Formal specification

Testing, debugging & Verification

Atze van der Ploeg

Today

- Formal specification: what and why
- A first glancy at Dafny
- Intro/Refresher on logic

Recall: Contract, bug

Bug = failure to meet specification

Specification = Contract:

Requires*: What the **client** must ensure*

Ensures*: What the **supplier** must ensure*

Bug = Breach of contract

Example, last week:

Java object specification:

```
public int hashCode()
```

- ...
- If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
- ...

Last week there was a breach of contract!

Bart and graduated bart where equal, but different hash code

How could we have detected this bug?

Aside: Check contract at runtime! (assertions)

```
class CheckContractStudent extends Student {
    public int equals(Object other) {
        boolean res = super.equals(other);
        assert !res || hashCode() == other.hashCode(); }
        return res;
    }
}
```

Assertions can be turned on an off (`java -ea` enables them)

Good idea, but problems:

- equals might not be called
- Runtime overhead
- Runtime check does not give certainty!

How could we have detected this bug?

Unit test: Must have specific test case for this to detect it

Property based testing:

- Generate random students
- if they are equal check if hashCode also equal

Assertions: Does not give safety, only detects problems

Conclusion: Hard to detect this bug (exept if you know what you are looking for)

Formal specification

Solution:

- Write specification in formal language
- (Automatically) prove that there can **never** be breach reach of contract
- Reject program otherwise

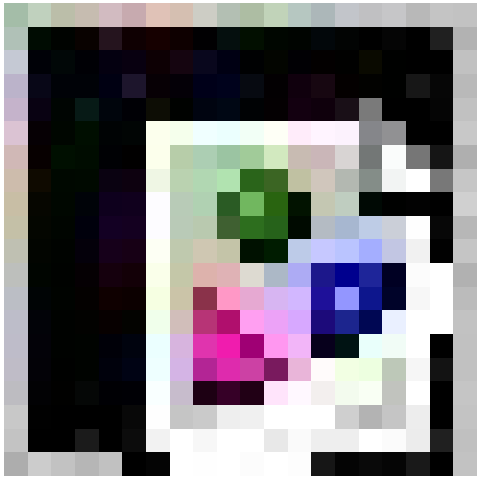
Formal verification programs

Program	Usage	Used in Industry?	Notes
Hol/Isabelle	Math & programming	Sometimes (link1)	
Ada/SPARK	Programming	Yes (link1, link2)	Subset of Ada
Dafny	Programming	No?	Java-like programming, this course
Coq	Math & Programming	Sometimes (link)	Dependent type theory
Agda	Math & Programming	Sometimes (link)	Dependent type theory, developed at Chalmers!

All are much more *researchy* than mainstream programming

Dependent type theory gives that
Spec lang = implementation language

Why are we using Dafny?



It is a research project that no-one in industry uses?

- Very similar to SPARK/Ada, which is used in Industry
- Easier to learn because it is Java like
- Knowledge about formal specification/verification is useful, even if you will not regularly use Isabelle/Dafny/Coq/etc. : it enables precise thinking

Formal specification - example

Informal Specification:

```
public int hashCode()
```

Requires: Nothing
Ensures: If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.

Formal Specification

Requires: Nothing
Ensures: $\forall x\ y$: Student, x.equals(y) $\Rightarrow$ x.hashCode() = y.hashCode()

Demo!

Dafny says no

```
datatype Student = Student(firstName : string, lastName : string, number : int, graduated : bool)

function hashCode(a : Student) : int
{
  a.number % 5 + if a.graduated then 15 else 31
}

function equals(a : Student, b : Student) : bool
{
  a.number == b.number && a.firstName == b.firstName
}

method Main() {
  assert forall x, y : Student :: equals(x,y) ==> hashCode(x) == hashCode(y);
}
```

Dafny says yes

```
datatype Student = Student(firstName : string, lastName : string, number : int, graduated : bool)

function hashCode(a : Student) : int
{
  a.number % 5
}

function equals(a : Student, b : Student) : bool
{
  a.number == b.number && a.firstName == b.firstName
}

method Main() {
  assert forall x, y : Student :: equals(x,y) ==> hashCode(x) == hashCode(y);
}
```

