Varphi: A Description Language for Turing Machines

Hassan El-Sheikha
University of Toronto
Toronto, Ontario, Canada

Mohammad A. Mahmoud
University of Toronto
Toronto, Ontario, Canada

Abstract

Conventional representations of Turing machines often overwhelm students of computability theory and obscure fundamental computational ideas. Varphi is a domain-specific language designed to streamline Turing machine specification with a minimalist syntax and integrated debugging tools. In preliminary classroom trials involving 184 participants tackling deliberately challenging tasks, 88% of students successfully completed them using Varphi, highlighting the tool's potential utility. This paper presents a technical overview of Varphi and analyzes its pedagogical benefits.

CCS Concepts

• **Theory of computation** → **Abstract machines**; **Turing machines**; • **Software and its engineering** → **Domain specific languages**; **Syntax**; **Semantics**; *Interpreters*.

Keywords

Turing Machines, Computer Science Education, Domain-Specific Languages

ACM Reference Format:
Hassan El-Sheikha and Mohammad A. Mahmoud. 2025. Varphi: A Description Language for Turing Machines. In *Proceedings of Western Canada Conference on Computing Education 2025 (WCCCE '25).* 2 pages. https://doi.org/10.60770/wzt9-s649

1 Introduction

Computability theory and Turing machines lie at the heart of computer science education. Yet, the conventional representations of Turing machines—diagrams and transition tables—often overwhelm students with unnecessarily complex and ambiguous forms. These varied representations tend to overshadow the elegance of Alan Tur-ing's original model, shifting student attention away from grasping core computational ideas.

Existing tools like JFLAP provide visual aids for designing Turing machines, but may lack integration with automated code grading systems since they are graphical tools rather than programming lan-guages [3]. Meanwhile, existing languages like Turing Machine Sim-ulation Language (TMSL) may be too abstract for Turing machine theory—featuring constructs like loops—and often lack debugging tools [2].

WCCCE '25, Calgary, AB

https://doi.org/10.60770/wzt9-s649

To address these challenges, we introduce Varphi—a domain-specific language that simplifies Turing machine representation. This paper details its design, implementation, and preliminary classroom trials in a third-year computability theory course, the results of which suggest enhanced student understanding and problem-solving skills. By focusing on core computational concepts and merging theory with practical coding, Varphi offers a promising, low-barrier approach to teaching Turing machines.

2 Technical Overview

2.1 Syntax and Semantics

Each Varphi program consists of transition rules that specify:

(1) the current state,
(2) the symbol at the current tape cell,
(3) the next state,
(4) the symbol to write at the current tape cell, and
(5) the head movement (one cell left or right).

This rule format corresponds with the standard unary Turing machine model with a single two-way infinite tape, where the blank symbol is denoted by 0. For example, a Varphi program describing a machine that adds one to a (unary) number is shown below:

```
q0 1 q0 1 R
q0 0 q1 1 L
```

Varphi adopts a "top-left" convention, meaning the initial state in the first transition rule is designated as the initial state of the described machine (in the example program above, the initial state would be q0).

2.2 Implementation and Usage

Turing machines described in Varphi are executed using the lightweight Varphi Interpreter (vpi), a dependency-free tool that simulates a universal Turing machine [4]. Users supply an input tape and receive an output tape after execution. For example, below is a shell session where the addition-by-one machine is simulated:

```
$ vpi -c add1.vp
Input Tape: 111
Output Tape: 1111
```

For debugging, invoking vpi with the –d flag initiates a step-by-step simulation. During debugging, the current tape cell is enclosed in square brackets ([]), and the cell that contained the leftmost tally on the input tape is marked with curly braces ({}):

```
$ vpi -d add1.vp
Input Tape: 111
State:  q0
Tape:  [{1}]11
Press ENTER to step...
```

Additionally, the Varphi language extension for Microsoft Visual Studio Code (VS Code) enhances the user experience by providing

syntax highlighting, integrated simulation, and a graphical debugger that highlights the next line of execution. This extension is especially useful for users less familiar with command-line interfaces.

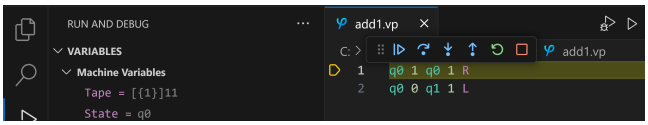

Figure 1: Varphi Visual Studio Code Debug Session.

