

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 22 Komplexität

Michael Köhler-Bußmeier

Unentscheidbarkeit in der Praxis

Unentscheidbarkeit in der Praxis

In der Praxis sind die relevanten Fragen entscheidbar.
Woran liegt das?

Unentscheidbarkeit in der Praxis

In der Praxis sind die relevanten Fragen entscheidbar.
Woran liegt das?

Antwort: Alle physikalischen Realisierungen basieren auf
endlichen Elementen.

Unentscheidbarkeit in der Praxis

In der Praxis sind die relevanten Fragen entscheidbar.
Woran liegt das?

Antwort: Alle physikalischen Realisierungen basieren auf
endlichen Elementen.

Allerdings: Sie sind nur *im Prinzip* entscheidbar.

Unentscheidbarkeit in der Praxis

In der Praxis sind die relevanten Fragen entscheidbar.
Woran liegt das?

Antwort: Alle physikalischen Realisierungen basieren auf
endlichen Elementen.

Allerdings: Sie sind nur *im Prinzip* entscheidbar.

Wenn wir auf die gesuchte Antwort bspw. 10^{42} Tage warten
müssen, dann ist das Problem *praktisch* unentscheidbar.

Unentscheidbarkeit in der Praxis

In der Praxis sind die relevanten Fragen entscheidbar.
Woran liegt das?

Antwort: Alle physikalischen Realisierungen basieren auf
endlichen Elementen.

Allerdings: Sie sind nur *im Prinzip* entscheidbar.

Wenn wir auf die gesuchte Antwort bspw. 10^{42} Tage warten
müssen, dann ist das Problem *praktisch* unentscheidbar.

Entscheidend ist somit die **Komplexität**.

Unentscheidbarkeit in der Praxis

In der Praxis sind die relevanten Fragen entscheidbar.
Woran liegt das?

Antwort: Alle physikalischen Realisierungen basieren auf endlichen Elementen.

Allerdings: Sie sind nur *im Prinzip* entscheidbar.

Wenn wir auf die gesuchte Antwort bspw. 10^{42} Tage warten müssen, dann ist das Problem *praktisch* unentscheidbar.

Entscheidend ist somit die **Komplexität**.

“Komplexität = (Un-)entscheidbarkeit des Praktikers”

Definition von Komplexität

Welcher Aufwand an Rechenzeit und Speicherplatz ist erforderlich, um ein gegebenes Problem zu lösen?

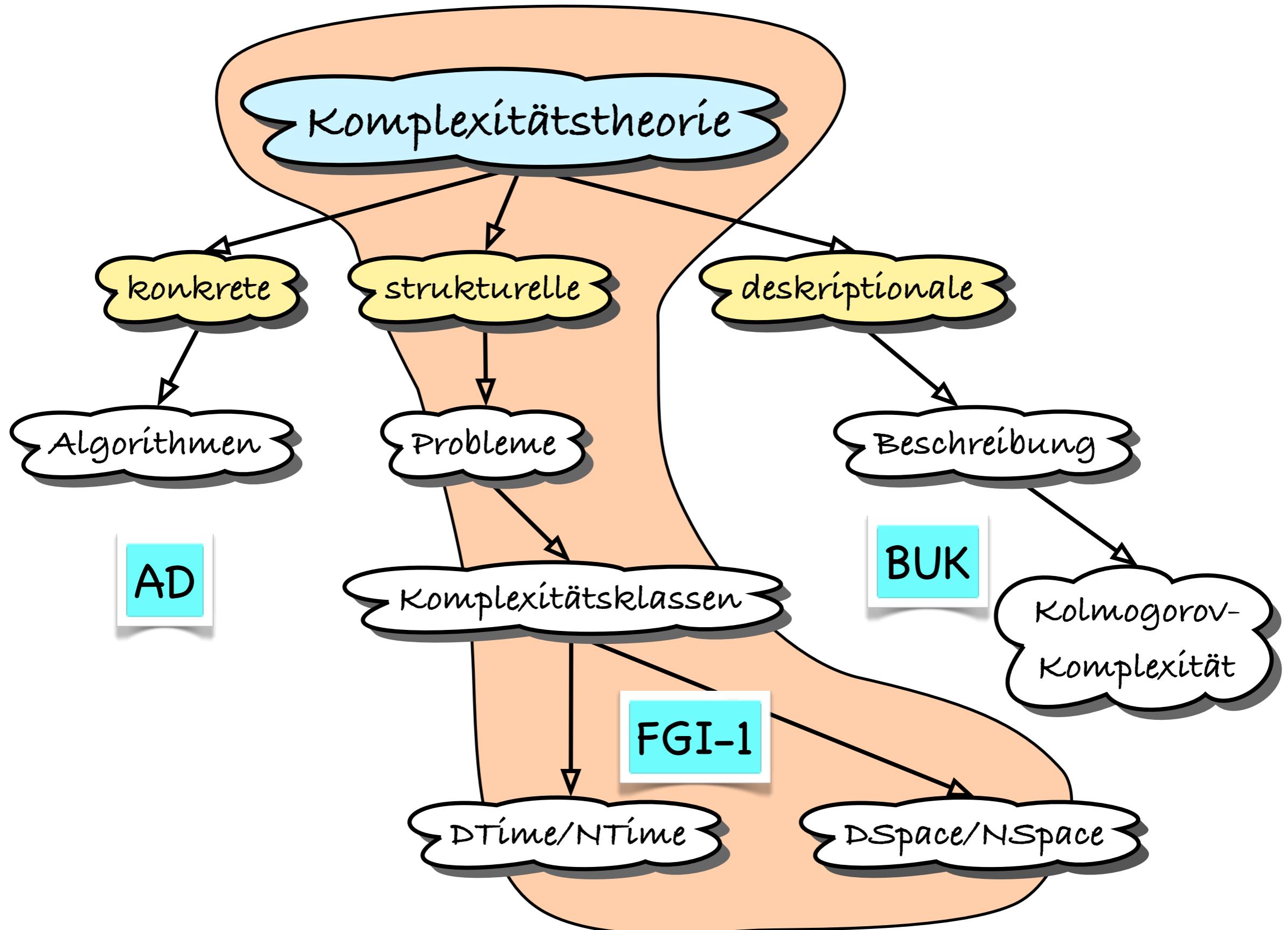
- Der für ein Problem minimal benötigter Rechenaufwand ist eine untere Schranken.

Der Nachweis unterer Schranken ist oft sehr schwer! Daher ging man in der Komplexitätstheorie dazu über, verschiedene Probleme miteinander zu vergleichen, um so wenigstens eine Aussage über deren relative Komplexitäten zu bekommen.
Komplexitätsklassen statt Chomsky-Hierarchie!

- Obere Schranke ist jeder bekannte Algorithmus.

- Strukturelle Komplexitätstheorie
- konkrete Komplexitätstheorie (Konstruktion konkreter Algorithmen)
- Algorithmik (Analyse und Konstruktion guter Algorithmen)

Komplexitätstheorie



Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Typ-0 Grammatik,
Turingmaschine
(NTM/DTM)

Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Typ 1 = kontextsensitive Sprachen

Typ-0 Grammatik,
Turingmaschine
(NTM/DTM)

Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Typ 1 = kontextsensitive Sprachen

Typ-0 Grammatik,
Turingmaschine
(NTM/DTM)

monotone Grammatik,
linear beschränkter
Automat (LBA)

Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Typ 1 = kontextsensitive Sprachen

Typ 2 = kontextfreie Sprachen

Typ-0 Grammatik,
Turingmaschine
(NTM/DTM)

monotone Grammatik,
linear beschränkter
Automat (LBA)

Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Typ 1 = kontextsensitive Sprachen

Typ 2 = kontextfreie Sprachen

Typ-0 Grammatik,
Turingmaschine
(NTM/DTM)

monotone Grammatik,
linear beschränkter
Automat (LBA)

kontextfreie Grammatik
Kellerautomat (PDA)

Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Typ 1 = kontextsensitive Sprachen

Typ 2 = kontextfreie Sprachen

Typ 3 = reguläre Mengen

Typ-0 Grammatik,
Turingmaschine
(NTM/DTM)

monotone Grammatik,
linear beschränkter
Automat (LBA)

kontextfreie Grammatik
Kellerautomat (PDA)

Chomsky Hierarchie

Typ 0 Sprachen = aufzählbare Mengen

Typ 1 = kontextsensitive Sprachen

Typ 2 = kontextfreie Sprachen

Typ 3 = reguläre Mengen

Typ-0 Grammatik,
Turingmaschine
(NTM/DTM)

monotone Grammatik,
linear beschränkter
Automat (LBA)

kontextfreie Grammatik
Kellerautomat (PDA)

reguläre Grammatik
endlicher Automat (NFA/
DFA)

Rechnungen (Wiederholung)

Eine endliche Rechnung $k_1 \vdash k_2 \vdash \dots \vdash k_t$ heißt
Erfolgsrechnung, wenn $k_1 \in \{q_0\} \cdot \Gamma^*$ und $k_t \in \Gamma^* \cdot F \cdot \Gamma^*$ gilt.

Es ist also erlaubt, dass auf k_t noch weitere Konfigurationen folgen. Die Rechnung könnte also noch weitergehen.

Für die DTM $A = (Z, \Sigma, \Gamma, \delta, q_0, \#, Z_{\text{end}})$ bezeichnet $L(A)$ die von A akzeptierte Sprache:

$$L(A) := \{w \in \Sigma^* \mid \exists u, v \in \Gamma^* \ \exists q \in Z_{\text{end}} : q_0 w \xrightarrow{*} uqv\}$$

Auch hier könnte die Rechnung also noch weitergehen!

Eine Turingmaschine (DTM oder NTM) **hält** nur, wenn es bei der vorliegenden Bandinschrift keine Folgekonfiguration gibt!

Randbedingungen für Zeit-Schranken

Zeitbedarf wird in Anzahl von Schritten gemessen.

- Es ist nützlich anzunehmen, dass eine TM ihre Eingabe stets vollständig liest und ein weiteres Feld vorrückt, um deren Ende festzustellen.

Definition 22.1:

Eine *NTM* arbeitet mit **Zeitbeschränkung** $t(n)$, wenn die *kleinste* Zahl der in einer w akzeptierenden Rechnung benutzten Konfigurationswechsel höchsten $t(|w|)$ ist, und das Wort vollständig gelesen wurde.
Die *NTM* ist dann $\max\{n + 1, t(n)\}$ -zeitbeschränkt.

Randbedingungen für Zeit-Schranken

Zeitbedarf wird in Anzahl von Schritten gemessen.

- Es ist nützlich anzunehmen, dass eine TM ihre Eingabe stets vollständig liest und ein weiteres Feld vorrückt, um deren Ende festzustellen.

Alternativ:**alle** Rechnungen...

Dies hat aber keine echte Einschränkung, da die NTM die Rechnungsschritte mitzählen kann.

Definition

Eine *NTM* arbeitet mit **Zeitbeschränkung** $t(n)$, wenn die kleinste Zahl der in einer w akzeptierenden Rechnung benutzten Konfigurationswechsel höchsten $t(|w|)$ ist, und das Wort vollständig gelesen wurde. Die *NTM* ist dann $\max\{n + 1, t(n)\}$ -zeitbeschränkt.

Randbedingungen für Platz-Schranken

Platzbedarf wird in benutzten Bandfeldern gemessen.

- Wenn eine NTM mit der Platzbeschränkung $s(n)$ arbeitet, dann sollte $s(n) \geq 1$ für jedes $n \in \mathbb{N}$ sein, denn die TM muss sich wenigstens ein Feld auf dem Arbeitsband ansehen!

Definition 22.2:

Eine NTM **arbeitet mit Platzbeschränkung** $s(n)$, wenn die kleinste Zahl der in einer w akzeptierenden Rechnung besuchten Bandzellen höchsten $s(|w|)$ ist. Die NTM ist dann $\max\{1, \lceil s(n) \rceil\}$ -platzbeschränkt.

Randbedingungen für Platz-Schranken

Platzbedarf wird in benutzten Bandfeldern gemessen.

- Wenn eine NTM mit der Platzbeschränkung $s(n)$ arbeitet, dann sollte $s(n) \geq 1$ für jedes $n \in \mathbb{N}$ sein, denn die TM muss sich wenigstens ein Feld auf dem Arbeitsband ansehen!

Alternativ:**alle** Rechnungen...

Dies hat aber keine echte Einschränkung, da die NTM die Rechnungsschritte mitzählen kann.

Definition 2

Eine NTM **arbeitet mit Platzbeschränkung** $s(n)$, wenn die kleinste Zahl der in einer w akzeptierenden Rechnung besuchten Bandzellen höchsten $s(|w|)$ ist. Die NTM ist dann $\max\{1, \lceil s(n) \rceil\}$ -platzbeschränkt.

Wie definiert man Komplexitätsklassen?

- Zusammenfassen ähnlicher Eingaben
 - über die Art des Problems (Semantik) ist in der Regel nur sehr schwer eine Aussage zu machen
 - Kategorisierung nur nach **Länge des Eingabewortes.**

Definition 22.3:

Seien $s : \mathbb{R} \rightarrow \mathbb{R}$, $t : \mathbb{R} \rightarrow \mathbb{R}$. Eine NTM T ist

- **t -zeitbeschränkt** gdw. sie für **jedes akzeptierte** Eingabewort der Länge n mit der Zeitbeschränkung $t(n)$ arbeitet.
- **s -platzbeschränkt** genau dann, wenn sie für **jedes akzeptierte** Eingabewort der Länge n mit der Platzbeschränkung $s(n)$ arbeitet.

Speed up und Kompression

Satz 22.1:

Speed up

Zu jeder $t(n)$ -zeitbeschränkten TM mit $\inf_{n \rightarrow \infty} \left(\frac{t(n)}{n} \right) = \infty$ und jedem $c \in \mathbb{R}$ mit $c > 0$ gibt es eine äquivalente $c \cdot t(n)$ -zeitbeschränkte TM.

Satz 22.2:

Kompression

Zu jeder $s(n)$ -platzbeschränkten TM und jeder reellen Zahl $c \in \mathbb{R}$ mit $c > 0$ gibt es eine äquivalente TM, die $c \cdot s(n)$ -platzbeschränkt ist.

Konstruktionen
ähnlich der
LBA-Konstruktion
zu einer
kontextsensitiven
Grammatik!
(Blockbildung zu
neuen Symbolen)

wichtige Komplexitätsklassen

Definition 22.4:

\mathcal{P} ist die Klasse der Sprachen (algorithmischen Probleme), die man deterministisch in Polynomialzeit erkennen (lösen) kann.

$\mathcal{P}Space$ ist die Klasse der Sprachen (algorithmischen Probleme), die man mit polynomiell viel Platz erkennen (lösen) kann.

- Obige Klassen sind unabhängig vom verwendeten Modell (*DTM*, Registermaschine, Programmiersprache, . . .).

grundlegende Komplexitätsklassen I

Definition 22.5

Für $t, s : \mathbb{N} \rightarrow \mathbb{N}$ mit $\forall n \in \mathbb{N} : t(n) \geq n$ und $s(n) \geq \log(n)$,

sowie $k \geq 1$ sei:

$\mathcal{DTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n nach höchstens

$\max(t(n), n + 1)$ Schritten hält.}

$\mathcal{NTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band NTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n und bei allen

Rechnungen nach höchstens $\max(t(n), n + 1)$

Schritten hält.}

$\mathcal{DSpace}_k(s) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n höchstens

$\lfloor s(n) \rfloor + 1$ Arbeitsbandfelder benötigt.}

grundlegende Komplexitätsklassen I

Definition 22.5

Für $t, s : \mathbb{N} \rightarrow \mathbb{N}$ mit $\forall n \in \mathbb{N} : t(n) \geq n$ und $s(n) \geq \log(n)$,

sowie $k \geq 1$ sei:

$\mathcal{DTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n nach höchstens
 $\max(t(n), n + 1)$ Schritten hält.}

$\mathcal{NTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band NTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n und bei allen
Rechnungen nach höchstens $\max(t(n), n + 1)$
Schritten hält.}

$\mathcal{DSpace}_k(s) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n höchstens
 $\lfloor s(n) \rfloor + 1$ Arbeitsbandfelder benötigt.}

grundlegende Komplexitätsklassen I

Definition 22.5

Für $t, s : \mathbb{N} \rightarrow \mathbb{N}$ mit $\forall n \in \mathbb{N} : t(n) \geq n$ und $s(n) \geq \log(n)$,

sowie $k \geq 1$ sei:

$\mathcal{DTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n nach höchstens
 $\max(t(n), n + 1)$ Schritten hält.}

$\mathcal{NTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band NTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n und bei allen
Rechnungen nach höchstens $\max(t(n), n + 1)$
Schritten hält.}

$\mathcal{DSpace}_k(s) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n höchstens
 $\lfloor s(n) \rfloor + 1$ Arbeitsbandfelder benötigt.}

grundlegende Komplexitätsklassen I

Definition 22.5

Für $t, s : \mathbb{N} \rightarrow \mathbb{N}$ mit $\forall n \in \mathbb{N} : t(n) \geq n$ und $s(n) \geq \log(n)$,

sowie $k \geq 1$ sei:

$\mathcal{DTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n nach höchstens
 $\max(t(n), n + 1)$ Schritten hält.}

$\mathcal{NTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band NTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n und bei allen
Rechnungen nach höchstens $\max(t(n), n + 1)$
Schritten hält.}

$\mathcal{DSpace}_k(s) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n höchstens
 $\lfloor s(n) \rfloor + 1$ Arbeitsbandfelder benötigt.}

grundlegende Komplexitätsklassen I

Definition 22.5

Für $t, s : \mathbb{N} \rightarrow \mathbb{N}$ mit $\forall n \in \mathbb{N} : t(n) \geq n$ und $s(n) \geq \log(n)$,

sowie $k \geq 1$ sei:

$\mathcal{DTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n nach höchstens
 $\max(t(n), n + 1)$ Schritten hält.}

$\mathcal{NTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band NTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n und bei allen
Rechnungen nach höchstens $\max(t(n), n + 1)$
Schritten hält.}

$\mathcal{DSpace}_k(s) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n höchstens
 $\lfloor s(n) \rfloor + 1$ Arbeitsbandfelder benötigt.}

grundlegende Komplexitätsklassen I

Definition 22.5

Für $t, s : \mathbb{N} \rightarrow \mathbb{N}$ mit $\forall n \in \mathbb{N} : t(n) \geq n$ und $s(n) \geq \log(n)$,

sowie $k \geq 1$ sei:

$\mathcal{DTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n nach höchstens
 $\max(t(n), n + 1)$ Schritten hält.}

$\mathcal{NTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band NTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n und bei allen
Rechnungen nach höchstens $\max(t(n), n + 1)$
Schritten hält.}

$\mathcal{DSpace}_k(s) := \{L \subseteq \Sigma^* \mid \text{es existiert eine } k\text{-Band DTM } M \text{ mit } L = L(M),$

die auf allen Eingaben der Länge n höchstens
 $\lfloor s(n) \rfloor + 1$ Arbeitsbandfelder benötigt.}

grundlegende Komplexitätsklassen II

Definition 22.6:

$$\mathcal{D}Time(t) := \bigcup_{k \geq 1} \mathcal{D}Time_k(t)$$

$$\mathcal{N}Time(t) := \bigcup_{k \geq 1} \mathcal{N}Time_k(t)$$

$$\mathcal{D}Space(s) := \bigcup_{k \geq 1} \mathcal{D}Space_k(s)$$

$$\mathcal{N}Space(s) := \bigcup_{k \geq 1} \mathcal{N}Space_k(s)$$

grundlegende Komplexitätsklassen III

Wegen der **Kompressions-** und **Speed Up-Sätze** gilt:

$$\mathcal{P} = \bigcup_{i \geq 1} \mathcal{DTime}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{DTime}(p(n))$$

$$\mathcal{NP} = \bigcup_{i \geq 1} \mathcal{NTime}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{NTime}(p(n))$$

$$\mathcal{P}\text{Space} = \bigcup_{i \geq 1} \mathcal{DSpace}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{DSpace}(p(n))$$

$$\mathcal{NP}\text{Space} = \bigcup_{i \geq 1} \mathcal{NSpace}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{NSpace}(p(n))$$

Hierbei bezeichnet POLY die Klasse aller Polynome aus $\mathbb{R}_{[x]}$!

Beispiele

$$\{a^k b^k c^k \mid k \in \mathbb{N}\} \quad \in \quad \mathcal{D}Space(\log n)$$

$$\{w\$w \mid w \in \Sigma^*\} \quad \in \quad \mathcal{D}Space(\log n)$$

$$\{p \in \{1\}\{0,1\}^* \mid [p]_2 \text{ ist Primzahl}\} \quad \in \quad \mathcal{D}Space(n)$$

$$\{p \in \{1\}\{0,1\}^* \mid [p]_2 \text{ ist Primzahl}\} \quad \in \quad \mathcal{D}Time(n^{12})$$

Beispiele

$$\{a^k b^k c^k \mid k \in \mathbb{N}\} \quad \in \quad \mathcal{D}Space(\log n)$$

$$\{w\$w \mid w \in \Sigma^*\} \quad \in \quad \mathcal{D}Space(\log n)$$

$$\{p \in \{1\}\{0,1\}^* \mid [p]_2 \text{ ist Primzahl}\} \quad \in \quad \mathcal{D}Space(n)$$

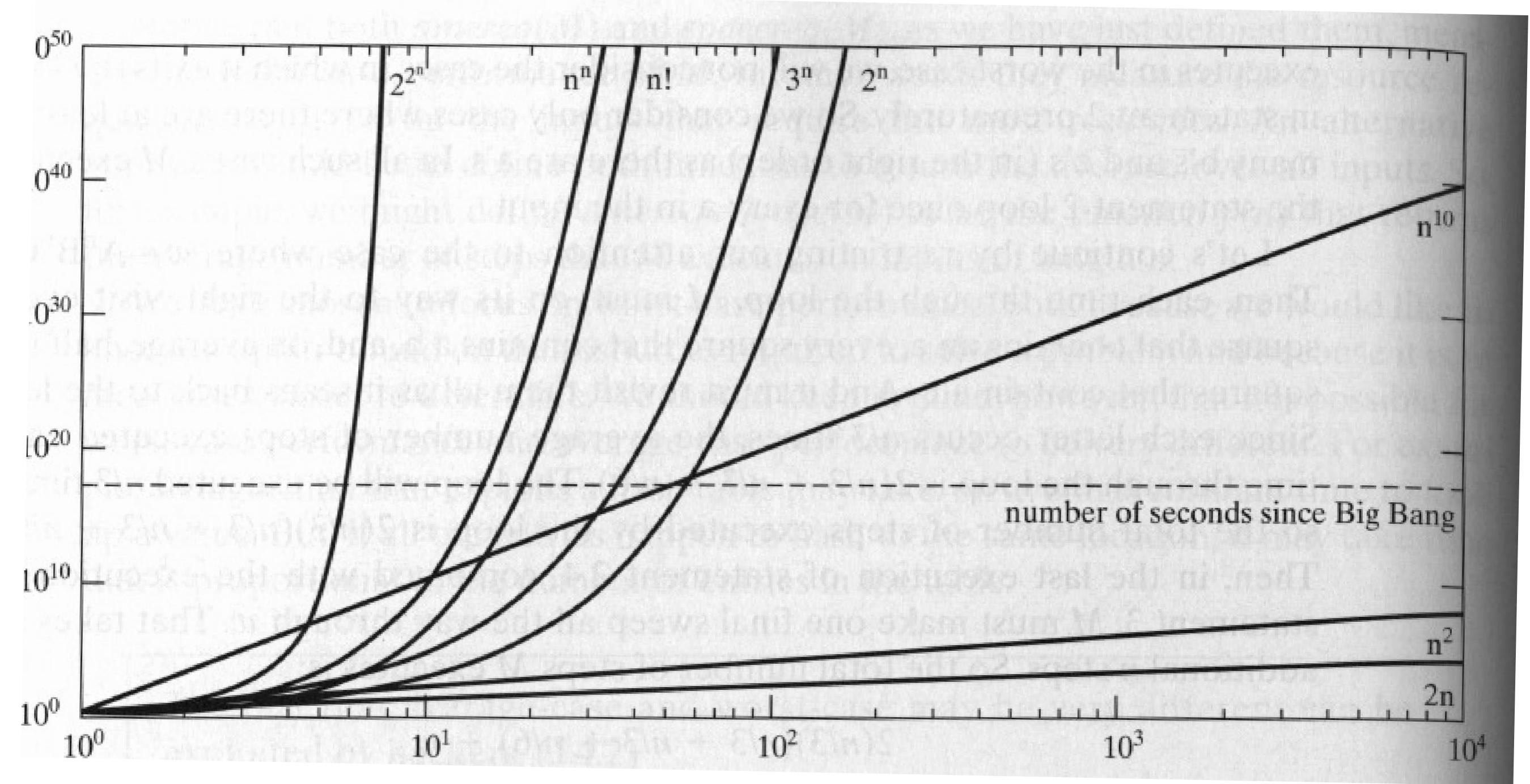
$$\{p \in \{1\}\{0,1\}^* \mid [p]_2 \text{ ist Primzahl}\} \quad \in \quad \mathcal{D}Time(n^{12})$$

Letzteres Ergebnis vom 10. Juli 2002 stammt von Manindra Agrawal und den Bachelor-Studenten Neeraj Kayal und Nitin Saxena.

Wachstumsvergleich

Wegen der Kompressions- und Speed-up Theoreme,
sind konstante Faktoren keine wirkliche
Unterscheidung zwischen den Funktionen.
Daher werden wir Funktionen nach der
„Größenordnung“ klassifizieren!

Wachstumsvergleich



Landau Notation

Definition 22.7:

Eine Funktion $f : \mathbb{N} \rightarrow \mathbb{R}$ wächst mit der Ordnung $g(n)$ bzw. g , notiert durch $f(n) \in O(g(n))$, falls

$g : \mathbb{N} \rightarrow \mathbb{R}$ eine Funktion ist und eine Konstante $c \in \mathbb{R}^+ := \{x \in \mathbb{R} \mid x > 0\}$ existiert, so dass $|f(n)| \leq c \cdot |g(n)|$ für alle, bis auf endlich viele $n \in \mathbb{N}$ gilt.

Etwas knapper notiert liest sich das so:

$$O(g(n)) = \left\{ f : \mathbb{N} \rightarrow \mathbb{R} \mid (\exists c \in \mathbb{R}^+) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [|f(n)| \leq c |g(n)|] \right\}.$$

Kurz: f wächst nicht schneller als g !

Beispiel

Als weitere Beispiele seien f und g definiert durch:

$$f(n) := 12n^4 - 11n^3 + 1993 \text{ und } g(n) := 7n^3 - n.$$

Dann ist $g \in O(f)$ aber nicht $f \in O(g)$.

Auch gilt $f \in O(n^4)$ oder $f \in O(12000n^4 - 30000)$.

Andere Schreibweisen für $f \in O(g)$ sind in der Literatur auch $f = O(g)$ oder sehr selten $f \simeq O(g)$.

Rechenregeln

$$n^m \in O(n^{m'}) \text{ falls } m \leq m' \quad (1)$$

$$f(n) \in O(f(n)) \quad (2)$$

$$cO(f(n)) = O(f(n)) \quad (3)$$

$$O(O(f(n))) = O(f(n)) \quad (4)$$

$$O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|) \quad (5)$$

$$O(f(n))O(g(n)) = O(f(n)g(n)) \quad (6)$$

$$O(f(n)g(n)) = f(n)O(g(n)) \quad (7)$$

Weiteres aus der O -Notation

Definition 22.8:

Wir definieren neben $O(f(n))$ noch weitere asymptotische Funktionenklassen:

$o(g(n))$ ("klein *oh*"),

$\Omega(g(n))$ ("groß Omega"),

$\omega(g(n))$ ("klein Omega") und

$\Theta(g(n))$ ("Theta",

was in diesem Zusammenhang nur als Großbuchstabe auftritt),

und sagen dann:

eine Funktion $f(n)$ ist **von der Ordnung** $\mathcal{O}(g(n))$, wenn sie für $\mathcal{O} \in \{O, o, \Omega, \omega, \Theta\}$ zur Menge $\mathcal{O}(g(n))$ gehört.

noch zu **Def. 22.8**

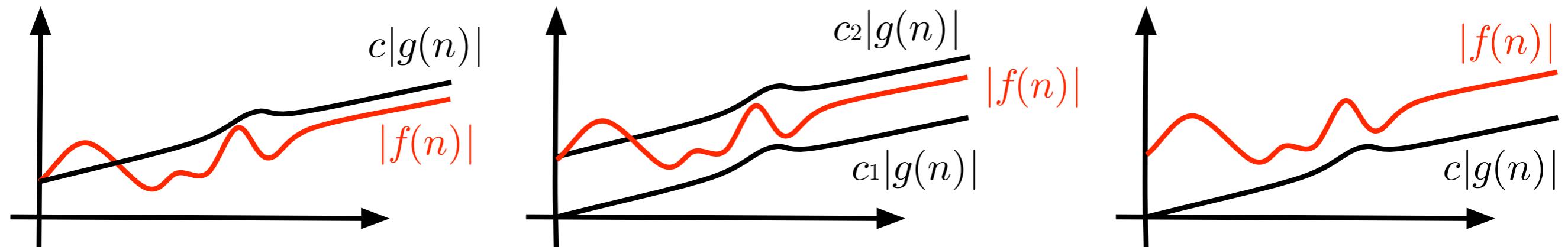
$f(n) \in o(g(n))$ ist äquivalent zu $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

$f(n) \in \omega(g(n)) \iff g(n) \in o(f(n))$

$f(n) \in \Omega(g(n)) \iff g(n) \in O(f(n))$

$f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \text{ und } f(n) \in \Omega(g(n)).$

Θ einmal bildlich



$f \in \Theta(g)$ statt umständlicher $f(n) \in \Theta(g(n))$

Beispiel

Wegen der speed-up Sätze charakterisieren alle Funktionen $g_1, g_2 \in \Theta(f)$ die gleichen Klassen:

$$\text{DTIME}(g_1) = \text{DTIME}(g_2)$$

$$\text{DSPACE}(g_1) = \text{DSPACE}(g_2)$$

$$\text{NTIME}(g_1) = \text{NTIME}(g_2)$$

$$\text{NSPACE}(g_1) = \text{NSPACE}(g_2)$$

eine Vergleichstabelle

A	B	O	o	Ω	ω	Θ
$\lg^k n$	n^ϵ	×	×			
n^k	c^n	×	×			
\sqrt{n}	$n^{\sin n}$					
2^n	$2^{n/2}$			×	×	
$n^{\lg \epsilon}$	$\epsilon^{\lg n}$	×		×		×
$\lg(n!)$	$\lg(n^n)$	×		×		×

Arten von Problemen

Je nach Aufgabenstellung lassen sich verschiedene Grundtypen von Problemen unterscheiden:

1. Entscheidungsprobleme
2. Suchprobleme
3. Optimierungsprobleme
4. Abzählungsprobleme
5. Anzahlprobleme

Notation von Problemen

Darstellung von Problemen grundsätzlich so:

Gegeben: ⟨ Angabe aller verwendbaren Eingabedaten. ⟩

Frage: ⟨ Problemstellung mit geforderter Ausgabe. ⟩

Antwort: ⟨ Gewünschte, erwartete Antwort(Möglichkeit)en. ⟩

Ein Problem beschreibt eine Klasse von gleichartigen Fragestellungen, nicht nur eine einzelne Frage:

NICHT:

„Ist die Formel $(x \vee (y \wedge \neg x))$ erfüllbar?“

SONDERN:

„Hat eine beliebig vorgegebene aussagenlogische Formel eine erfüllende Belegung?“

Probleminstanzen

In der Formulierung eines Problems kommen i.A. gewisse Parameter vor. Für die Bestimmung von

$$\text{ggT}(m, n)$$

sind dies $m, n \in \mathbb{N}$.

Werden alle vorkommenden Parameter durch konkrete Werte ersetzt, so erhält man eine konkrete Aufgabenstellung, eine **Instanz** des Problems.

Setzen wir in unserem Beispiel $m = 125$ und $n = 35$, so bedeutet

$$\text{ggT}(125, 35)$$

die konkrete Aufgabe, den größten gemeinsamen Teiler der beiden Zahlen 125 und 35, also 5, zu bestimmen

Entscheidungsproblem vs. Sprache akzeptieren

Problem **Prim?**

Gegeben: Die Binärdarstellung einer natürliche Zahl $n \in \mathbb{N}$.

Gesucht: Ist n eine Primzahl?

Antwort: JA oder NEIN

Problem **Prim?** ist *Entscheidungsproblem*, d.h., Die Lösungen können nur aus der Menge { JA , NEIN } stammen.

Die zum Problem **Prim?** gehörende Sprache wäre:

$$L_{\text{prim}} := \left\{ w \in \{0, 1\}^* \mid [w]_2 \text{ ist Primzahl} \right\}$$

Suchprobleme

... zum Beispiel:

Gegeben: ungerichteter Graph
 $G := (V, E)$ mit $E \subseteq V \times V$ und Knoten
 $v_1, v_2 \in V$.

Gesucht: ein Weg von v_1 nach v_2

Antwort: Kantenfolge eines Pfades **oder**
„Es gibt keinen!“

Optimierungsprobleme

... zum Beispiel:

Gegeben: gerichteter, bewerteter Graph
 $G := (V, E)$ mit $E \subseteq V \times \mathbb{N} \times V$ und Knoten
 $v_1, v_2 \in V$.

Gesucht: ein günstigster Weg von v_1 nach v_2

Antwort: Kantenfolge eines Pfades (evtl. mit Kosten)
oder „Es gibt keinen!“

Abzählprobleme

... zum Beispiel :

Gegeben: endliche Menge von Objekten

Gesucht: alle binären Suchbäume für diese Objekte

Antwort: Aufzählung der binären Suchbäume

Anzahlprobleme

Variation des Suchproblems:

Frage lautet nicht: "Gibt es eine Lösung?",
sondern: "**Wieviele** Lösungen gibt es?".

Das Entscheidungsproblem ist also einfacher
als das Anzahlproblem:

Anzahlprobleme

Variation des Suchproblems:

Frage lautet nicht: "Gibt es eine Lösung?",
sondern: "**Wieviele** Lösungen gibt es?".

Das Entscheidungsproblem ist also einfacher
als das Anzahlproblem:

"Ja, es gibt eine Lösung." gdw. "Es gibt $n > 0$ Lösungen."

Anzahlprobleme

Variation des Suchproblems:

Frage lautet nicht: "Gibt es eine Lösung?",
sondern: "**Wieviele** Lösungen gibt es?".

Das Entscheidungsproblem ist also einfacher
als das Anzahlproblem:

"Ja, es gibt eine Lösung." gdw. "Es gibt $n > 0$ Lösungen."

Gebiete konkreter Algorithmen und deren
Programmierung wird das Modul
„Algorithmen und Datenstrukturen (AD)“
behandeln.

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 23

Komplexitätsklassen: P, NP und PSpace

Michael Köhler-Bußmeier

grundlegende Komplexitätsklassen

Zur Erinnerung:

Definition 22.6:

$$\mathcal{D}Time(t) := \bigcup_{k \geq 1} \mathcal{D}Time_k(t)$$

$$\mathcal{N}Time(t) := \bigcup_{k \geq 1} \mathcal{N}Time_k(t)$$

$$\mathcal{D}Space(s) := \bigcup_{k \geq 1} \mathcal{D}Space_k(s)$$

$$\mathcal{N}Space(s) := \bigcup_{k \geq 1} \mathcal{N}Space_k(s)$$

Komplexitätsklassen

Wegen der **Kompressions-** und **Speed Up-Sätze** gilt:

$$\mathcal{P} = \bigcup_{i \geq 1} \mathcal{DTime}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{DTime}(p(n))$$

$$\mathcal{NP} = \bigcup_{i \geq 1} \mathcal{NTime}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{NTime}(p(n))$$

$$\mathcal{P}\text{Space} = \bigcup_{i \geq 1} \mathcal{DSpace}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{DSpace}(p(n))$$

$$\mathcal{NP}\text{Space} = \bigcup_{i \geq 1} \mathcal{NSpace}(n^i) = \bigcup_{p \in \text{POLY}} \mathcal{NSpace}(p(n))$$

Hierbei bezeichnet POLY die Klasse aller Polynome aus $\mathbb{R}_{[x]}$!

Aus den Definitionen ergibt sich:

Korollar:

Die sich direkt aus den Definitionen ergebenden trivialen Beziehungen zwischen diesen und den zuvor definierten Komplexitätsklassen sind offensichtlich die folgenden:

$$\mathcal{D}Space(f) \subseteq \mathcal{N}Space(f)$$

$$\mathcal{D}Time(f) \subseteq \mathcal{N}Time(f)$$

$$\mathcal{D}Time(f) \subseteq \mathcal{D}Space(f)$$

$$\mathcal{N}Time(f) \subseteq \mathcal{N}Space(f)$$

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

$$\mathcal{P}Space \subseteq \mathcal{NP}Space$$

Zusammenhang von Zeit- und Platzkomplexität

- Eine TM, die $t(n)$ Zeit (d.h. Schritte) zur Verfügung hat, kann nicht mehr als $t(n)$ Bandzellen besuchen.
 - Umgekehrt gilt dies nicht!
- Platz kann **wiederverwendet** werden, Zeit nicht!
- *Kann man trotzdem eine maximale Rechenzeit in Abhängigkeit vom verbrauchten Platz angeben?*
 - Ja! **Idee:** Sobald eine *DTM* eine Konfiguration ein zweites Mal besucht, befinden wir uns in einer Endlosschleife!
 - Bleibt zu berechnen, **wieviele verschiedene Konfigurationen** bei einer gegebenen Platzbeschränkung möglich sind.

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

- Entweder akzeptiert die $s(n)$ -beschränkte NTM nach spätestens nach m Schritten
- oder die NTM akzeptiert nie, weil sie danach keine neuen Konfigurationen mehr erreicht.

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

- Entweder akzeptiert die $s(n)$ -beschränkte NTM nach spätestens nach m Schritten
- oder die NTM akzeptiert nie, weil sie danach keine neuen Konfigurationen mehr erreicht.

$$\begin{aligned} m &= |Q| \cdot s(n) \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|)} \cdot |\Gamma|^{\log_{|\Gamma|}(s(n))} \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|) + \log_{|\Gamma|}(s(n)) + s(n)} \\ &\leq c^{s(n)} \quad \text{für ein } c \end{aligned}$$

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

- Entweder akzeptiert die $s(n)$ -beschränkte NTM nach spätestens nach m Schritten
- oder die NTM akzeptiert nie, weil sie danach keine neuen Konfigurationen mehr erreicht.

$$\begin{aligned} m &= |Q| \cdot s(n) \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|)} \cdot |\Gamma|^{\log_{|\Gamma|}(s(n))} \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|) + \log_{|\Gamma|}(s(n)) + s(n)} \\ &\leq c^{s(n)} \quad \text{für ein } c \end{aligned}$$

“Platzschranke
impliziert
Zeitschranke”

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

- Entweder akzeptiert die $s(n)$ -beschränkte NTM nach spätestens nach m Schritten
- oder die NTM akzeptiert nie, weil sie danach keine neuen Konfigurationen mehr erreicht.

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

- Entweder akzeptiert die $s(n)$ -beschränkte NTM nach spätestens nach m Schritten
- oder die NTM akzeptiert nie, weil sie danach keine neuen Konfigurationen mehr erreicht.

$$\begin{aligned} m &= |Q| \cdot s(n) \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|)} \cdot |\Gamma|^{\log_{|\Gamma|}(s(n))} \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|) + \log_{|\Gamma|}(s(n)) + s(n)} \\ &\leq c^{s(n)} \quad \text{für ein } c \end{aligned}$$

Maximalzahl möglicher Konfigurationen

Wie viele verschiedenen Konfigurationen hat eine NTM, die höchstens $s(n)$ Platz benötigt?

- Es gibt $|Q|$ viele Zustände.
- Es existieren $s(n)$ mögliche Kopfpositionen.
- Auf jedem Bandfeld sind $|\Gamma|$ verschiedene Symbole möglich.

Also: maximal $m := |Q| \cdot s(n) \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

- Entweder akzeptiert die $s(n)$ -beschränkte NTM nach spätestens nach m Schritten
- oder die NTM akzeptiert nie, weil sie danach keine neuen Konfigurationen mehr erreicht.

$$\begin{aligned} m &= |Q| \cdot s(n) \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|)} \cdot |\Gamma|^{\log_{|\Gamma|}(s(n))} \cdot |\Gamma|^{s(n)} \\ &= |\Gamma|^{\log_{|\Gamma|}(|Q|) + \log_{|\Gamma|}(s(n)) + s(n)} \\ &\leq c^{s(n)} \quad \text{für ein } c \end{aligned}$$

“Platzschranke
impliziert
Zeitschranke”

Zeit versus Platz I

Zeit versus Platz I

NTM: Im gerichteten Graphen aller Konfigurationen durchlaufen wir bis zum Akzeptieren maximal $s(n)$ Konfiguration.

Zeit versus Platz I

NTM: Im gerichteten Graphen aller Konfigurationen durchlaufen wir bis zum Akzeptieren maximal $s(n)$ Konfiguration.

Per Tiefensuche überprüfen wir alle Knoten auf Akzeptanz.

Zeitbedarf bei Tiefensuche: Linear in der Knotenzahl.

Zeit versus Platz I

NTM: Im gerichteten Graphen aller Konfigurationen durchlaufen wir bis zum Akzeptieren maximal $s(n)$ Konfiguration.

