# Das Zugriffsmatrix-Modell
Version 0.9

Carsten Schwarz*
Humboldt-Universität zu Berlin
Institut für Informatik

14. Dezember 2003

**Zusammenfassung**

Dieser Text befasst sich mit dem von Michael A. Harrison, Walter L. Ruzzo und Jeffrey D. Ullmann in [HRU76] beschriebenen Zugriffsmatrix-Modell. Dieses bietet die Möglichkeit, Objekte und Subjekte angepasst an die jeweiligen Anforderungen der zu konstruierenden Anwendung festzulegen, sowie Zugriffsrechte universell oder objektspezifisch zu modellieren. Im Folgenden werden einige Sicherheitsprobleme untersucht und gezeigt, dass das Safety-Problem unentscheidbar ist. Das Modell wird bewertet und in die von Claudia Eckert in [Eck03] vorgestellte Klassifikation einsortiert.

## 1 Vorstellung des Modells

Im Jahre 1976 haben Michael A. Harrison, Walter L. Ruzzo (beide University of California, Berkeley) und Jeffrey D. Ullmann (Princeton University) in [HRU76] das Zugriffsmatrix-Modell (Access Matrix Model / Referenzmonitor-Modell) vorgestellt. Dieses Modell ist das einfachste und älteste Sicherheitsmodell und bietet die Möglichkeit, Objekte und Subjekte angepasst an die jeweiligen Anforderungen der zu konstruierenden Anwendung festzulegen, sowie Zugriffsrechte universell oder objektspezifisch zu modellieren.

**Definition 1 (Zugriffsmatrix)** *Der Schutzzustand eines Systems zum Zeitpunkt $t$ wird durch eine $|S_t| \times |O_t|$-Matrix $P_t$ modelliert, wobei gilt: die Spalten der Matrix werden durch die Menge $O_t$ der Objekte und die Zeilen der Matrix durch die Menge $S_t$ der Subjekte zum Zeitpunkt $t$ definiert. Es gilt: $P_t : S_t \times O_t \rightarrow 2^R$, wobei $R$ die Menge der Zugriffsrechte festlegt. Ein Eintrag beschreibt die Menge der Rechte, die das Subjekt $s$ zum Zeitpunkt $t$ an dem Objekt $o$ besitzt.*

| | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $S_1$ | | | |
| $S_2$ | | $\{R_1, R_2\}$ | |
| $S_3$ | | | |

Tabelle 1: Beispiel Zugriffsmatrix 1

In Tabelle 1 bspw. besitzt das Subjekt $S_2$ die Rechte $R_1$ und $R_2$ am Objekt $O_2$, also $M_t(S_2, O_2) = \{R_1, R_2\}$.

*caschwar@informatik.hu-berlin.de

| | file 1 | file 2 | proc 1 | proc 2 | proc 3 | proc 4 |
|---|---|---|---|---|---|---|
| proc 1 | | | | control, send | | control |
| proc 2 | | | wait, signal | | control | |
| proc 3 | write | own | | receive | | send |
| proc 4 | | read | | | send | |

Tabelle 2: Beispiel Zugriffsmatrix 2

In der Zugriffsmatrix in Tabelle 2 sind Sende- und Empfangs-Rechte (send und receive) für Prozesse, Rechte zur Ausführung von Synchronisationsoperationen (wait und signal), sowie ein Eigentümerrecht (control) modelliert. Des Weiteren können die Prozesse noch das Besitz-, Lese- und Schreiberecht (own, read und control) an Dateien haben.
Der Prozess 3 bspw. hat das Besitzrecht an Datei 2 und darf daher auch Rechte an Datei 2 an andere Prozesse vergeben. Weiter kann Prozess 3 Nachrichten von Prozess 2 empfangen und an Prozess 4 senden.

