VSCode Extension for the Web and Coding Assistance for the Mizar Language

Haruka Miyata, Ryutaro Matsumoto and Kazuhisa Nakasho

August 28, 2023

VSCode Extension for the Web and Coding Assistance for the Mizar Language

Haruka Miyata[1], Ryutaro Matsumoto[1] and Kazuhisa Nakasho[1,*]

[1]*Yamaguchi University, 2-16-1 Tokiwa-dai, Ube, Japan*

Abstract

This paper discusses the enhancements we have made to the Mizar Extension for Visual Studio Code (VSCode). The first part of the paper explores the creation of the Mizar Server and client-side infrastructure, developed to support the web version of the Mizar Extension for VSCode. The latter part provides an update on the development of the formatter function and reports on the progress of our research into coding assistance using ChatGPT.

Keywords

Mizar, VSCode for the Web, server side verification, code formatter, coding assistance, ChatGPT

1. Introduction

Development of the Mizar Extension for VSCode [1] began in 2020. This extension seeks to offer an intuitive Mizar IDE for novices. It includes features such as syntax highlighting, execution and stop capabilities for Mizar commands, as well as definition jump and hover functions to reference cited theorems and definitions.

In this study, we have adapted the previously mentioned extension for use with VSCode on the Web, creating a Mizar IDE that operates within a web browser. As the Mizar verification program does not support WebAssembly, we developed a new web application that executes Mizar commands on a remote server. This feature alleviates the need for Mizar users to maintain a development environment on their personal computers or to update the Mizar Mathematical Library (MML). To improve coding assistance, we established Mizar coding conventions and implemented a code formatter in accordance with these conventions. Furthermore, as part of an exploratory study into coding assistance using generative AI, we examined code completion using ChatGPT. The source codes of the programs being developed in this study, as well as the video captures, are available on GitHub[1][2]. An architectural diagram of the system developed in this study is presented in Figure 1.

[1]https://github.com/mimosa-project/emformat/

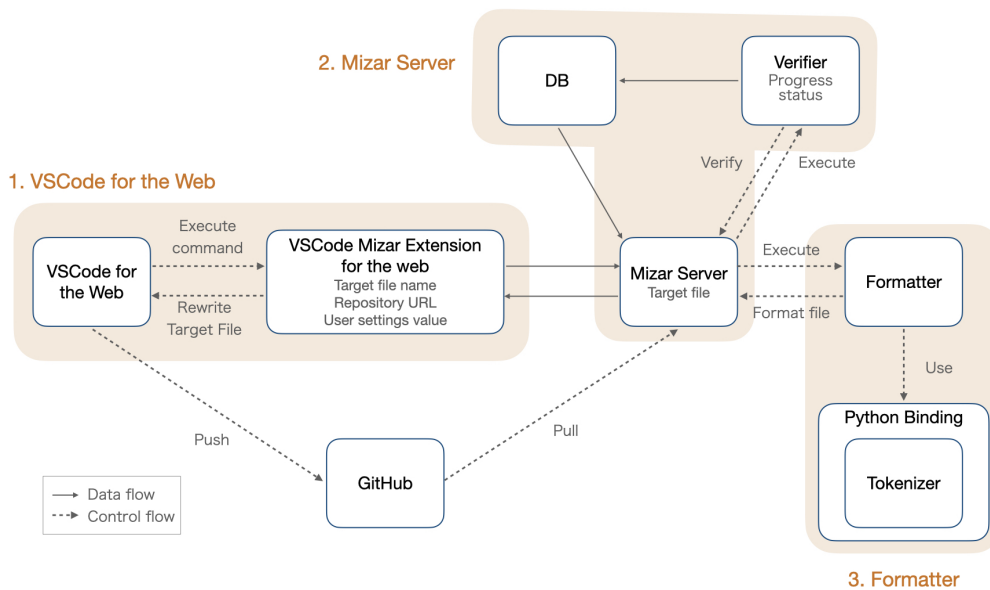[2]https://github.com/mhrwh/emvscode-web/

Figure 1: Architectural diagram

2. VSCode Extension for the Web

2.1. VSCode for the Web

VSCode for the Web[3] is a cloud-based code editor developed by Microsoft that operates within a web browser. It allows users to create projects, edit code, and manage versions all through a web interface. By extending the VSCode experience to the web, it eliminates the need for installation and configuration, providing a uniform development environment across all devices. Users can tailor the editor to their liking and incorporate extensions. In addition, multiple users have the capability to simultaneously share and modify a project in real time.