Per Tiefensuche überprüfen wir alle Knoten auf Akzeptanz.

Zeitbedarf bei Tiefensuche: Linear in der Knotenzahl.

Für N_{Space} gab es $c^{s(n)}$ verschiedene Konfigurationen. Also:

Zeit versus Platz I

NTM: Im gerichteten Graphen aller Konfigurationen durchlaufen wir bis zum Akzeptieren maximal $s(n)$ Konfiguration.

Per Tiefensuche überprüfen wir alle Knoten auf Akzeptanz.

Zeitbedarf bei Tiefensuche: Linear in der Knotenzahl.

Für $N\text{Space}$ gab es $c^{s(n)}$ verschiedene Konfigurationen. Also:

$$\mathcal{N}\text{Space}(s(n)) \subseteq \mathcal{D}\text{Time}(c^{s(n)})$$

Zeit versus Platz I

NTM: Im gerichteten Graphen aller Konfigurationen durchlaufen wir bis zum Akzeptieren maximal $s(n)$ Konfiguration.

Per Tiefensuche überprüfen wir alle Knoten auf Akzeptanz.

Zeitbedarf bei Tiefensuche: Linear in der Knotenzahl.

Für NSpace gab es $c^{s(n)}$ verschiedene Konfigurationen. Also:

$$\mathcal{N}Space(s(n)) \subseteq \mathcal{D}Time(c^{s(n)})$$

Da $\text{DSpace}(s(n)) \subseteq \text{NSpace}(s(n))$ gilt, haben wir auch:

Zeit versus Platz I

NTM: Im gerichteten Graphen aller Konfigurationen durchlaufen wir bis zum Akzeptieren maximal $s(n)$ Konfiguration.

Per Tiefensuche überprüfen wir alle Knoten auf Akzeptanz.

Zeitbedarf bei Tiefensuche: Linear in der Knotenzahl.

Für NSpace gab es $c^{s(n)}$ verschiedene Konfigurationen. Also:

$$\mathcal{N}Space(s(n)) \subseteq \mathcal{D}Time(c^{s(n)})$$

Da $\text{DSpace}(s(n)) \subseteq \text{NSpace}(s(n))$ gilt, haben wir auch:

$$\mathcal{D}Space(s(n)) \subseteq \mathcal{D}Time(c^{s(n)})$$

Zeit versus Platz II

Offensichtlich gilt ja

$$\mathcal{D}Time(f(n)) \subseteq \mathcal{D}Space(f(n)),$$

weil keine TM mehr Felder benutzen kann, als ihr Schritte zur Verfügung stehen!

Zeit versus Platz II

Offensichtlich gilt ja

$$\mathcal{D}Time(f(n)) \subseteq \mathcal{D}Space(f(n)),$$

weil keine TM mehr Felder benutzen kann, als ihr Schritte zur Verfügung stehen!

Das Ergebnis gilt sogar für nichtdeterministische TM:

Zeit versus Platz II

Offensichtlich gilt ja

$$\mathcal{D}Time(f(n)) \subseteq \mathcal{D}Space(f(n)),$$

weil keine TM mehr Felder benutzen kann, als ihr Schritte zur Verfügung stehen!

Das Ergebnis gilt sogar für nichtdeterministische TM:

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$

Zeit versus Platz II

Offensichtlich gilt ja

$$\mathcal{D}Time(f(n)) \subseteq \mathcal{D}Space(f(n)),$$

weil keine TM mehr Felder benutzen kann, als ihr Schritte zur Verfügung stehen!

Das Ergebnis gilt sogar für nichtdeterministische TM:

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$

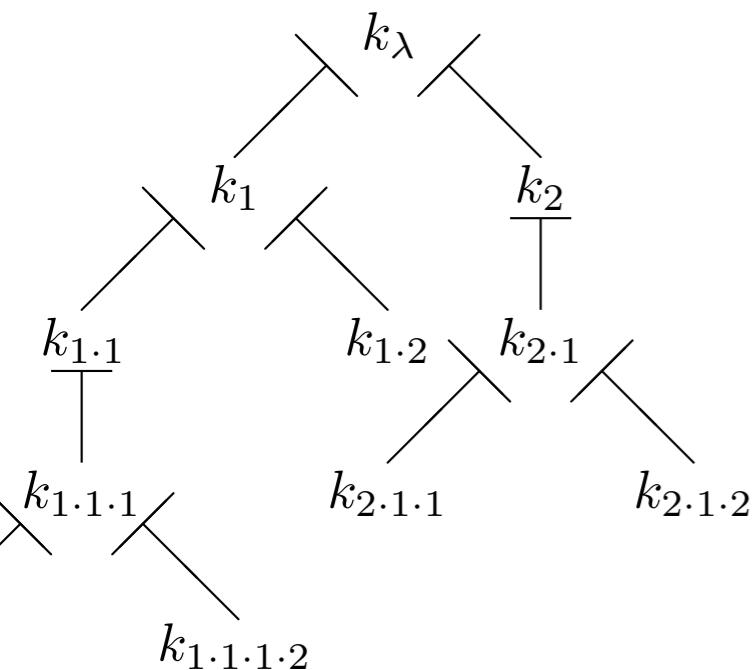
Dies ist bemerkenswert, denn die NTM kann ja Nachfolgekonfigurationen “raten”.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$

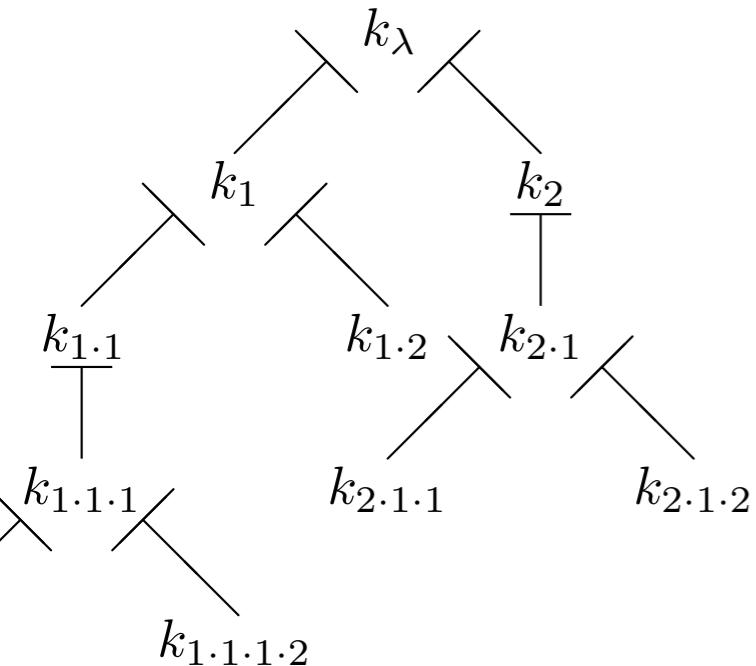
Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



Zeit versus Platz II

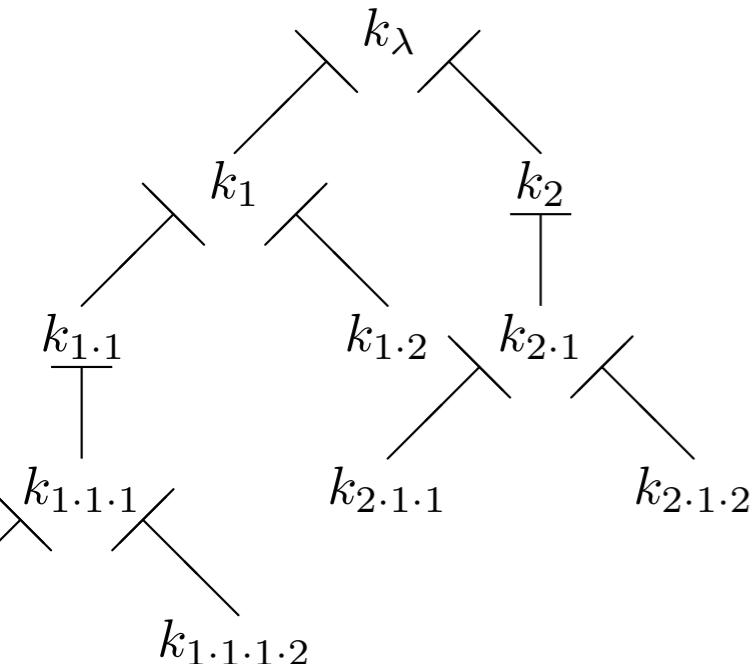
$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



Aufzählen der
Baumadressen:

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



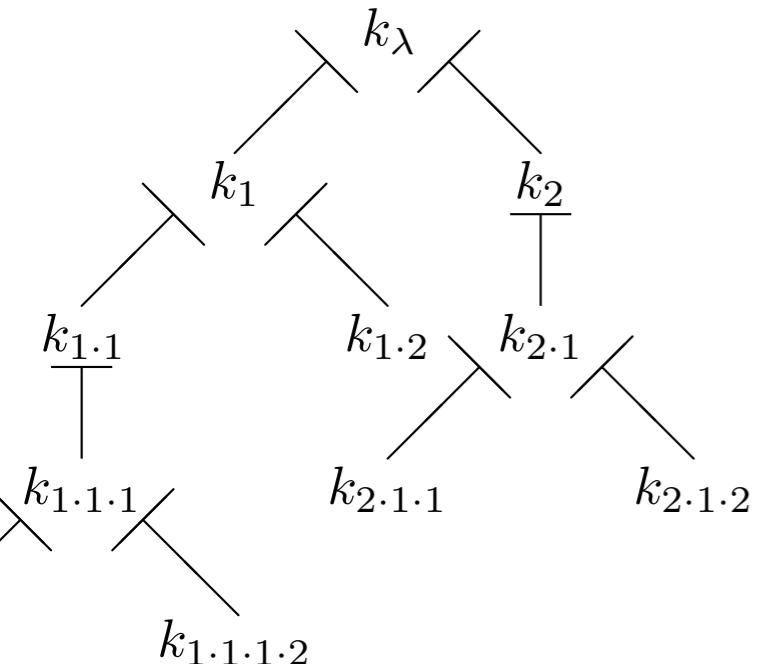
Aufzählen der ε
Baumadressen:

- 1
- 2
- 1.1
- 1.2
- 2.1
- 2.2
- 1.1.1
- 1.1.2
- usw.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$

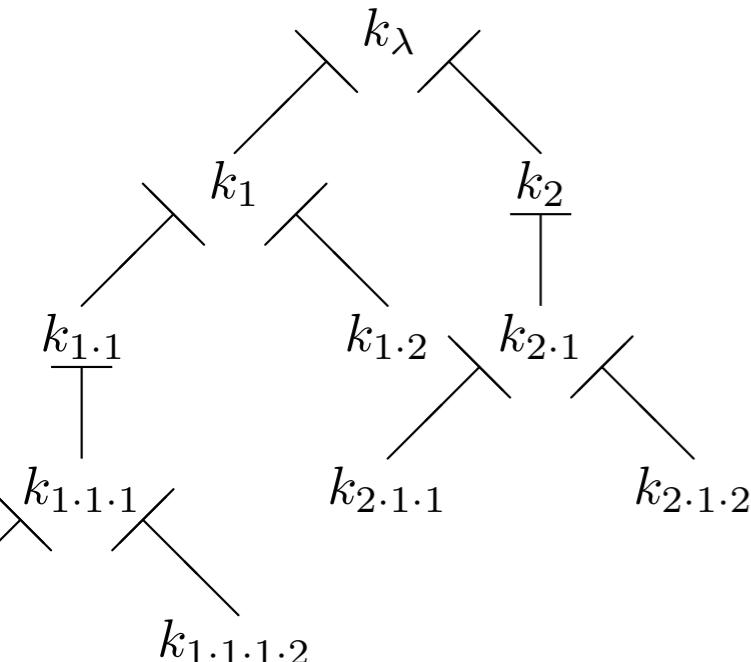
Beweisidee:



Aufzählen der ϵ
Baumadressen:
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



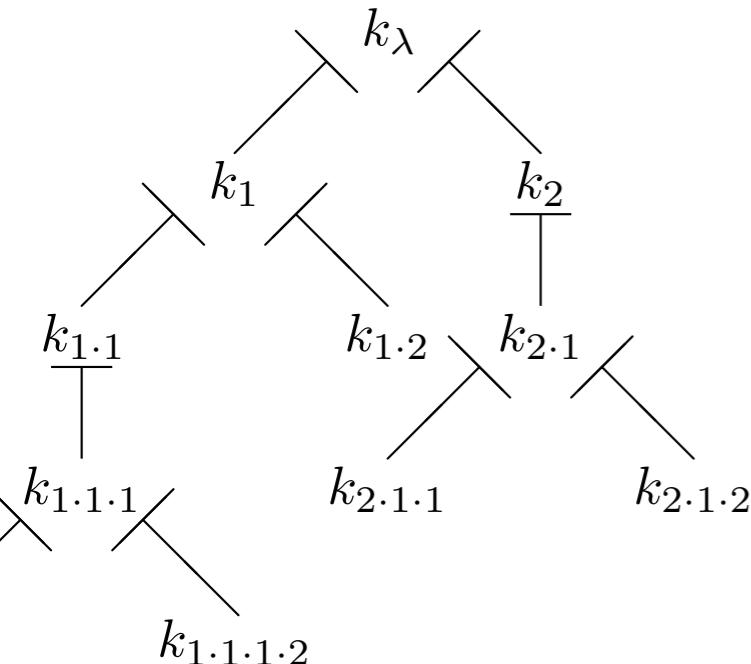
Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.

Aufzählen der ϵ
Baumadressen:
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



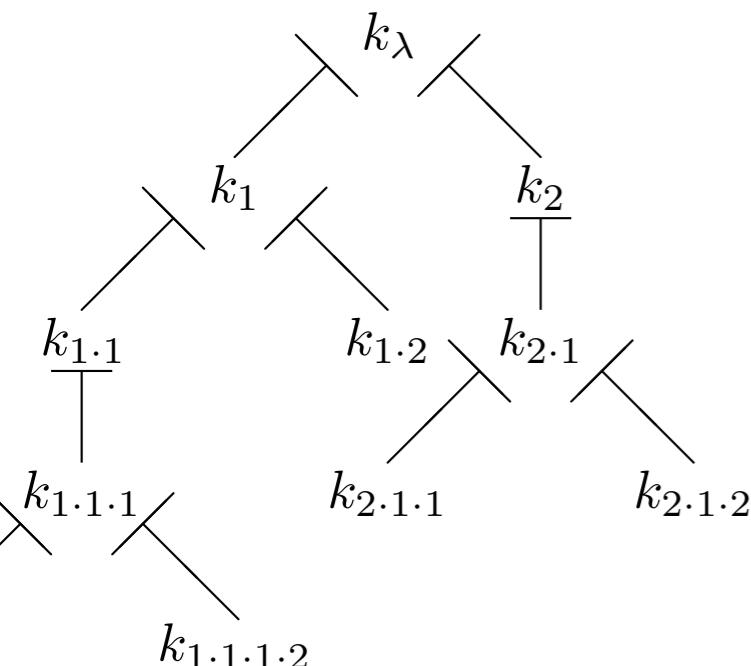
Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.
- Die DTM M_2 erzeugt aufsteigend alle Baumadressen w des Konfigurationsbaums von M_1 .

Aufzählen der ϵ
Baumadressen:
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



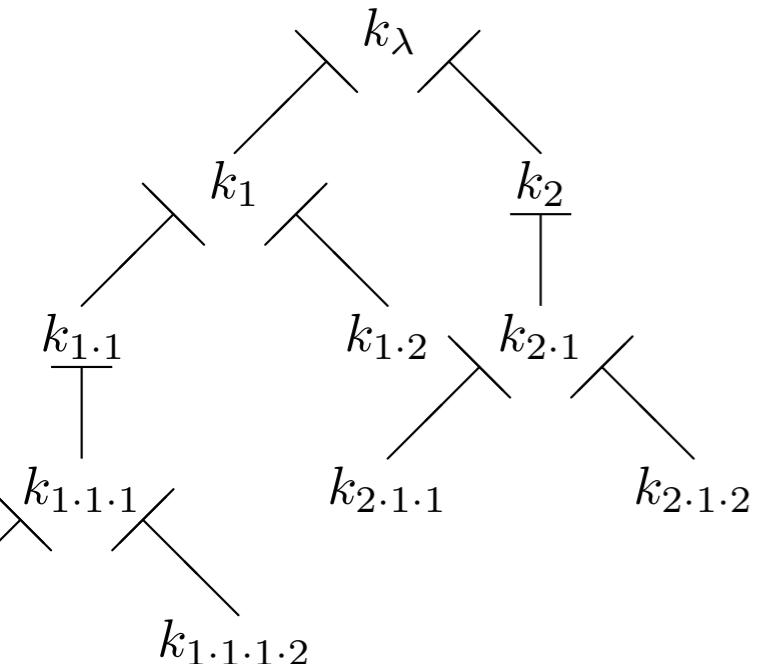
Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.
- Die DTM M_2 erzeugt aufsteigend alle Baumadressen w des Konfigurationsbaums von M_1 .
- Eine Baumadresse w ist ein Wort aus $\{1, \dots, d\}^{f(n)}$, wobei d der maximale Verzweigungsgrad ist.

Aufzählen der Baumadressen: ϵ
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



Aufzählen der Baumadressen:

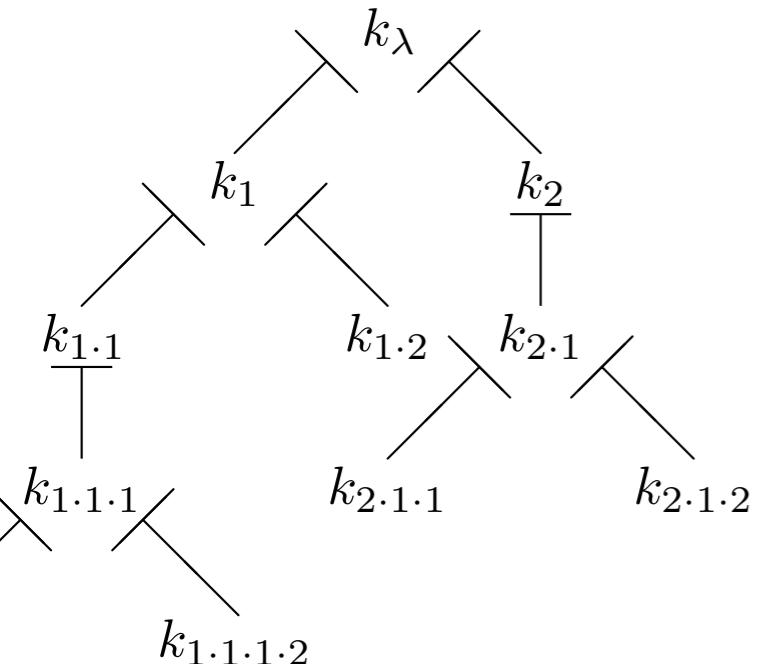
- ϵ
- 1
- 2
- 1.1
- 1.2
- 2.1
- 2.2
- 1.1.1
- 1.1.2
- usw.

Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.
- Die DTM M_2 erzeugt aufsteigend alle Baumadressen w des Konfigurationsbaums von M_1 .
- Eine Baumadresse w ist ein Wort aus $\{1, \dots, d\}^{f(n)}$, wobei d der maximale Verzweigungsgrad ist.
- Das Simulieren der von w erzeugten Rechnung benötigt nur $f(n)$ Zeit (und damit auch $f(n)$ Platz).

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



Aufzählen der
Baumadressen:

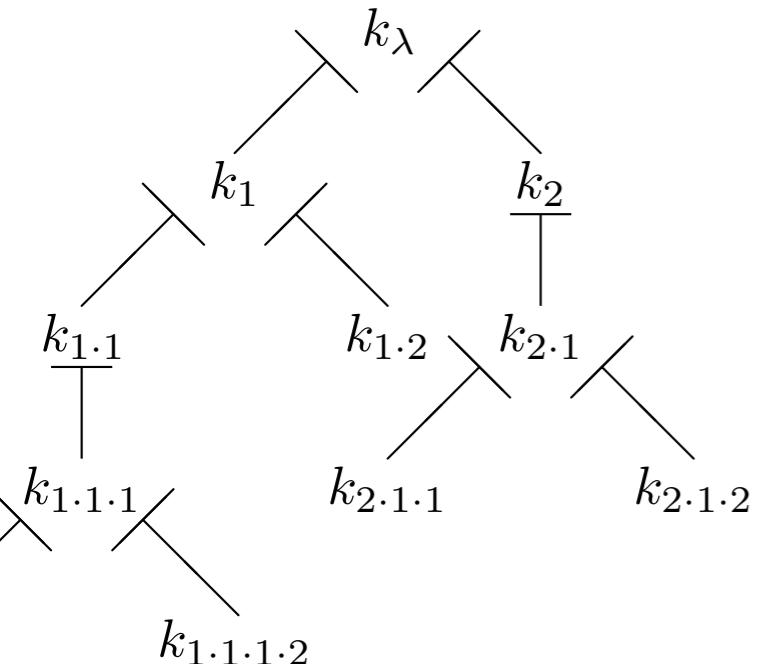
ϵ
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.
- Die DTM M_2 erzeugt aufsteigend alle Baumadressen w des Konfigurationsbaums von M_1 .
- Eine Baumadresse w ist ein Wort aus $\{1, \dots, d\}^{f(n)}$, wobei d der maximale Verzweigungsgrad ist.
- Das Simulieren der von w erzeugten Rechnung benötigt nur $f(n)$ Zeit (und damit auch $f(n)$ Platz).
- Es müssen exponentiell viele Rechnungen überprüft werden.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



Aufzählen der
Baumadressen:

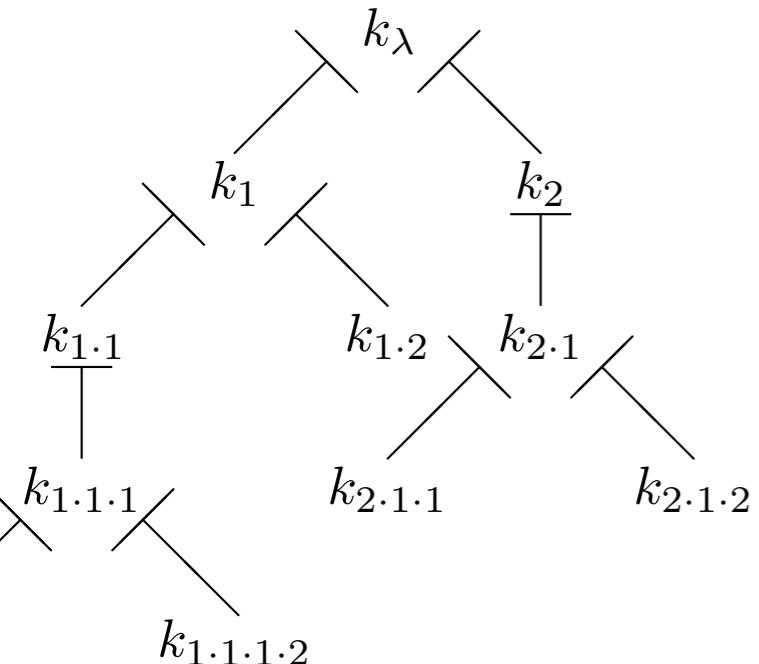
ϵ
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.
- Die DTM M_2 erzeugt aufsteigend alle Baumadressen w des Konfigurationsbaums von M_1 .
- Eine Baumadresse w ist ein Wort aus $\{1, \dots, d\}^{f(n)}$, wobei d der maximale Verzweigungsgrad ist.
- Das Simulieren der von w erzeugten Rechnung benötigt nur $f(n)$ Zeit (und damit auch $f(n)$ Platz).
- Es müssen exponentiell viele Rechnungen überprüft werden.
- Jede Simulation kann den gleichen Platz verwenden.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



Aufzählen der Baumadressen:

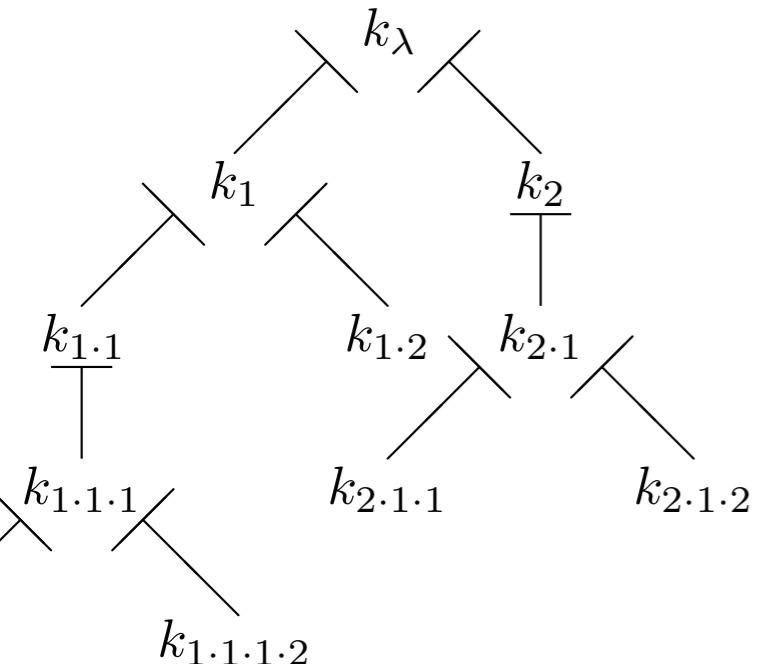
ϵ
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.
- Die DTM M_2 erzeugt aufsteigend alle Baumadressen w des Konfigurationsbaums von M_1 .
- Eine Baumadresse w ist ein Wort aus $\{1, \dots, d\}^{f(n)}$, wobei d der maximale Verzweigungsgrad ist.
- Das Simulieren der von w erzeugten Rechnung benötigt nur $f(n)$ Zeit (und damit auch $f(n)$ Platz).
- Es müssen exponentiell viele Rechnungen überprüft werden.
 - Jede Simulation kann den gleichen Platz verwenden.
 - Jede Folge w kann in $c \cdot f(n)$ Bandfeldern gespeichert werden.

Zeit versus Platz II

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$



Aufzählen der
Baumadressen:

ϵ
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2
usw.

Beweisidee:

- Sei M_1 eine NTM, die L in $f(n)$ Zeit akzeptiert.
- Die DTM M_2 erzeugt aufsteigend alle Baumadressen w des Konfigurationsbaums von M_1 .
- Eine Baumadresse w ist ein Wort aus $\{1, \dots, d\}^{f(n)}$, wobei d der maximale Verzweigungsgrad ist.
- Das Simulieren der von w erzeugten Rechnung benötigt nur $f(n)$ Zeit (und damit auch $f(n)$ Platz).
- Es müssen exponentiell viele Rechnungen überprüft werden.
 - Jede Simulation kann den gleichen Platz verwenden.
 - Jede Folge w kann in $c \cdot f(n)$ Bandfeldern gespeichert werden.
 - Simulation braucht insgesamt nur $c \cdot f(n)$ Bandfelder.

wichtige Folgerung

$$\mathcal{N}Space(s(n)) \subseteq \mathcal{D}Time(c^{s(n)})$$

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$

wichtige Folgerung

$$\mathcal{N}Space(s(n)) \subseteq \mathcal{D}Time(c^{s(n)})$$

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$

Korollar: Akzeptiert eine Turingmaschine eine Sprache L mit einer Zeitbeschränkung $t(n)$
oder mit einer Platzbeschränkung $s(n)$,
so ist L auch eine **entscheidbare** Menge.

wichtige Folgerung

$$\mathcal{N}Space(s(n)) \subseteq \mathcal{D}Time(c^{s(n)})$$

$$\mathcal{N}Time(f(n)) \subseteq \mathcal{D}Space(f(n))$$

Korollar: Akzeptiert eine Turingmaschine eine Sprache L mit einer Zeitbeschränkung $t(n)$ oder mit einer Platzbeschränkung $s(n)$, so ist L auch eine entscheidbare Menge.

Die Familien

\mathcal{P} , \mathcal{NP} , \mathcal{PSpace} , $\mathcal{NPspace}$

und **alle** über Komplexitätsbeschränkungen (mit berechenbaren Funktionen) definierte Sprachfamilien sind Teil der Familie der entscheidbaren Sprachen!

Beziehungen zwischen den Klassen

Es gilt:

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSpace} \subseteq \mathcal{NPspace}$$

Dies folgt aus den Definitionen und der bereits bewiesenen Inklusion: $\mathcal{NTime}(f(n)) \subseteq \mathcal{DSpace}(f(n))$

Beziehungen zwischen den Klassen

Es gilt:

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSpace} \subseteq \mathcal{NPspace}$$

Dies folgt aus den Definitionen und der bereits bewiesenen Inklusion: $\mathcal{NTime}(f(n)) \subseteq \mathcal{DSpace}(f(n))$

Nun zum Zusammenhang zwischen
Determinismus und Nichtdeterminismus
in Bezug auf den **Platz**.

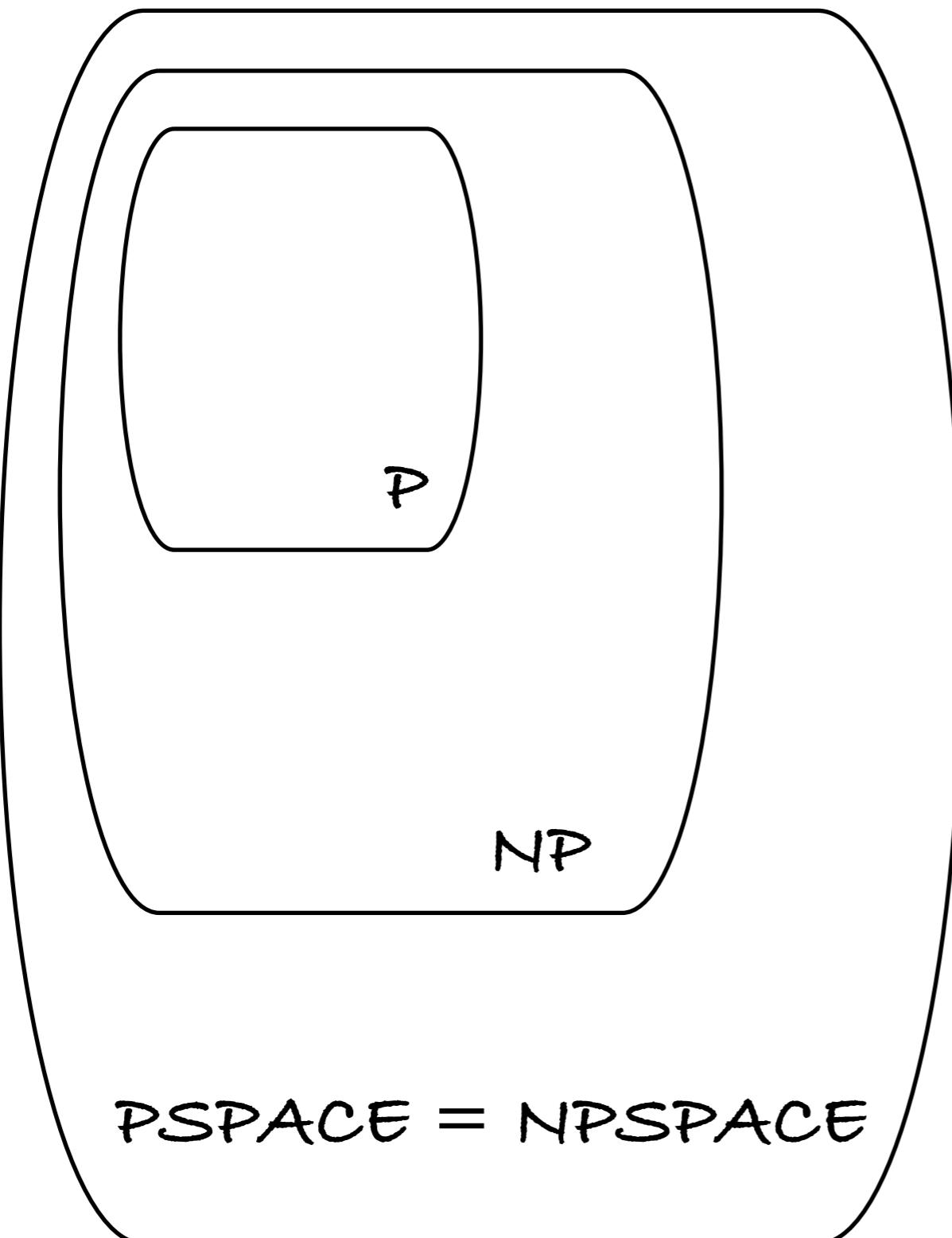
polynomiale Platzklassen

Definition 23.2:

$$\mathcal{P}Space := \bigcup_{i \geq 1} \mathcal{D}Space(n^i)$$

$$\mathcal{NP}Space := \bigcup_{i \geq 1} \mathcal{N}Space(n^i)$$

Zusammenhang



Walter J. Savitch
zeigte 1970:

$$\begin{aligned} & \text{NSpace}(s(n)) \\ & \subseteq \text{DSpace}(s(n)^2) \end{aligned}$$

Satz von Savitch

Satz 23.1:

$$NSpace(s) \subseteq DSpace(s^2)$$

für jede platzkonstruierbare Funktion s ,

für die stets $s(n) \geq \log n$ ist.

Platz- / Zeit-konstruierbar nennt man Funktionen,
die mit der selben Komplexitätsbeschränkung
berechnet werden können!

Beweis: Satz von Savitch

Da die NTM $s(n)$ platzbeschränkt ist, durchläuft sie bis zum Akzeptieren höchstens \max Konfigurationen:

$$\max = 2^{O(s(n))}$$

Beweis: Satz von Savitch

Da die NTM $s(n)$ platzbeschränkt ist, durchläuft sie bis zum Akzeptieren höchstens \max Konfigurationen:

$$\max = 2^{O(s(n))}$$

Nacheinander testen wir alle diese Konfigurationen aus:

Beweis: Satz von Savitch

Da die NTM $s(n)$ platzbeschränkt ist, durchläuft sie bis zum Akzeptieren höchstens \max Konfigurationen:

$$\max = 2^{O(s(n))}$$

Nacheinander testen wir alle diese Konfigurationen aus:

```
FOREACH  $k_n$  IN  $k_1, \dots, k_{\max}$  DO
    IF reachable( $q_{0W}, k_n, \max$ )
    AND  $k_n$  is Endkonfiguration THEN
        akzeptiere und terminiere;
    verwerfe und terminiere;
```

Beweis: Satz von Savitch

Da die NTM $s(n)$ platzbeschränkt ist, durchläuft sie bis zum Akzeptieren höchstens \max Konfigurationen:

$$\max = 2^{O(s(n))}$$

Nacheinander testen wir alle diese Konfigurationen:

```
FOREACH  $k_n$  IN  $k_1, \dots, k_{\max}$  DO  
    IF reachable( $q_{0W}, k_n, \max$ )  
    AND  $k_n$  is Endkonfiguration THEN  
        akzeptiere und terminiere;  
    verwerfe und terminiere;
```

von q_{0W} nach k_n
in \max Schritten

Beweis: Satz von Savitch

Da die NTM $s(n)$ platzbeschränkt ist, durchläuft sie bis zum Akzeptieren höchstens \max Konfigurationen:

$$\max = 2^{O(s(n))}$$

Nacheinander testen wir alle diese Konfigurationen:

```
FOREACH  $k_n$  IN  $k_1, \dots, k_{\max}$  DO  
    IF reachable( $q_{0W}, k_n, \max$ )  
        AND  $k_n$  is Endkonfiguration THEN  
            akzeptiere und terminiere;  
        verwerfe und terminiere;
```

von q_{0W} nach k_n
in \max Schritten

Wir brauchen dabei Platz für

- den Zähler n bzw. die Konfiguration k_n und
- den Speicherplatz der Prozedur `reachable()`.

Beweis: Satz von Savitch

Da die NTM $s(n)$ platzbeschränkt ist, durchläuft sie bis zum Akzeptieren höchstens \max Konfigurationen:

$$\max = 2^{O(s(n))}$$

Nacheinander testen wir alle diese Konfigurationen:

```
FOREACH  $k_n$  IN  $k_1, \dots, k_{\max}$  DO  
    IF reachable( $q_{0W}, k_n, \max$ )  
    AND  $k_n$  is Endkonfiguration THEN  
        akzeptiere und terminiere;  
    verwerfe und terminiere;
```

von q_{0W} nach k_n
in \max Schritten

$O(s(n))$

Wir brauchen dabei Platz für

- den Zähler n bzw. die Konfiguration k_n und
- den Speicherplatz der Prozedur `reachable()`.

Beweis: Satz von Savitch

Da die NTM $s(n)$ platzbeschränkt ist, durchläuft sie bis zum Akzeptieren höchstens \max Konfigurationen:

$$\max = 2^{O(s(n))}$$

Nacheinander testen wir alle diese Konfigurationen:

```
FOREACH  $k_n$  IN  $k_1, \dots, k_{\max}$  DO  
    IF reachable( $q_{0W}, k_n, \max$ )  
        AND  $k_n$  is Endkonfiguration THEN  
            akzeptiere und terminiere;  
        verwerfe und terminiere;
```

von q_{0W} nach k_n
in \max Schritten

Wir brauchen dabei Platz für

- den Zähler n bzw. die Konfiguration k_n und
- den Speicherplatz der Prozedur `reachable()`.

$O(s(n))$

$O(s(n)^2)$

Beweis: Satz von Savitch

Wir teilen die zu findende Rechnung in zwei Teile auf.
Beide Teilrechnungen sind dann halb so lang.

Beweis: Satz von Savitch

Wir teilen die zu findende Rechnung in zwei Teile auf.

Beide Teilrechnungen sind dann halb so lang.

```
PROC reachable( $k, k'$ , length) IS
  IF  $length=1$  THEN RETURN ( $k = k'$  OR  $k \vdash k'$ );
  FOREACH  $k_n$  IN  $k_1, \dots, k_{max}$  DO
    IF reachable( $k, k_n, length/2$ )
      AND reachable( $k_n, k', length/2$ )
      THEN RETURN true;
  RETURN false;
```

Beweis: Satz von Savitch

Wir teilen die zu findende Rechnung in zwei Teile auf.

Beide Teilrechnungen sind dann halb so lang.

```
PROC reachable( $k, k', length$ ) IS
    IF  $length=1$  THEN RETURN ( $k = k'$  OR  $k \vdash k'$ );
    FOREACH  $k_n$  IN  $k_1, \dots, k_{max}$  DO
        IF reachable( $k, k_n, length/2$ )
        AND reachable( $k_n, k', length/2$ )
        THEN RETURN true;
    RETURN false;
```

Stets ist zu jeder Länge $length$ höchstens ein Aufruf aktiv:

- $\text{reachable}(k, k', max)$,
- $\text{reachable}(k, k', max/2)$,
- $\text{reachable}(k, k', max/4)$,
-
- $\text{reachable}(k, k', 1)$

Beweis: Satz von Savitch

Wir teilen die zu findende Rechnung in zwei Teile auf.

Beide Teilrechnungen sind dann halb so lang.

```
PROC reachable( $k, k', length$ ) IS
    IF  $length=1$  THEN RETURN ( $k = k'$  OR  $k \vdash k'$ );
    FOREACH  $k_n$  IN  $k_1, \dots, k_{max}$  DO
        IF reachable( $k, k_n, length/2$ )
        AND reachable( $k_n, k', length/2$ )
        THEN RETURN true;
    RETURN false;
```

Stets ist zu jeder Länge $length$ höchstens ein Aufruf aktiv:

- $\text{reachable}(k, k', max)$,
- $\text{reachable}(k, k', max/2)$,
- $\text{reachable}(k, k', max/4)$,
-
- $\text{reachable}(k, k', 1)$

Anzahl der Aufrufe: $\log_2(max) = \log_2(2^{O(s(n))}) = O(s(n))$

Beweis: Satz von Savitch

Wir teilen die zu findende Rechnung in zwei Teile auf.

Beide Teilrechnungen sind dann halb so lang.

```
PROC reachable( $k, k', length$ ) IS
    IF  $length=1$  THEN RETURN ( $k = k'$  OR  $k \vdash k'$ );
    FOREACH  $k_n$  IN  $k_1, \dots, k_{max}$  DO
        IF reachable( $k, k_n, length/2$ )
        AND reachable( $k_n, k', length/2$ )
        THEN RETURN true;
    RETURN false;
```

Stets ist zu jeder Länge $length$ höchstens ein Aufruf aktiv:

- $\text{reachable}(k, k', max)$,
- $\text{reachable}(k, k', max/2)$,
- $\text{reachable}(k, k', max/4)$,
- ...
- $\text{reachable}(k, k', 1)$

Anzahl der Aufrufe: $\log_2(max) = \log_2(2^{O(s(n))}) = O(s(n))$

Also $O(s(n))$ Aufrufe, die jeweils $O(s(n))$ Platz benötigen: $O(s(n)^2)$ qed.

Beziehungen zwischen den Klassen

Also gilt:

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{P}Space = \mathcal{NP}Space$$

Beziehungen zwischen den Klassen

Also gilt:

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{P}\text{Space} = \mathcal{NP}\text{Space}$$

Nun zum Zusammenhang zwischen
Determinismus und Nichtdeterminismus
in Bezug auf die Zeit.

