UNIVERSITÉ DE MONTPELLIER

MODEL CHECKING

Suivi de Parcours Personnel

Christian Retoré

Professeur, Université de Montpellier

Enseignement:

département informatique de la faculté des sciences

licence et master informatique

Responsable du master IA 2020-2024

Recherche:

Laboratoire d'Informatique, de Robotique

et de Microélectronique de Montpellier

UNIVERSITÉ DE MONTPELLIER

Model Checking

Pourquoi parler de Model Checking?

- Très très utile et répandu dans l'industrie (80% de la sureté informatique)
- .... et donc beaucoup d'emplois intéressants et correctement rémunérés
- ... et non menacés par l'IA car ce sont des méthodes exactes.
- Belle et utile mise en œuvre de notions et d'algorithmes issus de la logique (qui est ma spécialité)
- Cours qui plait à tous les étudiants:
  marrant en pratique et avec une belle théorie.

C'est quoi le "Model Checking" ?

- WIKIPEDIA : *En informatique , la vérification de modèles , ou model checking en anglais, est le problème suivant : vérifier si le modèle d'un système (souvent informatique ou électronique) satisfait une propriété. Par exemple, on souhaite vérifier qu'un programme ne se bloque pas, qu'une variable n'est jamais nulle, etc. Généralement, la propriété est écrite dans un langage, souvent en logique temporelle . La vérification est généralement faite de manière automatique. Sur le plan pratique, la vérification de modèles est devenue, au niveau industriel, la méthode de vérification de code et de systèmes matériels la plus populaire et la plus utilisée aujourd'hui.*

L'informatique est "partout": pas seulement dans les programmes

Programme NSI mieux que le programme en info de la fac.

Info à la fac : un programme séquentiel calcule une valeur pour une entrée, en isolation.

- Acccès //BD (par ex BD des achats de tous les magasins Carrefour)
- Système d'exploitation / multitache (clavier, écran, disque,...) Réseaux.
- Systèmes parallèles + interactions avec le monde physique
- Ascenceur , voiture, avion,...
- Robots médicaux ou autres
- Distributeur automatique de billets
- Systèmes réactifs: en interaction constante avec l'environnement

Motivation: sureté logicielle et des systèmes cyberphysiques

Concernant les systèmes informatisés courants, nous préférons

1. Arriver vivants quand nous prenons l'avion, le train, la voiture, l'ascenseur (systèmes embarqués, circuits,...)
2. Ne pas ressortir blessé, handicapé, mort d'une salle d'opération (robots chirurgicaux, soins informatisés,...)
3. Ne pas nous faire voler nos données, ni notre argent. (réseaux, protocoles de connexion,...)

Bref, nous préférons que les systèmes informatisés fonctionnent BIEN

C'est-à-dire qu'ils respectent leur spécifications.

Comment s'en assurer? Vérification et top de la technique: Model Checking

Techniques de vérification

1. **Tests** (systèmes cyber physiques, parallèles, ...) problèmes:
   - Coûteux
   - Risqués
   - Et surtout jamais complets, c'est le plus embêtant.

2.Preuves de programmes

Seulement pour les programmes séquentiels - non parallèles en isolation (sans communication entre différents programmes), sans entrées sorties en cours d'exécution, sans effet de bord, 1 entrée -> 1 sortie en temps fini (cf. Notions de Calculabilité ou Preuves de Programmes)

3.Model checking: complet et pour TOUS les programmes et systèmes informatisés (systèmes cyberphysiques, parallèles, ...)

7 Accidents évitables si Vérif par Model Checking

Systèmes embarqués (aéronautique, automobile)

Protocoles de communication

Logiciels critiques (médical, ferroviaire, nucléaire)

Accidents évitables si Vérif par Model Checking

Accident ferroviaire de la gare de Clapham Junction (1988)

Contexte : Collision de trains à Londres, causant 35 morts.

Cause : Une erreur de câblage dans le système de signalisation.

Lien avec le model checking : Une modélisation formelle du système de signalisation et une vérification automatique auraient pu détecter cette erreur de conception.

Systèmes embarqués dans l'automobile vérifiés

Contexte : Les systèmes de freinage ABS, les régulateurs de vitesse adaptatifs, etc., sont critiques.

Utilisation réelle : Des constructeurs comme Toyota ou Bosch utilisent des outils de model checking

(ex.UPPAAL ,NuSMV) pour vérifier les systèmes temps réel embarqués.

Accidents évitables si Vérif par Model Checking

Missile Patriot (1991)

➤Cause : Bug dans la gestion du temps, entraînant une erreur de 600 mètres dans la détection.

➤Impact : 28 morts.

➤Model checking aurait pu détecter la dérive temporelle dans les systèmes embarqués.

.

Accidents évitables si Vérif par Model Checking

Accident du Therac -25 (années 80)

Système cyber physique combinant des faisceaux d'électrons et de rayons X pour traiter le cancer

Cause : Concurrence mal gérée dans le logiciel de radiothérapie.

Impact : Surdoses mortelles de radiation. (une dizaine de cas)

Une vérification formelle des propriétés de sécurité aurait pu prévenir ces erreurs.

