Theoretische Informatik

Hochschule Darmstadt, Sommersemester 2017

Bernd Baumgarten
(Lehrbeauftragter)

Der Großteil der Folieninhalte ist dankend
übernommen von Prof. Steffen Lange, h_da

Organisatorisches

- Vorlesung
  - zwei Vorlesungen pro Woche, ab 11.4.2017
    aber drei in der Anfangswoche, da auch in den Ü-Blocken 11./12.4.17
  - Folien im Netz *http://www.bernd-baumgarten.de*
- Übung
  - Übungsaufgaben im Netz *http://www.bernd-baumgarten.de*
    meist am Mittwochabend, ab 11.4.
  - ein Übungsblock pro Woche, erstmals 18./19.4.
  - Studierende stellen darin ihre Lösungen vor und erhalten Feedback.
  - Fragen zur Vorlesung können geklärt werden.
- Lösungen, schriftl. Kommentare & Fragen *baumgarten_bernd@web.de*

Organisatorisches

Prüfungsvorleistung (PVL)

- Abgabe der Lösungen für das aktuelle Übungsblatt
  - Termin: jeweils bis zum nächsten Montag, 14:00 Uhr
  - Wie: Word/txt/pdf-File (keine schummrigen Fotos) an *baumgarten_bernd@web.de*

- Die PVL ist erbracht, wenn Sie …

  jede Woche mindestens eine Aufgabenlösung für das aktuelle Übungsblatt abgegeben haben und dabei
  - zu Semesterende *in Summe ≥ 50%* der Aufgaben korrekt gelöst haben – es wird aber nur stichprobenweise korrigiert – und
  - im Rahmen der Übungsstunden gezeigt haben, dass Sie *mindestens eine Aufgabe* korrekt lösen und erklären können.

Literatur

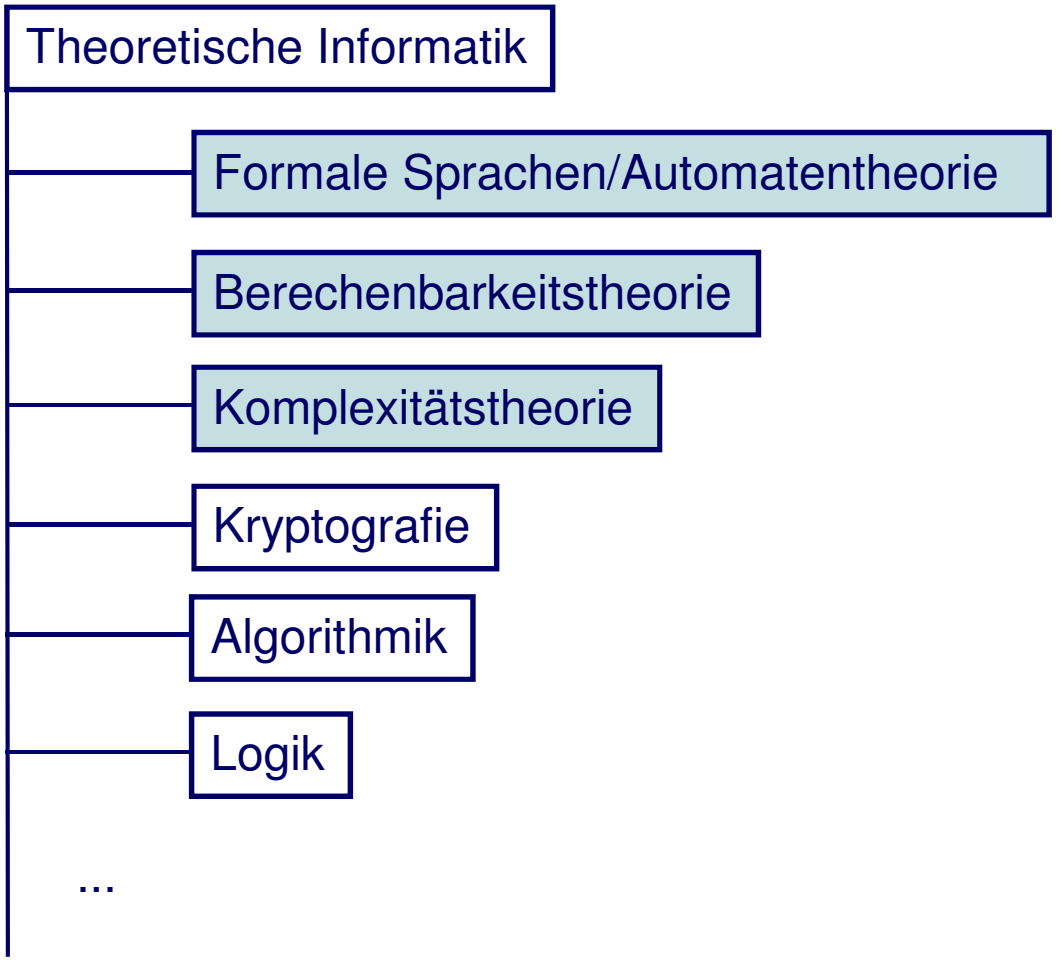
- S. Lange, M. Margraf, Theoretische Informatik, Lehrmaterial für den Bachelorstudiengang Informatik/IT-Sicherheit, 2016. (online verfügbar)
- K. Erk, L. Priese: Theoretische Informatik: Eine umfassende Einführung, Springer Verlag, 2008
- U. Hedtstück: Einführung in die Theoretische Informatik, Oldenbourg Verlag, München, 2000.
- J.E. Hopcroft, R. Motwani, J.E. Ullmann: Einführung in die Automaten-theorie, formale Sprachen und Komplexitätstheorie, Addison Wesley, Bonn, 2002.
- J. Hromkovic: Theoretische Informatik, Teubner Verlag, Stuttgart, 2002.
- U. Schöning: Theoretische Informatik – kurz gefasst, Spektrum Akademischer Verlag, Heidelberg, 1997.

