JAIST Repository
https://dspace.jaist.ac.jp/

Title	CafeOBJ を用いたハイブリッドシステムの形式的な仕様記述と検証
Author(s)	山岸，大悟
Citation	
Issue Date	2004-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1772
Rights	
Description	Supervisor:二木 厚吉，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

CafeOBJ を用いた
ハイブリッドシステムの形式的な仕様記述と検証

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

山岸 大悟

2004 年 3 月

修 士 論 文

CafeOBJ を用いた ハイブリッドシステムの形式的な仕様記述と検証

主指導教員　二木 厚吉 教授

審査委員主査　二木 厚吉 教授
審査委員　片山 卓也 教授
審査委員　落水 浩一郎 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

210097 山岸 大悟

提出年月: 2004 年 2 月

Copyright © 2004 by Daigo Yamagishi

概 要

ハイブリッドシステムは, 連続事象と離散事象が合成されたシステムで, 高信頼性を求められるものに関しては, まだ検証困難である. 本研究は, 代数仕様言語 CafeOBJ を用いてハイブリッドシステムの仕様を記述するためのモデルを提案することである. さらに, 例題を用いてその手法を用いた記述を行い, CafeOBJ で記述する. 最後に安全性を検証し, 得られた結果からモデルの有効性を確かめる.

CafeOBJ が属する代数仕様言語は, 代数に基づいた仕様が記述できる. そして等式推論を用いた検証を行うことが可能である. CafeOBJ が持つ特徴の 1 つに, 隠蔽代数を用いた抽象機械の記述, 検証がある. CafeOBJ で抽象機械を記述するための数学モデルとして観測遷移機械 (OTS : Ovservational Transition System) が提案されており, プロトコルの検証, 制御システムの検証等で成果があげられている. しかしながら観測遷移機械では, 連続系と離散系が混合されたハイブリッドシステムに対しての記述手法に関する提案は, まだされていない.

本研究では, 観測遷移機械を拡張し, ハイブリッドシステムに対して適応可能なもの (HOTS : Hybrid Observational Transition System) を新たに提案した. 提案した手法を用いるために, 2 つの例題を対象にモデル化を行った. そして CafeOBJ を用いて, モデル化されたシステムの記述を行い, 安全性の検証を行った.

結論として, 今回用いた例題に限れば, ハイブリッド観測遷移機械でモデル化された仕様に関しても, これまで検証を行ってきたものと同様に検証可能であることが確認できた. しかしながら, CafeOBJ が属する証明を用いた検証支援系では, 仕様記述者が証明をある程度推測する必要があり, その手間に関しては, 今回用いた例題に対しても同様であった. 詳しく述べると, 2 つの力学形が相互作用を及ぼし合うシステムを記述した場合, 仕様記述者は, 相互作用が行われた際の合成された解軌道を予測せねばならなかった. 相互作用を及ぼし合う力学形が増えた場合や, 力学形が持つ解軌道そのものが複雑になる場合では, 設計者は, 検証するために, 高い数学レベルでの推測をしなければならない. 今後の課題は, HOTS に対して連続変数の推測を決定づける手法の提案や, CafeOBJ で記述する際, 連続変数を処理系が自動的に推測可能な実装を行うことである.

目 次

第 1 章　序論 **1**
1.1　研究背景、目的、成果 1
1.2　本論文の構成 2

第 2 章　準備 **3**
2.1　ハイブリッドシステム 3
2.1.1　研究領域 3
2.2　代数仕様言語 CafeOBJ 6
2.3　観測遷移機械 (OTS : Observational Transition System) 12

第 3 章　観測遷移機械の拡張 **21**
3.1　ハイブリッド観測遷移機械 (HOTS : Hybrid Observational Transition System) 21
3.1.1　定義 21
3.1.2　アクティビティ規則 24

第 4 章　例題 **27**
4.1　サーモスタット (自動温度調節装置) 27
4.2　交差点問題 38

第 5 章　結論 **48**
5.1　研究成果 48
5.2　関連研究 49
5.2.1　フェーズ遷移機械 (PTS : Phase Transition System) 49
5.2.2　ハイブリッドオートマトン 51
5.2.3　連続力学系から離散遷移機械への抽象化 54
5.3　今後の課題 57

図 目 次

2.1 可視ソート Nat を用いた Peano 自然数と, 隠蔽ソートを用いたカウンター 7
2.2 $1 + 2 = 3$ の簡約と, 結合律の証明譜に関しての実行結果 8
2.3 抽象機械 `COUNTER` のグラフ 11
2.4 例題 : INCDEC 18
2.5 `INCDEC` モジュール 19
2.6 `INV, ISTEP` モジュール 19
2.7 `INCDEC` に対する証明譜 (`inv100` のみ) 20

3.1 ハイブリッドシステムの動作 22
3.2 アクティビティルール 24

4.1 例題 : サーモスタット 28
4.2 観測の記述 (サーモスタット) 31
4.3 初期値の記述 (サーモスタット) 31
4.4 条件付遷移規則の記述その 1(サーモスタット) 32
4.5 条件付遷移規則の記述その 2(サーモスタット) 33
4.6 証明譜 1 (activate1-on) 36
4.7 証明譜 2 (tick) 37
4.8 例題 : 交差点問題 38
4.9 車 1 のアクティビティ 39
4.10 車 2 のアクティビティ 39
4.11 観測の記述 (交差点) 43
4.12 初期値の記述 (交差点) 44
4.13 条件付遷移規則の記述 (交差点) 44
4.14 証明譜 (交差点: *tick*, `lem 107` が必要な場合) 47

第1章　序論

本章では, 本論文の前書きとして, 研究背景, 目的, 得られた成果に関して簡潔にまとめる. そして本論文の構成に関して述べる.

1.1　研究背景、目的、成果

ハードウェアの急激な進歩に伴い, 大規模, 複雑化された制御システム, あるいは, 要求が複雑になったソフトウェア開発行程では, ときとして, 予期せぬ障害を引き起こし, それが高信頼性を要求されるシステムであった場合には, 重大な事故を招く要因にもなり得る. 障害の原因は様々であるが, その 1 つに, システム設計者が, 仕様段階で間違ったことを記述している場合や, 作成された仕様が意図通りに開発者に伝わらないことがある. 従って, 仕様記述者は, 仕様を基に作業を行う下流行程の設計者に対して, 誤りがなく, かつ可読性にすぐれた仕様を提供しなければならない.

システム設計者が正しく設計を行うための手法の 1 つに, 形式仕様を用いた記述手法がある. 形式仕様で記述された仕様は, 数学を基にしているため, 読み方が統一される. また, システムが持つ安全性の検証は, 数学に基づいた証明により行うことが可能である. 検証の結果, 満たしたい性質が証明できた場合は, 記述された仕様の範囲において誤りが無いことが保証される. これらの点は, 自然語で記述される仕様と比較して強力であり, 高信頼性が要求される仕様記述を行うための手段として有効である.

形式仕様を用いた検証は, 仕様で定義された公理, 推論規則に基づいており, その証明は, 主に項書き換えによる簡約や, 帰納法によって行われる. 検証支援系として, 証明を自動化するツールの開発も行われており, 形式仕様によって記述された仕様を計算機上で実行し, 証明することも可能であり, 航空管制システム, 鉄道システムなどで適用が実際に行われている.

要求が複雑化するシステムの計算機支援を可能にするために, システムに時間制約が付加されたもの, いわゆるリアルタイムシステムに基づいた仕様記述法の提案や実装手法についても研究がされている. さらには, 組み込みシステム等, アナログ環境に組み込まれたデジタルな実時間システムに代表される, ハイブリッドシステムに関しても, 信頼性保証のための研究が行われている.

本研究の目的は, 計算機が実行可能なハイブリッドシステムの仕様記述手法を確立することである. 使用する言語は, 代数を基にした証明を行うことが可能な代数仕様言語

CafeOBJ [1][2] である．CafeOBJ では, 抽象機械を記述するための手法 (OTS：Observational Transition System) [3][4] がすでに提案されており, さらにハイブリッドシステムのサブクラスであるリアルタイムシステムに対しても, それを記述するための数学モデルがすでに提案されている (TOTS：Times Observational Transition System) [5]．しかしながら, TOTS は連続変数の進化は時間のみを扱っており, ある物理変数が常微分方程式に従って時間変化するハイブリッドシステムは表現できない．従って研究は, まず OTS を拡張することから行った．ハイブリッドシステムを記述できる OTS (HOTS：Hybrid Observational Transition System) を提案した後, 2 つの例題に対して, HOTS を用いたモデル化を行い, CafeOBJ 処理系を用いた安全性の検証を行った.

本研究により, 使用した例題に関しては, HOTS が有効であることが確認できた．しかしながら, 常微分方程式の解軌道や, 合成されたシステムの実数制約等は, システム設計者がある程度推測しなければならず, 挙動がより複雑になるシステムに関しては, 検証支援が困難になることが予想される．今後の課題は, 検証支援に関して, システム設計者の推測が軽減できるモデル化, 又は実数における決定手続きを CafeOBJ に反映させることである.

1.2 本論文の構成

本論文は全 5 章で構成される．本章では, これから述べる研究内容に関して簡潔にまとめた.

第 2 章では, 研究に必要な知識を提供するため, まず, 本研究で対象となるハイブリッドシステムに関する概要, 研究分野に関して述べる．次に, 検証支援系として用いる代数仕様言語 CafeOBJ に関する紹介を行う．最後に, CafeOBJ で記述可能な, 抽象機械を表現する数学モデル OTS の紹介を行う.

第 3 章では, ハイブリッドシステムが記述できる OTS すなわち HOTS と, 連続系を扱うための重要な要素であるアクティビティ規則に関して解説する.

第 4 章では, 2 つの例題：サーモスタット (自動温度制御装置), 交差点問題, に対して HOTS を用いたモデル化と CafeOBJ を用いた仕様記述, 検証を行い, 例題から得られる考察に関してまとめる.

第 5 章では, 得られた結論と今後の課題を述べる, そして, 本研究と関係があるいくつかの研究に関して紹介を行う.

第2章　準備

本章は, 本研究の基となる知識を紹介する. はじめに, 研究対象として用いたハイブリッドシステムについての概要と研究領域の紹介を行う. 次に, ハイブリッドシステムの記述, 検証で使用した代数仕様言語 CafeOBJ の紹介を行う. 最後に, 抽象機械を表現するための数学モデルである観測遷移機械 OTS の定義, OTS と CafeOBJ を用いた検証手法について, 簡単な例題を交えた解説を行う.

2.1　ハイブリッドシステム

ハイブリッドシステムは, より広範囲なシステムを表現するために, システムの内部状態に連続変数と離散変数の両方を含んだものである. 言い換えれば, アナログ環境にデジタルが組み込まれたようなシステムである. 連続変数と離散変数両方の複雑さを扱うことから, 連続変数の複雑さを扱うシステム制御理論と, 離散変数の複雑さを扱う離散事象システム理論の両研究者が, それぞれ専門とするシステム制御に, 新しい変数を導入することで研究が進められている. 又, ハイブリッドシステムが表現できるものには, 高信頼性を含んだものも存在し, それらが必要な安全性の検証についても, いくつかの研究が行われている. 具体的な研究対象として, 電気自動車, 化学プラント, ロボットアームの制御等が挙げられる

2.1.1　研究領域

ハイブリッドシステムの研究領域は大きく 2 つに分類される. 連続変数の複雑さを扱うシステム制御理論と, 離散変数の複雑さを扱う離散事象システム理論の 2 つである. システム制御理論では, これまで主に扱ってきた連続変数に, ベクトル場の切り替えによる離散変化を取り入れ, 拡張されることが多い. 逆に, 離散事象システム理論では, これまで離散的に表現されてきた事象の生起条件に連続変数を取り入れることで拡張を行っている. 以下に各分野の紹介といくつかの研究事例を挙げる [6].

システム制御理論

システム制御理論の立場でハイブリッドシステムを解析する際に持つ共通の問題点は, 不連続な常微分方程式が原因で, ハイブリッドシステムの最も簡単なクラスを考えた場合

でも, well-posedness の問題が生じることである．言い換えると, 全体システムを常微分方程式でモデル化した場合, 解の一意性が保証されないという問題である．解の一意性が保証されないままシステムを設計すると, 意図した通りの性能が得られない可能性がある．従ってこの問題はハイブリッドシステムの基本問題となっている．以下に, 連続変数の制御対象に対してハイブリッド制御器を用いた研究事例を 2 つ紹介する．

拘束システムの制御　入力制限に代表される拘束条件を持つシステムの制御である．制御法として, 複数の制御器を切り替えながら制御するスイッチング制御 [7] や, 外部から, 閉ループ系, すなわち制御入力に対する拘束を設計するリファレンスガバナがある [8].

非線形システム　連続な制御器では制御が不可能な非線形制御対象に対する制御法である．典型的な例は非ホロノミックシステムである．非ホロノミックシステムでは, 速度や加速度間に積分できない関係式があるために制御が不可制御におちいる問題があり, その解決策としてスイッチング制御を用いた制御法が提案されている [9].

上記の 2 つは, 制御器を用いることがハイブリッド性を持たせる要因となっている．これに対して, 制御対象自信がハイブリッドシステムの場合がある．この場合にはハイブリッドな入出力信号に対してのハイブリッド制御器を設計することになる．ハイブリッド制御器の設計法に関しては, まだ体系化がされていないが, その中で, Bemporad と Morari によって提案された混合論理ダイナミカルシステム (MLD) が注目されている [10]．MLD を用いた, システムモデル化のためのツール HYSDEL も開発されている これは, ハイブリッドシステムの検証問題にも適用できる [11][12].

離散事象システム理論

形式言語やオートマトンを用いた離散事象システムに基づくモデリングは, 時間情報を考察の対象から切り捨てて行うことが主流であった．しかしながらこのモデリングでは, 時間情報に関して最悪を仮定しなくてはならず, システムの設計が制限されてしまう．これを解決するための最も簡単な手段として, 時間の経過を表す, “*tick*” を導入し, その生起によって 1 単位時間の経過を表す手法が, Brandin と Wonham によって提案された [13]. 時間情報をさらに詳細に記述するために, 時間の経過を表す連続変数 x を導入し,

$$\dot{x} = 1$$

という常微分方程式により時間を変化させ, x を含む条件式により事象を生起させ, x をリセットさせる．このようなモデルは時間オートマトンと呼ばれる．このモデルは連続変数 x を導入していることからハイブリッドシステムの一種である．時間オートマトンをさらに発展したものとして, ハイブリッドオートマトンが提案された [14]．これは, 計算機科学においてハイブリッドシステムを研究するための標準的なモデルになっている．もう

一方では, ペトリネットにおいてトークンを離散値から連続値に拡張したハイブリッドペトリネット [15] に関する研究が行われている. これは連続変数と離散変数が同等に扱われているということで興味深いモデルである.

計算機科学においてもハイブリッドシステムが注目されている. システムが仕様通りに記述されているかどうかを計算機を用いてチェックするための手法として, 定理証明を用いた計算機支援と, モデルチェッキングを用いた解析がある. 以下にそれぞれの特徴と利点, 欠点を述べる.

定理証明	システムを, 計算機が推論可能な論理式で記述し, 計算機により検証したい性質を推論規則に基づき証明する手法である. システムの状態が無限になる場合でも検証が可能であるが, 計算機による証明が途中で終了した場合には, システム設計者が証明を行えるための補題を発見しなければならない. また定理証明を用いた証明では, 書かれた仕様に関する性質の正しさが, 証明を用いて検証できなかった場合, その性質が間違っていることを保証することは不可能である. 言い換えれば, 定理証明が保証できる範囲は, 記述された仕様に関して, 性質が正しい結果を返せば, それが正しいことのみである.
モデルチェッキング	モデルチェッキングとは, オートマトンと時相論理で記述された, システムの振る舞いとシステムが検証したいある特定された性質を, 抽象化によって, 計算機が解析可能, かつ元のシステムと等価な遷移システムへと返還し, 最後に計算機を用いて, 抽象化を行った遷移システムの全ての状態を探索することにより, 特定された性質の状態が出現することを確認する手法である. システム設計者は推論を行わずに, 性質を検証することが可能であるが, 検証したい性質各々に関して抽象化を行わなければならず, その作業は容易ではないことが多い.

計算機科学において検証すべき性質は, 主に安全性 (safety) と活性 (liveness) である. 安全性とは, 満たしたい性質が常に成り立つことを保証するものである. 例えば,

- 交差点において, 全ての信号が同時に青になることはない.
- 列車 a は, 速度 v を越えない範囲で運行しなければならない.

は安全性である. システム開発を行う上で, 安全性を発見し, 検証することは最も重要な要素の 1 つである, 安全性が保証されない, あるいは安全性が発見できないシステムは, いつか重大な事故を引き起こす可能性を持つ. これに対して活性とは, 満たしたい性質がいつか必ず生じることを保証するものである. 例えば,

- 交差点を通過しようとする車は, いつか必ず交差点を通過することができる
- 設定温度 t に設定された部屋では, 室温 θ は, 特別な外力が生じない限りいつか必ず t に到達する.

は活性である.

計算機科学では, モデルチェッキングを用いてハイブリッドシステムを検証する研究が行われている [16]. この時の問題点は, 完全な抽象化を行う際のクラスが 1 次導関数のみであり, ハイブリッドシステムの非常に狭いクラスに限定されてしまうことである. そのために, システムは何らかの近似を用いて設計しなければならず, この近似解法に関する研究は, 検証において重要な部分になっている.

