UNIVERSITÉ DE MONTPELLIER

MODEL CHECKING

Christian Retoré (UM dépt. Info / LIRMM)

Model Checking: ce n'est pas mon sujet mais... j'aime bien.

- Sujet accessible à un logicien de formation, tout comme BD ou Calculabilité...
- Perso: Rennes IRISA, Bordeaux LABRI, Davide Catta (doctorant) --> Cours logique modale ici, HAI816I
- Sujet plaisant pour l'enseignant et pour les étudiants (ou alors les retours sont trop polis ;-)
- Jolie Théorie + Utile en Pratique
- 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**

UNIVERSITÉ DE MONTPELLIER

Motivation

L'informatique est "partout"

Programme NSI mieux que programme en info de la fac. Info à la fac = un programme qui calcule séquentiellement une valeur en fonction d'une entrée, en isolation. (j'exagère, bien sûr)

- 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 convenablement.

Comment s'en assurer?

Techniques de vérification

1. Tests (systèmes cyber physiques, parallèles, ...)
    - Coûts
    - Risques
    - Jamais complets
2. Preuves de programmes
    - Seulement pour les programmes séquentiels - non parallèles en isolation, sans entrées sorties en cours d'exécutions, sans effet de bord, 1 entrée -> 1 sortie en temps fini (cf. Cours Calculabilité ou Preuves de Programmes)
3. Model checking programmes et systèmes informatisés (systèmes cyber physiques, parallèles, ...)

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

- **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.

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 (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.

Bugs dans les systèmes embarqués d'avions (ex. Airbus, Boeing)

- **Contexte** : Les systèmes de contrôle de vol sont critiques.
- **Utilisation réelle** : Airbus a utilisé le model checking (notamment avec l'outil **SPIN**) pour vérifier les propriétés de sécurité de ses systèmes embarqués.
- **Résultat** : Cela a permis d'éviter des erreurs logicielles qui auraient pu avoir des conséquences graves.

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 l'algorithme mathématique 1.

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:
- 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**, ou **TLA+**, 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

- 🛰️ **Étude de cas : L'échec du vol Ariane 5 – 4 juin 1996**
- 🔍 Contexte
  - 🧠 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'une **mauvaise capture des besoins** et d'une **conception 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** et **preuve 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 des algos de model checking est elle un problème?

- Oui et non
- OUI Si les algos de model checking très complexes, c'est un problème. 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 CTL polynomial.
- A la différence d'autres activités informatique, on vérifie **1 fois** le schéma d'un circuit ou d'un système cyber physique si ça prend des mois, pas grave l'important c'est que ce soit sûr (pb vital) -- très différent d'une requête BD ou de la recherche d'information
- Ex train d'atterrissage du rafale. Esterel, Berry années 90.

Exemple concret: Train d'atterrissage du Rafale avec Esterel

•**Esterel** 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).

• ✈️ 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).
  - 📘 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?
de nos jours l'IA est partout...
(même ici: un peu de copilote par-ci par-là ;-)

- Au sens apprentissage bof : 100% sûr --> méthodes exactes
  et si pb on veut pouvoir comprendre et rectifier le système
- Sauf diagnostic: aprentissage du "fonctionnement normal" (spécification implicite) par ex centrale nucléaire -> suite de mesures pression température hygrométrie -> apprentissage d'une grammaire
  puis si suite de mesure hors langage -> pb, alarme etc.
- IA sens modélisation avec des multi agents (c'est aussi de la logique modale): très important pour modéliser les questions d'intrusion, on modélise le savoir des intrus
- IA Thunders Des Agents de Test IA qui Génèrent, Exécutent, Analysent et Corrigent.

UNIVERSITÉ DE MONTPELLIER

Historique

Ancêtre du model checking: parallélisme, système parallèles concurrents (années 70)

- 1978 Communicating Sequential Processes Hoare
  un outil de spécification formelle de l'exécution concurrente de systèmes variés
  communication / diffusions : variables partagées

Je l'avoue volontiers, j'avais trouvé cela plutôt incompréhensible avec une syntaxe vraiment illisible.