2.2. Related Works on Web-based IDE for ITPs

jsCoq[4] is a system designed to run Coq in a web browser. It offers IDE features such as code highlighting, auto-completion, and indentation, all of which facilitate efficient editing of Coq scripts. Moreover, jsCoq can integrate Coq scripts within HTML documents and exhibit the advancement of proofs in real-time [2].

Clide is a web interface for Isabelle that expands on the CodeMirror editor, constructed using the Isabelle/PIDE framework [3]. It offers version control through Git, enables real-time previewing of proofs, facilitates proof debugging, among other features [4].

Lean Web Editor[5] is a web-based IDE offering an interactive proof environment for Lean 3 [5].

---

[3]https://vscode.dev/
[4]https://jscoq.github.io/
[5]https://leanprover-community.github.io/lean-web-editor/

It boasts useful features including syntax highlighting, auto-completion, and error checking, facilitating efficient development using the Lean 3 language. It also actively checks user input and presents error and warning messages. In the same vein, Lean 4 Web[6] serves as a web IDE for Lean 4 language. While Lean Web Editor operates the Lean 3 server in a browser, Lean 4 Web runs the Lean 4 server on a web server. In order to mitigate security risks, the Lean server operates within a Docker container and is safeguarded by security mechanisms such as gVisor.

2.3. Web Support for Extension Mizar for VSCode

We have transferred all features from the existing Mizar Extension to VSCode for the Web. The modifications made to attain web compatibility are detailed below.

The Mizar command function performs verification and improvement operations on Mizar articles within VSCode. The Mizar Extension is designed under the assumption that the Mizar system is installed in the local environment, as command execution takes place on the user's PC. To adapt this functionality for the web version, we created a web application known as the "Mizar Server" that runs Mizar commands on a remote server. The Mizar Server handles execution of Mizar commands, provision of verification results, and code formatting. Both the Mizar Server and VSCode for the Web share source code via GitHub, ensuring synchronization of their development environments.

A sequence diagram of the verifier command execution process is presented in Figure 2. Upon receipt of a request, the Mizar Server executes a command program like the verifier and promptly returns an ID tied to the request. The web extension periodically communicates with the Mizar Server, using the received ID as a parameter, to check on the verification progress. If the verification is ongoing, its progress is displayed on VSCode for the Web. Once the verification is complete, users are alerted if any errors have been found.

The definition jump and hover display functions serve as mechanisms to reference cited theorems and definitions. These functions are accomplished by the web extension through retrieving information from abstract files stored on GitHub. A screenshot depicting the hover display and command execution functions is provided in Figure 3.

3. Coding Assistance

Numerous programming languages have established style guides and best practices to enhance code readability and maintainability. Coding assistance tools, including formatters, linters, and static type checkers, are provided to guarantee that source code adheres to these conventions. Recently, coding assistant tools employing generative AI, such as GitHub Copilot[6, 7], have made their debut, significantly contributing to increased development productivity.

3.1. Related Works on Code Formatters for ITPs

One example of code formatters in ITPs is Mizar Mode [8], an Emacs authoring environment tailored for the Mizar language. Mizar Mode offers the following source code formatting functions:
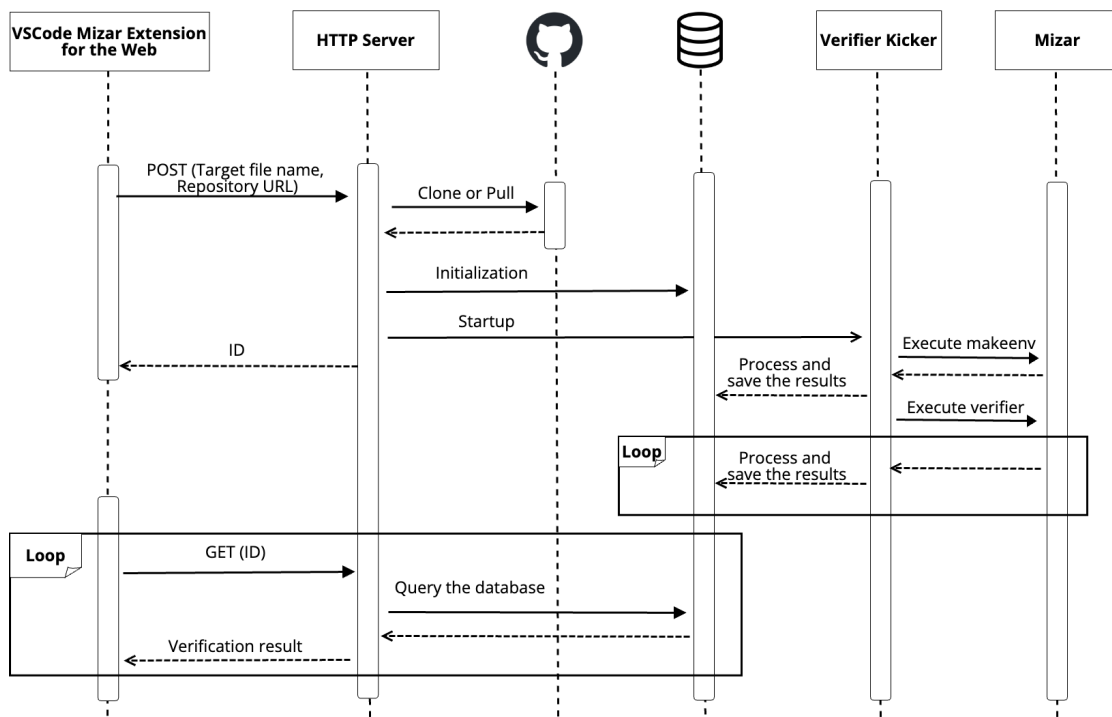
---