Accidents évitables si Vérif par Model Checking

Dispositifs médicaux vérifiés (pompes à insuline, pacemakers)

Contexte : Ces dispositifs doivent fonctionner de manière fiable et sûre.

Utilisation réelle : Des chercheurs ont utilisé le model checking pour vérifier les algorithmes de contrôle des pacemakers

afin d ’éviter des comportements dangereux.

Exemple : Projet Physiological Closed -Loop Control Systems de la FDA et de l ’Université de Pennsylvanie.

Accidents informatiques évitables

Bug du processeur Intel Pentium (1994)

Erreur dans l'algorithme de division flottante.

➤Impact : Coût de remplacement estimé à 475 millions $

➤Model checking aurait pu valider formellement cet algorithme de division

Accidents informatiques évitables

Bugs du protocole TCP ou SSL:

Vulnérabilité Heartbleed; triple handshake bug dans SSL/TLS; race conditions dans TCP; erreurs de validation de

certificats .

Succès du model checking, versions suivantes:

Le protocole Needham -Schroeder a été analysé par model checking, révélant une faille de sécurité.

Des variantes de TLS ont été modélisées avec des outils comme SPIN ,UPPAAL , ouTLA+ , permettant de détecter des incohérences dans le protocole.

Des travaux académiques ont montré que des implémentations réelles de SSL/TLS contiennent des erreurs que le model checking aurait pu prévenir.

Accidents évitables : le plus connu / étudié : Ariane 5

Étude de cas : L ’échec du vol Ariane 5 – 4 juin 1996

Contexte
Mission : Premier vol de qualification du lanceur Ariane 5.
Objectif : Mise en orbite de satellites scientifiques.
Résultat : Explosion 37 secondes après le lancement.
Perte estimée : Environ 1,9 milliard de francs français (~370 millions USD).

Cause immédiate

Une conversion de type dans le logiciel embarqué (Ada) a provoqué une exception d ’opérande .

Le logiciel tentait de convertir une valeur flottante 64 bits en entier signé 16 bits, ce qui a généré une erreur fatale. Ce code était hérité d ’Ariane 4, où il fonctionnait correctement, mais inutile pour Ariane 5.

Accidents évitables : le plus connu / étudié : Ariane 5

L ’échec du vol Ariane 5 – 4 juin 1996

Analyse approfondie (selon Gérard Le Lann, INRIA)
Le rapport de recherche RR-3079 , INRIA propose une lecture différente de celle du rapport officiel :

Erreurs non détectées :
Pas d ’erreur de spécification ou de conception logicielle en soi.
Les fautes proviennent d ’unemauvaise capture des besoins et d’uneconception système défaillante .

Problèmes identifiés :
Réutilisation de code sans validation formelle .
Absence de méthode de vérification rigoureuse (comme le model checking).
Manque de preuve de conception et de dimensionnement du système embarqué.

Accidents évitables : le plus connu / étudié : Ariane 5

Ce que le model checking aurait pu apporter :

Le model checking permet de :

- Vérifier automatiquement des propriétés comme l'absence d'erreurs d'exécution, la sûreté, la vivacité.
- Simuler tous les états possibles d'un système pour détecter des erreurs avant l'implémentation.

Dans le cas d'Ariane 5 :

- Il aurait permis de détecter l'exception de conversion dans tous les scénarios de vol.
- Il aurait révélé que certaines routines étaient inutiles ou dangereuses dans le nouveau contexte.
- Il aurait pu valider formellement que le système embarqué respecte les contraintes de sécurité.

Leçons tirées

L'accident a mis en lumière la nécessité de méthodes formelles dans les systèmes critiques.

Il a encouragé l'adoption de techniques comme :

TLA+ ,SPIN ,UPPAAL pour la vérification de modèles.

Analyse statique etpreuve de correction dans les systèmes embarqués

Vérification d'un modèle (et non du système réel)

Plus facile / moins cher:

pas besoin de fabriquer le système

… et de recommencer si le système n'est pas sûr.

Systèmes complexes composés de systèmes plus petits,

et communicants entre eux.

- Programmes, programmes parallèles, circuits,..
- Dispositifs physiques

La complexité algorithmique est-elle un problème?

Rarement: certains algos de model checking très complexes, mais:

- Travaux pour faire baisser la complexité des algos de model checking

ou pour étendre les bons résultats pour les spécifications simple à des spécifications plus compliquées.

  - LTL CTL* PSPACE en la taille de la formule à vérifier
  - CTL polynomial.

- A la différence d'autres activités informatiques, on vérifie **1** fois le schéma d'un circuit ou d'un système cyber physique avant sa mise en service....

Exemple: train d'atterrissage du Rafale. Esterel, Berry années 90.

Exemple concret : Train d' atterrissage du Rafale avec Esterel

Esterel (Gérard Berry, 1991, mon chef de Postdoc en 1993-1994) est

- un langage de programmation synchrone conçu pour les systèmes embarqués réactifs,
- où le temps et la précision sont cruciaux.

Il est particulièrement adapté aux systèmes avioniques comme ceux du Rafale, où les composants doivent réagir de manière déterministe à des événements (comme la sortie ou la rentrée du train d ’atterrissage).

