Tiende college complexiteit

16 april 2019

NP-volledigheid II

Zes zeer bekende NP-volledige problemen:

- CNF-satisfiability (SAT)
  Gegeven een logische formule ϕ in CNF. Bestaat er een waardering van de in ϕ voorkomende logische variabelen die ϕ waar maakt?
- Subset Sum (SUM)
  Gegeven een getal $t \in \mathbb{N}$ en een eindige verzameling $S \subset \mathbb{N}$. Bestaat er een deelverzameling $S' \subseteq S$ met $\Sigma_{s \in S'} s = t$?
- Hamiltonkring (HC)
  Gegeven een graaf $\mathcal{G} = (V, E)$. Heeft $\mathcal{G}$ een Hamiltonkring?
- Handelsreizigersprobleem (TSP)
  Gegeven een volledige, ongerichte graaf $\mathcal{G} = (V, E)$ met gewichten op de takken, en een geheel getal $k \geq 0$. Bestaat er in $\mathcal{G}$ een Hamiltonkring met totaalgewicht $\leq k$?
- Kliek
  Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal k $(0 \leq k \leq |V|)$. Bestaat er in $\mathcal{G}$ een kliek ter grootte $\geq k$ (equivalent met $= k$)?
- Graafkleuring (Kleur)
  Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal $k > 0$. Bestaat er een kleuring van $\mathcal{G}$ met $\leq k$ kleuren (equivalent met $= k$)?

1. Voor alle zes voorbeeldproblemen is eenvoudig een exponentieel algoritme op te schrijven.

2. In alle gevallen kan in polynomiale tijd gecontroleerd worden of een kandidaatoplossing een echte oplossing (= "dat wat je zoekt") is.

3. Voor al deze problemen geldt: het lijkt extreem moeilijk (exponentieel) om voor gegeven invoer x te bepalen of het antwoord "ja" of "nee" moet zijn.

4. Echter, *als* x een ja-instantie is, dan is er een eenvoudige (polynomiale) manier om iemand daarvan te overtuigen.

5. Voor ja-instanties bestaat er een zogenaamd certificaat (in de voorbeelden de gezochte oplossing), dat gebruikt kan worden om te laten zien dat het antwoord inderdaad "ja" is.

6. Bovendien is dit certificaat kort (polynomiaal) en kan het verifiëren/controleren ervan in polynomiale tijd.

7. Eigenschap 2 t/m 6: de genoemde problemen zitten in $\mathcal{NP}$. Straks wordt alles wat formeler gemaakt $\longrightarrow$ niet-determinisme.

8. Ja-instanties zijn eenvoudig te verifiëren met de juiste hint (certificaat). Hoe zit het met nee-instanties?

9. Er zijn ook makkelijke instanties / ingeperkte versies van het probleem.

$\mathcal{P}$: beslissingsproblemen die polynomiaal oplosbaar zijn.

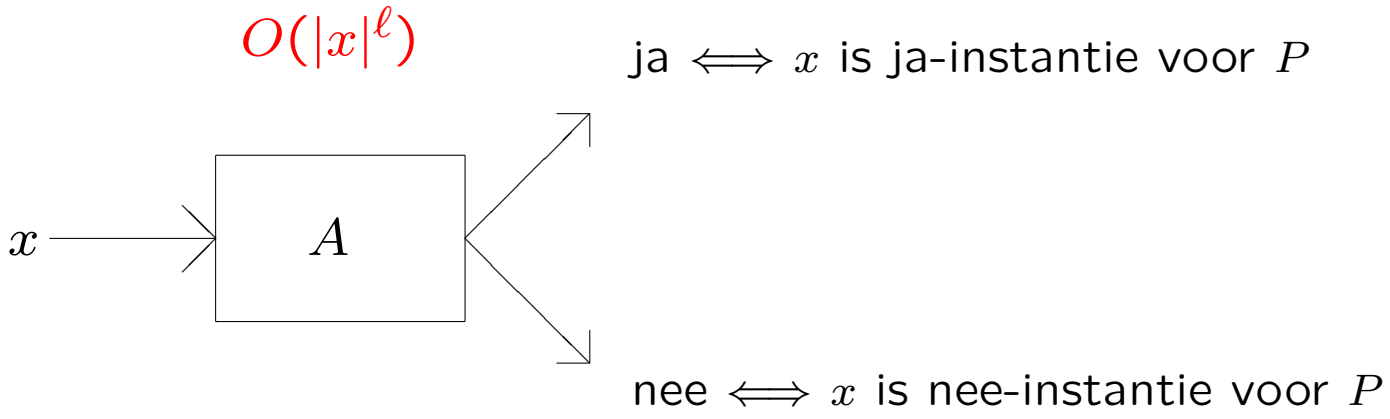
$\mathcal{NP}$: beslissingsproblemen die polynomiaal verifieerbaar zijn,

- althans, de ja-instanties
- en met de juiste hint/certificaat
- voor ja-instanties bestaat er altijd zo'n certificaat

$\mathcal{NPC}$: bevat de moeilijkste problemen uit $\mathcal{NP}$.

Voor problemen in $\mathcal{NP}$ duurt het *vinden* van een oplossing lang (tenzij ze in $\mathcal{P}$ zitten); het *verifiëren/controleren* kan echter in polynomiale tijd.

Gegeven een willekeurig beslissingsprobleem $P \in \mathcal{P}$. Dan is er een polynomiaal deterministisch algoritme A dat P oplost voor elke invoer x.



