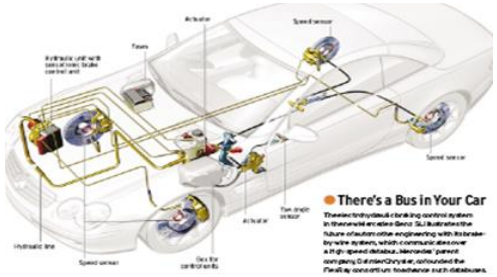
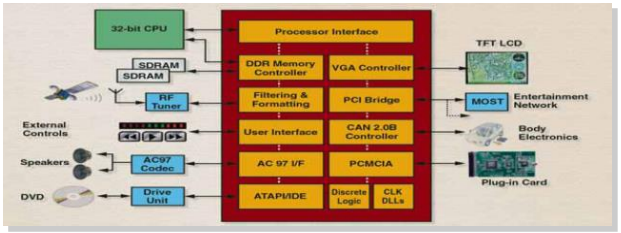
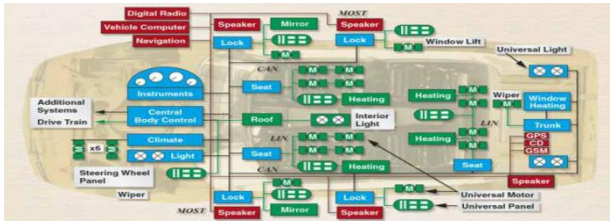
Validation des systèmes embarqués
Model checking

Nga Nguyen

Validation of Safety-Critical Embedded Systems







Do you trust your car?

Edmund Clarke

Turing Award Lecture – MODEL CHECKING

Aerospace Systems: Software Driven!

Mars Polar Lander (1999)
landing-logic error

Mission Loss

Spirit Mars Rover (2004)
file-system error

Airbus A380 Flight Deck

Do you trust flight software?

Clarke Turing Award Lecture

Intel Pentium FDIV Bug

- Try 4195835 – (4195835 / 3145727) * 3145727 = ?

  In ’94 Pentium, it doesn’t return 0, but 256.
- Intel uses the SRT algorithm for floating point division. Five entries in the lookup table are missing.
- Cost: **$400 - $500 million**

Objectifs

- Conception de systèmes fiables :
  - Spécification
  - Vérification et validation automatique

Objectifs

- Conception de systèmes fiables :
  - Spécification
  - Vérification et validation automatique

Méthode	Phase (conception ou codage)	Prise en main	Assisté par ordinateur	Difficulté / problèmes	Debug	Validation
Preuve de théorème						
Model checking						
Test basé modèle						
Analyse statique						
Testing						

Objectifs

- Conception de systèmes fiables :
  - Spécification
  - Vérification et validation automatique

Méthode	Phase (conception ou codage)	Prise en main	Assistée par ordinateur	Difficulté / problèmes	Debug	Validation
Preuve de théorème	conception	--	-	preuves	-	++
Model checking	conception	+	+	modélisation	+	+
Test basé modèle	conception	+	+	modélisation	+	-
Analyse statique	code	++	++	faux négatifs	+	+
Testing	code	++	+	couverture	++	-

Références

1. Model Checking, Edmund M. Clarke et al., Carnegie Mellon University
2. Principles of Model Checking, Joost-Pieter Katoen, University of Twente
3. Model Checking: A Hands-On Introduction, A. Cimatti, M. Pistore, and M. Roveri, http://nusmv.irst.itc.it/courses
4. Introduction au Model Checking, Sébastien Bardin, CEA, ENSTA
5. NuSMV 2.6 Tutorial
6. …

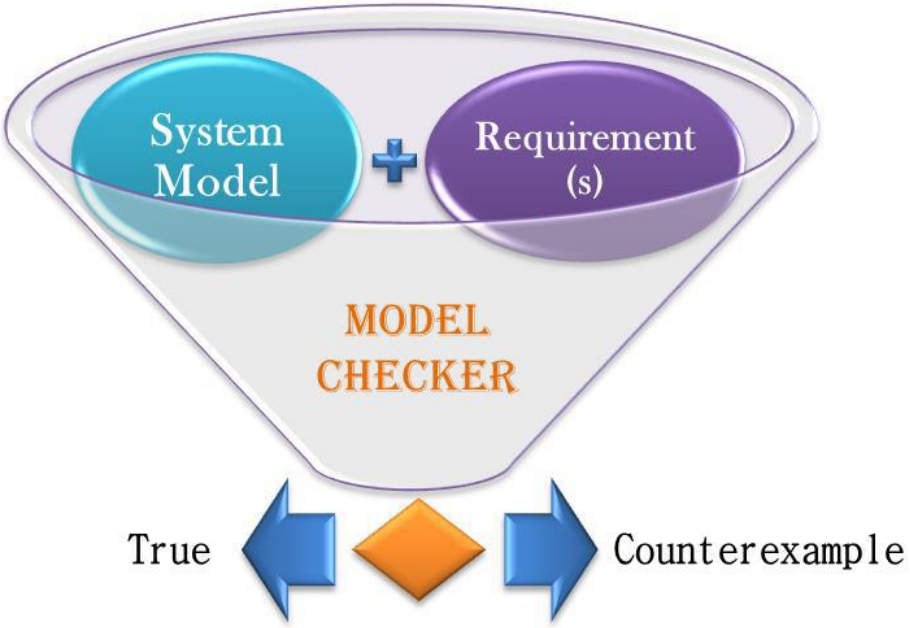
		Planning
1	Mardi 17/1	Introduction au Model checking et Logiques temporelles
2	Mardi 24/1	Modélisation des systèmes concurrents : structure Kripke
3	Mardi 7/2	NuSMV
4	Mardi 21/2	NuSMV
5	Mardi 7/3	NuSMV
6	Mardi 14/3	NuSMV - TP noté
7	Mardi 21/3	Article VAS
8	Mardi 28/3	Soutenance Article VAS

Cours 1 : Introduction

- Model checking
  - Avantages
  - Limites
- Systèmes concurrents / réactifs
- Propriétés temporelles

