Un programme annoté en vaut deux

Julien Groslambert **Alain Giorgetti**

JFLA 2007, 27-30 janvier 2007, Aix-les-Bains

Motivations

- Sécurité du logiciel critique

Motivations

- Sécurité du logiciel critique
  - par analyse statique (méthodes formelles)

Motivations

- Sécurité du logiciel critique
  - par analyse statique (méthodes formelles)
  - intégrée dans les pratiques usuelles de développement

Motivations

- Sécurité du logiciel critique
  - par analyse statique (méthodes formelles)
  - intégrée dans les pratiques usuelles de développement
- Comment ?
  - Insertion d'annotations formelles (assertions) dans le code source

Motivations

- Sécurité du logiciel critique
  - par analyse statique (méthodes formelles)
  - intégrée dans les pratiques usuelles de développement
- Comment ?
  - Insertion d'annotations formelles (assertions) dans le code source
  - A partir de propriétés de sécurité de haut niveau

Motivations

- Sécurité du logiciel critique
  - par analyse statique (méthodes formelles)
  - intégrée dans les pratiques usuelles de développement
- Comment ?
  - Insertion d'annotations formelles (assertions) dans le code source
  - A partir de propriétés de sécurité de haut niveau
  - Appliqué à Java annoté en JML

Motivations

- Sécurité du logiciel critique
  - par analyse statique (méthodes formelles)
  - intégrée dans les pratiques usuelles de développement
- Comment ?
  - Insertion d'annotations formelles (assertions) dans le code source
  - A partir de propriétés de sécurité de haut niveau
  - Appliqué à Java annoté en JML
  - Réduction automatique des propriétés en annotations JML (outil JAG)

Motivations

- Sécurité du logiciel critique
  - par analyse statique (méthodes formelles)
  - intégrée dans les pratiques usuelles de développement
- Comment ?
  - Insertion d'annotations formelles (assertions) dans le code source
  - A partir de propriétés de sécurité de haut niveau
  - Appliqué à Java annoté en JML
  - Réduction automatique des propriétés en annotations JML (outil JAG)

Un programme annoté en vaut deux . . .

Exemple fil rouge

Application Protocol Data Unit

- Classe `javacard.framework.APDU` de l'API Java Card 2.1

Exemple fil rouge

Application Protocol Data Unit

- Classe `javacard.framework.APDU` de l'API Java Card 2.1
- Protocole d'échange de données entre carte à puce et terminal

Exemple fil rouge

Application Protocol Data Unit

- Classe `javacard.framework.APDU` de l'API Java Card 2.1
- Protocole d'échange de données entre carte à puce et terminal
- Buffer de données

Exemple fil rouge

Application Protocol Data Unit

- Classe `javacard.framework.APDU` de l'API Java Card 2.1
- Protocole d'échange de données entre carte à puce et terminal
- Buffer de données
- Séquences de lectures et écritures

Exemple fil rouge

Application Protocol Data Unit

- Classe `javacard.framework.APDU` de l'API Java Card 2.1
- Protocole d'échange de données entre carte à puce et terminal
- Buffer de données
- Séquences de lectures et écritures
- Documentation (javadoc) de Sun Microsystems

http://java.sun.com/products/javacard/htmldoc/javacard/framework/APDU.html

Présentation de JML

JML : Java Modeling Language (Gary Leavens, Iowa)

Présentation de JML

JML : Java Modeling Language (Gary Leavens, Iowa)

- Commentaires spéciaux

Présentation de JML

JML : Java Modeling Language (Gary Leavens, Iowa)

- Commentaires spéciaux
- Langage logique : sous-ensemble *pur* de Java + extensions logiques \forall, mémoire \old . . .

Présentation de JML

JML : Java Modeling Language (Gary Leavens, Iowa)

- Commentaires spéciaux
- Langage logique : sous-ensemble *pur* de Java + extensions logiques \forall, mémoire \old . . .
- Traitement possible à différents niveaux

Présentation de JML

JML : Java Modeling Language (Gary Leavens, Iowa)

- Commentaires spéciaux
- Langage logique : sous-ensemble *pur* de Java + extensions logiques \forall, mémoire \old . . .
- Traitement possible à différents niveaux
  - documentation `jmldoc`
  - génération de code défensif `jmlc`
  - génération de pilotes de test `jmlunit`
  - vérification automatique `ESC/Java`
  - vérification interactive `LOOP, JIVE, Jack, Krakatoa`

Présentation de JML

JML : Java Modeling Language (Gary Leavens, Iowa)

- Commentaires spéciaux
- Langage logique : sous-ensemble *pur* de Java + extensions logiques \forall, mémoire \old . . .
- Traitement possible à différents niveaux
  - documentation `jmldoc`
  - génération de code défensif `jmlc`
  - génération de pilotes de test `jmlunit`
  - vérification automatique `ESC/Java`
  - vérification interactive `LOOP, JIVE, Jack, Krakatoa`

APDU annotée en JML

```
package javacard.framework;
public class APDU {
  //@ invariant getBuffer().length >= 37;
  //@ ghost int bufferPosition = 0;

  /*@ pure @*/ byte[] getBuffer() { ... }

  /*@ normal_behavior
    @ requires bufferPosition == 0;
    @ ensures bufferPosition == bOff - 1
    @   || bufferPosition == getBuffer().length
    @*/
  short receiveBytes(short bOff) {
    ...
    le = getBuffer().length;
    /*@ set bufferPosition =
      @   (bOff > le ? le : bOff) - 1;
      @*/
    ...
  }
}
```

Pourquoi aux JFLA ?

- Besoins
  - Valider des méthodes de vérification formelle
  - Développer rapidement des prototypes

Pourquoi aux JFLA ?

- Besoins
  - Valider des méthodes de vérification formelle
  - Développer rapidement des prototypes
- Technologie (parsing, transformations) pas à la hauteur
- Capacité à expérimenter des innovations (ASF-SDF, TOM, . . .)

Pourquoi aux JFLA ?

- Besoins
  - Valider des méthodes de vérification formelle
  - Développer rapidement des prototypes
- Technologie (parsing, transformations) pas à la hauteur
- Capacité à expérimenter des innovations (ASF-SDF, TOM, . . .)
- Collaborations gagnant-gagnant !
  - Vérification statique comme champ d'application des techniques de programmation fonctionnelle et de transformation
  - Donne un intérêt théorique à l'activité de développement

Pourquoi aux JFLA ?