[6]https://lean.math.hhu.de/

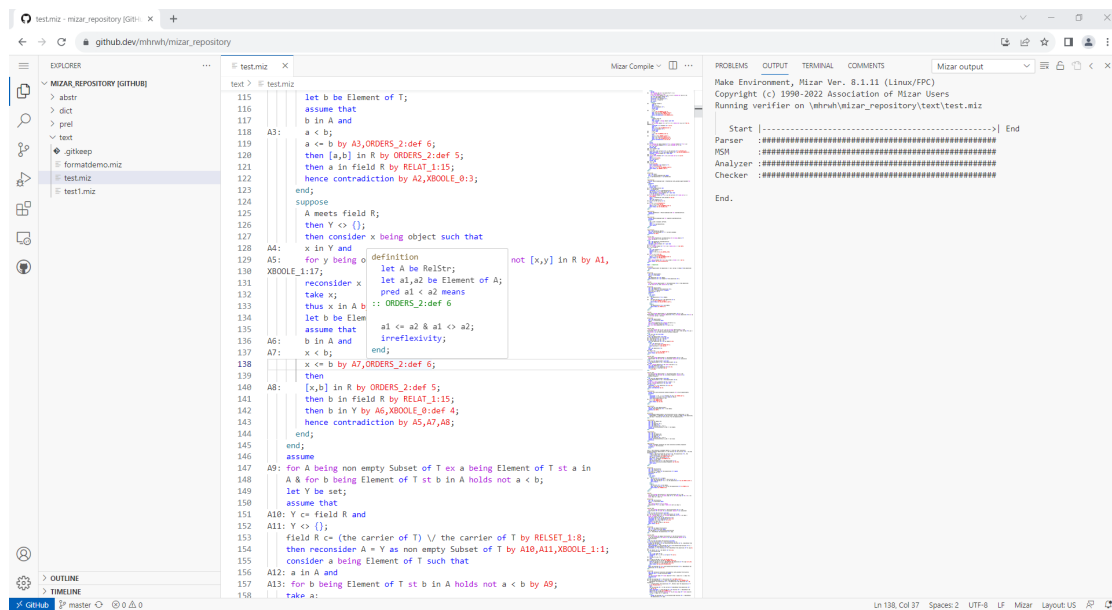**Figure 2:** Sequence diagram of the verifier command execution

Figure 3: Hover display and command execution functions

- comment-region: Comments out the selected range.
- mizar-indent-line: Indents the selected line appropriately.
- indent-region: Indents non-empty lines within a selected region.
- mizar-indent-width: Sets the indent width for Mizar articles.

In Lean4, there has been consideration to use a pretty printer for code formatting[7]. However, due to the complexity of the formatting rules and a limited number of combinators available for proper indentation formatting, it has yet to be put into practical use. In the case of Coq, code formatting can be achieved via the beautify command[8]. The command is "coqc -beautify [input file]", and the beautified code is output to a file named "[input file name].beautified".

