JAG
Recent and Future Works

Julien Groslambert Alain Giorgetti

Juin 2006 - Réunion Geccoo - Sophia-Antipolis

JAG: Remainder

Formal Verification of conformity between

- Requirements
- Implementation source code

Proposed Approach

Expressing **security properties** from requirements

Translating properties into **verification language** for the implementation

Verifying the properties on the **code**.

JAG: Remainder

Formal Verification of conformity between

- Requirements
- Implementation source code

Proposed Approach

Expressing **security properties** from requirements

Translating properties into **verification language** for the implementation

Verifying the properties on the **code**.

Temporal Logic

JAG: Remainder

Formal Verification of conformity between

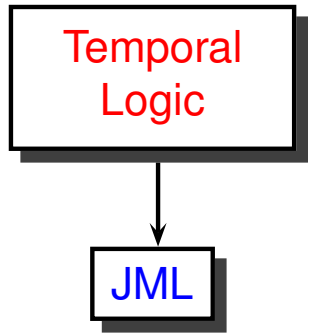
- Requirements
- Implementation source code

Proposed Approach

Expressing **security properties** from requirements

Translating properties into **verification language** for the implementation

Verifying the properties on the **code**.



JAG: Remainder

Formal Verification of conformity between

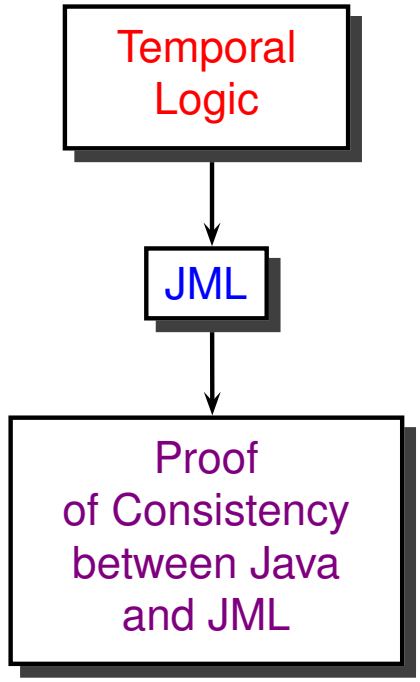
- Requirements
- Implementation source code

Proposed Approach

Expressing **security properties** from requirements

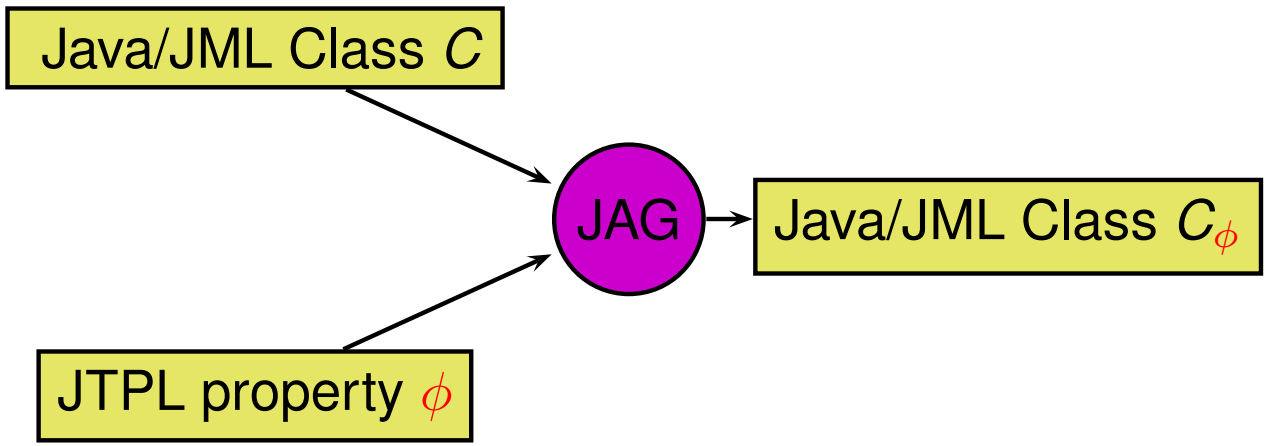
Translating properties into **verification language** for the implementation

Verifying the properties on the **code**.



JAG release 0.1

- Input: JTPL Formulae (Java Temporal Pattern Logic) [AMAST'02]
- Output: JML annotations



Extensions of JAG

Two way considered

- Other input format
  - LTL
  - Büchi Automata
  - Control Flow Automata
  - UML State Diagrams
- Other target languages
  - C♯/Spec♯
  - C/Caduceus
  - Java/AspectJ
  - ...

⇒ **Need a more generic structure**

Extensions of JAG

Two way considered

- Other input format
  - LTL
  - Büchi Automata
  - Control Flow Automata
  - UML State Diagrams
- Other target languages
  - C♯/Spec♯
  - C/Caduceus
  - Java/AspectJ
  - ...

⇒ **Need a more generic structure**

Extensions of JAG

Two way considered

- Other input format
  - LTL
  - Büchi Automata
  - Control Flow Automata
  - UML State Diagrams
- Other target languages
  - C♯/Spec♯
  - C/Caduceus
  - Java/AspectJ
  - ...

⇒ **Need a more generic structure**

Extensions of JAG

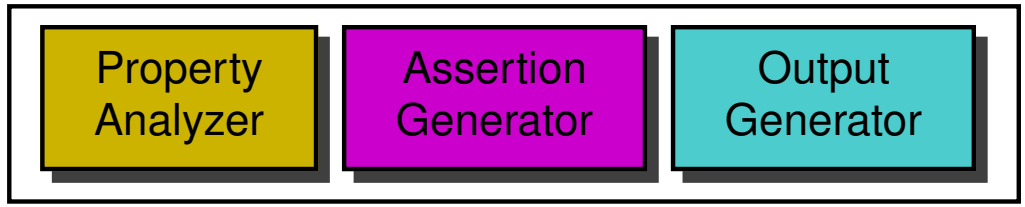
Two way considered

- Other input format
  - LTL
  - Büchi Automata
  - Control Flow Automata
  - UML State Diagrams
- Other target languages
  - C♯/Spec♯
  - C/Caduceus
  - Java/AspectJ
  - ...

