Forelesning 13

Bonusmateriale

Ting som ikke ble med i forelesningen, men som kanskje kan være av interesse

1:9

NP-Komplettthet

- **NP**: Mengden av språk med verifikasjonsalgoritmer som har polynomisk kjøretid

- **NP**: Mengden av språk med verifikasjonsalgoritmer som har polynomisk kjøretid
- **Verifikasjonsalgoritme** $\mathrm{A}(x, y)$ for et språk L:

L består av bitstrenger, dvs., $\mathrm{L} \subseteq \{0, 1\}^*$

- **NP**: Mengden av språk med verifikasjonsalgoritmer som har polynomisk kjøretid
- **Verifikasjonsalgoritme** $\mathrm{A}(x, y)$ for et språk L:
  - Hvis $x \in \mathrm{L}$ så er $\mathrm{A}(x, y) = 1$ for en eller annen y

x og y bitstrenger, dvs., $x, y \in \{0, 1\}^*$

- **NP**: Mengden av språk med verifikasjonsalgoritmer som har polynomisk kjøretid
- **Verifikasjonsalgoritme** $\mathrm{A}(x, y)$ for et språk L:
  - Hvis $x \in \mathrm{L}$ så er $\mathrm{A}(x, y) = 1$ for en eller annen y
  - Ellers er $\mathrm{A}(x, y) = 1$ for alle y

- **NP**: Mengden av språk med verifikasjonsalgoritmer som har polynomisk kjøretid
- **Verifikasjonsalgoritme** $\mathrm{A}(x, y)$ for et språk L:
  - Hvis $x \in \mathrm{L}$ så er $\mathrm{A}(x, y) = 1$ for en eller annen y
  - Ellers er $\mathrm{A}(x, y) = 1$ for alle y

Og det er egentlig det; verre er det ikke!

Reduce-to-B

La oss si vi har en reduksjon fra A til B

$$\beta = \textsc{Reduce-to-B}(\alpha)$$

Den tar in en A-instans α og lager en B-instans β

$$\beta = \textsc{Reduce-to-B}(\alpha)$$
$$\textsc{Solve-B}(\beta)$$

Denne kan vi bruke i en (hypotetisk) løsning for B

$$\beta = \textsc{Reduce-to-B}(\alpha)$$
$$\textsc{Solve-B}(\beta)$$

Vi krever at $\mathrm{B}(\beta) = \mathrm{A}(\alpha)$, så . . .

$\textsc{Solve-A}(\alpha)$
1 $\beta = \textsc{Reduce-to-B}(\alpha)$
2 **return** $\textsc{Solve-B}(\beta)$

Vi har nå en (hypotetisk) løsning for A!

$\Omega(2^n)$ $\textsc{Solve-A}(\alpha)$

1 $\beta = \textsc{Reduce-to-B}(\alpha)$

2 **return** $\textsc{Solve-B}(\beta)$

La oss si vi vet at A umulig kan løses bedre …

$\Omega(2^n)$ $\quad$ SOLVE-A(α)

$O(n^2)$ $\quad$ 1 $\quad \beta =$ REDUCE-TO-B(α)

2 $\quad$ **return** SOLVE-B(β)

…men at reduksjonen vår er bedre

$\Omega(2^n)$	SOLVE-A(α)
$O(n^2)$	1 β = REDUCE-TO-B(α)
	2 **return** SOLVE-B(β)

Hva kan vi si om SOLVE-B sin kjøretid?

$$
\begin{array}{ll}
\Omega(2^n) & \textsc{Solve-A}(\alpha) \\
\mathrm{O}(n^2) & 1 \quad \beta = \textsc{Reduce-to-B}(\alpha) \\
\Omega(2^n) & 2 \quad \textbf{return } \textsc{Solve-B}(\beta)
\end{array}
$$

Den må være eksponentiell for å «oppfylle» kjøretiden til A

$\mathrm{T_A}(n)$	$\textsc{Solve-A}(\alpha)$
$\mathrm{T_R}(n)$	1 $\beta = \textsc{Reduce-to-B}(\alpha)$
$\mathrm{T_B}(n)$	2 **return** $\textsc{Solve-B}(\beta)$

Så lenge $\mathrm{T_R}$ er «liten» så må $\mathrm{T_B}$ være «stor nok» for $\mathrm{T_A}$

$$
\begin{array}{ll}
\mathrm{T_A}(n) & \textsc{Solve-A}(\alpha) \\
\mathrm{T_R}(n) & 1 \quad \beta = \textsc{Reduce-to-B}(\alpha) \\
\mathrm{T_B}(n) & 2 \quad \textbf{return } \textsc{Solve-B}(\beta)
\end{array}
$$

En nedre grense for $\mathrm{T_A}$ vil bli en nedre grense for $\mathrm{T_B}$

$$
\begin{array}{ll}
T_A(n) & \textsc{Solve-A}(\alpha) \\
T_R(n) & 1 \quad \beta = \textsc{Reduce-to-B}(\alpha) \\
T_B(n) & 2 \quad \textbf{return } \textsc{Solve-B}(\beta)
\end{array}
$$

En enkel reduksjon fra A til B $\Longrightarrow$ B er minst like vanskelig som A

$$\bullet \; \xmapsto[\leqslant_{\mathbf{P}}]{\quad O(n^k) \quad} \; \bullet$$

Takeaway, $A \leq_{\mathbf{P}} B$: Om vi enkelt kan bruke B til å løse A . . .

$$\bullet \xmapsto[\leqslant_{\mathbf{P}}]{\mathrm{O}(n^k)} \bullet$$

…så kan det umulig være vanskeligere å løse B enn A

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.

› **Kompletthet:**
Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.

De komplette problemene er altså de vanskeligste i klassen

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.
- **NPC:**
  De komplette språkene i **NP**, under polynomiske reduksjoner.

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.
- **NPC:**
  De komplette språkene i **NP**, under polynomiske reduksjoner.

Altså de vanskeligste problemene i **NP**

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.

Vi kaller gjerne klassen **NP-hard**

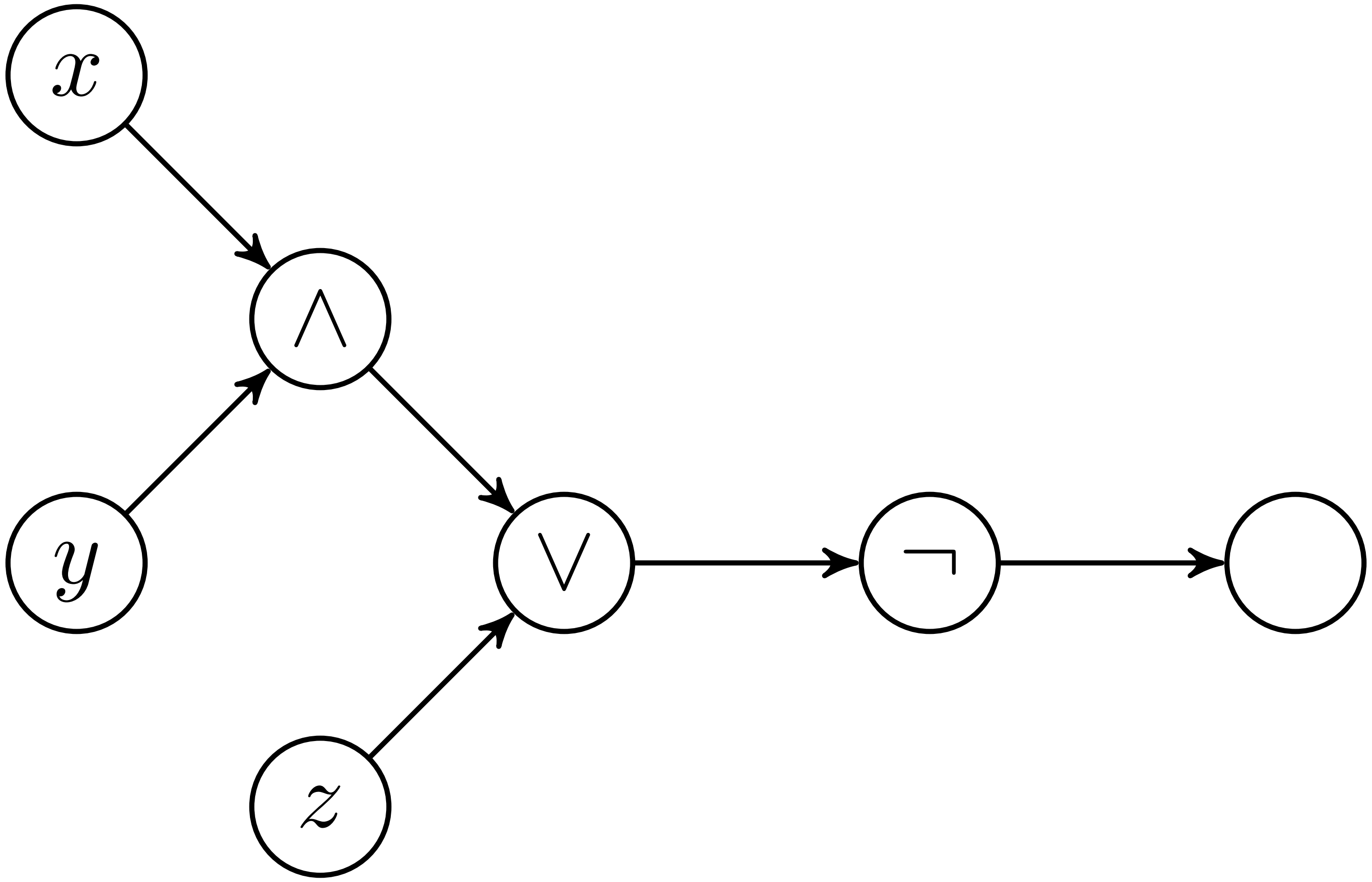
- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det
  - er **NP**-hardt, og

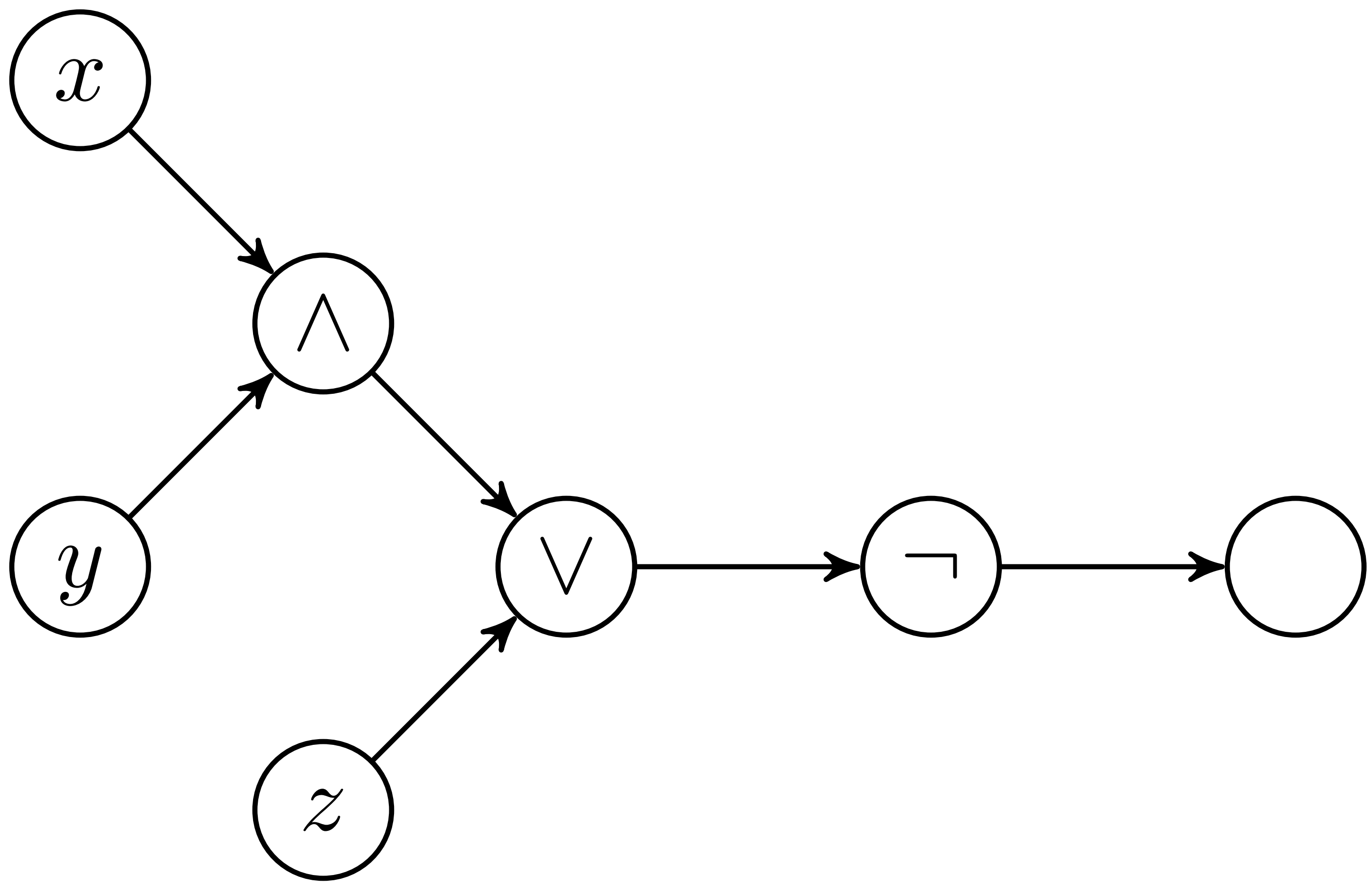
- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det
  - er **NP**-hardt, og
  - er i **NP**.

2:9

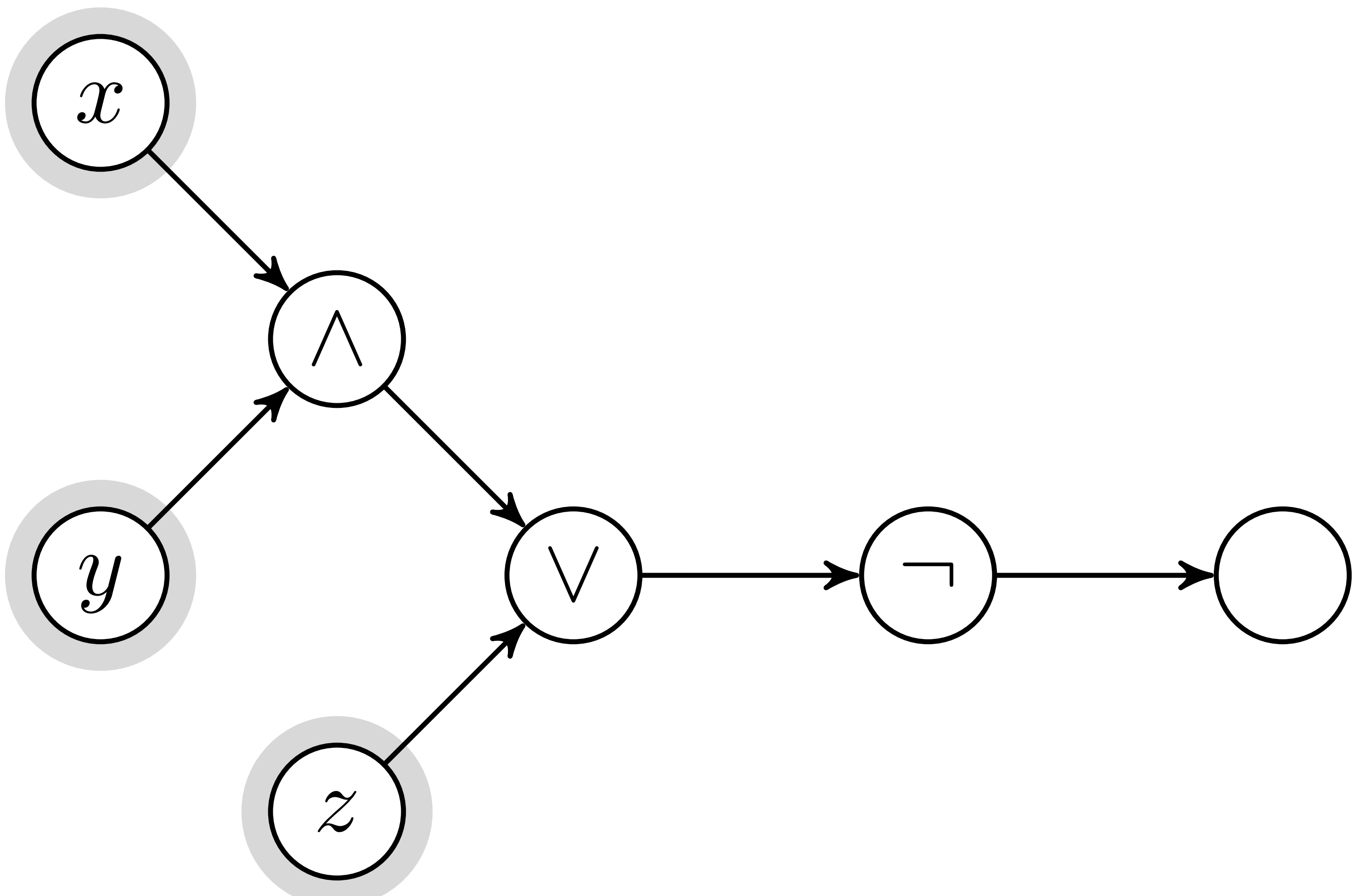
CIRCUIT-SAT



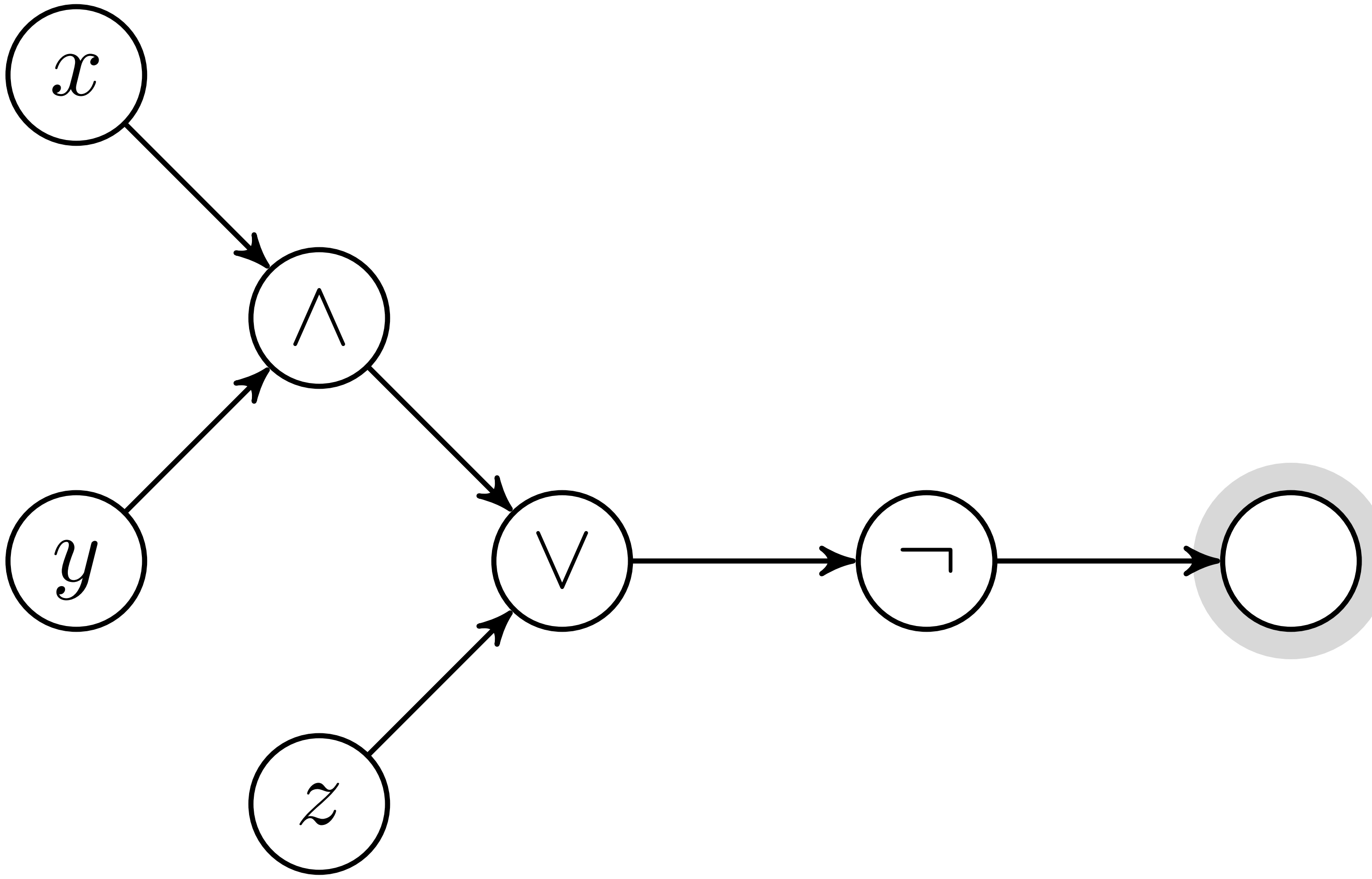
Kombinatorisk logisk krets



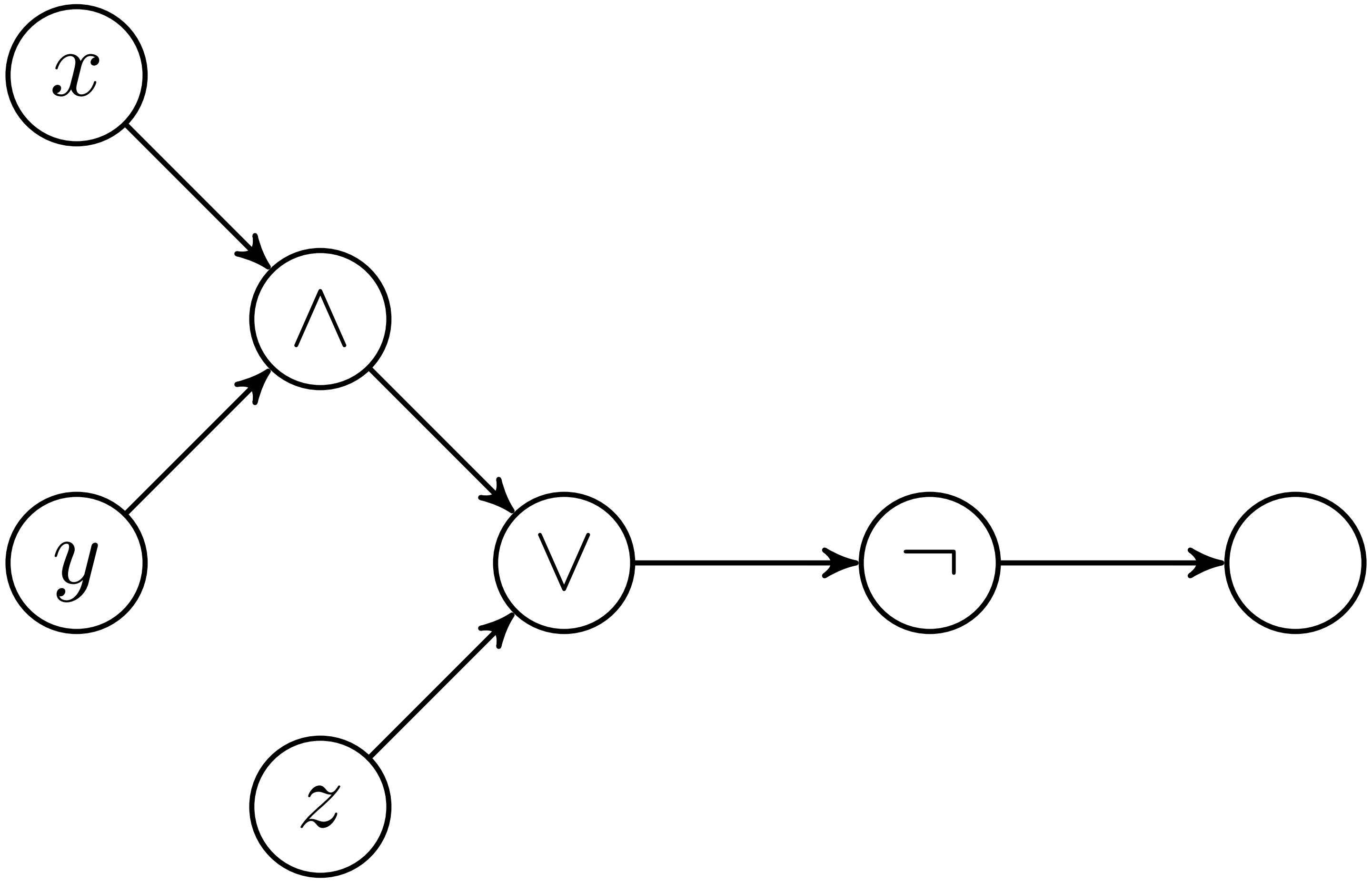
En DAG med variable og boolske operatorer på nodene



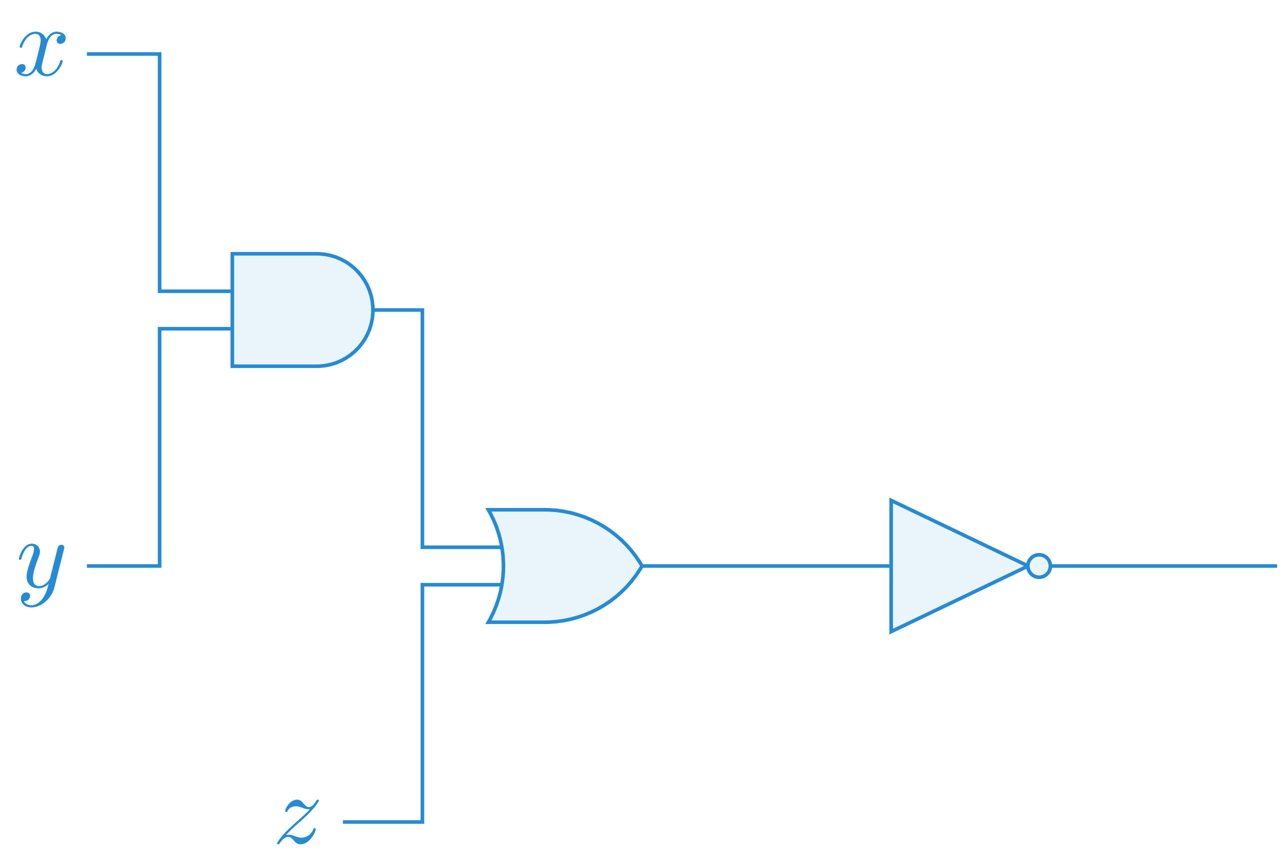
Noder uten inn-kanter: Innverdier (variable)



Noder uten ut-kanter: Utverdier



(Defineres gjerne annerledes: Inn- og ut-verdier er kanter med én node)



Dette er trolig en mer velkjent notasjon for mange

Instans: En kombinatorisk logisk krets med én utverdi.

Instans: En kombinatorisk logisk krets med én utverdi.

Spørsmål: Er det mulig å *tilfredsstille* kretsen? Det vil si, finnes det et sett med innverdier som gir utverdi 1?

› **Vil vise:**

- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**

- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**

Enkelt: La sertifikatet være innverdier, og simulér kretsen

- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**
  - $(\forall L \in \mathbf{NP})\ L \leq_{\mathbf{P}}$ CIRCUIT-SAT

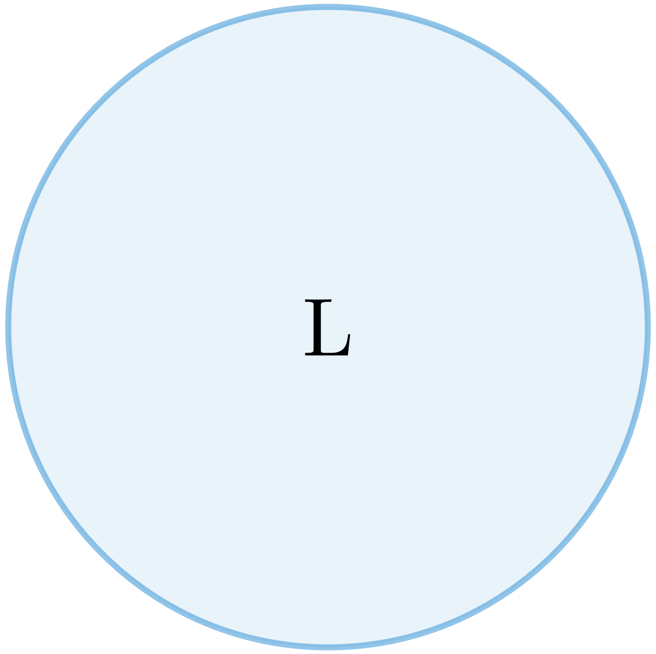
- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**
  - $(\forall L \in \mathbf{NP})\ L \leq_{\mathbf{P}}$ CIRCUIT-SAT
- La A være en verifikasjonsalgoritme for L

- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**
  - $(\forall \mathrm{L} \in \mathbf{NP})\ \ \mathrm{L} \leq_{\mathbf{P}}$ CIRCUIT-SAT
- La A være en verifikasjonsalgoritme for L
- For en gitt x vil vi simulere A med en kombinatorisk digital krets, slik at:

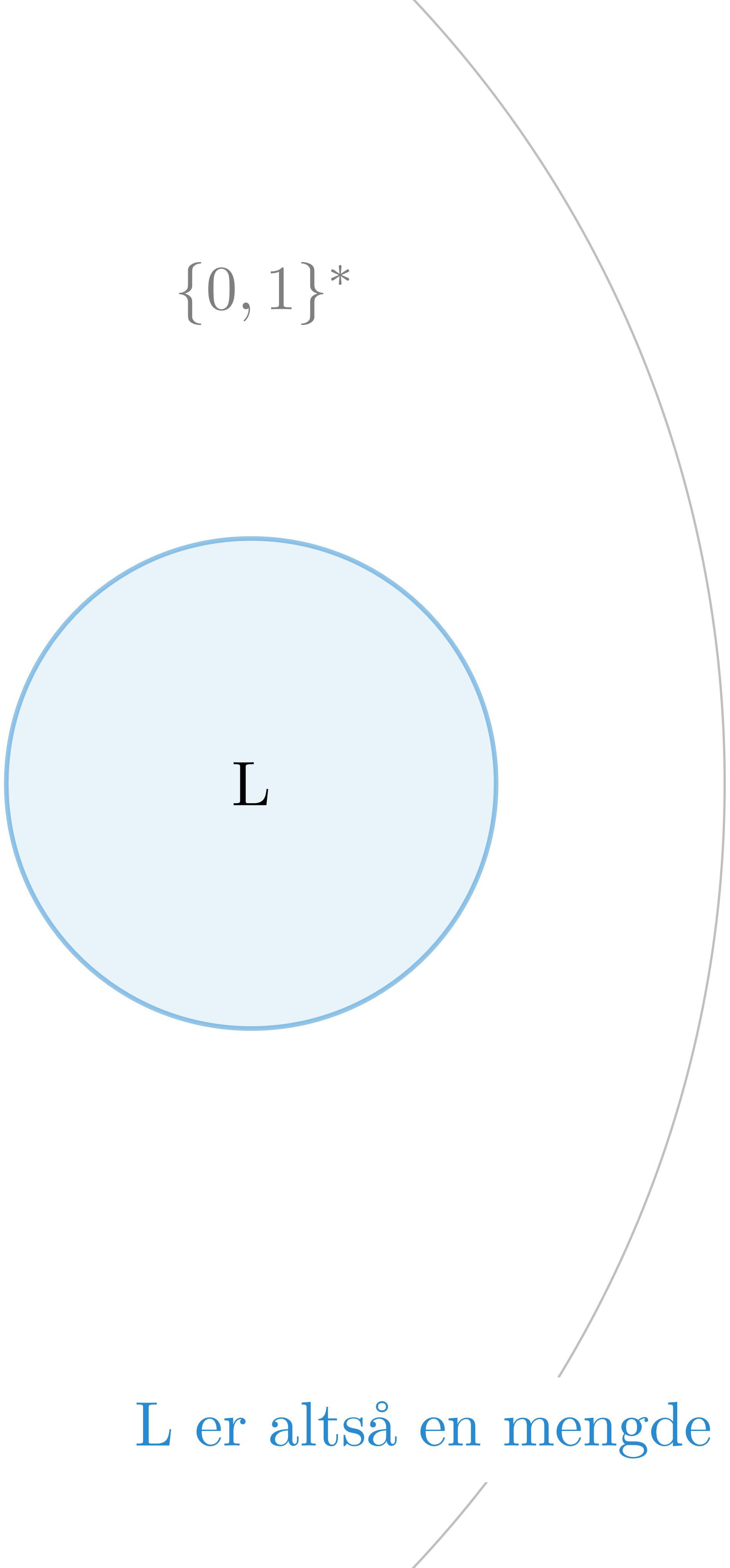
- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**
  - $(\forall \mathrm{L} \in \mathbf{NP})\ \mathrm{L} \leq_{\mathbf{P}}$ CIRCUIT-SAT
- La A være en verifikasjonsalgoritme for L
- For en gitt x vil vi simulere A med en kombinatorisk digital krets, slik at:
  - Kretsen kan tilfredsstilles hvis og bare hvis . . .

- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**
  - $(\forall \mathrm{L} \in \mathbf{NP})\ \mathrm{L} \leq_{\mathbf{P}}$ CIRCUIT-SAT
- La A være en verifikasjonsalgoritme for L
- For en gitt x vil vi simulere A med en kombinatorisk digital krets, slik at:
  - Kretsen kan tilfredsstilles hvis og bare hvis . . .
  - det finnes en y slik at $\mathrm{A}(x, y) = 1$

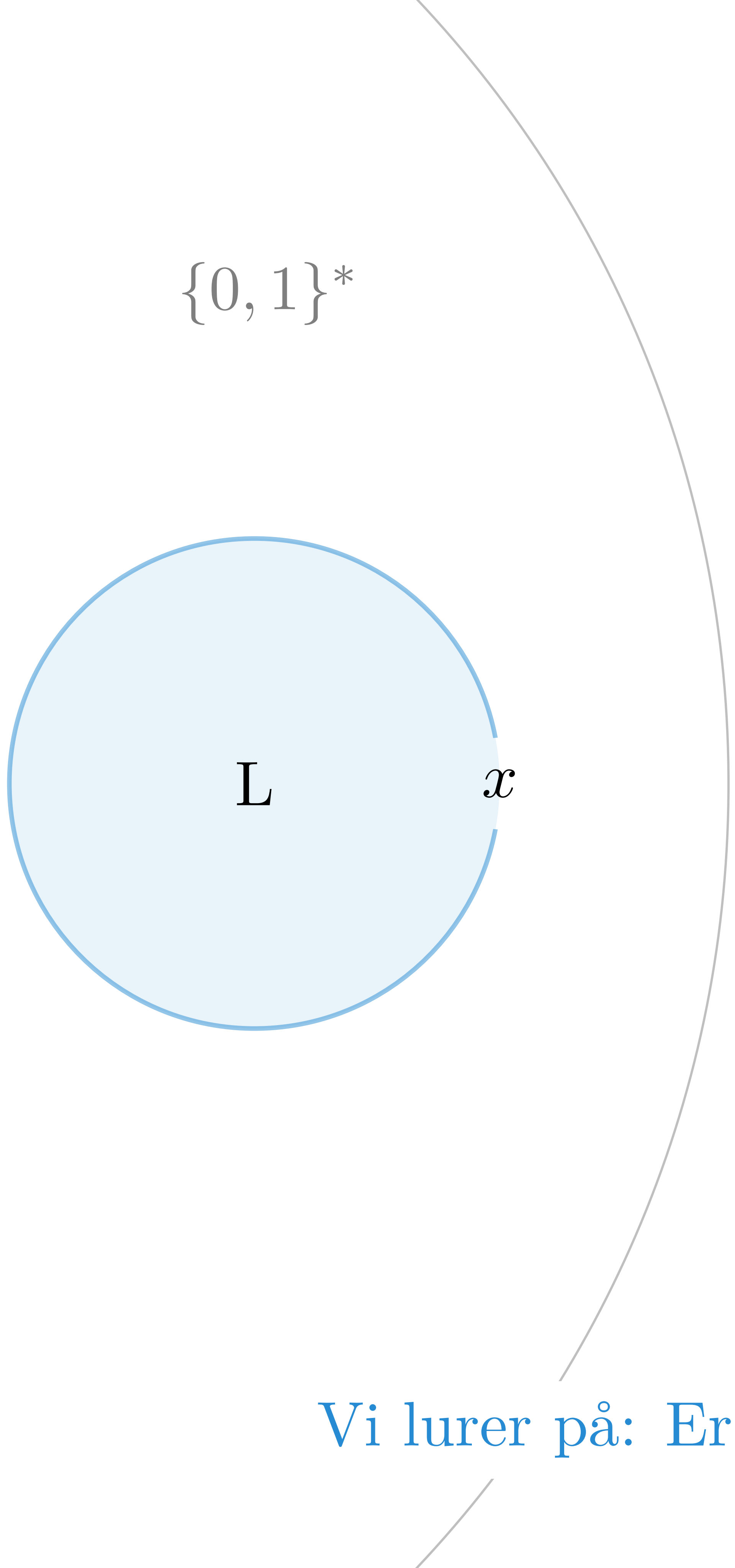
- **Vil vise:**
  - CIRCUIT-SAT $\in$ **NP**
  - $(\forall \mathrm{L} \in \mathbf{NP})\ \mathrm{L} \leq_{\mathbf{P}}$ CIRCUIT-SAT
- La A være en verifikasjonsalgoritme for L
- For en gitt x vil vi simulere A med en kombinatorisk digital krets, slik at:
  - Kretsen kan tilfredsstilles hvis og bare hvis . . .
  - det finnes en y slik at $\mathrm{A}(x, y) = 1$
- Det er jo akkurat tilfellene der $x \in \mathrm{L}$



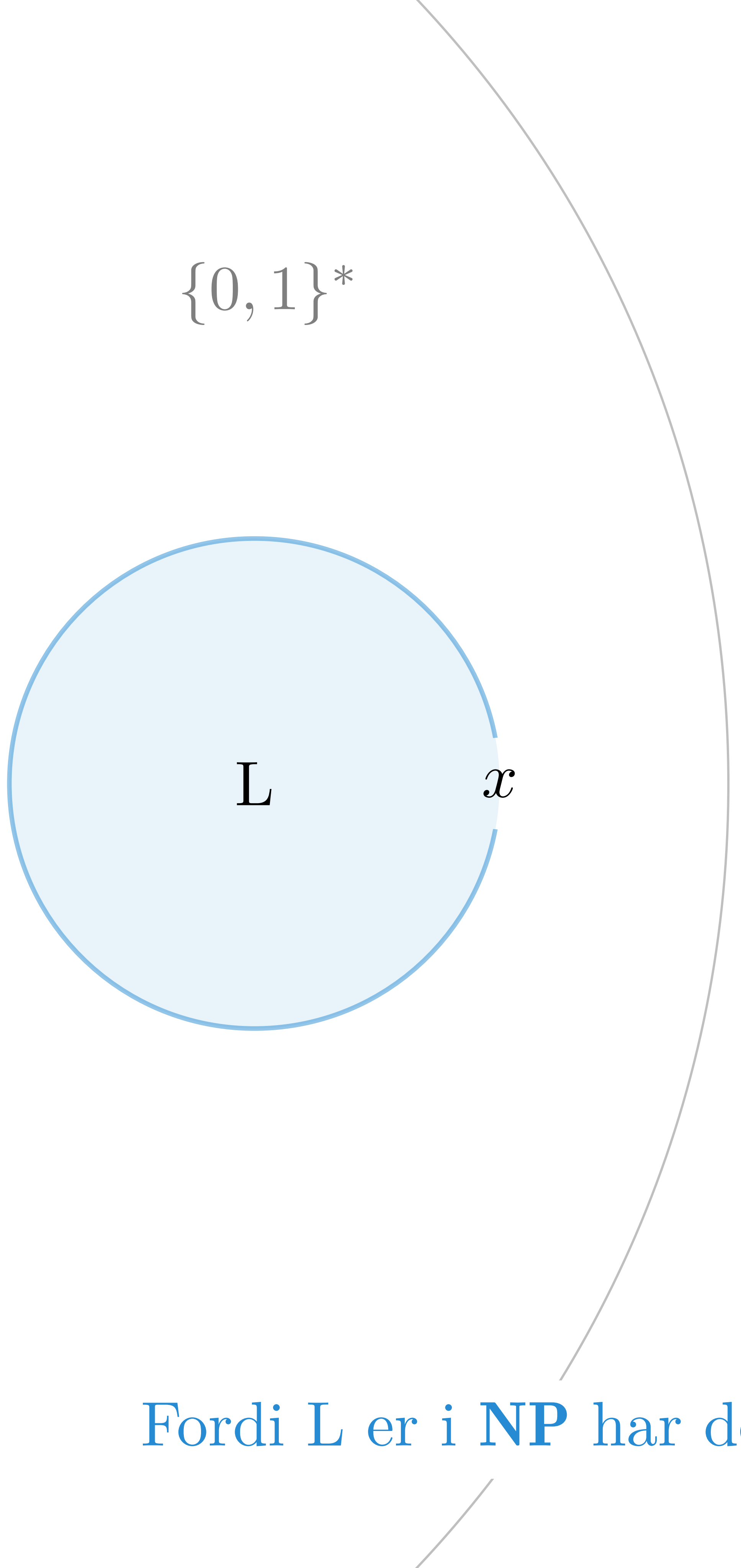
Vi starter med vilkårlig språk L fra **NP**



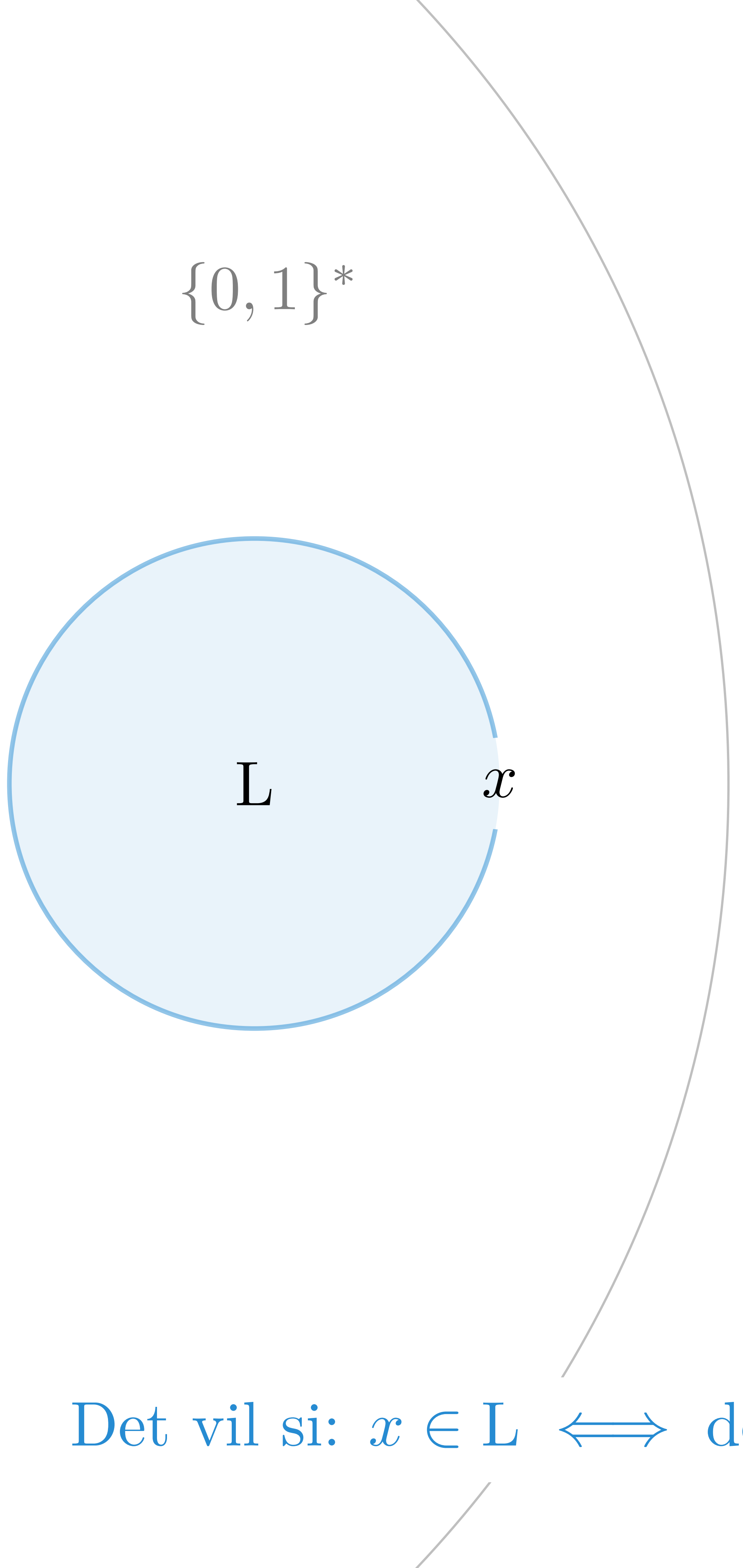
L er altså en mengde med bitstrenger $\langle 0, 1, 1, \ldots, 0 \rangle$



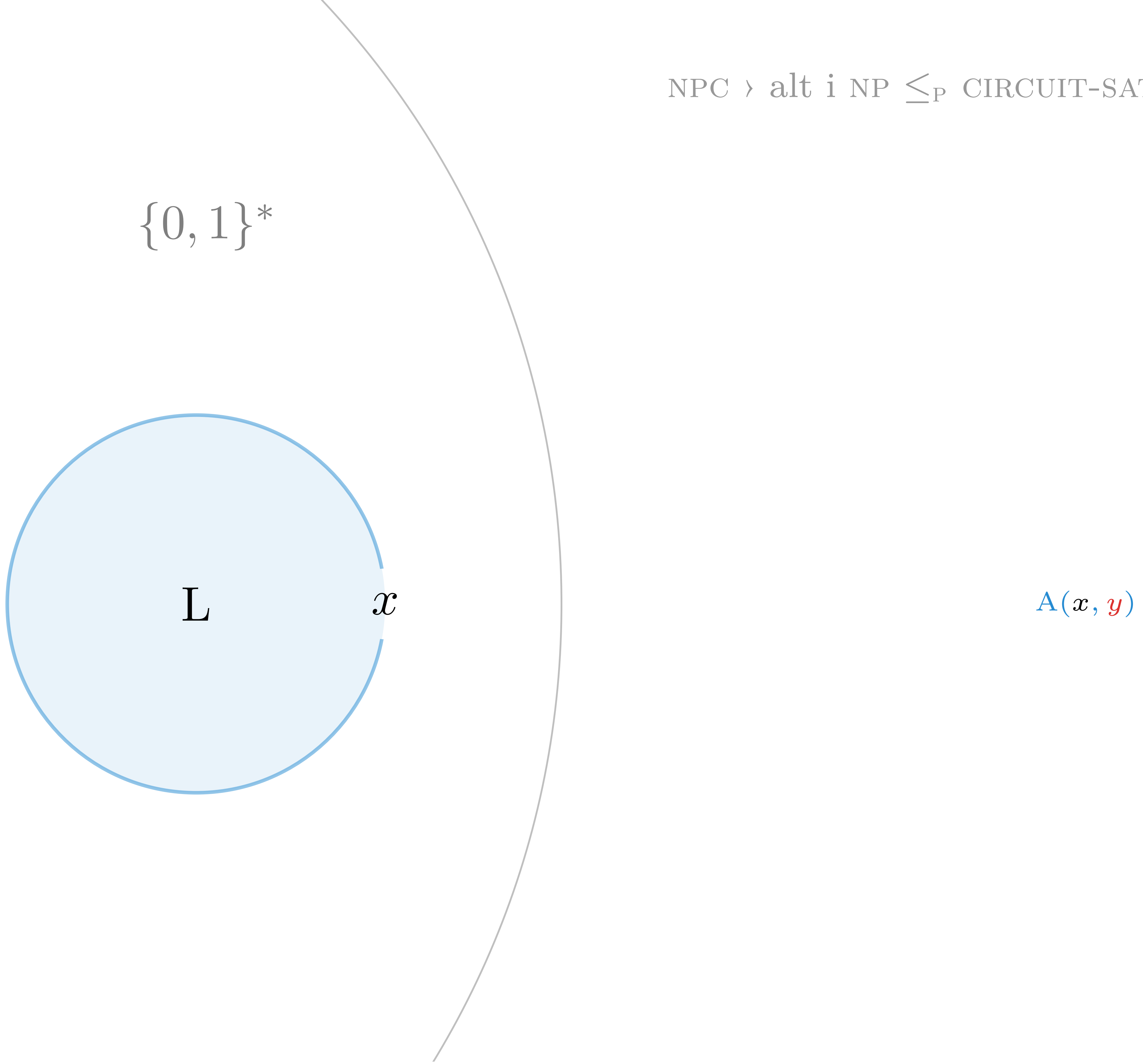
Vi lurer på: Er bitstrengen x med i L?



