SEAMAIL

Newsletter from Software Engineers Association

Vol. 12, Number 12 October, 2001

目　次

編集部から		1
sigfmセミナー「形式手法の魅力とは何か？」プログラム		2
形式手法のBからZまで	玉井 哲雄	3
応答系の仕様化と設計に向かって	山崎 利治	21
フォーマルメソッドの実践方法	荒木 啓二郎	30
鉄道信号に於けるソフトウェア安全性技術とフォーマルメソッド	荻野 隆彦	40
形式仕様記述言語はどう動くか	澤田 寿実	50
事務システムにおける形式仕様適用例	佐原 伸	68
UML state chart図と振舞仕様を使ったシステム設計	飯田 周作	101
ISFST2001 (in 中国・鄭州市) 参加者募集		107

ソフトウェア技術者協会
Software Engineers Asociation

ソフトウェア技術者協会 (SEA) は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者または研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985年 12月に設立されました．

その主な活動は，機関誌 SEAMAIL の発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約 200人にすぎなかった会員数もその後増加し，現在，北は北海道から南は沖縄まで，500 余名を越えるメンバーを擁するにいたりました．法人賛助会員も 26社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州，広島，東北の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

「現在のソフトウェア界における最大の課題は，技術移転の促進である」といわれています．これまでわが国には，そのための適切な社会的メカニズムが欠けていたように思われます．SEA は，そうした欠落を補うべく，これからますます活発な活動を展開して行きたいと考えています．いままで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために，ぜひとも，あなたのお力を貸してください．

代表幹事： 深瀬弘恭

常任幹事： 荒木啓二郎 高橋光裕 田中一夫 玉井哲雄 中野秀男

幹事： 伊藤昌夫 大場充 落水浩一郎 窪田芳夫 熊谷章 小林修 桜井麻里
酒匂寛 塩谷和範 篠崎直二郎 新谷勝利 新森昭宏 杉田義明 武田淳男
中来田秀樹 野中哲 野村行憲 野呂昌満 端山毅 平尾一浩
藤野誠治 松原友夫 山崎利治 和田喜久男

事務局長： 岸田孝一

会計監事： 橋本勝 吉村成弘

分科会世話人 環境分科会(SIGENV)：塩谷和範 田中慎一郎 渡邊雄一
教育分科会(SIGEDU)：君島浩 篠崎直二郎 杉田義明 中園順三
ネットワーク分科会(SIGNET)：人見膺 松本理恵
プロセス分科会 (SEA-SPIN))：伊藤昌夫 坂本啓司 高橋光裕 田中一夫 端山毅 藤野誠治

支部世話人 関西支部：臼井義美 小林修 中野秀男 横山博司
横浜支部：野中哲 藤野晃延 北條正顕
長野支部：市川寛
名古屋支部：筏井美枝子 石川 雅彦 角谷裕司 野呂昌満
九州支部：武田淳男 張漢明 平尾一浩
広島支部：大場充 佐藤康臣 谷純一郎
東北支部：河村一樹 布川博士 野村行憲 和田勇

賛助会員会社：ジェーエムエーシステムズ 東芝アドバンストシステム ＳＲＡ ＰＦＵ
東電ソフトウェア 構造計画研究所 富士通 オムロンソフトウェア
キヤノン 中央システム 富士通エフ・アイ・ピー 新日鉄ソリューションズ
ダイキン工業 東北インフォメーション・システムズ オムロン ＳＲＡ西日本
富士電機 ブラザー工業 オリンパス光学工業 リコー
アルテミスインターナショナル NTTデータ ヤマハ 福井コンピュータ
日本ネスト オープンテクノロジーズ （以上26社）

ＳＥＡＭＡＩＬ Ｖｏｌ．１２，Ｎｏ．１２ ２００１年１０月１日発行
編集人 熊谷章
発行人 ソフトウェア技術者協会（ＳＥＡ）
〒１６０-０００４ 東京都新宿区四谷３－１２ 丸正ビル５Ｆ
T: 03-3356-1077 F: 03-3356-1072 sea@sea.or.jp
印刷所 有限会社 錦正社 〒１３０-００１３ 東京都墨田区錦糸町４－３－１４
定価 １，０００円 （禁無断転載）

編集部から

今年の夏は暑かった。特に夏の初めの暑さは尋常でなかった。そのお蔭で農作物の出来が良い。特に果物は美味しく嬉しい限りである。何が幸いするか分らないものだ。桃、葡萄、梨などみな甘くて旨い。人間も自然の産物であるから、美味しくて出来の良いものがたくさん輩出されることを期待している。

というような暢気なことを考え果物を食べてテレビを見ていたら、アメリカのテロ惨劇の映像が映し出された。どのテレビ番組もこぞって旅客機がワールドトレーディングセンターに衝突し爆破する映像を何度も繰り返したので、その衝撃と共にあのシーンは生涯記憶から消える事はないだろうと思う。衛星放送でアメリカからの第一報がケネディ暗殺事件を運んできたときと同じような衝撃を受けた。ご多分に洩れずそれからは深夜に渡り、全世界が報じるニュースを見ることが日課になった。被害の凄惨さ、テロの実態の詳細、アメリカ市民の反応、アメリカ政府の見解と対応の仕方、日本政府のメッセージ、各国のメッセージなど興味が尽きなかった。非常事態における言動が本音だと理解しているからだ。

アメリカのブッシュ大統領を始めとしたメッセージの変遷をみれば、世論は徐々に誰かの意図に基いて作られていくものだということが良く分かった。例えば、“Acts of War”と言い出したのは、事件発生から一両日経ってからだし、大統領が搭乗していた飛行機が攻撃の対象だったというのはアメリカ政府が捏造したものだなどが挙げられる。そんなことは昔から日常茶飯事だといわれそうだが、テレビのライブ中継とインターネット経由のリアルタイムコミュニケーションは、身心にズシリと応える影響を及ぼすと感じた。インターネット経由でFBIが事件に関する情報収集をしており、この私でも該当Web Home Pageから情報や意見を提供できるのだ。

すべての国と団体がテロ攻撃に反対し、テロの撲滅に最大の努力を惜しまないと宣言している。それは、とても素直な反応であるが、表層的に聞こえる。人類が考えた今の社会制度がテロ現象を生み出していると考え、その原因を対症療法ではなく、もっとメタフィジカルなレベルで思考し、解決策を講じるのが我々の責務だろうと考えている。しかし、なにかしっくりしない。行方不明の家族や愛する人々を探している様子を映像と文から知ると悲哀が湧いてきてしょうがない。事件後の真犯人捜し、報復作戦、世界経済の動向、日本不況への影響などの才夥しい議論を見聞する度に憤りを感じるようになった。メタフィジカルなレベルでの思考や議論が一向に行われず、報復戦争用の予算や寄付などの金がすべてを決めるような論理だからだ。金で人類が救われるならば、輪転機を回せばよい。

と言うわけで、今回は現場とメタフィジカルのギャップを埋め、21世紀のキーテクノロジであるフォーマルメソッドのセミナーをご紹介する。ご堪能下さい。(くま)

sigfm セミナー　形式手法の魅力とは何か？

主催　ソフトウェア技術者協会フォーマルメソッド分科会(sigfm)
期日　２００１年５月２４日（木）－２５日（金）
場所　川崎市産業振興会館　第三研修室

形式手法（フォーマルメソッド）がソフトウェア開発の最大の要という共通認識があるが、現実的にあまり普及されていない。２１世紀の始めとて、新世紀において重要な技術は何かという特集が様々な専門誌で組まれたが、ソフトウェア関係では必ずフォーマルメソッドが核になる技術であると多くの者が言明している。今回のセミナーでは、フォーマルメソッドがなぜ今後のソフトウェアの核技術であり、どのような可能性を秘め、それが今後どのように実社会で使用されていくかを分かり易く説明することを目指した。講師の方々には、ここ一年間研究会に参加し討議を共にした学界と産業界の一線級の専門家に出演をお願いした。

プログラム

5月24日　　**チュートリアル**（9：00-17：00）
「形式手法のBからZまで」玉井哲雄（東京大学）
「応答系の仕様化と設計に向かって」山崎利治（sigfm 世話人）
「フォーマルメソッドの実践方法」荒木啓二郎（九州大学）

5月25日　　**事例研究とパネル討論**（9：00-17：00）
9時−10時　「鉄道信号に於けるソフトウェア安全性技術とフォーマルメソッド」
　荻野隆彦　（(財)鉄道総合技術研究所）
10時−11時　「形式仕様記述言語はどう動くか」澤田寿実（SRA）
11時−12時　「事務システムにおける形式仕様適用例」佐原伸（JFITS）
13時−14時　「UML state chart 図と振舞仕様を使ったシステム設計」
　飯田周作（専修大学）
14時−17時　パネル討論「21世紀のフォーマルメソッドの役割」
　司会：熊谷章（PFU）

パネラー：山崎利治・荒木啓二郎・飯田周作・佐原伸・張漢明
次期システム開発技術では、①高信頼性システムを事前に予測可能なコストと機能で開発する方法、②グローバルでスケーラブルなシステムの開発方法、③システムアーキテクチャを動的でかつアダプタブルにするメカニズム、が必要だといわれている。この観点からフォーマルメソッドの役割、機能、やり方、効用、課題を論ずる

形式手法のBからZまで

SIGFM　セミナー

2001年5月24日
玉井哲雄
（東京大学大学院情報学環）

形式手法についての個人的な想い

- 世の中は，FMへの「アレルギー反応」と「偏愛」とに二分
- 自分はambivalent（愛憎相半ば）
  - 形式論理のきちんとした修練を経ていないという負い目
    - 野矢茂樹「論理学」
  - 思考の節約に価値
    - 形式と非形式間の自由な飛躍
    - 抽象のレベルの自由な昇降
  - 壷に嵌まった形式化の美しさ

形式化が呼び起こす好悪感

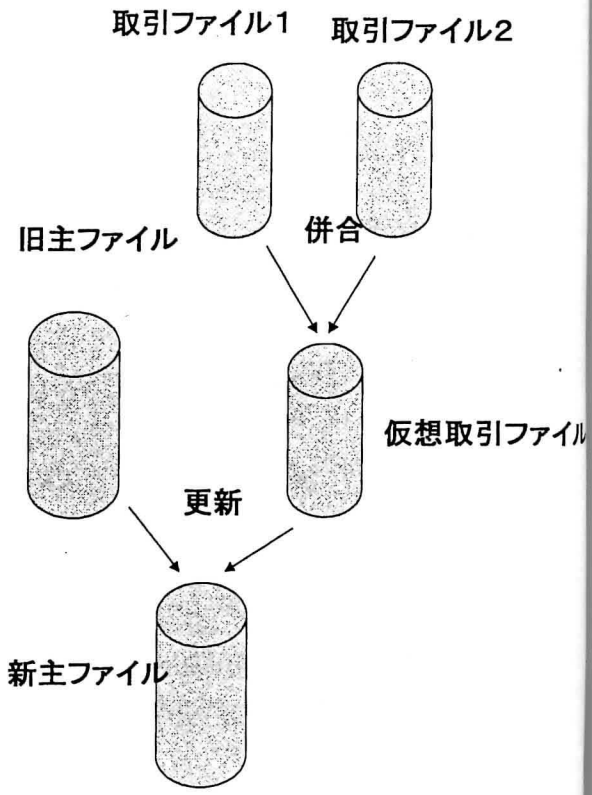
- （とくに日本語で）「形式的」 ⇒ 中身がない
- 「人の思考は形式的でない」
  - 形式的推論を追うことはできるが面倒
  - モデルに引きつけて理解
- 「抽象化＝形式化ではない」
  - 形式化は抽象化である.
    - 記号化で対象/意味を捨象
  - 非形式的な抽象化もある.

形式手法の経験（1）
ファイル処理プログラムの検証系開発

- 玉井・福永，1977－1980
- 技術的には
  - 半順序に特化した推論機能を持つ定理証明系
  - ループ不変量の自動生成
- より有益だと思われるアイディア
  - 仮想ファイル
    - 同じインターフェースに対する複数の実現
    - 同じ計算対象に対する複数のインターフェース

仮想ファイル

- 背景:n本ファイルの併合，更新，等の処理
  - 例:主ファイルを2本の取引ファイルで更新
    ⇒2本の取引ファイルを併合した仮想ファイルを導入
- ファイルへの操作インタフェース(open, read, eofなど)を定義
- 2本のファイルを実際に併合せず，それへの操作のみを公理を満たすように実現
- 関連する技術
  - ADT
  - ストリーム・プログラミング
  - Javaのimplement句



複数のインターフェース(視点)

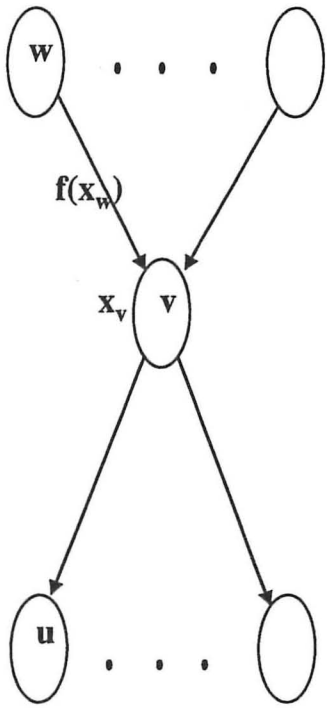
- ファイル＝レコードの並び
  ＝レコード・グループの並び
  ⇒ 「キーの切れ目処理」
- テキスト＝文字の並び
  ＝単語の並び
  ＝行の並び
  ⇒ D. Jacksonのエディタの仕様
- ここでも仮想ファイルの概念が使える.

形式手法の経験(2)
多様な問題の
統一的な扱い

- 共通構造をもつ問題への遭遇
  - 大規模線形計画問題の構造化(行列の構造変換)
  - 整数計画法(分枝限定法)
  - ネットワーク計画問題(最短路，最大流，最小費用流，多種流)
  - プログラムの静的解析(データフロー解析)
  - プログラムの記号実行
  - AIの探索問題(A*アルゴリズム)

グラフ上の不動点問題として統合化

- グラフと束と束の上の関数空間を用い，不動点方程式として定式化
- アルゴリズムの統一的記述と，正当性の証明
- 山本・萩谷等がHOLで記述，検証
- CafeOBJでも記述．パラメータ・モジュールで個別問題領域に写像．



モデル検査

- 最近はやり (FM99, ICSE2001)
- もとは1980年代初め．Clarke & Emerson'82など
- モデルが時相論理式を満たすかどうかを自動的に検査
- アルゴリズムはグラフ上の不動点計算
- ハードウェア(論理回路)の検証では日常的に使われている．
- セキュリティ・プロトコルの検査で注目された．

EJBフレームワークの振舞い仕様の分析

- S. Nakajima & T. Tamai (SPIN Workshop '01)
- EJBの仕様をモデル記述言語Promelaで記述
- 確かめたい性質（beanの機能のクライアントによる起動と，サーバーによる管理との干渉による弊害の有無）のLTLによる記述
- SPINによる検査

形式手法の意味

- 曖昧性のない厳密な仕様
- 正しさや性質の検証
- “実行”(シミュレーション)
- プログラムの自動合成

本当に実用性があるか？

- 形式手法の利点は広く認識されつつある.
- 実際的な適用例も増えてはきたが，まだ限定的
  - いつも同じ事例 (たとえばCICS)
  - 効果が明らかなのは
    - ハードウェア(論理回路)
    - 通信プロトコル
  - 上の分野ではモデル検査が活躍

ソフトウェアでは

- 従来は高信頼性が要求されるもの
  - 安全性，機密性
- これからは再利用性の高いもの／
  広い範囲で繰り返し使われるもの
  - ライブラリ，コンポーネント
  - スタンダード
  - 一般性の高い問題と解法

形式的抽象化の方法

- 機能の抽象化
- データの抽象化
- オブジェクトの抽象化
- 時間(順序，並列性)の抽象化

Zの教科書的な導入の仕方

- 状態機械の仕様記述
  - Spiveyの「お誕生日ノート」
- よく使われる表現

Zのくせ

- すべては基本集合の上で作られる.
  - M. Jacksonの批判: 時間的に不変, 互いに素という仮定
- 部分関数の多用
  ← 関数や関係の集合的な定義
- スキーマの引用 ⇒ 簡潔で暗黙な表現 (その意味で分かりにくい)

山崎さんの酒屋問題

ある酒類販売会社の倉庫では，毎日数個のコンテナが搬入
されてくる．その内容はビン詰めの酒で，1つのコンテナには10銘柄まで混載できる．扱い銘柄は約200種類ある．倉庫係は，コンテナを受け取りそのまま倉庫に保管し，積荷票を受付係へ手渡す．また受付係からの出庫指示によって内蔵品を出庫することになっている．内蔵品は別のコンテナに詰め替えたり，別の場所に保管することはない．
空になったコンテナはすぐに搬出される．

積荷票: コンテナ番号(5桁)
搬入年月，日時
内蔵品名，数量(の繰り返し)

酒屋問題(2)

さて受付係は毎日数十件の出庫依頼を受け，その都度倉庫係へ出庫指示書を出すことになっている．出庫依頼は出庫依頼票または電話によるものとし1件の依頼では，1銘柄のみに限られている．在庫がないか数量が不足の場合には，その旨依頼者に電話連絡し，同時に在庫不足リストに記入する．また空になるコンテナを倉庫係に知らせることになっている．倉庫内のコンテナ数はできる限り最小にしたいと考えているからである．

出庫依頼: 品名，数量
送り先名

酒屋問題(3)

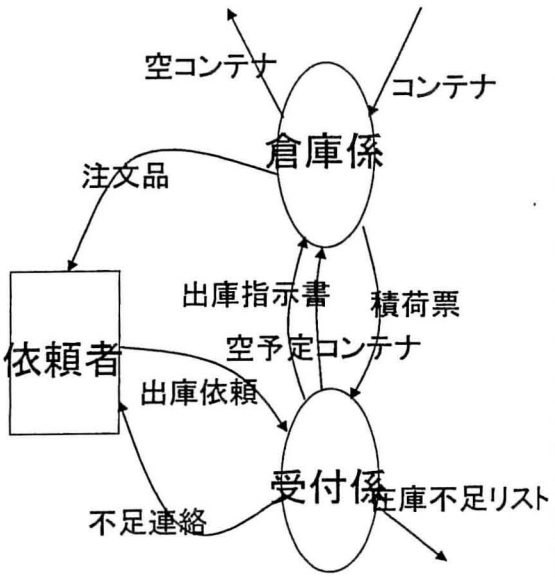
受付係の仕事(在庫なし連絡，出庫指示書作成および在庫不足リスト作成)のための計算機プログラムを作成せよ．

出庫指示書： 注文番号
送り先名
コンテナ番号
品名，数量
空コンテナ搬出マーク
}(の繰返し)

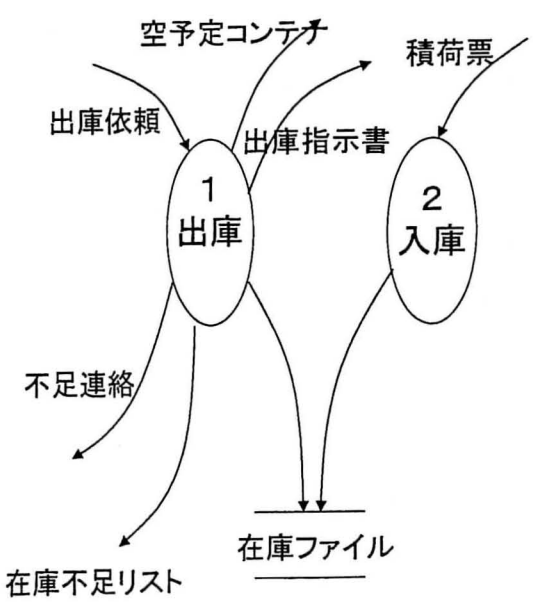
在庫不足リスト： 送り先名
品名，数量

- なお移送や倉庫保管中に酒類の損失は生じない．
- この課題は現実的でない部分もあるので，入力データのエラー処理などは簡略に扱ってよい．

酒屋問題の全体文脈図



酒屋問題 トップレベル図



Zによる酒屋問題の記述

- ❷ 高階関数の利用
- ❷ 非決定性の利用
- ❷ 宣言性の利用
  (入力を出力の関数として表す)

形式手法のBからZまで： Zをどう教えたらよいか？

5.24.2001 玉井哲雄

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．

¬ 否定
∧ 連言
∨ 選言
⇒ 含意
∀ 全称
∃ 存在

全称束縛の論理式は，一般に次のように書かれる．

∀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．

$[X, Y, Z]$

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

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

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．

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

(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．

$[X, Y, Z]$

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

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

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 $\pfun$ (品名 $\pfun$ 数量)

酒在庫 : 品名 $\pfun$ (コンテナ番号 $\pfun$ 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．$f : X \pfun Y$ 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，$f : X \pfun Y$ 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 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 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，

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

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

酒在庫 = *curry* ∘ *swap* ∘ *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 ⇸ 数量
---
cn? ∉ dom コンテナ
コンテナ′ = コンテナ ∪ {*cn*? ↦ *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
$cn? \notin$ 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

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

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′ = ∅ 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 ∧ と ∨ 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 ∧ の場合は 2 つのスキーマの公理部の連言が，∨ 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 ∧ 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 ≙ (入庫 ∧ 正常終了) ∨ 番号重複

と書ける．ここで ≙ 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．

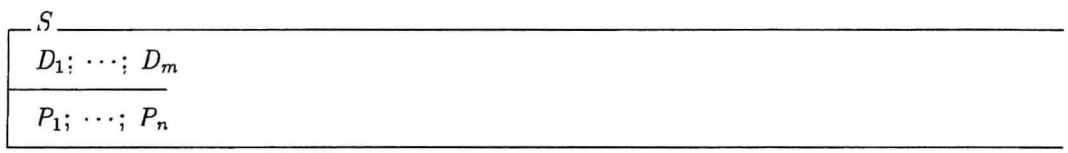
以下で，これらのスキーマを記述していこう．まず，基本的なデータとして，出庫依頼と出庫指示書を定義する．そのために，基本型として，“送り先名” を追加しておく．

[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，



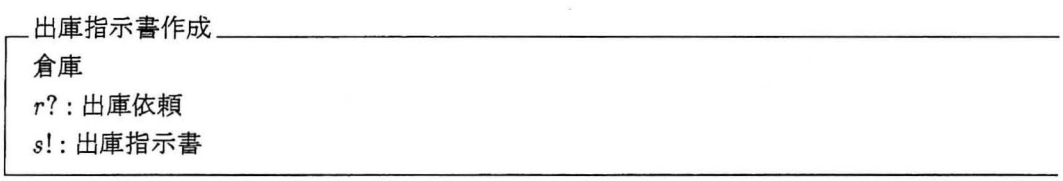
と，横書き表記

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．

$[X]$

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

ここで，$(\mathrm{dom}\ g) \lhd f$ 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

