Selbstreproduzierende Automaten und Programme

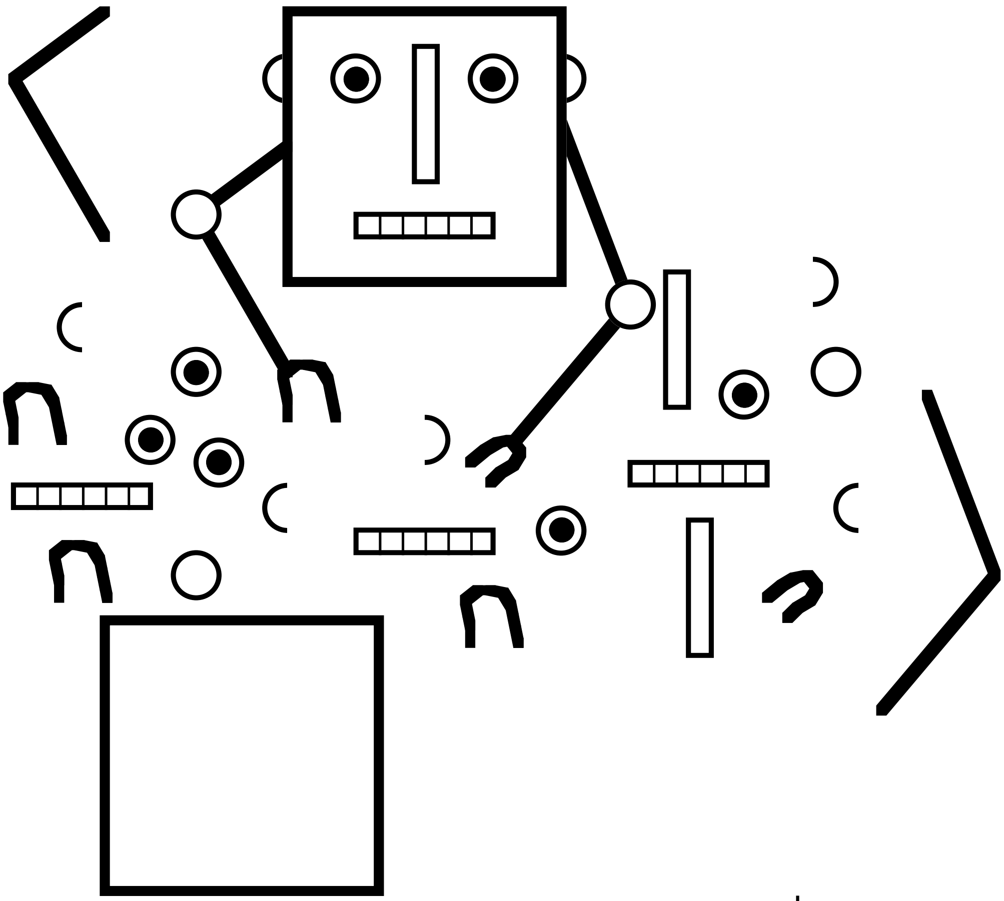
Andreas Schwill
Institut für Informatik
Universität Potsdam

Problem: Gesucht ist ein Automat, der bei Zuführung einer hinreichend großen Menge an Rohstoffen eine exakte Kopie von sich selbst anfertigt (**selbstreproduzierender Automat**)