本研究は, 上述したハイブリッドシステムの諸分野中においては, 計算機科学に属しており, 計算機を用いた定理証明を行うことになる. 研究動機は, ハイブリッドシステムの検証に関して, モデルチェッキングでは扱えないような広いクラスに関して, 計算機支援を可能にすることである. 尚, 本研究は安全性のみを検証対象とした.

2.2 代数仕様言語 CafeOBJ

代数仕様言語とは, 代数に基づいた仕様が記述可能な言語である. 代数仕様による記述には, 始代数をもとにした抽象データ型による記述と, 隠蔽代数を用いた記述ある.

始代数

抽象データ型は, データ型の表現形式をデータ型からデータ型への演算を定義することにより掌握できる. 演算の振る舞いは演算から生成される項の集まりによって定められる. データ型は, データを表すシステムの記述に必要な台集合であるソートとソート上の演算, 演算から生成される項によって表現される.

隠蔽代数

ある 2 つのシステムの等価性を抽象データ型で考える場合, それがもし, あらゆる実行に対して同じ振る舞いを示すのなら, そのシステムは等価であると判断したい. そのような場合は, 振る舞い等価性に基づく隠蔽代数を用いて考える. 隠蔽代数では, システムの状態は, 通常のデータ型である可視ソートと, 内部状態を含む集合である隠蔽ソートによって決まる. システムは, 状態のあらゆる遷移の後, その内部を観測した値が等しい場合に等価性が成り立つ.

CafeOBJ

CafeOBJ は, 始代数に基づいた抽象データ型と, 抽象機械を記述するための隠蔽代数を統一の枠組みで記述可能な言語である. それぞれの代数は, プログラム言語の型に相当するソートを用いて区別される. 抽象データ型は可視ソート (visible sort) を用いて表すことが可能であり, 抽象機械の状態は隠蔽ソート (hidden sort) を用いて表すことが可能である. CafeOBJ コードを用いたモジュール記述例を以下に示す. さらにモジュールを用いた簡約と証明を次ページで示す. 各コードが持つ意味はその次のページで示している.

```
mod! SIMPLE-NAT {                          -- モジュールの宣言
  [ Zero NzNat < Nat ]                     -- ソートの宣言
  op 0 : -> Zero                           -- Nat 上における定数定義
  op s_ : Nat -> NzNat                     -- Nat 上の演算定義
  op _+_ : Nat Nat -> Nat                  -- Nat 上の演算定義

  vars M N : Nat .                         -- Nat 上の変数定義
  eq N + 0 = N .                           -- 書き換え規則の定義
  eq N + (s M) = s(N + M) .                -- 書き換え規則の定義
}                                          -- モジュールの終了

mod* COUNTER{
  pr(SIMPLE-NAT)                           -- モジュールの輸入
  *[ Counter ]*
  bop value : Counter -> Nat               -- 状態を観測する関数
  bop increase : Nat Counter -> Counter -- 状態を変化させる関数

  var C : Counter
  var N : Nat
  eq value(increase(N,C)) = N + value(C) .
}
```

図 2.1: 可視ソート Nat を用いた Peano 自然数と, 隠蔽ソートを用いたカウンター

```
-- Reduction of 1 + 2 = 3 (For SIMPLE-NAT)
open SIMPLE-NAT                          -- 使用するモジュールの展開
red s (0) + s( s(0)) .                   -- 項の簡約
close                                    -- モジュール展開の終了

-- ----- 処理系を用いた実行結果 ------
[1]: s 0 + s (s 0)
---> s (s 0 + s 0)
[2]: s (s 0 + s 0)
---> s (s (s 0 + 0))
[3]: s (s (s 0 + 0))
---> s (s (s 0))
s (s (s 0)) : NzNat
-- ---------------------------------

-- Proof score of associativity (For SIMPLE-NAT)
open SIMPLE-NAT
ops i j k : -> Nat .                     -- 証明で使用する定数の指定
-- Base Case .
red i + (j + 0) == (i + j) + 0 .         -- 基底段階の簡約
-- Induction hypothesis .
eq i + (j + k) = (i + j) + k .           -- 帰納法の仮定の宣言
red i + (j + (s k)) == (i + j) + (s k) .  -- 帰納段階の簡約
close

--  ------------------   処理系を用いた実行結果   ------------------------
--       基底段階                                  帰納段階
[1]: i + (j + 0) == (i + j) + 0 | [1]: i + (j + s k) == (i + j) + s k
---> i + j == (i + j) + 0       | ---> i + s (j + k) == (i + j) + s k
[2]: i + j == (i + j) + 0       | [2]: i + s (j + k) == (i + j) + s k
---> i + j == i + j             | ---> s (i + (j + k)) == (i + j) + s k
[3]: i + j == i + j             | [3]: s (i + (j + k)) == (i + j) + s k
---> true                       | ---> s ((i + j) + k) == (i + j) + s k
true : Bool                     | [4]: s ((i + j) + k) == (i + j) + s k
                                | ---> s ((i + j) + k) == s ((i + j) + k)
                                | [5]: s ((i + j) + k) == s ((i + j) + k)
                                | ---> true
                                | true : Bool
```

図 2.2: $1 + 2 = 3$ の簡約と, 結合律の証明譜に関しての実行結果

CafeOBJ コードに関する解説 (モジュール)

`mod` {`!`,`*`} "*module-name*"	CafeOBJ 仕様の記述はモジュール内で行う．モジュールは `mod` の後, 宣言するモジュールが持つ意味論を指定し, "*module name*" で宣言する．モジュールの実装は宣言文後の { "*module element*" } 内で行う．CafeOBJ はモジュールに 2 種類の意味論を持たせることができる．`!` で宣言されたモジュールはきつい意味論を持つことになり, 宣言されたモジュールは, その記述範囲内のみで成立する．これに対して, `*` で宣言されたモジュールは緩い意味論を持つことになり, 宣言されたモジュールは, 記述範囲で満足されるものならば他の全ての物で成立する．
`pr(`"*module-name*"`)`	モジュールを輸入するコマンドである．オブジェクト指向型言語の継承に類似したものである．システムは, 輸入したモジュールをそのまま利用して新しい記述を行うことが可能である．
`[`"*sort-name*"`]`	モジュール内で用いるソートを宣言する．可視ソートは, `[`"*sort-name*"`]` で宣言され, 隠蔽ソートは `*[`"*sort-name*"`]*` で宣言される．また CafeOBJ のソートには半順序関係 (partial order set) を持たせることが可能である．`[`"*subsorts*"`<` "*supersort*"`]` と記述することで, 全ての *subsort* を包括する *supersort* を持つ集合の定義が可能である．
`op`"*symbol*"`:`"*list of sort*"`->`"*sort*"	モジュールで使用するシグネチャを宣言する．シグネチャは関数の形で宣言する．`op` の後にシグネチャ名を記述する, `:` の後にシグネチャが持つ引数 (arity) のソートを記述する．`->` の後にシグネチャが返すソート (coarity) を記述する．シグネチャ名は中間表記法を実現するための特殊記号 _ を持ち, 引数で宣言するソートが順番に対応する．引数を持たないシグネチャは返却値で宣言したソートに属する定数となる．隠蔽ソートを引数に持つ物は, `bop` で宣言する．引数と返却値が同じシグネチャを複数宣言する場合は `ops` を用いる．

`var` “*name*” : “*sort-name*” .	モジュールで宣言されているソートに属する変数の宣言である. `var` の後に, 宣言する変数名を記述する. : の後にその変数が属するソートを記述する. また, 同じソートを持つ複数の変数を記述する場合には, `vars` を用いて同じように宣言する.
`eq` “*term*” `=` “*term*” .	シグネチャを用いた意味の定義である. CafeOBJ 処理系を用いた項の簡約は, ここで記述された等式を用いて行われる. 特に指定が無い場合には, 左辺から右辺への簡約を行う.
`--` “*comment*”	コメント文

CafeOBJ コードに関する解説 (証明譜)

`open` “*module-name*”	“*module-name*” を処理系で展開するためのコマンドである. ここから `close` の範囲で証明譜を記述する. 証明において, 記述者はモジュール名 “*module-name*” で記述されているソート, シグネチャを用いることが可能である. 又, 証明譜中では, `op(ops)` “*arbitrary operator*” : `->` “*sort-name*” を使用してモジュールで宣言されているソート上の任意定数を宣言し, 利用することが可能である.
`red` “*term*” (`==` “*term*”)	項の簡約を行うためのコマンドである. 項の簡約は, モジュールで定義された等式の定義に基づき, 左辺から右辺への簡約として行われる. また, ある 2 つの項に関して, それぞれの簡約結果が正しいことを評価するための演算子として, CafeOBJ が提供する `==` がある. これは, 演算子の両辺が等しいソートを持つ項に関して, お互いを評価した後, 結果を真理値で返す演算子である.
`close`	`open` で宣言されたモジュールの展開を終了するコマンドである.

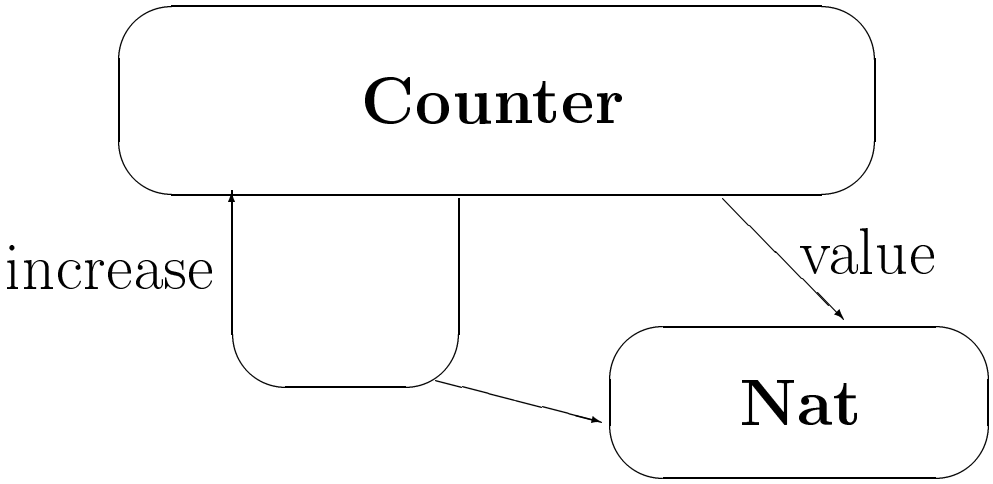

図 2.3: 抽象機械 COUNTER のグラフ

図 2.1 は Peano 自然数の公理と, その公理を用いて自然数の加算のみを扱うカウンターを記述したコード SIMPLE-NAT, COUNTER である.

SIMPLE-NAT　SIMPLE-NAT は自然数を表すソート Nat 上で全てのものが表現される. Nat は, その部分集合として Zero, NzNat を持つ. 0 は自然数 0 を表現するシグネチャであり, 定数で定義する. s _ は引数として, Nat 上に存在する値を持ち, NzNat に属する値を返す関数であり, 次の値を表現している. 自然数に関する定義はこの 2 つのシグネチャで満足される. 中間表記法を用いて表された演算子 + は, これらのシグネチャを用いて定義可能な, 加算に関しての演算子である. 加算に関する公理は, 等式 eq を用いて決定される. 等式は, Nat に属する変数 M, N を用いて記述される. Zero は自然数において基底を表すソートに意味付けられ, SIMPLE-NAT では, 0 が属するソートになる. NzNat は Zero 以外のソートを表現している. これに対しての意味付けは, Nat から NzNat への関数 s を用いて, 次の自然数を表現する際に行われる.

COUNTER　隠蔽ソート Counter と可視ソート SIMPLE-NAT 上で表現されるモジュール COUNTER は図 2.3 で表現される抽象機械で記述される. 抽象機械は, 引数の一つに隠蔽ソートを持ち可視ソートを返す観測関数と, 引数の一つに隠蔽ソートを持ち隠蔽ソートを返すアクションによって決定される. 図 2.1 では, 観測関数は, Counter の現在値を返す関数 value で表現され, アクションは, Counter に値を加算するための関数 increse で表現される. 等式 eq では, あるアクションが生じた後, 観測値がどう変化するかを記述する. この例では, アクション increse 後の Counter 値 value は, increse が引数として持っている可視ソートの値を加算して与えられる.

図 2.2 は, SIMPLE-NAT による s(0) + s (s (0)) の簡約と, 数学的帰納法を用いた演算子 + に対する結合律の証明譜である. 簡約は, 使用されるモジュールの等式を左辺から右辺への書き換え規則とみなして行われる. s(0) + s (s (0)) の簡約では, 図 2.2 上に記述されている [1] ∼ [3] のステップで簡約が行われ, 結果として自然数 3 である s (s (s 0)) : NzNat を返している.

次に帰納法を用いた結合律の証明だが, これは, ソート Nat 上の任意の定数である i, j, k のうち, どれか一つの要素について帰納法を用いることによって行われる. 2.2 では, k を選択している. 基底段階では, k を SIMPLE-NAT 上の基底である 0 として両辺を簡約している. 帰納段階においては, まず任意の k において両辺が成り立つことを等式を用いて仮定し, k の後者である s k に関して, 等式が成り立つことを簡約を用いることで行っている. 基底段階, 帰納段階両方の簡約結果は, 図 2.2 に示した通りであり. 出力結果が両方で true なので, 結合律について満たせたことになる. 今後本論文において CafeOBJ を用いて検証を行う際には, ここで示したような帰納法を主に使用する.

2.3 観測遷移機械 (OTS : Observational Transition System)

観測遷移機械 (以下 OTS) は, CafeOBJ で抽象機械を記述するための数学モデルの一つである. OTS でモデル化されたシステムでは, その任意の状態を包含する状態空間 Υ の 存在を仮定している. システムの任意の状態は, Υ の要素 υ として必ず存在している. システムを構成する興味深い値が遷移によってどのように変化するのかを, システムの外部から Υ を観測することで, システムのモデルが作成される. OTS $\mathcal{S}$ は以下の 3 つ組の集合で与えられる [3].

OTS $\mathcal{S} = \langle\ \mathcal{O}, \mathcal{I}, \mathcal{T}\ \rangle$ の定義

$\mathcal{O}$: 観測の集合	外部から Υ を観測する関数の集合である. 各観測 $o \in \mathcal{O}$ は, 状態空間 Υ から観測値である任意のデータ型 D への関数 $o : \Upsilon \to D$ である. OTS $\mathcal{S}$ 上に存在する 2 つの状態 $\upsilon_1, \upsilon_2 \in \Upsilon$ の等価性 $(\upsilon_1 = s\ \upsilon_2)$ は等式, $$(\upsilon_1 = s\ \upsilon_2) \stackrel{\mathbf{def}}{=} (\forall o \in \mathcal{O}.o(\upsilon_1) = o(\upsilon_2))$$ によって定義される. ただし, = については各 $o \in \mathcal{O}$ の値域ごとに定義されているとする.
$\mathcal{I}$: 初期状態の集合	初期状態の集合. $\mathcal{I} \subset \Upsilon$ である.

$\mathcal{T}$: 条件付遷移規則の集合	遷移規則 $\tau \in \mathcal{T}$ は, $= s$ で分類される Υ の商集合上の関数, $$\tau : \Upsilon/=s \to \Upsilon/=s$$ である. 各 $\upsilon \in \Upsilon$ に対し, $\tau(\upsilon)$ を, 同値類 $\tau([\upsilon])$ の代表元を表すものとし, υ の τ に対する事後状態と呼ぶ. 各 $\tau \in \mathcal{T}$ に付随する条件 $c_\tau : \Upsilon \to \{true, false\}$ を効力条件と呼ぶ. c_τ は以下の式を満足しなければならない. $$\forall \upsilon \in \Upsilon.((c_\tau([\upsilon]) = false) \Rightarrow (\upsilon = s\tau(\upsilon)))$$ これは, もし τ が υ に関して not であったとき τ は任意の観測に対して変化しないことを指す.

OTS の実行

観測と遷移規則の集合はそれぞれ, $o_{i_1}, ..., o_{j_m}$, $\tau_{j_1}, ..., \tau_{j_n}$ $(m, m \geq 0)$ のような対応付けが可能である. これらは全て, $k \in D_k$ $(k \in \{i_1, ...i_m, j_1, ...j_n\})$ となるようなデータ型 D_k に属していると仮定する. S の実行は以下の 3 定義を満たす状態の無限列 $\upsilon_0, \upsilon_1, ...$ である.

- 開始性 $\upsilon_o \in \mathcal{I}$ である.
- 連続性 各 $i \in \{0, 1, ...\}$ に対して $\upsilon_{v+1} = s \;\; \tau(\upsilon_i)$ を満たす $\tau \in \mathcal{T}$ が存在する.
- (公平性) 各 $\tau \in \mathcal{T}$ に関して, $\upsilon_{i+1} = s \;\; \tau(\upsilon_i)$ を満たす $i \in \{0, 1, ...\}$ が無限に存在する.

上記の定義のうち, 公平性は活性を議論する際に必要なものである. 本研究では, 活性は扱わないので, 公平性に関しては吟味しない.

到達可能状態

実行が状態 $\upsilon \in \mathcal{T}$ に存在するときに S は到達可能であると呼ぶ. S に関する全ての到達可能状態を $\mathcal{R}_s$ で表現する.

