7.7.2000 玉井哲雄

Z をどう教えたらよいか？

1　Z による酒屋問題の記述

Z はその基礎を集合論におき，すべてのデータは集合論に基づく型づけがなされる．そのようなデータの集まりで構成される情報単位とその性質，およびそれに対する操作を記述する枠組としてスキーマがあり，Z の言語の中で中心的な役割を果たす．

以下で，Z の紹介を兼ねながら酒屋問題の仕様を Z で記述する．記述の順序はこの問題の仕様化の過程に沿っているが，節の見出しは Z の導入という意味合いでつけている．

2　Z における型

Z で表現されるすべての対象 (式) は，型を持つ．型は集合とみなしてよいが，基本的な集合としてあらかじめ与えられたものか，それらから構成された構造を持ったものかのいずれかである．基本的な集合の代表的なものは整数集合であるが (Z では $\mathbb{Z}$ と書かれる)，個々の問題領域に応じて定められるものも多い．

われわれの問題では，“コンテナ番号” と “品名” を基本的なデータの集合として所与とする．また，“数量” という名称を 0 を含まない (1 以上の) 自然数として用いる．このことを，次のように書く．

$$[\text{コンテナ番号}, \text{品名}]$$

$$\text{数量} == \mathbb{N}_1$$

一般に，

$$[\text{識別子}, \ldots, \text{識別子}]$$

により，基本となる型 (集合) の名前が導入される．以降の Z の記述では，ここで導入された型を，その内部構造には立ち入らず所与のものとして用いる．

また，

$$\text{変数名} == \text{式}$$

によって，右辺の式を左辺の名前で指すようにできる．

基本型から構成される構造的な型には，巾集合型，直積集合型，およびスキーマ型があるが，それらの具体例は以降の節に登場する．

3　スキーマ

さて，倉庫にはコンテナが積まれている．これを倉庫という名前のスキーマとして表そう．

倉庫

$$\text{コンテナ} : \text{コンテナ番号} \pfun (\text{品名} \pfun \text{数量})$$

スキーマは一般に，スキーマ名，宣言部と公理部とからなり，全体を右が開いた矩形の枠で括る．スキーマ名は枠の上部の横線の中に書く．宣言部と公理部は矩形の中に書き，両者の間に横線を入れて区切るが，この例は宣言部のみからなり，公理部がない．この宣言部では，倉庫の状態を表す 1 つの変数 “コ

ンテナ" を，その型とともに示している．一般に，スキーマの宣言部には複数の変数宣言が並んでよい．それによって，このようなスキーマはいわゆるレコード型のデータ宣言とみなすことができる．しかし，変数の型には一般に関数なども含まれるから，むしろオブジェクト指向でいうオブジェクトに近い．

このコンテナの型がまさに関数となっている．記号 $\pfun$ は部分関数を表すので，コンテナはコンテナ番号を定義域，(品名 $\pfun$ 数量) を値域とする部分関数であると読める (部分関数とは，必ずしもすべての定義域で関数値が定められていない関数をいう．これに対し，すべての定義域で定義された関数を全関数という)．また，値域の (品名 $\pfun$ 数量) 自身が部分関数で，品名を定義域に，数量を値域にしている．すなわち，コンテナは数量が定められた品名の集まりである．

このような関数も，集合論的に解釈するのが Z の流儀である．すなわちコンテナの集合としての型は，

$$\text{コンテナ} \in \mathbb{P}(\text{コンテナ番号} \times \mathbb{P}(\text{品名} \times \text{数量}))$$

となる．ここで，$\mathbb{P}$ は巾集合を作る演算である．すなわち，$\mathbb{P}\,X$ は X の部分集合の集合である．また，$X \times Y$ は X と Y の直積集合を作る演算である．すなわち，$X \times Y$ の要素は，X の要素 x と Y の要素 y の順序対 (x, y) である．

Z ではこのように，関数や関係を集合としてとらえるところに特徴がある．領域 X から Y への関数や，領域 X と Y との間の関係は，いずれも直積集合 $X \times Y$ の部分集合としてとらえ，その基本型は同じであるとする．関数は関係の一種であるが，そのうち定義域の任意の要素に対しては，それを含む対がたかだか 1 つしか関係として含まれないもの，という制約を満たすものである．

集合 X と集合 Y の間の関係という型は，$X \rel Y$ と書かれる．Z では，これを

$$X \rel Y == \mathbb{P}(X \times Y)$$

と定義している．これは，ある関係 R が $X \rel Y$ の型を持つ，すなわち $R : X \rel Y$ ということは，$R \in \mathbb{P}(X \times Y)$ であること，つまり R は集合として $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$，ただし $x_i \in X, y_i \in Y (i = 1, \ldots, m)$，という形をしていることを意味している．

関係 $X \rel Y$ に対して，dom と ran という，それぞれ $(X \rel Y) \to \mathbb{P}\,X$ 型と $(X \rel Y) \to \mathbb{P}\,Y$ 型の関数が標準的に定義されている．関係 R に対し，$\mathrm{dom}\,R$ はその定義域 (domain) を，$\mathrm{ran}\,R$ は値域 (range) を表す．

