



Komplexitätstheorie

Prof. Dr. Björn Grohmann



Cook-Levin Theorem



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

THEOREM

SAT is NP-complete.

Cook-Levin Theorem



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

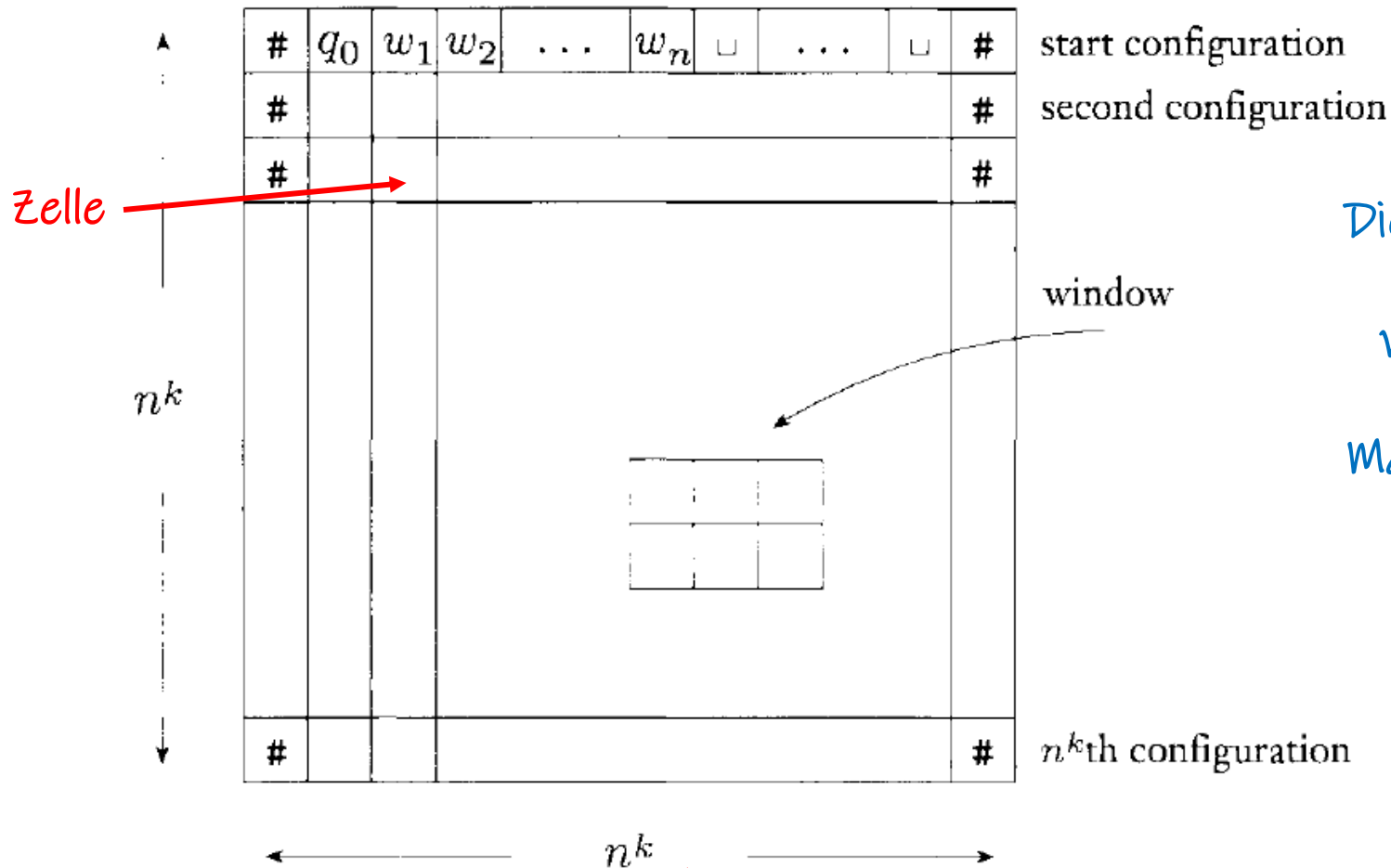
- 1) SAT ist in NP
- 2) Zeige für eine Sprache A in NP, dass sie polynomial reduzierbar auf SAT ist.

Cook-Levin Theorem

akzeptierendes
„Tableaux“



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Die Idee ist hier für jedes Wort w eine aussagenlogische Formel zu finden, welche **polynomiale Größe** hat und die Berechnung der (nichtdet.-) Turing-Maschine simuliert, d.h. die genau dann erfüllbar ist, falls die Maschine w akzeptiert.

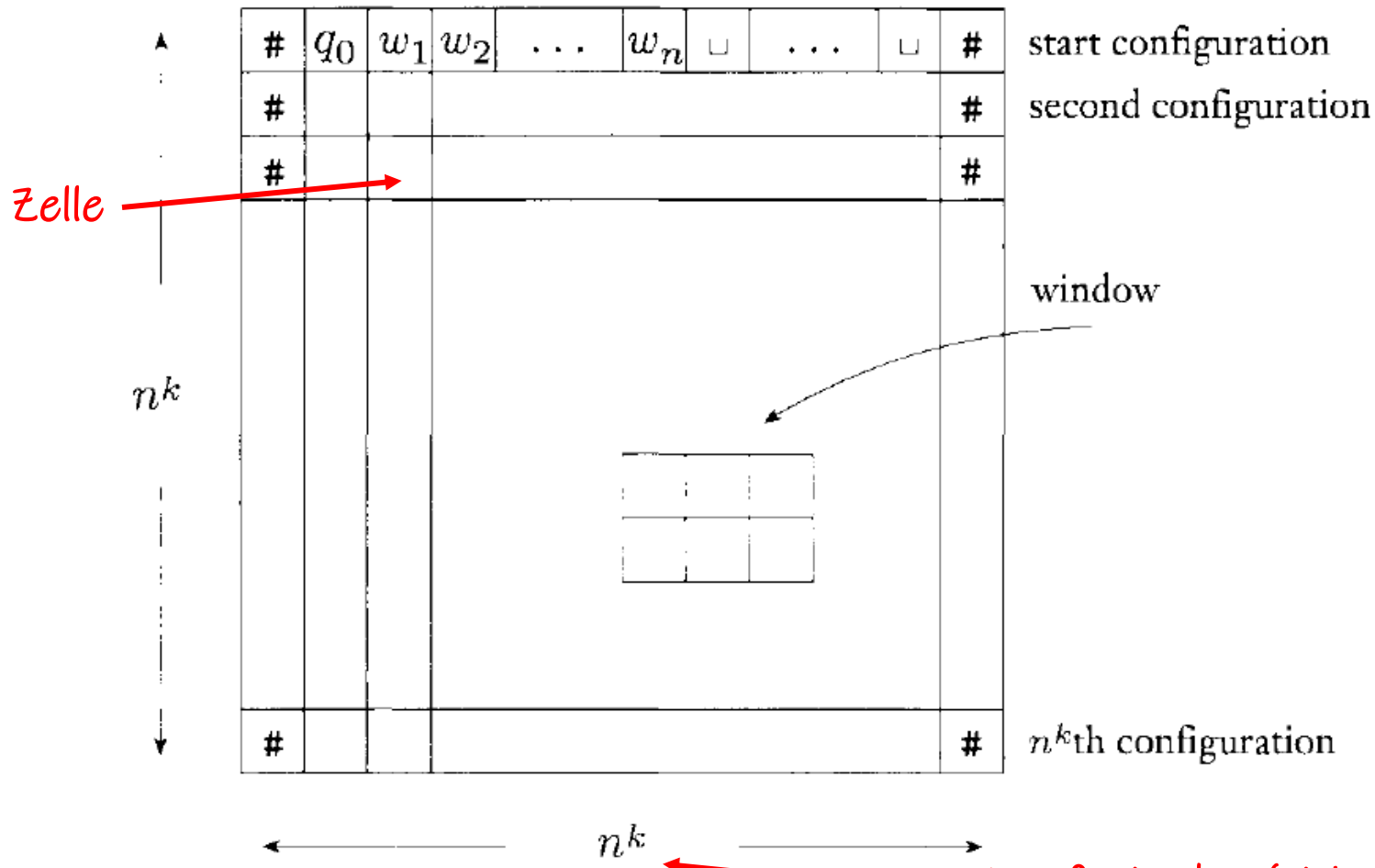
Laufzeit der (nichtdet.-) Turing-Maschine, welche A akzeptiert

Cook-Levin Theorem

akzeptierendes
„Tableaux“



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Unsere Formel sieht dann so aus:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

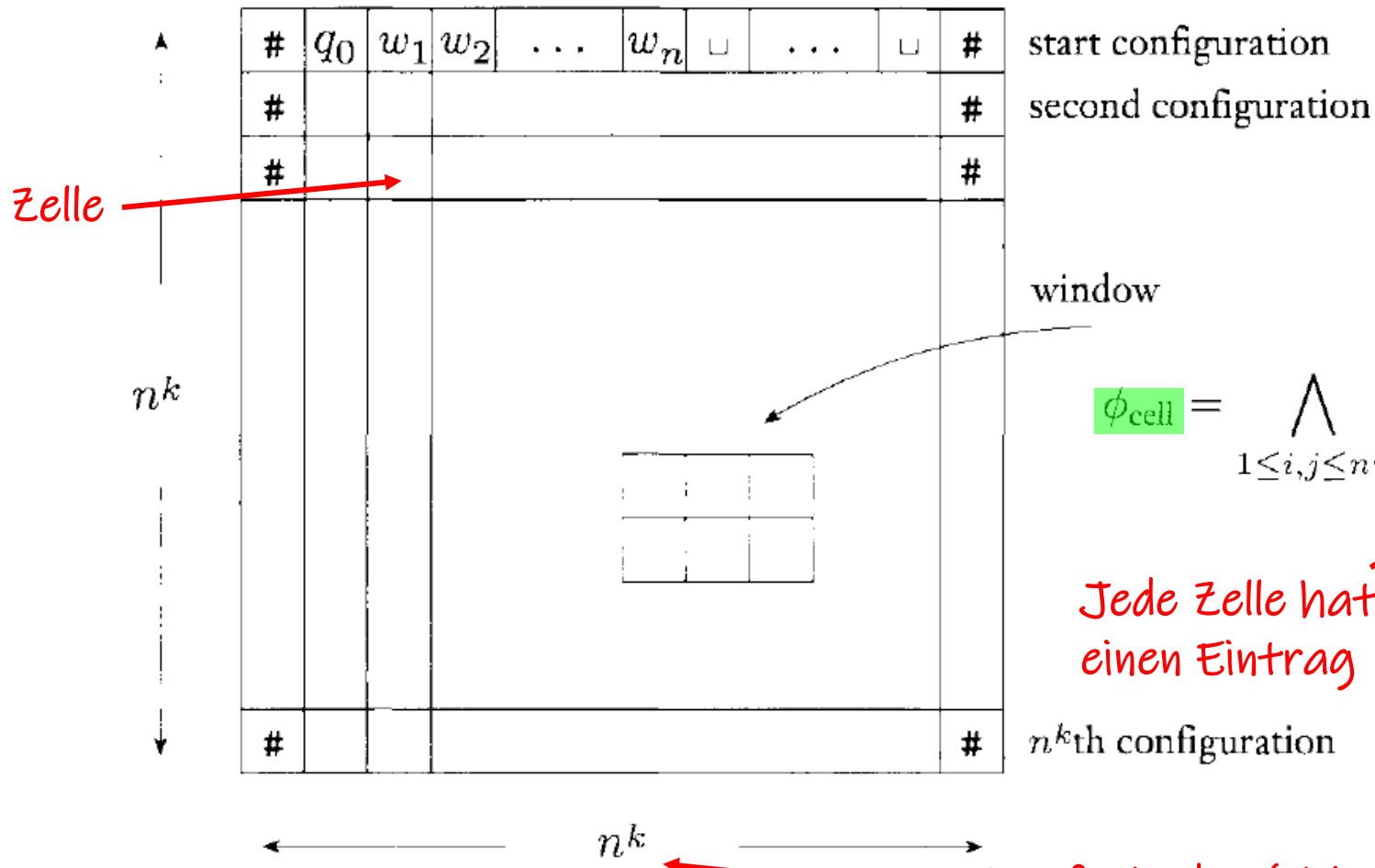
Laufzeit der (nichtdet.-) Turing-Maschine,
welche A akzeptiert

Cook-Levin Theorem

akzeptierendes
„Tableaux“



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Unsere Formel sieht dann so aus:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}},$$

wobei

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s, t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

Jede Zelle hat mind.
einen Eintrag

Jede Zelle hat
höchstens einen Eintrag

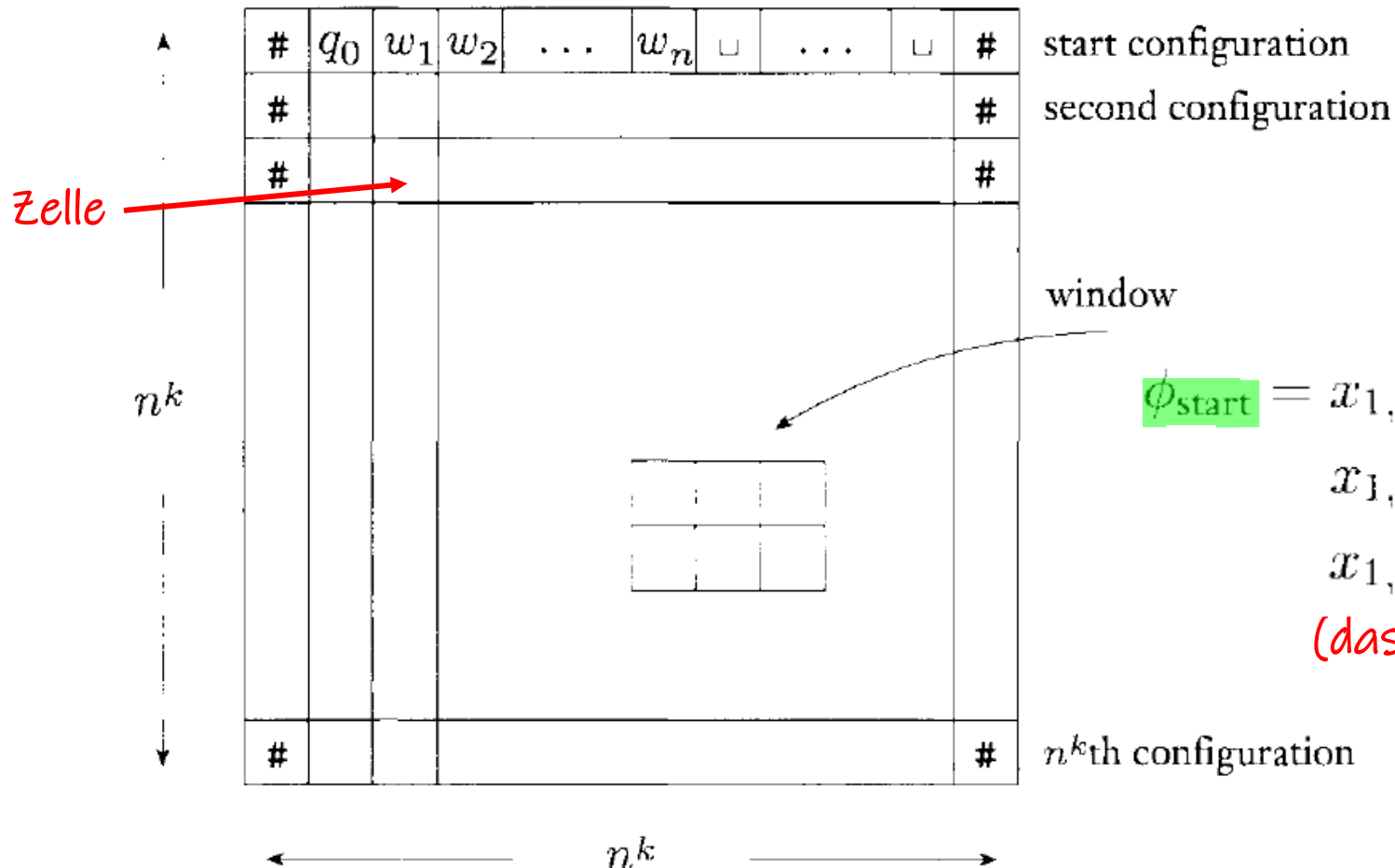
Laufzeit der (nichtdet.-) Turing-Maschine,
welche A akzeptiert

Cook-Levin Theorem

akzeptierendes
„Tableaux“



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Unsere Formel sieht dann so aus:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}},$$

wobei

$$\begin{aligned} \phi_{\text{start}} = & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\square} \wedge \dots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#} \end{aligned}$$

(das ist gerade die Anfangskonfiguration)

