

**Logik für Informatiker**  
**Prädikatenlogik: Anwendungen**  
**Erweiterungen**

19 und 21.07.2016

Viorica Sofronie-Stokkermans  
Universität Koblenz-Landau  
e-mail: [sofronie@uni-koblenz.de](mailto:sofronie@uni-koblenz.de)

# Bis jetzt

---

## Aussagenlogik

- Syntax, Semantik
- Kalküle (Resolution, Tableaux)
- Anwendungen

## Prädikatenlogik

- Syntax, Semantik
- Kalküle (Resolution, Tableaux)

# Bis jetzt

---

## Aussagenlogik

- Syntax, Semantik
- Kalküle (Resolution, Tableaux)
- Anwendungen

## Prädikatenlogik

- Syntax, Semantik
- Kalküle (Resolution, Tableaux)

**Heute:** Anwendungen und Erweiterungen

## Implementierungen

# Implementierung

---

## Aussagenlogik

Wahrheitstabellen, KNF, DNF

Erfüllbarkeit

# Implementierung

---

**Erfüllbarkeit, Aussagenlogik**

Prolog?

# Implementierung

---

## Erfüllbarkeit, Aussagenlogik

Prolog?

```
sat(true).
```

```
sat(not(false)).
```

```
sat(or([X|L])):- sat(X); sat(or(L)).
```

```
sat(and([])).
```

```
sat(and([X|L])):-sat(X), sat(and(L)).
```

# Aussagenlogik

---

## **Beweise durch Resolution:**

SPASS

Vampire

E

Theorembeweiser für Prädikatenlogik

# Aussagenlogik

---

## Tableaux

z.B. KRHyper

Theoremenbeweiser für Prädikatenlogik



# Implementierungen

---

zchaff

(Davis-Putnam Verfahren)

<https://www.princeton.edu/~chaff/software.html>

**SAT competition and benchmarks:**

<http://www.satcompetition.org>

Grosse Benchmarks, z.B. slp-synthesis-aes-top27.cnf

# Theoremenbeweiser: Prädikatenlogik

---

SPASS:

<http://www.spass-prover.org>

The web interface

<http://www.spass-prover.org/webpass/index.html>

provides a nice first-order example. If you add

```
list_of_settings(SPASS).  
  {*  
  set_flag(DocProof,1).  
  *}  
end_of_list.
```

before the end of the document, SPASS outputs a proof where you can see the intermediate inference steps.

# Theoremenbeweiser

---

Vampire:

<http://www.vprover.org/>

E:

<http://www.e prover.org/>

# Anwendungen

---

# Logical puzzles

---



## Beispiel: Tante Agatha

---

Jemand, der in Schloss Dreadbury wohnt, hat Tante Agatha ermordet. Agatha, ihr Butler und ihr Neffe Charles waren die einzigen Bewohner von Schloss Dreadbury. Ein Mörder hasst immer sein Opfer und ist niemals reicher als sein Opfer. Charles hasst niemanden, den Tante Agatha gehasst hat. Agatha hat jeden gehasst außer ihrem Butler. Der Butler hasst jeden, der nicht reicher ist als Tante Agatha. Der Butler hasst jeden, den Tante Agatha gehasst hat. Niemand hasst jeden. Agatha war nicht der Butler.

Wer hat Tante Agatha ermordet?

# Formalisieren

---

Jemand, der in Schloss Dreadbury wohnt, hat Tante Agatha ermordet.

- ▶  $\exists x (\text{schlossbewohner}(x) \wedge \text{ermordet}(x, a))$

Agatha, ihr Butler und ihr Neffe Charles waren die einzigen Bewohner von Schloss Dreadbury.

- ▶  $\forall x (\text{schlossbewohner}(x) \leftrightarrow (x \approx a \vee x \approx b \vee x \approx c))$

# Formalisieren

---

Ein Mörder hasst immer sein Opfer und ist niemals reicher als sein Opfer.

- ▶  $\forall x, y (\text{ermordet}(x, y) \rightarrow \text{hasst}(x, y))$   
 $\forall x, y (\text{ermordet}(x, y) \rightarrow \neg \text{reicher}(x, y))$

Charles hasst niemanden, den Tante Agatha gehasst hat.

- ▶  $\forall x (\text{hasst}(c, x) \rightarrow \neg \text{hasst}(a, x))$

Agatha hat jeden gehasst außer ihrem Butler.

- ▶  $\forall x (\neg \text{hasst}(a, x) \leftrightarrow x \approx b)$



# Formalisieren

---

Der Butler hasst jeden, der nicht reicher ist als Tante Agatha.

▶  $\forall x (\neg \text{reicher}(x, a) \rightarrow \text{hasst}(b, x))$

Der Butler hasst jeden, den Tante Agatha gehasst hat.

▶  $\forall x (\text{hasst}(a, x) \rightarrow \text{hasst}(b, x))$

Niemand hasst jeden.

▶  $\forall x \exists y (\neg \text{hasst}(x, y))$

Agatha war nicht der Butler.

▶  $\neg a \approx b$

# Klauselmenge

---

$\exists x (\text{schlossbewohner}(x) \wedge \text{ermordet}(x, a))$

$\mapsto$

$(\text{schlossbewohner}(sk) \wedge \text{ermordet}(sk, a))$

# Klauselmenge

---

$\exists x (\text{schlossbewohner}(x) \wedge \text{ermordet}(x, a))$

$\mapsto$

$(\text{schlossbewohner}(sk) \wedge \text{ermordet}(sk, a))$

Since we know that  $(\text{schlossbewohner}(x) \leftrightarrow (x \approx a) \vee (x \approx b) \vee (x \approx c))$

we use instead

$\text{ermordet}(a, a) \vee \text{ermordet}(b, a) \vee \text{ermordet}(c, a)$

# Klauselmenge

---

$\forall x \exists y (\neg \text{hasst}(x, y))$

$\mapsto$

$\neg \text{hasst}(x, sk1(x))$  Not what is meant.

$\neg \text{hasst}(x, sk1(x)) \wedge \text{schlossbewohner}(sk1(x))$

# Klauselmenge

---

$\forall x \exists y (\neg \text{hasst}(x, y))$

$\mapsto$

$\neg \text{hasst}(x, \text{sk1}(x))$  Not what is meant.

$\neg \text{hasst}(x, \text{sk1}(x)) \wedge \text{schlossbewohner}(\text{sk1}(x))$

Since we know that  $(\text{schlossbewohner}(x) \leftrightarrow (x \approx a) \vee (x \approx b) \vee (x \approx c))$

we use instead

$\neg \text{hasst}(x, a) \vee \neg \text{hasst}(x, b) \vee \neg \text{hasst}(x, c)$

# Klauselmenge

---

1.  $\text{ermordet}(a, a) \vee \text{ermordet}(b, a) \vee \text{ermordet}(c, a)$
2.  $\neg \text{schlossbewohner}(x) \vee (x \approx a \vee x \approx b \vee x \approx c)$
3.  $\text{schlossbewohner}(a)$
4.  $\text{schlossbewohner}(b)$
5.  $\text{schlossbewohner}(c)$
6.  $\neg \text{ermordet}(x, y) \vee \text{hasst}(x, y)$
7.  $\neg \text{ermordet}(x, y) \vee \neg \text{reicher}(x, y)$
8.  $\neg \text{hasst}(c, x) \vee \neg \text{hasst}(a, x)$
9.  $\text{hasst}(a, x) \vee x \approx b$
10.  $x \approx b \vee \neg \text{hasst}(a, x)$
11.  $\text{reicher}(x, a) \vee \text{hasst}(b, x)$
12.  $\neg \text{hasst}(x, a) \vee \neg \text{hasst}(x, b) \vee \neg \text{hasst}(x, c)$
13.  $\neg a \approx b$
14.  $\neg c \approx b$

