修 士 論 文

CafeOBJ による B-抽象機械モデルの検証法に関する研究

指導教官　二木厚吉 教授

北陸先端科学技術大学院大学

情報科学研究科情報システム学専攻

梅原 伸年

1998 年 2 月 13 日

要旨

近年ソフトウェアは大規模化し、高信頼性を持つことが求められている。この信頼性に対する解決方法として形式手法がある。しかしソフトウェアシステムの開発において形式手法の導入は一般に進んでいない。そのなかで B-Technology(以降 B とする) を用いた開発が評価を得ている。その大きな理由としてオブジェクトを基にした開発アプローチのサポートがあげられる。ここでいうオブジェクトは状態と操作から構成されるもので、B はこのオブジェクトを抽象機械として仕様の構成単位としている。また B-Method は抽象機械の無矛盾性の証明法を詳細化に対しても提供しており、その証明は B-Toolkit により自動的に行うことができる。B のようなモデル指向の形式仕様言語はいくつかあり、代数仕様言語との比較研究が行われている。しかし B の抽象機械記法にに対する比較は少なく、仕様の詳細化技法に着目した物はない。そこで本研究では最初にモデル指向の B 抽象機械表現が 代数指向の CafeOBJ によりどのように表現することが出来るか対応を考察しテーブルを作成した。特に内部状態への操作に着目し仕様作成の指針を与えた。次に B の開発手法である仕様の無矛盾性と詳細化に関する無矛盾性の証明が代数仕様の CafeOBJ でどのように表現されるか考察を行い、B がもつ開発手法が同様に CafeOBJ で実践できることを示した。また中規模の有名問題であるリフトの仕様を事例として利用し、検証に関する考察を行い、隠蔽代数を利用することで検証が容易に行えることを示した。

本研究では B 開発アプローチの特に上流工程である要求仕様を形式的に記述する段階、形式仕様からさらに詳細化された仕様を作成する段階を対象とする。またその過程においては、CafeOBJ の実行可能な特徴を利用し、仕様に求められる性質の保存に関する証明を行う。

目 次

第 1 章

はじめに

1.1 はじめに

近年コンピュータの進化によりソフトウェアの応用範囲が広がっている。ソフトウェアは大規模化し、高信頼性が求められるようになった。そのなかで多人数での開発には、要求に対する理解に個人差がでないように何らかの対策が必要である。理解の個人差が発生することで、システムが満たすべき制約が忠実に守られないからである。これは航空管制システムなど強い信頼性が求められるソフトウェアとって重要な問題である。この信頼性に対する解決方法として形式手法がある。しかしソフトウェアシステムの開発において形式手法の導入は一般に進んでいない。そのなかで B-Technology[10][6](以降 B とする) を用いた開発が評価を得ている。その大きな理由として現在一般的となったオブジェクトを基にした開発アプローチのサポートがあげられる。一般にシステムの仕様を作成するとき、その振る舞いにより状態を規定することで問題を表現できる場合は多い。B は仕様作成において状態と操作からなる抽象機械を構成単位としており、これは様々な問題の仕様作成に対して B の利用が有効であるともいえる。そしてオブジェクト単位で作成された抽象機械は、作成者本人以外の開発者が理解しやすいという側面もある。本研究の目的は、評価を得ている B の開発手法を CafeOBJ[11][1] 上で同様に行い、同じ開発アプローチが代数仕様言語で実践可能なことを示す。そして振る舞いでとらえたシステムの形式仕様を作成する時の指針を得ることである。また B の開発支援で提供される仕様の無矛盾性の証明は、型に対する確認が多くを占めるが、ソート上の演算として操作を記述できる代数仕様言語 CafeOBJ を用いることで B の開発アプローチが簡単に行えることを示し、代数仕様言語 CafeOBJ の有効性を示す。本研究では B 開発アプローチの特に上流工程である要求仕様を形式的に記述する段階、形式仕様からさらに詳細化された仕様を作成する段階を対象とする。またその過程においては、CafeOBJ の実行可能な特徴を利用し、仕様に求められる性質の保存に関する検証を行う。本研究で CafeOBJ を利用した理由の１つにこの実行可能な機能があげられる。CafeOBJ の実行可能な性質を利用することで検証を行った研究はいくつかありその有効性は実証されている [21][15]。

1.2 本論文の構成

本論文の構成は以下のようになっている。

第２章では、本研究の基盤となるモデル指向の形式仕様言語 B と代数を基礎とする形式仕様言語 CafeOBJ についての解説およびいくつかの定義を行なう。

第３章は、B 言語の抽象機械で表現された仕様が代数仕様言語 CafeOBJ 上でどのように表現できるかについて考察を行い、対応した仕様を得るための指針を与える。また B 言語の設計手法が CafeOBJ 上で実践可能なことを示す。

第 4 章では、CafeOBJ のもつ理論体系を用いることで検証の容易さについて有効な仕様表現が得られる事例を示す。

第 5 章を結論および今後の課題の章とし、最後に４章で利用した事例の B 言語による仕様と CafeOBJ による仕様を付録として添付する。

第 2 章

形式仕様言語

本章では形式仕様言語としてモデル指向の形式仕様言語 B と性質指向の形式仕様言語 CafeOBJ が形式手法としてどのように位置づけされるかを紹介する。またその基礎となる理論について簡単に説明を行う。

2.1 形式手法

近年コンピュータの発達によりさまざまな分野でコンピュータシステムが活用されるようになった。処理能力の向上は複雑なシステムのコンピュータ化につながり、現在ではシステム停止が多大な損害を生む現場においてもコンピュウターが活用されている。人命に関わる飛行機管制システムなどはその顕著な例といえる。そのシステムにおいて信頼性に対する要求は多大なもので、その設計は容易でない。そこで信頼性の高いシステムを作成するためにさまざまな研究が行われており、形式手法はシステム分析から設計、実装までを統合して管理しようとするアプローチである。ソフトウェア開発において実現したい要求を記述したり、存在する問題や制約を表現する時自然語や図を利用することは容易に理解ができる利点がある。しかし自然語や図だけですべてを仕様を作成することは人による理解の差から、要求事項に曖昧さが残され時に誤解を生じる問題がある。そこで問題対象を個人差なく統一的に記述を行うために数学理論に基づく記法を仕様言語として利用し問題を記述することが必要となる。形式仕様言語とはこの数学的体系に基づいた記述法で、論理や代数を基礎としたものがある。

2.2 抽象機械仕様言語 B

B 言語は Z 言語がもつ記法を拡張した記法をもち、抽象機械を仕様の記述単位とし B-Method による設計支援手法と開発支援環境を提供する B-Toolkit をもつ。

Z 言語について簡潔に述べたのち、B 言語の特徴を紹介する。

2.2.1 Z

Z [2] は Oxford 大学で開発されたモデル指向の形式仕様言語である。Z 記法は現在 ISO で標準化作業が行われており、広く知られた形式言語といえる。形式的表現として型付けされた集合論と一階述語論理を基礎としている。対象世界を集合、関係、関数、列などをもちいて記述する。Z の特徴として他にスキーマという図表記法がある。スキーマにより、仕様は名前のついた単位で記述でき仕様の他の場所から名前を利用して参照できるものとなり、構造化された仕様記述可能となっている。また周辺研究も盛んに行われており Type Checker などのツールやオブジェクト指向をサポートする Z の拡張言語 Object-Z や Z++ などの研究も行われている。

2.2.2 B-Technology

B-Techology [10] [6] とは AMN、B-METHOD、B-Toolkit を総称するもので、仕様の分析、設計、実装までをサポートする統合開発環境である。B-Method 及び Abstract Machine Notation は Oxford 大学の Programming Research Group において 1985 から研究が始められた。今日では市販されている形式言語であり開発が続いている。産業プロジェクトで実際に利用されており、形式手法の必要性が言われながらも普及しない現状において特に注目されている。

2.2.3 Abstract Machine Notation

数多く形式手法が存在する中で B の開発アプローチの特徴は、抽象機械モデルでソフトウェアシステムを捉えることである。B 仕様に用いられる記法は、 Z の記法に類似するものであるがスキーマは用いず抽象機械をその記述単位とする。抽象機械による仕様は、状態と状態に対する操作の組を単位として表現される。ソフトウェアシステムを分析することは、多くの場合静的な側面と動的な側面を明らかにしていくことである。ここで静的な側面とは状態の定義であり、動的な側面とは操作の定義である。また構造化された仕様記述を行うための輸入関係の定義が抽象機械には含まれる。大規模なシステムは、この輸入参照関係を利用し抽象機械を組み合わせることで作成できる。
輸入関係を持たない簡単な抽象機械は次のように記述される。

```
抽象機械(Abstract Machine)

MACHINE N(p)     N:machine name, p:paramers
CONSTRAINTS C    C:述語列（パラメタに対する制約）
SETS St          St:sets（BOOL など組み込みでも提供）
CONSTATNTS k     k:定数列
PROPERTIES B     B:述語列（定数に対する制約）
VARIABLES v      v:状態変数列
INVARIANT I      I:述語列（抽象機械が扱える変数に対する制約）
INITIALISATION T  T:初期化式（I を満たすもの）
OPERATIONS
 y <- op(x) =
     PRE P THEN S END
END
```

2.2.4 B-METHOD

B-METHOD は、抽象機械を単位とする仕様を記述、設計、実装するための数学に基づいた手法の総称である。B-Method は仕様の無矛盾性を調べる方法を規定し、設計と実装の正しさを調べる方法を提供する。また構造化された仕様を記述する方法を規定し、仕様やソフトウェアモジュールの再利用可能としたものである。B-Method による無矛盾性の証明などは B-Toolkit により支援される。その詳細については次章で説明する。

2.2.5 B-Toolkit

B-Toolkit [1] は B-METHOD のあらゆる側面に対する支援環境ソフトウェアの総称である。主なものに抽象機械記法のための文法チェッカーと型チェッカー、不変条件の保存と設計の正しさのための条件を自動生成する Proof Obligation Generator、自動証明システムと対話的証明支援システムがある。その他に、仕様の動作を与えるアニメーション機能や C 言語による実装を与えるトランスレータをもつ。また仕様のドキュメントの出力が行える。

[1] The B Toolkit is a trademark of B-Core (UK) Ltd.

2.3 代数仕様言語 CafeOBJ

CafeOBJ [1] [11] はOBJ の流れをくむ実行可能な代数仕様言語で、順序ソート代数、書換え論理、隠蔽代数 [7] を基礎としたマルチパラダイムな形式仕様言語である。
CafeOBJ はモジュールを仕様の構成単位とする。モジュールはソートの集合とソート上の関数を記述する指標 (signature) と関数の意味を記述する公理 (axiom) からなる。対象世界が満たすべき性質を公理をもちいて記述するため性質指向な言語といえる。CafeOBJ では記法としてモジュールの役割を明示するために 2 つの表現 **module!** と **module*** が提供されている。前者はそれ自身で唯一のモデルを表現するのに対し、後者はモデルのクラスを表現する。CafeOBJ では再利用可能なパラメータを用いたモジュール表現や、隠蔽代数を用いた抽象度の高いモジュール表現ができる。CafeOBJ はモジュールの輸入機能で、より大規模なシステムの仕様記述が行え、モジュール単位による仕様の検証をサポートする。

2.3.1 代数 (Algebra)

計算機科学において問題を記述する際、データとそれに対する演算を組として扱うことは一般的である。この組はデータ型と言われ、近年一般的となったオブジェクト指向でのオブジェクトの基本的記述でもある。基本的な概念として代数仕様における代数とはこのデータ型が対応し、代数モデルとはその抽象化である抽象データ型の数学モデルが対応する。すなわちシステムの構造、機能、性質を代数モデルに基づいて記述した形式仕様が代数仕様である [23]。代数仕様言語に関する研究は幾つか行われており OBJ、CSP、プロセス代数などが挙げられる。それらは独自の拡張を行っておりその基礎概念は異なっている。以降行う代数仕様言語の説明は CafeOBJ が拠り所とする数学的形式さを扱い、その説明のための記法は CafeOBJ に従うとする。
CafeOBJ は多ソート代数を基礎に順序ソート代数、書き換え論理、隠蔽代数への拡張を行った代数仕様言語である。
従ってその代数は

- 台集合と呼ばれるその代数を構成する対象の集合
  CafeOBJ は多ソート代数であることから、台集合は種類分けされた複数から構成され、その種類の名前としてソート名が与えられる。また順序ソート代数への拡張でサブソート関係をもつ。
- その集合上に定義された幾つかの関数
- それらの関数が満たすべき制約を表現した等式 (equation)、書き換え規則 (transition)
  の３つを与えることで一般に定義される。

代数によって仕様は作成されるが抽象化が容易であるため、代数仕様は特定の共通した性質を記述することができ、また詳細なデータ構造の記述も行える。共通した性質を容易に扱えることが仕様を設計する際に重要な性質となることを後に述べる。

2.3.2 書き換え論理

システムの仕様を与える際に、システムの動的側面に着目することで性質をうまく記述できることは多い。ある入力に対してどのようにシステムがもつ状態が変化するかを記述するとき書換え論理を利用することが出来る。

CafeOBJ は書き換え論理への拡張を行うことで、OBJ 等が行っていた等式を左辺から右辺への書き換えとみなし状態変化を記述する手法をより明示的に扱うことができる。

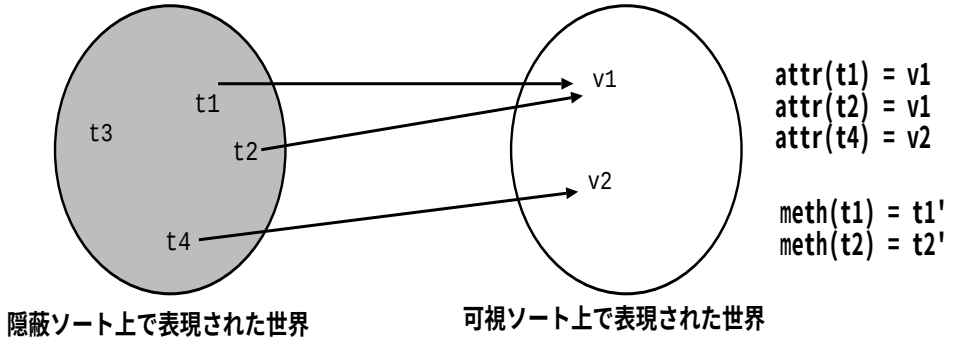
等式論理では完備な項書換えシステムにより証明を行うことを意識していた。書換え規則で与えられた仕様は、等式で必要であった完備であることの仮定を必要とせず、非決定な振る舞いが記述できる。

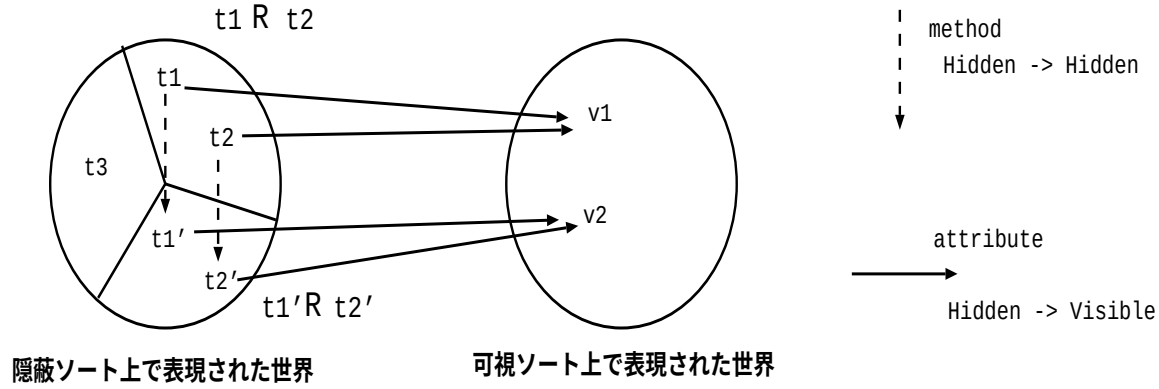
CafeOBJ のもつ実行可能環境は項書換えシステムであり、状態変化は状態を指す項から項への書換えで表現される。仕様を実行することで状態遷移可能であるかの調査や、現在状態からある回数後にどの状態であるかの調査を行うことができる。

2.3.3 隠蔽代数 (Hidden Algebra)

大規模システムは満たすべき性質は複雑なものである。いきなり全体を記述することは容易でないので、その設計は段階的に行われる。途中段階の仕様では、実装に対する自由度を保持することが望ましいことから抽象的に与えられる。CafeOBJ がもつ隠蔽代数の概念は詳細化されていない仕様を記述できる有効な手段を提供する。これは従来等式論理を用い始代数に基づく項の上で仕様を記述していたのに対し、状態を表現する具体的な項表現を考えずに仕様を記述するものである。

隠蔽代数は近年 Goguen らが提唱している振る舞いに基づく代数である [7] [9]。隠蔽代数においてソートは２つに明確に分けられる。観測可能なデータを記述した可視ソートとブラックボックス化されたデータを記述する隠蔽ソートである。ここで可視ソートは従来の代数と同様である。隠蔽ソートは振舞いレベルで抽象化された状態を表現する。ある状態はどのような構成か、どのような履歴を持つかなどの情報は隠蔽され、外からの観測の結果により初めて識別される。





h1 R h2 = attr(h1) == attr(h2)

隠蔽指標

ソート S は、可視ソート V(Visible) と隠蔽ソート H(Hidden) を含み

$S = (V \bigcup H)$ の関係を持つとする。Σを

$\rho \in \Sigma_{w,s}$ with $w \in V^*$, $s \in V$と、

$\rho \in \Sigma_{w,s}$ w の中に高々一個の H の要素を含むρ からなるものとする。

このとき**隠蔽指標 (Hidden Signature)** は$< H, \Sigma >$で与えられる。

$w \in S^*$が隠蔽ソートを含み、$s \in H$ のとき $\rho \in \Sigma_{w,s}$ を **method** と呼ぶ。

また、$w \in S^*$が隠蔽ソートを含み、$s \in V$ のとき $\rho \in \Sigma_{w,s}$ を **attribute** と呼ぶ。

$w \in V^*$, $s \in H$ の時 $\rho \in \Sigma_{w,s}$を**隠蔽定数 (hidden constant)** と呼ぶ。

隠蔽仕様

$< H, \Sigma >$を隠蔽指標、 E を Σ-等式の集合とした時、3 項組 $< H, \Sigma, E >$ を **隠蔽仕様 (振る舞い仕様)** と呼ぶ。

ここでΣ-等式は可視ソート上の等式を含まない。

隠蔽代数を用いることで大規模システムの仕様記述に有効な段階的詳細化技法が利用できることを後に示す。

第 3 章

CafeOBJ による抽象機械の記述

本章では前章で紹介した B 言語の抽象機械表現を CafeOBJ で表現する。B 言語は実際の開発現場で利用されるなど形式手法の導入が進まない中で注目されている。その主な理由として段階的詳細化をサポートする充実した開発支援環境と開発者の直観的理解を助ける抽象機械という記述単位がある。
近年 B 言語の抽象機械と同等の記述を行いその無矛盾性の証明や仕様詳細化について研究を行った例 [3] [4] は、型付けされた高階論理をベースとした PVS や HOL を用いたもののみである。また仕様言語としての研究も B 言語の記法の元になる Z と他の形式言語の比較はいくつか行われているが [22] [19]、B 言語と代数仕様言語と比較したものは少ない。そこで B 言語がもつその設計段階に注目し、そのアプローチを実践することで代数仕様言語 CafeOBJ の有効性を示す。
ここでの設計段階とは顧客との間でソフトウェアシステムに求められている課題や性質を理解する要求分析とプログラムへの翻訳作業との間にある段階である。すなわち与えられた要求を形式的に記述し、実装の自由度を保ちながら洗練して行く段階を対象とする。

3.1 抽象機械の各節の CafeOBJ 記述

CafeOBJ は複数の論理を基礎とすることから、B-抽象機械表現の中心である状態も複数の表現法が考えられる。しかし B はそのツール群において証明支援系を持つことから仕様の実行性を考慮した記述法をとる。Baumeister による implicit state approach [5] はモデル指向の仕様を代数仕様で表現した。implicit state approach では指標が状態を表現し、状態そのものは状態変化をおこす操作に対して暗黙にあつかわれる。しかしこの手法は CafeOBJ 上でそのまま実行できないので採用しなかった。そこで本論文ではまず仕様を直接実行可能な Explicit State Approach をもとに記述を行った。この手法は状態を明示的に扱うもので、状態変化を起こす操作の引数に状態を表現する項を取るものである。状態空間はソートで表現される。

B での仕様を CafeOBJ を用いた仕様で表現する主な指針を (表 3.1) に示す.