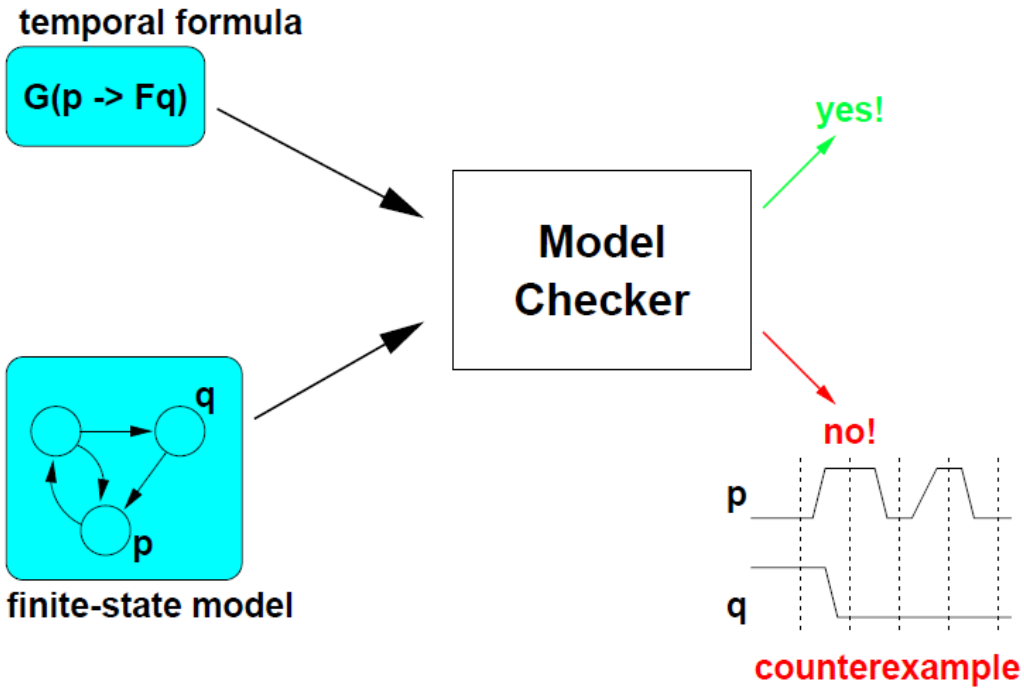
Model checking

- **Définition** : un ensemble de techniques de vérification automatique de propriétés temporelles sur des systèmes réactifs/concurrents.



Model checking

- **Définition** : un ensemble de techniques de vérification automatique de **propriétés temporelles** sur des **systèmes réactifs/concurrents.**



Systèmes réactifs

Programme standard

Exemples :

compilateur, algorithme de tri, …

Propriétés :

1) Termine
2) Retourne un résultat
3) Manipule des données complexes mais la structure de contrôle est assez simple

Système réactif

Exemples :

systèmes embarqués, systèmes d'exploitation, protocoles de communication, …

Propriétés :

1) Ne termine pas forcément
2) Ne calcule pas un résultat mais plutôt maintient une interaction
3) Manipule des données souvent simples mais le contrôle est complexe :
   - exécution de plusieurs composants en parallèle
   - interaction avec un environnement (capteurs + actionneurs)

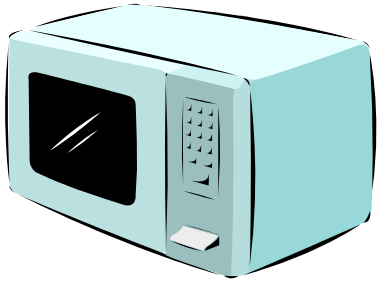
Propriétés temporelles

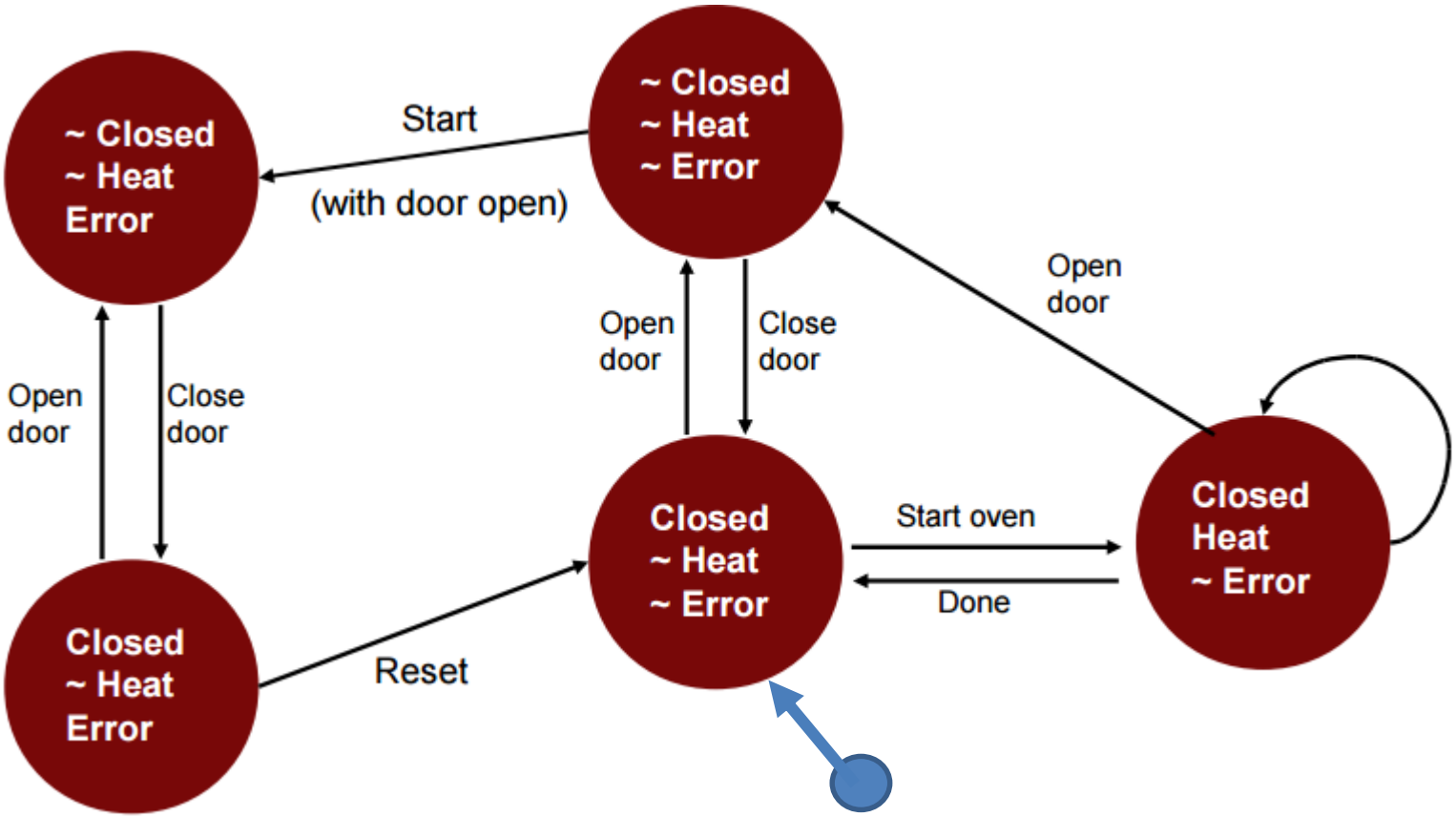
- Les propriétés que l'on veut prouver sur les systèmes réactifs :
  - Si un processus demande infiniment souvent à être exécuté, alors l'OS finira par l'exécuter
  - Il est toujours possible lors de l'exécution de revenir à l'état initial
  - Chaque fois qu'une panne est détectée, une alarme est émise
  - Chaque fois qu'une alarme a été émise, une panne avait été détectée
- Logiques temporelles :
  - Linear Temporal Logic (LTL)
  - Computation Tree Logic (CTL)
  - et CTL*