3.2. Coding Conventions for the Mizar Language

To discern the coding conventions of the Mizar language, we derived implicit rules common to the source code in the Mizar Mathematical Library (MML), as well as those found in Python's PEP8[9], the Google C++ Style Guide[10], and Bob's Concise Coding Conventions [9], among others. The coding conventions we established are as follows.

The **indentation** specification varies between the environment and body sections. In the environment section, a set number of spaces (one space at the beginning of a sentence, six spaces otherwise) are inserted for each directive. Within the body section, the contents of a block are indented by one step. A block in Mizar language refers to a region that commences with a start keyword such as *definition* or *registration* and concludes with *end*. **Line breaks** are added at the end of each sentence, right before a label, immediately preceding the ".=" keyword, and before and after specific keywords such as the commencement of a block. The **length of a single line** should not surpass a specific number of characters. A solitary **blank line** is placed between top-level blocks, and two or more consecutive blank lines are removed. As a general rule, a single **space character** is added between tokens, but this is omitted before and after certain tokens like parentheses and commas. Consecutive spaces within a sentence and surplus white space characters at the start or end of a sentence are eliminated. **Label names** are reordered to ascend within each top-level block. This approach helps overcome the difficulties associated with finding labels if they are randomly defined. The label names are transformed to follow the format 'A' + 'sequential number', which means the label names within the block will be A1, A2, A3... in that order. However, this rule excludes theorem and definition labels.

3.3. Code Formatter for the Mizar language

Leveraging the coding conventions outlined above, we incorporated a formatter that can operate within the Mizar Extension for VSCode. Users have the ability to customize certain specifications of the coding conventions in settings.json. Configurable items include the number of indents, the maximum character count per line, and keywords for eliminating whitespaces before and

---

[7]https://github.com/leanprover/lean4/issues/369

[8]https://github.com/coq/coq/blob/f3ff34276a166830db47f13d93aaf8cdc2c8fa6e/doc/sphinx/practical-tools/coq-commands.rst

[9]https://pep8-ja.readthedocs.io/ja/latest/

[10]https://google.github.io/styleguide/cppguide.html

after characters. Examples of source code before and after the formatting process can be found in Appendix A.

3.4. Coding Assistant using ChatGPT

In the realm of software development, the use of generative AI tools for coding assistance, such as GitHub Copilot and ChatGPT Code Interpreter, has gained popularity. In the world of ITPs, research conducted by P. Nie et al. (2020) [10] aimed to suggest suitable code snippets based on context, through learning from the Coq library with language models. As part of the LeanDojo project led by K. Yang (2023) [11], a ChatGPT plugin has been developed. This plugin integrates the Lean library with machine learning models, attempting to enable automated theorem proving through generative AI. To examine the potential of employing ChatGPT for coding assistance in the Mizar language, we evaluated the coding assist capabilities of ChatGPT-3.5. Three key aspects were investigated: variable name suggestion, theorem name and comment generation, and proof completion. Examples of the interaction, including questions posed to ChatGPT and its responses for each aspect, can be found in Appendix B.

Variable Name Suggestion： Variable names in proofs should ideally adhere to mathematical conventions, such as utilizing 'n' for a natural number, 'p' for a prime number, and so forth. Also, it is common practice to name groups of variables having equivalent relations with a combination of consecutive alphabetic and numeric characters. However, in our study, when we presented Mizar language source code to ChatGPT and requested it to suggest suitable variable names, its suggestions tended to favor terms like 'point' or 'vertex' — terms more commonly associated with programming implementation. This approach does not align well with traditional mathematical conventions.

Theorem Name and Comment Generation： In many instances, when we provided theorems written in the Mizar language to ChatGPT and requested it to generate comments and theorem names, we obtained fitting responses. However, there were also a few instances where the responses were either circuitous or overly direct, as in "p1 is not equal to p2, and p3 is not equal to p2."

Proof Completion： In many cases, when we provided elementary propositions written in the Mizar language to ChatGPT and requested it to complete the proofs, the AI generated appropriate proof statements. However, a significant bottleneck was the GPT-3.5 version's inability to search the MML on the web, resulting in theorems not being properly cited.

4. Future Works

Concerning the Mizar Extension for VSCode for the Web, the current Mizar Server does not support simultaneous connections from multiple users, nor does it secure communication using SSL/TLS. Moving forward, our goal is to complete these remaining tasks and integrate with the emwiki system [12]. As of now, an agreement with the Mizar community regarding the development of a Mizar coding standard is yet to be reached. In the future, we plan to refine our formatter program, improve upon the coding conventions, and integrate coding assistance tools that utilize generative AI. The use of generative AI in coding assistance will demand revised

prompts, fine tuning, and the development of models trained on the MML. In addition, usability evaluation by experts in these functions is needed.