安全性の検証

安全性は, システムを作成する際に, どんな遷移を持ってきても条件を満たすような, システムの最も大切な特性である. 到達可能状態における安全性は以下の様に定義される.

$$\textbf{invariant} p \stackrel{\textbf{def}}{=} \forall \upsilon \in \mathcal{R}_s.p(\upsilon)$$

上記は, ある商集合上の状態 $\Upsilon/=s$ を引数とする述語 $p : \Upsilon/=s \to \{\text{true, false}\}$ に対して, $\upsilon \in \Upsilon$ とする際の $p(\upsilon)$ が満たされる到達可能状態中の全ての υ を **invariant** p

とする定義である．システム状態中の全ての自由変数 を $\mathbf{x}$ とすると, (**invariant** p) は $\forall \mathbf{x}.(\mathbf{invariant} p)$ に解釈される．

OTS の記述

上記で定義された OTS のモデルを CafeOBJ で記述する．

状態空間 Υ	状態空間 Υ は隠蔽ソートで表現する．例えば, `Sys` を状態空間を表現する隠蔽ソートとして表現するためには, `*[Sys]*` として記述する．
観測の集合 $\mathcal{O}$	S の観測 o_i は CafeOBJ の観測演算で与えられる．返却値のデータ型 D および引数のデータ型 D_i は始代数で定義され, 対応する可視ソートが存在すると仮定する．V で D に対応する可視ソートを, V_i で D_i に対応するスペースで区切られた可視ソートの列を表すと, CafeOBJ では次のように記述される． `bop` o : $Sys\ V_i$ `->` V
初期状態の集合 $\mathcal{I}$	初期状態は, 初期状態を表す定数を宣言し, 各観測値を等式で宣言する．定数を $init \in I$ とした場合 CafeOBJ では次の様に記述する． `op` $init$: `->` Sys ここで, 観測 o_i の初期値が $f(i)$ で与えられると仮定すると, CafeOBJ では以下の様に記述できる． `eq` $o(init, X_i)$ `=` $f(X_i)$ ここで X_i は V_i に対応する CafeOBJ 変数の列であり, $f(X_i)$ は, $f(i)$ に対応する項である．
条件付遷移規則の集合 $\mathcal{T}$	S の遷移規則 τ_j は作用演算で表現される．ここで D_j は始代数で定義され, 対応する可視ソートが存在すると仮定する．V_j で D_j に対応するスペースで区切られた可視ソートの列を表すものとする．遷移規則に対応した作用演算は, CafeOBJ では次の様に記述される．

$$\texttt{bop}\ \tau\ :\ Sys\ V_j\ \texttt{->}\ Sys$$

効力条件が真である状態における遷移規則 τ_j の適用後の観測 o_j の変化は以下の等式で記述される.

$$\texttt{ceq}\ o(\tau(W,\, X_j))\ \texttt{=}\ e - \tau(W,\, X_j)\ \texttt{if}\ c - \tau(W,\, X_j)$$

ここで W はソート Sys の CafeOBJ 変数であり, $e - a(W,\, X_j)$ は遷移規則が効力条件を満たした場合の観測 o_i の事後状態における観測値に対応する CafeOBJ の項を表す. $c - a(W,\, X_j)$ は遷移規則の効力条件である. 本研究における OTS の記述では, 効力条件は, 以下の様に記述する.

$$\texttt{op}\ c - \tau\ :\ Sys\ X_j\ \texttt{->}\ \texttt{Bool}$$

上式に対する等式は,

$$\texttt{eq}\ c - \tau(W,\, X_j)\ \texttt{=}\ conditional\ terms$$

として記述する. 効力条件が偽の場合はどの観測値も変化しないので以下の様な等式を記述すればよい.

$$\texttt{ceq}\ o(\tau(W,\, X_j))\ \texttt{=}\ o(W,\, X_j)\ \texttt{if not}\ c - \tau(W,\, X_j)$$

ここで `not` は論理否定を表す演算子である.

OTS による invariant 検証の記述

S が **invariant** p を有するとは, S に関して到達可能な全ての状態において p が成り立つことである. このため, 遷移規則の適用回数に関する帰納法により検証することが可能である. 一般に **invariant** p を単独で検証しようとすると, 帰納法の仮定が十分に強くないため, 帰納段階の証明を遂行できなくなることがある. このような場合, 別の **invariant** を補題として用いる必要がある. 補題となる **invariant** を **invariant** $p_k(k = 1, ...d)$ としてこれらもまとめて検証を行う.

まず, $p_l(l = 0, 1, 2, ...d)$ を記述するモジュールを宣言する. (本研究では `INV`) このモジュールに各 p_l を表す演算子を次の様に記述する.

$$\texttt{op}\ p_0\ :\ Sys\ V_0\ \texttt{->}\ \texttt{Bool}$$
$$.....$$
$$\texttt{op}\ p_d\ :\ Sys\ V_d\ \texttt{->}\ \texttt{Bool}$$

ここで Sys は隠蔽ソートで, V_l は, p_l に含まれる状態を表すものを除く自由変数に対応する, スペースで区切られた可視ソートの列である. またこれらの演算子を定義する等式を次の様に記述する.

$$\texttt{eq}\ p_0(W, X_0)\ \texttt{=}\ p_0$$
$$\cdots\cdots$$
$$\texttt{eq}\ p_d(W, X_d)\ \texttt{=}\ p_d$$

ここで W はソート Sys の CafeOBJ 変数であり X_l はソート列 V_l に対応するコンマで区切られた CafeOBJ 変数の列である. モジュール `INV` では, V_l に含まれる可視ソートの任意の値を宣言する定数を宣言する. その定数のコンマで区切られた列を x_l で表す. なお, 本研究では, $p(= p_0)$ となる **invariant** を `inv`, $p_k(k = 1, ...d)$ となる補題の **invariant** を `lem`k, として記述を行う.

次に, 帰納段階を記述するモジュール (本研究では `ISTEP`) を宣言する. このモジュールでは, 2 つの定数 w, w' を宣言する. w は S の任意の状態を, w' は w で表される状態で遷移規則が適用された際の事後状態を表す. 帰納段階で示すべき述語は以下の様に記述する.

$$\texttt{op}\ istep_{p_0}\ \texttt{:}\ V_0\ \texttt{->}\ \texttt{Bool}$$
$$\cdots\cdots$$
$$\texttt{op}\ istep_{p_d}\ \texttt{:}\ V_d\ \texttt{->}\ \texttt{Bool}$$

$$\texttt{eq}\ istep_{p_0}(X_0)\ \texttt{=}\ p_0(w, X_0)\ \texttt{implies}\ p_0(w', X_0)$$
$$\cdots\cdots$$
$$\texttt{eq}\ istep_{p_d}(X_d)\ \texttt{=}\ p_d(w, X_d)\ \texttt{implies}\ p_d(w', X_d)$$

`implies` は, 論理含意を表す演算子である.

続いて, **invariant** $p_l(l = 0, 1, ...d)$ 検証のための証明譜を記述する.

任意の初期状態で, p_d が成り立つことを示すためには, 以下の証明譜を記述すればよい.

```
open module-name
  red p_l (init, x_l) .
close
```

次に各帰納段階における証明譜の記述を解説する. 帰納段階の証明譜は, 分割した各空間 (遷移規則ごとに分割された空間) ごとに, CafeOBJ で宣言された等式および, 分割した各空間を特徴付ける等式, さらに必要であれば, 用いているデータ構造に関する事実を表した等式から, 等式推論により, 望みの $istep_l$ が導けることを示す. 証明譜は以下のように記述される.

```
open ISTEP
  任意の値を表す等式の宣言
  場合を特徴づける等式の宣言
  (必要であれば) 事実を表す等式の宣言
  eq w' = τ(w, x_j) .
  red istep_l(x_l) .
close
```

まず, 作用演算 τ の引数などに使う任意の値を表す定数, ならびに議論中の場合を特徴づける等式を宣言する. また, 必要であれば, 使用するデータ型の性質等を等式をして宣言する. 続いて宣言する等式は, 定数 w' が定数 w に遷移規則 τ が適用された事後状態を表すことを意味する. x_j は, ソート列 V_j に対応するコンマで区切られた定数列である. 最後に $istep_l(x_l)$ を簡約する. 簡約の結果が期待どおりであれば (ここでは `true`) この場合の証明が成功したことを意味する. もし, そうでない場合は,

場合 1. この場合に対応する空間をさらに分割する

場合 2. この場合での帰納法の仮定をさらに強める

場合 3. S は **invariant** p_l を有していない

上記のどれかを判断しなければならない. もしも [場合 2] 帰納法の仮定をさらに強める必要がある場合, 必要な仮定 (例: LEM) を選び, 項, $istep_l(x_l)$ の代わりに, 項 LEM `implies` $istep_l(x_l)$ を簡約する.

OTS に関して, 以下の例題を使用し具体的な解説を行う.

例題 : INCDEC

INCDEC は, 図 2.4 に示すように, 2 つのプロセス Inc, Dec が整数変数 x を共有して作業を行うシステムである. Inc, Dec が行う動作は以下の通りである.

Inc	x を繰り返し増加させるプロセス
Dec	x を減少させるプロセス, ただし, このプロセスは, $flag$ と呼ばれる変数が `true` のときのみ実行可能である.

システムの初期状態と動作は以下で定義される.

初期状態	$x = 0$, $flag =$ `false`
動作	inc が生じたとき, $flag$ は `true` になる. dec が生じたとき, $flag$ は `false` になる.

また, 証明したいシステムの **invariant** p は, $p = (x \geq 0)$ とする.

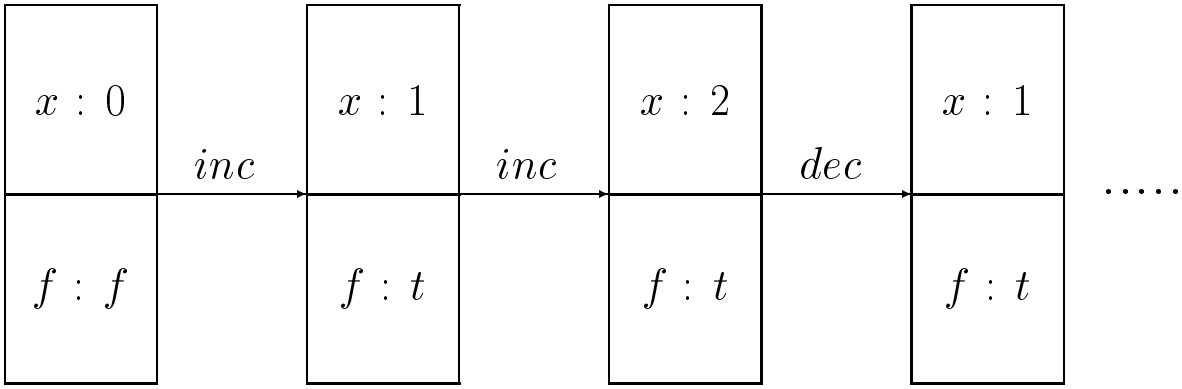

図 2.4: 例題 : INCDEC

OTS を用いた, このシステムのモデルは以下のとおりである.

Υ システムの状態空間は Sys と名付ける

$\mathcal{O}$ 観測の集合には, 共有整数を観測するための $x : Sys \to Integer$ と, $flag : Sys \to Boolean$ を加える.

$\mathcal{I}$ 初期状態の集合は, ここでは 1 つのシステムしか存在しないので $init \in \mathcal{I}$ とする.

$\mathcal{T}$ 遷移規則の集合には, x に対して増加, 減少を行うプロセス, $inc : Sys \to Sys$, $dec : Sys \to Sys$ を追加する. それぞれの効力条件 c_τ は, $c_{inc} = true$, $c_{dec} = (flag(Sys) = true)$, である.

上記でモデル化が行われたシステムを, 14 ページの規則に従って CafeOBJ で記述すると, 図 2.5 になる. ここで, `pr` を用いて輸入しているモジュール, `INT+` は, 整数が定義されたモジュールである. 初期状態に対する各観測値は, システムの初期状態の定義から `x(init) = 0, flag(init) = false` と記述する.

次に **invariant** $p = (x \geq 0)$ の検証を行う. 15 ページの規則に従うと, **invariant** を記述するモジュールと証明譜を記述すると, 図 2.6, 2.7 となる. 今回は, **invariant** p_0 を満たすために補題が 1 つ (p_1) 必要であった. さらに 補題 p_1 を満たすためには p_0 が必要であり, 相互参照を行っている. このような場合, ある述語 A, B, C に対する定義

$$A \;\Rightarrow\; (B \;\Rightarrow\; C)\; iff\; (A \cap B) \;\Rightarrow\; C$$

から導かれる新たな補題, "$p_2 = p_0\ and\ p_1$" を証明することで, 両者が共に正しいことを保証することができる.

```
mod* INCDEC {                | -- inc
  pr(INT+)                   |  op c-inc : Sys -> Bool
  *[Sys]*                    |  eq c-inc(S) = true .
-- any initial state         | --
  op init : -> Sys           |  ceq x(inc(S))    = x(S) + 1 if c-inc(S) .
-- observations              |  ceq flag(inc(S)) = true     if c-inc(S) .
  bop x    : Sys -> Int      |  ceq x(inc(S))    = x(S)     if not c-inc(S) .
  bop flag : Sys -> Bool     |  ceq flag(inc(S)) = flag(S)  if not c-inc(S) .
-- actions                   | -- dec
  bop inc : Sys -> Sys       |  op c-dec : Sys -> Bool
  bop dec : Sys -> Sys       |  eq c-dec(S) = flag(S) .
-- CafeOBJ variables         |  --
  var S : Sys                |  ceq x(dec(S))    = x(S) - 1 if c-dec(S) .
-- for any initial state     |  ceq flag(dec(S)) = false    if c-dec(S) .
  eq x(init)    = 0 .        |  ceq x(dec(S))    = x(S)     if not c-dec(S) .
  eq flag(init) = false .    |  ceq flag(dec(S)) = flag(S)  if not c-dec(S) .
                             |  }
```

図 2.5: INCDEC モジュール

```
mod INV {                          | mod ISTEP {
  pr(INCDEC)                       |  pr(INV)
-- declare invariants to prove     | -- arbitrary objects
  op inv100 : Sys -> Bool          |  ops s s' : -> Sys
  op lem110 : Sys -> Bool          |
                                   |  op istep100 : -> Bool
  op lem210 : Sys -> Bool          |  op istep110 : -> Bool
-- CafeOBJ variables               |  op istep210 : -> Bool
  var S : Sys                      |
-- define invariants to prove      | eq istep100 = inv100(s) implies inv100(s') .
  eq inv100(S) = (x(S) >= 0) .     | eq istep110 = lem110(s) implies lem110(s') .
  eq lem110(S) =                   | eq istep210 = lem210(s) implies lem210(s') .
   (flag(S) implies x(S) >= 1) .|  }
  eq lem210(S) =                   |
   inv100(S) and lem110(S) .       |
}                                  |
```

図 2.6: INV, ISTEP モジュール

```
-- 1.  基底段階
open INV
  red inv100(init) .
close
-- 2.  帰納段階
--> 1) inc(s)
--> 1.1                     | --> 1.2
open ISTEP                  | open ISTEP
-- assumptions              | -- assumptions
  eq x(s) >= 0 = true .     |   eq x(s) >= 0 = false .
-- successor state          | -- successor state
  eq s' = inc(s) .          |   eq s' = inc(s) .
  red istep100 .            |   red istep100 .
close                       | close
-------------------------------------------------------
--> 2) dec(s)
--> 2.1 flag(s) = true      | --> 2.2 flag(s) = false
open ISTEP                  | open ISTEP
-- assumptions              | -- assumptions
  eq flag(s) = true .       |   eq flag(s) = false .
  eq x(s) >= 1 = true .     | -- successor state
-- successor state          |   eq s' = dec(s) .
  eq s' = dec(s) .          |   red istep100 .
  red istep100 .            | close
close                       |
open ISTEP                  |
-- assumptions              |
  eq flag(s) = true .       |
  eq x(s) >= 1 = false .    |
-- successor state          |
  eq s' = dec(s) .          |
  red inv110(s)             |
        implies istep100 .  |
close                       |
```

図 2.7: INCDEC に対する証明譜 (inv100 のみ)

第3章　観測遷移機械の拡張

前章で紹介を行った OTS は, 離散遷移システムであるので, そのままでは連続変数を持つハイブリッドシステムを扱うことはできない. そこで本章では, 連続変数を扱うためにアクティビティを OTS に追加する.

3.1　ハイブリッド観測遷移機械 (HOTS : Hybrid Observational Transition System)

図 3.1 は, あるシステムの動作を, 横軸に時間 t, 縦軸にシステムが持つ連続変数 x を用いて表現したグラフである. このグラフでは, システムは複数のモードを持つ. 1 つのモードについて述べると, x は時間変化に従い進化し, ある一定の値に達したときに, 他の時間変化を持つモードに離散的に変化する. このように, 連続進化する値が, ある条件のもとで離散的にモードを変化させるシステムをハイブリッドシステムと呼ぶ.

ハイブリッド観測遷移機械 (以下 HOTS : Hybrid Observational Transition System) は, ハイブリッドシステムを記述可能にするために OTS を拡張した物である. 拡張は, 連続変数の導入, モードに対応する, アクティビティの追加を用いて行った. すなわちアクティビティとは, ハイブリッドシステムのアナログ動作を決定づけるものである. 以下の小節でモデルの定義とアクティビティ規則について説明を行う.