Gliederung der Vorlesung

Ziele der theoretischen Informatik

- zentrales Ziel ...
  - Analyse der in der Informatik auftretenden Probleme und Strukturen mit mathematischen Methoden
- einige wichtige Fragestellungen
  - Wie kann man in der Informatik auftretende **Probleme** sinnvoll **modellieren/formalisieren**?
  - Welche **Algorithmen** zur Lösung eines Problems gibt es?
  - Gibt es Probleme, die sich **nicht** algorithmisch lösen lassen **bzw. nicht effizient lösen** lassen?
  - Wie kann man Algorithmen mit Blick auf die Aspekte **Korrektheit** und **Effizienz** analysieren?
  - Wie lässt sich der algorithmische **Aufwand** zur Lösung von Problemen minimieren?

Teilgebiete der Theoretischen Informatik



- typische Fragestellungen aus dem Gebiet Berechenbarkeit
  - Welche Probleme sind überhaupt mit Hilfe eines Computerprogramms lösbar?
  - Von welchen Problemen sollte man dagegen die Finger lassen?

Geben Sie einem Computer die richtige Software und er wird tun, was immer Sie wünschen. Die Maschine selbst mag ihre Grenzen haben, doch für die Möglichkeiten von Software gibt es keine Grenzen.

Time-Magazine, 1984

- typische Fragestellungen aus dem Gebiet Komplexität

  - Anhand welcher Kriterien beurteilt man die Güte von Algorithmen / Programmen zur Lösung eines Problems?

    Der Begriff Güte bezieht sich auf den (möglichst geringen) Verbrauch an Ressourcen (Rechenzeit- bzw. Speicherplatz).

  - Von welcher Güte können Algorithmen / Programme zur Lösung eines Problems sein?

  - Kann man einem Problem ansehen, ob es einen effizienten Algorithmus / ein effizientes Programm zu seiner Lösung gibt?

    Effiziente Algorithmen / Programme haben eine Rechenzeit, die nur polynomiell von der Größe der Eingaben abhängt.

Gliederung der Vorlesung

Problemreduktion

- eine zentrale Herangehensweise: Problemreduktion
  - unterschiedliche Probleme miteinander in Beziehung setzen, nämlich
    - ein „beherrschtes“ Problem (Typ 1), d.h. man weiß, wie man dieses Problem lösen kann, und
    - ein „neues“ Problem (Typ 2), d.h. man weiß noch nicht, wie man dieses Problem lösen kann.

... mit dem Ziel, das Wissen oder die Mittel zur Lösung des „beherrschten“ Problems 1 zu benutzen, um das „neue“ Problem 2 zu lösen

Problemreduktion

- ein einführendes (älteres) Scherz-Beispiel

Wie fängt man einen Helmtiger?

- Man geht in den Urwald und läuft dort umher, bis man von einem Helmtiger verfolgt wird.
- Dem läuft man so lange davon, bis es dem Helmtiger vor Hitze, Luftfeuchtigkeit und Anstrengung so heiß wird, dass er den Helm ablegt, um den Schweiß abzuwischen.
- Dann fängt man ihn einfach wie einen gewöhnlichen Tiger …

Problemtyp 2: Helmtiger fangen

Problemtyp 1: gewöhnlichen Tiger fangen

Problemreduktion

- ein einführendes Alltags-Beispiel

Auf dem Weg nach Hause haben Sie eine Autopanne, die zwar im Prinzip reparabel ist, sich aber weder mit Ihrem mitgeführten Werkzeug bzw. Ihren Fähigkeiten noch mithilfe des Pannendienstes an Ort und Stelle beheben lässt.

- Sie lassen sich in eine geeignete Werkstatt abschleppen.
- Dort wird der Wagen repariert.
- Dann fahren Sie weiter nach Hause.

Problemtyp 2: unterwegs aufgetretene Autopanne beheben

Problemtyp 1: Auto in Werkstatt reparieren

Aspekte:

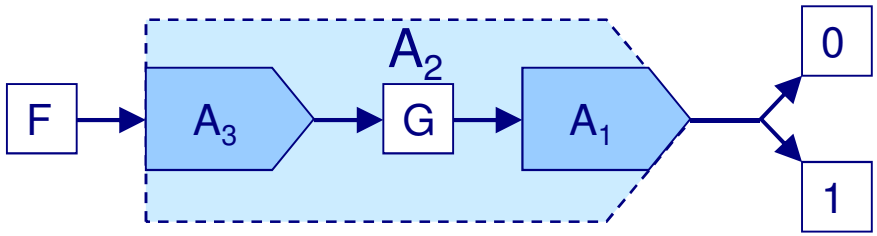
Algorithmen		Autofahren
Berechenbarkeit	←→	Machbarkeit überhaupt
Komplexität	←→	Aufwand (Zeit, Geld, Rohstoffe)

- Algorithmus A_2 für das neue Typ-2-Problem F (prinzipielle Idee)

  - Es sei A_1 ein Algorithmus (ein präzises Rezept) zur Lösung des beherrschten Typ-1-Problems.
    Hier ein Entscheidungsproblem (Ergebnis 0/1, ja/nein, …)

  - Es sei A_3 ein Algorithmus zur Umwandlung eines 2-Problems in ein „gleichwertiges“ Typ-1-Problem