- Besoins
  - Valider des méthodes de vérification formelle
  - Développer rapidement des prototypes
- Technologie (parsing, transformations) pas à la hauteur
- Capacité à expérimenter des innovations (ASF-SDF, TOM, . . .)
- Collaborations gagnant-gagnant !
  - Vérification statique comme champ d'application des techniques de programmation fonctionnelle et de transformation
  - Donne un intérêt théorique à l'activité de développement
- Approche multi-langages (générique)
- Modèles (quasi-)fonctionnels

Plan de l'exposé

Plan de l'exposé

Schémas temporels

- Java Temporal Pattern Language (JTPL)
- Origines
  - Specification Patterns (Dwyers et. al., 98)
  - Adaptation à JML (Huisman, Trentelman, AMAST'02)
  - Traduction des vivacités en JML (Bellegarde et. al., 04)

Schémas temporels

- Java Temporal Pattern Language (JTPL)
- Origines
  - Specification Patterns (Dwyers et. al., 98)
  - Adaptation à JML (Huisman, Trentelman, AMAST'02)
  - Traduction des vivacités en JML (Bellegarde et. al., 04)
- Spécificités
  - Logique linéaire (sémantique de trace)
  - Schémas plus intuitifs que la LTL
  - Logique mixte (états/événements)
  - Evénements d'appel et de sortie de méthode
  - Terminaison normale ou exceptionnelle
  - Se greffe sur JML
- Ici, zoom sur terminaison normale et lien avec LTL

JTPL

Logique temporelle mixte

Propriétés d'états

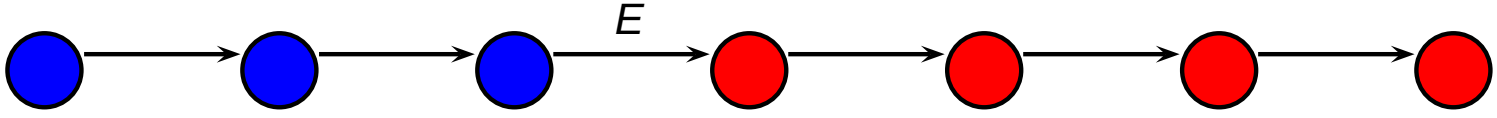
- Prédicats JML
- m **enabled**
- m **not enabled**

Evénements

- m **called**
- m **normal**
- m **exceptional**
- m **terminates**

Opérateurs temporels combinant événements *E* et propriétés d'états

after *E* *C*

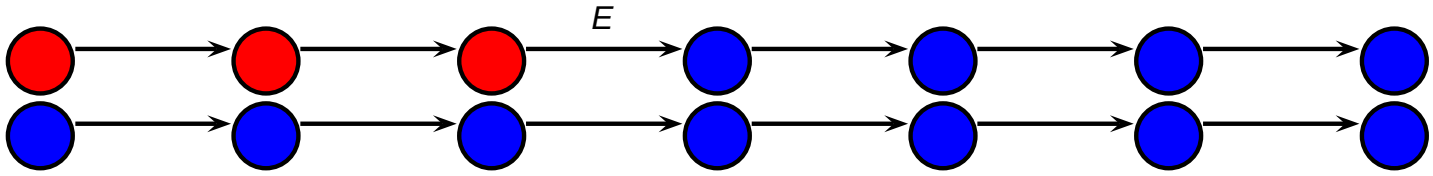


JTPL

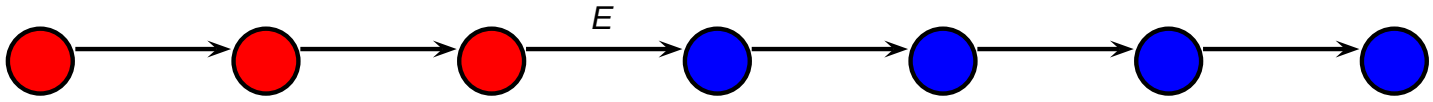
Opérateurs temporels mixtes

Evénement E, propriété de trace T

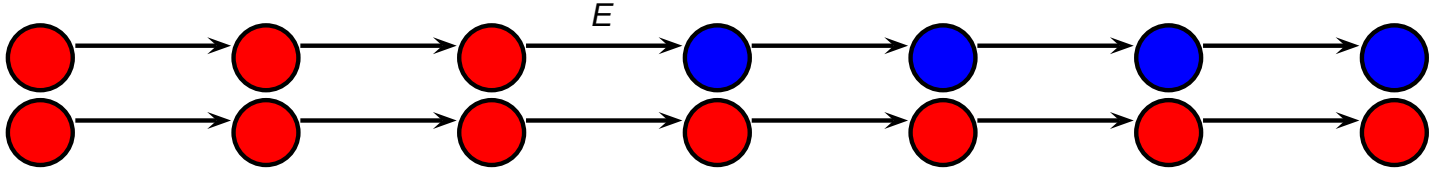
- **before** E T



- T **until** E



- T **unless** E



- **always** P, **eventually** P, $T \wedge T'$, $T \vee T'$
  Traces T, T', propriété d'états P

JTPL

Exemple pour l'APDU

Extrait de

http://java.sun.com/products/javacard/htmldoc/javacard/framework/APDU.html

setIncomingAndReceive

`public short setIncomingAndReceive() throws APDUException`

Throws: APDUException with the following reason codes:

`APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` already invoked . . .

JTPL

Exemple pour l'APDU

Extrait de

http://java.sun.com/products/javacard/htmldoc/javacard/framework/APDU.html

setIncomingAndReceive

`public short setIncomingAndReceive() throws APDUException`

Throws: APDUException with the following reason codes:

`APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` already invoked . . .

after setIncomingAndReceive() normal
always setIncomingAndReceive() **not enabled**

JTPL → LTL

Traduction des événements

- Soit *e* un événement élémentaire

JTPL $\longrightarrow$ LTL

Traduction des événements

- Soit e un événement élémentaire
- Opérateur $I(e)$
  - Traces commençant par une transition selon e

$$s\, s'\sigma \models I(e) \quad \text{ssi} \quad s \xrightarrow{e} s'$$

JTPL $\longrightarrow$ LTL

Traduction des événements

- Soit e un événement élémentaire
- Opérateur $I(e)$
  - Traces commençant par une transition selon e

$$s\ s'\sigma \models I(e) \quad \text{ssi} \quad s \xrightarrow{e} s'$$

  - Si déterministe (s' unique), se réduit à la propriété $\mathit{activable}(e)$ de l'état s

$$s\ \sigma' \models \mathit{activable}(e) \quad \text{ssi} \quad \exists s' \,.\, s \xrightarrow{e} s'$$

JTPL $\longrightarrow$ LTL

Traduction des événements

- Soit e un événement élémentaire
- Opérateur $I(e)$
  - Traces commençant par une transition selon e

$$s\ s'\sigma \models I(e) \quad \text{ssi} \quad s \xrightarrow{e} s'$$

  - Si déterministe (s' unique), se réduit à la propriété $\mathit{activable}(e)$ de l'état s

$$s\ \sigma' \models \mathit{activable}(e) \quad \text{ssi} \quad \exists s' \,.\, s \xrightarrow{e} s'$$

  - JTPL se traduit en LTL avec $\mathit{activable}(e)$ dans les propriétés d'états

JTPL $\longrightarrow$ LTL

Traduction des événements

- Soit e un événement élémentaire
- Opérateur $I(e)$
  - Traces commençant par une transition selon e

$$s\ s'\sigma \models I(e) \quad \text{ssi} \quad s \xrightarrow{e} s'$$

  - Si déterministe (s' unique), se réduit à la propriété $\mathit{activable}(e)$ de l'état s

$$s\ \sigma' \models \mathit{activable}(e) \quad \text{ssi} \quad \exists s' \,.\, s \xrightarrow{e} s'$$

