Machines de Turing et récursivité

Jérémie Cabessa

University Paris 2, France

17 Septembre 2015

Plan

- Contexte historique
- Machines de Turing
- $\mathbf{P} \neq \mathbf{NP}$?
- Indécidabilité et problème de l'arrêt
- Entscheidungsproblem de Hilbert
- Conclusion

Plan

- Contexte historique
- Machines de Turing
- $\mathbf{P} \neq \mathbf{NP}$?
- Indécidabilité et problème de l'arrêt
- Entscheidungsproblem de Hilbert
- Conclusion

Plan

- Contexte historique
- Machines de Turing
- $\mathbf{P} \neq \mathbf{NP}$?
- Indécidabilité et problème de l'arrêt
- Entscheidungsproblem de Hilbert
- Conclusion

Plan

- Contexte historique
- Machines de Turing
- $\mathbf{P} \neq \mathbf{NP}$?
- Indécidabilité et problème de l'arrêt
- Entscheidungsproblem de Hilbert
- Conclusion

Plan

- Contexte historique
- Machines de Turing
- $\mathbf{P \neq NP}$?
- Indécidabilité et problème de l'arrêt
- Entscheidungsproblem de Hilbert
- Conclusion

Plan

- Contexte historique
- Machines de Turing
- $\mathbf{P \neq NP}$?
- Indécidabilité et problème de l'arrêt
- Entscheidungsproblem de Hilbert
- Conclusion

Contexte historique

- En 1928, Hilbert et Ackermann proposent un fameux problème de décision, connu sous le nom de *Entscheidungsproblem*.

  > *"Das Entscheidungsproblem is gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt."*

- *Entscheidungsproblem :* étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, existe-t-il une « *procédure effective* » (*Verfahren*) qui détermine en un temps fini si φ est prouvable à partir de Γ ou non ?

Contexte historique

- En 1928, Hilbert et Ackermann proposent un fameux problème de décision, connu sous le nom de *Entscheidungsproblem*.

  > *"Das Entscheidungsproblem is gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt."*

- *Entscheidungsproblem :* étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, existe-t-il une « *procédure effective* » (*Verfahren*) qui détermine en un temps fini si φ est prouvable à partir de Γ ou non ?

Contexte historique

- En 1928, Hilbert et Ackermann proposent un fameux problème de décision, connu sous le nom de *Entscheidungsproblem*.

  > *"Das Entscheidungsproblem is gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt."*

- *Entscheidungsproblem :* étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, existe-t-il une « *procédure effective* » (*Verfahren*) qui détermine en un temps fini si φ est prouvable à partir de Γ ou non ?

Contexte historique

- Une telle « procédure effective » (algorithme) serait extrêmement pertinente ; par exemple :
- en prenant Γ comme une axiomatisation de l'arithmétique et φ comme étant la conjecture de Goldbach

Tout nombre entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers.

$$\forall k > 1 \; \exists p, q \; (2k = p + q \wedge prime(p) \wedge prime(q))$$

- alors la « procédure effective » de décision appliquée à Γ et φ serait capable de déterminer si la conjecture de Goldbach et « vraie » ou non, ou plutôt, prouvable ou non !
- La conjecture de Goldbach est toujours non démontrée...

Contexte historique

- Une telle « procédure effective » (algorithme) serait extrêmement pertinente ; par exemple :
- en prenant Γ comme une axiomatisation de l'arithmétique et φ comme étant la conjecture de Goldbach

  Tout nombre entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers.

  $$\forall k > 1 \; \exists p, q \; (2k = p + q \wedge prime(p) \wedge prime(q))$$

- alors la « procédure effective » de décision appliquée à Γ et φ serait capable de déterminer si la conjecture de Goldbach et « vraie » ou non, ou plutôt, prouvable ou non !
- La conjecture de Goldbach est toujours non démontrée...

Contexte historique

- Une telle « procédure effective » (algorithme) serait extrêmement pertinente ; par exemple :
- en prenant Γ comme une axiomatisation de l'arithmétique et φ comme étant la conjecture de Goldbach

  Tout nombre entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers.

  $$\forall k > 1 \; \exists p, q \; (2k = p + q \wedge prime(p) \wedge prime(q))$$

- alors la « procédure effective » de décision appliquée à Γ et φ serait capable de déterminer si la conjecture de Goldbach et « vraie » ou non, ou plutôt, prouvable ou non !
- La conjecture de Goldbach est toujours non démontrée...

Contexte historique

- Une telle « procédure effective » (algorithme) serait extrêmement pertinente ; par exemple :
- en prenant Γ comme une axiomatisation de l'arithmétique et φ comme étant la conjecture de Goldbach

  Tout nombre entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers.

  $$\forall k > 1 \; \exists p, q \; (2k = p + q \wedge prime(p) \wedge prime(q))$$

- alors la « procédure effective » de décision appliquée à Γ et φ serait capable de déterminer si la conjecture de Goldbach et « vraie » ou non, ou plutôt, prouvable ou non !
- La conjecture de Goldbach est toujours non démontrée...

Contexte historique

- Une telle « procédure effective » (algorithme) serait extrêmement pertinente ; par exemple :
- en prenant Γ comme une axiomatisation de l'arithmétique et φ comme étant la conjecture de Goldbach

  Tout nombre entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers.

  $$\forall k > 1 \; \exists p, q \; (2k = p + q \wedge prime(p) \wedge prime(q))$$

- alors la « procédure effective » de décision appliquée à Γ et φ serait capable de déterminer si la conjecture de Goldbach et « vraie » ou non, ou plutôt, prouvable ou non !
- La conjecture de Goldbach est toujours non démontrée...

Contexte historique

Idée générale de Hilbert

Proposer une « mécanisation effective » du raisonnement mathématique, du concept de preuve.

- Ceci a donné lieu à toute la « théorie de la démonstration », avec une multitude d'applications de nos jours...

Contexte historique

- Le concept de *machine de Turing* fut introduit par Alan Turing en 1936 afin de rendre compte d'une telle notion de « calculabilité effective » (computability) en mathématiques et non en informatique (pas de sens à ce moment de l'histoire).
- Autrement dit, une *machine de Turing* est un modèle de calcul qui permettrait de rendre compte du comportement d'une *personne qui calcule* – *a computer*.
- Selon Turing, les processus en oeuvre inhérents à toute activité (humaine) de calcul se réduisent au fonctionnement d'une *machine de Turing*.

Contexte historique

- Le concept de *machine de Turing* fut introduit par Alan Turing en 1936 afin de rendre compte d'une telle notion de « calculabilité effective » (computability) en mathématiques et non en informatique (pas de sens à ce moment de l'histoire).
- Autrement dit, une *machine de Turing* est un modèle de calcul qui permettrait de rendre compte du comportement d'une *personne qui calcule* – *a computer*.
- Selon Turing, les processus en oeuvre inhérents à toute activité (humaine) de calcul se réduisent au fonctionnement d'une *machine de Turing*.

Contexte historique

- Le concept de *machine de Turing* fut introduit par Alan Turing en 1936 afin de rendre compte d'une telle notion de « calculabilité effective » (computability) en mathématiques et non en informatique (pas de sens à ce moment de l'histoire).
- Autrement dit, une *machine de Turing* est un modèle de calcul qui permettrait de rendre compte du comportement d'une *personne qui calcule* – *a computer*.
- Selon Turing, les processus en oeuvre inhérents à toute activité (humaine) de calcul se réduisent au fonctionnement d'une *machine de Turing*.

Contexte historique

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

Remarque préliminaire

- Un *mot* est une suite finie de symboles sur un alphabet donné.
- Un *langage* un ensemble de mots sur un alphabet donné.
- En mathématiques, un *problème* peut être identifié avec un *langage* : l'ensemble des « mots-solutions » de ce problème.
- Le *problème* P de savoir si un graphe est connexe s'identifie au *langage* $L_P = \{\langle G \rangle : G \text{ est un graphe connexe}\}$, où $\langle . \rangle$ est une fonction de codage sur un alphabet donné.

Problème P		**Langage** L_P
G est-il connexe ?	*équivalent*	$\langle G \rangle \in L_P$?

- Dans la suite, on va considérer des *langages*, mais cela revient à considérer des *problèmes*.

Remarque préliminaire

- Un *mot* est une suite finie de symboles sur un alphabet donné.
- Un *langage* un ensemble de mots sur un alphabet donné.
- En mathématiques, un *problème* peut être identifié avec un *langage* : l'ensemble des « mots-solutions » de ce problème.
- Le *problème* P de savoir si un graphe est connexe s'identifie au *langage* $L_P = \{\langle G \rangle : G \text{ est un graphe connexe}\}$, où $\langle . \rangle$ est une fonction de codage sur un alphabet donné.

Problème P		**Langage** L_P
G est-il connexe ?	*équivalent*	$\langle G \rangle \in L_P$?

- Dans la suite, on va considérer des *langages*, mais cela revient à considérer des *problèmes*.

Remarque préliminaire

- Un *mot* est une suite finie de symboles sur un alphabet donné.
- Un *langage* un ensemble de mots sur un alphabet donné.
- En mathématiques, un *problème* peut être identifié avec un *langage* : l'ensemble des « mots-solutions » de ce problème.
- Le *problème* P de savoir si un graphe est connexe s'identifie au *langage* $L_P = \{\langle G \rangle : G \text{ est un graphe connexe}\}$, où $\langle . \rangle$ est une fonction de codage sur un alphabet donné.

Problème P		**Langage** L_P
G est-il connexe ?	*équivalent*	$\langle G \rangle \in L_P$?

- Dans la suite, on va considérer des *langages*, mais cela revient à considérer des *problèmes*.

Remarque préliminaire

- Un *mot* est une suite finie de symboles sur un alphabet donné.
- Un *langage* un ensemble de mots sur un alphabet donné.
- En mathématiques, un *problème* peut être identifié avec un *langage* : l'ensemble des « mots-solutions » de ce problème.
- Le *problème* P de savoir si un graphe est connexe s'identifie au *langage* $L_P = \{\langle G \rangle : G \text{ est un graphe connexe}\}$, où $\langle . \rangle$ est une fonction de codage sur un alphabet donné.

Problème P		**Langage** L_P
G est-il connexe ?	*équivalent*	$\langle G \rangle \in L_P$?

- Dans la suite, on va considérer des *langages*, mais cela revient à considérer des *problèmes*.

Remarque préliminaire

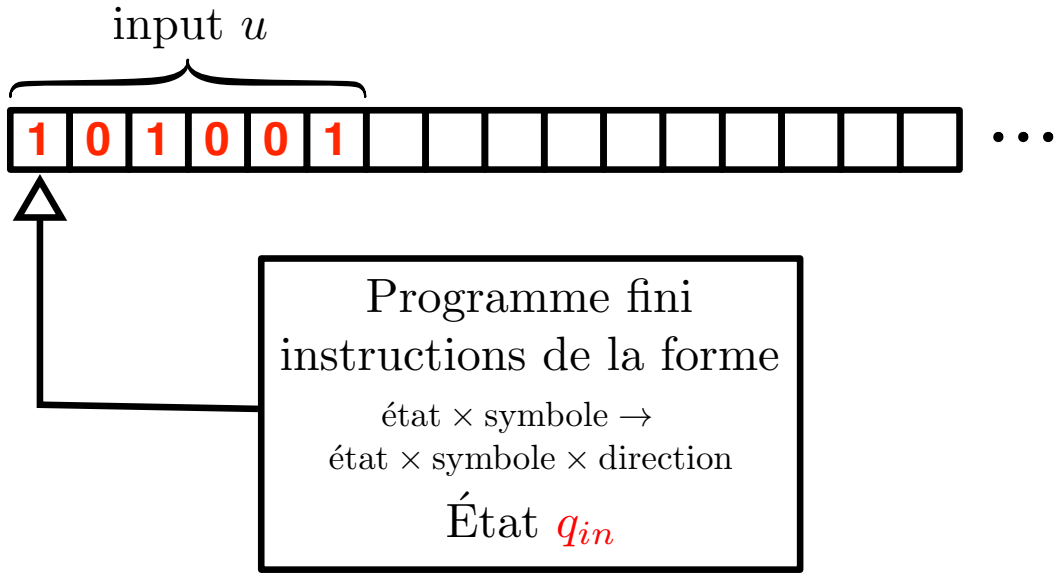
- Un *mot* est une suite finie de symboles sur un alphabet donné.
- Un *langage* un ensemble de mots sur un alphabet donné.
- En mathématiques, un *problème* peut être identifié avec un *langage* : l'ensemble des « mots-solutions » de ce problème.
- Le *problème* P de savoir si un graphe est connexe s'identifie au *langage* $L_P = \{\langle G \rangle : G \text{ est un graphe connexe}\}$, où $\langle . \rangle$ est une fonction de codage sur un alphabet donné.

Problème P		**Langage** L_P
G est-il connexe ?	*équivalent*	$\langle G \rangle \in L_P$?

- Dans la suite, on va considérer des *langages*, mais cela revient à considérer des *problèmes*.

Machine de Turing

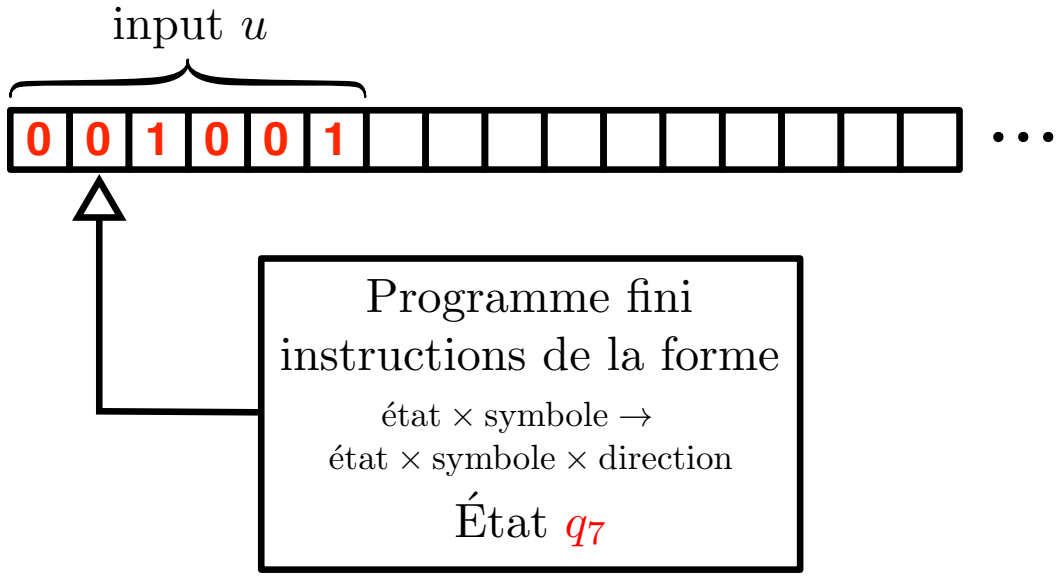
Une *machine de Turing* est un modèle de calcul abstrait qui se présente comme suit :



- l'input u est *accepté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{rej}
- l'input u est ni accepté ni rejeté par $\mathcal{M}$ si $\mathcal{M}(u)$ n'atteint aucun de ces états spéciaux

Machine de Turing

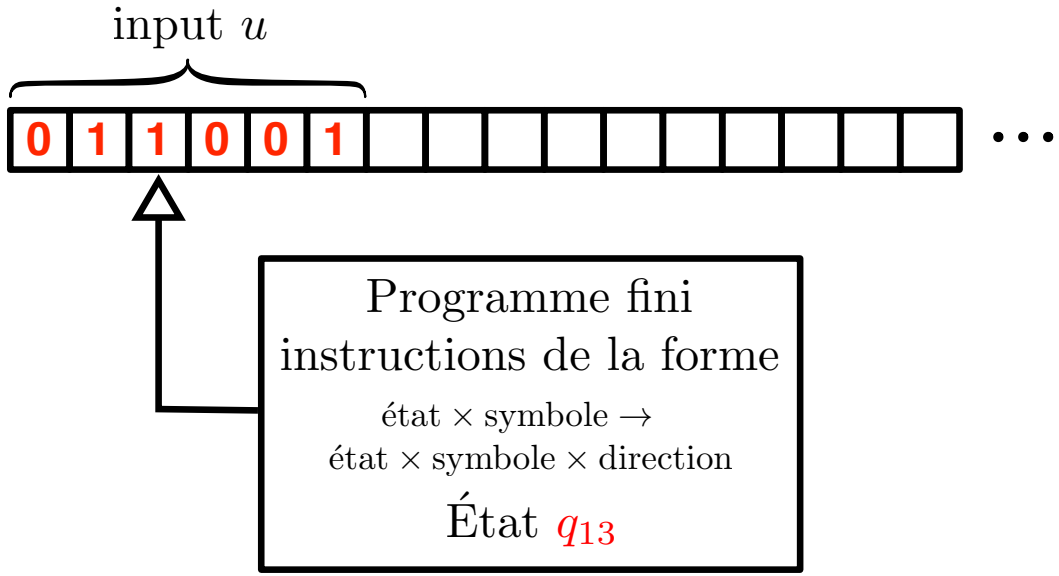
Une *machine de Turing* est un modèle de calcul abstrait qui se présente comme suit :



- l'input u est *accepté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{rej}
- l'input u est ni accepté ni rejeté par $\mathcal{M}$ si $\mathcal{M}(u)$ n'atteint aucun de ces états spéciaux

Machine de Turing

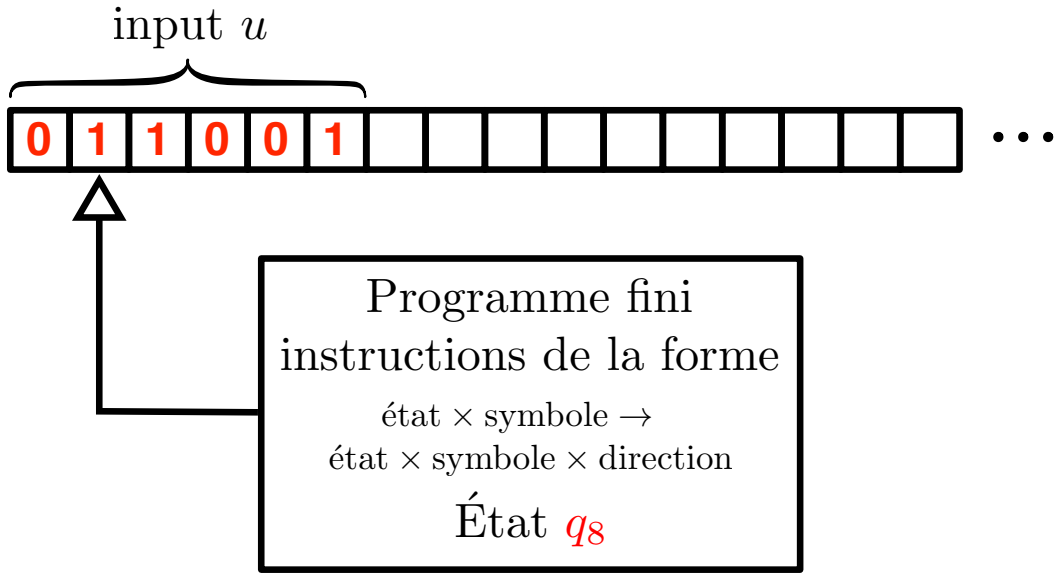
Une *machine de Turing* est un modèle de calcul abstrait qui se présente comme suit :



- l'input u est *accepté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{rej}
- l'input u est ni accepté ni rejeté par $\mathcal{M}$ si $\mathcal{M}(u)$ n'atteint aucun de ces états spéciaux

Machine de Turing

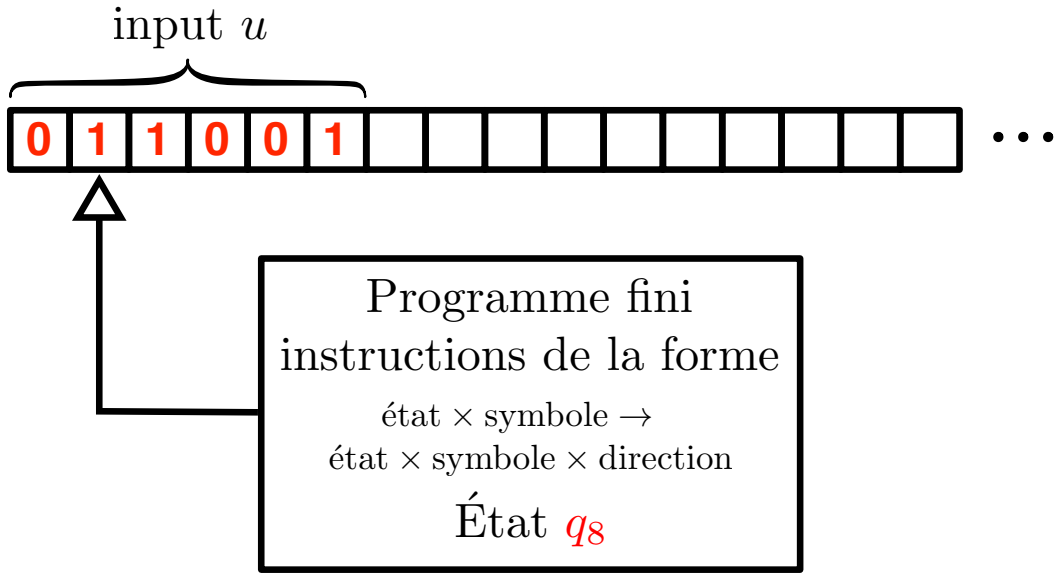
Une *machine de Turing* est un modèle de calcul abstrait qui se présente comme suit :



- l'input u est *accepté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{rej}
- l'input u est ni accepté ni rejeté par $\mathcal{M}$ si $\mathcal{M}(u)$ n'atteint aucun de ces états spéciaux

Machine de Turing

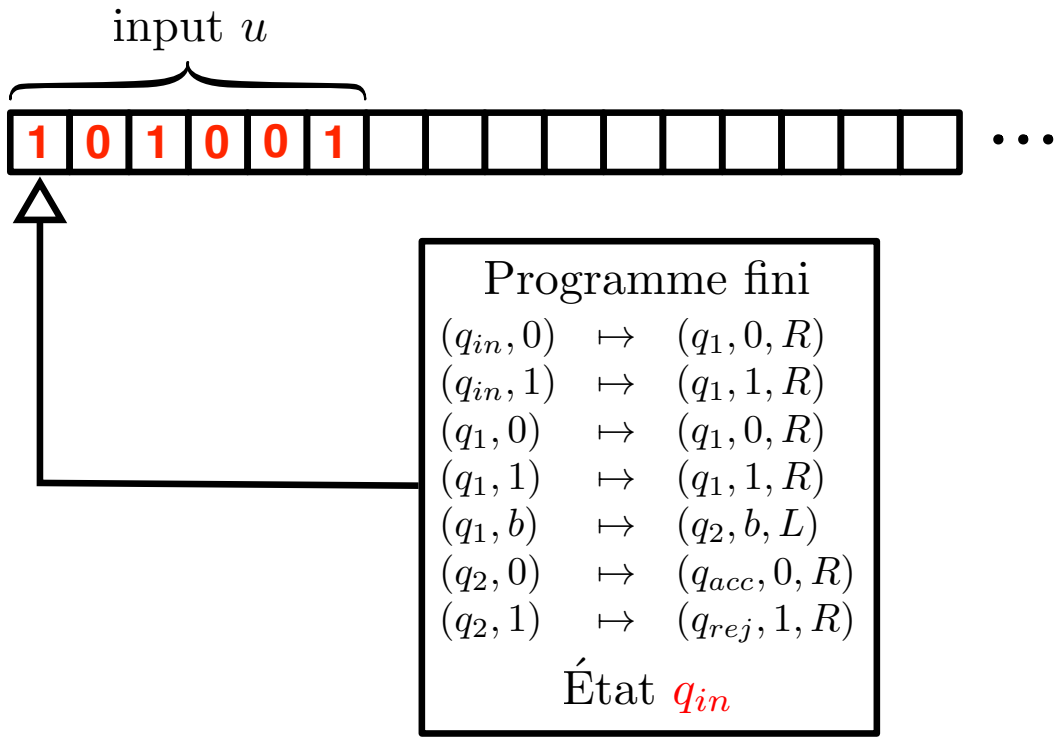
Une *machine de Turing* est un modèle de calcul abstrait qui se présente comme suit :



- l'input u est *accepté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si $\mathcal{M}(u)$ atteint l'état spécial q_{rej}
- l'input u est ni accepté ni rejeté par $\mathcal{M}$ si $\mathcal{M}(u)$ n'atteint aucun de ces états spéciaux

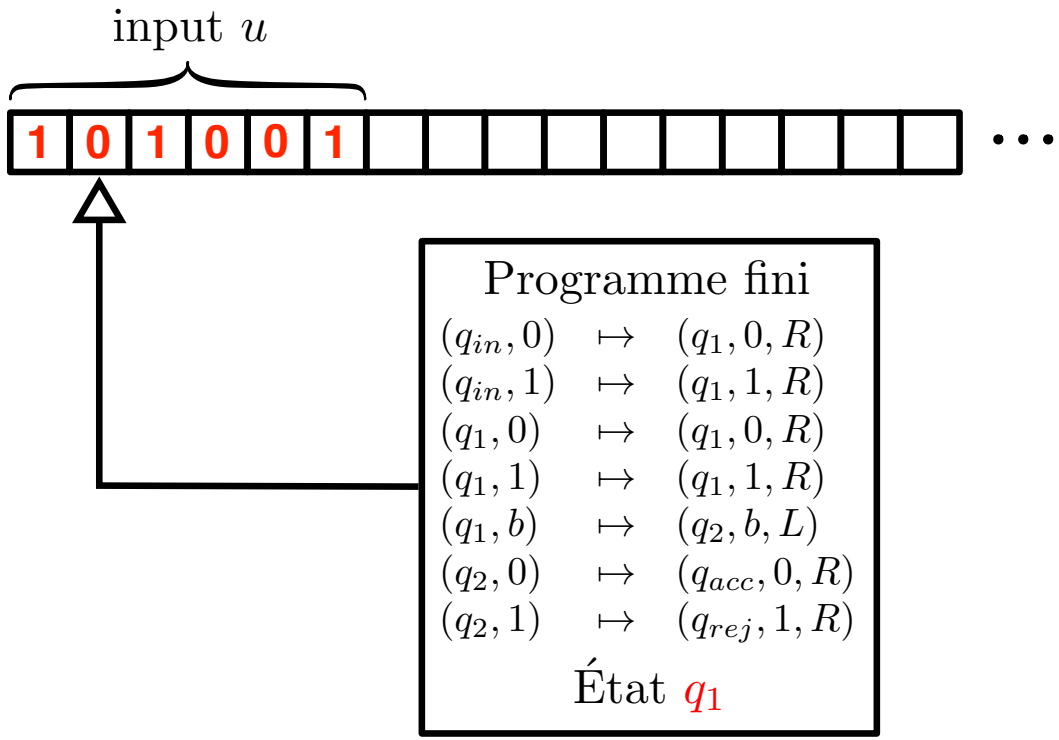
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



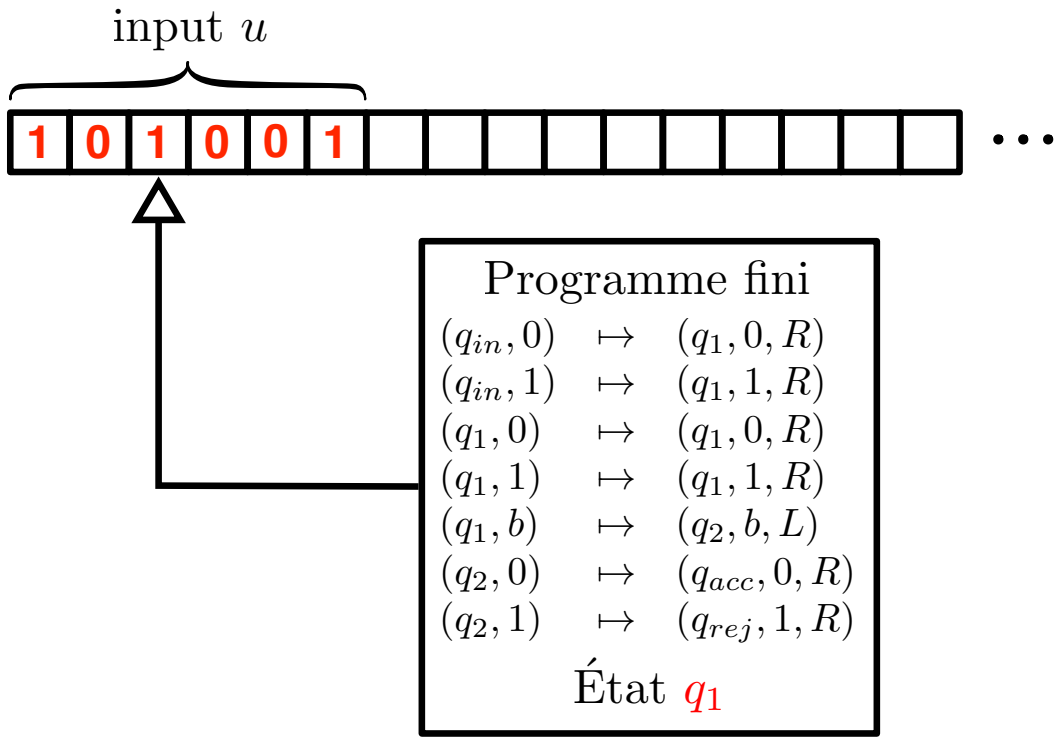
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



