Script generated by TTT

Title: seidl: Theoretische_Informatik (05.07.2012)

Date: Thu Jul 05 16:02:51 CEST 2012

Duration: 87:22 min

Pages: 107

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:
Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:
Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.
$M \mapsto$ GOTO $\mapsto$ WHILE:

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:
Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.
$M \mapsto$ GOTO $\mapsto$ WHILE:

```
pc := 1;
WHILE pc ≠ 0 DO
  IF pc = 1 THEN P_1 ELSE
  ...
  IF pc = k THEN P_k ELSE pc := 0
END
```

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:
Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.
$M \mapsto$ GOTO $\mapsto$ WHILE:

```
pc := 1;
WHILE pc ≠ 0 DO
  IF pc = 1 THEN P_1 ELSE
  ...
  IF pc = k THEN P_k ELSE pc := 0
END
```

wobei jeder Schleifendurchlauf einen Schritt von M simuliert.

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:
Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.
$M \mapsto$ GOTO $\mapsto$ WHILE:

```
pc := 1;
WHILE pc ≠ 0 DO
  IF pc = 1 THEN P_1 ELSE
  ...
  IF pc = k THEN P_k ELSE pc := 0
END
```

wobei jeder Schleifendurchlauf einen Schritt von M simuliert.
Es kann maximal $f(|w|)$ viele Schleifendurchläufe geben.
Ersetze `WHILE` $pc \neq 0$ `DO` durch $i := f(|w|)$; `LOOP` i `DO`.

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:
Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.
$M \mapsto$ GOTO $\mapsto$ WHILE:

```
pc := 1;
WHILE pc ≠ 0 DO
  IF pc = 1 THEN P_1 ELSE
  ...
  IF pc = k THEN P_k ELSE pc := 0
END
```

wobei jeder Schleifendurchlauf einen Schritt von M simuliert.
Es kann maximal $f(|w|)$ viele Schleifendurchläufe geben.
Ersetze `WHILE` $pc \neq 0$ `DO` durch $i := f(|w|)$; `LOOP` i `DO`.
Berechnung wie vorher, aber evtl mit zusätzlichen Durchläufen mit $pc = 0$. □

Satz 5.11
Ist $A \in TIME(f)$ und ist f LOOP-berechenbar,
dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:
Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.
$M \mapsto$ GOTO $\mapsto$ WHILE:

pc := 1;
WHILE $pc \neq 0$ DO
  IF $pc = 1$ THEN P_1 ELSE
  ...
  IF $pc = k$ THEN P_k ELSE $pc := 0$
END

wobei jeder Schleifendurchlauf einen Schritt von M simuliert.
Es kann maximal $f(|w|)$ viele Schleifendurchläufe geben.
Ersetze WHILE $pc \neq 0$ DO durch $i := f(|w|)$; LOOP i DO.
Berechnung wie vorher, aber evtl mit zusätzlichen Durchläufen mit $pc = 0$. □

Da $+$ und $*$ LOOP-berechenbar sind, folgt:

Korollar 5.12
Alle Sprachen in P sind LOOP-entscheidbar.

Da sich jede polynomiell zeitbeschränkte NTM (Zeit $O(p(n))$)
in eine exponentiell zeitbschränkte DTM (Zeit $O(c^{p(n)})$)

Da $+$ und $*$ LOOP-berechenbar sind, folgt:

Korollar 5.12
Alle Sprachen in P sind LOOP-entscheidbar.

Da sich jede polynomiell zeitbeschränkte NTM (Zeit $O(p(n))$)
in eine exponentiell zeitbschränkte DTM (Zeit $O(c^{p(n)})$)
umwandeln lässt und $c^{p(n)}$ LOOP-berechenbar ist, folgt:

Korollar 5.13
Alle Sprachen in NP sind LOOP-entscheidbar.