⇒ **Need a more generic structure**

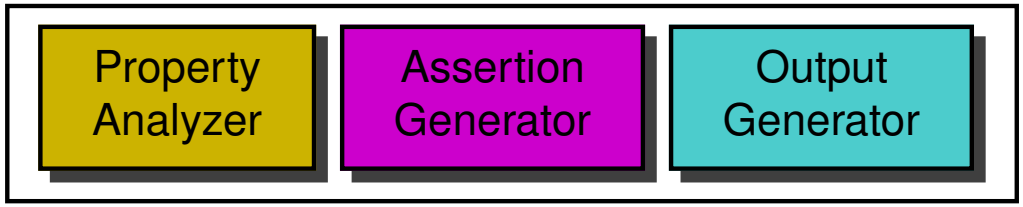
Needs and Proposed Solutions

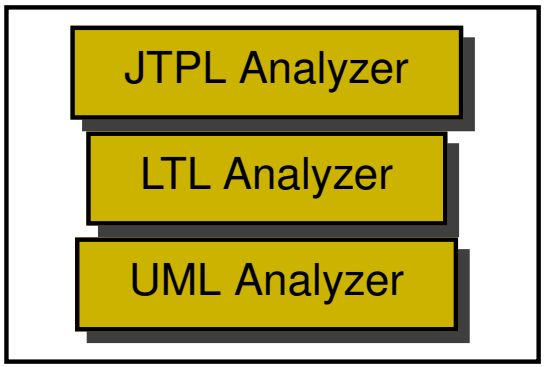
- Three needs
  - Analysis of several kind of properties.
  - Genericity of the annotation generated.
  - Several target languages support
- **Annotation Generator** composed of three modules

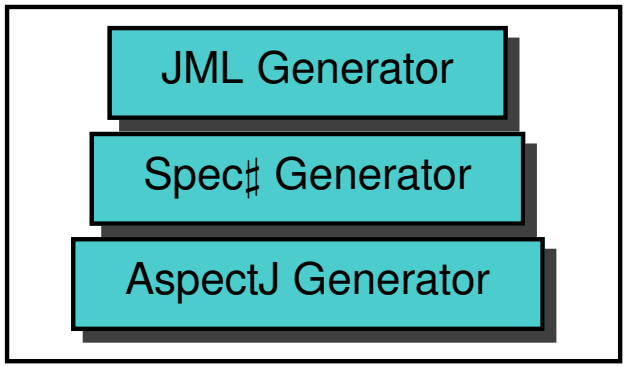


Use of inheritance

- Common Annotation Generator (internal language).
- Subclasses of a generic property analyzer.
- Subclasses of a generic output generator.