Het algoritme is polynomiaal in de lengte van de invoer, dus worst case is $O(|x|^\ell)$ $(\ell \geq 0)$. Een algoritme is deterministisch als het elke keer dat het wordt uitgevoerd op dezelfde invoer hetzelfde doet, en dus dezelfde uitvoer oplevert.

NP-problemen: de *ja-instanties* zijn eenvoudig (polynomiaal) te verifiëren met behulp van het juiste certificaat, de nee-instanties niet.

Veel beslissingsproblemen zijn geformuleerd als "is-er-een"-vragen. In dat geval kun je bij een certificaat denken aan een oplossing voor het probleem, dus datgene wat gezocht wordt en wat een ja-instantie een ja-instantie maakt.

Daarvan moet dan onder andere geverifieerd/gecontroleerd worden dat deze aan de criteria van het probleem voldoet, dus inderdaad een *echte* oplossing is, en derhalve een ja-antwoord rechtvaardigt.

Als deze verificatie/controle in polynomiale tijd kan, zit het probleem in $\mathcal{NP}$.

Gegeven een logische formule ϕ in CNF, bestaat er een waardering die ϕ waarmaakt?

In geval van een ja-instantie kunnen we als certificaat een waarmakende waardering nemen.
Er kan nu in polynomiale tijd geverifieerd worden dat deze waardering ϕ inderdaad waarmaakt, en dus dat ϕ een ja-instantie is.
Probleem SAT zit derhalve in $\mathcal{NP}$ (en overigens ook in $\mathcal{NPC}$).

Ook voor de andere vijf voorbeeldproblemen geldt: als x een ja-instantie voor het probleem is, dan is bestaat er een certificaat waarmee je eenvoudig (polynomiaal) kunt aantonen dat x inderdaad een ja-instantie is.

Probleem	Invoer	Certificaat
HC	$\langle \mathcal{G} \rangle$	Hamiltonkring
SAT	$\langle \phi \rangle$	waarmakende waardering
Kliek	$\langle \mathcal{G}, k \rangle$	kliek met ($\geq$) k knopen
Kleur	$\langle \mathcal{G}, k \rangle$	kleuring met ($\leq$) k kleuren
Sum	$\langle S, t \rangle$	deelverzameling met som $= t$
TSP	$\langle \mathcal{G}, k \rangle$	Hamiltonkring met totaalgewicht $\leq k$

1. Invoer: een array A met $n > 0$ gehele getallen
   Algoritme:

   for $i := 1$ **to** n **do**
   print $A[i]$;
   od

   Complexiteit: $\Theta(n)$

2. Invoer: een geheel getal $n > 0$
   Algoritme:

   for $i := 1$ **to** n **do**
   print '1';
   od

   Complexiteit: $\Theta(n) = \Theta(2^{\lg(n)})$

Vraag: wat is eigenlijk de lengte van de invoer?

Voorbeeldprobleem
Gegeven een geheel getal $n > 1$. Heeft n echte delers, met andere woorden, is $n = a * b$ voor zekere $a, b > 1$?

Algoritme:

```
// gewoon alle mogelijke delers proberen
    gevonden := False;
    m := 2;
    while not gevonden and m < n do
        if n % m = 0 then
            gevonden := True;
        else
            m := m + 1;
        fi
    od
    // m is nu de kleinste deler > 1 van n
```

De worst case complexiteit van dit algoritme is $\Theta(n)$ (indien we de berekening van n % m voor 1 stap tellen, anders $O(n^2)$).

De lengte van de invoer is het aantal karakters van de codering, in dit geval van het getal n. Als we de unaire codering gebruiken is het algoritme dus lineair in de lengte van de invoer. Nemen we de binaire codering, of in het algemeen de ℓ-aire codering met $\ell > 1$, dan is het algoritme exponentieel in de lengte van de invoer (voor elke $\ell > 1$ dus).

Vraag: is het algoritme polynomiaal of exponentieel?

In 2002 is door Manindra Agrawal en zijn PhD-studenten Neeraj Kayal en Nitin Saxena bewezen dat het probleem in $\mathcal{P}$ zit. De zogenaamde AKS-test is een priemgetaltest die polynomiaal is in het aantal cijfers (dus de lengte) van n.

In 2006 ontvingen zij hiervoor de Fulkerson Prize (discrete wiskunde) en de Gödel Prize (theoretische informatica).

Knapzakprobleem

Gegeven een knapzak met capaciteit S (een geheel getal > 0) en n objecten met gewicht $s_1, s_2, \ldots, s_n$ en met waarde $w_1, w_2, \ldots, w_n$. (Alle s_i en w_i zijn geheel en > 0.)
Gevraagd een deelverzameling van de objecten met totaalgewicht $\leq S$ en maximale totaalwaarde.

Het knapzakprobleem kan worden opgelost met een algoritme met complexiteit $O(n \cdot S)$ (dynamisch programmeren, zie Algoritmiek).

Dit is niet polynomiaal, maar exponentieel als functie van de lengte van de invoer!

De klassen $\mathcal{P}$ en $\mathcal{NP}$ worden formeel gedefinieerd met behulp van Turing machines. Een Turing machine is een uiterst simpel, maar krachtig model van een 'computer'.

Elk probleem dat met een 'normaal' computerprogramma in polynomiale tijd op te lossen is, blijkt ook polynomiaal oplosbaar te zijn met behulp van een Turing machine, en omgekeerd.

We mogen dus gewoon C++-achtige algoritmen blijven gebruiken, maar het is nuttig om iets weten over de werking van Turing machines.

