Practical Formal Methods

Formal Methods in Action

Ilya Sergey

November 2024

ilyasergey.net/PFM24

Today's Agenda

A hands-on overview of the tools and techniques

- Using simple examples from other classes
- Not aiming to showcase all features
- Skipping almost all the theory

The goal:
quick introduction to help you choose a project

The Three Case Studies

Concurrent Readers-Writers Problem in TLA+

- A “Hello World” of concurrent interaction protocols
- Interesting safety and liveness properties
- Focus on a high-level model rather than implementation

The Three Case Studies

Concurrent Readers-Writers Problem in TLA+

- A “Hello World” of concurrent interaction protocols
- Interesting safety and liveness properties
- Focus on a high-level model rather than implementation

Using SMT solvers for verification and synthesis

- What are SAT and SMT solvers and how are they useful
- Search problems as solutions to constraint systems

The Three Case Studies

Concurrent Readers-Writers Problem in TLA+

- A “Hello World” of concurrent interaction protocols
- Interesting safety and liveness properties
- Focus on a high-level model rather than implementation

Using SMT solvers for verification and synthesis

- What are SAT and SMT solvers and how are they useful
- Search problems as solutions to constraint systems

Deductive Verification in Dafny

- Specifying programs in Hoare logics
- Proving that programs do what they should (soundly)

Part I

Specifying Complex Systems in TLA+

Concurrent Reading and Writing

Safe concurrent programs:

Multiple concurrent reads of same memory: Not a problem
Multiple concurrent writes of same memory: Problem
Multiple concurrent read & write of same memory: Problem

So far:

If concurrent write/write or read/write might occur,
one can use synchronisation to ensure one-thread-at-a-time

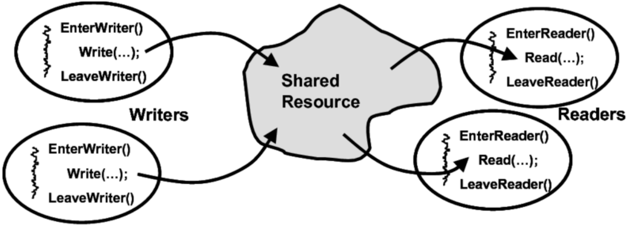
But this is unnecessarily conservative:

Could still allow multiple simultaneous readers!

Readers and Writers Problem

A variant of the mutual exclusion problem
where there are two classes of processes:

- writers which need exclusive access to resources
- readers which need not exclude each other



Concurrent Correctness

There are two types of correctness properties:

Safety properties

The property must always be true.

Liveness properties

The property must eventually become true.

Exercise: Designing the Protocol for Concurrent Reading and Writing

- What are the components of the system?
- What are its safety properties?
- What about liveness?

Live Demo: Basics of TLA+

- State and variables
- Actions as relations
- Specifying safety and liveness properties
- Detecting and analyzing the violations (bugs in the design!)

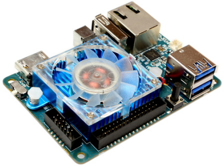
https://github.com/formal-and-practical/basic-examples/

folder “tlaplus”

Part II

SAT and SMT for Verification and Synthesis

The SAT/SMT Revolution



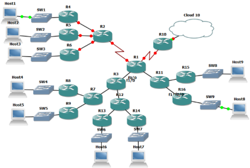
hardware verification

KLEE Dafny LiquidHaskell

software verification

Rosette Sketch Leon Synquid

software synthesis & repair



network configuration synthesis

biological modeling

architecture

Boolean SATisfiability

$$(\text{gin} \lor \text{tonic}) \land (\text{minor} \Rightarrow \neg\text{gin}) \land \text{minor}$$

Boolean SATisfiability

$$(\text{gin} \lor \text{tonic}) \land (\text{minor} \Rightarrow \neg\text{gin}) \land \text{minor}$$

Solution:

$\text{minor} \mapsto T$

$\text{gin} \mapsto F$

$\text{tonic} \mapsto T$

Satisfiability Modulo Theories

$$(\text{gin} \lor \text{tonic}) \land (\text{age} < 21 \Rightarrow \text{abv} = 0) \land (\text{age} = 20)$$

