CSC227
Formal Specification of Software(2)

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。

モデルの構築

VDM-SLを用いたモデルのガイドツアー

- モデルの演繹
- 化学プラント警報システム:
  - 要求仕様
  - データタイプと不変式
  - 関数と事前条件

例は、IFAD(デンマークの先端ソフトウェア技術の会社）が市内電話会社Tele Danmark Processのために開発した大規模な警報及び出動システムのサブコンポーネントから取ったものである

VDM-SLにおけるモデルの構成要素

タイプ定義の例

```
Altitude = real
inv alt == alt >= 0

Position :: lat  : Latitude
            long : Longitude
            alt  : Altitude
```

関数定義の例

```
Move: Id * Position * ATCSystem  ->  ATCSystem
Move(id, pos, sys) == … expression …
pre … expression …
```

VDM-SLにおけるモデルの構成要素

- データタイプ(**data types**)は、基本タイプ(整数、実数、文字、ブール)から、タイプ構成子(**type constructors**: セット、シーケンス、写像、レコード)を使用して作成される。
- 新しく構築されたタイプは名前を付けることが出来、モデルの全体にわたって使用することができる。
- データ型不変式(**data type invariants**)は、その式を満足する値だけを含むように、(元の)データ型を制限するために使用されるブール式である。
- 関数(**functions**)は、システムの機能を定義する。関数は、定義されている機能に関して透明である。即ち、副作用およびグローバル変数はない。グローバル変数を使用することが本質的に必要と思われるときは、異なるスタイルのモデル化(**operation**)が使用される。
- 事前条件(**pre-conditions**)は、(関数等への)入力が保持すると想定される制約を記述する為に用いられる入力パラメータに関するブール式である。

VDM-SLにおけるモデルの構成要素

- データにおける抽象は、VDM-SLでは、データ・タイプが制約を持たないことによって実現される。集合、シーケンスおよび写像は、有限であるが、その上限や精度などが設定されているわけではない。（通常の計算機言語には、例えば、整数は2^{31}が最大値であるとか、実質的に種々の制限が存在する。）

- 関数における抽象は、必要なら、インプリシットな仕様によって、実現できる。
  - `SquareRoot(x:nat)r:real`
  - **`pre`** `x >= 0`
  - **`post`** `r*r = x`

- 事後条件(Post-conditions)は入力と出力を関連づけるブール式であり、その関数の実行によって事後条件が成立することを意味している。事後条件は、どのような出力が返されるかを明示的に定義したくない場合や、直接定義があまりに具体的で細かすぎるようなときに用いられる。

ゼロからフォーマルモデルを作成する。

- 要求仕様から形式的モデルを構築するための正しい方法、あるいは、間違った方法というのは無い。
- モデルの目的を良く考えることから始まる。これが、モデルを作成していくときの抽象化の決定における指針となる。

- 次のようなステップを踏む:

1. 要求仕様をよく読む。
2. 想定されるデータ・タイプ(かなりの場合、仕様中の名詞に注目して抽出)および関数/機能(仕様中の動詞/アクションに注目して抽出)のリストを作成する。
3. データ・タイプ表現の概略を記述する。
4. 関数のシグネチャの概略を記述する。
5. 不変式を決定して、データ型定義を完成させる。
6. もし必要ならデータ型定義を修正して、関数定義を完成させる。
7. 要求仕様中の各項目がモデルの中でどう反映されているかを注目して、要求仕様をレビューする。

警報システムの例における要求仕様

- 化学プラントは、プラント中の条件に応じて警報を発することが出来るモニターを持っている。警報が発せられた場合、専門技術者がその場所へ出向く必要がある。専門技術者は、異なる種類の警報に関して対応できるように、異なる専門分野を持っている。

