

Chapitre I

1.1 Introduction

Tout est sujet de sécurité: des erreurs logicielles peuvent avoir des conséquences catastrophiques. Les défauts fatals dans le contrôle logiciels de la fusée Ariane-5 , la mission Mars Pathfinder, et les avions de la famille Airbus a conduit à la une des journaux partout dans le monde. Des Logiciels similaires sont utilisés pour le contrôle des processus de sécurité des systèmes critiques tels que les usines chimiques, les centrales nucléaires, le contrôle du trafic et les systèmes d'alerte. De toute évidence, des bugs dans de tels logiciels, peut avoir des conséquences désastreuses. Le recours croissant des applications critiques sur le traitement des informations nous amène à affirmer que, la fiabilité des systèmes TIC (Information and Communication Technology) est une question clé dans le processus de conception du système lui-même.

L'ampleur des systèmes TIC, ainsi que leur complexité, se développe rapidement. Ces derniers ne sont plus autonomes, mais ils sont généralement intégrés dans un contexte plus large, reliant et coopérant avec plusieurs autres composants et systèmes. Ils deviennent ainsi beaucoup plus vulnérables à des erreurs, le nombre de défauts augmente exponentiellement avec le nombre de composants interactives du système. En particulier, les phénomènes tels que la concurrence et le non-déterminisme qui sont au cœur de systèmes de modélisation des systèmes en interaction, se révèlent très difficile à manipuler avec des techniques standard.

Leur complexité croissante, avec la pression de réduire considérablement le temps de développement système (time-to-market), rend la prestation des systèmes TIC de faible défaut une activité extrêmement difficile et complexe.

1.2 La Vérification Software

Les Techniques de vérification des systèmes sont appliquées à la conception de systèmes TIC d'une manière plus fiable. En bref, la vérification des systèmes est utilisée pour établir que la conception ou un produit en cours d'examen possède certaines propriétés. Les propriétés à être validés peuvent être tout à fait élémentaire, par exemple, un système ne doit jamais être en mesure de parvenir à une situation dans laquelle aucun progrès peuvent être réalisés (un scénario de blocage), et sont pour la plupart obtenu à partir du système de cahier des charges. Cette spécification prescrit ce que le système doit faire et ne pas faire, et constitue donc la base de toute activité de vérification. Un défaut est détecté une fois que le système ne remplit pas l'une des propriétés de la spécification. Le système est considéré comme être «correcte» quand il répond à toutes les propriétés obtenues à partir de sa spécification. Si que

la correction est toujours relative à un cahier des charges, et n'est pas une propriété absolue d'un système. Une vue schématique de la vérification est représenté dans la figure 1.1 [4].

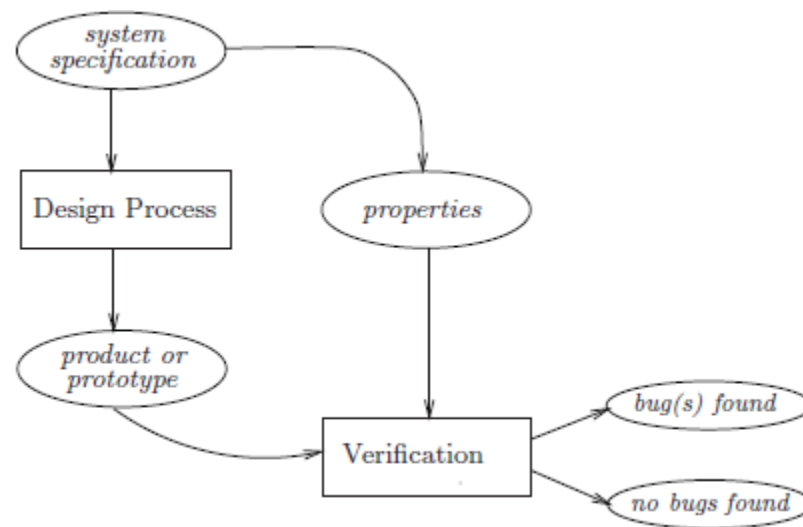


Figure 1.1 : Processus de vérification.[4]

1.2.1 La vérification :

Examiner et tester sont les deux principaux techniques utilisées dans la pratique.