Satisfiability Modulo Theories

$$(\text{gin} \lor \text{tonic}) \land (\text{age} < 21 \Rightarrow \text{abv} = 0) \land (\text{age} = 20)$$

> In the United States, "gin" is defined as an alcoholic beverage of no less than 40% ABV…
>
> Wikipedia

Satisfiability Modulo Theories

$$(\text{gin} \lor \text{tonic}) \land (\text{age} < 21 \Rightarrow \text{abv} = 0) \land (\text{age} = 20) \land (\text{gin} \Rightarrow \text{abv} \geq 40)$$

> In the United States, "gin" is defined as an alcoholic beverage of no less than 40% ABV…
>
> Wikipedia

Satisfiability Modulo Theories

$$(\text{gin} \lor \text{tonic}) \land (\text{age} < 21 \Rightarrow \text{abv} = 0) \land (\text{age} = 20) \land (\text{gin} \Rightarrow \text{abv} \geq 40)$$

$\text{age} \mapsto 20$

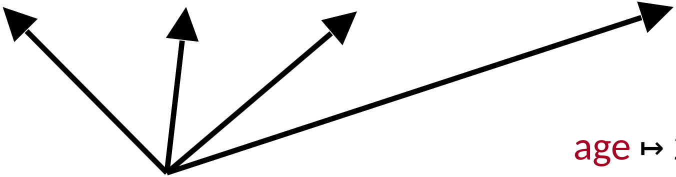
$\text{abv} \mapsto 0$

$\text{gin} \mapsto F$

$\text{tonic} \mapsto T$

Satisfiability Modulo Theories

$$(gin \lor tonic) \land (age < 21 \Rightarrow abv = 0) \land (age = 20) \land (gin \Rightarrow abv \geq 40)$$



theory of Linear Integer Arithmetic

$age \mapsto 20$

$abv \mapsto 0$

$gin \mapsto F$

$tonic \mapsto T$

Popular Solvers

Microsoft

Z3

Stanford

CVC5

SRI

Yices2

JKU Linz, Austria

Boolector

SMT competition: http://smtcomp.sourceforge.net

```
.smt2            // SMTLib format

(declare-fun (Int) age)
(declare-fun (Int) abv)
```

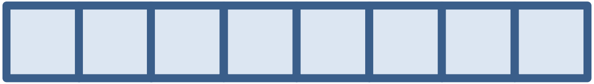
Plan for Today

How to use Z3 for:

1. Constraint programming
2. Program verification
3. Program synthesis

Problem: Array Partitioning

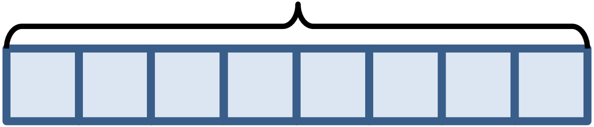
Partition an array of size *N* evenly into *P* sub-ranges



