Théorie de la complexité et calcul non déterministe

3.1 Temps polynomial non déterministe – NP

- Considérez l'algorithme suivant pour le problème de décision SAT

Algorithm *Naive SAT-solver.*

Input: a Boolean formula $f(x_1, \ldots, x_n)$ in CNF.
Output: true if f is satisfiable and false otherwise.
Algorithm:
for each possible truth assignment $\mathbf{x} \in \{0, 1\}^n$
    if $f(\mathbf{x}) = 1$ then output true.
next $\mathbf{x}$
output false

Cet algorithme est totalement impraticable

- car si f n'est pas satisfaisable, alors il essaiera toutes les 2^n affectations de vérité possibles avant de s'arrêter
- Nous souhaitons identifier ces problèmes de décision, tels que SAT, avec la propriété que si une instance donnée du problème est vraie, alors il existe un « certificat succinct » qui le vérifier dans un temps polynomial.
- Supposons que nous ayons une instance, f (x_1,..., x_n), de SAT que nous savions satisfiable.
- Pouvons-nous convaincre un observateur sceptique de ce fait dans un délai raisonnable ?
- Certes, donnez simplement à l'observateur l'instance f avec une affectation de vérité satisfaisante x.
- La procédure de contrôle de l’observateur pourrait clairement être mise en œuvre comme algorithme de temps polynomial
- une assignation de vérité satisfaisante est un certificat succinct de la satisfiabilité de f, car elle certifie que f est satisfiable et peut être vérifiée rapidement.

- Prenons l'exemple de langage COMPOSÉ
- Dans ce cas, un certificat succinct prouvant qu'un entier n particulier est composé et un facteur propre et non trivial pour n.
- Compte tenu d'un tel facteur, nous pouvons facilement vérifier en temps polynomial qu’il divise n exactement

Attention: trouver un tel facteur en temps polynomial est un problème complètement distinct

- Notre observateur sceptique pourrait utiliser l'algorithme de vérification du temps polynomial suivant :

Algorithm *Factor checking.*

Input: integer n and possible factor d.
Output: true iff d is a proper non-trivial factor of n.
Checking algorithm:
if d divides n exactly and $2 \leq d \leq n - 1$
    then output true
    else output false.

Si n est composé, alors pour un choix approprié de d, l'algorithme de vérification vérifiera ce fait.

Si n est premier, quelle que soit la valeur de d donnée à l'algorithme de vérification, il produira toujours faux

Il s'agit clairement d'un algorithme de temps polynomial.

- Lorsqu'un problème de décision Π a un certificat succinct qui peut être utilisé pour vérifier qu'une instance donnée est vraie en temps polynomial, alors nous disons que le langage associé L_Π est accepté en temps polynomial non déterministe.
- De manière équivalente, nous disons que L_Π appartient à la classe de complexité NP.
- Nous pouvons formaliser cette définition comme suit. Pour x, y $\in \sum_0^*$, on note x y la chaîne constituée de x suivi d'un carré vide, suivi de y. Un langage $L \subseteq \sum_0^*$ est dit appartenir à NP s'il existe une DTM M et un polynôme p(n) tels que $T_M(n) \leq p(n)$ et sur toute entrée $x \in \sum_0^*$:
- (i) si $x \in L$ alors il existe un certificat $y \in$ tel que $| y | \leq p(| x |)$ et M accepte la chaîne d'entrée x y;
- (ii) si $x \notin L$ alors pour toute chaîne $y \in \sum_0^*$, M rejette la chaîne d'entrée x y.

- Une question évidente à se poser est de savoir comment la classe NP est liée à P. Il est facile de voir que $P \subseteq NP$.
- Proposition: $P \subseteq NP$.
- Preuve: Si $L \in P$ alors il y a une DTM à temps polynomial qui décide L. Par conséquent, nous n'avons pas besoin d'un certificat pour vérifier qu'une entrée particulière $x \in \sum_0^*$ appartient à L. Notre algorithme de vérification prend simplement une entrée $x \in \sum_0^*$ et décide si x appartient ou non à L directement, en temps polynomial.
- Vérifier un certificat semble être une tâche beaucoup plus facile que de décider si un tel certificat existe. En effet, il est largement admis que $P \neq NP$. Si ce le cas alors Combien NP peut-il être plus grand que P?

