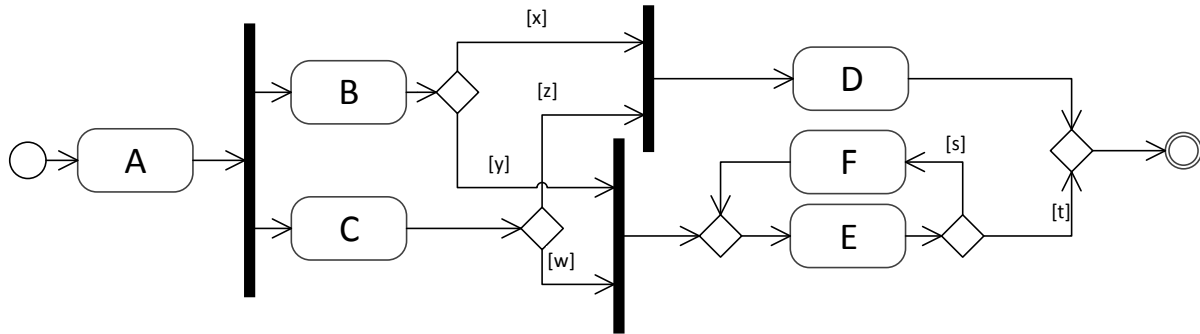


4. Übung – Überprüfung und Testen von Modellen

1 Statische Analyse des Prozesses

Überprüfen Sie das folgende Prozessmodell.



- Unter welchen Bedingungen ist der Prozess vollständig spezifiziert?
- Unter welchen Bedingungen ist der Prozess auch deterministisch?
- Unter welchen Bedingungen ist der Prozess verklemmungsfrei?
- Unter welchen weiteren Bedingungen wird der Prozess sicher terminieren?
- Ist der Prozess wohlstrukturiert? Falls nicht, dann wie sollte er dazu geändert werden? Lösen diese Änderungen die Probleme in den früheren Teilaufgaben auf?

2 Dynamische Analyse mit Testen

Wir haben die folgende *Anforderungen* gegenüber der Funktion $f()$:

R1 Die Funktion $f()$ soll bei jedem Ablauf mindestens einmal etwas ausgeben.

R2 Die Funktion $f()$ soll für beliebige Eingabereihen terminieren.

R3 Bei dem Ablauf der Funktion $f()$ soll der zuletzt ausgegebene Wert null sein.

Eine mögliche Realisation der Funktion gibt das folgende C-Kodefragment an:

```

1 int readInput();
2 void writeOutput(int out);
3
4 void f() {
5     int x = readInput();
6     int y = readInput();
7     int z = x + y;
8     writeOutput(x * y);
9     while (x > 0 && y > 0) {
10        if (1 == readInput() % 2) {
11            y--;
12            z--;
13        } else {
14            x--;
15            y++;
16        }
17        writeOutput(z + x * y * y - x - y);
18    }
19 }
```

Überprüfen Sie die Funktionalität der Funktion durch die folgenden Schritte!

- Stellen Sie den Kontrollfluss von $f()$ als ein Prozessmodell dar!
- Warum können wir sicher sein, dass R1 erfüllt wird?
- Warum können wir sicher sein, dass R2 erfüllt wird?
- (Optionale Aufgabe) Entwickeln Sie eine Zustandsmaschine, die äquivalent mit der Funktion $f()$ funktioniert. Modellieren Sie die Aufrufe `readInput()` als ein Eingabekanal, bzw. den Aufruf `writeOutput()` als ein Ausgabekanal. Modellieren Sie die Terminierung der Funktion $f()$ so, dass der Automat ein spezielles Zeichen ausgibt, und in einen Fallenzustand (in einen Zustand ohne ausgehende Übergänge) übergeht.

- e. Die Eingabesequenz $t_1 = \langle 2, 3, 5, 7, 11, 13, \dots \rangle$ dient als Testfall für die Anforderung R3. Detektieren wir damit einen Fehler?
- f. Berechnen Sie die *Instruktionsabdeckung* der Testfall (auf dem Prozessmodell), also dass wie großes Anteil der Instruktionen der getesteten Funktion während der Ausführung des Testfalles berührt wird! Wie erscheint diese Messzahl am Steuerungsprozess?
- g. Die Eingabesequenz $t_2 = \langle 1, 2, 4, 1, 2, 4, \dots \rangle$ ist unser zweiter Testfall zu der Anforderung R3. Detektiert dieser Testfall einen Fehler? Wie groß ist die gemeinsame Instruktionsabdeckung des aus den zwei Testfällen bestehenden Testsatzes?
- h. (Optionale Aufgabe) Was für eine Abdeckungsmetrik kann man aus der früher gebauten Zustandsmaschine berechnen?
- i. Für das Testen von R3 wird ein *Testorakel* gebaut. Entwickeln sie die Zustandsmaschine dieses Orakels, welches durch Beobachtung der Ein- und Ausgaben und die Terminierung von $f()$ entscheiden kann, ob die Anforderung während des gegebenen Ablaufs verletzt wurde. Wie benimmt sich das Orakel für die obige Testeingabe?
- j. Geben Sie einen Testfall an, der einen Fehler in dem Programm erkennt. Aufgrund welches Prinzips hätten Sie vermuten können, dass der frühere Testsatz ergänzt werden muss?
- k. (Optionale Aufgabe) Nehmen Sie die folgenden Eingabesequenzen als weitere Testfälle zu Ihrem Testsatz zu: $t_3 = \langle 0, 1, 2, 3, 4, 5, \dots \rangle$ und $t_4 = \langle 1, 2, 3, 4, 5, 6, \dots \rangle$. Können wir mit denen einen Fehler detektieren? Wie verändern sich die Abdeckungszahlen?
- l. (Optionale Aufgabe) Bestimmen Sie, bei welchen Eingabesequenzen R3 verletzt wird, und empfehlen Sie eine Fehlerkorrektur!