- 1980 Calculus of Communicating Systems Milner Les processus communiquent 1-1 sur des canaux
- 1980 Henessy Milner Logic (multimodale temprorelle 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 :-)

Model Checking

- Amir Pnueli découvre la logique temporelle de Arthur Prior (philosophe) à la fin des années 70 aux USA.
  - « 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)

Système Réactif

- 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.

Prix Turing

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)

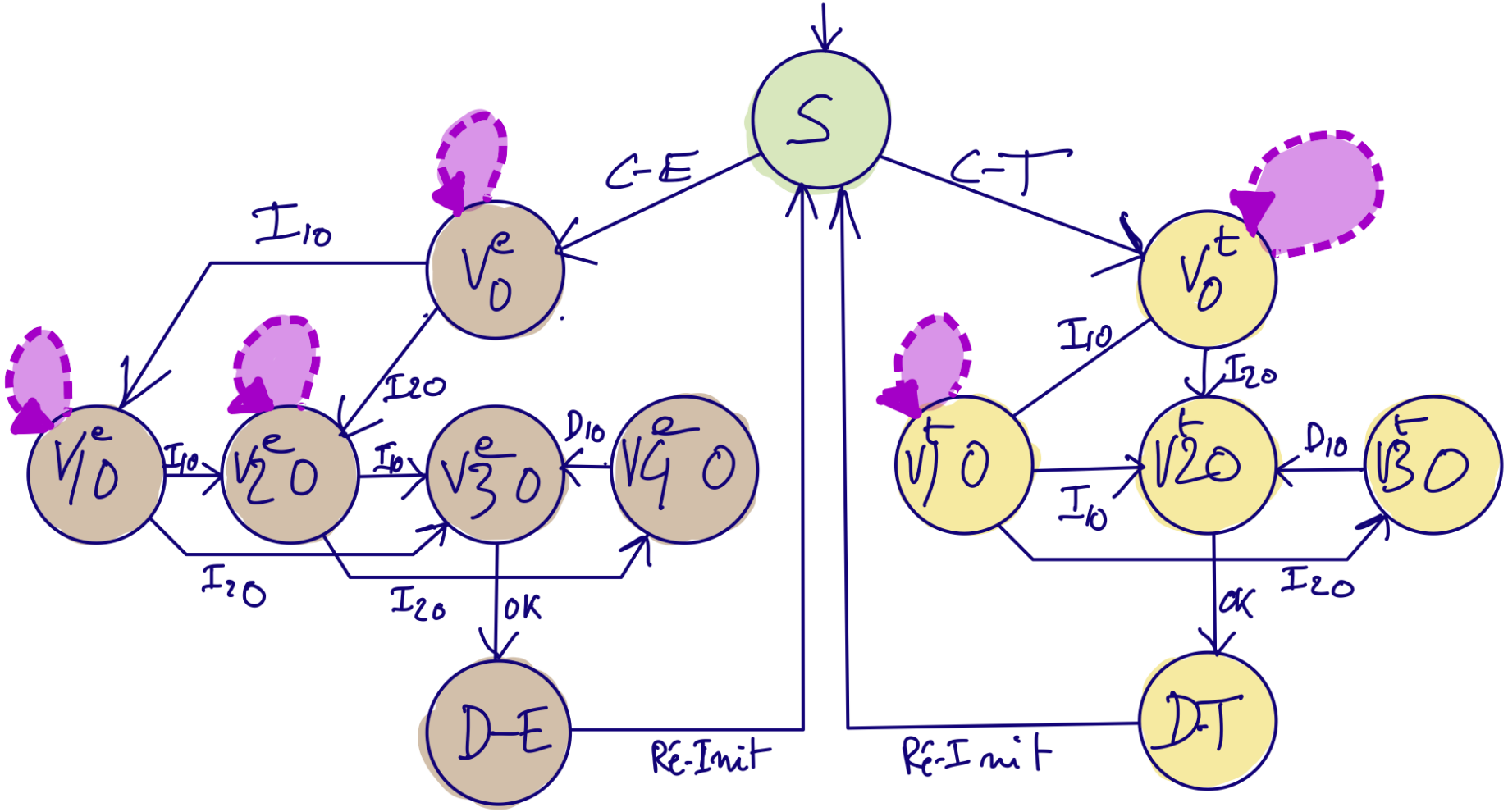
Lamport: merci pour LATEX !!!

