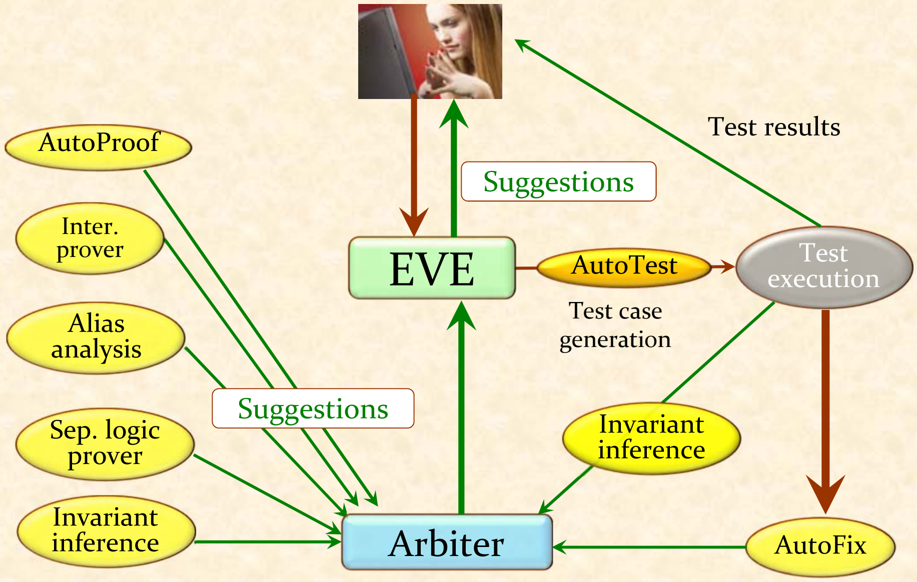
POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

Introduction to AutoProof and theory-based (model-based) specifications

Bertrand Meyer
Toulouse, 29 June 2017

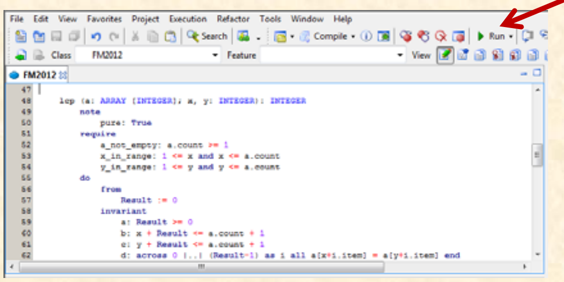
Verification As a Matter Of Course



Auto-active user/tool interaction

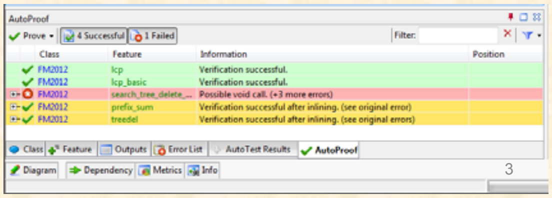
1. Code + Annotations

2. Push button



```
lcp (a: ARRAY [INTEGER]; x, y: INTEGER): INTEGER
    note
        pure: True
    require
        a_not_empty: a.count >= 1
        x_in_range: 1 <= x and x <= a.count
        y_in_range: 1 <= y and y <= a.count
    do
        from
            Result := 0
        invariant
            a: Result >= 0
            b: x + Result <= a.count + 1
            c: y + Result <= a.count + 1
            d: across 0 |..| (Result-1) as i all a[x+i.item] = a[y+i.item] end
```

3. Verification outcome



	Class	Feature	Information	Position
✓	FM2012	lcp	Verification successful.	
✓	FM2012	lcp_basic	Verification successful.	
⊗	FM2012	search_tree_delete_...	Possible void call. (+3 more errors)	
✓	FM2012	prefix_sum	Verification successful after inlining. (see original error)	
✓	FM2012	treedel	Verification successful after inlining. (see original errors)	