Dafny

Java-like language

(Automatically)
Proves that formal specification will **never** be violated

Also proves absence of runtime errors (implicit in formal specification):

- Non-Termination
- Array index out of bound
- Dereference null

Another motivating example: Zune leap year bug

- Zune was a portable media player released by Microsoft (ipod competitor)
- At approximately midnight Pacific Standard Time, on December 31, 2008, all Zune 30s froze
- Problem: compute year from number of days since 1980 looped
- The official fix was to drain the device battery and then recharge after midday GMT on 1 January 2009

```
year = ORIGINYEAR; /* = 1980 */

while (days > 365)
{
    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    }
    else {
        days -= 365;
        year += 1;
    }
}
```

Our running example: ATM.dfy

```
class ATM {
   // fields:
   var insertedCard           : BankCard;
   var wrongPINCounter        : int;
   var customerAuthenticated  : bool;

   // methods:
   method insertCard (card : BankCard) { ... }
   method enterPIN (pin : int)         { ... } ...
}
```

Informal spec

Informal specification of `enterpin(int pin)`:

> “Enter the PIN that belongs to the currently inserted bank card into the ATM.
>
> If a wrong PIN is entered three times in a row, the card is invalidated and confiscated.
>
> After having entered the correct PIN, the customer is regarded as authenticated.“

Making it a bit more formal

```
enterpin(int pin)
```

Requires : Card is inserted, user not yet authenticated

Ensures:

- If pin is correct then the user is authenticated
- If pin is incorrect and wrongPINCounter < 2 then wrongPINCounter is increased by 1 and user is not authenticated
- If pin is incorrect and wrongPINCounter >= 2 then card is confiscated and user is not authenticated

Let's try

```
class ATM {
   // fields:
   var insertedCard           : BankCard;
   var wrongPINCounter        : int;
   var customerAuthenticated  : bool;

   // methods:
   method insertCard (card : BankCard) { ... }
   method enterPIN (pin : int)         { ... }
...
}
```

method insertCard(card:BankCard)

Informal spec:
"Inserts a bank card into the ATM if the card slot is free and provided the card is valid."

pre and postconditions? (also not in informal spec?)

Preconditions:

- ATM card slot is free
- Card is valid
- (card is non null)

Postconditions:

- The ATM card slot is occupied
- Insertedcard = card
- (The user is not authenticated.)
- ((wrongPINCounter is 0)

Refresher: Logic

Want to express specification this completely formal, such that computer can enforce it, which language? *Logic!*

A small refresher/intro now on:

- Propositional logic
- SAT Solving
- SMT Solving
- Predicate logic

Propositional logic

A propositional logic formula is built from:

- (boolean) variables p,q,r ...
- connectives:

Connective	Means	Dafny syntax
¬p	not p	`!p`
p ⩛ q	p or q	`p || q`
p ⩓ q	p and q	`p && q`
p ⇒ q	if p then q	`p ==> q`
p ⇔ q	if p then q and vice versa	`p <==> q`

Propositional logic: Truth tables

Given a propositional formula, we can construct a truth table:

p	q	p ⊻ q	q ⇒ p	(p ⊻ q) ⅄ (q ⇒ p)
F	F	F	T	F
F	T	T	F	F
T	F	T	T	T
T	T	T	T	T

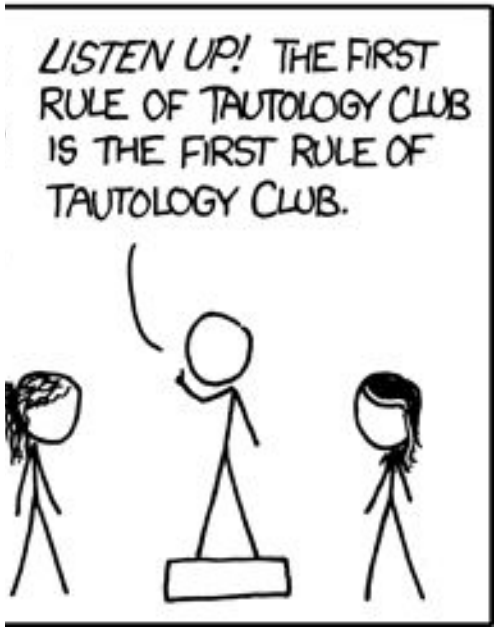
A propositional formula is...

