Ansagen:

- letzte Vorlesung: 25. Juni
- letzte Vorlesungswoche (8. oder 9. Juli): Projektabschluss
- bitte an alle Firmen: KI in svn laden!
- Hinweis: attraktive Preise beim Projektabschluss!

Berechenbarkeit:

Universelle Sprache

$$U = \{\langle M\rangle w \mid M \text{ akzeptiert } w\}$$

U ist nicht rekursiv aber rekursiv aufzählbar.

Spezielles Halteproblem:

$$H_\varepsilon = \{\langle M\rangle \mid M \text{ hält an auf } \varepsilon\}$$

H_ε ist nicht rekursiv $\Rightarrow$ $\overline{H_\varepsilon}$ nicht rekursiv.

Berechenbare Funktion:

$Q \subseteq \Sigma^*$, $f : Q \to \Sigma^*$ heißt berechenbar, falls es eine DTM gibt, die auf Eingabe $x \in Q$ die Ausgabe $f(x)$ liefert und auf Eingabe $x \notin Q$ nicht anhält oder nicht akzeptiert.

Spezielles Beispiel:

$Q = \emptyset$ (leere Menge)

$u : \emptyset \to \Sigma^*$ ist berechenbar

zugehörige DTM: Gehe im ersten Schritt in nicht-akz. Endzustand.

Satz von Rice:

Es sei: R Menge aller berechenbaren Fkt. und S nicht-triviale Teilmenge von R, d.h. $\emptyset \neq S \subsetneq R$. Definiere Sprache

$L(S) = \{ <M> \mid M$ berechnet Fkt. aus $S \}$

Dann ist $L(S)$ nicht rekursiv.

Das heißt: Ein Computer kann keinerlei bedeutende Aussage über einen gegebenen Algorithmus treffen.

<u>Beweis:</u> Widerspruchsbeweis: Annahme M_S entscheidet $L(S)$. Konstruiere daraus eine DTM M', die $\overline{H_\varepsilon}$ entscheidet. ($\Rightarrow$ ↯)

Betrachte im Folgenden die spezielle Fkt. $u \in R$ mit

$$u: \emptyset \to \Sigma^*$$

Nehme o.B.d.A. an, dass $u \in S$ (denn: andernfalls ersetze S durch $R \setminus S$ und beachte, dass $L(R \setminus S) = \overline{L(S)}$).

Weiter sei $f \in R \setminus S$ und M_f eine DTM, die f berechnet.

Die DTM M' arbeitet wie folgt (Eingabe $<M>$):

1) Berechne Gödelnummer $<M''>$ der DTM M'', die auf Eingabe x das Folgende tut:

M'' simuliert M auf Eingabe ε;
falls M auf ε anhält, so simuliert M''
die DTM M_f auf x.

2) Simuliere DTM M_S auf $\langle M'' \rangle$.

<u>Dann gilt:</u>

1. Fall: M hält auf ε nicht an
   $\Rightarrow$ M'' berechnet $u \in S$
   $\Rightarrow$ M_S akzeptiert $\langle M'' \rangle$
   $\Rightarrow$ M' akzeptiert $\langle M \rangle$

2. Fall: M hält auf ε an:
   $\Rightarrow$ M'' berechnet $f \in R \setminus S$.
   $\Rightarrow$ M_S akzeptiert $\langle M'' \rangle$ nicht.
   $\Rightarrow$ M' akzeptiert $\langle M \rangle$ nicht.

Folglich entscheidet M' die Sprache H_ε $\lightning$ $\Box$

<u>Satz:</u> Es seien L, L_1, L_2 rekursive Sprachen.
Dann sind $\overline{L}$, $L_1 \cup L_2$ und $L_1 \cap L_2$ ebenfalls rekursiv.

<u>Beweis:</u> $\overline{L}$ klar, vertausche akz. und nicht-akz. Endzustände der DTM, die L entscheidet.

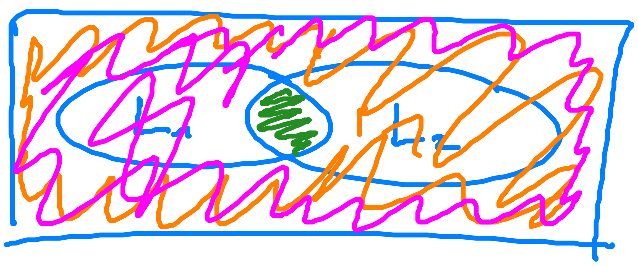
<u>$L_1 \cup L_2$</u>: Rufe zunächst DTM auf, die ent-

scheidet, ob Eingabe $x \in L_1$. Falls ja $\Rightarrow$ fertig. Falls nein, rufe DTM auf, die entscheidet, ob $x \in L_2$. Falls ja $\Rightarrow$ ja, falls nein $\Rightarrow$ nein.

