CSC227
Formal Specification of Software(6)

John Fitzgerald
Centre for Software Reliability
University of New Castle

Translated by
Takahiko Ogino
Railway Technical Research Institute

Based upon the book
Modelling Systems: Practical Tools and Techniques in Software Development
(ISBN 0 521 623480 Cambridge University Press 1998)
by John Fitzgerald and Peter Gorm Larsen (IFAD)

凡例

- 本翻訳は、John Fitzgerald博士（Centre for Software Reliability、University of New Castle）の講義用OHPを翻訳したものである。
- 翻訳にあたっては、なるべく忠実な訳をしたつもりである。ただし原稿が、授業の教材であることを勘案し、出来るだけ意味的な注釈を付けたり、空白になっておりクラスで埋められるのであろうと思われる部分は、出来るだけ講義のベースとなっている本、Modelling Systemsからの引用によって中身を埋めた。
- そのような部分をオリジナルと区別するため、フォントの色を変化させることで出来るだけ区別している。
- また、同じ鉄道総研の寺田夏樹君には、日本語と内容のプルーフリーディングをお願いしたことを記載し感謝の意を表します。
- 原OHPの修正も含めて全ての日本語版内容に関する責任は、荻野にあります。
- 誤訳も含めて、内容に関するご指摘をお待ちしています。

写像：マッピング

- マッピングとは
- マッピングの定義
- マッピング上のオペレーター
- トラッキング・マネージャーでの例

マッピングとは　　定義

- 計算機システムは、値の集合相互間の関係をしばしば用いる。
- そのような関係は、ひとつの集合(領域(domain: dom)といわれる）の要素から、他方の集合(範囲(range: rng)といわれる）の要素へのマッピングとしてモデル化することができる。
- マッピングは、テーブルと見なすことができ、そのテーブルを用いて領域の要素から範囲の要素を読み取ることが出来る。

John	-500
Peter	-750
Susan	1025

- テーブルの各行はマッピングの一部を構成し、mapletと呼ばれる
- mapletは次の形を持つ。　“John” |-> -500 。
- ひとつのマッピングにおいて、各領域要素は単一のmapletしか持つことが出来ない。(多対1である）
- VDM-SLでは、マッピングは、マップレットの集合で表される。
- {“John” |-> -500, “Peter” |-> -750, “Susan” |-> 1025}

マッピングとは　　性質

- マッピングは、mapletの集合として定義されるので、mapletの順序は重要で無い。

{“John” |-> -500, “Peter” |-> -750, “Susan” |-> 1025} =

{“Susan” |-> 1025, “John” |-> -500, “Peter” |-> -750}

- 二つのマッピングが等しいのは、おなじ領域要素が同じ範囲要素をポイントするときのみである。

{“John” |-> -500, “Peter” |-> -750, “Susan” |-> 1025} <>

{“Susan” |-> 1025, “John” |-> -500, “Peter” |-> 0}

- 違う領域要素が同じ範囲要素をマップすることは、問題ないが、ひとつの領域要素は、正確に一つの範囲要素をマップする必要がある。

{“John” |-> -500, “Peter” |-> 1025, “Susan” |-> 1025}　　OK

{“John” |-> -500, “Peter” |-> -750, “Susan” |-> 1025 , “Peter” |-> 63} ×

マッピングタイプ構成子

- map 領域 to 範囲
  - **`map AccountNo to int`** は、通帳番号から(マイナスかもしれない)貯金残高へのマッピングを表している。
- 数え上げ(enumeration) mapletの集合として直接表せる。
  - 空のマッピング **`{|->}`**
  - **`{“9311290” |-> -500,`**
    `“1392842” |-> 3129,`
    `“445829n” |-> 0}`
  - **`{‘a’ |-> {2 |-> 7}, ‘b’ |-> {|->}}`**
- 内包(comprehension)
  - マッピング **`{1|->1, 2|-> 4, 3|->9, 4|->16, 5|->25}`** は内包を用いても表せる。
    - タイプ束縛で **`{n |-> n*n | n:nat1 & n < 5}`**
    - セット束縛で **`{n |-> n*n | n in set {1,...,5}}`**

