Formally Modeling and Analyzing Mathematical Algorithms with Software Specification Languages & Tools

Status Report on Master Thesis

Daniela Ritirc

07.12.2015

Table of Contents

Aim of the Thesis

Investigate the behaviour of software specification languages and tools on mathematical algorithms:

- show how mathematical algorithms can be modeled with software specification languages
- investigating how far simulating, visualizing, model checking and verifying is possible

Formal Modeling I

- **Modeling a System**
  - transfer the system into an abstract model
  - translation into some software specification language
- **Simulation**
  - execution of the formal model which imitates the execution of the real system
- **Visualization**
  - “pretty-printed“ (graphically illustrated) run of the model

Formal Modeling II

- **Specification**
  - formally state the properties the program shall have
  - expressed in the software specification language
- **Model Checking**
  - investigation whether the system model fulfills the specified property by elaborating all possible executions
- **Verification**
  - investigation whether the system model fulfills the specified property by mathematical proofs

DPLL algorithm

- solving propositional satisfiability problem
- deciding if a formula in conjunctive normal form is satisfiable
- backtracking based search algorithm

Require: *(F, n)*

Input condition: $n \geq 1 \land F \in \text{Formula}_n$

Ensure: *s*

Output condition: $s = 1 \Leftrightarrow (F, n)$ is satisfiable

Input and output conditions

Input condition:

$\text{Literal}_n := \{l \in \mathbb{Z} \mid 0 < l \leq n \lor -n \leq l < 0\}$

$\text{Clause}_n := \{c \in \mathbb{P}(Literal_n) \mid \forall l \in \mathbb{Z} : \neg(l \in c \land -l \in c)\}$

$\text{Formula}_n := \mathbb{P}(Clause_n)$

Output condition:

$\text{Valuation}_n := \text{Clause}_n$

$$\text{(F, n) is satisfiable} :\Leftrightarrow \exists v \in Valuation_n : \underbrace{\forall c \in F : \underbrace{\exists l \in c : l \in v}_{\text{ValSatClause(c,v)}}}_{\text{ValSatFormula(F,v)}}$$

Pseudo-code

Algorithm DPLL(Φ) recursive

```
Require: A formula Φ
Ensure: A truth value
if Φ is empty then
    return true
else if Φ contains empty clause then
    return false
end if
select a variable v occurring in Φ
if DPLL(substitute(Φ, v, true))=true
then
    return true
else
    return DPLL(substitute(Φ, v, false))
end if
```

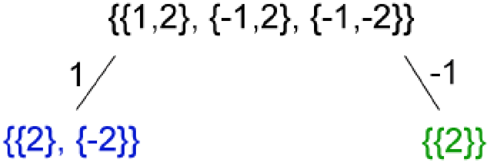
Algorithm DPLL(Φ) iterative

```
Require: A formula Φ
Ensure: A truth value
stack ∈ empty
while true do
    if Φ is empty then
        return true
    else if Φ contains an empty clause
    then
        if stack.isEmpty() then
            return false
        end if
        Φ ← stack.pop()
    else
        select a variable v occurring in Φ
        stack.push(substitute(Φ, v, false))
        Φ ← substitute(Φ, v, true)
    end if
end while
```

