Theoretische Informatik

Komplexitätstheorie

Inhalt

- Komplexität
  - Nichtdeterministisch Polynomiale Probleme
  - SAT ist NP-hart
  - Polynomiale Reduzierbarkeit
  - NP-Vollständige Probleme

Effizienz von Lösungen

- Wir haben viele Probleme gesehen
  - einige unlösbar
    - hält P_m mit Input n ?
    - berechnen P_i und P_j die gleiche Funktion ?
  - andere lösbar
    - ist $w \in L(A)$ für A ein Automat ?
    - ist $w \in L(G)$ für G kontextfrei ?
  - einige zwar lösbar, aber extrem aufwendig
    - ist k = ack(m,n) ?
- Bisherige Fragestellung
  - lösbar oder nicht lösbar
  - prinzipielle Untersuchung
    - keine Betrachtung des Aufwands
- Jetzt
  - was ist realistisch lösbar, oder
  - was ist in der Praxis nur für kleine Inputs lösbar

Generate and Test

- Oft benutzter Lösungsalgorithmus:
  - Erzeuge alle Lösungskandidaten
    - Generate
  - Prüfe ob der Kandidat eine Lösung ist
    - Test (bzw. Check)
- Naiver Lösungsweg, aber
  - für theoretische Zwecke o.k.
  - für die Praxis oft zu ineffizient.
- In diesem Kapitel lernen wir:
  - es gibt (lösbare) Probleme, für die (vermutlich) kein besserer Algorithmus existiert
  - könnte man eines effizient lösen, so könnte man alle effizient lösen

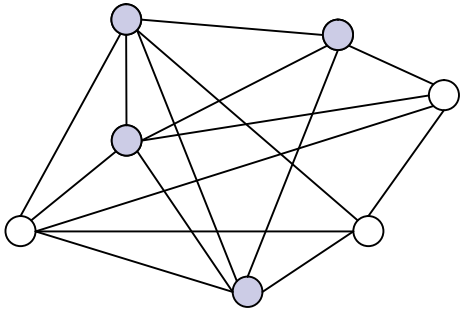
SAT-Problem

- Gegeben:
  - eine Formel der Aussagenlogik
    - Ausdruck $F(x_1,\ldots,x_n)$ aus
      - Variablen $x_1,x_2,\ldots$
      - Operatoren $\wedge$, $\vee$, $\neg$
- Gefragt:
  - ist sie erfüllbar (engl.: *satisfiable*)
    - gibt es eine Belegung
      - Zuweisung der Werte 0,1 an die Variablen
      - 0 = false, 1 = true
    - so dass F = 1
  - Äquivalentes Problem:
    - Ist $\neg F$ eine Tautologie

SAT : Generate and Test

- Lösungsmethode für SAT Problem
  - Generate
    - Erzeuge alle potentiellen Kandidaten
      - alle n-Tupel $(b_1,\ldots,b_n)$ mit $b_i \in \{0,1\}$
  - Test
    - prüfe, ob ein Kandidat eine Lösung ist
      - einigermaßen effizient
      - linear in Formelgröße |op|+n
- Problematik
  - Bei n Variablen gibt es 2^n Kandidaten
  - Algorithmus hat exponentielle Laufzeit
    - in Anzahl der Variablen
    - im average/worst case

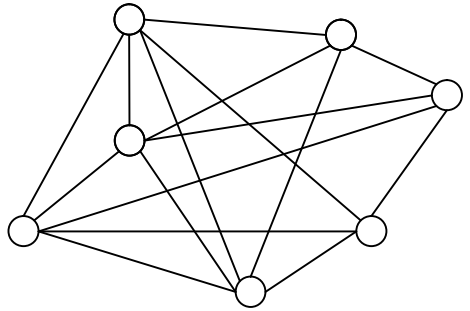
CLIQUE



Symmetrischer Graph mit 4-Clique

- G=(V,E) ein Graph:
  - V eine Menge, $E \subseteq V \times V$ eine Relation
    - V : Menge der Knoten, (engl: *vertex*)
    - E : Menge der Kanten, (engl.: *edge*)
- Eine Clique ist ein vollständiger Teilgraph
  - $C \subseteq V$ mit $\forall v_1, v_2 \in C: (v_1, v_2) \in E$
- Beispiel:
  - V = Studenten der VL
  - $(v_1, v_2) \in E \ :\Leftrightarrow v_1$ kennt v_2
  - Ein Clique ist eine Gruppe von Studenten die sich gegenseitig kennen

Cliquenproblem

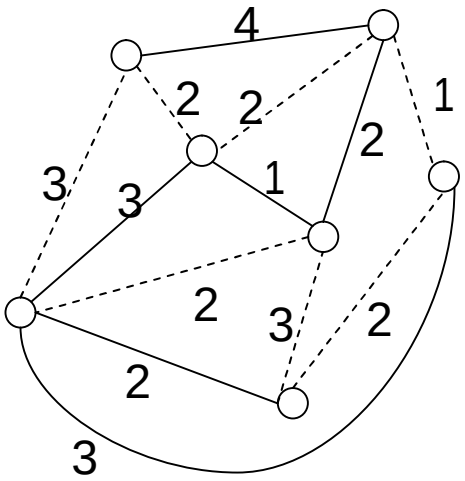


Größtmögliche Clique ?

- Gegeben ein Graph
  - maximale Größe einer Clique ?
- Lösungsmethode (generate & test)
  - erzeuge alle Teilmengen $C \subseteq V$
  - prüfe jeweils, ob C eine Clique ist
- Effizienz
  - Generate
    - ineffizient, weil ~ $2^{|V|}$ Kandidaten
  - Test
    - vergleichbar schnell
    - maximal $\frac{1}{2} \cdot |C|^2$ Paare zu testen

TSP-travelling salesman problem

- Gegeben
  - eine bewerteter Graph
    - Knoten sind Städte
    - Kanten sind Strassen
    - Bewertung = Entfernung
- Gesucht
  - minimale Rundreise
    - Reise durch alle Städte
    - Gesamtstrecke minimal



Wegenetz mit Rundreise der Länge 15

Ist 15 optimal ?

Ja/nein-Probleme

- Fragestellung mit zwei möglichen Lösungen
  - ja / nein
- Entspricht
  - Berechnung einer charakteristischen Funktion
    - P = alle Probleme (oft $P = \Sigma^*$ für geeignete Codierung)
    - $L \subseteq P$ die Menge aller lösbaren Probleme
  - Ist w eine Lösung ?
    - Gegeben: $w \in P$
    - Gefragt: $w \in L$?