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

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 scq X は X の要素からなる列 (scqucncc) 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，

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

在庫不足処理 0

倉庫
Δ在庫不足管理
$r?$: 出庫依頼

$r?.b \notin$ dom 酒在庫 $\lor$ ($r?.b \in$ 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 ∧ 充足報告
在庫不足処理 ≙ 在庫不足処理 0 ∧ 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．

応答系の仕様記述

山崎　利治

はじめに

ワープロや組込系などの応答系はそのおかれた環境と交信しながら一定の機能をはたすシステムである．これは変換系と呼ばれることもある伝統的なシステムと違った側面が多く仕様記述や設計にも別の工夫が必要になる．これは，L. Lamportによる時間論理の体系であるTLA[2]をその応答系の仕様記述に適用する試みの紹介である．

1　応答系とは

変換系 (tranformational system) は入力データを与えプログラムを始動しその停止とともに出力データを得るものであり，プログラムは必ず止まることを要請している．変換系は入力と出力をもち，出力は入力の関数とみることができる．したがって，プログラムの仕様は関数定義，あるいは，前件 (precondition) と後件 (postcondition) の組を与えればよかった．

これに対して応答系 (reactive system) は系の利用者などの外部環境と相互作用を継続しつつ一定の機能を提供するものであり，つぎのような特徴をもっている．

1．　継続性（非停止性）
2．　対話性
3．　並行性

応答系のプログラムに対しては必ずしも停止することを期待していない．系がもっている時計が止まっては困るわけである．このことは，前件と後件との組としてプログラムの性質が記述できないことを意味し，変換系で行ったHoare論理などによるプログラムの性質の推論がそのままではできなくなっている．

応答系は環境と交信しながらその機能を果たすのでプロセス間通信が問題になる．交信が同期か非同期かまたそのプロトコルなどを考えなければならない．

応答系は交信問題などによって並行プロセスとして実現されることが多い．その実現が純粋の同時並行でなく差込 (interleaving) による並行性の場合もあるが，並行性が引き起こす非決定性が問題になる．

このような性質をもつ応答系の仕様をつぎのように考えてみる．応答系をその許された挙動の集合とみなす．そこで，挙動を外からみえる系の状態 (状態とは系あるいはプログラムのもつ変数にその値を割り付ける関数である) 成分の変化列として記述する．外からみえない状態成分も便宜上記述したい場合はそれを許してあとで隠蔽すればよい．その状態列を時間論理の述語として表現し，それを仕様と考えるのである．

変換系を議論する機構として抽象プログラム機械があったが，応答系の場合には推移系 (transition system) が役立つ．(札付き) 推移系とは，状態と札の集合 S, A があるとき，写像 $\theta : S \to \mathbb{P}(A \times S)$($\mathbb{P}(X)$ は X の部分集合全体を表す) をいう，つまり，θ は状態 s から札 a によって推移する状態 t たちを示すものである．

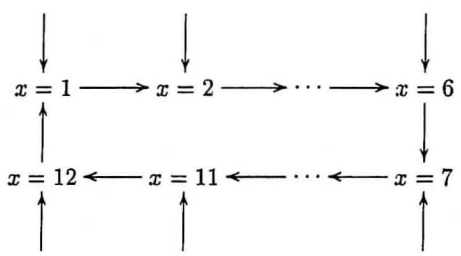
2　例1

例をみてみよう．つぎは時間だけを表示する時計のプログラムである．これは止まっては困るも

のである．

```
...
const hr ∈ {1..12}
{ x = hr }
do  true → x := (x mod 12) + 1
od
...
```

ここで，プログラム仕様として後件を $\{false\}$ とするのは意味がない．そこでプログラムを推移系と考えればつぎの推移図が書ける．



さて，つぎの状態の無限列はプログラムの実行列になっている．

$\sigma \triangleq x = 11, x = 12, x = 1, \cdots$

状態の無限列を挙動 (実行列) ということにしよう．状態列で，それぞれ連続した 2 状態を作用 (行動，事象，推移) などといい，前状態と後状態の関係として記述する．たとえば，

$Next \triangleq x' = (x \bmod 12) + 1$

とする．ただし，x' は後状態を参照する．また，初期状態は

$Init \triangleq x \in \{1..12\}$

である．とすると挙動は

$StateSequence \triangleq Init \wedge \Box Next$

と表せ，これを時計プログラムの仕様と考えることができる．つまり，つぎのように考えるのである．ここに挙げたような述語 $StateSequnce$ を挙動の上で解釈してその真偽を定める．プライムなしの変数と定数記号による論理式を状態述語といい，この真偽は挙動の最初の状態の上で定める．$Init$ は上の挙動 σ に対してその最初の状態 $x = 11$ で真である．したがって，上の σ に対して真となる．プライムなしおよびプライムつきの変数を含む論理式を作用述語といい，連続した 2 状態の組でその真偽を定める．$\Box$ は時間論理の「常に」演算子である．$\Box Next$ は，$Next$ が作用述語であるから続いた 2 状態に対して真偽が定まるが「常に」演算子をつけた場合，それを時間述語というが，すべての連続 2 状態に対して真の場合に真と定める．$\Box Next$ は上の σ に対しては真となる．つまり，順序対列

$(x = 11, x = 12), (x = 12, x = 1), \cdots$

を考えれば，そのそれぞれに対して $Next$ は真となる．

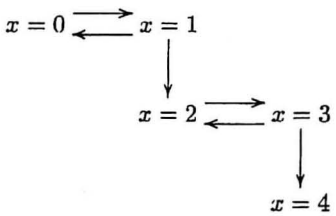
時間述語の厳密な解釈やプログラムの仕様としてこれで十分かなどについてはいまは問わない．

3 例 2

つぎのプログラムは非決定性を含む例である．

```
{ x = 0 }
do  x ≤ 3 → x := x + 1
□   odd(x) → x := x − 1
od
```

これもつぎのような推移図が書ける．



またプログラムの実行に伴う状態列の全体は正規表現風に

$(AB)^\omega + (AB)^+(CD)^\omega + (AB)^+(CD)^+E^\omega$

と書いていいだろう，ただし，$A, B, \cdots$ は $A : x = 0$, $B : x = 1$, $C : x = 2$, $D : x = 3$, $E : x = 4$ を表している (有限状態列となるものに対してはその後に最終状態を無限個連接して無限列とする)．例 1 のように考えればプログラムの仕様がつぎのように記述できる．

$Init \triangleq x = 0$
$Inc \triangleq x \leq 3 \wedge x' = x + 1$
$Dec \triangleq \mathrm{odd}(x) \wedge x' = x - 1$
$Next \triangleq Inc \vee Dec$
$Spec \triangleq Init \wedge \Box Next$

ここで非決定性に付随する公平性について補足する．挙動 $ABABABAB\cdots$ は公平さを欠くといえる．状態 B から状態 C へ推移する機会，つまり，作用 Inc が実行される機会が断続的ではあるが無限回あるにも拘わらず一度もそうなっていないからである．
普通，公平さはつぎのように定義されている．
挙動 σ が作用 A に対して弱公平であるとは，A を実行できる状態が無限回連続して生起していれば必ず A が実行されているときにいい，連続しての条件を除いたとき強公平という．
$ABABAB\cdots$ は作用 (B, C) に関して，また，$ABABCDCDCD\cdots$ は作用 (D, E) に関して強公平 (無論弱公平) ではない．すなわち，それらの作用を実行できる状態が無限回起こっているのに実行されていないからである．強公平を保証すれば例題プログラムは必ず E 状態に到達するだろう．

4 TLA

ここで例1,2で試みたような仕様記述を実務で使うために Leslie Lamport の TLA(Temporal Logic of Actions)[2] を紹介する．これはプログラム（システム，成分）の仕様記述と推論のための離散線形時間論理体系である．TLA による仕様は対象のもつ変数の時間的変化を記述するものであり，変数，状態，状態の対である作用，状態の無限列である挙動を基本概念とする．とくに，作用（行動，事象，推移）は TLA を，いわゆる，stuttering 不変な言語にするための鍵になっている．この性質は仕様作成，実現に際して必要な抽象・洗練，分解・合成などの側面において重要な役割を果たすものである．

4.1 構文

まず，TLA 構文を述べる． 基本式としてつぎを定義する．

$c, \cdots$	定数記号
$m, \cdots$	時間非依存変数 (rigid variable)
$x, x', \cdots$	(時間依存) 変数 (flexible variable)(プライムなし，プライムつき)
$f, f', \cdots$	状態関数 (state function)(プライムなし変数，時間非依存変数，定数記号を用いた (非論理) 式)
$p, p', \cdots$	状態述語 (state predicate)(プライムなし変数，時間非依存変数，定数記号を用いた論理式)
$A, \cdots$	作用 (action)(プライムなし，プライムつき変数，時間非依存変数，定数記号を用いた論理式)

基本式の上に **TLA 式** (時間述語)(TLA formula, temporal predicate) をつぎのように定義する (これは stuttering 不変のために通常の時間論理式に比べ著しく制限したものになっている)．

$F, G ::= p \mid \mathrm{Enabled}A \mid \Box[A]_f$
$\quad \mid F \wedge G \mid \neg F \mid \Box F \mid \exists m : F \mid \exists x : F$

便宜上つぎの省略形を定義しておく．
$F \vee G \triangleq \neg(\neg F \wedge \neg G)$
$F \Rightarrow G \triangleq \neg(F \wedge \neg G)$
$\mathrm{Unchanged}f \triangleq f' = f$
$\Diamond F \triangleq \neg\Box\neg F$
$[A]_f \triangleq A \vee (f' = f)$
$\langle A\rangle_f \triangleq A \wedge (f' \neq f)$
$\mathrm{WF}_f(A) \triangleq$
$\quad \Box\Diamond\neg(\mathrm{Enabled}\langle A\rangle_f) \vee \Box\Diamond\langle A\rangle_f$
$\mathrm{SF}_f(A) \triangleq$

$\Diamond\Box\neg(\mathrm{Enabled}\langle A\rangle_f) \vee \Box\Diamond\langle A\rangle_f$

4.2 意味

TLA 式の意味を以下に定義する．順順に定義していく．

$d, \cdots$	値 例 -1, 22, $''abc''$, $\{1..12\}, \cdots$
$\xi \cdots$	時間非依存変数解釈 (時間非依存変数と定数記号から値への関数)
$s, t, \cdots$	状態 (変数から値への関数)
$\sigma, \tau, \cdots$	挙動 (状態の無限列)

以上をもとに，状態の対に対する基本式の意味をつぎのように定義する．

$[\![c]\!](s,t) \triangleq \xi(c)$
$[\![m]\!](s,t) \triangleq \xi(m)$
$[\![x]\!](s,t) \triangleq s(x)$
$[\![x']\!](s,t) \triangleq t(x)$
$[\![f]\!](s,t) \triangleq f(\forall v : s(v)/v)$
$[\![f']\!](s,t) \triangleq f(\forall v : t(v)/v)$
$[\![p]\!](s,t) \triangleq p(\forall v : s(v)/v)$
$[\![A]\!](s,t) \triangleq A(\forall v : s(v)/v, t(v)/v')$
$[\![\mathrm{Enablcd}A]\!](s,t) \triangleq \exists u : [\![A]\!](s,u)$

$(\forall\, v : \cdots/v, \cdots/v')$ はすべての変数 v を $\cdots$ で置換したものを表す．
つぎに挙動に対する TLA 式の意味をつぎのように定義する．

$[\![p]\!]\sigma \triangleq [\![p]\!](\sigma_0, \sigma_1)$
$[\![\mathrm{Enabled}A]\!]\sigma \triangleq [\![\mathrm{Enabled}A]\!](\sigma_0, \sigma_1)$
$[\![F \wedge G]\!]\sigma \triangleq [\![F]\!]\sigma \wedge [\![F]\!]\sigma$
$[\![\neg F]\!]\sigma \triangleq \neg[\![F]\!]\sigma$
$[\![\Box F]\!]\sigma \triangleq \forall n \in \mathbb{N} : [\![F]\!]\sigma^n$
$[\![\exists m : F]\!]\sigma \triangleq \exists c \in Val : [\![F]\!]\sigma$
$[\![\exists x : F]\!]\sigma \triangleq \exists \rho, \tau : (\natural\sigma = \natural\rho) \wedge (\rho =_x \tau) \wedge [\![F]\!]\tau$

以上の定義で σ_i は挙動 σ の i(0 から始まる) 番目状態を指し，σ^n は σ の σ_n に始まる無限列 $(\sigma_n, \sigma_{n+1}, \cdots)$ を指す．$\natural, =_x$ はつぎによって定義するが，$\natural\sigma$ は挙動 σ のなかで連続した同一状態有限列を単一状態に置換した挙動，ただし，前半の有限列を除いて全部同一の状態列はそのままとした挙動のことであり，$\sigma =_x \tau$ は挙動 σ と τ の状態対 $(\sigma_i, \tau_i)(\forall i \in \mathbb{N})$ が x に対する値を除いて同一であることを主張するものである．

$$\natural\sigma \triangleq \begin{cases} \sigma & \text{if } \forall n \in \mathbb{N} : \sigma_n = \mathrm{hd}\sigma \\ \natural(\mathrm{tl}\sigma) & \text{if } \mathrm{hd}\sigma = \mathrm{hd}(\mathrm{tl}\sigma) \\ \mathrm{hd}\sigma \circ \natural(\mathrm{tl}\sigma) & \text{otherwise} \end{cases}$$

$\sigma =_x \tau \triangleq \forall\, n \in \mathbb{N} : \forall\, v \neq x : [\![v]\!]\sigma_n = [\![v]\!]\tau_n$
つぎに式とその意味について若干補足する．
状態述語 p がある挙動に対して真であるとはその挙動の最初の状態に対して真であるときまたそのときである．
任意の作用 A に対して，述語 Enabled$A$ を，それがある状態で真であるのはその状態からはじまる A 段階が存在するときであると定義する．例 $\mathrm{Enabled}(y = x' + n) \equiv \exists c : y = c + n$
演算子 $\Box$ は時間論理の常に，いつも，今後ずっと **(always,henceforth)** 演算子である．すなわち，$\Box F$ は F が未来に対して常に真であることを意味する．具体的には，挙動 σ が $\Box F$ を満たすとは，すべての $n \in \mathbb{N}$ に対して σ^n が F を満たすことである，つまり，σ のすべての状態が F を満たすのである．
演算子 $\Diamond$ は時間論理のいつか，やがて **(sometime, eventually)** 演算子である．$\Diamond F$ はやがていつか F が真になるときがくることを意味する．具体的には，ある $n \in \mathbb{N}$ が存在して σ^n が F を満たす，つまり，n 時点では F を満たすというものである．
$\Box[A]_f$ は未来において隣り合ったすべての状態の対が作用 A かあるいは $f' = f$ を満たすことを意味する．これは作用 A のうちに状態関数 f を変えない部分段階の存在を許すというもので，仕様のモデルは洗練された仕様のモデルでもなければな

らないという要請に基づくものである.
式$\exists x.P$はPが成立するxの値を選択できる方法があることを意味しxを隠蔽する目的をもつ.

5 安全性と活性

仕様作成において安全性と活性(応答性)の記述が一番大切である. 安全性 **(safety)** とは悪いことは決して起こらないという性質であり, $\Box F$の形の式で表現できる. これに対して活性 **(liveness)** とは良いことはやがて起こるという性質であり, $\Box\Diamond F$あるいは$\Diamond\Box F$の形の式で表現できる.
安全性については, すでに例1,2の$\Box Next$の例でみたように, $\Box F$は挙動のすべての状態(の組)に対してFが真となるわけであるから, 挙動のいつの時点でも必ずFが成立する, つまり, F以外は決して起こらないことを示すものである.
仕様として安全性だけを書いたのでは不足である. なにもしない消極的な系も仕様を満足するからである. したがって活性を明示的に書かなければならない. ところで活性の方は時間演算子が2個付いているからすこしややこしい.
式$\Box\Diamond F$がある挙動に対して真であるのは, その挙動のすべての状態(時点n)において$\Diamond F$が真であるときであり, そして, 時点nにおいて$\Diamond F$が真であるのは ある時点$m \geq n$にてFが真であるときである. すなわち,

$$[\Box\Diamond F]\sigma \equiv \forall n \in \mathbb{N} : \exists m \in \mathbb{N} : [F]\sigma^{n+m}$$

$\forall n : \exists m : p(n+m)$の形の式は$p(i)$が無限に多くの$i$の値に対して真となることを主張している. したがって, 挙動が$\Box\Diamond F$を満たすとはその挙動の中で無限回Fが真となることである. このことをFが無限回しばしば **(infinitely often)** 真になるなどという.
式$\Diamond\Box F$はやがて, つまり, 未来のある時点以降は常にFが真になることを主張している. 形式的には

$$[\Diamond\Box F]\sigma \equiv \exists m \in \mathbb{N} : \forall n \in \mathbb{N} : [F]\sigma^{n+m}$$

換言すれば, 挙動が$\Diamond\Box F$を満たすとはある時点が存在してその時点以降は永続してFが真になることである. このことをFがほとんど常に **(almost always)** 真になるなどという.
さて, 作用Aについての弱公平性 **(weak fairness)** をつぎのように定義する.

$$\begin{aligned} \mathrm{WF}_f(A) &\equiv \Box(\Box\mathrm{Enabled}\langle A\rangle_f \Rightarrow \Diamond\langle A\rangle_f) \\ &\equiv \Box\Diamond\neg\mathrm{Enabled}\langle A\rangle_f \vee \Box\Diamond\langle A\rangle_f \\ &\equiv \Diamond\Box\mathrm{Enabled}\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f \end{aligned}$$

これらは, それぞれ口語でつぎのようにいえるだろう.
「つぎが常に成り立つ : Aの実行可能が継続すればAはやがて実行される.」
「Aが無限回しばしば実行不可能であるか, あるいは, Aが無限回しばしば実行される.」
「Aが未来のある時点以降実行可能が継続すればAが無限回しばしば実行される.」
また, 作用Aについての強公平性 **(strong fairness)** をつぎのように定義する.

$$\begin{aligned} \mathrm{SF}_f(A) &\equiv \Diamond\Box\neg\mathrm{Enabled}\langle A\rangle_f \vee \Box\Diamond\langle A\rangle_f \\ &\equiv \Box\Diamond\mathrm{Enabled}\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f \end{aligned}$$

「Aがほとんど常に実行不能になるか, あるいは, Aが無限回しばしば実行される.」
「Aが無限回しばしば実行可能になればAが無限回しばしば実行される.」
以上の定義において, 作用が$\langle A\rangle_f$の形であることを注意したい. 上の口語表現は正確ではない. $\langle A\rangle_f$を単にAといっている.
時間演算子を伴った式の扱いのためにつぎの成立を注意しておく.
$\Box\Diamond\neg F \equiv \Box\neg\Box F \equiv \neg\Diamond\Box F$
$\neg\Box F \equiv \Diamond\neg F$
$\Diamond(F \vee G) \equiv \Diamond F \vee \Diamond G$
$\Box\Diamond(F \vee G) \equiv \Box\Diamond F \vee \Box\Diamond G$
例1, 2で仕様と称したものはこの活性を欠いていた. 例1では$\mathrm{WF}_x(Next)$を, 例2では$\mathrm{SF}_x(Next)$を連言させなければならなかった.

6 stuttering 不変性

TLA では作用と時間演算子を伴う TLA 式を $[A]_f$, あるいは, $\langle A \rangle_f$ の形に限っている. これは stuttering 不変性を求めたためである. 挙動の隣り合った 2 状態が同一のとき, その 2 状態を stuttering 段階という. TLA 式の真偽はそれを定める挙動のなかで同じ状態の有限個の反復を追加あるいは除去しても変わらない. この性質を stuttering 不変であるという. この不変性の追求はつぎのような事情によるものであった.

プログラムを扱う伝統的な時間論理は次状態演算子 ○ を使った. プログラム

$Prog1 \quad :: x := x + 1$

は

$\Box(x = 0 \Rightarrow \bigcirc(x = 1)) \cdots (*)$

を満たす. ところで,

$Prog2 \quad :: \textbf{local}\ t;$
$t := 1\,;\, x := x + t$

は $(*)$ を満たさない. $Prog2$ を $Prog1$ の洗練と考え $(*)$ を $Prog1$ の仕様とみなすとき, $Prog2$ が $(*)$ を満たさないのは具合が悪い.

また,

$Prog3 \quad :: y := y - 1$

は

$\Box(y = 1 \Rightarrow \bigcirc(y = 0)) \cdots (\dagger)$

を満たす. つぎの並行合成を考える.

$Prog4 \quad :: Prog1 \parallel Prog3$

この場合 $Prog4$ が $(*) \wedge (\dagger)$ を満たすことを期待したい. しかし, 実際は $(*)$ も $(\dagger)$ も満たさない. つまり, プログラム成分の仕様が並行合成の仕様の一部にならない. このような状況は実際の仕様作成に際しては好ましくない. 時間論理式が stuttering 不変であるとこの状況が好転する. 時間論理式の stuttering 不変性が望まれた所以である.

7 TLA による仕様

TLA による仕様記述の一例を挙げる. これは環境と系の作用の同時並行実行を許さない系またはその成分 (component) の差込方式による記述例である. 系は 1 個の入力 (環境) 変数 e, 1 個の出力変数 c, 1 個の内部変数 x をもつとする. この系の仕様はつぎの形式で記述できる:

$$\exists x.\ (Init \wedge \Box[Next]_{c,x} \wedge Live)$$

ただし :

$Init$ は c と x の初期値を表す状態述語.

$Next$ は系の段階を記述する作用. この系の場合入力変数を不変に保つから $Next$ は $e' = e$ を含意する.

c, x は $Next$ の添え字に現れる. というのは $\Box[Next]_{c,x}$ が c と x とを不変に保つ状態変化を許容するからである.

$Live$ はそれぞれが $\mathrm{WF}_{c,x}(A)$ か $\mathrm{SF}_{c,x}(A)$ の形の公平性条件の連言である. 一般に $Next$ は複数の作用の選言になる.

この方式による仕様作成はほぼつぎのような段階を追うことになるだろう.

(0) 系を記述する部分と抽象度を選べ.

(1) 系のもつ変数を選び, その型と初期値を書け. 定数引数やその仮定が定まる.

(2) 次状態関係を書け (作用の例を挙げてみよ).

(3) 仕様の時間に関する部分を書け. 安全性と活性 (公平性) を加味せよ. 仕様は初期述語, 次状態作用, 公平性の連言になる.

(4) 仕様から導出できる定理を述べよ.

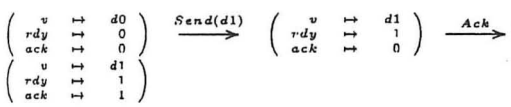
注意 系の機能については最初は無視し, あとで次状態作用に選言として追加せよ.