マッピング上のオペレータ

- **マッピングの適用** : map A to B * A -> B
  m = {“John”|-> -500,“Peter”|-> -750,”Susan”|->1025}
  m(“John”) = -500
  m(“Peter”) = -750
- **dom** : map A to B -> set of A
  domain(領域)を抽出する
  dom {3 |-> 8, 7 |-> 4, 4 |-> 8} = {3, 7, 4}
  dom {|->} = {}
- **rng** : map A to B -> set of B
  range(範囲)を抽出する
  rng {3 |-> 8, 7 |-> 4, 4 |-> 8} = {8, 4}
  rng {|->} = {}

マッピング上のオペレータ

- **munion ： マッピングの結合** domainが同じ maplet については range も同じでなければならない（従って部分的）

```
_ munion _ : (map A to B)*(map A to B)-> (map A to B)

{5|->22, 7|->11} munion {3|-> -7, 12|->0, 44|->7}
          = {5|->22, 7|->11, 3|-> -7, 12|->0, 44|->7}
{3 |-> -7, 12 |-> 0} munion {12 |-> 4}×定義されない
```

- **++ ： マッピングの上書き** ドメインが共通の時は前者のマッピングを後者のマッピングで上書きする

```
_ ++ _: (map A to B) * (map A to B) -> map A to B

{3 |-> 8, 7 |-> 4, 4 |-> 8} ++ {3 |-> 3, 5|-> 4} =
                {3 |-> 3, 7 |-> 4, 4 |-> 8, 5 |-> 4}
```

マッピング上のオペレータ

- **<-:, <:** domain の限定 :

```
_ <-: _ , _ <: _ : set of A * map A to B -> map A to B
as <-: m = {x |-> m(x) | x in set dom m &
                         x not in set as}
as <: m = {x |-> m(x) | x in set dom m & x in set as}
```

- **:->, :>** range の限定

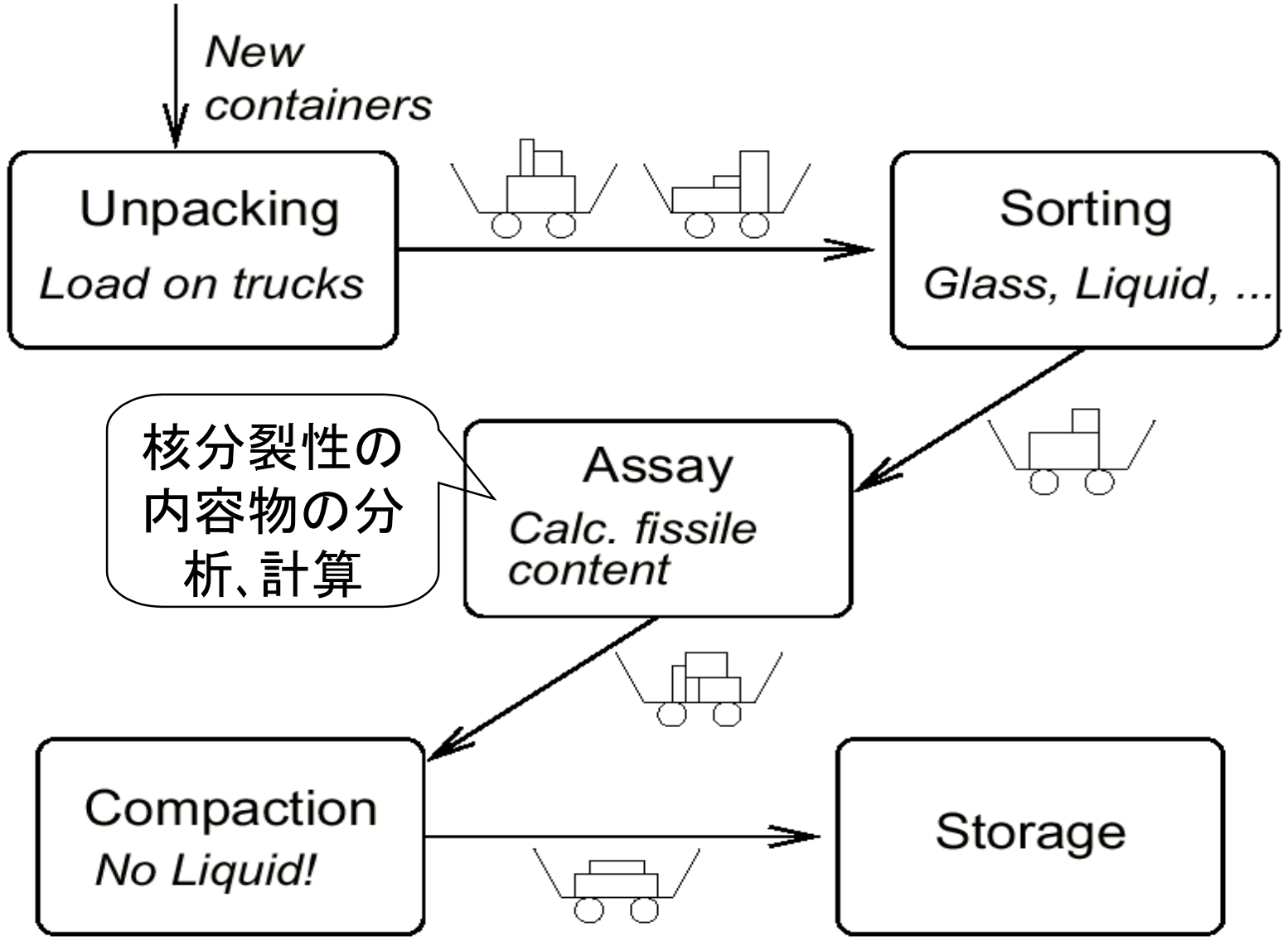
```
_ :-> _, _ :> _ : map A to B * set of B -> map A to B
m :-> as = {x |-> m(x) | x in set dom m &
                         m(x) not in set as}
m :> as = {x |-> m(x) | x in set dom & m(x) in set as}
```