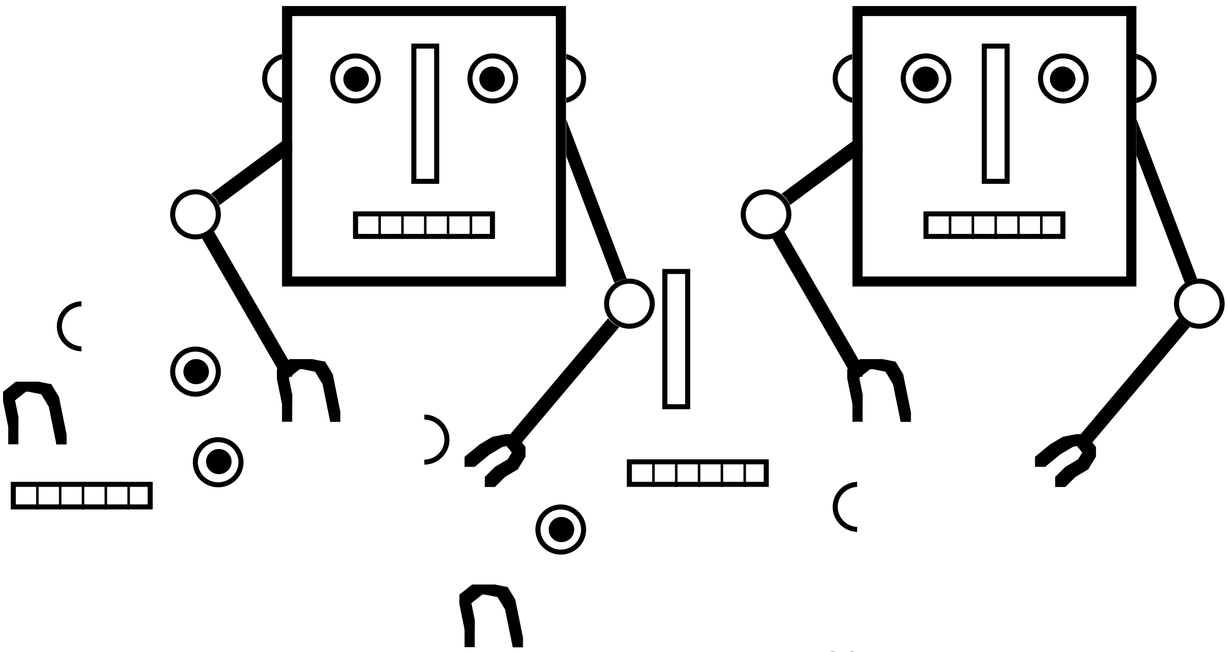
Motivation:

- biologische Prozesse: Selbstreproduktion und Mutation (d.h. „fehlerhafte“ Selbstreproduktion)
- vollautomatische Fabrik: Fabrik fertigt sich selbst.

Erste Untersuchungen von **John von Neumann** (1966).



vorher



nachher

Naive Intuition: Solch einen Automaten gibt es nicht.

Begründung: Fertigende Maschinen stets komplexer als zu Fertigende.
Beispiel: Automatische Drehbank recht kompliziert, aber
produzierte Kurbelwelle ausgesprochen „einfach“.

Genauer: Die Maschine M soll eine Maschine M’ herstellen.
M benötigt offenbar:

- eine exakte Beschreibung (einen **Bauplan**) von M’
- eine Reihe von **Bauteilen**
- Greifarme zur Montage etc.
- eine **Steuereinheit**, die den Bauplan interpretiert und in Steuerbefehle für die Arme etc. umsetzt.

Folgerung: Die Struktur von M ist wesentlich komplexer als die von M’.

Diese naive Anschauung ist **falsch**. Unsere Erfahrung täuscht hier. Warum?

- Im täglichen Leben nur sehr einfache Automaten und
- sehr ungenaue Vorstellung über prinzipielle Fähigkeiten von sehr komplexen Automaten.

a) Anschaulicher Beweis

b) Formaler Beweis mit dem Rekursionstheorem

Anschaulicher Beweis:

Gegeben: unerschöpflicher Vorrat an Bauteilen (z.B. Schrauben, Räder, Zahnräder, Bleche, Magnetbänder zum Speichern usw.)

Aufgabe: Konstruiere einen selbstreproduzierenden Automaten R.

Lösungsidee (J. von Neumann): Entwirf drei Baupläne für drei Maschinen, die geeignet zusammengeschaltet selbstreproduzierend (=R) sind:

- Universalkonstruktor U,
- Magnetbandkopierer D,
- Kontrollautomat C.

