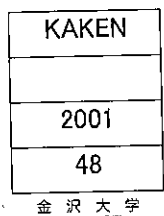


# 時相論理と並行計算、オートマトンの統合化による自律性のある分散システムの設計支援

メタデータ	言語: Japanese 出版者: 公開日: 2019-05-13 キーワード: 作成者: メールアドレス: 所属:
URL	<a href="https://doi.org/10.24517/00053966">https://doi.org/10.24517/00053966</a>

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 International License.





研究題名

時相論理と並行計算、オートマトンの統合化による自律性の  
ある分散システムの設計支援

課題番号：11680360

平成11年度～平成13年度科学研究費補助金（基盤研究（C）（2））研究成果報告書

平成14年3月

研究代表者 山根 智  
(金沢大学工学部)

金沢大学附属図書館



8011-05290-X

学

KAKEN  
2001  
48

# は し が き

## 研 究 組 織

研究代表者：山根 智（金沢大学工学部）

## 交付決定額（配布額）

（金額単位：千円）

	直接経費	間接経費	合 計
平成11年度	500	0	500
平成12年度	500	0	500
平成13年度	700	0	700
総 計	1700	0	1700

## 研 究 発 表

### 1. 学会誌等

- (1)山根 智：“統一セマンティックモデルに基づくリアクティブシステムの形式的開発方法論”，情報処理学会論文誌，Vol. 41, No. 3, pp. 767-782, 2000
- (2)山根 智：“Assume-Guarantee 検証による実時間システムの階層的設計支援手法”，情報処理学会論文誌，Vol. 41, No. 12, pp. 3352-3362, 2000
- (3)山根 智：“時間ステートチャートによる仕様記述と演繹的検証”，電子情報通信学学

著 者 寄 贈

- 会研究報告, SS99-56, pp. 9-16, 2000
- (4) 山根 智, 山ノ口崇” 実時間システムの演繹的検証と自動検証の実験的研究”, 情報処理学会研究報告, MPS2000, pp. 11-18, 2000
- (5) 山ノ口崇, 山根 智” 実時間システムの演繹的検証”, 電子情報通信学会研究報告, SS2000-6, pp. 1-8, 2000
- (6) 菅藤 真, 山根 智:” OS スケジューラプログラムの自動検証の実験的研究”, 電子情報通信学会研究報告 SS2000-7, pp. 9-16, 2000
- (7) 山根 智:” リアルタイムソフトウェアの形式化と検証”, 電子情報通信学会研究報告, SS2000-5, pp. 33-40, 2000
- (8) 山ノ口崇, 山根 智:” AND 構造と OR 構造の分解による実時間ソフトウェアの安全性の演繹的検証”, Foundation Of Software Engineering' 2000, 日本ソフトウェア科学会, pp. 237-244, 近代科学社, 2000
- (9) 山根 智, 山ノ口崇” Assume-Guarantee 形式による実時間ソフトウェアの演繹的詳細化検証手法”, 電子情報通信学会研究報告, SS2000-31, pp. 1-8, 2001
- (10) 佐野範佳, 山根 智, 直井 徹:” 制御ソフトウェアの開発方法論”, 電子情報通信学会研究報告 SS2000-42, pp. 1-7, 2001
- (11) 梅田謹公, 山根 智:” E-commerce ソフトウェアの形式的開発方法論”, 電子情報通信学会研究報告 SS2000-44, pp. 17-24, 2001
- (12) 梅田謹公, 山根 智:” 演繹的モデル検査による自動詳細化検証手法”, 電子情報通信学会研究報告 SS2001-2, pp. 9-16, 2001
- (13) 菅藤 真, 瀬口俊和, 山根 智:” プログラムスライジングによるプログラムのモデル検査手法”, 電子情報通信学会研究報告 SS2001-3, pp. 17-24, 2001
- (14) 山根 智:” ハードリアルタイムシステムの形式的仕様記述と形式的検証”, 情報処理学会論文誌, Vol. 42, No. 6, pp. 1623-1635, 2001
- (15) 山根 智:” ハイブリッドシステムのモジュール演繹的検証手法”, 電子情報通信学会研究報告, CST2001-4, pp. 21-28, 2001
- (16) 山ノ口崇, 山根 智 (T. Yamanokuchi, S. Yamane):” Deductive Refinement Verification Methods based on Assume-Guarantee Style and their Experimental Evaluations of Real-Time Software”, 2nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 254-261, IACIS, 2001
- (17) 館 宣伸, 山根 智:” ハイブリッドモジュールの詳細化自動検証手法”, 情報処理学会プログラミング言語研究会, 2001 (印刷中)
- (18) 館 宣伸, 山根 智:” 分散システムのハイブリッドモジュールによる形式化と詳細化自動検証”, 情報処理学会研究報告 2001-DPS-105, Vol. 2001, No. 107, pp. 1-6, 2001
- (19) 山根 智:” ハイブリッドシステムの高信頼性設計方法論”, 電子情報通信学会研究報告

