UNIVERSITÄT KARLSRUHE (TH)
Institut für Logik, Komplexität und Deduktionssysteme
Prof. Dr. P. H. Schmitt
Dr. B. Beckert

Formale Systeme, WS 2000/2001

Übungsblatt 9

Diese Aufgaben werden in der Übung am 14.02.01 besprochen.

Aufgabe 1

Die Semantik des *until*-Operators ist in der linearen temporalen Logik definiert durch:

$$s \models A\ \mathit{until}\ B \quad \text{gdw.} \quad \text{es gibt } s' > s \text{ mit } t \models A \text{ für alle } s < t < s' \text{ und } s' \models B$$

Die Semantik der alternativen Operatoren until_1 und until_2 sei nun definiert durch:

$$s \models A\ \mathit{until}_1\ B \quad \text{gdw.} \quad \text{(1) es gibt } s' > s \text{ mit } t \models A \text{ für alle } s < t < s' \text{ und } s' \models B, \text{ oder (2) für alle } s' > s \text{ gilt } s' \models A$$

$$s \models A\ \mathit{until}_2\ B \quad \text{gdw.} \quad \text{es gibt } s' \geq s \text{ mit } t \models A \text{ für alle } s \leq t < s' \text{ und } s' \models B$$

(a) Beschreiben Sie in eigenen Worten, was der Unterschied der Semantik von until_1 bzw. until_2 im Vergleich zu der von *until* ist.

(b) Zeigen Sie, daß *until* durch until_1 ausgedrückt werden kann und daß until_1 durch *until* ausgedrückt werden kann.

(c) Zusätzlich sei nun die Semantik des einstelligen Operators *next* definiert durch:

$$s \models \mathit{next}\ A \quad \text{gdw.} \quad s' \models A \text{ für } s' \text{ mit: } s < s' \text{ und es gibt kein } t \text{ mit } s < t < s'$$

Zeigen Sie, daß *until* durch (zusammen) until_2 und *next* ausgedrückt werden kann und daß until_2 und *next* (jeweils) durch *until* ausgedrückt werden können.

Hinweis: Um zu zeigen, daß ein logischer Operator *op* durch einen anderen Operator *op'* ausgedrückt werden kann, muß man eine zu $A\ op\ B$ äquivalente Formel angeben, die – neben A, B und $\wedge, \vee, \neg, \mathit{true}, \mathit{false}$ – nur den Operator op' (und nicht op) enthält.

Aufgabe 2

Das folgende Programm sei mit α_{square} bezeichnet:

$$\begin{array}{l} \underline{\textbf{while}}\ i < x\ \underline{\textbf{do}} \\ \quad y := s(y + s(s(0)) * i); \\ \quad i := s(i) \\ \underline{\textbf{end}} \end{array}$$

Außerdem sei *Nat* eine DL-Formel, die die natürlichen Zahlen mit Addition, Multiplikation und den Ordnungen $<$ bzw. $\leq$ bis auf Isomorphie eindeutig charakterisiert, wobei die Funktionssymbole 0, s (Nachfolger), $+$ und $*$ wie in der Arithmetik üblich interpretiert werden.

Dann ist die Formel

$$(Nat \wedge i \doteq 0 \wedge y \doteq 0) \rightarrow [\alpha_{\text{square}}]\, y \doteq x * x \quad ,$$

sie sei mit ϕ bezeichnet, eine allgemeingültige DL-Formel.

Um zu beweisen, daß eine DL-Formel wie ϕ allgemeingültig ist, ist folgende Regel sehr hilfreich: Aus der Allgemeingültigkeit von

- $(Nat \wedge \gamma) \rightarrow inv$
- $(Nat \wedge inv \wedge \epsilon) \rightarrow [\alpha]\, inv$
- $(Nat \wedge inv \wedge \neg\epsilon) \rightarrow \phi$

folgt die Allgemeingültigkeit von

$$Nat \wedge \gamma \rightarrow [\underline{\textbf{while}}\ \epsilon\ \underline{\textbf{do}}\ \alpha\ \underline{\textbf{end}}]\, \phi$$

Diese Regel ist für alle Formeln γ, ϕ, inv, Programme α und Boolesche Ausdrücke ϵ korrekt.

(a) Was sagt die Formel ϕ aus? Geben Sie eine natürlichsprachliche Übersetzung an (d.h., in (Informatik-)Umgangssprache).

(b) Geben Sie eine geeignete Invariante inv an, so daß die obige Regel zur Ableitung der Formel ϕ eingesetzt werden kann.

Hinweis: Beachten Sie, daß die Invariante so gewählt werden muß, daß tatsächlich *alle* drei Prämissen der Regel erfüllt sind.

(c) Begründen Sie, warum dabei die zweite Prämisse der Regel, also

$$(Nat \wedge inv \wedge i < x) \rightarrow [y := s(y + s(s(0)) * i);\ i := s(i)]\, inv \quad ,$$

für Ihre Invariante inv gilt.

Aufgabe 3

Gegeben Sei das aus der Vorlesung bekannte UML-Klassendiagramm, daß ein „Loyalty-Programm“ modelliert.

(a) Geben Sie Übersetzungen der folgenden OCL-Constraints in natürliche Sprache an.