1.2.1.1 L'examen : désigne l'inspection de logiciels réalisée par une équipe d'ingénieurs, de préférence qui n'a pas été impliqué dans le développement du logiciel sous revue. Le Code non compilé n'est pas exécuté, mais analysé d'une façon complètement statique.

1.2.1.2 Le test : Constitue une partie importante de tout projet de génie logiciel. Par opposition à l'examen, qui analyse le code statique sans l'exécuter, le test est une technique dynamique en fait il exécute le logiciel.

Le Test prend le morceau de logiciel en cours d'examen et fournit son code compilé avec les entrées, appelés tests. La correction est donc déterminée en forçant le logiciel à traverser un ensemble de chemins d'exécution, des séquences d'instructions de code représentent une exécution du logiciel. Sur la base des observations au cours de l'exécution du test, les résultats produits réellement par le logiciel sont comparées aux outputs tels que documentées dans le cahier des charges. Bien que la génération et l'exécution du test puissent être en partie automatisées, la comparaison est généralement réalisée par des êtres humains.

Le principal avantage des tests, c'est qu'elle peut être appliquée à toutes sortes de logiciels, allant de logiciels d'application (par exemple, l'e-business logiciels) aux compilateurs et systèmes d'exploitation.

Comme l'expérimentation exhaustive de tous les chemins d'exécution est pratiquement infaisable, en pratique, seule une petite partie de ces chemins est traitée. Les Tests ne peuvent donc jamais être complets. C'est-à-dire, les tests peuvent montrer la présence d'erreurs, cependant pas leur absence. Un autre problème avec les tests est de déterminer quand s'arrêter [5][6].

Les examens et les tests détectent différentes classes de défauts à différents stades du cycle de développement. Ils sont donc souvent utilisés ensemble.

Pour augmenter la fiabilité des logiciels, ces approches de vérification de logiciels sont complétés avec des techniques d'amélioration des processus logiciels, de la conception structurée et les méthodes de spécification (telles que l'Unified Modeling Language UML), et l'utilisation de la version et la configuration systèmes de contrôle de gestion. Les Techniques formelles sont utilisées, sous une forme ou une autre.

1.2.2 Détecter des erreurs logicielles:

Le plus tôt sera le mieux. Il est d'une grande importance pour localiser les bogues du logiciel. Le slogan est: le plus tôt sera le mieux. Les coûts de réparation d'une faille logicielle lors de l'entretien sont environ 500 fois supérieurs à celui d'une correction dans une phase de conception initiale (voir Figure 1.2). La vérification du système devrait donc prendre place au début de l'étape du processus de conception.

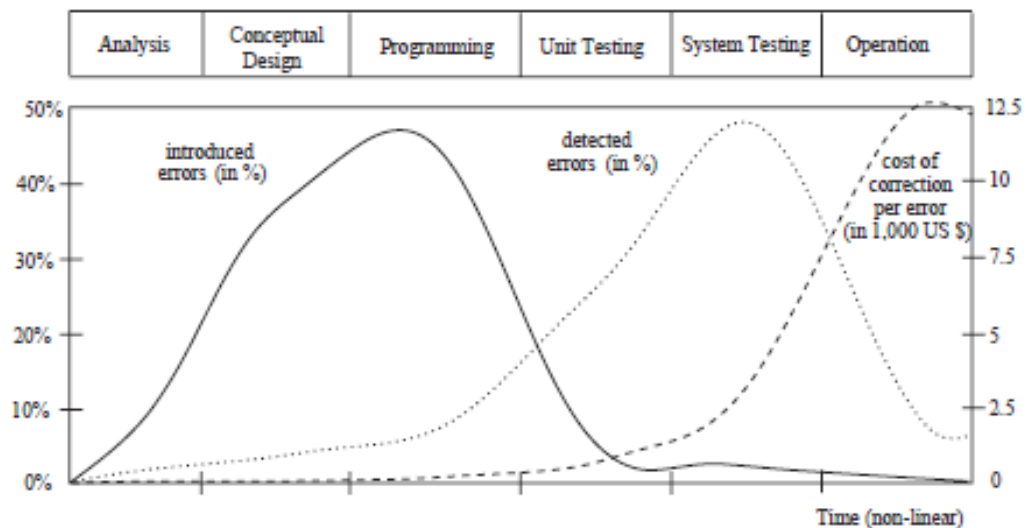


Figure 1.2 : cycle de vie du logiciel et le coût de détection, et réparation des erreurs [4]

1.3 Vérification formelle:

Les méthodes formelle peuvent être considérées comme "les mathématiques appliquées pour la modélisation et l'analyse des systèmes TIC . Leur but est d'établir la correction du système avec une rigueur mathématique. Leur grand potentiel a conduit à une utilisation croissante par les ingénieurs des méthodes formelles pour la vérification des systèmes logiciels et matériels complexes. En outre, les méthodes formelles sont l'une des techniques de vérification «fortement recommandées» pour le développement de systèmes logiciels safety-critical.