3.1.1　定義

HOTS では, ある時間における実数の値が一意に定まるモードの集合を 1 つのアスペクトとして捕らえる. さらに, 1 つのアスペクトが持つ, 離散変化から離散変化までの相をアクティビティと呼ぶ. つまり, 1 つのアスペクトは, 1 つ, 又は複数のアクティビティによって構成される. ここからの定義では, モデル化を行いたいシステムが持つアスペクトの数を定数 α ($\alpha \geq 1$) を用いて表す. さらに, i ($i = 1, 2, ... \alpha$) 番目のアスペクトが持つアクティビティの数を n_i とする.

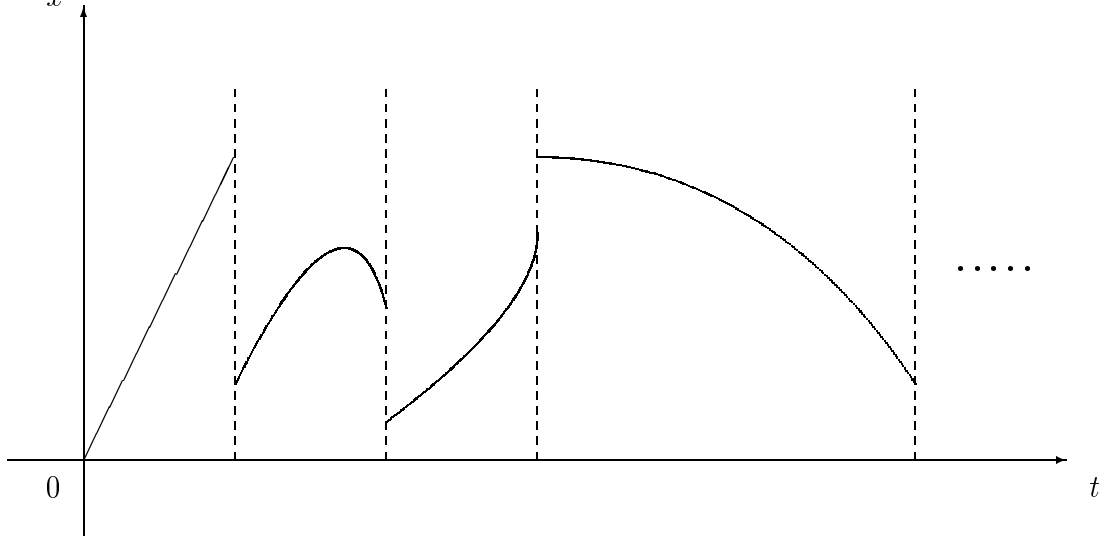

図 3.1: ハイブリッドシステムの動作

HOTS では, OTS で用いた状態空間 Υ に関しては同じ仮定を用いる. すなわち, システムの状態空間は Υ に包括されており, システムの任意の状態は $v \in \Upsilon$ として必ず存在するものとする. HOTS $\mathcal{S}$ に関する定義をこれから述べる.

HOTS $\mathcal{S} = \langle\ \mathcal{O}, \mathcal{I}, \mathcal{T} \cup \{tick_r | r \in R^+\}\ \rangle$ の定義

観測の集合 $\mathcal{O}$

観測の集合は以下の用に定義される.

$$\mathcal{O} \ = \ \mathcal{D} \ \cup \ \mathcal{P}$$

$\mathcal{D}$ は離散値を返す観測の集合であり. 定義は 12 ページで用いたものと同じである. $\mathcal{P}$ は連続変数に関する観測の集合であり, 各 $p \in \mathcal{P}$ は

$$p \ : \ \Upsilon \ \rightarrow \ C$$

で定義される. ここで C は実数と無限大を含むデータ型 $R \cup \{\infty, -\infty\}$ である. HOTS の観測では, アクティビティを表現するための特殊な観測として, 以下の 4 つを新たに加える.

$active_i^j$	$\Upsilon \rightarrow \{\text{true, false}\}$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
$value_i^j$	$\Upsilon \rightarrow R \cup \{\infty, \text{-}\infty\}$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
$boundary_i^j$	$\Upsilon \rightarrow R \cup \{\infty, \text{-}\infty\}$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
now	$\Upsilon \rightarrow R^+$

定義中の i, j はそれぞれ, アスペクト番号, アクティビティ番号に対応している. R^+ は非負の実数を表すデータ型である. これらの観測に対する規則は次節で詳細に述べる.
2 つの状態 $\upsilon \in \Upsilon$ の等価性に関しても, 12 ページで定義されたものと一緒であり, 離散変数を返す観測 o に対して,

$$(\upsilon_1 = s\ \upsilon_2) \stackrel{\mathbf{def}}{=} (\forall o \in \mathcal{O}.o(\upsilon_1) = o(\upsilon_2))$$

が満たされる場合に, 2 つの状態は等しいと定義する. つまり, 連続変数の値は状態の等価性には関わらない.

初期状態の集合 $\mathcal{I}$

初期状態の定義は, 12 ページと一緒である. ただし, 観測

$$now \quad : \Upsilon \ \rightarrow \ R^+$$

の初期値は 0 とする.

条件付遷移規則の集合 $\mathcal{T} \cup \{tick_r | r \in R^+\}$

条件付遷移規則は, 12 ページで与えた定義に基づく. それに加えて HOTS では, アクティビティに関わる遷移規則として, 以下の 2 つを新しく追加する.

$activate_i^j$	$\Upsilon \rightarrow \Upsilon$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
$deactivate_i^j$	$\Upsilon \rightarrow \Upsilon$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$

これら 2 つの遷移規則の詳細な解説は次節で行う.

条件付遷移規則の集合に含まれている, $\{\ tick_r \mid r \ \in \ R^+\ \}$ は連続変数を変化させるために必要な時間前進に関する遷移規則である. 観測 now を変化させることができるのは, この遷移規則のみである.

HOTS では, これら集合の組とは別に以下の関数を用いる.

$$\boxed{delta_i^j \quad R^+ \rightarrow R \quad , \text{where } i \in \{1, ..., \alpha\} \quad , j \in \{1, ..., n_i\}}$$

これはシステムの連続変数の進化を表現する関数であり, 常微分方程式で定義されるものである.

HTOS の実行, 到達可能状態, **invariant** に関しては, 12 ページと同じである.

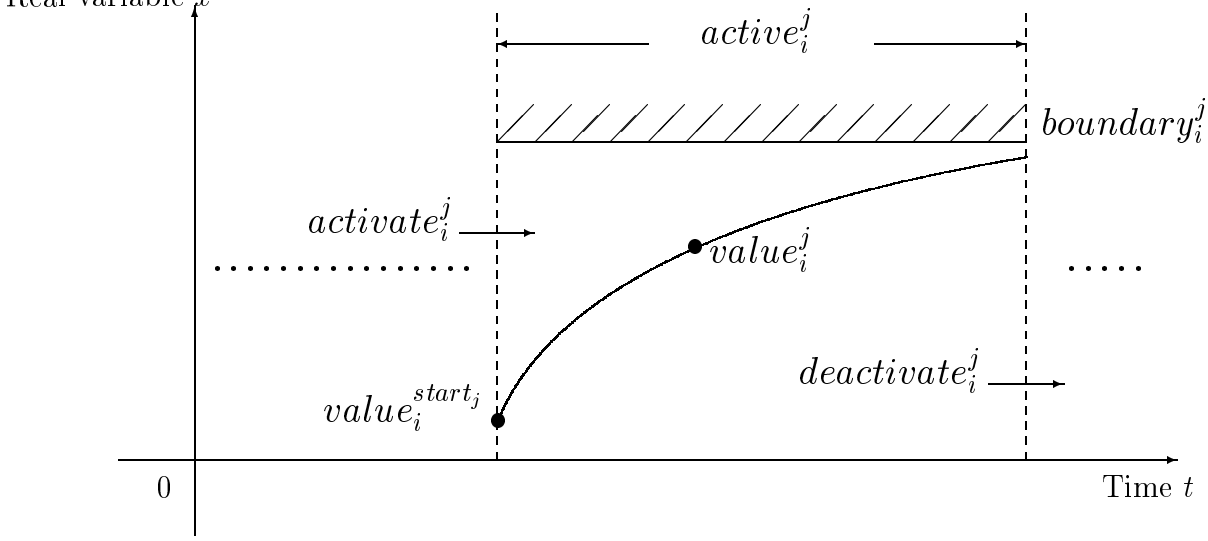

図 3.2: アクティビティルール

3.1.2 アクティビティ規則

HOTS では, アスペクト i におけるアクティビティ j は, 以下の 4 つの観測と 2 つの遷移規則, 1 つの関数を用いて定義する.

$value_i^j$	アクティビティ j が持つ値を返す観測.
$value_i^{start_j}$	アクティビティ j の初期値.
$boundary_i^j$	アクティビティ j の境界, j における $value_i^j$ は, この値を越えることはない.
$active_i^j$	アクティビティ j における $value_i^j$ が動作可能かどうかを示すフラグ.
$activate_i^j$	アクティビティ j における $value_i^j$ を動作可能にする遷移規則.
$deactivate_i^j$	動作中のアクティビティ j を動作不能にする遷移規則
$delta_i^j$	アクティビティ j 内の連続動作を決める関数.

これらの定義を図 3.2 に示す.

また, アクティビティ j における遷移規則, $activate_i^j$, $deactivate_i^j$ 実行後の各観測の値は以下の通りである. ($v \in \Upsilon$, $r \in \mathcal{R}$)

$activate_i^j$, $deactivate_i^j$ について, 最低限満たすべき効力条件は, 以下の通りである.

$activate_i^j$ ：アスペクト i に属する全てのアクティビティが動作不能である.

$deactivate_i^j$ ：アクティビティ j が動作中である.

効力条件は, 上記に対して任意のものを追加しても良い.
効力条件を満たした際の, 事後状態に対する観測の変化は, 以下の通りである.

$acrtive_i^j(activate_i^j(v))$	$=$ true
$acrtive_i^j(deactivate_i^j(v))$	$=$ false
$value_i^j(activate_i^j(v))$	$= value_i^{start_j}$
$value_i^j(deactivate_i^j(v))$	$= value_i^j(v)$
$boundary_i^j(activate_i^j(v))$	$= p$
$boundary_i^j(deactivate_i^j(v))$	$=$ (∞ or -∞)

$acrtive_i^j(activate_i^j(v)) =$ true

$activate$ を起こすと, アクティビティは動作可能状態になるので $active$ の値は true である.

$acrtive_i^j(deactivate_i^j(v)) =$ false

$deactivate$ 後は, アクティビティは動作不可能なので $active$ の値は false である.

$value_i^j(activate_i^j(v)) = value_i^{start_j}$

$activate$ が実行されるとアクティビティは動作可能になる. そのときの $value_i^j$ は $value_i^{start_j}$ に設定される.

$value_i^j(deactivate_i^j(v)) = value_i^j(v)$

$deactivate$ が実行されるとアクティビティは動作不可能になる. そのときの $value_i^j$ は $deactivate$ 実行前の値になる.

$boundary_i^j(activate_i^j(v)) = p$

$activate$ 後の $boundary$ は, システムで定義されている適当な実数 r を宣言する.

$boundary_i^j(deactivate_i^j(v)) =$ (∞ or -∞)

$deactivate$ 後の $boundary$ は ∞ か -∞ を選択する. どちらを選択するかは, 実際に記述を行うときに決める.

HOTS では, システムの連続動作は, 時間前進規則 $tick_r$ でのみ行うことが可能である. 状態 $v \in \Upsilon$ に対する $tick_r$ の条件は以下のとおりである.

$$value_i^j(tick_r(v)) \text{ が } boundary_i^j \text{ を越えない}$$

またこの時の事後状態は以下の通りである.

- もし $active_i^j(tick_r(v)) = \text{true}$ ならば $value_i^j(tick_r(v))$ を $delat_i^j$ に従って変化させる.
- もし $active_i^j(tick_r(v)) = \text{false}$ ならば $value_i^j(tick_r(v))$ は変化しない (遷移前の状態が持つ値のままである)

次章では, ハイブリッドシステムに関する例題を, 本章で提案した HOTS を用いてモデル化し, CafeOBJ による記述, 検証を行う.

第4章　例題

本章では, 前章で提案した HOTS (ハイブリッド観測遷移機械) を用いて, 2 つの例題に対するモデル化と CafeOBJ を用いた記述検証を行う. 例題は, アスペクトが 1 ($\alpha = 1$) である, サーモスタット (自動温度調節機) [16] と, アスペクトが 2 ($\alpha = 2$) であり, アスペクト同士が相互作用を及ぼし合う, 分散システムである交差点問題 [17] である.

4.1　サーモスタット (自動温度調節装置)

サーモスタットは, 温度 θ の制御を行う装置である. 上昇状態の温度は, ある温度に到達した時に停止する. そして温度は減少状態になる. 減少中の温度はある温度に達すると停止し, 今度は上昇中に切り替わる. 詳細な定義は以下の通りである. ここで $\mathcal{R}$ は実数上に存在する変数の集合である.

- 温度は変数 $\theta \in \mathcal{R}$ で与えられる.
- 温度が $\theta \leq 70$ でヒーターは on になり温度は上昇に切り替わる.
- 温度が $\theta \geq 80$ でヒーターは off になり温度は減少に切り替わる.
- ヒーターが on 時では, 温度 θ は常に $\theta < 82$ である.
- ヒーターが off 時では, 温度 θ は常に $\theta > 68$ である.
- 上昇中 (on) では, 温度は $\dot{\theta} = -\theta + 100$ に従って上昇する.
- 減少中 (off) では, 温度は $\dot{\theta} = -\theta$ に従って減少する.

○ 初期状態は, ヒーター off, $\theta = 75$ である.

図 4.1 に上記のグラフを示す.

HOTS を用いたモデル化

3 章で提案した HOTS を用いてシステムをモデル化する.

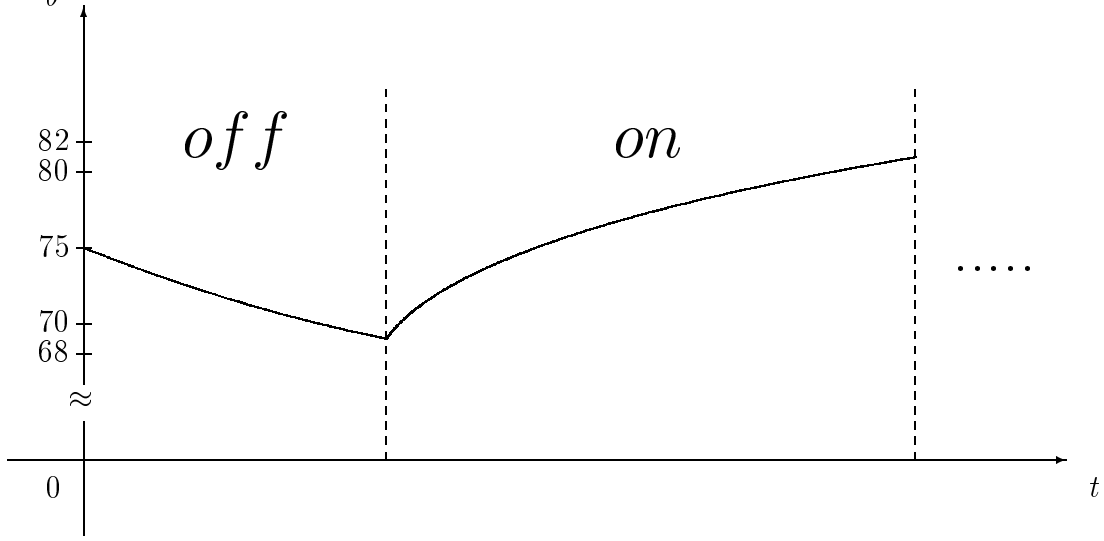

図 4.1: 例題：サーモスタット

HOTS を用いたモデル化

- HOTS のアスペクトにみなせるものは 1 つのサーモスタットしかない．従って，$\alpha = 1$ である．
- $\alpha = 1$ は, on と off の 2 つのアクティビティを持つ. 従って $n_1 = 2$ である.

$$\mathcal{S} = \langle\ \mathcal{O}, \mathcal{I}, \mathcal{T} \cup \{tick_r | r \in R^+\}\ \rangle$$

観測の集合 $\mathcal{O}$

22 ページの定義に基づき,

$active_i^j$	$\Upsilon \rightarrow \{\text{true, false}\}$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
$value_i^j$	$\Upsilon \rightarrow R \cup \{\infty, \text{-}\infty\}$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
$boundary_i^j$	$\Upsilon \rightarrow R \cup \{\infty, \text{-}\infty\}$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
now	$\Upsilon \rightarrow R^+$

をサーモスタットに対応させる.

$active_i^j$ $\star$ $active_1^{1(on)}$, $active_1^{2(off)}$ とする.
$*$ $active_1^{1(on)}$ はヒーターが on (上昇中) であるかを示すフラグである.
$*$ $active_1^{2(off)}$ は off (減少中) であるかを示すフラグである.

$value_i^j$ $\star$ $value_1^{1(on)}$, $value_1^{2(off)}$ とする.
$*$ $value_1^{1(on)}$ はヒーターが on 時の温度 θ を知る観測である.
$*$ $value_1^{2(off)}$ はヒーターが off 時の温度 θ を知る観測である.