Problem: Array Partitioning

Partition an array of size *N* evenly into *P* sub-ranges

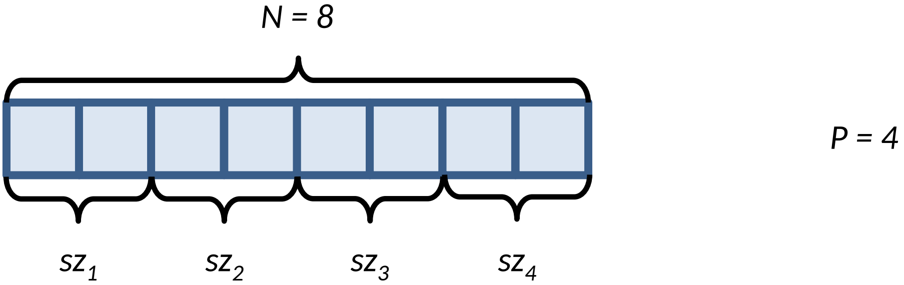
N = 8



P = 4

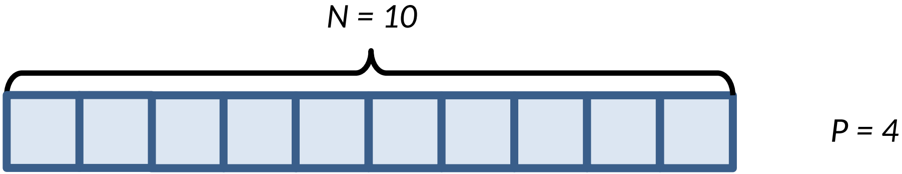
Problem: Array Partitioning

Partition an array of size *N* evenly into *P* sub-ranges



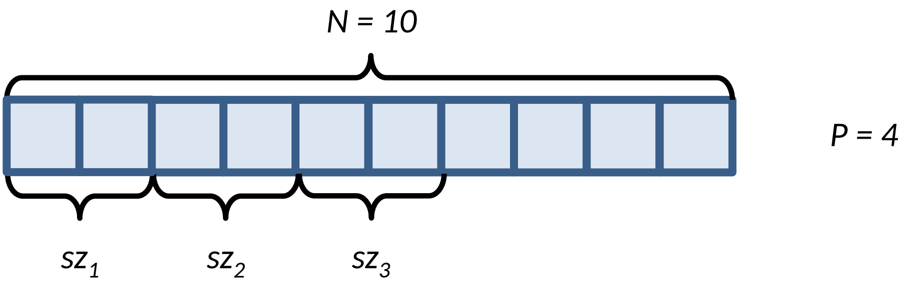
Problem: Array Partitioning

Partition an array of size *N* evenly into *P* sub-ranges



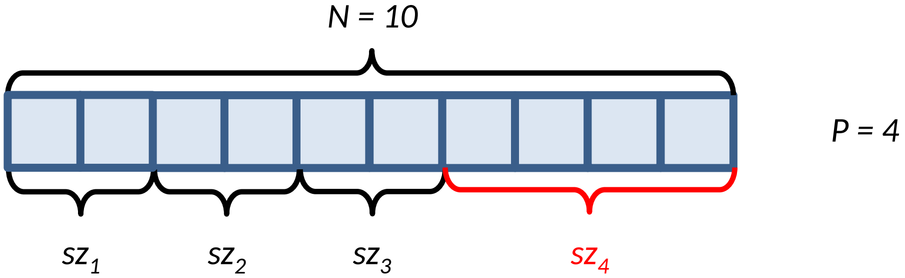
Problem: Array Partitioning

Partition an array of size *N* evenly into *P* sub-ranges



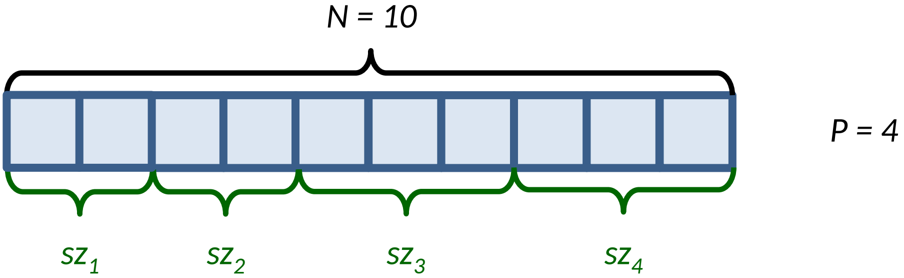
Problem: Array Partitioning

Partition an array of size N evenly into P sub-ranges



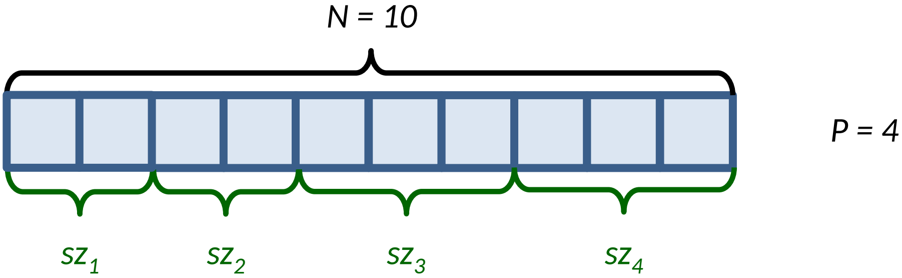
Problem: Array Partitioning

Partition an array of size N evenly into P sub-ranges



Problem: Array Partitioning

Partition an array of size *N* evenly into *P* sub-ranges



Can we always make them differ by at most 1?

