Les limites théoriques de l'informatique

Les problèmes indécidables

Samuel Fiorini - Gilles Geeraerts - Jean-François Raskin
Université Libre de Bruxelles

Académie Royale des Sciences
Bruxelles
3/3/2010 et 21/4/2010

Organisation du cours

- Deux leçons
  - Introduction à la notion d'indécidabilité (Jean-François Raskin)
  - Conséquences de l'indécidabilité sur la recherche :
    - en sciences informatiques (Gilles Geeraerts)
    - et en sciences mathématiques (Samuel Fiorini).

Leçon 2 - première partie

- Conséquences de l'indécidabilité sur la recherche en informatique
  - Rappel des concepts et résultats de la première leçon
  - Indécidabilité de l'arrêt
  - Motivation de la Vérification Assistée par Ordinateur (VAO)
  - Techniques de vérification
    - Le *model checking* à base d'automates

Rappels

Rappels de la leçon 1

Rappels de la leçon 1

- Question centrale : peut-on établir des limites théoriques aux problèmes qui sont solvables à l'aide des ordinateurs ?
  - Pour ce faire, on doit formaliser les notions de problème et d'algorithme (= solution au problème)

Rappels de la leçon 1

- Question centrale : peut-on établir des limites théoriques aux problèmes qui sont solvables à l'aide des ordinateurs ?
  - Pour ce faire, on doit formaliser les notions de problème et d'algorithme (= solution au problème)
- Pour formaliser la notion de problème (de décision), on utilise la notion de langage
  - Un langage est un ensemble de mots
  - Chaque mot encode une instance positive du problème
  - Exemple : Le problème « décider si un nombre est premier » peut être formalisé par l'ensemble (infini) de tous les encodages binaires de nombres premiers : {10, 11, 101, 111, ...}

Rappels de la leçon 1

Rappels de la leçon 1

- Résoudre un problème revient donc à trouver une procédure automatique, une «machine», qui est capable de dire si un mot appartient ou non à un ensemble donné

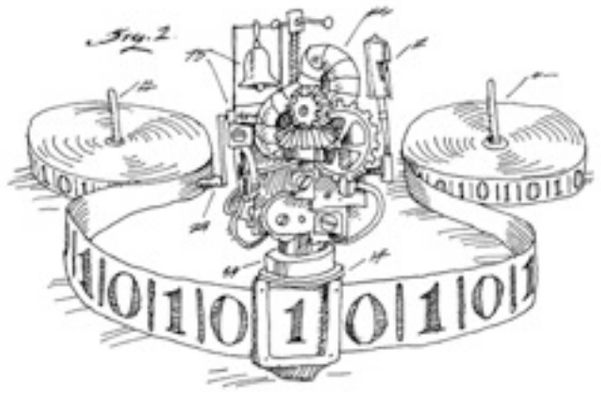
Rappels de la leçon 1

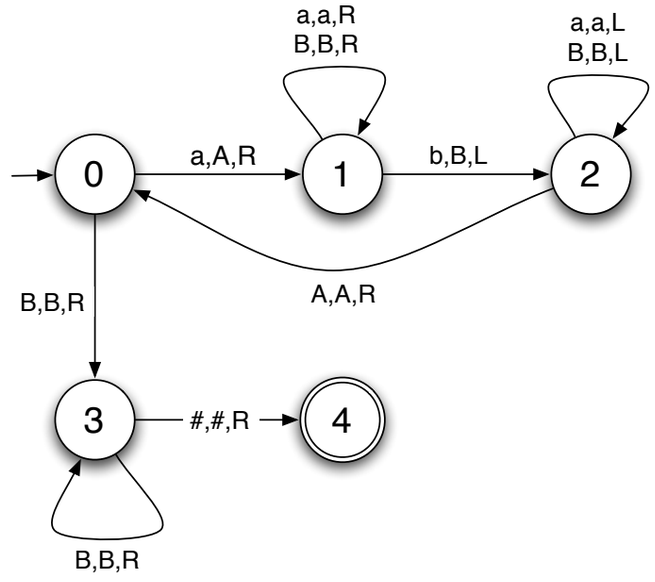
- Résoudre un problème revient donc à trouver une procédure automatique, une «machine», qui est capable de dire si un mot appartient ou non à un ensemble donné
- Pour formaliser cette notion, on utilise la machine de Turing
  - Une machine de Turing est un automate qui dispose d'un ruban infini sur lequel elle peut lire et écrire en guise de mémoire.

Rappels de la leçon 1

- Résoudre un problème revient donc à trouver une procédure automatique, une «machine», qui est capable de dire si un mot appartient ou non à un ensemble donné
- Pour formaliser cette notion, on utilise la machine de Turing
  - Une machine de Turing est un automate qui dispose d'un ruban infini sur lequel elle peut lire et écrire en guise de mémoire.

Alan Turing





Rappels de la leçon 1

Rappels de la leçon 1

- Résoudre un problème revient donc à trouver une machine de Turing qui reconnaît le langage associé au problème

Rappels de la leçon 1

- Résoudre un problème revient donc à trouver une machine de Turing qui reconnaît le langage associé au problème
- Malheureusement, on a vu qu'il existe plus de problèmes différents que de machines différentes

Rappels de la leçon 1

- Résoudre un problème revient donc à trouver une machine de Turing qui reconnaît le langage associé au problème
- Malheureusement, on a vu qu'il existe plus de problèmes différents que de machines différentes
- Il existe donc nécessairement des problèmes pour lesquels on ne peut pas trouver de procédure automatique de résolution

Rappels de la leçon 1

- Résoudre un problème revient donc à trouver une machine de Turing qui reconnaît le langage associé au problème
- Malheureusement, on a vu qu'il existe plus de problèmes différents que de machines différentes
- Il existe donc nécessairement des problèmes pour lesquels on ne peut pas trouver de procédure automatique de résolution
- Ces problèmes sont appelés problèmes **indécidables**
  - Ces problèmes sont donc considérés hors de portée des ordinateurs, car tout semble indiquer que le modèles de la machine de Turing correspond exactement à ce que les ordinateurs (tels que nous les connaissons) peuvent faire

Rappels de la leçon 1

Rappels de la leçon 1

- Une classe de problèmes intéressants est celle des problèmes qui posent une question sur les machines de Turing
  - Par exemple : décider si une machine de Turing a une exécution infinie

Rappels de la leçon 1

- Une classe de problèmes intéressants est celle des problèmes qui posent une question sur les machines de Turing
  - Par exemple : décider si une machine de Turing a une exécution infinie
- Malheureusement le théorème de Rice nous dit que « *Toute propriété non-triviale sur les langages des machines de Turing est indécidable* »
  - Intuitivement, on ne peut donc pas trouver de machine de Turing qui analyse les machines de Turing...
  - ... ou les programmes informatiques !

Rappels de la leçon 1

- Une classe de problèmes intéressants est celle des problèmes qui posent une question sur les machines de Turing
  - Par exemple : décider si une machine de Turing a une exécution infinie
- Malheureusement le théorème de Rice nous dit que « *Toute propriété non-triviale sur les langages des machines de Turing est indécidable* »
  - Intuitivement, on ne peut donc pas trouver de machine de Turing qui analyse les machines de Turing...
  - ... ou les programmes informatiques !
- Les questions intéressantes sur la sémantique des programmes informatiques (« ce programme est-il correct ? ») sont donc indécidables

Indécidabilité de l'arrêt

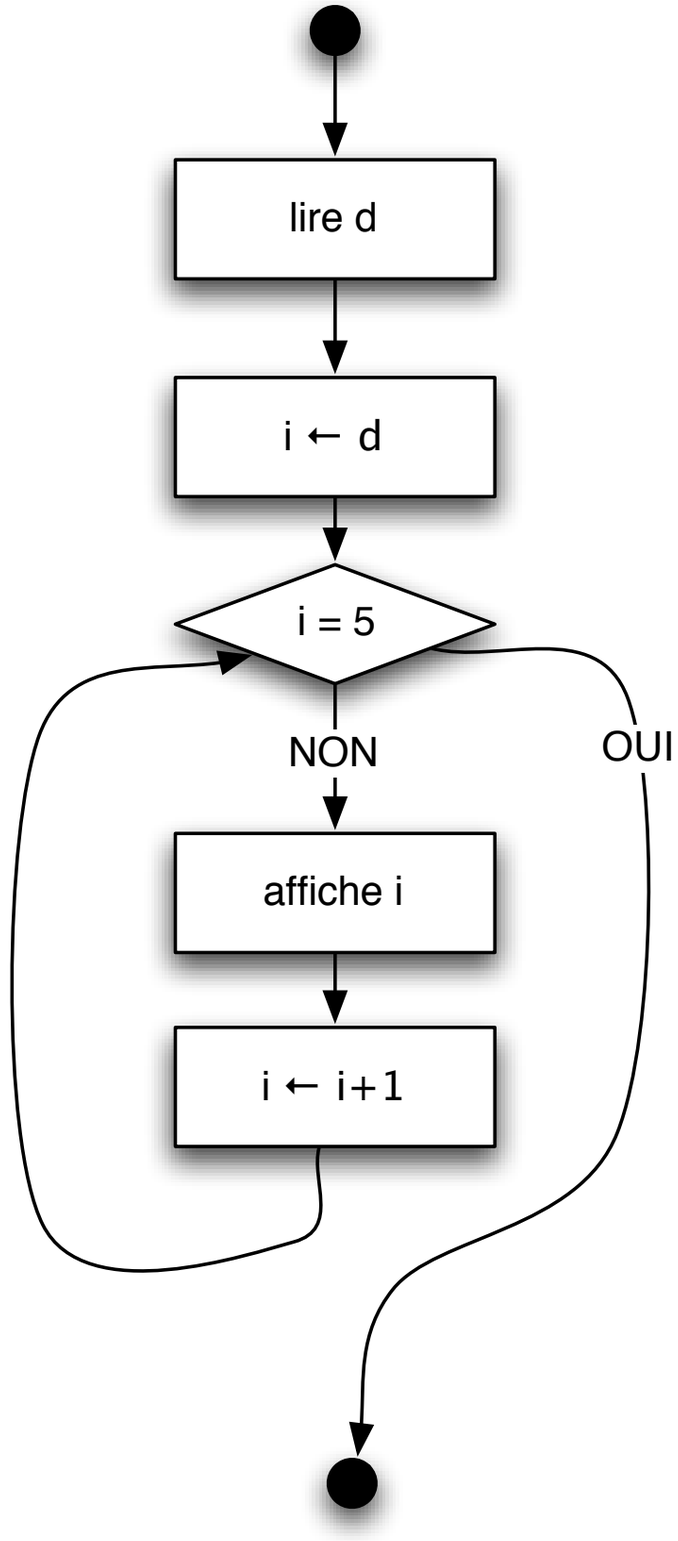
Exemple

Exemple

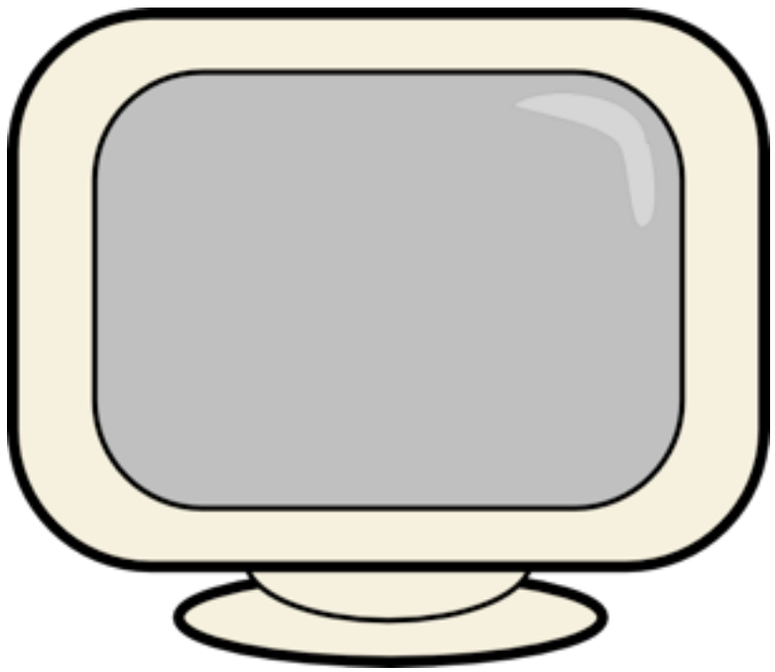
- On voudrait un programme qui prend une valeur d en entrée et qui affiche tous les nombres compris entre d et 4 (bornes comprises)
- On propose le programme ci-contre

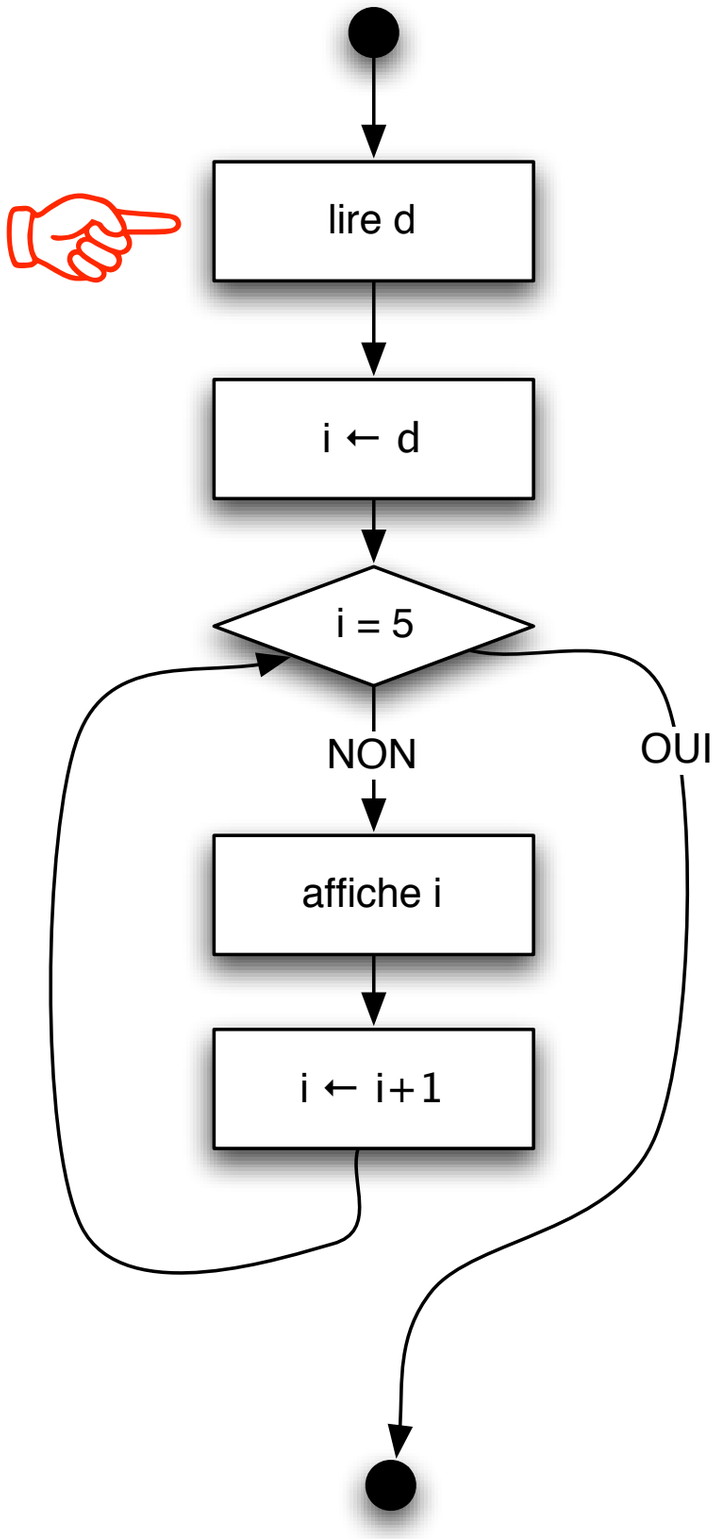
Exemple

- On voudrait un programme qui prend une valeur d en entrée et qui affiche tous les nombres compris entre d et 4 (bornes comprises)
- On propose le programme ci-contre