値の演算子　　マップの合成(merge)

- 練習:
  - compatibleと呼ばれるブール関数を定義する。それは、二つのマッピングが合成(merge)出来るとき、またその時のみ真を返す関数である。

```
Compatible: (map A to B) * (map A to B) -> bool
  Compatible(m, n) ==
   forall a in set (dom m inter dom n) & m(a)=n(a)
```

核廃棄物処理追跡システムの例



核廃棄物処理追跡システムの概要

- 追跡システムは、プラントの中で処理する様々な段階を通り抜ける核廃棄物コンテナの位置を監視する。
- 典型的なプラントは物理的なプロセスに対応する多くのフェーズから成る。
- 例えば、クレート(crate)は最初の開梱過程に到着する。そこでは開封され、パッケージの内部が取り出され、それらが含んでいる物質(グラス、金属、プラスチック、液体)のタイプに従って違ったバスケットへ分別される。
- それに引き続く工程では、その次に送り出す前に、それぞれのバスケットの物質の放射能の測定と、復元可能な物質の含有度を測定する。その後の工程では、金属物質の溶解、またガラスの破砕を含みうる。
- 最終的に、プロセスの製品は梱包され顧客に返されるか、あるいは倉庫に格納される。
- 図は、典型的なプラントの概要を示す。

核廃棄物処理追跡システムの概要

- 各コンテナは独自な識別子を持っている。
- それは放射性物質(fissile mass)を含んでおり、単一の種類(ガラス、金属、液体など)の物質を含んでいる。
- 各工程は一意な識別子を持っている。
- 常に複数のコンテナがその工程中に存在する。
- これらのコンテナはすべて、ある1種類だけの物質(例えば、破砕工程は、それらの中のガラスを入れたコンテナだけを受け付ける。)だけを含んでいなければならない。
- 各工程には、最大容量という概念がある:即ち、任意の時間に、工程中に含まれうるコンテナ数の最大値を意味する。
- 危険な物質を取り扱う場合、明らかなことでは在るが、次の事柄は非常に重要である。
  - 物質が行方不明にならないこと
  - 一つのエリアに危険な物質の蓄積がある場合、ある工程にあまりに多くの物質が入ってこないようにする　->臨界値であろう。
- 追跡管理者は、危険な状況を回避するために処理する工程間のコンテナの移動許可を与える権限を持っている。

