Automated Reasoning in Python

Philip Zucker

2026-01-11

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

2 Terms

When we do mathematics by hand on paper and pencil or on a chalkboard, it is not entirely clear how to model what is being written at it's most literal level.

The human mind can quickly determine the intent behind a scrawl of $sin^2(x) + cos^2(x)$ or that a *drawing* of a circle represents the mathematical abstraction the mathemtically perfect circle, despite the imperfections of rendering, the finite width of it;s edge on the board.

In order to be precise enough about our topic to make it mechanical, we must try to invent of model of mathematical symbolism that is possible to translate to a machine.

Ultimately, the choice is somewhat arbitrary. We could perhaps store our mathematical expressions as PNG, editting them with photoshop. This choice would require excessive memory and computation and not be particularly easy to work with (except perhaps by neural net, so maybe there is something there).

Another choice might be strings or byte sequences. This is a lowest common denomiator of data in machines and communication. These too have their issues.

It is very typical that strings must be parsed into a more structural form.

Terms are trees

`NamedTuple` is a python standard library function to make record/struct datatypes.

```python
from typing import NamedTuple


class App(NamedTuple):
    f: str
    args: tuple["App", ...]

    def __repr__(self):
        if len(self.args) == 0:
            return self.f
        else:
            return f"{self.f}({",".join(map(repr, self.args))})"
```

It is convenient to make helper functions to make construction look more natural.

Constants are merely applications to zero arguments.

```python
def add(x, y):
    return App("add", (x, y))


x = App("x", ())
y = App("y", ())
z = App("z", ())

add(add(x, y), z)
```

```
add(add(x,y),z)
```

2.1 Subterm

```python
def is_subterm(t: App, s: App) -> bool:
    if t == s:
        return True
    else:
        return any(is_subterm(arg, s) for arg in t.args)


assert is_subterm(add(x, y), x)
assert not is_subterm(add(x, y), z)
assert not is_subterm(x, add(x, y))
```

3 Interpreting

```python
def interp(t: App, env: dict[str, int]) -> int:
    match t:
        case App("add", (x, y)):
            return interp(x, env) + interp(y, env)
        case App(name, ()):
            return env[name]
        case _:
            raise Exception("Unexpected case")


env = {"x": 3, "y": 14}
assert interp(x, env) == 3
assert interp(add(x, y), env) == 17
```

Finite intepretations

4 Contexts and Zippers

5 Hash Consing / Interning

6 Terms with Variables

A standard starting point

```
class Var(NamedTuple):
    name: str


type VTerm = App | Var
```

7 Patterns

8 Rewriting

9 Term Ordering

10 Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.