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

Modul: Compilerbau

Aufgabe A2.1 – Deterministischer PDA für `L = { w ∈ {a,b,c}* | #a = 2·#c }`

Ziel: Entwirf einen *deterministischen* PDA (DPDA), der genau dann akzeptiert, wenn die Anzahl der `a` doppelt so groß ist wie die Anzahl der `c`. Zeichen `b` sind neutral.

Symbolische Notation:

- `#a` = Anzahl der `a` in einem Wort `w`
- `#c` = Anzahl der `c` in demselben Wort

Beispiel: Für `w = "aacbc"` gilt `#a = 2` und `#c = 2`.
Bedingung `#a = 2·#c` heißt: Es gibt **doppelt so viele** `a` wie `c`.

Intuition: Wir speichern die Differenz `D = (#a) − 2·(#c)` über Stack-Marker.

- `P` steht für „+1“, `N` für „−1“.
- Bei `a`: `D := D+1` → entweder `N` abbauen oder `P` pushen.
- Bei `c`: `D := D−2` → entweder zwei `P` abbauen (in zwei Schritten) oder zwei `N` pushen.

Akzeptanz: <u>nur</u>, wenn die gesamte Eingabe gelesen ist <u>und</u> der Stack leer ist (nur `⊥`).

Formale Spezifikation (7-Tupel)

`P = (Q, Σ, Γ, δ, q0, ⊥, F)`

Zustände: `Q = { q, qC1 }` (Basiszustand `q`, Zwischenzustand `qC1` für den 2. Abbau bei `c`)
Eingabealphabet: `Σ = { a, b, c }`
Stackalphabet: `Γ = { ⊥, P, N }`
Startzustand: `q0 = q`
Startstapelzeichen: `⊥`
Endzustände: `F = ∅` (Akzeptanz durch leeren Stack)

Übergangsfunktion δ (deterministisch)

```
Basiszustand q:

  # Eingabe 'a'  → D := D + 1
  δ(q, a, ⊥) = (q, P⊥)
  δ(q, a, P) = (q, PP)
  δ(q, a, N) = (q, ε)

  # Eingabe 'b'  → D unverändert
  δ(q, b, ⊥) = (q, ⊥)
  δ(q, b, P) = (q, P)
  δ(q, b, N) = (q, N)

  # Eingabe 'c'  → D := D − 2
  δ(q, c, ⊥) = (q, NN⊥)
  δ(q, c, N) = (q, NNN)
  δ(q, c, P) = (qC1, ε)        # 1. P abbauen; 2. Schritt folgt in qC1

Zwischenzustand qC1 (nur ε-Übergänge, keine Eingabe):

  δ(qC1, ε, P) = (q, ε)        # zweites P abbauen (es waren ≥2 P vorhanden)
  δ(qC1, ε, ⊥) = (q, N⊥)       # es war nur 1 P vorhanden → jetzt D = −1 → ein N push

(Anmerkung: P über N entsteht mit diesen Regeln nicht.)
```

Determinismus: In `q` gibt es ausschließlich eingabegesteuerte Übergänge (keine ε), in `qC1` ausschließlich ε-Übergänge (keine Eingabe). Für jede Kombination ist höchstens ein Übergang definiert.

Akzeptanz: durch *leeren Stack nach vollständiger Eingabe*.

Schritt-für-Schritt: Lauf auf `bcaba` (akzeptiert)

Schritt	Gelesen	Zustand	Stack (oben→unten)	Kommentar
0	–	q	⊥	Start
1	b	q	⊥	No-Op

Schritt	Gelesen	Zustand	Stack (oben→unten)	Kommentar
2	c	q	N, N, ⊥	0 → −2
3	a	q	N, ⊥	−2 → −1
4	b	q	N, ⊥	No-Op
5	a	q	⊥	−1 → 0
Eingabe zu Ende & Stack = ⊥ ⇒ akzeptiert				

Schritt-für-Schritt: Lauf auf `bccac` (abgelehnt)

Schritt	Gelesen	Zustand	Stack (oben→unten)	Kommentar
0	–	q	⊥	Start
1	b	q	⊥	No-Op
2	c	q	N, N, ⊥	0 → −2
3	c	q	N, N, N, N, ⊥	−2 → −4
4	a	q	N, N, N, ⊥	−4 → −3
5	c	q	N, N, N, N, N, ⊥	−3 → −5
Eingabe zu Ende & Stack ≠ ⊥ ⇒ abgelehnt				

Kurz-Begründung

Wenn #a = 2·#c: Jede Aktion aktualisiert `D` korrekt (+1 für `a`, −2 für `c`, 0 für `b`). Am Ende gilt `D=0` ⇒ Stack = `⊥` ⇒ akzeptiert.

Nur wenn Stack leer ist: Dann gilt `#a - 2·#c = 0`. Bleibt ein Marker, ist `D ≠ 0` ⇒ abgelehnt.

Determinismus: getrennte Zustände für Eingabe- und ε-Schritte; keine konkurrierenden Übergänge.