Technische Universität München
Institut für Informatik
Prof. Manfred Broy
A. Gruler, B. Spanfelner

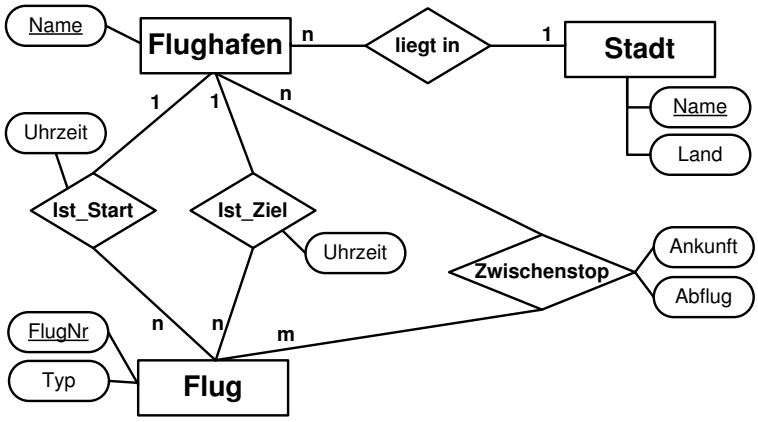
WS 2006/07

20. November 2006
Musterlösung zu Blatt 4

Übungen zu „Formale Methoden“

Aufgabe 1 E/R-Modellierung und OO-Datenmodellierung [LÖSUNG]

(a) Das folgende E/R-Diagramm liefert eine mögliche Beschreibung des geforderten Ankunftsystems. Eine Stadt kann dabei mehrere Flughäfen besitzen. Ein Flug hat genau einen Start- und Zielflughafen (dargestellt durch die “Ist_Start“ und “Ist_Ziel“-Beziehungen). Die Beziehung “Zwischenstop“ legt die Flughäfen fest, auf denen ein Flug vor der Ankunft an seinem Zielflughafen jeweils einen Zwischenstop macht. Ein Flughafen kann dabei das Zwischenstopziel mehrerer Flüge sein. Somit sind alle durch einen Flug verbundenen Flughäfen bestimmbar. Desweiteren lassen sich durch das Attribut “Typ“ der Entität “Flug“ nationale bzw. internationale Flüge unterscheiden.



(b) Das gewünschte Auskunftsmodell lässt sich auch vollständig mittels algebraischer Spezifikationen beschreiben. Insbesondere läßt sich das angegebene E/R-Diagramm als eine „Kurzschreibweise“ einer algebraischen Spezifikation auffassen. Die Entitäten “Flug“, “Stadt“ und “Flughafen“ beschreiben gleichermaßen eine Sorte, eine Entitästmenge und deren Aufbau (Sorte der Entitätsmenge), wie in der nachfolgenden Tabelle dargestellt.

Entität	Sorte	Entitätsmenge	Sorte der Entitätsmenge
Stadt	Stadt	stadt	Fset Stadt
Flughafen	Flughafen	flughafen	Fset Flughafen
Flug	Flug	flug	Fset Flug

Sämtliche Entitäten lassen sich als Rekord-Sorten modellieren. Für die Entität "Stadt" hieße die Sortenspezifikation z.B.:

$$\textbf{sort}\ \mathsf{Stadt} = \mathsf{stadt}(\mathsf{name}\colon \mathsf{Seq\ Char},\ \mathsf{land}\colon \mathsf{Seq\ Char})$$

Relationship-Beziehung lassen sich durch 2-stellige mathematische Relationen zwischen den Entitäten ausdrücken. Dazu führen wir z.B. eine Relation $\mathsf{Rel}(E_1, E_2)$ ein, die die Menge der endlichen Relationen über den Sorten E_1 und E_2 beschreibt. Formal beschreibt Rel also eine Sorte der Art:

$$\begin{array}{rl} & \textbf{sort}\ \mathsf{Rel}(\alpha, \beta) = \mathsf{Fset\ Instance}\ \alpha\ \beta \\ \text{mit} & \textbf{sort}\ \mathsf{Instance}\ \alpha\ \beta = \mathsf{paar}(sel_1 :\ \alpha,\ \ sel_2 :\ \beta) \end{array}$$

Die Relationship liegt_in zwischen den beiden Entitäten Flughafen und Stadt entspricht somit den folgenden Bestandteilen einer Signatur:

flugh:	Set Flughafen
s:	Set Stadt
liegt_in:	Rel(Flughafen, Stadt)

Primärschlüssel können realisiert werden, daß in jeder Spezifikation einer Entität ein Attribut *pkey* zusammen mit einem entsprechenden Axiom aufgenommen werden. Für die Relationship liegt_in heißt das entsprechende Axiom:

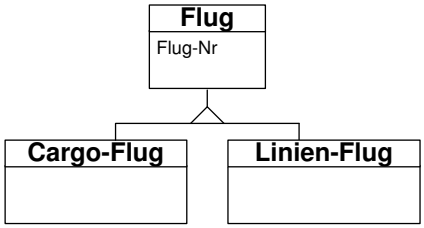
$$\Big(s_1 \in \mathsf{Stadt}\ \wedge\ s_2 \in \mathsf{Stadt}\ \wedge\ \big(pkey(s_1) = pkey(s_2)\big)\Big)\ \rightarrow\ s_1 = s_2$$

Im Unterschied zur E/R-Diagrammen erlauben Spezifikationen durch die Angabe von beliebigen Axiomen eine detailliertere Beschreibung von Datenmodellen. Es läßt sich in einem E/R-Diagramm z.B. nicht angeben, daß jede Stadt genau 3 Flughäfen besitzt. Eine solche Eigenschaft läßt sich mit einfachen Axiomen in einer algebraischen Spezifikation sehr leicht ausdrücken.