	B-抽象機械	CafeOBJ
仕様記述単位	抽象機械	モジュール
パラメタ機能	集合パラメタ，要素パラメタ パラメタの制約	パラメタ パラメタモジュールの等式
構造化	輸入参照関係	輸入
宣言	集合 定数 集合，定数の性質	ソート ソート上の項 等式
状態の表現	変数 不変条件	公理中の変数，状態空間はソート 条件つき等式，ソート表現
初期値	初期化	等式
状態変化	操作 事前条件 操作の本体	ソート上の関数 条件付きの等式 等式，書換え規則

図 3.1: B-抽象機械の CafeOBJ 表現

- 仕様記述単位 (MACHINE)

  抽象機械はモジュールに対応させる。

  代数仕様はデータ型、抽象データ型の記述が自然に行える。従って B がもつオブジェクトの単位で仕様を提供するアプローチは代数仕様でも実現できる。

  しかし仕様全体としては CafeOBJ のモジュールが多くなることがある。これは B が抽象機械記法として集合、列、2 項関係などを豊富に提供しているのに対し、CafeOBJ は表現の自由度を損なわないように、記法においても特に推奨する書き方を規定していないためである。

- パラメタ機能 (parameta, CONSTRAINTS)

  パラメタ機能はパラメタモジュール、もしくはモジュールの輸入により表現する。

  注意しなければならない事は、B-抽象機械でいうパラメタという言葉の意味と CafeOBJ で標準的に利用するパラメタの意味が異なる事である。B のパラメタはプログラミング言語でいう一般的なパラメタであるが、CafeOBJ では再利用可能なセオリーを意味する。しかし B-抽象機械で CONSTRAINTS 節により定義されるにおけるパラメタの制約 (性質) 条件は、CafeOBJ での公理で与えることができ

```
MACHINE LIFT (LFT, topfloor, bottomfloor)
CONSTRAINTS topfloor > bottomfloor
INCLUDE DIR
== ========================
module LIFT {
        protecting(LFT)
        protecting (NAT  { sort Nat → Floor,
                           op zero → bottomfloor,
                           op ten → topfloor})
        using(DOR)
```

ここで bottomfloor というパラメタを自然数を定義した。
モジュールの要素を名前替えする事でバインドしている。

図 3.2: B-抽象機械パラメタの CafeOBJ 表現

る。従って B のパラメタの機能は、CafeOBJ ではパラメタモジュールで制約を表現し、名前替えを行うことで同様の表現を得られる。これはパラメタとして利用したい値を定義したモジュールを輸入すること同じである。(図 3.2)

B-抽象機械では普通のプログラミング言語と同様にパラメタとして定数を扱える。CafeOBJ では値を渡すという概念はない。従って明示的な値は輸入を利用しで被輸入モジュール内で定義される定数を採用することで代用する。このことは CafeOBJ の機能が低いという意味でない。抽象機械で定義したパラメタは、B-Toolkit により提供される実行機能において対話的に与えることが出来る。抽象機械仕様をこの機能を考慮して記述するできることは、B 言語の特異な性質といえる。CafeOBJ の仕様の実行において与えられるものは、既に定義した仕様により到達可能な項に限られる。ここでの実行は B、CafeOBJ とも純粋に定義された仕様をシミュレートする意味で、証明を行うという意味ではない。B-抽象機械がパラメタとして集合をもつ際は、B ではその実行時に集合の要素を文字列により与えることができる。

- 構造化 (INCLUDES, USES, SEE)
  3.3 節で詳しく述べる。

- 宣言 (SET，CONSTATS, PROPERTIES)
  集合はソートに、集合の要素はソート上の項に対応させる。

すなわち集合名がソートに対応し、集合の要素はソート上の定数に対応する。再帰的に構成を持たないような要素の集合は、アリティが空であるような関数宣言のみからなる。抽象機械での要素を持たない抽象的な集合の宣言はソートの宣言のみ。

定数はソートの一個の要素 (項) に対応させる。
B-抽象機械は CONSTANTS 節と PROPERTIES 節の宣言を組み合わせて関数を定義できる。このように宣言された定数は CafeOBJ で関数として宣言される。この B 言語の定数関数は操作のように状態変化を起こすものではない。(図 3.3)

集合と定数の論理的な性質は等式で与える。
多くの場合与えられる性質は型の情報であるので、ソート表現を行うことで十分である。

```
SETS DIR = {up, dn}
CONSTANTS opp
PROPERTIES opp ∈ DIR → DIR ⋀
                  opp(up) = dn ⋀
                  opp(dn) = up
== ========================
[ Dir ]
op up : → Dir
op dn : → Dir
op opp : Dir → Dir
eq opp(up) = dn .
eq opp(dn) = up .
```

2 要素の集合をソート Dir で表現し、要素はそれぞれアイティをもたない関数で表現
この対応で B 言語の定数関数を関数として定義している

図 3.3: B-抽象機械の定数、集合の CafeOBJ 表現

- 状態の表現 (VARIAVLES, INVARIANT)
  状態をあらわす変数はそれぞれ公理に記述されるソート付けされた変数に対応させる。
  現在状態を指す変数はソート上のいずれかの項で表現され、状態空間はそのソートである。到達可能な状態をソート上で構成可能な項として表現する。条件つきの等式 (書換え) で構成可能な項を制限する。

  状態の不変的な性質は等式で表現される。(図 3.4)

定数の時と同様に型に関する情報はソート表現を行うことが対応する。

VARIABLES cstate
INVARIANT cstate ∈ CSTATE
== ========================
[Cstate]
op coin-present : → Cstate
op coin-absent : → Cstate
op give-change : Cstate → Cstate
var C-CSTATE : Cstate
eq give-change(CCSTATE) = coin-present .

状態空間がソート Cstate である。
現在状態は変数 C-CSTATE がとる値。
不変条件は状態が集合 CSTATE 上に存在することで関数 give-change を与えてもソートは保存される
説明の為に関数の宣言をここで紹介している

図 3.4: B-抽象機械の変数、不変条件の CafeOBJ 表現

- 状態変化 (OPERATIONS)

  操作はソート上の関数に対応させ、公理における等式で意味を与える。事前条件付き操作は、条件付きの関数で表現する。

  状態遷移の事前状態を等式を左辺、事後状態を等式の右辺で表現する。(図 3.5) CafeOBJ がもつ項書換えエンジンでは等式を書換え規則とみなし状態変化をシミュレートすることができる。CafeOBJ はマルチパラダイムな言語で多ソート代数から書換え論理への拡張を行っている。従って仕様に書換え規則を記述することで、より明示的に状態遷移で表現されるようなシステムを表現できる。また順序ソート代数への拡張も行っている。可能であれば事前条件を満たすようなサブソートを導入することで操作表現を簡潔にする。

  以降の説明において B-抽象機械上の状態変化を起こす操作を' 操作' とし、CafeOBJ 上の対応する操作を表現するのに' 関数' をつかう。

- 初期化 (INITIALISATION)

  初期化は公理において等式であらわす。(図 3.6)

  ここで初期化は初期状態を定義する物で、システムに初期状態が存在することを宣言するだけである。初期状態に戻る操作を宣言するのではない。

```
OPERATIONS
give_change(cc) =
                PRE cc ∈ COINS
                THEN cstate := coin_absent
                END;
== ========================
[ CState Coins CSlot ]
op _|_ : Cstate Coins → CSlot
op give-change : CSlot Coins → CSlot
var CL : CSlot
vars CO1,: Coins
eq give-change(CL,CO1) = (coin-absent | CO1) .
```

状態空間をあらわす CState の中身が関数で変化している。
CState の状態をあらわす変数は CSlot の状態に内包されている。
事前条件に対応する部分は変数 CO1 を利用し引数となりうる型を制限している。

図 3.5: B-抽象機械の操作の CafeOBJ 表現

3.2 複数の状態を持つシステムの表現に関する考察

本節では今回の研究の進める過程で判明した抽象機械を代数仕様で扱うための指針について解説を行う。代数仕様はシステムの状態変化を関数を利用して表現する。すなわち状態変化を起こす関数の引数はある抽象レベルの状態を表現したものである。従って状態変化を引き起こす関数の引数を、この状態をあらわす１つの変数に制限することが自然な仕様記述につながる。実際、複数の状態を引数にとる関数がどのようなシステムの状態変化を表現しているのかを理解することは容易でない。

```
INITIALISATION
cstate := coin_absent
current_coin := 10
== ========================
[ CState Coins CSlot ]
op _|_ : Cstate Coins → CSlot
eq empty-cslot = coin-absent | tenc .
```

初期化により初期状態を coin-absent,tenc の二つの部分状態からなる全体状態 CSlot を規定している。

図 3.6: B-抽象機械の初期化の CafeOBJ 表現

MACHINE Person
VARIABLES
names, date_of_birth, age, address, gender, work
OPERATIONS
birthday(nm) =
　　PRE nm ∈ names
　　THEN age(nm) := age(nm) + 1
ad ← current_address(nm) =
　　PRE nm ∈ names
　　THEN ad := address(nm)
make_person(nm) =
　　names(nm) := null | date_of_birth(nm) := null |
　　age(nm) := null | address(nm) := null |
　　gender(nm) := null | work(nm) := null
ここで操作 birthday は内部状態 age,persons のみを扱うのに対し、操作 make_person は抽象機械 Person 全体に対する操作である。このように操作により扱うシステムの範囲が異なる操作が仕様内に混在する。

図 3.7: 混在する内部状態へのアクセス

B-抽象機械表現では仕様記述単位である一つの抽象機械の中で、内部状態変化を引き起こす操作が複数宣言されることが多い。(図 3.7) その際、操作ごとに影響を及ぼす内部システムの抽象レベルが異なることがあり、理解を複雑にする。

これは抽象機械記法が、次の特徴を意識した記法であることから生じる問題である。その特徴とは B-Toolkit の開発支援システムが全体を表現する変数を保持することである。このため支援システムを用いない純粋な仕様内では、システム全体に対して操作の及ぼす影響範囲が明示されない。

module! Person {
[Name Birth Age Address Gender Work]
[Private Public Person]
op person : Name Birth Age Address Gender Work → Person
op private : Name Birth Age Gender → Private
op public : Name Addrress Work → Private
op birthday : Name Private → Age
op make-person : Name Person → Person
関数 person や private で内部状態と全体システムの状態関係を記述している。
また関数 birthday は求めるデータをもつ抽象状態 Private のみを扱う。

図 3.8: 明確な名前付けによる関数適応範囲の明示

B-抽象機械を関数で表現する作業である本研究では、この問題を解決するために抽象レベルごとに積極的にソートを宣言することが有益であると考えられる。(図 3.8) これは複数の内部状態から構成される抽象化された状態を意識することが有益であるという主張でる。さらなる理解のために仕様内で全体システムと抽象化で得られる部分システムの関係をすべて記述することが可能である。しかし、実際的にはある組合せにおいてしか状態変化を起こす関数の適応はされないことが多くある。その時、無理に抽象状態を作成する必要はない。対象システムが小さいときは理解も難しくなく、抽象状態を作成することにより仕様がふくれることによる理解の妨げが生じる。従って機械的に組み合わせを考え関数の引数を１つの状態のみに制限する必要は無い場合が考えられることも意識しなければならない。理解しやすい抽象レベルには個人差が存在する．対象システムが小さいとは，振舞いが単純なシステムを指す。CafeOBJ 上では公理として与える等式内に出現する項の変化から考えることができる。

操作が扱える状態を一つに制約することは段階的な仕様設計の考え方に沿うものである。対象システムの設計を行うとき現在判明している事実 (性質) を記述し、それを基に拡張していくことでより詳細かつ完全な全体システムの設計が可能である。従って仕様作成の途中段階では内部状態に関しては考慮されず、この時点である状態を構成する複数のサブ状態を記述することはできない。このことはトップダウンによりシステムを記述する際に判明する事実である。逆に前記の事実はボトムアップ時に判明する事実である。

3.3 仕様の構造化

B は仕様を構造化する手段を提供しているが、同様の機能を CafeOBJ はもつ。どのように実現すればよいか B 言語における構造化の節の意味を説明し、同時に対応法を与える。
B-Technology では詳細化に関する手法が B-Method で与えられている。今までに説明を行った仕様は詳細化が行われていない初期の仕様であった。B-抽象機械記法では、詳細段階ごとに仕様記述単位である抽象機械の名前を変えている。抽象機械 (MACHINE)、設計変更された機械 (REFINEMENT)、実装のための機械 (IMPLEMENTATION) の３段階の名前である。構造化を助ける節は抽象機械 (MACHINE)、設計変更された機械 (REFINEMENT)、実装のための機械 (IMPLEMENTATION) のどのレベルで利用可能であるか、そして節の意味もそれぞれ異なる。ここでは抽象機械レベルと設計変更された機械レベルについて CafeOBJ での記述法を与える。実装のための機械は B-Technology の１つとして提供される実行コードを利用した仕様である。これは B-Technology が統合された環境を提供しているために実現できるものである。CafeOBJ は実行可能な仕様記述言語ではあるが、実装そのものについてはその範囲でない。したがって本研究では実装のための機械レベルでの構造化の対応について扱わない。

最初に抽象機械上での構造化について CafeOBJ での対応を与える。

SEES

B の SEES 節は参照される抽象機械の定数、集合、変数に対する読みだしのみのアクセスを許す。これは問い合わせ関数 (ショートカット) の利用を許す意味である。また SEES 節は参照される抽象機械の操作の利用を許す。SEES 節で参照される抽象機械は参照する抽象機械とは独立した存在であり、参照する抽象機械の設計変更に影響を受けない。また参照する抽象機械に影響が発生するような設計変更を行った抽象機械は SEES 節で参照される抽象機械にならない。このことは後述する詳細化の際に考慮する必要がないことを意味する。
SEES 節は CafeOBJ 上の protecting 命令で対応する。
一般に制約が厳しいため SEES 節で参照される抽象機械は定数のみが多い。

USES

B の USES 節は半隠蔽機能をもつ輸入関係で、参照される抽象機械の定数、集合、変数に対するアクセスを許すが、操作の利用を許さない。変数は不変条件で利用可能であり、また意味を損なわない時参照する機械上の操作の左辺で利用できる。
USES 節は CafeOBJ 上の extending、using 命令で対応する。不変条件に出現することが可能な為、別モジュールで等式を付加される可能性があることを意味する。これは一般に protecting 命令では許される物

でない。輸入に際して、被輸入モジュール上の関数の利用はできない。この制限を実現する CafeOBJ の輸入機構は無い。この問題に対しては本論文の最後で考察を与える。

INCLUDES

B の INCLUDES 節は参照される抽象機械の定数や集合を、参照する抽象機械の物として扱うことを許す。参照される抽象機械の変数は、参照する抽象機械の不変条件に利用可能である。また変数は参照する抽象機械の変数と互いに素なものでなければならない。したがって参照される変数は読みだしのみで、参照する抽象機械上の操作の左辺には出現できなく、その利用は参照される抽象機械の操作を通して扱う。当然であるが参照される抽象機械の操作は、参照する抽象機械上で利用可能である。
これを実現する CafeOBJ の輸入命令は extending で対応する。USES 節と同様に変数が不変条件内に出現することから protecting 命令は不適当で、操作の左辺に出現することがなく、また互いに素であることから using 命令では条件を満たさない。

次に設計変更された機械レベルでの構造化について CafeOBJ での対応を与える。

SEES

設計変更された機械レベルで利用可能な構造化に関する節は SEES のみである。参照される機械の変数、集合、定数は参照する機械上で利用可能である。しかし変数は不変条件節では利用できない。
これを実現する CafeOBJ の輸入命令は protecting である。

このような対応で B-抽象機械の構造化は CafeOBJ 上で実現可能である。しかし INVARIANT 節などで、参照される抽象機械の変数の性質を記述する時に意味上の破壊を伴わないのでものには protecting 命令が利用できる。これは輸入されるモジュール上の構成子関数を利用することで同じ機能が提供できるときは、特に extending,using 命令を利用する必要はない。従って必ず上記の対応がつくとは限らない。すなわち構造化の対応は上記を基礎とし次のような指針を別に与える。他の抽象機械を参照するためだけの輸入は CafeOBJ 上の protecting 命令で、構成要素を加える輸入は extending 命令で、意味変更を伴う輸入は using 命令を用いることで解決する。

```
抽象機械(Abstract Machine)

MACHINE N(p)     N:machine name, p:paramers
CONSTRAINTS C    C:述語列(パラメタに対する制約)
SETS St          St:sets(BOOLなど組み込みでも提供)
CONSTATNTS k     k:定数列
PROPERTIES B     B:述語列(定数に対する制約)
VARIABLES v      v:状態変数列
INVARIANT I      I:述語列(抽象機械が扱える変数に対する制約)
INITIALISATION T  T:初期化式(Iを満たすもの)
OPERATIONS
 y <- op(x) =
     PRE P THEN S END
END
```

図 3.9: B-抽象機械の一般形

3.4 仕様の無矛盾性の証明

B-Method では抽象機械の設計において仕様に矛盾が無いかを確かめる為に証明すべきことを定義している。ここで B-抽象機械の一般形 3.9を再び紹介する。

このとき仕様の無矛盾性に対する証明は次のように定義されている。この証明は B-抽象機械記法が型付けされた集合論を利用している。

(1) $\exists p.C$

制約 (CONSTRAINTS) を満たすパラメタが存在することを示す。

(2) $C \Rightarrow \exists(St,k).B$

制約 (PROPERTIES) を満たす定数と集合が存在することを示す。

(3) $B \bigwedge C \Rightarrow \exists v.I$

制約 (INVARIANT) を満たす抽象機械の状態が空でないことを示す。

(4) $B \bigwedge C \Rightarrow [T]I$

正しく初期化が出来ること、すなわち初期化が停止することを示す。

初期後も不変条件を満たす, すなわち不変条件を満たす初期化ができる。

(5) $B \bigwedge C \bigwedge I \bigwedge P \Rightarrow [S]I$

正しく演算が出来ることを示す。演算後も不変条件を満たす。

(1-3) がモデルの存在についての証明で、(4,5) が不変条件の保存についてである。これらが不成立では

仕様を満たす実行可能なシステムが得られない。

本研究で利用する CafeOBJ はそのサブセットとして項書換えシステムを内部にもつ。等式を左辺から右辺への書換え規則と見ないし項を書換えることで仕様の検証をすることができる。B のアプローチと同様に仕様に対し次のような調査を行う。
CafeOBJ 上での仕様の無矛盾性に関する証明
(1,2) 制約を満たすパラメータ、定数、集合の存在と、
(3) 不変条件を満たす変数 (状態) の存在を、
制約を表現するソート上の項が存在するかを調べることで証明する。
代数仕様言語である CafeOBJ はソートの概念をもち、その仕様はソート付けされた世界で表現される。CafeOBJ システムは仕様をシステムに取り込む時に項のソート情報を計算し、公理に記述される等式の右辺と左辺のソートが正しいかの確認をする。また宣言されていないソートに関する演算が定義されていないかを確かめる。始代数に基づく仕様ではモデルの存在は明示的に宣言されることである。また隠蔽代数に基づく仕様では一定の抽象レベルでモデルが存在することを仮定した上での仕様記述である。従ってこれら (1-3) の証明は、システムに仕様を取り込むことで確かめることができる。
B はこのモデルの潜在についての証明を型付けしないパラメタを含む抽象機械 (CONSTRAINTS 節を持たない抽象機械) に対しても行える。CafeOBJ 上ではソートの概念を持つことから、型付けされないものに対してはソート Elt で表現することが対応する。ここでソート Elt は明確に型付けされていない型を意味する。
(4) 初期化は、初期化を示す公理が不変条件を保存するかを調べることで証明する。
(5) 演算後も不変条件を満たすことは、演算後の出力項のソートを調べることで証明する。このとき条件に応じた観測関数を補助的に用いる。
先に述べたように CafeOBJ システムは仕様をシステムに取り込む時に項のソート情報を計算する。従って実際には演算後に書換えられた項のソートが望む物であるかは最初に保証される。そこで (5) の証明は演算に渡す引数のソートが望むソートであることを示すことになる。