UNIVERSITÉ DE MONTPELLIER

LTS Labelled Transtion Systems + Temporal Logic

Petit exemple: machine expresso (30cts ou thé

Boucle si pièce qui dépasse le montant? Ou pas?



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
- Plusieurs sortes d'accessibilités entre états (étiquetées par les actions)
- Dans chaque état, des propositions atomiques (des lettres) sont vraies ou fausses: ces propositions.

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

Logique modale, modèles de Kripke : décrire l'évolution du système.

Pour S état, t transition on définit: S |= α par induction sur la formule

- S |= p (donné)
- S |= α & β ssi (S|= α et S|=β)
- S |= α v β ssi (S|= α ou S|=β)
- S |= α v β ssi (si S|= α alors S|=β)
- S |= [a] α ssi pour tout état S' accessible par a depuis S on a S'|=α
- S |= <a> α ssi pour au moins un état S' accessible par a depuis S on a S'|= α

Déf. LTS |= α ssi pour les états initiaux IS on a IS|= α

Logique modale, modèles de Kripke : décrire l'évolution du système.

Oublions les transitions, on définit: S|= A

Déf. LTS |= α ssi pour les états initiaux IS on a IS|= α

- S |= p (donné)
- S |= α * β ssi (S|= α * S|=β)
- S |= [] α ssi pour tout état S' accessible depuis S on a S'|=A
- S |= <> α ssi pour au moins un état S' accessible depuis S on a S'|=F

Logique modale, modèles de Kripke : décrire l'évolution du système.

Oublions les transitions, on définit: S|= A + clôture transitive

- S |= p (donné)
- S |= α * β ssi (S|= α * S|=β)
- S |= [] α ssi pour tout état S' accessible depuis S on a S'|=A
- S |= <> α ssi pour au moins un état S' accessible depuis S on a S'|=F
- S |= EF α s'il existe un état accessible S' en un nombre fini d'étapes depuis S tel que S' |= α
- S |= AG α si dans tout état accessible S' en un nombre fini d'étapes depuis S tel que S' |= α

Déf. LTS |= α ssi pour les états initiaux IS on a IS|= α

Récapitulatif

Système de transition étiqueté

Logique modale, modèle de Kripke pour parler de son évolution dans le temps.

Pour formuler et vérifier les spécifications attendues il faut être plus précis et plus fin au niveau temporel: sur un chemin, sur tous les chemins, dans un instant suivant, dans tout instant suivant instant futur,…

Il faut spécifier le modèle:

- Temps linéaire (étude d'une exécution sans référence aux autres)
- Temps Branchant (étude simultanée de toutes les exécutions)

Logique propositionnelle temporelle

- Logique propositionnelle: parfait pour exprimer des propriétés
- Mais comment quantifier dans le temps:
  - Si le train approche alors la barrière est fermée.
  - Dans toute exécution si on rencontre ta alors bf à l'instant suivant.
  - Si la barrière est fermée alors plus tard elle sera ouverte.
  - Dans tout état accessible s où bf, il y a un état accessible s' depuis s où bo
- 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

UNIVERSITÉ DE MONTPELLIER

Modèle Checking LTL

+ automates de Büchi

LTL linear temporal logic

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

X next

F à instant futur – maintenant inclus

G à tout instant futur – maintenant inclus

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

R: $\neg(\neg\varphi U \neg\psi)$

LTL interprétation dans un modèle linéaire π

— π,i |= ⊤pour toute position i;

— jamais π,i |= ⊥, pour toute position i;