$\mathcal{NP}$ is de klasse van beslissingsproblemen waarvoor een niet-deterministisch polynomiaal algoritme bestaat:

- niet-deterministisch:
  je mag een oplossing gokken $\sim$ certificaat

- polynomiaal:
  deze kan daarna in polynomiale tijd geverifieerd (gecontroleerd) worden

- ja-instantie:
  voor een ja-instantie bestaat er een oplossing, en dus een certificaat waarmee je aantoont dat het inderdaad een ja-instantie is

We kunnen een niet-deterministisch algoritme omschrijven als bestaande uit een gok-fase, verificatie-fase, en output-fase. We gebruiken een dergelijke omschrijving in de (precieze) definitie van $\mathcal{NP}$.

Zo'n niet-deterministisch algoritme bestaat uit 3 fasen:

1. Niet-deterministische gokfase
   Er wordt een willekeurige string s in het geheugen geschreven. Elke keer dat het algoritme executeert kan dit een andere string zijn.

   // deze string is vaak een soort gok van de oplossing van
   // het probleem: het beoogde certificaat;
   // s kan echter ook een onzinstring blijken te zijn.

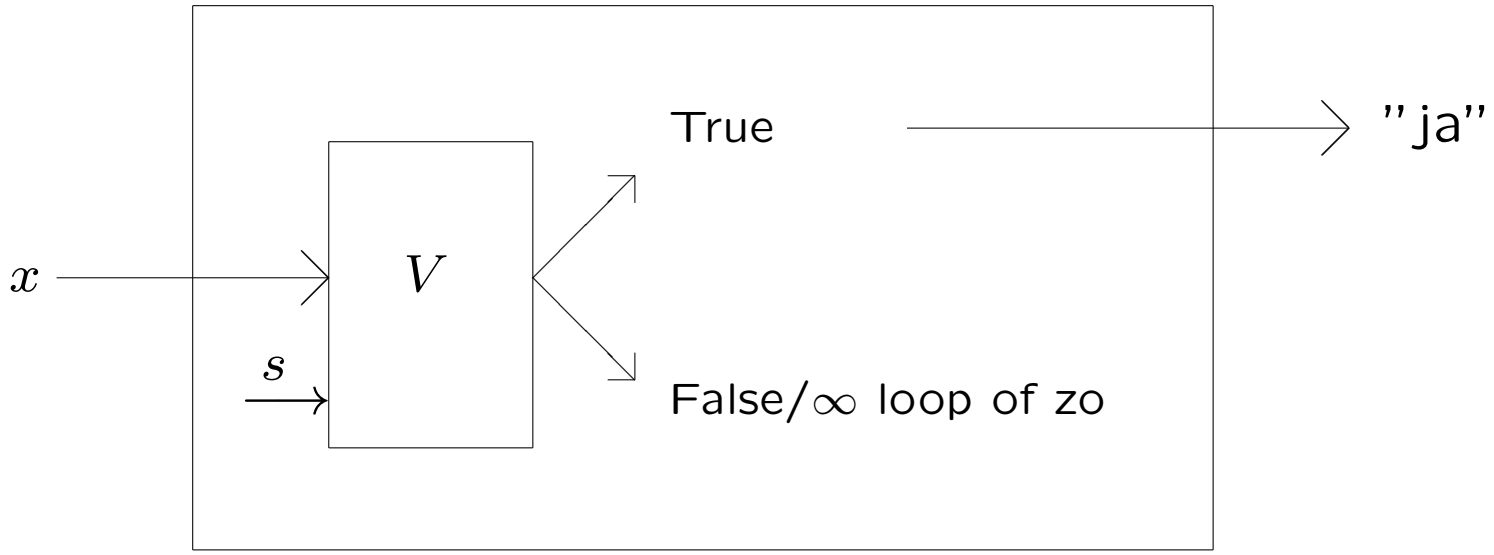
2. Deterministische verificatiefase
   Zowel de invoer x van het probleem als de string s mogen hier gebruikt worden. Er wordt True of False geretourneerd, of het programma stopt nooit (het kan bijvoorbeeld in een oneindige loop raken).

   // hier wordt gecontroleerd of s een oplossing van het
   // probleem is bij de gegeven invoer x, m.a.w. er wordt
   // gecontroleerd of s een ja-antwoord rechtvaardigt.

3. Uitvoerstap
   Als fase 2 True retourneert geeft het algoritme antwoord "ja". Anders is er geen uitvoer.

Het aantal stappen dat een niet-deterministisch algoritme doet is het aantal stappen nodig om s te schrijven (dus het aantal karakters waaruit s bestaat) + het aantal stappen dat gedaan wordt in de verificatiefase (+1 voor fase 3).

Schematisch: een executie



fase 1: s wordt gegenereerd; (niet-deterministisch)

fase 2: verificatiefase (deterministisch)

fase 3: uitvoerstap

Bij een niet-deterministisch algoritme zijn verschillende executies mogelijk voor dezelfde invoer x, afhankelijk van de gegokte s. Dus:

(a) Wat is *het* antwoord (ja/nee) van het algoritme voor invoer x?

(b) Wanneer noemen we zo'n algoritme polynomiaal?

Definitie

Het antwoord van een niet-deterministisch algoritme A voor invoer x is "ja" $\iff$ er is een executie* van A die "ja" als uitvoer geeft $\iff$ er is een string s waarvoor fase 2 True oplevert.
Het antwoord van A is "nee" als er voor geen enkele executie, dus voor geen enkele string s, een uitvoer is.