Exemple

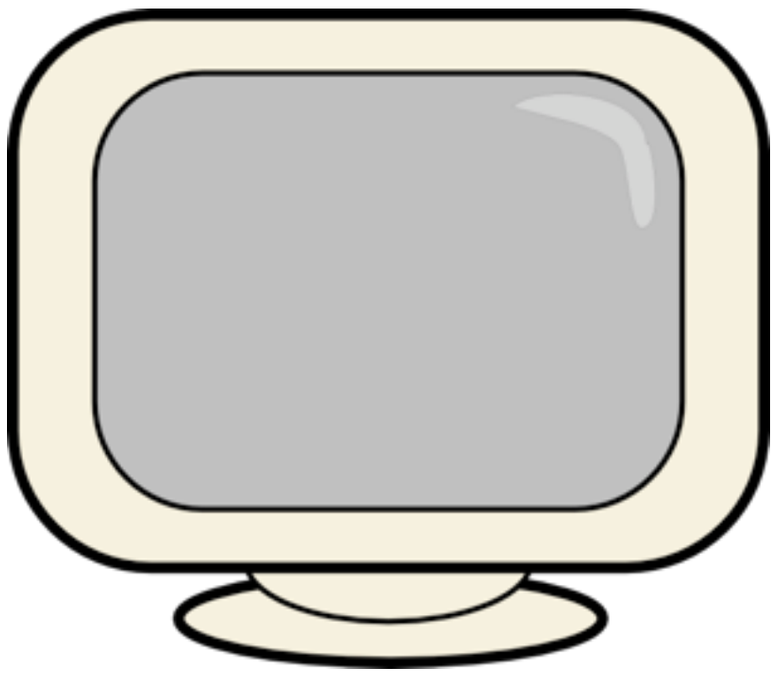


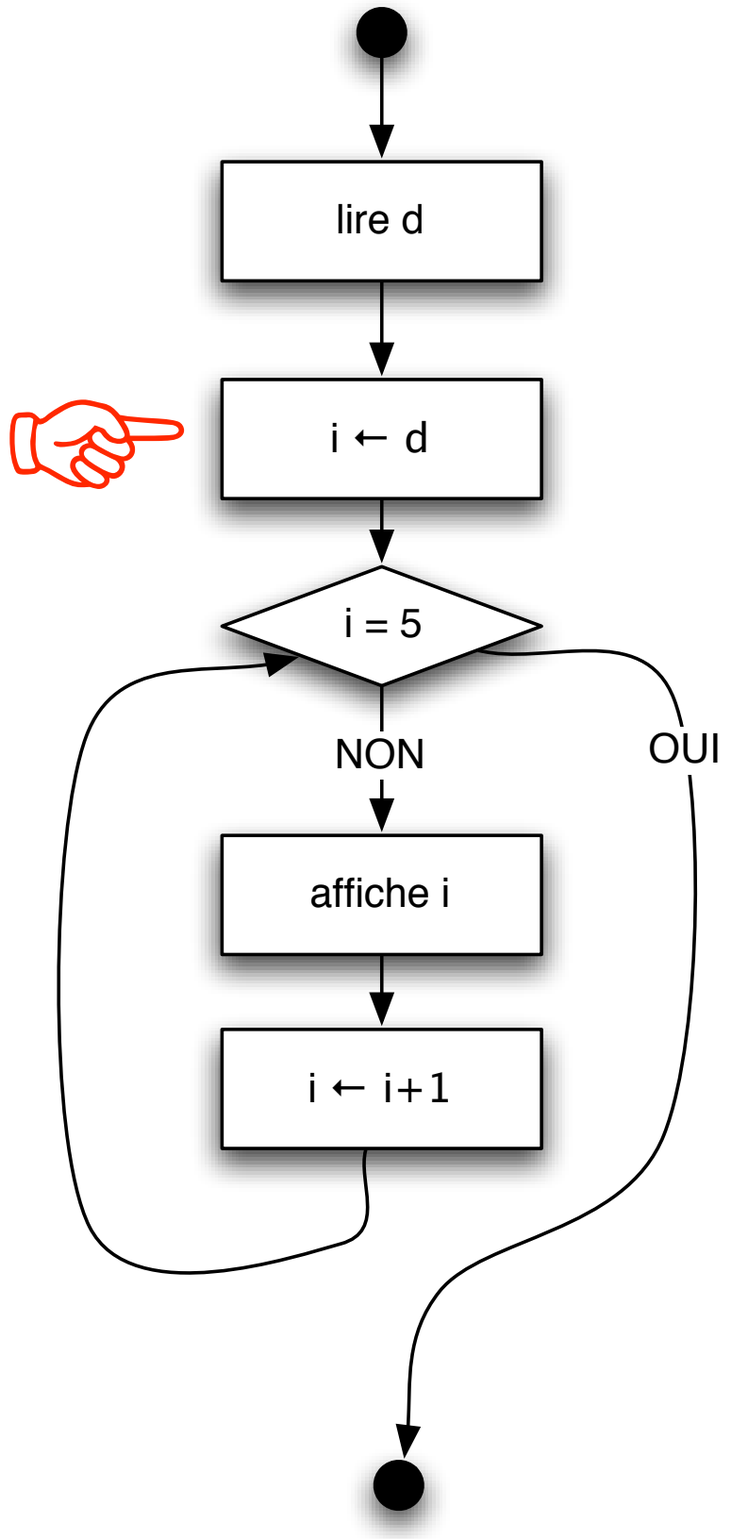


d =

i =

Exemple

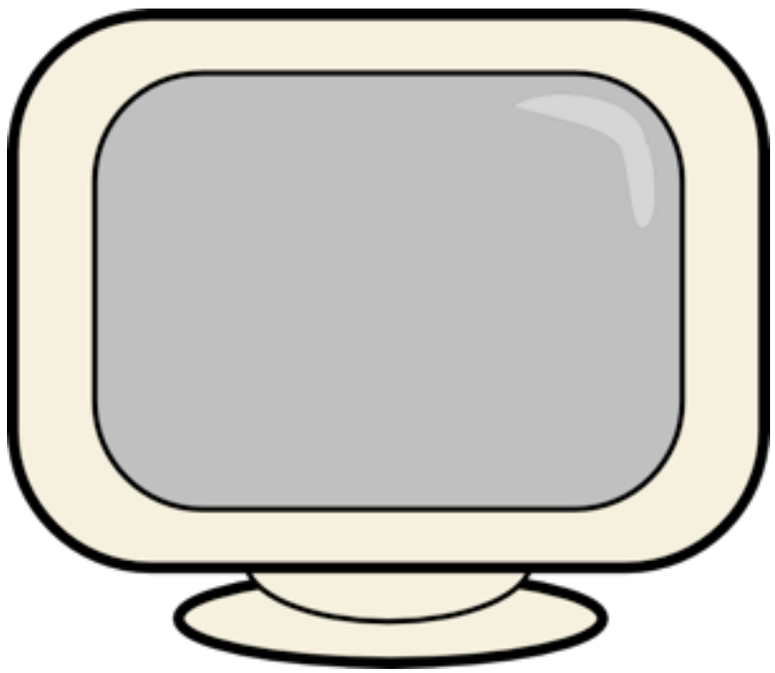


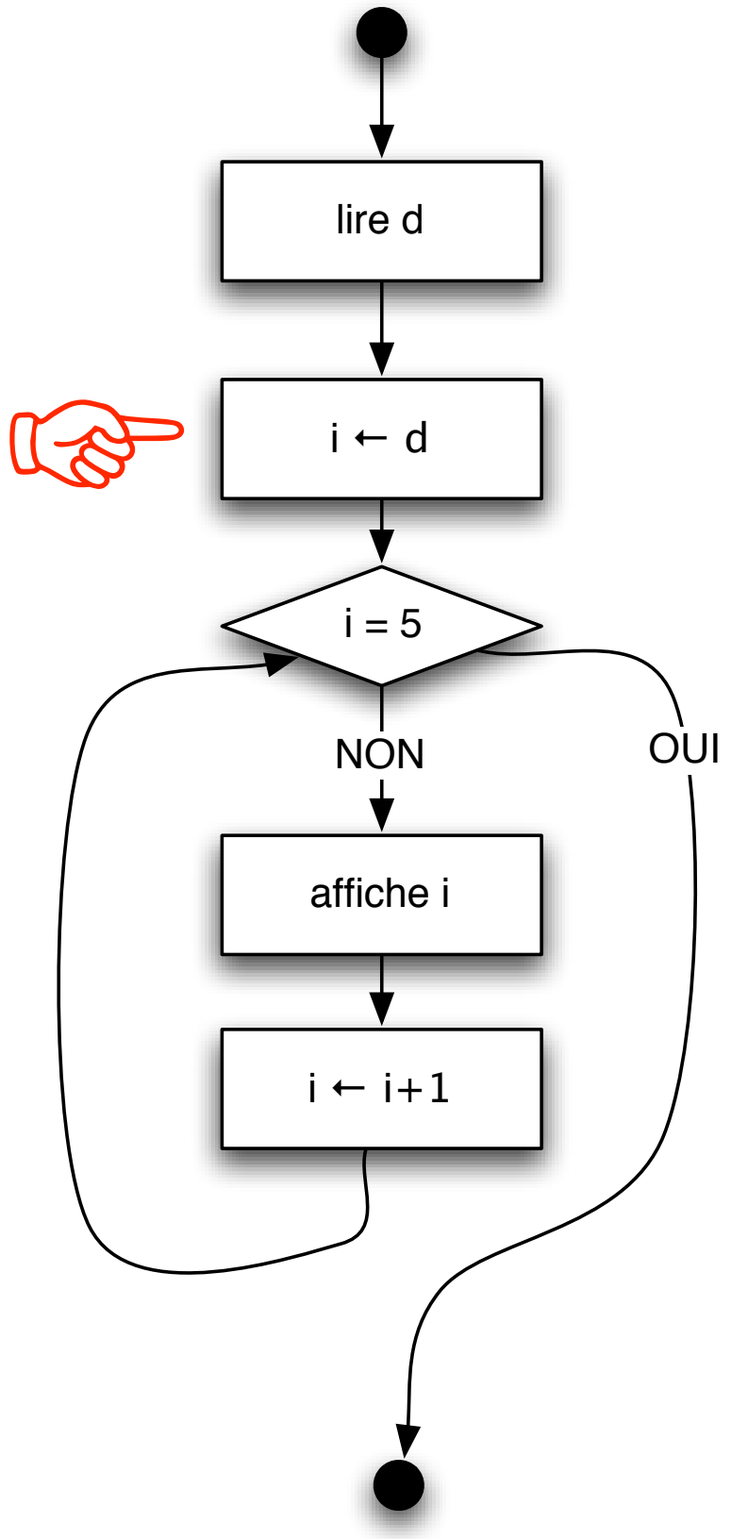


d = 3

i =

Exemple

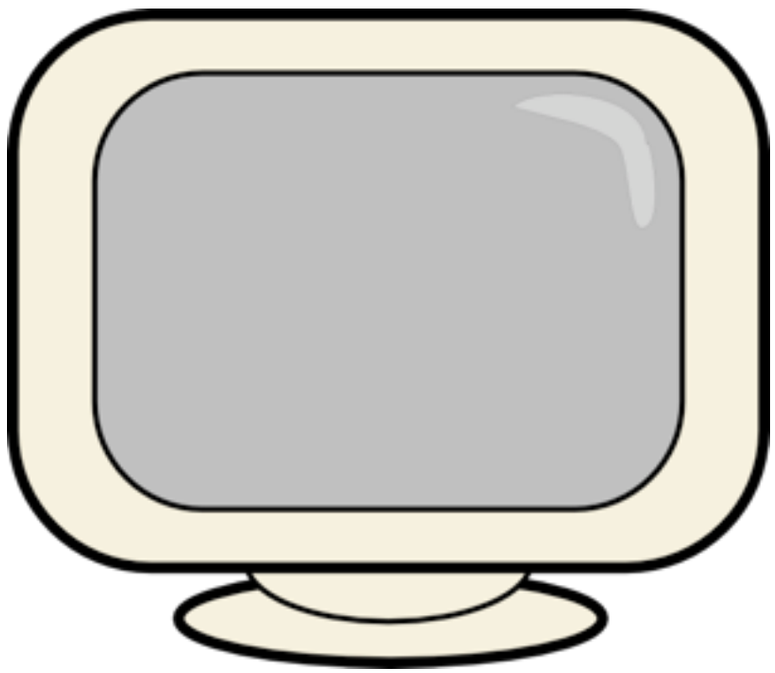


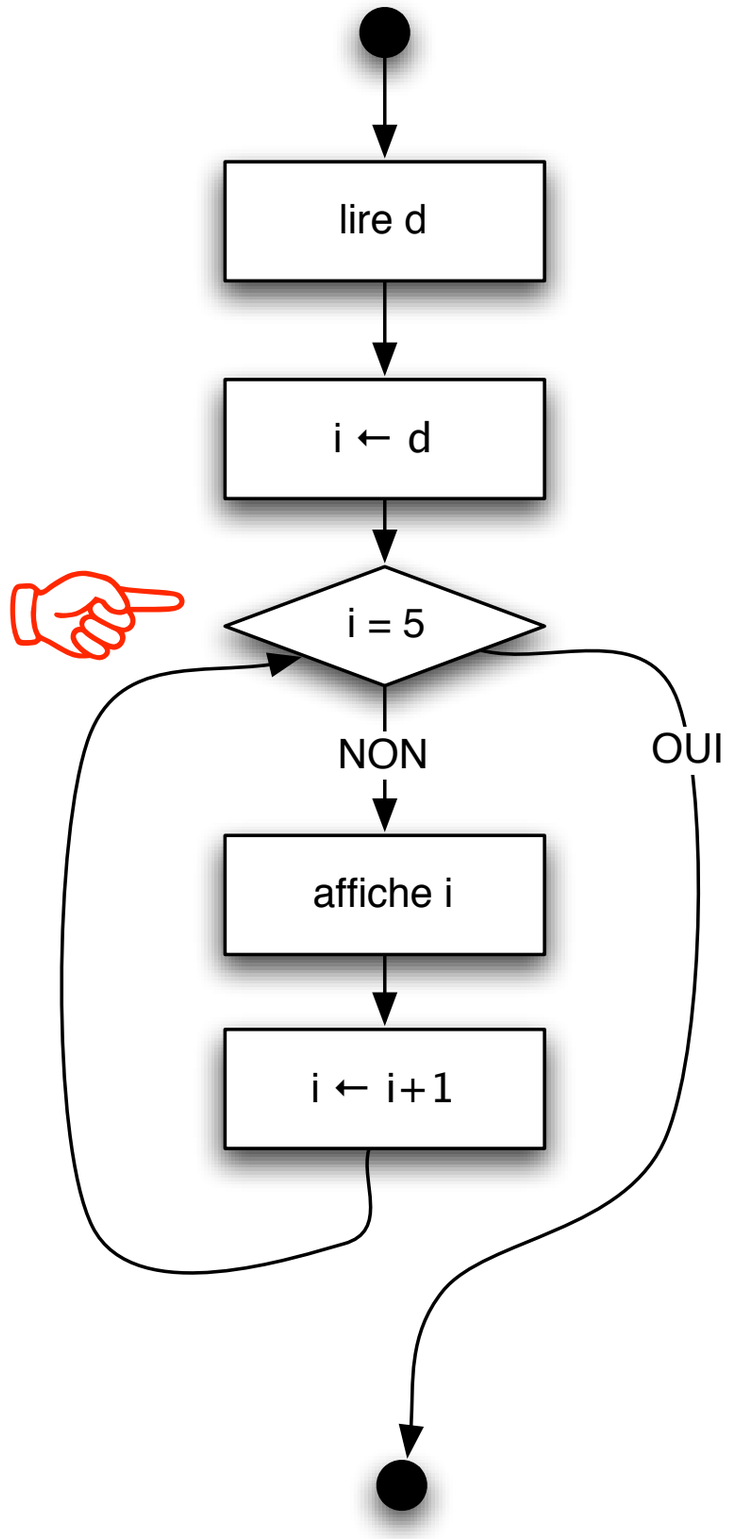


d = 3

i = 3

Exemple

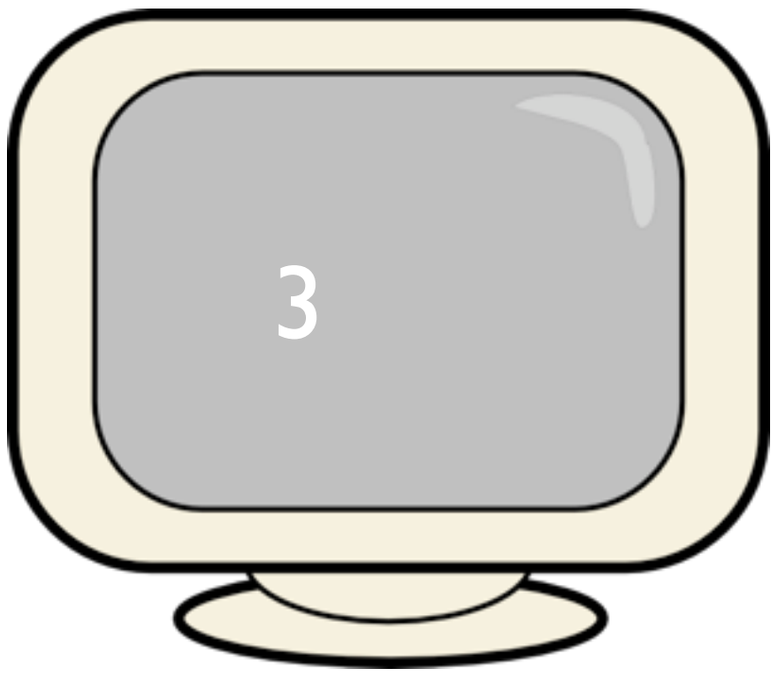


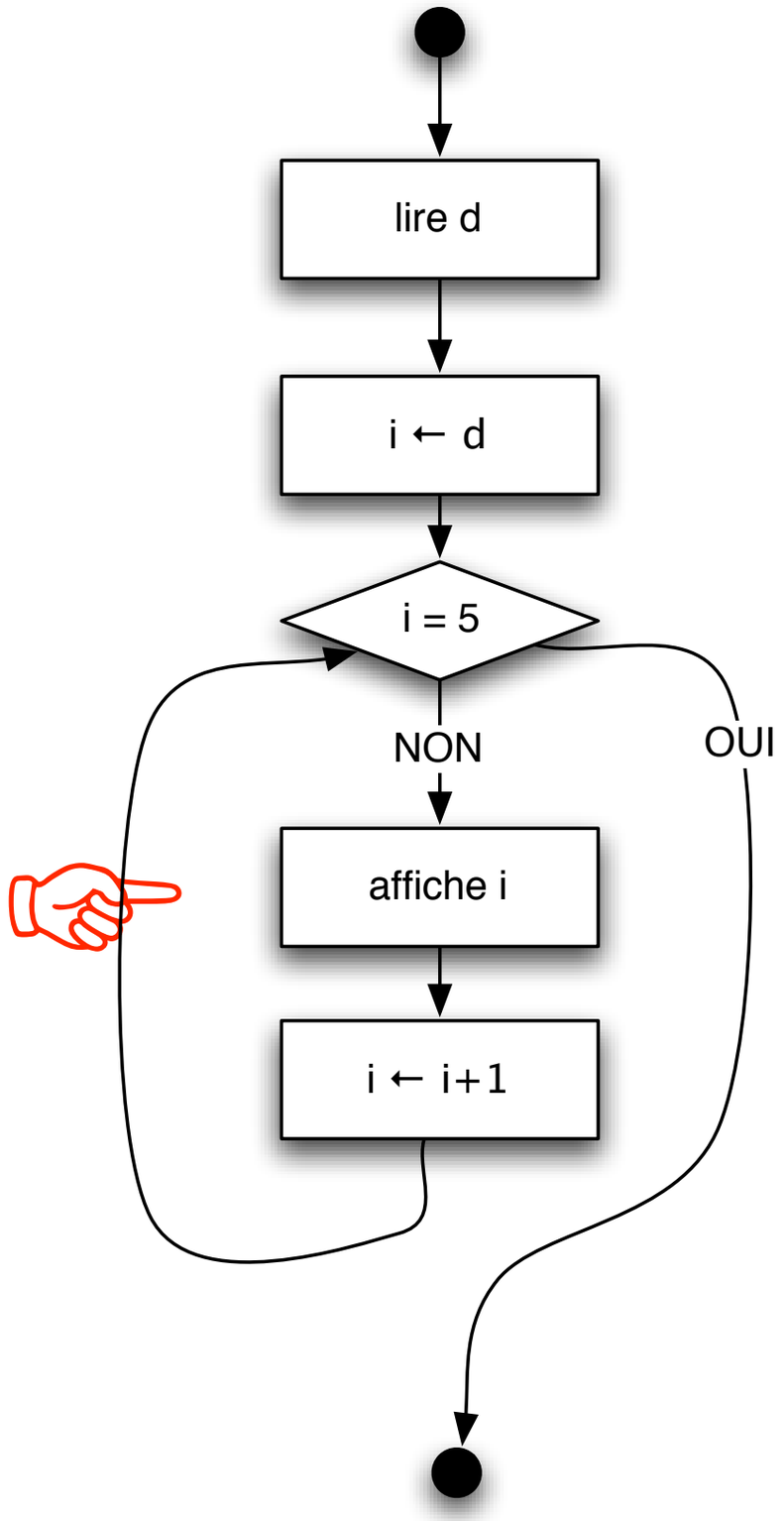


d = 3

i = 3

Exemple

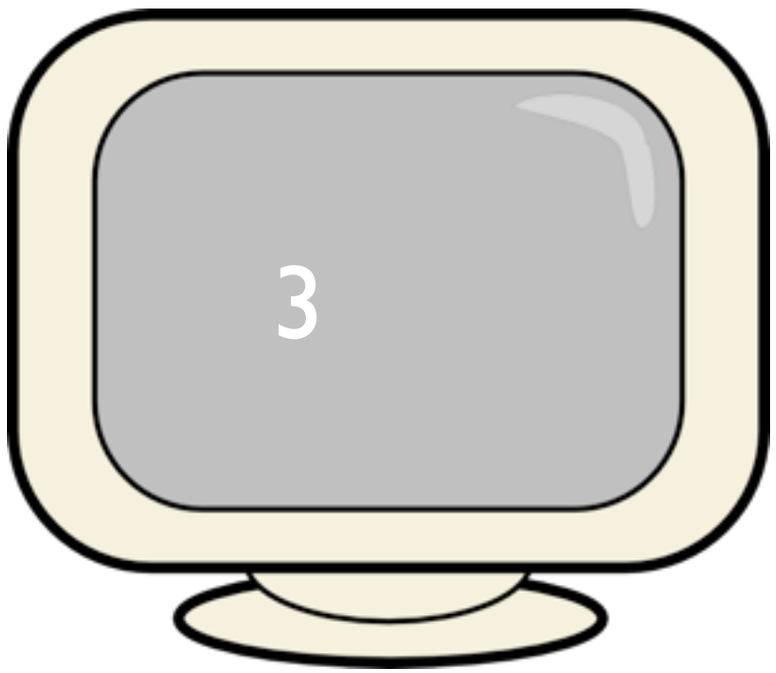


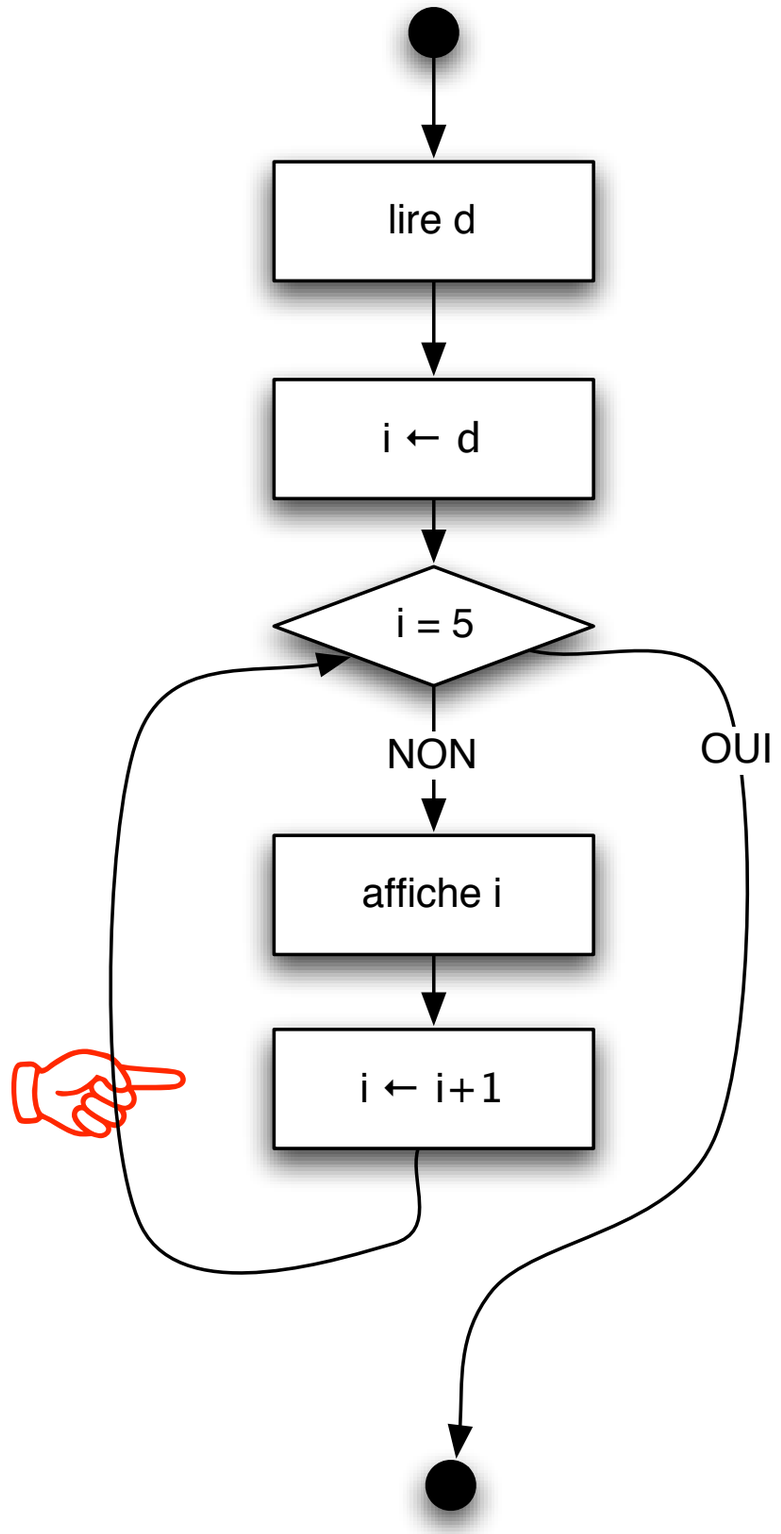


d = 3

i = 3

Exemple

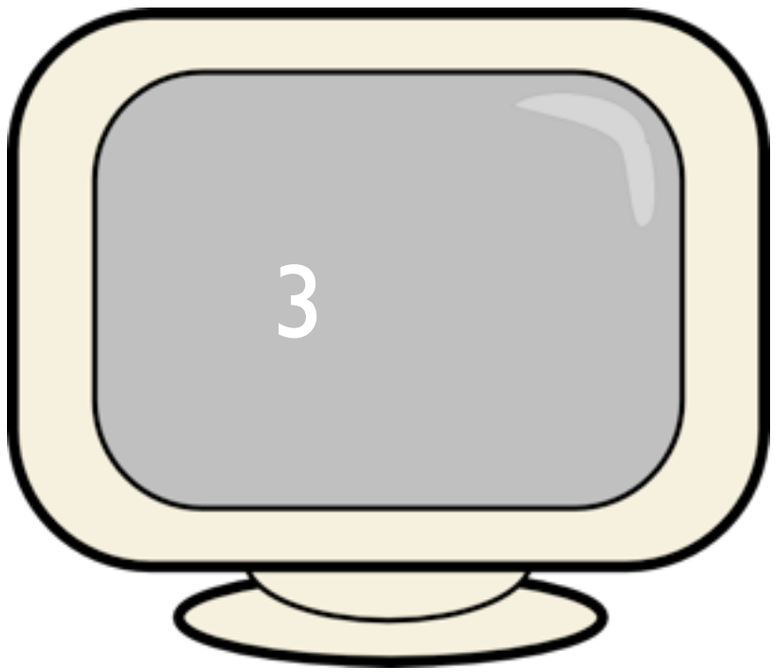


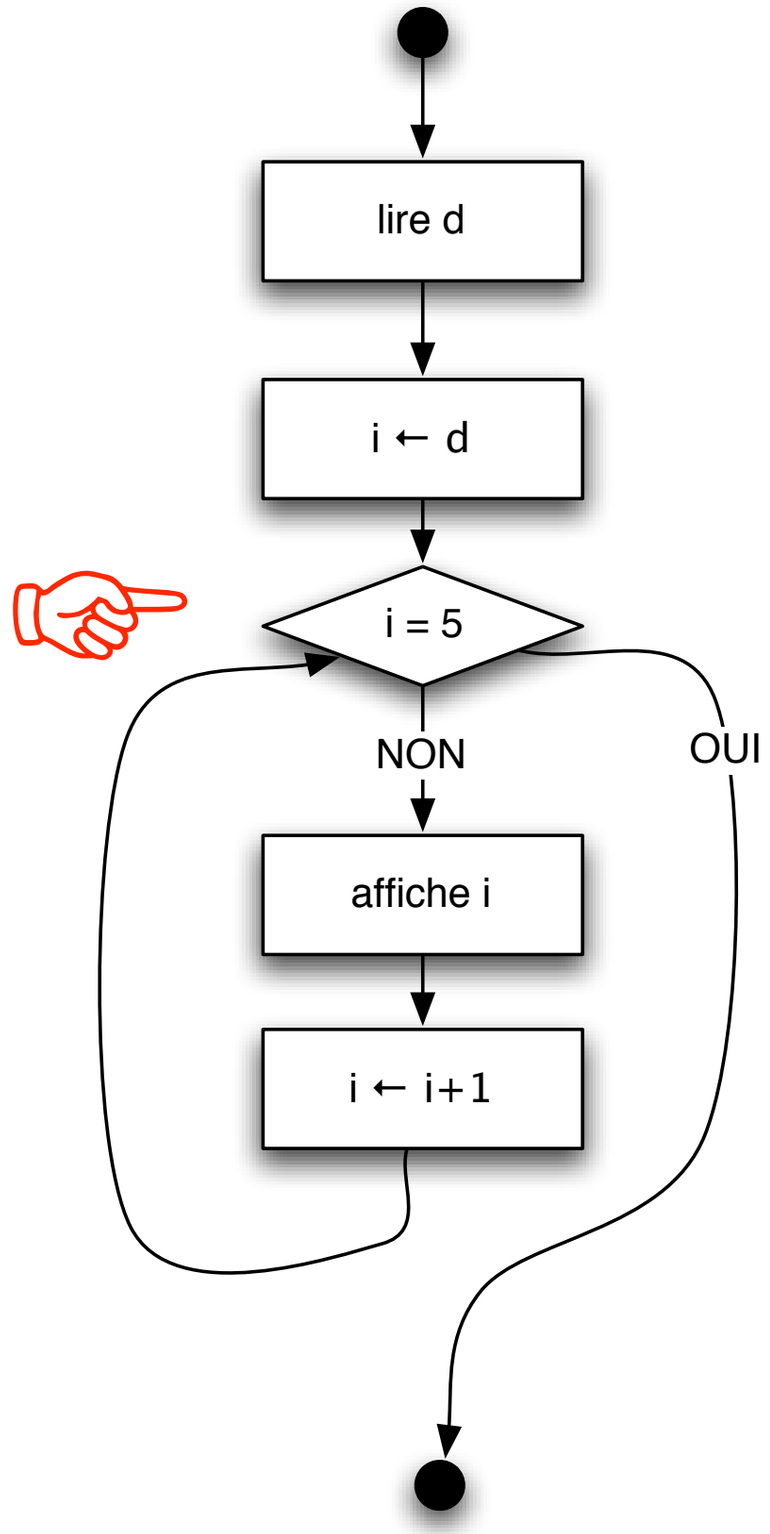


d = 3

i = 4

Exemple

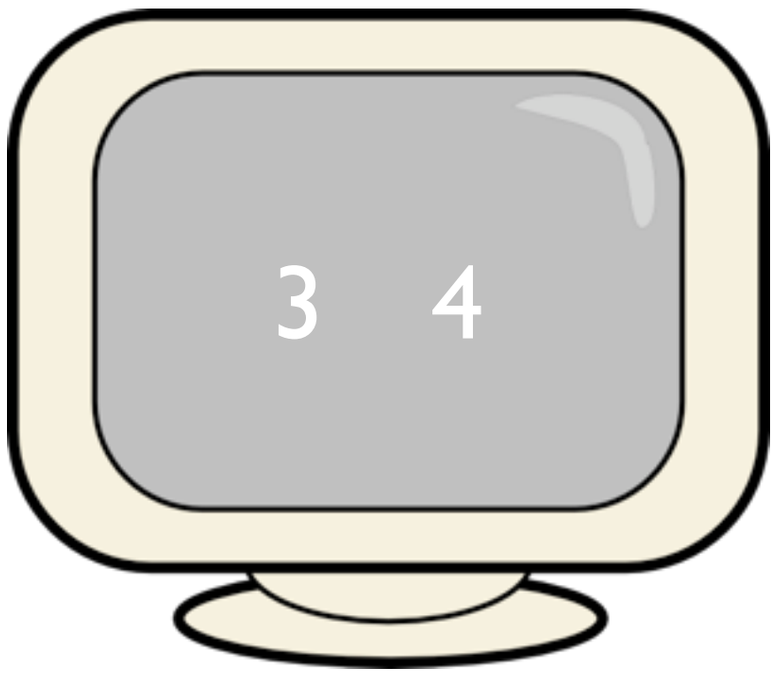


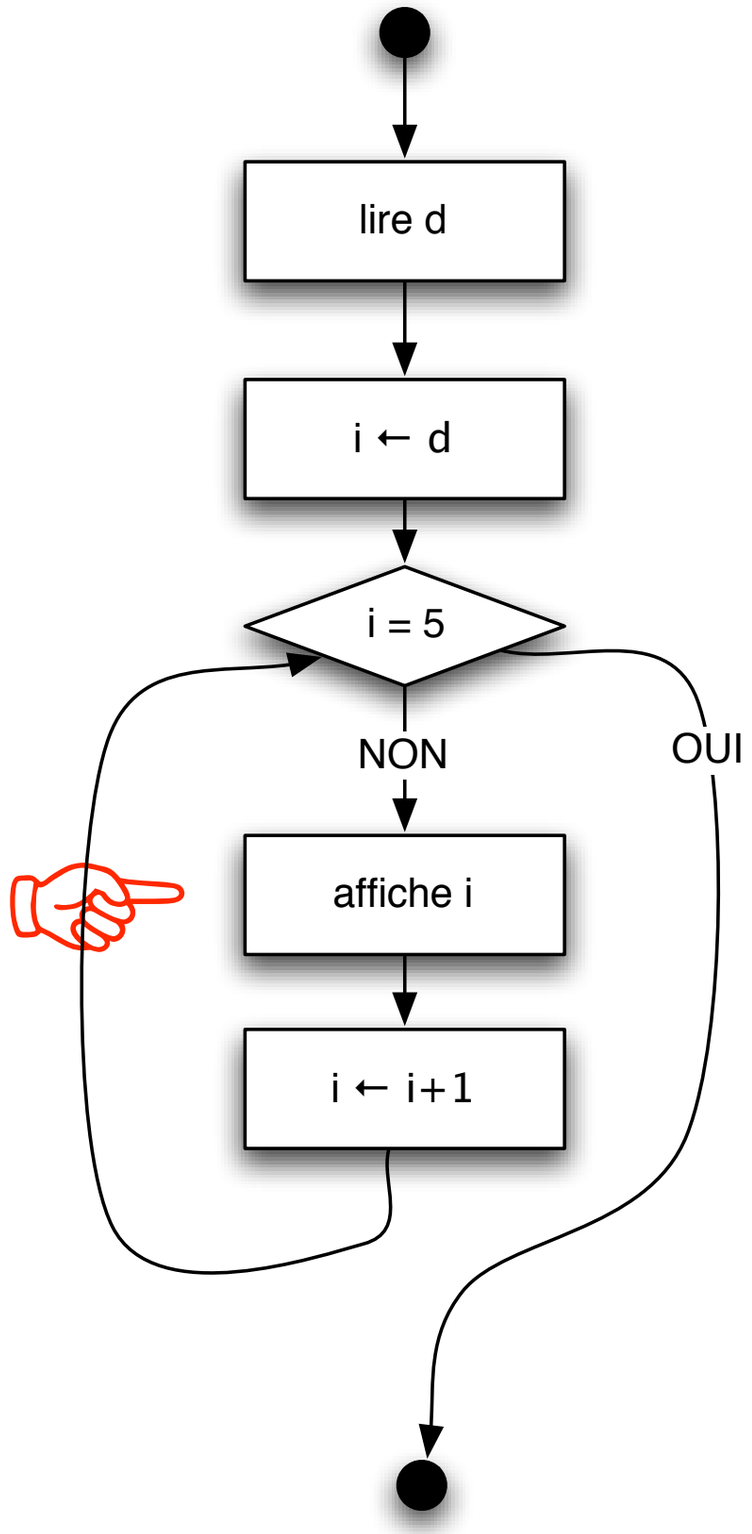


d = 3

i = 4

Exemple

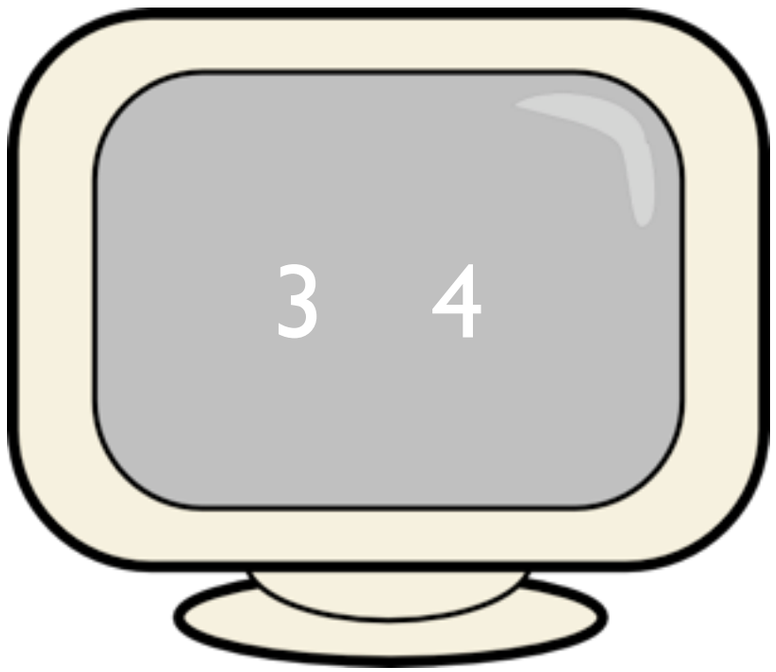


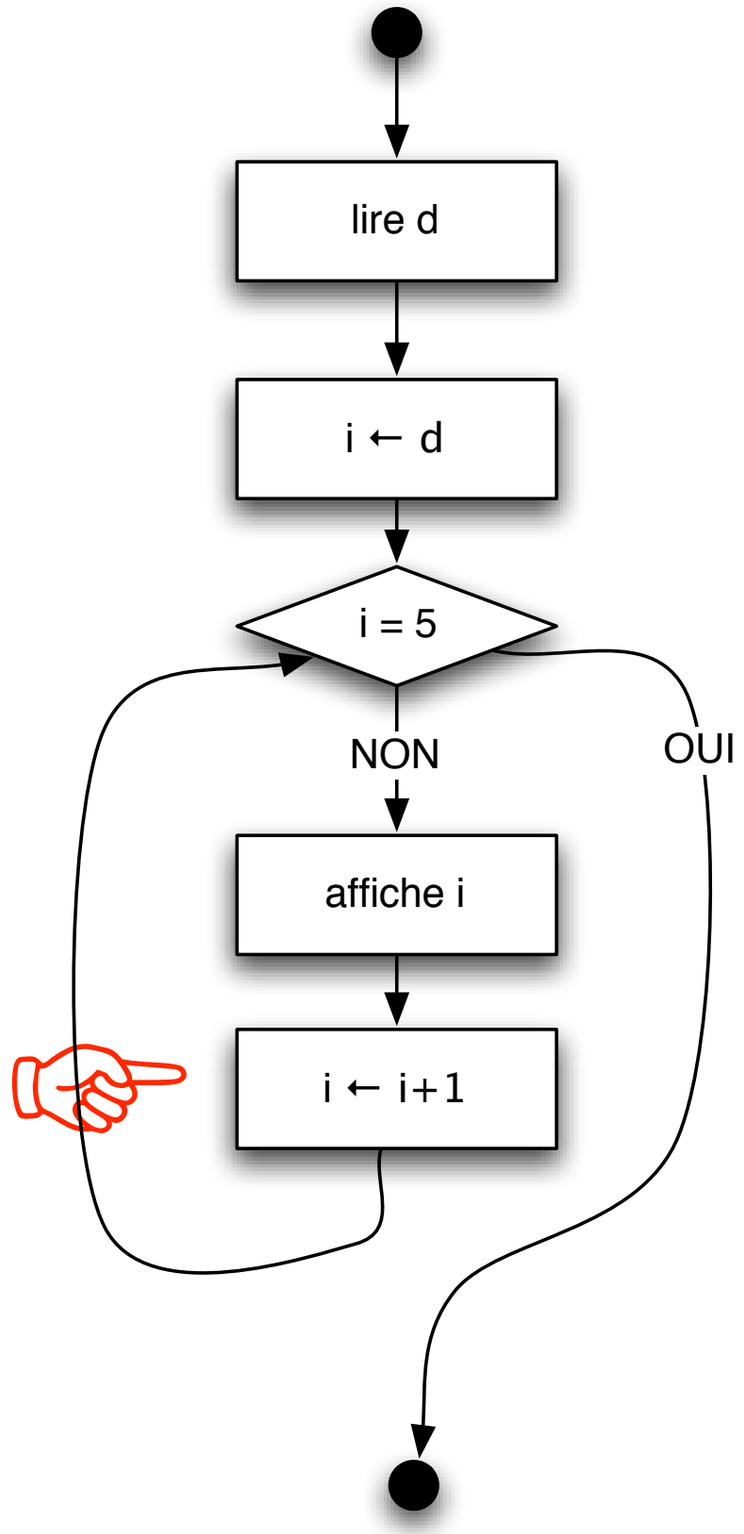


d = 3

i = 4

Exemple

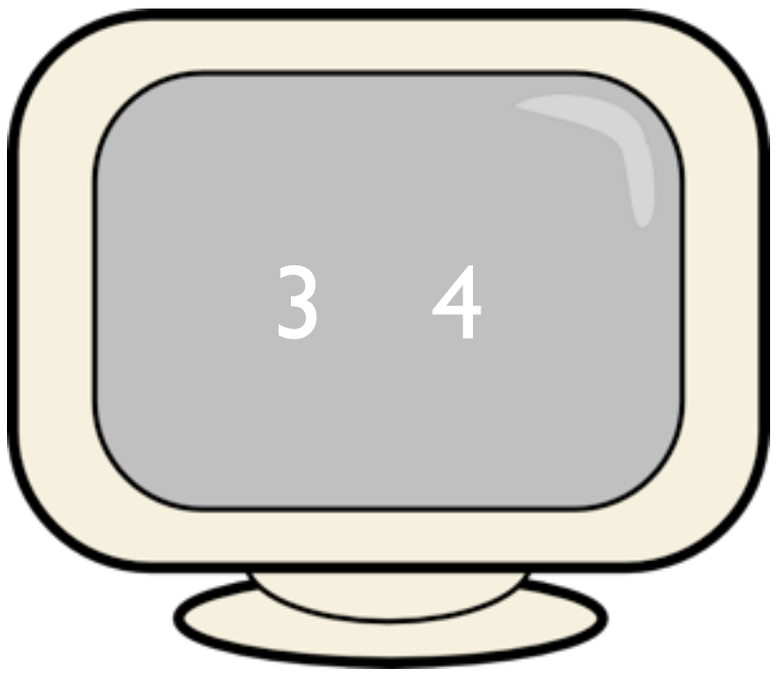


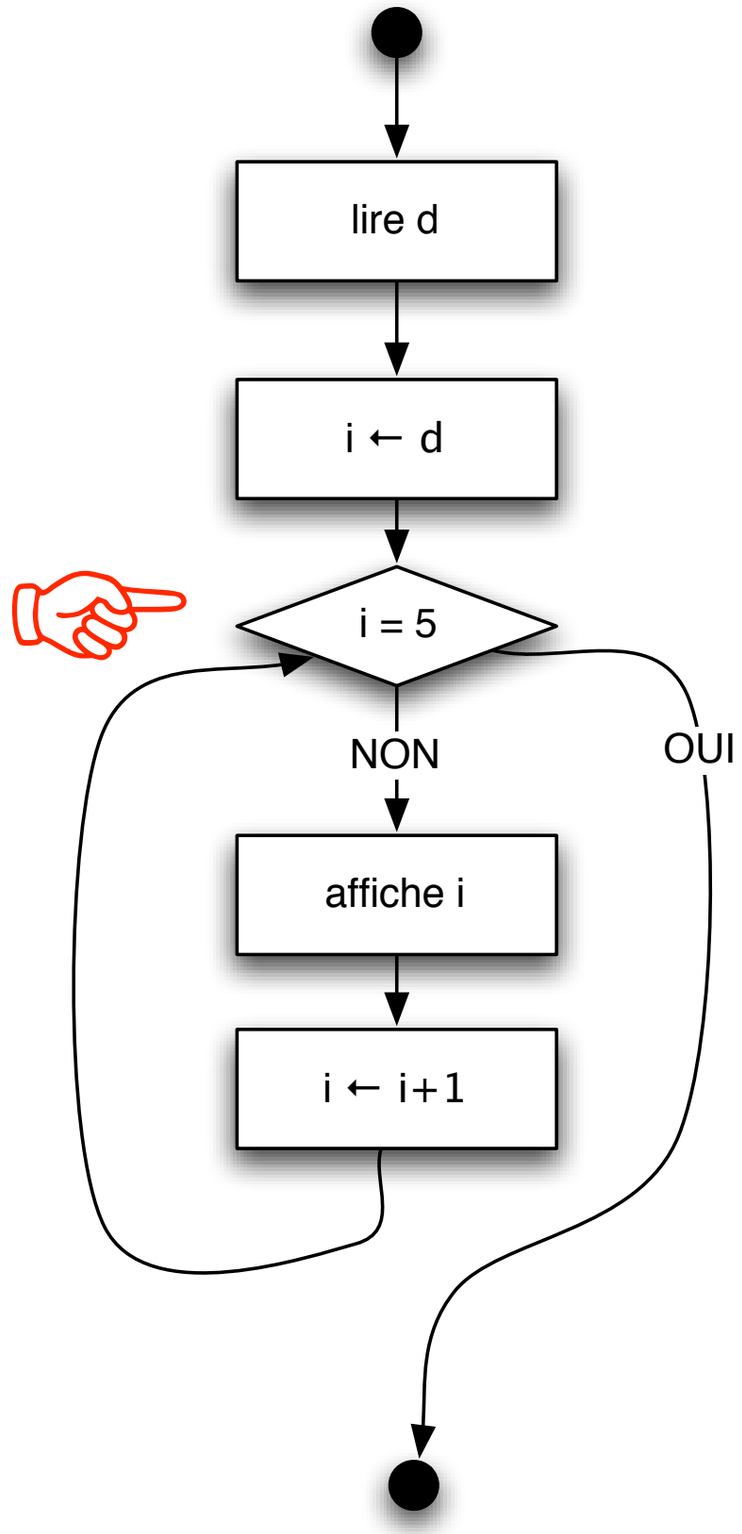


d = 3

i = 5

Exemple

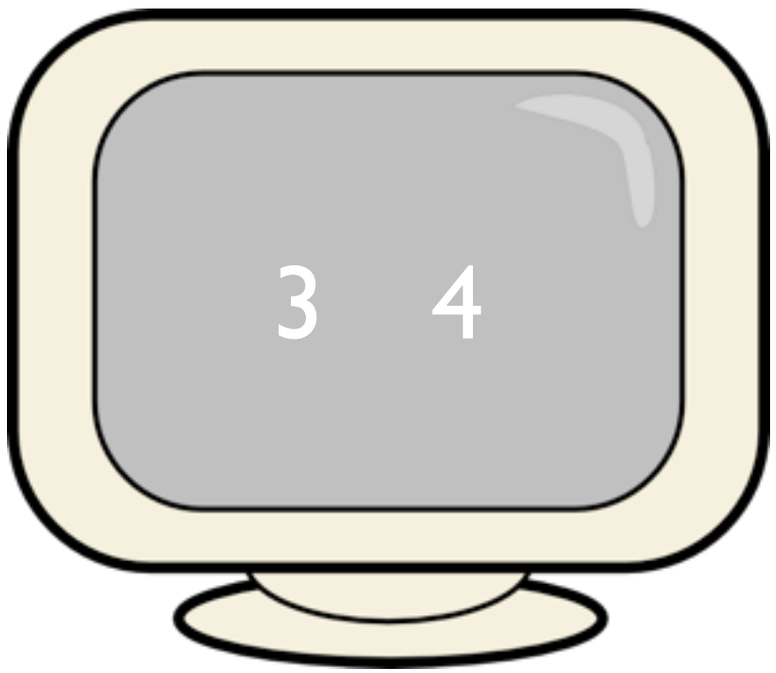


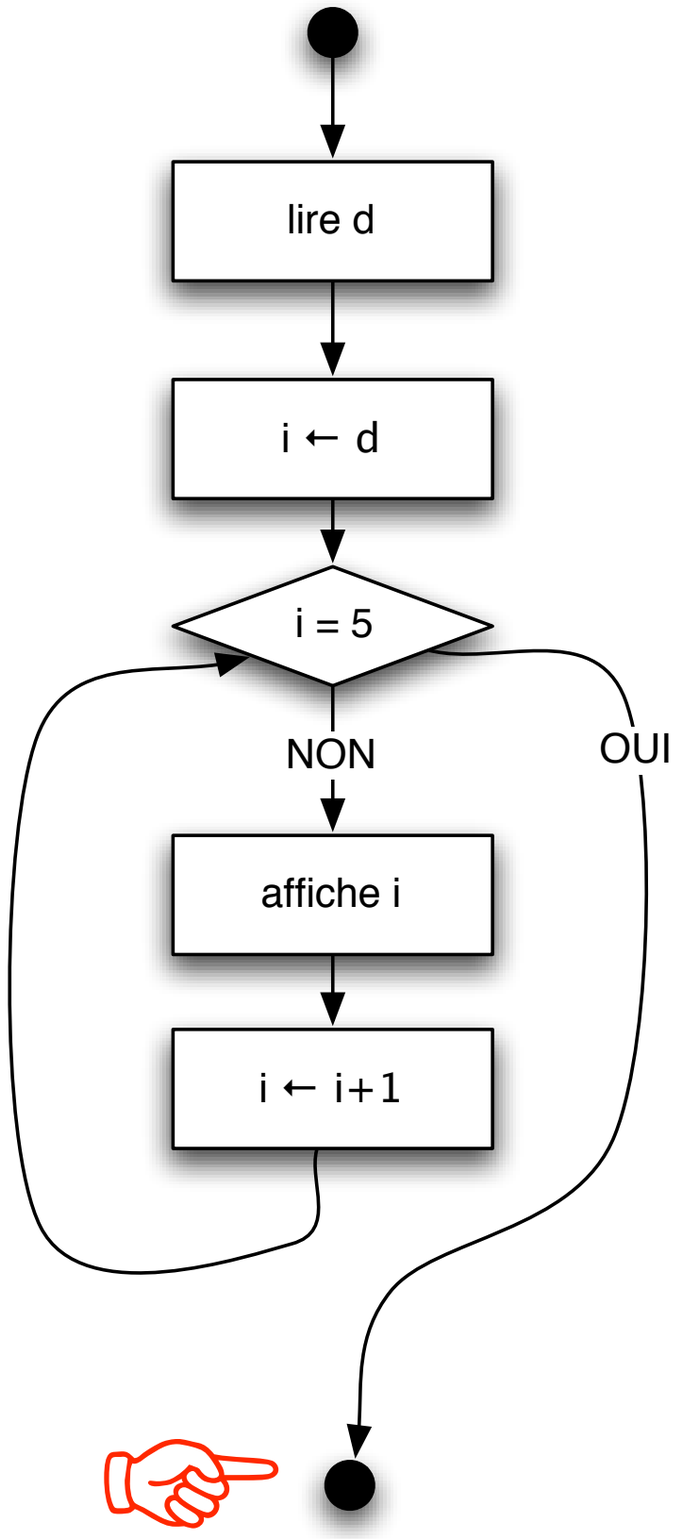


d = 3

i = 5

Exemple





d = 3

i = 5

Question

Question

- Ce programme a-t-il une exécution finie pour toute valeur en entrée ?

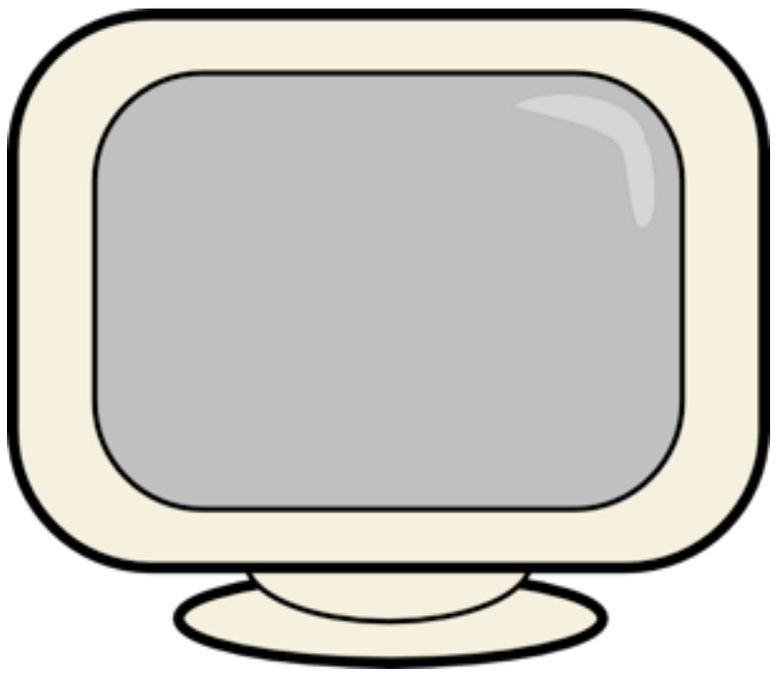
Question

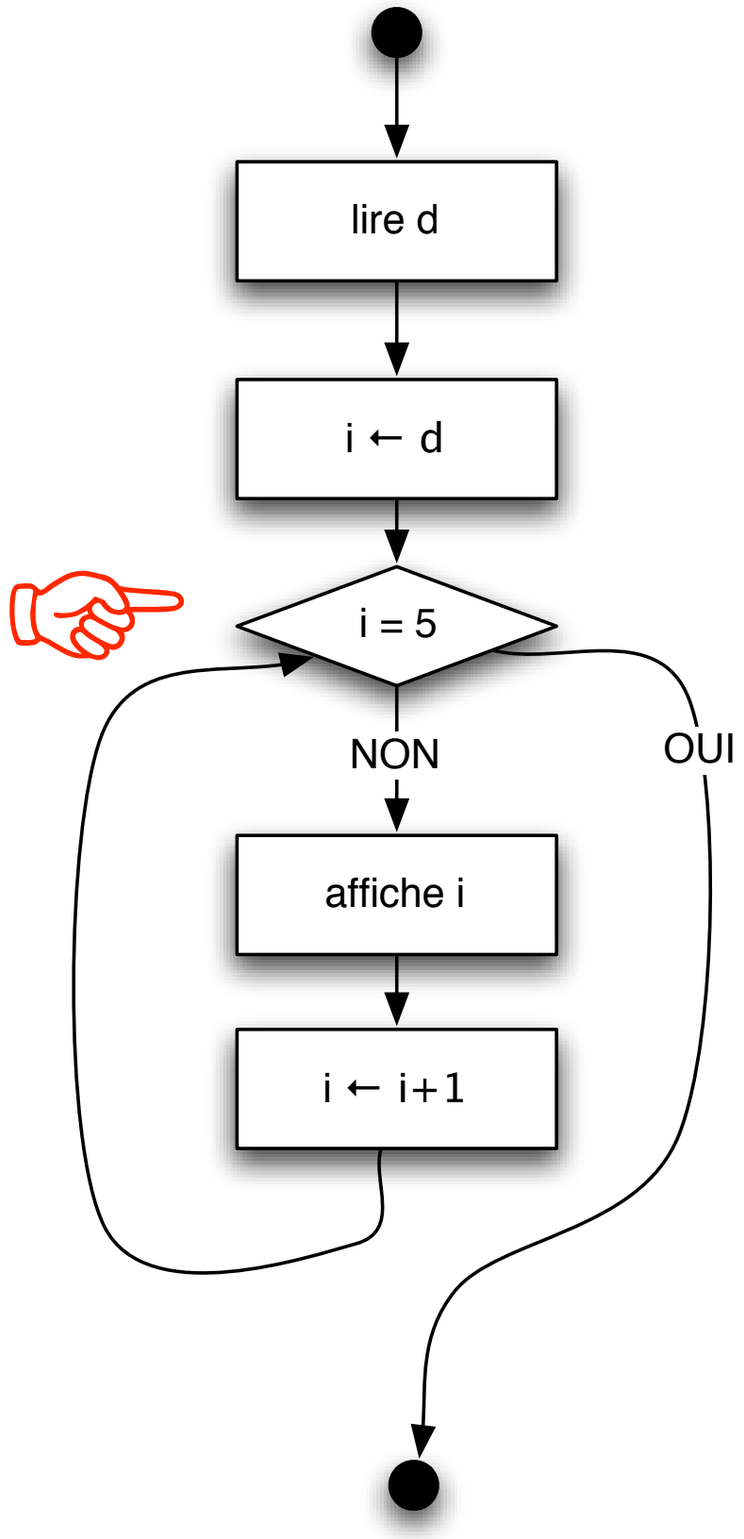
- Ce programme a-t-il une exécution finie pour toute valeur en entrée ?
- Non, si l'utilisateur entre une valeur > 5, le programme ne devrait rien afficher et terminer immédiatement

Question

- Ce programme a-t-il une exécution finie pour toute valeur en entrée ?
- Non, si l'utilisateur entre une valeur > 5, le programme ne devrait rien afficher et terminer immédiatement
- Au lieu de cela il «cycle» et affiche toutes les valeurs supérieures ou égales à la valeur entrée