— π,i |= p ssi p vrai en i;

— π,i |= φ∧ψ ssi π,i |= φ et π,i |= ψ;

— π,i |= φ∨ψ ssi π,i |= φ ou π,i |= ψ;

— π,i |= φ→ψ ssi π,i |= φ entraîne π,i |= ψ;

— π,i |= ¬φ ssi c'est n'est pas le cas que π,i |= φ

LTL interprétation dans un modèle linéaire π

— π,i |= Xφ ssi π,i +1 |= φ;

— π,i |= Fφ ssi il existe j ⩾i tel que π,j |= φ;

— π,i |= Gφ ssi pour tout j ⩾i on a que π,j |= φ;

— π,i |= φUψ ssi il existe k ⩾i tel que π,k |= ψ et π,j |= φ pour tout j tel que i ⩽j <k.

— π,i |= φRψssi pour tout k ⩾i pour tout k ⩾i on a l'une au moins des deux propriétés π,k |= ψ ou il existe j tel que i ⩽j <k et π,j |= ϕ.

Vérification d'une propriété dans toute exécution (infinie!) du LTS

- LTS -> Automate de Büchi A (mêmes exécutions)
- Spécif β = formule LTL
- ¬ β -> Automate de Büchi B
  exécutions de B = tous les modèles linéaires
  où ¬ β est vraie (où β fausse)

- Calcul automate intersection $A \cap B$
  Modèles lineaires communs = exécutions du LTS ne satisfaisant pas β
- Test: langage de $A \cap B$ vide ou pas?
  - Toutes les exécutions du LTS satisfont elles la spécif β ?

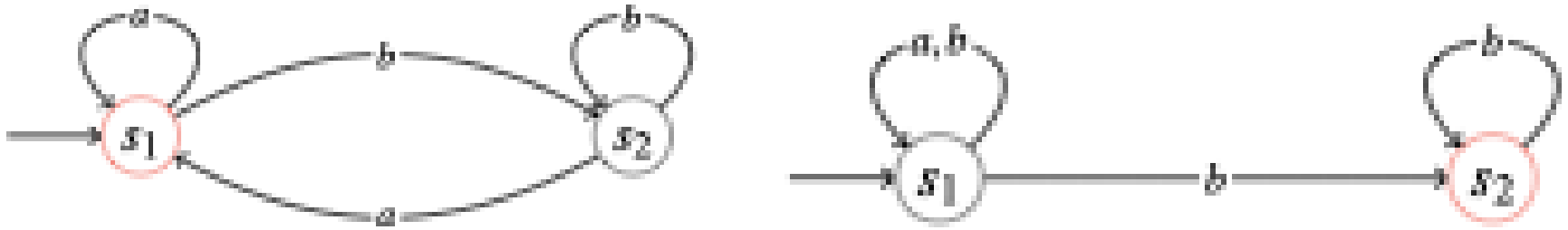
Automate de Büchi

- Un nombre fini d'états
- Un alphabet étiquetant les transitions
- Un ou plusieurs états initiaux
- Des états finaux

- Un mot infini est dit accepté (ou reconnu) lorsqu'il passe une infinité de fois fois par l'un des états finaux.

Automate de Büchi

- Deux exemples: infinité de a / nombre fini de a



- /!\ Celui de droite n'est pas déterminisable.

Intersection de deux automates de Büchi

Un peu plus compliqué que pour les autoamtes finis usuel.

