Beispiel für eine Turing Maschine (TM) M

$p \in K,$	$\sigma \in \Sigma$	$\delta(p,\sigma)$
$s,$	0	$(s, 0, \rightarrow)$
$s,$	1	$(s, 1, \rightarrow)$
$s,$	$\sqcup$	$(q, \sqcup, \leftarrow)$
$s,$	$\triangleright$	$(s, \triangleright, \rightarrow)$
$q,$	0	$(h, 1, -)$
$q,$	1	$(q, 0, \leftarrow)$
$q,$	$\triangleright$	$(h, \triangleright, \rightarrow)$

zB Eingabe $x = 1011$: $(s, \triangleright, 1011) \stackrel{M}{\rightarrow} (s, \triangleright 1, 011) \stackrel{M}{\rightarrow}$ $(s, \triangleright 10, 11) \stackrel{M}{\rightarrow} (s, \triangleright 101, 1) \stackrel{M}{\rightarrow} (s, \triangleright 1011, \varepsilon) \stackrel{M}{\rightarrow}$ $(s, \triangleright 1011\sqcup, \varepsilon) \stackrel{M}{\rightarrow} (q, \triangleright 1011, \sqcup) \stackrel{M}{\rightarrow} (q, \triangleright 101, 0\sqcup) \stackrel{M}{\rightarrow}$ $(q, \triangleright 10, 00\sqcup) \stackrel{M}{\rightarrow} (h, \triangleright 11, 00\sqcup)$, d.h. die Ausgabe ist 1100.

Was macht diese TM? Was könnte man noch erweitern?

Linearer Geschwindigkeitsgewinn von TMs

- Durch Verarbeitung mehrerer Symbole pro Schritt kann eine TM beliebig linear beschleunigt werden (allerdings braucht man mehr Zustände und Symbole).
- Es kommt nur auf Wachstumsrate an, $\rightarrow O(f(n))$, nicht aber auf multiplikative oder additive Konstante.
- $\sim$ Realität: schnellere Hardware.
- Falls L polynomiell entscheidbar, dann

$$\exists\, k \in \mathbb{N} \quad L \in \textbf{TIME}(n^k).$$

Definition: Die Menge aller Sprachen, die in polynomieller Zeit durch TMs entscheidbar sind, ist

$$\textbf{P} = \bigcup_{k>0} \textbf{TIME}(n^k).$$

Random Access Maschinen

- RAMs ähneln realen Maschinensprachen,
- es gibt also Register und einen Akkumulator, direkte und indizierte Adressierung, „primitive“ Operationen wie Addition, Division durch zwei (shift-right) sowie bedingte und unbedingte Verzweigung, allerdings
- wird mit unbeschränkt großen Integers gerechnet,
- dafür gibt es aber keine Multiplikation.

Theorem: RAM kann TM in $O(f(n))$ simulieren.
Theorem: 7$ TM kann RAM in $O(f(n)^3)$ simulieren.
$\Rightarrow$ TM kann RAM in $O(f(n)^6)$ simulieren.

$\rightarrow$ Bezüglich **P** sind TM und RAMs gleichwertig.

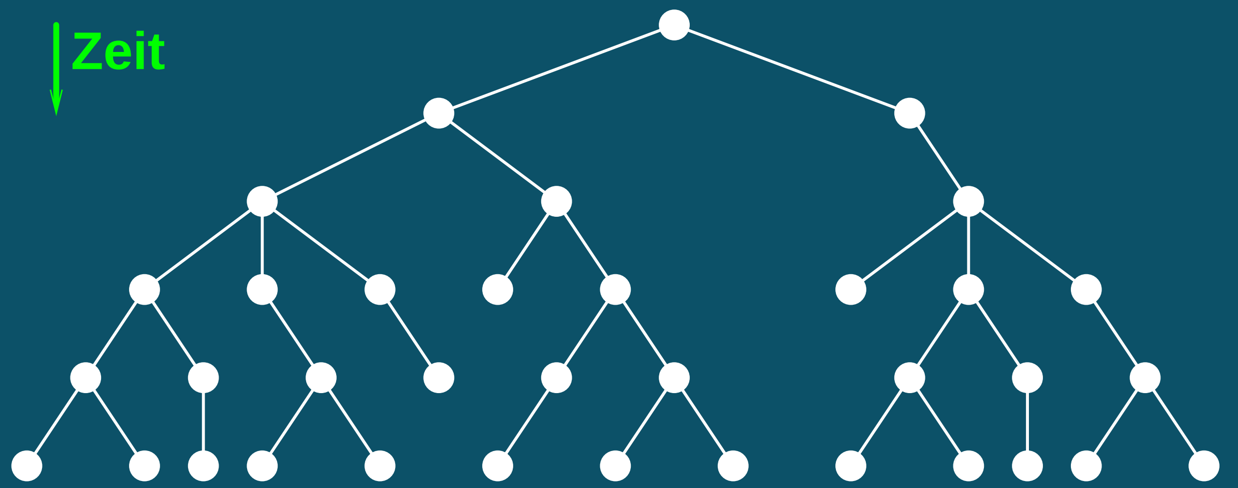
Nichtdeterministische Turing Maschinen (NTM)