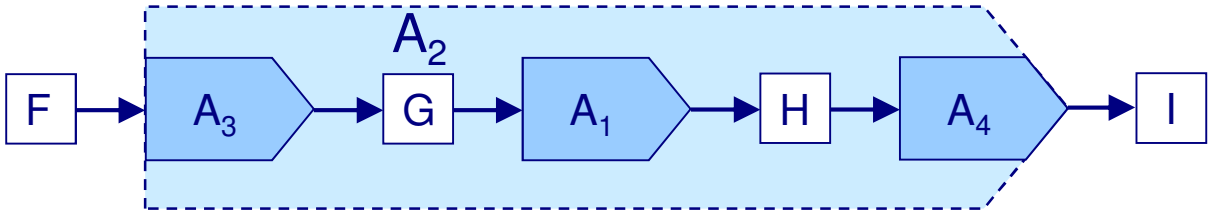
Dann ist auch jedes Problem F des Typs 2 zu lösen:



Reduktion: A3 bildet die A2-Eingaben so in die A1-Eingaben ab, dass es jede von A2 „zu akzeptierende“ (Ausgabe=1) bzw. nicht zu akzeptierende Eingabe in eine von A1 „akzeptierte“ bzw. nicht akzeptierte verwandelt.

Problemreduktion

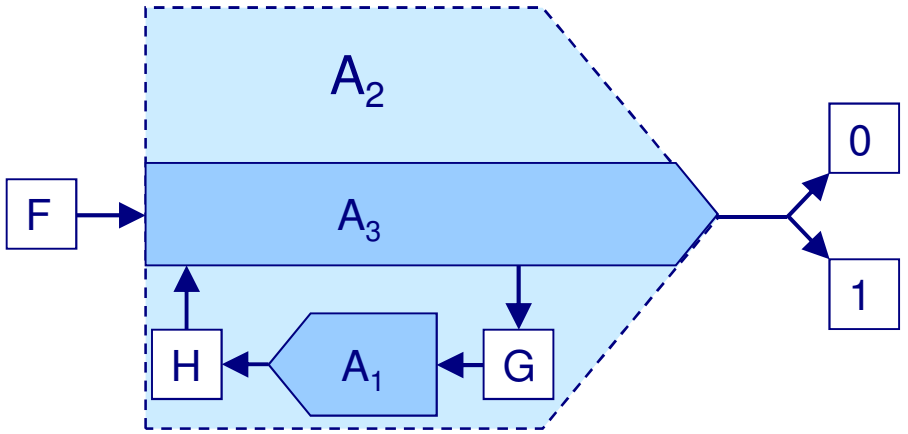
- OK, manchmal ist es etwas komplizierter, z.B. so (Ergebnis nicht nur 0/1):



- Nicht nur ist die Eingabe F für A_2 in eine entsprechende Eingabe G für A_1 umzuwandeln (das erledigt A_3),
- sondern auch die Ausgabe H von A_1 in die entsprechende Ausgabe I von A_2 (das erledigt A_4).

- Und das gewünschte Ergebnis I ist nicht unbedingt nur 0 oder 1.

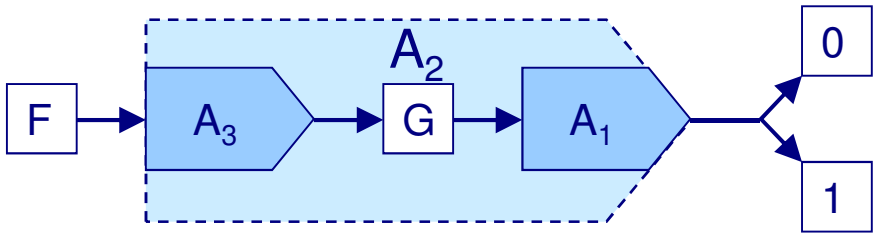
- Oder so (Ergebnis hier z.B. 0/1):
  - Vielleicht wird innerhalb von A_2 das Verfahren A_1 sogar mehrfach durchlaufen – wie oft, kann von der Eingabe F abhängen.



Komplexitätsaspekt

- Es sei A_1 ein **effizienter** Algorithmus zur Lösung des beherrschten Typ-1-Problems
- Es sei A_3 ein **effizienter** Algorithmus zur Umwandlung eines Typ-2-Problems in ein „gleichwertiges“Typ-1-Problem

Dann ist auch ein Problem des Typs 2 **effizient** zu lösen:



(hier: ein Entscheidungsproblem: 0/1)

Was verstehen wir genau unter effizient? → Komplexitätstheorie

Problemreduktion – Listen

- ein Beispiel aus der Informatik (1) – „beherrschtes“ Problem – Typ 1

... zwei Wörter lexikalisch miteinander vergleichen
(auf Identität „=“ bzw. Ordnung „<“ oder „>“)

- Beispielsweise wird berechnet
  Vgl(“Anlieferung“,“Anleitung“) = ‘>‘

Algorithmus **A_1:**
- Buchstabenweise vergleichen
- Das Wort mit dem ersten „kleineren“ Buchstaben ist kleiner.
- Wenn aber bei bis dahin gleichen Buchstaben eines der Wörter zu Ende ist:
  - Ist das andere auch zu Ende, dann „=“
  - Ansonsten ist das kürzere Wort „kleiner“.

Problemreduktion – Listen

- Beispiel (1) – ein „neues“ Problem – Typ 2

... ein gegebenes Wort in einer nummerierten und lexikalisch geordneten Liste von Wörtern suchen:

Kommt es darin vor oder nicht?

Ist “Anlieferung“ in der Liste ?

⋮
2731: anleinen
2732: anleiten
2733: Anleitung
2734: anlernen
2735: anlesen
2736: anliefern
2737: Anliegen
⋮

Problemreduktion – Listen

- Beispiel (1) – den Typ-1-Algorithmus für Typ 2 benutzen (Alternativen!)

Algorithmus A_2 (verwendet A_1 wiederholt)
i := 1
Weiter: Vergleiche das Wort mit Listenelement i.
Bei Übereinstimmung gib „ja“ aus und stoppe.
Ansonsten: Hört die Liste bei i auf?
Wenn ja, gib „nein“ aus und stoppe.
Wenn nein, setze i:= i+1 und gehe zu Weiter.