Model checking

- Soit ***M*** un **graphe d'états-transitions**
- Soit ***f*** une **spécification** en logique temporelle
- Trouver tous les états ***s*** de ***M*** pour que ***M, s |= f***

Exemple de micro-onde





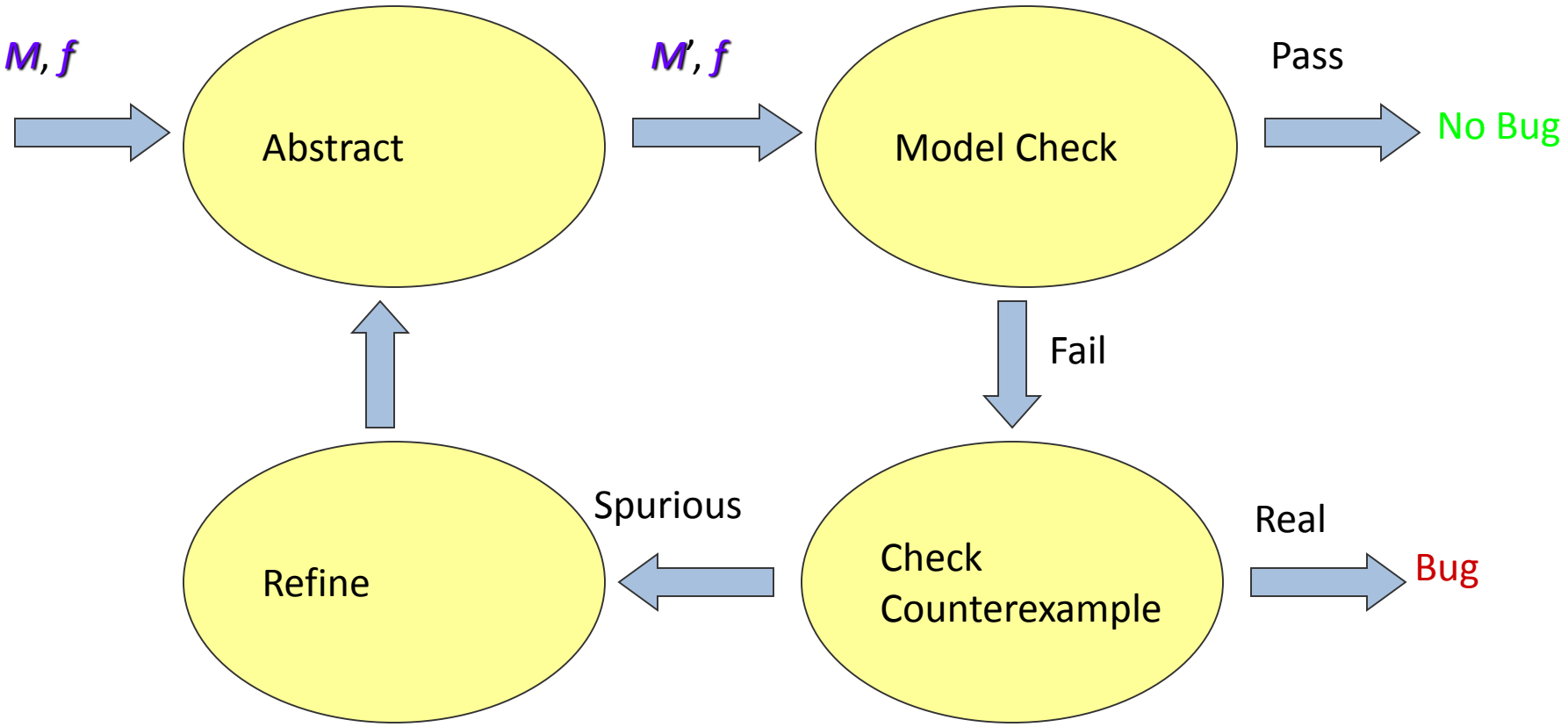
Exemple de micro-onde

- Le micro-onde ne chauffe pas (heat up) jusqu'à ce que la porte soite fermée
- Not heat_up holds until door_closed
- (~ heat_up) U door_closed