Definition: Eine NTM ist ein Quadrupel (K, Σ, Δ, s) wie eine normale TM, außer daß Δ eine Relation (anstelle einer Funktion) ist:

$$\Delta \subset (K \times \Sigma) \times [(K \cup \{h, \text{„ja“}, \text{„nein“}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}]$$

Definition: Eine NTM N entscheidet eine Sprache L in Zeit $f(n)$ genau dann wenn $\forall\, x \in \Sigma^* \quad [x \in L \Leftrightarrow$

$$\exists w, u \in \Sigma^* \;\; \exists k \in \mathbb{N} \quad k \leq f(|x|) \wedge (s, \triangleright, x) \xrightarrow{N^k} (\text{„ja“}, w, u)]$$



▷

Nichtdeterministische Turing Maschinen (cont)

- NTM sind zwar unrealistisch, dafür aber praktisch zum Charakterisieren realistischer Probleme (TSP, . . .).
- Die Menge aller Sprachen, die in polynomieller Zeit durch NTMs entscheidbar sind, ist

$$\mathbf{NP} = \bigcup_{k>0} \mathbf{NTIME}(n^k) \ .$$

- Da Funktionen auch Relationen sind, gilt **P** $\subseteq$ **NP**.

Theorem: TM kann NTM in $O(c^{f(n)})$ simulieren.

- Geht es auch polynomiell, d.h. ist **P** $\stackrel{?}{=}$ **NP** ???
- Weiterer Grund für Bedeutung des **P**-**NP** Problems: schöne Strukturtheorie (Cook 1971, Karp 1972, Levin 1973). Zentraler Begriff: **NP**-Vollständigkeit.

NP-Vollständigkeit

$\rightarrow$ Die meisten **NP**-Probleme, für die kein polynomieller Algorithmus bekannt ist, sind so miteinander verknüpft, daß entweder alle polynomiell sind (falls **P** $=$ **NP**) oder keines (falls **P** $\neq$ **NP**).

Definition: Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen. Dann ist A auf B polynomiell reduzierbar, symbolisch mit $A \leq_p B$ falls es eine totale und in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \longrightarrow \Gamma^*$ gibt, so daß

$$\forall x \in \Sigma^* \quad (x \in A \Leftrightarrow f(x) \in B) \ .$$

Bemerkung: $\leq_p$ ist eine transitive Relation auf Sprachen.

Lemma: Falls $A \leq_p B$ und $B \in$ **P** (bzw. $B \in$ **NP**), so gilt auch $A \in$ **P** (bzw. $A \in$ **NP**).

$\triangleright$

NP-Vollständigkeit (cont)

Definition: Eine Sprache A heißt **NP-hart**, falls

$$\forall L \in \mathbf{NP} \quad L \leq_p A \, .$$

Definition: Eine Sprache heißt **NP-vollständig**, falls sie in **NP** liegt (membership) und **NP**-hart ist (hardness).

Theorem: Sei A **NP**-vollständig. Dann gilt

$$A \in \mathbf{P} \Leftrightarrow \mathbf{P} = \mathbf{NP} \, .$$

$\rightarrow$ Zum Nachweis von $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \neq \mathbf{NP}$ würde es also genügen, von irgendeinem **NP**-vollständigen Problem A zu zeigen, daß $A \in \mathbf{P}$, oder daß $A \notin \mathbf{P}$.

- **NP**-vollständige Probleme sind gewissermaßen die schwierigsten Probleme in **NP**.
- Die allgemeine Erwartung ist $\mathbf{P} \neq \mathbf{NP}$ (aber zB Gödel).