4. Correct/Revise

Tools for different users

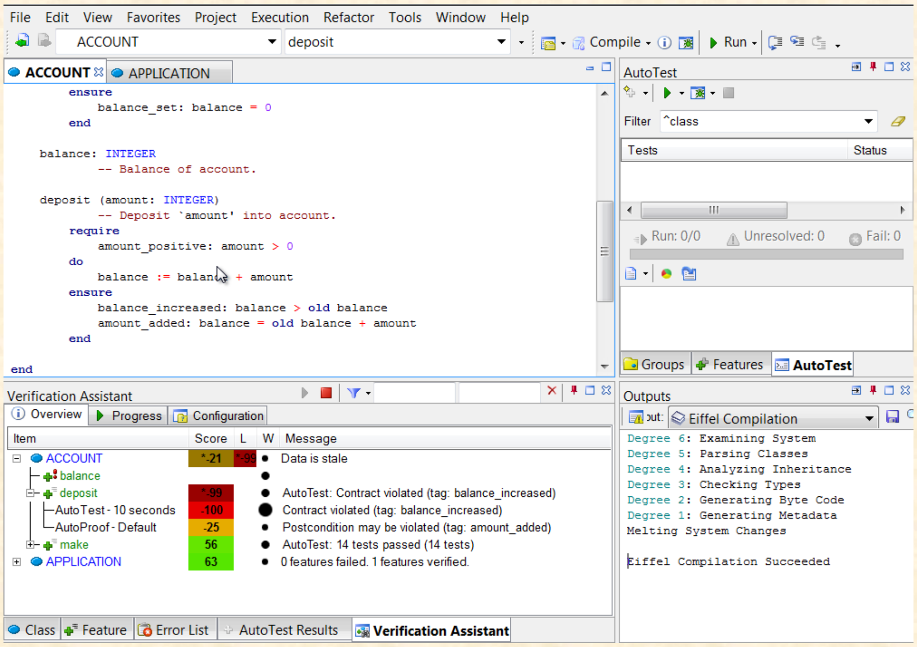
Verification expert

- full-fledged verification: AutoProof (auto-active prover)
- strong specs: AutoInfer (dynamic inference)

Joe the programmer

- find bugs: AutoTest (random testing)
- fix bugs: AutoFix (program repair)
- weak specs: two-step verification and implicit contracts

Verification As a Matter Of Course



```
        ensure
            balance_set: balance = 0
        end

    balance: INTEGER
            -- Balance of account.

    deposit (amount: INTEGER)
            -- Deposit `amount' into account.
        require
            amount_positive: amount > 0
        do
            balance := balance + amount
        ensure
            balance_increased: balance > old balance
            amount_added: balance = old balance + amount
        end

end
```

Item	Score	L	W	Message
ACCOUNT	*-21	*-99	●	Data is stale
balance			●	
deposit	*-99		●	AutoTest: Contract violated (tag: balance_increased)
AutoTest - 10 seconds	-100		●	Contract violated (tag: balance_increased)
AutoProof - Default	-25		●	Postcondition may be violated (tag: amount_added)
make	56		●	AutoTest: 14 tests passed (14 tests)
APPLICATION	63		●	0 features failed. 1 features verified.

```
Degree 6: Examining System
Degree 5: Parsing Classes
Degree 4: Analyzing Inheritance
Degree 3: Checking Types
Degree 2: Generating Byte Code
Degree 1: Generating Metadata
Melting System Changes

Eiffel Compilation Succeeded
```

Debugging failed verification

When verification fails with verifiers such as AutoProof (modular, sound, incomplete):

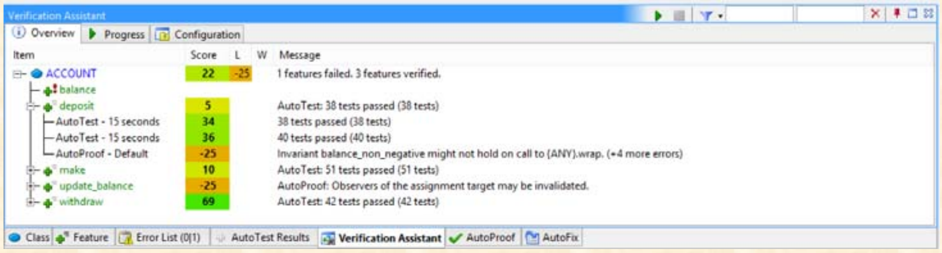
- Is there a bug?
- Is the program correct, but the specification insufficient?