Eine Matrix $M_t$ kann durch die Ausführung von Kommandos zur Erzeugung und zum Löschen von Subjekten bzw. Objekten oder zur Weitergabe bzw. Rücknahme von Zugriffsrechten verändert werden. Harrison, Ruzzo und Ullmann definieren dazu in [HRU76] sechs Elementaroperationen:

enter $r$ into $(X_s, X_o)$
delete $r$ from $(X_s, X_o)$
create subject $X_s$
create object $X_o$
destroy subject $X_s$
destroy subject $X_o$

Aus diesen Elementaroperationen werden die Kommandos des konkreten Schutzsystems zusammengesetzt:

**Definition 2 (Schutzsystem)** *Ein Schutzsystem besteht aus einer endlichen Menge von Rechten $R$ und einer endlichen Menge $C$ von Kommandos der Form:*

```
command α(X1, X2, ..., Xk)
  if
    r1 in (Xs1, Xo1) and
    r2 in (Xs2, Xo2) and
    ...
    rm in (Xsm, Xom)
  then
    op1
    op2
    ...
    opn
end
```

*oder im Falle $m = 0$:*

```
command α(X1, X2, ..., Xk)
    op1
    op2
    ...
    opn
end
```

Zum Beispiel können die Subjekte Prozesse und die Objekte (außer den Objekten, die auch Subjekte sind) Dateien sein. Die Subjekte können das Besitzrecht (own) an einer Datei haben oder das Recht, eine Datei zu lesen, zu schreiben oder auszuführen (read, write und execute). Nun können aus den Elementarkommandos Kommandos erstellt werden, die eine neue Datei erzeugen (CREATE), Rechte vergeben (CONFER) und Rechte entziehen (REMOVE):

```
command CREATE (process, file)
    create object file
    enter own into (process, file)
end

command CONFER_r (owner, friend, file)
    if own in (owner, file)
    then enter r into (friend, file)
end

command REMOVE_r (owner, exfriend, file)
    if own in (owner, file) and r in (exfriend, file)
    then delete r from (exfriend, file)
end
```

Ein spezielles Schutzsystem setzt sich also aus einer Menge von Rechten und einer Menge von Kommandos zusammen. Diese Kommandos werden zwar aus den sechs Elementarkommandos gebildet, unterscheiden sich aber von Schutzsystem zu Schutzsystem.
Um den Effekt der Elementaroperationen formal definieren zu können, benötigen wir den Begriff der Systemkonfiguration:

**Definition 3 (Konfiguration)** *Eine Systemkonfiguration (zum Zeitpunkt t) ist festgelegt durch ein Tripel* $(S_t, O_t, P_t)$*, wobei* $P_t$ *die Zugriffsmatrix und* $O_t$ *und* $S_t$ *die (zu diesem Zeitpunkt existierenden) Mengen von Objekten und Subjekten sind.*

Der Effekt der Elementaroperationen formal definiert ist:

**Definition 4** *Seien* $(S, O, P)$ *und* $(S', O', P')$ *Konfigurationen eines Schutzsystems und sei op eine Elementaroperation. Wir definieren*

$$(S, O, P) \Rightarrow (S', O', P')$$

*wenn entweder*

1. *op = enter r into* $(s, o)$ *und* $S = S', O = O', s \in S, o \in O, P'[s_1, o_1] = P[s_1, o_1]$ *wenn* $(s_1, o_1) \neq (s, o)$ *und* $P'[s, o] = P[s, o] \cup \{r\}$*, oder*
2. *op = delete r from* $(s, o)$ *und* $S = S', O = O', s \in S, o \in O, P'[s_1, o_1] = P[s_1, o_1]$ *wenn* $(s_1, o_1) \neq (s, o)$ *und* $P'[s, o] = P[s, o] - \{r\}$*, oder*
3. *op = create subject* $s'$*, wobei* $s'$ *ein neues Zeichen und nicht in* $O$ *ist,* $S' = S \cup \{s'\}, O' = O \cup \{s'\}, P'[s, o] = P[s, o]$ *für alle* $(s, o)$ *aus* $S \times O$*,* $P'[s', o] = \emptyset$ *für alle* $o \in O'$*, und* $P[s, s'] = \emptyset$ *für alle* $s \in S'$*, oder*
4. *op = create object* $o'$*, wobei* $o'$ *ein neues Zeichen und nicht in* $O$ *ist,* $S' = S, O' = O \cup \{o'\}, P'[s, o] = P[s, o]$ *für alle* $(s, o)$ *aus* $S \times O$ *und* $P'[s, o'] = \emptyset$ *für alle* $s \in S$*, oder*
5. *op = destroy subject* $s'$*, wobei* $s' \in S, S' = S - \{s'\}, O' = O - \{s'\}$*, und* $P'[s, o] = P[s, o]$ *für alle* $(s, o) \in S' \times O'$*, oder*
6. *op = destroy object* $o'$*, wobei* $o' \in O - S, S' = S, O' = O - \{o'\}$*, und* $P'[s, o] = P[s, o]$ *für alle* $(s, o) \in S' \times O'$*.*

