MGL7160 Méthodes formelles et semi-formelles

Méthodologie

Roger Villemaire

Département d'informatique
UQAM

2 avril 2013

Plan

Plan

Méthodologie

- Une vérification réussie peut demander un effort important, il est donc important d'appliquer une approche systématique pour permettre de :
  - contrôler l'effort de vérification nécessaire,
  - éviter d'effectuer des vérifications redondantes,
  - éviter de faire des vérifications inutiles,
  - ou encore de bien déterminer la portée de la vérification effectuée.

Méthodologie

- Une vérification réussie peut demander un effort important, il est donc important d'appliquer une approche systématique pour permettre de :
  - contrôler l'effort de vérification nécessaire,
  - éviter d'effectuer des vérifications redondantes,
  - éviter de faire des vérifications inutiles,
  - ou encore de bien déterminer la portée de la vérification effectuée.

Méthodologie

- Une vérification réussie peut demander un effort important, il est donc important d'appliquer une approche systématique pour permettre de :
  - contrôler l'effort de vérification nécessaire,
  - éviter d'effectuer des vérifications redondantes,
  - éviter de faire des vérifications inutiles,
  - ou encore de bien déterminer la portée de la vérification effectuée.

Méthodologie

- Une vérification réussie peut demander un effort important, il est donc important d'appliquer une approche systématique pour permettre de :
  - contrôler l'effort de vérification nécessaire,
  - éviter d'effectuer des vérifications redondantes,
  - éviter de faire des vérifications inutiles,
  - ou encore de bien déterminer la portée de la vérification effectuée.

Méthodologie

- Une vérification réussie peut demander un effort important, il est donc important d'appliquer une approche systématique pour permettre de :
  - contrôler l'effort de vérification nécessaire,
  - éviter d'effectuer des vérifications redondantes,
  - éviter de faire des vérifications inutiles,
  - ou encore de bien déterminer la portée de la vérification effectuée.

Méthodologie

- Une vérification réussie peut demander un effort important, il est donc important d'appliquer une approche systématique pour permettre de :
  - contrôler l'effort de vérification nécessaire,
  - éviter d'effectuer des vérifications redondantes,
  - éviter de faire des vérifications inutiles,
  - ou encore de bien déterminer la portée de la vérification effectuée.

Spécification

- L'approche JML est de considérer, pour un programme, une spécification formée :
  - d'invariants, qui contraignent les valeurs des attributs,
  - de contrats, qui contraignent le comportement des opérations.
- Un contrat est formé de préconditions, de postconditions et d'un ensemble de références qui peuvent être modifiées.

Spécification

- L'approche JML est de considérer, pour un programme, une spécification formée :
  - d'invariants, qui contraignent les valeurs des attributs,
  - de contrats, qui contraignent le comportement des opérations.
- Un contrat est formé de préconditions, de postconditions et d'un ensemble de références qui peuvent être modifiées.

Spécification

- L'approche JML est de considérer, pour un programme, une spécification formée :
  - d'invariants, qui contraignent les valeurs des attributs,
  - de contrats, qui contraignent le comportement des opérations.
- Un contrat est formé de préconditions, de postconditions et d'un ensemble de références qui peuvent être modifiées.

Spécification

- L'approche JML est de considérer, pour un programme, une spécification formée :
  - d'invariants, qui contraignent les valeurs des attributs,
  - de contrats, qui contraignent le comportement des opérations.
- Un contrat est formé de préconditions, de postconditions et d'un ensemble de références qui peuvent être modifiées.

Spécification

- L'approche JML est de considérer, pour un programme, une spécification formée :
  - d'invariants, qui contraignent les valeurs des attributs,
  - de contrats, qui contraignent le comportement des opérations.
- Un contrat est formé de préconditions, de postconditions et d'un ensemble de références qui peuvent être modifiées.

États observés

- L'approche privilégiée par KeY est celle dite des *états observés*, où
  - chaque opération doit satisfaire ses contrats,
  - les invariants doivent être vérifiés à l'initialisation du système,
  - les invariants doivent être préservés par toutes les opérations.

États observés

- L'approche privilégiée par KeY est celle dite des *états observés*, où
  - chaque opération doit satisfaire ses contrats,
  - les invariants doivent être vérifiés à l'initialisation du système,
  - les invariants doivent être préservés par toutes les opérations.