Beziehungen zwischen den Klassen

Also gilt:

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{P}\text{Space} = \mathcal{NP}\text{Space}$$

Nun zum Zusammenhang zwischen
Determinismus und Nichtdeterminismus
in Bezug auf die Zeit.

Insbesondere: Ist die
Inklusion $\mathcal{P} \subseteq \mathcal{NP}$ echt?

P und die Praxis

- Von Problemen aus der Komplexitätsklasse \mathcal{P} sagt man, dass sie **effizient lösbar** seien.
- Probleme in \mathcal{P} sind anscheinend wirklich in der Praxis noch mit vertretbarem Aufwand lösbar.
 - Nehmen wir also an, dass tatsächlich alle Probleme in \mathcal{P} in der Praxis vernünftig gelöst werden können.
 - Was ist dann mit \mathcal{NP} ?
 - Sind Probleme aus \mathcal{NP} nicht mehr praktisch verwendbar/lösbar?
 - Handelt es sich nur um praktisch nicht relevante Probleme?

P und die Praxis

- Von Problemen aus der Komplexitätsklasse \mathcal{P} sagt man, dass sie **effizient lösbar** seien.
- Probleme in \mathcal{P} sind anscheinend wirklich in der Praxis noch mit vertretbarem Aufwand lösbar.
- Nehmen wir also an, dass tatsächlich alle Probleme in \mathcal{P} in der Praxis vernünftig gelöst werden können.

- Was ist dann mit \mathcal{NP} ?
 - Sind Probleme aus \mathcal{NP} nicht mehr praktisch verwendbar/lösbar?
 - Handelt es sich nur um praktisch nicht relevante Probleme?

P und die Praxis

- Von Problemen aus der Komplexitätsklasse \mathcal{P} sagt man, dass sie **effizient lösbar** seien.
- Probleme in \mathcal{P} sind anscheinend wirklich in der Praxis noch mit vertretbarem Aufwand lösbar.
- Nehmen wir also an, dass tatsächlich alle Probleme in \mathcal{P} in der Praxis vernünftig gelöst werden können.
 - Was ist dann mit \mathcal{NP} ?
 - Sind Probleme aus \mathcal{NP} nicht mehr praktisch verwendbar/lösbar?
 - Handelt es sich nur um praktisch nicht relevante Probleme?

Beispiele von Problemen in P und NP:

- Ähnliche Probleme liegen trotzdem oft in (vielleicht wirklich) unterschiedlichen Komplexitätsklassen!

Gegeben: Matrix $A \in \mathbb{Z}^{m \times n}$ und Vektor $b \in \mathbb{Z}^m$	Frage: Gibt es $x \in \mathbb{Z}^n$ mit $Ax = b$	Frage: Gibt es $x \in \mathbb{N}^n$ mit $Ax = b$
\mathcal{P}		\mathcal{NP}

Gegeben: Ungerichteter, kantenbewerteter Graph $G = (V, E)$, eine Gewichtsfunktion $g : E \rightarrow \mathbb{N}$, zwei Knoten $s, t \in V$ und eine Schranke $k \in \mathbb{N}$	Frage: Gibt es einen einfachen Pfad p von s nach t mit $g(p) \leq k$? („Kürzester Weg zwischen zwei Knoten“)	Frage: Gibt es einen einfachen Pfad p von s nach t mit $g(p) \geq k$? („Längster Weg zwischen zwei Knoten“)
\mathcal{P}		\mathcal{NP}

weitere Beispiele von Problemen in P und NP:

- $L_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } S \text{ nach } t \text{ mit } g(p) \geq k\}$ ist die dem Problem „Längster Weg zwischen zwei Knoten“ zugeordnete Sprache.

- **Theorem:** Das Problem zu L_w ist nichtdeterministisch in Polynomzeit zu lösen, d.h. $L_w \in \mathcal{NP}$.
Idee: Raten eines geeigneten Pfades.

- $K_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } s \text{ nach } t \text{ mit } g(p) \leq k\}$ ist die Sprache, die zu dem Problem „Kürzester Weg zwischen zwei Knoten“ gehört.

- **Theorem:** Das Problem zu K_w ist in \mathcal{P} .
Idee: Systematische Suche der minimalen Pfadlängen.

weitere Beispiele von Problemen in P und NP:

- $L_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } S \text{ nach } t \text{ mit } g(p) \geq k\}$ ist die dem Problem „Längster Weg zwischen zwei Knoten“ zugeordnete Sprache.
- **Theorem:** Das Problem zu L_w ist nichtdeterministisch in Polynomzeit zu lösen, d.h. $L_w \in \mathcal{NP}$.
Idee: Raten eines geeigneten Pfades.

- $K_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } s \text{ nach } t \text{ mit } g(p) \leq k\}$ ist die Sprache, die zu dem Problem „Kürzester Weg zwischen zwei Knoten“ gehört.
- **Theorem:** Das Problem zu K_w ist in \mathcal{P} .
Idee: Systematische Suche der minimalen Pfadlängen.

weitere Beispiele von Problemen in P und NP:

- $L_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } S \text{ nach } t \text{ mit } g(p) \geq k\}$ ist die dem Problem „Längster Weg zwischen zwei Knoten“ zugeordnete Sprache.

- **Theorem:** Das Problem zu L_w ist nichtdeterministisch in Polynomzeit zu lösen, d.h. $L_w \in \mathcal{NP}$.
Idee: Raten eines geeigneten Pfades.

- $K_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } s \text{ nach } t \text{ mit } g(p) \leq k\}$ ist die Sprache, die zu dem Problem „Kürzester Weg zwischen zwei Knoten“ gehört.

- **Theorem:** Das Problem zu K_w ist in \mathcal{P} .
Idee: Systematische Suche der minimalen Pfadlängen.

weitere Beispiele von Problemen in P und NP:

- $L_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } S \text{ nach } t \text{ mit } g(p) \geq k\}$ ist die dem Problem „Längster Weg zwischen zwei Knoten“ zugeordnete Sprache.

- **Theorem:** Das Problem zu L_w ist nichtdeterministisch in Polynomzeit zu lösen, d.h. $L_w \in \mathcal{NP}$.
Idee: Raten eines geeigneten Pfades.

- $K_w := \{\langle G, s, t, k \rangle \mid G \text{ besitzt einen Pfad } p \text{ von } s \text{ nach } t \text{ mit } g(p) \leq k\}$ ist die Sprache, die zu dem Problem „Kürzester Weg zwischen zwei Knoten“ gehört.

- **Theorem:** Das Problem zu K_w ist in \mathcal{P} .
Idee: Systematische Suche der minimalen Pfadlängen.

weitere Probleme in P und NP

Gegeben: Ungerichteter Graph $G = (V, E)$

Frage: Gibt es einen geschlossenen Kreis, in dem jede Kante genau einmal auftritt? („Euler-Kreis“)

\mathcal{P}

Frage: Gibt es einen geschlossenen Kreis, in dem jeder Knoten genau einmal auftritt? („Hamilton-Kreis“)

\mathcal{NP}

- (Leider) sind häufig Probleme aus \mathcal{NP} die (für die Praxis) interessantesten!
- Manchmal kann man das aber auch ausnutzen. Z.B. für die Kryptographie.

Warum ist Eulerkreis in P ?

Formel in Baumdarstellung bottom-up auswerten

Problem: "Erfüllt die Belegung A die Formel F ?"

Formel in Baumdarstellung bottom-up auswerten

Problem: "Erfüllt die Belegung A die Formel F ?"

Umformuliert als Sprache:

$$L = \{ \langle F, A \rangle \mid F \text{ ist eine Formel und } A(F)=1 \}$$

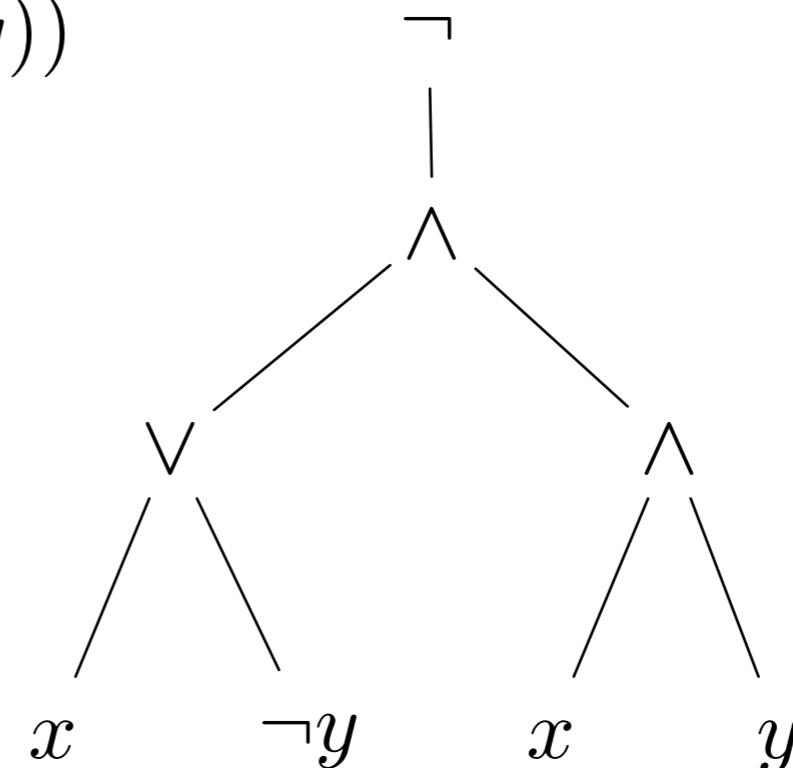
Formel in Baumdarstellung bottom-up auswerten

Problem: "Erfüllt die Belegung A die Formel F ?"

Umformuliert als Sprache:

$$L = \{ \langle F, A \rangle \mid F \text{ ist eine Formel und } A(F)=1 \}$$

Beispiel: $\neg((x \vee \neg y) \wedge (x \wedge y))$

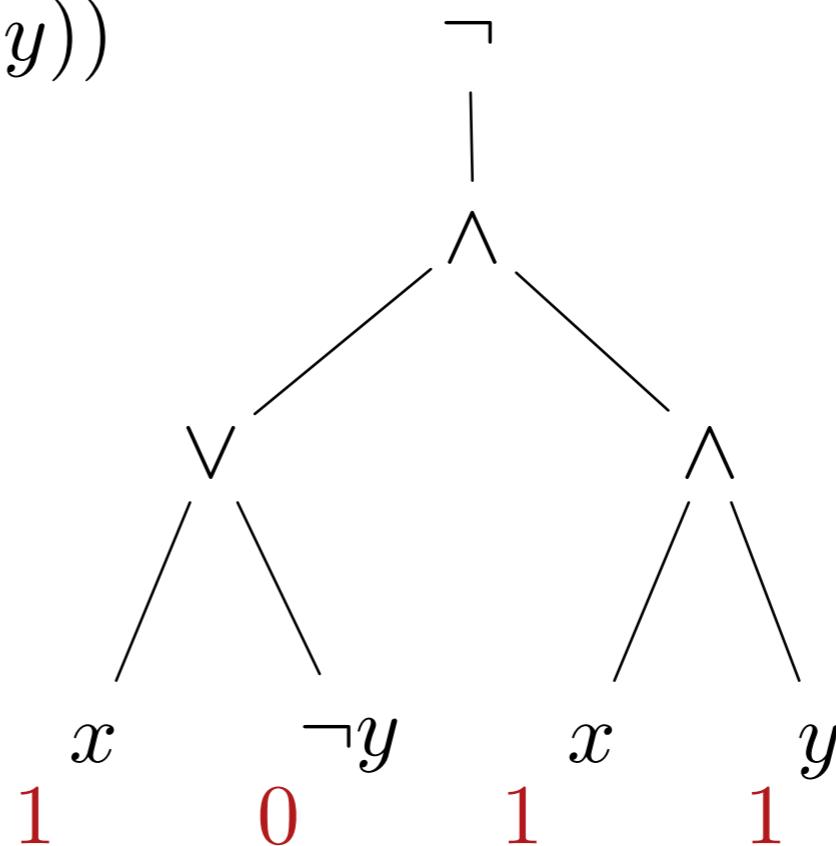


Belegung an Blätter notieren

$$\neg((x \vee \neg y) \wedge (x \wedge y))$$

$x := 1$

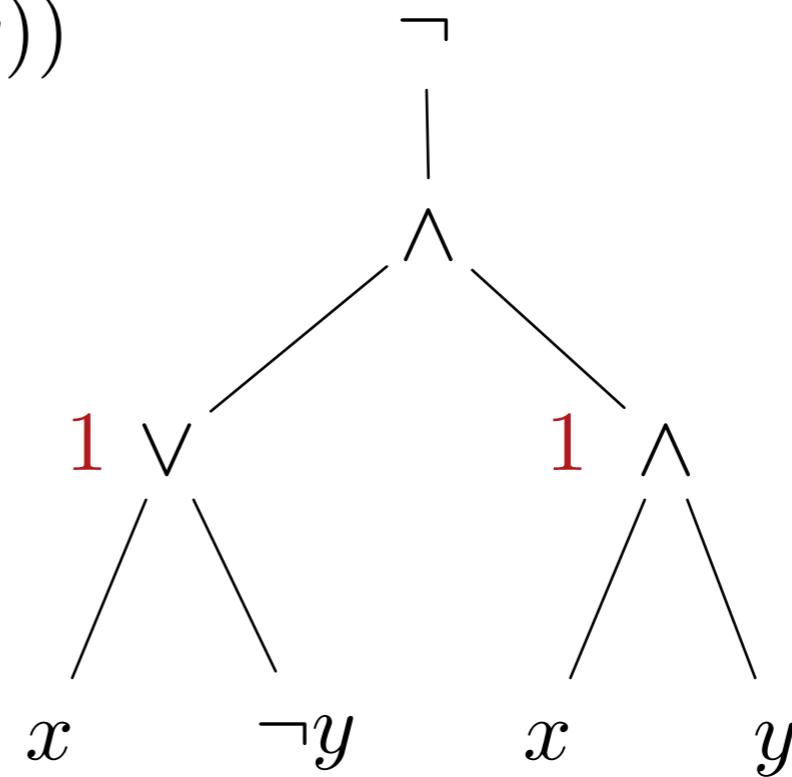
$y := 1$



Operator bekommt Auswertungswert

$$\neg((x \vee \neg y) \wedge (x \wedge y))$$

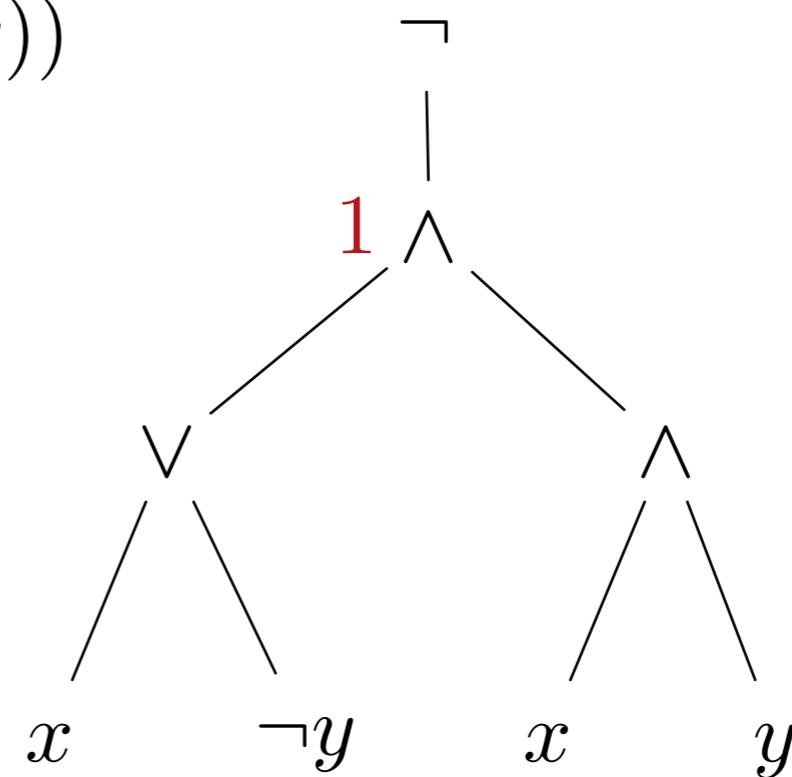
$x := 1$
 $y := 1$



Operator bekommt Auswertungswert

$$\neg((x \vee \neg y) \wedge (x \wedge y))$$

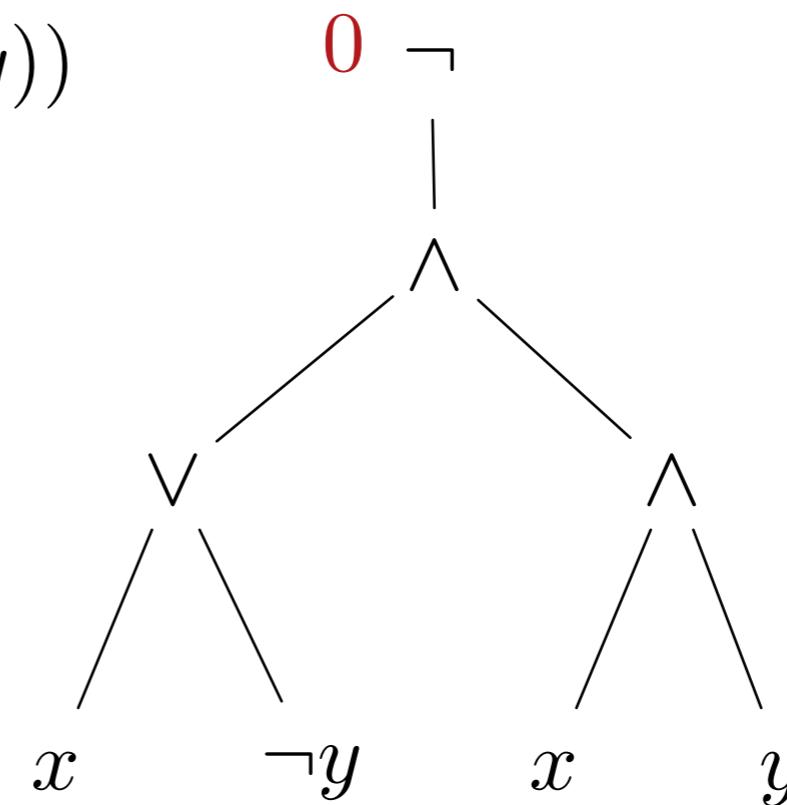
$x := 1$
 $y := 1$



Wurzel bekommt Auswertungswert der Formel

$$\neg((x \vee \neg y) \wedge (x \wedge y))$$

$$\begin{aligned}x &:= 1 \\y &:= 1\end{aligned}$$



Anzahl aller Knoten ist n . An jedem Knoten ist konstanter Aufwand zur Wertberechnung nötig. Insgesamt ist Berechnung also polynomial in der Länge der Formel.

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 24

NP-Vollständigkeit, Reduktionen

Michael Köhler-Bußmeier

Machbarkeit bei Polynomzeit?

- Jedes Problem $L \in \mathcal{NP}$, ist mit exponentiellem Zeitaufwand (in $\mathcal{DTIME}(2^{p(n)})$ für ein Polynom p) deterministisch lösbar.
 - Bessere Transformationen sind i.a. nicht bekannt.
- Viele der *praktisch wichtigen* Fragestellungen haben Lösungen mit Algorithmen, die **nur dann in Polynomzeit arbeiten, wenn sie nicht-deterministisch sind**, jedenfalls kennt man zur Zeit noch nicht Besseres!
- Eine **Implementierung** erfordert ein **deterministisches Verfahren**.
- Unterscheiden sich \mathcal{P} und \mathcal{NP} überhaupt?!

DTM-Simulation der NTM ist schlecht

Da die Simulation einer NTM durch eine DTM auf die bekannte Weise **exponentiell** mehr Schritte benötigt, ist dadurch für NP-vollständige Probleme nichts gewonnen!

Bisher ist die Frage:

„ $P = NP?$ “ bzw. „ $P \neq NP?$ “

nicht beantwortet.

Viele vermuten $P \neq NP$, aber bewiesen ist es nicht!

DTM-Simulation der NTM ist schlecht

Da die Simulation einer NTM durch eine DTM auf die bekannte Weise eingeschränkt ist, benötigt, ist dadurch für NP-vollständige Probleme gewonnen!

Würde die Simulation nur **polynomiell** mehr Schritte benötigen, dann wäre:

$$P = NP$$

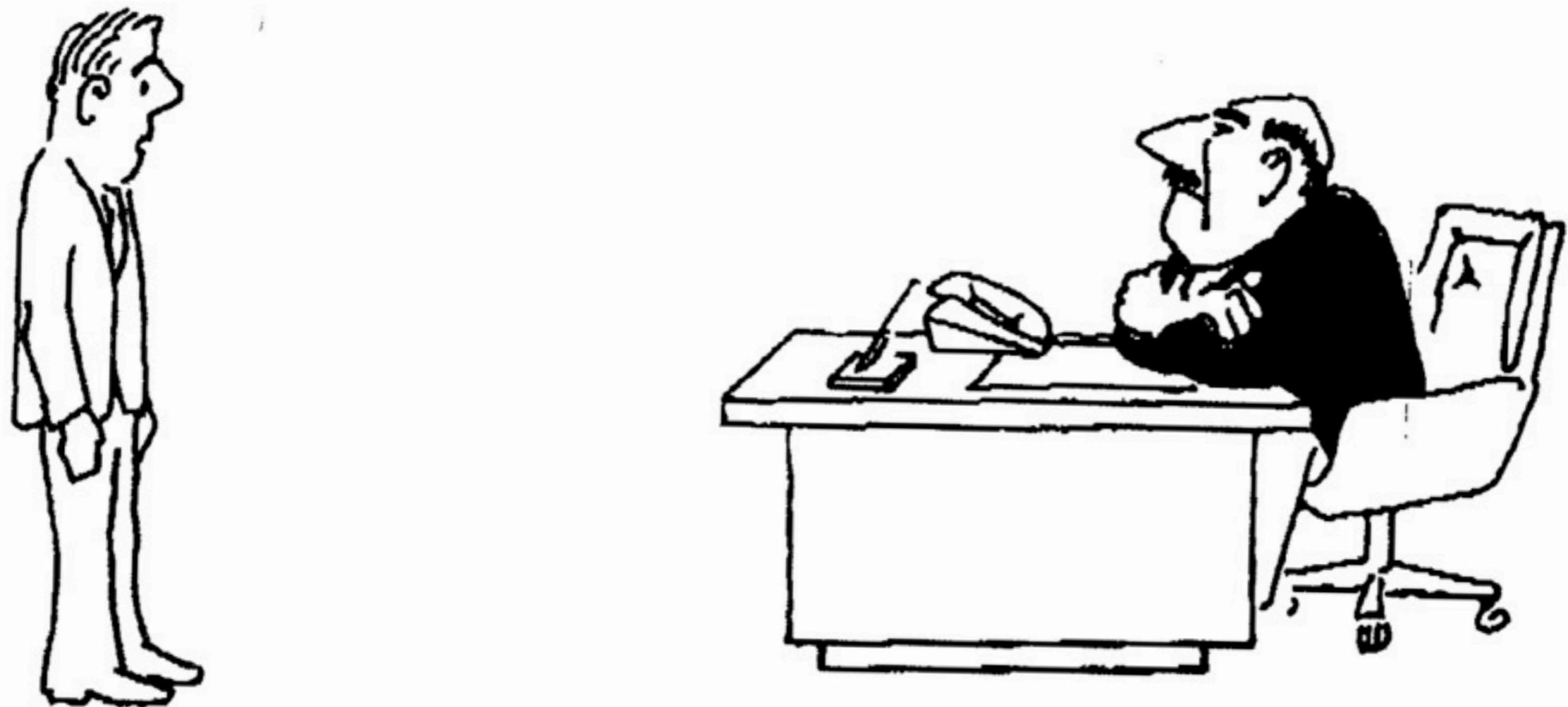
Bisher ist ~~die~~ Frage:

$$\text{„}P = NP? \text{“ bzw. „}P \neq NP? \text{“}$$

nicht beantwortet.

Viele vermuten $P \neq NP$, aber bewiesen ist es nicht!

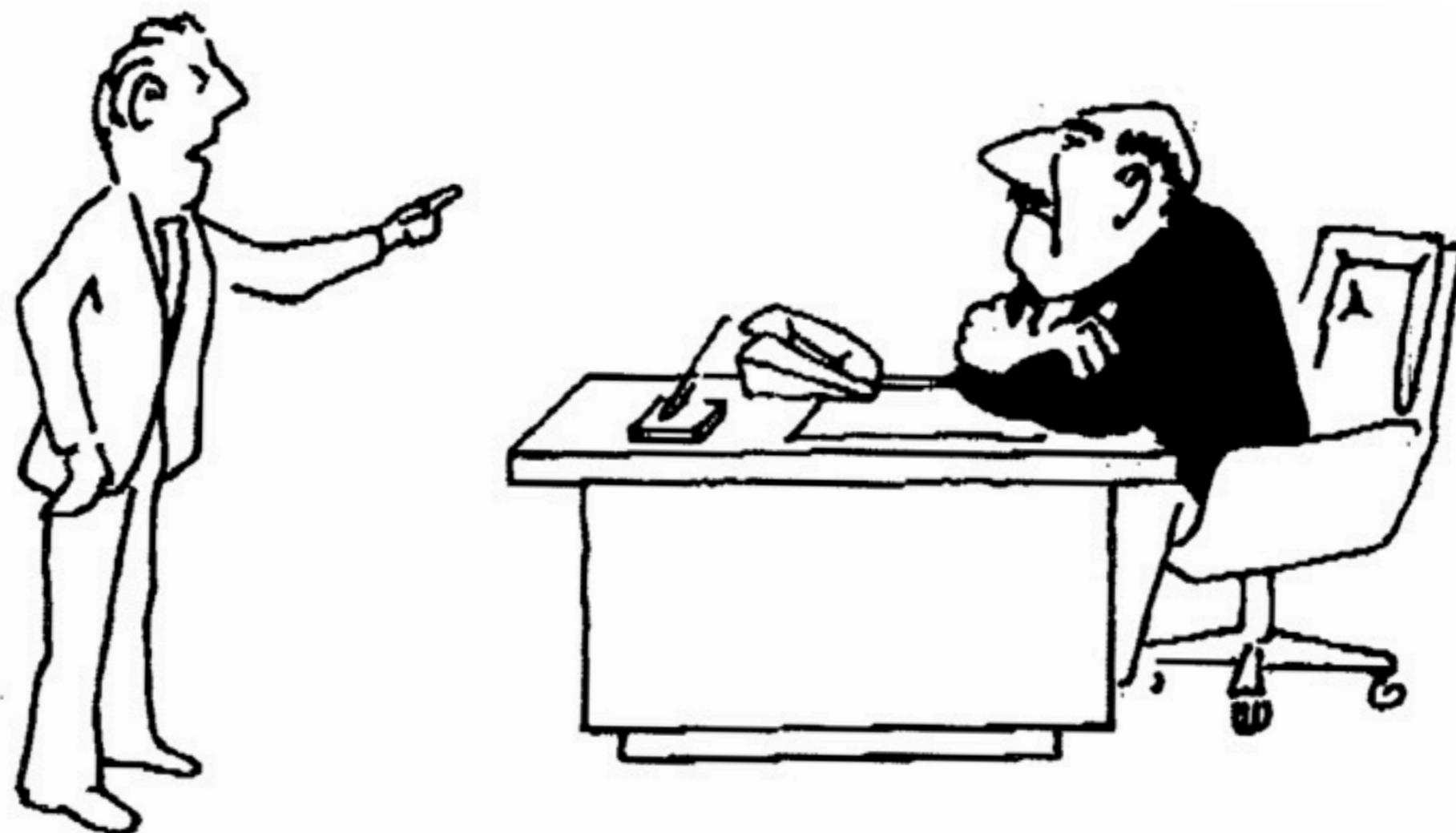
Eine wirklich dumme Antwort



“I can’t find an efficient algorithm, I guess I’m just too dumb.”

Obwohl korrekt, für Praktiker inakzeptabel!

Manchmal kennen wir untere Schranken der Komplexität:



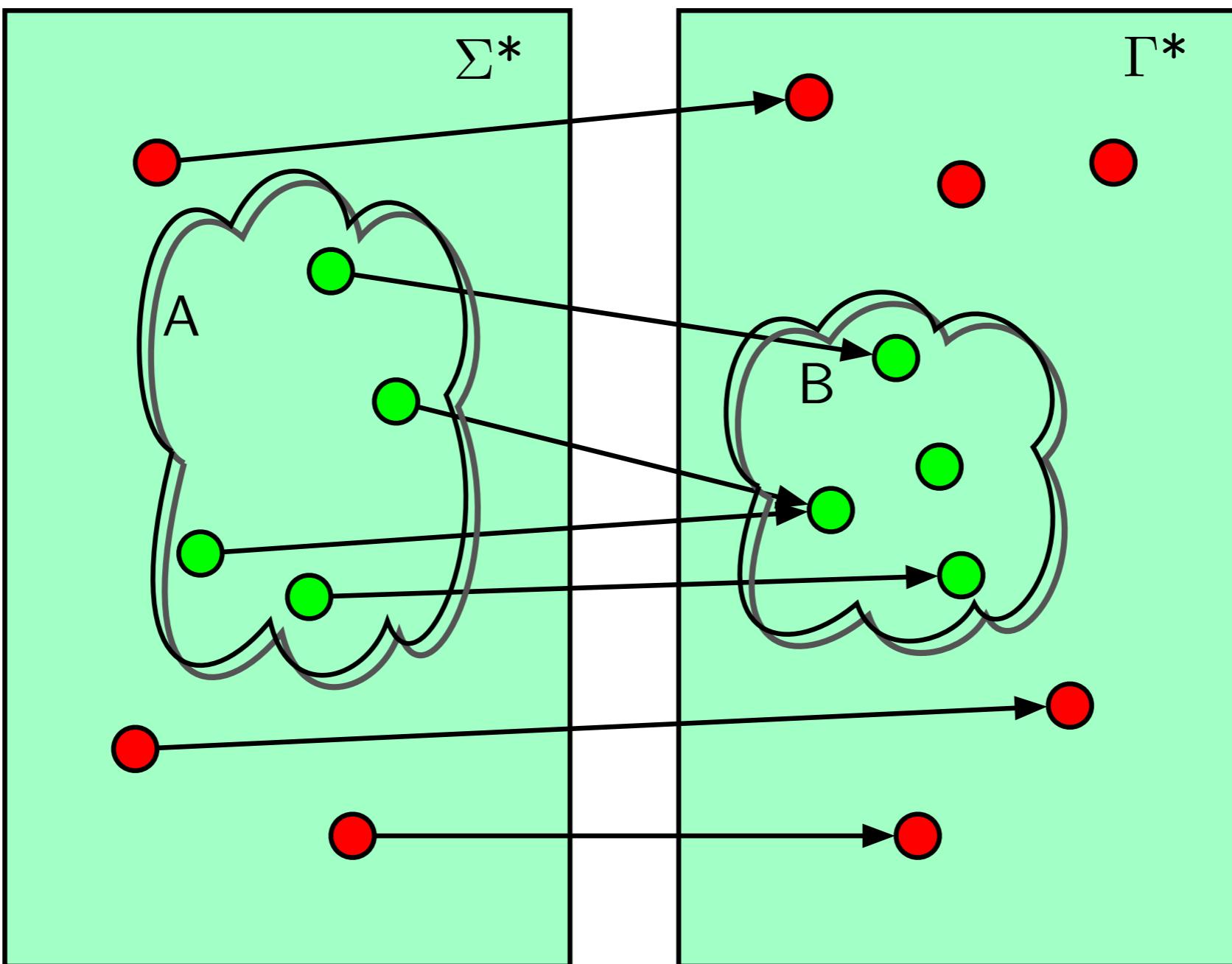
“I can't find an efficient algorithm, because no such algorithm is possible!”

Das sichert Ihre Position



“I can't find an efficient algorithm, but neither can all these famous people.”

berechenbare Reduktion f



Polynomzeit-Reduktion

Definition 24.1:

- Seien $A \subset \Sigma^*$ und $B \subset \Gamma^*$.
- Man sagt „ A ist **polynomial reduzierbar** auf B “ ($A \leq_{\text{pol}} B$), wenn es eine Polynomialzeit beschränkte DTM gibt, die eine totale, berechenbare Funktion

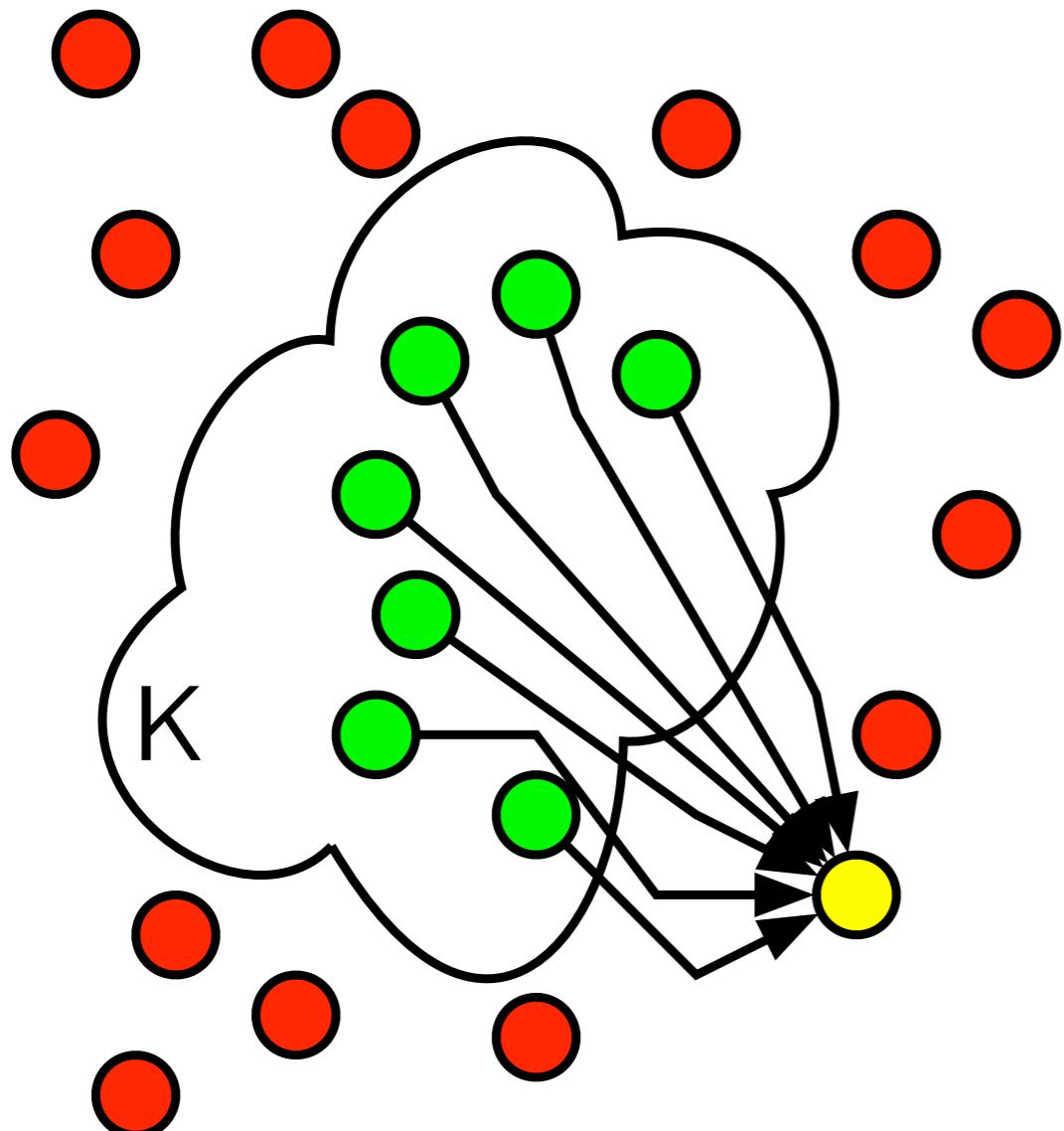
$$f : \Sigma^* \rightarrow \Gamma^*$$

berechnet, für die

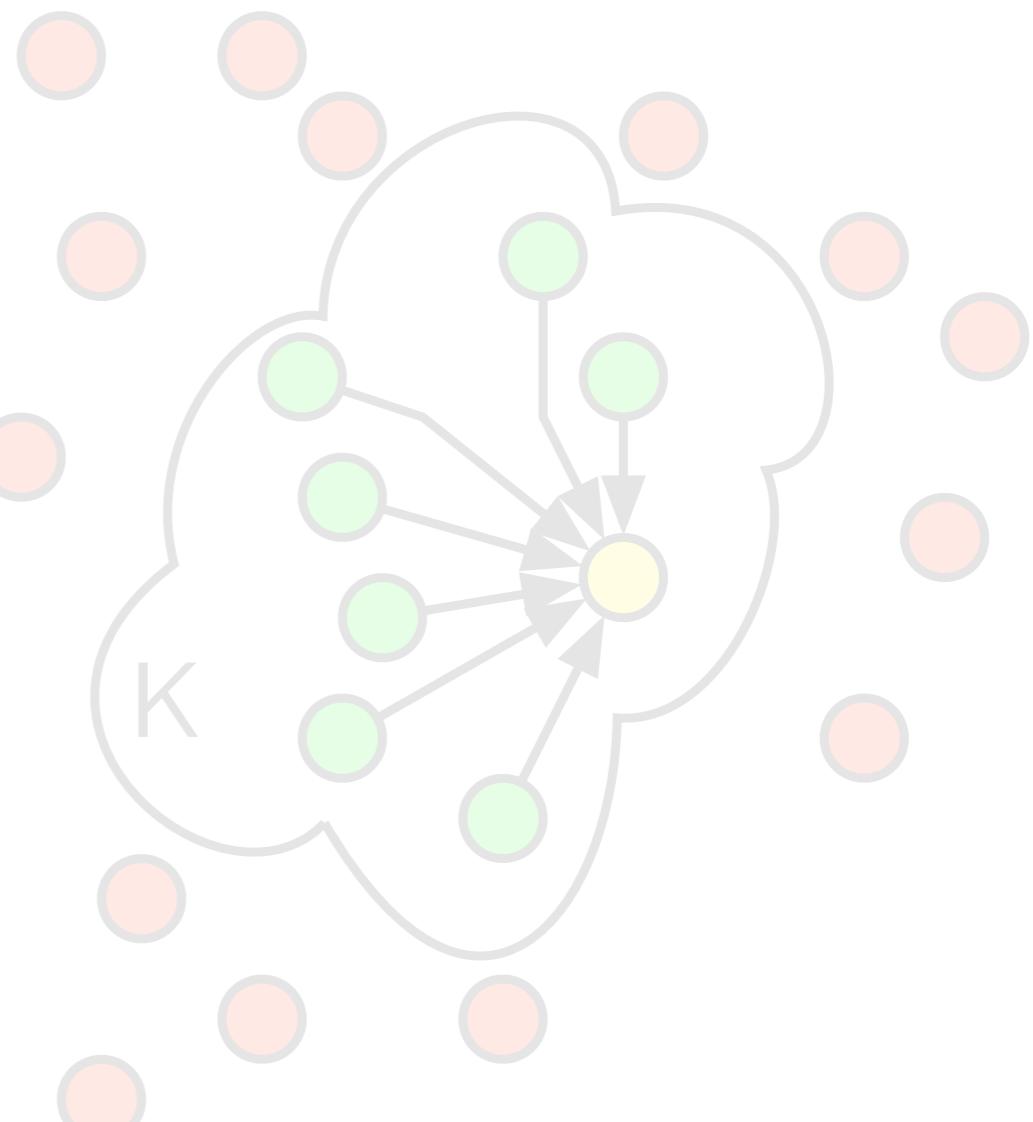
$$\forall x \in \Sigma^* : x \in A \iff f(x) \in B$$

gilt.