Exemple concret : Train d' atterrissage du Rafale avec Esterel

Application à la vérification du train d ’atterrissage

Dans le cas du Rafale, Esterel a été utilisé pour :

- Modéliser le comportement du système de commande du train d ’atterrissage :
  - Extension/rétraction
  - Verrouillage mécanique
  - Détection d ’anomalies
- Vérifier formellement les propriétés de sécurité :
  - Le train ne doit pas se rétracter au sol.
  - Il doit être complètement sorti avant l ’atterrissage.
  - Les capteurs doivent être cohérents avec les commandes.
- Générer automatiquement du code fiable :
  - Le code Esterel peut être compilé en C pour être embarqué dans les calculateurs du Rafale.
  - Cela réduit les erreurs humaines et facilite la certification DO -178B (norme aéronautique).

Exemple concret : Train d' atterrissage du Rafale avec Esterel

Étude emblématique:

Un projet mené par CEA, Dassault Aviation et Esterel Technologies a démontré que l'utilisation d ’Esterel permettait de vérifier exhaustivement les scénarios de fonctionnement du train d ’atterrissage, y compris les cas d ’erreur ou de panne. Ce travail a été présenté dans plusieurs conférences sur les systèmes embarqués et la vérification formelle.

Aide IA Machine Learning ChatGPT... possible? (difficile de ne pas en parler aujourd'hui)

NON car ON NE VEUT PAS d'une réponse du genre.

" Les circuits et programmes du train. d'atterrissage du Rafale semblent sûrs, ils respectent bien leurs spécifications en toute situation. "

Car on ne peut se satisfaire de "en général ça marche" (sans même % de succès) qui serait basée sur le fait que les autres systèmes similaires marchent bien et ont connu peu voire aucun incident. On veut être totalement sûr, car ce sont des risques de vie ou de mort.

Notons qu'on peut améliorer les tests grâce à l'IA, mais on est alors dans des hauts pourcentages de fiabilité, pas "sûr et certain". Par exemple un ancien doctorant travaille ici:

IA Thunders Des Agents de Test IA qui Génèrent, Exécutent, Analysent et Corrigent: mais ce n'est pas automatisés, et ce n'est pas du 100% sûr.

UNIVERSITÉ DE MONTPELLIER

Historique

Ancêtre du model checking: le parallélisme

(j'ai découvert ces sujets à Londres en 1989)

1978 Communicating Sequential Processes Hoare
un outil de spécification formelle de l'exécution concurrente de systèmes variés
communication / diffusions : variables partagées
1980 Calculus of Communicating Systems Milner

Les processus communiquent 1-1 sur des canaux

1980 Henessy Milner Logic (multimodale temporelle cf. Systèmes de Transitions Etiquetées)

1988 Pi calcul Milner Les processus peuvent échanger des noms de canaux

Les philosophes qui mangent des nouilles avec deux fourchettes sur une table de trois avec trois assiettes et trois fourchettes, une entre chaque paire d'assiettes : -)Je l'avoue volontiers, j'avais trouvé cela plutôt incompréhensible avec une syntaxe vraiment illisible.

Model Checking

Amir Pnueli découvre la logique temporelle de Arthur Prior (philosophe) à la fin des années 70:
« En mathématiques, la logique est statique. Elle traite des liens entre des entités qui existent dans le même cadre temporel. Lorsque l'on conçoit un système informatique dynamique qui doit réagir à des conditions en constante évolution, [...] on ne peut pas concevoir le système sur la base d'une vision statique. Il est nécessaire de caractériser et de décrire les comportements dynamiques qui relient les entités, les événements et les réactions à différents moments. La logique temporelle traite donc d'une vision dynamique du monde qui évolue au fil du temps. »
(Amar Pnueli)

1977 « The Temporal Logic of Programs » a introduit la notion de raisonnement sur les programmes en tant que chemins d'exécution, ce qui a insufflé une nouvelle vie au domaine de la vérification des programmes qui, jusque-là, était une vérification à la Hoare , précondition, instruction(s), postcondition.

Plusieurs prix Turing pour le Parallelisme et le Model Checking

Model Checking

1996 Amir Pnueli Pour ses travaux novateurs introduisant la logique temporelle dans les sciences informatiques et pour ses

contributions exceptionnelles à la vérification des programmes et des systèmes.

2007 Edmund Clarke, E. Allen Emerson et Joseph Sifakis pour leur rôle dans le développement de la vérification de modèles en une technologie de vérification hautement efficace largement adoptée dans les industries du matériel et des logiciels.

Parallélisme :

1976 Robin Milner LCF, preuve assistée par ordinateur ; ML, premier langage avec inférence de types polymorphes; CCS, une théorie générale de la concurrence.