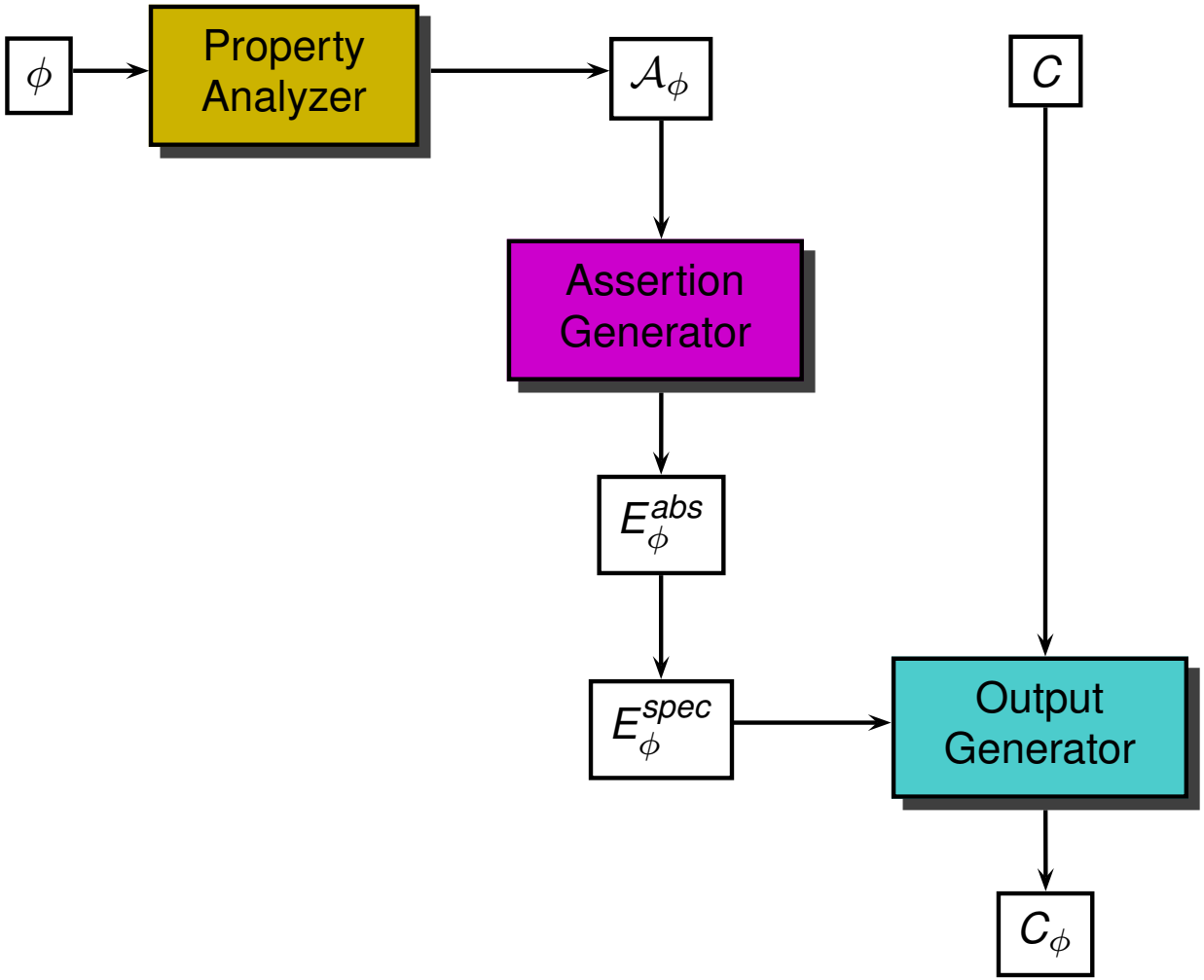




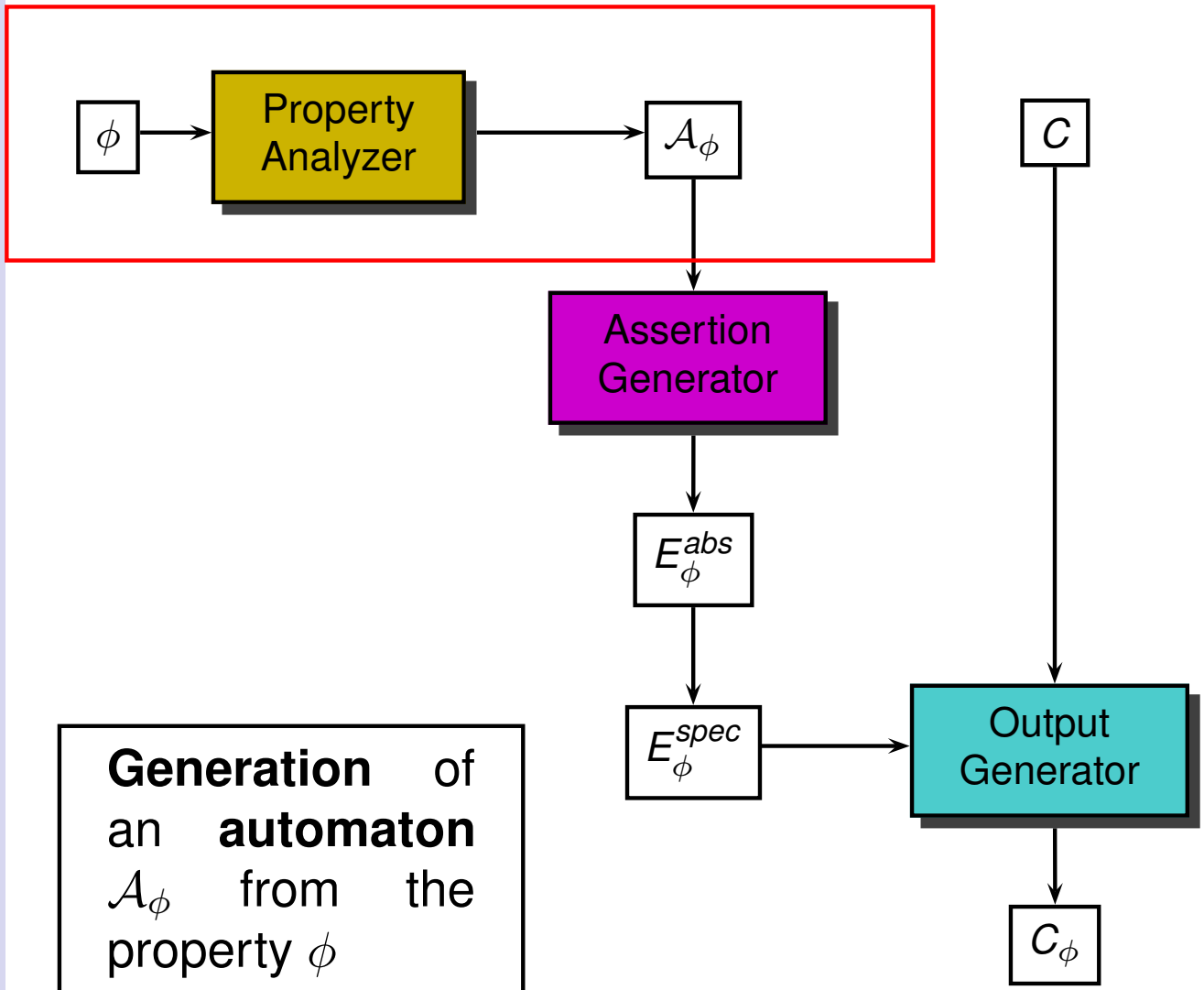


Use of Analyzer/Generator Couple

JAG Structure

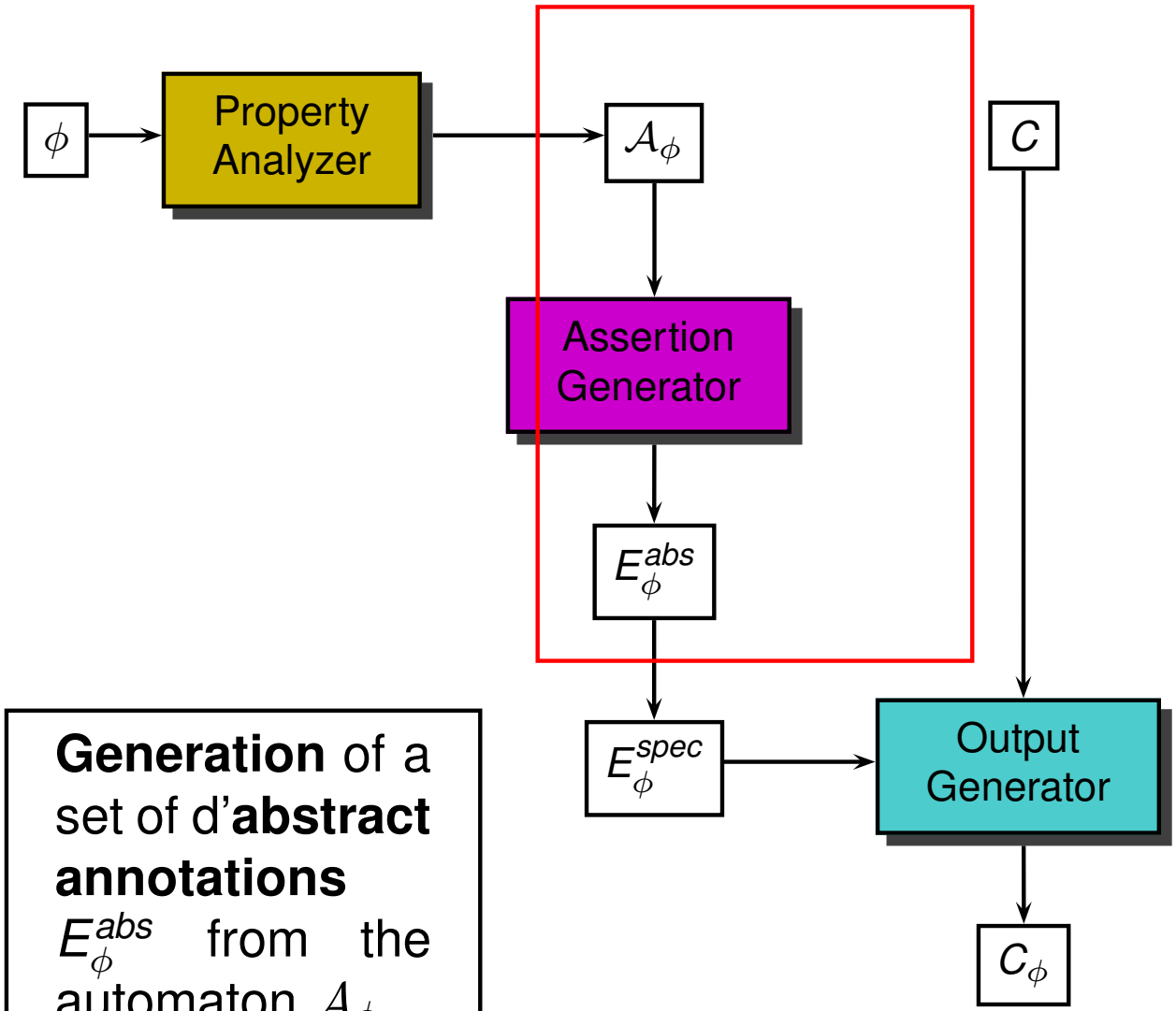


JAG Structure



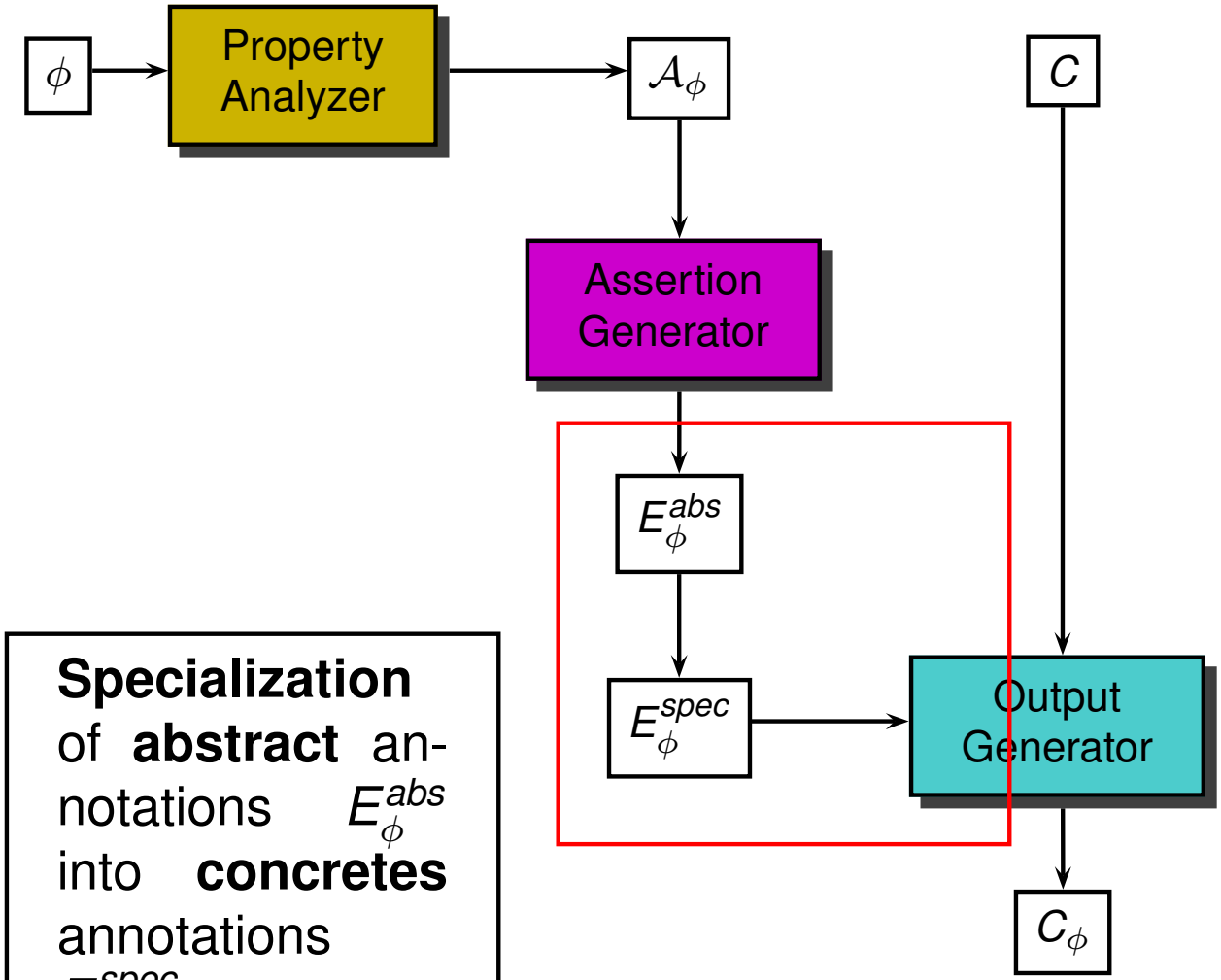