To help debug failed verification attempts AutoProof features two-step verification.

Combined testing and proving

The verification assistant runs on the version of ACCOUNT patched by AutoFix:

deposit cannot be proved, but passes all tests
→ reasonable confidence in its correctness.



Item	Score	L	W	Message
ACCOUNT	22	-25		1 features failed. 3 features verified.
balance				
deposit	5			AutoTest: 38 tests passed (38 tests)
AutoTest - 15 seconds	34			38 tests passed (38 tests)
AutoTest - 15 seconds	36			40 tests passed (40 tests)
AutoProof - Default	-25			Invariant balance_non_negative might not hold on call to {ANY}.wrap. (+4 more errors)
make	10			AutoTest: 51 tests passed (51 tests)
update_balance	-25			AutoProof: Observers of the assignment target may be invalidated.
withdraw	69			AutoTest: 42 tests passed (42 tests)

Static verification: correctness proofs

Bernd Schoeller Julian Tschannen, Nadia Polikarpova, Carlo Furia et al
VSTTE, ICSE etc.

Software verification, static techniques: AutoProof

- Automatic correctness proofs of OO software equipped with contracts
- Full correctness (using technique of "model queries")
- Uses MSR's Boogie (and Z3)
- Covers most of Eiffel language, including tricky constructs such as exceptions
- Online tutorial
- Major achievement: correctness proof of full data structure & algorithms library (EiffelBase 2)

FNS, Hasler

The AutoProof technology stack

Eiffel classes with complete contracts

AutoProof

Boogie (prover)

Z3 (SMT solver)

Rustan Leino

Nikolaj Bjorner

Leonardo de Moura

AutoProof in a nutshell

AutoProof is an auto-active verifier for Eiffel

Prover for functional properties

All-out support of object-oriented idiomatic structures (e.g. patterns)

- Based on class invariants

Flexible: incrementality

- Proving simple properties requires little annotation
- Proving complex properties is possible with more effort

AutoProof on production software

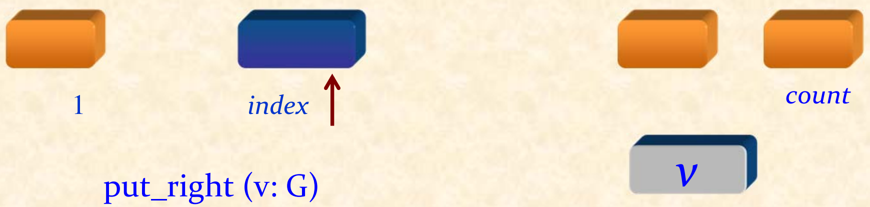
Verification benchmarks:

# programs	LOC	SPEC/CODE	Verification time
25	4400	Lines: 1.0 Tokens: 1.9	Total: 3.4 min Longest method: 12 sec Average method: < 1 sec