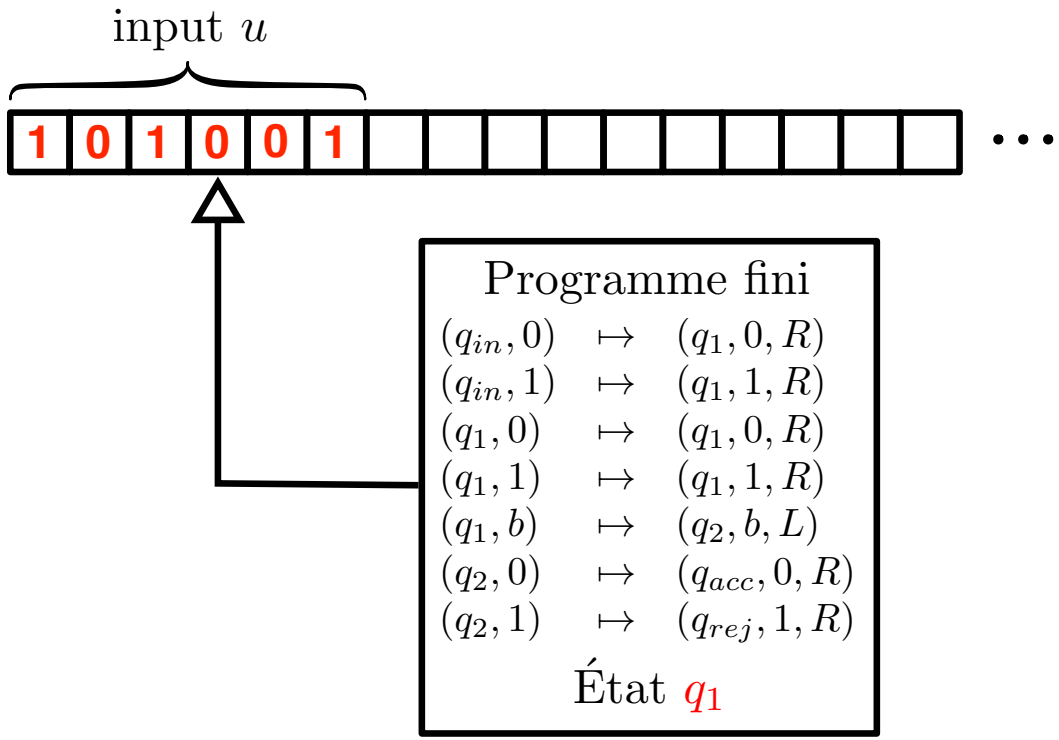
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



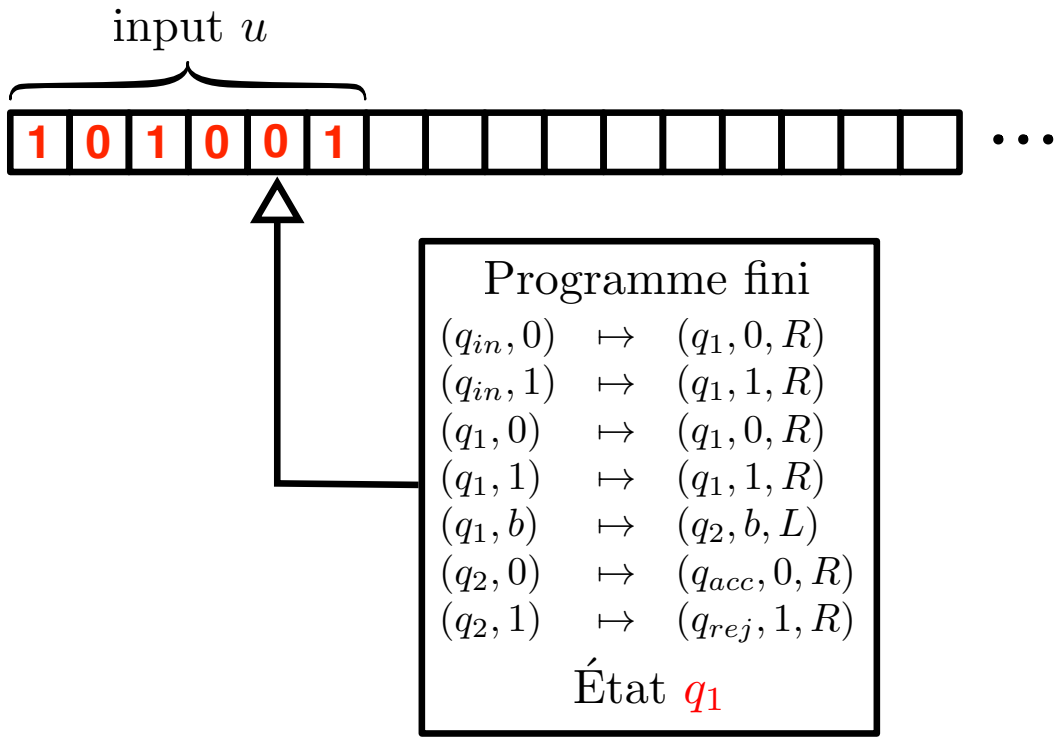
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



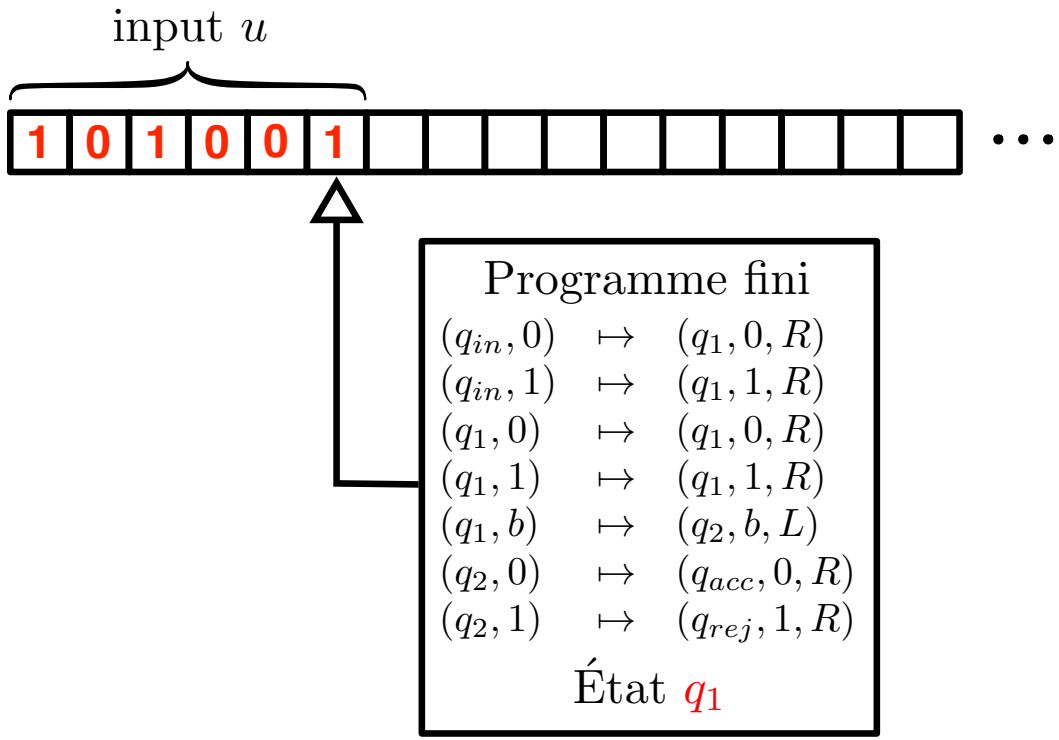
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



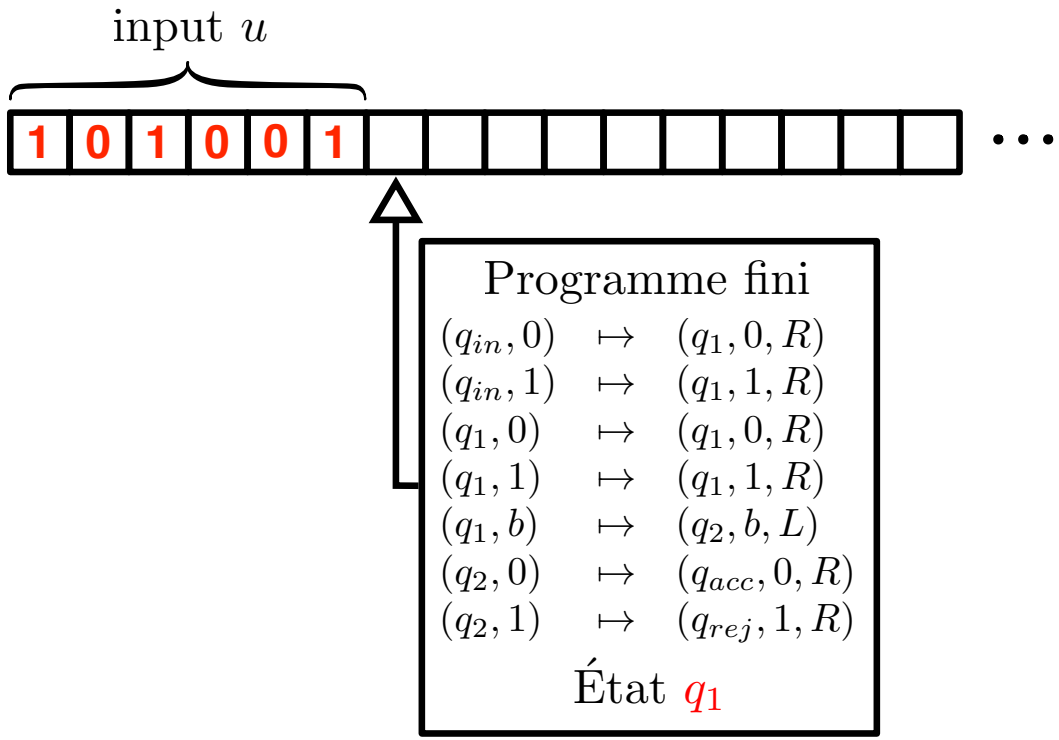
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



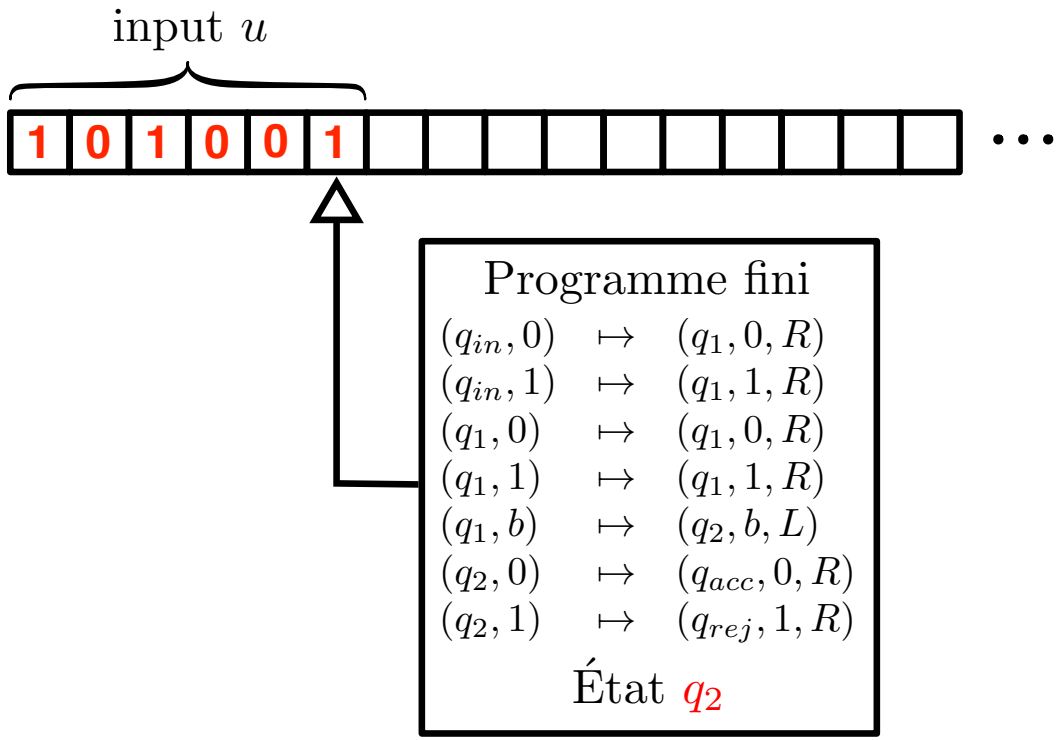
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



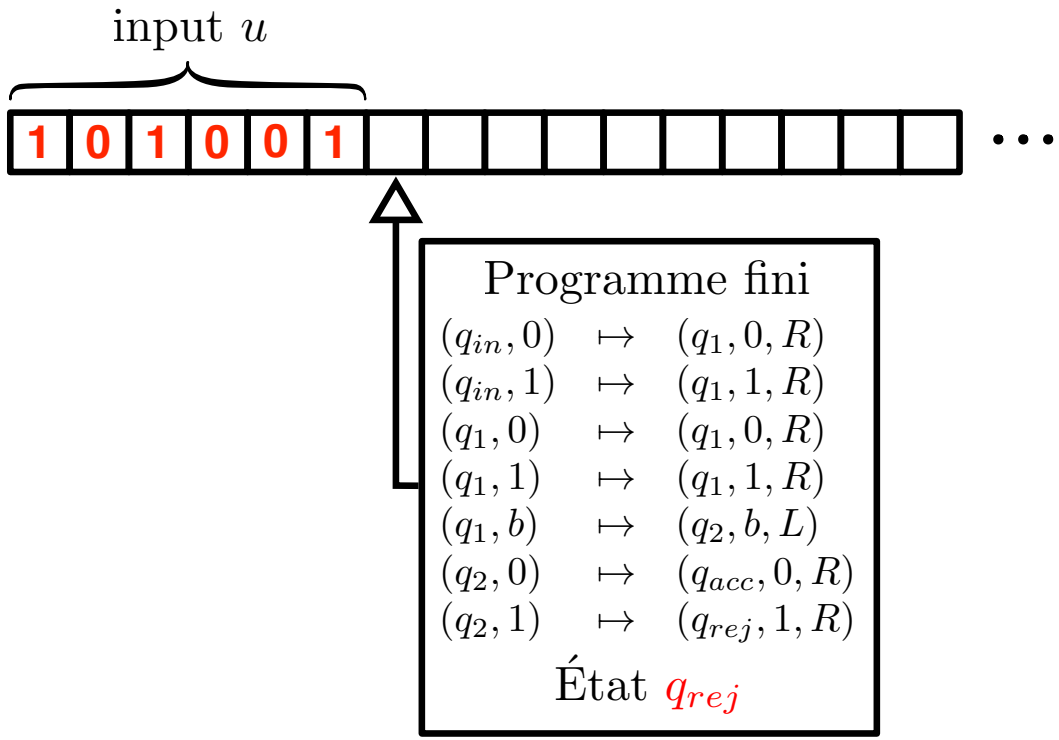
Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



Machine de Turing

- La machine de Turing ci-dessous décide si l'input se termine par 0 ou non, i.e., si c'est un nombre pair ou non.



Machine de Turing

▸ Link to the video

Machine de Turing

Définition 1 (Machine de Turing)

Une *machine de Turing* déterministe est un tuple fini $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition
- $q_{in} \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing

Définition 1 (Machine de Turing)

Une *machine de Turing* déterministe est un tuple fini $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition
- $q_{in} \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing

Définition 1 (Machine de Turing)

Une *machine de Turing* déterministe est un tuple fini $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition
- $q_{in} \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing

Définition 1 (Machine de Turing)

Une *machine de Turing* déterministe est un tuple fini $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition
- $q_{in} \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing

Définition 1 (Machine de Turing)

Une *machine de Turing* déterministe est un tuple fini $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition
- $q_{in} \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing

Définition 1 (Machine de Turing)

Une *machine de Turing* déterministe est un tuple fini $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition
- $q_{in} \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing

Définition 1 (Machine de Turing)

Une *machine de Turing* déterministe est un tuple fini $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition
- $q_{in} \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing

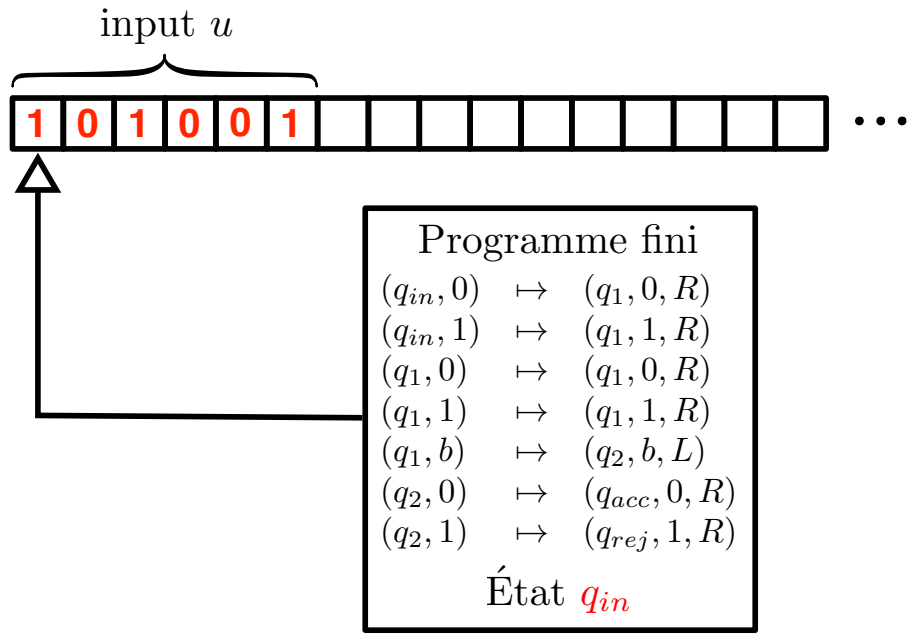
- Si $\mathcal{M}$ est une machine de Turing, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage de* $\mathcal{M}$ ou *langage reconnu par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.
- Le langage de la machine de Turing $\mathcal{M}$ ci-dessous est

$$L(\mathcal{M}) = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}.$$

Machine de Turing

- Si $\mathcal{M}$ est une machine de Turing, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage de* $\mathcal{M}$ ou *langage reconnu par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.
- Le langage de la machine de Turing $\mathcal{M}$ ci-dessous est

$$L(\mathcal{M}) = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}.$$



Machine de Turing

Définition 2 (Reconnaissabilité ou Réc. énum.)

Un langage X est dit *Turing-reconnaissable* ou *récursivement énumérable* s'il existe une machine de Turing $\mathcal{M}$ qui le reconnait, i.e., s'il existe $\mathcal{M}$ telle que $L(\mathcal{M}) = X$.

- Notre exemple précédent montre que le langage $X = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$ est Turing-reconnaissable, puisqu'on a trouvé une machine $\mathcal{M}$ qui reconnait L.

Machine de Turing

Définition 2 (Reconnaissabilité ou Réc. énum.)

Un langage X est dit *Turing-reconnaissable* ou *récursivement énumérable* s'il existe une machine de Turing $\mathcal{M}$ qui le reconnait, i.e., s'il existe $\mathcal{M}$ telle que $L(\mathcal{M}) = X$.

- Notre exemple précédent montre que le langage $X = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$ est Turing-reconnaissable, puisqu'on a trouvé une machine $\mathcal{M}$ qui reconnait L.

Machine de Turing

Démo simulateur :

▸ Link to the simulator

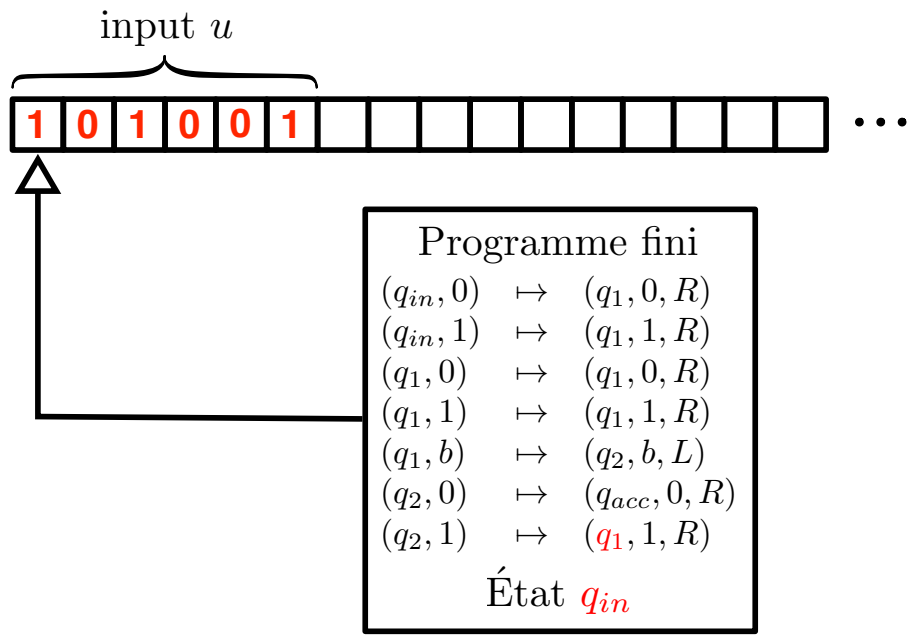
- Montrer que le langage des nombres binaires divisibles par 3 est récursivement énumérable.
- Montrer que le langage des palindromes binaires est récursivement énumérable.

Machine de Turing

- **Remarque :** Etant donné une machine de Turing $\mathcal{M}$, il se peut que, sur certains inputs, $\mathcal{M}$ ne s'arrête jamais, ni dans l'état acceptant q_{acc}, ni dans l'état rejetant q_{rej}.
- Par exemple, la machine de Turing ci-dessous ne s'arrête que sur les inputs qui se terminent par 0 ; elle boucle à l'infini sur tous les inputs qui se terminent par 1.

Machine de Turing

- **Remarque :** Etant donné une machine de Turing $\mathcal{M}$, il se peut que, sur certains inputs, $\mathcal{M}$ ne s'arrête jamais, ni dans l'état acceptant q_{acc}, ni dans l'état rejetant q_{rej}.
- Par exemple, la machine de Turing ci-dessous ne s'arrête que sur les inputs qui se terminent par 0 ; elle boucle à l'infini sur tous les inputs qui se terminent par 1.



Machine de Turing

- Une machine de Turing $\mathcal{M}$ qui s'arrête sur tous les inputs possibles est dite *décideuse* (*decider*).
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Définition 3 (Décidabilité ou Recursivité)

Un langage X est dit *Turing-décidable* ou *récursif* s'il existe une machine de Turing décideuse $\mathcal{M}$ qui le décide, i.e., s'il existe $\mathcal{M}$ telle que $L(\mathcal{M}) = X$.

- Notre exemple précédent montre que le langage $X = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$ est Turing-décidable, puisqu'on a trouvé une machine $\mathcal{M}$ qui décide L.

Machine de Turing

- Une machine de Turing $\mathcal{M}$ qui s'arrête sur tous les inputs possibles est dite *décideuse* (*decider*).
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Définition 3 (Décidabilité ou Recursivité)

Un langage X est dit *Turing-décidable* ou *récursif* s'il existe une machine de Turing décideuse $\mathcal{M}$ qui le décide, i.e., s'il existe $\mathcal{M}$ telle que $L(\mathcal{M}) = X$.

- Notre exemple précédent montre que le langage $X = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$ est Turing-décidable, puisqu'on a trouvé une machine $\mathcal{M}$ qui décide L.

Machine de Turing

- Une machine de Turing $\mathcal{M}$ qui s'arrête sur tous les inputs possibles est dite *décideuse* (*decider*).
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Définition 3 (Décidabilité ou Recursivité)

Un langage X est dit *Turing-décidable* ou *récursif* s'il existe une machine de Turing décideuse $\mathcal{M}$ qui le décide, i.e., s'il existe $\mathcal{M}$ telle que $L(\mathcal{M}) = X$.

- Notre exemple précédent montre que le langage $X = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$ est Turing-décidable, puisqu'on a trouvé une machine $\mathcal{M}$ qui décide L.

Machine de Turing

- Une machine de Turing $\mathcal{M}$ qui s'arrête sur tous les inputs possibles est dite *décideuse* (*decider*).
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Définition 3 (Décidabilité ou Recursivité)

Un langage X est dit *Turing-décidable* ou *récursif* s'il existe une machine de Turing décideuse $\mathcal{M}$ qui le décide, i.e., s'il existe $\mathcal{M}$ telle que $L(\mathcal{M}) = X$.

- Notre exemple précédent montre que le langage $X = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$ est Turing-décidable, puisqu'on a trouvé une machine $\mathcal{M}$ qui décide L.

Machine de Turing

- Une machine de Turing $\mathcal{M}$ qui s'arrête sur tous les inputs possibles est dite *décideuse* (*decider*).
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Définition 3 (Décidabilité ou Recursivité)

Un langage X est dit *Turing-décidable* ou *récursif* s'il existe une machine de Turing décideuse $\mathcal{M}$ qui le décide, i.e., s'il existe $\mathcal{M}$ telle que $L(\mathcal{M}) = X$.

- Notre exemple précédent montre que le langage $X = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$ est Turing-décidable, puisqu'on a trouvé une machine $\mathcal{M}$ qui décide L.

Machine de Turing

Démo simulateur :

▸ Link to the simulator

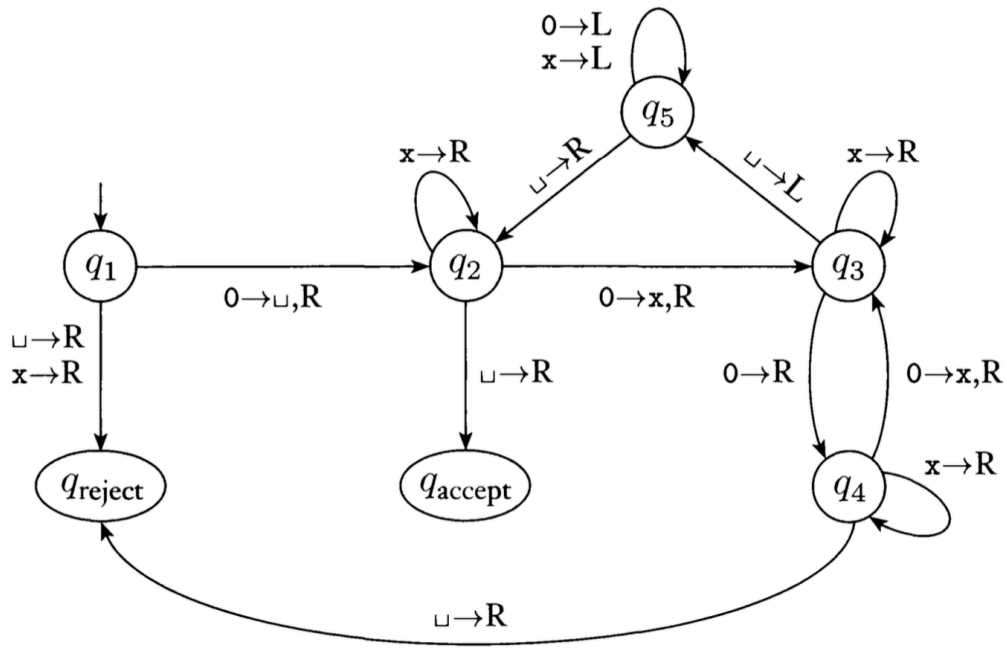
- Montrer que le langage des nombres binaires divisibles par 3 est récursif.
- Montrer que le langage des palindromes binaires est récursif.
- Modifier les programmes de sorte que la Machine de Turing reconnaisse ces langages mais ne les décide pas.

  Remplacer qReject par qNew et ajouter les instructions qNew,0 qNew,1 qNew,0,- qNew,1,-

  On remarque que la machine boucle sur les inputs qui devraient être rejetés...

Machine de Turing

- Une machine de Turing peut être représentée par un graphe.
- Par exemple, la machine de Turing ci-dessous décide le langage $L = \{0^{2^n} : n \geq 0\}$



Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

Résumé :

- Un langage X *Turing-reconnaissable* ou *récursivement énumérable* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$ ou $\mathcal{M}(u)$ ne s'arrête pas
- Un langage X *Turing-décidable* ou *récursif* ssi il existe une machine de Turing $\mathcal{M}$ telle que :
  - si $u \in X$, alors $\mathcal{M}(u) = q_{accept}$
  - si $u \notin X$, alors $\mathcal{M}(u) = q_{reject}$
- Donc « récursif » implique « récursivement énumérable ».
- La récursivité est une notion plus forte...

Machine de Turing

- Toute la théorie peut se transposer facilement du cas des « langages » à celui des « fonctions ».
- Supposons que nos machines de Turing ne contiennent plus qu'un seul état final q_{final}.

Définition 4 (fonction récursive)

Une fonction $f : \Sigma^* \to \Sigma^*$ est dite *Turing-calculable* ou *récursive* s'il existe une machine de Turing $\mathcal{M}$ (sur l'alphabet Σ) telle que, pour tout input u, la computation de $\mathcal{M}$ sur u s'arrête dans l'état final q_{final} avec $f(u)$ écrit sur sa bande.

Machine de Turing

- Toute la théorie peut se transposer facilement du cas des « langages » à celui des « fonctions ».
- Supposons que nos machines de Turing ne contiennent plus qu'un seul état final q_{final}.

Définition 4 (fonction récursive)

Une fonction $f : \Sigma^* \to \Sigma^*$ est dite *Turing-calculable* ou *récursive* s'il existe une machine de Turing $\mathcal{M}$ (sur l'alphabet Σ) telle que, pour tout input u, la computation de $\mathcal{M}$ sur u s'arrête dans l'état final q_{final} avec $f(u)$ écrit sur sa bande.

Machine de Turing

- Toute la théorie peut se transposer facilement du cas des « langages » à celui des « fonctions ».
- Supposons que nos machines de Turing ne contiennent plus qu'un seul état final q_{final}.

Définition 4 (fonction récursive)

Une fonction $f : \Sigma^* \to \Sigma^*$ est dite *Turing-calculable* ou *récursive* s'il existe une machine de Turing $\mathcal{M}$ (sur l'alphabet Σ) telle que, pour tout input u, la computation de $\mathcal{M}$ sur u s'arrête dans l'état final q_{final} avec $f(u)$ écrit sur sa bande.

Machine de Turing

- Toute la théorie peut se transposer facilement du cas des « langages » à celui des « fonctions ».
- Supposons que nos machines de Turing ne contiennent plus qu'un seul état final q_{final}.