Schwerheit (*hardness*) / Vollständigkeit (*completeness*)

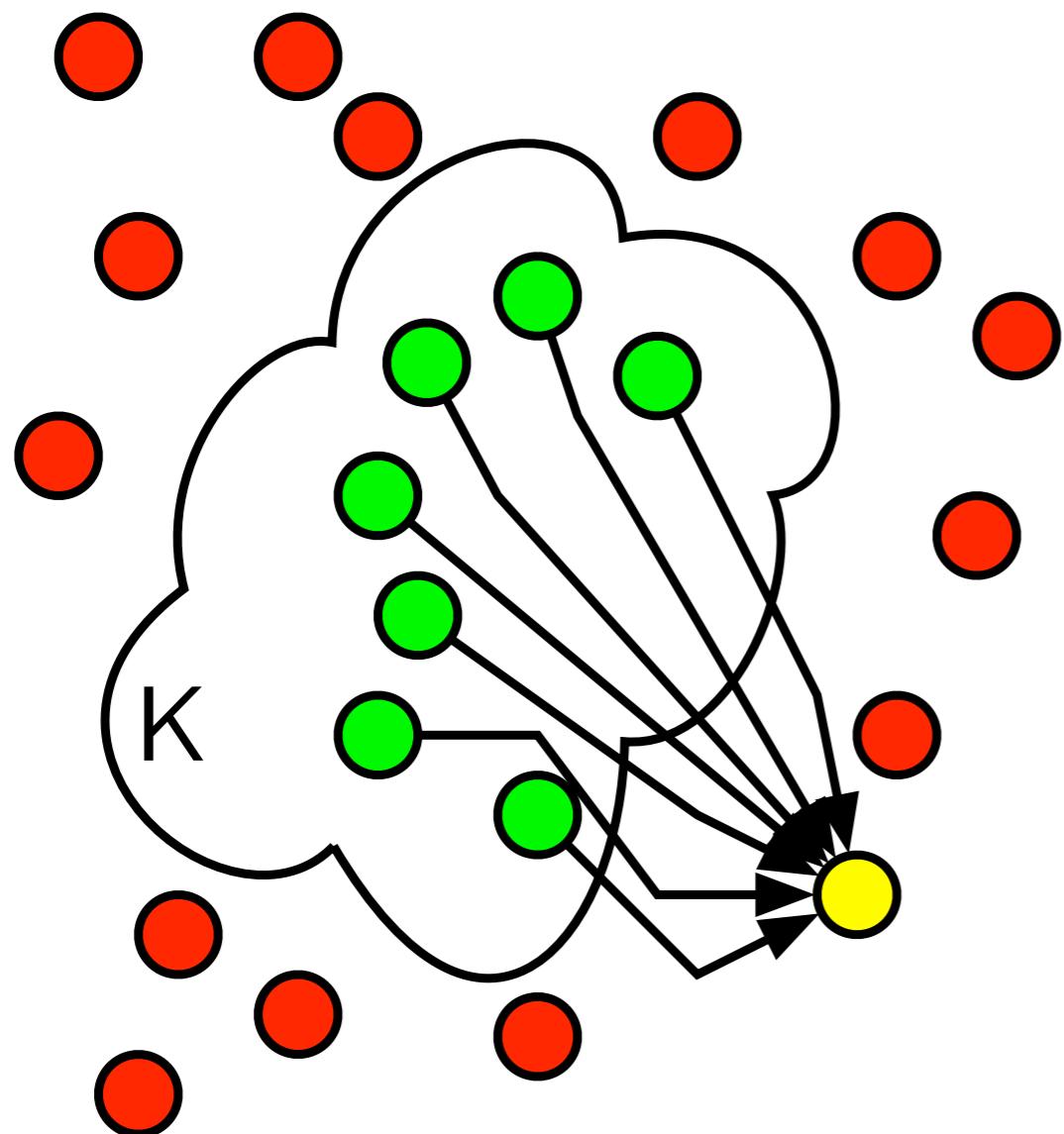


schwer

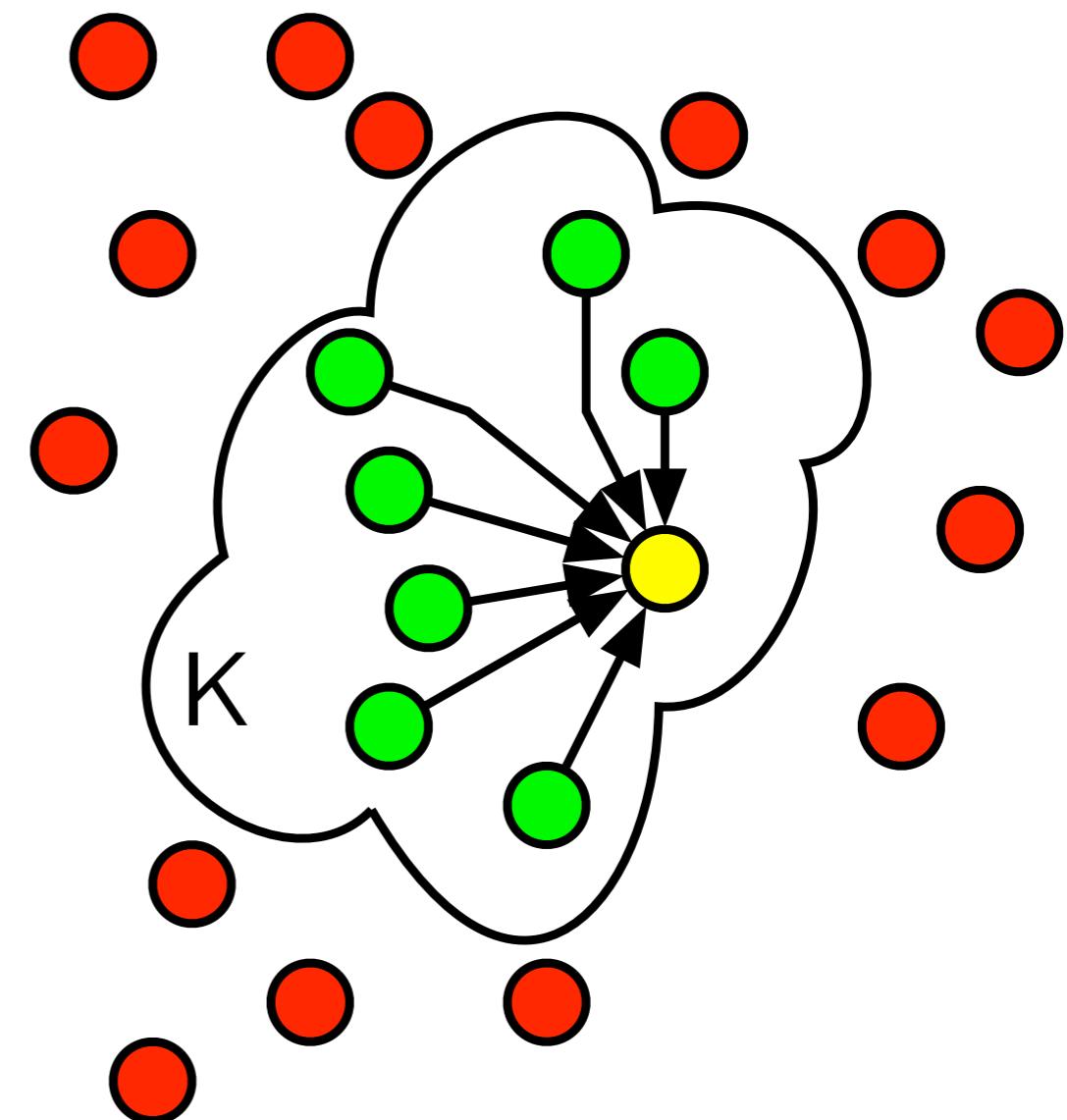


vollständig

Schwerheit (*hardness*) / Vollständigkeit (*completeness*)



schwer



vollständig

Vollständigkeit (*completeness*)

von speziellem Interesse:

Polynomialzeitreduktion (\leq_{pol}),

Ist von DTM in Polynomialzeit berechenbar.

Definition 24.2:

- Eine Menge $A \subseteq \Sigma^*$ heißt **hart** (oder besser: **schwer**) für eine Klasse \mathcal{K} gdw. $\forall B \in \mathcal{K} : B \leq_{\text{pol}} A$
- Eine Menge $A \subseteq \Sigma^*$ heißt **vollständig** für eine Klasse \mathcal{K} gdw. $A \in \mathcal{K} \wedge \forall B \in \mathcal{K} : B \leq_{\text{pol}} A$

NP-Vollständigkeit (NP-completeness)

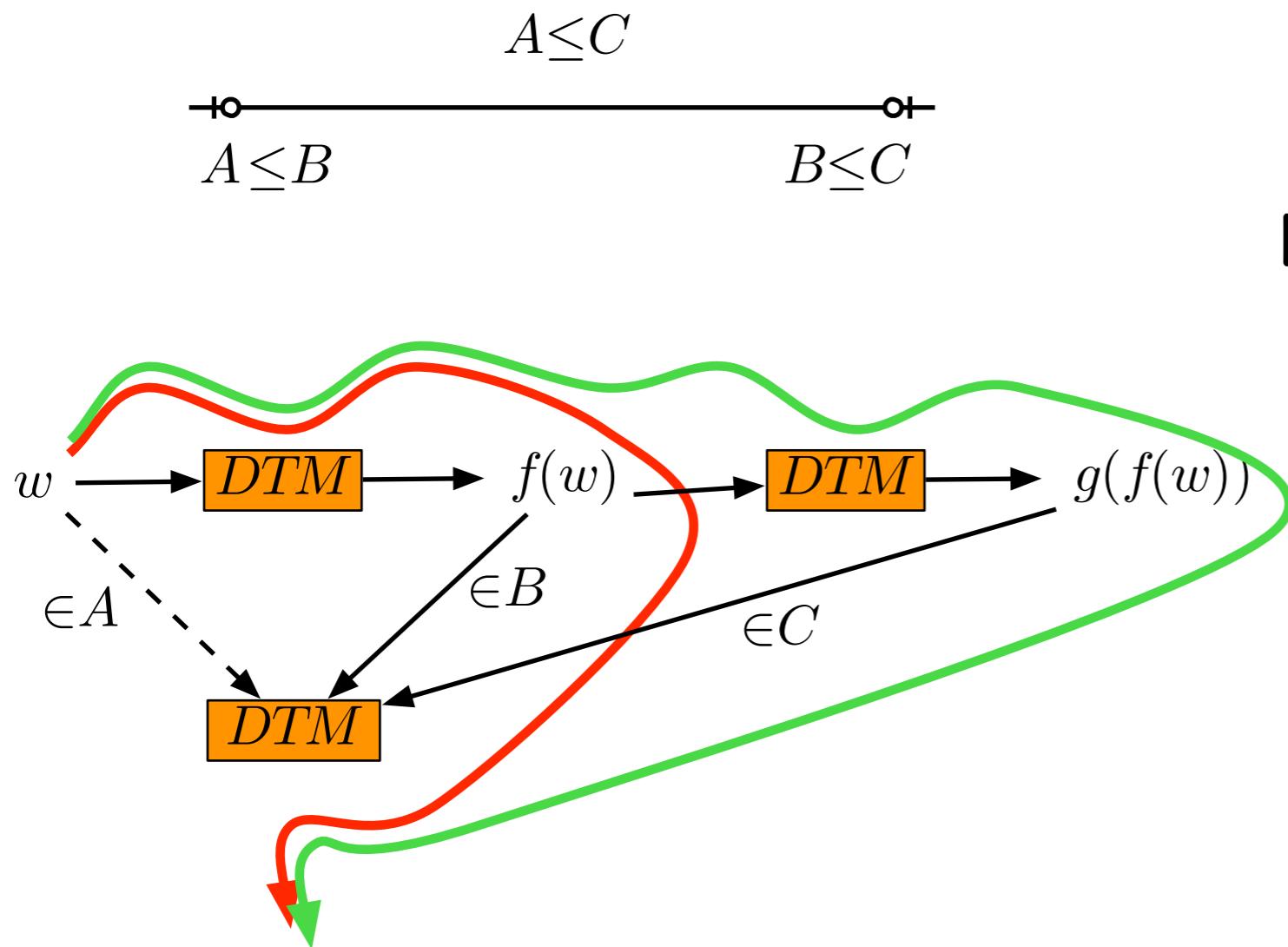
Definition 24.3: (NP-Vollständigkeit)

- besonders interessanter Spezialfall der Vollständigkeit:
- Eine Menge $A \subseteq \Sigma^*$ heißt **NP-vollständig** gdw.

$$A \in \mathcal{NP} \wedge \forall B \in \mathcal{NP} : B \leq_{\text{pol}} A$$

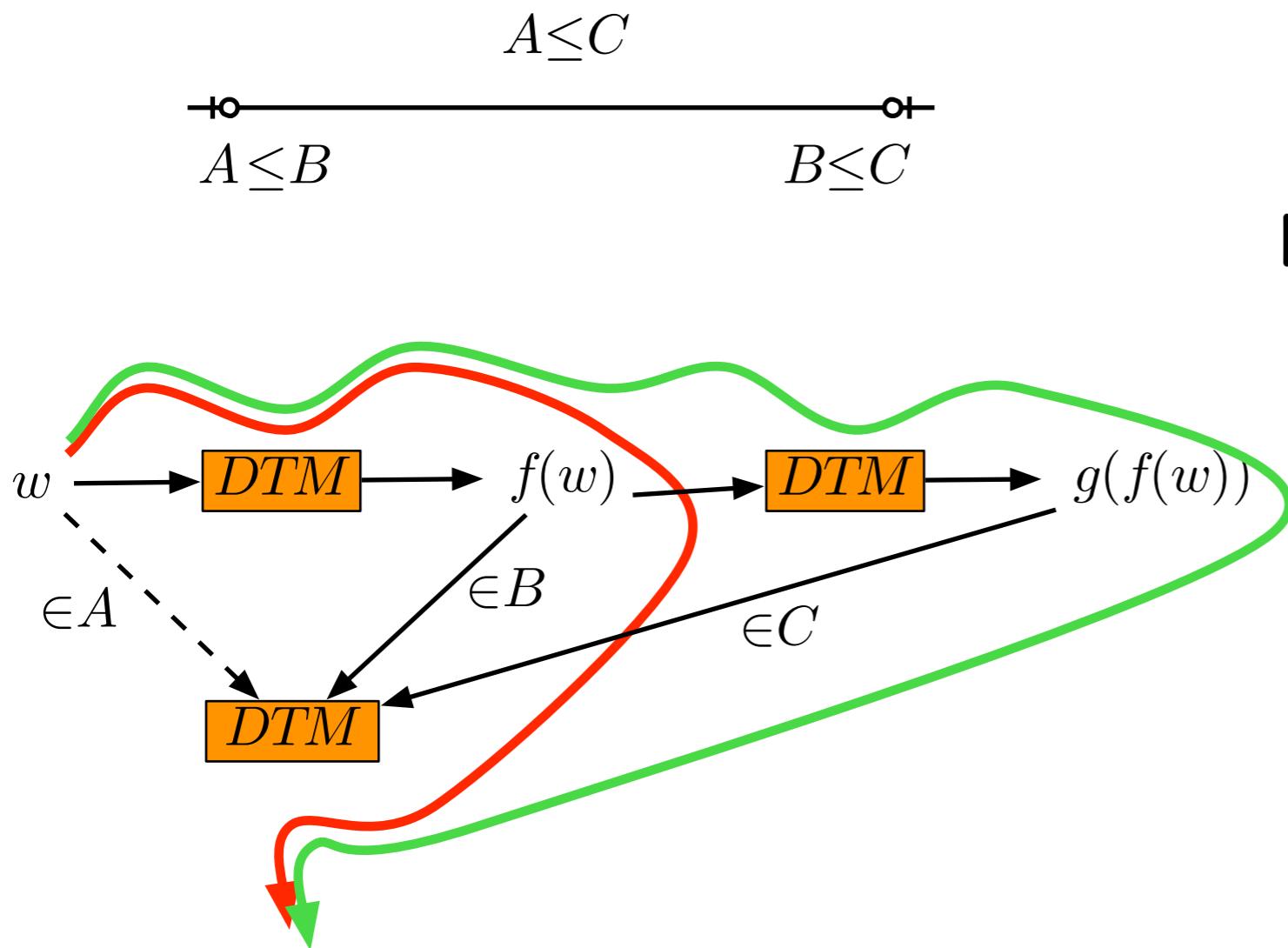
- Ein **NP-vollständiges** Problem ist somit eines der **schwersten** bzw. **umfassendsten** Probleme innerhalb der Klasse \mathcal{NP} .
- Wichtige Eigenschaft: **Transitivität von \leq_{pol}** .

Transitivität der Reduktion



Generelles Problem bei
Komplexitätsbeschränkung
für die Reduktion:
**Wie groß wird das
Zwischenergebnis?**

Transitivität der Reduktion



Generelles Problem bei
Komplexitätsbeschränkung
für die Reduktion:
**Wie groß wird das
Zwischenergebnis?**

Verkettung von Polynomialzeitreduktion:

- Laufzeit für $f(|w|)$ ist höchstens $p(|w|)$ für ein Polynom $p(x)$.
- Laufzeit für $g(|x|)$ ist höchstens $q(|w|)$ für ein Polynom $q(x)$.
- Platz auf Arbeitsbändern ist nicht begrenzt.
- Das Endergebnis $g(f(|w|))$ wird in Polynomzeit berechnet, denn $q(p(x))$ ist wiederum ein Polynom.

Wichtige Folgerung

Satz 24.4:

Sei L NP -vollständig, $L \leq_{\text{pol}} M$ und $M \in \mathcal{NP}$.
Dann ist M gleichfalls NP -vollständig.

Beweis:

Die Aussage folgt direkt aus der Definition von NP -Vollständigkeit und der Transitivität der Polynomzeitreduktionen.

- typisches Beispiel KNF ist NP -vollständig.
- $KNF := \{w \in X^* \mid w \text{ ist eine erfüllbare boolesche Formel in konjunktiver Normalform}\}$

wichtige Folgerung

Ein einziges NP-vollständiges Problem erlaubt es uns, die ganze Klasse NP zu verstehen:

Korollar 24.5:

Für eine NP -vollständige Menge L gilt:

$$L \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}.$$

Beweis:

- L ist NP -vollständig, also folgt $\forall M \in \mathcal{NP} : M \leq_{\text{pol}} L$. Wegen $L \in \mathcal{P}$ folgt nun sofort $M \in \mathcal{P}$.
- Weil L NP -vollständig ist, gilt $L \in \mathcal{NP}$. Aus $\mathcal{P} = \mathcal{NP}$ folgt sofort $L \in \mathcal{P}$.

NP-Vollständigkeit (Stephen A. Cook 1971)

- Bisher nur Definition der *NP*-Vollständigkeit
- Aber: Gibt es überhaupt solche Probleme?
 - Diese Frage war lange Zeit ungeklärt!
 - Nach dem ersten folgten aber sofort viele *NP*-vollständige Probleme.
 - Warum?!
- Idee für ein erstes *NP*-vollständiges Problem:
 - Codierung aller Polynomialzeit-TM-Rechnungen als boolesche Formel (SAT)
 - als Kachelproblem (2D-Domino) und viele andere ...

NP-Vollständigkeit (Stephen A. Cook 1971)

- Bisher nur Definition der *NP*-Vollständigkeit
- Aber: Gibt es überhaupt solche Probleme?
 - Diese Frage war lange Zeit ungeklärt!
 - Nach dem ersten folgten aber sofort viele *NP*-vollständige Probleme.
 - Warum?!
- Idee für ein erstes *NP*-vollständiges Problem:
 - Codierung aller Polynomialzeit-TM-Rechnungen als boolesche Formel (SAT)
 - als Kachelproblem (2D-Domino) und viele andere ...

Erfüllbarkeitsproblem (SAT)

Definition 24.4:

$SAT := \{\langle w \rangle \in \{0, 1\}^* \mid w \in X^* \text{ ist ein erfüllbarer boolescher Ausdruck}\}$ ist das **Erfüllbarkeitsproblem**:

Gegeben: Eine Menge V von Variablen und eine boolesche (aussagenlogische) Formel $B \in X^*$ mit $X := V \cup \{0, 1, \vee, \wedge, \neg, (,)\}$. $\langle \rangle$ ist übliche Kodierung.

Gesucht: Gibt es eine Belegung der Variablen mit TRUE (1) und FALSE (0), so dass B zu TRUE (1) evaluiert?

Antwort: JA oder NEIN

NP-Vollständigkeit von SAT

Satz 24.3:

SAT ist *NP*-vollständig.

Beweisidee:

- Es ist einfach zu sehen, dass $SAT \in \mathcal{NP}$ gilt.
 - Zu einer gegebenen Formel B mit den Variablen x_1, x_2, \dots, x_n wird in Linearzeit nichtdeterministisch eine Belegung der Variablen mit TRUE oder FALSE geraten.
 - Es wird geprüft, ob B mit dieser Belegung den Wert TRUE bekommt. Dies ist in Polynomzeit möglich.
- Bleibt zu Zeigen, dass jedes andere Problem aus \mathcal{NP} in Polynomialzeit auf SAT reduzierbar ist.

die Cook'sche Konstruktion

Zu jeder NTM wird boolesche Formel gebildet, die ihre Rechnungen kodiert! Arbeitet die NTM in polynomieller Zeit, wird die Formel nicht zu groß und ist erfüllbar genau dann, wenn die NTM eine Erfolgsrechnung besitzt!

Da die Formel aus jeder NTM in polynomieller Zeit erstellt werden kann, ist die Konstruktion eine Reduktion JEDER Sprache aus \mathcal{NP} auf SAT .

Wenn SAT schnell gelöst werden kann, so auch jedes andere Problem aus \mathcal{NP} !

die Cook'sche Konstruktion

Zu jeder NTM wird boolesche Formel gebildet, die ihre Rechnungen kodiert! Arbeitet die NTM in polynomieller Zeit, wird die Formel nicht zu groß und ist erfüllbar genau dann, wenn die NTM eine Erfolgsrechnung besitzt!

Da die Formel aus jeder NTM in polynomieller Zeit erstellt werden kann, ist die Konstruktion eine Reduktion JEDER Sprache aus \mathcal{NP} auf SAT .

Wenn SAT schnell gelöst werden kann, so auch jedes andere Problem aus \mathcal{NP} !

die Cook'sche Konstruktion

Zu jeder NTM wird boolesche Formel gebildet, die ihre Rechnungen kodiert! Arbeitet die NTM in polynomieller Zeit, wird die Formel nicht zu groß und ist erfüllbar genau dann, wenn die NTM eine Erfolgsrechnung besitzt!

Da die Formel aus jeder NTM in polynomieller Zeit erstellt werden kann, ist die Konstruktion eine Reduktion JEDER Sprache aus \mathcal{NP} auf SAT .

Wenn SAT schnell gelöst werden kann, so auch jedes andere Problem aus \mathcal{NP} !

die Cook'sche Konstruktion

Zu jeder NTM wird boolesche Formel gebildet, die ihre Rechnungen kodiert! Arbeitet die NTM in polynomieller Zeit, wird die Formel nicht zu groß und ist erfüllbar genau dann, wenn die NTM eine Erfolgsrechnung besitzt!

Da die Formel aus jeder NTM in polynomieller Zeit erstellt werden kann, ist die Konstruktion eine Reduktion JEDER Sprache aus \mathcal{NP} auf SAT .

Wenn SAT schnell gelöst werden kann, so auch jedes andere Problem aus \mathcal{NP} !

Die verwendeten atomaren Aussagen

- $\text{FELD}(i, j, t) \hat{=} \text{In Konfiguration } k_t \text{ steht das Zeichen } x_j \text{ in Feld } i.$
- $\text{ZUSTAND}(r, t) \hat{=} \text{In der Konfiguration } k_t \text{ befindet sich die TM } A_L \text{ im Zustand } z_r.$
- $\text{KOPF}(i, t) \hat{=} \text{In der Konfiguration } k_t \text{ steht der LSK auf dem Feld } i.$
- Anzahl dieser Variablen in $O(p(n)^2)$, wenn $p(n)$ die Zeitschranke ist.

Die verwendeten atomaren Aussagen

- $\text{FELD}(i, j, t) \hat{=} \text{In Konfiguration } k_t \text{ steht das Zeichen } x_j \text{ in Feld } i.$
- $\text{ZUSTAND}(r, t) \hat{=} \text{In der Konfiguration } k_t \text{ befindet sich die TM } A_L \text{ im Zustand } z_r.$
- $\text{KOPF}(i, t) \hat{=} \text{In der Konfiguration } k_t \text{ steht der LSK auf dem Feld } i.$
- Anzahl dieser Variablen in $O(p(n)^2)$, wenn $p(n)$ die Zeitschranke ist.

ein Blick in den Beweis gibt ein wenig Einblick

$$F_L := A \wedge B \wedge C \wedge D \wedge E \wedge F \wedge G$$

mit $A = A_1 \wedge \dots \wedge A_{p(n)}$ für die Laufzeit $p(n)$

Die einzelnen Teilformeln bedeuten:

$A_t \hat{=} \text{In der Konfiguration } k_t \text{ steht der LSK von } A_L \text{ auf genau einem Feld.}$

$B \hat{=} \text{In jeder Konfiguration } k_t \text{ enthält jedes Feld genau ein Zeichen.}$

$C \hat{=} \text{In jeder Konfiguration } k_t \text{ befindet sich } A_L \text{ in genau einem Zustand.}$

$D \hat{=} \text{Bei jedem Übergang wird genau das Feld verändert, auf das der LSK zeigt.}$

$E \hat{=} \text{Jeder Übergang im Zustand } z_k \text{ mit } x_j \text{ unter dem LSK entspricht der Turing-Tafel.}$

$F \hat{=} \text{Die erste Konfiguration ist } k_0 = z_1 w \# \dots \#.$

$G \hat{=} \text{Der Zustand in der letzten Konfiguration } k_{p(n)} \text{ ist Endzustand aus } Z_{\text{end}}.$

Und, wichtig: Die Formel F_L kann in Polynomzeit konstruiert werden.

ein Einblick in den Beweis...

- Es ist $A := A_1 \wedge A_2 \wedge \dots \wedge A_{p(n)}$ und
 $\forall t \leq p(n) : A_t := \oplus(\text{KOPF}(1, t), \text{KOPF}(2, t), \dots, \text{KOPF}(p(n), t))$

exklusives oder

- Auch B und C sind zusammengesetzte Formeln:

$$B := \bigwedge_{1 \leq i, t \leq p(n)} B(i, t) \text{ mit}$$

$$B(i, t) := \oplus(\text{FELD}(i, 1, t), \dots, \text{FELD}(i, m, t)), \quad m := |Y|$$

$$C := \bigwedge_{1 \leq t \leq p(n)} C_t \text{ mit}$$

$$C_t := \oplus(\text{ZUSTAND}(1, t), \dots, \text{ZUSTAND}(s, t)), \quad s := |Z|$$

ein Einblick in den Beweis...

- Es ist $A := A_1 \wedge A_2 \wedge \dots \wedge A_{p(n)}$ und
 $\forall t \leq p(n) : A_t := \oplus(\text{KOPF}(1, t), \text{KOPF}(2, t), \dots, \text{KOPF}(p(n), t))$
 
 exklusives oder
- Auch B und C sind zusammengesetzte Formeln:

$$B := \bigwedge_{1 \leq i, t \leq p(n)} B(i, t) \text{ mit}$$

$$B(i, t) := \oplus(\text{FELD}(i, 1, t), \dots, \text{FELD}(i, m, t)), \quad m := |Y|$$

$$C := \bigwedge_{1 \leq t \leq p(n)} C_t \text{ mit}$$

$$C_t := \oplus(\text{ZUSTAND}(1, t), \dots, \text{ZUSTAND}(s, t)), \quad s := |Z|$$

ein Einblick in den Beweis...

- Es ist $A := A_1 \wedge A_2 \wedge \dots \wedge A_{p(n)}$ und
 $\forall t \leq p(n) : A_t := \oplus(\text{KOPF}(1, t), \text{KOPF}(2, t), \dots, \text{KOPF}(p(n), t))$
 
 exklusives oder
- Auch B und C sind zusammengesetzte Formeln:

$$B := \bigwedge_{1 \leq i, t \leq p(n)} B(i, t) \text{ mit}$$

$$B(i, t) := \oplus(\text{FELD}(i, 1, t), \dots, \text{FELD}(i, m, t)), \quad m := |Y|$$

$$C := \bigwedge_{1 \leq t \leq p(n)} C_t \text{ mit}$$

$$C_t := \oplus(\text{ZUSTAND}(1, t), \dots, \text{ZUSTAND}(s, t)), \quad s := |Z|$$

ein Einblick in den Beweis...

- Es ist $A := A_1 \wedge A_2 \wedge \dots \wedge A_{p(n)}$ und
 $\forall t \leq p(n) : A_t := \oplus(\text{KOPF}(1, t), \text{KOPF}(2, t), \dots, \text{KOPF}(p(n), t))$
 
 exklusives oder
- Auch B und C sind zusammengesetzte Formeln:

$$B := \bigwedge_{1 \leq i, t \leq p(n)} B(i, t) \text{ mit}$$

$$B(i, t) := \oplus(\text{FELD}(i, 1, t), \dots, \text{FELD}(i, m, t)), \quad m := |Y|$$

$$C := \bigwedge_{1 \leq t \leq p(n)} C_t \text{ mit}$$

$$C_t := \oplus(\text{ZUSTAND}(1, t), \dots, \text{ZUSTAND}(s, t)), \\ s := |Z|$$

ein Einblick in den Beweis...

- Es ist $A := A_1 \wedge A_2 \wedge \dots \wedge A_{p(n)}$ und
 $\forall t \leq p(n) : A_t := \oplus(\text{KOPF}(1, t), \text{KOPF}(2, t), \dots, \text{KOPF}(p(n), t))$
 
 exklusives oder
- Auch B und C sind zusammengesetzte Formeln:

$$B := \bigwedge_{1 \leq i, t \leq p(n)} B(i, t) \text{ mit}$$

$$B(i, t) := \oplus(\text{FELD}(i, 1, t), \dots, \text{FELD}(i, m, t)), \quad m := |Y|$$

$$C := \bigwedge_{1 \leq t \leq p(n)} C_t \text{ mit}$$

$$C_t := \oplus(\text{ZUSTAND}(1, t), \dots, \text{ZUSTAND}(s, t)), \\ s := |Z|$$

Zuletzt steht eine Formel, die genau dann erfüllbar ist, wenn die NTM eine Erfolgsrechnung mit $p(n)$ begrenzter Schrittzahl hat.

Konjunktive Normalform (KNF)

Wir definieren eine Einschränkung von SAT:

Definition 24.5:

Das Erfüllbarkeitsproblem boolescher Formeln in konjunktiver Normalform dargestellt als Sprache:

$KNF := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist eine erfüllbare boolesche Formel in konjunktiver Normalform }\}$

Konjunktive Normalform (KNF)

Wir definieren eine Einschränkung von SAT:

Definition 24.5:

Das Erfüllbarkeitsproblem boolescher Formeln in konjunktiver Normalform dargestellt als Sprache:

$KNF := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist eine erfüllbare boolesche Formel in konjunktiver Normalform }\}$

Nun kann KNF höchstens so schwer sein wie SAT.
Ist es leichter zu lösen?

Konjunktive Normalform (KNF)

Wir definieren eine Einschränkung von SAT:

Definition 24.5:

Das Erfüllbarkeitsproblem boolescher Formeln in konjunktiver Normalform dargestellt als Sprache:

$KNF := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist eine erfüllbare boolesche Formel in konjunktiver Normalform }\}$

Nun kann KNF höchstens so schwer sein wie SAT.
Ist es leichter zu lösen?

Leider
nicht!

Konjunktive Normalform (KNF)

Wir definieren eine Einschränkung von SAT:

Definition 24.5:

Das Erfüllbarkeitsproblem boolescher Formeln in konjunktiver Normalform dargestellt als Sprache:

$KNF := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist eine erfüllbare boolesche Formel in konjunktiver Normalform }\}$

Nun kann KNF höchstens so schwer sein wie SAT.
Ist es leichter zu lösen?

Leider
nicht!

Satz 24.1:

KNF ist \mathcal{NP} -vollständig.

Konjunktive Normalform (KNF)

Beweisskizze. KNF ist in NP, das SAT schon in NP ist.

NP-Vollständigkeit: Reduktion von SAT auf KNF.

Konjunktive Normalform (KNF)

Beweisskizze. KNF ist in NP, das SAT schon in NP ist.

NP-Vollständigkeit: Reduktion von SAT auf KNF.

(1. Versuch). Sei F eine Formel.

Konstruiere zu F eine äquivalente KNF H .

Konjunktive Normalform (KNF)

Beweisskizze. KNF ist in NP, das SAT schon in NP ist.

NP-Vollständigkeit: Reduktion von SAT auf KNF.

(1. Versuch). Sei F eine Formel.

Konstruiere zu F eine äquivalente KNF H .

• Forme F so zu G um, das in G die Negationen nur noch vor den Atomen vorkommen:

$$\neg(X \wedge Y) \equiv \neg X \vee \neg Y, \quad \neg(X \vee Y) \equiv \neg X \wedge \neg Y \quad \text{und} \quad \neg\neg X \equiv X$$

• Forme G in eine KNF H um:

$$X \vee (Y \wedge Z) \equiv (X \vee Y) \wedge (X \vee Z) \quad \text{und} \quad (X \wedge Y) \vee Z \equiv (X \vee Z) \wedge (Y \vee Z)$$

Konjunktive Normalform (KNF)

Beweisskizze. KNF ist in NP, das SAT schon in NP ist.

NP-Vollständigkeit: Reduktion von SAT auf KNF.

(1. Versuch). Sei F eine Formel.

Konstruiere zu F eine äquivalente KNF H .

• Forme F so zu G um, das in G die Negationen nur noch vor den Atomen vorkommen:

$$\neg(X \wedge Y) \equiv \neg X \vee \neg Y, \quad \neg(X \vee Y) \equiv \neg X \wedge \neg Y \quad \text{und} \quad \neg\neg X \equiv X$$

• Forme G in eine KNF H um:

$$X \vee (Y \wedge Z) \equiv (X \vee Y) \wedge (X \vee Z) \quad \text{und} \quad (X \wedge Y) \vee Z \equiv (X \vee Z) \wedge (Y \vee Z)$$

Leider klappt das so nicht!

Konjunktive Normalform (KNF)

Beweisskizze. KNF ist in NP, das SAT schon in NP ist.

NP-Vollständigkeit: Reduktion von SAT auf KNF.

(1. Versuch). Sei F eine Formel.

Konstruiere zu F eine äquivalente KNF H .

• Forme F so zu G um, das in G die Negationen nur noch vor den Atomen vorkommen:

$$\neg(X \wedge Y) \equiv \neg X \vee \neg Y, \quad \neg(X \vee Y) \equiv \neg X \wedge \neg Y \quad \text{und} \quad \neg\neg X \equiv X$$

• Forme G in eine KNF H um:

$$X \vee (Y \wedge Z) \equiv (X \vee Y) \wedge (X \vee Z) \quad \text{und} \quad (X \wedge Y) \vee Z \equiv (X \vee Z) \wedge (Y \vee Z)$$

Leider klappt das so nicht!

Warum nicht?

Konjunktive Normalform (KNF)

Beweisskizze. KNF ist in NP, das SAT schon in NP ist.

NP-Vollständigkeit: Reduktion von SAT auf KNF.

(1. Versuch). Sei F eine Formel.

Konstruiere zu F eine äquivalente KNF H .

• Forme F so zu G um, das in G die Negationen nur noch vor den Atomen vorkommen:

$$\neg(X \wedge Y) \equiv \neg X \vee \neg Y, \quad \neg(X \vee Y) \equiv \neg X \wedge \neg Y \quad \text{und} \quad \neg\neg X \equiv X$$

• Forme G in eine KNF H um:

$$X \vee (Y \wedge Z) \equiv (X \vee Y) \wedge (X \vee Z) \quad \text{und} \quad (X \wedge Y) \vee Z \equiv (X \vee Z) \wedge (Y \vee Z)$$

Leider klappt das so nicht!

Warum nicht? Die Umformung von G nach H geschieht i.a. nicht in Polynomzeit.

Konjunktive Normalform (KNF)

Beweisskizze. KNF ist in NP, das SAT schon in NP ist.

NP-Vollständigkeit: Reduktion von SAT auf KNF.

(1. Versuch). Sei F eine Formel.

Konstruiere zu F eine äquivalente KNF H .

• Forme F so zu G um, das in G die Negationen nur noch vor den Atomen vorkommen:

$$\neg(X \wedge Y) \equiv \neg X \vee \neg Y, \quad \neg(X \vee Y) \equiv \neg X \wedge \neg Y \quad \text{und} \quad \neg\neg X \equiv X$$

• Forme G in eine KNF H um:

$$X \vee (Y \wedge Z) \equiv (X \vee Y) \wedge (X \vee Z) \quad \text{und} \quad (X \wedge Y) \vee Z \equiv (X \vee Z) \wedge (Y \vee Z)$$

Leider klappt das so nicht!

Warum nicht? Die Umformung von G nach H geschieht i.a. nicht in Polynomzeit.

Bsp.:

$$(A_1 \wedge B_1) \vee \dots \vee (A_n \wedge B_n) \equiv (A_1 \vee A_2 \vee \dots \vee A_n) \wedge \dots \wedge (B_1 \vee B_2 \vee \dots \vee B_n)$$

Also: 2^n Klauseln.

Diese können in Polynomzeit nicht einmal hingeschrieben werden.

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

- Forme F -wie zuvor- zu äquivalenter Formel G um.

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

- Forme F -wie zuvor- zu äquivalenter Formel G um.
- Konstruiere die KNF $H=H(G)$ rekursiv aus G :

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

- ➊ Forme F -wie zuvor- zu äquivalenter Formel G um.
- ➋ Konstruiere die KNF $H=H(G)$ rekursiv aus G :
 1. $G = A$: Setze $H(G) = A$

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

- Forme F -wie zuvor- zu äquivalenter Formel G um.
- Konstruiere die KNF $H=H(G)$ rekursiv aus G :
 1. $G = A$: Setze $H(G) = A$
 2. $G = (G_1 \wedge G_2)$: Setze $H(G) = H(G_1) \wedge H(G_2)$.

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

• Forme F -wie zuvor- zu äquivalenter Formel G um.

• Konstruiere die KNF $H=H(G)$ rekursiv aus G :

1. $G = A$: Setze $H(G) = A$

2. $G = (G_1 \wedge G_2)$: Setze $H(G) = H(G_1) \wedge H(G_2)$.

3. $G = (G_1 \vee G_2)$:

Sei $H(G_1) = (K_1 \wedge \dots \wedge K_m)$ und $H(G_2) = (L_1 \wedge \dots \wedge L_n)$.

Wähle ein frisches Atom A und setze:

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

Forme F -wie zuvor- zu äquivalenter Formel G um.

Konstruiere die KNF $H=H(G)$ rekursiv aus G :

1. $G = A$: Setze $H(G) = A$

2. $G = (G_1 \wedge G_2)$: Setze $H(G) = H(G_1) \wedge H(G_2)$.

3. $G = (G_1 \vee G_2)$:

Sei $H(G_1) = (K_1 \wedge \dots \wedge K_m)$ und $H(G_2) = (L_1 \wedge \dots \wedge L_n)$.

Wähle ein frisches Atom A und setze:

$H(G) = (A \vee K_1) \wedge \dots \wedge (A \vee K_m) \wedge (\neg A \vee L_1) \wedge \dots \wedge (\neg A \vee L_n)$

Konjunktive Normalform (KNF)

Beweisskizze. Ziel: Reduktion von SAT auf KNF.

(2. Versuch). Sei F eine Formel.

Konstruiere zu F eine KNF H mit:

F erfüllbar gdw. H erfüllbar

Forme F -wie zuvor- zu äquivalenter Formel G um.

Konstruiere die KNF $H=H(G)$ rekursiv aus G :

1. $G = A$: Setze $H(G) = A$

2. $G = (G_1 \wedge G_2)$: Setze $H(G) = H(G_1) \wedge H(G_2)$.

3. $G = (G_1 \vee G_2)$:

Sei $H(G_1) = (K_1 \wedge \dots \wedge K_m)$ und $H(G_2) = (L_1 \wedge \dots \wedge L_n)$.

Wähle ein frisches Atom A und setze:

$H(G) = (A \vee K_1) \wedge \dots \wedge (A \vee K_m) \wedge (\neg A \vee L_1) \wedge \dots \wedge (\neg A \vee L_n)$

Es gilt: $H(G)$ ist in Polynomzeit konstruierbar und ist genau dann erfüllbar, wenn F dies ist. qed.

Erfüllbarkeit von 3-SAT

Wir definieren noch eine Einschränkung von SAT:

Definition 24.6:

Die Sprache $3\text{-SAT} \subsetneq KNF \subsetneq SAT$ ist gegeben durch:

$3\text{-SAT} := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist erfüllbare boolesche Formel in konjunktiver Normalform mit genau 3 Literalen pro Klausel}\}$.

Erfüllbarkeit von 3-SAT

Wir definieren noch eine Einschränkung von SAT:

Definition 24.6:

Die Sprache $3\text{-SAT} \subsetneq \text{KNF} \subsetneq \text{SAT}$ ist gegeben durch:

$3\text{-SAT} := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist erfüllbare boolesche Formel in konjunktiver Normalform mit genau 3 Literalen pro Klausel}\}.$

Nun kann 3-SAT höchstens so schwer sein wie KNF oder SAT.
Ist es leichter zu lösen?

Erfüllbarkeit von 3-SAT

Wir definieren noch eine Einschränkung von SAT:

Definition 24.6:

Die Sprache $3\text{-SAT} \subsetneq \text{KNF} \subsetneq \text{SAT}$ ist gegeben durch:

$3\text{-SAT} := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist erfüllbare boolesche Formel in konjunktiver Normalform mit genau 3 Literalen pro Klausel}\}.$

Nun kann 3-SAT höchstens so schwer sein wie KNF oder SAT.
Ist es leichter zu lösen?

Wieder
nicht!

Erfüllbarkeit von 3-SAT

Wir definieren noch eine Einschränkung von SAT:

Definition 24.6:

Die Sprache $3\text{-SAT} \subsetneq \text{KNF} \subsetneq \text{SAT}$ ist gegeben durch:

$3\text{-SAT} := \{\langle w \rangle \in \{0, 1\}^* \mid w \text{ ist erfüllbare boolesche Formel in konjunktiver Normalform mit genau 3 Literalen pro Klausel}\}.$

Nun kann 3-SAT höchstens so schwer sein wie KNF oder SAT.
Ist es leichter zu lösen?

Wieder
nicht!

Satz 24.2:

3-SAT ist NP -vollständig.

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, da SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, da SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, da SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, da SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

⌚ $K_i = A$: Wähle zwei neue Atome B und C .

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, da SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

• $K_i = (A \vee B)$: Wähle ein neues Atom C .