Z で関係や関数を集合として扱うのはそんなに分かりにくいことではないが，型と値，あるいは型の階層を混同しがちである．たとえば $R == \mathbb{P}\,\mathbb{P}\,S$ という型 R を考えた場合，R を型とする変数の宣言 $x : R$ や，x が R の要素であるという述語 $x \in R$ によって，x の値は S の部分集合の集合の一つとなる．さらに，$y \in x$ とすれば，y の値は S の部分集合の一つになる．またさらに，$z \in y$ とすれば，z の値は S の要素になる．このように，型宣言:や述語 $\in$ の左辺は，集合の集合という関係が作る階層で，右辺より 1 レベル下になる．一方，集合の階層レベルが上がるのは，$\mathbb{P}$ を直接適用した場合の他に，関係の定義を見れば分かるように，関係や関数をとることによっても生じるので，慣れない内はそこに注意がいる．

スキーマの公理部とは，宣言部で定義された変数が満たすべき制約条件を，公理として与えるためのものである．この例で，問題文にある「1 つのコンテナには 10 銘柄まで混載できる」という表現を制約条件として与えるとすると，スキーマを次のように記述すればよい．

倉庫

$$\text{コンテナ} : \text{コンテナ番号} \pfun (\text{品名} \pfun \text{数量})$$

$$\forall\, c : \mathrm{dom}\ \text{コンテナ} \bullet \#(\text{コンテナ}\ c) \leq 10$$

ここで，dom はすでに述べたように，関数 (一般には関係) を引数としその定義域を与える関数であり，# は有限集合を引数としてその要素数を与える関数である．いずれも Z では標準的な関数として定義されている．公理部には述語論理式が書かれる．論理演算子としては，以下が用いられる．

$\neg$　否定
$\wedge$　連言
$\vee$　選言
$\Rightarrow$　含意
$\forall$　全称
$\exists$　存在

全称束縛の論理式は，一般に次のように書かれる．

$\forall$ 宣言 [; 宣言; ...] • 論理式

ここで，宣言とは

宣言 ::= 変数名 [,変数名,...] : 型

ただし，型とは集合を表す式である．存在束縛の論理式も同様である．

なお，スキーマの公理部には，複数の論理式を改行によって並べて記述してもよい．その場合は，全体は各行の論理式の連言からなるものとみなされる．すなわち，改行を連言記号の代わりに用いることができる．

Zでは，述語も集合論的に解釈される．すなわち，述語 $P(x)$ と書いた場合，P は集合を表し，その意味は $x \in P$ と等価である．したがって，新たな述語の定義も，集合を構成的に与えることでなされる．

コンテナを倉庫に入れる入庫処理は，データベースへのデータの追加という典型的な処理で，Zの種々の教科書にも必ず最初に出てくる例なので，ここでは後回しにする．

4　汎用構成子

4.1　高階関数

問題は，顧客の注文に応じた出庫の処理である．客は，ある品名の酒を何本というように注文してくる．これに応じるには，倉庫に入っているコンテナ単位の酒を，品名単位に検索する必要が生じる．そのために，酒在庫という概念を考えよう．

酒在庫 : 品名 $\pfun$ (コンテナ番号 $\pfun$ 数量)

これを，倉庫の宣言部に加える．酒在庫の型を集合の型として示すと，

酒在庫 $\in \mathbb{P}$(品名 $\times \mathbb{P}$(コンテナ番号 $\times$ 数量))

である．

このコンテナと酒在庫が実質的に同じものであるというのが，この問題のみそである．そのために，少し一般化して考えてみよう．コンテナや酒在庫の型は，$(X \pfun (Y \pfun Z))$ という形をしているが，これは $((X \times Y) \pfun Z)$ に1対1の関係で変換することができる．この変換は，関数型プログラミングで，多変数関数を1変数関数の列に変えるカリー化の操作のちょうど逆になっている．そこで，型 $(X \pfun (Y \pfun Z))$ の関数を型 $((X \times Y) \pfun Z)$ の関数に変換する全単射の (高階な) 関数 $uncurry$ を定義してみよう．この関数を定義する際，対象となる型 X, Y, Z は，特定のものである必要はなく，一般に任意の型でよい．このように型をパラメータとして汎用的な関数を定義するのに便利な記述形式として，Zでは汎用定義というものが用意されている．これは，次のように書く．

$[X, Y, Z]$

$$uncurry : (X \pfun (Y \pfun Z)) \rightarrowtail ((X \times Y) \pfun Z)$$

$$\forall f : (X \pfun (Y \pfun Z)) \bullet$$
$$uncurry\, f =$$
$$\{ x : X;\ y : Y;\ z : Z \mid x \in \mathrm{dom}\, f \wedge y \in (f\ x) \wedge z = f\ x\ y \bullet (x, y) \mapsto z \}$$