États observés

- L'approche privilégiée par KeY est celle dite des *états observés*, où
  - chaque opération doit satisfaire ses contrats,
  - les invariants doivent être vérifiés à l'initialisation du système,
  - les invariants doivent être préservés par toutes les opérations.

États observés

- L'approche privilégiée par KeY est celle dite des *états observés*, où
  - chaque opération doit satisfaire ses contrats,
  - les invariants doivent être vérifiés à l'initialisation du système,
  - les invariants doivent être préservés par toutes les opérations.

États observés

- L'approche privilégiée par KeY est celle dite des *états observés*, où
  - chaque opération doit satisfaire ses contrats,
  - les invariants doivent être vérifiés à l'initialisation du système,
  - les invariants doivent être préservés par toutes les opérations.

Plan

Obligations

- L'outil KeY génère automatiquement à partir d'une spécification JML des formules JavaDL.
- KeY utilise des gabarits qui produisent des formules (proof obligations), correspondant à l'approche de vérification des états observés.
- Il peut donc y avoir des façons différentes de justifier des choses analogues.
- C'est pourquoi il faut être attentif à la méthodologie de vérification utilisée ainsi qu'à ses avantages et limitations.

Obligations

- L'outil KeY génère automatiquement à partir d'une spécification JML des formules JavaDL.
- KeY utilise des gabarits qui produisent des formules (proof obligations), correspondant à l'approche de vérification des états observés.
- Il peut donc y avoir des façons différentes de justifier des choses analogues.
- C'est pourquoi il faut être attentif à la méthodologie de vérification utilisée ainsi qu'à ses avantages et limitations.

Obligations

- L'outil KeY génère automatiquement à partir d'une spécification JML des formules JavaDL.
- KeY utilise des gabarits qui produisent des formules (proof obligations), correspondant à l'approche de vérification des états observés.
- Il peut donc y avoir des façons différentes de justifier des choses analogues.
- C'est pourquoi il faut être attentif à la méthodologie de vérification utilisée ainsi qu'à ses avantages et limitations.

Obligations

- L'outil KeY génère automatiquement à partir d'une spécification JML des formules JavaDL.
- KeY utilise des gabarits qui produisent des formules (proof obligations), correspondant à l'approche de vérification des états observés.
- Il peut donc y avoir des façons différentes de justifier des choses analogues.
- C'est pourquoi il faut être attentif à la méthodologie de vérification utilisée ainsi qu'à ses avantages et limitations.

Obligations

- L'outil KeY génère automatiquement à partir d'une spécification JML des formules JavaDL.
- KeY utilise des gabarits qui produisent des formules (proof obligations), correspondant à l'approche de vérification des états observés.
- Il peut donc y avoir des façons différentes de justifier des choses analogues.
- C'est pourquoi il faut être attentif à la méthodologie de vérification utilisée ainsi qu'à ses avantages et limitations.

Le rôle des invariants

- Les invariants d'instance d'une classe sont des contraintes sur les attributs qui doivent être satisfaites sur des instances "correctes" de la classe.
- Les invariants font donc partie intégrante de la définition de la classe, même si le langage de programmation n'offre pas de support pour eux.
- Néanmoins, lors de la conception du système, les invariants sont, souvent implicitement, considérés comme étant satisfaits.
- En particulier, lors de la réalisation des opérations, on présuppose normalement que les invariants sont satisfaits sur le receveur (`self`).
  - L'opération sera correcte que sur des arguments corrects !

Le rôle des invariants

- Les invariants d'instance d'une classe sont des contraintes sur les attributs qui doivent être satisfaites sur des instances "correctes" de la classe.
- Les invariants font donc partie intégrante de la définition de la classe, même si le langage de programmation n'offre pas de support pour eux.
- Néanmoins, lors de la conception du système, les invariants sont, souvent implicitement, considérés comme étant satisfaits.
- En particulier, lors de la réalisation des opérations, on présuppose normalement que les invariants sont satisfaits sur le receveur (`self`).
  - L'opération sera correcte que sur des arguments corrects !

Le rôle des invariants

- Les invariants d'instance d'une classe sont des contraintes sur les attributs qui doivent être satisfaites sur des instances "correctes" de la classe.
- Les invariants font donc partie intégrante de la définition de la classe, même si le langage de programmation n'offre pas de support pour eux.
- Néanmoins, lors de la conception du système, les invariants sont, souvent implicitement, considérés comme étant satisfaits.
- En particulier, lors de la réalisation des opérations, on présuppose normalement que les invariants sont satisfaits sur le receveur (`self`).
  - L'opération sera correcte que sur des arguments corrects !

