Détection d'erreurs

F. Dadeau[1] **A. Giorgetti**[2]

[1]LSR-IMAG

[2]Université de Franche-Comté

Janvier 2007 - Réunion finale Geccoo - Orsay

Détection d'erreurs

Détection d'erreurs

- Conformité code/spécification (Java/JML)
- Correction du modèle JML

Détection d'erreurs

- Conformité code/spécification (Java/JML)
  Comment la vérifier ?
  - Vérification à la volée (JMLtools RAC)
  - Analyse statique renforcée (ProVal, Everest)
  - Preuves plus automatiques (ProVal, CASSIS)
  - Tests avec JML comme verdict (TFC)
- Correction du modèle JML

Détection d'erreurs

- Conformité code/spécification (Java/JML)
  Comment la vérifier ?
  - Vérification à la volée (JMLtools RAC)
  - Analyse statique renforcée (ProVal, Everest)
  - Preuves plus automatiques (ProVal, CASSIS)
  - Tests avec JML comme verdict (TFC)
- Correction du modèle JML

Détection d'erreurs

- Conformité code/spécification (Java/JML)
  Comment la vérifier ?
  - Vérification à la volée (JMLtools RAC)
  - Analyse statique renforcée (ProVal, Everest)
  - Preuves plus automatiques (ProVal, CASSIS)
  - Tests avec JML comme verdict (TFC)
- Correction du modèle JML

Détection d'erreurs

- Conformité code/spécification (Java/JML)
  Comment la vérifier ?
  - Vérification à la volée (JMLtools RAC)
  - Analyse statique renforcée (ProVal, Everest)
  - Preuves plus automatiques (ProVal, CASSIS)
  - Tests avec JML comme verdict (TFC)
- Correction du modèle JML

Détection d'erreurs

- Conformité code/spécification (Java/JML)
- Correction du modèle JML

Détection d'erreurs

- Conformité code/spécification (Java/JML)
- Correction du modèle JML
  Comment la vérifier ?
  - Propagation d'annotations JML (Everest)
  - Recherche d'erreurs par animation contrainte du modèle JML (TFC)
  - Génération d'obligations de preuve (TFC)
  - Garantie pour JML généré à partir de spécifications de plus haut niveau (TFC)

Détection d'erreurs

- Conformité code/spécification (Java/JML)
- Correction du modèle JML
  Comment la vérifier ?
  - Propagation d'annotations JML (Everest)
  - Recherche d'erreurs par animation contrainte du modèle JML (TFC)
  - Génération d'obligations de preuve (TFC)
  - Garantie pour JML généré à partir de spécifications de plus haut niveau (TFC)

Détection d'erreurs

- Conformité code/spécification (Java/JML)
- Correction du modèle JML
  Comment la vérifier ?
  - Propagation d'annotations JML (Everest)
  - Recherche d'erreurs par animation contrainte du modèle JML (TFC)
  - Génération d'obligations de preuve (TFC)
  - Garantie pour JML généré à partir de spécifications de plus haut niveau (TFC)

Détection d'erreurs

- Conformité code/spécification (Java/JML)
- Correction du modèle JML
  Comment la vérifier ?
  - Propagation d'annotations JML (Everest)
  - Recherche d'erreurs par animation contrainte du modèle JML (TFC)
  - Génération d'obligations de preuve (TFC)
  - Garantie pour JML généré à partir de spécifications de plus haut niveau (TFC)

Détection d'erreurs

- Conformité code/spécification (Java/JML)
- Correction du modèle JML
  Comment la vérifier ?
  - Propagation d'annotations JML (Everest)
  - Recherche d'erreurs par animation contrainte du modèle JML (TFC)
  - Génération d'obligations de preuve (TFC)
  - Garantie pour JML généré à partir de spécifications de plus haut niveau (TFC)

Principales difficultés de la détection d'erreurs

- Formaliser la sécurité
- Propager les contraintes de sécurité jusqu'au code
- Résoudre les conditions de conformité entre modèles
- Faire remonter les erreurs

Outils

JML-Testing-Tools

- Animateur symbolique
  - Animation symbolique du modèle JML en PLC
  - A partir des comportements extraits des annotations JML des méthodes
- Générateur automatique de tests “aux limites”
  - Extraction des cibles de test à partir du modèle (activation des comportements)
  - Calcul d’une séquence d’exécution qui mène à la cible
  - Concrétisé en cas de test Java exécutable
- Transfert de l’expérience BZ-Testing-Tools
- Ne considère pas le corps des méthodes en Java, seulement les annotations JML

