

Vers des logiciels sans erreurs

Alexandre Miquel
PPS

`Alexandre.Miquel@pps.jussieu.fr`

Pierre Moro
LIAFA

`Pierre.Moro@liafa.jussieu.fr`

Fête de la Science 2005
Université Paris 7 – Denis Diderot
UFR d'informatique

Ces logiciels qui nous envahissent...

Ces logiciels qui nous envahissent...

Les logiciels prennent de plus en plus de place dans notre vie

On leur confie de plus en plus de responsabilités...

... une erreur peut avoir de très lourdes conséquences

Ces logiciels qui nous envahissent...

Les logiciels prennent de plus en plus de place dans notre vie

On leur confie de plus en plus de responsabilités...

... une erreur peut avoir de très lourdes conséquences

Les logiciels **critiques**

⇒ *Ce sont les logiciels dans lesquels la moindre erreur de programmation (un **bug**) peut coûter très cher, humainement et/ou financièrement*

Ces logiciels qui nous envahissent...

Les logiciels prennent de plus en plus de place dans notre vie

On leur confie de plus en plus de responsabilités...

... une erreur peut avoir de très lourdes conséquences

Les logiciels **critiques**

⇒ *Ce sont les logiciels dans lesquels la moindre erreur de programmation (un **bug**) peut coûter très cher, humainement et/ou financièrement*

Problème : Comment concevoir des logiciels **zéro-défaut** ?

Les logiciels embarqués

On les trouve :

dans les avions, les fusées...

dans les satellites...



On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...



On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...

... mais aussi dans le métro



Les logiciels embarqués

On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...

... mais aussi dans le métro



En médecine :

Les logiciels embarqués

On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...

... mais aussi dans le métro



En médecine :

dans les pacemakers...

Les logiciels embarqués

On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...

... mais aussi dans le métro



En médecine :

dans les pacemakers...

dans les robots chirurgicaux



Les logiciels embarqués

On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...

... mais aussi dans le métro



En médecine :

dans les pacemakers...

dans les robots chirurgicaux



Et dans la vie de tous les jours :

Les logiciels embarqués

On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...

... mais aussi dans le métro



En médecine :

dans les pacemakers...

dans les robots chirurgicaux



Et dans la vie de tous les jours :

dans les téléphones mobiles, dans les cartes à puce...

On les trouve :

dans les avions, les fusées...

dans les satellites...

dans les centrales nucléaires...

... mais aussi dans le métro



En médecine :

dans les pacemakers...

dans les robots chirurgicaux



Et dans la vie de tous les jours :

dans les téléphones mobiles, dans les cartes à puce...

dans le four à micro-ondes, dans la machine à laver...

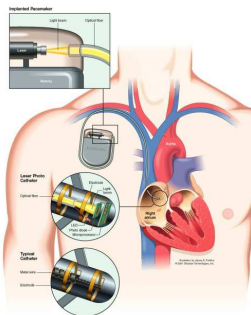
Le Pacemaker



Le Pacemaker



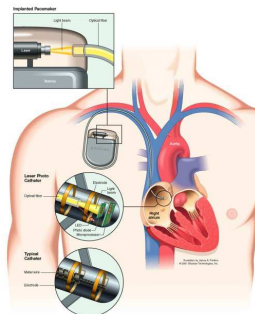
Le plus petit possible
serait intéressant
non ?



Le Pacemaker



Le plus petit possible
serait intéressant
non ?



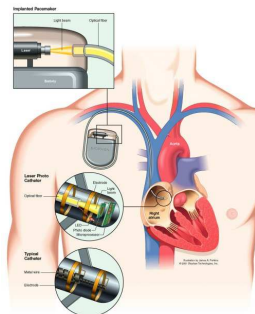
Pour modéliser les battements de coeur, 2 possibilités

- 1 256 valeurs
- 2 65536 valeurs

Le Pacemaker



Le plus petit possible
serait intéressant
non ?



