Qualité du logiciel : certification vs. vérification

Gérard Berry

INRIA Sophia-Antipolis

Collège de France
Chaire informatique et sciences numériques

Gerard.Berry@inria.fr
www-sop.inria.fr/members/Gerard.Berry

Journées nationales du GDR GPL, Pau, Mars 2010

Agenda

1. Alerte aux pucerons !
2. Qu'est-ce qu'un bug ?
3. L'approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion

Agenda

1. Alerte aux pucerons !
2. Qu'est-ce qu'un bug ?
3. L'approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion

Alerte aux pucerons!

- infestation massive par pucerons enfouis partout
- qui apprennent à se coordonner par radio
- de plus en plus d'applications critiques

Applications et contraintes variées

pilotage, frein, distribution électrique, carburant, etc.
safety-critical => certification

trajectoire et attitude, imagerie, transmission
mission-critical => qualité supérieure

téléphone, audio, TV, DVD, jeux, ...
business critical => time-to-market

pacemakers, contrôle du glucose, robots chirurgiens,...
life-critical => TBD (!)

Parallélisme

- Les systèmes embarqués comportent toujours des activités fondamentalement parallèles

  compactes, temps-réel fort : pilotage 3 axes
  freinage 4 roues
  réception / émission radio

  distribuées, temps réel faible : contrôle aérien
  surveillance des fonctions

- Le parallélisme a aussi des raisons internes

  répartition des calculs

  redondance des équipements

Déterminisme ou non-déterminisme

- Déterminisme : réactions toujours identiques aux mêmes sollicitations

  freinage, contrôle de vol, comptage de roues de train

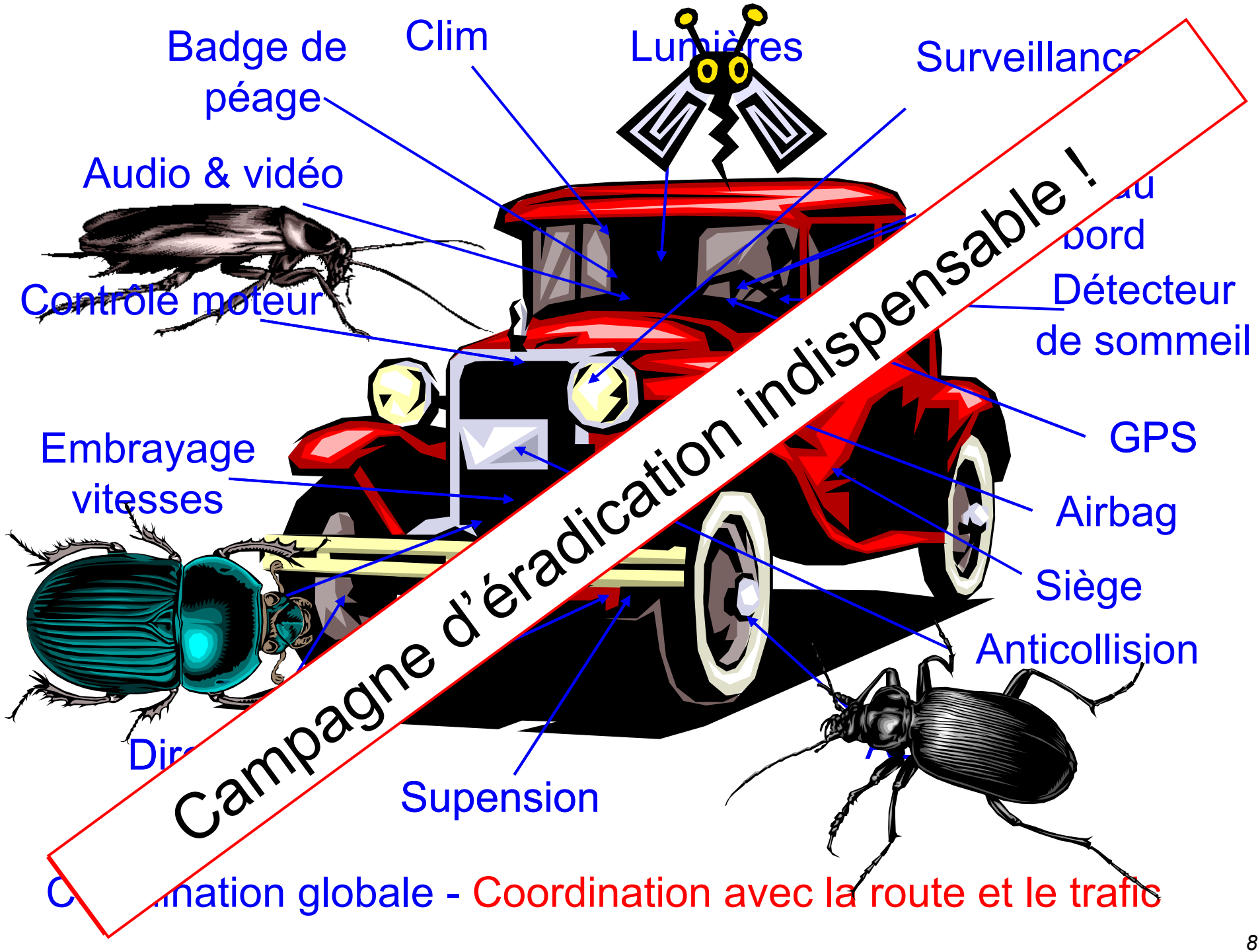
- Non-déterminisme : réactions non constantes

  accès Web, tirage au sort

Non-déterminisme externe: celui de l'environnement (toujours présent)

≠

Non-déterminisme interne : celui du système (rarement souhaitable)

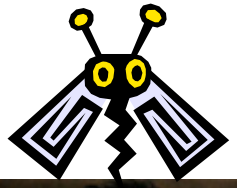

Badge de péage
Clim
Lumières
Surveillance
Audio & vidéo
Contrôle moteur
Détecteur de sommeil
GPS
Embrayage vitesses
Airbag
Siège
Anticollision
Supension
Campagne d'éradication indispensable !
Coordination globale - Coordination avec la route et le trafic

Pourquoi l'éradication reste difficile

- La complexité des applications va en croissant
  - de plus en plus de fonctions, d'algorithmes, d'interactions
  - spécifications globales et locales très difficiles à écrire
- La recherche de performance accroît la complexité
  - répartir les fonctions accroît le non-déterminisme
  - minimiser la puissance dissipée complique la logique
- La vérification est le vrai goulot d'étranglement
  - le non-déterminisme fait exploser le nombre de tests
  - l'interconnection des parties est très difficile à tester
  - idem pour l'interaction matériel / logiciel
  - la répartition simplifie le matériel, pas la vérification !

Agenda

1. Alerte aux pucerons !
2. Qu’est-ce qu’un bug ?
3. L’approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion



Intuition

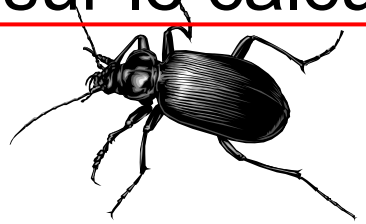
Rigueur

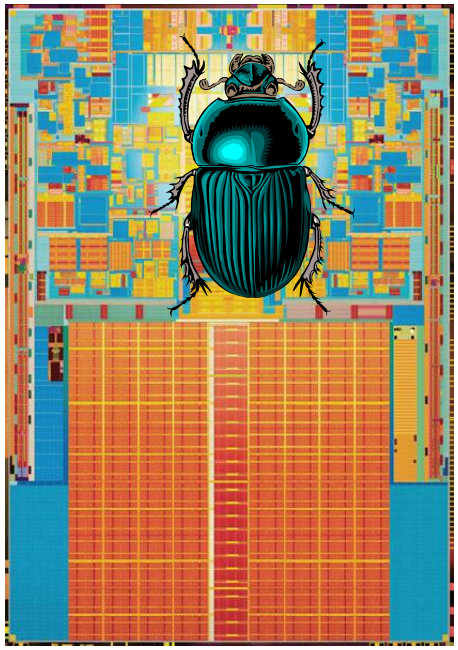
Lenteur

Modèles de calcul

Maîtrise ?

Calculer sur le calcul

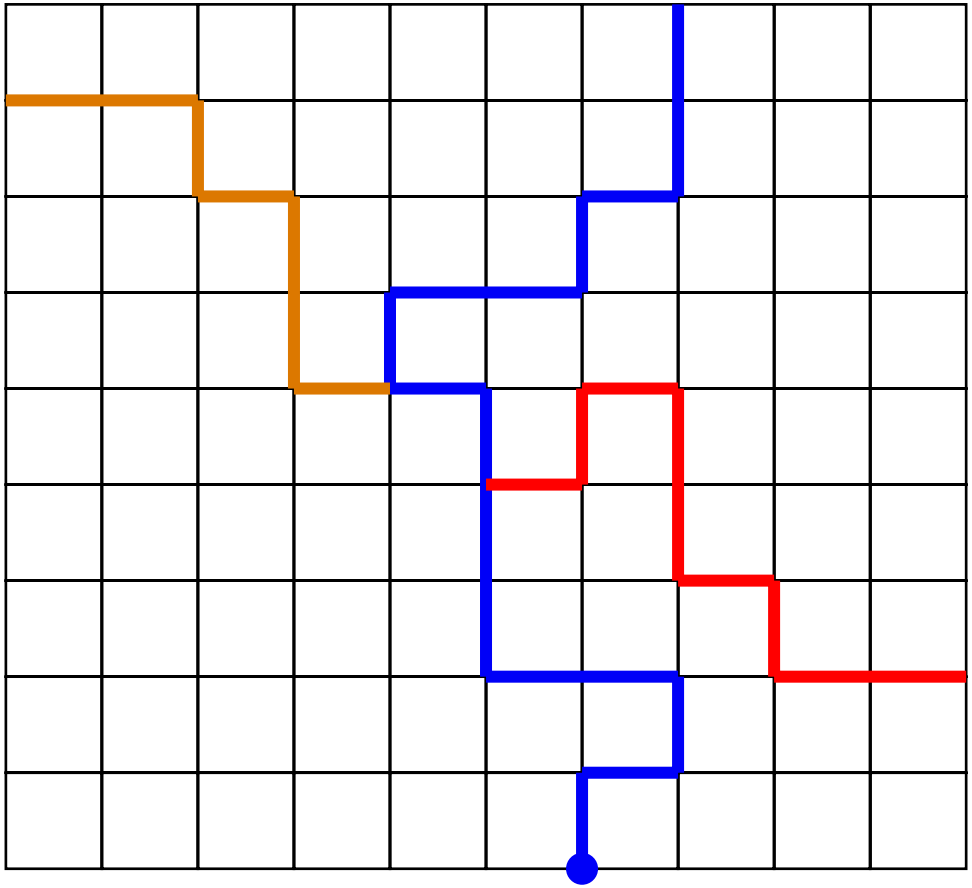




Rapidité

Exactitude

Stupidité



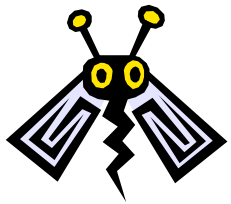
TDGGTDTTGDDTGDGT

TDGGTDTDGDDTGDGT

TDGGTDTTGTDTGDGT

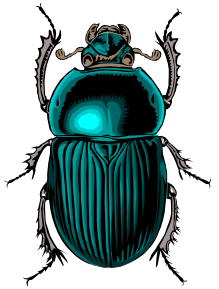
Question pour PDG

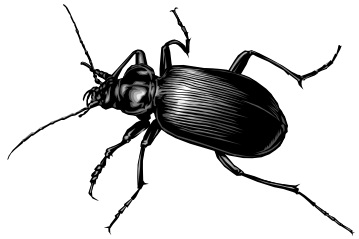
Sauriez vous faire marcher votre entreprise si tous les employés faisaient exactement ce qu'ils sont censés faire, à l'heure dite?



Travaux pratiques

- Plantages des ordinateurs, ATM, PS3, etc.
  systèmes de réservation, téléphones, sites Web, etc.
- Micro-bugs navigateurs Web => macro-attaques pirates
- Blocages d'appareils photo et téléphones
- Grattages de tête des conducteurs et garagistes
- Crash du téléphone interurbain américain
  une accolade mal placée
- Explosion d'Ariane 501, pertes de satellites
  Ariane : débordement arithmétique dans un calcul inutile
- Bug subtil dans la division flottante du Pentium
  coût : 470 millions de dollars pour Intel
- Bugs des satellites et Rovers martiens





Comment éviter ou contrôler les bugs?