8 非同期交信界面

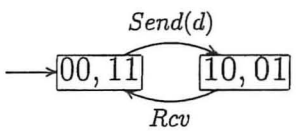
早速仕様記述に応用してみよう．つぎは非同期データ送受信界面（2 phase handshake protocol）の仕様例である．データを val 上で送り，rdy と ack 回線を同期のために使う．送信側はつぎのデータを送信する前に受信通知を受信しなければならない．
状況を図示すれば

$$Sender \xrightarrow{c} Receiver$$

交信状況はつぎのようなものである．



さらにこの界面はつぎのような推移図として説明できる．



つぎが TLA による記述例である．正確には TLA+ 風に書いたものである．TLA+は仕様記述向きに構文を補ったものである．モジュール化機構をもち，モジュールとして自然数，整数，実数、有限集合，列などをもっている．論理演算として，Hilbert の ϵ 作用子をもつ．また状態関数や述語記述のための構成子 IF THEN ELSE 式,CASE 式，LET 式などを用意し，さらに，読みやすさのための工夫をさまざまに行っている．以下の記述はその TLA+ に忠実にしたがったものではない．

module *Chan*

CONSTANTS D, $Two \triangleq \{0, 1\}$

VARIABLE c

$TInv \triangleq c \in [v : D,\ rdy : Two,\ ack : Two]$

$Init \triangleq TInv \wedge c.rdy = c.ack$

$Send(d) \triangleq c.rdy = c.ack$
$\quad \wedge c' = [c \text{ EXCEPT } !.v = d,\ !.rdy = 1 - @]$

$Rcv \triangleq c.rdy \neq c.ack$
$\quad \wedge c' = [c \text{ EXCEPT } !.ack = 1 - @]$

$Next \triangleq (\exists d \in D : Send(d)) \vee Rcv$

$Spec \triangleq Init \wedge \Box[Next]_c \wedge \mathrm{WF}_c(Rcv)$

THEOREM $Spec \Rightarrow \Box TInv$

記述中の $[r \text{ EXCEPT } !.c_1 = e_1, \cdots, !.c_n = e_n]$ は r においてそれぞれの i に対して $r.c_i$ を e_i で置き換えてえられたレコードを表示する．また，式 e_i に出現する@は $r.c_i$ を意味する．

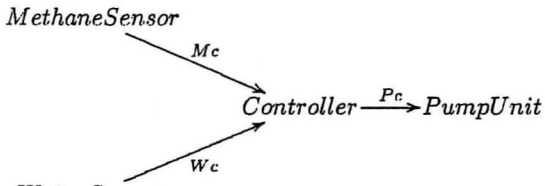
9 炭坑排水制御系

例題として炭坑の浸水に対して排水ポンプを作動させる制御プログラムの作成を考える [1][3][4]．浸水してもメタン濃度が高ければポンプを運転することはできない．
系はつぎの構成成分をもっている．
ポンプは制御プログラムからポンプの「起動」と「停止」の命令を受ける．
メタン感知器は制御プログラムへ坑内メタン濃度が「安全」か「危険」を知らせる．
水位感知器は制御プログラムへ坑内水位が「高い」か「低い」を知らせる．そこでプログラムはつぎを要請する．

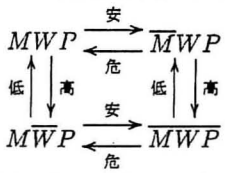
(1) メタン濃度安全かつ水位高ならばポンプを作動
(2) メタン濃度安全かつ水位低ならばポンプを停止
(3) メタン濃度危険かつ水位低ならばポンプを停止

(4) メタン濃度危険かつ水位高ならばポンプを停止

状況は次図のとおりである.



この系を推移図として表せば



図の中の文字はつぎのような意味をもっている：

$M \equiv Mcrit$ $MWP \equiv M \wedge W \wedge P$

$W \equiv Wlow$ $M\overline{W}P \equiv M \wedge \neg W \wedge P$

$P \equiv Poff$

つぎが TLA+風の記述例である.

---

module *MinePumpController*

CONSTANTS $M = \{$安,危$\}$

$W = \{$低,高$\}$

$P = \{$入,断$\}$

VARIABLE m, w, p, s

$Mc \triangleq$ **instance** $Chan$ **with** $D \leftarrow M, c \leftarrow m$

$Wc \triangleq$ **instance** $Chan$ **with** $D \leftarrow W, c \leftarrow w$

$Mc \triangleq$ **instance** $Chan$ **with** $D \leftarrow P, c \leftarrow p$

$TInv1 \triangleq s \in [ms : M,\ ws : W,\ ps : P]$

$Mcrit \triangleq s.ms =$危

$Wlow \triangleq s.ws =$低

$Poff \triangleq s.ps =$断

$Tinv \triangleq Mc!TInv \wedge Wc!TInd \wedge Pc!Inv \wedge TInv1$

$Init \triangleq Tinv \wedge Mc!Init \wedge Wc!Init \wedge Pc!Init \wedge$
$Mcrit \wedge Wlow \wedge Poff$

$MSend(d) \triangleq Mc!Send(d) \wedge Mc!Rcv \wedge$
UNCHANGED $\langle wc \rangle \wedge$
case $Mcrit \wedge Poff \wedge d =$安 $\rightarrow$
$Pc!Send($入$) \wedge$
$s' = [s$ EXCEPT $!.ms = d, !.ps =$入$]$
$\neg Poff \wedge d =$危 $\rightarrow Pc!Send($断$) \wedge$
$s' = [s$ EXCEPT $!.ms = d, !.ps =$断$]$
other $\rightarrow s' = [s$ EXCEPT $!.ms = d,] \wedge$
UNCHANGED $\langle pc \rangle$

$WSend(d) \triangleq Wc!Send(d) \wedge Wc!Rcv \wedge$
UNCHANGED $\langle mc \rangle \wedge$
case $\neg Poff \wedge d =$低 $\rightarrow Pc!Send($断$) \wedge$
$s' = [s$ EXCEPT $!.ws = d, !.ps =$断$]$
$\neg Poff \wedge d =$低 $\rightarrow$
$Poff \wedge \neg Mcrit \wedge d =$高 $\rightarrow Pc!Send($入$) \wedge$
$s' = [s$ EXCEPT $!.ws = d, !.ps =$入$]$
other $\rightarrow s' = [s$ EXCEPT $!.ws = d] \wedge$
UNCHANGED $\langle pc \rangle$

$PRcv \triangleq Pc!Rcv$

$Next \triangleq (\exists d \in M : MSend(d)) \vee$
$(\exists d \in W : WSend(d)) \vee PRcv$

$Spec \triangleq Init \wedge \Box[Next]_{\langle mc, wc, pc, s \rangle} \wedge$
$\mathrm{WF}_s(PRcv)$

まとめ

TLA を応答系の仕様記述に利用することを紹介したが，TLA 自身の説明も不充分である．形式的な証明については完全に省いた．実際に利用するために必要な仕様の洗練，実現，また，分解，合成についても一切説明していない．実時間系 (real-time system) や混合系 (hybrid system) のための機構についても同様である．しかし，TLA は組込系 (embedded system) などで是非採用・実用化してみたいと思う．

設計パターンやコンポーネントなどについてはこの枠組で厳密な議論ができると考えている．

参考文献

[1] I. J. Hayes and B. Mahony, A case study in timed refinement: A mine pump. IEEE TSE, 1992.

[2] L. Lamport, The temporal logic of actions. ACM TOPLAS,1994.

[3] K. Lano, Formal Object-Oriented Development. Springer, 1995.

[4] ibid., The B Language and Method. Springer, 1996.

[5] 田辺誠他，コンピュータサイエンス入門　論理とプログラム意味論．岩波，1999.

SIGFM セミナー
形式手法の魅力とは何か？

フォーマル・メソッドの実践方法

2001年5月24日

荒木啓二郎
（九州大学大学院システム情報科学研究院/
（財）九州システム情報技術研究所）

1

構成

- はじめに
  - 形式手法に対する認識
  - 動向
- 事例
  - 配電線監視制御システム
  - 自動倉庫システム
- 考察
  - 経過と効果
- おわりに
  - 実践におけるポイント

2

はじめに

- 形式手法 (Formal Methods)に対する認識
  - J. A. Hall:
    Seven Myths of Formal Methods,
    IEEE Software, Vol.7, No.5, pp.11-19, 1990.
  - J.P. Bowen and M.G. Hinchey:
    Seven More Myths of Formal Methods,
    IEEE Software, Vol.12, No.4, pp.34-41,
    1995.
- 日本では話題になることすらあまりなかった
  - Pre-Myths : 神話以前？(Araki, 1995)

3

形式手法に関する七つの誤認識 (Hall, 1990)

- 形式手法はソフトウェアが完全であることを保証できる
- 形式手法とは須らくプログラムの証明である
- 形式手法はセーフティクリティカルシステムにのみ有効である
- 形式手法は高度に訓練された数学者を必要とする
- 形式手法は開発コストを増加させる
- 形式手法はユーザには受け入れられない
- 形式手法は現実の大規模ソフトウェアには使われない

4

形式手法に関する七つの事実 (Hall, 1990)

- 開発の初期の段階での誤りの発見に有効
- 開発対象のシステム自体を深く考えさせることに寄与
- いかなる応用分野にも有効
- 数学を基礎としてはいるものの、プログラムよりは理解し易い
- 開発コストを減少させる
- 顧客の理解を助ける
- 産業界での実用プロジェクトに適用されて成功

5

形式手法に関するもう七つの誤認識 (Bowen & Hinchey, 1995)

- 形式手法は開発過程を遅らせる
- 形式手法にはツールがない
- 形式手法は従来の設計手法に置換わる
- 形式手法はソフトウェアにのみ適用できる
- 形式手法は必要ない
- 形式手法は支持されない
- 形式手法の人達は常に形式手法を用いる

6

形式手法の適用事例

- CICS: IBM Hursley Lab. & Oxford Univ.(Z)
- 原子力発電:Rolls-Royce (VDM)、カナダダーリントン(SCR/Darlington Method)
- 鉄道:パリ地下鉄信号システム(B method)、中国鉄道局 & UNU/IIST (RAISE)
- 航空管制:NASA (theorem provers)
- 航空機:A330/340 (Z)
- 医療:放射線システム(Z)、サイクロトロン、HP

7

形式手法の適用事例 (続き)

- ハードウェア設計:Tektronix (Z)
- マイクロプロセッサ開発:Inmos (Z, CSP, ML)、FM8501, FM9001(Boyer-Moore)
- ウィンドウ処理プログラム:日本ユニシス (Z)
- 鉄道連動装置の仕様記述:JR総研 (VDM)
- 事務システム: 日本フィッツ (VDM)
- その他多数

8

CICS: Customer Information Control System

- IBM Hursley Lab. & Oxford Univ.
- 800,000行の中の300,000行を改訂
  - 37000行: Z 仕様記述、設計
  - 11000行: Z (部分的)
- 2000頁の形式的な文書
- 開発コスト:9%削減

9

規格 / ガイドライン 等

- 一般
  - PES(英国、1987)
  - IEC65A 122(欧州、1991)
  - P1228(米国、1993)
- 航空/宇宙
  - DO-178A(米国、1985), DO-178B(米国、1992)
  - PSS-05-0(欧州、1991)
- 原子力
  - IEC880(欧州、1986)
  - 2.234.1(カナダ、1992)
- その他

10

技術移転 / 教育

- 技術移転
  - Z では30年の歴史
  - 表明 (Assertions) の技法
  - 標準化(特に、ツール)
- 教育、訓練
  - 教科書、コースの充実
  - Praxis: 全社的な共通の素養
- 教育コースや技術者の認定制度
  - アクレディテーション : British Comp. Soc.

11

事例研究

- 配電線ネットワーク監視制御システム
- 自動倉庫システム
- 多様なシステム記述・分析
  - 静的性質
  - 動的振舞い
  - 形式仕様
  - 非形式/半形式的記述

12

事例1：配電線監視制御システム

- 福岡の中堅企業
- 電力会社に納入
- 最初は自社開発
- 今は、設計までで実現は外部に発注

問題点

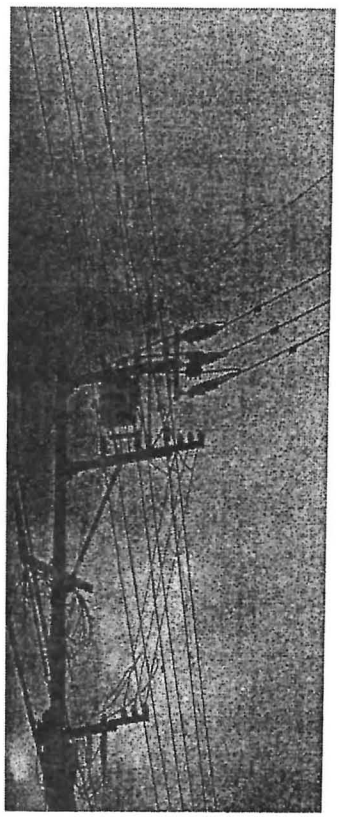
技術移転
システムの全容が理解できていない
発注する際の仕様と検収

13

形式手法適用の目的

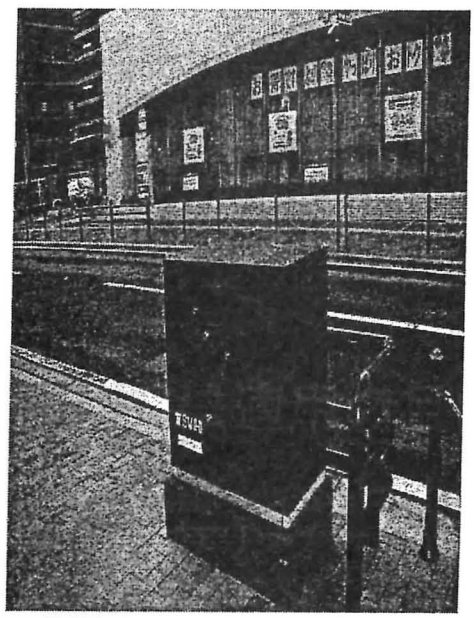
- リエンジニアリング
  - 汎用機からPCへ
- 再利用性
  - 異なる電力会社のシステムへの対応
- 技術移転/新人研修
- 社内文書の体系化
- システム開発の効率化
  - 高信頼性

14



開閉器（電線、電柱）

15



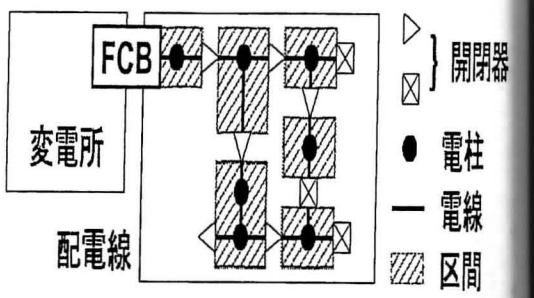
開閉器（地下ケーブル）

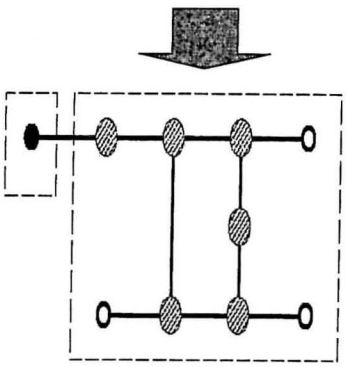
16

言語とツール(配電線)

- **RAISE**
  - モデル化
  - 仕様記述
- **Z**
  - 仕様の再記述
- **Java**
  - プロトタイピング
  - 視覚化
- **VDM-SL**
  - 仕様の再記述
  - 仕様レベルでの解釈・実行(アニメーション)
- グラフ分析ツール
  - ドメインモデル化、分析

17

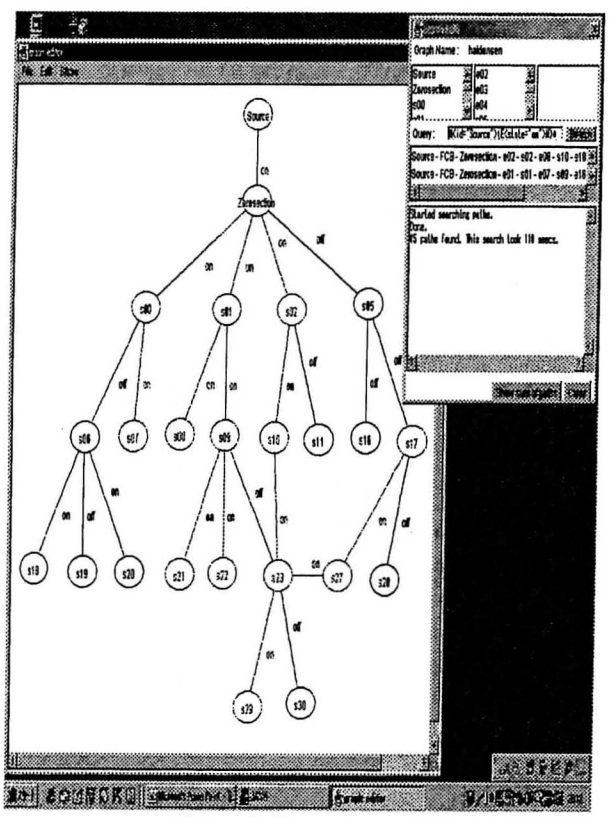




双対性
開閉器:枝
区間:節点

抽象モデル

18



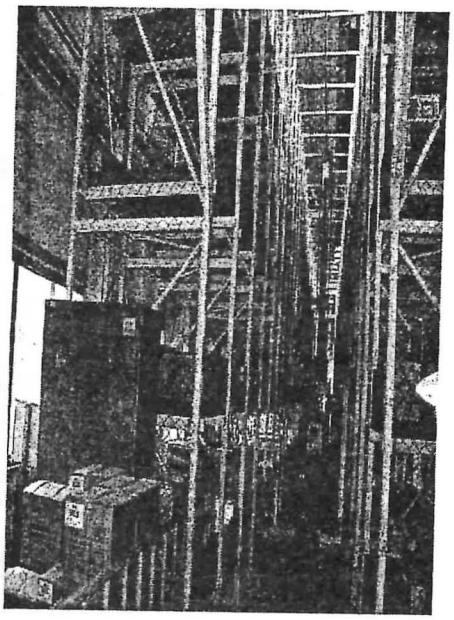
グラフ分析ツール

19

事例2:自動倉庫システム

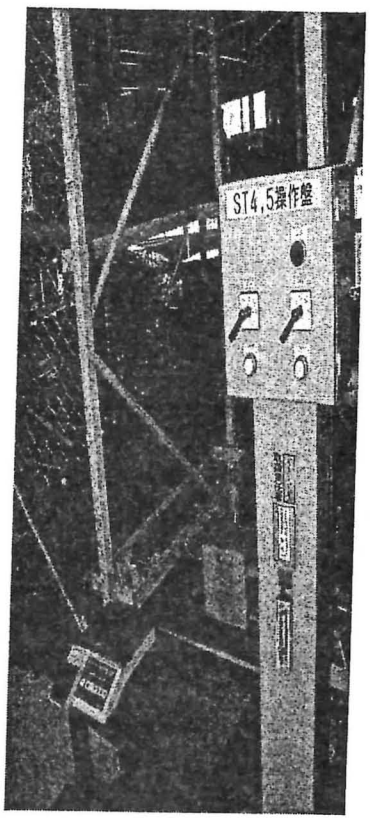
- 福岡の中堅企業
- 装置の制御とデータ管理
  - 設備(ラック、クレーン、搬送路、等)は別会社
- 多様性
  - 施設、設備、システム構成、用途、運用
- 再利用性
- 再構成の容易さ

20



自動倉庫（棚とクレーン）

21



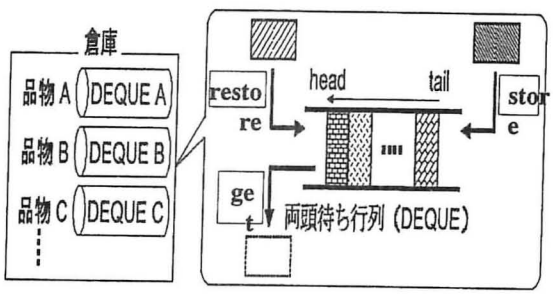
自動倉庫（入出庫作業場）

22

言語とツール（自動倉庫）

- Z
  - モデル化
  - 仕様記述
- VDM-SL
  - 仕様の再記述
- ML
  - プロトタイピング

23



自動倉庫システムに対する抽象モデル

24

形式手法適用の経緯

- 売込みと対象の選定
- Z のチュートリアル
- ドメインエキスパートから聞き取り
- 仕様記述の種々の版の作成と検討
  - 機能
  - モデル化
- 共通語としての Z
  - コミュニケーションの道具
- アニメーション、プロトタイピング
- 文書の体系化
  - 用語集＋形式的定義へのリンク

25

時間と労力

- 配電線システム
  - 約半年：月に2回程度会合
  - ドメインエキスパート：1〜2名
  - 仕様記述とレビュー：4〜6名
- 自動倉庫システム
  - 約半年：月に2回程度会合
  - ドメインエキスパート：1〜2名
  - 仕様記述とレビュー：4〜6名
- ドメインの理解とモデル/仕様記述の熟成

26

形式手法適用の効果

- コミュニケーションの道具
- 論点が明確
- ドメインの再認識

  暗黙の前提、常識：共通ではない
  曖昧性の顕在化

- 知識の共有、継承
  - 卒論生の理解と習得の早さ

27

形式仕様

- 読むのは、難しくない
  - 簡単な解説
  - 形式仕様のレビュー (OJT)
- 書くには、経験が必要
  - 言語仕様 (syntax)
  - 対象のモデル化 (semantics)
  - イディオム (pragmatics)
- アニメーション / 視覚化：非常に有効
  - multi-lingual
  - multi-aspect

28

おわりに

- 有効性は確認できた
  - システムの一部
  - モデル化と仕様記述
  - アニメーション、プロトタイピング
- 設計と実現
  - **stepwise refinement ?**
- 保守
- 文書化
  - 参照モデル

多様性、多面性、柔軟性

- 仕様記述の事例研究
  - セキュリティプロトコルの形式仕様記述と検証:SSL
  - 配電線監視制御システム:モデル化、形式仕様
  - 自動倉庫システム:モデル化、形式仕様
- 図式表現と形式仕様との融合
  - ERD、DFD、IDEF1X、UML : Z、VDM-SL
  - グラフ分析ツール :ドメインモデル化、分析
- 関係データベースシステムから
  オブジェクト関係データベースシステムへ
  - IDEF1X ⇒ UML
- 多様で柔軟な記述:方法、道具建て、内容
  - Z, RAISE, VDM-SL, ML, Java, ...
- 詳細化 **(Refinement)** とソフトウェアライフサイクル

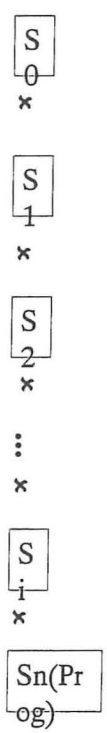
段階的詳細化(Stepwise Refinement)

- 詳細化の関係: S1 × S2
  - S1 を S2 が満足する
- 段階的詳細化:
  S1 × S2 × S3 × ... × Sn × Impl

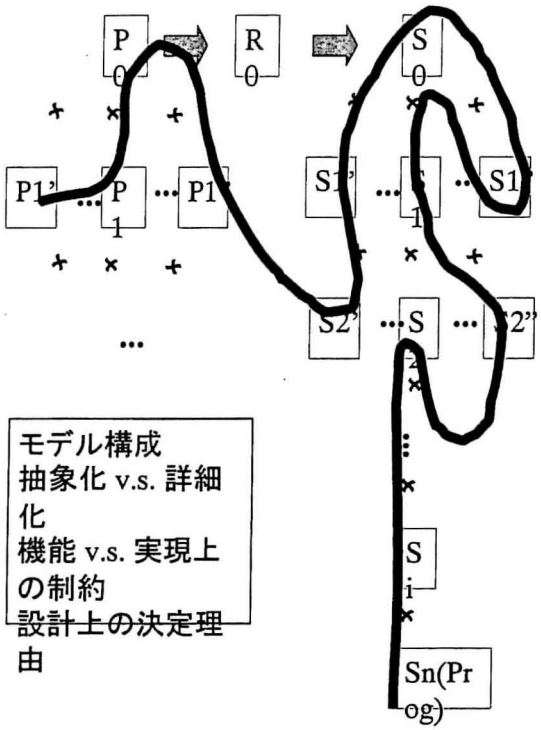
[Carroll Morgan : Programming from Specifications, Second Edition, Prentice Hall, 1994.]