# 2 Untersuchung der Sicherheitseigenschaften

## 2.1 Safety-Problem

Das Safety-Problem erfasst die Fragestellung, ob ausgehend von einem gegebenen Schutzzustand $M_t$ ein Subjekt $s$ das Recht $r$ an dem Objekt $o$ erhalten kann, wenn es dieses Recht im Zustand $M_t$ noch nicht besitzt und die Erlangung dieses Rechts im Sicherheitssystem nicht vorgesehen ist.

**Definition 5 (Safety-Problem)** *Systemkonfiguration sei $K_t = (S_t, O_t, P_t)$, Konfigurationsübergang sei $K_t \models^{op} K_{t+1}$ (wird durch op bewirkt), $K_0$ sei Anfangskonfiguration und für $K_t$ gelte: $r \notin M_t(s, o)$. Das Safety-Problem reduziert sich auf die Frage, ob von der Konfiguration $K_t$ durch Ausführung von Kommandos $op_1, \ldots, op_n$, also*

$$K_0 \models^* K_t \models^{op_1} K_{t+1} \models^{op_2} \ldots \models^{op_n} K_{t+n},$$

*eine Konfiguration $K_{t+n}$ erreicht werden kann, mit $r \in M_{t+n}(s, o)$.*

**Satz 1** *Das Safety-Problem ist unentscheidbar. D. h., es gibt keinen Algorithmus, der bei gegebener Anfangskonfiguration eines Systems mit einer beliebig festgelegten Menge von Kommandos entscheiden kann, ob ein Recht in den Besitz eines Subjekts gelangen kann.*

*Beweis:* Wir führen das Safety-Problem auf das unentscheidbare Halteproblem von Turing-Maschinen[1] zurück.
Angenommen, es gäbe einen Entscheidungsalgorithmus für das Safety-Problem. Wir zeigen, dass es dann auch einen Algorithmus für das Halteproblem gäbe. Dies ist ein Widerspruch. (Das Halteproblem ist unentscheidbar.[2]) Also gibt es keinen Entscheidungsalgorithmus für das Safety-Problem.

Als ersten Schritt müssen wir die Konfiguration der zu simulierenden Turing-Maschine in die Konfiguration einer Zugriffsmatrix umwandeln:
Dabei soll die Menge der Rechte unseres Schutzsystems die Menge der Zustände ($Z$) und die Menge der Bandinschriften ($\Gamma$) der zu simulierenden Turing-Maschine umfassen. Zu jedem Zeitpunkt besitzt die Turing-Maschine eine endliche Anzahl von Bandfeldern $(1, 2, \ldots, k)$, auf denen sich der Schreib-Lesekopf der Turing-Maschine bereits befunden hat. Diese Situation kann durch $k$ Subjekte $s_1, s_2, \ldots, s_k$ simuliert werden, indem $s_i$ das Besitzrecht (own) an $s_{i+1}$ für alle $1 \leq i < k$ besitzt. Wir benutzen also das Besitzrecht, um die Subjekt linear zu ordnen und so das Band der Turing-Maschine zu simulieren. Das Subjekt $s_i$ repräsentiert das Bandfeld $i$. Die Tatsache, dass das Bandfeld $i$ die Inschrift $X$ trägt wird dadurch repräsentiert, dass das Subjekt $s_i$ das Recht $X$ an sich selbst besitzt. Die Tatsache, dass $q$ der momentane Zustand der Turing-Maschine ist und der Schreib-Lesekopf sich auf Feld $j$ befindet, wird simuliert, indem das Subjekt $s_j$ das Recht $q$ an sich selbst hat.[3]
Ein weiteres Recht (end) markiert das letzte Subjekt $s_k$ bzw. zeigt an, dass bisher kein weiteres Subjekt $s_{k+1}$ erzeugt wurde.
Eine Turing-Maschine mit der Bandinschrift $WXYZ$, deren Schreib-Lesekopf sich auf dem zweiten Feld befindet und die sich im Zustand $q$ befindet, würde durch die Zugriffsmatrix in Tabelle 3 simuliert.