これは，任意の型，X,Y,Z をパラメータとする大域的な関数 $uncurry$ を定義したものである．真中の線より下の公理部で，関数の性質を定めている．ここで，$\rightarrowtail$ は全単射を表す[1]．また，$x \mapsto y$ は，関数の要素として値 x を値 y に写像する対を表している．すなわち，$x \mapsto y \in f$ であれば $f(x) = y$ である．集合としての表現と対応させると，$x \mapsto y$ は実は (x, y) と同じものである．なお，Z では関数型プログラミングの習慣に従い，関数適用を表すのに必要ない限り括弧は通常用いない．したがって，$f(x)$ は $f\,x$ と書くのが普通である．

この関数定義には，集合の内包記法を用いている．集合の内包記法は一般に次の形をしている．

$$\{D \mid P \bullet E\}$$

ここで，D は変数宣言，P は制約条件を表す論理式，E は式である．D で宣言された変数がとるその型の範囲内のすべて値のうち，論理式 P を満たすもののみを選び，それを E に代入して得られる値を集めた集合がこれによって表される．たとえば $\{x : \mathbb{N}_1 \mid x \leq 5 \bullet x^2\}$ は $\{1, 4, 9, 16, 25\}$ と等価である．全称あるいは存在束縛の論理式でも $\bullet$ を区切り記号として用いているが，集合の内包記法と混乱しないように適切な括弧を入れる必要がある場合があるので，注意がいる．

なお，E は省略可能で，その場合は P を満たす D の値そのものを集めた集合が表されることになる．たとえば $\{x : \mathbb{N}_1 \mid x \leq 5\}$ は $\{1, 2, 3, 4, 5\}$ と等価である．実際上は，この形式の方が使われることが多いかもしれない．

これで，述語論理式の書き方と集合の内包記法の書き方を見たことになるので，両者を使った関数の定義を見ておくことにする．すでに述べたように，関数は関係の一種で，定義域の任意の要素に対して，それを含む対がたかだか 1 つしか含まれないもの，であった．これは，Z で次のように定義される．

$$X \nrightarrow Y == \{f : X \leftrightarrow Y \mid (\forall\, x : X;\ y_1, y_2 : Y \bullet \\ (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f \Rightarrow y_1 = y_2)\}$$

(この式に現れる $\bullet$ は，集合の内包記法に出てくる $\bullet$ ではなく，全称束縛の $\bullet$ であることに注意．)

さて，Uncurry によってコンテナを 2 引数の関数とみなす変換ができたので，次に第 1 引数と第 2 引数を入れ換える変換を考える．

$[X, Y, Z]$

$$swap : ((X \times Y) \nrightarrow Z) \rightarrowtail ((Y \times X) \nrightarrow Z)$$

$$\forall f : (X \times Y) \nrightarrow Z \bullet \\ swap\, f = \{\, x : X;\ y : Y;\ z : Z \mid (x, y) \in \mathrm{dom}\, f \land z = f(x, y) \bullet (y, x) \mapsto z \,\}$$

結局，一連の変換は $uncurry^{\sim} \circ swap \circ unurry$ と表すことができる．ここで，$\circ$ は関数の合成を，$uncurry^{\sim}$ は $uncurry$ の逆関数を表す (単射の関数には逆関数が定義できる)．この逆関数は $curry$ と名付けるのが自然だから，

$$curry == uncurry^{\sim}$$

としておく．

これらを用いて，倉庫のスキーマを改めて次のように書くことができる．

倉庫

コンテナ : コンテナ番号 $\nrightarrow$ (品名 $\nrightarrow$ 数量)
酒在庫 : 品名 $\nrightarrow$ (コンテナ番号 $\nrightarrow$ 数量)

酒在庫 $=$ $curry \circ swap \circ uncurry$ コンテナ

[1]全単射は，全射でかつ単射という意味である．$f : X \nrightarrow Y$ が全射とは，その値域が Y 全体を覆う場合，すなわち $\mathrm{dom}\, f = Y$ の場合で，X から Y の上への写像とも呼ばれる．単射は，いわゆる 1 対 1 の写像で，$f : X \nrightarrow Y$ が単射とは，$\forall\, x_1, x_2 : \mathrm{dom}\, f \bullet f\, x_1 = f\, x_2 \Rightarrow x_1 = x_2$ を意味する．

4.2 多重集合

出庫依頼があった場合，指定された銘柄の酒の在庫量が，注文に応じられるだけあるかという判断が必要となる．また，コンテナが空の場合にはコンテナを搬出するという要求があるので，コンテナに現在入っている酒の総量も必要かも知れない．これらを与える関数は，

銘柄在庫量 : 品名 $\pfun$ 数量
コンテナ在庫量 : コンテナ番号 $\pfun$ 数量

という形式となろう．この 2 つの関数の仕様を書いてみよう．

これらの関数と，酒在庫およびコンテナの定義を比べれば，それぞれの関係は明らかである．銘柄在庫量は酒在庫と同じく品名を定義域とし，その写像先の コンテナ番号 $\pfun$ 数量 に関して数量の総和をとったものである．同様に，コンテナ在庫量は，コンテナの値域の要素に対してやはりその数量の総和をとったものとみなせる．このことを形式的に記述すればよい．