- Notre prochain résultat dit que n'importe quel langage dans NP peut être décidé dans un espace polynomial.
- Théorème : NP ⊆ PSPACE
- Preuve: Nous essayons simplement chaque certificat possible à tour de rôle. Comme tout certificat possible est de longueur polynomiale, nous pouvons vérifier tous les certificats possibles en utilisant une quantité d'espace polynomiale en réutilisant les mêmes carrés de bande pour les certificats successifs.

3.2 Réductions à temps polynomiales

- Il existe de nombreuses situations où la capacité de résoudre un problème Π1 nous permettrait également de résoudre un problème Π2. L'exemple le plus simple de ce phénomène est lorsque nous pouvons convertir une instance, I, de Π1 en une instance, f (I), de Π2 et en résolvant f (I) on obtenait une réponse pour I.
- Considérez le problème de décision suivant.

ENSEMBLE INDÉPENDANT

Entrée: un graphe G et un entier k.

Question: G contient-il un ensemble indépendant d'ordre k?

Ceci est évidemment étroitement lié au problème CLIQUE

- Supposons que nous ayons un algorithme efficace pour CLIQUE puis donné une instance de ENSEMBLE INDÉPENDANT
- consistant en un graphe G et un entier k, nous pourrions former le graphe Gc, le complément de G.
- C'est le graphe sur le même ensemble de sommets que G mais avec une arête présente dans Gc si et seulement si elle est absente de G.
- Passez maintenant Gc et k à notre algorithme pour CLIQUE. Il renverra la réponse vraie si et seulement si le graphe original contenait un ensemble indépendant d'ordre k.
- Par conséquent, la capacité de résoudre CLIQUE nous permettrait également de résoudre l’ENSEMBLE INDÉPENDANT
- De plus, la conversion d'une instance de l’ENSEMBLE INDÉPENDANT en une instance de CLIQUE pourrait clairement être réalisée en temps polynomial.

- Nous formalisons cette idée de réduction du temps polynomiale comme suit.
- Si $A, B \subseteq \sum_0^*$ et $f: \sum_0^* \rightarrow \sum_0^*$ vérifie $x \in A \Leftrightarrow f(x) \in B$, alors f est une réduction de A à B.
- Si en plus $f \in FP$ alors f est une réduction du temps polynomiale de A à B.
- Lorsqu'une telle fonction existe, nous disons que A est polynomialement réductible à B et écrivons $A \leq_m B$.
- Si $A \leq_m B$ et B est « facile » alors A est lui aussi
- Si $A \leq_m B$ et $B \in P$ alors $A \in P$
- Si $B \in NP$ et $A \leq_m B$ alors $A \in NP$.
- Si $A \leq_m B$ et $B \leq_m C$ alors $A \leq_m C$. (la relation $\leq_m$ est transitive)
- Si B est au moins aussi difficile que A, et C est au moins aussi difficile que B, alors C est au moins aussi difficile que A.

3.3 NP-Complétude

- un langage L est NP-complet si
- (i) $L \in NP$,
- (ii) si $A \in NP$ alors $A \leq_m L$.
- C.à.d. L est un langage qui appartiennent à NP et qui est au moins aussi difficiles que n'importe quel autre langage dans NP
- Proposition: Si un langage NP-complet appartient à P alors P = NP.
- Preuve : Puisque $P \subseteq NP$ il suffit de montrer que $NP \subseteq P$. Supposons que L est NP-complet et appartient aussi à P. Si $A \in NP$ alors $A \leq_m L$ et donc selon slide11 $A \in P$. D'où $NP \subseteq P$.
- Si L est NP-complet, $A \in NP$ et $L \leq_m A$ alors A est aussi NP-complet.
- Exemples très naturel d'un langage NP-complet: SAT et 3-SAT

3.4 Réductions de Turing

- Une restriction des réductions de temps polynomiales est que nous devons convertir une instance d'un problème en une seule instance d'un autre.
- Il y a des situations où la capacité de résoudre un problème Π_1 de manière efficace nous permettra de résoudre un problème Π_2 de manière efficace, mais dans le processus, nous devons être capables de résoudre plus d'une instance de Π_1.
- Par exemple, 2-SAT(voir TD 2) peut être résolu en appelant à plusieurs reprises un sous-programme pour le problème ACCESSIBILITÉ.
- Ce type de réduction plus général est souvent très utile et est en fait la réduction couramment utilisée dans la pratique, par exemple lorsqu'un algorithme appelle un sous-programme à plusieurs reprises.