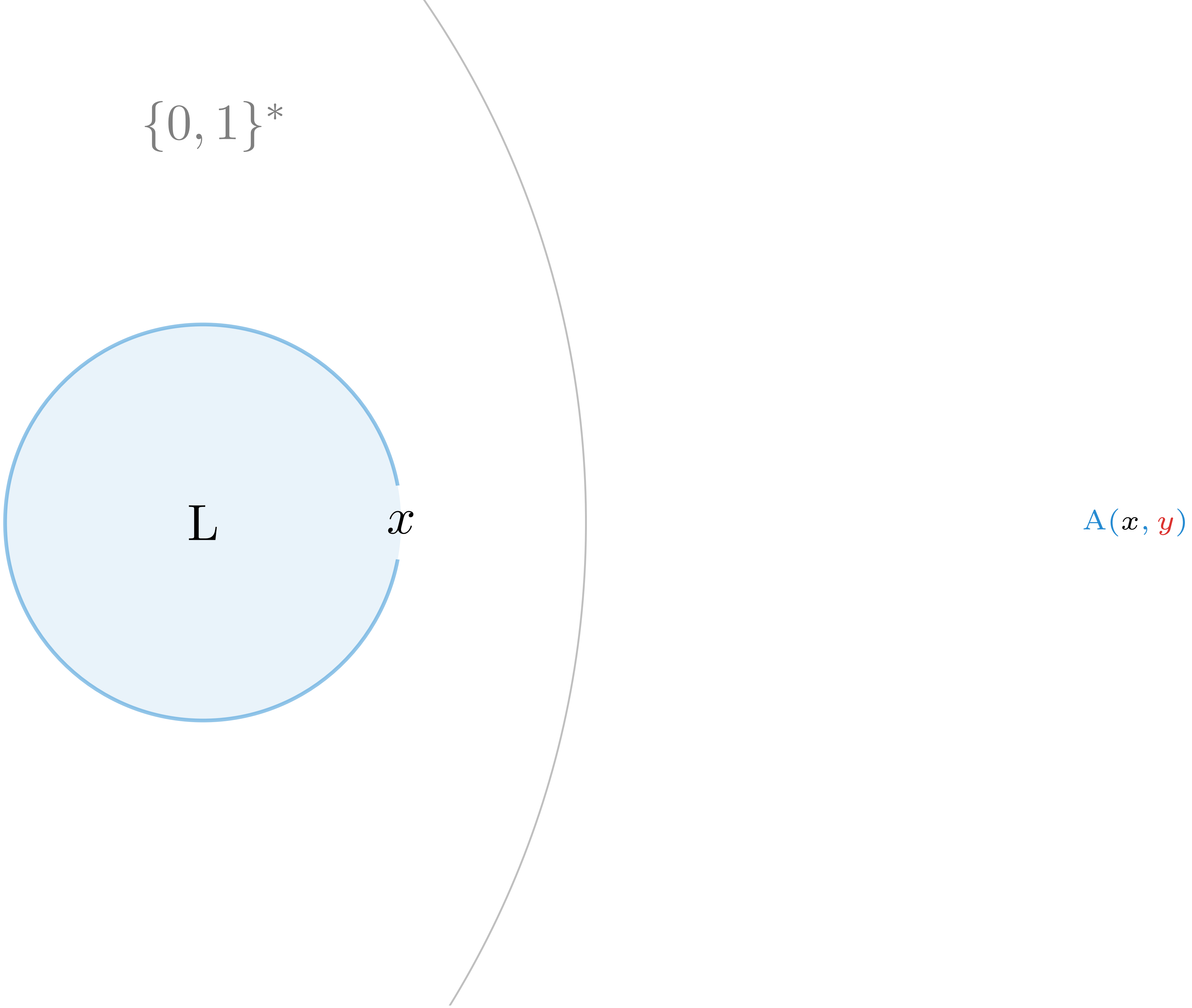
Fordi L er i **NP** har det en verifikasjonsalgoritme A



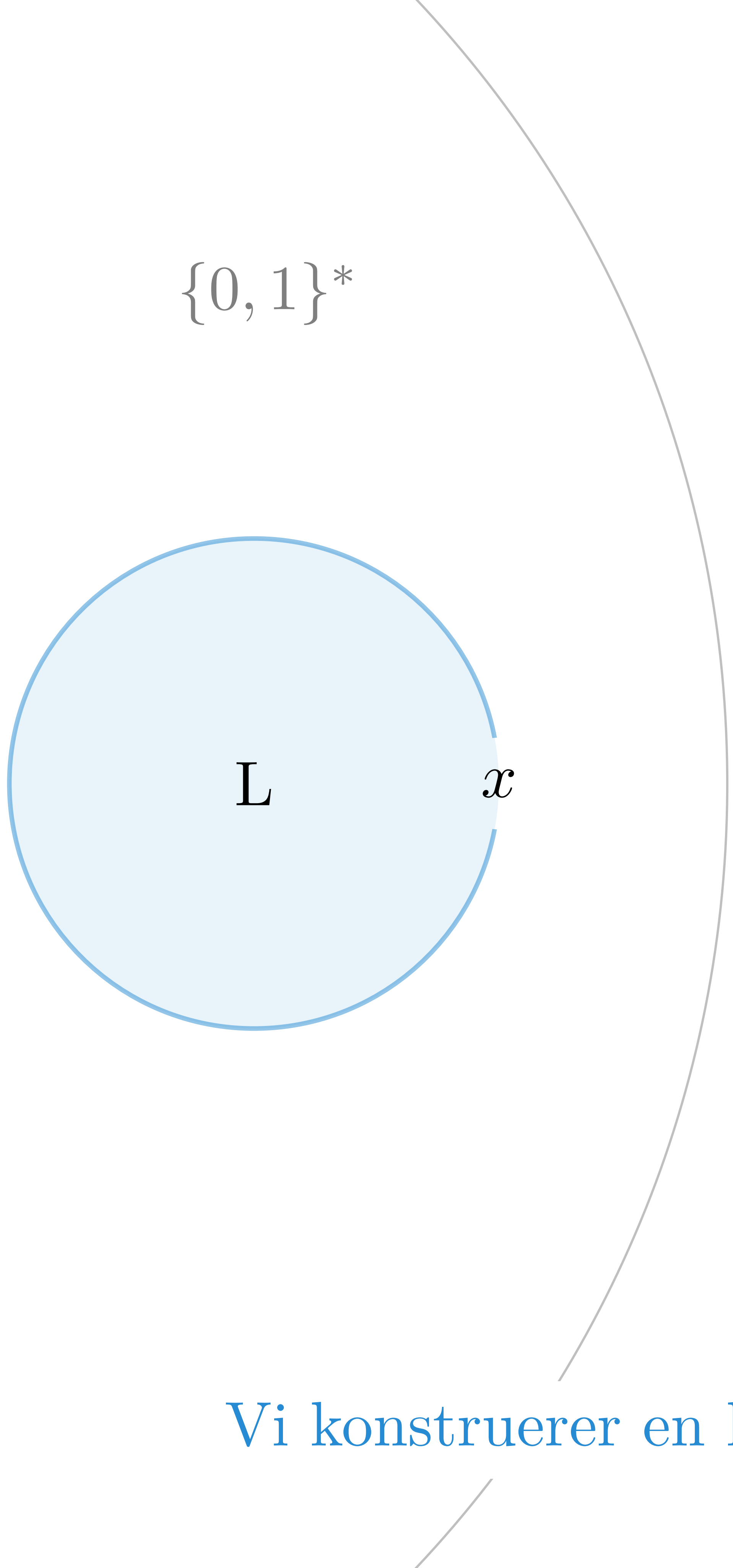
Det vil si: $x \in \mathrm{L} \iff$ det finnes en y slik at $\mathrm{A}(x, y) = 1$

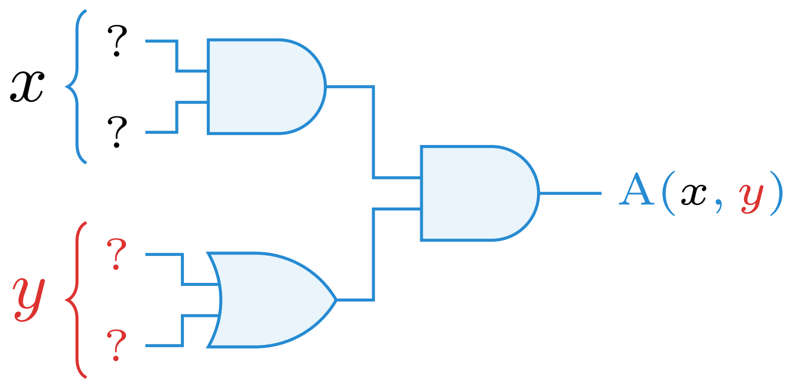


Det vil si: $x \in \mathrm{L} \iff$ det finnes en y slik at $\mathrm{A}(x, y) = 1$

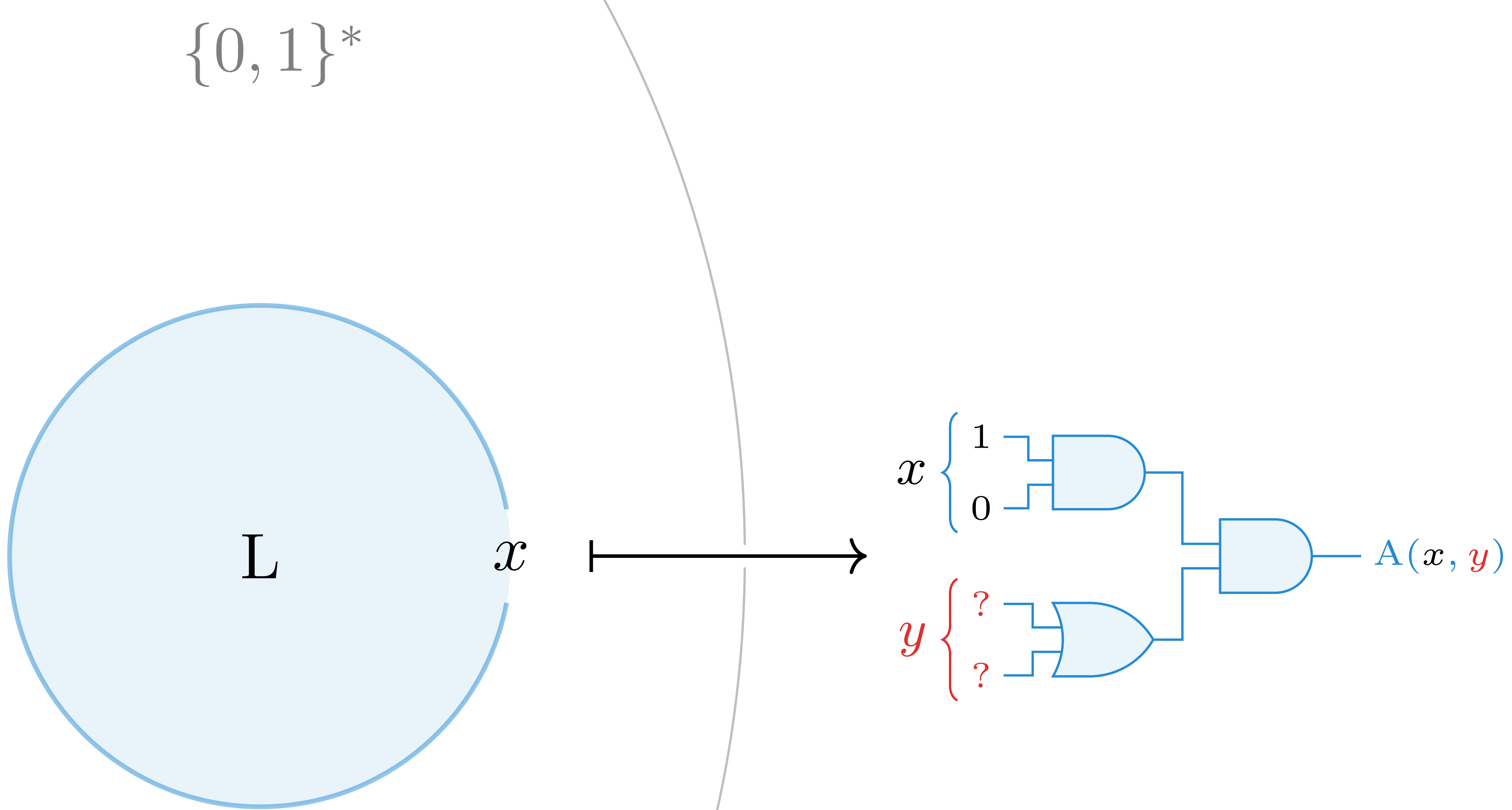


Vi konstruerer en krets som implementerer A

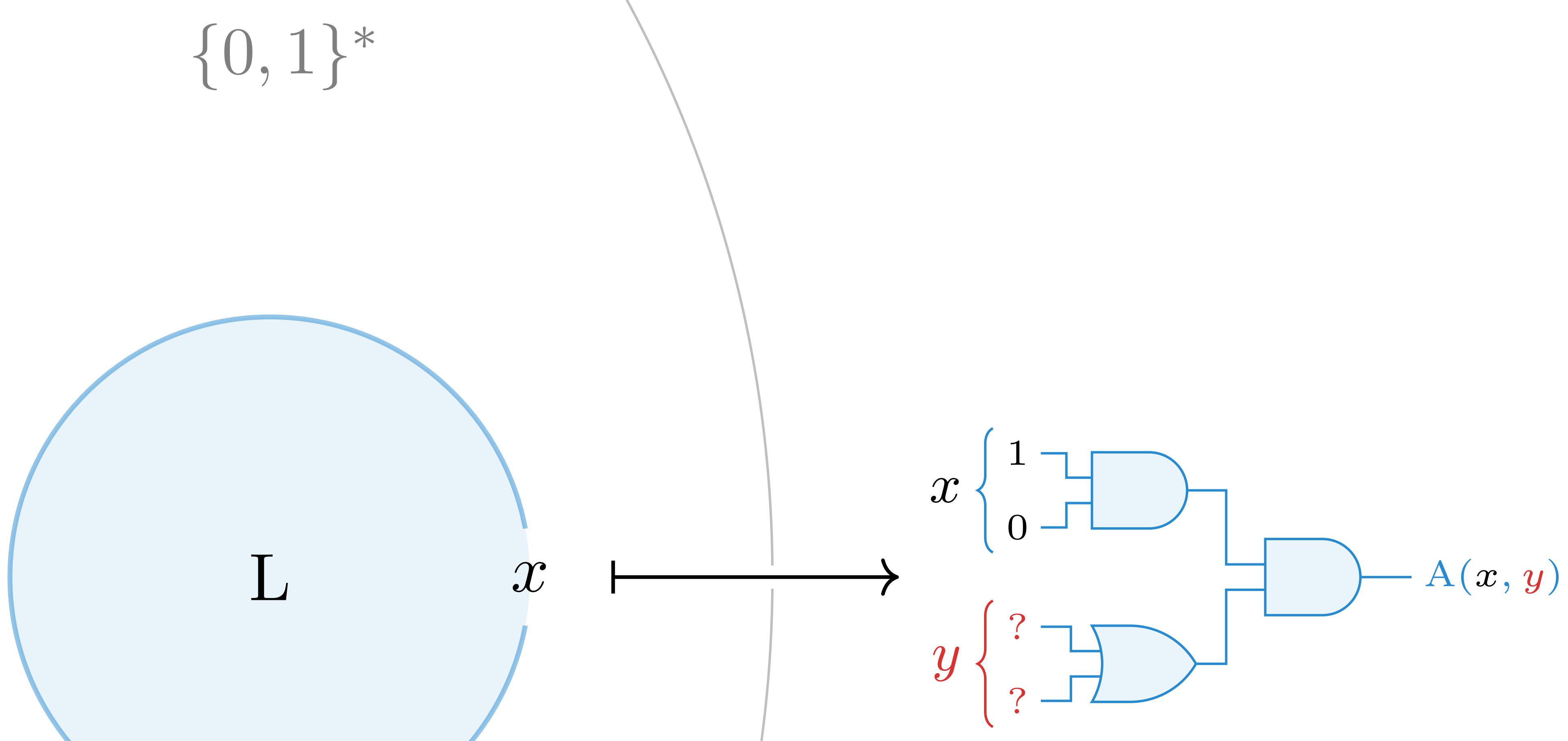




Vi konstruerer en krets som implementerer A

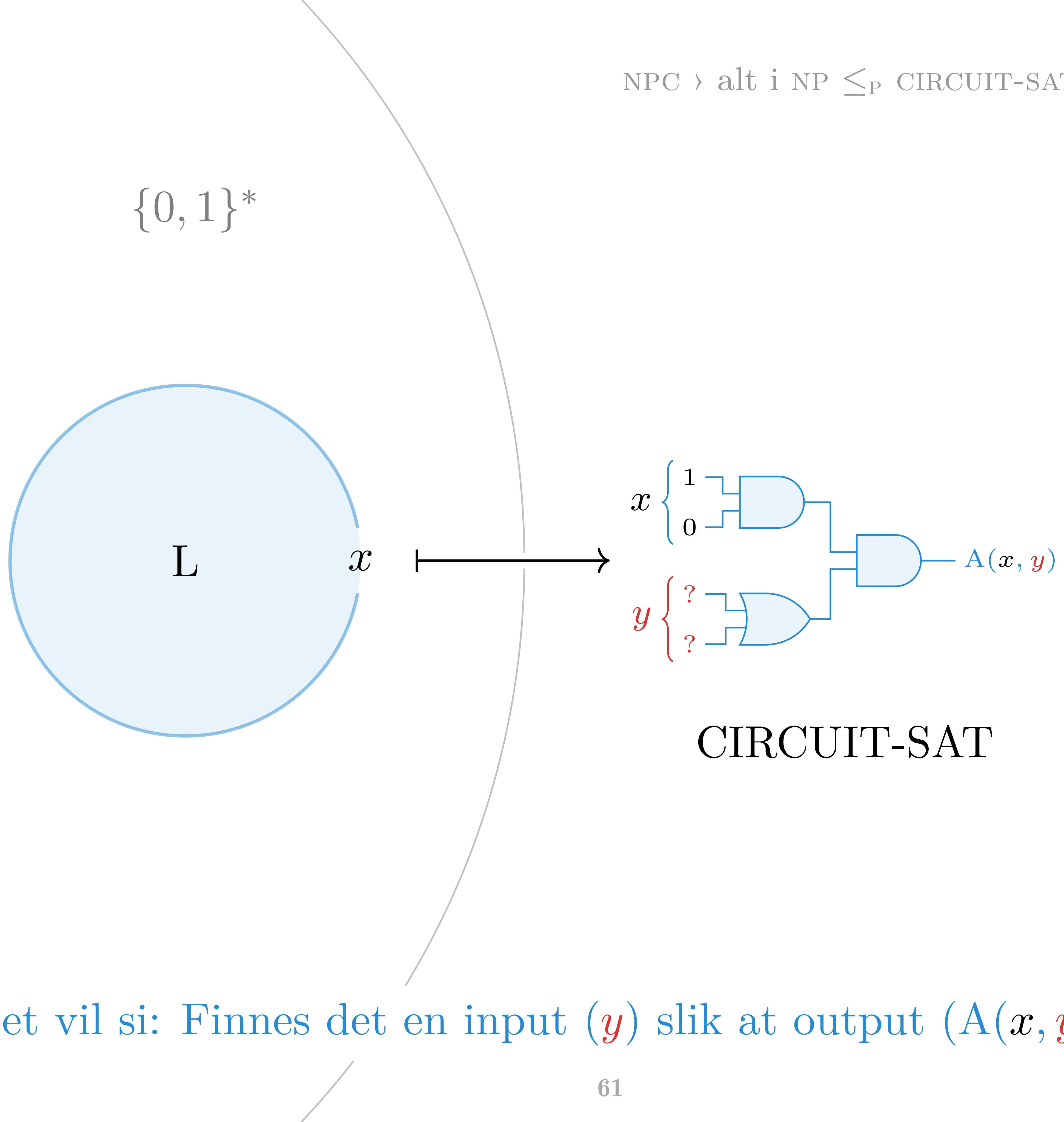


Reduksjonen blir at vi «bygger inn» x i kretsen



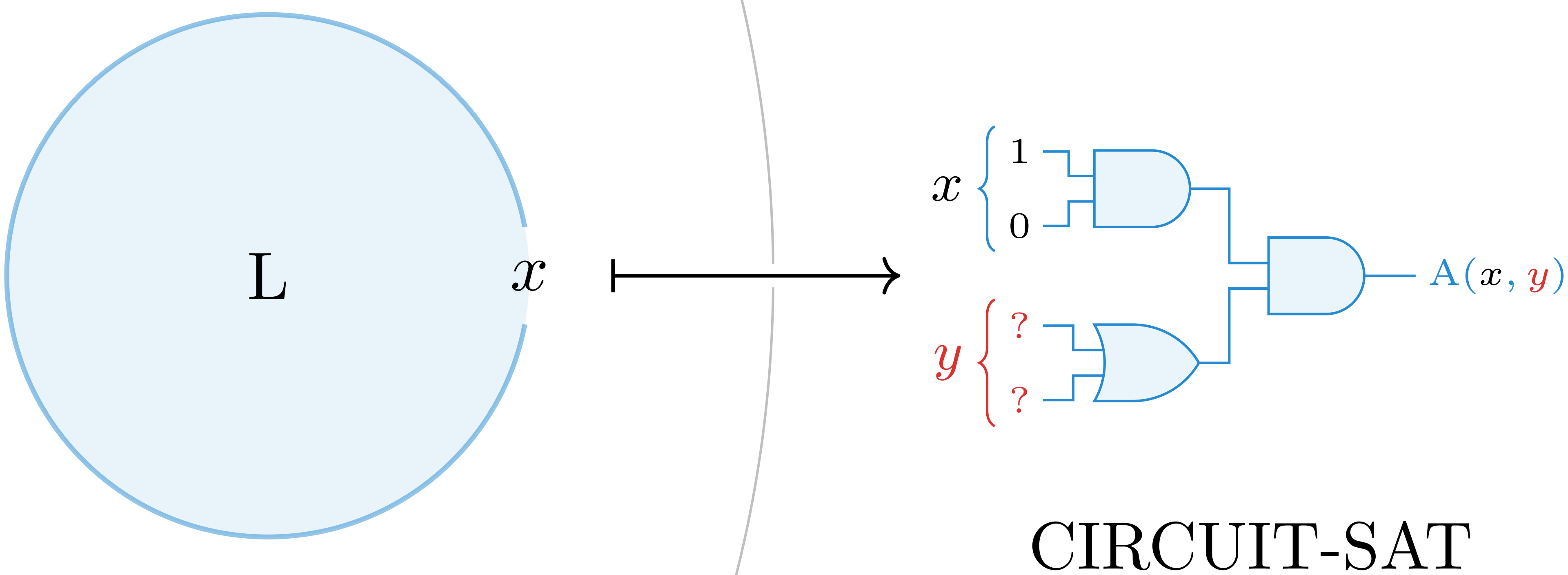
CIRCUIT-SAT

Kan denne kretsen tilfredsstilles?

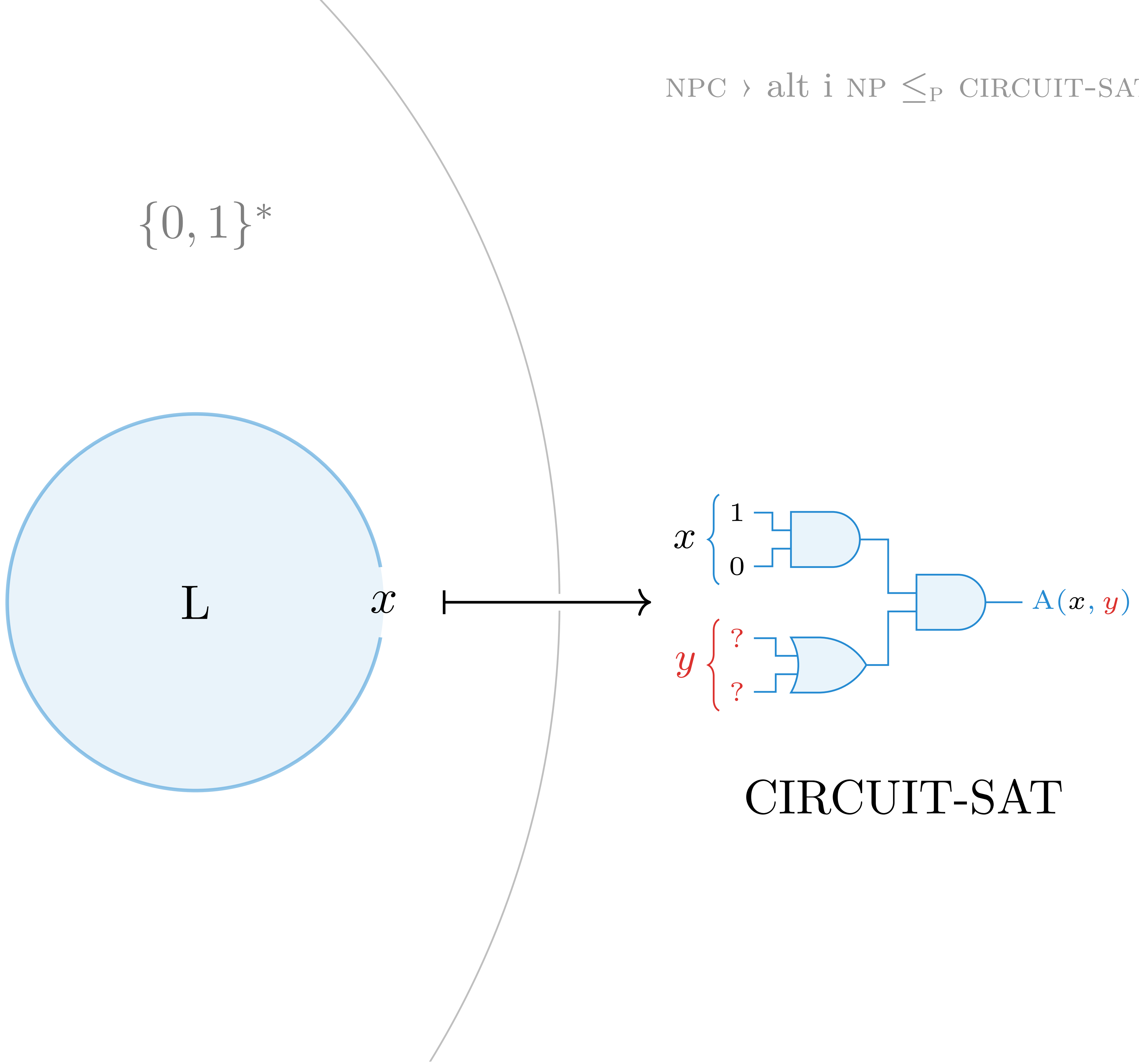


Det vil si: Finnes det en input (y) slik at output ($A(x, y)$) blir 1?

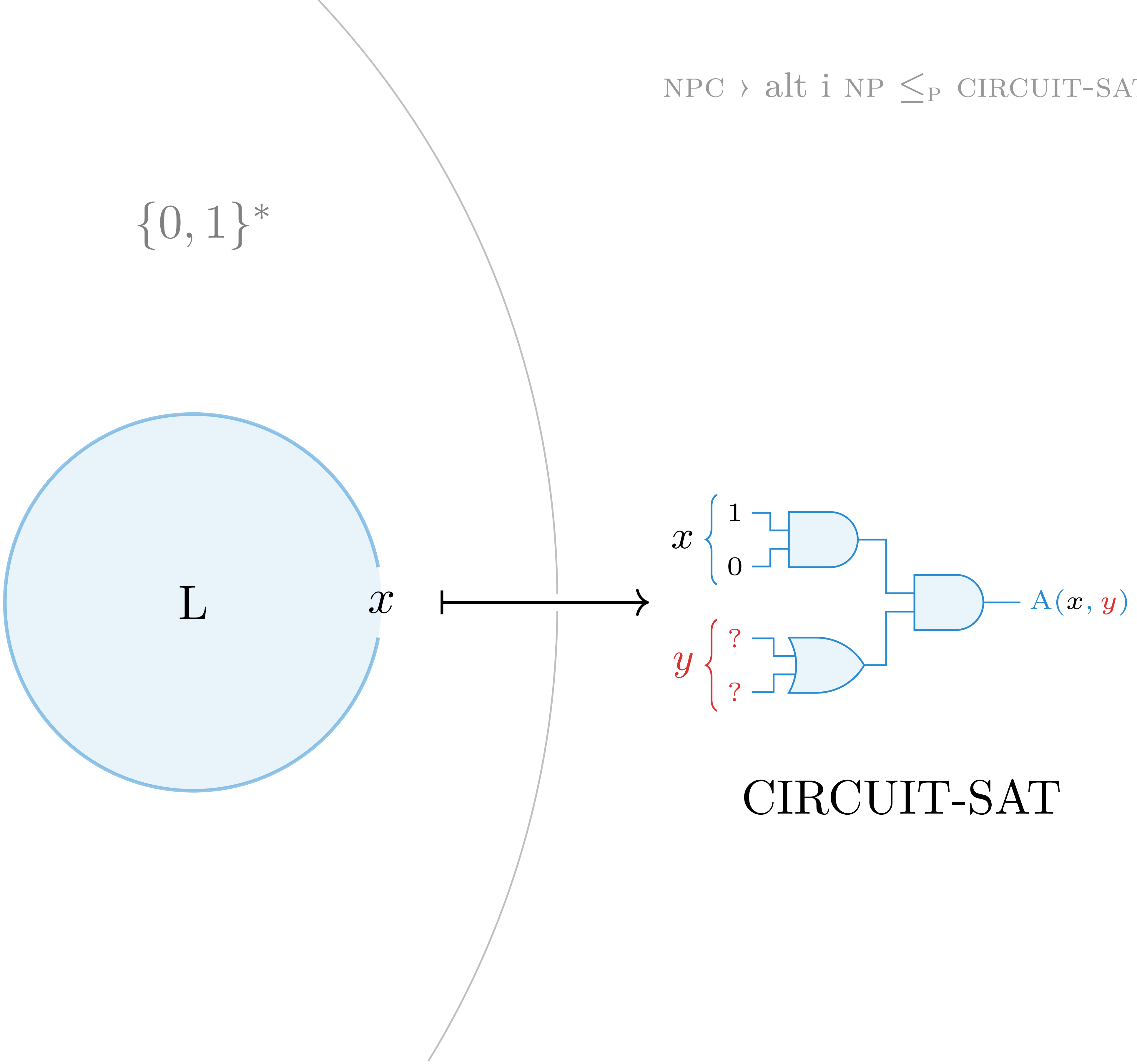
$\{0,1\}^*$



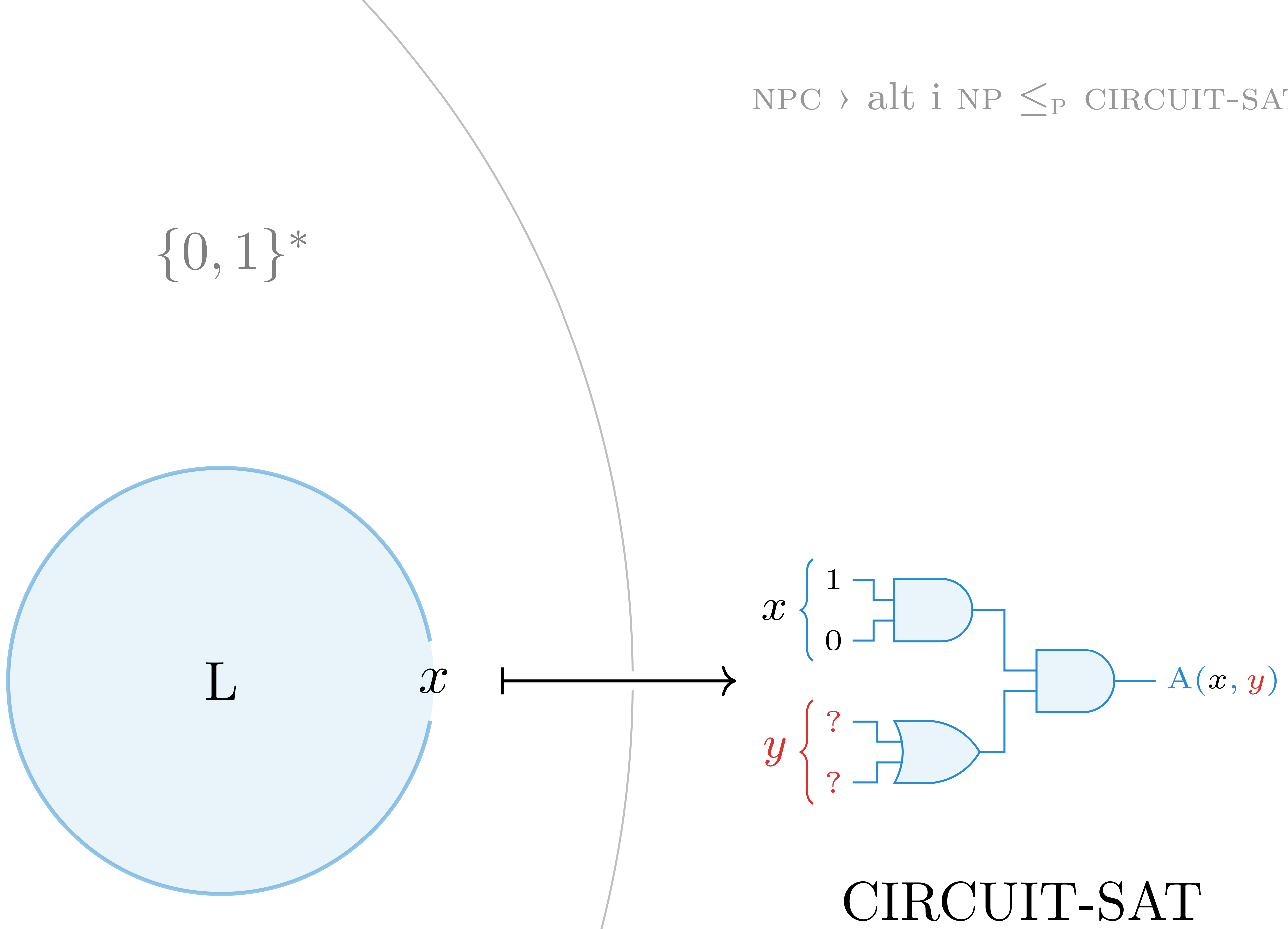
Husk: $x \in L \iff$ det finnes en y slik at $A(x, y) = 1$



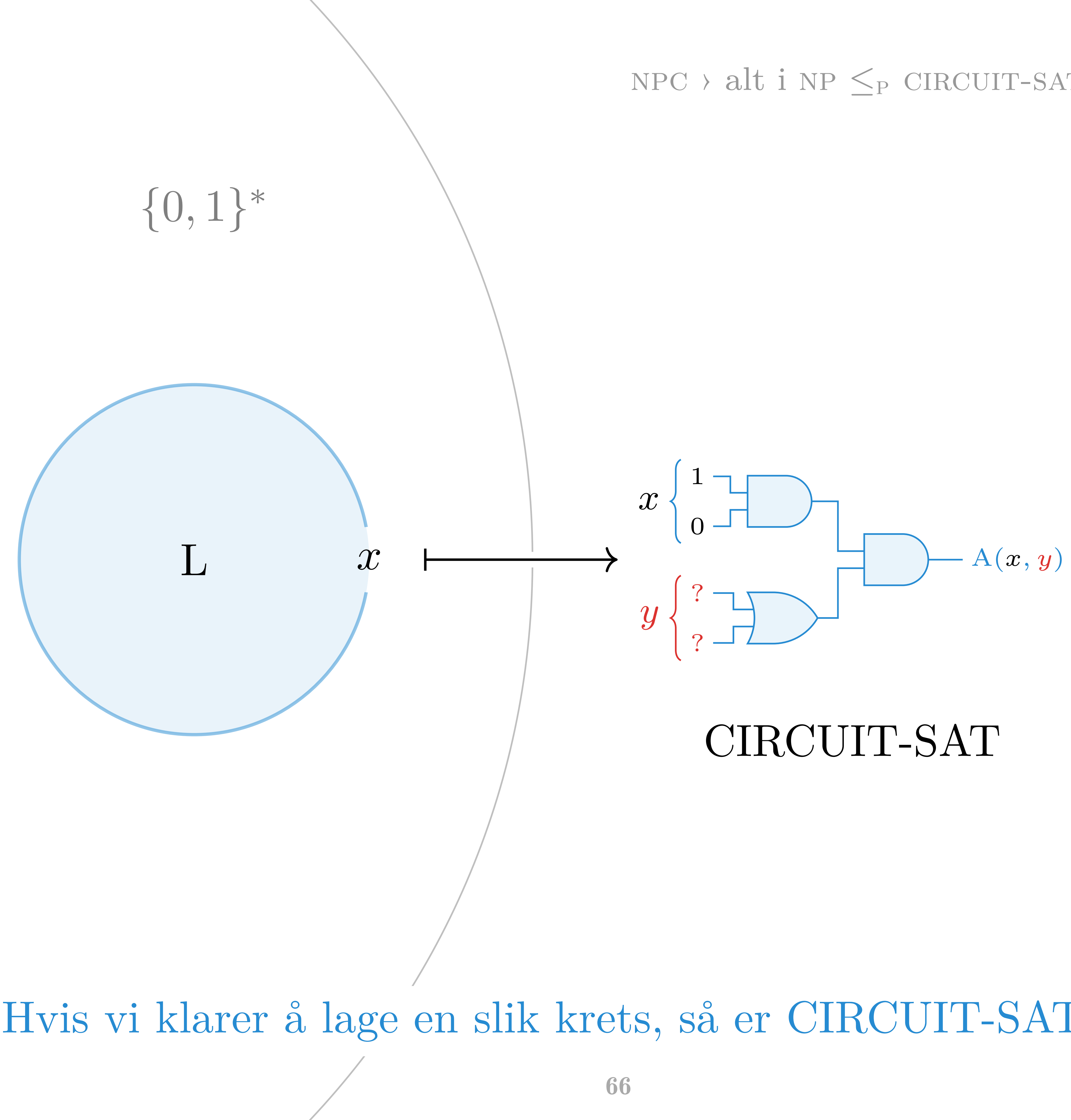
Med andre ord: Svaret på de to spørsmålene blir det samme!



Vi valgte L vilkårlig fra **NP**, så reduksjonen gjelder alt i **NP**

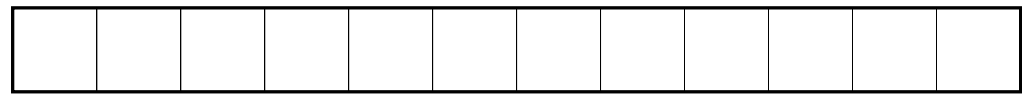


Reduksjonen er polynomisk og kretsen verifiseres lett

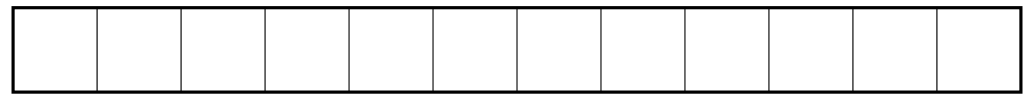


Hvis vi klarer å lage en slik krets, så er CIRCUIT-SAT i **NPC**!

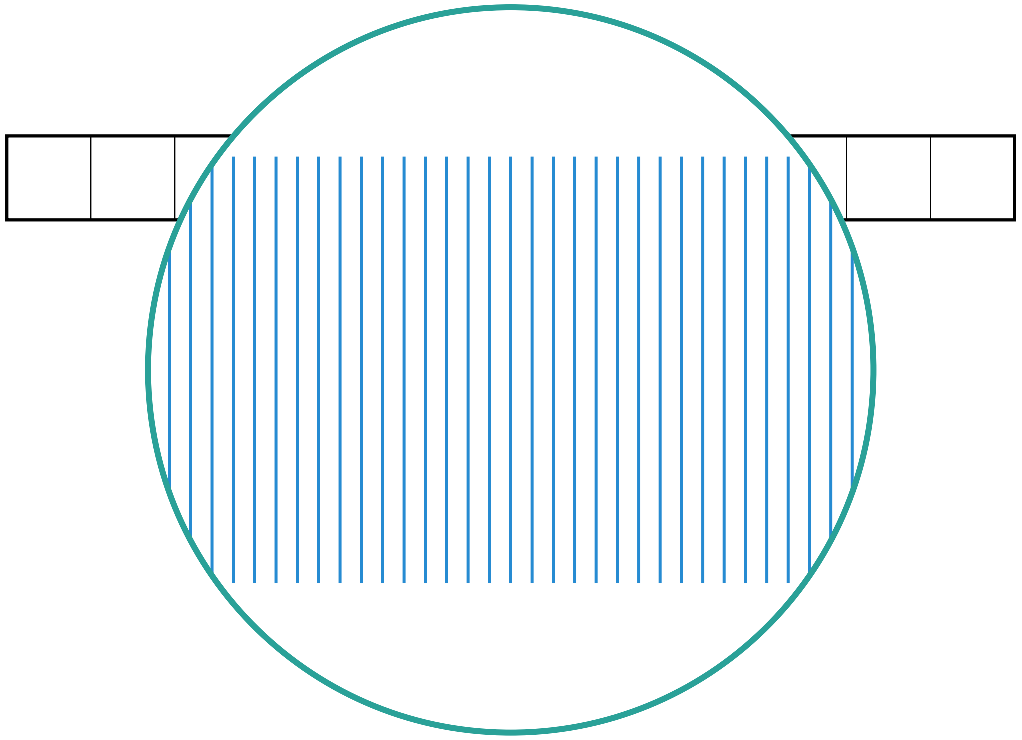
Vi simulerer en datamaskin som kan utføre A



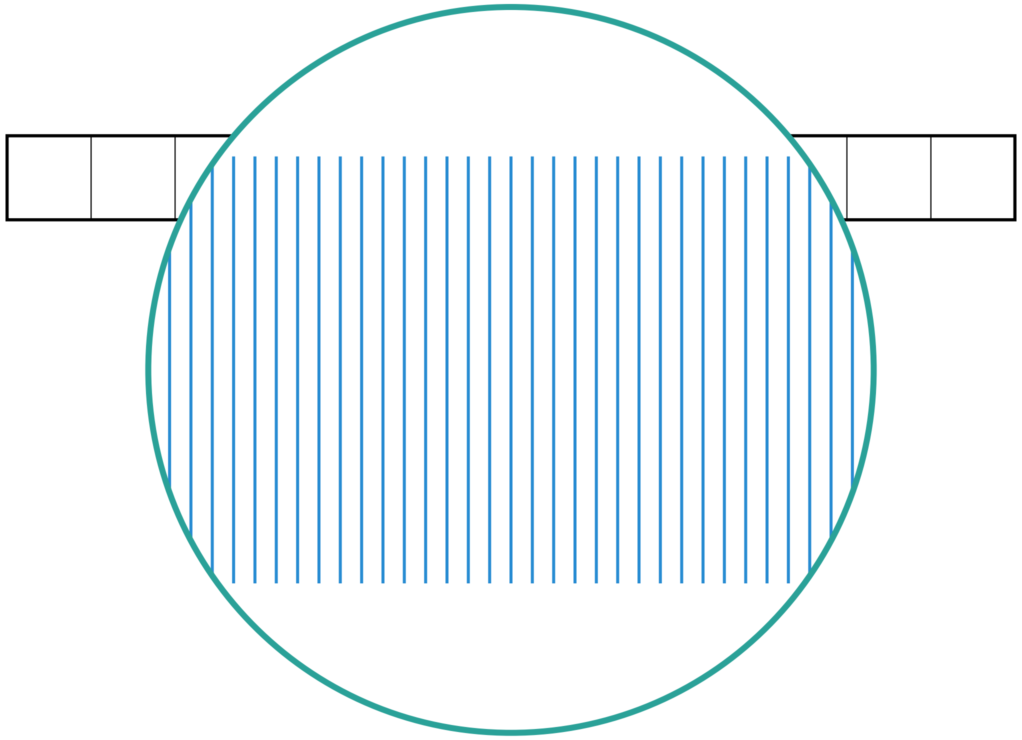
Vi har «snapshots» av minnet – såkalte *konfigurasjoner*



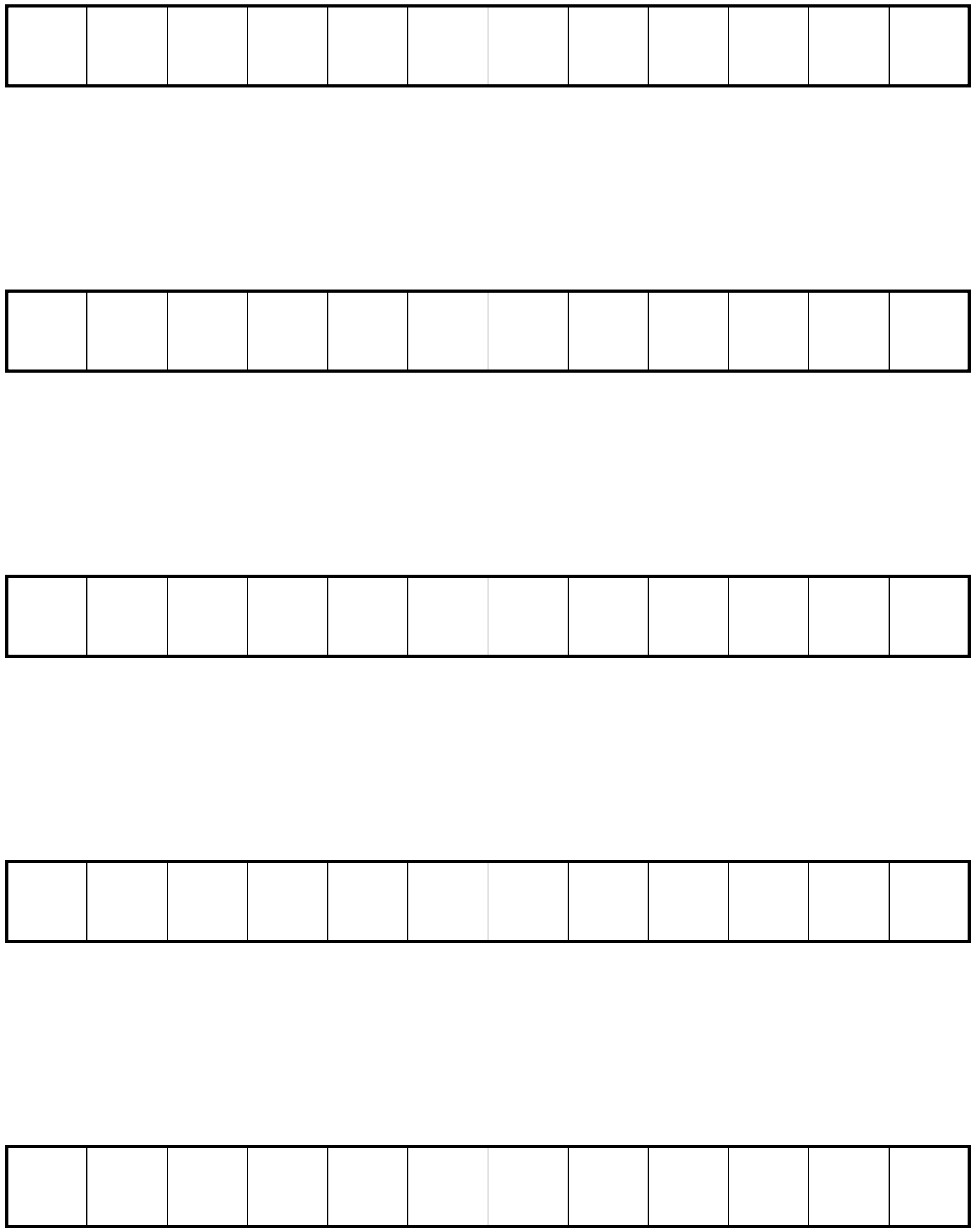
En slik konfigurasjon er bare et sett med kanter (signaler, *wires*)



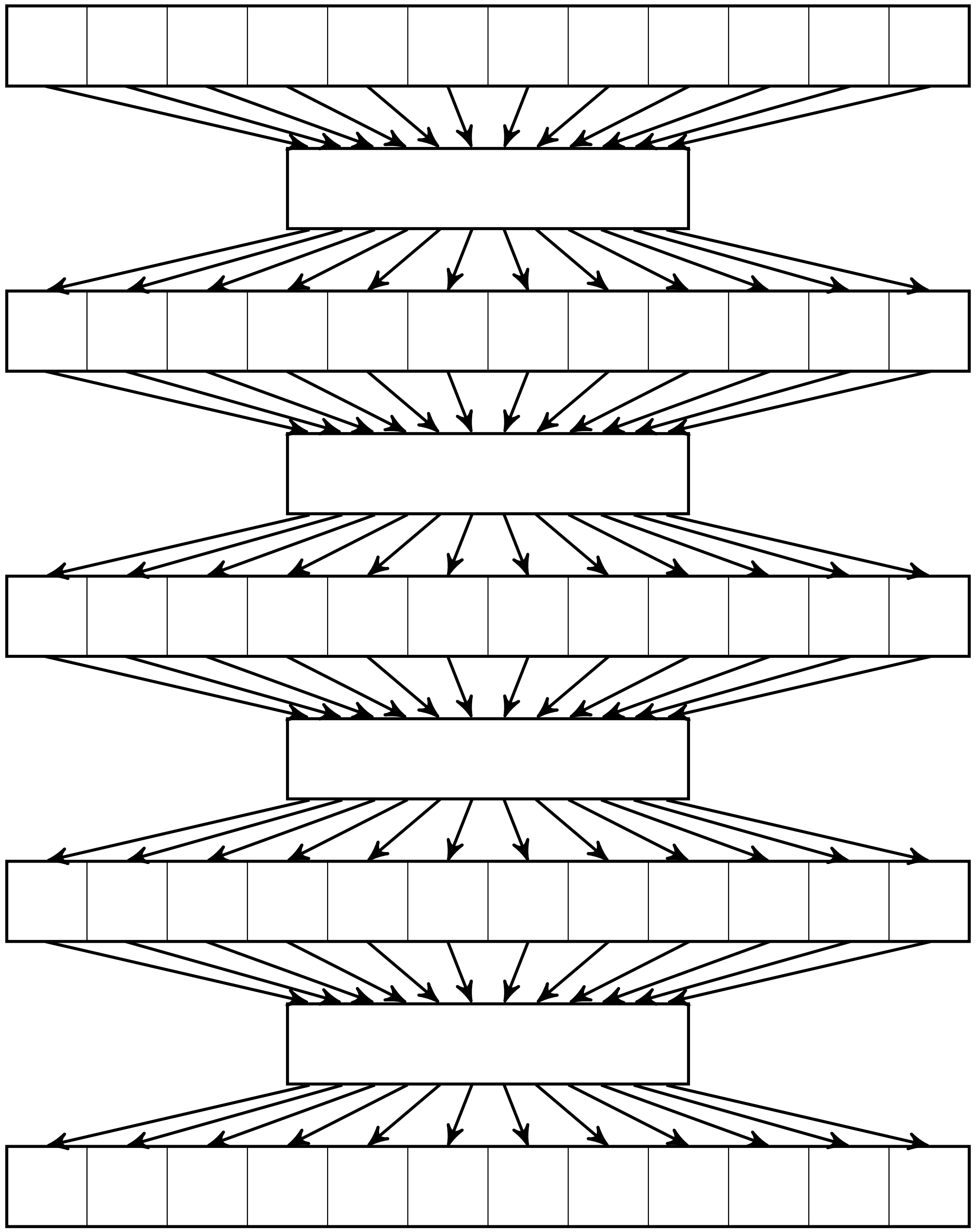
En slik konfigurasjon er bare et sett med kanter (signaler, *wires*)



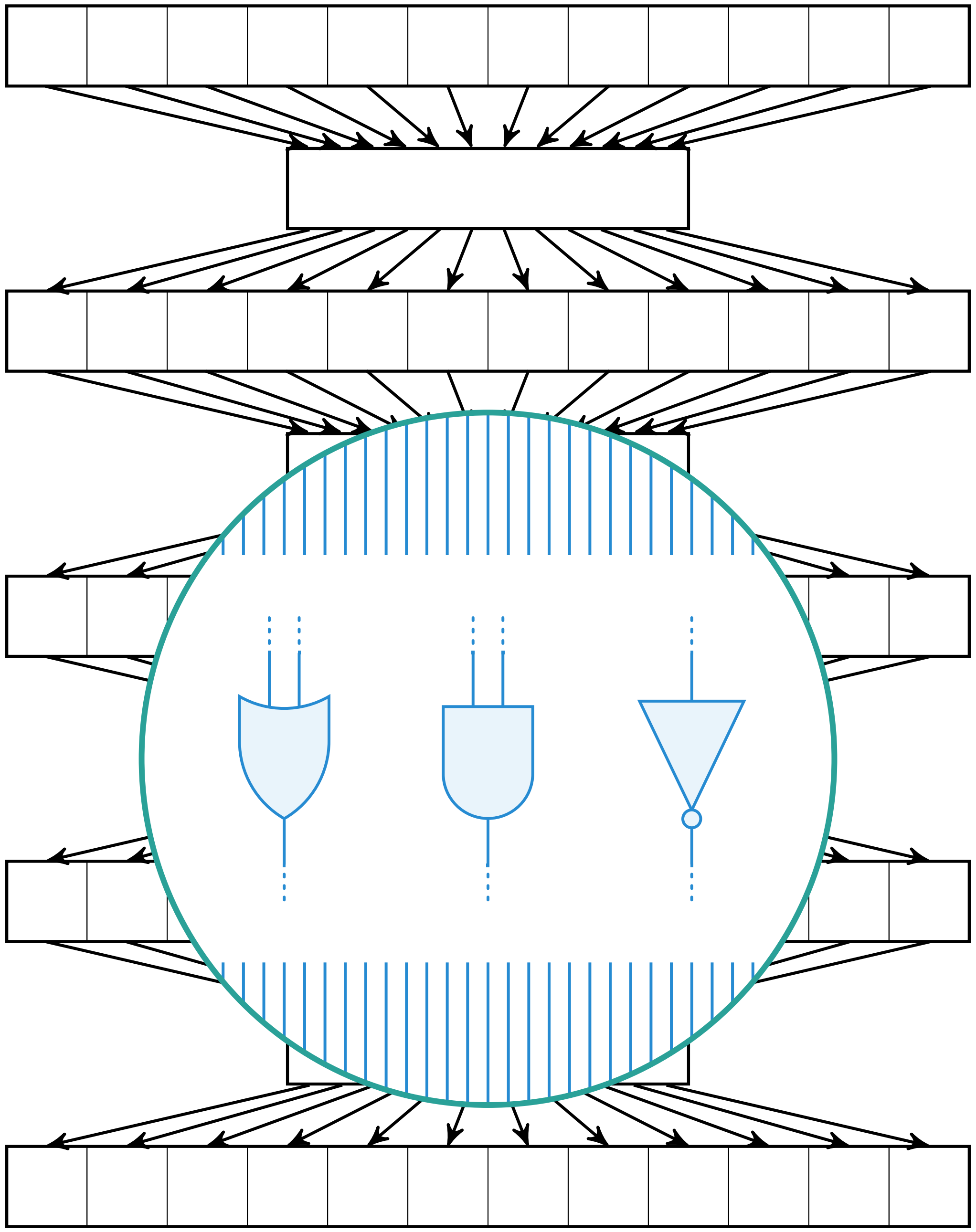
Hvert signal er 0 eller 1, og representerer én bit av minnet