Algorithmus A'_2 (verwendet A_1 wiederholt)
min := 1; max := Länge der Liste
Weiter: Falls Wort < Listenelement min oder > Listenel. max,
gib „nein“ aus und stoppe.
mitte := ganz((max+min)/2).
Vergleiche das Wort mit Listenelement mitte.
Bei ‚=‘ Übereinstimmung gib „ja“ aus und stoppe.
Wenn <, setze max := mitte –1 und gehe zu Weiter.
Wenn >, setze min := mitte +1 und gehe zu Weiter.

Effizienzunterschiede? Gesamtaufwand von A_2?

Problemreduktion – Listen

- ein Beispiel aus der Informatik (2) – „beherrschtes“ Problem – Typ 1

... überprüfen, ob zwei gleich lange Listen L_1 und L_2 von Zeichenketten identisch sind (Ergebnis 1, sonst 0)

Zulässige Eingaben:
- zwei Listen L_1 und L_2 der Länge n

Zulässige Ausgaben:
- die Zahlen 0 oder 1:
  1 soll ausgegeben werden, wenn L_1 und L_2 identisch sind;
  0 soll ausgegeben werden, wenn L_1 und L_2 nicht identisch sind.

Beispielsweise wird berechnet
Ident((erste,Liste),(zweite,Liste)) = 0
Ident((eine,Liste),(eine,Liste)) = 1

- Beispiel (2) – ein „neues“ Problem – Typ 2

... überprüfen, ob eine Liste L_2 eine Permutation einer gleich langen Liste L_1 ist

Zulässige Eingaben:
- zwei Listen L_1 und L_2 der Länge n

Zulässige Ausgaben:
- die Zahlen 0 oder 1:
  1 soll ausgegeben werden, wenn L_2 eine Permutation von L_1 ist;
  0 soll ausgegeben werden, wenn L_2 keine Permutation von L_1 ist

Beispielsweise wird berechnet
SindPermu((erste,Liste),(Liste,zweite)) = 0
SindPermu((L,I,S,T,E),(S,T,E,I,L)) = 1

Mit welcher Transformation kann jedes Typ-2-Problem auf ein Typ-1-Problem reduziert werden?

Problemreduktion – Listen

- Beispiel (2) – Problemreduktion – den Typ-1-Algorithmus für Typ 2 benutzen

  - Gegeben seien die Listen L_1 und L_2 gleicher endlicher Länge.
  - Es gibt eine „natürliche“ Ordnung „$\leq$“ zwischen Listenelementen (oder mehrere, dann für eine davon entscheiden).

  - Transformation :
    Bestimme die $\leq$-sortierten Versionen L_1' von L_1 und L_2' von L_2
  - Löse das Identitätsproblem (Typ 1) für L_1' und L_2'
    und übernimm die Antwort.

Die Transformation kann effizient durchgeführt werden.

- ein Beispiel aus der Informatik (3) – „beherrschtes“ Problem – Typ 1

zulässige Eingaben:

- ein ungerichteter Graph G = (V,E)
- eine Zahl k $\in$ N

V : „vertices“, Knoten
E : „edges“, Kanten
N = { 0, 1, 2, ... }

zulässige Ausgaben:

- die Zahlen 0 oder 1
- Die Zahl 1 soll ausgegeben werden, wenn es im Graphen G (mindestens) eine Clique der Größe größer gleich k gibt.
- Die Zahl 0 soll ausgegeben werden, wenn es im Graphen G keine Clique der Größe größer gleich k gibt.

Einschub: Begriff Clique

- Es sei G = (V,E) ein ungerichteter Graph.
- Es sei $k \in \mathbb{N}$.

G′ = (V′,E′) heißt Clique der Größe k von G = (V,E), falls gilt:

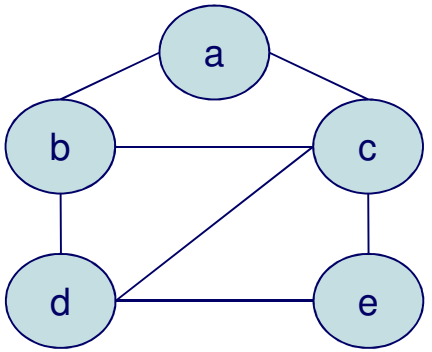
- $V' \subseteq V$ mit $|V'| = k$
- $E' \subseteq E$
- für alle $x,y \in V'$ mit $x \neq y$ gilt: $\{x,y\} \in E'$

G‘ ist ein Teilgraph der Größe k von G, in dem je zwei verschiedene Knoten durch eine Kante verbunden sind.
(Merksatz: „In unserer Clique fühlt sich jeder mit jedem verbunden.“)

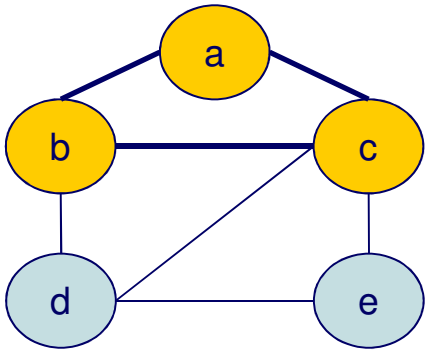
Problemreduktion – Graphen

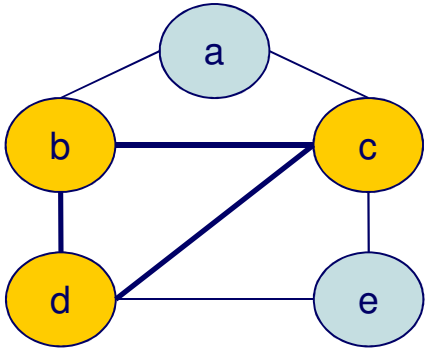
- Beispiel

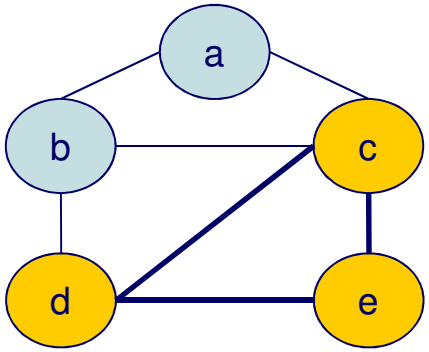
  - ungerichteter Graph G = (V,E) mit
    - V = { a,b,c,d,e }
    - E = { {a,b},{a,c},{b,c},{b,d},{c,d},{c,e},{d,e} }