# Klauselmenge

---

1.  $\text{ermordet}(a, a) \vee \text{ermordet}(b, a) \vee \text{ermordet}(c, a)$
2.  $\neg \text{schlossbewohner}(x) \vee (x \approx a \vee x \approx b \vee x \approx c)$
3.  $\text{schlossbewohner}(a)$
4.  $\text{schlossbewohner}(b)$
5.  $\text{schlossbewohner}(c)$
6.  $\neg \text{ermordet}(x, y) \vee \text{hasst}(x, y)$
7.  $\neg \text{ermordet}(x, y) \vee \neg \text{reicher}(x, y)$
8.  $\neg \text{hasst}(c, x) \vee \neg \text{hasst}(a, x)$
9.  $\text{hasst}(a, x) \vee x \approx b$
10.  $x \approx b \vee \neg \text{hasst}(a, x)$
11.  $\text{reicher}(x, a) \vee \text{hasst}(b, x)$
12.  $\neg \text{hasst}(x, a) \vee \neg \text{hasst}(x, b) \vee \neg \text{hasst}(x, c)$
13.  $\neg a \approx b$
14.  $\neg c \approx b$
15.  $\neg \text{ermordet}(c, y) \vee \neg \text{hasst}(a, y)$  [Res. 6, 8]
16.  $\neg \text{ermordet}(c, y) \vee y \approx b$  [Res. 15, 9]
17.  $\neg \text{ermordet}(c, a)$  [Res. 16, 13]
18.  $\text{hasst}(a, a)$  [Res. 13, 9]
19.  $\text{hasst}(a, c)$  [Res. 14, 9]
20.  $\neg \text{hasst}(c, a)$  [Res. 19, 8]
21.  $\text{hasst}(b, a)$  [Res. 11, 18]
22.  $\text{hasst}(b, c)$  [Res. 11, 19]
23.  $\neg \text{hasst}(b, b)$  [Res. 21, 22, 12]
24.  $\text{reicher}(b, a)$  [Res. 23, 11]
25.  $\neg \text{ermordet}(b, a)$  [Res. 24, 7]
26.  $\text{ermordet}(a, a)$  [Res. 25, 17, 1]

# Ähnliche Probleme

---



## Programmverifikation

```
int [] BUBBLESORT(int[] a) {  
    int i, j, t;  
    for (i := |a| - 1; i > 0; i := i - 1) {  
        for (j := 0; j < i; j := j + 1) {  
            if (a[j] > a[j + 1]) { t := a[j];  
                a[j] := a[j + 1];  
                a[j + 1] := t; }  
        }  
    } return a}
```



# Ähnliche Probleme

---



## Sicherheitsprotokolle



# Ähnliche Probleme

---



## Mathematik



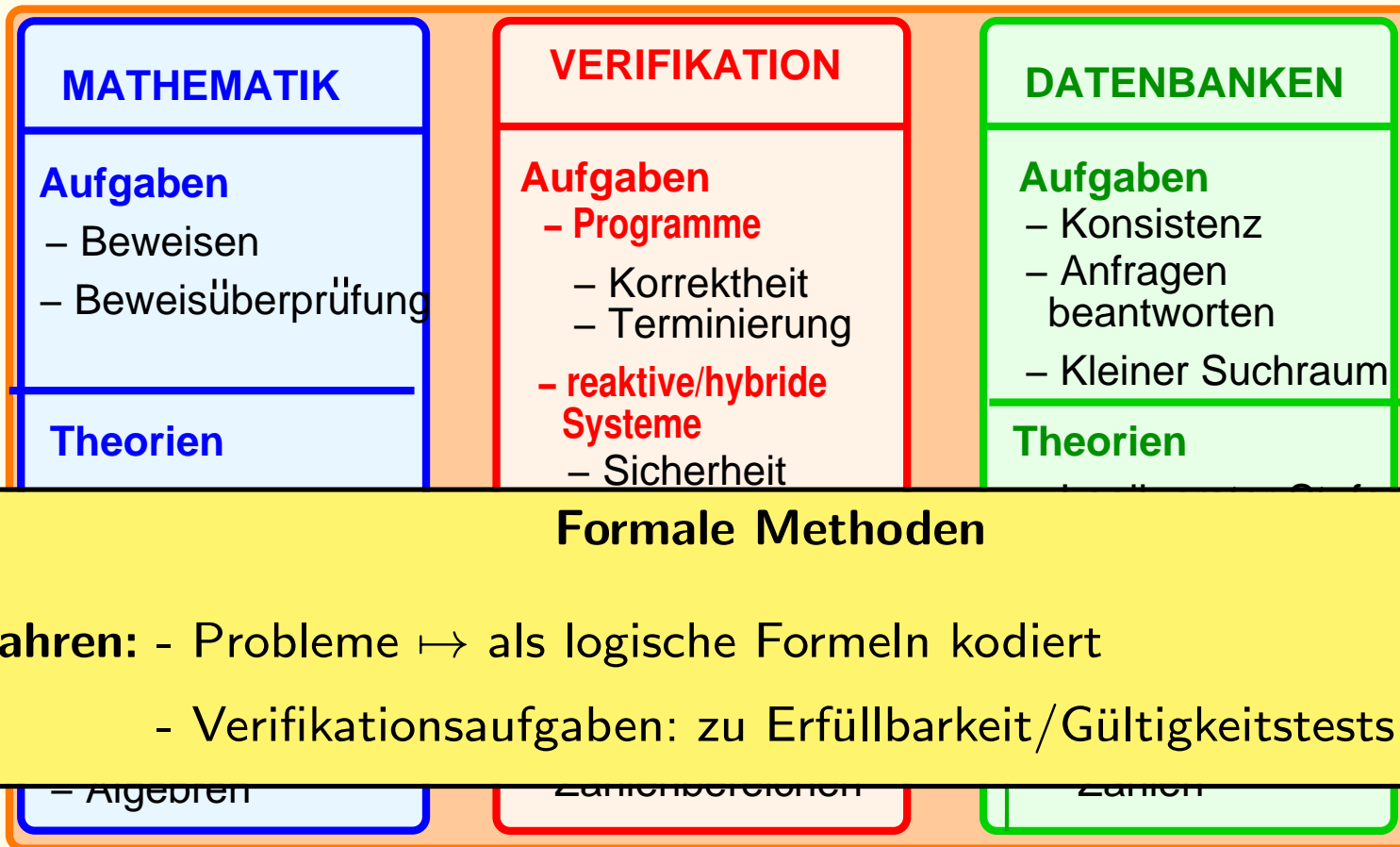
# Anwendungsbereiche

---

MATHEMATIK	VERIFIKATION	DATENBANKEN
<b>Aufgaben</b> <ul style="list-style-type: none"><li>– Beweisen</li><li>– Beweisüberprüfung</li></ul>	<b>Aufgaben</b> <ul style="list-style-type: none"><li>– <b>Programme</b><ul style="list-style-type: none"><li>– Korrektheit</li><li>– Terminierung</li></ul></li><li>– <b>reaktive/hybride Systeme</b><ul style="list-style-type: none"><li>– Sicherheit</li></ul></li></ul>	<b>Aufgaben</b> <ul style="list-style-type: none"><li>– Konsistenz</li><li>– Anfragen beantworten</li><li>– Kleiner Suchraum</li></ul>
<b>Theorien</b> <ul style="list-style-type: none"><li>– Zahlen</li><li>– Polynomen</li><li>– Funktionen auf Zahlenbereichen</li><li>– Algebren</li></ul>	<b>Theorien</b> <ul style="list-style-type: none"><li>– Zahlen</li><li>– Datentypen</li><li>– Funktionen auf Zahlenbereichen</li></ul>	<b>Theorien</b> <ul style="list-style-type: none"><li>– Logik erster Stufe</li><li>– Datalog</li><li>– ...</li><li>– Komplexe Theorien<ul style="list-style-type: none"><li>– Funktionen</li><li>– Zahlen</li></ul></li></ul>

**komplexe Systeme** (MAS, mit eingebetteter Software, Datenbanken)

# Anwendungsbereiche



**komplexe Systeme** (MAS, mit eingebetteter Software, Datenbanken)

# Anwendungsbereiche

## MATHEMATIK

### Aufgaben

- Beweisen
- Beweisüberprüfung

### Theorien

- Zahlen
- Polynomen
- Funktionen auf Zahlenbereichen
- Algebren

## VERIFIKATION

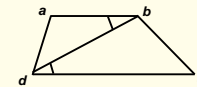
### Aufgaben

- **Programme**
  - Korrektheit
  - Terminierung
- **reaktive/hybride Systeme**
  - Sicherheit