Le rôle des invariants

- Les invariants d'instance d'une classe sont des contraintes sur les attributs qui doivent être satisfaites sur des instances "correctes" de la classe.
- Les invariants font donc partie intégrante de la définition de la classe, même si le langage de programmation n'offre pas de support pour eux.
- Néanmoins, lors de la conception du système, les invariants sont, souvent implicitement, considérés comme étant satisfaits.
- En particulier, lors de la réalisation des opérations, on présuppose normalement que les invariants sont satisfaits sur le receveur (`self`).
  - L'opération sera correcte que sur des arguments corrects !

Le rôle des invariants

- Les invariants d'instance d'une classe sont des contraintes sur les attributs qui doivent être satisfaites sur des instances "correctes" de la classe.
- Les invariants font donc partie intégrante de la définition de la classe, même si le langage de programmation n'offre pas de support pour eux.
- Néanmoins, lors de la conception du système, les invariants sont, souvent implicitement, considérés comme étant satisfaits.
- En particulier, lors de la réalisation des opérations, on présuppose normalement que les invariants sont satisfaits sur le receveur (`self`).
  - L'opération sera correcte que sur des arguments corrects !

Le rôle des invariants

- Les invariants d'instance d'une classe sont des contraintes sur les attributs qui doivent être satisfaites sur des instances "correctes" de la classe.
- Les invariants font donc partie intégrante de la définition de la classe, même si le langage de programmation n'offre pas de support pour eux.
- Néanmoins, lors de la conception du système, les invariants sont, souvent implicitement, considérés comme étant satisfaits.
- En particulier, lors de la réalisation des opérations, on présuppose normalement que les invariants sont satisfaits sur le receveur (`self`).
  - L'opération sera correcte que sur des arguments corrects !

Le rôle des invariants en KeY

- Conformément à cette vision des choses, le système KeY ajoute les invariants aux préconditions lors de la vérification d'un contrat.
- Le choix des invariants est configurable au moment du choix du contrat (onglet "Assumed Invariants").
- Comme on l'a déjà vu, KeY ajoute déjà aux contrats les invariants implicites suivants.
  - `self` est créé et non-nul,
  - les arguments de l'opération sont créés et non-nuls.

Le rôle des invariants en KeY

- Conformément à cette vision des choses, le système KeY ajoute les invariants aux préconditions lors de la vérification d'un contrat.
- Le choix des invariants est configurable au moment du choix du contrat (onglet "Assumed Invariants").
- Comme on l'a déjà vu, KeY ajoute déjà aux contrats les invariants implicites suivants.
  - `self` est créé et non-nul,
  - les arguments de l'opération sont créés et non-nuls.

Le rôle des invariants en KeY

- Conformément à cette vision des choses, le système KeY ajoute les invariants aux préconditions lors de la vérification d'un contrat.
- Le choix des invariants est configurable au moment du choix du contrat (onglet "Assumed Invariants").
- Comme on l'a déjà vu, KeY ajoute déjà aux contrats les invariants implicites suivants.
  - `self` est créé et non-nul,
  - les arguments de l'opération sont créés et non-nuls.

Le rôle des invariants en KeY

- Conformément à cette vision des choses, le système KeY ajoute les invariants aux préconditions lors de la vérification d'un contrat.
- Le choix des invariants est configurable au moment du choix du contrat (onglet "Assumed Invariants").
- Comme on l'a déjà vu, KeY ajoute déjà aux contrats les invariants implicites suivants.
  - `self` est créé et non-nul,
  - les arguments de l'opération sont créés et non-nuls.

Le rôle des invariants en KeY

- Conformément à cette vision des choses, le système KeY ajoute les invariants aux préconditions lors de la vérification d'un contrat.
- Le choix des invariants est configurable au moment du choix du contrat (onglet "Assumed Invariants").
- Comme on l'a déjà vu, KeY ajoute déjà aux contrats les invariants implicites suivants.
  - `self` est créé et non-nul,
  - les arguments de l'opération sont créés et non-nuls.

Le rôle des invariants en KeY

- Conformément à cette vision des choses, le système KeY ajoute les invariants aux préconditions lors de la vérification d'un contrat.
- Le choix des invariants est configurable au moment du choix du contrat (onglet "Assumed Invariants").
- Comme on l'a déjà vu, KeY ajoute déjà aux contrats les invariants implicites suivants.
  - `self` est créé et non-nul,
  - les arguments de l'opération sont créés et non-nuls.