Processus du model checking

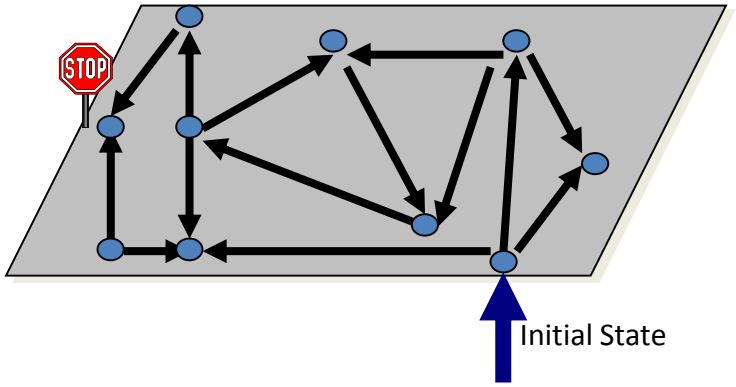
1. Modéliser :
   a) le système (abstraction du comportement du système réactif - système de transitions ***M***)
   b) la spécification (logique temporelle ***f***).
2. Vérifier si ***M*** satisfait ***f*** ou non (recherche exhaustive). Si non, retourner un contre-exemple.
3. Analyser les résultats obtenus :
   a) Si oui, le modèle ***M*** est sûr. *Attention : le système réel est-il sûr pour autant ?*
   b) Si non, rejouer le contre-exemple sur le système réel.
      i. Si c'est un vrai bug, avertir le concepteur et attendre qu'il corrige.
      ii. Si le bug vient de la modélisation, repartir à (1) en raffinant le modèle grâce au faux bug.

Boucle d'abstraction – raffinement



Avantages

- Automatique
- Contre-exemple lorsqu'une propriété n'est pas vérifiée

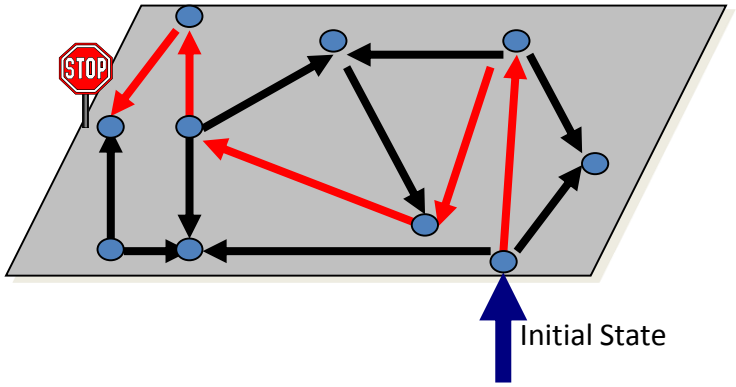


Safety Property:

bad state  unreachable

Avantages

- Automatique
- Contre-exemple lorsqu'une propriété n'est pas vérifiée



Safety Property:

bad state  unreachable

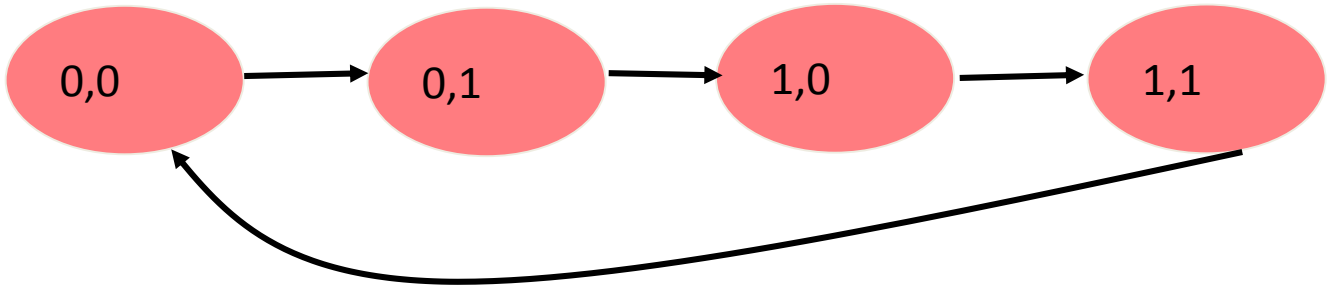
Counterexample

Limites

- Système de transitions doit être fini (limite théorique)
- Explosion combinatoire du nombre d'états du système (limite pratique)

Limites

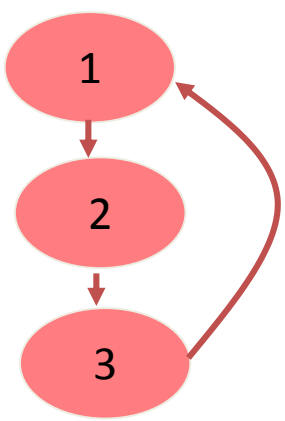
State Explosion Problem:

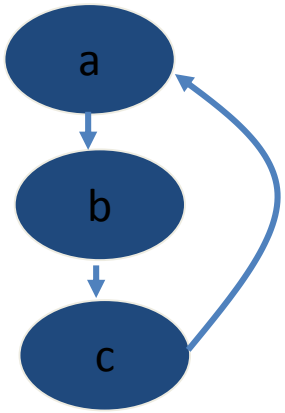


2-bit counter

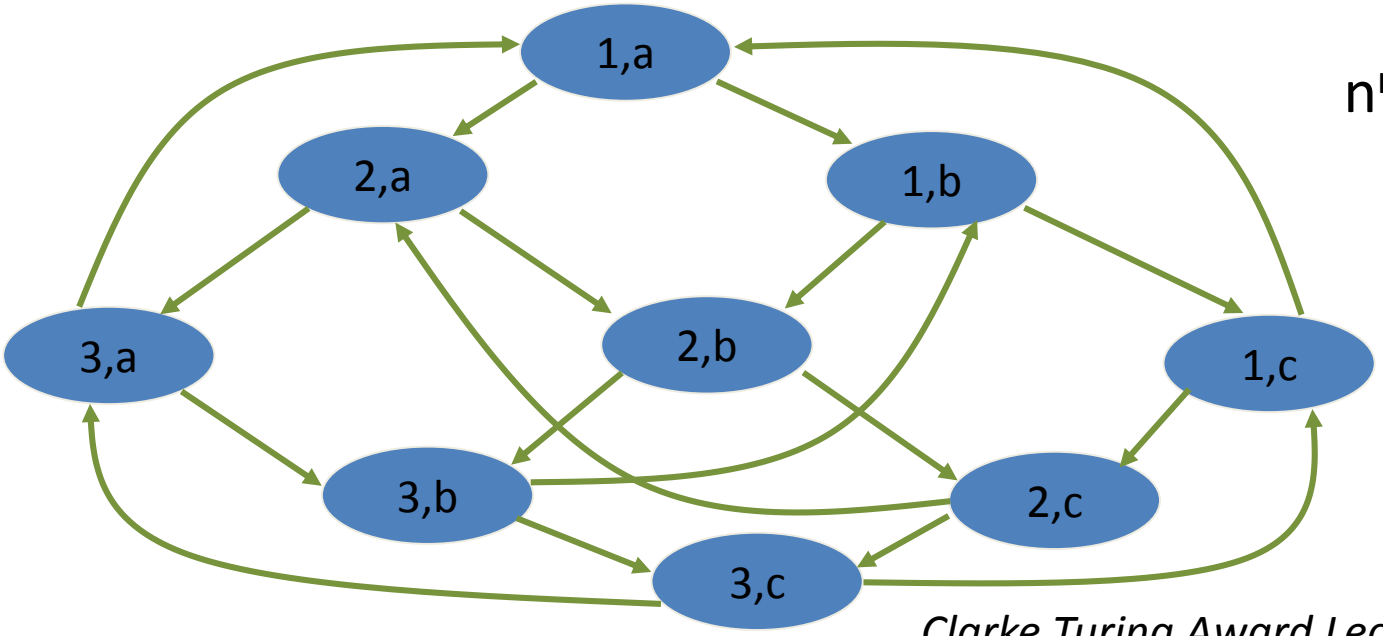
n-bit counter has 2^n states

Limites





n states,
m processes



n^m states

Limites

- Système de transitions doit être fini (limite théorique)
- Explosion combinatoire du nombre d'états du système (limite pratique)
  - Nombre de variables et leur taille :
    - 10 variables codées sur 8 bits : 10^255 états !
    - **Model checking symbolique par BDD (Binary Decision Diagram)**
  - Nombre de composants du système
    - **Techniques d'ordres partiels**

Historiques

- 1980 : résultats théoriques
  - Pnueli, 1977 : logique temporelle
  - Clarke et Emerson, Queille et Sifakis, 1981 : algorithmes de model checking
- 1990 : techniques pour combattre le phénomène d'explosion combinatoire
- 2000 : adoption industrielle
  - Validation des composants électroniques (Siemens, Bull, IBM, Lucent Technologies, Cadence, Intel, Motorola)
  - Validation des protocoles de communication (IEEE Futurebus+, commerce électronique, …)
  - Software model checking : combinaison avec l'analyse statique, génération de tests, preuves (Microsoft)

"Things like even software verification, this has been the ***Holy Grail*** *of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability."* **Bill Gates, April 18, 2002.**

Outils

- https://en.wikipedia.org/wiki/List_of_model_checking_tools

- SMV (Carnegie Mellon University) :
  - Model checking symbolique par BDD
  - Langage de modélisation : SMV (Symbolic Model Verifier)
  - Logique temporelle : CTL, LTL
  - NuSMV, PVS (theorem prouver + model checker)

- SPIN (Bell Labs) :
  - Ordres partiels
  - Langage de modélisation : Promela
  - Logique temporelle : LTL

Exemple 1 : IEEE Futurebus+

- En 1992, Clarke et ses étudiants au CMU ont utilisé SMV pour vérifier la cohérence de cache du protocole IEEE Futurebus+ Standard
- Ils ont construit un modèle précis du protocole et essayé de montrer qu'il satisfait une spécification formelle de cohérence de cache
- Ils ont trouvé un nombre d'erreurs non détectées dans la conception du protocole
- C'était la 1ère fois que les méthodes formelles ont été utilisées pour trouver des erreurs dans un standard IEEE
- Bien que le développement a été commencé en 1988, les tentatives précédentes pour valider Futurebus+ ont été basées sur les techniques informelles.

Exemple 2 : HDLC

- High-level Data Link Controller (HDLC) a été conçu chez AT&T à Madrid
- En 1996, des chercheurs chez Bell Labs ont proposé de vérifier certaines propriétés de la conception (bientôt finie, pas d'erreurs attendues)
- Pendant 5 heures, 6 propriétés ont été spécifiées et 5 ont été vérifiées (FormalCheck)
- 6ème propriété échoue, relevant un bug qui risque de réduire le débit ou causer une perte de transmission
- L'erreur a été corrigée en quelques minutes et puis vérifiée formellement

Model Checking NASA Missions

- **Spin Model Checker + Promela**
- Applied to **most missions launched** in the last decade.

Mars Curiosity

Logiques temporelles

- Logique linéaire : LTL
- Logique branchant : CTL, CTL*
- Panorama de propriétés temporelles

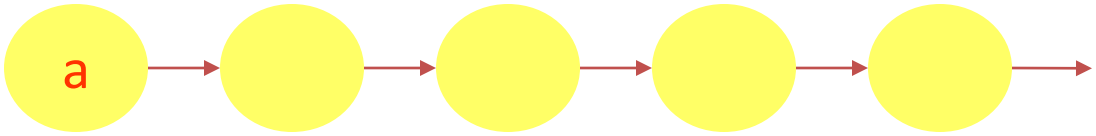
Logique temporelle

- Opérateurs temporels :
  - X : neXt
  - F : Future (finally, eventually)
  - G : Globally
  - U : Until
- Quantificateurs de chemin :
  - E : Exists (possibly)
  - A : All (inevitably)

LTL - Linear Time Logic

Determines Patterns on Infinite Traces

Atomic Propositions
Boolean Operations
Temporal operators

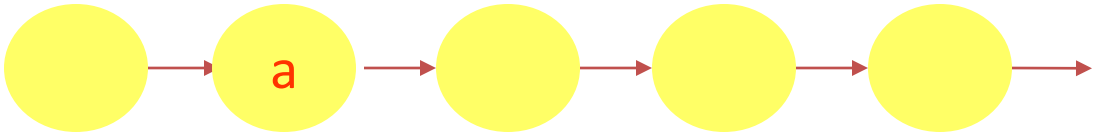


→	**a**	**“a is true now”**
	X a	“a is true in the neXt state”
	F a	“a will be true in the Future”
	G a	“a will be Globally true in the future”
	a U b	“a will hold true Until b becomes true”