R1: 警報に対して、専門技術者を対応させることを目的とする計算機管理システムの開発が予定されている。

R2: 4つの専門分野が存在する: 電気、機械、生物学、及び化学。

R3: いつでも(複数の)専門技術者が当直で待機している。

R4: 各専門技術者は、1つだけではなく複数の（専門分野の）資格のリストを持つことができる。

R5: 各警報に対して専門分野と警報の意味の記述(専門技術者のためのテキスト)がある。

R6: 警報が発せられた場合、システムから、分野が合っていてそのとき対応可能な専門技術者の名前が出力されること。

R7: 指定された専門技術者がいつ対応可能かチェックすることが可能であること。

R8: 指定された期間中に当番である専門技術者の数を調べることが可能であること。

モデルの目的...

- 輪番勤務当直表作成のルールや、警報に対して専門技術者を呼び出しを決定する基準となるルールの明確化。

- 余談ではあるが（原著者達の経験によると）:
  - 専門家としてのこれまでの経験によると、モデルを作る目的が、本当にまれにしか明快にならない事例に度々遭遇している。
  - しかしながら、その目的を明確化することこそ、モデルを作成していく上で、抽象化を行う場合の選択における決定を助け、ひいてはモデル作成自体の成功や、使いやすさにつながるものである。

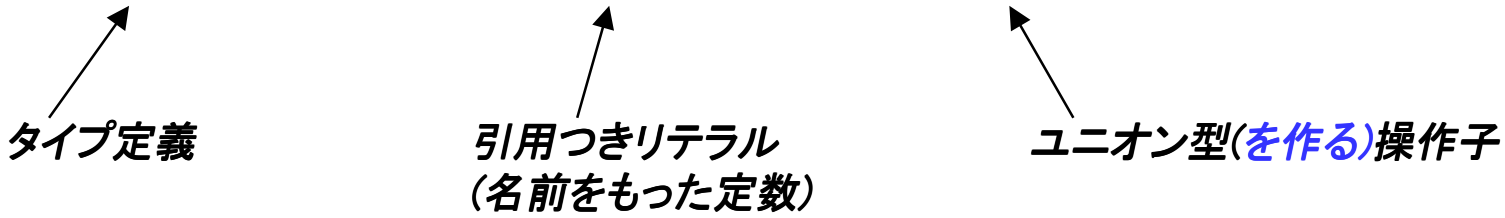
想定されるデータタイプと関数

Types	*Functions*
Plant	ExpertToPage
Qualification	ExpertIsOnDuty
Alarm	NumberOfExperts
Period	
Expert	
Description	

タイプ表現の概略　Enumerated types

R2: 4つの専門分野:　電気、機械、生物学、及び化学

```
Qualification = <Elec> | <Mech> | <Bio> | <Chem>
```



- | はタイプや引用つきリテラルのユニオン型を構築する。
- 個々の引用値は、かぎ括弧 <...> で括られる。
- このタイプは、4つの警報と専門分野に対応する4つのエレメントからなる。
- ちょうど、計算機言語のenumerated typeと考えればよい。

タイプ表現の概要 Record types

R5: 各警報に対して専門分野と警報の意味の記述(専門技術者のためのテキスト)がある。

顧客に、ひとつか、複数か、少なくともひとつかを尋ねる価値がある。

シーケンスタイプ構造作成子

```
Alarm :: alarmtext : seq of char
         quali     : Qualification
```

レコードタイプの構造作成子

レコード内のフィールドの名前

それぞれのフィールドの型

タイプ表現の概要 Record types

```
Alarm :: alarmtext : seq of char
         quali     : Qualification
```

値 v がタイプ Tを持つというときには次のように書く、

```
v : T
```

aが警報であるという宣言は、次の言うに書く

```
a : Alarm
```

レコードからフィールドを抽出するためには、ドットノーテーションを用いる:

```
a.alarmtext
```

aがある値の集まりから成り立っていることを言うには、レコード構成子(**record constructor**) “mk_”を用いる:

```
a = mk_Alarm(“Disaster - get here fast!”, <Elec>)
```

