Les méthodes formelles pour la vérification des contrats intelligents

Moez Krichen

Les méthodes formelles pour la vérification des contrats intelligents

Moez Krichen

Laboratoire ReDCAD, Université de Sfax, Tunisie
moez.krichen@redcad.org

Résumé. D'une part, les contrats intelligents sont des contrats numériques qui s'appuient sur la technologie Blockchain pour rendre leurs termes et conditions d'exécution infalsifiables. Le but d'un contrat intelligent est d'éliminer le besoin d'un intermédiaire dans les affaires et le commerce entre les participants anonymes et identifiés. Depuis 2016, les contrats intelligents gagnent du terrain dans divers domaines, notamment la gestion publique, la chaîne d'approvisionnement, l'énergie, la finance, la communication et la santé. Quiconque interagit avec un contrat intelligent après son lancement sur le système blockchain sera en danger s'il contient des vulnérabilités ou des défauts. D'autre part, les méthodes formelles sont des techniques mathématiques permettant de modéliser, de concevoir et de tester des systèmes logiciels et matériels pour s'assurer qu'ils sont construits correctement. Dans cet article, nous passons en revue les méthodes formelles de pointe appliquées au domaine émergent de la spécification et de la vérification des contrats intelligents pour minimiser le risque d'apparition de fautes et de bogues et éviter les éventuels coûts qui en résultent.

1 Introduction

Les contrats intelligents sont les technologies qui façonneront la façon dont les humains interagissent à l'avenir, et ils ont déjà commencé à exploser en popularité. Les contrats intelligents sont un code informatique qui exécute automatiquement tout ou partie d'un contrat et est stocké sur une plateforme basée sur la blockchain. Le code peut soit être la seule représentation de l'accord des parties, soit compléter un contrat textuel typique en réalisant certaines clauses, telles que des transferts d'argent de la partie X à la partie Y. Parce que le code est dupliqué sur plusieurs nœuds de une blockchain, elle bénéficie de la permanence, de la sécurité et de l'immuabilité de la blockchain. Les contrats intelligents sont désormais mieux adaptés pour automatiser deux types de « transactions » présentes dans de nombreux contrats : (1) assurer le paiement des sommes lors de la survenance d'événements déclencheurs spécifiés et (2) appliquer des sanctions pécuniaires si certaines conditions objectives ne sont pas remplies. L'intervention humaine, par exemple par l'intermédiaire d'un tiers de confiance ou même du système judiciaire, n'est pas nécessaire une fois que le contrat intelligent a été lancé

et est opérationnel dans chaque cas, ce qui réduit les coûts d'exécution et d'application du processus contractuel. Les contrats intelligents, par exemple, pourraient réduire ce que l'on appelle les écarts entre l'approvisionnement et le paiement. Lorsqu'un produit est livré à un entrepôt et scanné, un contrat intelligent peut immédiatement demander les approbations nécessaires et, une fois accordées, envoyer de l'argent de l'acheteur au vendeur. Les vendeurs seraient payés plus rapidement et n'auraient plus besoin de se livrer à des relances, tandis que les frais de compte à payer des acheteurs seraient réduits. Cela pourrait influencer les besoins en fonds de roulement et faciliter les opérations financières pour les deux parties. Du côté de l'application, un contrat intelligent peut être configuré pour désactiver l'accès à un élément connecté à Internet si le paiement n'est pas reçu. Par exemple, l'accès à certains contenus peut être automatiquement interdit si le paiement n'est pas reçu.

Des scientifiques des deux universités NUS [1] et UCL [2] ont découvert que 34 200 contrats intelligents Ethereum sur environ un million sont vulnérables aux attaques, dont 2 365 ne prennent que 10 quelques secondes à pirater à l'aide d'un outil qu'ils ont développé appelé MAIAN (42). Au moment de l'enquête, leur analyse a révélé que plus de 13,8 millions de dollars d'éther étaient en danger. Parce que de tels risques et même de légères failles peuvent avoir des implications catastrophiques et permanentes, travailler avec des contrats intelligents nécessite une vigilance extrême. La mesure dans laquelle un système logiciel atteint l'objectif pour lequel il a été conçu est la mesure fondamentale du succès dans le contexte commercial d'aujourd'hui. D'un autre côté, un marché plus concurrentiel s'attend souvent à une meilleure qualité, à des délais d'exécution plus courts et à des logiciels moins chers. De nombreuses organisations informatiques ont du mal à fournir des produits de haute qualité dans les délais et dans les limites de leurs contraintes budgétaires. Le nombre de défauts découverts au cours du développement a un impact significatif sur les métriques logicielles décrites ci-dessus. Le budget du projet peut être réduit si un problème est identifié tôt dans le processus de développement du programme.

Lorsque les développeurs de logiciels découvrent une erreur commise lors de la phase d'exigences lors des tests, ils doivent corriger les mauvaises exigences, vérifier toutes les conséquences via la conception et la mise en œuvre, puis tester à nouveau le produit. Des solutions rentables qui gèrent les principaux risques et donnent des preuves visibles de fiabilité sont nécessaires pour produire un certain produit logiciel et résoudre le problème de dépassement de budget (en raison d'inexactitudes dans les spécifications des exigences). Les problèmes mentionnés ci-dessus peuvent être résolus en utilisant des méthodes formelles (9). Les méthodes formelles (47) sont des approches mathématiques utilisées pour spécifier, construire et vérifier des systèmes logiciels et matériels. Un langage de spécification formel fait référence à la représentation mathématique utilisée dans les techniques formelles. Comme l'illustre la figure 1, le système étudié et l'ensemble des exigences d'intérêt sont traduits dans des langages de spécification formels appropriés. Ensuite, les techniques formelles adoptées seront chargées de vérifier si la description mathématique du système est correcte ou non concernant les propriétés mathématiques obtenues. Ces techniques sont en effet utiles à différentes