Outils

JML-Testing-Tools

- Animateur symbolique
  - Animation symbolique du modèle JML en PLC
  - A partir des comportements extraits des annotations JML des méthodes
- Générateur automatique de tests “aux limites”
  - Extraction des cibles de test à partir du modèle (activation des comportements)
  - Calcul d’une séquence d’exécution qui mène à la cible
  - Concrétisé en cas de test Java exécutable
- Transfert de l’expérience BZ-Testing-Tools
- Ne considère pas le corps des méthodes en Java, seulement les annotations JML

Outils

JML-Testing-Tools

- Animateur symbolique
  - Animation symbolique du modèle JML en PLC
  - A partir des comportements extraits des annotations JML des méthodes
- Genérateur automatique de tests “aux limites”
  - Extraction des cibles de test à partir du modèle (activation des comportements)
  - Calcul d’une séquence d’exécution qui mène à la cible
  - Concrétisé en cas de test Java exécutable
- Transfert de l’expérience BZ-Testing-Tools
- Ne considère pas le corps des méthodes en Java, seulement les annotations JML

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
- Couverture des décisions
- Couverture des données

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
  - Normaux (terminaison normale de la méthode)
  - Exceptionnels (terminaison de la méthode par une exception)
- Couverture des décisions
- Couverture des données

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
  - Normaux (terminaison normale de la méthode)
  - Exceptionnels (terminaison de la méthode par une exception)
- Couverture des décisions
- Couverture des données

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
- Couverture des décisions
- Couverture des données

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
- Couverture des décisions
  Diverses décompositions des conditions disjonctives $P \vee Q$
  - Aucune : un seul comportement
  - Choix entre P et Q : deux comportements
  - Choix entre $P \wedge Q$, $P \wedge \neg Q$ et $\neg P \wedge Q$ (Full Predicate Coverage criterion)
- Couverture des données

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
- Couverture des décisions
- Couverture des données

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
- Couverture des décisions
- Couverture des données
  Sélectionne les valeurs limites des données
  - Valeurs aux bornes pour les numériques
    `max = 0` | `max = 32767`
  - Valeurs systématiques pour des objets
    Pointeur `null`, référence `this`, objet du type statique attendu, objet d'un sous-type du type attendu, cas d'aliasing

JML-Testing-Tools

Génération de tests

Critères de couverture

- Couverture des comportements
- Couverture des décisions
- Couverture des données
  Sélectionne les valeurs limites des données
  - Valeurs aux bornes pour les numériques
    `max = 0` | `max = 32767`
  - Valeurs systématiques pour des objets
    Pointeur `null`, référence `this`, objet du type statique attendu, objet d'un sous-type du type attendu, cas d'aliasing

Outils

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

Outils

JAG : JML Annotation Generator

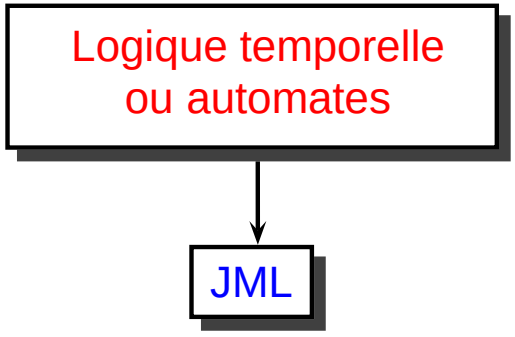
- Générateur de spécifications JML
- Principe
  - Formaliser des **propriétés de sécurité** à partir du cahier des charges
  - Traduire les propriétés dans un **format permettant la vérification formelle** sur le code Java
  - Vérifier les propriétés sur le code Java

