Why3 and Proving A* Automatically

A Case Study of Why3 as a Tool for Automated Software Verification

1. The Goal

To explore the capabilities and limitations of Why3 as a tool for automated software verification, through a case study of verifying the correctness of the A* algorithm.

2. Introduction to Formal Verification

"Program testing can be used to show the presence of bugs, but never to show their absence!"

~ Edsger W. Dijkstra

Formal Verification - a method of software verification whereby the correctness of a program is proven through formal mathematical reasoning.

- Stronger guarantee of correctness than software testing
- Time-consuming
- Difficult

There are different types of computer-aided verification software that can be used to automate this process:

- **Interactive Theorem Provers**: Rocq, Isabelle
- **Automated Theorem Provers (ATPs)**: Vampire, E Prover, SPASS
- **SMT Solvers**: Alt-Ergo, CVC5, Z3

3. What is Why3?

Why3 is a platform for automated deductive program verification.

It allows users to implement programs, and then offloads the proofs to external theorem provers (e.g. Z3, E Prover, etc.).

It provides various tools to aid in the verification process:

- a language (**WhyML**) for expressing logic and programs
- many **logic transformations** to guide the proving process
- **proof sessions** to save and replicate the proofs
- an **IDE** to view and edit code/ proofs

WhyML Verification Constructs:

Keyword(s)	Description
`assert`	Asserts that a statement holds on this line.
`requires`	Introduces a function pre-condition.
`ensures`	Introduces a function post-condition.
`diverges`	Marks a function as nonterminating.
`writes`	Lists variables mutated by this function.
`invariant`	Introduces a loop invariant.
`variant`	Introduces a decreasing variant which proves loop termination.
`old`	Used in condition to reference the initial state of a mutable variable.

WhyML Logical Constructs:

Keyword(s)	Description
`/\`, `\/`, `not`	Conjunction, disjunction, negation.
`->`, `<->`	Implication and bi-implication.
`forall`, `exists`	Universal and existential quantifiers.
`function`	Used to define logical functions.
`predicate`	Used to define simple predicates.
`inductive`	Used to define inductive predicates.
`constant`	Introduces some arbitrary constant.
`axiom`	Introduces a logical axiom.

WhyML Program Constructs:

Keyword(s)	Description
`&&`, `||`, `not`	AND, OR, and NOT operators.
`for..to..do..done`	For loop.
`while..do..done`	While loop.
`if..then..else..end`	If statement.
`begin..end`	Code block.
`let..in..`	Let binding.
`match..with..end`	Used for pattern matching.
`ghost`	Marks code as "ghost code", which is only added for verification purposes.
`type`	Introduces a new data type.
`abstract`	Makes all fields in a record accessible only in ghost code.
`mutable`	Makes a record field mutable.

4. Why A*?

A* is a **heuristic-based algorithm** which finds the shortest distance through a graph from a given source to a given destination. It can be thought of as an **extension of Dijkstra's algorithm**, by introducing a heuristic to estimate the distance from a vertex to the destination

There are a couple of reasons to chose A* for our case study:

- Its complexity could showcase more of Why3's features and lead to more interesting observations.
- Dijkstra's algorithm has been previously verified in Why3 by J.C.Filliâtre, which was a useful resource on how to approach the proof.

5. Implementation Approach

Procedure:

1. Define the algorithm with pseudocode and introduced a few properties we know must hold for a correct implementation.
2. Implement this in WhyML and use Why3's toolset to prove these properties.

The Properties We Proved:

- **Optimal Substructre Property of Shortest Paths**
- **Consistency Implies Admissibility**
- **Termination and Completeness Properties**
- **Optimal Efficiency Property**
- **Minimal Expansion Property**
- **Open Optimum Property**

6. Results

Run-time Figures for Each External Prover Used in Our Proof:

Prover	# of Goals	Min	Max	Mean
Alt-Ergo 2.6.2	21	0.01 s	1.73 s	0.10 s
CVC5 1.2.1	53	0.01 s	2.66 s	0.34 s
Eprover 2.0	2	0.07 s	0.37 s	0.22 s
Z3 4.15.0	52	0.00 s	0.69 s	0.03 s
Total	128	0.00 s	2.66 s	0.17 s

Number of Logical Transformations Used in The Proof:

Transformation	# of Uses
`assert`	18
`unfold`	9
`destruct_rec`	9
`split_vc`	8
`inst_rem`	6
`exists`	6
`case`	6
`induction_pr`	4
`introduce_premises`	3
`induction`	1
Total	70

7. Observations

Usability:

- High expressiveness of WhyML
- Simple installation process
- Why3 IDE lacks the "undo" feature
- Limited/out-dated documentation

Automation:

- Fast, consistent, and reproducible proof generation
- ATPs are useful despite out-dated versions supported by Why3
- Manual proving is still a large component

Program Verification:

- Possibility of principle of explosion in Why3
- Pre/post-loop states are linked through invariants

8. Conclusions and Future Work

We found that the main **advantages** of Why3:

- highly expressive language
- the speed, consistency, & reproducibility of its proofs
- its simple installation process.

However, there were a few **limitations** we found

- there's missing documentation for in the Why3 Manual for a lot of features
- proofs still involve a large amount of manual proving
- there is the danger of a proof by the principle of explosion.

We outline the following **future work** in this field:

- explore Why3's ability to run code written in WhyML by applying it to our implementation.
- attempt to prove different variations/optimisations of A*.
- testing Why3 on a wider variety of problems and comparing it against other software tools.

Kajetan Neumann TU Delft, CSE3000 Research Project https://repository.tudelft.nl/