Pour modéliser les battements de coeur, 2 possibilités

- 1 256 valeurs
- 2 65536 valeurs

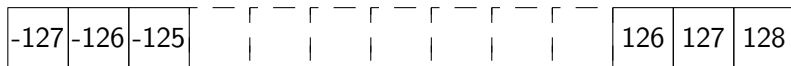
⇒ 256 est bien suffisant.

Le Pacemaker

Oui mais ...

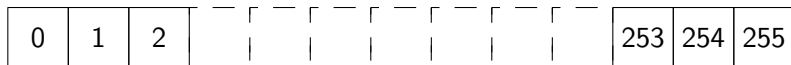


Le problème c'est que ça ne marche pas comme ça (par défaut)...

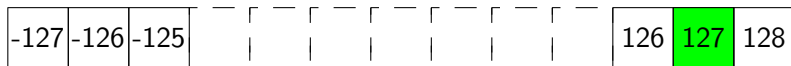


Le Pacemaker

Oui mais ...

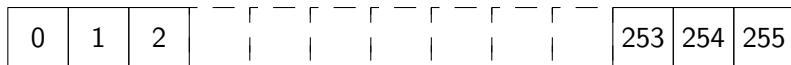


Le problème c'est que ça ne marche pas comme ça (par défaut)...

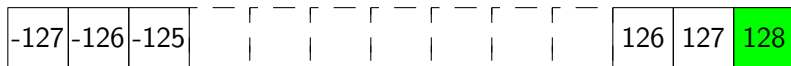


Le Pacemaker

Oui mais ...

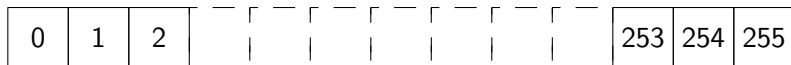


Le problème c'est que ça ne marche pas comme ça (par défaut)...

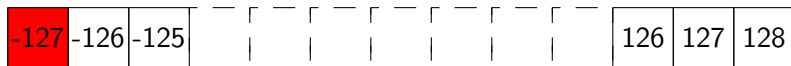


Le Pacemaker

Oui mais ...



Le problème c'est que ça ne marche pas comme ça (par défaut)...



Le programme conclut à une baisse du rythme cardiaque

et enclenche le pacemaker.

Ariane 5



Pour le lancement d'Ariane 5,

- ① 189 vols d'Ariane 4 réussis.

Ariane 5



Pour le lancement d'Ariane 5,

- 1 189 vols d'Ariane 4 réussis.
- 2 \implies On ré-utilise une partie du logiciel de lancement d'Ariane 4
- 3 On ajoute le nécessaire pour la nouvelle fusée.

Ariane 5



Pour le lancement d'Ariane 5,

- 1 189 vols d'Ariane 4 réussis.
- 2 \implies On ré-utilise une partie du logiciel de lancement d'Ariane 4
- 3 On ajoute le nécessaire pour la nouvelle fusée.



Ariane 5



Pour le lancement d'Ariane 5,

- 1 189 vols d'Ariane 4 réussis.
- 2 \implies On ré-utilise une partie du logiciel de lancement d'Ariane 4
- 3 On ajoute le nécessaire pour la nouvelle fusée.



Le logiciel **d'Ariane 4** était bugué pour des cas qui ne pouvait pas apparaître pour Ariane 4.

le bug du pentium



Au lancement de la série Pentium en 1994, le processeur du même nom, ne savait pas faire ...

le bug du pentium



Au lancement de la série Pentium en 1994, le processeur du même nom, ne savait pas faire ... **la division**

le bug du pentium



Au lancement de la série Pentium en 1994, le processeur du même nom, ne savait pas faire ... **la division**

- Optimisation \implies une erreur est apparue
- Tous les tests ont été fait avant cette dernière optimisation

Cette erreur a fait perdre 500 000 000 \$ à Intel

Le détecteur de bugs...