Définition 4 (fonction récursive)

Une fonction $f : \Sigma^* \to \Sigma^*$ est dite *Turing-calculable* ou *récursive* s'il existe une machine de Turing $\mathcal{M}$ (sur l'alphabet Σ) telle que, pour tout input u, la computation de $\mathcal{M}$ sur u s'arrête dans l'état final q_{final} avec $f(u)$ écrit sur sa bande.

Machine de Turing

Démo simulateur :

▸ Link to the simulator

- L'addition binaire est Turing-calculable.
- La multiplication binaire est Turing-calculable.

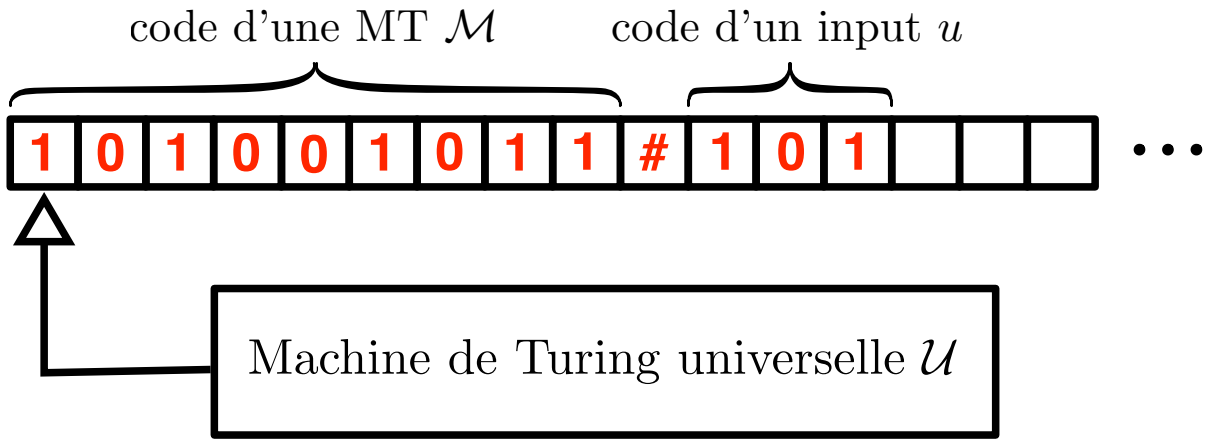
Machine de Turing universelle

- Il est possible de construire une machine de Turing dite *universelle*, i.e., qui peut simuler n'importe quelle autre machine de Turing sur n'importe quel input.

6. *The universal computing machine.*

It is possible to invent a single machine which can be used to compute any computable sequence. If this machine $\mathfrak{U}$ is supplied with a tape on the beginning of which is written the S.D of some computing machine $\mathfrak{M}$, then $\mathfrak{U}$ will compute the same sequence as $\mathfrak{M}$. In this section I explain in outline the behaviour of the machine. The next section is devoted to giving the complete table for $\mathfrak{U}$.

Machine de Turing universelle



$\Rightarrow$ Renvoie le résultat de $\mathcal{M}(u)$

Thèse de Church-Turing

- Les fonctions calculables en terme de machines de Turing, de μ-récursion, de lambda calcul, et autre modèles encore, sont toutes équivalentes, et appelées *récursives*.
- Ceci amenèrent Church et Turing à formuler la proposition informelle suivante :

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions récursives.

- Cette thèse n'est pas un théorème mathématique, car la notion de « calculabilité effective » est informelle.

Thèse de Church-Turing

- Les fonctions calculables en terme de machines de Turing, de μ-récursion, de lambda calcul, et autre modèles encore, sont toutes équivalentes, et appelées *récursives*.
- Ceci amenèrent Church et Turing à formuler la proposition informelle suivante :

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions récursives.

- Cette thèse n'est pas un théorème mathématique, car la notion de « calculabilité effective » est informelle.

Thèse de Church-Turing

- Les fonctions calculables en terme de machines de Turing, de μ-récursion, de lambda calcul, et autre modèles encore, sont toutes équivalentes, et appelées *récursives*.
- Ceci amenèrent Church et Turing à formuler la proposition informelle suivante :

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions récursives.

- Cette thèse n'est pas un théorème mathématique, car la notion de « calculabilité effective » est informelle.

Thèse de Church-Turing

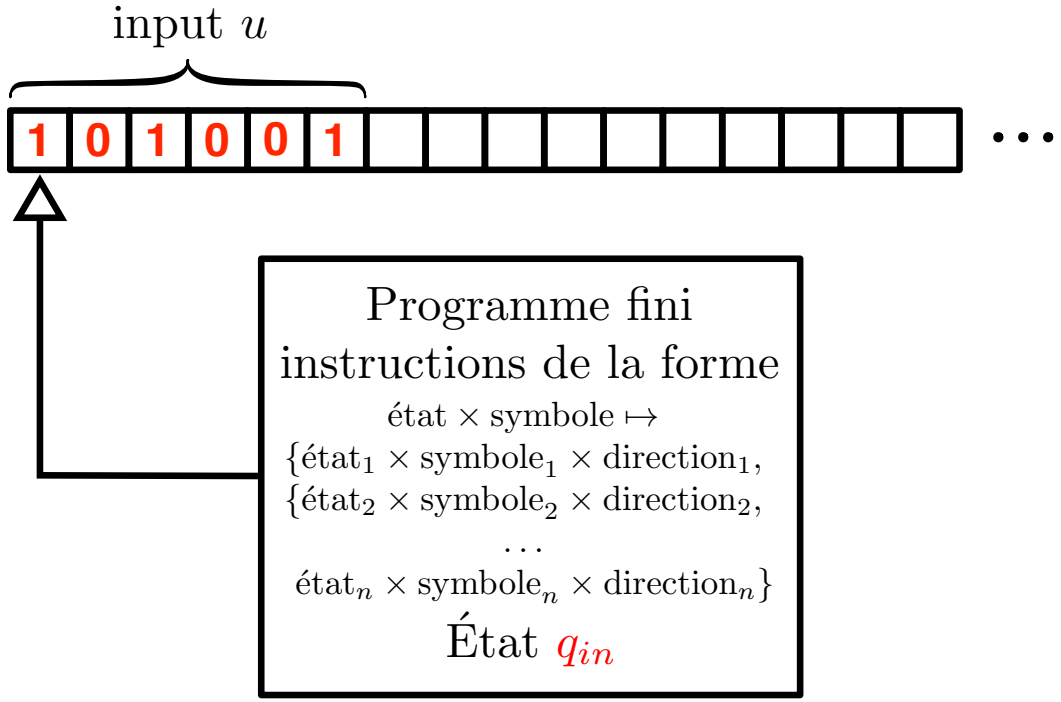
- Les fonctions calculables en terme de machines de Turing, de μ-récursion, de lambda calcul, et autre modèles encore, sont toutes équivalentes, et appelées *récursives*.
- Ceci amenèrent Church et Turing à formuler la proposition informelle suivante :

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions récursives.

- Cette thèse n'est pas un théorème mathématique, car la notion de « calculabilité effective » est informelle.

Machine de Turing non-déterministe

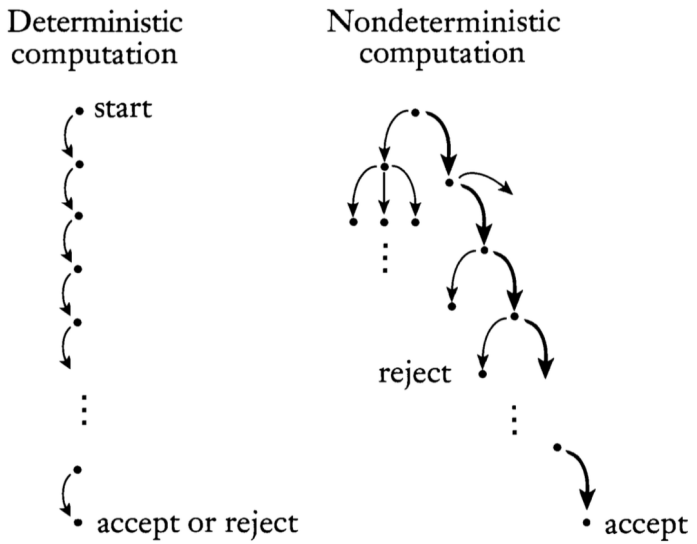
Une *machine de Turing non-déterministe* est une machine de Turing qui, à chaque pas de calcul, a possiblement le choix entre plusieurs possibilités, i.e., entre plusieurs transitions possibles.



Machine de Turing non-déterministe

Pour une machine de Turing non-déterministe, on définit

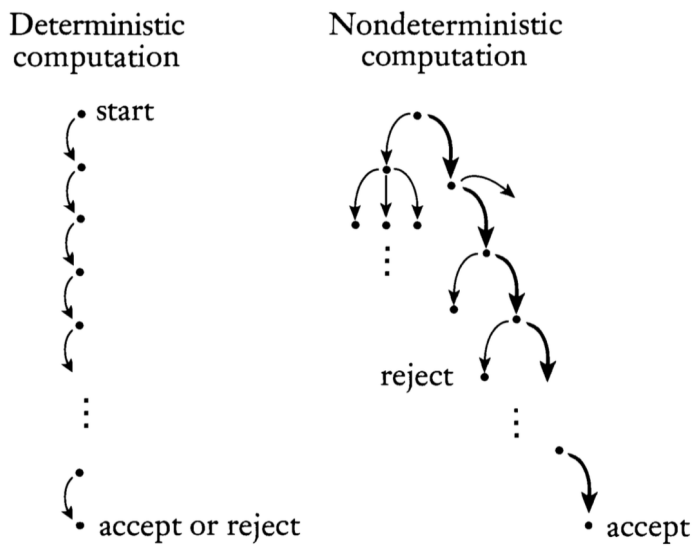
- l'input u est *accepté* par $\mathcal{M}$ s'*il existe* un branche de l'arbre de computation de $\mathcal{M}(u)$ qui atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si *toutes* les branches de l'arbre de computation de $\mathcal{M}(u)$ atteignent l'état spécial q_{rej}
- l'input u n'est ni accepté ni rejeté par $\mathcal{M}$ sinon, i.e., s'il existe une branche de l'arbre de computation qui ne s'arrête jamais



Machine de Turing non-déterministe

Pour une machine de Turing non-déterministe, on définit

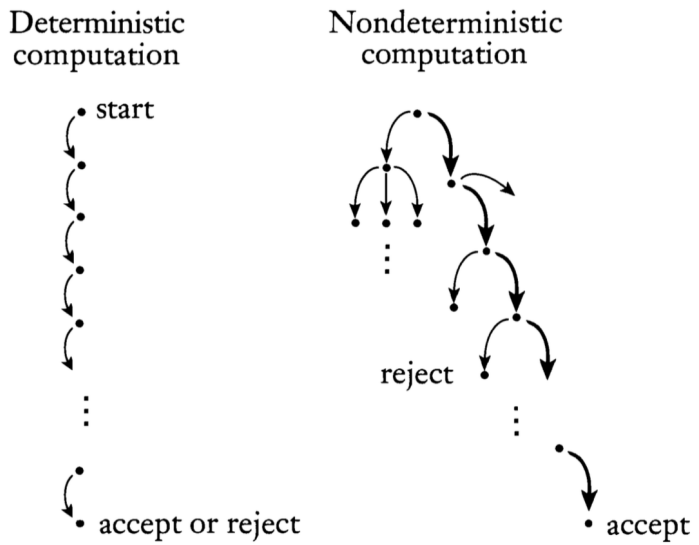
- l'input u est *accepté* par $\mathcal{M}$ s'*il existe* un branche de l'arbre de computation de $\mathcal{M}(u)$ qui atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si *toutes* les branches de l'arbre de computation de $\mathcal{M}(u)$ atteignent l'état spécial q_{rej}
- l'input u n'est ni accepté ni rejeté par $\mathcal{M}$ sinon, i.e., s'il existe une branche de l'arbre de computation qui ne s'arrête jamais



Machine de Turing non-déterministe

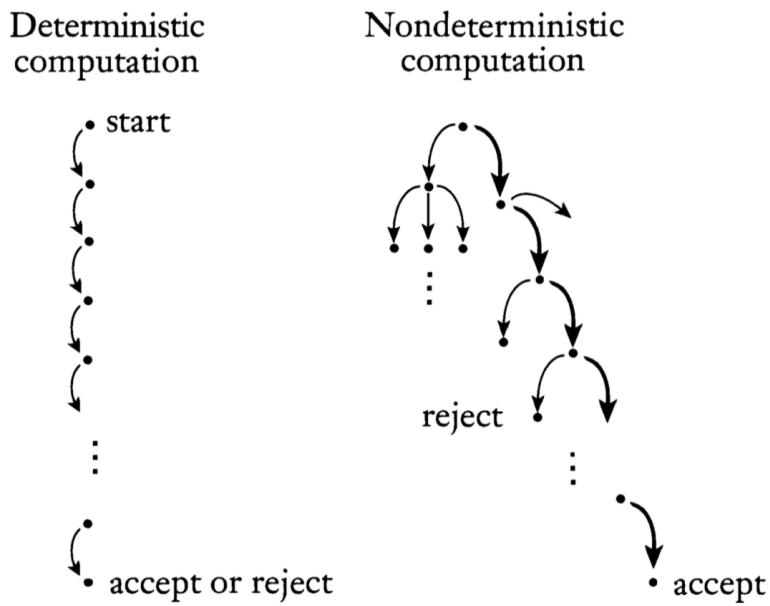
Pour une machine de Turing non-déterministe, on définit

- l'input u est *accepté* par $\mathcal{M}$ s'*il existe* un branche de l'arbre de computation de $\mathcal{M}(u)$ qui atteint l'état spécial q_{acc}
- l'input u est *rejecté* par $\mathcal{M}$ si *toutes* les branches de l'arbre de computation de $\mathcal{M}(u)$ atteignent l'état spécial q_{rej}
- l'input u n'est ni accepté ni rejeté par $\mathcal{M}$ sinon, i.e., s'il existe une branche de l'arbre de computation qui ne s'arrête jamais



Machine de Turing non-déterministe

- Une machine de Turing déterministe est donc un cas spécifique (dégénéré) de machine de Turing non-déterministe dont les choix de transitions possibles sont réduits, à chaque pas de calcul, à une seule possibilité.



Machine de Turing non-déterministe

Définition 5 (MT non-déterministe)

Une *machine de Turing* non-déterministe est un tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ est la fonction de transition
- $q_0 \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing non-déterministe

Définition 5 (MT non-déterministe)

Une *machine de Turing* non-déterministe est un tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ est la fonction de transition
- $q_0 \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing non-déterministe

Définition 5 (MT non-déterministe)

Une *machine de Turing* non-déterministe est un tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ est la fonction de transition
- $q_0 \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing non-déterministe

Définition 5 (MT non-déterministe)

Une *machine de Turing* non-déterministe est un tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ est la fonction de transition
- $q_0 \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing non-déterministe

Définition 5 (MT non-déterministe)

Une *machine de Turing* non-déterministe est un tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ est la fonction de transition
- $q_0 \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing non-déterministe

Définition 5 (MT non-déterministe)

Une *machine de Turing* non-déterministe est un tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ est la fonction de transition
- $q_0 \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing non-déterministe

Définition 5 (MT non-déterministe)

Une *machine de Turing* non-déterministe est un tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ où

- Q est un ensemble fini d'états
- Σ est un alphabet d'input fini (avec $b \notin \Sigma$)
- Γ est un alphabet de bande, tel que $b \in \Gamma$ et $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ est la fonction de transition
- $q_0 \in Q$ est l'état initial
- $q_{acc} \in Q$ est l'état acceptant
- $q_{rej} \in Q$ est l'état rejetant

Machine de Turing non-déterministe

- Si $\mathcal{M}$ est une machine de Turing non-déterministe, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage de* $\mathcal{M}$ ou *langage reconnu par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.
- Une machine de Turing non-déterministe $\mathcal{M}$ qui, sur tous les inputs possibles, possède toutes ses branches de computation qui s'arrêtent toujours est dite *décideuse*.
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Machine de Turing non-déterministe

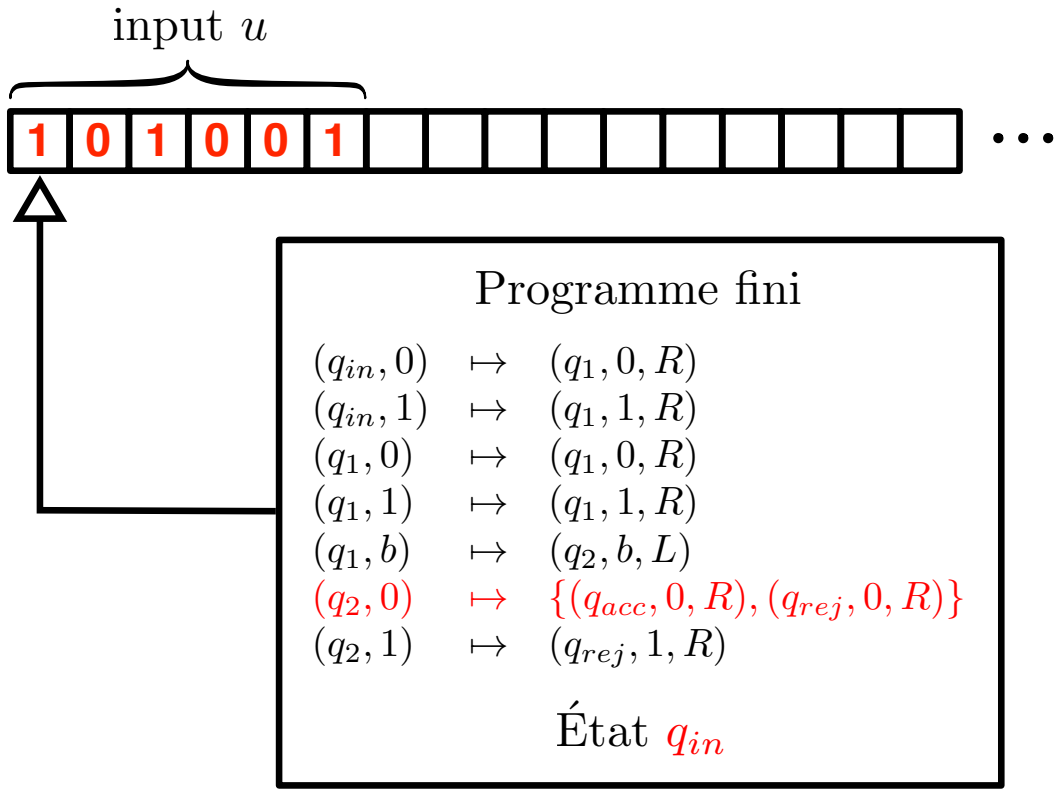
- Si $\mathcal{M}$ est une machine de Turing non-déterministe, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage de* $\mathcal{M}$ ou *langage reconnu par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.
- Une machine de Turing non-déterministe $\mathcal{M}$ qui, sur tous les inputs possibles, possède toutes ses branches de computation qui s'arrêtent toujours est dite *décideuse*.
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Machine de Turing non-déterministe

- Si $\mathcal{M}$ est une machine de Turing non-déterministe, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage de* $\mathcal{M}$ ou *langage reconnu par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.
- Une machine de Turing non-déterministe $\mathcal{M}$ qui, sur tous les inputs possibles, possède toutes ses branches de computation qui s'arrêtent toujours est dite *décideuse*.
- Si $\mathcal{M}$ est une machine de Turing décideuse, l'ensemble des inputs qui sont acceptés par $\mathcal{M}$ est appelé *langage décidé par* $\mathcal{M}$; il est noté $L(\mathcal{M})$.

Machine de Turing non-déterministe

- La machine de Turing non-déterministe ci-dessous *reconnait* et *décide* le langage $L = \{u \in \{0,1\}^* : u \text{ se termine par } 0\}$.



Machine de Turing non-déterministe

Proposition 6 (Equivalence dét. et non dét.)

Toute machine de Turing non-déterministe $\mathcal{N}$ est équivalente à une machine de Turing déterministe $\mathcal{D}$.

- La proposition inverse est triviale : toute machine de Turing déterministe est équivalente à une machine de Turing non-déterministe, à savoir elle-même (puisque le déterminisme n'est qu'in cas dégénéré de non déterminisme où les choix sont réduits à néant).

Machine de Turing non-déterministe

Proposition 6 (Equivalence dét. et non dét.)

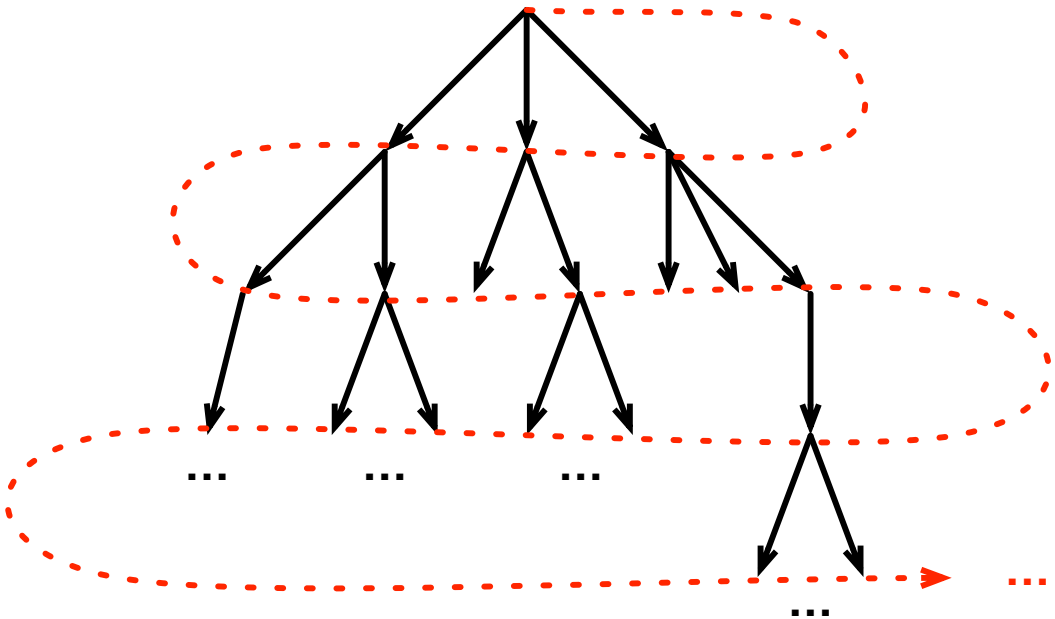
Toute machine de Turing non-déterministe $\mathcal{N}$ est équivalente à une machine de Turing déterministe $\mathcal{D}$.

- La proposition inverse est triviale : toute machine de Turing déterministe est équivalente à une machine de Turing non-déterministe, à savoir elle-même (puisque le déterminisme n'est qu'in cas dégénéré de non déterminisme où les choix sont réduits à néant).

Machine de Turing non-déterministe

Preuve :

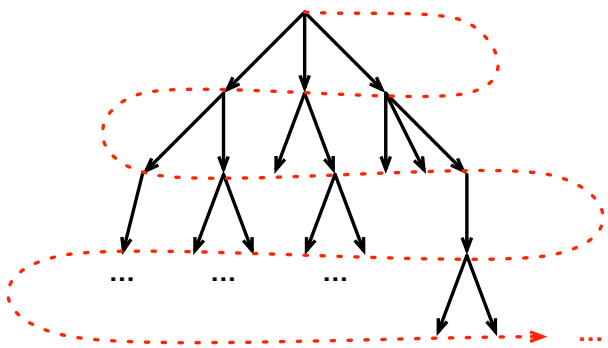
- On construit une MT dét. $\mathcal{D}$ qui simule l'arbre de computation de la MT non-dét. $\mathcal{N}$ *étage par étage*.



Simulation par $\mathcal{D}$ de l'arbre de computation de $\mathcal{N}$

Machine de Turing non-déterministe

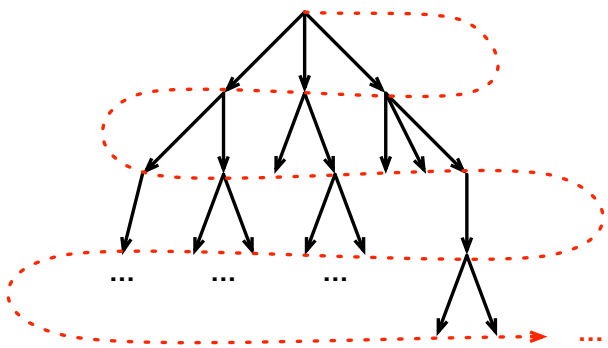
- Cela est possible car la largeur de chaque étage de l'arbre de computation de $\mathcal{N}$ est toujours finie, du fait que $\mathcal{N}$ possède toujours un nombre fini de choix non-dét.
- Si $\mathcal{D}$ tombe sur un état acceptant de $\mathcal{N}$, elle accepte l'input ; si $\mathcal{D}$ ne tombe que sur des états rejetant pour toutes les branches de $\mathcal{N}$, elle rejète l'input ; sinon, elle ne s'arrête pas.



Simulation par $\mathcal{D}$ de l'arbre de computation de $\mathcal{N}$

Machine de Turing non-déterministe

- Cela est possible car la largeur de chaque étage de l'arbre de computation de $\mathcal{N}$ est toujours finie, du fait que $\mathcal{N}$ possède toujours un nombre fini de choix non-dét.
- Si $\mathcal{D}$ tombe sur un état acceptant de $\mathcal{N}$, elle accepte l'input ; si $\mathcal{D}$ ne tombe que sur des états rejetant pour toutes les branches de $\mathcal{N}$, elle rejète l'input ; sinon, elle ne s'arrête pas. □



Simulation par $\mathcal{D}$ de l'arbre de computation de $\mathcal{N}$

Machine de Turing non-déterministe

Corollaire 7

- *Un langage est Turing-reconnaissable si et seulement si il est reconnaissable par une machine de Turing non-déterministe.*
- *Un langage est Turing-décidable si et seulement si il est décidable par une machine de Turing non-déterministe.*

Preuve :

($\Rightarrow$) L Turing-rec. (resp. Turing-déc.) ssi (par déf.) L rec. (resp. déc.) par une MT dét. $\mathcal{D}$, qui est également une MT non-dét.

($\Leftarrow$) Si L est rec. (resp. déc.) par une MT non-dét. $\mathcal{N}$, alors, par la Proposition 6, L est rec. (resp. déc.) par une MT dét. $\mathcal{D}$, i.e. L Turing-rec. (resp. Turing-déc.) $\square$

Machine de Turing non-déterministe

Corollaire 7

- *Un langage est Turing-reconnaissable si et seulement si il est reconnaissable par une machine de Turing non-déterministe.*
- *Un langage est Turing-décidable si et seulement si il est décidable par une machine de Turing non-déterministe.*

Preuve :

($\Rightarrow$) L Turing-rec. (resp. Turing-déc.) ssi (par déf.) L rec. (resp. déc.) par une MT dét. $\mathcal{D}$, qui est également une MT non-dét.

($\Leftarrow$) Si L est rec. (resp. déc.) par une MT non-dét. $\mathcal{N}$, alors, par la Proposition 6, L est rec. (resp. déc.) par une MT dét. $\mathcal{D}$, i.e. L Turing-rec. (resp. Turing-déc.) $\square$

Machine de Turing non-déterministe

Corollaire 7

- *Un langage est Turing-reconnaissable si et seulement si il est reconnaissable par une machine de Turing non-déterministe.*
- *Un langage est Turing-décidable si et seulement si il est décidable par une machine de Turing non-déterministe.*

Preuve :

($\Rightarrow$) L Turing-rec. (resp. Turing-déc.) ssi (par déf.) L rec. (resp. déc.) par une MT dét. $\mathcal{D}$, qui est également une MT non-dét.

($\Leftarrow$) Si L est rec. (resp. déc.) par une MT non-dét. $\mathcal{N}$, alors, par la Proposition 6, L est rec. (resp. déc.) par une MT dét. $\mathcal{D}$, i.e. L Turing-rec. (resp. Turing-déc.) $\square$

$\mathbf{P \neq NP}$?

- Nous venons de voir (Corollaire 7) que le non-déterminisme n'apporte aucun pouvoir computationnel supplémentaire aux machines de Turing, i.e., tout langage décidable par une machine de Turing non-déterministe l'est également par une machine de Turing déterministe (et vice versa).
- Mais les langages décidables par des machines de Turing non-déterministes sont ils globalement plus difficiles à résoudre – en terme de *temps de calcul* – que ceux décidables par des machines de Turing déterministes ?
- Ceci constitue le plus important problème ouvert en informatique théorique, à savoir $\mathbf{P \neq NP}$? (1 Mo Dollars).

$\mathbf{P} \neq \mathbf{NP}$?

- Nous venons de voir (Corollaire 7) que le non-déterminisme n'apporte aucun pouvoir computationnel supplémentaire aux machines de Turing, i.e., tout langage décidable par une machine de Turing non-déterministe l'est également par une machine de Turing déterministe (et vice versa).
- Mais les langages décidables par des machines de Turing non-déterministes sont ils globalement plus difficiles à résoudre – en terme de *temps de calcul* – que ceux décidables par des machines de Turing déterministes ?
- Ceci constitue le plus important problème ouvert en informatique théorique, à savoir $\mathbf{P} \neq \mathbf{NP}$? (1 Mo Dollars).

$\mathbf{P} \neq \mathbf{NP}$?

- Nous venons de voir (Corollaire 7) que le non-déterminisme n'apporte aucun pouvoir computationnel supplémentaire aux machines de Turing, i.e., tout langage décidable par une machine de Turing non-déterministe l'est également par une machine de Turing déterministe (et vice versa).
- Mais les langages décidables par des machines de Turing non-déterministes sont ils globalement plus difficiles à résoudre – en terme de *temps de calcul* – que ceux décidables par des machines de Turing déterministes ?
- Ceci constitue le plus important problème ouvert en informatique théorique, à savoir $\mathbf{P} \neq \mathbf{NP}$? (1 Mo Dollars).

$\mathbf{P \neq NP}$?