3.4.1 B の仕様 Coin Slot の無矛盾性の証明

ここで Coin Slot(図 3.10) に対する無矛盾性の証明は次のようになる。
(1) 制約 (CONSTRAINTS) を満たすパラメタが存在可能
(2) 制約 (PROPERTIES) を満たす定数、集合が存在可能
(3) 不変条件 (INVARIANT) を満たす状態が存在可能

```
MACHINE Coin_slot
SEES Vend_data
VARIABLES
        cstate, current_coin
INVARIANT
        cstate ∈ CSTATE
        current_coin ∈ COINS
INITIALISATION
        cstate := coin_absent ||
        current_coin := 5
OPERATIONS
        give_change(cc) def
                PRE cc ∈ COINS
                THEN cstate := coin_absent
                END;
        coin ← accept_coin def
                ANY cc
                WHERE cc ∈ COINS
                THEN
                current_coin := cc ||
                cstate := coin_present ||
                coin := cc
                END
END
```

図 3.10: B-抽象機械仕様 Coin Slot

$\exists\, cstate.(cstate \in CSTATE)$

$\exists\, current\_coin.(current\_coin \in COINS)$

(4) 初期化後も不変条件を満足

$[cstate := coin\_absent](cstate \in CSTATE)$

$[current\_coin :\in COINS](current\_coin \in COINS)$

$i.e. : \forall\, vv.(vv \in COINS \Rightarrow (current\_coin \in COINS)[vv/current\_coin])$

(5) 演算後も不変条件を満足

$cc \in COINS \bigwedge cstate \in CSTATE \Rightarrow coin\_absent \in CSTATE$

$cc \in COINS \bigwedge current\_coin \in COINS \bigwedge cstate \in CSTATE \Rightarrow coin \in COINS \bigwedge coin\_present \in CSTATE$

これらが満たされるとき仕様の無矛盾性が証明される。B-Toolkit はこの証明を支援する。

3.4.2 CafeOBJ の仕様 Coin Slot の無矛盾性の証明

CafeOBJ で作成したCOIN-SLOT(図3.11) 仕様はB-抽象機械で定義された仕様よりも詳細なものとなっている。具体的にはB-抽象機械において操作 give_change で引数として cc を持たしているが、利用法は記述されていない。CafeOBJ の仕様では内部状態として cc を保持するように仕様を記述している。次で行う証明はこの記述に対しても十分なものである。またB-抽象機械仕様で accept_coin 操作は2つの作用 (内部状態へのアクセスと出力) がある。これを操作を分解して2つの関数の合成で実現している。従ってB での accept_coin 操作は CafeOBJ での entered-coin(accept-coin(CSlot,Coin)) が対応する。関数 currnet-state は全体システムを表現する CSlot から目的の内部状態を得るための関数である。状態変化が関数の合成で表現されるときについては４章で考察する。

CafeOBJ で Coin Slot の仕様の無矛盾性の証明を行う。(図 3.12)
この証明において CSlot の構造に関する帰納法を用いている。ここで Coin Slot はスタンダードなモデルで記述した物がすべてであると考える。そこで証明に構造帰納法を利用することができる [20]。

B での証明と同様に (1)、(2) に対する証明は必要無い。reduce コマンドにより書換えられた項はすべて望むソートを返す。
(1-3)(ここでは (3)) に対する証明は書換えを必要としないので parse コマンドで同様の意味をもつ。

このように、B-抽象機械で行っている仕様の無矛盾性に対する証明は型チェックであり、CafeOBJ がも

```
module! VEND-DATA1 {
        [ Cstate, Coins ]
        op coin-present : → Cstate
        op coin-absent : → Cstate
        op zeroc : → Coins
        op fivec : → Coins
        op tenc : → Coins
}

module! COIN-SLOT {
        protecting(VEND-DATA1)
        signature {
            [ CSlot ]
            op empty-cslot : → CSlot
            op _ | _ : Cstate Coins → CSlot
            op give-change : CSlot Coins → CSlot
            op accept-coin : CSlot Coins → CSlot
            op entered-coin : CSlot → Coins
            op current-state : CSlot → Cstate – for proof
        }
        axioms {
            var CS : Cstate
            vars CO1,CO2 : Coins
            var CL : CSlot
            eq empty-cslot = coin-absent | fivec .
            eq accept-coin(CL, CO2) = (coin-present | CO2) .
            eq give-change(CL,CO1) = (coin-absent | CO1) .
            eq current-state(CS | CO2) = CS . – for proof
            eq entered-coin(CS | CO2) = CO2 .
        }
}
```

図 3.11: CafeOBJ 仕様 Coin Slot

```
(3) 不変条件 (INVARIANT) を満たす状態が存在可能
reduce coin-present .
reduce fivec .
(4) 初期化後も不変条件を満足
reduce current-state(empty-cslot) .
reduce entered-coin(empty-cslot) .
(5) 演算後も不変条件を満足 (entered-coin も同様)
open .
op base-cslot : → CSlot .
reduce current-state(give-change(empty-cslot, fivec)) .
reduce current-state(give-change(base-cslot, fivec)) .
reduce current-state(give-change(give-change(base-cslot, fivec), fivec)) .
reduce current-state(give-change(give-change(base-cslot, tenc), fivec)) .
reduce current-state(give-change(accept-coin(base-cslot, fivec), fivec)) .
reduce current-state(give-change(accept-coin(base-cslot, tenc), fivec)) .
reduce current-state(give-change(empty-cslot, tenc)) .
reduce current-state(give-change(base-cslot, tenc)) .
reduce current-state(give-change(give-change(base-cslot, fivec), tenc)) .
reduce current-state(give-change(give-change(base-cslot, tenc), tenc)) .
reduce current-state(give-change(accept-coin(base-cslot, fivec), tenc)) .
reduce current-state(give-change(accept-coin(base-cslot, tenc), tenc)) .
close
```

図 3.12: CafeOBJ 仕様 Coin Slot の無矛盾性の証明

```
MACHINE M(p)                    REFINEMENT N
CONSTRAINTS C                   REFINES M
CONSTANTS k                     CONSTANTS k1
PROPERTIES B                    PROPERTIES B1
VARIABLES v                     VARIABLES w
INVARIANT I                     INVARIANT J
INITIALISATION T                INITIALISATION T1
OPERATIONS                      OPERATIONS
 y <- op(x) =                    y <- op(x) =
     PRE P THEN S END                PRE P1 THEN S1 END
 ...                             ...
END                             END
```

図 3.13: B-抽象機械と詳細化された機械

つソートの概念と CafeOBJ のシステムにより保証されるものである。したがって証明の為に用意した関数 currnet-state は本来必要無く、この仕様の無矛盾性は満たされる。

3.5 段階的詳細化 (Refinement)

近年コンピューターの発達により様々な分野でソフトウェアシステムが利用されるようになっている。そしてそのシステムは巨大な物となっている。このような巨大なシステムの詳細な設計を一度に決定することは難しい。そこでシステムの設計を行う際、 現時点で決定している設計を記述し、それをより具体的な設計へと変えていく方法をとると、大規模システムの構築が容易に行える。具体化する際，先に記述された設計を新たな設計に反映させる。これを繰り返すことで完全な設計を得る。この過程を段階的詳細化という。この節では B-Method で行われる段階的詳細化手法が代数仕様言語 CafeOBJ で実現できることを示す。

B-抽象機械の詳細化は詳細化される前の仕様と後の仕様が与えられた時 (図 3.13)、次のような詳細化に関する無矛盾性が満たされていれば正しい段階的な設計が行えたとしている。

このとき B-仕様の詳細化に関する無矛盾性の証明は次の３つからなる。

(1) $C \bigwedge B \bigwedge B1 \Rightarrow \exists(v, w).(I \bigwedge J)$

抽象機械の不変条件 I と具体的に定義された機械の条件 J を満たすような状態が実現できる。

(2) $C \bigwedge B \bigwedge B1 \Rightarrow [T1]\neg[T]\neg J$

具体化された初期化は抽象機械で宣言されている性質も満たす。
(3) $C \bigwedge B \bigwedge B1 \bigwedge I \bigwedge J \bigwedge P \Rightarrow P1 \bigwedge [S1']\neg[S]\neg(J \bigwedge y' = y)$
具体的に定義されたされたすべての演算に対し、同じ初期状態において、その得られる結果が同じである (結果の関係が等しい) 演算が存在する。

これらが成立する時仕様が詳細化されたと定義している。
これらが不成立なとき正しく詳細化されず仕様を満たす実行可能なシステムが得られない。

3.5.1 CafeOBJ における詳細化

CafeOBJ 仕様において詳細化に関する無矛盾性は型に関するチェックをソートの概念で説明できるため、詳細化される前のモジュールにおける制約が詳細化後のモジュールで実現されているかを示すことである。そして詳細化されたすべての演算に対し、同じ初期状態において、その得られる結果の関係が等しい演算が存在することを示すことである。ところでCafeOBJ では状態に対する制約も操作の本体も同じように公理で与えられる。そこで公理が詳細化後に保存されているかを調べることが詳細化の証明となる。CafeOBJ における詳細化は形式的には次のように定義されている [7] [8]。

詳細化
仕様 (Σ, E) の詳細化により得られるひとつの実現を仕様 (Σ', E') とする。このときΣのソート集合を S,Σ′のソート集合を S'とする。
指標Σから指標Σ′への写像φ が存在し、φ により得られる公理 Eの像$\varphi(E)$ が (Σ', E') 上で成立する。すなわちあらゆるΣ′-代数 A'において$\varphi(A') \models E'$の時，(Σ', E') を (Σ, E) の詳細化仕様とする。
ここで$\varphi : \Sigma \to \Sigma'$, $\varphi :< f, g >$. ソートのマップ $f : S \to S'$は、$f([]) = [], f(s_1 ... s_n) = f(s_1)...f(s_n)$ の関係をもつ。また $g_{w,s} : \Sigma_{w,s} \to \Sigma'_{f(w),f(s)}$とする。
このときソート集合 S の実データの表現である観測可能なソートは共通である。また公理、等式を写像するとは、詳細化モジュールでの解釈を与えることを意味する。
理解の為に説明をケース分けをし詳細化の意味を説明する。

- 隠蔽代数間での詳細化は、振る舞いに関しての等式が詳細化された仕様の上で定義される新しい振る舞いに関して等しい時 (振る舞いの関係が保存されている時) で、その仕様を詳細化仕様とする。

- 隠蔽代数から可視な世界 (始代数) で表現された仕様への詳細化は、振る舞いに関して等しいと定義されていた等式が可視な世界での等式において成立すれば良い。すなわち具体的に等しい関係で表現されるように解釈を与え、その解釈に基づいて成立するときその仕様を詳細化仕様とする。

- 可視な世界で表現された仕様から可視な世界で表現された仕様への詳細化はもとの等式がそのまま成立する時で、その仕様を詳細化仕様とする。
  成立しない時は詳細化ではなく異なる設計と考える。

3.5.2 CafeOBJ による詳細化事例

CafeOBJ が隠蔽代数を利用して振る舞いに注目した仕様記述ができることは既に述べた。抽象機械のように状態と状態を変化させる操作をもつシステムがどのように隠蔽代数で記述されるか比較する。
Explicit State Approach

- 状態空間は１つのソートで表現, それぞれの状態はそのソートの要素として明示的に項で表現
- 初期状態は状態空間を表すソートの１つの要素 (項)
- 状態変化を行う操作は、状態を引数にとり操作を加えた後の状態を返す構成子関数として表現

Hidden sort Approach

- 状態空間は一つの隠蔽ソートで表現。それぞれの状態はそのソートの要素と解釈
- 初期状態は隠蔽ソートの中の初期状態の意味を持つ一つの要素
- 状態変化を行う操作は、状態を引数にとり操作を加えた後の状態を返す構成子関数として表現
- 状態は観測関数を用いて観測

これからわかるように仕様の表現としては似たものである。しかし記述する対象システムをどのように扱うかにおいて意識が大きく異なる。
初期状態は隠蔽代数の仕様においては定義されている必要は特にない。また状態変化を記述した関数に与える現在の状態がどのような履歴を持っているかにも関心はない。観測した結果得られる可視ソート上のデータ

```
module* COIN-SLOT2 {
        protecting(VEND-DATA1)
        signature {
            *[ CSlot ]*
          – op empty-cslot : → CSlot
            bop give-change : CSlot Coins → CSlot – method
            bop accept-coin : CSlot Coins → CSlot – method
            bop entered-coin : CSlot → Coins – attribute
            bop current-state : CSlot → Cstate – attribute
        }
        axioms {
            var CO : Coins
            var CL : CSlot
          – eq current-state(empty-cslot) = coin-absent .
            eq current-state(give-change(CL, CO)) = coin-absent .
            beq entered-coin(accept-coin(CL, CO)) = CO .
        }
}
初期化に関する行をコメントアウトしている.
```

図 3.14: CafeOBJ での抽象的仕様 COIN-SLOT2

を利用して記述する。すなわち観測関数に隠蔽ソートで表現された状態を渡した結果、システムがどのように振る舞うかを記述する。このようにして詳細な設計が確立していないシステムの仕様を与えることができる。

仕様の無矛盾性の証明において利用した CafeOBJ 仕様 Coin-Slot が詳細化されていないとき、図 3.14 のように記述できる。

COIN-SLOT2 システムがどのような内部状態をもつかを隠蔽し，その構成を記述していない．先に利用した COIN-SLOT では図 3.15のように宣言されていた.

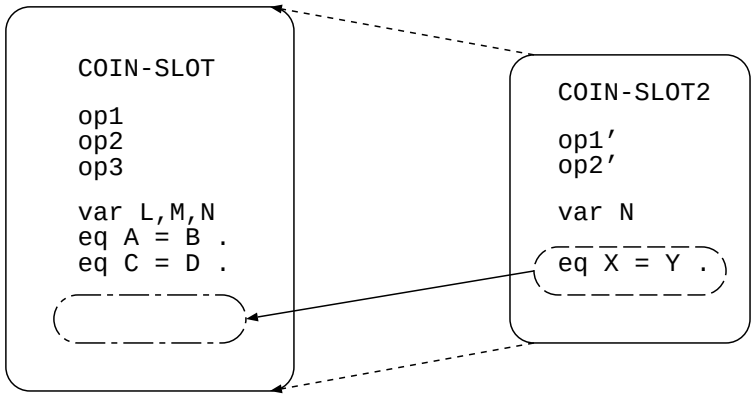
それ以外の関数はソートも同じように宣言されている。しかし等式は異なる。公理は観測関数 (attribute) を与えて記述されている。詳細化において表現したい関数が複数に分割され、その合成によって得られるような設計が考えられる。これについては次章で説明する。一般に計算したい結果に必要な物と計算によって得られる値は同じであるからであるのでソートに関する変更は現れない。

module COIN-SLOT { [Cstate Coins CSlot]

op _ | _ : Cstate Coins → CSlot

内部状態 Cstate と Coins からなる状態 CSlot

図 3.15: COIN-SLOT の内部状態




```
eq current-state(give-change(CL, CO)) = coin-absent
eq entered-coin(accept-coin(CL, CO)) = CO .
-- eq current-state(empty-cslot) = coin-absent .
```

図 3.16: COIN-SLOT システムの詳細化

仕様COIN-SLOT2 の公理が、その詳細化仕様COIN-SLOT において保存されている (図3.15) ことを証明する。仕様COIN-SLOT2 の指標から仕様COIN-SLOT の指標への写像$\varphi : \Sigma^{COIN-SLOT2} \rightarrow \Sigma^{COIN-SLOT}$は表 3.17のように定義する。

この事例では指標写像により関数名が変わらず、扱うソートも同じである。したがって表3.16のように仕様COIN-SLOT2 の等式はそのまま詳細化仕様COIN-SLOT において成立することを示せば証明できる。この証明も CafeOBJ の実行機能を用いて行う。(図 3.18)

このように CafeOBJ の reduce コマンドでシステムは true を返す。
これから隠蔽代数を用いて定義された詳細化前仕様 COIN-SLOT2 に定義された等式が詳細化後の仕様 COIN-SLOT で満たされることが証明された。これは詳細化前の仕様の性質が保存されていることを示す。

	COIN-SLOT2	COIN-SLOT
Sort	CSlot	CSlot
	Cstate	Cstate
	Coins	Coins
$\Sigma_{CSlotCoins,CSlot}$	give-change	give-change
	accept-coin	accept-coin
$\Sigma_{CSlot,Cstate}$	current-state	current-state
$\Sigma_{CSlot,Coins}$	entered-coin	entered-coin
$\Sigma_{[],CSlot}$	empty-cslot	empty-cslot

図 3.17: 詳細化の指標写像

red current-state(give-change(empty-cslot, fivec)) == coin-absent . COIN-SLOT⟩ open .
– opening module COIN-SLOT.. done.
%COIN-SLOT⟩ op cslot-tmp : → CSlot .
%COIN-SLOT⟩ op coin-tmp : → Coins .