Le détecteur de bugs...

Et si on construisait une **machine à détecter les bugs**?

Le détecteur de bugs...

Et si on construisait une **machine à détecter les bugs**?

Une telle machine permettrait de détecter automatiquement les boucles infinies...

Le détecteur de bugs...

Et si on construisait une **machine à détecter les bugs**?

Une telle machine permettrait de détecter automatiquement les boucles infinies...

Or une telle machine n'existe pas...



Théorème de l'arrêt (Alan Turing, 1936)

Il n'existe aucun programme permettant de décider si un programme termine ou pas.

Le détecteur de bugs... n'existera jamais

Et si on construisait une **machine à détecter les bugs**?

Une telle machine permettrait de détecter automatiquement les boucles infinies...

Or une telle machine n'existe pas...



Théorème de l'arrêt (Alan Turing, 1936)

Il n'existe aucun programme permettant de décider si un programme termine ou pas.

⇒ *A fortiori*, on ne pourra jamais détecter tous les bugs (de manière entièrement automatique)

Mais au fait, qu'est-ce qu'un bug ?

C'est quand un programme ne respecte pas sa **spécification**

Mais au fait, qu'est-ce qu'un bug ?

C'est quand un programme ne respecte pas sa **spécification**

⇒ *La **spécification** d'un programme, c'est l'ensemble des propriétés que le programme doit satisfaire (du point de vue de l'utilisateur, du commanditaire)*

Mais au fait, qu'est-ce qu'un bug ?

C'est quand un programme ne respecte pas sa **spécification**

⇒ *La **spécification** d'un programme, c'est l'ensemble des propriétés que le programme doit satisfaire (du point de vue de l'utilisateur, du commanditaire)*

Le cycle de vie d'un logiciel ...

Mais au fait, qu'est-ce qu'un bug ?

C'est quand un programme ne respecte pas sa **spécification**

⇒ *La **spécification** d'un programme, c'est l'ensemble des propriétés que le programme doit satisfaire (du point de vue de l'utilisateur, du commanditaire)*

Le cycle de vie d'un logiciel ...

- 1 **L'industriel** écrit la **spécification** (le **cahier des charges**)
... et la propose au programmeur

Mais au fait, qu'est-ce qu'un bug ?

C'est quand un programme ne respecte pas sa **spécification**

⇒ *La **spécification** d'un programme, c'est l'ensemble des propriétés que le programme doit satisfaire (du point de vue de l'utilisateur, du commanditaire)*

Le cycle de vie d'un logiciel ...

- 1 **L'industriel** écrit la **spécification** (le **cahier des charges**)
... et la propose au programmeur
- 2 **Le programmeur** écrit le **logiciel** (le **programme**)
... et le fournit à l'industriel

Mais au fait, qu'est-ce qu'un bug ?

C'est quand un programme ne respecte pas sa **spécification**

⇒ *La **spécification** d'un programme, c'est l'ensemble des propriétés que le programme doit satisfaire (du point de vue de l'utilisateur, du commanditaire)*

Le cycle de vie d'un logiciel ...

- 1 **L'industriel** écrit la **spécification** (le **cahier des charges**)
... et la propose au programmeur
- 2 **Le programmeur** écrit le **logiciel** (le **programme**)
... et le fournit à l'industriel

En pratique, de nombreux allers-retours sont nécessaires

Mais au fait, qu'est-ce qu'un bug ?

C'est quand un programme ne respecte pas sa **spécification**

⇒ *La **spécification** d'un programme, c'est l'ensemble des propriétés que le programme doit satisfaire (du point de vue de l'utilisateur, du commanditaire)*

Le cycle de vie d'un logiciel ...

- 1 **L'industriel** écrit la **spécification** (le **cahier des charges**)
... et la propose au programmeur
- 2 **Le programmeur** écrit le **logiciel** (le **programme**)
... et le fournit à l'industriel

En pratique, de nombreux allers-retours sont nécessaires

Comment s'assurer que le **logiciel** est conforme à sa **spécification** ?