Exemple

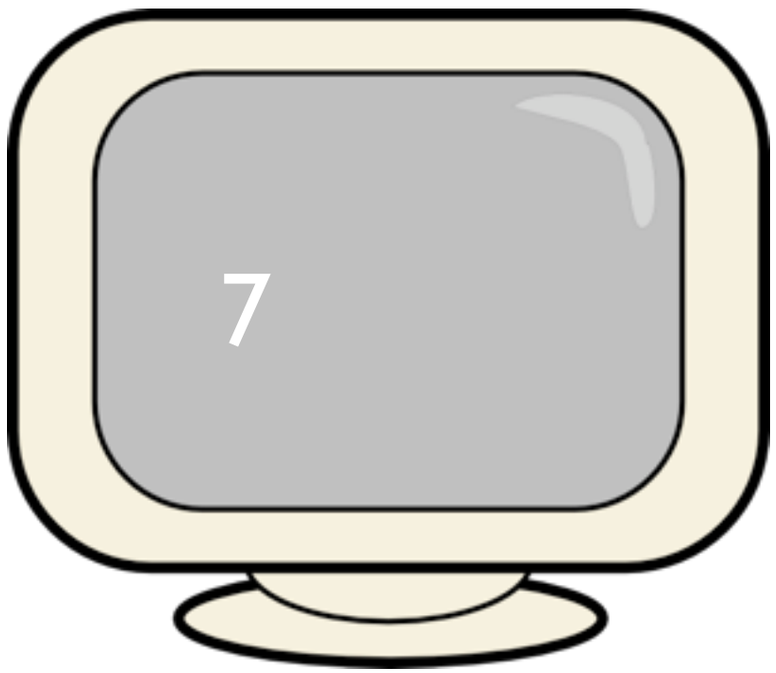


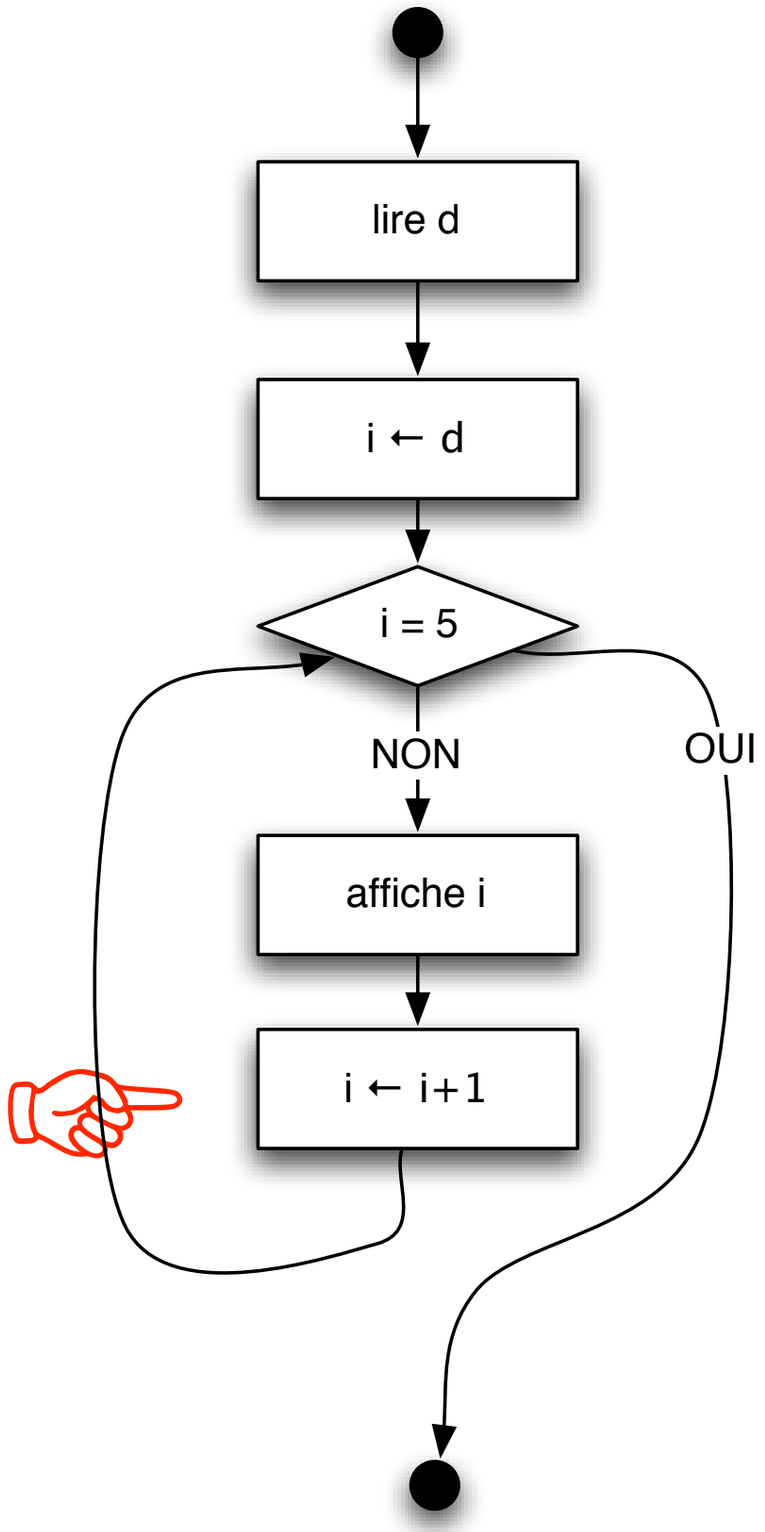


d = 7

i = 7

Exemple

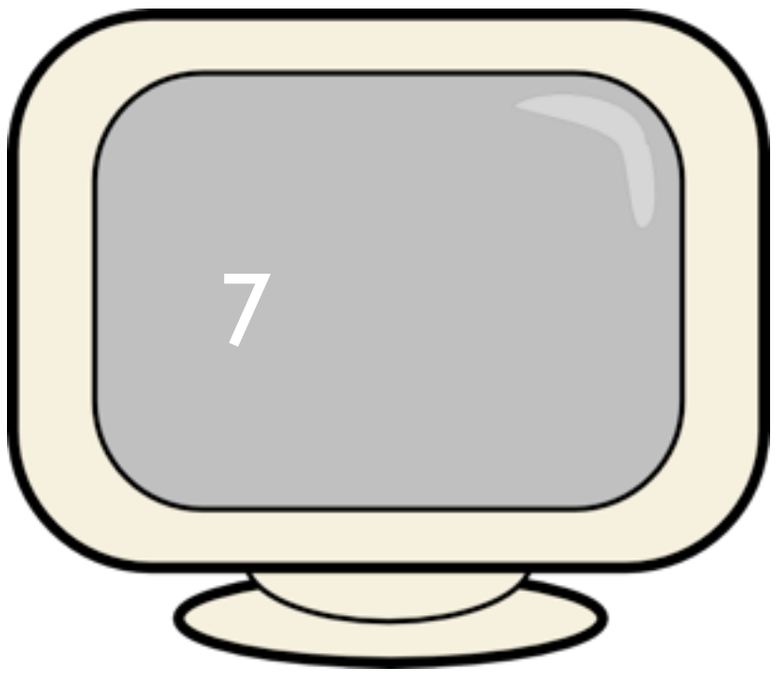


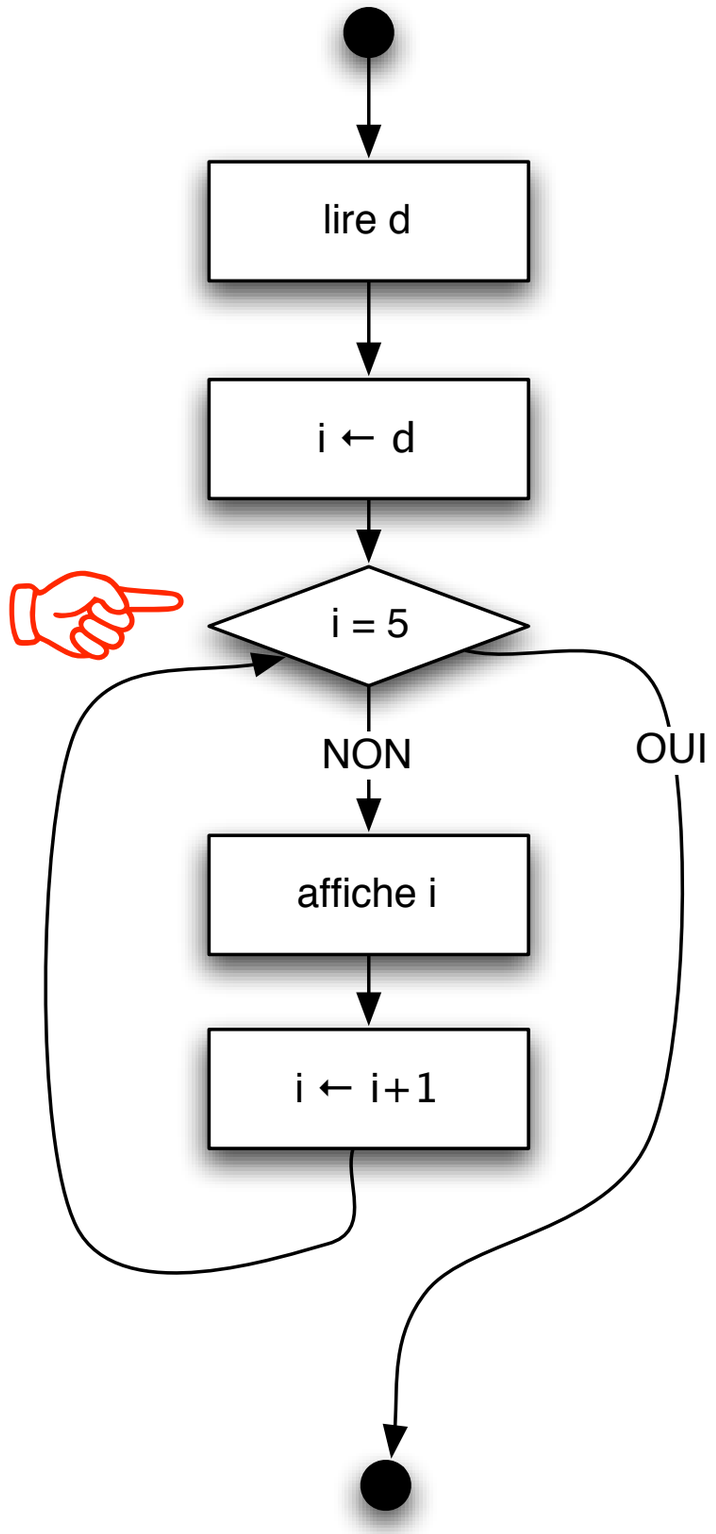


d = 7

i = 8

Exemple

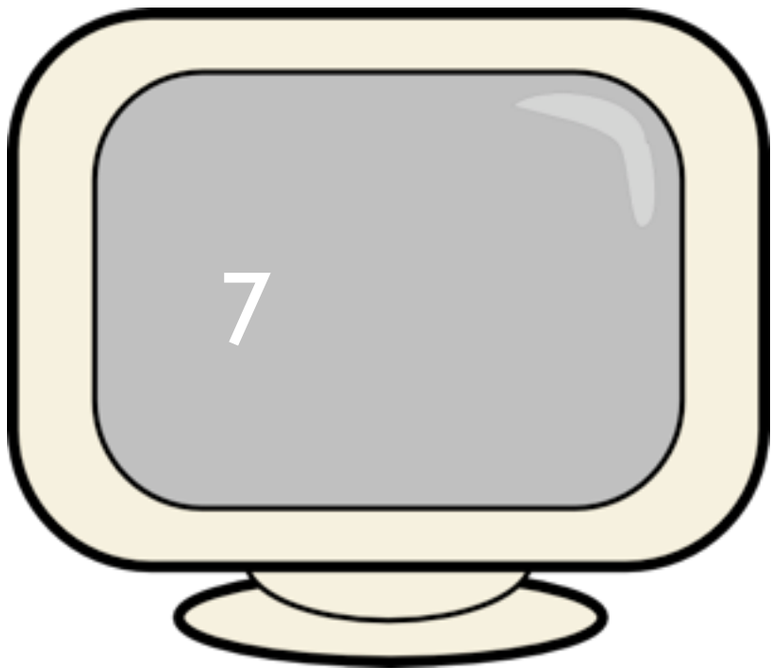


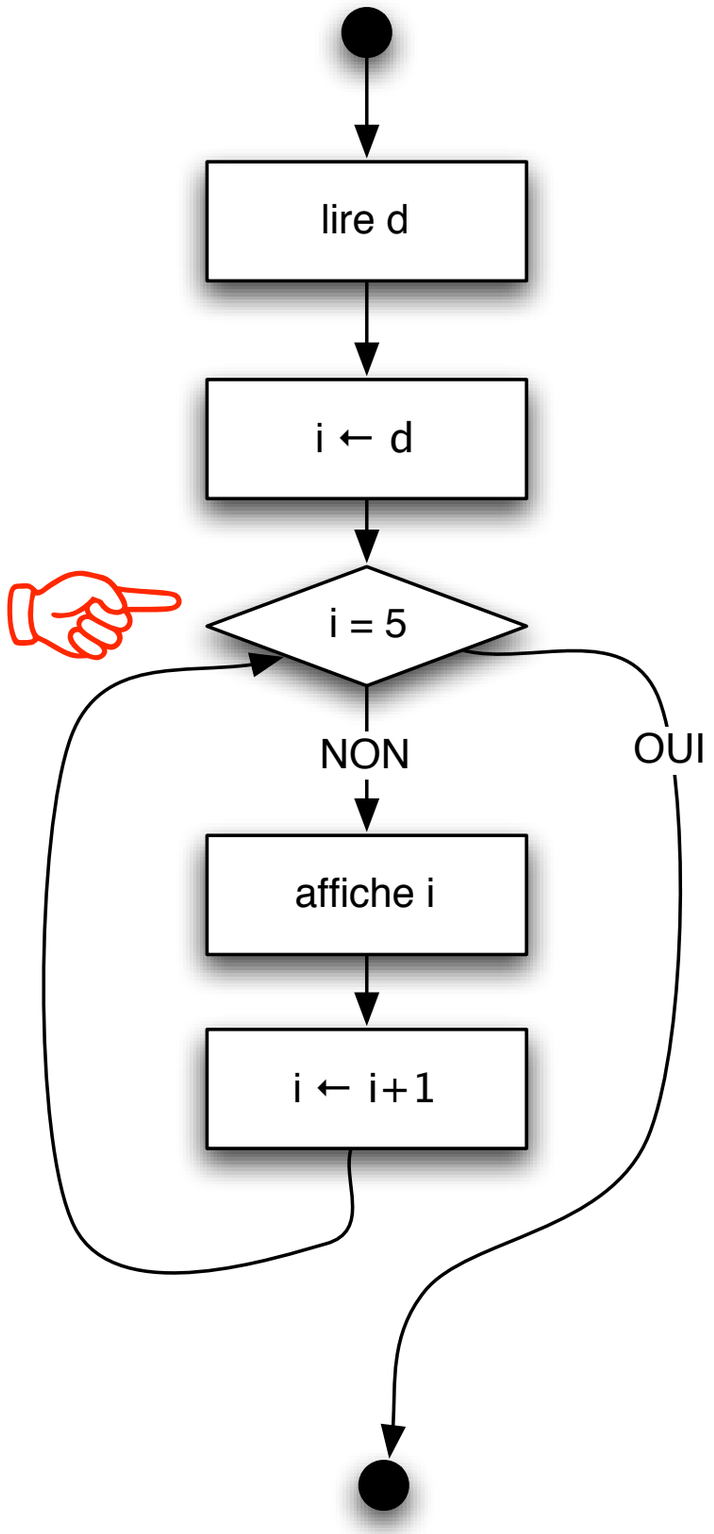


d = 7

i = 8

Exemple



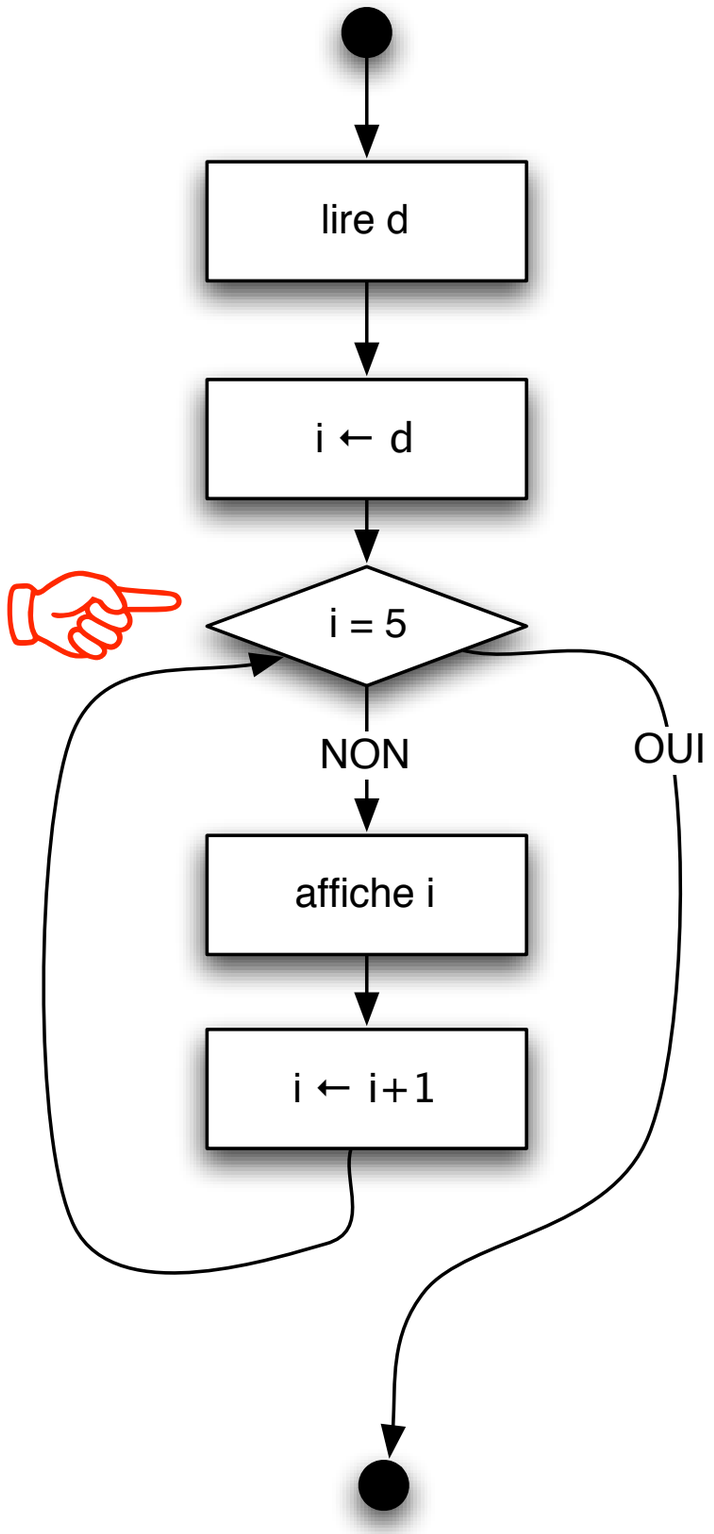


d = 7

i = 8

La valeur de i va croître continuellement et n'atteindra jamais 5 !

Exemple



d = 7

i = 8

La valeur de i va croître continuellement et n'atteindra jamais 5 !

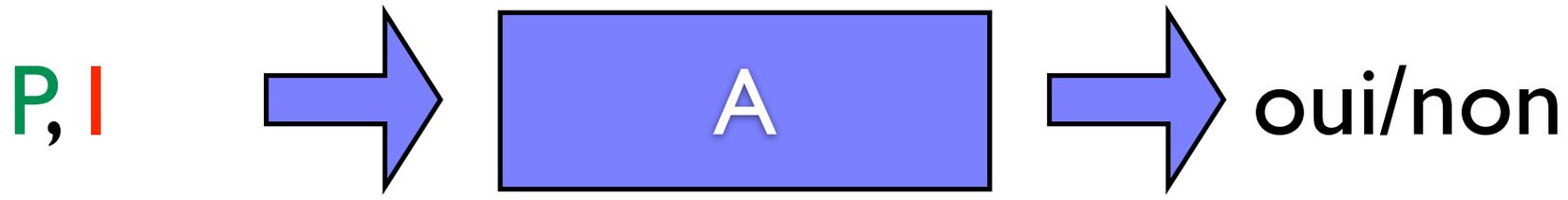
Problème de l'arrêt

- Un programme informatique reçoit des données en entrée, les traite (exécution du programme) et produit une sortie

- La comportement (exécution) du programme dépend donc des données en entrée

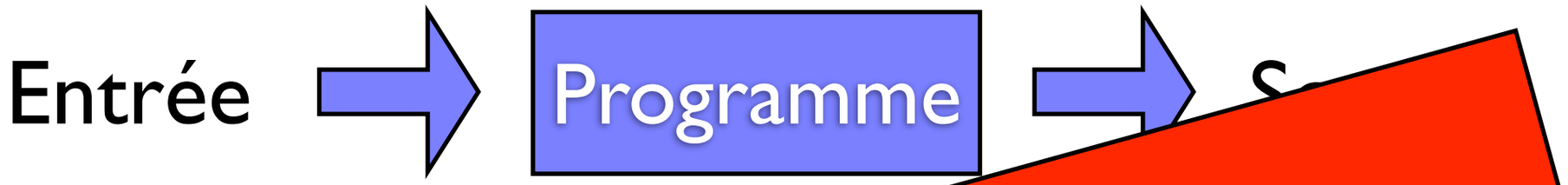
- Problème naturel: étant donné un programme P et une entrée I, déterminer si l'exécution de P sur l'entrée I produit un résultat au bout d'un temps fini

  - On aimerait donc disposer d'un programme A qui reçoit P et I en entrée et produit une sortie de type oui/non

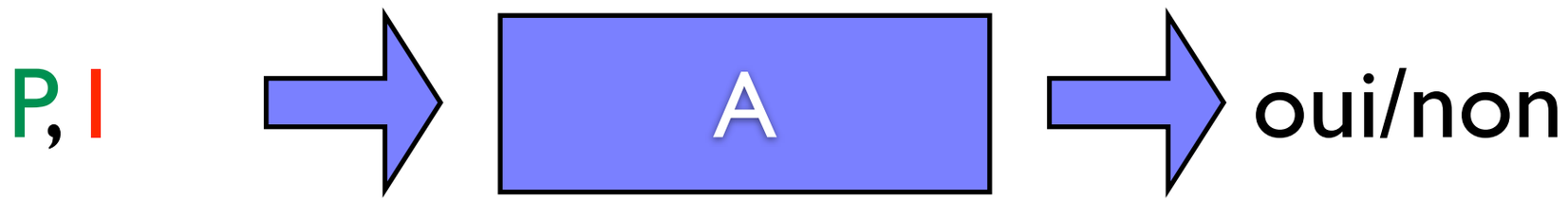


Problème de l'arrêt

- Un programme informatique reçoit des données en entrée, les traite (exécution du programme) et produit une sortie



- La comportement (exéc depend donc des donn
- tant donné un programme P et une , déterminer si l'exécution de P sur l'entrée I produit un résultat au bout d'un temps fini
  - On aimerait donc disposer d'un programme A qui reçoit P et I en entrée et produit une sortie de type oui/non



L’arrêt est indécidable

L'arrêt est indécidable

- Théorème : il n'existe pas de programme A qui, étant donné un programme quelconque P et une entrée I pour P répond oui **si et seulement si** P a une exécution finie sur I

L'arrêt est indécidable

- Théorème : il n'existe pas de programme A qui, étant donné un programme quelconque P et une entrée I pour P répond oui **si et seulement si** P a une exécution finie sur I
- Preuve : La preuve fonctionne par contradiction

L'arrêt est indécidable

- Théorème : il n'existe pas de programme A qui, étant donné un programme quelconque P et une entrée I pour P répond oui **si et seulement si** P a une exécution finie sur I
- Preuve : La preuve fonctionne par contradiction
  - On va supposer que le programme A existe

L'arrêt est indécidable

- Théorème : il n'existe pas de programme A qui, étant donné un programme quelconque P et une entrée I pour P répond oui **si et seulement si** P a une exécution finie sur I
- Preuve : La preuve fonctionne par contradiction
  - On va supposer que le programme A existe
  - Comme A existe, on peut s'en servir pour construire une autre programme

L'arrêt est indécidable

- Théorème : il n'existe pas de programme A qui, étant donné un programme quelconque P et une entrée I pour P répond oui **si et seulement si** P a une exécution finie sur I
- Preuve : La preuve fonctionne par contradiction
  - On va supposer que le programme A existe
  - Comme A existe, on peut s'en servir pour construire une autre programme
  - On va voir que ce nouveau programme aura une propriété qui est un paradoxe

L'arrêt est indécidable

- Théorème : il n'existe pas de programme A qui, étant donné un programme quelconque P et une entrée I pour P répond oui **si et seulement si** P a une exécution finie sur I
- Preuve : La preuve fonctionne par contradiction
  - On va supposer que le programme A existe
  - Comme A existe, on peut s'en servir pour construire une autre programme
  - On va voir que ce nouveau programme aura une propriété qui est un paradoxe
  - On va en déduire que la construction exposée contient une erreur qui ne peut être que l'hypothèse que A existe

L'arrêt est indécidable

- Théorème : il n'existe pas de programme A qui, étant donné un programme quelconque P et une entrée I pour P répond oui **si et seulement si** P a une exécution finie sur I
- Preuve : La preuve fonctionne par contradiction
  - On va supposer que le programme A existe
  - Comme A existe, on peut s'en servir pour construire une autre programme
  - On va voir que ce nouveau programme aura une propriété qui est un paradoxe
  - On va en déduire que la construction exposée contient une erreur qui ne peut être que l'hypothèse que A existe
  - On en déduit donc que A n'existe pas

L’arrêt est indécidable

L'arrêt est indécidable

- Hypothèse : On a à notre disposition un programme A qui reçoit deux entrées P et I et qui a la propriété suivante :

L'arrêt est indécidable

- Hypothèse : On a à notre disposition un programme A qui reçoit deux entrées P et I et qui a la propriété suivante :

Pour tout P et pour tout I : A(P,I) renvoie oui
si et seulement si
P(I) est une exécution finie

L'arrêt est indécidable

- Hypothèse : On a à notre disposition un programme A qui reçoit deux entrées P et I et qui a la propriété suivante :

Pour tout P et pour tout I : A(P,I) renvoie oui
si et seulement si
P(I) est une exécution finie

- A(P,I) dénote l'exécution de A sur P et I

L'arrêt est indécidable

- Hypothèse : On a à notre disposition un programme A qui reçoit deux entrées P et I et qui a la propriété suivante :

Pour tout P et pour tout I : A(P,I) renvoie oui
si et seulement si
P(I) est une exécution finie

- A(P,I) dénote l'exécution de A sur P et I
- P(I) dénote l'exécution de P sur I

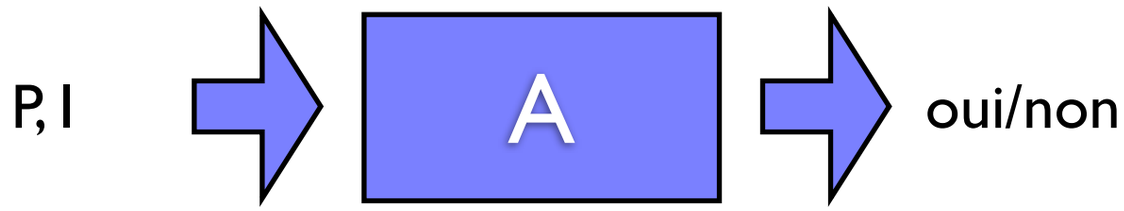
L'arrêt est indécidable

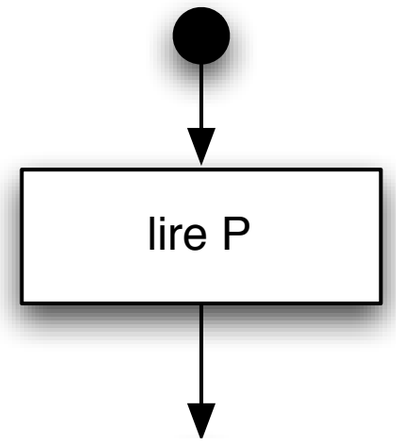
- Hypothèse : On a à notre disposition un programme A qui reçoit deux entrées P et I et qui a la propriété suivante :

Pour tout P et pour tout I : A(P,I) renvoie oui
si et seulement si
P(I) est une exécution finie

- A(P,I) dénote l'exécution de A sur P et I
- P(I) dénote l'exécution de P sur I
- On suppose que P et I sont encodés de la même manière (par exemple, en binaire, ou sous forme d'une chaîne de caractère)

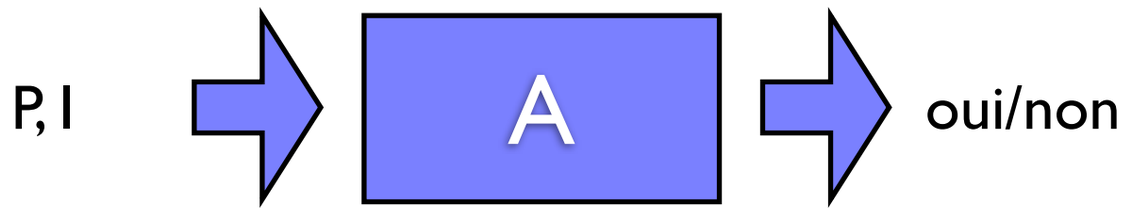
L'arrêt est indécidable

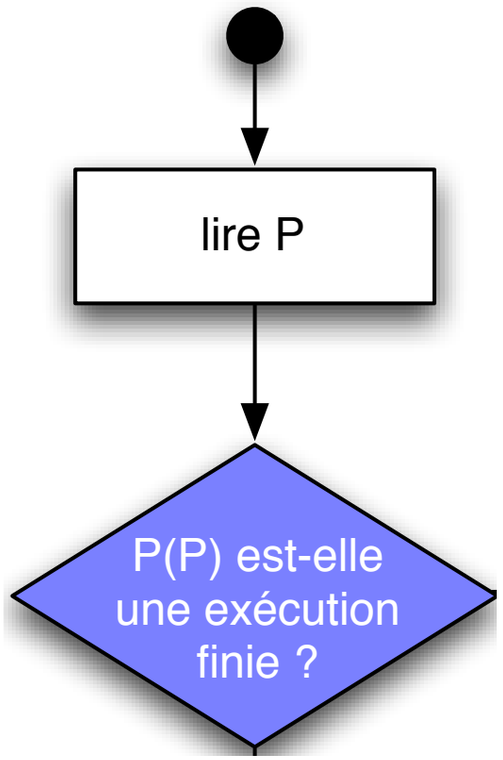




- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**

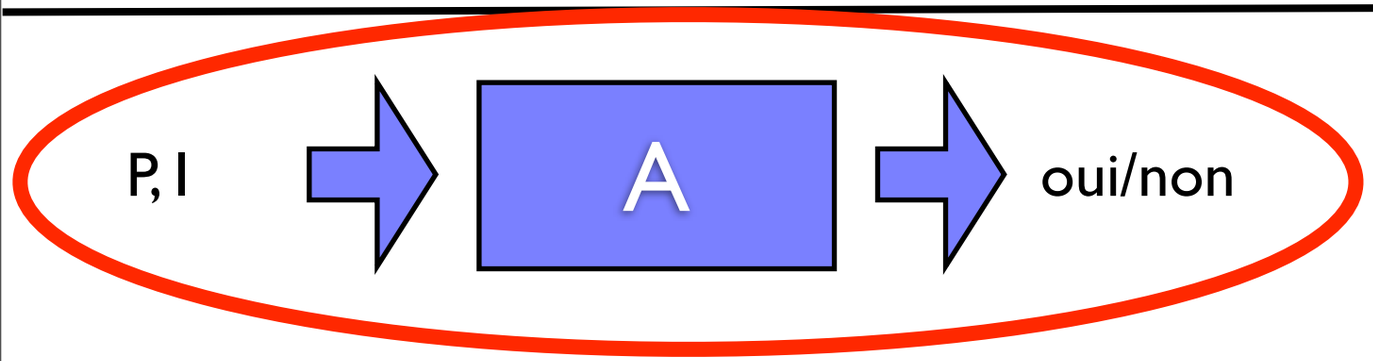
L'arrêt est indécidable

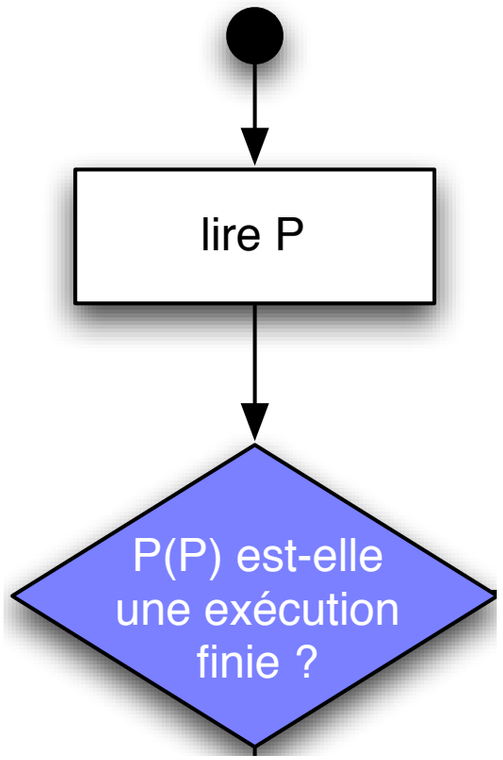




- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**

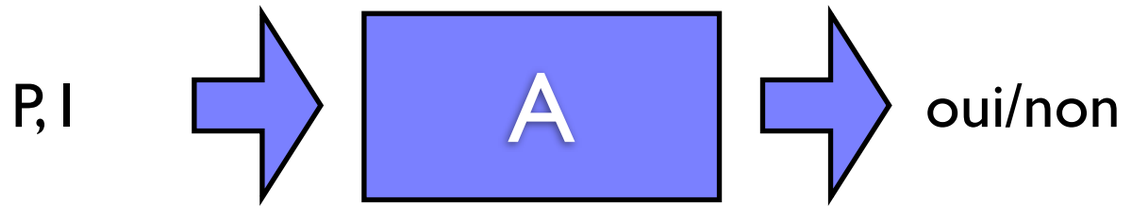
L'arrêt est indécidable



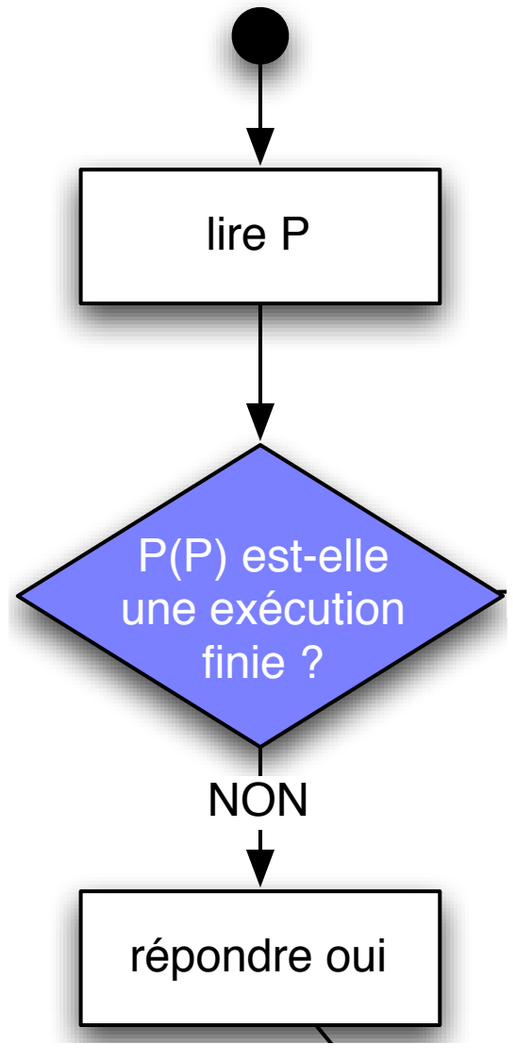


- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**

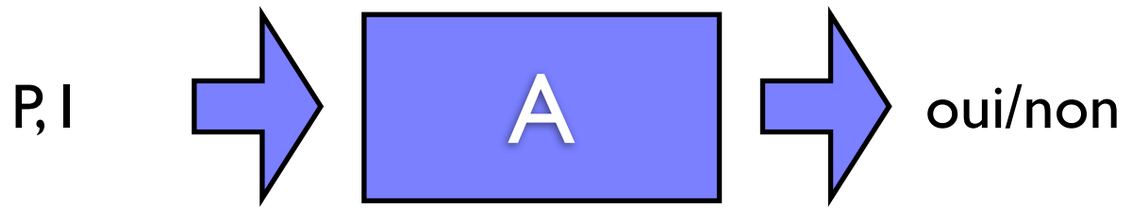
L'arrêt est indécidable



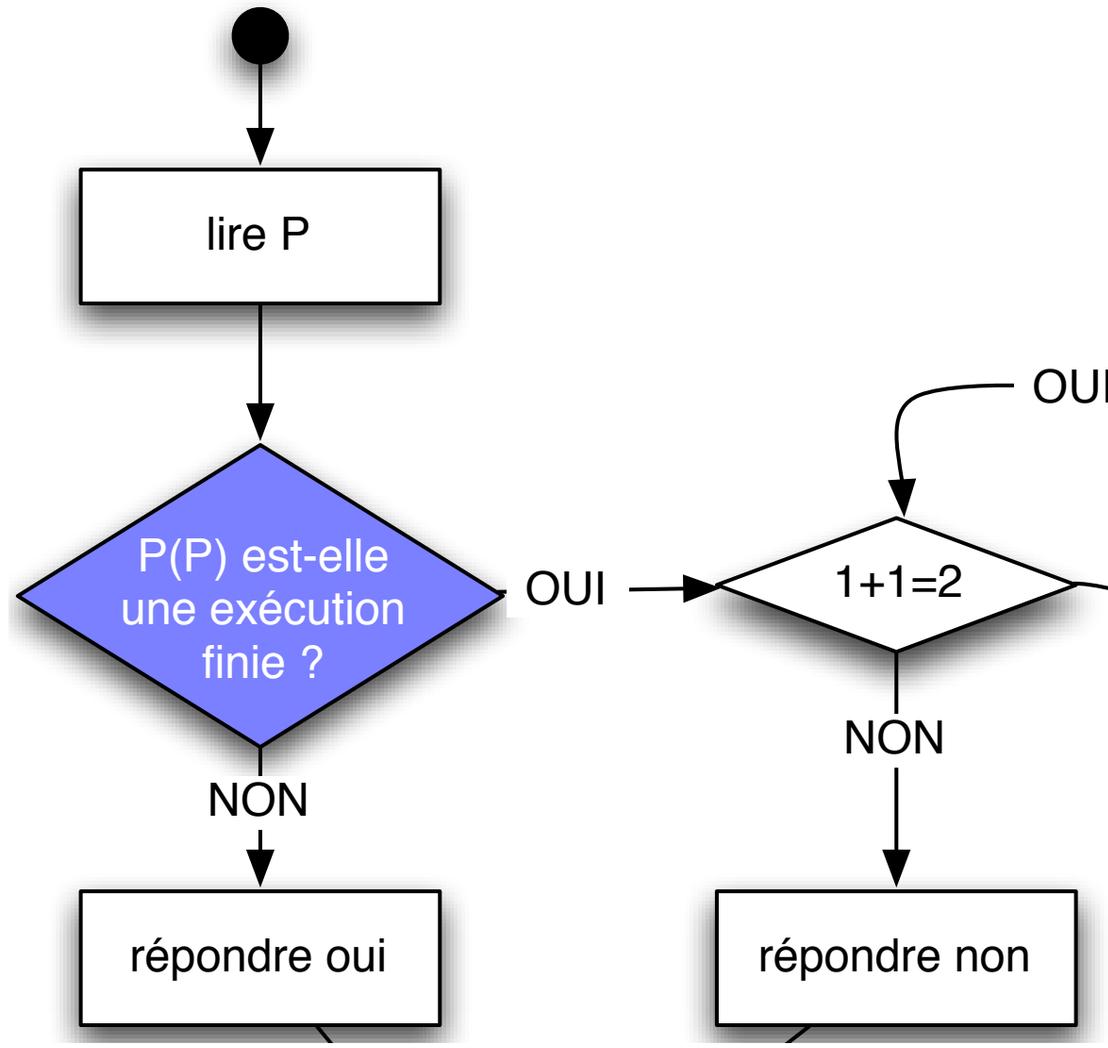
- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**



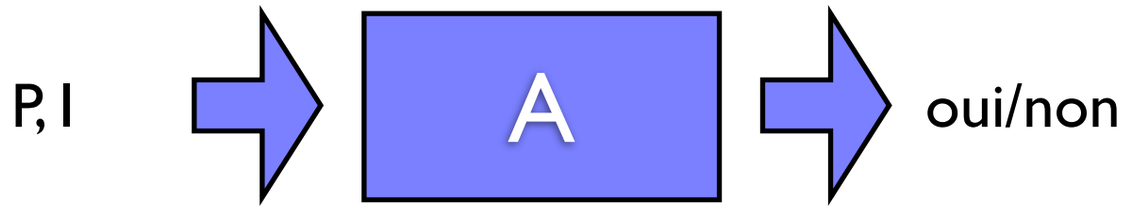
L'arrêt est indécidable



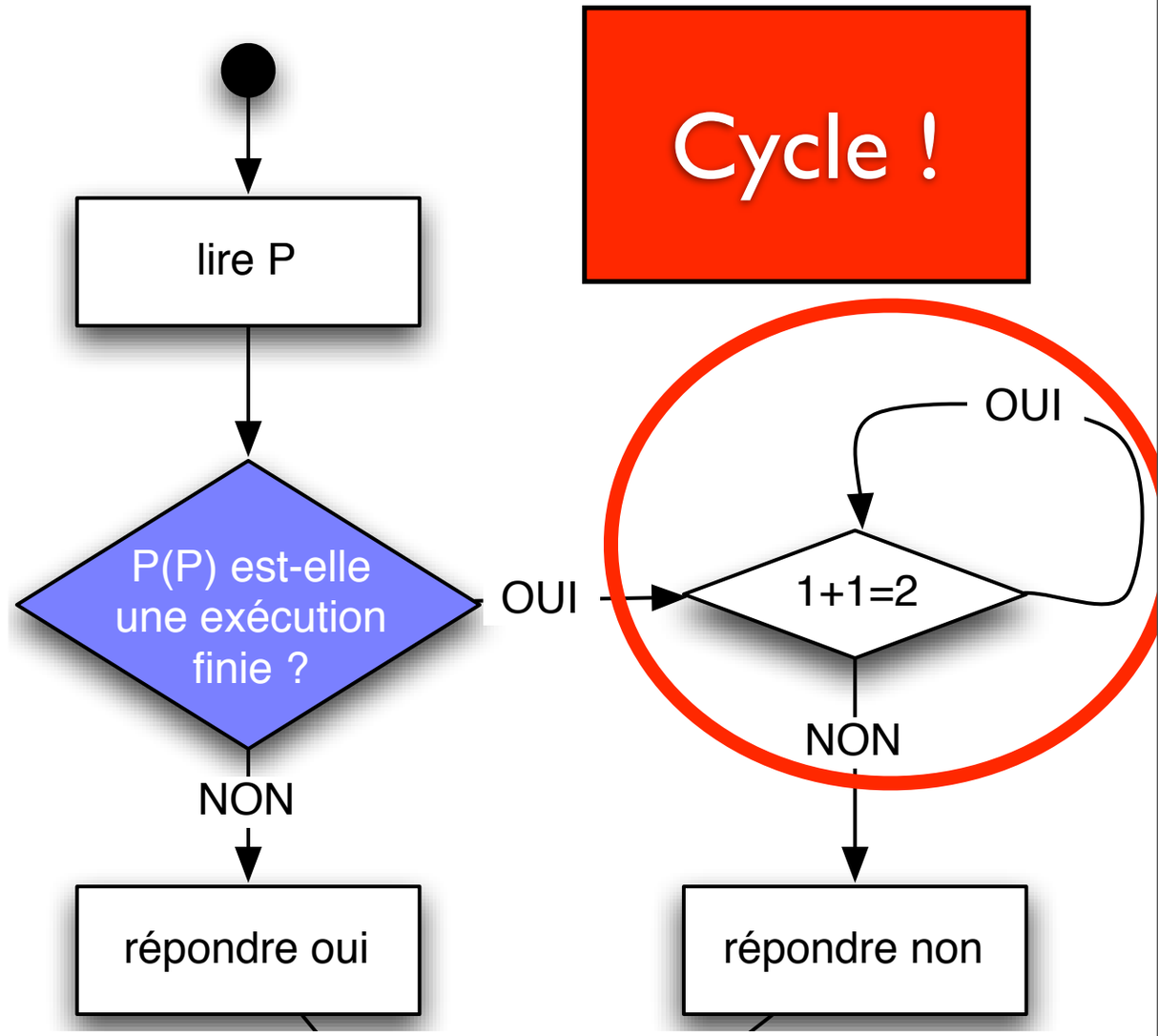
- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**