段階的詳細化

詳細化の道筋:
初めから一意に定まるのか?



試行錯誤



実践のポイント

- 管理職の意識
  - 形式手法への関心、期待
  - 担当ドメインエキスパートの意欲への影響
- 厳密な記述と議論
  - 最初の戸惑い
  - 新鮮な感動
  - 慣れ
  - 維持継続の努力と支援
- コミュニティ
  - 普段の相談相手
  - guru

形式手法の十戒

[Bowen & Hinchey: Ten Commandments of Formal Methods, IEEE Computer, Vol.12, No.4, pp.34-41, 1995]

- 汝、適切な表記法を選ぶべし
- 汝、形式化を行うべし、されど過ぐること勿れ
- 汝、コスト予測をすべし
- 汝、形式手法の尊師を身近に持つべし
- 汝、従来よりの開発法を棄つること勿れ
- 汝、十分に文書化すべし
- 汝、品質に妥協すること勿れ
- 汝、独善となる勿れ
- 汝、テストすべし、またテストすべし、さらにテストすべし
- 汝、再利用すべし

まとめ

- 形式手法は有効：体験しなけりゃわからない
- 概念の整理やモデル構築 v.s. 実現
- 使えるところ、使えることから始めよう
- コミュニケーションの道具：相互認識、共通の理解
- 上司の理解と意欲：部下は敏感
- アウトソーシング v.s. 技術の空洞化
- 技術・経験の共有と継承
- 教育・訓練：教科書、課程
- コミュニティ形成

参考文献

- K. Araki: Are Formal Methods Relevant? - How to Explore the Seven Myths in Japan -, Proc. APSEC '95, pp.514-515, 1995.
- A. Kawamura, S. Oda, H.-M. Chang and K. Araki: Software Development with Common Formal Descriptions, Proc. ISFST-96, pp.146-153, 1996.
- J.P. Bowen and M.G. Hinchey: High-Integrity System Specification and Design, Springer-Verlag, 1999.
- 日本ユニシス株式会社:特集:オブジェクト指向と形式仕様, 技報 UNISYS TECHNOLOGY REVIEW, Vol.18, No.4, 1999年2月.
- K. Araki, H.-M. Chang and T. Tanaka: Case Studies of Formal Approaches to Domain Modelling and Specification, Proc. ISFST-2000, pp.213-218, 2000.
- 佐原伸:事務システムにおける形式仕様記述適用事例、ソフトウェアシンポジウム2001論文集, 2001年6月.
- http://www.afm.sbu.ac.uk/
- http://www.shinsahara.com/www/sigfm/

発表の概要

- 鉄道におけるコンピュータ利用の歴史
- コンピュータ制御鉄道信号システム
  - 高安全性システム、フェールセーフ
  - ハードウェア技術
  - ソフトウェア技術
- 安全性技術指針
  - 安全性ライフサイクルとインテグリティレベル
- 鉄道信号システムの紹介
- VDMの適用
  1. 運動シミュレータについて
  (1)運動シミュレータの概要
  (2)形式的仕様
  2. 開発時のはなしディジタルATCの概要

鉄道におけるコンピュータ利用の歴史

- 1957年:ベンディックス(Bendix)社 G−15形コンピュータが日本最初の輸入コンピュータとして2台輸入された
  - その内の1台が鉄道技術研究所(鉄道総研の前身)へ
- 1959年:東京−大阪間 電車特急「こだま」運転開始
- 1960年:座席予約システム MARS-1が完成(磁気ドラム使用)。
  - 特急列車4本。2000座席

鉄道におけるコンピュータ利用の歴史

- 1964年:MARS101
  - 3万座席(当初は、新幹線予約できず。 65年東京−大阪3時間10分運転時に予約開始)
- 1968年:MARS103
  - 20万座席
- 1972年:MARS105
  - 100万座席
- 1993年:MARS305
  - 120万座席/日 260万コール/日 +コンサートやイベント

鉄道におけるコンピュータ利用の歴史

- 1967年より貨物関係のシステム開発を行った。
  - 快速貨物情報システム(FOCS)
  - コンテナ情報システム(EPOCS)
  - ヤード自動化システム(YACS)
  - フレートターミナル情報システム(FIS)
- 貨物輸送の衰退

鉄道におけるコンピュータ利用の歴史

- 1972年:新幹線岡山開業
  - COMTRAC (新幹線用プログラム進路設定システム)
  - 列車ダイヤに従って、列車の進路を制御する進路制御系コンピュータ
    ＋
  - 列車ダイヤが乱れた場合に警報を出したり、列車ダイヤの変更の提案を行う運転整理系コンピュータの2系統のコンピュータからなる

鉄道におけるコンピュータ利用の歴史

- 1980年:DACS
  - データ交換システム
- 1985年:電子連動装置
  - 京浜東北線 東神奈川駅

高安全性システム

- フォールトアボイダンス
  - 障害やエラーの原因となるフォールトが含まれないようにする
- フォールトトレランス
  - フォールトの発生は不可避であるとの前提に立ってシステムとして対処する。
  - フォールトが発生しても機能を維持する。
  - フォールトが発生した場合部分的に機能する(フェールソフト)
  - フォールトが発生しても危険な出力は出さない。(フェールセーフ)
  - 出来るだけ早く正常状態に復帰する。

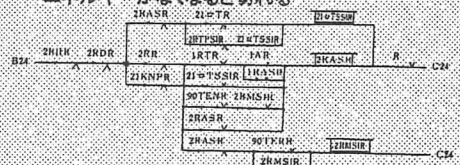
鉄道信号

- 列車を安全にかつ効率的に運転するための保安装置
  - 閉そく装置、連動装置、踏切装置
  - 自動列車制御装置(ATC)など
- フェールセーフ性
  - 故障したときにも安全な状態を維持
  - 鉄道信号システムでは、安全な状態を定義可能

⇒ 列車を停止

フェールセーフ

- リレー回路のフェールセーフ
  - 安全性の確保を原則としてメーク接点
    動作したら(リレーのコイルに電流が流れる:エネルギーが加わる)スイッチが入る接点
  - エネルギーがなくなると切れる



マイクロエレクトロニクスの安全性手法

- 安全性確保のための原理
  1. 冗長構成
  2. 故障診断
  3. 異常検出時の出力安全側固定

フェールセーフ技術 と 各種高信頼化手法
＋ 厳格な安全性管理のアプローチ

従来 非対称特性(リレーなど)

ハードウェア安全性技術

- 冗長構成
  - パスレベルの比較(2 out of 3, 2 out of 2)
  - 演算結果の比較
- 入出力制御
  - 入力データのタイミング
  - 入力回路の故障診断
  - 故障時に安全側出力
    (フリップフロップによる交番出力が代表的)

ソフトウェアと協調

- 故障診断
- 出力の安全側固定

ハードウェア安全性技術

(1)システム構成法
- 多重系システム
  - ・バス同期2重系
    バスデータ照合方式、不一致時の処理方法
  - ・2重系　：独立な処理の結果の照合、系切り替え
  - ・3重系　：多数決論理
- マルチプロセッサ
- 動作モニタ

(2)入出力方式
- 多重系
  - ・低位優先回路の採用
  - ・2個のCPUの論理積で制御出力あるは信号入力
- 故障検出
- 入出力データチェック

ソフトウェア

故障診断
- ソフトウェアによる故障検出
  - ハードウェア故障の検出 (CPU、メモリ、入出力回路)
  - ソフトウェアの実行時異常の検出 (実行順序、結果照合)
- ハードウェアによる故障検出
  - ソフトウェアの実行時異常の検出 (無限ループ、禁止領域)

縮退
- 割り込みやOSに制限
- 安全な状態を明確に定め、割り付け

妥当性
- 実績のある制御アルゴリズム
- 各種試験(意地悪試験を含む)

安全側と危険側の情報の区分け(電子連動)

情報種別		論理値0	論理値1
入力	進路設定	復位	設定
	軌道回路	列車あり	列車なし
	転てつ機状態	転換中	転換完了
内部状態	転てつ鎖錠	鎖錠	解錠
	接近・進路鎖錠	鎖錠	解錠
出力	信号機	停止	進行
	転てつ機	現状維持	転換
故障検出		故障	正常

安全性と信頼性

- 信頼性の向上は、安全性確保のために重要
  しかし、高信頼性で安全性を置き換えることは、不適当
  - 数値のみの評価　　モデルの限界
  - 想定しうる故障モード、外乱に対する配慮
  - 結果としてえられた評価値が目標を下回らない

鉄道信号の安全性技術指針/安全性規格

列車保安制御の安全性技術検討委員会('94～'95年)
大学，JR・民鉄，メーカ，事務局：鉄研

コンピュータによる列車保安制御の新たな安全性技術指針

規格

フェールセーフ

安全性ライフサイクル
安全性インテグリティレベル

我が国で培ってきた列車保安制御の安全性技術
電子連動，ATC

国際安全性規格
IEC 61508
一般産業分野

IEC61508=ISO9000+安全性方法論

IEC 61508

- Functional safety: safety-related systems

◆背景と概要
- ➢一般産業分野を対象とした安全性技術規格
  コンピュータ制御機器を対象
  →包括的な標準規格になる可能性あり
- ➢IEC(世界電気標準会議)規格
- ➢JIS化　JIS C0508-1～7（翻訳JIS）
- ➢ISO9000との関係
  (これは内容の話ではなく出続き論)

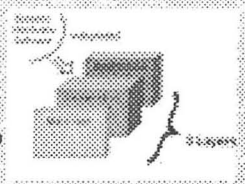
➢内容
- ➢安全性ライフサイクル、安全性インテグリティレベル

CENELECとの違い

CENELEC
- 法的な規制
- ハードウェア/ソフトウェア に分離

安全性技術指針
- ガイドライン
- 総合
- 本文 - 解説 - 資料

ヨーロッパの鉄道信号規格
EN 50126 RAMS
EN 50128 ソフトウェア
ENV 50129 認可・受入条件 （ドキュメントによる立証）



安全性ライフサイクルとインテグリティ

- 安全性ライフサイクル
  要求される保安機能を達成・維持するために、保安制御システムの概念設計から廃棄までに必要とされる活動。厳密に区分管理することで不安全な要素を排除
  ⇒設計だけ良くても故障である
- 安全性インテグリティ
  安全性に対する要求の大きさを表す指標（概念）

安全性インテグリティレベル

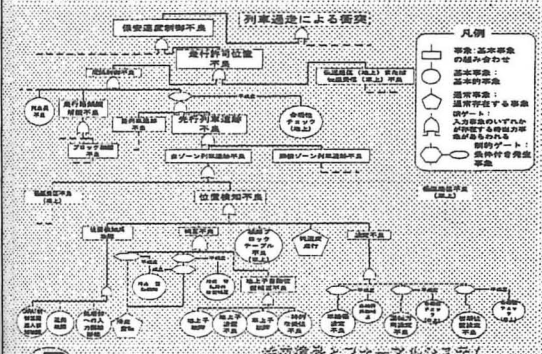
- リスク
  – 事故の発生確率と被害深刻さを示す。
- 安全性インテグリティレベル(被害は0, 1)
  $= R - R_{target}$
  安全対策を取らない場合のリスク R
  目標とするリスク R_{target}
- 定量的なリスク解析は可能？
  データベースが必要。新しい装置の場合は？
- インテグリティレベルと経済性
  こちらの方から概念が出ている。

ハザード解析とリスク解析

- ハザード解析 (FTA と FMECA)
  1. 対象システムについてより深い洞察
  2. システム的なアプローチによって解析の確度が向上
- リスク解析
  – 発生頻度－影響度
  1. 故障のみ監視することの危険性
  2. 絶対的評価 - 絶対的評価
  3. データベースの充実

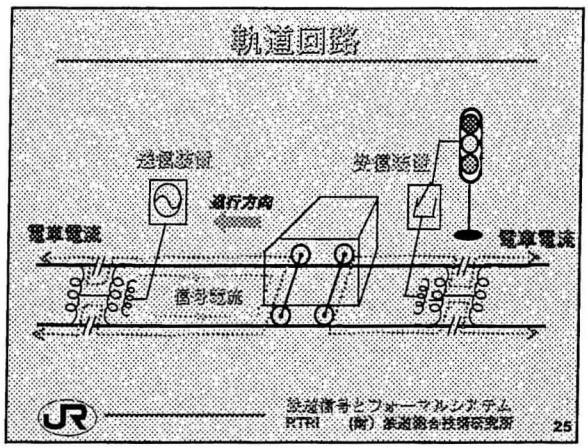
確定論での検討
- ハザード解析でのハザードの特定が重要
- 特定ハザードに対してフェールセーフ対策

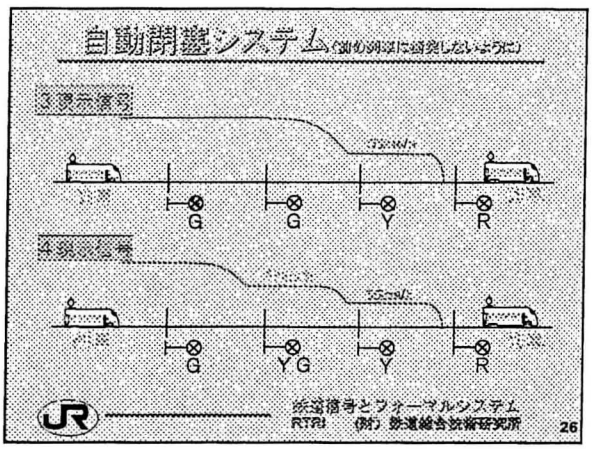
CARATのFTA(列車過走による衝突)

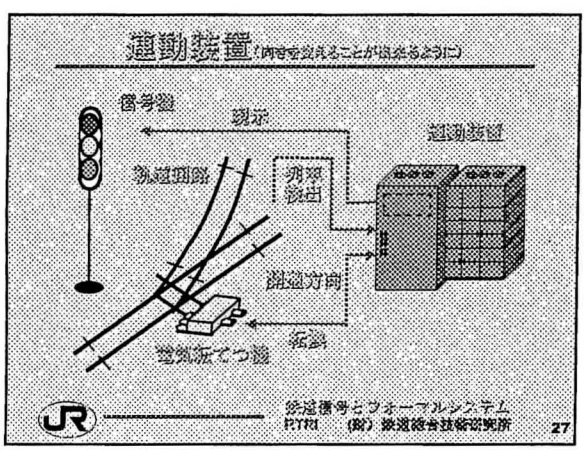


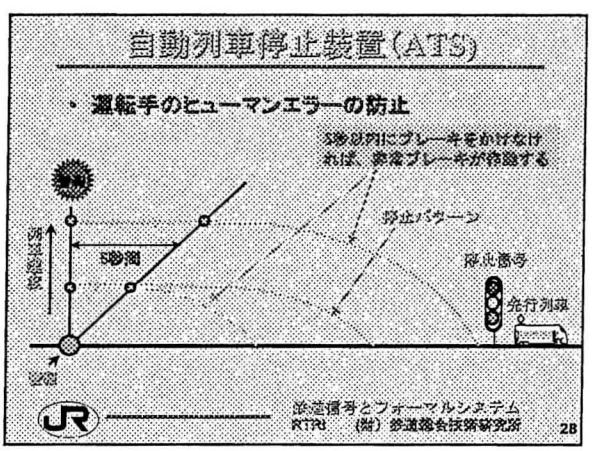
鉄道信号システムの紹介

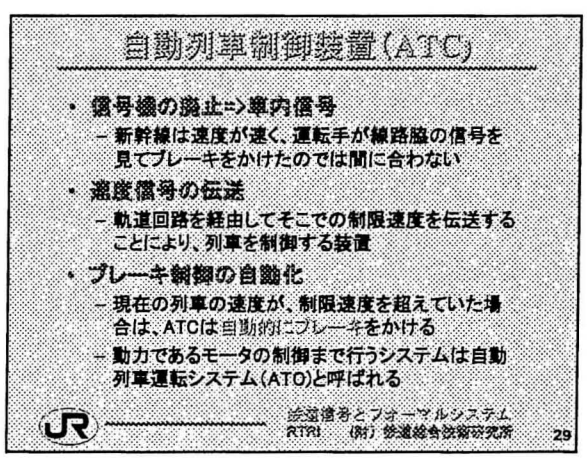
- ブレーキ距離
  – 列車は自動車などに比べブレーキ距離が長い
- 閉塞制御
  – 安全の確保のため、ある区間内に1列車のみ入れるようにする閉塞制御が通常使用される
  – この区間を閉塞区間と呼ぶ
- 軌道回路
  – 閉塞区間に列車がいるかいないかは、軌道回路によって判定される
- 自動閉塞システム

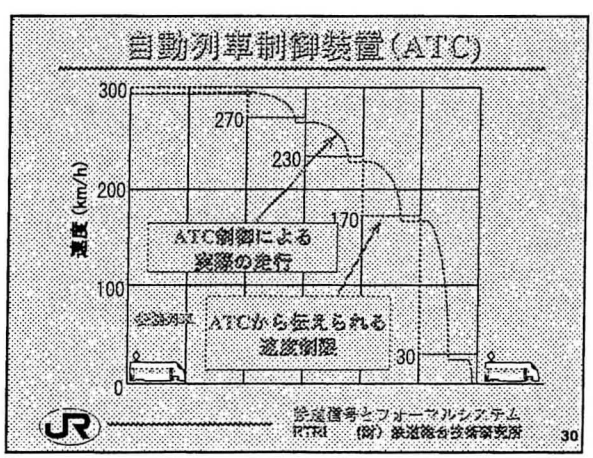

軌道回路
送信装置
受信装置
進行方向
電車電流
電車電流
信号電流
鉄道信号とフォーマルシステム
RTRI (財) 鉄道総合技術研究所
25


自動閉塞システム
3現示信号
G
G
Y
R
4現示信号
G
YG
Y
R
鉄道信号とフォーマルシステム
RTRI (財) 鉄道総合技術研究所
26


運動装置
信号機
現示
運動装置
軌道回路
列車検出
開通方向
転換
電気転てつ機
鉄道信号とフォーマルシステム
RTRI (財) 鉄道総合技術研究所
27


自動列車停止装置(ATS)
・運転手のヒューマンエラーの防止
5秒以内にブレーキをかけなければ、非常ブレーキが作動する
停止パターン
5秒間
停止信号
発行列車
鉄道信号とフォーマルシステム
RTRI (財) 鉄道総合技術研究所
28


自動列車制御装置(ATC)
・信号機の廃止=>車内信号
– 新幹線は速度が速く、運転手が線路脇の信号を見てブレーキをかけたのでは間に合わない
・速度信号の伝送
– 軌道回路を経由してそこでの制限速度を伝送することにより、列車を制御する装置
・ブレーキ制御の自動化
– 現在の列車の速度が、制限速度を超えていた場合は、ATCは自動的にブレーキをかける
– 動力であるモータの制御まで行うシステムは自動列車運転システム(ATO)と呼ばれる
鉄道信号とフォーマルシステム
RTRI (財) 鉄道総合技術研究所
29


自動列車制御装置(ATC)
300
270
230
200
170
100
30
0
速度 (km/h)
ATC制御による実際の走行
ATCから伝えられる速度制限
鉄道信号とフォーマルシステム
RTRI (財) 鉄道総合技術研究所
30

VDMの適用

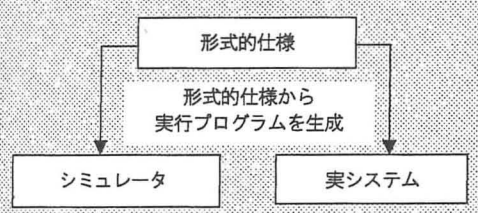
連動シミュレータについて

(1)連動シミュレータの概要

シミュレータ開発の目的

- 連動装置の仕様作成
  - 鉄道信号への形式的手法適用の検討
  - 仕様記述のトレーニング
- シミュレータ作成
  - 安全性検討のためのツール
  - デモンストレーション
  - ユーザの仕様検証手法の確立

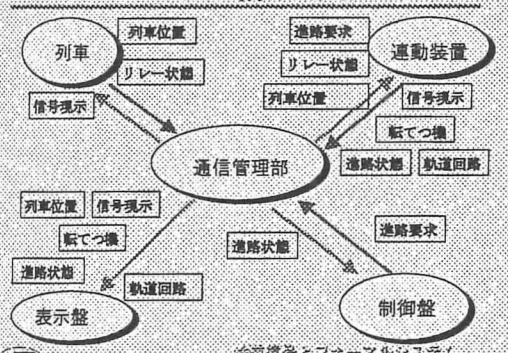
ユーザの仕様確認手法の確立



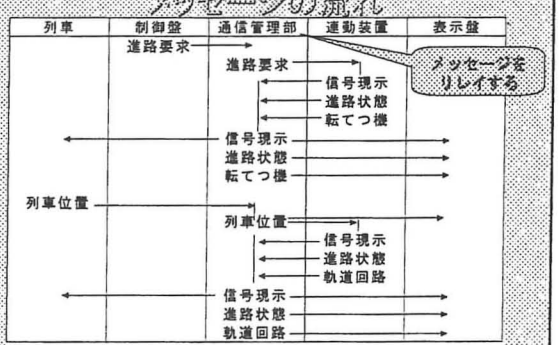
システム構成

- 連動装置、列車、表示盤、制御盤の4つのエージェントと通信管理部から構成

システム構成(図)