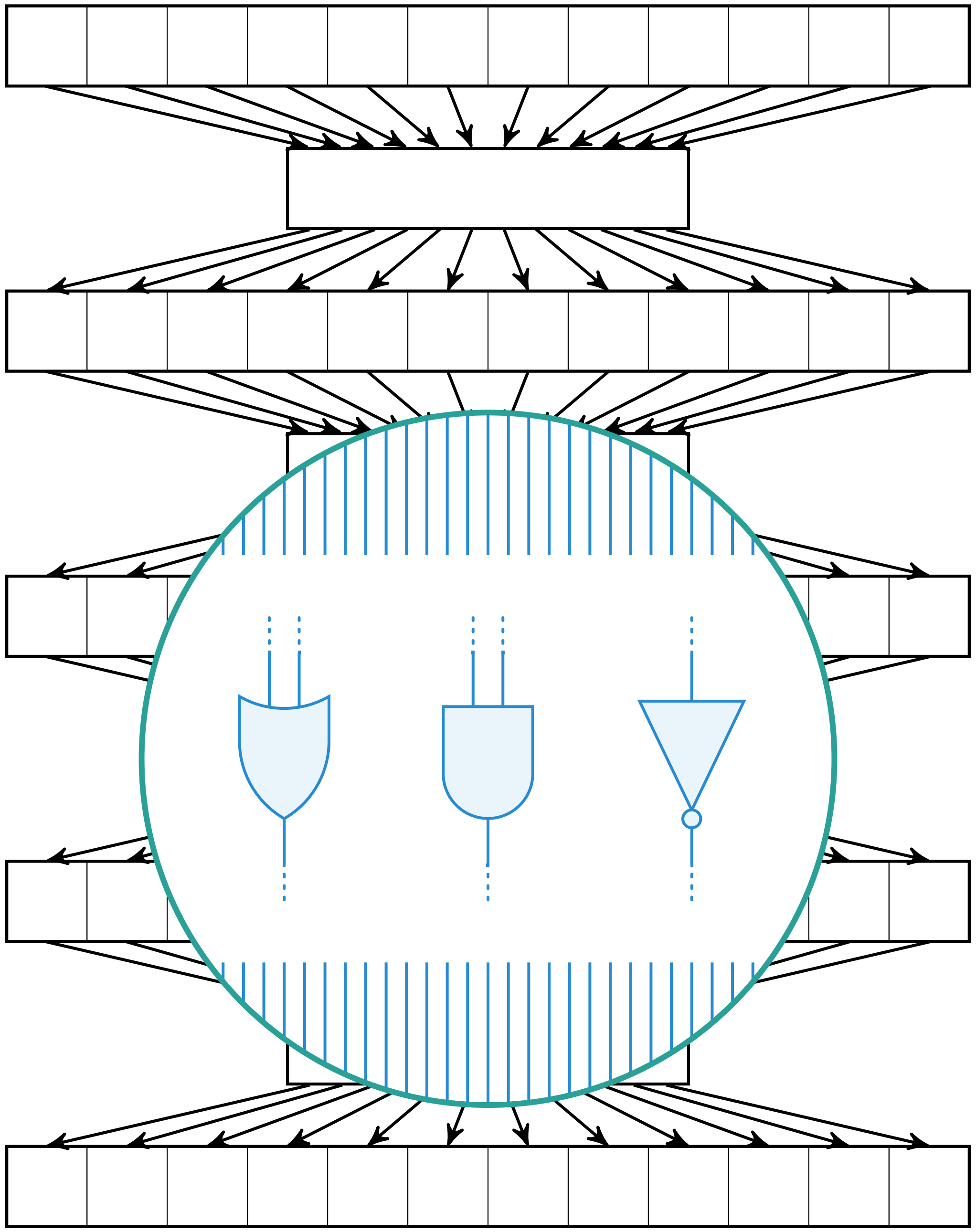
Vi har nok konfigurasjoner til å simulere *worst-case* for A



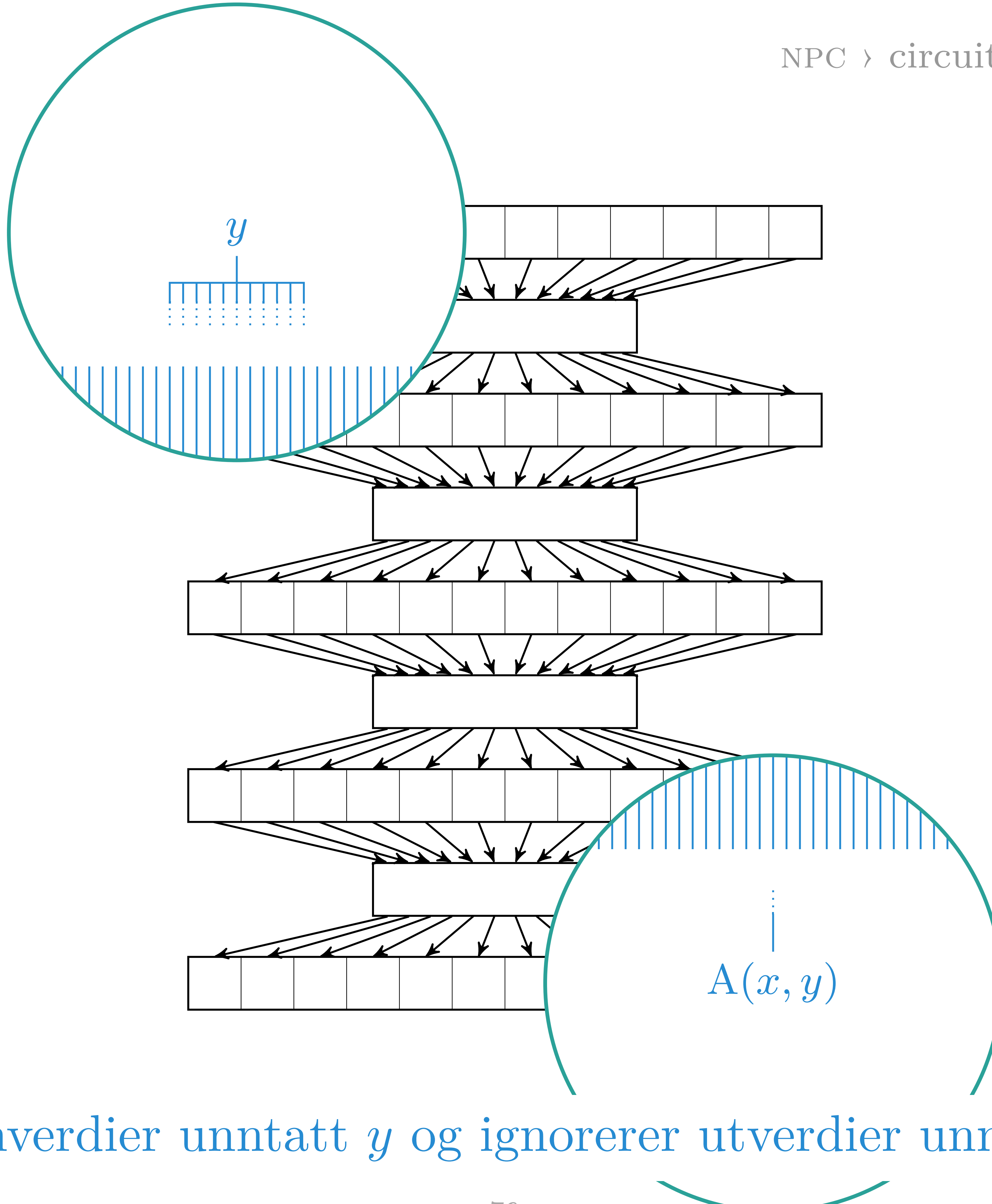
Selve maskina representerer vi med en krets, som vi kopierer opp



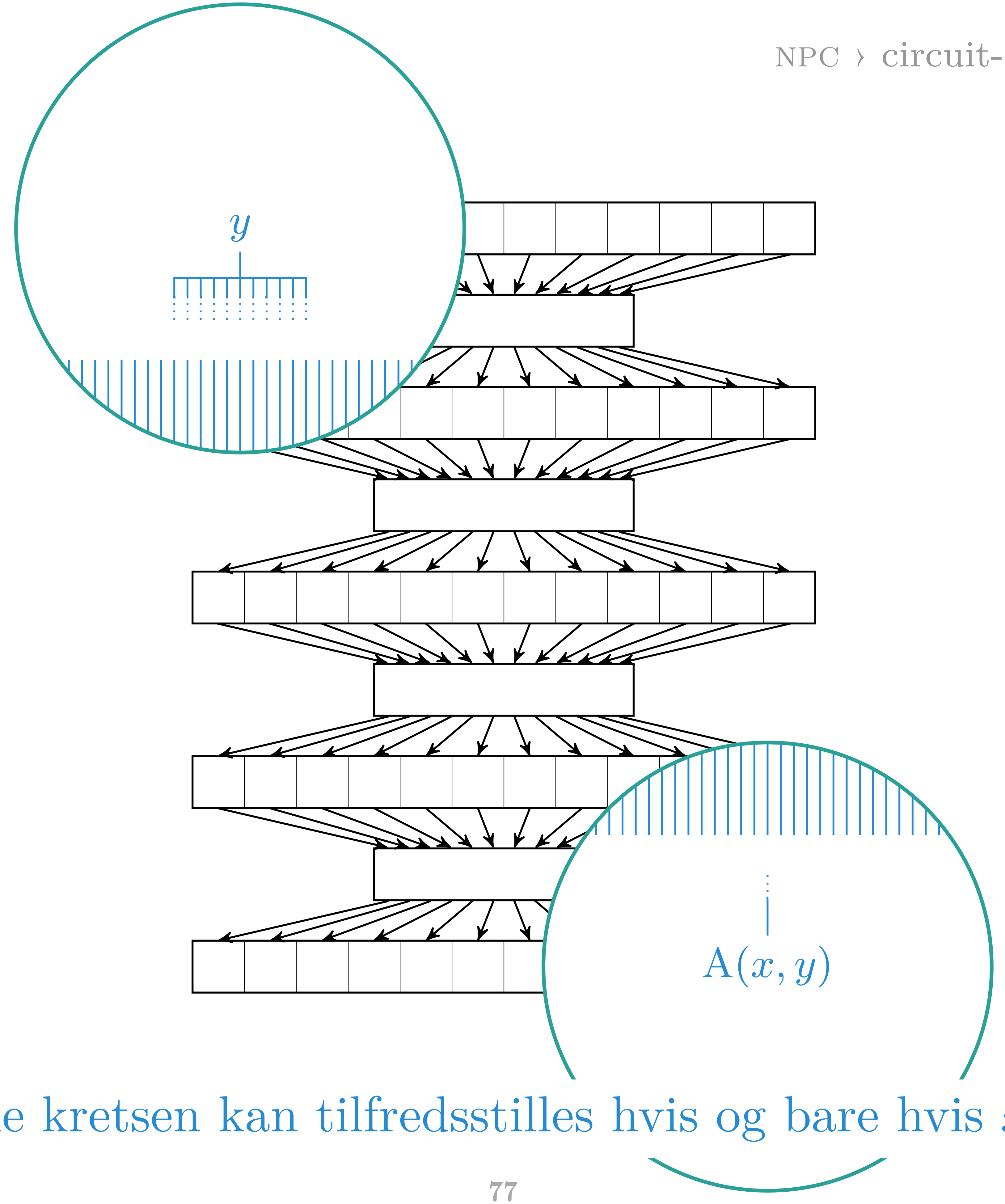
Selve maskina representerer vi med en krets, som vi kopierer opp



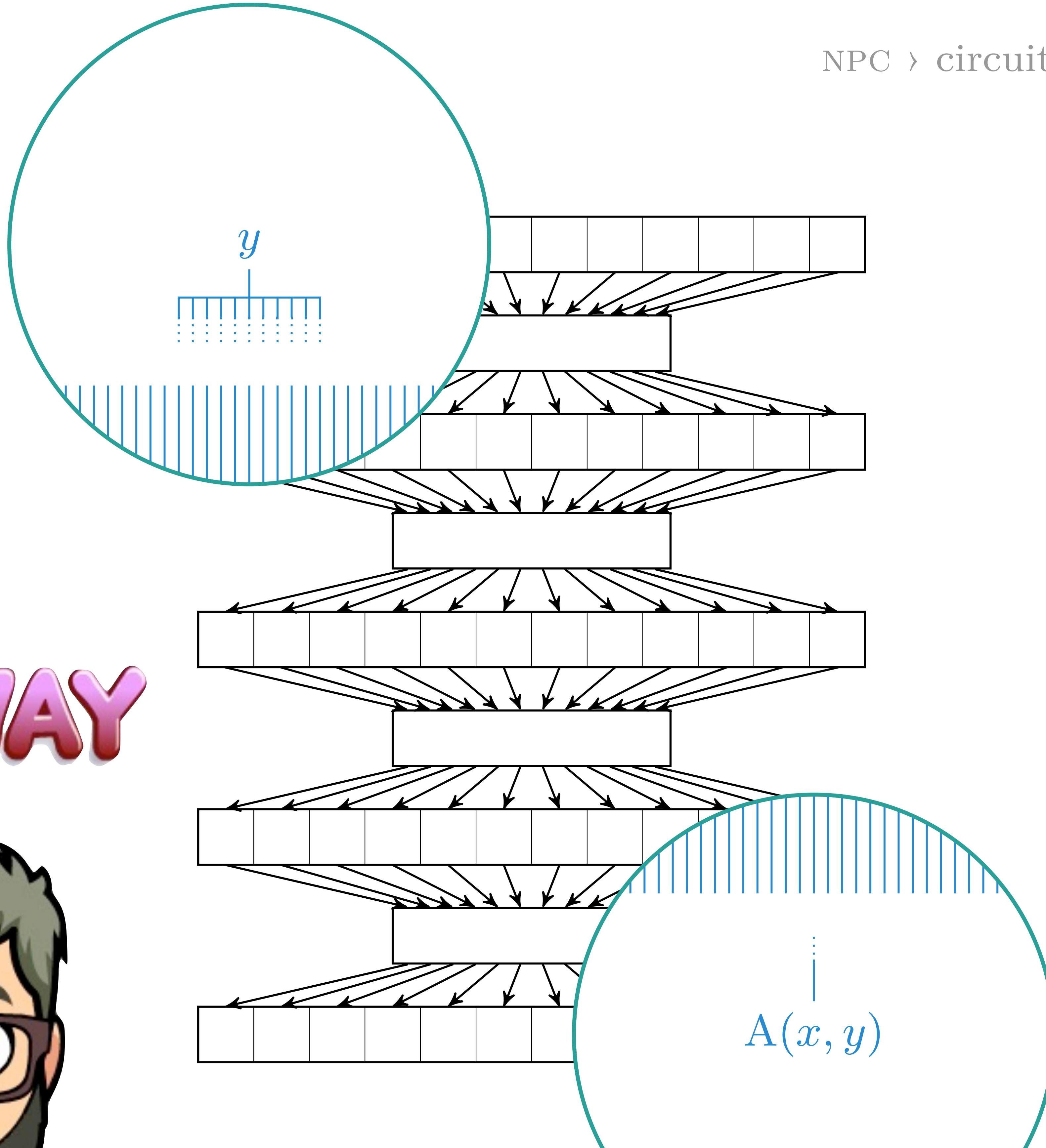
Kretsen simulerer overgangen fra én tilstand til den neste



Vi låser innverdier unntatt y og ignorerer utverdier unntatt $\mathrm{A}(x, y)$



Denne kretsen kan tilfredsstilles hvis og bare hvis $x \in L$



…e kretsen kan tilfredsstilles hvis og bare hvis $x \in L$

› **CIRCUIT-SAT**

- **CIRCUIT-SAT**
  - **Instans:** En krets med logiske porter og én utverdi

- **CIRCUIT-SAT**
  - **Instans:** En krets med logiske porter og én utverdi
  - **Spørsmål:** Kan utverdien bli 1?

- **CIRCUIT-SAT**
  - **Instans:** En krets med logiske porter og én utverdi
  - **Spørsmål:** Kan utverdien bli 1?
- Vi har et vilkårlig språk/problem L $\in$ **NP**

- **CIRCUIT-SAT**
  - **Instans:** En krets med logiske porter og én utverdi
  - **Spørsmål:** Kan utverdien bli 1?
- Vi har et vilkårlig språk/problem $L \in \mathbf{NP}$
- Vi vil redusere dette til CIRCUIT-SAT

- **CIRCUIT-SAT**
  - **Instans:** En krets med logiske porter og én utverdi
  - **Spørsmål:** Kan utverdien bli 1?
- Vi har et vilkårlig språk/problem $\mathrm{L} \in \mathbf{NP}$
- Vi vil redusere dette til CIRCUIT-SAT
- Det eneste vi vet er at $x \in \mathrm{L}$ kan verifiseres i polynomisk tid

- **CIRCUIT-SAT**
  - **Instans:** En krets med logiske porter og én utverdi
  - **Spørsmål:** Kan utverdien bli 1?
- Vi har et vilkårlig språk/problem $L \in \mathbf{NP}$
- Vi vil redusere dette til CIRCUIT-SAT
- Det eneste vi vet er at $x \in L$ kan verifiseres i polynomisk tid
- Vi simulerer trinnene i verifikasjonsalgoritmen A med kretser!

- **CIRCUIT-SAT**
  - **Instans:** En krets med logiske porter og én utverdi
  - **Spørsmål:** Kan utverdien bli 1?
- Vi har et vilkårlig språk/problem L $\in$ **NP**
- Vi vil redusere dette til CIRCUIT-SAT
- Det eneste vi vet er at $x \in$ L kan verifiseres i polynomisk tid
- Vi simulerer trinnene i verifikasjonsalgoritmen A med kretser!
- Spørsmålet blir: Kan A (for et eller annet sertifikat) svare 1?

$\bullet \longmapsto \bullet$

$$x \in \{0, 1\}^*$$

Er x med i språket L? $\iff$ Kan utverdien bli 1?

L er i **NP**, så . . .

Det finnes en pol. alg. A, som er slik at

- $x \in \mathrm{L}$

nøyaktig når minst én $y \in \{0,1\}^*$ gir

- $\mathrm{A}(x, y) = 1\,,$

der $|y| = \mathrm{O}(|x|^c)$, for en eller annen c.

Er x med i språket L? $\Longleftrightarrow$ Kan utverdien bli 1?

L er i **NP**, så . . .

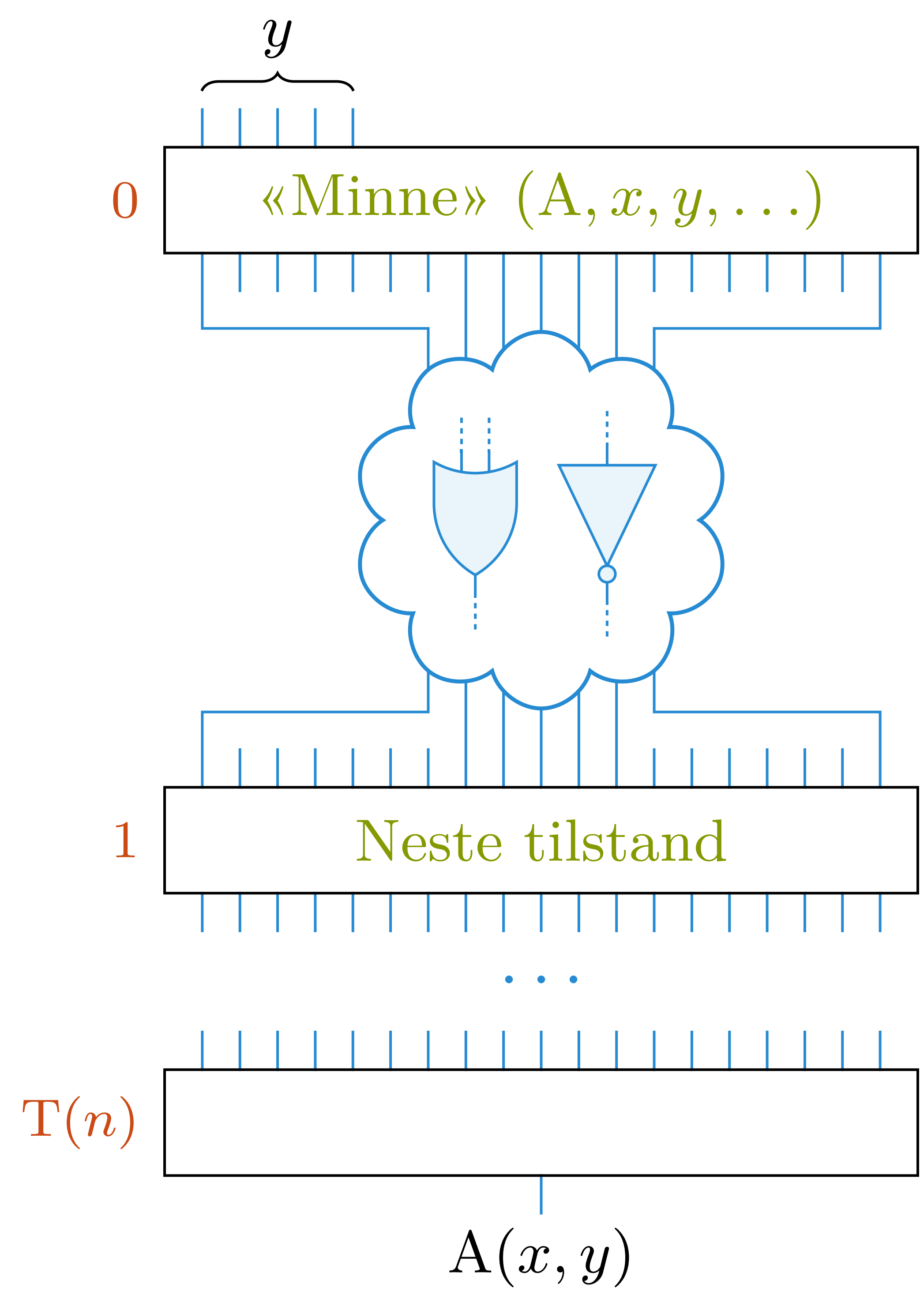
Det finnes en pol. alg. A, som er slik at

› $x \in \mathrm{L}$

nøyaktig når minst én $y \in \{0, 1\}^*$ gir

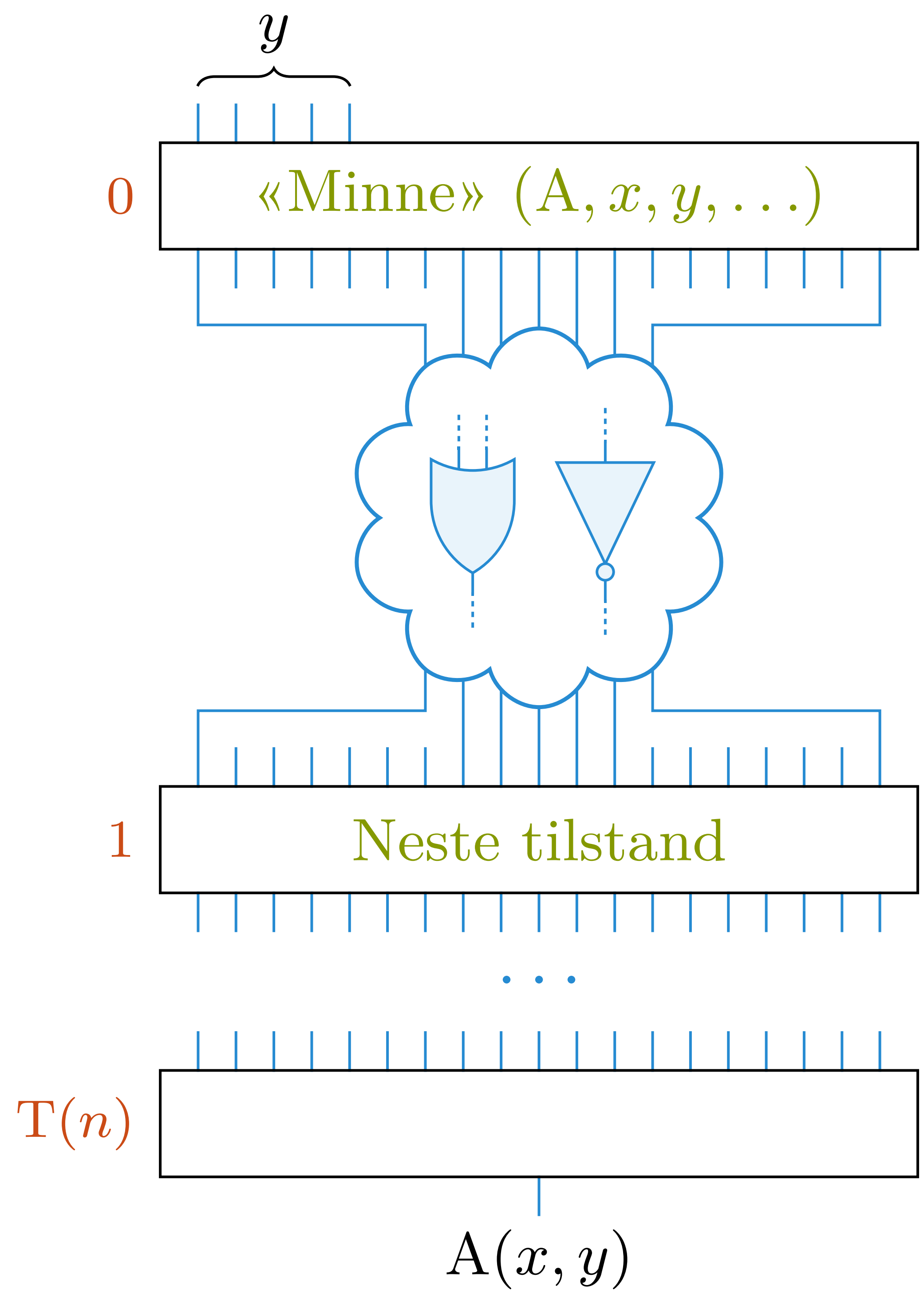
› $\mathrm{A}(x, y) = 1$,

der $|y| = \mathrm{O}(|x|^c)$, for en eller annen c.



Er x med i språket L? $\Longleftrightarrow$ Kan utverdien bli 1?

$$x \in \{0,1\}^*$$



Er x med i språket L? $\iff$ Kan utverdien bli 1?

3:9

SAT

› SAT

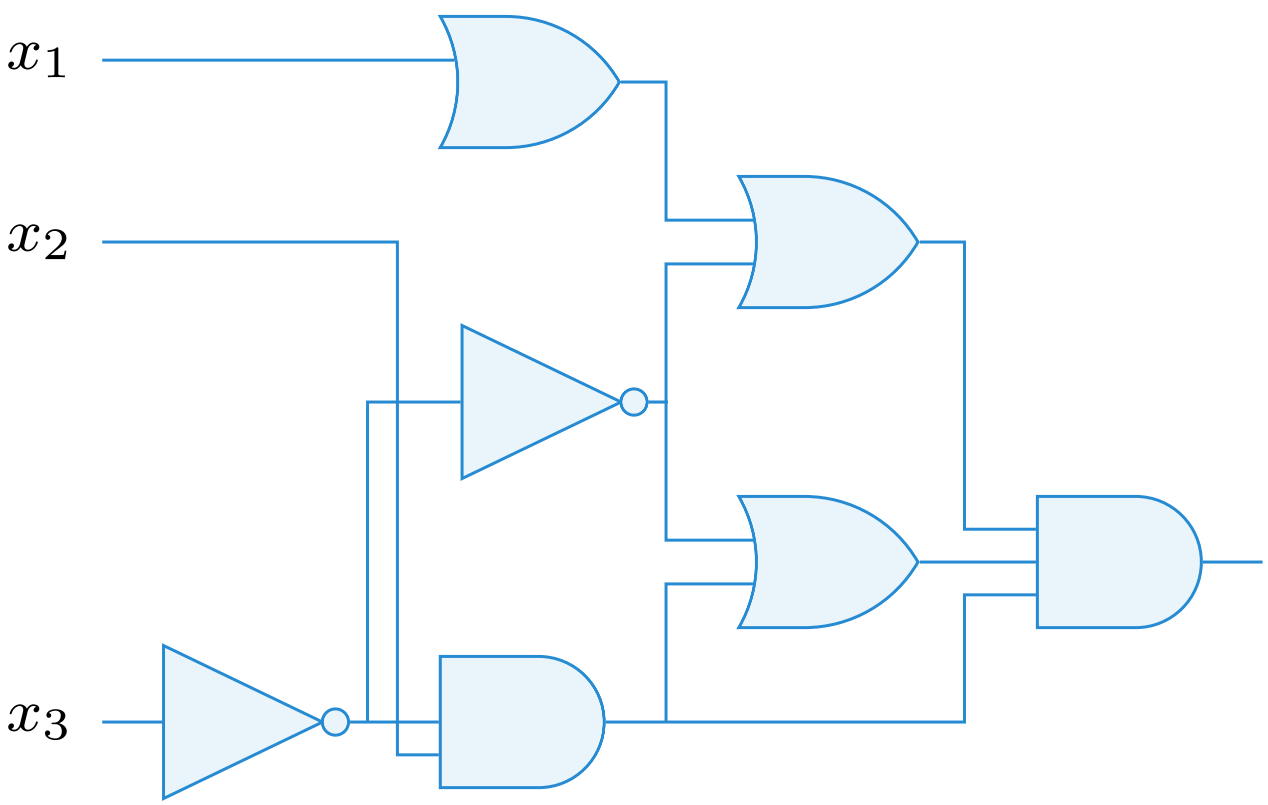
- **SAT**
  - **Instans:** En logisk formel

- **SAT**
  - **Instans:** En logisk formel
  - **Spørsmål:** Kan formelen være sann?

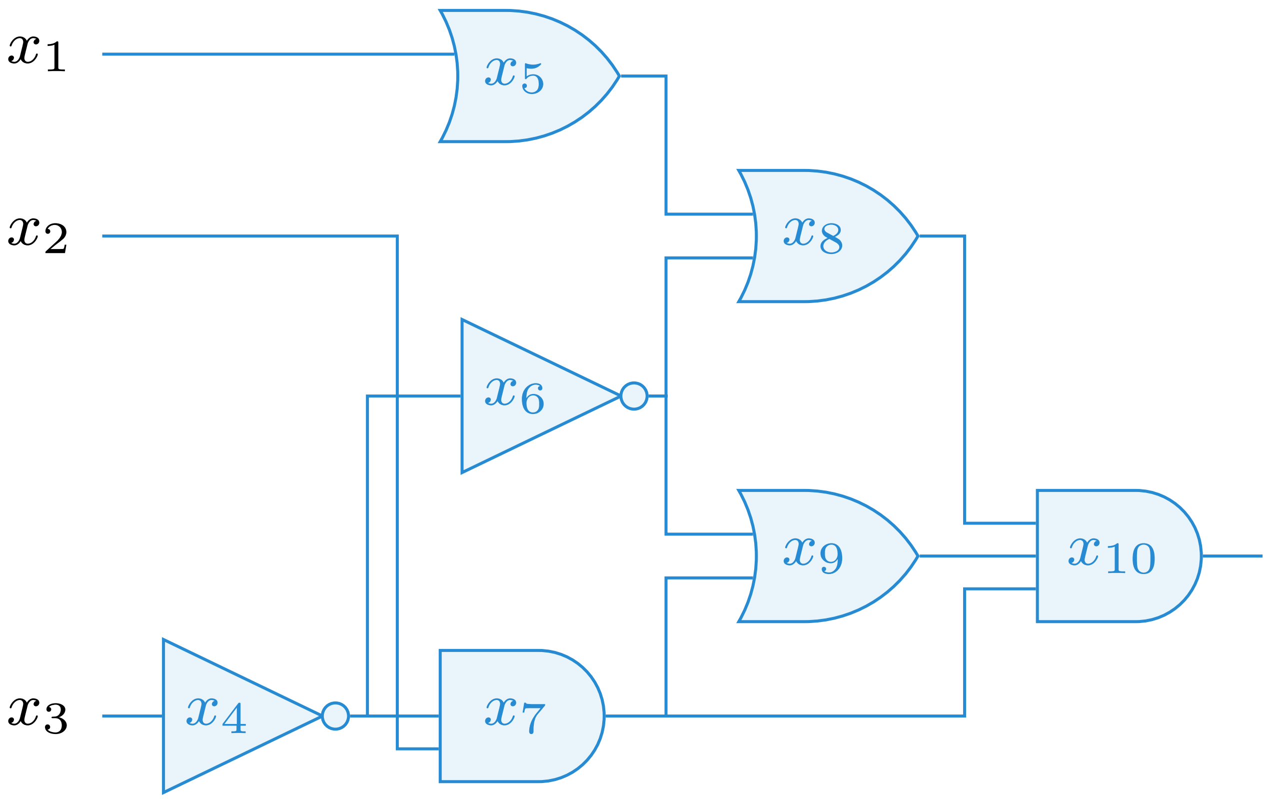
- **SAT**
  - **Instans:** En logisk formel
  - **Spørsmål:** Kan formelen være sann?
- Direkte oversettelse av logisk krets?

- **SAT**
  - **Instans:** En logisk formel
  - **Spørsmål:** Kan formelen være sann?
- Direkte oversettelse av logisk krets?
- Kan gi eksponentielt stor formel!

$\phi =$

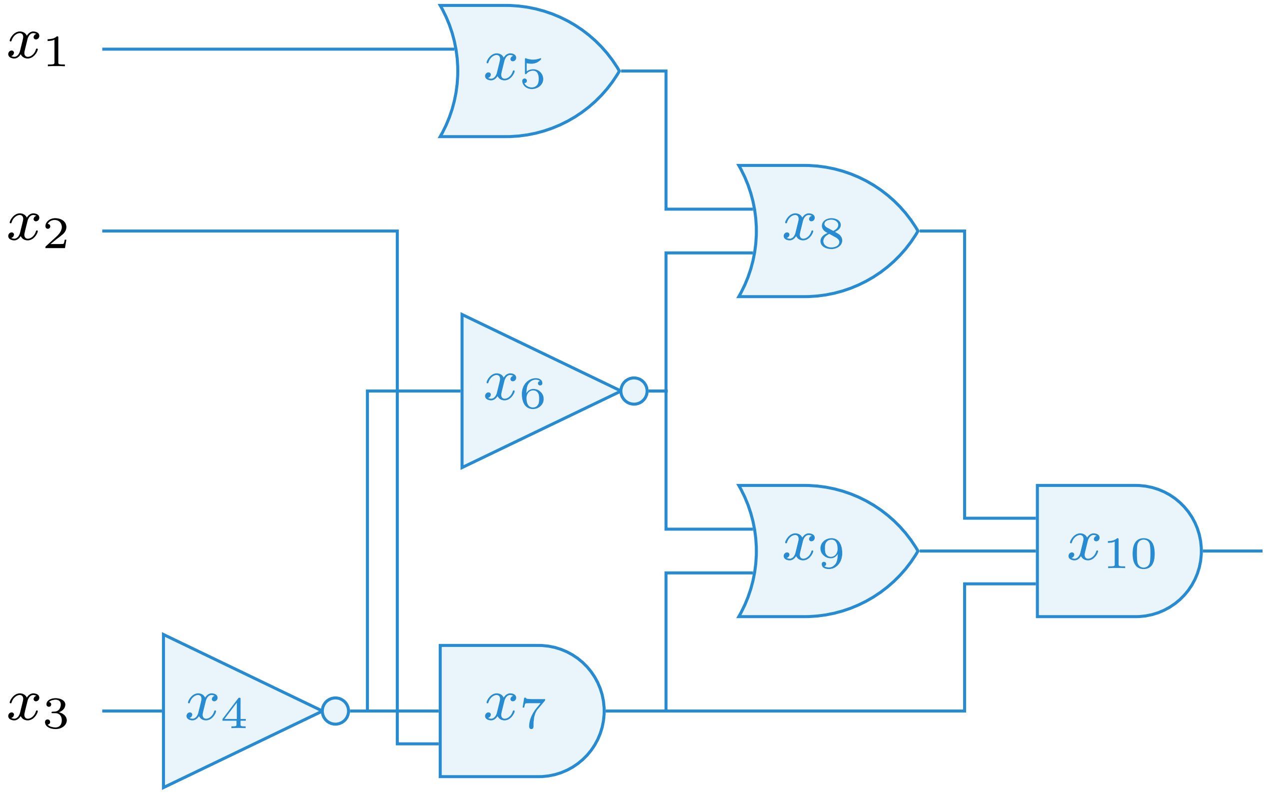


Kan utverdien bli 1? $\iff$ Kan ϕ være sann?



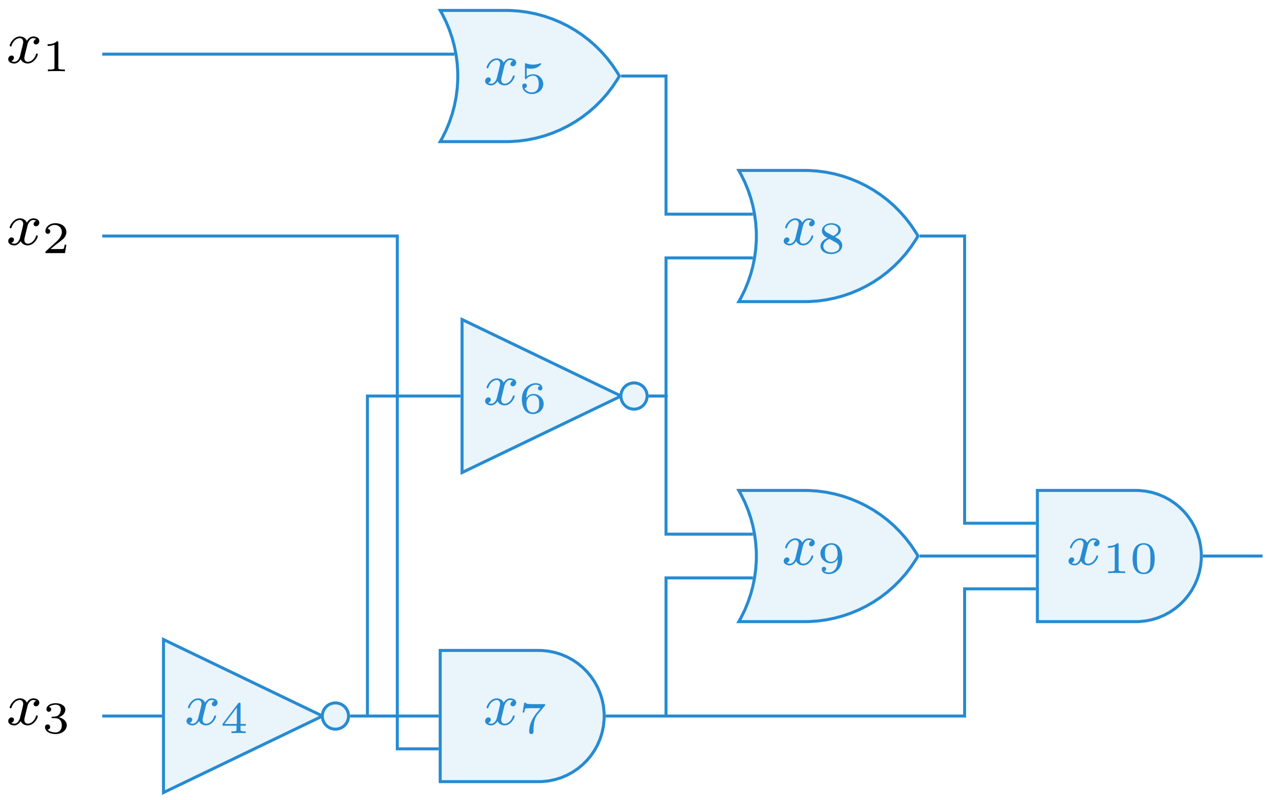
$\phi =$

Kan utverdien bli 1? $\Longleftrightarrow$ Kan ϕ være sann?



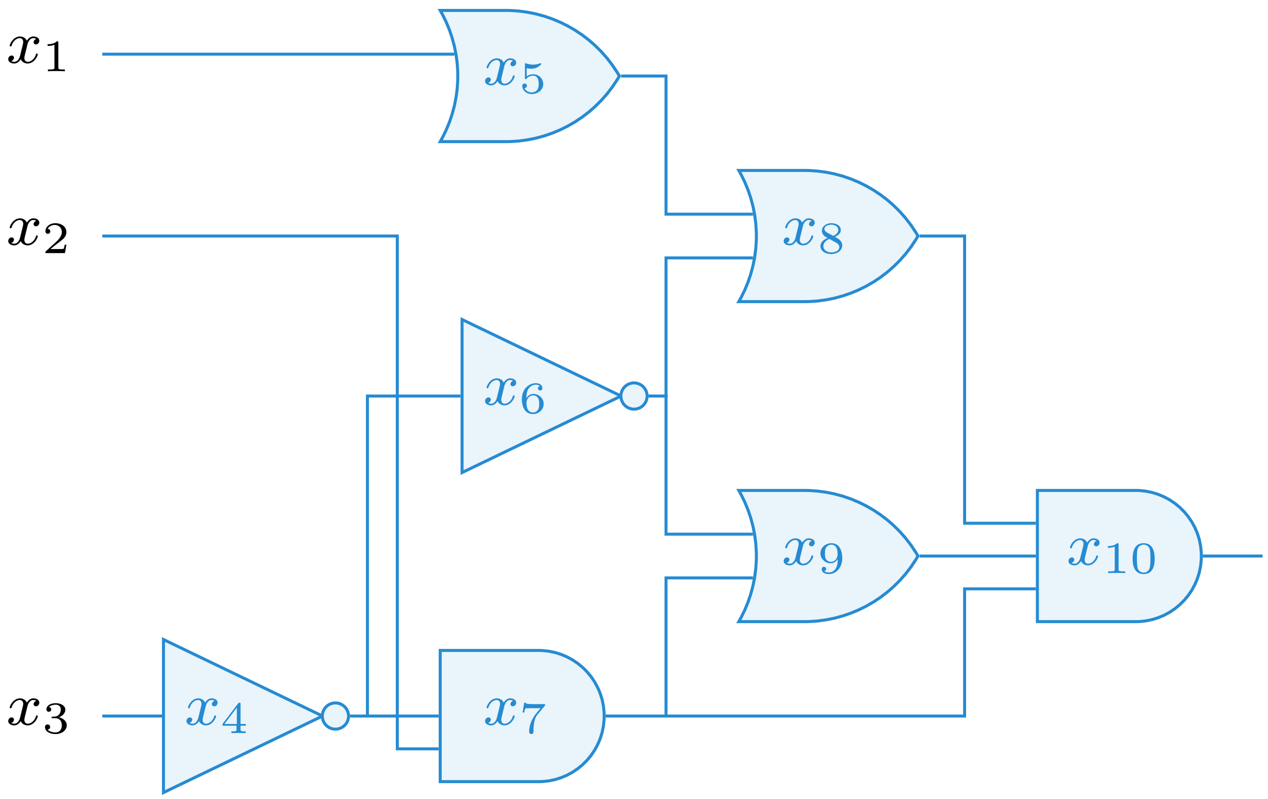
$$\phi = x_{10}$$

Kan utverdien bli 1? $\iff$ Kan ϕ være sann?



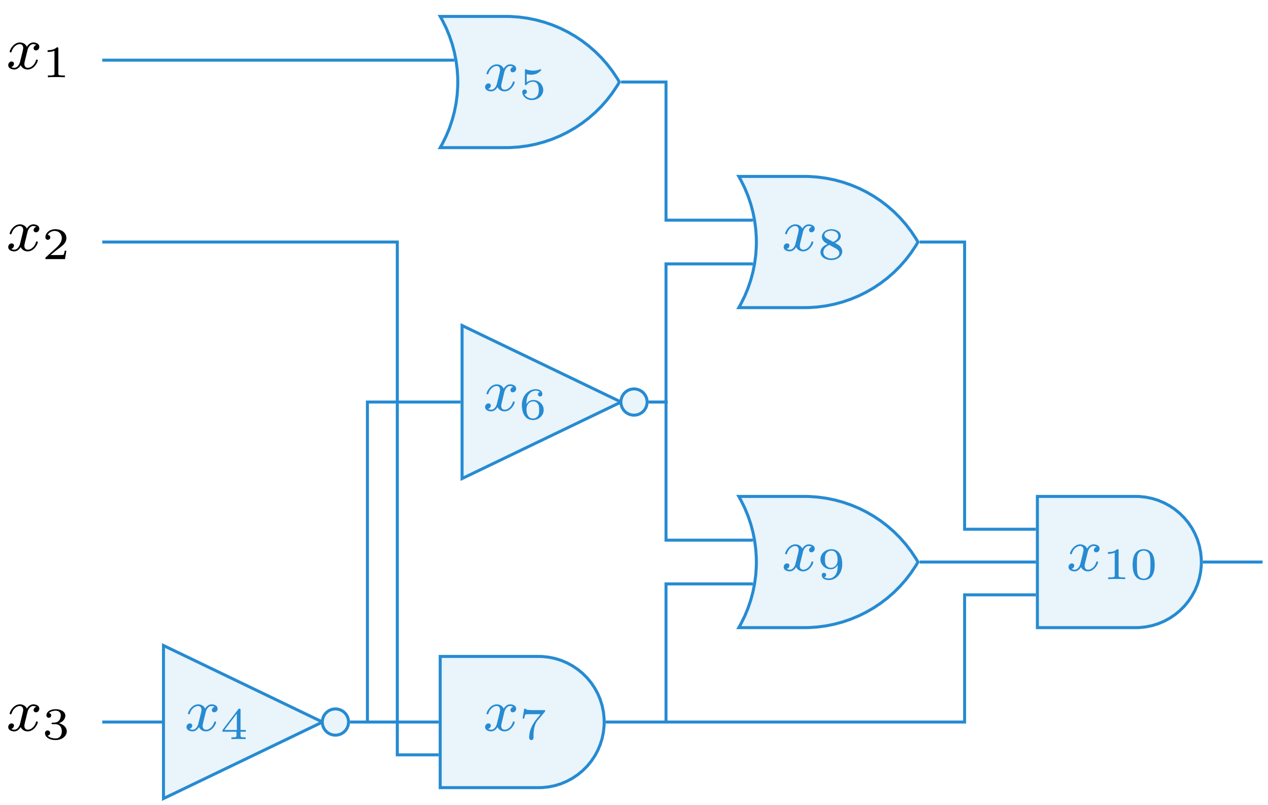
$$
\begin{aligned}
\phi = x_{10} \;&\wedge\; (x_4 \leftrightarrow \quad) \\
&\wedge\; (x_5 \leftrightarrow \quad) \\
&\wedge\; (x_6 \leftrightarrow \quad) \\
&\wedge\; (x_7 \leftrightarrow \quad) \\
&\wedge\; (x_8 \leftrightarrow \quad) \\
&\wedge\; (x_9 \leftrightarrow \quad) \\
&\wedge\; (x_{10} \leftrightarrow \quad)
\end{aligned}
$$

Kan utverdien bli 1? $\Longleftrightarrow$ Kan ϕ være sann?



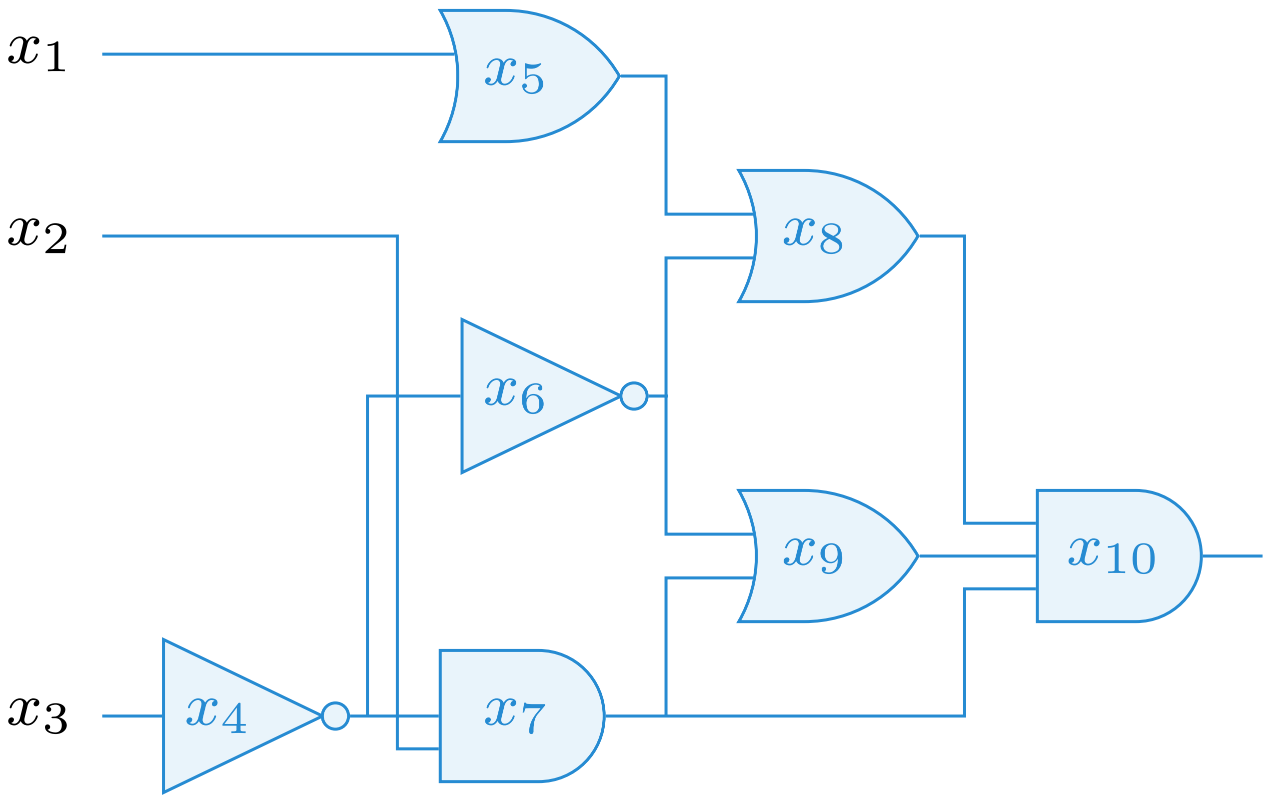
$$
\begin{aligned}
\phi = x_{10} \;&\wedge\; (x_4 \leftrightarrow \neg x_3) \\
&\wedge\; (x_5 \leftrightarrow \qquad\qquad) \\
&\wedge\; (x_6 \leftrightarrow \qquad) \\
&\wedge\; (x_7 \leftrightarrow \qquad\qquad\qquad\qquad) \\
&\wedge\; (x_8 \leftrightarrow \qquad\qquad) \\
&\wedge\; (x_9 \leftrightarrow \qquad\qquad) \\
&\wedge\; (x_{10} \leftrightarrow \qquad\qquad\qquad\qquad)
\end{aligned}
$$

Kan utverdien bli 1? $\Longleftrightarrow$ Kan ϕ være sann?



$$
\begin{aligned}
\phi = x_{10} \;&\wedge\; (x_4 \leftrightarrow \neg x_3)\\
&\wedge\; (x_5 \leftrightarrow (x_1 \vee x_2))\\
&\wedge\; (x_6 \leftrightarrow \qquad)\\
&\wedge\; (x_7 \leftrightarrow \qquad\qquad\qquad)\\
&\wedge\; (x_8 \leftrightarrow \qquad\qquad)\\
&\wedge\; (x_9 \leftrightarrow \qquad\qquad)\\
&\wedge\; (x_{10} \leftrightarrow \qquad\qquad\qquad)
\end{aligned}
$$

Kan utverdien bli 1? $\Longleftrightarrow$ Kan ϕ være sann?



$$
\begin{aligned}
\phi = x_{10} \;&\wedge\; (x_4 \leftrightarrow \neg x_3) \\
&\wedge\; (x_5 \leftrightarrow (x_1 \vee x_2)) \\
&\wedge\; (x_6 \leftrightarrow \neg x_4) \\
&\wedge\; (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
&\wedge\; (x_8 \leftrightarrow (x_5 \vee x_6)) \\
&\wedge\; (x_9 \leftrightarrow (x_6 \vee x_7)) \\
&\wedge\; (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))
\end{aligned}
$$

Kan utverdien bli 1? $\iff$ Kan ϕ være sann?

4:9

3-CNF-SAT

› **3-CNF-SAT**

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots \land (\neg x_7 \lor x_8 \lor x_9)$

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?
- Vi kan bruke ca. samme reduksjon, på syntakstreet til ϕ!