Selon, Les meilleures pratiques standards de l'IEC (International Electrotechnical Commission) et les normes de l'ESA (Agence spatiale européenne). Le rapport qui en résulte d'une enquête par la FAA (Federal Aviation Authority) et la NASA (National Aeronautics and Space Administration) sur l'utilisation des méthodes formelles conclut que : Les méthodes formelles devraient faire partie de l'éducation de chaque chercheur en informatique et ingénieur en logiciel, tout comme la branche des Maths appliquées est une partie nécessaire de l'éducation de tous les autres ingénieurs.[4]

1.4 La Vérification à base de modèles

Les Techniques de vérification à base de modèle sont basées sur des modèles décrivant le comportement possible du système d'une manière mathématiquement précise et sans ambiguïté. Il s'avère qu'avant toute forme de vérification, la modélisation précise des systèmes conduit souvent à la découverte d'incomplétude, des ambiguïtés, et des incohérences dans les spécifications du système informel [7].

Ces problèmes ne sont généralement découverts qu'à un stade beaucoup plus avancé de la conception. Les modèles systèmes sont accompagnés par des algorithmes qui systématiquement explorent tous les états du modèle. Ceci fournit la base pour toute une gamme de techniques de vérification allant d'une exploration exhaustive (model checking) aux expériences avec un ensemble restrictif de scénarios dans le modèle (simulation), ou dans la réalité (les tests).

En raison des améliorations incessantes des algorithmes et les structures de données sous-jacentes, ainsi que la disponibilité des ordinateurs plus rapides et avec plus de capacité mémoire, les techniques basées sur des modèles, alors qu'il y a une décennie étaient seulement destinées pour des exemples très simples sont aujourd'hui applicables aux dessins réalistes. Comme le point départ de ces techniques est un modèle du système sous étude, on peut dire que: Toute vérification utilisant des techniques basées sur le modèle est seulement aussi meilleur que le modèle du système lui-même.

Dans la conception des systèmes logiciels et matériels complexes, plus de temps et d'efforts sont consacrés à la vérification qu'à la construction. Les techniques sont orientées à chercher à réduire et faciliter la vérification des efforts tout en augmentant leur couverture. Les méthodes formelles offrent un grand potentiel pour obtenir une intégration précoce de la vérification dans le processus de conception, de fournir une vérification plus efficace techniques, et pour réduire le temps de vérification.

Les propriétés typiques qui peuvent être vérifiées en utilisant le model checking sont de nature qualitative : (Est le résultat généré par OK?),

Que le système peut atteindre une situation de blocage, par exemple, lorsque deux programmes concurrents sont en attente pour l'autre et donc l'arrêt de l'ensemble du système? Mais également des propriétés de synchronisation peuvent être vérifiées: Peut se produire dans une impasse 1 heure après qu'un système soit réinitialisé?, ou, est une réponse toujours reçu dans les 8 minutes?

Le model-checking nécessite un énoncé précis et sans ambiguïté des propriétés à examiner. Concevoir un modèle de système précis, cette étape conduit souvent à la découverte de plusieurs ambiguïtés et les incohérences dans la documentation informelle.

Le modèle de système est généralement généré automatiquement à partir d'une description du modèle qui est spécifiée dans quelque dialecte appropriée des langages de programmation comme le C ou Java ou des langages de description du hard tels que Verilog ou VHDL. Notons que la spécification de propriété prescrit ce que le système doit faire et ce qu'il ne devrait pas faire, alors que le modèle de description aborde comment le système se comporte. Le vérificateur de modèle examine tous les états pertinents du système pour vérifier si elles répondent à la propriété désirée. Si un Etat est rencontrée qui viole la propriété en cours d'examen, le vérificateur de modèle fournit un contre-exemple qui indique comment le modèle pourrait atteindre l'état désiré. Le contre-exemple décrit un chemin d'exécution qui mène de l'état initial du système à un état qui viole la propriété en cours de vérification. Avec l'aide d'un simulateur, l'utilisateur peut rejouer ce scénario, et de cette manière obtenir des informations de débogage utiles, et adapter le modèle (ou le la propriété) en conséquence.