Rückführung

- Lösungssuche kann oft auf Frage nach Existenz einer Lösung zurückgeführt werden

- Beispiel SAT: Finde eine Lösung für $F(x_1,\ldots,x_n)$
  - Hat $F(0,x_2,\ldots,x_n)$ eine Lösung ?
    - wenn ja: $x_1 = 0$ und weiter mit $F(0,x_2,\ldots,x_n)$
  - Hat $F(1,x_2,\ldots,x_n)$ eine Lösung ?
    - wenn ja: $x_1 = 1$ und weiter mit $F(1,x_2,\ldots,x_n)$

- Maximal n ja/nein Fragen führen zur Lösung
  - falls sie existiert

Clique als ja/nein-Problem

- G=(V,E)
  - Gesucht: Maximale Größe c einer Clique ?
  - obere Schranke: $c \leq |V|$
- Ja/Nein Problem (für beliebiges $k \leq |V|$):
  - Hat G(V,E) Clique der Größe k (für ein $k \leq |V|$)
    - ja: $c \geq k$
    - nein: $c < k$
- Maximal $\log_2|V|$ viele Ja/Nein Fragen führen zur Lösung
  - Effiziente Lösung des Ja/Nein-Problems liefert effiziente Lösung des Cliquenproblems

Allgemeine Rückführung

- Berechnung einer Funktion $f:\Sigma^* \rightarrow Nat$
  - Bekannt:
    - obere Schranke $f(w) \leq max$
    - untere Schranke $min \leq f(w)$
  - Finde f(w) durch Schachtelung
    - Ist $f(w) \leq ½(min+max)$ – ja/nein ?
      - ja - neue obere Schranke
      - nein – neue untere Schranke
  - Maximal $\log_2 max$ viele ja/nein-Fragen liefern Ergebnis
    - Anwendbar auch auf TSP
- Konsequenz:
  - Im folgenden betrachten wir nur noch ja/nein-Fragen

Probleme sind Sprachen

- SAT-Probleme
  - Alphabet $\Sigma = \{ x, 0, 1, \neg, \vee, \wedge \}$
    - statt $x_1, x_2, x_3, \ldots$ verwende x, xx, xxx,... etc.
  - Jedes SAT-Problem ist Wort über Σ
  - Gegeben durch CF-Grammatik G
  - Sat-Probleme = $L(G) \subseteq \Sigma^*$
- Lösbare (solvable) SAT-Probleme:
  - $SAT \subseteq L(G) \subseteq \Sigma^*$
- Gesucht also:
  - Entscheidungsverfahren für Sprache SAT

Cliquenproblem

- Alphabet Σ ={ x, 0, 1, (,), \, }
- Codiere durch einfache Teilsprache $L \subseteq \Sigma^*$
  - Knoten durch x, xx, xxx, …
  - Kanten durch Paare (v_1,v_2) von Knoten
  - k als Binärzahl
  - k-Cliquenproblem durch $(k,(v_{s1},v_{t2}),\ldots,(v_{sn},v_{tn}))$
- Lösbare k-Cliquenprobleme
  - $CLIQUE_k \subseteq L \subseteq \Sigma^*$

Rahmen in diesem Kapitel

- Gegeben ein Problem
  - Codierbar als Sprache $L \subseteq \Sigma^*$
  - L auf jeden Fall entscheidbar
  - es gibt also Turingmaschine T, die L entscheidet
- Frage
  - Ist L effizient entscheidbar ?
  - was heißt das ?
- Aufwand von T für ein Wort w ?
  - $time_T(w)$ = Anzahl der Schritte von T um w zu entscheiden
- Zeitaufwand von T:
  - $time_T(n) = max \{ time_T(w) \mid w \in \Sigma^*, |w| \leq n \}$
  - Kostenfunktion: $time_T : N \rightarrow N$

Algorithmenkomplexität

- T entscheide $L \subseteq \Sigma^*$
  - Kostenfunktion $time_T : N \to N$
  - meist nur abschätzbar
  - $t : N \to N$ sei Schätzfunktion
- T hat Komplexität O(t) falls $time_T \in O(t)$
  - Erinnerung:
    - $\exists c \in N.\ \exists n_0 \in N.\ \forall n \geq n_0.\ time_T(n) \leq c * t(n)$
    - ab n_0 wird $time_T$ durch c*t dominiert
- T gilt als effizient, falls sie polynomiale Komplexität hat
  - Falls es ein Polynom p(n) gibt mit $time_T \in O(p)$

Sprachkomplexität

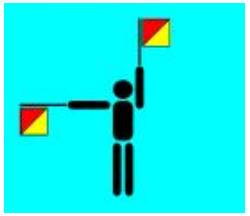
- $L \subseteq \Sigma^*$ sei entscheidbare Sprache
  - $t : N \to N$ irgendeine (Kosten)funktion

- L hat Komplexität O(t) falls es eine TM T gibt mit $time_T \in O(t)$

  - L hat polynomielle Komplexität, falls es ein Polynom p(n) gibt, so dass L Komplexität O(p) hat.

- Beispiele:
  - L = Palindrome $\subseteq \Sigma^*$ hat Komplexität $O(n^2)$
    - Lese erstes Zeichen, laufe ans Ende, vergleiche, etc.
    - Höchstens $n+(n-2)+(n-4)+\ldots+2 \leq n^2$ Schritte
  - L = SAT :
    - kein polynomieller Entscheidungsalgorithmus bekannt
  - L = CLIQUE, L=TSP :
    - kein polynomieller Entscheidungsalgorithmus bekannt