Generation of an **automaton** $\mathcal{A}_\phi$ from the property ϕ

JAG Structure



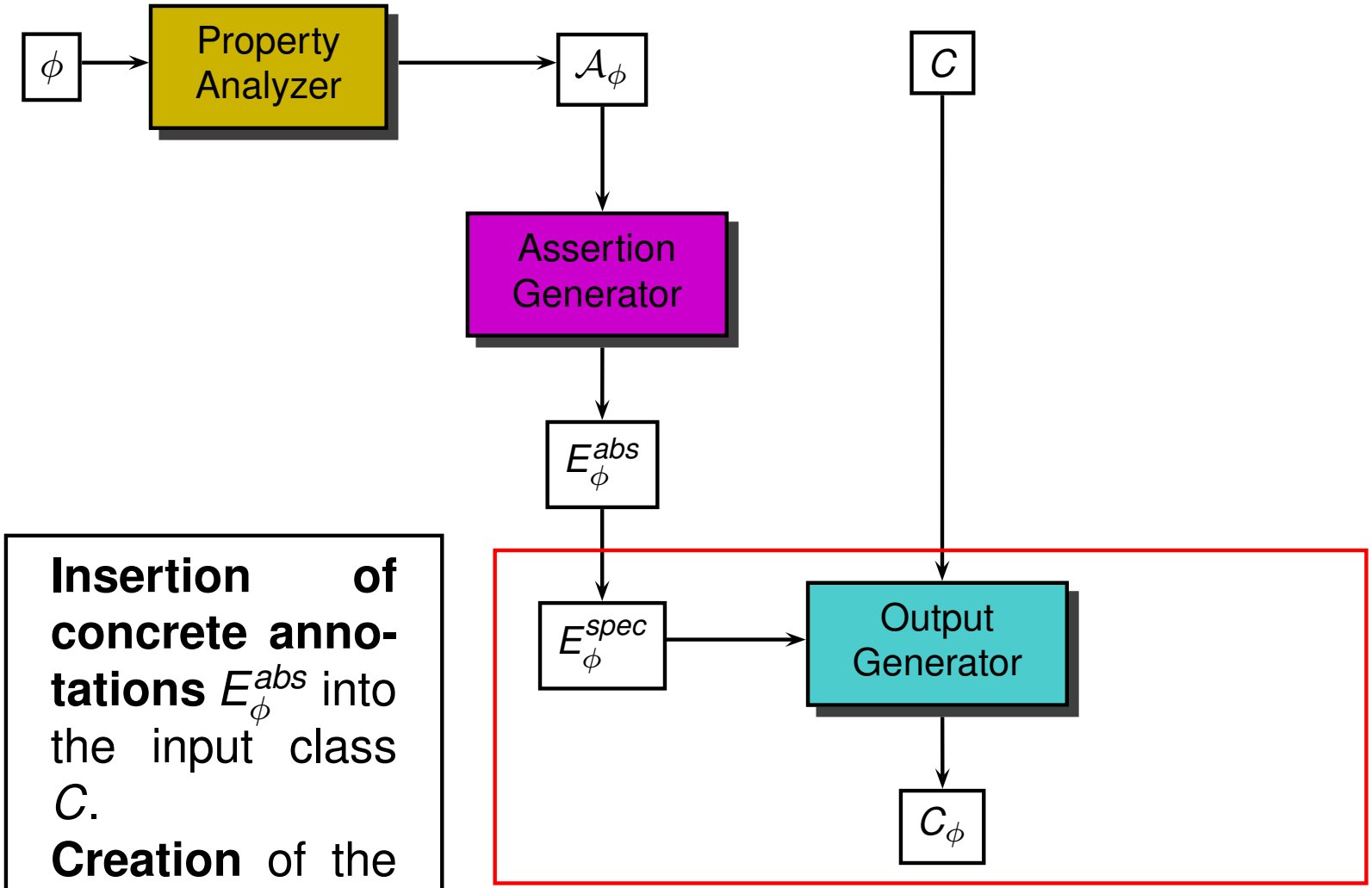
Generation of a set of d'**abstract annotations** E_ϕ^{abs} from the automaton $\mathcal{A}_\phi$