<u>Alternativer Beweis:</u> Lasse die beiden DTM für L_1 und L_2 „parallel" auf einer 2-Band-TM laufen. Akzeptiere, falls mindestens eine von beiden akzeptiert.

<u>$L_1 \cap L_2$:</u> Wie oben.

Alternativ: $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$



<u>Satz:</u> Es seien L_1, L_2 rekursiv aufzählbar, dann sind auch $L_1 \cup L_2$ und $L_1 \cap L_2$ rekursiv aufzählbar.

<u>Beweis:</u> Nutze Parallelsimulation von oben. □

<u>Lemma:</u> Sind L und $\overline{L}$ rekursiv aufzählbar, so ist L rekursiv.

<u>Beweis:</u> Parallelsimulation (s.o.). □

<u>Satz:</u> Die Menge der rekursiv aufzähl-

baren Fkt. ist nicht abgeschlossen unter Komplementbildung. Insbesondere ist U rekursiv aufzählbar, $\overline{U}$ jedoch nicht.

Beweis: U ist rekursiv aufzählbar aber nicht rekursiv. $\square$

Korollar: Für jede Sprache L gilt genau eine der folgenden 3 Eigenschaften.

1) L und $\overline{L}$ sind rekursiv.
2) L und $\overline{L}$ sind nicht rekursiv aufzählbar.
3) Genau eine der beiden Sprachen L und $\overline{L}$ ist rekursiv aufzählbar (aber nicht rekursiv!)

Der Gödelsche Unvollständigkeitssatz.

(Literatur: siehe beispielsweise Buch von Michael Sipser: Introduction to the Theory of Computing, Kapitel 6.2)

Betrachte mathematische Ausdrücke der Form:

1) $\forall q \in \mathbb{N}_0\ \exists p \in \mathbb{N}_0\ \forall x,y \in \mathbb{N}_0\ (p>q \wedge$
$(x,y>1 \Rightarrow x \cdot y \neq p))$
„Es gibt unendlich viele Primzahl"

2) $\forall a,b,c,n \in \mathbb{N}_0\ ((a,b,c>0 \wedge n>2) \Rightarrow$
$a^n + b^n \neq c^n)$

Fermat's letzter Satz
Satz von Wiles

3) $\forall q \in \mathbb{N}_0\ \exists p \in \mathbb{N}_0\ \forall x,y \in \mathbb{N}_0$
$(p>q \wedge (x,y>1 \Rightarrow (x \cdot y \neq p \wedge x \cdot y \neq p+2)))$
„Es gibt unendlich viele Primzahlzwillinge"

Eine Version von Gödels Unvollst.-Satzes besagt, dass es wahre Ausdrücke dieser Form gibt, die jedoch nicht beweisbar sind.

Um diesen Satz präzise zu formulieren, müssen wir zunächst genau definieren, welche Art von Ausdrücken wir betrachten.

Es sei $\Sigma = \{\wedge, \vee, \neg, (,), \forall, \exists, x, R_1, \ldots, R_k\}$.

Das Symbol x wird verwendet, um Variablen über $\mathbb{N}_0$ zu kodieren. Wir verwenden im Folgenden $x_i := \underbrace{xx\ldots x}_{i\text{-mal}}$ für $i \geq 1$

$R_1, \ldots, R_k$ stellen Relationen über $x_1, \ldots$ dar.

<u>Beispiele:</u>

Additionsrelation:

$R_1(x_i, x_j, x_k) =$ „wahr" $:\Longleftrightarrow x_i + x_j = x_k$

Multiplikationsrelation

$R_2(x_i, x_j, x_k) =$ „wahr" $:\Longleftrightarrow x_i \cdot x_j = x_k$

$\leq$ - Relation:

$R_3(x_i, x_j) =$ „wahr" $:\Longleftrightarrow x_i \leq x_j$

<u>Def.:</u> Ein Satz ist ein Ausdruck $w \in \Sigma^*$ der Form:

$Q_1 x_1 Q_2 x_2 \ldots Q_\ell x_\ell \phi$

wobei: $Q_1, \ldots, Q_\ell \in \{\forall, \exists\}$ und die Formel ϕ ist rekursiv wie folgt definiert:

1) $\phi = R_j(x_{i_1}, x_{i_2}, \ldots, x_{i_m})$ für $1 \leq j \leq k$, $1 \leq i_r \leq \ell$ ist Formel.

2) $\phi = \phi_1 \wedge \phi_2$ oder $\phi = \phi_1 \vee \phi_2$ oder $\phi = \neg \phi_1$ wobei: ϕ_1, ϕ_2 kürzere Formeln sind.

Ein Satz ist also ein mathematischer

Ausdruck, der wahr oder falsch ist.