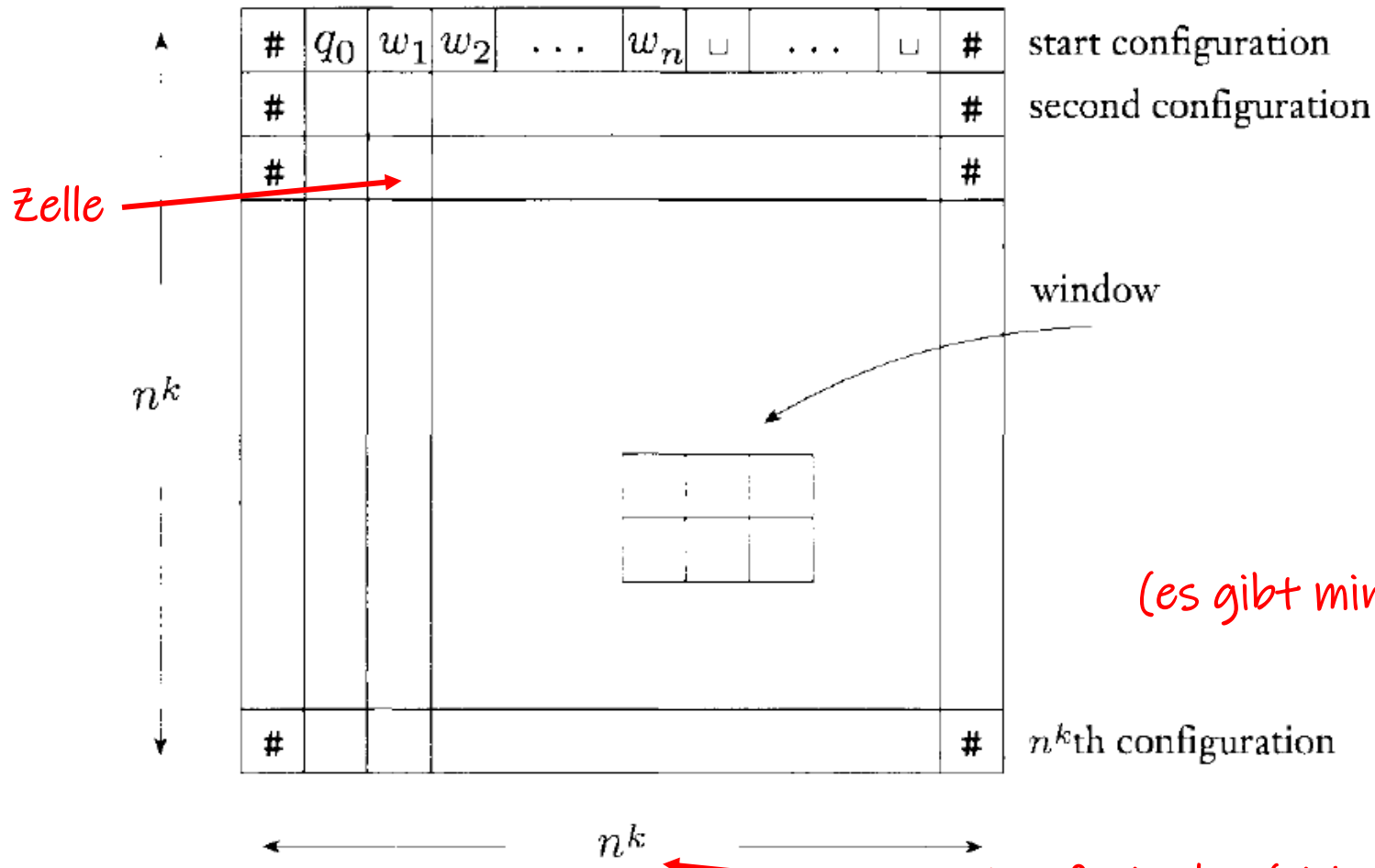
Laufzeit der (nichtdet.-) Turing-Maschine,
welche A akzeptiert

Cook-Levin Theorem

akzeptierendes
„Tableaux“



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Unsere Formel sieht dann so aus:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}},$$

wobei

$$\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i, j, q_{\text{accept}}}$$

(es gibt mind. eine akzeptierende Konfiguration)

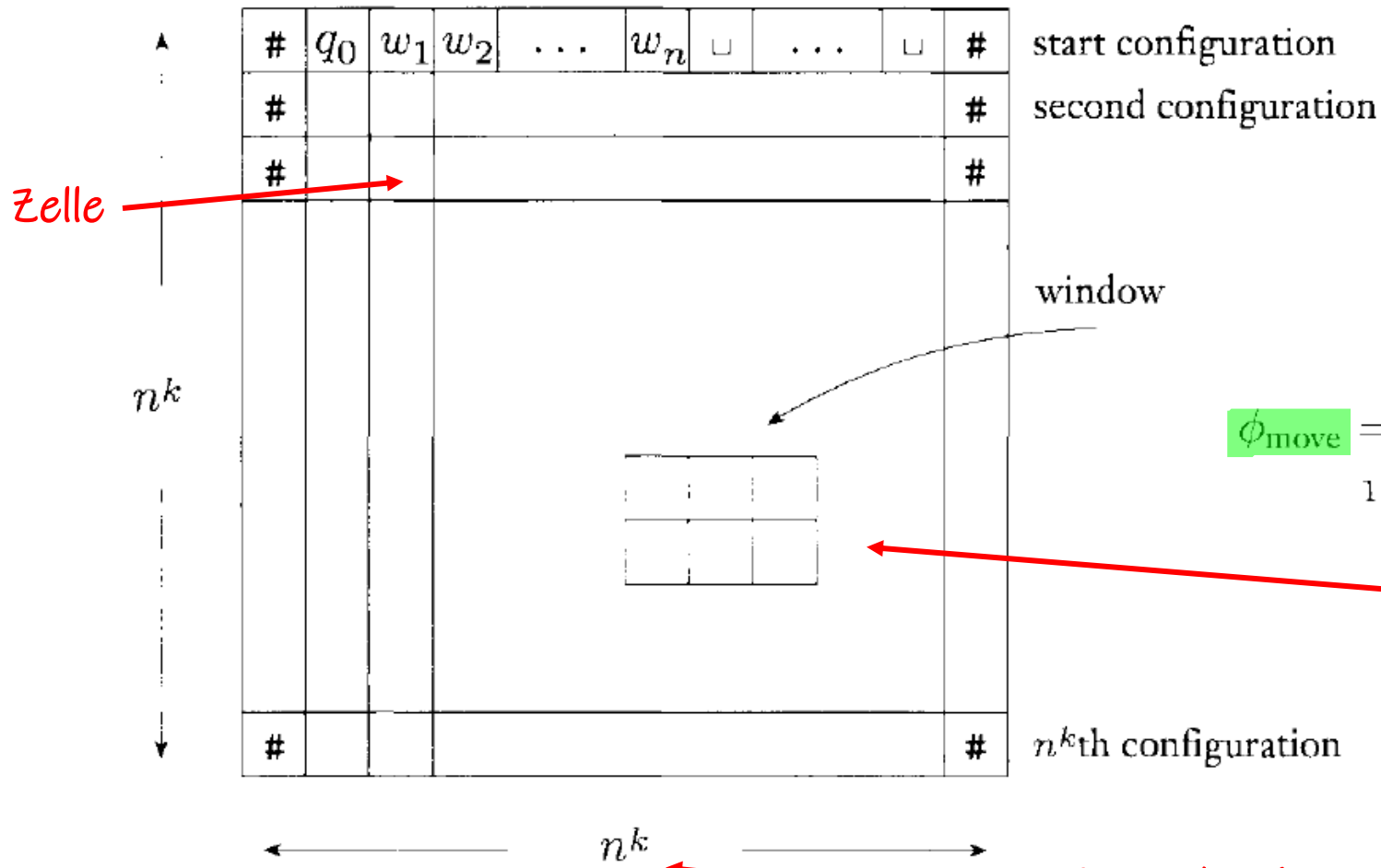
Laufzeit der (nichtdet.-) Turing-Maschine,
welche A akzeptiert

Cook-Levin Theorem

akzeptierendes
„Tableaux“



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Unsere Formel sieht dann so aus:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}},$$

wobei

$$\phi_{\text{move}} = \bigwedge_{1 \leq i \leq n^k, 1 \leq j < n^k} (\text{the } (i, j) \text{ window is legal})$$

Ein Fenster hat hier die
Größe 2×3

Laufzeit der (nichtdet.-) Turing-Maschine,
welche A akzeptiert

Cook-Levin Theorem



Ein Fenster ist **legal**, falls es nicht der **Spezifikation der Übergangsfunktion** der entsprechenden Maschine widerspricht:

Beispiel: $\delta(q_1, a) = \{(q_1, b, R)\}$
 $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

Diese Fenster sind alle
„legal“

a	b	a
a	a	a

a	q ₁	b
q ₂	a	c

a	q ₁	b
a	a	q ₂

a	a	q ₁
a	a	b

Diese nicht:

a	q ₁	b
q ₁	a	a

#	b	a
#	b	a

a	b	a
a	b	q ₂

b	b	b
c	b	b

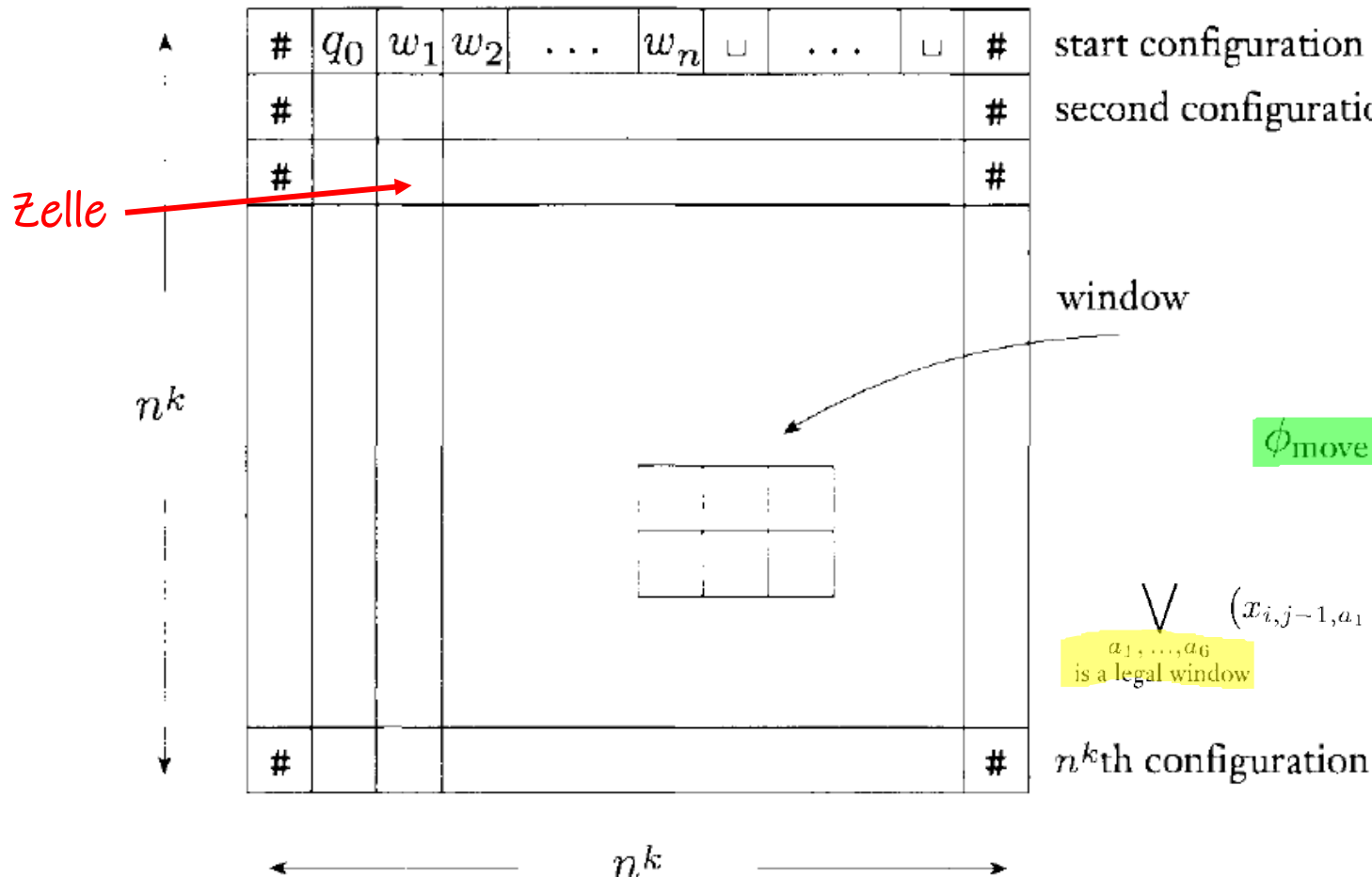
b	q ₁	b
q ₂	b	q ₂

Cook-Levin Theorem

akzeptierendes
„Tableaux“



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Unsere Formel sieht dann so aus:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}},$$

wobei

$$\phi_{\text{move}} = \bigwedge_{1 \leq i \leq n^k, 1 \leq j < n^k} (\text{the } (i, j) \text{ window is legal})$$

$$\bigvee_{\substack{a_1, \dots, a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6})$$

Laufzeit der (nichtdet.-) Turing-Maschine,
welche A akzeptiert



Cook-Levin Theorem

CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s, t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$



Cook-Levin Theorem

CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$$O(n^{2k})$$

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s, t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

Cook-Levin Theorem



CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\begin{aligned} & \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}} \\ O(n^{2k}) & \rightarrow \phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \dots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \end{aligned}$$

Cook-Levin Theorem



CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\begin{array}{l} O(n^{2k}) \xrightarrow{\quad} \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}} \\ O(n^k) \xrightarrow{\quad} \phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ \quad x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,n+2,w_n} \wedge \\ \quad x_{1,n+3,\sqcup} \wedge \dots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \end{array}$$

Cook-Levin Theorem

CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$O(n^{2k})$ $O(n^k)$

$$\phi_{\text{move}} = \bigwedge_{1 \leq i \leq n^k, 1 \leq j < n^k} (\text{the } (i, j) \text{ window is legal})$$

Cook-Levin Theorem



CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\begin{array}{ccccc} & & \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}} & & \\ & \nearrow & & \nwarrow & \\ O(n^{2k}) & & & & O(n^{2k}) \\ & \nearrow & & \nwarrow & \\ & O(n^k) & & & \end{array}$$
$$\phi_{\text{move}} = \bigwedge_{1 \leq i \leq n^k, 1 \leq j < n^k} (\text{the } (i, j) \text{ window is legal})$$

Cook-Levin Theorem



CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\begin{array}{ccccc} & & \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}} & & \\ & \nearrow & & \nwarrow & \\ O(n^{2k}) & & & & O(n^{2k}) \\ & \nearrow & & \nwarrow & \\ & O(n^k) & & & \end{array}$$
$$\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i, j, q_{\text{accept}}}$$

Cook-Levin Theorem



CLAIM

If the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Bleibt noch die Größe der Formel:

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$O(n^{2k})$ $O(n^k)$ $O(n^{2k})$ $O(n^{2k})$

$$\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i, j, q_{\text{accept}}}$$

3SAT



COROLLARY

3SAT is NP-complete.

(es reicht hier einzusehen, dass zu jeder Formel F eine Formel F' in 3KNF existiert, welche **erfüllbarkeitsäquivalent** zu F ist: schreibe F in KNF und ersetze jede Klausel mit $l > 3$ Literalen, $(a_1 \vee a_2 \vee \dots \vee a_l)$, durch die $l-2$ Klauseln $(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \dots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee a_l)$)

Hamiltonian Circuit Problem



Das Hamiltonian Circuit Problem ist NP-vollständig, und zwar schon falls der Graph planar und 3-regulär ist.

