[1]
- it should be noted, however, that until the new syntax is generally accepted, articles written in this new "dialect" could not be accepted for inclusion to the MML

[1] http://mizar.uwb.edu.pl/~softadm/linking/

Conclusions

Conclusions

- the analysis of data collected during students' courses provides valuable information which can be used to improve the user experience of future versions of the system

Conclusions

- the analysis of data collected during students' courses provides valuable information which can be used to improve the user experience of future versions of the system
- the statistics of frequently occurring errors encountered by the students are a good approximation of input typical to any novice-user texts in general

Conclusions

- the analysis of data collected during students' courses provides valuable information which can be used to improve the user experience of future versions of the system
- the statistics of frequently occurring errors encountered by the students are a good approximation of input typical to any novice-user texts in general
- a further analysis is needed to identify typical user scenarios and provide ways to handle them in a more user-friendly manner