告, FTS2001-71, pp.17-24, 2001

(20)山根 智:”ハイブリッドモデルの詳細化理論による組込み型システムの開発方法論”,  
電子情報通信学会研究報告, SS2001-40, pp.7-14, 2002

(21)山根 智 (S. Yamane) :”Modular Specification and Verification Method for Hybrid  
Real-time Systems”,Proc. 8<sup>th</sup> International Conference on Real-Time Computing  
Systems and Applications, pp.201-208, IEEE Computer Society, 2002

(22)山根 智:”ハイブリッドシステムの構成的証明とその計算機実験”,電子情報通信学  
会研究報告, SS2001-49, pp.25-32, 2002

## 2. 口頭発表

なし

## 3. 出版物

なし

## 目 次

1. はじめに.....	1
2. 実時間型開放分散システムの Assume-guarantee 方式による自動的設計支援.....	4
3. 実時間型開放分散システムの演繹的設計支援.....	23
4. 実時間型開放分散システムの Assume-guarantee 方式による演繹的設計支援.....	43
5. ハイブリッドモデルに基づく開放分散システム的设计支援.....	71
6. まとめ.....	99
参考文献.....	100

# 1 はじめに

近年、コンピュータとインターネットの融合により、分散システムは一般化しており、その設計方法論の構築は重要な研究テーマである。従来より、分散システムの仕様記述やプログラミング言語などの設計に関する研究は多くなされている [1]。しかし、従来の設計支援の対象は仕様記述やプログラミングなどの各開発工程に閉じていたり、大規模化及び多様化した分散システムを対象としていない。本研究では、新たな視点から、分散システムの特徴を把握して、大規模化及び多様化した分散システムの設計支援手法を実現する。

まず、大規模化及び多様化した分散システムの設計には、以下のような特徴があると考えられる：

1. 分散システムでは、プロセスが増減したり、プロセスの機能の変化が生じて、各プロセスにとっては、外部の環境の変化が激しい。
2. 分散システムは、多くのプロセスが並行動作しており、システム全体を把握することが困難であるから、漸増的かつ階層的に、抽象度の高い仕様から低い仕様を作成する必要がある。
3. 設計者に負担をかけない自動化手法を基礎として、状態数組合せ爆発を抑制して大規模システムを支援する必要がある。

従来、この分散システムの設計問題は、プロセス代数や $\omega$ -オートマトン、時相論理、時間オートマトンなどの形式的検証手法を基礎として研究されていた [2]。しかし、従来の研究は、個別に研究されており、しかも設計方法論があまりないために、局所的な実用化しか図られていない。本研究では、以下のように、従来個別に研究されていた手法を設計方法論として統合化して、実用的な設計支援システムを開発する。

1. まず、発展性や自律性のある分散プロセスが表現できる仕様記述言語、すなわち、開放型のオートマトンベースの言語を形式的に定義する。この開放型オートマトンは、従来のオートマトンと異なり、すべての外部環境に対する自律的な動作が仕様記述できる。
2. 次に、分散システムを階層的に設計して、抽象度の高い仕様と低い仕様との間の整合性を保証するために、時相論理とプロセス代数を統合化した時間模倣関係を開発する。
3. 最後に、上記方法論を不動点計算で形式化してアルゴリズム化したり、演繹的な検証を実現した支援システムを開発する。また、構成的に検証できる Assume-Guarantee style の検証手法を開発する。

さらに、具体的には、以下のとおりである。分散システムのモデル化のポイントは、開放型システムとして分散システムをとらえて、実時間モデル及びハイブリッドモデルの観点からモデル化して、自動検証及び演繹的検証により設計の正当性を効率的に保証するものである。以上の観点より、我々は、以下の4つのモデルから、設計支援を実現した。

1. 実時間型開放分散システムの Assume-guarantee 方式による自動的設計支援：

開放型分散システムの仕様記述言語として開放型時間オートマトンを形式化して, Assume-guarantee 方式による時間模倣関係の自動検証及び receptiveness の自動検証を実現した. これにより, 実用レベルの分散システムの段階的詳細化設計が可能となった.

2. 実時間型開放分散システムの演繹的設計支援:

開放型分散システムの仕様記述言語として時間状態チャートを形式化して, 演繹的な詳細化検証の公理系を開発した. これにより, 無限状態を有する分散ソフトウェアの段階的詳細化設計が可能となった.

3. 実時間型開放分散システムの Assume-guarantee 方式による演繹的設計支援:

開放型分散システムの仕様記述言語としてクロック遷移モジュールを形式化して, Assume-guarantee 方式による演繹的な詳細化検証の公理系及び receptiveness の演繹的検証を開発した. これにより, 実用レベルの分散ソフトウェアの段階的詳細化設計が可能となった.

