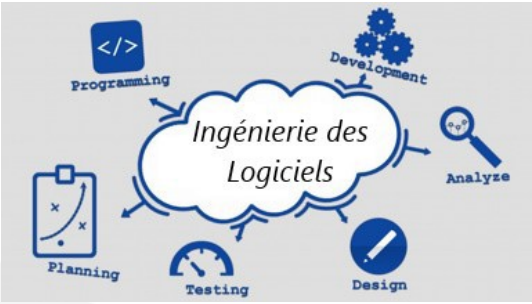
Ingénierie des Logiciels



Dr. Said MEGHZILI

Centre Universitaire de Mila

Département de Mathématique et Informatique

Email: s.meghzili@univ-mila.dz

1.0

Février 2022

Table des matières

I Chapitre 04 : Méthodes Formelles et ses Applications

1. Introduction

Définition : Système critique

c'est un système dont lequel les ***erreurs*** peuvent conduire a une ***catastrophe humaine*** ou ***matérielle***.

Figure 4.1 : Exemple des Systèmes Critiques

Exemple : Explosion d'Ariane 5 : Bugs & Erreurs des logiciels

Figure 4.2 : La fusée Ariane 5 de l'Agence Spatiale Européenne

- L'explosion d'Ariane 5 après 37 s de son départ, le 4 juin 1996, qui a coûté un demi milliard de dollars.
- ***Cause*** : une ***faute logicielle*** d'une composante dont le fonctionnement n'était pas indispensable durant le vol.
- Il contenait une conversion d'un flottant sur 64 bits en un entier signé sur 16 bits.
- Pour Ariane 5, la valeur du flottant dépassait la valeur maximale pouvant être convertie.
  ⇒ Nécessité de « ***vérifier*** » certains logiciels/systèmes en utilisant les ***Méthodes Formelles***: Test, Démonstrateur de théorèmes, Model-checking.

Définition : Méthodes Formelles

Les méthodes formelles (MFs) sont des techniques basées sur des principes mathématiques pour décrire le comportement de systèmes et les propriétés à vérifier. Elles fournissent un cadre très rigoureux pour spécifier, développer et vérifier des systèmes d'une façon systématique, plutôt que d'une façon ad hoc, afin de démontrer leur validité par rapport à une certaine spécification [20]*. Dans la littérature, il existe plusieurs classifications des méthodes formelles. Selon la référence [22]*, on peut distinguer les trois classes suivantes:

1) Algébriques :

- CCS (Milner:79)
- CSP (Hoar:85)
- ACP (Bergstra:85)
- LOTOS (Bolognesi et al : 87)
- Pi-calculus (Milner:92)
- RT-LOTOS (Courtiat et al :93)
- ET-LOTOS (Leduc et al : 93)
- D-LOTOS (Saidouni et al : 2003)

2) Langages :

- Estelle
- Maude (Logique de réécriture)
- Mobile Maude
- Real Time Maude

3) Réseaux de Petri :

- Places/Transitions
- Prédicat
- Colorés
- Algébriques
- Temporel
- Temporisé
- Temporisés stochastiques

2. Modèles Formels: Réseaux de Petri

2.1. Définition Formelle et Représentation des RdPs

Définition : Réseaux de Petri (RdP)

Les RdPs sont un outil utilisé pour la spécification et la vérification des systèmes. ils ont été introduits initialement par C.A PETRI (1962). Ils disposent d'une représentation graphique facilement compréhensible par les utilisateurs et également d'une fondation mathématique solide permettant l'application de techniques de vérification sur les systèmes modélisés avant leur déploiement.

Un réseau de Petri R est un quadruplet {P, T, Pre, Post}, avec :

- P = {p1, p2,..., pn} un ensemble de places (les états du système).
- T = {t1,..., tn}, un ensemble de transitions, avec P∩T = Ø.
- Pré: PXT -> N est la fonction d'incidence avant.
- Post : PXT -> N est la fonction d'incidence arrière.