1. De la vérification, encore et toujours
   mais peut-on vraiment en faire plus?
2. De meilleures techniques de design
   modèles de calcul plus simples et plus adaptés
   spécifications formelles lisibles par tous les acteurs
3. De meilleurs outils
   exécution immédiate des spécification, maquettes virtuelles
   génération automatique des circuits et codes embarqués
   vérification formelle des propriétés critiques

Approche à la fois scientifique et industrielle

- Utiliser des techniques rigoureuses de génie logiciel
  - documentation, revues de code, tests intensifs
  - certification externe (avionique)
- Rendre visuel ce qui est invisible
  - environnements de débogage
  - prototypes virtuels animés
- Utiliser des méthodes formelles
  - algorithmes validés mathématiquement
  - langages de programmation plus abstraits
  - et formellement définis
  - compilateurs certifiés ou vérifiés formellement

Calculer sur les programmes
pour détecter les bugs avant l'exécution

Que veut dire vérifier ?

- Assurer qu'un circuit ou programme
  1. fait ce qu'il doit faire
  2. ne fait pas ce qu'il ne doit pas faire
- Toujours relativement à des critères d'observation
  – une spécification complète de la fonctionnalité
  – ou un jeu d'objectifs de couverture à atteindre
  – ou un jeu de propriétés à respecter
- En supposant des propriétés de l'environnement

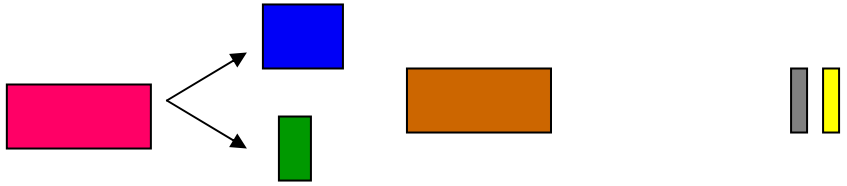
  exemple : pour aller du 1er au 3eme, l'ascenseur passe toujours par le 2eme

Agenda

1. Alerte aux pucerons !
2. Qu’est-ce qu’un bug ?
3. L’approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion

Approche classique : tâches et OS

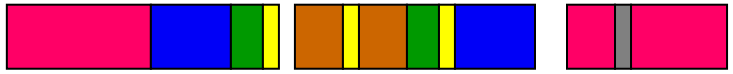
- Découpe en tâches périodiques / sporadiques



- Ordonnancement dynamique par événements / timers

  paramètres: masquages, interruptions, priorités, etc.

  souple, mais très difficile à maîtriser et valider (PathFinder)



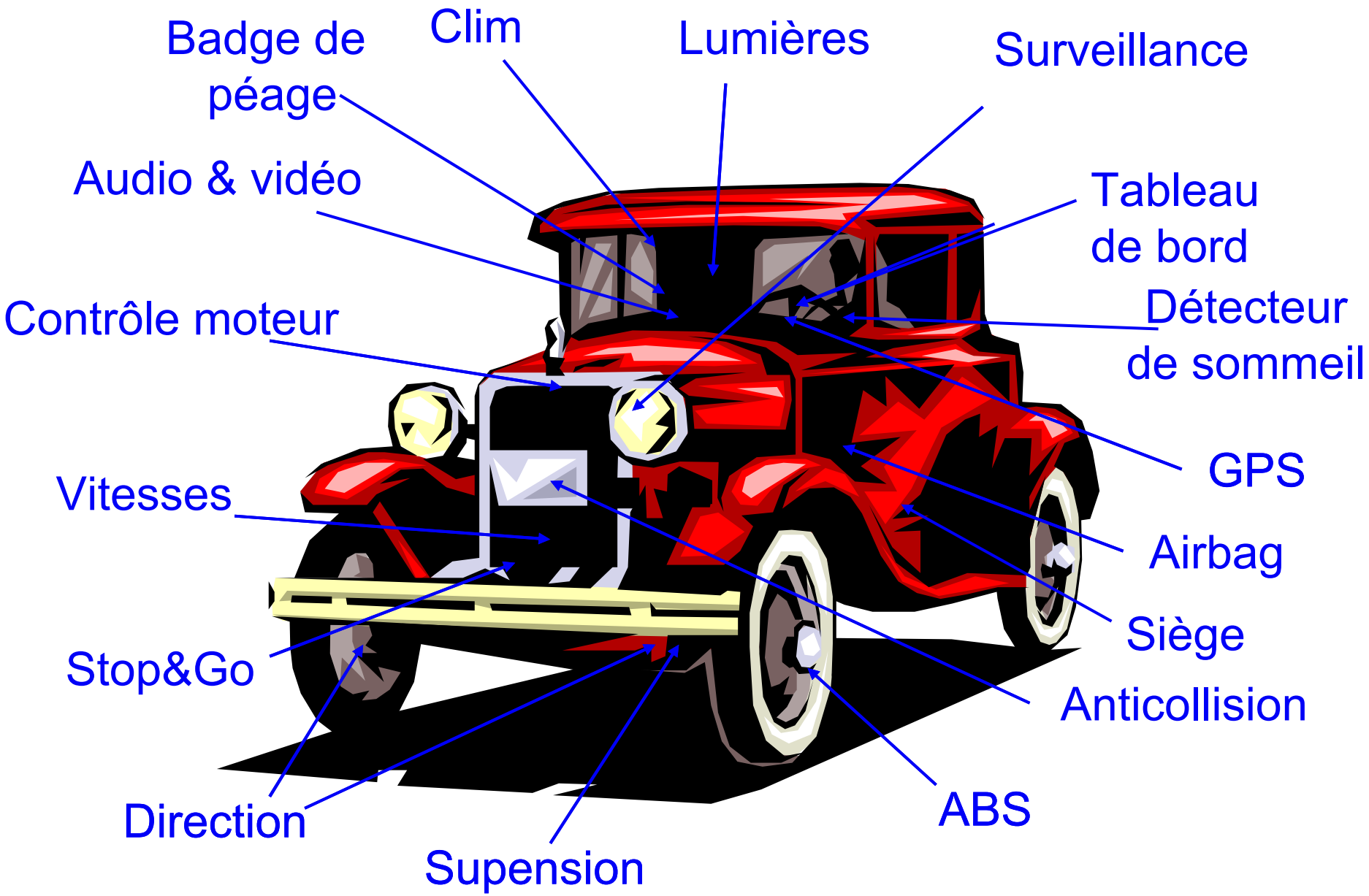
- Ordonnancement statique

  statique pour périodiques, trous pour sporadiques

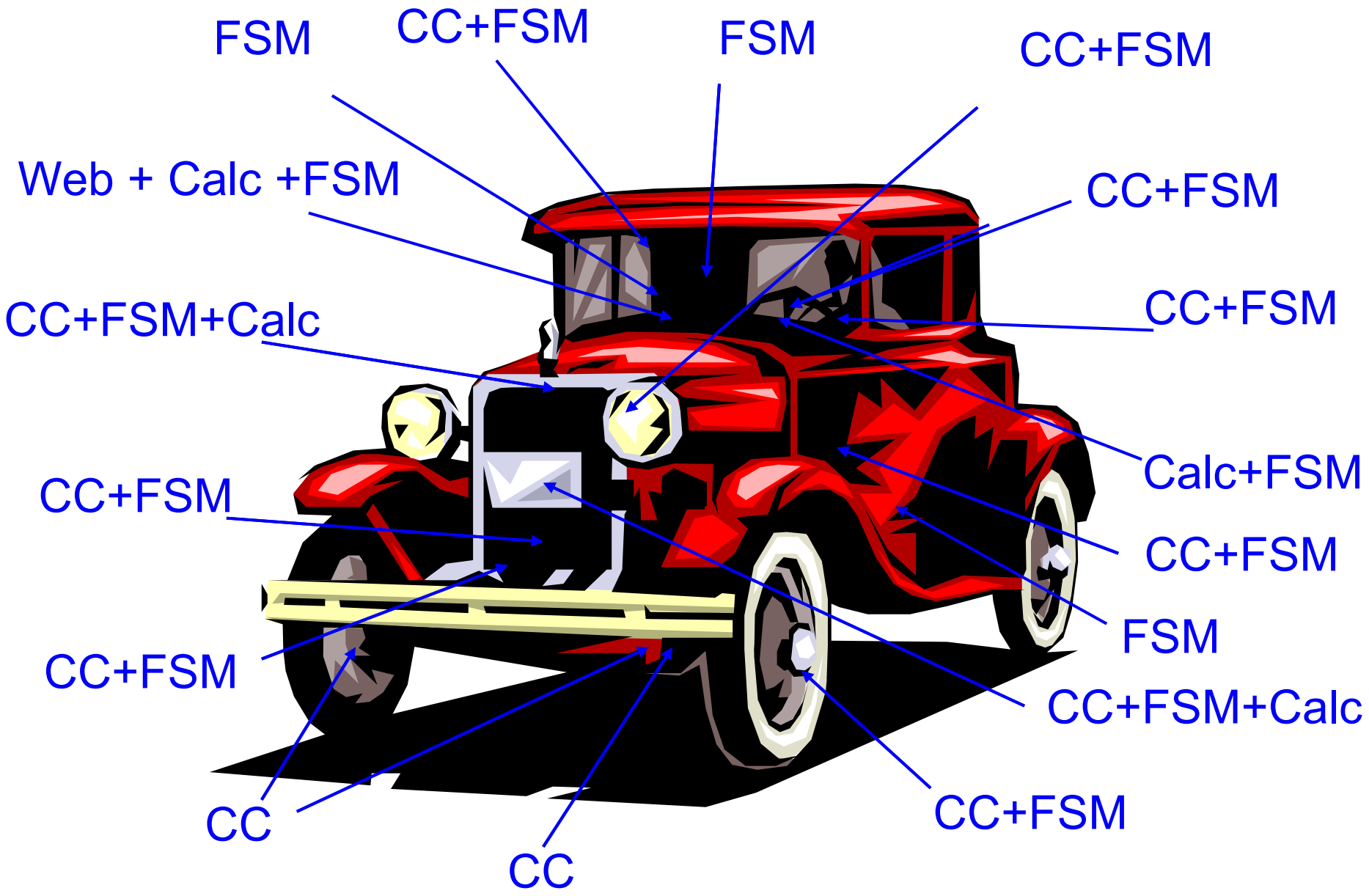
  maîtrisable mais difficile à calculer, instable aux changements

Anatomie des applications embarquées

- CC : contrôle continu, traitement du signal
  résolution d'équations différentielles, filtrage numérique
  spécification et simulation en Matlab / Scilab
- FSM : automates finis, graphes de transition
  contrôle discret, protocoles, sécurité, etc.
  automates plats ou hiérarchiques
- Calc : calcul intensif
  navigation, cryptage, etc.
  C + librairies
- Web : navigation, audio / video
  interaction graphique / audio / vidéo
  réseaux flots de données, Java



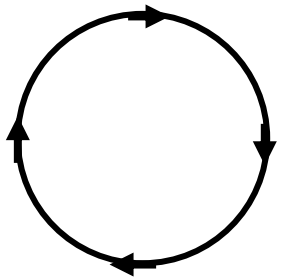
Coordination globale


FSM
CC+FSM
FSM
CC+FSM
Web + Calc +FSM
CC+FSM
CC+FSM+Calc
CC+FSM
Calc+FSM
CC+FSM
CC+FSM
FSM
CC+FSM
CC+FSM+Calc
CC
CC
CC+FSM

CC+FSM+Calc

Logiciel synchrone

Exécution cyclique



lire les entrées

calculer la réaction

produire les sorties

Synchrone = délai 0 = dans le même cycle

propagation parallèle du contrôle

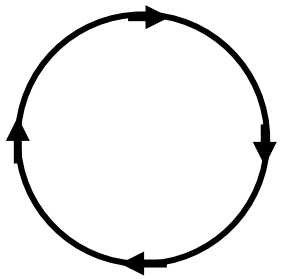
propagation parallèle des signaux

Mathématiques très élégantes

Pas d'interférences => déterminisme par construction

Taille de la pièce = Worst Case Execution Time (AbsInt)

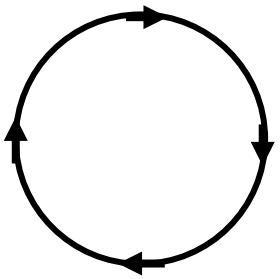
Parallélisme = fusion de cycles



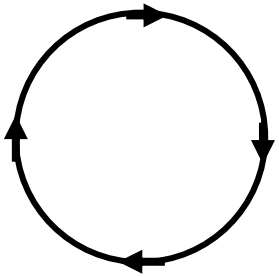
```
input X, Z;
output Y, T;
Y = X+1;
T = Z/2
```

