3.9. Konstruktion der Steuertabellen

3.9.1. Benötigte Tabellen

a) Für jeden Zustand muß man wissen, ob er ein akzeptierender Zustand ist oder nicht, und falls er es ist, zu welcher Regel er akzeptiert: state ↦ rule (oder −1, falls nicht akzept.)

b) Für jedes Paar (Zustand, Eingabesymbol) muß man den Folgezustand kennen (ggf. ist das der Fehlerzustand, wenn es einen solchen Übergang nicht gibt): (state, char) ↦ state.

3.9.2. Tabellenkompaktierung

Die Zustandsübergangstabelle ist

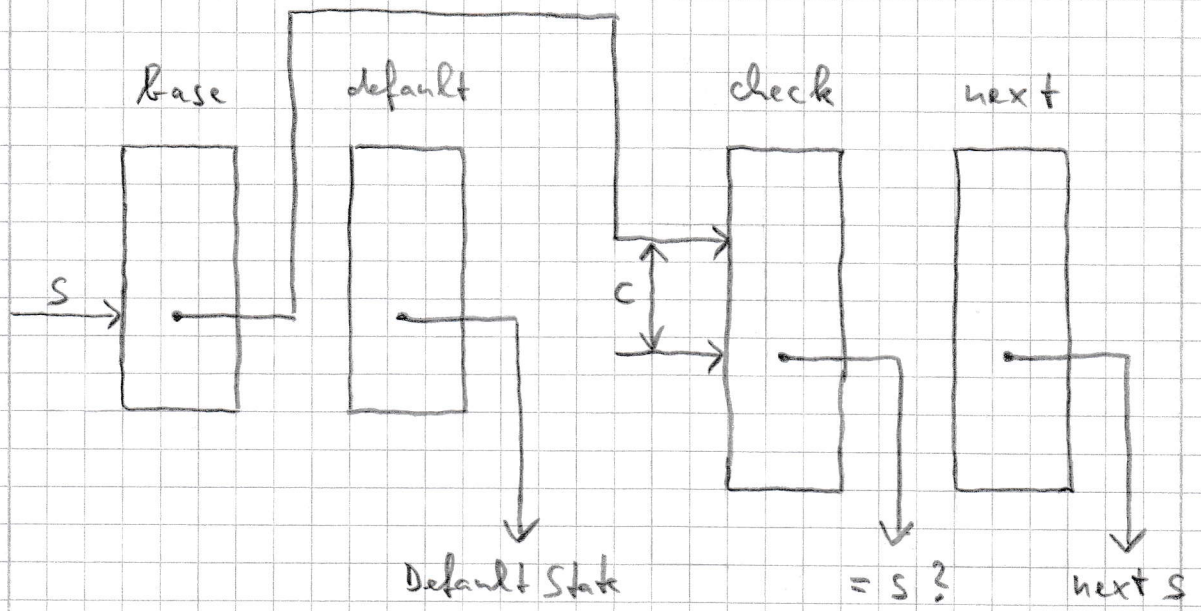
- schwach besetzt und
- ihre Zeilen weisen häufig große Übereinstimmungen auf.

Beide Eigenschaften kann man zur Kompaktierung ausnutzen.

a) Die "leeren" Plätze für einen best. Zustand (d.h. Übergänge in den Fehlerzustand) werden für Übergänge aus anderen Zuständen genutzt. Dann muß es aber eine Möglichkeit geben, diese Situation zu erkennen.

b) Man notiert für einen Zustand, der große Ähnlichkeit mit einem anderen Zustand hat, diesen als "Default-Zustand", und vermerkt für den Zustand nur die Unterschiede zu seinem Default-Zustand. Findet man keinen Übergang in den Unterschieden, fällt man auf den Default-Zust. zurück.

Realisiert werden kann das durch 4 Tabellen, von denen jeweils zwei durch denselben Index angesteuert werden:



Benutzung:

```
int nextstate (int s, char c) {
    int l = base[s] + c;
    if (check[l] == s) return next[l];
    return nextstate (default[s], c);
}
```

Bem.: Spezielle Werte (z.B. −1) können benutzt werden, um besondere Situationen zu signalisieren, z.B. "Zustand ist gleich dem Default".

3.9.3. Tabellenkonstruktion

Problem "Finde kleinste Tabelle" ist NP-vollständig → Heuristiken!

Beispiel:

```
for every state s = 0.. numStates-1 do
    find a default state d < s with minimal number of differences
    find a place for the differing entries: this is the smallest
        index in check which avoids any clash of entries
    add entries to check and next, remember base and default
end for
```

3.9.4. Interaktive Eingabe

Es gibt ein Problem bei interaktiver Eingabe: Da der Automat weiterlaufen soll, bis er zu einem Paar (state, input) kommt, für das es keinen Übergang gibt (Grund: die längste Übereinstimmung mit einem regulären Ausdruck soll gefunden werden!), wird der Automat ein Zeichen nach dem Zeilenumbruch lesen wollen, selbst wenn er diesen als Token akzeptieren soll und es keinen Zustandsübergang aus dem betr. Zustand heraus gibt. Folge: man muß nach <ret> noch und <ret> drücken, damit das erste <ret> erkannt wird. Das ist nicht akzeptabel. Beachte aber: Falls es irgend einen Übergang aus dem Zustand heraus gibt, ist das Verhalten richtig und notwendig!

Abhilfe: Man markiert diejenigen Zustände, die gar keine Übergänge besitzen (z.B. in dem man base[] und default[] für solche Zustände auf −1 setzt). Dann kann der Treiber schon vor dem Lesen des nächsten Eingabezeichens erkennen, daß (egal welches Zeichen gelesen würde) kein Zustandsübergang möglich ist und kann das Lesen des Zeichens unterdrücken.

Bem.: Das Verhalten des Automaten wird unter keinen Umständen falsch. Er wird sogar etwas schneller laufen, da das Stoppen in manchen Fällen vor dem Zugriff auf check[] erkannt wird.