メッセージの流れ



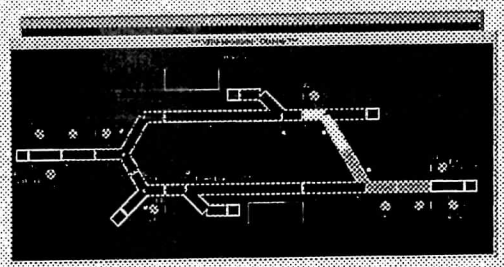
連動装置部

- VDM-SLで連動論理を記述
- IFAD VDM-SL Toolbox の機能により、C++コードを自動生成
- 他エージェントとの通信インターフェースをC++で別途作成

その他の特徴

- 連動論理を変更しても、他のエージェントを改修する必要がない。
  - 複数の連動論理を比較検討可能
- 通信管理部でログを保存し、処理を確認できる。

シミュレータ画面



VDMの適用

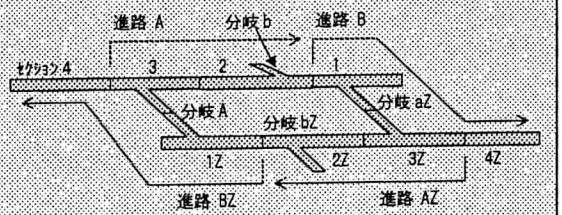
連動シミュレータについて

(2)形式的仕様

実現した連動の機能

- 各進路には最大1列車まで進入可能
- 進路の各要素が他の進路、または、列車に占有されている場合は、進路構成できない。
- 進路内に列車が存在しない場合は、進路を解除できる。
- 進路の最終セクションに列車が到達すると、進路は自動的に解除される。

駅のモデル



主な状態変数(1)

- 構成要素とその関連付け
  - routes: 進路の集合
  - sections: セクション(軌道回路)の集合
  - occupants: 占有者の集合(3-2, 3Z 等)
  - branches: 分岐の集合
  - route-branch: 進路と分岐の対応
  - route-occupant: 進路と占有者の対応
  - route-open-section: 進路に対応するｾｸｼｮﾝ

主な状態変数(2)

- 状態をあらわすもの
  - free-branch: フリーな分岐の集合
  - unoccupied-section: フリーなｾｸｼｮﾝの集合
  - route-state: 各進路とその状態の対応
  - current-occupant: 進路上の列車位置(占有者)

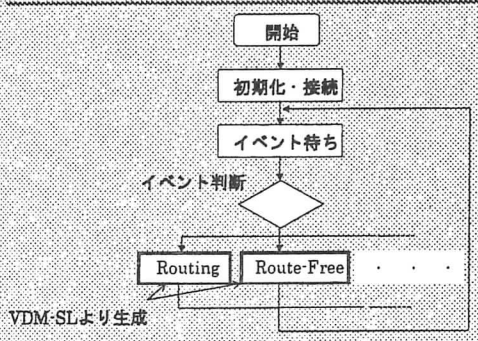
主な操作

- Nstate: ネットワーク状態生成
- Routing: 進路設定
- Route-free: 進路復位
- Route-Busy: 進路への進入
- Route-Run: 列車の進行

不変式

- データの制約条件
- 安全性に関わる条件
  - 安全の概念(衝突・脱線しないなど)
  - 連動の条件(鎖錠など)

外付けコードからの呼出



VDMの適用

開発時のはなし

入社2－3年が中心に開発した。

最初にやっていたこと

- 教科書の輪読
  - 「Systematic Software Development using VDM」
  - 「Proof in VDM」
- JRの運動の形式的なモデル化検討

その時思っていたこと

- 「Systematic Software Development using VDM」「Proof in VDM」の中で書かれていることを、実際どのように使うかわからない
- 運動が複雑であり、
  - モデル化のイメージがどんどん複雑になる
  - VDM−SLで記述することのスキルが低い

状況打開のための方針

- 単純な機能に限定した簡易モデル
- (証明にはこだわらずに)
  とりあえず、動くものを作ってみる
  - 当初は、Systematic Software Development using VDMやProof in VDMからスタートしたため、証明を非常に意識していた。

進め方

- Zで書かれた運動図表検証を目的とした簡易モデルからスタート
- このZモデルからimplicit な仕様の作成
- implicit な仕様からexplicit な仕様の作成
- explicit な仕様をもとに、
  実行プログラム作成→シミュレータ

仕様の書換作業

- Z → VDM-SL
  - 機能毎に担当者を決めて分業
  - Zで書かれた論理
  - VDM-SLの例題
- 学習期間の短い担当者でも、書換作業可能であった
  (各担当は学習期間0.5年未満〜1.5年)

仕様を書いてみて

- Explicit に書きなおして、いろいろとチェックできた
- 実際に書いたものを、VDM Tools のインタプリタで扱ってみて、仕様の意味、解釈のされかたがわかった。
- 他人が書いたものより、自分で書いたものの方が、チェックに身が入る
  - 他人が書いた例は、そこそこきちんとしている。

C++コードの自動生成

- 静的なチェック完了済みであること
- サポートされていない演算子
- explicit な記述のみOK
- implicit な部分は別途C++コードを作成
- 専用のライブラリにより、VDM-SLの型をサポート

外付けコード

- プログラムの制御をする部分が必要
  → 外付けコードでプログラム制御
- pre-, post- の関数のチェックや関数・操作の呼出を行う

(インタプリタ上でチェックしていた時は、各関数がどのようにつながるか良くわからなかった)

外付けコードの作成

- 通信用に共通のライブラリを作成
- 呼出にかかる手続きが煩雑
  - 各操作・関数に対応する呼出関数を作成
  - 変数の型変換
  - pre-, post- のチェック
- 型変換が大変なので、今後は、VDM-SLの外で作成したデータを使っての呼出を少なくしたい

まとめ(1)

- シミュレータの今後について
  - 現実の運動装置と同等の機能
  - 時間要素の検討

まとめ(2)

- 0からの出発という意味で、
  - まず、書いてみたのが良かった
    - 内容の手本(Z)と書き方のがあって良かった
  - インタプリタで動かすことにより、(記述することに関して)理解が深まった
  - C++とリンクして実際のアプリケーションとすることで、有用性が実感できた
    - 今後定量的な評価が欲しい(したい)

形式仕様言語の処理系

— CafeOBJ言語を例に —

(株)SRA/KTL
澤田 寿実
sawada@sra.co.jp

2001年5月23日

Slide 1

形式的方法

- アブリオリに正当性を保証するものでは無い：
  - システムに対する理解を深め，
  - 正しいものである，という確信を高めてくれる．
- 形式仕様言語
  OBJ, TLA, PVS, CafeOBJ, Larch, CASL, CCS, VDM, Z, ...
- 様々な数学 (論理) 的枠組と対応する言語が存在する．
  問題に応じて適宜使い分けるのが良いが，技術者支援 — 計算機によるサポート (処理系・ツール)— が不十分．
- 検証:
  - 要求された性質をシステムが持つかどうかを調べる．
  - 二つの主要な方法:
    * モデル検査
    * 定理証明

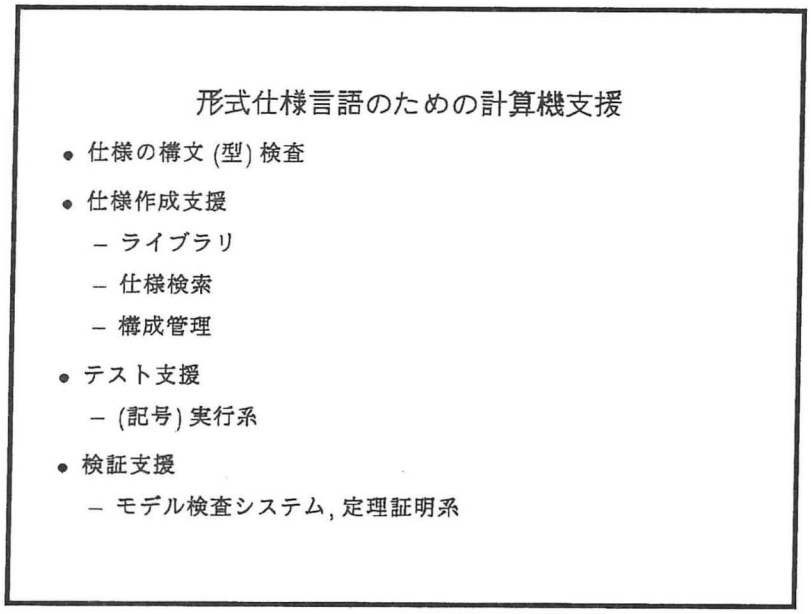
形式技法に関して，万能の道具は存在しない．問題の分析過程で，目的に応じて「どこに，何を，どのように」適用するのかを明確にして，用いるのが良い．また，対象とする問題の全てに形式技法を用いるとするのは，一般的には不経済で，無意味なことが多い．

計算機支援で最も重要となるものの一つが，検証を支援するツールである．そもそも形式技法を用いる大きな目的の一つは，問題を厳密に分析し，記述すること．記述されたものの正しさを出来るだけ厳密に調べること，である．現在広く用いられている検証方法としては，大きく，モデル検査，定理証明の二つの方法がある．

モデル検査では，対象システムの有限状態モデルを構築し，可能な全ての状態において所望の性質が保証されるかどうかを調べる方法である．検査自体は完全に自動的で，高速である．定理証明の場合は，システムとその性質を論理式で表現し，所望の性質が成立するかどうかを定理を証明することにより行う．さまざまな定理証明ツールがある．

SRA 先端技術研究所

Slide 2



一口に形式的技法といっても様々なものがあり, またそれに対応した計算機支援ツール, 言語も当然異なる. Slide 2 で「仕様」とあるのは, その手法に対応した仕様言語によって記述された仕様の意味である.

仕様の構文検査は最低限必要とされる計算機支援である. その昔, C 言語のための lint というツールがあった. コードを静的に解析して, 型の不整合や, 初期化されていない変数の参照などの検出を行うものであるが, プログラマが犯す多くのつまらないミスを, これでチェックすることが出来る. 形式仕様言語の場合も全く同じである.

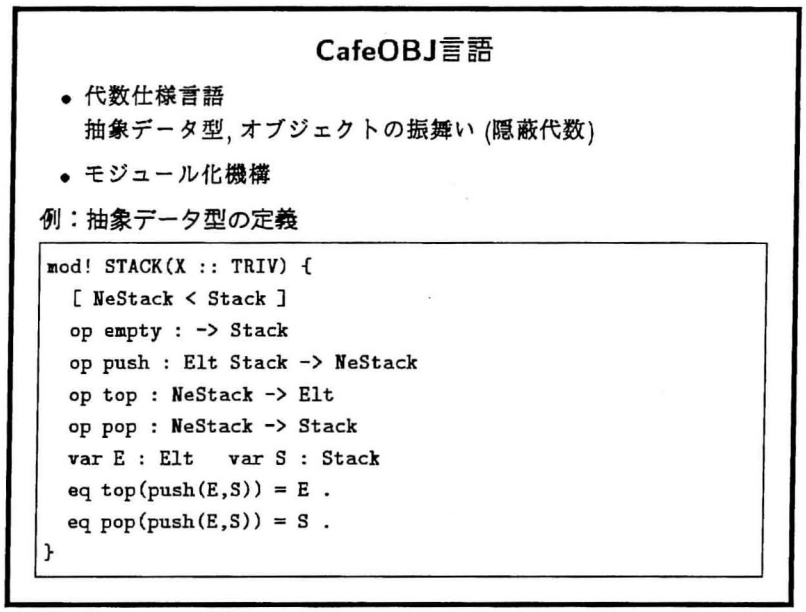
通常のプログラム言語の場合と同様に, 仕様を作成することを支援するための道具が必要である. コーディングのためのツールやライブラリ, 構成管理等, プログラム言語のための道具と本質的に変わりはない. 仕様検索というのは, 目新しいかもしれないが, シグネチャマッチングという技法を用いて, 特定のインタフェース条件を満足するような仕様を探索するものである. 通常のプログラム言語で, シグネチャ(インタフェース仕様) を記述することの出来るものならば, 同じ事が出来る. 形式仕様言語の場合は, もし適切な処理系があれば, これに加えて意味的な条件を付加した探索が可能である.

テスト支援と検証支援は重なる部分がある. どちらも書かれた仕様が, 意図したものかどうかを調べるためのものである. ここでテスト支援と称しているのは, 書かれた仕様をなんらかの方法で記号実行するような仕組みを想定したものである. 代数仕様の場合は, 項書換え系と呼ばれる体系によって, 比較的簡単に実現できる. 検証支援を行うためには, 単なる記号実行にとどまらず, 定理証明あるいはモデル検査などを実行するためのシステムが必要となる.

以下では, 代数仕様言語の一種である CafeOBJ 言語の処理系を例にして, その構成要素, 成立ちについて具体的な説明を行う.

SRA 先端技術研究所

Slide 3



CafeOBJの処理系について説明するため, 最初に CafeOBJ言語の概要を紹介して置く必要がある. CafeOBJは一般に「代数仕様」と呼ばれるグループに属する形式仕様言語である. このグループに属する言語として, 他に OBJ, CASL, ActOne, Larch 等の言語が有名である. もともと「データ型とは何か」という研究を起源として発展してきたものであり, 「抽象データ型 = 代数」 という定式化を基礎とする. 近年「隠蔽代数 (hidden algebra」という体系を用いて, 状態を持ったシステムを簡潔に記述できるようになった. 言語としてこれを最初に採り入れたのは CafeOBJである. その後 BOBJ という言語も出現した. これら以外に, 状態を陽に扱える代数仕様言語は無い.

Slide 3 では, CafeOBJ言語でスタックを定義した例を示してある. Slide 4 に代数仕様言語の特徴をまとめた. 代数仕様の大きな欠点として, 情報隠蔽 (抽象化) が充分でないということがあげられる. まとめると以下のようになるであろう:

- 実装の詳細を隠していない
- 始代数意味論
- 同じ (同型) の実装で無い限り別物
- ポインターと配列でスタックを実装不可能

隠蔽代数を用いた状態表現は, これらの欠点を克服するものである. その概要を Slide 5 と 6 にまとめた.

SRA 先端技術研究所

Slide 4

代数仕様の特徴

- 強力なモジュール化機構
- 等式によりデータ間の同値関係を定義
- 項表現により基準実装の存在を保証 (始代数)
  - $\mathtt{push}(a_1, \mathtt{push}(a_2, \mathtt{push}(\ldots, \mathtt{push}(a_n, \mathtt{empty}))))$
  - $\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\ldots \mathtt{s}(0)))))$
- データ帰納法
  - $\mathtt{P}(\mathtt{empty})$
  - $(\forall \mathtt{S} : \mathtt{Stack})(\forall \mathtt{E} : \mathtt{Elt})\ \mathtt{P}(\mathtt{S}) \Rightarrow \mathtt{P}(\mathtt{push}(\mathtt{E}, \mathtt{S}))$
- 項書換えにより記号実行可能

  $\mathtt{top}(\mathtt{pop}(\mathtt{push}(1, \mathtt{push}(2, \mathtt{E})))) \Rightarrow 2$

Slide 5

振舞の記述

振舞抽象

- 振舞いが同じオブジェクトは同じ (違うと観測されない限り同じ.)
- あらゆるメソッド適用について同じ属性値を返すもののみ等しい.

例：フラグオブジェクト

- メソッド: 上げる，下げる，反転する
- 属性: 上がっているかの真偽値
- 実装：

  整数フラグ：(二倍，二倍して一加える，一加える，偶数か)

  ブールフラグ: (真，偽，否定，真偽値)

  ヒストリフラグ: (真偽値の履歴，トップ要素の真偽値)

SRA 先端技術研究所

Slide 6

CafeOBJにおける振舞仕様

- 振舞を等式で (代数的に) 定義
  - 隠蔽代数 (hidden algebra, Goguen)
  - 可視ソート (データ型) と隠蔽ソート (状態)
- 振舞等式 s = t:
  振舞が同じオブジェクト s と t は "置換可能"
  `beq pop(push(E,S)) = S .` (無限の振舞)
- 一階述語論理 (PigNose)
  隠蔽ソート上の述語 = 状態の記号表現 (抽象化)
- 振舞仕様 = 抽象データ型 + 抽象状態機械

Slide 7

CafeOBJシステム構成

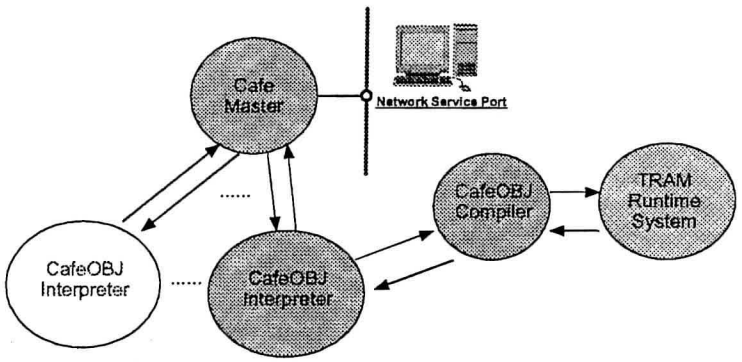


Figure 1: Cafe システムの構成

Slide 7 に CafeOBJシステム全体の構成を示した.

- CafeMaster : IP ポートを介して他システムと CafeOBJインタプリタとのセッションを実現する.
- CafeOBJインタプリタ : CafeOBJ言語のインタプリタ.
- CafeOBJ compiler : CafeOBJ 言語から項書換え機械 (TRAM) へのコンパイラ. CafeOBJインタプリタから直接起動出来る.
- TRAM Rntime System : 項書換え機械の実行系.

以下では CafeOBJ インタプリタに話を絞って説明を行う.

SRA 版 CafeOBJインタプリタは Common Lisp で記述されており (約 84,000行), Common Lisp が稼働するプラットフォームであれば容易にインストール出来る. 一部システム依存の部分 (file I/O, プロセス制御) があり, 現在サポートされている処理系は以下の通りである :

GCL 但し UNIX(Linux) 上の処理系のみ.

ACL(Allegro Common Lisp) version 5.0 以降の ACL. Unix(Linux), Windows のもとで動作するもの.

CMU CL Unix(Linux) 上の処理系のみ.

CLISP Unix, Windows 上の処理系のみ完全サポート. Mac 上のものは外部のプロセス起動が未対応.

MCL version 4.0 以降の MCL (Machintosh Common Lisp).

現在, ACL6.0 版によるバイナリ配布が, Linux(RH6.0 相当) と Windows 用に配布されており, SRA の FTP サイトからダウンロード出来る :

`ftp://ftp.sra.co.jp/pub/lag/CafeOBJ/cafeobj/*`

- `cafeobj-1.4.5-i386linux.tar.gz` – 3M bytes(8M)
- `cafeobj-1.4.5-win.zip` – 7M bytes(19M*2)

Slide 8

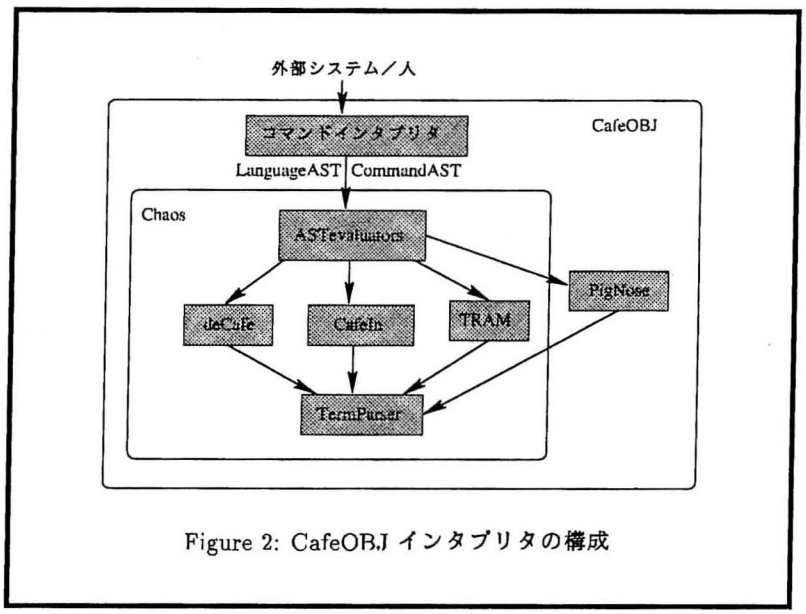


Figure 2: CafeOBJ インタプリタの構成

Figure 2 にCafeOBJインタプリタの構成を示した.

CafeOBJ CafeOBJ 言語構文の定義, 対話環境の提供を行う.

コマンドインタプリタ 利用者や他システムとの対話を行う. コマンドインタプリタは, 入力されたコマンドや CafeOBJ 言語の構文を, 後で説明する **Chaos** システムレベルの抽象構文に変換し, 適当な実行環境を設定しつつそれらの評価系を起動する, というのが主要な機能である.

システムの実際の処理系は, **Chaos** であり, **CafeOBJ** は, その表層的な外部インタフェースを提供するものである.

Chaos システムの中核. CafeOBJ 言語やトップレベルコマンドに対応した抽象構文定義 (図ではそれぞれ, `Language AST` および `Command AST` として示されている) その評価系 (**AST evaluators**) から構成される. つまり処理系の本体は **Chaos** である.

AST evaluators それぞれの言語要素やコマンドの抽象構文に対応して, 適切な実行環境を整えた上で, 他の内部サブシステムから外部インタフェースとして提供されている機能群を呼び出す.

CafeIn 項書換えエンジン. 現在文脈に対応したモジュールの環境のもとでの項書換えに関する一切の処理を行う.

deCafe モジュールの階層構造に関する処理一切を扱う. モジュールの輸入関係, モジュール式の評価系などから構成される.

TermParser 項の構文解析系. モジュールで定義されたシグニチャに従って, 与えられた文字列を構文解析し, 項のシステム内部データ構造へ変換する.

TRAM 外部の項書換えエンジン (**brute**) とのインタフェース機構を提供する. 内部処理については特別なことはなく, 相手をするエンジンの要求に従った項書換え系定義や, 項表現の相互変換が主要な機能である.

SRA 先端技術研究所

PigNose CafeOBJ言語の検証支援システムである. 等号の付いた多ソート一階述語論理の定理証明系をベースとするものであり, 仕様詳細化の検証, 安全性モデル検査機能を提供する.

この他に図では示さなかったが ThStuff というサブシステムがある. これは, apply 系コマンド群のインタプリタとその実行系であり, CafeOBJ と Chaos の中間に位置付けられる. これについては説明を省略する.