SAT: ein erstes NP-vollständiges Problem

Definition: Das Erfüllbarkeitsproblem der Aussagenlogik, kurz SAT für satisfiability:

Gegeben: Eine Formel F, zB $(a \wedge \neg(b \vee \neg c))$.

Gefragt: Gibt es eine Belegung der Variablen mit **`wahr`** und **`falsch`**, sodaß F **`wahr`** ist?

$\rightarrow$ Als Sprache formuliert: SAT $= \{\text{code}(F) \in \Sigma^* | F$ ist eine erfüllbare Formel der Aussagenlogik$\}$.

Theorem: (Cook 1971) SAT ist **NP**-vollständig.

Beweis: Membership: Eine NTM N kann erfüllbare Formeln F wie folgt erkennen. Zunächst stellt N fest, welche Variablen in F vorkommen, nehmen wir an $x_1, \ldots, x_k$. In der nichtdeterministischen Phase „rät“ N Werte $a_1, \ldots, a_k \in \{\texttt{wahr}, \texttt{falsch}\}$ für die Variablen ▷

und setzt sie in F ein (das heißt, zu diesem Zeitpunkt existieren 2^k nichtdeterministische unabhängige Rechnungen). Dann rechnet N in deterministischer Art F aus und geht in einen akzeptierenden Endzustand über genau dann, wenn $F \equiv$ **wahr** . All dies geschieht in (nichtdeterministischer) polynomieller Zeit, also ist SAT $\in$ **NP**. $\surd$

Hardness: Sei L ein beliebiges **NP**-Problem. Dann muß es $c \in \mathbb{N}$ und eine NTM $N = (K, \Sigma, \Delta, z_0)$ geben, die L in $p(n) = O(n^c)$ Schritten akzeptiert. Sei $x = x_1 x_2 \ldots x_n \in \Sigma^*$ eine Eingabe für N. Wir definieren weiters $x_0 \stackrel{\text{def}}{=} \triangleright$ und $\forall j \in \{n+1, \ldots, p(n)\}\ x_j \stackrel{\text{def}}{=} \sqcup \in \Sigma$. Wir geben nun eine SAT Formel F an, so daß gilt:

$$x \in L \quad \Leftrightarrow \quad F \text{ ist erfüllbar}$$

Sei $K = \{z_0, z_1, \ldots, z_k\}$ und $\Sigma = \{a_1, \ldots, a_l\}$. $\triangleright$

Die Formel F enthält folgende Boolschen Variablen:

Variable	Indizes	intendierte Bedeutung
$\text{zust}_{t,z}$	$t \in \{0, \ldots, p(n)\}$ $z \in K$	$\text{zust}_{t,z} \equiv$ **wahr** $\Leftrightarrow$ nach t Schritten befindet sich N im Zustand z
$\text{pos}_{t,i}$	$t, i \in \{0, \ldots, p(n)\}$	$\text{pos}_{t,i} \equiv$ **wahr** $\Leftrightarrow$ nach t Schritten befindet sich N's Cursor auf Position i
$\text{band}_{t,i,a}$	$t, i \in \{0, \ldots, p(n)\}$ $a \in \Sigma$	$\text{band}_{t,i,a} \equiv$ **wahr** $\Leftrightarrow$ nach t Schritten befindet sich auf Bandposition i das Zeichen a

▷

F besteht aus mehreren Bestandteilen:

$$F \stackrel{\text{def}}{=} R \wedge A \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$$

Dabei sind R gewisse Randbedingungen, A die Anfangsbedingungen, $\ddot{U}_1$ und $\ddot{U}_2$ beschreiben gewisse Übergangsbedingungen und E die Endbedingung.

Weiters kommt mehrfach eine Hilfsformel G vor:

$$G(x_1, \ldots, x_m) \equiv \textbf{wahr} \Leftrightarrow \text{für genau ein } i \text{ ist } x_i \equiv \textbf{wahr}$$

Wir können G wie folgt konstruieren:

$$G(x_1, \ldots, x_m) \stackrel{\text{def}}{=} \left(\bigvee_{i=1}^{m} x_i \right) \wedge \left(\bigwedge_{j=1}^{m-1} \bigwedge_{i=j+1}^{m} \neg(x_j \wedge x_i) \right)$$

Die Formel G hat offensichtlich die Größe $O(m^2)$, was später noch wichtig sein wird.