そこで，たとえば 品名 $\pfun$ 数量，あるいは同じことだが，$\mathbb{P}$(品名 $\times$ 数量) の要素に対して，それぞれの第 2 成分である数量を集めるという操作をまず考え，次にその総和をとることにすればよい．それには一般に関係に対して定義されている関数 ran を使えばよいようだが，問題は第 2 成分を集めたものを集合とすると都合が悪いことである．集合は同じ要素を重複して考えないが，この場合は同じ数量の値が複数回現れても，それぞれ別のものとして扱う必要がある．そのような集まりを多重集合 (bag または multiset) という．

要素 $a_1, \ldots, a_n$ からなる多重集合を，Z では $[\![a_1, \ldots, a_n]\!]$ と書く．たとえば，$[\![1,2,2,3]\!]$ と $[\![1,2,3]\!]$ とは異なる (しかし順序は関係ないから，$[\![1,2,2,3]\!]$ と，たとえば $[\![1,2,3,2]\!]$ は同じである)．Z では多重集合を，要素から 1 以上の自然数への関数として定義している．写像先の自然数が多重集合内の出現回数を表している．すなわち，一般に型 X の多重集合 $\mathrm{bag}\, X$ は，

$$\mathrm{bag}\, X == X \pfun \mathbb{N}_1$$

と定義される．たとえば $[\![1,2,2,3]\!]$ は $\mathrm{bag}\,\mathbb{N}$ の要素であるが，定義によれば $\{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 1\}$ と表される．

われわれは ran の多重集合版が欲しいので，参考のために dom と ran の定義を見てみよう．

$[X, Y]$

$$\begin{array}{l} \mathrm{dom} : (X \rel Y) \to \mathbb{P}\, X \\ \mathrm{ran} : (X \rel Y) \to \mathbb{P}\, Y \\ \hline \forall\, R : X \rel Y \bullet \\ \quad \mathrm{dom}\ R = \{\, x : X;\ y : Y \mid x \mathrel{\underline{R}} y \bullet x \,\} \land \\ \quad \mathrm{ran}\ R = \{\, x : X;\ y : Y \mid x \mathrel{\underline{R}} y \bullet y \,\} \end{array}$$

ここで $\underline{R}$ は関係 R を中置記号の演算子として使う場合の記法で，$x \mathrel{\underline{R}} y$ は $(x, y) \in R$ と等価である．

これをまねて，多重集合用の dom と ran に相当する関数 *bdom* と *bran* の定義を次のように書いてみる．

$[X, Y]$

$$\begin{array}{l} \mathit{bdom} : (X \rel Y) \to \mathrm{bag}\, X \\ \mathit{bran} : (X \rel Y) \to \mathrm{bag}\, Y \\ \hline \forall\, R : X \rel Y \bullet \\ \quad \mathit{bdom}\ R = [\![\, x : X;\ y : Y \mid x \mathrel{\underline{R}} y \bullet x \,]\!] \land \\ \quad \mathit{bran}\ R = [\![\, x : X;\ y : Y \mid x \mathrel{\underline{R}} y \bullet y \,]\!] \end{array}$$

残念ながら，現在の Z にこのような書き方はない．多重集合を記述する $[\![\cdots]\!]$ には，外延的な記述はできても内包的な記述は定められていない．

そこで，

$$
\begin{array}{|l}
[X, Y] \\
\hline
bdom : (X \leftrightarrow Y) \to \mathrm{bag}\, X \\
bran : (X \leftrightarrow Y) \to \mathrm{bag}\, Y \\
\hline
\forall R : X \leftrightarrow Y \bullet \\
\quad bdom\, R = \{ x : \mathrm{dom}\, R \bullet x \mapsto \#\{ y : Y \mid x \mathrel{\underline{R}} y \} \} \land \\
\quad bran\, R = \{ y : \mathrm{ran}\, R \bullet y \mapsto \#\{ x : X \mid x \mathrel{\underline{R}} y \} \} \\
\hline
\end{array}
$$

とすればよいだろう．

次に，自然数 $\mathbb{Z}$ の多重集合について，その要素の総和をとる関数 Σ を定義しよう．

$$
\begin{array}{|l}
\Sigma : \mathrm{bag}\, \mathbb{Z} \to \mathbb{Z} \\
\hline
\Sigma[\![]\!] = 0 \\
\forall x : \mathbb{Z} \bullet \Sigma[\![x]\!] = x \\
\forall B, C : \mathrm{bag}\, \mathbb{Z} \bullet \Sigma(B \uplus C) = \Sigma B + \Sigma C \\
\end{array}
$$

ここで用いた記法は，スキーマの外枠がない形をしているが，公理記述と呼ばれ，大域変数を宣言しその性質を公理として与えるものである．スキーマで宣言された変数はそのスキーマ名を引用しないと参照できないが，公理記述で宣言された変数は大域的なのでいつでも参照できる．その点では汎用定義で宣言される変数と似ているが，汎用定義はパラメータ化された大域変数 (通常は関数) を定めるのに対し，公理記述はコンスタントとしての変数 (関数) を定めているところが異なる．