- a *tautology* if the rightmost column is T for each row
- *satisfiable* if there is at least one row where the rightmost collumn is T

Kahoot!

Some tautologies

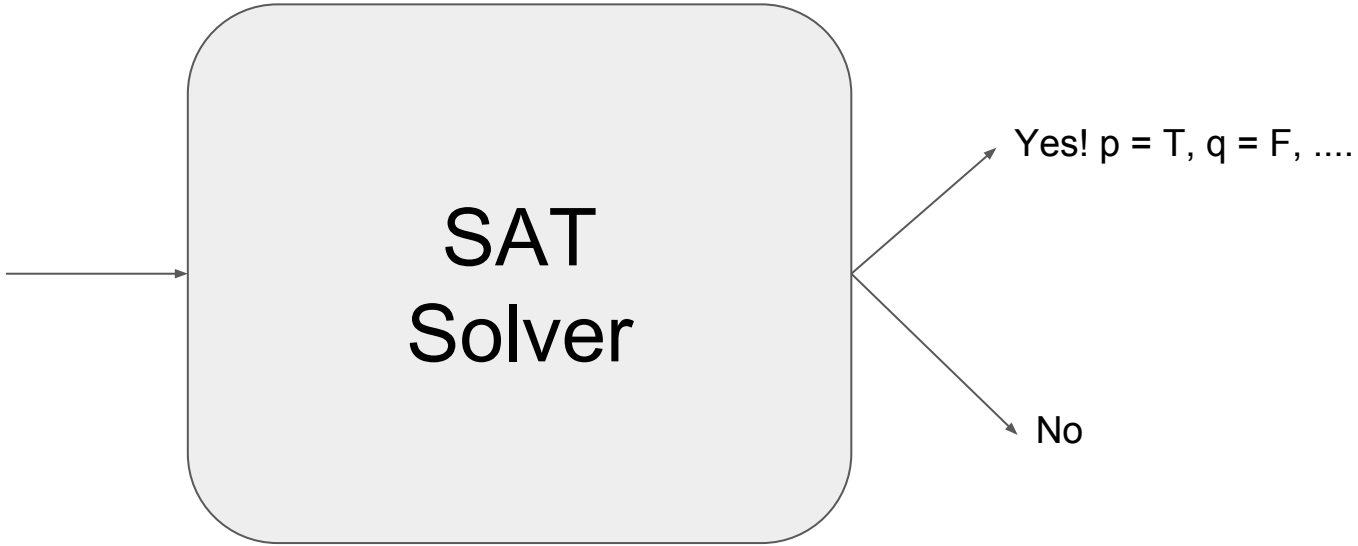
- $\neg\neg x \leftrightarrow x$
- $\neg(\varphi \wedge \psi) \leftrightarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \leftrightarrow \neg\varphi \wedge \neg\psi$
- $\text{false} \rightarrow \varphi$
- $(\varphi \rightarrow \psi) \leftrightarrow (\neg\varphi \vee \psi)$



Propositional Satisfiability Problem(SAT) Solver

Program that solves wether formula is satisfiable

Propositional formula
eg.
$(p \veebar q) \wedge (q \Rightarrow p)$
$p \wedge (q \Rightarrow \neg q) \wedge \neg p$



Can also be used to check if formula P is a tautology:
Check that ¬P is *not* satisfiable

NP complete-problem, but over last +- 15 years SAT solver have become very fast for many inputs!

Predicate logic + theory of linear inequalities

Predicate logic is not very expressive....

Let’s add:

- Variables of type Real
- Constants 0,0.1, 1,2,3,4,....
- Operations +, *, -, ≤, ≥, =

Example formulae:

$x + 2 * y \leq z \wedge z \geq x + 20$

$x + y \geq z \wedge x = z + 1$

Satisfiable?