Er geldt dus: het antwoord van zo'n niet-deterministisch algoritme A voor invoer x is "ja"† $\iff$ er bestaat een string s (certificaat) waarmee je kunt aantonen (gebeurt in fase 2) dat x een ja-instantie is.

*dus een string s

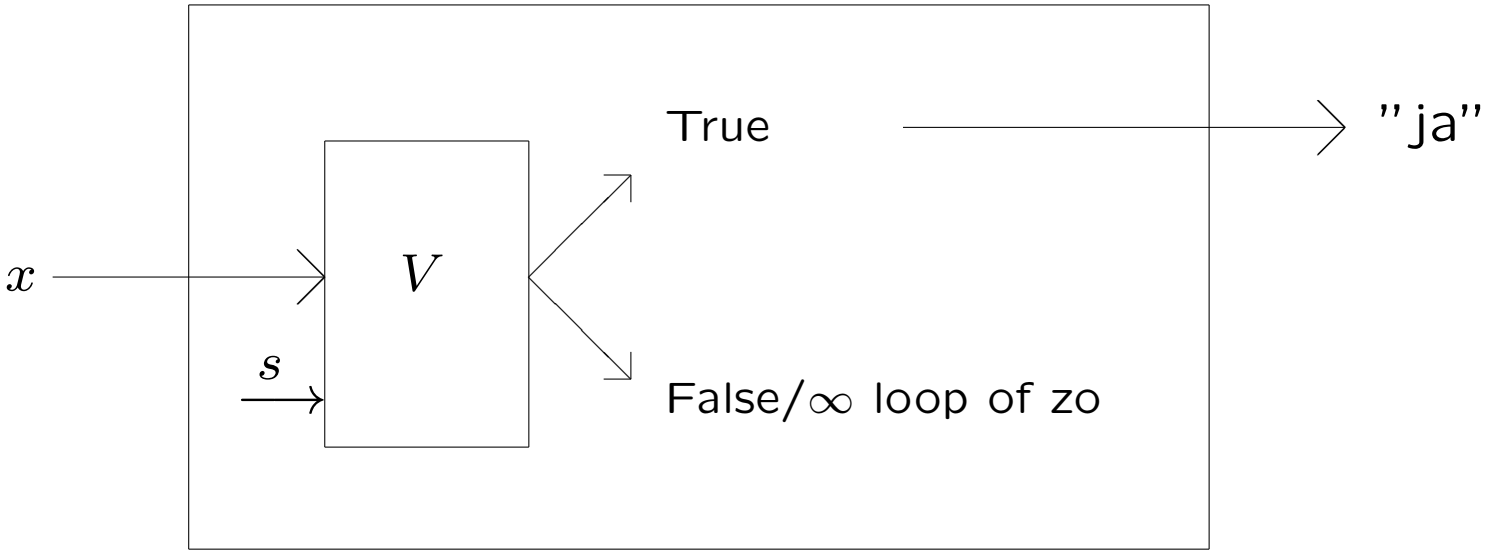
†ofwel: x is een ja-instantie

- Een niet-deterministisch algoritme heet polynomiaal begrensd als voor elke invoer x waarvoor het antwoord "ja" is, er een executie van het algoritme is die "ja" oplevert in hooguit $O(|x|^\ell)$ stappen (voor zekere $\ell \geq 0$).
  Dientengevolge mag in dat geval de string s niet te lang zijn (polynomiaal in $|x|$), en het verificatie-algoritme uit fase 2 moet polynomiaal begrensd zijn in $|x|$ (en $|s|$).

- $\mathcal{NP}$ is nu de klasse van beslissingsproblemen waarvoor er een polynomiaal begrensd niet-deterministisch algoritme bestaat.

- $\mathcal{NP}$ betekent: $\mathcal{N}$on-deterministic $\mathcal{P}$olynomial time.

Niet-deterministisch + polynomiaal:



fase 1: $O(|s|) \subseteq O(|x|^{\ell})$ voor zekere

fase 2: $O(|s|^{p} \cdot |x|^{q}) \subseteq O(|x|^{r})$ ja-executie

fase 3: $O(1)$

Stelling. Alle zes voorbeeldproblemen zitten in $\mathcal{NP}$.

Voorbeeld 1: Kleur
Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal $k > 0$. Bestaat er een kleuring van $\mathcal{G}$ die hooguit k kleuren gebruikt (ofwel, is $\mathcal{G}$ k-kleurbaar)?

Voorbeeld 2: HC
Gegeven een (gerichte of ongerichte) graaf $\mathcal{G} = (V, E)$. Vraag: heeft deze graaf een Hamiltonkring?
Laat hierna $V = \{1, 2, \ldots, n\}$, en dus $|V| = n$.

Voorbeeld 3: SAT
Gegeven een logische formule ϕ in CNF. Bestaat er een waardering van de in ϕ voorkomende logische variabelen die ϕ waar maakt?