Im zweiten Schritt muss die Überführungsfunktion der Turing-Maschine durch die Elementaroperationen der Zugriffsmatrix simuliert werden:

[1]siehe Anhang A
[2]siehe Anhang B
[3]Wir setzen voraus, dass die Menge der Zustände und das Arbeitsalphabet disjunkt sind.

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|
| $o_1$ | $\{W\}$ | $\{\text{own}\}$ | | |
| $o_2$ | | $\{X, q\}$ | $\{\text{own}\}$ | |
| $o_3$ | | | $\{Y\}$ | $\{\text{own}\}$ |
| $o_4$ | | | | $\{Z, \text{end}\}$ |

Tabelle 3: Simulation Turing-Maschine durch Zugriffsmatrix 1

Dazu wird $\delta(q, X) = (p, Y, N)$ überführt in:

```
command A_qX(s, s')
  if
    own in (s, s') and
    q in (s', s') and
    X in (s', s')
  then
    delete X from (s', s')
    enter Y into (s', s')
end
```

$\delta(q, X) = (p, Y, L)$ wird überführt in:

```
command B_qX(s, s')
  if
    own in (s, s') and
    q in (s', s') and
    X in (s', s')
  then
    delete q from (s', s')
    delete X from (s', s')
    enter p into (s, s)
    enter Y into (s', s')
end
```

$\delta(q, X) = (p, Y, R)$ wird überführt in:

```
command C_qX(s, s')
  if
    own in (s, s') and
    q in (s, s) and
    X in (s, s)
  then
    delete q from (s, s)
    delete X from (s, s)
    enter p into (s', s')
    enter Y into (s, s)
end
```

Wenn sich die Turing-Maschine bei Ausführung von $\delta(q, X) = (p, Y, R)$ auf ein neues Feld des Bandes bewegt, muss das Kommando so abgewandelt werden, dass die Zugriffsmatrix durch Hinzufügen eines neuen Subjektes vergrößert wird:

```
command D_qX(s, s')
  if
    end in (s, s) and
    q in (s, s) and
```

```
    X in (s, s)
  then
    delete q from (s, s)
    delete X from (s, s)
    create subject s'
    enter □ into (s', s')
    enter p into (s', s')
    enter Y into (s, s)
    enter end into (s', s')
    enter own into (s, s')
end
```

Bei Ausführung von command $C_{qX}$ bspw. würde Tabelle 3 zu Tabelle 4:

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|
| $o_1$ | $\{W, p\}$ | {own} | | |
| $o_2$ | | $\{Y\}$ | {own} | |
| $o_3$ | | | $\{Y\}$ | {own} |
| $o_4$ | | | | $\{Z$, end$\}$ |

Tabelle 4: Simulation Turing-Maschine durch Zugriffsmatrix 2

Also: Wenn es einen Algorithmus gäbe, der feststellen könnte, ob ein Recht $q_f$ unsicher ist, könnte man mit diesem Algorithmus feststellen, ob eine Turing-Maschine einen Endzustand $q_f$ erreicht. Dann gäbe es einen Algorithmus für das Halteproblem. Dies ist ein Widerspruch zur Unentscheidbarkait des Halteproblems. Also gibt es keinen Entscheidungsalgorithmus für das Safety-Problem.

q. e. d.

Für ein konkretes Schutzsystem muss das Safety-Problem aber nicht unbeantwortbar sein. Satz 1 sagt nur, dass es kein generelles Entscheidungsverfahren gibt.
Das Safety-Problem kann allerdings nicht für jedes Schutzsystem entschieden werden:

**Satz 2** *Es gibt nicht für jedes Sicherheitsmodell einen Algorithmus, der das Safety-Problem entscheidet.*

*Beweis:* Wenn es für jedes Sicherheitsmodell einen Algorithmus gäbe, der das Safety-Problem entscheiden würde, dann gäbe es auch einen Algorithmus für die Universelle Turing-Maschine. Dies ist ein Widerspruch, da das Halteproblem für Universelle Turing-Maschinen ebenfalls nicht entscheidbar ist.[4]

q. e. d.