この構成子によってそれらの値をフィールド値とするレコードが作られる。

タイプ表現の概要　　Set types

R4: 各専門技術者は、1つだけではなく複数の資格のリストを持つことができる。

顧客に「本当にリストなのですか、即ち、記載された順序は重要なのですか」と聞く必要がある。

```
Expert :: expertId : ExpertId
          quali    : set of Qualification
```

- 自然言語で与えられた要求仕様は、そこに書かれたことが文字どうり正確な意味では使用されていないことがしばしばある。もし、疑問があれば、権威者や顧客に意見を求めることが必要。
- ここでは、リストは、シーケンスではなくセットの意味で使用されていた。
- できるだけ抽象的に形式的モデルを作成しようと努める。
- モデルの目的の為に必要とする情報のみを記録する。
- 何が関係し、何が無関係かの選択は、特に安全性が関係している場合、重大な技術的判断の問題である。

タイプ表現の概要　Token types

- 非形式的な要求仕様では、我々が専門技術者のIDの中身を検討する必要が無いことを示している。
- このように、タイプ定義は必要であるが、その正確な表現に関しては必要でない場合、特別なシンボルであるtokenを使用する。

```
ExpertId = token
```

- さらにプラント予定表において、分割された個々の期間についても同じことが言える。

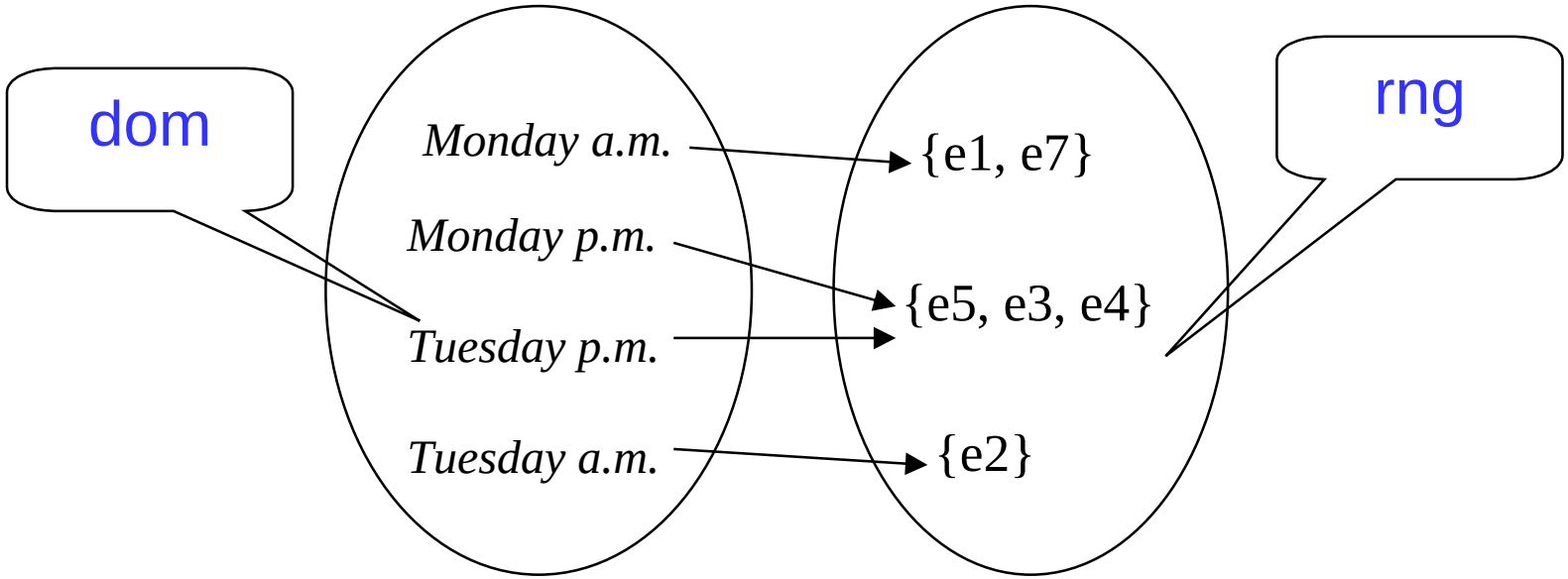
```
Period = token
```

タイプ表現の概要 Mapping types

R3: いつでも(複数の)専門技術者が当直で待機している。
R7: 指定された専門技術者がいつ対応可能かチェックすることが可能であること。

これらの要求仕様は、各期間に対して、その期間に当直でいる専門技術者の集合を関連づけるある種のスケジュールがあるに違いないことを示唆している:

```
Schedule = map Period to (set of Expert)
```



タイプ表現の概要

The whole plant

R1: 警報に対して、専門技術者を対応させることを目的とする計算機管理システムの開発が予定されている。

```
Plant :: sch    : Schedule
         alarms : set of Alarm
```

これまでのところのモデル - タイプ定義

```
Plant :: sch    : Schedule
         alarms : set of Alarm

Schedule = map Period to set of Expert

Period = token

Expert :: expertid : ExpertId
          quali    : set of Qualification

ExpertId = token

Qualification = <Elec> | <Mech> | <Bio> | <Chem>

Alarm :: alarmtext : seq of char
         quali     : Qualification
```

関数シグネチュアの概要

考えられる関数:

```
ExpertToPage
ExpertIsOnDuty
NumberOfExperts