Pre(pi, tj) représente le poids de l'arc allant de pi vers tj.

Post(pi, tj) représente le poids de l'arc allant de tj vers pi.

Complément : Représentation des Réseaux de Petri

Un RdP possède deux représentations : Graphique & Matricielle. La représentation graphique est un graphe biparti orienté et valu:

- Places: cercles
- Transitions: bars or rectangles
- Arcs: flèches étiquetées par des valeurs
- Jetons: points noirs (black dots)

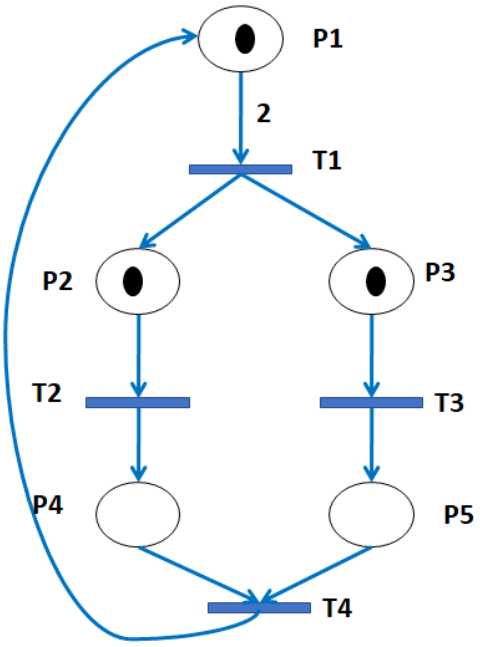


Figure 4.3 : Représentation graphique d'un Réseau de Petri

La matrice **pré:**

	T1	T2	T3	T4
P1	2	0	0	0
P2	0	1	0	0
P3	0	0	1	0
P4	0	0	0	1
P5	0	0	0	1

La matrice **post:**

	T1	T2	T3	T4
P1	0	0	0	1
P2	1	0	0	0
P3	1	0	0	0
P4	0	1	0	0
P5	0	0	1	0

Le marquage initial:

	M0
P1	1
P2	1
P3	1
P4	0
P5	0

Figure 4.4 : Représentation Matricielle d'un Réseau de Petri

Complément : Marquage

Un réseau de pétri marqué est définie par le couple {R, M}, où R est un réseau de pétri, et M : P-> N est une application appelé marquage qui associe à chaque place de R un nombre de jetons. Le marquage initial est noté m0 son rôle est de spécifier l'état initial du système. Par exemple, le marquage initial de RdP dans la figure 4.3 est : m0 (1, 1, 1 , 0, 0).

2.2. Franchissement des Transitions et Graphe de marquage

Définition : Franchissement d'une transition

- Pour une transition t, on dénote •t (resp. t•) l'ensemble de places d'entrée (resp. de sortie) de la transition t.
- Une transition est franchissable (“is enabled”) si et seulement s'il y a assez de jetons dans chaque place en entrée

  $\forall p \in {}^\bullet t : M(p) >= pré(p,t)$
- Notation M [t > : t est franchissable dans le marquage M
- Le franchissement d'une transition modifie le marquage en consommant des jetons dans les places d'entrée et en produisant des jetons dans les places de sortie.
- Si une transition t est sensibilisée pour un marquage M, t peut être franchie à partir de M, produisant un nouveau marquage M′, dénoté M −(t)-> M′, où

  $\forall p \in P, M'(p) = M(p) - pré(p, t) + post(p, t)$

  La figure 4.5 montre le franchissement de la transition t1.