- Définition : une fonction f est Turing réductible en une fonction g si un algorithme de temps polynomial pour le calcul de g produirait un algorithme de temps polynomial pour le calcul de f.
- Pour décrire plus précisément la réductibilité de Turing, nous introduisons un nouveau type de machine de Turing : une machine de Turing oracle déterministe (DOTM).
- Une telle machine a une bande supplémentaire connue sous le nom de bande de requête et un état spécial: γ_Q, l'état de la requête.
- Il existe également une fonction oracle O associée à la machine.
- Une fonction f est dite Turing réductible en une fonction g, notée $f \leq_T g$, si f peut être calculé en temps polynomial par une DOTM équipée d'un oracle pour g.

La classe NP-dure

- Une fonction f est dite NP-dure s'il existe un langage NP-complet L tel que $L \leq_T f$, où ici on identifie L à f_L (la fonction correspondante au langage)

$$f_L : \Sigma_0^* \to \{0, 1\}, \quad f_L(x) = \begin{cases} 1, & x \in L \\ 0, & x \notin L. \end{cases}$$

Ainsi, une fonction NP-dure est «au moins aussi difficile » que n'importe quel langage dans NP dans le sens où un algorithme de temps polynomial pour calculer une telle fonction produirait un algorithme de temps polynomial pour chaque langage de NP.

Notez qu'un langage peut être NP-dure et en particulier tout langage NP-complet est également NP-dure.

- Soit le problème de décision de graphe suivant

MAX CLIQUE

Entrée: un graphe G et un entier k.

Question: la plus grande clique de G a-t-elle l'ordre k?

Ce problème est clairement au moins aussi difficile que le problème NP-complet CLIQUE mais il n'y a pas de moyen évident de montrer qu'il appartient à NP (car il n'y a pas de certificat évident).

Ni de réduction polynomiale évidente de CLIQUE à MAX CLIQUE.

MAX CLIQUE est NP-dure.

3.5 Compléments de langages dans NP

- Si L ⊆ est un langage alors le complément de L est $L^c = \{x \in \Sigma_0^* \mid x \notin L\}$.

Si C est une classe de complexité alors la classe des compléments de langages en C est notée

$$\text{co-C} = \{L \subseteq \Sigma_0^* \mid L^c \in \text{C}\}.$$

L'exemple le plus important d'une telle classe est co-NP, la collection de compléments de langages dans NP

- Définition: un langage $L \subseteq \sum_0^*$ appartient à co-NP si et seulement s'il existe un DTM M et un polynôme p (n) tels que $T_M(n) \leq p(n)$ et sur toute entrée x $\in \sum_0^*$:
- (i) si $x \notin L$ alors il existe un certificat $y \in \sum_0^*$ tel que $| y | \leq p(| x |)$ et M accepte la chaîne d'entrée x y;
- (ii) si $x \in L$ alors pour toute chaîne $y \in \sum_0^*$, M rejette la chaîne d'entrée x y.

- Pour un problème de décision, le langage complémentaire a une interprétation très naturelle : inverser simplement le vrai et le faux dans la sortie.

- Par exemple, considérez le problème

UNSAT

Entrée: une formule booléenne f en CNF.

Question: est-ce que f est insatisfaisable?

Depuis SAT ∈ NP donc UNSAT ∈ co-NP par définition.

Mais qu'en est-il de SAT lui-même ?

Pour prouver que SAT appartient à co-NP, nous aurions besoin de décrire un certificat succinct pour qu'une formule booléenne CNF soit insatisfaisable.

il apparaît que la seule façon de confirmer qu'une instance de SAT n'est pas satisfaisable est de vérifier toutes les attributions possibles de vérité et de vérifier qu'aucune d'entre elles est satisfaisable, mais c'est évidemment un algorithme de temps exponentiel. Donc, On ne sait pas si SAT appartient à co-NP.

- Cela met en évidence une différence importante entre les classes P et NP
- Dans le cas d'un langage en P nous avons une DTM de temps polynomial qui peut décider L
- donc en inversant la sortie de notre DTM est nous avons une DTM de temps polynomial pour décider L_c
- donc P = co-P.
- Si L ∈ NP on ne peut pas simplement prendre la DTM donné par la définition de NP et produire une nouvelle DTM pour montrer que L_c ∈ NP
- NP $\overset{?}{=}$ co-NP (Question ouverte)
- Si L est NP-complet et L appartient à co-NP alors NP = co-NP.