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?
- Vi kan bruke ca. samme reduksjon, på syntakstreet til ϕ!
- Vi får da en formel ϕ' av pol. størrelse

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?
- Vi kan bruke ca. samme reduksjon, på syntakstreet til ϕ!
- Vi får da en formel ϕ' av pol. størrelse
- ϕ' er en konjunksjon av termer, hver med maks 3 literaler

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?
- Vi kan bruke ca. samme reduksjon, på syntakstreet til ϕ!
- Vi får da en formel ϕ' av pol. størrelse
- ϕ' er en konjunksjon av termer, hver med maks 3 literaler
  - Dvs.: de to argumentene, samt resultatet av operatoren

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?
- Vi kan bruke ca. samme reduksjon, på syntakstreet til ϕ!
- Vi får da en formel ϕ' av pol. størrelse
- ϕ' er en konjunksjon av termer, hver med maks 3 literaler
  - Dvs.: de to argumentene, samt resultatet av operatoren
- Hver term gjøres om til CNF vha. en sannhetstabell

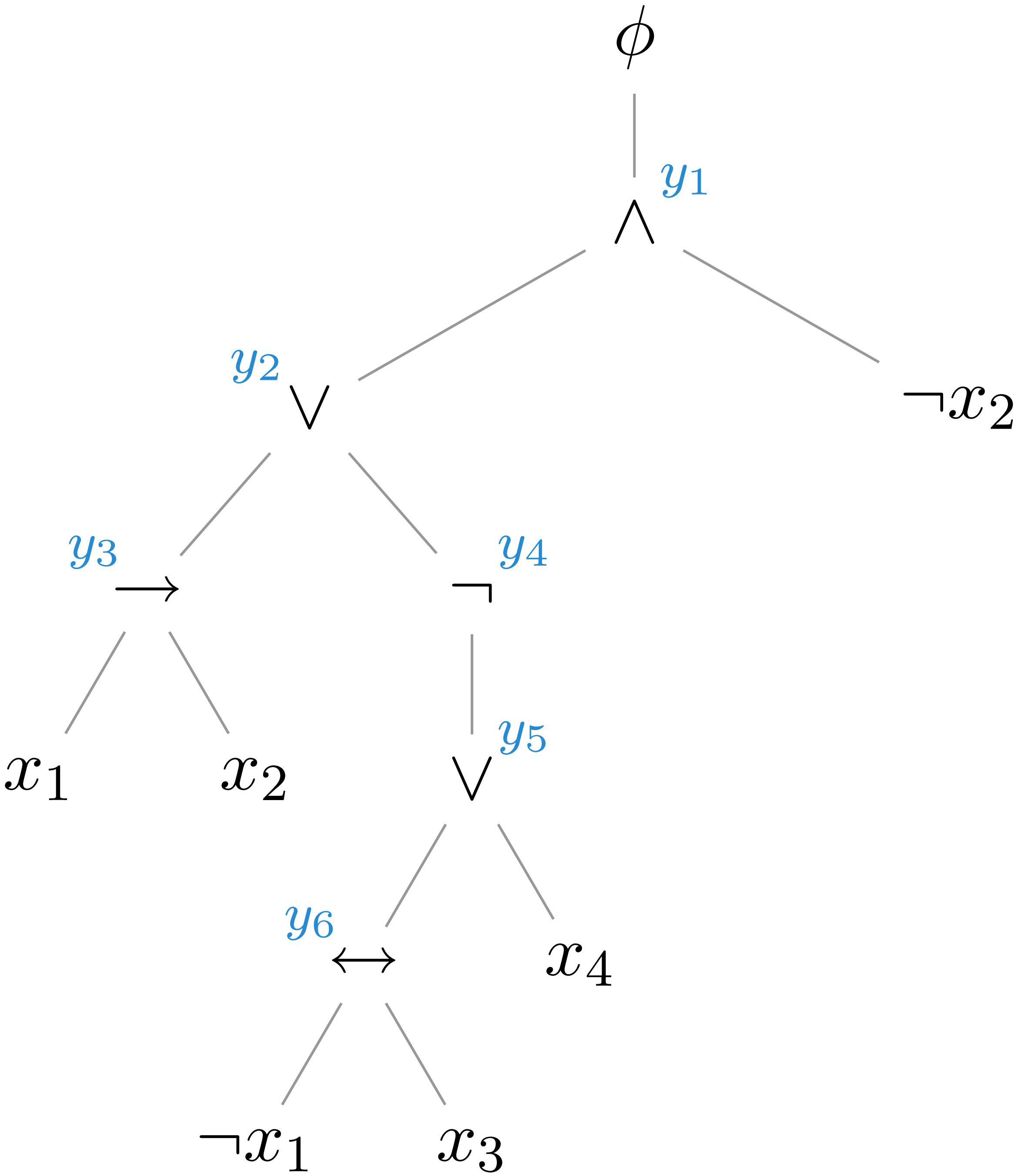
- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?
- Vi kan bruke ca. samme reduksjon, på syntakstreet til ϕ!
- Vi får da en formel ϕ' av pol. størrelse
- ϕ' er en konjunksjon av termer, hver med maks 3 literaler
  - Dvs.: de to argumentene, samt resultatet av operatoren
- Hver term gjøres om til CNF vha. en sannhetstabell
- $(x \vee y)$ gjøres om til $(x \vee y \vee z) \wedge (x \vee y \vee \neg z)$

- **3-CNF-SAT**
  - **Instans:** En logisk formel på 3-CNF-form
    F.eks.: $\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge \ldots \wedge (\neg x_7 \vee x_8 \vee x_9)$
  - **Spørsmål:** Kan formelen være sann?
- Vi kan bruke ca. samme reduksjon, på syntakstreet til ϕ!
- Vi får da en formel ϕ' av pol. størrelse
- ϕ' er en konjunksjon av termer, hver med maks 3 literaler
  - Dvs.: de to argumentene, samt resultatet av operatoren
- Hver term gjøres om til CNF vha. en sannhetstabell
- $(x \vee y)$ gjøres om til $(x \vee y \vee z) \wedge (x \vee y \vee \neg z)$
- Tilsv. blir (x) til fire nye termer

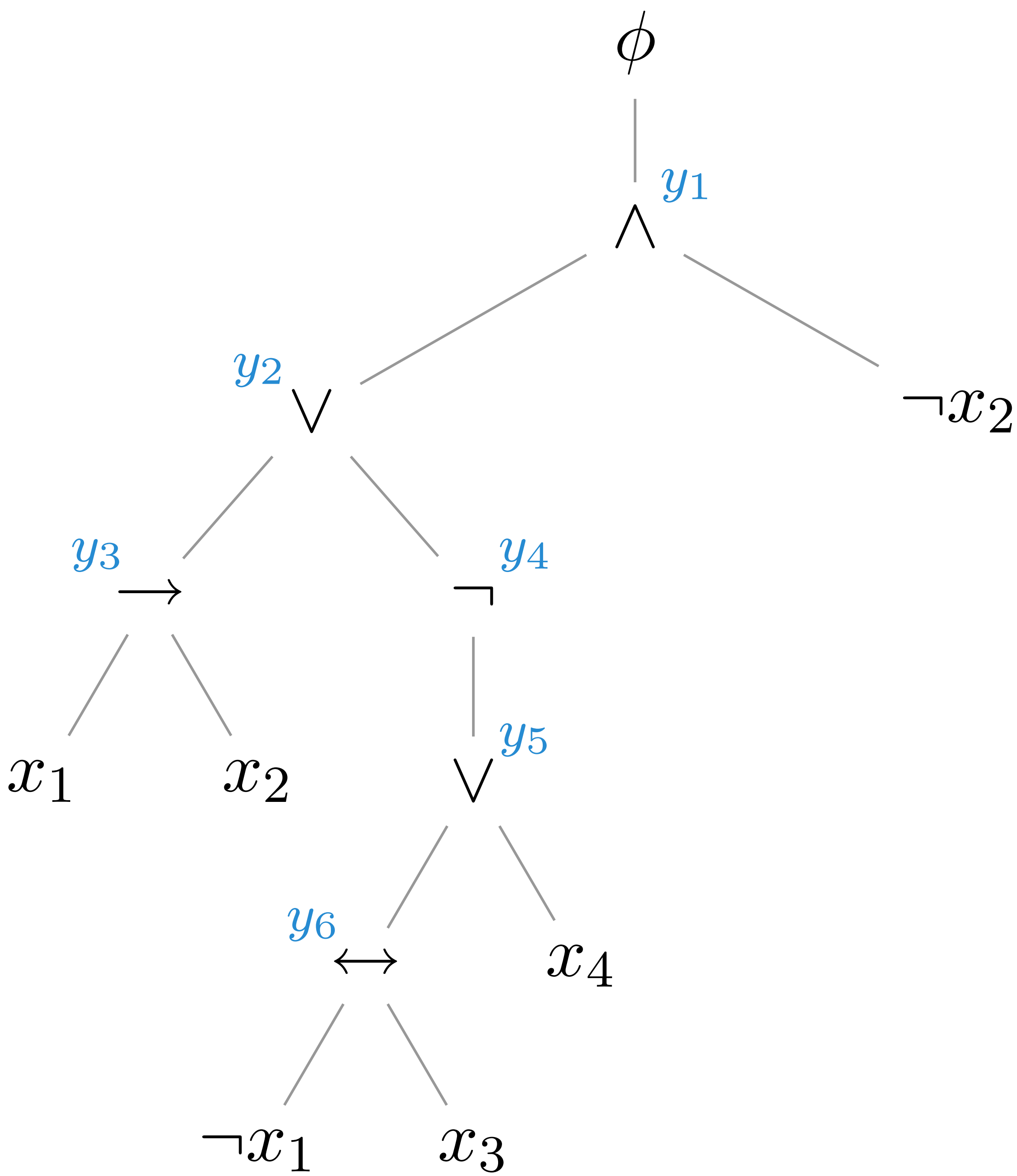
$$\phi = ((x_1 \to x_2) \vee$$

$$\neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

Kan ϕ være sann? $\iff$ Kan ϕ''' være sann?

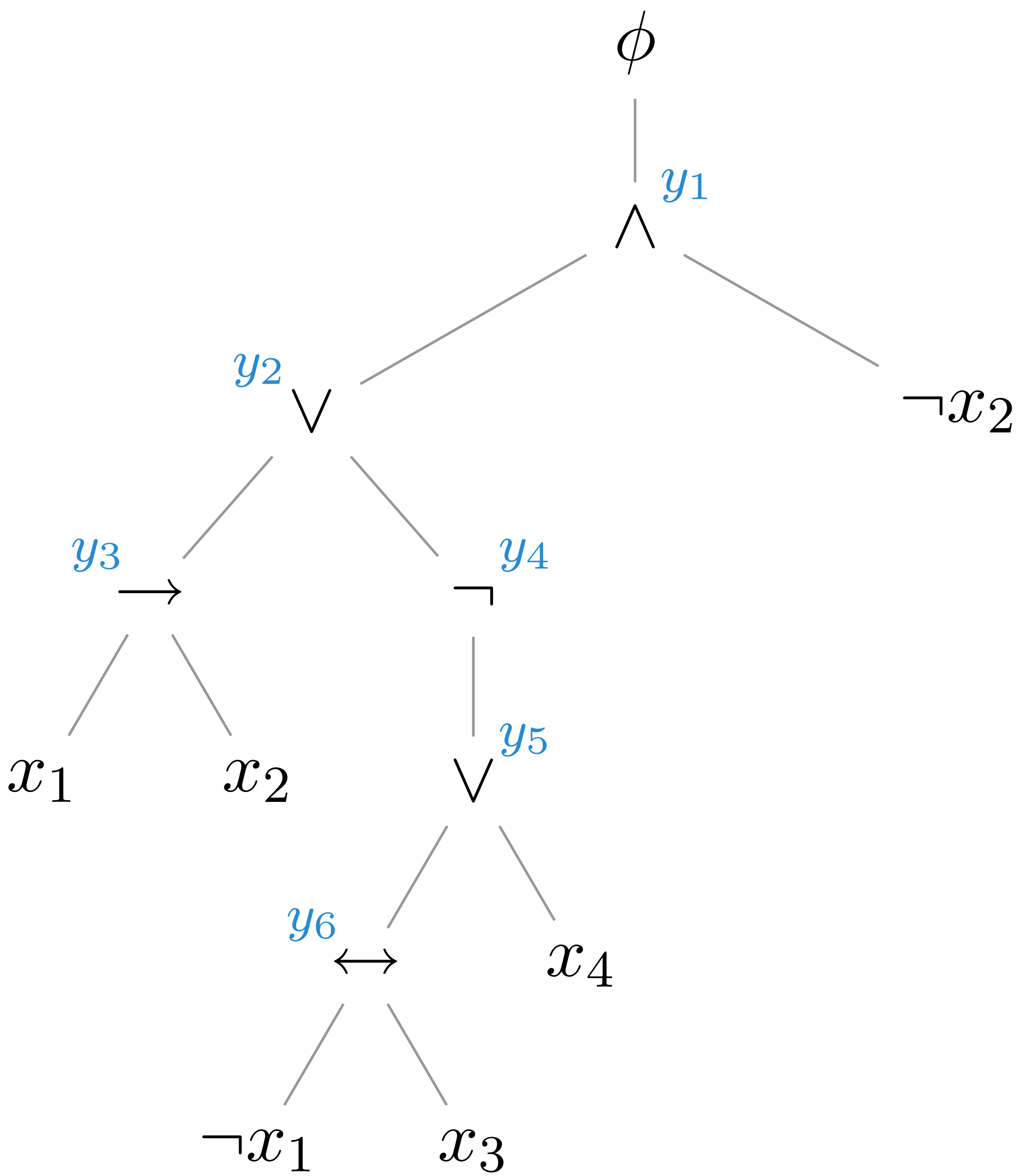


Kan ϕ være sann? $\iff$ Kan ϕ''' være sann?



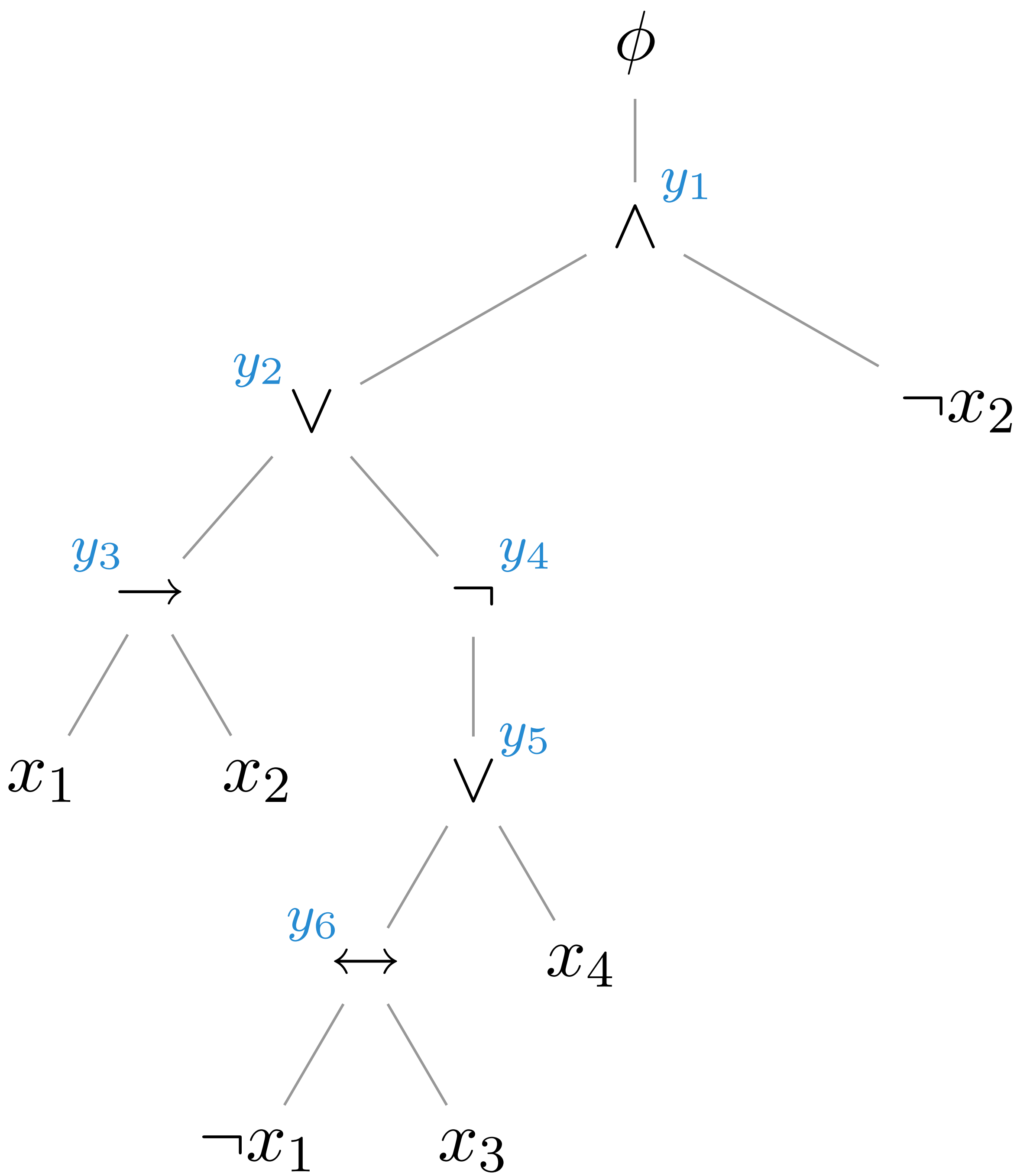
$$
\begin{aligned}
\phi' \quad = \quad & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
& \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\
& \wedge (y_3 \leftrightarrow (x_1 \longrightarrow x_2)) \\
& \wedge (y_4 \leftrightarrow \neg y_5) \\
& \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\
& \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))
\end{aligned}
$$

Kan ϕ være sann? $\Longleftrightarrow$ Kan ϕ''' være sann?



$$
\begin{aligned}
\phi' \quad &= \quad y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
& \qquad \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\
& \qquad \wedge (y_3 \leftrightarrow (x_1 \longrightarrow x_2)) \\
& \qquad \wedge (y_4 \leftrightarrow \neg y_5) \\
& \qquad \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\
& \qquad \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)) \\
\phi'' \quad &= \quad \text{CNF, vha. sannhetstabeller}
\end{aligned}
$$

Kan ϕ være sann? $\iff$ Kan ϕ''' være sann?



$$
\begin{aligned}
\phi' &= y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
&\quad \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\
&\quad \wedge (y_3 \leftrightarrow (x_1 \longrightarrow x_2)) \\
&\quad \wedge (y_4 \leftrightarrow \neg y_5) \\
&\quad \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\
&\quad \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)) \\
\phi'' &= \text{CNF, vha. sannhetstabeller} \\
\phi''' &= \text{3-CNF, vha. dummy-variable}
\end{aligned}
$$

Kan ϕ være sann? $\iff$ Kan ϕ''' være sann?

5:8

CLIQUE

› **CLIQUE**

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen
- Ingen kanter mellom noder fra samme term

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen
- Ingen kanter mellom noder fra samme term
- Ellers: Kanter mellom literaler som kan være sanne samtidig

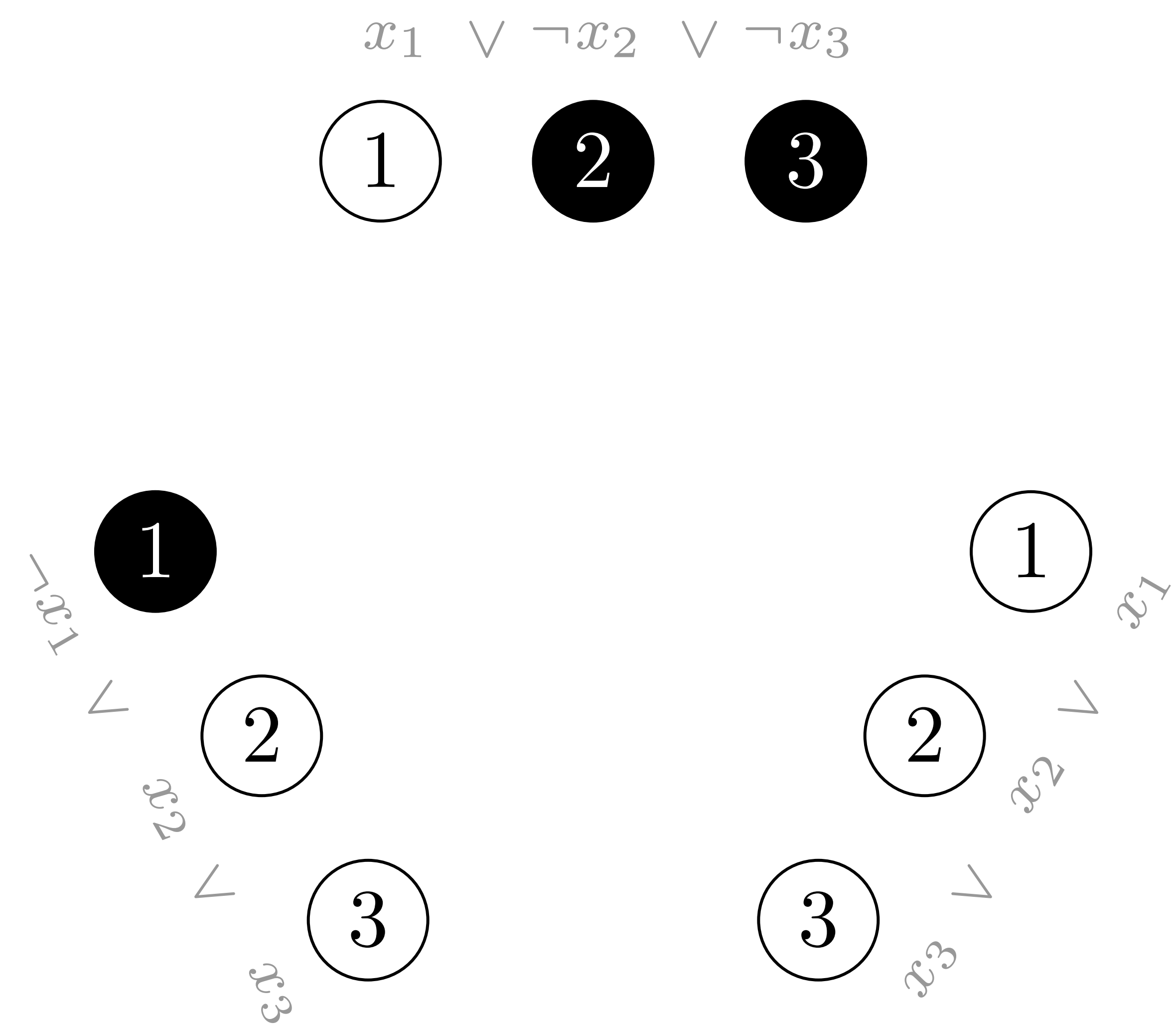
- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen
- Ingen kanter mellom noder fra samme term
- Ellers: Kanter mellom literaler som kan være sanne samtidig
- La k være antall termer

$$\bullet \longmapsto \bullet$$

$$\begin{aligned}\phi = (\ x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\ (\neg x_1 \vee x_2 \vee x_3) \wedge \\ (\ x_1 \vee x_2 \vee x_3)\end{aligned}$$

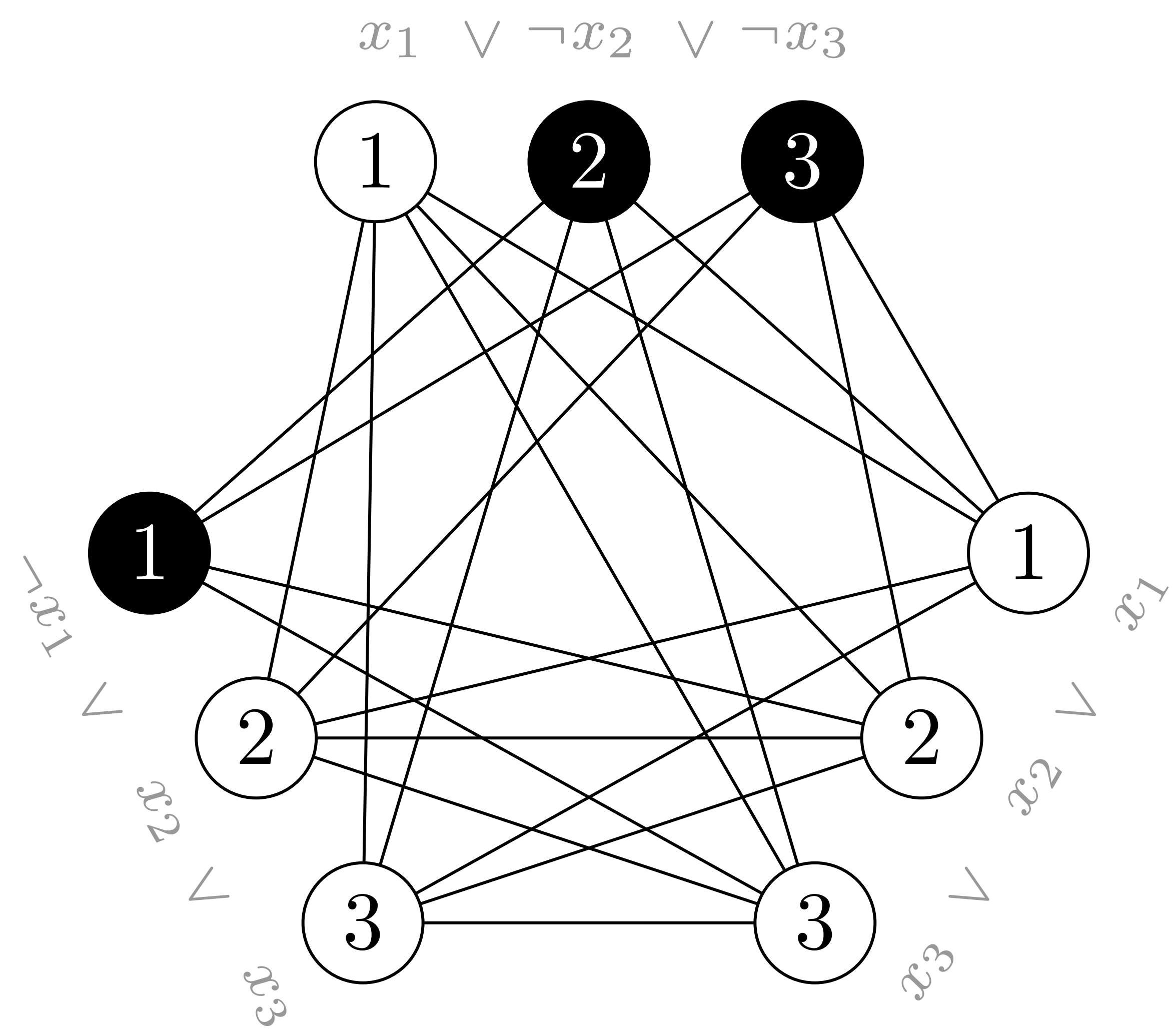
Kan ϕ være sann? $\Longleftrightarrow$ Finnes en k-klikk?

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



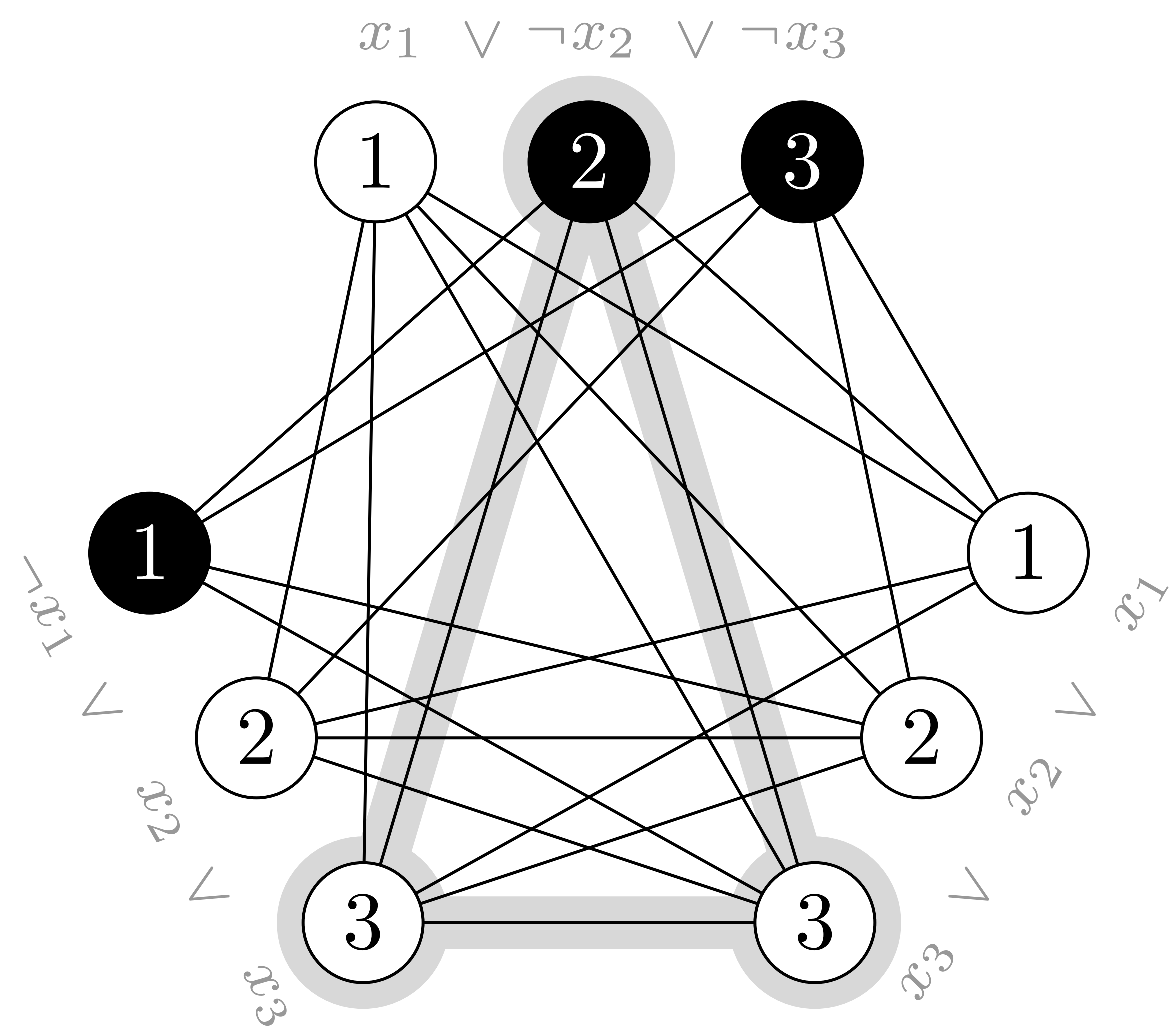
Kan ϕ være sann? $\iff$ Finnes en k-klikk?

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



Kan ϕ være sann? $\Longleftrightarrow$ Finnes en k-klikk?

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



Tilsv. $x_1, x_2, x_3 = -, 0, 1$

Kan ϕ være sann? $\Longleftrightarrow$ Finnes en k-klikk?

6:9

VERTEX-COVER

› VERTEX-COVER

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene
- En klikk er en komplett delgraf

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene
- En klikk er en komplett delgraf
- Tilsvarer en uavhengig mengde (kantfri delgraf) i komplementet $\bar{G} = (V, \bar{E})$

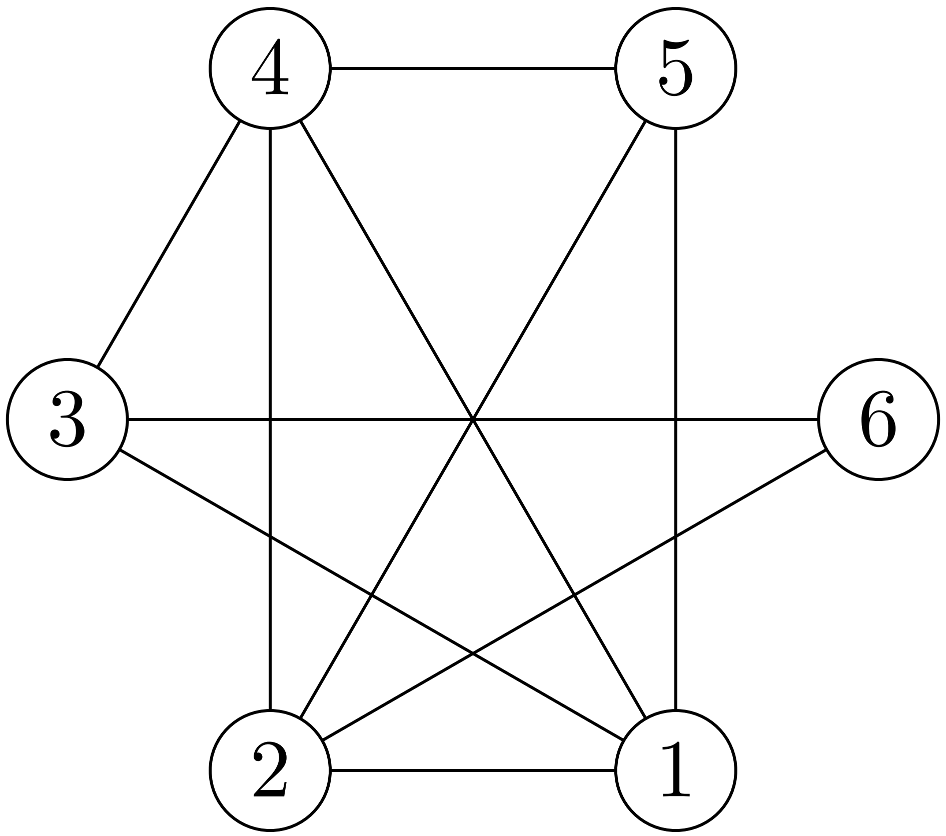
- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene
- En klikk er en komplett delgraf
- Tilsvarer en uavhengig mengde (kantfri delgraf) i komplementet $\bar{\mathrm{G}} = (\mathrm{V}, \bar{\mathrm{E}})$
- Nodene utenfor en uavhengig mengde utgjør et nodedekke

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene
- En klikk er en komplett delgraf
- Tilsvarer en uavhengig mengde (kantfri delgraf) i komplementet $\bar{G} = (V, \bar{E})$
- Nodene utenfor en uavhengig mengde utgjør et nodedekke
- Hvis G har en k-klikk . . .

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene
- En klikk er en komplett delgraf
- Tilsvarer en uavhengig mengde (kantfri delgraf) i komplementet $\bar{\mathrm{G}} = (\mathrm{V}, \bar{\mathrm{E}})$
- Nodene utenfor en uavhengig mengde utgjør et nodedekke
- Hvis G har en k-klikk …
  - … så har $\bar{\mathrm{G}} = (\mathrm{V}, \bar{\mathrm{E}})$ en uavh. mengde med k noder …

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene
- En klikk er en komplett delgraf
- Tilsvarer en uavhengig mengde (kantfri delgraf) i komplementet $\bar{G} = (V, \bar{E})$
- Nodene utenfor en uavhengig mengde utgjør et nodedekke
- Hvis G har en k-klikk …
  - … så har $\bar{G} = (V, \bar{E})$ en uavh. mengde med k noder …
  - … og dermed også et $(|V| - k)$-nodedekke

- **VERTEX-COVER**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en et nodedekke med k noder?
    Dvs., k noder som tilsammen ligger inntil alle kantene
- En klikk er en komplett delgraf
- Tilsvarer en uavhengig mengde (kantfri delgraf) i komplementet $\bar{G} = (V, \bar{E})$
- Nodene utenfor en uavhengig mengde utgjør et nodedekke
- Hvis G har en k-klikk …
  - … så har $\bar{G} = (V, \bar{E})$ en uavh. mengde med k noder …
  - … og dermed også et $(|V|-k)$-nodedekke
- Samme resonnement holder i motsatt retning

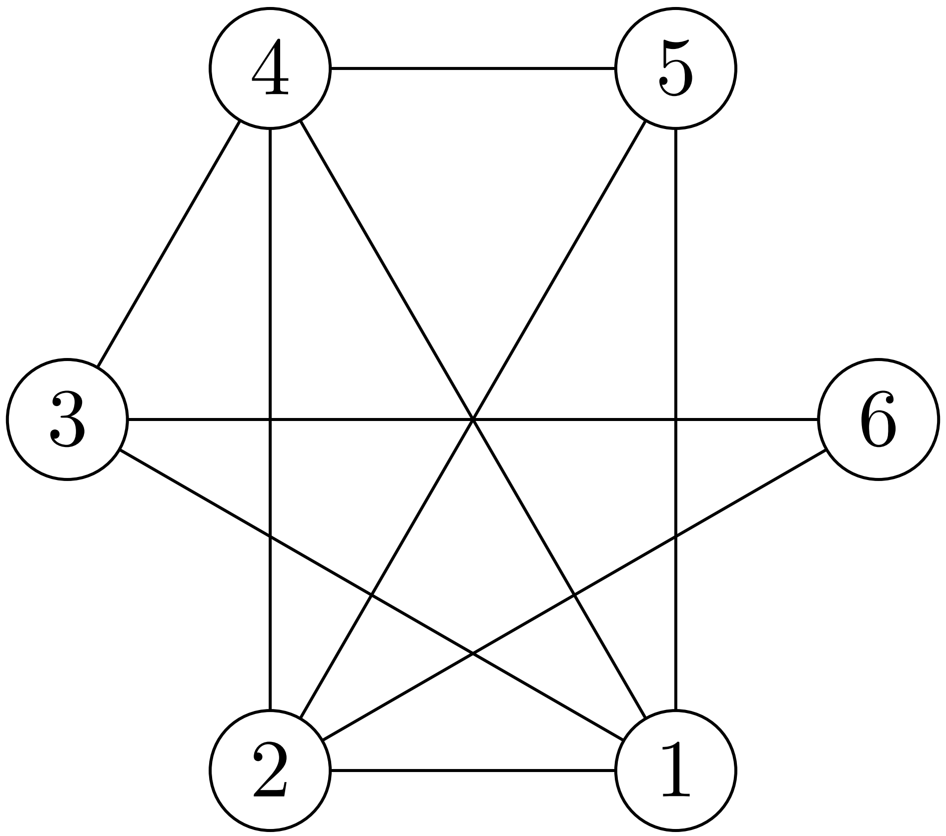


G

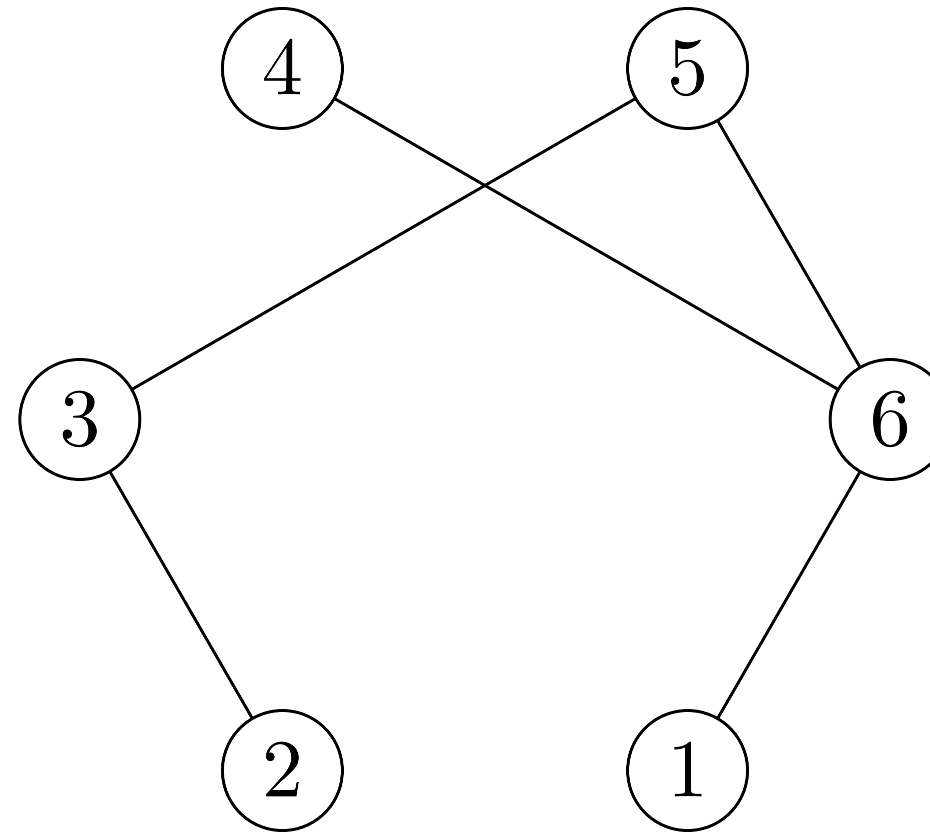
$\bar{G}$

Finnes en k-klikk? $\Longleftrightarrow$ Finnes et $(|V|-k)$-dekke?

$\bullet \longmapsto \bullet$

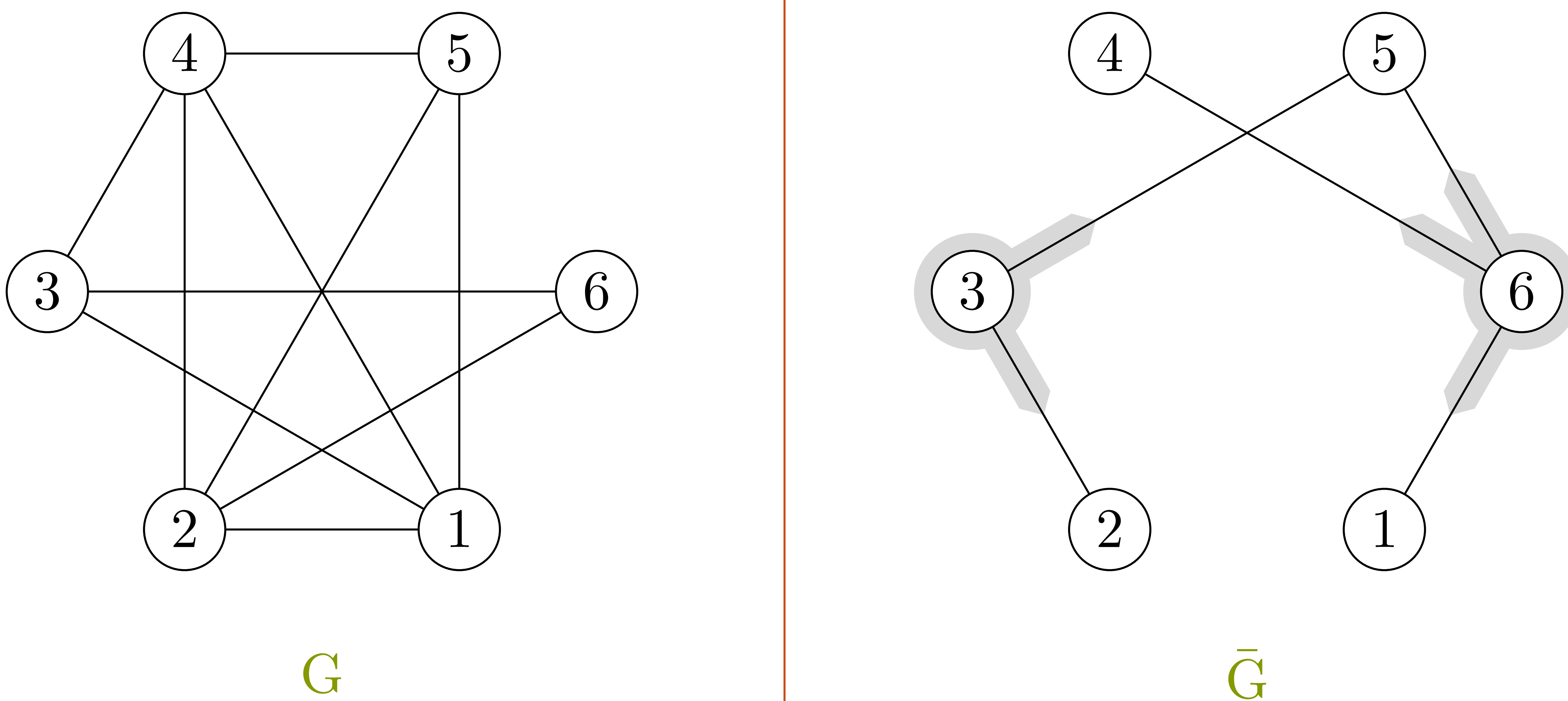


G

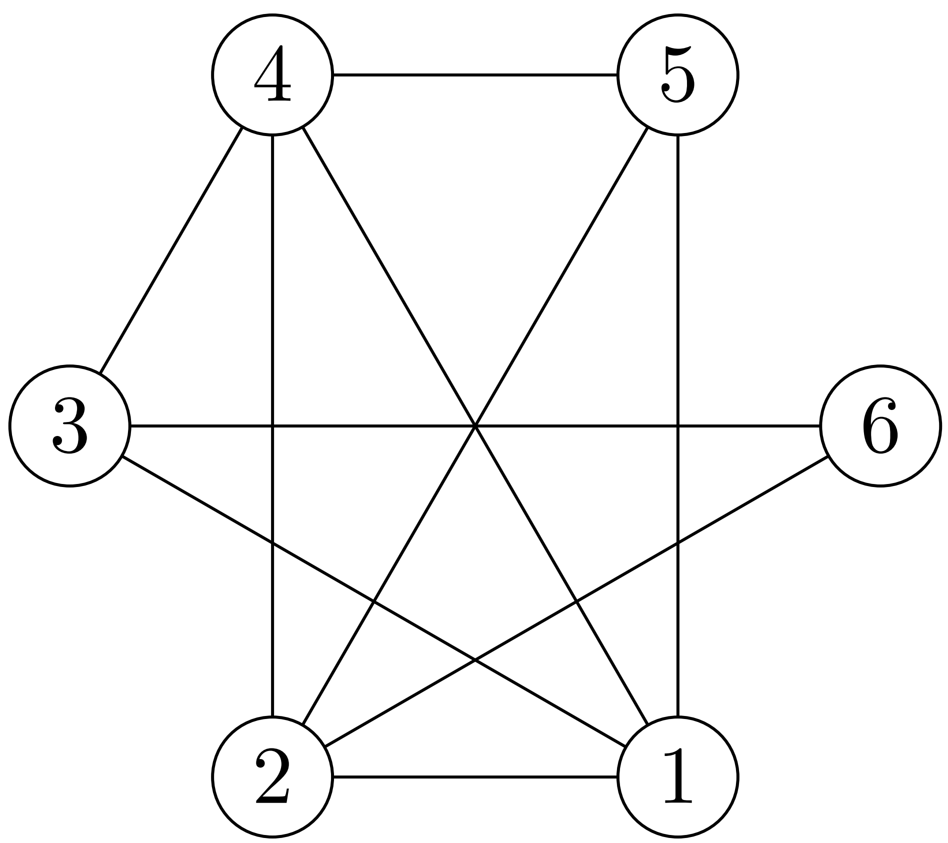


$\bar{G}$

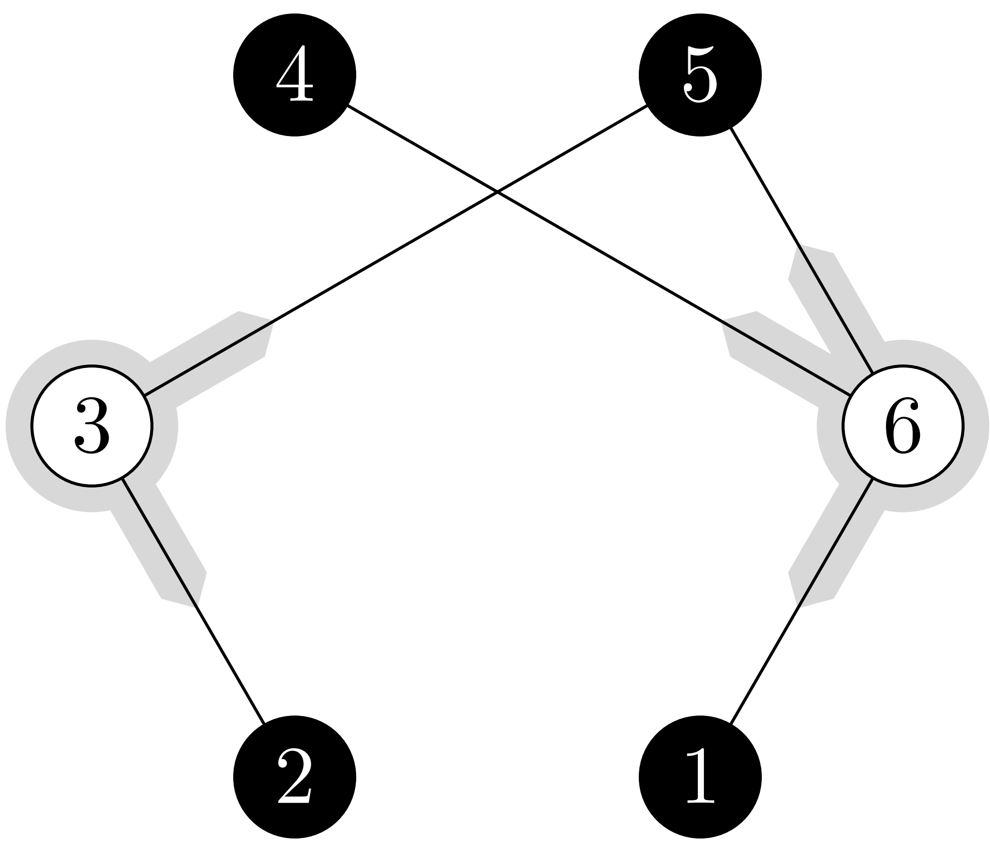
Finnes en k-klikk? $\Longleftrightarrow$ Finnes et $(|V|-k)$-dekke?



Finnes en k-klikk? $\Longleftrightarrow$ Finnes et $(|V|-k)$-dekke?

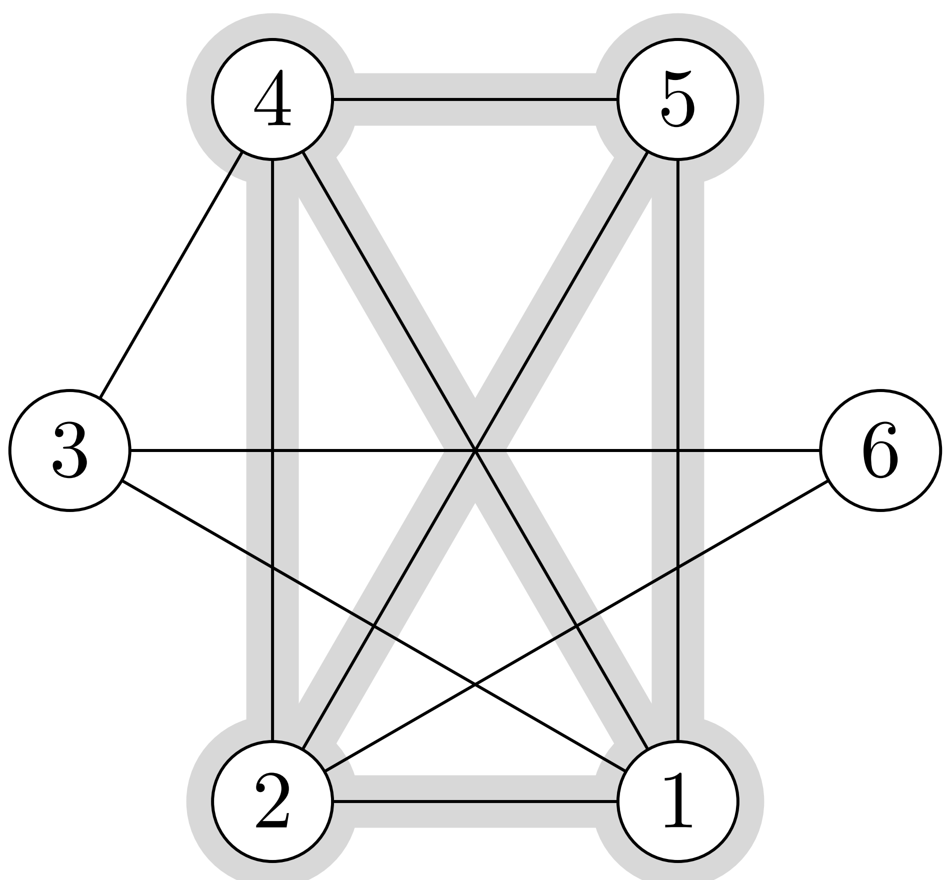


G

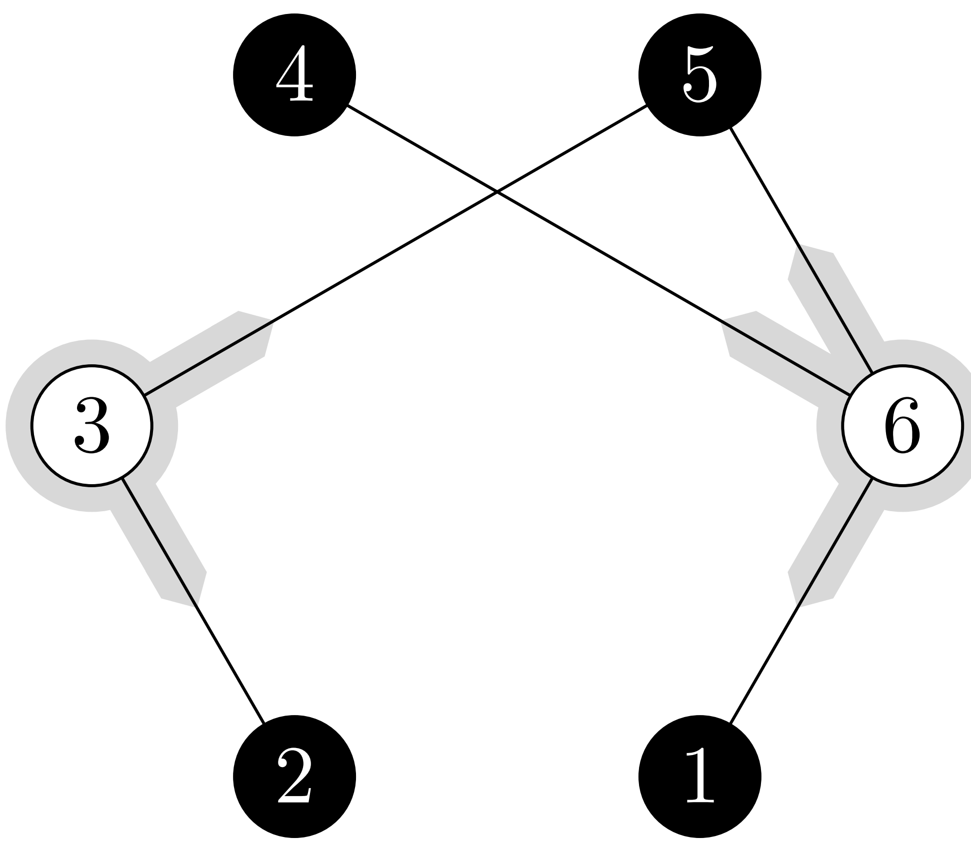


$\bar{G}$

Finnes en k-klikk? $\iff$ Finnes et $(|V| - k)$-dekke?



G



$\bar{G}$

Finnes en k-klikk? $\Longleftrightarrow$ Finnes et $(|V|-k)$-dekke?

7.9

HAM-CYCLE

› HAM-CYCLE

- **HAM-CYCLE**
  - **Instans:** En urettet graf G

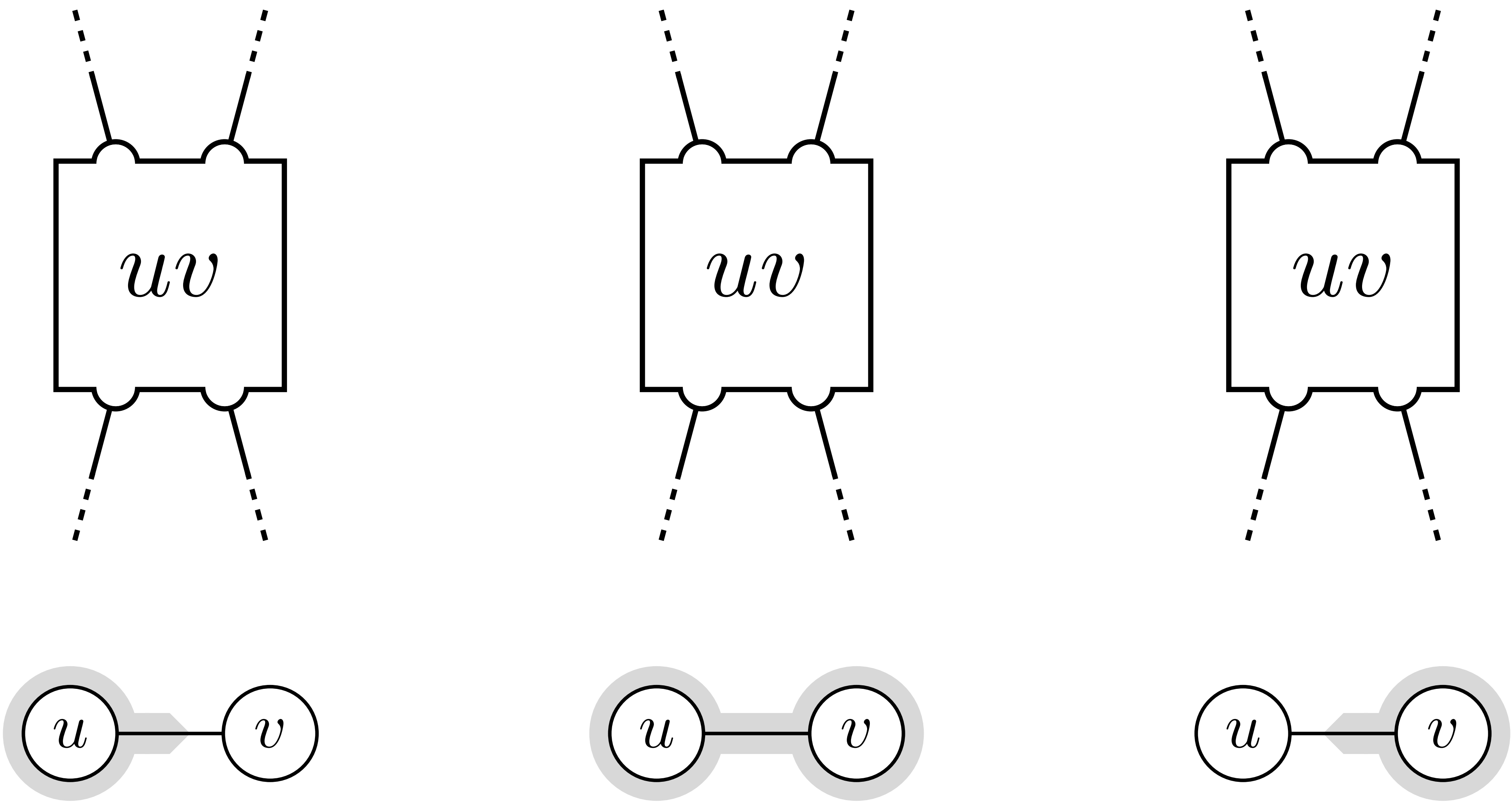
- **HAM-CYCLE**
  - **Instans:** En urettet graf G
  - **Spørsmål:** Finnes det en sykel som inneholder alle nodene?