  - JTPL se traduit en LTL avec $\mathit{activable}(e)$ dans les propriétés d'états
  - Sinon, traduction en LTL+I

JTPL $\longrightarrow$ LTL

Traduction des événements

- Soit e un événement élémentaire
- Opérateur $I(e)$
  - Traces commençant par une transition selon e

$$s\ s'\sigma \models I(e) \quad \text{ssi} \quad s \xrightarrow{e} s'$$

  - Si déterministe (s' unique), se réduit à la propriété $\mathit{activable}(e)$ de l'état s

$$s\ \sigma' \models \mathit{activable}(e) \quad \text{ssi} \quad \exists s' \,.\, s \xrightarrow{e} s'$$

  - JTPL se traduit en LTL avec $\mathit{activable}(e)$ dans les propriétés d'états
  - Sinon, traduction en LTL+I
  - Ensemble (disjonction) E d'événements

$$s\ s'\sigma \models I(E) \quad \text{ssi} \quad \exists\, e \in E \,.\, s\ s'\sigma \models I(e)$$

JTPL

Traduction en LTL+I

Nature	ψ en JTPL	$t(\psi)$ en LTL
Trace (T, T', C)	$T \wedge T'$	$t(T) \wedge t(T')$
	$T \vee T'$	$t(T) \vee t(T')$
	always P	G P
	eventually P	F P
Mixte (C)	after E C	G $(r(E) \Rightarrow t(C))$
	always P until E	P U $r(E)$
	eventually P until E	$\neg(\neg P$ U $r(E))$
	before E T	$t(T$ until $E) \vee$ G$\neg\, I(E)$
	T unless E	$t(T$ until $E) \vee ((\mathsf{G}\neg\, I(E)) \wedge t(T))$

E disjonction d'événements, P propriété d'état, $r(E) = \mathsf{X}^{-} I(E)$

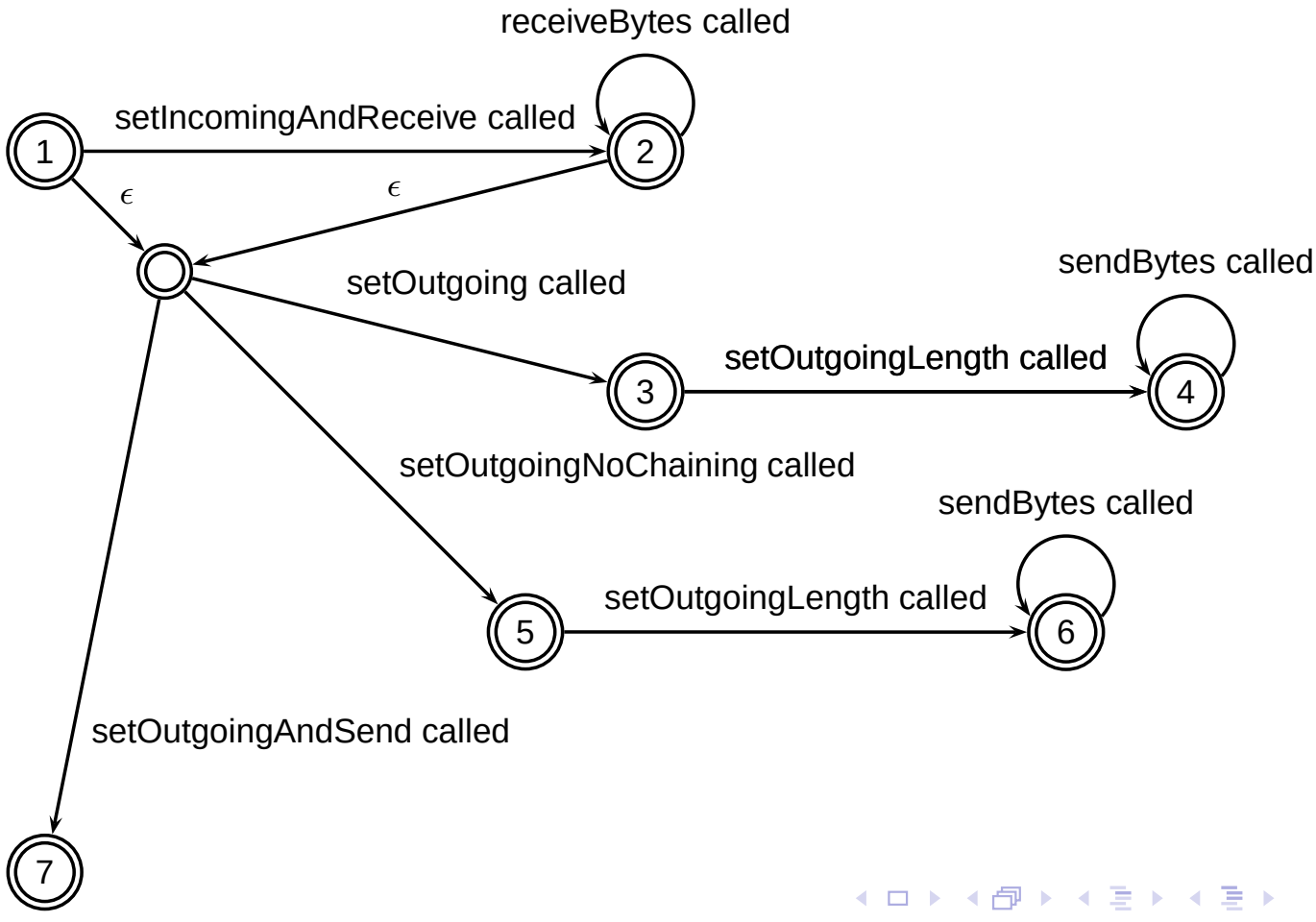
Plan de l'exposé

Automates étiquetés

- Ensemble fini S d'états décorés par une propriété d'état
  - Prédicats JML
  - m **enabled**
  - m **not enabled**
- Transitions décorées par ϵ ou par des événements
  - m **called**
  - m **normal**
  - m **exceptional**
  - m **terminates**
- Ensemble d'états d'acceptation $S_a \subseteq S$
- Sémantique de traces infinies (Buchi)

Exemple d'automate

APDU



Traduction d'un automate en annotations

- Etats de l'automate $\longrightarrow$ variables témoins
  - Si l'automate est déterministe, une variable témoin de type entier
  - Si l'automate est indéterministe, *n* variables témoin de type booléen
  - Mise à jour selon les transitions de l'automate

Traduction d'un automate en annotations

- Etats de l'automate $\longrightarrow$ variables témoins
  - Si l'automate est déterministe, une variable témoin de type entier
  - Si l'automate est indéterministe, *n* variables témoin de type booléen
  - Mise à jour selon les transitions de l'automate
- Etiquettes d'états
  - Chaque prédicat devient un invariant
  - Chaque *m* enabled / *m* not enabled devient une post-condition