### Theorien

- Zahlen
- Datentypen
- Funktionen auf Zahlenbereichen

### Beispiel: Geometrie



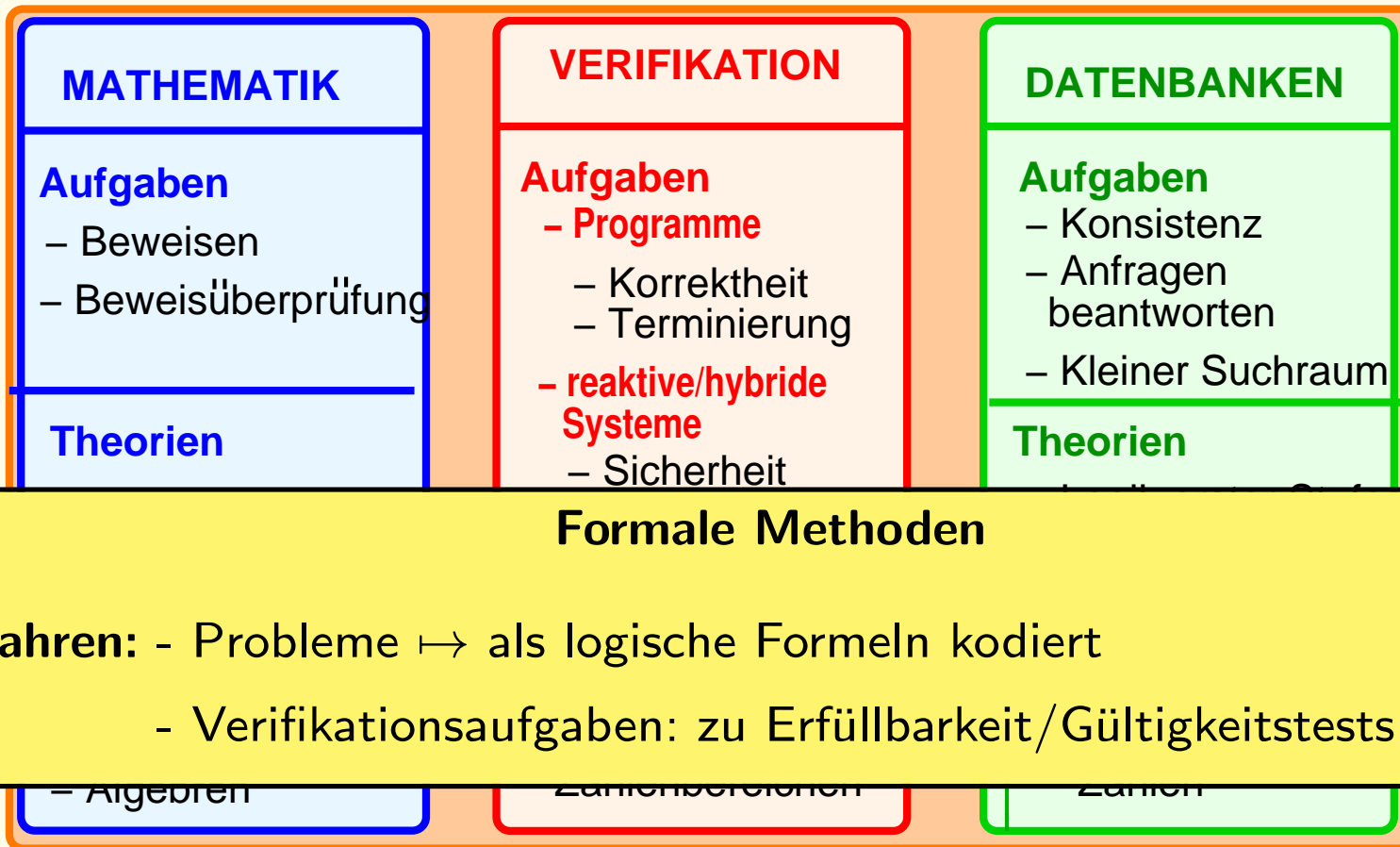
$$(A1) \forall x \forall y \forall u \forall v (T(x, y, u, v) \rightarrow P(x, y, u, v))$$

$$(A2) \forall x \forall y \forall u \forall v (P(x, y, u, v) \rightarrow E(x, y, v, u, v, y))$$

$$(A3) T(a, b, c, d)$$

$$A1 \wedge A2 \wedge A3 \rightarrow E(a, b, d, c, d, b)$$

# Anwendungsbereiche



**komplexe Systeme** (MAS, mit eingebetteter Software, Datenbanken)

# Beispiel: Sicherheitsprotokole

---

**Ziel:** zwei Personen (Alice und Bob) wollen miteinander kommunizieren

- über ein **unsicheres** Daten- oder Telefonnetz,
- **sicher**, d. h., ohne daß ein Eindringling (Charlie) mithören oder sich als Alice oder Bob ausgeben kann.

# Beispiel: Sicherheitsprotokole

---

**Ziel:** zwei Personen (Alice und Bob) wollen miteinander kommunizieren

- über ein **unsicheres** Daten- oder Telefonnetz,
- **sicher**, d. h., ohne dass ein Eindringling (Charlie) mithören oder sich als Alice oder Bob ausgeben kann.

**Hilfsmittel:** Verschlüsselung

- Alice und Bob vereinbaren einen gemeinsamen Schlüssel und nutzen ihn, um ihr Gespräch zu verschlüsseln.
- Nur wer den Schlüssel kennt, kann das Gespräch entschlüsseln.



# Beispiel: Sicherheitsprotokolle

---

Problem: wie kommen die Gesprächspartner an den gemeinsamen Schlüssel?

- Persönliche Übergabe kommt nicht immer in Frage.
- Wird der gemeinsame Schlüssel über das Netz unverschlüsselt verschickt, könnte Charlie ihn abfangen oder austauschen.
- Annahme: es gibt eine sichere Schlüsselzentrale, mit der Alice und Bob jeweils einen gemeinsamen Schlüssel vereinbart haben.

# Beispiel: Sicherheitsprotokolle

---

Das folgende Schlüsselaustauschverfahren wurde 1993 von den beiden Kryptographen Neuman und Stubblebine vorgeschlagen:

## Schritt 1:



Alice schickt (offen) Identifikation und Zufallszahl an Bob.

# Beispiel: Sicherheitsprotokolle

---

Das folgende Schlüsselaustauschverfahren wurde 1993 von den beiden Kryptographen Neuman und Stubblebine vorgeschlagen:

## Schritt 1:



Alice schickt Bob die Nachricht 1:  $A, N_a$ .

Die Nachricht enthält den Namen von Alice,  $A$ , und eine Zufallszahl  $N_a$ .

Die Aufgabe von  $N_a$  liegt darin, diesen Protokollauf eindeutig zu kennzeichnen, um das Wiederholen von Nachrichten durch einen Angreifer sinnlos werden zu lassen.

Alice schickt (offen) Identifikation und Zufallszahl an Bob.

# Beispiel: Sicherheitsprotokolle

---

## Schritt 2:



Bob leitet Nachricht weiter an Schlüsselzentrale („Trust“).

# Beispiel: Sicherheitsprotokolle

---

Bob schickt die Nachricht 2 :  $B, E(K_{bt}, A, N_a, T_b), N_b$  an Trust

Nachdem Bob die Nachricht von Alice erhalten hat, weiss er, dass sie einen sicheren Schlüssel mit ihm etablieren will.

Deshalb schickt Bob die Nachricht 2 an Trust.