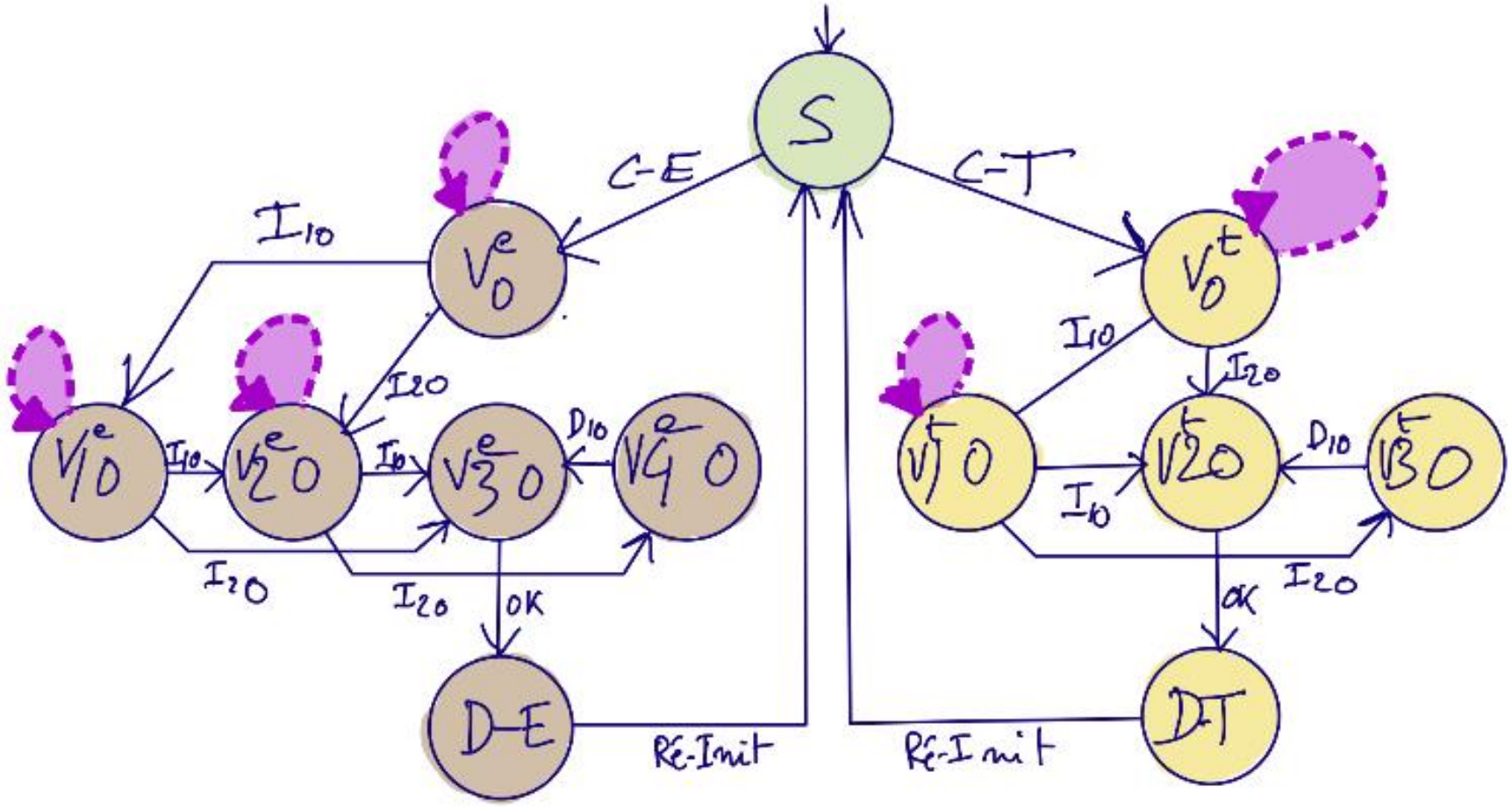
2013 Leslie Lamport théorie et pratique des systèmes distribués et concurrents (notamment causalité et horloges logiques) @Lamport : merci pour LATEX !!!

UNIVERSITÉ DE MONTPELLIER

Pour commencer il faut....
modéliser le système cyberphysique
-> LTS Labelled Transtion Systems

25 Petit exemple : machine expresso (30cts) ou thé (20cts)

Boucle si pièce qui dépasse le montant? Ou pas?



26 Système de transition étiqueté (LTS)

- Nombre fini d'états
  - Etat initial
  - Pas d'état final.
- De tout état on peut accéder à un état
- Nombre fini de propositions atomiques décrivant l'état du système.

Plusieurs sortes d'accessibilités entre états (étiquetées par les actions)