- Cliquen der Größe 3 (und zwar alle)







- drittes Beispiel: Problem Typ 2 (Entscheidung bzgl. logischer Formeln)

zulässige Eingaben:

- eine aussagenlogische Formel F in konjunktiver Normalform

KNF: Konjunktion (und) von Disjunktionen (oder) von Literalen (x/¬x)

zulässige Ausgaben:

- die Zahlen 0 oder 1
- die Zahl 1 soll ausgegeben werden, wenn es mindestens eine Belegung β gibt, die die Formel F erfüllt
- die Zahl 0 soll ausgegeben werden, wenn es keine Belegung β gibt, die die Formel F erfüllt

Beispiel

- $F = (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$

β(x)	β(y)	β(z)	β(F)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Belegung β_1, die F erfüllt

Belegung β_2, die F nicht erfüllt

- Beobachtung
  - Problem 1 und Problem 2 sehen ziemlich verschieden aus.
    - Bei Problem 1 geht es um die Frage, ob ein gegebener ungerichteter Graph eine bestimmte Eigenschaft hat.
    - Bei Problem 2 geht es darum, ob eine aussagenlogische Formel in konjunktiver Normalform eine bestimmte Eigenschaft hat.
- ... Zusammenhang zwischen Problem 1 und Problem 2

> Wenn es einen Algorithmus / ein Programm zur Lösung von Problem 1 gibt, so gibt es auch einen Algorithmus / ein Programm zur Lösung von Problem 2.

- Algorithmus 3 (Umformung)

zulässige Eingaben:

- eine aussagenlogische Formel F in konjunktiver Normalform

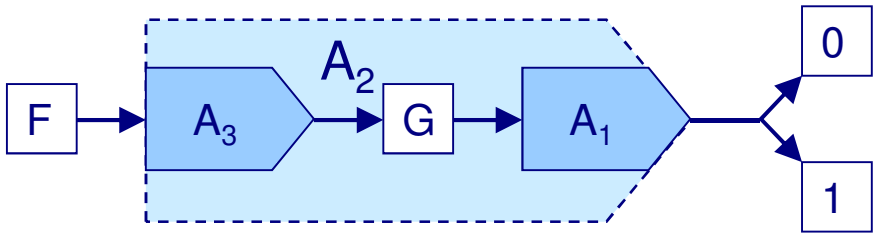
zulässige Ausgaben:

- ein ungerichteter Graph G und eine Zahl $k \in N$, wobei gelten soll:
  - wenn es mindestens eine Belegung β gibt, die die Formel F erfüllt, so soll der Graph G mindestens eine Clique der Größe größer gleich k haben
  - wenn es keine Belegung β gibt, die die Formel F erfüllt, so soll der Graph G keine Clique der Größe größer gleich k haben

- Algorithmus A_2 für Problem 2 (prinzipielle Idee)

  - es sei A_1 ein Algorithmus zur Lösung von Problem 1
  - es sei A_3 ein Algorithmus zur Lösung von Problem 3

Eingabe:	eine aussagenlogische Formel F in konjunktiver Normalform	
Schritt 1:	-	bestimme mit Algorithmus A_3 einen zugehörigen ungerichteten Graphen G = (V,E) und eine Zahl $k \in N$
Schritt 2:	-	bestimme mit Algorithmus A_1, ob G = (V,E) eine Clique der Größe größer gleich k hat
	-	falls ja, gib die Zahl 1 aus; andernfalls gib die Zahl 0 aus



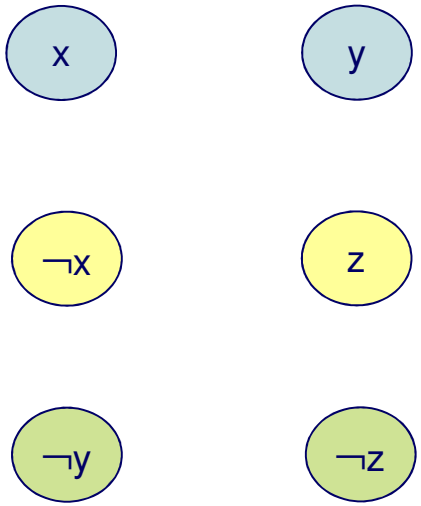
- Transformationsalgorithmus A_3 für Problem 3 (am Beispiel)

  - $F = (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$

Je Literalvorkommen in jeder Klausel gibt es einen Knoten (zur Unterscheidung werden unterschiedliche Farben verwendet).