なお，$[\![]\!]$ は空の多重集合を表し，記号 $\uplus$ は 2 つの多重集合の合併をとる演算を表す．

この Σ のような関数は，単位元を持ち可換で結合的な 2 項演算の定義された集合 X の上の多重集合 $\mathrm{bag}\, X$ に対して，一般化して定義することも可能だが，ここではそこまで一般化せず，自然数の上の加算に対して定義した．

以上の準備の元に，酒在庫およびコンテナからそれぞれ銘柄在庫量およびコンテナ在庫量をえる変換を $subtotal$ という (高階) 関数として定義すると，

$$
\begin{array}{|l}
[X, Y] \\
\hline
subtotal : (X \pfun (Y \pfun \mathbb{Z})) \to (X \to \mathbb{Z}) \\
\hline
\forall f : X \pfun (Y \pfun \mathbb{Z}) \bullet subtotal\, f = \{ x : dom\, f \bullet x \mapsto \Sigma \circ bran \circ f\, x \} \\
\hline
\end{array}
$$

これを用いて，

銘柄在庫量 $==$ $subtotal$ 酒在庫
コンテナ在庫量 $==$ $subtotal$ コンテナ

となる．

倉庫のスキーマを，この 2 つの関数を含めた形に拡張して定義し直しておく．

倉庫

コンテナ : コンテナ番号 $\pfun$ (品名 $\pfun$ 数量)
酒在庫 : 品名 $\pfun$ (コンテナ番号 $\pfun$ 数量)
銘柄在庫量 : 品名 $\pfun$ 数量
コンテナ在庫量 : コンテナ番号 $\pfun$ 数量

酒在庫 $= curry \circ swap \circ uncurry$ コンテナ
銘柄在庫量 $== subtotal$ 酒在庫
コンテナ在庫量 $== subtotal$ コンテナ

5 抽象機械

これまでの Z による記述は，対象とする問題領域の，主として静的な側面をとらえていた．その方法としては抽象化された関数を利用し，対象のもつ論理的な構造をなるべく簡潔にとらえるというものであった．しかし，Z の大きな特徴の一つは，対象を状態を持つ抽象機械としてとらえ，その動的な動作を記述するところにある．スキーマは，そのために用いられることが多い．

操作の仕様 たとえば，倉庫というスキーマは，これまでのところはコンテナと酒在庫という構成要素を持つデータ型の値がとり得る値の空間を規定したもの，という扱いであった．これを抽象機械として考える場合は，任意の時点でそのような値の 1 つを状態として持つ機械 (またはプロセス，またはシステム) を想定することになる．機械は，外から受ける操作によって状態を変化させる．その状態変化を表すには，操作前と操作後の状態間の関係を記述する仕組みが必要である．

たとえば，コンテナの入庫処理を考えてみよう．

入庫

倉庫
倉庫′
$cn?$: コンテナ番号; $s?$: 品名 $\pfun$ 数量

$cn? \notin \mathrm{dom}$ コンテナ
コンテナ′ $=$ コンテナ $\cup \{cn? \mapsto s?\}$

このスキーマ定義では，他のスキーマの引用と修飾という重要な記法が使われている．まずスキーマの引用として，ここでは “倉庫” スキーマが引用されている．この場合，引用されたスキーマの宣言部はそのままここで定義しているスキーマ “入庫” の宣言部に加えられ，またその公理部はそのまま “入庫” の公理部に加えられる．一方，“倉庫′” という記法はスキーマの修飾つき引用で，その意味は倉庫の宣言部で宣言されたすべての変数 (“コンテナ”，“酒在庫”，“コンテナ在庫量”，“銘柄在庫量”) に ’ を付け，またその公理部に現れるそれらの変数もすべて ’ 付きに置き換えてえられるスキーマを引用するということである．この ’ 付きの変数は，通常はここで定義しようとしている “入庫” という操作を表すスキーマの，操作終了後の状態を指し示すと解釈される．

状態機械の操作を定義するスキーマでは，このように状態を表すスキーマと，それに ’ を付けたスキーマの両者を引用することが普通なので，それをいっぺんに表す記法も定められている．それには Δ(ギリシャ文字デルタの大文字) を引用するスキーマ名の頭につける．つまり，上の例では

入庫

Δ倉庫

$cn?$: コンテナ番号; $s?$: 品名 $\pfun$ 数量

$cn? \notin \mathrm{dom}$ コンテナ

コンテナ$' =$ コンテナ $\cup \{cn? \mapsto s?\}$

と書くことができる.

入力/出力条件　このスキーマでは別に，$cn?$ と $s?$ という 2 つの変数を導入している．このように後ろに?を付けた変数は，この操作に対する入力データを表す変数と解釈される．また，この例では現れないが，出力を表す変数は後ろに!を付ける決まりである．これらはしかし，あくまでも状態機械の操作を表しているという解釈のための便宜上の約束で，論理式としては通常の変数とまったく同様の変数にすぎない.