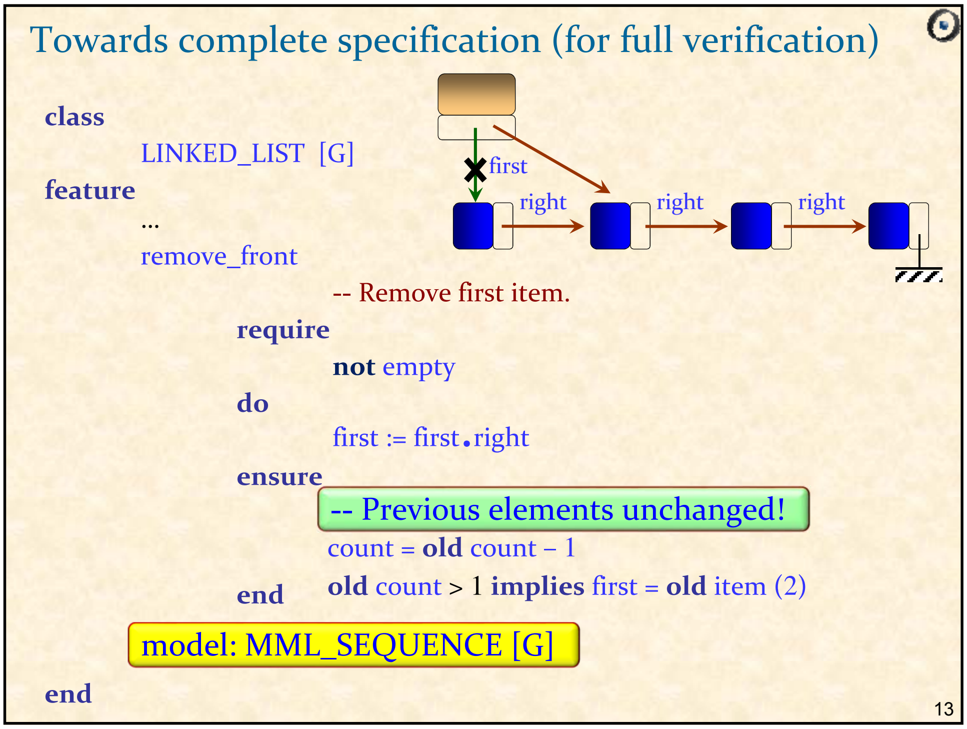
EiffelBase2 – a full container library:

# classes	LOC	SPEC/CODE	Verification time
46	8400	Lines: 1.4 Tokens: 2.7	Total: 7.2 min Longest method: 12 sec Average method: < 1 sec

An example: list insertion



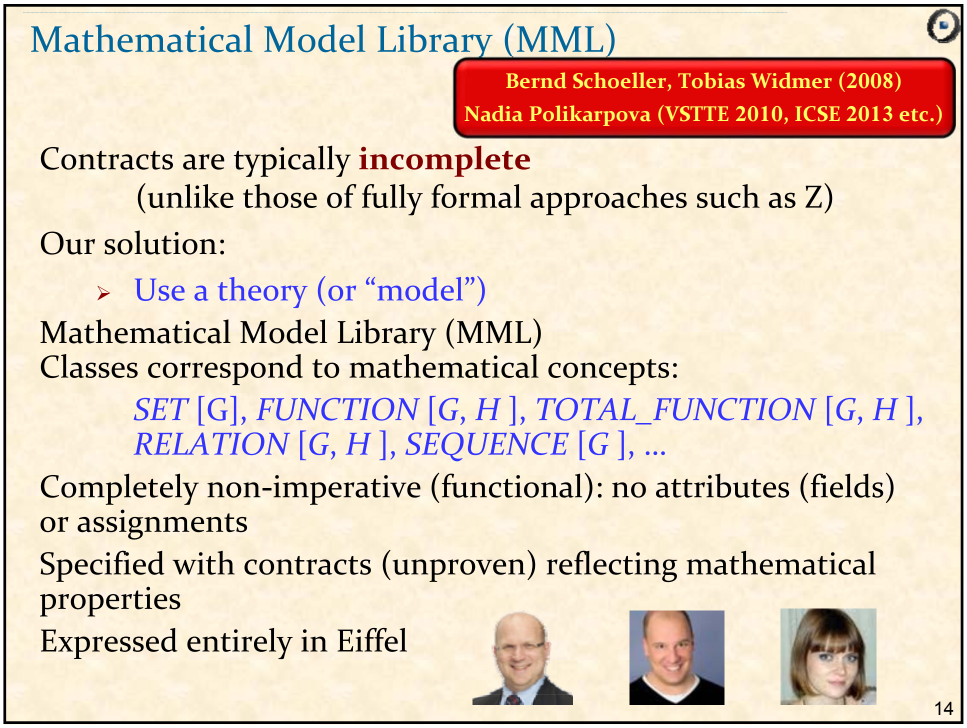
```
put_right (v: G)
                -- Insert v to the right of cursor
        require
                index <= count
        ensure
                i_th (index + 1) = v
                count = old count + 1
                index = old index
                -- Previous elements unchanged!
        end
```



Towards complete specification (for full verification)

```
class
        LINKED_LIST  [G]
feature
        ...
        remove_front
                        -- Remove first item.
                require
                        not empty
                do
                        first := first.right
                ensure
                        -- Previous elements unchanged!
                        count = old count – 1
                end     old count > 1 implies first = old item (2)
        model: MML_SEQUENCE [G]
end
```

first
right
right
right



Mathematical Model Library (MML)

Bernd Schoeller, Tobias Widmer (2008)
Nadia Polikarpova (VSTTE 2010, ICSE 2013 etc.)