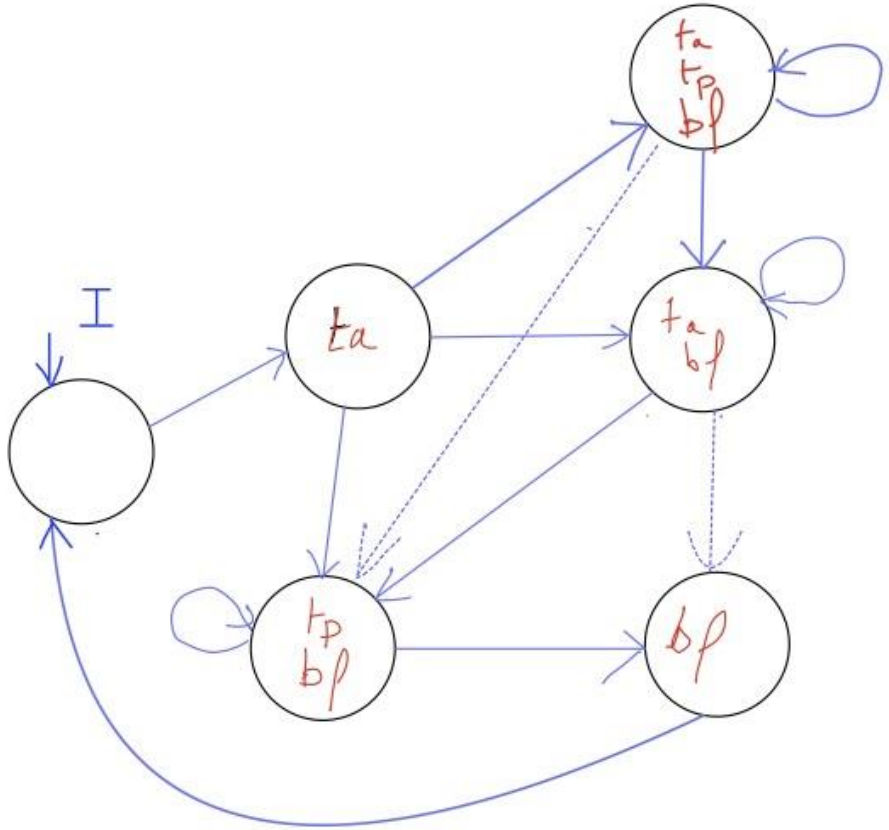
Dans chaque état, des propositions atomiques (des lettres) sont vraies ou Fausses.

Exécution: chemin infini, suite d'états décrits par l'ensemble fini des propositions qui y sont vraies.

Avec la définition, dessiner le LTS d'un passage à niveau.

Propositions:

- bf barrière totalement fermée
- bo = ¬bf barrière partiellement ouverte ou ouverte;
- ta un train approche
- tp un train passe



Logique propositionnelle temporelle

Logique propositionnelle: parfait pour exprimer des propriétés

Mais comment quantifier dans le temps?

- Si le train approche ta à un instant alors à l'instant d'après la barrière est fermée bf.

- Si la barrière est fermée à un instant alors il existe un instant où elle sera ouverte.

Logique propositionnelle temporelle

Plusieurs sortes plus ou moins expressives:

- LTL temps linéaire (1 exécution à la fois)
- CTL temps branchant opérateurs temporels toujours quantifiés par les chemins
- CTL* temps branchant tous les opérateurs temporels quantifiés ou non

On dira qu'une proposition est vraie dans une exécution ou dans un LTS lorsqu'elle est vrai dans les états initiaux. Dans les diverses logiques temporelles considérées, il ya toujours moyen de dire que la propriété est toujours vraie à partir de maintenant, donc cette définition en change rien.

UNIVERSITÉ DE MONTPELLIER

LTL Logique Temporelle Linéaire pour parler d'une exécution

LTL linear temporal logic: pour parler d'UNE exécution du LTS

φ::= ⊥ | ⊤ | p | ¬φ | φ∧φ | Xφ | Fφ | Gφ | φUφ

X next

F à instant futur – maintenant inclus

G à tout instant futur – maintenant inclus

U: until φUψ la formule φ est vraie de maintenant jusqu'à ce que ψ soit vraie, si ψ est vraie maintenant, ok.

R: ¬(¬ φU¬ψ)

LTL linear temporal logic: pour parler d'UNE exécution du LTS

$\varphi ::= \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi U \varphi$

On peut maintenant écrire par une formule:

- Si le train approche ta à un instant alors à l'instant d'après la barrière est fermée bf.
- Si la barrière est fermée à un instant alors il existe un instant où elle sera ouverte.

Model Checking: LTS A, formule φ...
φ est vraie dans toute exécution de A ?

Comment faire? On voit le LTS comme un automate A, propriétés sur les flèches et non sur les états. De tels automates peuvent aussi être associés aux formules. Une exécution est un chemin infini.

On associe à une formule φ un automate B dont toutes les exécutions sont exactement celles où φ n'est pas vraie. L'automate est très grand par rapport à la formule. Algo compliqué à programmer soi-même mais des plateformes industrielles existent et fonctionnent bien.

LTL permet d'exprimer et de vérifier mais il faut faire attention à la complexité algortihmique, et donc à al taille des formules à vérifier.

On construit l'automate AnB dont les exécutions sont les éxécutions communes à A et à B. S'il n'a aucune exécution, alors toute exécution de A n'est pas une exécution de B, ne satisfait pas la propriété (non φ) et donc satisfait φ!!!

Outils industriels Model Checking LTL

1. SPIN

Utilisation : Vérification de protocoles de communication, systèmes concurrents.

Fonctionnement : Traduit les propriétés LTL en automates de Büchi pour vérifier leur satisfaction par le modèle.

Avantage : Très mature, bien documenté, utilisé dans l ’industrie et l ’enseignement.

2. NuSMV / NuXMV

Langage de modélisation : Langage SMV

Type de logique : LTL, CTL

Utilisation : Vérification de systèmes embarqués, circuits logiques, logiciels critiques.

NON TRAITÉ

UNIVERSITÉ DE MONTPELLIER

CTL Logique Temporelle Arborescente
Computational Time Logic

Système de transition "non étiqueté "-> computational tree

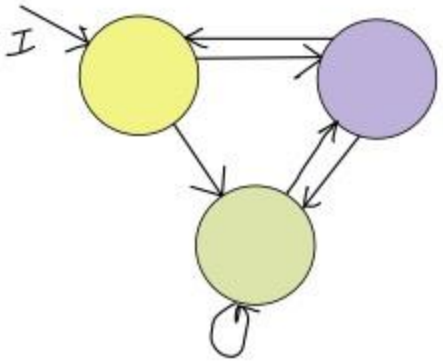
Arbre des calculs: arbre infini, à branchements finis, non déterministe.