JAG Structure



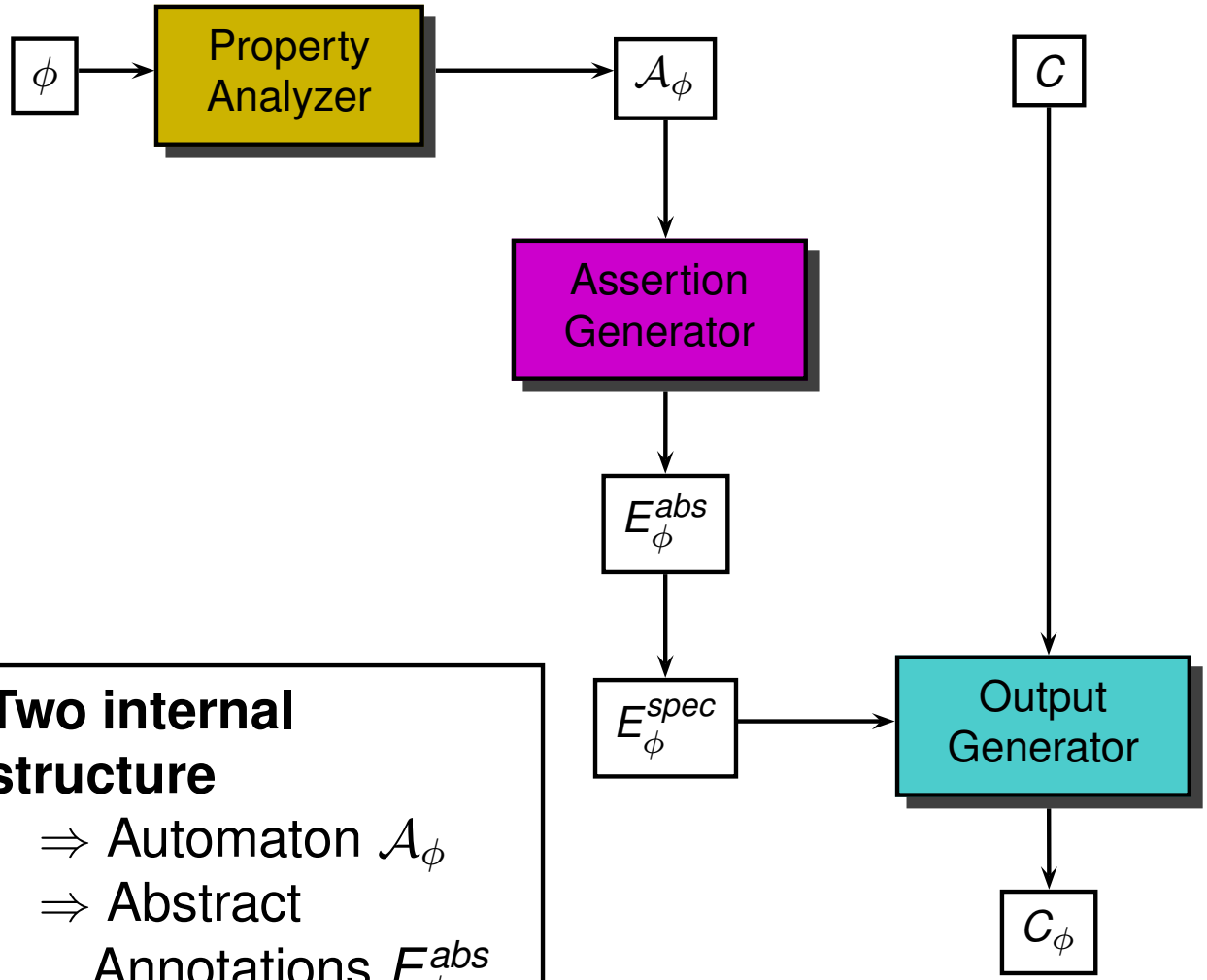
Specialization of **abstract** annotations E_ϕ^{abs} into **concretes** annotations E_ϕ^{spec}

JAG Structure



Insertion of concrete annotations E_ϕ^{abs} into the input class C.
Creation of the class C_ϕ

JAG Structure



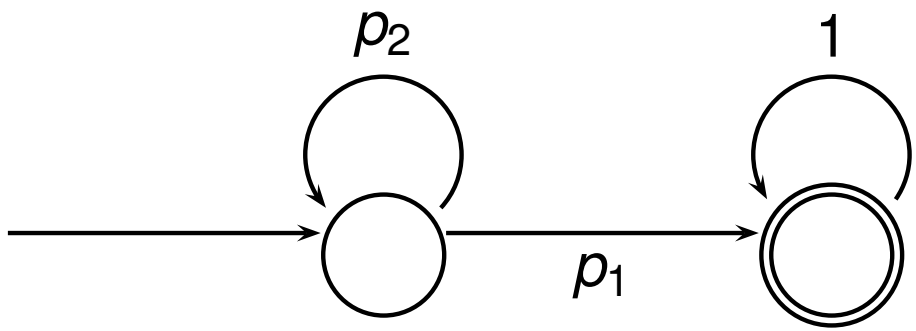
Two internal structure

$\Rightarrow$ Automaton $\mathcal{A}_\phi$

$\Rightarrow$ Abstract Annotations E_ϕ^{abs}

Büchi Automata

- Finite set S of states.
- Set $S_a \subseteq S$ of accepting states
- Alphabet A
- Set of transitions $T \subseteq S \times A \times S$



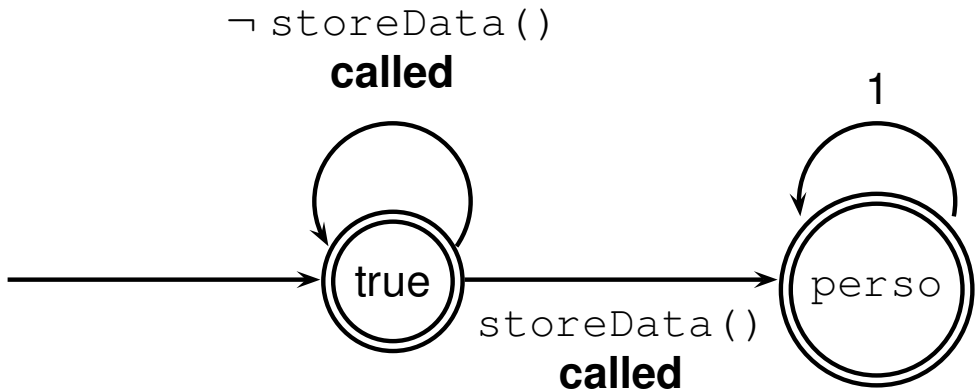
Extension of Büchi Automaton

- State Label
  - Predicate
  - State Properties: M enabled M not enabled
  - ...
- Transition label
  - Event (m called...)
  - Predicate
  - ...