5.3 NP-Vollständigkeit

5.3 NP-Vollständigkeit

5.3 NP-Vollständigkeit

1. Polynomielle Reduzierbarkeit $\leq_p$

5.3 NP-Vollständigkeit

1. Polynomielle Reduzierbarkeit $\leq_p$
2. NP-vollständige Probleme = härteste Probleme in NP, alle anderen Probleme in NP darauf polynomiell reduzierbar
3. Satz: SAT ist NP-vollständig

Definition 5.14
Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.
Dann heißt A polynomiell reduzierbar auf B, $A \leq_p B$,

Definition 5.14
Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.
Dann heißt A polynomiell reduzierbar auf B, $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt,

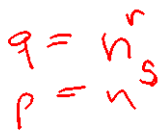
Definition 5.14
Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.
Dann heißt A polynomiell reduzierbar auf B, $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in A \iff f(w) \in B$$

Definition 5.14
Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.
Dann heißt A polynomiell reduzierbar auf B, $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in A \iff f(w) \in B$$

Da $q(p(n))$ ein Polynom ist falls p und q Polynome sind:



Definition 5.14
Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.
Dann heißt A polynomiell reduzierbar auf B, $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit
berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in A \iff f(w) \in B$$

Da $q(p(n))$ ein Polynom ist falls p und q Polynome sind:

Lemma 5.15
Die Relation $\leq_p$ ist transitiv.

Lemma 5.16
Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Lemma 5.16
Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f, die von DTM M_f berechnet wird.

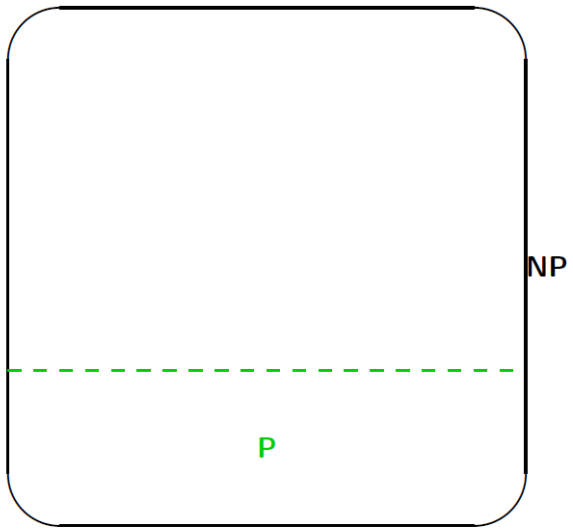
Ein Problem ist NP-hart,
wenn es mindestens so hart wie alles in NP ist:

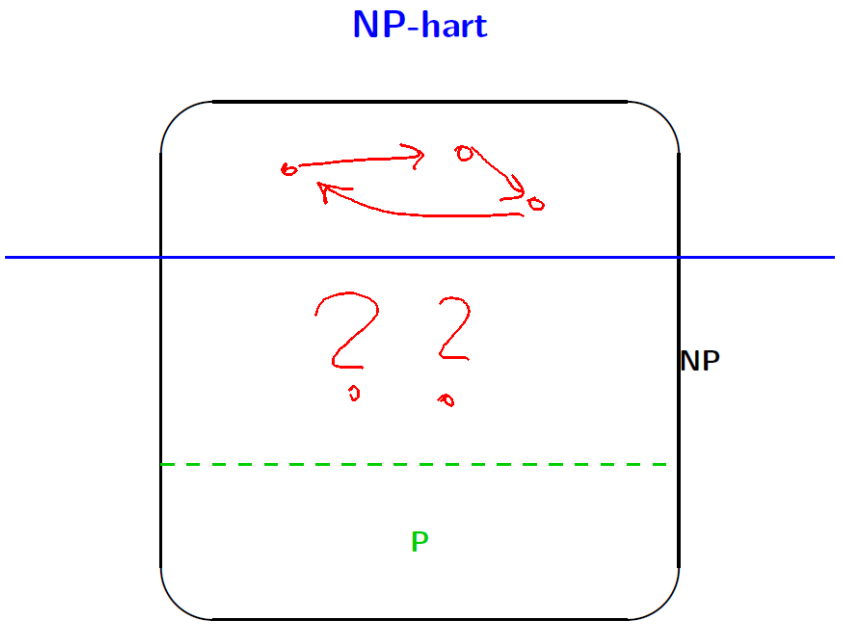
Definition 5.17
Eine Sprache L heißt NP-hart gdw $A \leq_p L$ für alle $A \in$ NP.