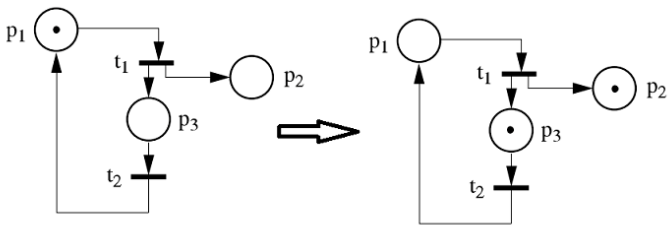


Figure 4.5 : Franchissement de la transition t1

Définition : Graphe de marquage (graphe d'accessibilité)

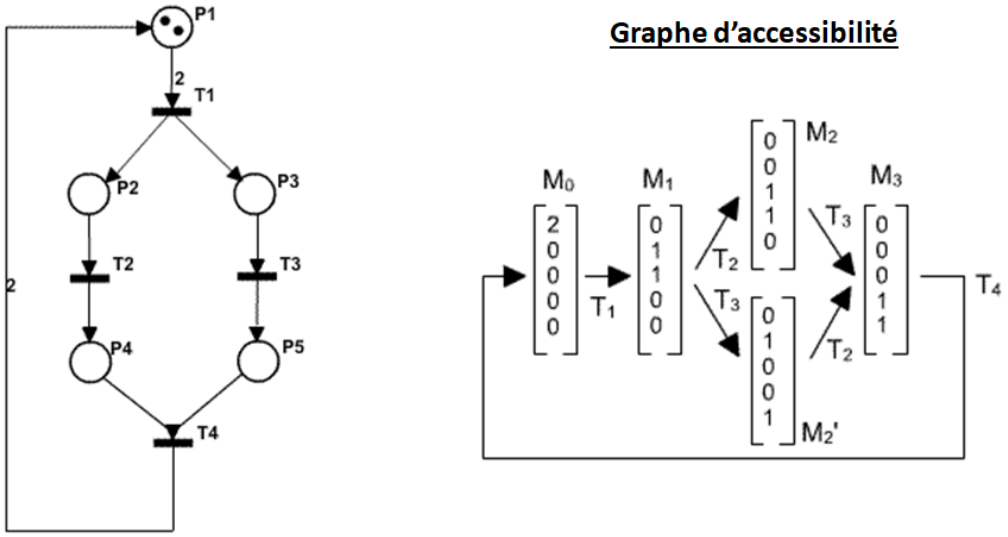


Figure 4.6 : Exemple d'un RdP et son graphe de marquage

- A(R,M0) est l'ensemble des marquages accessibles.
- Le graphe d'accessibilité GA(R,M0) de ce réseau à partir du marquage initial M0 est un graphe dirigé et étiqueté dont les sommets sont les éléments de A(R,M0).
- Un arc relie deux marquages Mi et Mj si et ssi il existe une transition t de R tel que: Mi −(t)-> Mj.
  La figure 4.6 montre Un RdP et son graphe de marquage.

2.3. Propriétés des Réseaux de Petri

Définition

- ***Accessibilité*** (reachability): un marquage m est accessible (à partir de m0) s'il existe une séquence de transitions t1,t2,.....,tn qui conduit de m0 à m.
  C à d :m0 [t1> m1 [t2> m2.....mn [tn> m
- ***Deadlock-free***: si chaque marquage accessible permet au moins une transition.
- Une transition t d'un réseau de Petri (R,M0) est ***quasi-vivante*** :
  ssi ∃ M accessible à partir de M0, tel que M [t >
- Une transition t d'un réseau de Petri (R,M0) est ***vivante***: ssi pour chaque marquage accessible M′, il existe un marquage M′′accessible à partir de M′ où t est sensibilisée.
- Un réseau de Petri (R,M0) est ***vivant*** (resp. quasi-vivant) si et ssi toutes ses transitions sont ***vivantes*** (resp. quasi-vivantes)

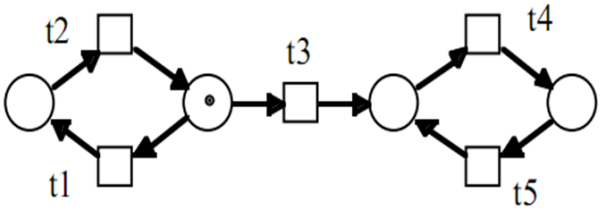


Figure 4.7 : Exemple d'un RdP qui est deadlock-free mais non vivant

Complément

- ***Borné*** : un RdP est dit K-borné si le nombre de jetons dans chaque place p est toujours inférieur ou égale à k pour chaque marquage accessible à partir de m0. La figure 4.8 montre un RdP non borné.
- ***Safe*** (1-borné): un Rdp est safe s'il est 1-borné.
- ***Réversible*** : si le marquage initial m0 est accessible depuis n'importe quel autre marquage.