核廃棄物処理追跡システムの例

- 有害廃棄物のコンテナが再処理を受ける工程の中で、それらの移動を追跡するためのシステムモデルは、1995年に、BNFL(英国核燃料技術社)とマンチェスター大学情報科学科のチームによって開発された。
- モデルの目的はトラッキング・マネージャーが実行しなければならない廃棄物のコンテナーの移動を管理する規則を確定することである。
- モデルは安全性と関連するが、しかし重要なことは、モデルはその問題をよりよく理解するためだけに、単純化して構築されたのであり、ソフトウェア開発の拠り所として作られたのではないことに注意されたい。
- モデルは単に仕様書として使われてはならない。

核廃棄物処理追跡システム　基本タイプ

- 一番上の階層では、追跡システムはコンテナ、およびプラントの工程に関する情報を保持する:

```
Tracker :: containers : ContainerInfo
           phases     : PhaseInfo
```

- コンテナおよび工程情報は、それぞれの識別子から詳細記述へのマッピングとしてモデル化される。これは、識別子を領域(domain)に、詳細記述を範囲(range)にという、非常に一般的なマッピングの使用のされ方である。

```
ContainerInfo = map ContainerId to Container

PhaseInfo = map PhaseId to Phase
```

核廃棄物処理追跡システム　　基本タイプ

- 識別子がどう表わされるかの詳細はここでは不必要である:

```
ContainerId = token

PhaseId = token
```

- 各コンテナーについて、その内容の放射性成分や何を入れているのかを記録する。

```
Container :: fiss_mass : real
             material : Material


Material = token
```

核廃棄物処理追跡システム　　基本タイプ

- 各工程(Phase)は多くのコンテナーを収容し、特定の物質のタイプが来ることを想定しており、容量の最大値がある。

- 自分でこれをモデル化しなさい。:

```
Phase ::    contents : コンテナの集まり
            expected_materials : 物質のタイプ
            capacity : コンテナの数

Phase ::    contents : set of ContainerId
            expected_materials : set of Material
            capacity : nat
```

核廃棄物処理追跡システム　　基本タイプ

- 実際の核廃棄物処理追跡システムプロジェクトでは、BNFLからの領域エキスパートが、形式的モデルの開発で緊密に関係していた。
- 追跡システムによって守られなければならない安全特性を指摘してもらうことは、領域エキスパートに依存した。
- 例えば、工程中のコンテナーの数は工程の最大容量を超過してはならない:

```
Phase ::  contents : set of ContainerId
          expected_materials : set of Material
          capacity : nat
inv p == card p.contents <= p.capacity
```

- BNFLからの領域エキスパートは、不変式として制約を形式的に記録出来るこの機能は、非常に価値があると頻繁にコメントした。

核廃棄物処理追跡システム　不変式

```
Tracker :: containers : ContainerInfo
           phases     : PhaseInfo
inv mk_Tracker(containers,phases) ==
```

1. 工程中にある全てのコンテナは、containersのマッピングによって知ることが出来る。
2. 異なる2つの工程では、いかなるコンテナも共有されることは無い。
3. 任意の工程で、全てのコンテナは、想定されている種類の物質を収容している。

```
inv mk_Tracker(containers,phases) ==
      Consistent(containers,phases) and
      PhasesDistinguished(phases) and
      MaterialSafe(containers,phases)
```

核廃棄物処理追跡システム　不変式

```
-- 工程中にある全てのコンテナは containersのマッピングによって知ることが
   出来る。
Consistent(containers,phases) ==
   forall ph in set rng phases &
       ph.contents subset dom containers

-- 異なる2つの工程では、いかなるコンテナーも共有されることは無い

PhaseDistinguished(phases) ==
   not exists p1, p2 in set dom phases &
       p1 <> p2 and phases(p1).contents inter
                    phases(p2).contents <> {}

-- 任意の工程で、全てのコンテナは、想定されている種類の物質を収容している。

MaterialSafe(containers, phases) ==
   forall ph in set rng phases &
       forall cid in set ph.contents &
             cid in set dom containers and
containers(cid).material in set ph.expected_materials
```

核廃棄物処理追跡システム　　追跡システムの機能

- 新しいコンテナを追跡システムに導入し、その識別子を与え、内容を与える。
- コンテナが与えられた工程へ移動する許可を与える。
- 工程からコンテナを取り除く。
- 追跡システムからコンテナを削除する。

