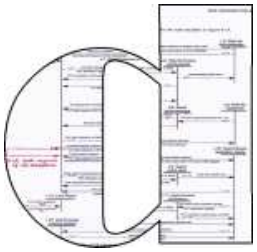
DEPENDABLE SYSTEMS AND SOFTWARE

Fachrichtung 6.2 — Informatik
Prof. Dr.-Ing. Holger Hermanns
Dipl.-Inform. Lijun Zhang



Übungsblatt 14 (Programmierung I)

Lesen Sie im Skript: Kapitel 13, 19

Für dieses Übungsblatt legen wir die in den Kapiteln 18 und 19 eingeführte Sprache CCS zugrunde. Eine fast vollständige Modell-Implementierung der Sprach L aus Kapitel 18 finden Sie im Netz. Ihre Aufgabe ist es, daraus eine Implementierung von CCS zu erstellen.

Die Übungen dieser Woche haben zum großen Teil einen deutlich anderen Charakter als bisher. Erstmals konfrontieren wir Sie mit einem größeren Programm (etwa 200 Zeilen), das Sie verstehen und erweitern sollen (wesentliche Teile des Programms sind im Skript erklärt). Die wesentlichen Stellen, an denen Sie eingreifen müssen, sind durch `TODO` markiert. Wir erwarten von Ihnen, dass Sie das Programm im Detail verstehen und dessen Teile auf einem Interpreter austesten. Die Arbeit mit dem Interpreter ist wesentlicher Bestandteil der Übung.

Für die Klausur setzen wir voraus, dass Sie mit der Semantik von CCS und deren Modellimplementierung vertraut sind.

Aufgabe 14.1 Vervollständigen Sie den fehlenden Teile der Implementierung von `steps` für L, also die Behandlung von `Pre`, und fügen Sie diese in den Code ein.

Aufgabe 14.2 Compilieren Sie das Programm zu einem ausführbaren Binary. Unter *Windows* geschieht dies mit `'mosmlc <file.sml> -o <binary.exe>'` . Unter *linux* geht dies anders, siehe dazu die ersten Zeilen des Codes. Testen Sie das Tool auf dem *Tempomat*, wie Sie ihn auf dem Netz finden.

Aufgabe 14.3 Vervollständigen Sie die Prozedur `complement` aus Abbildung 19.6, und fügen Sie diese in den Code ein. Hinweis: Betrachten Sie auch die Verwendung der Prozedur, um sie korrekt zu implementieren.

Aufgabe 14.4 Implementieren Sie die Prozedur einer Funktion $mapPartial : (A \rightharpoonup B) \to \mathcal{L}(A) \to \mathcal{L}(B)$, die zu einer *partiellen* Funktion $f : A \rightharpoonup B$ und einer Liste $xs \in \mathcal{L}(A)$ eine Liste $ys \in \mathcal{L}(B)$ liefert, so dass ein Element $x' \in B$ genau dann in ys

ist, wenn es in xs ein Element $x \in A$ gibt, so dass $f(x) = x'$ gilt. Hinweis: Eine partielle Funktion können Sie durch eine Prozedur realisieren, die eine Ausnahme genau dann wirft, wenn die Funktion für den entsprechenden Parameter undefiniert ist.

Aufgabe 14.5 Vervollständigen Sie die Prozedur `successors` aus Abbildung 19.6, und fügen Sie diese in den Code ein. Sie liefert zu einer Markierung m und einer Liste xs von Paaren aus $mark * ccs$ eine maximale Liste von Termen $P \in ccs$ findet, so dass (m, P) in xs enthalten ist. Hinweis: Sie können die Prozedur `mapPartial` aus Aufgabe 14.4 verwenden.

Aufgabe 14.6 Erweitern Sie den Datentyp `lterm` zu einem Typ `ccsterm`, welcher *CCS* darstellt.

Aufgabe 14.7 Vervollständigen Sie die fehlenden Teile der Implementierung von `steps`, also die Behandlung '|' und '\'.

Aufgabe 14.8 Nun zur konkreten Syntax: Führen Sie neue Token für die zusätzlichen Operatoren ein. Erweitern Sie die Prozedur `lex` entsprechend.

Aufgabe 14.9 Erweiteren Sie den Parser der Sprache, und beachten Sie dabei die Klammersparregeln aus Abschnitt 19.3. Sie können sich an der Prozedur `chcexp` orientieren, benötigen aber Verständnis von Abschnitt 13.5.

Erweitern Sie den Code jetzt so, dass auch Restriktion geparst werden kann. Sie können dies tun, indem sie die Prozedur `atexp` so erweitern, dass Sie nach geklammerten Ausdrücken (...) testen, ob eine Restriktion folgt. Benutzen Sie die Hilfsprozeduren `setexp`, um die Restriktionsmenge zu parsen. Sie müssen diese Prozedur mit einer Hilfsprozedur versorgen. Diese ist bereits deklariert. Orientieren Sie sich an der Verwendung von `setexp` in der Prozedur `procexp`.