1. NUS=National University of Singapore
2. UCL=University College London

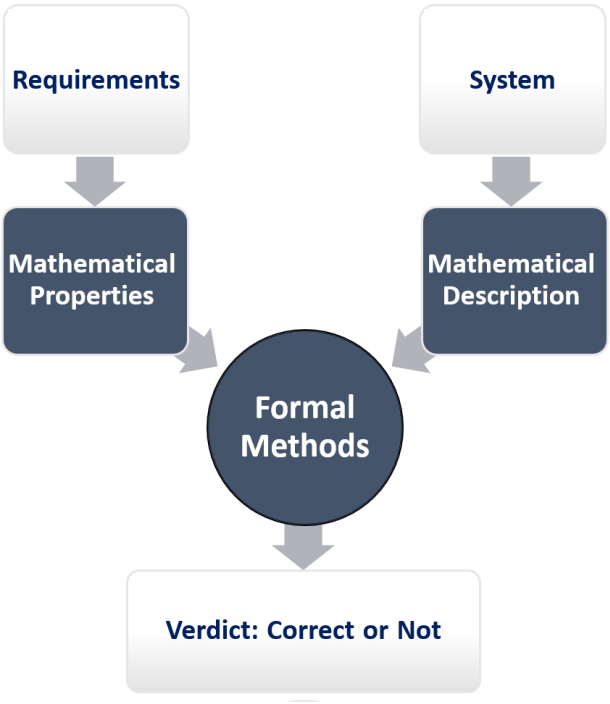

FIG. 1 – *Une illustration simplifiée du fonctionnement des méthodes formelles.*

étapes du processus de développement d'un large éventail de systèmes (43), et elles font maintenant leur chemin dans les normes de l'industrie.

Dans ce sens, plusieurs efforts ont été faits pour aider les développeurs à créer des contrats intelligents corrects et sécurisés en utilisant des méthodes formelles. Les chercheurs ont adopté différentes approches, notamment la démonstration de théorèmes (21; 4), la vérification de modèles (3; 41; 39) et les tests de logiciels (5; 31). Dans ce travail, notre objectif est de mener une revue qui traite de l'application de méthodes formelles pour vérifier l'exactitude des contrats intelligents. De plus, nous avons identifié plusieurs défis et lignes directrices de recherche futures liés à ce sujet de recherche.

Ce document est structuré comme suit. La section 2 présente des préliminaires sur les contrats intelligents et la blockchain. Par la suite, une brève introduction aux méthodes formelles est donnée dans la section 3. Ensuite, plusieurs approches de recherche appliquant des méthodes formelles pour vérifier et valider l'exactitude des contrats intelligents sont discutées dans la section 4. La section 5 énumère les limites et les défis liés à l'utilisation de contrats intelligents et formels. méthodes. Enfin, la section 6 résume les contributions des articles tout en identifiant les domaines potentiels de recherche future.

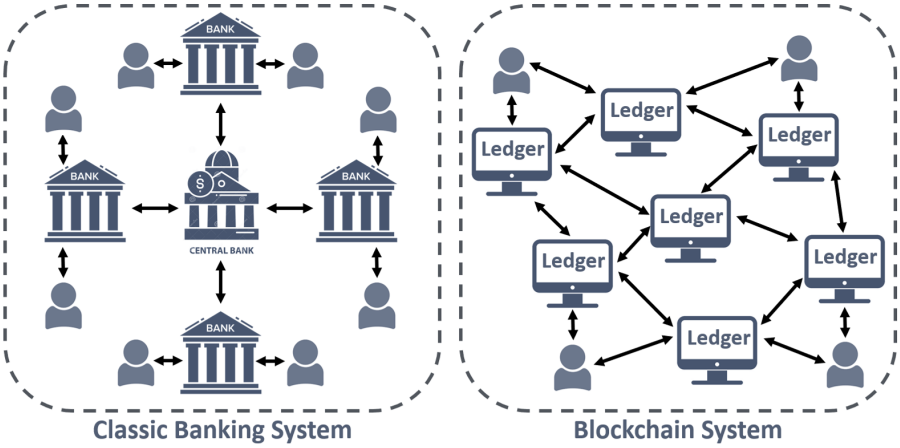

FIG. 2 – *Illustration de la différence entre le système bancaire classique (à gauche) et le système blockchain (à droite).*

2 Préliminaires sur les contrats intelligents

Dans cette section, nous proposons un bref aperçu des contrats intelligents. Étant donné que les contrats intelligents reposent principalement sur la technologie blockchain, nous commençons par un bref aperçu de la blockchain.

2.1 Qu'est-ce que la blockchain ?

Une blockchain (27; 23; 2; 22; 34) est une base de données distribuée partagée par des nœuds de réseau informatique. Une blockchain fonctionne comme une base de données, stockant les données sous forme numérique. La blockchain est bien reconnue pour son rôle vital dans le maintien d'un enregistrement sécurisé et décentralisé des transactions dans les systèmes de crypto-monnaie tels que Bitcoin. La blockchain est unique en ce sens qu'elle garantit l'exactitude et la sécurité d'un enregistrement de données tout en créant la confiance sans avoir besoin d'un tiers. La structure de données d'une blockchain diffère de celle d'une base de données traditionnelle. Une blockchain organise les données en blocs, dont chacun comprend la collecte de données. Lorsque la capacité de stockage d'un bloc est atteinte, il est fermé et lié au bloc précédent, ce qui donne une chaîne de données connue sous le nom de blockchain. Toutes les informations ultérieures incorporées après ce bloc nouvellement ajouté sont compilées dans un nouveau bloc, qui est ensuite incorporé dans la chaîne une fois celle-ci terminée. Une base de données organise les données dans des tables, tandis qu'une blockchain, comme son nom l'indique, organise les données en morceaux (blocs) qui sont enchaînés. Cette structure de données produit une chronologie des données irréversible lorsqu'elle est mise en œuvre de manière décentralisée. Lorsqu'un bloc est terminé, il devient permanent et fait partie de la chronologie. Un nouveau bloc est ajouté à la chaîne et se voit attribuer un horodatage.

2.2 Qu'est-ce qu'un contrat intelligent ?

Les contrats intelligents sont des programmes exécutés lorsque certains critères sont remplis et stockés sur une blockchain. Ils sont généralement utilisés pour informatiser l'exécution d'un consensus afin que toutes les parties prenantes puissent être certaines du résultat immédiatement, sans avoir besoin de courtiers ni de perte de temps. Ils pourraient également automatiser une charge de travail, en commençant l'étape suivante lorsque certaines conditions sont remplies. Pour faire fonctionner les contrats intelligents, de simples lignes "si/quand... alors..." sont écrites en code sur une blockchain. Lorsque certaines conditions sont remplies et validées, les opérations sont réalisées par un réseau informatique. Ces opérations peuvent inclure le transfert de fonds aux parties appropriées, l'enregistrement d'un véhicule, l'envoi d'alertes ou l'émission de billets. La blockchain est mise à jour une fois la transaction terminée. Cela indique que la transaction ne peut pas être modifiée et que les résultats ne sont accessibles qu'aux personnes autorisées.