Logique temporelle ou automates

Outils

JAG : JML Annotation Generator

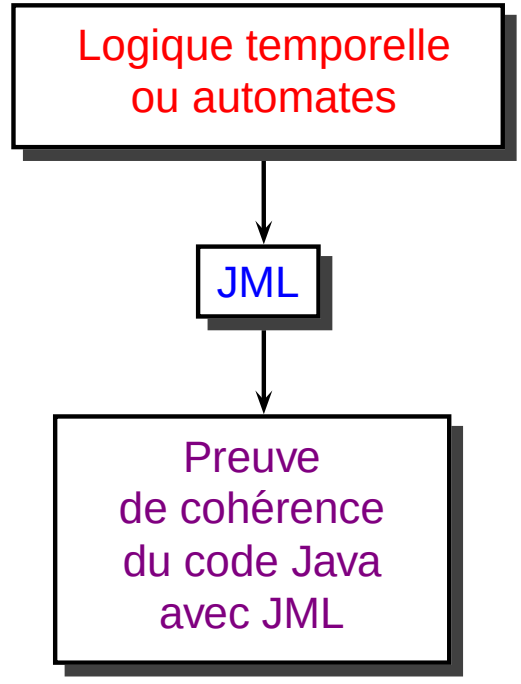
- Générateur de spécifications JML
- Principe
  - Formaliser des **propriétés de sécurité** à partir du cahier des charges
  - Traduire les propriétés dans un **format permettant la vérification formelle** sur le code Java
  - Vérifier les propriétés sur le code Java



Outils

JAG : JML Annotation Generator

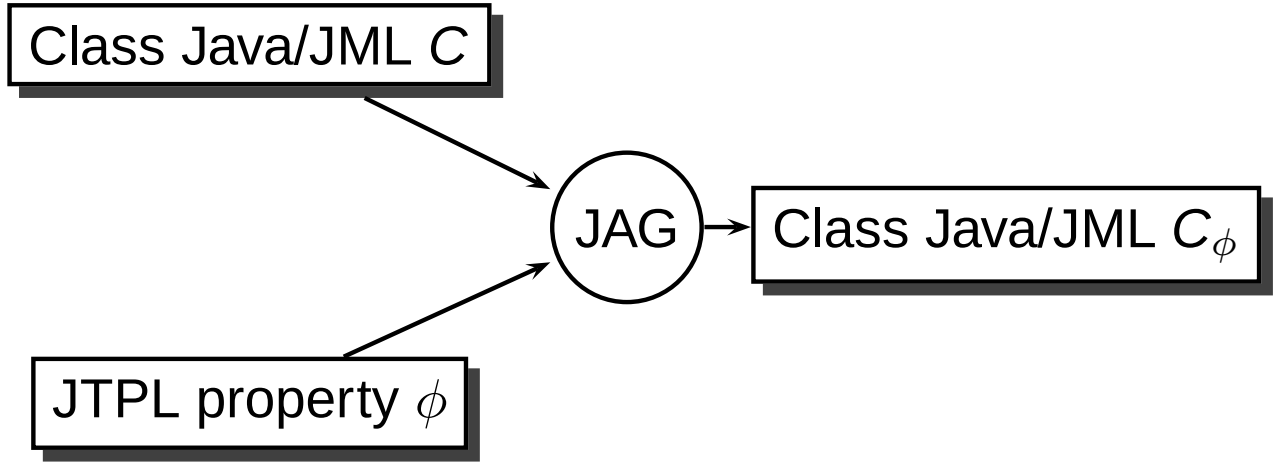
- Générateur de spécifications JML
- Principe
  - Formaliser des **propriétés de sécurité** à partir du cahier des charges
  - Traduire les propriétés dans un **format permettant la vérification formelle** sur le code Java
  - Vérifier les propriétés sur le code Java



Version 0.1 de JAG

Distribuée en mai 2006

- En entrée, formules JTPL (Huisman, Trentelman AMAST'02)
- En sortie, annotations JML
- Outil présenté à FASE'06 et AFADL'06



Résultats récents

- Correction du modèle JML
- Génération JML à partir d'automates (JAG 0.2)

Résultats récents

- Correction du modèle JML
- Génération JML à partir d'automates (JAG 0.2)

Correction du modèle JML

Motivations

- Requise pour extraire les tests du modèle
  - Cohérence du modèle de données
    - Préservation des invariants
    - Préservation des contraintes historiques
    - Cohérence entre invariants de différentes classes
  - Conditions d'activation des comportements issus des spécifications de méthodes mutuellement exclusives
  - Activabilité (consistance) des comportements (en isolation ou sous hypothèse des invariants)

Correction du modèle JML

Motivations

- Requise pour extraire les tests du modèle
  - Cohérence du modèle de données
    - Préservation des invariants
    - Préservation des contraintes historiques
    - Cohérence entre invariants de différentes classes
  - Conditions d'activation des comportements issus des spécifications de méthodes mutuellement exclusives
  - Activabilité (consistance) des comportements (en isolation ou sous hypothèse des invariants)

Correction du modèle JML

Motivations

- Requise pour extraire les tests du modèle
  - Cohérence du modèle de données
    - Préservation des invariants
    - Préservation des contraintes historiques
    - Cohérence entre invariants de différentes classes
  - Conditions d'activation des comportements issus des spécifications de méthodes mutuellement exclusives
  - Activabilité (consistance) des comportements (en isolation ou sous hypothèse des invariants)

Correction du modèle JML

Comment faire ?