Une exécution: 1 chemin infini, 1 suite d'états décrits par l'ensemble fini des propositions vraies dans l'état considéré.

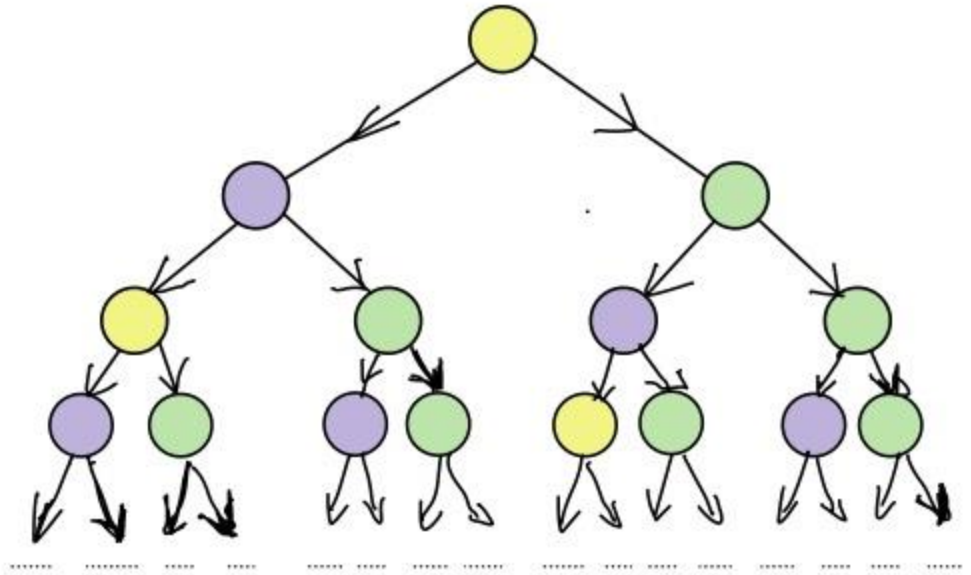
Les noms des transitions sont inutiles et omis

NON TRAITÉ

Système de transition "non étiqueté "-> computational tree



NON TRAITÉ



Un LTS ….. et l'arbre de tous ses chemins/exécution possibles.
Une exécution est un chemin infini passant de la racine.

47 LANGAGE CTL

NON TRAITÉ

Tout opérateur temporel X, F, G, U est utilisé avec une quantification universelle ou existentielle sur les chemins .

φ::= ⊤|p | ¬φ | φ∧φ | EXφ | AX φ | EF φ | AF φ |EG φ | AG φ | E φUφ| A φUφ

Exemples:

- T,s |=EFφ ssi, il existe un chemin ρ tel que ρ |= s et il existe j ⩾1 tel que sur ce chemin T,ρj|= φ
- T,s |=AφUψ ssi pour tout chemin ρ tel que ρ 1= s il existe j ⩾1 tel que T,ρj|= ψ et T,ρk |= φ pour tout 1 ⩽k < j

Le temps dans CTL s'écrit avec uniquement EX φ EGφ E φ Uψ

47 LANGAGE CTL

NON TRAITÉ

On peut donc exprimer dans CTL, sur l'exemple précédent, des propriétés du genre:

- Il existe un chemin ouj au départ on a jaune et ensuite que des verts pour lesquels il existe un chemin commençant par un violet.
- Sur tout chemin, s'il y a un vert alors il y a un chemin partant de ce vert et qui ne contient que des verts.
- Sur tout chemin, s'il y a un jaune, alors il part un chemin de ce jaune sur lequel il y a un autre jaune.

MODEL CHECKING CTL

NON TRAITÉ

[[φ]] est l'ensemble des états du système dans lesquels φ est vraie.

Si les états initiaux appartiennent à [[φ]] alors φ est vraie dans le computational tree.

Si W est un ensemble d'états, Pré(W) désigne l'ensemble des états d'où une transition conduit en un état de W. Par exemple: [[EX φ]]= Pré([[φ]])

On calcule en suivant la construction de la formule [[φ]] si les état initiaux en font partie, alors le LTS satisfait φ.

MODEL CHECKING CTL: calcul de [[φ]] par induction sur φ

On remarque que:

1. [[EG φ]] est le plus grand point fixe de EG(W)= [[φ]] ∩(Pre(W))
2. [[E φUψ]]est le plus petit point fixe de EU(W)= [[ψ]] ∪([[φ]] ∩Pre(W))

[[p]] est donné

[[¬φ]]= complémentaire de [[φ]]

[[EX φ]]= Pré([[φ]])

[[EG φ]]point fixe comme ci -dessus

[[E φUψ]]point fixe comme ci -dessus

NON TRAITÉ

MODEL CHECKING CTL

Le calcul du point fixe n'est pas très couteux -- en fonction de la taille des transitions, quadratique en le nombre d'états.
Le nombre de sous formules est linéaire en la taille de la formule.