Un contrat intelligent peut inclure autant de spécifications que nécessaire pour persuader les participants que le travail sera fait correctement. Pour définir les conditions, les participants doivent s'entendre sur la manière dont les transactions et les données associées sont exprimées sur la blockchain, sur les règles "si/quand...alors..." régissant ces transactions, inspecter toutes les exceptions possibles et concevoir un cadre pour résoudre des disputes. Un programmeur pourrait alors programmer le contrat intelligent. Cependant, les entreprises qui utilisent la blockchain pour les entreprises fournissent de plus en plus de modèles, d'interfaces Web et d'autres outils en ligne pour simplifier la création de contrats intelligents.

Smart contracts can be created using a variety of programming languages. Below, we list some examples :

- Solidity : Solidity est un langage de programmation orienté objet, à typage statique, qui a été créé pour permettre la création de contrats intelligents. [3]
- Yul : Yul peut être utilisé comme langage intermédiaire dans la compilation pour compiler des contrats intelligents Ethereum basés sur Solidity.
- Rust : Rust est un langage de contrat intelligent idéal car il n'a pas de comportement indéfini et est sûr pour le type et la mémoire. Il produit des binaires compacts.
- Vyper est un langage de programmation de type python utilisé pour produire des contrats intelligents compatibles avec la machine virtuelle Ethereum.
- Javascript : par exemple, Nebulas est une plateforme de création de contrats intelligents. Il propose une méthode pour créer des contrats intelligents avec JavaScript.

2.3 Exemples d'utilisation de contrats intelligents

Les contrats intelligents ont de nombreuses applications, notamment dans les secteurs de la santé, de la chaîne d'approvisionnement et de la finance. Voici quelques

3. L'image affiche le code Solidity pour une opération d'incrémentation et de décrémentation. De plus, il comporte une méthode qui accepte les entrées et une autre fonction qui récupère ce numéro lorsque le contrat est exécuté.

```
pragma solidity ^0.4.11;
// define new contract
contract ArithValue{
    uint number;
    function ArithValue() public {  //constructor function with default value
        number = 100;
    }
// constructor function to set new value
    function setNumber(uint theValue) public {
        number = theValue;
    }
// constructor function to fetch the new value
    function fetchNumber() public constant returns (uint) {
        return number;
    }
// constructor function to increment by one
    function incrementNumber() public {
        number=number + 1;
    }
// constructor function to decrement by one
    function decrementNumber() public {
        number=number - 1;
    }
}
```

FIG. 3 – *Un exemple de contrat intelligent écrit en Solidity (40).*

exemples :

- Secteur financier : De diverses manières, les contrats intelligents contribuent à la transformation des services financiers traditionnels. En ce qui concerne les réclamations d'assurance, ils vérifient les erreurs, les acheminent, puis envoient de l'argent à l'utilisateur si tout se vérifie. Les contrats intelligents incluent des capacités de comptabilité essentielles et éliminent le risque de falsification des registres comptables. Ils permettent également aux actionnaires de participer à un processus décisionnel transparent. Ils aident également à la compensation commerciale, le processus de transfert des paiements après que les montants des règlements commerciaux ont été calculés.
- Construction : Les contrats sont bien établis dans le secteur de la construction. Tout se fait normalement par le biais de procédures contractuelles spécifiées, de la livraison du matériel au transfert de fonds en passant par la mesure des performances. Les contrats intelligents peuvent aider le secteur de la construction à améliorer sa productivité et à réduire les délais de traitement des paiements. Le traitement des factures de construction peut également être rationalisé avec des contrats intelligents.
- Propriété : Les contrats intelligents peuvent être utilisés pour suivre la propriété d'un large éventail d'actifs, y compris des bâtiments, des terrains et même des téléphones et des montres. Les contrats intelligents sur le marché immobilier peuvent éliminer le besoin de services coûteux tels que ceux fournis par les avocats et les agents immobiliers. Les vendeurs pourront gérer la transaction entièrement par eux-mêmes grâce à cette nouvelle technologie.

— Systèmes de vote : les contrats intelligents créent un environnement de vote sécurisé moins vulnérable à la falsification. Les votes dans les contrats intelligents seraient protégés par un grand livre, ce qui les rendrait extrêmement difficiles à interpréter. De plus, les contrats intelligents peuvent augmenter la participation électorale, qui a toujours été faible en raison d'un système inefficace qui oblige les électeurs à faire la queue, à demander une pièce d'identité et à remplir des formulaires. Une fois que le vote est transmis en ligne à l'aide de contrats intelligents, il peut augmenter le nombre de répondants dans un système de vote.
— Soins de santé : les données hautement sensibles, y compris les dossiers des patients, peuvent être cryptées et partagées en toute sécurité entre les départements/instituts de recherche à l'aide de la technologie blockchain. D'un autre côté, les organisations de recherche médicale ont une énorme quantité de données à protéger, y compris les résultats des tests et les nouvelles formules de médicaments. S'ils doivent divulguer l'une de ces informations à un tiers, des contrats intelligents peuvent être utilisés pour la protéger. Une application de blockchain de contrat intelligent, en particulier, pourrait être extrêmement bénéfique pour le secteur de la recherche médicale.

2.4 Principaux avantages des contrats intelligents

Voici les avantages les plus importants de l'utilisation de contrats intelligents :

— Autonomie : L'avantage le plus important des contrats intelligents est leur automatisation. En termes simples, il n'y aura aucune interruption et aucun tiers ne pourra modifier l'accord ou la décision. Cette automatisation peut grandement aider les entreprises à automatiser certains aspects de leurs opérations.
— Sécurité : étant donné que les enregistrements de transaction blockchain sont cryptés, ils sont très difficiles à pirater. De plus, étant donné que chaque enregistrement d'un grand livre distribué est lié aux entrées suivantes et précédentes, les pirates devraient modifier toute la chaîne pour modifier n'importe quel enregistrement.
— Transparence et confiance : étant donné qu'aucune tierce partie n'est impliquée et que les journaux de transactions cryptés sont échangés entre les participants, il n'y a pas lieu de s'inquiéter de la falsification d'informations à des fins personnelles.
— Sauvegarde : Tous les documents conservés sur la blockchain sont dupliqués plusieurs fois, permettant la restauration des originaux en cas de perte de données.
— Efficacité, précision et rapidité : Le contrat est exécuté rapidement si une exigence est satisfaite. Étant donné que les contrats intelligents sont informatisés et automatisés, il n'y a pas de paperasse à traiter et pas de temps perdu à corriger les erreurs lors du remplissage des documents à la main.
— Réduction des coûts : les contrats intelligents éliminent le besoin d'intermédiaires et tous les coûts et retards associés.