Plan de l'exposé

Outil JAG

JML Annotation Generator

- Générateur de spécifications JML

Outil JAG

JML Annotation Generator

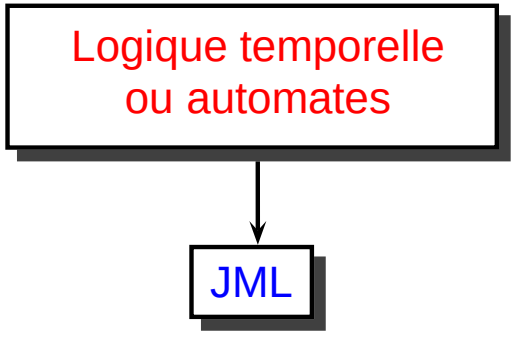
- Générateur de spécifications JML
- Principe
  - Formaliser des **propriétés de sécurité** à partir du cahier des charges

Logique temporelle ou automates

Outil JAG

JML Annotation Generator

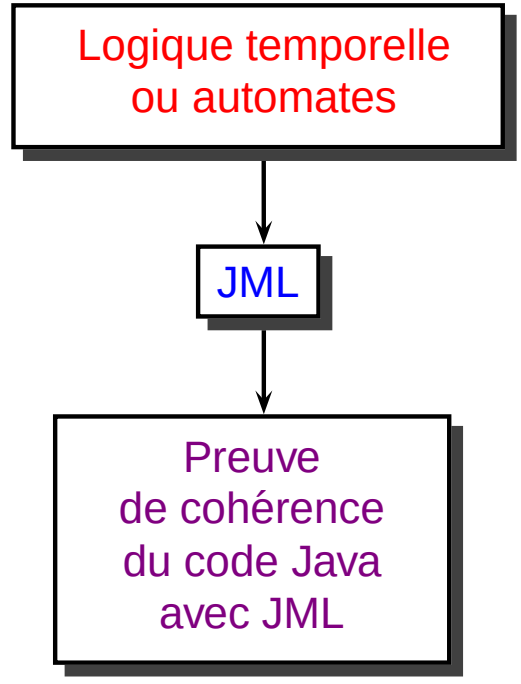
- Générateur de spécifications JML
- Principe
  - Formaliser des **propriétés de sécurité** à partir du cahier des charges
  - Traduire les propriétés dans un **format permettant la vérification formelle** sur le code Java



Outil JAG

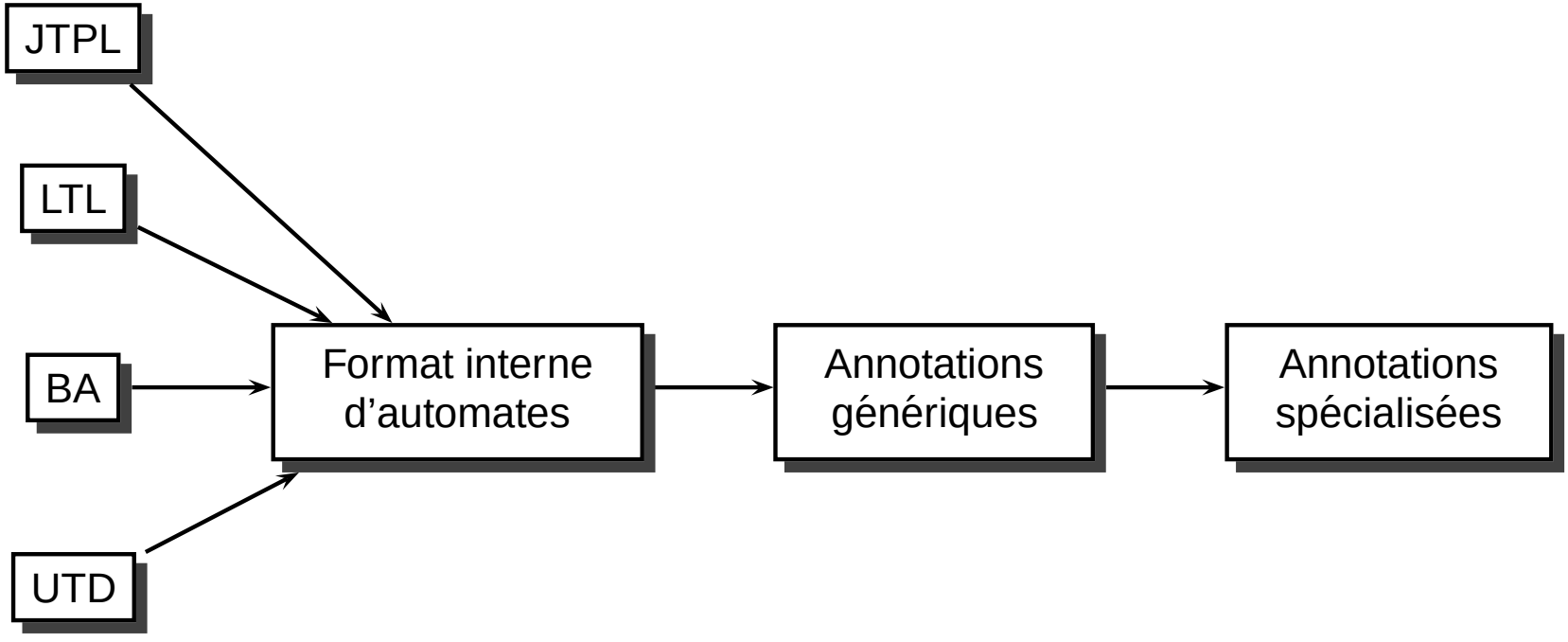
JML Annotation Generator

- Générateur de spécifications JML
- Principe
  - Formaliser des **propriétés de sécurité** à partir du cahier des charges
  - Traduire les propriétés dans un **format permettant la vérification formelle** sur le code Java
  - Vérifier les propriétés sur le code Java



