Formale Grundlagen der Informatik 3

Exkurs: Komplexitätstheorie

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Stefan Katzenbeisser
Security Engineering Group
Technische Universität Darmstadt

skatzenbeisser@acm.org
http://www.seceng.informatik.tu-darmstadt.de

SECENG SECURITY ENGINEERING

Komplexität von Algorithmen und Komplexität von Problemen

TECHNISCHE UNIVERSITÄT DARMSTADT

Komplexität von Algorithmen:

- Analyse eines bestimmten Algorithmus
- Konkrete Laufzeit angängig von Implementierung, daher meist Nutzung von O-Notation
  - LTL-Model Checking: $O(|\mathcal{M}|2^{|\varphi|})$ CTL: $O(|\mathcal{M}|\ |\varphi|)$

Komplexität von Problemen:

- Es kann mehrere Algorithmen geben die das gleiche Ergebnis erzielen, aber drastisch unterschiedliche Laufzeit haben
  ➔ Beispiel: Sortieralgorithmen
- Versuch die „minimale" Komplexität des Problems zu bestimmen
- Problem: „optimale" Algorithmen sind meist nicht bekannt!

SECENG SECURITY ENGINEERING

Turing-Maschinen (1)

Quelle: http://aturingmachine.com

Turing-Maschinen (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Charakterisiert durch:

- Endlicher Zustand
- Unendliches Band, das in einzelne Zellen eingeteilt ist
- Jede Zelle enthält ein Symbol
- Ein Berechnungsschritt:
  - Liest das „aktuelle" Symbol auf dem Band
  - Bestimmt das neu zu schreibende Symbol basierend auf aktuellem Zustand und aktuellem Bandsymbol; endliches Regelwerk!
  - Schreibt das „neue" Symbol; wechselt Zustand
  - Verschiebt das Band potentiell einen Schritt nach links oder rechts.
- Maschine terminiert falls Endzustand erreicht wird.

➔ Formale Definition siehe FGDI1

SECENG SECURITY ENGINEERING

Entscheidungsprobleme

TECHNISCHE UNIVERSITÄT DARMSTADT

- Fokus der Komplexitätstheorie sind **Entscheidungsprobleme**
  - Antwort binär: trifft zu/nicht zu
  - Beispiele: Ist die SAT-Formel erfüllbar? Ist eine Kripke-Struktur Modell einer Formel? Ist der kürzeste Pfad zwischen zwei Knoten in einem Graphen kleiner als 5 Schritte? …

- Formalisierung durch **Sprachen**
  - Gegeben: Sprache $L \subseteq \Sigma^*$ über einem endlichen Alphabet Σ
  - Wir codieren Instanzen eines Problems als Element in *L*
  - Lösung eines konkreten Problems erfordert Entscheidung ob es in *L* („positiv“) oder $\Sigma^* \setminus L$ („negativ“) liegt.

SECENG SECURITY ENGINEERING

Deterministische und Nichtdeterministische Turing-Maschinen (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

Deterministische TM:

- Zu jedem internen Zustand der TM (Zustand und Bandsymbol) gibt es genau eine Möglichkeit die Berechnung fortzusetzen.
- Lösung des Entscheidungsproblems:
  - Probleminstanz wird vor Berechnung auf Band geschrieben
  - Maschine terminiert
  - Maschine endet entweder in akzeptierendem oder nicht akzeptierendem Zustand
- **Zeitkomplexität**: Zähle die Zahl der Zustandsübergänge einer TM

Eine deterministische TM entscheidet die Sprache *L* in Zeit $f(n)$ falls die TM für alle Instanzen $x \in L$ nach maximal $f(|x|)$ Schritten in einem akzeptierendem Zustand und für alle $x \notin L$ nach maximal $f(|x|)$ in einem nicht-akzeptierenden Zustand endet.

SECENG SECURITY ENGINEERING

Deterministische und Nichtdeterministische Turing-Maschinen (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Deterministische TM:

Eine deterministische TM entscheidet die Sprache *L* in Zeit $f(n)$ falls die TM für alle Instanzen $x \in L$ nach maximal $f(|x|)$ Schritten in einem akzeptierendem Zustand und für alle $x \notin L$ nach maximal $f(|x|)$ in einem nicht-akzeptierenden Zustand endet.

Platzkomlexität:

- Zähle die Anzahl der „benutzen" Zellen auf TM-Band bis TM

Eine deterministische TM entscheidet die Sprache *L* in Platz $f(n)$ falls die TM für alle Instanzen $x \in L$ in einem akzeptierenden und für alle Instanzen $x \notin L$ in einem nicht akzeptierenden Zustand endet und dabei maximal $f(|x|)$ Zellen des Bandes nutzt.