## 2.2 Soll-Ist-Vergleich

Gegeben sei ein konkretes Schutzsystem mit einer festgelegten Menge von Kommandos und eine Zugriffsmatrix $P_t$. Die gewünschten Sicherheitseigenschaften des modellierten Systems seien informell (oder formell) gegeben, so dass zu untersuchen ist, ob das modellierte System (Ist-Eigenschaften) die gestellten Anforderungen (Soll-Eigenschaften) erfüllt.
Bspw. kann untersucht werden, ob ein Subjekt Rechte erhalten kann, die im Widerspruch zu den festgelegten Sicherheitsanforderungen stehen.

[4] [Sch01]

| | ··· | Konto_Bill | Drucke_Auszug |
|---|---|---|---|
| Joe | | | execute |
| Drucke_Auszug | | lese_Kontostand | |
| ⋮ | | | |

Tabelle 5: Soll-Ist-Vergleich

In dem Beispiel in Tabelle 5 hat das Subjekt Joe zwar nicht das explizite Recht lese_Kontostand an dem Objekt Konto_Bill, da Joe allerdings das Recht execute an Drucke_Auszug hat, und Druck_Auszug wiederum das Rechte lese_Kontostand an Konto_Bill hat, erlangt auch Joe das Recht lese_Kontostand an Konto_Bill. Die Soll-Eigenschaften stimmt nicht mit der Ist-Eigenschaften überein. Die Modellierung oder Rechtevergabe muss korrigiert werden.

Eine Lösung dieses Problems ist, für alle Subjekte die potentiellen Rechte zu ermitteln.

### 2.3 Modellierung von Zugriffsrechten

Die Modellierung von Zugriffsrechten hat entscheidenden Einfluss auf die Sicherheitseigenschaften des modellierten Systems. Werden Rechte grobkörning vergeben, können dadurch Sicherheitslücken bzw. Abweichungen von den Soll-Eigenschaften entstehen.
Zum Beispiel würden im Unix-Betriebssystem Sicherheitsprobleme durch eine zu grobkörnige Modellierung des Schreibrechts auf Verzeichnis auftreten:

Gegeben sei ein Directory $D$ mit einer Datei $f$:

1. Für $D$ gelte: Alle Benutzer erhalten Schreibrecht auf $D$.
2. Für $f$ gelte: Nur der Eigentümer darf $f$ modifizieren.

Also:

| | | | |
|---|---|---|---|
| Rechtevergabe für $D$: | r w x | - w x | - w - |
| Rechtevergabe für f: | r w - | r - - | - - - |

Aber:
Wegen w-Recht an $D$ ist ein Löschen bzw. Modifizieren von $f$ in $D$ für alle Benutzer möglich. Die Modellierung des write-Rechts ist also zu grob.

## 3 Bewertung und Klassifikation

### 3.1 Bewertung

Beim Zugriffsmatrix-Modell handelt es sich um ein sehr einfaches Sicherheitsmodell. Es ist sehr flexibel, da Subjekt und Objekte beliebig granular festgelegt und Zugriffsrechte universell und objektspezifisch modelliert werden können. Es ist einfach zu implementieren, gleichzeitig aber auch unübersichtlich in der Darstellung. Diese Unübersichtlichkeit führt dazu, dass es in den meisten Fällen nicht manuell zu beherrschen ist und man auf Werkzeugunterstützung angewiesen ist. Es können sehr komplexe Sicherheitseigenschaften beschrieben werden, deren Nachweis allerdings aufwendig oder gar unmöglich ist. Unter Umständen kann es sehr komplexe Wechselwirkungen geben. In einem verteilten System mit vielen sich ändernden Objekten und Subjekten ist eine Implementation sehr schwierig.

Im Zugriffsmatrix-Modell sind benutzerbestimmbare Sicherheitsstrategien über die Modellierung von Eigentümerrechten formulierbar, eine rollenbasierte Strategie kann nur rudimentär beschrieben werden. Informationsflüsse sind gar nicht modellierbar.