$boundary_i^j$ $\star$ $boundary_1^{1(on)}$, $boundary_1^{2(off)}$ とする.
$*$ $boundary_1^{1(on)}$ はヒータが on 時の上限を返す観測.
$*$ $boundary_1^{2(off)}$ はヒーターが off 時の下限を返す観測.

now : 現在時刻を知るための観測

初期状態の集合 $\mathcal{I}$
22 ページのままモデル化を行う.

条件付遷移規則の集合 $\mathcal{T} \cup \{tick_r | r \in R^+\}$
22 ページの定義に基づき,

$activate_i^j$	$\Upsilon \to \Upsilon$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$
$deactivate_i^j$	$\Upsilon \to \Upsilon$, where $i \in \{1, ..., \alpha\}$, $j \in \{1, ..., n_i\}$

をサーモスタットに対応させる.

$activate_i^j$ $\star$ $activate_1^{1(on)}$, $activate_1^{2(off)}$ とする.
$*$ $activate_1^{1(on)}$ は, on モードを動作可能状態にする遷移規則である.
効力条件は, 25 ページの条件に, “ $value_1^{2(off)}$ ≤ 70 ” を加えたものである.
$*$ $activate_1^{2(off)}$ は, off モードを動作可能状態にする遷移規則である.
効力条件は, 25 ページの条件に, “ $value_1^{1(on)}$ ≥ 80 ” を加えたものである.

$deactivate_i^j$ $\star$ $deactivate_1^{1(on)}$, $deactivate_1^{2(off)}$ とする.
$*$ $deactivate_1^{1(on)}$ は on モードを動作不可能状態にする遷移規則である.
効力条件は, 25 ページの条件に, “ $value_1^{1(on)}$ ≥ 80 ” を加えたものである.
$*$ $deactivate_1^{2(off)}$ は off モードを動作不可能状態にする遷移規則である.
効力条件は, 25 ページの条件に, “ $value_1^{2(off)}$ ≤ 70 ” を加えたものである.

効力条件を満たした際の各観測値の変化は, 25 ページを参考にして, 以下の用に定義する. なお, ここで, 各 $value_i^j$ の初期値 $value_i^{start_j}$ は,

$$
\begin{aligned}
&* \ value_1^{start_{1(on)}} \text{ は, } value_1^{2(off)} \\
&* \ value_1^{start_{2(off)}} \text{ は, } value_1^{1(on)}
\end{aligned}
$$

である.

$acrtive_i^j(activate_i^j(v))$	= true
$acrtive_i^j(deactivate_i^j(v))$	= false
$value_i^j(activate_i^j(v))$	$= value_i^{start_j}$
$value_i^j(deactivate_i^j(v))$	$= value_i^j(v)$
$boundary_i^j(activate_i^j(v))$	$= p$
$boundary_i^j(deactivate_i^j(v))$	= (∞ or -∞)

上記の表をサーモスタットの例題に対応させると以下になる.

$activate_1^{1(on)}(v)$, $deactivate_1^{1(on)}(v)$	
$acrtive_1^{1(on)}(activate_1^{1(on)}(v))$	= true
$acrtive_1^{1(on)}(deactivate_1^{1(on)}(v))$	= false
$value_1^{1(on)}(activate_1^{1(on)}(v))$	$= value_1^{2(off)}$
$value_1^{1(on)}(deactivate_1^{1(on)}(v))$	$= value_1^{1(on)}(v)$
$boundary_1^{1(on)}(activate_1^{1(on)}(v))$	= 82
$boundary_1^{1(on)}deactivate_1^{1(on)}(v))$	$= \infty$
$activate_1^{2(off)}(v)$, $deactivate_1^{2(off)}(v)$	
$acrtive_1^{2(off)}(activate_1^{2(off)}(v))$	= true
$acrtive_1^{2(off)}(deactivate_1^{2(off)}(v))$	= false
$value_1^{2(off)}(activate_1^{2(off)}(v))$	$= value_1^{1(on)}$
$value_1^{2(off)}(deactivate_1^{2(off)}(v))$	$= value_1^{2(off)}(v)$
$boundary_1^{2(off)}(activate_1^{2(off)}(v))$	= 68
$boundary_1^{2(off)}deactivate_1^{2(off)}(v))$	$= -\infty$

$delta_i^j$ に関しては以下の用にモデル化する.

$$
\begin{aligned}
* \quad delta_1^{1(on)}(r) &= -e^{-r} + 100 \\
* \quad delta_1^{2(off)}(r) &= -e^{-r}
\end{aligned}
$$

上記の式は, 定義の常微分方程式の解となっている. なお, ここで e は自然対数の底である.

CafeOBJ を用いた記述

14 ページで行った記述法と, 39 ページから定義されてきたモデル化を用いて, CafeOBJ を用いたモデルの記述を行う.

◇ 観測 $active_1^{1(on)}$, $active_1^{2(off)}$, $value_1^{1(on)}$, $value_1^{2(off)}$, $boundary_1^{1(on)}$, $boundary_1^{2(off)}$, now は図 4.2 として記述する.

```
bop active1-on : Sys -> Bool
bop active2-off : Sys -> Bool
bop value1-on : Sys -> Real+
bop value2-off : Sys -> Real+
bop boundary1-on : Sys -> Timeval
bop boundary2-off : Sys -> TImeval
bop now : Sys -> Real+
```

図 4.2: 観測の記述 (サーモスタット)

図 4.2 中の `Sys` は, このシステムを表現する状態空間の名前である. `Real+` は非負の実数型を表すソートであり. `Timeval` は ∞, $-\infty$ を含む実数のソートである.

◇ 初期状態に関しては, 14 ページと同様に `init` を用いて定義する. `init` に関する各観測の初期値は, 27 ページの定義を基に, 図 4.3 として記述する.

```
eq active1-on(init) = false
eq active2-off(init) = true .
eq value1-on(init) = -oo .
eq value2-off(init) = 75 .
eq boundary1-on(init) = 82 .
eq boundary2-off(init) = 68 .
eq now(init) = 0 .
```

図 4.3: 初期値の記述 (サーモスタット)

◇ 条件付遷移規則 $activate_1^{1(on)}$, $activate_1^{2(off)}$, $deactivate_1^{1(on)}$, $deactivate_1^{2(off)}$ に関しても, 14 ページの定義を用いて図 4.4, 4.5 のように記述を行う. この図では, 効果を及ぼさない場合の遷移規則を省略してあるが, 記述法は 14 ページと全て等しい.

```
-- action activate1-on
bop activate1-on : Sys -> Sys
-- effective condition:
op c-activate1-on : Sys -> Bool
eq c-activate1-on (S:Sys) =
((value2-off(S) <= 70) and not (active1-on(S)) and not (active2-off(S))) .
-- behavior:
ceq value1-on (activate1-on (S:Sys)) = (value2-off(S))
    if c-activate1-on (S) .
ceq boundary1-on (activate1-on (S:Sys)) = (82)
    if c-activate1-on (S) .
ceq active1-on (activate1-on (S:Sys)) = (true)
    if c-activate1-on (S) .
     . . . . . 以下省略

-- set of transition: deactivate1-on
-- action deactivate1-on
bop deactivate1-on : Sys -> Sys
-- effective condition:
op c-deactivate1-on : Sys -> Bool
eq c-deactivate1-on (S:Sys) = ((80 <= value1-on(S)) and active1-on(S)) .
-- behavior:
ceq boundary1-on (deactivate1-on (S:Sys)) = (oo)
    if c-deactivate1-on (S) .
ceq active1-on (deactivate1-on (S:Sys)) = (false)
    if c-deactivate1-on (S) .
     . . . . . 以下省略
```

図 4.4: 条件付遷移規則の記述その 1(サーモスタット)

ここで注目するのは, *tick* が持つ条件である. 今回の記述では, 条件がすべて *and* で結び付く記述を行った. 従って, 全ての i, j に対して, `tick` の条件 (26 ページ) を満たす必要がある. 従って, 現在動作中ではない *value* に関しても, そのすべてが *tick* の条件を満たすような *boundary* の設定を行っている.

これに対して, もし, *tick* が持つ条件を *or* で結び付けるつもりならば, 現在動作中のアクティビティのみが `tick` の条件を満たすように *boundary* の設定を変更しなければならない.

```
 -- set of transition: activate2-off
-- action activate2-off
bop activate2-off : Sys -> Sys
-- effective condition:
op c-activate2-off : Sys -> Bool
eq c-activate2-off (S:Sys) =
((80 <= value1-on(S)) and not (active1-on(S)) and not (active2-off(S))) .
-- behavior:
ceq value2-off (activate2-off (S:Sys)) = (value1-on(S))
    if c-activate2-off (S) .
ceq boundary2-off (activate2-off (S:Sys)) = (68)
    if c-activate2-off (S) .
ceq active2-off (activate2-off (S:Sys)) = (true)
    if c-activate2-off (S) .
     . . . . . 以下省略

-- set of transition: deactivate2-off
-- action deactivate2-off
bop deactivate2-off : Sys -> Sys
op c-deactivate2-off : Sys -> Bool
eq c-deactivate2-off (S:Sys) = ((value2-off(S) <= 70) and active2-off(S)) .
-- behavior:
ceq boundary2-off (deactivate2-off (S:Sys)) = (-oo)
    if c-deactivate2-off (S) .
ceq active2-off (deactivate2-off (S:Sys)) = (false)
    if c-deactivate2-off (S) .
     . . . . . 以下省略

-- set of transitions: tick
-- action tick
bop tick : Real+ Sys -> Sys
op c-tick : Real+ Sys -> Bool
eq c-tick (R:Real+,S:Sys) =
(((value1-on(S) + delta1-on(R)) < boundary1-on(S))
and (boundary2-off(S) < (value2-off(S) - delta2-off(R)))) .
-- behavior:
ceq value1-on (tick (R:Real+,S:Sys)) =
    (if active1-on(S) then
      (value1-on(S) + delta1-on(R)) else value1-on(S) fi) if c-tick (R,S) .
ceq value2-off (tick (R:Real+,S:Sys)) =
    (if active2-off(S) then
      (value2-off(S) - delta2-off(R)) else value2-off(S) fi) if c-tick (R,S) .
ceq now (tick (R:Real+,S:Sys)) = (now(S) + R)
    if c-tick (R,S) .
     . . . . . 以下省略
```

図 4.5: 条件付遷移規則の記述その 2(サーモスタット)

CafeOBJ を用いた安全性検証

15 ページを参考に, サーモスタットの安全性検証を行う. 検証する **invariant** は,

$$\boxed{68 < \theta < 82}$$

である. これは, HOTS の記述では,

$$\boxed{68 < \bigwedge value_i^j < 82}$$

となる. CafeOBJ コードでは, 下記で記述される.

```
eq inv(S) = not ((82 <= value1-on(S)) and (value2-off(S) <= 68)
            and (value1-on(S) <= 68) and (82 <= value2-off(S))) .
```

この **invariant** に対して検証を行う. 証明は各遷移規則に対する帰納法を用いて行う. 図 4.6 は, $activate_1^{1(on)}$ の **invariant** 証明譜である. 基底段階では, 証明譜を実行した結果, 以下の 3 規則,

```
eq 68 < 75 = true .
eq -oo < 82 = true .
eq (-oo = 82) = false .
```

を新たに定義することで `true` となった. 帰納段階については, 遷移規則の条件における場合分けを行うことで `true` となった. 他の $tick$ 以外の遷移規則に関しても同様であったので, ここでは省略する. $tick$ に関しては, 図 4.7 に記述される場合分けが必要でだった. ここでは,

```
eq ((value1-on(s) - (exp(r) - 100)) < boundary1-on(s)) = true .
eq (boundary2-off(s) < (value2-off(s) - exp(r))) = true .
```

以下の 2 パターンに注目する, すなわち, 効力条件が満たされる場合である. この場合では, $value$ に関して

```
eq active1-on(s) = true .
eq active2-off(s) = true .
```

とさらに深い場合分けを行う必要がある. 上記の場合わけを行ない, 検証をすすめる際には, 3 つの補題が必要となった. 必要となった補題は以下の通りである.

```
eq lem101(S) = not ((active1-on(S)) and (active2-off(S))) .
eq lem102(S) = active1-on(S) implies (boundary1-on(S) = 82) .
eq lem103(S) = active2-off(S) implies (boundary2-off(S) = 68) .
```

補題の意味は以下の通りである.

`lem101` on, off モードが同時に生じることはない.
`lem102` on, モード時では, $boundary_1^{1(on)}$ は 82 である.
`lem103` off, モード時では, $boundary_1^{2(off)}$ は 68 である.

上記の補題を図 4.7 中の証明譜 `Patern[1-1],[1-2],[1-3]` で用いることで `tick` は `true` となり, 全ての遷移規則で **invariant** が満たせすことが示せた. なお, 証明譜において, 例えば `Patern[1-2]` の場合

```
eq ((value1-on(s) - (exp(r) - 100)) < boundary1-on(s)) = true .   ...(1)
.................................
.................................
eq (boundary1-on(s) = 82) = true .                               ...(2)
-- Namely
eq ((value1-on(s) - (exp(r) - 100)) < 82) = true .               ...(3)
```

と記述しているが, これは, (2) の場合わけにおいて, (1) を用いて (3) が明示的に示せることを記述している.

補題 `lem101, lem102, lem103` の検証に関しても同様の手法で証明譜を記述した結果, 場合分けのみで証明を完了させることができた.

```
 --> I) 基底段階
open INV
  red inv(init) .
close
--> II) 帰納段階                          |
-- activate1-on のみ                      |
--> Patern[1]                             | --> Patern[5]
open ISTEP                                | open ISTEP
-- assumptions                            | -- assumptions
  eq active1-on(s) = true .               |  eq active1-on(s) = false .
  eq active2-off(s) = true .              |  eq active2-off(s) = true .
  eq (value2-off(s) <= 70) = true .       |  eq (value2-off(s) <= 70) = true .
-- successor state                        | -- successor staet
  eq s' = activate1-on(s) .               |   eq s' = activate1-on(s) .
  red istep1 .                            |  red istep1 .
close                                     | close
--> Patern[2]                             | --> Patern[6]
open ISTEP                                | open ISTEP
-- assumptions                            | -- assumptions
  eq active1-on(s) = true .               |  eq active1-on(s) = false .
  eq active2-off(s) = true .              |  eq active2-off(s) = true .
  eq (value2-off(s) <= 70) = false .      |  eq (value2-off(s) <= 70) = false .
-- successor state                        | -- successor state
  eq s' = activate1-on(s) .               |  eq s' = activate1-on(s) .
  red istep1 .                            |  red istep1 .
close                                     | close
--> Patern[3]                             | --> Patern[7]
open ISTEP                                | open ISTEP
-- assumptions                            | -- assumptions
  eq active1-on(s) = true .               |  eq active1-on(s) = false .
  eq active2-off(s) = false .             |  eq active2-off(s) = false .
  eq (value2-off(s) <= 70) = true .       |  eq (value2-off(s) <= 70) = true .
-- successor state                        | -- successor state
  eq s' = activate1-on(s) .               |  eq s' = activate1-on(s) .
  red istep1 .                            |  red istep1 .
close                                     | close
--> Patern[4]                             | --> Patern[8]
open ISTEP                                | open ISTEP
-- assumptions                            | -- assumptions
  eq active1-on(s) = true .               |  eq active1-on(s) = false .
  eq active2-off(s) = false .             |  eq active2-off(s) = false .
  eq (value2-off(s) <= 70) = false .      |  eq (value2-off(s) <= 70) = false .
-- successor state                        | -- successor state
  eq s' = activate1-on(s) .               |  eq s' = activate1-on(s) .
  red istep1 .                            |  red istep1 .
close                                     | close
```

図 4.6: 証明譜 1 (activate1-on)

```
-- assumptions
-- -[1]
-- eq ((value1-on(s) + delta1-on(r)) < boundary1-on(s)) = true .
-- eq (boundary2-off(s) < (value2-off(s) - delta2-off(r))) = true .
-- Namely
  eq ((value1-on(s) + ((c * exp(- r)) - 100)) < boundary1-on(s)) = true .
  eq (boundary2-off(s) < (value2-off(s) - (c * exp(- r)))) = true .
-- --[2]
   eq active1-on(s) = true .
   eq active2-off(s) = true .
-- successor state
  eq s' = tick(r,s) .
  red lem101(s) implies istep1 .
close
--> Patern[1-2]
open ISTEP
-- -[1]
  eq ((value1-on(s) + ((c * exp(- r)) - 100)) < boundary1-on(s)) = true .
  eq (boundary2-off(s) < (value2-off(s) - (c * exp(- r)))) = true .
-- --[2]
   eq active1-on(s) = true .
   eq active2-off(s) = false .
-- ---[3]
    eq (boundary1-on(s) = 82) = true .
-- Namely
    eq ((value1-on(s) + ((c * exp(- r)) - 100)) < 82) = true .
  eq s' = tick(r,s) .
  red lem102(s) implies istep1 .
close
--> Patern[1-3]
open ISTEP
-- -[1]
  eq ((value1-on(s) + ((c * exp(- r)) - 100)) < boundary1-on(s)) = true .
  eq (boundary2-off(s) < (value2-off(s) - (c * exp(- r)))) = true .
-- --[2]
   eq active1-on(s) = false .
   eq active2-off(s) = true .
-- ---[3]
    eq (boundary2-off(s) = 68) = true .
-- Namely
    eq (68 < (value2-off(s) - (c * exp(- r)))) = true .
  eq s' = tick(r,s) .
  red lem103(s) implies istep1 .
close
--> Patern[1-4]
open ISTEP
-- -[1]
  eq ((value1-on(s) + ((c * exp(- r)) - 100)) < boundary1-on(s)) = true .
  eq (boundary2-off(s) < (value2-off(s) - (c * exp(- r)))) = true .
-- --[2]
   eq active1-on(s) = false .
   eq active2-off(s) = false .
  eq s' = tick(r,s) .
  red istep1 .
close
--> Patern[2]                                  | --> Patern[4]
open ISTEP                                    | open ISTEP
  eq ((value1-on(s) + ((c * exp(- r)) - 100))|  eq ((value1-on(s) + ((c * exp(- r)) - 100))
    < boundary1-on(s)) = true .               |   < boundary1-on(s)) = false .
  eq (boundary2-off(s)                        |  eq (boundary2-off(s)
 < (value2-off(s) - (c * exp(- r)))) = false |   < (value2-off(s) - (c * exp(- r)))) = false .
  eq s' = tick(r,s) .                         |  eq s' = tick(r,s) .
  red istep1 .                                |  red istep1 .
close                                         |close
--> Patern[3]                                 |
open ISTEP                                    |
  eq ((value1-on(s) + ((c * exp(- r)) - 100))|
    < boundary1-on(s)) = false .              |
  eq (boundary2-off(s)                        |
   < (value2-off(s) - (c * exp(- r))))= true |
  eq s' = tick(r,s) .                         |
  red istep1 .                                |

close                                         |
```