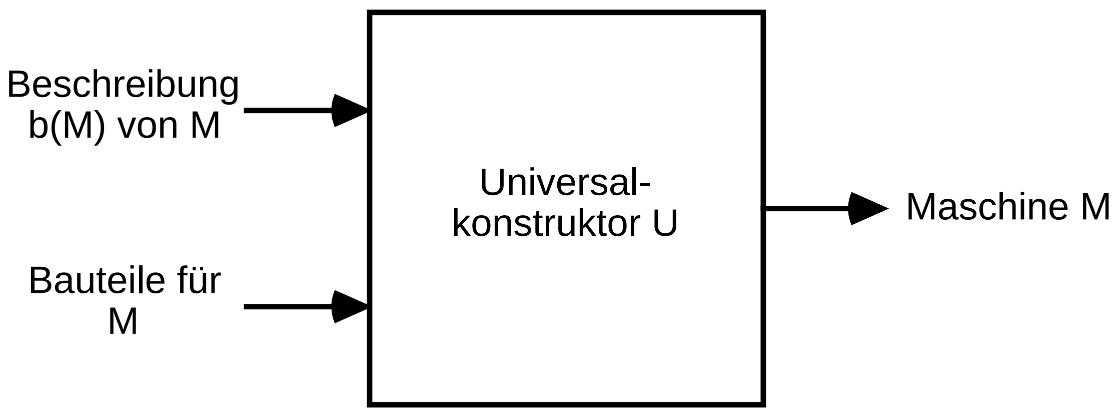
Universalkonstruktor U.

Eingabe: Beschreibung b(M) und Bauteile für M

Ausgabe: Maschine M

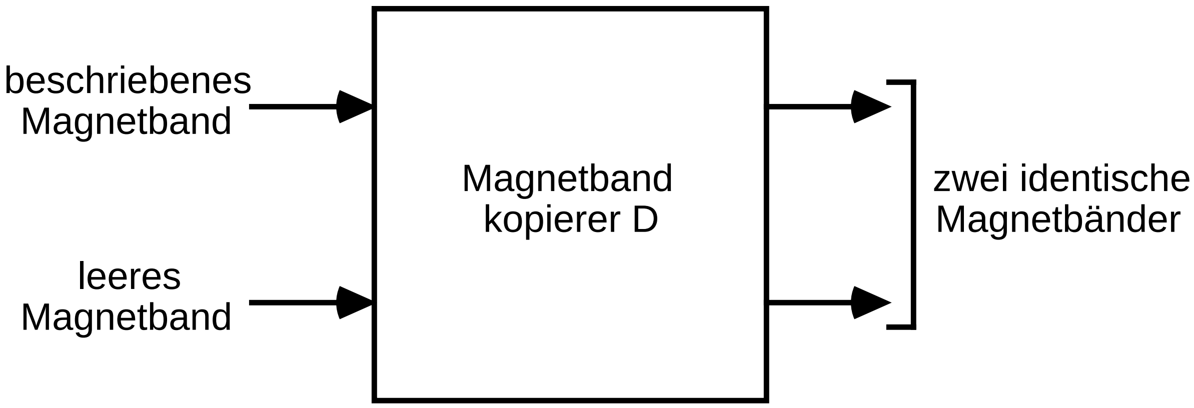
Beschreibungen b(M) seien auf Magnetbändern gespeichert.

U entspricht etwa einer automatischen Fabrik.



Magnetbandkopierer D.

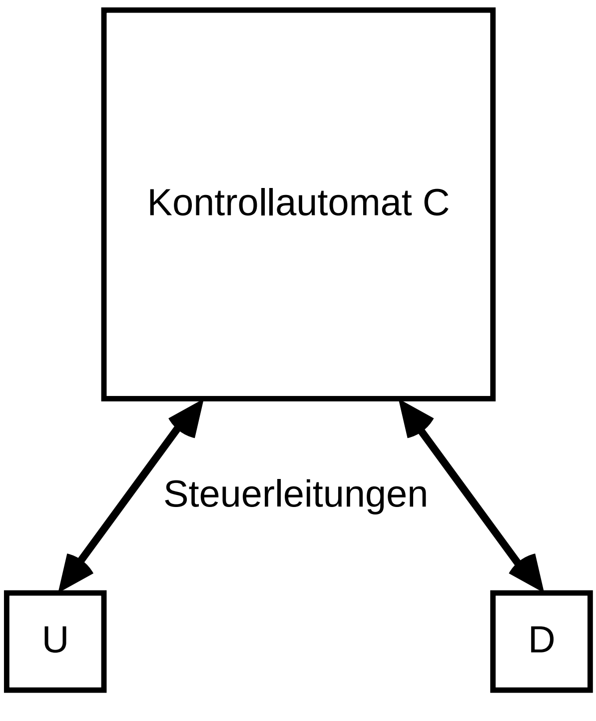
Kopiert eingegebenes Magnetband auf ein anderes leeres Band.