3 Evaluation and Evidence

Preliminary feedback from a cohort of mathematics and computer science students enrolled in a third-year undergraduate computability course is encouraging. In an evaluation, participants were given two hours to review Varphi's documentation and experiment freely with it, followed by interaction with a "mystery" Varphi program (35 lines long) deliberately designed to cause challenges for manual analysis (i.e., extremely tedious to interpret by hand). Subsequently, they were instructed to report, using the Varphi Interpreter, the machine's output for a lengthy input tape (containing 28 tallies) and to identify the function computed by the machine. Finally, they were asked to provide feedback on Varphi and describe how it could help them and future students.

3.1 Quantitative Analysis

Based on data aggregated from 184 students, the average rating of Varphi was 4.28 out of 5 with a standard deviation of 0.97. The documentation received an average rating of 4.23 with a standard deviation of 1.08. Additionally, 88% of student submissions were correct.

Given the task's deliberate difficulty for manual analysis, a correctness rate of 88% is a noteworthy outcome. While it is reasonable to consider why the remaining 12% did not arrive at the correct answer, the high success rate and positive ratings suggest that Varphi was generally perceived as intuitive and self-explanatory.

3.2 Qualitative Analysis

The qualitative feedback underscores several key themes. Many students emphasized the immediate benefits of Varphi's features: multiple students noted that the debugger was extremely useful, while others highlighted the ease of integration and mentioned that they would love to see the tool become a part of the course. Such comments reveal that Varphi is not only functionally effective but also has the potential to be embedded within the curriculum to enhance practical learning. On the more critical side, many students also suggested creating a web-based Varphi environment for running and debugging programs through a web browser, as discussed in Section 5 (Future Work).

4 Pedagogical Benefits

Varphi's uniform syntax and integrated debugging tools directly address key challenges in teaching Turing machines. Preliminary classroom trials indicate several pedagogical benefits:

- **Reduced Cognitive Load:** By replacing ambiguous natural language descriptions with structured rules, Varphi allows students to concentrate on algorithmic logic.
- **Enhanced Engagement:** The interactive debugging tools, especially within familiar integrated development environments like VS Code, transform a traditionally abstract topic into a hands-on learning experience. Student feedback reinforces that Varphi not only demystifies Turing machine operations but also promotes active experimentation and deeper understanding.
- **Error Localization:** Varphi's compiler flags syntax errors (such as invalid state names or formatting issues) during compilation. This immediate feedback helps students correct mistakes early, unlike handwritten diagrams where errors may go unnoticed until manual grading.
- **Scalability:** The minimalist design makes it feasible for students to build more complex machines, even for projects simulating interactive games. By supporting descriptive state names (e.g., qRock or qTie for a game like "Rock, Paper, Scissors"), Varphi encourages a modular approach that mirrors high-level programming practices.

5 Future Work

Building on these promising results, future efforts will focus on:

- **Improved Usability:** Exploring the development of a web-based interface to further lower barriers to entry.
- **Integration with Autograders**: Integrating Varphi with automated grading systems like MarkUs in large-scale educational settings, which have been shown to decrease assessment times and promote better adherence to deadlines [1].
- **Quantitative Studies**: Conducting controlled, pre-/post-assessments and detailed error analyses to rigorously evaluate learning gains and debugging efficiency compared to existing tools.
- **Extending Language Features:** Expanding the language to support richer Turing machine variants (e.g., those with multiple tapes and arbitrary alphabets) to cater to advanced theoretical coursework or research, while maintaining Varphi's clarity.

References

[1] Morgan Magnin, Guillaume Moreau, Nelle Varoquaux, Benjamin Vialle, Karen Reid, Mike Conley, and Severin Gehwolf. 2012. MarkUs: An Open-Source Web Application to Annotate Student Papers On-Line. *Proceedings of the ASME 11th Biennial Conference On Engineering Systems Design And Analysis (ESDA 2012)* (07 2012). doi:10.1115/ESDA2012-82141

[2] Isaac McGarvey, Joshua Gordon, Keerti Joshi, and Snehit Prabhu. 2008. *TMSL (Turing Machine Simulation Language): Language Manual and Project Report.* Technical Report. https://www.cs.columbia.edu/~sedwards/classes/2008/w4115-fall/reports/TMSL.pdf

[3] Susan H Rodger and Thomas W Finley. 2006. *JFLAP: an interactive formal languages and automata package.* Jones & Bartlett Learning.

[4] Alan Turing. 1936. On computable numbers, with an application to the Entscheidungs problem. *Proceedings of the London Mathematical Society Series/2 (42)* (1936), 230–42.