LTL - Linear Time Logic

Determines Patterns on Infinite Traces

Atomic Propositions
Boolean Operations
Temporal operators



	a	"a is true now"
→	**X a**	**"a is true in the neXt state"**
	F a	"a will be true in the Future"
	G a	"a will be Globally true in the future"
	a U b	"a will hold true Until b becomes true"

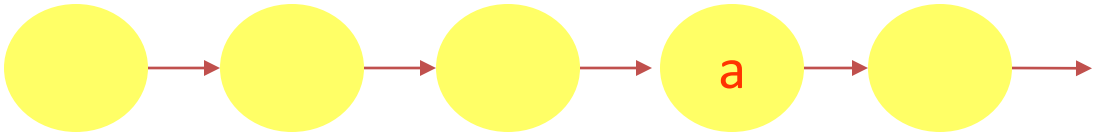
LTL - Linear Time Logic

Determines Patterns on Infinite Traces

Atomic Propositions

Boolean Operations

Temporal operators



	a	"a is true now"
	X a	"a is true in the neXt state"
→	**F a**	**"a will be true in the Future"**
	G a	"a will be Globally true in the future"
	a U b	"a will hold true Until b becomes true"

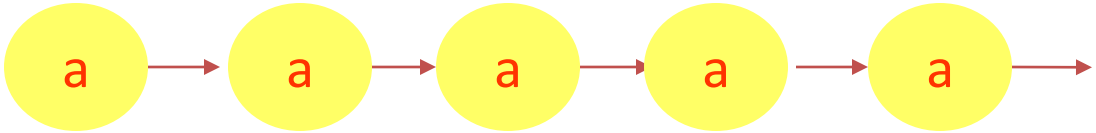
LTL - Linear Time Logic

Determines Patterns on Infinite Traces

Atomic Propositions

Boolean Operations

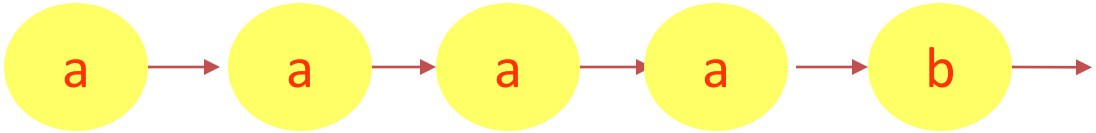
Temporal operators



	a	"a is true now"
	X a	"a is true in the neXt state"
	F a	"a will be true in the Future"
→	**G a**	**"a will be Globally true in the future"**
	a U b	"a will hold true Until b becomes true"