Suffit-il de faire des tests ?

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Plus le logiciel est critique...

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Plus le logiciel est critique...
plus le nombre de tests doit être grand...

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Plus le logiciel est critique...

plus le nombre de tests doit être grand...

et plus le choix des tests doit être soigneux

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Plus le logiciel est critique...

plus le nombre de tests doit être grand...

et plus le choix des tests doit être soigneux

Mais la batterie de tests la plus complète...

... est une goutte d'eau dans l'océan des cas possibles

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Plus le logiciel est critique...

plus le nombre de tests doit être grand...

et plus le choix des tests doit être soigneux

Mais la batterie de tests la plus complète...

... est une goutte d'eau dans l'océan des cas possibles

1 entier machine = 1 mot de 32 chiffres binaires (0 ou 1)

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Plus le logiciel est critique...

plus le nombre de tests doit être grand...

et plus le choix des tests doit être soigneux

Mais la batterie de tests la plus complète...

... est une goutte d'eau dans l'océan des cas possibles

1 entier machine = 1 mot de 32 chiffres binaires (0 ou 1)
 = 4 294 967 296 possibilités

Suffit-il de faire des tests ?

En pratique, les logiciels subissent des **séries de tests** avant d'être mis en service

La règle d'or :

Plus le logiciel est critique...

plus le nombre de tests doit être grand...

et plus le choix des tests doit être soigneux

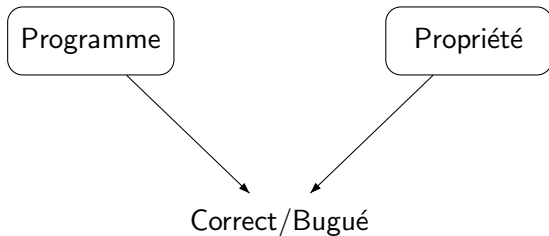
Mais la batterie de tests la plus complète...

... est une goutte d'eau dans l'océan des cas possibles

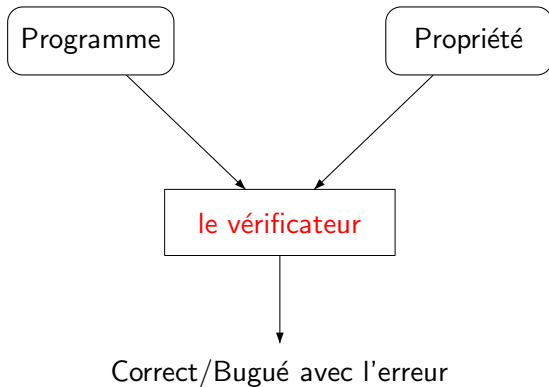
1 entier machine = 1 mot de 32 chiffres binaires (0 ou 1)
= 4 294 967 296 possibilités

2 entiers machine = 18 446 744 073 709 551 616 possibilités

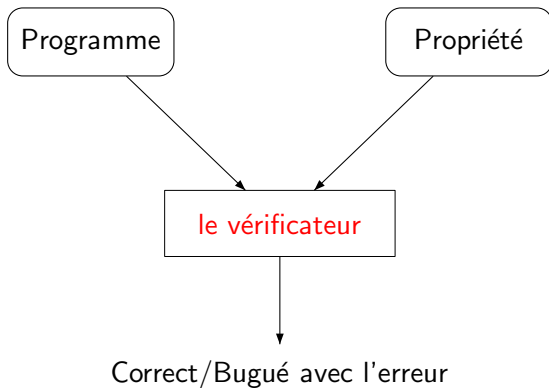
Objectif : tout vérifier automatiquement