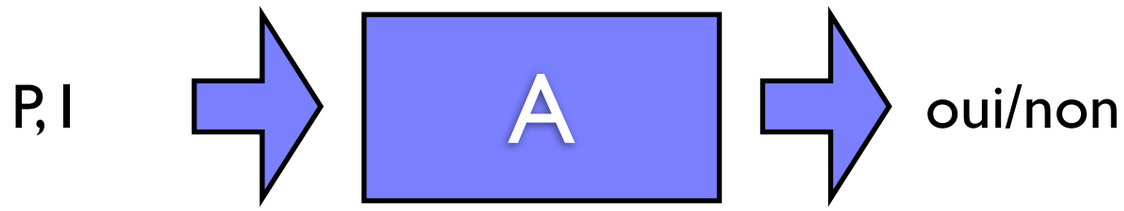
L'arrêt est indécidable



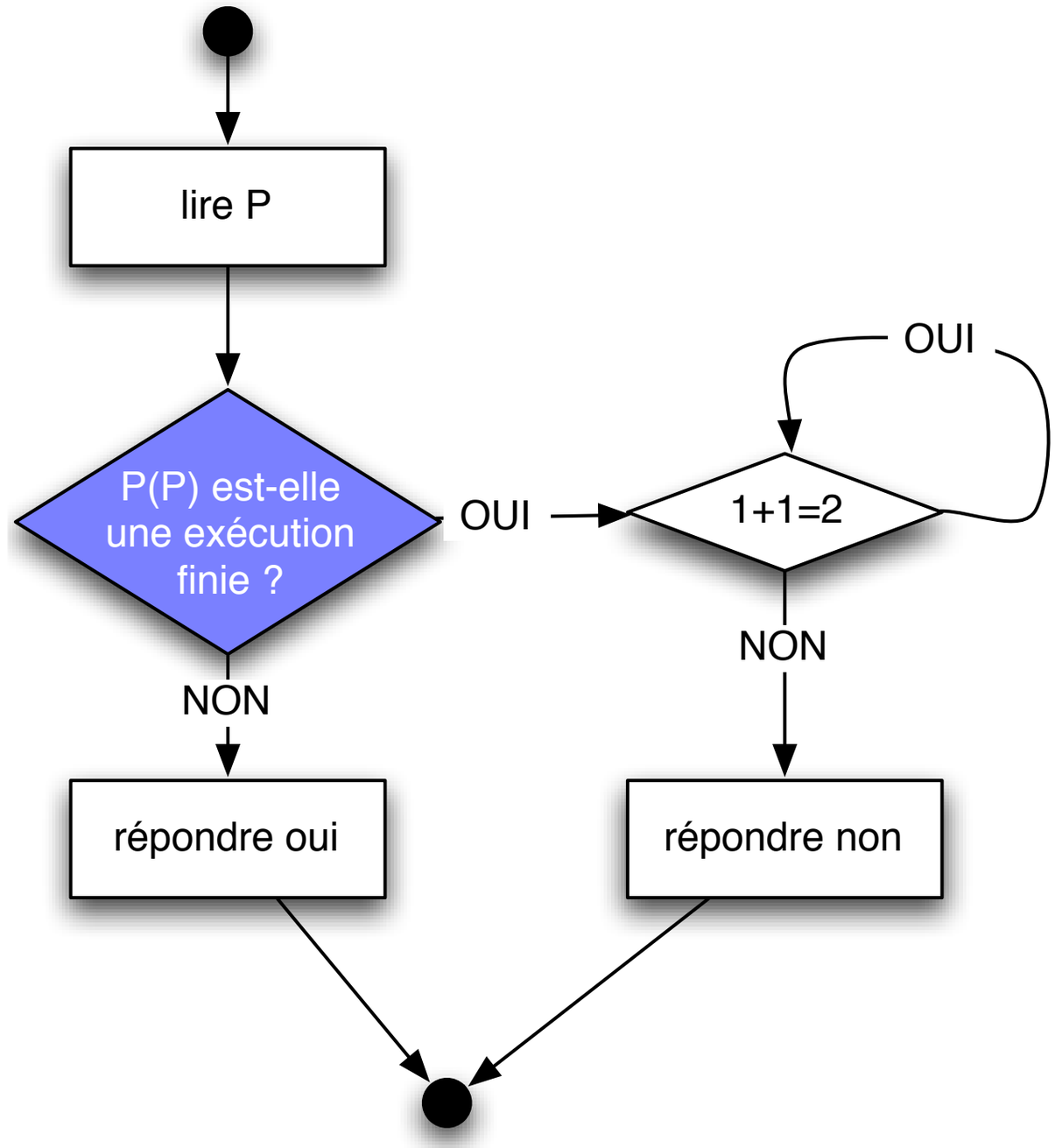
- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**



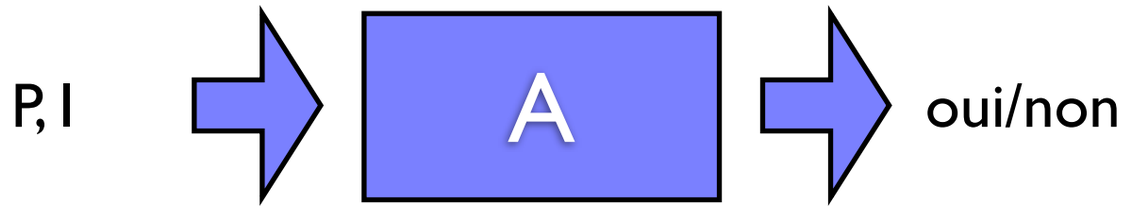
L'arrêt est indécidable



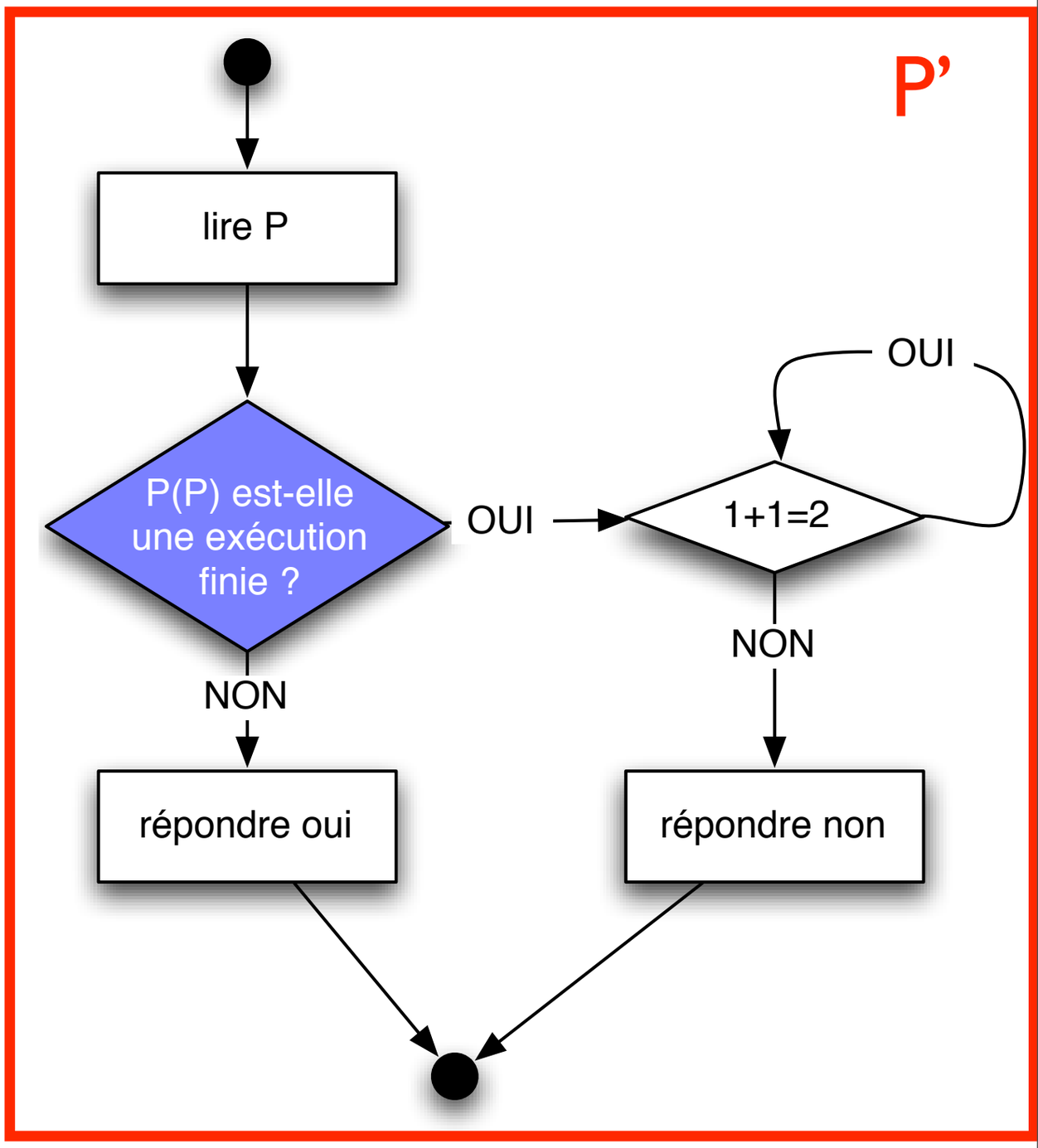
- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**



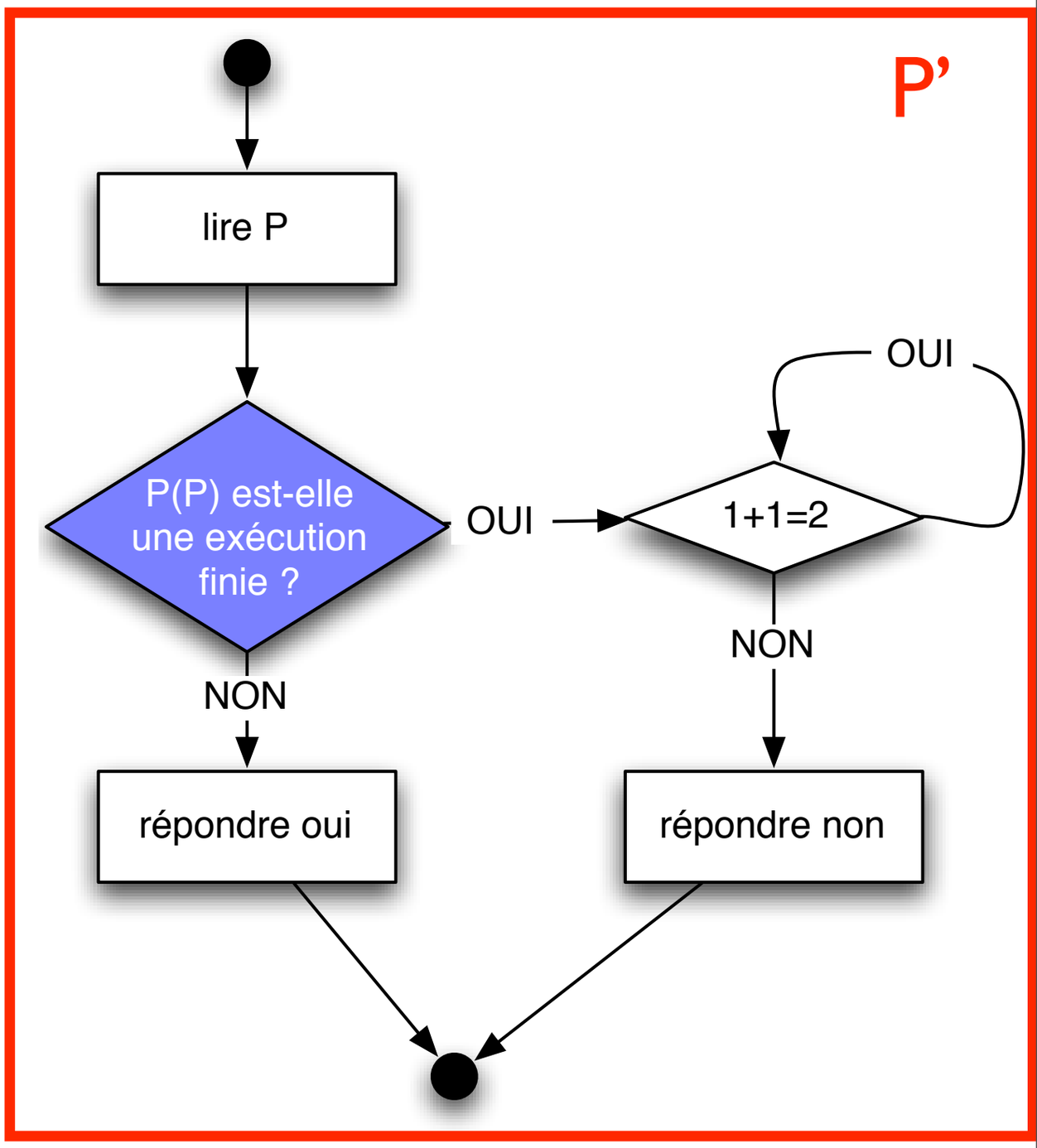
L'arrêt est indécidable



- Puisque A existe, je peux construire un nouveau programme qui reçoit en entrée un programme P et qui répond par **oui** ou **non**

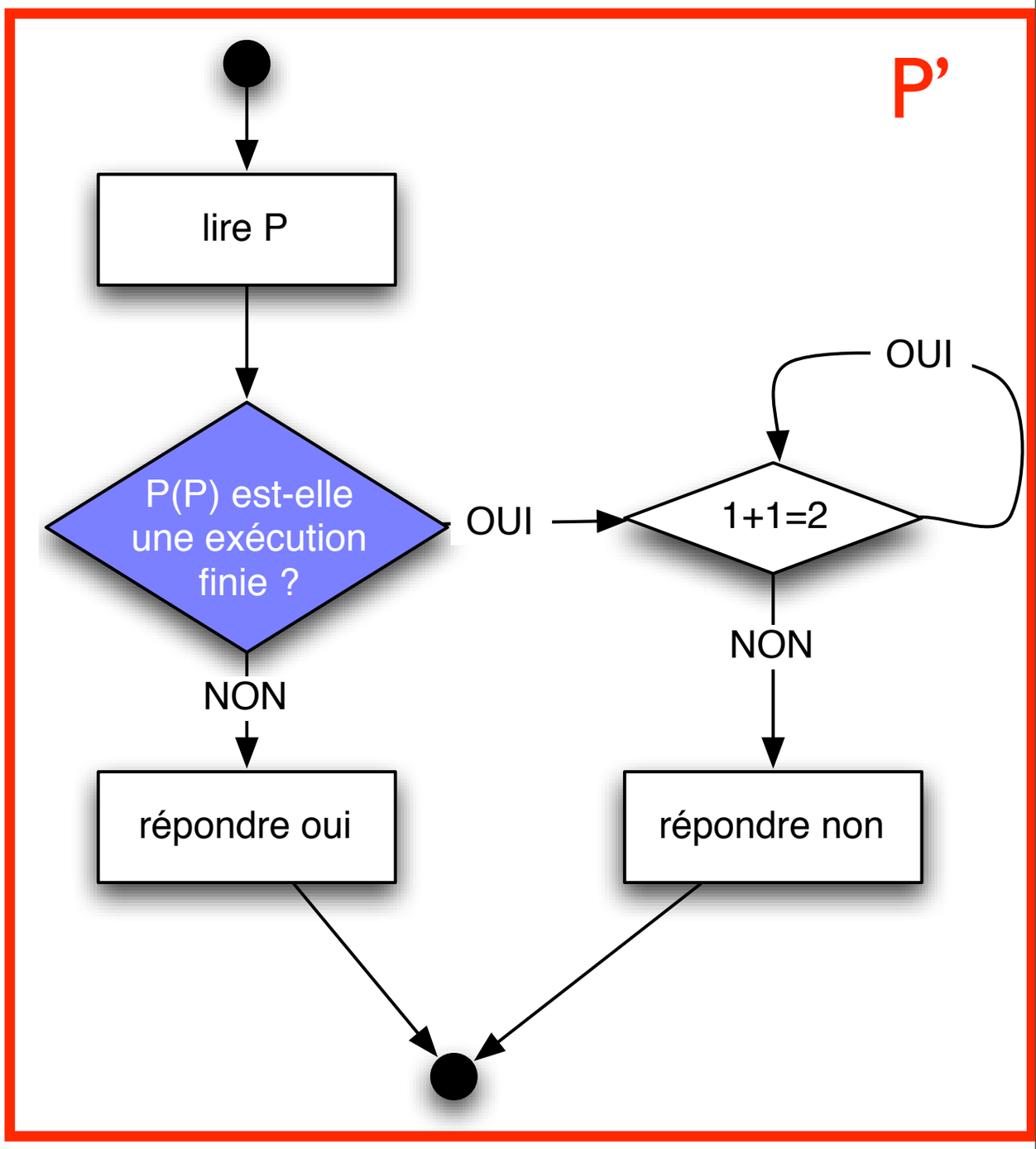


L'arrêt est indécidable



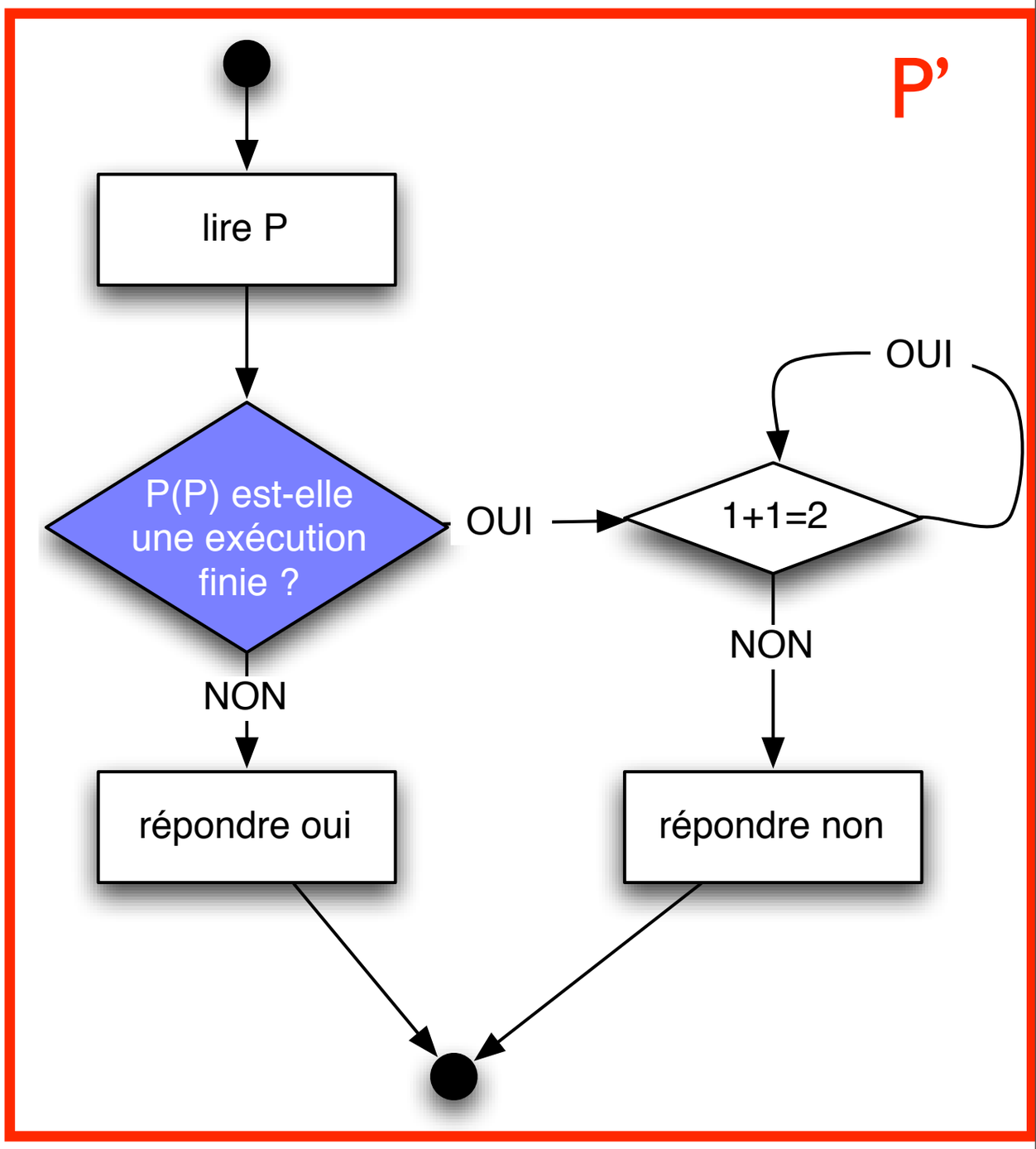
L'arrêt est indécidable

- Quelles sont les propriétés de P' ?



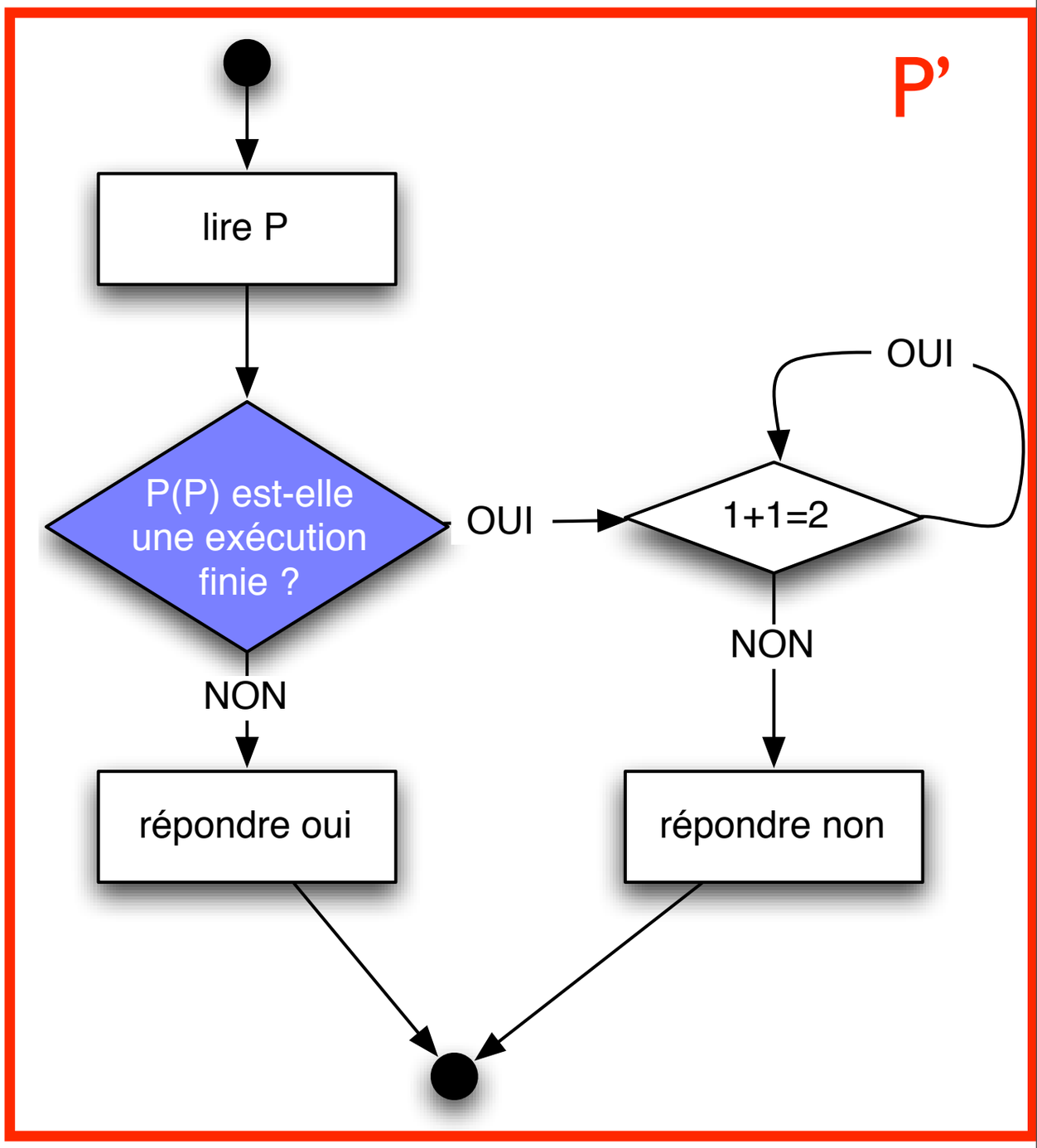
L'arrêt est indécidable

- Quelles sont les propriétés de P' ?
- Si P(P) n'est pas une exécution finie, P' répond **oui** en un temps fini
- Donc P'(P) est une exécution finie



L'arrêt est indécidable

- Quelles sont les propriétés de P' ?
- Si P(P) n'est pas une exécution finie, P' répond **oui** en un temps fini
  - Donc P'(P) est une exécution finie
- Si P(P) est une exécution finie, P' exécute une **boucle infinie**
  - Donc P'(P) n'est pas une exécution finie



L'arrêt est indécidable

- Quelles sont les

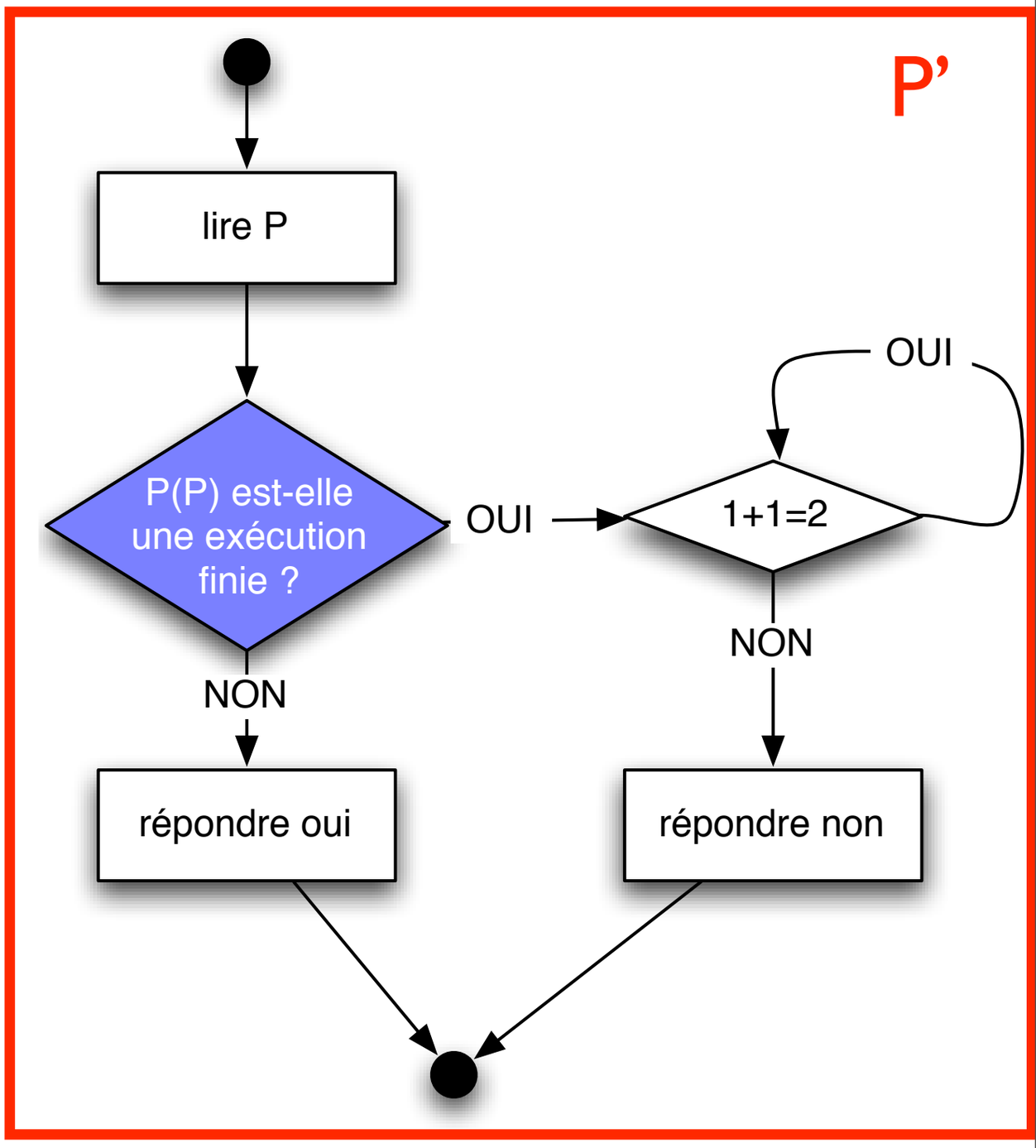
Conclusion

P'(P) est une exécution finie

si et seulement si

P(P) n'est pas une exécution finie

- Donc P' (P') n'est pas une exécution finie



L’arrêt est indécidable

L’arrêt est indécidable

- **SI** il existe un programme A qui teste l’arrêt de n’importe quel programme P sur I

L'arrêt est indécidable

- **SI** il existe un programme A qui teste l'arrêt de n'importe quel programme P sur I
- **ALORS** il existe un programme P' qui lit un programme P tel que :
  P'(P) est une exécution finie ssi P(P) n'est pas une exécution finie

L'arrêt est indécidable

- **SI** il existe un programme A qui teste l'arrêt de n'importe quel programme P sur I
- **ALORS** il existe un programme P' qui lit un programme P tel que :
  P'(P) est une exécution finie ssi P(P) n'est pas une exécution finie
- **ALORS** Comme on peut exécuter P' sur n'importe quel programme P, on peut l'exécuter en particulier sur lui-même !

L'arrêt est indécidable

- **SI** il existe un programme A qui teste l'arrêt de n'importe quel programme P sur I
- **ALORS** il existe un programme P' qui lit un programme P tel que :
  P'(P) est une exécution finie ssi P(P) n'est pas une exécution finie
- **ALORS** Comme on peut exécuter P' sur n'importe quel programme P, on peut l'exécuter en particulier sur lui-même !
  - Dans ce cas, l'exécution va-t-elle être finie ?

L'arrêt est indécidable

- **SI** il existe un programme A qui teste l'arrêt de n'importe quel programme P sur I
- **ALORS** il existe un programme P' qui lit un programme P tel que :
  P'(P) est une exécution finie ssi P(P) n'est pas une exécution finie
- **ALORS** Comme on peut exécuter P' sur n'importe quel programme P, on peut l'exécuter en particulier sur lui-même !
  - Dans ce cas, l'exécution va-t-elle être finie ?
- **ALORS** P'(P') est une exécution finie **ssi** P'(P') n'est pas une exécution finie

L'arrêt est indécidable

- **SI** il existe un programme A qui teste l'arrêt de n'importe quel programme P sur I
- **ALORS** il existe un programme P' qui lit un programme P tel que :
  P'(P) est une exécution finie ssi P(P) n'est pas une exécution finie
- **ALORS** Comme on peut exécuter P' sur n'importe quel programme P, on peut l'exécuter en particulier sur lui-même !
  - Dans ce cas, l'exécution va-t-elle être finie ?
- **ALORS** P'(P') est une exécution finie **ssi** P'(P') n'est pas une exécution finie

L'arrêt est indécidable

- **SI** il existe un programme A qui teste l'arrêt de n'importe quel programme P sur I
- **ALORS** il existe un programm
  P tel que :
  P'(P) est une exécution finie ss
  exécution finie
- **ALORS** Comme on peut exé
  programme P, on peut l'exécut
  même !
  - Dans ce cas, l'exécution va-t-elle être finie ?
- **ALORS** P'(P') est une exécution finie **ssi** P'(P') n'est pas une exécution finie

Cette conclusion est une **contradiction** qui nous oblige à **rejeter** l'hypothèse d'existence de A

L'arrêt est indécidable

- **SI** il existe un programme A qui teste l'arrêt de n'importe quel programme P sur I
- **ALORS** il existe un programme P tel que :
  P'(P) est une exécution finie ssi P(P) est une exécution finie
- **ALORS** Comme on peut exécuter le programme P, on peut l'exécuter lui-même !
  - Dans ce cas, l'exécution va-t-elle être finie ?
- **ALORS** P'(P') est une exécution finie **ssi** P'(P') n'est pas une exécution finie

Cette conclusion est une **contradiction** qui nous oblige à **rejeter** l'hypothèse d'existence de A

Autres problèmes

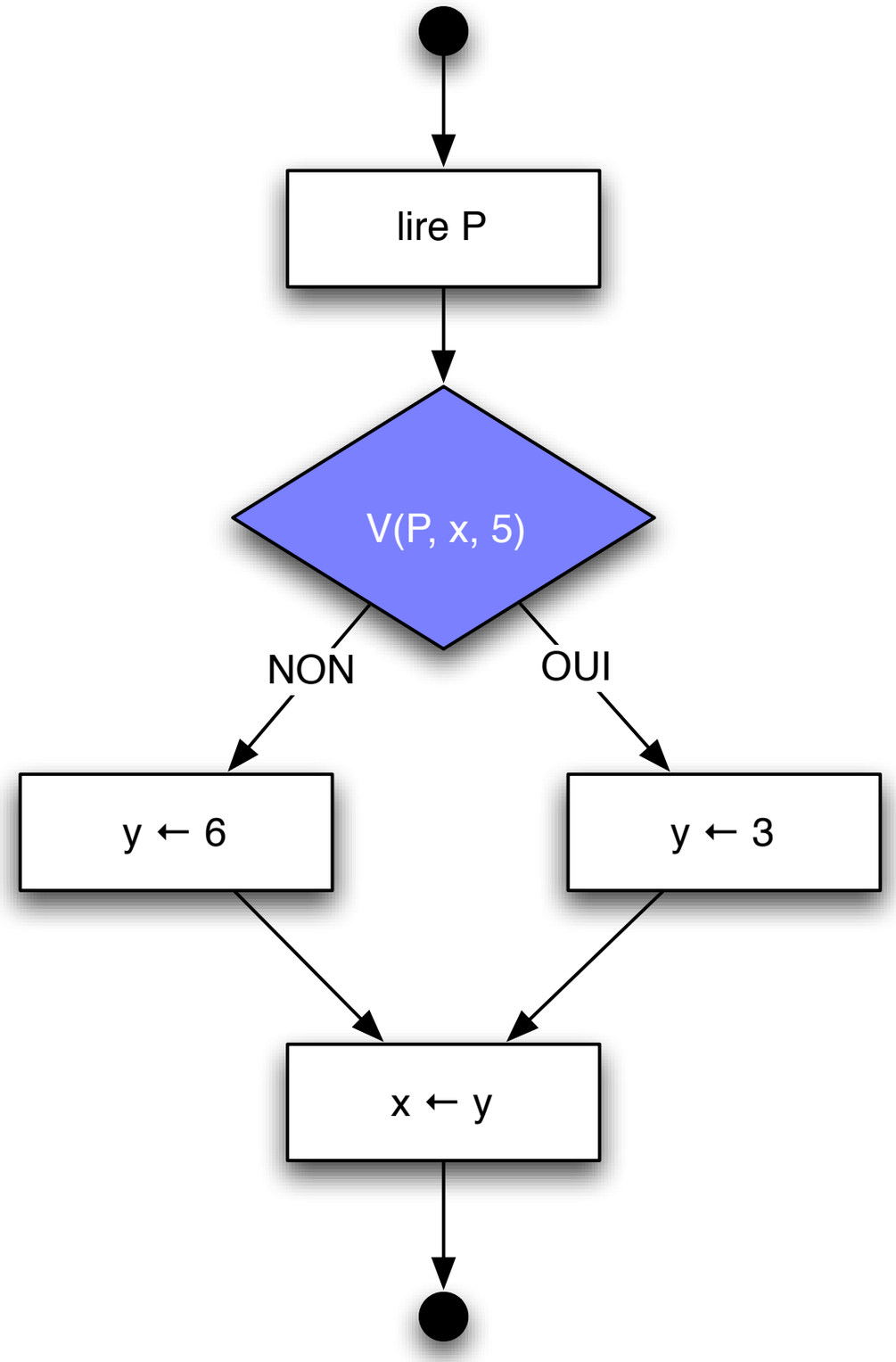
- A l'aide d'un raisonnement similaire, on peut prouver l'indécidabilité d'autres problèmes naturels et intéressants
- Exemple : existe-t-il un programme V

qui reçoit :

- un programme P
- un nom de variable x du programme P
- une valeur b

et qui répond **oui si et seulement s'il** existe une exécution du programme P dans laquelle la variable x reçoit une valeur supérieur à b

Autres problèmes

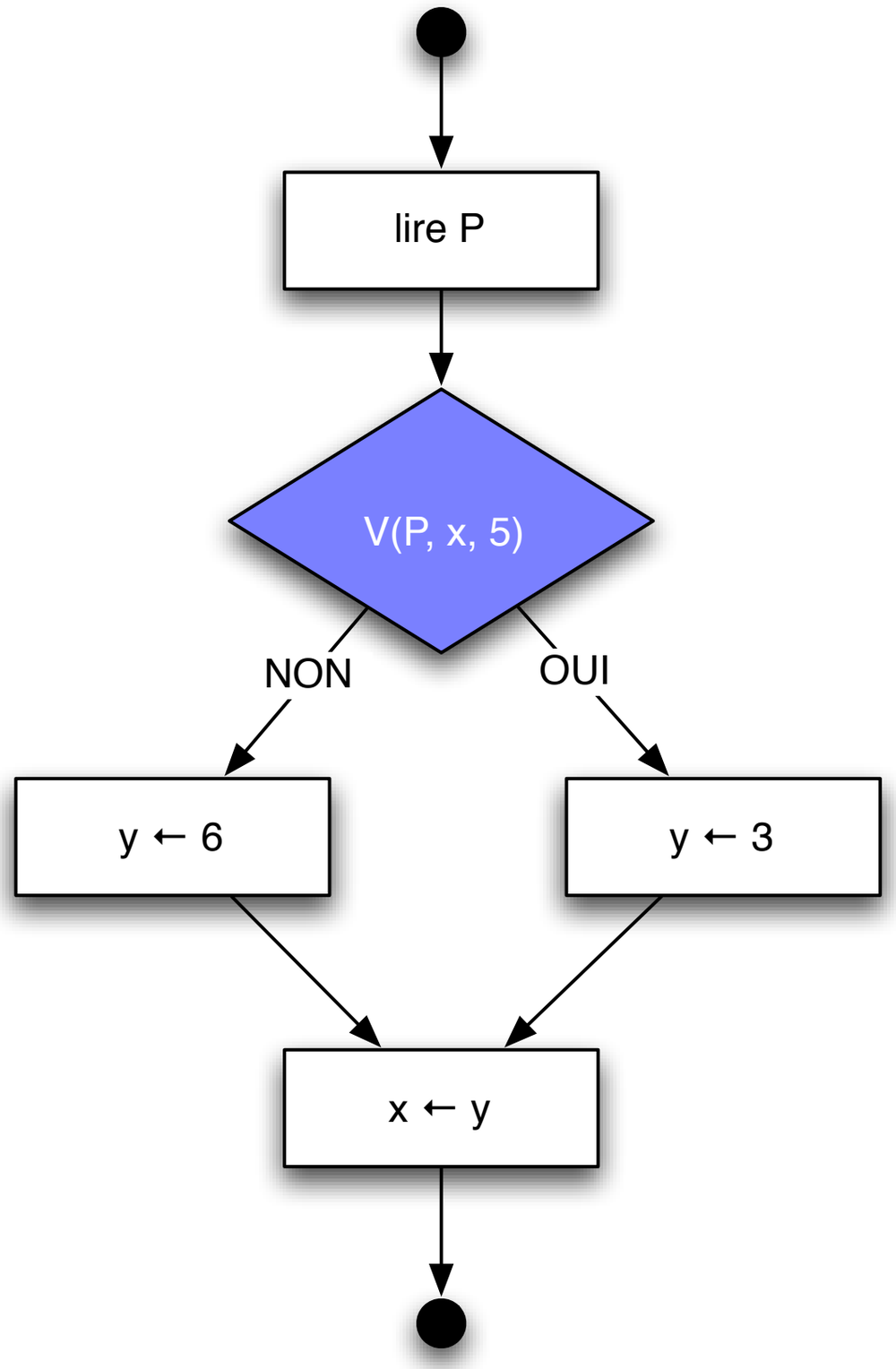


Autres problèmes

- En supposant que V existe on construit P'' ci-contre
- Ce programme P'' a la propriété suivante :

