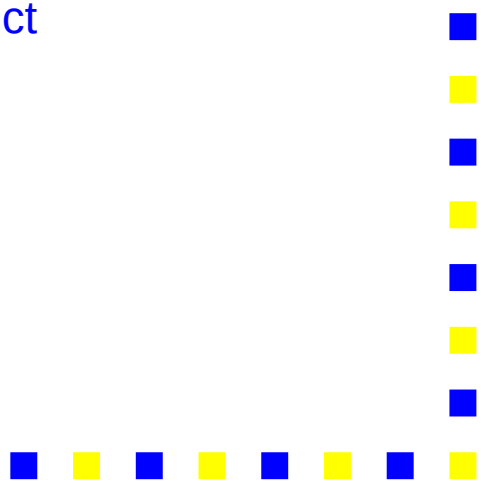
High-level languages for security properties

Marieke Huisman

Marieke.Huisman@inria.fr

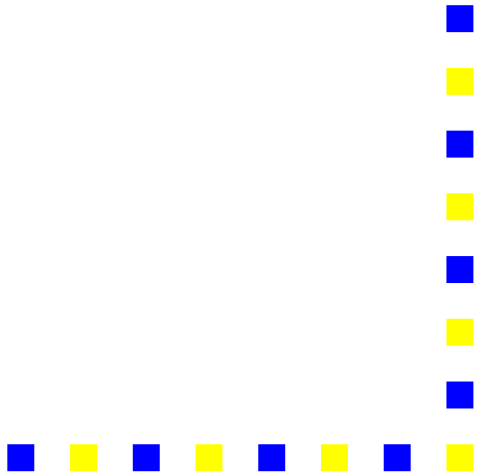
INRIA Sophia Antipolis

Contributions by LIFC, Vasco and Everest project



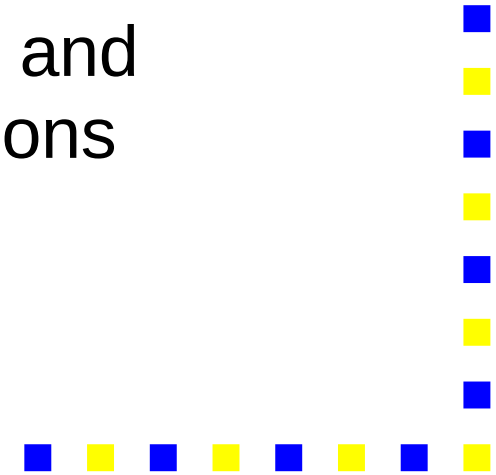
Overview

- Motivation
- Contributions
- Tools, implementations and on-going work



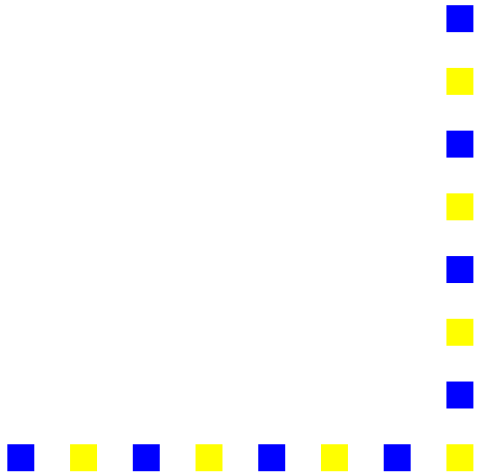
Enforcing security properties

- Security expert: from global notions of security to set of rules
- Developers: try to obey rules
- Security audit: manual code inspection whether rules obeyed
- Need for tools to help security audit
- Gap between global notion and rules, and between security rules and specifications understood by tools



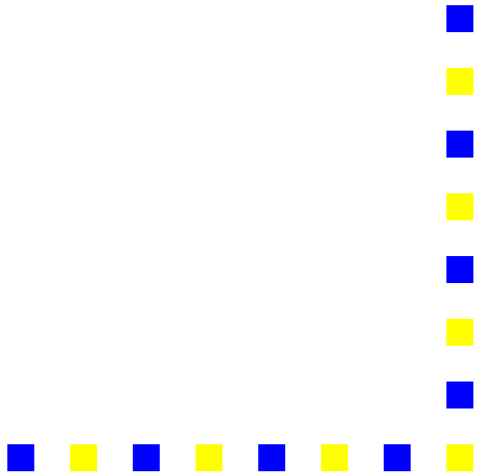
Typical examples of security rules

- Atomicity
- Applet life cycle
- Exception handling
- Access control



Two different approaches

- Express security properties in existing language
  Advantage: reuse of existing tools
- Develop specific languages
  Advantage: tailored techniques



Use of existing languages

- Different formats available to express such properties
  - Different kinds of automata
  - Temporal logic
- Define mapping of such a format into JML annotations
- Safety and liveness properties
- Implementation
  - JAG: from temporal property to JML
  - JACK: propagation and verification