LTL - Linear Time Logic

Determines Patterns on Infinite Traces



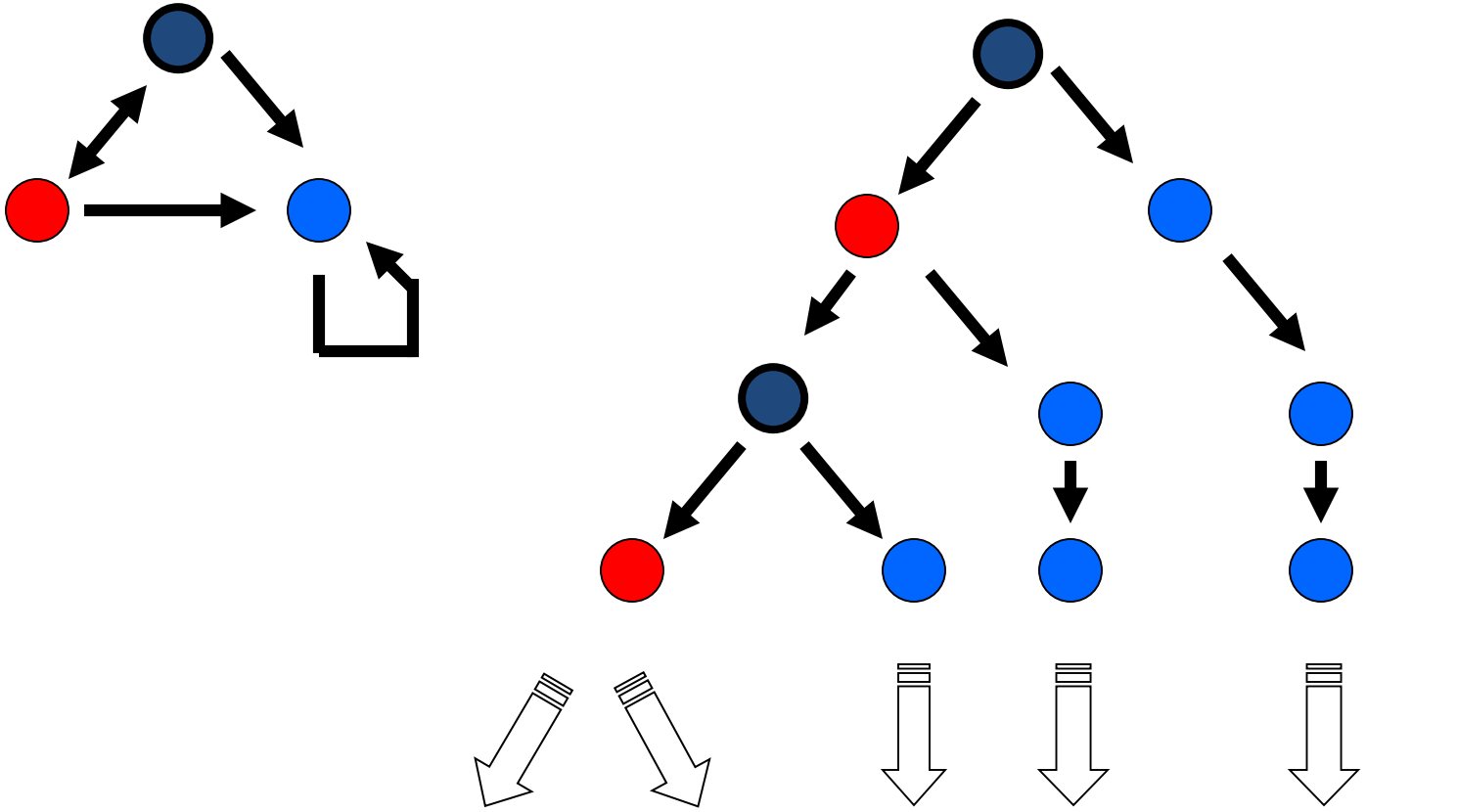
Atomic Propositions

Boolean Operations

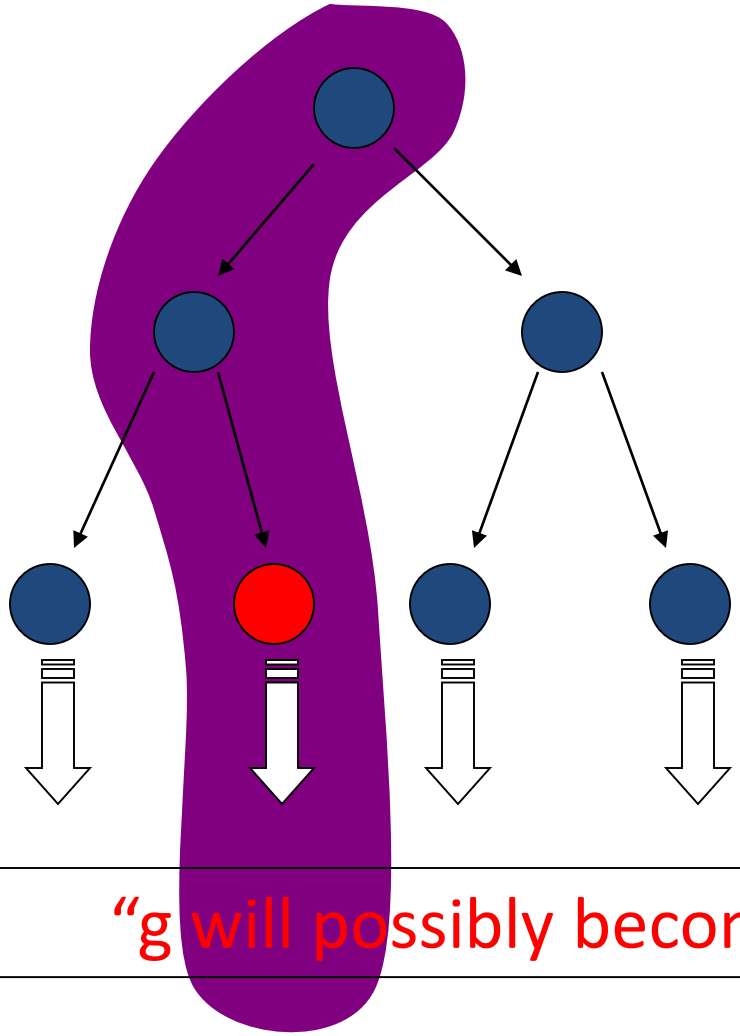
Temporal operators

	a	"a is true now"
	X a	"a is true in the neXt state"
	F a	"a will be true in the Future"
	G a	"a will be Globally true in the future"
→	**a U b**	**"a will hold true Until b becomes true"**

Branching Time

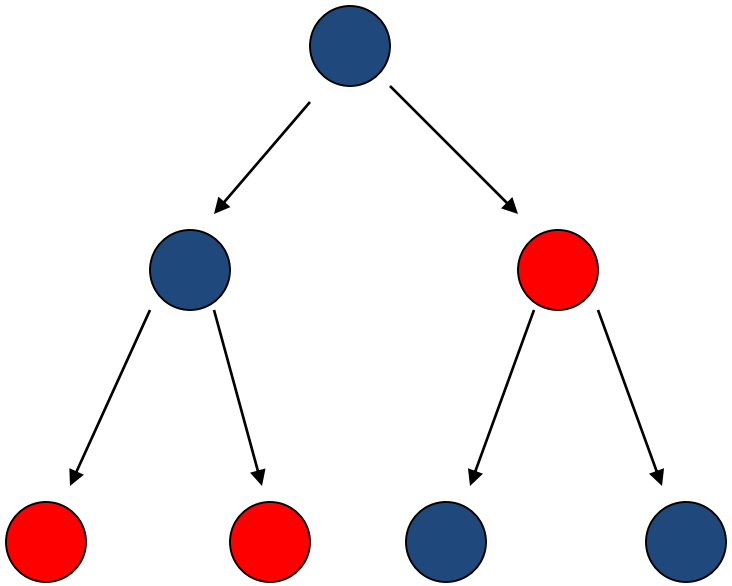


CTL: Computation Tree Logic



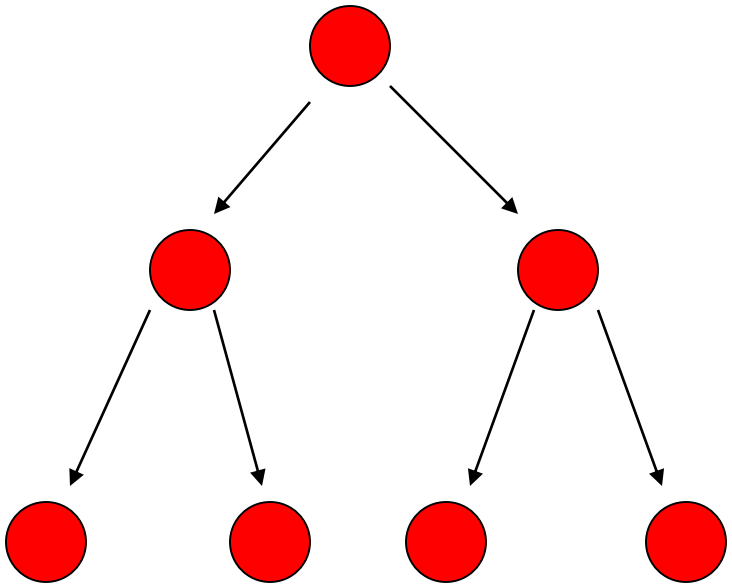
EF g "g will possibly become true"