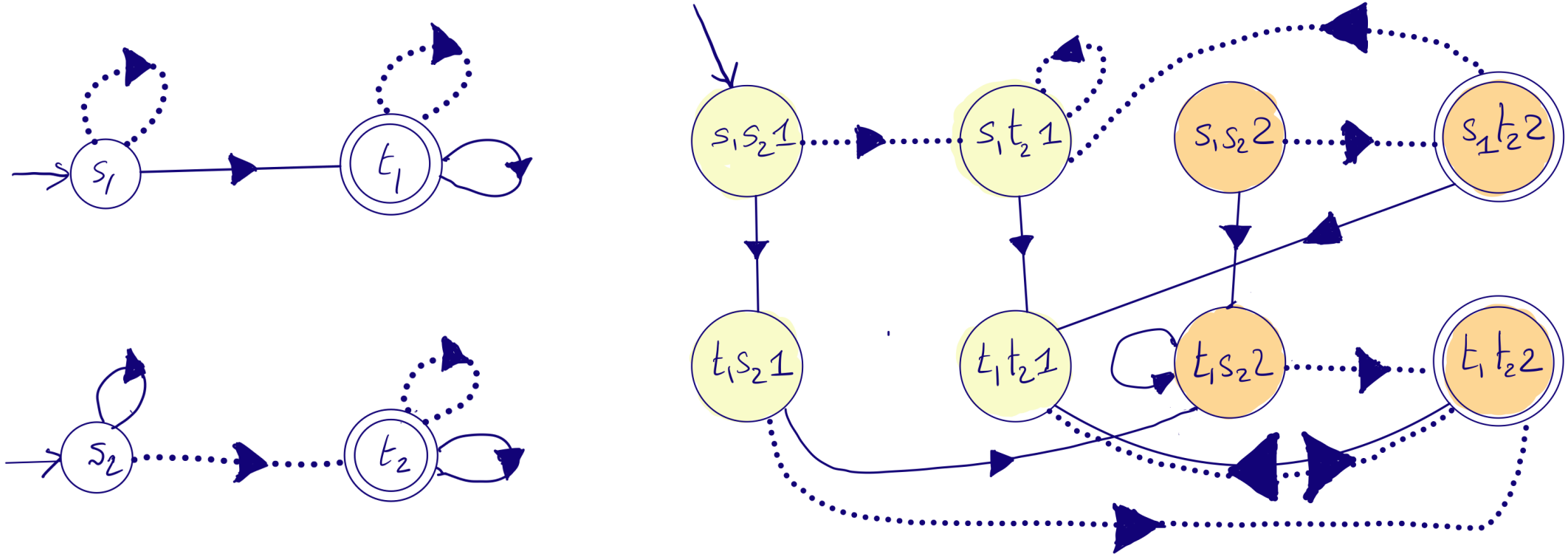
Dans le cas usuel on fait le produit et à la fin du mot il faut être sur un état final égal qui soit un couple (S1,S2) avec S1 final pour A1 et S2 final pour A2.

On fait deux copies I et II du produit des deux automates:

I. Dans la copie I quand on arrive dans un état final pour A1 on repart du même état dans la copie II.
II. Dans la copie II quand on arrive dans un état final pour A2 on repart du même état dans la copie I.

Ainsi le mot qui passe bien une infinité de fois et par un état gagnat pour A1 et par un état gagnant pour A2