Aufgabe 14.10 Es gibt noch drei Stellen im Programm, die angepasst werden müssen! Passen Sie die Prozeduren `lequation, procLexp` und `parse` an `ccsterm` bzw. an unsere Sprache *CCS* an! Ausserdem sollten Sie noch den *Pretty-printer* um die neuen Operatoren erweitern.

Aufgabe 14.11 Kompilieren und testen Sie das Ergebnis. Wenn Sie alles richtig gemacht haben, sollte Ihr Werkzeug zum Beispiel in der Lage sein, die *Dining Philosophers* (siehe Netz) zu verarbeiten – es sei denn Sie haben eine andere konkrete Syntax gewählt. Klappt? Gut gemacht!

Aufgabe 14.12 Die Prozedur `steps` terminiert nicht für alle *CCS*-Terme, und es gibt dafür drei verschiedene Gründe. Um diese Gründe kennenzulernen, untersuchen Sie die Adjazenzmenge von X für die folgenden drei Umgebungen: $\Gamma = \{(X, X)\}$, $\Gamma = \{(X, a!.0 + X)\}, \Gamma = \{(X, a!.0 \mid X)\}$. Die Gründe dafür sind verschieden, aber die gemeinsame Ursache ist, dass in jedem Fall die strukturelle Rekursion direkt von X auf X zurückführt. Dies können Sie vermeiden, indem Sie festlegen, daß jede Rekursionsvariable erst durch mindestens eine Transition erreicht wird. Syntaktisch bedeutet dies, daß Sie in der abstrakten Grammatik statt mit X mit $m.X$ arbeiten. Wie muss man Ihr Werkzeug abändern? Damit terminiert die Prozedur `steps` für alle Terme der Sprache. Sind Sie furchtlos genug, um dies zu beweisen? Was ist eine natürliche Terminierungsfunktion?

Aufgabe 14.13 Repräsentieren Sie die Terme $a!.0$, $a!.(0+a!.0)$, $a!.0+a!.0$, $a!.a!.a?.a!.a!.0$, $a!.0 + (b?.0)$ und $a!.0 + a!.b?.X$ in SML gemäß der Typdefinitionen aus Abbildung 19.2. Sie können dies auch von Ihrem Werkzeug erledigen lassen.

Aufgabe 14.14 Geben Sie die Adjazenzfunktion zum Graph des folgenden Prozesses an:

$$Reach([\![\ a!.c?.0|(a?.0 + c!.0) + a!.0\ ]\!])$$

Aufgabe 14.15 Geben Sie eine oder mehrere Gleichungen Ihrer Wahl für *SOUND'* an, so daß sich der Prozess $(BUTTON \mid IDLE \mid SOUND') \setminus \{$ *on!,on?,off!,off?,ok!,ok?* $\}$ verklemmen *muß*.

Aufgabe 14.16 Ersetzen Sie in der definierenden Gleichung des *Controller*s, die Gleichung für *ON* durch *ON = ok!.off?. IDLE +ok!.brake?. SUSPEND*. Sind die beiden Prozesse *IDLE* (vor und nach dem Ersetzen) spuräquivalent? Und wie steht es mit

$$(BUTTON \mid IDLE \mid SOUND) \setminus \{on!,on?,off!,off?,ok!,ok?\}$$

Können Sie in einem der beiden Varianten dieses Systemes eine Verklemmung beobachten?

Aufgabe 14.17 Zeigen Sie, daß die Relation $\sim = \{(x, y) \in \mathbb{Z}^2 \mid |x| = |y|\}$ eine Äquivalenzrelation auf $\mathbb{Z}$ ist.

Aufgabe 14.18 Ist $Id(CCS)$ eine Kongruenz auf CCS? Und wie steht es mit $CCS \times CCS$?

Aufgabe 14.19 Zeigen Sie

- $0 \sim 0 + 0$,
- $a.0 \sim a.0 + a.0$,
- $a.0 \sim a.(0 + 0) + a.0$.
- $a.(b.0 + c.0) + a.(b.0 + c.0) \sim a.(b.0 + c.0) + a.(c.0 + b.0)$.

Wie sieht es mit $a.(b.0+c.0)$, $a.b.0+a.c.0$ und $a.(b.0+c.0)+a.b.0+a.c.0$ aus? Untersuchen Sie paarweise. Gilt $a.\tau.\tau.b.0 \sim a.\tau.b.0$?

Aufgabe 14.20 Betrachten Sie die Prozesse $X = a.a.a.X$, $X = a.a.X$ und $X = a.(a.(X + 0) + a.X)$, $X = a.(a.0 + a.X)$. Welche dieser Prozesse sind bisimilar?

Aufgabe 14.21 Beweisen Sie, dass $\sim$ verträglich mit dem Auswahloperator ist, also, daß $P \sim Q$ auch $P + R \sim Q + R$ impliziert.