Klauseln: $(x \vee y), (\neg x \vee z), (\neg y \vee \neg z)$

Literale: $x, y, \neg x, z, \neg y, \neg z$

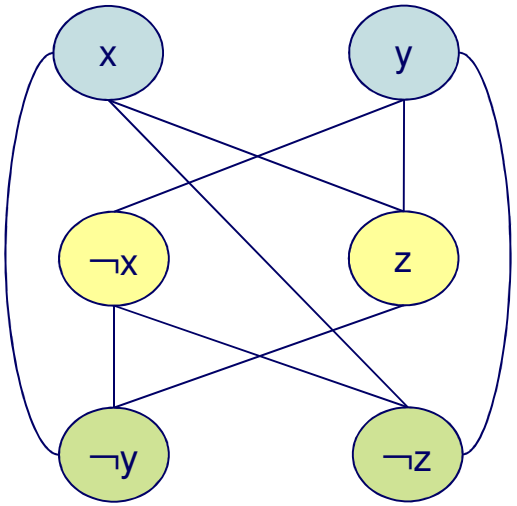


- Transformationsalgorithmus A_3 für Problem 3 (am Beispiel)

  - $F = (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$

Zwei Knoten werden genau dann mit einer Kante verbunden, wenn

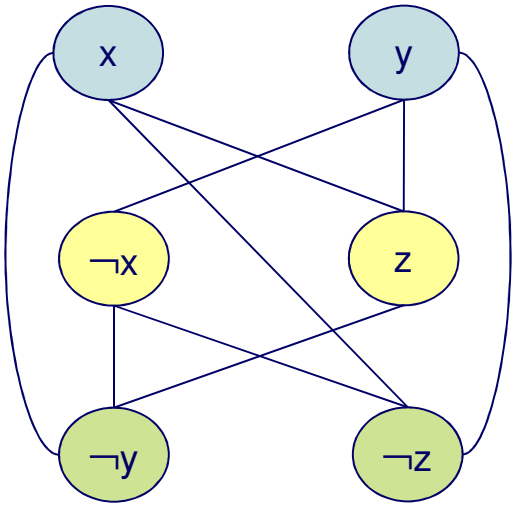
- *sie Literale in unterschiedlichen Klauseln repräsentieren (also unterschiedliche Farben haben) und*
- *diese Literale nicht komplementär sind (u.a. werden die Knoten x und ¬x nicht verbunden)*



- Transformationsalgorithmus A_3 für Problem 3 (am Beispiel)

  - $F = (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$

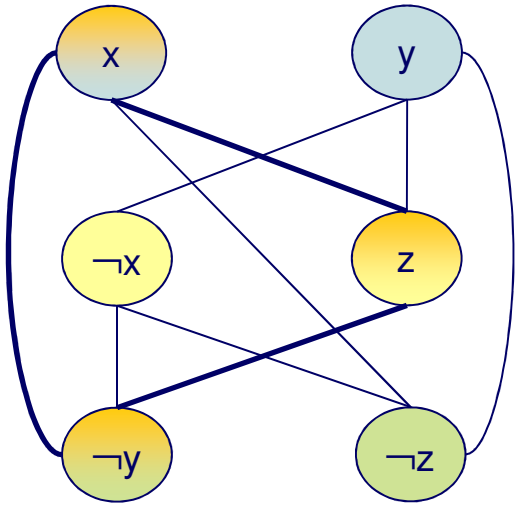
  - *Der zu F gehörige Graph G = (V,E) sieht wie folgt aus ...*
  - *Die Zahl $k \in N$ wird gewählt als k = Anzahl der Klauseln/Farben*



- Beziehung zwischen F und dem zugehörigen G (am Beispiel)

$$F = (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$$

β(x)	β(y)	β(z)	β(F)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

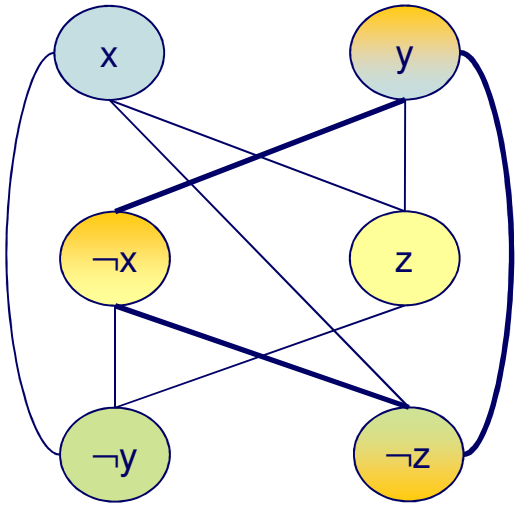


Belegung, die *F* erfüllt ⟷ 3-Clique

- Beziehung zwischen F und dem zugehörigen G (am Beispiel)

$$F = (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$$

β(x)	β(y)	β(z)	β(F)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Belegung, die *F* erfüllt ⟷ 3-Clique

- ... etwas genauer (für die Transformation)

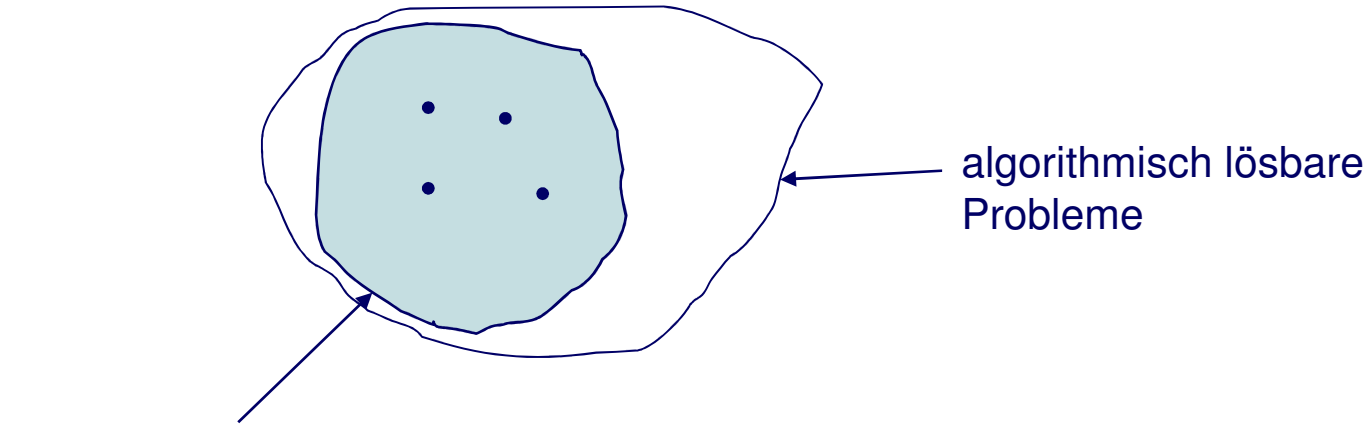
> Es gibt einen effizienten Algorithmus / ein effizientes Programm zur Transformation.