Intersection de deux automates de Büchi

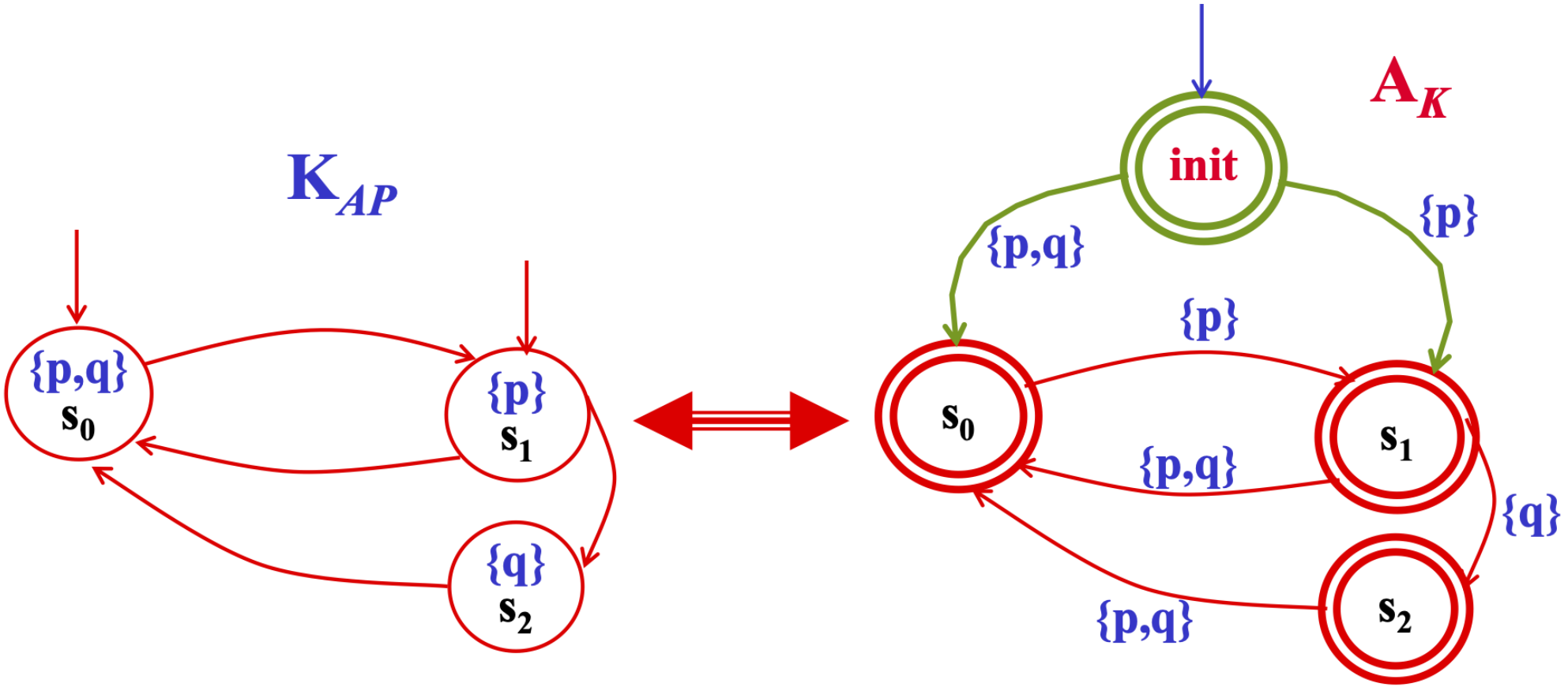


Test à vide d'un automate de Buchi

C'est un graphe fini...

Il suffit de voir s'il y a un circuit accessible sur lequel figure au moins un état final (par ex parcours en profondeur depuis un état initial)

Modèle de Kripke / LTS et automate de Büchi Inversion Etats / Transitions



Formule -> automate de Büchi

Dont les modèles sont exactement les mots infinis reconnu par l'automate.

Par induction sur la formule écrite avec & v operateurs temporels U et R ce qui n'est pas aussi immédiat, négation sur les atomes exclusivement. Par ex. Gα=αU⊥.

Déf compliquée (calcul PSPACE complet en fonction de la taille de la formule)

Démonstration que les mots infinis reconnus par l'automate de G sont exactement les modèles linéaires dans lesquels G est vraie,... c'est vraiment très compliqué.

Outils industriels Model Checking LTL

🛠️ **Outils industriels de model checking avec 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.

UNIVERSITÉ DE MONTPELLIER

Model Checking CTL

Temps arborescent

Système de transition "non étiqueté "-> computational tree

Les noms des transisions sont inutiles et omis

- Nombre fini d'états
- Nombre fini de propositions atomiques
- Plusieurs sortes d'accessibilités entre états (étiquetées par les actions)
- Dans chaque état, les propositions atomiques (des lettres) sont vraies ou fausses.

- Arbre des calculs: arbre infini, branchement non déterminisme.
- Exécution: chemin infini, suite d'états décrits par l'ensemble fini des propositions vraies dans l'état considéré.