- la classe des langages co-NP-complets est simplement la classe des compléments des langages NP-complets.
- P = co-P et P ⊆ NP et P ⊆ co-NP. Donc, P ⊆ NP ∩ co-NP.

  ?
- P ⚌ NP ∩ co-NP (Question ouverte)

- le langage COMPOSÉ appartient à NP (puisqu'un certificat succinct pour qu'un entier soit composé est qu'il a un diviseur propre et non trivial)
- le langage complémentaire de COMPOSÉ est PRIMIER, constitué de codages binaires d'entiers premiers.
- Étant donné un entier n, il est difficile de vérifier que n est premier en temps polynomial(Il n'y a pas de certificat succinct immédiatement évident pour la primalité).

- Théorème: Le langage PRIMIER appartient à NP ∩ co-NP.
- En fait, nous avons le résultat exceptionnel suivant grâce à Agrawal en 2002.
- Théorème: Le langage PRIMIER appartient à P.

3.6 Conteneurs entre classes de complexité

- P $\stackrel{?}{=}$ NP (Première question ouverte en théorie de la complexité)
- On sait que $P \subseteq NP \cap co\text{-}NP \subseteq NP$ et on pense généralement que tous ces inclusions sont strictes.
- Nous avons vu de nombreux exemples de langages qui sont soit NP-complets, soit appartiennent à P.
- De plus, le complément de tout langage NP-complet est clairement co-NP-complet(nous pourrions donc facilement donner beaucoup d'exemples de tels langages)
- Les exemples naturels de langages qui sont dans $NP \cap co\text{-}NP$ mais dont on sait qu'ils n'appartiennent pas à P sont relativement rares.
- Un tel exemple est donné par le problème de décision suivant.

FACTEUR

Entrée: entiers n et k.

Question: n a-t-il un facteur d non trivial, satisfaisant $1 < d \leq k$?

- Clairement FACTOR $\in$ NP puisqu'un certificat évident est un facteur d satisfaisant $1 < d \leq k$. Aussi FACTOR $\in$ co-NP en inversant les résultats.
- Cependant, on ne sait pas si FACTOR $\in$ P.
- Si cela était vrai, cela aurait un impact très significatif sur la cryptographie
- Nous n'avons pas encore vu d'exemple de langage qui appartient à NP, mais qui n'est pas considéré comme NP-complet ni comme appartenant à co-NP.
- Un exemple possible est donné par l'ISOMORPHISME DE GRAPHE

- Rappelons que deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ sont dits isomorphes s'il y a une bijection $f: V_G \rightarrow V_H$ telle que $\{f(v), f(w)\} \in E_H \Leftrightarrow \{v, w\} \in E_G$.
- Considérez le problème de décision suivant.

ISOMORPHISME DE GRAPHE
Entrée: deux graphiques G et H.
Question: G et H sont-ils isomorphes?

Cela appartient clairement à NP car un certificat évident est un isomorphisme, mais il n'est pas connu pour être NP-complet.

Il est également difficile de voir comment il pourrait appartenir à co-NP puisque le seul moyen évident que deux graphes ne sont pas isomorphes est de parcourir toutes les bijections possibles entre les ensembles de sommets et de vérifier qu'aucune de celles-ci ne sont des isomorphismes.

- Si P ≠ NP, alors le résultat suivant dû à Ladner (1975) nous indique qu'il doit exister des langages dans NP qui n'appartiennent ni à P ni ne sont NP-complets (encore une fois l'ISOMORPHISME DE GRAPHE est un langage candidat pour cette classe).
- Théorème: Si P ≠ NP alors il existe un langage dans NP \ P qui n'est pas NP-complet.
- Une approche à la question de savoir si P est égal à NP est la conjecture dite d'isomorphisme de Berman et Hartmanis (1977) qui, si elle était prouvée, impliquerait que P ≠ NP.

- Deux langages sur des alphabets de bande éventuellement différents,

$A \subseteq \sum_0^*$ et $B \subseteq \prod_0^*$, sont p-isomorphes s'il existe une fonction f telle que :

- (i) f est une bijection entre $\sum_0^*$ et $\prod_0^*$;
- (ii) $x \in A \Leftrightarrow f(x) \in B$;
- (iii) f et f^{-1} appartiennent tous deux à FP.