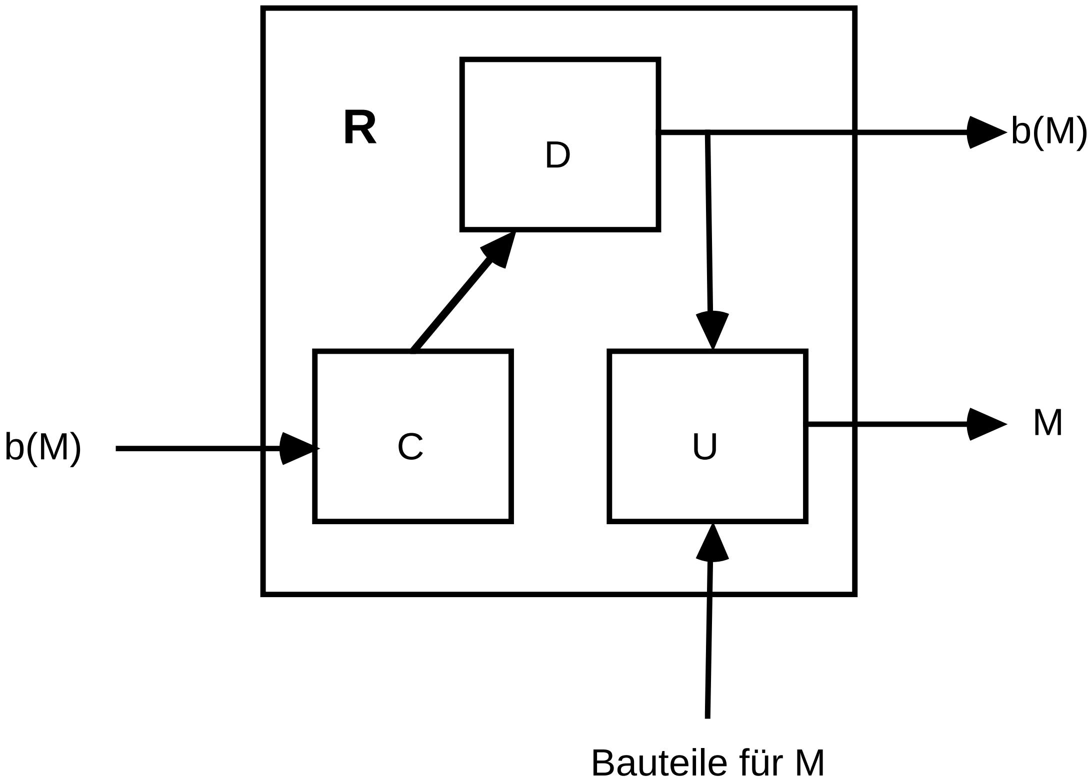
Kontrollautomat C.

Überwacht die Arbeitsweise von U und D.

C kann U und D zu beliebigen Zeiten starten, stoppen und mit Magnetbändern versorgen.



Zusammenbau von U, D und C zum selbstreprod. Automaten R:



M beliebige Maschine.

b(M) die auf Magnetband gespeicherte Beschreibung von M:

1. Übergib b(M) auf Band an die Maschine R:=U+D+C.
2. C spannt das Band in den Kopierer D ein. D kopiert das Band. Ergebnis: zwei identische Kopien von b(M).
3. C gibt eine der beiden Kopien an U. U fertigt aus b(M) die Maschine M.
4. C übergibt anschließend die andere Kopie von b(M) an M.

Zusammenfassung: R+b(M)=U+D+C+b(M) $\rightarrow$ M+b(M).

Setze M:=R. Dann gilt: R+b(R)=U+D+C+b(U+D+C) $\rightarrow$ R+b(R).

Beachte: b(M) ist endlich, denn

- M besteht nur aus endlich vielen Bauteilen
- die Bauteile können nur auf endlich viele Arten kombiniert werden.

Formaler Beweis:

Modifiziertes Problem:

Gibt es ein Programm ohne Eingabe, das seinen eigenen Text ausgibt?