1.5 Le model checking

Le Model Checking est une technique de vérification automatique des systèmes informatiques (logiciels, circuits logiques, protocoles de communications). Il s'agit de tester algorithmiquement si un modèle donné, le système lui-même ou une abstraction du système, satisfait une spécification logique, généralement formulée en termes de logique temporelle. [8]

1.5.1 L'approche du Model Checking

Le model checker est une approche algorithmique qui repose sur l'idée simple : si l'on énumère toutes les situations possibles dans lesquelles peut se trouver le système, on saura s'assurer qu'aucune de ces situations n'est en contradiction avec les comportements que l'on désirait [9].

Le model-checking est une technique de vérification qui explore tous les états du système possibles dans une manière force-brute. Semblable à un programme d'échecs informatique qui vérifie les mouvements possibles. Si aucune erreur n'est détectée, l'utilisateur peut affiner sa description du modèle. Un model checker, est l'outil logiciel qui effectue la vérification de modèles, examine la mesure du possible des scénarios de systèmes d'une manière systématique. De cette façon, il peut être démontré que le modèle d'un système donné satisfait vraiment une certaine propriété. Même des erreurs subtiles qui n'ont pas été découvertes en utilisant l'émulation, les tests et la simulation peuvent potentiellement être révélées à l'aide de model checking [4]. L'idée est qu'en s'assurant que le modèle satisfait suffisamment les propriétés du système, nous augmentons notre confiance dans la correction du modèle [10]. Le principe du model checking est illustré à la figure 1.3

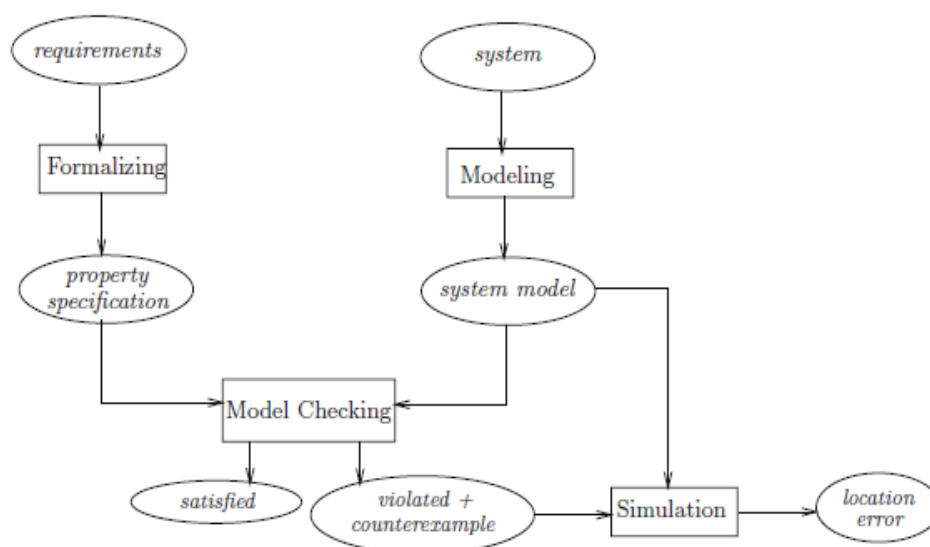


Figure 1.3 : L'approche de model-checking.[4]

.5.2 Les Caractéristiques du Model Checking

Le model-checking est une technique automatisée qui, compte tenu d'un modèle à états finis d'un système et une propriété formelle, vérifie systématiquement si cette propriété est vérifiée (un état input) pour le modèle.

1.5.2.1 Le processus du model checking

. En appliquant le model checking pour une conception, les différentes phases suivantes peuvent être distinguées:

1.5.2.1.1 Phase de modélisation:

- Modeler le système à l'étude en utilisant le langage de description de modèle du model-checker utilisé.
- Comme un test de cohérence première et une évaluation rapide, le modèle effectue quelques simulations.
- Formaliser la propriété à vérifiée à l'aide du langage de spécification de propriété.