  Pour tout programme P :
  x≥5 dans l'exécution P''(P)
  ssi
  x≤5 dans toute exécution de P
- En appelant P'' sur lui-même on a :

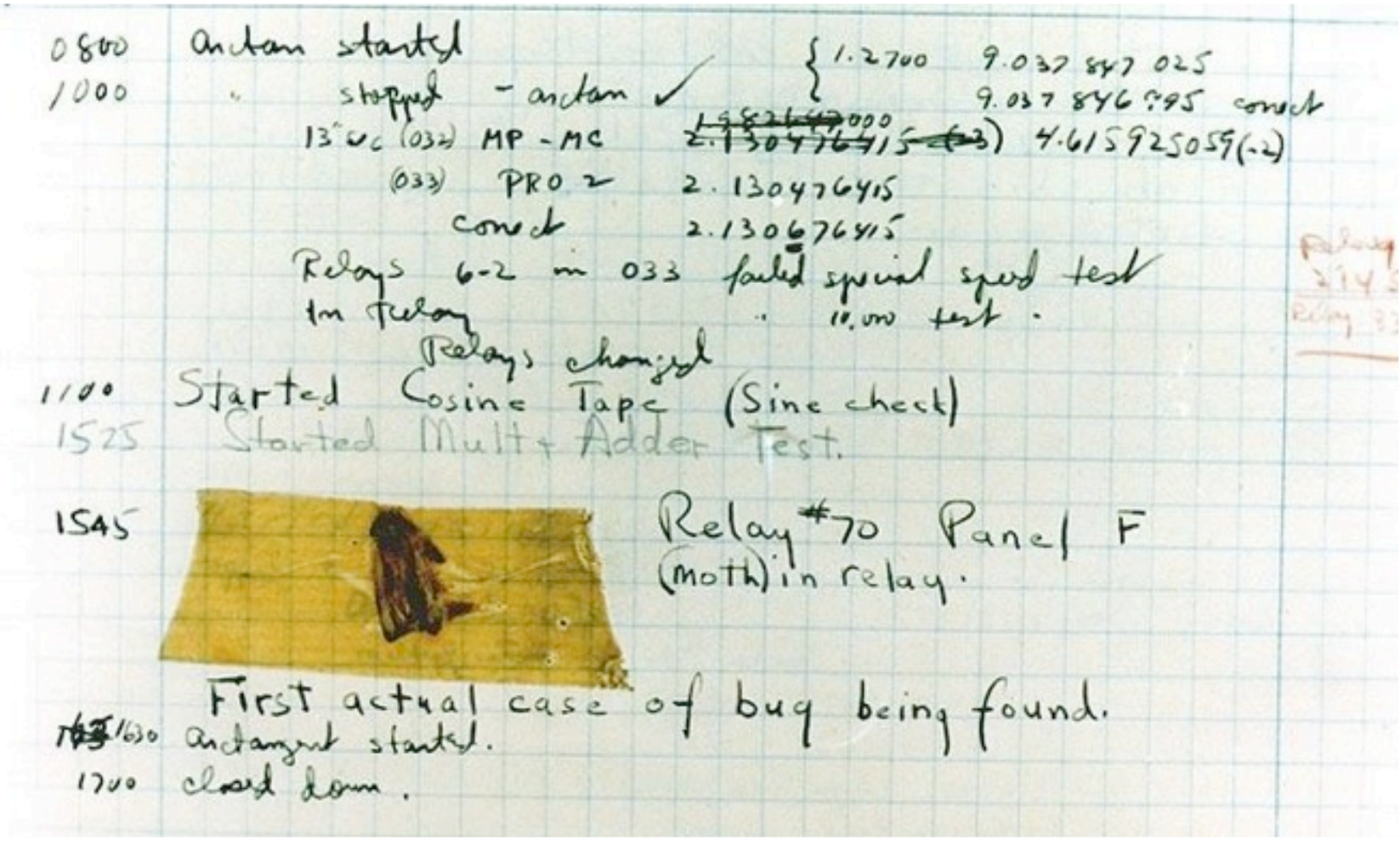
  x≥5 dans l'exécution P''(P'')
  ssi
  x≤5 dans toute exécution de P''



Vérification assistée par ordinateur

Motivation

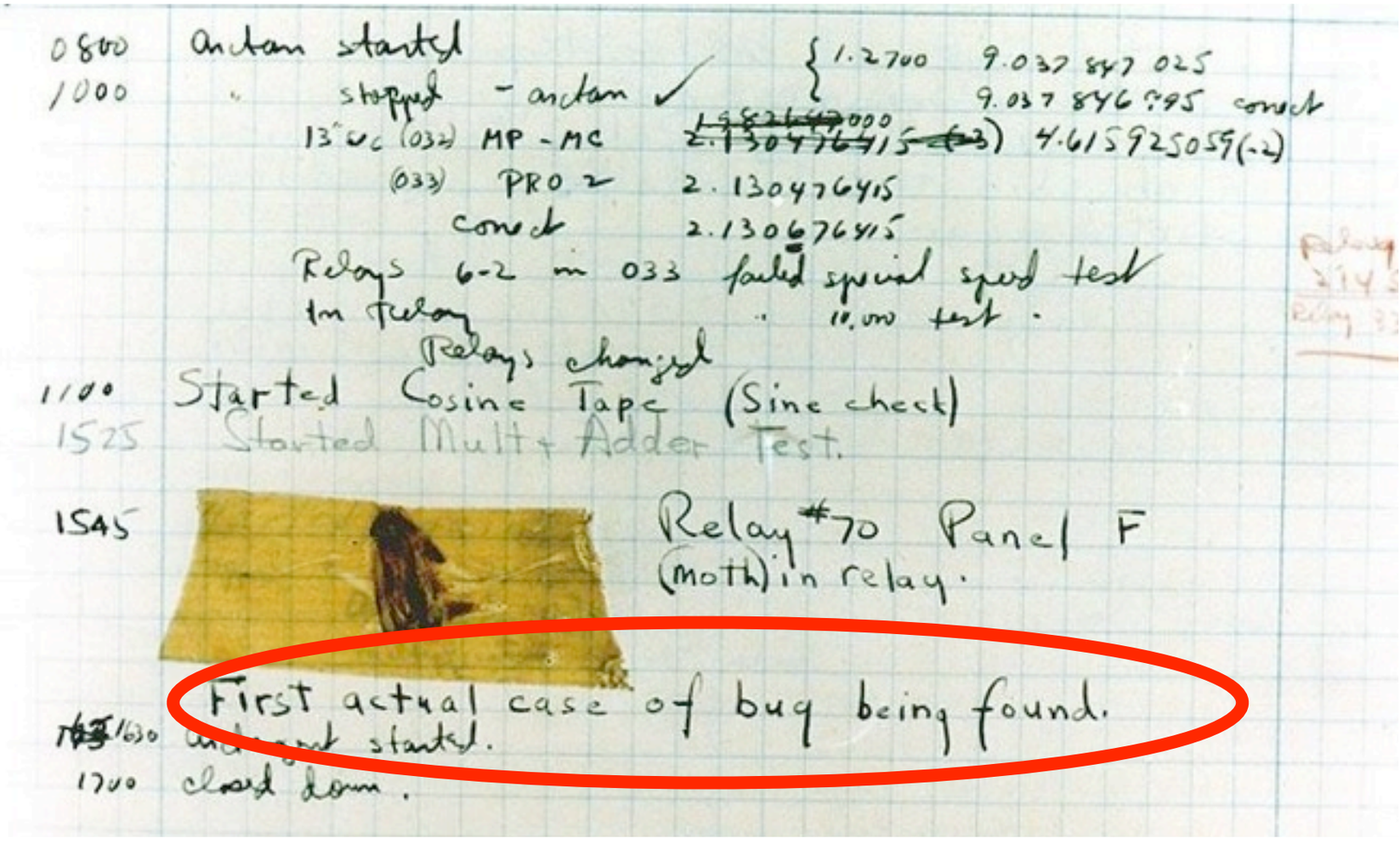
- L'histoire veut que le premier *bug* informatique a été causé par une mite qui a créé un faux contact dans le Mark II (Université de Harvard) en 1945



Extrait du journal de bord de l'ordinateur

Motivation

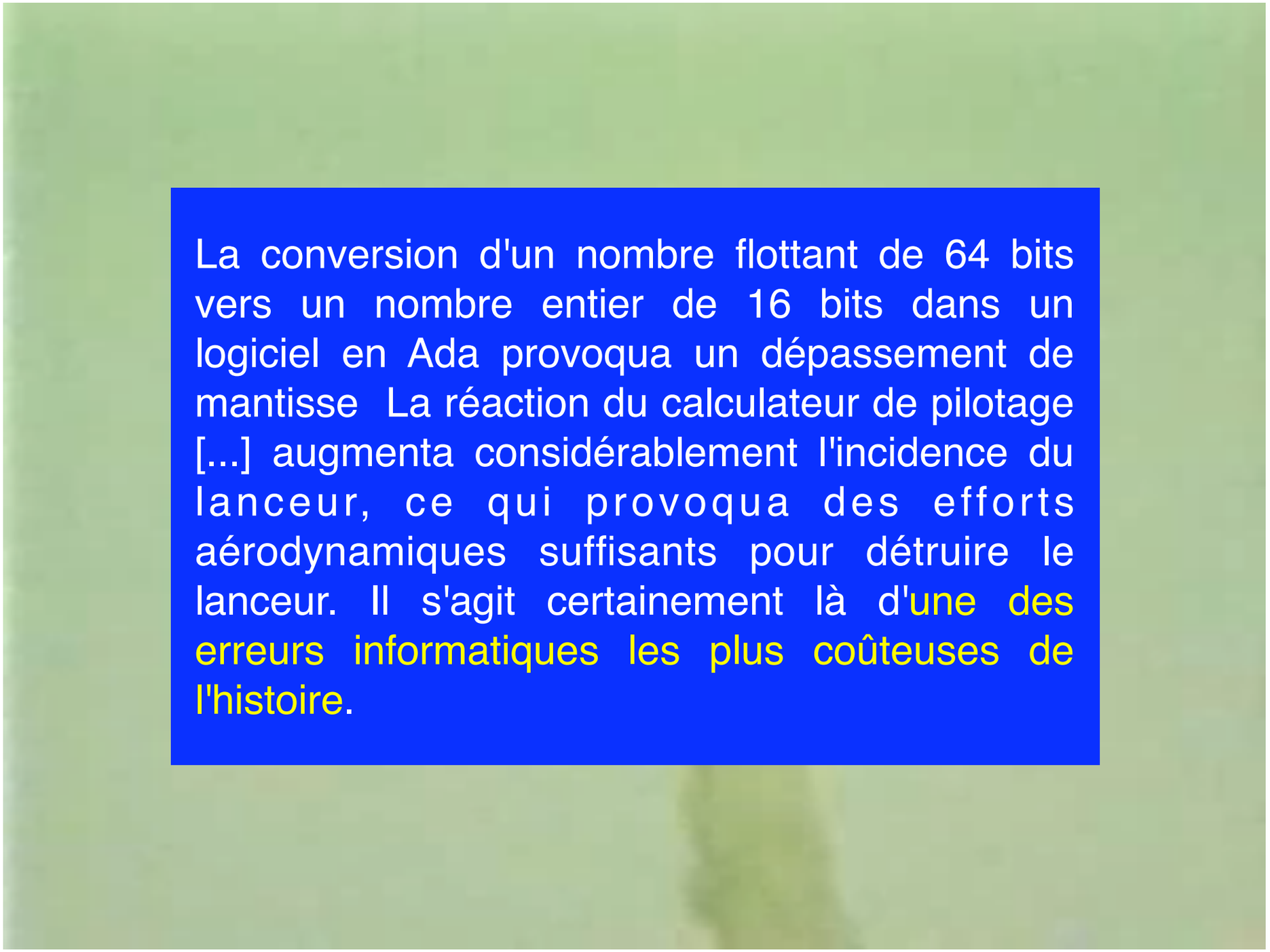
- L'histoire veut que le premier *bug* informatique a été causé par une mite qui a créé un faux contact dans le Mark II (Université de Harvard) en 1945



Extrait du journal de bord de l'ordinateur

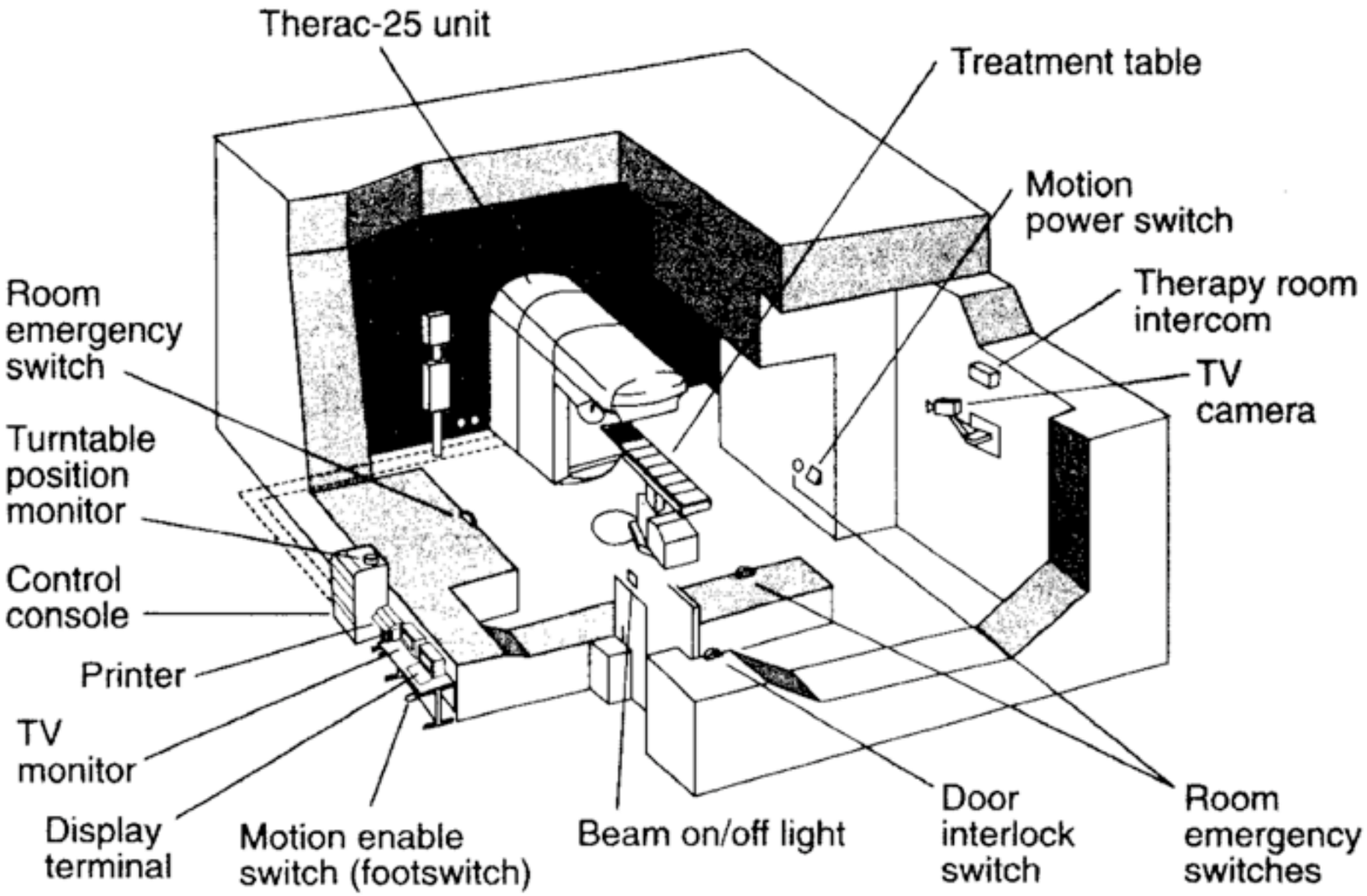
Motivation

Motivation


La conversion d'un nombre flottant de 64 bits vers un nombre entier de 16 bits dans un logiciel en Ada provoqua un dépassement de mantisse La réaction du calculateur de pilotage [...] augmenta considérablement l'incidence du lanceur, ce qui provoqua des efforts aérodynamiques suffisants pour détruire le lanceur. Il s'agit certainement là d'une des erreurs informatiques les plus coûteuses de l'histoire.

Motivation

- Entre 1985 et 1987 au moins 5 patients sont morts de dose trop massive de radiations émise par un appareil de traitement du cancer (le Therac-25) dont le logiciel de contrôle était défectueux.



Windows

A fatal exception 0E has occurred at 0137:BFFA21C9. The current application will be terminated.

* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue _

Motivation

Motivation

- Les ordinateurs sont souvent «enfouis» dans des systèmes plus larges et critiques

Motivation

- Les ordinateurs sont souvent «enfouis» dans des systèmes plus larges et critiques
- Toute déficience du système informatique peut avoir des conséquences catastrophiques

Motivation

- Les ordinateurs sont souvent «enfouis» dans des systèmes plus larges et critiques
- Toute déficience du système informatique peut avoir des conséquences catastrophiques
- Il y a donc un véritable besoin de méthodes permettant de prouver qu'un système informatique ou un programme est correct

Motivation

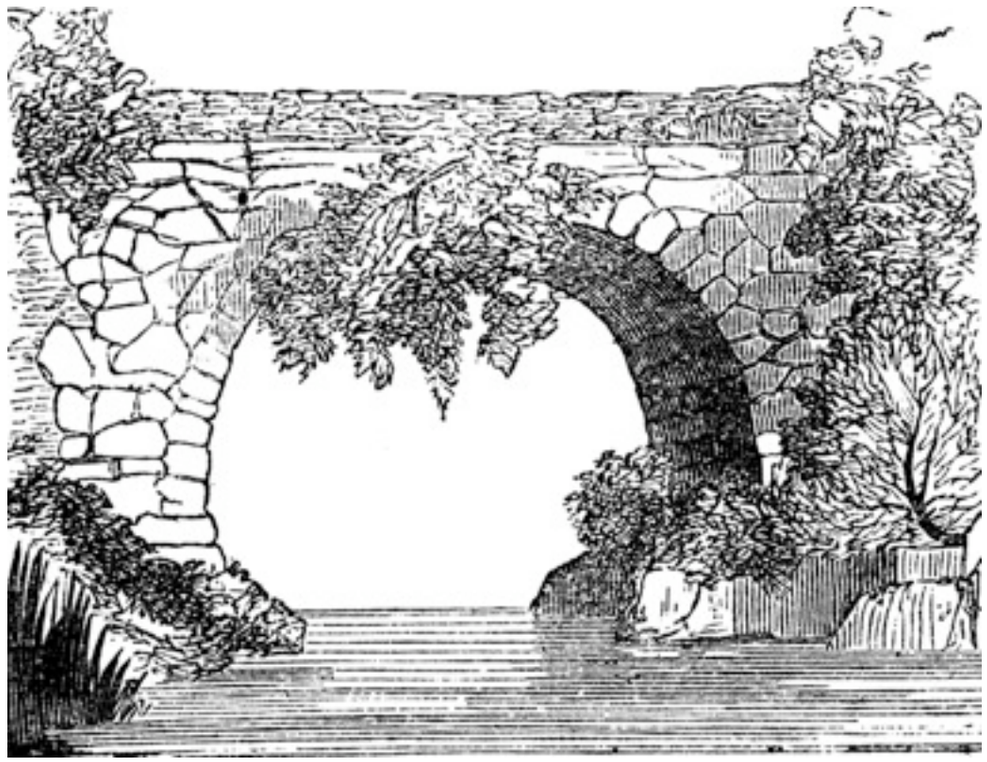
- Les ordinateurs sont souvent «enfouis» dans des systèmes plus larges et critiques
- Toute déficience du système informatique peut avoir des conséquences catastrophiques
- Il y a donc un véritable besoin de méthodes permettant de prouver qu'un système informatique ou un programme est correct
- Ces méthodes doivent être automatiques, car les programmes contemporains sont trop complexes pour être analysés à la main
  - Exemple: la suite bureautique OpenOffice est constituée de 9'000'000 de lignes de code. A raison de 60 lignes par page A4, le listing complet aurait une longueur de 50 Km (la distance Bruxelles-Gand)

Démarche d'abstraction

Abstraction

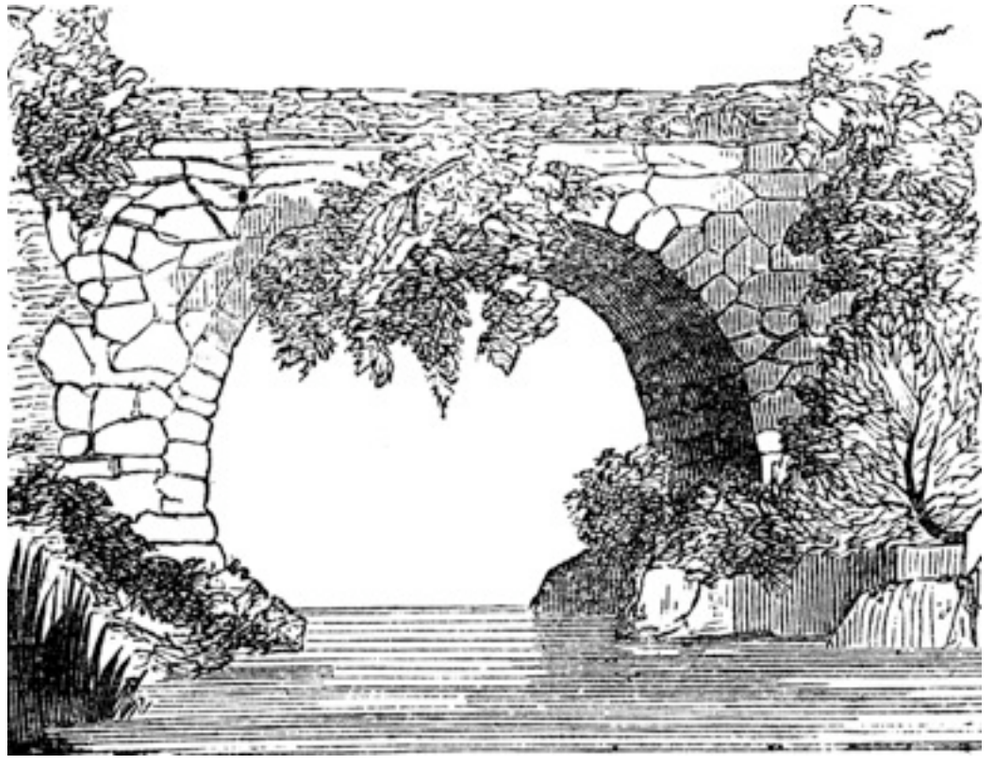
- De manière générale, prouver la correction d'un programme est indécidable
- Les programmes informatiques (ou les machines de Turing) sont des objets «trop compliqués»
- Peut-on considérer des modèles mathématiques plus simples, plus abstraits ?

Abstraction



Abstraction

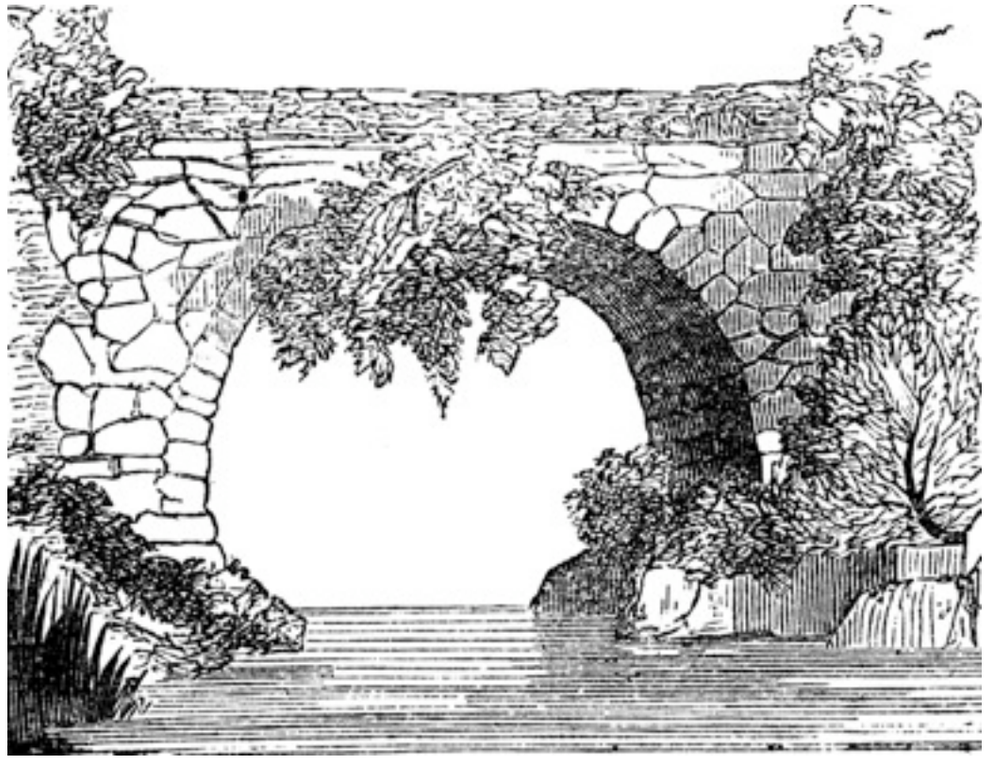
- Illustration : On désire construire un pont et garantir qu'il résiste à une certaine charge



Abstraction

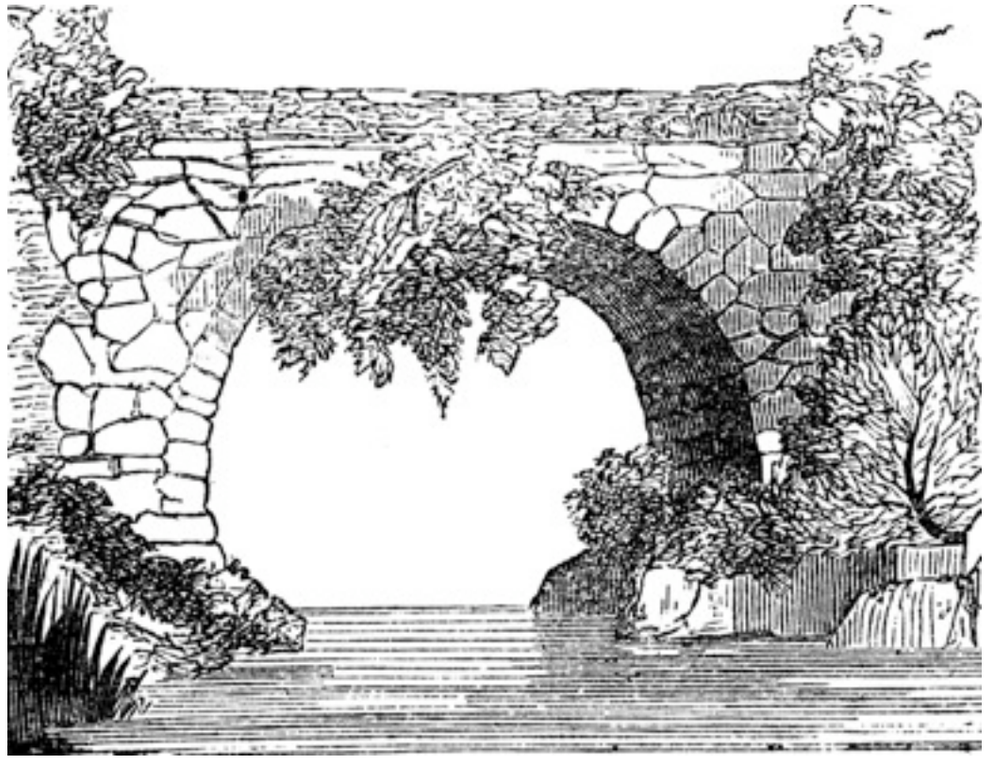
- Illustration : On désire construire un pont et garantir qu'il résiste à une certaine charge

- Solution 1 : construire le pont et essayer...



Abstraction

- Illustration : On désire construire un pont et garantir qu'il résiste à une certaine charge



  - Solution 1 : construire le pont et essayer...
  - Solution 2 : utiliser des modèles mathématiques pour prévoir la résistance des matériaux

Abstraction

Abstraction

Réalité

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Modèle

$$\frac{\partial F}{\partial t} = - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \quad (1)$$

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Modèle

$$\frac{\partial F}{\partial t} = - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \quad (1)$$

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Modèle

$$\frac{\partial F}{\partial t} = - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \quad (1)$$

Question sur le modèle

L'équation a-t-elle
une solution telle
que... ??

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Modèle

$$\begin{aligned}\frac{\partial F}{\partial t} = & - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) \\ & - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) \\ & - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) \\ & + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \qquad (1)\end{aligned}$$

Méthode de résolution

Question sur le modèle

L'équation a-t-elle
une solution telle
que... ??

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Modèle

$$\begin{aligned}\frac{\partial F}{\partial t} = & - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) \\ & - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) \\ & - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) \\ & + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \qquad (1)\end{aligned}$$

Méthode de résolution

Question sur le modèle

L'équation a-t-elle
une solution telle
que... ??

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Modèle

$$\begin{aligned}\frac{\partial F}{\partial t} = & - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) \\ & - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) \\ & - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) \\ & + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \qquad (1)\end{aligned}$$

Méthode de résolution

Réponse

Question sur le modèle

L'équation a-t-elle
une solution telle
que... ??

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Réponse

Modèle

$$\begin{aligned}\frac{\partial F}{\partial t} = & - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) \\ & - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) \\ & - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) \\ & + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \qquad (1)\end{aligned}$$

Méthode de résolution

Question sur le modèle

L'équation a-t-elle
une solution telle
que... ??

Abstraction

Réalité

Question

Le pont résistera-t-il
a une charge
de 20 tonnes ?

Réponse

Modèle

$$\frac{\partial F}{\partial t} = - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \quad (1)$$

Réponse

Méthode de résolution

Question sur le modèle

L'équation a-t-elle
une solution telle
que... ??

Abstraction

Réalité

Abstraction

Modèle

$$\begin{aligned}\frac{\partial F}{\partial t} = & - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) \\ & - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) \\ & - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) \\ & + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \qquad (1)\end{aligned}$$

Réponse

Méthode de résolution

Question

Le pont résistera-t-il a une charge de 20 tonnes ?

Abstraction

Question sur le modèle

L'équation a-t-elle une solution telle que... ??

Réponse

Abstraction

Réalité

Abstraction

Modèle

$$\begin{aligned}\frac{\partial F}{\partial t} = & - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) \\ & - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) \\ & - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) \\ & + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \qquad (1)\end{aligned}$$

Méthode de résolution

Réponse

Prévision

Question

Le pont résistera-t-il a une charge de 20 tonnes ?

Abstraction

Question sur le modèle

L'équation a-t-elle une solution telle que... ??

Réponse

Abstraction

Réalité

Abstraction

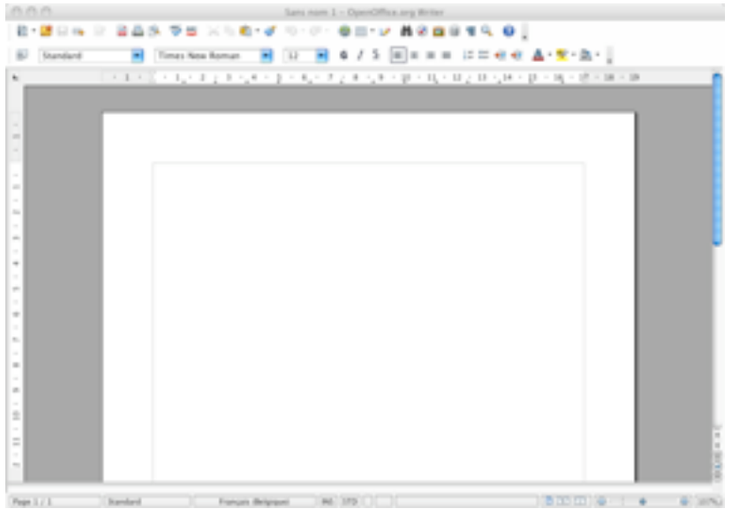
Modèle

$$\begin{aligned}\frac{\partial F}{\partial t} = & - \frac{\partial}{\partial R}\left(F\left\{u_R + \frac{\partial D_{RR}}{\partial R} + \frac{D_{RR}}{R}\right\}\right) \\ & - \frac{\partial}{\partial Z}\left(F\left\{u_Z + \frac{\partial D_{ZZ}}{\partial Z}\right\}\right) \\ & - \frac{\partial}{\partial p}\left(F\left\{-\frac{p}{3}\nabla\cdot\mathbf{u} + \frac{1}{p^2}\frac{\partial p^2 D_{pp}}{\partial p} - \sum_k \frac{p}{t_{loss,k}(p)}\right\}\right) \\ & + \frac{\partial^2}{\partial R^2}(FD_{RR}) + \frac{\partial^2}{\partial Z^2}(FD_{ZZ}) + \frac{\partial^2}{\partial p^2}(FD_{pp}) \qquad (1)\end{aligned}$$

- Le modèle mathématique retient les caractéristiques essentielles du pont
  - Ce sont les caractéristiques qui nous permettent de répondre à la question posée
- On va suivre la même démarche pour vérifier des programmes informatiques

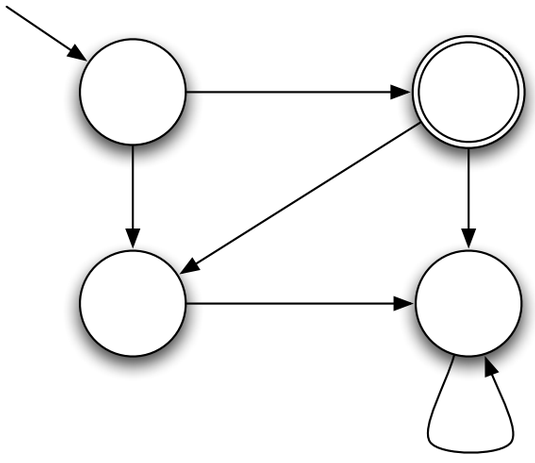
Abstraction

Réalité



Abstraction

Modèle



Question

La variable x prendra-t-elle une valeur ≥ 5 ?

Abstraction

Question sur le modèle

... ?

Réponse

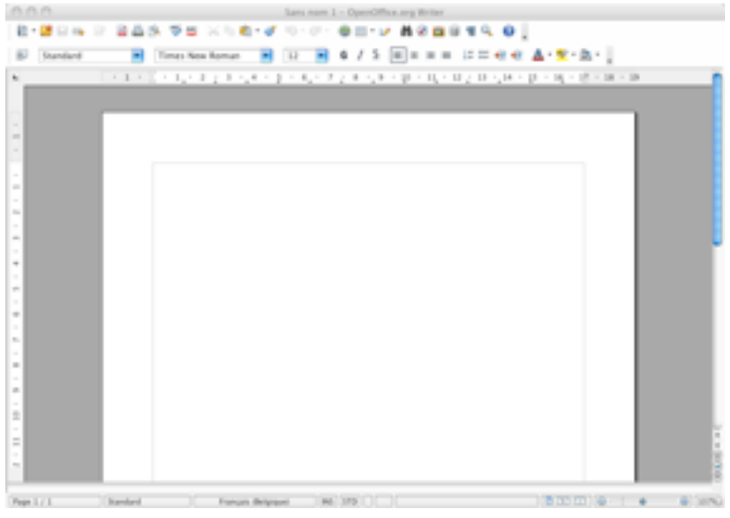
Prévision

Réponse

Méthode de vérification

Abstraction

Réalité



Question

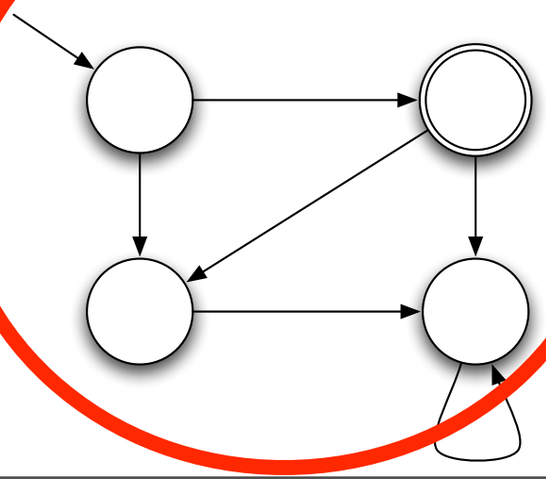
La variable x prendra-t-elle une valeur ≥ 5 ?

Réponse

Réponse

Quels modèles ?

Modèle



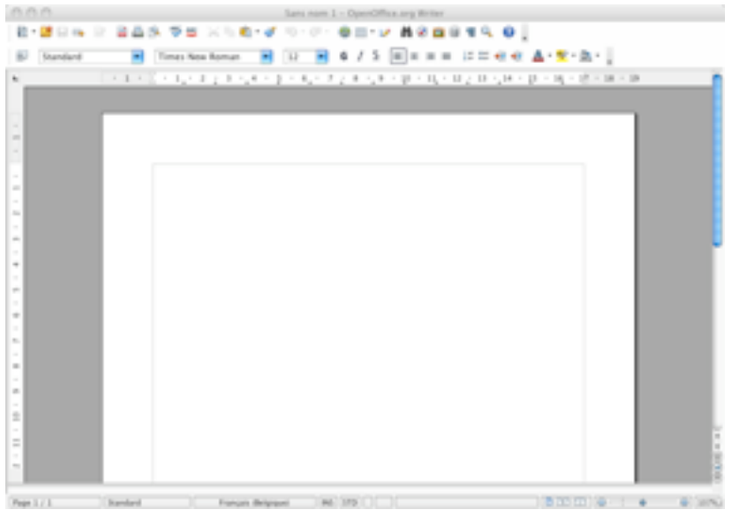
Méthode de vérification

Question sur le modèle

... ?

Abstraction

Réalité



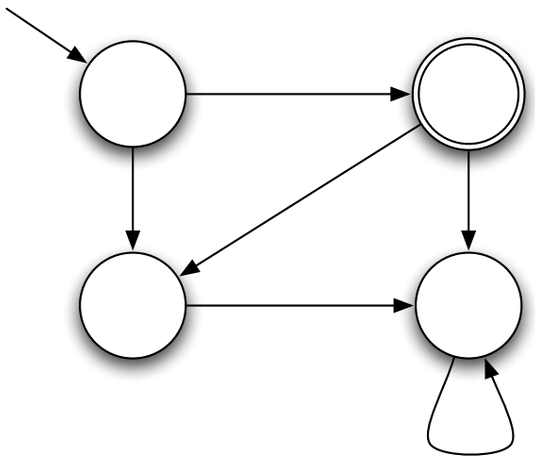
Question

La variable x prendra-t-elle une valeur ≥ 5 ?

Réponse

Quelles propriétés ?

