Blatt 3 w

Tuesday, May 5, 2020 1:55 PM

A3.1:

L_1, L_2 entscheidbar auf Σ.

a) Beh.: $L_1 \cup L_2$ ist entscheidbar.

Bew.: Sei M_1 (M_2) die DTM, die L_1 (L_2) entscheidet.

Sei auch $L' = L_2 \setminus L_1$.

Nach Theorem 2.3 können wir M_1 (M_2) mit einer TM U simulieren

Wir konstruieren eine TM welche für $w \in L_1$ M_1 simuliert und für $w \in L'$ M_2 simuliert. (L' statt L_2 damit $\forall w \in L_1 \cup L_2 \exists!$ TM welche w entscheidet).

Das mergen der TM könnte man wie folgt umsetzen:

Für eine Eingabe w wird o.B.d.A erst M_1 auf diese Eingabe simuliert. Das Ablehnen von M_1 wird dann umgeschrieben zu einem Zustand, der wieder an den Anfang wandert und von dort wird M_2 normal simuliert.

Damit hätten wir eine TM U konstruiert, welche alle $w \in L_1 \cup L_2$ akzeptiert und alle $w \notin L_1 \cup L_2$ verwirft. $\Rightarrow L_1 \cup L_2$ ist entscheidbar $\square$

$L_1 \cap L_2$ entscheidbar lässt man analog mit dem unterschied, dass erst die Simulation von M_1 akzeptieren muss und dann die von M_2.

Der Beweis für $\bar{L}_1 = \Sigma^* \setminus L_1$ entscheidbar folgt direkt aus der Definition 1.3 (Def. für entscheidbare Sprachen). Man muss die TM die L_1 entscheidet nur "umkehren",

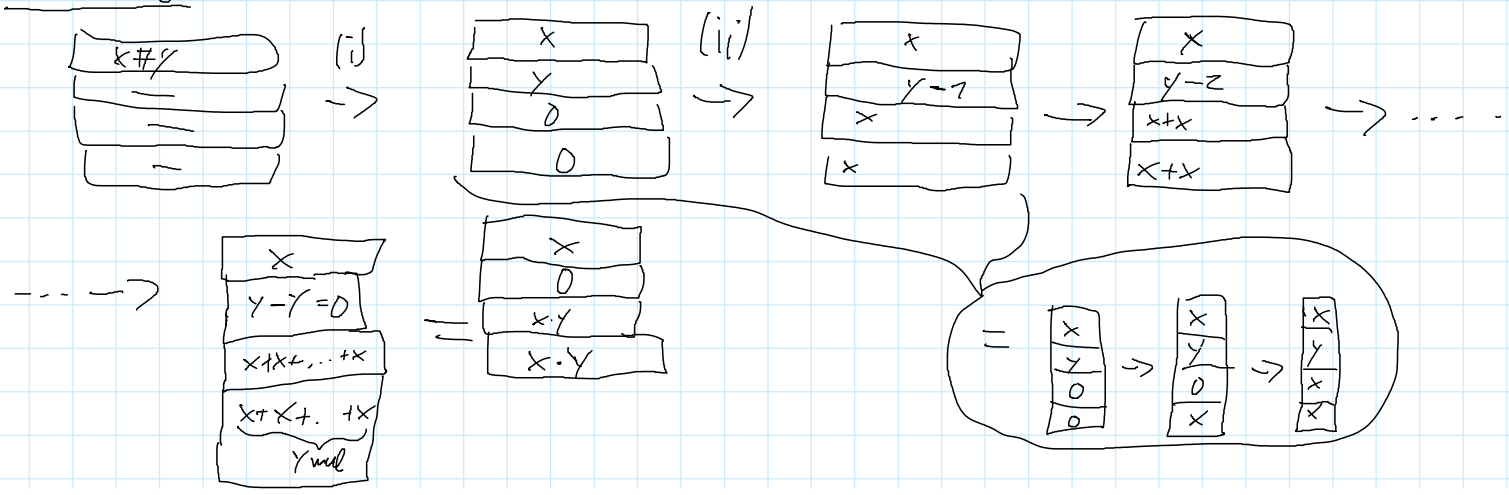
also alle Eingaben die vorher akzeptiert wurden, werden jetzt verworfen und andersrum.

b) Man kann sich auch hier, ähnlich wie beim Beweis für $L_1 \cup L_2$ entscheidbar, überlegen, dass man eine universelle TM U benutzt, um die Ausgabe von M_1 auf w_1 und M_2 auf w_2 zu berechnen. Hierbei sind M_1/M_2 die TM, welche die Sprachen L_1/L_2 entscheiden. Die Ausgabe von U ist dann die Konjunktion der Ausgaben von M_1 und M_2.