2.5 Bogues possibles dans les contrats intelligents

Ci-dessous, nous présentons cinq vulnérabilités courantes des contrats intelligents (16) :

- Réentrance : cela se produit lorsqu'un contrat intelligent malveillant est intégré dans le contrat intelligent de la victime, ce qui permet au contrat malveillant d'interférer avec la première exécution du contrat de la victime.
- Manipulation du mineur : dans ces types de failles, un mineur malveillant manipule des variables en dehors de l'exécution du contrat intelligent pour manipuler le système à son avantage.
- Contrôle d'accès : le contrôle d'accès dans les contrats intelligents peut être lié à la gouvernance et à une logique cruciale telle que la frappe de jetons, le retrait d'argent, l'arrêt et la mise à niveau de contrats, le transfert de propriété, le vote sur des propositions, etc.
- Integer overflow/underflow : lorsque des opérations arithmétiques aboutissent à une sortie supérieure à la taille maximale qui lui est allouée, un débordement se produit ; lorsque la sortie est inférieure à la taille minimale, un sous-dépassement se produit.
- Déni de service distribué : les contrats intelligents qui ont été conçus de manière malveillante peuvent surcharger les nœuds de la blockchain avec de la bande passante. Cela obligerait les ressources de traitement du réseau à être détournées pour faire face à la surcharge, rendant le réseau inutilisable.

3 Méthodes formelles

3.1 Bref aperçu

Les méthodes formelles (28; 26; 36; 10; 24; 30; 37; 35) sont des outils mathématiques permettant de modéliser des systèmes complexes. Ils appliquent les mathématiques à l'ingénierie logicielle et matérielle pour renforcer la confiance dans la conception et les tests de ces systèmes. Avant la conception, des méthodes formelles sont utilisées pour décrire les fonctions d'un système avec des langages descriptifs, garantissant la fonctionnalité du système. Selon la rigueur avec laquelle le système est spécifié, des approches formelles peuvent être utilisées dans le développement. Les spécifications formelles peuvent servir de feuille de route pour déterminer les exigences. Les méthodes d'analyse formelle décrivent des fonctions qui peuvent être utilisées pour vérifier un programme. Le degré d'utilisation des procédures formelles peut varier. Ce n'est que dans le cas de systèmes essentiels et de conceptions compliquées, lorsque des erreurs peuvent entraîner des reconceptions ou des refabrications coûteuses, qu'une mise en œuvre complète de méthodes formelles est jugée d'un coût prohibitif. Les approches formelles sont fréquemment utilisées simplement pour décrire et orienter le développement de la fonction visée. C'est ce qu'on appelle les approches formelles de niveau 0 ou lite. Il est considéré comme niveau 1 lorsque des méthodes formelles sont également utilisées pour valider les fonctions. Le système complet est validé par machine à travers toutes ses fonctions au niveau 2, le plus haut niveau de procédures formelles.

3.2 Différents types

Il existe différents types de méthodes formelles. Ci-dessous, nous en énumérons quelques-uns :

— Vérification de modèle : la vérification de modèle est une méthode permettant de confirmer automatiquement l'exactitude des systèmes à états finis. Il fait référence aux algorithmes d'évaluation complète et automatique de l'espace d'état d'un système de transition pour vérifier si un modèle de système particulier satisfait à une spécification donnée.

— Démonstration de théorèmes : la démonstration de théorèmes est une technique formelle qui offre une preuve logique symbolique. L'inférence déductive est utilisée. Chaque phase de la preuve (a) présentera une prémisse ou une hypothèse ; (b) utiliser uniquement des règles d'inférence juridiques pour donner une déclaration qui découle logiquement des résultats qui ont déjà été établis. Ces preuves formelles sont souvent longues et fastidieuses. Une grande partie de la procédure peut être automatisée à l'aide de logiciels sophistiqués appelés démonstrateurs de théorèmes.

— Exécution symbolique : L'exécution symbolique est une technique pour exécuter des programmes de manière abstraite, permettant à une seule exécution abstraite de couvrir toutes les entrées potentielles du programme qui suivent le même chemin d'exécution à travers le code. Ces entrées sont gérées symboliquement tout au long de l'exécution, et le résultat est exprimé en termes de constantes qui symbolisent les valeurs d'entrée.

— Vérification d'exécution : la vérification d'exécution est une méthode de vérification simple qui examine les attributs d'un programme pendant son fonctionnement. La vérification d'exécution offre souvent une défense réactive contre les défauts ou les violations d'exactitude au moment de l'exécution. Il a le potentiel de révéler des situations sensibles que la vérification de modèle ou l'exécution symbolique ne pourraient pas atteindre en raison de l'explosion de l'état ou du chemin.

3.3 Principaux avantages

Voici les principaux avantages de l'utilisation de méthodes formelles (9) :

— Abstraction : Une spécification formelle est une description abstraite, précise et, à certains égards, complète. L'abstraction permet à un lecteur humain de saisir facilement la vue d'ensemble du produit logiciel.

— Analyse rigoureuse : des descriptions formelles sont créées à partir de plusieurs perspectives pour déterminer des aspects importants tels que le respect de critères de haut niveau ou la précision d'une conception proposée.

— Découverte précoce des défauts : les méthodes formelles peuvent être appliquées aux premiers artefacts de conception, ce qui permet une détection et une élimination précoces des défauts de conception.

— Garanties d'exactitude : les vérificateurs de modèles et autres outils d'analyse formelle analysent toutes les voies d'exécution potentielles à travers le système.

Un vérificateur de modèle détectera les défauts ou les erreurs qui peuvent exister. Les tests ne pourront pas fournir ce degré de couverture.
— Fiabilité : Dans une industrie hautement réglementée comme l'aviation, les techniques formelles fournissent le type de preuves requises. Ils montrent et prouvent la fiabilité du produit.
— Scénarios de test efficaces : nous pouvons systématiquement créer des cas de test efficaces directement à partir de la spécification formelle. Il s'agit d'une méthode peu coûteuse pour générer des scénarios de test (29; 25).

4 Méthodes formelles pour les contrats intelligents

Selon le niveau d'abstraction auquel la modélisation et l'analyse sont menées, les approches de formalisation des contrats intelligents peuvent être divisées en deux grandes catégories.