Modèle



Méthode de vérification

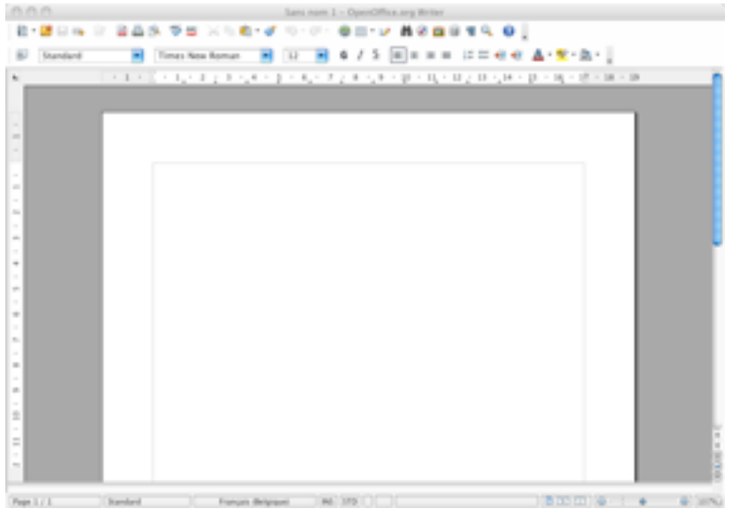
Réponse

Question sur le modèle

... ?

Abstraction

Réalité



Question

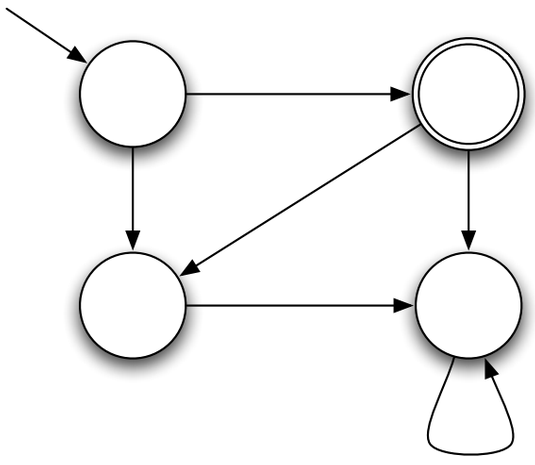
La variable x prendra-t-elle une valeur ≥ 5 ?

Réponse

Quelles méthodes ?

Réponse

Modèle



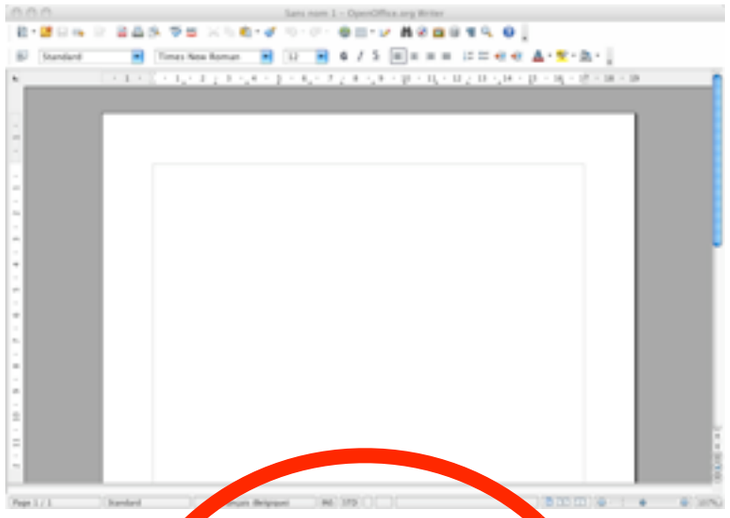
Méthode de vérification

Question sur le modèle

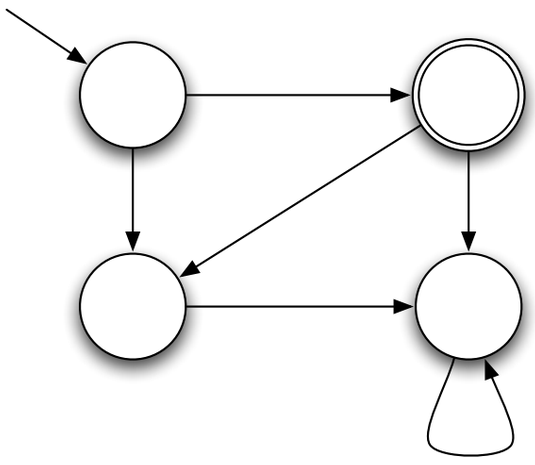
... ?

Abstraction

Réalité

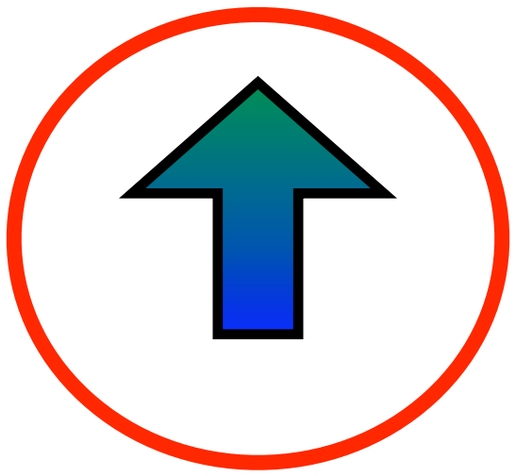


Modèle



Comment ?

Réponse

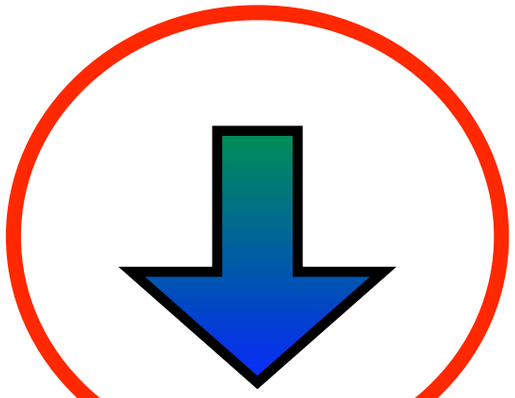


Réponse

Méthode de vérification

Question

La variable x prendra-t-elle une valeur ≥ 5 ?



Question sur le modèle

... ?

Une technique de vérification: le *model-checking* à base d'automates

Model-checking

Model-checking

- Dans cet exposé, on va considérer, pour l'exemple, le *model-checking* à base d'automates :

Model-checking

- Dans cet exposé, on va considérer, pour l'exemple, le *model-checking* à base d'automates :
  - modèle = automates finis

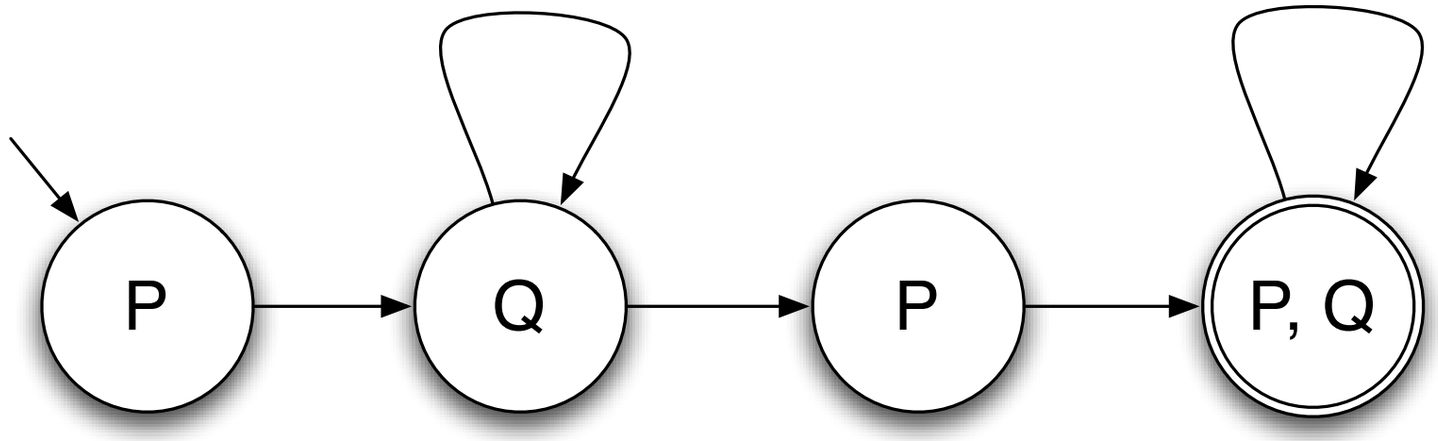
Model-checking

- Dans cet exposé, on va considérer, pour l'exemple, le *model-checking* à base d'automates :
  - modèle = automates finis
  - propriétés = propriétés exprimables dans la logique LTL

Model-checking

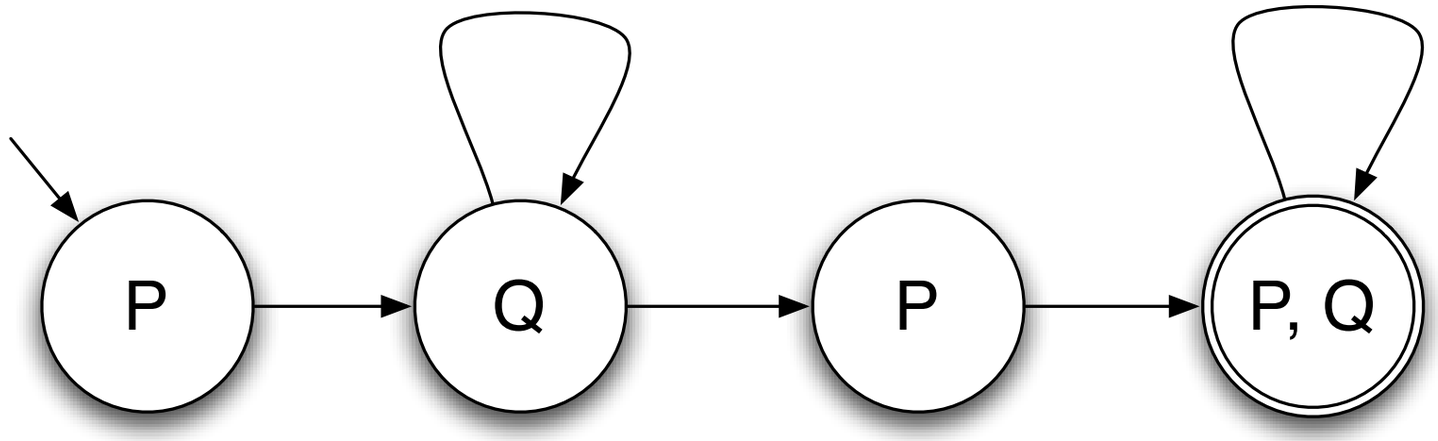
- Dans cet exposé, on va considérer, pour l'exemple, le *model-checking* à base d'automates :
  - modèle = automates finis
  - propriétés = propriétés exprimables dans la logique LTL
  - algorithme = inclusion de langages

Automate fini



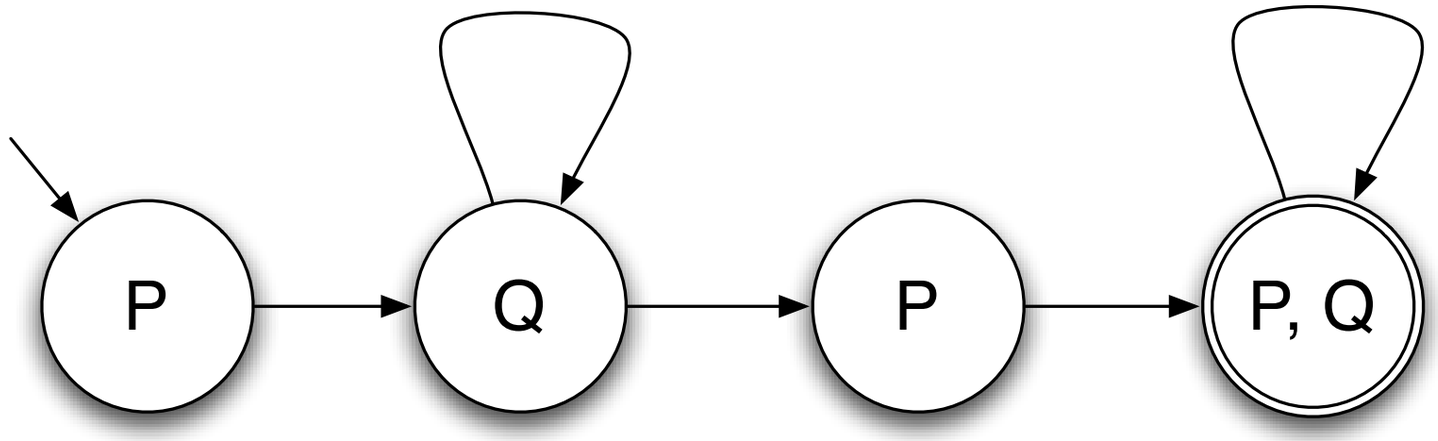
Automate fini

- Un automate fini est composé d'un ensemble d'états et d'une relation de transition



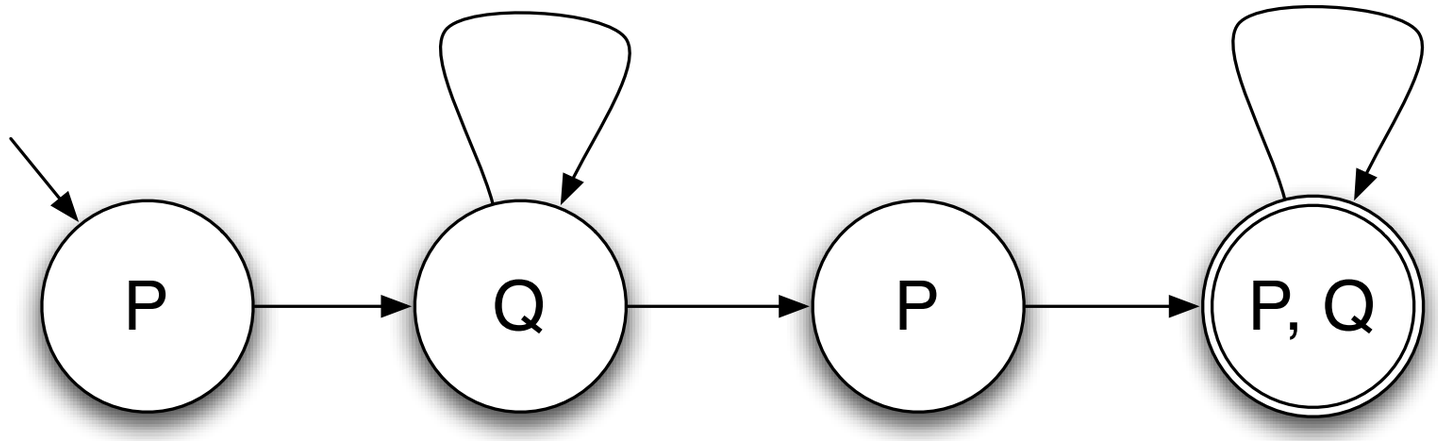
Automate fini

- Un automate fini est composé d'un ensemble d'états et d'une relation de transition
  - C'est donc une machine de Turing sans ruban



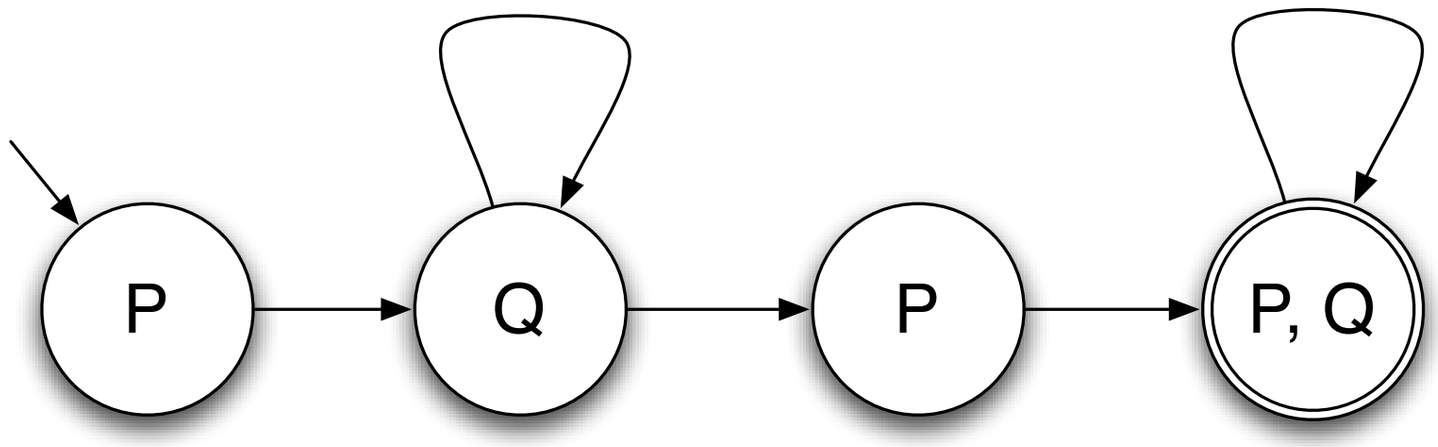
Automate fini

- Un automate fini est composé d'un ensemble d'états et d'une relation de transition
  - C'est donc une machine de Turing sans ruban
- Certains états sont accepteurs



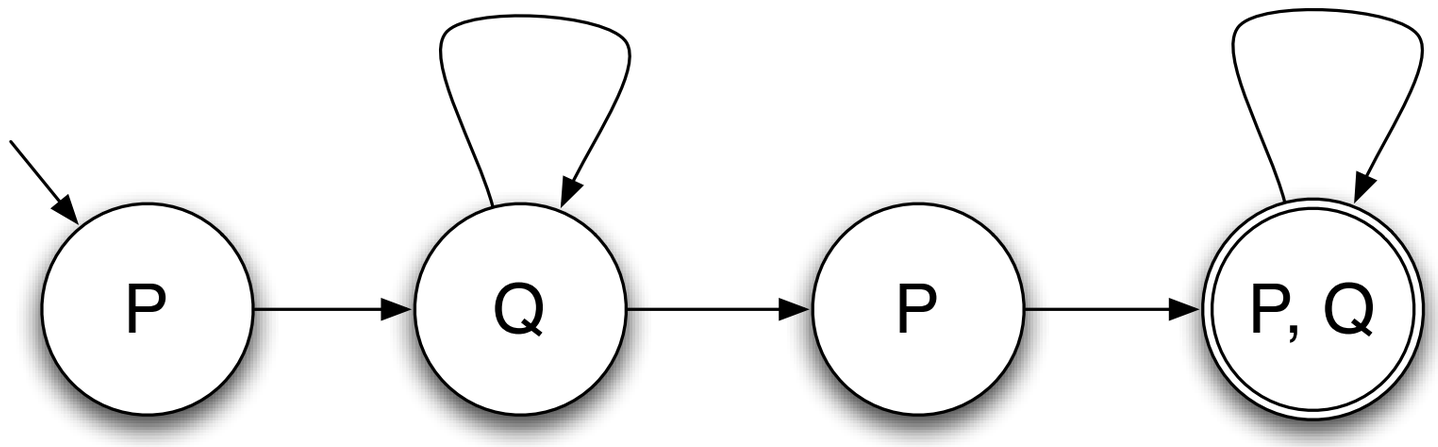
Automate fini

- Un automate fini est composé d'un ensemble d'états et d'une relation de transition
  - C'est donc une machine de Turing sans ruban
- Certains états sont accepteurs
- Les états de l'automate sont étiquetés par des propositions (vrai/faux) qui donnent des renseignements sur l'état du programme représenté



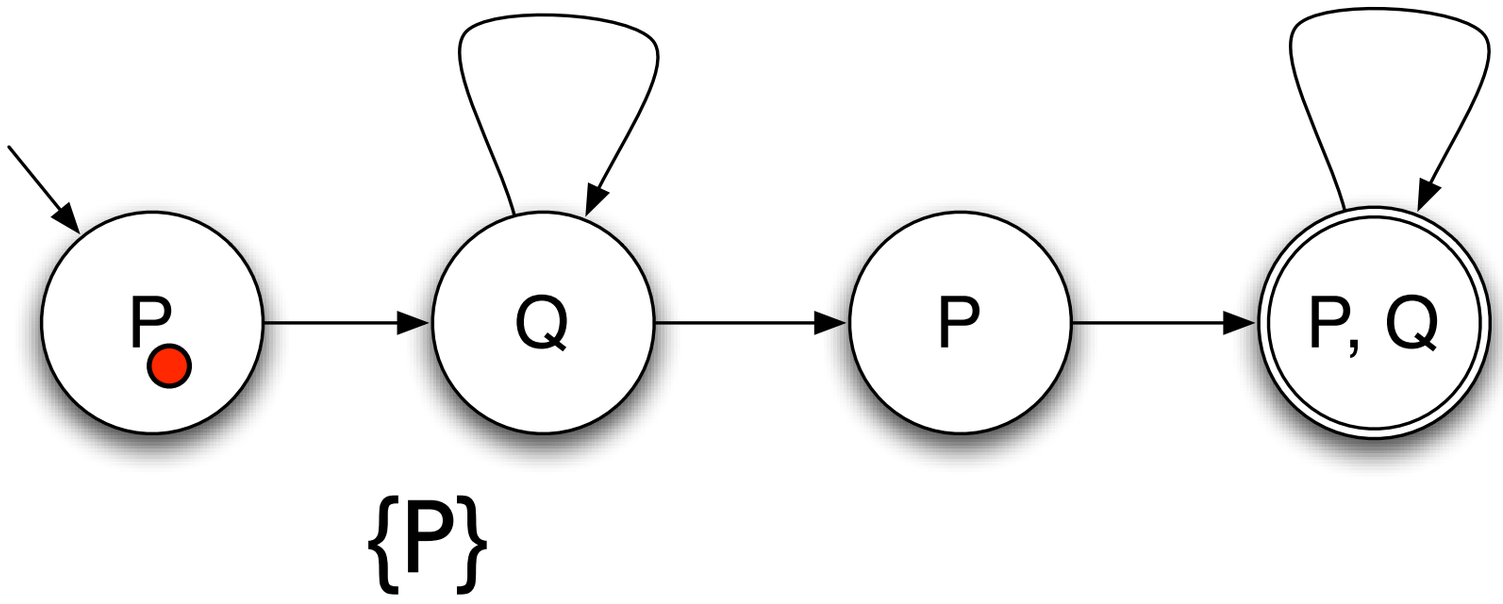
Automate fini

- Un automate fini est composé d'un ensemble d'états et d'une relation de transition
  - C'est donc une machine de Turing sans ruban
- Certains états sont accepteurs
- Les états de l'automate sont étiquetés par des propositions (vrai/faux) qui donnent des renseignements sur l'état du programme représenté
  - On n'indique que les propositions qui sont vraies



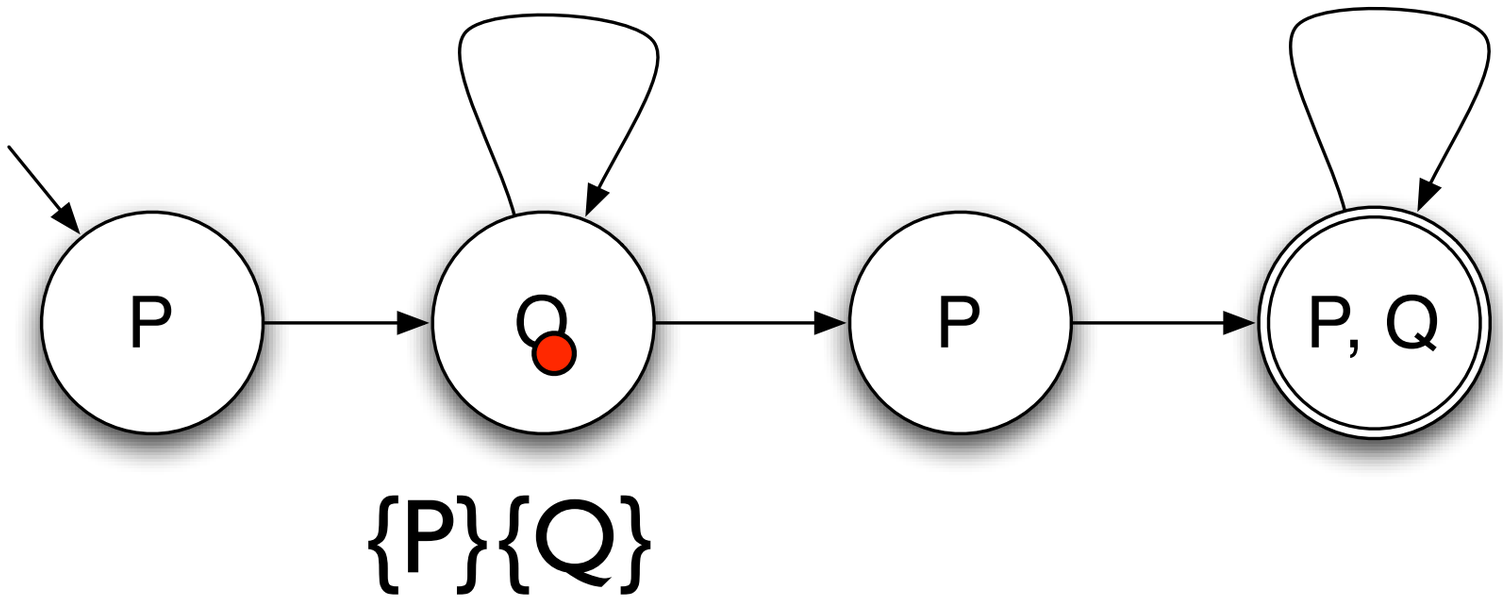
Automate fini

- Comme une machine de Turing, un automate définit un langage
  - On associe un mot à chaque chemin dans l'automate
  - Ce mot est la suite des ensembles de propositions rencontrés le long du chemin
    - Exemple : {P}, {Q}, {P}, {P,Q}, {P,Q}
  - Le langage accepté par l'automate est l'ensemble de mots qui correspondent à un chemin se terminant dans un état accepteur



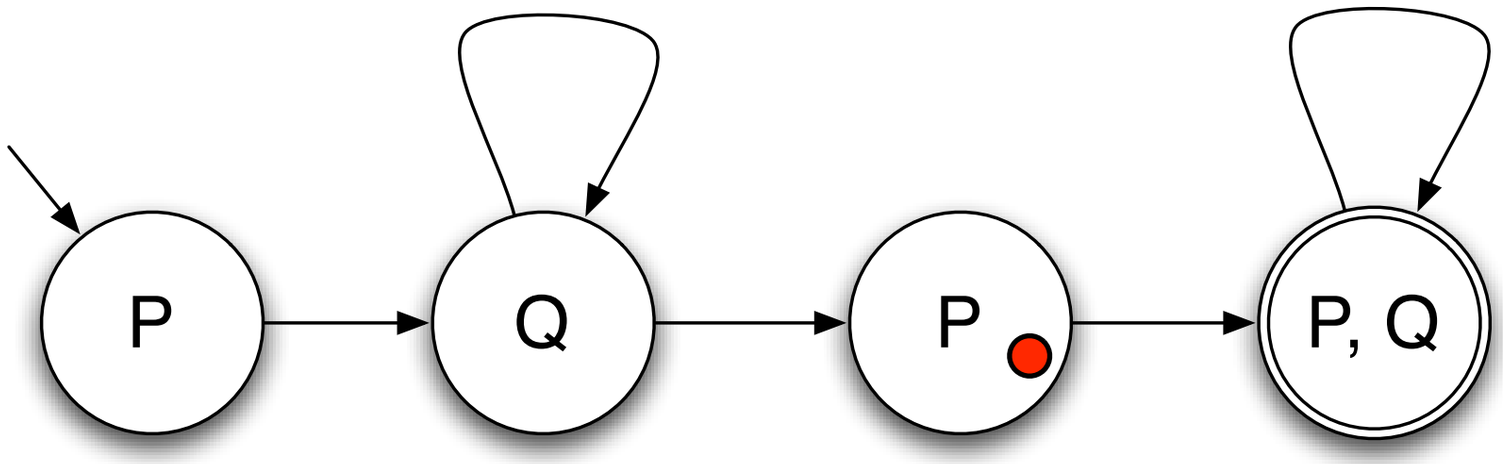
Automate fini

- Comme une machine de Turing, un automate définit un langage
  - On associe un mot à chaque chemin dans l'automate
  - Ce mot est la suite des ensembles de propositions rencontrés le long du chemin
    - Exemple : {P}, {Q}, {P}, {P,Q}, {P,Q}
  - Le langage accepté par l'automate est l'ensemble de mots qui correspondent à un chemin se terminant dans un état accepteur



Automate fini

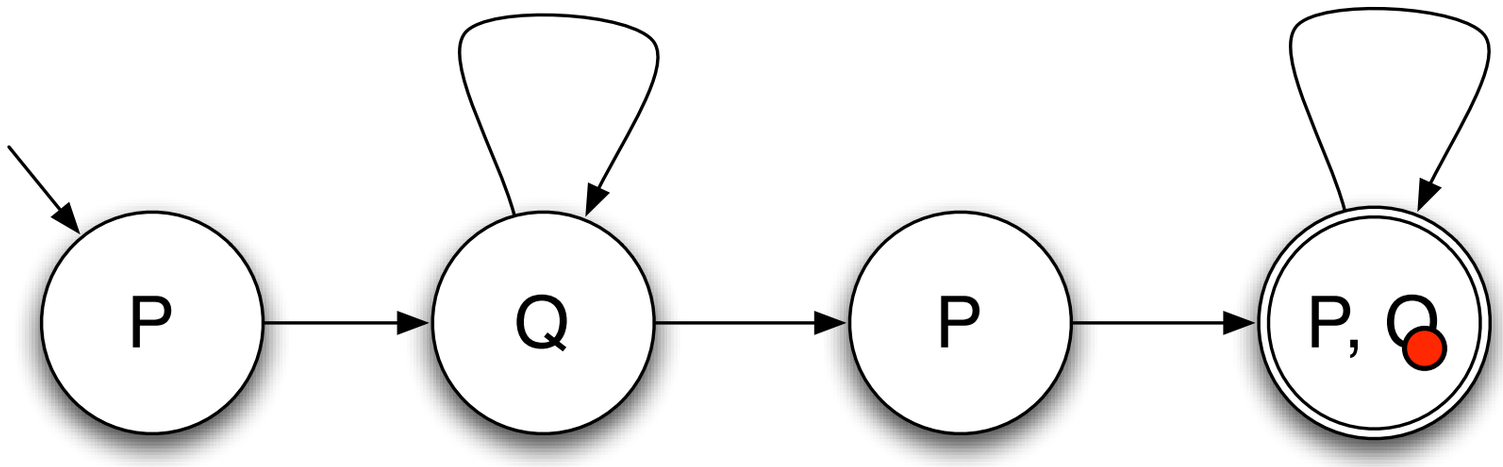
- Comme une machine de Turing, un automate définit un langage
  - On associe un mot à chaque chemin dans l'automate
  - Ce mot est la suite des ensembles de propositions rencontrés le long du chemin
    - Exemple : {P}, {Q}, {P}, {P,Q}, {P,Q}
  - Le langage accepté par l'automate est l'ensemble de mots qui correspondent à un chemin se terminant dans un état accepteur



{P}{Q} {P}

Automate fini

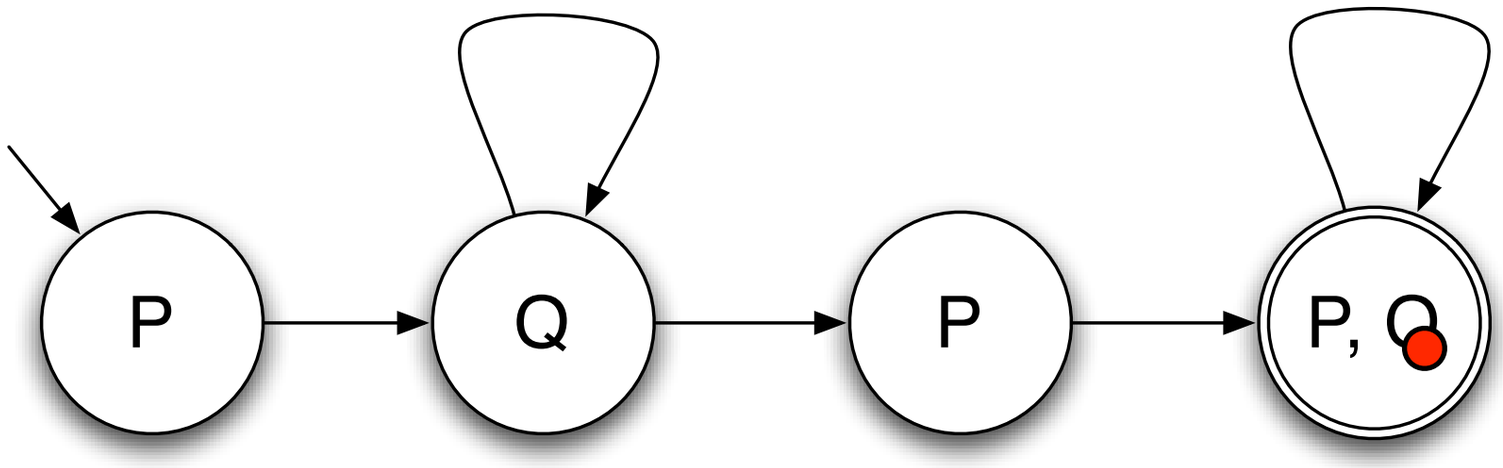
- Comme une machine de Turing, un automate définit un langage
  - On associe un mot à chaque chemin dans l'automate
  - Ce mot est la suite des ensembles de propositions rencontrés le long du chemin
    - Exemple : {P}, {Q}, {P}, {P,Q}, {P,Q}
  - Le langage accepté par l'automate est l'ensemble de mots qui correspondent à un chemin se terminant dans un état accepteur



{P}{Q} {P}{P,Q}

Automate fini

- Comme une machine de Turing, un automate définit un langage
  - On associe un mot à chaque chemin dans l'automate
  - Ce mot est la suite des ensembles de propositions rencontrés le long du chemin
    - Exemple : {P}, {Q}, {P}, {P,Q}, {P,Q}
  - Le langage accepté par l'automate est l'ensemble de mots qui correspondent à un chemin se terminant dans un état accepteur



{P}{Q} {P}{P,Q}{P,Q}

Automate fini

- Con finit un lang
  - O
  - C és le long du ch
  -
- Le i correspondent à un chemin se terminant dans un état accepteur

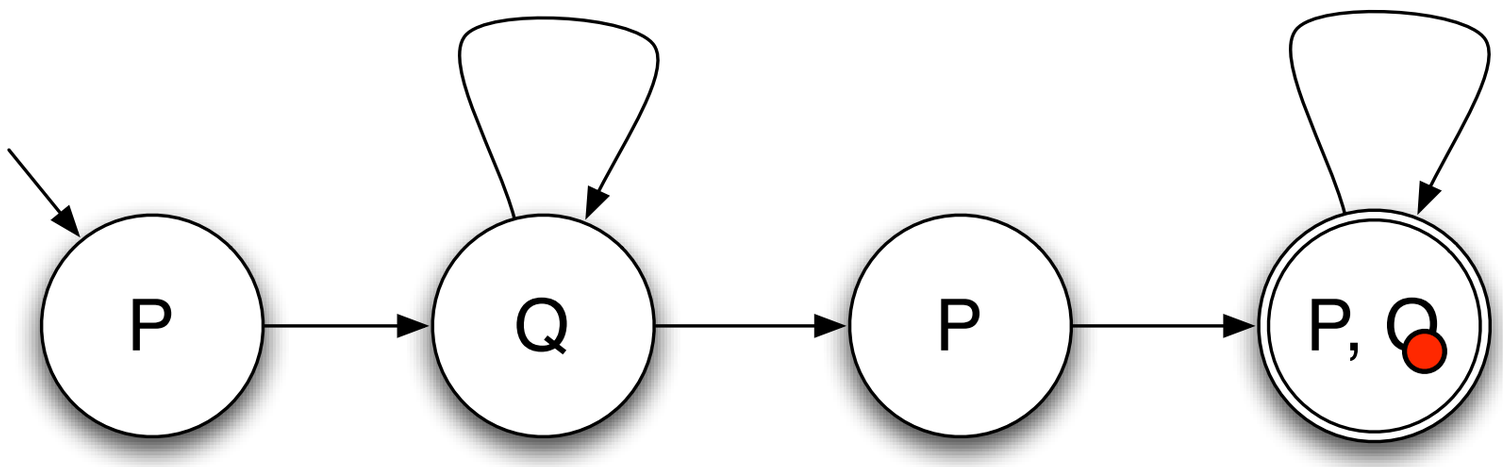
Le langage de cet automate est l'ensemble des mots de la forme