Y

Z



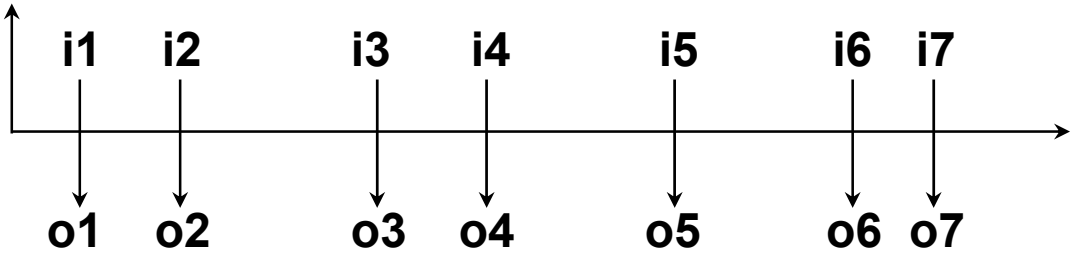
```
input Y;
output Z;
Z = Y*3;
```



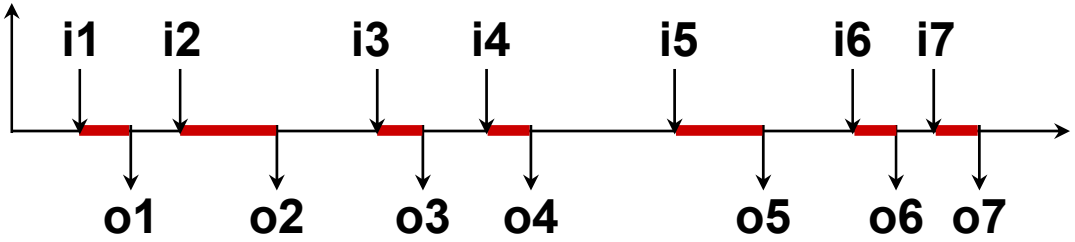
```
input X;
output T;
local Y, Z;
Y = X+1;
Z = Y*3;
T=Z/2;
```

Pas de synchronisation, pas de blocages,
pas de changements de contextes

Temps logique et temps physique



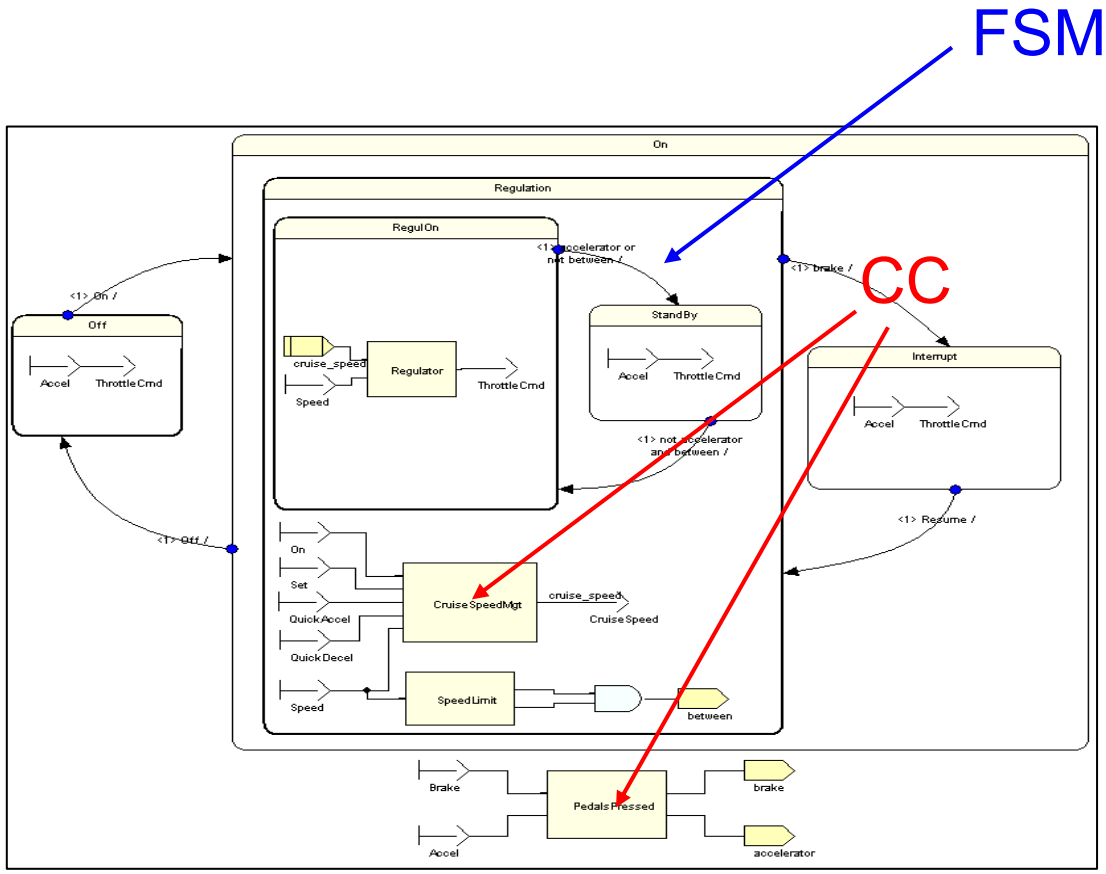
Temps logique

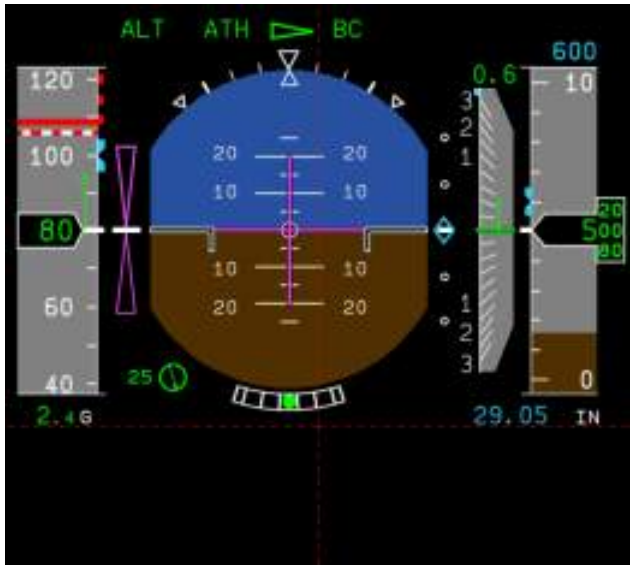


Temps physique

WCET OK = absence de recouvrement

SCADE 6 = CC + FSM + Calc + Visu





SCADE Display

Intégration de diagrammes de blocs, d'automates et de visualiseurs

Sémantique mathématique

- Définit le sens des programmes
  - Lustre: équations aux flots infinis
  - Esterel : règles d'inférences logique
- Apporte la sûreté de programmation
- Définit ce que compiler veut dire
- Définition de SCADE 6 : purement formelle
  le manuel en anglais n'est un appoint

Se méfier des contrefaçons !

Agenda

1. Alerte aux pucerons !
2. Qu'est-ce qu'un bug ?
3. L'approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion

Flots de conception et de vérification

- La notion de flot est centrale en développement industriel
- Flot : chemin complet des requirements à l'objet final
- Une méthode n'est pertinente que si elle est intégrée à un flot officiel (non-R&D)
- La vérification n'est pas une activitée séparée. Elle intervient partout et doit être elle-même vérifiée

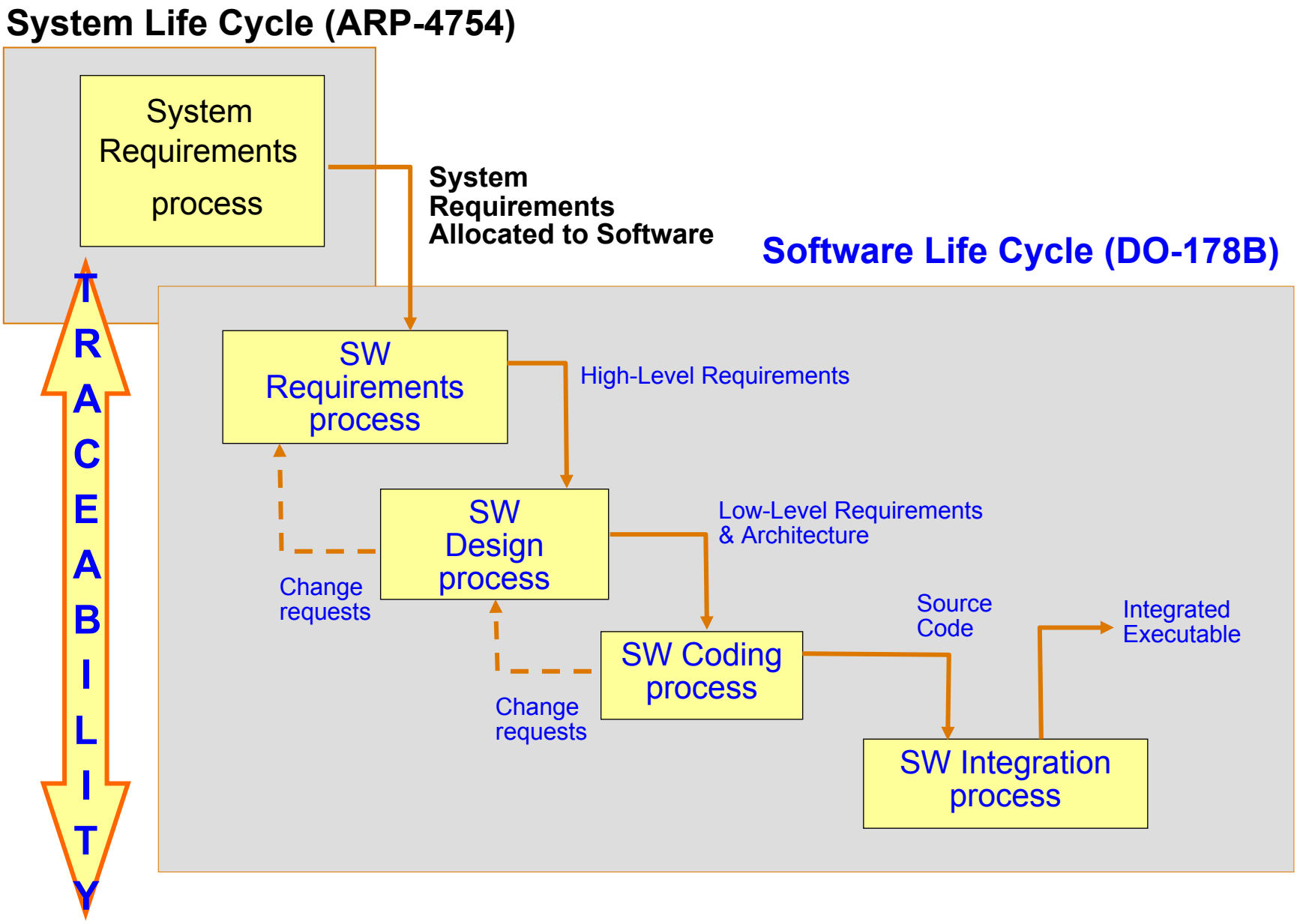
Les flots ne peuvent pas évoluer vite

Le flot certifié DO-178B (avionique)

- Certification du process de développement par des autorités indépendantes (FAA, CEAT, JAA, etc.), obigatoire depuis 1992
- But : détecter and reporter les erreurs introduites au cours du développement du logiciel
- Définit des objectifs de vérification, ne spécifie pas de technique particulière
- La vérification comporte des test, des revues, et une analyse profonde du process reposant sur la traçabilité des actions
- La vérification de la vérification est obligatoire

Les outils formelles seront intégrés dans la DO-178C

DO-178B Development Process



Level	Effect of anomalous behavior
A	Catastrophic failure condition (crash)
B	Hazardous/severe failure condition for the aircraft (several persons could be injured)
C	Major failure condition for the aircraft (Flight Management failure, manual management required)
D	Minor failure condition for the aircraft (Pilot/Ground communication lost)
E	No effect on aircraft operation or pilot workaround (entertainment features down)

Vérification de la Vérification

- Monter que les testent couvrent les High-level Requirements (HLR)
- Monter que les testent couvrent les Low-Level Requirements (LLR)
- Montrer que le code source a été bien secoué :
  - Level C: 100% Statement Coverage
  - Level B: 100% Decision Coverage
  - Level A: 100% Modified Condition / Decision Coverage