Définition 8 (temps de calcul)

Soit $\mathcal{M}$ une machine de Turing déterministe décideuse. On appelle *temps de calcul de* $\mathcal{M}$ la fonction $f : \mathbb{N} \to \mathbb{N}$, où $f(n)$ est le nombre maximum d'étapes de calcul effectuées par $\mathcal{M}$ pour décider un input de longueur n.

$\mathbf{P \neq NP}$?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :
  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : n étapes pour la ligne 1 ; n étapes pour le scan 1 à la ligne 2 ; n étapes pour repositionner la tête en début de bande ; Pour la ligne 2 et 3 : au plus $n/2$ scan (1 scan aller et retour) de n étapes pour scanner, n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Pour la ligne 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = O(n^2)$

$\mathbf{P \neq NP\,?}$

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : 0 (pour l'étape 1) + n étapes pour le scan + n étapes pour repositionner la tête en début de bande ; Boucle 2 et 3 : au plus $n/2$ scan (décroissant) et n étapes pour scanner + n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Boucle 4 : n étapes pour le scan ; En tout : $2n^2 + 2n = O(n^2)$

$\mathbf{P \neq NP}$?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : $n+1$ étapes pour le scan ; n étapes pour repositionner la tête en début de bande ; Pour i allant de 1 à $n/2$: $n-2(i-1)$ étapes pour scanner, 1 étape pour se repositionner au début, donc $n/2 + \sum_{i=1}^{n/2}(n-2(i-1)) = n^2$ étapes ; Point fixe : n étapes pour le scan ; Enfin : $2n^2 + 2n = O(n^2)$.

$\mathbf{P \neq NP\,?}$

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : n étapes pour le scan, $+ n$ étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ cycles, chacun avec n étapes pour scanner, $+ n$ étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = O(n^2)$

$\mathbf{P} \neq \mathbf{NP}$?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : Point 1 : n étapes pour le scan $+$ n étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ scans, chacun prenant n étapes pour scanner $+$ n étapes pour se repositionner au début, donc $n/2 \cdot (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = O(n^2)$.

P ≠ NP ?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : n étapes pour le scan, n étapes pour repositionner la tête en début de bande, $n/2 + n/4 + n/8 + \ldots + 1$ étapes pour scanner, n étapes pour repositionner au début, donc $n/2 + (n/4 + n/8 + \ldots + 1) + n$ étapes, donc n étapes pour le scan, en tout $\leq 2n + n + n = 4n$ étapes.

$\mathbf{P \neq NP}$?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : Point 1 : n étapes pour le scan + n étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ scans, chacun prenant n étapes pour scanner + n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = \mathcal{O}(n^2)$.

$\mathbf{P \neq NP}$?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : Point 1 : n étapes pour le scan + n étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ scans, chacun prenant n étapes pour scanner + n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = \mathcal{O}(n^2)$.

P ≠ NP ?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : Point 1 : n étapes pour le scan + n étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ scans, chacun prenant n étapes pour scanner + n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = \mathcal{O}(n^2)$.

P ≠ NP ?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : Point 1 : n étapes pour le scan + n étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ scans, chacun prenant n étapes pour scanner + n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = \mathcal{O}(n^2)$.

P ≠ NP ?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : Point 1 : n étapes pour le scan + n étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ scans, chacun prenant n étapes pour scanner + n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = \mathcal{O}(n^2)$.

P ≠ NP ?

- Par exemple, soit le langage $L = \{0^k 1^k : k \geq 0\}$ décidé par la machine de Turing déterministe $\mathcal{M}$ suivante :

  Étant donné l'input w (de taille n) :

  1. scanner l'input et *rejeter* l'input si un 0 est trouvé après un 1
  2. tant qu'un 0 et un 1 demeurent sur la bande, répéter :
  3. scanner la bande et biffer un 0 puis un 1
  4. s'il reste des 0 alors que les 1 ont tous été biffés, ou vice versa : *rejeter* l'input ; sinon *accepter* l'input.

- Temps de calcul de $\mathcal{M}$, au pire des cas : Point 1 : n étapes pour le scan + n étapes pour repositionner la tête en début de bande ; Points 2 et 3 : au plus $n/2$ scans, chacun prenant n étapes pour scanner + n étapes pour se repositionner au début, donc $n/2 * (n + n) = n^2$ étapes ; Point 4 : n étapes pour le scan ; En tout : $2n + n^2 + n = \mathcal{O}(n^2)$.

$\mathbf{P \neq NP}$?

- Dans l'exemple précédent, le langage L est décidé par une machine de Turing déterministe en temps polynomial $\mathcal{O}(n^2)$.

Définition 9 (classe PTIME)

On appelle **PTIME** ou **P** l'ensemble de tous les langages décidables en temps polynomial par un machine de Turing déterministe.

$\mathbf{P \neq NP}$?

- Dans l'exemple précédent, le langage L est décidé par une machine de Turing déterministe en temps polynomial $\mathcal{O}(n^2)$.

Définition 9 (classe PTIME)

On appelle **PTIME** ou **P** l'ensemble de tous les langages décidables en temps polynomial par un machine de Turing déterministe.

$\mathbf{P \neq NP\,?}$

Définition 10 (temps de calcul non-dét)

Soit $\mathcal{M}$ une machine de Turing non-déterministe décideuse. On appelle *temps de calcul de* $\mathcal{M}$ la fonction $f : \mathbb{N} \to \mathbb{N}$, où $f(n)$ est le nombre maximum d'étapes de calcul effectuées par $\mathcal{M}$ sur n'importe quelle branche de son arbre de computation pour un input de longueur n.

P ≠ NP ?

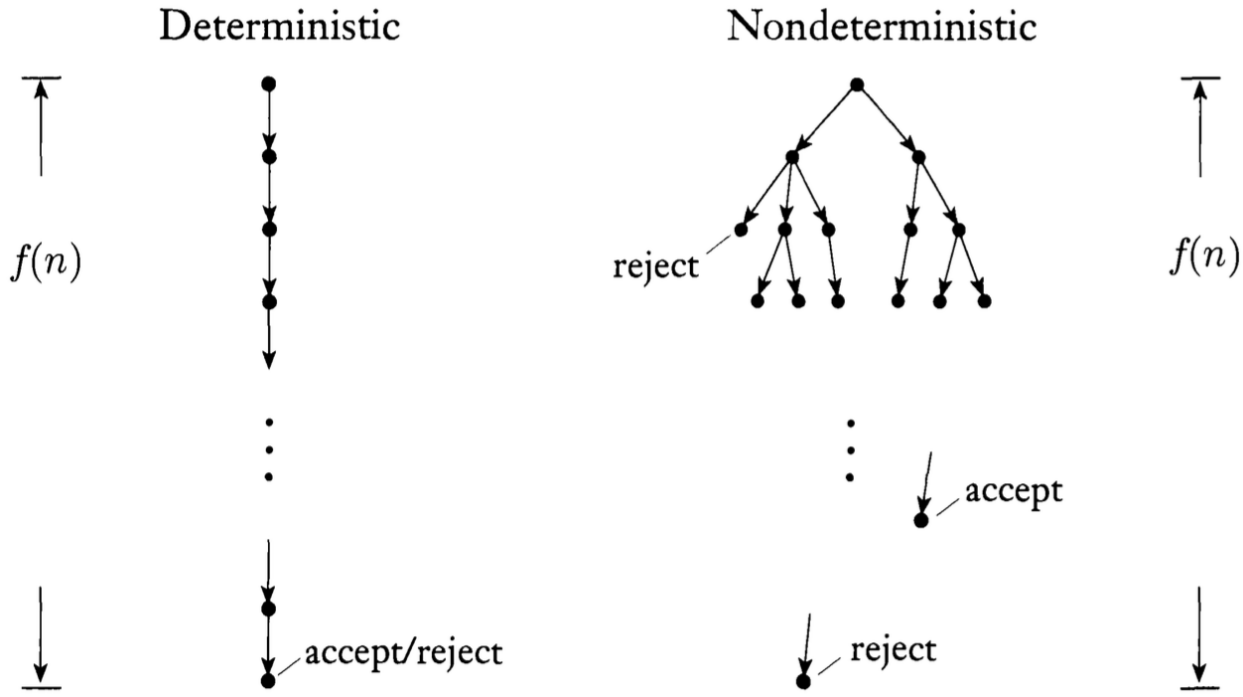


Figure : Illustration des temps de calcul dans les cas déterministes et non-déterministes

$\mathbf{P \neq NP\,?}$

- Considérons le problème $HPATH$ de savoir si un graphe possède un chemin Hamiltonien ou non, i.e.,

$$HPATH = \{\langle G, s, t\rangle : G \text{ est un graphe dirigé possédant un chemin Hamiltonien de } s \text{ à } t\}$$

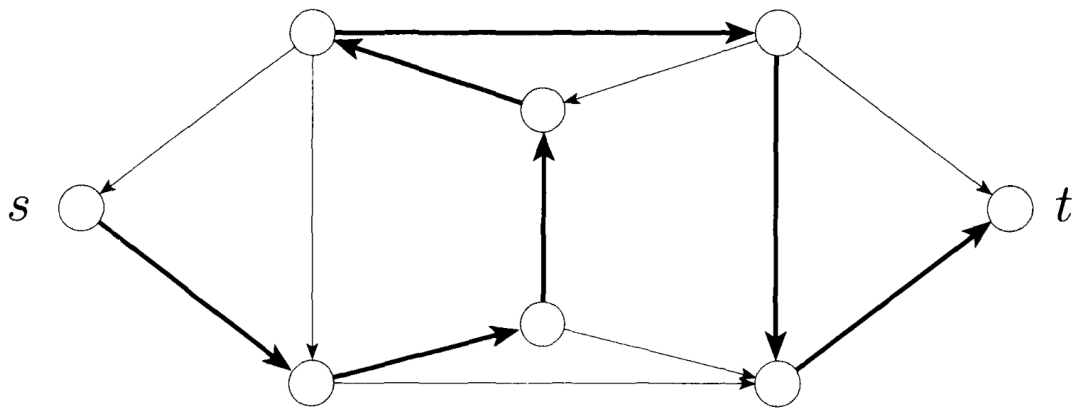


Figure : Un chemin Hamiltonien de s à t est un chemin de s à t qui visite tous les noeuds du graphe une et une seule fois

P ≠ NP ?

- Le langage $HPATH$ décidé par la machine de Turing non-déterministe $\mathcal{N}$ suivante :

  Étant donné l'input $\langle G, s, t\rangle$ (de taille n), où G est un graphe avec m noeuds, et s et t sont des noeuds de G :

  1. écrire à la fin de l'input une liste de m nombres (noeuds) $1 \leq p_1, \ldots, p_m \leq m$; chaque nombre procède d'un choix *non-déterministe* parmi m possibilités
  2. s'il y a une répétition parmi ces nombre, *rejeter* l'input
  3. si $p_1 \neq s$ ou $p_m \neq t$, *rejeter* l'input
  4. pour tout $i = 1, \ldots, m-1$, vérifier que (p_i, p_{i+1}) est une arrête du graphe ; si ce n'est pas le cas, *rejeter* l'input, sinon *accepter* l'input

- Temps de calcul de $\mathcal{N}$, au pire des cas : on peut remarquer que les points 1, 2, 3 et 4 s'effectuent tous en temps polynomial ; donc on a un temps total $\mathcal{O}(p(n))$, où p polynôme.

$\mathbf{P \neq NP}$?

- Le langage $HPATH$ décidé par la machine de Turing non-déterministe $\mathcal{N}$ suivante :

  Étant donné l'input $\langle G, s, t \rangle$ (de taille n), où G est un graphe avec m noeuds, et s et t sont des noeuds de G :

  1. écrire à la fin de l'input une liste de m nombres (noeuds) $1 \leq p_1, \ldots, p_m \leq m$; chaque nombre procède d'un choix *non-déterministe* parmi m possibilités
  2. s'il y a une répétition parmi ces nombre, *rejeter* l'input
  3. si $p_1 \neq s$ ou $p_m \neq t$, *rejeter* l'input
  4. pour tout $i = 1, \ldots, m-1$, vérifier que (p_i, p_{i+1}) est une arrête du graphe ; si ce n'est pas le cas, *rejeter* l'input, sinon *accepter* l'input

- Temps de calcul de $\mathcal{N}$, au pire des cas : on peut remarquer que les points 1, 2, 3 et 4 s'effectuent tous en temps polynomial ; donc on a un temps total $\mathcal{O}(p(n))$, où p polynôme.

P ≠ NP ?

- Le langage $HPATH$ décidé par la machine de Turing non-déterministe $\mathcal{N}$ suivante :

  Étant donné l'input $\langle G, s, t\rangle$ (de taille n), où G est un graphe avec m noeuds, et s et t sont des noeuds de G :

  1. écrire à la fin de l'input une liste de m nombres (noeuds) $1 \leq p_1, \ldots, p_m \leq m$; chaque nombre procède d'un choix *non-déterministe* parmi m possibilités
  2. s'il y a une répétition parmi ces nombre, *rejeter* l'input
  3. si $p_1 \neq s$ ou $p_m \neq t$, *rejeter* l'input
  4. pour tout $i = 1, \ldots, m-1$, vérifier que (p_i, p_{i+1}) est une arrête du graphe ; si ce n'est pas le cas, *rejeter* l'input, sinon *accepter* l'input

- Temps de calcul de $\mathcal{N}$, au pire des cas : on peut remarquer que les points 1, 2, 3 et 4 s'effectuent tous en temps polynomial ; donc on a un temps total $\mathcal{O}(p(n))$, où p polynôme.

P ≠ NP ?

- Le langage $HPATH$ décidé par la machine de Turing non-déterministe $\mathcal{N}$ suivante :

  Étant donné l'input $\langle G, s, t\rangle$ (de taille n), où G est un graphe avec m noeuds, et s et t sont des noeuds de G :

  1. écrire à la fin de l'input une liste de m nombres (noeuds) $1 \leq p_1, \ldots, p_m \leq m$; chaque nombre procède d'un choix *non-déterministe* parmi m possibilités
  2. s'il y a une répétition parmi ces nombre, *rejeter* l'input
  3. si $p_1 \neq s$ ou $p_m \neq t$, *rejeter* l'input
  4. pour tout $i = 1, \ldots, m-1$, vérifier que (p_i, p_{i+1}) est une arrête du graphe ; si ce n'est pas le cas, *rejeter* l'input, sinon *accepter* l'input

- Temps de calcul de $\mathcal{N}$, au pire des cas : on peut remarquer que les points 1, 2, 3 et 4 s'effectuent tous en temps polynomial ; donc on a un temps total $\mathcal{O}(p(n))$, où p polynôme.

P ≠ NP ?

- Le langage $HPATH$ décidé par la machine de Turing non-déterministe $\mathcal{N}$ suivante :

  Étant donné l'input $\langle G, s, t\rangle$ (de taille n), où G est un graphe avec m noeuds, et s et t sont des noeuds de G :

  1. écrire à la fin de l'input une liste de m nombres (noeuds) $1 \leq p_1, \ldots, p_m \leq m$; chaque nombre procède d'un choix *non-déterministe* parmi m possibilités
  2. s'il y a une répétition parmi ces nombre, *rejeter* l'input
  3. si $p_1 \neq s$ ou $p_m \neq t$, *rejeter* l'input
  4. pour tout $i = 1, \ldots, m-1$, vérifier que (p_i, p_{i+1}) est une arrête du graphe ; si ce n'est pas le cas, *rejeter* l'input, sinon *accepter* l'input

- Temps de calcul de $\mathcal{N}$, au pire des cas : on peut remarquer que les points 1, 2, 3 et 4 s'effectuent tous en temps polynomial ; donc on a un temps total $\mathcal{O}(p(n))$, où p polynôme.

P ≠ NP ?

- Le langage $HPATH$ décidé par la machine de Turing non-déterministe $\mathcal{N}$ suivante :

  Étant donné l'input $\langle G, s, t\rangle$ (de taille n), où G est un graphe avec m noeuds, et s et t sont des noeuds de G :

  1. écrire à la fin de l'input une liste de m nombres (noeuds) $1 \leq p_1, \ldots, p_m \leq m$; chaque nombre procède d'un choix *non-déterministe* parmi m possibilités
  2. s'il y a une répétition parmi ces nombre, *rejeter* l'input
  3. si $p_1 \neq s$ ou $p_m \neq t$, *rejeter* l'input
  4. pour tout $i = 1, \ldots, m-1$, vérifier que (p_i, p_{i+1}) est une arrête du graphe ; si ce n'est pas le cas, *rejeter* l'input, sinon *accepter* l'input

- Temps de calcul de $\mathcal{N}$, au pire des cas : on peut remarquer que les points 1, 2, 3 et 4 s'effectuent tous en temps polynomial ; donc on a un temps total $\mathcal{O}(p(n))$, où p polynôme.

P ≠ NP ?

- Le langage $HPATH$ décidé par la machine de Turing non-déterministe $\mathcal{N}$ suivante :

  Étant donné l'input $\langle G, s, t\rangle$ (de taille n), où G est un graphe avec m noeuds, et s et t sont des noeuds de G :

  1. écrire à la fin de l'input une liste de m nombres (noeuds) $1 \leq p_1, \ldots, p_m \leq m$; chaque nombre procède d'un choix *non-déterministe* parmi m possibilités
  2. s'il y a une répétition parmi ces nombre, *rejeter* l'input
  3. si $p_1 \neq s$ ou $p_m \neq t$, *rejeter* l'input
  4. pour tout $i = 1, \ldots, m-1$, vérifier que (p_i, p_{i+1}) est une arrête du graphe ; si ce n'est pas le cas, *rejeter* l'input, sinon *accepter* l'input

- Temps de calcul de $\mathcal{N}$, au pire des cas : on peut remarquer que les points 1, 2, 3 et 4 s'effectuent tous en temps polynomial ; donc on a un temps total $\mathcal{O}(p(n))$, où p polynôme.

$\mathbf{P \neq NP\,?}$

- Dans l'exemple précédent, le langage $HPATH$ est décidé par une machine de Turing non-déterministe en temps polynomial $\mathcal{O}(p(n))$.

Définition 11 (classe NPTIME)

On appelle **NPTIME** ou **NP** l'ensemble de tous les langages décidables en temps polynomial par un machine de Turing non-déterministe.

$\mathbf{P \neq NP\,?}$

- Dans l'exemple précédent, le langage $HPATH$ est décidé par une machine de Turing non-déterministe en temps polynomial $\mathcal{O}(p(n))$.

Définition 11 (classe NPTIME)

On appelle **NPTIME** ou **NP** l'ensemble de tous les langages décidables en temps polynomial par un machine de Turing non-déterministe.

$\mathbf{P} \neq \mathbf{NP}$?

- On a vu que $HPATH$ était décidable par une machine de Turing non-déterministe en temps poly, i.e., $HPATH \in \mathbf{NP}$.
- On ne connaît à ce jour aucune manière de décider $HPATH$ par une machine de Turing déterministe en temps polynomial, i.e., on ignore si $HPATH \in \mathbf{P}$.
- Ainsi les problèmes de $\mathbf{P}$ semblent pouvoir être décidables bien plus rapidement que ceux de $\mathbf{NP}$.
- $\mathbf{P}$ = collection des problèmes qui sont Turing-décidables « rapidement ».
- $\mathbf{NP}$ = collection des problèmes dont la validité d'une solution potentielle (générée de manière non-déterministe) peut être décidée « rapidement ».

$\mathbf{P} \neq \mathbf{NP}$?

- On a vu que $HPATH$ était décidable par une machine de Turing non-déterministe en temps poly, i.e., $HPATH \in \mathbf{NP}$.
- On ne connaît à ce jour aucune manière de décider $HPATH$ par une machine de Turing déterministe en temps polynomial, i.e., on ignore si $HPATH \in \mathbf{P}$.
- Ainsi les problèmes de $\mathbf{P}$ semblent pouvoir être décidables bien plus rapidement que ceux de $\mathbf{NP}$.
- $\mathbf{P}$ = collection des problèmes qui sont Turing-décidables « rapidement ».
- $\mathbf{NP}$ = collection des problèmes dont la validité d'une solution potentielle (générée de manière non-déterministe) peut être décidée « rapidement ».

$\mathbf{P} \neq \mathbf{NP}$?

- On a vu que $HPATH$ était décidable par une machine de Turing non-déterministe en temps poly, i.e., $HPATH \in \mathbf{NP}$.
- On ne connaît à ce jour aucune manière de décider $HPATH$ par une machine de Turing déterministe en temps polynomial, i.e., on ignore si $HPATH \in \mathbf{P}$.
- Ainsi les problèmes de $\mathbf{P}$ semblent pouvoir être décidables bien plus rapidement que ceux de $\mathbf{NP}$.
- $\mathbf{P}$ = collection des problèmes qui sont Turing-décidables « rapidement ».
- $\mathbf{NP}$ = collection des problèmes dont la validité d'une solution potentielle (générée de manière non-déterministe) peut être décidée « rapidement ».

$\mathbf{P} \neq \mathbf{NP}$?

- On a vu que $HPATH$ était décidable par une machine de Turing non-déterministe en temps poly, i.e., $HPATH \in \mathbf{NP}$.
- On ne connaît à ce jour aucune manière de décider $HPATH$ par une machine de Turing déterministe en temps polynomial, i.e., on ignore si $HPATH \in \mathbf{P}$.
- Ainsi les problèmes de $\mathbf{P}$ semblent pouvoir être décidables bien plus rapidement que ceux de $\mathbf{NP}$.
- $\mathbf{P}$ = collection des problèmes qui sont Turing-décidables « rapidement ».
- $\mathbf{NP}$ = collection des problèmes dont la validité d'une solution potentielle (générée de manière non-déterministe) peut être décidée « rapidement ».

$\mathbf{P} \neq \mathbf{NP}$?

- On a vu que $HPATH$ était décidable par une machine de Turing non-déterministe en temps poly, i.e., $HPATH \in \mathbf{NP}$.
- On ne connaît à ce jour aucune manière de décider $HPATH$ par une machine de Turing déterministe en temps polynomial, i.e., on ignore si $HPATH \in \mathbf{P}$.
- Ainsi les problèmes de $\mathbf{P}$ semblent pouvoir être décidables bien plus rapidement que ceux de $\mathbf{NP}$.
- $\mathbf{P}$ = collection des problèmes qui sont Turing-décidables « rapidement ».
- $\mathbf{NP}$ = collection des problèmes dont la validité d'une solution potentielle (générée de manière non-déterministe) peut être décidée « rapidement ».

$\mathbf{P \neq NP}$?

- Par définition, on a $\mathbf{P} \subseteq \mathbf{NP}$: en effet, soit $L \in \mathbf{P}$, alors L est décidable en temps poly. par une MT dét. M, et donc L est également décidable en temps poly par la même MT non-dét. M (pour laquelle le non-dét. est réduit à sa forme dégénérée), i.e. $L \in \mathbf{NP}$.
- Par contre, on ignore si $\mathbf{P} \subsetneq \mathbf{NP}$ ou si $\mathbf{P} = \mathbf{NP}$.
- On conjecture généralement que $\mathbf{P} \subsetneq \mathbf{NP}$, mais cela n'est pas prouvé, et constitue un des problèmes ouverts les plus importants en informatique théorique (1 Mo de Dollars).
- Les conséquences d'une preuve de $\mathbf{P} = \mathbf{NP}$ seraient doyennes : car beaucoup de processus de sécurité et de cryptage reposent sur des problèmes NP-complets.

$\mathbf{P} \neq \mathbf{NP}$?

- Par définition, on a $\mathbf{P} \subseteq \mathbf{NP}$: en effet, soit $L \in \mathbf{P}$, alors L est décidable en temps poly. par une MT dét. $\mathcal{M}$, et donc L est également décidable en temps poly par la même MT non-dét. $\mathcal{M}$ (pour laquelle le non-dét. est réduit à sa forme dégénérée), i.e. $L \in \mathbf{NP}$.
- Par contre, on ignore si $\mathbf{P} \subsetneq \mathbf{NP}$ ou si $\mathbf{P} = \mathbf{NP}$.
- On conjecture généralement que $\mathbf{P} \subsetneq \mathbf{NP}$, mais cela n'est pas prouvé, et constitue un des problèmes ouverts les plus importants en informatique théorique (1 Mo de dollars).
- Les conséquences d'une preuve de $\mathbf{P} \neq \mathbf{NP}$ seraient énormes, car beaucoup de processus de sécurité informatique reposent sur des problèmes NP-complets.

$\mathbf{P} \neq \mathbf{NP}$?

- Par définition, on a $\mathbf{P} \subseteq \mathbf{NP}$: en effet, soit $L \in \mathbf{P}$, alors L est décidable en temps poly. par une MT dét. $\mathcal{M}$, et donc L est également décidable en temps poly par la même MT non-dét. $\mathcal{M}$ (pour laquelle le non-dét. est réduit à sa forme dégénérée), i.e. $L \in \mathbf{NP}$.
- Par contre, on ignore si $\mathbf{P} \subsetneq \mathbf{NP}$ ou si $\mathbf{P} = \mathbf{NP}$.
- On conjecture généralement que $\mathbf{P} \subsetneq \mathbf{NP}$, mais cela n'est pas prouvé, et constitue un des problèmes ouverts les plus importants en informatique théorique (1 Mo de Dollars).
- Les conséquences d'une preuve de $\mathbf{P} \neq \mathbf{NP}$ seraient énormes, car beaucoup de processus de sécurité et de cryptage reposent sur des problèmes NP-complets.

$\mathbf{P} \neq \mathbf{NP}$?

- Par définition, on a $\mathbf{P} \subseteq \mathbf{NP}$: en effet, soit $L \in \mathbf{P}$, alors L est décidable en temps poly. par une MT dét. $\mathcal{M}$, et donc L est également décidable en temps poly par la même MT non-dét. $\mathcal{M}$ (pour laquelle le non-dét. est réduit à sa forme dégénérée), i.e. $L \in \mathbf{NP}$.
- Par contre, on ignore si $\mathbf{P} \subsetneq \mathbf{NP}$ ou si $\mathbf{P} = \mathbf{NP}$.
- On conjecture généralement que $\mathbf{P} \subsetneq \mathbf{NP}$, mais cela n'est pas prouvé, et constitue un des problèmes ouverts les plus importants en informatique théorique (1 Mo de Dollars).
- Les conséquences d'une preuve de $\mathbf{P} \neq \mathbf{NP}$ seraient dévastatrices, car beaucoup de processus de sécurité et de cryptage repose sur des problème $\mathbf{NP}$-complets.

$\mathbf{P} \neq \mathbf{NP}$?

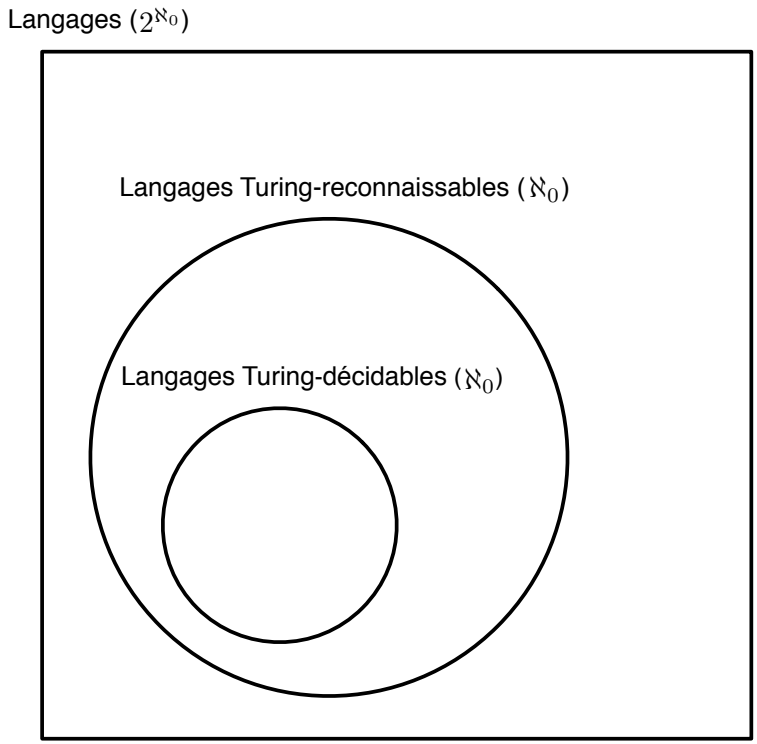
- Par définition, on a $\mathbf{P} \subseteq \mathbf{NP}$: en effet, soit $L \in \mathbf{P}$, alors L est décidable en temps poly. par une MT dét. $\mathcal{M}$, et donc L est également décidable en temps poly par la même MT non-dét. $\mathcal{M}$ (pour laquelle le non-dét. est réduit à sa forme dégénérée), i.e. $L \in \mathbf{NP}$.
- Par contre, on ignore si $\mathbf{P} \subsetneq \mathbf{NP}$ ou si $\mathbf{P} = \mathbf{NP}$.
- On conjecture généralement que $\mathbf{P} \subsetneq \mathbf{NP}$, mais cela n'est pas prouvé, et constitue un des problèmes ouverts les plus importants en informatique théorique (1 Mo de Dollars).
- Les conséquences d'une preuve de $\mathbf{P} \neq \mathbf{NP}$ seraient dévastatrices, car beaucoup de processus de sécurité et de cryptage repose sur des problème $\mathbf{NP}$-complets.

$\mathbf{P} \neq \mathbf{NP}$?

- Par définition, on a $\mathbf{P} \subseteq \mathbf{NP}$: en effet, soit $L \in \mathbf{P}$, alors L est décidable en temps poly. par une MT dét. $\mathcal{M}$, et donc L est également décidable en temps poly par la même MT non-dét. $\mathcal{M}$ (pour laquelle le non-dét. est réduit à sa forme dégénérée), i.e. $L \in \mathbf{NP}$.
- Par contre, on ignore si $\mathbf{P} \subsetneq \mathbf{NP}$ ou si $\mathbf{P} = \mathbf{NP}$.
- On conjecture généralement que $\mathbf{P} \subsetneq \mathbf{NP}$, mais cela n'est pas prouvé, et constitue un des problèmes ouverts les plus importants en informatique théorique (1 Mo de Dollars).
- Les conséquences d'une preuve de $\mathbf{P} \neq \mathbf{NP}$ seraient dévastatrices, car beaucoup de processus de sécurité et de cryptage repose sur des problème $\mathbf{NP}$-complets.