- Conjecture: Tous les langages NP-complets sont p-isomorphes.

- Théorème: Si la conjecture du p-isomorphisme est vraie alors $P \neq NP$.
- Preuve : Si P = NP alors tous les langages de P sont NP-complets, mais il y a des langages finis dans P et ceux-ci ne peuvent pas être p-isomorphes à des langages infinis.

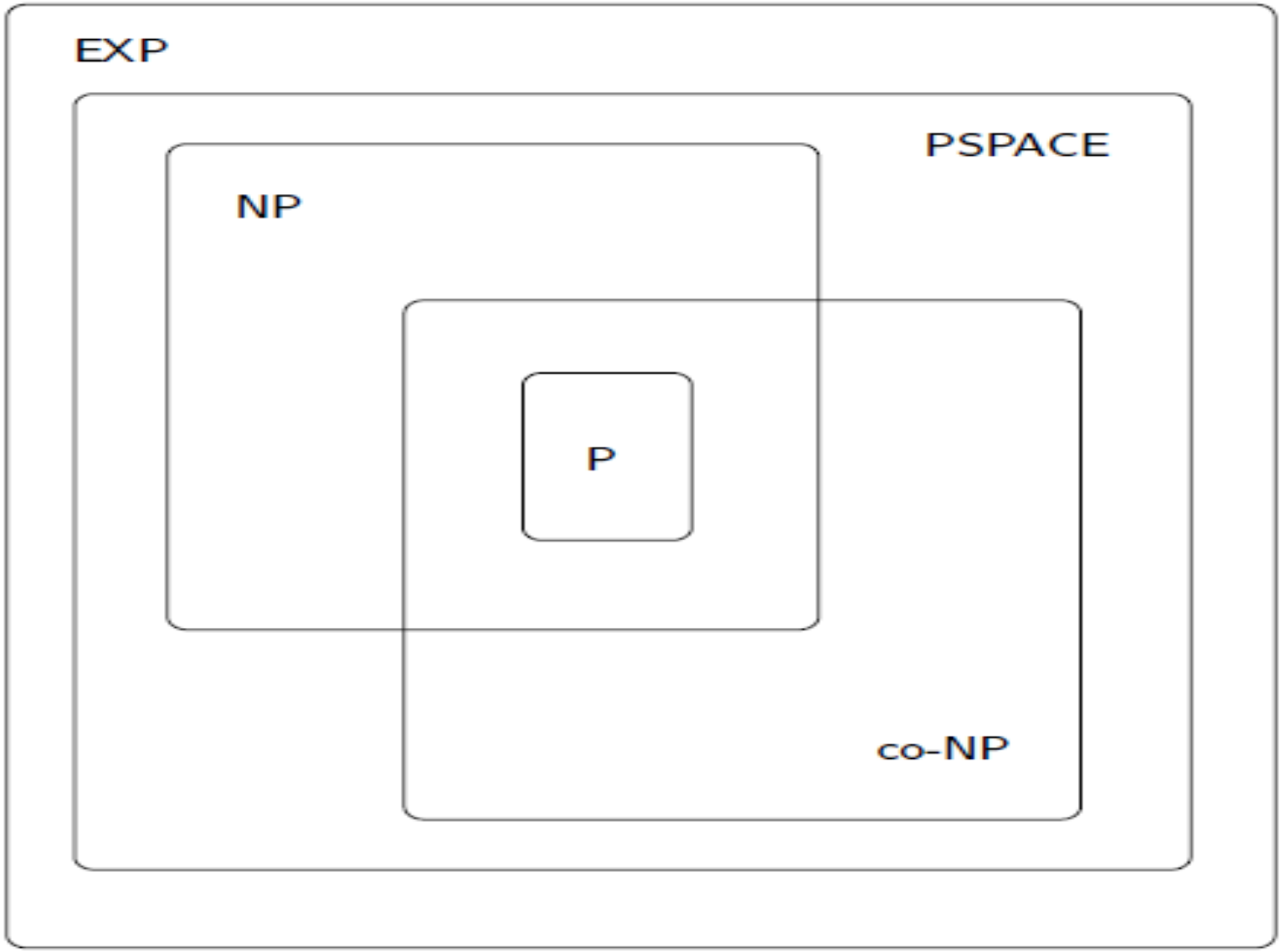

Figure. Conteneurs entre classes de complexité.