4.1 Niveau du programme

Les modèles au niveau du programme visent à fournir une vue en boîte blanche des contrats cibles. Dans ce cas, les approches adoptées analysent principalement la mise en œuvre du contrat et sont donc dépendantes de la plate-forme. De telles approches permettent un raisonnement précis sur les détails de bas niveau du processus d'exécution des contrats intelligents. Ci-dessous, nous présentons quatre types de modèles que ces approches au niveau du programme peuvent utiliser :
— Graphe de flux de contrôle : un graphe de flux de contrôle décrit chaque chemin pouvant être emprunté dans un programme pendant son exécution en utilisant la notation graphique. Les analyses de flux de contrôle/flux de données et l'exécution symbolique sont deux approches courantes qui peuvent être effectuées sur des graphes de flux de contrôle pour des applications logicielles conventionnelles. Certaines techniques reposent sur des graphes de flux de contrôle dérivés de contrats intelligents, tandis que d'autres sont construites à partir de zéro pour s'adapter aux fonctionnalités linguistiques spécifiques des programmes de contrats. Les auteurs de (11), par exemple, ont développé un outil appelé Ethainter, qui utilise l'analyse des flux d'informations pour découvrir des vulnérabilités composites. D'autres outils acceptent des extensions pour contrôler les graphes de flux avec des annotations, permettant la spécification et la validation de propriétés spécifiques (48).
— Arbre de syntaxe abstraite : Un arbre de syntaxe abstraite est une représentation formelle d'arbre qui peut être utilisée pour raisonner sur la syntaxe d'un programme donné. Les arbres de syntaxe abstraite peuvent améliorer les pré-analyses légères, maximisant l'efficacité et l'évolutivité des enquêtes en aval plus lourdes. Par exemple, les auteurs de (13) ont proposé une analyse formelle basée sur des arbres syntaxiques abstraits pour vérifier si un smart contract est conforme à la norme ERC20. Dans le même sens, les auteurs de (49) ont suggéré une méthode pour analyser les arbres de syntaxe abstraite des contrats intelligents afin de rechercher les importations de bibliothèques et les modèles de

code potentiellement dangereux. Ces méthodologies purement syntaxiques ont l'inconvénient évident de ne pas toujours respecter la sémantique opérationnelle des smart contracts, ce qui compromet la validité et la rigueur de l'analyse.

4.2 Niveau du contrat

Dans ce cas, les contrats intelligents sont considérés comme des "boîtes noires" qui acceptent les messages de transaction extérieurs et peuvent fonctionner en fonction de ceux-ci. Les modèles au niveau des contrats se concentrent principalement sur trois éléments : 1. Les contrats (fonctions publiques, effets visibles, etc.) ; 2. État de la blockchain (variables d'environnement, variables globales, etc.) ; 3. Utilisateurs (adresse du compte, solde, transactions, etc.). Les interactions entre les contrats intelligents et le monde extérieur sont bien capturées par les modèles au niveau du contrat, qui sont généralement énoncés en termes de :

- STS [4] : un STS est composé d'états et de transitions entre états qui peuvent être étiquetés avec des symboles d'un ensemble ; la même étiquette peut apparaître sur plusieurs transitions. Les comportements d'un contrat intelligent peuvent être naturellement interprétés comme un STS. En effet, certains langages de contrats intelligents utilisent même un modèle de programmation orienté état, notamment Bamboo (1) et Obsidian (14). De nombreux formalismes de transition d'état sont utilisés dans la littérature pour modéliser et vérifier les contrats intelligents :
  - CG [5] Un CG est un groupe d'agents autonomes opérant dans un environnement à états finis. Chaque joueur choisit une action à entreprendre tout au long de chaque tour dans un jeu avec des tours sans fin. Par exemple, un contrat intelligent d'échange atomique était considéré comme un CG dans (46). Pour déterminer si un contrat est juste et exact compte tenu des comportements des utilisateurs fournis, les auteurs ont utilisé le vérificateur de modèle MCK. Des objectifs similaires ont été poursuivis dans (12) ; cependant, l'analyse proposée était fondée sur la fonction objectif du modèle CG.
  - TA [6] : Un TA est un graphe fini avec un nombre fini d'emplacements et d'arêtes étiquetées qui ont été étendues avec des variables à valeurs réelles. Ce type de graphe peut être considéré comme un modèle abstrait de systèmes temporisés. Par exemple, les auteurs de (3) ont créé des modèles TA du contrat intelligent, des utilisateurs du contrat et d'un système de blockchain compacté. Pour déterminer la probabilité d'une attaque réussie contre le contrat, la vérification de modèles statistiques est utilisée pour examiner les relations entre ces éléments.
  - PN [7] : Un PN est un graphe bipartite avec deux types d'éléments : les lieux et les transitions, qui sont représentés respectivement par des cercles et des rectangles blancs. Un emplacement peut avoir un nombre illimité de jetons représentés par des cercles noirs. Par exemple, pour analyser la sécurité d'un

4. STS=State Transition System.
5. CG=Concurrent Game.
6. TA=Timed Automaton.
7. PN=Réseau de Petri.

contrat intelligent, les auteurs de (33; 17) ont utilisé des réseaux de Petri colorés et la boîte à outils cpntools. L'analyse est effectuée manuellement dans (33) alors qu'elle est effectuée automatiquement dans (17). Dans le même contexte, les auteurs de (50) ont proposé une méthode pour créer un contrat intelligent à partir d'un modèle décrit comme un processus PN et valider automatiquement le modèle conformément à des caractéristiques communes comme l'évitement des transitions mortes. [8]

- BIP [9] : BIP est une plate-forme qui génère du code exécutable qui peut être utilisé pour simuler, exécuter, explorer et déboguer des modèles formels. Il comprend des moteurs d'exécution, qui agissent comme des planificateurs d'exécution pour les codes produits. Les modèles BIP de contrat intelligent comportent généralement deux couches : les composants de la couche Comportement définissent la logique de chaque contrat intelligent comme une machine à états finis. En revanche, la couche Interaction spécifie les règles de communication entre eux. Par exemple, les auteurs de (39) ont développé un framework basé sur BIP appelé VeriSolid, qui permet de produire des contrats intelligents dont la sécurité est prouvée.
- BPMN [10] : BPMN est un organigramme représentant les étapes d'un processus métier planifié du début à la fin. Il s'agit d'un composant essentiel de la gestion des processus métier car il représente graphiquement la séquence détaillée des activités métier et des flux d'informations nécessaires à la réalisation d'un processus. Par exemple, les auteurs de (15) ont suggéré de traduire les modèles BPMN, qui décrivent les processus métier collaboratifs, en Solidity et de les déployer sur la blockchain. La traduction dans un langage formel, tel que PN, est une méthode potentielle de validation d'un modèle BPMN.
- UML [11] : UML est un langage de modélisation graphique utilisé en génie logiciel pour offrir une méthode normalisée de visualisation de la conception de systèmes. Les notations UML permettent aux utilisateurs de modéliser le flux de contrôle, le contrôle d'accès et la gestion des actifs dans les contrats intelligents produits. Par exemple, dans (19), un modèle d'état UML a été utilisé pour créer une implémentation Solidity d'un système cyber-physique.