%COIN-SLOT⟩ red current-state(give-change(cslot-tmp, coin-tmp)) == coin-absent .
–
– reduce in % : current-state(give-change(cslot-tmp,coin-tmp)) == coin-absent
true : Bool
(0.017 sec for parse, 3 rewrites(0.000 sec), 3 match attempts)

%COIN-SLOT⟩ red entered-coin(accept-coin(cslot-tmp, coin-tmp)) == coin-tmp .
– reduce in % : entered-coin(accept-coin(cslot-tmp,coin-tmp)) == coin-tmp
true : Bool
(0.000 sec for parse, 3 rewrites(0.000 sec), 3 match attempts)

%COIN-SLOT⟩ red current-state(empty-cslot) == coin-absent .
– reduce in % : current-state(empty-cslot) == coin-absent
true : Bool
(0.033 sec for parse, 3 rewrites(0.017 sec), 3 match attempts)

%COIN-SLOT⟩ close
初期化の式を与えたとして証明を行っている。

図 3.18: 詳細化の証明スコア

3.6 抽象レベルのソートに対する考察

本節では、前節で紹介した隠蔽代数の事例を踏まえ、3.2 節で行った内部状態に名前付けをおこなう提案に対する考察を別の観点から行う。
その他の仕様言語による仕様作成においても内部状態に関する問題が生じる可能性がある。さきほどの仕様の可読性のアプローチと異なり、仕様作成の時に内部状態の問題がどのように出現するかを考える。
システムの仕様を記述するにあたり、詳細化された仕様のイメージを持ちながら、詳細の決定を保留した抽象的な仕様を記述することが一般的と考えられる。すなわちある程度実装を意識しながら仕様記述は行われる。システム全体に対して同じ抽象度をもって、仕様を記述することは難しい。いくつかの内部状態を利用する操作の記述を行ったとき、ある部分に対しては詳細が決定されていないので、それらを扱う操作をうまく記述できない可能性がある。例えば図 3.11において出現した give-change の引数 cc の扱いの問題。内部状態をあまり認識していないとき、すなわち B-抽象機械で表現されたあいまいな仕様 Coin Slot(図 3.10) を忠実に記述した時、仕様の実行は不可能になる。しかし図 3.14で示した隠蔽代数を用いた例では抽象度を明確にしているので仕様の実行が可能である。これは仕様の性質を証明するときなどに大きな利点となる。実行可能な仕様を得るために無理に決定されていないデータ構造を記述することは好ましくない。抽象状態を明確にして仕様を記述することは有効で、特に隠蔽代数は振る舞いをもとに仕様の記述が行える。このことからも内部状態に名前付けをおこなうことは重要であると考えられる。

3.7 まとめ

- B-抽象機械で記述された状態と状態に対する操作から定義される仕様が CafeOBJ でどのように記述されるかを調べ、記述の指針を与えた。

- B-抽象機械の操作が内部状態を変更する際に、記述対象が明確にされていない問題をあげ CafeOBJ において明確に表現する為に内部状態を扱うソートを利用するように指針を与えた。

- B-抽象機械記法がもつ構造化に関する記法を CafeOBJ でどのように表現するかを考察した。

- B-Method が提供する仕様の無矛盾性に対する証明が CafeOBJ でどのように扱われるかを示し、CafeOBJ がもつ仕様の実行機能を用いて証明を行った。

- モデル指向で設計された抽象的な仕様を代数仕様言語で記述する為の技術として Goguen らが提唱している隠蔽代数を利用した。B-Method で提供される詳細化に関する無矛盾性の証明が CafeOBJ でどのように扱われるかを示し、これに対しても CafeOBJ がもつ仕様の実行機能を用いて証明を行った。

第 4 章

事例研究

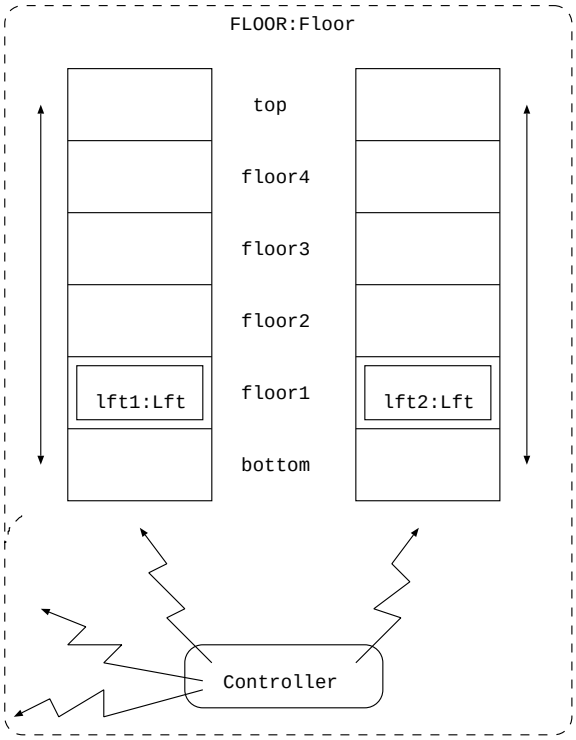
前章で B 言語がもつその設計段階に注目し、そのアプローチが代数仕様言語 CafeOBJ 上で実践できることを示し有効性を示した。
本章では CafeOBJ がもつ理論をに利用することで問題に有効に記述した仕様が得られることを示す。
本章の事例においても B 仕様は幾つかの標準ライブラリを利用している。そこで今回利用するライブラリに相当する機能を module 内で作成している。したがって CafeOBJ 仕様の方が記述量は多くなっている。これはライブラリの表現に左右されない自由な仕様作成が行える環境を CafeOBJ が提供しているからにすぎない。もちろん CafeOBJ におけるライブラリの研究は広く行われている [17][18]。
形式手法の習得は多くの場合ドキュメントが提供されそれを理解する、もしくは専用の研修を利用することで進められる。B-Technology の特徴として言語習得のために CaseStudy を重視し多くの事例を提供している所があげられる。本章で用いる事例は、その CaseStudy として利用されている事例である。

4.1 事例 LIFT

本事例は複数のリフトからなるエレベータシステム (図 4.1) である。３章で与えた指針に基づき B-抽象機械仕様を CafeOBJ で記述し、中規模の仕様を作成した。詳細な仕様は付録にまわすが、６つの内部状態を持ち、１２個の操作から形成される。また構造化を持たない仕様である。

4.2 隠蔽代数を用いた段階的詳細化事例

まず隠蔽代数を用いた１機のリフトからなるエレベータシステムを与え、その詳細化を考える。この詳細化は隠蔽代数を用いた仕様から可視な世界で表現された仕様への詳細化である。前章で考察した詳細化




```
op request-lift :
                  Floor Direct State -> State
op request=floor :
                     Lft Floor State -> State
op continue-up :
                              Lft State -> State
op continue-down :
                              Lft State -> State
op pickup-or-deliver-and-continue :
                              Lft State -> State
op stop :
                              Lft State -> State
op pickup-and-change-dir :
                              Lft State -> State
op pickup-and-same-dir :
                              Lft State -> State
op close-door :
                                  State -> State
op open-door :
                                  State -> State
op depart-and-move-up :
                              Lft State -> State
op depart-and-move-down :
                              Lft State -> State
```

```
op state : FlrSet DirSet DorSet MovSet InSet OutSet LftSet FloorSet DirectSet
           -> State
```

図 4.1: 事例 リフトシステム

```
module HSLFT {
  protecting(FLOOR)
  protecting(LFT)
  signature {
    *[ Hslft ]*
    op initial-hslft : Lft -> Hslft
    bop request-slft : Floor Hslft -> Hslft    -- method
    bop request-sfloor : Floor Hslft -> Hslft  -- method
    bop sfloor? : Hslft -> Floor               -- attribute
  }
  axioms {
    vars FLOOR FLOOR1 : Floor
    var LFT : Lft
    var HSLFT : Hslft
    eq sfloor?(initial-hslft(LFT)) = floor1 . -- LFT the name of current lft
    eq sfloor?(request-slft(FLOOR, HSLFT)) = FLOOR .
    eq sfloor?(request-sfloor(FLOOR, HSLFT)) = FLOOR .
  }
}
```

図 4.2: 隠蔽代数をもちいたリフトシステムの仕様

との違いは詳細化により操作を表す関数が細かく定義されることである。関数は詳細化された仕様内の関数の合成で表現される。

詳細化の定義を次のように変更する。

詳細化２

仕様 (Σ, E) の詳細化により得られるひとつの実現を仕様 (Σ', E') とする。このときΣのソート集合を S,Σ' のソート集合を S'とする。

指標Σから指標 $Der(\Sigma')$ への写像φ が存在し、φ により得られる公理 Eの像$\varphi(E)$ が (Σ', E') 上で成立する。すなわちあらゆるΣ'-代数 A'において$\varphi(A') \models E'$の時，(Σ', E') を (Σ, E) の詳細化仕様とする。

ここですべての $w \in S^*, s \in S$に対して $Der(\Sigma)_{w,s} = T_\Sigma({}^\omega X)_s$である。

隠蔽代数をもちいた詳細化前の仕様 HSLFT(図 4.2) は利用者が一人であるようなリフトシステムである。リフトが現在までに動いてきた経歴やその向きに関する情報は意識せず、request-slft 命令の後リフトは呼んだ階に到着し、request-sfloor 命令の後リフトは望んだ階に到着するようなシステムである。リフトシステム全体がどのような情報を保持しているかは隠蔽されている。またこれら命令が適応された後、瞬時に移動が終る物とし移動途中にその他の命令は受け付けないとする。

今、このリフトシステムを５つの内部状態をもつシステムとして具体化する。システムは内部状態として、

	HSLFT		HLFT
Sort	Hslft	Sort	Hlft
	Floor		Floor
	Lft		Lft
$\Sigma_{Floor Hslft,Hslft}$	request-slft	$\Sigma_{Floor Hlft,Hlft}$	open-door(continue-n-up(request-lft(close-door()))))
	request-sfloor		open-door(continue-n-down(request-floor(close-door()))))
$\Sigma_{Lft,Hslft}$	initial-hslft	$\Sigma_{Lft,Hlft}$	initial-hlft
$\Sigma_{Hslft,Floor}$	sfloor?	$\Sigma_{Hlft,Floor}$	floor?

図 4.3: HSLFT から HLFT への指標写像

リフトが現在存在する階、リフトが動作している向き、リフトのドアの状態、現在の命令が与えた目的の階、現在の命令が与えた目的の向きを持つとする。また６つの状態遷移関数と１つの観測関数をもつ。
詳細後の仕様 HLFT は図 4.4のよう記述できる。これが HSLFT の正しい詳細化となっていることを示す。
詳細前仕様 HSLFT がもつ関数は表 4.3のように射影される。

その他の可視なソート Door,Direct は新たに導入されたもの。
request-slft と request-sfloor は導来演算で表現されている。

詳細化した仕様 HLFT において詳細前仕様 HSLFT に記述されている公理が保存されていることを示す。

公理 **eq sfloor?(initial-hslft(LFT)) = floor1 .**
変数 LFT はこのリフトシステムが１機からなることから意味の無いものである。後述する並行性の証明に利用するためで、ここでは名前をつけている程度の意味と考える。従ってリフトは１つしか存在せず、リフト１つに対してこの公理が満たされれば十分である。あるリフトを指す意味で定数lft を宣言し証明に利用している。

```
select HLFT .
open HLFT .
op lft : -> Lft .
red floor?(initial-hlft(lft)) == floor1 .
close .
```

公理 **eq sfloor?(request-slft(FLOOR, HSLFT)) = FLOOR .**
HSLFT に渡される名前をもつリフトを FLOOR 階から呼び、瞬時に状態遷移が完了した後リフトは呼び出した階に存在する性質

まず指標写像の表に記述したように、詳細化された仕様 HLFT の関数を組み合わせてできる１連の操作 (合成関数) が request-slft に対応する。open-door(continue-n-up(request-lft(close-door()))) 中の関数 continue-n-up は continue-up 関数が n 回適応されることを示している。これはリフトを呼ぶ階と、その時点でリフトが存在する階との関係により決定される。この等式は HSLFT で表現されるいかなる状態に対しても成立する必要がある。しかし詳細化前の仕様 HSLFT がもつ可視なソートは Floor と Lft のみで、前述の一機のリフトシステムであるという理由から。Lft は証明中定数で扱え、考慮すべきなのは Floor のみである。しかもこの request-slft 関数の意味は階数に意味をもたず、リフトを呼ぶ階とリフトが今いる階の階数の差にのみ意味をもつ。したがって証明は可能な階数の差すべてに対して行えばよい。またリフトシステムの内部状態を表現するのに用いられているソート Floor の項はリフトに対する命令を受け付ける時必ず同じである。また移動途中に他の命令を受け付けない。これは詳細化前のリフトシステムで定義されている事実である。その他の詳細化後に定義される可視なソート Direct や、Door についは詳細化システムで新たに定義することに注意しなければならない。証明において詳細化で関数の引数が増えているので詳細化前の request-lft 操作の意味を壊さない引数を増やした操作 (関数)request-lft2 を便宜上利用している。その他の場合分けとして階下にリフトがある状態、階上にリフトがあり状態、現在の階にリフトがあり状態での呼出についての証明が必要である。

```
module HLFT {
  protecting(FLOOR)
  protecting(DIRECT)
  protecting(DOOR)
  protecting(LFT)
  signature {
    [Hlft]
    op initial-hlft : Lft -> Hlft
    op floor? : Hlft -> Floor -- attribute
    op state : Floor Direct Door Floor Direct -> Hlft
    op request-lft : Floor Direct Hlft -> Hlft
    op request-floor : Floor Hlft -> Hlft
    op continue-up : Hlft -> Hlft
    op continue-down : Hlft -> Hlft
    op close-door : Hlft -> Hlft
    op open-door : Hlft -> Hlft
  }
  axioms {
    vars FLOOR FLOOR1 : Floor
    vars DIRECT DIRECT1 : Direct
    var DOOR : Door
    var LFT : Lft
    eq initial-hlft(LFT) = state(floor1, up, close, floor1, up) .
    eq floor?( state(FLOOR, DIRECT, DOOR, FLOOR1, DIRECT1)) = FLOOR .
    eq request-lft(FLOOR, DIRECT,
                   state(FLOOR1, DIRECT1, DOOR, FLOOR1, DIRECT1))
      = state(FLOOR1, DIRECT1, DOOR, FLOOR, DIRECT) .
    eq request-floor(FLOOR, state(FLOOR1, DIRECT1, DOOR, FLOOR1, DIRECT))
     = state(FLOOR1, DIRECT, DOOR, FLOOR, DIRECT) .
    eq continue-up(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
      = state(upper(FLOOR1), up, close, FLOOR, DIRECT) .
    eq continue-down(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
      = state(lower(FLOOR1), down, close, FLOOR, DIRECT) .
    eq close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT))
      = state(FLOOR, DIRECT1, close, FLOOR, DIRECT) .
    eq open-door( state(FLOOR, DIRECT1, close, FLOOR, DIRECT))
      = state(FLOOR, DIRECT1, open, FLOOR, DIRECT) .
  }
}
```

図 4.4: 詳細化後のリフトシステムの仕様

Continue-up 関数の階数に関して帰納法を利用し証明を行った。このとき Direct, Door は詳細化前のリフトシステムには存在しない要素であったことからこの証明で考慮する状態の意味を左右する物であってはならない。これらに関する新たに宣言された制約はこの証明において常に満たされる。

```
--> 詳細化前モデルの性質 (axiom) の証明２
--    eq sfloor?(request-slft(FLOOR, HSLFT)) = FLOOR .
-- for proof 階下にリフトがある状態での呼出についての証明
open .
ops floor upper-floor : -> Floor .
ops direct1 direct2 : -> Direct .    op door : -> Door .
op request-lft2 : Floor Direct Hlft -> Hlft .
vars FLOOR UFLOOR FLOOR1 : Floor .
vars DIRECT1 DIRECT2 : Direct .      vars DOOR : Door .
-- proof 1
-- continue-up 関数が一回含まれる時。すなわち一階上からリフトを呼ぶ時。
--> VVVV Base 1 should be true VVVV
op floor-tmp : -> Floor .
eq request-lft2(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(continue-up(request-lft(UFLOOR, DIRECT2,
     close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))))) .
red floor?(request-lft2(upper(floor-tmp), direct2,
state(floor-tmp, direct1, open, floor-tmp, direct1))) == upper(floor-tmp) .
--> VVVV Base n induction hypothesis be true VVVV
op upper-n : Floor -> Floor .         op continue-n-up : Hlft -> Hlft .
eq continue-n-up(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
   = state(upper-n(FLOOR1), up, close, FLOOR, DIRECT) .
eq request-lft2(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(continue-n-up(request-lft(UFLOOR, DIRECT2,
     close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))))) .
eq floor?(request-lft2(upper-n(FLOOR1), DIRECT2,
         state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))) = upper-n(FLOOR1) .
--> VVVV Base n+1 should be true VVV
op request-lft3 : Floor Direct Hlft -> Hlft .
eq request-lft3(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(continue-up(continue-n-up(request-lft(UFLOOR, DIRECT2,
     close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1)))))) .
red floor?(request-lft3(upper(upper-n(floor-tmp)), direct2,
        state(floor-tmp, direct1, open, floor-tmp, direct1)))
   == upper(upper-n(floor-tmp)) .
close .
-- for proof 階上にリフトがある状態での呼出
-- for proof 現在の階にリフトがある状態での呼出
```

公理 **eq sfloor?(request-sfloor(FLOOR, HSLFT)) = FLOOR .**

HSLFT に渡される名前をもつリフトに乗り込み目的の階 FLOOR をリクエストした後、瞬時に状態遷移が完了し、リフトが目的の階に存在する性質

証明は２個目の公理の時と同様の考え方で行える。

```
--> 詳細化前モデルの性質(axiom)の証明3
--    eq sfloor?(request-sfloor(FLOOR, HSLFT)) = FLOOR .
-- for proof リフトがある状態から階下に降りたい時の証明
open .
ops floor floor1 : -> Floor .
ops direct direct1 : -> Direct .
vars FLOOR FLOOR1 : Floor .
vars DIRECT DIRECT1 : Direct .
vars DOOR : Door .

--> Base 1 VVVVV shuld be true
op request-floor2 : Floor Hlft -> Hlft .
eq request-floor2(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(continue-down(request-floor(FLOOR,
    close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))))) .
red floor?(request-floor2(lower(floor),
   state(floor, direct, open, floor, direct1))) == lower(floor) .
-- proof n continue-down 関数が n 回含まれる時。すなわち n 階下にリフトで降りる時
--> Base n VVVVV be true
op lower-n : Floor -> Floor .
op continue-n-down : Hlft -> Hlft .
eq continue-n-down(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
   = state(lower-n(FLOOR1), down, close, FLOOR, DIRECT) .
eq request-floor2(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(continue-n-down(request-floor(FLOOR,
    close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))))) .
-- induction hypothises
eq floor?(request-floor2(lower-n(FLOOR),
   state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))) = lower-n(FLOOR) .
-- proof n continue-down 関数が n+1 回含まれる時。すなわち n+1 階下にリフトで降りる時
--> Base n+1 VVVVV shuld be true
op request-floor3 : Floor Hlft -> Hlft .
eq request-floor3(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(continue-down(continue-n-down(request-floor(FLOOR,
    close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT)))))) .
red floor?(request-floor3(lower(lower-n(floor)),
    state(floor, direct, open, floor, direct1))) == lower(lower-n(floor)) .
close .
--> 同様に上がりたい時も行う。
```

これらの簡約はすべて true を返す。

CafeOBJ の実行機能を用いることで仕様 HLFT が仕様 HSLFT の有効な詳細化であることが証明できた。

このように段階的詳細化技法を行って仕様を作成できることは大規模システム開発において重要である。

```
        op continue-down : Hlft -> Hlft
        eq continue-down(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
          = state(lower(FLOOR1), down, close, FLOOR, DIRECT) .
```

図 4.5: continue-down 関数

```
HLFT> match continue-down(state(bottom, down, close, floor3, up)) to +rules .
== matching rules to term : continue-down(state(bottom,down,close,
floor3,up))
+HLFT.6 is eq continue-down(state(FLOOR1,DIRECT1,close,FLOOR,DIRECT)
    ) = state(lower(FLOOR1),down,close,FLOOR,DIRECT)
 substitution : { FLOOR |-> floor3, DIRECT1 |-> down, FLOOR1 |->
       bottom, DIRECT |-> up }
```

図 4.6: match コマンドの例

4.3 CafeOBJ による到達不可能な状態の調査

本節では CafeOBJ が複雑なシステム開発に有効に利用できることを LIFT の事例で示す。CafeOBJ は開発支援として幾つかの便利な機能をもっている。そのひとつである **match** コマンドを利用して前節で詳細化した仕様 HLFT を用いて操作が適切に定義されているかを調べることができる。

リフトを１階降ろす操作 continue-down について考えてみる。continue-down 関数は図 4.5のように定義されていた。continue-down 関数が適応される状態 state を詳しくみてみる。　　変数 FLOOR1 は現在リフトが存在する階

　　変数 DIRECT1 は現在のリフトが動く方向

　　定数 close はリフトのドアの状態

　　変数 FLOOR はリフトが移動する目的の階

　　変数 DIRECT はリフトが目的の階に到着後に動く方向である。

関数の適応に関してはドアが開いているリフトに対して適応できないことのみ規定している。これでは最下階にリフトがいる時に continue-down 関数が適応できる。このことは match コマンドで確かめられる。(図 4.6) このように continue-down コマンドが最下階にいるリフトを引数にとれることがわかる。実際に簡約を行うと次のように帰ってくる。

現在の階は lower(bottom) となり、リフトは地下に潜ってしまう。(図 4.7) continue-down 関数の等式を条件付き等式に変更することで最下階にいるリフトに対して関数が適応されないように変更できる。(図 4.8)

他にソート Floor の要素を定義しているモジュールで bottom に対する関数 lower が bottom をとる時エ

```
HLFT> red continue-down(state(bottom, down, close, floor3, up)) .
-- reduce in HLFT : continue-down(state(bottom,down,close,floor3,
    up))
state(lower(bottom),down,close,floor3,up) : Hlft
(0.000 sec for parse, 1 rewrites(0.000 sec), 6 match attempts)
```