> *effizient heißt mit einer Rechenzeit polynomiell in der Größe der gegebenen Formel*

- ... Zusammenhang zwischen Problem 1 und Problem 2 (etwas genauer)

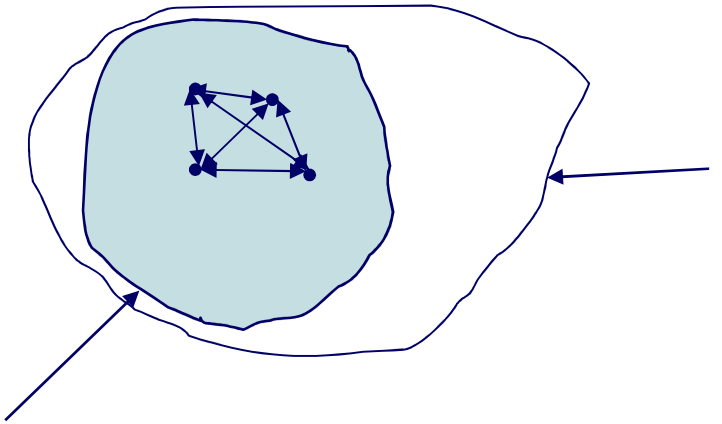
> Wenn es einen effizienten Algorithmus / ein effizientes Programm zur Lösung von Problem 1 gibt, so gibt es auch einen effizienten Algorithmus / ein effizientes Programm zur Lösung von Problem 2.

- Doch leider …
  - ist bisher **kein effizienter** Algorithmus / effizientes Programm zur Lösung von Problem 1 bekannt.
    Man vermutet sogar, dass es gar keinen effizienten Algorithmus / kein effizientes Programm dazu gibt.
  - Die beiden Probleme sind Vertreter einer ganzen Klasse ähnlich gearteter Probleme



Einordnung

- Bisher ist kein effizienter Algorithmus / effizientes Programm zur Lösung von Problem 1 bekannt.
  Man vermutet sogar, dass es gar keinen effizienten Algorithmus / kein effizientes Programm dazu gibt.
- Die beiden Probleme sind Vertreter einer ganzen Klasse ähnlich gearteter Probleme



Probleme wie Problem 1 und 2 (.... wenn man eines dieser Problem effizient lösen kann, dann jedes: „NP-vollständig“)

Praxisbezug

- Einschub: Wo kommen Cliquen in der Praxis vor?

Praxisbezug

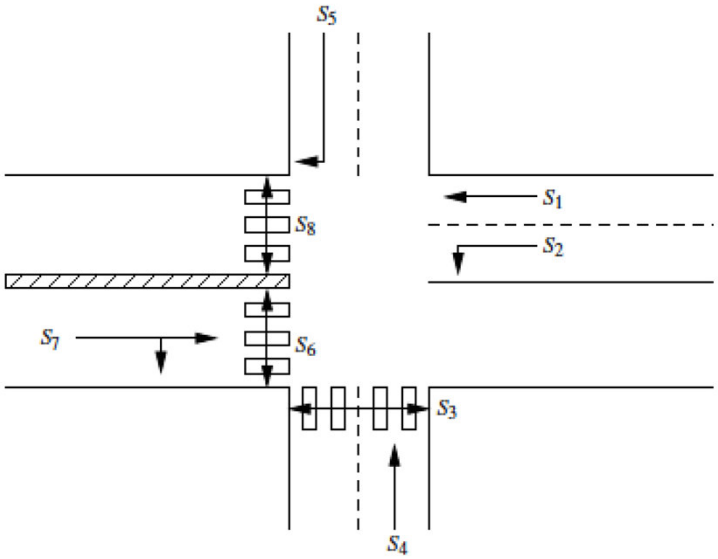
Beispiel Ampelkreuzung

An einer signalgesteuerten Verkehrskreuzung sind die Verkehrsströme sinnvoll zu regeln, d.h.:

- Zwischen den gleichzeitig freigegebenen Verkehrsströmen dürfen keine „Kollisionen“ auftreten. Unfälle sollen nicht durch einander kreuzende Ströme provoziert werden.
- Die Verkehrsströme sollen möglichst „gut“ bedient werden (wenig Wartezeit, hohe Durchlässigkeit, Fairness).