Ein Problem ist NP-hart,
wenn es mindestens so hart wie alles in NP ist:

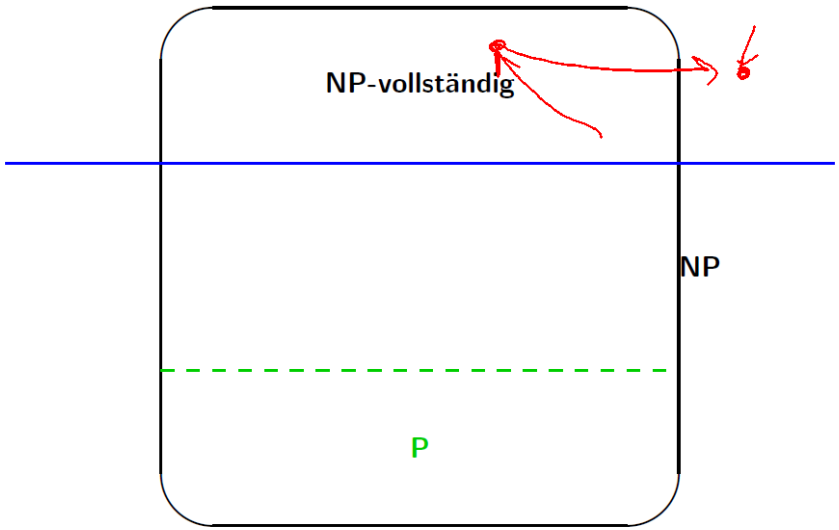
Definition 5.17
Eine Sprache L heißt NP-hart gdw $A \leq_p L$ für alle $A \in$ NP.

Definition 5.18
Eine Sprache L heißt NP-vollständig gdw
L NP-hart ist und $L \in$ NP.





NP-hart



Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 5.19
Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 5.19
Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Beweis:
„⇒“: Falls P=NP, so liegt jedes NP-vollständige Problem in P.

Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 5.19
Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Beweis:
„⇒“: Falls P=NP, so liegt jedes NP-vollständige Problem in P.

„⇐“: Sei L ein NP-vollständiges Problem in P.
Dann gilt P $\supseteq$ NP:
Ist $A \in$ NP, so gilt $A \leq_p L$ (da L NP-hart)
und nach Lemma 5.16 $A \in$ P (da $L \in$ P). □

Starke Vermutung:

- P $\neq$ NP
- dh kein NP-vollständiges Problem ist in P.

Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 5.19
Es gilt P=NP gdw ein NP-vollständiges Problem in P liegt.

Beweis:
„$\Rightarrow$“: Falls P=NP, so liegt jedes NP-vollständige Problem in P.

„$\Leftarrow$“: Sei L ein NP-vollständiges Problem in P.
Dann gilt P $\supseteq$ NP:
Ist $A \in$ NP, so gilt $A \leq_p L$ (da L NP-hart)
und nach Lemma 5.16 $A \in$ P (da $L \in$ P). □

Starke Vermutung:

- P $\neq$ NP
- dh kein NP-vollständiges Problem ist in P.

Aber gibt es überhaupt NP-vollständige Probleme?

Aussagenlogik

Syntax der Aussagenlogik:

$$\text{Formeln:} \quad F \rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X$$

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{aligned} \text{Formeln:} \quad & F \rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \\ \text{Variablen:} \quad & X \rightarrow x \mid y \mid z \mid \ldots \end{aligned}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und $\wedge$ bindet stärker als $\vee$: $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{aligned} \text{Formeln:} \quad & F \rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \\ \text{Variablen:} \quad & X \rightarrow x \mid y \mid z \mid \ldots \end{aligned}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und $\wedge$ bindet stärker als $\vee$: $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Semantik der Aussagenlogik:

- Eine Belegung ist eine Funktion von Variablen auf $\{0, 1\}$.