Live Demo

Z3

to the rescue!

Plan for Today

How to use Z3 for:

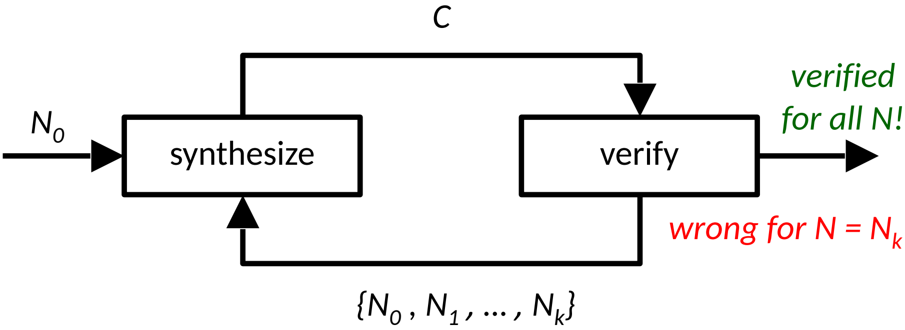
1. Constraint programming
2. Program verification
3. Program synthesis

Plan for Today

How to use Z3 for:

1. Constraint programming
2. Program verification
3. Program synthesis

CEGIS



C
N_0
synthesize
verify
verified
for all N!
wrong for $N = N_k$
$\{N_0, N_1, \ldots, N_k\}$

What we have seen:

How to use Z3 for:

1. Constraint programming
2. Program verification
3. Program synthesis

https://github.com/formal-and-practical/basic-examples/

folder “smt”

Part III

Deductive Hoare-style Program Verification in Dafny

Program specification

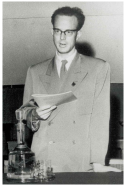
$$\{ P \} \ \mathbf{c} \ \{ Q \}$$

precondition postcondition

Meaning:
If the ***initial*** state satisfies P,
then the program **c** is safe to run and its ***final*** state satisfies Q.

Example: $\{ \text{True} \} \quad \mathbf{x := 3} \quad \{ x = 3 \}$

Symbolic execution

A method for establishing partial correctness

Independently discovered by ***Robert W. Floyd*** in 1967 and ***Tony Hoare*** in 1969

also hinted by Turing in 1949;

Also known as ***Hoare-style program logic***, ***Axiomatic program semantics***;

Symbolic execution allows us to abstract over specific ***values***

e.g., instead of x being $1, 2, 3, \ldots$, we can consider input $x \in \mathbb{N} \wedge x > 0$, reasoning out of these assertions about x;

Specifies ***what*** a program is doing without saying ***how*** it is doing that;

specifications $\{P\}$ **c** $\{Q\}$ are sometimes called ***Hoare triples***.

Program verification via symbolic execution

Verification is the process of ensuring that the program satisfies the ***specification*** (i.e., pre/postconditions), ascribed to it;

For the purpose of verification, the program is decomposed into ***primitive*** and ***composite*** statements:

- Primitive statements are variable ***assignments*** and calls to external functions;
- Composite statements are ***conditionals*** (**if-then-else**), **while**-loops and sequential compositions.

Preconditions are assumed/inferred, ***postconditions*** are obtained/checked via ***inference rules*** of symbolic execution.

Live Demo

Verifying a program in

https://github.com/formal-and-practical/basic-examples/

folder “dafny”

Summary of This Lecture

- We have seen three families of tools in action
  - TLA+ for specification and model checking
  - Z3 for constraint solving
  - Dafny for sound logic-based verification
- In the rest of the module, we will learn to use the tools for various applications
- We will also learn about how they work internally