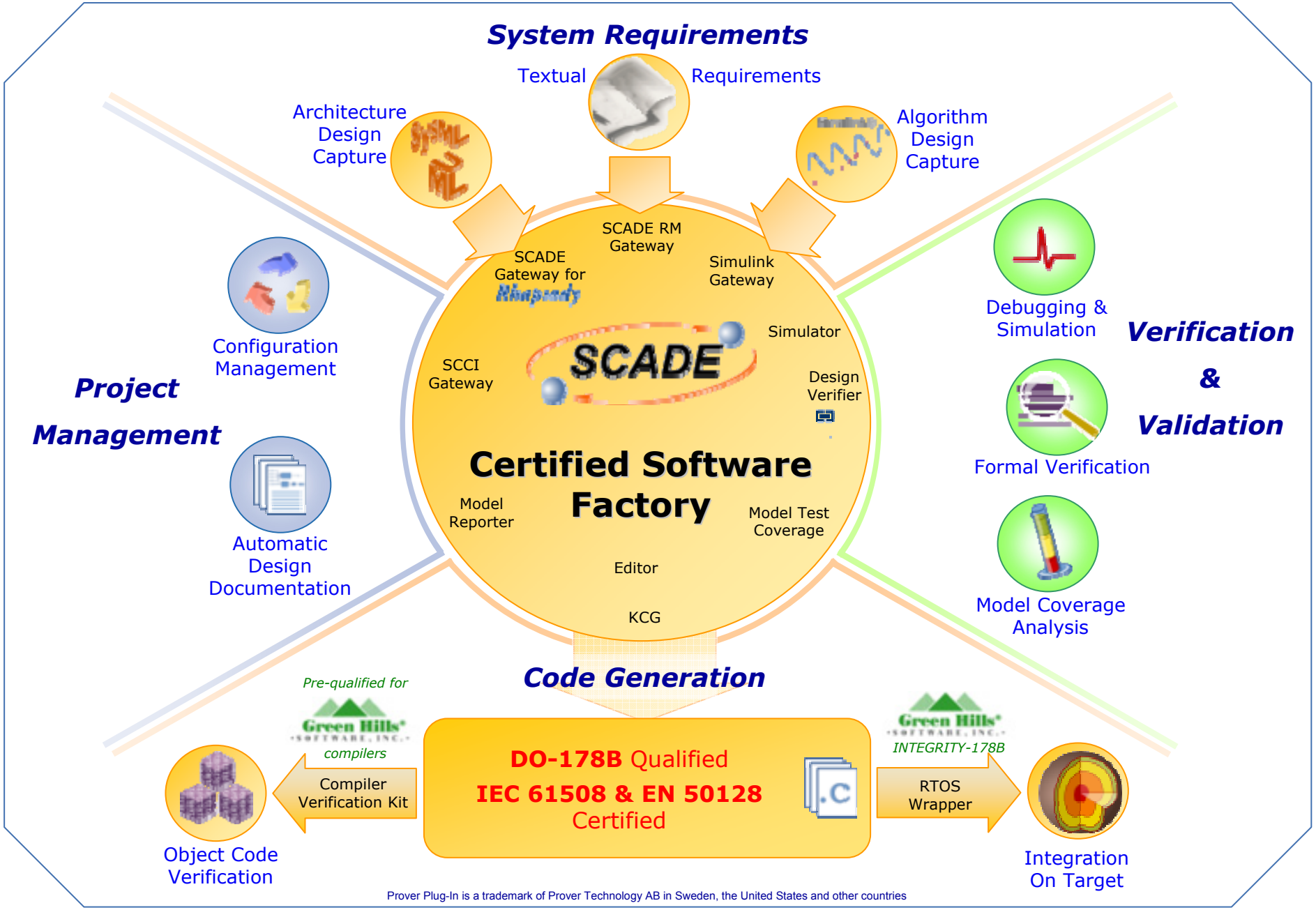
Mais comprend-on ce que ça vérifie vraiment ?

Autres standards

- DO-254: avionics hardware development
- IEC 61508: function safety of systems made with Electrical, Electronic, Programmable electronic components
- EN 50128: Adaptation of IEC 61508 to Railways
- MIL-STD-498: Military standard for SW development
- DEF-STD-055/056: Safety management for Defense Systems
- Chinese Standards

Agenda

1. Alerte aux pucerons !
2. Qu'est-ce qu'un bug ?
3. L'approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion


System Requirements
Textual Requirements
Architecture Design Capture
Algorithm Design Capture
SCADE RM Gateway
SCADE Gateway for Rhapsody
Simulink Gateway
Simulator
SCCI Gateway
SCADE
Design Verifier
Certified Software Factory
Model Reporter
Model Test Coverage
Editor
KCG
Configuration Management
Project Management
Automatic Design Documentation
Debugging & Simulation
Verification & Validation
Formal Verification
Model Coverage Analysis
Code Generation
Pre-qualified for
Green Hills SOFTWARE, INC.
compilers
Compiler Verification Kit
Object Code Verification
DO-178B Qualified
IEC 61508 & EN 50128
Certified
.C
Green Hills SOFTWARE, INC.
INTEGRITY-178B
RTOS Wrapper
Integration On Target
Prover Plug-In is a trademark of Prover Technology AB in Sweden, the United States and other countries

Code généré par KCG

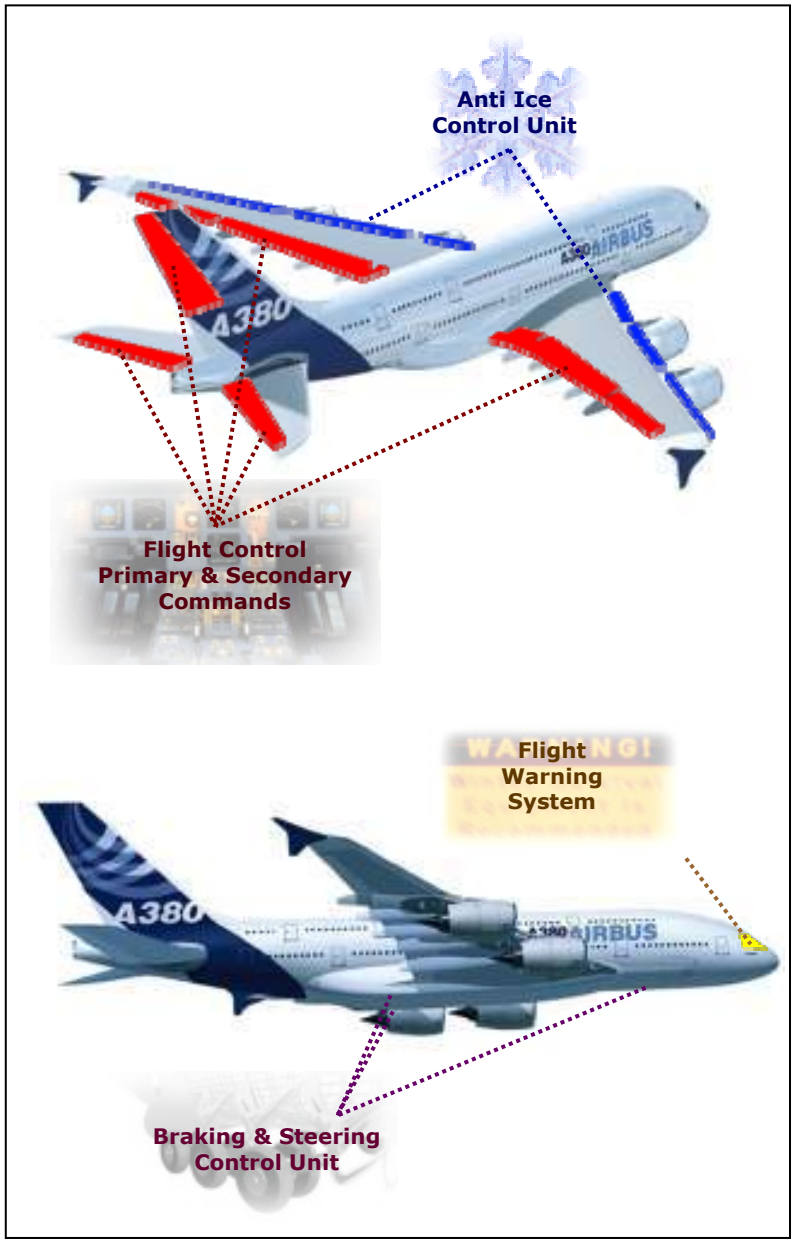
- Petit sous-ensemble portable de C
  indépendant du processeur, du compilateur et de l'OS
- Structuré, lisible, traçable
- Avec allocation mémoire statique
- Sans arithmétique de pointeurs
- Sans récursion
- A temps d'exécution borné et prévisible
- Optimisé en temps et mémoire

Certification de KCG

⇒ disparition du test unitaire du code généré

⇒ certification et recertification des applis bien plus rapide

SCADE dans l'Airbus A380



- Contrôle de vol
- FADEC (contrôle moteur)
- Freinage et direction
- Gestion électrique
- Anti-givrage
- Système d'alarmes
- Cockpit:
  - PFD : Primary Flight Display
  - ND : Navigation Display
  - EWD : Engine Warning Display
  - SD : System Display
- ...

Architecture

components
dimensioning
communication

Word, Excel, Visio
System C

Micro-Architecture

concurrency
pipeline
resource sharing

Word, Visio, C

RTL design

gates, clocks
registers, RAMs
critical path

VHDL, Verilog

circuits

cells, clock trees
area, speed

netlists

DFT (test)

testability
scan insertion

netlists

Place&Route

physical & electrical
constraints

P&R netlists

$ 1,000,000 Masks

printing

pseudo-rectangles

Chips

fabrication

silicon dies

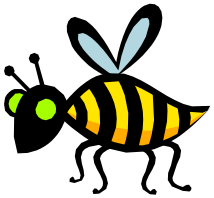
Architecture

$$ 

Micro-Architecture

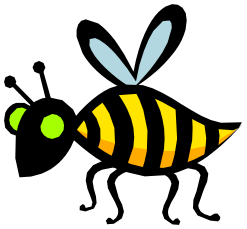
$$ 

RTL design

$$ 

circuits

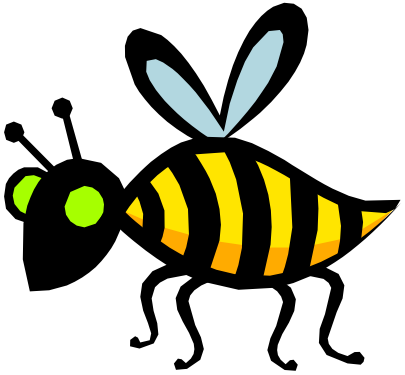
DFT (test)

$$ 

Place&Route

$ 1,000,000

Masks

$$ 

Chips

Architecture

functionality OK?
throughput OK?
marketing OK?

Experience
Reviews

↓

Micro-Architecture

breakdown OK?
performance OK?

C-based modeling

↓

RTL design

functionality OK?
area/speed OK?
power OK?

Random-directed testing,
Formal verification

↓

circuits

equivalent to RTL?

formal equivalence checking

↓

DFT (test)

test coverage ~100% ?

ATPG

↓

Place&Route

connections?
electrical constraints?
timing?

Design Rules Checking (DRC)

↓

$ 1,000,000 Masks

No fab fault?

Scan test run

↓

Chips

Esterel

Architecture

Micro-Architecture

RTL design

circuits

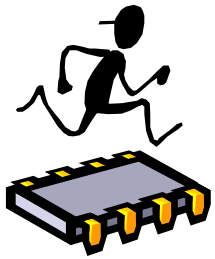
DFT (test)

Place&Route

Masks

Chips

$ 1,000,000



Agenda

1. Alerte aux pucerons !
2. Qu'est-ce qu'un bug ?
3. L'approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion

La vérification de modèles (Model-Checking)

- Un programme définit un système de transitions

$S = \{s_i \mid i \in N\}$

s_0 : état initial

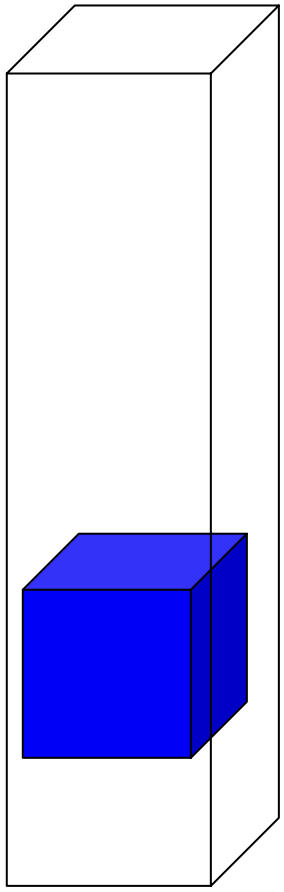
$s_i \rightarrow s_j$: transition

- Propriété P = prédicat sur les états potentiels

- Questions sur les états accessibles depuis l'initial
  - P peut-elle être vraie (ou fausse)?
  - P est-elle inévitablement vraie sur toute trajectoire infinie?