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{array}{rrcl} \text{Formeln:} & F & \to & \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \\ \text{Variablen:} & X & \to & x \mid y \mid z \mid \ldots \end{array}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und $\wedge$ bindet stärker als $\vee$: $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Semantik der Aussagenlogik:

- Eine Belegung ist eine Funktion von Variablen auf $\{0, 1\}$.
  Bsp: $\sigma = \{x \mapsto 0, y \mapsto 1, z \mapsto 0, \ldots\}$
- Belegungen werde mittels Wahrheitstabellen auf Formeln erweitert.

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{array}{rrcl} \text{Formeln:} & F & \to & \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \\ \text{Variablen:} & X & \to & x \mid y \mid z \mid \ldots \end{array}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und $\wedge$ bindet stärker als $\vee$: $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Semantik der Aussagenlogik:

- Eine Belegung ist eine Funktion von Variablen auf $\{0, 1\}$.
  Bsp: $\sigma = \{x \mapsto 0, y \mapsto 1, z \mapsto 0, \ldots\}$
- Belegungen werde mittels Wahrheitstabellen auf Formeln erweitert. Bsp: $\sigma((\neg x \wedge y) \vee (x \wedge \neg z)) = 1$

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{array}{rrcl} \text{Formeln:} & F & \to & \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \\ \text{Variablen:} & X & \to & x \mid y \mid z \mid \ldots \end{array}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und $\wedge$ bindet stärker als $\vee$: $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Semantik der Aussagenlogik:

- Eine Belegung ist eine Funktion von Variablen auf $\{0, 1\}$.
  Bsp: $\sigma = \{x \mapsto 0, y \mapsto 1, z \mapsto 0, \ldots\}$
- Belegungen werde mittels Wahrheitstabellen auf Formeln erweitert. Bsp: $\sigma((\neg x \wedge y) \vee (x \wedge \neg z)) = 1$
- Eine Formel F ist erfüllbar gdw es eine Belegung σ gibt mit $\sigma(F) = 1$ $\exists \sigma. \sigma(F) = 1$ $\forall \sigma. \sigma(F) = 1$

SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

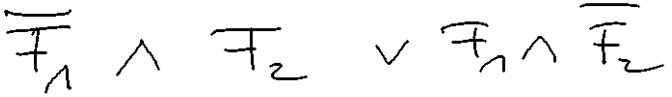
SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen



SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

Fakt 5.20

Zwei Formeln F_1 und F_2 sind genau dann äquivalent ($F_1 \leftrightarrow F_2$), wenn $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar ist.

SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

Fakt 5.20

Zwei Formeln F_1 und F_2 sind genau dann äquivalent ($F_1 \leftrightarrow F_2$), wenn $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar ist.

Lemma 5.21

SAT $\in$ NP

SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

Fakt 5.20

Zwei Formeln F_1 und F_2 sind genau dann äquivalent ($F_1 \leftrightarrow F_2$), wenn $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar ist.

Lemma 5.21

SAT $\in$ NP

Beweis:

Belegungen sind Zertifikate, die in polynomieller Zeit geprüft werden können: Es gibt eine DTM, die bei Eingabe einer Formel F und einer Belegung σ für die Variablen in F, in polynomieller Zeit $\sigma(F)$ berechnet. □

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:
Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:
Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit $\mathit{ntime}_M(w) \leq p(|w|)$.

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:
Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit $\mathit{ntime}_M(w) \leq p(|w|)$.
Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:
Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit $\mathit{ntime}_M(w) \leq p(|w|)$.
Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.
In polynomieller Zeit.

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit $ntime_M(w) \leq p(|w|)$.
Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.
In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit $ntime_M(w) \leq p(|w|)$.
Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.
In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Die Variablen beschreiben das *mögliche* Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \dots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit $ntime_M(w) \leq p(|w|)$.
Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.
In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Die Variablen beschreiben das *mögliche* Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \dots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q

$pos_{t,i}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots p(n)$	$pos_{t,i} = 1 \Leftrightarrow$ Kopfposition nach t Schritten ist i

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-hart.
Sei $A \in$ NP. Wir zeigen $A \leq_p$ SAT.
Da $A \in$ NP gibt es NTM M mit $A = L(M)$ und
Polynom p mit $ntime_M(w) \leq p(|w|)$.
Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.
In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Die Variablen beschreiben das *mögliche* Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \dots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q
$pos_{t,i}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots p(n)$	$pos_{t,i} = 1 \Leftrightarrow$ Kopfposition nach t Schritten ist i
$band_{t,i,a}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \Leftrightarrow$ Bandinhalt nach t Schritten auf Bandposition i ist Zeichen a