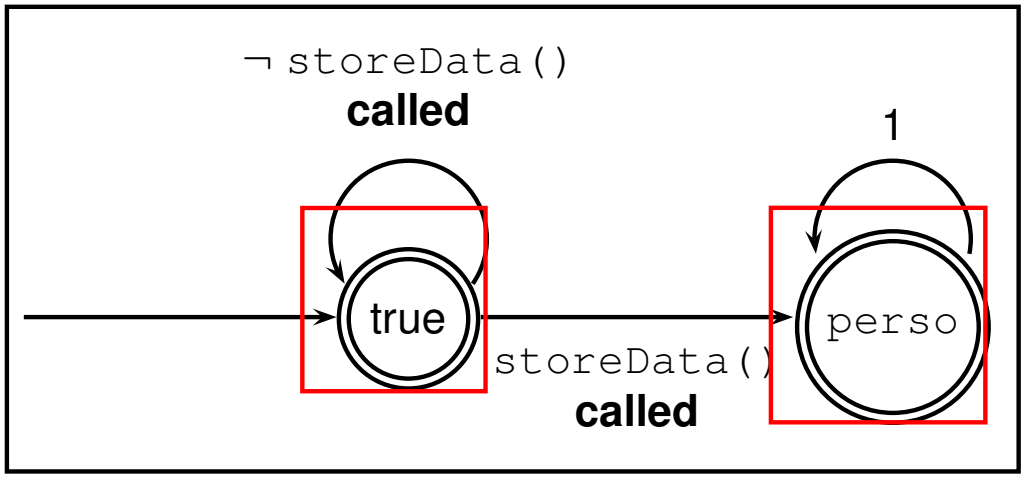
Translation of an Automata into Generic Annotations

Example of the unique personalization.

After `storeData()` **called always** `perso`



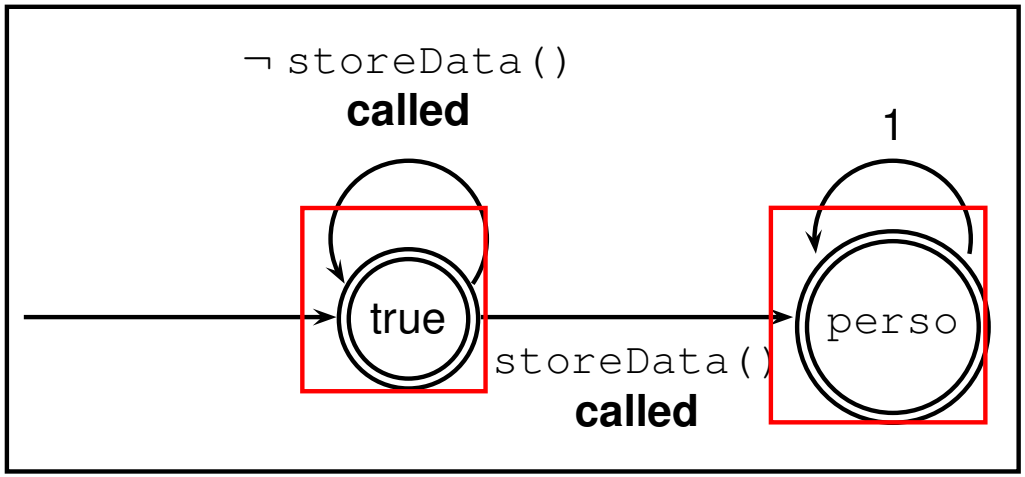
Translation of an Automata into Generic Annotations



1. Automata States
⇒ Declaration of a witness variable
2. Automata Transitions
⇒ Assignment of variable
3. State Label
⇒ Class Invariant

Variable `a`
Type: boolean
Init: false
Assignment of `a`
`storeData()`
called
Invariant
Condition:
`a == true`
Property:
`perso == true`

Translation of an Automata into Generic Annotations



1. Automata States
⇒ Declaration of a witness variable

2. Automata Transitions
⇒ Assignment of variable

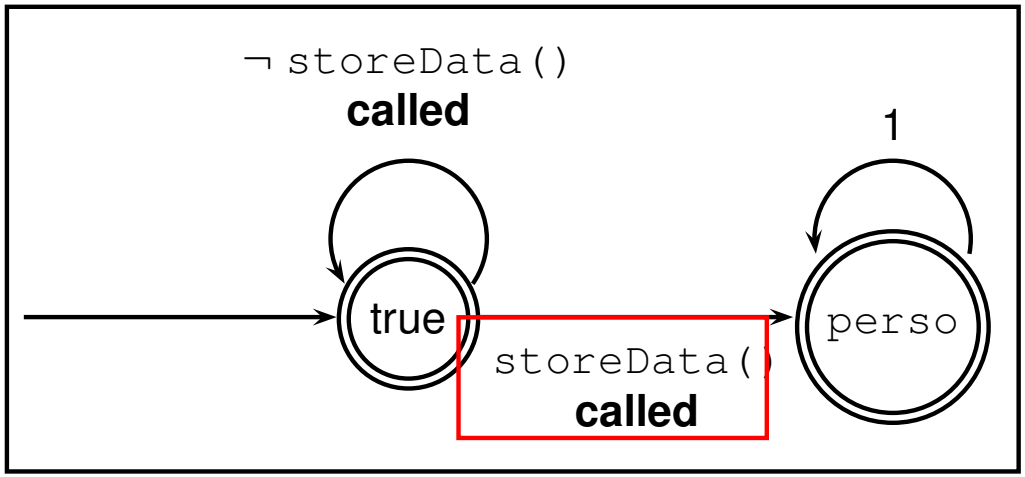
3. State Label
⇒ Class Invariant

Variable `a`
Type: boolean
Init: false

Assignment of `a`
`storeData()`
called

Invariant
Condition:
`a == true`
Property:
`perso == true`

Translation of an Automata into Generic Annotations



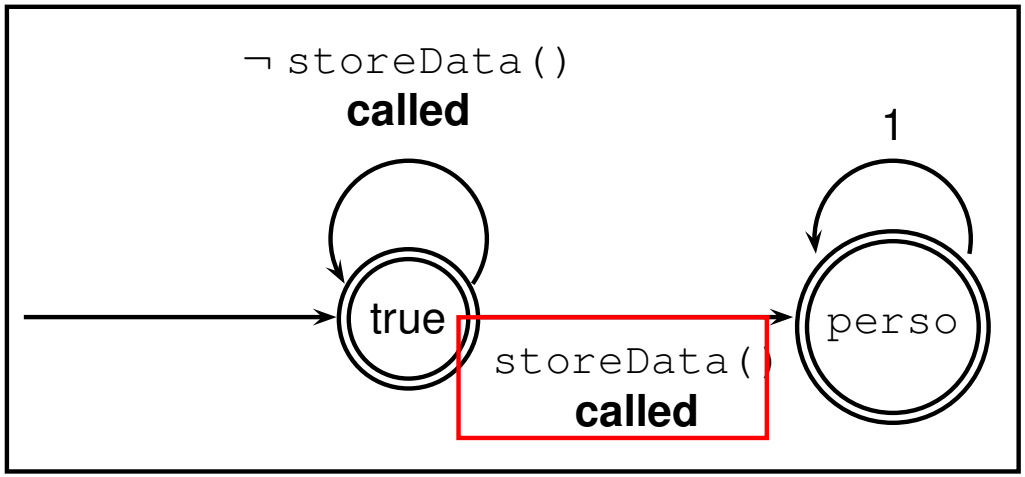
1. **Automata States**
   ⇒ Declaration of a witness variable
2. **Automata Transitions**
   ⇒ Assignment of variable
3. **State Label**
   ⇒ Class Invariant