Slide 9

Chaos の AST の例

モジュール宣言の AST インスタンス例

```
(%module-decl "2TUPLE" :object :system
 ((%import :protecting "TRIV" "C1")
  (%sort-decl (%sort-ref "2Tuple" nil) nil)
  (%op-decl ("<<" "_" ";" "_" ">>")
   ((%sort-ref "Elt" "C1") (%sort-ref "Elt" "C2"))
   (%sort-ref "2Tuple" nil)
   (%opattrs nil nil nil nil nil t nil)
   nil)
        :
  (%axiom-decl :equation (":BDEMOD")
   ("2*" "<<" "e1" ";" "e2" ">>")
   ("e2") nil nil)))
```

SRA 先端技術研究所

Slide 10

CafeOBJインタプリタの構成要素

- コマンドインタプリタ
  外界 (人, 他システム) とのインタフェース
- モジュール式評価系 (deCafe)
  - モジュールの輸入
  - モジュールに関する演算
  - 改名, 和, インスタンス作成
- 項書換え系 (CafeIn)
  - 項 (式) の記号実行 (評価) 系
  - 実行トレース, ステップ実行
  - 種々の実行コマンド
    $t == t', t = b = t', t => t'$

Slide 11

- 定理証明系 (PigNose)
  - (等号付き) 多ソート一階述語論理系の自動定理証明系
  - 仕様詳細化検証
  - 安全性モデル検査
- ユーティリティ群

Slide 12

主要な内部データ構造

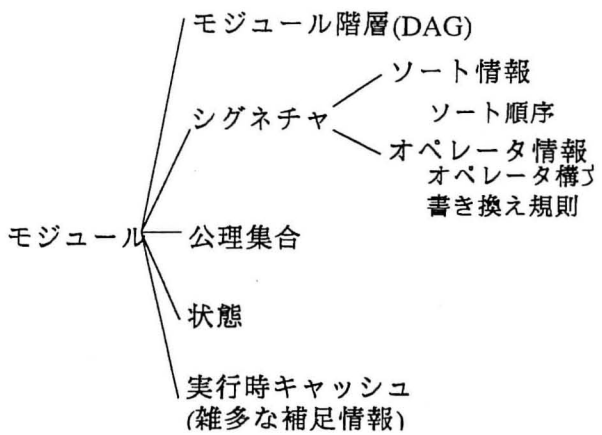


Figure 3: モジュール

Slide 13

項書換え系

- CafeOBJモジュールの操作的意味の基盤の一つ.
- あるモジュールの文脈で 項 の評価を行う：
  - 条件付き順序ソート項書換え系.
  - モジュールの証明計算 (proof calculus) に関して健全 (sound).
- コマンド (一部)
  - red：項の簡約化
  - bred：振舞文脈を考慮した項の簡約化
  - cbred：(circular) coinductive rewriting による振舞等価判定

Slide 14

TRS の構成

- 平坦化されたモジュール $M :=< \Sigma, E \cup B \cup T >$
  - $\Sigma :=< S, \leq, O >$: 順序ソートシグネチャ
  - E : 等式
  - B : 振舞等式
  - T : 遷移ルール
- 対応する項書換えシステム (TRS)

$$\mathcal{TRS} :=< \Sigma', \mathcal{A}, \mathcal{R} >$$

  - Σ' : モジュール M のシグネチャ + エラーソート & エラーオペレータ
  - $\mathcal{A}$: オペレータの E-セオリ (associative, commutative, etc.)
  - $\mathcal{R}$: モジュールの公理から得られた書き換え規則
    - $\mathcal{R} = E^{\rightarrow} \uplus B^{\rightarrow} \uplus T^{\rightarrow}$

書き換え規則の集合は, 各公理を左辺から右辺への書き換え規則とみなして得られたものである.

- $E^{\rightarrow}$: rules from E (等式から得られたもの)
- $B^{\rightarrow}$: rules from B (振舞等式から得られたもの)
- $T^{\rightarrow}$: rules from T (遷移規則から得られたもの)

CafeIn が扱える E-セオリは以下の通りである :

E セオリ	公理
結合則 (A)	$X + (Y + Z) = (X + Y) + Z$
可換則 (C)	$X + Y = Y + X$
単位元 (Z)	$X + 0 = X$
	$0 + X = X$
ベキ等則 (I)	$X + X = X$

SRA 先端技術研究所

Slide 15

書き換え関係 $\rightarrow_{\mathcal{R}}$

$$\frac{t,\ p,\ (l \rightarrow r \text{ if } C) \in \mathcal{R},\ t|_p \equiv_{\mathcal{A}} \theta(l),\quad \theta(C) \rightarrow^*_{\mathcal{R}} \text{true}}{t \rightarrow_{\mathcal{R}} t[p \leftarrow \theta(r)]}$$

ここで下の条件が満足されなければならない.

(C-1) $t[p \leftarrow \theta(r)]$ is a well-sorted term.

- t : Σ-項 (シグネチャ Σ の構文的に正しい項).
- p : 項の出現位置
- $t|_p$: 出現位置 p にある, 項 t の副項 (subterm).
- θ : a Σ-置換. i.e., a function from Σ-項から Σ-項への関数で,
  - ソート S に関して, もし項 $t \in S$ なら $\theta(t) \in S$.
  - $\theta(f(s_1, \ldots, s_n)) = f(\theta(s_1), \ldots, \theta(s_n))$.
- $t[p \leftarrow t']$: 項 t の出現位置 p にある副項を, 項 t' で置き換えられて得られる項.

SRA 先端技術研究所

Slide 16

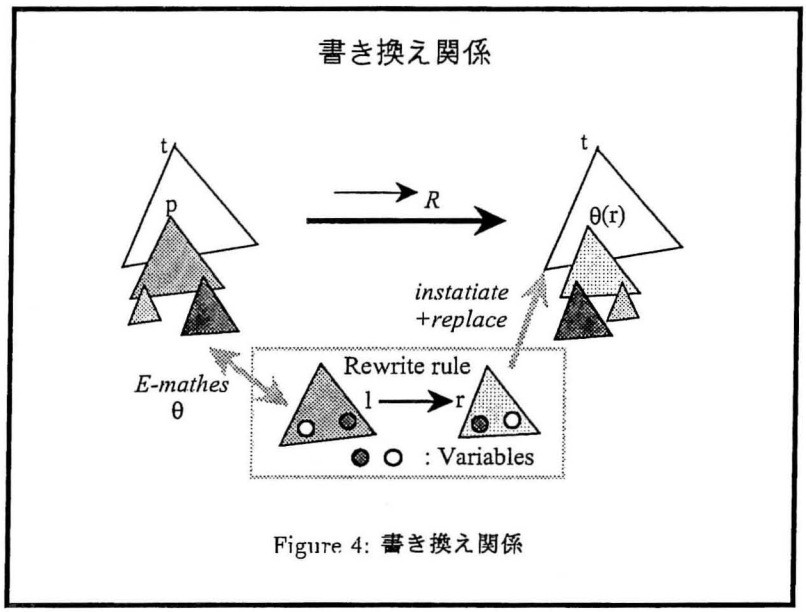


Figure 4: 書き換え関係

Figure 4 に示した書き換え関係 ($\rightarrow_{\mathcal{R}}$) は, 次の条件を満足する必要がある：

- 置換 θ は Σ-置換でなければならない, つまりある項 t について, $\theta(t)$ が正しい項とならなければならない.
- $t[p \leftarrow \theta(r)]$ は正しい項とならなければならない.

これらの条件は, 順序ソートの場合必ずしも無条件に満足されるとは限らない.

SRA 先端技術研究所

Slide 17

順序ソートの特徴

```
module SPEC1 {                          B    D
  [ A C < B, C < D ]                   / \ /
  op a : -> A   op b : -> B           A   C
  op c : -> C   op d : -> D
  eq a = b .  ** a = d が成り立つことを演繹できる.
  eq b = c .  ** しかしソート A と D には順序関係がない.
  eq c = d .
}
```

CafeIn では下の条件が満足されていれば良い：

(C-2) 公理の左右両辺の項 t と t' のソートは同じ結合コンポーネント内にあれば良い.

Slide 18

互換性が成り立たないケース

- 条件 (C-2) により, (C-1) つまり互換性が成立しない場合がある.

下のモジュールから得られる TRS は互換でない：

```
module non-compatible1 {
  [A < B]
  op a  : → A
  op a' : → A
  op b  : → B
  op f  : A →  A
  eq a  = b .
  eq a' = b . }
```

- $a = b = a'$ なので, $f(a) = f(a')$ が成り立たなければならない.

しかし

- $f(a) \longleftrightarrow_{\mathcal{R}}^{*} f(a')$ であることを言えない.

なぜなら, $f(b)$ は構文的に正しくない.

Slide 19

エラーソートとエラーオペレータ

モジュール non-compatible1 は次のように変換される：

```
module non-compatible1'
{
  [A < B < ?B] -- ?B is an error sort of sort A and B.
  op a  : → A
  op a' : → A
  op b  : → B
  op f  : ?B → ?B -- error operator of f.
  op f  : A →  A
  eq a  = b .
  eq a' = b .
}
```

non-compatible1' に対応する TRS は互換になる．つまり，

実行時に条件 (C-1) をチェックする必要がない

Slide 20

- モジュール M の Σ-項 t と t' に関して

$$\mathcal{TRS}_{M'} \vdash t \longleftrightarrow_{\mathcal{R}}^{*} t' \implies M \models t = t'$$

- pop(push(1,push(2,empty))) の top を求める事が出来る：

```
module STACK-OF-NAT {
  [ NeStack < Stack ]
  protecting (NAT)
  op empty : -> Stack
  op push : Nat Stack -> NeStack
  op top : NeStack -> Nat
  op pop : NeStack -> Stack
  eq top(push(X:Nat, S:Stack)) = X .
  eq pop(push(X:Nat, S:Stack)) = S .
}
```

- $t \downarrow \equiv t'$ かつ，t' がエラーソートの項である場合，エラー状態を表示するものと見ることが出来る．

SRA 先端技術研究所

Slide 21

意味的 (同値) 関係の扱い

	equational	behavioural	rewriting
semantic relation	$t = t'$ (eq)	$t = t'$ (beq)	$t \Rightarrow t'$
built-in predicate	`t == t'`	`t =b= t'`	`t ==> t'`
rewriting relation	$\to_{\mathcal{E}}$	$\to_{\mathcal{B}}$	$\to_{\mathcal{R}}$

- $l = l'$(eq) : データの同値関係
  可視ソートに関しては厳密だが隠れソートに関しては弱い
  書き換え関係 $\to_{\mathcal{E}}^*$ によって実装される.
- $l = l'$(beq) : 隠れソート上の振舞い同値関係
  - 全ての可能な観測のもとでの同値性によって規定される.
    書き換え関係 $\to_{\mathcal{B}}^*$ によって実装される.
- $t \Rightarrow t'$: t から t' へ少なくとも一つの遷移がある, という意味.
  - 書き換え関係 $\to_{\mathcal{T}}^*$ によって実装される.

Slide 22

$\to_{\mathcal{E}}$ の実装と reduce コマンド

- 隠れソート H の $t = t'$ は次のとき成り立つ :
  - t と t' が構文的に等しい ($t \equiv_{\mathcal{A}} t'$.)
  - t と t' が振舞的文脈で等しい.
- $\to_{\mathcal{E}}$ は $\overrightarrow{B}$ に含まれる書き換え規則に関して, 振舞的文脈のチェックを行う.

Rcl-$\mathcal{E}$:

$$\frac{t|_p,\ (l \to r \text{ if } C) \in E^{\to} \cup B^{\to},\ t|_p \equiv_{\mathcal{A}} \theta(l),\quad \theta(C) \to_{\mathcal{E}}^* \text{true}}{l \to_{\mathcal{E}} l[p \leftarrow \theta(r)]}$$

- もし $(l \to r \text{ if } C) \in B^{\to}$ なら, $t|_p$ が振舞的文脈にあるときにのみ適用される.
- reduce コマンドは $t{\downarrow_{\mathcal{E}}}$ を計算する. (項を $\to_{\mathcal{E}}^*$ によって評価する).

Slide 23

組み込み述語 == の実装

- 述語 == は各ソート S について定義されている

  pred _==_ : S S

- CafeIn は t == t' を組み込み手続きによって次に示すように実装：

  1. $t{\downarrow_\mathcal{E}}$ と $t'{\downarrow_\mathcal{E}}$ を計算する.
  2. $t{\downarrow_\mathcal{E}} \equiv_\mathcal{A} t'{\downarrow_\mathcal{E}}$ かどうかを調べる.
  3. 上のテストが満足されれば **true** を返す, さもなければ **false**.

- t == t' は TRS (using $\rightarrow_\mathcal{E}$ as rewriting relation) が (ground) confluent で terminating の時のみ決定可能.

Slide 24

例[a]

```
module* HSS (X :: TRIV){
  *[ Hss ]*
  bop put : Elt Hss -> Hss
  bop rest : Hss -> Hss
  bop get : Hss -> Elt
  eq get(put(E, S)) = E .
  beq rest(put(E,S)) = S .
}
```

```
• reduce rest(put(e, st)) == st .
      reduce put(get(rest(put(e1, put(e2, st1)))),
                 st2)
        == put(e2,st2) .
```

[a] from "CafeOBJ report" by Kokichi Futatugi and Razvan Diaconescu

SRA 先端技術研究所

Slide 25

$\rightarrow_{\mathcal{B}}$ とコマンド bred の実装

- $E^{\rightarrow} \cup B^{\rightarrow}$ を書き換え規則として使用する.
  Rcl-$\mathcal{E}$, と同様だが可視な文脈でなくとも良い.
- bred は $t{\downarrow_{\mathcal{B}}}$ を計算する.
- 述語 t =b= t' は組み込みの手続きによって次のように実装されている:
  1. $t{\downarrow_{\mathcal{B}}}$ と $t'{\downarrow_{\mathcal{B}}}$ を計算する.
  2. $t{\downarrow_{\mathcal{B}}} =_{\mathcal{A}} t'{\downarrow_{\mathcal{B}}}$ が等しいかどうかを調べる.
  3. うえのテストが満足されれば **true**, さもなくば **false**.

事務システムにおける形式仕様適用例

佐原伸
日本フィッツ(株)
トレードワン事業部
ss@shinsahara.com

事務システムへの形式仕様適用例 1

事務処理ソフトウェアの問題点

- 曖昧な仕様に起因するトラブルが非常に多い
  - 自然言語により曖昧かつ手続的に仕様を記述
  - それを手続的にプログラミングしていく
  - 重大な欠陥を含み，保守性が無く，再利用性のないソフトウェアが大量に生産されている

事務システムへの形式仕様適用例 2

自然言語仕様の問題点

- 自然言語の曖昧さに起因する設計の欠陥
- 手続的な記述から「意味」を把握する困難
- 仕様記述段階で不完全な手続的設計が混入することによる後続工程の困難
- 自然言語による記述といっても，一種の疑似コード的記述が必要となり，疑似コードの構文をその場で「発明あるいは決定」する時間も馬鹿にならない

事務システムへの形式仕様適用例 3

形式仕様記述言語の採用

- **VDM**採用
  - デンマーク**IFAD**社の**VDM++[2]**とツール **VDMTools**
    - 型チェック・仕様インタープリタ・**Rose**リンク・**C++/Java**生成
  - 採用理由
    - 実用的で「買える」製品は**VDMTools**と**B**
    - 教科書や例題が**VDM**の方が多い
      - **VDM**の方が多少の経験があった
      - **VDM**の日本での使用実績も多少あった
    - **VDM-SL**は**ISO**標準だが、**B**は２社が喧嘩していた
    - **IFAD**社と縁があった

事務システムへの形式仕様適用例 4

対象システム

- トレードワン・システムのサブシステム開発に使用
  - 証券バックオフィス・システムのパッケージ
  - マル優管理システム（昨年秋）
  - オプション・システム（現在）
- システムのアーキテクチャ
  - 3層モデル
  - クライアント側はWebLogic Server上にJavaで開発
  - サーバー側はWebLogic Enterprise上に C++で開発
  - データベースはオラクルを使用している

事務システムへの形式仕様適用例 5

マル優管理システム

- 開発期間
  - 2000年９月～2001年１月初旬
- 開発要員
  - ソフトウェア開発歴20年以上で，フルタイムが3人，0.4人月/月が2人，0.2人月/月が1人の計6人で，投入工数としては4人月/月である
    - 一人を除いて全員証券業務作成の経験はなく，VDM++の使用も初めて
    - C++，Javaを初めて本格的に使用した
    - 形式手法・形式仕様に関しては，4人が知識を持っていて，ある程度の経験もあった

事務システムへの形式仕様適用例 6

マル優管理システム

- 工程
  - サブシステムの設計工程から
    - 「要求仕様定義・分析工程は終わった」ことになっていた
      - ドメイン専門家は３代目
        - 「私にもどうしてこうなっているか分からないんです」
- スケジュール
  - 開発着手から約2ヶ月で一部機能を単体テスト終了
  - 2ヶ月半で全機能を単体テスト終了
- 予想ステップ数
  - 3万行

事務システムへの形式仕様適用例 7

マル優管理システム

- 入力
  - 業務詳細仕様書
    - システム要求」を記述した文書であるが，機能中心に分割され，手続的に記述されていて，実際にはいわゆる「詳細設計書」に近い
  - データベース仕様書
  - 画面仕様書
  - チェック仕様書

事務システムへの形式仕様適用例 8

マル優管理システム

- **VDM++**適用機能
  - **VDMTools**の構文チェック・型チェック機能を主として使用し，詳細設計
  - クラス図のドキュメント化は，**Rational**社**Rose**ツールとの相互接続機能使用
    - だんだん使わなくなった
  - 「トレードワン」システムと独立の**VDM++**共通ライブラリーのテストには，**VDMTools**の仕様実行インタープリタとコードカバレージ機能を使用

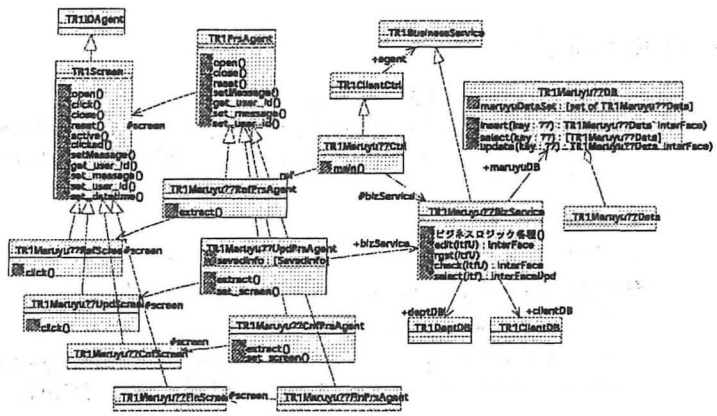
事務システムへの形式仕様適用例 9

マル優管理システム

- クラス構成
  - **GUI**層・ビジネスロジック層・データ層を次図のように構成
- 共通クラス
  - **Object > TR1Object > TR1Date**
  - 日付、集合、系列、文字列、整数、文字

事務システムへの形式仕様適用例 10

マル優管理システム



事務システムへの形式仕様適用例 11

マル優管理システム

- 制約の記述
  - システム開発の最終段階で，サブシステム間インタフェースや，業務詳細仕様書に起因する欠陥が数多く発見されたため，業務詳細仕様書を見て修正することが困難になり，マル優管理システムの制約を記述し整理しなければならなくなった．これは，本来，設計に先立って行うべきことであり，前述の時間的制約のため省略した作業であるが，やはり必要となった．

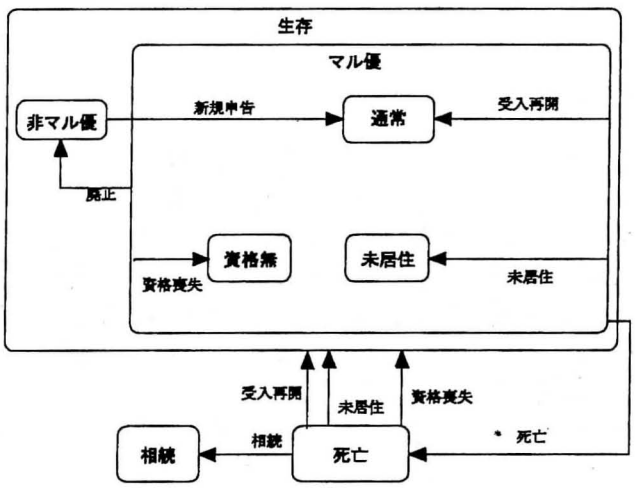
事務システムへの形式仕様適用例 12

マル優管理システム

- 主要な制約
  - マル優顧客の状態（次図）
    - 業務詳細仕様では一つの区分の値として，状態と遷移が混在する形で記述されていたため，仕様に様々な漏れが生じていた
  - 管理項目の制約式
    - 全部で22個抽出
  - 帳票を作成するためのデータの時系列制約
    - 業務上は重要ではないが...
    - バッチ処理時における、オンライン時との乖離の修正に使っていたため，設計仕様および効率上重要な制約となった

事務システムへの形式仕様適用例 13

マル優管理システム



事務システムへの形式仕様適用例 14

マル優管理システム

- 制約式の例
  - 「マル優口座が存在し，かつ，その口座のマル優取扱区分が，通常か未居住か資格無であり，かつ，以下の式が成り立つこと．ここで，Σは「顧客毎の合計」を意味する．」
    - マル優申告額 = マル優余裕額 ＋ Σ一般投信マル優残高 ＋ Σマル限限度額
    - マル優残高 = Σ累投証券残 ＋Σ一般投信マル優残高
    - 累投証券残 ≦ マル限限度額
  - この制約式は，「ある特定の条件下で，マル優限度額を越えた買付がいくらでもできてしまう」という業務詳細仕様書に隠れていた重大な意味的欠陥を発見することに役立った

事務システムへの形式仕様適用例 15

マル優管理システム(適用結果)

分類	VDM++仕様	Java	C++	プログラム合計
事務処理共通クラス	774行			
トレードワン共通クラス	1342行			
制約仕様	1539行			
詳細設計	8102行	9403行	9028行	1万8431行
合計	1万1856行	9403行	9028行	1万8431行

事務システムへの形式仕様適用例 16

マル優管理システム(連結前の欠陥)

欠陥・修正原因	件数
サブシステム間インタフェースの了解違いによるもの	9
ユーザーレビューによる機能追加	1
業務詳細仕様書の考慮漏れ	1
用語集の未整備による項目定義の誤解	1
合計	１２

事務システムへの形式仕様適用例 17

マル優管理システム

- リリース後の欠陥
  - ０
- 単体テスト以前の欠陥の原因＝５７
  - 業務詳細仕様書の考慮漏＝３５
  - チェック仕様書の考慮洩れ＝１３
  - 管理上のミス＝３
    - エラーメッセージの申請忘れなど
  - ユーザー要求＝２
  - 画面仕様書・テーブル仕様書＝それぞれ２

事務システムへの形式仕様適用例 18

マル優管理システム

- 最終履歴
  - 開発期間＝３ヶ月
  - 工数　＝13.3人月
    - 開発　＝12人月
      - 仕様作成　＝５人月
      - プログラム作成　＝６人月
      - 管理　＝１人月
    - 修正　＝1.5人月
      - 仕様修正　=0.8人月
      - プログラム修正　=0.5人月
      - 管理　=0.5人月