▷

$$R \stackrel{\text{def}}{=} \bigwedge_{t=0}^{p(n)} \Big(G(\text{zust}_{t,z_0}, \ldots, \text{zust}_{t,z_k}) \wedge G(\text{pos}_{t,0}, \ldots, \text{pos}_{t,p(n)})$$

$$\wedge \bigwedge_{i=0}^{p(n)} G(\text{band}_{t,i,a_1}, \ldots, \text{band}_{t,i,a_l}) \Big)$$

$\longrightarrow$ R (Randbedigungen): Es gibt zu jedem Zeitpunkt genau einen Zustand und der Cursor befindet sich an genau einer Position, und zu jedem Zeitpunkt und an jeder Bandposition steht genau ein Symbol. Es gilt

$$|R| = O(p(n)^3)$$

$\longrightarrow$ R ist polynomiell in der Länge des Inputs x beschränkt.

⚠ Was wir hier abschätzen ist nur die Anzahl der Variablenpositionen. Diese hängt aber sicher polynomiell mit der eigentlichen Codierungslänge zusammen.

▷

$$A \stackrel{\text{def}}{=} \text{zust}_{0,z_0} \wedge \text{pos}_{0,0} \wedge \bigwedge_{j=0}^{p(n)} \text{band}_{0,j,x_j}$$

$\rightarrow$ A (Anfangsbedingungen): Der Zustand der Variablen zum Zeitpunkt $t = 0$. Es gilt offensichtlich

$$|A| = O(p(n))$$

$\rightarrow$ A ist ebenfalls polynomiell beschränkt. $\triangleright$

$$\ddot{U}_1 \stackrel{\text{def}}{=} \bigwedge_{t,i\in\{0,\dots,p(n)\},\ z\in K,\ a\in\Sigma} \Bigg(\big(\mathsf{zust}_{t,z} \wedge \mathsf{pos}_{t,i} \wedge \mathsf{band}_{t,i,a}\big) \rightarrow \bigvee_{(z',a',r)\,\in\,\Delta(z,a)} \big(\mathsf{zust}_{t+1,z'} \wedge \mathsf{pos}_{t+1,i+D(r)} \wedge \mathsf{band}_{t+1,i,a'}\big) \Bigg)$$

wobei $D(-) \stackrel{\text{def}}{=} 0$, $D(\rightarrow) \stackrel{\text{def}}{=} 1$, und $D(\leftarrow) \stackrel{\text{def}}{=} -1$
sowie $a \rightarrow b \equiv \neg a \vee b$.

$\rightarrow$ $\ddot{U}_1$ (Übergangsbedingung 1): Beschreibt den Übergang vom Zeitpunkt t nach $t+1$ an denjenigen Bandpositionen, wo sich der Cursor befindet. Es gilt

$$|\ddot{U}_1| = O(p(n)^2)$$

$\rightarrow$ $\ddot{U}_1$ ist ebenfalls polynomiell beschränkt. $\triangleright$

$$\ddot{U}_2 \stackrel{\text{def}}{=} \bigwedge_{t,i\in\{0,\ldots,p(n)\},\ a\in\Sigma} \Big((\neg \mathsf{pos}_{t,i} \wedge \mathsf{band}_{t,i,a}) \rightarrow \mathsf{band}_{t+1,i,a} \Big)$$

$\rightarrow$ $\ddot{U}_2$ (Übergangsbedingung 2): Besagt, daß auf Bandfeldern, auf denen nicht der Cursor steht, sich der Bandinhalt nicht ändern darf. Es gilt wieder

$$|\ddot{U}_2| = O(p(n)^2)$$

$\rightarrow$ $\ddot{U}_2$ ist ebenfalls polynomiell beschränkt. $\triangleright$

$$E \stackrel{\text{def}}{=} \bigvee_{z \in \{h, \text{„ja“}, \text{„nein“}\}} \text{zust}_{p(n),z}$$

$\rightarrow$ E (Endbedingung): Prüft, ob ein Endzustand erreicht ist. Wir nehmen dabei an, daß ein einmal erreichter Endzustand nie mehr verlassen wird; dadurch brauchen wir den Endzustand nur zum Zeitpunkt $t = p(n)$ überprüfen. Es gilt

$$|E| = O(1)$$

$\rightarrow$ E ist ebenfalls polynomiell beschränkt.

$\rightarrow$ Da alle Teilformeln polynomiell beschränkt sind, ist also auch F insgesamt polynomiell beschränkt.