- Générer des obligations de preuve pour effectuer ces vérifications
- Décharger les obligations de preuve dans un prouveur, ou les traiter par résolution de contraintes
- Produire un contre-exemple en cas de propriété non satisfaite

Correction du modèle JML

Comment faire ?

- Générer des obligations de preuve pour effectuer ces vérifications
- Décharger les obligations de preuve dans un prouveur, ou les traiter par résolution de contraintes
- Produire un contre-exemple en cas de propriété non satisfaite

Correction du modèle JML

Comment faire ?

- Générer des obligations de preuve pour effectuer ces vérifications
- Décharger les obligations de preuve dans un prouveur, ou les traiter par résolution de contraintes
- Produire un contre-exemple en cas de propriété non satisfaite

Correction du modèle JML

Une première expérimentation : JML2B

- Vérification de la cohérence d'un modèle JML (préservation invariants et contraintes historiques)
- Passage par une machine B représentant le modèle JML et ses conditions de correction
- Utilisation des outils de vérification de B : AtelierB, B4Free, . . .
- Principe publié à ZB'2005
- Prototype JML2B présenté à B'2007

Correction du modèle JML

Une première expérimentation : JML2B

- Vérification de la cohérence d'un modèle JML (préservation invariants et contraintes historiques)
- Passage par une machine B représentant le modèle JML et ses conditions de correction
- Utilisation des outils de vérification de B : AtelierB, B4Free, . . .
- Principe publié à ZB'2005
- Prototype JML2B présenté à B'2007

Correction du modèle JML

Une première expérimentation : JML2B

- Vérification de la cohérence d'un modèle JML (préservation invariants et contraintes historiques)
- Passage par une machine B représentant le modèle JML et ses conditions de correction
- Utilisation des outils de vérification de B : AtelierB, B4Free, ...
- Principe publié à ZB'2005
- Prototype JML2B présenté à B'2007

Correction du modèle JML

Une première expérimentation : JML2B

- Vérification de la cohérence d'un modèle JML (préservation invariants et contraintes historiques)
- Passage par une machine B représentant le modèle JML et ses conditions de correction
- Utilisation des outils de vérification de B : AtelierB, B4Free, ...
- Principe publié à ZB'2005
- Prototype JML2B présenté à B'2007

Correction du modèle JML

Une première expérimentation : JML2B

- Vérification de la cohérence d'un modèle JML (préservation invariants et contraintes historiques)
- Passage par une machine B représentant le modèle JML et ses conditions de correction
- Utilisation des outils de vérification de B : AtelierB, B4Free, ...
- Principe publié à ZB'2005
- Prototype JML2B présenté à B'2007

Résultats récents

Théoriques

- Condition de progrès pour les vivacités
  - Présenté à SAVCBS'06
- Vérification de propriétés LTL sur des systèmes d'événements B
  - Transfert d'expérience inverse : de JML vers B
  - Sera présenté à B'2007

Résultats récents

Théoriques

- Condition de progrès pour les vivacités
  - Présenté à SAVCBS'06
- Vérification de propriétés LTL sur des systèmes d'événements B
  - Transfert d'expérience inverse : de JML vers B
  - Sera présenté à B'2007

Résultats récents

Version 0.2 de JAG

- Nouveaux formats d'entrée
  - LTL
  - Automate de Buchi (syntaxe Promela)
- Format interne d'annotations génériques
- Sortie en janvier 2007
- Présentée aux JFLA 2007
- Règles d'inclusion de JTPL dans LTL

Résultats récents

Version 0.2 de JAG

- Nouveaux formats d'entrée
  - LTL
  - Automate de Buchi (syntaxe Promela)
- Format interne d'annotations génériques
- Sortie en janvier 2007
- Présentée aux JFLA 2007
- Règles d'inclusion de JTPL dans LTL

Résultats récents

Version 0.2 de JAG

- Nouveaux formats d'entrée
  - LTL
  - Automate de Buchi (syntaxe Promela)
- Format interne d'annotations génériques
- Sortie en janvier 2007
- Présentée aux JFLA 2007
- Règles d'inclusion de JTPL dans LTL