Contracts are typically **incomplete**
(unlike those of fully formal approaches such as Z)

Our solution:

- Use a theory (or "model")

Mathematical Model Library (MML)

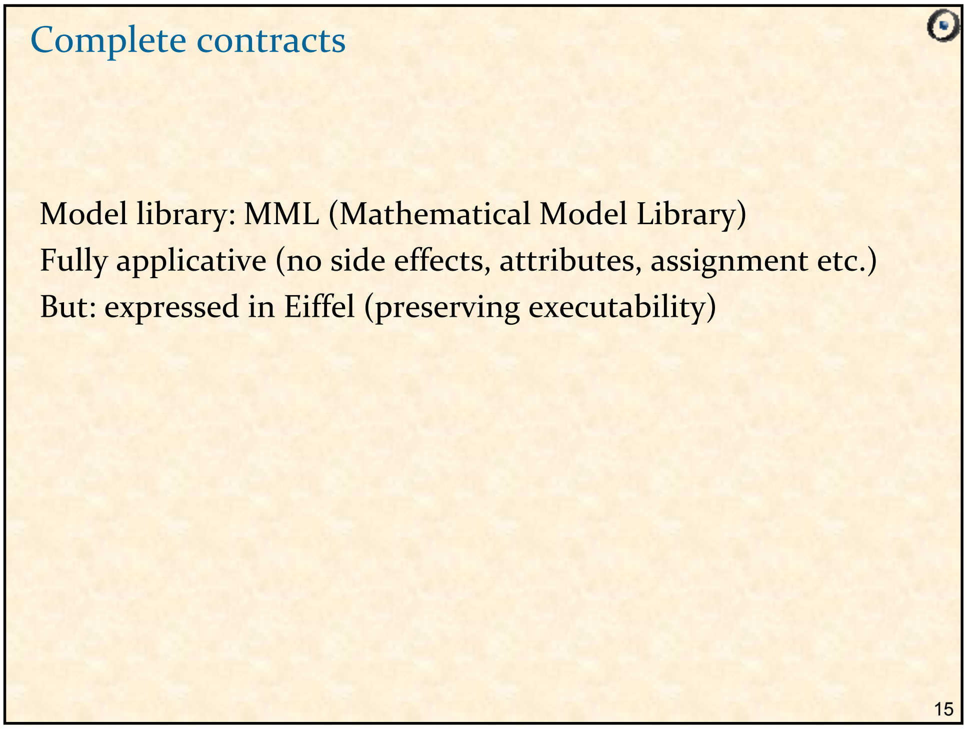
Classes correspond to mathematical concepts:

SET [G], *FUNCTION* [*G*, *H*], *TOTAL_FUNCTION* [*G*, *H*], *RELATION* [*G*, *H*], *SEQUENCE* [*G*], ...