Die Klasse P

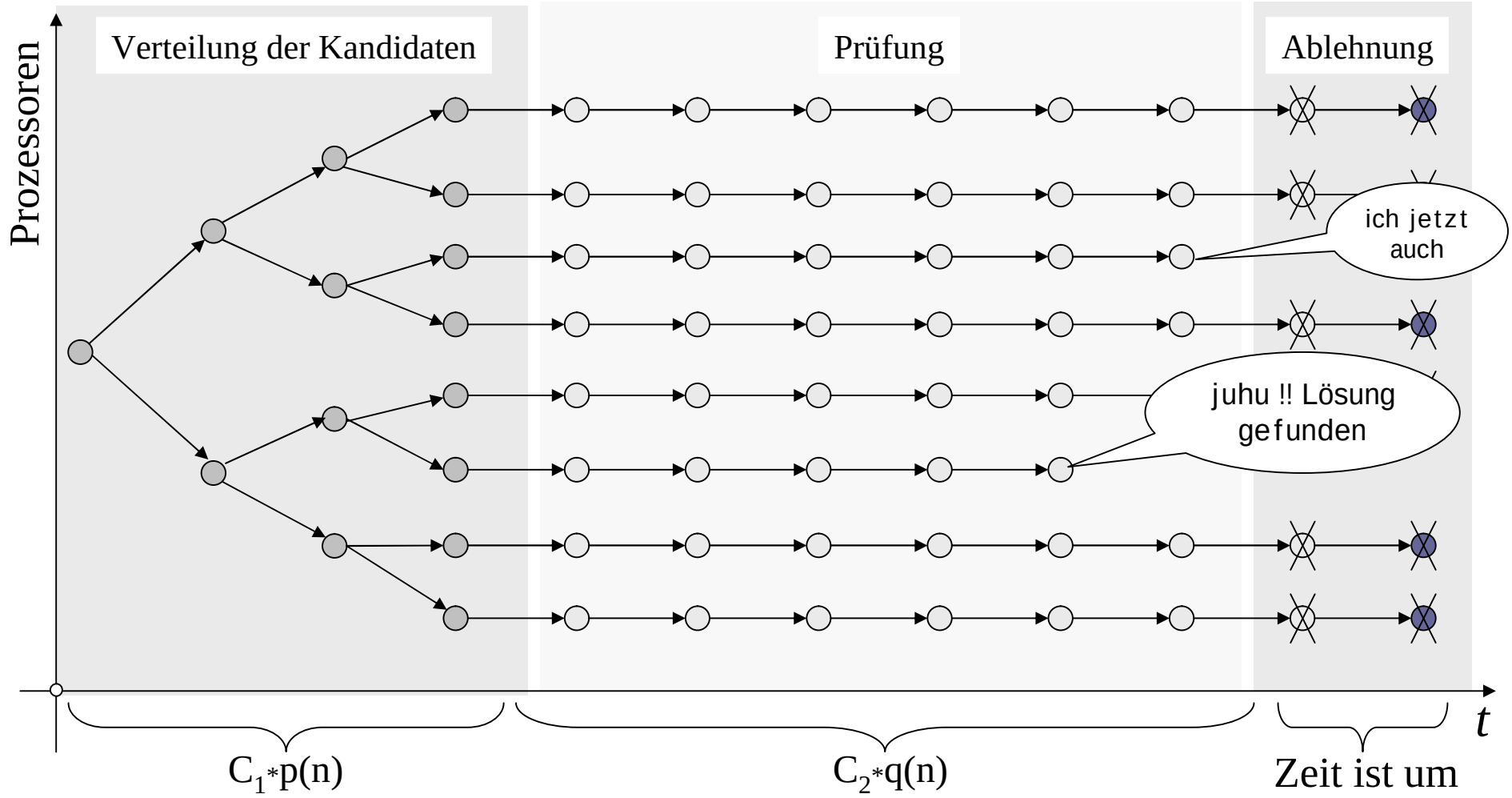


- P = Klasse aller Sprachen mit polynomieller Komplexität
  - Palindrom $\in$ P
  - SAT : unbekannt
    - Vermutung: SAT $\notin$ P
  - CLIQUE, TSP : unbekannt
    - Vermutung: CLIQUE $\notin$ P, TSP $\notin$ P
- Wir werden sehen:
  - SAT $\in$ P $\Leftrightarrow$ CLIQUE $\in$ P $\Leftrightarrow$ TSP $\in$ P $\Leftrightarrow$ ...
    - Wenn eine davon in P ist, dann alle
    - Wenn eine davon nicht in P ist, dann alle
  - Bis heute ungelöstes Problem
    - Wenn Sie berühmt (und reich) werden wollen:
      - lösen Sie es
      - sonst wird es jemand anderes tun ...

Parallelität

- Welchen Effizienzgewinn kann Parallelität bringen?
  - unbegrenzt viele Rechner
  - Vernachlässigung von Verwaltungskosten
- Problemstruktur
  - Generate and Test

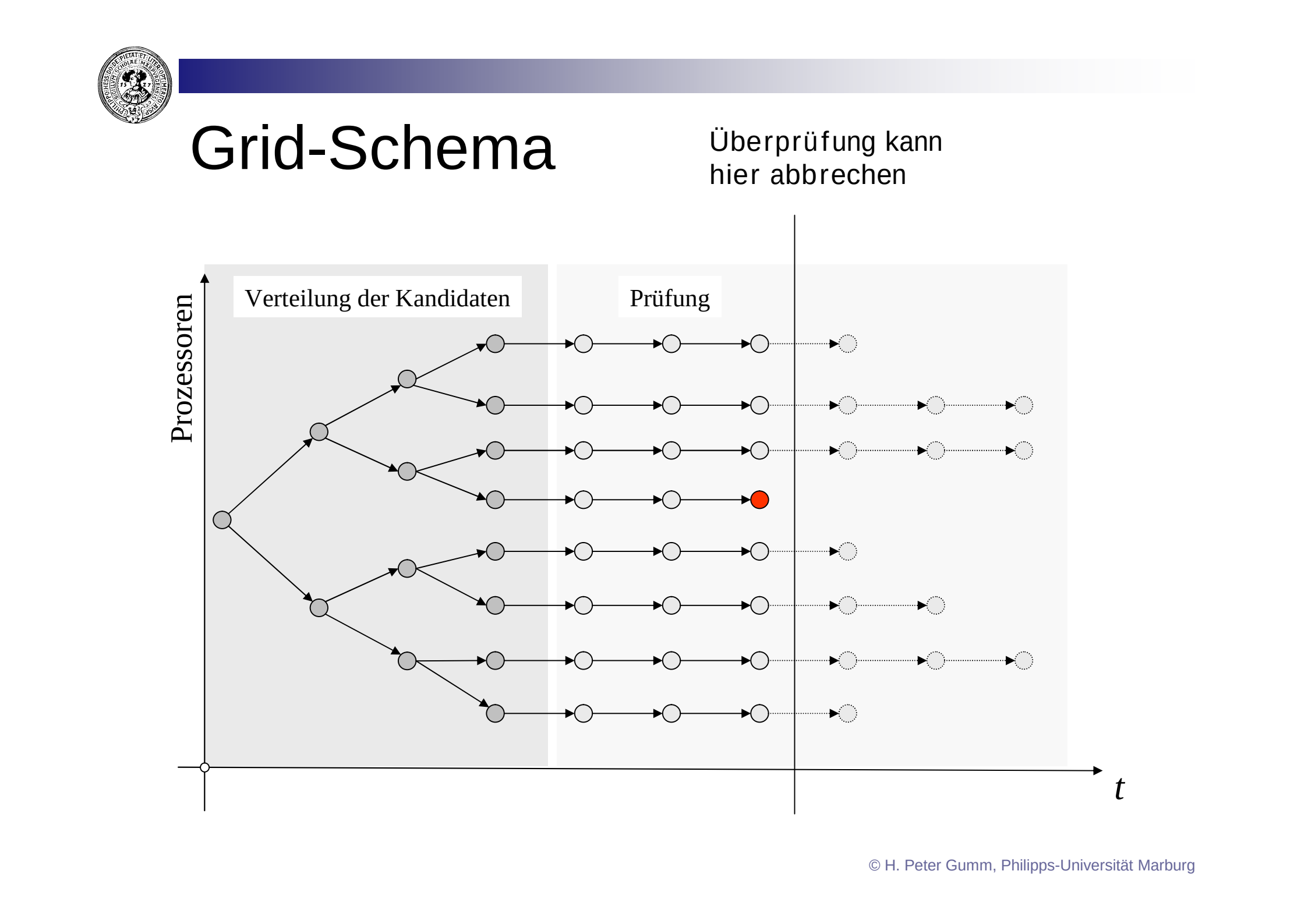
  legt Parallelität nahe.
- Annahme
  - unbegrenzt viele Rechner
  - Vernachlässigung von Verwaltungskosten
- Generate
  - Jeder Rechner bekommt einen Kadidaten
- Test
  - Gleichzeitig prüft jeder Rechner seinen Kandidaten
    - relativ effizient



Parallelismus

Grid-Computing

- SETI@Home am bekanntesten
  - PCs sollen in der Leerlauf-Zeit (idle time) etwas nützliches machen
    - nach Signalen intelligenter Wesen suchen
  - Search for ExtraTerrestical Intelligence
  - setiathome.ssl.berkely.edu
- Zentrale verteilt Lösungskandidaten auf PCs
  - Jeder PC rechnet für sich
    - falls ein Kandidat Lösung ist
    - melde an Zentrale
  - wenn alle einzelnen PCs erfolglos terminiert haben
    - keine Lösung
- Modifikation
  - Hierarchische Verteilung der Kandidaten, oder
  - Broadcast



Grid-Schema

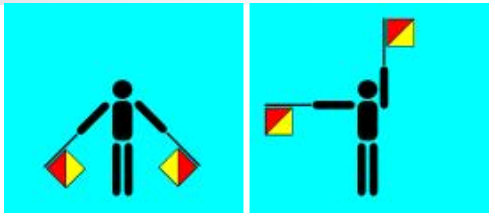
Überprüfung kann
hier abbrechen

Beispiel: SAT und CLIQUE im Grid

- Gegeben eine Formel mit
  - n Variablen und
  - |op| Variablen
- 2^n Lösungskandidaten
  - Verteilung:
    - $t_V = O(\log_2(2^n)) = O(n)$
  - Prüfung eines Kandidaten
    - $t_P = O(|op|+n)$