Clarke, Emerson, Sifakis : Prix Turing 2007

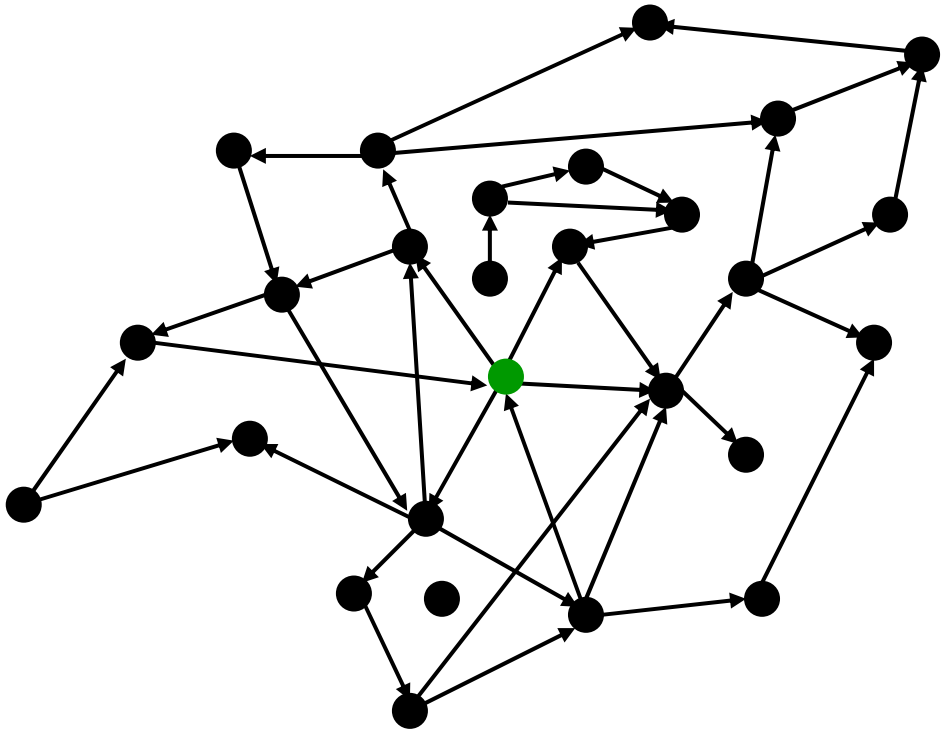
Exemple : l'ascenseur



- Sûreté : l'ascenseur ne voyage jamais la porte ouverte

- Vivacité : l'ascenseur atteint tous les étages et y ouvre ses portes (plus dur)

Graphe des états / transitions potentiels (extrait du programme)