### 3.2 Klassifikation

Claudia Eckert hat in [Eck03] eine Klassifikationsschema für Sicherheitsmodelle entwickelt und in dieses die bekanntesten und gebräuchlichsten Modelle, unter anderem das Zugriffsmatrix-Modell, eingeordnet. Das Klassifikationsschema (Abbildung 1) soll als einfacher Leitfaden zur Wahl eines geeigneten Modells dienen.

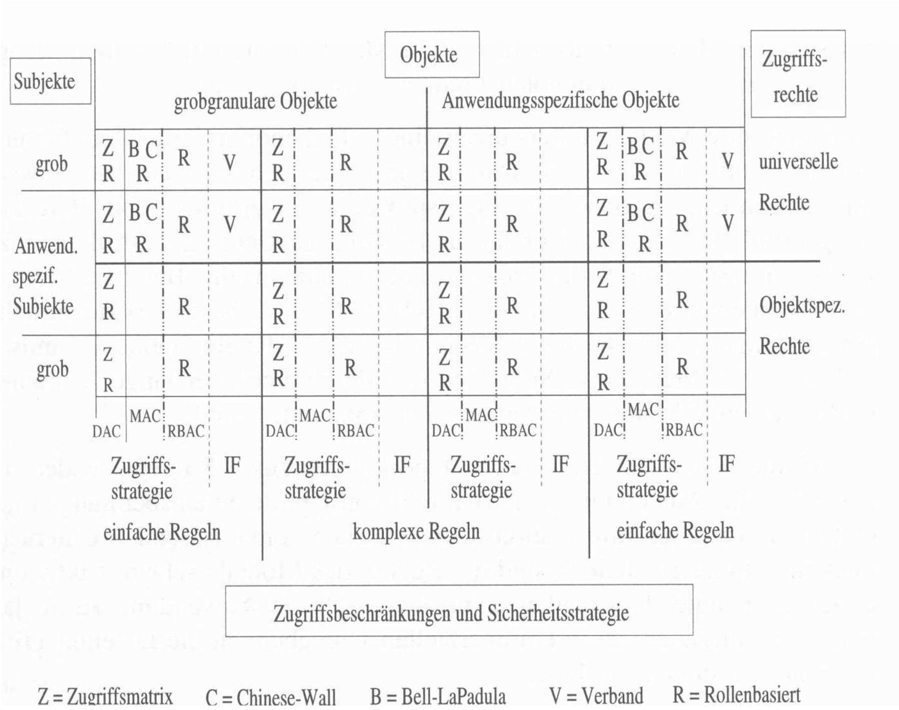


Abbildung 1: Klassifikation bekannter Sicherheitsmodelle (Quelle: [Eck03])

# A Turing-Maschine

**Definition 6 (Turing-Maschine)** *Eine Turing-Maschine ist gegeben durch ein 7-Tupel* $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$*. Hierbei sind:*

- $Z$ *die endliche Zustandsmenge,*
- $\Sigma$ *das Eingabealphabet,*
- $\Gamma \supset \Sigma$ *das Arbeitsalphabet,*
- $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$ *im deterministischen Fall (bzw.* $\delta : Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$ *im nichtdeterministischen Fall) die Überführungsfunktion,*
- $z_0 \in Z$ *der Startzustand,*

- $\Box \in \Gamma - \Sigma$ *das Blank-Zeichen,*
- $E \subseteq Z$ *die Menge der Endzustände.*

# B Halteproblem

**Definition 7 (Halteproblem)** *Das (allgemeine) Halteproblem ist die Sprache*

$$H = \{w\#x|\, M_w \,\text{angesetzt auf}\, x\, \text{hält}\}.$$

**Satz 3** *Das Halteproblem H ist nicht entscheidbar.*

Den Beweis dieses Satzes findet man in [Sch01].

# Literatur


[Eck03] Claudia Eckert. *IT-Sicherheit : Konzepte – Verfahren – Protokolle.* Oldenbourg Wissenschaftsverlag GmbH, München, Wien, 2. edition, 2003.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, pages 461–471, 1976.

[Mar01] Michael T. Marrotte. Is the harrison, ruzzo, ullman undecidable proof flawed?, 2001.

[San92] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–136, 1992.

[Sch01] Uwe Schöning. *Theoretische Informatik – kurzgefasst.* Spektrum, Akademischer Verlag, Heidelberg, Berlin, 4. edition, 2001.