{P} {Q}....{Q} {P} {P,Q}....{P,Q}

ou encore l'ensemble des mots de la forme

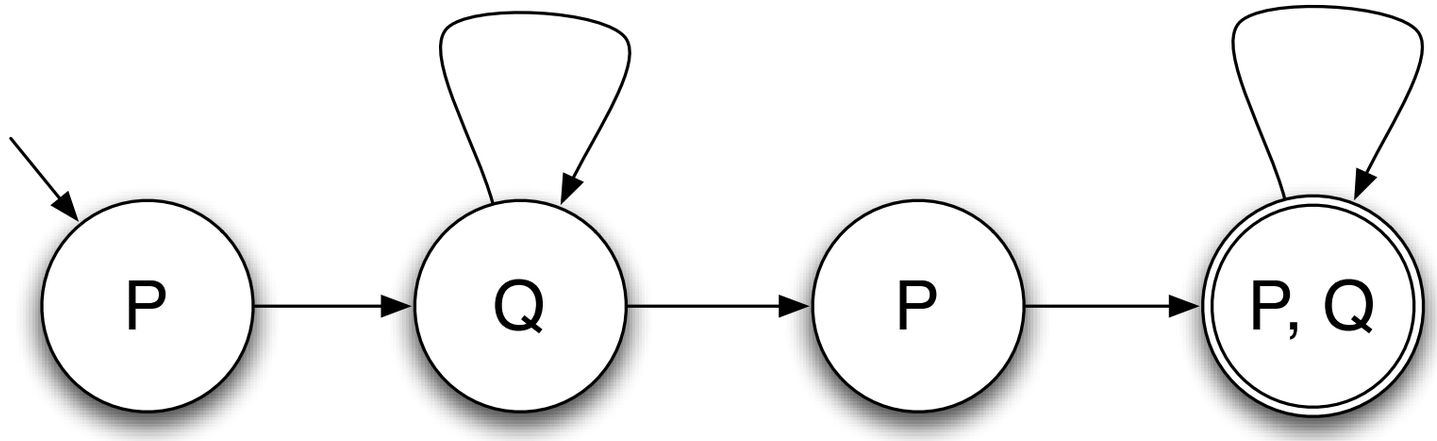
$\{P\}\ \{Q\}^n\ \{P\}\ \{P,Q\}^m$

pour $n, m \geq 1$



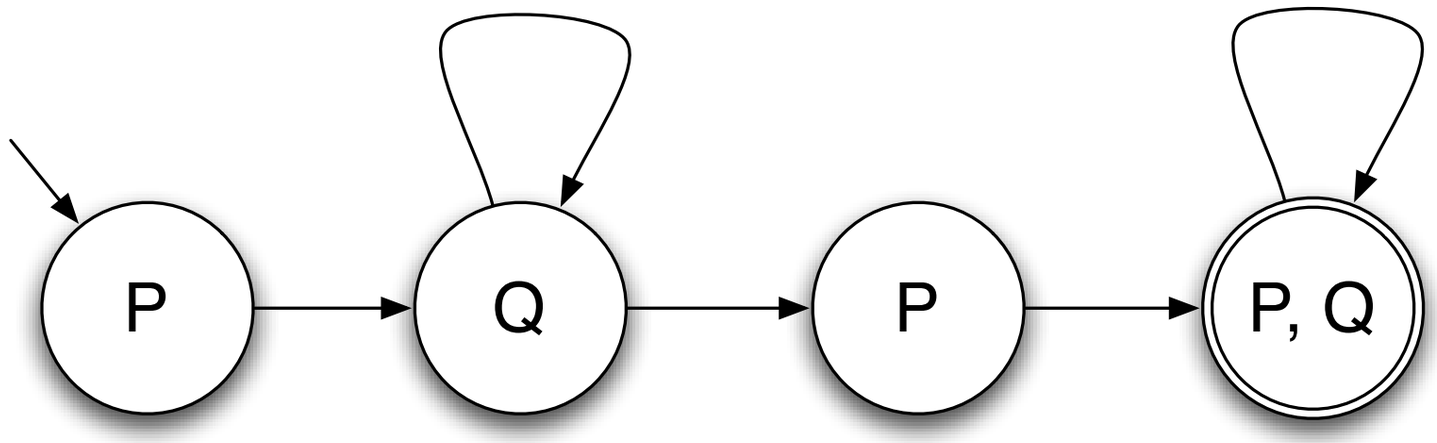
{P}{Q} {P}{P,Q}{P,Q}

Automate fini



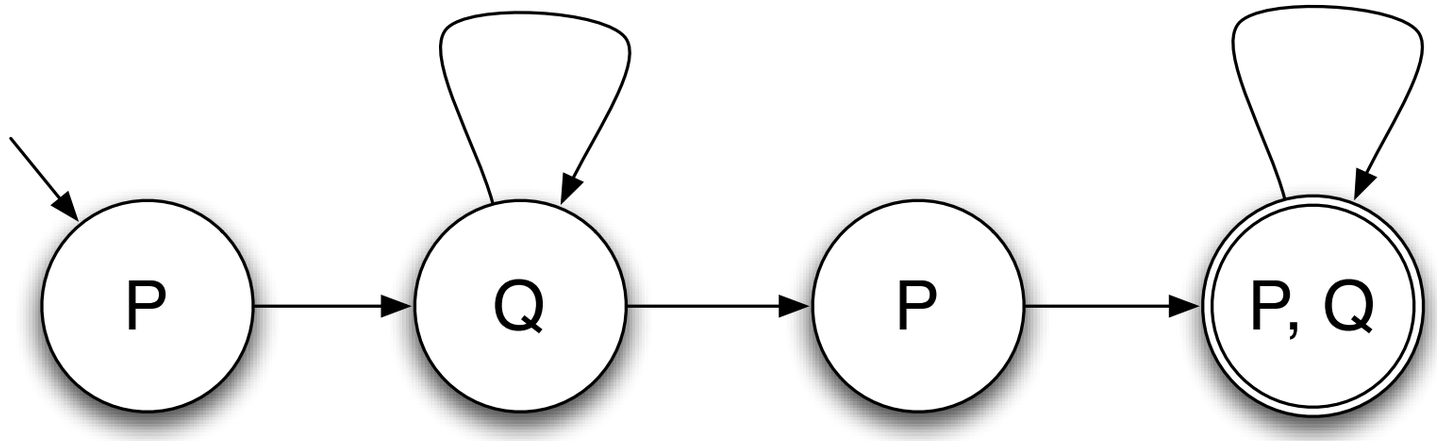
Automate fini

- On utilise les automates finis pour représenter les exécutions d'un programme informatique



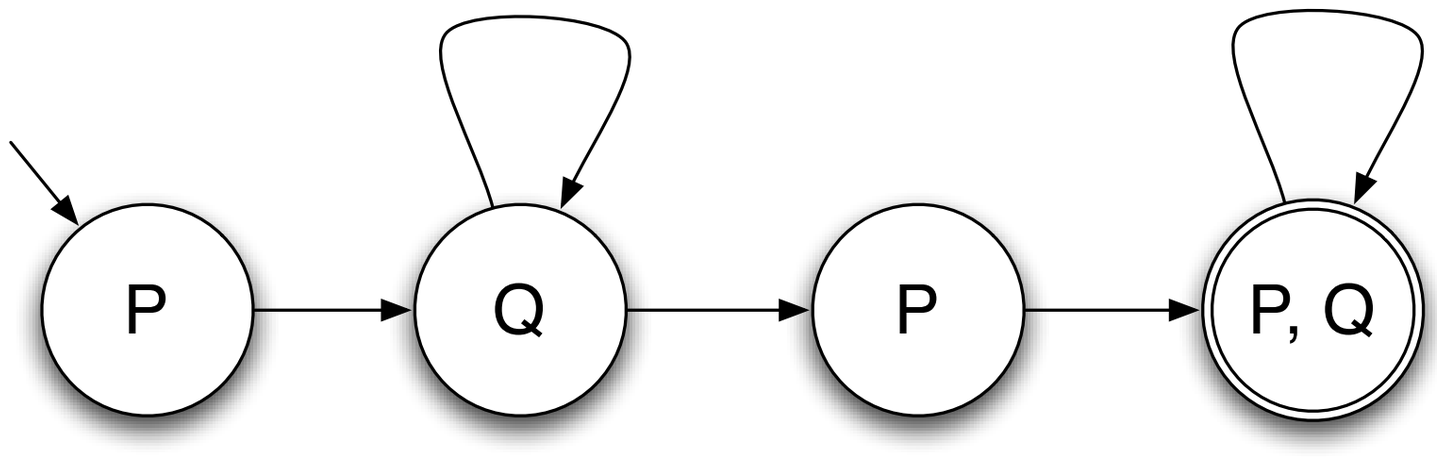
Automate fini

- On utilise les automates finis pour représenter les exécutions d'un programme informatique
  - Les propositions encodent des informations sur les états du programme



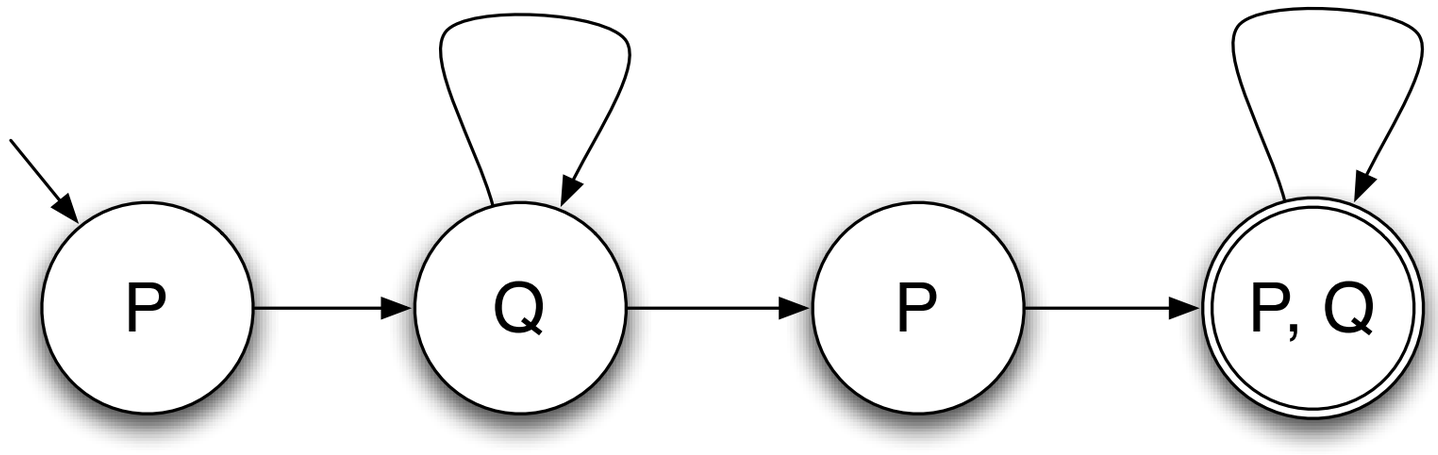
Automate fini

- On utilise les automates finis pour représenter les exécutions d'un programme informatique
  - Les propositions encodent des informations sur les états du programme
    - Par exemple: P signifie que la variable x vaut 5, Q signifie que la variable y vaut 9



Automate fini

- On utilise les automates finis pour représenter les exécutions d'un programme informatique
  - Les propositions encodent des informations sur les états du programme
    - Par exemple: P signifie que la variable x vaut 5, Q signifie que la variable y vaut 9
  - Chaque mot accepté correspond à une exécution possible du programme



Logique temporelle linéaire

- Comment spécifier les propriétés de manière naturelle ?
  - En français : «chaque fois qu'on visite un état où P est vrai, celui-ci doit obligatoirement être suivi d'un état où Q est vrai»
  - Le français est trop ambigu : on a besoin d'un outil plus formel
- En 1977, Amir Pnueli propose d'utiliser la logique temporelle linéaire (*Linear Temporal Logic* - LTL) pour spécifier les propriétés des programmes de manière formelle mais intuitive
  - Prix Turing en 1996 pour ces travaux

A. Pnueli
1941 - 2009

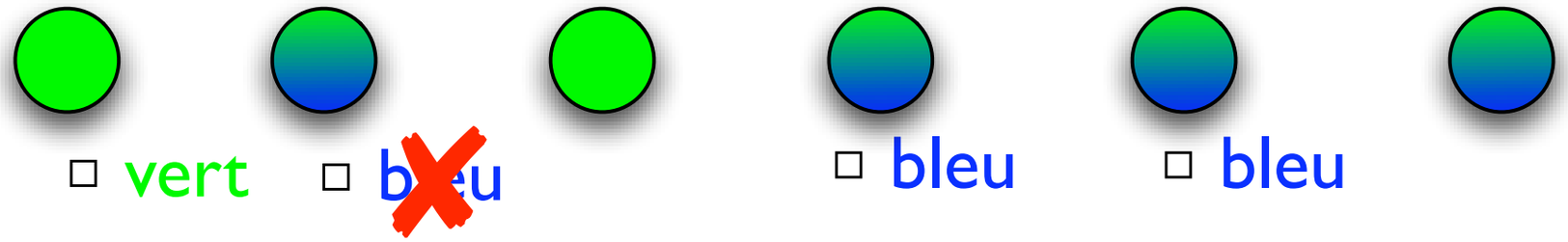
Logique temporelle linéaire

- Une formule de LTL permet de définir un ensemble de mots (= exécutions), en spécifiant les propriétés que ceux-ci doivent respecter, à l'aide de modalités temporelles

Logique temporelle linéaire

- Une formule de LTL permet de définir un ensemble de mots (= exécutions), en spécifiant les propriétés que ceux-ci doivent respecter, à l'aide de modalités temporelles

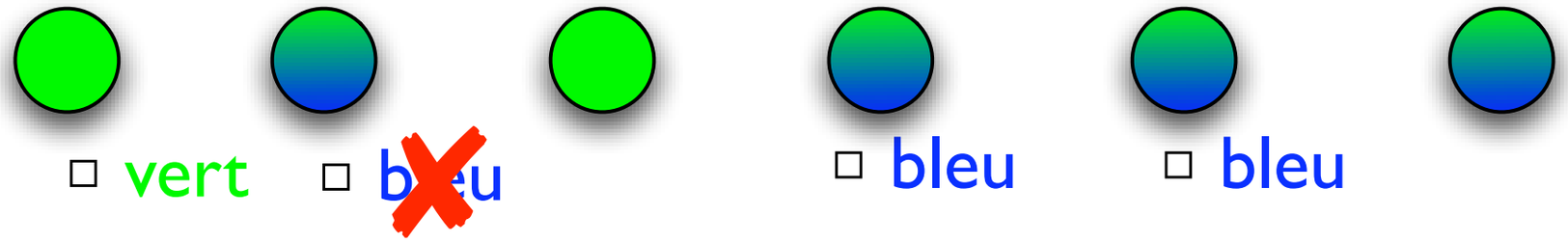
« Toujours » : □ φ = la propriété φ est vérifiée dans tous les états futurs



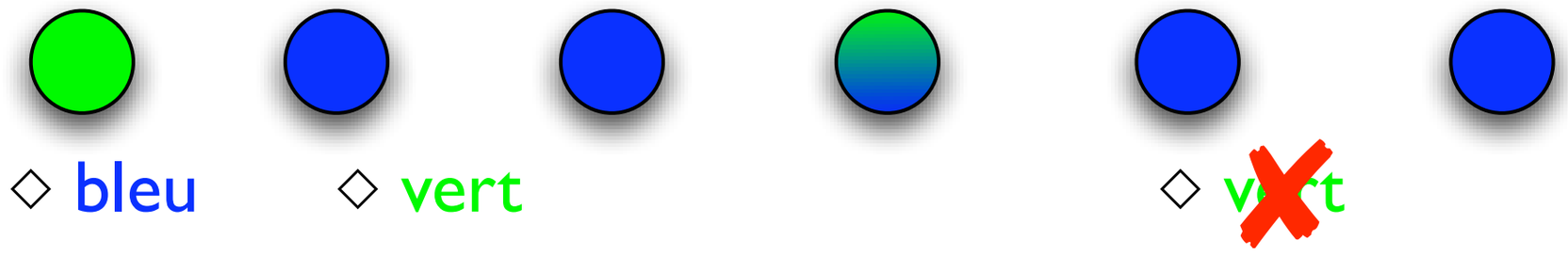
Logique temporelle linéaire

- Une formule de LTL permet de définir un ensemble de mots (= exécutions), en spécifiant les propriétés que ceux-ci doivent respecter, à l'aide de modalités temporelles

« Toujours » : □ φ = la propriété φ est vérifiée dans tous les états futurs



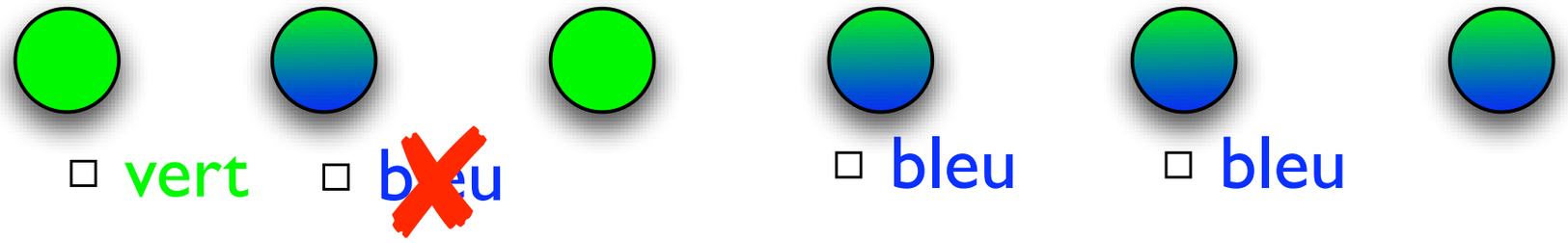
« Finalement » : ◇ φ = il existe un état futur où la propriété φ est vérifiée



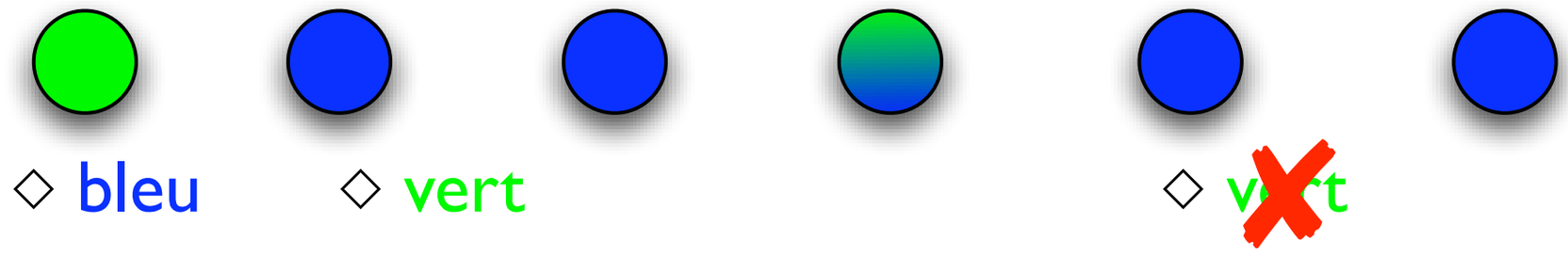
Logique temporelle linéaire

- Une formule de LTL permet de définir un ensemble de mots (= exécutions), en spécifiant les propriétés que ceux-ci doivent respecter, à l'aide de modalités temporelles

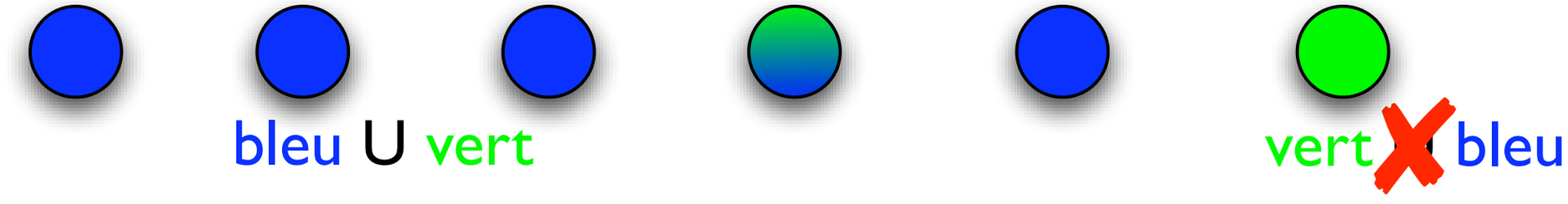
« Toujours » : □ φ = la propriété φ est vérifiée dans tous les états futurs



« Finalement » : ◇ φ = il existe un état futur où la propriété φ est vérifiée



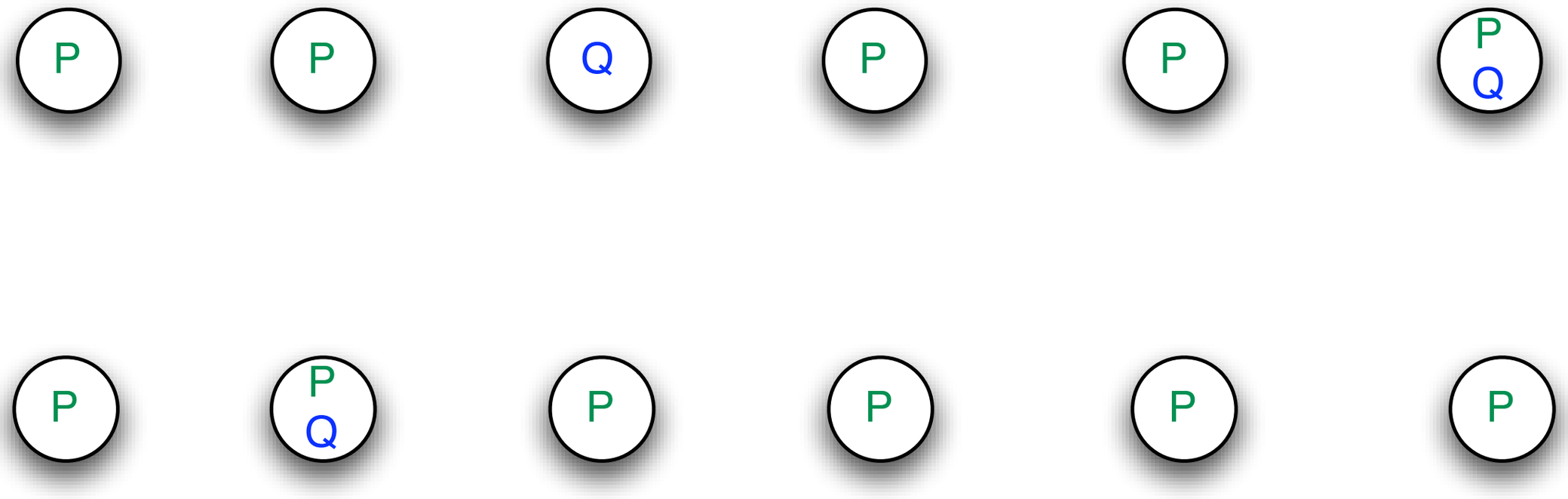
« Jusqu'à ce que » : $\varphi_1 \, U \, \varphi_2$ = φ_1 est vérifiée jusqu'à ce que φ_2 le soit



Logique temporelle linéaire

- Exemple :
  « Chaque fois qu'on visite un état où P est vrai, celui-ci doit obligatoirement être suivi d'un état où Q est vrai »

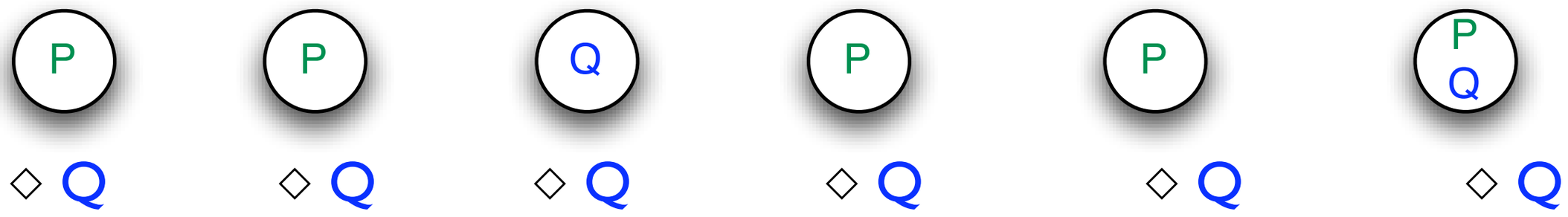
$$\Box (P \rightarrow \Diamond Q)$$



Logique temporelle linéaire

- Exemple :
  « Chaque fois qu'on visite un état où P est vrai, celui-ci doit obligatoirement être suivi d'un état où Q est vrai »

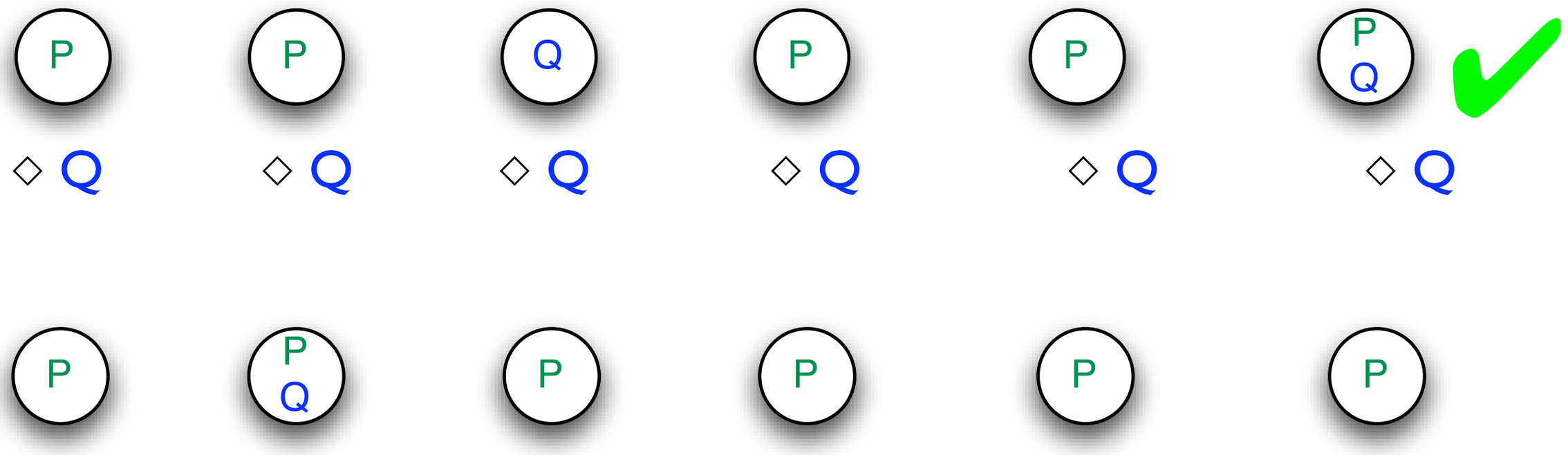
$$\Box (P \rightarrow \Diamond Q)$$



Logique temporelle linéaire

- Exemple :
  « Chaque fois qu'on visite un état où P est vrai, celui-ci doit obligatoirement être suivi d'un état où Q est vrai »

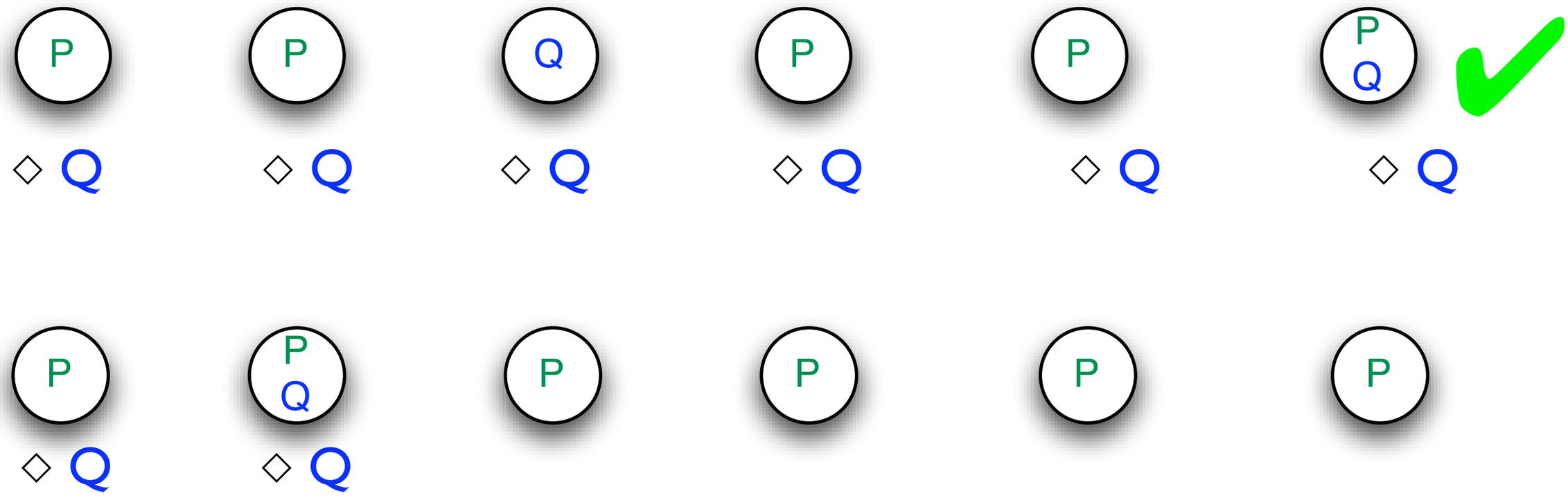
$$\Box (P \rightarrow \Diamond Q)$$



Logique temporelle linéaire

- Exemple :
  « Chaque fois qu'on visite un état où P est vrai, celui-ci doit obligatoirement être suivi d'un état où Q est vrai »

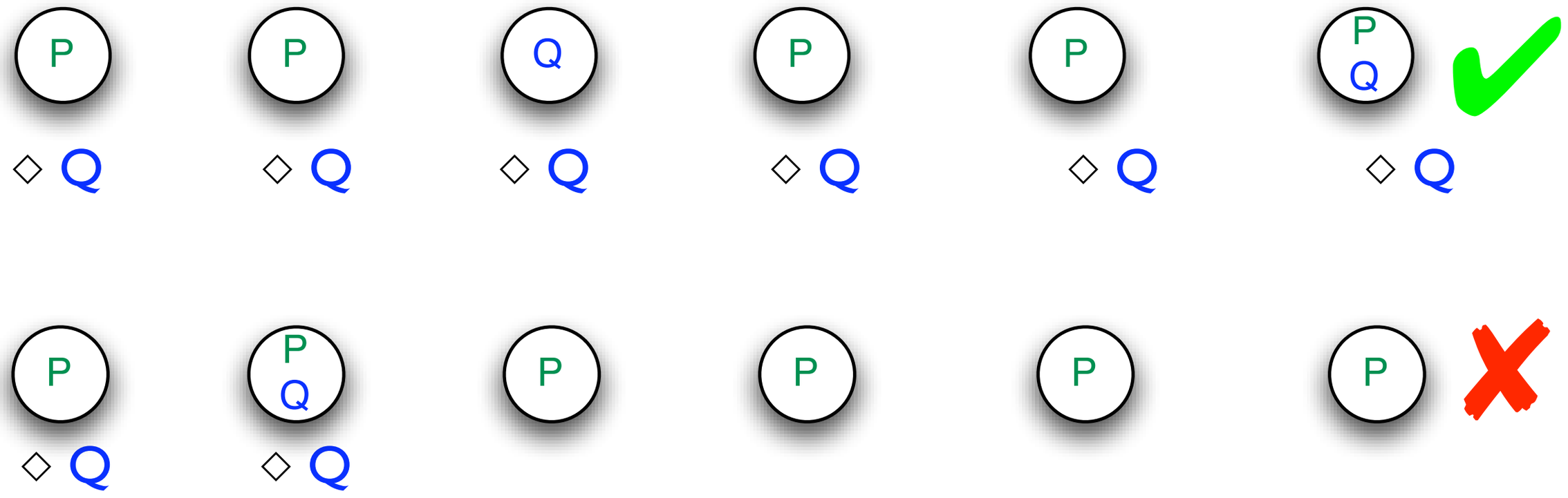
$$\Box (P \rightarrow \Diamond Q)$$



Logique temporelle linéaire

- Exemple :
  « Chaque fois qu'on visite un état où P est vrai, celui-ci doit obligatoirement être suivi d'un état où Q est vrai »

$$\Box (P \rightarrow \Diamond Q)$$



Model-Checking

Model-Checking

- Soit un automate A qui spécifie un système informatique
  - Cet automate définit un langage L(A) qui est un ensemble de mots
  - Ces mots représentent tous les comportements possibles du système étudié

Model-Checking

- Soit un automate A qui spécifie un système informatique
  - Cet automate définit un langage L(A) qui est un ensemble de mots
  - Ces mots représentent tous les comportements possibles du système étudié
- Soit une formule de LTL φ qui représentent les exécutions admissibles du système
  - Cette formule définit un autre langage L(φ) qui est aussi un ensemble de mots

Model-Checking

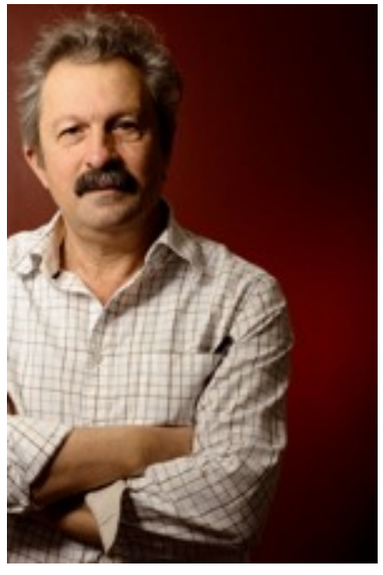
- Soit un automate A qui spécifie un système informatique
  - Cet automate définit un langage L(A) qui est un ensemble de mots
  - Ces mots représentent tous les comportements possibles du système étudié
- Soit une formule de LTL φ qui représentent les exécutions admissibles du système
  - Cette formule définit un autre langage L(φ) qui est aussi un ensemble de mots
- Le problème de *model-checking* consiste à déterminer si toutes les exécutions de A donnent lieu à un mot qui satisfait φ
  - Autrement dit : si $L(A) \subseteq L(\varphi)$

Model-Checking

- Clarke, Emerson, Sifakis (1981) : Le problème de *model checking* est décidable
  - Ce n'est pas le cas si le système est spécifié sous forme d'une machine de Turing (même si on spécifie les propriétés à l'aide de LTL)
  - Prix Turing en 2007 pour ces résultats

E. M. Clarke

E. A. Emerson



J. Sifakis

LTL et automates

- LTL et les automates finis définissent des langages
- Quel est le rapport formel entre ces modèles ?
- En 1986, M. Vardi et P. Wolper proposent d'exprimer les formules de LTL sous forme d'un automate qui accepte le même langage

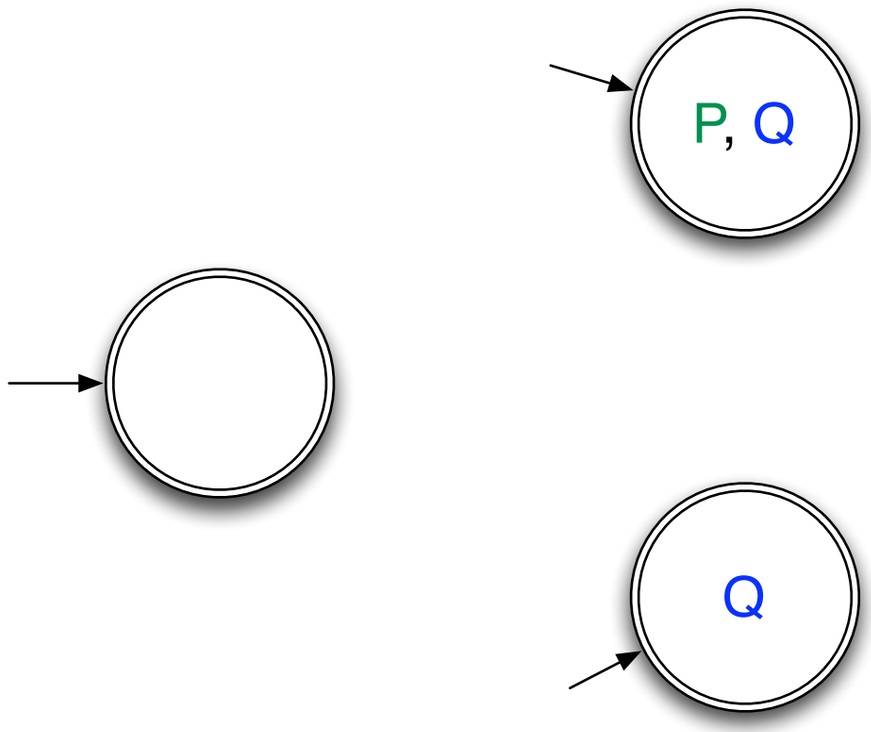
- Pour toute formule LTL φ, il existe un automate A_φ t.q. $L(\varphi) = L(A_\varphi)$
- Prix Gödel en 2000 pour une extension de ces travaux

P. Wolper

M. Vardi

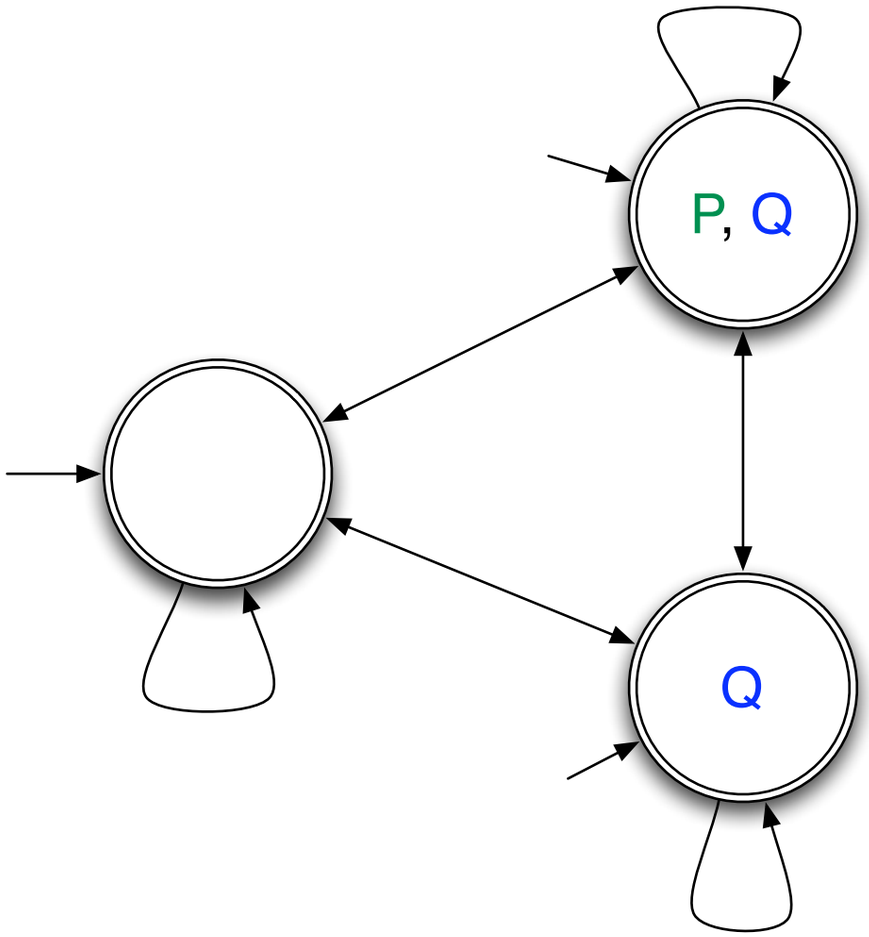
LTL et automates

Exemple : □ (P → ◇ Q)



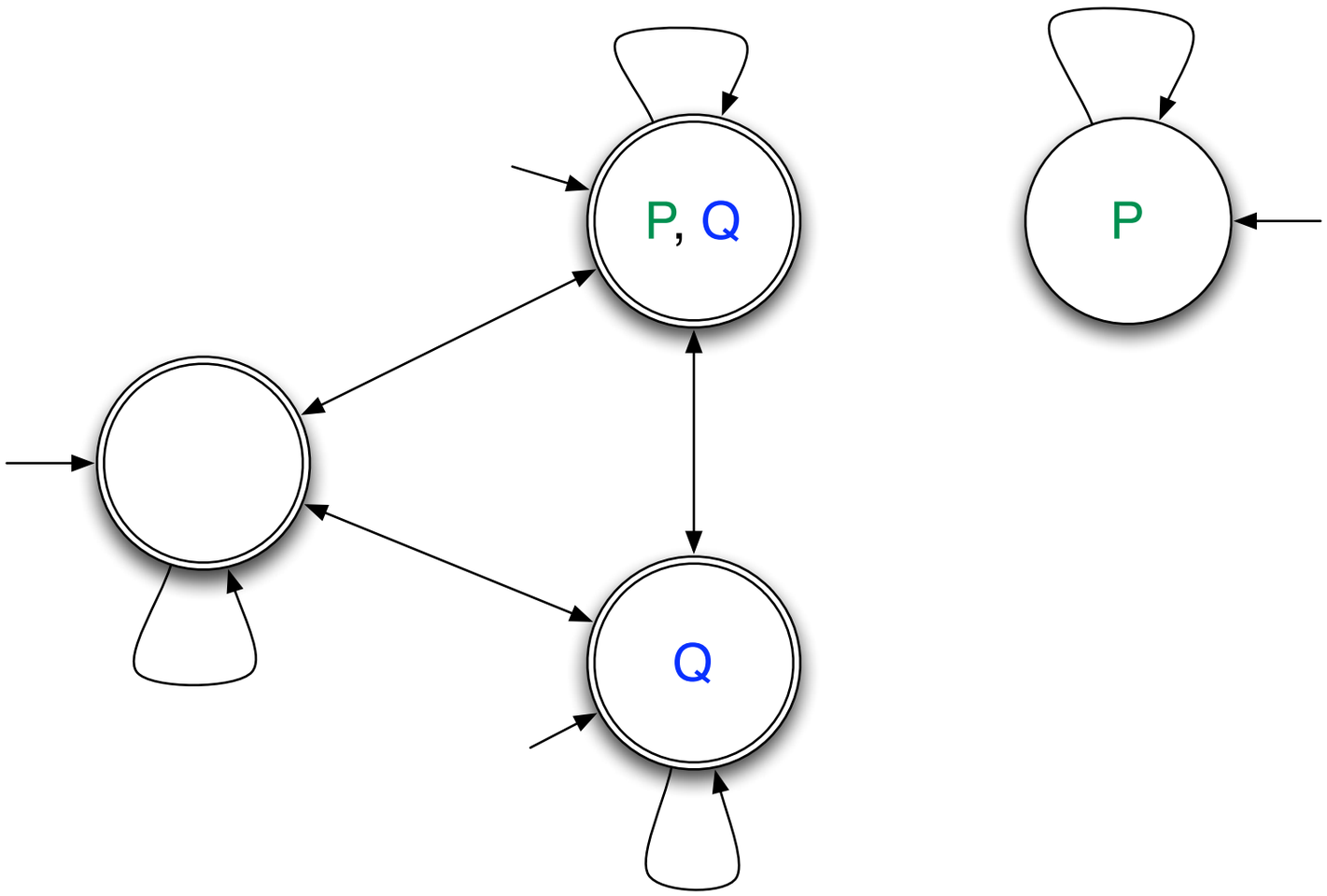
LTL et automates

Exemple : □ (P → ◇ Q)