▷

$\Rightarrow$) Angenommen, $x \in L$. Dann gibt es eine Sequenz von Konfigurationen der Länge $\leq P(n)$, die in den Endzustand führt. Wenn alle Variablen gemäß der angegebenen Intention und bezogen auf diese akzeptierende Rechnung mit Wahrheitswerten belegt werden, so erhalten alle Teilformeln von F den Wert **`wahr`**, also auch F, und daher ist F erfüllbar. Damit ist die eine Richtung des Beweises gezeigt.

$\triangleright$

$\Leftarrow$) Angenommen, F sei durch eine gewisse Variablenbelegung erfüllbar. Da die Belegung auch R erfüllen muß, hat diese Belegung die Eigenschaft, daß für jedes t die Variablenwerte von $\text{zust}_{t,z}$, $\text{pos}_{t,i}$, und $\text{band}_{t,i,a}$ sinnvoll als Konfiguration von N interpretiert werden können.

Da die Belegung auch A erfüllt, entspricht die für $t = 0$ aus den Variablenwerten abzulesende Konfiguration genau der Startkonfiguration von N bei Eingabe x.

Da die Belegung auch $\ddot{U}_1$ und $\ddot{U}_2$ erfüllen muß, ist zwischen t und $t+1$ immer die Nachfolgebedingung erfüllt. $\rightarrow$ Es wird durch die Variablenbelegung eine mögliche nichtdeterministische Rechnung bestimmt.

Da die Belegung auch E erfüllt, kommt in dieser Rechnung eine Endkonfiguration vor. $\rightarrow x \in L$. $\surd$

Q.E.D.

Weitere NP-vollständige Probleme

$\rightarrow$ Ab nun sind die Beweise einfacher: $L \in$ **NP** zusammen mit SAT $\leq_p L$ (oder jede andere **NP**-vollständige Sprache) reichen, um die **NP**-Vollständigkeit von L zu beweisen.

- Man zeigt dadurch, daß man ein beliebiges Problem in **NP** lösen könnte, indem man es (mit nur geringem Mehraufwand) auf die Frage $x \in L$? reduziert.
- L muß also mindestens genauso schwer wie die schwersten Probleme in **NP** sein.
- Denn wäre L leicht (= in polynomieller Zeit zu beantworten), so könnte man ja jedes Problem in **NP** (also auch die schwersten darunter) schnell (polynomiell) in eine Frage $x \in L$? umwandeln, darauf schnell die Antwort finden, und das ursprüngliche angeblich schwere Problem so leicht lösen $\rightarrow$ Widerspruch.

Definition: 3CNF (auch 3SAT genannt)

Gegeben: Eine Boolsche Formel F in konjunktiver Normalform mit höchstens 3 Literalen pro Klausel.

Beispiel: $(a) \wedge (b) \wedge (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee c)$.

Gefragt: Ist F erfüllbar?

Theorem: 3CNF ist **NP**-vollständig.

Beweis:

Membership: Guess und check Argument $\surd$

Hardness: Wir zeigen SAT $\leq_p$ 3CNF.

Das bedeutet: Wir müssen ein polynomielles Verfahren angeben, das eine beliebige Boolsche Formel F in eine 3CNF Formel F' umformt, sodaß gilt