References

[1] H. Taniguchi, K. Nakasho, Visual Studio Code Extension and Auto-completion for Mizar Language, in: 2021 Ninth International Symposium on Computing and Networking (CANDAR), IEEE, 2021, pp. 182–188.

[2] E. J. G. Arias, B. Pin, P. Jouvelot, jsCoq: Towards Hybrid Theorem Proving Interfaces, arXiv preprint arXiv:1701.07125 (2017).

[3] M. Wenzel, Isabelle/PIDE After 10 Years of Development, in: 13th International Workshop on User Interfaces for Theorem Provers (UITP 2018), Oxford, UK, 2018.

[4] C. Lüth, M. Ring, A Web Interface for Isabelle: The Next Generation, in: International Conference on Intelligent Computer Mathematics, Springer, 2013, pp. 326–329.

[5] L. de Moura, S. Kong, J. Avigad, F. Van Doorn, J. von Raumer, The lean theorem prover (system description), in: Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25, Springer, 2015, pp. 378–388.

[6] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, Z. M. J. Jiang, GitHub Copilot AI pair programmer: Asset or Liability?, Journal of Systems and Software 203 (2023) 111734.

[7] N. Nguyen, S. Nadi, An Empirical Evaluation of GitHub Copilot's Code Suggestions, in: Proceedings of the 19th International Conference on Mining Software Repositories, 2022, pp. 1–5.

[8] J. Urban, MizarMode—An Integrated Proof Assistance Tool for the Mizar Way of Formalizing Mathematics, Journal of Applied Logic 4 (2006) 414–427.

[9] R. S. Laramee, Bob's Concise Coding Conventions (c^3), Advances in Computer Science and Engineering (ACSE) 4 (2010) 23–26.

[10] P. Nie, K. Palmskog, J. J. Li, M. Gligoric, Learning to Format Coq Code Using Language Models, arXiv preprint arXiv:2006.16743 (2020).

[11] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, A. Anandkumar, LeanDojo: Theorem Proving with Retrieval-Augmented Language Models, arXiv preprint arXiv:2306.15626 (2023).

[12] H. Furushima, D. Yamamichi, S. Shigenaka, K. Nakasho, K. Wasaki, An Integrated Web Platform for the Mizar Mathematical Library, in: International Conference on Intelligent Computer Mathematics, Springer, 2022, pp. 141–146.

A. Sample of Formatter Result

Listing 1 is the Mizar source code before formatting by the formatter, and Listing 2 is the source code after formatting.

Listing 1: Before Code Formatting

```
environ
  vocabularies SUBSET_1, XBOOLE_0, TARSKI, ZFMISC_1;
    notations TARSKI, XBOOLE_0, SUBSET_1, ZFMISC_1;
begin

theorem Th3  :
  for a ,b being   set holds a is Subset of b iff a c= b
  proof let a   ,b be set;
  hereby
  assume a is Subset of b  ;then a in bool b by SUBSET_1  :def 1 ;
  hence a c= b by  ZFMISC_1:def 1;
    end;assume a c= b; then a  in bool b by  ZFMISC_1  :  def 1;
  hence thesis by  SUBSET_1:def 1;end;

theorem
for a, b, c being set st a in b & b is Subset of c holds a is Element of c
proof let a, b, c be set;assume that A5: a in b and A3: b is Subset of c;
b c= c by A3, Th3; then a in c by A5,TARSKI:def 3;
hence thesis by SUBSET_1:def 1;end;
```

Listing 2: After Code Formatting

```
environ

 vocabularies SUBSET_1, XBOOLE_0, TARSKI, ZFMISC_1;
 notations TARSKI, XBOOLE_0, SUBSET_1, ZFMISC_1;

begin

theorem Th3:
  for a, b being set holds a is Subset of b iff a c= b
proof
  let a, b be set;
  hereby
    assume a is Subset of b;
    then a in bool b by SUBSET_1:def 1;
    hence a c= b by ZFMISC_1:def 1;
  end;
  assume a c= b;
  then a in bool b by ZFMISC_1:def 1;
  hence thesis by SUBSET_1:def 1;
end;

theorem
  for a, b, c being set st a in b & b is Subset of c holds a is Element of c
proof
  let a, b, c be set;
```

```
    assume that
    A1: a in b and
    A2: b is Subset of c;
    b c= c by A2, Th3;
    then a in c by A1, TARSKI:def 3;
    hence thesis by SUBSET_1:def 1;
end;
```