SECENG SECURITY ENGINEERING

Deterministische und Nichtdeterministische Turing-Maschinen (3)

TECHNISCHE UNIVERSITÄT DARMSTADT

Deterministische Komplexitätsklassen:

Die Klasse $\mathbf{TIME}(f(n))$ ist die Menge aller Sprachen $L \subseteq \Sigma^*$ für die eine deterministische Turing-Maschine existiert, die *L* in Zeit $f(n)$ entscheidet.

Die Klasse $\mathbf{SPACE}(f(n))$ ist die Menge aller Sprachen $L \subseteq \Sigma^*$ für die eine deterministische Turing-Maschine existiert, die *L* in Platz $f(n)$ entscheidet.

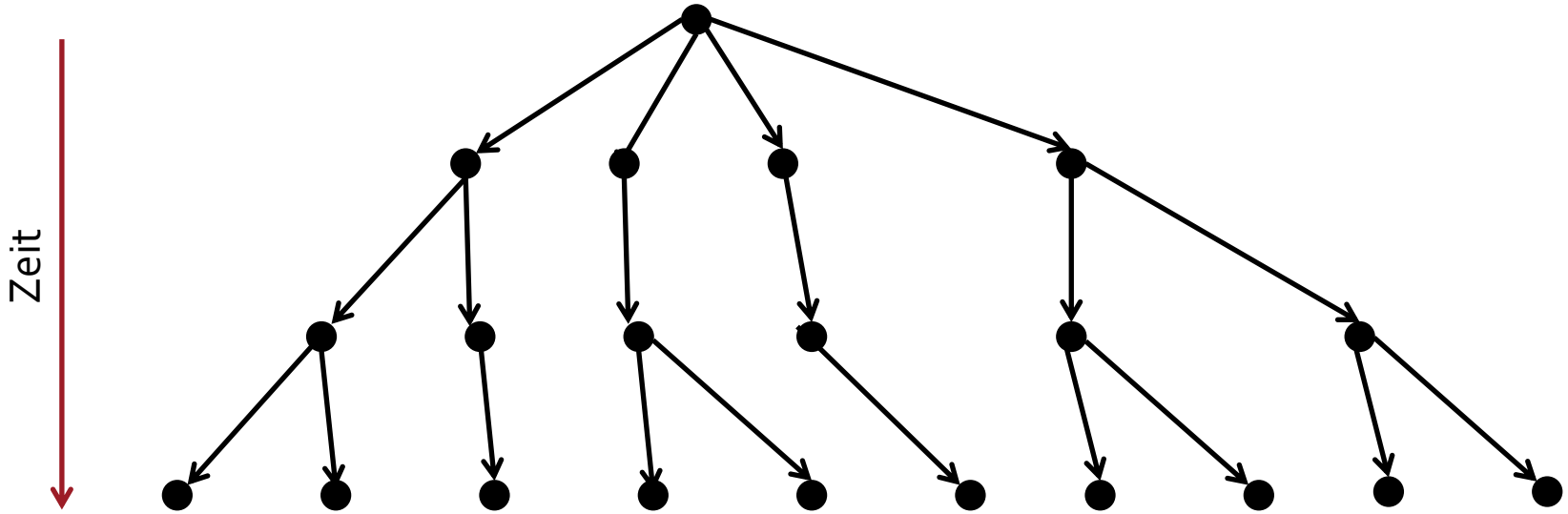
SECENG SECURITY ENGINEERING

Deterministische und Nichtdeterministische Turing-Maschinen (4)

TECHNISCHE UNIVERSITÄT DARMSTADT

Nichtdeterministische TM:

- Zu jedem internen Zustand der TM (Zustand und Bandsymbol) gibt es mehrere Möglichkeiten die Berechnung fortzusetzen.
- Berechnungsbaum: Knoten sind Konfigurationen der Maschine, Kanten mögliche Nachfolger



SECENG SECURITY ENGINEERING

Deterministische und Nichtdeterministische Turing-Maschinen (5)

TECHNISCHE UNIVERSITÄT DARMSTADT

Nichtdeterministische TM:

- Für „positive" Probleminstanzen reicht ein akzeptierender Pfad aus!
- Zeitkomplexität ist die maximale Tiefe des Baumes.