LANGAGE CTL

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

- $\varphi ::= \top | p | \neg\varphi | \varphi \wedge \varphi | EX\varphi | AX\varphi | EF\varphi | AF\varphi | EG\varphi | AG\varphi | E\varphi U\varphi | A\varphi U\varphi$

- Exemples:
  - T,s |= EFφ ssi, il existe un chemin ρtel que ρ1 = s et il existe j ⩾1 tel que T,ρj |= φ
  - T,s |= AφUψ ssi pour tout chemin ρ tel que ρ1 = s il existe j ⩾1 tel que T,ρj |= ψ et T,ρk |= φ pour tout 1 ⩽k < j

  •Toute formule de CTL eut s'écrire avec la négation (non restreient aux atomes) avec EX φ EG φ E φ U ψ

MODEL CHECKING CTL

- [[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 les exécutions.
- 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]])

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

MODEL CHECKING CTL

Le calcul du point fixe 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.

Comment dit-on en CTL:

- sur chaque exécution il y a un p suivi d'un q?

Impossible!

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

UNIVERSITÉ DE MONTPELLIER

Model Checking CTL*

Combinaison model checkings LTL et CTL

Les formules de CTL*

- A sur tout chemin partant de l'état courant
- E sur au moins un chemin partant de l'état courant
- X dans l'états suivant, F dans un états futurs, G dans tous les états futurs, G U 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.
- A F (p&Xq)
- Sur tout chemin il y aura dans le futur un p suivi de q (même chemin)
- ≠ 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*

- 1. **Formules d'état** (state formulas)
  - Ce sont les formules qui peuvent être évaluées dans un état donné.
  - Exemples :
    - pp (une proposition atomique)
    - ¬ϕ
    - ϕ∧ψ
    - Aψ: "pour **tous les chemins** à partir de l'état courant, la formule de chemin ψψ est vraie"
    - Eψ : "il **existe un chemin** à partir de l'état courant où ψψ est vraie"
- 2. **Formules de chemin** (path formulas)
  - Ce sont les formules qui s'appliquent à des chemins (séquences d'états).
  - Exemples :
    - Xϕ : "dans le **prochain état**, ϕ est vraie"
    - Fϕ: "dans le **futur**, ϕ sera vraie"
    - Gϕ : "ϕ est **toujours** vraie"
    - ϕUψ: "ϕ est vraie **jusqu'à ce que** ψ le soit"

Model checking CTL*

- Algorithme de réduction de CTL* à LTL :

1 : Identifier les sous-formules d'état maximales
2 : Pour chaque sous-formule d'état ψ :
- Calculer sa dénotation ⟦ψ⟧ avec l'algorithme CTL
- Remplacer ψ par un nouvel atome p_ψ
- Ajouter la contrainte p_ψ ≡ ⟦ψ⟧
3 : Obtenir une formule LTL pure
4 : Vérifier la formule LTL sur le modèle
5 : Combiner le résultats avec les contraintes

Model checking CTL*

- Exemple concret :formule CTL* originale : A[X (EX p U A[F q])]

  Sous-formules d'état identifiées :
  ψ_1 = EX p → calcul ⟦EX p⟧ → remplace par a
  ψ_2 = A[F q] → calcul ⟦A[F q]⟧ → remplace par b

  Formule LTL résultante : A[X (a U b)]

  Contraintes :
  a ≡ EX p (vérifié avec algorithme CTL)
  b ≡ A[F q] (vérifié avec algorithme CTL)

  Résultat final = états satisfaisant A[X (a U b)] ∩ états respectant toutes les contraintes

UNIVERSITÉ DE MONTPELLIER

Tendances actuelles: lutte contre les intrusions

Lutte contre les intrusions / logique modale temporelle + multiagents

Système 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.

Système Multi-Agents

- 📚 **Quand sont apparues les logiques épistémiques ?**

•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**

Système Multi-Agents Logique Modale Epistémique + Temporelle

- 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 combinasons de logique SMA (genre S5) ert temproelle pour faire de la vérif contre les intrusions.**

UNIVERSITÉ DE MONTPELLIER

Des questions?