Naive Lösung:

```
program repro(output);
begin
        writeln(‚program repro(output);‘);
        writeln(‚begin‘);
        writeln(‚writeln(‚‘program repro(output);’’);’);
        writeln(‚writeln(‚‘begin’’);’);
        writeln(‚writeln(‚writeln(‚‘’’program repro(output);’’’’);’);
        ...
end.
```

1. Problem: die Hochkommas.

   Lösung: Standardfunktion chr zu Hilfe nehmen.

   Statt writeln(‚...‘’...’) schreibe writeln(‚...‘,chr(39),’...’).

2. Problem: Ausgabe hinkt der Verarbeitung hinterher („Überholproblem“).

   Lösung: durch Textkonstanten oder Prozeduren.

Grundlegende Definitionen:

a) $\mathbb{P}_A{}^i$ = {f: $(A^*)^i \rightarrow A^*$ | f ist Turing-berechenbar} Menge der berechenbaren Funktionen über einem Alphabet A.

Beispiel: <u>var</u> $w_1, w_2, ..., w_i$, y: text;

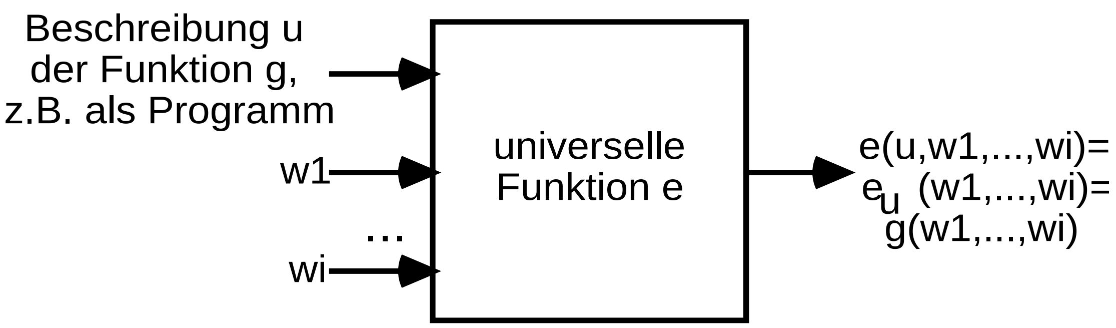
read($w_1, w_2, ..., w_i$); ...; write(y).

$\mathbb{P}_A{}^0$: Menge aller konstanten Funktionen (**keine read's**) mit Werten in A*, also $\mathbb{P}_A{}^0 = A^*$.

b) Für $f \in \mathbb{P}_A{}^{i+1}$ sei $f_u \in \mathbb{P}_A{}^i$ definiert durch $f_u(x_1, ..., x_i) = f(u, x_1, ..., x_i)$.

Beispiel: f(x,y)=x+y => $f_x(y)$=x+y, z.B. $f_7(y)$=7+y.

c) $e \in \mathbb{P}_A{}^{i+1}$ sei eine **universelle Funktion** für $\mathbb{P}_A{}^i$.



Definition A:

Ein Programm $u \in A^*$ heißt **selbstreproduzierend**, wenn gilt:

- u berechnet eine Funktion aus $\mathbb{P}_A{}^0 = A^*$, also eine konstante Funktion,
- $e_u = u$, d.h., u gibt sich selbst aus.

Das Rekursionstheorem

Hilssatz B: S-n-m-Theorem.

Zu allen natürlichen Zahlen m und n existiert eine (totale) Funktion $S_n^m \in \mathbb{P}_A^{m+1}$, so daß für alle $u, x_1, ..., x_m, y_1, ..., y_n \in A^*$ gilt:

$$e_u(x_1, ..., x_m, y_1, ..., y_n) = e_{S_n^m(u, x_1, ..., x_m)}(y_1, ..., y_n).$$

Beweisidee aus Sicht der Programmierung:

u: read($a_1, ..., a_m, b_1, ..., b_n$); A; write(c).

Eingabe für die Variablen $a_1, ..., a_m$ von u: die Werte $x_1, ..., x_m$.

Dann gilt: $S_n^m(u, x_1, ..., x_m) = u'$ mit

u': $a_1 := x_1$; ...; $a_m := x_m$; read($b_1, ..., b_n$); A; write(c).