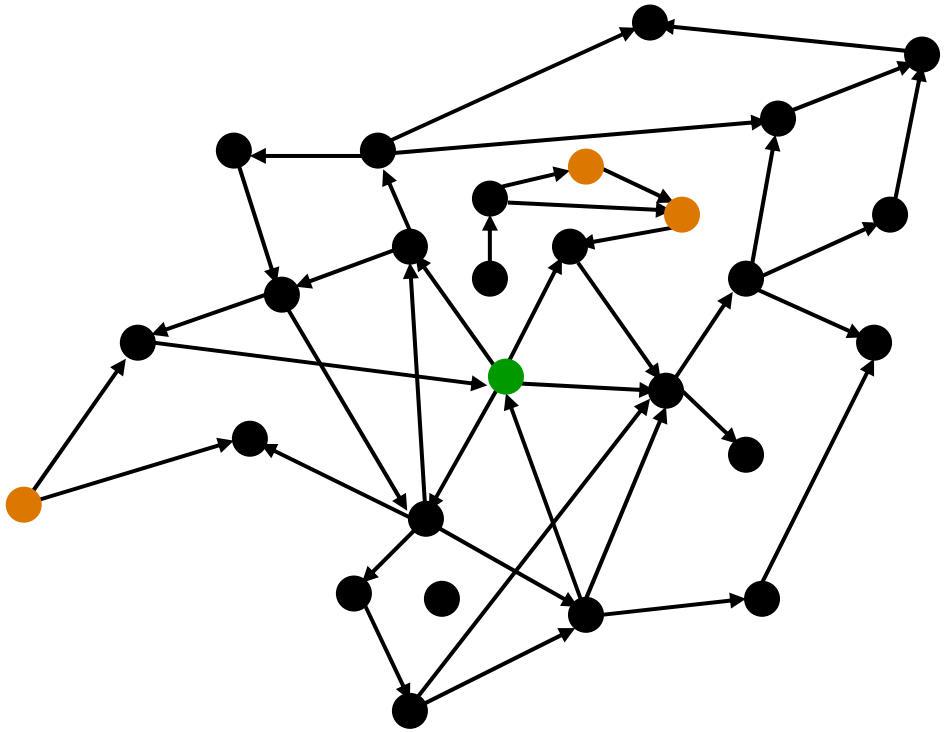


état initial

état potentiel

transition potentielle

Propriété P = ensemble d'états



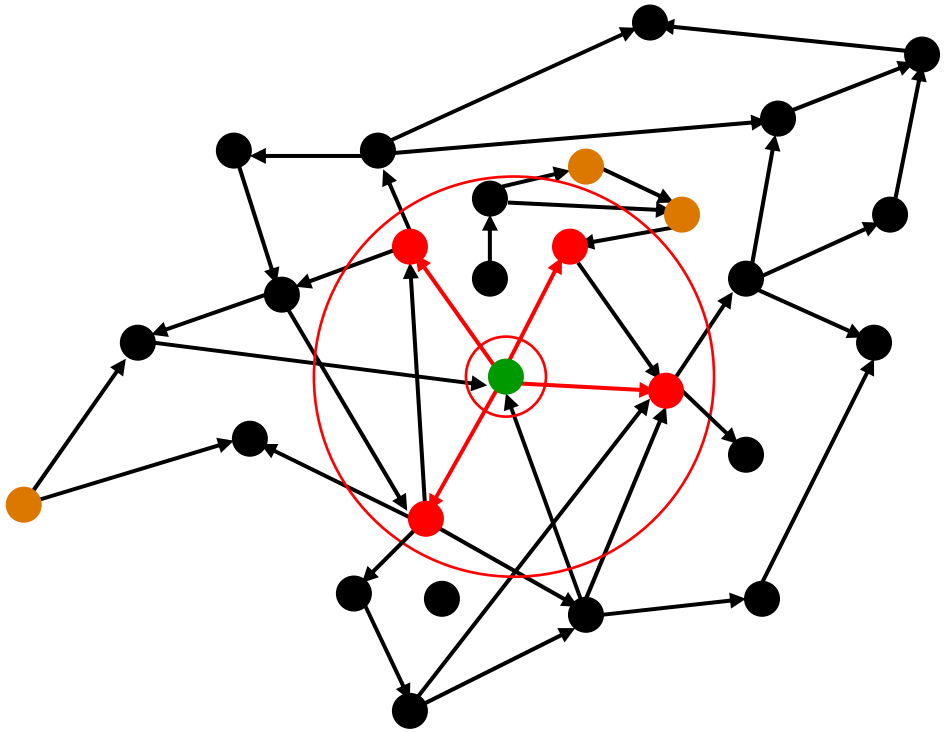
état initial

état potentiel

transition potentielle

P fausse

Analyse en avant
Etats atteignables en 1 transition

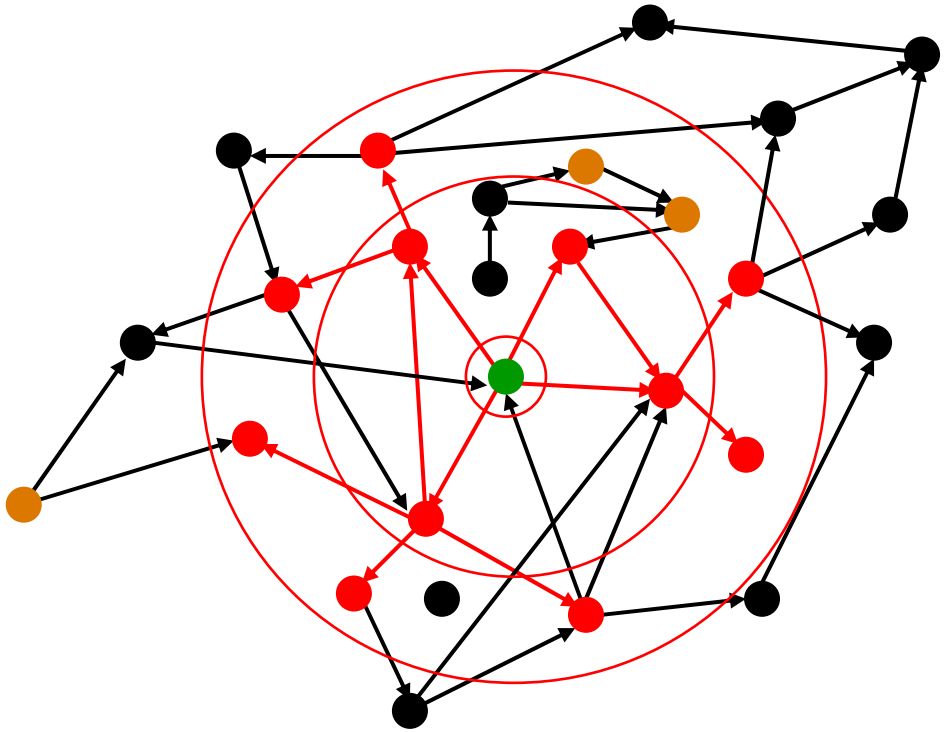


état initial

état atteignable

transition atteignable

Etats atteignables en 2 transitions

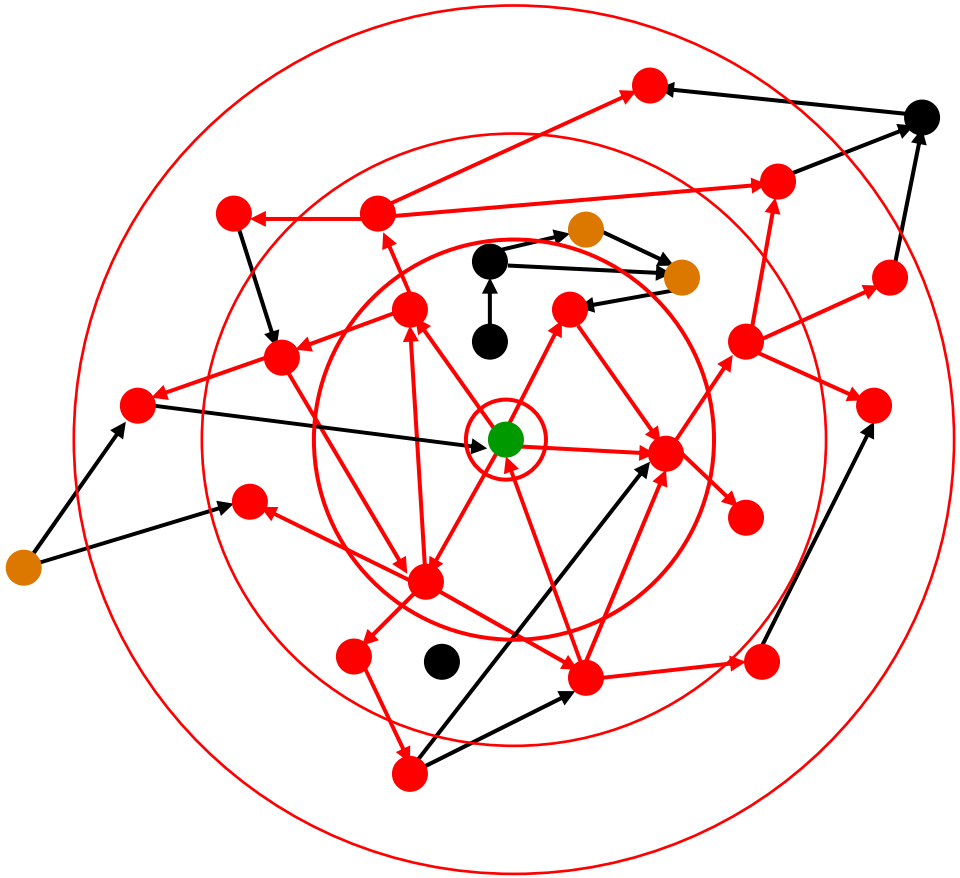


état initial

état atteignable

transition atteignable

Etats atteignables en 3 transitions

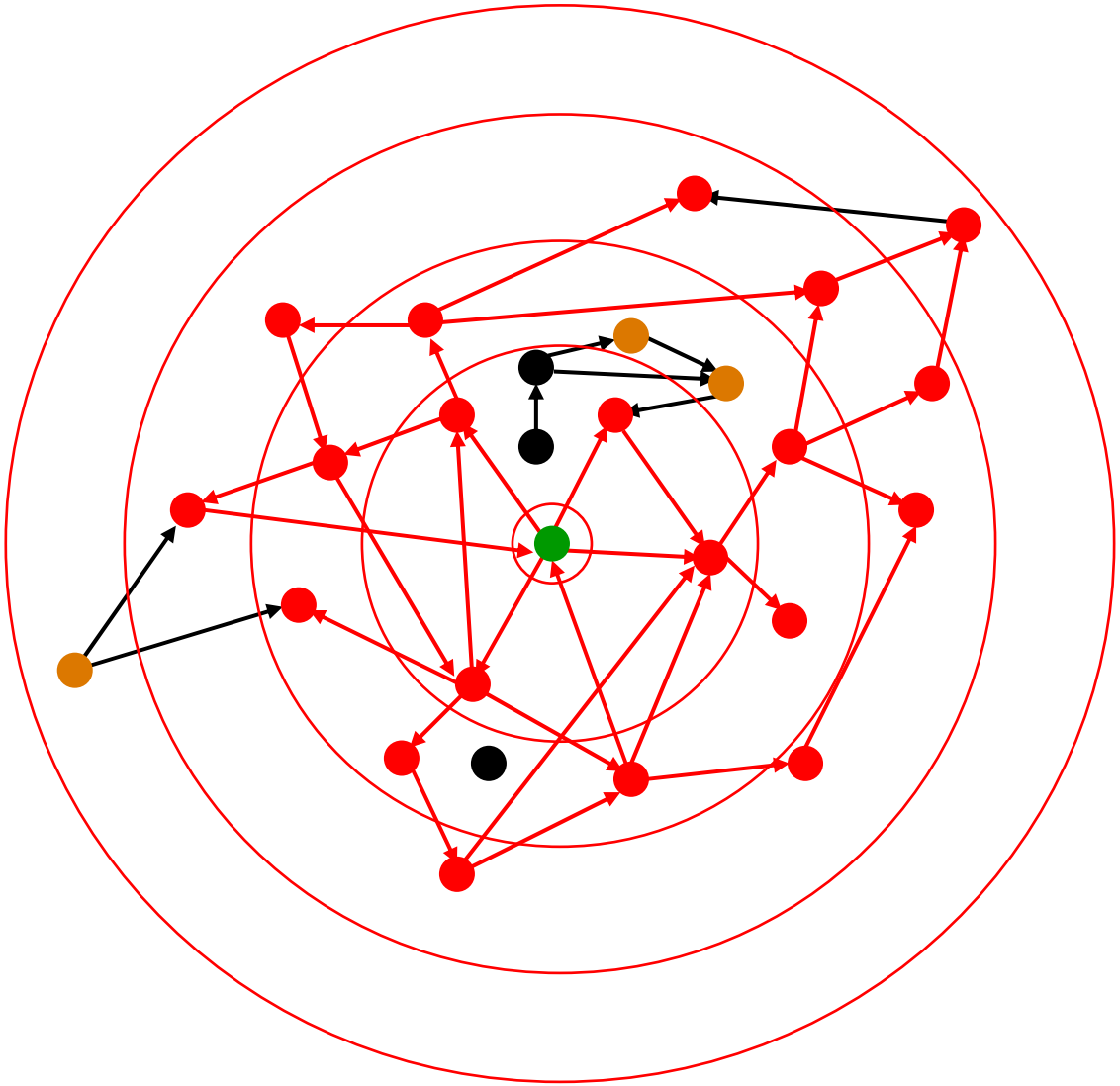


état initial

état atteignable

transition atteignable

Etats atteignables en 4 transitions

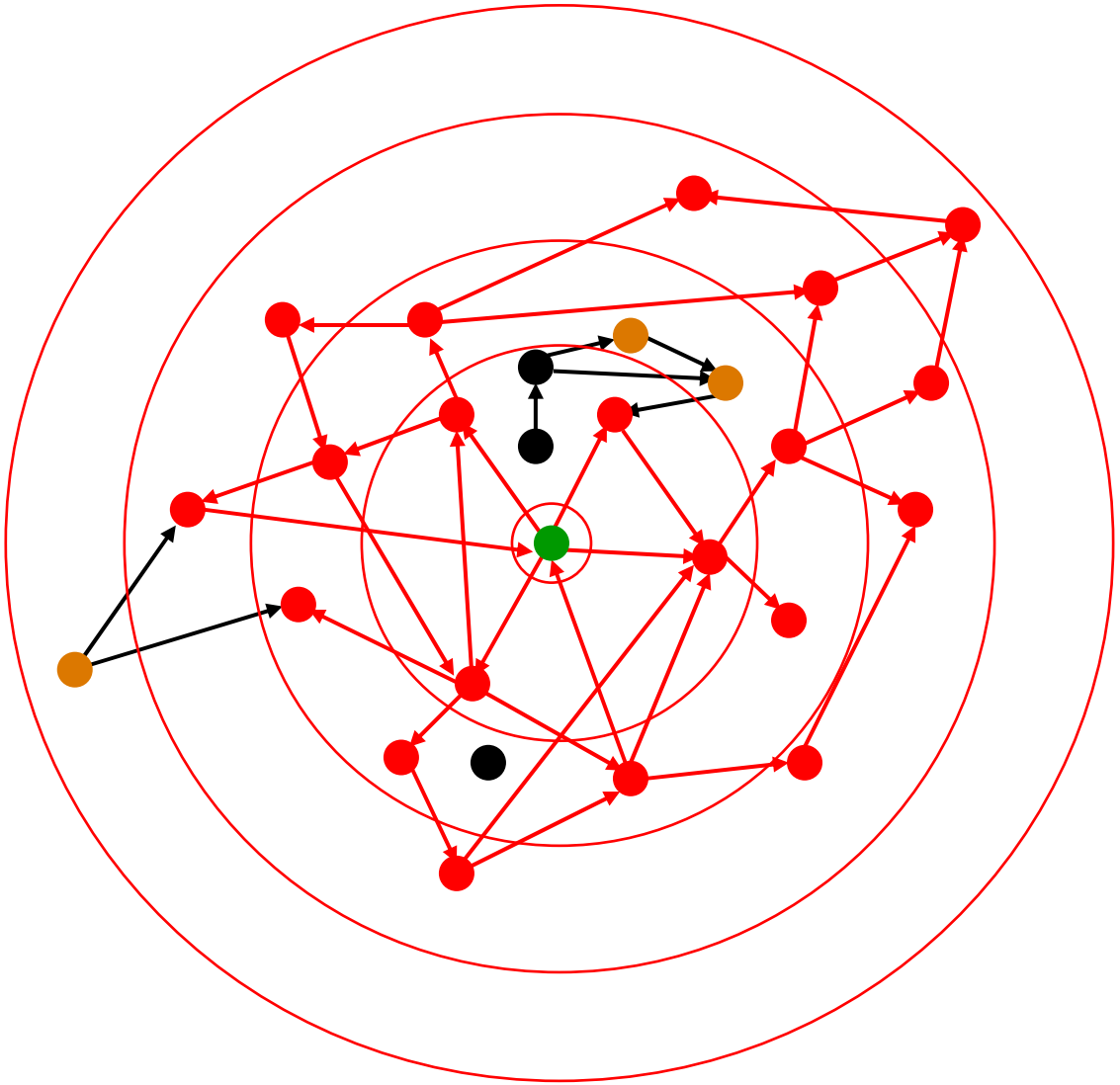


état initial

état atteignable

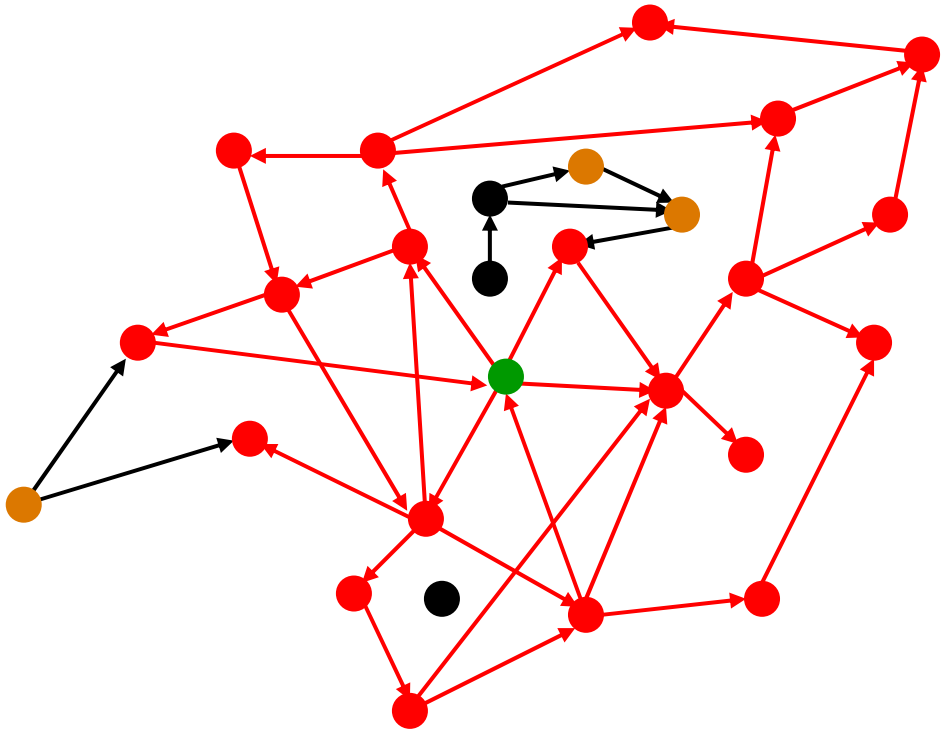
transition atteignable

Denier marquage,
Pas de nouveaux états => point fixe

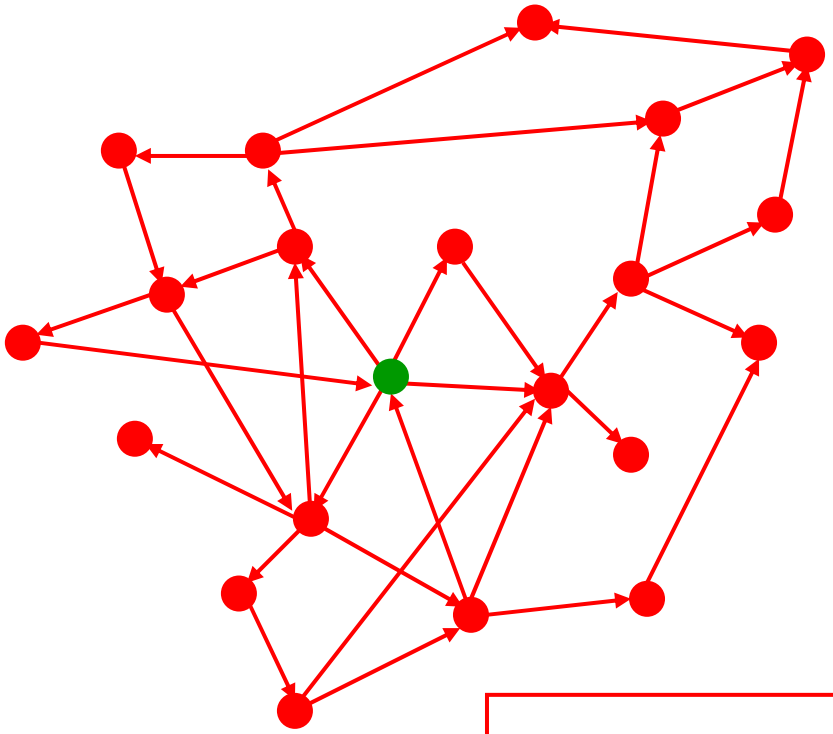


diamètre = 4

Ignorer les états non atteignables



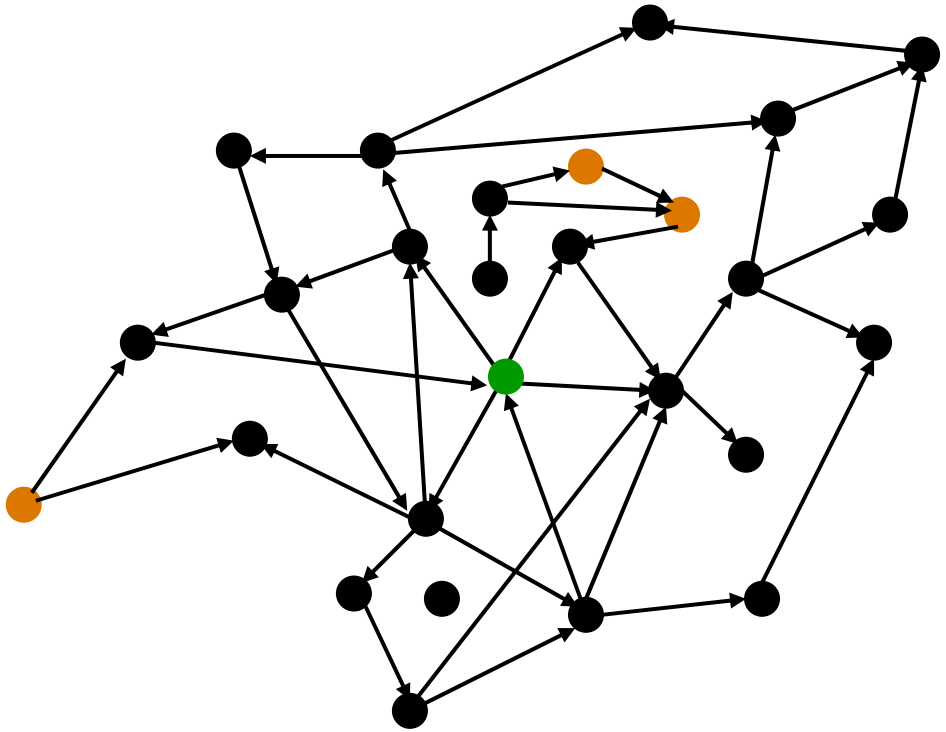
Ignorer les états non atteignables



Pas d'états où P est fausse

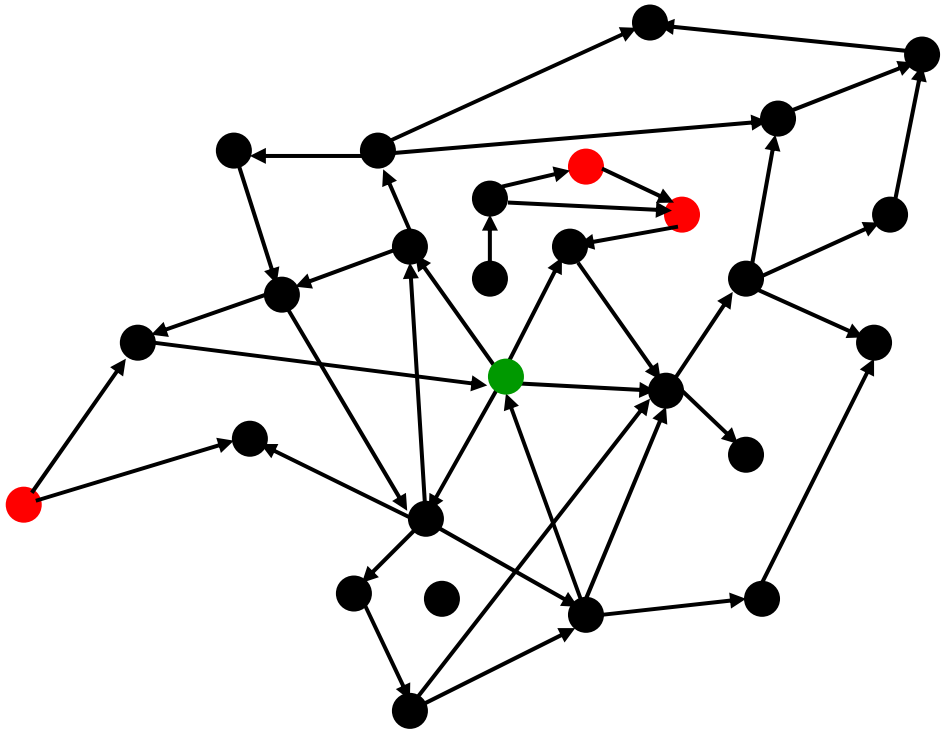
=> P est vraie pour le système