3.7 Machines de Turing non déterministes

- Définition : Une machine de Turing non déterministe ou NTM est définie de manière similaire à une DTM accepteur avec une différence importante. Au lieu d'une fonction de transition, il a une relation de transition, de sorte qu'à tout moment dans un calcul, il y a un certain nombre d'actions possibles qu'il peut entreprendre et il en choisit une de manière non déterministe.

Rappelons que la fonction de transition d'une DTM est une fonction à valeur unique

$$\delta : \Gamma \times \Sigma \rightarrow \Gamma \times \Sigma \times \{\leftarrow, \rightarrow\}.$$

- Compte tenu du contenu du carré de bande en cours de balayage, ainsi que de l'état actuel de la machine, un NTM a le choix d'actions possibles, dont l'une est choisie de manière non déterministe. Plus précisément si N est un NTM ; la machine est actuellement dans l'état γ_c et le contenu du carré courant balayé est σ_c, puis à l'étape suivante N choisit une action possible de manière non déterministe de l'ensemble

$$\Delta(\gamma_c, \sigma_c) = \{(\gamma_n, \sigma_n, m_n) \mid ((\gamma_c, \sigma_c), (\gamma_n, \sigma_n, m_n)) \in \Delta\}.$$

Ceci détermine ce qu'il faut écrire dans le carré courant ; le nouvel état pour N et le mouvement de la tête de lecture-écriture.

- Étant donné $x \in \sum_0^*$, un calcul sur l'entrée x est le résultat du démarrage de la machine avec x écrit sur la bande d'entrée, puis de l'application répétée de la relation de transition, en s'arrêtant si un état d'arrêt est atteint. (Notez que pour toute entrée x donnée, il y aura généralement plus d'un calcul possible.)

On dit qu'une entrée $x \in \sum_0^*$ est acceptée par un NTM s'il y a un calcul sur l'entrée x qui s'arrête dans l'état γ_T. Un tel calcul est appelé un calcul acceptant.

On dit qu'un NTM s'arrête si pour chaque entrée $x \in \sum_0^*$ et chaque calcul possible sur l'entrée x la machine s'arrête après un nombre fini de pas. Désormais, nous ne considérerons que les NTM qui s'arrêtent (s'arrête dans l'état γ_T).

- Pour un NTM, M, on définit le langage accepté par M comme suit :

$$L(M) = \{x \in \Sigma_0^* \mid x \text{ is accepted by } M\}.$$

Une étape d'un calcul est le résultat de l'application de la relation de transition une fois.

Pour x ∈ L (M), nous définissons le temps nécessaire pour accepter x comme étant le nombre d'étapes dans le calcul d'acceptation le plus court

$$t_M(x) = \min\{t \mid \text{there is an accepting computation of } M \text{ on input } x \text{ that halts in } t \text{ steps}\}.$$

La complexité temporelle de M est alors définie comme suit

$$T_M(n) = \max\{t \mid \exists x \in L(M) \text{ such that } |x| = n \text{ and } t_M(x) = t\}.$$

- L'ensemble des calculs possibles d'un NTM sur une entrée particulière peut facilement être représenté par un arbre. Un seul calcul possible est un chemin de la racine à une feuille. En supposant que la machine arrête chaque calcul possible est fini et donc l'arbre est également fini. Dans ce cas, le temps nécessaire pour accepter une entrée x est simplement la longueur du chemin le plus court de l'arbre qui se termine par l'état γ_T.

Il est intuitivement évident qu'un langage L est accepté par une NTM à temps polynomial si et seulement s'il appartient à NP.

L'idée clé est de considérer l'arbre de calcul d'un NTM à temps polynomial. À n'importe quel nœud de l'arbre, il y a un nombre fini de choix pour la transition vers l'étape suivante.

- Ainsi une chaîne de certificat possible $y \in \sum_0^*$ pour une entrée $x \in \sum_0^*$ est une liste de choix de branches nous indiquant quelle branche de l'arbre de calcul suivre à chaque étape du calcul.
- Si $x \in L$ alors il y a un chemin de longueur polynomiale dans l'arbre menant à l'état γ_T et ce chemin peut être décrit par une chaîne de longueur polynomiale y.
- Alors que si $x \notin L$ alors aucun chemin ne mène à l'état d'acceptation et donc aucune chaîne y ne peut décrire un chemin d'acceptation dans l'arbre.

Théorème: La classe de langages acceptée par les NTM à temps polynomial est égale à NP.