- **HAM-CYCLE**
  - **Instans:** En urettet graf G
  - **Spørsmål:** Finnes det en sykel som inneholder alle nodene?
- Vi reduserer fra VERTEX-COVER

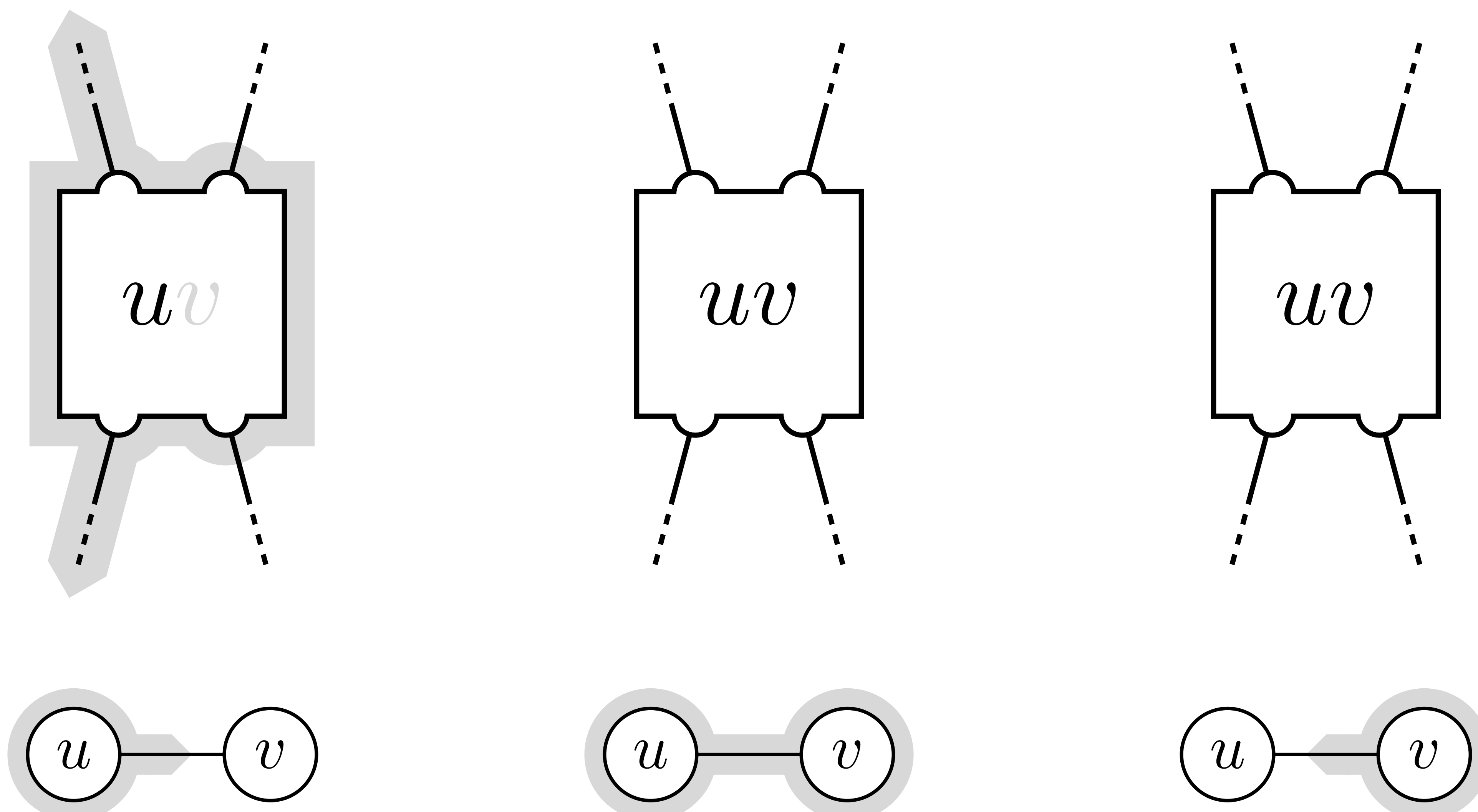
- **HAM-CYCLE**
  - **Instans:** En urettet graf G
  - **Spørsmål:** Finnes det en sykel som inneholder alle nodene?
- Vi reduserer fra VERTEX-COVER
- Sykelen må kunne velge noder til nodedekket

- **HAM-CYCLE**
  - **Instans:** En urettet graf G
  - **Spørsmål:** Finnes det en sykel som inneholder alle nodene?
- Vi reduserer fra VERTEX-COVER
- Sykelen må kunne velge noder til nodedekket
- En kant (u, v) kan dekkes av u eller v eller begge

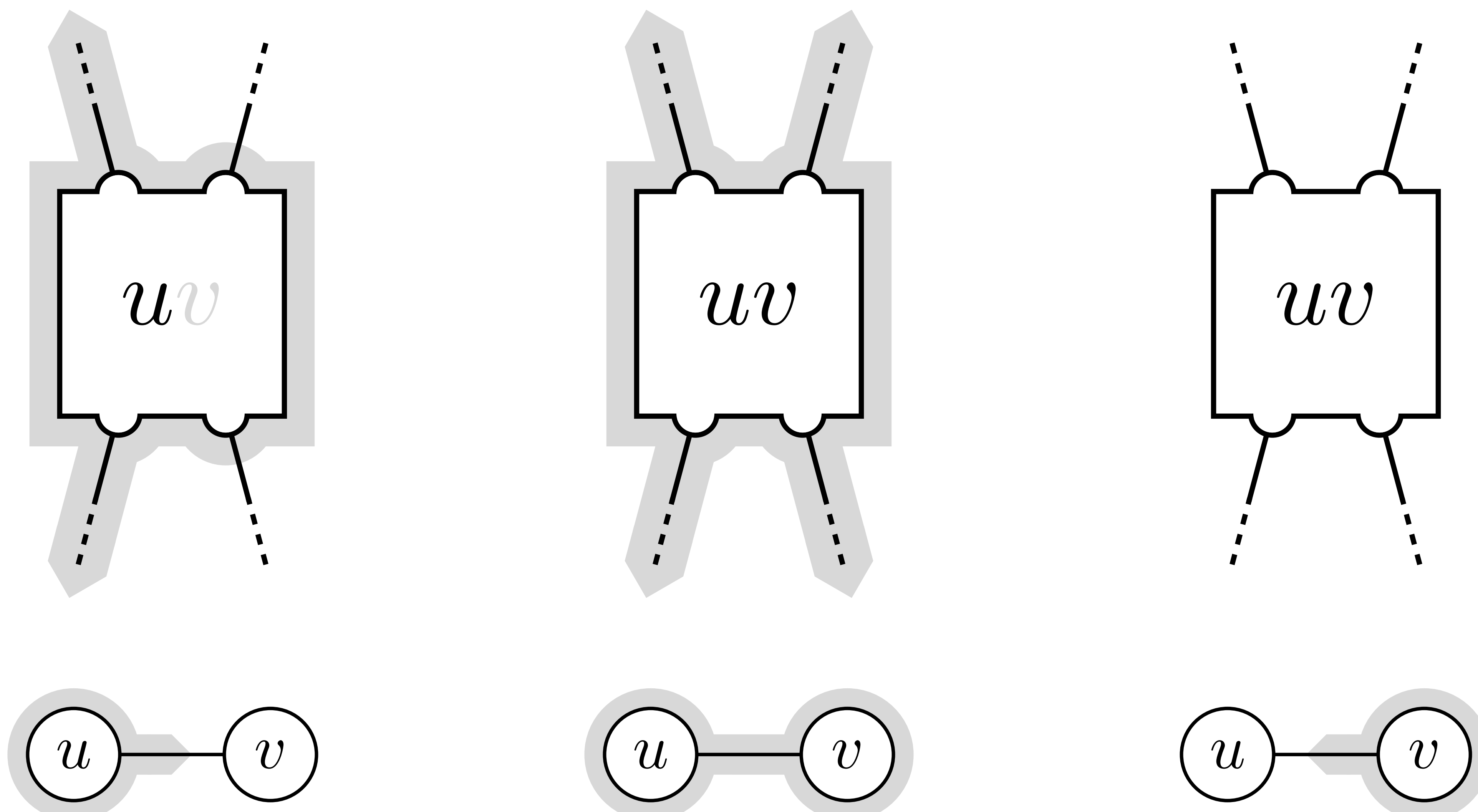
- **HAM-CYCLE**
  - **Instans:** En urettet graf G
  - **Spørsmål:** Finnes det en sykel som inneholder alle nodene?
- Vi reduserer fra VERTEX-COVER
- Sykelen må kunne velge noder til nodedekket
- En kant (u, v) kan dekkes av u eller v eller begge
- Vi lager oss en «widget» som hjelper oss med dette



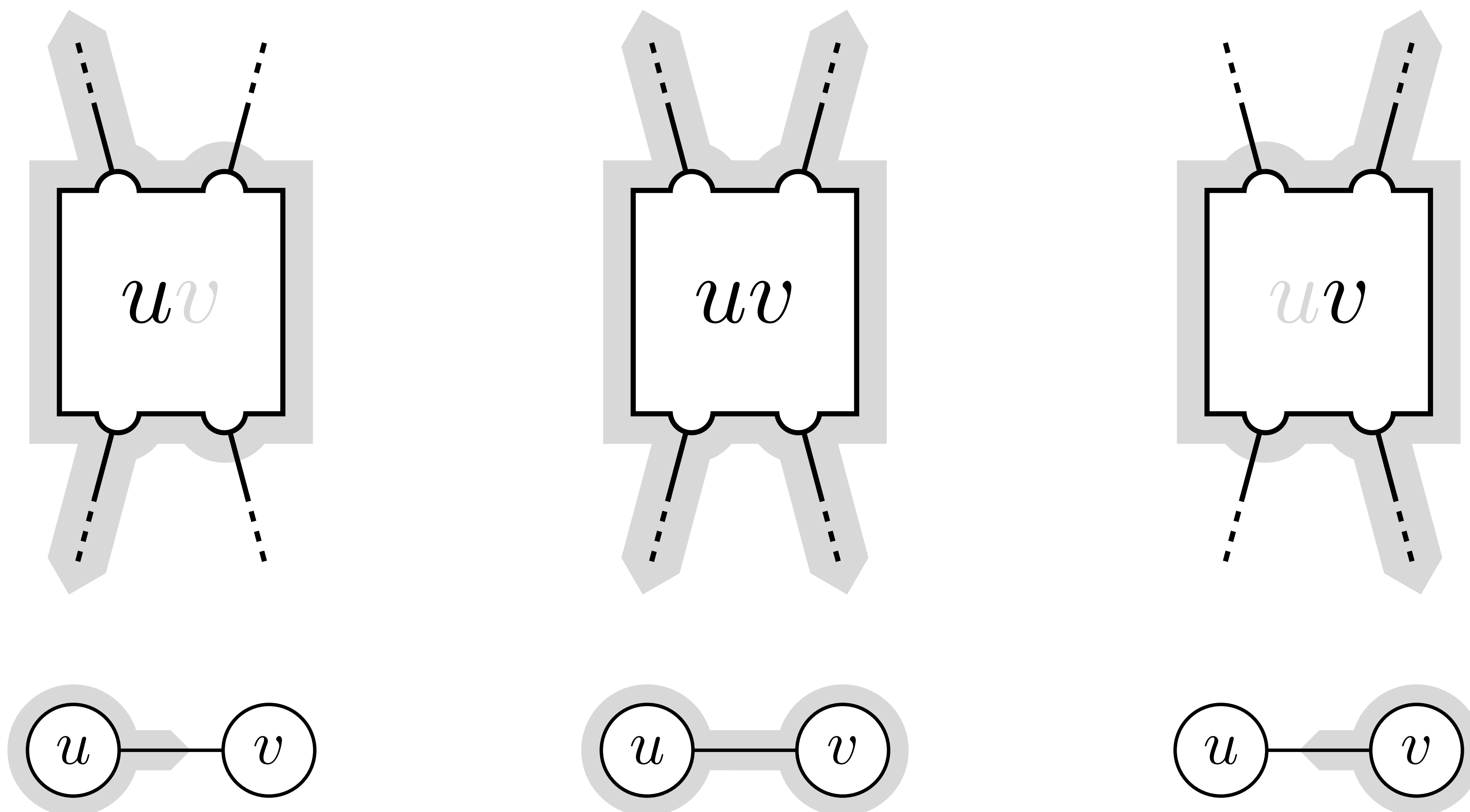
En widget uv representerer kanten mellom u og v



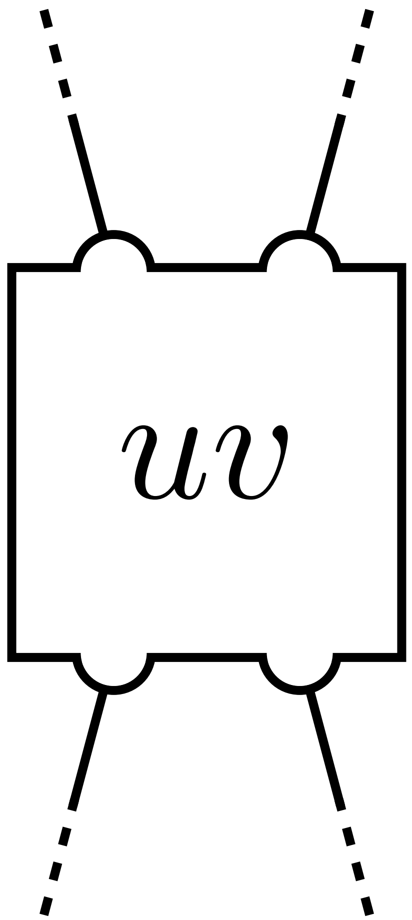
Om sykelen går igjennom på venstre side, velges u



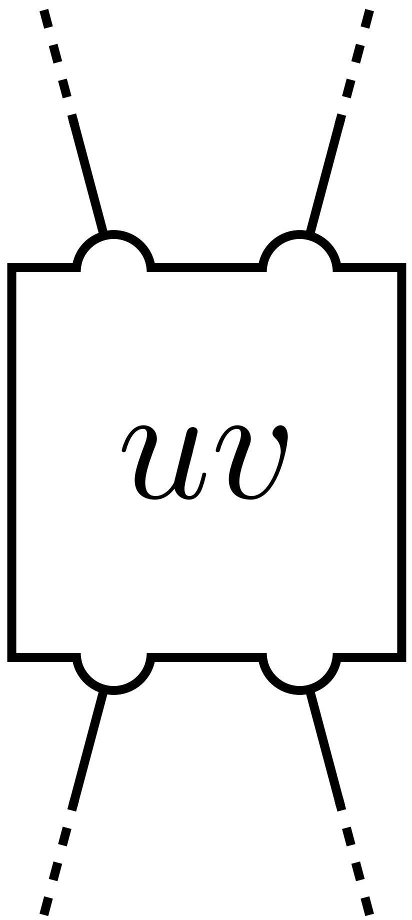
Om sykelen går igjennom på begge sider, velges begge

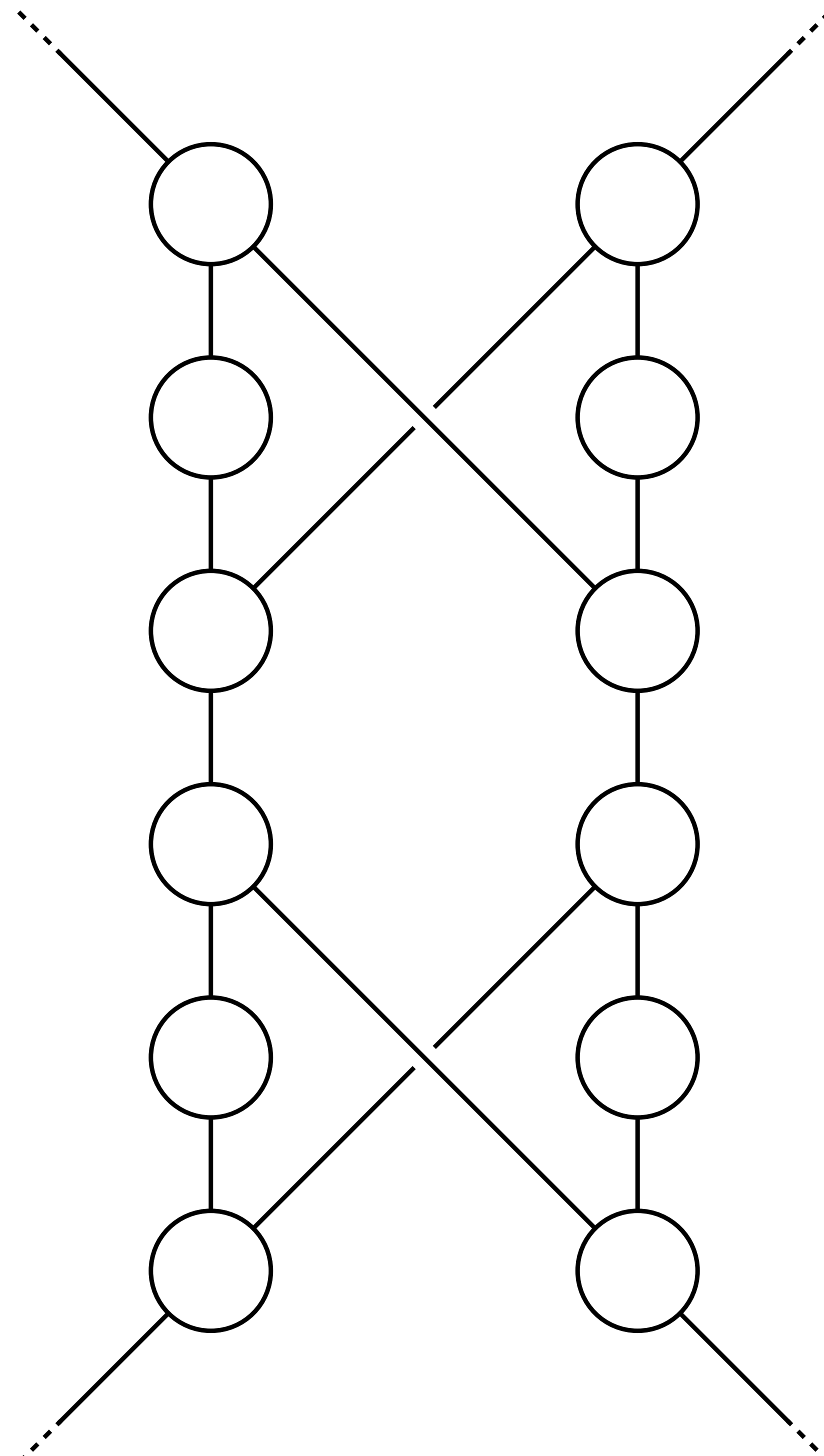


Om sykelen går igjennom på høyre side, velges v

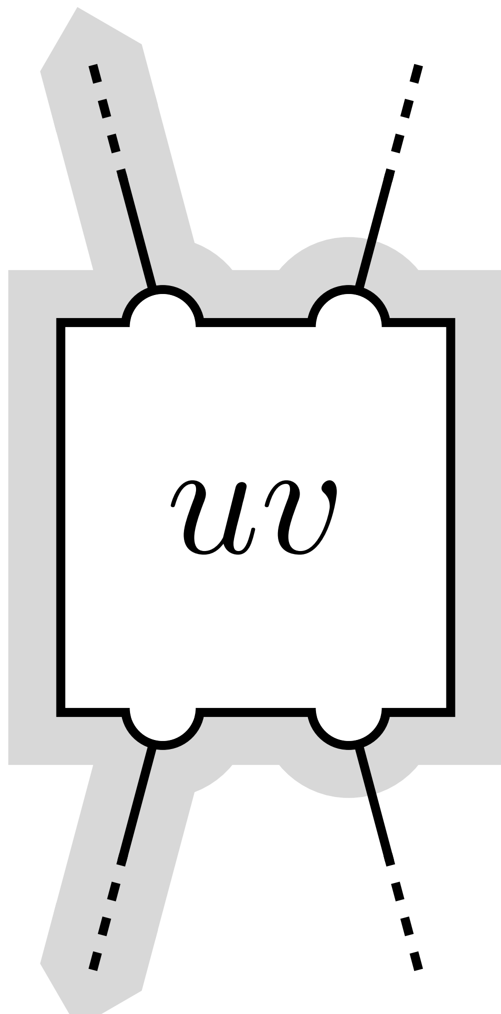


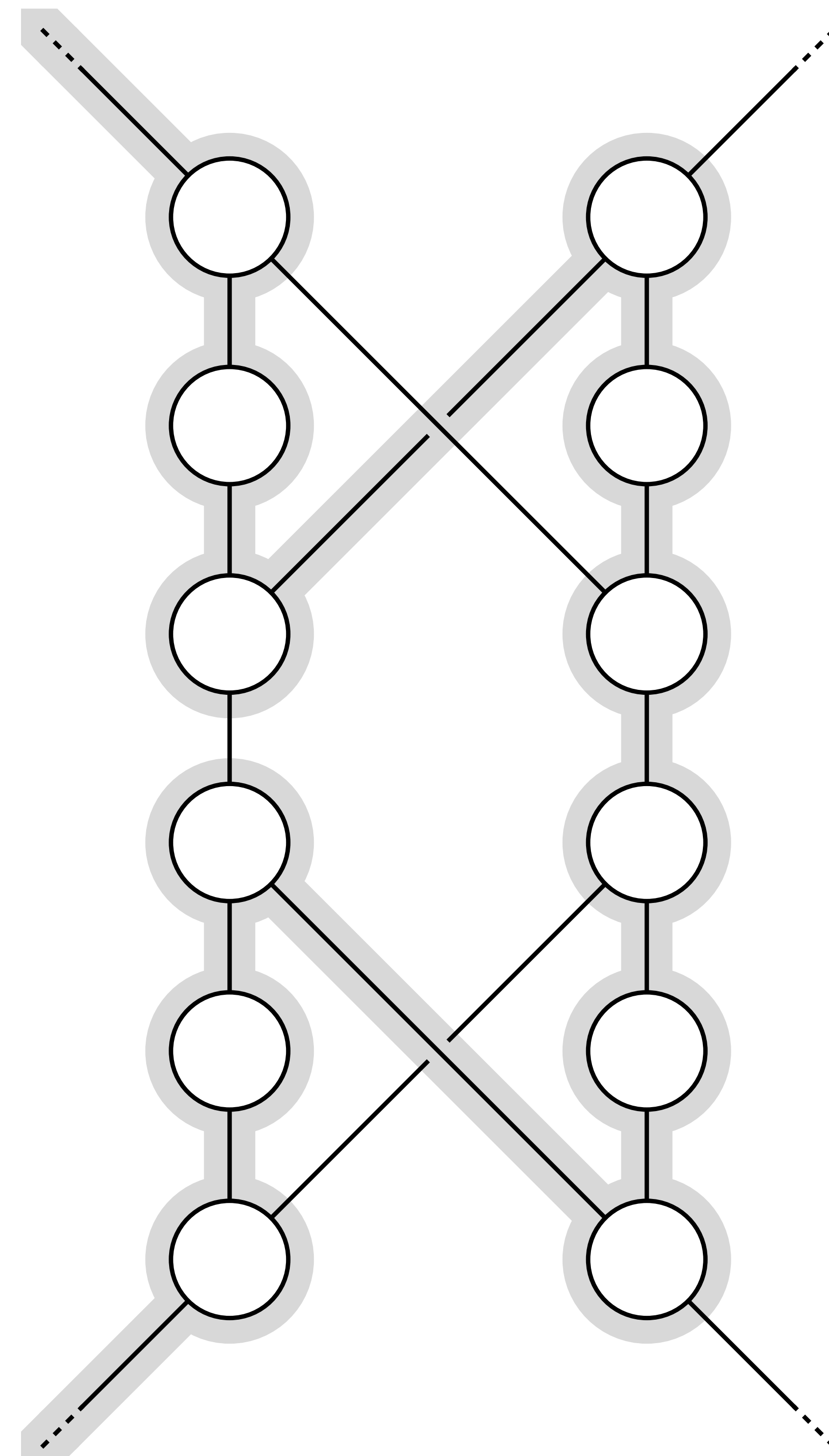
Hvordan lager vi slike widgets?

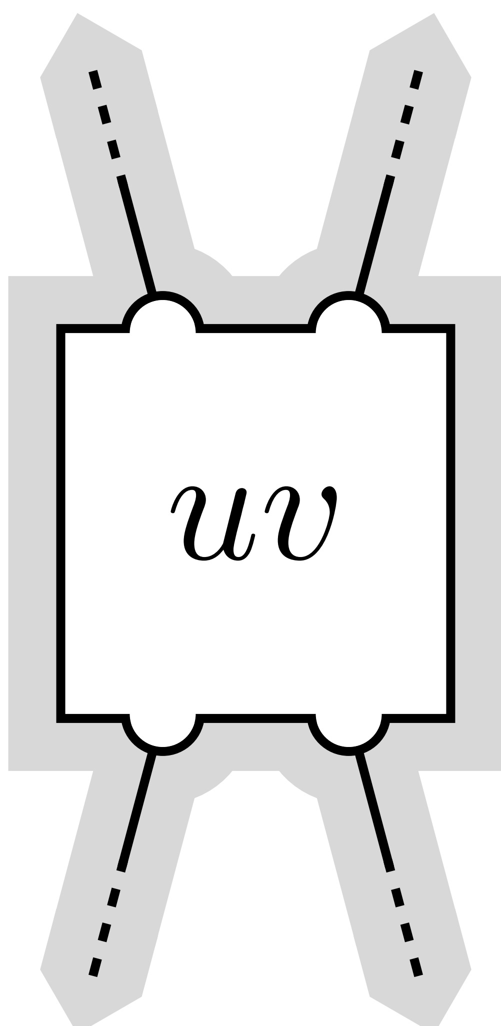


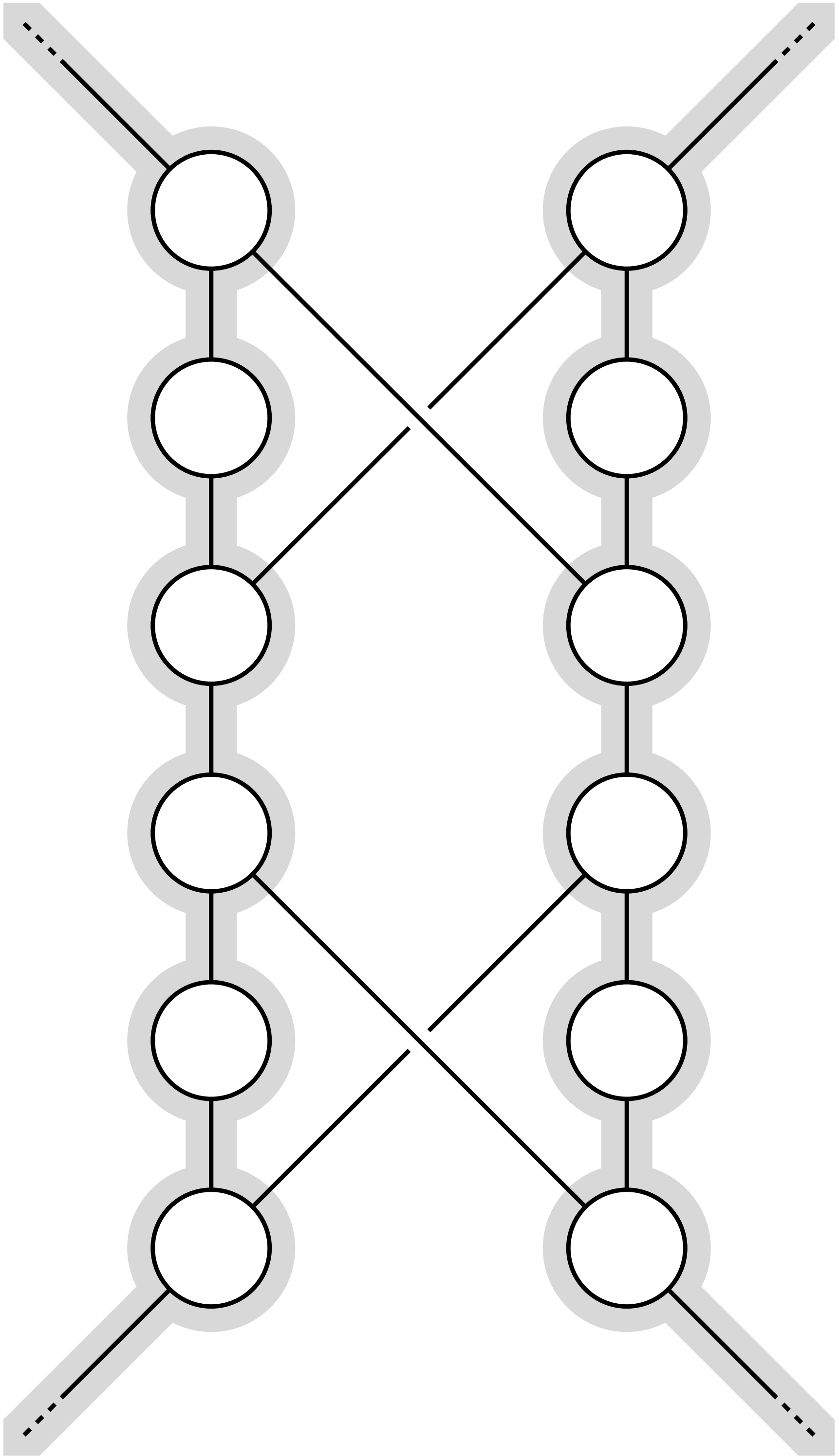


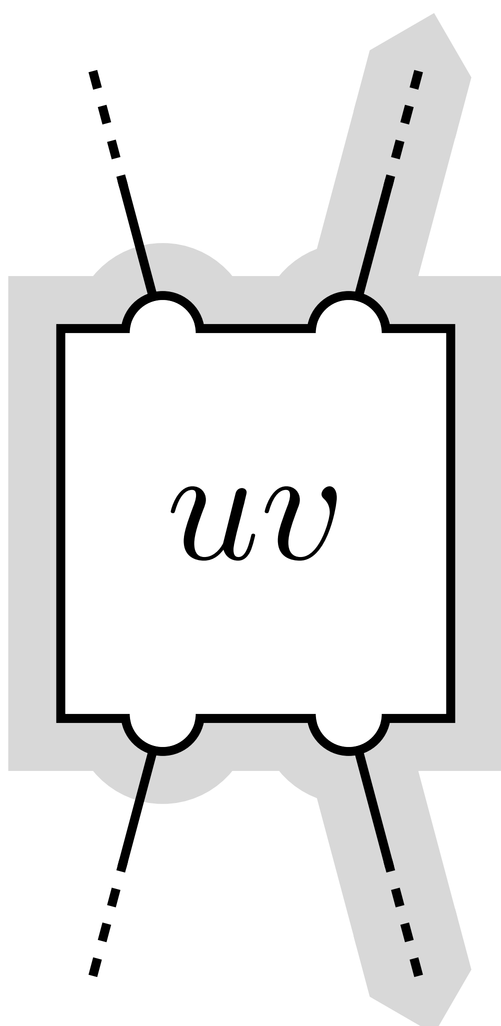
Kan uansett besøke alle

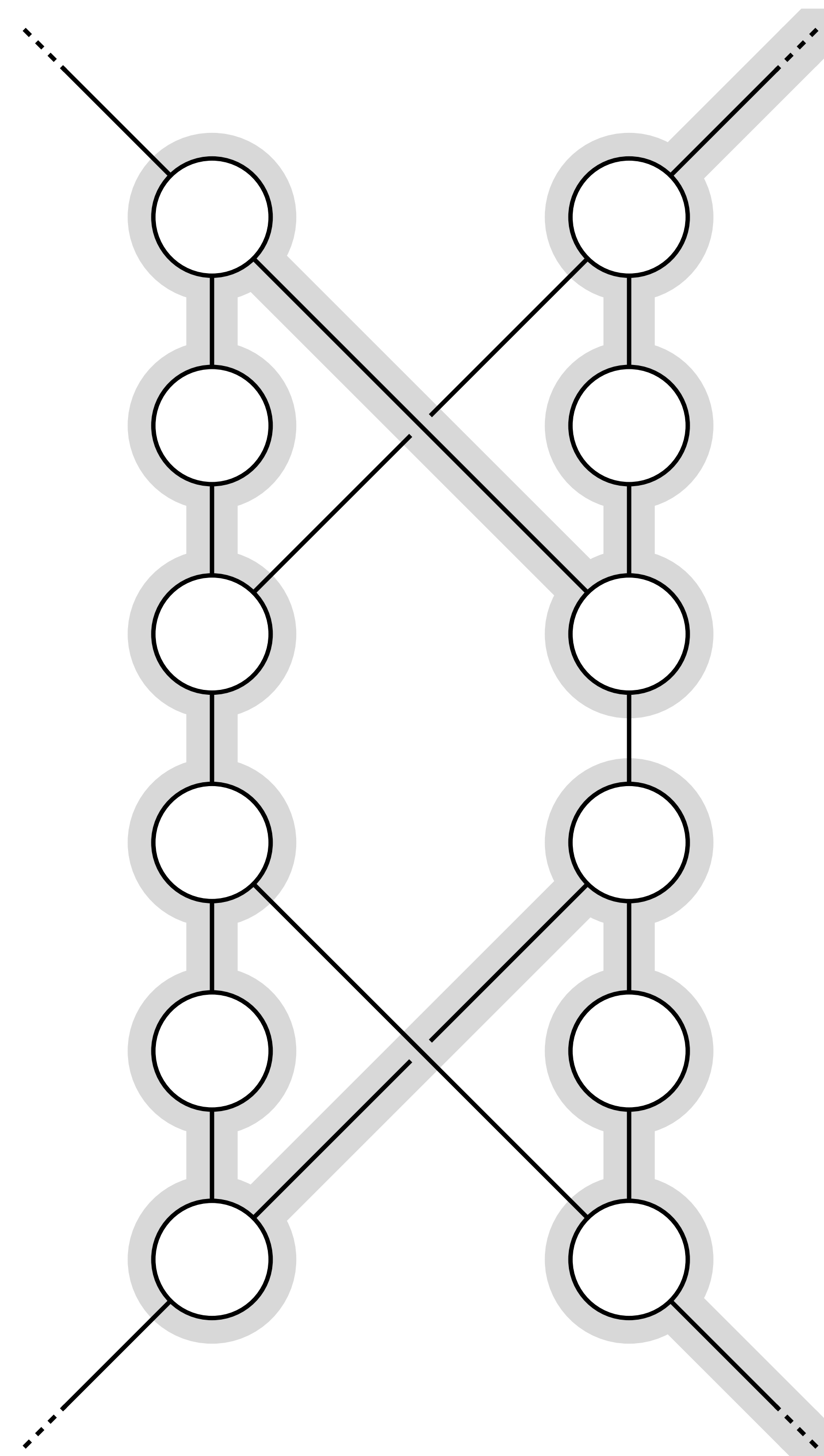

uv




uv




uv



- Lag «nabolister» ved hjelp av widgets

Husk: Hver widget representerer én av kantene i grafen

- Lag «nabolister» ved hjelp av widgets
  - Ordne utkanter for hver originale node vilkårlig

Vi lager oss en liste med ut-kanter (altså av widgets) for v

- Lag «nabolister» ved hjelp av widgets
  - Ordne utkanter for hver originale node vilkårlig
  - Koble nederste widget-node til øverste widget-node for neste

På den siden av widget-en som representerer v

- Lag «nabolister» ved hjelp av widgets
  - Ordne utkanter for hver originale node vilkårlig
  - Koble nederste widget-node til øverste widget-node for neste
- Koble k «selektornoder» til første øverste og siste nederste

Selektornode i representerer den i-ende noden i nodedekket

- Lag «nabolister» ved hjelp av widgets
  - Ordne utkanter for hver originale node vilkårlig
  - Koble nederste widget-node til øverste widget-node for neste
- Koble k «selektornoder» til første øverste og siste nederste
- Hver selektornode kan da velge seg en naboliste

Velger node og «dekker» kanter (sender Ham.-sykel gjennom widgets)

- Lag «nabolister» ved hjelp av widgets
  - Ordne utkanter for hver originale node vilkårlig
  - Koble nederste widget-node til øverste widget-node for neste
- Koble k «selektornoder» til første øverste og siste nederste
- Hver selektornode kan da velge seg en naboliste
  - Den «velger» ved å sende Hamilton-stien til den første, øverste widget-noden i nabolista

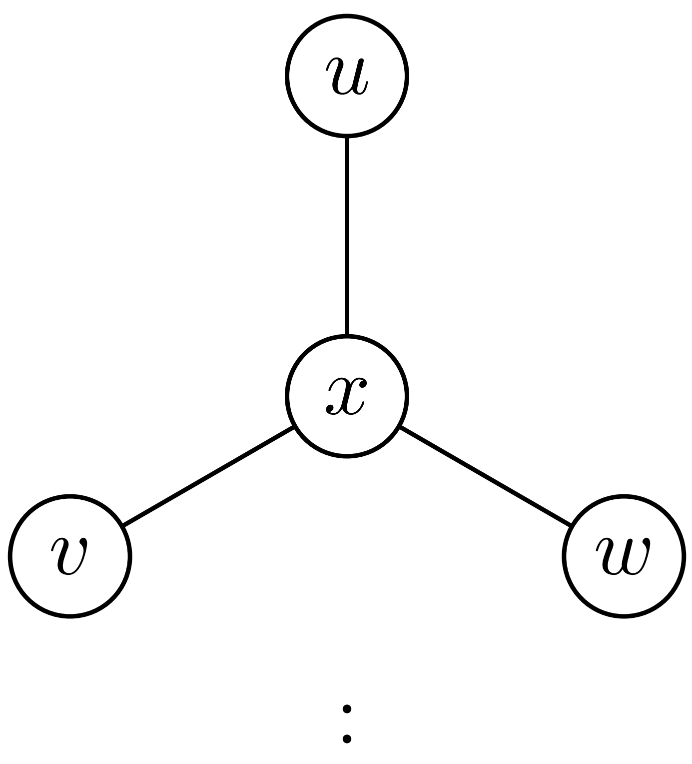
For v sin nabokant-liste: Går gjennom widgets i lista på v-siden

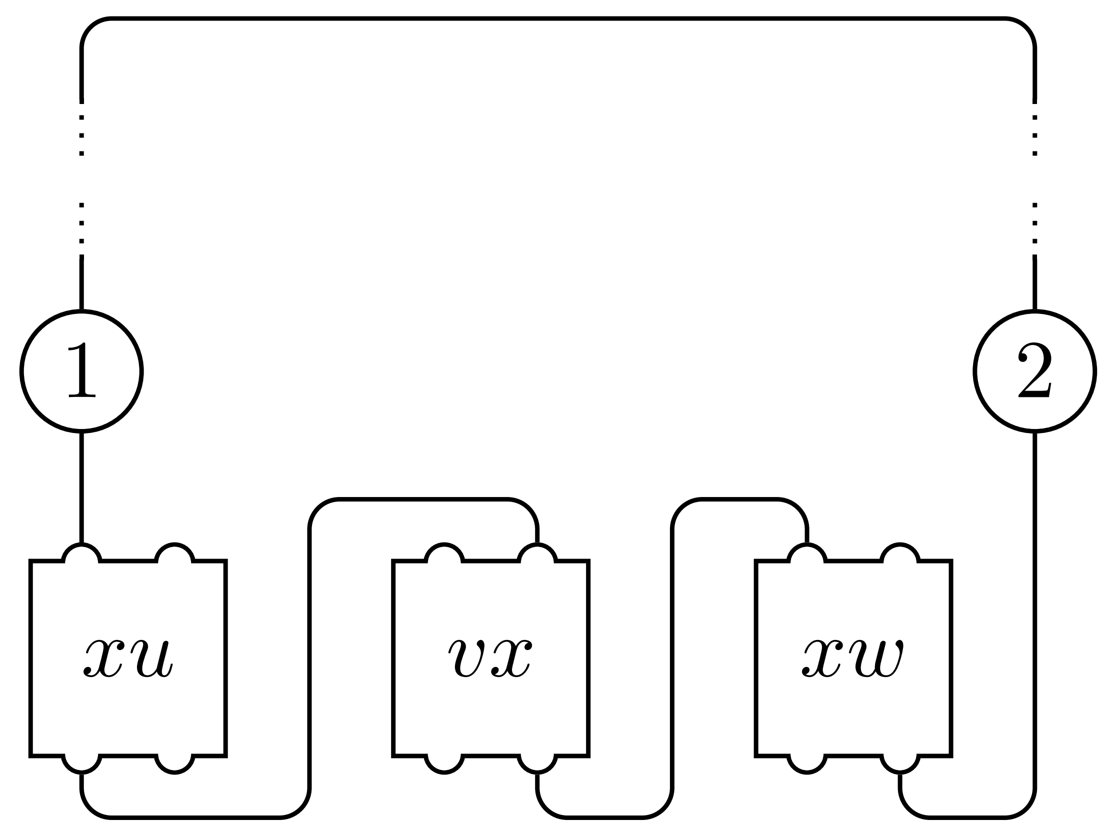
- Lag «nabolister» ved hjelp av widgets
  - Ordne utkanter for hver originale node vilkårlig
  - Koble nederste widget-node til øverste widget-node for neste
- Koble k «selektornoder» til første øverste og siste nederste
- Hver selektornode kan da velge seg en naboliste
  - Den «velger» ved å sende Hamilton-stien til den første, øverste widget-noden i nabolista
- Første og siste widget i lista er koblet til alle k selektornoder

Siste widget sender sykelen videre til neste (evt. første) selektornode

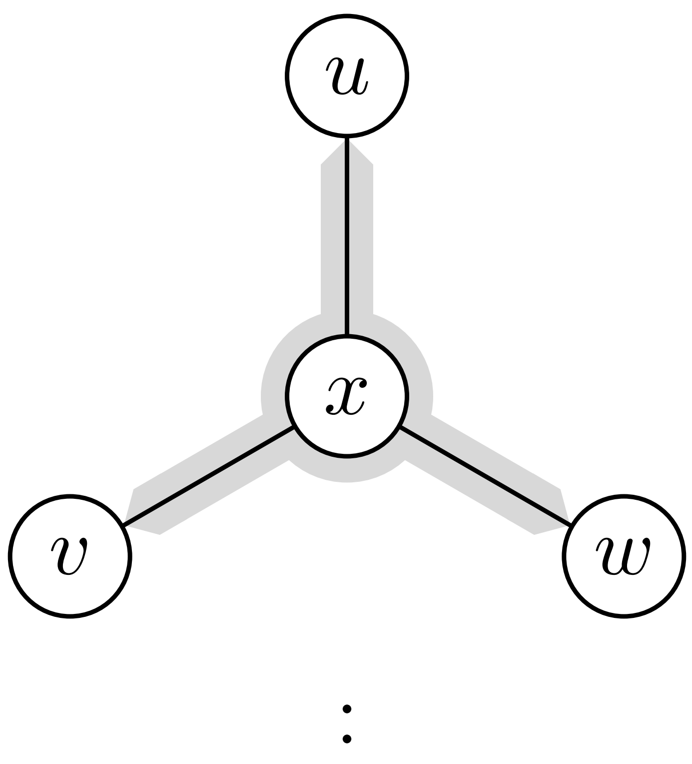
- Lag «nabolister» ved hjelp av widgets
  - Ordne utkanter for hver originale node vilkårlig
  - Koble nederste widget-node til øverste widget-node for neste
- Koble k «selektornoder» til første øverste og siste nederste
- Hver selektornode kan da velge seg en naboliste
  - Den «velger» ved å sende Hamilton-stien til den første, øverste widget-noden i nabolista
- Første og siste widget i lista er koblet til alle k selektornoder
- Vi har en Hamilton-sykel hvis og bare hvis hver naboliste velges