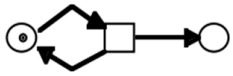


Figure 4.8 : Exemple d'un RdP non borné (unbounded)

3. Intégration des Méthodes Formelles avec l'IDM

Méthodes formelles (MFs)

- Les MFs permettent à la fois de modéliser le système et de vérifier les propriétés attendues.
- Les MFs reposent sur l'utilisation de langages formels dotés une sémantique mathématique rigoureuse.
- Principaux obstacles d'utilisation des MFs dans les activités de développement des systèmes sont liés à :
    - La difficulté réelle de manipuler les concepts théoriques et les méthodes d'analyse associées.
    - La difficulté pour les développeurs d'exprimer les propriétés du système d'une façon aisée.

Ingénierie Dirigée par les Modèles (IDM)

- La vérification des modèles comptent aujourd'hui parmi les enjeux les plus importants de l'IDM.
- L'IDM joue un rôle essentiel dans l'introduction des MFs dans les activités de développement des systèmes.
- Celle-ci repose sur:
    - La définition de langages formels par le biais de la méta-modélisation.
    - L'utilisation de transformations de modèles pour générer des modèles décrits dans ces langages formels.

Combinaison de l'IDM avec les MFs

	Avantages	*Inconvénients*
IDM	√ Notations conviviales et souvent graphiques √ Génération automatique d'outils de développement √ Transformations automatiques de modèles	✗ Manque de sémantiques formelles ✗ Analyse de modèles impossible
MFs	√ Fondements mathématiques rigoureux √ Analyse formelle de modèles	✗ Notations complexes ✗ Absence d'outils de développement ✗ Manque d'intégration

Figure 4.9 : Avantages et Inconvénients de l'IDM et des Méthodes Formelles

Chacune des deux approches a des points forts et des points faibles. Donc, les deux approches peuvent être combinées dans le sens où les inconvénients de l'un peuvent être surmontés grâce aux apports de l'autre [1]*.

4. Vérification d'une Transformation

4.1. Pourquoi la vérification d'une transformation ?

Problématique

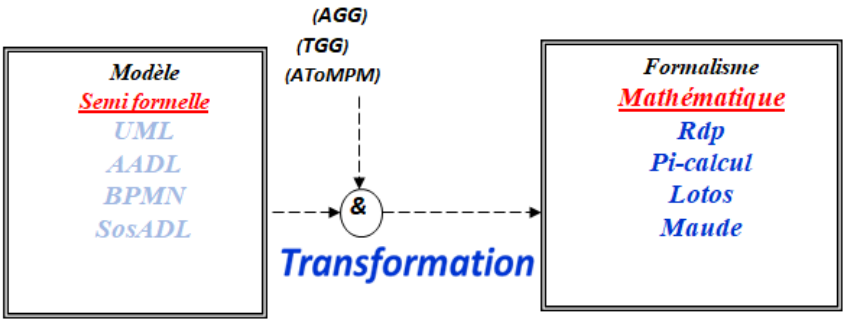


Figure 4.10 : Exemple d'approches de transformation de modèles

- Plusieurs outils de la transformation sont développés : TGG, AGG, AToMPM, . . . etc.
- Plusieurs transformations sont réalisées notamment des modèles semi-formelles (UML, BPMN, AADL) vers des modèles formels.
- Mais ces ***transformations*** et leurs ***outils*** souffrent de ***manque de vérification***
- ***Propriétés*** à Vérifier:
    - Terminaison de la transformation
    - Préservation de la sémantique du modèle source
    - Confluence
    - Invariants
    - Complétude
    - Absence d'interblocage & boucle infinie ...
- Donc, il est intéressant d'appeler à de nouvelles méthodes, techniques et outils pour le développement des transformations de modèles.