1.5.2.1.2 **Phase d'Exécution:** exécuter le vérificateur de modèle pour vérifier la validité de la propriété dans le modèle de système.

1.5.2.1.3 Phase d'analyse :

- La propriété satisfaite? → vérifier la propriété suivante (le cas échéant);
- La propriété violée? →
 1. analyser les contre-exemples générés par simulation;
 2. affiner le modèle, la conception, ou les propriétés;
 3. répéter toute la procédure.
- Hors de la mémoire? → essayer de réduire le modèle et essayez à nouveau.

En plus de ces étapes, la vérification doit être toute entière, planifiée, administrée, et organisée. C'est ce qu'on appelle organisme de vérification.

1.5.3 La Modélisation

Les entrées préalables à la vérification de modèle est un modèle du système sous examen et une caractérisation formelle de la propriété à vérifier.

Les Modèles de systèmes décrivent le comportement des systèmes d'une manière précise et sans ambiguïté. Ils sont pour la plupart exprimé en utilisant des automates finis, constitué d'un ensemble fini d'États et un ensemble de transitions. Les Etats comprennent des informations sur les valeurs actuelles des variables, l'instruction exécutée précédemment (par exemple, un compteur de programme), et analogue.

Les Transitions décrivent comment le système évolue d'un état à un autre. pour les systèmes réalistes, automates finis sont décrits en utilisant un langage de description de modèle comme un dialecte approprié / extension de C, Java, VHDL (*VHSIC Hardware Description Langage*), ou le. Comme Modélisation des systèmes, en particulier les concurrentes, au niveau d'abstraction le droit est plutôt complexe et il est vraiment un art.

Afin d'améliorer la qualité du modèle, une simulation avant la vérification du modèle peut avoir lieu. La simulation peut être utilisée efficacement pour se débarrasser de la plus simple catégorie d'erreurs de modélisation. L'élimination de ces erreurs simples avant toute forme de contrôle approfondi peut réduire l'effort de vérification coûteuse et chronophage.

Pour effectuer une vérification rigoureuse possible, les propriétés doivent être décrites de manière précise et non ambiguë. Cela se fait habituellement en utilisant un langage de spécification de propriété. L'accent est mise en particulier sur l'utilisation d'une logique temporelle comme langage de spécification de propriétés, une forme de logique modale qui est approprié pour définir les propriétés pertinentes des systèmes TIC.

1.5.3.1 Les propriétés pertinentes

En termes de logique mathématique, on vérifie que la description du système est un modèle d'une formule logique temporelle. C'est ce qui explique le terme "model checking". La logique temporelle est essentiellement une extension de la logique propositionnelle classique avec les opérateurs qui se réfèrent au comportement de systèmes dans le temps.

Elle permet la spécification d'un large éventail de propriétés pertinentes du système telles que :

- La correction fonctionnelle (le système ne faire ce qu'il est censé à faire?),
- L'accessibilité (est-il possible de se retrouver dans un état de blocage?),
- La sécurité («quelque chose mauvaise n'arrive jamais »),

- La vivacité (« quelque chose de bon finira par arriver »),
- L'équité (le fait, sous certaines conditions, un événement se produit à plusieurs reprises?), et en temps réel des propriétés (c'est le système agissant dans le temps?).

1.5.3.2 La Validation

Bien que les étapes mentionnées ci-dessus sont souvent bien compris, en pratique, il peut être un délicat et grave problème de juger si l'énoncé du problème formalisé (modèle + propriétés) est une description adéquate du problème de vérification réelle. Ceci est également connu sous le nom de problème de validation. La complexité du système impliqué, ainsi que le manque de précision de la spécification informelle de la fonctionnalité du système peut avérer difficile de répondre à cette question de manière satisfaisante. Vérification et validation ne doivent pas être confondues.

La vérification s'intéresse à vérifier que la conception satisfait aux exigences qui ont été identifiés, à savoir "vérifier que nous construisons la bonne chose". Dans la validation, il est à vérifier si le modèle formel est compatible avec la conception informelle "vérifier que nous vérifions la bonne chose".

1.5.4 L'exécution du Modèle-checker

Le modèle-checker doit d'abord être initialisé d'une manière appropriée en fixant les différentes options et les directives qui peuvent être utilisées pour réaliser la vérification exhaustive. Par la suite, le model checking effective aura lieu. C'est essentiellement une approche purement algorithmique dans laquelle la validité de la propriété en cours d'examen est vérifiée dans tous les Etats du modèle du système.