CTL: Computation Tree Logic



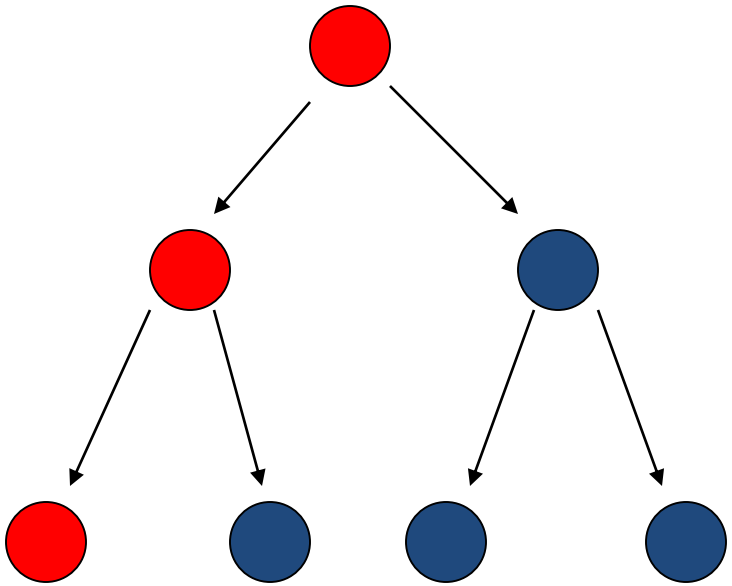
AF g	“g will necessarily become true”

CTL: Computation Tree Logic



AG g “g is an invariant”

CTL: Computation Tree Logic



EG g “g is a potential invariant”

Exercices

- AX p
- EX p
- A pUq
- E pUq

CTL: Computation Tree Logic

CTL uses the temporal operators

AX, AG, AF, AU

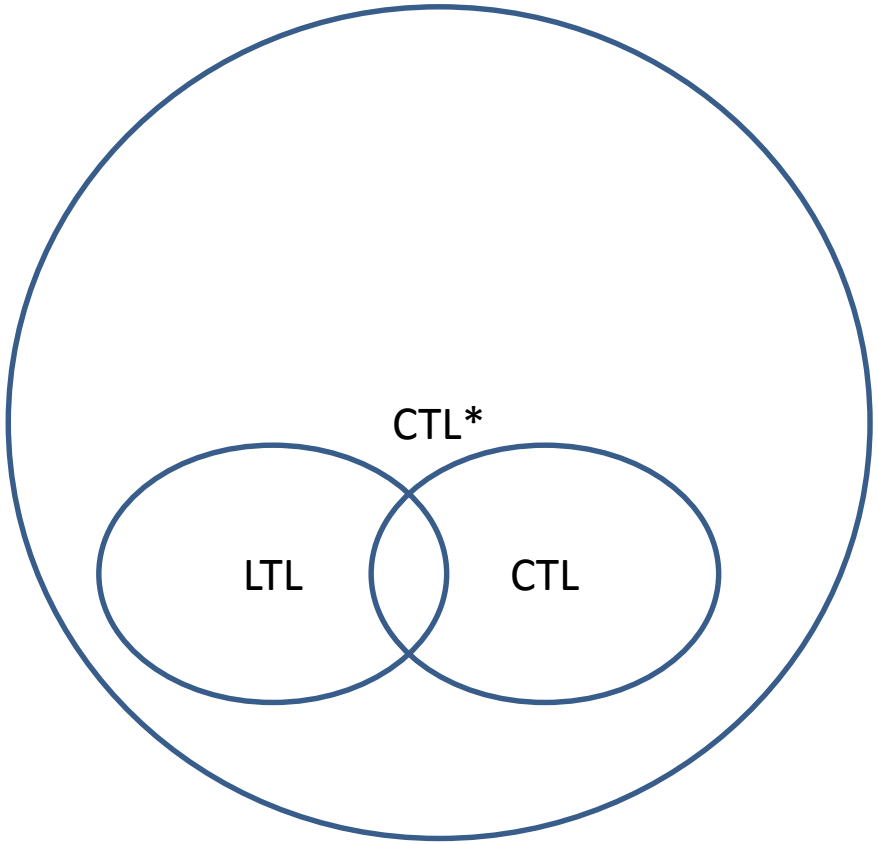
EX, EG, EF, EU

CTL* allows complex nestings such as

AXX, AGX, EXF, …

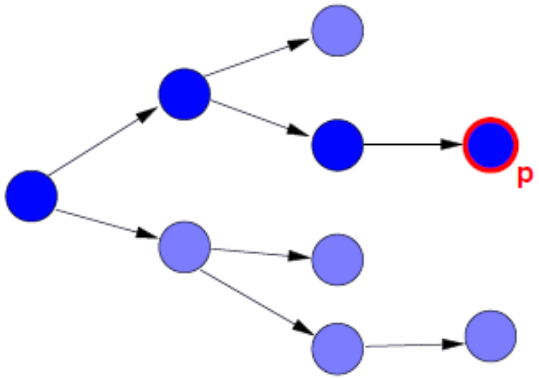
CTL: linear model checking algorithm !

Hiérarchie



Propriétés temporelles des systèmes réactifs

- **Sûreté (safety) :** quelque chose de mauvais n'arrive jamais : AG ~bad
  - deadlock : 2 processus s'attendent mutuellement
  - exclusion mutuelle : 2 processus entrent en section critique en même temps
  - (correction partielle) quand la précondition du programme est respectée et que le programme termine alors la postcondition est respectée
- Peut être invalidée par un comportement fini



Propriétés temporelles des systèmes réactifs

- **Vivacité (liveness):** quelque chose de bon finira par arriver : AG (p -> AF q)
  - ✓ quand une impression est lancée, elle finira par s'achever
  - ✓ quand un message est envoyé, il finira par être reçu
  - ✓ (correction totale) quand la précondition du programme est respectée, alors le programme termine et la postcondition est respectée
- Peut être invalidée par un comportement infini (boucle)