このような操作のスキーマの公理部は，操作の入力条件と出力条件を表している．Z 以外の形式的仕様記述言語では，入力条件と出力条件を構文的に区別して記述するようにしているものが多いが (たとえば VDM)，Z ではそのような区分をしない．しかし，公理部を構成する論理式 (全体はそれらを連言で結合したものと解釈される) のうち，'あるいは!を接尾辞としてもつ変数を一つも含まないものは入力条件であり，それらを一つでも含むものは出力条件である解釈できるので，あえて入力/出力条件を区分する構文要素を導入しない方針をとっているわけである.

この' 記法は強力で，とくに引用されたスキーマの公理部も自動的に取り込まれることで，操作によって不変な条件を明示的に書く必要がない．上の例で，“コンテナ” に関する出力条件のみを記述して，“酒在庫” について記述していないのは，修飾つきのスキーマの引用により，暗に

酒在庫$' = curry \circ swap \circ uncurry$ コンテナ$'$

が成立している (入庫のスキーマの公理部に加えられている) からである．このやり方は Z 独特で便利ではあるが，時として操作の仕様を分かりにくくすることもある.

参照型の操作　操作の中には状態を変化させないで，状態を参照するためのものもありうる．その場合は，引用するスキーマの頭に Ξ(ギリシャ文字クシーの大文字) を付けると，Δ を付けた場合と同様に，そのスキーマと' 付きのスキーマとを同時に引用するだけでなく，公理部には暗にスキーマで宣言されているすべての変数 x に対し，$x = x'$ という等式が追加される．(しかし，実際に公理部に記述する論理式には，' 付きの変数は現れないはずである．)

たとえばすでに定義した関数 “銘柄在庫量” を使って，銘柄を指定し在庫量を調べる参照操作を定義してみよう.

銘柄在庫確認

Ξ倉庫

$b?$: 品名, $v!$: 数量

$v! =$ 銘柄在庫量 $b?$

初期化　状態機械ならば，その初期状態をなんらかの方法で定める必要がある．それも一つのスキーマとして定義すればよい．倉庫に対しては，たとえば

初期倉庫

倉庫

コンテナ $= \emptyset$

これを初期化の操作として定義するなら，Δ 倉庫を用いて コンテナ$' = \emptyset$ とすることになるが，上の定義は倉庫の初期状態を宣言的に記述したものである．

エラー処理　“入庫” 操作では，入庫されるコンテナのコンテナ番号が，すでに倉庫にあるコンテナのコンテナ番号と一致しないという入力条件をおいている ($cn? \notin \mathrm{dom}$ コンテナ)．この条件を満たさないような例外事象を，仕様としてどのように記述したらよいだろうか．Z ではそのような例外処理の仕様は，別のスキーマとして分けて記述するのが，通常のスタイルである．

まず，エラーを識別するための記号からなる型，“エラー条件” を定義する．

エラー条件 ::= 正常 | コンテナ番号重複

ここで ::= は自由型の宣言子で，右辺で | で区切られて列挙された要素で構成される型を宣言している．この例では単純な列挙型だが，再帰的な定義が可能で，それにより木のような再帰的に定義される型を導入できる．

次に，正常な場合を表すスキーマを，“正常終了” として定義する．

正常終了

結果! : エラー条件

結果! = 正常

これはつまらないスキーマに見えるが，スキーマ式という仕組みを使うことで役に立つようになる．スキーマ式とはスキーマ同士を演算で結んだもので，その演算子にはいくつかの種類があるが，ここでは論理型スキーマ演算子の $\wedge$ と $\vee$ のみをあげておく．どちらも 2 つのスキーマをとり，新たなスキーマを構成する．2 つのスキーマの宣言部は，どちらの場合も合併がとられる．その際，同じ名前の変数が両者にある場合は，それぞれのスキーマにおけるその変数の型 (集合) の共通部分をとって，構成されるスキーマでは，それを型とする一つの変数が宣言される．また，公理部は $\wedge$ の場合は 2 つのスキーマの公理部の連言が，$\vee$ の場合は 2 つのスキーマの公理部の選言がとられる．

そこで，“入庫 $\wedge$ 正常終了” とすれば，さきほど定義した入庫に正常という結果を返すという機能が付加されることになる．

次にコンテナ番号の重複が起こるケースは，次のようなスキーマで記述すればよい．

番号重複

Ξ倉庫

$cn?$: コンテナ番号

結果! : エラー条件

$cn? \in \mathrm{dom}$ コンテナ

結果! = コンテナ番号重複

これを用いて，例外条件を含めた入庫操作は，

一般化入庫 $\widehat{=}$ (入庫 $\wedge$ 正常終了) $\vee$ 番号重複

と書ける．ここで $\widehat{=}$ は，左辺のスキーマを右辺のスキーマ式で定義するものである．

6　仕様の完成

仕様の残りの部分を完成させよう．具体的には，酒の出庫の操作と在庫不足処理についての仕様を記述する必要がある．

6.1 出庫

出庫処理は，受付係が出庫依頼を受けて出庫指示書を作り，それに基づいて倉庫係が出庫する，という一連の処理となる．在庫不足の場合は，在庫なし連絡をして，在庫不足リストを作成する．

