Adrian Kramkowski, Abdelhadi Fares, Yousef Al Sahli und Abdelraoof Sahli

Modul: Compilerbau

Aufgabe A2.2 – Akzeptierte Sprache, Determinismus & Darstellung

Gegeben: PDA mit folgenden Übergängen, akzeptierender Zustand `q4` :

δ(q0, a, ⊥) = (q0, A⊥)
δ(q0, a, A) = (q0, AA)
δ(q0, b, A) = (q1, BA)

δ(q1, b, B) = (q1, BB)
δ(q1, c, B) = (q2, ε)

δ(q2, c, B) = (q2, ε)
δ(q2, d, A) = (q3, ε)

δ(q3, d, A) = (q3, ε)
δ(q3, d, A) = (q3, AA) ← doppelt (Konflikt)
δ(q3, ε, ⊥) = (q4, ε)

Ist der PDA deterministisch?

Nein.

Begründung: In Zustand `q3` bei Eingabe `d` und Stacktop `A` sind *zwei* unterschiedliche Übergänge definiert:

- `δ(q3, d, A) = (q3, ε)` (A entfernen)
- `δ(q3, d, A) = (q3, AA)` (A durch zwei A ersetzen)

Damit verletzt der Automat die DPDA-Bedingung „höchstens ein Übergang je (Zustand, Eingabesymbol, Stacktop)“.

Hinweis: Der ε-Übergang `δ(q3, ε, ⊥)` widerspricht dem Determinismus *nicht*, weil es in `q3` für `⊥` keinen Eingabe-Übergang gibt. Das Problem ist

ausschließlich das doppelte `δ(q3,d,A)` .

7-Tupel des PDAs

`P = (Q, Σ, Γ, δ, q0, ⊥, F)`
Zustände: `Q = { q0, q1, q2, q3, q4 }`
Eingabealphabet: `Σ = { a, b, c, d }`
Stackalphabet: `Γ = { ⊥, A, B }`
Startzustand: `q0`
Startstapelzeichen: `⊥`
Akzeptierende Zustände: `F = { q4 }`

Automat (Beschreibung der Phasen)

Phase / Zustand	Gelesen	Stack-Aktion	Interpretation
q0 (a-Phase)	`a`	`A` pushen (auf ⊥: `A⊥` , auf `A` : `AA`)	Zähle `a` als Anzahl `A` auf dem Stack.
q0 → q1	`b` (bei `A` oben)	`B` vor `A` legen (`BA`)	Wechsel in die b-Zählphase, erste Markierung `B` oben.
q1 (b-Phase)	`b` (bei `B`)	`B` pushen → `BB`	Zähle weitere `b` über einen B-Stack.
q1 → q2	`c` (bei `B`)	`B` pop	Start der c-Phase, erstes `B` abbauen.
q2 (c-Phase)	`c` (bei `B`)	`B` pop	Alle `B` müssen durch `c` abgebaut werden.
q2 → q3	`d` (bei `A`)	`A` pop	Wechsel in die d-Phase, beginnt `A` abzubauen.
q3 (d-Phase)	`d` (bei `A`)	nicht-deterministisch: entweder `A` pop *oder* `AA` push	Pro gelesenem `d` kann die A-Menge um ±1 verändert werden.
q3 → q4	`ε` (bei `⊥`)	—	Akzeptiere im Endzustand, wenn der Stack leer ist.

Welche Sprache akzeptiert der PDA?

Wörter haben eine **Phasenform** `a^i b^j c^k d^t` mit jeweils mindestens einem Symbol in jeder Phase (weil es sonst keine passende Regel gibt): `i, j, k, t ≥ 1`.

1. **a-Phase (q0):** legt `i` mal `A` auf den Stack.
2. **b-Phase (q1):** legt `j` mal `B` oben drauf.
3. **c-Phase (q1/q2):** muss `j` mal `B` abbauen → **erzwingt** `k = j`. Sobald alle `B` weg sind, liegt oben `A`.
4. **d-Phase (q2/q3):** pro `d` darf die Anzahl der `A` um `±1` geändert werden (wegen der zwei Regeln in `q3`). Akzeptiert wird nur, wenn nach allen `d` der Stack leer ist.

Folgerung (existiert ein akzeptierender Lauf):
Aus `A^i` kann man mit `t` Schritten, die jeweils ±1 ändern, genau dann 0 machen, wenn

- `t ≥ i` (man braucht mindestens so viele Schritte wie anfängliche `A`), und
- `t ≡ i (mod 2)` (jede ±1-Änderung kippt die Parität).

Damit akzeptiert der gegebene *nicht-deterministische* PDA genau die Wörter

$$L = \{ a^i b^j c^k d^t \mid i,j,k,t \geq 1,\ k = j,\ t \geq i,\ t \equiv i \pmod 2 \}.$$

Intuitiv: Erst gleich viele `b` und `c`, dann eine `d`-Strecke, deren Länge zur anfangs eingelesenen Anzahl `a` passt (gleiche Parität und nicht kürzer).