```

関数定義は、入力パラメータと結果のタイプをシグネチュアの形で示す:

```
ExpertToPage: Alarm * Period * Plant -> Expert

ExpertIsOnDuty: Expert * plant -> set of Period

NumberOfExperts: Period * Plant -> nat
```

データタイプ定義を完成させる　Data type invariants

- 対象のシステムおいて、いつでも成立する必要がある値に関しての付随的な制約があるかもしれない。そのような制約はデータ型不変式と呼ばれる。
- 例:専門技術者が少なくとも1つの専門分野を常に持っていることを顧客との間で認識しているとすると、これはタイプExpertに対する制約となる。この制約を記述するために、タイプExpertを一般的に表す値exを考える。:
  - `ex.quali <> {}`
- 我々は不変式を関連するデータ型の定義に付け加える。

```
Expert :: expertid : ExpertId
          quali    : set of Qualification
        inv ex == ex.quali <> {}
```

これはそのタイプの値を一般的に表すパラメータである。

不変式の本体はタイプの値を代表するパラメータに関して制約を記述するブール式である。

データタイプ定義を完成させる　Data type invariants

R3: いつでも(複数の)専門技術者が当番で待機している。

これはすべての期間について、専門技術者の集合が空ではないことを保証するスケジュールに対する制約である。

また、これをフォーマルに述べるために、スケジュール型を一般的に表す変数schを考える。

```
forall exs in set rng sch & exs <> {}
```

“schのrngの中の専門技術者の全てのグループに対して、そのグループは空で無い”

この制約を関係するタイプ定義に付加して:

```
Schedule = map Period to set of Expert
inv sch == forall exs in set rng sch & exs <> {}
```

関数定義を完成させる

- 関数の定義は次のものを含む:
- *シグネチュア*
  - `NumberOfExperts: Period * Plant -> nat`
- *パラメータのリスト*
  - `NumberOfExperts(per,pl) ==`
- *本体*
  - `card pl.sch(per)`
- *事前条件(pre-condition)(任意)*
  - **`pre`** `per in set dom pl.sch`
- もし省略されれば、事前条件は`true`とみなされ、タイプ条件さえ合っていれば任意の入力に関数を適用することができる。

関数定義を完成させる

R7: 指定された専門技術者がいつ対応可能かチェックすることが可能であること。

```
ExpertIsOnDuty: Expert * Plant -> set of Period
ExpertIsOnDuty(ex,pl) ==
     {per | per in set dom pl.sch &
            ex in set pl.sch(per)}
```