Plan

Invariants

- Il est donc recommandé de mettre au clair, sous forme d'invariants, toute restriction importante pour comprendre le sens ou pour réaliser le système.
- Au niveau de la vérification formelle, ceci aura comme effet de centraliser ces contraintes et donc de simplifier les préconditions des contrats.

Invariants

- Il est donc recommandé de mettre au clair, sous forme d'invariants, toute restriction importante pour comprendre le sens ou pour réaliser le système.
- Au niveau de la vérification formelle, ceci aura comme effet de centraliser ces contraintes et donc de simplifier les préconditions des contrats.

Invariants

- Il est donc recommandé de mettre au clair, sous forme d'invariants, toute restriction importante pour comprendre le sens ou pour réaliser le système.
- Au niveau de la vérification formelle, ceci aura comme effet de centraliser ces contraintes et donc de simplifier les préconditions des contrats.

Exemple d'invariants

```
public class Camion {...
 /*@ public instance invariant
     prochaineLivraison < noLivraisonsMax;

     public instance invariant
     prochaineLivraison >= 0;

     public instance invariant
     livraisons.length == noLivraisonsMax;
 @*/
}
```

Vérification des invariants

- Il est aussi nécessaire de s'assure que les invariants sont satisfaits.
- Ceci est particulièrement vrai lorsque les invariants servent à justifier les contrats des opérations :
  - Un invariant qui est faux pourrait même justifier n'importe quel contrat !

Vérification des invariants

- Il est aussi nécessaire de s'assure que les invariants sont satisfaits.
- Ceci est particulièrement vrai lorsque les invariants servent à justifier les contrats des opérations :
  - Un invariant qui est faux pourrait même justifier n'importe quel contrat !

Vérification des invariants

- Il est aussi nécessaire de s'assure que les invariants sont satisfaits.
- Ceci est particulièrement vrai lorsque les invariants servent à justifier les contrats des opérations :
  - Un invariant qui est faux pourrait même justifier n'importe quel contrat !

Vérification des invariants

- Il est aussi nécessaire de s'assure que les invariants sont satisfaits.
- Ceci est particulièrement vrai lorsque les invariants servent à justifier les contrats des opérations :
  - Un invariant qui est faux pourrait même justifier n'importe quel contrat !

KeY

- Key permet de vérifier qu'un invariant est satisfait après l'appel d'une opération.
- En fait, "PreservesInv" génère un nouveau contrat en conservant la précondition du contrat actuel et en prenant l'invariant comme postcondition.

KeY

- Key permet de vérifier qu'un invariant est satisfait après l'appel d'une opération.
- En fait, "PreservesInv" génère un nouveau contrat en conservant la précondition du contrat actuel et en prenant l'invariant comme postcondition.

KeY

- Key permet de vérifier qu'un invariant est satisfait après l'appel d'une opération.
- En fait, "PreservesInv" génère un nouveau contrat en conservant la précondition du contrat actuel et en prenant l'invariant comme postcondition.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Méthodologie

- Concrètement, notre méthode d'analyse des programmes consistera à
  - rédiger des invariants qui décrivent les propriétés naturelles des instances,
  - rédiger des contrats qui décrivent le comportement des opérations.
- Faire vérifier par KeY que
  - les invariants sont préservés par les opérations,
  - les contrats sont satisfaits par les méthodes.
  - À chaque fois, il sera nécessaire de choisir les invariants pertinents comme préconditions.

Aspects techniques (avant la preuve)

- KeY génère des obligations en Java-DL, la logique dynamique de l'outil.
- Il est nécessaire de bien observer la formule produite, puisqu'il peut y avoir des nuances qui auront un impact important sur la vérification.
- Il est important de bien comprendre le contenu de la formule en terme d'information. Par exemple :
  - quelles sont les hypothèses sur les références (attributs du receveur, arguments).

Aspects techniques (avant la preuve)

- KeY génère des obligations en Java-DL, la logique dynamique de l'outil.
- Il est nécessaire de bien observer la formule produite, puisqu'il peut y avoir des nuances qui auront un impact important sur la vérification.
- Il est important de bien comprendre le contenu de la formule en terme d'information. Par exemple :
  - quelles sont les hypothèses sur les références (attributs du receveur, arguments).