NON TRAITÉ

Les limites de CTL

NON TRAITÉ

Comment dit -on en CTL:
- sur chaque exécution il y a un p suivi d'un q?
On ne le dit pas, c'est impossible à exprimer en CTL.

AF (p&Xq) n'est pas une formule, il faudrait dire:

- Soit AF(p&AXq) sur tout chemin il ya un p et sur tout chemin partant de ce p il y a un q à l'instant d'après.
- Soit AF(p&EXq) sur tout chemin il ya un p et sur au moins un chemin partant de ce p il y a un q à l'instant d'après.

CTL oblige a requantifier sur les chemins commençant par l'état où il y a p

Model Checking CTL utile, très efficace et donc très utilisé:

NuSMV / NuXMV

- Sans doute le plus utilisé dans l'industrie (aéronautique, ferroviaire, spatial, nucléaire, automobile).
- Support **complet de CTL** (vérification symbolique BDD et SAT-based).
- Extension **nuXmv** encore plus performante et maintenue.
- Utilisé dans plusieurs chaînes de certification (DO-178C, EN-50128, etc.).

Cadence SMV / CMU SMV

- Ancêtre de NuSMV, toujours utilisé dans certains secteurs industriels.
- Support direct de **CTL**.

Cadence JasperGold

- Outil industriel leader dans la vérification de circuits (EDA).
- Support du model checking CTL via sa logique temporelle orientée propriété (propriété CTL transformées en équivalents).
- Industrie : Intel, ARM, AMD, NVIDIA, etc.

NON TRAITÉ

NON TRAITÉ

UNIVERSITÉ DE MONTPELLIER

CTL* =CTL+LTL

Les formules de CTL*

NON TRAITÉ

Comme CTL mais si ni A ni E ne précède l'opérateur temporel, alors on reste sur le même chemin.

Dans toute exécution il y a un p qui est suivi d'un q.

A F (p&Xq)

Sur tout chemin il y aura dans le futur un p suivi de q (même chemin)

Comme déjà dit, ce n'est pas expremiable en CTL qui propose seulement:

≠ AF (p&AXq) un p dont tout état suivant sur tout chemin aura q

≠ AF (p&EXq) un p dont un état suivant sur un chemin aura q

Les formules de CTL*

NON TRAITÉ

A sur tout chemin partant de l'état courant

E sur au moins un chemin partant de l'état courant

X dans l'état suivant, F dans un états futurs, G dans tous les états futurs,

GU H G est vrai jusqu'à ce que H le soit...

Dans toute exécution il y a un p qui est suivi d'un q.

I|= A F (p&Xq)

Sur tout chemin il y aura dans le futur un p suivi de q (même chemin)

≠ i|= AF (p&AXq) un p dont tout état suivant sur tout chemin aura q

≠ i|= AF (p&EXq) un p dont un état suivant sur un chemin aura q

52 Model Checking CTL en pratique

CTL* intègre la complexité du model checking de LTL qui se trouve itéré pour chaque sous formule de la formule de CTL*, à vérifier. C'est donc complexe, car chaque étape de model checking pour LTL est déjà complexe. Il y a donc moins d'outils.

- **NuSMV / NuXMV**

- Support **partiel** de CTL* via une traduction interne vers automates.
- Très utilisé lorsque l'industrie a besoin du mélange CTL + LTL.

- **SPIN**

- N'accepte pas CTL* directement, mais supporte certaines formules équivalentes via automates Büchi.
- En pratique, utilisé dans l'aérospatial et les télécoms.

- **Isabelle/HOL** + model checking plugins

- Permet d'exprimer **CTL et CTL*** dans un cadre de vérification formelle interactive (ex. domaine de l'aéronautique).

NON TRAITÉ

NON TRAITÉ

UNIVERSITÉ DE MONTPELLIER

Un sujet très actuel
Lutte contre les intrusions.
Méthode:
logique modale temporelle + multiagents

Systèmes Multi-Agents

Les SMA sont issus de l ’intelligence artificielle distribuée (IAD) , qui s ’est développée dans les années 1970 -1980 :
1950 -1975 : IA mono -agent (Turing, STRIPS, Hearsay I)
1975 -1985 : naissance de l ’IAD, avec des systèmes comme le tableau noir (blackboard systems)
1985 -1995 : apparition des agents rationnels et des premières architectures SMA
1995 et après : maturité du domaine, avec des applications en robotique, simulation sociale, négociation, etc.
Les SMA modélisent des entités autonomes capables de percevoir, raisonner, interagir et apprendre dans un environnement distribué.

NON TRAITÉ

Système Multi -Agents

Quand sont apparues les logiques épistémiques (qui modélisent le savoir et parfois les croyances des agents) ?

Les logiques épistémiques sont une branche de la logique modale, introduite dans les années 1960 :

Jaakko Hintikka est l'un des pionniers, avec son ouvrage Knowledge and Belief (1962)

Elles permettent de formaliser ce que les agents savent ou croient , y compris la connaissance commune et la connaissance mutuelle

Elles sont utilisées en informatique pour modéliser les systèmes distribués, les protocoles, la sécurité, et les SMA et les jeux.