1.5.5 L'Analyse

L'Analyse des résultats qui sont essentiellement en forme de trois réponses possibles: la propriété spécifiée est soit valide dans le modèle donné ou non, ou le modèle se révèle être trop grand pour s'adapter à l'intérieur des limites physiques de la mémoire de l'ordinateur.

Dans le cas où la propriété est valide, la propriété suivante peut être vérifiée, ou, en cas toutes les propriétés ont été vérifiées, le modèle est conclu comme possédé toutes les propriétés souhaitées.

Chaque fois qu'une propriété est falsifiée, le résultat négatif peut avoir différentes causes. Ça peut être une erreur de modélisation : le modèle ne reflète pas la conception du système. Cela implique une correction du modèle, et la vérification doit être redémarré avec le modèle amélioré.

Cette revérification comprend la vérification de ces propriétés qui ont été vérifiées précédemment sur le modèle erroné et dont la vérification peut être invalideé par le modèle de correction!

Si l'analyse des erreurs montre qu'il n'ya pas d'écart injustifié entre la conception et son modèle, alors soit une erreur de conception a été exposée, ou une erreur de la propriété a eu lieu.

Dans le cas d'une erreur de conception, la vérification est conclue avec un résultat négatif, et la conception (en collaboration avec son modèle) doit être améliorée. Il peut être le cas que lors de l'étude d'erreur exposée, on découvre que la propriété ne reflète pas l'exigence informelle qui a dû être validée. Cela implique une modification de la propriété, et une nouvelle vérification du modèle doit être effectuée.

Si le modèle n'a pas changé, aucune revérification des propriétés déjà vérifiées précédemment n'aura lieu. La conception est vérifiée si et seulement si toutes les propriétés ont été vérifiées par respecter à un modèle valide.

Chaque fois que le modèle est trop grand pour être manipulés - espaces d'états de la vie réelle des systèmes peut être de plusieurs ordres de grandeur plus grand que ce qui peut être stocké par les mémoires actuellement disponibles - Il existe plusieurs façons de procéder.

Une autre possibilité consiste à appliquer des techniques qui tenteraient d'exploiter les régularités implicites dans la structure du modèle. Des exemples de ces techniques sont les représentation de l'espace d'états en utilisant des techniques symboliques tels que les diagrammes de décision binaires ou la réduction d'ordre partiel.

Alternativement, les abstractions rigoureuses du modèle du système complet sont utilisés. Ces abstractions doivent préserver la validité (non-) des propriétés qui ont besoin d'être vérifiées. Souvent, les abstractions peuvent être obtenues, qui soient suffisamment petites pour une seule propriété. Dans ce cas, les abstractions différentes doivent être prises pour le modèle considéré.

Une autre façon de traiter avec des espaces étatiques qui sont trop grands est de produire un résultat de vérification avec une certaine précision. Les méthodes de vérification probabilistes explorer une partie seulement de l'espace d'état en faisant un sacrifice (souvent négligeable) dans la vérification la couverture.

1.5.6 Organisation de vérification

L'ensemble de processus model-checking devraient être bien organisé, bien structuré, et bien planifié. Les applications industrielles de model checking ont fourni la preuve que l'utilisation de la version et la gestion de configuration est particulièrement pertinente.

Pendant le processus de vérification, par exemple, les descriptions des différents modèles en fait décrivent les différentes parties du système, différentes versions des modèles de vérification sont disponibles (par exemple, en raison de l'abstraction), et beaucoup de paramètres de vérification (par exemple, model-checking) les options et les résultats (des traces de diagnostic, des statistiques) sont disponibles.

Cette information doit être documentée et conservée très soigneusement afin de gérer un processus model-checking pratique et de permettre la reproduction des expériences effectuées.

1.5.7 Le succès du model checking

Le model-checking a été appliquée avec succès à plusieurs systèmes TIC et leurs applications. Par exemple, les blocages ont été détectés dans les systèmes de réservation des compagnies aériennes en ligne, les protocoles du commerce électronique moderne ont été vérifiés, et plusieurs études de droit international de normalisation IEEE pour la communication interne des appareils domestiques ont conduit à des améliorations significatives des spécifications du système.