Objectif : tout vérifier automatiquement

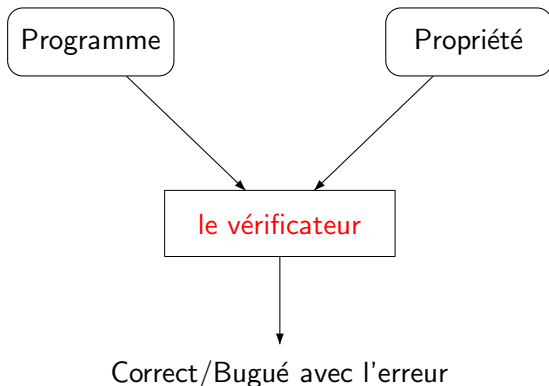


Objectif : tout vérifier automatiquement



Problème c'est impossible 😞

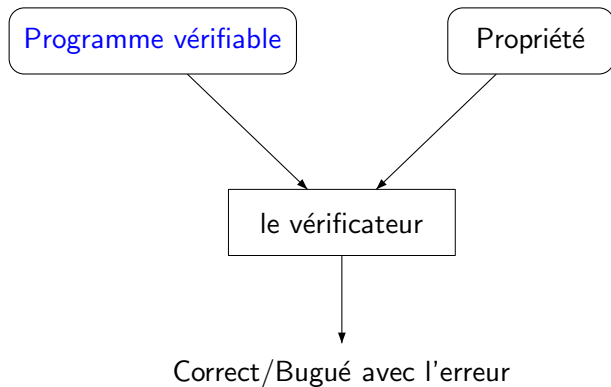
Objectif : tout vérifier automatiquement



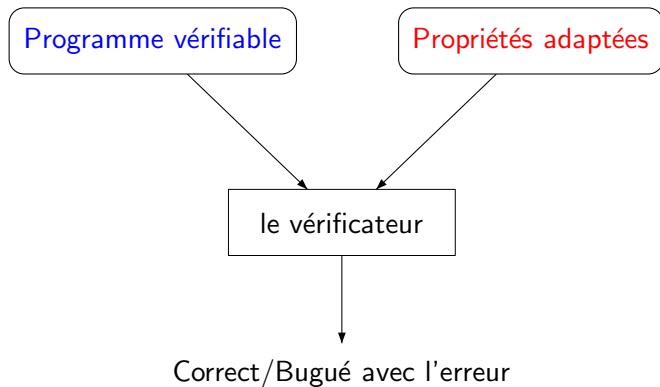
Problème c'est impossible 😞

Mais seulement dans le cas **général** 😊

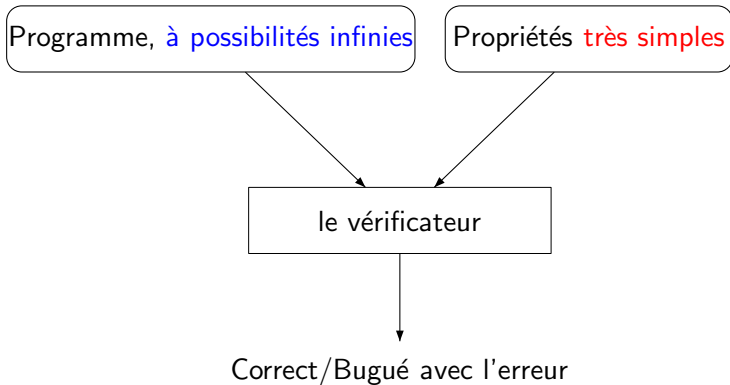
Objectif : tout vérifier automatiquement



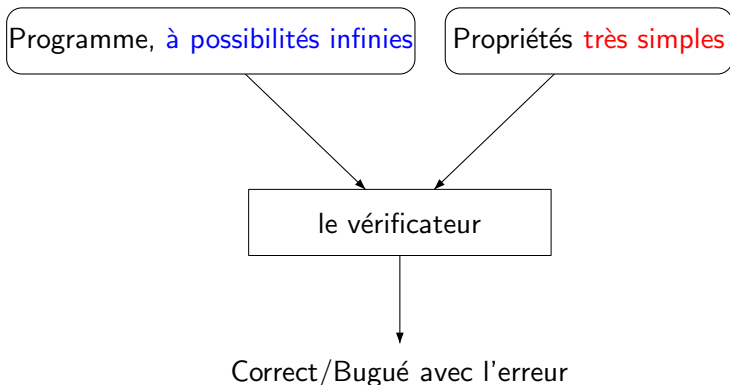
Objectif : tout vérifier automatiquement