Laat $V = \{1, 2, \ldots, n\}$, en de mogelijke kleuren $1, 2, \ldots, k$.
Een polynomiaal begrensd *niet-deterministisch algoritme* A voor Kleur, met als invoer $x = \langle \mathcal{G}, k \rangle$, is:

1. Fase 1 (gokfase)
   Er wordt een string s gegenereerd, hierna te interpreteren als een rij gehele getallen.

2. Fase 2 (verificatiefase)
   Er wordt gecontroleerd of s een goede kleuring voorstelt (s_i wordt geïnterpreteerd als de kleur van knoop i):

   (1) controleer of er precies $n = |V|$ integers staan (elke knoop een kleur): kan polynomiaal, $O(|s|)$

   (2) controleer of elke integer tussen 1 en k is (er worden k kleuren gebruikt): kan polynomiaal, $O(|s|)$

   (3) controleer of aangrenzende knopen verschillend gekleurd zijn. Takken (v, w) aflopen en in s de kleur van v en w opzoeken en vergelijken: $O(|E| \cdot |s|) \subseteq O(|\mathcal{G}| \cdot |s|) \subseteq O(|x| \cdot |s|)$.

Als de drie tests positief zijn wordt True geretourneerd, zodra een test negatief uitvalt wordt False teruggegeven (of er wordt in een oneindige loop gegaan).

3. Fase 3 (uitvoerfase)
   Als fase 2 True oplevert wordt "ja" uitgevoerd, anders geen uitvoer.

Merk op. Tests (1) en (2) controleren of s een kleuring van de knopen voorstelt, test (3) controleert of het een correcte kleuring is.

Er geldt: het antwoord van A op invoer $x = \langle \mathcal{G}, k \rangle$ is "ja" $\iff$ er bestaat een goede string s waarop fase 2 True oplevert $\iff$ er bestaat een goede kleuring van de knopen van $\mathcal{G}$ $\iff$ $x = \langle \mathcal{G}, k \rangle$ is een ja-instantie voor Kleur.

Verder: voor een ja-executie, dus met s een goede kleuring, is $|s| \in O(|V|) \subseteq O(|x|)$. Ergo: A is polynomiaal begrensd.

Een polynomiaal begrensd *niet-deterministisch algoritme* voor HC (invoer $\mathcal{G}$, $n = |V|$, $V = \{1, 2, \ldots, n\}$) (zie ook dictaat):

1. Fase 1 (gokfase)
   Er wordt een string s gegenereerd, hierna te interpreteren als een rij gehele getallen.

2. Fase 2 (verificatiefase)
   Er wordt gecheckt of s een Hamiltonkring voorstelt:

   (1) controleer dat er precies n integers staan: $O(|s|)$

   (2) controleer dat elke integer tussen 1 en n is: $O(|s|)$

   (3) controleer dat alle knopen uit s verschillen: $O(|s|^2)$

   (4) controleer dat tussen opeenvolgende knopen uit s een tak zit in de graaf (en tussen de laatste en de eerste): $O(|s| \cdot |E|) \subseteq O(|s| \cdot |\mathcal{G}|)$.

Als de vier tests positief zijn wordt True geretourneerd, anders False (of er wordt in een oneindige loop gegaan).

3. Fase 3 (uitvoerfase)
   Als fase 2 True oplevert wordt "ja" uitgevoerd, anders geen uitvoer.

Merk op:
Test (1) t/m (3) controleert dat s een rij van n verschillende knopen van $\mathcal{G}$ voorstelt (soort syntactische controle), test (4) controleert dat het een Hamiltonkring is.
Voor ja-instanties bestaat er een string s die een Hamiltonkring voorstelt, en dus een executie die "ja" oplevert, waarbij $|s| \in O(|\mathcal{G}|)$. Zo'n ja-executie is dus polynomiaal in $|\mathcal{G}|$. Voor nee-instanties bestaat zo'n string s niet.

SAT: Gegeven een logische formule ϕ in CNF. Bestaat er een waardering van de in ϕ voorkomende logische variabelen zodat ϕ de waarde True krijgt (dus een waardering die ϕ waar maakt)?

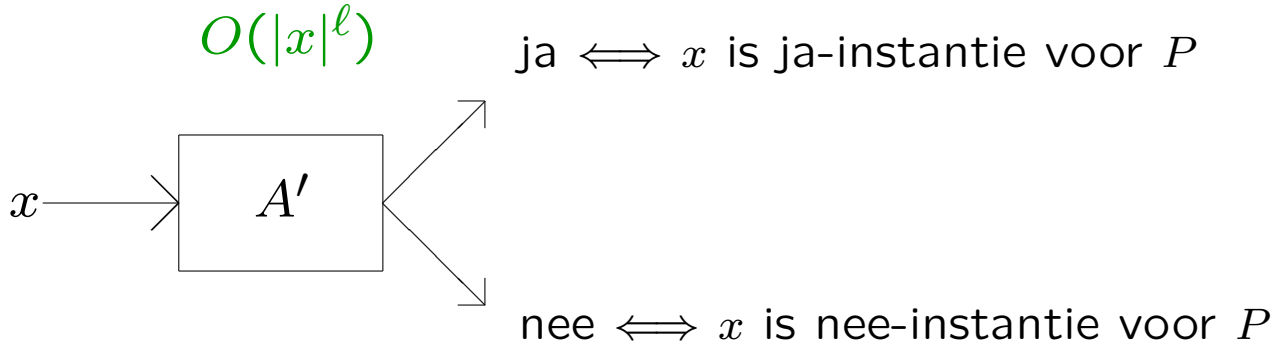
Een logische formule ϕ staat in CNF (conjunctieve normaalvorm) als hij bestaat uit een conjunctie ($\wedge$) van clausules (= disjuncties).

Voorbeeld: $\phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_1$.