Variable `a`
Type: boolean
Init: false

Assignment of `a`
`storeData()`
called

Invariant
Condition:
`a == true`
Property:
`perso == true`

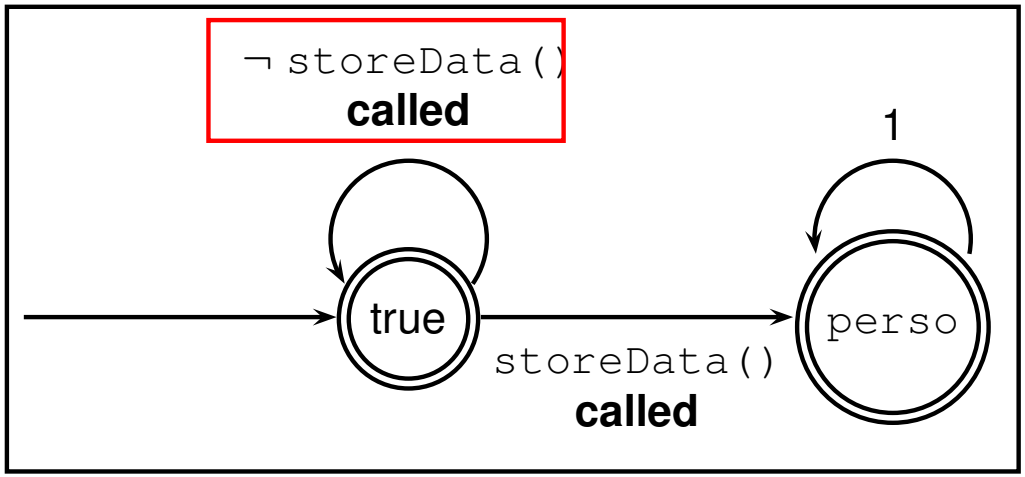
Translation of an Automata into Generic Annotations



1. **Automata States**
   ⇒ Declaration of a witness variable
2. **Automata Transitions**
   ⇒ Assignment of variable
3. **State Label**
   ⇒ Class Invariant

Variable `a`
Type: boolean
Init: false
Assignment of `a`
`storeData()`
called
Invariant
Condition:
`a == true`
Property:
`perso == true`

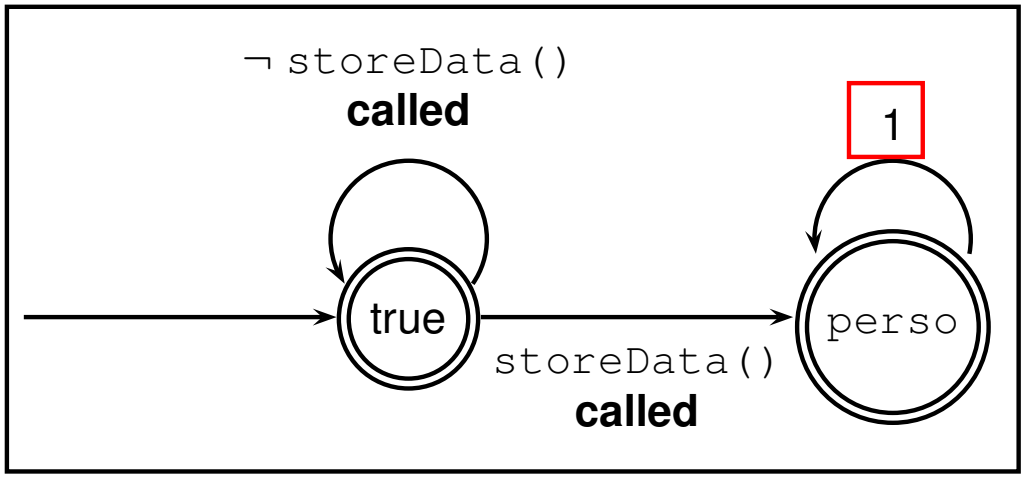
Translation of an Automata into Generic Annotations



1. **Automata States**
   ⇒ Declaration of a witness variable
2. **Automata Transitions**
   ⇒ Assignment of variable
3. **State Label**
   ⇒ Class Invariant

Variable `a`
Type: boolean
Init: false
Assignment of `a`
`storeData()`
called
Invariant
Condition:
`a == true`
Property:
`perso == true`

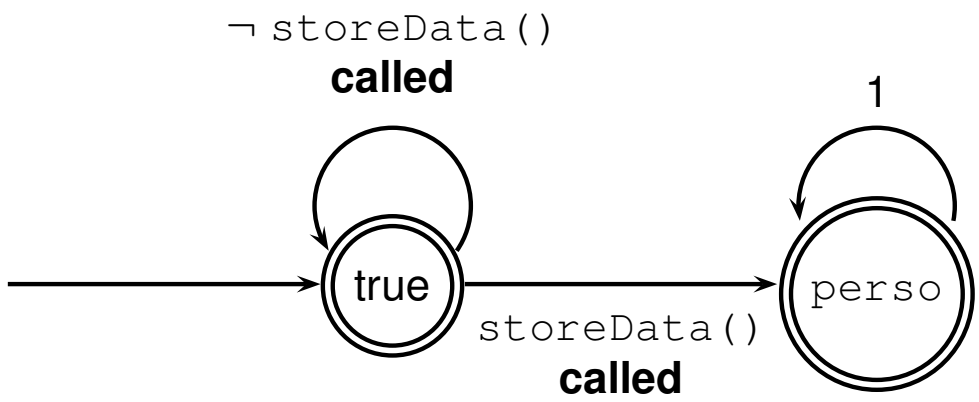
Translation of an Automata into Generic Annotations



1. **Automata States**
   ⇒ Declaration of a witness variable
2. **Automata Transitions**
   ⇒ Assignment of variable
3. **State Label**
   ⇒ Class Invariant

Variable `a`
Type: boolean
Init: false
Assignment of `a`
`storeData()`
called
Invariant
Condition:
`a == true`
Property:
`perso == true`