4.2. Approche de trois dimensions

Définition

Cette approche[12]* est composée de trois dimensions qui sont présentées dans la Figure 4.11. Ces dimensions sont : la transformation elle-même, le type de propriété à vérifier et la technique de vérification à utiliser pour modéliser la transformation, spécifier les propriétés attendues et vérifier si la transformation satisfait ses propriétés attendus.

- ***Transformation*** : la notion de transformation de modèles constitue l'élément central de l'ingénierie diriges par les modelés. Un modèle qui est conformé à un méta modèle source est transformé dans un autre modèle qui est conforme à un méta modèle cible en utilisant un ensemble de règles de transformation.
- ***Propriétés à vérifier*** : les propriétés sont les propriétés fonctionnelles de la transformation elle-même et les propriétés liées aux modelés source et cible.

- ***Techniques de vérification*** : les techniques utilisées pour vérifier formellement la correction d'une transformation sont le ***Model checker*** et les ***démonstrateurs de théorèmes***.

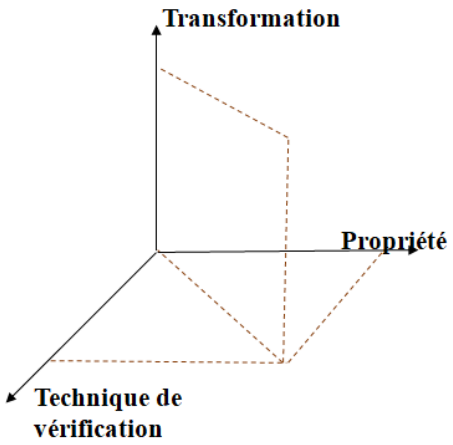


Figure 4.11 : Approche de trois dimensions [Amrani et al. 2015]

Complément : Propriétés à vérifier

Il existe deux classes de propriétés [12]*:

1. Propriétés fonctionnelles de la transformation elle-même, il existe 2 types:
   1.1) Terminaison:
   - Assurer que la transformation termine après un nombre des étapes finie.
   - Assurer l'existence d'un modèle cible.

   1.2)Confluence:
   - l'ordre d'application des règles de transformation n'est pas important.
   - Assurer que la transformation toujours produire le même résultat.
2. Propriétés liées aux modelés source et cible, il existe trois types:
   2.1) Conformité:
   - Un modèle est valide s'il est conforme à son méta modèle.
   - Est vérifié automatiquement dans des Framework de modélisation.

   2.2) Relation syntaxique :
   - Certains éléments(source)seront transformé en d'autres éléments (cible)

   2.3) Relation sémantique :
   - Liée la signification des deux modèles (source et cible)
   - Construire les LTS de la sémantique des deux modèles.