- PA [12] : un PA est un ensemble de techniques mathématiques utilisées pour caractériser les comportements de systèmes parallèles ou distribués en tant que processus concurrents en interaction. Un PA décrit les interactions entre un ou plusieurs contrats intelligents connectés et des utilisateurs agissant simultanément. En traduisant un contrat intelligent dans l'un des formalismes algébriques de processus, la validité de ces interactions peut être vérifiée. Par exemple, les auteurs de (40) ont proposé une traduction des fonctions publiques de contrat Ethereum de Solidity pour traiter les notations représentées dans une version

8. Une transition morte est une transition qui ne peut jamais être exécutée.
9. BIP= Behavior Interaction Priority.
10. BPMN=Modèle de processus métier et notation.
11. UML=Unified Modeling Language.
12. PA=Algèbre de processus.

SAPIC π-calcul appliquée.[13] De même, les auteurs de (44) ont proposé une traduction des fonctions publiques de Solidity en processus CSP.[14] De plus, plusieurs langages de programmation spécifiques à un domaine utilisent des modèles algébriques de processus pour un contrat intelligent comme BitML (Bitcoin Modeling Language) (8; 7) , Rholang (langage d'ordre supérieur réfléchissant) (18) et SCL (32).

La logique temporelle est une branche de la logique symbolique concernée par les problèmes impliquant des propositions avec des valeurs de vérité dépendant du temps. Les spécifications au niveau du contrat sont écrites dans une variété de logiques. Les plus courantes sont la logique temporelle linéaire (LTL) (45) et la logique arborescente de calcul (CTL) (38). D'autres logiques incluent Defeasible Logic (20) et Deontic Logic (6), qui aident à la définition des obligations et des droits des parties au contrat intelligent.

5 Limites et défis

Ci-dessous, nous énumérons les principales limites et défis liés à l'utilisation de contrats intelligents et de méthodes formelles.

5.1 Limites des contrats intelligents

Les contrats intelligents ont cinq caractéristiques majeures qui en font des cibles attrayantes pour les pirates :

- Faible niveau de maturité : les outils de développement, de déploiement, de test et de sécurité des contrats intelligents en sont encore à leurs balbutiements. Cela crée une opportunité pour les développeurs de faire des erreurs, ainsi que des opportunités pour les acteurs malveillants.
- Aucune conséquence juridique stricte : Les contrats intelligents sont ouverts au public et tout le monde peut s'y engager. Ceci, associé à l'anonymat des contrats intelligents, implique que les pirates sont moins susceptibles de faire face à des conséquences juridiques pour les pirater que pour le piratage d'autres types de réseaux.
- Paiements plus élevés : les assauts réussis sur les contrats intelligents se traduisent souvent par des récompenses pouvant atteindre 10 millions de dollars. Cela est particulièrement vrai pour les attaques contre les portefeuilles et les échanges qui stockent une quantité importante d'argent d'un client. Les pirates sont attirés par cela car la possibilité d'obtenir ne serait-ce qu'une partie de sommes aussi importantes dépasse les récompenses d'une attaque bancaire classique.
- Réutiliser les outils existants : pour causer des dommages, les pirates n'ont pas besoin d'être des experts en contrats intelligents. Ils réutilisent les outils existants pour détecter les vulnérabilités et les chevaux de Troie bancaires courants pour voler les clés secrètes des portefeuilles numériques.

13. SAPIC=Stateful Applied pi Calculus.
14. CSP=Communicating Sequential Processes.

— Difficile à identifier : On n'est rien de plus qu'une adresse sur la blockchain. L'identification de la personne derrière l'adresse est difficile ou impossible si l'adresse n'est associée à aucune autre information identifiable.

5.2 Limites des méthodes formelles

Dans le cycle de vie du développement logiciel, les approches formelles sont essentielles. Ces approches présentent cependant des limites considérables. Ces faiblesses entravent l'utilité des approches formelles pour les produits logiciels. Voici quelques-unes des lacunes des méthodes formelles :

— Alors que des descriptions rigoureuses promettent d'améliorer la fiabilité du système, le temps de conception et la compréhension ; ils se font au prix d'une courbe d'apprentissage plus longue ; les disciplines mathématiques employées pour décrire formellement les systèmes informatiques sortent du cadre de la formation traditionnelle en ingénierie. De plus, la plupart des métamodèles des méthodes formelles sont souvent contraints d'améliorer la prouvabilité. Il existe un compromis substantiel entre l'exigence de rigueur et la capacité à simuler tous les comportements.

— En général, les exigences réelles de l'utilisateur peuvent différer de ce que l'utilisateur spécifie, et elles changeront très probablement avec le temps. Il n'y a aucun moyen de garantir la validité et l'exhaustivité d'une spécification en termes de besoins quotidiens de l'utilisateur tout en utilisant des méthodes formelles. Cependant, différentes façons de réduire la probabilité de spécifications inexactes existent dans la littérature, mais toutes les approches doivent commencer par un point de départ informel. Il est impossible d'être certain que vous avez correctement rassemblé tous les besoins des utilisateurs.

— Pour établir l'exactitude du programme, une description formelle du programme doit inclure une description de la façon dont le programme doit fonctionner en conjonction avec le matériel et le système d'exploitation utilisés. Une description formelle de l'environnement technique utilisé dans le développement de logiciels industriels n'est pas accessible dans la plupart des cas. Le défi est exacerbé par le fait que, selon l'approche formelle choisie, une telle description formelle doit avoir une forme très spécifique.

— La nature des approches formelles est descriptive et analytique. Ils n'étaient pas considérés comme imaginatifs. Il n'y a que des techniques formelles pour décrire et analyser les conceptions dans la réalité. Il n'existe pas de processus de conception formel. Nous devons mélanger les méthodes formelles avec d'autres approches pour construire un système pratique.

— Des approches formelles sont utilisées pour traiter le logiciel et sa documentation. D'autres aspects clés des produits logiciels, tels que la formation, le support client, la maintenance et l'installation, doivent être gérés de manière indépendante. Ensemble, ces composants et leur qualité créent un produit de haute qualité. Les approches formelles n'ont aucune incidence sur la qualité des produits logiciels. Par conséquent, la plupart des fournisseurs de produits logiciels performants doivent consacrer beaucoup de temps et d'attention à la gestion de tous les composants essentiels d'un produit logiciel.