Completely non-imperative (functional): no attributes (fields) or assignments

Specified with contracts (unproven) reflecting mathematical properties

Expressed entirely in Eiffel

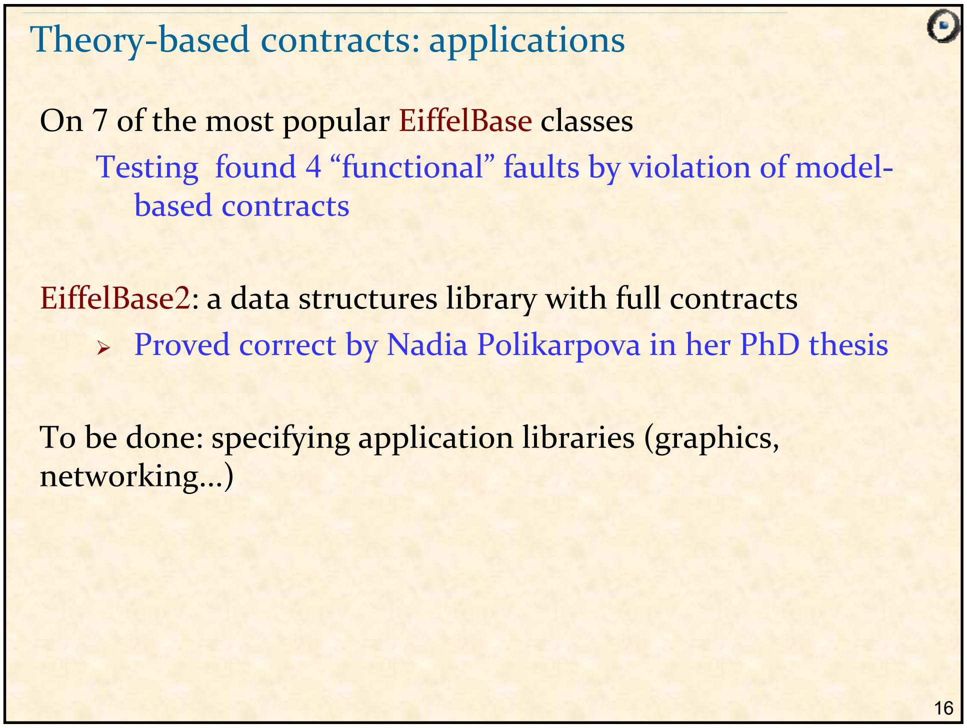


Complete contracts

Model library: MML (Mathematical Model Library)

Fully applicative (no side effects, attributes, assignment etc.)

But: expressed in Eiffel (preserving executability)



Theory-based contracts: applications

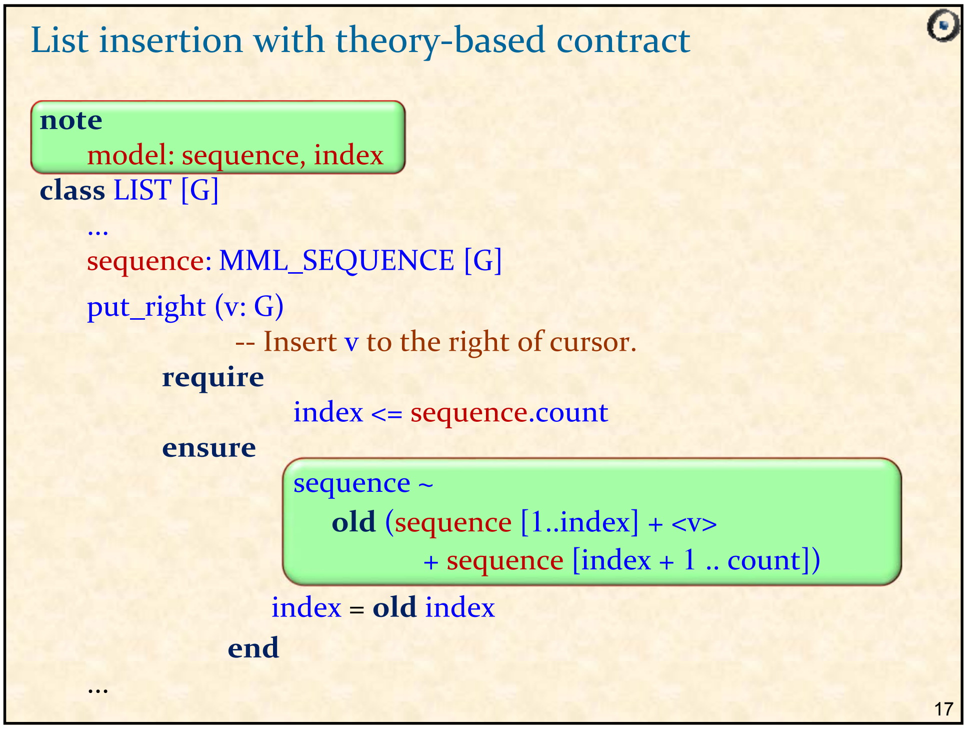
On 7 of the most popular EiffelBase classes