図 4.7: 不適切な状態遷移

```
op continue-down : Hlft -> Hlft
ceq continue-down(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
   = state(lower(FLOOR1), down, close, FLOOR, DIRECT)
   if FLOOR1 =/= bottom .
```

図 4.8: 変更後の continue-down 関数

ラーを返すように定義することもできる。
このようにして定義した仕様が望む性質を持っているかを調べることもできる。これはシステム仕様を作成していく際に、CafeOBJ の証明支援のためのコマンドを利用して開発を進めることを示しており、複雑なシステム設計において CafeOBJ が有効な開発環境であることを示している。

4.3.1 投影演算を用いた並行システムの記述

リフトシステムは数機のリフトから構成することができる並行システムであるが、最初から複数のリフトを扱う仕様を与えるのでは並行性に関する検証は容易ではない。そこで投影演算を利用して CafeOBJ の仕様を記述し、並行性に関する検証が容易に行えることを確かめる。

並行オブジェクトを合成して得られるようなシステムの仕様は投影演算を用いることで有効に記述できる [13][24][12]。ここでは先程用いた 1 機からなるリフトシステムを合成して得られるマルチリフトシステムを記述し、リフトに対する操作が並行独立に実行できることを示す。ここで投影演算は次のように定義される。

投影演算 (図 4.9)
合成オブジェクト O 上の隠蔽ソートを h、その要素オブジェクト群 O_n 上の隠蔽ソート群を h_n とし、合成オブジェクトから要素オブジェクト群への写像群を π_n とする。
h 上の定数は写像 π_n により、それぞれの要素オブジェクトの隠蔽ソート上の定数が得られる。

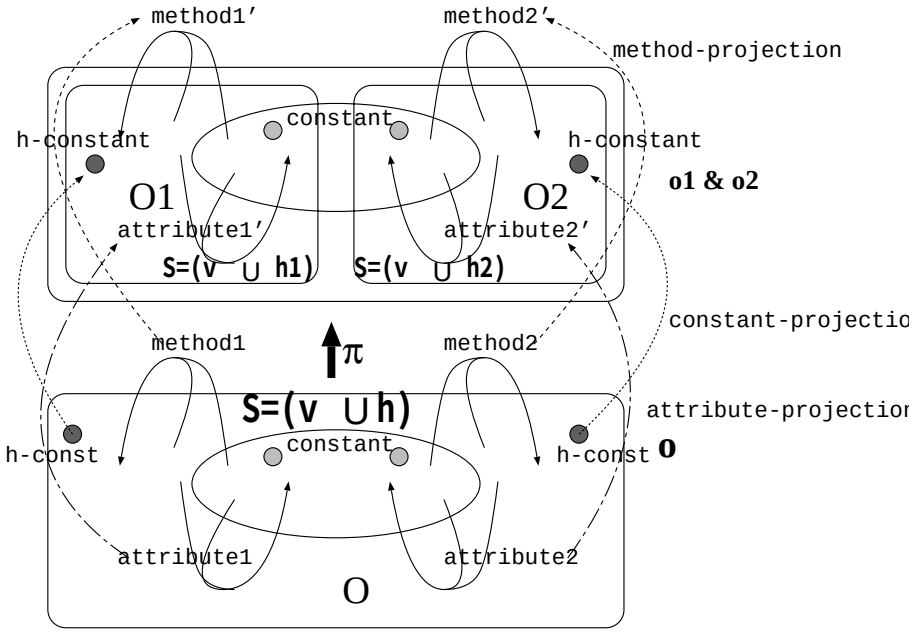


図 4.9: 投影演算イメージ

h上の method は写像π_nにより、それぞれの要素オブジェクトの隠蔽ソート上の method 列が得られる。
ブジェクトの隠蔽ソート上の attribute が得られる。
合成オブジェクト上の attribute は写像π_nにより、それぞれの要素オブジェクトの隠蔽ソート上の attribute が得られる。
このような演算π_nを投影演算と定義する.
ここで合成オブジェクトとは複数のリフトを動作させるコントローラーを含む全体システムモデルである。
要素オブジェクトは個々のリフトのモデルでり、部分システムモデルを表現する。

投影演算を用いたリフトシステムの仕様を図 4.10のように記述した。輸入を行っている HSLFT は前出の仕様である。
投影演算の定義は$\pi_n : h \rightarrow h_n$である。定義に忠実に従えば
op slft1 : HslftSys → Hslft
op slft2 : HslftSys → Hslft というように２つの演算を定義することになるが、ここでは構成オブジェクトが全く同じのリフトであることから、op slft? : Lft HslftSys → Hslft で代用している。

このように投影演算を利用することで、単純な部分システム単位で仕様を記述し、それを用いて構成を行い複雑なシステムの仕様が得られる。ここではリフト１機で表現された仕様を利用してリフト数機が動作する複雑なシステム仕様が得られることを示した。このことは複雑なシステムの記述を最初から行わず

```
module* HSLFTSYS {
  protecting(HSLFT)
  signature {
    *[ HslftSys ]*
    op initial-ssys : -> HslftSys
    bop slft? : Lft HslftSys -> Hslft       -- projection operator
    bop request-slft : Lft Floor HslftSys -> HslftSys   -- method
    bop request-sfloor : Lft Floor HslftSys -> HslftSys -- method
  }
  axioms {
    var FLOOR : Floor
    vars LFT LFT1 : Lft
    var HSLFTSYS : HslftSys
    beq  slft?(LFT, initial-ssys) = initial-hslft(LFT) .
    bceq slft?(LFT, request-slft(LFT1, FLOOR, HSLFTSYS))
        = request-slft(FLOOR, slft?(LFT, HSLFTSYS)) if LFT == LFT1 .
    bceq slft?(LFT, request-slft(LFT1, FLOOR, HSLFTSYS))
        = slft?(LFT, HSLFTSYS) if LFT =/= LFT1 .
    bceq slft?(LFT, request-sfloor(LFT1, FLOOR, HSLFTSYS))
        = request-sfloor(FLOOR, slft?(LFT, HSLFTSYS)) if LFT == LFT1 .
    bceq slft?(LFT, request-sfloor(LFT1, FLOOR, HSLFTSYS))
        = slft?(LFT, HSLFTSYS) if LFT =/= LFT1 .
  }
}
```

図 4.10: 投影演算を用いた合成リフトシステム

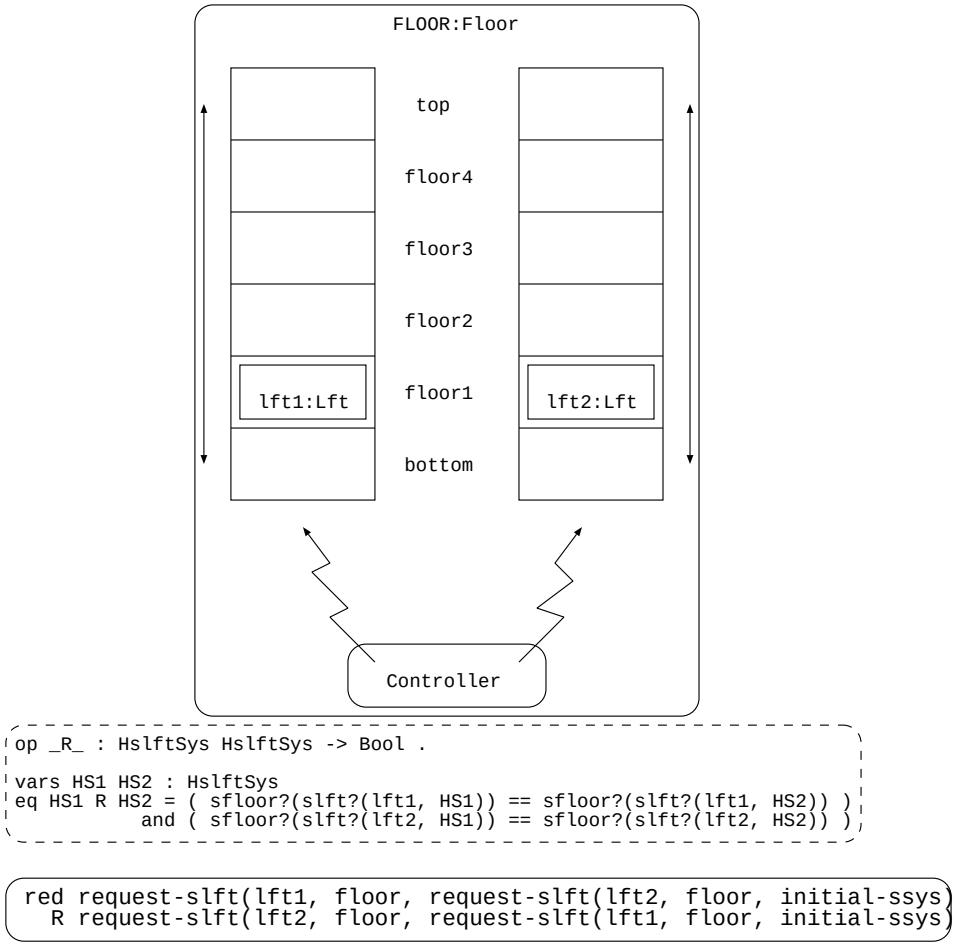


図 4.11: 振る舞いに関する等価な並行システム

に容易に大規模システムの設計が行えることを示しており、CafeOBJ が大規模システム仕様作成に利用できることを示すものである。

4.3.2 並行システムの検証

前節の仕様において個々のリフトに対する操作命令の順序によらず得られるシステム状態が同じであることを証明する。今、隠蔽代数を用いて記述された仕様 HSLFTSYS において状態が等しいことを次のように定義する。

合成システムは２機のリフトからなるマルチリフトシステムであり、その状態は２機のリフトが現在何階に存在するかの情報により決定される。(図 4.11)

そこで合成システム HslftSys が外からの観測により等しい状態であることを投影演算と観測関数を用い次の式が供に成立する時とする。

```
select HSLFTSYS .
open .
op _R_ : HslftSys HslftSys -> Bool .
--> give a candidate hidden congruence relation R .
vars HS1 HS2 : HslftSys .
eq HS1 R HS2 = ( sfloor?(slft?(lft1, HS1)) == sfloor?(slft?(lft1,HS2)) )
       and ( sfloor?(slft?(lft2, HS1)) == sfloor?(slft?(lft2, HS2)) ) .
--> hypothiesis .
ops hs1 hs2 : -> HslftSys .
eq [ hypo1 ]: sfloor?(slft?(lft1, hs1)) = sfloor?(slft?(lft1, hs2)) .
eq [ hypo2 ]: sfloor?(slft?(lft2, hs1)) = sfloor?(slft?(lft2, hs2)) .
-- prove the R is a congruence .
-- ops lft1 lft2 : -> Lft .
op floor : -> Floor .
red request-slft(lft1, floor, hs1) R request-slft(lft1, floor, hs2) .
red request-slft(lft2, floor, hs1) R request-slft(lft2, floor, hs2) .
red request-sfloor(lft1, floor, hs1) R request-sfloor(lft1, floor, hs2) .
red request-sfloor(lft2, floor, hs1) R request-sfloor(lft2, floor, hs2) .
```

図 4.12: 振る舞い等価の証明

sfloor?(slft?(lft1, HS1)) == sfloor?(slft?(lft1, HS2))

sfloor?(slft?(lft2, HS1)) == sfloor?(slft?(lft2, HS2))

ここで等しさを定義している 2 項演算 == は可視なソート Floor の項が字づらで等しい時に true を返す。

最初に先に定義した状態の等しさ R について仕様 HSLFTSYS が振る舞いなシステムであることを証明している (図 4.12)。証明には Coinduction[7][11] を用いた。これらの簡約は true を返し、CafeOBJ の仕様の実行機能により振る舞い等価が証明できた。

次に，得られる合成システムの状態が操作の順序によらないことを証明する。すなわち要素システムに対する演算が並行に適応可能なことを証明する。

これは CafeOBJ の実行機能を用いて図 4.13の簡約を行うことで示すことができる。

以上により操作の並行性が証明できた。

4.3.3 検証の容易さに関する考察

4.1.3 節で隠蔽代数で記述されたシングルリフトシステムを用いマルチリフトシステムの仕様を与え、並行性の検証を行った。また 4.1.1 節では隠蔽代数で記述されたシングルリフトシステムの段階的詳細化を

```
-- reduce in % : request-slft(lft1,floor,request-slft(lft2,floor,
    initial-ssys)) R request-slft(lft2,floor,request-slft(lft1,floor,
    initial-ssys))
true : Bool
(0.033 sec for parse, 36 rewrites(0.033 sec), 102 match attempts)
```

図 4.13: 演算の振る舞いに関して並行実行可能であることの証明

行った。これらで利用したシングルリフトシステムは同じ仕様である。従って先に与えた詳細化仕様は並行性の検証で利用した要素システムの詳細化である。(図 4.14)

詳細化前の Lift システムの任意の状態は隠蔽ソート **Hslft** で構成されており検証は容易であった。しかし詳細化後の Lift システムの任意の状態は可視ソート **Floor,Direct,Door,Floor,Direct** で構成されている。このような仕様をシステム設計初期の段階から作成し、並行性の検証を行うとする。当然、任意の状態に対して操作を与えた結果が等しいことを示さなければならない。この時考慮すべき状態は５つ可視ソートのそれぞれの構成を組み合わせた物である。例えば (floor1,up,open,floor3,down) であり、(floor4,down,close,floor2,up) である。投影演算を用いた詳細化システムでも、これら全てを考慮することは難しいことは明らかである。付録につけた機能を制限していない仕様では状態は次のように９つの集合の組で記述されている。

(FlrSet DirSet DorSet MovSet InSet OutSet LftSet FloorSet DirectSet)

このシステム関して先の並行性の証明を行うことは考慮すべき状態の数が多く不可能である。

このように隠蔽代数を用いることで容易に検証が行えることが事例に基づいて示すことができた。これは隠蔽代数の有効性を示す十分な結果と考えられる。

4.4 並行性に関する考察

代数仕様言語における並行性の議論はいくつも行われている [14]。ここで示した LIFT システムの並行性の記述は極めて簡単な物である。従ってある問題で並行システムが容易に拡張記述できたことを示したのみで、並行性記述をもつその他の仕様記述言語に対した優位性を示した物ではない。Object-Z や Z++ が temporal logic で VDM++が denotic logic で１つのオブジェクトに対するメソッドの起動順序に関する制約を記述できるが、このような制約は記述できず、B がもつ同時演算可能な並行オペレータの記述に対することは表現できない。特に VDM++などは並列動作の制約 [Syncronisationpart] オブジェクトの動的な振舞い [ThreadPart] の記述などがあり、研究の対象として注目すべき物だと考えている。より複雑なシステムや近年重要視されている並行リアルタイムシステムの記述などに対する考慮は CafeOBJ のシス

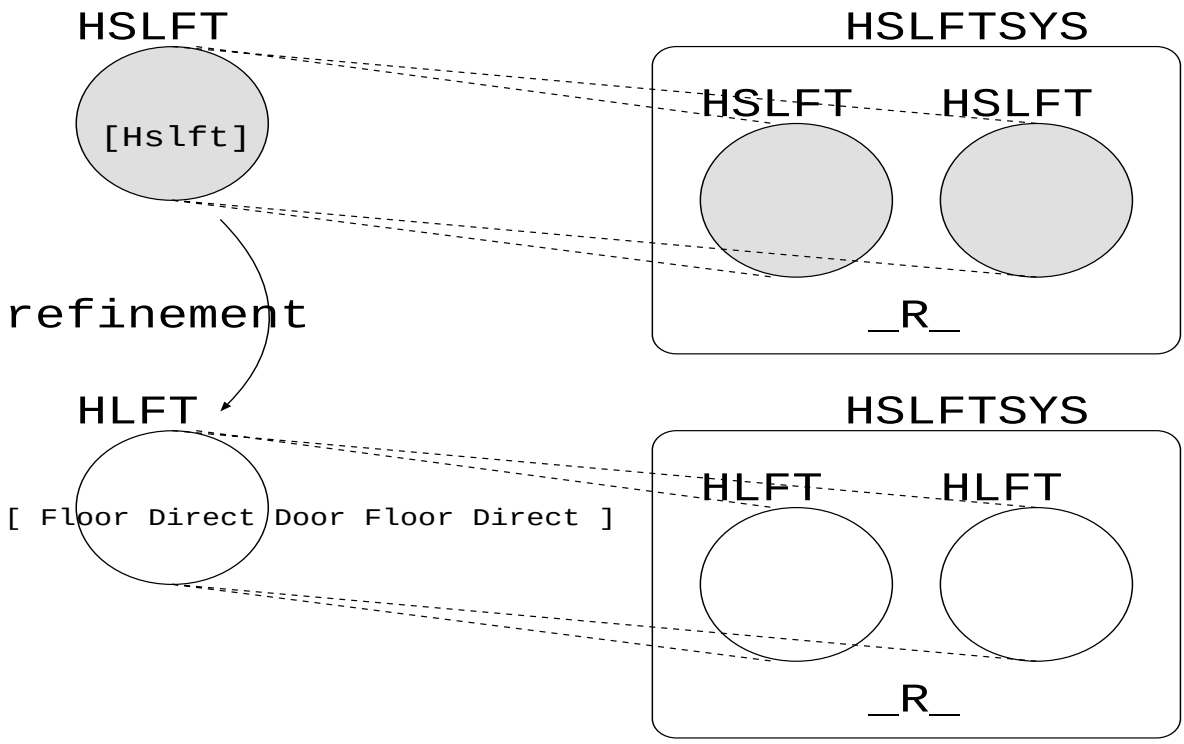


図 4.14: マルチリフトシステムの詳細化

テムに求められる今後の課題といえる。本論文中では仕様として並行性を明示した物でなく、意識せずに記述された仕様から並行性の性質を検証できることに注目している。従ってCSPのように並行性を記述できる言語を意識しなかった。

4.5 まとめ

- B-Technologyの特徴であるCase Studyによる形式仕様言語の学習につかわれるLIFTシステムの事例を3章であたえた指針に基づいてCafe OBJで記述を行った。

- 隠蔽代数を用いてLIFTシステムの一部機能を定義した仕様を与え、詳細化を行った。その際B-Methodで定義されている仕様の詳細化に関する無矛盾性を満たしているかCafeOBJの仕様の実行可能な機能を用いて証明を行った。

- CafeOBJがもつmatchコマンドを用い仕様の所望の性質が満たされているかを確かめ、詳細化したLIFTシステムの洗練を行った。

- 複数のリフトからなるLIFTシステムの仕様を投影演算を用いて記述し、並行システムの仕様が1機のリフトで定義されたLIFTシステムを利用し容易に記述できることを示した。

- 投影演算を用いて記述を行った並行リフトシステムがそれぞれのリフトに対する操作を並行に行えることを証明した。

- 並行性の証明を利用して、隠蔽代数を用いて検証のための必要な抽象レベルを定義し検証を行うことが、考慮すべき状態が多くなるシステムの検証に対して有効であることを示した。
- 並行性の記述に対して他の形式言語との考察を行った。

第 5 章

まとめ

5.1 研究に対する考察とまとめ

本論文では様々な形式手法がある中で B を CafeOBJ でのシステム開発の指針を得るための手本とした。形式言語を理解するためには、それぞれが基礎とする数学概念の知識が必要であることは広くいわれており、このことが教育を十分な時間を割いてできないソフトウェア開発現場で形式手法導入されないひとつの理由になっているのは否定できない。これを踏まえた上で他の形式手法を比べたとき、B がもつオブジェクトをとらえた抽象機械という仕様記述単位が実際に現場で利用されるために大きな意味をもっていることがわかる。また今後もソフトウェアの大規模化は確実に進むといえ、信頼性をもとめられる分野で実際にこれら形式言語が利用されるためには、開発支援環境が充実していることが必要となることは明らかである。すなわち形式言語としてでなく形式手法として比較を行うことが重要であるとい言える。このことから B に着目し開発を進めるの手本とすることの意義は十分である。
本研究は CafeOBJ が基礎とする幾つかの論理のうちで特に隠蔽代数に注目し、設計が有効に行えることを示すものである。CafeOBJ の仕様を事例に基づいて記述した例は少なく、形式仕様言語として様々な事例に対応できる適応力を示すことは必要とされている。また段階的な仕様の詳細化を扱う事例は隠蔽代数を提唱した Goguen らも進めているが数は多くない。本研究で行ったモデル指向の言語を代数で記述した研究は幾つかあるが [16]、モデル指向の言語を用いた開発手法に対する代数仕様の研究を扱う物は少ない。