- Gegeben Graph mit
  - n Knoten
  - $\leq \frac{1}{2}n(n+1)$ Kanten
- 2^n Kandidaten
  - Verteilung:
    - $t_V = O(n)$
  - Prüfung eines Kandidaten
    - $t_P = O(n^2)$

NP – intuitive Definition



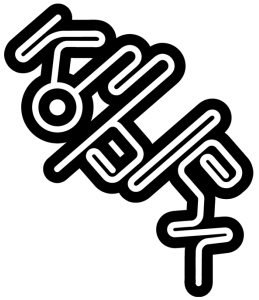
- P
  - Probleme, die effizient lösbar sind
    - in polynomieller Zeit
- NP
  - Probleme, deren Lösungskandidaten effizient überprüft werden können
    - in polynomieller Zeit
  - Es dürfen exponentiell viele Lösungskandidaten sein
    - hierarchische Verteilung

Nichtdeterministische TM

- $\delta: Q \times \Sigma \rightarrow \wp(Q \times \Sigma \times \{L,R\})$
  - wähle stets aus einer Menge von Alternativen
    - $\delta(q_i,e) = \{ (q_{i0},e_0,d_0), \ldots, (q_{ik},e_k,d_k) \}$
  - Stopp,
    - falls Endzustand erreicht, oder falls
    - $\delta(q_i,e) = \{ \}$
- L(M) für NDTM
  - $w \in L(M)$, falls akzeptierende Berechnung mit Input w existiert
  - $nTime_T(w)$ = minimale Anzahl von Schritten für eine akzeptierende Berechnung
  - optimistischer Ansatz !

Nichtdeterministische Komplexität

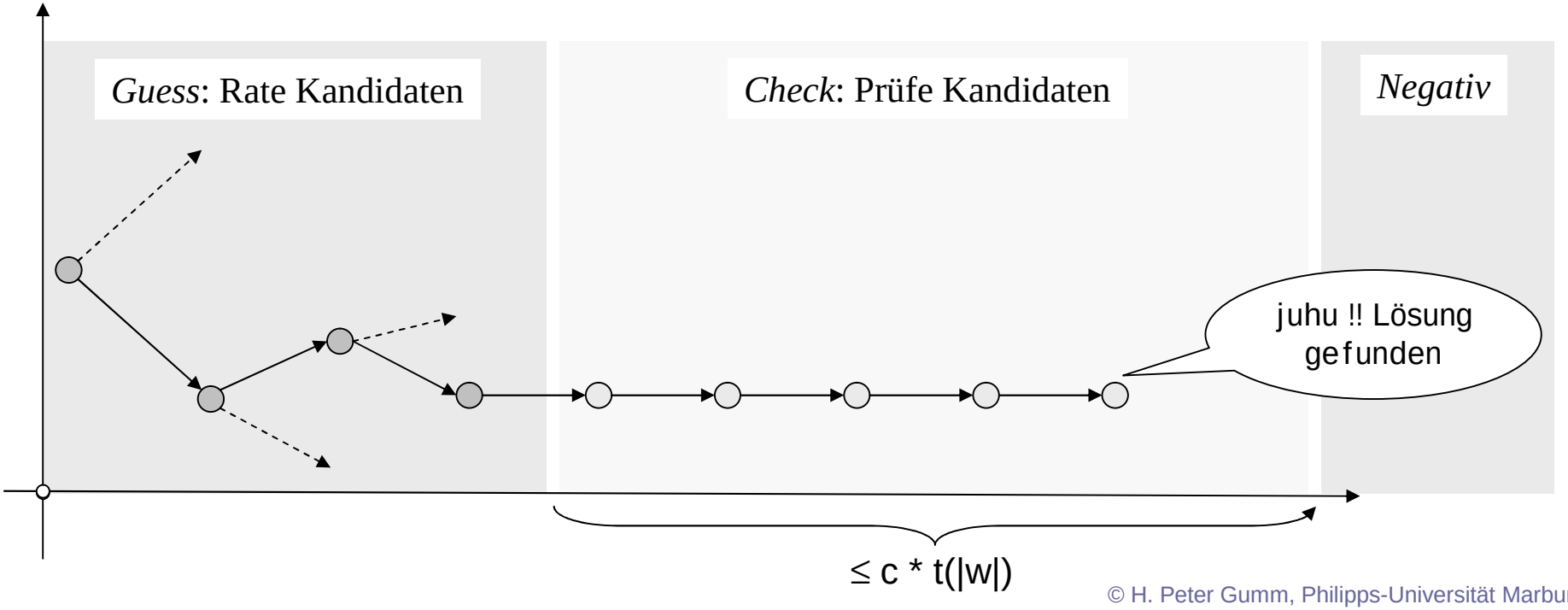
- Gegeben
  - $t : N \to N$ Schätzfunktion,
  - entscheidbare Sprache $L \subseteq \Sigma^*$



- nTime(L) = O(t), falls eine nicht-deterministische Turingmaschine T existiert,
  - die genau L akzeptiert
  - mit $\forall w \in \Sigma^*.\ nTime_T(|w|) = O(\ t(|w|)\ )$

Nondeterminismus – Guess and Check

- Nondeterminismus:
  - betrachtet sofort erfolgreichen Pfad
- „Guess“ statt „Generate“
  - rate einen Kandidaten
- Check
  - prüfe, dass es eine Lösung ist



NP = Nichtdeterministisch Polynomial

- NP = Klasse aller Sprachen L mit nichtdeterministisch polynomialer Komplexität
  - NP = Klasse aller Sprachen L für die ein Polynom p(n) existiert mit nTime(L) = O(p(n))
- Nichtdeterminismus modelliert Parallelität
  - Jede nichtdeterministische Auswahl führt zur Erzeugung von entsprechend vielen parallel ablaufenden Prozessen
  - Der erste, der eine Entscheidung bringt, veranlasst Terminierung aller Prozesse.

Guess and Check: Beispiele

- SAT
  - Guess
    - Rate eine Belegung $(x_1,\ldots,x_n) \in 2^n$
    - n Schritte
  - Check
    - Prüfe, ob $F(x_1,\ldots,x_n)=1$
    - Proportional zu Größe der Formel
      - $|op|+n$
  - $SAT \in NP$

- k-CLIQUE
  - Guess
    - Rate eine k-Clique (k-elementige Teilmenge von $C \subseteq V$)
  - Check
    - Prüfe, ob $\forall c_1,c_2 \in C: (c_1,c_2) \in E$
    - Proportional $|C|^2$,
    - insbesondere polynomial in $|V|$
  - $k\text{-CLIQUE} \in NP$

Reduzierbarkeit

$L \subseteq \Sigma^*$, $M \subseteq \Gamma^*$ Sprachen $f: \Sigma^* \to \Gamma^*$ berechenbare Abbildung

- L heißt f-reduzierbar auf M falls

$$w \in L \Leftrightarrow f(w) \in M$$

  - Es gilt: M entscheidbar $\Rightarrow$ L entscheidbar

- L polynomial reduzierbar auf M,

$$L \leq_p M$$

  falls f-reduzierbar und time(f(w)) = O(p(|w|)) für eine Polynom p

  - d.h. es gibt eine TM T_f, die f berechnet und für jeden Input $w \in \Sigma^*$ höchstens p(|w|) Schritte benötigt

- $L \leq_p M \in NP \quad \Rightarrow \quad L \in NP$

$\leq_p$ ist transitiv

- Lemma: Aus $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$ folgt $L_1 \leq_p L_3$

- Beweis: Seien $L_i \subseteq \Sigma_i^*$ (i=1,2,3) und $f_k: \Sigma_k^* \to \Sigma_{k+1}^*$ (k=1,2) berechenbar von polynomieller Komplexität p_k, d.h.
  - es gibt TM T_k, die in $c_k * p_k(|w|)$ Schritten f_k berechnen.