Eine nichtdeterministische TM entscheidet die Sprache *L* in Zeit $f(n)$ falls es für alle $x \in L$ einen Berechnungspfad gibt, der in einem akzeptierenden Zustand endet sowie für alle $x \notin L$ alle Berechnungspfade in einem nicht-akzeptierenden Zustand enden und alle Pfade maximal $f(|x|)$ Schritte lang sind.

Eine nichtdeterministische TM entscheidet die Sprache *L* in Platz $f(n)$ falls es für alle $x \in L$ einen Berechnungspfad gibt, der in einem akzeptierenden Zustand endet sowie für alle $x \notin L$ alle Berechnungspfade in einem nicht-akzeptierenden Zustand enden und in jedem Pfad maximal $f(|x|)$ Zellen des Bandes nutzt.

SECENG SECURITY ENGINEERING

Deterministische und Nichtdeterministische Turing-Maschinen (6)

TECHNISCHE UNIVERSITÄT DARMSTADT

Nichtdeterministische Komplexitätsklassen:

Die Klasse $\mathbf{NTIME}(f(n))$ ist die Menge aller Sprachen $L \subseteq \Sigma^*$ für die eine nichtdeterministische Turing-Maschine existiert, die *L* in Zeit $f(n)$ entscheidet.

Die Klasse $\mathbf{NSPACE}(f(n))$ ist die Menge aller Sprachen $L \subseteq \Sigma^*$ für die eine nichtdeterministische Turing-Maschine existiert, die *L* in Platz $f(n)$ entscheidet.

SECENG SECURITY ENGINEERING

Konkrete Komplexitätsklassen

TECHNISCHE UNIVERSITÄT DARMSTADT

Zeitkomplexität:

- Polynomielle Zeit: $\mathbf{P} = \mathbf{TIME}(n^k)$
- Nichtdeterministisch-polynomielle Zeit: $\mathbf{NP} = \mathbf{NTIME}(n^k)$
- Exponentielle Zeit: $\mathbf{EXP} = \mathbf{TIME}(2^{n^k})$

Platzkomplexität

- Logarithmischer Platz: $\mathbf{L} = \mathbf{SPACE}(\log n)$
- Nichtdeterministischer logarithmischer Platz: $\mathbf{NL} = \mathbf{NSPACE}(\log n)$
- Polynomieller Platz: $\mathbf{PSPACE} = \mathbf{SPACE}(n^k)$
- Exponentieller Platz: $\mathbf{EXPSPACE} = \mathbf{SPACE}(2^{n^k})$

SECENG SECURITY ENGINEERING

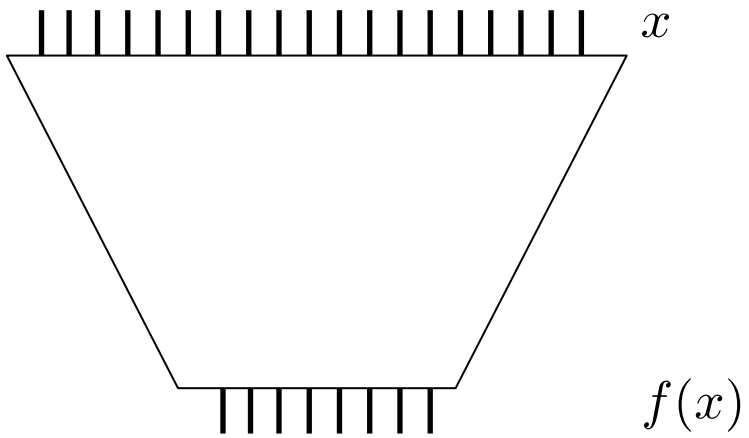
Typische Probleme: Klasse P

TECHNISCHE UNIVERSITÄT DARMSTADT

- **P** enthält alle in polynomieller Zeit lösbaren Entscheidungsprobleme
- **CIRCUITVALUE** liegt in **P**

CIRCUITVALUE

Eingabe: Boolescher Schaltkreis und Werte aller Drähte der Eingabe
Ausgabe: (Boolescher) Wert des Schaltkreises