Vi er innom selektornodene, og går gjennom nabolister mellom dem

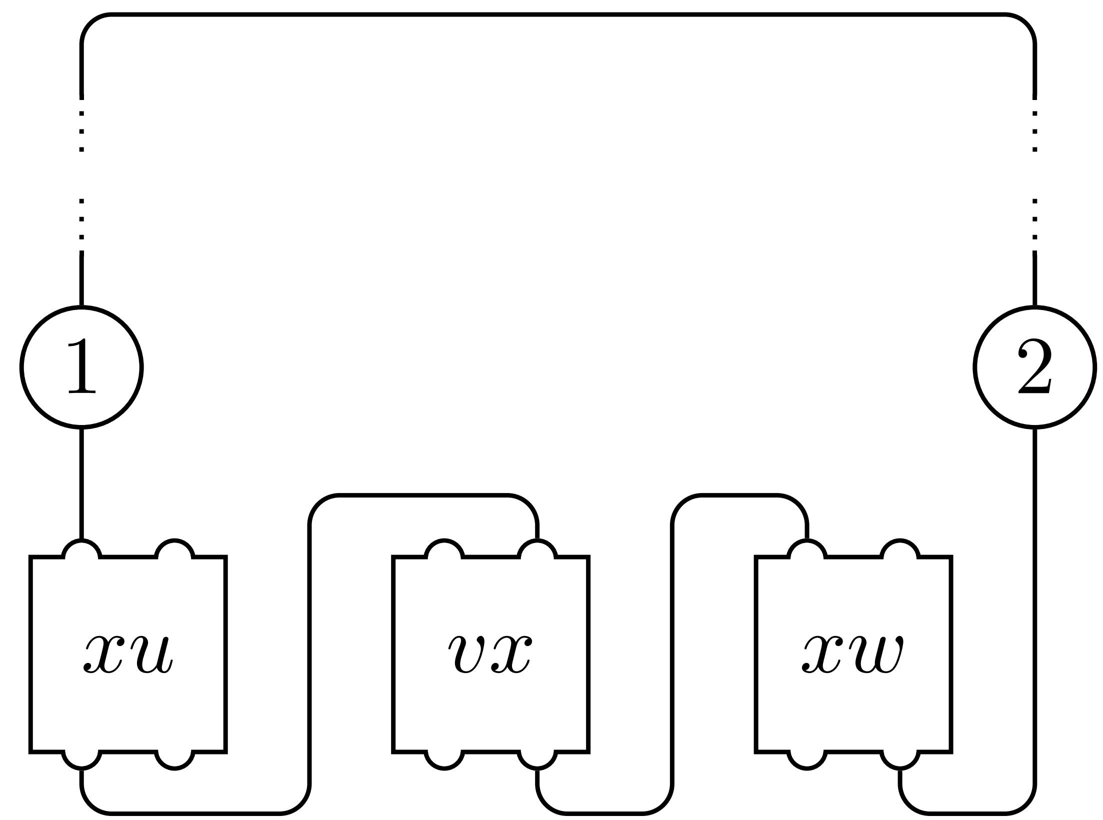


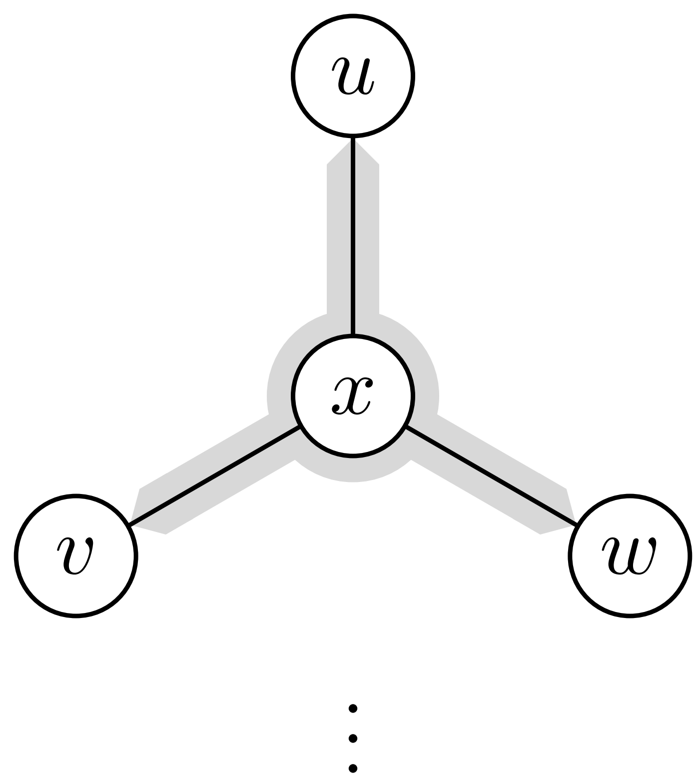


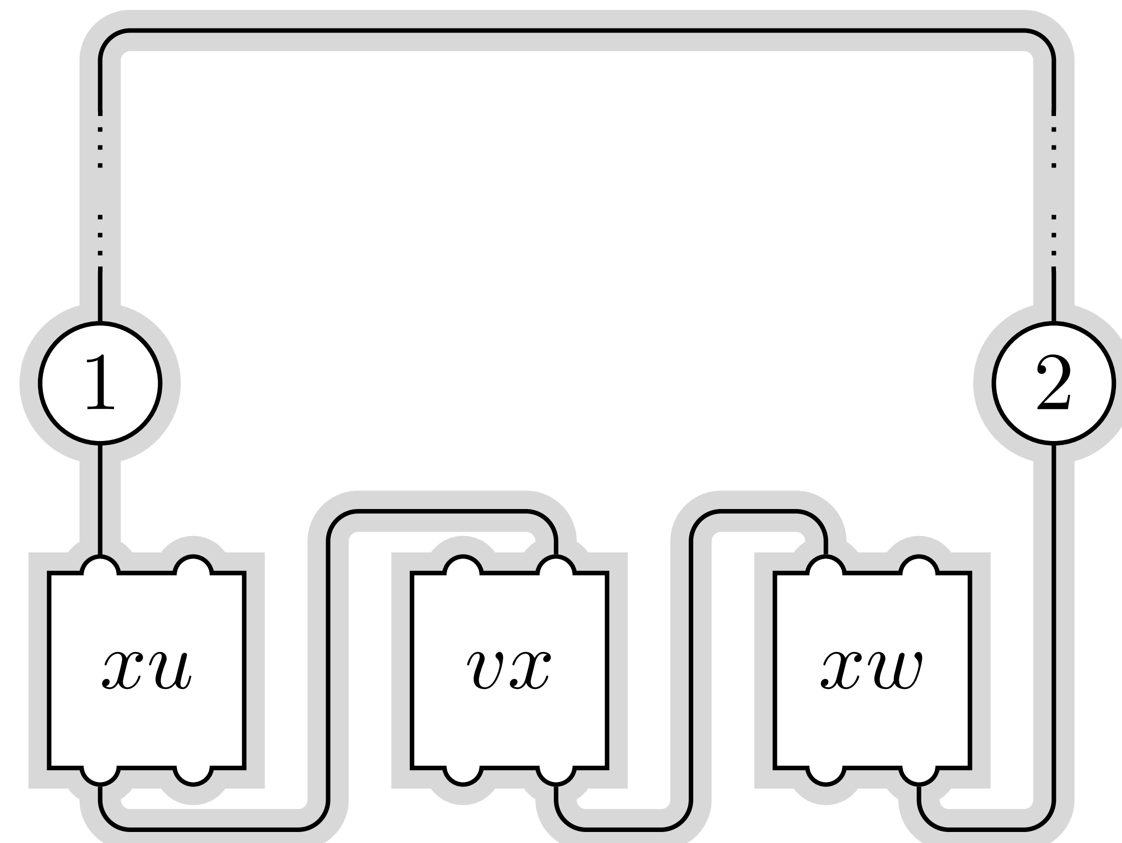
Vi velger x med selektor 1



Dekker kanter xu, xv og xw

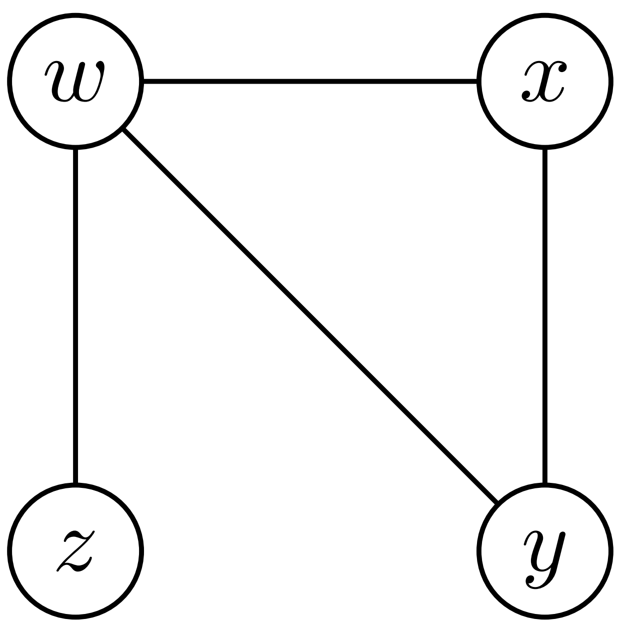




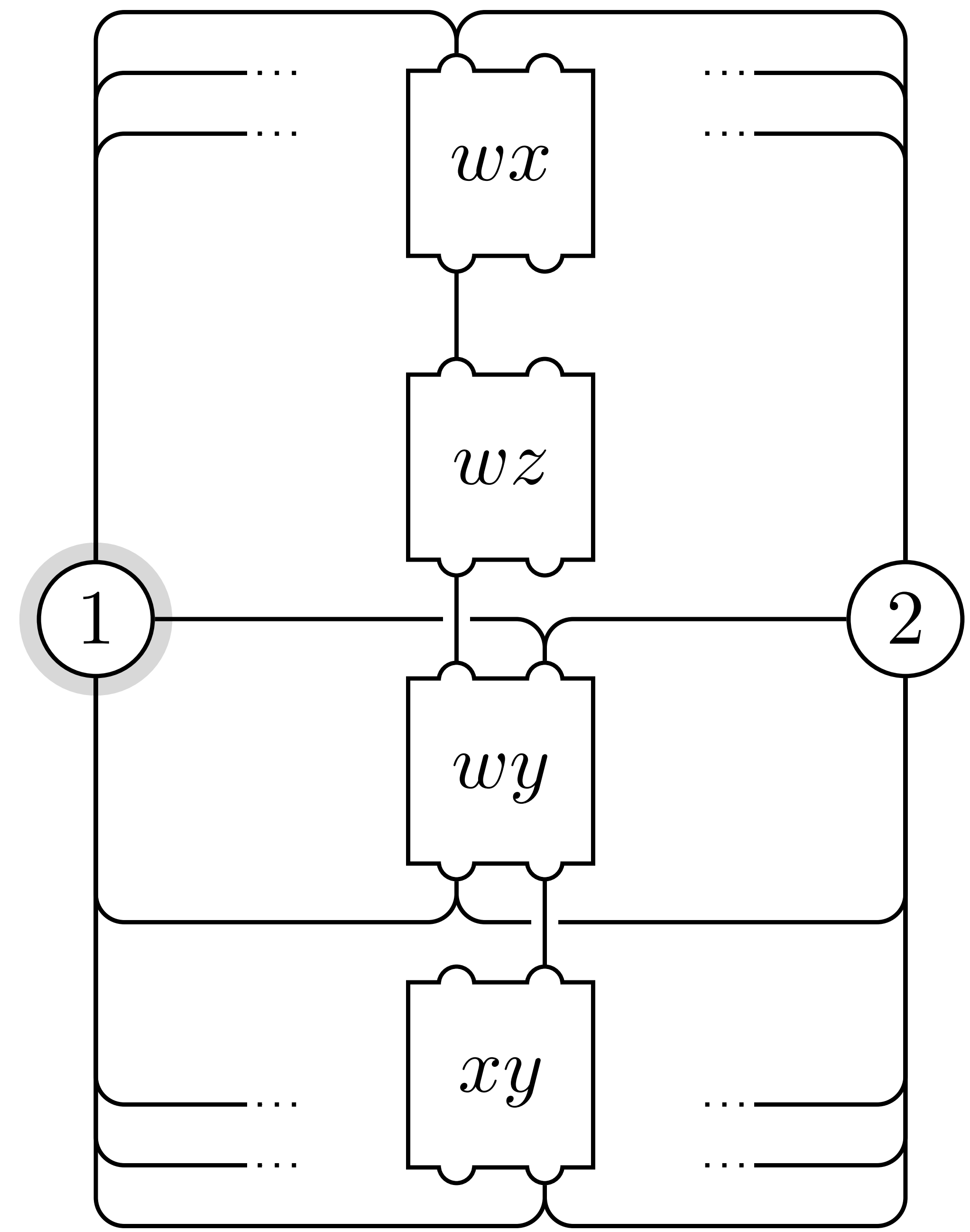


Dekker kanter xu, xv og xw

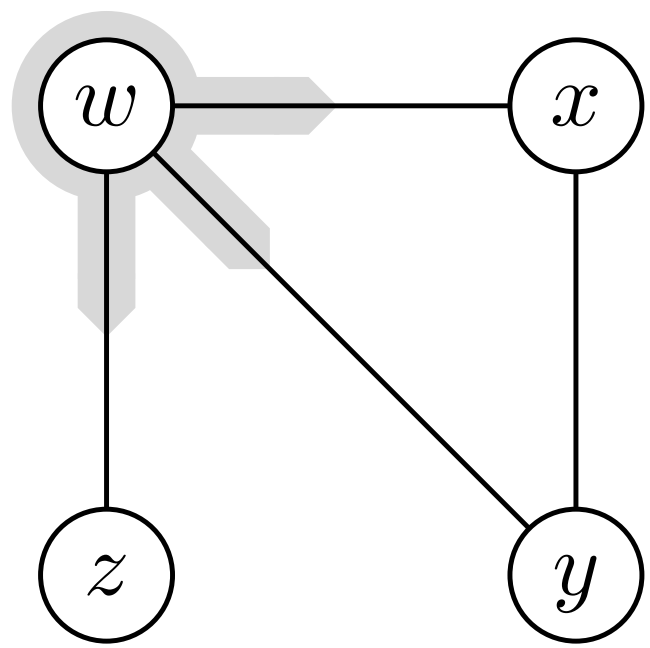
Sykel via tilsvarende widgets



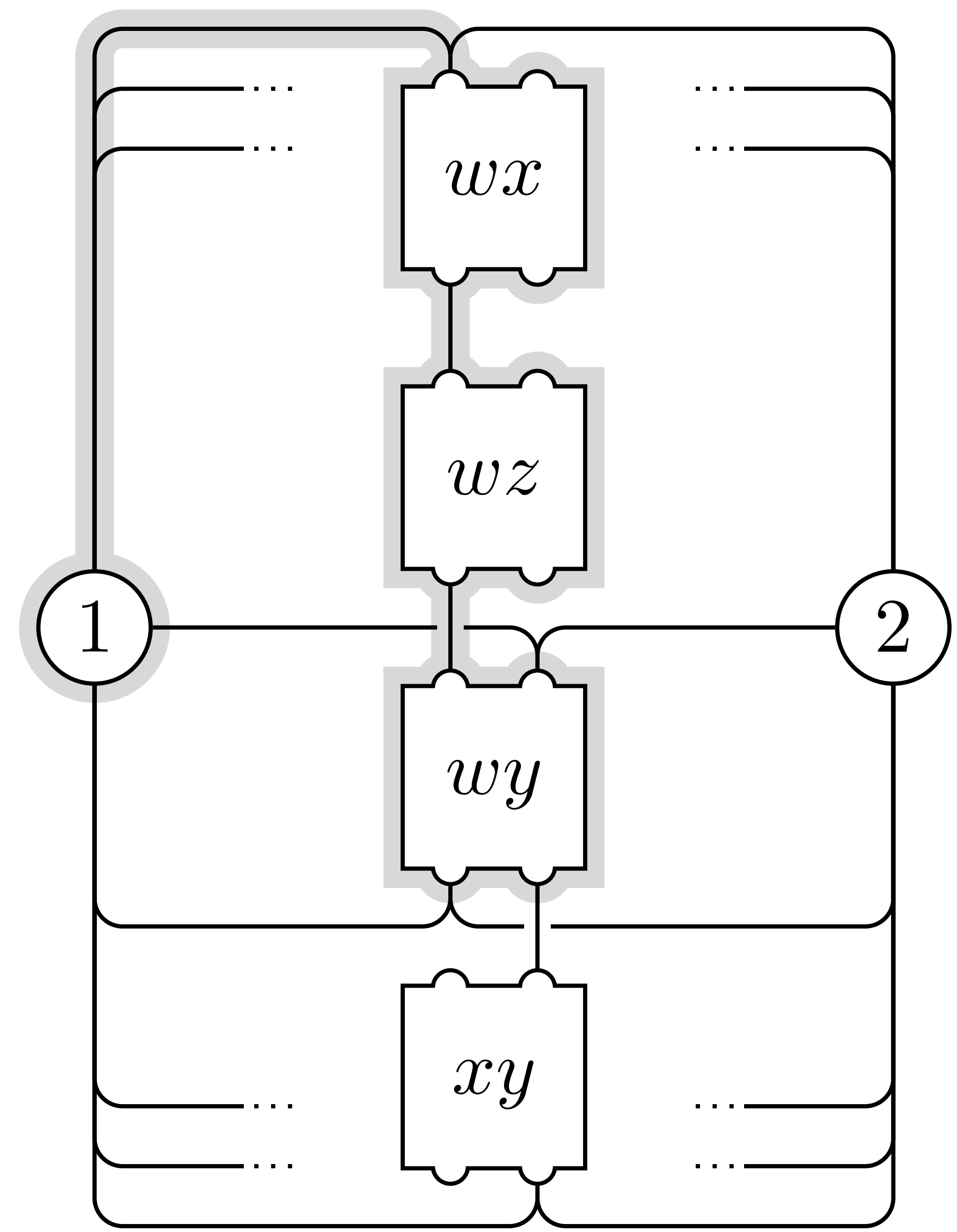
$k = 2$



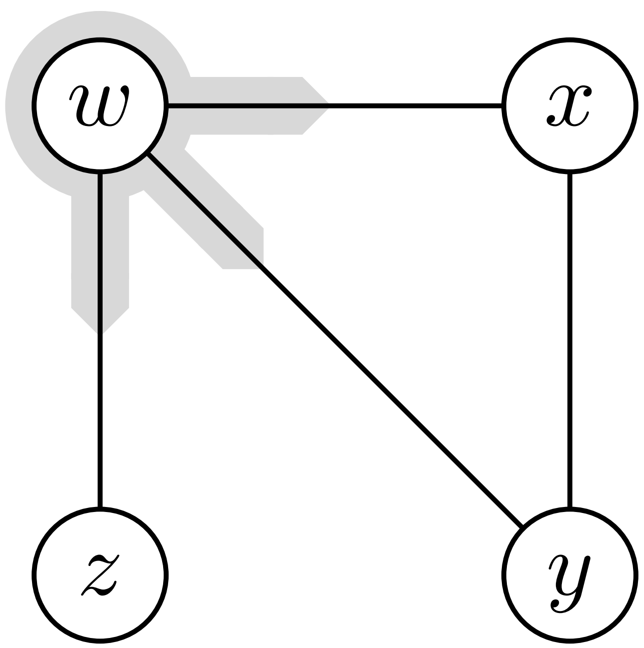
Finnes et k-dekke? $\iff$ Finnes en Hamilton-sykel?

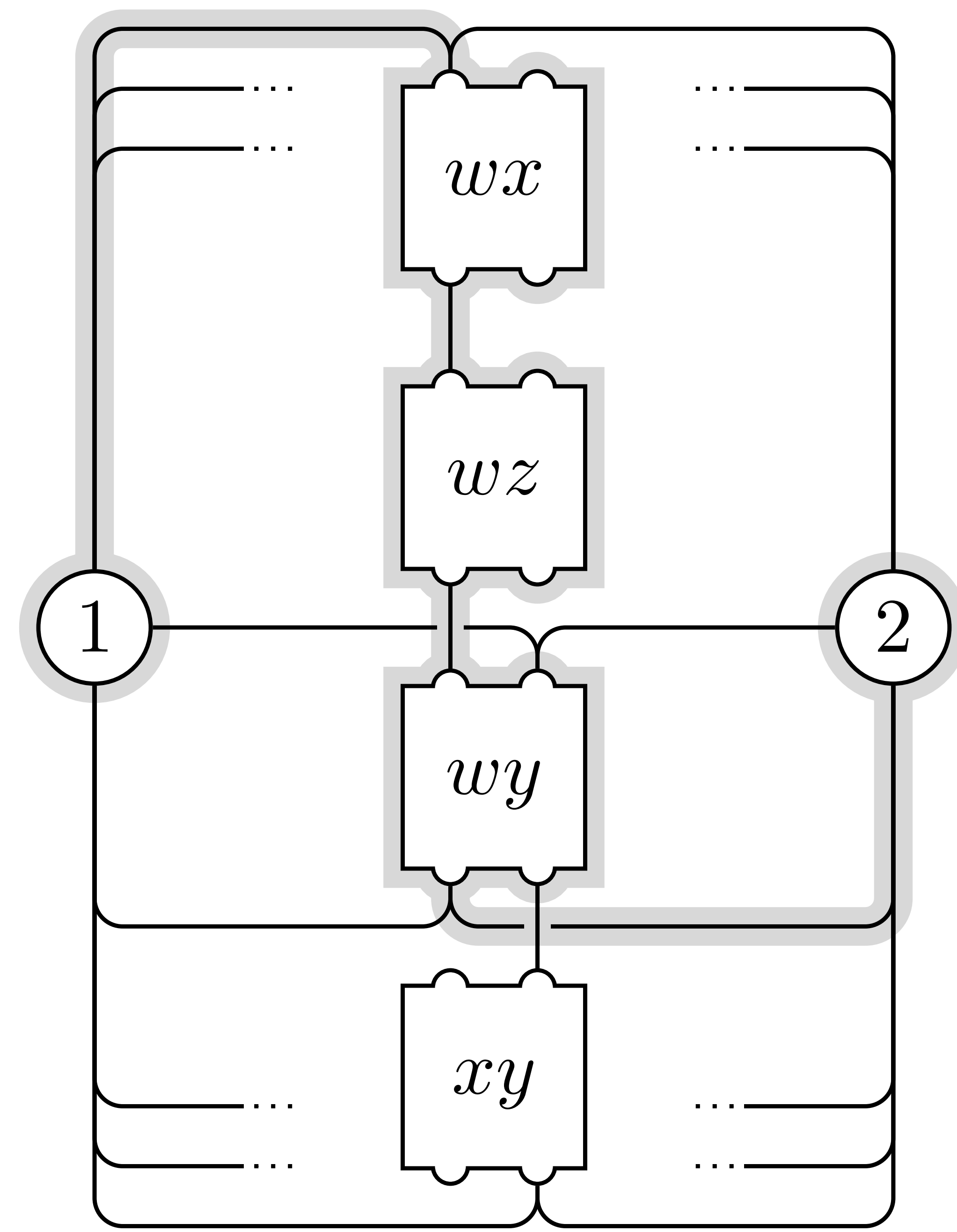


$k = 2$



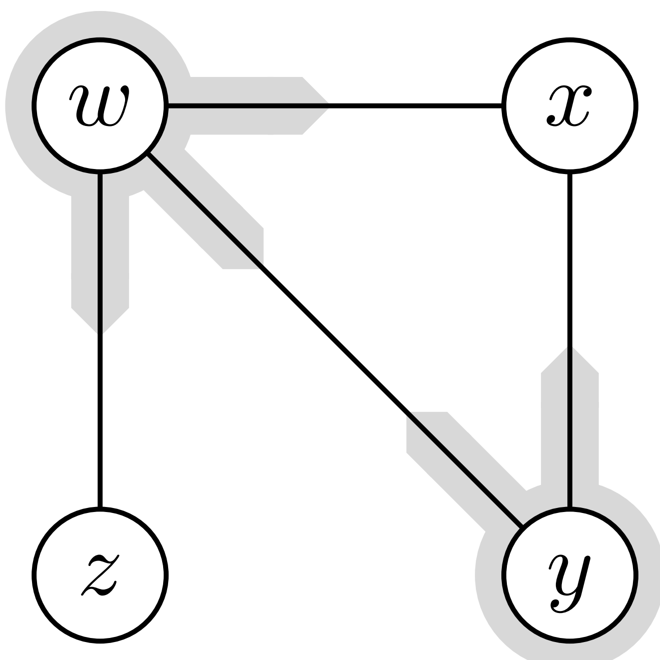
Finnes et k-dekke? $\Longleftrightarrow$ Finnes en Hamilton-sykel?



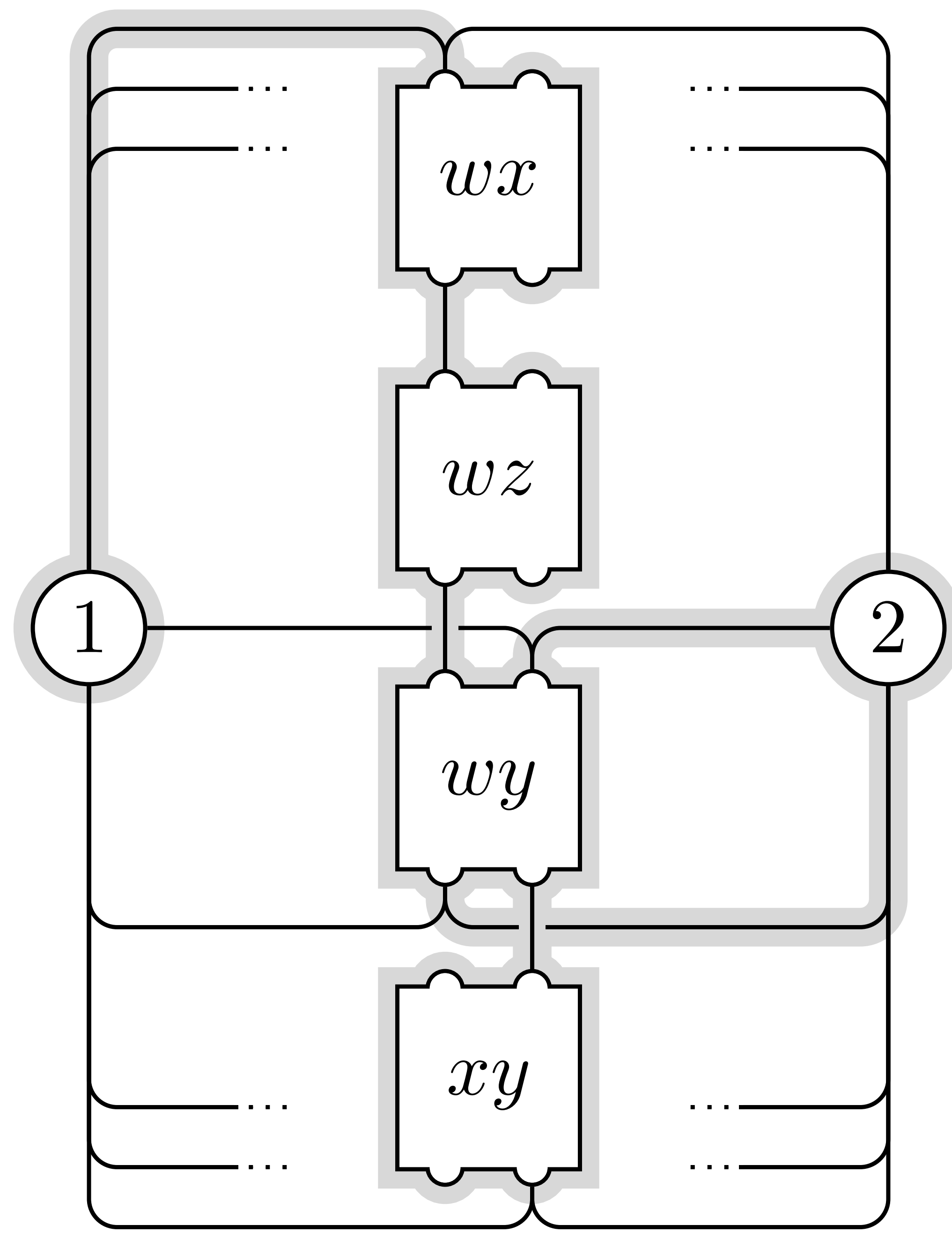


$k = 2$

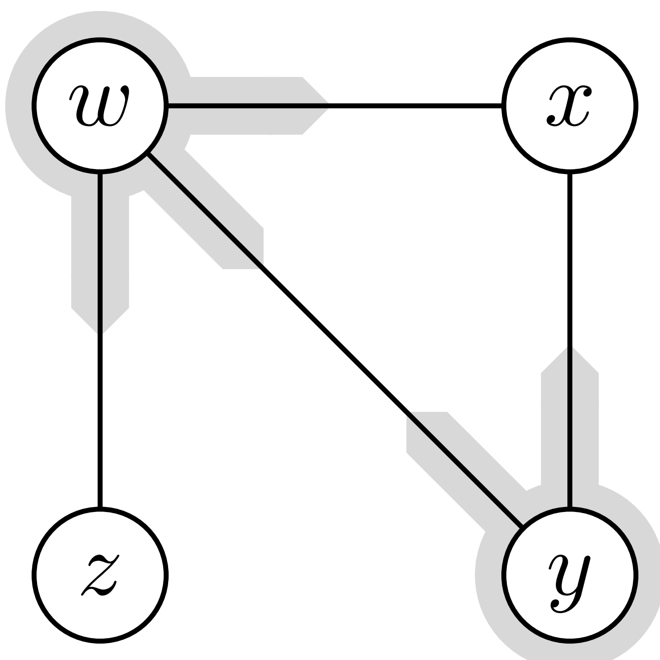
Finnes et k-dekke? $\iff$ Finnes en Hamilton-sykel?



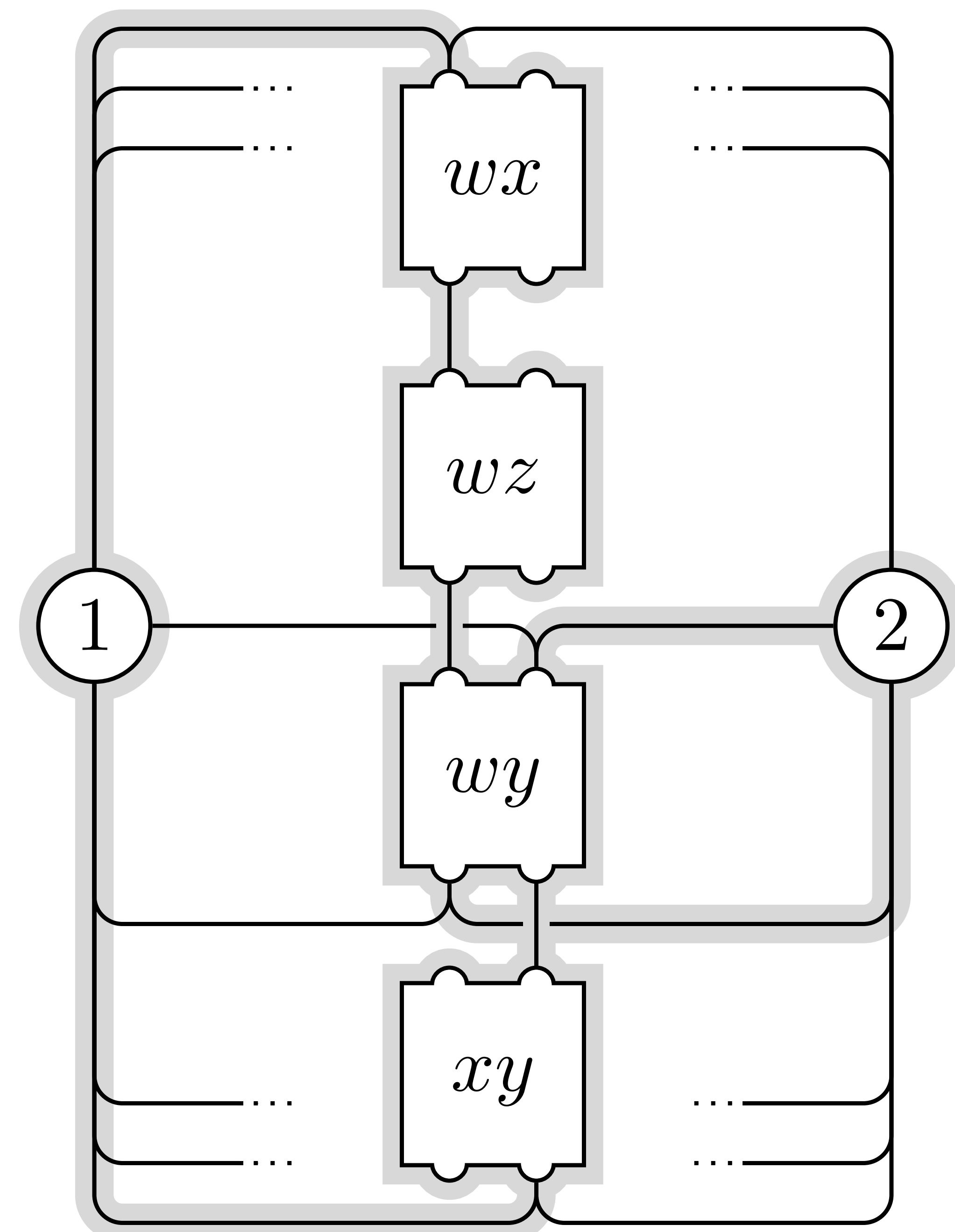
$k = 2$



Finnes et k-dekke? $\Longleftrightarrow$ Finnes en Hamilton-sykel?



$k = 2$



Finnes et k-dekke? $\Longleftrightarrow$ Finnes en Hamilton-sykel?

8:9

TSP

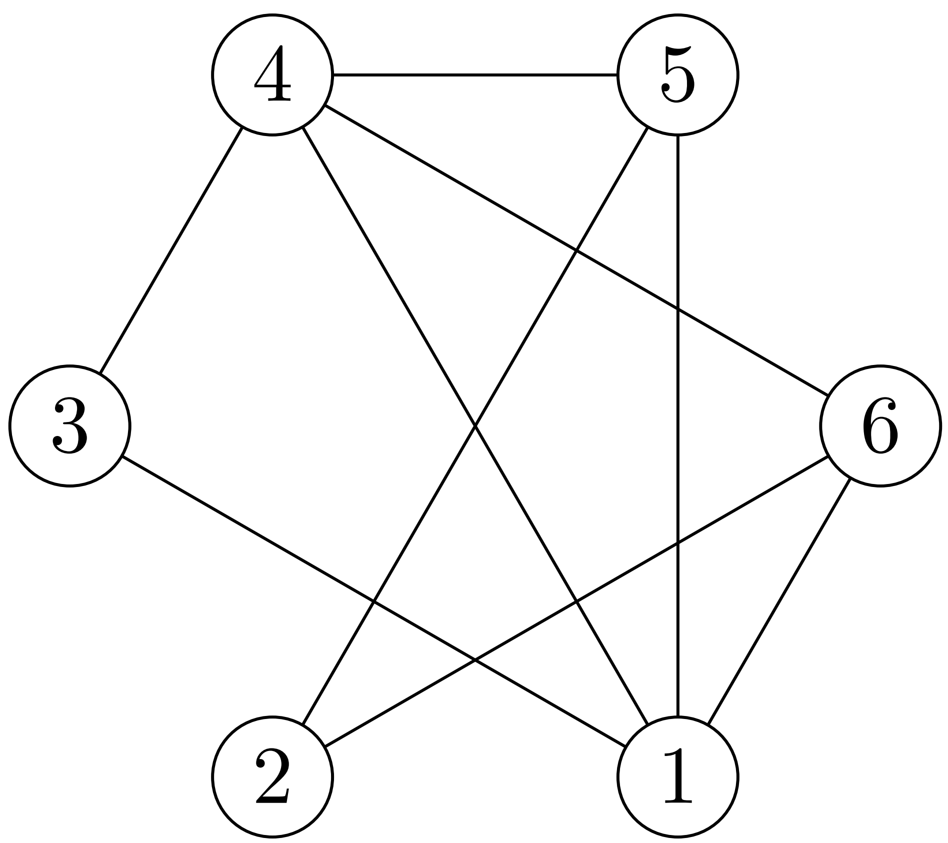
› **TSP**

- **TSP**
  - **Instans:** En komplett graf med heltallsvekter og et heltall k

- **TSP**
  - **Instans:** En komplett graf med heltallsvekter og et heltall k
  - **Spørsmål:** Finnes det en rundtur med kostnad $\leq k$?

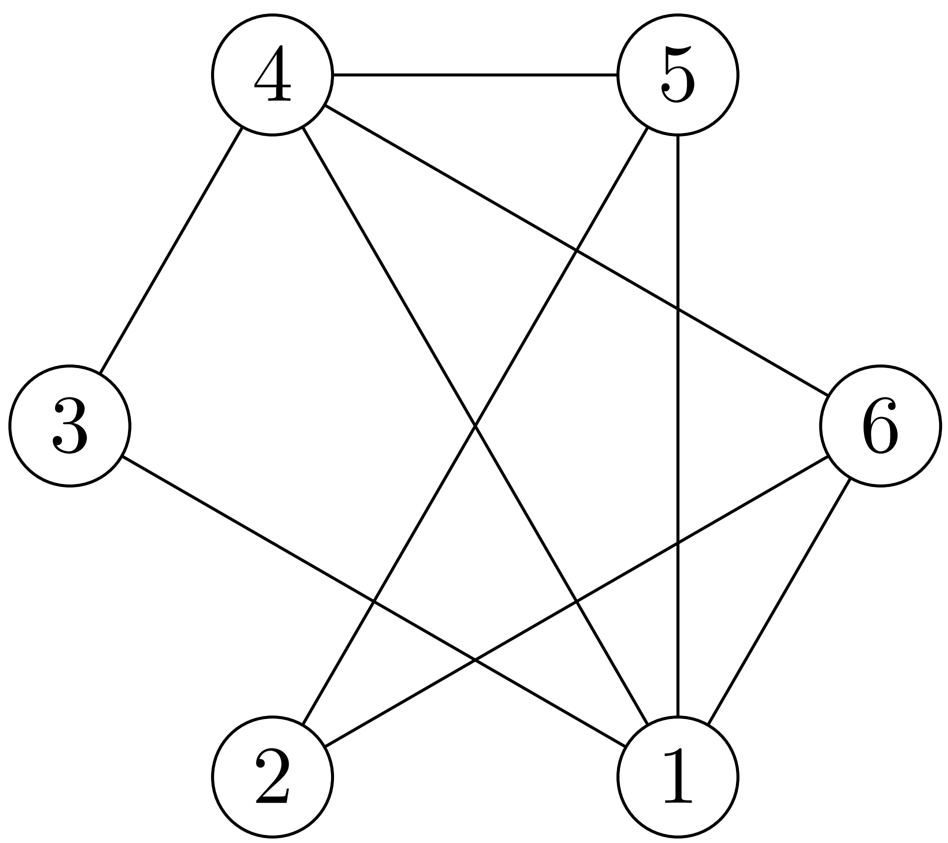
- **TSP**
  - **Instans:** En komplett graf med heltallsvekter og et heltall k
  - **Spørsmål:** Finnes det en rundtur med kostnad $\leq k$?
- Billigste Hamilton-sykel!

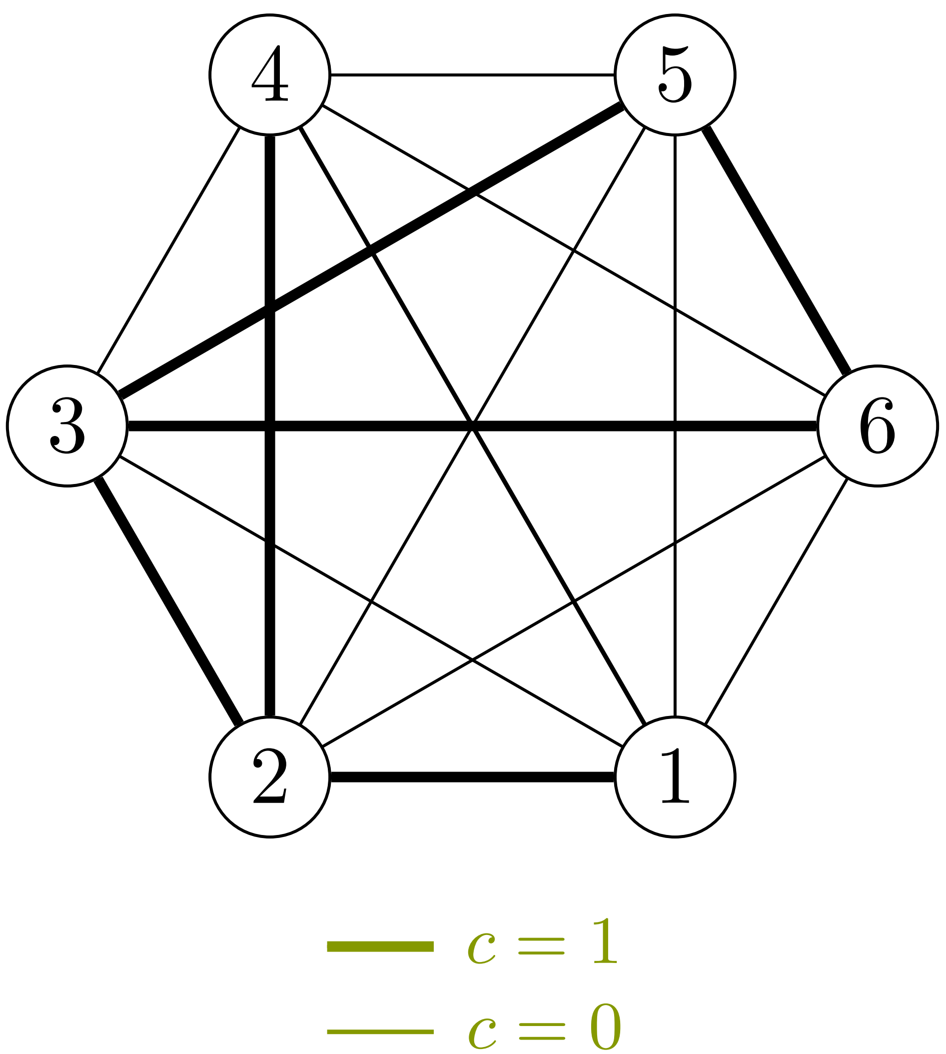
- **TSP**
  - **Instans:** En komplett graf med heltallsvekter og et heltall k
  - **Spørsmål:** Finnes det en rundtur med kostnad $\leq k$?
- Billigste Hamilton-sykel!
- Trenger bare gjøre originalgrafen veldig billig



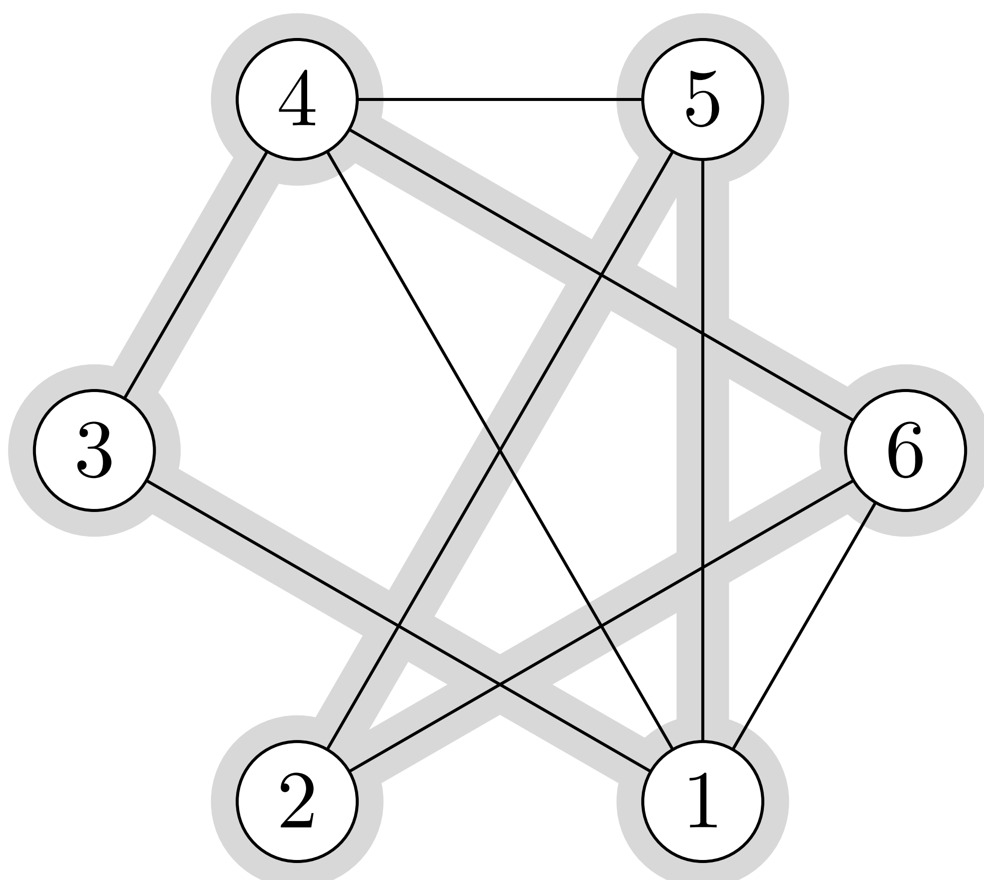
Finnes en Hamilton-sykel? $\iff$ Finnes tur m/kostnad 0?

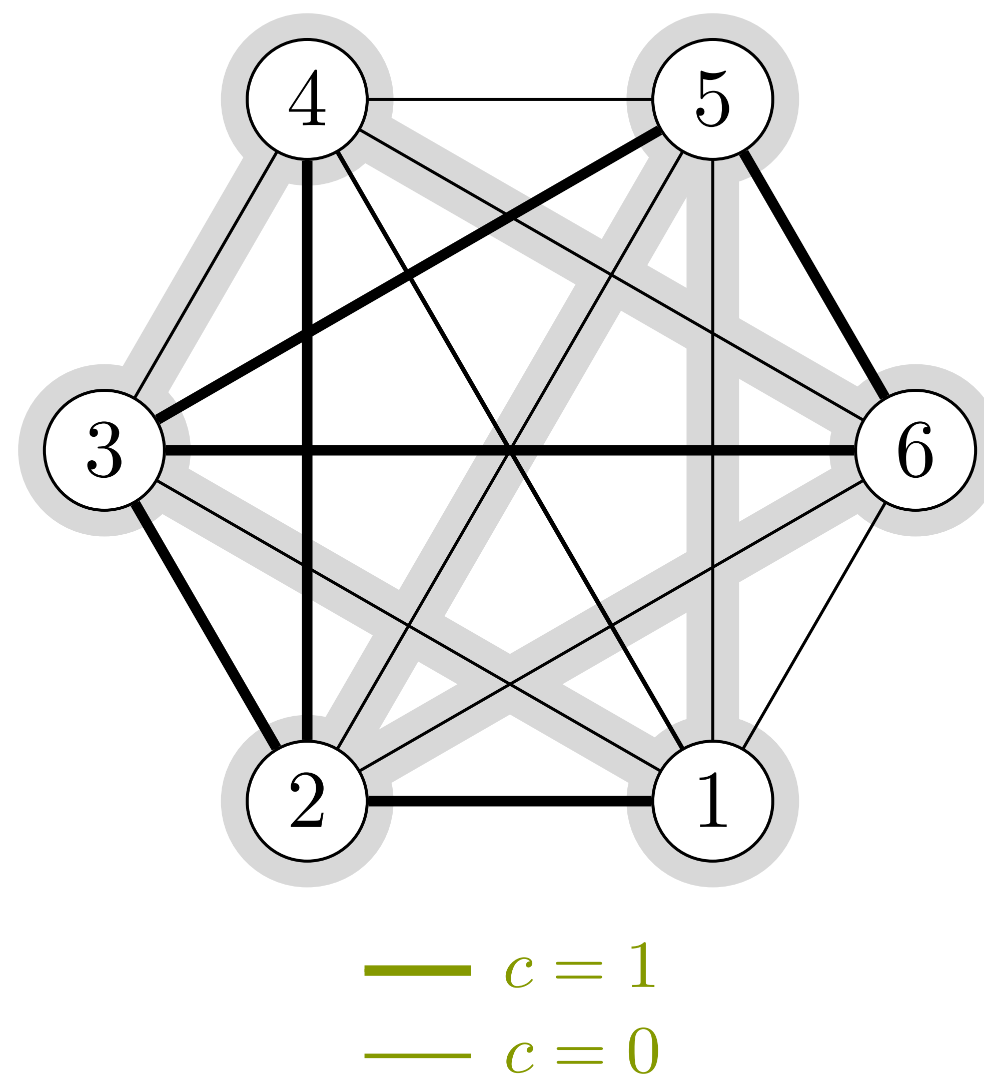
$\bullet \longmapsto \bullet$





Finnes en Hamilton-sykel? $\Longleftrightarrow$ Finnes tur m/kostnad 0?



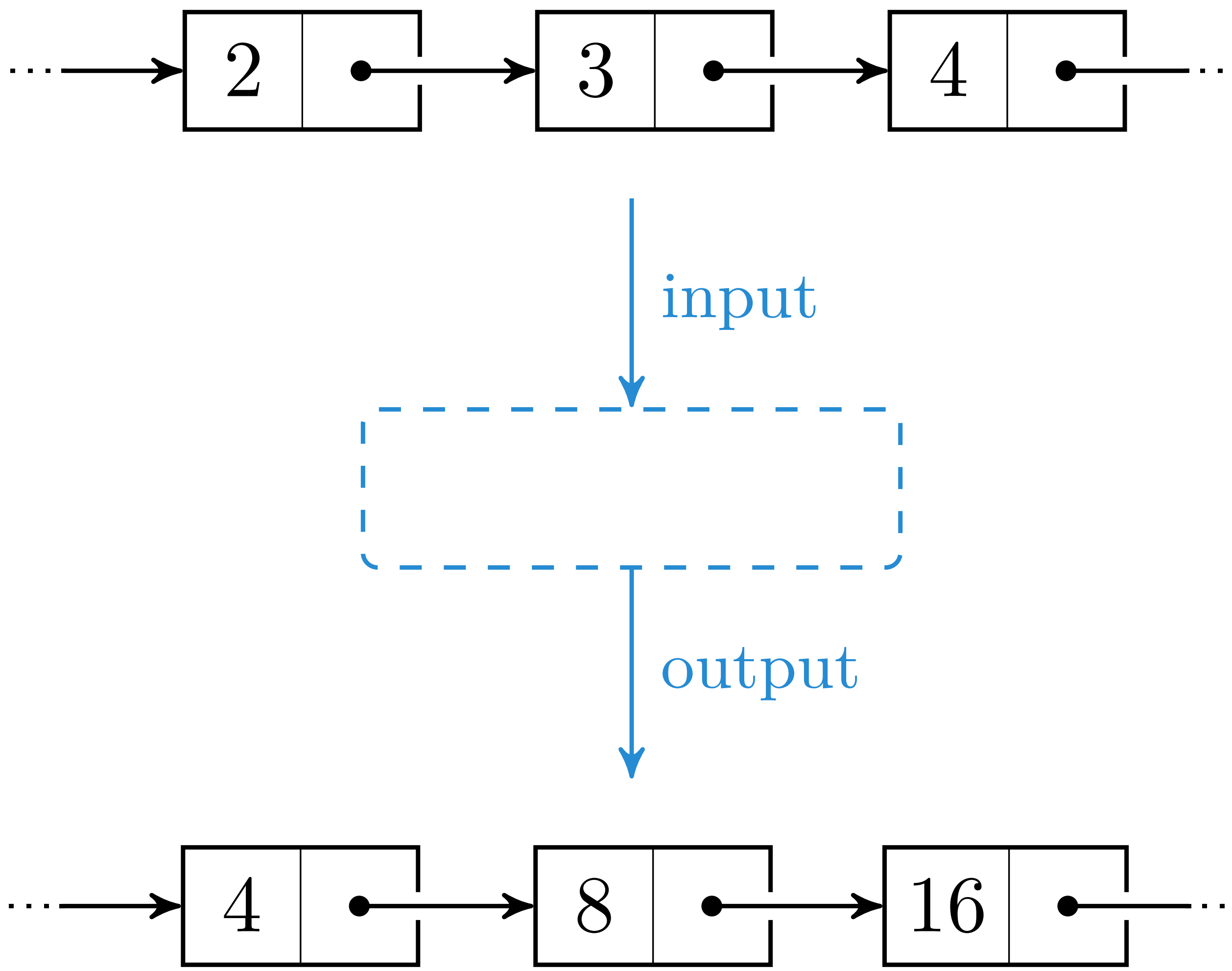


Finnes en Hamilton-sykel? $\iff$ Finnes tur m/kostnad 0?

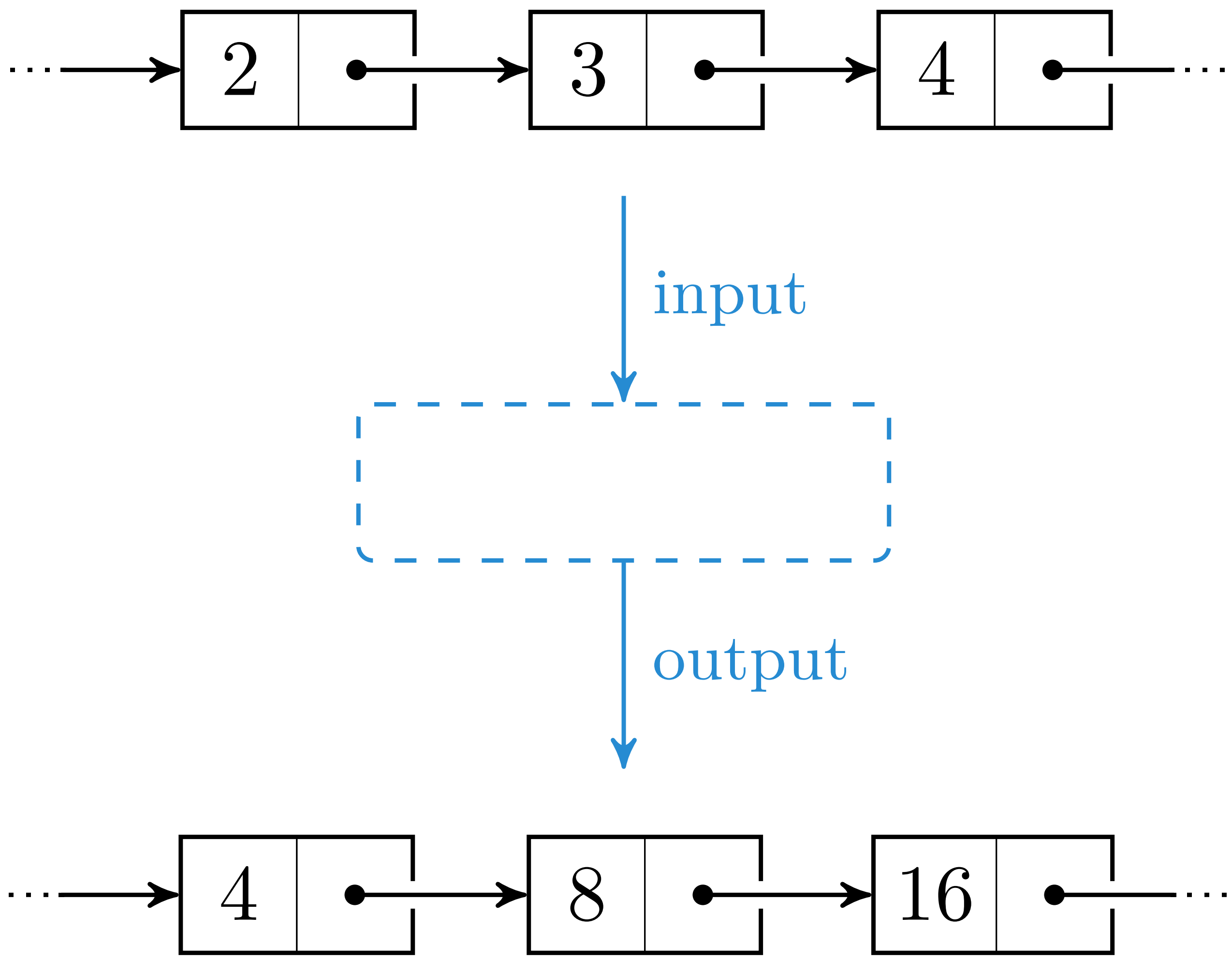
9:9

En del detaljer

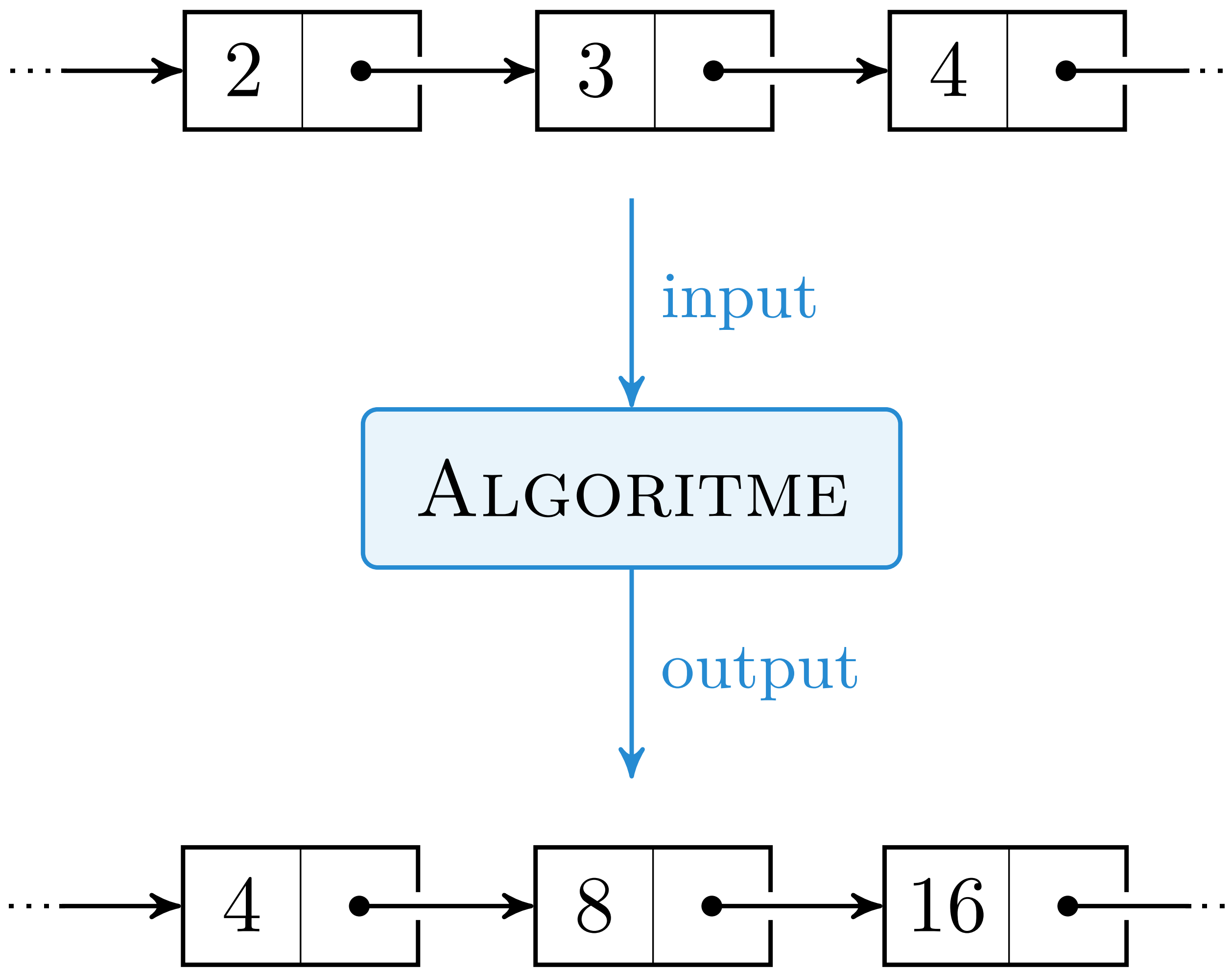
Problemer



Et problem er en relasjon mellom input og output



Input og output kan være vilkårlige abstrakte objekter



Jobben vår er å produsere gyldig output

$01110111011011101011\cdots$

input

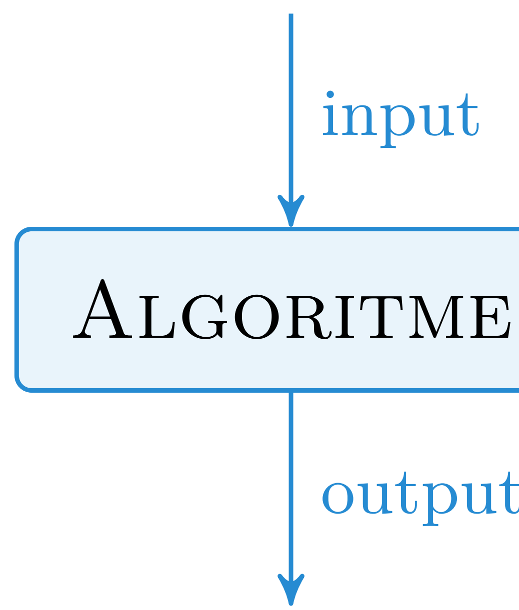
Algoritme

output

$00111101000001000011\cdots$

Et problem kalles <u>konkret</u> hvis input og output er bitstrenger

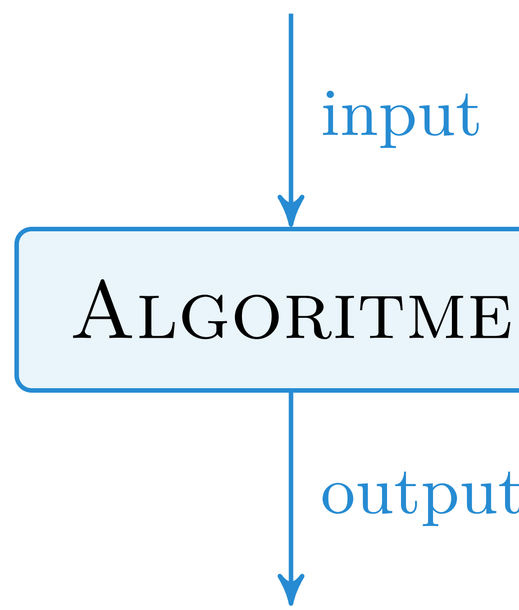
$0111011101101110101 1\cdots$



$0011110100000100001 1\cdots$

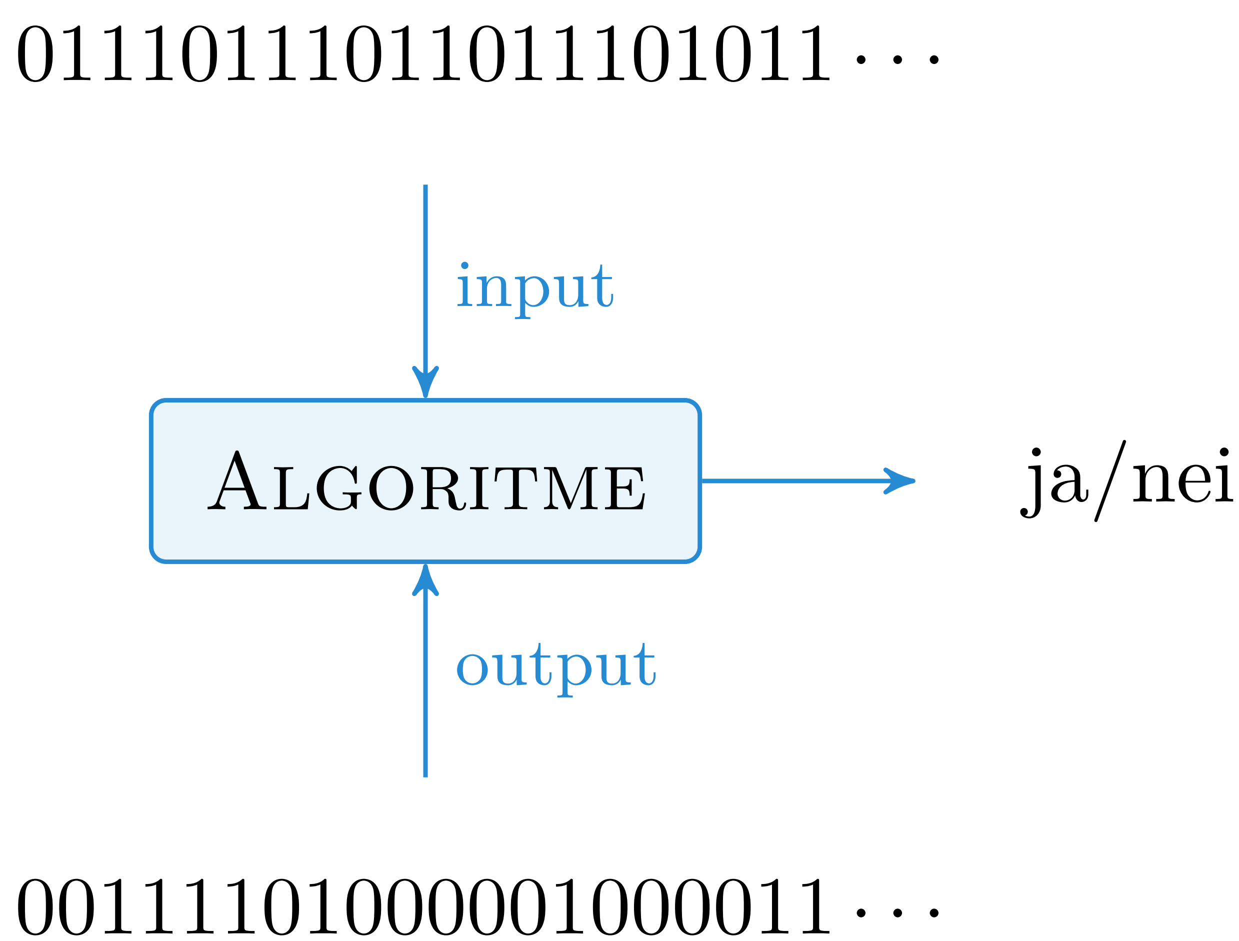
Vi koder instanser og resultater som bits