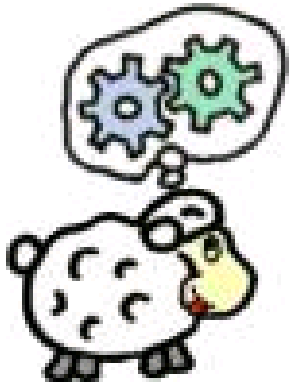
- Für die Berechnung von $f_2(f_1(w))$ braucht T_2 maximal $c_2 * p_2(|f_1(w)|)$ Schritte.

- Für die Berechnung von $f_1(w)$ braucht T_1 maximal $c_1 * p_1(|w|)$ Schritte. Da in jedem Schritt höchstens ein Zeichen geschrieben werden kann, folgt auch: $|f_1(w)| \leq c_1 * p_1(|w|)$.

- Folglich gilt
  - $time(f_2(f_1(w))) \leq c_2 * p_2(|f_1(w)|) \leq c_2 * p_2(c_1 * p_1(|w|)) =: q(|w|)$

- Damit ist $q(|w|)$ eine polynomiale Schranke für $f_2 \circ f_1$.

Darstellung logischer Formeln



- Literal:
  - Variable oder negierte Variable
    - Beispiele: x_1, $\neg x_2$, …
- Klausel
  - Disjunktion von Literalen
    - Beispiel: $x_1 \vee x_2 \vee \neg x_3$
  - 3-Klausel: Klausel mit höchstens 3 Variablen.
- Klauselmenge
  - entspricht Konjunktion von Klauseln
    - Beispiel: $\{ x_1 \vee x_2 \vee \neg x_3 , x_1 \vee x_2 , x_1 \vee \neg x_3 \}$
- Erinnerung:
  - Jede Aussagenlogische Formel ist äquivalent zu einer Klauselmenge

3-SAT

- Gegeben: Menge M von 3-Klauseln
- Gefragt : Ist M erfüllbar

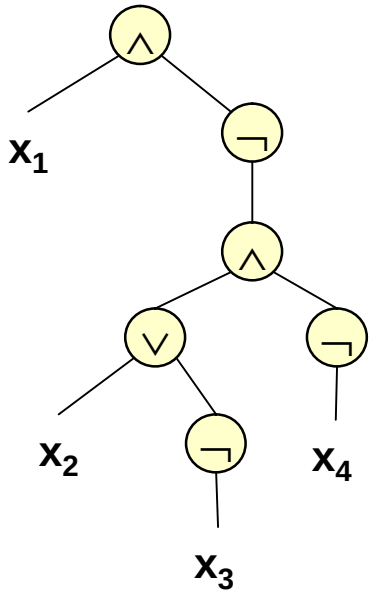
- SAT $\leq_p$ 3-SAT

- Beweis: Zu jeder Formel $F(x_1,\ldots,x_n)$ konstruiere (in höchstens polynomiell vielen Schritten) erfüllungsäquivalente Menge von 3-Klauseln.

- 1. Negationen nach innen
- 2. Formel zerschneiden:
  - $\wedge$: kein Problem: $F_1(x_1,\ldots,x_n) \wedge F_2(x_1,\ldots,x_n) \;\mapsto\; \{ F_1(x_1,\ldots,x_n),\; F_2(x_1,\ldots,x_n) \}$
    dann weiter mit F_1 und F_2
  - $\vee$: neue Variablen $z_1,z_2,\ldots$ an Schnittstellen.
    $F_1(x_1,\ldots,x_n) \vee F_2(x_1,\ldots,x_n) \mapsto \{z_1 \vee z_2,\; z_1 = F_1(x_1,\ldots,x_n),\; z_2=F_2(x_1,\ldots,x_n) \}$
  - dann jeweils
    $z = G_1 \vee G_2 \quad \mapsto \quad \{ \neg z \vee G_1 \vee G_2 ,\; \neg G_1 \vee z ,\; \neg G_2 \vee z \}$
    $z = G_1 \wedge G_2 \quad \mapsto \quad \{ \neg z \vee G_1 ,\; \neg z \vee G_2 ,\; \neg G_1 \vee \neg G_2 \vee z \}$
  - weiter mit diesen Klauseln

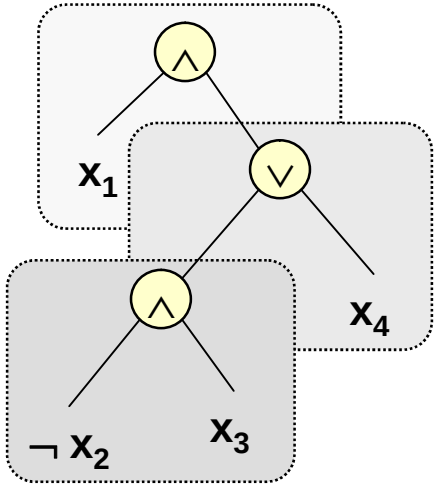
Beispiel: Rückführung auf 3-SAT

Ausgangsformel



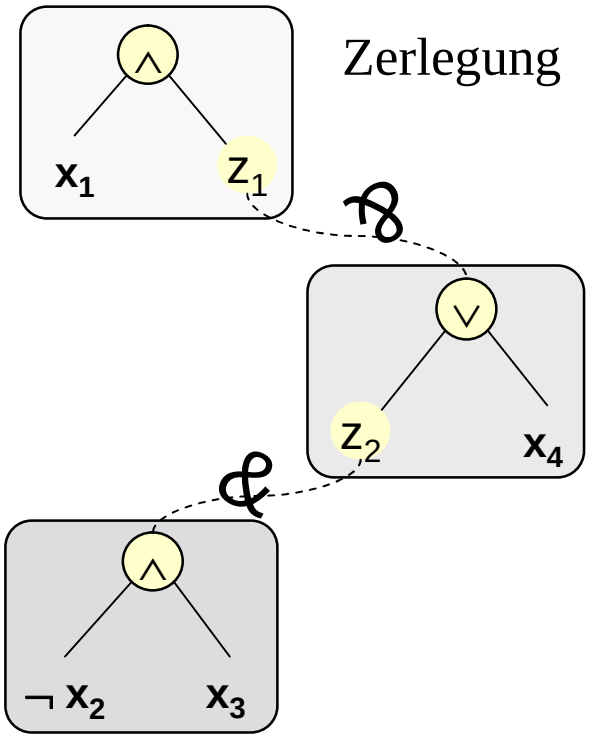
$x_1 \wedge \neg((x_2 \vee \neg x_3) \wedge \neg x_4)$

Negationen nach innen



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

Zerlegung

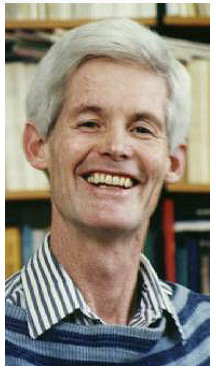


$\{ x_1 , z_1 ,$
$(z_1 \Leftrightarrow z_2 \vee x_4) ,$
$(z_2 \Leftrightarrow \neg x_2 \wedge x_3) \}$

erfüllungsäquivalente Menge von 3-Klauseln:

$\{ x_1 , z_1 , \neg z_1 \vee z_2 \vee x_4 , \neg z_2 \vee z_1 , \neg x_4 \vee z_1 ,$
$\neg z_2 \vee \neg x_2 , \neg z_2 \vee x_3 , x_2 \vee \neg x_3 \vee z_2 \}$

Satz von Cook



Stephen Cook
*1939

- Jedes Problem in NP ist polynomial auf SAT reduzierbar
  - Beweis: siehe später
- Korollar
  - SAT ist das schwerste Problem in NP
  - Wenn SAT effizientere Lösung hat, dann jedes Problem in NP

NP-hart

- Problem P ist NP-hart, wenn für jedes $Q \in NP$ gilt:
  $$Q \leq_p P$$
  d.h.
  - Jedes Problem in NP lässt sich auf P zurückführen
  - ein effizienter Algorithmus für P würde einen effizienten Algorithmus für jedes Problem in NP liefern
- Satz von Cook: SAT ist NP-hart
  - Folgerung: 3-SAT ist NP-hart
    - Beweis: Sei $P \in NP$. Nach Cook gilt $P \leq_p SAT$.
      Wegen $SAT \leq_p 3\text{-}SAT$ folgt $P \leq_p 3\text{-}SAT$

NP-vollständig

- $L \subseteq \Sigma^*$ heißt NP-vollständig, wenn
  - $L \in$ NP, und
  - L ist NP-hart
- SAT ist NP-vollständig
- Folgerung: 3-SAT ist NP-vollständig

  Beweis: Zu zeigen:
  - 3-SAT $\in$ NP
    - Guess: Wähle nichtdeterministisch eine Belegung
    - Check: Prüfe in polynomieller Zeit, ob es sich um eine Lösung handelt
  - 3-SAT ist NP-hart.
    - Wegen SAT $\leq_p$ 3-SAT

CLIQUE ist NP-vollständig

1. k-Clique ist in NP

   - Guess :
     - Rate Kandidaten für eine k-Clique
     - Es gibt $\binom{n}{k}$ viele Kandidaten
   - Check:
     - Überprüfe, ob es sich um eine Clique handelt
     - Das geht in polynomialer Zeit
       1. es sind maximal k*(k-1)/2 Kanten zu prüfen

2. k-Clique ist NP-hart

   - Strategie : Zeige 3-SAT $\leq_p$ k-Clique
     - Weil 3-SAT NP-hart ist, folgt die Behauptung

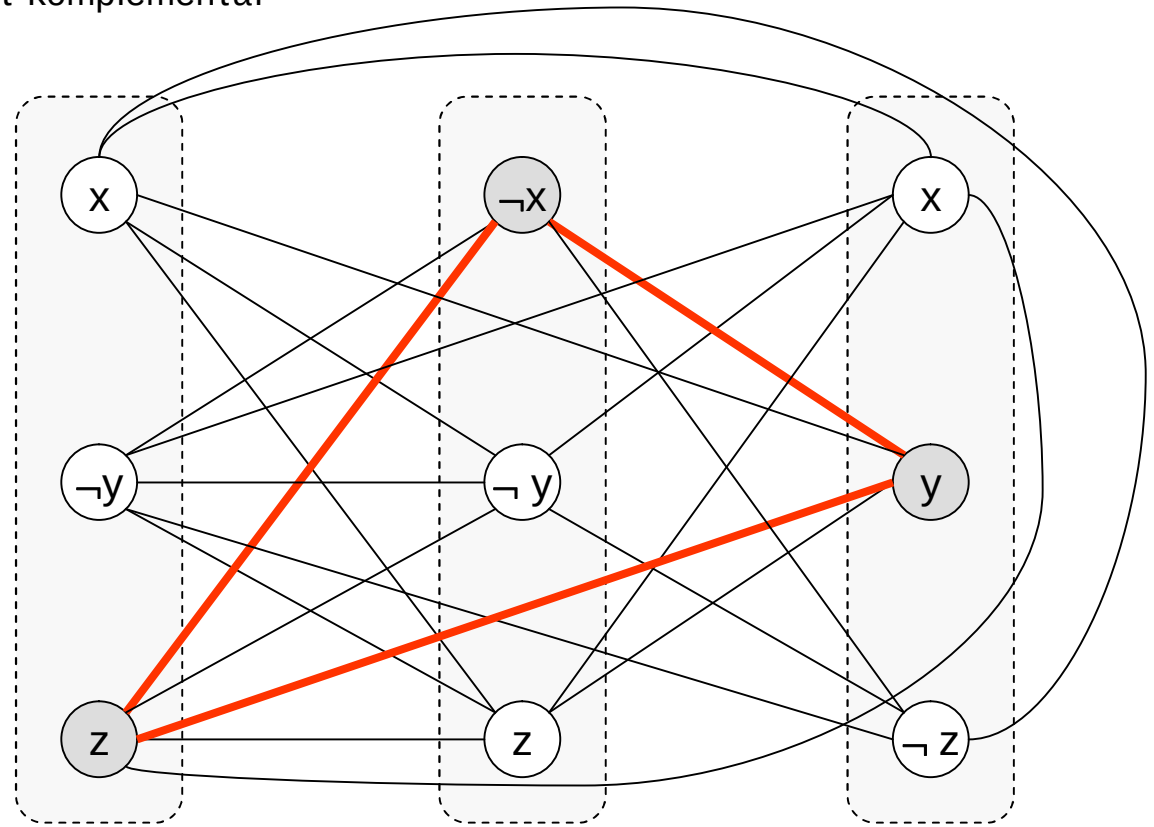
3-SAT $\leq_p$ CLIQUE

- Methode:
  - Codiere beliebiges 3-SAT Problem als CLIQUEN-problem
    - Wichtig: Übersetzung in polynomial vielen Schritten
- Idee
  - Gegeben Menge $M=\{K_1,\ldots,K_n\}$ von 3-Klauseln
  - Bastele Graphen G_M so dass
    - jede n-Clique entspricht einer erfüllenden Belegung und
    - jede erfüllende Belegung entspricht einer n-Clique
- Konstruktion:
  - Die Knoten:
    - Für jede Klausel K_i und jedes Literal L_{ij} in K_i einen Knoten (i,L_{ij})
      - also maximal 3*n viele Knoten
  - Die Kanten:
    - Verbinde (i,L_{ij}) mit (k,L_{kl}) falls
      - $(i \neq k)$ und (L_{ij} nicht komplementär zu L_{kl}).
  - Dann gilt:
    - Erfüllende Belegungen sind gerade n-Cliquen in diesem Graphen

Der Graph zu einer Klauselmenge

- Jede der n Spalten enthält die Literale einer Klausel
- Zwei Literale verbunden falls
  - in verschiedenen Klauseln und
  - nicht komplementär

- F hat erfüllende Belegung gdw.
- ⇔ es gibt eine Auswahl von Literalen
  - aus jeder Klausel eines
  - keine zwei widersprüchlich
- ⇔ es gibt eine Clique der Größe n



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

Praktische Anwendung von SAT

- Model Checking (Modellüberprüfung)
  - Gegeben ein endliches System
    - Hardware, Software, verteilte Anwendung,…
  - Gefragt: Erfüllt es eine bestimmte Eigenschaft
    - timing constraints, wechselseitiger Ausschluss, deadlock-Freiheit, …
- Symbolisches Model Checking:
  - Modelliere das System durch propositionale Formel S
  - Beschreibe die Eigenschaft durch propositionale Formel P
  - Zeige, dass $S \wedge \neg P$ erfüllbar/nicht erfüllbar ist