- Cinq erreurs récemment inconnues ont été identifiés dans un module d'exécution de la commande satellite Deep Space 1 .dans un cas l'identification d'un défaut de conception majeur.
- Un bug identique à celui découvert par model checking échappé d'essai et a provoqué une impasse au cours d'une expérience de vol 96.000.000 km à partir de la terre.
- Aux Pays-Bas, le model checking a révélé plusieurs défauts de conception graves dans le logiciel de contrôle d'un barrage anti-tempête qui protège le port principal de Rotterdam contre l'inondation.

1.5.8 Les Points forts et faibles

1.5.8.1 Les points forts du model checking

- Il s'agit d'une approche de vérification générale qui est applicable à un large éventail d'applications telles que les systèmes embarqués, génie logiciel, et la conception du matériel.

- Il prend en charge la vérification partielle, c'est à dire, les propriétés peuvent être contrôlés individuellement, ce qui permettant de se concentrer sur les propriétés essentielles d'abord. Aucune spécification complète des besoins n'est nécessaire.
- Il n'est pas vulnérable à la probabilité qu'une erreur est exposée, ce qui en contraste avec le test et la simulation qui visent à retracer les défauts les plus probables.
- Il fournit des informations de diagnostic dans le cas où une propriété est invalidée, ce qui est très utile à des fins de débogage.
- Il s'agit d'une technologie "bouton-poussoir" potentielle, l'utilisation de model checking ne nécessite ni un degré élevé d'interaction utilisateur, ni un haut degré d'expertise.
- Il bénéficie d'un intérêt en croissance rapide de l'industrie; plusieurs sociétés de matériel ont lancé leurs propres laboratoires de vérification, les offres d'emploi avec les compétences requises dans le model checking apparaissent fréquemment, et les dames de modèle commerciales sont devenues disponibles.
- Il peut être facilement intégré dans les cycles de développement existants; sa courbe d'apprentissage n'est pas très raides, et des études empiriques indiquent que cela peut conduire à temps de développement court.
- Il dispose d'un fondement et mathématique solide, il est basé sur la théorie des algorithmes de graphes, des structures de données, et logiques.

1.5.8.2 Les faiblesses du model checking

- Il est surtout approprié pour les applications à contrôle intensif et moins adaptés aux applications data-intensive comme les données varient généralement sur les domaines infinis.
- Son application est soumise à des questions de décidabilité; pour systèmes à états infinis, ou un raisonnement sur les types de données abstraits (qui exige des logiques indécidables ou semi-décidable), le model checking n'est en général pas effectivement computable.
- Il vérifie un modèle de système, et non pas le système actuel (produit ou d'un prototype) lui-même; un résultat obtenu est donc aussi bon que le modèle de système l'est. Des techniques complémentaires, comme les tests, sont nécessaires pour trouver des défauts de fabrication (pour le matériel) ou des erreurs de codage (pour les logiciels).

- Il ne vérifie que les exigences énoncées, c'est à dire, il n'existe aucune garantie d'exhaustivité. La validité des propriétés qui ne sont pas vérifiées ne peuvent pas être jugés.
- Il souffre du problème d'explosion de l'espace-état, c'est à dire, le nombre d'états nécessaires à modéliser avec précision pour le système peut facilement dépasser la capacité mémoire disponible de l'ordinateur. Malgré le développement de plusieurs méthodes très efficaces pour lutter contre ce problème, des modèles de systèmes réalistes peuvent-être encore trop grand pour tenir en mémoire.
- Son utilisation nécessite un certain expertise dans la recherche des abstractions appropriées pour obtenir de plus réduit modèles de systèmes et de spécifier les propriétés dans le formalisme logique utilisé.
- Il n'est pas garanti d'obtenir des résultats corrects: comme avec n'importe quel outil, un vérificateur de modèle peut contiennent des défauts logiciels.
- Il ne permet pas de vérifier les généralisations: en général, la vérification des systèmes avec un nombre arbitraire de composants ou systèmes paramétrées, ne peut pas être traitée. Modèle checker peut, cependant, suggérer les résultats pour les paramètres arbitraires qui peuvent être vérifiées en utilisant des assistants de preuve.

Nous croyons qu'on ne peut jamais atteindre une correction absolue garantie pour les systèmes de taille réaliste. Malgré les limitations ci-dessus, nous concluons que : le Model checker est une technique efficace pour exposer les erreurs potentielles d'une conception. Le model checker peut augmenter le degré de confiance dans la conception du système.