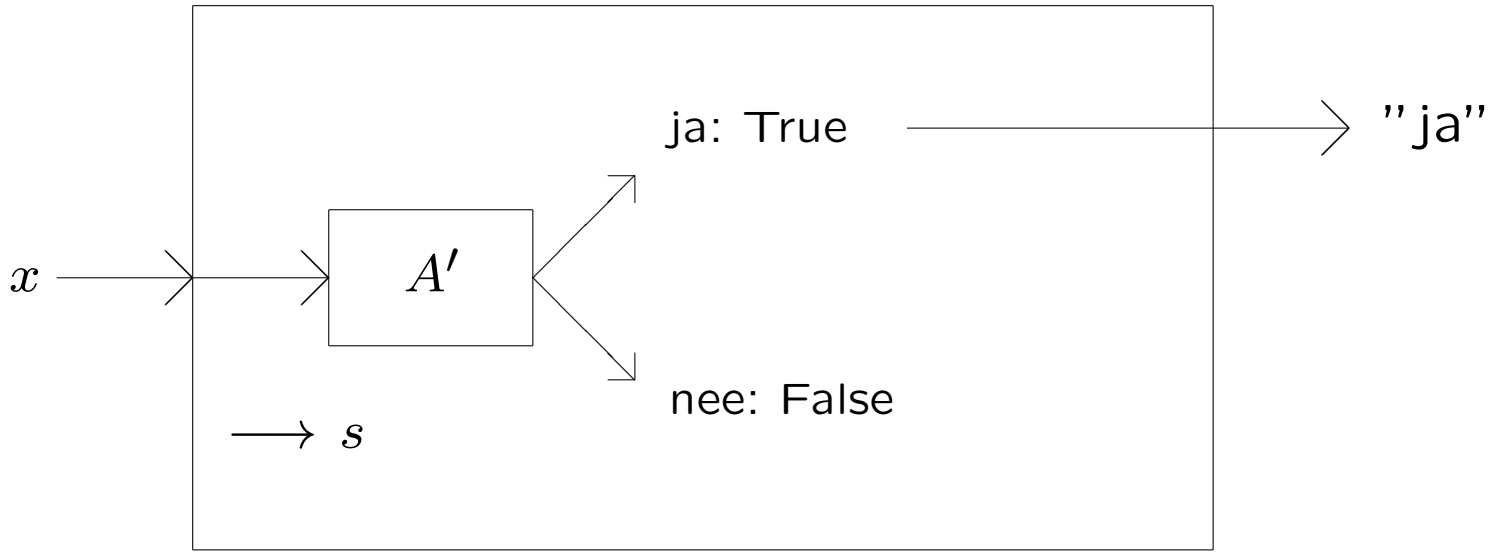
Opgave
Geef een niet-deterministisch algoritme voor SAT.

Stelling: $\mathcal{P} \subseteq \mathcal{NP}$

Bewijs:
Neem een willekeurig probleem $P \in \mathcal{P}$. Dan is er een polynomiaal deterministisch algoritme A' dat P oplost.



Het volgende niet-deterministische algoritme A is dan een polynomiaal begrensd algoritme voor P.



fase 1: s wordt gegenereerd; $O(|s|)$

fase 2: negeer de string s en voer A' uit op x; $O(|x|^{\ell})$

fase 3: $O(1)$

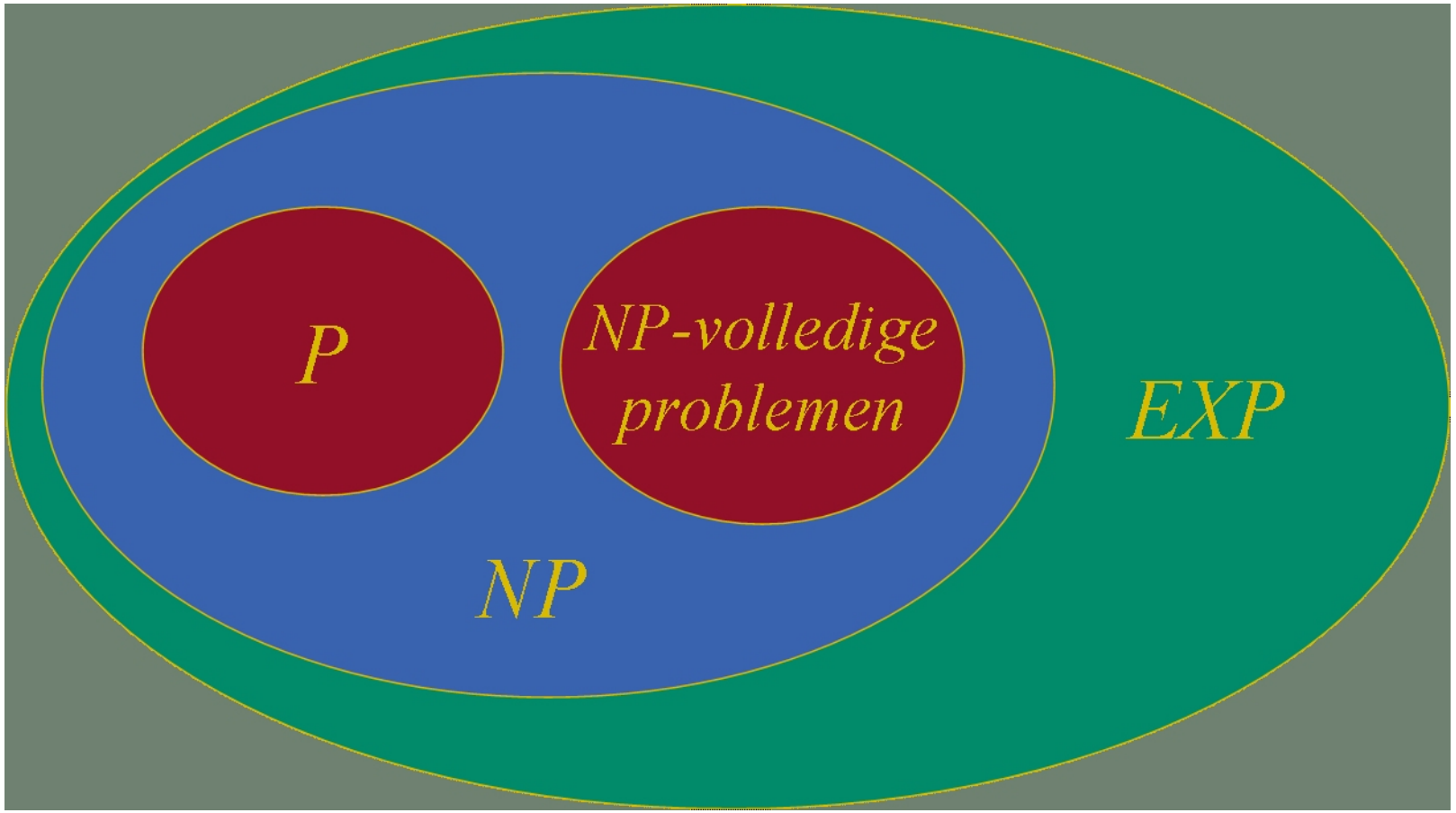
De klasse van NP-volledige problemen $\mathcal{NPC}$ (Engels: NP-complete) bevat de *moeilijkste* problemen in $\mathcal{NP}$ en heeft de volgende interessante eigenschap:

1. Als er een polynomiaal algoritme bestaat voor willekeurig welk NP-volledig probleem, dan is meteen elk NP-volledig* probleem in polynomiale tijd oplosbaar.

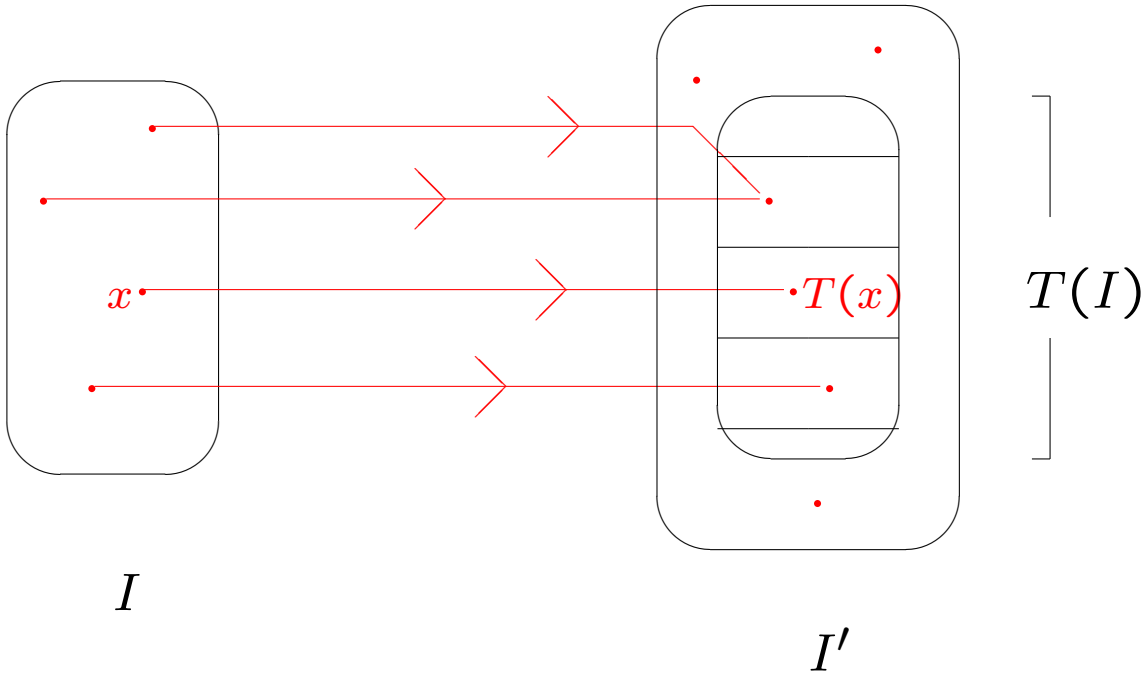
2. Omgekeerd: als er van één enkel NP-volledig probleem bewezen wordt dat het onhandelbaar is, dan zijn alle NP-volledige problemen onhandelbaar.

*zelfs: dan is elk probleem uit $\mathcal{NP}$ in polynomiale tijd oplosbaar, en dus $\mathcal{P} = \mathcal{NP}$

Meest waarschijnlijke relatie



Zij T een functie van de invoerverzameling I van een beslissingsprobleem P naar de invoerverzameling I' van een beslissingsprobleem Q.
T beeldt dus elke $x \in I$ af op een $T(x) \in I'$.



Definitie

T heet een polynomiale reductie (of *polynomiale transformatie*) van P naar Q als geldt:

1. T kan berekend worden in polynomiaal begrensde tijd (als functie van $|x|$). D.w.z.: de constructie van $T(x)$ uit x kan in $O(|x|^k)$ stappen in de worst case ($k \geq 0$).