## Schritt 2:



Die Nachricht enthält:

- seinen Namen,  $B$ ,
- einen verschlüsselten Mittelteil
- eine von Bob generierte Zufallszahl,  $N_b$ ,
- $T_b$ : Zeitspanne für die Gültigkeitsdauer des Schlüssels

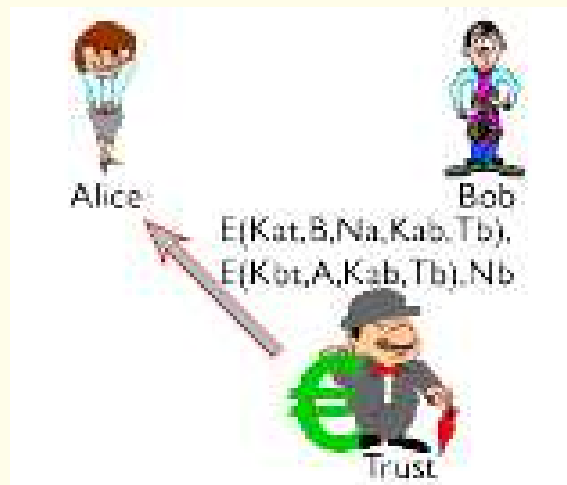
Der Term  $E(K_{bt}, A, N_a, T_b)$  steht für die Nachricht  $A, N_a, T_b$  verschlüsselt mit dem Schlüssel  $K_{bt}$ , dem sicheren Schlüssel zwischen Bob und Trust.

Bob leitet Nachricht weiter an Schlüsselzentrale („Trust“).

# Beispiel: Sicherheitsprotokolle

---

## Schritt 3:



Trust schickt Nachricht an Alice. Darin: ein neuer gemeinsamer Schlüssel, einmal für Alice und einmal für Bob verschlüsselt.

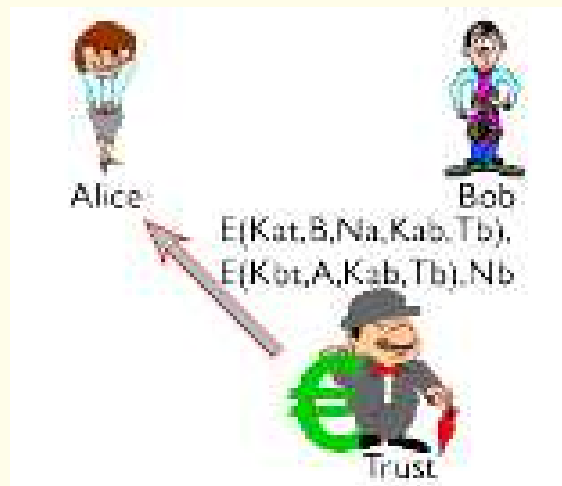
# Beispiel: Sicherheitsprotokolle

---

Trust schickt die Nachricht 3

$E(K_{at}, B, Na, K_{ab}, T_b), E(K_{bt}, A, K_{ab}, T_b), N_b$  an Alice.

## Schritt 3:



Nachdem Trust Bob's Nachricht gelesen und entschlüsselt hat, generiert er den sicheren Schlüssel  $K_{ab}$  für die Kommunikation zwischen Alice und Bob.

Er verschickt den Schlüssel mit Nachricht 3 an Alice.

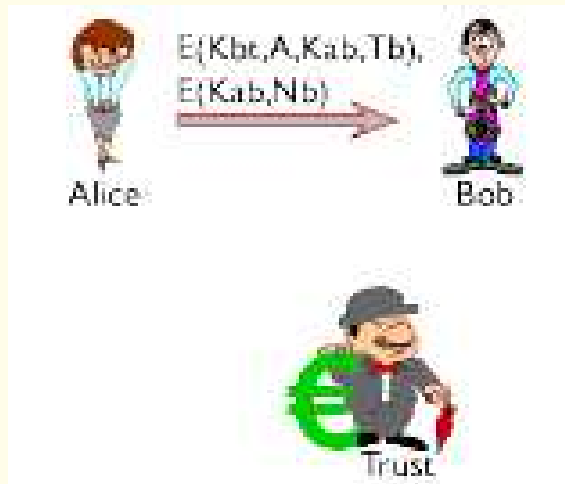
Der erste Teil der Nachricht kann von Alice entschlüsselt werden während sie den zweiten Teil später einfach an Bob weiterleitet, siehe Schritt 4.

Trust schickt Nachricht an Alice. Darin: ein neuer gemeinsamer Schlüssel, einmal für Alice und einmal für Bob verschlüsselt.

# Beispiel: Sicherheitsprotokolle

---

## Schritt 4:



Alice leitet den neuen gemeinsamen Schlüssel weiter an Bob.



# Beispiel: Sicherheitsprotokolle

---

Alice schickt die Nachricht 4:

$E(K_{bt}, A, K_{ab}, T_b), E(K_{ab}, N_b)$  an Bob.

## Schritt 4:



Alice liest Nachricht 3, entschlüsselt den ersten Teil durch den gemeinsamen Schlüssel  $K_{at}$  mit Trust und erhält so den neuen Schlüssel  $K_{ab}$ , um mit Bob zu kommunizieren.

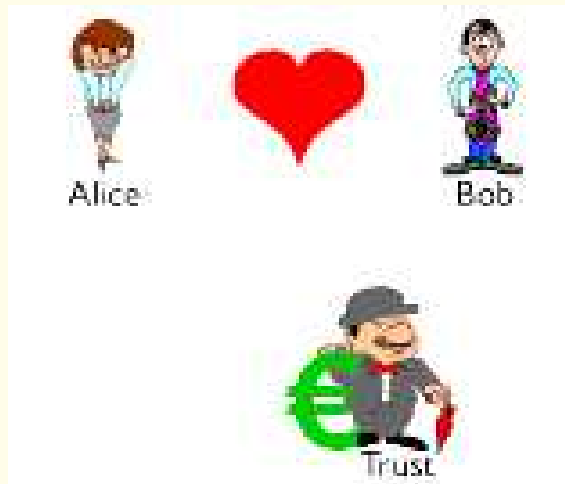
Sie leitet den zweiten Teil von Trust's Nachricht an Bob weiter und fügt die Nachricht  $E(K_{ab}, N_b)$  hinzu.

Alice leitet den neuen gemeinsamen Schlüssel weiter an Bob.

# Beispiel: Sicherheitsprotokolle

---

## Schritt 5:



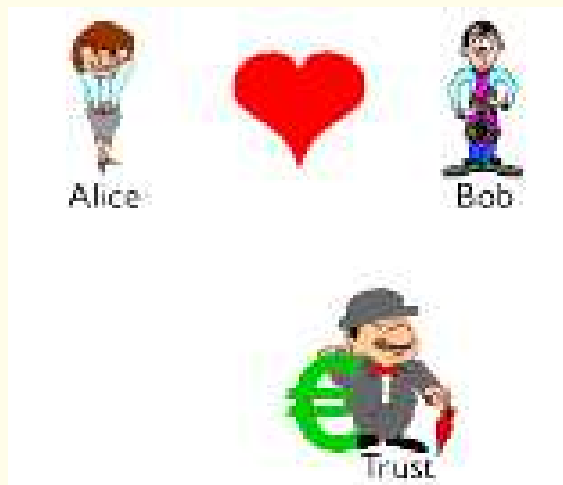
Alice und Bob können nun mit dem gemeinsamen Schlüssel kommunizieren.

# Beispiel: Sicherheitsprotokolle

---

Bob liest Nachricht 4, entschlüsselt den ersten Teil und erhält so auch den Schlüssel  $K_{ab}$ .