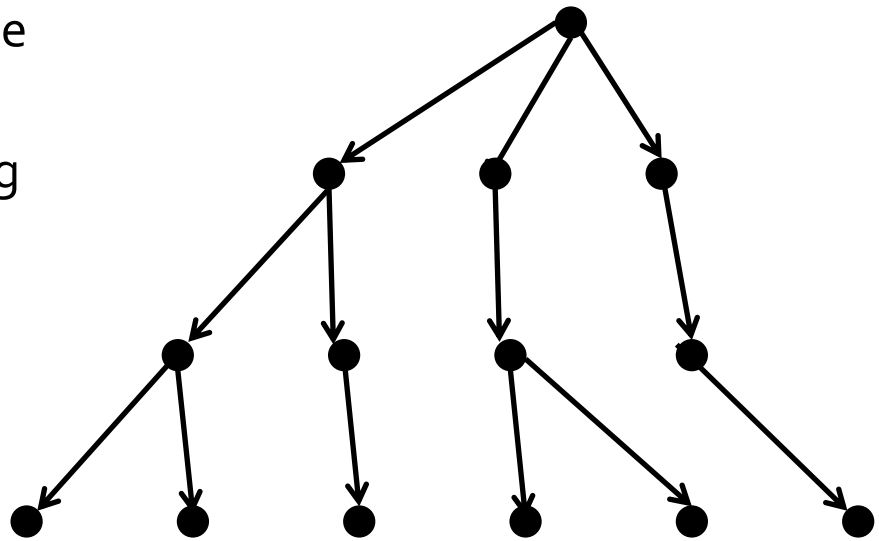


SECENG SECURITY ENGINEERING

Klasse NP

- **NP** enthält alle Probleme, die von einer nichtdeterministischen TM in polynomieller Zeit gelöst werden können.

- Äquivalent: **NP** enthält alle Probleme bei denen man in polynomieller Zeit eine Lösung „verifizieren kann".
- Idee: rate einen „Zeugen" für die Lösung und verifiziere dass diesen Zeugen; ist beides polynomiell möglich so ist das Problem in **NP**.
- „Zeuge" kann Pfad im Berechnungs-baum sein.



Typische Probleme: Klasse NP

TECHNISCHE UNIVERSITÄT DARMSTADT

- **NP** enthält alle Probleme, die von einer nichtdeterministischen TM in polynomieller Zeit gelöst werden können.
- **SAT** liegt in **NP**

SAT

Eingabe: Aussagenlogische Formel

Ausgabe: 1 falls Formel erfüllbar ist, d.h. eine Belegung der Variablen existiert die die Formel zu TRUE evaluiert

Beweisidee: „rate" Belegung und verifiziere dass diese Belegung die Formel zu TRUE evaluieren lässt. Existiert daher eine „erfolgreiche" Belegung, so gibt es einen Berechnungspfad der in einen akzeptierenden Zustand führt; gibt es keine „erfolgreiche" Belegung so enden alle Pfade in einem nicht-akzeptierenden Zustand.