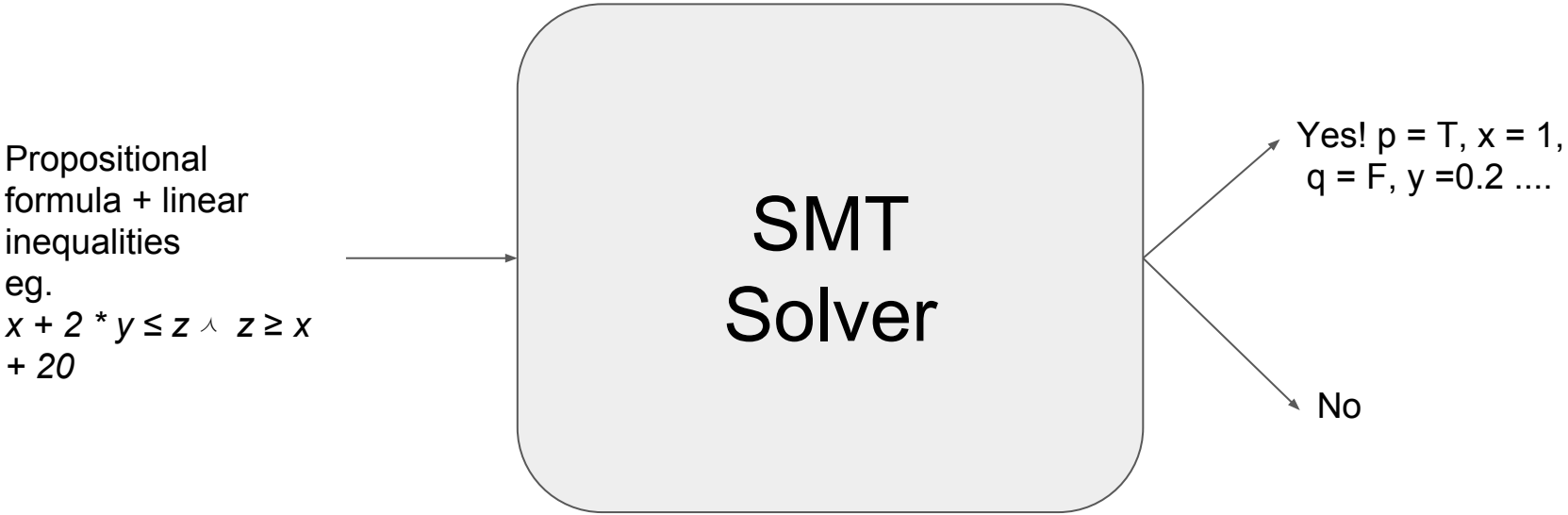
Yes: x = 0, y = 0, z = 30

Yes: x = 1, y = 6000, z = 0

Kahoot!

Satisfiablity modulo theories

Program that solves wether formula is satisfiable



NP complete-problem, but over last +- 15 years SMT solver have become very fast for many inputs!

Theories

linear inequalities is an example of a *theory,* an extension of predicate logic

Other (decicable) theories supported by SMT solvers:

- Arrays
- Bitvectors
- Uninterpreted functions

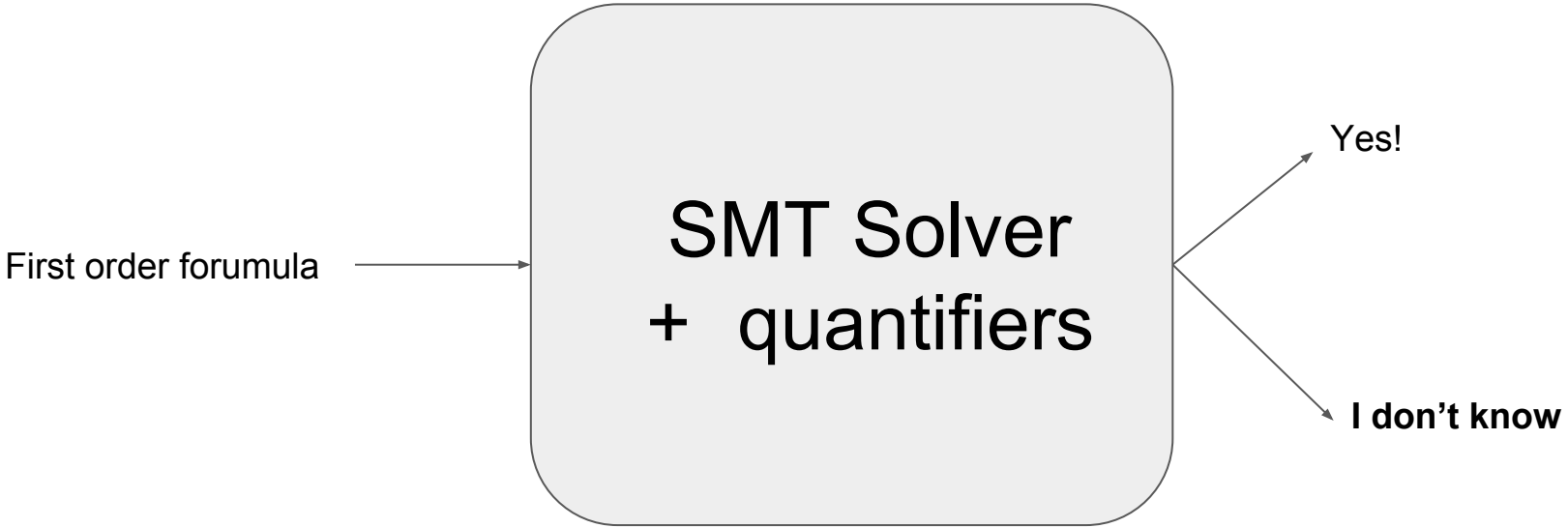
First order logic

Take Predicate logic + theories and add *quantifiers:*

Quantifier	*Meaning*	*Dafny*
∀ x : t, P	Forall x of type t, P holds	`forall x : t :: P`
∃ x : t, P	There exists at least one x of type t, such that P holds	exists x : t :: P

Example	
All elements in array a are bigger than zero	∀ i : int, 0 ≤ *i* < *a.length* ⇒ a[i] > 0
There is an element which is even in array a	∃ i : int, 0 ≤ *i* < *a.length* ∧ isEven(a[i])

Satisfiablity modulo theories + Quantifiers



Semidecidable problem, but often gives good results

Valid formulas

A first order logic formula is *valid* if it is true in every interpretation (however we interpret the functions and constants)

Examples:

- $\neg(\exists\ x : t.\ \neg\varphi) \leftrightarrow \forall\ x : t.\ \varphi$
- $(\forall\ x : t.\ \varphi \wedge \psi) \leftrightarrow (\forall\ x : t.\ \varphi) \wedge (\forall\ x : t.\ \psi)$
- $(\exists\ x : t.\ \varphi \vee \psi) \leftrightarrow (\exists\ x : t\ \varphi) \vee (\exists\ x : t.\ \psi)$

Non-examples:

Formula must hold for every interpretation of +,
does not have to be regular +, can also be for example:
a + b = 2

- $\forall\ x : \text{int}, x + 0 = x$
- $(\forall\ x : t.\ \varphi \vee \psi) \leftrightarrow (\forall\ x : t.\ \varphi) \vee (\forall\ x : t.\ \psi)$
- $(\exists\ x : t.\ \varphi \wedge \psi) \leftrightarrow\ (\exists\ x : t\ \varphi)\ \wedge\ (\exists\ x : t.\ \psi)$

Kahoot!

Formal specification examples

int[] sort(int[] a)

Requires: $a \neq null$
Ensures: $isSorted(sort(a)) \wedge isPermutationOf(sort(a),a)$

int binarySearch(int[] a,int elem)

Requires: $a \neq null \wedge isSorted(a)$
Ensures: $(result = -1 \wedge \forall\ i : int,\ 0 \leq i < a.length \Rightarrow a[i] \neq elem) \veebar$
$(a[result] = elem \wedge \forall\ i : int,\ 0 \leq i < result \Rightarrow a[i] \neq elem)$

More examples:

int maximum(int[] a)

Let's try

Requires: a ≠ null /\ a is non-empty
Ensures: geqAll(a, result) ⩚ exists i : int, a[i] = result
geqAll(int[] a, int elem) = ∀ i : int, 0 ≤ *i* < *a.length* ⇒ a[i] ≤ elem

Conclusion & Next times

Today we say:

- Formal specification: what and why
- A first glancy at Dafny
- Intro/Refresher on logic

This Wednesday:

Stateful property based testing + guest lecture John Hughes

Next week:

More formal specification & Dafny!