Outil JAG

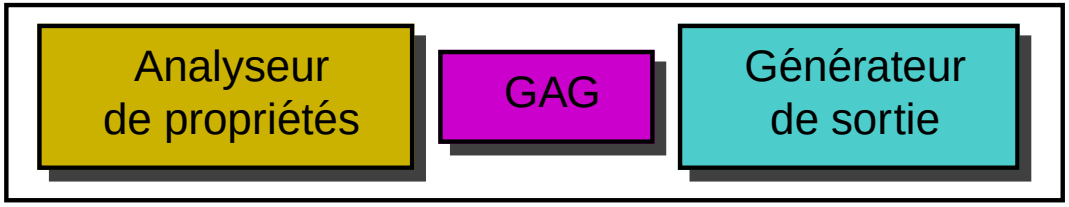
Formats d'entrée



UTD : State Machine Diagram d'UML 2.0

Architecture interne de JAG

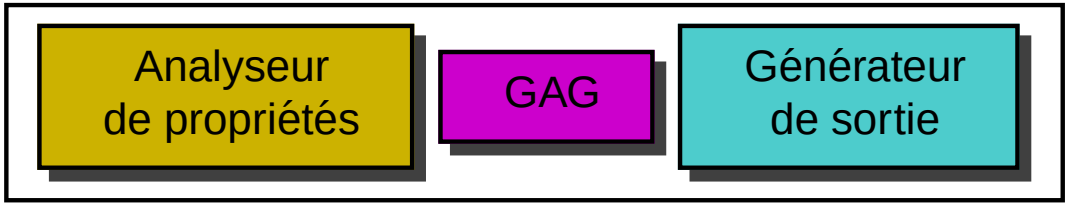
- Trois modules



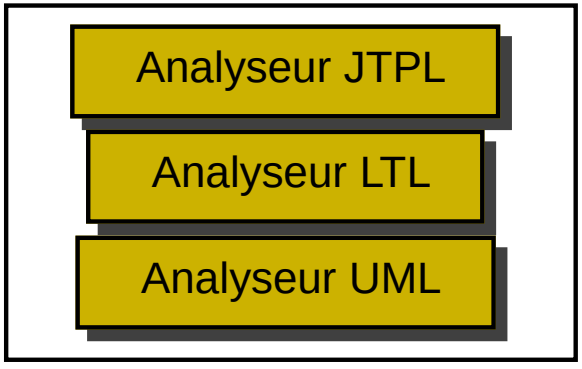
- GAG : Générateur d'Annotations Générique

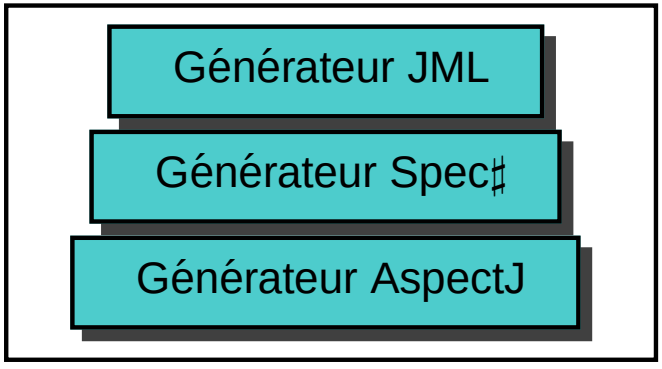
Architecture interne de JAG

- Trois modules

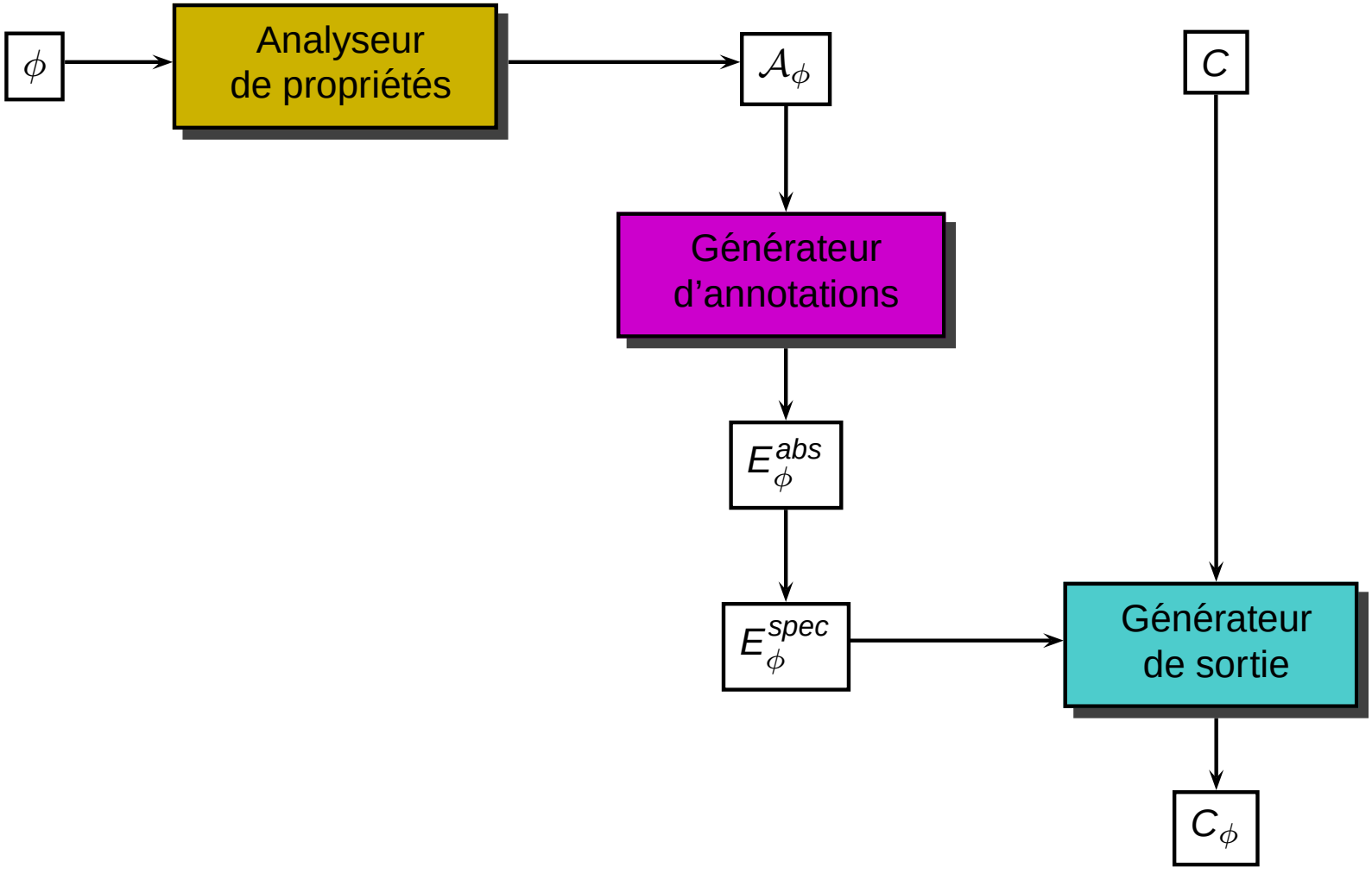


- GAG : Générateur d'Annotations Générique
- Instantiation par des analyseurs de propriétés et des générateurs de sortie particuliers