$0111011101101110 1011\cdots$

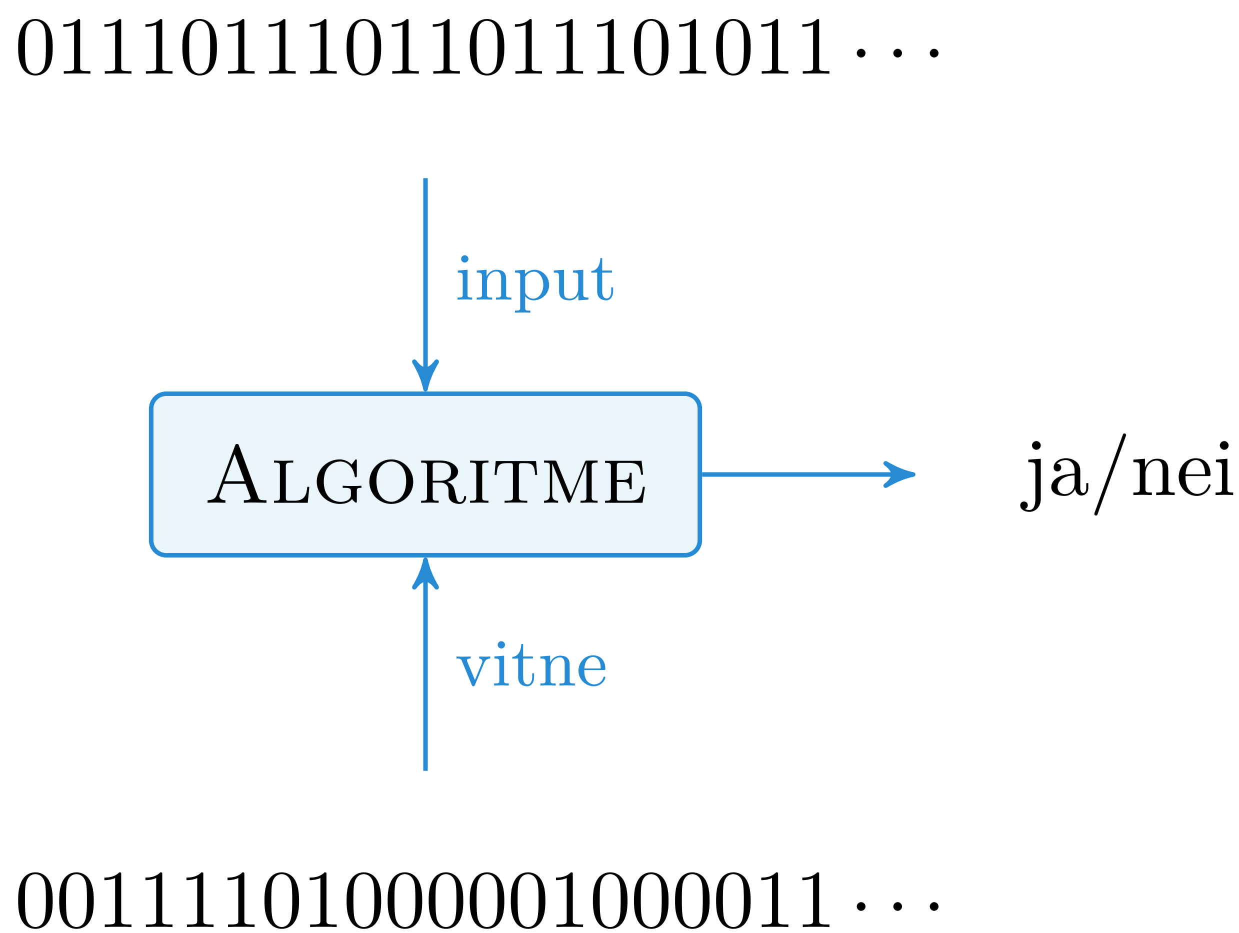


$0011110100000 1000011\cdots$

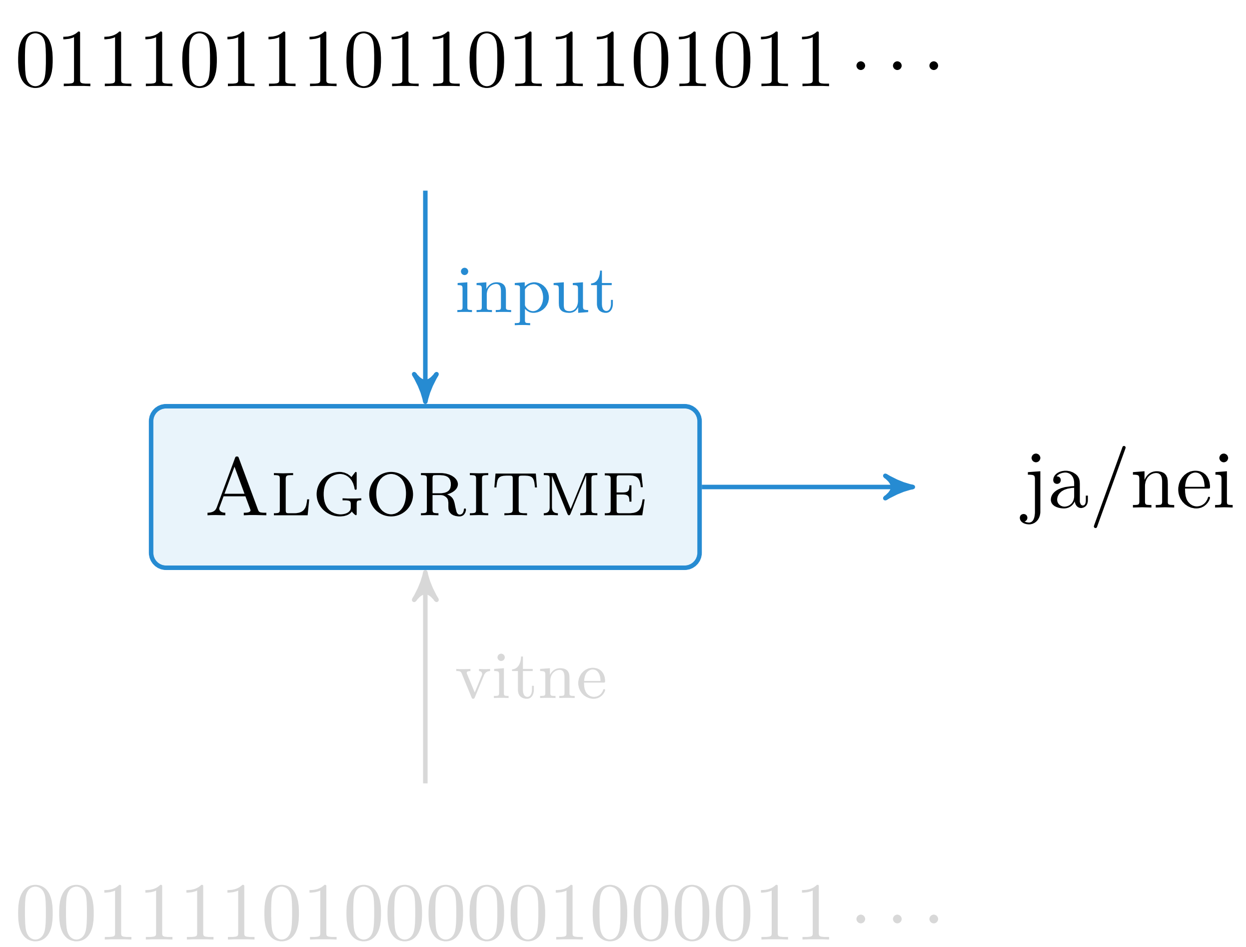
Egentlig ikke så viktig; vi vil bare forenkle «universet» vårt



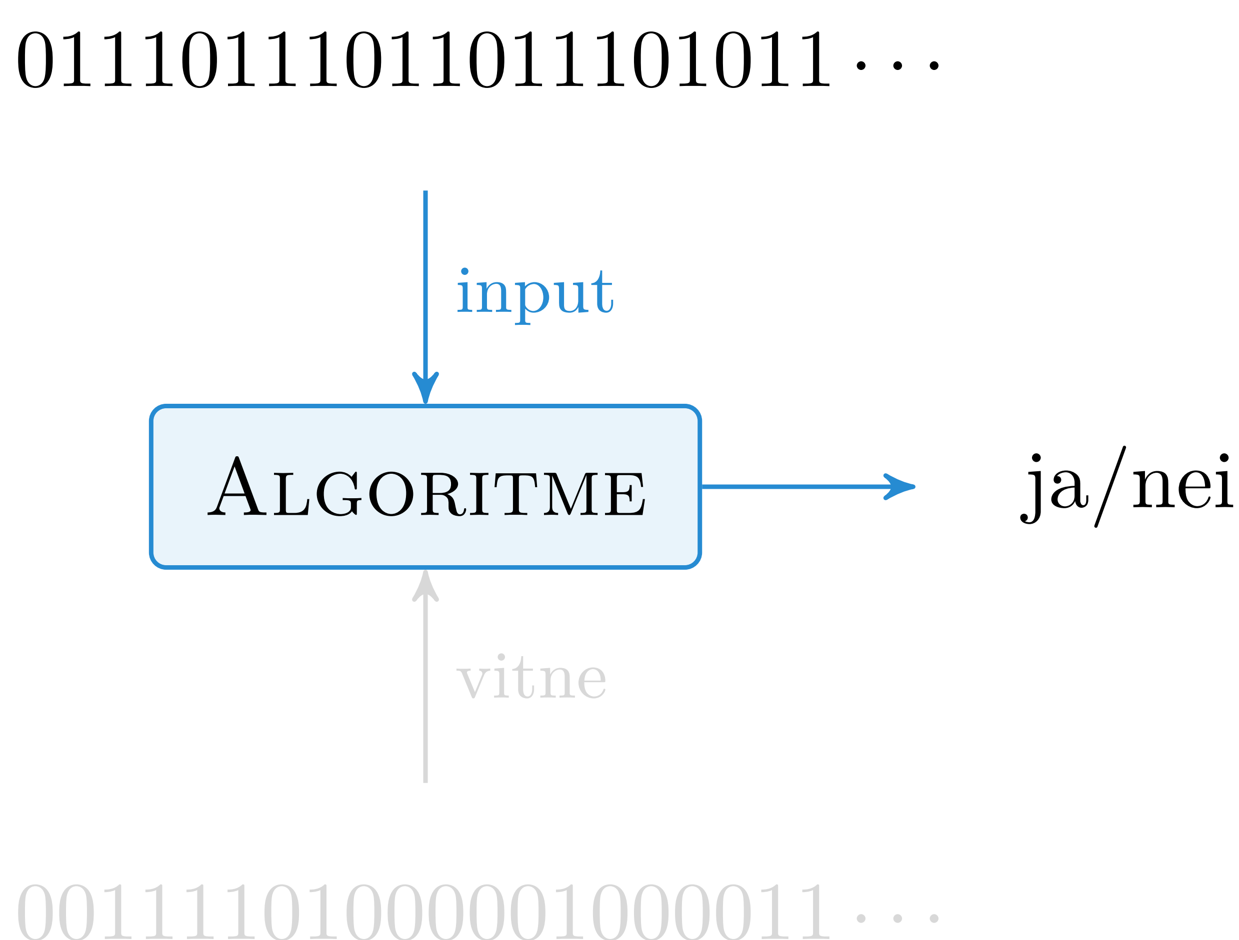
En <u>verifikasjonsalgoritme</u> sjekker om en løsning stemmer



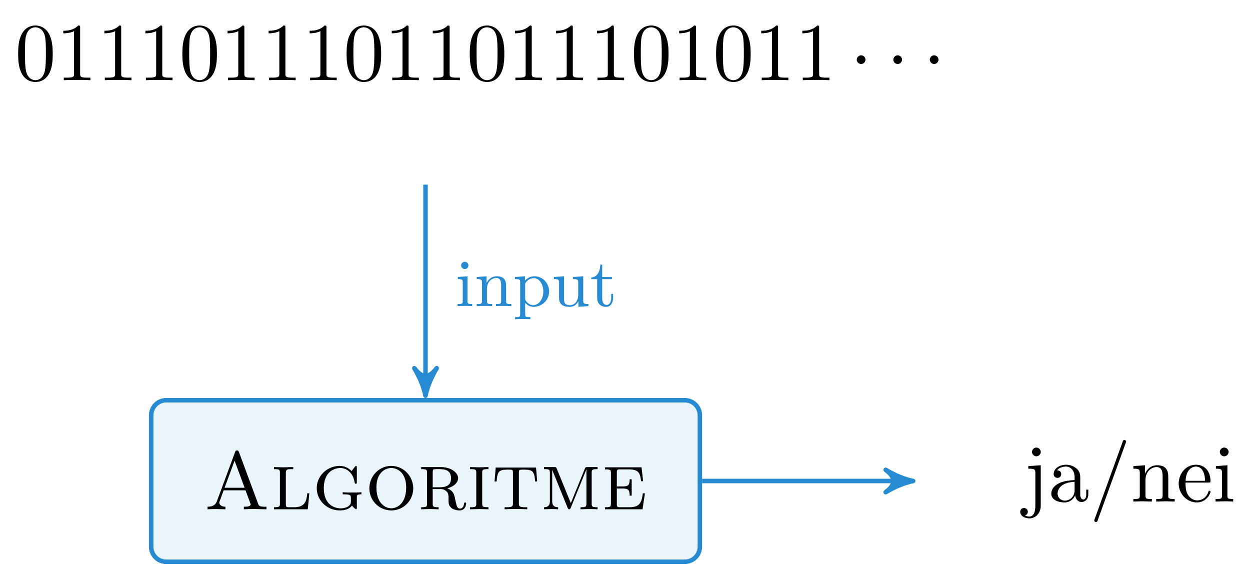
Vi kaller da gjerne løsningen et «sertifikat» eller «vitne»



Et beslutningsproblem kan vi tenke på som å stille spørsmålet . . .



«Finnes det et vitne?»



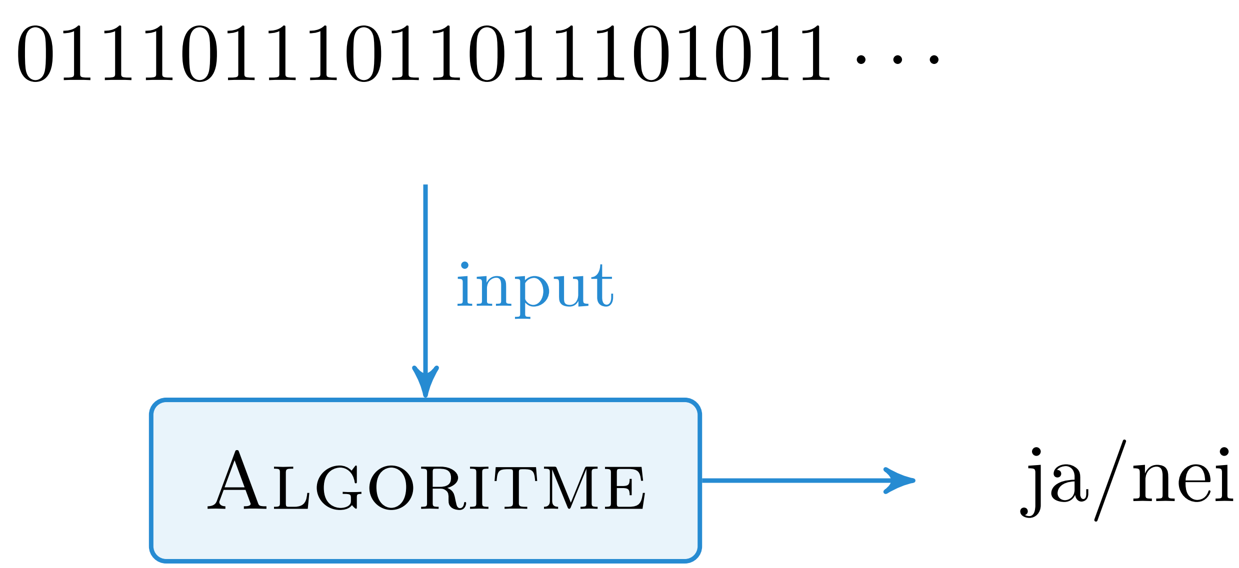
Mer generelt: Egentlig et hvilket som helst ja/nei-spørsmål

$$0111011101101110101 1\cdots$$

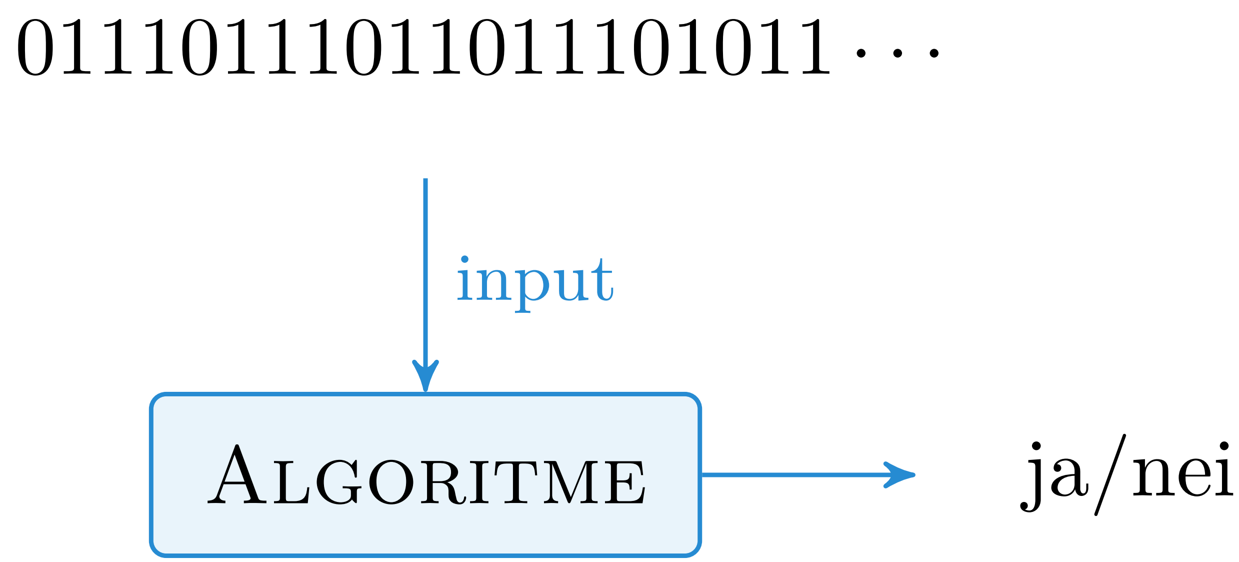
input

ALGORITME → ja/nei

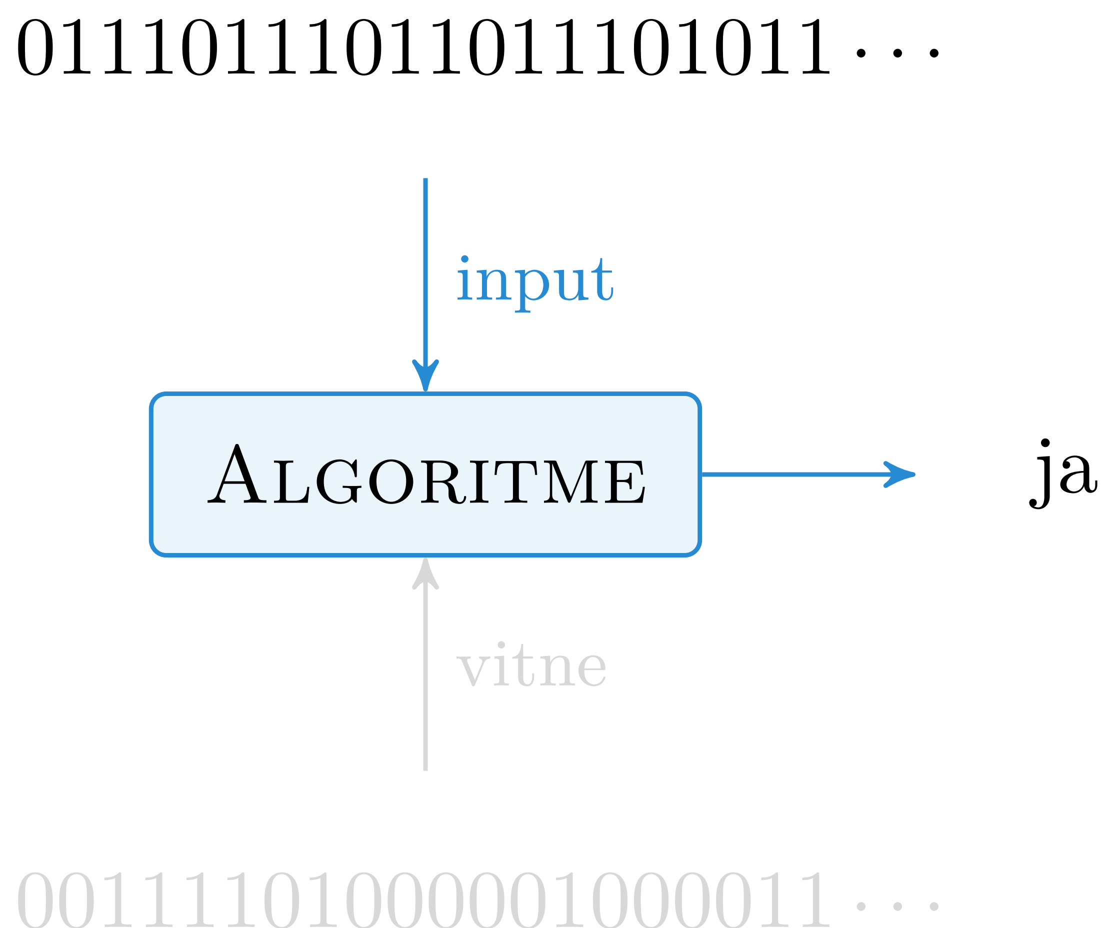
Mer generelt: Egentlig et … som helst ja/nei-spørsmål



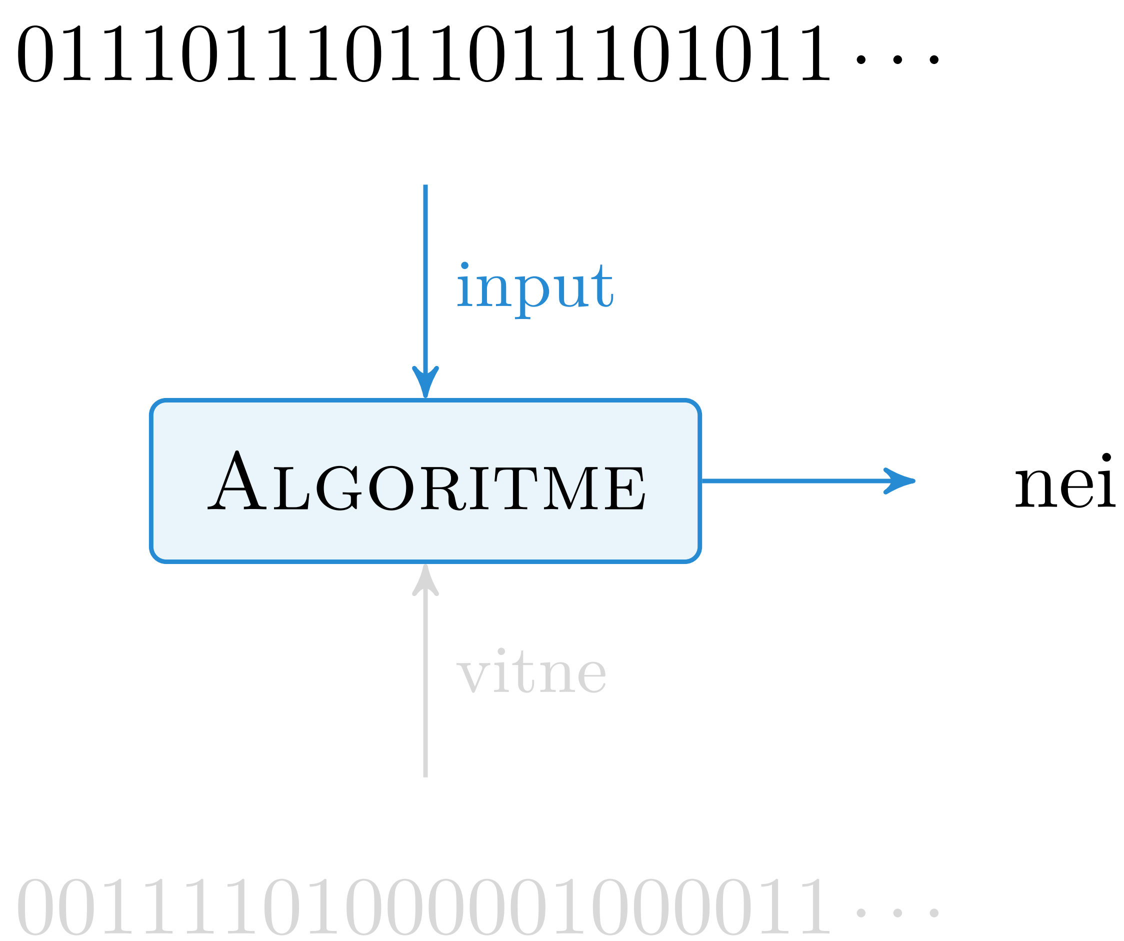
Mer generelt: Egentlig et hvilket som helst ja/nei-spørsmål



Klassen **P** er slike problemer som kan løses i polynomisk tid



NP: Ja-svar har vitner som kan sjekkes i pol. tid



co-NP: Nei-svar har vitner som kan sjekkes i pol. tid

- **Optimering: Ikke nødvendigvis noe vitne**
- **Lag beslutningsproblem med terskling**
- **Avgjøres vha. optimering, som da er minst like vanskelig**
- **Hvis P = NP kan vi finne optimum vha. binærsøk med terskelen**

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom …

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom . . .
  - $x \in \mathrm{L} \to \mathrm{A}(x) = 1$

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom . . .
  - $x \in \mathrm{L} \to \mathrm{A}(x) = 1$
  - $x \notin \mathrm{L} \to \mathrm{A}(x) = 0$.

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom . . .
  - $x \in \mathrm{L} \to \mathrm{A}(x) = 1$
  - $x \notin \mathrm{L} \to \mathrm{A}(x) = 0.$
- **Accept vs decide:**
  Selv om L er språket som *aksepteres* av A, så trenger ikke A *avgjøre* L, siden den kan la være å svare for nei-instanser (ved å aldri terminere)

› **Kompleksitetsklasse:** En mengde språk

- **Kompleksitetsklasse:** En mengde språk
- **P:** Språkene som kan avgjøres i polynomisk tid

- **Kompleksitetsklasse:** En mengde språk
- **P:** Språkene som kan avgjøres i polynomisk tid

Pussig nok, også språkene som aksepteres i pol. tid! (Thm. 34.2)

- **Kompleksitetsklasse:** En mengde språk
- **P:** Språkene som kan avgjøres i polynomisk tid
- **Cobham's tese:**
  Det er disse problemene vi kan løse i praksis

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x
- En algoritme A **verifiserer** x hvis det eksisterer et sertifikat y slik at $\mathrm{A}(x, y) = 1$

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x
- En algoritme A **verifiserer** x hvis det eksisterer et sertifikat y slik at $\mathrm{A}(x, y) = 1$
- **Intuitivt:**
  Algoritmen «sjekker svaret». Om en graf har en Hamilton-sykel, kan sertifikatet være noderekkefølgen i sykelen.

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x
- En algoritme A **verifiserer** x hvis det eksisterer et sertifikat y slik at $\mathrm{A}(x, y) = 1$
- **Intuitivt:**
  Algoritmen «sjekker svaret». Om en graf har en Hamilton-sykel, kan sertifikatet være noderekkefølgen i sykelen.
- **Asymmetrisk:**
  Det finnes ikke «motbevis» eller «anti-sertifikater»

› **NP:** Språkene som kan verifiseres i polynomisk tid

› **NP:** Språkene som kan verifiseres i polynomisk tid

N = Nondeterministic: Kan løses om vi klarer «gjette» sertifikater

› **NP:** Språkene som kan verifiseres i polynomisk tid

› **HAM-CYCLE**
Språket for Hamilton-sykel-problemet

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

$\bar{\mathrm{L}}$ er *komplementet* til L: $x \in \bar{\mathrm{L}} \iff x \notin \mathrm{L}$

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

F.eks.: TAUTOLOGY

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

F.eks.: TAUTOLOGY

(Det er komplementet til negasjonen av SAT … som vi ser igjen siden)

› **P vs NP**

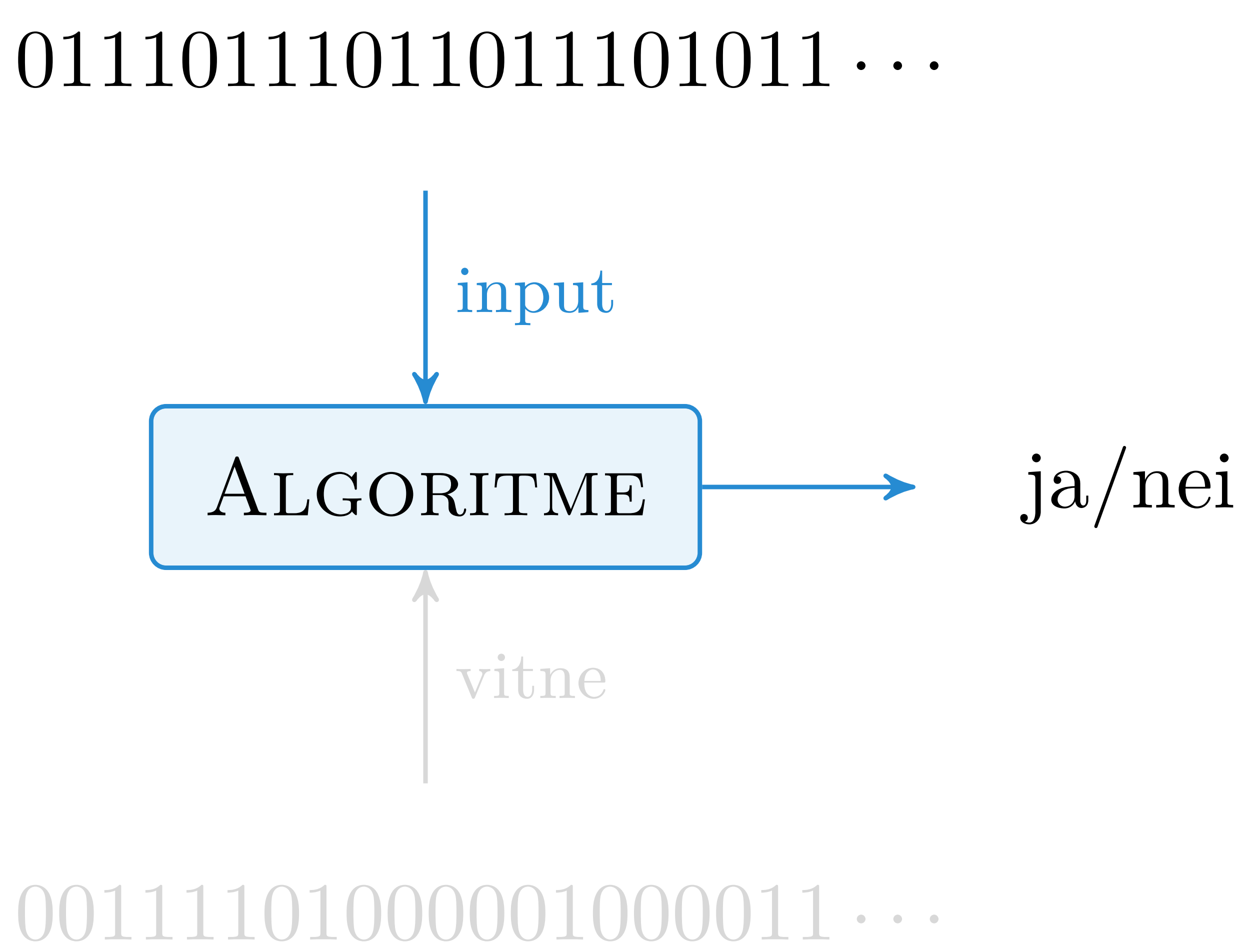
Om vi kan *løse* problemet, så kan vi *verifisere* det med samme algoritme, og bare ignorere sertifikatet

Dvs.: $\mathbf{P} \subseteq \mathbf{NP}$ og $\mathbf{P} \subseteq \mathbf{co\text{-}NP}$

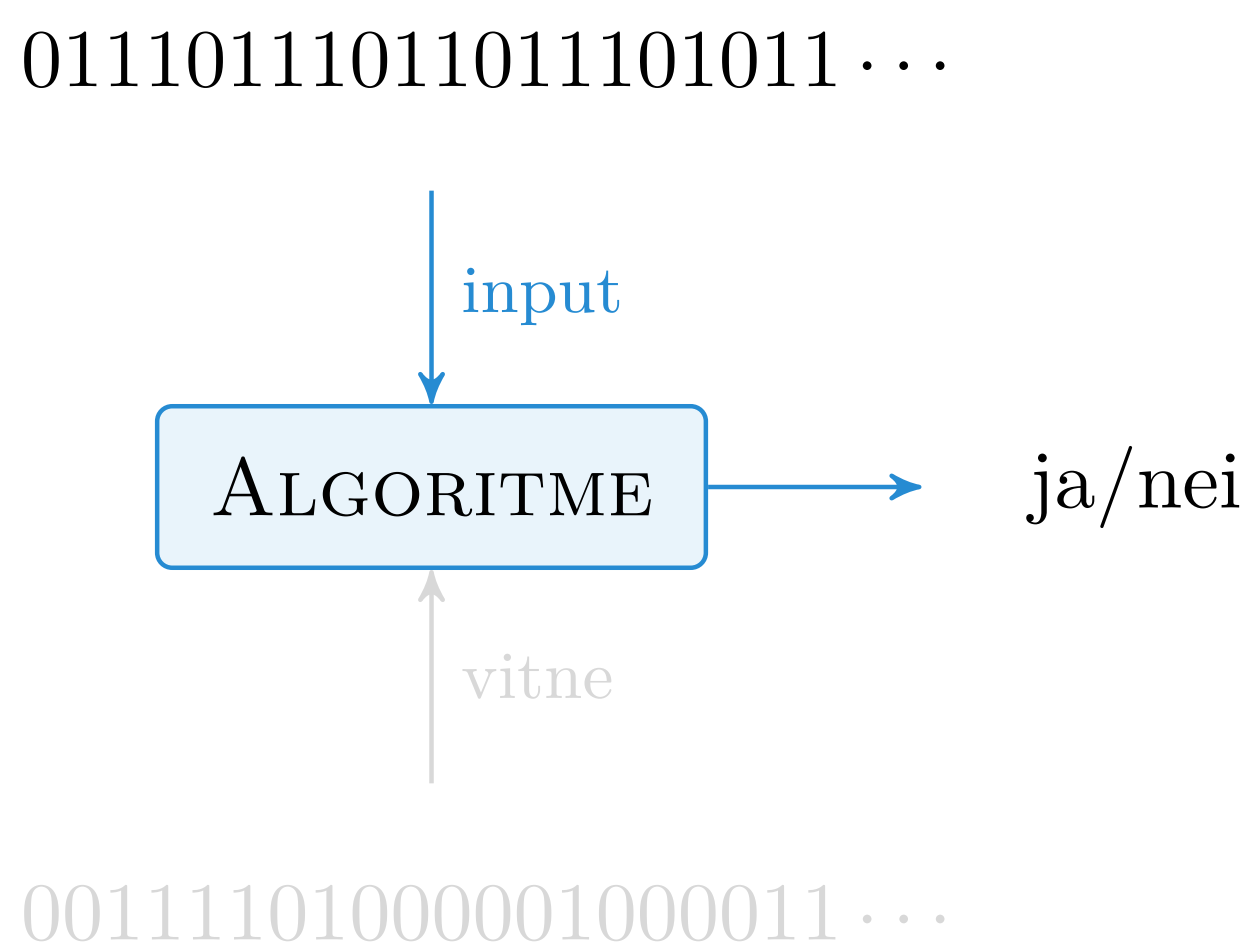
› **P vs NP**
Om vi kan *løse* problemet, så kan vi *verifisere* det med samme algoritme, og bare ignorere sertifikatet
Dvs.: $\mathbf{P} \subseteq \mathbf{NP}$ og $\mathbf{P} \subseteq \textbf{co-NP}$

› **Vi vet ikke** om $\mathbf{P} = \mathbf{NP} \cap \textbf{co-NP}$

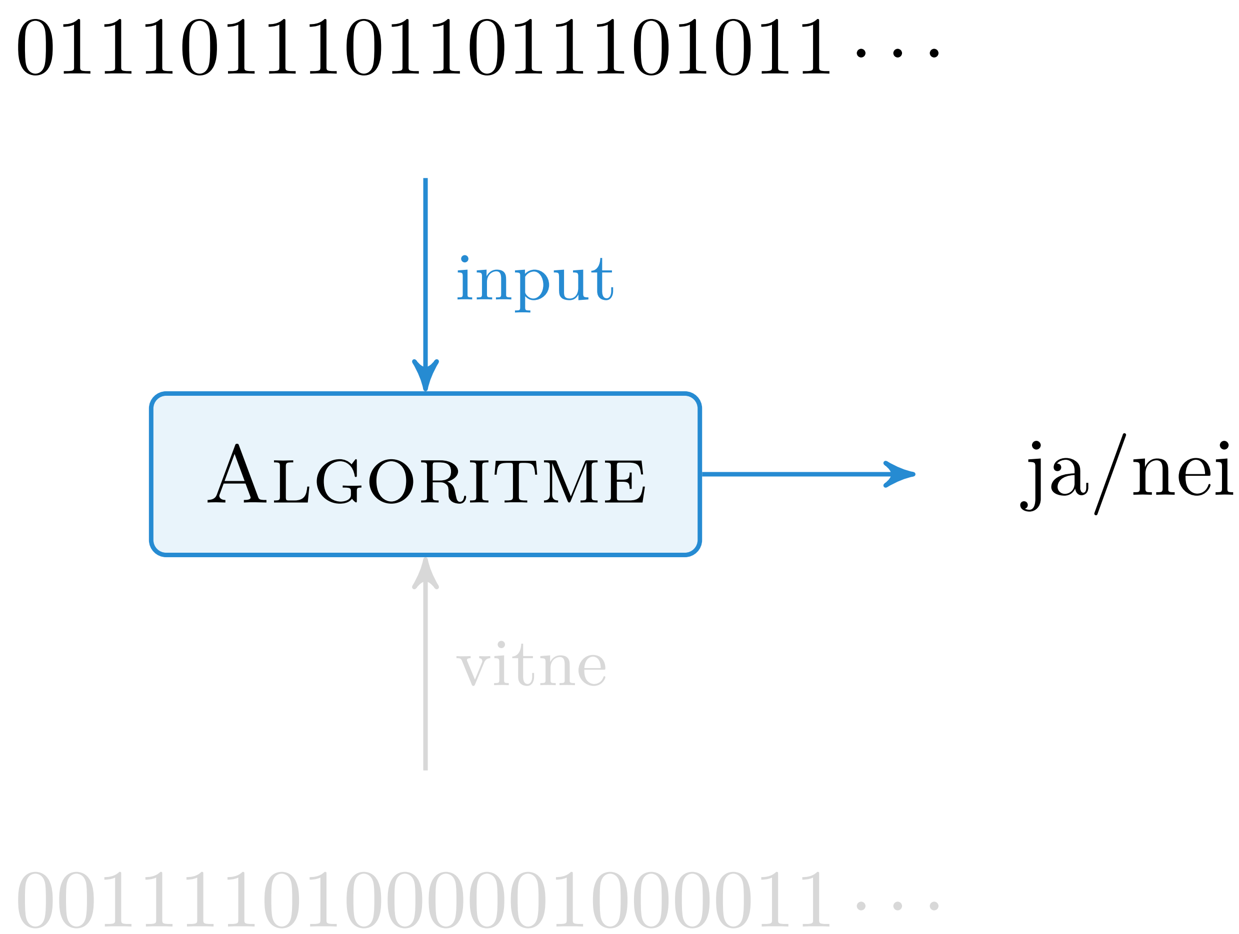
Rekonstruksjon av sertifikater



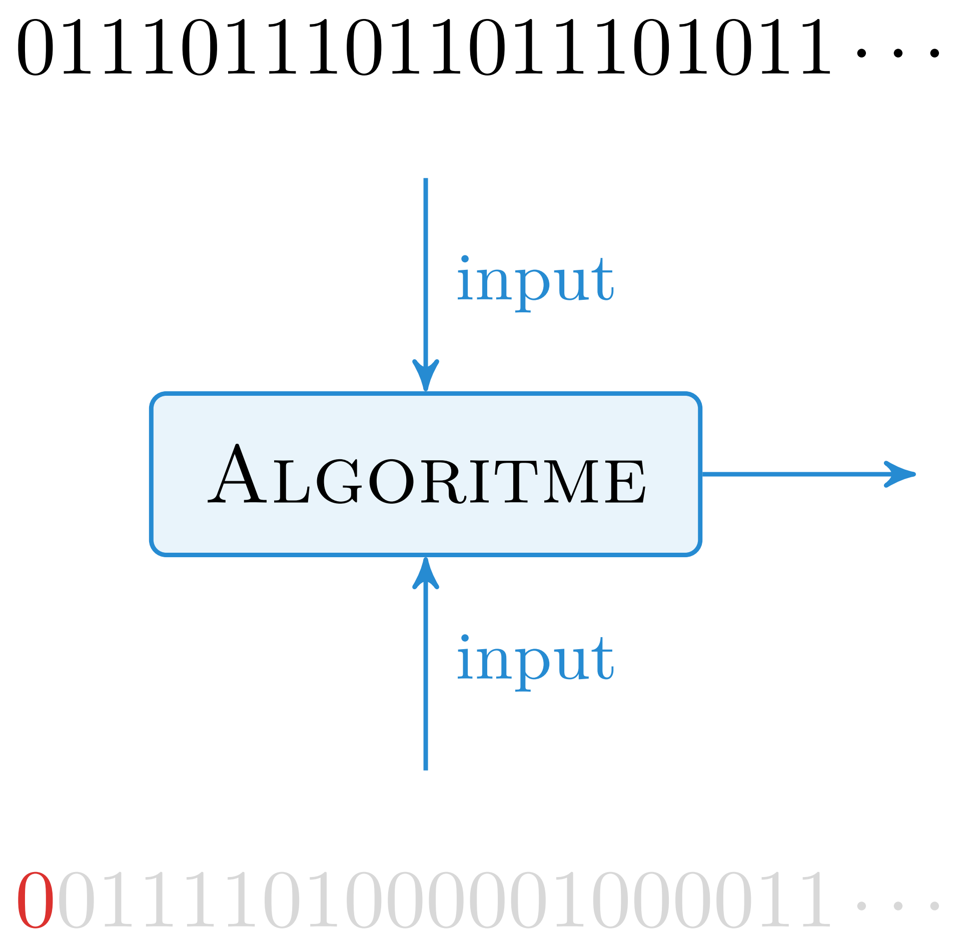
Hvis **P** = **NP** kan vi svare på om det finnes et vitne



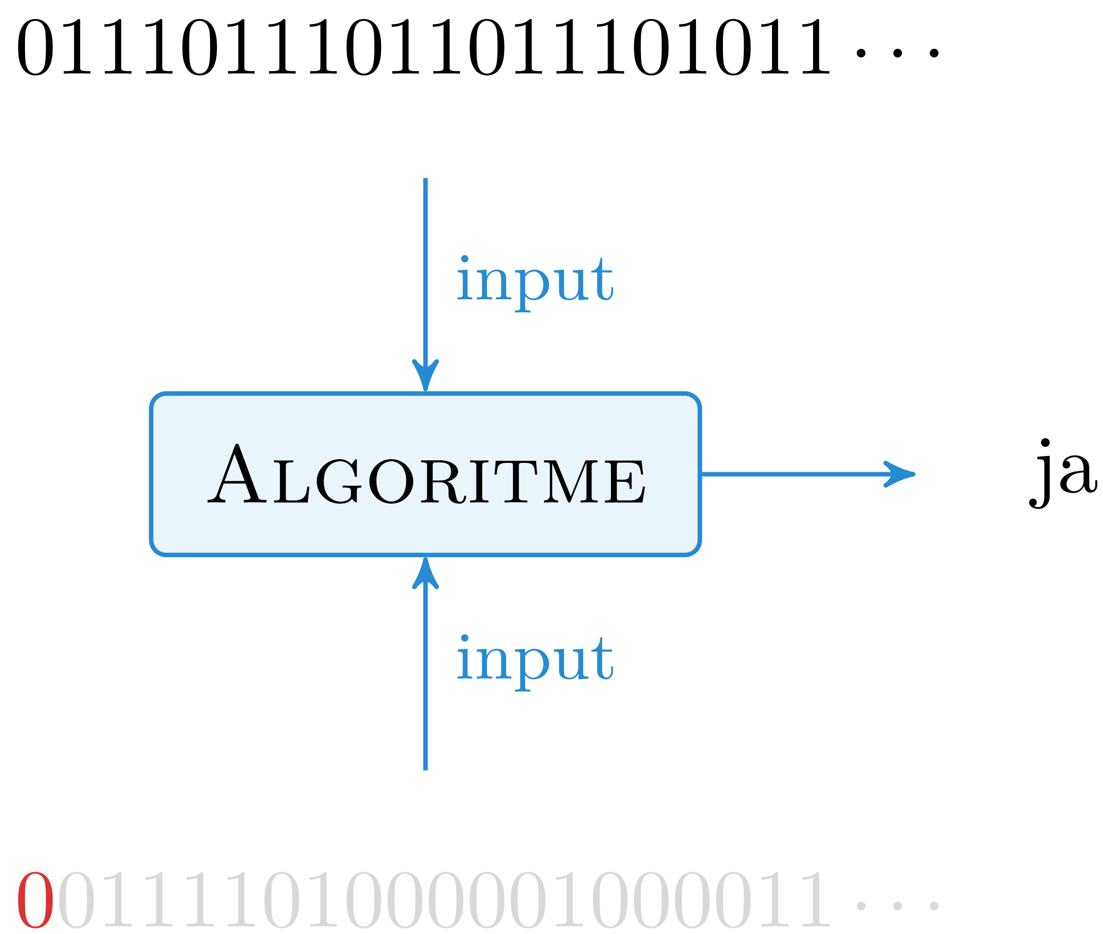
Ikke bare det: Vi kan rekonstruere vitnet!



Vi lager oss nye beslutningsproblemer, med deler av vitnet som input

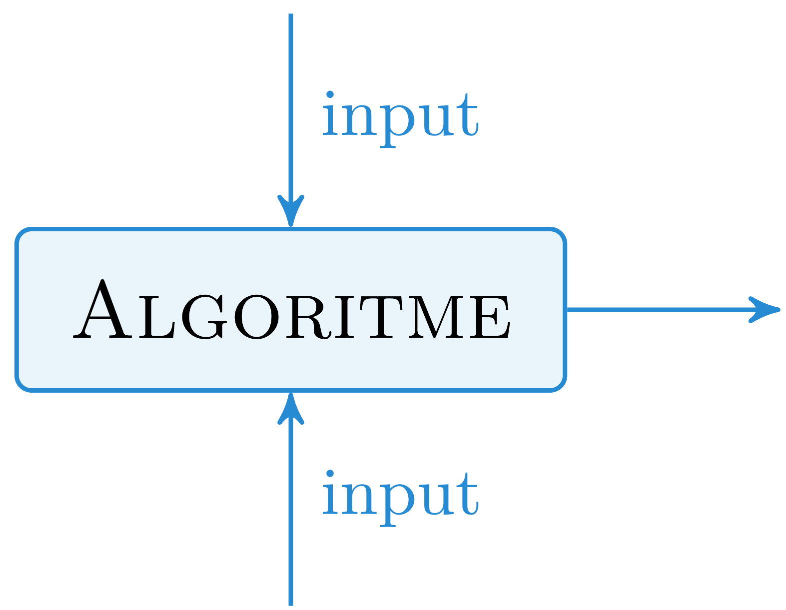


Problem 1: Finnes et vitne som starter med 0?