本研究は最初にモデル指向の B 抽象機械表現が 代数指向の CafeOBJ によりどのように表現することが出来るか対応を考察しテーブルを作成した。CafeOBJ での仕様を作成する指針を得るために B 抽象機械の状態への操作の記述に特に注目した。そして個々の操作の影響範囲の把握が難しい問題を解決する必要を挙げ、解決法として内部状態を表現するソートを用い仕様を記述することを指針として与えた。これに

より操作がもつ性質を検証する際、考慮すべき部分仕様を得るための情報を仕様から容易に得ることができる。
次に B の開発手法である仕様の無矛盾性と詳細化に関する無矛盾性の証明が代数仕様の CafeOBJ でどのように表現されるか考察を行った。そして CafeOBJ の実行機能を用いて無矛盾性の証明を行った。これにより B がもつ開発手法が同様に CafeOBJ で実践できることを示した。
最後に中規模の有名問題であるリフトの仕様を事例として利用し、検証に関する考察を行った。具体的にシングルリフトの事例より隠蔽代数を利用することで振る舞いのみに着目した詳細化前の仕様記述を行い状態遷移に関する性質が保存され詳細化後も同様に振る舞うかを検証した。これにより代数を用いた仕様で、具体的データの詳細化が行えるだけでなくのでなくシステムの振る舞いに関する性質が保存されることを示した。また投影演算と隠蔽代数を用い振る舞いを記述したマルチリフトの事例を考察した。この事例において個々のリフトへの操作が独立に実行可能であることを検証し、振る舞い仕様で記述された仕様が検証を容易にできることを示した。これらにより検証を目的の抽象レベルで行い、求める性質に関与しない他の部分を隠蔽することで、大規模システムの性質の検証において問題になる状態爆発を回避できる可能性を示すことができた。
これら方法論を事例で簡潔に示しせたことで CafeOBJ の実用性を示せた。

5.2 今後の課題

本研究で行った LIFT の事例は同期を持たないシステムであった。しかし現在仕様記述が特に求められるシステムは同期をもつような並行分散システムが挙げられる。そのようなシステム事例において抽象状態を捉え検証が容易に行えるか研究を進めることが重要と考えられる。
また CafeOBJ がもつ書換え論理を用いることで非決定的な振る舞いをもつようなシステム事例の記述を行い、抽象状態を利用し検証が容易に行えるかを考えることを今後の研究の課題としてあげる。

5.3 CafeOBJ にたいする考察

B は他の抽象機械の輸入において被輸入抽象機械上の操作の利用制限ができる。これは仕様記述の段階から実装をある程度意識した物であるる。CafeOBJ は仕様記述言語であり実装を行うものでない。しかし実装に近いイメージを同じ仕様言語上で延長として得ることができれば開発をトータルに支援する形式手法として CafeOBJ を利用できる。そこで仕様と実装のための仕様を明確に区別した上で、関数に対するアクセス制限を記述できる機構を CafeOBJ の機能としてもつことは有効であると考える。

謝辞

本研究を終始ご指導してくださった二木厚吉先生に感謝致します．有益な助言をしてくださった渡部卓雄先生、緒方和博先生、Răzvan Diaconescu 先生に感謝致します。また，研究に関する議論につきあっていただいた言語設計学講座の皆様にお礼を申し上げます．

参考文献

[1] Ataru. T. Nakagawa, Toshimi Sawada, and Kokichi Futatsugi. CafeOBJ user's manual, 1997. http://ldl.jaist.ac.jp/cafeobj/.

[2] B.Potter, J.Sinclair, and D.Till 田中武二監訳An Introduction to Formal Specification and Z ソフトウェア仕様記述の先進技法-Z 言語Prentice-Hall, 1990 トッパン,1993

[3] C.H.Pratten An Introduction To Proving AMN Specifications With PVS And The AMN-PROOF Tool In Henri HABRIAS, editor, Proc. Z Twenty Years on - What is its Future, pages 149-165. IRIN-IUT de Nantes, October 1995.

[4] C.H.Pratten Proving AMN Specifications With The HOL Theorem Prover May 1995.

[5] Hubert Baumeister Using Algebraic Specification Language for Model-Oriented Specifications Technical Report MIP-I-96-2-003, Max-Planck-Institut fuer Informatik, Feb 1996

[6] J.-R Abrial The B Book a Assigning Programs to Meanings. Cambridge University Press, August 1996.

[7] Joseph A. Goguen and Grant Malcolm. A Hidden Agenda. UCSD Technical Report, CS97-538. (1997)

[8] Joseph A. Goguen. Theorem Proving and Algebra MIT Press, Cambridge, Massachusetts, 1996.?????

[9] Joseph A. Goguen and Răzvan Diaconescu. Towards an Algebraic Semantics for the Object Paradigm. In Harmut Ehrig and Fernando Orejas, editors, Recent Trends in Data Type Specification, volume 785 of Lecture Notes in Computer Science. Springer, 1994.

[10] Kevin Lano. The B Language and Method. Springer Verlag London Ltd., 1996.

[11] Răzvan Diaconescu, Kokichi Futatsugi CafeOBJ Report AMAST Series in Computing, World Scientific, to appear. (1998)

[12] Shusaku Iida, Michihiro Matsumoto, Răzvan Diaconescu, Kokichi Futatsugi, and Dorel Lucanu Concurrent Object Composition in CafeOBJ JAIST Rearch Report, to appear. (1998)

[13] 飯田 周作, Răzvan Diaconescu, 二木 厚吉代数仕様におけるオブジェクトの合成日本ソフトウェア科学会第 14 回大会論文集.(1997) 521-524.

[14] 飯田 周作 並行オブジェクト指向モデルに基づく並行分散システムの形式仕様作成法 北陸先端科学技術大学院大学修士論文 (1996)

[15] 飯田 周作, 二木 厚吉, 渡部 卓雄 実行可能な代数仕様言語による仕様の検証 日本ソフトウェア科学会第 13 回大会論文集.(1996) 61-64

[16] 梅原 伸年, 二木 厚吉 CafeOBJ による B オブジェクトモデルの表現 電気関係学会 北陸支部連合大会 1997

[17] 海野 浩 形式仕様のライブラリに関する研究 北陸先端科学技術大学院大学修士論文 (1996)

[18] 海野 浩, 二木 厚吉 モジュールの総称性を活用した代数仕様言語 CafeOBJ のライブラリの設計 日本ソフトウェア科学会第 13 回大会論文集.(1996) 69-72

[19] 杉山 智倫, 二木 厚吉 形式仕様の記述スタイルに関する考察 ソフトウェアシンポジウム 1997

[20] 中川 中, 谷津 弘一, 本間 毅寛 CafeOBJ への誘い Technical report, IPA (1996).

[21] 中島 震, 登内 敏夫 OBJ を用いた実行可能なオブジェクト指向仕様 日本ソフトウェア科学会第 13 回大会論文集.(1996) 61-64

[22] 浜口 正孝 形式仕様言語 OBJ によりオブジェクト指向仕様記述 北陸先端科学技術大学院大学修士論文 (1995)

[23] 二木 厚吉 代数モデルの基礎 コンピュータソフトウェア Vol.13, No.1,pp.3-22(1996)

[24] 松本 充広, 二木 厚吉 振舞意味論に基づく仕様の検証法 日本ソフトウェア科学会第 14 回大会論文集.(1997) 473-476

第 A 章

事例 LIFT

A.1 事例 LIFT の B-抽象機械仕様

この仕様書は B-Toolkit により自動的に作成できる。

MACHINE

$Lift\ (\ LFT\ ,\ topfloor\ ,\ bottomfloor\ )$

CONSTRAINTS

$topfloor > bottomfloor$

SETS

$DIR = \{\ up\ ,\ dn\ \}$;

$DOR = \{\ opn\ ,\ clo\ \}$

CONSTANTS

opp

PROPERTIES

$opp \in DIR \rightarrow DIR \wedge$

$opp = \{\ up \mapsto dn\ ,\ dn \mapsto up\ \} \wedge$

$opp\ (\ up\ ) = dn \wedge$

$opp\ (\ dn\ ) = up$

VARIABLES

$in\ ,\ out\ ,\ mov\ ,\ dir\ ,\ flr\ ,\ dor$

INVARIANT

$in \in bottomfloor\ ..\ topfloor \leftrightarrow DIR\ \wedge$

$out \in LFT \leftrightarrow bottomfloor\ ..\ topfloor\ \wedge$

$mov \subseteq LFT\ \wedge$

$dir \in LFT \rightarrow DIR\ \wedge$

$flr \in LFT \rightarrow bottomfloor\ ..\ topfloor\ \wedge$

$dor \in LFT \rightarrow DOR\ \wedge$

$dor\ [\ mov\ ] \subseteq \{\ clo\ \}$

INITIALISATION

$in := \quad \|$

$out := \quad \|$

$mov := \quad \|$

$dir := LFT \times \{\ up\ \}\ \|$

$flr := LFT \times \{\ bottomfloor\ \}\ \|$

$dor := LFT \times \{\ clo\ \}$

OPERATIONS

Request_Lift

$Request\_Lift(\ fl\ ,\ dd\ ) \quad \widehat{=}$

PRE

$fl \in bottomfloor\ ..\ topfloor\ \wedge$

$dd \in DIR$

THEN

$in := in \cup \{ fl \mapsto dd \}$

END ;

Request_Floor

$Request\_Floor(\ ll\ ,\ fl\ ) \quad \widehat{=}$

PRE

$fl \in bottomfloor\ ..\ topfloor \wedge$

$ll \in LFT$

THEN

$out := out \cup \{ ll \mapsto fl \}$

END ;

Continue_Up

$Continue\_Up(\ ll\ ) \quad \widehat{=}$

PRE

$ll \in mov \wedge$

$dir\ (\ ll\ ) = up \wedge$

$flr\ (\ ll\ ) < topfloor \wedge$

$ll \mapsto flr\ (\ ll\ ) \notin out \wedge$

$flr\ (\ ll\ ) \mapsto up \notin in \wedge$

$attr\_up\ (\ ll\ )$

THEN

$flr\ (\ ll\ ) := flr\ (\ ll\ ) + 1$

END ;

$Continue\_Down(\ ll\ ) \quad \widehat{=}$

PRE

$ll \in mov \wedge$

$dir\ (\ ll\ ) = dn \wedge$

$flr\ (\ ll\ ) > bottomfloor \wedge$

$ll \mapsto flr\ (\ ll\ ) \notin out \wedge$

$flr\ (\ ll\ ) \mapsto dn \notin in \wedge$

$attr\_dn\ (\ ll\ )$

THEN

$flr\ (\ ll\ ) := flr\ (\ ll\ ) - 1$

END ;

Pickup_Or_Deliver_And_Continue

$Pickup\_Or\_Deliver\_And\_Continue(\ ll\ ) \quad \widehat{=}$

PRE

$ll \in mov \wedge$

$flr\ (\ ll\ ) \in in^{-1}\ [\ \{\ dir\ (\ ll\ )\ \}\ ] \cup out\ [\ \{\ ll\ \}\ ] \wedge$

$(\ dir\ (\ ll\ ) = up \Rightarrow$

$attr\_up\ (\ ll\ )\ ) \wedge$

$(\ dir\ (\ ll\ ) = dn \Rightarrow$

$attr\_dn\ (\ ll\ )\ )$

THEN

$mov := mov - \{\ ll\ \} \ \|$

$dor\ (\ ll\ ) := opn \ \|$

$out := out - \{\ ll \mapsto flr\ (\ ll\ )\ \} \ \|$

$in := in - \{\ flr\ (\ ll\ ) \mapsto dir\ (\ ll\ )\ \}$

END ;

Stop

$Stop(\ ll\ ) \quad \widehat{=}$

PRE

$ll \in mov \wedge$

$(\ dir \ (\ ll \ ) = up \Rightarrow \neg \ (\ attr\_up \ (\ ll \ ) \ ) \ ) \wedge$

$(\ dir \ (\ ll \ ) = dn \Rightarrow \neg \ (\ attr\_dn \ (\ ll \ ) \ ) \ ) \wedge$

$flr \ (\ ll \ ) \notin \mathsf{dom} \ (\ in \ )$

THEN

$mov := mov - \{ \ ll \ \} \ \|$

$dor \ (\ ll \ ) := opn \ \|$

$out := out - \{ \ ll \mapsto flr \ (\ ll \ ) \ \}$

END ;

$Pickup\_And\_Change\_Dir(\ ll \ ) \ \widehat{=}$

PRE

$ll \in mov \wedge$

$(\ dir \ (\ ll \ ) = up \Rightarrow$

$\neg \ (\ attr\_up \ (\ ll \ ) \ ) \ ) \wedge$

$(\ dir \ (\ ll \ ) = dn \Rightarrow$

$\neg \ (\ attr\_dn \ (\ ll \ ) \ ) \ ) \wedge$

$flr \ (\ ll \ ) \mapsto dir \ (\ ll \ ) \notin in \wedge$

$flr \ (\ ll \ ) \mapsto opp \ (\ dir \ (\ ll \ ) \ ) \in in$

THEN

$mov := mov - \{ \ ll \ \} \ \|$

$dor \ (\ ll \ ) := opn \ \|$

$out := out - \{ \ ll \mapsto flr \ (\ ll \ ) \ \} \ \|$

$in := in - \{ \ flr \ (\ ll \ ) \mapsto opp \ (\ dir \ (\ ll \ ) \ ) \ \} \ \|$

$dir \ (\ ll \ ) := opp \ (\ dir \ (\ ll \ ) \ )$

END ;

Pickup_And_Same_Dir

$Pickup\_And\_Same\_Dir(\ ll\ ) \quad \widehat{=}$

PRE

$ll \in mov \wedge$

$(\ dir\ (\ ll\ ) = up \Rightarrow$

$\neg\ (\ attr\_up\ (\ ll\ )\ )\ ) \wedge$

$(\ dir\ (\ ll\ ) = dn \Rightarrow$

$\neg\ (\ attr\_dn\ (\ ll\ )\ )\ ) \wedge$

$flr\ (\ ll\ ) \mapsto dir\ (\ ll\ ) \in in$

THEN

$mov := mov - \{\ ll\ \} \ \|$

$dor\ (\ ll\ ) := opn \ \|$

$out := out - \{\ ll \mapsto flr\ (\ ll\ )\ \} \ \|$

$in := in - \{\ flr\ (\ ll\ ) \mapsto dir\ (\ ll\ )\ \}$

END ;

$Close\_Door \quad \widehat{=}$

PRE

$dor^{-1}\ [\ \{\ opn\ \}\ ] \neq$

THEN

ANY ll **WHERE**

$ll \in dor^{-1}\ [\ \{\ opn\ \}\ ]$

THEN

$dor\ (\ ll\ ) := clo$

END

END ;

Open_Door

$Open\_Door \quad \widehat{=}$

PRE

$dor^{-1}[\{clo\}] \cap LFT - mov \neq$

THEN

ANY ll **WHERE**

$ll \in dor^{-1}[\{clo\}] \wedge$

$ll \notin mov$

THEN

$dor(ll) := opn \parallel$

$in := in - \{flr(ll) \mapsto dir(ll)\}$

END

END ;

Depart_And_Move_Up

$Depart\_And\_Move\_Up(ll) \quad \widehat{=}$

PRE

$ll \in dor^{-1}[\{clo\}] \wedge$

$ll \notin mov \wedge$

$attr\_up(ll)$

THEN

$mov := mov \cup \{ll\} \parallel$

$flr(ll) := flr(ll) + 1 \parallel$

$dir(ll) := up$

END ;

$Depart\_And\_Move\_Down(ll) \quad \widehat{=}$

PRE

$ll \in dor^{-1}[\{clo\}] \wedge$

$ll \notin mov \wedge$

$attr\_dn\ (\ ll\ )$

THEN

$mov := mov \cup \{\ ll\ \} \ \|$

$flr\ (\ ll\ ) := flr\ (\ ll\ ) - 1 \ \|$

$dir\ (\ ll\ ) := dn$

END

DEFINITIONS

$attr\_up\ (\ l\ ) \ \widehat{=} \ out\ [\ \{\ l\ \}\ ] \cup \mathsf{dom}\ (\ in\ ) \cap flr\ (\ l\ ) + 1\ ..\ topfloor \neq$;

$attr\_dn\ (\ l\ ) \ \widehat{=} \ out\ [\ \{\ l\ \}\ ] \cup \mathsf{dom}\ (\ in\ ) \cap bottomfloor\ ..\ flr\ (\ l\ ) - 1 \neq$