## Schritt 5:



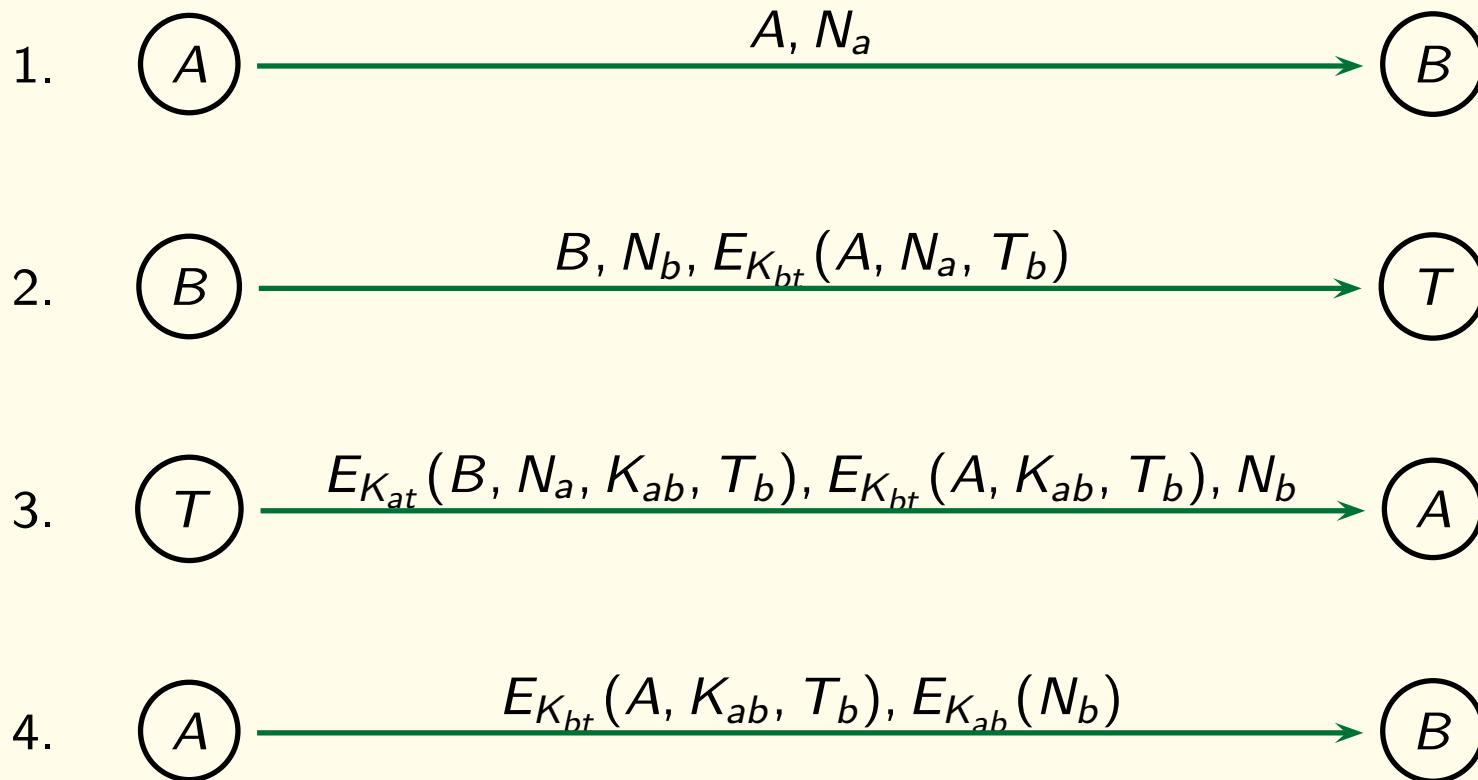
Diesen verwendet er dann zur Entschlüsselung des zweiten Teils und vergleicht die enthaltene Zufallszahl  $N_b$  mit der, die er ursprünglich in Nachricht 2 an Trust schickte.

Auf diese Weise ist sich Bob sicher, dass die Nachricht von Alice stammt und  $K_{ab}$  der gewünschte Schlüssel ist.

Alice und Bob können nun mit dem gemeinsamen Schlüssel kommunizieren.

# Beispiel: Sicherheitsprotokolle

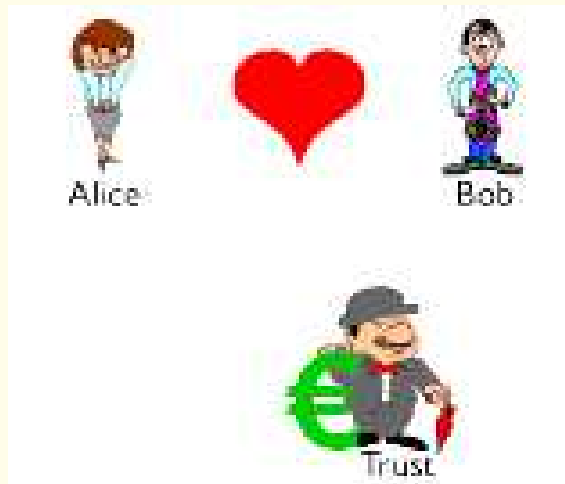
---



# Beispiel: Sicherheitsprotokolle

---

Ist das Verfahren sicher?



Wir übersetzen das Problem in Formeln und lassen sie von einem Theorembeweiser untersuchen.

# Beispiel: Sicherheitsprotokolle

---

Zuerst formalisieren wir die Eigenschaften des Protokolls:

- Wenn Alice/Bob/Trust eine Nachricht in einem bestimmten Format bekommt, dann schickt er/sie eine andere Nachricht ab.

# Beispiel: Sicherheitsprotokolle

---

Zuerst formalisieren wir die Eigenschaften des Protokolls:

- Wenn Alice/Bob/Trust eine Nachricht in einem bestimmten Format bekommt, dann schickt er/sie eine andere Nachricht ab.

Dann formalisieren wir die Eigenschaften des Angreifers:

- Wenn eine Nachricht übermittelt wird, kann Charlie sie mithören.
- Wenn Charlie eine verschlüsselte Nachricht bekommt und den passenden Schlüssel hat, kann er sie entschlüsseln.
- Wenn Charlie eine Nachricht hat, dann kann er sie an Alice/Bob/Trust abschicken.

...

# Formalisierung der Eigenschaften des Protokolls

---

1.  $A, Na$

(1)  $Ak(key(at, t))$

(2)  $M(sent(a, b, pair(a, na)))$

**Formel (1)** beschreibt, dass Alice am Anfang den Schlüssel  $at$  für die Kommunikation mit Trust besitzt. (Wir lassen “ $K$ ” bei der Benennung von Schlüsseln weg.)

**Formel (2)** formalisiert Nachricht 1.

Nachrichten werden durch die dreistellige Funktion “sent” ausgedrückt, wobei das erste Argument den Absender bezeichnet, das zweite den Empfänger und das dritte den Inhalt der Nachricht.

Die Konstante  $a$  bezeichnet Alice,  $b$  Bob,  $t$  Trust und  $i$  Intruder.

Die Funktionen “pair” (triple, quadr) gruppieren Sequenzen von Nachrichten entsprechender Länge.



# Formalisierung der Eigenschaften des Protokolls

---

2.  $B, Nb, E(Kbt, A, Na, Tb)$

(3)  $Bk(key(bt, t))$

(4)  $\forall xa, xna[M(sent(xa, b, pair(xa, xna))) \rightarrow M(sent(b, t, triple(b, nb(xna), encr(triple(xa, xna, tb(xna)), bt)))))]$

**Formel (3):** Bob besitzt den Schlüssel  $bt$  zur Kommunikation mit Trust.

**Formel (4):** Wann immer er eine Nachricht in der Form von Nachricht 1 erhält (Formel 2), schickt er die Anfrage nach einem Schlüssel an Trust wie in Nachricht 2 festgelegt.

Die Verschlüsselung wird mit Hilfe der zweistelligen Funktion “encr” beschrieben.

Das erste Argument der Funktion ist die zu verrschlüsselnde Nachricht und das zweite Argument der benutzte Schlüssel.

Alle Symbole die mit einem kleinen  $x$  anfangen, bezeichnen Variablen.

Die Funktionen  $nb$  und  $tb$  generieren Bob's Zufallszahl und Ablaufdatum aus  $xna$ ,  $xa$ 's (Alice's ) Zufallszahl.