Aspects techniques (avant la preuve)

- KeY génère des obligations en Java-DL, la logique dynamique de l'outil.
- Il est nécessaire de bien observer la formule produite, puisqu'il peut y avoir des nuances qui auront un impact important sur la vérification.
- Il est important de bien comprendre le contenu de la formule en terme d'information. Par exemple :
  - quelles sont les hypothèses sur les références (attributs du receveur, arguments).

Aspects techniques (avant la preuve)

- KeY génère des obligations en Java-DL, la logique dynamique de l'outil.
- Il est nécessaire de bien observer la formule produite, puisqu'il peut y avoir des nuances qui auront un impact important sur la vérification.
- Il est important de bien comprendre le contenu de la formule en terme d'information. Par exemple :
  - quelles sont les hypothèses sur les références (attributs du receveur, arguments).

Aspects techniques (avant la preuve)

- KeY génère des obligations en Java-DL, la logique dynamique de l'outil.
- Il est nécessaire de bien observer la formule produite, puisqu'il peut y avoir des nuances qui auront un impact important sur la vérification.
- Il est important de bien comprendre le contenu de la formule en terme d'information. Par exemple :
  - quelles sont les hypothèses sur les références (attributs du receveur, arguments).

Aspects techniques (après la preuve)

- Lorsque la vérification échoue, il est aussi important d'interpréter les feuilles (séquents) ouvertes. Par exemple :
  - N'y a-t-il pas des hypothèses trop générales (`a>=b` plutôt que `a=b`) ?
  - N'y a-t-il pas des contraintes manquantes (pas de contrainte sur `a,b`) ?
  - Y a-t-il un lien entre les hypothèses et les conclusions du séquent ?
  - Est-ce que le séquent est vide, ou clairement non-démontrable ?

Aspects techniques (après la preuve)

- Lorsque la vérification échoue, il est aussi important d'interpréter les feuilles (séquents) ouvertes. Par exemple :
  - N'y a-t-il pas des hypothèses trop générales (`a>=b` plutôt que `a=b`) ?
  - N'y a-t-il pas des contraintes manquantes (pas de contrainte sur `a,b`) ?
  - Y a-t-il un lien entre les hypothèses et les conclusions du séquent ?
  - Est-ce que le séquent est vide, ou clairement non-démontrable ?

Aspects techniques (après la preuve)

- Lorsque la vérification échoue, il est aussi important d'interpréter les feuilles (séquents) ouvertes. Par exemple :
  - N'y a-t-il pas des hypothèses trop générales (`a>=b` plutôt que `a=b`) ?
  - N'y a-t-il pas des contraintes manquantes (pas de contrainte sur `a,b`) ?
  - Y a-t-il un lien entre les hypothèses et les conclusions du séquent ?
  - Est-ce que le séquent est vide, ou clairement non-démontrable ?

Aspects techniques (après la preuve)

- Lorsque la vérification échoue, il est aussi important d'interpréter les feuilles (séquents) ouvertes. Par exemple :
  - N'y a-t-il pas des hypothèses trop générales (`a>=b` plutôt que `a=b`) ?
  - N'y a-t-il pas des contraintes manquantes (pas de contrainte sur `a,b`) ?
  - Y a-t-il un lien entre les hypothèses et les conclusions du séquent ?
  - Est-ce que le séquent est vide, ou clairement non-démontrable ?

Aspects techniques (après la preuve)

- Lorsque la vérification échoue, il est aussi important d'interpréter les feuilles (séquents) ouvertes. Par exemple :
  - N'y a-t-il pas des hypothèses trop générales (`a>=b` plutôt que `a=b`) ?
  - N'y a-t-il pas des contraintes manquantes (pas de contrainte sur `a,b`) ?
  - Y a-t-il un lien entre les hypothèses et les conclusions du séquent ?
  - Est-ce que le séquent est vide, ou clairement non-démontrable ?

Aspects techniques (après la preuve)

- Lorsque la vérification échoue, il est aussi important d'interpréter les feuilles (séquents) ouvertes. Par exemple :
  - N'y a-t-il pas des hypothèses trop générales (`a>=b` plutôt que `a=b`) ?
  - N'y a-t-il pas des contraintes manquantes (pas de contrainte sur `a,b`) ?
  - Y a-t-il un lien entre les hypothèses et les conclusions du séquent ?
  - Est-ce que le séquent est vide, ou clairement non-démontrable ?