Résultats récents

Version 0.2 de JAG

- Nouveaux formats d'entrée
  - LTL
  - Automate de Buchi (syntaxe Promela)
- Format interne d'annotations génériques
- Sortie en janvier 2007
- Présentée aux JFLA 2007
- Règles d'inclusion de JTPL dans LTL

Résultats récents

Version 0.2 de JAG

- Nouveaux formats d'entrée
  - LTL
  - Automate de Buchi (syntaxe Promela)
- Format interne d'annotations génériques
- Sortie en janvier 2007
- Présentée aux JFLA 2007
- Règles d'inclusion de JTPL dans LTL

JAG 0.2

Langage/format interne d'annotations

- Générique
  - Indépendant du langage de programmation d'entrée
  - Indépendant du langage d'annotation de sortie
- Comment ?
  - Pré-process de traduction en automates
  - Post-process de concrétisation des annotations
- Variables témoin (affectation), invariant, variant (pour vivacités), postcondition condition

JAG 0.2

Langage/format interne d'annotations

- Générique
  - Indépendant du langage de programmation d'entrée
  - Indépendant du langage d'annotation de sortie
- Comment ?
  - Pré-process de traduction en automates
  - Post-process de concrétisation des annotations
- Variables témoin (affectation), invariant, variant (pour vivacités), postcondition condition

JAG 0.2

Langage/format interne d'annotations

- Générique
  - Indépendant du langage de programmation d'entrée
  - Indépendant du langage d'annotation de sortie
- Comment ?
  - Pré-process de traduction en automates
  - Post-process de concrétisation des annotations
- Variables témoin (affectation), invariant, variant (pour vivacités), postcondition condition

JAG 0.2

Traduction des automates de Buchi

- Etats de l'automate $\rightarrow$ variables témoins
  - Si l'automate est déterministe, une variable témoin de type entier
  - Si l'automate est indéterministe, *n* variables témoin de type boolean
  - 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 de *m*

JAG 0.2

Traduction des automates de Buchi

- Etats de l'automate $\rightarrow$ variables témoins
  - Si l'automate est déterministe, une variable témoin de type entier
  - Si l'automate est indéterministe, *n* variables témoin de type boolean
  - 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 de *m*

Perspectives

- Correction du modèle JML
- Génération JML à partir d'automates

Correction du modèle JML

Autres expérimentations envisagées

- Par résolution de contraintes avec le solveur CLPS-BZ (moteur symbolique de JML-TT)
- Directement de JML aux prouveurs, dans Krakatoa ou JACK
- Traiter toutes les conditions identifiées
- Ajout des conditions de raffinement
- JML2B
  - Vers une sortie multi prouveurs (via Why ou JACK)
  - Collaboration avec JML-TT pour la recherche de contre-exemples

Correction du modèle JML

Autres expérimentations envisagées

- Par résolution de contraintes avec le solveur CLPS-BZ (moteur symbolique de JML-TT)
- Directement de JML aux prouveurs, dans Krakatoa ou JACK
- Traiter toutes les conditions identifiées
- Ajout des conditions de raffinement
- JML2B
  - Vers une sortie multi prouveurs (via Why ou JACK)
  - Collaboration avec JML-TT pour la recherche de contre-exemples

Correction du modèle JML

Autres expérimentations envisagées

- Par résolution de contraintes avec le solveur CLPS-BZ (moteur symbolique de JML-TT)
- Directement de JML aux prouveurs, dans Krakatoa ou JACK
- Traiter toutes les conditions identifiées
- Ajout des conditions de raffinement
- JML2B
  - Vers une sortie multi prouveurs (via Why ou JACK)
  - Collaboration avec JML-TT pour la recherche de contre-exemples