Analyse en arrière



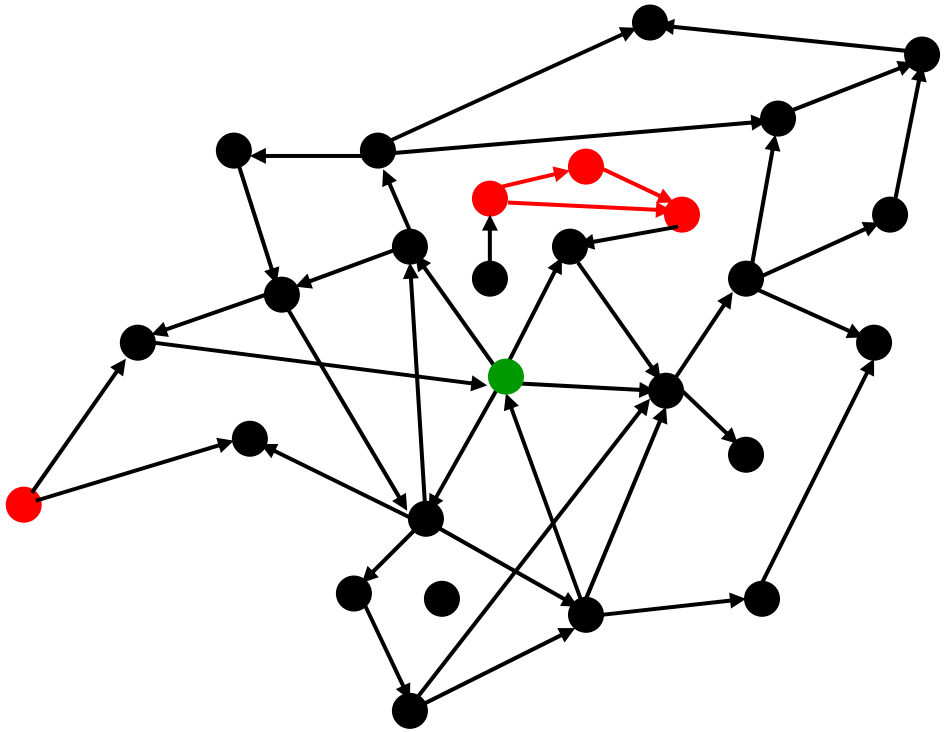
- état initial
- état potentiel
- transition potentielle
- P fausse

Marquer les états où P est fausse



- état initial
- état potentiel
- transition potentielle
- P fausse

Marquer itérativement leurs prédécesseurs

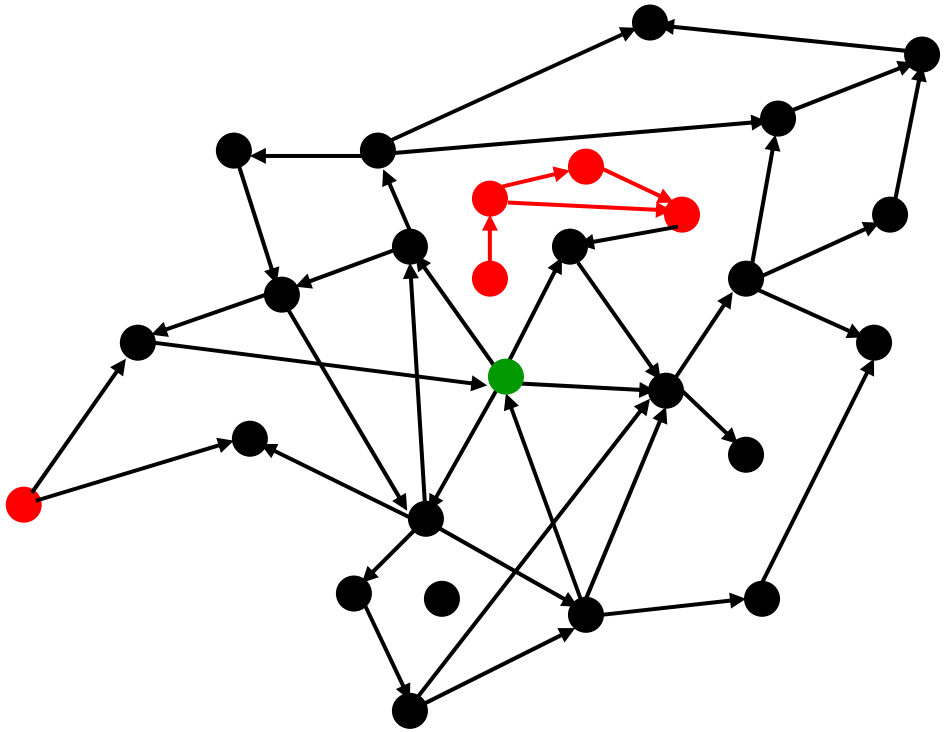


état initial

état potentiel

transition potentielle

Marquer itérativement leurs prédécesseurs

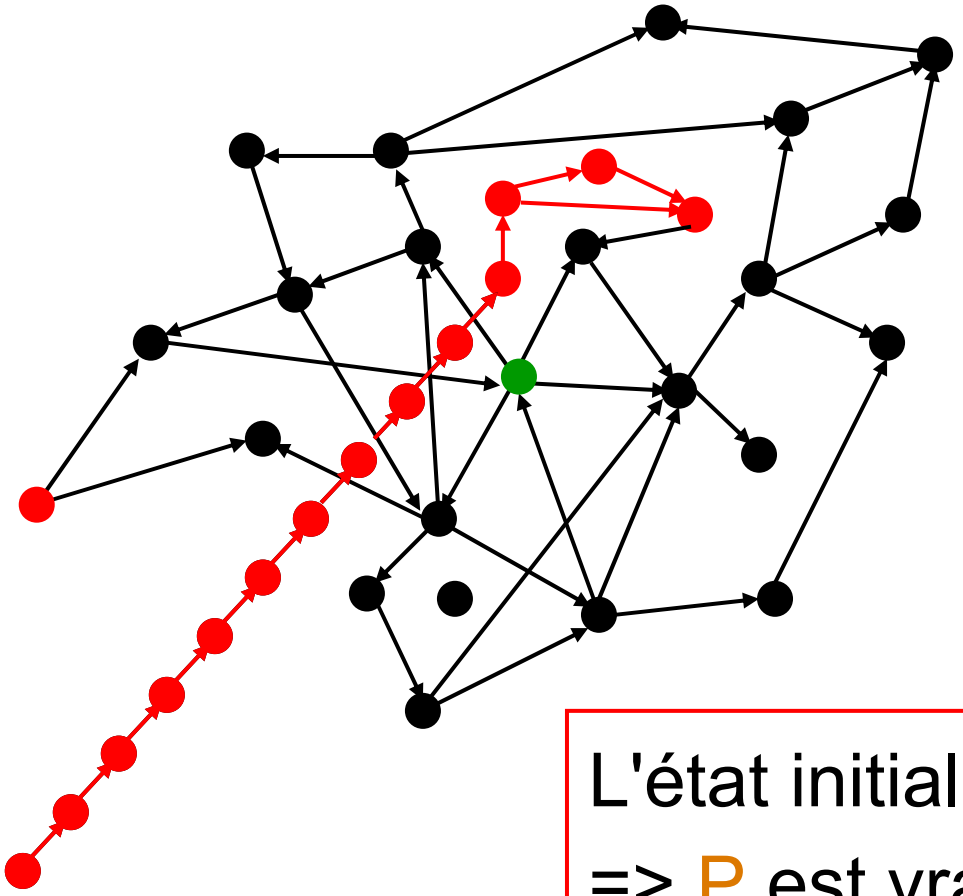


état initial

état potentiel

transition potentielle

Fin du marquage



état initial

état potentiel

transition potentielle

L'état initial n'est pas marqué

=> P est vraie pour le système

Eviter l'explosion de la taille

- Exploration explicite très rapide, randomisation

  SPIN : protocoles, caches, etc.

- Exploration symbolique par calcul sur formules

  fonctions caractéristiques : BDDs

  dépliement symbolique et récurrence : SAT

  couplage de moteurs Booléens, symboliques et numériques (SMT)

- Interprétation abstraite

  travailler sur une abstraction du système

Nombreux systèmes académiques et industriels

SMV, Prover, Magellan, Formal Check, SLAM, Yices,...