Beweis (Forts.):

Sei $Q = \{q_1, \dots, q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

Beweis (Forts.):

Sei $Q = \{q_1, \dots, q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$
$$R := \bigwedge_t [G(zust_{t,q_1}, \dots, zust_{t,q_k})$$

Beweis (Forts.):

Sei $Q = \{q_1, \dots, q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$
$$R := \bigwedge_t [G(zust_{t,q_1}, \dots, zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, \dots, pos_{t,p(n)})$$

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

$$R := \bigwedge_t [G(zust_{t,q_1}, \ldots, zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \wedge \bigwedge_i G(band_{t,i,a_1}, \ldots, band_{t,i,a_l})]$$

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

$$R := \bigwedge_t [G(zust_{t,q_1}, \ldots, zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \wedge \bigwedge_i G(band_{t,i,a_1}, \ldots, band_{t,i,a_l})]$$

$$A := zust_{0,q_1}$$

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

$$R := \bigwedge_t [G(zust_{t,q_1}, \ldots, zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \wedge \bigwedge_i G(band_{t,i,a_1}, \ldots, band_{t,i,a_l})]$$

$$A := zust_{0,q_1} \wedge pos_{0,1}$$

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

$$R := \bigwedge_t [G(zust_{t,q_1}, \ldots, zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \wedge \bigwedge_i G(band_{t,i,a_1}, \ldots, band_{t,i,a_l})]$$

$$A := zust_{0,q_1} \wedge pos_{0,1} \wedge \bigwedge_{j=1}^{n} band_{0,j,x_j}$$

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$
\begin{aligned}
F &:= R \wedge A \wedge T_1 \wedge T_2 \wedge E \\
R &:= \bigwedge_t [G(zust_{t,q_1}, \ldots, zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \wedge \\
&\qquad \bigwedge_i G(band_{t,i,a_1}, \ldots, band_{t,i,a_l})] \\
A &:= zust_{0,q_1} \wedge pos_{0,1} \wedge \bigwedge_{j=1}^{n} band_{0,j,x_j} \wedge \\
&\qquad \bigwedge_{j=-p(n)}^{0} band_{0,j,\square} \wedge \bigwedge_{j=n+1}^{p(n)} band_{0,j,\square}
\end{aligned}
$$

Beweis (Forts.):

$$
T_1 := \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a}
$$

Beweis (Forts.):

$$
\begin{aligned}
T_1 &:= \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\
&\qquad \rightarrow \bigvee_{(q',a',y) \in \delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})]
\end{aligned}
$$

Beweis (Forts.):

$$
\begin{aligned}
T_1 &:= \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\
&\qquad \rightarrow \bigvee_{(q',a',y) \in \delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})] \\
T_2 &:= \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a})
\end{aligned}
$$

Beweis (Forts.):

$$
\begin{aligned}
T_1 &:= \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\
&\quad \rightarrow \bigvee_{(q',a',y)\in\delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})] \\
T_2 &:= \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a}) \\
E &:= \bigvee_{t} \bigvee_{q\in F} zust_{t,q}
\end{aligned}
$$

Von der Lösbarkeit zur Lösung

Beweis (Forts.):

$$
\begin{aligned}
T_1 &:= \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\
&\quad \rightarrow \bigvee_{(q',a',y)\in\delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})] \\
T_2 &:= \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a}) \\
E &:= \bigvee_{t} \bigvee_{q\in F} zust_{t,q}
\end{aligned}
$$

$$
G(v_1,\ldots,v_r) := (\bigvee_{i=1}^{r} v_i) \wedge \left(\bigwedge_{i=1}^{r-1} \bigwedge_{j=i+1}^{r} \neg(v_i \wedge v_j) \right)
$$

□

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1,\ldots,x_k$:

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

wobei $F[x := b] = F$ mit x ersetzt durch b.

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

wobei $F[x := b]$ = F mit x ersetzt durch b.