# Formalisierung der Eigenschaften des Protokolls

---

3.  $E(K_{at}, B, Na, K_{ab}, T_b), E(K_{bt}, A, K_{ab}, T_b), Nb$

(5)  $Tk(key(at, a)) \wedge Tk(key(bt, b))$

(6)  $\forall xb, xnb, xa, xna, xbet, xbt, xat, xk$   
 $[(M(sent(xb, t, triple(xb, xnb, encr(triple(xa, xna, xbet), xbt)))))) \wedge$   
 $Tk(key(xbt, xb)) \wedge Tk(key(xat, xa))] \rightarrow$   
 $M(sent(t, xa, triple(encr(quad(xb, xna, kt(xna), xbet), xat),$   
 $encr(triple(xa, kt(xna), xbet), xbt),$   
 $xnb))))]$

(5) Trust besitzt die Schlüssel für Alice und Bob und ...

(6) ... antwortet entsprechend des Protokolls auf Nachricht 2.

Entschlüsselung wird hierbei durch Unifikation mit einer entsprechenden Termstruktur realisiert, wobei zusätzlich überprüft wird, dass Trust die benötigten Schlüssel auch besitzt.

Trust generiert den neuen Schlüssel durch die Funktion  $kt$  aus der Zufallszahl  $xna$ .

# Formalisierung der Eigenschaften des Protokolls

---

## 4. $E(Kbt, A, Kab, Tb), E(Kab, Nb)$

(7)  $\forall xnb, xbet, xk, xm, xb, xna$

$[M(\text{sent}(t, a, \text{triple}(\text{encr}(\text{quadr}(xb, xna, xk, xbet), at), xm, xnb)) \rightarrow$   
 $(M(\text{sent}(a, xb, \text{pair}(xm, \text{encr}(xnb, xk)))) \wedge Ak(\text{key}(xk, xb)))]$

(8)  $\forall xbet, xk, xnb, xa, xna$

$[M(\text{sent}(xa, b, \text{pair}(\text{encr}(\text{triple}(xa, xk, tb(xna)), bt), \text{encr}(nb(xna), xk))) \rightarrow$   
 $Bk(\text{key}(xk, xa))]$

**Formel (7):** Alice antwortet in korrekter Weise auf Nachricht 3 und merkt sich den neuen Schlüssel.

**Formel (8)** beschreibt Bob's Verhalten nachdem er Alices Nachricht erhalten hat. Bob entschlüsselt die Nachricht und extrahiert den neuen Schlüssel.

# Formalisierung des Angreifers

---

Der Angreifer wird als unermüdlicher Hacker implementiert, der keine Möglichkeit unversucht lässt, das Protokoll anzugreifen.

Er hört alle Nachrichten ab und zerlegt sie in ihre Bestandteile, solange er dazu nicht einen Schlüssel braucht, den er nicht hat.

Die Bestandteile werden dann auf jeder Art und Weise zu neuen Nachrichten beliebiger Länge komponiert.

Jeder Bestandteil wird als Schlüssel in Betracht gezogen und sowohl zum Verschlüsseln als auch zum Entschlüsseln verwendet.

Alle so generierten Nachrichten werden verschickt.

Die Menge aller Nachrichten(teile), die der Intruder so erhält, wird durch das Prädikat "*Im*" ausgedrückt.

# Der Angreifer

---

- Die Teilnehmer (Participants) sind Alice, Bob, Trust und Intruder:

$$(9) P(a) \wedge P(b) \wedge P(t) \wedge P(i)$$

# Der Angreifer

---

- Die Teilnehmer (Participants) sind Alice, Bob, Trust und Intruder:

$$(9) P(a) \wedge P(b) \wedge P(t) \wedge P(i)$$

- Intruder hört alle Nachrichten ab.

$$(10) \forall xa, xb, xm[M(sent(xa, xb, xm)) \rightarrow Im(xm)]$$

# Der Angreifer

---

- Die Teilnehmer (Participants) sind Alice, Bob, Trust und Intruder:

$$(9) P(a) \wedge P(b) \wedge P(t) \wedge P(i)$$

- Intruder hört alle Nachrichten ab.

$$(10) \forall xa, xb, xm[M(sent(xa, xb, xm)) \rightarrow Im(xm)]$$

- Er dekomponiert und entschlüsselt alle Nachrichten, soweit er dafür den Schlüssel besitzt.

$$(11) \forall u, v[Im(pair(u, v)) \rightarrow (Im(u) \wedge Im(v))]$$

# Der Angreifer

---

- Die Teilnehmer (Participants) sind Alice, Bob, Trust und Intruder:

$$(9) P(a) \wedge P(b) \wedge P(t) \wedge P(i)$$

- Intruder hört alle Nachrichten ab.

$$(10) \forall xa, xb, xm[M(sent(xa, xb, xm)) \rightarrow Im(xm)]$$

- Er dekomponiert und entschlüsselt alle Nachrichten, soweit er dafür den Schlüssel besitzt.

$$(11) \forall u, v[Im(pair(u, v)) \rightarrow (Im(u) \wedge Im(v))]$$

$$(12) \forall u, v, w[Im(triple(u, v, w)) \rightarrow (Im(u) \wedge Im(v) \wedge Im(w))]$$

$$(13) \forall u, v, w, z[Im(quadr(u, v, w, z)) \rightarrow (Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z))]$$

$$(14) \forall u, v, w[(Im(incr(u, v)) \wedge Ik(key(v, w))) \rightarrow Im(u)]$$



# Der Angreifer

---

- Alle Nachrichten(teile) werden auf alle möglichen Arten neu komponiert.

$$(15) \forall u, v [(Im(u) \wedge Im(v) \rightarrow Im(pair(u, v)))]$$

$$(16) \forall u, v, w [(Im(u) \wedge Im(v) \wedge Im(w)) \rightarrow Im(triple(u, v, w))]$$

$$(17) \forall u, v, w, z [(Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)) \rightarrow Im(quadr(u, v, w, z))]$$

# Der Angreifer

---

- Alle Nachrichten(teile) werden auf alle möglichen Arten neu komponiert.

$$(15) \forall u, v [(Im(u) \wedge Im(v) \rightarrow Im(pair(u, v)))]$$

$$(16) \forall u, v, w [(Im(u) \wedge Im(v) \wedge Im(w)) \rightarrow Im(triple(u, v, w))]$$

$$(17) \forall u, v, w, z [(Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)) \rightarrow Im(quadr(u, v, w, z))]$$

- Jeder Nachrichtenteil wird auch als Schlüssel aufgefasst und zur Verschlüsselung benutzt.

$$(18) \forall u, v, w [(Im(v), P(w) \rightarrow Ik(key(v, w)))]$$

$$(19) \forall u, v, w [(Im(v), P(w), Ik(key(v, w))) \rightarrow Im(encr(u, v))]$$

# Der Angreifer

---

- Alle Nachrichten(teile) werden auf alle möglichen Arten neu komponiert.

$$(15) \forall u, v [(Im(u) \wedge Im(v) \rightarrow Im(pair(u, v)))]$$

$$(16) \forall u, v, w [(Im(u) \wedge Im(v) \wedge Im(w)) \rightarrow Im(triple(u, v, w))]$$

$$(17) \forall u, v, w, z [(Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)) \rightarrow Im(quadr(u, v, w, z))]$$

- Jeder Nachrichtenteil wird auch als Schlüssel aufgefasst und zur Verschlüsselung benutzt.

$$(18) \forall u, v, w [(Im(v), P(w) \rightarrow Ik(key(v, w)))]$$

$$(19) \forall u, v, w [(Im(v), P(w), Ik(key(v, w))) \rightarrow Im(encr(u, v))]$$

- Der Angreifer schickt alles was er hat an jeden.

$$(20) \forall x, y, u [(P(x) \wedge P(y) \wedge Im(u)) \rightarrow M(sent(x, y, u))]$$

# Formalisierung

---

Zum Schluss müssen wir noch formalisieren was es bedeutet, dass der Angreifer Erfolg hat.

Dies ist dann der Fall, wenn er einen Schlüssel zur Kommunikation mit Bob hat, von dem Bob glaubt, es sei ein Schlüssel für Alice.

$$\exists x [Ik(key(x, b)) \wedge Bk(key(x, a))]$$

# Automatische Analyse

---

Die Formalisierung des Protokolls, Formeln (1)-(8) zusammen mit den Angreiferformeln (9)-(20) und der Erfolgsbedingung für den Angreifer kann man nun in einen Theorembeweiser eingeben.