Un domaine en pleine évolution

- Passer des cas précis aux cas génériques

  des ascenseurs à 4 étages aux ascenseurs à n étages

- Coupler vérification assistée + moteurs logiques

  + moteurs numériques

  se ramener à des sous problèmes finis

  ou traitables automatiquement

- Profiter des machines multi-coeurs et réseaux

  lancer 10 algos différents et prendre le 1er qui gagne

  mettre un algorithme sur multi-coeurs ?

Question industrielle majeure :
comment prévoir le temps de vérification ?

Agenda

1. Alerte aux pucerons !
2. Qu'est-ce qu'un bug ?
3. L'approche synchrone
4. Flots de développement et certification
5. Les flots formels SCADE et Esterel
6. Vérification formelle de modèles finis
7. Assistants de preuve
8. Conclusion

Assistants de preuves

- Logiques classique : prédicats, Scott, Hoare

  B (Abrial) : métro et RER, PVS : NASA

  HOL (Cambridge), HOL Light (Intel) : calculs flottants Intel

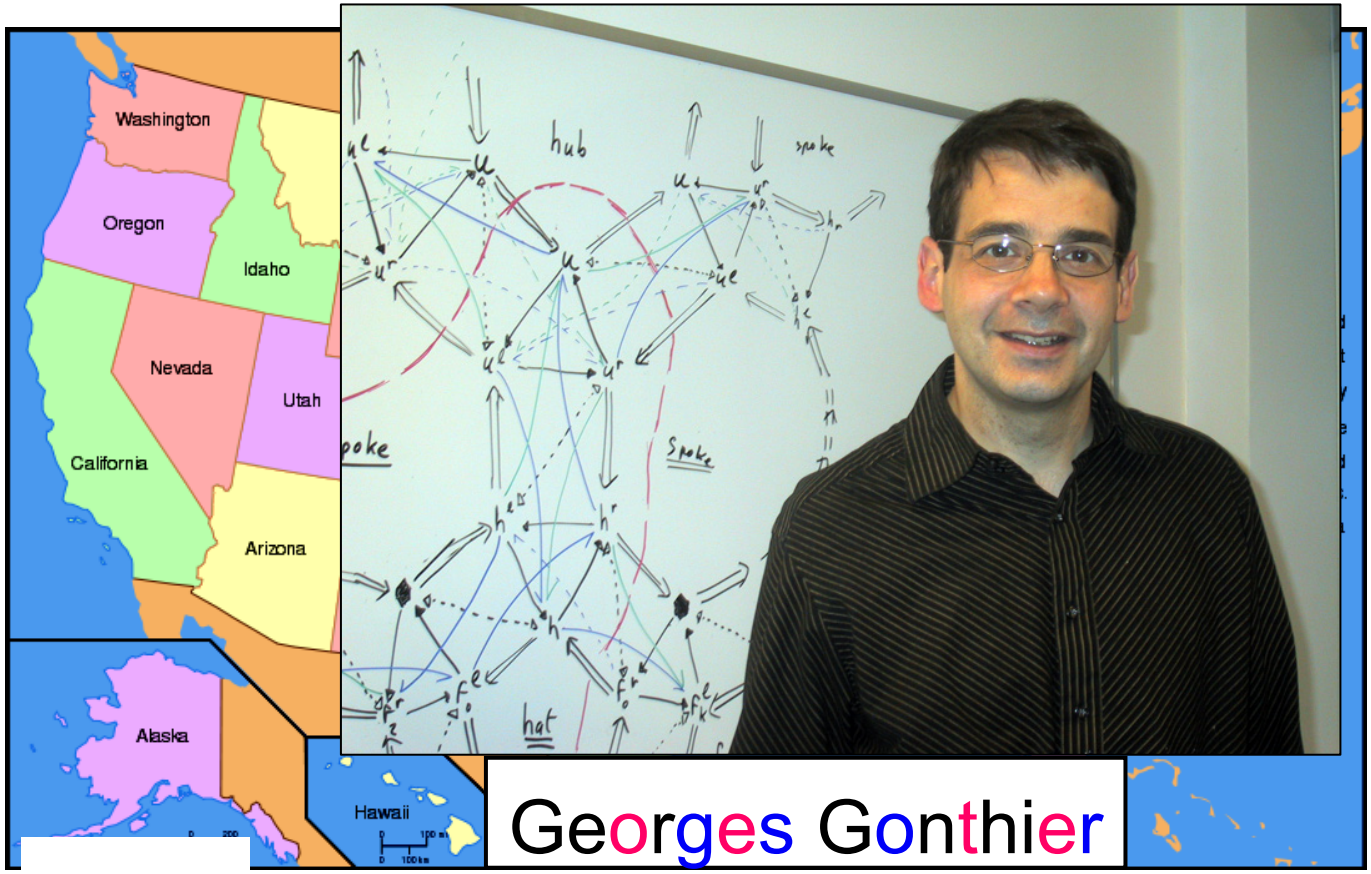
  Isabelle (Cambridge) : OS (NICTA), circuits

  ACL-2 (MIT) : calculs flottants AMD

- Logiques constructives (CoQ)
  - Un algorithme est une preuve de consistance de la spécification.
  - On développe en même temps l'algorithme et la preuve, puis on extrait automatiquement le programme.

  compilateur C vérifié (Leroy et. al), protocoles bancaires, théorème des 4 couleurs (Gonthier)

Le théorème des 4 couleurs en CoQ



- 1852 Guthrie
- 1976 Appel – Haken
- 2005 Gonthier (en CoQ)

Preuve omise, évidente mais longue

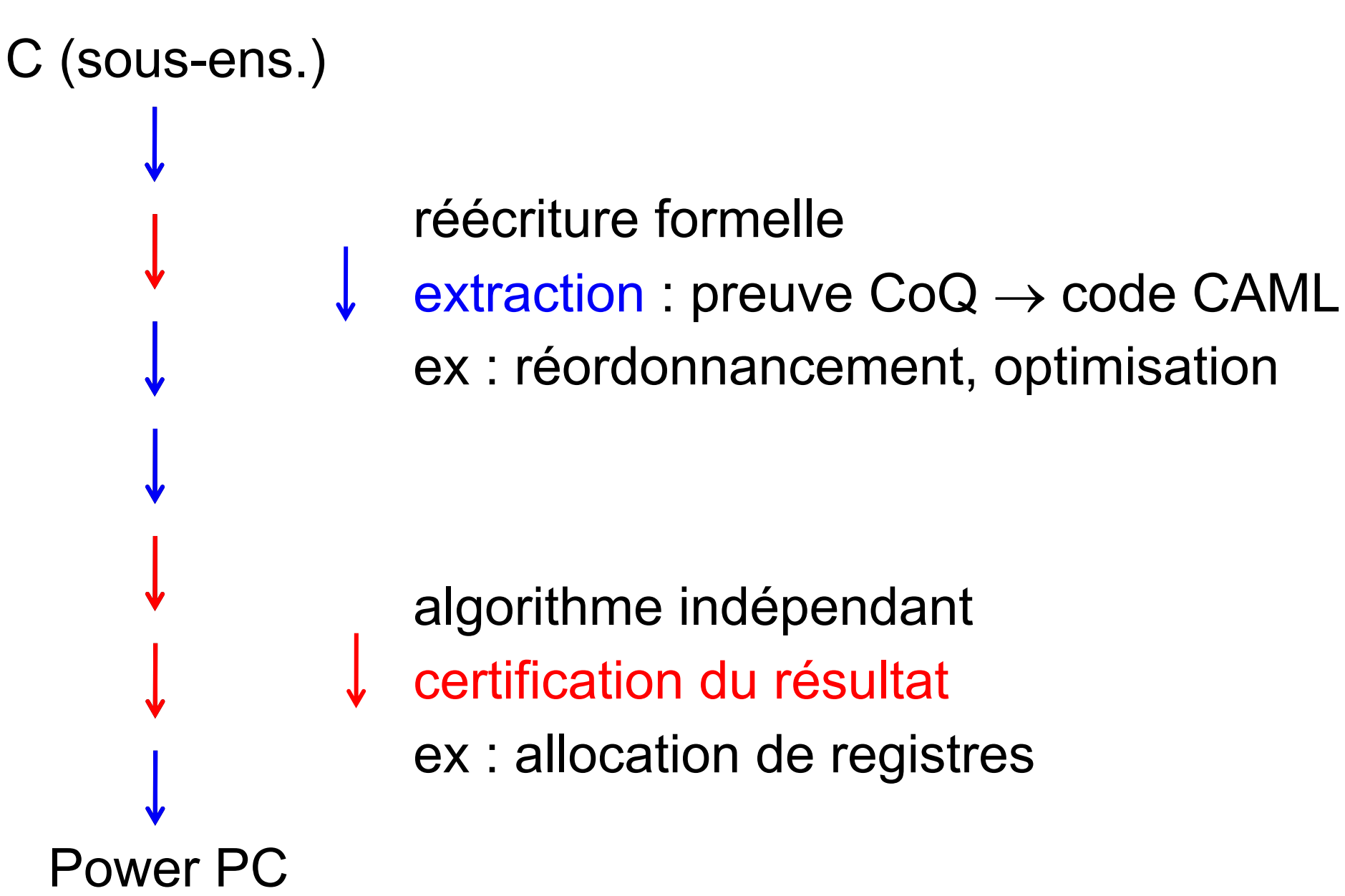
=> génie logicien

Le compilateur vérifié CompCert (X. Leroy)

• Compilateur certifiant : étant donnés un programme et une preuve qu'il est correct, produire du code machine équivalent et une preuve que ce code est correct.

• Compilateur vérifié : prouver que le compilateur préserve la correction du programme, i.e., que toute propriété vraie du programme source est également vraie du code machine produit.

≠ compilateur certifié !


C (sous-ens.)
réécriture formelle
extraction : preuve CoQ → code CAML
ex : réordonnancement, optimisation
algorithme indépendant
certification du résultat
ex : allocation de registres
Power PC

Ce dont nous n'avons pas parlé

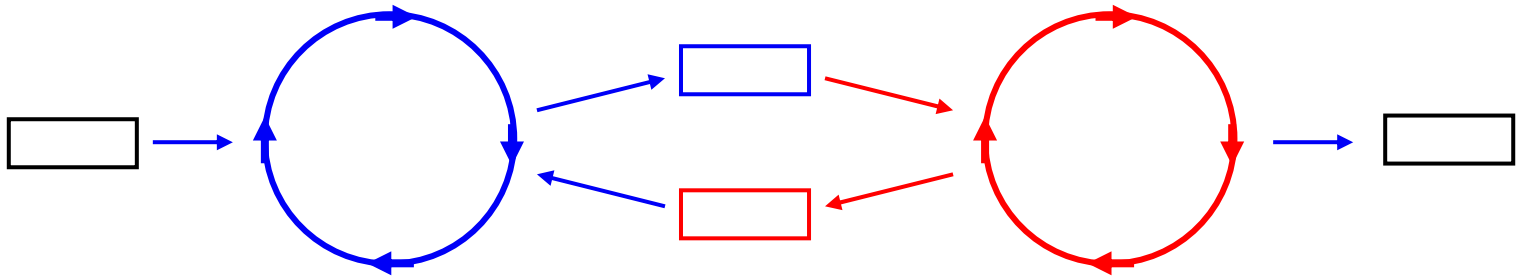
- Les calculs et logiques de l'informatique
  - lambda-calcul, domaines de Scott
  - logiques constructives, théorie des constructions
  - automates, logiques temporelles
- L'interprétation abstraite (un standard!)
  - Astrée (flottant), AbsInt (WCET), SLAM (Microsoft), etc.
- La génération automatique de tests
  - randomisés, dirigés par un prouveur, etc.
- Les preuves de sécurité
  - protocoles bancaires, sécurité anti-intrusions, etc.
  - proof-carrying code

Conclusion

- La vérification est une activité clef
  pourquoi programmer plus vite si vérifier est plus long ?
- La programmation doit s'adapter à la vérification
  et non l'inverse (cf. ponts et résistance des matériaux)
- Le test reste indispensable et dominant
- Les méthodes formelles commencent à se répandre
  elles trouvent des bugs introuvables par tests
  et permettent des preuves infaisables à la main

La certification du futur devra être plus formelle

Extension du modèle: des pièces aux châteaux



Echantillonage mutuel (Caspi, et. al.)

- Fonctionne à causes de raisons d'automatique, pas d'informatique (Théorème de Nyquist distribué)
- Semblable aux circuits multi-horloges, mais plus simple (pas de métastabilité des mémoires)
- Alternative : Time Triggered Networks (TTP, FlexRay)