1.5.9 Outils de Model Cheking

Java PathFinder : est un outil pour la vérification de programmes Java, sur la base du bytecode Java. Par conséquent, il peut vérifier n'importe quelle langue qui peut être compilé en bytecode Java. L'utilisateur peut sélectionner différents algorithmes selon laquelle Java PathFinder explore l'espace d'état. [2]

MoonWalker : est un vérificateur de modèle développé sur la plate-forme Mono. MoonWalker est un programme pour détecter automatiquement les erreurs dans les programmes de bytecode CIL, les applications écrites pour savoir la plate-forme. NET. La version actuelle de MoonWalker est capable de trouver les blocages et les violations des assertions programmes CIL, générés avec Mono compilateur C #. La conception de MoonWalker est inspiré par et sur la base du PathFinder Java. [11]

UPPAAL : est un environnement de l'outil intégré pour la modélisation, la simulation (via la simulation graphique) et, à la vérification des systèmes temps-réel embarqués. Domaines d'application typiques de UPPAAL comprend des contrôleurs temps réel et des protocoles de communication en particulier, ceux où les aspects de synchronisation sont essentielles. [12]

PRISM : est un probabiliste model checker, un outil de modélisation formelle et d'analyse des systèmes qui présentent un comportement aléatoire ou probabiliste. Il prend en charge un large éventail de modèles probabilistes et a été utilisé pour analyser les systèmes de nombreux domaines d'application différents, y compris les protocoles de communication et multimédia, Algorithmes distribués, protocoles de sécurité, les systèmes biologiques et de nombreux autres. [13]

SPIN : Spin est un outil logiciel open-source, utilisé par des milliers de personnes dans le monde, qui peut être utilisé pour la vérification formelle des systèmes logiciels distribués. L'outil a été développé aux Bell Labs dans le groupe d'origine Unix du Centre de Calcul de recherche en sciences, en commençant en 1980. Le logiciel est disponible librement depuis 1991, et continue à évoluer au rythme des nouveaux développements dans le domaine [14].

1.5.10 SPIN Model Cheking

SPIN (Simple Promela interprète) est un vérificateur de modèle développé par Gerard J. Holzmann. Il a depuis été largement utilisé dans les industries qui construisent des systèmes critiques et concurrents.

Le Spin dispose de deux modes principales de fonctionnement: la simulation et la vérification [15]. La vérification requière en effet une recherche exhaustive, contrairement à la simulation, de ce fait la simulation peut manipuler des espaces d'états plus grands. Tout comme toutes les autres formes des tests, la simulation peut seulement indiquer des erreurs et ne peut jamais démontrer leur l'absence. Elle est certainement utile, mais nous allons nous concentrer sur la vérification.

En Spin, la vérification est subdivisée en deux aspects: la sécurité (rien ne se passe mal) et la vivacité (éventuellement quelque chose de bon arrive). Tout comme avec la démonstration de théorèmes, la vérification de la sécurité est généralement fait en premier, Elle est plus importante, et elle est plus facile.

Un modèle à être vérifiée par Spin est déclarée au moyen d'un script dans le Protocole Métalangage Promela [16]. Un tel script définit un certain nombre de processus ou threads, qui s'exécutent en même temps (par l'entrelacement de la sémantique) et qui n'ont besoin de se terminés! donc, la première source de non-déterminisme est le choix d'un entrelacement arbitraire de processus. Les processus sont définis avec le mot-clé proctype.

1.6 Conclusion

La vérification est certes une phase très importante dans le développement d'un logiciel et même du matériel. De la vérification traditionnelle à la vérification formelle jusqu'au model-cheking, un trajet intéressant et riche qui a abouti à des outils de vérification puissant tel le Spin model-checker.

Le Spin est populaire et très utilisé dans la vérification à base de modèles. L'objectif de ce projet est de profiter de cet outil pour vérifier les programmes écrits en C. Seulement le Spin model-checker est dédié à la vérification de modèles. Néant-moins une nouvelle tendance commence a émergé, celle d'adapter les programmes écrits dans des langages de programmations modernes pour qu'on puisse leur appliquer le Spin model checker.

Dans le chapitre suivant on va discuter cette nouvelle discipline.