Graph, welcher in der Ebene dargestellt werden kann, so dass sich keine zwei Kanten schneiden

jeder Knoten besitzt genau 3 Nachbarn

für einen (ungerichteten) Graph soll entschieden werden, ob ein Rundweg existiert, welcher jeden Knoten genau einmal enthält



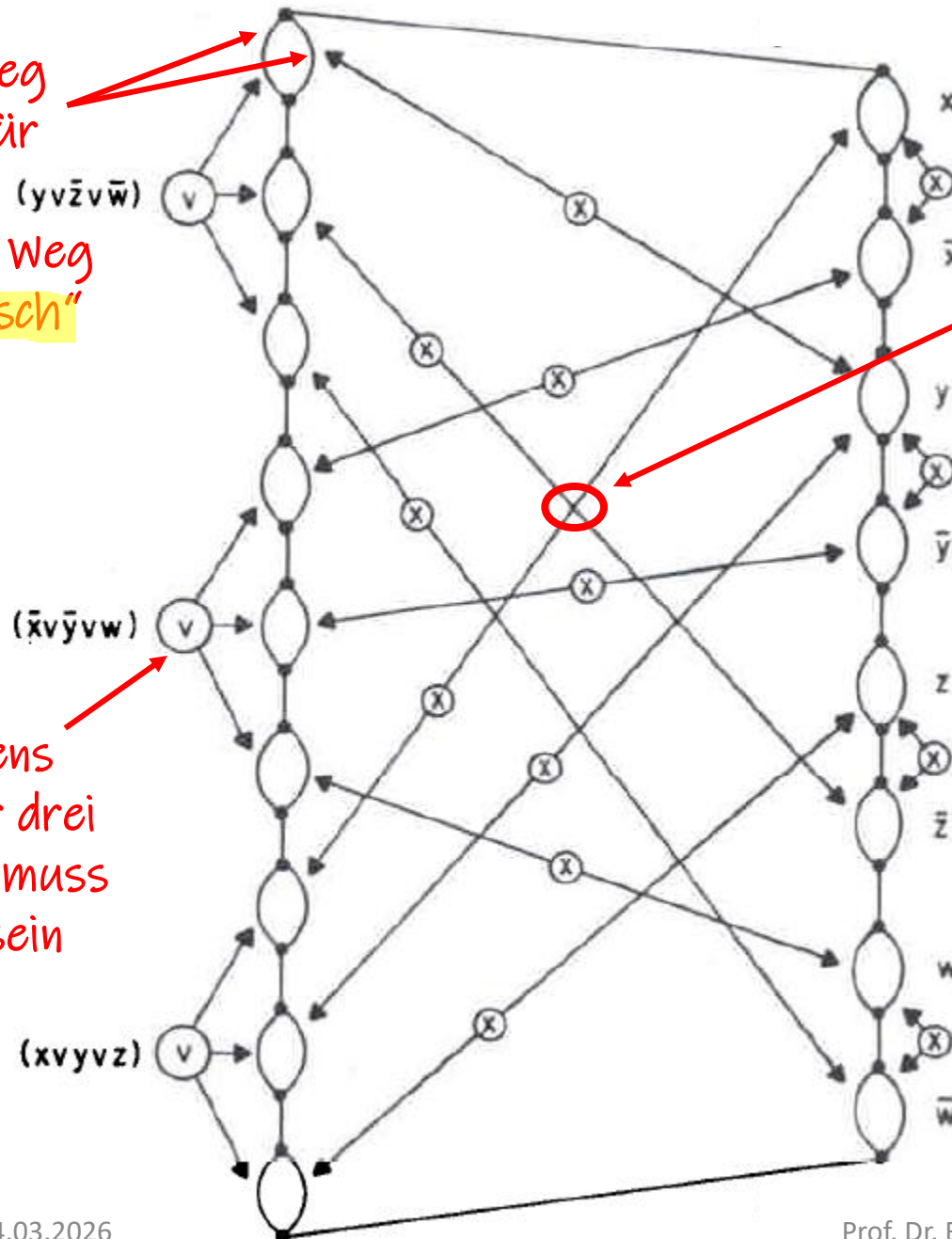
linker Weg
steht für
„wahr“, $(yv\bar{z}v\bar{w})$
rechter Weg
für „falsch“

„Kreuzung“: Kanten des Graphen
dürfen sich nicht kreuzen

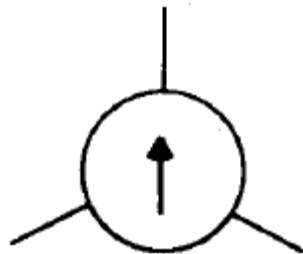
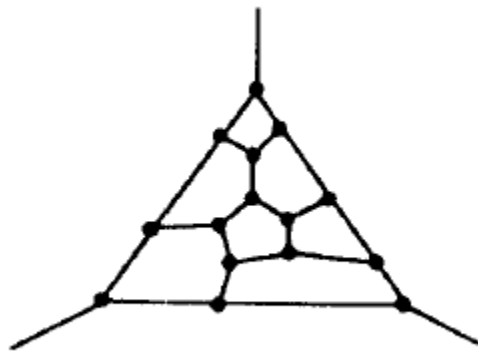
Wir zeigen die Konstruktion an
einem Beispiel: gesucht ist ein
Graph (mit den genannten
Eigenschaften) welcher genau dann
einen (Hamilton-) Rundweg besitzt,
falls die unten stehende Formel
erfüllbar ist.

$$F = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee w) \wedge (y \vee \bar{z} \vee \bar{w})$$

„xor“: genau einer der
beiden Wege gehört
zum Pfad

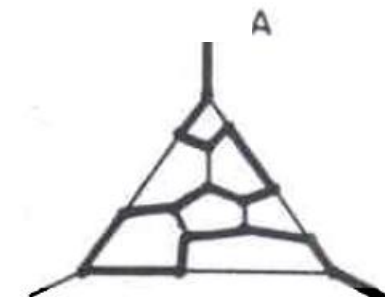
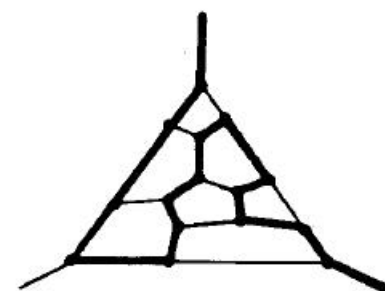
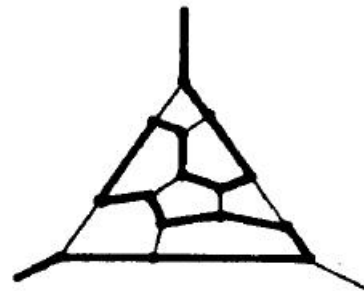
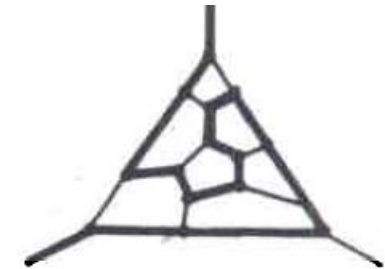
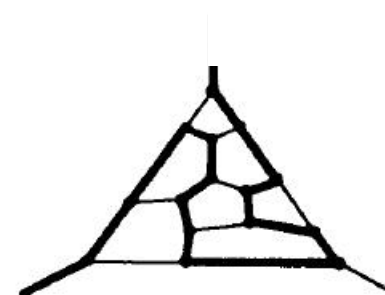
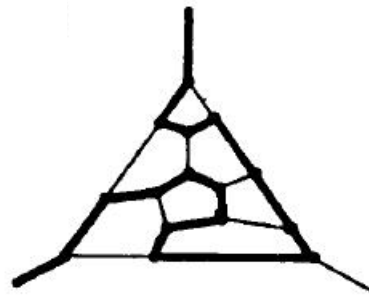


Tutte Fragment

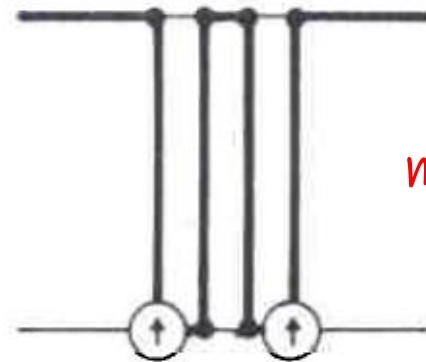
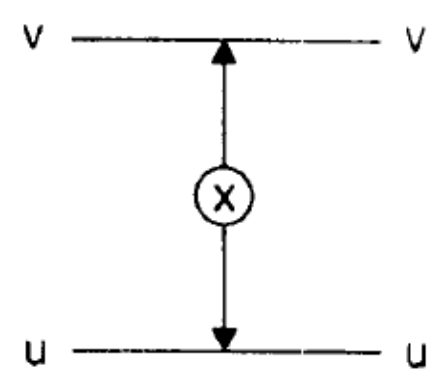
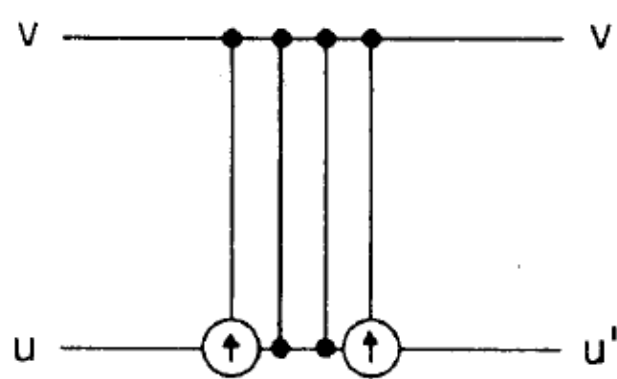


Symbol

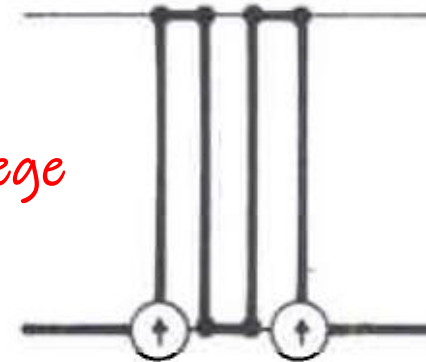
mögliche Wege



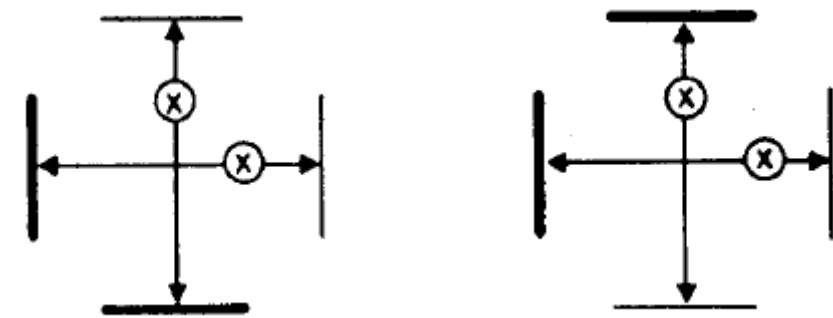
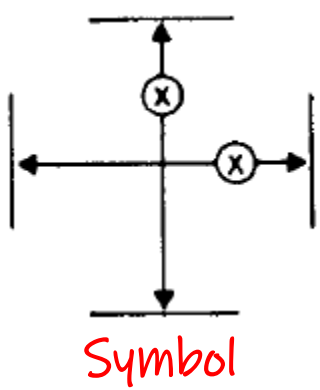
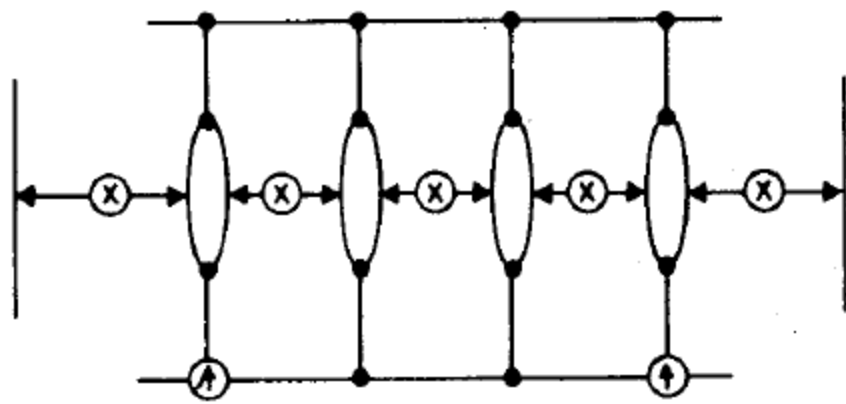
„Exclusive-OR“



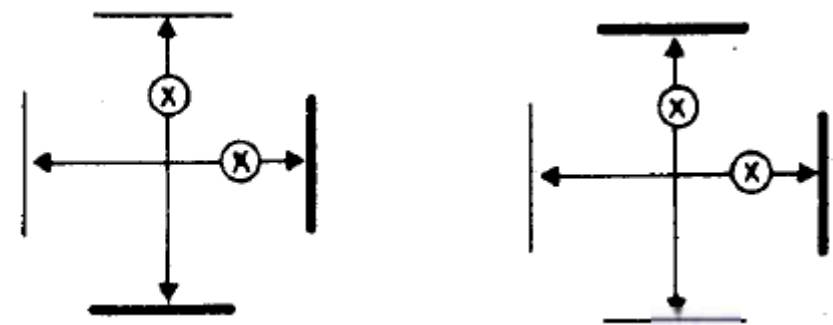
mögliche Wege



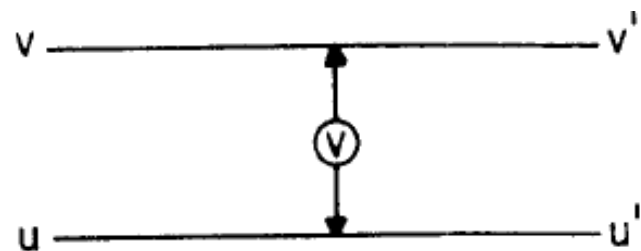
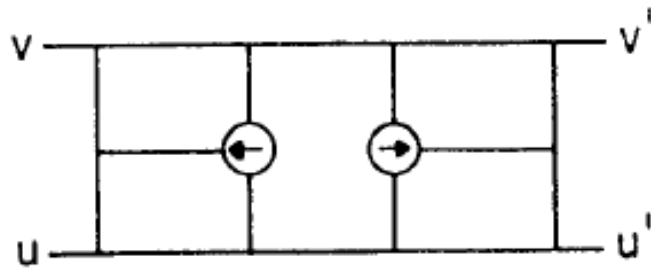
„Kreuzung“



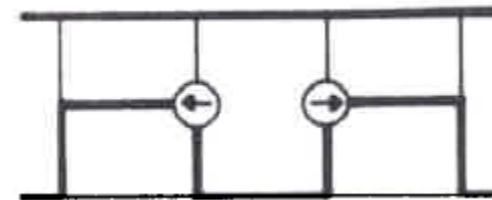
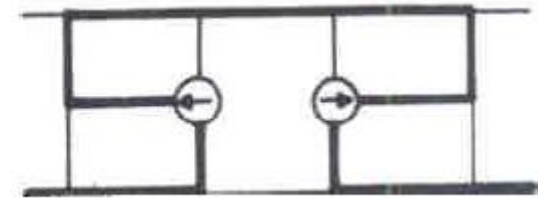
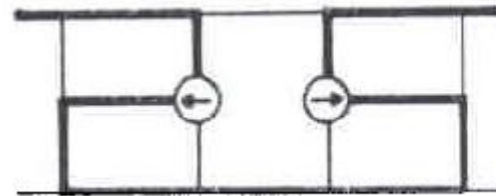
mögliche Wege



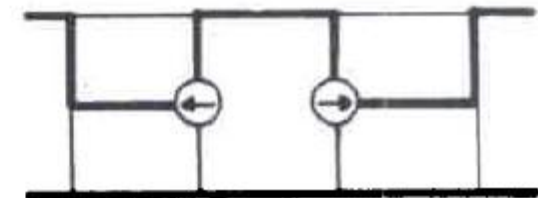
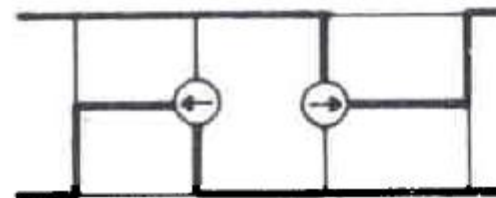
„OR“



Symbol

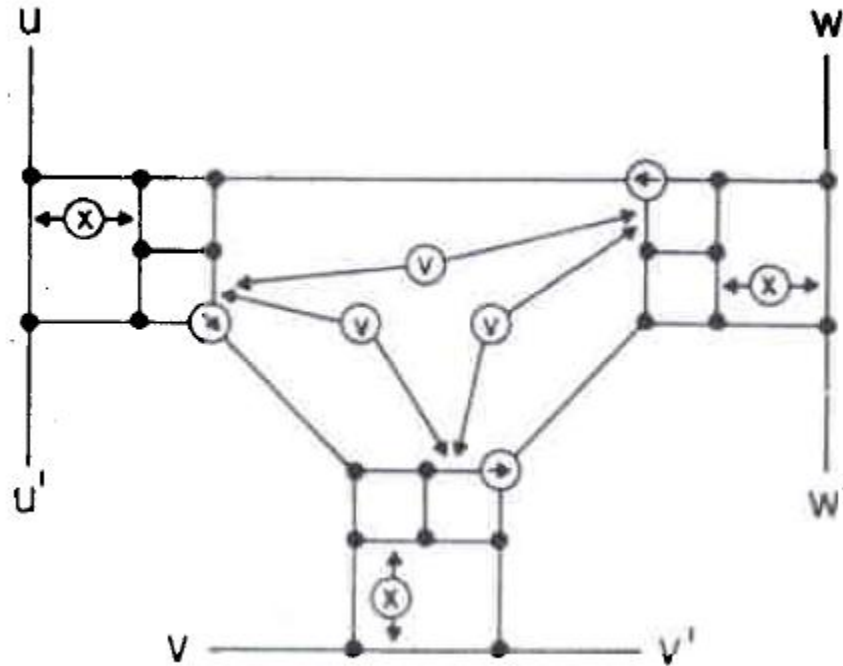


mögliche Wege

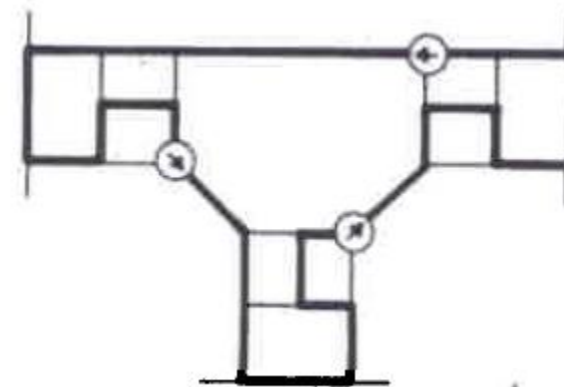
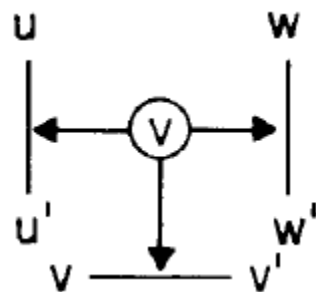