Falls u nicht von der gewünschten Form ist, dann:

$S_n^m(u, x_1, ..., x_m)$=<u>while</u> <u>true</u> <u>do</u>;

Satz C: Rekursionstheorem, Version 1.

Zu jedem $i \geq 0$ und zu jeder berechenbaren Funktion $f \in \mathbb{P}_A^{i+1}$ gibt es ein Wort $w \in A^*$, so daß für alle $x_1,\dots,x_i \in A^*$ gilt:

$$f(w,x_1,\dots,x_i) = e_w(x_1,\dots,x_i).$$

Beweis:

Sei $S = S_i^1$ die Funktion aus Satz B für m=1 und n=i. Sei $i \geq 0$ und e eine universelle Funktion, und sei $f \in \mathbb{P}_A^{i+1}$ eine beliebige Funktion. Bilde dann die Funktion $g \in \mathbb{P}_A^{i+1}$, die für alle $u,x_1,\dots,x_i \in A^*$ definiert ist durch:

$$g(u,x_1,\dots,x_i) = f(S(u,u),x_1,\dots,x_i).$$

Wegen $g \in \mathbb{P}_A^{i+1}$ existiert ein $v \in A^*$ mit $g = e_v$. Satz C ist nun für das Wort $w = S(v,v)$ erfüllt. Denn es gilt für alle $x_1,\dots,x_i \in A^*$:

$$
\begin{aligned}
f(S(v,v),x_1,\dots,x_i) &= g(v,x_1,\dots,x_i) && \text{nach Definition von } g\\
&= e_v(v,x_1,\dots,x_i) && \text{nach Definition von } v\\
&= e_{S(v,v)}(x_1,\dots,x_i) && \text{nach Satz B.}
\end{aligned}
$$

Also ist $f(w,x_1,\dots,x_i) = e_w(x_1,\dots,x_i)$ für alle $x_1,\dots,x_i \in A^*$.

♦

Satz D: Rekursionstheorem, Version 2.

Zu jeder totalen Funktion $f \in \mathbb{P}_A^1$ und zu jedem $i \geq 0$ gibt es ein Wort $w \in A^*$, so daß für alle $x_1,...,x_i \in A^*$ gilt:

$$e_{f(w)}(x_1,...,x_i) = e_w(x_1,...,x_i),$$

wobei e eine universelle Funktion für $\mathbb{P}_A^i$ ist.

Beweis: Analog zu Satz C:

Sei $S = S_i^1$ die Funktion aus Satz B für m=1 und n=i. Sei $i \geq 0$ und e eine universelle Funktion für $\mathbb{P}_A^i$, und sei $f \in \mathbb{P}_A^1$ eine beliebige totale Funktion. Bilde dann die Funktion $g \in \mathbb{P}_A^{i+1}$, die für alle $u,x_1,...,x_i \in A^*$ definiert ist durch:

$$g(u,x_1,...,x_i) = e_{f(S(u,u))}(x_1,...,x_i).$$

Wegen $g \in \mathbb{P}_A^{i+1}$ existiert ein $v \in A^*$ mit $g = e_v$. Satz D ist nun für das Wort $w = S(v,v)$ erfüllt. Denn es gilt für alle $x_1,...,x_i \in A^*$:

$$\begin{aligned} e_{f(S(v,v))}(x_1,...,x_i) &= g(v,x_1,...,x_i) && \text{nach Definition von g} \\ &= e_v(v,x_1,...,x_i) && \text{nach Definition von v} \\ &= e_{S(v,v)}(x_1,...,x_i) && \text{nach Satz B.} \end{aligned}$$

Also ist $e_{f(w)}(x_1,...,x_i) = e_w(x_1,...,x_i)$ für alle $x \in A^*$. ♦

Aussage von Satz D:

- e besitzt mindestens einen Fixpunkt w bezgl. f.
- w kann man effektiv als w = S(v,v) ermitteln.

Folgerung E:

Es gibt ein selbstreproduzierendes Programm.