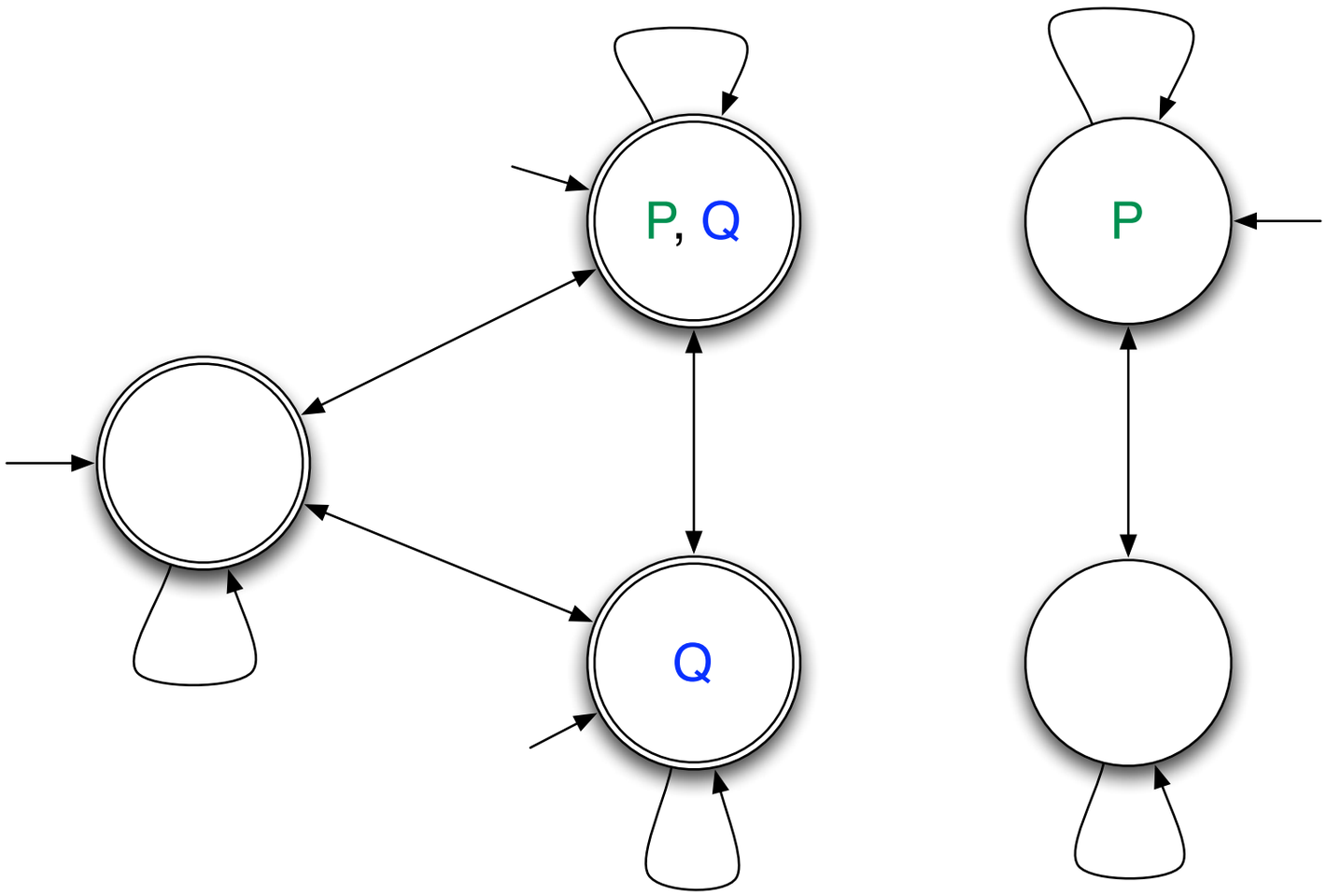
LTL et automates

Exemple : □ (P → ◇ Q)



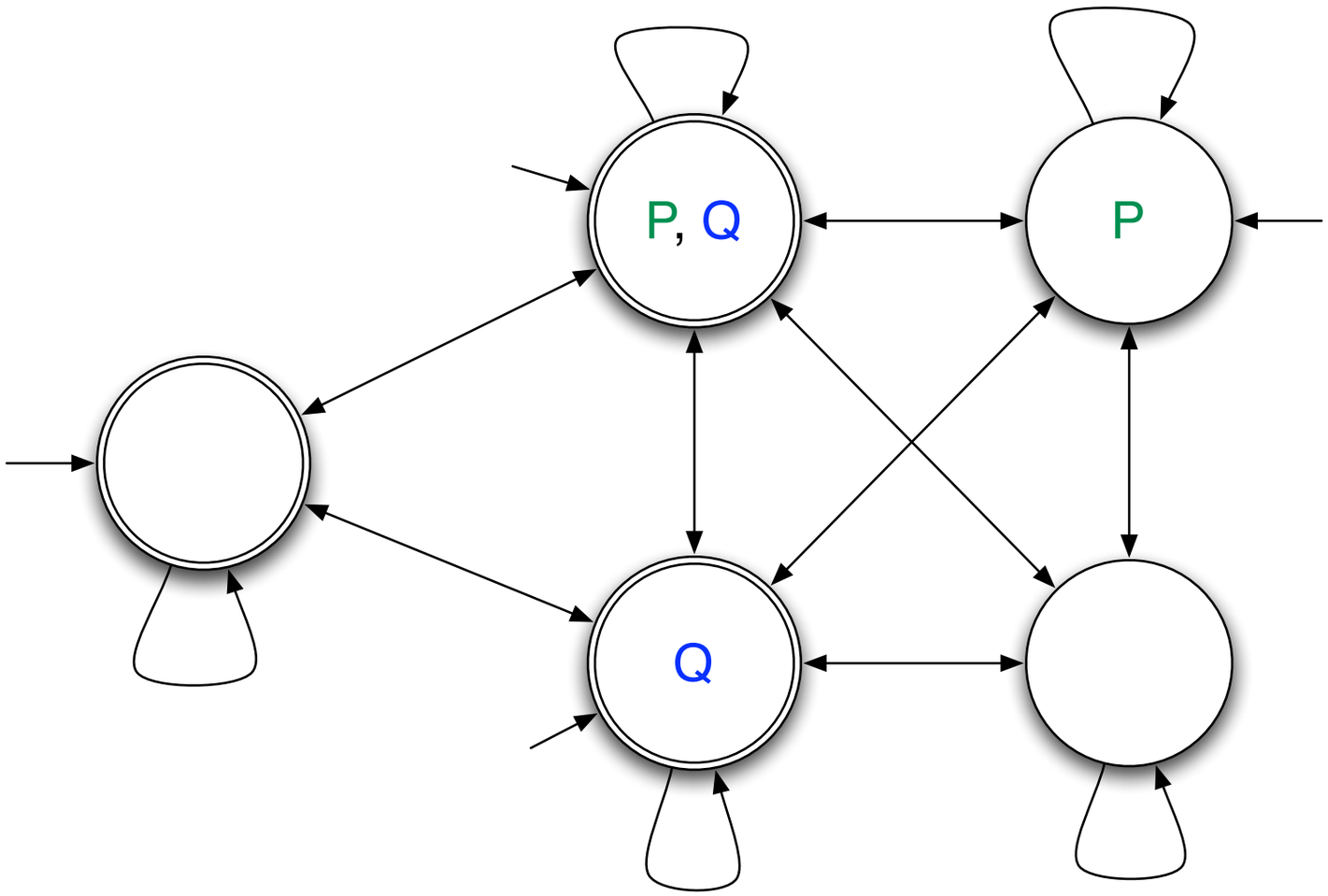
LTL et automates

Exemple : □ (P → ◇ Q)



LTL et automates

Exemple : □ (P → ◇ Q)

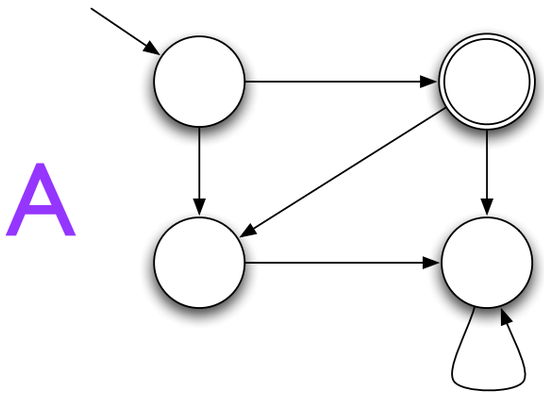


Model checking avec les automates

Système

Propriété

φ $\quad \Box (P \rightarrow \Diamond Q)$

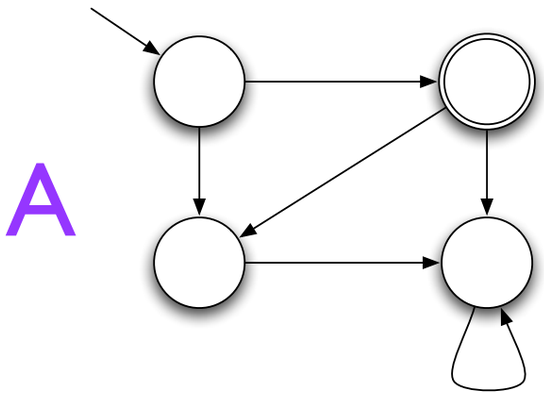


Model checking avec les automates

Système

Propriété

φ $\quad \Box (P \rightarrow \Diamond Q)$

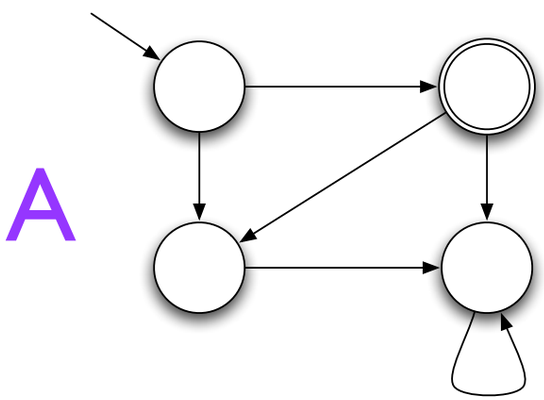


Model checking avec les automates

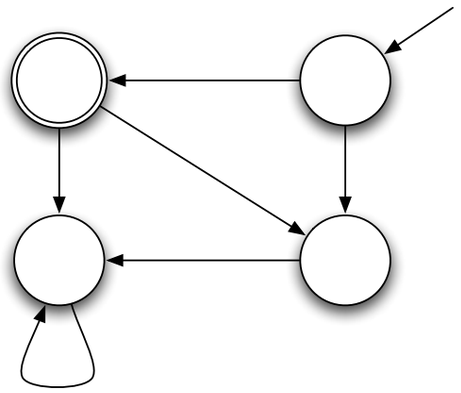
Système

Propriété

φ $\quad \Box (P \rightarrow \Diamond Q)$



A_φ

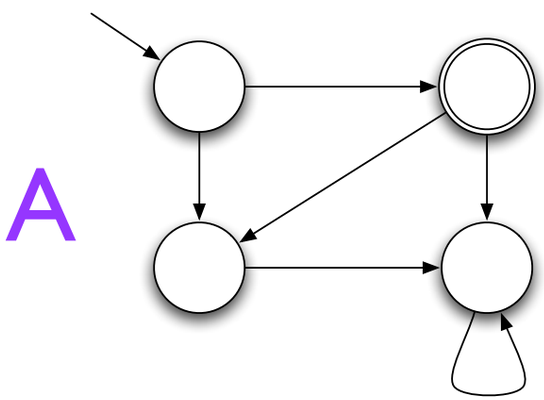


Model checking avec les automates

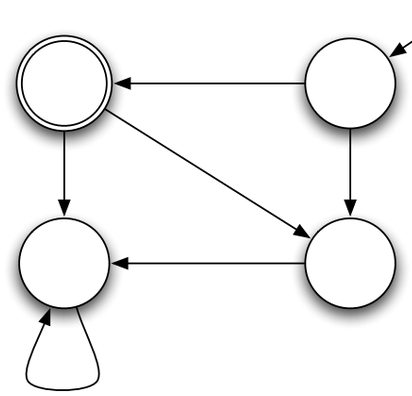
Système

Propriété

φ $\quad \Box (P \rightarrow \Diamond Q)$



A_φ



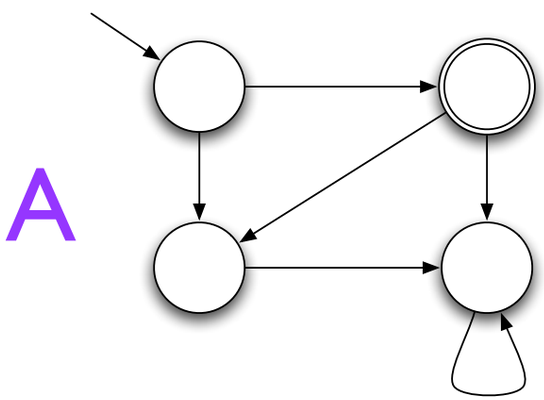
$L(A) \subseteq L(A_\varphi)$?

Model checking avec les automates

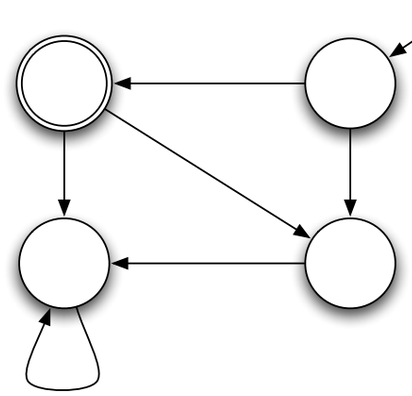
Système

Propriété

φ $\quad \Box (P \rightarrow \Diamond Q)$



A_φ



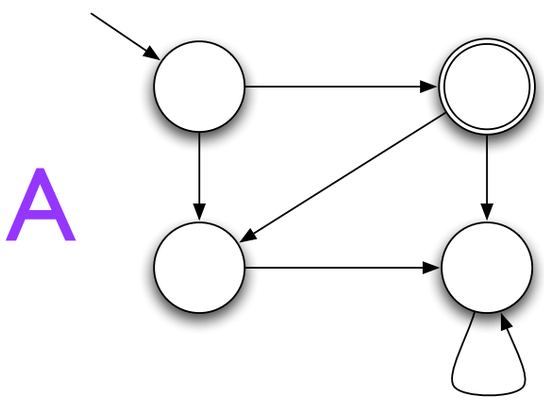
$L(A) \subseteq L(A_\varphi)$?

Model checking avec les automates

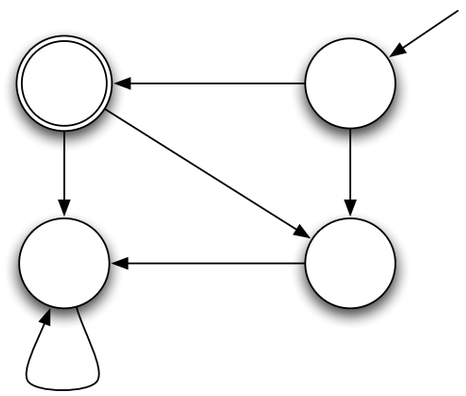
Système

Propriété

φ $\quad \Box (P \rightarrow \Diamond Q)$

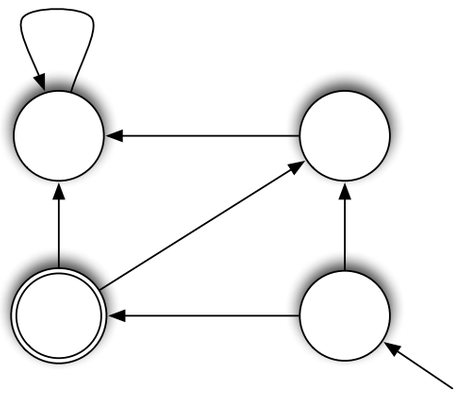


A_φ



$L(A) \subseteq L(A_\varphi)$?

$A_{\neg\varphi}$

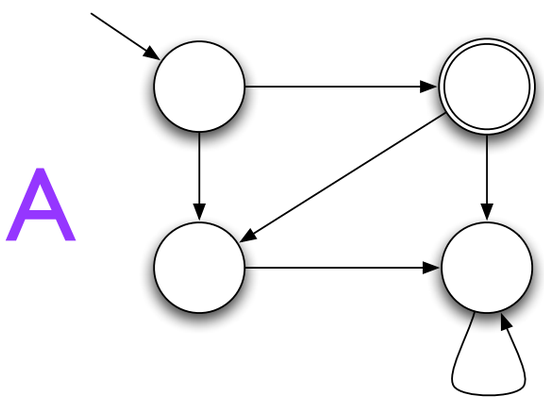


Model checking avec les automates

Système

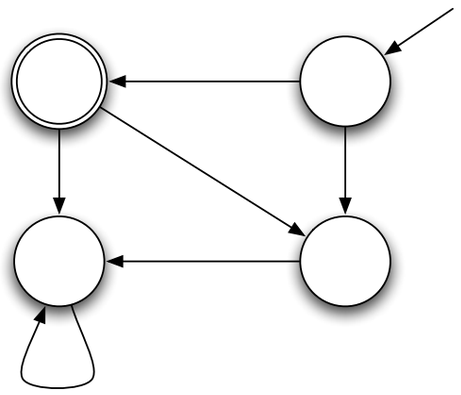
Propriété

φ $\quad \Box (P \rightarrow \Diamond Q)$

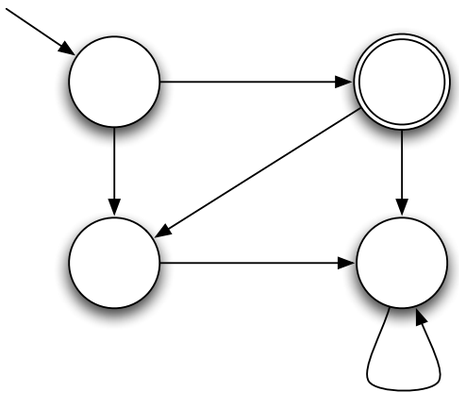


A

A_φ

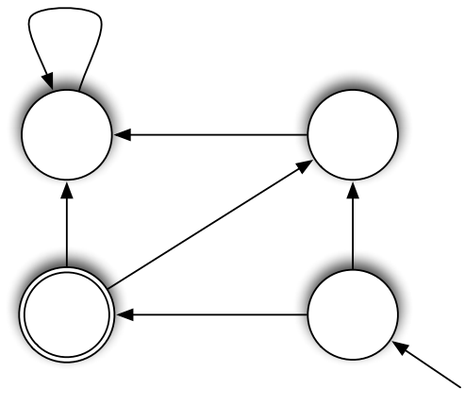


$L(A) \subseteq L(A_\varphi)$?



A

$A_{\neg\varphi}$



$L(A) \cap L(A_{\neg\varphi}) = \varnothing$?

Langages réguliers

Langages réguliers

- Le problème d'inclusion de langages est décidable sur les automates finis, pas sur les machines de Turing
  - Les automates sont « plus faibles » que les machines de Turing, et donc plus faciles à analyser

Langages réguliers

- Le problème d'inclusion de langages est décidable sur les automates finis, pas sur les machines de Turing
  - Les automates sont « plus faibles » que les machines de Turing, et donc plus faciles à analyser
- Que peut-on représenter avec une TM mais pas avec un automate fini ?
  - De tels langages doivent nécessairement exister pour ne pas contredire le résultat d'indécidabilité

Langages réguliers

- Le problème d'inclusion de langages est décidable sur les automates finis, pas sur les machines de Turing
  - Les automates sont « plus faibles » que les machines de Turing, et donc plus faciles à analyser
- Que peut-on représenter avec une TM mais pas avec un automate fini ?
  - De tels langages doivent nécessairement exister pour ne pas contredire le résultat d'indécidabilité
- Y a-t-il des langages qu'on peut représenter avec un automate mais pas avec une formule de LTL ?

Langages réguliers

Langages réguliers

- Le langage $\{P^nQ^n \mid n \geq 1\}$ n'est pas représentable par un automate fini
  - Ce langage contient tous les mots de la forme P...PQ...Q où le nombre de positions où P est vraie est égale au nombre de positions où Q est vraie. Par exemple : PPPQQQ, mais pas PPQQQQQQQQ

Langages réguliers

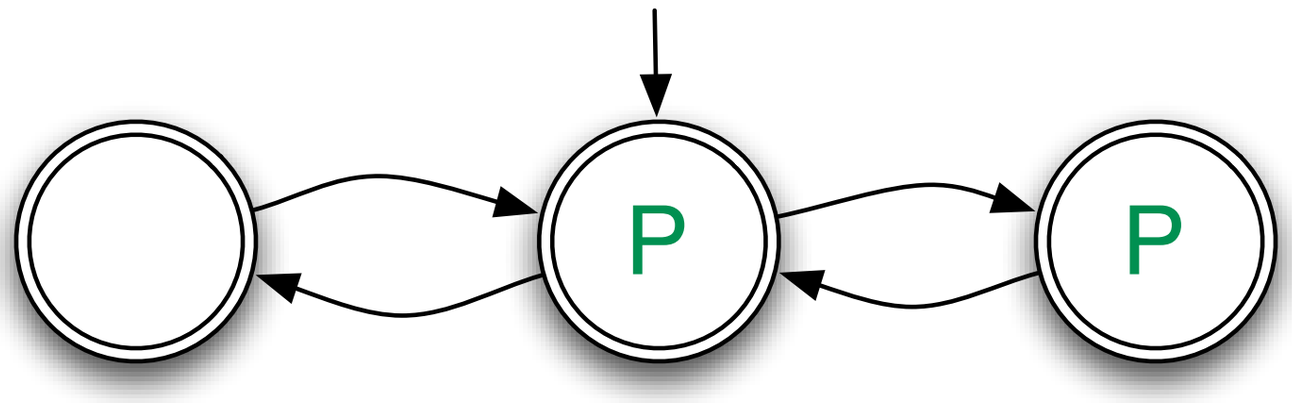
- Le langage $\{P^nQ^n \mid n \geq 1\}$ n'est pas représentable par un automate fini
  - Ce langage contient tous les mots de la forme P...PQ...Q où le nombre de positions où P est vraie est égale au nombre de positions où Q est vraie. Par exemple : PPPQQQ, mais pas PPQQQQQQQQ

- Le langage où la proposition P est vraie à chaque position paire (et peut être vraie ou pas aux positions impaires) n'est pas représentable par une formule de LTL, mais bien par un automate

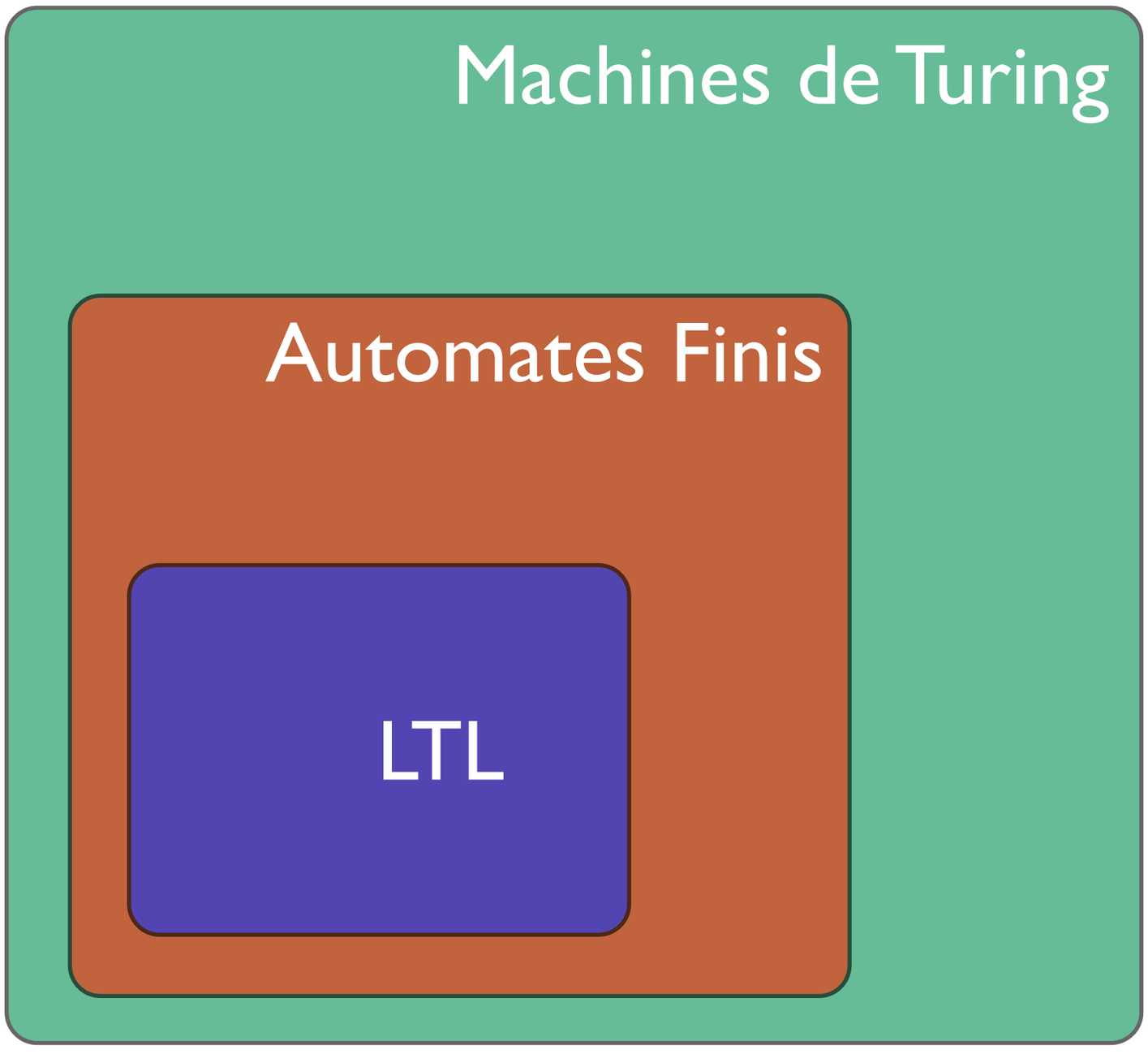
Langages réguliers

- Le langage $\{P^nQ^n \mid n \geq 1\}$ n'est pas représentable par un automate fini
  - Ce langage contient tous les mots de la forme P...PQ...Q où le nombre de positions où P est vraie est égale au nombre de positions où Q est vraie. Par exemple : PPPQQQ, mais pas PPQQQQQQQQ

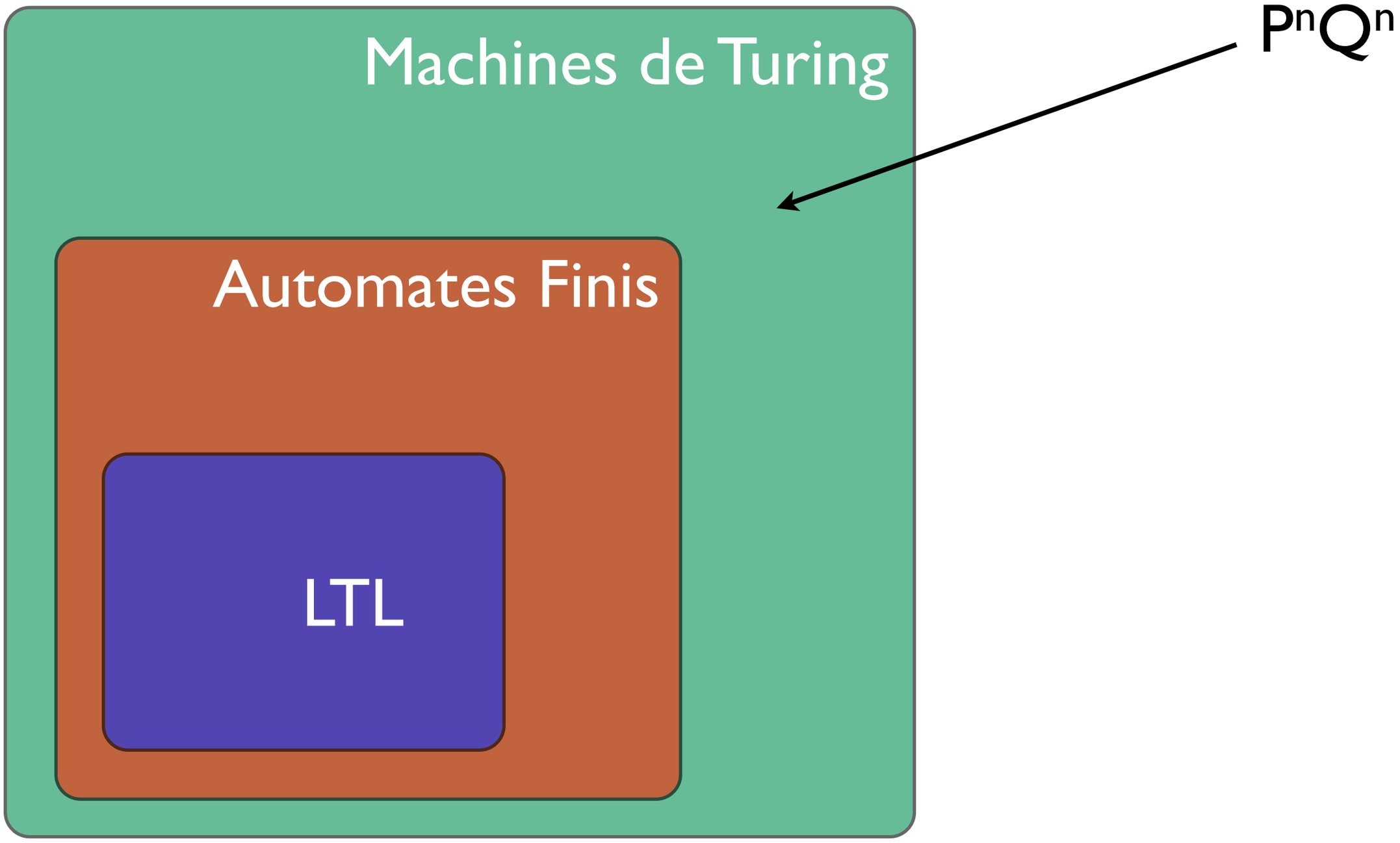
- Le langage où la proposition P est vraie à chaque position paire (et peut être vraie ou pas aux positions impaires) n'est pas représentable par une formule de LTL, mais bien par un automate



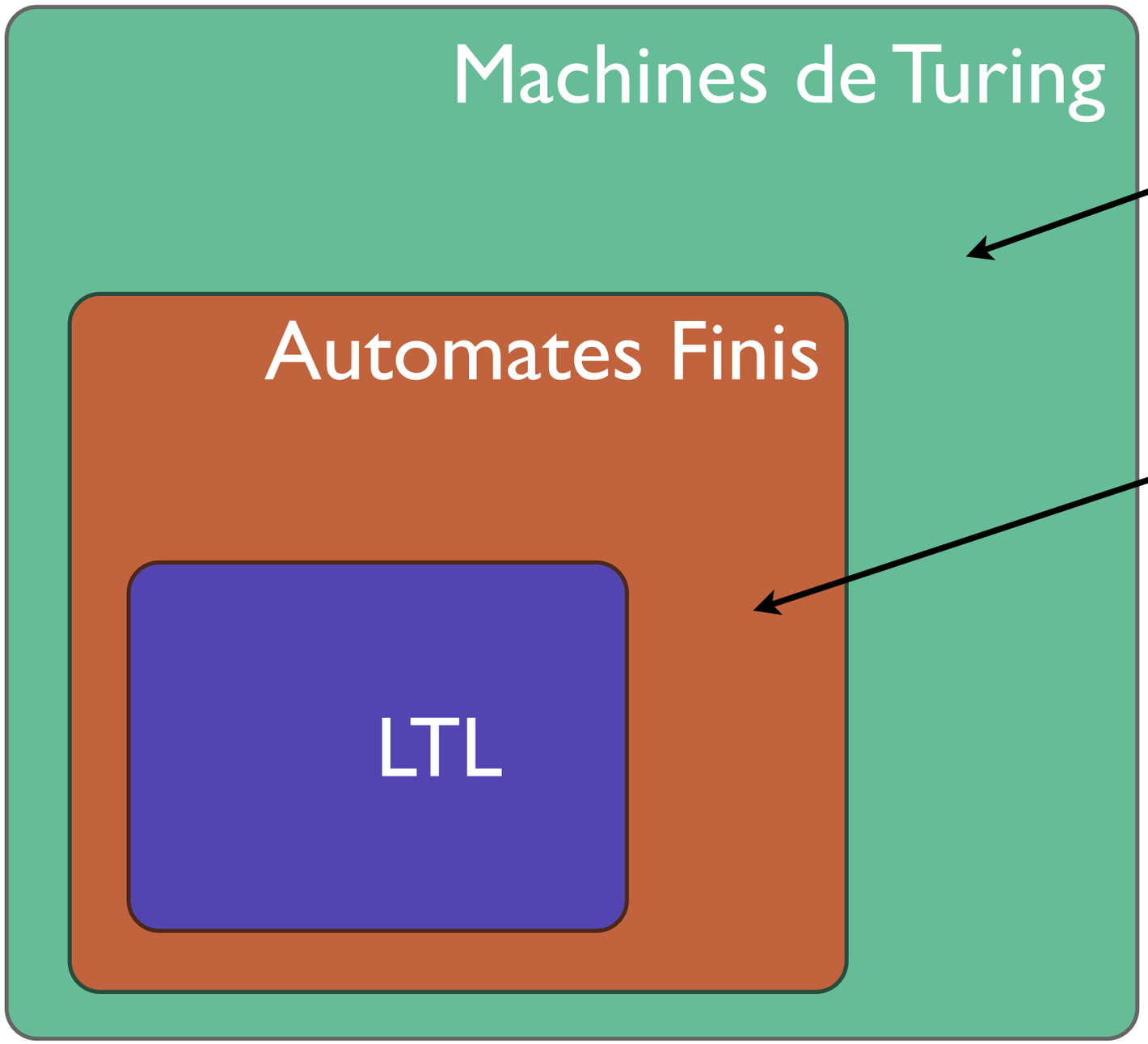
Classes de langages



Classes de langages



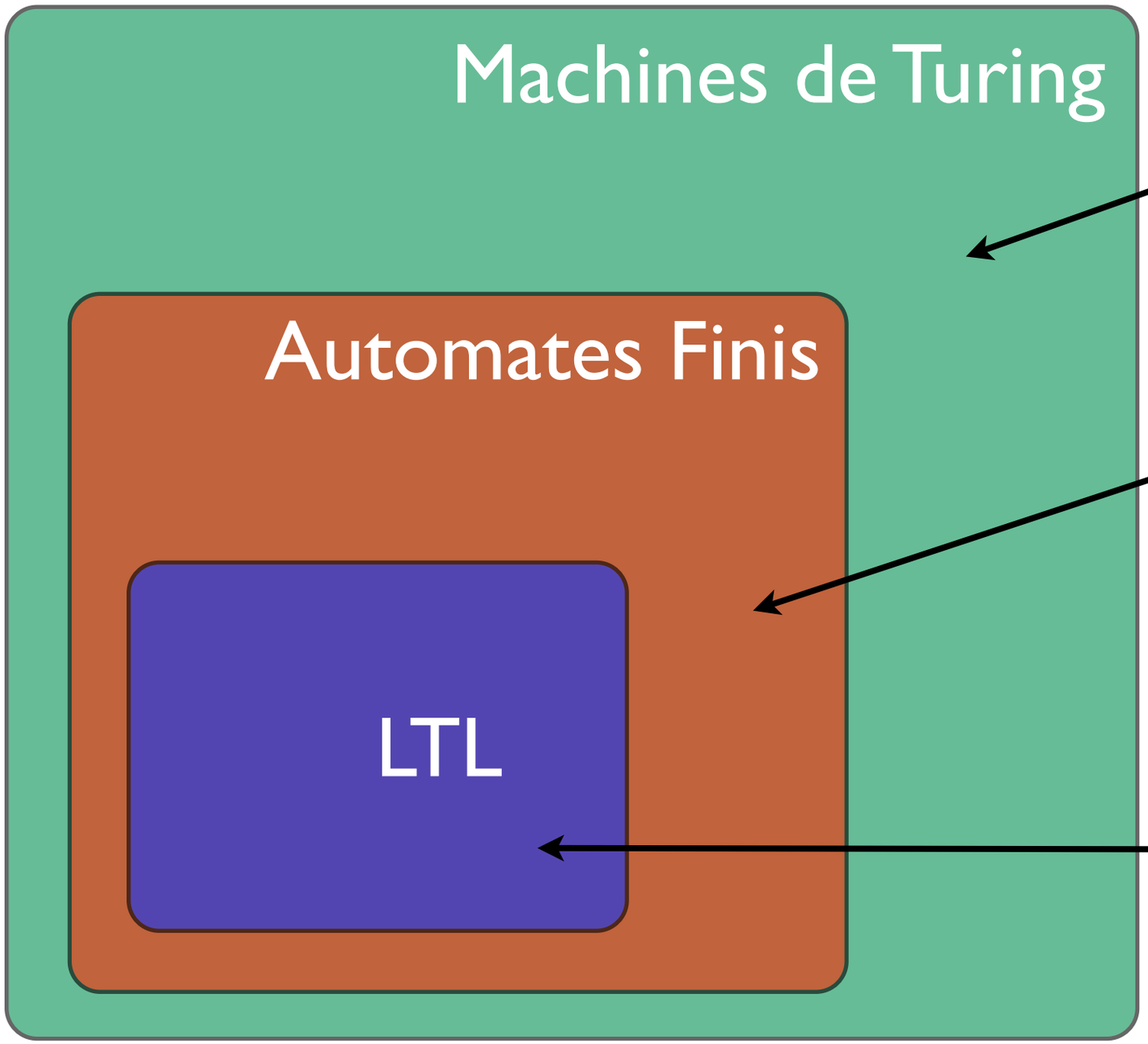
Classes de langages



P^nQ^n

«P vrai toutes les positions paires»

Classes de langages



Vérification en pratique

Vérification en pratique

- Les automates utilisés définissent des mots infinis
  - Automates de Büchi, Street, Rabin...

Vérification en pratique

- Les automates utilisés définissent des mots infinis
  - Automates de Büchi, Street, Rabin...
- Peut-on utiliser d'autres modèles plus riches ?
  - Automates temporisés, réseaux de Petri,...
  - Problème : le *model-checking* peut devenir indécidable

Vérification en pratique

- Les automates utilisés définissent des mots infinis
  - Automates de Büchi, Street, Rabin...
- Peut-on utiliser d'autres modèles plus riches ?
  - Automates temporisés, réseaux de Petri,...
  - Problème : le *model-checking* peut devenir indécidable
- Comment extraire le modèle du système informatique ?
  - Problème de recherche ouvert...

Vérification en pratique

- Les automates utilisés définissent des mots infinis
  - Automates de Büchi, Street, Rabin...
- Peut-on utiliser d'autres modèles plus riches ?
  - Automates temporisés, réseaux de Petri,...
  - Problème : le *model-checking* peut devenir indécidable
- Comment extraire le modèle du système informatique ?
  - Problème de recherche ouvert...
- Ces techniques sont de plus en plus utilisées dans l'industrie
  - Intel (vérification de circuits), Microsoft (projet SLAM),...

Questions