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar.}$$

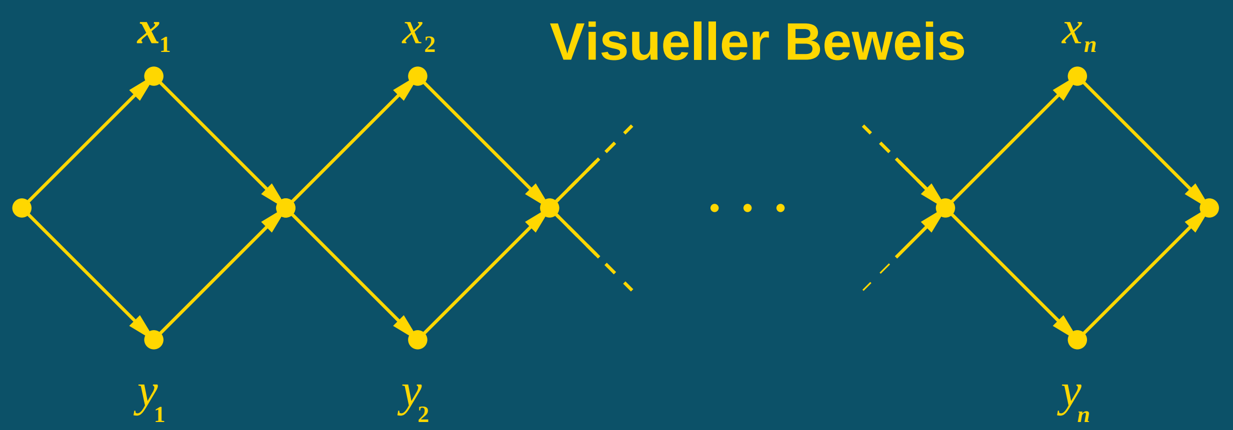
▷

Erstes Problem: Äquivalente Umgewandlung CNF $\leftrightarrow$ DNF hat i.a. exponentiellen Aufwand, weiters werden nicht notwendigerweise Klauseln mit nur 3 Literalen erzeugt.

Beispiel: $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n) \equiv$

$$\left.\begin{array}{rl} & (x_1 \wedge x_2 \wedge \cdots \wedge x_{n-1} \wedge x_n) \\ \vee & (y_1 \wedge x_2 \wedge \cdots \wedge x_{n-1} \wedge x_n) \\ \vee & (x_1 \wedge y_2 \wedge \cdots \wedge x_{n-1} \wedge x_n) \\ & \vdots \\ \vee & (x_1 \wedge y_2 \wedge \cdots \wedge y_{n-1} \wedge y_n) \\ \vee & (y_1 \wedge y_2 \wedge \cdots \wedge y_{n-1} \wedge y_n) \end{array}\right\} 2^n$$

Visueller Beweis



Lösung: Wir zeigen nur Erfüllbarkeitsäquivalenz und führen dazu neue Variable ein. Die Umformung geschieht in mehreren Schritten. Beispiel: $F = \neg(\neg(x_1 \vee \neg x_3) \vee x_2)$.

1. Anwendung der Regeln von DeMorgan, um alle Negationszeichen zu den Variablen zu verschieben.
   $\rightarrow ((x_1 \vee \neg x_3) \wedge \neg x_2)$. Aufwand: $O(n)$.
2. Wir ordnen jedem $\wedge$ und $\vee$ eine neue Variable $\in \{y_0, y_1, \ldots\}$ zu.
   $\rightarrow ((x_1 \overset{y_1}{\vee} \neg x_3) \overset{y_0}{\wedge} \neg x_2)$. Aufwand: $O(n^3)$.
3. Wir klammern so um, daß nur noch binäre Ausdrücke mit $\wedge$ und $\vee$ vorhanden sind und ordnen jedem solchen Ausdruck $(a \overset{y_j}{\circ} b)$ mit $\circ \in \{\wedge, \vee\}$ und $a, b \in \{x_1, \ldots, x_n, y_0, y_1, \ldots\}$, eine Teilformel der Form

$$(y_j \leftrightarrow (a \circ b))$$

▷

zu. Alle diese Formeln sowie y_0 werden mit $\wedge$ zu einer neuen Formel F_1 verknüpft.
$\rightarrow F_1 = [y_0] \wedge [y_0 \leftrightarrow (y_1 \wedge \neg x_2)] \wedge [y_1 \leftrightarrow (x_1 \vee \neg x_3)]$.
Aufwand: $O(n)$.
$\rightarrow F$ und F_1 sind erfüllbarkeitsäquivalent.

4. Jeder Ausdruck $[\ldots]$ in F_1 wird nun in CNF umgeformt:
$[a \leftrightarrow (b \vee c)] \mapsto (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg c)$
$[a \leftrightarrow (b \wedge c)] \mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$
Aufwand: $O(n)$.

Wir erhalten also die gewünschte 3CNF Formel

$$\begin{aligned} \rightarrow \quad F' = {} & y_0 \wedge (\neg y_0 \vee y_1) \wedge (\neg y_0 \vee \neg x_2) \wedge (y_0 \vee \neg y_1 \vee x_2) \\ & \wedge (y_1 \vee \neg x_1) \wedge (\neg y_1 \vee x_1 \vee \neg x_3) \wedge (y_1 \vee x_3) \end{aligned}$$

$\rightarrow$ Gesamtaufwand ist polynomiell, damit haben wir SAT $\leq_p$ 3CNF gezeigt. $\surd$