Das Problem ist es jedoch, zu bestimmen, wann w_1 endet und wann w_2 beginnt. Mit einer modifizierten Menge, welches ein Trennzeichen besitzt hätte man das Problem nicht (mir fällt jedenfalls gerade nichts ein, wie man Trennzeichen mit $L_1 \cdot L_2$ realisieren könnte, geht das irgendwie?)

A3.2: (Was ist hier mit der Wertigkeit der Bits gemeint?)

Idee Skizze:



Die TM soll folgende Schritte durchführen:

(i) Zunächst werden 0,1 im ersten Band übersprungen, bis zum #. Die darauf folgenden Zeichen werden auf das zweite Band kopiert und im ersten durch □ ersetzt (das # auch). (vielleicht wäre es besser von rechts nach links zu kopieren, damit die binären Zahlen direkt richtig untereinander stehen und mit Nullen aufgefüllt wird, damit sie die gleiche Länge haben.)
Dann werden in das dritte und vierte Band genauso viele Nullen direkt unter x und y geschrieben, s.d. alle gleich lang sind.

ii) Dann wird eine Art for-loop ausgeführt, welche wie folgt arbeitet:
Im zweiten Band wird gecheckt ob alles Nullen sind. Ist das nicht der Fall, wechselt man in einen Zustand, welcher die Zahl des zweiten Bandes um 1 verringert und danach wird wie bei schriftlichen addition untereinander die Inhalte von Band 1 und 3 addiert und in Band 4 geschrieben.
Danach wird der Inhalt von 4 nach 3 kopiert.
Das passiert so lange, bis in Band 2 nur Nullen stehen.

Alternativ, könnte man das auch mit 2 for loops machen, wo die addition so implementiert wird dass ein Band iterativ mit Einserschritten den Wert eines anderen Bands um 1 erhöht. Also in der Form:

Add(x_1, x_2)
LOOP x_2
$x_1 := x_1 + 1$
END
return x_1

Dann müsste man sich nicht überlegen wie man die schriftliche Addition umsetzt. Diese wäre aber schneller, weil sie nur einmal über die Eingabe laufen muss.

A 3.3: Maximum von n Zahlen $a_1, \dots, a_n$ finden

Eingabe: $c(i) = a_i$ für $i \in \{1, \dots, n\}$

Idee: iterieren über Eingabe und speichern das aktuell größte im Akkumulator $c(0)$. Am Ende return $c(0)$.

CODE:

1. CLOAD n
2. STORE $n+1$

$\rightsquigarrow c(n+1) = n$

3. CLOAD 1
4. STORE $n+2$

$\rightsquigarrow c(n+2) = 1$ (iterationsindex init)

5. INDLOAD $n+2$
6. STORE $n+3$

$\rightsquigarrow c(\underbrace{n+3}_{\text{"temp"}}) = c(1)$ (Zwischenspeichern für max)

7. LOAD $n+2$
8. ADD 1

$\rightsquigarrow c(ind) += 1$.

9. LOAD $n+3$

$c(0) = c(ind)$

10. IF $c(0) < c(c(n+2))$ GO TO

WTF, warum ist das so ein Konstrukt? Ich mehr...
Keine Ahnung wie man geschickt Code mit Registermaschinen schreibt...

A3.4

Im Skript steht nichts zu DFA, also habe ich mich das Skript zu LuDs von Röglin geholt und da nachgeschaut.

Ich würde vermuten, dass man keine universellen DFA konstruieren kann, weil es für den Beweis der universellen TM entscheidend war, dass wir die Informationen der zu simulierenden TM abspeichern und auslesen konnten. Der DFA kann jedoch nichtmal die Sprache $L = \{0^n 1^n \mid n \geq 1\}$ entscheiden (Theorem 3.7, LuDs Skript Röglin, 2017), also sich nicht "merken" ob die Anzahl an Nullen und Einsen gleich ist.
↳ Das ist allerdings eher meine Vermutung/Intuition und kein Beweis.