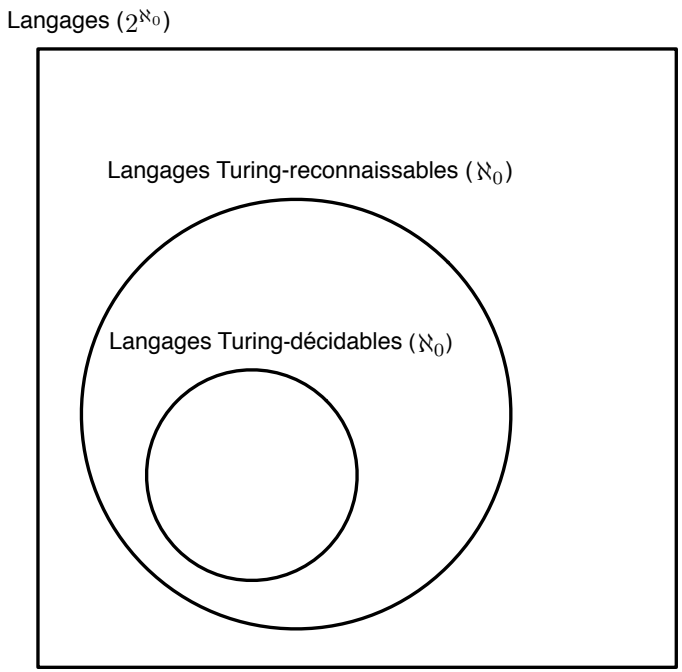
Indécidabilité

- Nous allons montré qu'il existe une infinité non dénombrable de problèmes qui sont non Turing-reconnaissables et non Turing-décidables – ou *Turing-indécidables*.



Indécidabilité

- Ces théorèmes de limitation constituent, via la thèse de Church-Turing, une restriction drastique de la notion générale de calculabilité (pas si simple en réalité, notions alternatives de calculabilité, etc.).



Indécidabilité

- Par définition, une machine de Turing $\mathcal{M}$ est un tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

dont chaque élément est une suite finie de symboles.

- Ainsi, formellement, une machine de Turing $\mathcal{M}$ n'est rien d'autre qu'une suite finie de symboles sur un alphabet fini.
- Si l'on code chaque symbole de notre alphabet par un entier, une machine de Turing $\mathcal{M}$ peut-être codée par une suite d'entiers, qui peut ensuite elle-même être codée de manière non-ambiguë par un unique entier, noté $\langle \mathcal{M} \rangle$.
- Ainsi il n'y a pas plus de MT que d'entiers, i.e. $|\mathcal{MT}| \leq \aleph_0$. D'autre part, il existe facilement $\aleph_0$ MT différentes, donc $|\mathcal{MT}| \geq \aleph_0$. Ainsi $|\mathcal{MT}| = \aleph_0$.

Indécidabilité

- Par définition, une machine de Turing $\mathcal{M}$ est un tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

dont chaque élément est une suite finie de symboles.

- Ainsi, formellement, une machine de Turing $\mathcal{M}$ n'est rien d'autre qu'une suite finie de symboles sur un alphabet fini.
- Si l'on code chaque symbole de notre alphabet par un entier, une machine de Turing $\mathcal{M}$ peut-être codée par une suite d'entiers, qui peut ensuite elle-même être codée de manière non-ambiguë par un unique entier, noté $\langle \mathcal{M} \rangle$.
- Ainsi il n'y a pas plus de MT que d'entiers, i.e. $|\mathcal{MT}| \leq \aleph_0$. D'autre part, il existe facilement une infinité de MT, donc $|\mathcal{MT}| \geq \aleph_0$. Ainsi $|\mathcal{MT}| = \aleph_0$.

Indécidabilité

- Par définition, une machine de Turing $\mathcal{M}$ est un tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

dont chaque élément est une suite finie de symboles.

- Ainsi, formellement, une machine de Turing $\mathcal{M}$ n'est rien d'autre qu'une suite finie de symboles sur un alphabet fini.
- Si l'on code chaque symbole de notre alphabet par un entier, une machine de Turing $\mathcal{M}$ peut-être codée par une suite d'entiers, qui peut ensuite elle-même être codée de manière non-ambiguë par un unique entier, noté $\langle \mathcal{M} \rangle$.
- Ainsi il n'y a pas plus de MT que d'entiers, i.e. $|\mathcal{MT}| \leq \aleph_0$. D'autre part, il existe (au moins) une MT pour chaque entier, donc $|\mathcal{MT}| \geq \aleph_0$. Ainsi $|\mathcal{MT}| = \aleph_0$.

Indécidabilité

- Par définition, une machine de Turing $\mathcal{M}$ est un tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

dont chaque élément est une suite finie de symboles.

- Ainsi, formellement, une machine de Turing $\mathcal{M}$ n'est rien d'autre qu'une suite finie de symboles sur un alphabet fini.
- Si l'on code chaque symbole de notre alphabet par un entier, une machine de Turing $\mathcal{M}$ peut-être codée par une suite d'entiers, qui peut ensuite elle-même être codée de manière non-ambiguë par un unique entier, noté $\langle \mathcal{M} \rangle$.
- Ainsi il n'y a pas plus de MT que d'entiers, i.e. $|\mathcal{MT}| \leq \aleph_0$.
  D'autre part, il est très facile d'exhiber $\aleph_0$ MT différentes, donc $|\mathcal{MT}| \geq \aleph_0$. Ainsi $|\mathcal{MT}| = \aleph_0$.

Indécidabilité

- Par définition, une machine de Turing $\mathcal{M}$ est un tuple
$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$
dont chaque élément est une suite finie de symboles.
- Ainsi, formellement, une machine de Turing $\mathcal{M}$ n'est rien d'autre qu'une suite finie de symboles sur un alphabet fini.
- Si l'on code chaque symbole de notre alphabet par un entier, une machine de Turing $\mathcal{M}$ peut-être codée par une suite d'entiers, qui peut ensuite elle-même être codée de manière non-ambiguë par un unique entier, noté $\langle \mathcal{M} \rangle$.
- Ainsi il n'y a pas plus de MT que d'entiers, i.e. $|\mathcal{MT}| \leq \aleph_0$.
D'autre part, il est très facile d'exhiber $\aleph_0$ MT différentes, donc $|\mathcal{MT}| \geq \aleph_0$. Ainsi $|\mathcal{MT}| = \aleph_0$.

Indécidabilité

- Par définition, une machine de Turing $\mathcal{M}$ est un tuple
$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$
dont chaque élément est une suite finie de symboles.
- Ainsi, formellement, une machine de Turing $\mathcal{M}$ n'est rien d'autre qu'une suite finie de symboles sur un alphabet fini.
- Si l'on code chaque symbole de notre alphabet par un entier, une machine de Turing $\mathcal{M}$ peut-être codée par une suite d'entiers, qui peut ensuite elle-même être codée de manière non-ambiguë par un unique entier, noté $\langle \mathcal{M} \rangle$.
- Ainsi il n'y a pas plus de MT que d'entiers, i.e. $|\mathcal{MT}| \leq \aleph_0$. D'autre part, il est très facile d'exhiber $\aleph_0$ MT différentes, donc $|\mathcal{MT}| \geq \aleph_0$. Ainsi $|\mathcal{MT}| = \aleph_0$.

Indécidabilité

- Par définition, une machine de Turing $\mathcal{M}$ est un tuple
$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$
dont chaque élément est une suite finie de symboles.
- Ainsi, formellement, une machine de Turing $\mathcal{M}$ n'est rien d'autre qu'une suite finie de symboles sur un alphabet fini.
- Si l'on code chaque symbole de notre alphabet par un entier, une machine de Turing $\mathcal{M}$ peut-être codée par une suite d'entiers, qui peut ensuite elle-même être codée de manière non-ambiguë par un unique entier, noté $\langle \mathcal{M} \rangle$.
- Ainsi il n'y a pas plus de MT que d'entiers, i.e. $|\mathcal{MT}| \leq \aleph_0$. D'autre part, il est très facile d'exhiber $\aleph_0$ MT différentes, donc $|\mathcal{MT}| \geq \aleph_0$. Ainsi $|\mathcal{MT}| = \aleph_0$.

Indécidabilité

Lemme 12

- *Il y a $\aleph_0$ langages Turing-reconnaissables ;*
- *Il y a $\aleph_0$ langages Turing-décidables.*

Preuve :

Il y autant de langages Turing-reconnaissables et Turing-décidables qu'il y a de MT pour les reconnaître et les décider, respectivement, à savoir $\aleph_0$ (par l'argument précédent). □

Indécidabilité

Lemme 12

- *Il y a $\aleph_0$ langages Turing-reconnaissables ;*
- *Il y a $\aleph_0$ langages Turing-décidables.*

Preuve :

Il y autant de langages Turing-reconnaissables et Turing-décidables qu'il y a de MT pour les reconnaître et les décider, respectivement, à savoir $\aleph_0$ (par l'argument précédent). □

Indécidabilité

Lemme 13

Soit Σ un alphabet fini (avec au moins deux lettres). Alors il y a $2^{\aleph_0}$ langages possibles sur Σ.

Preuve :

- On suppose (sans perte de généralité) que $\Sigma = \{0, 1\}$. Tout langage L sur Σ peut-être identifié de manière bijective par une suite infinie de bits χ_L de la manière suivante :

$$\begin{array}{lccccccccc} \Sigma^* & = & \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & \ldots & \} \\ L & = & \{ & & & 1, & & 01, & 10, & & 000, & \ldots & \} \\ \chi_L & = & & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & \ldots & \end{array}$$

Indécidabilité

Lemme 13

Soit Σ un alphabet fini (avec au moins deux lettres). Alors il y a $2^{\aleph_0}$ langages possibles sur Σ.

Preuve :

- On suppose (sans perte de généralité) que $\Sigma = \{0, 1\}$. Tout langage L sur Σ peut-être identifié de manière bijective par une suite infinie de bits χ_L de la manière suivante :

$$\begin{array}{rcccccccccc} \Sigma^* & = & \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & \ldots & \} \\ L & = & \{ & & 1, & & & 01, & 10, & & 000, & \ldots & \} \\ \chi_L & = & & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & \ldots & \end{array}$$

Indécidabilité

Lemme 13

Soit Σ un alphabet fini (avec au moins deux lettres). Alors il y a $2^{\aleph_0}$ langages possibles sur Σ.

Preuve :

- On suppose (sans perte de généralité) que $\Sigma = \{0, 1\}$. Tout langage L sur Σ peut-être identifié de manière bijective par une suite infinie de bits χ_L de la manière suivante :

$$\begin{array}{lccccccccccc} \Sigma^* & = & \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & \ldots & \} \\ L & = & \{ & & 1, & & & 01, & 10, & & 000, & \ldots & \} \\ \chi_L & = & & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & \ldots & \end{array}$$

Indécidabilité

Lemme 13

Soit Σ un alphabet fini (avec au moins deux lettres). Alors il y a $2^{\aleph_0}$ langages possibles sur Σ.

Preuve :

- On suppose (sans perte de généralité) que $\Sigma = \{0, 1\}$. Tout langage L sur Σ peut-être identifié de manière bijective par une suite infinie de bits χ_L de la manière suivante :

$$\begin{array}{lcccccccccc} \Sigma^* & = & \{ & \varepsilon\,, & 0\,, & 1\,, & 00\,, & 01\,, & 10\,, & 11\,, & 000\,, & \ldots & \} \\ L & = & \{ & & 1\,, & & 01\,, & 10\,, & & & 000\,, & \ldots & \} \\ \chi_L & = & & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & \ldots & \end{array}$$

Indécidabilité

Lemme 13

Soit Σ un alphabet fini (avec au moins deux lettres). Alors il y a $2^{\aleph_0}$ langages possibles sur Σ.

Preuve :

- On suppose (sans perte de généralité) que $\Sigma = \{0, 1\}$. Tout langage L sur Σ peut-être identifié de manière bijective par une suite infinie de bits χ_L de la manière suivante :

$$\begin{array}{rcccccccccccc} \Sigma^* & = & \{ & \varepsilon\,, & 0\,, & 1\,, & 00\,, & 01\,, & 10\,, & 11\,, & 000\,, & \ldots & \} \\ L & = & \{ & & & 1\,, & & 01\,, & 10\,, & & 000\,, & \ldots & \} \\ \chi_L & = & & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & \ldots & \end{array}$$

Indécidabilité

Lemme 13

Soit Σ un alphabet fini (avec au moins deux lettres). Alors il y a $2^{\aleph_0}$ langages possibles sur Σ.

Preuve :

- On suppose (sans perte de généralité) que $\Sigma = \{0, 1\}$. Tout langage L sur Σ peut-être identifié de manière bijective par une suite infinie de bits χ_L de la manière suivante :

$$\begin{array}{lccccccccccc} \Sigma^* & = & \{ & \varepsilon\,, & 0\,, & 1\,, & 00\,, & 01\,, & 10\,, & 11\,, & 000\,, & \ldots & \} \\ L & = & \{ & & & 1\,, & & 01\,, & 10\,, & & 000\,, & \ldots & \} \\ \chi_L & = & & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & \ldots & \end{array}$$

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{array}{lccccccc}
\text{suite n° } 0 : s_0 = & & 1 & 1 & 0 & 1 & \cdots \\
\text{suite n° } 1 : s_1 = & 1 & & 1 & 1 & 1 & \cdots \\
\text{suite n° } 2 : s_2 = & 0 & 0 & & 1 & 0 & \cdots \\
\text{suite n° } 3 : s_3 = & 0 & 0 & 0 & & 0 & \cdots \\
\text{suite n° } 4 : s_4 = & 1 & 0 & 0 & 1 & & \cdots \\
\vdots & & & & & & \ddots
\end{array}$$

- On considère la suite $s = 1\,1\,0\,0\,1\cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^\aleph$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{array}{lcccccc} \text{suite n° } 0 : s_0 = & 0 & 1 & 1 & 0 & 1 & \cdots \\ \text{suite n° } 1 : s_1 = & 1 & & 1 & 1 & 1 & \cdots \\ \text{suite n° } 2 : s_2 = & 0 & 0 & & 1 & 0 & \cdots \\ \text{suite n° } 3 : s_3 = & 0 & 0 & 0 & & 0 & \cdots \\ \text{suite n° } 4 : s_4 = & 1 & 0 & 0 & 1 & & \cdots \\ \vdots & & & & & & \end{array}$$

- On considère la suite $s = 1\,1\,0\,0\,1\cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^\aleph$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{array}{lccccccc}
\text{suite n° } 0 : s_0 = & 0 & 1 & 1 & 0 & 1 & \cdots \\
\text{suite n° } 1 : s_1 = & 1 & 0 & 1 & 1 & 1 & \cdots \\
\text{suite n° } 2 : s_2 = & 0 & 0 & & 1 & 0 & \cdots \\
\text{suite n° } 3 : s_3 = & 0 & 0 & 0 & & 0 & \cdots \\
\text{suite n° } 4 : s_4 = & 1 & 0 & 0 & 1 & & \cdots \\
\vdots & & & & & &
\end{array}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^\aleph$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{aligned}
\text{suite n° } 0 : s_0 &= 0\ 1\ 1\ 0\ 1\ \cdots \\
\text{suite n° } 1 : s_1 &= 1\ 0\ 1\ 1\ 1\ \cdots \\
\text{suite n° } 2 : s_2 &= 0\ 0\ 1\ 1\ 0\ \cdots \\
\text{suite n° } 3 : s_3 &= 0\ 0\ 0\ \ 0\ \cdots \\
\text{suite n° } 4 : s_4 &= 1\ 0\ 0\ 1\ \ \cdots \\
&\vdots
\end{aligned}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^\aleph$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{aligned}
\text{suite n° } 0 : s_0 &= 0\ 1\ 1\ 0\ 1\ \cdots \\
\text{suite n° } 1 : s_1 &= 1\ 0\ 1\ 1\ 1\ \cdots \\
\text{suite n° } 2 : s_2 &= 0\ 0\ 1\ 1\ 0\ \cdots \\
\text{suite n° } 3 : s_3 &= 0\ 0\ 0\ 1\ 0\ \cdots \\
\text{suite n° } 4 : s_4 &= 1\ 0\ 0\ 1\ \ \cdots \\
&\vdots
\end{aligned}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^\aleph$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{array}{lllllll} \text{suite n° } 0 : s_0 = & 0 & 1 & 1 & 0 & 1 & \cdots \\ \text{suite n° } 1 : s_1 = & 1 & 0 & 1 & 1 & 1 & \cdots \\ \text{suite n° } 2 : s_2 = & 0 & 0 & 1 & 1 & 0 & \cdots \\ \text{suite n° } 3 : s_3 = & 0 & 0 & 0 & 1 & 0 & \cdots \\ \text{suite n° } 4 : s_4 = & 1 & 0 & 0 & 1 & 0 & \cdots \end{array}$$

$$\vdots$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^{\aleph}$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{array}{lllllll} \text{suite n° } 0 : s_0 = & 0 & 1 & 1 & 0 & 1 & \cdots \\ \text{suite n° } 1 : s_1 = & 1 & 0 & 1 & 1 & 1 & \cdots \\ \text{suite n° } 2 : s_2 = & 0 & 0 & 1 & 1 & 0 & \cdots \\ \text{suite n° } 3 : s_3 = & 0 & 0 & 0 & 1 & 0 & \cdots \\ \text{suite n° } 4 : s_4 = & 1 & 0 & 0 & 1 & 0 & \cdots \\ \vdots & & & & & & \end{array}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^\aleph$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{aligned}
\text{suite n° } 0 : s_0 &= 0\ 1\ 1\ 0\ 1\ \cdots \\
\text{suite n° } 1 : s_1 &= 1\ 0\ 1\ 1\ 1\ \cdots \\
\text{suite n° } 2 : s_2 &= 0\ 0\ 1\ 1\ 0\ \cdots \\
\text{suite n° } 3 : s_3 &= 0\ 0\ 0\ 1\ 0\ \cdots \\
\text{suite n° } 4 : s_4 &= 1\ 0\ 0\ 1\ 0\ \cdots \\
&\vdots
\end{aligned}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^\aleph$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{aligned}
\text{suite n° } 0 : s_0 &= 0\ 1\ 1\ 0\ 1\ \cdots \\
\text{suite n° } 1 : s_1 &= 1\ 0\ 1\ 1\ 1\ \cdots \\
\text{suite n° } 2 : s_2 &= 0\ 0\ 1\ 1\ 0\ \cdots \\
\text{suite n° } 3 : s_3 &= 0\ 0\ 0\ 1\ 0\ \cdots \\
\text{suite n° } 4 : s_4 &= 1\ 0\ 0\ 1\ 0\ \cdots \\
&\vdots
\end{aligned}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^{\aleph}$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{aligned}
\text{suite n° } 0 : s_0 &= 0\ 1\ 1\ 0\ 1\ \cdots \\
\text{suite n° } 1 : s_1 &= 1\ 0\ 1\ 1\ 1\ \cdots \\
\text{suite n° } 2 : s_2 &= 0\ 0\ 1\ 1\ 0\ \cdots \\
\text{suite n° } 3 : s_3 &= 0\ 0\ 0\ 1\ 0\ \cdots \\
\text{suite n° } 4 : s_4 &= 1\ 0\ 0\ 1\ 0\ \cdots \\
&\vdots
\end{aligned}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^{\aleph}$. □

Indécidabilité

- On montre par *diagonalisation* que l'ensemble des suites infinies de bits, et donc l'ensemble des langages, est non dénombrable. Par l'absurde, supposons cet ensemble dénombrable :

$$\begin{aligned}
\text{suite n° } 0 : s_0 &= \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad \cdots \\
\text{suite n° } 1 : s_1 &= \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad \cdots \\
\text{suite n° } 2 : s_2 &= \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad \cdots \\
\text{suite n° } 3 : s_3 &= \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad \cdots \\
\text{suite n° } 4 : s_4 &= \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad \cdots \\
&\vdots
\end{aligned}$$

- On considère la suite $s = 1\ 1\ 0\ 0\ 1\ \cdots$. Par construction, on a $s \neq s_i, \forall i \in \mathbb{N}$, i.e. s ne fait pas partie du dénombrement, contradiction. On peut montrer que $|\{0,1\}^\omega| = 2_0^{\aleph}$. □

Indécidabilité

Corollaire 14

- *Il y a $2^{\aleph_0}$ langages non Turing-reconnaissables ;*
- *Il y a $2^{\aleph_0}$ langages non Turing-décidables.*

Preuve :

Une conséquence directe des lemmes 12 et 13. □

Indécidabilité

Corollaire 14

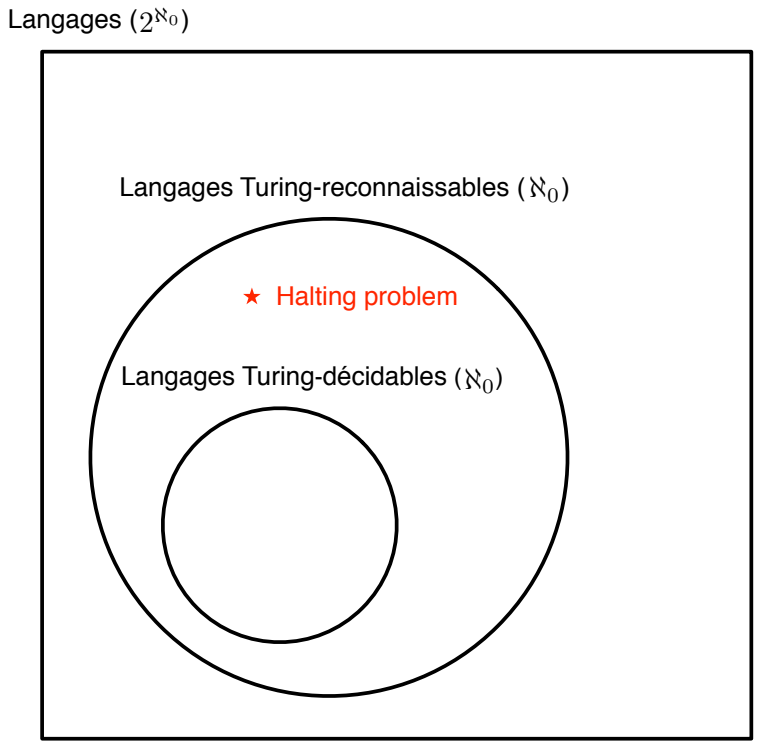
- *Il y a $2^{\aleph_0}$ langages non Turing-reconnaissables ;*
- *Il y a $2^{\aleph_0}$ langages non Turing-décidables.*

Preuve :

Une conséquence directe des lemmes 12 et 13. □

Problème de l'arrêt

- Nous allons exhibé un problème – *le problème de l'arrêt* ou *Halting problem* – qui est Turing-reconnaissable, mais qui n'est pas Turing-décidable.



Problème de l'arrêt

- Le *problème de l'arrêt* a joué un rôle historique crucial, car il a permis de montrer l'indécidabilité du *Entscheidungsproblem* (ou problème de décision) de Hilbert.
- *Problème de l'arrêt :* étant donné un programme P et un input u de ce programme, est-il possible de décider algorithmiquement si P va s'arrêter sur l'input u ou non ?
- *Problème de l'arrêt :* existe-t-il une machine de Turing $\mathcal{H}$ qui, étant donné le code d'une machine de Turing $\mathcal{M}$ et le code d'un input u, permet de décider si $\mathcal{M}$ va s'arrêter sur l'input u ou non ?

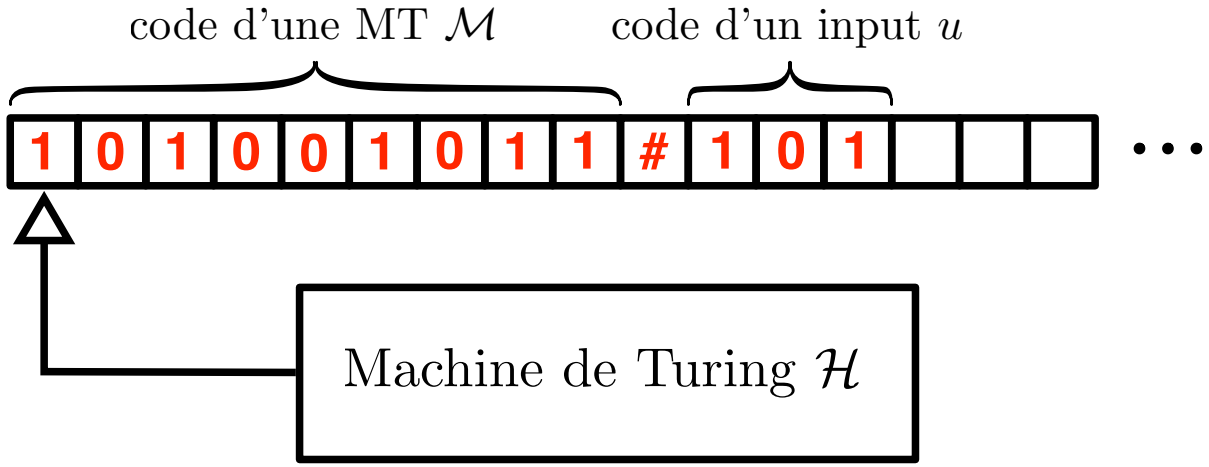
Problème de l'arrêt

- Le *problème de l'arrêt* a joué un rôle historique crucial, car il a permis de montrer l'indécidabilité du *Entscheidungsproblem* (ou problème de décision) de Hilbert.
- *Problème de l'arrêt :* étant donné un programme P et un input u de ce programme, est-il possible de décider algorithmiquement si P va s'arrêter sur l'input u ou non ?
- *Problème de l'arrêt :* existe-t-il une machine de Turing $\mathcal{H}$ qui, étant donné le code d'une machine de Turing $\mathcal{M}$ et le code d'un input u, permet de décider si $\mathcal{M}$ va s'arrêter sur l'input u ou non ?

Problème de l'arrêt

- Le *problème de l'arrêt* a joué un rôle historique crucial, car il a permis de montrer l'indécidabilité du *Entscheidungsproblem* (ou problème de décision) de Hilbert.
- *Problème de l'arrêt :* étant donné un programme P et un input u de ce programme, est-il possible de décider algorithmiquement si P va s'arrêter sur l'input u ou non ?
- *Problème de l'arrêt :* existe-t-il une machine de Turing $\mathcal{H}$ qui, étant donné le code d'une machine de Turing $\mathcal{M}$ et le code d'un input u, permet de décider si $\mathcal{M}$ va s'arrêter sur l'input u ou non ?

Problème de l'arrêt



- l'input $\langle \mathcal{M}, u \rangle$ est *accepté* par $\mathcal{H}$ si $\mathcal{M}(u)$ s'arrête (soit dans l'état acceptant soit dans l'état rejetant)
- l'input $\langle \mathcal{M}, u \rangle$ est *rejeté* par $\mathcal{H}$ si $\mathcal{M}(u)$ ne s'arrête jamais, i.e. boucle indéfiniment

Problème de l'arrêt

- En terme de langage, le problème de l'arrêt correspond donc à l'ensemble des codes des machines de Turing $\mathcal{M}$ et inputs u tels que $\mathcal{M}(u)$ s'arrête (dans un état acceptant ou rejetant), i.e. $\mathcal{M}$ ne boucle pas infiniment.

Définition 15 (Problème de l'arrêt)

Le *problème de l'arrêt* est le langage

$$L_{\mathcal{H}} = \{\langle \mathcal{M}, u \rangle : \mathcal{M} \text{ est une MT qui s'arrête sur l'input } u\}$$

où $\langle \ . \ \rangle$ correspond à une fonction de codage (e.g. binaire).

Problème de l'arrêt

- En terme de langage, le problème de l'arrêt correspond donc à l'ensemble des codes des machines de Turing $\mathcal{M}$ et inputs u tels que $\mathcal{M}(u)$ s'arrête (dans un état acceptant ou rejetant), i.e. $\mathcal{M}$ ne boucle pas infiniment.

Définition 15 (Problème de l'arrêt)

Le *problème de l'arrêt* est le langage

$$L_{\mathcal{H}} = \{\langle \mathcal{M}, u \rangle : \mathcal{M} \text{ est une MT qui s'arrête sur l'input } u\}$$

où $\langle \ . \ \rangle$ correspond à une fonction de codage (e.g. binaire).

Problème de l'arrêt

Lemme 16

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, est Turing-reconnaissable.

Preuve :

- Considérons une machine de Turing *universelle* $\mathcal{M}_U$ qui peut simuler le comportement de toute machine de Turing $\mathcal{M}$ et input w donnés en input codé sous la forme $\langle \mathcal{M}, w \rangle$.
- Modifions la MT $\mathcal{M}_U$ en la MT $\mathcal{M}'_U$ comme suit :
- Dans ce cas, $\mathcal{M}'_U$ accepte toujours $\langle \mathcal{M}, w \rangle$ si $\mathcal{M}$ s'arrête (dans l'état q_{accept}) sur w, et $\mathcal{M}'_U$ ne s'arrête pas

Problème de l'arrêt

Lemme 16

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, est Turing-reconnaissable.

Preuve :

- Considérons une machine de Turing *universelle* $\mathcal{M}_U$ qui peut simuler le comportement de toute machine de Turing $\mathcal{M}$ et input u donnés en input codé sous la forme $\langle \mathcal{M}, u \rangle$.
- Modifions la MT $\mathcal{M}_U$ en la MT $\mathcal{M}'_U$ comme suit :
  - Si $\mathcal{M}_U$ accepte l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ rejette l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ ne s'arrête pas sur $\langle \mathcal{M}, u \rangle$, $\mathcal{M}'_U$ ne s'arrête pas.
- Dans ce cas, $\mathcal{M}'_U$ accepte tous les $\langle \mathcal{M}, u \rangle$ tels que $\mathcal{M}(u)$ s'arrête (dans q_{accept} ou q_{reject}), i.e. $\mathcal{M}'_U$ reconnaît $L_{\mathcal{H}}$. □

Problème de l'arrêt

Lemme 16

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, est Turing-reconnaissable.