Méthode : Model Checker

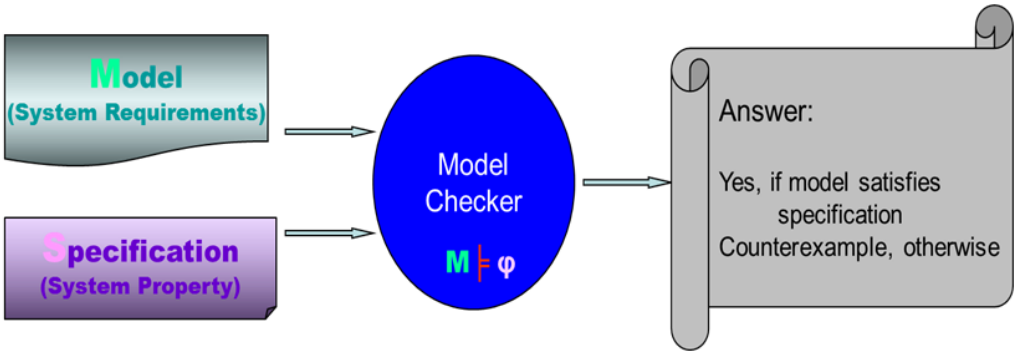


Figure 4.12 : Principe du Model Checker

Le principe de fonctionnement du modèle Checker consiste en trois phases. La première phase est la modélisation du comportement du système. La deuxième phase est la spécification des propriétés attendues du système dans la logique temporel LTL ou CTL. Finalement, il répond avec oui si la propriété est satisfaite sinon un contre-exemple est généré automatiquement qui montre un scénario possible d'erreur. Cependant,modèle Checker souffre de problème de l'explosion combinatoire de graphe d'états.

Méthode : Démonstrateur de théorèmes

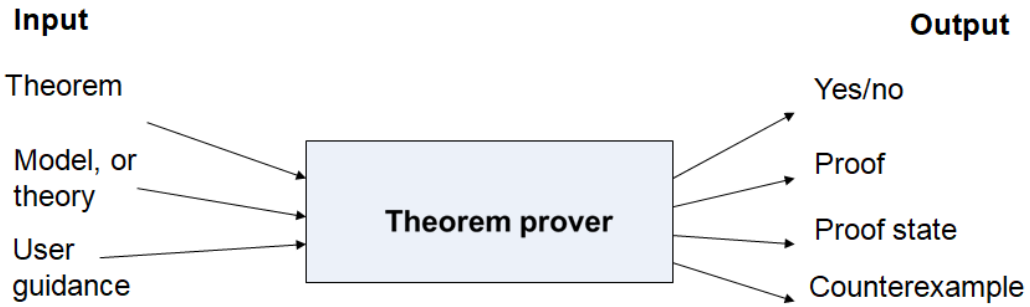


Figure 4.14 : Principe de démonstrateurs de théorèmes

Dans le cas du démonstrateur de théorèmes, le système et les propriétés attendues sont exprimés en utilisant des descriptions dans une logique mathématique par exemple Coq utilise le langage Galina. La preuve d'une propriété consiste en une ou plusieurs étapes et chaque étape peut faire appel aux axiomes, aux règles, aux définitions ou aux lemmes qui ont été déjà prouvés. L'avantage est qu'ils peuvent spécifier et effectuer des preuves sur des systèmes infinis à travers des techniques comme l'induction. Cependant,les démonstrateurs de théorèmes sont très coûteux, nécessitent des compétences avancées en matière de preuves ainsi que parfois de l'interaction de l'utilisateur pour la construction d'une preuve.

4.3. Approches de Vérification formelle de transformations de modèles

Méthode : Vérification d'une transformation basée sur le Model checker

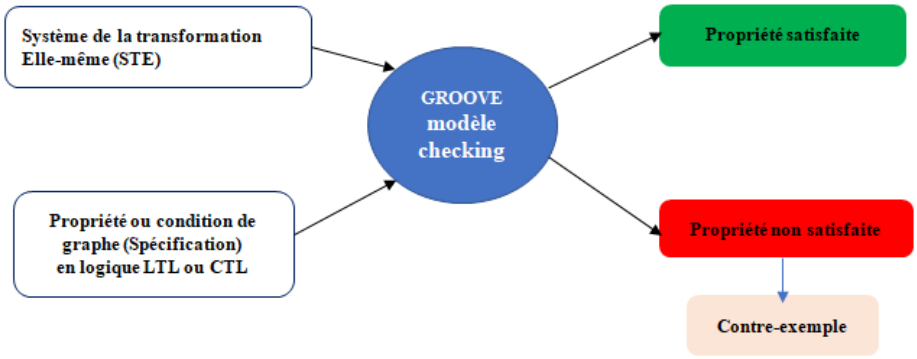


Figure 4.13 : Approche GROOVE modèle-checking

Cette approche [19]* permet de vérifier les transformations de graphes en utilisant l'outil GROOVE et son modèle checking. GROOVE considère la transformation comme un Système de Transition étiquetée (STE). Cette approche consiste en trois parties:

1. GROOVE génère le STE de la transformation contenant un ensemble des états et des transitions.
    - Chaque transition connecte deux états et est étiquetée avec le nom de la règle appliquée.
    - Chaque état contient les propriétés ou les conditions de graphes qui sont valides.
2. L'utilisateur exprime les conditions à vérifier avec la logique LTL ou CTL.
3. Modèle checking répond avec oui si le système de la transformation (STE) satisfait la propriété (la spécification) sinon il répond avec non et génère un contre-exemple.

Méthode : Vérification d'une transformation basée sur le démonstrateur de théorèmes

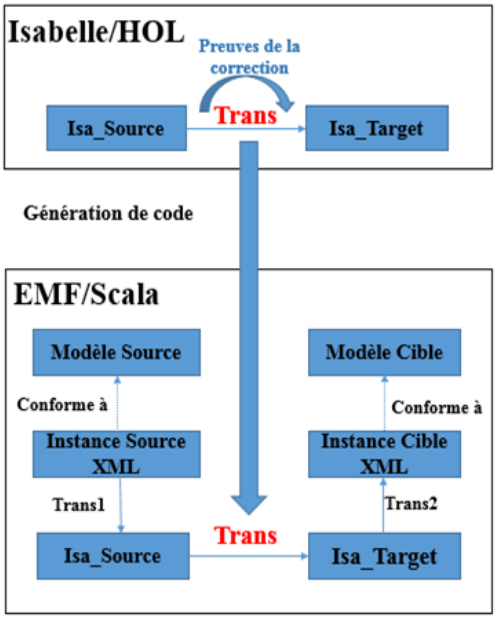


Figure 4.15 :Approche basée sur le démonstrateur de théorèmes

Cette approche [21]* considère la transformation comme une fonction mathématique qui compose de plusieurs fonction récursives. Elle consiste en deux parties:

- En Isabelle/HOL:

1. Description de modèle source
2. Description de modèle cible
3. Description de la transformation
4. Spécification des propriétés attendues
5. Preuve de correction des propriétés attendues

- En EMF/Scala:

1. Définition des méta-modèles en utilisant EMF.
2. Génération de code Scala de la fonction Trans.
3. Définition des fonctions Trans1 et Trans2.
4. Composition des fonctions :Trans2 (Trans (Trans1(modèle source)))-->modèle cible.

5. Conclusion

Dans ce chapitre, nous avons concentré sur les méthodes formelles et leur application pour le développement de logiciels sûrs. Nous avons présenté les concepts des réseaux de Petri : la définition formelle, les représentations des RdPs, le franchissement des transitions, le graphe de marquage et les propriétés des réseaux de Petri telles que la vivacité et la bornitude. Ensuite, nous avons abordé l'intégration des méthodes formelles avec l'IDM. Enfin, nous avons présenté un aperçu sur la vérification et la validation des transformations de modèles à l'aide des techniques de vérification formelles : le Model Checker et les démonstrateurs de théorèmes.

Références

1
Kerkouche Elhillali, Modélisation multi-paradigme, Thèse de Doctorat en sciences , Université Mentouri Constantine, 2012.

12
Amrani Moussa, et al. Formal verification techniques for model transformations: A tridimensional classification. The Journal of Object Technology 14.3 (2015).

19 GROOVE home page, https://groove.ewi.utwente.nl/.

20
Jeannette M. Wing, "A Specifier's Introduction to Formal Methods", Computer, vol. 23(9):8-23, 1990.

21
Meghzili Said, et al. "Verification of Model Transformations Using Isabelle/HOL and Scala." Information Systems Frontiers 21.1 (2019): 45-65.

22
Cours “ Méthodes formelles pour le parallélisme”, Saidouni Djamel Eddine, Université de Constantine 2.