Design of specific languages

- Concrete applications used to identify relevant properties
- Results:
  - Property language for Génésyst
  - Access control policy language for MECA
  - JTPL (Java Temporal Pattern Language)

JAG: JML Annotation Generator

- Generates JML annotations for a given temporal property, including liveness properties
- Tool accepts different input formats (internal representation with Büchi automata)
- For liveness properties a progress condition on the environment has to be established: use of Hoare logic with progress condition

$$\{|P|\}S\{|Q|\}$$

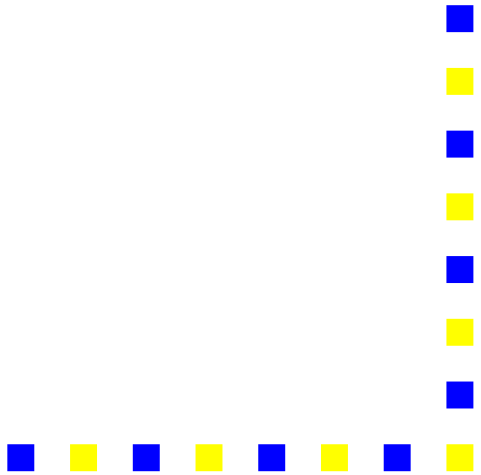
  - Partial correctness $\{P\}S\{Q\}$
  - If S diverges, it satisfies a progress hypothesis

Current work: generalisation to other programming/annotation languages

- Define trace-based semantics of programming language
- Define semantics of the annotations
- Characterising the annotations needed for the verification of LTL properties
- Characterising verification conditions for the appropriate annotations

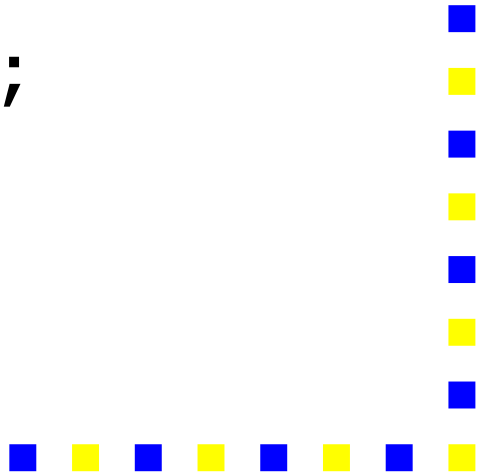
JACK: Java Applet Correctness Kit

- Implements an annotation propagation algorithm
- Core-annotations for methods directly involved in the method
- Propagation ensures possibility of static verification



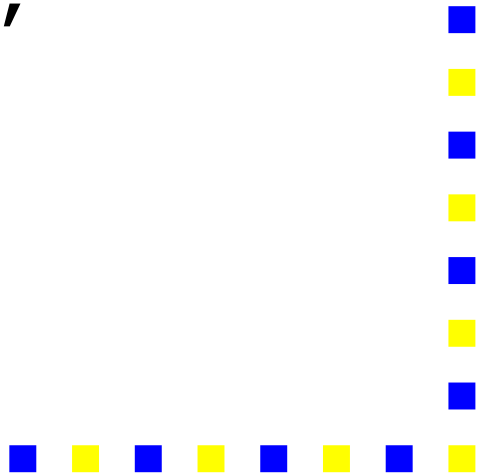
Propagation example

```
public void m() {
    ...
    // requires TRANSACT == 0
    JCSystem.beginTransaction();
    // modifies TRANSACT
    // ensures TRANSACT == 1
    ...
    // requires TRANSACT == 1
    JSSystem.commitTransaction();
    // modifies TRANSACT
    // ensures TRANSACT == 0
    ...
    }
```



Propagation example

```
//@ requires TRANSACT == 0;
//@ modifies TRANSACT;
//@ ensures TRANSACT == 0;
public void m() {
    ...
    JCSystem.beginTransaction();
    ...
    JSSystem.commitTransaction();
    ...
    }
```



Current work: formalisation of the annotation generation

- Properties expressed as MVA automata
  - Transitions labelled with method names
  - Conditional branching
  - Can inspect program state
  - Transitions can update automaton state
- MVA monitors program execution
- MVA state encoded with ghost variables, transitions defined as appropriate set-statements, unreachable error state expressed with invariant

Current work: formalisation of the annotation generation

- Two step translation, using special preset and postset constructs as part of the method specification
- Monitoring program with MVA does not get stuck iff run-time checking does not find (new) assertion violation
- Future work: extend with propagation, and prove static verification result

MECA: Models for Access Control

- Tool to describe different types of access control policies
  - DAC - Discretionary Access Control
  - MAC - Mandatory Access Control
  - RBAC - Role Based Access Control
- Access control policy and application modeled as B-machines
- Meca defines necessary information to describe the access control policy
- Meca produces a B reference monitor enforcing the security policy (precondition for verification, condition for monitoring)

To summarise

- Two different approaches:
  - Development of special targeted language
  - Encoding of existing format in annotations
- Work continues in different directions:
  - Generalisation
  - Formalisation