SECENG SECURITY ENGINEERING

Typische Probleme: Klasse NL

TECHNISCHE UNIVERSITÄT DARMSTADT

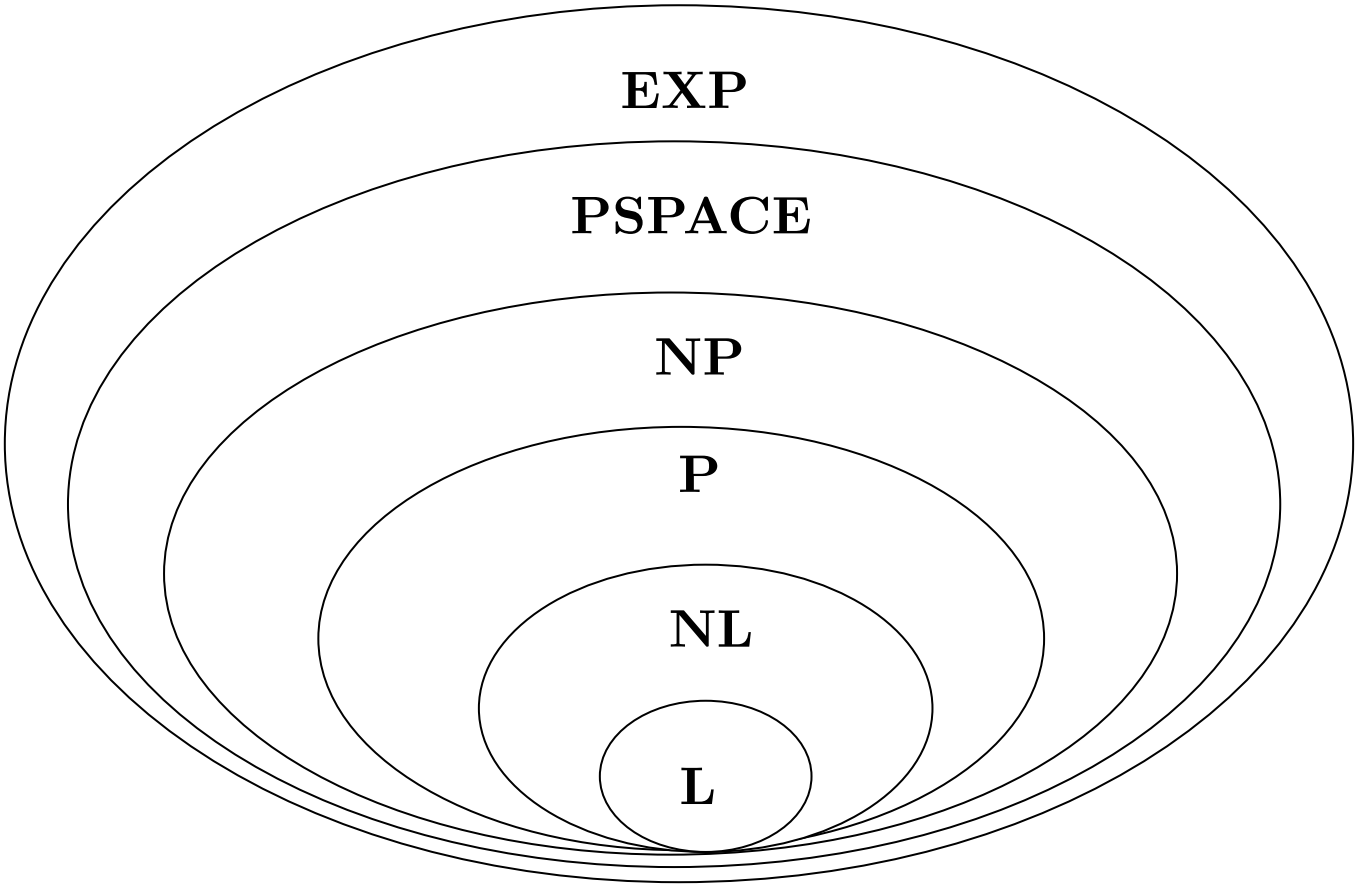
- **NL** enthält alle Probleme, die von einer nichtdeterministischen TM mit logarithmischem Platz gelöst werden können.
- **REACHABILITY** liegt in **NL**

REACHABILITY
Eingabe: Graph G sowie zwei Knoten x und y
Ausgabe: 1 falls in G zwischen x und y ein Pfad existiert

Beweisidee: Schreibe x aufs Band; rate iterativ die Nummer eines neuen Knotens z und teste ob dieser von x erreichbar ist: falls z=y, gib 1 aus, falls z nicht erreichbar gib 0 aus, sonst ersetze y mit z und führe die nächste Iteration aus. Nach |G| Iterationen bricht der Algorithmus mit Ausgabe 0 ab.
➔ Gibt es einen Pfad, so gibt es einen Ablauf des Algorithmus der 1 ausgibt!
Platz (=2 Knotennummern!) ist maximal logarithmisch in der Größe der Eingabe

SECENG SECURITY ENGINEERING

Relationen zwischen den Klassen



Aber: $\mathbf{P} \subset \mathbf{EXP}$

Es gilt nach Definition:

$\mathbf{L} \subseteq \mathbf{NL}$

$\mathbf{P} \subseteq \mathbf{NP}$

Für jede Space-Klasse gilt:

$\mathbf{NSPACE}(f(n)) \subseteq \mathbf{TIME}(k^{f(n)+\log n})$

$\mathbf{NTIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$

Daher:

$\mathbf{NL} \subseteq \mathbf{P}$

$\mathbf{NP} \subseteq \mathbf{PSPACE}$

Reduktionen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Problem**: wie kann man die Komplexität verschiedener Probleme vergleichen?
- Der „bestmögliche" Algorithmus ist oft nicht bekannt!
- Daher: Vergleich zwischen Problemen in einer Klasse die beste Option
- Reduktion ermöglicht das „Umschreiben" einer Probleminstanz in eine andere.
- Reduktion selbst muss ein „effizientes" Verfahren sein
  - … in der Regel ein polynomieller Algorithmus
  - … oder sogar schwächer: logspace-Reduktionen