END

A.2 事例 LIFT の CafeOBJ 仕様

LIFT システム全体に対する CafeOBJ の LIFT システムの仕様である。
B の Abstract Machine Notation は集合や２項関係の演算子を持っているが、CafeOBJ には存在しないので別に記述し輸入している。事例の理解のために階数を表す Floor は明示的なものを利用している。

```
Lift example multi lift version
-- require in-out-set3 ./in-out-set3.mod

module LIFT3 {
--  protecting(MOV-SET)
 protecting(LFT-SET)
 protecting(DIR-SET)
 protecting(FLR-SET)
 protecting(DOR-SET)
 protecting(IN-OUT-SET)
 protecting(FLOOR-SET)
 protecting(DIRECT-SET)

signature {
 [ State ]
-- LIFT システム全体 State で表現され、以下のような９つの内部状態をもつ
 op state : FlrSet DirSet DorSet MovSet InSet OutSet LftSet FloorSet DirectSet -> State

-- operations 状態変化を起こす１２個の操作 --
 op request-lift : Floor Direct State -> State
 op request-floor : Lft Floor State -> State
 op continue-up : Lft State -> State
 op continue-down : Lft State -> State
 op pickup-or-deliver-and-continue : Lft State -> State
 op stop : Lft State -> State
 op pickup-and-change-dir : Lft State -> State
 op pickup-and-same-dir : Lft State -> State
 op close-door : State -> State
 op open-door : State -> State
 op depart-and-move-up : Lft State -> State
 op depart-and-move-down : Lft State -> State
-- support operations 補助関数 --
 op attr-up : Lft Floor InSet OutSet -> Bool
 op attr-dn : Lft Floor InSet OutSet -> Bool
 op sub-open-door : DorSet MovSet -> DorSet
}
```

```
axioms {
 vars FLOOR FLOOR1 FLOOR2 : Floor
 var FLRSET : FlrSet
 vars DIRECT DIRECT1 DIRECT2 : Direct
 var DIRSET : DirSet
 vars DOOR DOOR1 DOOR2 : Door
 var DOR : Dor
 var DORSEQ : DorSeq
 var DORSET : DorSet
 vars LFT LFT1 LFT2 : Lft
 var MOVSET : MovSet
 var INSET : InSet
 var OUTSET : OutSet
 var LFTSET : LftSet
 var FLOORSET : FloorSet
 var DIRECTSET : DirectSet

 ceq request-lift(FLOOR1, DIRECT1,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
  =
state(FLRSET, DIRSET, DORSET, MOVSET, ({FLOOR1 in DIRECT1} U INSET), OUTSET, LFTSET, FLOORSET, DIRECTSET)
  if (FLOOR1 within FLOORSET )
      and (DIRECT1 within DIRECTSET) .

 ceq request-floor(LFT, FLOOR,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
   =
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, ( {LFT out FLOOR} U OUTSET ), LFTSET, FLOORSET, DIRECTSET )
   if (FLOOR within FLOORSET)
      and (LFT within LFTSET) .

 ceq continue-up(LFT1,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 =
state(up-floor-about-lft(LFT1,FLRSET), DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET)
   if (LFT1 within MOVSET)
      and (direct-from-lft-in-dirset(LFT1,DIRSET) == up)
      and ((floor-from-lft-in-flrset(LFT1,FLRSET)) =/= top )
      and not( (LFT1 out floor-from-lft-in-flrset(LFT1,FLRSET))  within OUTSET )
      and not( (floor-from-lft-in-flrset(LFT1,FLRSET) in up) within INSET )
      and attr-up(LFT1, floor-from-lft-in-flrset(LFT1,FLRSET), INSET, OUTSET) .

 eq attr-up(LFT, top, INSET, OUTSET)  = false .
 eq attr-up(LFT, FLOOR, INSET, OUTSET)
 =  floor-in-outset?(upper(FLOOR), OUTSET)
    or floor-in-inset?(upper(FLOOR), INSET)
```

```
    or attr-up(LFT, upper(FLOOR), INSET, OUTSET) .

ceq continue-down(LFT1,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 =
state(down-floor-about-lft(LFT1,FLRSET), DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET)
 if (LFT1 within MOVSET)
 and (direct-from-lft-in-dirset(LFT1,DIRSET) == down)
 and ((floor-from-lft-in-flrset(LFT1,FLRSET)) =/= bottom )
 and not( (LFT1 out floor-from-lft-in-flrset(LFT1,FLRSET))  within OUTSET )
 and not( (floor-from-lft-in-flrset(LFT1,FLRSET) in down) within INSET )
 and attr-dn(LFT1, floor-from-lft-in-flrset(LFT1,FLRSET), INSET, OUTSET) .

 eq attr-dn(LFT, bottom, INSET, OUTSET)  = false .
 eq attr-dn(LFT, FLOOR, INSET, OUTSET)
 =  floor-in-outset?(lower(FLOOR), OUTSET)
    or floor-in-inset?(lower(FLOOR), INSET)
    or attr-dn(LFT, lower(FLOOR), INSET, OUTSET) .

 ceq pickup-or-deliver-and-continue(LFT,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 =
state(FLRSET, DIRSET, open-door-about-lft(LFT,DORSET), (MOVSET \ LFT),
 ( INSET \ (floor-from-lft-in-flrset(LFT,FLRSET) in  direct-from-lft-in-dirset(LFT, DIRSET) ) ),
 ( OUTSET \ (LFT out floor-from-lft-in-flrset(LFT,FLRSET) ) ) , LFTSET, FLOORSET, DIRECTSET)
 if (LFT within MOVSET )
    and ( floor-in-outset?(floor-from-lft-in-flrset(LFT,FLRSET), OUTSET)
     or ((floor-from-lft-in-flrset(LFT,FLRSET) in direct-from-lft-in-dirset(LFT, DIRSET) ) within INSET) )
    and ((if (direct-from-lft-in-dirset(LFT, DIRSET) == up)
          then attr-up(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET)
          else attr-dn(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET) fi)
 == true) .

 ceq stop(LFT, state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
  = state(FLRSET, DIRSET, open-door-about-lft(LFT,DORSET), (MOVSET \ LFT), INSET,
         ( OUTSET \ (LFT out floor-from-lft-in-flrset(LFT,FLRSET) )) , LFTSET, FLOORSET, DIRECTSET)
  if (LFT within MOVSET )
     and ((if (direct-from-lft-in-dirset(LFT, DIRSET) == up)
            then not(attr-up(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET))
            else not(attr-dn(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET)) fi)
           == true)
     and not(floor-in-inset?(floor-from-lft-in-flrset(LFT,FLRSET),INSET)) .

 ceq pickup-and-change-dir(LFT,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 =
```

```
state(FLRSET, opp-direct-about-lft(LFT, DIRSET), open-door-about-lft(LFT,DORSET), (MOVSET \ LFT),
     ( INSET \ (floor-from-lft-in-flrset(LFT,FLRSET) in  opp(direct-from-lft-in-dirset(LFT, DIRSET)) ) ),
     ( OUTSET \ (LFT out floor-from-lft-in-flrset(LFT,FLRSET) ) ), LFTSET, FLOORSET, DIRECTSET )
 if LFT within MOVSET
    and ((if (direct-from-lft-in-dirset(LFT, DIRSET) == up)
           then not(attr-up(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET))
           else not(attr-dn(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET)) fi)
          == true)
    and not( (floor-from-lft-in-flrset(LFT,FLRSET) in direct-from-lft-in-dirset(LFT, DIRSET)) within INSET )
    and (floor-from-lft-in-flrset(LFT,FLRSET) in opp(direct-from-lft-in-dirset(LFT, DIRSET)) ) within INSET .

ceq pickup-and-same-dir(LFT,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 = state(FLRSET, DIRSET, open-door-about-lft(LFT,DORSET), (MOVSET \ LFT),
   ( INSET \ (floor-from-lft-in-flrset(LFT,FLRSET) in  direct-from-lft-in-dirset(LFT, DIRSET) ) ),
   ( OUTSET \ (LFT out floor-from-lft-in-flrset(LFT,FLRSET) ) ), LFTSET, FLOORSET, DIRECTSET )
   if LFT within MOVSET
      and ((if (direct-from-lft-in-dirset(LFT, DIRSET) == up)
            then not(attr-up(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET))
            else not(attr-dn(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET)) fi)
            == true)
      and (floor-from-lft-in-flrset(LFT,FLRSET) in direct-from-lft-in-dirset(LFT, DIRSET)) within INSET .

 ceq close-door(
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 =
state(FLRSET, DIRSET, close-all-open-door(DORSET), MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET)
 if (get-lftset-from-door-in-dorset(open,DORSET) =/= {} ) .

 ceq open-door(
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 =
state(FLRSET, DIRSET, sub-open-door(DORSET, MOVSET), MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET)
   if ( get-lftset-from-door-in-dorset(open,DORSET) - (LFTSET - MOVSET) ) =/= {} .

 eq sub-open-door({},MOVSET) = {} .
 eq sub-open-door({DOR, DORSEQ}, MOVSET)
   = if (dor-door?(DOR) == close
       and not(dor-lft?(DOR) within MOVSET) )
              then open-door-about-lft(dor-lft?(DOR), DORSET) U sub-open-door({DORSEQ}, MOVSET)
              else {DOR} U sub-open-door({DORSEQ}, MOVSET) fi .

 ceq depart-and-move-up(LFT,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 = state(up-floor-about-lft(LFT,FLRSET),
   up-direct-about-lft(LFT, DIRSET), DORSET, MOVSET U {LFT}, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET)
```

```
 if (door-from-lft-in-dorset(LFT,DORSET) == close)
    and not(LFT within MOVSET)
    and attr-up(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET) .

 ceq depart-and-move-down(LFT,
state(FLRSET, DIRSET, DORSET, MOVSET, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET) )
 = state(down-floor-about-lft(LFT,FLRSET),
   down-direct-about-lft(LFT, DIRSET), DORSET, MOVSET U {LFT}, INSET, OUTSET, LFTSET, FLOORSET, DIRECTSET)
   if (door-from-lft-in-dorset(LFT,DORSET) == close)
      and not(LFT within MOVSET)
      and attr-dn(LFT, floor-from-lft-in-flrset(LFT,FLRSET), INSET, OUTSET) .
  }
}
```

A.3 事例 LIFT の CafeOBJ 仕様のシミュレーション

B Toolkit のアニメーション機能は B 抽象機械仕様を実行可能な物とする。事例 LIFT は Case Study として挙げられているもので、その実行例としてある人が４階から３階におりる事象を示している。このことを CafeOBJ の仕様の実行機能を用いてシミュレートした。

```
caraway: {1} % cafeobj
-- loading standard prelude
Loading /usr/local/cafeobj-1.4/prelude/std.bin
Finished loading /usr/local/cafeobj-1.4/prelude/std.bin

           -- CafeOBJ system Version 1.4.0(Beta-5) --
                built: 1997 Nov 18 Tue 8:55:11 GMT
                      prelude file: std.bin
                               ***
                     1998 Feb 8 Sun 15:41:49 GMT
                           Type ? for help
                               ---
                     uses GCL (GNU Common Lisp)
            Licensed under GNU Public Library License
              Contains Enhancements by W. Schelter

CafeOBJ> in lift3.mod
-- processing input : /usr/home/nume/master/lift3.mod
-- processing input : /usr/home/nume/master/in-out-set3.mod
-- defining module! MY-SET_*_*.....__._......_.............* done.
-- defining module TRIV2.._* done.
-- defining module RELATION_*_*........_..* done.
-- defining module LFT....._* done.
-- defining module MOVE._* done.
-- defining module! FLOOR.............._.........* done.
-- defining module FLR.._* done.
-- defining module FLR-REL,,,,,,_,,,,,,,,,_,,*._* done.
-- defining module FLR-SET,,,,,,_,,,,,,,,,_,,*..........._.........* done.
-- defining module DIRECT....._..* done.
-- defining module DIR.._* done.
-- defining module DIR-REL,,,,,,_,,,,,,,,,_,,*._* done.
-- defining module DOOR...._* done.
-- defining module DOR.._* done.
-- defining module DOR-REL,,,,,,_,,,,,,,,,_,,*._* done.
-- defining module OUT.._* done.
-- defining module OUT-REL,,,,,,_,,,,,,,,,_,,*._* done.
-- defining module IN.._* done.
-- defining module IN-REL,,,,,,_,,,,,,,,,_,,*._* done.
-- defining module LFT-SET,,,,,,_,,,,,,,,,_,,*.._* done.
-- defining module DOR-SET.,,,,,,_,,,,,,,,,_,,*............._.
```

```
-- defining module FLOOR-SET,,,,,,_,,,,,,,,,,_,,*._* done.
-- defining module DIRECT-SET,,,,,,_,,,,,,,,,,_,,*._* done.
-- defining module DIR-SET,,,,,,_,,,,,,,,,,_,,*............_.............* done.
-- defining module IN-OUT-SET,,,,,,_,,,,,,,,,,_,,*.,,,,,,_,,,,,,,,,,_,,*.............._........* done.
-- defining module LIFT3......................................_.

-- reduce in LIFT3 :
stop(lft1,
 depart-and-move-down(lft1,
  request-floor(lft1,floor3,
   close-door(
    pickup-and-change-dir(lft1,
     continue-up(lft1,
      continue-up(lft1,
       depart-and-move-up(lft1,
        request-lift(floor4,down,
state( -- これが初期状態を表現したもの
       { ((lft1 flr floor1) , (lft2 flr floor1)) },
       { ((lft1 dir up) , (lft2 dir up)) },
       { ((lft1 dor close) , (lft2 dor close)) },
       { },{ },{ },
       { (lft1 , lft2) },
       { (bottom, floor1, floor2, floor3, floor4,top) },
       { (up , down) }
      ))))))))))

state( -- 状態遷移後の結果状態
       { ((lft1 flr floor3) , (lft2 flr floor1)) },
       { ((lft1 dir down), (lft2 dir up)) },
       { ((lft1 dor open) , (lft2 dor close)) },
       { },{ },{ },
       { (lft1 , lft2) },
       { (bottom , floor1 , floor2 , floor3   , floor4 , top) },
       { (up , down) }
      ) : State
(70.656 sec for parse, 591 rewrites(0.133 sec), 1115 match attempts)

LIFT3>
```

A.4 事例 LIFT の詳細化の検証スコア

LIFT システムの検証作業において等式を書換え規則とみなしているが、LIFT の仕様で右辺に if 文を利用している。従って等式を右辺から左辺への書換え規則とみなして利用することはない。

```
-- -- proof section -- ---
-- 詳細化前モデルの性質 (axiom) の証明1
--    eq sfloor?(initial-hslft(LFT)) = floor1 .
open .
-- lft には依存しないので全称の意味で定数を定義
op lft : -> Lft .
--> VVVVV shuld be true
red floor?(initial-hlft(lft)) == floor1 .
close .

--> 詳細化前モデルの性質 (axiom) の証明2
--    eq sfloor?(request-slft(FLOOR, HSLFT)) = FLOOR .
-- for proof 階下にリフトがある状態での呼出についての証明
open .
ops floor upper-floor : -> Floor .
ops direct1 direct2 : -> Direct .
op door : -> Door .
vars FLOOR UFLOOR FLOOR1 : Floor .
vars DIRECT1 DIRECT2 : Direct .
vars DOOR : Door .
op request-lft2 : Floor Direct Hlft -> Hlft .

-- 詳細化前モデルの request-lft は詳細化モデルにおいて一つ以上の操作から構成され
-- その構成は場合により異なる。例えばリフトが一階にある時4階からと2階から呼ぶのでは
-- continue-up 関数の適応回数が異なる。そこで continue-up に関して帰納法で証明する。

-- proof 1
-- continue-up 関数が一回含まれる時。すなわち一階上からリフトを呼ぶ時。
--> VVVV Base 1 should be true VVVV

op floor-tmp : -> Floor .
eq request-lft2(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(
      continue-up(
       request-lft(UFLOOR, DIRECT2,
        close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))))) .
red floor?(request-lft2(upper(floor-tmp), direct2,
            state(floor-tmp, direct1, open, floor-tmp, direct1))) == upper(floor-tmp) .

--> VVVV Base n induction hypothesis be true VVVV
```

```
op upper-n : Floor -> Floor .
op continue-n-up : Hlft -> Hlft .
eq continue-n-up(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
   = state(upper-n(FLOOR1), up, close, FLOOR, DIRECT) .
eq request-lft2(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(
      continue-n-up(
       request-lft(UFLOOR, DIRECT2,
        close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))))) .
eq floor?(request-lft2(upper-n(FLOOR1), DIRECT2,
           state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))) = upper-n(FLOOR1) .

--> VVVV Base n+1 should be true VVV
op request-lft3 : Floor Direct Hlft -> Hlft .
eq request-lft3(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(
      continue-up(
       continue-n-up(
        request-lft(UFLOOR, DIRECT2,
         close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1)))))) .
red floor?(request-lft3(upper(upper-n(floor-tmp)), direct2,
  state(floor-tmp, direct1, open, floor-tmp, direct1))) == upper(upper-n(floor-tmp)) .

-- for proof 階上にリフトがある状態での呼出
--> VVVV Base 1 should be true VVVV
op request-lft5 : Floor Direct Hlft -> Hlft .
eq request-lft5(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(
      continue-down(
       request-lft(UFLOOR, DIRECT2,
        close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))))) .
red floor?(request-lft5(lower(floor-tmp), direct2,
  state(floor-tmp, direct1, open, floor-tmp, direct1))) == lower(floor-tmp) .
--> VVVV Base n should be true VVVV
op lower-n : Floor -> Floor .
op continue-n-down : Hlft -> Hlft .
eq continue-n-down(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
   = state(lower-n(FLOOR1), down, close, FLOOR, DIRECT) .
eq request-lft5(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(
      continue-n-down(
       request-lft(UFLOOR, DIRECT2,
        close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))))) .
eq floor?(request-lft5(lower-n(FLOOR1), DIRECT2,
    state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))) = lower-n(FLOOR1) .
--> VVVV Base n+1 should be true VVVV
```

```
op request-lft6 : Floor Direct Hlft -> Hlft .
eq request-lft6(UFLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(
      continue-down(
       continue-n-down(
        request-lft(UFLOOR, DIRECT2,
         close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1)))))) .
red floor?(request-lft6(lower(lower-n(floor-tmp)), direct2,
   state(floor-tmp, direct1, open, floor-tmp, direct1))) == lower(lower-n(floor-tmp)) .

-- for proof 現在の階にリフトがある状態での呼出
op request-lft4 : Floor Direct Hlft -> Hlft .
eq request-lft4(FLOOR, DIRECT2, state(FLOOR, DIRECT1, open, FLOOR, DIRECT1))
   = open-door(request-lft(FLOOR, DIRECT2,
      close-door(state(FLOOR, DIRECT1, open, FLOOR, DIRECT1)))) .
red floor?(request-lft4(floor-tmp, direct2,
            state(floor-tmp, direct1, open, floor-tmp, direct1))) == floor-tmp .

close .

--> 詳細化前モデルの性質 (axiom) の証明３
--    eq sfloor?(request-sfloor(FLOOR, HSLFT)) = FLOOR .
-- for proof リフトがある状態から階下に降りたい時の証明
open .
ops floor floor1 : -> Floor .
ops direct direct1 : -> Direct .
vars FLOOR FLOOR1 : Floor .
vars DIRECT DIRECT1 : Direct .
vars DOOR : Door .

--> Base 1 VVVVV shuld be true
op request-floor2 : Floor Hlft -> Hlft .
eq request-floor2(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(
     continue-down(
      request-floor(FLOOR,
       close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))))) .
red floor?(request-floor2(lower(floor),
       state(floor, direct, open, floor, direct1))) == lower(floor) .

-- proof n continue-down 関数が n 回含まれる時。すなわち n 階下にリフトで降りる時
--> Base n VVVVV be true
op lower-n : Floor -> Floor .
op continue-n-down : Hlft -> Hlft .
eq continue-n-down(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
   = state(lower-n(FLOOR1), down, close, FLOOR, DIRECT) .
```

```
eq request-floor2(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(
     continue-n-down(
      request-floor(FLOOR,
       close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))))) .
-- induction hypothises
eq floor?(request-floor2(lower-n(FLOOR),
           state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))) = lower-n(FLOOR) .

-- proof n continue-down 関数が n+1 回含まれる時。すなわち n+1 階下にリフトで降りる時
--> Base n+1 VVVVV shuld be true
op request-floor3 : Floor Hlft -> Hlft .
eq request-floor3(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(
     continue-down(
      continue-n-down(
       request-floor(FLOOR,
        close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT)))))) .
red floor?(request-floor3(lower(lower-n(floor)),
    state(floor, direct, open, floor, direct1))) == lower(lower-n(floor)) .

--> 同様に上がりたい時も行う。
--> Base 1 VVVVV shuld be true
op request-floor4 : Floor Hlft -> Hlft .
eq request-floor4(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(
     continue-up(
      request-floor(FLOOR,
       close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))))) .
red floor?(request-floor4(upper(floor),
       state(floor, direct, open, floor, direct1))) == upper(floor) .

-- proof n continue-down 関数が n 回含まれる時。すなわち n 階上にリフトであがる時
--> Base n VVVVV be true
op upper-n : Floor -> Floor .
op continue-n-up : Hlft -> Hlft .
eq continue-n-up(state(FLOOR1, DIRECT1, close, FLOOR, DIRECT))
   = state(upper-n(FLOOR1), up, close, FLOOR, DIRECT) .
op request-floor5 : Floor Hlft -> Hlft .
eq request-floor5(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(
     continue-n-up(
      request-floor(FLOOR,
       close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))))) .
-- induction hypothises
eq floor?(request-floor5(upper-n(FLOOR),
```

```
    state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))) = upper-n(FLOOR) .

-- proof n continue-down 関数が n+1 回含まれる時。すなわち n+1 階上にリフトであがる時
--> Base n+1 VVVVV shuld be true
op request-floor6 : Floor Hlft -> Hlft .
eq request-floor6(FLOOR, state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT))
  = open-door(
     continue-up(
      continue-n-up(
       request-floor(FLOOR,
        close-door(state(FLOOR1, DIRECT1, open, FLOOR1, DIRECT)))))) .
red floor?(request-floor6(upper(upper-n(floor)),
  state(floor, direct, open, floor, direct1))) == upper(upper-n(floor)) .
close .
```