図 4.7: 証明譜 2 (tick)

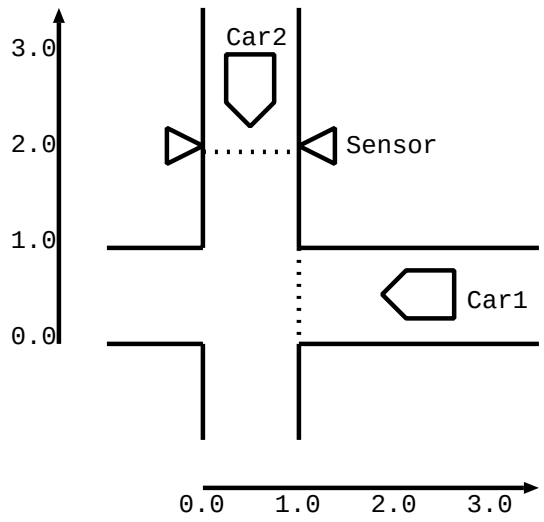

図 4.8: 例題：交差点問題

4.2 交差点問題

前節では, アスペクトが 1 ($\alpha = 1$) の場合における例題を解説した. 本節では, アスペクトが 2 ($\alpha = 2$) のケースにおける交差点制御システムの例題を取り扱う. 交差点問題は, 左方向に交差点へ進入してくる車 (車 1) と, 下方向に交差点へ進入してくる車 (車 2) の制御を行うシステムである. (図 4.8 参照) システムの定義を以下に示す.

- 車 1, 2 それぞれの位置は, 変数 $\{x_{1(car1)}, x_{2(car2)}\} \subset \mathcal{R}$ で与えられる.
- $x_{1(car1)}$ は, 交差点の出口を 0 とし, 右方向に延びる軸上に存在する.
- $x_{2(car2)}$ は, 交差点の出口を 0 とし, 上方向に延びる軸上に存在する.
- 車 2 は, 交差点進入について車 1 より高い優先順位を持っており,
  常に進行することが可能である.
- 車 2 の位置 $x_{2(car2)} = 2$ にはセンサーがある.
  センサーは, 車 2 がセンサーを通過すると作動し, 車 2 が交差点を通過すると停止する.
- 車 1 は, 位置 $x_{1(car1)} = 1$ で交差点に進行するか停止するか指示をうける.
  ▷ 進行する場合：センサーが軌道していない場合
  ▷ 停止する場合：センサーが軌道している場合
- 車 1 は, 位置 $x_{1(car1)} = 0$ で交差点から出る.
- 車 2 は, 位置 $x_{2(car2)} = 0$ で交差点から出る.
- 車 1 の速度は, $\dot{x}_{1(car1)} = -1$ である.
- 車 2 の速度は, $\dot{x}_{2(car2)} = -1$ である.
- 車 1, 車 2 共に 1 次元で与えられるものとする. すなわち車の大きさは考慮しない.
- 車 1, 車 2 共に加速度は一定とする. すなわち, 加速減速に関しては考慮しない.
- 交差点問題では, 3 を座標の最大値とする. すなわち交差点が制御可能なのは, 3 以下の車である.
- ○ 初期状態は, $x_{1(car1)} = 3$, $x_{2(car2)} = 3$ である.

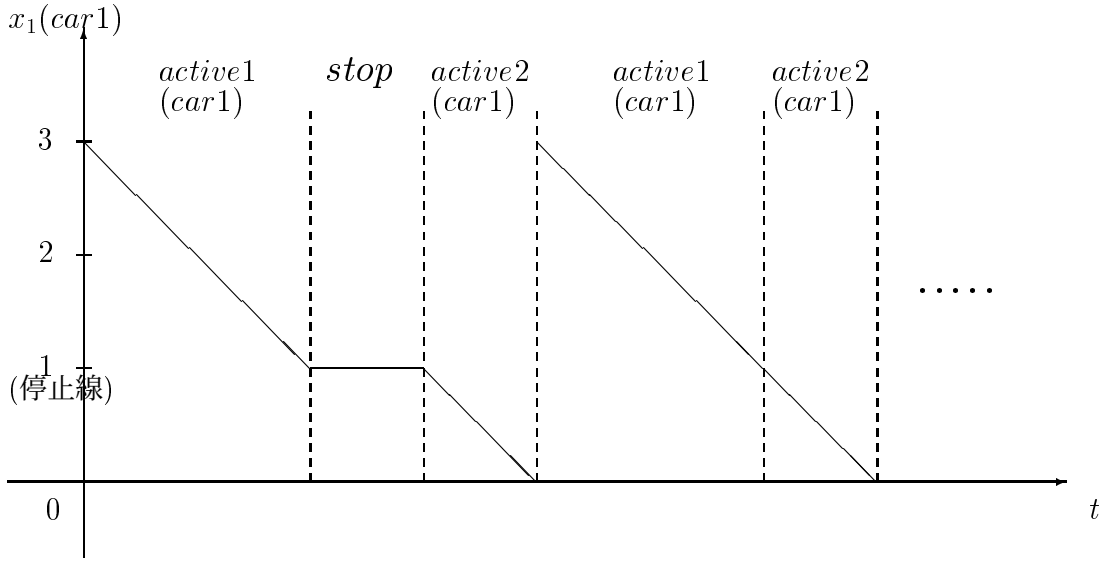

図 4.9: 車 1 のアクティビティ

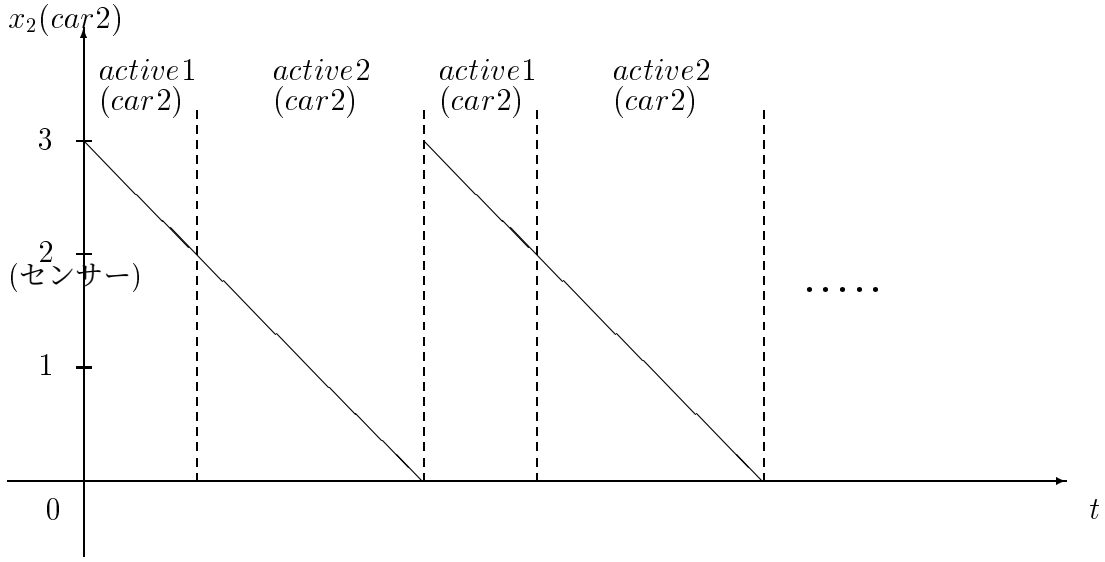

図 4.10: 車 2 のアクティビティ

交差点問題における, 車 1, 車 2 それぞれの動作例を図 4.9, 4.10 に記述した. 図中の *active* モードに関しては, 次の HOTS を用いたモデル化で解説する.

HOTS を用いたモデル化

39 ページと同様の手順で, 交差点問題をモデル化する.

- 車 1 と 車 2 は別のアスペクトである. 従って, $\alpha = 2$ である. 1 番目のアスペクトは車 1 , 2 番目のアスペクトは車 2 がそれぞれ対応する.
- 1 番目のアスペクト (car1) は停止線に到達するまでのモード $active1(car1)$ と, 停止線を越えて交差点に進入するモード $active2(car1)$ を持つ. (図 4.9 参照) 従って, $n_1 = 2$ である.
- 2 番目のアスペクト (car2) はセンサーに到達するまでのモード $active1(car2)$ と, 交差点に進入するモード $active2(car2)$ を持つ. (図 4.10 参照). 従って, $n_2 = 2$ である.

39 ページと同様に, モデル化を行う.

観測の集合 $\mathcal{O}$

$active_i^j$ ⋆ $active_1^{1(active1)}$, $active_1^{2(active2)}$, $active_2^{1(active1)}$, $active_2^{2(active2)}$ とする.
∗ $active_1^{1(active1)}$ は車 1 が $active1(car1)$ であることを示すフラグである.
∗ $active_1^{2(active2)}$ は車 1 が $active2(car1)$ であることを示すフラグである.
∗ $active_2^{1(active1)}$ は車 2 が $active1(car2)$ であることを示すフラグである.
∗ $active_2^{2(active2)}$ は車 2 が $active2(car2)$ であることを示すフラグである.

$value_i^j$ ⋆ $value_1^{1(active1)}$, $value_1^{2(active2)}$, $value_2^{1(active1)}$, $value_2^{2(active2)}$ とする.
∗ $value_1^{1(active1)}$ は車 1 が $active1(car1)$ 時の位置を知る観測である.
∗ $value_1^{2(active2)}$ は車 1 が $active2(car1)$ 時の位置を知る観測である.
∗ $value_2^{1(active1)}$ は車 2 が $active1(car2)$ 時の位置を知る観測である.
∗ $value_2^{2(active2)}$ は車 2 が $active2(car2)$ 時の位置を知る観測である.

$boundary_i^j$ ⋆ $boundary_1^{1(active1)}$, $boundary_1^{2(active2)}$, $boundary_2^{1(active1)}$, $boundary_2^{2(active2)}$ とする.
∗ $boundary_1^{1(active1)}$ は車 1 が $active1(car1)$ 時の上限を返す観測である.
∗ $boundary_1^{2(active2)}$ は車 1 が $active2(car1)$ 時の上限を返す観測である.
∗ $boundary_2^{1(active1)}$ は車 2 が $active1(car2)$ 時の上限を返す観測である.
∗ $boundary_2^{2(active2)}$ は車 2 が $active2(car2)$ 時の上限を返す観測である.

now : 現在時刻を知るための観測である.

初期状態の集合 $\mathcal{I}$

22 ページのままモデル化を行う.

条件付遷移規則の集合 $\mathcal{T} \cup \{tick_r | r \in R^+\}$

$activate_i^j$

$\star$ $activate_1^{1(active1)}$, $activate_1^{2(active2)}$, $activate_2^{1(active1)}$, $activate_2^{2(active2)}$ とする.

$*$ $activate_1^{1(active1)}$ は, $active1(car1)$ モードを動作可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “ $value_1^{2(active2)} = 0$” を加えたものである.

$*$ $activate_1^{2(active2)}$ は, $active2(car1)$ モードを動作可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “$value_1^{1(active1)} = 1$” と “$\boldsymbol{active_2^{2(active2)} = false}$” を加えたものである.

$*$ $activate_2^{1(active1)}$ は, $active1(car2)$ モードを動作可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “ $value_2^{2(active2)} = 0$” を加えたものである.

$*$ $activate_2^{2(active2)}$ は, $active2(car2)$ モードを動作可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “ $value_1^{2(active2)} = 2$” を加えたものである.

$deactivate_i^j$

$\star$ $deactivate_1^{1(active1)}$, $deactivate_1^{2(active2)}$, $deactivate_2^{1(active1)}$, $deactivate_2^{2(active2)}$ とする.

$*$ $deactivate_1^{1(active1)}$ は $active1(car1)$ モードを動作不可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “$value_1^{1(active1)} = 1$” を加えたものである.

$*$ $deactivate_1^{2(active2)}$ は $active2(car1)$ モードを動作不可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “$value_1^{2(active2)} = 0$” を加えたものである.

$*$ $deactivate_2^{1(active1)}$ は $active1(car2)$ モードを動作不可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “$value_2^{1(active1)} = 2$” を加えたもので

ある.

$*$ $deactivate_2^{2(active2)}$ は $active2(car2)$ モードを動作不可能状態にする遷移規則である.

効力条件は, 25 ページの条件に, “$value_2^{2(active2)} = 0$” を加えたものである.

また, 遷移後の各観測の変化は下記の通りである.

$activate_1^{1(active1)}(v)$, $deactivate_1^{1(active1)}(v)$	
$acrtive_1^{1(active1)}(activate_1^{1(active1)}(v))$	$=$ true
$acrtive_1^{1(active1)}(deactivate_1^{1(active1)}(v))$	$=$ false
$value_1^{1(active1)}(activate_1^{1(active1)}(v))$	$= 3$
$value_1^{1(active1)}(deactivate_1^{1(active1)}(v))$	$= value_1^{1(active1)}(v)$
$boundary_1^{1(active1)}(activate_1^{1(active1)}(v))$	$= 1$
$boundary_1^{1(active1)}deactivate_1^{1(active1)}(v))$	$= -\infty$
$activate_1^{2(active2)}(v)$, $deactivate_1^{2(active2)}(v)$	
$acrtive_1^{2(active2)}(activate_1^{2(active2)}(v))$	$=$ true
$acrtive_1^{2(active2)}(deactivate_1^{2(active2)}(v))$	$=$ false
$value_1^{2(active2)}(activate_1^{2(active2)}(v))$	$= 1$
$value_1^{2(active2)}(deactivate_1^{2(active2)}(v))$	$= value_1^{2(active2)}(v)$
$boundary_1^{2(active2)}(activate_1^{2(active2)}(v))$	$= -\infty$
$boundary_1^{2(active2)}deactivate_1^{2(active2)}(v))$	$= -\infty$
$activate_2^{1(active1)}(v)$, $deactivate_2^{1(active1)}(v)$	
$acrtive_2^{1(active1)}(activate_2^{1(active1)}(v))$	$=$ true
$acrtive_2^{1(active1)}(deactivate_2^{1(active1)}(v))$	$=$ false
$value_2^{1(active1)}(activate_2^{1(active1)}(v))$	$= 3$
$value_2^{1(active1)}(deactivate_2^{1(active1)}(v))$	$= value_2^{1(active1)}(v)$
$boundary_2^{1(active1)}(activate_2^{1(active1)}(v))$	$= 2$
$boundary_2^{1(active1)}deactivate_2^{1(active1)}(v))$	$= -\infty$
$activate_2^{2(active2)}(v)$, $deactivate_2^{2(active2)}(v)$	
$acrtive_2^{2(active2)}(activate_2^{2(active2)}(v))$	$=$ true
$acrtive_2^{2(active2)}(deactivate_2^{2(active2)}(v))$	$=$ false
$value_2^{2(active2)}(activate_2^{2(active2)}(v))$	$= 2$
$value_2^{2(active2)}(deactivate_2^{2(active2)}(v))$	$= value_2^{2(active2)}(v)$
$boundary_2^{2(active2)}(activate_2^{2(active2)}(v))$	$= 0$
$boundary_2^{2(active2)}deactivate_2^{2(active2)}(v))$	$= -\infty$

ここで, $boundary_2^{2(active2)}(activate_2^{2(active2)}(v)) = 0$ としているのは, センサーが交差点をでた時点で停止することを表現したいためである.