Transitionssysteme

- Allgemeines Modell zustandsbasierter Systeme
  - Zustandsmenge Q
  - Übergangsrelation $R \subseteq Q \times Q$
  - Anfangszustand $q_0 \in Q$
  - Endzustände $F \subseteq Q$

- Formal
  - $M = (Q, R, q_0, T)$ mit $R \subseteq Q \times Q$, $q_0 \in Q$, $F \subseteq Q$
  - Wir schreiben $p \rightarrow q$, falls $(p,q) \in R$

- Pfad: Endliche Folge $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_n$

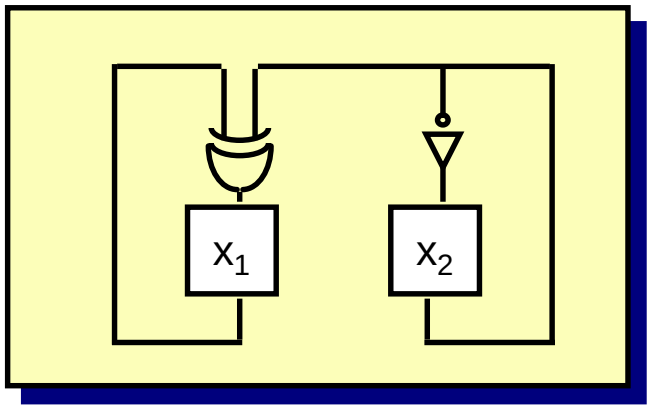
- Typische Fragen :
  - Gibt es einen Pfad vom Anfangszustand in einen (bestimmten) Endzustand

Turingmaschine als Transitionssystem

- Turingmaschinen kann man als Transitionssysteme auffassen:
  - Zustände: Konfigurationen
    - $\alpha\, q\, \beta$ mit $\alpha, \beta \in (\Sigma \cup \{\#\})^*$
  - Übergangsrelation
    - $\alpha\, q\, \beta \;|\!-\; \alpha'\, q'\, \beta'$ - gemäß der Turingtafel
  - Anfangszustand
    - $\varepsilon\; q_0\; w_1 \# \ldots \# w_n$
  - Endzustände
    - $\alpha\, q\, \beta$ mit $q \in T$ oder kein passender Eintrag in Turingtafel
- Halteproblem (für Input $w_1,\ldots,w_n$):
  - Gibt es einen Pfad von $\varepsilon\; q_0\; w_1 \# \ldots \# w_n$ in einen Endzustand ?

Hardware-Bausteine als Transitionssysteme

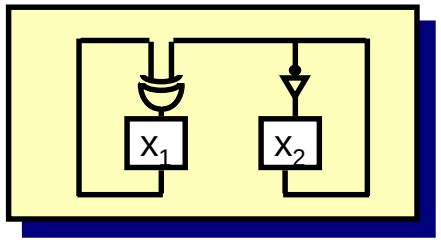
- Getaktete Schaltungen können als Transitionssysteme modelliert werden
  - Zustände : Bitvektoren
  - Transition : Veränderung in einem Takt
  - Anfangszustand, Endzustände : je nach Fragestellung
    - Zielzustände - sollen erreicht werden
    - Fehlerzustände – dürfen nicht erreicht werden

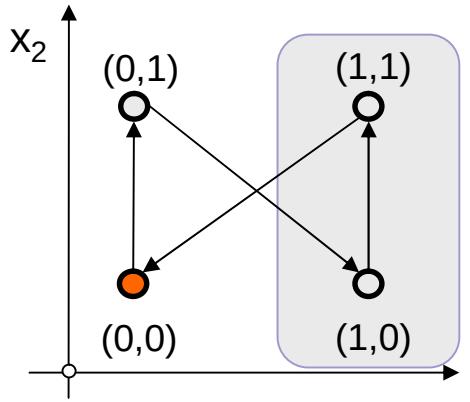


Eine Schaltung

Modellierung endlicher Transitionssysteme

- Endliche Transitionssysteme können durch Bitfolgen beschrieben werden
- Q endlich $\Rightarrow |Q| \leq 2^n$ für ein $n \in \mathbb{N}$
  - jedes $q \in Q$ entspricht einem Bitvektor $x = (x_1,\ldots,x_n) \in \{0,1\}^n$.
  - jeder Teilmenge von Q entspricht Menge von n-Bitvektoren
  - jedem Übergang $(x_1,\ldots,x_n) \rightarrow (y_1,\ldots,y_n)$ entspricht Bitvektor der Länge 2n
    - $(x_1,\ldots,x_n,y_1,\ldots,y_n) \in \{0,1\}^{2n}$
  - jeder Relation auf Q entspricht Menge von 2n-Bitvektoren





$Q = \{ (x_1,x_2) \mid x_i \in \{0,1\} \}$

$R = \{ (0,0,0,1), (0,1,1,0), (1,0,1,1), (1,1,0,0) \}$

und beispielsweise:

$q_0 = (0,0)$

$E = \{ (1,0), (1,1) \}$

Modellierung durch Formeln

- Jedem n-Bitvektor $b=(b_1,\ldots,b_n)$ entspricht $\wedge$-Klausel

$$K_b(x_1,\ldots,x_n)$$

  - $K_b(x_1,\ldots,x_n)=\text{True} \Leftrightarrow (x_1,\ldots,x_n)=(b_1,\ldots,b_n)$

- Jeder Menge von n-Bitvektoren $M=\{b_1,\ldots,b_m\}$ entspricht Disjunktion von Klauseln

$$F_M := K_{b1} \vee \ldots \vee K_{bm}$$

  - $FM(x_1,\ldots,x_n) = \text{True} \Leftrightarrow (x_1,\ldots,x_n) \in M$

- Jeder Relation R auf n-Bitvektoren entspricht eine Formel F_R mit 2n Variablen

$$F_R(x_1,\ldots,x_n,y_1,\ldots,y_n)$$

  - $F_R(x_1,\ldots,x_n,y_1,\ldots,y_n) = \text{True} \Leftrightarrow ((x_1,\ldots,x_n) , (y_1,\ldots,y_n)) \in R$

Beispiel (n=4):

$K_{(0,1,1,0)}$ =

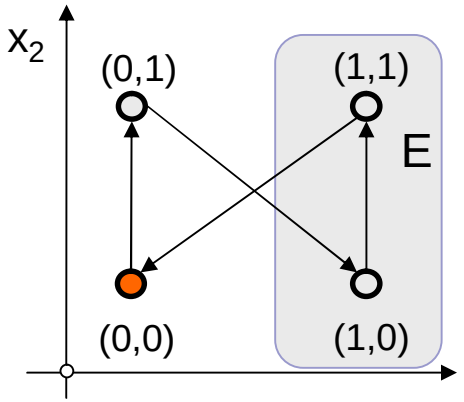
$$\neg x_1 \wedge x_2 \wedge x_3 \wedge \neg x_4$$

M={ (0,1,0,1), (1,1,0,0), (1,0,1,0),(0,0,1,1) }

$$F_M = (\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4)$$
$$\vee (x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4)$$
$$\vee (x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4)$$
$$\vee (\neg x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4)$$

Modell und Formeln

- Ein Modell und die Formeln, die es beschreiben



n = 2

$F_{q0}(x_1,x_2) = \neg x_1 \wedge \neg x_2$

$F_E(x_1,x_2) = (x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$

$$F_R(x_1,x_2,y_1,y_2) = (\neg x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge y_2) \vee (\neg x_1 \wedge x_2 \wedge y_1 \wedge \neg y_2) \vee (x_1 \wedge \neg x_2 \wedge y_1 \wedge y_2) \vee (x_1 \wedge x_2 \wedge \neg y_1 \wedge \neg y_2)$$

Optimierungen möglich, z.B.