- 便宜のため、入力パラメーターの中でレコード構成子(mk_)を使用することができる。それによって、レコードplのフィールドをセレクターを使用すること無しに引用できる:

```
ExpertIsOnDuty: Expert * Plant -> set of Period
ExpertIsOnDuty(ex,mk_Plant(sch,alarms)) ==
     {per | per in set dom sch &
            ex in set sch(per)}
```

関数定義を完成させる

mk_Plant(sch、alarms)のパラメータalarmsコンポーネントは、関数の本体では使用されず、したがって、-と取り替えてもよい。関数の最終バージョンは次のとおりである:

```
ExpertIsOnDuty: Expert * Plant -> set of Period

ExpertIsOnDuty(ex, mk_Plant(sch,-)) ==

   {per | per in set dom sch & ex in set sch(per)}
```

関数定義を完成させる

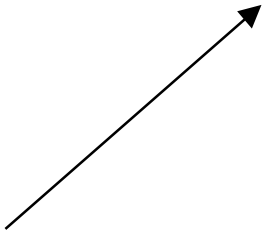
R6: 警報が発せられた場合、システムは分野が合っていてそのとき対応可能な専門技術者の名前が出力されること。

```
ExpertToPage: Alarm * Period * Plant -> Expert
ExpertToPage(al,per,pl) == ???
```

（どのように専門技術者を選ぶか？）

- 我々がどのような方法でそれを見つけるかについて考慮しなくても結果として何を返さなければならないかについて記入することができるだろうか？そのためには陰定義(*implicit definition)*を使用する:

```
ExpertToPage(al:Alarm, per:Period, pl:Plant) r:Expert
pre   …
post  r in set pl.sch(per) and
      al.quali in set r.quali
```



陰定義では、シグネチュアは与えず、ヘッダ表示に入力パラメータのタイプ、および結果を指定するフォーマルパラメータを与える。

あなたは、システムに関する問題を見抜いたか

要求仕様は、正しい専門分野を持つ専門技術者が常にいる保証に関しては何も言っていない。顧客と面談した後に、各種類の専門分野に関して、少なくとも1つの専門家が常にいることを保証することが必要であることが判明した。どのようにこの事実をモデルに反映させるか？

```
Plant :: sch    : Schedule
         alarms : set of Alarm
inv mk_Plant(sch,alarms) ==
   forall a in set alarms &
      forall per in set dom sch &
         exists ex in set sch(per) &
            a.quali in set ex.quali
```

最終的に、要求仕様をレビューする

R1: 警報に応じてエキスパートの呼び出しを管理するシステム。

R2: 4つの専門分野

R3: いつでも専門技術者が待機する。

R4: 専門技術者は、専門分野をリストとして持つことができる。

R5: 各警報はその内容の記述と対応する専門分野を持っている。

R6: その専門であり、呼び出し可能な専門技術者の名前を出力する

R7: 指定された専門技術者がいつ呼び出し可能かチェックする

R8: 指定された期間に当直である専門技術者の数を調べる。

要求仕様の問題点

- その少なくとも一人の適切な専門技術者が呼び出し可能か否かの保証には言及されてなかった。
- 専門技術者の識別子の使用は明示的ではなかった。
- “list"は実際に"set"を意味した。
- 専門分野の無い専門技術者が意味が無いという事実には、言及されてなかった。
- 「ひとつの専門分野」は、「いくつかの専門分野」を意味した。

まとめ

モデルを作成する過程は、モデルの目的に、決定的に依存する。

VDM-SLモデルは、タイプ定義と関数定義を基本とする。

抽象化は、基礎的なデータタイプおよびタイプ構成子、および陰定義可能な関数を用いることによって実現される。

- **Basic types:**
- **Type constructors:**
- **Invariants:**
- **Functions:**