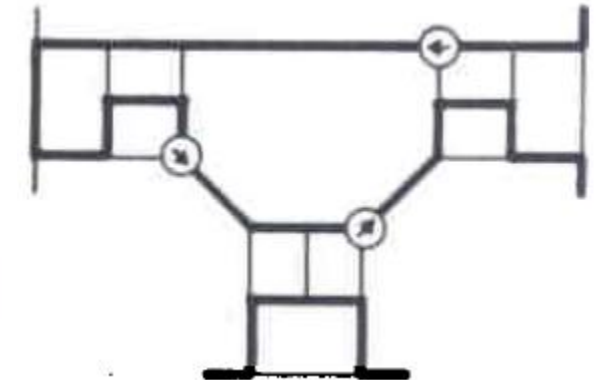
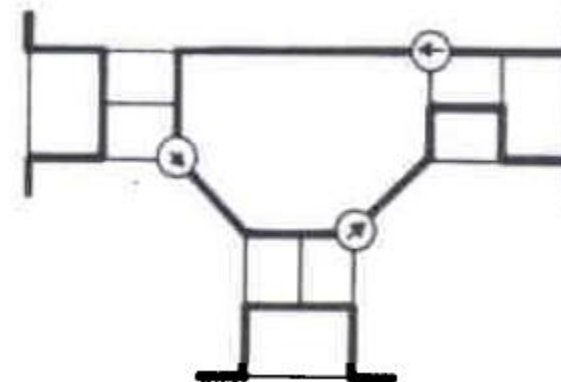
„(trippel-)OR“



Symbol



mögliche Wege
(ohne Symmetrie)



MINESWEEPER



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



MINESWEEPER



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

	2	2	2	2	
	2	0	0	2	
	2	0	0	2	
	2	2	2	2	

MINESWEEPER ist NP-vollständig

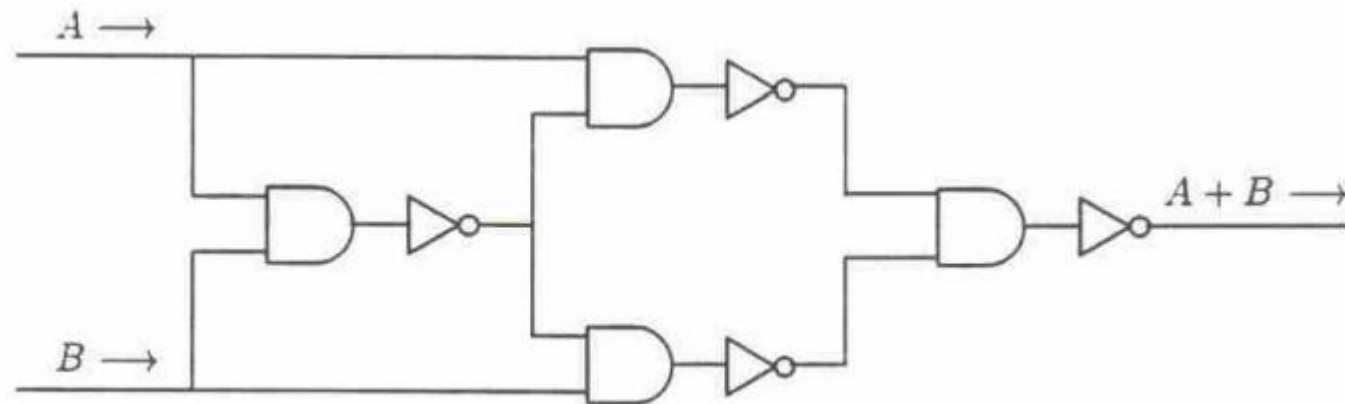


Figure 12. Making an XOR gate with AND and NOT gates.

MINESWEEPER ist NP-vollständig



$X \rightarrow$

...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
...	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
...	x	1	x'	x	1	x'	x	1	x'	x	1	x'	x	1	x'	x	...
...	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...

Figure 6. A wire.

MINESWEEPER ist NP-vollständig



$X \longrightarrow$				1	2	2	1	
...	1	1	1	2	*	*	3	1
...	1	x'	x	1	x'	*	*	2
...	1	1	1	1	2	x	*	2
					1	1	2	1
					1	x'	1	
					1	x	1	X
					1	1	1	\downarrow
					\vdots	\vdots	\vdots	

1	1	1	$X \rightarrow$					
2	*	3	1	1	1	1	1	...
3	*	x'	x	1	x'	x	1	...
2	*	3	1	1	1	1	1	...
1	1	1						

			\vdots	\vdots	\vdots				
			1	x	1		\uparrow	X	
			1	x'	1				
$X \rightarrow$			1	1	1		$X' \rightarrow$		
...	1	1	1	1	x	1	1	1	...
...	x'	x	1	x'	2	x'	1	x	x'
...	1	1	1	1	x	1	1	1	...
			1	1	1				
			1	x'	1				
			1	x	1		X		
			\vdots	\vdots	\vdots		\downarrow		

MINESWEEPER ist NP-vollständig

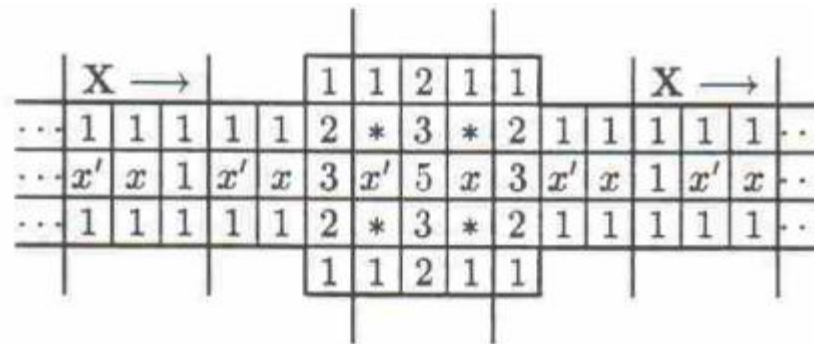


Figure 10. A phase-changer made from two NOT gates.

MINESWEEPER ist NP-vollständig

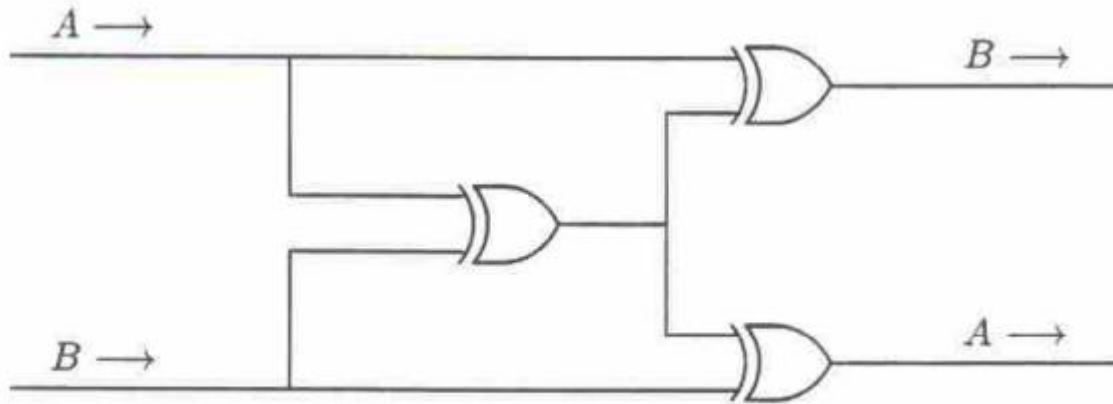


Figure 11. Crossing two wires with three XOR gates.

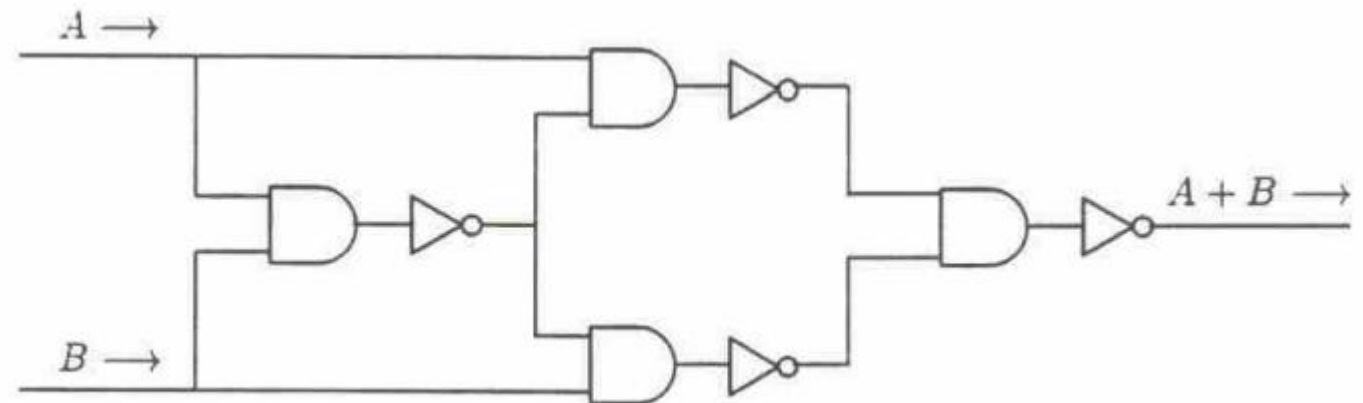


Figure 12. Making an XOR gate with AND and NOT gates.

MINESWEEPER ist NP-vollständig



X →					1 1 1			X' →						
...	1	1	1	1	1	2	*	2	1	1	1	1	1	...
...	x'	x	1	x'	x	3	x'	3	x	x'	1	x	x'	...
...	1	1	1	1	1	2	*	2	1	1	1	1	1	...
					1 1 1									

Figure 9. A NOT gate.

MINESWEEPER ist NP-vollständig

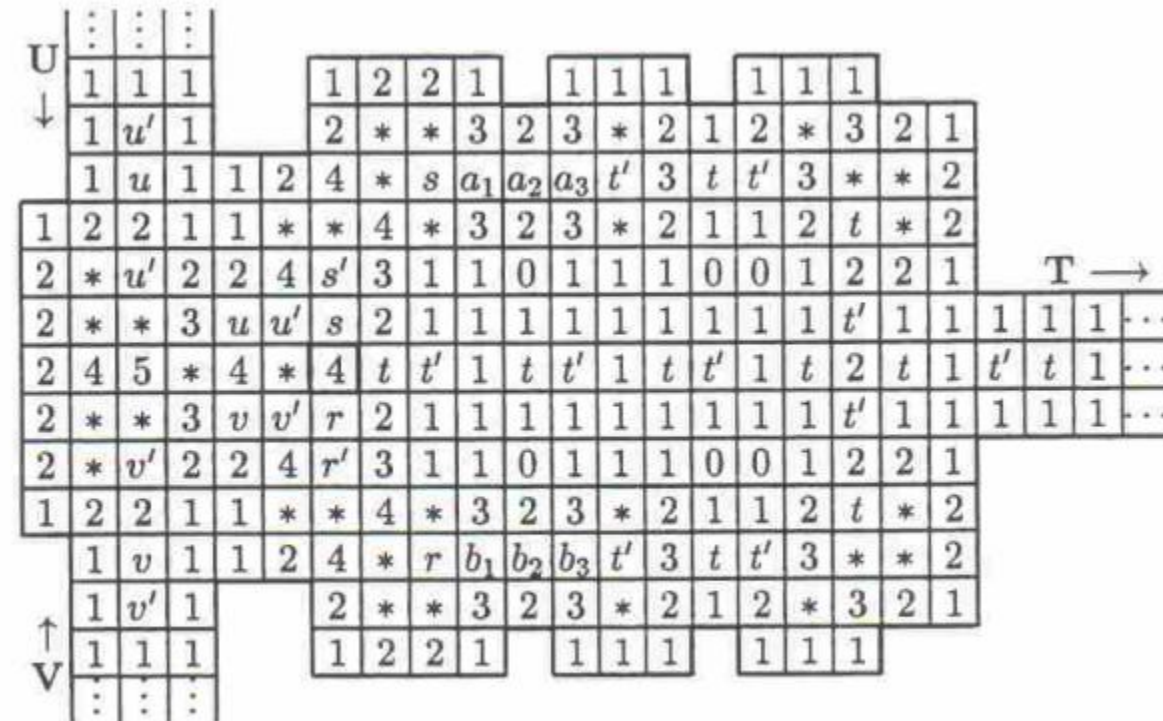


Figure 13. An AND gate.

„Gaming is hard“, Part I



↙ d.h. bestimmte Punkte
müssen erreicht werden



(a) Initial configuration.



(b) Configuration after a traversal from left to right.



(c) On the second traversal attempt, a boulder blocks the way.

Metatheorem 1. Any game exhibiting both *location traversal* (with or without a starting location or an exit location) and *single-use paths* is NP-hard.

Wir bauen einen Level, welcher einen *planaren, 3-regulären Graph* darstellt, wobei die Knoten die Punkte sind, welche besucht werden müssen und *jede Kante ein „single-use-path“* darstellt. Nach Wahl eines Startpunktes (Knoten) „S“ verbinden wir diesen mit einem weiteren (End-)Punkt, welcher nur eine Kante zum Startknoten hat und ansonsten keine weiteren Nachbarn besitzt. Der Level ist nun genau dann *lösbar*, falls ein *Hamilton-Kreis* von „S“ nach „S“ existiert.

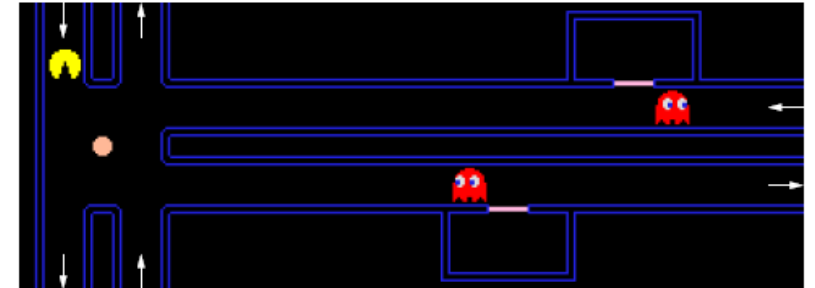
„Gaming is hard“, Part I

Metatheorem 2. *A game is NP-hard if either of the following holds:*

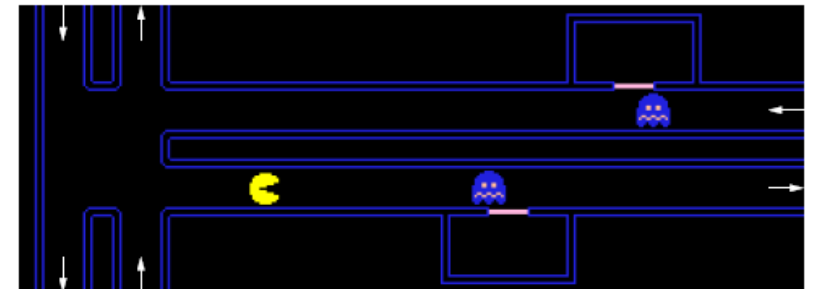
- (a) *The game features collectible tokens, toll roads, and location traversal.*
- (b) *The game features cumulative tokens, toll roads, and location traversal.*
- (c) *The game features collectible cumulative tokens, toll roads, and the avatar has to reach an exit location.*

um diesen Weg zu passieren,
muss ein Token eingesammelt
werden

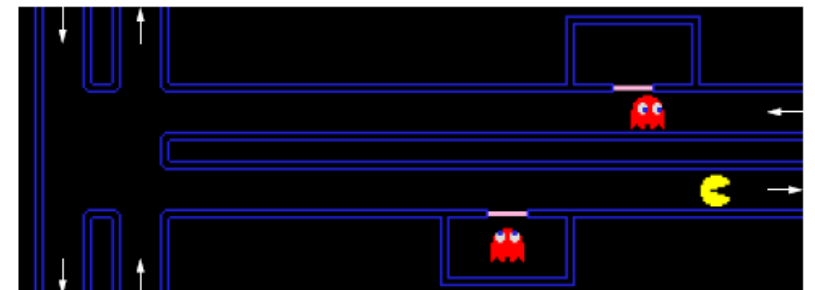
(a) und (b) wie bei MT 1: „tokens“ und „toll roads“ ergeben „single-use-paths“. Bei (b) hat der Spieler zu begin $n+1$ „tokens“. Bei (c) plazieren wir 2 „tokens“ an jeden Punkt und jede Kante wird eine „toll road“, außer der Kante zwischen Start- und Endpunkt, welche eine Sequenz von n „toll roads“ ist.