NON TRAITÉ

SMA + Temps: sécurité contre les intrusions

1.Modélisation de la connaissance dans les SMA

Les SMA ont rapidement intégré des logiques épistémiques pour modéliser les états mentaux des agents (connaissance, croyance, intention).

Cela a permis de formaliser des comportements comme la coopération , la coordination , ou la planification conjointe .

2.Logiques multi -modales

Des travaux dès les années 1990 ont combiné logique

épistémique et logique temporelle pour modéliser des univers multi -

agents dynamiques.

3. Depuis 2020 combinaisons de **logique SMA (genre S5) et temporelle** pour faire de la vérif contre les intrusions.

NON TRAITÉ

Des questions?

UNIVERSITÉ DE MONTPELLIER

Parcours personnel

Formation primaire et secondaire

NON TRAITÉ

- Né en 1964 dans la banlieue de Rouen (Normandie)
- CP CE1 école Albert Camus, Mont St Aignan dans la banlieue de Rouen
- puis CE2 CM1 CM2 école de la Mission Française à Tlemcen (Algérie)
- 6e au lycée Pasteur (Oran, Algérie)
- 5e 4e 3e Collège Paul Éluard, Ste Geneviève des Bois, Essonne
- 2nde 1ere Tale lycée Albert Einstein, Ste Geneviève des Bois, Essonne
- Bac C en 1980.
- Bilan du lycée: très bons résultats en maths et en philo;
  bons résultats en français, latin et grec, physique (mécanique du point);
  moyen dans les autres matières

Parcours: supérieur

NON TRAITÉ

- Maths sup, lycée St Louis, Paris, échec:
  - Ma mère voulait que j'y aille, et moi pas, et j'avais 16 ans...
  - L'ambiance était très déplaisante.
  - Je n'aimais que les maths et une partie de la physique.
  - Je séchais tous les cours qui ne me plaisaient pas....
- Université Paris 6 (Paris Sorbonne) : DEUG Sciences des Structures et de la Matière, avec le plus de maths possible et le moins de physique chimie possible. Pas d'info à l'époque.
- Université Paris 6 (Paris Sorbonne): 3e et 4e année de maths pures.
- Université Paris 7 (Paris Cité): 5e année initiation à la recherche en maths.
- Université Paris 7 (Paris Cité): doctorat de mathématique, en logique mathématique.

Pendant les études et la thèse... salarié à temps

NON TRAITÉ

1983-1987: de la 2e année de fac (19 ans) à la la bourse de thèse (24 ans) Surveillant d'externat et remplaçant prof de maths dans divers collèges du Val de Marne (94, Académie de Créteil)

1989-1991: service national en coopération en tant que chargé de recherche à Imperial College (Londres) dans un labo d'informatique...

... découverte de l'informatique et des applications informatiques de mes travaux mathématiques.

Ensuite... chercheur (1993-2003) puis enseignant chercheur depuis 2003

- Recherches: en logique mathématique, pour elle-même et suivant le lieu et la période sur diverses applications in formatique de la logique: programmation fonctionnelle, parallélisme, traitement automatique du langage naturel,...
- 1993-2003 Chargé de recherche INRIA (Institut National de la Recherche en Informatique et Automatique) en divers endroits: Nice, Nancy, Rennes,...
- 2003-2014 Professeur d'Informatique Université de Bordeaux
  Responsable du master Algorithmes et méthodes formelles
- 2014-2025 Professeur d'Informatique, Université de Montpellier
  - Responsable du master IA et science des données
- 2025-2030 Professeur émérite / Retraité (activités de recherche uniquement).

NON TRAITÉ

Votre orientation

NON TRAITÉ

Trouver ce que vous aimez FAIRE.

C'est difficile. C'est un travail d'introspection. Chacun est différent.

Ne pas écouter les autres. Pas trop vos parents. Regardez où vous avez de bonnes notes et discutez vos profs.

Ce que vous aimez faire ce n'est pas forcément ce qui vous intéresse à étudier ou à travailler.

Par ex jouer avec des IA c'est bluffant. Programmer des applications d'IA des jeux video c'est du calcul numérique (réseaux de neurones très proches des statistiques et probabilités avec pas mal de calcul vectoriel sur des vecxteurs de grande dimension).

Faire ce que vous aimez, c'esqt très important. On passe beaucoup de temps au travail, autant s'y plaire, et de plus, pour les études, on travaille plus facilement et on réussit mieux quand on aime le sujet.

Votre orientation... et l'informatique, l'IA etc.

L'informatique a un atout: il ya des emplois d'informaticiens et à tous niveau dans toutes les entreprises des différents secteurs d'activité: santé, aéronautique, automobile, finance, assurance, agriculture,… et bien sûr dans les sociétés de services informatiques — et dans l'enseignement.

Attention à l'IA: développée presqu'exclusivement par les GAFAM... Donc il n'y a pas beaucoup d'emplois dans ce secteur, et presque pas avec des études courtes.

NON TRAITÉ

/!\ A mon avis, l'IA va supprimer tous les emplois de développeurs! La plupart des programmes peuvent être bien écrits par des IA comme ChatGPT et c'est moins cher que de payer des développeurs.

Je vous renvoie aussi aux problèmes sociaux-économiques et écologiques dûs à l'IA.