Praxisbezug

- konkrete Verkehrskreuzung mit Verkehrsströmen



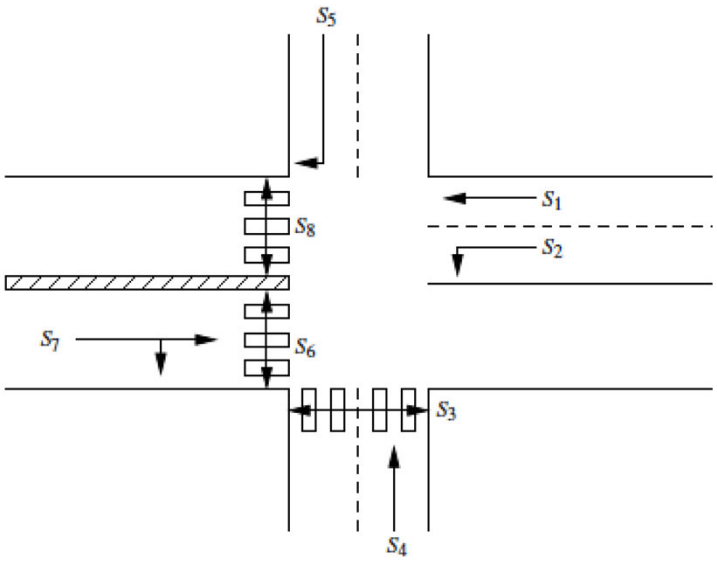
Offenbar kann man jeweils einen Verkehrsstrom s_i allein auf „grün“ schalten und so alle Verkehrsströme nacheinander zyklisch bedienen.
Das Verfahren wäre fair – kein s_i „verhungert“; die Kapazität der Kreuzung würde dabei allerdings nicht optimal genutzt.

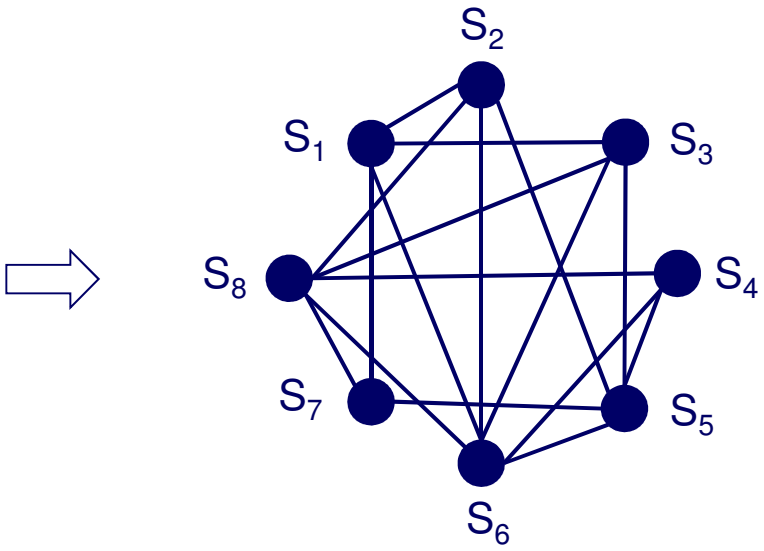
- auf dem Weg zu einer besseren Lösung

  - Man verwendet einen ungerichteten Graphen G, um die Beziehungen zwischen den möglichen Verkehrsströmen zu modellieren, dabei gilt:
    - Der Graph G enthält für jeden Verkehrsstrom s_i einen Knoten
    - Zwischen zwei Knoten gibt es genau dann eine Kante, wenn die zugehörigen Verkehrsströme gleichzeitig kollisionsfrei auf „grün“ geschaltet werden können.

Praxisbezug

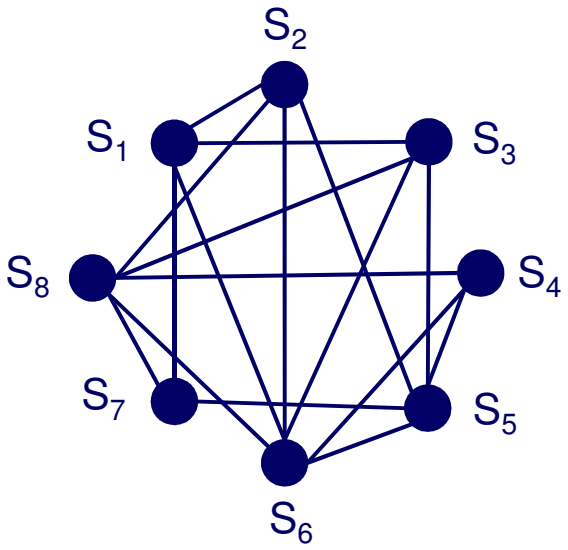
- Illustration





Praxisbezug

- Analyse des zugehörigen Graphen



- Im ungerichteten Graphen findet man u.a. die folgenden Cliquen, d.h. Teilmengen der Knotenmenge, in denen je zwei verschiedene Knoten mit einer Kante verbunden sind:
  - $C_1 = \{ S_1,S_2,S_6 \}$; $C_2 = \{ S_2,S_5,S_6 \}$; $C_3 = \{ S_3,S_5,S_6 \}$
  - $C_4 = \{ S_4,S_5,S_6 \}$; $C_5 = \{ S_4,S_6,S_8 \}$; $C_6 = \{ S_7,S_8 \}$

Zwischen den Verkehrsströmen in einer Clique gibt es keine Kollisionen.

Praxisbezug

Konsequenzen

- Die Verkehrsströme in einer Clique können gemeinsam auf „grün“ geschaltet und bedient werden. Eine Abfolge von Cliquen, in denen insgesamt alle Verkehrsströme vorkommen (→ **Fairness**), kann nun zyklisch benutzt werden, um die an der Kreuzung auftretenden Verkehrsströme zu regeln.
- Damit nun kein Verkehrsstrom unnötig rot hat (→ **Optimierung** der Durchlässigkeit), verwendet man *maximale* Cliquen, zu denen also jeweils keine echt größeren Cliquen existieren, …
- in unserem Beispiel etwa:
  - 1. Grünphase: $C_1 = \{ S_1, S_2, S_6 \}$
  - 2. Grünphase: $C_3 = \{ S_3, S_5, S_6 \}$
  - 3. Grünphase: $C_5 = \{ S_4, S_6, S_8 \}$
  - 4. Grünphase: $C_6 = \{ S_7, S_8 \}$

Auf diese Weise können die Verkehrsströme „besser“ bedient werden.