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

• $K_i = (A \vee B)$: Wähle ein neues Atom C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C).$$

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

• $K_i = (A \vee B)$: Wähle ein neues Atom C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C).$$

• $K_i = (A \vee B \vee C)$: Setze $L_i := K_i$.

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

• $K_i = (A \vee B)$: Wähle ein neues Atom C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C).$$

• $K_i = (A \vee B \vee C)$: Setze $L_i := K_i$.

• $K_i = (A_1 \vee \dots \vee A_n)$ mit $n > 3$:

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

• $K_i = (A \vee B)$: Wähle ein neues Atom C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C).$$

• $K_i = (A \vee B \vee C)$: Setze $L_i := K_i$.

• $K_i = (A_1 \vee \dots \vee A_n)$ mit $n > 3$:

$$L_i := (A_1 \vee A_2 \vee B_1) \wedge (A_3 \vee \neg B_1 \vee B_2) \wedge (A_4 \vee \neg B_2 \vee B_3) \wedge \dots \wedge (A_{n-1} \vee A_n \vee B_{n-3})$$

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

• $K_i = (A \vee B)$: Wähle ein neues Atom C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C).$$

• $K_i = (A \vee B \vee C)$: Setze $L_i := K_i$.

• $K_i = (A_1 \vee \dots \vee A_n)$ mit $n > 3$:

$$L_i := (A_1 \vee A_2 \vee B_1) \wedge (A_3 \vee \neg B_1 \vee B_2) \wedge (A_4 \vee \neg B_2 \vee B_3) \wedge \dots \wedge (A_{n-1} \vee A_n \vee B_{n-3})$$

für neue Atome B_1, \dots, B_{n-3} .

Erfüllbarkeit von 3-SAT

Beweisskizze. 3-SAT ist in NP, das SAT schon in NP ist.

Ziel: Reduktion von KNF auf 3-SAT.

Sei F eine Formel in KNF.

Konstruiere zu F eine äquivalente Formel in 3-KNF.

Sei $F = (K_1 \wedge \dots \wedge K_m)$ die KNF.

Fallunterscheidung in Bezug auf die Anzahl der Literale in K_i :

• $K_i = A$: Wähle zwei neue Atome B und C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C).$$

• $K_i = (A \vee B)$: Wähle ein neues Atom C .

$$L_i := (A \vee B \vee C) \wedge (A \vee B \vee \neg C).$$

• $K_i = (A \vee B \vee C)$: Setze $L_i := K_i$.

• $K_i = (A_1 \vee \dots \vee A_n)$ mit $n > 3$:

$$L_i := (A_1 \vee A_2 \vee B_1) \wedge (A_3 \vee \neg B_1 \vee B_2) \wedge (A_4 \vee \neg B_2 \vee B_3) \wedge \dots \wedge (A_{n-1} \vee A_n \vee B_{n-3})$$

für neue Atome B_1, \dots, B_{n-3} .

Dann ist $G = (L_1 \wedge \dots \wedge L_m)$ äquivalent zu F und in 3-SAT. qed.

Das Hamilton-Kreis-Problem

Satz. HC ist NP -vollständig.

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Wir zeigen NP-Vollständigkeit per Reduktion von 3-SAT auf HC.

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Wir zeigen NP-Vollständigkeit per Reduktion von 3-SAT auf HC.

Wir konstruieren dazu zu jeder Formel F einen Graphen G_F :

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Wir zeigen NP-Vollständigkeit per Reduktion von 3-SAT auf HC.

Wir konstruieren dazu zu jeder Formel F einen Graphen G_F :

F erfüllbar gdw. G_F besitzt einen HC

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Wir zeigen NP-Vollständigkeit per Reduktion von 3-SAT auf HC.

Wir konstruieren dazu zu jeder Formel F einen Graphen G_F :

F erfüllbar gdw. G_F besitzt einen HC

Sei $F = (A_1 \vee B_1 \vee C_1) \wedge \dots \wedge (A_n \vee B_n \vee C_n)$.

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Wir zeigen NP-Vollständigkeit per Reduktion von 3-SAT auf HC.

Wir konstruieren dazu zu jeder Formel F einen Graphen G_F :

F erfüllbar gdw. G_F besitzt einen HC

Sei $F = (A_1 \vee B_1 \vee C_1) \wedge \dots \wedge (A_n \vee B_n \vee C_n)$.

Wir erzeugen uns mehrere "Bausteine":

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Wir zeigen NP-Vollständigkeit per Reduktion von 3-SAT auf HC.

Wir konstruieren dazu zu jeder Formel F einen Graphen G_F :

F erfüllbar gdw. G_F besitzt einen HC

Sei $F = (A_1 \vee B_1 \vee C_1) \wedge \dots \wedge (A_n \vee B_n \vee C_n)$.

Wir erzeugen uns mehrere "Bausteine":

- Zu jedem Literal A_i erzeugen wir einen Teilgraph.

Das Hamilton-Kreis-Problem

Satz. HC ist NP-vollständig.

Beweis: Dass HC in NP liegt, ist klar: Wir raten einen Kreis und testen, ob alle Knoten genau einmal enthalten sind.

Wir zeigen NP-Vollständigkeit per Reduktion von 3-SAT auf HC.

Wir konstruieren dazu zu jeder Formel F einen Graphen G_F :

F erfüllbar gdw. G_F besitzt einen HC

Sei $F = (A_1 \vee B_1 \vee C_1) \wedge \dots \wedge (A_n \vee B_n \vee C_n)$.

Wir erzeugen uns mehrere "Bausteine":

- ➊ Zu jedem Literal A_i erzeugen wir einen Teilgraph.
- ➋ Zu jeder Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir einen Teilgraph.

Das Hamilton-Kreis-Problem II

Das Hamilton-Kreis-Problem II

1. Baustein:

Das Hamilton-Kreis-Problem II

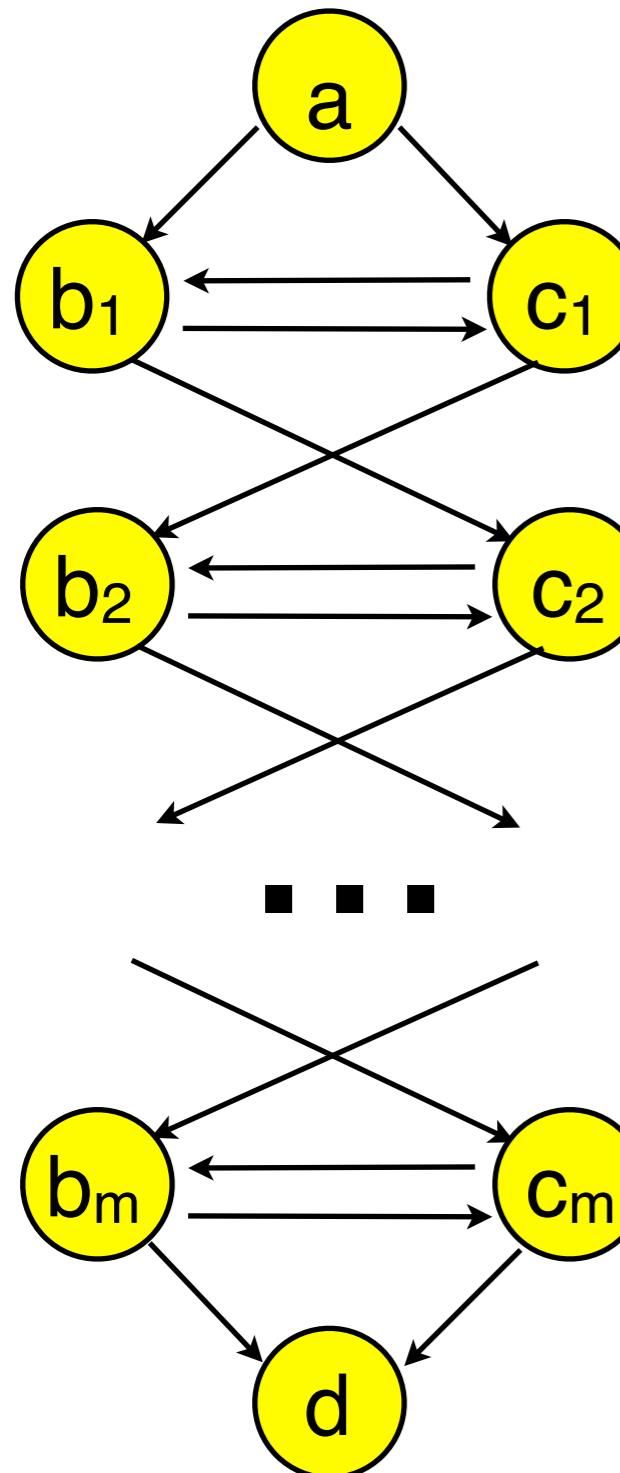
1. Baustein:

Sei m die Anzahl des Auftretens des Literals A in F .

Das Hamilton-Kreis-Problem II

1. Baustein:

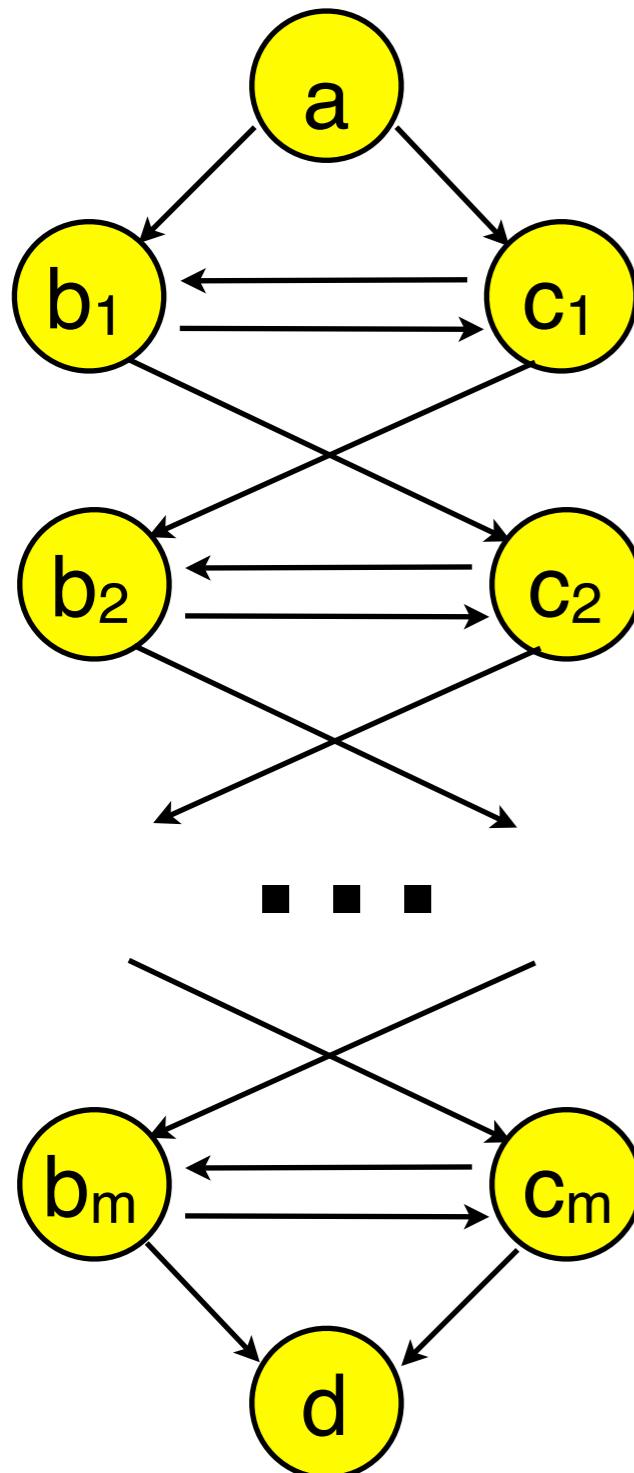
Sei m die Anzahl des Auftretens des Literals A in F .



Das Hamilton-Kreis-Problem II

1. Baustein:

Sei m die Anzahl des Auftretens des Literals A in F.

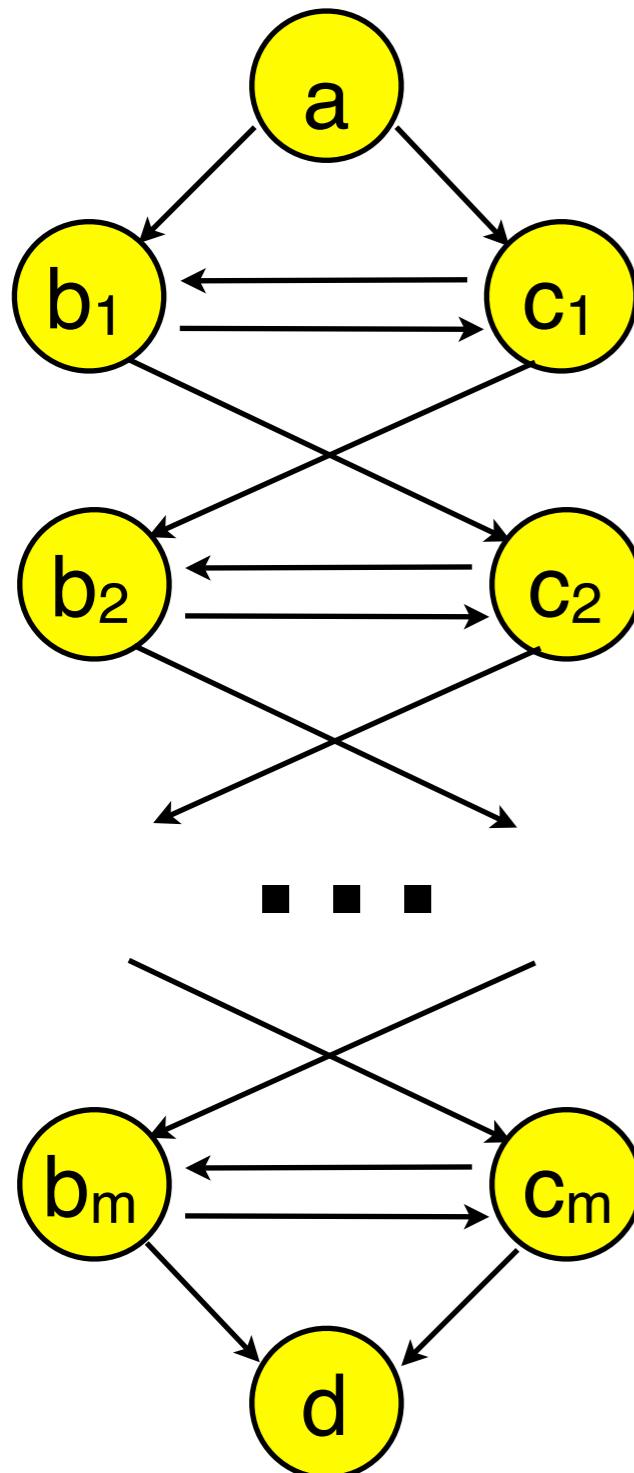


Wir suchen Pfade, die alle Knoten (genau einmal) besuchen.

Das Hamilton-Kreis-Problem II

1. Baustein:

Sei m die Anzahl des Auftretens des Literals A in F.

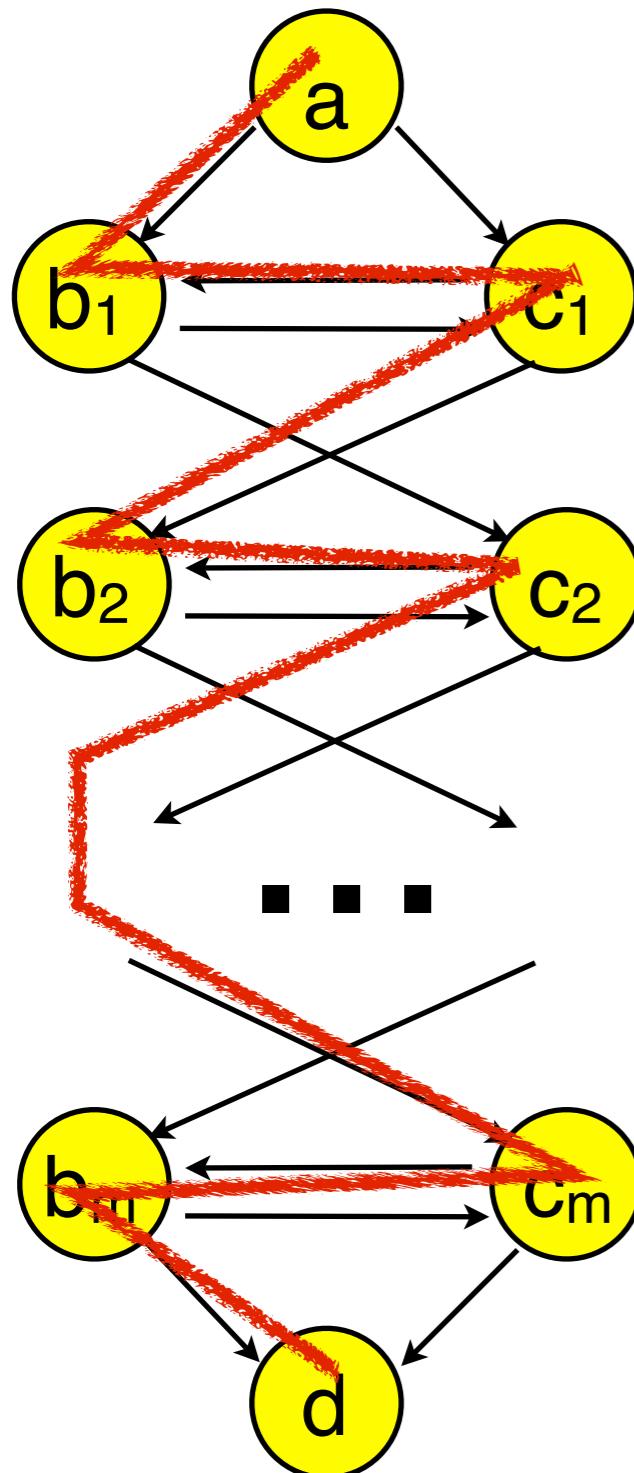


- Wir suchen Pfade, die alle Knoten (genau einmal) besuchen.
- Von a nach d existieren nur zwei Pfade, die alle Knoten besuchen.

Das Hamilton-Kreis-Problem II

1. Baustein:

Sei m die Anzahl des Auftretens des Literals A in F.

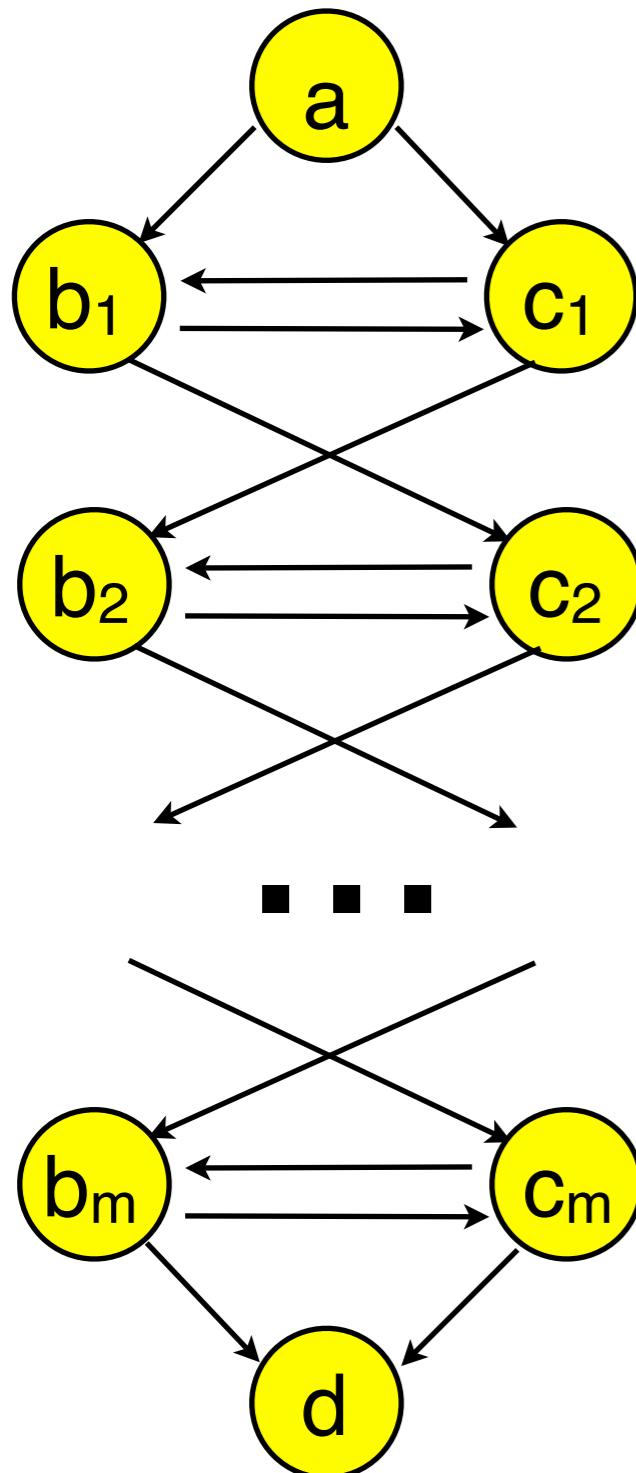


- ➊ Wir suchen Pfade, die alle Knoten (genau einmal) besuchen.
- ➋ Von a nach d existieren nur zwei Pfade, die alle Knoten besuchen.

Das Hamilton-Kreis-Problem II

1. Baustein:

Sei m die Anzahl des Auftretens des Literals A in F.

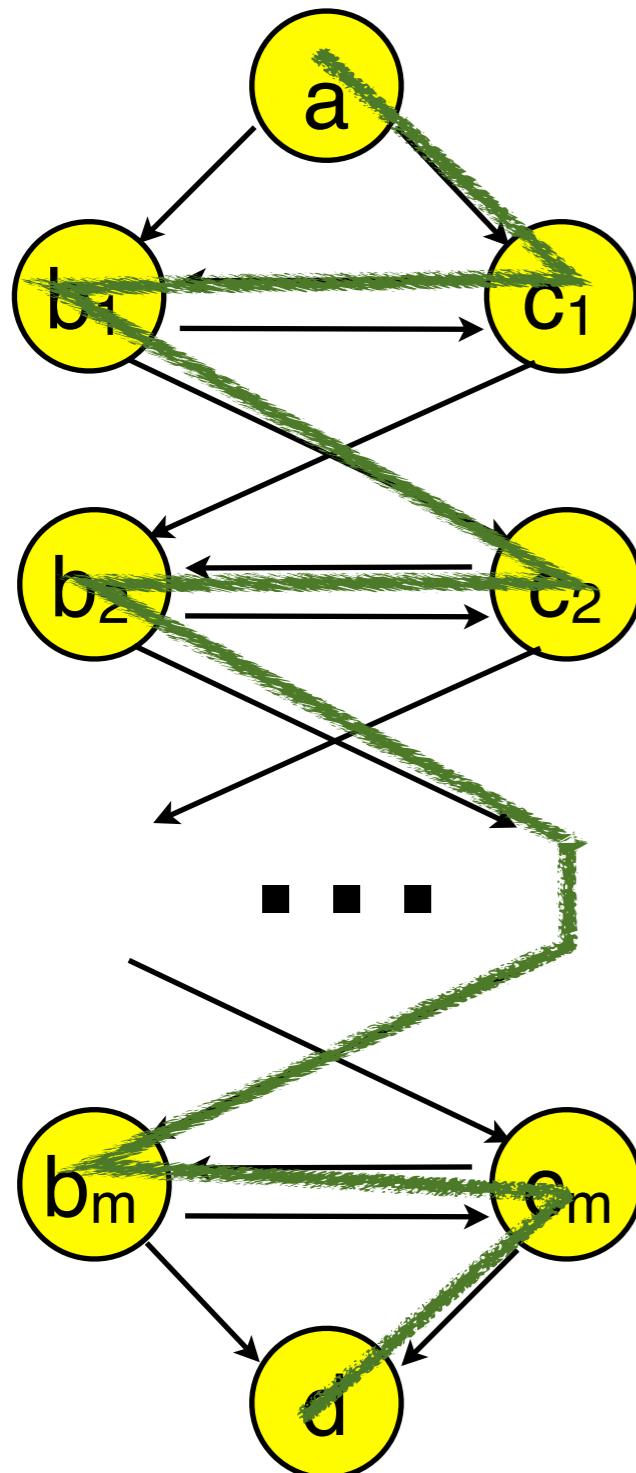


- Wir suchen Pfade, die alle Knoten (genau einmal) besuchen.
- Von a nach d existieren nur zwei Pfade, die alle Knoten besuchen.

Das Hamilton-Kreis-Problem II

1. Baustein:

Sei m die Anzahl des Auftretens des Literals A in F.

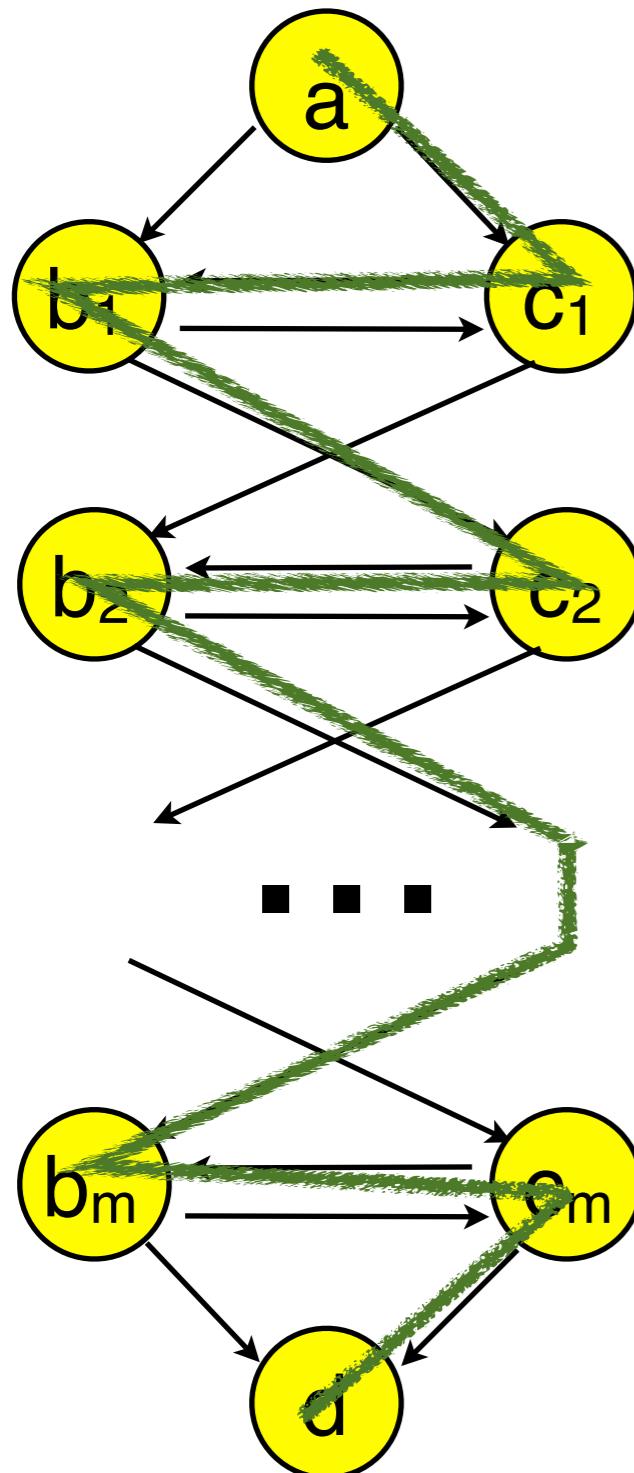


- Wir suchen Pfade, die alle Knoten (genau einmal) besuchen.
- Von a nach d existieren nur zwei Pfade, die alle Knoten besuchen.

Das Hamilton-Kreis-Problem II

1. Baustein:

Sei m die Anzahl des Auftretens des Literals A in F.



- ➊ Wir suchen Pfade, die alle Knoten (genau einmal) besuchen.
- ➋ Von a nach d existieren nur zwei Pfade, die alle Knoten besuchen.

- ➌ Die Wahl der Kante in Knoten a legt den Pfad bis zum Ende fest.
- ➍ Idee: Der eine Pfad kodiert die Belegung "wahr", der andere "falsch".

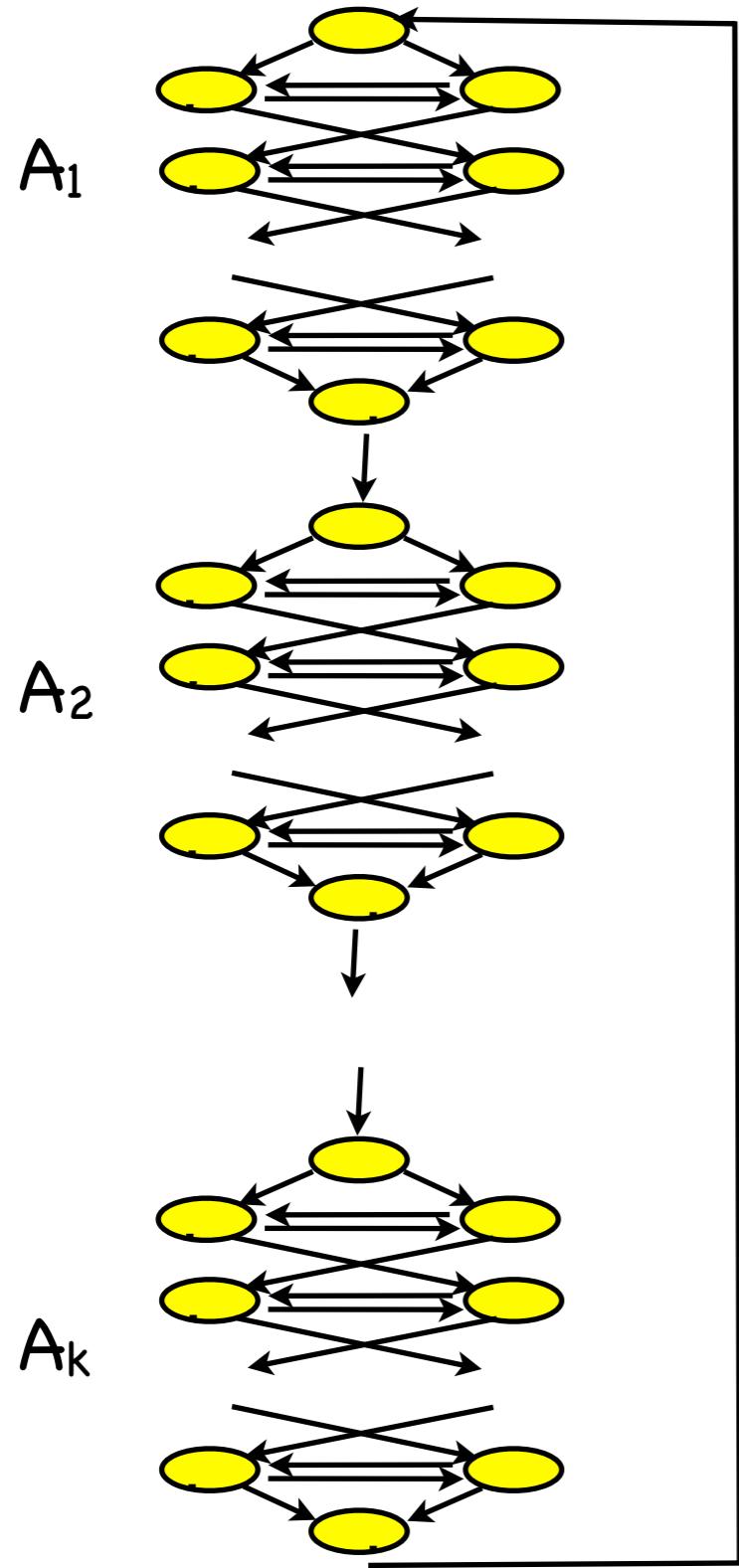
Das Hamilton-Kreis-Problem III

Das Hamilton-Kreis-Problem III

Wir erstellen für jedes Literal A der Formel F einen Teilgraph und verknüfen diese Graphen zu einem Kreis:

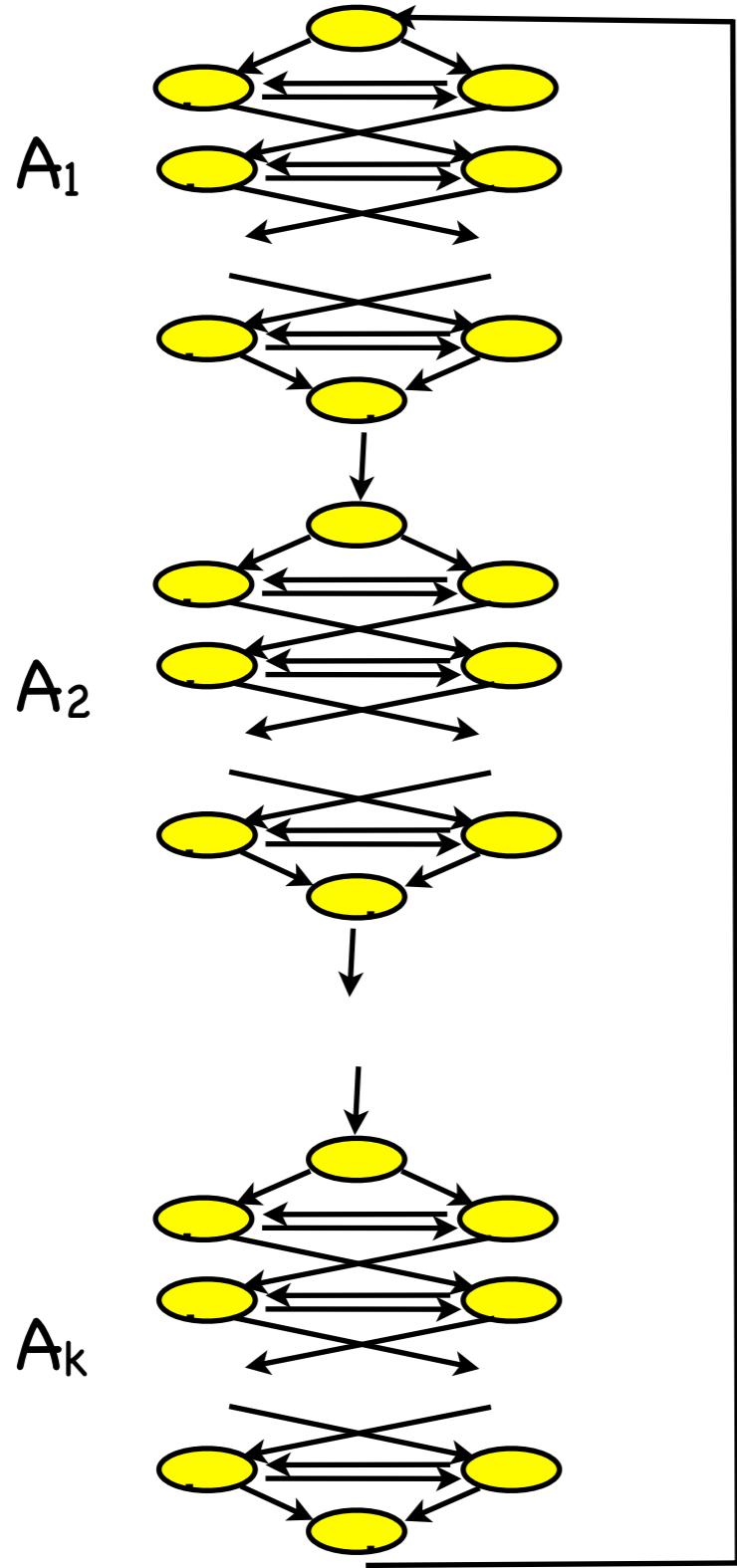
Das Hamilton-Kreis-Problem III

Wir erstellen für jedes Literal A der Formel F einen Teilgraph und verknüfen diese Graphen zu einem Kreis:



Das Hamilton-Kreis-Problem III

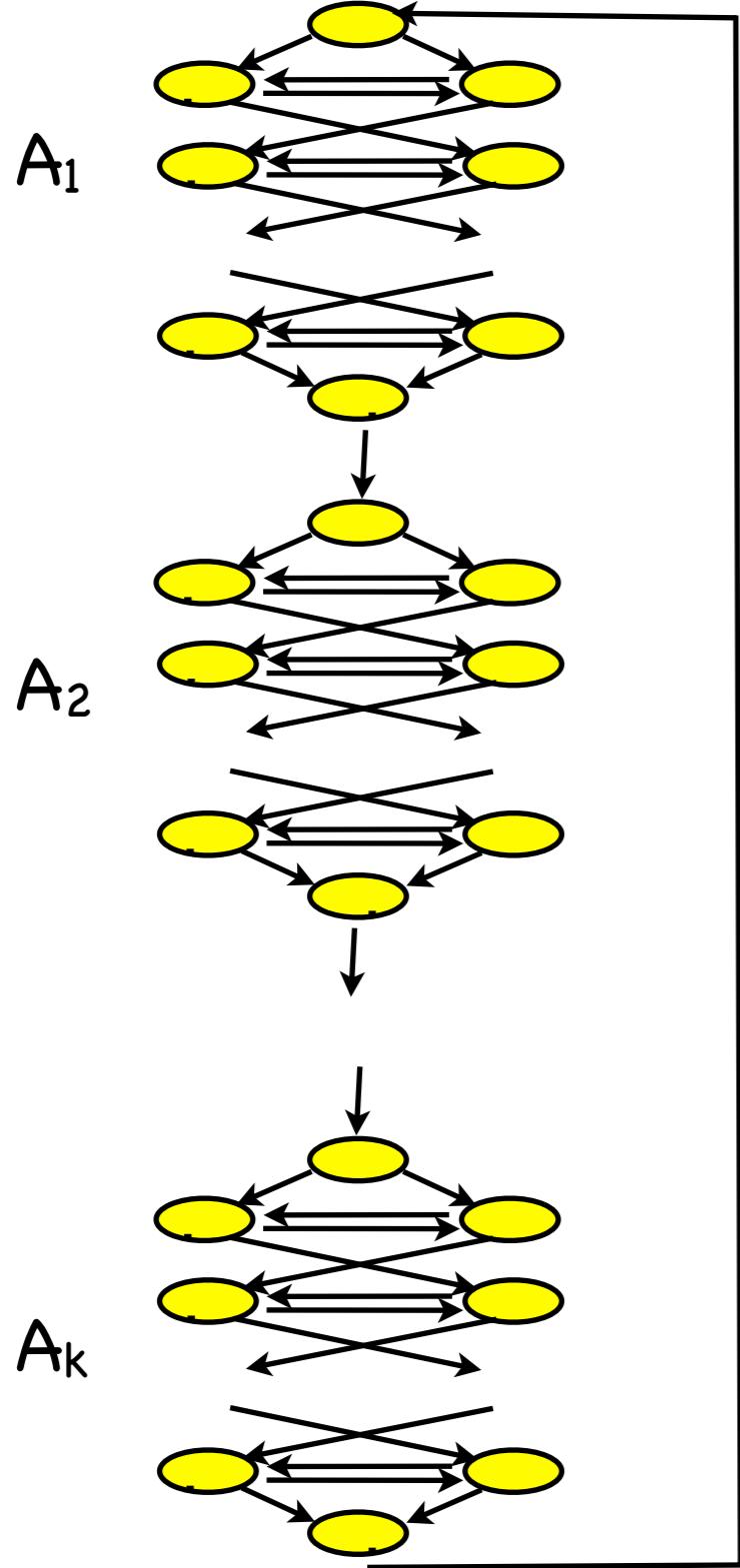
Wir erstellen für jedes Literal A der Formel F einen Teilgraph und verknüfen diese Graphen zu einem Kreis:



Betrachten wir einen
Hamilton-Kreis:

Das Hamilton-Kreis-Problem III

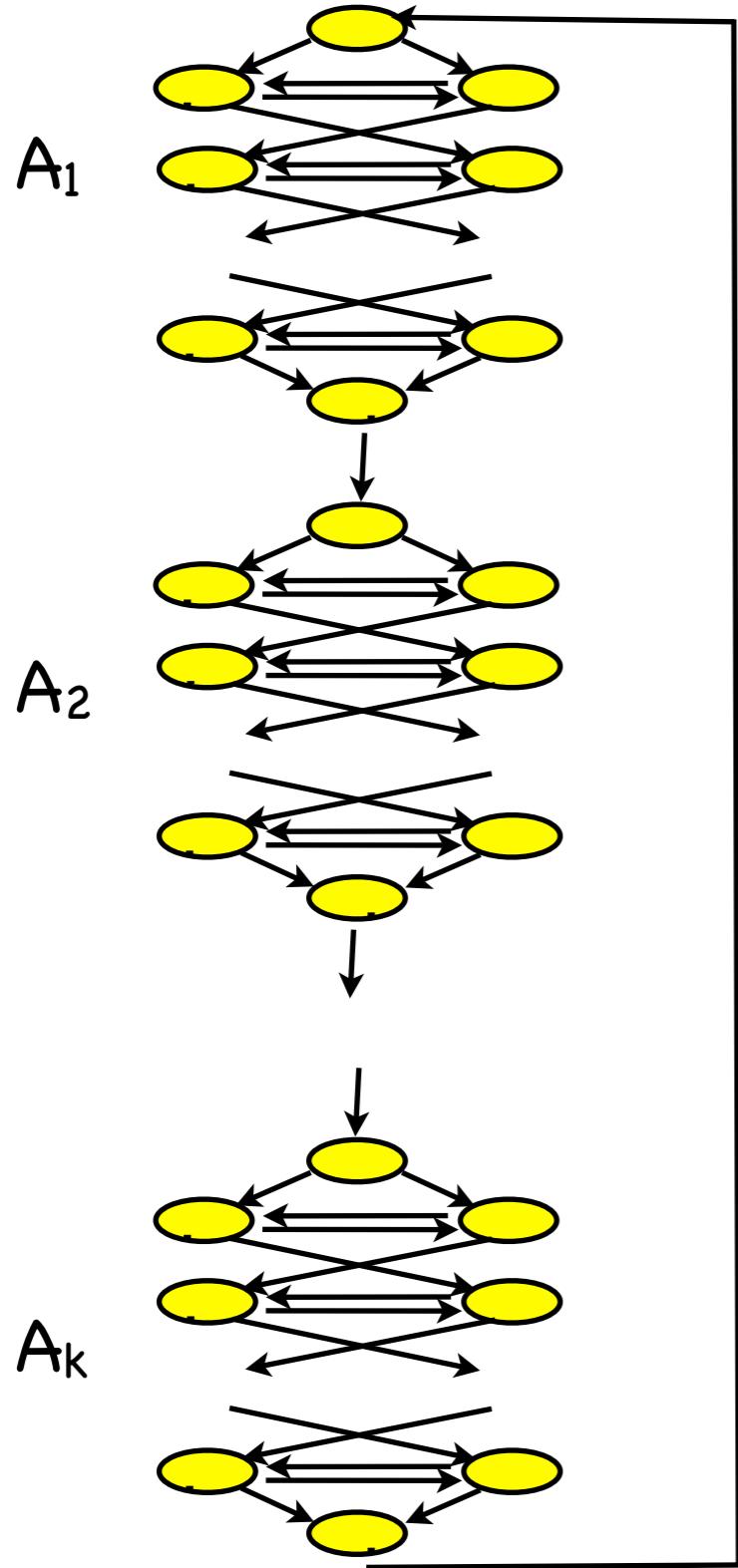
Wir erstellen für jedes Literal A der Formel F einen Teilgraph und verknüfen diese Graphen zu einem Kreis:



- Betrachten wir einen Hamilton-Kreis:
- Dieser muss oben ein- und unten austreten.

