Aufgabe 1

//Halteproblem?

```
Programm P1 {
  extern function halteproblemLoesung(Programm p) returns boolean;
  function main(Programm P2) returns boolean {
    return halteproblemLoesung(P2);
  }
}

Programm P3 {
  extern function P1.main(Programm p) returns boolean;
  function main(Programm P2) returns int {
    boolean variable_haltet = P1.main(P2);
    if (variable_haltet) while(1) { }          // Endlosschleife
    else return 7;                             // Rückgabe und anhalten
  }
}
```

Falls P1 existiert, so muss auch P3 existieren (weil P3 steht ja da und P3 ist ein beliebiges und von P1 verschiedenes („anderes“) Programm).
Angenommen, P3(P3) hält an, dann ist variable_haltet == true und somit hält P3 nicht an.
Angenommen, P3(P3) hält nicht an, dann ist variable_haltet == false und gibt 7 zurück, hält also an.
Beide Fälle führen zum Widerspruch, also kann P3 nicht existieren. P3 existiert aber, wenn P1 existiert. Also kann auch P1 nicht existieren.

Aufgabe 2

//Turingmaschine?

$L_1 := \{a^i b^j c^k \mid i, j, k \in \mathbb{N} \wedge k = ij\} = \{a^i b^j c^{ij} \mid i, j \in \mathbb{N}\}$

//Dürfen wir hierfür auch eine Mehrband-Turing-Maschine verwenden?
Verwende 3-Band-Turing-Maschine mit Eingabeband, Band2 und Band3.

1. Kopiere die a^i auf Band2. Prüfe dabei i > 0. Spule Band2 zurück.
2. Kopiere die b^j auf Band3. Prüfe dabei j > 0. Spule Band3 zurück.
3. Lese c^i auf Eingabeband. Nutze Band2, um das i abzumessen.
4. Wenn Band2 leer, spule Band2 zurück und gehe auf Band3 eins weiter.
5. Wenn Band3 leer, muss auch das Eingabeband leer sein.

Die Bandkomplexität beträgt Wortlänge einschließlich der Blanksymbole rechts und links. $s(v) = |v| + 2$
Die Zeitkomplexität beträgt $t(v) = 2i + 2j + 2ij \leq |v|^2$

Aufgabe 3

//Read-only-Turingmaschine?

3.1: //Turingmaschine für reguläre Sprache „gerade Anzahl 1en“

Bringe zunächst am Ende der Eingabe ein Sonderzeichen für Eingabeende an, gefolgt von einem Sonderzeichen für den Schreiblesekopf.
Sodann kopiere die Eingabe nach hinten (mit der Lese-erstes-Zeichen-Methode, dabei wird die Eingabe gelöscht), lasse aber immer 1 Platz zwischen den Zeichen frei, außer beim ersten Zeichen, dafür brauchen wir 3 Plätze.
//Beispiel: 12345 ├ X2345#K1□□□2□3□4□5□#
Jetzt berechnen wir den ersten Schritt und schreiben in die erste freie Stelle die Kopfbewegung und in die Zweite den neuen Zustand.
//Beispiel: 12345 ├ X2345#K1L7□2□3□4□5□#
Wir kopieren das Ganze wieder und müssen gegebenenfalls die Bandgröße erweitern. Um zu testen, welches Zeichen bereits kopiert wurde, verwenden wir die übrigen freien Kästchen.
//Beispiel: 12345 ├ X2345#K1L7□2□3□4□5□# ├ ...#K1L7☒2☒3☒4☒5☒#K□□□□7□2□3□4□5□#
Jetzt beginnen wir wieder bei „Jetzt berechnen wir...“.

Die Turingmaschine benötigt also Sonderzeichen Γ \ Σ = {□, ☒, K, R, L, N, #}
Die Zustandsmenge wird sehr viel größer: Q x Z mit
Z = {calc, copy_isHead x Γ x Y, copy_isNotHead x Γ x Y, copy_beforeHead x Γ x Y, copy_atHead x Γ x H, ...}
Y = {content, blank}
...

3.2: //Mächtigkeit

Diese Maschine ist gleich mächtig, weil sie mit obigem Algorithmus alles berechnen kann, was auch eine normale Turingmaschine kann. Die umgekehrte Richtung ist trivial.

Aufgabe 4

//Phrasenstrukturgrammatik?

$S \rightarrow AB \mid x$
$AB \rightarrow S$

Beispiel: $S \Rightarrow AB \Rightarrow S \Rightarrow AB \Rightarrow S \Rightarrow AB \Rightarrow S \Rightarrow x$

Mit einer kontextsensitiven Grammatik ist dies nicht möglich, weil die Wortlänge einer kontextsensitiven Grammatik monoton steigend ist.