Preuve :

- Considérons une machine de Turing *universelle* $\mathcal{M}_U$ qui peut simuler le comportement de toute machine de Turing $\mathcal{M}$ et input u donnés en input codé sous la forme $\langle \mathcal{M}, u \rangle$.
- Modifions la MT $\mathcal{M}_U$ en la MT $\mathcal{M}'_U$ comme suit :
  - Si $\mathcal{M}_U$ accepte l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ rejète l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ ne s'arrête pas sur $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ non plus.
- Dans ce cas, $\mathcal{M}'_U$ accepte tous les $\langle \mathcal{M}, u \rangle$ tels que $\mathcal{M}(u)$ s'arrête (dans q_{accept} ou q_{reject}), i.e. $\mathcal{M}'_U$ reconnaît $L_{\mathcal{H}}$. □

Problème de l'arrêt

Lemme 16

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, est Turing-reconnaissable.

Preuve :

- Considérons une machine de Turing *universelle* $\mathcal{M}_U$ qui peut simuler le comportement de toute machine de Turing $\mathcal{M}$ et input u donnés en input codé sous la forme $\langle \mathcal{M}, u \rangle$.
- Modifions la MT $\mathcal{M}_U$ en la MT $\mathcal{M}'_U$ comme suit :
  - Si $\mathcal{M}_U$ accepte l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ rejète l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ ne s'arrête pas sur $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ non plus.
- Dans ce cas, $\mathcal{M}'_U$ accepte tous les $\langle \mathcal{M}, u \rangle$ tels que $\mathcal{M}(u)$ s'arrête (dans q_{accept} ou q_{reject}), i.e. $\mathcal{M}'_U$ reconnaît $L_{\mathcal{H}}$. □

Problème de l'arrêt

Lemme 16

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, est Turing-reconnaissable.

Preuve :

- Considérons une machine de Turing *universelle* $\mathcal{M}_U$ qui peut simuler le comportement de toute machine de Turing $\mathcal{M}$ et input u donnés en input codé sous la forme $\langle \mathcal{M}, u \rangle$.
- Modifions la MT $\mathcal{M}_U$ en la MT $\mathcal{M}'_U$ comme suit :
  - Si $\mathcal{M}_U$ accepte l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ rejète l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ ne s'arrête pas sur $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ non plus.
- Dans ce cas, $\mathcal{M}'_U$ accepte tous les $\langle \mathcal{M}, u \rangle$ tels que $\mathcal{M}(u)$ s'arrête (dans q_{accept} ou q_{reject}), i.e. $\mathcal{M}'_U$ reconnaît $L_{\mathcal{H}}$. □

Problème de l'arrêt

Lemme 16

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, est Turing-reconnaissable.

Preuve :

- Considérons une machine de Turing *universelle* $\mathcal{M}_U$ qui peut simuler le comportement de toute machine de Turing $\mathcal{M}$ et input u donnés en input codé sous la forme $\langle \mathcal{M}, u \rangle$.
- Modifions la MT $\mathcal{M}_U$ en la MT $\mathcal{M}'_U$ comme suit :
  - Si $\mathcal{M}_U$ accepte l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ rejète l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ ne s'arrête pas sur $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ non plus.
- Dans ce cas, $\mathcal{M}'_U$ accepte tous les $\langle \mathcal{M}, u \rangle$ tels que $\mathcal{M}(u)$ s'arrête (dans q_{accept} ou q_{reject}), i.e. $\mathcal{M}'_U$ reconnaît $L_{\mathcal{H}}$. □

Problème de l'arrêt

Lemme 16

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, est Turing-reconnaissable.

Preuve :

- Considérons une machine de Turing *universelle* $\mathcal{M}_U$ qui peut simuler le comportement de toute machine de Turing $\mathcal{M}$ et input u donnés en input codé sous la forme $\langle \mathcal{M}, u \rangle$.
- Modifions la MT $\mathcal{M}_U$ en la MT $\mathcal{M}'_U$ comme suit :
  - Si $\mathcal{M}_U$ accepte l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ rejète l'input $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ accepte $\langle \mathcal{M}, u \rangle$.
  - Si $\mathcal{M}_U$ ne s'arrête pas sur $\langle \mathcal{M}, u \rangle$, alors $\mathcal{M}'_U$ non plus.
- Dans ce cas, $\mathcal{M}'_U$ accepte tous les $\langle \mathcal{M}, u \rangle$ tels que $\mathcal{M}(u)$ s'arrête (dans q_{accept} ou q_{reject}), i.e. $\mathcal{M}'_U$ reconnaît $L_{\mathcal{H}}$. □

Problème de l'arrêt

Théorème 17 (Turing 1936)

Le problème de l'arrêt, i.e. le langage $L_{\mathcal{H}}$, n'est pas Turing-décidable.

Problème de l'arrêt

Preuve :

- Par l'absurde, supposons $L_{\mathcal{H}}$ Turing-décidable par une MT $\mathcal{H}$, i.e. :

$$\mathcal{H}(\langle \mathcal{M}, u \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{M}(u) \text{ s'arrête} \\ q_{reject} & \text{si } \mathcal{M}(u) \text{ ne s'arrête pas.} \end{cases}$$

- **Remarque :** le code $\langle \mathcal{M} \rangle$ est une suite de bits, il peut donc faire office d'input !
- On modifie légèrement le programme de $\mathcal{H}$ pour obtenir la MT $\mathcal{H}'$ suivante :

$$\mathcal{H}'(\langle \mathcal{M} \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{accept} \\ q_{reject} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{reject}. \end{cases}$$

Problème de l'arrêt

Preuve :

- Par l'absurde, supposons $L_{\mathcal{H}}$ Turing-décidable par une MT $\mathcal{H}$, i.e. :

$$\mathcal{H}(\langle \mathcal{M}, u \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{M}(u) \text{ s'arrête} \\ q_{reject} & \text{si } \mathcal{M}(u) \text{ ne s'arrête pas.} \end{cases}$$

- **Remarque :** le code $\langle \mathcal{M} \rangle$ est une suite de bits, il peut donc faire office d'input !
- On modifie légèrement le programme de $\mathcal{H}$ pour obtenir la MT $\mathcal{H}'$ suivante :

$$\mathcal{H}'(\langle \mathcal{M} \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{accept} \\ q_{reject} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{reject}. \end{cases}$$

Problème de l'arrêt

Preuve :

- Par l'absurde, supposons $L_{\mathcal{H}}$ Turing-décidable par une MT $\mathcal{H}$, i.e. :

$$\mathcal{H}(\langle \mathcal{M}, u \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{M}(u) \text{ s'arrête} \\ q_{reject} & \text{si } \mathcal{M}(u) \text{ ne s'arrête pas.} \end{cases}$$

- **Remarque :** le code $\langle \mathcal{M} \rangle$ est une suite de bits, il peut donc faire office d'input !
- On modifie légèrement le programme de $\mathcal{H}$ pour obtenir la MT $\mathcal{H}'$ suivante :

$$\mathcal{H}'(\langle \mathcal{M} \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{accept} \\ q_{reject} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{reject}. \end{cases}$$

Problème de l'arrêt

- On a donc :

$$\mathcal{H}'(\langle \mathcal{M} \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{accept} \\ q_{reject} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{reject} \end{cases}$$

- On modifie facilement le programme de $\mathcal{H}'$ pour obtenir la MT $\mathcal{H}''$ suivante :

$$\mathcal{H}''(\langle \mathcal{M} \rangle) = \begin{cases} \infty - loop & \text{si } \mathcal{H}'(\langle \mathcal{M} \rangle) = q_{accept} \\ q_{accept} & \text{si } \mathcal{H}'(\langle \mathcal{M} \rangle) = q_{reject} \end{cases}$$

Problème de l'arrêt

- On a donc :

$$\mathcal{H}'(\langle \mathcal{M} \rangle) = \begin{cases} q_{accept} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{accept} \\ q_{reject} & \text{si } \mathcal{H}(\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle) = q_{reject} \end{cases}$$

- On modifie facilement le programme de $\mathcal{H}'$ pour obtenir la MT $\mathcal{H}''$ suivante :

$$\mathcal{H}''(\langle \mathcal{M} \rangle) = \begin{cases} \infty - loop & \text{si } \mathcal{H}'(\langle \mathcal{M} \rangle) = q_{accept} \\ q_{accept} & \text{si } \mathcal{H}'(\langle \mathcal{M} \rangle) = q_{reject} \end{cases}$$

Problème de l'arrêt

- Si l'on considère la computation de $\mathcal{H}''$ sur son propre code, on a :

$$\mathcal{H}''(\langle\mathcal{H}''\rangle) = \infty\text{-}loop \quad \text{si} \quad \mathcal{H}'(\langle\mathcal{H}''\rangle) = q_{accept}$$
$$\text{si} \quad \mathcal{H}(\langle\mathcal{H}'', \langle\mathcal{H}''\rangle\rangle) = q_{accept}$$
$$\text{si} \quad \mathcal{H}''(\langle\mathcal{H}''\rangle) \text{ s'arrête}$$

$$\mathcal{H}''(\langle\mathcal{H}''\rangle) = q_{accept} \quad \text{si} \quad \mathcal{H}'(\langle\mathcal{H}''\rangle) = q_{reject}$$
$$\text{si} \quad \mathcal{H}(\langle\mathcal{H}'', \langle\mathcal{H}''\rangle\rangle) = q_{reject}$$
$$\text{si} \quad \mathcal{H}''(\langle\mathcal{H}''\rangle) \text{ ne s'arrête pas}$$

- Donc $\mathcal{H}''(\langle\mathcal{H}''\rangle)$ s'arrête si et seulement si $\mathcal{H}''(\langle\mathcal{H}''\rangle)$ ne s'arrête pas, contradiction. □

Problème de l'arrêt

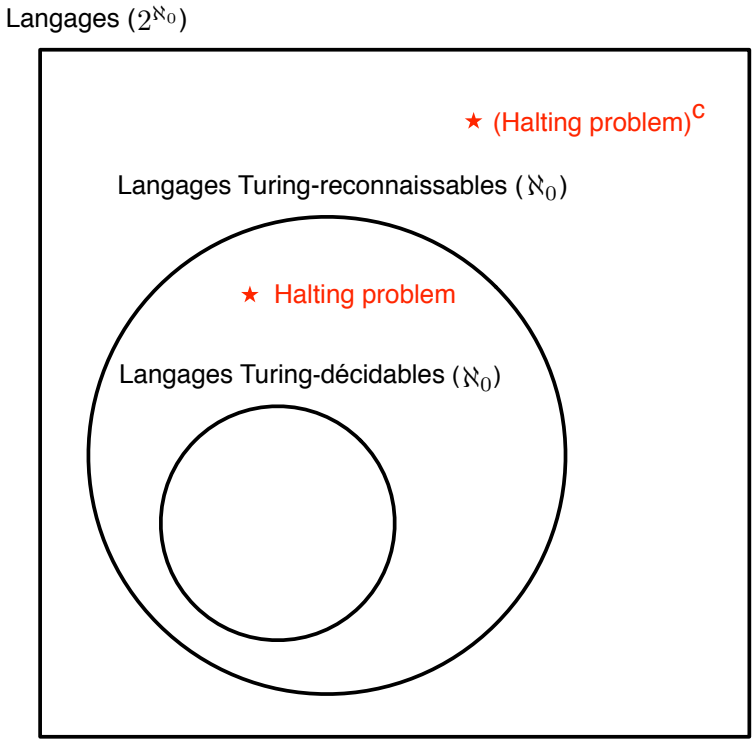
- Si l'on considère la computation de $\mathcal{H}''$ sur son propre code, on a :

$$\begin{aligned} \mathcal{H}''(\langle \mathcal{H}'' \rangle) = \infty - loop \quad & \text{si} \quad \mathcal{H}'(\langle \mathcal{H}'' \rangle) = q_{accept} \\ & \text{si} \quad \mathcal{H}(\langle \mathcal{H}'', \langle \mathcal{H}'' \rangle \rangle) = q_{accept} \\ & \text{si} \quad \mathcal{H}''(\langle \mathcal{H}'' \rangle) \text{ s'arrête} \\ \mathcal{H}''(\langle \mathcal{H}'' \rangle) = q_{accept} \quad & \text{si} \quad \mathcal{H}'(\langle \mathcal{H}'' \rangle) = q_{reject} \\ & \text{si} \quad \mathcal{H}(\langle \mathcal{H}'', \langle \mathcal{H}'' \rangle \rangle) = q_{reject} \\ & \text{si} \quad \mathcal{H}''(\langle \mathcal{H}'' \rangle) \text{ ne s'arrête pas} \end{aligned}$$

- Donc $\mathcal{H}''(\langle \mathcal{H}'' \rangle)$ s'arrête si et seulement si $\mathcal{H}''(\langle \mathcal{H}'' \rangle)$ ne s'arrête pas, contradiction. □

Problème de l'arrêt

- Nous allons finalement exhibé un problème – le complémentaire du problème de l'arrêt – qui n'est pas Turing-reconnaissable (donc pas Turing-décidable non plus).



Problème de l'arrêt

Lemme 18

Un langage L est Turing-décidable si et seulement si L et L^c sont tous deux Turing-reconnaissables.

Preuve :

($\Rightarrow$) Soit M une MT qui décide L ; alors M reconnaît L. De plus, la MT M' obtenue en permutant les états acceptant et rejetant de M reconnaît (et même décide) L^c.

$$M'(u) = \begin{cases} \text{accepte si } M(u) \text{ rejette} \\ \text{rejette si } M(u) \text{ accepte} \end{cases}$$

Problème de l'arrêt

Lemme 18

Un langage L est Turing-décidable si et seulement si L et L^c sont tous deux Turing-reconnaissables.

Preuve :

($\Rightarrow$) Soit $\mathcal{M}$ une MT qui décide L ; alors $\mathcal{M}$ reconnaît L. De plus, la MT $\mathcal{M}^c$ obtenue en permutant les états acceptant et rejetant de $\mathcal{M}$ reconnaît (et même décide) L^c.

$$\mathcal{M}(u) = \begin{cases} q_{acc} & \text{si } u \in L \\ q_{rej} & \text{si } u \notin L \end{cases} \qquad \mathcal{M}^c(u) = \begin{cases} q_{rej} & \text{si } u \notin L^c \\ q_{acc} & \text{si } u \in L^c \end{cases}$$

Problème de l'arrêt

Lemme 18

Un langage L est Turing-décidable si et seulement si L et L^c sont tous deux Turing-reconnaissables.

Preuve :

($\Rightarrow$) Soit $\mathcal{M}$ une MT qui décide L ; alors $\mathcal{M}$ reconnaît L. De plus, la MT $\mathcal{M}^c$ obtenue en permutant les états acceptant et rejetant de $\mathcal{M}$ reconnaît (et même décide) L^c.

$$\mathcal{M}(u) = \begin{cases} q_{acc} & \text{si } u \in L \\ q_{rej} & \text{si } u \notin L \end{cases} \qquad \mathcal{M}^c(u) = \begin{cases} q_{rej} & \text{si } u \notin L^c \\ q_{acc} & \text{si } u \in L^c \end{cases}$$

Problème de l'arrêt

Lemme 18

Un langage L est Turing-décidable si et seulement si L et L^c sont tous deux Turing-reconnaissables.

Preuve :

($\Rightarrow$) Soit $\mathcal{M}$ une MT qui décide L ; alors $\mathcal{M}$ reconnaît L. De plus, la MT $\mathcal{M}^c$ obtenue en permutant les états acceptant et rejetant de $\mathcal{M}$ reconnaît (et même décide) L^c.

$$\mathcal{M}(u) = \begin{cases} q_{acc} & \text{si } u \in L \\ q_{rej} & \text{si } u \notin L \end{cases} \qquad \mathcal{M}^c(u) = \begin{cases} q_{rej} & \text{si } u \notin L^c \\ q_{acc} & \text{si } u \in L^c \end{cases}$$

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L. □

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L.

□

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L. □

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L. □

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L.

□

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L.

□

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L.

Problème de l'arrêt

($\Leftarrow$) Supposons L et L^c reconnaissables par les MT $\mathcal{M}_1$ et $\mathcal{M}_2$, respectivement. Alors L est décidé par la MT $\mathcal{M}$ ci-dessous :

Étant donné l'input u :

1. simuler les MT $\mathcal{M}_1$ et $\mathcal{M}_2$ en parallèle sur l'input u ;
2. si $\mathcal{M}_1(u) = q_{accept}$, *accepter* l'input ;
3. si $\mathcal{M}_2(u) = q_{accept}$, *rejeter* l'input.

- Si $u \in L$, alors $\mathcal{M}_1(u) = q_{accept}$ (puisque $\mathcal{M}_1$ reconnaît L) ; donc $\mathcal{M}(u) = q_{accept}$.
- Si $u \notin L$, i.e. $u \in L^c$, alors $\mathcal{M}_2(u) = q_{accept}$ (puisque $\mathcal{M}_2$ reconnaît L^c) ; donc $\mathcal{M}(u) = q_{reject}$.
- Ainsi, $\mathcal{M}$ s'arrête sur tous les inputs possibles et décide L.

□

Problème de l'arrêt

Corollaire 19

Le complémentaire du problème de l'arrêt, $L^c_{\mathcal{H}}$, n'est pas Turing-reconnaissable.

Preuve :

Si $L^c_{\mathcal{H}}$ est Turing-reconnaissable, on a $L_{\mathcal{H}}$ et $L^c_{\mathcal{H}}$ tous deux Turing-reconnaissables, donc, par le Lemme 18, $L_{\mathcal{H}}$ Turing-décidable, contradiction (Théorème 17). □

Problème de l'arrêt

Corollaire 19

Le complémentaire du problème de l'arrêt, $L_{\mathcal{H}}^c$, n'est pas Turing-reconnaissable.

Preuve :

Si $L_{\mathcal{H}}^c$ est Turing-reconnaissable, on a $L_{\mathcal{H}}$ et $L_{\mathcal{H}}^c$ tous deux Turing-reconnaissables, donc, par le Lemme 18, $L_{\mathcal{H}}$ Turing-décidable, contradiction (Théorème 17). □

Problème de l'arrêt

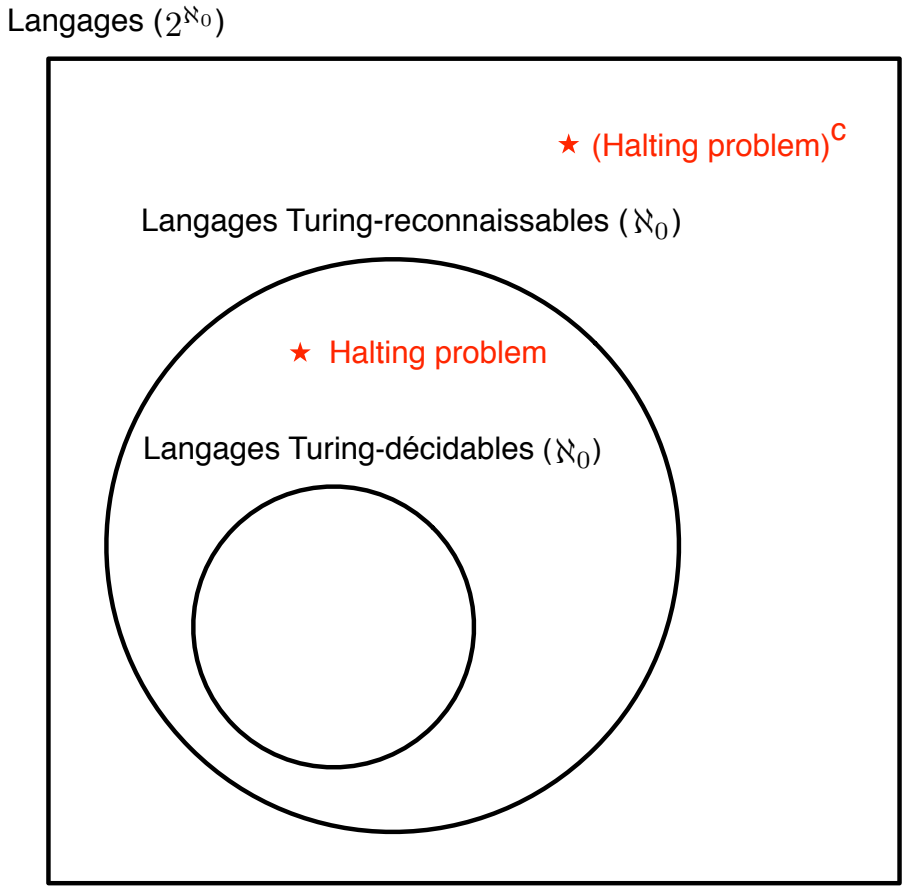


Figure : Résumé de la situation

Entscheidungsproblem

- En 1928, Hilbert et Ackermann proposent un fameux problème de décision, connu sous le nom de *Entscheidungsproblem*.

  > *"Das Entscheidungsproblem is gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt."*

- *Entscheidungsproblem :* étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, existe-t-il une « *procédure effective* » (*Verfahren*) qui détermine en un temps fini si φ est prouvable à partir de Γ ou non ?

Entscheidungsproblem

- En 1928, Hilbert et Ackermann proposent un fameux problème de décision, connu sous le nom de *Entscheidungsproblem*.

  > *"Das Entscheidungsproblem is gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt."*

- *Entscheidungsproblem :* étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, existe-t-il une « *procédure effective* » (*Verfahren*) qui détermine en un temps fini si φ est prouvable à partir de Γ ou non ?

Entscheidungsproblem

- En 1928, Hilbert et Ackermann proposent un fameux problème de décision, connu sous le nom de *Entscheidungsproblem*.

  > *"Das Entscheidungsproblem is gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt."*

- *Entscheidungsproblem :* étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, existe-t-il une « *procédure effective* » (*Verfahren*) qui détermine en un temps fini si φ est prouvable à partir de Γ ou non ?

Entscheidungsproblem

Idée générale de Hilbert

Proposer une « mécanisation effective » du raisonnement mathématique, du concept de preuve.

- a donné lieu à toute la « théorie de la démonstration », avec une multitude d'applications de nos jours...

Entscheidungsproblem

- Problème : à ce moment-là, il n'y a pas de formulation claire de ce qu'est une « *procédure effective* » (*Verfahren*).
- En 1936, Alonso Church et Alan Turing introduisent deux notions de "procédure effective", le *lambda calcul* et la *machine de Turing*, respectivement, et prouvent indépendamment que le Entscheidungsproblem de la logique du premier ordre est indécidable (relativement à ces notions de calculabilité).
- Ces notions de calculabilité (lambda calcul et machine de Turing) sont en fait équivalentes.
- Church donne la première preuve. Celle de Turing est plus intuitive...

Entscheidungsproblem

- Problème : à ce moment-là, il n'y a pas de formulation claire de ce qu'est une « *procédure effective* » (*Verfahren*).
- En 1936, Alonso Church et Alan Turing introduisent deux notions de "procédure effective", le *lambda calcul* et la *machine de Turing*, respectivement, et prouvent indépendamment que le Entscheidungsproblem de la logique du premier ordre est indécidable (relativement à ces notions de calculabilité).
- Ces notions de calculabilité (lambda calcul et machine de Turing) sont en fait équivalentes.
- Church donne la première preuve. Celle de Turing est plus intuitive...

Entscheidungsproblem

- Problème : à ce moment-là, il n'y a pas de formulation claire de ce qu'est une « *procédure effective* » (*Verfahren*).
- En 1936, Alonso Church et Alan Turing introduisent deux notions de "procédure effective", le *lambda calcul* et la *machine de Turing*, respectivement, et prouvent indépendamment que le Entscheidungsproblem de la logique du premier ordre est indécidable (relativement à ces notions de calculabilité).
- Ces notions de calculabilité (lambda calcul et machine de Turing) sont en fait équivalentes.
- Church donne la première preuve. Celle de Turing est plus intuitive...

Entscheidungsproblem

- Problème : à ce moment-là, il n'y a pas de formulation claire de ce qu'est une « *procédure effective* » (*Verfahren*).
- En 1936, Alonso Church et Alan Turing introduisent deux notions de "procédure effective", le *lambda calcul* et la *machine de Turing*, respectivement, et prouvent indépendamment que le Entscheidungsproblem de la logique du premier ordre est indécidable (relativement à ces notions de calculabilité).
- Ces notions de calculabilité (lambda calcul et machine de Turing) sont en fait équivalentes.
- Church donne la première preuve. Celle de Turing est plus intuitive...

Entscheidungsproblem

Théorème 20 (Church-Turing 1936)

- *Le Entscheidungsproblem pour la logique du premier ordre est Turing-indécidable.*
- *Étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, le problème de savoir si $\Gamma \vdash \varphi$ ou non est Turing-indécidable.*
- *Le langage $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ est Turing-indécidable.*

Entscheidungsproblem

Théorème 20 (Church-Turing 1936)

- *Le Entscheidungsproblem pour la logique du premier ordre est Turing-indécidable.*
- *Étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, le problème de savoir si $\Gamma \vdash \varphi$ ou non est Turing-indécidable.*
- *Le langage $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ est Turing-indécidable.*

Entscheidungsproblem

Théorème 20 (Church-Turing 1936)

- *Le Entscheidungsproblem pour la logique du premier ordre est Turing-indécidable.*
- *Étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, le problème de savoir si $\Gamma \vdash \varphi$ ou non est Turing-indécidable.*
- *Le langage $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ est Turing-indécidable.*

Entscheidungsproblem

- **Remarque :** Le Entscheidungsproblem pour la logique du premier ordre est *en général* Turing-indécidable
- Mais pour certaines théories Γ (complètes) « faibles », le problème de décision "$\Gamma \vdash \varphi$?" est Turing-décidable.
- le calcul des prédicats monadiques du premier ordre
- la théorie des ordres denses
- l'arithmétique de Presburger

Entscheidungsproblem

- **Remarque :** Le Entscheidungsproblem pour la logique du premier ordre est *en général* Turing-indécidable
- Mais pour certaines théories Γ (complètes) « faibles », le problème de décision "$\Gamma \vdash \varphi$?" est Turing-décidable.
- le calcul des prédicats monadiques du premier ordre
- la théorie des ordres denses
- l'arithmétique de Presburger

Entscheidungsproblem

- **Remarque :** Le Entscheidungsproblem pour la logique du premier ordre est *en général* Turing-indécidable
- Mais pour certaines théories Γ (complètes) « faibles », le problème de décision "$\Gamma \vdash \varphi$?" est Turing-décidable.
- le calcul des prédicats monadiques du premier ordre
- la théorie des ordres denses
- l'arithmétique de Presburger

Entscheidungsproblem

- **Remarque :** Le Entscheidungsproblem pour la logique du premier ordre est *en général* Turing-indécidable
- Mais pour certaines théories Γ (complètes) « faibles », le problème de décision "$\Gamma \vdash \varphi$?" est Turing-décidable.
- le calcul des prédicats monadiques du premier ordre
- la théorie des ordres denses
- l'arithmétique de Presburger

Entscheidungsproblem

- **Remarque :** Le Entscheidungsproblem pour la logique du premier ordre est *en général* Turing-indécidable
- Mais pour certaines théories Γ (complètes) « faibles », le problème de décision "$\Gamma \vdash \varphi$?" est Turing-décidable.
- le calcul des prédicats monadiques du premier ordre
- la théorie des ordres denses
- l'arithmétique de Presburger

Entscheidungsproblem

Preuve (idée) :

- *On réduit le Entscheidungsproblem au problème de l'arrêt qui est Turing-indécidable.*
- Pour toute machine de Turing $\mathcal{M}$ et input u, on associe une théorie et un énoncé de la logique du premier ordre $\Gamma_{(\mathcal{M},u)}$ et $\varphi_{(\mathcal{M},u)}$ tels que

$$\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)} \text{ ssi la computation } \mathcal{M}(u) \text{ s'arrête.}$$

Entscheidungsproblem

Preuve (idée) :

- *On réduit le Entscheidungsproblem au problème de l'arrêt qui est Turing-indécidable.*
- Pour toute machine de Turing $\mathcal{M}$ et input u, on associe une théorie et un énoncé de la logique du premier ordre $\Gamma_{(\mathcal{M},u)}$ et $\varphi_{(\mathcal{M},u)}$ tels que

$$\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)} \text{ ssi la computation } \mathcal{M}(u) \text{ s'arrête.}$$

Entscheidungsproblem

- Ainsi si le Entscheidungsproblem $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ était Turing-décidable, alors le problème de l'arrêt

$$\{\langle \mathcal{M}, u \rangle : \mathcal{M}(u) \text{ s'arrête}\}$$

le serait également par la machine de Turing suivante :

Étant donné l'input $\langle \mathcal{M}, u \rangle$:

1. transformer $\langle \mathcal{M}, u \rangle$ en $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$;
2. lancer la procédure de décision du Entscheidungsproblem sur l'input $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$.
3. Si la procédure accepte $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *accepter* $\langle \mathcal{M}, u \rangle$;
4. Si la procédure rejète $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *rejeter* $\langle \mathcal{M}, u \rangle$.

- Ceci est une contradiction (Théorème 17) ; donc le Entscheidungsproblem est Turing-indécidable.

Entscheidungsproblem

- Ainsi si le Entscheidungsproblem $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ était Turing-décidable, alors le problème de l'arrêt

$$\{\langle \mathcal{M}, u \rangle : \mathcal{M}(u) \text{ s'arrête}\}$$

le serait également par la machine de Turing suivante :

Étant donné l'input $\langle \mathcal{M}, u \rangle$:

1. transformer $\langle \mathcal{M}, u \rangle$ en $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$;
2. lancer la procédure de décision du Entscheidungsproblem sur l'input $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$.
3. Si la procédure accepte $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *accepter* $\langle \mathcal{M}, u \rangle$;
4. Si la procédure rejète $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *rejeter* $\langle \mathcal{M}, u \rangle$.

- Ceci est une contradiction (Théorème 17) ; donc le Entscheidungsproblem est Turing-indécidable.