Das Hamilton-Kreis-Problem III

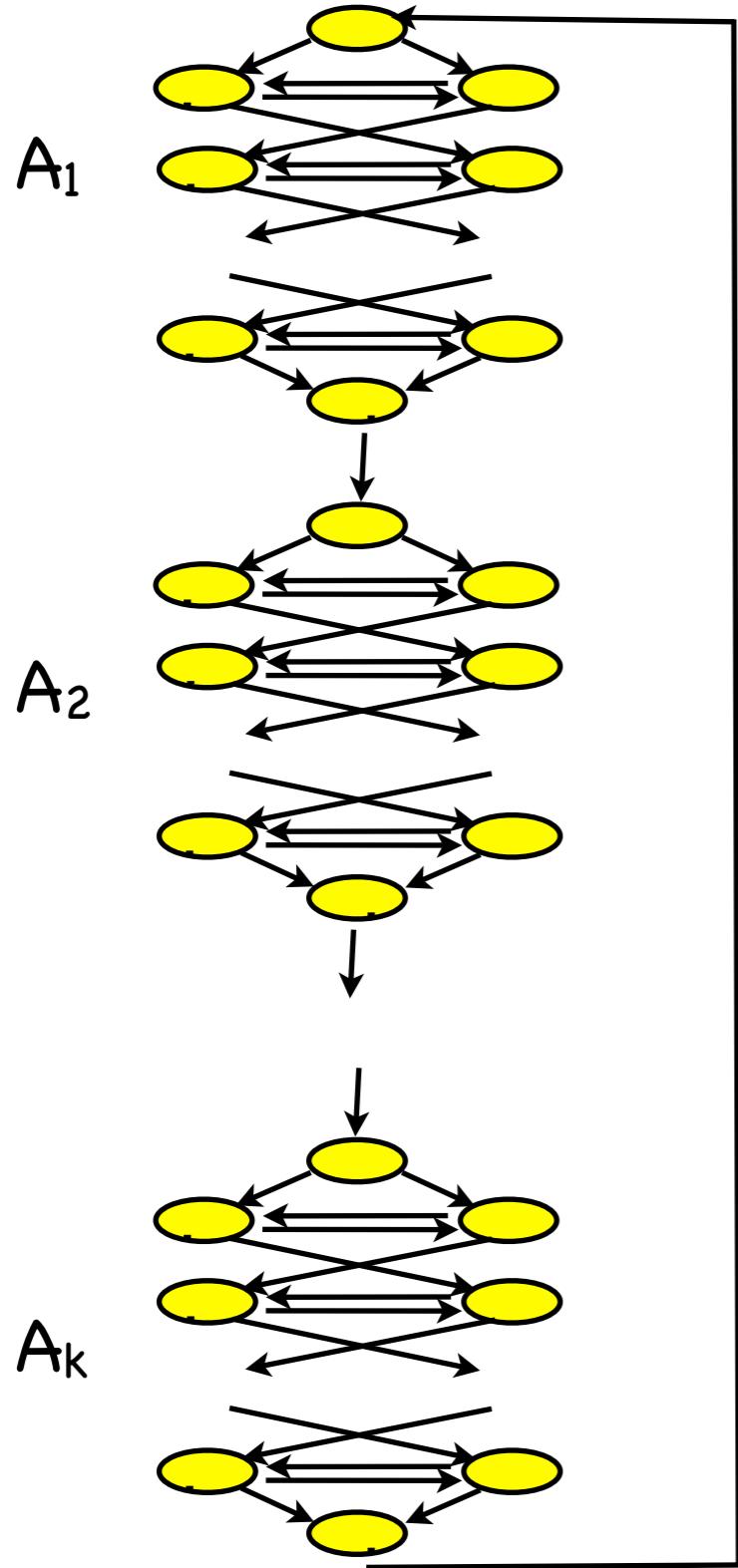
Wir erstellen für jedes Literal A der Formel F einen Teilgraph und verknüfen diese Graphen zu einem Kreis:



- Betrachten wir einen Hamilton-Kreis:
- Dieser muss oben ein- und unten austreten.
- Für jeden A_i -Teilgraph haben wir 2 Möglichkeiten.

Das Hamilton-Kreis-Problem III

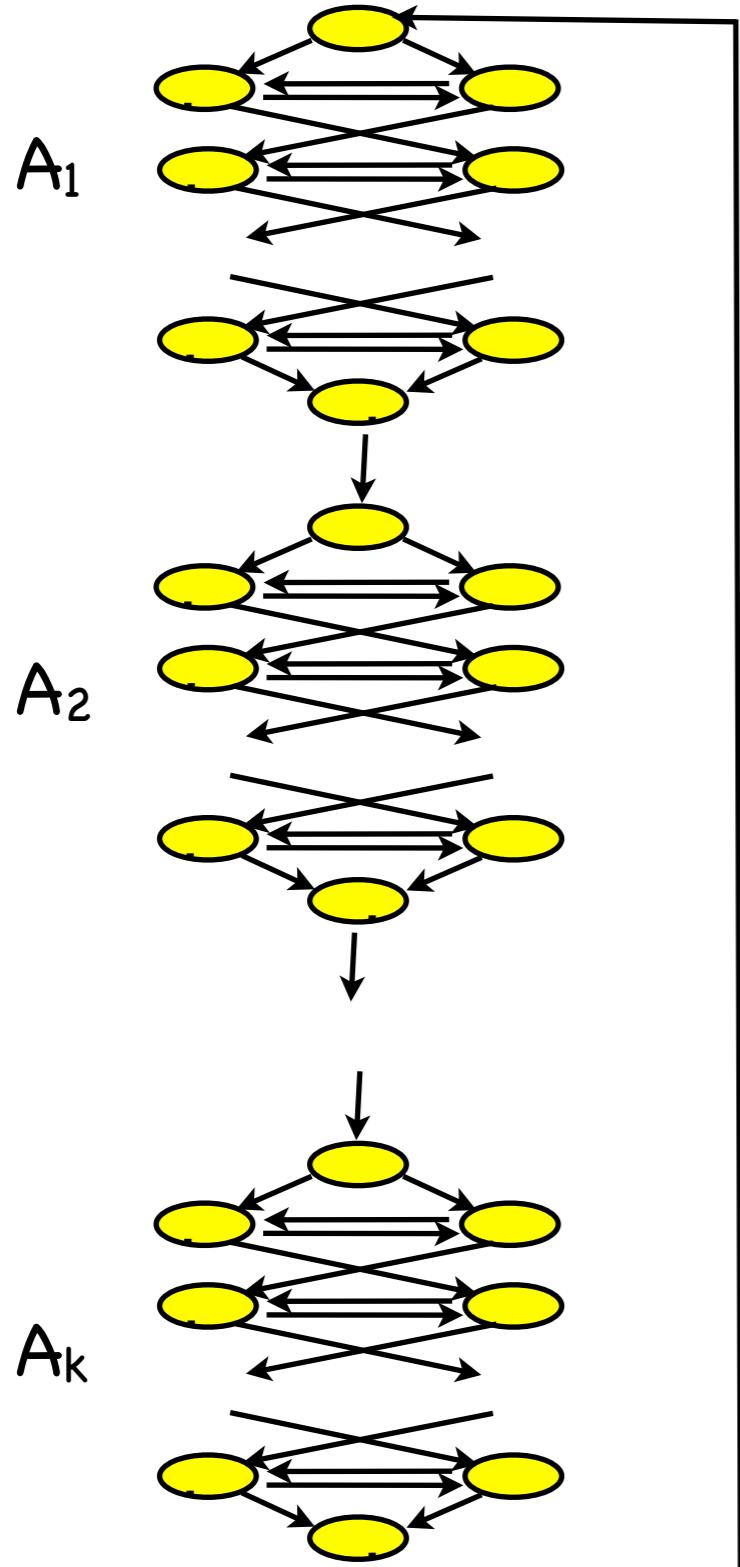
Wir erstellen für jedes Literal A der Formel F einen Teilgraph und verknüfen diese Graphen zu einem Kreis:



- ➊ Betrachten wir einen Hamilton-Kreis:
- ➋ Dieser muss oben ein- und unten austreten.
- ➌ Für jeden A_i -Teilgraph haben wir 2 Möglichkeiten.
- ➍ Bei k Atomen haben wir damit 2^k mögliche Pfade.

Das Hamilton-Kreis-Problem III

Wir erstellen für jedes Literal A der Formel F einen Teilgraph und verknüfen diese Graphen zu einem Kreis:



- Betrachten wir einen Hamilton-Kreis:
- Dieser muss oben ein- und unten austreten.
- Für jeden A_i -Teilgraph haben wir 2 Möglichkeiten.
- Bei k Atomen haben wir damit 2^k mögliche Pfade.
- Jeder Hamilton-Kreis kodiert somit genau eine Belegung der Formel.

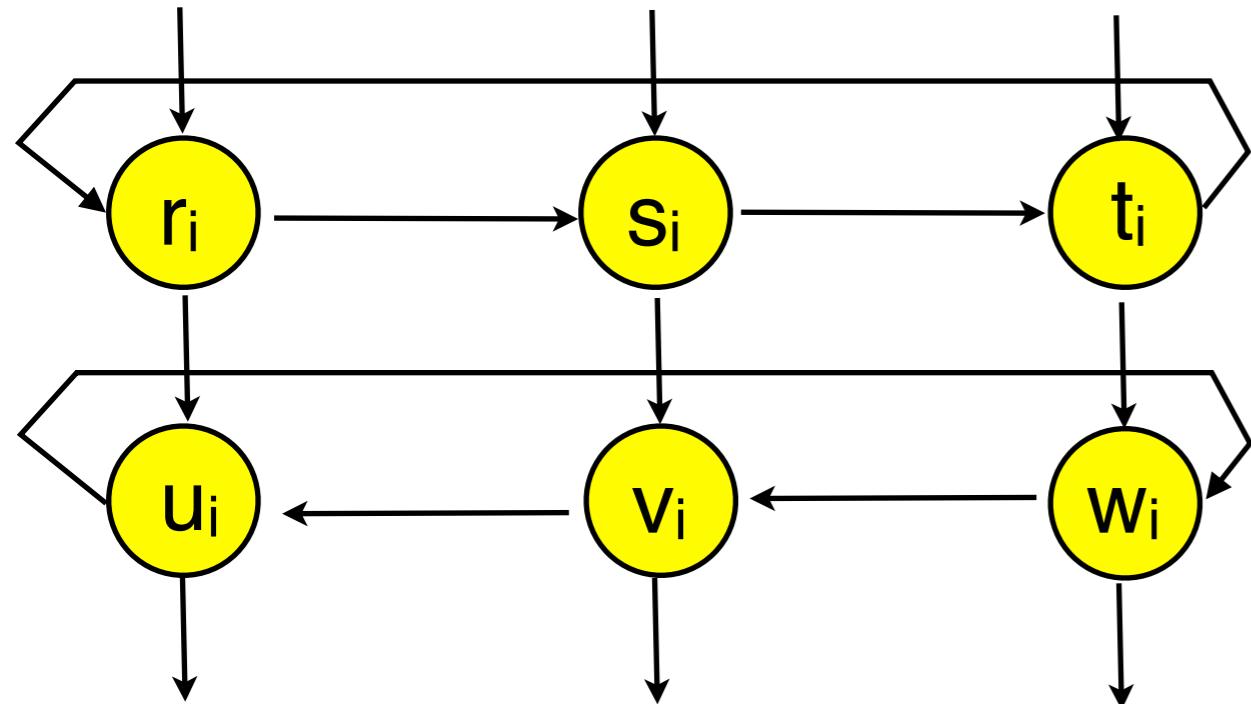
Hamilton-Kreis Problem IV

Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:

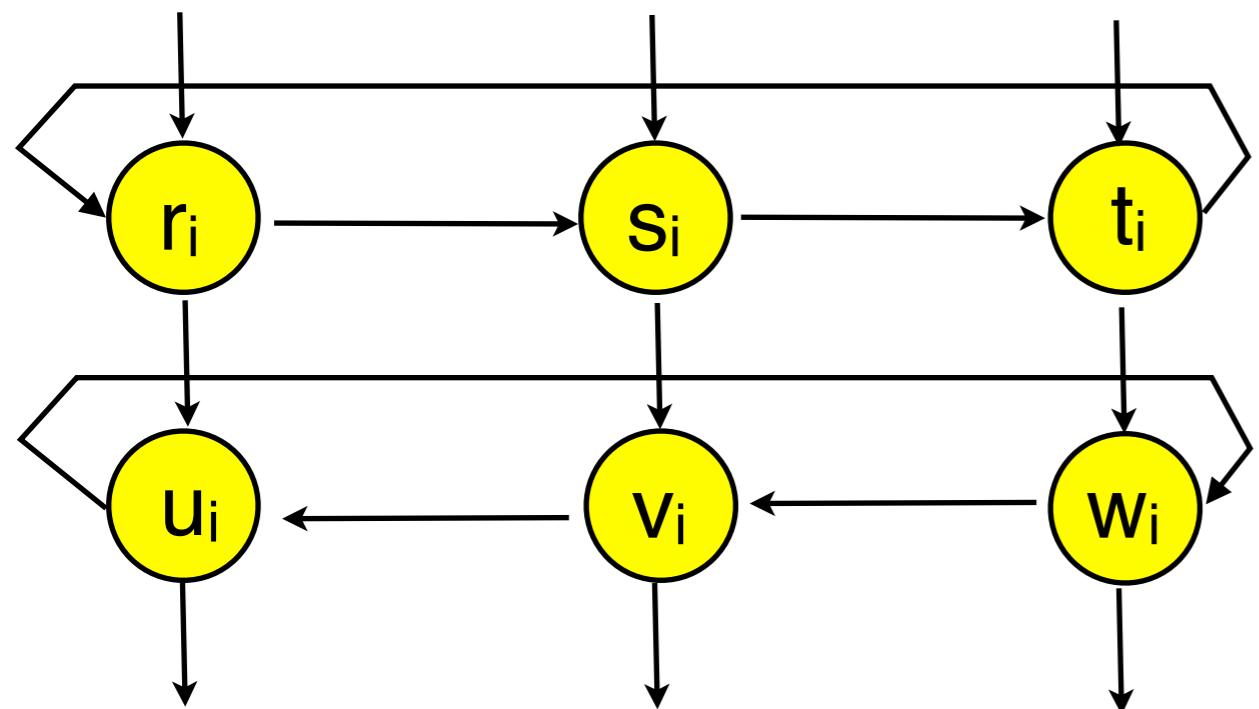
Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



Hamilton-Kreis Problem IV

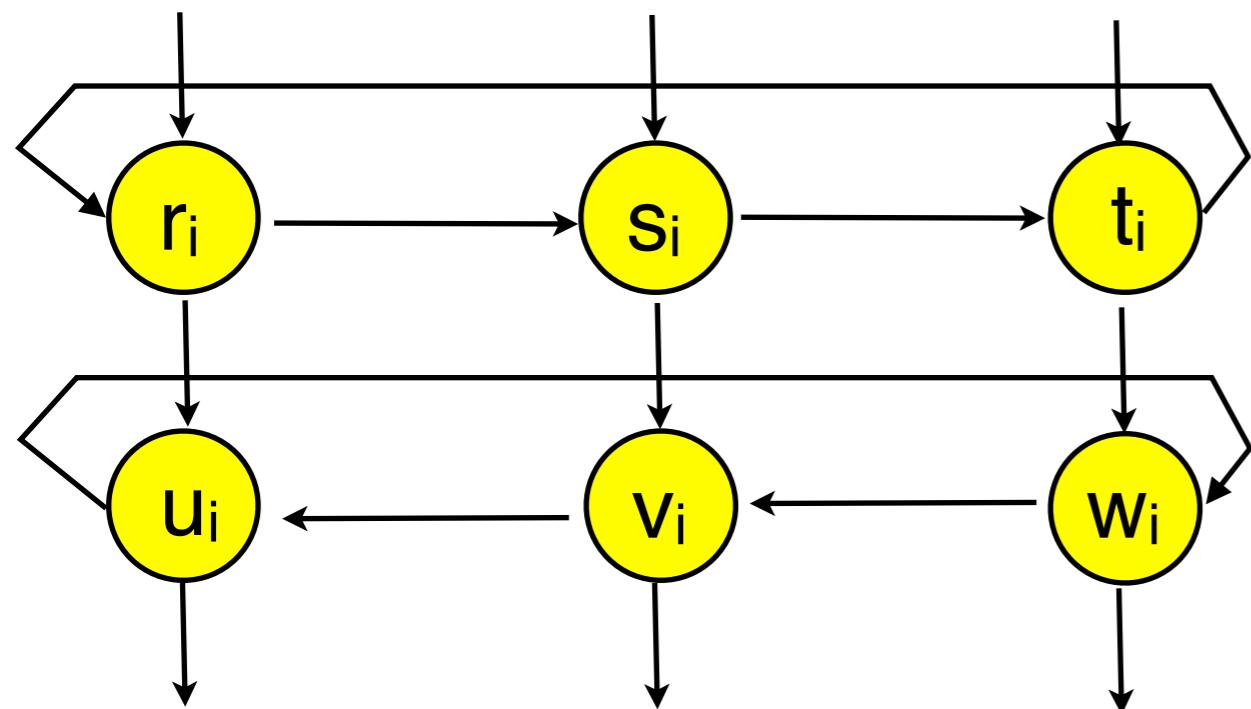
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



• Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.

Hamilton-Kreis Problem IV

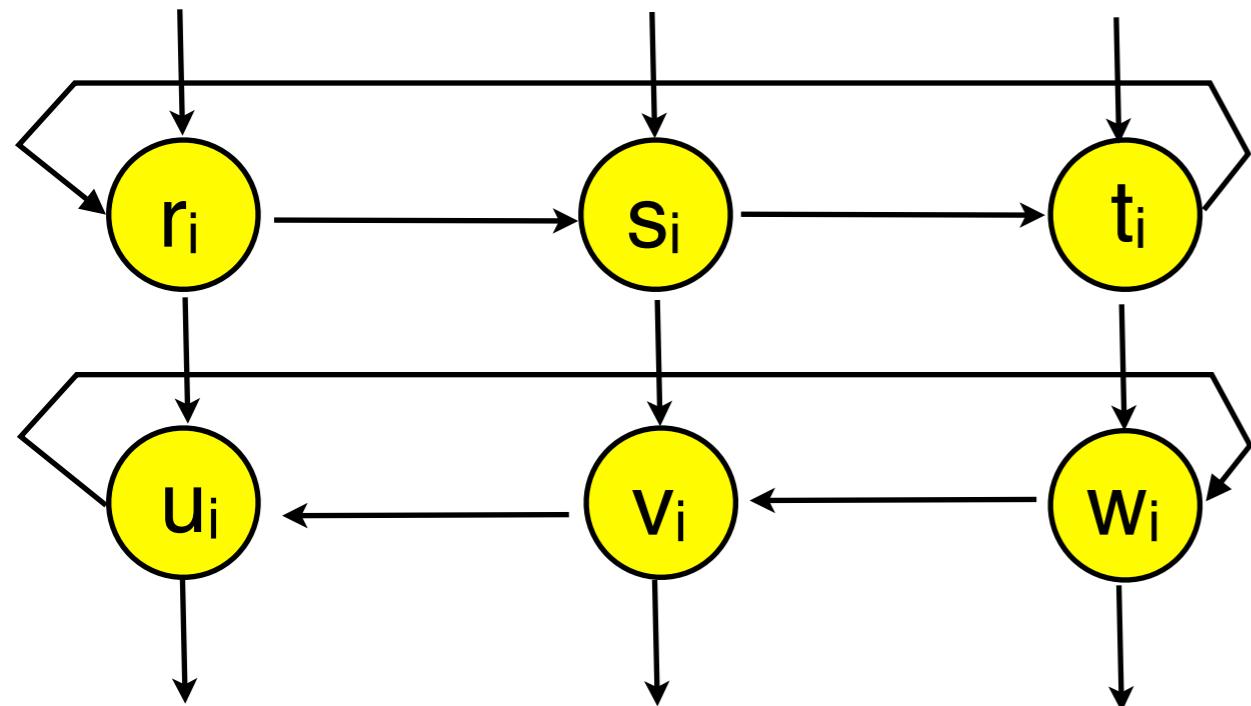
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:

Hamilton-Kreis Problem IV

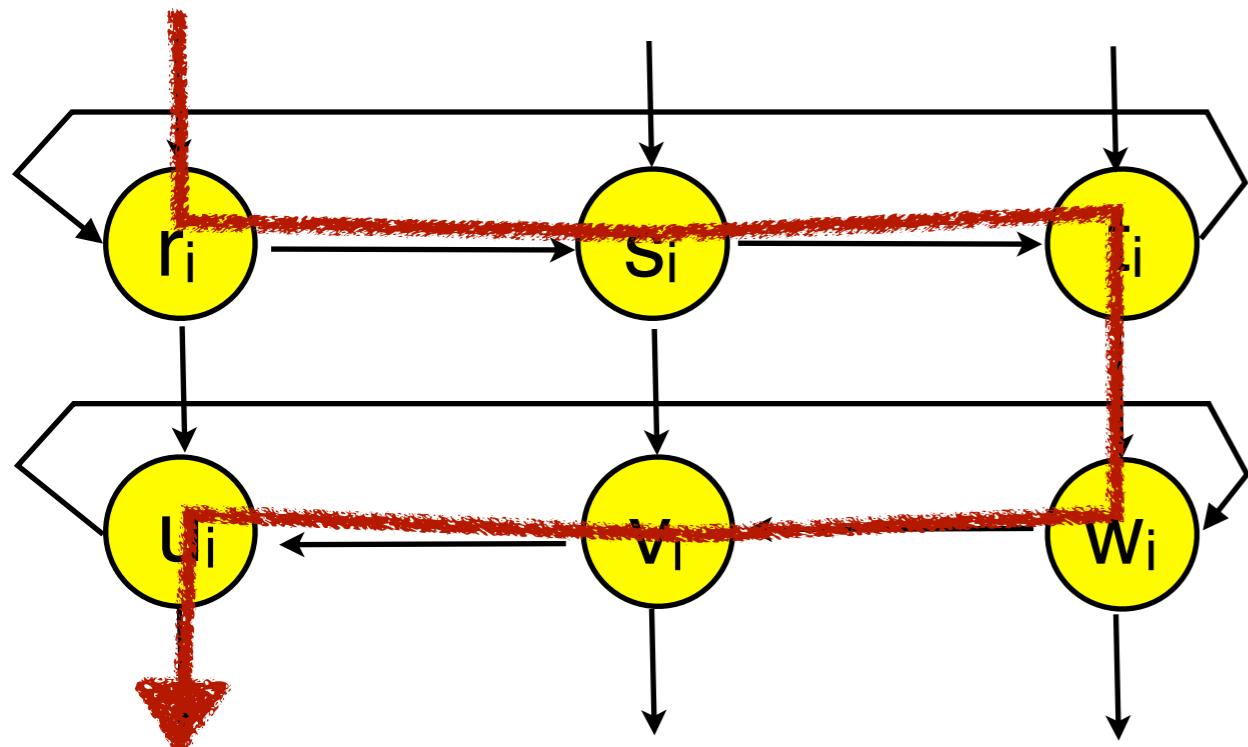
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

Hamilton-Kreis Problem IV

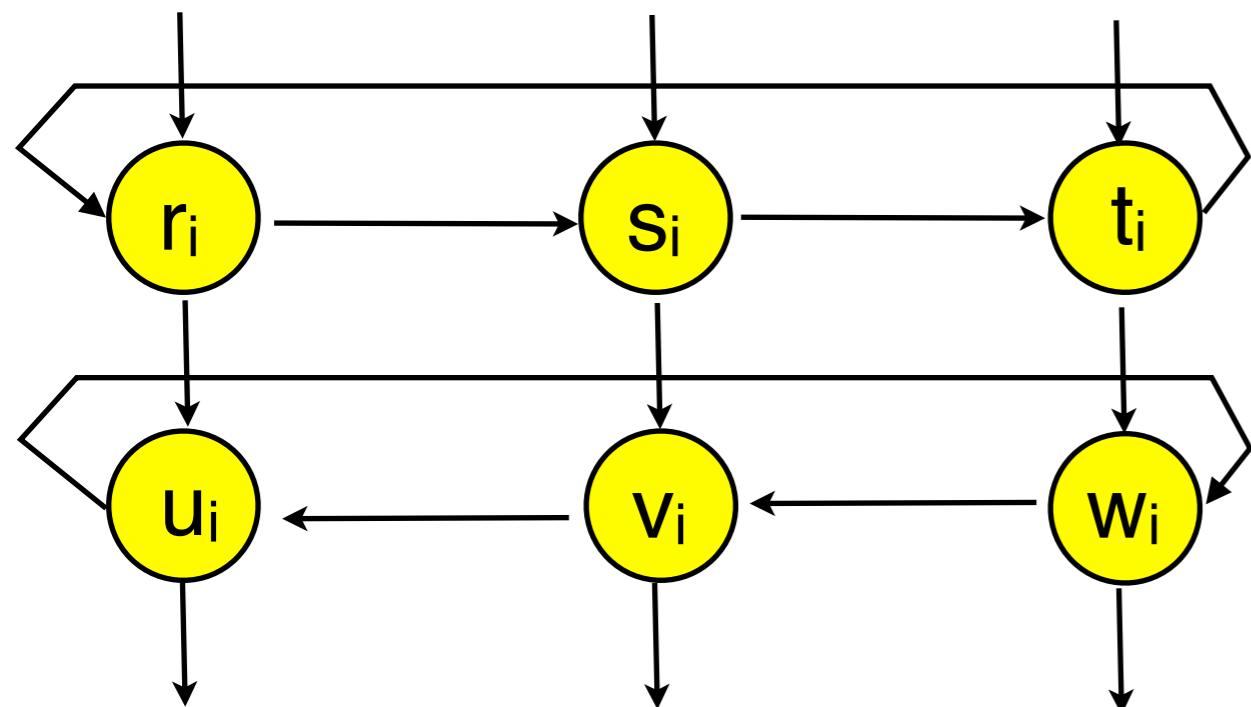
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

Hamilton-Kreis Problem IV

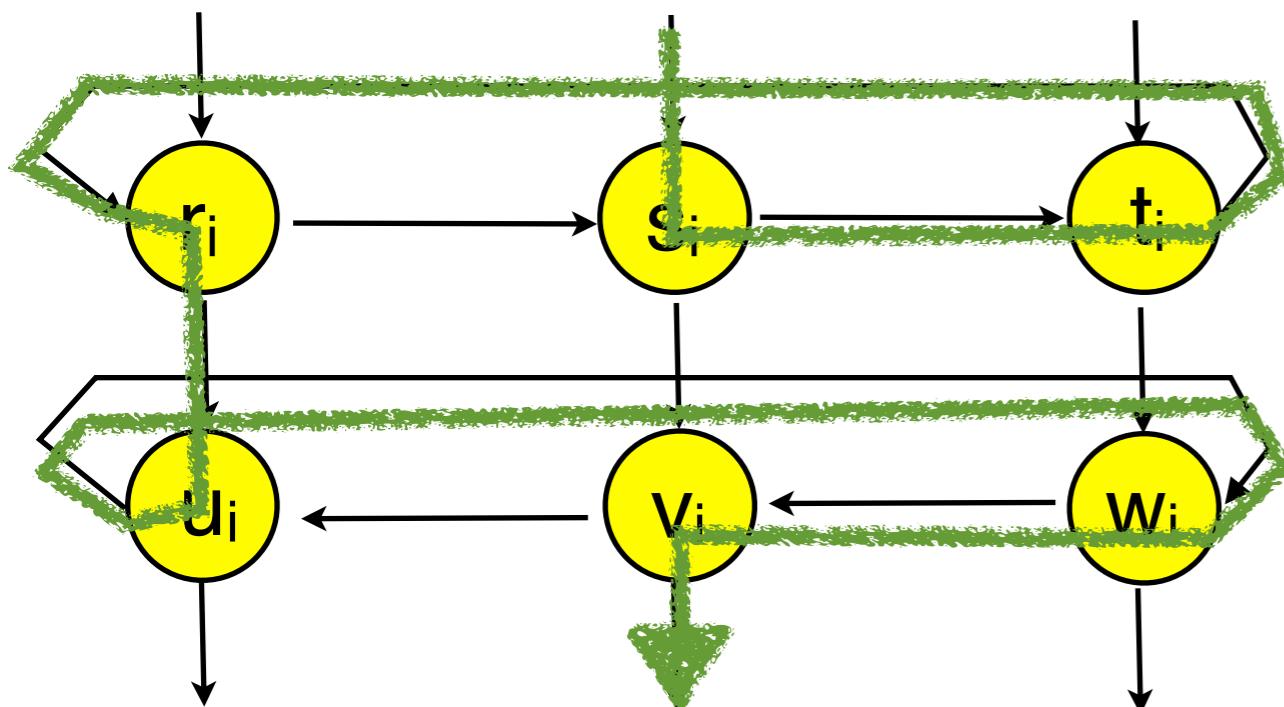
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

Hamilton-Kreis Problem IV

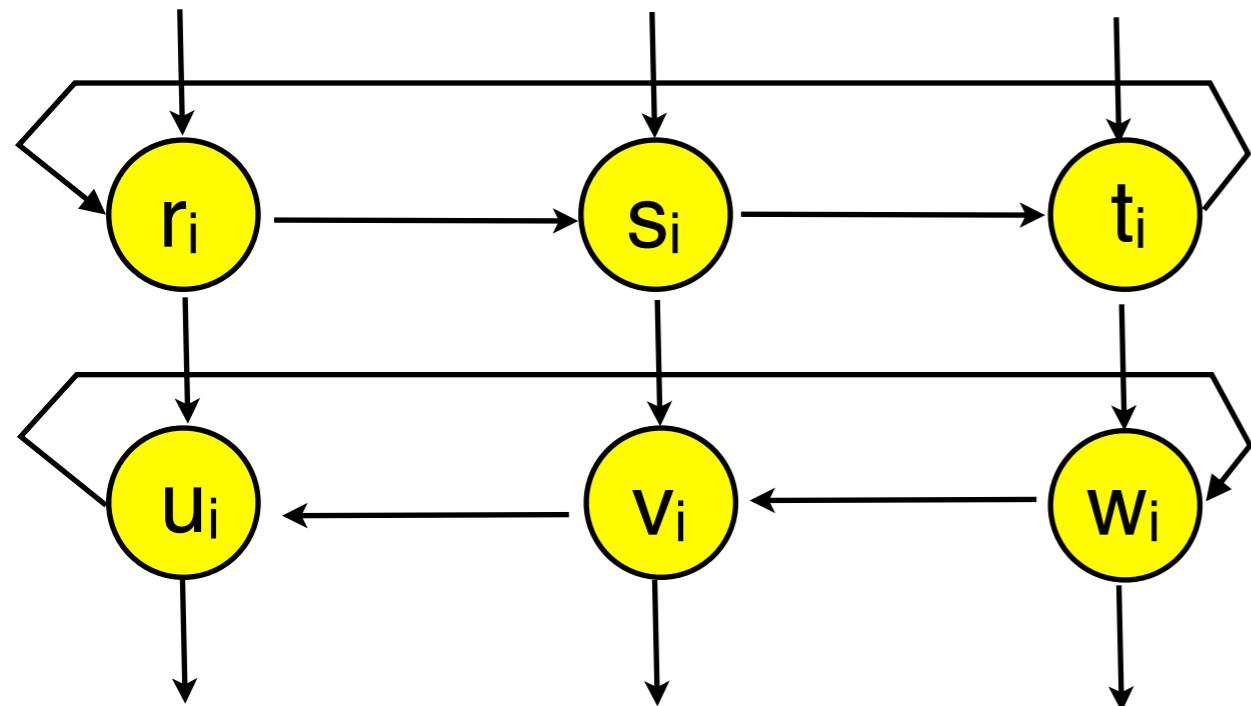
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

Hamilton-Kreis Problem IV

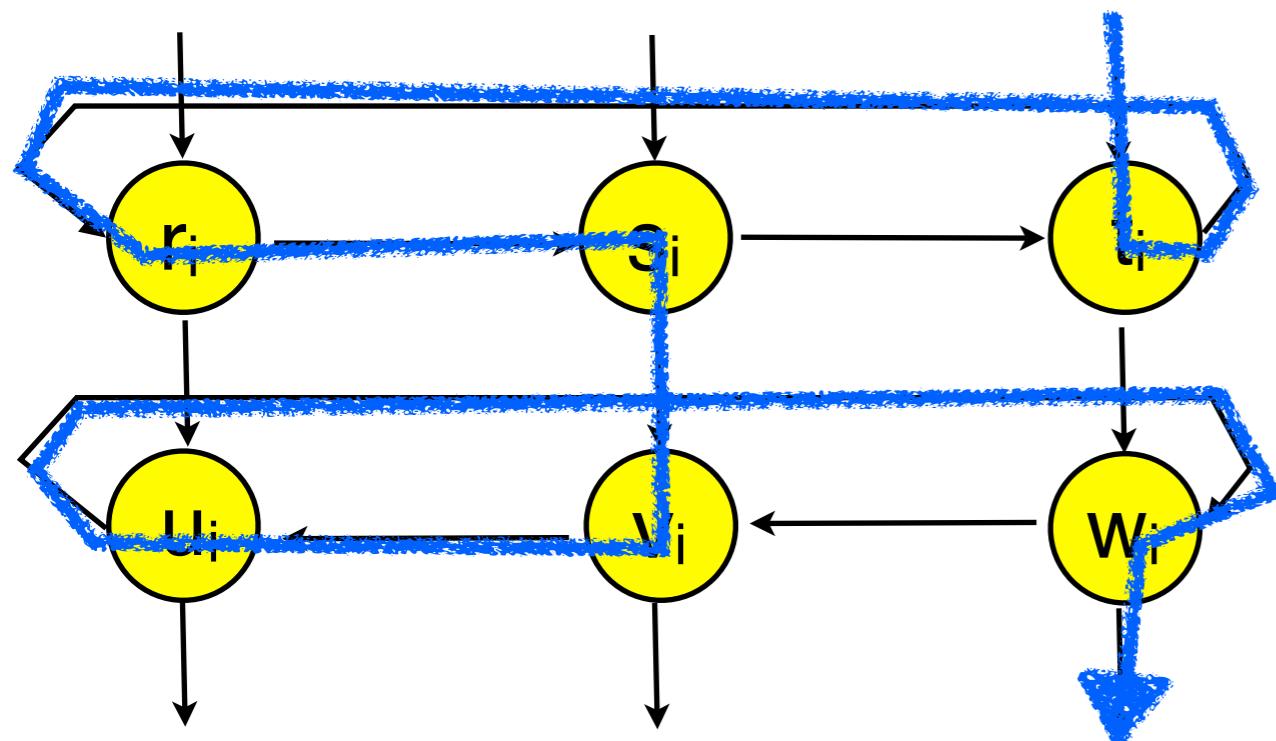
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

Hamilton-Kreis Problem IV

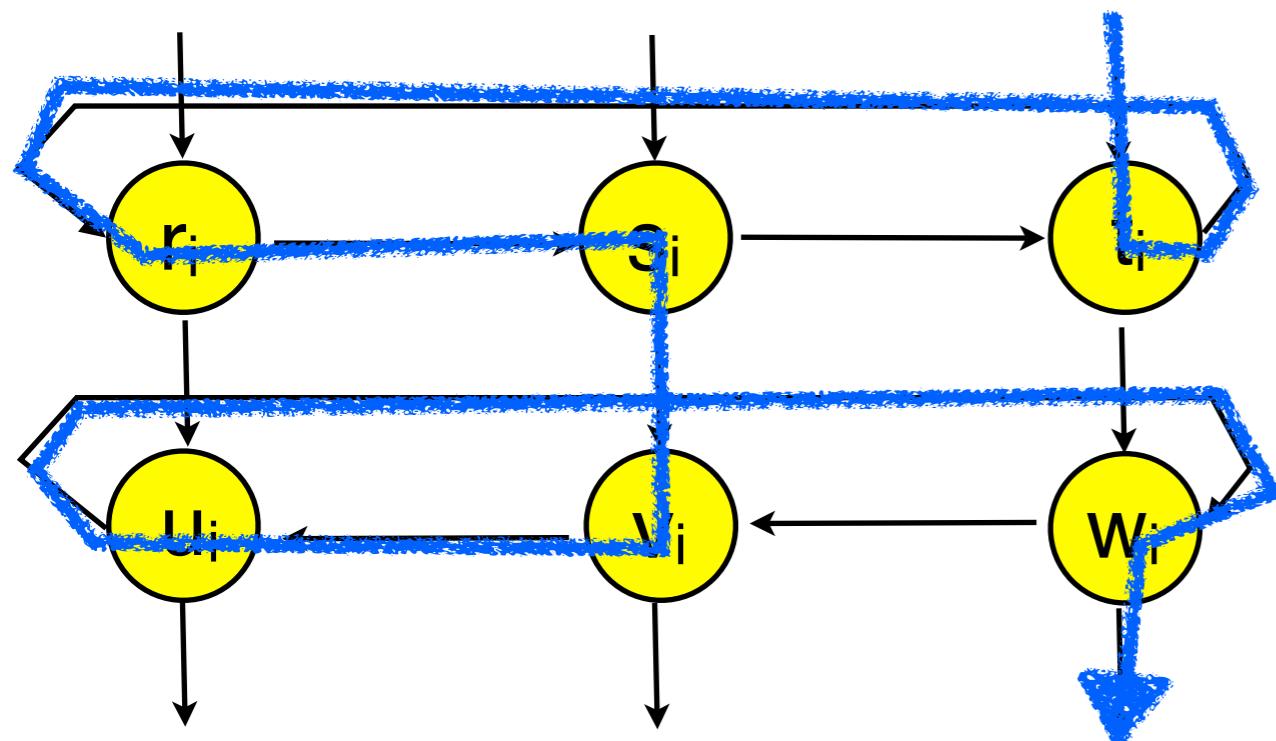
2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:

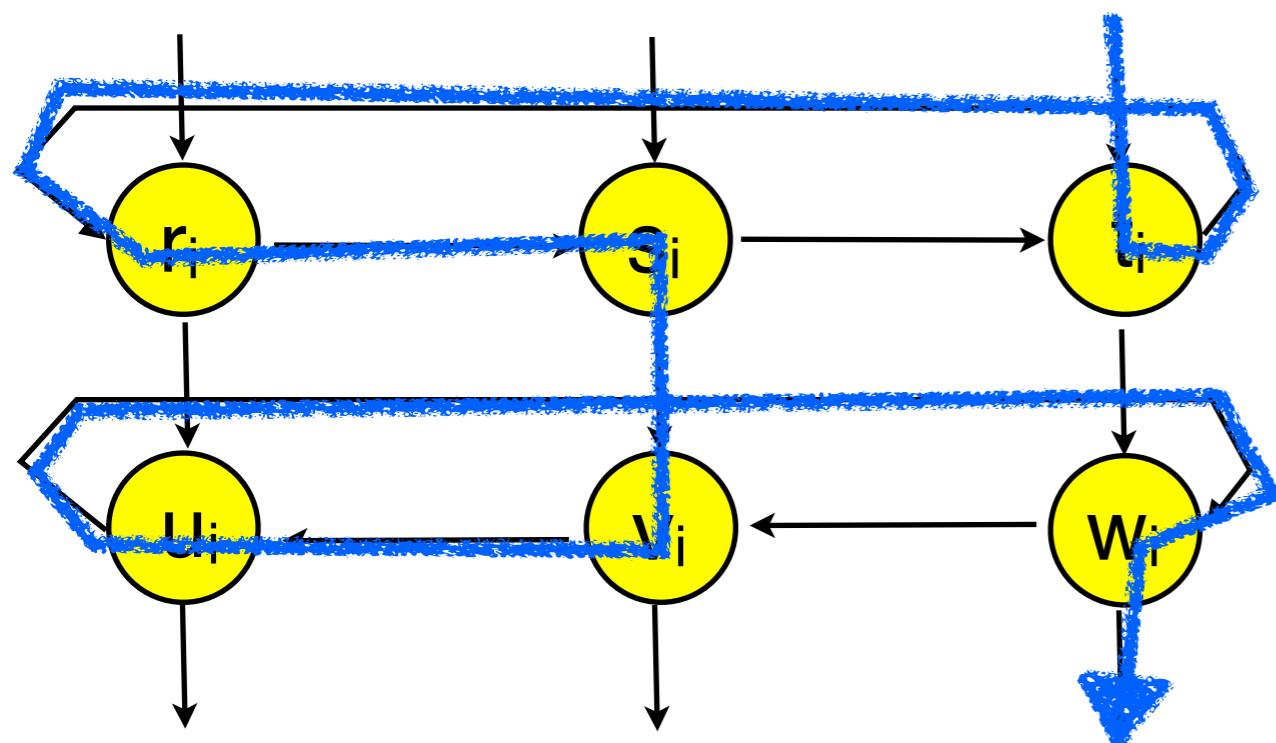


- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

• Wir verbinden jetzt die Bausteine:

Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



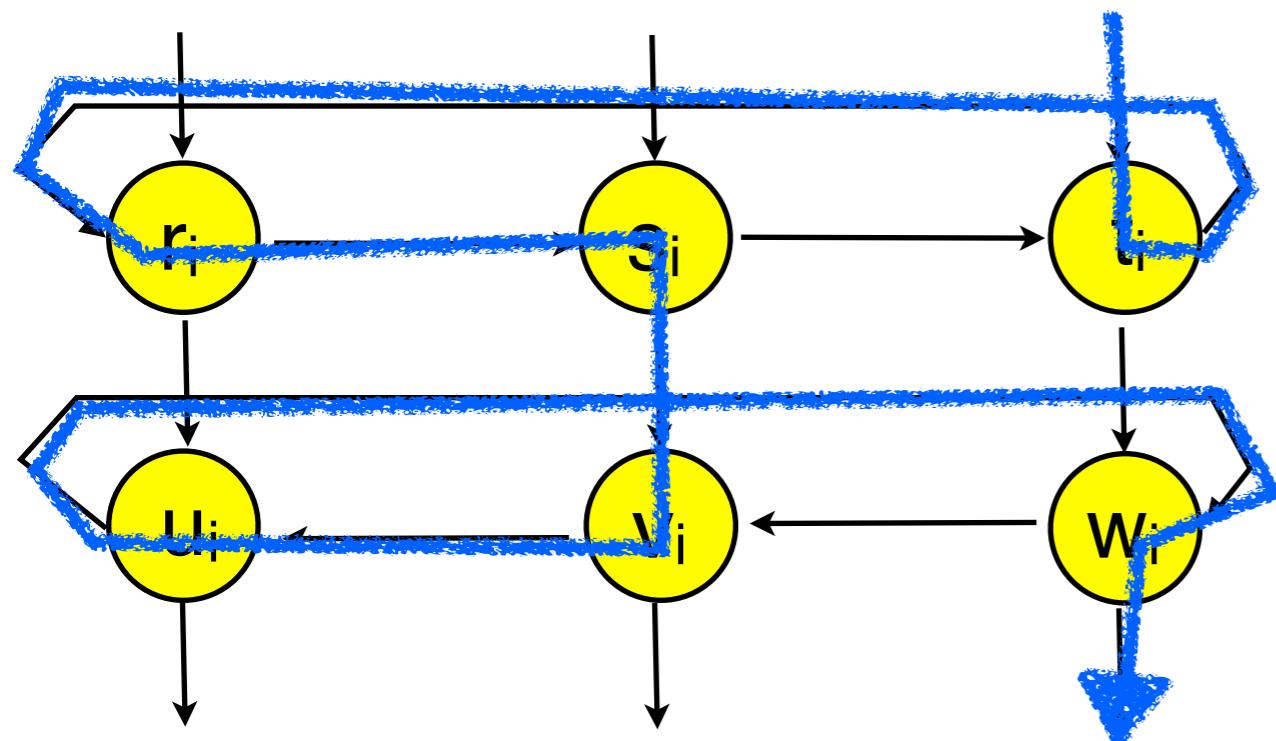
- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

• Wir verbinden jetzt die Bausteine:

• Wenn das Literal A_i unnegiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" b-Knoten aus dem A_i -Graph zu r_i und

Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



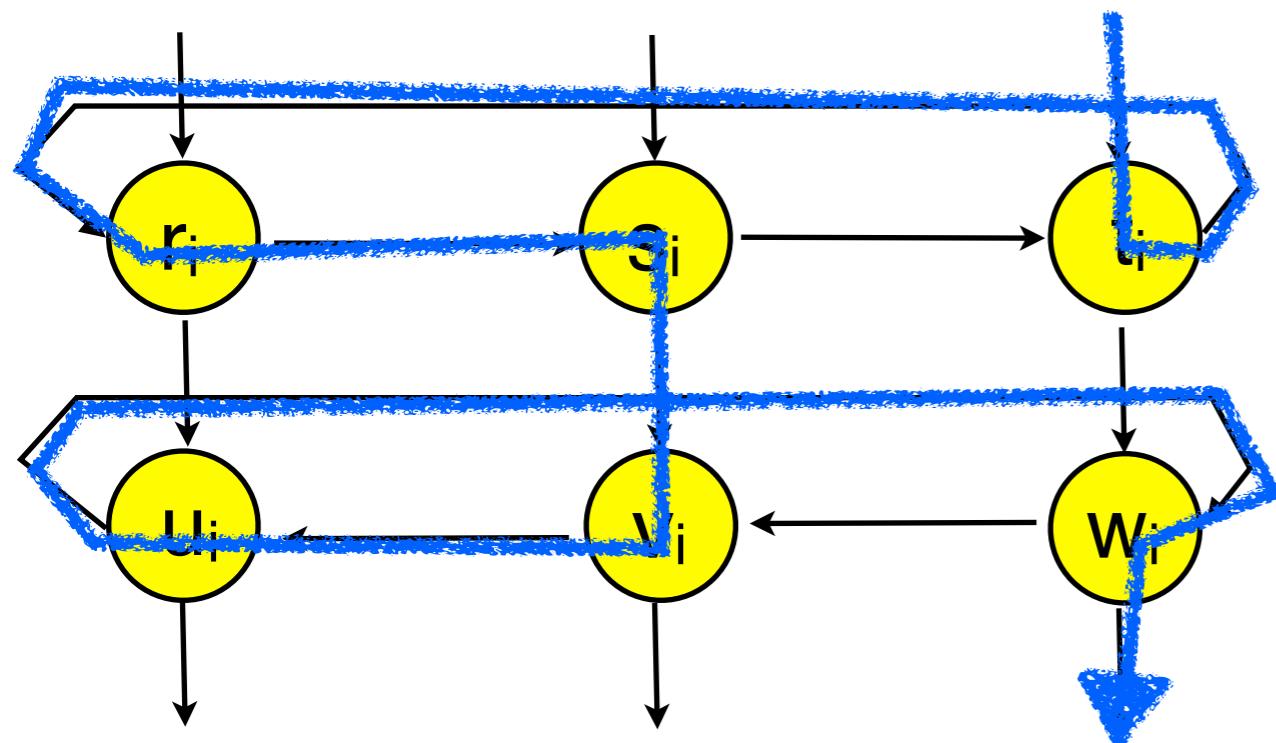
- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

• Wir verbinden jetzt die Bausteine:

• Wenn das Literal A_i unnegiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" **b-Knoten** aus dem A_i -Graph zu r_i und von u_i zu dem **c-Knoten**, der diagonal unter dem **b-Knoten** liegt.

Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



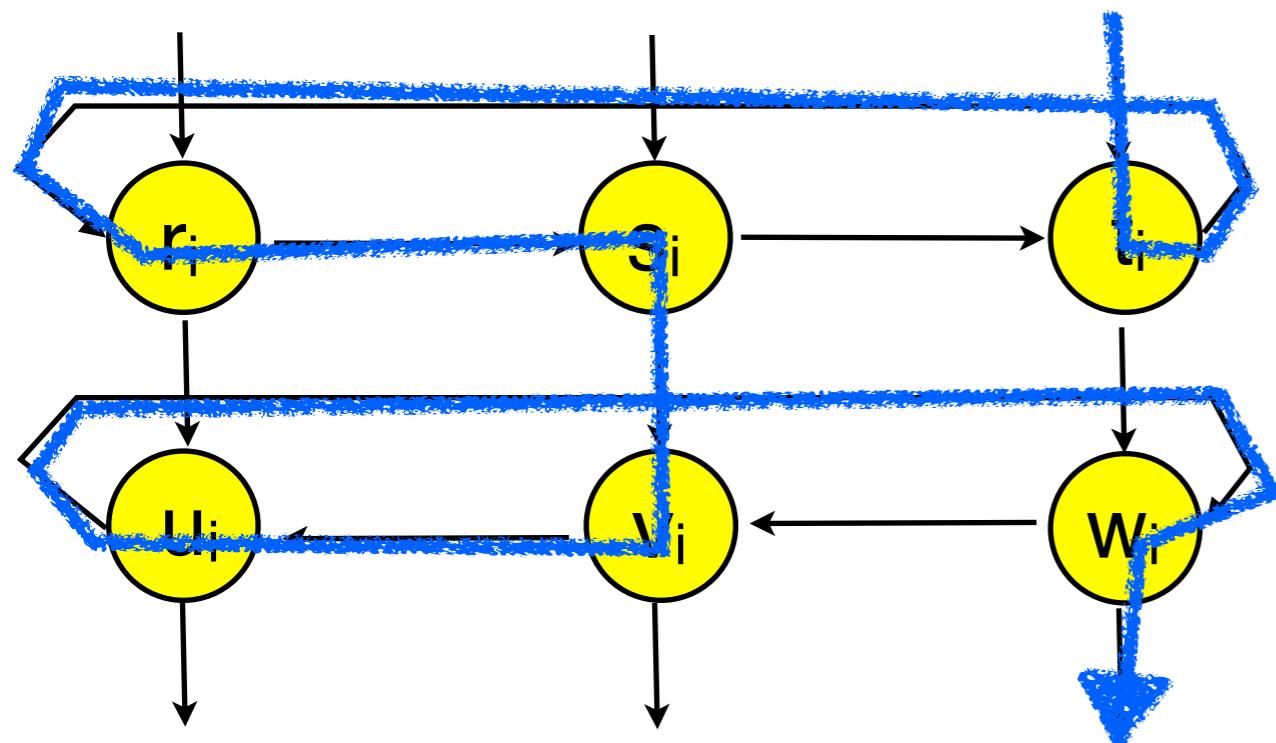
- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

• Wir verbinden jetzt die Bausteine:

- Wenn das Literal A_i unnegiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" **b-Knoten** aus dem A_i -Graph zu r_i und von u_i zu dem **c-Knoten**, der diagonal unter dem b-Knoten liegt.
- Umgekehrt: Wenn das Atom A_i negiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" **c-Knoten** zu r_i und

Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

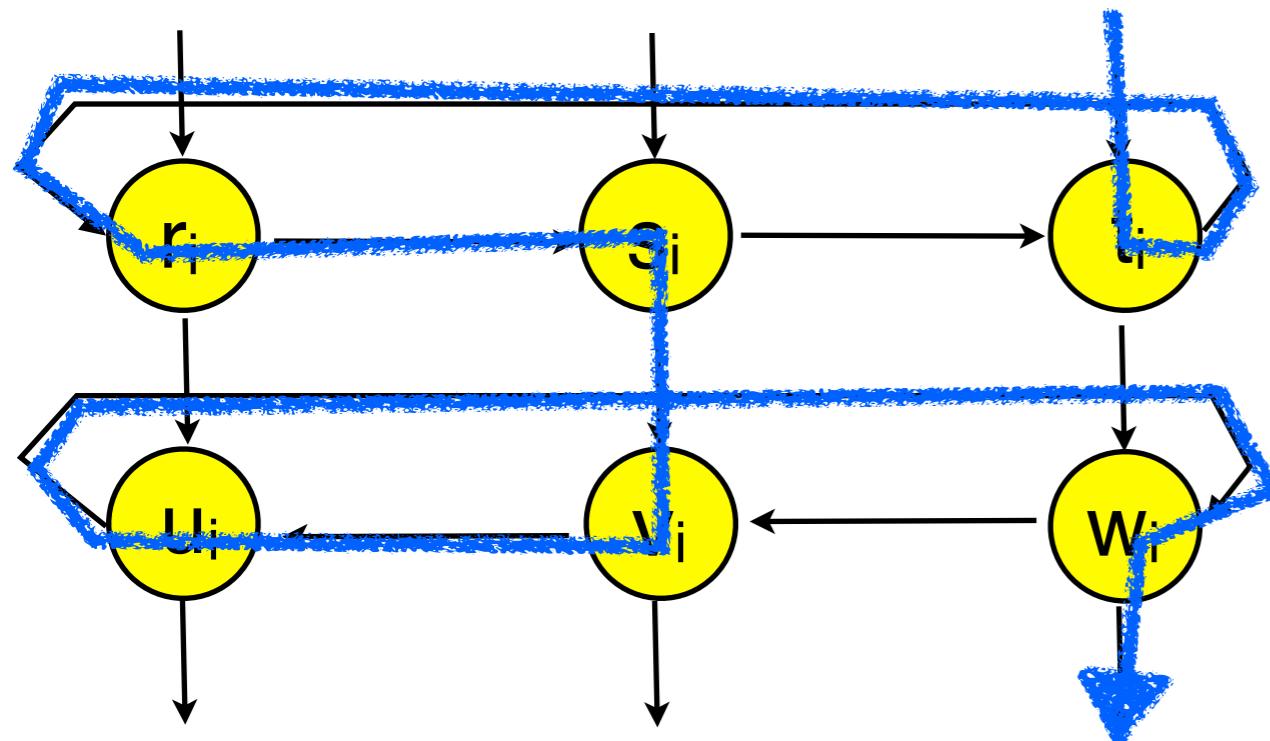
• Wir verbinden jetzt die Bausteine:

• Wenn das Literal A_i unnegiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" **b-Knoten** aus dem A_i -Graph zu r_i und von u_i zu dem **c-Knoten**, der diagonal unter dem **b-Knoten** liegt.

• Umgekehrt: Wenn das Atom A_i negiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" **c-Knoten** zu r_i und von u_i zu dem **b-Knoten**, der diagonal unter dem **c-Knoten** liegt.

Hamilton-Kreis Problem IV

2. Baustein: Für jede Klausel $K_i = (A_i \vee B_i \vee C_i)$ erzeugen wir den Teilgraph:



- Wenn der Teilgraph in einem Kreis vorkommt, dann wird er oben betreten und unten verlassen.
- Es gibt genau drei Wege:
 r_i/u_i , s_i/v_i und t_i/w_i

• Wir verbinden jetzt die Bausteine:

• Wenn das Literal A_i unnegiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" **b-Knoten** aus dem A_i -Graph zu r_i und von u_i zu dem **c-Knoten**, der diagonal unter dem b-Knoten liegt.

• Umgekehrt: Wenn das Atom A_i negiert auftritt, dann ziehen wir eine Kante von dem ersten "freien" **c-Knoten** zu r_i und von u_i zu dem **b-Knoten**, der diagonal unter dem c-Knoten liegt.

• Analog für die Literale B_i und C_i .

Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$

A

$(A \vee B \vee \neg C)$

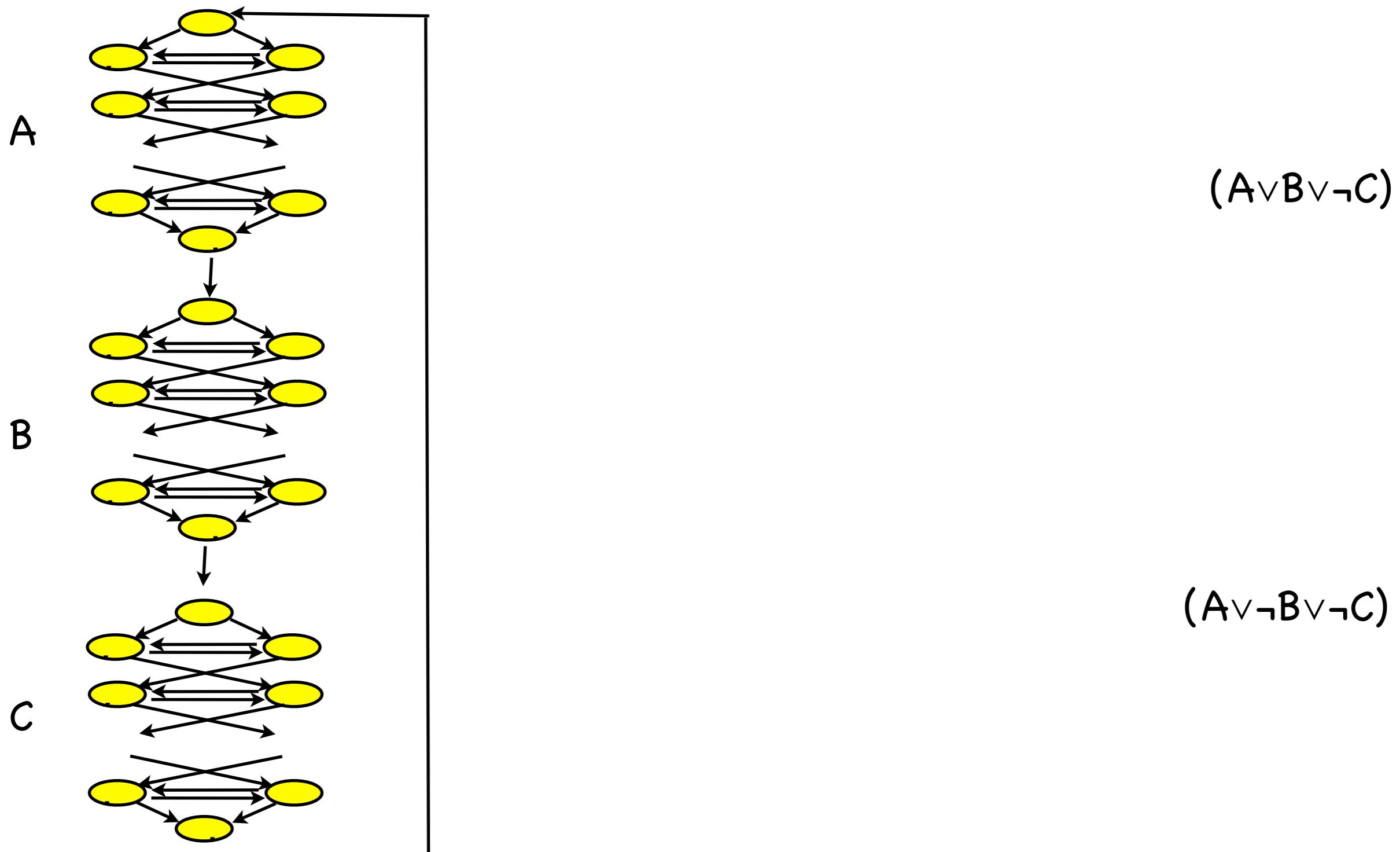
B

$(A \vee \neg B \vee \neg C)$

C

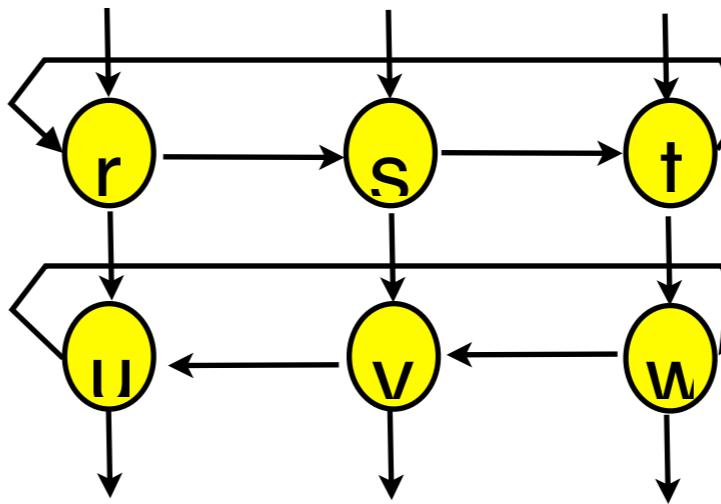
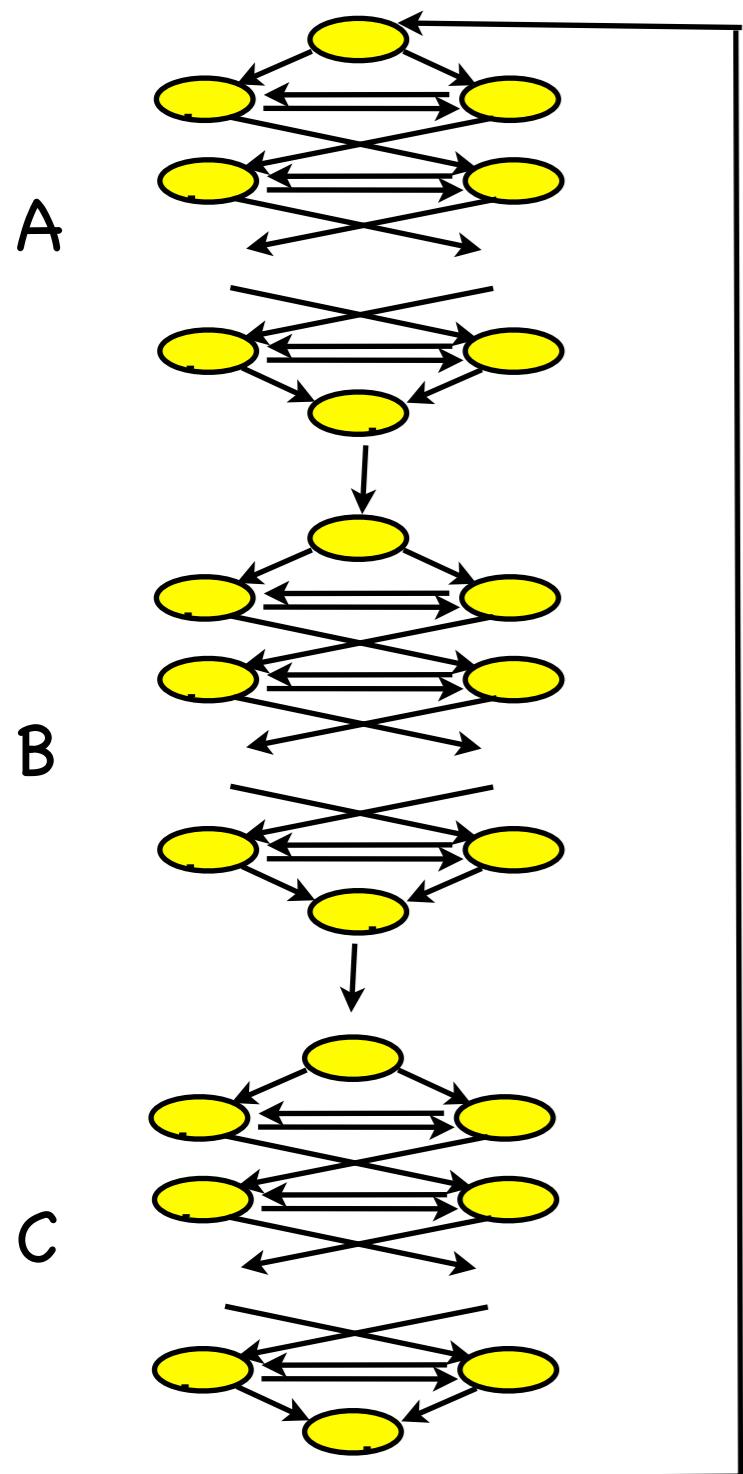
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$

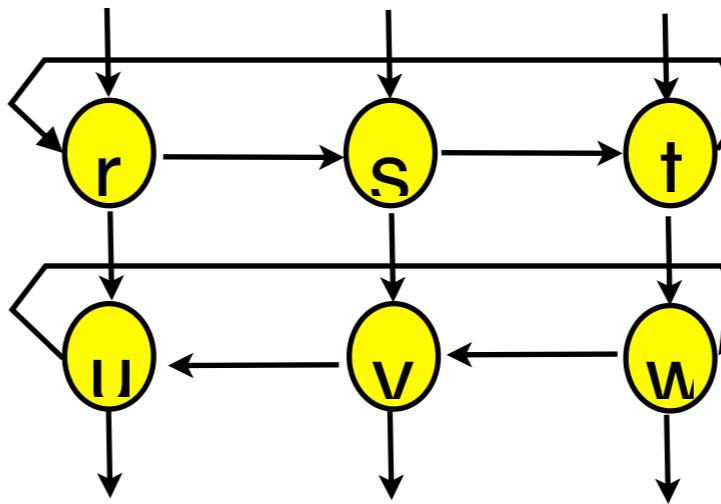
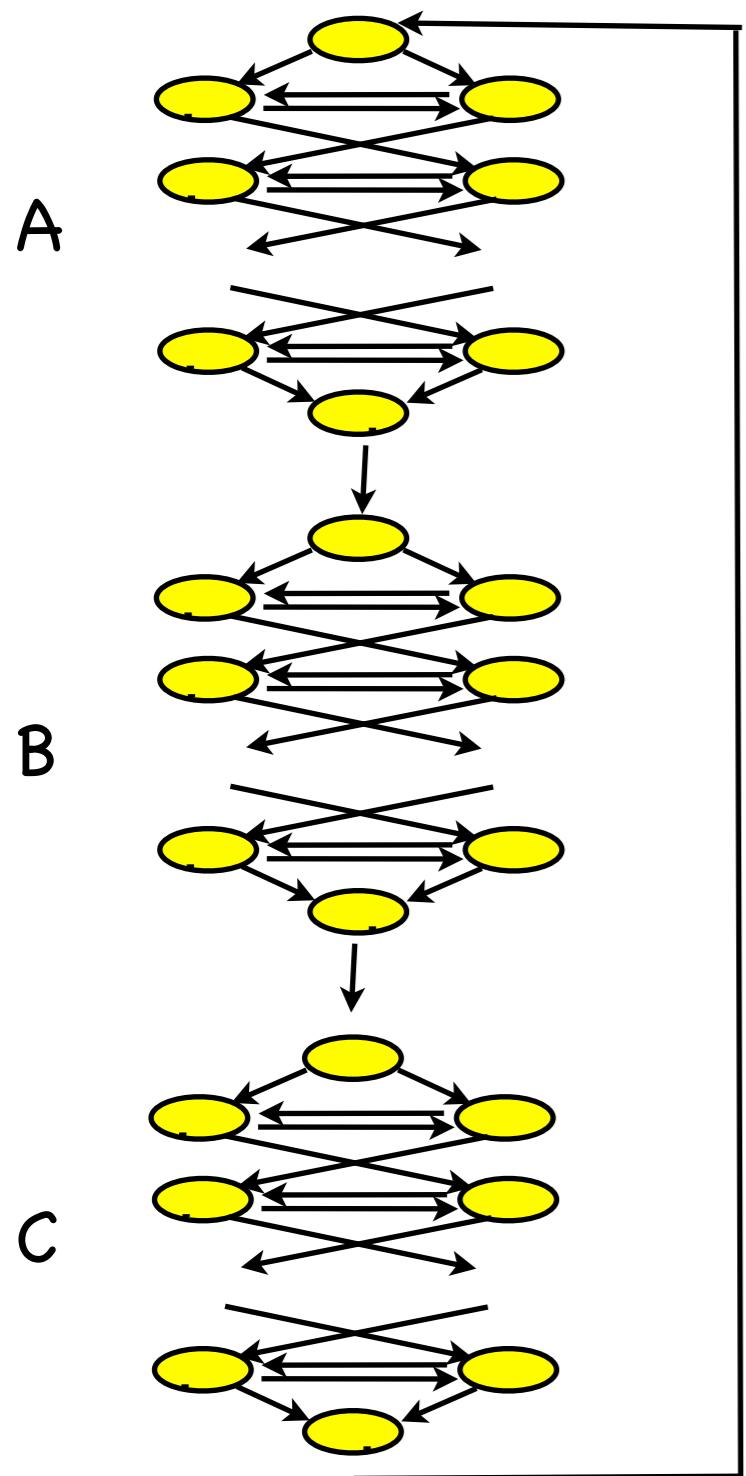


$$(A \vee B \vee \neg C)$$

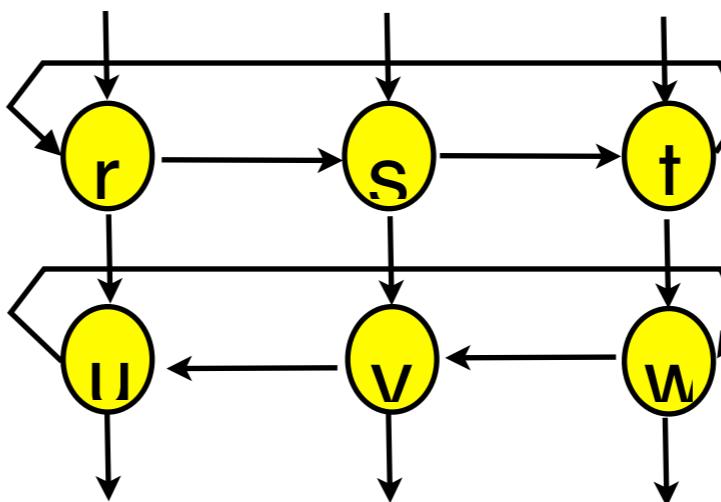
$$(A \vee \neg B \vee \neg C)$$

Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



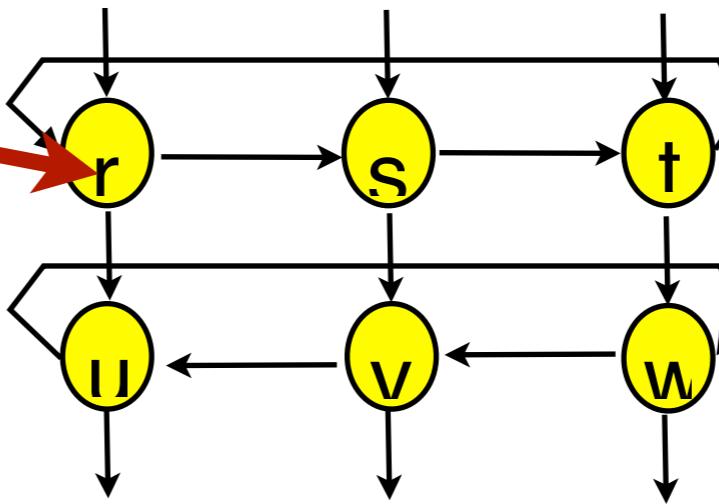
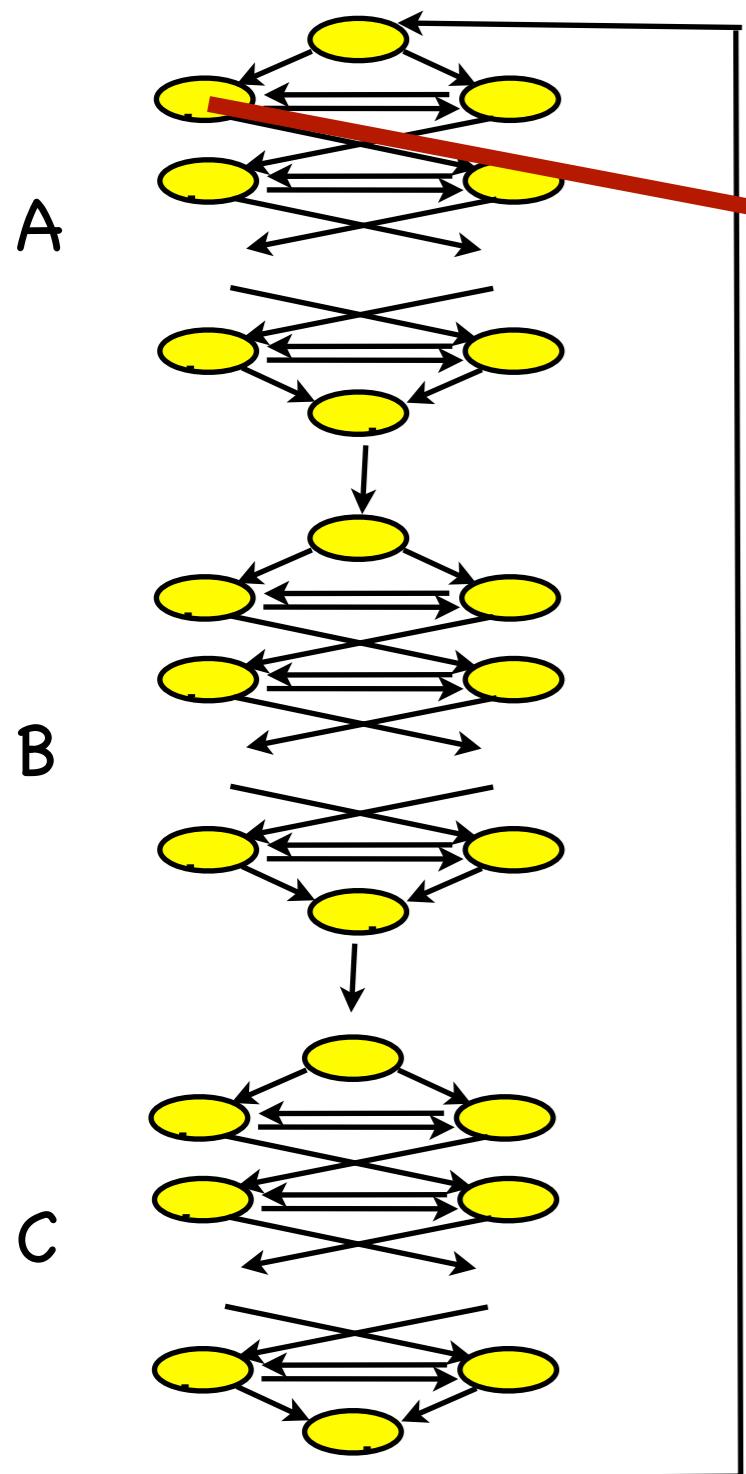
$$(A \vee B \vee \neg C)$$



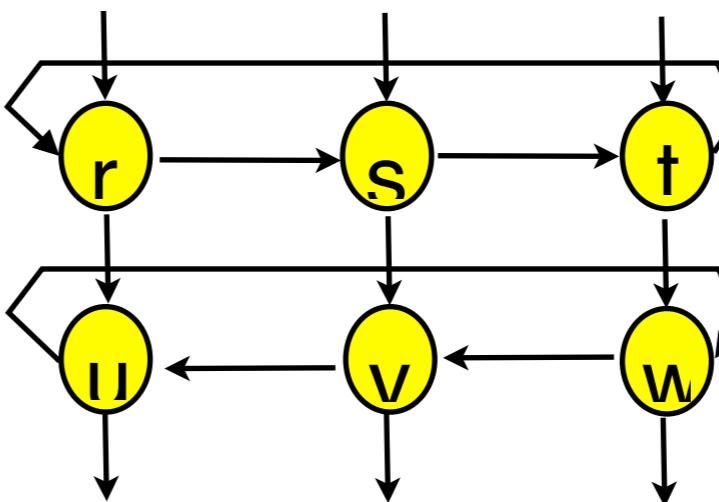
$$(A \vee \neg B \vee \neg C)$$

Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



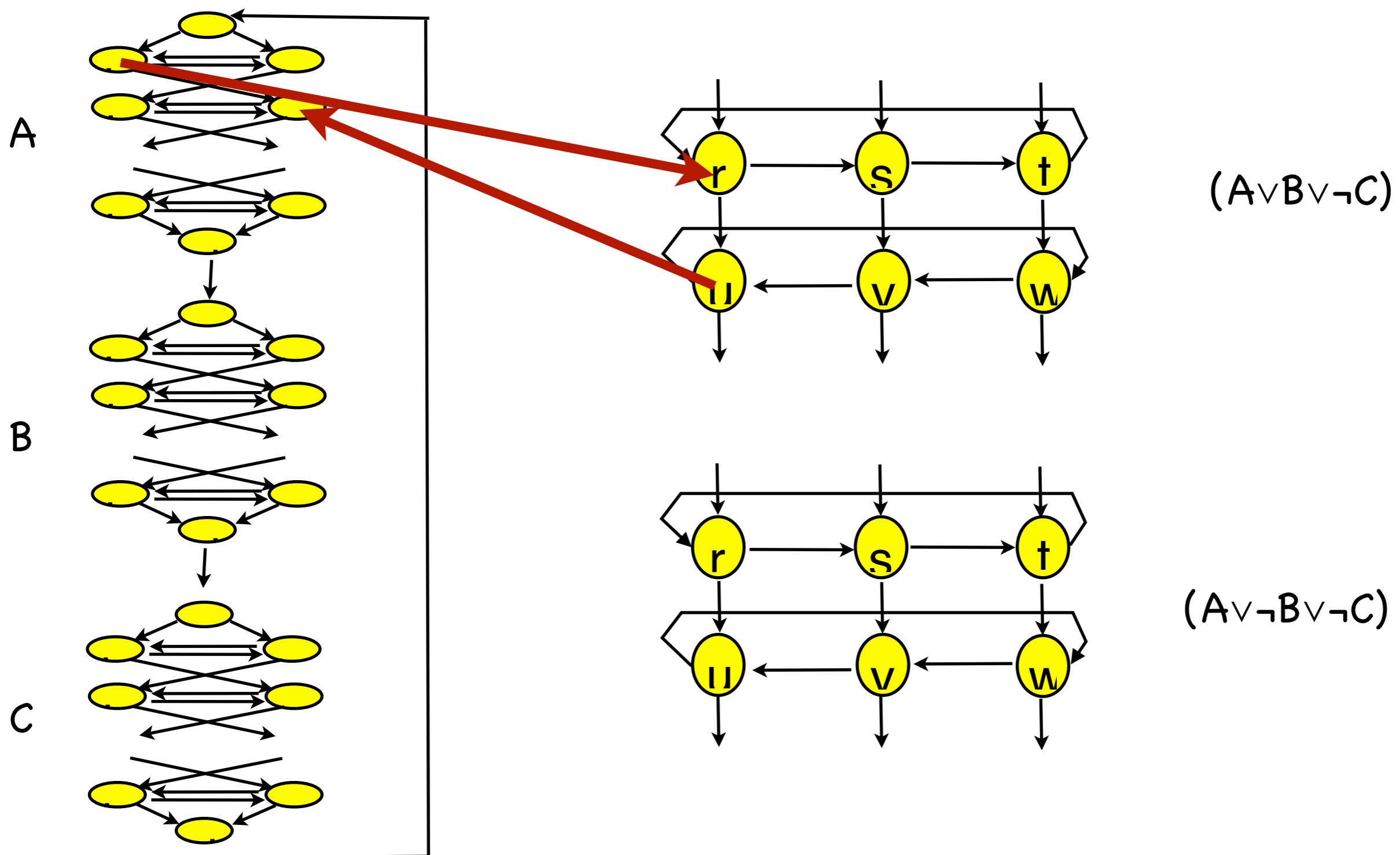
$$(A \vee B \vee \neg C)$$



$$(A \vee \neg B \vee \neg C)$$

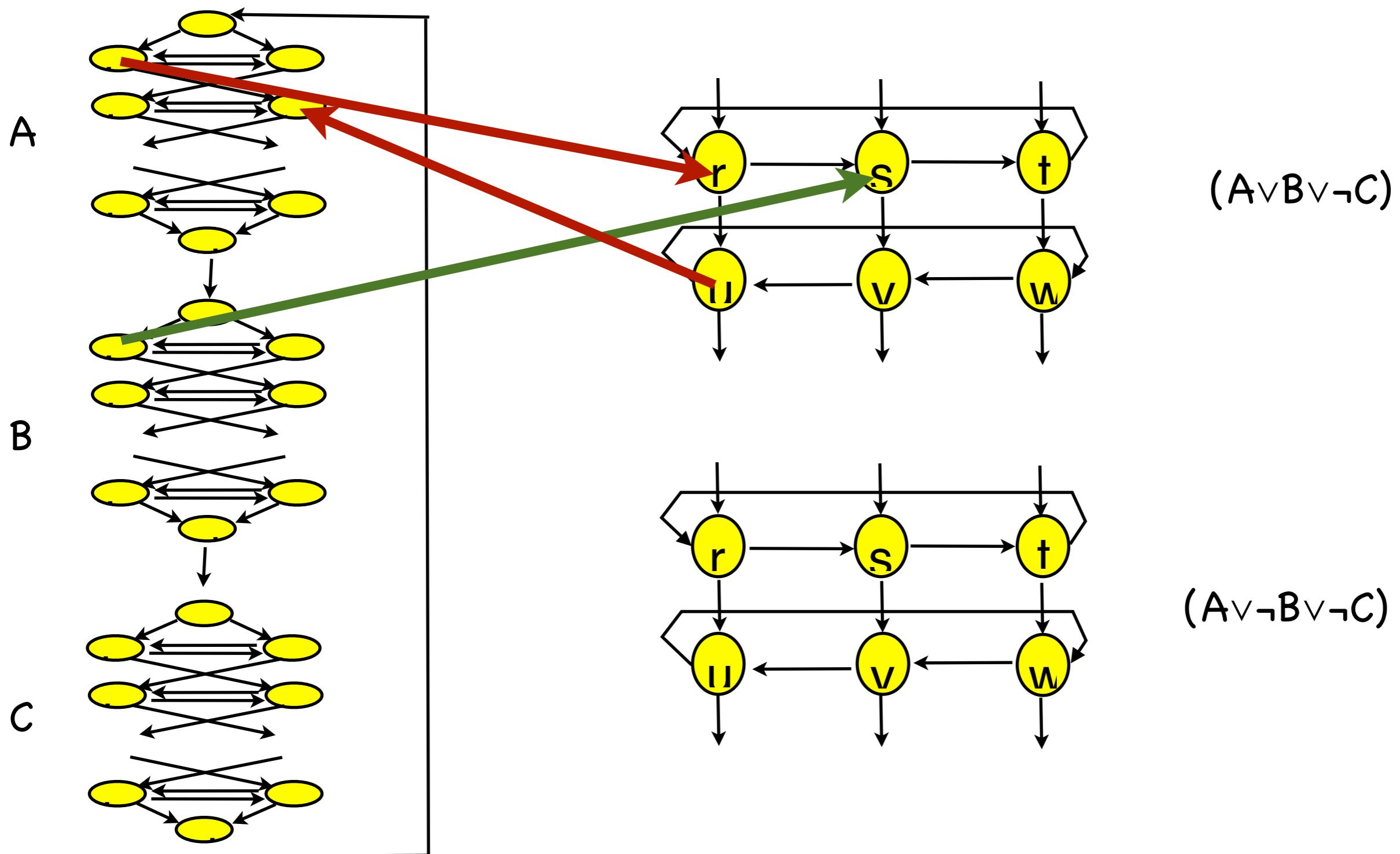
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