(i) <u>`LoyaltyProgram`</u>
`self.customer->size < 10000`

(ii) <u>`LoyaltyAccount`</u>
`transaction->collect( points )->exists( p :  Integer | p=500 )`

(iii) <u>`LoyaltyProgram::cancel(c :  Customer)`</u>
`post :  customer = customer@pre->excluding(c)`

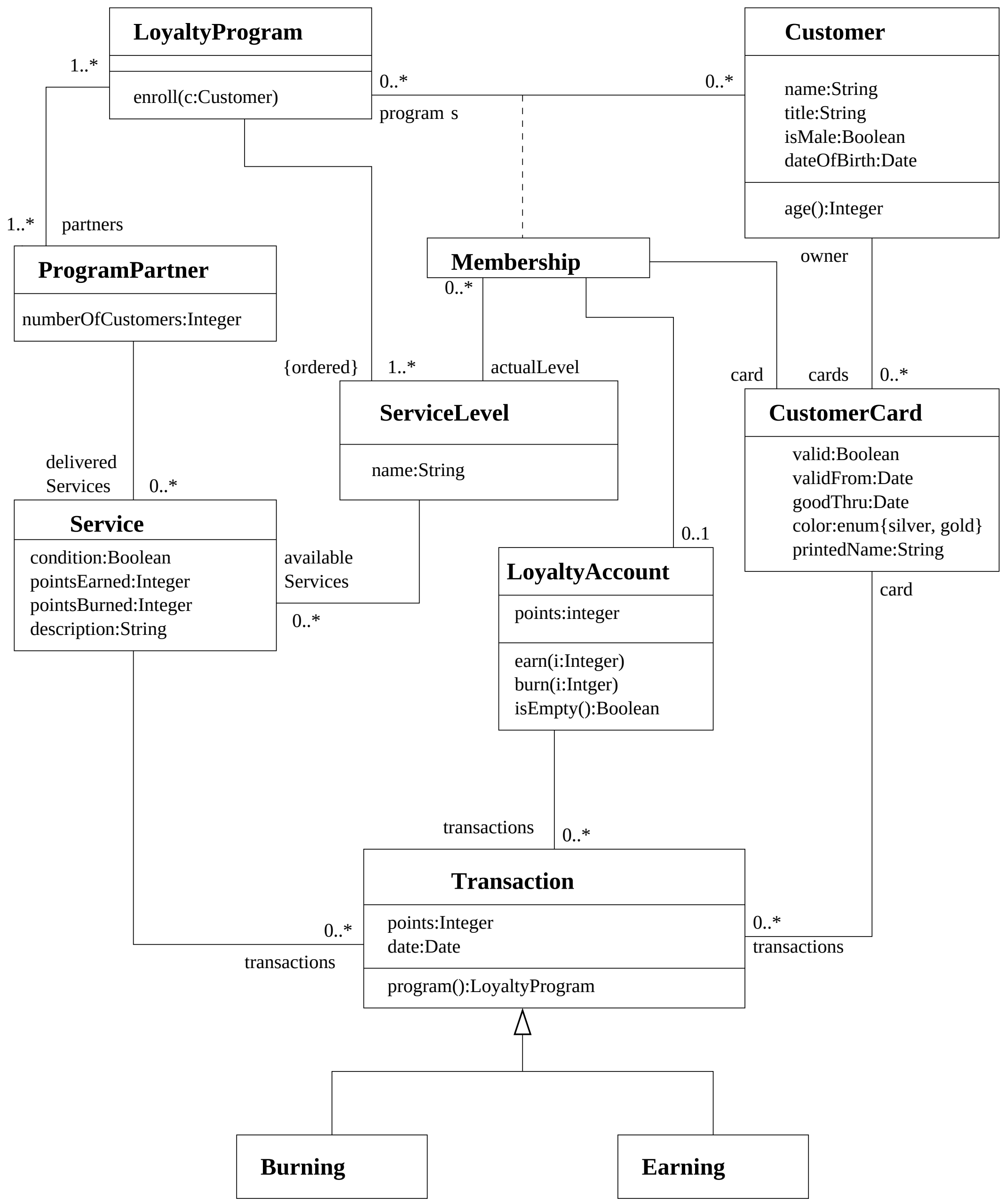
(b) Geben Sie OCL-Constraints an, die folgende Sachverhalte ausdrücken:

(i) Der mit einer Karte assoziierte Kunde ist mit dem Kunden identisch, der mit derjenigen Mitgliedschaft assoziiert ist, welche mit der Karte assoziiert ist.

(ii) Verschiedene Kunden eines Loyalty-Programms haben verschiedene Karten.

(iii) Wenn es mehr als 0 Punkte in einem Loyalty-Account gibt, dann muß eine Transaktion mit dem Account assoziiert sein, deren Punktwert größer als 0 ist.

(iv) Die Summe der Punktwerte derjenigen mit einem Loyalty-Account assoziierten Transaktionen, die einen Punktwert kleiner als 10 haben, ist kleiner als 500.



LoyaltyProgram
enroll(c:Customer)
Customer
name:String
title:String
isMale:Boolean
dateOfBirth:Date
age():Integer
1..*
0..*
0..*
program s
1..*
partners
ProgramPartner
numberOfCustomers:Integer
Membership
0..*
owner
{ordered}
1..*
actualLevel
card
cards
0..*
ServiceLevel
name:String
CustomerCard
valid:Boolean
validFrom:Date
goodThru:Date
color:enum{silver, gold}
printedName:String
delivered
Services
0..*
Service
condition:Boolean
pointsEarned:Integer
pointsBurned:Integer
description:String
available
Services
0..*
0..1
LoyaltyAccount
points:integer
earn(i:Integer)
burn(i:Intger)
isEmpty():Boolean
card
transactions
0..*
Transaction
points:Integer
date:Date
program():LoyaltyProgram
0..*
transactions
0..*
transactions
Burning
Earning