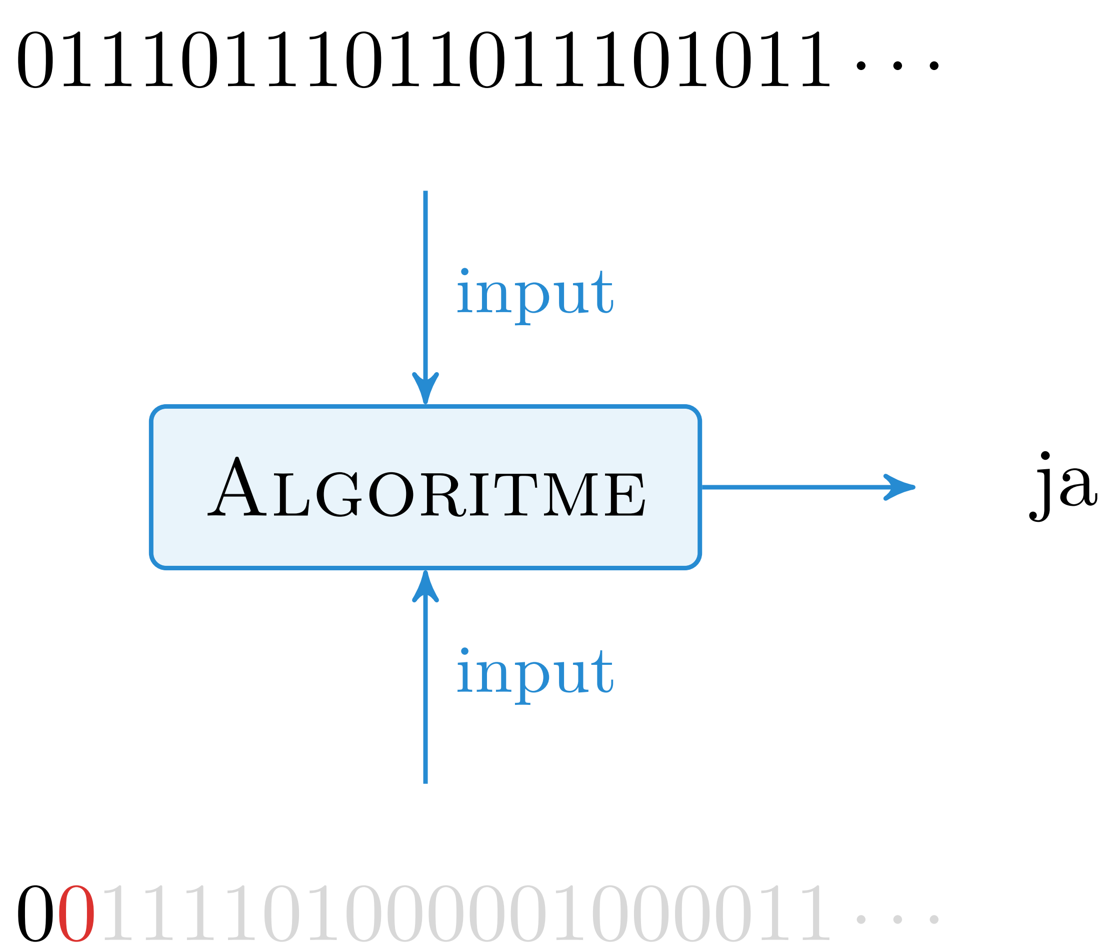
Ja, det gjør det. Vi setter første siffer til 0

$0111011101101110101 1\cdots$

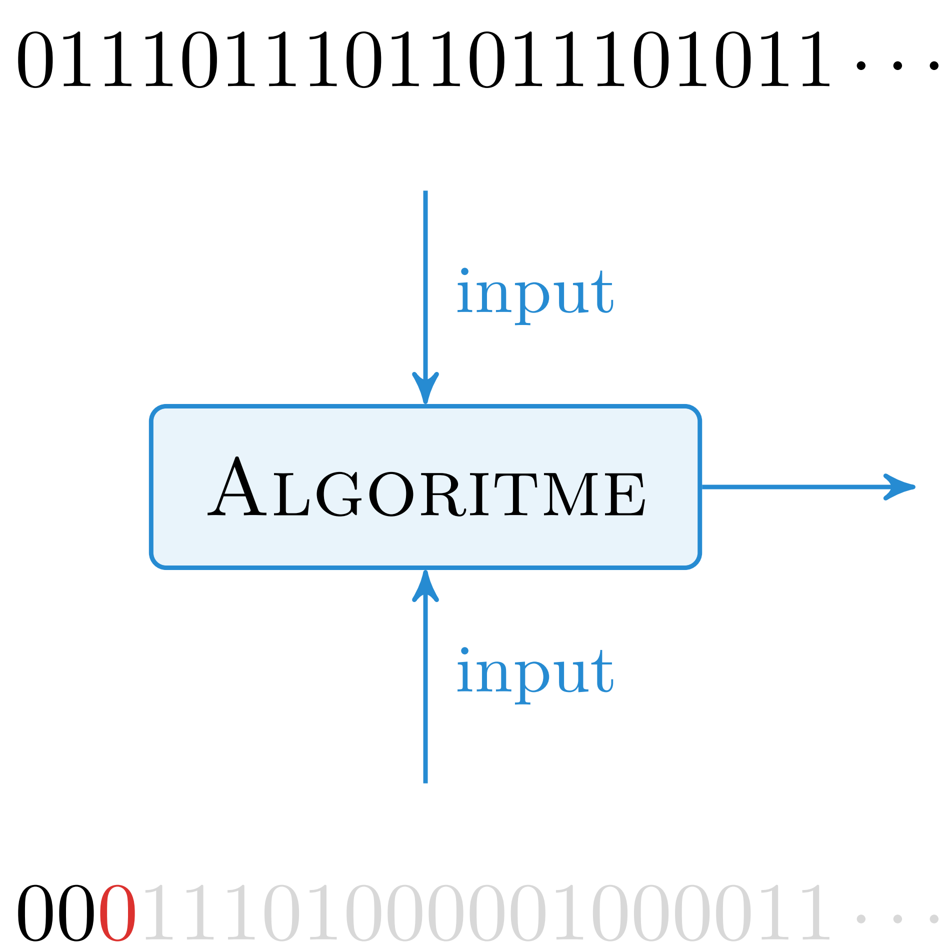


$0\textcolor{red}{0}\textcolor{lightgray}{1111010000010000 11\cdots}$

Problem 2: Finnes et vitne som starter med 00?

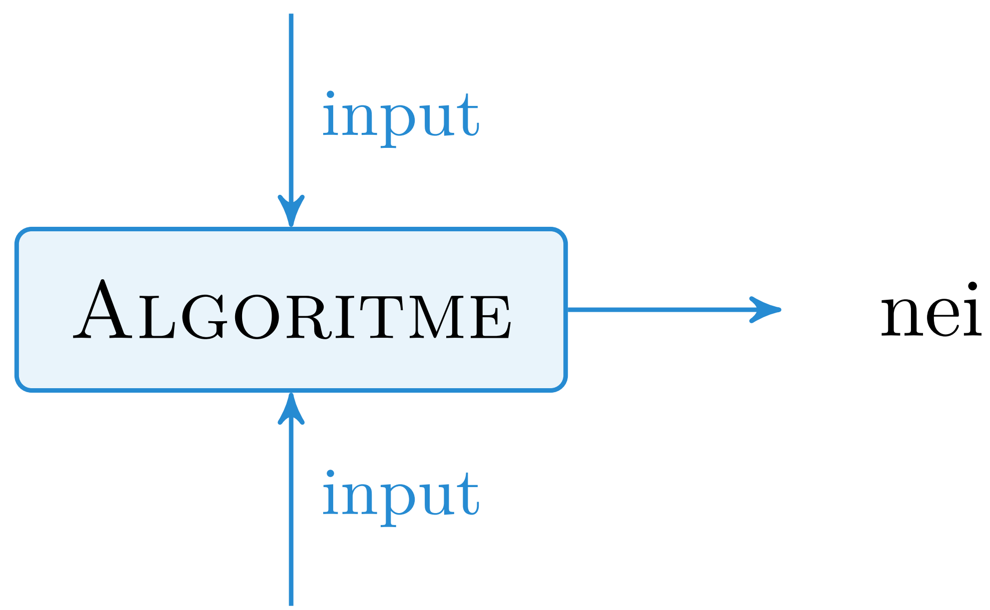


Ja, det gjør det. Vi setter andre siffer til 0



Problem 3: Finnes et vitne som starter med 000?

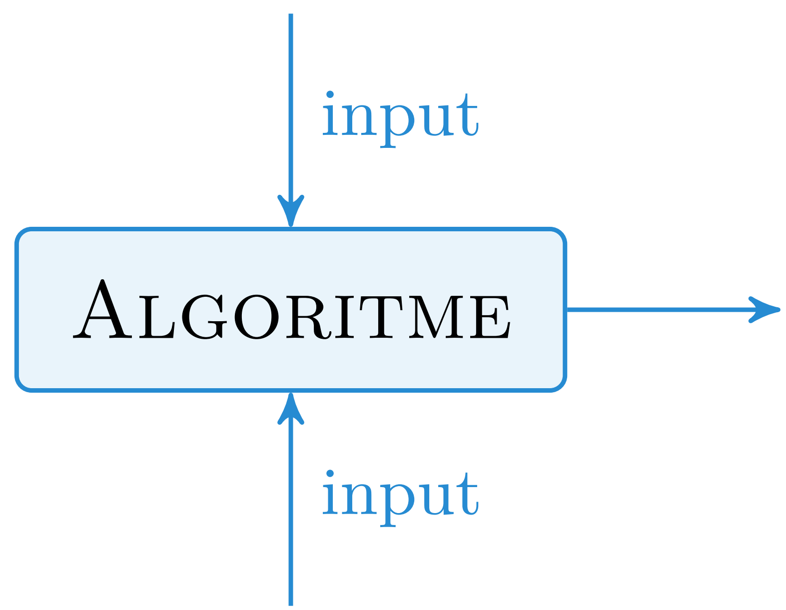
01110111011011101011 · · ·



000111010000001000011 · · ·

Nei, det gjør det ikke. Vi setter tredje siffer til 1

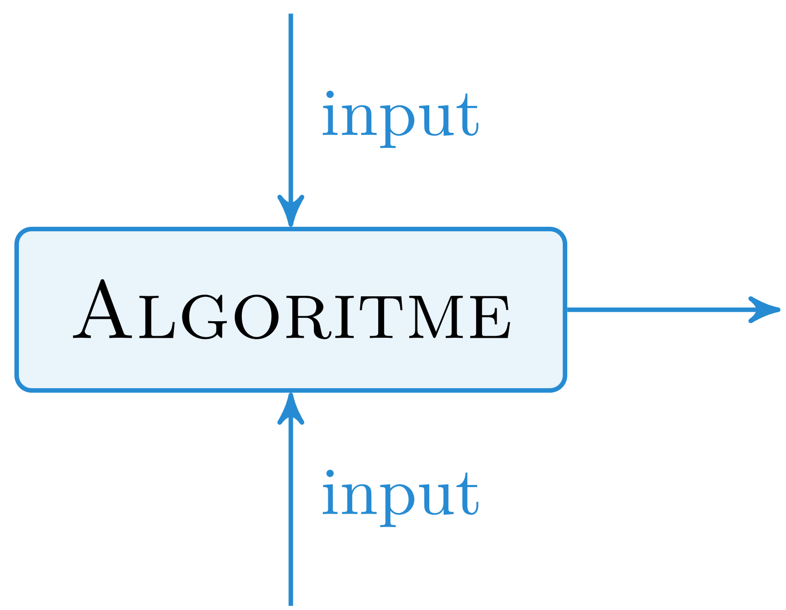
$0111011101101110101 1\cdots$



$00100110100000100001 1\cdots$

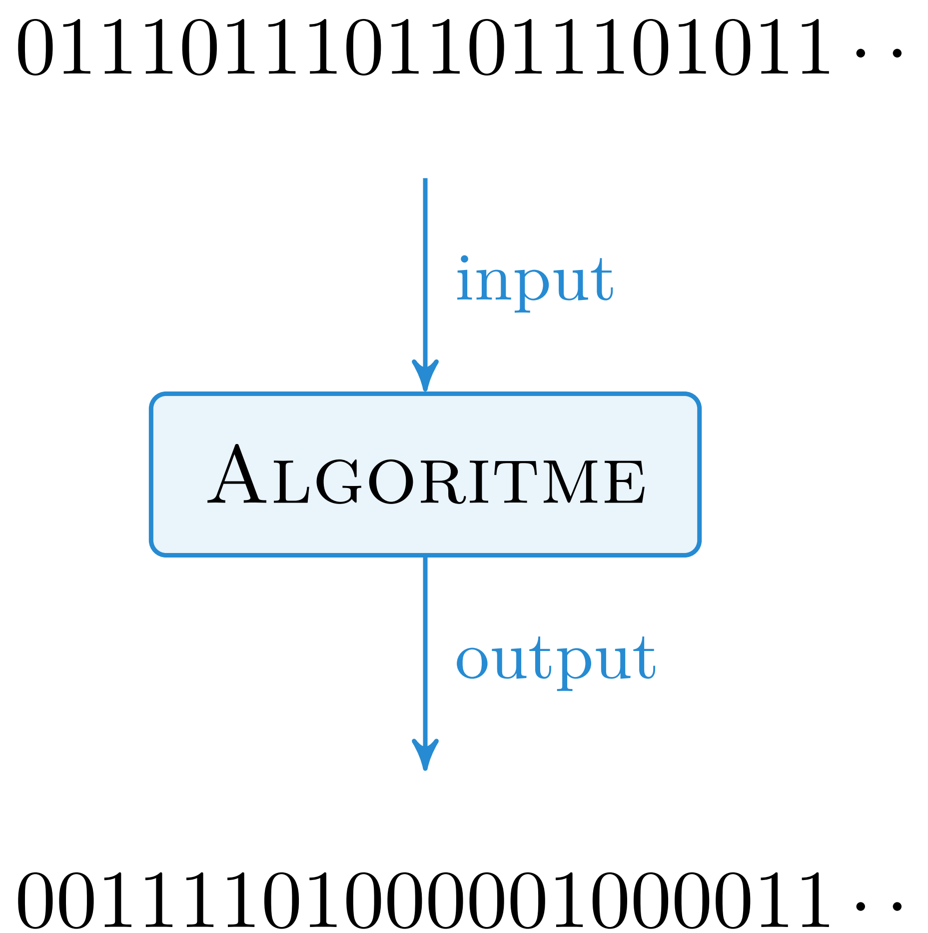
Problem 4: Finnes et vitne som starter med 0010?

$0111011101101110 1011\cdots$



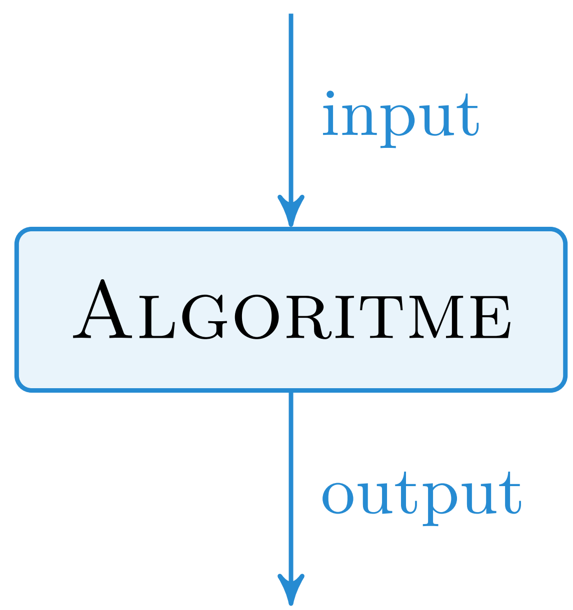
$00111101000001000011\cdots$

Og slik fortsetter vi, til vi har konstruert et vitne



Med andre ord: Om vi kan løse beslutningsproblemer . . .

$01110111011011101011\cdots$



$00111101000001000011\cdots$

…så kan vi løse «søkeproblemer» også, og finne gyldig output!

Litt mer om problemer

- **Abstrakt problem:**
  Binær relasjon $\mathrm{Q} \subseteq \mathrm{I} \times \mathrm{S}$
- **Encoding:**
  Funksjon til $\{0, 1\}^*$
- **Polynomisk kjøretid:**
  $\mathrm{T}(n) = \mathrm{O}(n^k)$, for en eller annen $k > 0$, der n er antall bits i encodingen til instansene.
- **Polynomisk relaterte encodings:**
  Kan du transformere mellom dem i pol. tid, spiller det ingen rolle hvilken du bruker.
- **Rimelige encodings:**
  Unngå spesielt éntallssystemet!

› **0-1 Knapsack:**
Vår DP-løsning har kjøretid $T(n, W) = \Theta(nW)$

Egentlig skal funksjonen ha ett argument; det lar seg gjøre

- **0-1 Knapsack:**
  Vår DP-løsning har kjøretid $T(n, W) = \Theta(nW)$
- **Encoding:**
  For enkelhets skyld, la oss si vi bruker $\Theta(n)$ bits på objektene. En rimelig encoding vil bruke $\Theta(m)$ bits på kapasiteten, der $m = \lg W$.

Objektene tar egentlig $\Theta(n \lg n)$ plass; ikke så viktig her

- **0-1 Knapsack:**
  Vår DP-løsning har kjøretid $T(n, W) = \Theta(nW)$
- **Encoding:**
  For enkelhets skyld, la oss si vi bruker $\Theta(n)$ bits på objektene. En rimelig encoding vil bruke $\Theta(m)$ bits på kapasiteten, der $m = \lg W$.
- **Polynomisk?**
  T er polynomisk som funksjon av n og W, men er den «polynomisk»?

Implisitt: Som funksjon av lengden på instans-encodingen vår

› **0-1 Knapsack:**
Vår DP-løsning har kjøretid $\mathrm{T}(n, \mathrm{W}) = \Theta(n\mathrm{W})$

› **Encoding:**
For enkelhets skyld, la oss si vi bruker $\Theta(n)$ bits på objektene. En rimelig encoding vil bruke $\Theta(m)$ bits på kapasiteten, der $m = \lg \mathrm{W}$.

› **Polynomisk?**
T er polynomisk som funksjon av n og W, men er den «polynomisk»?

› **Nei!**
Vi må da skrive den som funksjon av n og m, og får $\mathrm{T}(n, m) = \Theta(n2^m)$

Egentlig som funksjon av $n + m$, men konklusjonen blir den samme

- **Beslutningsproblem:**
  Ja-/nei-spørsmål. Output er 0 eller 1; ikke tvetydig (altså, en funksjon, ikke generell relasjon)
- **Konkret beslutningsproblem:**
  En funksjon fra $\{0,1\}^*$ til $\{0,1\}$.
  Ugyldige strenger mappes til 0.
- **Optimeringsproblem:**
  Vil maksimere eller minimere en verdi
- **Terskling:**
  Terskelversjonen av et optimeringsproblem er et beslutningsproblem. Kan løses vha. opt., og kan altså ikke være vanskeligere.

Beslutning vanskelig $\implies$ optimering vanskelig

Reduksjoner

Vi vil redusere fra beslutningsproblem Q til beslutningsproblem Q′

Vi formaliserer det som en reduksjon fra språk L_1 til språk L_2

Input: En bitstreng x.

Input: En bitstreng x.

Output: En bitstreng $f(x)$, der

$$x \in \mathrm{L}_1 \iff f(x) \in \mathrm{L}_2\,.$$

Input: En bitstreng x.

Output: En bitstreng $f(x)$, der

$$x \in \mathrm{L}_1 \iff f(x) \in \mathrm{L}_2 \,.$$

Om vi kan avgjøre om $f(x) \in \mathrm{L}_2$ kan vi også avgjøre om $x \in \mathrm{L}_1$

Input: En bitstreng x.

Output: En bitstreng $f(x)$, der

$$x \in \mathrm{L}_1 \iff f(x) \in \mathrm{L}_2 \,.$$

Men ikke omvendt!

› **Redusibilitet:**
Hvis A kan reduseres til B i polynomisk tid, skriver vi $A \leq_{\mathbf{P}} B$

- **Redusibilitet:**
  Hvis A kan reduseres til B i polynomisk tid, skriver vi $A \leq_{\mathbf{P}} B$
- **Ordning:**
  Relasjonen $\leq_{\mathbf{P}}$ utgjør en *preordning*

- **Redusibilitet:**
  Hvis A kan reduseres til B i polynomisk tid, skriver vi $A \leq_{\mathbf{P}} B$
- **Ordning:**
  Relasjonen $\leq_{\mathbf{P}}$ utgjør en *preordning*

Delvis ordning uten antisymmetri, dvs., vi tillater sykler

- **Redusibilitet:**
  Hvis A kan reduseres til B i polynomisk tid, skriver vi $A \leq_{\mathbf{P}} B$
- **Ordning:**
  Relasjonen $\leq_{\mathbf{P}}$ utgjør en *preordning*
- **Hardhetsbevis:**
  For å vise at B er vanskelig, redusér fra et vanskelig problem A, dvs., etablér at $A \leq_{\mathbf{P}} B$

Delvis ordning uten antisymmetri, dvs., vi tillater sykler

Kompletthet

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.

› **Kompletthet:**
Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.

De komplette problemene er altså de vanskeligste i klassen

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.
- **NPC:**
  De komplette språkene i **NP**, under polynomiske reduksjoner.

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.
- **NPC:**
  De komplette språkene i **NP**, under polynomiske reduksjoner.

Altså de vanskeligste problemene i **NP**

› **NP-hardhet:**
Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.

› **NP-hardhet:**
Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.

Vi kaller gjerne klassen **NP-hard**

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det
  - er **NP**-hardt, og

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det
  - er **NP**-hardt, og
  - er i **NP**.

$$\mathrm{L} \in \mathbf{NPC}$$

Hvordan viser vi at L er **NP**-komplett?

- Vis at L $\in$ **NP**

At sertifikat for ja-svar kan verifiseres i pol. tid

- Vis at L $\in$ **NP**
- Velg et kjent **NP**-komplett språk L$'$

- Vis at $\mathrm{L} \in \mathbf{NP}$
- Velg et kjent **NP**-komplett språk L'
- Beskriv en algoritme som beregner en funksjon

$$f : \{0,1\}^* \to \{0,1\}^*$$

som mapper instanser av L' til instanser av L

Dette er altså reduksjonen fra L' til L, som viser $\mathrm{L}' \leq_{\mathbf{P}} \mathrm{L}$

- Vis at $\mathrm{L} \in \mathbf{NP}$
- Velg et kjent **NP**-komplett språk L'
- Beskriv en algoritme som beregner en funksjon

$$f : \{0,1\}^* \to \{0,1\}^*$$

som mapper instanser av L' til instanser av L

- Vis at

$$x \in \mathrm{L}' \iff f(x) \in \mathrm{L}\,,$$

for alle $x \in \{0,1\}^*$

Vi må sørge for at vi får samme svar for $f(x)$

- Vis at $\mathrm{L} \in \mathbf{NP}$
- Velg et kjent **NP**-komplett språk L'
- Beskriv en algoritme som beregner en funksjon

  $$f : \{0,1\}^* \to \{0,1\}^*$$

  som mapper instanser av L' til instanser av L
- Vis at

  $$x \in \mathrm{L}' \iff f(x) \in \mathrm{L}\,,$$

  for alle $x \in \{0,1\}^*$
- Vis at algoritimen som beregner f har polynomisk kjøretid

Illustrasjon av P, NP og NPC

NP

Nondeterministic Polynomial Time

Beslutningsproblemer* som kan verifiseres i polynomisk tid.†

* Uttrykt som formelle språk

† Gitt sertifikat av pol. størrelse

problemer som kan
løses i konstant tid

$O(1)$

$$\omega(n^c)$$

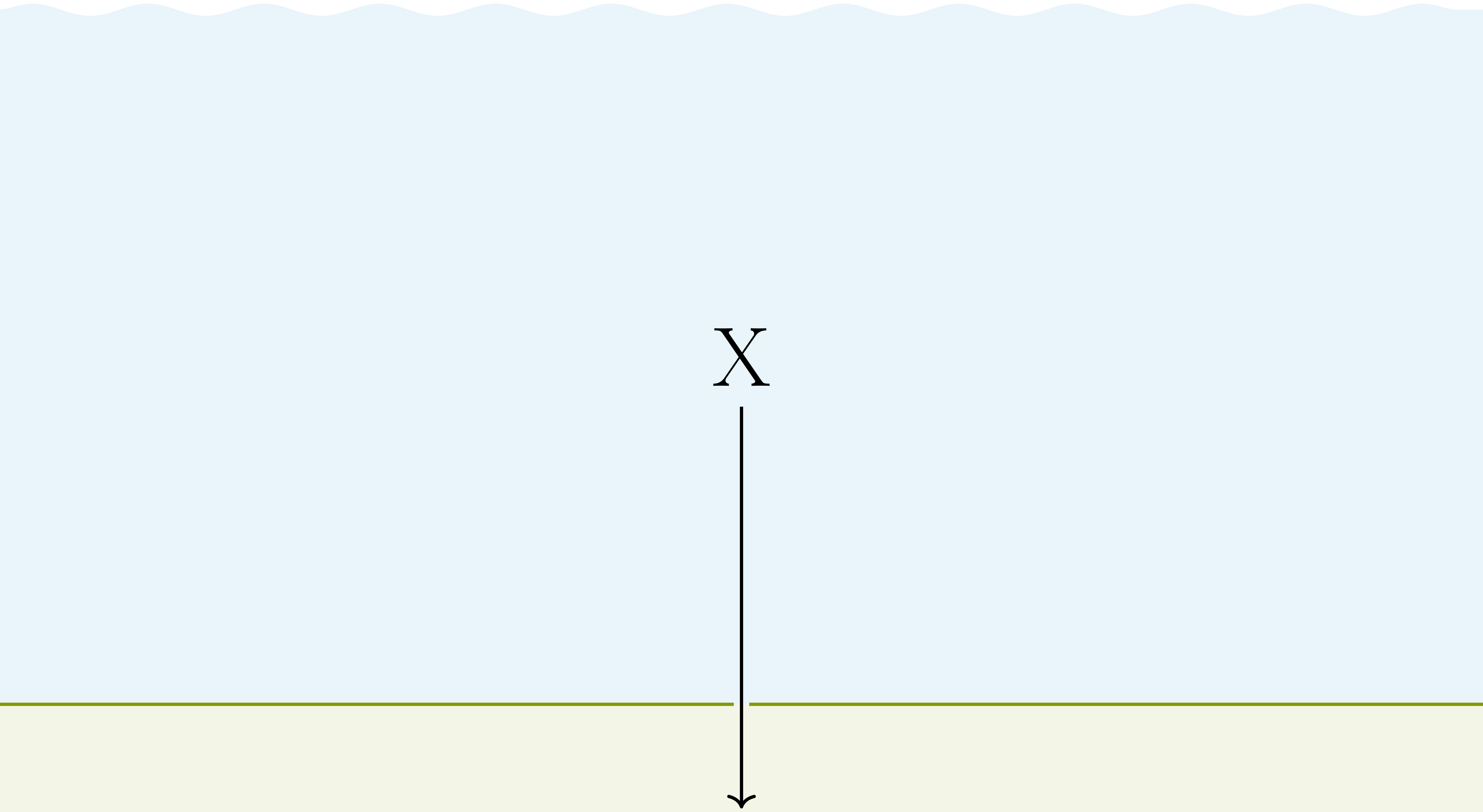
problemer som krever
superpolynomisk tid

$$O(1)$$

$\omega(n^c)$

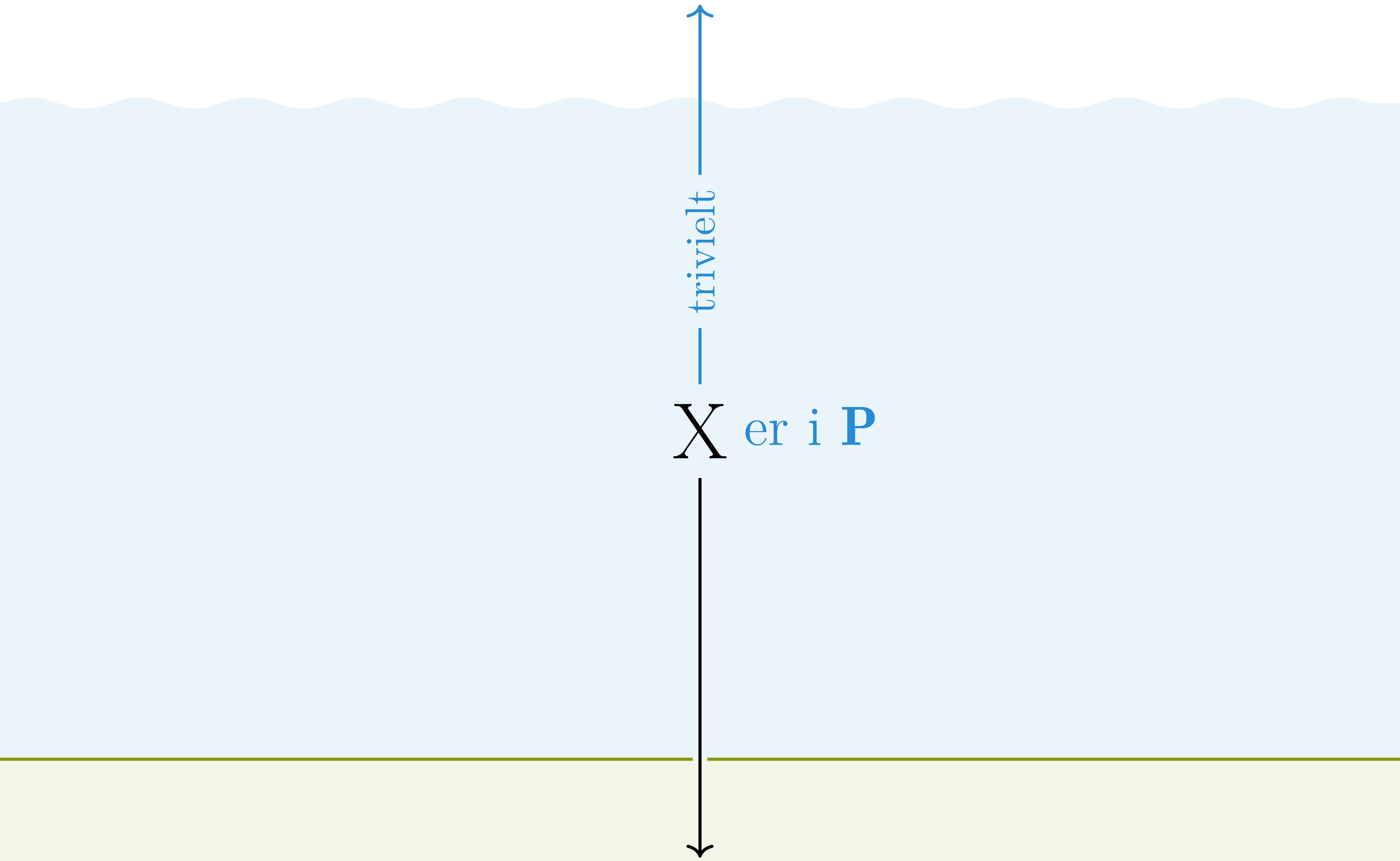
$O(1)$

$$\omega(n^c)$$



$\omega(n^c)$

X er i **P**

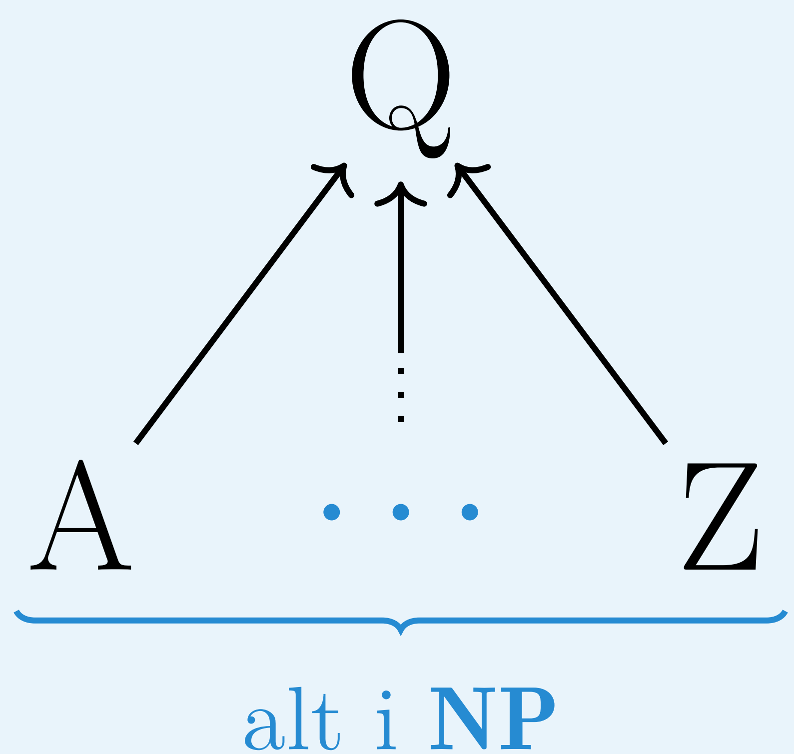

trivielt
X er i **P**

$$\omega(n^c)$$

$$\underbrace{\mathrm{A} \quad \cdots \quad \mathrm{Z}}_{\text{alt i } \mathbf{NP}}$$

$$\mathrm{O}(1)$$

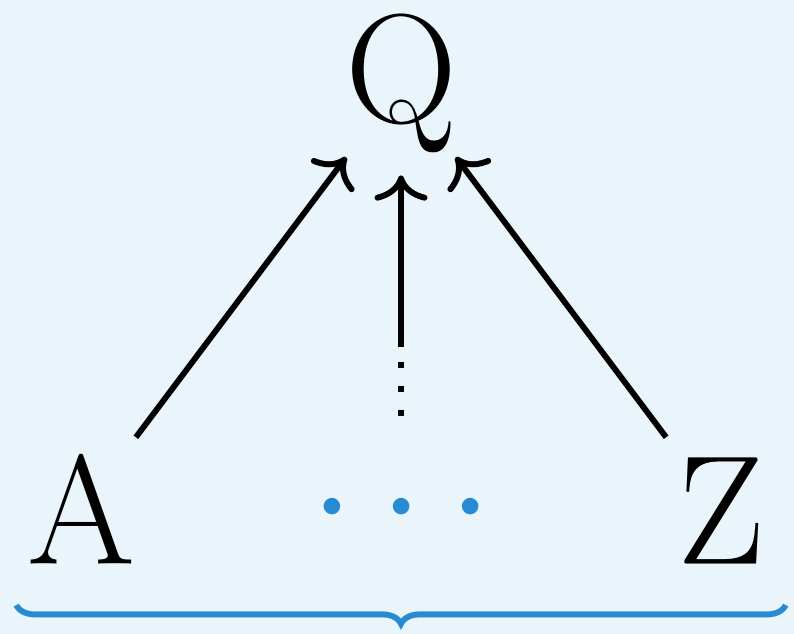
$\omega(n^c)$



$O(1)$

$\omega(n^c)$

i **NPC**, per def.

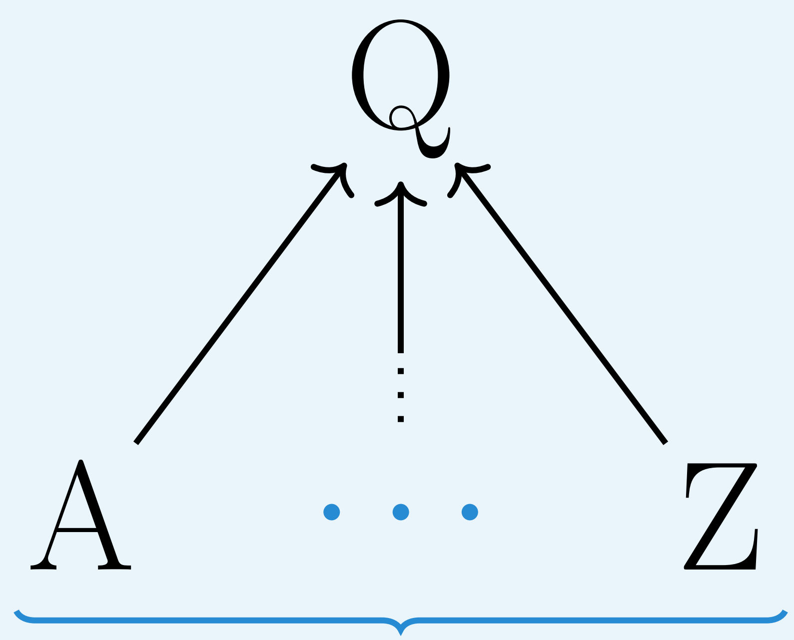


alt i **NP**

$O(1)$

$\omega(n^c)$

i **NPC**



alt i **NP**, inkl. **P** og **NPC**

$O(1)$

$\omega(n^c)$

i **NPC**

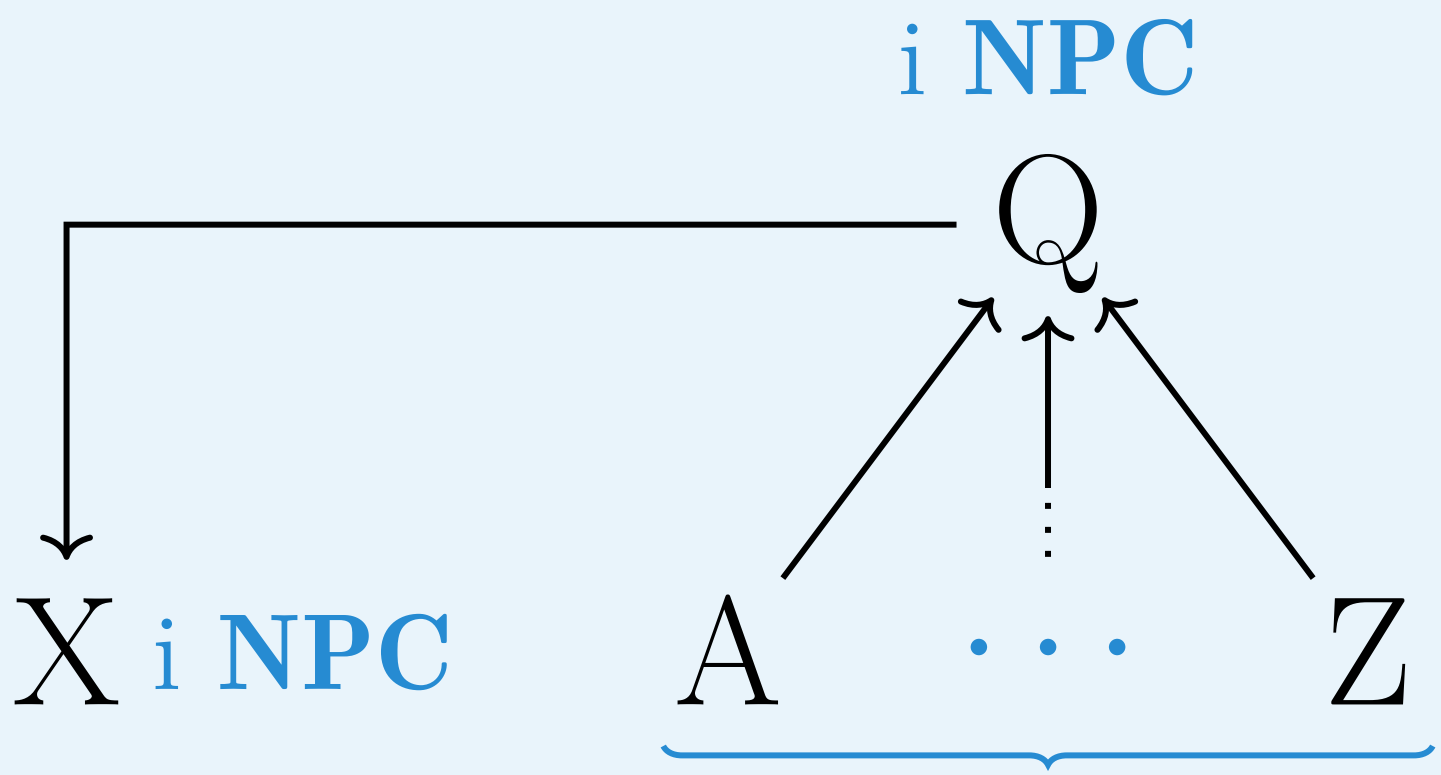
$$\begin{tikzcd} & Q & \\ A \arrow[ur] & \cdots \arrow[u, dotted] & Z \arrow[ul] \end{tikzcd}$$

alt i **NP**, inkl. **P** og **NPC**

(så probl. i **NPC** reduserer til hverandre)

$O(1)$

$\omega(n^c)$

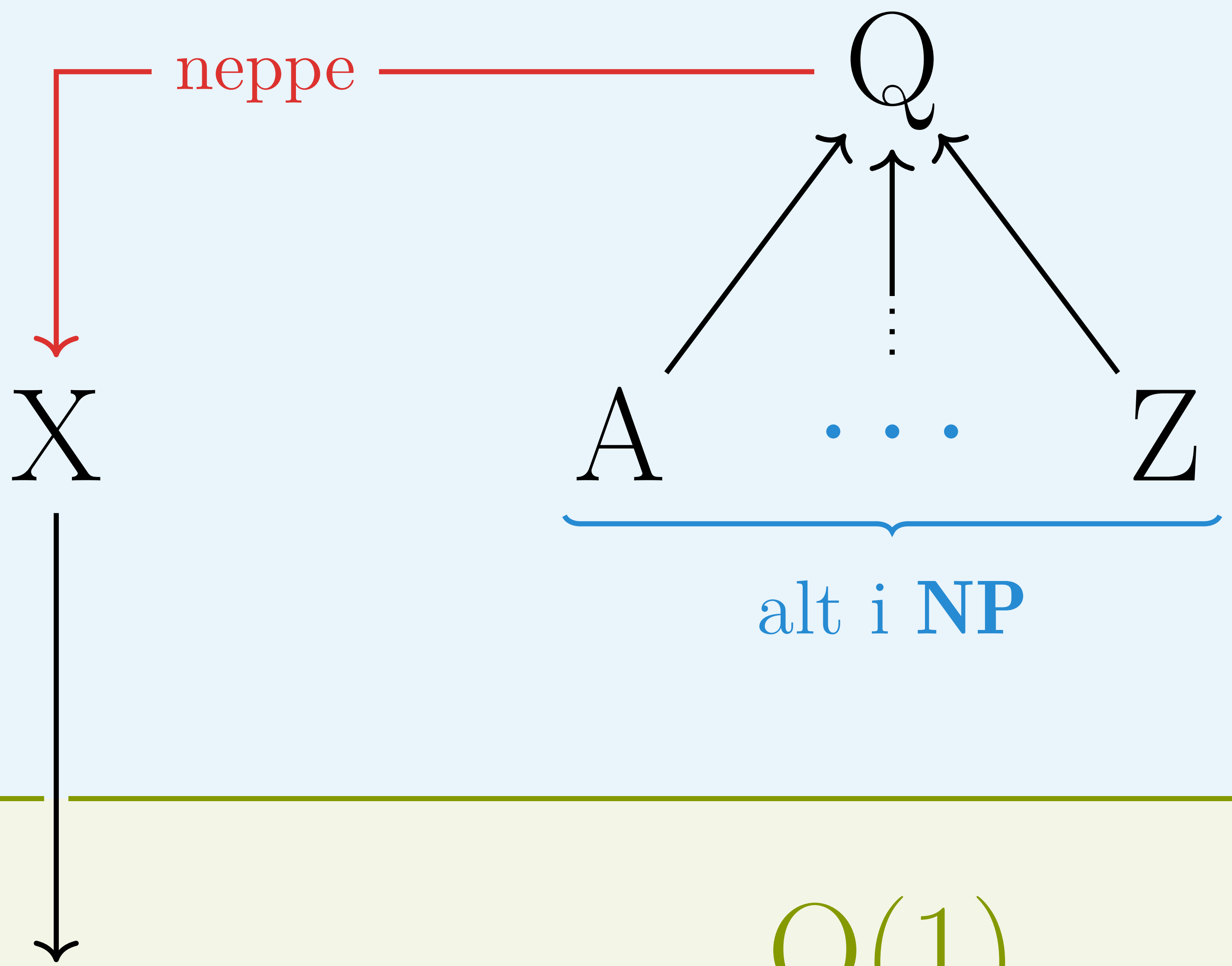


NPC-bevis:
Redusér *fra* et
problem i **NPC**

alt i **NP**, inkl. **P** og **NPC**
(så probl. i **NPC** reduserer til hverandre)

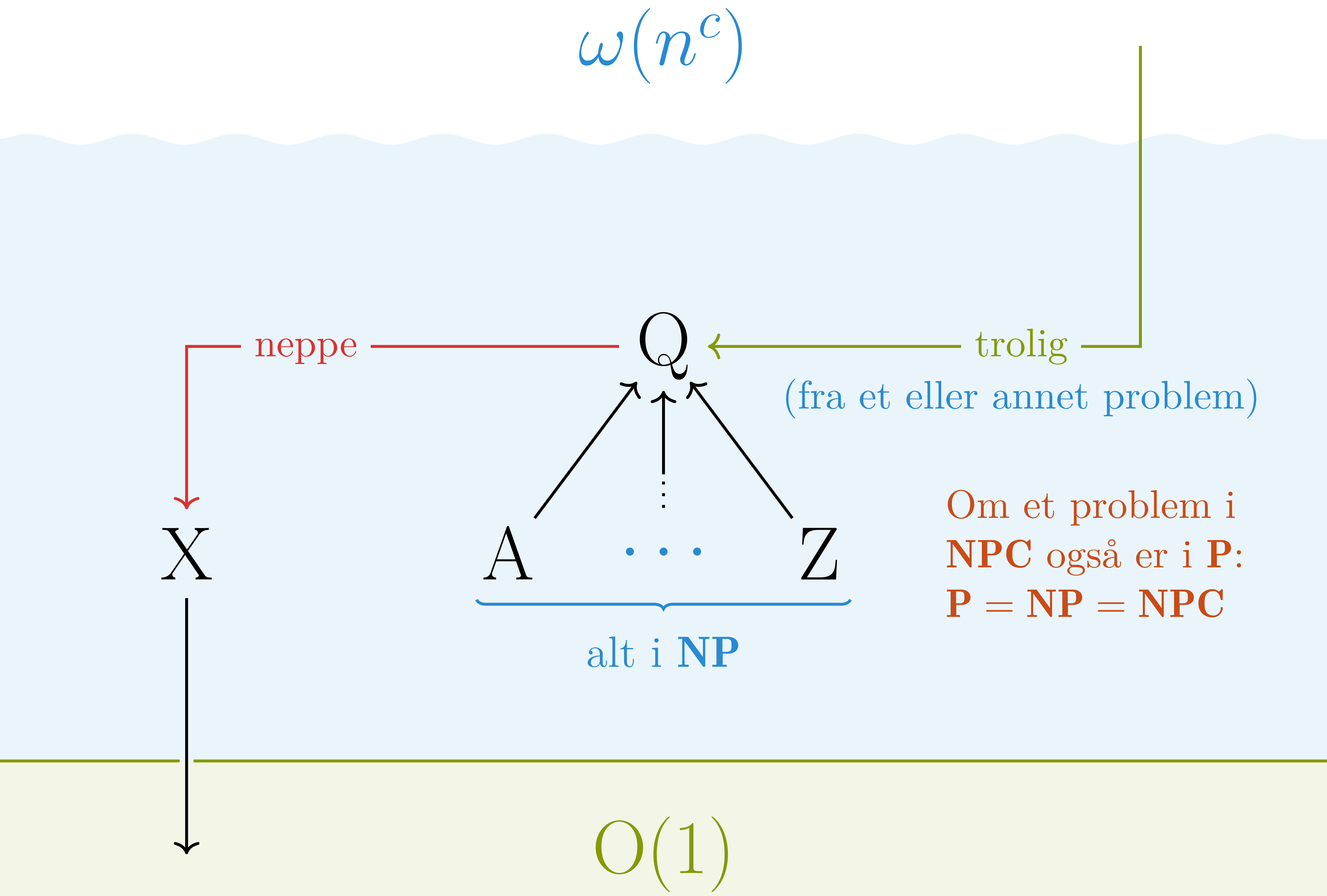
O(1)

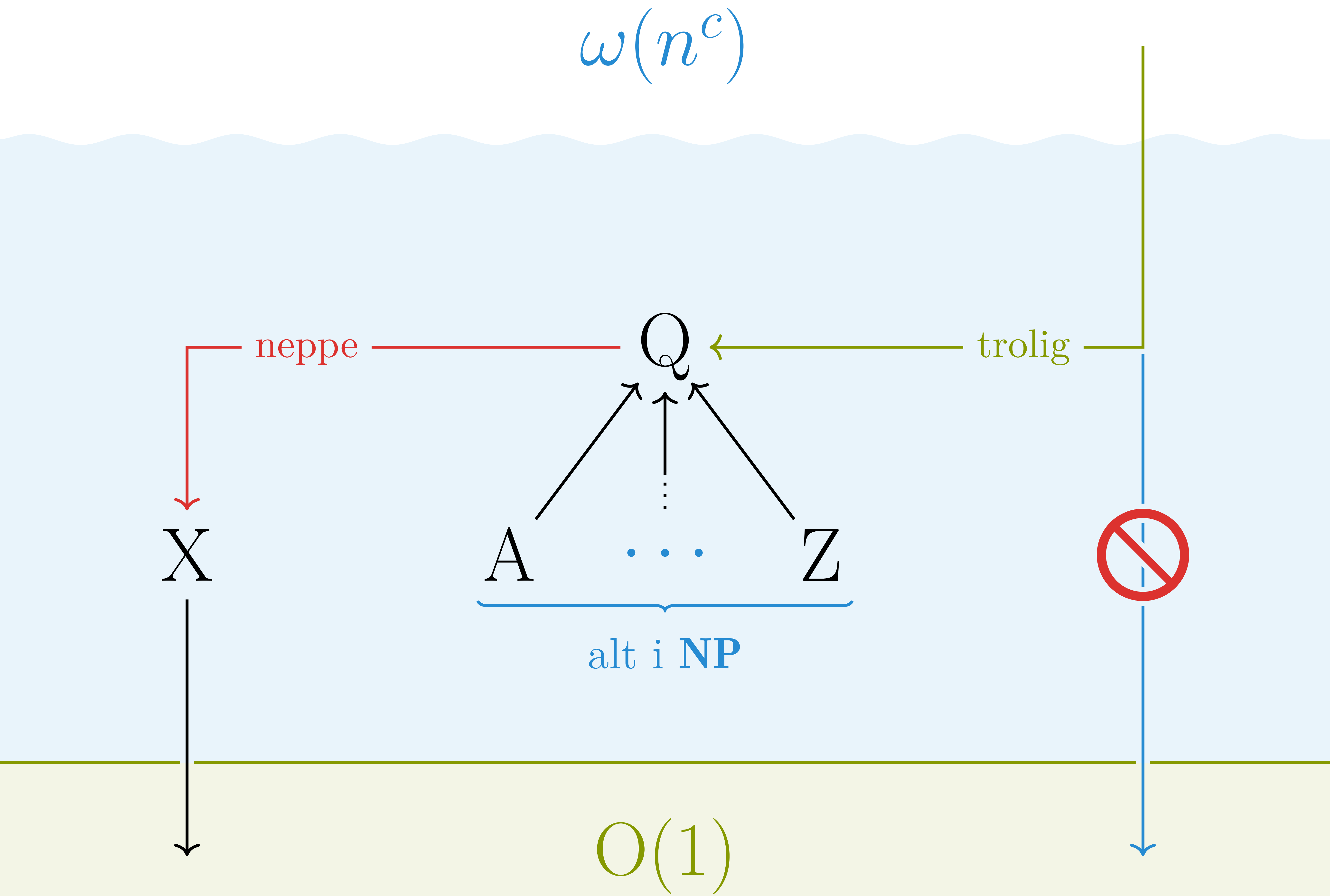
$\omega(n^c)$

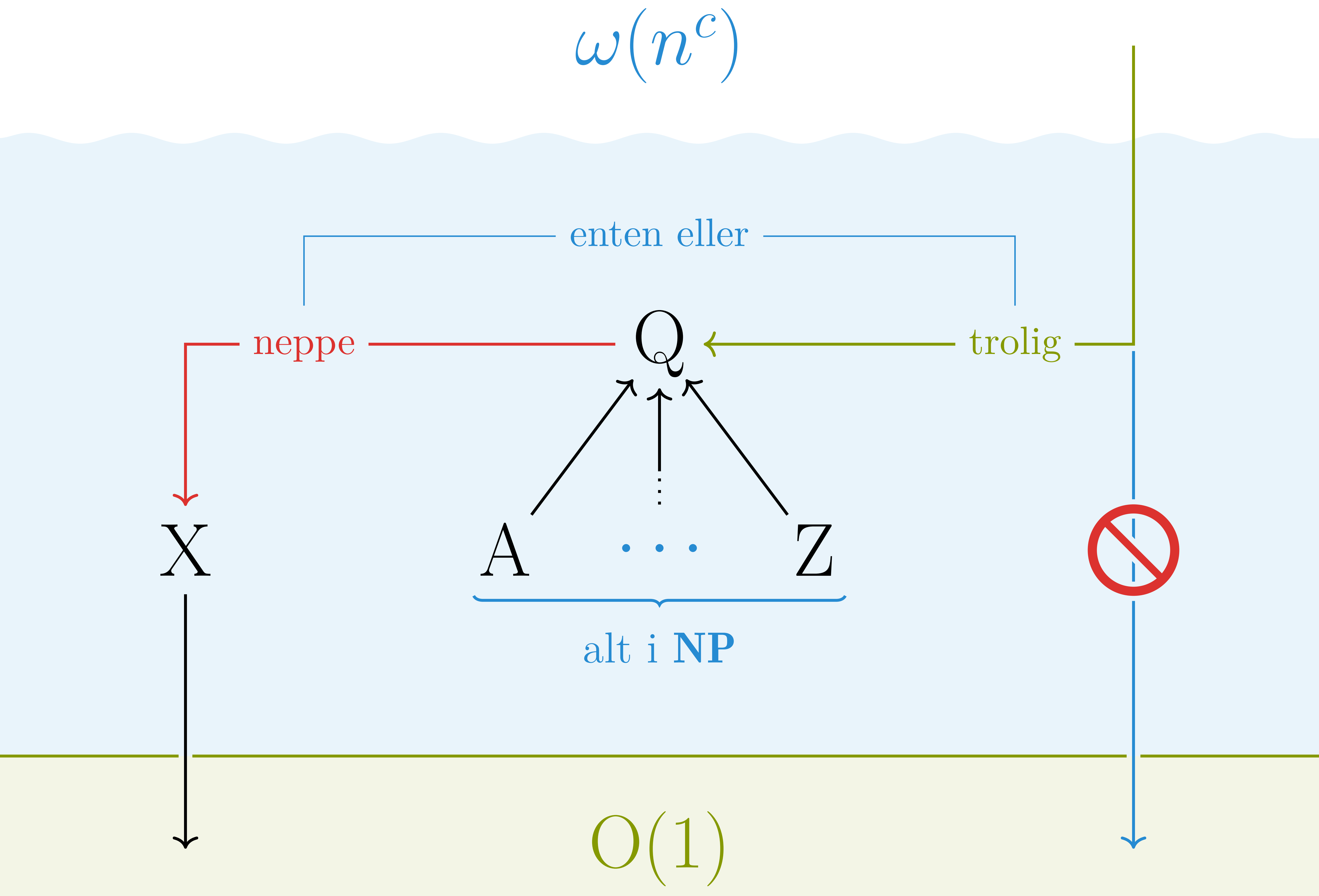


Om et problem i **NPC** også er i **P**:
P = **NP** = **NPC**

$O(1)$


$\omega(n^c)$
neppe
Q
trolig
(fra et eller annet problem)
X
A
$\cdots$
Z
alt i **NP**
Om et problem i **NPC** også er i **P**:
P = **NP** = **NPC**
O(1)


$\omega(n^c)$
neppe
Q
trolig
X
A
. . .
Z
alt i **NP**
O(1)



$\omega(n^c)$
enten eller
neppe
Q
trolig
X
A
Z
alt i **NP**
$O(1)$