Dieser antwortet “erfüllbar”.

Das beweist dann automatisch, dass der Angreifer das Protokoll brechen kann und der kritische Schlüssel die Zufallszahl  $N_a$  ist.

# Beispiel: Sicherheitsprotokolle

---

Was kann passieren?

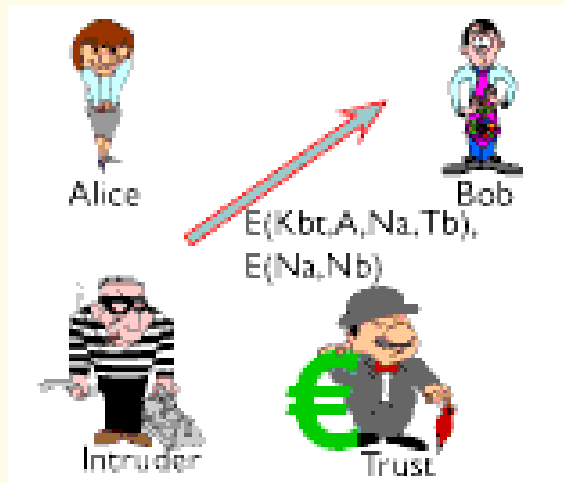


Charlie fängt die letzte Nachricht von Alice ab.

# Beispiel: Sicherheitsprotokolle

---

Was kann passieren?



Charlie schickt eine veränderte Nachricht an Bob.

# Beispiel: Sicherheitsprotokolle

Was kann passieren?

Intruder schickt die Nachricht 5:

$E(K_{bt}, A, N_a, T_b), E(N_a, N_b)$  an Bob.

Das Problem des Protokolls liegt darin, dass die Nachricht  $E(K_{bt}, A, K_{ab}, T_b)$  in Nachricht 4 und die Nachricht  $E(K_{bt}, A, N_a, T_b)$  in Nachricht 2 fast identisch sind.

Die beiden Nachrichten unterscheiden sich nur an der 2. Position (Nachricht 4 enthält dort den Schlüssel  $K_{ab}$ , Nachricht 2 die Zufallszahl  $N_a$ .)



Da der Intruder alle Nachrichten lesen und auch selbst Nachrichten schicken kann, fängt er also Nachricht 4 ab, wiederholt den obigen Teil aus Nachricht 2 und fügt  $E(N_a, N_b)$  hinzu.

Die beiden Zufallszahlen wurden bereits unverschlüsselt verschickt

Charlie schickt eine veränderte Nachricht an Bob.



# Beispiel: Sicherheitsprotokolle

---

Was kann passieren?



Bob entschlüsselt die Nachricht und denkt, sie komme von Alice, und dass  $Na$  der sichere Schlüssel zur Kommunikation mit Alice ist.

# Beispiel: Sicherheitsprotokolle

---

Was kann passieren?



Bob startet Kommunikation mit Alice . . .

# Beispiel: Sicherheitsprotokolle

---

Was kann passieren?



... aber spricht in Wirklichkeit mit Charlie.

# Beispiel: Sicherheitsprotokolle

---

Der Fehler ist tatsächlich leicht zu beheben.

Mit ein wenig Erfahrung erkennt man aus der Ausgabe des Theorembeweislers, wie es geht.

Dieser Angriff lässt sich reparieren, in dem man die Elemente des Protokolls typisiert, so dass Zufallszahlen und Schlüssel nicht mehr austauschbar sind.

Dies lässt sich dann auch wieder formalisieren; der Theorembeweis antwortet jetzt “unerfüllbar”.

# Beispiel: Sicherheitsprotokolle

---

## Literatur

[1] Neuman, B. C. and Stubblebine, S. G., 1993, A note on the use of timestamps as nonces, ACM SIGOPS, Operating Systems Review, 27(2), 10-14.

[2] Weidenbach, C., 1999, Towards an automatic analysis of security protocols in first-order logic, in H. Ganzinger, ed., 16th International Conference on Automated Deduction, CADE-16, Vol. 1632 of LNAI, Springer, pp. 378-382.

# Andere Anwendungsbereiche

## MATHEMATIK

### Aufgaben

- Beweisen
- Beweisüberprüfung

### Theorien

- Zahlen
- Polynomen
- Funktionen auf Zahlenbereichen
- Algebren

## VERIFIKATION

### Aufgaben

- **Programme**
  - Korrektheit
  - Terminierung
- **reaktive/hybride Systeme**
  - Sicherheit

### Theorien

- Zahlen
- Datentypen
- Funktionen auf Zahlenbereichen

### Beispiel: Programmverifikation

```
int [] BUBBLESORT(int[] a) {  
  int i, j, t;  
  for (i := |a| - 1; i > 0; i := i - 1) {  
    for (j := 0; j < i; j := j + 1) {  
      if (a[j] > a[j + 1]) { t := a[j];  
                           a[j] := a[j + 1];  
                           a[j + 1] := t};  
    }  
  }  
  return a;  
}
```

# Andere Anwendungsbereiche

## MATHEMATIK

### Aufgaben

- Beweisen
- Beweisüberprüfung

### Theorien

- Zahlen
- Polynomen
- Funktionen auf Zahlenbereichen
- Algebren

## VERIFIKATION

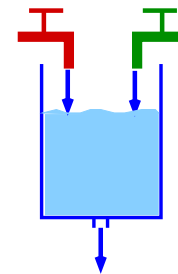
### Aufgaben

- **Programme**
- Korrektheit
- Terminierung
- **reaktive/hybride Systeme**
- Sicherheit

### Theorien

- Zahlen
- Datentypen
- Funktionen auf Zahlenbereichen

## Controllers



### Überprüfe:

- Kein "overflow"
- Substanzen in der richtigen Proportion
- ... andernfalls:  
Tank kann in  $\leq 200s$  geleert werden.