2. Voor elke x uit I geldt: als x een ja-instantie is voor P dan is $T(x)$ een ja-instantie voor Q.

3. Voor elke x uit I geldt: als x een nee-instantie is voor P dan is $T(x)$ een nee-instantie voor Q.

3′. Voor elke x uit I geldt: als $T(x)$ een ja-instantie is voor Q dan is x een ja-instantie voor P.
(Dit is equivalent met 3.)

Definitie

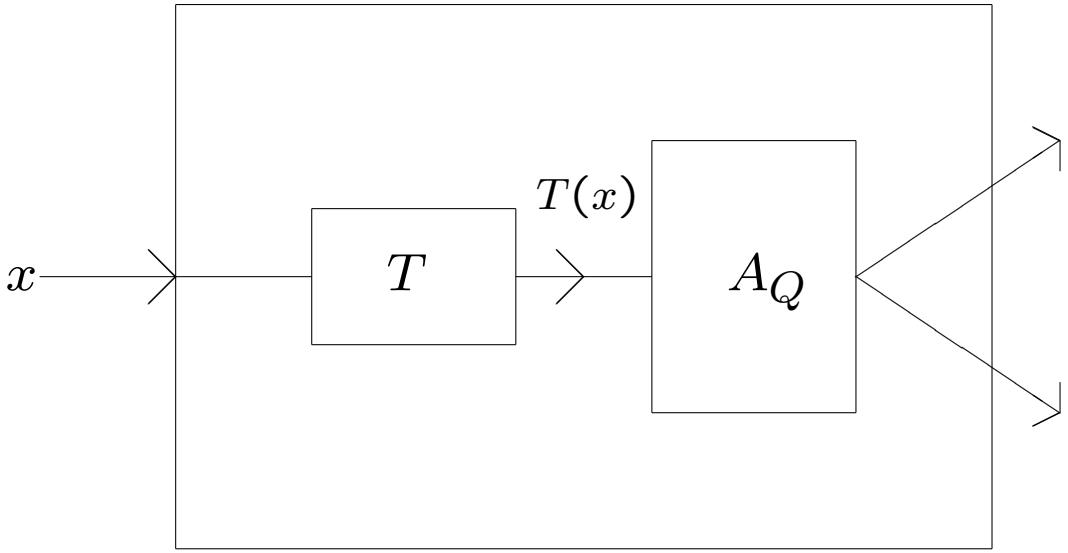
Een probleem P is polynomiaal reduceerbaar (of polynomiaal transformeerbaar) naar Q als er een polynomiale reductie bestaat van P naar Q.

Notatie: $P \leq_{\mathsf{P}} Q$.

Stelling

Als $P \leq_{\mathsf{P}} Q$ en Q zit in $\mathcal{P}$, dan zit P ook in $\mathcal{P}$.

Laat A_Q een polynomiaal deterministisch algoritme zijn voor Q.
Een polynomiaal deterministisch algoritme voor P is dan:



"ja" $\Longleftrightarrow$ $y = T(x)$ is ja-instantie van Q $\Longleftrightarrow$ x is ja-instantie van P

"nee" $\Longleftrightarrow$ $T(x)$ is nee-instantie van Q $\Longleftrightarrow$ x is nee-instantie van P

$O(|x|^k)$ $O(|y|^\ell)$

$O(|x|^{k\ell})$

reductie-eigenschap

$P \leq_{\mathsf{P}} Q$ betekent dat er een polynomiale reductie T van P naar Q bestaat:

1. T beeldt elke invoer x van beslissingsprobleem P af op een invoer $T(x)$ van beslissingsprobleem Q.

2. De constructie van $T(x)$ uit x is polynomiaal ($O(|x|^k)$.

3. Reductie-eigenschap: voor elke x uit I (= invoerverzameling van P) geldt: x is een ja-instantie voor $P \iff T(x)$ is een ja-instantie voor Q.

Definitie
Een probleem Q is NP-hard (ook wel: NP-moeilijk) als elk probleem P in $\mathcal{NP}$ polynomiaal reduceerbaar is tot Q, dat wil dus zeggen dat $P \leq_{\mathsf{P}} Q$ voor alle $P \in \mathcal{NP}$.

Definitie
Een probleem Q is NP-volledig als

1. $Q \in \mathcal{NP}$

2. Q is NP-hard

Notatie
De klasse van NP-volledige problemen geven we aan met $\mathcal{NPC}$ (NP-complete).

Stelling

Als een of ander willekeurig NP-volledig probleem in $\mathcal{P}$ zit, dan is $\mathcal{P} = \mathcal{NP}$.

Dit betekent dus: als één enkel NP-volledig probleem P polynomiaal begrensd is, dan zijn alle problemen uit $\mathcal{NP}$ polynomiaal begrensd.
Omgekeerd: als *een* willekeurig probleem in $\mathcal{NP}$ zeker niet polynomiaal begrensd is, dan zijn *alle* NP-volledige problemen niet polynomiaal begrensd.

- Volgende college:
  dinsdag 23 april, 11.00 – 12.45, zaal 174
  Let op: geen college op dinsdag 30 april, wel werkcollege

- Eerstvolgende werkcollege:
  dinsdag 16 april, 13.30 – 15.15, zaal 174
  Opgaven 41, 42, 43, 44, 45

- Derde huiswerkopgave (van de vier):

  * deadline: dinsdag 23 april; LaTeX; print → college

  * www.liacs.leidenuniv.nl/~graafjmde/COMP/