(a) Initial configuration, approached from the top.



(b) Collecting the power pill to traverse the corridor.



(c) Exiting to the right while the ghosts recover.

„Gaming is hard“, Part I



eine Tür wird von einem Schlüssel
geöffnet, welcher danach als
verbraucht gilt



Metatheorem 3. *A game is NP-hard if it contains **doors** and one-way paths, and either of the following holds:*

- (a) The game features **collectible keys** and location traversal.*
- (b) The game features **cumulative keys** and location traversal.*
- (c) The game features collectible cumulative keys and the avatar has to reach an exit location.*

(alles genau wie bei MT 2: „keys“ = „tokens“, „doors“ = „toll roads“)

„Gaming is hard“, Part I

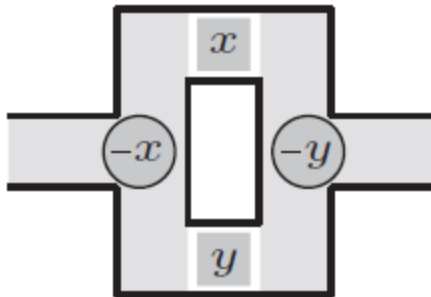


es gibt sowohl Druckplatten, welche
Türen öffnen, als auch schließen



Metatheorem 4. *If a game features **doors** and **pressure plates**, and the avatar has to reach an exit location in order to win, then:*

*Even if no two pressure plates control the same door, the game is **NP-hard**.*



Alles wie bei MT 1: damit alle Punkte besucht werden
müssen, sind vor dem Ausgang n Türen, welche mit
Druckplatten bei jedem Punkt geöffnet werden
müssen

„single-use-path“ mit Druckplatten

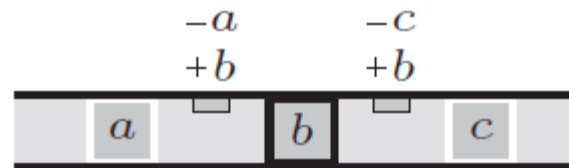
„Gaming is hard“, Part I



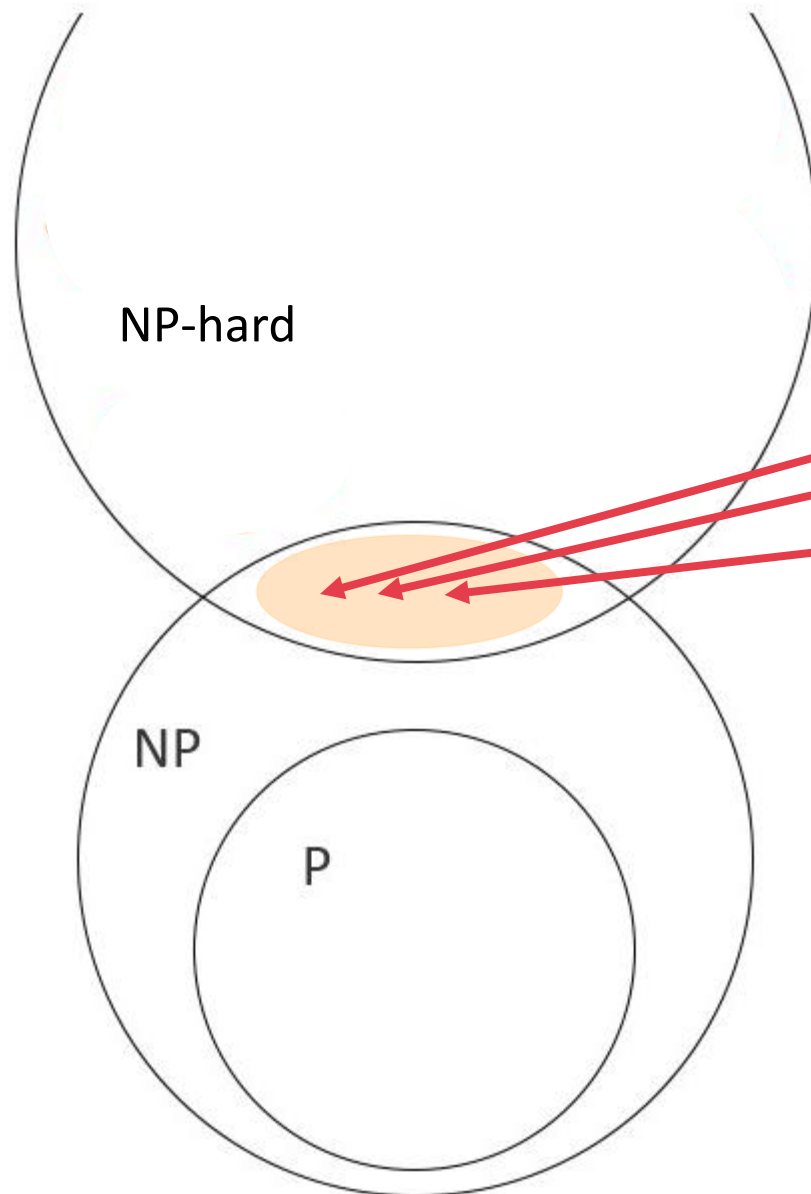
Knöpfe sind wie Druckplatten, außer dass der Spieler entscheiden kann, ob er ihn drücken will.
Ein k -Button beeinflusst k Türen

Metatheorem 5. *If a game features **doors** and **k -buttons**, and the avatar has to reach an exit location in order to win, then:*

*If $k \geq 2$, then the game is **NP-hard**.*



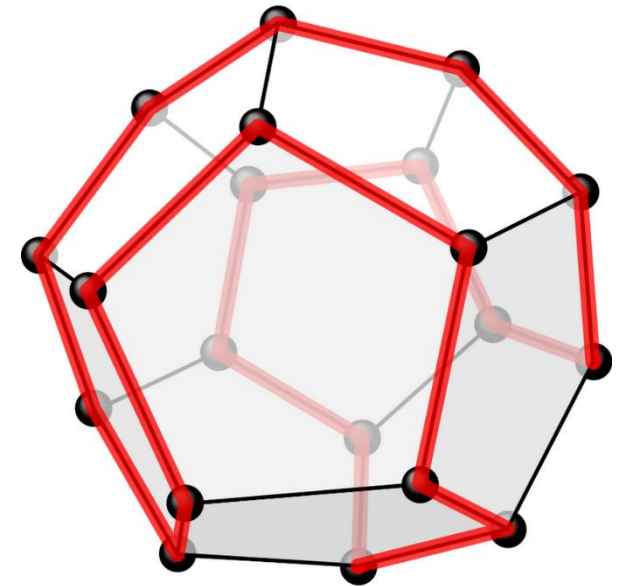
„single-use-path“ mit Türen und „2-buttons“

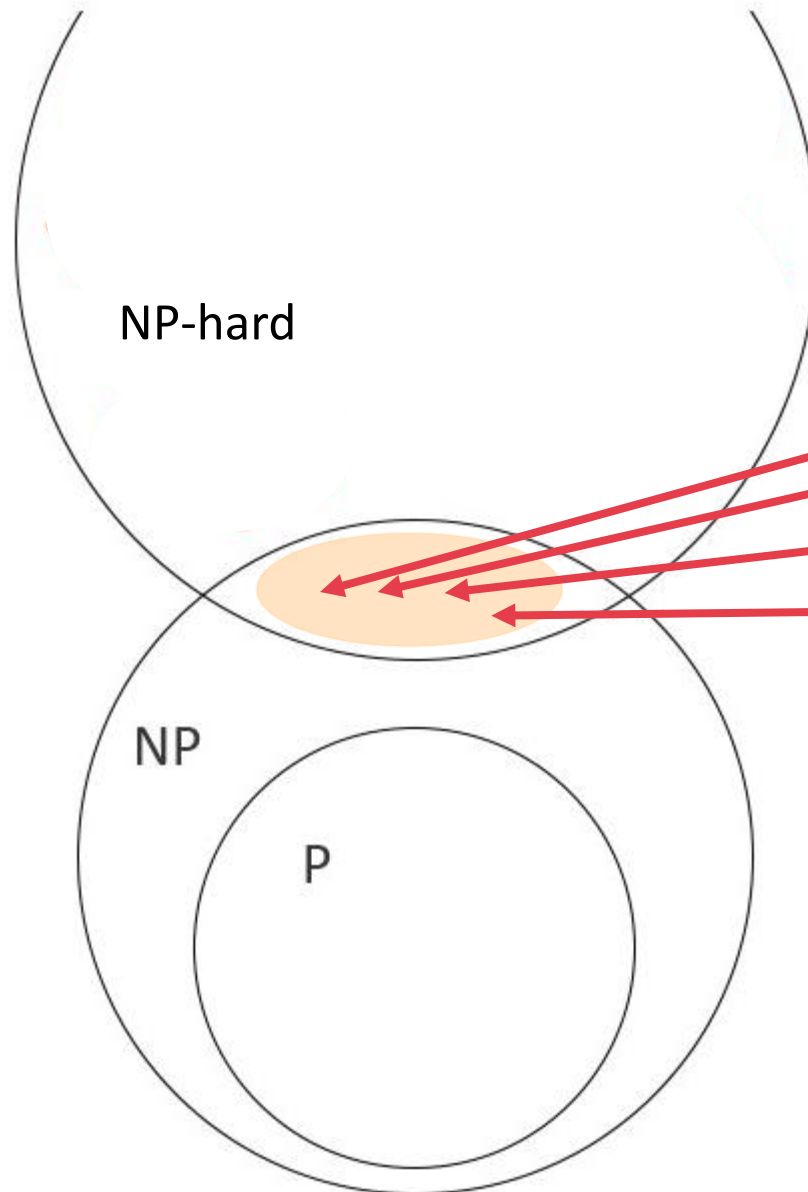


$SAT = \{F \mid F \text{ ist aussagenlogische Formel und erfüllbar}\}$

MINESWEEPER ist NP-vollständig

Hamiltonkreis ist NP-vollständig





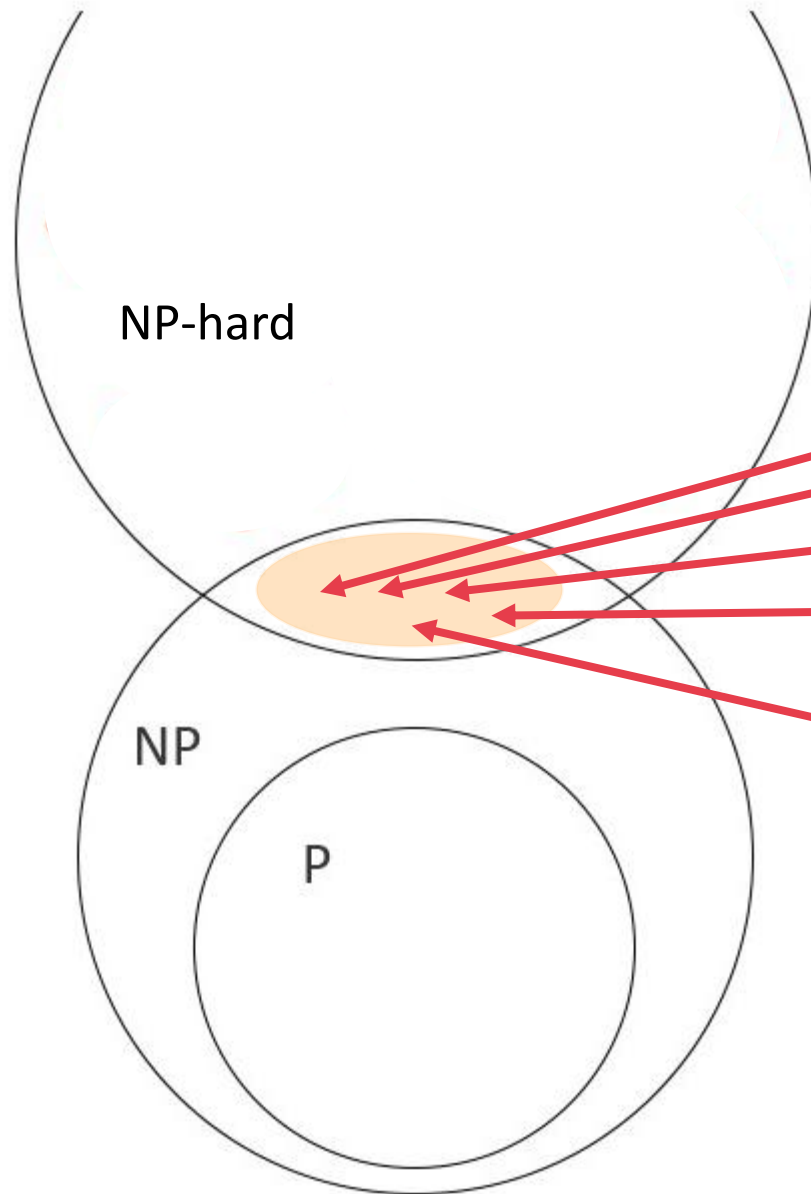
$$SAT = \{F \mid F \text{ ist aussagenlogische Formel und erfüllbar}\}$$

MINESWEEPER ist NP-vollständig

Hamiltonkreis ist NP-vollständig

Traveling Salesman ist NP-vollständig





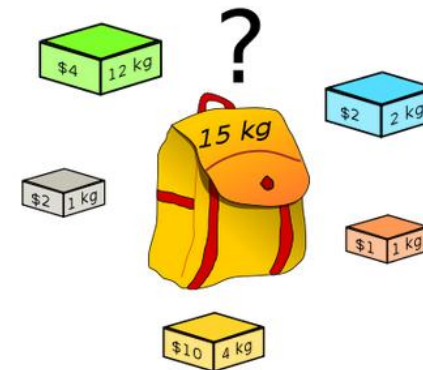
$SAT = \{F \mid F \text{ ist aussagenlogische Formel und erfüllbar}\}$