事務システムへの形式仕様適用例 19

マル優管理システム

- 考察
  - 品質
    - 連結テスト以後の欠陥
      - すべてVDM++適用以前の工程で発生
      - VDM++適用以前の工程で発生した欠陥のうち，発見可能だった業務詳細仕様書の欠陥は，単体テスト以前にすべて発見し修正
  - 生産性
    - COCOMOによる見積の３分の１

事務システムへの形式仕様適用例 20

マル優管理システム

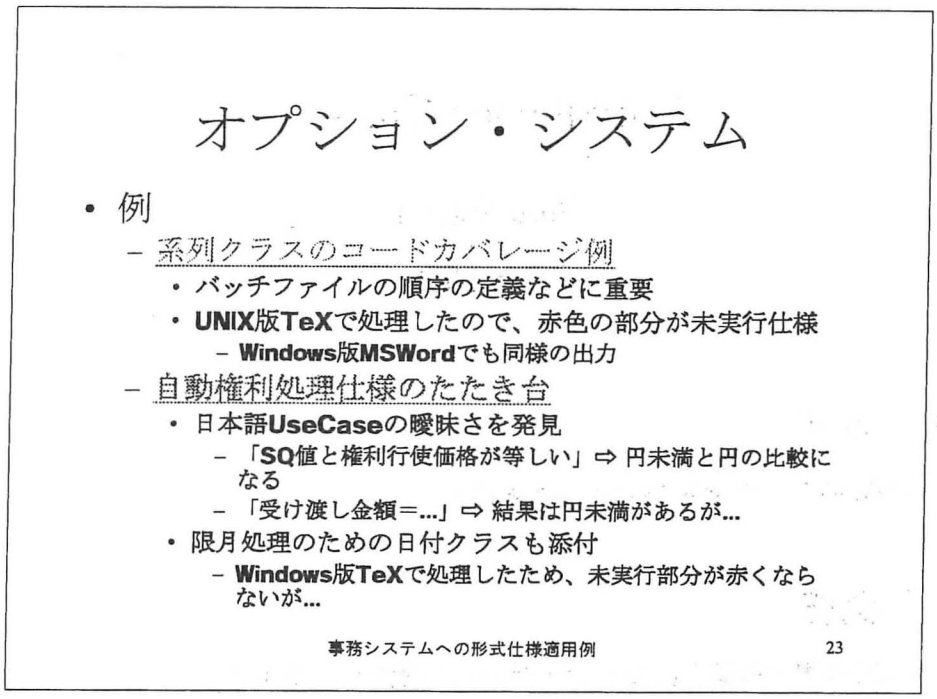
- **VDM++**
  - 仕様を２～３回書き直し
  - 仕様の修正余波を型チェックシステムが指摘
  - 共通クラスは仕様実行とカバレージで**100%**テストできた
  - 従来手法より，仕様記述言語の曖昧さが無く，言語レベルが高いので，仕様を書きやすい
    - 特に役立ったもの
      - » 集合・系列の内包表記、パターンや束縛、構成子、限量子

事務システムへの形式仕様適用例 21

オプション・システム

- 日経**225**のオプション・システム
- **VDM++**適用範囲
  - 要求分析工程から適用
  - **VDM++**インタープリタによる要求仕様検証
  - テスト・フレームワーク作成
    - **UseCase**に対応したテストシナリオでビジネス要求をテスト
  - 設計フレームワーク詳細化
    - 通称「トレイ・モデル」
  - **VDM++**の日本語化
    - 識別子、文字列、注釈

事務システムへの形式仕様適用例 22


オプション・システム
• 例
– 系列クラスのコードカバレージ例
• バッチファイルの順序の定義などに重要
• UNIX版TeXで処理したので、赤色の部分が未実行仕様
– Windows版MSWordでも同様の出力
– 自動権利処理仕様のたたき台
• 日本語UseCaseの曖昧さを発見
– 「SQ値と権利行使価格が等しい」⇨ 円未満と円の比較になる
– 「受け渡し金額＝...」⇨ 結果は円未満があるが...
• 限月処理のための日付クラスも添付
– Windows版TeXで処理したため、未実行部分が赤くならないが...
事務システムへの形式仕様適用例
23

Sequence Class

Shin Sahara

May, 2001

1 責任

VDM のシーケンス (sequence) を表す。

2 概要

VDM 標準の演算子にないシーケンスの機能を定義する。

class $Sequence$is subclass of $Object$
functions
public

$sum[@elem] : @elem^* \to @elem$
$sum\,(s) \triangleq$
$\quad accsum[@elem]\,(s, 0)$
pre $\mathsf{is\_}(s, \mathbb{Z}^*) \lor \mathsf{is\_}(s, \mathbb{N}^*) \lor$
$\quad \mathsf{is\_}(s, \mathbb{R}^*) \lor$
$\quad \mathsf{is\_}(s, \mathbb{Q}^*)$;
$accsum[@elem] : @elem^* \times @elem \to @elem$
$accsum\,(s, sum) \triangleq$
$\quad$ cases s :
$\quad\quad [] \to sum,$
$\quad\quad [x] \curvearrowright xs \to accsum[@elem]\,(xs, sum + x)$
$\quad$ end;

public

$isAscendingTotalOrder[@elem] : (@elem \times @elem \to \mathbb{B}) \to @elem^* \to \mathbb{B}$
$isAscendingTotalOrder\,(f)(s) \triangleq$
$\quad \forall i, j \in \mathsf{inds}\ s \cdot f\,(i, j) \Rightarrow f\,(s\,(i), s\,(j)) \lor s\,(i) = s\,(j);$

public

$isDescendingTotalOrder[@elem] : (@elem \times @elem \to \mathbb{B}) \to @elem^* \to \mathbb{B}$
$isDescendingTotalOrder\,(f)(s) \triangleq$
$\quad \forall i, j \in \mathsf{inds}\ s \cdot f\,(i, j) \Rightarrow f\,(s\,(j), s\,(i)) \lor s\,(i) = s\,(j);$

public

$isAscending[@elem] : @elem^* \to \mathbb{B}$
$isAscending\,(s) \triangleq$
$\quad isAscendingTotalOrder[@elem]\,(\lambda x : @elem, y : @elem \cdot x < y)\,(s);$

public

$isDescending[@elem] : @elem^* \to \mathbb{B}$
$isDescending\,(s) \triangleq$
$\quad isDescendingTotalOrder[@elem]\,(\lambda x : @elem, y : @elem \cdot x < y)\,(s);$

public

$sort[@e] : (@e \times @e \to \mathbb{B}) \to @e^* \to @e^*$
$sort\,(f)(s) \triangleq$
cases s :
$[] \to []$,
$[x] \frown \text{-} \to$
let $l = sort[@e]\,(f)\,([s\,(i) \mid i \in \text{inds } s \cdot f\,(s\,(i), x)])$,
$m = [s\,(i) \mid i \in \text{inds } s \cdot x = s\,(i)]$,
$r = sort[@e]\,(f)\,([s\,(i) \mid i \in \text{inds } s \cdot f\,(x, s\,(i))])$ in
$l \frown m \frown r$
end;

public

$ascendingSort[@e] : @e^* \to @e^*$
$ascendingSort\,(s) \triangleq$
$sort[@e]\,(\lambda\, x : @e, y : @e \cdot x < y)\,(s)$;

public

$descendingSort[@e] : @e^* \to @e^*$
$descendingSort\,(s) \triangleq$
$sort[@e]\,(\lambda\, x : @e, y : @e \cdot x > y)\,(s)$;

public

$take[@elem] : \mathbb{Z} \times @elem^* \to @elem^*$
$take\,(i, s) \triangleq$
cases mk-(i, s) :
mk-$(0, \text{-}) \to []$,
mk-$(n, [x] \frown xs) \to [x] \frown take[@elem]\,(dec\,(n), xs)$,
mk-$(\text{-}, []) \to []$
end;
$dec : \mathbb{Z} \to \mathbb{Z}$
$dec\,(i) \triangleq$
$i - 1$;

public

$drop[@elem] : \mathbb{Z} \times @elem^* \to @elem^*$
$drop\,(i, s) \triangleq$
cases mk-(i, s) :
mk-$(n, [x] \frown xs) \to$
if $n > 0$
then $drop[@elem]\,(n-1, xs)$
else $[x] \frown xs$,
mk-$(-, []) \to []$
end;

public

$last[@elem] : @elem^* \to @elem$
$last\,(x) \triangleq$
cases x :
$[x] \to x$,
$[-] \frown xs \to last[@elem]\,(xs)$
end
pre $x \neq []$;

public

$fmap[@elem1, @elem2] : (@elem1 \to @elem2) \to @elem1^* \to @elem2^*$
$fmap\,(f)(l) \triangleq$
cases l :
$[] \to []$,
$[x] \frown xs \to [f\,(x)] \frown (fmap[@elem1, @elem2]\,(f)\,(xs))$
end

end *Sequence*

Test Suite : vdm.tc

Name	#Calls	Coverage
Sequence'accsum	0	0%
Sequence'ascendingSort	2	88%
Sequence'dec	0	0%
Sequence'descendingSort	0	0%
Sequence'drop	0	0%
Sequence'fmap	0	0%

Name	#Calls	Coverage
Sequence`isAscending	0	0%
Sequence`isAscendingTotalOrder	0	0%
Sequence`isDescending	0	0%
Sequence`isDescendingTotalOrder	0	0%
Sequence`last	0	0%
Sequence`sort	16	96%
Sequence`sum	0	0%
Sequence`take	0	0%
Total Coverage		**26%**

索 引

accsum, **2**
ascendingSort, **3**

dec, **3**
descendingSort, **3**
drop, **4**

fmap, **4**

isAscending, **2**
isAscendingTotalOrder, **2**
isDescending, **2**
isDescendingTotalOrder, **2**

last, **4**

Sequence, 2
sort, **3**
sum, **2**

take, **3**

AutoExRights Class

佐原伸

2001年5月22日

1 責任

自動権利処理・自動権利行使／自動権利放棄の要件定義。

2 概要

インザマネーの時権利行使し、アオウトオブマネーの時権利放棄する。ただし、実際には受渡代金を考慮して、受渡代金が正の場合のみ権利行使する。

```
class AutoExRights
types
        public ExRightsStatus = AtTheMoney | InTheMoney | OutOfTheMoney;
        public CallPut = Call | Put;
        public Money2 = ℚ
        inv money ≜ isCorrectDecimalPlace (money, 2);
        public Money = ℕ;
        OptionRec :: key : char*
                     dt : Date
                     amount : ℕ
                     commission : ℕ
                     tax : ℕ
functions
public
        isCorrectDecimalPlace : ℚ × ℕ → 𝔹
        isCorrectDecimalPlace (x, y) ≜
          let f = λx : ℚ · x × 10 ↑ y in
          f (x) = floor f (x);
public
        getExRightsStatus : CallPut × Money2 × Money
                                →
                            ExRightsStatus
        getExRightsStatus (callPut, SQ, ExRightsPrice) ≜
          let x = SQ − ExRightsPrice,
              y = SQ > ExRightsPrice in
          cases mk-(callPut, x, y) :
            mk-(-, 0, -) → AtTheMoney,
            mk-(Call, -, true), mk-(Put, -, false) → InTheMoney,
            mk-(Call, -, false), mk-(Put, -, true) → OutOfTheMoney
          end;
public
```

$test : () \to ExRightsStatus \times ExRightsStatus \times ExRightsStatus \times ExRightsStatus \times ExRightsStatus$

$test\ () \triangleq$

mk- ($getExRightsStatus$ (CALL, 10, 10),
$getExRightsStatus$ (CALL, 10.1, 10),
$getExRightsStatus$ (CALL, 10.1, 11),
$getExRightsStatus$ (PUT, 10.1, 10),
$getExRightsStatus$ (PUT, 10.1, 11));

public

$moneyBlc : Money2 \times Money \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to Money$

$moneyBlc\ (SQ, ExRightsPrice, amount, commission, tax) \triangleq$

floor (abs $(SQ - ExRightsPrice) \times amount - commission - tax$)

end *AutoExRights*

Test Suite : vdm.tc

Name	#Calls	Coverage
AutoExRights'getExRightsStatus	5	√
AutoExRights'isCorrectDecimalPlace	6	√
AutoExRights'moneyBlc	1	√
AutoExRights'test	1	√
Total Coverage		**100%**

class $Date$is subclass of $Object$
values
$jdMinusMjd = 2400000.5;$
private
$asString = \text{new}Integer().asString;$
private
$STR = \text{new}String();$
$DayNameList = [\text{MON}, \text{TUE}, \text{WED}, \text{THR}, \text{FRI}, \text{SAT}, \text{SUN}]$
types
public$Time = \mathbb{R};$
public$DateTime = \mathbb{R};$
public$TermLength = \mathbb{R};$
public$DayName = \text{MON} \mid \text{TUE} \mid \text{WED} \mid \text{THR} \mid \text{FRI} \mid \text{SAT} \mid \text{SUN};$
public$DayNumber = \mathbb{N}$
inv $d \triangleq 0 \leq d \wedge d \leq 6$
instance variables
private $modifiedJulianDate : \mathbb{R} := 0;$

functions
public
$today : () \rightarrow Date$
$today\,() \triangleq$
$fromDate\,(2001, 3, 1);$
public
$lt2 : Date \times Date \rightarrow \mathbb{B}$
$lt2\,(d1, d2) \triangleq$
$mjd\,(d1) < mjd\,(d2);$
public
$gt2 : Date \times Date \rightarrow \mathbb{B}$
$gt2\,(d1, d2) \triangleq$
$mjd\,(d1) > mjd\,(d2);$
public
$le2 : Date \times Date \rightarrow \mathbb{B}$
$le2\,(d1, d2) \triangleq$
$\neg\, gt2\,(d1, d2);$

public

$ge2 : Date \times Date \to \mathbb{B}$
$ge2\,(d1, d2) \triangleq$
$\quad \neg\, lt2\,(d1, d2);$

public

$former : Date \times Date \to \mathbb{B}$
$former\,(d1, d2) \triangleq$
$\quad lt2\,(d1, d2);$

public

$eq2 : Date \times Date \to$
$\quad \mathbb{B}$
$eq2\,(d1, d2) \triangleq$
$\quad mjd\,(d1) = mjd\,(d2);$

public

$leapYear : \mathbb{Z} \to$
$\quad \mathbb{B}$
$leapYear\,(y) \triangleq$
$\quad y \text{ mod } 400 = 0 \vee (y \text{ mod } 100 \neq 0 \wedge y \text{ mod } 4 = 0);$

public

$dayNumber : Date \to$
$\quad DayNumber$
$dayNumber\,(d) \triangleq$
$\quad \text{let } d1 = \text{floor } (mjd\,(d)) \text{ in}$
$\quad (d1 - 4) \text{ mod } 7;$

public

$dateToYmd2 : Date \to \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
$dateToYmd2\,(d) \triangleq$
$\quad \text{mk-}\,(year\,(d), month\,(d), day\,(d));$

public

$weekDay : Date \to DayName$
$weekDay\,(d) \triangleq$
$\quad DayNameList\,(dayNumber\,(d));$

public

$dayOfWeek : DayName \rightarrow DayNumber$
$dayOfWeek\,(dayName) \triangleq$
cases $dayName$:
SUN $\rightarrow 0$,
MON $\rightarrow 1$,
TUE $\rightarrow 2$,
WED $\rightarrow 3$,
THR $\rightarrow 4$,
FRI $\rightarrow 5$,
SAT $\rightarrow 6$
end;

public

$firstDayOfWeek : \mathbb{Z} \times \mathbb{Z} \times DayName \rightarrow Date$
$firstDayOfWeek\,(y, m, dayName) \triangleq$
let $dnum = dayOfWeek\,(dayName)$,
$firstDate = firstDayOfMonth\,(y, m)$,
$diff = dnum - dayNumber\,(firstDate)$ in
cases true :
$(diff = 0) \rightarrow firstDate$,
$(diff > 0) \rightarrow firstDate.addDays\,(diff)$,
$(diff < 0) \rightarrow firstDate.addDays\,((7 + diff)$ mod $7)$
end;

public

$lastDayOfWeek : \mathbb{Z} \times \mathbb{Z} \times DayName \rightarrow Date$
$lastDayOfWeek\,(y, m, dayName) \triangleq$
$firstDayOfWeek\,(y, (m + 1), dayName).subtractDays\,(7)$;

public

$nthDayOfWeek : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times DayName \rightarrow$
$Date \times \mathbb{B}$
$nthDayOfWeek\,(y, m, n, dayName) \triangleq$
let $firstDate = firstDayOfWeek\,(y, m, dayName)$,
$RESULT = firstDate.addDays\,(7 \times (n - 1))$ in
cases $month\,(RESULT)$:
$(m) \rightarrow$ mk-$(RESULT$, true),
others $\rightarrow$ mk-$(RESULT$, false)
end;

public
$firstDayOfMonth : \mathbb{Z} \times \mathbb{Z} \rightarrow Date$
$firstDayOfMonth\,(y, m) \triangleq$
$fromDate\,(y, m, 1)$;

public
$lastDayOfMonth : \mathbb{Z} \times \mathbb{Z} \rightarrow Date$
$lastDayOfMonth\,(y, m) \triangleq$
$fromDate\,(y, m + 1, 1).subtractDays\,(1)$;

public
$year : Date \rightarrow \mathbb{Z}$
$year\,(d) \triangleq$
if $monthAux\,(d) < 14$
then $yearAux\,(d) - 4800$
else $yearAux\,(d) - 4799$;

public
$month : Date \rightarrow \mathbb{Z}$
$month\,(d) \triangleq$
if $monthAux\,(d) < 14$
then $monthAux\,(d) - 1$
else $monthAux\,(d) - 13$;

public
$day : Date \rightarrow \mathbb{Z}$
$day\,(d) \triangleq$
$dayOfMonth\,(d)$;

public

$dayOfYear : Date \rightarrow \mathbb{Z}$
$dayOfYear\,(d) \triangleq$
  let $theFirstDayOfYear = fromDate\,(year\,(d), 1, 0)$ in
  floor $(mjd\,(d) - mjd\,(theFirstDayOfYear))$;
$dayOfMonth : Date \rightarrow \mathbb{Z}$
$dayOfMonth\,(d) \triangleq$
  floor $(dayAsReal\,(d))$;
$dayAsReal : Date \rightarrow \mathbb{R}$
$dayAsReal\,(d) \triangleq$
  $dateToYmdAux\,(d) + 122.1 -$
  floor $(365.25 \times yearAux\,(d)) -$ floor $(30.6001 \times monthAux\,(d))$;
$monthAux : Date \rightarrow$
  $\mathbb{Z}$
$monthAux\,(d) \triangleq$
  floor $((dateToYmdAux\,(d) + 122.1 -$ floor $(365.25 \times yearAux\,(d)))/30.6001)$;
$dateToYmdAux : Date \rightarrow \mathbb{R}$
$dateToYmdAux\,(d) \triangleq$
  let $jd = mjdToJd\,(mjd\,(d))$,
    $century =$ floor $((jd + 32044.9)/36524.25)$ in
  if $jd > 2299160$
  then $jd + 32044.9 + century - century$ div $4 + 0.5$
  else $jd + 32082.9 + 0.5$;
$yearAux : Date \rightarrow$
  $\mathbb{Z}$
$yearAux\,(d) \triangleq$
  floor $(dateToYmdAux\,(d)/365.25)$;

public

$addDays2 : Date \times \mathbb{Z} \rightarrow Date$
$addDays2\,(d, n) \triangleq$
  $d.addDays\,(n)$;

public

$subtractDate : Date \times Date \rightarrow \mathbb{Z}$
$subtractDate\,(d1, d2) \triangleq$
  floor $(mjd\,(d1) - mjd\,(d2))$;

public

$subtractDays2 : Date \times \mathbb{Z} \rightarrow Date$
$subtractDays2\,(d, n) \triangleq$
$d.subtractDays\,(n);$

public

$length : Date \times Date \rightarrow TermLength$
$length\,(d1, d2) \triangleq$
$subtractDate\,(d1, d2)$post cases true :
$(former\,(d1, d2)) \rightarrow RESULT \geq 0,$
others $\rightarrow RESULT \leq 0$
end;

public

$find : Date \times TermLength \rightarrow Date$
$find\,(d, l) \triangleq$
$addDays2\,(d, l)$post $length\,(d, RESULT) = l;$

public

$within : Date \times Date \times TermLength \rightarrow \mathbb{B}$
$within\,(d1, d2, l) \triangleq$
$length\,(d1, d2) \leq l;$

public

$mjdToJd : \mathbb{R} \rightarrow \mathbb{R}$
$mjdToJd\,(r) \triangleq$
$r + jdMinusMjd;$

public

$jdToMjd : \mathbb{R} \rightarrow \mathbb{R}$
$jdToMjd\,(r) \triangleq$
$r - jdMinusMjd;$

public

$ymdToYear : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \rightarrow$
$\mathbb{R}$
$ymdToYear\,(y, m, d) \triangleq$
$y + (m - 1)/12 + (d - 1)/365.25;$

public

$date2string : Date \rightarrow$ char*
$date2string\,(d) \triangleq$
$d.dateToYmdString\,();$

```
public
        string2date : char* → [Date]
        string2date (s) ≜
          let mk-(status, d) = fromString (s) in
          if status = true
          then d
          else nil ;
public
        time2string : Time → char*
        time2string (t) ≜
          is not yet specified;
public
        string2time : char* → [Time]
        string2time (s) ≜
          is not yet specified
operations
public
        mjd : Date →o
              ℝ
        mjd (d) ≜
          (   return d.getMjd ()
          );
public
        fromDate : ℤ × ℤ × ℤ →o
                   Date
        fromDate (y, m, d) ≜
          let [year, month] = if (m > 2)
                              then [y + 4800, m + 1]
                              else [y + 4799, m + 13],
              century = year div 100,
              cc = if (ymdToYear (y, m, d) > 1582.78)
                   then century div 4 − century − 32167
                   else − 32205 in
          (   dcl dt : Date := newDate();
```

```
                dt.setMjd(floor (365.25×year)+floor (30.6001×month)+
d + cc − 0.5 − jdMinusMjd) ;
                return dt
            );
public
        fromString : char* →o
                        B × Date
        fromString (yyyymmdd) △
          (   if ¬ STR.isDigits (yyyymmdd)
              then return (mk- (false, today ())) ;
              let ymd = STR.asInteger (yyyymmdd),
                  y = ymd div 10000,
                  mmdd = ymd mod 10000,
                  m = mmdd div 100,
                  d = mmdd mod 100 in
              if fromDate (y, m, d).dateToYmdString () = yyyymmdd
              then return mk- (true, fromDate (y, m, d))
              else return mk- (false, fromDate (y, m, d))
          );
public
        isDateString : char* →o
                        B
        isDateString (yyyymmdd) △
          return fromString (yyyymmdd).#1;
public
        fromMjd : R →o
                        Date
        fromMjd (modieiedJd) △
          (   dcl dt : Date := newDate();
              dt.setMjd(modieiedJd) ;
              return dt
          );
public
```