B. Interaction with ChatGPT for Coding Assistant

B.1. Variable Name Suggestion

USER: To enhance readability, please recommend better names for the variables. The variable names should be concise and convey a mathematical property or role. For example, "n" represents natural numbers, and "p1," "p2," etc. represent each vertex of a figure. """

CHATGPT: q is replaced by dimensions to indicate the number of dimensions in the real space. x, y, and z are replaced by vertex1, vertex2, and vertex3 respectively, indicating the vertices of the triangle. a, b, and c are replaced by coeff1, coeff2, and coeff3 respectively, to represent the coefficients of the vertices. p is replaced by point to represent a point in the real space. P is replaced by Predicate to indicate that it is a predicate in the proof.

Listing 3: Given Source Code

```
definition
  let q be Element of NAT,x,y,z be Point of TOP-REAL q;
  func closed_inside_of_triangle(x,y,z) -> Subset of TOP-REAL q equals
  {p where p is Point of TOP-REAL q:
    ex a,b,c being Real st 0<=a & 0<=b & 0<=c
  & a+b+c=1 & p=a*x+b*y+c*z};
  correctness
  proof
    defpred P[set] means
    ex a,b,c being Real st 0<=a & 0<=b & 0<=c & a+
    b+c=1 & $1=a*x+b*y+c*z;
    {p where p is Element of TOP-REAL q: P[p]} is Subset of TOP-REAL q
    from DOMAIN_1:sch 7;
    hence thesis;
  end;
end;
```

B.2. Theorem Name and Comment Generation

USER: Answer the following theorem in the following form. You do not need to include '' or '' in your answer. Explain in natural language rather than directly using the variable or function name in the equation. The summary should be brief and explanatory.

Answer Format """ Name: A suitable name for the theorem. Assumptions: Enumerate assumptions Summary: summary of what the theorem states """

Listing 4: Source Code

```
theorem
  for p1,p2,p3 st p2<>p1 & p1<>p3 & p3<>p2 & angle(p2,p1,p3)<PI holds
  angle(p2,p1,p3)+angle(p1,p3,p2)+angle(p3,p2,p1)=PI
proof
  let p1,p2,p3;
  assume that
A1: p2<>p1 & p1<>p3 and
A2: p3<>p2 and
A3: angle(p2,p1,p3)<PI;
A4: euc2cpx(p1)<> euc2cpx(p2) & euc2cpx(p1)<> euc2cpx(p3) by A1,Th4;
A5: euc2cpx(p3)<> euc2cpx(p2) by A2,Th4;
  per cases by COMPLEX2:70;
  suppose
A6: 0 = angle(euc2cpx(p2),euc2cpx(p1),euc2cpx(p3));
    now
      per cases by A4,A5,A6,COMPLEX2:87;
      suppose
        angle(euc2cpx(p1),euc2cpx(p3),euc2cpx(p2)) = 0 & angle(euc2cpx
        (p3),euc2cpx(p2),euc2cpx(p1)) = PI;
        hence thesis by A6;
      end;
      suppose
        angle(euc2cpx(p1),euc2cpx(p3),euc2cpx(p2)) = PI & angle(
        euc2cpx(p3),euc2cpx(p2),euc2cpx(p1)) = 0;
        hence thesis by A6;
      end;
    end;
    hence thesis;
  end;
  suppose
    0 < angle(euc2cpx(p2),euc2cpx(p1),euc2cpx(p3));
    hence thesis by A3,A4,COMPLEX2:84;
  end;
end;
```

B.3. Proof Completion

> **USER**: The following is a theorem described in Mizar. Please complete this proof. Please provide the completed source code and an explanation of how and where you completed it.

Listing 5: Source Code Before Completion

```
theorem Th2:
  r is Integer implies ex k st r = k or r = - k
proof
end;
```

Listing 6: Source Code After Completion

```
theorem Th2:
  r is Integer implies ex k st r = k or r = - k
proof
  assume
    r is Integer;
  per cases;
  suppose
    r >= 0;
    then consider k being Integer such that
    k = r;
    take k;
    thus thesis;
  end;
  suppose
    r < 0;
    then consider k being Integer such that
    k = -r;
    take k;
    thus thesis;
  end;
end;
```