この出庫処理全体の仕様を一つのスキーマとすれば，在庫がある場合の出庫というスキーマと在庫不足の場合の処理を記述するスキーマに分け，

出庫処理 ≙ 出庫 ∨ 在庫不足処理

のような構造とすればよいであろう．また，出庫は受付係の仕事と倉庫係の仕事に分け，その順に処理するように記述するのが自然だろう．それには，

出庫 ≙ 出庫指示書作成 >> 出庫実施

と書けばよい．ここで，>> はパイプ結合を表すスキーマ演算子で，$A >> B$ は直観的には A の実行の後，その出力を B の入力として B を実行することを意味する．より正確な意味は後で示す．

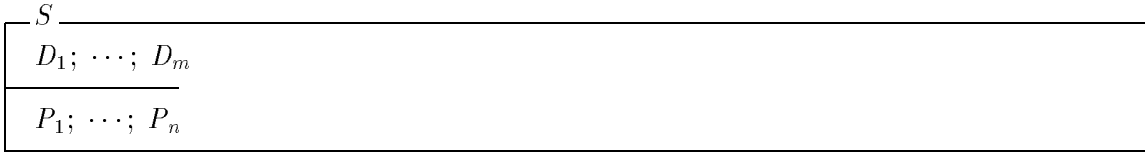
以下で，これらのスキーマを記述していこう．まず，基本的なデータとして，出庫依頼と出庫指示書を定義する．そのために，基本型として，“送り先名” を追加しておく．

[送り先名]

これを用いて，出庫依頼と出庫指示書を次のように定義する．

出庫依頼 ≙ [b : 品名; a : 数量; to : 送り先名]
出庫指示書 ≙ [to : 送り先名; b : 品名; f : コンテナ番号 ⇸ 数量]

今までは，スキーマを定義するのには常に箱を用いてきた．しかし，この場合のように，単に組 (tuple) のデータ構造を定義するために導入するような短いスキーマ定義には，箱の表記は大げさ過ぎる．そこでこのような「横書き」の表記法がある．一般に，箱を用いた縦書き表記，



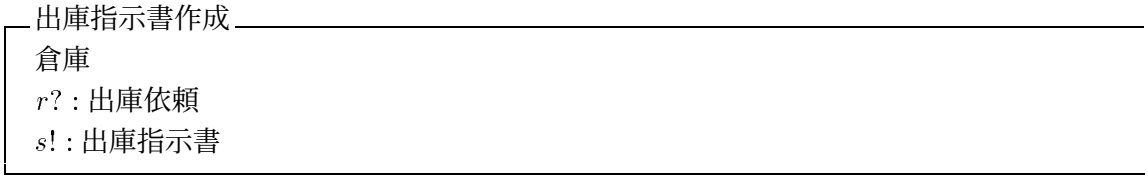
と，横書き表記

S ≙ [D_1; ···; D_m | P_1; ···; P_n],

とは等価である．ただし，縦書き表記では，セミコロンの代わりに改行を用いてもよい．

上の定義で，問題文にある出庫指示書の構造定義から「注文番号」を省き，また品名は 1 つの送り先について共通なので，コンテナに関するデータの繰り返しの外側に括り出した．さらに，ここではとりあえず空コンテナに関する仕様は後回しにすることにし，省いている．

出庫指示書作成の宣言部は，次のような形式になろう．



ここでスキーマを型として用いている．スキーマの宣言部は，変数名と型の対の並びであり，公理部は宣言部で宣言された変数の間に成り立つべき制約条件を与えるものであった．したがって，スキーマはいわゆるレコード型を，それが保つべき不変条件とともに定義しているものとみなすことができる．

公理部には，出庫依頼にある品名の数量が現在の在庫で充足されるという入力条件と，出庫指示書にある数量の合計がその依頼数量と一致するという出力条件とを規定する必要がある．そのためには，すでに準備した銘柄在庫量という便利な関数があるので，それを使えばよい．すなわち，

出庫指示書作成

$$
\begin{array}{l}
\text{倉庫} \\
r? : \text{出庫依頼} \\
s! : \text{出庫指示書} \\
\hline
r?.b \in \mathrm{dom}\ \text{銘柄在庫量} \land r?.a \leq \text{銘柄在庫量}\ r?.b \\
s!.to = r?.to \land s!.b = r?.b \land \Sigma \circ bran\, s!.f = r?.a \land \\
\forall\, cn : \mathrm{dom}\ s!.f \bullet cn \in \mathrm{dom}\ \text{コンテナ} \land f\ cn \leq \text{酒在庫}\ r?.b\ cn
\end{array}
$$