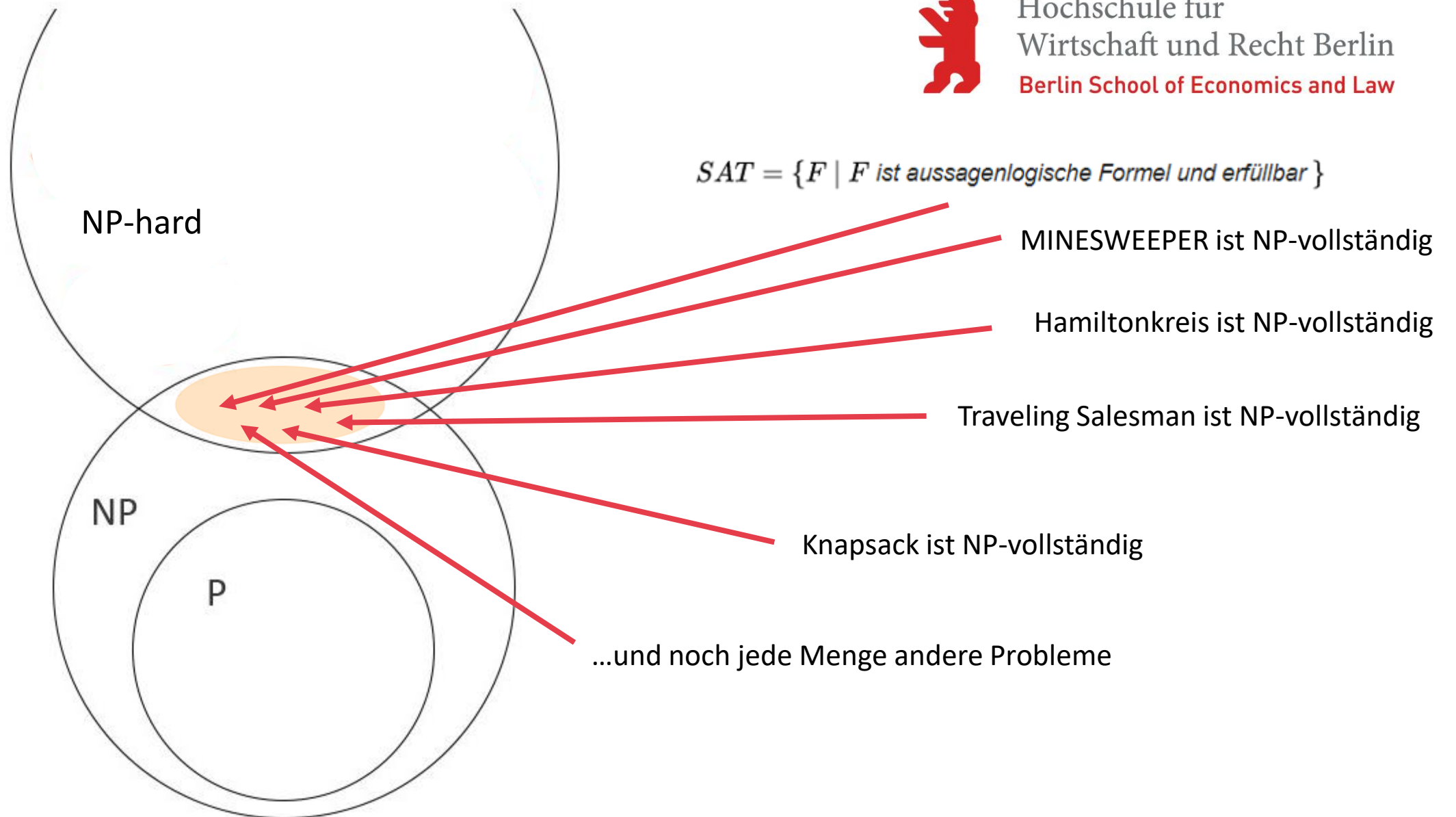
MINESWEEPER ist NP-vollständig

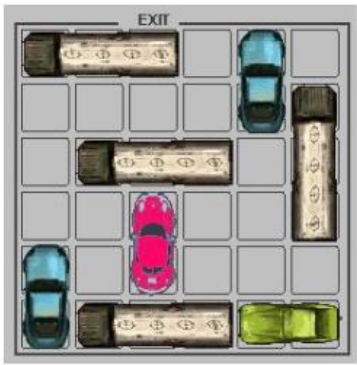
Hamiltonkreis ist NP-vollständig

Traveling Salesman ist NP-vollständig

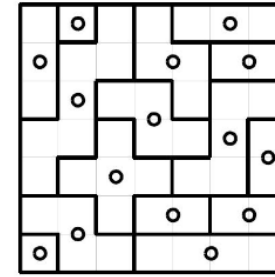
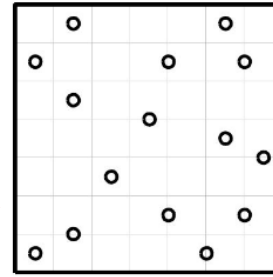
Knapack ist NP-vollständig



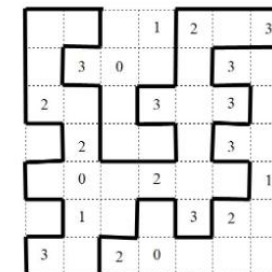
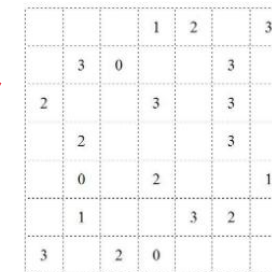
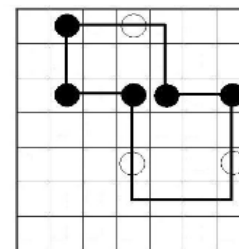
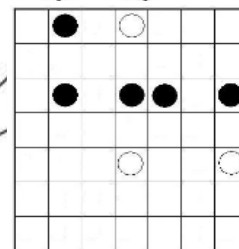
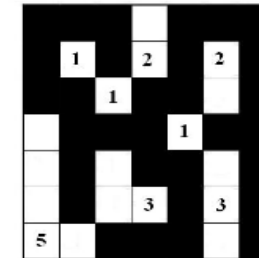
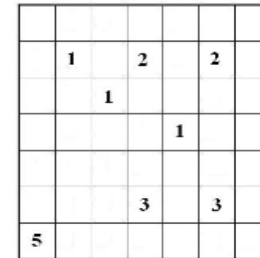
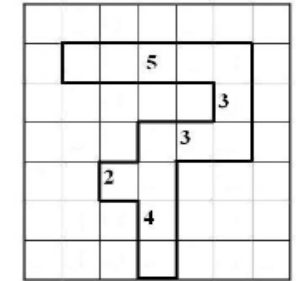
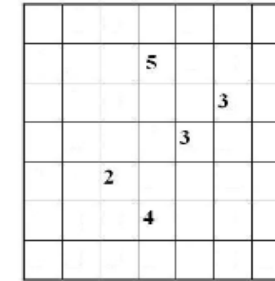
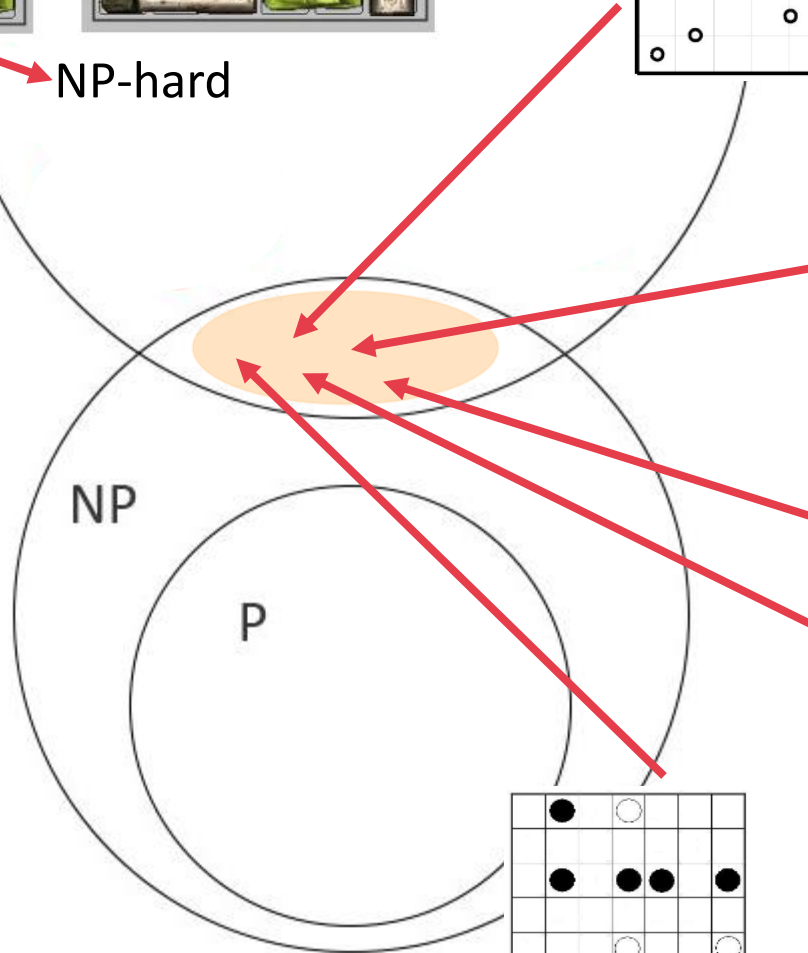




NP-hard



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Karp's List

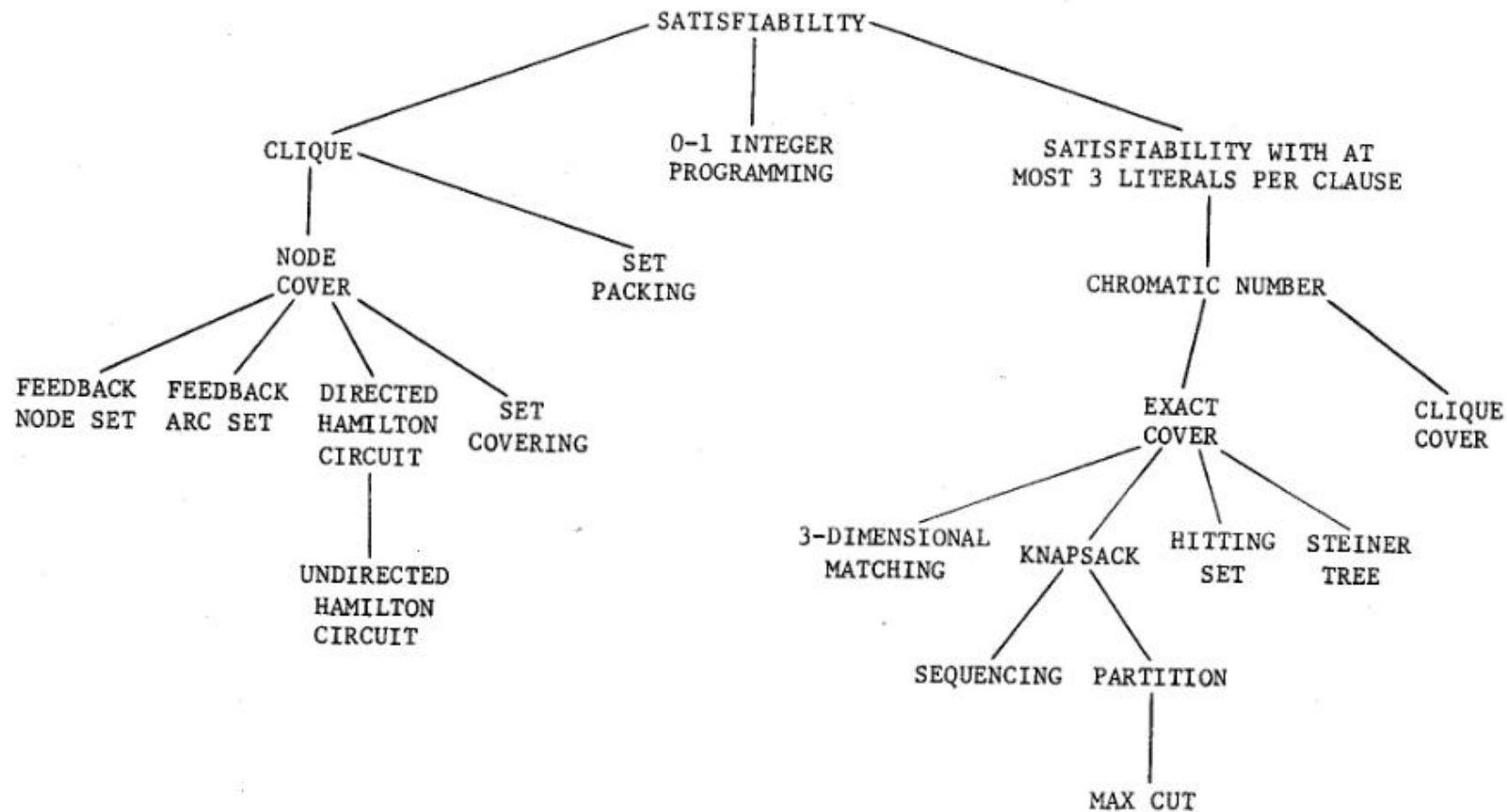


FIGURE 1 - Complete Problems



Space Complexity

Let M be a deterministic Turing machine that halts on all inputs. The **space complexity** of M is the function $f: \mathcal{N} \rightarrow \mathcal{N}$, where $f(n)$ is the **maximum number of tape cells** that M scans on any input of length n . If the space complexity of M is $f(n)$, we also say that M runs in space $f(n)$.

If M is a **nondeterministic** Turing machine wherein all branches halt on all inputs, we define its space complexity $f(n)$ to be the maximum number of tape cells that M scans on **any branch** of its computation for any input of length n .

(in der Literatur wird hier gelegentlich angenommen, dass die TM ein Eingabe- und ein Arbeitsband besitzt und dass **nur der die Zellen auf dem Arbeitsband** gezählt werden)



Space Complexity

Let $f: \mathcal{N} \rightarrow \mathcal{R}^+$ be a function. The *space complexity classes*, **SPACE**($f(n)$) and **NSPACE**($f(n)$), are defined as follows.

$\text{SPACE}(f(n)) = \{L \mid L \text{ is a language decided by an } O(f(n)) \text{ space deterministic Turing machine}\}.$

$\text{NSPACE}(f(n)) = \{L \mid L \text{ is a language decided by an } O(f(n)) \text{ space nondeterministic Turing machine}\}.$

Savitch's Theorem



THEOREM

Savitch's theorem For any function $f: \mathcal{N} \longrightarrow \mathcal{R}^+$, where $f(n) \geq n$,
 $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$.

*gilt auch für
 $f(n) \geq \log(n)$*





Savitch's Theorem

Für eine nichtdet. TM N , welche die Sprache A mit Speicher $f(n)$ entscheidet konstruieren wir eine detem. TM M , welche die Sprache A in $O(f(n)^2)$ entscheidet:

$M =$ "On input w :

1. Output the result of $\text{CANYIELD}(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

Da $f(n)$ zu Beginn nicht bekannt ist, muss M nacheinander $f(n) = 1, 2, 3, \dots$ durchtesten

Jeder Level benötigt $O(f(n))$ Speicher

Die Rekursionstiefe beträgt $O(\log t)$, und damit insgesamt $O(f(n)^2)$

Entscheide ob Konfig. c_2 von Konfig. c_1 in t Schritten erreicht werden kann

$\text{CANYIELD} =$ "On input c_1, c_2 , and t :

1. If $t = 1$, then test directly whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of N . *Accept* if either test succeeds; *reject* if both fail.
2. If $t > 1$, then for each configuration c_m of N on w using space $f(n)$:
3. Run $\text{CANYIELD}(c_1, c_m, \frac{t}{2})$.
4. Run $\text{CANYIELD}(c_m, c_2, \frac{t}{2})$.
5. If steps 3 and 4 both accept, then *accept*.
6. If haven't yet accepted, *reject*."

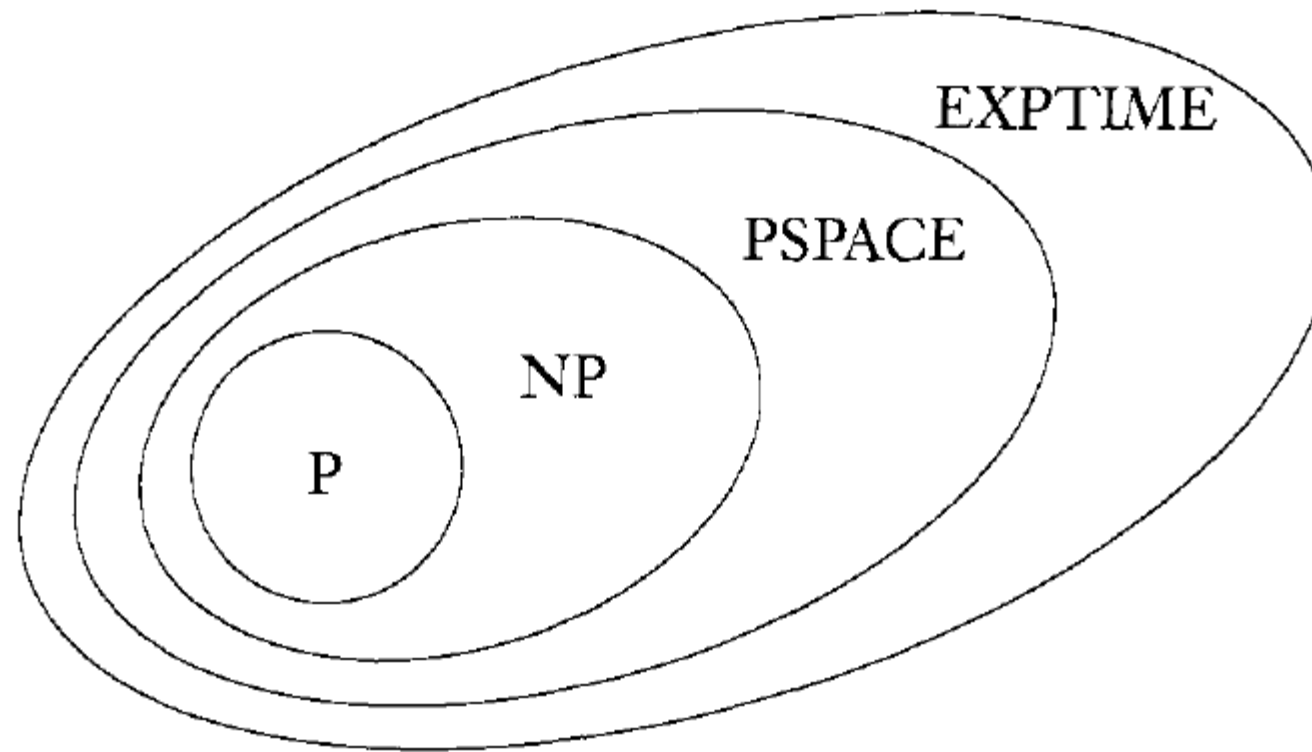
PSPACE



PSPACE is the class of languages that are decidable in **polynomial space on a deterministic** Turing machine. In other words,

$$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k).$$

Time & Space



$$P \subseteq NP \subseteq PSPACE = NPSpace \subseteq EXPTIME = \bigcup_k \text{TIME}(2^{n^k})$$

PSPACE-Vollständigkeit



A language B is **PSPACE-complete** if it satisfies two conditions:

1. B is in PSPACE, and
2. every A in PSPACE is polynomial time reducible to B .

If B merely satisfies condition 2, we say that it is **PSPACE-hard**.