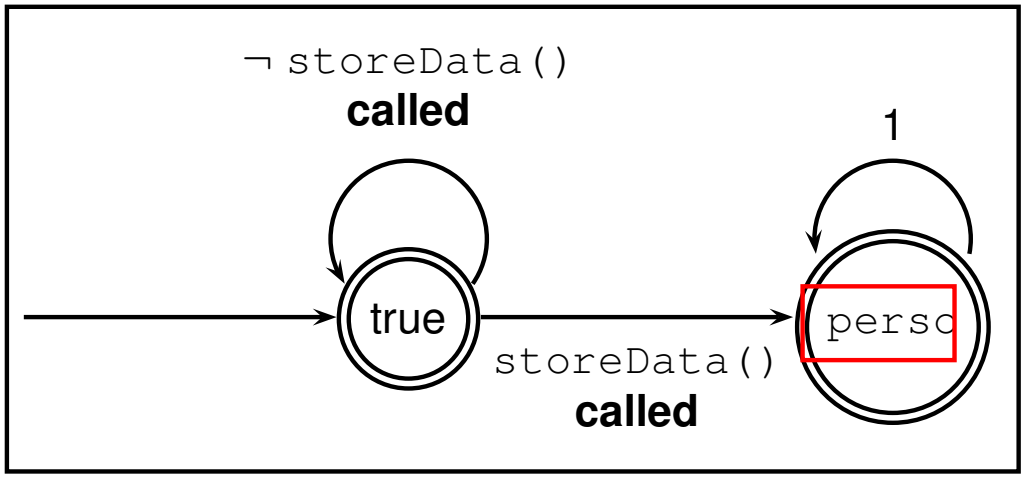
Translation of an Automata into Generic Annotations



1. **Automata States**
   ⇒ Declaration of a witness variable
2. **Automata Transitions**
   ⇒ Assignment of variable
3. **State Label**
   ⇒ Class Invariant

Variable `a`
Type: boolean
Init: false
Assignment of `a`
`storeData()`
called
Invariant
Condition:
`a == true`
Property:
`perso == true`

Translation of an Automata into Generic Annotations



1. **Automata States**
   ⇒ Declaration of a witness variable
2. **Automata Transitions**
   ⇒ Assignment of variable
3. **State Label**
   ⇒ Class Invariant

Variable `a`
Type: boolean
Init: false
Assignment of `a`
`storeData()`
called
Invariant
Condition:
`a == true`
Property:
`perso == true`

Generic Annotations

Annotations:

- Witness Variable
- Assignment of variable
- Invariant
- Variant (Liveness).
- Postcondition (Enabled/not enabled).

Properties:

- Independence to target output language (JML...).

Generic Annotations

Annotations:

- Witness Variable
- Assignment of variable
- Invariant
- Variant (Liveness).
- Postcondition (Enabled/not enabled).

Properties:

- Independence to target output language (JML...).

Generic Annotations

Annotations:

- Witness Variable
- Assignment of variable
- Invariant
- Variant (Liveness).
- Postcondition (Enabled/not enabled).

Properties:

- Independence to target output language (JML...).

Generic Annotations

Annotations:

- Witness Variable
- Assignment of variable
- Invariant
- Variant (Liveness).
- Postcondition (Enabled/not enabled).

Properties:

- Independence to target output language (JML...).

Annotations Specialization

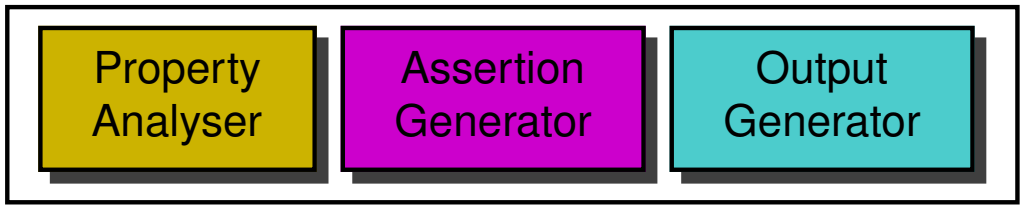
- w.r.t. Target Language.
  - Variant $\rightsquigarrow$ `constraint` for JML.
  - Variant $\rightsquigarrow$ `invariant` + `variable` for spec♯
- w.r.t the back-end tool.
  - Assignment $\rightsquigarrow$ `set` for Krakatoa.
  - Assignment $\rightsquigarrow$ `ensures` for JML-TT.
- w.r.t. user needs:
  - Witness variables $\rightsquigarrow$ `ghost` or `model`

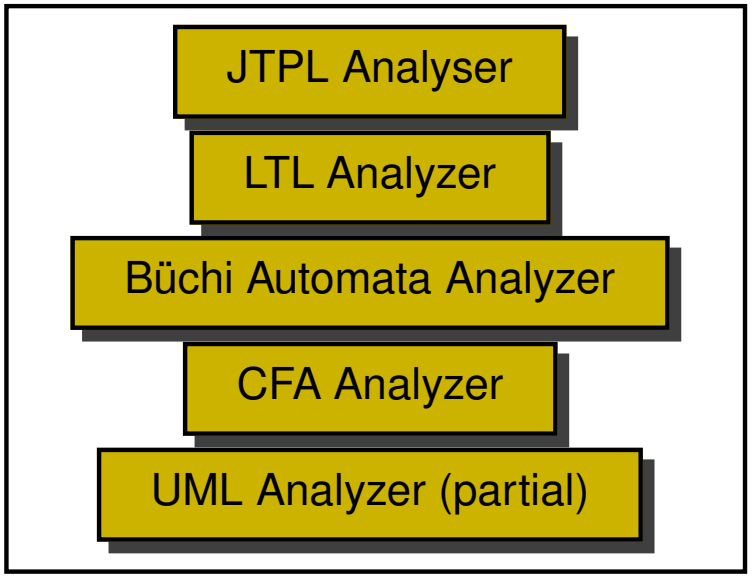
Demonstration

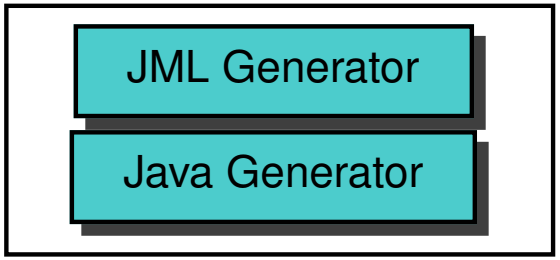
Conclusion

- **New Structure** of the JAG tool
- Based on an **internal automata structure**
- **Generic Translation** from automata to annotations (internal language)
- **Specialization of generic annotations** w.r.t. target language.
- **Easier** extension development
- **Traceability** of the generation

JAG release 0.2







Future Works

- **Extension** to other couples language/specification
  - C♯/Spec♯ (on development)
  - C/Caduceus...
  - Java/ AspectJ (on experimentation)
- Automatic **Test Generation** from internal automata (JML-TT, Tobias)
- Complete **Traceability** with Jack
- Evolutions of internal structures (Automata, Annotations)
- Eclipse Plug-in