ここですでに定義した Σ と $bran$ を用いた．

この仕様は，非決定的である．すなわち，この仕様を満たす出力 $s!$ は一般に一意には定まらない．しかし，仕様は決定的である必要はまったくない．むしろ本質的でない性質についてはあえて自由度を残して非決定的に記述した方が，仕様として簡明で分かりやすいことが多い．ただし，あまりに自由度が大きな仕様は，設計者の負担が大きくなるという面もあり，その兼ね合いに配慮がいる．

出庫実施のスキーマを記述するには，少し準備がいる．次のような総称関数 $ransum$ を用意する．これは同じ定義域を持ち，値域が整数である 2 つの関数から，同じ定義域の要素に対してそれぞれの関数の適用結果の和を与えるような関数を作る高階関数である．

$[X]$

$$
\begin{array}{l}
ransum : (X \pfun \mathbb{Z}) \to (X \pfun \mathbb{Z}) \to (X \pfun \mathbb{Z}) \\
\hline
\forall\, f, g : X \pfun \mathbb{Z} \bullet \\
ransum\, f\ g = \{x : (\mathrm{dom}\, f) \cap (\mathrm{dom}\, g) \bullet x \mapsto ((f\ x) + (g\ x))\} \cup \\
\qquad\qquad (\mathrm{dom}\, g) \ndres f \cup (\mathrm{dom}\, f) \ndres g
\end{array}
$$

ここで，$(\mathrm{dom}\, g) \ndres f$ は，f の内定義域が g と重なるものを除くことを表す．これを用いて，出庫実施は次のように書ける．

出庫実施

$$
\begin{array}{l}
\Delta\text{倉庫} \\
s? : \text{出庫指示書} \\
\hline
s?.b \in \mathrm{dom}\ \text{酒在庫} \land \\
\forall\, cn : \mathrm{dom}\ s?.f \bullet cn \in \mathrm{dom}\ \text{コンテナ} \land f\ cn \leq \text{酒在庫}\ s?.b\ cn \\
((s?.b \in \mathrm{dom}\ \text{酒在庫}' \land \text{酒在庫}\ s?.b = ransum\ (\text{酒在庫}'\ s?.b)\ s?.f) \lor \\
\qquad (s?.b \notin \mathrm{dom}\ \text{酒在庫}' \land \text{酒在庫}\ s?.b = s?.f)) \\
\{s?.b\} \ndres \text{酒在庫} = \{s?.b\} \ndres \text{酒在庫}'
\end{array}
$$

この記述も非決定的である．とくに $ransum$ という 2 つの項の “和” を取る関数を，操作後の状態値である酒在庫’ と入力変数とに対して用いている点に注意しよう．一般に，状態変数 S と 2 項演算 $\oplus$ があって，

$$S' = S \oplus A$$

のように書ける時は決定的であるが，

$$S = S' \oplus A$$

のように書ける場合は，$\oplus$ が逆演算をもたない限り決定的ではない．これは，S を S' と A に “分割” するが，分割の方法が一通りとは限らない場合と解釈することができる．しかし，あえて条件を付加して決定的な形に書き換えるよりも，この形式の方が仕様らしい記述となる場合がある．

出庫指示書作成と出庫実施をパイプ結合でつなげる．

出庫 0 ≙ 出庫指示書作成 >> 出庫実施

この結合は，出庫指示書作成の出力 $s!$ と出庫実施の入力 $s?$ を結合して同一の変数とし，さらにその変数を存在限量子で束縛して外部から隠すというものである．

6.2 在庫不足処理

在庫不足で出庫できない依頼については，それを在庫不足リストとして登録する．そのために在庫不足管理というスキーマを定義する．

在庫不足管理 ≙ [在庫不足リスト : seq 出庫依頼]

ここで $\mathrm{seq}\, X$ は X の要素からなる列 (sequence) を表す．Z では列は自然数から X への関数として定義される．すなわち，

$$\mathrm{seq}\, X == \{f : \mathbb{N} \pfun X \mid \mathrm{dom}\, f = 1..\#f\}$$

在庫不足処理 0

倉庫
Δ在庫不足管理
$r?$: 出庫依頼

$r?.b \notin \mathrm{dom}$ 酒在庫 $\lor$ ($r?.b \in \mathrm{dom}$ 酒在庫 $\land$ $r?.a >$ 銘柄在庫量 $r?.b$)
在庫不足リスト′ = 在庫不足リスト$^\frown$ < $r?$ >

ここで $^\frown$ は 2 つの列の連接演算子である．

在庫不足の場合は，依頼元に報告することになっているので，その分を常套的な方法で付け加える．

報告 ::= 充足 | 不足
不足報告 ≙ [r : 報告 | r = 不足]
充足報告 ≙ [r : 報告 | r = 充足]

結局，

出庫 ≙ 出庫 0 $\land$ 充足報告
在庫不足処理 ≙ 在庫不足処理 0 $\land$ 不足報告

となる．

これで仕様がようやく完成した．ただし，この仕様は酒屋問題の第 1 版に対応するものであり，在庫不足で応じられなかった出庫依頼に対して，後に入荷されたコンテナにより出庫が可能となった場合の出庫処理については記述していない．