$delta_i^j$ に関しては以下の用にモデル化する.

$$
\begin{array}{lll}
* & delta_1^{1(active1)}(r) & = & -r \\
* & delta_1^{2(active2)}(r) & = & -r \\
* & delta_2^{1(active1)}(r) & = & -r \\
* & delta_2^{2(active2)}(r) & = & -r
\end{array}
$$

CafeOBJ を用いた記述

◇ 観測 $active_1^{1(active1)}$, $active_1^{2(active2)}$, $active_2^{1(active1)}$, $active_2^{2(active2)}$ は, 図 4.11 として記述される.

```
bop car1value : Sys -> Real+
bop car2value : Sys -> Real+
bop car1boundary1 : Sys -> Timeval
bop car1boundary2 : Sys -> Timeval
bop car2boundary1 : Sys -> Timeval
bop car2boundary2 : Sys -> Timeval
bop car1active1 : Sys -> Bool
bop car1active2 : Sys -> Bool
bop car2active1 : Sys -> Bool
bop car2active2 : Sys -> Bool
bop now : Sys -> Real+
```

図 4.11: 観測の記述 (交差点)

$value_i^j$ について, 本例題では, 1 アスペクト内の全アクティビティでの常微分方程式が等しいことより, 1 アスペクトの全アクティビティについて, 統一した $value$ を用いている.

◇ 各観測に対しての初期値は, 図 4.12 となる.

```
eq car1value(init) = 3 .
eq car1active1(init) = true .
eq car1boundary1(init) = 1 .
eq car1active2(init) = false .
eq car1boundary2(init) = -oo .
eq car2value(init) = 3 .
eq car2active1(init) = true .
eq car2boundary1(init) = 2 .
eq car2active2(init) = false .
eq car2boundary2(init) = -oo .
eq now(init) = 0 .
```

図 4.12: 初期値の記述 (交差点)

条件付遷移規則に関しては, 図 4.13 となる. 記述テクニックは, 全てが 32 ページのサーモスタットの例題と等しいため, ここでは省略する.

```
-- action car1-activate1
bop car1-activate1 : Sys -> Sys
-- effective condition:
op c-car1-activate1 : Sys -> Bool
eq c-car1-activate1 (S:Sys) =
 ((car1value(S) <= 0) and (not car1active2(S)) and (not car1active1(S))) .
-- behavior:
ceq car1value (car1-activate1 (S:Sys)) = (3)
    if c-car1-activate1 (S) .
ceq car1boundary1 (car1-activate1 (S:Sys)) = (1)
    if c-car1-activate1 (S) .
ceq car1active1 (car1-activate1 (S:Sys)) = (true)
    if c-car1-activate1 (S) .
.......... 以下省略
```

図 4.13: 条件付遷移規則の記述 (交差点)

CafeOBJ を用いた安全性検証

15 ページを参考に, サーモスタットの安全性検証を行う. 検証したい **invariant** は,

車 1 と車 2 が同時に交差点に進入することはない

である. これは, HOTS の記述では,

$$\boxed{0 < \bigwedge value_i^j < 1}$$

となる. CafeOBJ コードでは, 下記となる.

```
eq inv(S) = not ((car2value(S) < 1) and (0 < car2value(S))
                 and (car1value(S) < 1) and (0 < car1value(S))) .
```

上記の **invariant** に対して, 検証を行った結果, サーモスタットの例と同様に $tick$ に関する補題が必要となった. 最終的に必要となった補題は以下の 7 つである.

```
eq lem101(S) = not (car1active1(S) and car1active2(S)) .
eq lem102(S) = not (car2active1(S) and car2active2(S)) .
eq lem103(S) = car1active1(S) implies (car1boundary1(S) = 1) .
eq lem104(S) = (not car2active1(S)) and (not car2active2(S)) implies
             ((car2value(S) = 0) or (car2value(S) = 2)) .
eq lem105(S) = car2active1(S) implies (car2boundary1(S) = 2) .
eq lem106(S) = (not car1active1(S)) and (not car1active2(S)) implies
             ((car1value(S) <= 0) or (car1value(S) = 1)) .
eq lem107(S) = (car1active2(S) and car2active2(S)) implies
             ((car1value(S) < 0) or (1 <= (car2value(S) - car1value(S)))) .
```

補題の意味は以下の通りである.

- `lem101` 1 番目のアスペクト (車 1) における全てのアクティビティが同時に動作可能にならない.
- `lem102` 2 番目のアスペクト (車 2) における全てのアクティビティが同時に動作可能にならない.
- `lem103` $active1(car1)$ ならば $boundary_1^{1(active1)}$ は 1 である.
- `lem104` 2 番目のアスペクト (車 2) において, 全てのアクティビティが動作不能ならば, 車 2 の位置は 0 か 2 どちらかである.
- `lem105` $active1(car2)$ ならば $boundary_2^{1(active1)}$ は 2 である.
- `lem106` 1 番目のアスペクト (車 1) において, 全てのアクティビティが動作不能ならば, 車 1 の位置は 0 以下 か 1 どちらかである.
- `lem107` 以下で詳しく解説

▷ 補題 `lem107` は, 記述者が補題を推測する上で一番手間を要するものなので, ここで詳しく解説を行っておく. この補題の意味は, "車 *1* が $active2(car1)$ モード, かつ 車 *2* が $active2(car2)$ モードである際, 車 *1* の位置は, 0 より小さいか, もしくは, 車 *2* の位置 から車 *1* の位置を引いた値が 1 以上" である. よりくだけた解説を行うと, "車 *1* が停止線を越えており, 車 *2* がセンサーを越えている場合, 車 *1* は交差点を越えているか, もしくは, 両者の位置の差の絶対値は 1 以上である. " **両者の位置の差の絶対値は 1 以上である**

という推測は少々困難であるが, これは, 両アクティビティ ($active2(car1)$, $active2(car2)$) の初期値の差と, $active2(car1)$ が動作可能になるための条件から推測できた. 証明譜中で `lem 107` が必要になる場合を図 4.14 に示す. 図中の `Patern[1-11-3]` において

```
--> Patern[1-11-3](Used Lemma7)
open ISTEP
-- assumptions
-- -[1]
eq (car1boundary1(s) <= (car1value(s) - r)) = true .
   ....................
-- ---[3]
eq (car1value(s) < 0) = false .
eq (1 <= (car2value(s) - car1value(s))) = true .                  ..... (1)
-- Namely
-- eq ((car2value(s) - car1value(s)) < 1) = false .               ..... (2)
-- Namely
-- eq (((car2value(s) - r) - (car1value(s) - r)) < 1) = false .   ..... (3)
-- Namely
eq ((car2value(s) - r) < 1) = false .                             ..... (4)
   ...................
```

と記述されているが, これは, (1) の等式が正しいならば, 正の実数 a, b, c に関して,

$$a \leq b$$

ならば,

$b < a$ になることはない

となる規則を適用した結果 (2) になり, (2) に関して,

$$a < b$$

ならば

$$(a - c) < b$$

となる規則を用いて (3) になり, さらに (3) より,

$$(a - b) < c$$

ならば

$$a < c$$

となる規則を用いて (4) を導くことを記述者が推測した結果である.

他の全ての補題に関しても, サーモスタットの例題と同様にして, 証明を行うことが可能である.

```
--> Patern[1-11-1](Used Lemma7)
open ISTEP
-- assumptions
-- -[1]
  eq (car1boundary1(s) <= (car1value(s) - r)) = true .
  eq (car1boundary2(s) <= (car1value(s) - r)) = true .
  eq (car2boundary1(s) <= (car2value(s) - r)) = true .
  eq (car2boundary2(s) <= (car2value(s) - r)) = true .
-- Namely
-- eq ((car1value(s) - r) < car1boundary1(s)) = false .
-- eq ((car1value(s) - r) < car1boundary2(s)) = false .
-- eq ((car2value(s) - r) < car2boundary1(s)) = false .
-- eq ((car2value(s) - r) < car2boundary2(s)) = false .
-- --[2]
    eq car1active1(s) = false .
    eq car1active2(s) = true .
    eq car2active1(s) = false .
    eq car2active2(s) = true .
-- ---[3]
     eq (car1value(s) < 0) = true .
     eq (1 <= (car2value(s) - car1value(s))) = true .
-- successor state
  eq s' = tick(r, s) .
  red lem107(s) implies istep1 .
close

--> Patern[1-11-2](Used Lemma7)
open ISTEP
-- assumptions
-- -[1]
  eq (car1boundary1(s) <= (car1value(s) - r)) = true .
  eq (car1boundary2(s) <= (car1value(s) - r)) = true .
  eq (car2boundary1(s) <= (car2value(s) - r)) = true .
  eq (car2boundary2(s) <= (car2value(s) - r)) = true .
-- --[2]
    eq car1active1(s) = false .
    eq car1active2(s) = true .
    eq car2active1(s) = false .
    eq car2active2(s) = true .
-- ---[3]
     eq (car1value(s) < 0) = true .
     eq (1 <= (car2value(s) - car1value(s))) = false .
-- successor state
  eq s' = tick(r, s) .
  red lem107(s) implies istep1 .
close
```

```
--> Patern[1-11-3](Used Lemma7)
open ISTEP
-- assumptions
-- -[1]
  eq (car1boundary1(s) <= (car1value(s) - r)) = true .
  eq (car1boundary2(s) <= (car1value(s) - r)) = true .
  eq (car2boundary1(s) <= (car2value(s) - r)) = true .
  eq (car2boundary2(s) <= (car2value(s) - r)) = true .
-- --[2]
    eq car1active1(s) = false .
    eq car1active2(s) = true .
    eq car2active1(s) = false .
    eq car2active2(s) = true .
-- ---[3]
      eq (car1value(s) < 0) = false .
      eq (1 <= (car2value(s) - car1value(s))) = true .
-- Namely
--  eq ((car2value(s) - car1value(s)) < 1) = false .
-- Namely
--  eq (((car2value(s) - r) - (car1value(s) - r)) < 1) = false .
-- Namely
eq ((car2value(s) - r) < 1) = false .
-- successor state
  eq s' = tick(r, s) .
  red lem107(s) implies istep1 .
close

--> Patern[1-11-4](Used Lemma7)
open ISTEP
-- assumptions
-- -[1]
  eq (car1boundary1(s) <= (car1value(s) - r)) = true .
  eq (car1boundary2(s) <= (car1value(s) - r)) = true .
  eq (car2boundary1(s) <= (car2value(s) - r)) = true .
  eq (car2boundary2(s) <= (car2value(s) - r)) = true .
-- --[2]
    eq car1active1(s) = false .
    eq car1active2(s) = true .
    eq car2active1(s) = false .
    eq car2active2(s) = true .
-- ---[3]
       eq (car1value(s) < 0) = false .
       eq (1 <= (car2value(s) - car1value(s))) = false .
-- successor state
  eq s' = tick(r, s) .
  red lem107(s) implies istep1 .
close
```

図 4.14: 証明譜 (交差点: *tick*, `lem 107` が必要な場合)

第5章　結論

本章では, まず, 3 章, 4 章で述べた HOTS 提案手法と例題への適用についてまとめる. 次に, 関連研究の紹介を行う. 最後に, 本研究で得られた知見にから, 今後の課題を述べる.

5.1　研究成果

本研究では, 連続事象と離散事象が混合されたハイブリッドシステムに対して, ハイブリッドシステムの検証可能な数学モデルを提案するために, まず OTS にアナログ事象を制御するためのアクティビティを追加した (HOTS の提案). 次に, HOTS の有効性を確かめるため, 2 つの例題に対して, HOTS を用いたモデル化, CafeOBJ を用いた記述, 検証を行った. 以下に研究成果を示す.

- 新たに提案した数学モデル (HOTS) は, 2 つの例題に限れば, 記述, 検証が可能であった.
- HOTS は, 複数システムが合成されたもの (アスペクト $\alpha \geq 2$) つまり, 方程式が複数混在し, 相互作用を及ぼし合うシステムに対しても例題に限れば記述が可能であった.

本研究の例題から CafeOBJ で検証が行えるクラスには, ある制限があることも判明した. 以下に問題点を示す.

- CafeOBJ を用いた記述に関して, 常微分方程式は, 定量解が定まっているもの (あるいは保証された近似解が存在するもの) しか現段階では記述ができない.
- CafeOBJ が検証可能なものは, 1 階線形常微分方程式に限られる.
- $\alpha \geq 2$ となるシステムで, 相互作用を及ぼし合うシステムの検証では, 合成されたシステムの挙動を記述者がある程度予測する必要がある. (47 ページの `lem107` 参照) 合成システムは定性的にシステムの挙動を理解することは難しいことから, 検証支援系として強力ではない.

2 番目の問題点に関しては, 5.2.3 節 [16] で, 解決策となる指針が示されている. 問題点に対する所見は, 5.3 節で述べる.　また, 5.2.2 節から, ハイブリッドシステムを記述するためのモデルの 1 つであるハイブリッドオートマトンを用いて記述された, サーモスタット

例題の定義を, HOTS の定義で解釈することが可能であった. このことから, 意味的なモデルであるハイブリッドオートマトンを, HOTS を用いることで, CafeOBJ で記述し, 検証支援系を用いた簡約による証明を行える可能性が示唆できた.

5.2 関連研究

本節では, ハイブリッドシステムのモデル化に関して, 本研究と関わりが深い 2 つのシステム (PTS[18],[19], ハイブリッドオートマトン [16]) について紹介を行い, HOTS との関連を述べる. さらに, 連続力学系から離散遷移機械への抽象化 [16] を紹介し, 本研究との関連を述べる.

5.2.1 フェーズ遷移機械 (PTS : Phase Transition System)

フェーズ遷移モデル (以下 PTS) は, ハイブリッドシステムの安全性, 活性について, 時相論理を用いた演繹的検証を行うための数学モデルであり, 安全性や活性の検証ルールが開発されている. PTS FTM は以下の 4 つの集合と, 1 つの条件によって与えられる. PTS は, 1 つ, あるいは複数のモジュールの合成によって構成される.

$$\boxed{FTM = \langle\ V,\ \Theta,\ \mathcal{T},\ \mathcal{A},\ \Pi\ \rangle}$$

V : **システム変数の有限集合**

任意型の離散変数の集合 $D = \{u_1, ..., u_n\}$ と実数型の連続変数の集合 $I = \{x_1, ..., x_n\}$ の和集合, $V = D \cup I$ で表される. さらにリセットされないマスタークロック $T \in I$ を導入する. 集合 V はモジュールのみが参照できる変数 V^P とモジュール外部からでも参照できる V^S の和集合 $V = V^P \cup V^S$ によって定義される, T は共有変数である.

Θ : **初期条件**

すべての初期条件を特徴付ける表明である. マスタークロック T に関しては,

$$\Theta \ \rightarrow\ T \ = \ 0$$

が要求される.

$\mathcal{T}$: **状態遷移の有限集合**

各状態遷移 $\tau \in \mathcal{T}$ は, 各状態 $s \in \Sigma$ に次状態 τ-successor $\tau(s) \subseteq \Sigma$ を写像するもので, 関数は表明 $\rho_\tau(V, V')$ で表現される.

$\mathcal{A}$: **アクティビティの有限集合**

アクティビティ $\alpha \in \mathcal{A}$ はアクティビティ関係,

$$p_\alpha \ \rightarrow \ I(t) = F^\alpha(V^0, t)$$

で表現される. ここで, p_α は α の活性条件とよばれる D 上の述語である. $I(t)$ は, 連続変数の集合を行列 $\{x_1, x_2, ...x_m = T\}$ とした際の略記である. $F^\alpha(V^0, t)$ は常微分方程式の解軌道を表現するものであるがここでは詳しい解説は省略する.

Π : **時間前進条件**

時間前進の宣言を表すものである. つまり変数の制約条件になる.

HOTS は PTS に対して以下の様に対応が取れる.

観測の集合 $\mathcal{O}$

HOTS における離散値を返す観測の集合 $\mathcal{D}$ は, PTS における変数の集合 D に対応する. 同様に HOTS の $\mathcal{P}$ は PTS の I に対応する. マスタクロックは, 観測 now に対応が取れる. PTS におけるローカル変数 V^P は, HOTS におけるアクティビティ内を動く連続変数 $value_i^j$ に対応し, V^S は HOTS におけるその他の観測に対応する.

初期状態の集合 $\mathcal{I}$

PTS の Θ であたえられるものと等しい. ただし PTS において, 初期状態は, その中に OTS における観測の初期値も含まれる.

条件付遷移規則の集合 $\mathcal{T} \cup \{tick_r | r \in R^+\}$

OTS では, 効力条件と事後状態で遷移関数 τ を表現していたが, PTS においては表明を用いて表現している. 例えば, "$c_\tau : o(\upsilon) = 1$ で事後状態が $o(\tau(\upsilon)) = 2$ のような遷移規則は, 観測 o を変数 x と置き, PTS の表明では,

$$\rho_\tau : x = 1 \ \wedge \ x = 2 \ \wedge T' = T$$

と表現される.

アクティビティ規則

HOTS におけるアクティビティ規則は, PTS の"$\mathcal{A}$: アクティビティの有限集合" に対応する. HOTS における. 時間前進条件は, HOTS の $tick$ の条件に対応する.

PTS は, 演繹的手法を用いてシステムの検証を行っているため, 証明の正しさに関しては, 帰納法を用いた本研究の検証よりも保証できる. しかし演繹的検証では, 状態爆発の問題などから, 大規模なハイブリッドシステムの検証はまだ難しい, また設計者の高度な数学的知識も要求されることから, 検証の容易さでは, 検証ルールが確立されており, 検証支援系を用いた解析を行える本研究に利点がある.

5.2.2 ハイブリッドオートマトン

ハイブリッドシステムの代表的な表現手法の一つであるハイブリッドオートマトンに関して解説する. ハイブリッドオートマトンの厳密な定義はここでは避けるが, HOTS との比較を行う上で必要なものを挙げる. ハイブリッドオートマトン HS は以下の 6 つ組で与えられる.

$$\boxed{HS \ = \ \langle \ Q, X, Init, Inv, t, f \ \rangle}$$

Q : **離散状態変数の集合**
Q は有限集合であり, 離散変数 q は Q に値を取る.