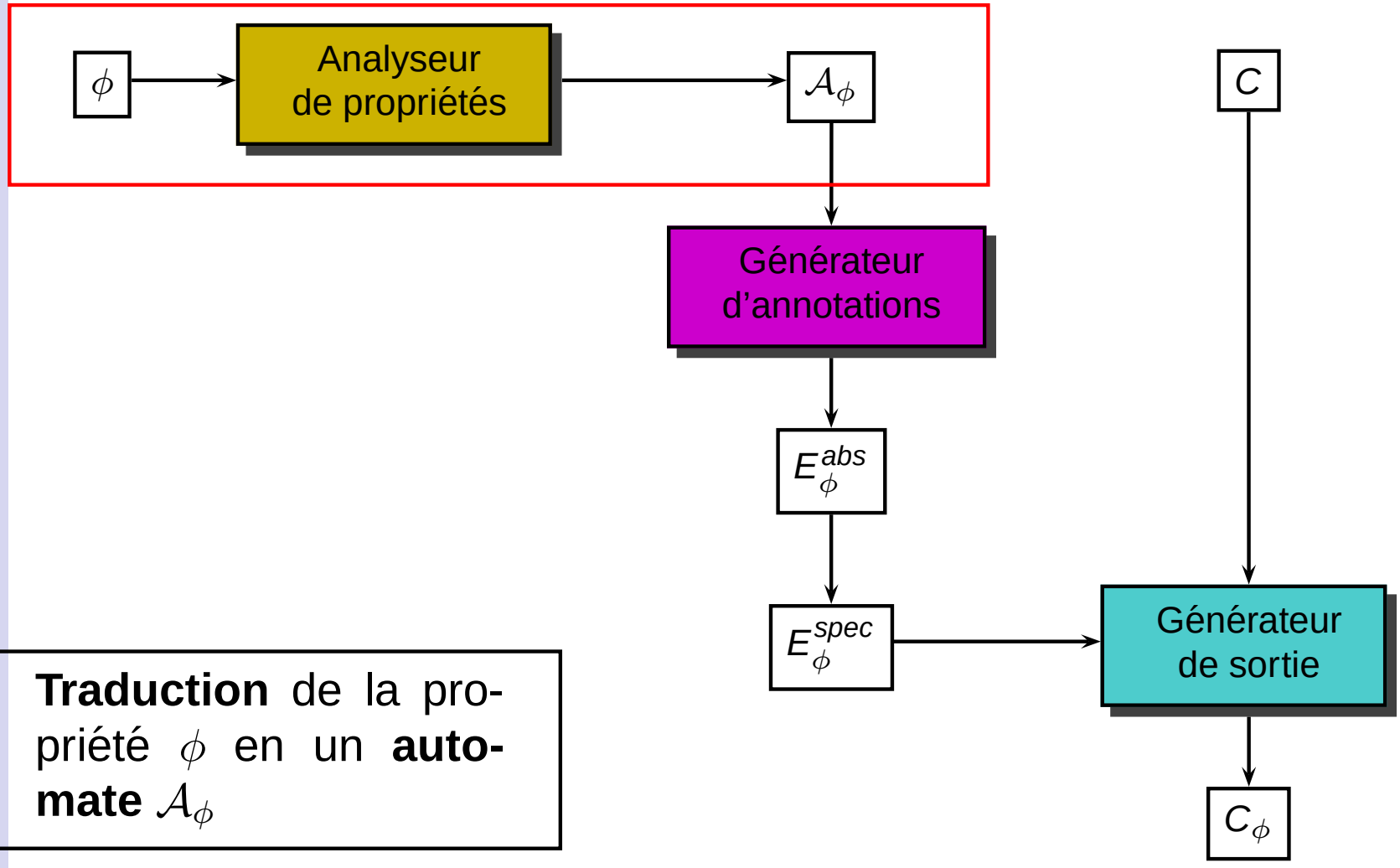




Flot de données

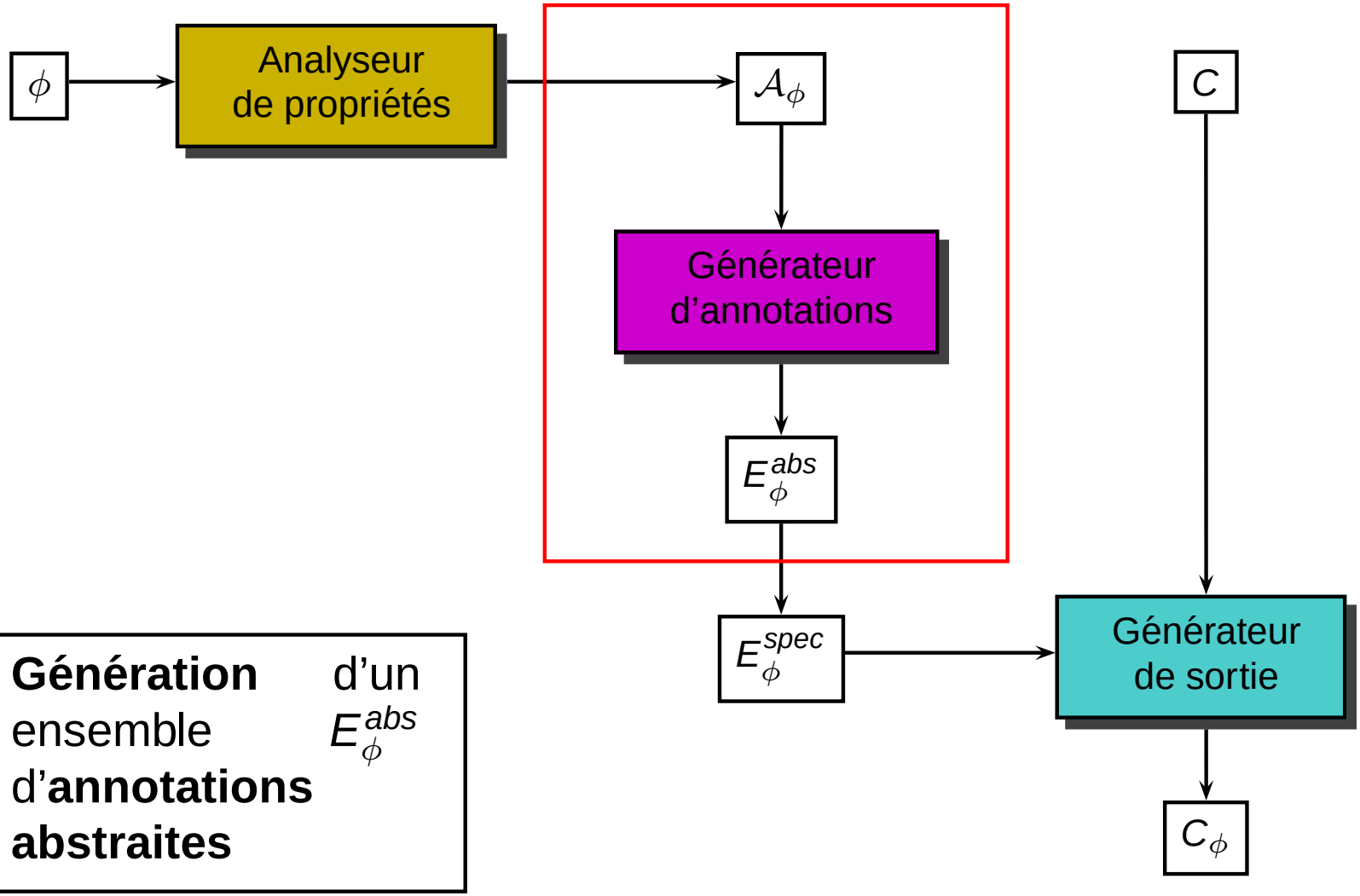


Flot de données



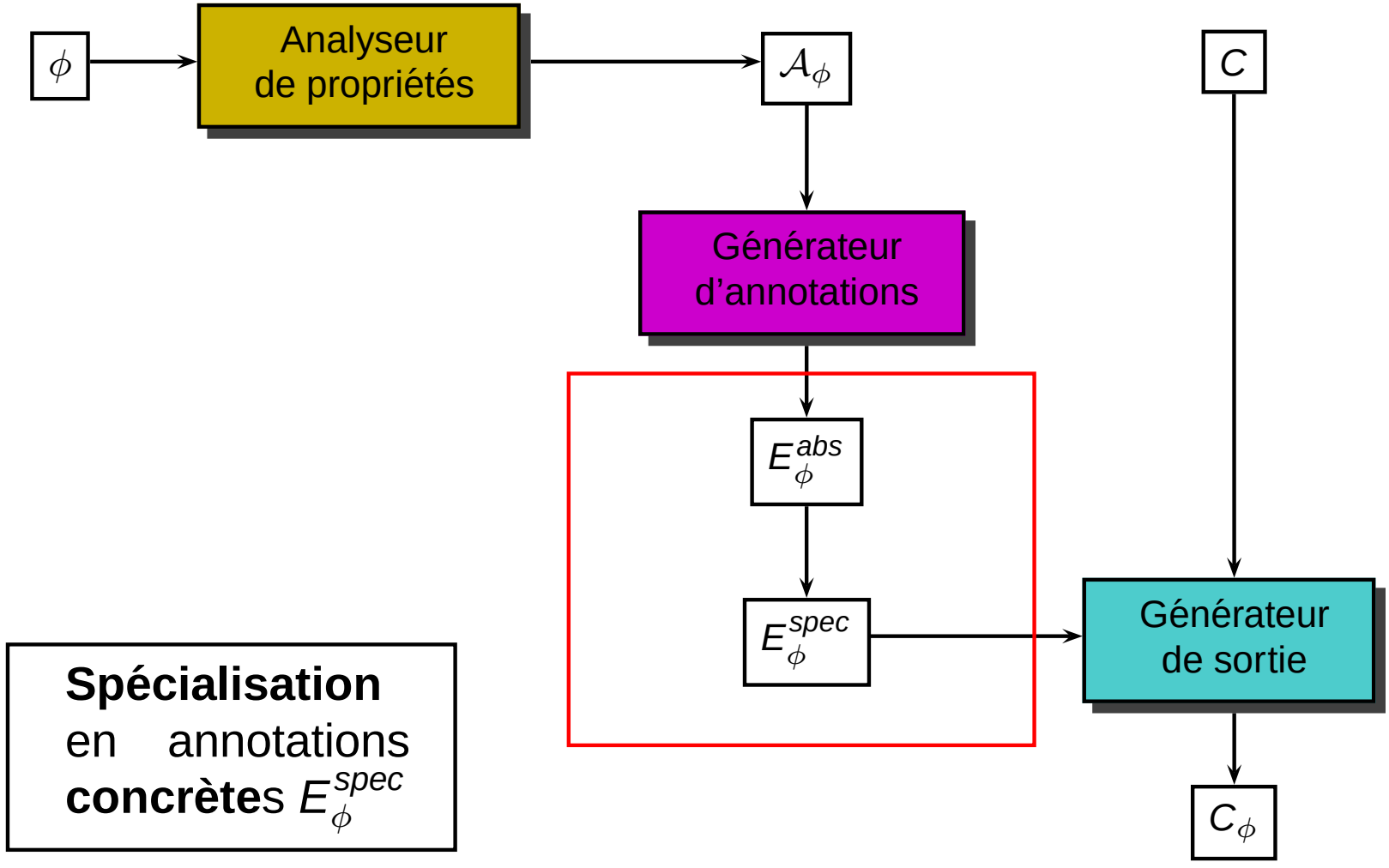