Entscheidungsproblem

- Ainsi si le Entscheidungsproblem $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ était Turing-décidable, alors le problème de l'arrêt

$$\{\langle \mathcal{M}, u \rangle : \mathcal{M}(u) \text{ s'arrête}\}$$

le serait également par la machine de Turing suivante :

Étant donné l'input $\langle \mathcal{M}, u \rangle$:

1. transformer $\langle \mathcal{M}, u \rangle$ en $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$;
2. lancer la procédure de décision du Entscheidungsproblem sur l'input $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$.
3. Si la procédure accepte $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *accepter* $\langle \mathcal{M}, u \rangle$;
4. Si la procédure rejète $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *rejeter* $\langle \mathcal{M}, u \rangle$.

- Ceci est une contradiction (Théorème 17) ; donc le Entscheidungsproblem est Turing-indécidable.

Entscheidungsproblem

- Ainsi si le Entscheidungsproblem $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ était Turing-décidable, alors le problème de l'arrêt

$$\{\langle \mathcal{M}, u \rangle : \mathcal{M}(u) \text{ s'arrête}\}$$

le serait également par la machine de Turing suivante :

Étant donné l'input $\langle \mathcal{M}, u \rangle$:

1. transformer $\langle \mathcal{M}, u \rangle$ en $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$;
2. lancer la procédure de décision du Entscheidungsproblem sur l'input $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$.
3. Si la procédure accepte $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *accepter* $\langle \mathcal{M}, u \rangle$;
4. Si la procédure rejète $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *rejeter* $\langle \mathcal{M}, u \rangle$.

- Ceci est une contradiction (Théorème 17) ; donc le Entscheidungsproblem est Turing-indécidable.

Entscheidungsproblem

- Ainsi si le Entscheidungsproblem $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ était Turing-décidable, alors le problème de l'arrêt

$$\{\langle \mathcal{M}, u \rangle : \mathcal{M}(u) \text{ s'arrête}\}$$

le serait également par la machine de Turing suivante :

Étant donné l'input $\langle \mathcal{M}, u \rangle$:

1. transformer $\langle \mathcal{M}, u \rangle$ en $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$;
2. lancer la procédure de décision du Entscheidungsproblem sur l'input $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$.
3. Si la procédure accepte $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *accepter* $\langle \mathcal{M}, u \rangle$;
4. Si la procédure rejète $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *rejeter* $\langle \mathcal{M}, u \rangle$.

- Ceci est une contradiction (Théorème 17) ; donc le Entscheidungsproblem est Turing-indécidable.

Entscheidungsproblem

- Ainsi si le Entscheidungsproblem $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ était Turing-décidable, alors le problème de l'arrêt

$$\{\langle \mathcal{M}, u \rangle : \mathcal{M}(u) \text{ s'arrête}\}$$

le serait également par la machine de Turing suivante :

Étant donné l'input $\langle \mathcal{M}, u \rangle$:

1. transformer $\langle \mathcal{M}, u \rangle$ en $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$;
2. lancer la procédure de décision du Entscheidungsproblem sur l'input $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$.
3. Si la procédure accepte $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *accepter* $\langle \mathcal{M}, u \rangle$;
4. Si la procédure rejète $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *rejeter* $\langle \mathcal{M}, u \rangle$.

- Ceci est une contradiction (Théorème 17) ; donc le Entscheidungsproblem est Turing-indécidable.

Entscheidungsproblem

- Ainsi si le Entscheidungsproblem $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ était Turing-décidable, alors le problème de l'arrêt

$$\{\langle \mathcal{M}, u \rangle : \mathcal{M}(u) \text{ s'arrête}\}$$

le serait également par la machine de Turing suivante :

Étant donné l'input $\langle \mathcal{M}, u \rangle$:

1. transformer $\langle \mathcal{M}, u \rangle$ en $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$;
2. lancer la procédure de décision du Entscheidungsproblem sur l'input $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$.
3. Si la procédure accepte $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *accepter* $\langle \mathcal{M}, u \rangle$;
4. Si la procédure rejète $\langle \Gamma_{(\mathcal{M},u)}, \varphi_{(\mathcal{M},u)} \rangle$, *rejeter* $\langle \mathcal{M}, u \rangle$.

- Ceci est une contradiction (Théorème 17) ; donc le Entscheidungsproblem est Turing-indécidable.

Entscheidungsproblem

Preuve :

- Pour toute machine de Turing $\mathcal{M}$ et input u, on associe une théorie et un énoncé de la logique du premier ordre $\Gamma_{(\mathcal{M},u)}$ et $\varphi_{(\mathcal{M},u)}$ tels que

$$\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)} \text{ ssi la computation } \mathcal{M}(u) \text{ s'arrête.}$$

- Sans perte de généralité, one se limite au machines de Turing sur un alphabet unaire ("1") avec une bande infinie à gauche et à droite ; la preuve de l'indécidabilité du problème de l'arrêt pour ces machines est identique.
- Soient $\mathcal{M}$ et u une machine de Turing et un input.

Entscheidungsproblem

Preuve :

- Pour toute machine de Turing $\mathcal{M}$ et input u, on associe une théorie et un énoncé de la logique du premier ordre $\Gamma_{(\mathcal{M},u)}$ et $\varphi_{(\mathcal{M},u)}$ tels que

$$\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)} \text{ ssi la computation } \mathcal{M}(u) \text{ s'arrête.}$$

- Sans perte de généralité, one se limite au machines de Turing sur un alphabet unaire (“1”) avec une bande infinie à gauche et à droite ; la preuve de l'indécidabilité du problème de l'arrêt pour ces machines est identique.
- Soient $\mathcal{M}$ et u une machine de Turing et un input.

Entscheidungsproblem

Preuve :

- Pour toute machine de Turing $\mathcal{M}$ et input u, on associe une théorie et un énoncé de la logique du premier ordre $\Gamma_{(\mathcal{M},u)}$ et $\varphi_{(\mathcal{M},u)}$ tels que

$$\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)} \text{ ssi la computation } \mathcal{M}(u) \text{ s'arrête.}$$

- Sans perte de généralité, one se limite au machines de Turing sur un alphabet unaire (“1”) avec une bande infinie à gauche et à droite ; la preuve de l'indécidabilité du problème de l'arrêt pour ces machines est identique.
- Soient $\mathcal{M}$ et u une machine de Turing et un input.

Entscheidungsproblem

- **Syntaxe** (basée sur $\mathcal{M}$ et u) : le langage contient un symbole de constante ($\mathbf{0}$), un de symbole de prédicat unaire Q_i pour chaque état non final q_i de $\mathcal{M}$, et quatre symboles de prédicats binaires

$$\mathcal{L} = \left\{ \mathbf{0}, (Q_i)_{i=1}^{k}, S, <, P, M \right\}$$

Entscheidungsproblem

- **Sémantique** (basée sur $\mathcal{M}$ et u) : on considère la $\mathcal{L}$-structure

$$\mathcal{M}^* = \left\langle \mathbb{Z}, \mathbf{0}^{\mathcal{M}^*}, (Q_i^{\mathcal{M}^*})_{i=1}^k, S^{\mathcal{M}^*}, <^{\mathcal{M}^*}, P^{\mathcal{M}^*}, M^{\mathcal{M}^*} \right\rangle$$

  - $\mathbf{0}^{\mathcal{M}^*} = 0$
  - $Q_i^{\mathcal{M}^*} = \{t \in \mathbb{N} : \mathcal{M}(u) \text{ est dans l'état } q_i \text{ au temps } t\}$
  - $S^{\mathcal{M}^*}$: relation de successeur usuelle sur $\mathbb{Z}$
  - $<^{\mathcal{M}^*}$: relation d'ordre stricte usuelle sur $\mathbb{Z}$
  - $P^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ scanne la cellule } x\}$
  - $M^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ a sa cellule } x \text{ marquée}\}$

Entscheidungsproblem

- **Sémantique** (basée sur $\mathcal{M}$ et u) : on considère la $\mathcal{L}$-structure

$$\mathcal{M}^* = \left\langle \mathbb{Z}, \mathbf{0}^{\mathcal{M}^*}, (Q_i^{\mathcal{M}^*})_{i=1}^k, S^{\mathcal{M}^*}, <^{\mathcal{M}^*}, P^{\mathcal{M}^*}, M^{\mathcal{M}^*} \right\rangle$$

  - $\mathbf{0}^{\mathcal{M}^*} = 0$
  - $Q_i^{\mathcal{M}^*} = \{t \in \mathbb{N} : \mathcal{M}(u) \text{ est dans l'état } q_i \text{ au temps } t\}$
  - $S^{\mathcal{M}^*}$: relation de successeur usuelle sur $\mathbb{Z}$
  - $<^{\mathcal{M}^*}$: relation d'ordre stricte usuelle sur $\mathbb{Z}$
  - $P^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ scanne la cellule } x\}$
  - $M^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ a sa cellule } x \text{ marquée}\}$

Entscheidungsproblem

- **Sémantique** (basée sur $\mathcal{M}$ et u) : on considère la $\mathcal{L}$-structure

$$\mathcal{M}^* = \left\langle \mathbb{Z}, \mathbf{0}^{\mathcal{M}^*}, (Q_i^{\mathcal{M}^*})_{i=1}^k, S^{\mathcal{M}^*}, <^{\mathcal{M}^*}, P^{\mathcal{M}^*}, M^{\mathcal{M}^*} \right\rangle$$

  - $\mathbf{0}^{\mathcal{M}^*} = 0$
  - $Q_i^{\mathcal{M}^*} = \{t \in \mathbb{N} : \mathcal{M}(u) \text{ est dans l'état } q_i \text{ au temps } t\}$
  - $S^{\mathcal{M}^*}$: relation de successeur usuelle sur $\mathbb{Z}$
  - $<^{\mathcal{M}^*}$: relation d'ordre stricte usuelle sur $\mathbb{Z}$
  - $P^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N} \times \mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ scanne la cellule } x\}$
  - $M^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N} \times \mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ a sa cellule } x \text{ marquée}\}$

Entscheidungsproblem

- **Sémantique** (basée sur $\mathcal{M}$ et u) : on considère la $\mathcal{L}$-structure

$$\mathcal{M}^* = \left\langle \mathbb{Z}, \mathbf{0}^{\mathcal{M}^*}, (Q_i^{\mathcal{M}^*})_{i=1}^k, S^{\mathcal{M}^*}, <^{\mathcal{M}^*}, P^{\mathcal{M}^*}, M^{\mathcal{M}^*} \right\rangle$$

  - $\mathbf{0}^{\mathcal{M}^*} = 0$
  - $Q_i^{\mathcal{M}^*} = \{t \in \mathbb{N} : \mathcal{M}(u) \text{ est dans l'état } q_i \text{ au temps } t\}$
  - $S^{\mathcal{M}^*}$: relation de successeur usuelle sur $\mathbb{Z}$
  - $<^{\mathcal{M}^*}$: relation d'ordre stricte usuelle sur $\mathbb{Z}$
  - $P^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N} \times \mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ scanne la cellule } x\}$
  - $M^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N} \times \mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ a sa cellule } x \text{ marquée}\}$

Entscheidungsproblem

- **Sémantique** (basée sur $\mathcal{M}$ et u) : on considère la $\mathcal{L}$-structure

$$\mathcal{M}^* = \left\langle \mathbb{Z}, \mathbf{0}^{\mathcal{M}^*}, (Q_i^{\mathcal{M}^*})_{i=1}^k, S^{\mathcal{M}^*}, <^{\mathcal{M}^*}, P^{\mathcal{M}^*}, M^{\mathcal{M}^*} \right\rangle$$

  - $\mathbf{0}^{\mathcal{M}^*} = 0$
  - $Q_i^{\mathcal{M}^*} = \{t \in \mathbb{N} : \mathcal{M}(u) \text{ est dans l'état } q_i \text{ au temps } t\}$
  - $S^{\mathcal{M}^*}$: relation de successeur usuelle sur $\mathbb{Z}$
  - $<^{\mathcal{M}^*}$: relation d'ordre stricte usuelle sur $\mathbb{Z}$
  - $P^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ scanne la cellule } x\}$
  - $M^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ a sa cellule } x \text{ marquée}\}$

Entscheidungsproblem

- **Sémantique** (basée sur $\mathcal{M}$ et u) : on considère la $\mathcal{L}$-structure

$$\mathcal{M}^* = \left\langle \mathbb{Z}, \mathbf{0}^{\mathcal{M}^*}, (Q_i^{\mathcal{M}^*})_{i=1}^k, S^{\mathcal{M}^*}, <^{\mathcal{M}^*}, P^{\mathcal{M}^*}, M^{\mathcal{M}^*} \right\rangle$$

  - $\mathbf{0}^{\mathcal{M}^*} = 0$
  - $Q_i^{\mathcal{M}^*} = \{t \in \mathbb{N} : \mathcal{M}(u)$ est dans l'état q_i au temps $t\}$
  - $S^{\mathcal{M}^*}$: relation de successeur usuelle sur $\mathbb{Z}$
  - $<^{\mathcal{M}^*}$: relation d'ordre stricte usuelle sur $\mathbb{Z}$
  - $P^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} :$ au temps t, $\mathcal{M}(u)$ scanne la cellule $x\}$
  - $M^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} :$ au temps t, $\mathcal{M}(u)$ a sa cellule x marquée$\}$

Entscheidungsproblem

- **Sémantique** (basée sur $\mathcal{M}$ et u) : on considère la $\mathcal{L}$-structure

$$\mathcal{M}^* = \left\langle \mathbb{Z}, \mathbf{0}^{\mathcal{M}^*}, (Q_i^{\mathcal{M}^*})_{i=1}^k, S^{\mathcal{M}^*}, <^{\mathcal{M}^*}, P^{\mathcal{M}^*}, M^{\mathcal{M}^*} \right\rangle$$

  - $\mathbf{0}^{\mathcal{M}^*} = 0$
  - $Q_i^{\mathcal{M}^*} = \{t \in \mathbb{N} : \mathcal{M}(u) \text{ est dans l'état } q_i \text{ au temps } t\}$
  - $S^{\mathcal{M}^*}$: relation de successeur usuelle sur $\mathbb{Z}$
  - $<^{\mathcal{M}^*}$: relation d'ordre stricte usuelle sur $\mathbb{Z}$
  - $P^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ scanne la cellule } x\}$
  - $M^{\mathcal{M}^*} = \{(t, x) \in \mathbb{N}\times\mathbb{Z} : \text{au temps } t, \mathcal{M}(u) \text{ a sa cellule } x \text{ marquée}\}$

Entscheidungsproblem

On donne les formules qui forment $\Gamma_{(\mathcal{M},u)}$.

1. Premières formules de $\Gamma_{(\mathcal{M},u)}$: le "background information".

$$\forall u\ \forall v\ \forall w\ (((S(u,v) \wedge S(u,w)) \rightarrow v = w) \wedge ((S(v,u) \wedge S(w,u)) \rightarrow v = w)) \tag{1}$$

$$\forall u\ \forall v\ (S(u,v) \rightarrow u < v) \wedge \forall u\ \forall v\ \forall w\ ((u < v \wedge v < w) \rightarrow u < w) \tag{2}$$

$$\forall u\ \forall v\ (u < v \rightarrow u \neq v) \tag{3}$$

   - Ces formules sont satisfaites dans $\mathcal{M}^*$ (les relations de successeur et d'ordre strictes sur $\mathbb{Z}$ satisfont ces formules).

Entscheidungsproblem

On donne les formules qui forment $\Gamma_{(\mathcal{M},u)}$.

1. Premières formules de $\Gamma_{(\mathcal{M},u)}$: le "background information".

$$\forall u\ \forall v\ \forall w\ (((S(u,v) \wedge S(u,w)) \rightarrow v = w) \wedge ((S(v,u) \wedge S(w,u)) \rightarrow v = w)) \tag{1}$$

$$\forall u\ \forall v\ (S(u,v) \rightarrow u < v) \wedge \forall u\ \forall v\ \forall w\ ((u < v \wedge v < w) \rightarrow u < w) \tag{2}$$

$$\forall u\ \forall v\ (u < v \rightarrow u \neq v) \tag{3}$$

- Ces formules sont satisfaites dans $\mathcal{M}^*$ (les relations de successeur et d'ordre strictes sur $\mathbb{Z}$ satisfont ces formules).

Entscheidungsproblem

- On introduit quelques abréviations...
  - $S_0(u, v)$ pour $u = v$
  - $S_1(u, v)$ pour $S(u, v)$
  - $S_2(u, v)$ pour $\exists y \ (S(u, y) \wedge S(y, v))$
  - $S_3(u, v)$ pour $\exists y_1 \ \exists y_2 \ (S(u, y_1) \wedge S(y_1, y_2) \wedge S(y_2, v))$
  - etc.
  - $S_{-i}(u, v)$ pour $S_i(v, u)$

Entscheidungsproblem

- On introduit quelques abréviations...
  - $S_0(u, v)$ pour $u = v$
  - $S_1(u, v)$ pour $S(u, v)$
  - $S_2(u, v)$ pour $\exists y \ (S(u, y) \wedge S(y, v))$
  - $S_3(u, v)$ pour $\exists y_1 \ \exists y_2 \ (S(u, y_1) \wedge S(y_1, y_2) \wedge S(y_2, v))$
  - etc.
  - $S_{-i}(u, v)$ pour $S_i(v, u)$

Entscheidungsproblem

- On introduit quelques abréviations...
  - $S_0(u, v)$ pour $u = v$
  - $S_1(u, v)$ pour $S(u, v)$
  - $S_2(u, v)$ pour $\exists y \ (S(u, y) \wedge S(y, v))$
  - $S_3(u, v)$ pour $\exists y_1 \ \exists y_2 \ (S(u, y_1) \wedge S(y_1, y_2) \wedge S(y_2, v))$
  - etc.
  - $S_{-i}(u, v)$ pour $S_i(v, u)$

Entscheidungsproblem

- On introduit quelques abréviations...
  - $S_0(u, v)$ pour $u = v$
  - $S_1(u, v)$ pour $S(u, v)$
  - $S_2(u, v)$ pour $\exists y \ (S(u, y) \wedge S(y, v))$
  - $S_3(u, v)$ pour $\exists y_1 \ \exists y_2 \ (S(u, y_1) \wedge S(y_1, y_2) \wedge S(y_2, v))$
  - etc.
  - $S_{-i}(u, v)$ pour $S_i(v, u)$

Entscheidungsproblem

- On introduit quelques abréviations...
  - $S_0(u, v)$ pour $u = v$
  - $S_1(u, v)$ pour $S(u, v)$
  - $S_2(u, v)$ pour $\exists y \ (S(u, y) \wedge S(y, v))$
  - $S_3(u, v)$ pour $\exists y_1 \ \exists y_2 \ (S(u, y_1) \wedge S(y_1, y_2) \wedge S(y_2, v))$
  - etc.
  - $S_{-i}(u, v)$ pour $S_i(v, u)$

Entscheidungsproblem

- On introduit quelques abréviations...
  - $S_0(u, v)$ pour $u = v$
  - $S_1(u, v)$ pour $S(u, v)$
  - $S_2(u, v)$ pour $\exists y \ (S(u, y) \wedge S(y, v))$
  - $S_3(u, v)$ pour $\exists y_1 \ \exists y_2 \ (S(u, y_1) \wedge S(y_1, y_2) \wedge S(y_2, v))$
  - etc.
  - $S_{-i}(u, v)$ pour $S_i(v, u)$

Entscheidungsproblem

- Les formules suivantes sont prouvable à partir de (1)–(3), donc satisfaites dans $\mathcal{M}^*$.

$$\forall u\ \forall v\ \forall w\ (((S_i(u,v) \wedge S_i(u,w)) \rightarrow v = w) \wedge ((S_i(v,u) \wedge S_i(w,u)) \rightarrow v = w)) \tag{4}$$

$$\forall u\ \forall v\ (S_i(u,v) \rightarrow u < v) \quad \text{si } i \neq 0 \tag{5}$$

$$\forall u\ \forall v\ (S_i(u,v) \rightarrow u \neq v) \quad \text{si } i \neq 0 \tag{6}$$

$$\forall u\ \forall v\ \forall w\ ((S_i(u,w) \wedge S(w,v)) \rightarrow S_j(u,v)) \quad \text{si } j = i+1 \tag{7}$$

$$\forall u\ \forall v\ \forall w\ ((S_j(u,w) \wedge S(v,w)) \rightarrow S_i(u,v)) \quad \text{si } i = j-1 \tag{8}$$

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_1(\mathbf{2})$ pour $\exists y\,(y = \mathbf{2} \wedge Q_1(y))$ etc.

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_1(\mathbf{2})$ pour $\exists y\,(y = \mathbf{2} \wedge Q_1(y))$ etc.

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_1(\mathbf{2})$ pour $\exists y\,(y = \mathbf{2} \wedge Q_1(y))$ etc.

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_1(\mathbf{2})$ pour $\exists y\,(y = \mathbf{2} \wedge Q_1(y))$ etc.

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_1(2)$ pour $\exists y\,(y = 2 \wedge Q_1(y))$ etc.

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_i(\mathbf{2})$ pour $\exists y\,(y = \mathbf{2} \wedge Q_i(y))$

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_i(\mathbf{2})$ pour $\exists y \, (y = \mathbf{2} \wedge Q_i(y))$ i.e.

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_i(\mathbf{2})$ pour $\exists y \ (y = \mathbf{2} \wedge Q_i(y))$ i.e.
  - $\neg Q_i(\mathbf{2})$ pour $\forall y \ (y = \mathbf{2} \Rightarrow \neg Q_i(y))$ i.e.
  - $S(\mathbf{2}, \mathbf{2})$ pour $\exists y \ (y = \mathbf{2} \wedge S(\mathbf{2}, y))$ i.e.

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_i(\mathbf{2})$ pour $\exists y \ (y = \mathbf{2} \wedge Q_i(y))$ i.e.

$$\exists y \ (S_2(\mathbf{0}, y) \wedge Q_i(y)) \text{ i.e.}$$

$$\exists y \ (\exists z \ (S(\mathbf{0}, z) \wedge S(z, y)) \wedge Q_i(y))$$

- $S(2, n)$ pour $\exists y \ (y = 2 \wedge S(y, n))$ ...

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_i(\mathbf{2})$ pour $\exists y\ (y = \mathbf{2} \wedge Q_i(y))$ i.e.

    $\exists y\ (S_2(\mathbf{0}, y) \wedge Q_i(y))$ i.e.

    $\exists y\ (\exists z\ (S(\mathbf{0}, z) \wedge S(z, y)) \wedge Q_i(y))$
  - $S(\mathbf{2}, u)$ pour $\exists y\ (y = \mathbf{2} \wedge S(y, u)) \equiv \ldots$

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_i(\mathbf{2})$ pour $\exists y\ (y = \mathbf{2} \wedge Q_i(y))$ i.e.
    $\exists y\ (S_2(\mathbf{0}, y) \wedge Q_i(y))$ i.e.
    $\exists y\ (\exists z\ (S(\mathbf{0}, z) \wedge S(z, y)) \wedge Q_i(y))$
  - $S(\mathbf{2}, u)$ pour $\exists y\ (y = \mathbf{2} \wedge S(y, u)) \equiv \ldots$

Entscheidungsproblem

- On introduit encore quelques abréviations...
  - $y = \mathbf{1}$ pour $S_1(\mathbf{0}, y)$
  - $y = \mathbf{2}$ pour $S_2(\mathbf{0}, y)$
  - etc.
  - $y = -\mathbf{1}$ pour $S_{-1}(\mathbf{0}, y)$
  - $y = -\mathbf{2}$ pour $S_{-2}(\mathbf{0}, y)$
  - etc.
- Ainsi, on peut écrire des formules du type :
  - $Q_i(\mathbf{2})$ pour $\exists y\ (y = \mathbf{2} \wedge Q_i(y))$ i.e.
    $\exists y\ (S_2(\mathbf{0}, y) \wedge Q_i(y))$ i.e.
    $\exists y\ (\exists z\ (S(\mathbf{0}, z) \wedge S(z, y)) \wedge Q_i(y))$
  - $S(\mathbf{2}, u)$ pour $\exists y\ (y = \mathbf{2} \wedge S(y, u)) \equiv \ldots$

Entscheidungsproblem

- Avec ces abréviations, les formules suivantes sont prouvables à partir de (1)–(3), donc satisfaites dans $\mathcal{M}^*$ (où p et q désignent des entiers relatifs).

$$\boldsymbol{p} \neq \boldsymbol{q} \quad \text{si } p \neq q \tag{9}$$

$$\forall v\ (S(\boldsymbol{m}, v) \to v = \boldsymbol{k}) \quad \text{où } k = m + 1 \tag{10}$$

$$\forall u\ (S(u, \boldsymbol{k}) \to u = \boldsymbol{m}) \quad \text{où } m = k - 1 \tag{11}$$

Entscheidungsproblem

2. Deuxième formule de $\Gamma_{(\mathcal{M},u)}$: la "configuration au temps 0" : l'input $1 \cdots 1$ de taille n est écrit sur la bande.

$$Q_0(\mathbf{0}) \wedge P(\mathbf{0},\mathbf{0}) \wedge M(\mathbf{0},\mathbf{0}) \wedge M(\mathbf{0},\mathbf{1}) \wedge \ldots \wedge M(\mathbf{0},\boldsymbol{n}) \wedge \forall x \, ((x \neq \mathbf{0} \wedge x \neq \mathbf{0} \wedge \ldots \wedge x \neq \boldsymbol{n-1}) \rightarrow \neg M(\mathbf{0},x)) \tag{12}$$

- Cette formule est satisfaite dans $\mathcal{M}^*$.

Entscheidungsproblem

2. Deuxième formule de $\Gamma_{(\mathcal{M},u)}$: la "configuration au temps 0" : l'input $1 \cdots 1$ de taille n est écrit sur la bande.

$$Q_0(\mathbf{0}) \wedge P(\mathbf{0},\mathbf{0}) \wedge M(\mathbf{0},\mathbf{0}) \wedge M(\mathbf{0},\mathbf{1}) \wedge \ldots \wedge M(\mathbf{0},\boldsymbol{n}) \wedge \forall x \, ((x \neq \mathbf{0} \wedge x \neq \mathbf{0} \wedge \ldots \wedge x \neq \boldsymbol{n-1}) \rightarrow \neg M(\mathbf{0},x)) \tag{12}$$

- Cette formule est satisfaite dans $\mathcal{M}^*$.

Entscheidungsproblem

3. Troisièmes formules de $\Gamma_{(\mathcal{M},u)}$: une formule pour chaque instruction non-terminale de $\mathcal{M}$.
   - Rappel : notre MT est sur un alphabet unaire (lettre "1").
   - On suppose sans perte de généralité que les instructions de type "marquage" et de type "mouvement" sont bien séparées dans le programme de la MT $\mathcal{M}$ (cf. deux slides suivants).
   - On abrège les prédicat M par M_1 et sa négation $\neg M$ par M_b.

Entscheidungsproblem

3. Troisièmes formules de $\Gamma_{(\mathcal{M},u)}$: une formule pour chaque instruction non-terminale de $\mathcal{M}$.

- Rappel : notre MT est sur un alphabet unaire (lettre "1").
- On suppose sans perte de généralité que les instructions de type "marquage" et de type "mouvement" sont bien séparées dans le programme de la MT $\mathcal{M}$ (cf. deux slides suivants).
- On abrège les prédicat M par M_1 et sa négation $\neg M$ par M_b.

Entscheidungsproblem

3. Troisièmes formules de $\Gamma_{(\mathcal{M},u)}$: une formule pour chaque instruction non-terminale de $\mathcal{M}$.

- Rappel : notre MT est sur un alphabet unaire (lettre "1").
- On suppose sans perte de généralité que les instructions de type "marquage" et de type "mouvement" sont bien séparées dans le programme de la MT $\mathcal{M}$ (cf. deux slides suivants).
- On abrège les prédicat M par M_1 et sa négation $\neg M$ par M_b.

Entscheidungsproblem

3. Troisièmes formules de $\Gamma_{(\mathcal{M},u)}$: une formule pour chaque instruction non-terminale de $\mathcal{M}$.

- Rappel : notre MT est sur un alphabet unaire (lettre "1").
- On suppose sans perte de généralité que les instructions de type "marquage" et de type "mouvement" sont bien séparées dans le programme de la MT $\mathcal{M}$ (cf. deux slides suivants).
- On abrège les prédicat M par M_1 et sa négation $\neg M$ par M_b.

Entscheidungsproblem

- Instruction de "marquage/effacement" : le symbole est modifié mais la tête de lecture reste sur place ;
- $(q_i, e) \mapsto (q_j, e')$, avec $e \neq e'$ qui valent b (blank) ou 1.

$$\begin{aligned} &\forall t\ \forall x\ (Q_i(t) \wedge P(t,x) \wedge M_e(t,x) \rightarrow \\ &\exists u\ (S(t,u) \wedge P(u,x) \wedge M_{e'}(u,x) \wedge Q_j(u) \wedge \\ &\qquad \forall y\ ((y \neq x \wedge M(t,y)) \rightarrow M(u,y)) \wedge \\ &\qquad \forall y\ ((y \neq x \wedge \neg M(t,y)) \rightarrow \neg M(u,y)))) \end{aligned} \tag{13}$$

- Ces formules sont satisfaites dans $\mathcal{M}^*$.

Entscheidungsproblem

- Instruction de "marquage/effacement" : le symbole est modifié mais la tête de lecture reste sur place ;
- $(q_i, e) \mapsto (q_j, e')$, avec $e \neq e'$ qui valent b (blank) ou 1.

$$\begin{aligned}
&\forall t\ \forall x\ (Q_i(t) \wedge P(t,x) \wedge M_e(t,x) \rightarrow \\
&\exists u\ (S(t,u) \wedge P(u,x) \wedge M_{e'}(u,x) \wedge Q_j(u) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge M(t,y)) \rightarrow M(u,y)) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge \neg M(t,y)) \rightarrow \neg M(u,y))))
\end{aligned} \tag{13}$$

- Ces formules sont satisfaites dans $\mathcal{M}^*$.

Entscheidungsproblem

- Instruction de "marquage/effacement" : le symbole est modifié mais la tête de lecture reste sur place ;
- $(q_i, e) \mapsto (q_j, e')$, avec $e \neq e'$ qui valent b (blank) ou 1.

$$\begin{aligned}
&\forall t\ \forall x\ (Q_i(t) \wedge P(t,x) \wedge M_e(t,x) \rightarrow \\
&\exists u\ (S(t,u) \wedge P(u,x) \wedge M_{e'}(u,x) \wedge Q_j(u) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge M(t,y)) \rightarrow M(u,y)) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge \neg M(t,y)) \rightarrow \neg M(u,y))))
\end{aligned} \tag{13}$$

- Ces formules sont satisfaites dans $\mathcal{M}^*$.