True Quantified Boolean Formulas



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

$TQBF = \{\langle \phi \rangle \mid \phi \text{ is a true fully quantified Boolean formula}\}$

THEOREM

$TQBF$ is PSPACE-complete.

True Quantified Boolean Formulas



- 1) TQBF ist in PSPACE
- 2) Für ein Wort w und eine Sprache A in PSPACE, konstruieren wir eine Formel, welche genau dann erfüllbar ist, falls w in A

Die Idee ist, eine Formel $\phi_{c_1, c_2, t}$ zu konstruieren, welche genau wahr ist, falls von der Konfig. c_1 zur Konfig. c_2 in t Schritten gelangt werden kann. Die Formel $\phi_{c_{\text{start}}, c_{\text{accept}}, h}$ mit $h = 2^{O(n^k)}$ wäre dann die gesuchte Formel.

Ein erster (rekursiver) Ansatz wäre

$$\phi_{c_1, c_2, t} = \exists m_1 \left[\phi_{c_1, m_1, \frac{t}{2}} \wedge \phi_{m_1, c_2, \frac{t}{2}} \right]$$

$$\exists x_1, \dots, x_l$$

wird exponentiell groß

True Quantified Boolean Formulas



- 1) TQBF ist in PSPACE
- 2) Für ein Wort w und eine Sprache A in PSPACE, konstruieren wir eine Formel, welche genau dann erfüllbar ist, falls w in A

Die Idee ist, eine Formel $\phi_{c_1, c_2, t}$ zu konstruieren, welche genau wahr ist, falls von der Konfig. c_1 zur Konfig. c_2 in t Schritten gelangt werden kann. Die Formel $\phi_{c_{\text{start}}, c_{\text{accept}}, h}$ mit $h = 2^{O(n^k)}$ wäre dann die gesuchte Formel.

Verbesserter Ansatz
$$\phi_{c_1, c_2, t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\} [\phi_{c_3, c_4, \frac{t}{2}}]$$

$\forall x \in \{y, z\} [\dots]$ steht für $\forall x [(x = y \vee x = z) \rightarrow \dots]$

muss in der Formel durch
„Äquivalenz“ ersetzt werden

True Quantified Boolean Formulas



- 1) TQBF ist in PSPACE
- 2) Für ein Wort w und eine Sprache A in PSPACE, konstruieren wir eine Formel, welche genau dann erfüllbar ist, falls w in A

Die Idee ist, eine Formel $\phi_{c_1, c_2, t}$ zu konstruieren, welche genau wahr ist, falls von der Konfig. c_1 zur Konfig. c_2 in t Schritten gelangt werden kann. Die Formel $\phi_{c_{\text{start}}, c_{\text{accept}}, h}$ mit $h = 2^{O(n^k)}$ wäre dann die gesuchte Formel.

Verbesserter Ansatz
$$\phi_{c_1, c_2, t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\} [\phi_{c_3, c_4, \frac{t}{2}}]$$

Jeder Rekursionslevel vergrößert die Formel um $O(n^k)$. Die Rekursionstiefe beträgt ebenfalls $O(n^k)$ und damit insgesamt $O(n^{2k})$.

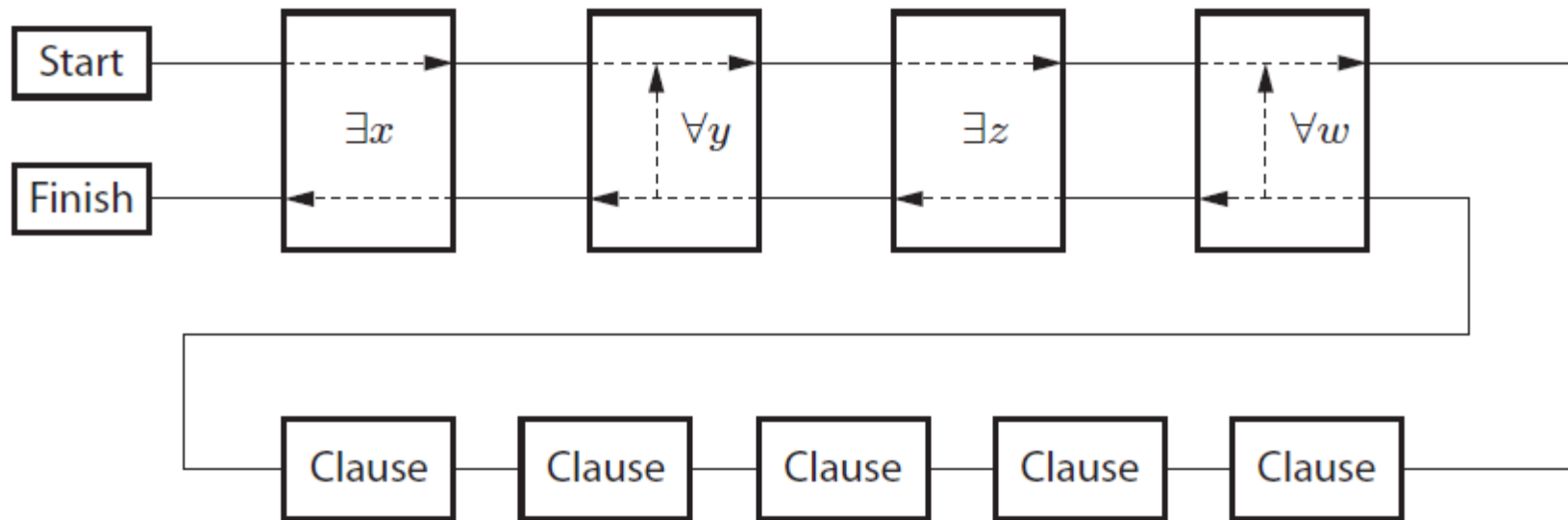


„Gaming is hard“, Part II

Metatheorem 4! *If a game features doors and pressure plates, and the avatar has to reach an exit location in order to win, then:*

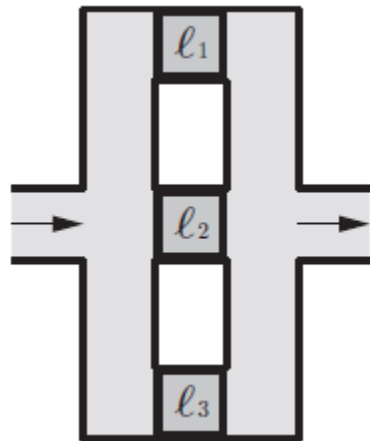
*If each door may be controlled by two pressure plates, then the game is **PSPACE-hard**.*

„Gaming is hard“, Part II

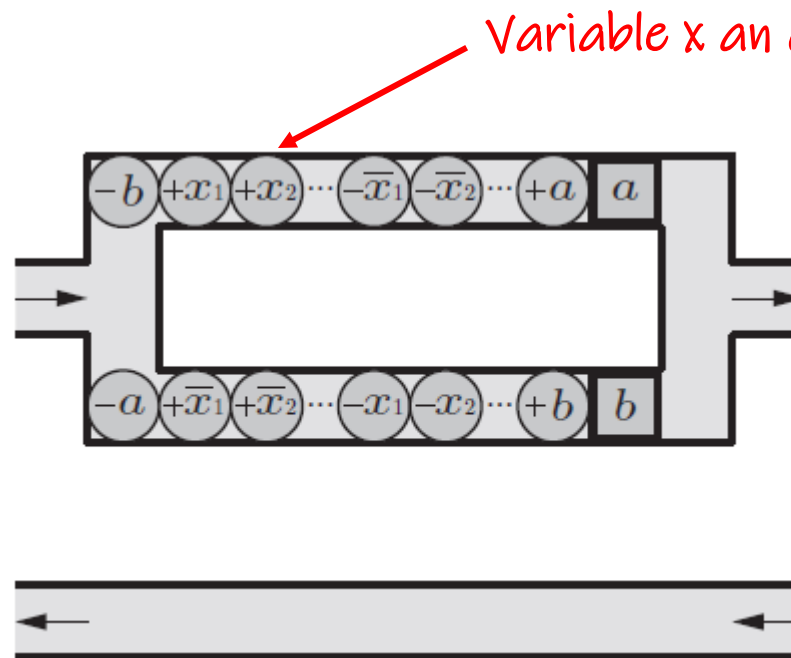


(Aufbau eines Levels, welcher eine TQBF simuliert)

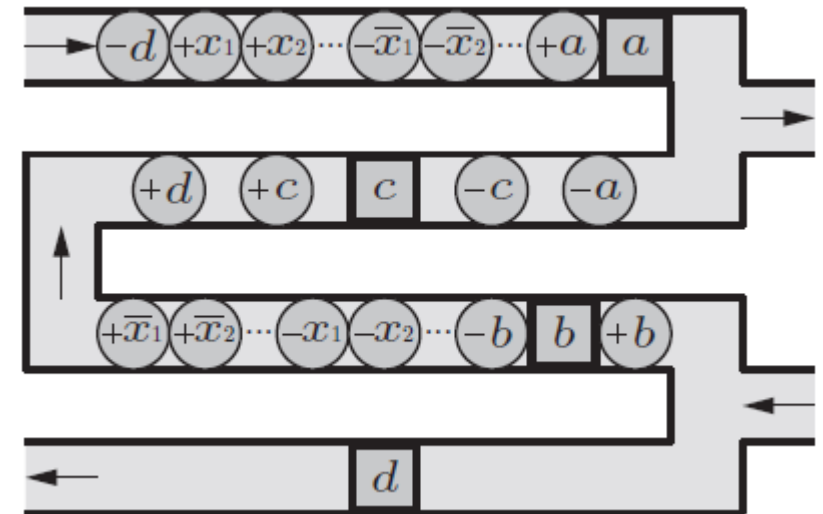
„Gaming is hard“, Part II



Klausel, realisiert
mit Türen



„Existenzquantor“



„Allquantor“



„Gaming is hard“, Part II

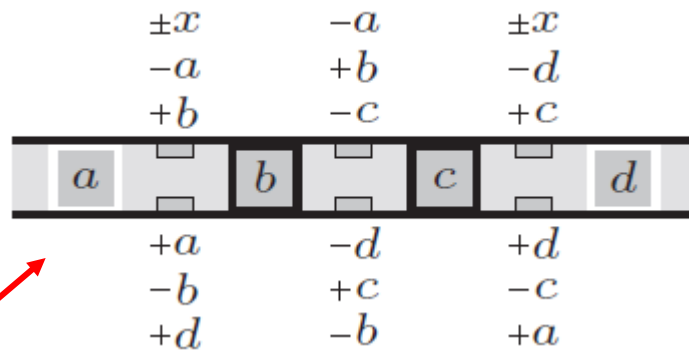
Metatheorem 5! *If a game features doors and k -buttons, and the avatar has to reach an exit location in order to win, then:*

*If $k \geq 3$, then the game is **PSPACE**-hard.*

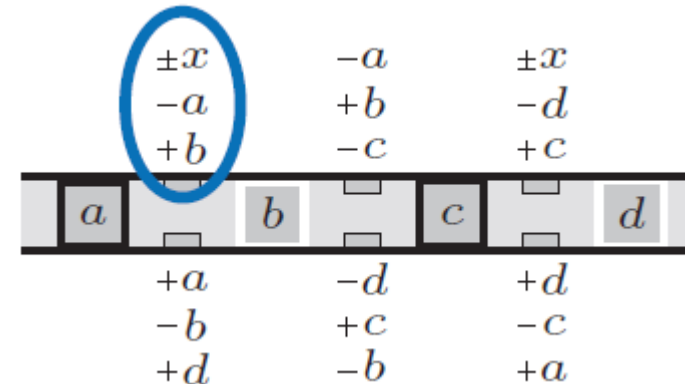
„Gaming is hard“, Part II



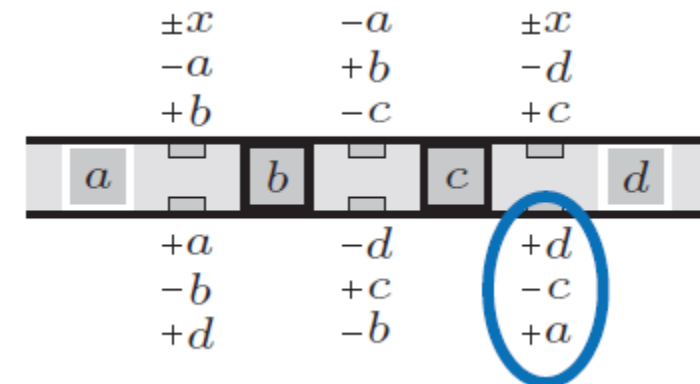
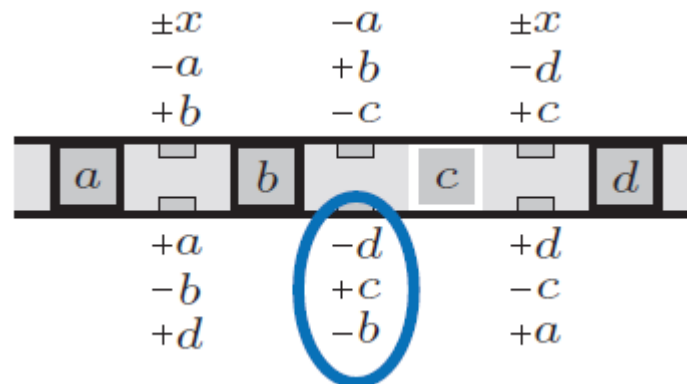
Simulation einer Druckplatte mit „3-buttons“



Startkonfiguration



Spieler kommt von links



Hierarchy Theorems



A function $t: \mathcal{N} \rightarrow \mathcal{N}$, where $t(n)$ is at least $O(n \log n)$, is called **time constructible** if the function that maps the string 1^n to the binary representation of $t(n)$ is computable in time $O(t(n))$.

A function $f: \mathcal{N} \rightarrow \mathcal{N}$, where $f(n)$ is at least $O(\log n)$, is called **space constructible** if the function that maps the string 1^n to the binary representation of $f(n)$ is computable in space $O(f(n))$.

Hierarchy Theorems



THEOREM

Space hierarchy theorem For any space constructible function $f: \mathcal{N} \rightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

Der folgende Algorithmus D entscheidet eine Sprache A mit maximal $O(f(n))$ Speicher:

$D =$ “On input w :

1. Let n be the length of w .
2. Compute $f(n)$ using space constructibility, and mark off this much tape. If later stages ever attempt to use more, *reject*.
3. If w is not of the form $\langle M \rangle 10^*$ for some TM M , *reject*.
4. Simulate M on w while counting the number of steps used in the simulation. If the count ever exceeds $2^{f(n)}$, *reject*.
5. If M accepts, *reject*. If M rejects, *accept*.”

Falls nun, angenommen, eine Maschine M die Sprache A mit $g(n)$ Speicher entscheidet und $g(n)$ in $o(f(n))$, dann ergibt sich hier ein Widerspruch.



Hierarchy Theorems



THEOREM

Space hierarchy theorem For any space constructible function $f: \mathcal{N} \rightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

COROLLARY

For any two real numbers $0 \leq \epsilon_1 < \epsilon_2$,

$$\text{SPACE}(n^{\epsilon_1}) \subsetneq \text{SPACE}(n^{\epsilon_2}).$$

COROLLARY

$$\text{PSPACE} \subsetneq \text{EXPSPACE} = \bigcup_k \text{SPACE}(2^{n^k})$$

Hierarchy Theorems



THEOREM

Time hierarchy theorem For any time constructible function $t: \mathcal{N} \rightarrow \mathcal{N}$, a language A exists that is decidable in $O(t(n))$ time but not decidable in time $o(t(n)/\log t(n))$.

Ähnlich wie beim Space Hierarchy Theorem entscheidet D die Sprache A in $O(t(n))$ Schritten:

$D =$ “On input w :

1. Let n be the length of w .
2. Compute $t(n)$ using time constructibility, and store the value $\lceil t(n)/\log t(n) \rceil$ in a binary counter. Decrement this counter before each step used to carry out stages 3, 4, and 5. If the counter ever hits 0, *reject*.
3. If w is not of the form $\langle M \rangle 10^*$ for some TM M , *reject*.
4. Simulate M on w .
5. If M accepts, then *reject*. If M rejects, then *accept*.”