Beweis: Sei die Funktion $f: A^* \to A^*$ definiert durch

$$f(x)=x \text{ für alle } x \in A^*.$$

f ist berechenbar. Nach dem Rekursionstheorem, Version 1, gibt es ein Wort

$w \in A^*$, so daß gilt:

$$f(w) = e_W.$$

Andererseits folgt aus der Defintion von f

$$f(w) = w \quad \text{und daher} \quad e_W = w.$$

w ist das gesuchte selbstreproduzierende Programm. ♦

Aufgabe:

Man zeige, daß es zu jedem Programm P mit einem Ein- und einem Ausgabewert eine selbstreproduzierende Version gibt, also ein Programm P', das zunächst dieselbe Funktion berechnet wie P und sich anschließend selbst reproduziert.

Lösung: Sei P ein beliebiges Programm mit einem Ein- und einem Ausgabewert. $g: A^* \to A^*$ sei die durch P berechnete Funktion. Wir definieren eine Funktion

$$f: A^* \times A^* \to A^* \text{ durch}$$

$$f(x,y) = g(y) \cdot x.$$

Da g nach Voraussetzung berechenbar ist, ist offenbar auch f berechenbar. Nach dem Rekursionstheorem, Version 1, gibt es dann ein Wort $w \in A^*$, so daß für alle $x \in A^*$ gilt:

$$f(w,x) = e_W(x).$$

Andererseits folgt aus der Definition von f die Bedingung

$$f(w,x) = g(x) \cdot w$$

und daher

$$e_W(x) = g(x) \cdot w.$$

w ist das gesuchte selbstreproduzierende Programm. Es berechnet zunächst die Funktion g von P und gibt sich anschließend selbst aus. ♦

Wie findet man ein selbstreproduzierendes Programm in PASCAL?

Satz F:

Es gibt ein selbstreproduzierendes PASCAL-Programm.

Beweis:

```
program repro2(output);
const  d = 39;
       b = ‚;begin writeln(c,chr(d),b,chr(d),chr(59));
            writeln(chr(67),chr(61),chr(d),c,chr(d),b) end.’;
       c = ‚program repro2(output); const d=39;b=‘;
begin
      writeln(c, chr(d), b, chr(d), chr(59));
      writeln(chr(67), chr(61), chr(d), c, chr(d), b)
end.                         ♦
```

Aufgabe

Eine feindliche Macht hat Zugang zum zentralen Rechenzentrum des Gegners bekommen, in dem **alle** Programme

$$P_1, P_2, P_3, \ldots$$

aufbewahrt werden. Alle Programme mögen Funktionen von A* nach A* berechnen.

Die Programme sollen zerstört werden. Einfach alle Programme zu löschen, hilft nicht weiter, da der Gegner dann den Sabotageakt sofort merkt und entsprechende Gegenmaßnahmen einleiten kann. Daher sollen *alle* diese Programme systematisch nach einem vorgegebenen Verfahren verändert werden, so daß nachher *jedes* Programm fehlerhafte Resultate errechnet. Prüfen Sie mithilfe des Rekursionstheorems (1. oder 2. Fassung) nach, ob für das Unternehmen Aussicht auf Erfolg besteht.

Beispiele:

a) Ersetze überall + durch - und - durch +.

b) Ersetze überall while B do A end durch if B then A.

Lösung: Alle Programme P_1, P_2, P_3, ... sollen systematisch, also durch ein algorithmisches Verfahren α so in Programme P_1', P_2', P_3', ... abgeändert werden, daß für alle $i \geq 1$ gilt: P_i und P_i' berechnen unterschiedliche Funktionen.

α berechnet also eine Funktion

$$f: A^* \rightarrow A^* \text{ mit}$$

$$f(x) = \begin{cases} P_i', \text{ falls } x = P_i \text{ für ein } i \in \mathbb{N}, \\ \text{„read(x); write(x)“, sonst.} \end{cases}$$

f ist nach Voraussetzung berechenbar.

Nach dem Rekursionstheorem, Version 2, gibt es dann ein $w \in A^*$, so daß für alle $x \in A^*$ gilt:

$$e_{f(w)}(x) = e_w(x),$$

wobei e eine universelle Funktion ist. Daraus folgt also: w und f(w) berechnen die gleiche Funktion. Bei w muß es sich offenbar um eines der Programme P_i handeln. Folglich berechnen das Originalprogramm P_i und das geänderte Programm $f(P_i)=P_i'$ die gleiche Funktion.

Der Sabotageakt, alle Programme systematisch zu verändern, kann also nicht zum Erfolg führen.