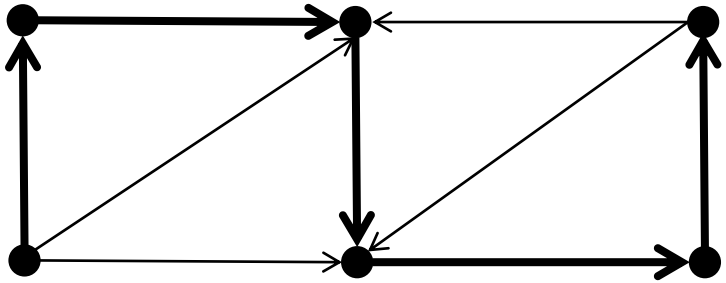
SECENG SECURITY ENGINEERING

Beispiel: Hamilton-Pfad in einem Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

HAMILTON PATH
Eingabe: Graph G
Ausgabe: 1 wenn ein Pfad existiert, der alle Knoten von G besucht, sonst 0

Beispiel:



- Reduziere **HAMILTON PATH** auf **SAT**
- Wenn wir **SAT** lösen können, dann auch **HAMILTON PATH** mit einem ähnlich hohen Aufwand!

SECENG SECURITY ENGINEERING

Reduktion: HAMILTON PATH auf SAT (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Idee: Konstruiere eine SAT-Formel, die genau dann erfüllbar ist wenn es in einem Graphen einen Hamilton-Pfad gibt
- Nutzung von Variablen x_{ij}
- Intuitiv: die i-te Position des Hamilton-Pfades wird von Knoten j eingenommen
- Konstruktion der SAT-Formel:
  - Jeder Knoten muss auf dem Pfad vorkommen: für alle j:

    $x_{1j} \vee x_{2j} \vee \ldots \vee x_{nj}$
  - Kein Knoten kommt zwei Mal vor: für alle $i, j, k,\ i \neq k$:

    $\neg x_{ij} \vee \neg x_{kj}$
  - Jede Position im Pfad muss „besetzt" sein: für alle i:

    $x_{i1} \vee x_{i2} \vee \ldots \vee x_{in}$
  - ...

SECENG SECURITY ENGINEERING

Reduktion: HAMILTON PATH auf SAT (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- ...
- Keine zwei Knoten besetzen die gleiche Stelle des Pfades: für alle $i, j, k,\ j \neq k$
  $\neg x_{ij} \vee \neg x_{ik}$
- Nicht benachbarte Knoten können nicht benachbart im Pfad sein:
  $\neg x_{ki} \vee \neg x_{k+1,j}$
  für alle $(i, j) \neq G,\ k = 1, 2, \ldots, n-1$
- Alle so generierten Disjunktionen werden durch Konjunktionen verbunden; man erhält eine CNF

- So erzeugte Formel ist nur dann erfüllbar wenn ein Hamilton-Pfad existiert
- Größe der Formel: $O(n^3)$

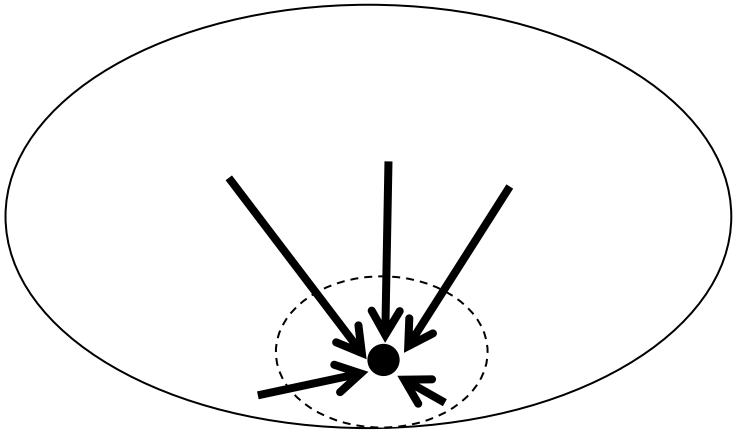
SECENG SECURITY ENGINEERING

Vollständigkeit (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Idee: bestimme die „**härtesten**" Probleme in einer Klasse

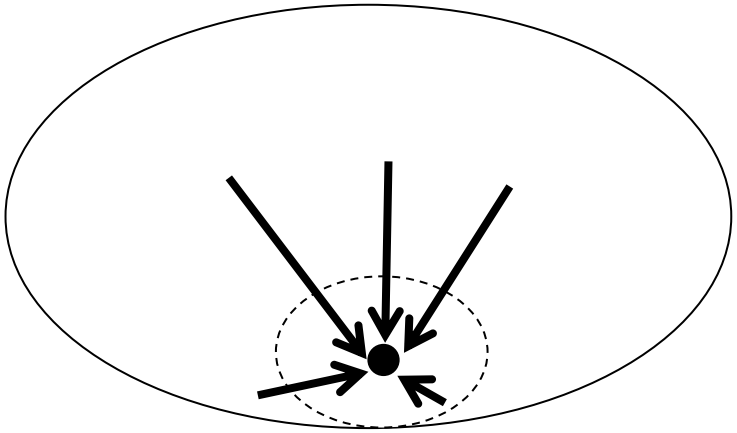
- Ein Problem ist vollständig für eine Klasse wenn…
  - es in der Klasse liegt und
  - alle anderen Probleme der Klasse auf das gegebene Problem reduziert werden können.