Testing found 4 "functional" faults by violation of model-based contracts

EiffelBase2: a data structures library with full contracts

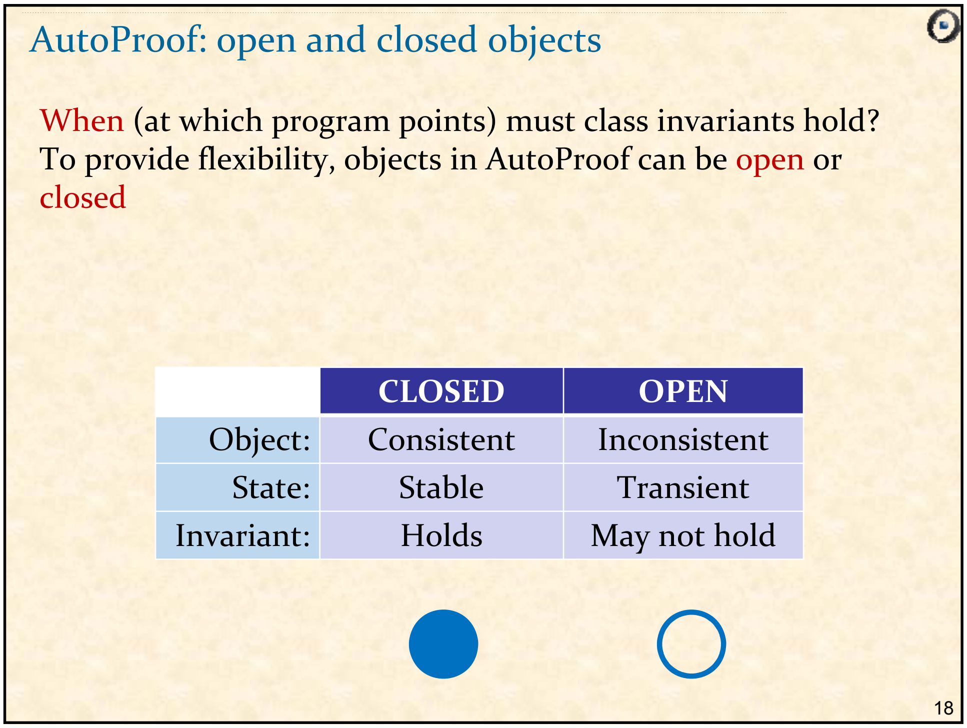
- Proved correct by Nadia Polikarpova in her PhD thesis

To be done: specifying application libraries (graphics, networking...)



List insertion with theory-based contract

```
note
    model: sequence, index
class LIST [G]
    ...
    sequence: MML_SEQUENCE [G]
    put_right (v: G)
            -- Insert v to the right of cursor.
        require
                index <= sequence.count
        ensure
                sequence ~
                    old (sequence [1..index] + <v>
                            + sequence [index + 1 .. count])
              index = old index
          end
    ...
```