6 Conclusion et orientations futures

Un contrat intelligent est un code de transaction auto-exécutable automatisé qui gère les conditions d'accord d'un contrat dans des chaînes de blocs programmables. L'objectif principal d'un contrat intelligent est d'éliminer le besoin d'un agent humain dans les affaires et le commerce entre des participants anonymes et identifiés. Comme les contrats intelligents sont des techniques informatisées autonomes qui sont déployées sur des chaînes de blocs pour mettre en œuvre l'accord entre l'acheteur et le vendeur, il est automatiquement vérifié et exécuté via un réseau informatique. Par conséquent, plusieurs techniques formelles ont été proposées dans la littérature pour fournir un moyen de vérifier et de valider l'exactitude des contrats intelligents en évolution. Par conséquent, cet article étudie systématiquement les méthodes formelles existantes utilisées pour le domaine émergent de la spécification et de la vérification des contrats intelligents afin de minimiser le risque de pannes et de bogues et d'éviter les coûts éventuels. Comme recommandations futures, nous recommandons les futures directions de recherche suivantes pour la vérification des contrats intelligents à l'aide de méthodes formelles :

— 7 Conclusion et orientations futures

Diminuer le coût et le temps d'exécution des approches formelles dans les différentes phases de vérification (26).

— Intégrer plusieurs concepts mathématiques et langages de modélisation à l'aide de techniques d'abstraction.
— Rendre les méthodes formelles plus simples et plus accessibles aux développeurs et aux testeurs de logiciels.
— Posséder des modèles et des théories réutilisables et paramétrés plutôt que de définir des modèles et des théories à partir de zéro à chaque fois.
— Utilisation des ressources modernes de super-informatique, de cloud computing et de calcul du brouillard pour accélérer le calcul et raccourcir la durée de l'ensemble du processus de vérification.

Références

[1] Github - pirapira/bamboo : Bamboo see https ://github.com/cornellblockchain/bamboo. `https://github.com/pirapira/bamboo`. (Accessed on 06/25/2022).

[2] Asad Abbas, Roobaea Alroobaea, Moez Krichen, Saeed Rubaiee, S Vimal, and Fahad M Almansour. Blockchain-assisted secured data management framework for health information analysis based on internet of medical things. *Personal and Ubiquitous Computing*, pages 1–14, 2021.

[3] Tesnim Abdellatif and Kei-Léo Brousmiche. Formal verification of smart contracts based on users and blockchain behaviors models. In *2018 9th IFIP International*

Conference on New Technologies, Mobility and Security (NTMS), pages 1–5. IEEE, 2018.

[4] Sidney Amani, Myriam Bégel, Maksym Bortin, and Mark Staples. Towards verifying ethereum smart contract bytecode in isabelle/hol. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 66–77. ACM, 2018.

[5] Pedro Antonino and AW Roscoe. Formalising and verifying smart contracts with solidifier : a bounded model checker for solidity. *arXiv preprint arXiv :2002.02710*, 2020.

[6] Shaun Azzopardi, Gordon J Pace, and Fernando Schapachnik. On observing contracts : deontic contracts meet smart contracts. In *Legal Knowledge and Information Systems*, pages 21–30. IOS Press, 2018.

[7] Massimo Bartoletti and Roberto Zunino. Bitml : a calculus for bitcoin smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 83–100, 2018.

[8] Massimo Bartoletti and Roberto Zunino. Verifying liquidity of bitcoin contracts. In *International Conference on Principles of Security and Trust*, pages 222–247. Springer, 2019.

[9] Mona Batra. Formal methods : Benefits, challenges and future direction. *Journal of Global Research in Computer Science*, 4(5) :21–25, 2013.

[10] Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. A game approach to determinize timed automata. *Formal Methods in System Design*, 46(1) :42–80, 2015.

[11] Lexi Brent, Neville Grech, Sifis Lagouvardos, Bernhard Scholz, and Yannis Smaragdakis. Ethainter : a smart contract security analyzer for composite vulnerabilities. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 454–469, 2020.

[12] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Yaron Velner. Quantitative analysis of smart contracts. In *European Symposium on Programming*, pages 739–767. Springer, Cham, 2018.

[13] Jiachi Chen, Xin Xia, David Lo, and John Grundy. Why do smart contracts selfdestruct ? investigating the selfdestruct function on ethereum. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(2) :1–37, 2021.

[14] Michael Coblenz, Reed Oei, Tyler Etzel, Paulette Koronkevich, Miles Baker, Yannick Bloem, Brad A Myers, Joshua Sunshine, and Jonathan Aldrich. Obsidian : Typestate and assets for safer blockchain programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 42(3) :1–82, 2020.

[15] Claudio Di Ciccio, Alessio Cecconi, Marlon Dumas, Luciano García-Bañuelos, Orlenys López-Pintado, Qinghua Lu, Jan Mendling, Alexander Ponomarev, An Binh Tran, and Ingo Weber. Blockchain support for collaborative business processes. *Informatik Spektrum*, 42(3) :182–190, 2019.

[16] Wesley Dingman, Aviel Cohen, Nick Ferrara, Adam Lynch, Patrick Jasinski,

Paul E Black, and Lin Deng. Defects and vulnerabilities in smart contracts, a classification using the nist bugs framework. *International Journal of Networked and Distributed Computing*, 7(3) :121–132, 2019.

[17] Wang Duo, Huang Xin, and Ma Xiaofeng. Formal analysis of smart contract based on colored petri nets. *IEEE Intelligent Systems*, 35(3) :19–30, 2020.

[18] Ede Eykholt, Lucius Gregory Meredith, and Joseph Denman. Rchain architecture documentation. *Retrieve. Jan*, 19 :2019, 2017.

[19] Péter Garamvölgyi, Imre Kocsis, Benjámin Gehl, and Attila Klenik. Towards model-driven engineering of smart contracts for cyber-physical systems. In *2018 48th annual IEEE/IFIP international conference on dependable systems and networks workshops (DSN-W)*, pages 134–139. IEEE, 2018.

[20] Guido Governatori, Florian Idelberger, Zoran Milosevic, Regis Riveret, Giovanni Sartor, and Xiwei Xu. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law*, 26(4) :377–409, 2018.

[21] Yoichi Hirai. Defining the ethereum virtual machine for interactive theorem provers. In *International Conference on Financial Cryptography and Data Security*, pages 520–535. Springer, 2017.