SECENG SECURITY ENGINEERING

Vollständigkeit (2)

- Ist ein Problem für eine Klasse vollständig, so ist es „genau so“ hart wie alle anderen Probleme der Klasse.
- Es ist daher „unwahrscheinlich“ dass effizientere Algorithmen existieren – es sei denn zwei Klassen fallen gänzlich zusammen.
- Offenes Problem der Informatik: Ist **P=NP**?



Vollständiges Problem für NP (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

SAT ist NP-vollständig

Beweisidee (Satz von Cook-Levin):

- **SAT** ist in NP: NP-Maschine rät Belegung aller Variablen und verifiziert dass Belegung korrekt ist
- Alle anderen Probleme in **NP** lassen sich auf **SAT** reduzieren
  - Transformiere Berechnungsbaum einer beliebigen NP-TM in SAT-Formel
  - Variablen beschreiben Konfiguration der TM zu einem gewissen Zeitpunkt
  - Formel beschreibt Startzustände, Bandinhalte, Endbedingungen und Übergangsrelation
  - Formel ist genau dann erfüllbar wenn es einen akzeptierenden Berechnungspfad gibt
  - Länge der Formel ist polynomiell

SECENG SECURITY ENGINEERING

Vollständiges Problem für NP (2)
Satz von Cook-Levin

TECHNISCHE UNIVERSITÄT DARMSTADT

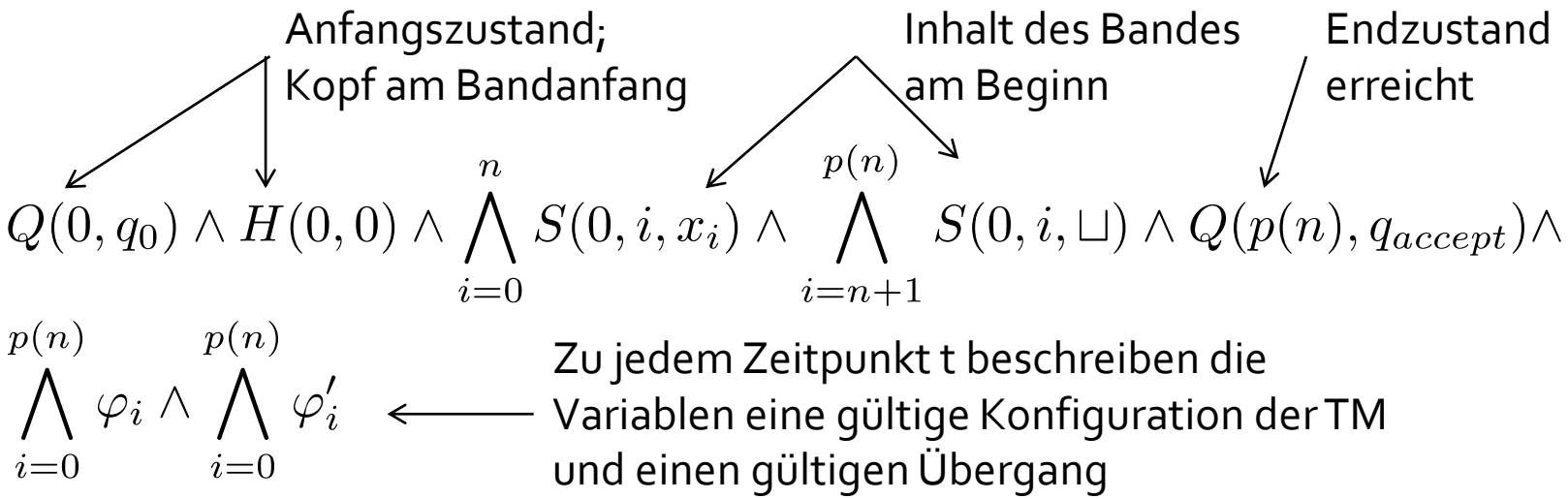
Nutze folgende Variable:

$Q(t,k)$ ... zum Zeitpunkt t befindet sich die TM im Zustand k

$H(t,j)$ ... zum Zeitpunkt t befindet sich der Lesekopf im Zustand j

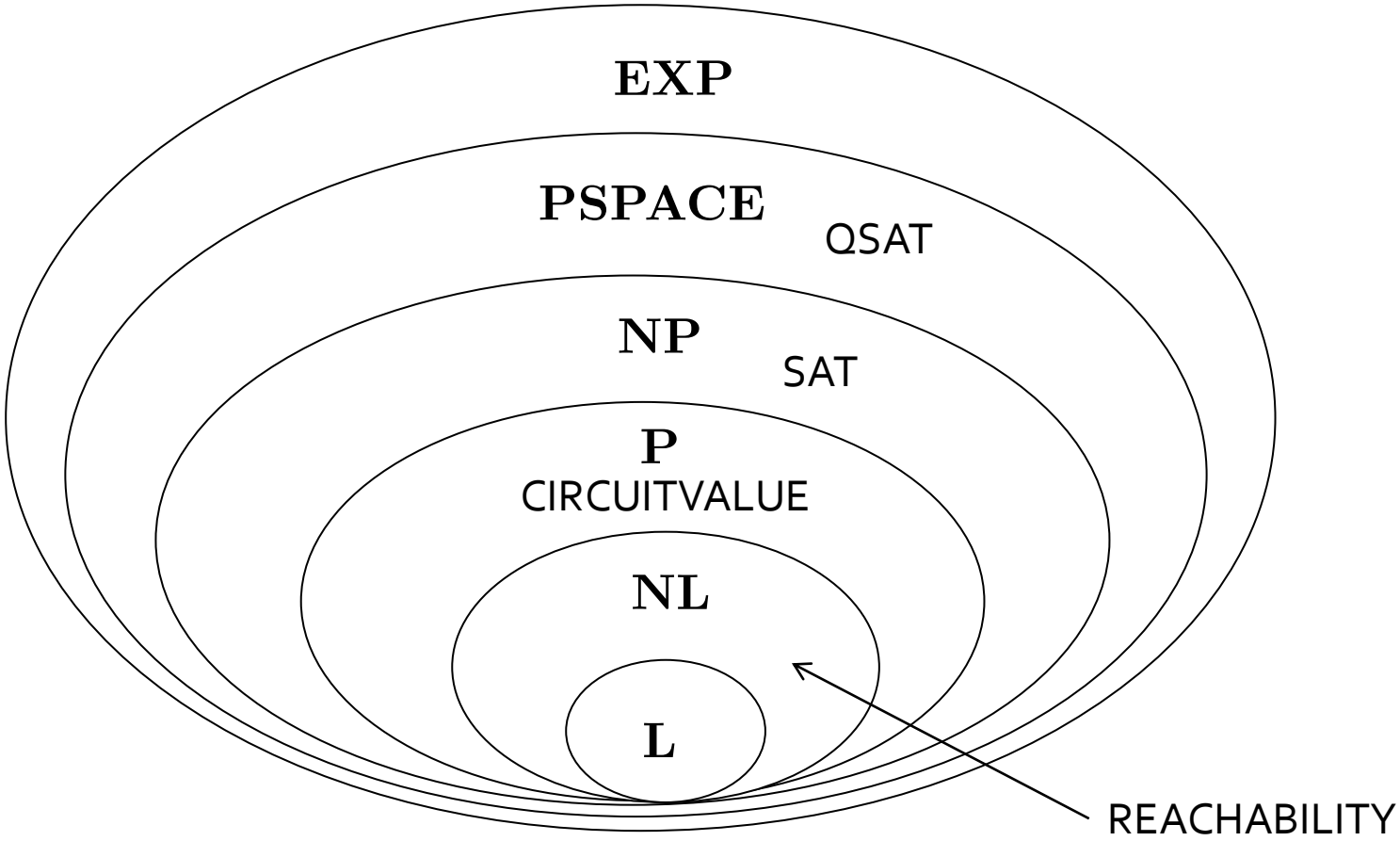
$S(t,j,a)$ ... zum Zeitpunkt t steht auf Zelle j des Bandes der Inhalt a

Resultierende Formel:



SECENG SECURITY ENGINEERING

Vollständige Probleme



Komplexität der Model-Checking Probleme

TECHNISCHE UNIVERSITÄT DARMSTADT

CTL Model-Checking ist P-vollständig

LTL Model-Checking ist PSPACE-vollständig

CTL* Model-Checking ist PSPACE-vollständig

SECENG SECURITY ENGINEERING