Objectif : tout vérifier automatiquement



Objectif : tout vérifier automatiquement



- 1 Les calculs ne s'arrêtent pas nécessairement.
- 2 La simplicité des propriétés est relative, on peut faire beaucoup de choses.

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**
Mais les ressources disponibles (temps/argent) sont **finies**...

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**

Mais les ressources disponibles (temps/argent) sont **finies**...

Un problème insoluble ?

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**

Mais les ressources disponibles (temps/argent) sont **finies**...

Un problème insoluble ?

Une solution vieille de 2500 ans :

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**

Mais les ressources disponibles (temps/argent) sont **finies**...

Un problème insoluble ?

Une solution vieille de 2500 ans : **la preuve mathématique**

Un raisonnement **fini** pour traiter un nombre de cas **infini**

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**

Mais les ressources disponibles (temps/argent) sont **finies**...

Un problème insoluble ?

Une solution vieille de 2500 ans : **la preuve mathématique**

Un raisonnement **fini** pour traiter un nombre de cas **infini**

Objection :

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**

Mais les ressources disponibles (temps/argent) sont **finies**...

Un problème insoluble ?

Une solution vieille de 2500 ans : **la preuve mathématique**

Un raisonnement **fini** pour traiter un nombre de cas **infini**

Objection : Et si la preuve est fausse ?

Accéder à l'infini : un rêve impossible ?

Problème : Ensemble de situations potentiellement **infinies**

Mais les ressources disponibles (temps/argent) sont **finies**...

Un problème insoluble ?

Une solution vieille de 2500 ans : **la preuve mathématique**

Un raisonnement **fini** pour traiter un nombre de cas **infini**

Objection : Et si la preuve est fausse ?

Solution : Faire vérifier la preuve... par **une machine**

La démonstration assistée par ordinateur

Un **vérificateur de preuve** est un programme qui décide si une preuve mathématique est **correcte** ou non

La démonstration assistée par ordinateur

Un **vérificateur de preuve** est un programme qui décide si une preuve mathématique est **correcte** ou non

Principe de fonctionnement :

La démonstration assistée par ordinateur

Un **vérificateur de preuve** est un programme qui décide si une preuve mathématique est **correcte** ou non

Principe de fonctionnement :

En entrée : une formule mathématique + un **texte de preuve**

La démonstration assistée par ordinateur

Un **vérificateur de preuve** est un programme qui décide si une preuve mathématique est **correcte** ou non

Principe de fonctionnement :

En entrée : une formule mathématique + un **texte de preuve**

En sortie : une réponse de type : *Correct/Incorrect*

La démonstration assistée par ordinateur

Un **vérificateur de preuve** est un programme qui décide si une preuve mathématique est **correcte** ou non

Principe de fonctionnement :

En entrée : une formule mathématique + un **texte de preuve**

En sortie : une réponse de type : *Correct/Incorrect*

Exemple : Le système Coq (<http://coq.inria.fr>)

La démonstration assistée par ordinateur

Un **vérificateur de preuve** est un programme qui décide si une preuve mathématique est **correcte** ou non

Principe de fonctionnement :

En entrée : une formule mathématique + un **texte de preuve**

En sortie : une réponse de type : *Correct/Incorrect*

Exemple : Le système Coq (<http://coq.inria.fr>)

- Il faut définir un langage de preuves

La démonstration assistée par ordinateur

Un **vérificateur de preuve** est un programme qui décide si une preuve mathématique est **correcte** ou non

Principe de fonctionnement :