$dateToYmd : () \xrightarrow{o} \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
$dateToYmd\ () \triangleq$
return mk-$(year\ (\text{self}), month\ (\text{self}), day\ (\text{self}))$;

public

$dateToYmdString : () \xrightarrow{o} \text{char}^*$
$dateToYmdString\ () \triangleq$
(let $y = year\ (\text{self})$,
$m = month\ (\text{self})$,
$d = day\ (\text{self})$ in
cases true:
$(m < 10 \wedge d < 10) \rightarrow$ return $(asString\ (y) \frown$ "0" $\frown asString\ (m) \frown$ "0" $\frown asString\ (d))$,
$(m < 10) \rightarrow$ return $(asString\ (y) \frown$ "0" $\frown asString\ (m) \frown asString\ (d))$,
$(d < 10) \rightarrow$ return $(asString\ (y) \frown asString\ (m) \frown$ "0" $\frown asString\ (d))$,
others $\rightarrow$ return $(asString\ (y) \frown asString\ (m) \frown asString\ (d))$
end
);

public

$lt : Date \xrightarrow{o} \mathbb{B}$
$lt\ (d) \triangleq$
return $mjd\ (\text{self}) < mjd\ (d)$;

public

$gt : Date \xrightarrow{o} \mathbb{B}$
$gt\ (d) \triangleq$
return $mjd\ (\text{self}) > mjd\ (d)$;

public

$le : Date \xrightarrow{o} \mathbb{B}$
$le\ (d) \triangleq$
return $\neg\, gt\ (d)$;

public

$ge : Date \xrightarrow{o} \mathbb{B}$
$ge\ (d) \triangleq$
return $\neg\, lt\ (d)$;

```
public
        eq : Date ->o
             B
        eq (d) ≜
          return (mjd (self) = mjd (d));
public
        addDays : Z ->o Date
        addDays (n) ≜
          (   dcl dt : Date := newDate();
              dtsetMjd(mjd (self) + n) ;
              return dt
          );
public
        subtractDays : Z ->o Date
        subtractDays (n) ≜
          (   dcl dt : Date := newDate();
              dtsetMjd(mjd (self) - n) ;
              return dt
          );
        setMjd : R ->o ()
        setMjd (d) ≜
          modifiedJulianDate := d;
        getMjd : () ->o R
        getMjd () ≜
          return modifiedJulianDate
end Date
```

Test Suite : vdm.tc

Name	#Calls	Coverage
Date'addDays2	0	0%
Date'date2string	0	0%
Date'dateToYmd2	0	0%
Date'dateToYmdAux	23	85%
Date'day	1	√
Date'dayAsReal	1	√

Name	#Calls	Coverage
Date'dayNumber	3	√
Date'dayOfMonth	1	√
Date'dayOfWeek	3	62%
Date'dayOfYear	0	0%
Date'eq2	0	0%
Date'find	0	0%
Date'firstDayOfMonth	3	√
Date'firstDayOfWeek	3	65%
Date'former	0	0%
Date'ge2	0	0%
Date'gt2	0	0%
Date'jdToMjd	0	0%
Date'lastDayOfMonth	0	0%
Date'lastDayOfWeek	0	0%
Date'le2	0	0%
Date'leapYear	0	0%
Date'length	0	0%
Date'lt2	0	0%
Date'mjdToJd	23	√
Date'month	4	68%
Date'monthAux	10	√
Date'nthDayOfWeek	3	88%
Date'string2date	0	0%
Date'string2time	0	0%
Date'subtractDate	0	0%
Date'subtractDays2	0	0%
Date'time2string	0	0%
Date'today	0	0%
Date'weekDay	0	0%
Date'within	0	0%
Date'year	1	68%
Date'yearAux	12	√
Date'ymdToYear	3	√

Name	#Calls	Coverage
Date'addDays	6	√
Date'dateToYmd	1	√
Date'dateToYmdString	0	0%
Date'eq	0	0%
Date'fromDate	3	86%
Date'fromMjd	0	0%
Date'fromString	0	0%
Date'ge	0	0%
Date'getMjd	32	√
Date'gt	0	0%
Date'isDateString	0	0%
Date'le	0	0%
Date'lt	0	0%
Date'mjd	32	√
Date'setMjd	9	√
Date'subtractDays	0	0%
Total Coverage		**40%**

UML statechart図と振舞仕様を使ったシステム設計

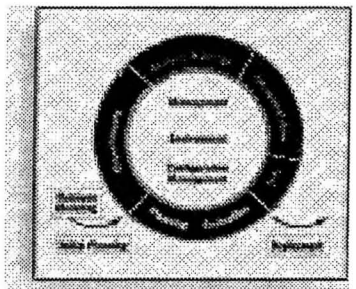
専修大学
ネットワーク情報学部
飯田 周作

背景（1）方法論の発展

- 軽量化（柔軟化）
  - スケーラビリティ
  - アダプタビリティ
  - 細かい流れよりも、全体的な思想

背景（2）スパイラル方式

- 早い段階からテストをする
- リスク管理がしやすい
- やり直しがきく



http://www.rational.co.jp/products/unified_process/capture1.htmlより引用

背景（3）問題の分割

- リスクが高い部分の特定
- その部分に資源を集中させる

背景（4）仕様

- 視点（viewpoint）の導入
  - 複数の視点を使ってシステムをモデル化
- UML

研究の目的

- 方法論の発展に適応した形式手法
  - ピンポイントで形式仕様を使う
- 完璧よりもコストに見合った性能

実際にやったこと

- UML statechartに着目して
  - 設計時の問題点を洗い出し
  - 形式仕様を使って問題点の解決を図った
- 具体的には
  - statechartと振舞仕様の対応を考え
  - statechartの詳細化を振舞仕様でチェックした
- 事例報告というよりは実験結果報告

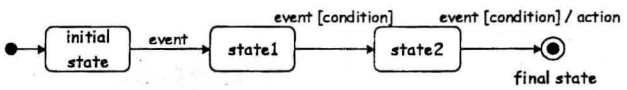
実験の前提

- 方法論は特に決めない
  - 開発過程においてstatechartを使うことを仮定する
- スパイラル型の開発を想定
  - しかし、より上流では実装よりもstatechartによる振舞の記述とそれに基づくテストを行うことを想定

状態機械（state machine）

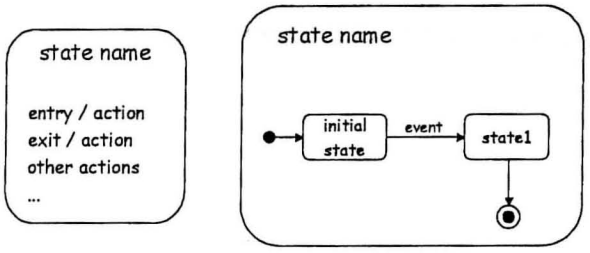
- 状態の集合と、状態の遷移によって特徴付けられるような機械のモデル
- 状態の数は有限（有限状態機械、finite state machine）
- ある瞬間を見ると、必ず1つの状態にある
- 遷移は、イベントによって引き起こされる
- statechartおよび振舞仕様は、この状態機械に対する記法であると考えられる

UML statechart図



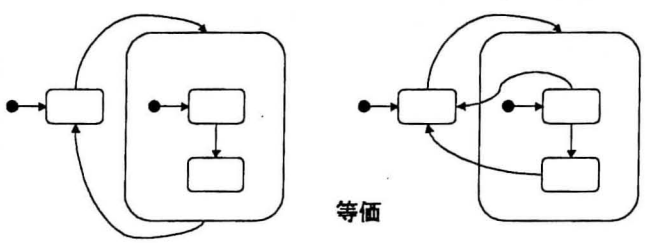
- イベントが遷移を引き起こす
- イベントは引数を取ることが出来る
- アクションは、遷移が引き起こされた時に実行される処理である

状態

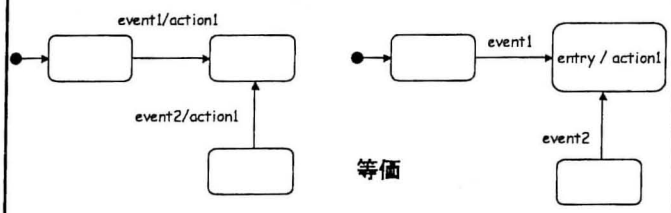


階層化された状態

階層化された状態



アクション



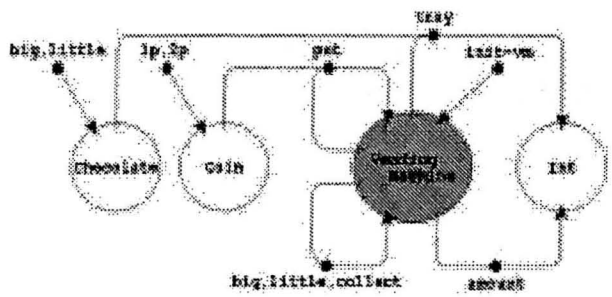
問題点

- どうやって状態を特定するのか？
- 作成したstatechartは正しいのか？
  - 要求を満たしているか？
  - 論理的矛盾はないか？

振舞仕様

- 状態機械の振舞を、観測と操作という2種類の演算で記述
- 状態機械の状態は、操作演算によって変化する
- 状態機械の状態は、観測演算によって観測できる
- 状態を陽に特定しない

例：自動販売機



statechartと振舞仕様の対応

statechart	振舞仕様
状態	なし
イベント	操作演算
変数	観測演算
アクション	観測演算

前提条件

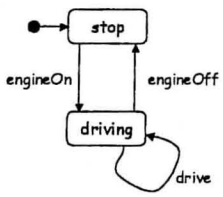
- statechartのイベントに付加できる条件は、変数参照と整数値を使ったbool型の式
- statechartのアクションは、変数への値の代入のみ
- statechartは、必ず初期状態を持つ
- 並行性は入れない

例題：自動車の自動巡航システム

- CCS(Cruised Control System)
- 車をある一定のスピードに保つようにコントロールするシステム
- 走行中にCCSを作動させると、その時点の時速が記録され、その速度が保たれる
- ブレーキを踏むかアクセルを踏むか、自動巡航をoffにすると、自動巡航を中断する
- 自動巡航中断中にresumeボタンが押されると、記録されている速度にもどして自動巡航に入る

単純な車（car0）

前提
・慣性は考えない
・エンジンを切ったら車は瞬間的に止まる



car0に対する振舞仕様

属性が特定されていない段階では、
観測として状態名を考える

```
mod! STATUS {
  [ Status ]
  ops stop driving -> Status
}
```

car0に対する振舞仕様

```
mod* CAR0 {
  *[ Car0 ]*
  op init-car0 : -> Car0
  bop engineOn : Car0 -> Car0
  bop engineOff : Car0 -> Car0
  bop status : Car0 -> Status

  var C : Car0
  eq status(init-car0) = stop .
  ceq status(engineOn(C)) = driving    if status(C) == stop .
  ceq status(engineOff(C)) = status(C) if status(C) == stop .
  ceq status(engineOn(C)) = driving    if status(C) == driving .
  ceq status(engineOff(C)) = status(C) if status(C) == driving .
}
```

状態遷移表

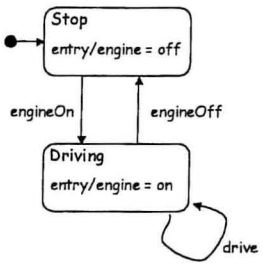
	stop	driving
engineOn	-> driving	
engineOff		-> stop

方針

- 詳細化によって、単純な車(car0)からCCS付の車を設計する
- statechartの各状態は、クラスで定義された属性の値の組によって特徴づけられると考える
- 対応する振舞仕様が詳細化されている => statechart図も詳細化になっている
- 詳細化をしながら、必要な属性と制約を明らかにしていく

属性の詳細化

- エンジンの状態がcar0の状態を分割している
- このように、属性を特定してstatechartを詳細化することを属性の詳細化と呼ぶことにする



car1に対する振舞仕様

```
mod* CAR1 {
  *[ Car1 ]*
  op init-car1 : -> Car1
  bop engineOn : Car1 -> Car1
  bop engineOff : Car1 -> Car1
  bop engine : Car1 -> Bool

  var C : Car1
  eq engine(init-car1) = false .
  ceq engine(engineOn(C)) = true      if engine(C) == false .
  ceq engine(engineOff(C)) = false    if engine(C) == false .
  ceq engine(engineOn(C)) = engine(C) if engine(C) == true .
  ceq engine(engineOff(C)) = false    if engine(C) == true .
}
```

属性の詳細化

- 観測演算を以下のように定義する
  engine : Car1 -> Bool
- 属性は以下のようなマップを用いて詳細化される

  Status -> Bool
  stop -> false
  drive -> true

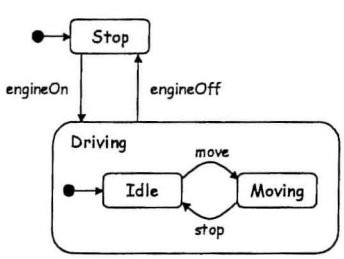
- 演算のマップは自明

詳細化検証

```
open CAR1 .
  red engine(init-car1) == false .
  op c : -> Car1 .
  eq engine(c) = false .
  red engine(engineOn(c)) == true .
close .
open CAR1 .
  op c : -> Car1 .
  eq engine(c) = false .
  red engine(engineOff(c)) == false .
close .
...
```

状態の詳細化

- Driveの状態を詳細化している

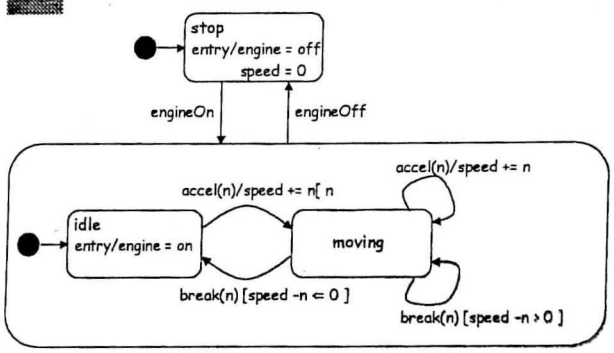


状態の詳細化

```
mod! STATUS {
  [ Status ]
  ops stop idle moving : -> Status
}
```

- 属性のマップ
  Status -> Status
  driving -> idle or moving
  stop -> stop
- イベントのマップ
  engineOn -> engineOn
  engineOff -> engineOff
  driving -> move if statusがidle
  stop elseif statusがmoving

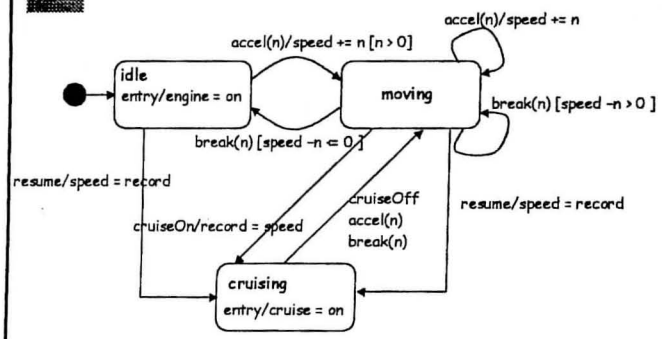
car3



状態と属性

- stop
  - engine = off and speed = 0
- idle
  - engine = on and speed = 0
- moving
  - engine = on and speed > 0

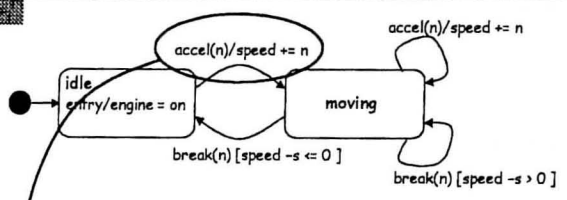
CCS付の自動車car4



まとめ

- 振舞仕様はいつ使うのか
  - いつでも
  - 使っても良いし使わなくてもよい
- 振舞仕様はどの部分に使うのか
  - 切り出し可能な部分であればどこでも
- 一番大きな効用は何か
  - statechartは強力だがそれ故に迷う
  - 迷ったときに形式的な世界に行ってみると考えが整理されるかもしれない
- 部分的に形式仕様を使う場合には、全体としての完全さは期待できないが、品質のアップには貢献するのでは？

今後の課題



nが0の時には、speed=0にもかかわらず
movingに遷移してしまって、movingの条件である
engine = on and speed > 0と矛盾する

Call for Participation

ISFST 2001

International Symposium on Future Software Technology 2001

Nov, 5-8, 2001
ZhengZhou (Capital of Henan Province), China

Co-Sponsored by
SEA, UNU/IIST, ZhengZhou Univ, and Henan Univ

This is the 6-th symposium dedicated to the future software technology, which has been held in China since 1996. It is organized by SEA (Software Engineers Association of Japan), UNU/IIST (International Institute of Software Technology of United Nations University), ZhengZhou University, and Henan University. The symposium will bring together researchers, practitioners, and educators in the leading-edge software technologies.

The theme of the ISFST2001 is: "First Step of Software Technology in 21st Century".

The unique feature of the ISFST symposium is that the program includes informal but intensive working-group style discussion sessions besides regular paper presentations. In the case of ISFST2001, the afternoon of the second day will be assigned for working-group discussions on four topics. Also, there is scheduled one-day excursion to Kaifeng city on the fourth day to provide an opportunity for wrapping-up discussions in the open-air atmosophre.

From about 90 submissions, 60 papers were selected for technical presentations and also as materials for working group discussions (see program framework below). The symposium features three keynote speakers: Professor Gerhard Fischer (University of Colorado, USA), Professor Kyochul Kang (Postech, Korea), and one other invited speaker from China).

1. Conference Venue: Science Hall of Zhengzhou Univercity

2. Lodging: Songshan Hotel (within walking distance from Zhengzhou University)
156 Yihe Road, Zhengzhou, Henan 450006, China
Tel: +86-371-797-6677, Fax: +86-371-797-4076

3. Program Framework:

Day 1 (Nov 5, Mon)	9:00 - 9:30	Opening
	9:30 - 10:30	Keynote 1: Prof. Gerhard Fischer (Univ of Colorado, USA)
	10:30 - 11:00	Break
	11:00 - 12:30	Technical Paper Sessions: 1A - Formal Method], 1B - OO (1), 1C - Network
	12:30 - 14:00	Lunch Break
	14:00 - 16:00	Technical Paper Sessions: 2A - Algorithm, 2B - OO (2), 2C - WIS
	16:00 - 16:30	Break
	16:30 - 18:00	Technical Paper Sessions: 3A - Language, 3B - Process, 3Ci Req&Design
	18:00 - 18:30	Break
	18:00	Reception
Day 2 (Nov 6, Tue)		
	9:00 - 10:00	Keynote 2: Prof. Kyochul Kang (Postech, Korea)
	10:00 - 10:30	Break
	10:30 - 12:30	Technical Paper Sessions: 4A - Testing, 4B - Implementation, 4C: DB
	12:30 - 14:00	Lunch Break
	14:00 - 18:00	Working Group: A - CHI, B - DB/Method, C - Mobile, D - Network
Day 3 (Nov 7, Wed)		
	9:00 - 10:00	Kenote 2: TBD (China)
	10:00 - 10:30	Break
	10:30 - 12:30	Closing Panel: "SPI/SPA Activities in Asia"
Day 4 (Nov 8, Thu)	Allday	1-day Excursion to Kaifeng (Open-air discussion)
Day 5 (Nov 9, Fri)		Leave Zhengzhou or 2-days **Optional Tour** to Luoyang city

Detailed Advance Program will be found at : http://www.media.osaka-cu.ac.jp/ISFST2001

****** **Registration Form is on the Back Page** ******

ISFST 2001 2000 Conference Registration Form

Please complete and return this form by e-mail or Fax until Oct 26 (Fri) to ISFST 2001 Local Arrangement Chair:
Prof Ming Fan
Department of Computer Science
Zhengzhou University
Zhengzhou City, Henan Province, 450052, China
Tel & Fax: +86-371-776-1542, E-Mail: mfan_cs@sina.com

Title: Prof. ___ Dr. ___ Mr. ___ Mrs. ___ Ms. ___

Name: (Surname)_____ (First name)_____ Sex: _____

Nationality: _____ Passport No.: _____

Mailing Address: _____

Tel: _____ Fax: _____ E-Mail: _____

Acompanied Person (if any):

Name: (Surname)_____ (First name)_____ Sex: _____

Nationality: _____ Passport No.: _____

Registration Fee:

Regular: US$ 800.00
Student: US$ 400.00 (2 persons share 1 hotel room)
Accompanied Person: US$ 400.00 (shares same room with participant)
(The registration fee includes a copy of the proceedings, 5 nights - from Nov 4 to 8 of accommodation at SongShan Hotel, banquet, and after-symposium tour to Kaifeng on Nov 8)

Optional Tour: We are planning to do optional sight-seeing tour to Luoyang city after the symposium (on Nov 9-10). Additional cost will be US$ 200.
Will you join this optional tour or not? [ ] Yes i [ ] No

Method of Payment: Please pay in Cash or Traveller's Check on site.

Access to Zhengzhou: Zhengzhou is reachable by air from major cities in China. There are several number of daily domestic flights from Beijing or Shanghai. Please inform your arrival schedule to our local arrangement chair (Prof. Ming Fan, his contact address is above) in advance. ISFST2001 staff will see you at Zhengzhou airport.

VISA: If you need to have a formal invitation letter to get VISA to China, please contact to Chinese Program Co-Chair: Dr. Beijun Shen of ASTI Shanghai via e-mail. Her e-mail address is:

beijun.shen@netease.com

Zhengzhou: Located at the middle and lower reaches of the Yellow River, Zhengzhou is the capital of Henan Province. Beijing-Guangzhou and Lanzhou-Lianyungang railways hub here. it owns the biggest railway marshalling yard in the Asia and the longest bulk cargo yard transfer station in China. The newly-built Zhengzhou Xinzheng Airport has, opened flights to over 30 big and medium- sized cities in the country including Hong Kong, and tourist chartered airplane flights to Japan, Singapore. With 371 whole-sale markets including state grain wholesale market, Zhengzhou agricultural machinery exchange market, Zhengzhou is a famous commercial and trade city. The city of Zhengzhou came into shape more than 3,500 years ago. Its tourist attractions include the Songshan Shaolin Temple, Yellow River cultural group with Yellow River Scenic Spot, the Yellow River Grand View and Dahecun Village Remains as main, the root-seeking and respects-paying tour with Xuanyuan's Native Place, Yellow Emperor's Palace, Yellow Emperor and Emperor Yan.

ソフトウェア技術者協会
〒160-0004　東京都 新宿区 四谷 3 - 12 丸正ビル 5F
Tel: 03 - 3356 - 1077　Fax: 03 - 3356 - 1072
E-mail: sea@sea.or.jp
URL: http://www.iijnet.or.jp/sea