Entscheidung von SAT in Zeit $O(f(n))$

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

wobei $F[x := b]$ = F mit x ersetzt durch b.

Entscheidung von SAT in Zeit $O(f(n))$

$\Longrightarrow$ Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ SAT **then** $b := 0$ **else** $b := 1$
    output(x_i "=" b)
    $F := F[x_i := b]$

wobei $F[x := b]$ = F mit x ersetzt durch b.

Entscheidung von SAT in Zeit $O(f(n))$
$\Longrightarrow$ Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$

(falls es eine gibt.)

$f(n)$ polynomiell $\Longrightarrow$ $n \cdot (f(n) + n)$ polynomiell

Bemerkungen:

- Die Reduzierung der Lösungsberechnung auf SAT ist eine rein theoretische Konstruktion.

Bemerkungen:

- Die Reduzierung der Lösungsberechnung auf SAT ist eine rein theoretische Konstruktion.
- Sie zeigt, dass man sich auf SAT beschränken kann, wenn man nur an polynomiell/exponentiell interessiert ist.
- Alle bekannten Entscheidungsverfahren für SAT berechnen auch eine Lösung.

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 10 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*):

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 10 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*): `http://satcompetition.org`

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 10 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von
  SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 10 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von
  SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

- Jetzt:

  NP-vollständig = Hoffnung durch SAT

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 10 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von
  SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

- Jetzt:

  NP-vollständig = Hoffnung durch SAT

- Paradigma:

  SAT (Logik!) als universelle Sprache
  zur Kodierung kombinatorischer Probleme

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 10 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von
  SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

- Jetzt:

  NP-vollständig = Hoffnung durch SAT

- Paradigma:

  SAT (Logik!) als universelle Sprache
  zur Kodierung kombinatorischer Probleme

Reduktion auf SAT manchmal schneller als problemspezifische Löser!

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 10 Jahren:
  Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*): `http://satcompetition.org`
  Stand der Kunst: Erfüllbar: bis 10^5 Variablen
  Unerfüllbar: bis 10^3 Variablen

- Jetzt:

  NP-vollständig = Hoffnung durch SAT

- Paradigma:

  SAT (Logik!) als universelle Sprache
  zur Kodierung kombinatorischer Probleme

  Reduktion auf SAT manchmal schneller als problemspezifische Löser! Und fast immer einfacher.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3})$$

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Bemerkungen

- Zeigt 3COL $\leq_p$ SAT und damit 3COL $\in$ NP.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Bemerkungen

- Zeigt 3COL $\leq_p$ SAT und damit 3COL $\in$ NP.
- Zeigt nicht, dass 3COL NP-vollständig ist.

Ähnlich direkt polynomiell auf SAT reduzierbar:

- HAMILTON
- SUDOKU
- ...

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

> *Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?*

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

> *Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?*

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

> *Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?*

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit:

Variable	$=$	Knoten

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

> *Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?*

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit:

Variable	$=$	Knoten
u und v verbunden	$=$	$u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit:

Variable	$=$	Knoten
u und v verbunden	$=$	$u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind
Farbe	$=$	Register

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit:

Variable	$=$	Knoten
u und v verbunden	$=$	$u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind
Farbe	$=$	Register
k-Färbung	$=$	Zuordnung eines Registers zu jeder Variablen

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit:

Variable	$=$	Knoten
u und v verbunden	$=$	$u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind
Farbe	$=$	Register
k-Färbung	$=$	Zuordnung eines Registers zu jeder Variablen

Sowohl k-Färbbarkeit als auch Registerverteilung ist für $k \geq 3$ NP-vollständig.

Eine Anwendung von k-Färbbarkeit: **Registerverteilung**

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Variable ist an einem Punkt lebendig
gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k-Färbbarkeit:

Variable	$=$	Knoten
u und v verbunden	$=$	$u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind
Farbe	$=$	Register
k-Färbung	$=$	Zuordnung eines Registers zu jeder Variablen

Sowohl k-Färbbarkeit als auch Registerverteilung ist für $k \geq 3$ NP-vollständig.

Mehr Information: Vorlesung *Programmoptimierung*