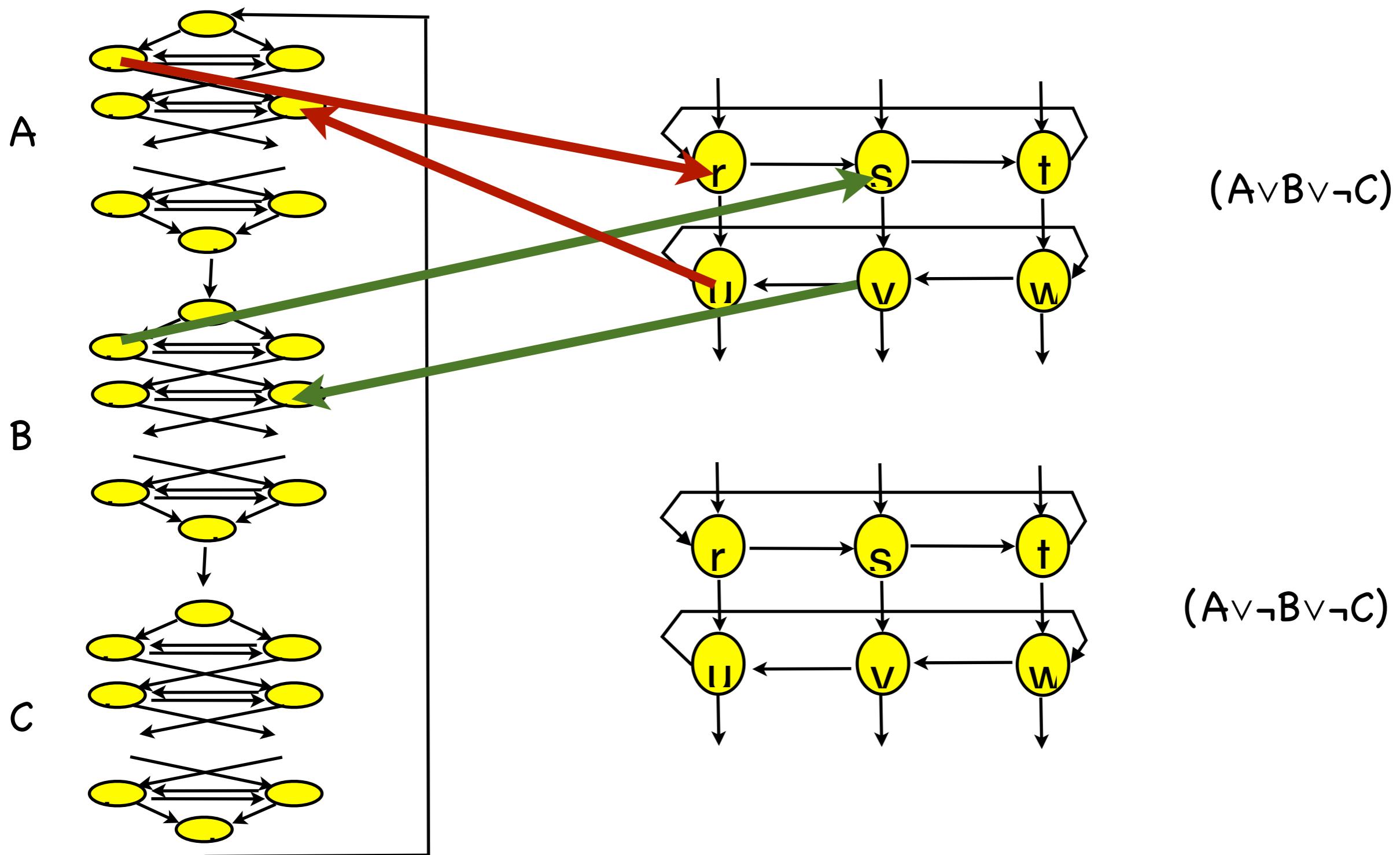
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



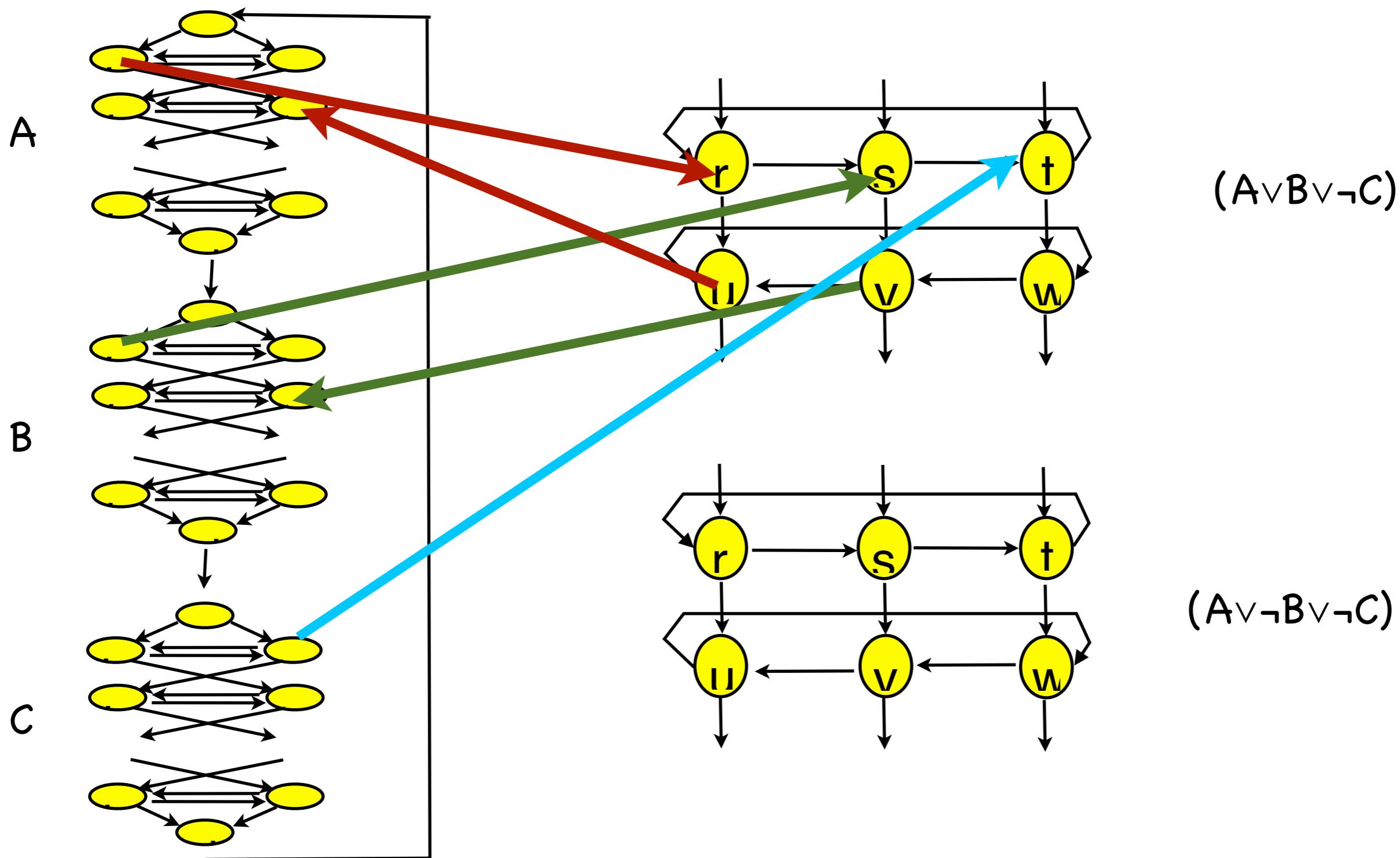
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



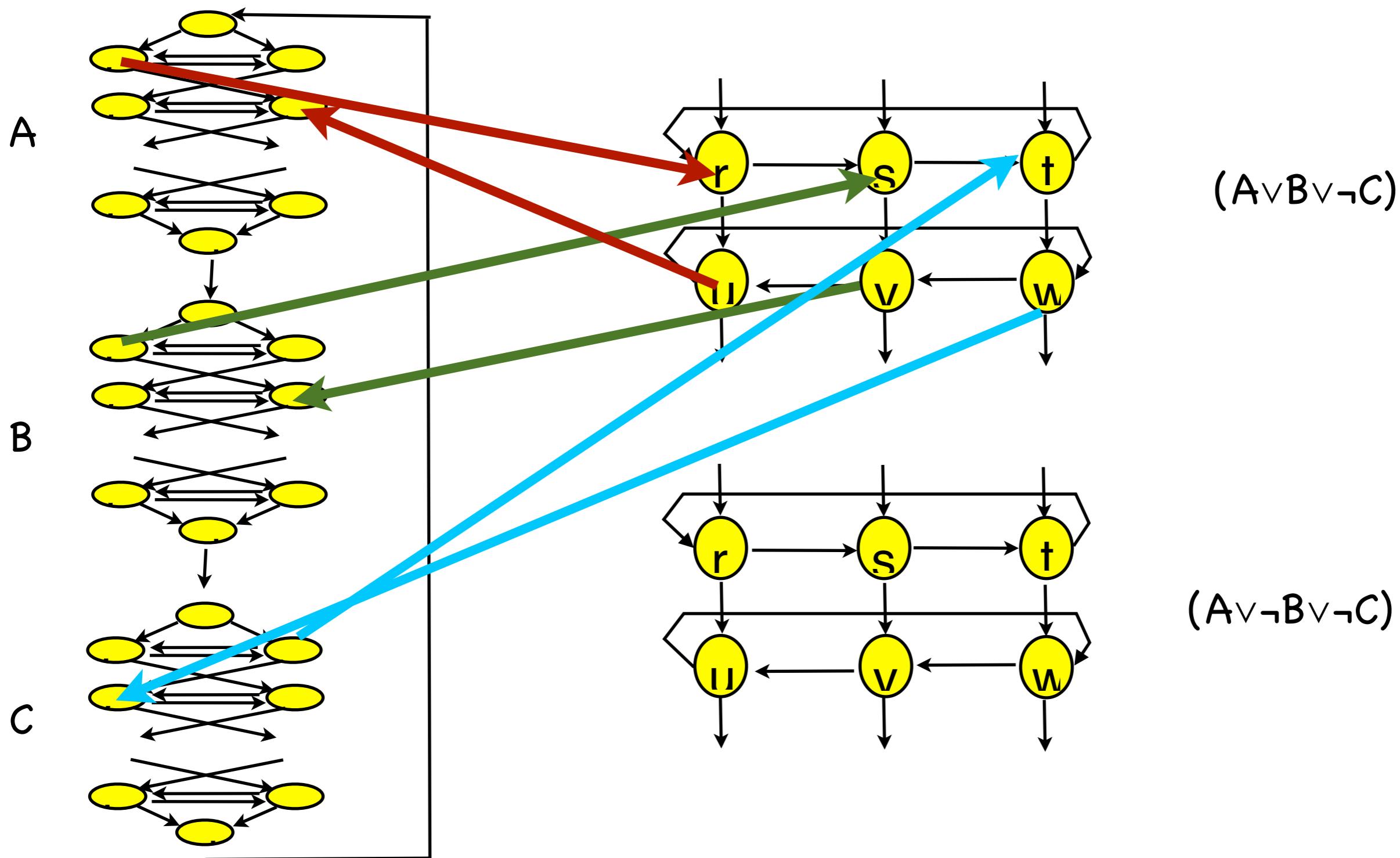
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



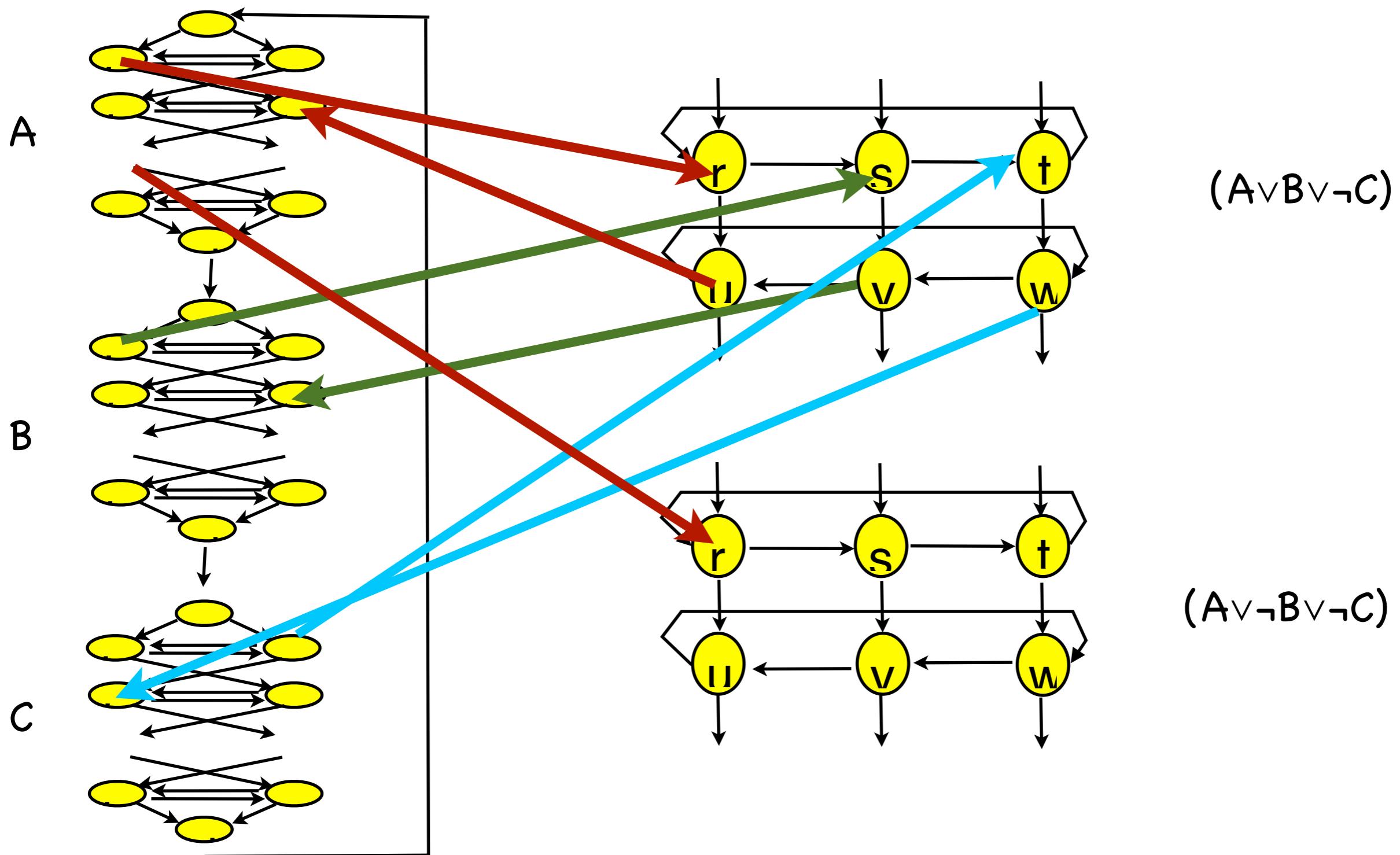
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



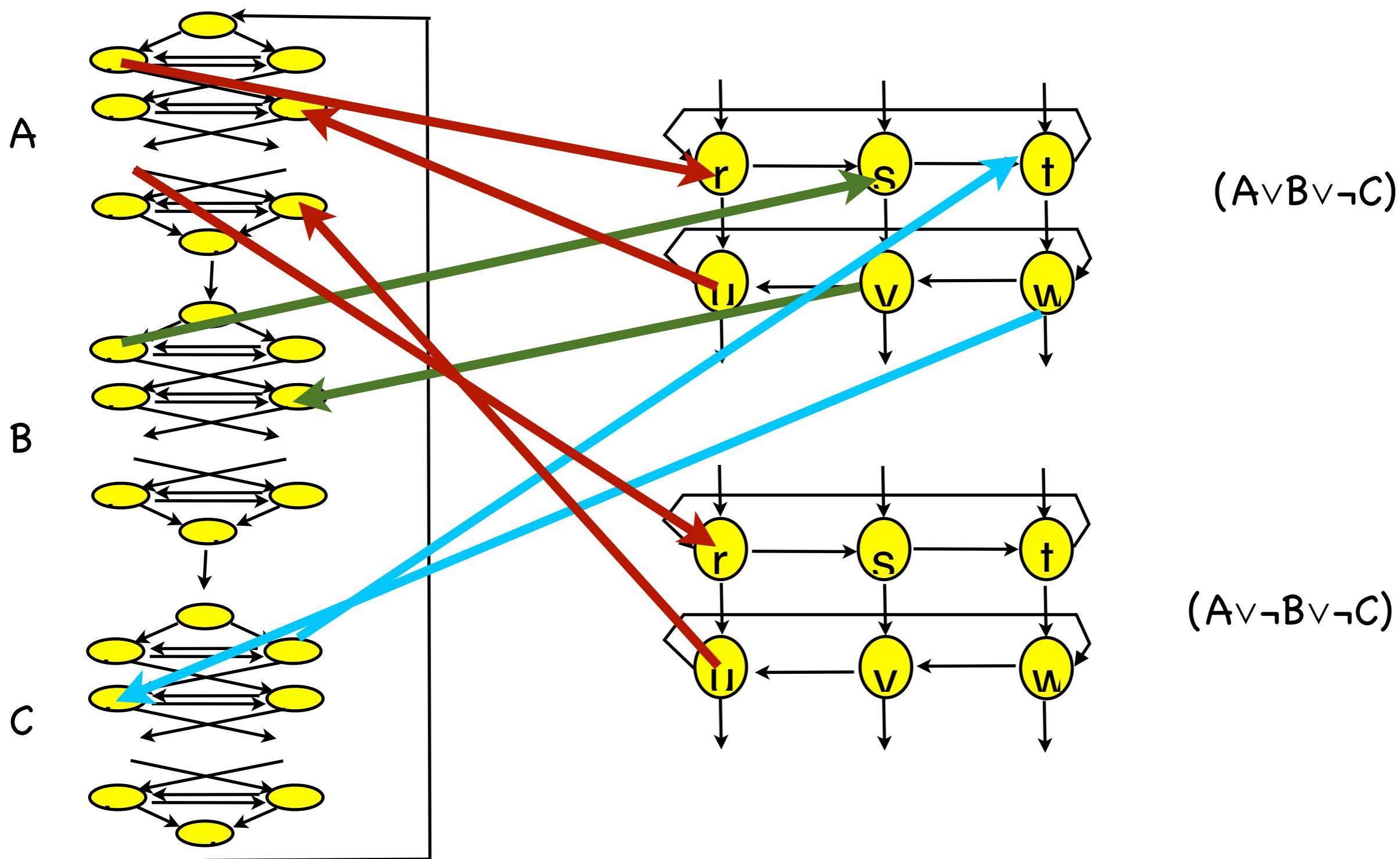
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



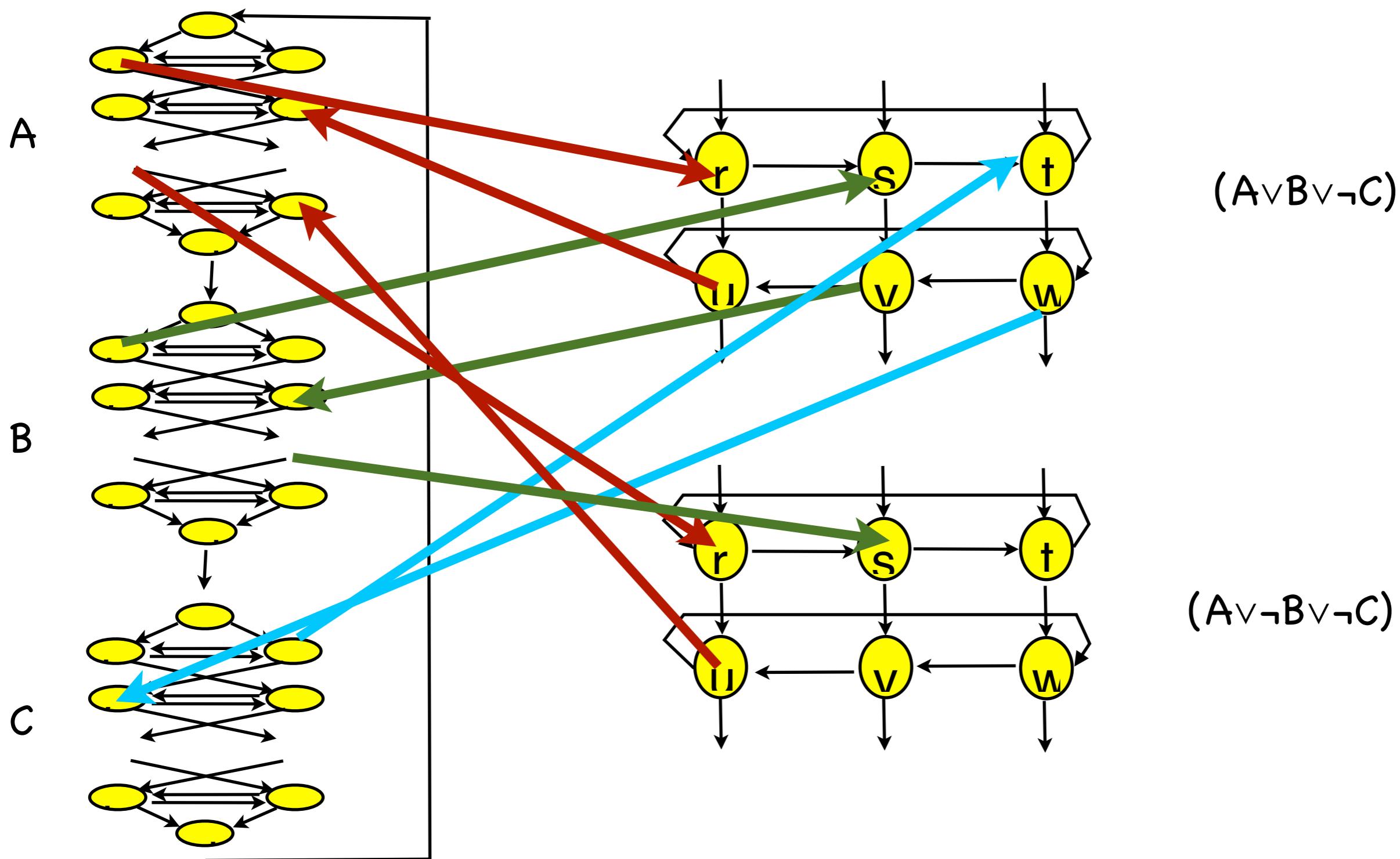
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



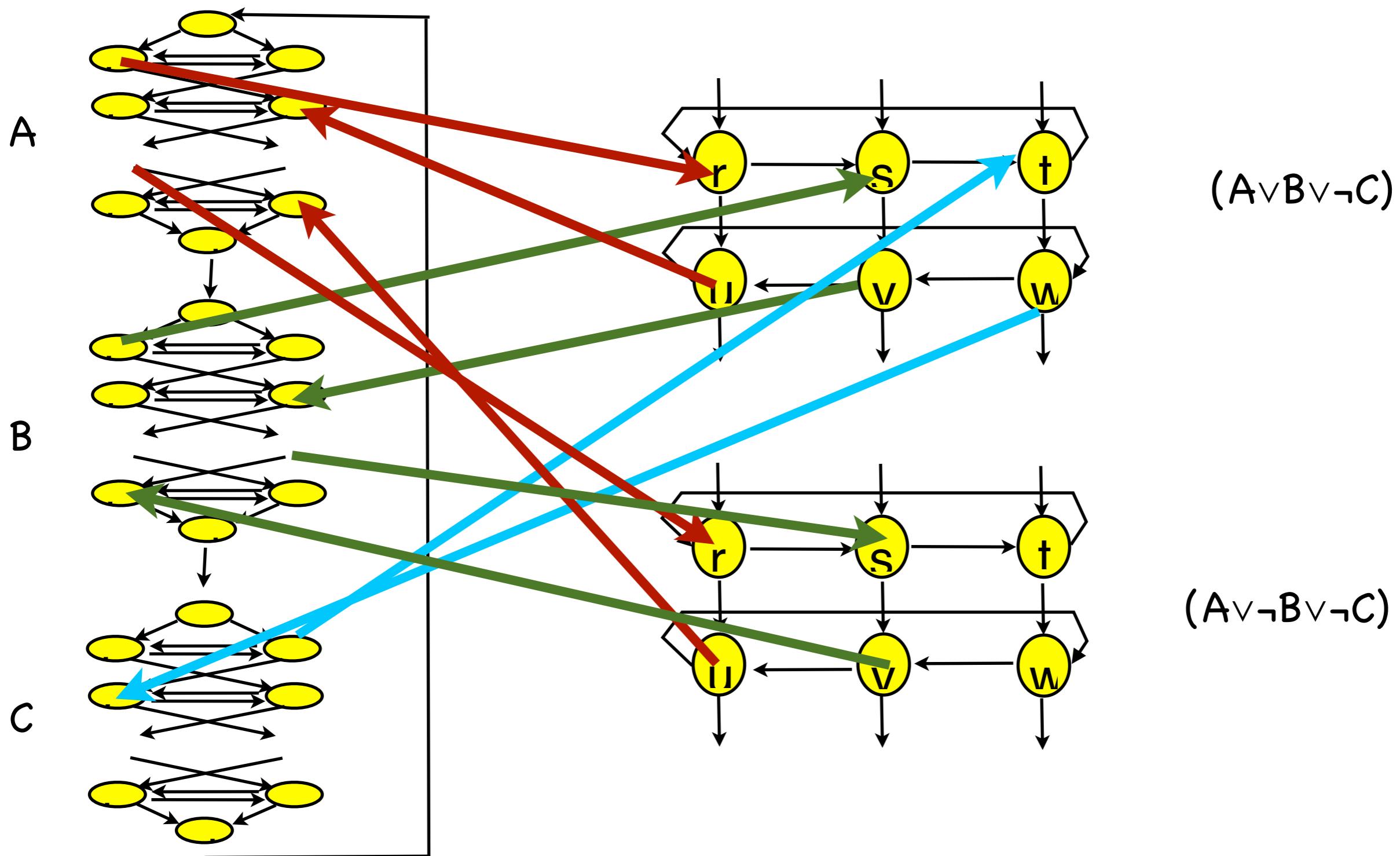
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$

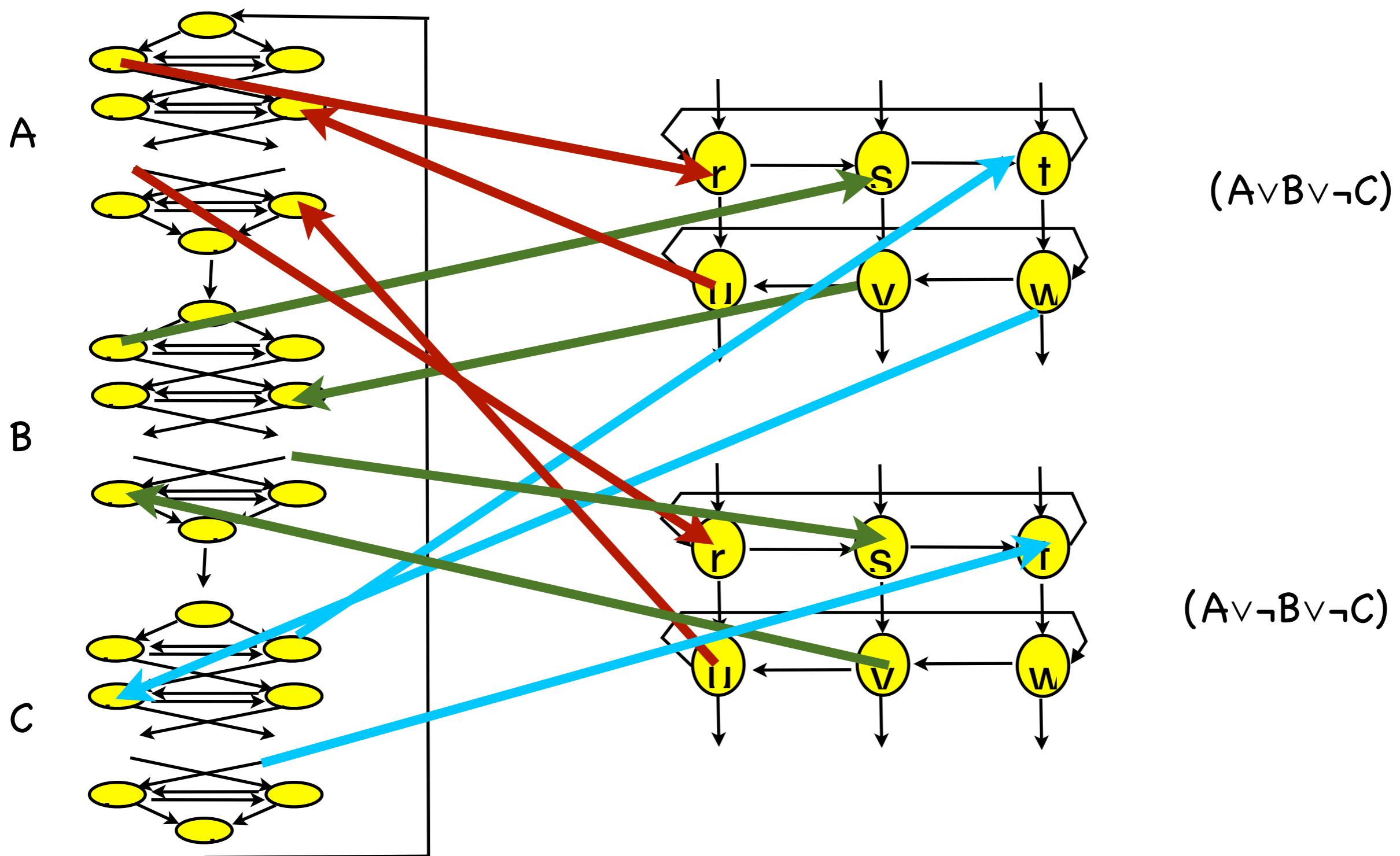


$$(A \vee B \vee \neg C)$$

$$(A \vee \neg B \vee \neg C)$$

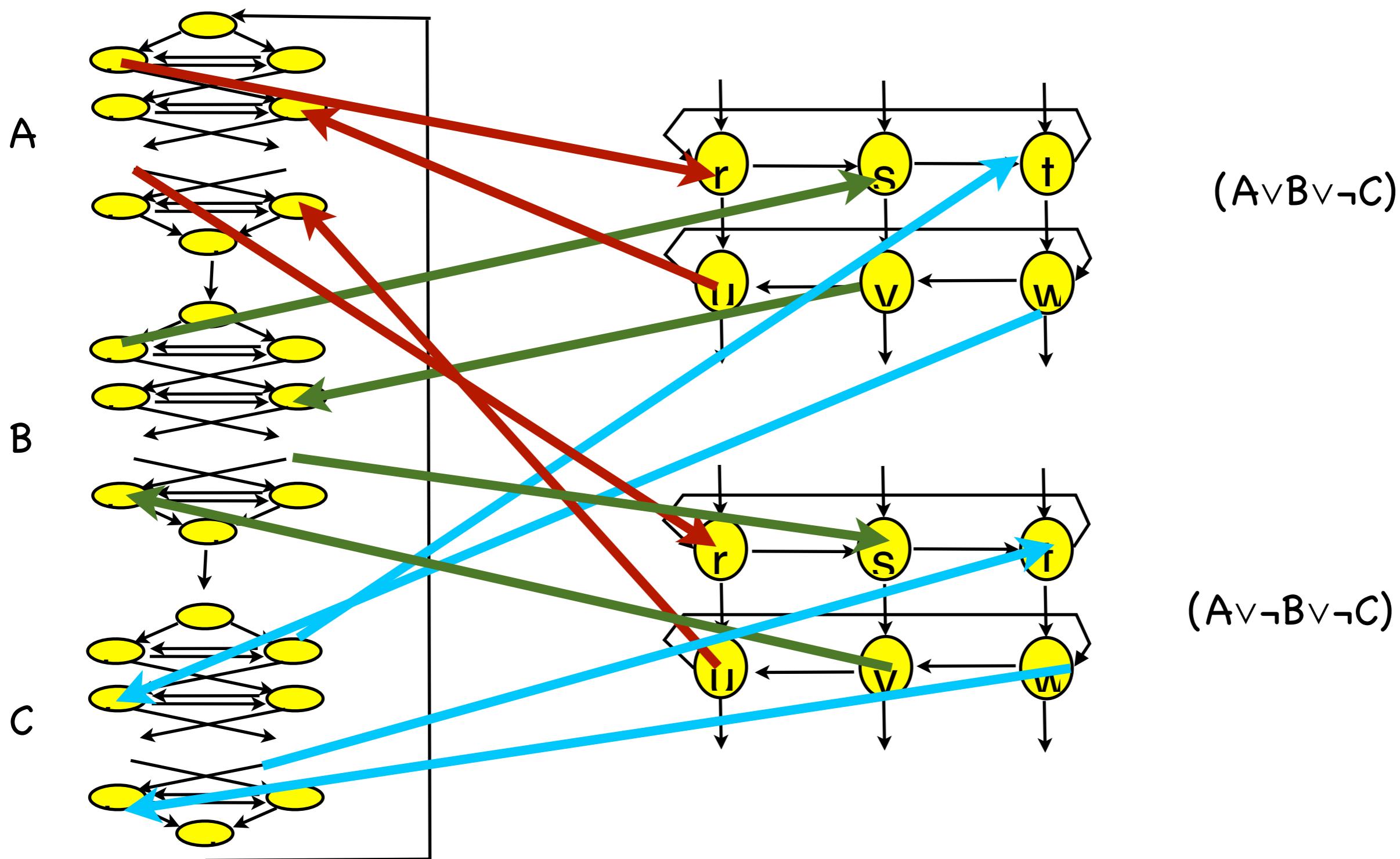
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



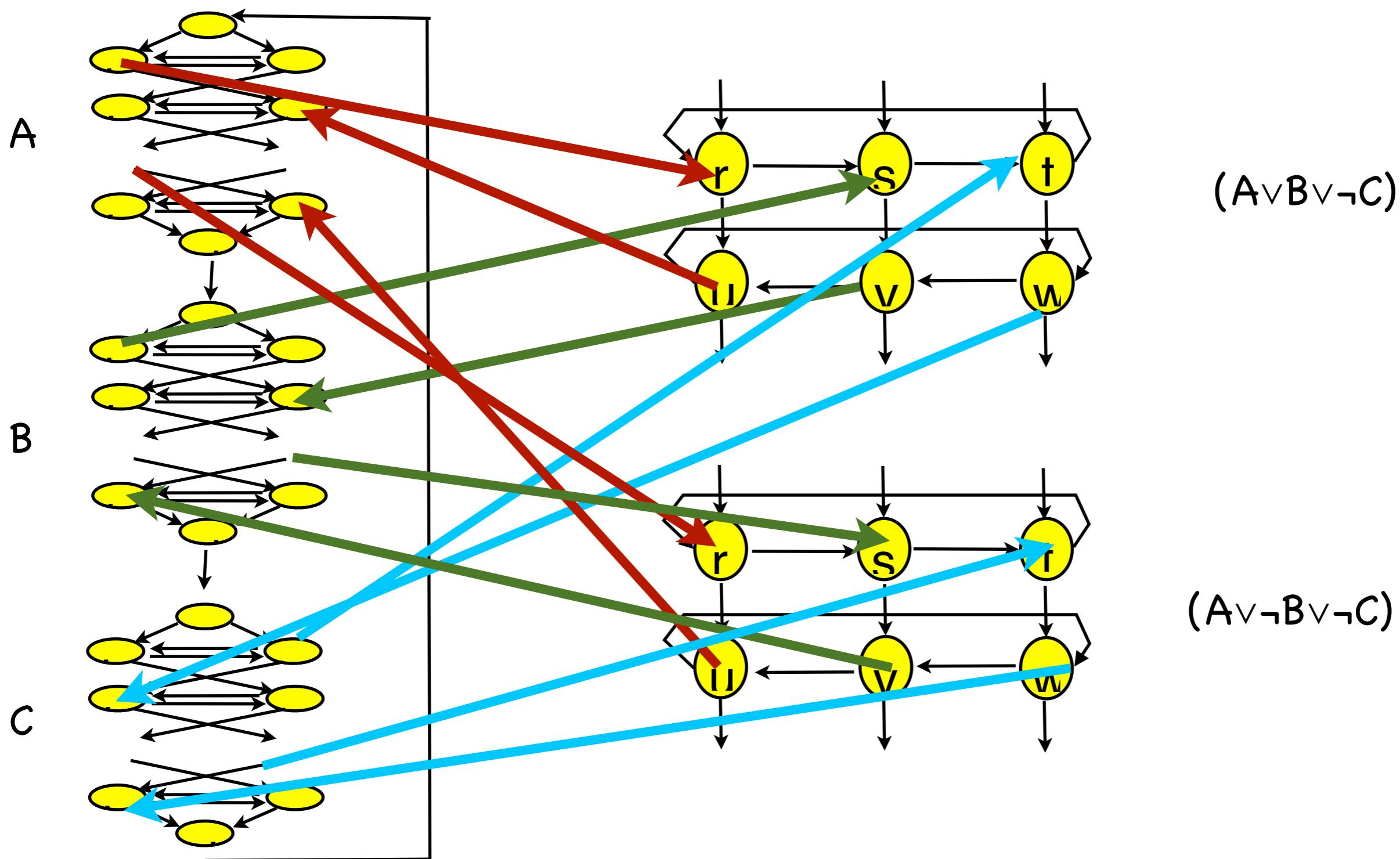
Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



Das Hamilton-Kreis-Problem V

Beispiel: $F = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$



Graph hat HC gdw. F erfüllbar ist. qed.

weitere NP-vollständige Probleme:

Einige *NP*-vollständige Probleme:

- Das Hamilton-Kreis Problem H_c ist *NP*-vollständig.
- Das Problem L_w („Längster Weg zwischen zwei Knoten“) ist *NP*-vollständig.
- Das Rucksackproblem ist *NP*-vollständig.
- Das Partitionierungsproblem von Graphen ist *NP*-vollständig.
- Das Problem *CLIQUE* ist *NP*-vollständig.

vollständige Probleme für andere Sprachfamilien

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{P}Space = \mathcal{NP}Space$$

Und was kommt dann?

“Wo wollen sie hin; was ist der Sinn und was kommt dann?”

[Zitat ohne Fußnote]

vollständige Probleme für andere Sprachfamilien

$$\mathcal{P} \underset{?}{\subseteq} \mathcal{NP} \subseteq \mathcal{P}Space = \mathcal{NP}Space$$

Und was kommt dann?

Es gibt auch für andere Sprachklassen, wie \mathcal{P} , $\mathcal{P}Space$ und viele weitere Komplexitätsklassen vollständige Probleme!

Das Studium dieser Zusammenhänge ist Thema von Vertiefungsveranstaltungen zur (strukturellen) Komplexitätstheorie und erst im Masterstudium von Bedeutung.

“Wo wollen sie hin; was ist der Sinn und was kommt dann?”

[Zitat ohne Fußnote]