以下が実行結果

```
[mercury:UN 1] % cafeobj
-- loading standard prelude
Loading /tmp_mnt/local.ldl/ldl/src/lang/cafe/cafeobj-1.4/prelude/std.bin
Finished loading /tmp_mnt/local.ldl/ldl/src/lang/cafe/cafeobj-1.4/prelude/std.bin

           -- CafeOBJ system Version 1.4.0(Beta-5) --
                built: 1997 Dec 3 Wed 11:34:27 GMT
                     prelude file: std.bin
                              ***
                   1998 Feb 12 Thu 9:03:43 GMT
                        Type ? for help
                              ---
                   uses GCL (GNU Common Lisp)
            Licensed under GNU Public Library License
              Contains Enhancements by W. Schelter

CafeOBJ> in proof1
-- processing input : ./proof1.mod
-- processing input : /glico/home2/n-ume/F12/PefPro/start.mod
-- defining module! FLOOR.............._.........* done.
-- defining module DIRECT....._..* done.
-- defining module DOOR...._* done.
-- defining module LFT....._* done.
-- defining module HSLFT........._...*
** system already proved =*= is a congruence of HSLFT done.
-- defining module HLFT................._........* done.
-- opening module HLFT.. done.
--> VVVVV shuld be true_*
-- reduce in % : floor?(initial-hlft(lft)) == floor1
```

```
true : Bool
(0.000 sec for parse, 3 rewrites(0.000 sec), 3 match attempts)
--> 詳細化前モデルの性質(axiom)の証明2
-- opening module HLFT.. done.
--> VVVV Base 1 should be true VVVV_*
-- reduce in % : floor?(request-lft2(upper(floor-tmp),direct2,state(
    floor-tmp,direct1,open,floor-tmp,direct1))) == upper(floor-tmp)
true : Bool
(0.017 sec for parse, 7 rewrites(0.000 sec), 22 match attempts)
--> VVVV Base n induction hypothesis be true VVVV_
--> VVVV Base n+1 should be true VVV_*
-- reduce in % : floor?(request-lft3(upper(upper-n(floor-tmp)),direct2,
    state(floor-tmp,direct1,open,floor-tmp,direct1))) == upper(upper-n(floor-tmp))
true : Bool
(0.017 sec for parse, 8 rewrites(0.000 sec), 24 match attempts)
--> VVVV Base 1 should be true VVVV_*
-- reduce in % : floor?(request-lft5(lower(floor-tmp),direct2,state(
    floor-tmp,direct1,open,floor-tmp,direct1))) == lower(floor-tmp)
true : Bool
(0.017 sec for parse, 7 rewrites(0.000 sec), 23 match attempts)
--> VVVV Base n should be true VVVV_
--> VVVV Base n+1 should be true VVVV_*
-- reduce in % : floor?(request-lft6(lower(lower-n(floor-tmp)),direct2,
    state(floor-tmp,direct1,open,floor-tmp,direct1))) == lower(lower-n(floor-tmp))
true : Bool
(0.017 sec for parse, 8 rewrites(0.017 sec), 25 match attempts)
-- reduce in % : floor?(request-lft4(floor-tmp,direct2,state(floor-tmp,
    direct1,open,floor-tmp,direct1))) == floor-tmp
true : Bool
(0.000 sec for parse, 6 rewrites(0.000 sec), 8 match attempts)
--> 詳細化前モデルの性質(axiom)の証明3
-- opening module HLFT.. done.
--> Base 1 VVVVV shuld be true _*
-- reduce in % : floor?(request-floor2(lower(floor),state(floor,direct,
    open,floor,direct1))) == lower(floor)
true : Bool
(0.000 sec for parse, 7 rewrites(0.000 sec), 22 match attempts)
--> Base n VVVVV be true _
--> Base n+1 VVVVV shuld be true _*
-- reduce in % : floor?(request-floor3(lower(lower-n(floor)),state(
    floor,direct,open,floor,direct1))) == lower(lower-n(floor))
true : Bool
(0.017 sec for parse, 8 rewrites(0.017 sec), 24 match attempts)
--> 同様に上がりたい時も行う。
--> Base 1 VVVVV shuld be true _*
-- reduce in % : floor?(request-floor4(upper(floor),state(floor,direct,
```

```
    open,floor,direct1))) == upper(floor)
true : Bool
(0.033 sec for parse, 7 rewrites(0.000 sec), 23 match attempts)
--> Base n VVVVV be true __
--> Base n+1 VVVVV shuld be true _*
-- reduce in % : floor?(request-floor6(upper(upper-n(floor)),state(
    floor,direct,open,floor,direct1))) == upper(upper-n(floor))
true : Bool
(0.017 sec for parse, 8 rewrites(0.000 sec), 25 match attempts)
HLFT>
```

A.5 事例 LIFT の並行性の検証スコア

```
select HSLFTSYS .
open .
op _R_ : HslftSys HslftSys -> Bool .

--> give a candidate hidden congruence relation R .
vars HS1 HS2 : HslftSys .
eq HS1 R HS2 =
      ( sfloor?(slft?(lft1, HS1)) == sfloor?(slft?(lft1, HS2)) ) and
      ( sfloor?(slft?(lft2, HS1)) == sfloor?(slft?(lft2, HS2)) ) .

--> hypothiesis .
ops hs1 hs2 : -> HslftSys .
eq [ hypo1 ]: sfloor?(slft?(lft1, hs1)) = sfloor?(slft?(lft1, hs2)) .
eq [ hypo2 ]: sfloor?(slft?(lft2, hs1)) = sfloor?(slft?(lft2, hs2)) .

-- prove the R is a congruence .
-- ops lft1 lft2 : -> Lft .
op floor : -> Floor .
red request-slft(lft1, floor, hs1) R request-slft(lft1, floor, hs2) .
red request-slft(lft2, floor, hs1) R request-slft(lft2, floor, hs2) .
red request-sfloor(lft1, floor, hs1) R request-sfloor(lft1, floor, hs2) .
red request-sfloor(lft2, floor, hs1) R request-sfloor(lft2, floor, hs2) .

-- prove beq sfloor(slft?(lft1, request-slft(lft1, floor, request-slft(lft2, floor, initial-ssys))))
--          = sfloor(slft?(lft1, request-slft(lft2, floor, request-slft(lft1, floor, initial-ssys)))) .
-- prove beq sfloor(slft?(lft2, request-slft(lft1, floor, request-slft(lft2, floor, initial-ssys))))
--          = sfloor(slft?(lft2, request-slft(lft2, floor, request-slft(lft1, floor, initial-ssys)))) .

red request-slft(lft1, floor, request-slft(lft2, floor,initial-ssys))
  R request-slft(lft2, floor, request-slft(lft1, floor, initial-ssys)) .

close .
```

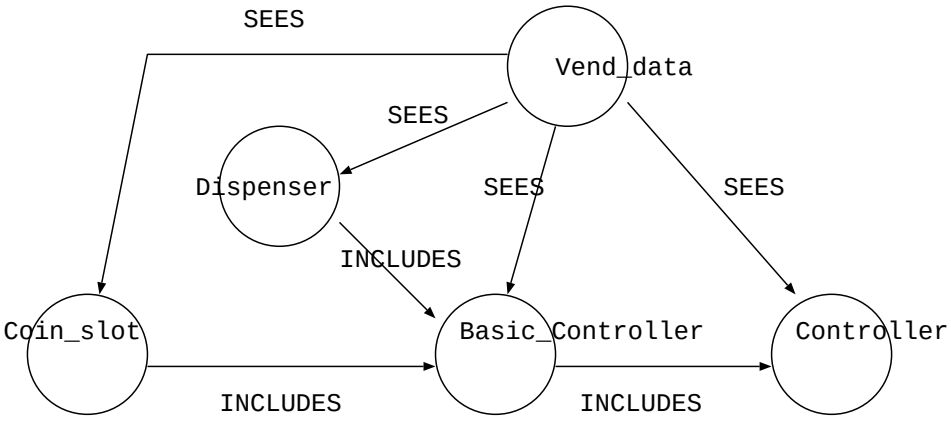
以下実行結果

```
[mercury:UN 1] % cafeobj
-- loading standard prelude
Loading /tmp_mnt/local.ldl/ldl/src/lang/cafe/cafeobj-1.4/prelude/std.bin
Finished loading /tmp_mnt/local.ldl/ldl/src/lang/cafe/cafeobj-1.4/prelude/std.bin

           -- CafeOBJ system Version 1.4.0(Beta-5) --
                built: 1997 Dec 3 Wed 11:34:27 GMT
                      prelude file: std.bin
```

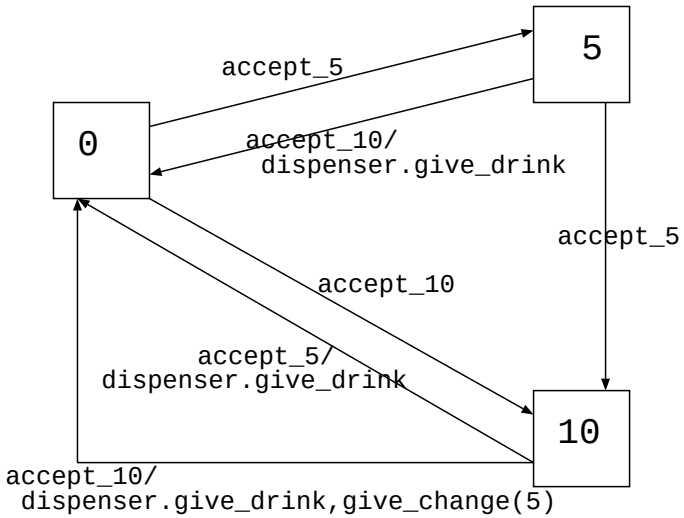
```
                          ***
                  1998 Feb 8 Sun 16:26:55 GMT
                        Type ? for help
                          ---
                  uses GCL (GNU Common Lisp)
            Licensed under GNU Public Library License
              Contains Enhancements by W. Schelter

CafeOBJ> in proof2
-- processing input : ./proof2.mod
-- processing input : /glico/home2/n-ume/F9/start.mod
-- defining module! FLOOR.............._.........* done.
-- defining module DIRECT....._..* done.
-- defining module DOOR...._* done.
-- defining module LFT....._* done.
-- defining module HSLFT........._...*
** system already proved =*= is a congruence of HSLFT done.
-- defining module* HSLFTSYS........._.....*
** system already proved =*= is a congruence of HSLFTSYS done.
-- opening module HSLFTSYS.. done.
--> give a candidate hidden congruence relation R ._
--> hypothiesis .__*
-- reduce in % : request-slft(lft1,floor,hs1) R request-slft(lft1,
    floor,hs2)
true : Bool
(0.017 sec for parse, 19 rewrites(0.050 sec), 145 match attempts)
-- reduce in % : request-slft(lft2,floor,hs1) R request-slft(lft2,
    floor,hs2)
true : Bool
(0.000 sec for parse, 19 rewrites(0.033 sec), 146 match attempts)
-- reduce in % : request-sfloor(lft1,floor,hs1) R request-sfloor(
    lft1,floor,hs2)
true : Bool
(0.000 sec for parse, 19 rewrites(0.017 sec), 147 match attempts)
-- reduce in % : request-sfloor(lft2,floor,hs1) R request-sfloor(
    lft2,floor,hs2)
true : Bool
(0.000 sec for parse, 19 rewrites(0.017 sec), 148 match attempts)
-- reduce in % : request-slft(lft1,floor,request-slft(lft2,floor,
    initial-ssys)) R request-slft(lft2,floor,request-slft(lft1,floor,
    initial-ssys))
true : Bool
(0.033 sec for parse, 36 rewrites(0.033 sec), 102 match attempts)

HSLFTSYS>
```

第 B 章

事例 Vending Machine



Development Architecture of Vending Machine Development



Statechart of Vending Machine

ここでControllerは仕様のシミュレーションの為の環境

B.1 事例 Vending Machine のB-抽象機械仕様

```
  MACHINE Vend_data
SETS
        CSTATE = { coin_present, coin_absent};
        DSTATE = { stocked, unstocked }
CONSTANTS COINS, STATE
PROPERTIES
        COINS = { 5, 10 } ⋀ STATE = { 0, 5, 10 }
END


  MACHINE Dispenser
SEES Vend_data
VARIABLES
        dstate, given
INVARIANT
        dstate ∈ DSTATE ⋀ given ∈ N
INITIALISATION
        dstate := unstocked || given := 0
OPERATIONS
        restock ≡ dstate := stocked;
        give_drink ≡
            PRE dstate = stocked
            THEN dstate :∈ DSTATE || given := given + 1
            END
END
MACHINE Coin_slot
SEES Vend_data
VARIABLES
      cstate, current_coin
INVARIANT
      cstate ∈ CSTATE
      current_coin ∈ COINS
INITIALISATION
      cstate := coin_absent ||
      current_coin :∈ COINS
OPERATIONS
      give_change(cc) def
            PRE cc ∈ COINS
            THEN cstate := coin_absent
            END;
      coin ← accept_coin def
```

```
            ANY cc
            WHERE cc ∈ COINS
            THEN
            current_coin := cc ||
            cstate := coin_present ||
            coin := cc
            END
END
```

```
  MACHINE BasicController
SEES Vend_data
INCLUDES Dispenser, Coin_slot
PROMOTES restock, accept_coin
VARIABLES state
INVARIANT
        state ∈ STATE
INITIALISATION
        state := 0
OPERATIONS
        accept_10 def
           BEGIN
              IF state = 0
              THEN state := 10
              ELSE state := 0
              END ||
              IF state = 5
              THEN give_drink
              ELSE
                 IF state = 10
                 THEN give_drink || give_change(5)
                 END
              END
           END;
        accept_5 def
           BEGIN
              state := (state + 5) mod 15 ||
              IF state = 10
              THEN give_drink
              END
           END
END
```

B.2 事例 Vending Machine の CafeOBJ 仕様

```
  module! VEND-DATA1 {
            [ Cstate, Coins ]
            op coin-present : → Cstate
            op coin-absent : → Cstate
            op zeroc : → Coins
            op fivec : → Coins
            op tenc : → Coins
}
module! VEND-DATA2 {
            [ Dstate ]
            op stocked : → Dstate
            op unstocked : → Dstate
}
module VEND-DATA3 {
            [ State ]
            op zeros : → State
            op fives : → State
            op tens : → State
}
module! DISPENSER {
            protecting(VEND-DATA2)
            protecting(SIMPLE-GIVEN)
            signature {
               [ Dispenser ]
               op restock : Dstate → Dstate
               op give-drink : Dispenser → Dispenser
               op empty-dispenser : → Dispenser
               op _~_ : Dstate Given → Dispenser
            }
            axioms {
               var DS : Dstate
               var G : Given
               eq empty-dispenser = (unstocked žero) .
               eq restock(DS) = stocked .
               ceq give-drink(DS G̃) = stocked šucc(G) if DS == stocked .
               ceq give-drink(DS G̃) = unstocked šucc(G) if DS == stocked .
            }
}
module! COIN-SLOT {
            protecting(VEND-DATA1)
            signature {
               [ CSlot ]
               op empty-cslot : → CSlot
               op _ | _ : Cstate Coins → CSlot
```

```
        op give-change : CSlot Coins → CSlot
        op accept-coin : CSlot Coins → CSlot
        op entered-coin : CSlot → Coins
        op current-state : CSlot → Cstate – for proof
    }
    axioms {
        var CS : Cstate
        vars CO1,CO2 : Coins
        var CL : CSlot
        eq empty-cslot = coin-absent | fivec .
        eq empty-cslot = coin-absent | tenc .
        eq accept-coin(CL, CO2) = (coin-present | CO2) .
        eq give-change(CL,CO1) = (coin-absent | CO1) .
        eq current-state(CS | CO2) = CS . – for proof
        eq entered-coin(CS | CO2) = CO2 .
    }
}
module! BASIC-COINTROLLER {
    protecting(VEND-DATA3)
    using(DISPENSER)
    using(COIN-SLOT)
    signature {
        [ Basic BContrl ]
        op empty-basic-contrl : → BContrl
        op _ ^ _ : Dispenser CSlot → Basic
        op _ @ _ : State Basic → BContrl
        op accept10 : BContrl → BContrl
        op accept5 : BContrl → BContrl
    }
    axioms {
        var DS : Dispenser
        var ST : State
        var CL : CSlot
        eq empty-basic-contrl = zeros @ (empty-dispenser ^ empt y-cslot) .
        eq accept10(zeros @ (DS ^ CL)) = tens @ (DS ^ (give -change(CL, zeroc))) .
        eq accept10(fives @ (DS ^ CL))= zeros @ (give-drink(DS) ^ give-change(CL, zeroc)) .
        eq accept10(tens @ (DS ^ CL)) = zeros @ (give-drink(DS) ^ give-change(CL, fivec)) .
        .....
        eq accept5(zeros @ (DS ^ CL)) = fives @ (DS ^ give-change(CL, zeroc)) .
        eq accept5(fives @ (DS ^ CL)) = tens @ (DS ^ give-change(CL, zeroc)) .
        eq accept5(tens @ (DS ^ CL)) = zeros @ (give-drink(DS) ^ give-change(CL, zeroc)) .
    }
}
```

B.3 事例 Vending Machine の CafeOBJ 仕様のシミュレーション

```
[mercury:UN 62] % cafeobj
-- loading standard prelude
Loading /tmp_mnt/local.ldl/ldl/src/lang/cafe/cafeobj-1.4/prelude/std.bin
Finished loading /tmp_mnt/local.ldl/ldl/src/lang/cafe/cafeobj-1.4/prelude/std.bin

           -- CafeOBJ system Version 1.4.0(Beta-5) --
                built: 1997 Dec 3 Wed 11:34:27 GMT
                     prelude file: std.bin
                              ***
                   1998 Feb 12 Thu 4:52:56 GMT
                         Type ? for help
                              ---
                   uses GCL (GNU Common Lisp)
            Licensed under GNU Public Library License
              Contains Enhancements by W. Schelter

CafeOBJ> in basic-controller
-- defining module VEND-DATA..............._* done.
-- defining module VEND-DATA1........_* done.
-- defining module VEND-DATA2..._* done.
-- defining module VEND-DATA3...._* done.
-- defining module! COIN-SLOT..........._......* done.
-- defining module SIMPLE-GIVEN..._* done.
-- defining module DISPENSER........._....* done.
-- defining module BASIC-CONTROLLER..

BASIC-CONTROLLER> red empty-basic-contrl .
-- reduce in BASIC-CONTROLLER : empty-basic-contrl
zeros @ ((stocked ~ zero) ^ (coin-absent | fivec)) : BContrl
(0.000 sec for parse, 3 rewrites(0.017 sec), 3 match attempts)

BASIC-CONTROLLER> red accept10(empty-basic-contrl) .
-- reduce in BASIC-CONTROLLER : accept10(empty-basic-contrl)
tens @ ((stocked ~ zero) ^ (coin-absent | zeroc)) : BContrl
(0.000 sec for parse, 5 rewrites(0.017 sec), 5 match attempts)

BASIC-CONTROLLER> red accept5(accept10(empty-basic-contrl)) .
-- reduce in BASIC-CONTROLLER : accept5(accept10(empty-basic-contrl)
    )
zeros @ ((stocked ~ succ(zero)) ^ (coin-absent | zeroc)) : BContrl
(0.017 sec for parse, 9 rewrites(0.017 sec), 11 match attempts)
```

```
BASIC-CONTROLLER> red accept5(accept5(accept10(empty-basic-contrl))) .
-- reduce in BASIC-CONTROLLER : accept5(accept5(accept10(empty-basic-contrl)
    ))
fives @ ((stocked ~ succ(zero)) ^ (coin-absent | zeroc)) : BContrl
(0.000 sec for parse, 11 rewrites(0.000 sec), 13 match attempts)

BASIC-CONTROLLER> red accept5(accept5(accept5(accept10(empty-basic-contrl)))) .
-- reduce in BASIC-CONTROLLER : accept5(accept5(accept5(accept10(
    empty-basic-contrl))))
tens @ ((stocked ~ succ(zero)) ^ (coin-absent | zeroc)) : BContrl
(0.000 sec for parse, 13 rewrites(0.017 sec), 16 match attempts)

BASIC-CONTROLLER> red accept10(accept5(accept5(accept5(accept10(empty-basic-contrl))))) .
-- reduce in BASIC-CONTROLLER : accept10(accept5(accept5(accept5(
    accept10(empty-basic-contrl)))))
zeros @ ((stocked ~ succ(succ(zero))) ^ (coin-absent | fivec)) : BContrl
(0.017 sec for parse, 17 rewrites(0.017 sec), 22 match attempts)

BASIC-CONTROLLER>
```