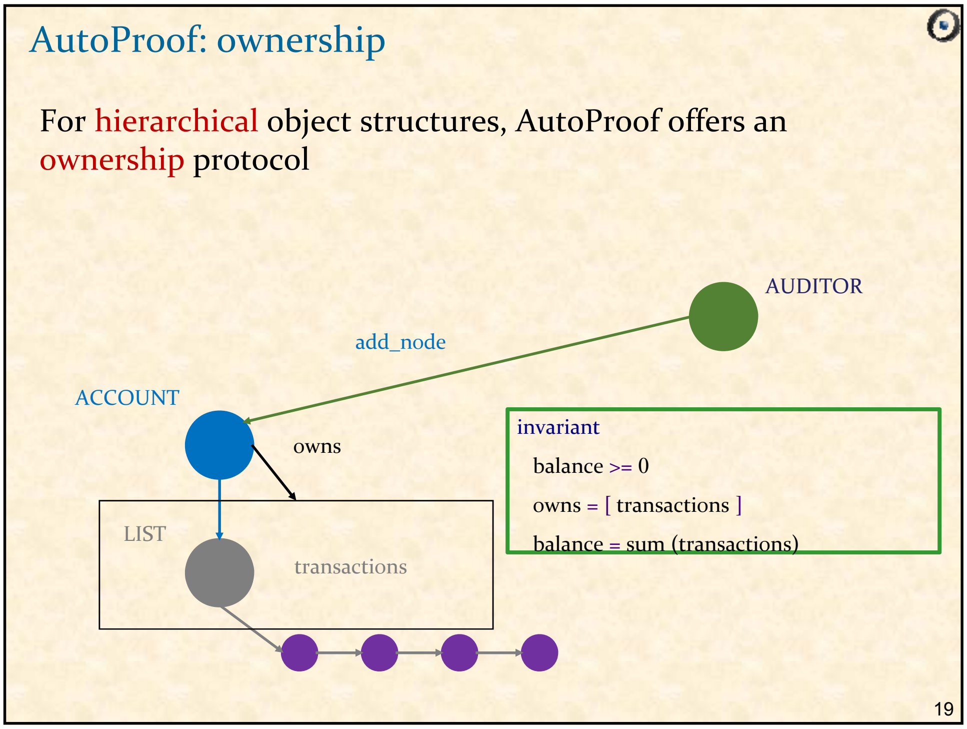


AutoProof: open and closed objects

When (at which program points) must class invariants hold?
To provide flexibility, objects in AutoProof can be open or closed

	CLOSED	OPEN
Object:	Consistent	Inconsistent
State:	Stable	Transient
Invariant:	Holds	May not hold

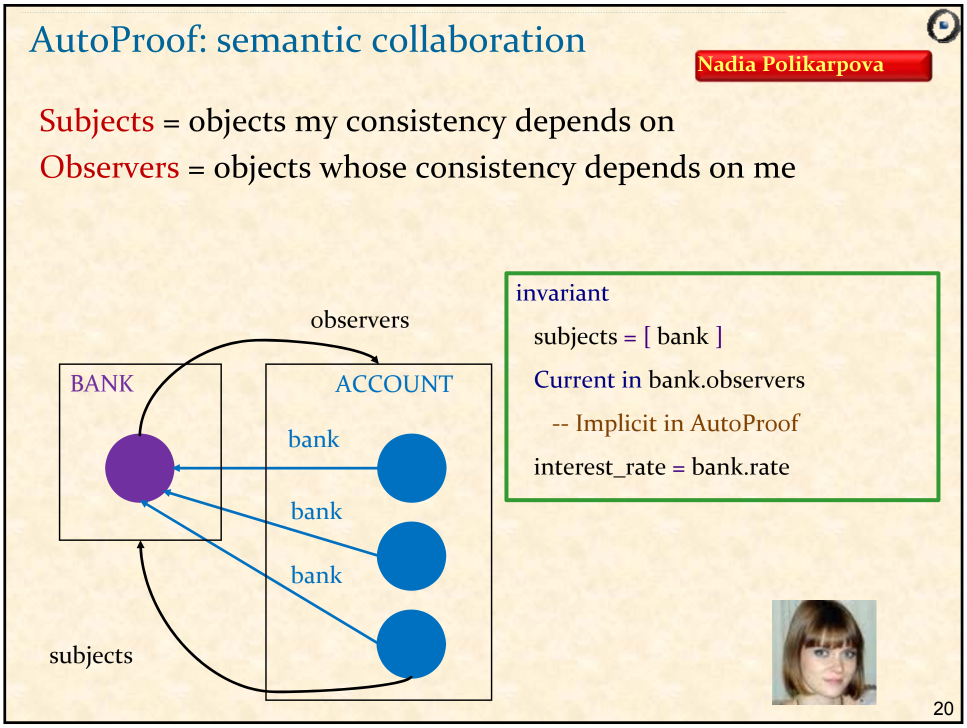
● ○



AutoProof: ownership

For hierarchical object structures, AutoProof offers an ownership protocol

```
invariant
   balance >= 0
   owns = [ transactions ]
   balance = sum (transactions)
```



AutoProof: semantic collaboration

Nadia Polikarpova

Subjects = objects my consistency depends on

Observers = objects whose consistency depends on me

```
invariant
   subjects = [ bank ]
   Current in bank.observers
      -- Implicit in AutoProof
   interest_rate = bank.rate
```