[22] Rateb Jabbar, Eya Dhib, Ahmed ben Said, Moez Krichen, Noora Fetais, Esmat Zaidan, and Kamel Barkaoui. Blockchain technology for intelligent transportation systems : A systematic literature review. *IEEE Access*, 2022.

[23] Rateb Jabbar, Noora Fetais, Mohamed Kharbeche, Moez Krichen, Kamel Barkaoui, and Mohammed Shinoy. Blockchain for the internet of vehicles : How to use blockchain to secure vehicle-to-everything (v2x) communication and payment ? *IEEE Sensors Journal*, 21(14) :15807–15823, 2021.

[24] Moez Krichen. A formal framework for conformance testing of distributed real-time systems. In *International Conference On Principles Of Distributed Systems*, pages 139–142. Springer, 2010.

[25] Moez Krichen. *Contributions to model-based testing of dynamic and distributed real-time systems.* PhD thesis, École Nationale d'Ingénieurs de Sfax (Tunisie), 2018.

[26] Moez Krichen. Improving formal verification and testing techniques for internet of things and smart cities. *Mobile Networks and Applications*, pages 1–12, 2019.

[27] Moez Krichen, Meryem Ammi, Alaeddine Mihoub, and Mutiq Almutiq. Blockchain for modern applications : A survey. *Sensors*, 22(14) :5274, 2022.

[28] Moez Krichen, Afef Jmal Maâlej, and Mariam Lahami. A model-based approach to combine conformance and load tests : an ehealth case study. *International Journal of Critical Computer-Based Systems*, 8(3-4) :282–310, 2018.

[29] Moez Krichen, Seifeddine Mechti, Roobaea Alroobaea, Elyes Said, Parminder Singh, Osamah Ibrahim Khalaf, and Mehedi Masud. A formal testing model for operating room control system using internet of things. *Computers, Materials & Continua*, 66(3) :2997–3011, 2021.

[30] Moez Krichen and Stavros Tripakis. State identification problems for timed automata. In *IFIP International Conference on Testing of Communicating Systems*, pages 175–191. Springer, Berlin, Heidelberg, 2005.

[31] Mariam Lahami, Afef Jmal Maâlej, Moez Krichen, and Mohamed Amin Hammami. A comprehensive review of testing blockchain oriented software. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2022, Online Streaming, April 25-26, 2022*, pages 355–362. SCITEPRESS, 2022.

[32] Cosimo Laneve, Claudio Sacerdoti Coen, and Adele Veschetti. On the prediction of smart contracts' behaviours. In *From Software Engineering to Formal Methods and Tools, and Back*, pages 397–415. Springer, 2019.

[33] Zhentian Liu and Jing Liu. Formal verification of blockchain smart contract based on colored petri net models. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 555–560. IEEE, 2019.

[34] Elio Jordan Lopes, Shaolin Kataria, Shashank Keshav, Sumaiya Thaseen Ikram, Muhammad Rukunuddin Ghalib, Achyut Shankar, and Moez Krichen. Live video streaming service with pay-as-you-use model on ethereum blockchain and interplanetary file system. *Wireless Networks*, pages 1–15, 2022.

[35] Afef Jmal Maâlej and Moez Krichen. A model based approach to combine load and functional tests for service oriented architectures. In *VECoS*, pages 123–140, 2016.

[36] Afef Jmal Maâlej, Moez Krichen, and Mohamed Jmaiel. Conformance testing of ws-bpel compositions under various load conditions. In *2012 IEEE 36th annual computer software and applications conference*, pages 371–371. IEEE, 2012.

[37] Afef Jmal Maâlej, Mariam Lahami, Moez Krichen, and Mohamed Jmaïel. Distributed and resource-aware load testing of ws-bpel compositions. In *ICEIS (2)*, pages 29–38, 2018.

[38] Gabor Madl, Luis Bathen, German Flores, and Divyesh Jadav. Formal verification of smart contracts using interface automata. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 556–563. IEEE, 2019.

[39] Anastasia Mavridou, Aron Laszka, Emmanouela Stachtiari, and Abhishek Dubey. Verisolid : Correct-by-design smart contracts for ethereum. In *International Conference on Financial Cryptography and Data Security*, pages 446–465. Springer, 2019.

[40] Mayukh Mukhopadhyay. *Ethereum Smart Contract Development : Build blockchain-based decentralized applications using solidity.* Packt Publishing Ltd, 2018.

[41] Keerthi Nelaturu, Anastasia Mavridou, Andreas Veneris, and Aron Laszka. Verified development and deployment of multiple interacting smart contracts with verisolid. In *Proc. of the 2nd IEEE International Conf. on Blockchain and Cryptocurrency (ICBC)*, 2020.

[42] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor.

Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th annual computer security applications conference*, pages 653–663, 2018.

[43] Laure Petrucci, Cristina Seceleanu, and Ana Cavalcanti, editors. *Critical Systems : Formal Methods and Automated Verification - Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems - and - 17th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2017, Turin, Italy, September 18-20, 2017, Proceedings*, volume 10471 of *Lecture Notes in Computer Science*. Springer, 2017.

[44] Meixun Qu, Xin Huang, Xu Chen, Yi Wang, Xiaofeng Ma, and Dawei Liu. Formal verification of smart contracts from the perspective of concurrency. In *International Conference on Smart Blockchain*, pages 32–43. Springer, 2018.

[45] Indrani Ray. Security vulnerabilities in smart contracts as specifications in linear temporal logic. Master's thesis, University of Waterloo, 2021.

[46] Ron van der Meyden. On the specification and verification of atomic swap smart contracts. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 176–179. IEEE, 2019.

[47] Fujun Wang, Zining Cao, Lixing Tan, and Hui Zong. Survey on learning-based formal methods : Taxonomy, applications and possible future directions. *IEEE Access*, 8 :108561–108578, 2020.

[48] Konrad Weiss and Julian Schütte. Annotary : A concolic execution system for developing secure smart contracts. In *European Symposium on Research in Computer Security*, pages 747–766. Springer, 2019.

[49] Kazuhiro Yamashita, Yoshihide Nomura, Ence Zhou, Bingfeng Pi, and Sun Jun. Potential risks of hyperledger fabric smart contracts. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 1–10. IEEE, 2019.

[50] Nejc Zupan, Prabhakaran Kasinathan, Jorge Cuellar, and Markus Sauer. Secure smart contract generation based on petri nets. In *Blockchain Technology for Industry 4.0*, pages 73–98. Springer, 2020.