核廃棄物処理追跡システム　　コンテナの導入

```
Introduce: Tracker * ContainerId * real * Material
           -> Tracker
Introduce(trk, cid, quan, mat) ==
  mk_Tracker(trk.containers munion
      {cid |-> mk_Container(quan, mat)}, trk.phases)

pre cid not in set dom trk.containers
-- cidは今までに存在しているコンテナのidと異なっていること。
```

核廃棄物処理追跡システム　　コンテナの移動許可

```
Permission: Tracker * ContainerId * PhaseId -> bool

Permission(mk_Tracker(containers,phases),cid,dest) ==
```

-- 移動によって追跡システムの不変式が変化しないことをチェックする必要がある。

```
  cid in set dom containers and
      dest in set dom phases and
           card phases(dest).contents <
                phases(dest).capacity and
           containers(cid).material in set
                phases(dest).expected_materials
```

不変式1 コンテナIdの条件

Phaseの容量の不変式

不変式の3番目の条件

不変式2はあるか？

核廃棄物処理追跡システム　　コンテナの取り出し

```
Remove: Tracker * Containerid * PhaseId -> Tracker
Remove(mk_Tracker(containers,phases),cid,pid) ==
 mk_Tracker(containers,
    phases ++ {pid |->
        mk_Phase(phases(pid).contents
              ¥ {cid},
        phases(pid).expected_materials,
              phases(pid).capacity)
```

書き換え
オペレータ

不変式2の
保証のため

```
pre pid in set dom phases and
      cid in set phases(pid).contents
```

核廃棄物処理追跡システム　　コンテナの取り出し

- Let式の使用--ローカル変数の使用により関数定義を単純化することができる:

```
Remove: Tracker * Containerid * PhaseId -> Tracker

Remove(mk_Tracker(containers,phases),cid,pid) ==

let pha = mk_Phase(phases(pid).contents ¥ {cid},
                   phases(pid).expected_materials,
                   phases(pid).capacity)

in

mk_Tracker(containers, phases ++ {pid |-> pha})

pre pid in set dom phases and

    cid in set phases(pid).contents
```

核廃棄物処理追跡システム　　コンテナの削除

2つのことを終了させる必要がある:

- コンテナマッピングからコンテナーを取り除かなければならない;
- また、コンテナが現在存在している(ちょうどRemove関数のように)工程からコンテナを取り除かなければならない。

核廃棄物処理追跡システム　コンテナの削除

```
Delete: Tracker * Containerid * PhaseId -> Tracker
Delete(mk_Tracker(containers,phases), cid, pid) ==
 let pha = mk_Phase(phases(pid).contents ¥ {cid},
                    phases(pid).expected_materials,
                    phases(pid).capacity),
     con = {c |-> containers(c) |
            c in set dom containers & c <> cid}
 in mk_Tracker(con, phases ++ {pid |-> pha})
pre pid in set dom phases and
    cid in set phases(pid).contents
```

核廃棄物処理追跡システム　　コンテナの削除

```
Delete: Tracker * Containerid * PhaseId -> Tracker
Delete(mk_Tracker(containers,phases), cid, pid) ==
 let pha = mk_Phase(phases(pid).contents ¥ {cid},
                    phases(pid).expected_materials,
                    phases(pid).capacity),
in mk_Tracker({cid} <-: containers,
              phases ++ {pid |-> pha})
pre pid in set dom phases and
    cid in set phases(pid).contents
```

domainから cid を除く

核廃棄物処理追跡システム　　コンテナの削除

```
Delete: Tracker * Containerid * PhaseId -> Tracker
Delete(tkr, cid, pid) ==
mk_Tracker({cid} <-: containers,
           Remove(tkr, cid, pid).phases)
pre pre_Remove(tkr, cid, source)
```

まとめ

- マッピング　―　領域(domain) と 範囲(range) という2つの集合の間の関数に似た関係をモデル化している。
- マッピングは数え上げあるいは内包によって表現される。
- 以下のような演算子がある
  - 領域および範囲　を取り出す演算子
  - マッピングを結合したり(部分)、上書きしたりする演算子
  - 領域や範囲を絞り込んでマッピングを小さくする演算子