X : **連続状態変数の集合**
$x \in X$ は, 各 $q \in Q$ ごとに定められる.

$Init$: **初期状態の集合**
初期状態とその状態が取る値を持つ集合である.

Inv : **不変集合**
モデルの定義される範囲を指す.

t : **離散変数に対する条件付遷移規則の集合**

f : **連続変数に対する遷移規則の集合**

このシステムは, HOTS は以下の様に対応が取れる.

観測の集合 $\mathcal{O}$
HS における "Q : 離散状態変数の集合" は, HOTS では, アスペクトの集合に対応す

る．各 $q \in Q$ は, HS における 1 つのアスペクトである．p において, 現在どの領域で連続変数が動作しているかは, $active_i^j$ で知ることができる．

"X：連続状態変数の集合" は HOTS における $value_i^j$ に対応する．

HS における不変集合 Inv はアクティビティに関する感想 $boundary_i^j$ に対応する．

初期状態の集合 $\mathcal{I}$

HS における初期状態の集合 $Init$ がそのまま対応する．ただし, 初期状態における観測値の値も含まれる．

条件付遷移規則の集合 $\mathcal{T} \cup \{tick_r | r \in R^+\}$

HS における, "t： 離散変数に対する条件付遷移規則の集合" に対応． HOTS の $deactivate_i^j$, $deactivate_i^j$ が対応する, さらに, 連続変数を進化させる遷移規則, $tick$ が追加される．$tick$ による連続変数の変化は, HS の f に従う．

表 5.1 は, 例題：サーモスタットに対して, 両モデルの定義上の比較を行ったものである．ハイブリッドオートマトンでは, HOTS と比較して, 動作中の離散状態 $\mathcal{Q}$ を表現する観測 ($active_i^j$) に対応するものは定義されていない．さらに, 遷移規則に対する記述も $activate_i^j$, $deactivate_i^j$ のように詳細な記述は行わず, ある条件下におけるモードの変化のみを記述している．ハイブリッドオートマトンの f については, f は常微分方程式がそのまま記述されているのに対して, HOTS では, 常微分方程式の定量解を記述しなけらばならない．このことから, HOTS はハイブリッドオートマトンに対して, より制限がかけられた記述法であることが分かる．しかしながらサーモスタット例題に限れば, 解が定量的に求まるものであるので, ハイブリッドオートマトンで記述されたモデルを HOTS で記述することが可能である．このことから, 例題に限ればハイブリッドオートマトンで記述されたモデルを CafeOBJ で記述できることがいえる．

表 5.1: HS と HOTS 対応 (例題：サーモスタット)

HS	HOTS
Q	アスペクト $i = \{1, 2, ...\alpha\}$ の集合
$\mathcal{Q}$	全アスペクトが持つアクティビティの集合. 要素数は, n_i 個
X	$value_i^J$ の集合
$\mathcal{X}$	R
$Init$	初期状態の集合 $\mathcal{I}$
Inv	$boundary_i^j$ の集合
t	条件付遷移規則の集合 $\mathcal{T}$

例題：サーモスタットに対する適用

Q	$\{q_1\}$：離散変数
$\mathcal{Q}$	$\{on, off\}$：離散状態 (従って, $q_1 \in \{on, off\}$)
X	$\{\theta_1\}$：連続変数
$\mathcal{X}$	$\{\mathcal{R}\}$：連続状態
$Init$	$\{(off, \theta) : 70 < \theta < 80\}$
Inv	$\{(on, \theta) : \theta < 82\} \cup \{(off, \theta) : \theta > 68\}$
t	$\{(on, \theta, off) : \theta \geq 80\} \cup \{(off, \theta, on) : \theta \leq 70\}$
$f(on) = -x + 100,\ f(off) = -x$	

$$\Updownarrow$$

HS	HOTS
Q	$\{\ 1\ \}$ $(\alpha = 1)$
$\mathcal{Q}$	$\{1,\ 2\}$ $(n_1 = 2)$
X	$value_1^{1(on)}, value_1^{2(off)}$
$\mathcal{X}$	R
$Init$	初期状態の集合 $\mathcal{I}$
Inv	$boundary_1^{1(on)}, boundary_1^{2(off)}$
t	$activate_1^{1(on)}, activate_1^{2(off)},\ deactivate_1^{1(on)}, deactivate_1^{2(off)}$
f	$delta_1^{1(on)}, delta_1^{2(off)}$

5.2.3 連続力学系から離散遷移機械への抽象化

ある複雑なシステムに対して, 解析ツールを用いた検証を可能にするために, 元のシステムと等価な抽象化されたシステムを構築する手法を抽象化 (Abstraction) と呼ぶ. ハイブリッドシステムに関しても, 連続力学系から離散遷移機械への抽象化手法が提案されている [16]. 本節では, 抽象化手法に関して簡単にまとめ, 本研究との関連を述べる.

連続力学系から離散遷移機械への抽象化

まず, 抽象化で必要となる連続力学系と離散遷移機械系に関して解説を行う. これから行う抽象化は, 全て実数の一階述語論理上で行う.

連続力学系 (Countinuous Dynamical System : CS)
$CS = \langle\ X,\ Init,\ Inv,\ f\ \rangle$

連続力学系 CS の状態は, 変数の有限集合 X 上の全評価集合 $\mathbf{X} = \Re^X$ で表現される.

▷ X
X は, 実数 $\Re$ 上で対応付けられる, 変数の有限集合である.

▷ $Init$
初期状態の集合である.

▷ Inv
モデル定義範囲の集合である.

▷ f
f はベクトル場の変化を示す関数,

$$f:\ \mathbf{X} \to T\mathbf{X}$$

であり常微分方程式で与えられる. ここで, 解は必ず存在し, かつ一意に定まる必要がある.

CS の意味

CS の意味は, 下記の 3 つの規則を満たす写像,

$$\sigma:\ I \to \mathbf{X}$$

を満足する間隔 $I\ =\ [\tau_a,\ \tau_z]$ で表現される. σ についての規則は以下の通りである,

- 初期条件 (Initial condition) : $\sigma(\tau_\alpha) \in Init$
- 連続進化 (Coutinuous evolution) : for all $\tau \in (\tau_\alpha, \tau_z)$, $\dot{\sigma}(\tau) = f(\sigma(\tau))$

- 不変条件 (Invariant) : for all $\tau \in [\tau_a, \tau_z]$, $\sigma(\tau) \in Inv$

これは, HOTS を用いて考えると, 1 つのアクティビティが 1 つの σ に対応していることになる.

離散遷移機械 (Discrete Transition System : DS)

$$DS = \langle\ Q,\ Init, t\ \rangle$$

離散遷移機械 DS の状態は, 領域を示す変数 Q 上の全評価集合 $\mathbf{Q}$ によって表現される. $\mathbf{Q}$ は加算可能な集合である.

▷ Q
離散変数の有限集合であり, 加算可能な領域で存在する.
▷ $Init$
初期状態の集合である.
▷ t
遷移規則の集合であり,

$$t\ \subseteq\ Q\ \times\ Q$$

で表現される.

DS の意味

DS の意味は, 下記の 2 つの規則を満たす写像,

$$\theta :\ \mathcal{N}\ \rightarrow\ \mathbf{Q}$$

の集まりである.

- 初期条件 (Initial condition) : $\theta(0) \in Init$
- 離散進化 (Discrete evolution) : for all $i \in \mathcal{N}, (\theta(i), \theta(i+1)) \in t$

これらの条件は, OTS における開始性, 連続性とほぼ同じである.

抽象化

CS から DS への抽象化は以下の 2 ステップで行われる.

ステップ 1	DS の Q として利用可能な多項式 $p\ \subseteq \Re[X]$ を抽出する.
ステップ 2	抽象化された状態の初期状態 $Init\ Q$ と各遷移関係を規則的に作り出す

ステップ 1　CS で構築されたモデルから, 興味深い性質や, 間隔 I の外に出るような多項式を取り出す. そして, 取り出した多項式の導関数を追加する. 追加された導関数は, そのままでは除去できないが, 線形時間不変システム (Nilpotent Systems) [16] 等の特殊な条件を持つシステムでは, 除去することが可能である.

ステップ 2　ステップ 1 で新たに追加された多項式に対して, ドメイン { pos, neg, $zero$ } で評価を行う. 抽象化されたシステムの初期状態 $Init\ Q$ を抽出したのち, 遷移規則を作り出し, 解析ツール等による到達不可能状態の検出と, その状態に至る遷移規則の除去を行う.

以上の手続きを踏むことによって抽象化を行うことが可能である.

DS の連続変数 X 上から抽出された多項式の集合 P から生成された離散状態遷移システムは, 前説で解説を行ったハイブリッドシステムと一致することが知られている. 従って, ハイブリッドシステムで記述されたシステムは, ステップ 2 による抽象化を行うことが可能である.

抽象化されたシステムを用いた解析は, モデルチェッキングツール SAL [20][21] 等によって実際に行われている.

線形時間不変システムに関して

連続変数のベクトル列として X を定義すると以下の式になる.

$f = AX$ すなわち $\dot{X} = AX$ (A は定数行列) ..... [1]

$p = \sum_i a_i x_i = a^T X$ ..... [2]

$\dot{p} = a^T \dot{X} = a^T AX$ and $\ddot{p} = a^T A^2 X$ ..... [3]([1], [2] より)

従って, $A^n = 0$ となれば $(\dot{p}) = a^T A^n x = 0$ となる. 微分による時間変化が 0 だと示せれば, 非線型常微分方程式において単調増加と単調減少に式を分割できることが言える. 従って, HOTS において, 常微分方程式が非線型の場合でも, このような手法を用いて, 方程式の領域を分割することでき, CafeOBJ で記述が行えることが示唆できる.

本研究の帰納法による証明は, **invariant** に対する初期条件と, 境界に関する, 仕様との関係を記述者が推測している. 従ってこのような抽象化に関する決定手続きがあれば検証に対する記述者の手間を少なくすることが可能である.

5.3 今後の課題

本研究で対象としたシステムは, 常微分方程式が,

- 1 階線形微分方程式で記述されるもの

に限定されている. しかし参考文献 [16] と, アクティビティを CafeOBJ で記述できるのは, 単調増加であることを用いて, 常微分方程式が,

- n 階微分の結果が定数であるもの

も記述可能であると予測可能である. 今後の課題の 1 つは n 階微分方程式が定数であるような例題システムの記述検証を行うことである.

もう 1 つの課題は, 前節で述べた抽象化手法に類似した手法を考えだし HOTS に適用させることである. さらに参考文献 [16] で用いられている, 実数を用いた述語に関する決定手続きに対する提案を行い, 実装結果を CafeoBJ に反映させるような機構を考え出すことである.

謝辞

本研究の遂行と論文の執筆, そして研究生活全般に関して, 終始的確なご指導, ご鞭撻を賜りました 二木厚吉 教授にこころからから感謝いたします.

修士 1 年でのゼミ, 研究室でのシステム管理の仕事等で, まだ不馴れでご迷惑ばかりかけていた私を, 常に的確にサポートしてくださった 中村正樹 助手に感謝いたします.

本研究を始めるきっかけを与えてくださり, さらに研究で必要な知識を丁寧に教えてくださった, 客員研究員 緒方和博氏に感謝いたします.

本研究を遂行する際, 研究全般におきまして常に的確な助言を与えてくださり, また研究を円滑に進めるための CafeOBJ 前処理系を提供していただく等, 本研究における全ての面でご指導いただいた 研究員 清野貴博氏 に深く感謝いたします.

研究室配属後, 不安だらけであった私をやさしく励まして下さり, また, 研究生活, 研究方針等で公私にわたりまして, 助言をいただきました 博士後期課程 3 年 宗像一樹氏 に感謝いたします.

研究生活におきまして, 私の未熟な英会話を聞いてくださるなど, 常にやさしく接してくださった, 博士後期課程 2 年 向剣文氏 に感謝いたします.

研究, 研究環境の諸事で有益な知識を私に与えていただき, また質問に伺った際には, 研究が多忙にもかかわらず, 常に助言をくださる等, 公私に渡り大変お世話になりました 博士後期課程 1 年 中野昌弘氏 に深く感謝致します.

わずか半年の間でしたが, 英語によるコミュニケーション等さまざまな経験を私に与えてくださった, 博士後期課程 1 年 孔維強氏 に感謝致します. 孔維強氏 には, 本来ならばこちらが生活に関する助言等を行うはずが, 逆に研究に関して励まして下さる等, 大変優しく接してくださいました. ここにお詫びと感謝を重ねて申し上げます.

研究室配属から現在に至るまで, 公私に渡り様々な面でお世話になりました, 博士前期課程 2 年 及部佳代子さん, 加藤淳君, 河本孝久君, Jittisak SENACHAK 君に感謝します.

わずか 1 年足らずでしたが, 研究生活でお世話になりました 博士前期課程 1 年 原光太朗君, 渡辺真啓君 に感謝します.

本年度は残念ながらご指導いただけませんでしたが, 修士 1 年の際, 常に厳しくもやさしくご指導していただきました, 天野憲樹 助手 に感謝いたします.

修士 1 年の際, システム管理等, 研究環境に関する基礎知識等を丁寧に教えていただいた, 卒業生 浅羽義之氏 に感謝いたします.

本学入学後, 学生生活全般に対して, 共に励まし合い, または叱咤激励をくださった友人全てに感謝いたします. さらに, 本学入学に至るまで, 私をここまで導いてくださった, 全ての先生, 先輩, 後輩, 友人諸兄に深く感謝いたします.

末筆になりますが, 私が生まれてからこれまでの人生全てにおいて励ましていただき, 特にこの 2 年間については, 本学に入学することを黙って認めてくださったうえ, 本学における学生生活中において研究を円滑に遂行できるよう, わがままな私を陰ながらやさしくサポートしてくださいました, 父, 母, 祖父, 祖母, 妹, 親戚一同にこころから御礼と感謝を申し上げます.

参考文献

[1] Kokichi Futatsugi Razvan Diaconescu. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object - Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, ISBN981 - 02 - 3513 - 5, 1998.

[2] CafeOBJ official homepage
URL http://www.ldl.jaist.ac.jp/cafeobj/.

[3] 二木厚吉 緒方和博. 書き換えによるセキュリティプロトコルの帰納的検証. **コンピュータソフトウェア**, 20(3):54–72, 2003.

[4] K. Ogata. and K. Futatsugi. Specifying and verifying a railroad crossing with cafeobj. In *6th International Workshop on Formal Methods for Parallel Programming: Theory and Applications*. IEEE CS Press, 2001.

[5] K. Ogata and K. Futatsugi. Modeling and verification of distributed real-time systems based on cafeobj. In *16th IEEE International Conference on Automated Software Engineering*, pages 185–192. IEEE CS Press, 2001.

[6] 潮俊光. ハイブリッドシステムへの期待. **システム \ 制御 \ 情報**, 46(3):105–109, 2002.

[7] 平田 藤田. スイッチング制御. **計測と制御**, 38(3):176–181, 1999.

[8] A.Bemporad. and E.Mosca. Fulfilling hard constraints in uncertain linear systems by reference managing. *Automatica*, 34(4):451–461, 1998.

[9] 潮 小林, 柴田. 非ホロノミックチェインドシステムのハイブリッド時間軸状態形による最適制御. **システム制御情報学会論文誌**, 15(5), 2002.

[10] A. Bemporad. and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[11] A. Bemporad., F.D.Torrisi., and M.Morari. Discretetime hybrid modeling and verification of the batch evaporator process benchmark. *European Journal of Control*, 7(4):382–399, 2001.

[12] The Hybrid System Project - Research
URL http://control.ee.ethz.ch/ hybrid/hysdel/.

[13] B.Brandin. and W.M.Wonham. Supervisory control of timed discrete-event systems. *IEEE Trans. Automatic Control*, 39(2):329–342, 1994.

[14] F.Vaandrager. N.Lynch., R.Segala. and H.B.Weinberg. Hybrid i/o automata. *Hybrid System III, LNCS 1066, Springer-Verlag*, pages 496–510, 1996.

[15] 潮俊光. ハイブリッドペトリネット. **計測と制御**, 38(3):169–175, 1999.

[16] A Tiwari and G Khanna. Series of abstractions for hybrid automata. In *Proc of 5th International Workshop on Hybrid Systems: Computation and Control*, volume 2289, pages 465–478, 2002.

[17] 高井重昌 潮俊光. 強制事象のあるハイブリッドシステムにおける状態フィードバック制御. **システム制御情報学会論文誌**, 15(3):143–149, 2002.

[18] Yonit Kesten, Zohar Manna, and Amir Pnueli. Verification of clocked and hybrid systems. *Acta Informatica*, 36(11):837–912, 2000.

[19] 山根智. ハイブリッドシステムのモジュールの仕様記述と検証の手法. **情報処理学会論文誌**, 44(3):897–914, 2003.

[20] Saddek Bensalem, Vijay Ganesh, Yassine Lakhnech, César Mu noz, Sam Owre, Harald Rueß, John Rushby, Vlad Rusu, Hassen Saïdi, N. Shankar, Eli Singerman, and Ashish Tiwari. An overview of SAL. In C. Michael Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, Hampton, VA, jun 2000. NASA Langley Research Center.

[21] SAL homepage
URL http://sal.csl.sri.com/.