Entscheidungsproblem

- Instruction de "mouvement" : la position de la tête est modifiée (left or right) mais le symbole (e) reste identique ;
- $(q_i, e) \mapsto (q_j, e, L\ ou\ R)$, avec e qui vaut b (blank) ou 1.

$$\begin{aligned}
&\forall t\ \forall x\ (Q_i(t) \wedge P(t,x) \wedge M_e(t,x) \rightarrow \\
&\exists u\ (S(t,u) \wedge \exists y\ (S_{\pm 1}(x,y) \wedge P(u,y) \wedge M_e(u,x)) \wedge Q_j(u) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge M(t,y)) \rightarrow M(u,y)) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge \neg M(t,y)) \rightarrow \neg M(u,y)))
\end{aligned} \tag{14}$$

- Ces formules sont satisfaites dans $\mathcal{M}^*$.

Entscheidungsproblem

- Instruction de "mouvement" : la position de la tête est modifiée (left or right) mais le symbole (e) reste identique ;
- $(q_i, e) \mapsto (q_j, e, L\ ou\ R)$, avec e qui vaut b (blank) ou 1.

$$\begin{aligned}
&\forall t\ \forall x\ (Q_i(t) \wedge P(t,x) \wedge M_e(t,x) \rightarrow \\
&\exists u\ (S(t,u) \wedge \exists y\ (S_{\pm 1}(x,y) \wedge P(u,y) \wedge M_e(u,x)) \wedge Q_j(u) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge M(t,y)) \rightarrow M(u,y)) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge \neg M(t,y)) \rightarrow \neg M(u,y))))
\end{aligned} \tag{14}$$

- Ces formules sont satisfaites dans $\mathcal{M}^*$.

Entscheidungsproblem

- Instruction de "mouvement" : la position de la tête est modifiée (left or right) mais le symbole (e) reste identique ;
- $(q_i, e) \mapsto (q_j, e, L\ ou\ R)$, avec e qui vaut b (blank) ou 1.

$$\begin{aligned}
&\forall t\ \forall x\ (Q_i(t) \wedge P(t,x) \wedge M_e(t,x) \rightarrow \\
&\exists u\ (S(t,u) \wedge \exists y\ (S_{\pm 1}(x,y) \wedge P(u,y) \wedge M_e(u,x)) \wedge Q_j(u) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge M(t,y)) \rightarrow M(u,y)) \wedge \\
&\qquad \forall y\ ((y \neq x \wedge \neg M(t,y)) \rightarrow \neg M(u,y))))
\end{aligned} \tag{14}$$

- Ces formules sont satisfaites dans $\mathcal{M}^*$.

Entscheidungsproblem

On donne la formule $\varphi_{(\mathcal{M},u)}$.

- Pour chaque instruction terminale de $\mathcal{M}$, i.e. de type

$$F_k : (q_k, e) \mapsto (q_{acc} \ ou \ q_{rej}, \_, \_),$$

on ajoute une formule φ_k comme suit :

$$\exists t \ \exists x \ (Q_k(t) \wedge P(t,x) \wedge M_e(t,x)) \tag{15}$$

- On prend $\varphi_{(\mathcal{M},u)}$ comme la disjonction de toutes ces formules, i.e., $\varphi_{(\mathcal{M},u)} = \bigvee_{k=1}^{K} \varphi_k$.
- $\varphi_{(\mathcal{M},u)}$ est satisfaite dans $\mathcal{M}^*$ si et seulement si la computation $\mathcal{M}(u)$ se voit appliquer une instruction de type F_k à un certain moment.

Entscheidungsproblem

On donne la formule $\varphi_{(\mathcal{M},u)}$.

- Pour chaque instruction terminale de $\mathcal{M}$, i.e. de type

$$F_k : (q_k, e) \mapsto (q_{acc} \ ou \ q_{rej}, \_, \_),$$

on ajoute une formule φ_k comme suit :

$$\exists t \ \exists x \ (Q_k(t) \wedge P(t,x) \wedge M_e(t,x)) \tag{15}$$

- On prend $\varphi_{(\mathcal{M},u)}$ comme la disjonction de toutes ces formules, i.e., $\varphi_{(\mathcal{M},u)} = \bigvee_{k=1}^{K} \varphi_k$.
- $\varphi_{(\mathcal{M},u)}$ est satisfaite dans $\mathcal{M}^*$ si et seulement si la computation $\mathcal{M}(u)$ se voit appliquer une instruction de type F_k à un certain moment.

Entscheidungsproblem

On donne la formule $\varphi_{(\mathcal{M},u)}$.

- Pour chaque instruction terminale de $\mathcal{M}$, i.e. de type

$$F_k : (q_k, e) \mapsto (q_{acc}\ ou\ q_{rej}, \_, \_),$$

on ajoute une formule φ_k comme suit :

$$\exists t\ \exists x\ (Q_k(t) \wedge P(t,x) \wedge M_e(t,x)) \tag{15}$$

- On prend $\varphi_{(\mathcal{M},u)}$ comme la disjonction de toutes ces formules, i.e., $\varphi_{(\mathcal{M},u)} = \bigvee_{k=1}^{K} \varphi_k$.
- $\varphi_{(\mathcal{M},u)}$ est satisfaite dans $\mathcal{M}^*$ si et seulement si la computation $\mathcal{M}(u)$ se voit appliquer une instruction de type F_k à un certain moment.

Entscheidungsproblem

- $\Gamma_{(\mathcal{M},u)}$ est formée des formules (1), (2), (3), (12), (13), (14).
- $\varphi_{(\mathcal{M},u)}$ est formée de la disjonction des formules (15).
- On va montrer que $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ si et seulement si la computation $\mathcal{M}(u)$ s'arrête.

Entscheidungsproblem

- $\Gamma_{(\mathcal{M},u)}$ est formée des formules (1), (2), (3), (12), (13), (14).
- $\varphi_{(\mathcal{M},u)}$ est formée de la disjonction des formules (15).
- On va montrer que $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ si et seulement si la computation $\mathcal{M}(u)$ s'arrête.

Entscheidungsproblem

- $\Gamma_{(\mathcal{M},u)}$ est formée des formules (1), (2), (3), (12), (13), (14).
- $\varphi_{(\mathcal{M},u)}$ est formée de la disjonction des formules (15).
- On va montrer que $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ si et seulement si la computation $\mathcal{M}(u)$ s'arrête.

Entscheidungsproblem

Lemme 21

Si $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$, alors la computation $\mathcal{M}(u)$ s'arrête.

Preuve :

- Supposons que $\mathcal{M}(u)$ ne s'arrête pas. Alors, $\mathcal{M}^* \not\models \varphi_{(\mathcal{M},u)}$.
- D'autre part, on a vu au cours de notre construction que $\mathcal{M}^* \models \Gamma_{(\mathcal{M},u)}$.
- Ainsi, $\Gamma_{(\mathcal{M},u)} \not\vdash \varphi_{(\mathcal{M},u)}$, et donc, par complétude, on a $\Gamma_{(\mathcal{M},u)} \forall \varphi_{(\mathcal{M},u)}$.

□

Entscheidungsproblem

Lemme 21

Si $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$, alors la computation $\mathcal{M}(u)$ s'arrête.

Preuve :

- Supposons que $\mathcal{M}(u)$ ne s'arrête pas. Alors, $\mathcal{M}^* \not\models \varphi_{(\mathcal{M},u)}$.
- D'autre part, on a vu au cours de notre construction que $\mathcal{M}^* \models \Gamma_{(\mathcal{M},u)}$.
- Ainsi, $\Gamma_{(\mathcal{M},u)} \not\models \varphi_{(\mathcal{M},u)}$, et donc, par complétude, on a $\Gamma_{(\mathcal{M},u)} \not\vdash \varphi_{(\mathcal{M},u)}$.

□

Entscheidungsproblem

Lemme 21

Si $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$, alors la computation $\mathcal{M}(u)$ s'arrête.

Preuve :

- Supposons que $\mathcal{M}(u)$ ne s'arrête pas. Alors, $\mathcal{M}^* \not\models \varphi_{(\mathcal{M},u)}$.
- D'autre part, on a vu au cours de notre construction que $\mathcal{M}^* \models \Gamma_{(\mathcal{M},u)}$.
- Ainsi, $\Gamma_{(\mathcal{M},u)} \not\models \varphi_{(\mathcal{M},u)}$, et donc, par complétude, on a $\Gamma_{(\mathcal{M},u)} \not\vdash \varphi_{(\mathcal{M},u)}$.

□

Entscheidungsproblem

Lemme 21

Si $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$, alors la computation $\mathcal{M}(u)$ s'arrête.

Preuve :

- Supposons que $\mathcal{M}(u)$ ne s'arrête pas. Alors, $\mathcal{M}^* \not\models \varphi_{(\mathcal{M},u)}$.
- D'autre part, on a vu au cours de notre construction que $\mathcal{M}^* \models \Gamma_{(\mathcal{M},u)}$.
- Ainsi, $\Gamma_{(\mathcal{M},u)} \not\models \varphi_{(\mathcal{M},u)}$, et donc, par complétude, on a $\Gamma_{(\mathcal{M},u)} \not\vdash \varphi_{(\mathcal{M},u)}$.

□

Entscheidungsproblem

Lemme 22

Si la computation $\mathcal{M}(u)$ s'arrête, alors $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$.

Preuve :

- On introduit les formules suivantes : si, au temps $a \geq 0$, la MT $\mathcal{M}$ sur l'input u est dans l'état q_i, à la cellule no p, avec les cellules no $p_1, \ldots, p_m$ qui sont marquées, alors la *description au temps a* "*time a*" est :

$$Q_i(\mathbf{a}) \wedge P(\mathbf{a}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge \forall x \left((x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x) \right) \tag{16}$$

Entscheidungsproblem

Lemme 22

Si la computation $\mathcal{M}(u)$ s'arrête, alors $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$.

Preuve :

- On introduit les formules suivantes : si, au temps $a \geq 0$, la MT $\mathcal{M}$ sur l'input u est dans l'état q_i, à la cellule no p, avec les cellules no $p_1, \ldots, p_m$ qui sont marquées, alors la *description au temps* a "$time\ a$" est :

$$Q_i(\mathbf{a}) \wedge P(\mathbf{a}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge \forall x \left((x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x) \right) \tag{16}$$

Entscheidungsproblem

- **Remarque :** la formule (16) contient directement ou indirectement l'information de si la cellule courante p est marquée ou non, en effet :

- si p est marquée au temps a, alors p est un des p_i's ;
  donc $M(\mathbf{a}, \mathbf{p})$ apparaît directement dans (16).
- si p n'est pas marquée au temps a, alors p n'est un des p_i's ;
  par (9), on peut prouver que $\mathbf{p} \neq \mathbf{p_i}$ pour tout $i = 1, \ldots, m$;
  par (16), on a $\neg M(\mathbf{a}, \mathbf{p})$;
  donc $\neg M(\mathbf{a}, \mathbf{p})$ est prouvable à partir de (16) et $\Gamma_{(\mathcal{M},u)}$.

Entscheidungsproblem

- **Remarque :** la formule (16) contient directement ou indirectement l'information de si la cellule courante p est marquée ou non, en effet :
- si p est marquée au temps a, alors p est un des p_i's ;
  donc $M(\mathbf{a}, \mathbf{p})$ apparaît directement dans (16).
- si p n'est pas marquée au temps a, alors p n'est un des p_i's ;
  par (9), on peut prouver que $\mathbf{p} \neq \mathbf{p_i}$ pour tout $i = 1, \ldots, m$;
  par (16), on a $\neg M(\mathbf{a}, \mathbf{p})$;
  donc $\neg M(\mathbf{a}, \mathbf{p})$ est prouvable à partir de (16) et $\Gamma_{(\mathcal{M},u)}$.

Entscheidungsproblem

- **Remarque :** la formule (16) contient directement ou indirectement l'information de si la cellule courante p est marquée ou non, en effet :
- si p est marquée au temps a, alors p est un des p_i's ;
  donc $M(\mathbf{a}, \mathbf{p})$ apparaît directement dans (16).
- si p n'est pas marquée au temps a, alors p n'est un des p_i's ;
  par (9), on peut prouver que $\mathbf{p} \neq \mathbf{p_i}$ pour tout $i = 1, \ldots, m$;
  par (16), on a $\neg M(\mathbf{a}, \mathbf{p})$;
  donc $\neg M(\mathbf{a}, \mathbf{p})$ est prouvable à partir de (16) et $\Gamma_{(\mathcal{M},u)}$.

Entscheidungsproblem

Fait 1

Si $\mathcal{M}(u)$ s'arrête au temps $b = a + 1$, alors

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \varphi_{(\mathcal{M},u)}.$$

Entscheidungsproblem

- Si $\mathcal{M}(u)$ s'arrête au temps $b = a + 1$, alors l'instruction appliquée au temps a est du type $(q_i, e) \mapsto (q_{acc}\ ou\ q_{rej}, \_, \_)$.
- La description au temps a nous dit que $\mathcal{M}$ est dans l'état q_i et scanne la cellule p, i.e., $time\ a \vdash Q_i(\mathbf{a}), P(\mathbf{a}, \mathbf{p})$.
- De plus, par la remarque précédente, si p est marquée au temps a, alors $time\ a \vdash M(\mathbf{a}, \mathbf{p})$, et si p n'est pas marquée au temps a, alors $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$.
- On déduit que pour $e = 1$ ou b :

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$$

- Puisque $\varphi_{(\mathcal{M},u)} = \bigvee_{i=1}^{K} \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$, on déduit $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \varphi_{(\mathcal{M},u)}$. □

Entscheidungsproblem

- Si $\mathcal{M}(u)$ s'arrête au temps $b = a + 1$, alors l'instruction appliquée au temps a est du type $(q_i, e) \mapsto (q_{acc}\ ou\ q_{rej}, \_, \_)$.
- La description au temps a nous dit que $\mathcal{M}$ est dans l'état q_i et scanne la cellule p, i.e., $time\ a \vdash Q_i(\mathbf{a}), P(\mathbf{a}, \mathbf{p})$.
- De plus, par la remarque précédente, si p est marquée au temps a, alors $time\ a \vdash M(\mathbf{a}, \mathbf{p})$, et si p n'est pas marquée au temps a, alors $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$.
- On déduit que pour $e = 1$ ou b :

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$$

- Puisque $\varphi_{(\mathcal{M},u)} = \bigvee_{i=1}^{K} \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$, on déduit $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \varphi_{(\mathcal{M},u)}$. □

Entscheidungsproblem

- Si $\mathcal{M}(u)$ s'arrête au temps $b = a + 1$, alors l'instruction appliquée au temps a est du type $(q_i, e) \mapsto (q_{acc}\ ou\ q_{rej}, \_, \_)$.
- La description au temps a nous dit que $\mathcal{M}$ est dans l'état q_i et scanne la cellule p, i.e., $time\ a \vdash Q_i(\mathbf{a}), P(\mathbf{a}, \mathbf{p})$.
- De plus, par la remarque précédente, si p est marquée au temps a, alors $time\ a \vdash M(\mathbf{a}, \mathbf{p})$, et si p n'est pas marquée au temps a, alors $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$.
- On déduit que pour $e = 1$ ou b :

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$$

- Puisque $\varphi_{(\mathcal{M},u)} = \bigvee_{i=1}^{K} \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$, on déduit $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \varphi_{(\mathcal{M},u)}$. □

Entscheidungsproblem

- Si $\mathcal{M}(u)$ s'arrête au temps $b = a + 1$, alors l'instruction appliquée au temps a est du type $(q_i, e) \mapsto (q_{acc}\ ou\ q_{rej}, \_, \_)$.
- La description au temps a nous dit que $\mathcal{M}$ est dans l'état q_i et scanne la cellule p, i.e., $time\ a \vdash Q_i(\mathbf{a}), P(\mathbf{a}, \mathbf{p})$.
- De plus, par la remarque précédente, si p est marquée au temps a, alors $time\ a \vdash M(\mathbf{a}, \mathbf{p})$, et si p n'est pas marquée au temps a, alors $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$.
- On déduit que pour $e = 1$ ou b :

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$$

- Puisque $\varphi_{(\mathcal{M},u)} = \bigvee_{i=1}^{K} \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$, on déduit $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \varphi_{(\mathcal{M},u)}$. □

Entscheidungsproblem

- Si $\mathcal{M}(u)$ s'arrête au temps $b = a + 1$, alors l'instruction appliquée au temps a est du type $(q_i, e) \mapsto (q_{acc}\ ou\ q_{rej}, \_, \_)$.
- La description au temps a nous dit que $\mathcal{M}$ est dans l'état q_i et scanne la cellule p, i.e., $time\ a \vdash Q_i(\mathbf{a}), P(\mathbf{a}, \mathbf{p})$.
- De plus, par la remarque précédente, si p est marquée au temps a, alors $time\ a \vdash M(\mathbf{a}, \mathbf{p})$, et si p n'est pas marquée au temps a, alors $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$.
- On déduit que pour $e = 1$ ou b :

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$$

- Puisque $\varphi_{(\mathcal{M},u)} = \bigvee_{i=1}^{K} \exists t\ \exists x\ (Q_i(t) \wedge P(t, x) \wedge M_e(t, x))$, on déduit $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \varphi_{(\mathcal{M},u)}$. □

Entscheidungsproblem

Fait 2

Si $\mathcal{M}(u)$ ne s'arrête pas au temps $b = a + 1$, alors

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash time\ b \quad \textit{i.e.,}$$

$$\Gamma_{(\mathcal{M},u)} \vdash time\ a \rightarrow time\ b$$

- On fait la preuve si l'instruction (entre le temps a et $b = a+1$) est du type "marquage", i.e., $(q_i, b) \mapsto (q_j, 1)$; la preuve pour les autres types d'instructions est similaire.

Entscheidungsproblem

Fait 2

Si $\mathcal{M}(u)$ ne s'arrête pas au temps $b = a + 1$, alors

$$\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash time\ b \quad \textit{i.e.,}$$

$$\Gamma_{(\mathcal{M},u)} \vdash time\ a \rightarrow time\ b$$

- On fait la preuve si l'instruction (entre le temps a et $b = a+1$) est du type "marquage", i.e., $(q_i, b) \mapsto (q_j, 1)$; la preuve pour les autres types d'instructions est similaire.

Entscheidungsproblem

- La formule $time\ a$ est du type :

$$Q_i(\mathbf{a}) \wedge P(\mathbf{a}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x))$$

- Par la remarque, $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$
- La formule de $\Gamma_{(\mathcal{M},u)}$ relative à l'instruction en question est :

$$\forall t\ \forall x\ (Q_i(t) \wedge P(t, x) \wedge \neg M(t, x) \rightarrow$$
$$\exists u\ (S(t, u) \wedge P(u, x) \wedge M(u, x) \wedge Q_j(u) \wedge$$
$$\forall y\ ((y \neq x \wedge \neg M(t, y)) \rightarrow \neg M(u, y))))$$

- On peut donc déduire "$time\ b$" des deux formules précédentes

$$Q_i(\mathbf{b}) \wedge P(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p} \wedge x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x)) \quad \square$$

Entscheidungsproblem

- La formule $time\ a$ est du type :

$$Q_i(\mathbf{a}) \wedge P(\mathbf{a}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x))$$

- Par la remarque, $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$
- La formule de $\Gamma_{(\mathcal{M},u)}$ relative à l'instruction en question est :

$$\forall t\ \forall x\ (Q_i(t) \wedge P(t,x) \wedge \neg M(t,x) \rightarrow$$
$$\exists u\ (S(t,u) \wedge P(u,x) \wedge M(u,x) \wedge Q_j(u) \wedge$$
$$\forall y\ ((y \neq x \wedge \neg M(t,y)) \rightarrow \neg M(u,y))))$$

- On peut donc déduire "$time\ b$" des deux formules précédentes

$$Q_i(\mathbf{b}) \wedge P(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p} \wedge x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x)) \quad \square$$

Entscheidungsproblem

- La formule $time\ a$ est du type :

$$Q_i(\mathbf{a}) \wedge P(\mathbf{a}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x))$$

- Par la remarque, $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$
- La formule de $\Gamma_{(\mathcal{M},u)}$ relative à l'instruction en question est :

$$\forall t\ \forall x\ (Q_i(t) \wedge P(t, x) \wedge \neg M(t, x) \rightarrow$$
$$\exists u\ (S(t, u) \wedge P(u, x) \wedge M(u, x) \wedge Q_j(u) \wedge$$
$$\forall y\ ((y \neq x \wedge \neg M(t, y)) \rightarrow \neg M(u, y))))$$

- On peut donc déduire "$time\ b$" des deux formules précédentes

$$Q_i(\mathbf{b}) \wedge P(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p} \wedge x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x)) \quad \square$$

Entscheidungsproblem

- La formule $time\ a$ est du type :

$$Q_i(\mathbf{a}) \wedge P(\mathbf{a}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x))$$

- Par la remarque, $\Gamma_{(\mathcal{M},u)} \cup \{time\ a\} \vdash \neg M(\mathbf{a}, \mathbf{p})$
- La formule de $\Gamma_{(\mathcal{M},u)}$ relative à l'instruction en question est :

$$\forall t\ \forall x\ (Q_i(t) \wedge P(t, x) \wedge \neg M(t, x) \rightarrow$$
$$\exists u\ (S(t, u) \wedge P(u, x) \wedge M(u, x) \wedge Q_j(u) \wedge$$
$$\forall y\ ((y \neq x \wedge \neg M(t, y)) \rightarrow \neg M(u, y))))$$

- On peut donc déduire "$time\ b$" des deux formules précédentes

$$Q_i(\mathbf{b}) \wedge P(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{b}, \mathbf{p}) \wedge M(\mathbf{a}, \mathbf{p_1}) \wedge \cdots \wedge M(\mathbf{a}, \mathbf{p_m}) \wedge$$
$$\forall x\ ((x \neq \mathbf{p} \wedge x \neq \mathbf{p_1} \wedge \cdots \wedge x \neq \mathbf{p_m}) \rightarrow \neg M(\mathbf{a}, x)) \qquad \square$$

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- On finit la preuve du Lemme 22.
- Supposons que $\mathcal{M}(u)$ s'arrête au temps n.
- Par définition, $time\ 0 \in \Gamma_{(\mathcal{M},u)}$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 0$.
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 0 \rightarrow time\ 1$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 1$
- (Fait 2) $\Gamma_{(\mathcal{M},u)} \vdash time\ 1 \rightarrow time\ 2$, donc $\Gamma_{(\mathcal{M},u)} \vdash time\ 2$
- Après itération du Fait 2 pour tous les temps k auxquels la machines ne s'arrête pas, on obtient $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1)$
- (Fait 1) $\Gamma_{(\mathcal{M},u)} \vdash time\ (n-1) \rightarrow \varphi_{(\mathcal{M},u)}$
- (modus ponens) $\Gamma_{(\mathcal{M},u)} \vdash \varphi_{(\mathcal{M},u)}$ □

Entscheidungsproblem

- Finalement, la preuve du Théorème 23 est une conséquence directe des lemmes 21 et 22. □

Théorème 23 (Church-Turing 1936)

- *Le Entscheidungsproblem pour la logique du premier ordre est Turing-indécidable.*
- *Étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, le problème de savoir si $\Gamma \vdash \varphi$ ou non est Turing-indécidable.*
- *Le langage $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ est Turing-indécidable.*

Entscheidungsproblem

- Finalement, la preuve du Théorème 23 est une conséquence directe des lemmes 21 et 22. □

Théorème 23 (Church-Turing 1936)

- *Le Entscheidungsproblem pour la logique du premier ordre est Turing-indécidable.*
- *Étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, le problème de savoir si $\Gamma \vdash \varphi$ ou non est Turing-indécidable.*
- *Le langage $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ est Turing-indécidable.*

Entscheidungsproblem

- Finalement, la preuve du Théorème 23 est une conséquence directe des lemmes 21 et 22. □

Théorème 23 (Church-Turing 1936)

- *Le Entscheidungsproblem pour la logique du premier ordre est Turing-indécidable.*
- *Étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, le problème de savoir si $\Gamma \vdash \varphi$ ou non est Turing-indécidable.*
- *Le langage $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ est Turing-indécidable.*

Entscheidungsproblem

- Finalement, la preuve du Théorème 23 est une conséquence directe des lemmes 21 et 22. □

Théorème 23 (Church-Turing 1936)

- *Le Entscheidungsproblem pour la logique du premier ordre est Turing-indécidable.*
- *Étant donné un ensemble d'axiomes Γ et un énoncé φ de la logique du premier ordre, le problème de savoir si $\Gamma \vdash \varphi$ ou non est Turing-indécidable.*
- *Le langage $\{\langle \Gamma, \varphi \rangle : \Gamma \vdash \varphi\}$ est Turing-indécidable.*

Entscheidungsproblem

- L'utilisation de symboles de prédicats binaires est cruciale dans notre preuve d'indécidabilité.
- Si le langage ne contient que des symboles de prédicats unaires et de constante (pas de symboles de fonction), alors le calcul des prédicats égalitaire du premier ordre correspondant, le *calcul des prédicats monadiques du premier ordre*, est décidable.
- Par ailleurs, il existe aussi des théories décidables dont le langage contient un symbole de prédicat binaire : *la théorie des ordres denses*, ou encore *l'arithmétique de Presburger*.

Entscheidungsproblem

- L'utilisation de symboles de prédicats binaires est cruciale dans notre preuve d'indécidabilité.
- Si le langage ne contient que des symboles de prédicats unaires et de constante (pas de symboles de fonction), alors le calcul des prédicats égalitaire du premier ordre correspondant, le *calcul des prédicats monadiques du premier ordre*, est décidable.
- Par ailleurs, il existe aussi des théories décidables dont le langage contient un symbole de prédicat binaire : *la théorie des ordres denses*, ou encore *l'arithmétique de Presburger*.

Entscheidungsproblem

- L'utilisation de symboles de prédicats binaires est cruciale dans notre preuve d'indécidabilité.
- Si le langage ne contient que des symboles de prédicats unaires et de constante (pas de symboles de fonction), alors le calcul des prédicats égalitaire du premier ordre correspondant, le *calcul des prédicats monadiques du premier ordre*, est décidable.
- Par ailleurs, il existe aussi des théories décidables dont le langage contient un symbole de prédicat binaire : *la théorie des ordres denses*, ou encore *l'arithmétique de Presburger*.

Indécidabilité et incomplétude

It should perhaps be remarked that what I shall prove is quite different from the well-known results of Gödel†. Gödel has shown that (in the formalism of Principia Mathematica) there are propositions $\mathfrak{A}$ such that neither $\mathfrak{A}$ nor $-\mathfrak{A}$ is provable. As a consequence of this, it is shown that no proof of consistency of Principia Mathematica (or of **K**) can be given within that formalism. On the other hand, I shall show that there is no general method which tells whether a given formula $\mathfrak{A}$ is provable in **K**, or, what comes to the same, whether the system consisting of **K** with $-\mathfrak{A}$ adjoined as an extra axiom is consistent.

If the negation of what Gödel has shown had been proved, *i.e.* if, for each $\mathfrak{A}$, either $\mathfrak{A}$ or $-\mathfrak{A}$ is provable, then we should have an immediate solution of the Entscheidungsproblem. For we can invent a machine $\mathscr{K}$ which will prove consecutively all provable formulae. Sooner or later $\mathscr{K}$ will reach either $\mathfrak{A}$ or $-\mathfrak{A}$. If it reaches $\mathfrak{A}$, then we know that $\mathfrak{A}$ is provable. If it reaches $-\mathfrak{A}$, then, since **K** is consistent (Hilbert and Ackermann, p. 65), we know that $\mathfrak{A}$ is not provable.

Figure : Incomplétude et indécidabilité selon Turing

Résumé

- Les machines de Turing représentent un modèle de calculabilité pertinent.

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions Turing-calculables.

- Le concept de machine de Turing englobe le pouvoir computationnel de tous les ordinateurs (digitaux) actuels.
- Il existe une multitude de problèmes Turing-indécidables (problème de l'arrêt).
- On a montré l'indécidabilité de la logique du premier ordre par réduction à l'indécidabilité du problème de l'arrêt.

Résumé

- Les machines de Turing représentent un modèle de calculabilité pertinent.

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions Turing-calculables.

- Le concept de machine de Turing englobe le pouvoir computationnel de tous les ordinateurs (digitaux) actuels.
- Il existe une multitude de problèmes Turing-indécidables (problème de l'arrêt).
- On a montré l'indécidabilité de la logique du premier ordre par réduction à l'indécidabilité du problème de l'arrêt.

Résumé

- Les machines de Turing représentent un modèle de calculabilité pertinent.

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions Turing-calculables.

- Le concept de machine de Turing englobe le pouvoir computationnel de tous les ordinateurs (digitaux) actuels.
- Il existe une multitude de problèmes Turing-indécidables (problème de l'arrêt).
- On a montré l'indécidabilité de la logique du premier ordre par réduction à l'indécidabilité du problème de l'arrêt.

Résumé

- Les machines de Turing représentent un modèle de calculabilité pertinent.

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions Turing-calculables.

- Le concept de machine de Turing englobe le pouvoir computationnel de tous les ordinateurs (digitaux) actuels.
- Il existe une multitude de problèmes Turing-indécidables (problème de l'arrêt).
- On a montré l'indécidabilité de la logique du premier ordre par réduction à l'indécidabilité du problème de l'arrêt.

Résumé

- Les machines de Turing représentent un modèle de calculabilité pertinent.

  Thèse de Church-Turing : Les fonctions calculables de manière effective correspondent exactement aux fonctions Turing-calculables.

- Le concept de machine de Turing englobe le pouvoir computationnel de tous les ordinateurs (digitaux) actuels.
- Il existe une multitude de problèmes Turing-indécidables (problème de l'arrêt).
- On a montré l'indécidabilité de la logique du premier ordre par réduction à l'indécidabilité du problème de l'arrêt.