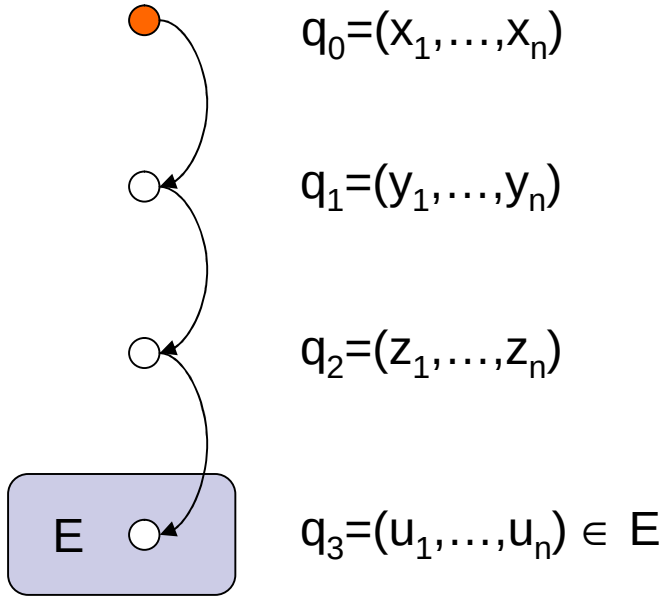
$F_E(x_1,x_2) = x_1$

$F_R(x_1,x_2,y_1,y_2) = (y_2 \Leftrightarrow \neg x_2) \wedge (y_1 \Leftrightarrow ((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)))$

Beschreibung von Pfaden

- Es gibt einen Pfad der Länge 3 von q_0 in ein $q \in E$ falls folgende Formel erfüllbar ist:

$$
\begin{aligned}
& F_{q0}(x_1,\ldots,x_n) \\
1 \quad & \wedge F_R(x_1,\ldots,x_n,y_1,\ldots,y_n) \\
2 \quad & \wedge F_R(y_1,\ldots,y_n,z_1,\ldots,z_n) \\
3 \quad & \wedge F_R(z_1,\ldots,z_n,u_1,\ldots,u_n) \\
& \wedge F_E(u_1,\ldots,u_n)
\end{aligned}
$$



Das funktioniert natürlich analog für jedes feste k.

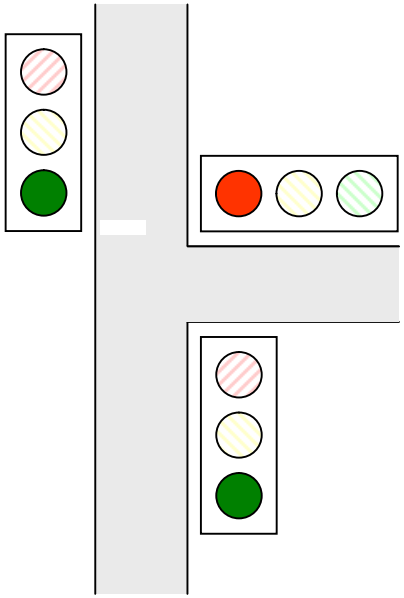
k-Pfade in Transitionssystemen

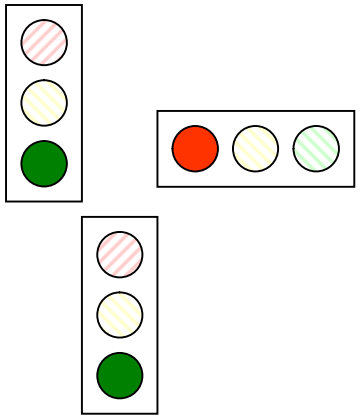
- Sei $T=(Q,R,q_0,E)$ ein endliches Transitionensystem mit $|Q| \leq 2^n$
- Satz: Für jede natürliche Zahl k kann man eine aussagenlogische Formel P_k mit n*k vielen Variablen konstruieren, so dass gilt

  P_k erfüllbar $\Leftrightarrow$ Es existiert ein Pfad der Länge k von q_0 nach E.

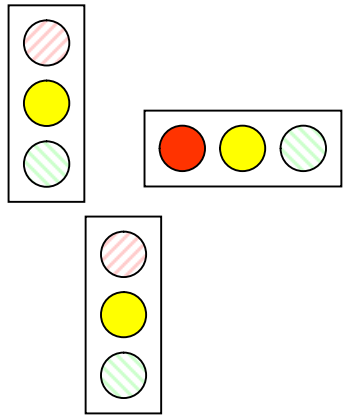
Praxisbeispiel: Ampelschaltungen

- Endlich viele Zustände gegeben durch Kombinationen der Ampelzustände
- Verifiziere: Es gibt keinen Pfad in einen verbotenen Zustand

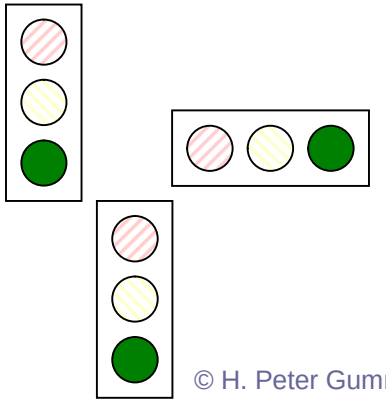




ein möglicher Übergang



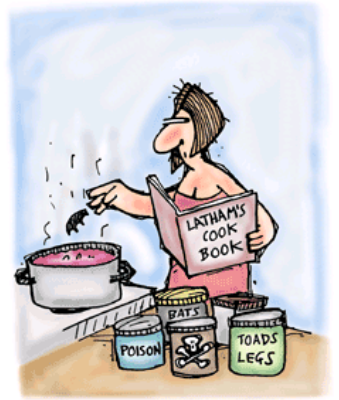
ein verbotener Zustand :



Der Satz von Cook - Beweisidee

- SAT ist NP-vollständig.
  - Schon bekannt: SAT ist in NP.
  - Zeige: SAT ist NP-hart
- Sei P ein beliebiges Problem in NP, d.h.
  - L eine entscheidbare Sprache und das Problem „$w \in L$?“ in NP
  - Es gibt ein Polynom p und eine NDTM T die „$w \in L$?“ in $p(|w|)$ Schritten entscheidet.
  - $p(|w|)$ liefert obere Schranke für
    - die Anzahl der Schritte
    - die Anzahl der besuchten Bandzellen

Konstruktion des Transitionssystems



- Wir konstruieren in polynomial vielen Schritten zur TM $T=(Q,\Sigma,\delta,q_0,F)$ ein endliches Transitionssystem:
  - Zustände:
    - Konfigurationen $\alpha\, q\, \beta$ - maximal $p(|w|) * |Q| * p(|w|)$ viele
  - Übergangsrelation:
    - $\alpha\, q\, \beta \;|-\; \alpha'\, q'\, \beta'$ - gemäß der Turingtafel
  - Anfangszustand:
    - $\varepsilon\, q_0\, w$
  - Endzustände
    - Jedes $\alpha\, q\, \beta$ mit $q \in F$.
- Es gilt : $w \in L \Leftrightarrow$ Es gibt einen Pfad der Länge $p(|w|)$ von q_0 zu einem Endzustand
- Wir konstruieren die zugehörige Pfadformel P, so dass gilt:

$w \in L \Leftrightarrow P$ ist erfüllbar