(c) Für das obige E/R-Diagramm kann ein geeignetes OO-Diagramm angegeben werden. Statt Entitätsmengen sprechen wir hier von *Klassen*. Der Entwurf eines OO–Programmsystems erfolgt durch die Festlegung seiner Klassen, ihrer Beziehungen untereinander und ihrer Schnittstellen unter Verwendung einer Reihe von Standardrelationen. Typischerweise sind dies

- is_a (ist_ein): Vererbung, Teilklassenbildung,
- is_part_of: Bestandteilsrelation, Aggregation,
- clientship, uses, knows: Verwendungsrelation, Assoziation.

Wir illustrieren diesen Sachverhalt am Beispiel von Flug. Nehmen wir an, wir wollen neben *Flüge* auch *Cargo-Flüge* und *Linien-Flüge* berücksichtigen. Dann könnten wir folgendes Teildiagramm verwenden, um auszudrücken, dass sowohl *Cargo-Flüge* als auch *Linien-Flüge* jeweils *Flüge* sind (Vererbungsbeziehung).



Aufgabe 2 Fehlermodellierung [LÖSUNG]

Zur Behandlung von Fehlerfällen haben wir mehrere Möglichkeiten. Die folgenden Lösungsvorschläge verwenden eine einfache Totalisierung, ein universelles Fehlerelement $\perp$ bzw. die Einführung von sortenspezifischen Fehlerelementen.

SPEC BTREE_F1_F2 =

$\{$ **based_on** BOOL,

sort BTree α,

etree : BTree α,
$\langle\_\rangle$: $\alpha \rightarrow$ BTree α,
conc : BTree α, BTree $\alpha \rightarrow$ BTreeα,
left, right : BTree $\alpha \rightarrow$ BTreeα,
isetree : BTree $\alpha \rightarrow$ Bool,

BTree α **generated_by** etree, $\langle\_\rangle$, conc,

isetree(etree) = true,
isetree($\langle a \rangle$) = false,
isetree(conc(t_1, t_2)) = isetree(t_1) $\wedge$ isetree(t_2),

left(conc(t_1, t_2)) = t_1,
right(conc(t_1, t_2)) = t_2,

left(etree) = etree, left(etree) = $\perp$,
oder
right(etree) = etree right(etree) = $\perp$ $\}$

SPEC BTREE_F3 =

$\{$ **based_on** BOOL,

sort BTree α,

error : BTree α,
etree : BTree α,
$\langle\_\rangle$: $\alpha \rightarrow$ BTree α,
conc : BTree α, BTree $\alpha \rightarrow$ BTreeα,
left, right : BTree $\alpha \rightarrow$ BTreeα,
isetree : BTree $\alpha \rightarrow$ Bool,

BTree α **generated_by** etree, $\langle\_\rangle$, conc, error

isetree(etree) = true,
isetree($\langle a \rangle$) = false,
isetree(conc(t_1, t_2)) = isetree(t_1) $\wedge$ isetree(t_2),

left(conc(t_1, t_2)) = t_1,
right(conc(t_1, t_2)) = t_2,

isetree(error) = false,
left(etree) = error,
right(etree) = error $\}$

Aufgabe 3 **Algebraische Spezifikation** [**LÖSUNG**]

SPEC DIARY $= \{$

based_on DATE, SET,
sort Diary,

emptyd : Diary ,
isfreed : Diary, Date $\longrightarrow$ Bool ,
putd : Diary, Date $\longrightarrow$ Diary ,
getd : Diary, Date $\longrightarrow$ Set Date ,
deld : Diary, Date $\longrightarrow$ Diary ,

Diary **generated_by** emptyd, putd ,

$(A1)$ isfreed(emptyd, d) $= true$,

$(A2)$ isfreed(y, d) $\Rightarrow$
isfreed(putd(y, d), e) $=$ ($\neg$ conflict(d, e) $\wedge$ isfreed(y, e)) ,

$(A3)$ isfreed(y, d) $\Rightarrow$ getd(y, d) $= \{\ \}$,

$(A4)$ isfreed(y, d) $\Rightarrow$ getd(putd(y, d), d) $= \{d\}$,

$(A5)$ conflict(d, e) $\wedge$ isfreed(y, d) $\Rightarrow$
getd(putd(y, d), e) $= \{d\} \cup$ getd(y, e) ,

$(A6)$ $\neg$ conflict(d, e) $\wedge$ isfreed(y, d) $\Rightarrow$
getd(putd(y, d), e) $=$ getd(y, e) ,

$(A7)$ deld(putd(y, d), e) $=$
if $d == e$ then y else putd(deld(y, e), d) fi

$\}$

Die Sorte *Set* basiert auf z.B. der Sorte *FSet*, erweitert diese jedoch um Funktionen wie z.B. die Mengenvereinigung, die Funktion {_} zur bequemen Mengenbildung, usw.