Correction du modèle JML

Autres expérimentations envisagées

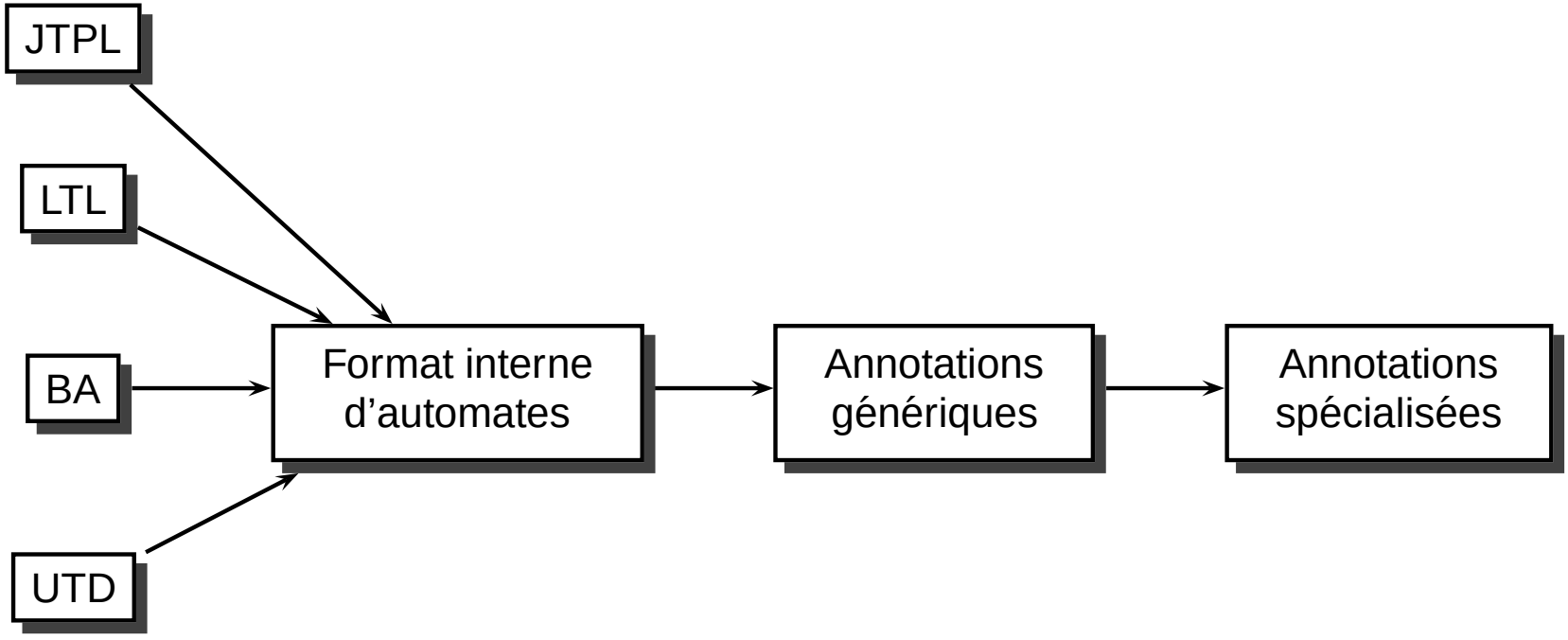
- Par résolution de contraintes avec le solveur CLPS-BZ (moteur symbolique de JML-TT)
- Directement de JML aux prouveurs, dans Krakatoa ou JACK
- Traiter toutes les conditions identifiées
- Ajout des conditions de raffinement
- JML2B
  - Vers une sortie multi prouveurs (via Why ou JACK)
  - Collaboration avec JML-TT pour la recherche de contre-exemples

Correction du modèle JML

Autres expérimentations envisagées

- Par résolution de contraintes avec le solveur CLPS-BZ (moteur symbolique de JML-TT)
- Directement de JML aux prouveurs, dans Krakatoa ou JACK
- Traiter toutes les conditions identifiées
- Ajout des conditions de raffinement
- JML2B
  - Vers une sortie multi prouveurs (via Why ou JACK)
  - Collaboration avec JML-TT pour la recherche de contre-exemples

Vers JAG 1.0



UTD : UML State Diagram

Formats d'automates

Un automate générique ?

- Supporter différents formats d'entrées
- Assurer la traçabilité des erreurs
- Mixer propriétés d'états et d'événements
- Contrôler, limiter le nombre d'annotations générées

$\Rightarrow$ Extension des automates de Buchi
Comparaison entre LabelBA (TFC) et MVA (Everest)

Perspectives

JML-TT

Collaborations entre outils

- Krakatoa/JML-TT : Validation par test en cas d'échec de la preuve
- JAG/JML-TT : Généralisation de la génération de tests orientée par les propriétés à d'autres logiques temporelles

Perspectives

JAG 1.0

- Format interne d'automates enrichis (avec Everest)
- Génération automatique de tests à partir de l'automate intermédiaire
- Traçabilité complète avec JACK
- Autres cibles de sortie (partenariats) : C♯/Spec♯ (en cours), C/Caduceus, Java/AspectJ, . . .
- Plug-in Eclipse (en cours)