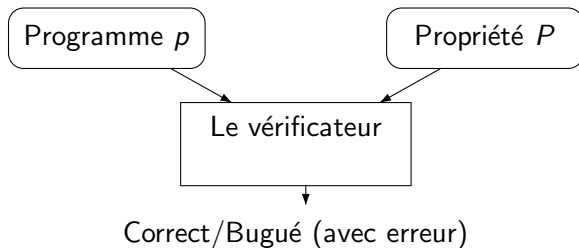
En entrée : une formule mathématique + un **texte de preuve**

En sortie : une réponse de type : *Correct/Incorrect*

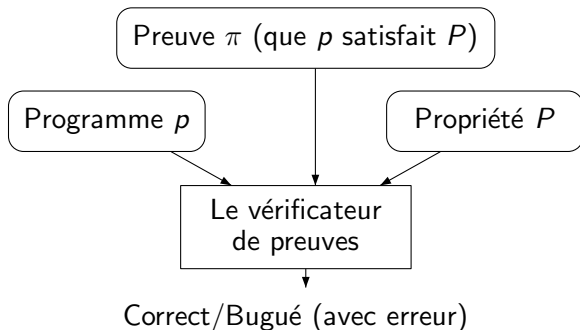
Exemple : Le système Coq (<http://coq.inria.fr>)

- Il faut définir un langage de preuves
- On peut automatiser la détection d'erreurs dans les preuves (... mais pas dans les programmes)

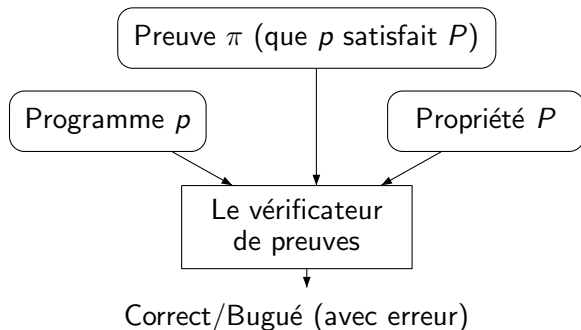
La preuve de programme



La preuve de programme

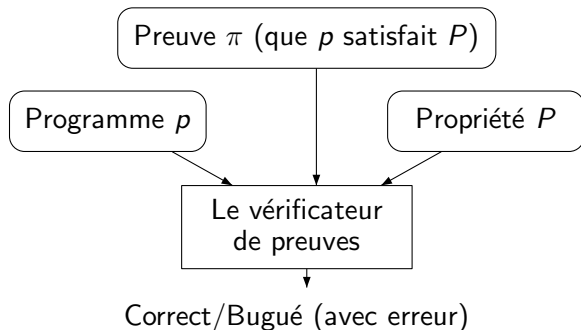


La preuve de programme



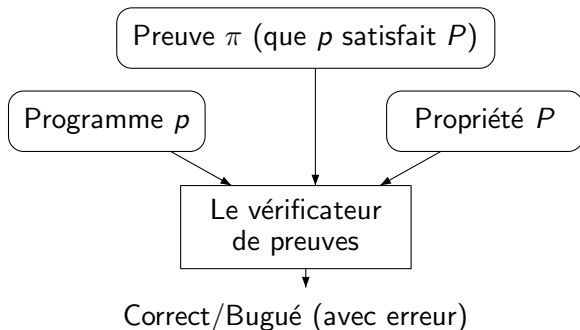
- Le vérificateur teste **la correction de la preuve** (et indirectement, la correction du programme)

La preuve de programme



- Le vérificateur teste **la correction de la preuve** (et indirectement, la correction du programme)
- Plus aucune restriction sur le programme p ou la propriété P

La preuve de programme



- Le vérificateur teste **la correction de la preuve** (et indirectement, la correction du programme)
- Plus aucune restriction sur le programme p ou la propriété P
- Mais l'utilisateur doit lui-même **construire la preuve π**