Traduction de la propriété ϕ en un **automate** $\mathcal{A}_\phi$

Flot de données



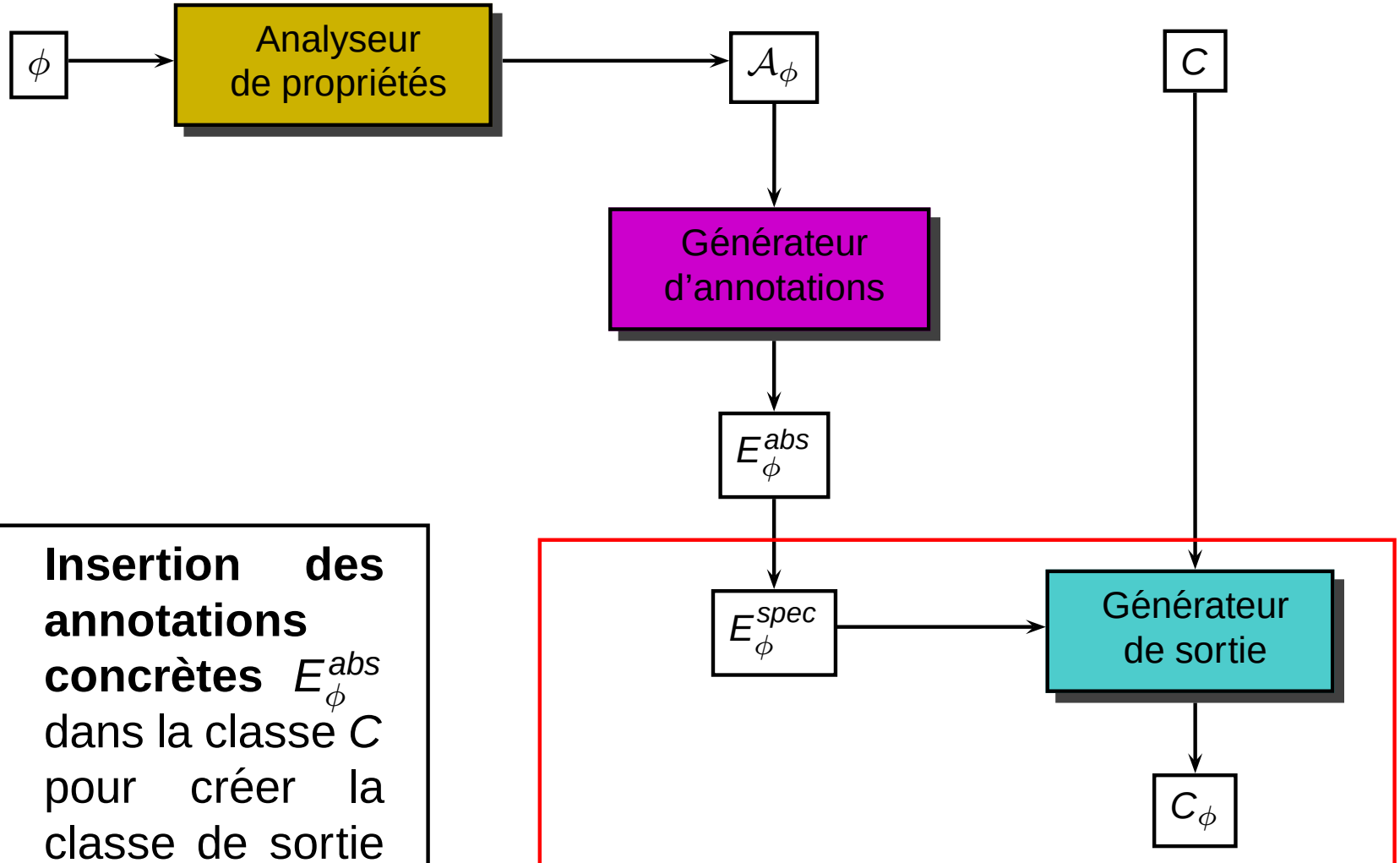
Génération d'un ensemble E^{abs}_ϕ d'**annotations abstraites**