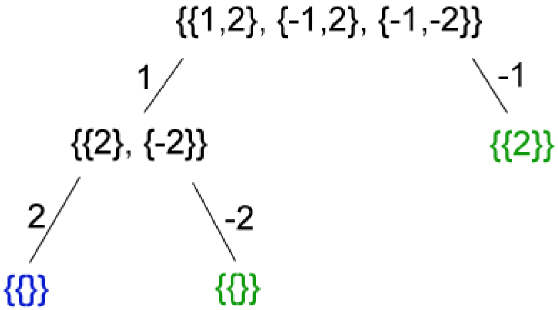
Example

{{1,2}, {-1,2}, {-1,-2}}

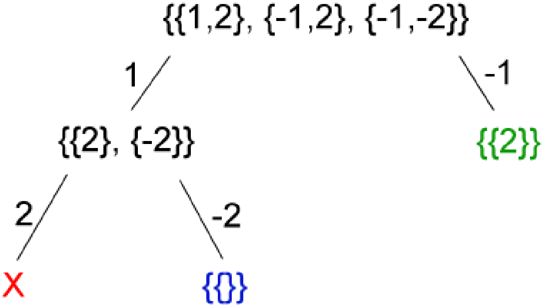
Example



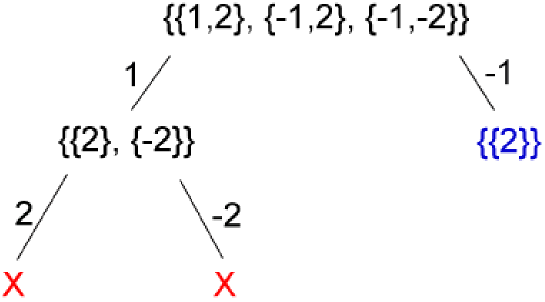
Example



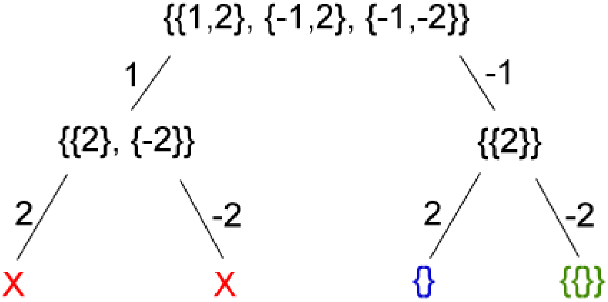
Example



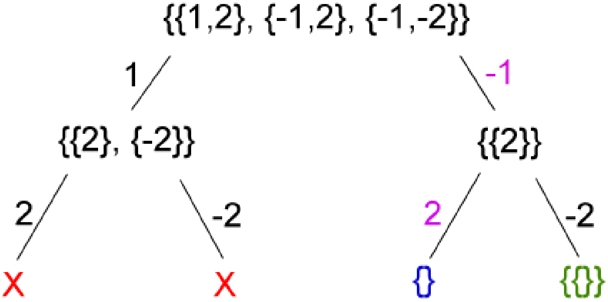
Example



Example



Example



TLA/PlusCal

- combines temporal logic with a logic of actions
- everything is described as a logical formula

- PlusCal is an algorithmic language
- a PlusCal algorithm is translated to a TLA specification by the PlusCal translator

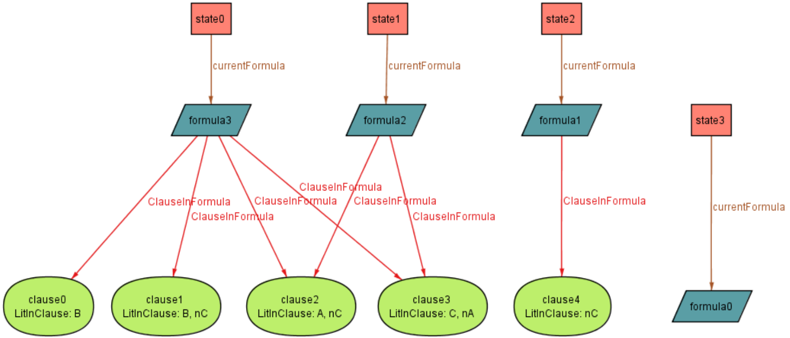
- TLC model checker generates a finite set of initial states and performs a breadth-first search

Alloy

- specification language for expressing structural constraints and behaviour of a system
- based on relational logic
- used for finite models

- generates instances of models
- simulates the execution of operations
- check user-specified properties of a model

Visualization of an instance



Event - B

- describe a system with events
- develop a series of more and more accurate models of the system
- automatically generates proof obligations for each level of abstraction
- use of automatic provers
- use of interactive provers

Conclusion & Current work I

TLA

- easy implementation in PlusCal
- language is based on mathematics
- model checking is comprehensive and traceable
- scope for model checking is defined by the values of the constants
- no verification

Alloy

- complicated implementation
- gain visualizations of the algorithm
- model checking is not traceable
- scope for model checking needs to be defined for each object
- no verification

Conclusion & Current work II

Event-B

- specification with events and invariants
- no model checking
- automatic verification only possible for simple data types
- interactive prover is not well documented
- verification calculus seems not complete
- idea of refinement is not really applicable

Current work

- analysis of Dijkstra's Shortest Path Algorithm