Das Counterupdate kostet pro Schritt jeweils $O(\log t(n))$.
Damit also insgesamt $O(t(n))$ Schritte.

Sollte eine Maschine M in weniger (wie im Theorem) Schritten entscheiden, so ergäbe sich hier ein Widerspruch.

Hierarchy Theorems



THEOREM

Time hierarchy theorem For any time constructible function $t: \mathcal{N} \rightarrow \mathcal{N}$, a language A exists that is decidable in $O(t(n))$ time but not decidable in time $o(t(n)/\log t(n))$.

COROLLARY

For any two real numbers $1 \leq \epsilon_1 < \epsilon_2$,

$$\text{TIME}(n^{\epsilon_1}) \subsetneq \text{TIME}(n^{\epsilon_2}).$$

COROLLARY

$P \subsetneq \text{EXPTIME}$.

Hierarchy Theorems



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Warum ist die Voraussetzung, dass eine Funktion
„time-constructible“ sein muss notwendig?

Gap Theorem



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

...weil es auch „seltsame“ Funktionen gibt:

Theorem (Gap Theorem, Trakhtenbrot, Borodin) *There exists a computable function f such that $\text{TIME}(f(n)) = \text{TIME}(2^{f(n)})$.*

Gap Theorem



Theorem (Gap Theorem, Trakhtenbrot, Borodin) *There exists a computable function f such that $\text{TIME}(f(n)) = \text{TIME}(2^{f(n)})$.*

Zum Beweis sei $\{M_e\}$ eine Aufzählung aller (unbeschränkten) det. TMs. Wir konstruieren eine Funktion f in Schritten, wobei f in jedem Schritt e folgende Bedingung erfüllen soll:

$$\forall x (|x| = n \geq e \wedge M_e(x) \downarrow \text{ in } t \text{ steps} \implies t \notin (f(n), 2^{f(n)}])$$

Falls nun eine Sprache $A \in \text{TIME}(2^{f(n)})$, was durch eine Maschine M_e bezeugt wird, gilt nach Voraussetzung für alle x der Länge größer als e , dass M_e die Eingabe x in höchstens $f(n)$ Schritten akzeptiert und damit $A \in \text{TIME}(f(n))$.

Gap Theorem



In Schritt n wird $f(n)$ nun wie folgt definiert:

Betrachte folgende Sequenz von Zahlen: $k_0 = 0$, $k_{l+1} = 2^{k_l}$

Für alle Berechnungen $M_e(x)$ mit $e \leq n = \text{Länge von } x$, so dass $M_e(x)$ in $t_{e,x}$ Schritten hält, gibt es dann ein kleinstes k_l , so dass keines der $t_{e,x}$ in dem Intervall $(k_l, 2^{k_l}]$ liegt.

Definiere $f(n) = k_l$.



Speed-Up Theorem

...ebenso gibt es auch „seltsame“ Sprachen:

Theorem (Speed-Up Theorem, M. Blum) *There exists a computable set A such that for every index e for A there is another index i for A such that*

$$\forall^\infty x (\Phi_i(x) \leq \log \Phi_e(x)).$$

Anzahl der Berechnungsschritte,
falls $M_e(x)$ hält.

That is, the program M_i computes A exponentially faster than M_e .

(„ \forall^∞ “ bedeutet hier „fast alle“, also alle, bis auf endlich viele)



Speed-Up Theorem

Zunächst setzen wir $g(x) = 2^x$, $g^{(1)}(x) = g(x)$ und $g^{(n+1)}(x) = g(g^{(n)}(x))$. Für $x > e + 1$ ist dann $h_e(x) = g^{(x-e)}(0)$ damit ist h_e , wegen $g(h_{e+1}(x)) = h_e(x)$, eine Familie abnehmender Funktionen.

Wir bestimmen nun für jedes x den Wert $A(x)$, d.h. ob x zu A gehört oder nicht, indem wir für alle $e \leq x$ testen, ob e noch nicht **markiert („cancelled“)** und ob gilt $\Phi_e(x) < h_e(x)$. Für das kleinste e dieserart definieren wir $A(x) = 1 - M_e(x)$ und markieren e .

↖
Diagonalisierung



Speed-Up Theorem

Zunächst setzen wir $g(x) = 2^x$, $g^{(1)}(x) = g(x)$ und $g^{(n+1)}(x) = g(g^{(n)}(x))$. Für $x > e + 1$ ist dann $h_e(x) = g^{(x-e)}(0)$ damit ist h_e , wegen $g(h_{e+1}(x)) = h_e(x)$, eine Familie abnehmender Funktionen.

Wir bestimmen nun für jedes x den Wert $A(x)$, d.h. ob x zu A gehört oder nicht, indem wir für alle $e \leq x$ testen, ob e noch nicht **markiert („cancelled“)** und ob gilt $\Phi_e(x) < h_e(x)$. Für das kleinste e dieserart definieren wir $A(x) = 1 - M_e(x)$ und markieren e .

Damit gilt für jedes e , dass, falls für unendlich viele x gilt $\Phi_e(x) < h_e(x)$, folgt $M_e \neq A$ (M_e steht hier auch für die Sprache, welche die Maschine akzeptiert). Oder anders geschrieben: $\forall e (M_e = A \implies \forall^\infty x \ h_e(x) \leq \Phi_e(x))$.

← (bedenke aber, dass h_e mit wachsendem e immer kleiner wird)



Speed-Up Theorem

Um nun $A(x)$ zu berechnen, kann man für jedes $e \leq x$ $M_e(x)$ für $h_e(x)$ Schritte laufen lassen. Mit Hilfe der endlichen Menge $F_u = \{(e, x, A(x)) : e < u \wedge e \text{ cancelled at stage } x\}$, für eine natürliche Zahl u , genügt es aber auch, dies nur für $u \leq e \leq x$ zu tun (da die benötigte Information für $e < u$ aus F_u abgelesen werden kann).

Diese Berechnung dauert aber nur $h_u(x) + \dots + h_x(x)$ Schritte, und damit ist die Laufzeit, für die ersten x Stufen, von oben beschränkt durch $x \cdot (h_u(x) + \dots + h_x(x)) \leq h_{u-1}(x)$ (für fast alle x).

Da u beliebig groß gewählt werden kann gilt somit $\forall e \exists i (M_i = A \wedge \forall^\infty x \Phi_i(x) \leq h_{e+1}(x))$ und damit insgesamt

$$\Phi_i(x) \leq h_{e+1}(x) \leq \log h_e(x) \leq \log \Phi_e(x).$$



Orakel Turing Maschine

An **oracle** for a language A is a device that is capable of reporting whether any string w is a member of A . An **oracle Turing machine** M^A is a modified Turing machine that has the additional capability of querying an oracle. Whenever M^A writes a string on a special **oracle tape** it is informed whether that string is a member of A , in a single computation step.

Let P^A be the class of languages decidable with a **polynomial time oracle Turing machine** that uses oracle A . Define NP^A similarly.

Relativierung



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Kann Relativierung dabei helfen
Komplexitätsklassen zu separieren?

Relativierung



THEOREM

1. An oracle A exists whereby $P^A \neq NP^A$.
2. An oracle B exists whereby $P^B = NP^B$.



Relativierung

2. Wähle für B etwa $TQBF$ und damit $NP^{TQBF} \subseteq NPSPACE \subseteq PSPACE \subseteq P^{TQBF}$.

1. Wir konstruieren ein Orakel A , so dass die Sprache

$$L_A = \{w \mid \exists x \in A [|x| = |w|]\}$$

nicht in P^A liegt (offensichtlich liegt sie in NP^A).

Dazu betrachten wir die Liste M_1, M_2, M_3, \dots aller polynomial-Zeit-Orakel-TMs.

Wir konstruieren A in Schritten:

Schritt 1: Wähle endliche viele beliebige Strings und packe sie in A .

Schritt i : Wähle n , so dass 2^n größer als die Laufzeit $p_i(n)$ von M_i und größer als die Länge aller bisherigen Strings in A .

p_i ist ein Polynom

Relativierung



2. Wähle für B etwa $TQBF$ und damit $NP^{TQBF} \subseteq NPSpace \subseteq PSpace \subseteq P^{TQBF}$.

1. Wir konstruieren ein Orakel A , so dass die Sprache

$$L_A = \{w \mid \exists x \in A [|x| = |w|]\}$$

nicht in P^A liegt (offensichtlich liegt sie in NP^A).

Dazu betrachten wir die Liste M_1, M_2, M_3, \dots aller polynomial-Zeit-Orakel-TMs.

Wir konstruieren A in Schritten:

Schritt 1: Wähle endliche viele beliebige Strings und packe sie in A .

Schritt i : Wähle n , so dass 2^n größer als die Laufzeit $p_i(n)$ von M_i und größer als die Länge aller bisherigen Strings in A . Lasse M_i mit Eingabe 1^n laufen. Falls M_i das Orakel nach y fragt und der Status von y feststeht, antworte gemäß dem Status. Falls der Status nicht feststeht antworte mit „Nein“ und setze den Status von y als „nicht in A “. Lass M_i solange laufen bis es hält. Falls M_i akzeptiert, erhalten alle verbleibenden Strings der Länge n den Status „nicht in A “.

Diagonalisierung



Relativierung

2. Wähle für B etwa $TQBF$ und damit $NP^{TQBF} \subseteq NPSpace \subseteq PSpace \subseteq P^{TQBF}$.

1. Wir konstruieren ein Orakel A , so dass die Sprache

$$L_A = \{w \mid \exists x \in A [|x| = |w|]\}$$

nicht in P^A liegt (offensichtlich liegt sie in NP^A).

Dazu betrachten wir die Liste M_1, M_2, M_3, \dots aller polynomial-Zeit-Orakel-TMs.

Wir konstruieren A in Schritten:

Schritt 1: Wähle endliche viele beliebige Strings und packe sie in A .

Schritt i : Wähle n , so dass 2^n größer als die Laufzeit $p_i(n)$ von M_i und größer als die Länge aller bisherigen Strings in A . Lasse M_i mit Eingabe 1^n laufen. Falls M_i das Orakel nach y fragt und der Status von y feststeht, antworte gemäß dem Status. Falls der Status nicht feststeht antworte mit „Nein“ und setze den Status von y als „nicht in A “. Lass M_i solange laufen bis es hält. Falls M_i akzeptiert, erhalten alle verbleibenden Strings der Länge n den Status „nicht in A “. Falls M_i nicht akzeptiert, so wähle ein Element, nach welchem M_i das Orakel nicht gefragt hat und füge es zu A hinzu. Fahre mit Schritt $i+1$ fort.



Polynomial Hierachy

Für eine Klasse C definieren wir $\exists C = \{x : \exists^{p(|x|)} y \langle x, y \rangle \in B\}$ (und analog für den Allquantor).

„es existiert ein String der
Länge $\leq p(|x|)$ “

Demnach gilt also z.B. $\exists P = NP$, bzw. $\forall P = \text{co-NP}$.

Für eine Klasse C ist ein Problem in
 $\text{Co-}C$, falls das *duale Problem* in C ist

Wir setzen jetzt $\Sigma_0^P = \Pi_0^P = P$, und dann für alle n :

$$\Sigma_{n+1}^P = \exists \Pi_n^P$$

$$\Pi_{n+1}^P = \forall \Sigma_n^P$$

Polynomial Hierachy



For every $i \geq 1$, a language L is in Σ_i^P if there exists a polynomial-time TM M and a polynomial q such that

$$x \in L \Leftrightarrow \exists u_1 \in \{0, 1\}^{q(|x|)} \forall u_2 \in \{0, 1\}^{q(|x|)} \dots Q_i u_i \in \{0, 1\}^{q(|x|)} M(x, u_1, \dots, u_i) = 1,$$

die
Quantoren
alternieren
hier

where Q_i denotes \forall or \exists depending on whether i is even or odd respectively.

We say that L is in Π_i^P if there exists a polynomial-time TM M and a polynomial q such that

$$x \in L \Leftrightarrow \forall u_1 \in \{0, 1\}^{q(|x|)} \exists u_2 \in \{0, 1\}^{q(|x|)} \dots Q_i u_i \in \{0, 1\}^{q(|x|)} M(x, u_1, \dots, u_i) = 1,$$

where Q_i denotes \exists or \forall depending on whether i is even or odd respectively.

The *polynomial hierarchy* is the set $\mathbf{PH} = \cup_i \Sigma_i^P$.

Polynomial Hierachy



Sei $NP^C = NP(\mathcal{C}) = \bigcup \{NP^C : C \in \mathcal{C}\}$. Dann gilt für jedes $n > 0$: $\Sigma_{n+1}^P = NP(\Sigma_n^P)$.

Für jedes S in Σ_{n+1}^P gibt es nach Definition ein S' aus Π_n^P , sodass $S = \{x, \text{ex-poly. } y \text{ mit } (x,y) \text{ in } S'\}$.
Damit ist S aber in $NP(\Pi_n^P) = NP(\Sigma_n^P)$.

Sei umgekehrt S in $NP(\Sigma_n^P)$. Dann existiert zu einer Eingabe x also eine Reihe von Queries $q_i(x,y)$ an ein Orakel S' aus Σ_n^P . Da diese adaptiv sein können nehmen wir an, dass die TM die Antworten $a_i(x,y)$ schon im Vorfeld rät. Es gilt dann, dass x in S , genau dann wenn

$$\exists y \left(A(x, y) \wedge \bigwedge_{i=1}^{q(x,y)} \left((a^{(i)}(x, y) = 1) \Leftrightarrow (q^{(i)}(x, y) \in S') \right) \right) \in \Sigma_{n+1}^P$$

es existiert eine Reihe von Queries

so dass die TM x akzeptiert

und die geratenen Antworten mit den Antworten des Orakels übereinstimmen

Polynomial Hierachy



Vollständiges Problem für Level i:

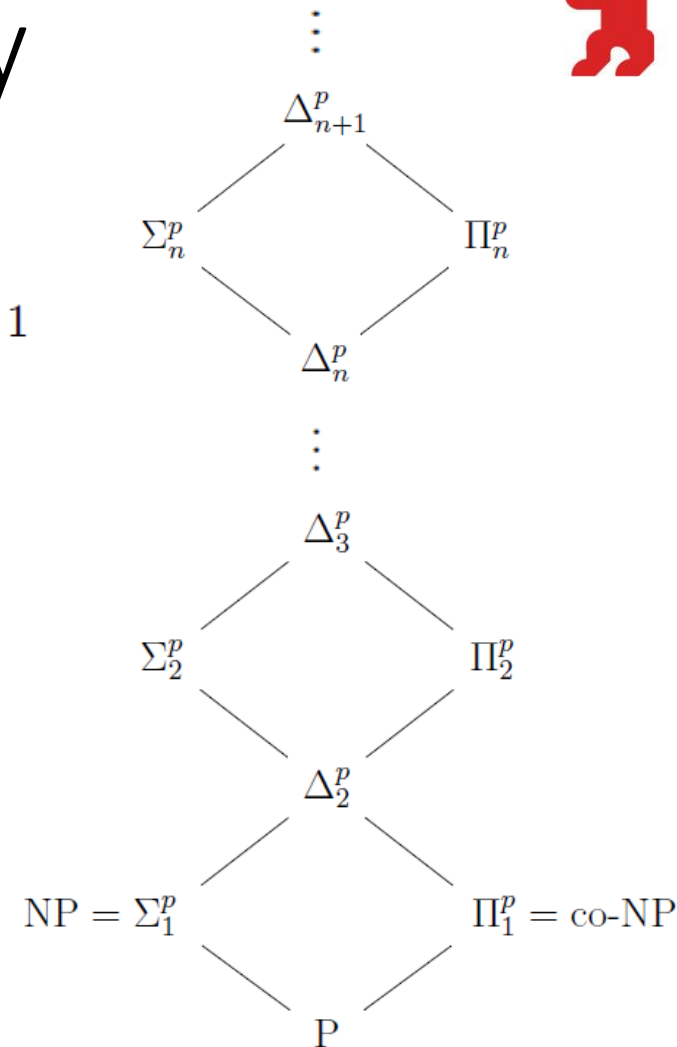
$$\Sigma_i \text{SAT} = \exists u_1 \forall u_2 \exists \dots Q_i u_i \varphi(u_1, u_2, \dots, u_i) = 1$$

Falls es ein vollständiges Problem für PH
geben sollte, so kollabiert PH.

es ist leicht einzusehen, dass

$$\text{PH} \subseteq \text{PSPACE}$$

Falls jedoch $\text{PH} = \text{PSPACE}$
gelten sollte, so kollabiert PH.



$$\Sigma_0^P = \Pi_0^P = P.$$

$$\Sigma_{n+1}^P = \text{NP}(\Sigma_n^P).$$

$$\Pi_{n+1}^P = \text{co-}\Sigma_{n+1}^P.$$

$$\Delta_{n+1}^P = P(\Sigma_n^P).$$

$$\text{PH} = \bigcup_{n \geq 0} \Sigma_n^P.$$