Flot de données



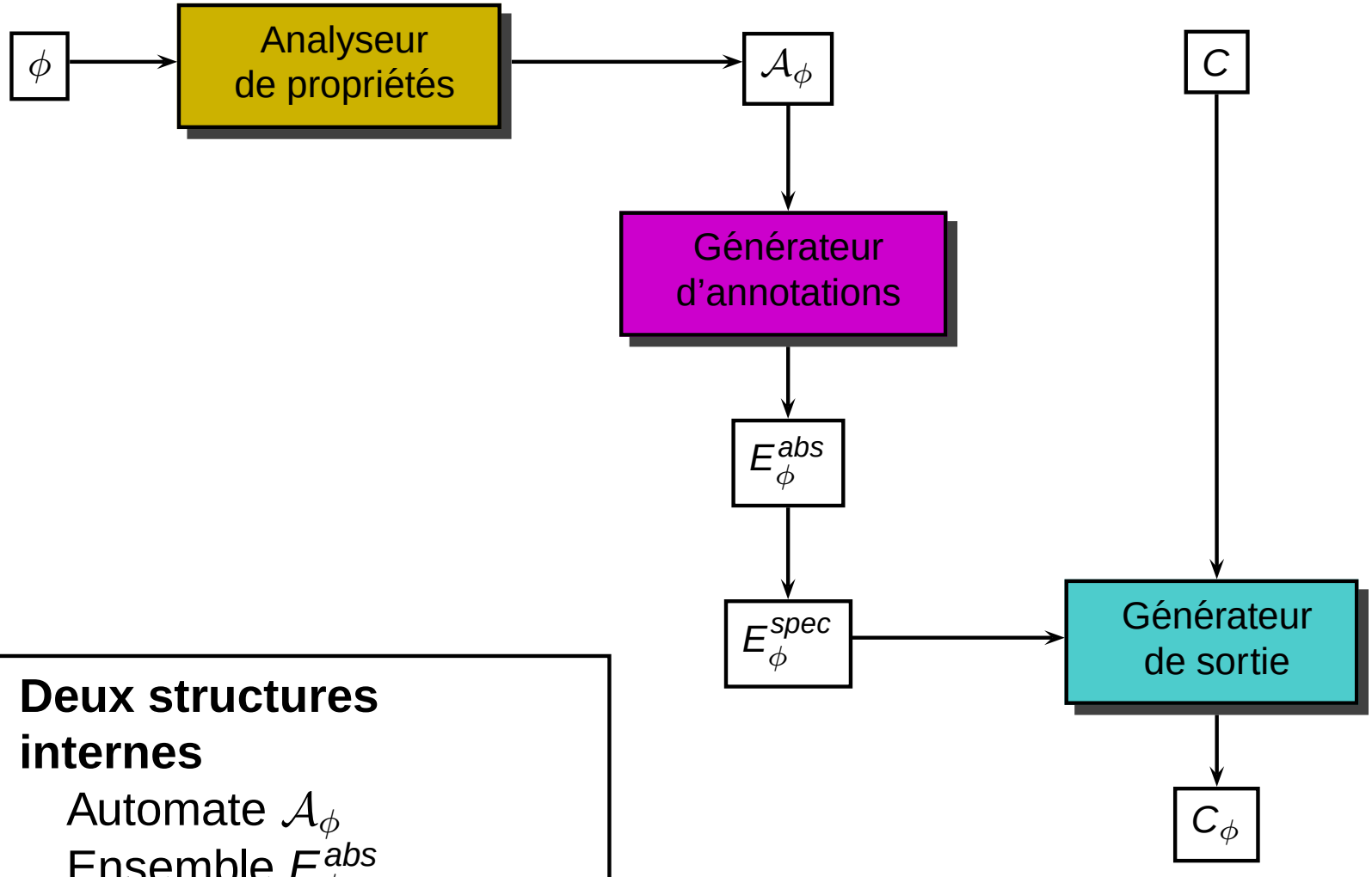
Spécialisation en annotations **concrètes** E_ϕ^{spec}

Flot de données



Insertion des annotations concrètes E_ϕ^{abs} dans la classe C pour créer la classe de sortie C_ϕ

Flot de données



Deux structures internes

- Automate $\mathcal{A}_\phi$
- Ensemble E_ϕ^{abs} d'annotations abstraites

Plan de l'exposé

Démonstration

- JAG 0.2
  - `http://lifc.univ-fcomte.fr/~jag`
  - Apdu.java
  - TransactionSystem.java

Démonstration

- JAG 0.2
  - `http://lifc.univ-fcomte.fr/~jag`
  - Apdu.java
  - TransactionSystem.java
- Traitement du code produit
  - Jack
  - Krakatoa
  - JMLTools

Plan de l'exposé

Conclusion

- Propriétés temporelles pour Java
  - Intégrées dans un processus de développement familier (commentaires)
  - Permet d'utiliser de nombreux outils de vérification
  - Multi-langages (générique)

Conclusion

- Propriétés temporelles pour Java
  - Intégrées dans un processus de développement familier (commentaires)
  - Permet d'utiliser de nombreux outils de vérification
  - Multi-langages (générique)
- Difficultés
  - Technologiques
    - Outils de développement mal adaptés (sortir entrée enrichie, tracer les erreurs)
    - Fragilité de la chaîne d'outils de vérification

Conclusion

- Propriétés temporelles pour Java
  - Intégrées dans un processus de développement familier (commentaires)
  - Permet d'utiliser de nombreux outils de vérification
  - Multi-langages (générique)
- Difficultés
  - Technologiques
    - Outils de développement mal adaptés (sortir entrée enrichie, tracer les erreurs)
    - Fragilité de la chaîne d'outils de vérification
  - Scientifiques
    - Variabilité des langages (JML, Java)
    - Sans sémantique formelle

Perspectives

- Autres sorties pour JAG (partenariats) : C♯/Spec♯ (en cours), C/Caduceus, Java/AspectJ, . . .
- Plug-in Eclipse (en cours)
- Couplage outils JAG et Krakatoa
- Utilisation de méthodes de transformation avancées
- Transfert aux langages fonctionnels
- Autres applications des annotations

Questions ?