# Weitere Anwendungsbereiche

## MATHEMATIK

### Aufgaben

- Beweisen
- Beweisüberprüfung

### Theorien

- Zahlen
- Polynomen
- Funktionen auf Zahlenbereichen
- Algebren

## VERIFIKATION

### Aufgaben

- **Programme**
- Korrektheit
- Terminierung

### – reaktive/hybride Systeme

- Sicherheit

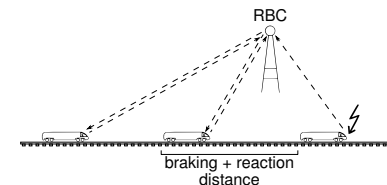
### Theorien

- Zahlen
- Datentypen
- Funktionen auf Zahlenbereichen

## Controllers



### Zugkontrollsysteme

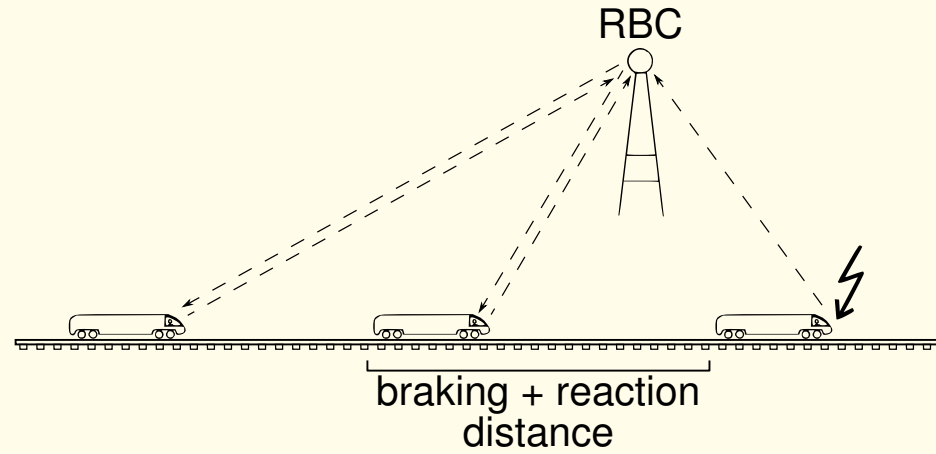


- **Task:** Keine Kollision



# Beispiel: ETCS Fallstudie

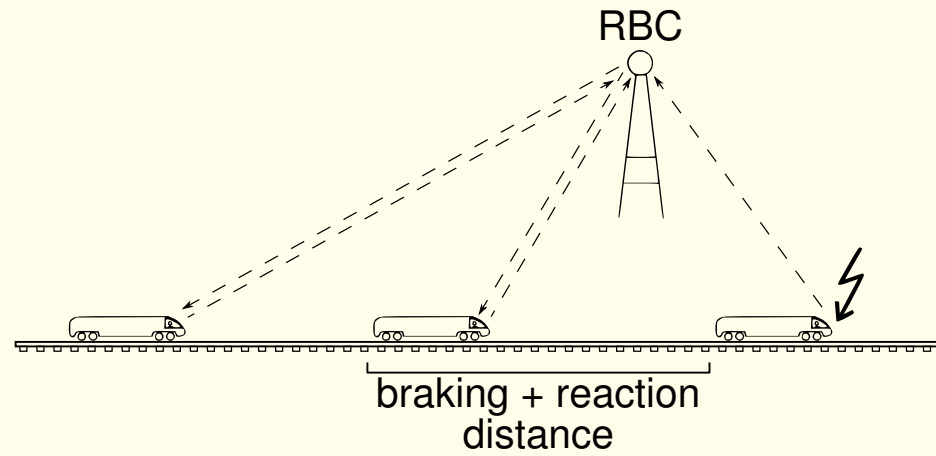
AVACS: ETCS Fallstudie [Jacobs,VS'06,'07; Faber,Jacobs,VS'07]



Anzahl der Züge:	$n > 0$	$\mathbb{Z}$
Minimaler sicherer Abstand:	$l_{\text{alarm}} > 0$	$\mathbb{R}$
Minimale / maximale Geschwindigkeit :	$0 \leq \text{min} < \text{max}$	$\mathbb{R}$
Zeit zwischen Updates:	$\Delta t > 0$	$\mathbb{R}$
Zugpositionierung vor/nach Update:	$\text{pos}, \text{pos}'$	$: \mathbb{Z} \rightarrow \mathbb{R}$

# Beispiel: ETCS Fallstudie

AVACS: ETCS Fallstudie [Jacobs,VS'06,'07; Faber,Jacobs,VS'07]



Update(pos, pos') :

- $\forall i (i = 0 \rightarrow pos(i) + \Delta t * \min \leq pos'(i) \leq pos(i) + \Delta t * \max)$
- $\forall i (0 < i < n \wedge pos(i-1) > 0 \wedge pos(i-1) - pos(i) \geq l_{\text{alarm}} \rightarrow pos(i) + \Delta t * \min \leq pos'(i) \leq pos(i) + \Delta t * \max)$

...

**Induktive Invariante: Keine Kollision**

$Sicher(pos) \wedge Update(pos, pos') \wedge \neg Sicher(pos') \models \perp$

**Sicher(pos) :**

$\forall i, j (i < j \rightarrow pos(i) > pos(j))$

# Anwendungen

---

- **Vorlesung:** Decision procedures for verification:  
BSc, MSc, Normalerweise in Wintersemester
- **Vorlesung:** Formal specification and verification:  
MSc, Normalerweise in Sommersemester

# Weitere Logiken

---

- Alternativen zur klassischen Logik
- Erweiterungen der klassischen Logik

# Weitere Logiken

---

- **Alternativen:** z.B. Mehrwertige Logiken

Ausgangspunkt für die Entwicklung mehrwertiger Logiken war die Frage, ob Annahme, das es nur zwei Wahrheitswerte 0 (falsch) und 1 (wahr) geben kann nicht zu Einschränkungen führen kann.

Für Aussagen über die Zukunft stellte bereits Aristoteles diese Frage, indem er argumentierte, dass die Wahrheit einer Aussage wie

“Morgen wird eine Seeschlacht stattfinden”

erst am Abend des morgigen Tages feststehen wird und dass sie bis zu diesem Zeitpunkt noch als unbestimmt betrachtet werden muss.

# Weitere Logiken

---

- **Alternativen:** z.B. Mehrwertige Logiken

Wahrheitswerten: Menge  $W$  mit  $0, 1 \in W$

z.B:  $W = \{0, 1, \text{unbekannt}\}$

$W = \{0, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}, 1\}$  oder  $W = [0, 1]$ .

# Weitere Logiken

---

- **Alternativen:** z.B. Mehrwertige Logiken.

Wahrheitswerten: Menge  $W$  mit  $0, 1 \in W$

z.B:  $W = \{0, 1, \text{unbekannt}\}$

$$W = \{0, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}, 1\} \text{ oder } W = [0, 1].$$

In neuerer Zeit haben mehrwertige Logiken im Bereich der Informatik hohe praktische Bedeutung gewonnen.

Sie ermöglichen den Umgang mit der Tatsache, dass Datenbanken nicht nur eindeutig bestimmte, sondern auch unbestimmte, fehlende oder sogar widersprüchliche Informationen enthalten können.

# Weitere Logiken

---

- **Erweiterungen:**

In einer nichtklassischen Erweiterung der klassischen Logik werden zusätzliche logische Operatoren hinzugefügt.

**Beispiele:**

- Modale Logiken
- Dynamische Logik
- Temporale Logik



# Weitere Logiken

---

- **Erweiterungen:**
  - **Modale Logiken:** Zusätzliche logische Operatoren:
    - “ $\square$ ” : steht für “Es ist notwendig, dass...” .
    - “ $\diamond$ ” : steht für “Es ist möglich, dass...” .

# Weitere Logiken

---

- **Erweiterungen:**
  - **Modale Logiken:** Zusätzliche logische Operatoren:
    - “ $\Box$ ” : steht für “Es ist notwendig, dass...” .
    - “ $\Diamond$ ” : steht für “Es ist möglich, dass...” .
  - **Dynamische Logik:** Zusätzliche logische Operatoren:
    - $[\alpha]$ ,  $\langle \alpha \rangle$ ,  $\alpha$  Programm.
    - $[\alpha]F$ :  $F$  gilt nach jeder Ausführung von  $\alpha$
    - $\langle \alpha \rangle$ : es gibt eine Ausführung von  $\alpha$  nach der  $F$  gilt

# Weitere Logiken

---

- **Erweiterungen:**

In einer nichtklassischen Erweiterung werden zusätzliche logische Operatoren hinzugefügt.

- **Temporale Logiken:** Erweiterungen der Logik, durch die zeitliche Abläufe erfasst werden können.

Die Aussagenlogik kann Aussagen, deren Wahrheitswerte sich mit der Zeit ändern, nicht oder nur mit Mühe adäquat behandeln.

So ist “Es regnet” nur wahr, wenn es am Ort und zur Zeit der Äusserung gerade regnet, sonst nicht.

Klassische Logiken zählen daher den Äußerungszeitpunkt zu den Wahrheitsbedingungen.

# Weitere Logiken

---

- **Erweiterungen:**

In einer nichtklassischen Erweiterung werden zusätzliche logische Operatoren hinzugefügt.

- **Temporale Logiken:** Erweiterungen der Logik, durch die zeitliche Abläufe erfasst werden können.

Zeitlogiken führen Operatoren ein, so dass jeder Fall der Äusserung des Satzes unter denselben Wahrheitsbedingungen steht.

Diese Operatoren lassen es zu, differenziertere zeitliche Aussagen logisch zu analysieren, so dass der Wahrheitswert von “Es hat geregnet”, “Es wird regnen”, “Es regnet immer” von der Erfüllung von “Es regnet” zu bestimmten Zeitpunkten abhängig ist.

Temporale Logik wird in der Informatik für die Programm Spezifikation und Verifikation benutzt.

# Weitere Logiken

---

- **Vorlesung:** Nicht-klassische Logiken  
MSc, Normalerweise in Wintersemester.

# Logik höherer Stufe

---

Logik höherer Stufe erweitert die Prädikatenlogik erster Stufe um die Möglichkeit, über alle Relationen/Funktionen zu quantifizieren.

# Jetzt

---

**Themen**

**Question/Answer Session**