4. ハイブリッドモデルに基づく開放分散システムの設計支援:

開放型分散システムの仕様記述言語としてフェーズ遷移モジュールを形式化して, 演繹的な詳細化検証の公理系を開発した. これにより, 制御法則を含む分散ソフトウェアの段階的詳細化設計が可能となった.

また, 本研究の特色と意義は以下のとおりである.

1. 開放型分散システムの仕様記述言語を開発することにより, 従来扱えなかった分散システムの自律性や発展性, 通信遅延が形式的な数学的モデルとして表現できる.
2. 実時間モデル及びハイブリッドモデルの模倣関係の不動点計算及び演繹的検証を提案することにより, 時相論理とプロセス代数が統合化できて, 上位レベルの仕様の性質が完全に下位レベルの仕様に継承できる.
3. Assume-Guarantee style の検証により, 大規模な実問題が設計できるようになる.

また, 分散システムの設計に関する理論は多数研究されており, 以下のとおりである. 並行計算理論や時相論理などによる形式的手法は, 1991年に R. Milner が並行計算理論 [3] で, 1996年に A. Pnueli が時相論理 [4] で, それぞれチューリング賞を受賞して, 多くの研究者によって研究されている. 例えば, 米国のカーネギーメロン大学の記号モデル検査 [5] や北カロリナ大学の並行計算理論を基礎とする自動検証 [24], スウェーデンのアップサラ大学の実時間モデル検査 [28] などである. しかし, どんな環境においても動作するようなモデル化, すなわち, 開放型システムを対象とした自動検証はほとんど存在しない. 本研究は, 不動点計算による形式化を基礎として, 自律性や発展性を狙いとした形式的手法による設計方法論の確立を目指している. 本研究は, 理論的には分散システムの形式化, 工学的には設計方法論として, 大いに威力を発揮するものと考えられる.

本稿の構成は以下のとおりである. まず, 2節では, 実時間型開放分散システムの Assume-guarantee 方式による自動的設計支援を述べる. 次に, 3節では, 実時間型開放分散システムの演繹的設計支援を述べる. 次に, 4節では, 実時間型開放分散システムの Assume-guarantee 方式による演繹的設計支

援を述べる。次に、5節では、ハイブリッドモデルに基づく開放分散システムの設計支援を述べる。最後に、6節では、まとめと今後の展望を述べる。

## 2 実時間型開放分散システムの Assume-guarantee 方式による自動的設計支援

### 2.1 導入

通信システムや制御システムの多くは実時間システムである。さらに、これらは他のシステムと協調して動作するリアクティブシステムであることもしばしばである。したがって、これらのシステムの設計においては、各システムにおける処理時間が他のシステムに及ぼす影響や、システム間の通信遅延といった実時間性をも考慮する必要がある。このことにより、(リアクティブ) 実時間システムの設計作業は複雑になりがちであり、また、システムの信頼性保証も一般的には困難である [8, 21]。したがって、実時間システムを階層的に詳細化して設計できること、また、階層間の整合性の保証によって、最終的に得られるシステムの信頼性保証ができることが重要である [10, 11]。このことをふまえ、本論文では、要求仕様から設計仕様を得るまでの各階層の仕様を同一の形式的仕様記述言語で記述し、階層間の整合性を構成的に自動検証するための枠組みを提案する。具体的にいうと、本論文で提案するのは、仕様記述のためのリアクティブな時間オートマトンの一種、および、この時間オートマトンにおける時間模倣関係の Assume-Guarantee 検証である。Assume-Guarantee 検証形式 [12] は、並列合成プロセス間の整合性検証を効率的に行うための既知の形式 [13] であり、合成プロセスにおける状態の組み合わせ爆発が抑制できるものである。

従来の実時間システムの形式的検証の研究としては、時間オートマトンの言語包含検証 [8] やモデルチェッキング検証 [21]、時間模倣関係の証明系 [10]、時間模倣関係の自動検証 [11] などがある。階層的設計においては時間模倣関係の自動検証が適当であり、本論文では、時間模倣関係の自動検証手法 [11] を基礎とする。

また、検証を効率化するために、Assume-Guarantee 検証 [12] が注目されている。Assume-Guarantee 検証では、環境の動作を仮定して構成的に検証するものであり、状態の組み合わせ爆発が抑制できる。しかし、従来 of Assume-Guarantee 検証は安全性や活性を対象とするものがほとんどであり、段階的詳細化における階層間の整合性検証に関するものはない。唯一の例外として、時間モジュールの Assume-Guarantee 形式による階層的設計支援がある [13]。この研究では、Assume-Guarantee 検証の健全性を保証するために、receptiveness [14] の自動検証を実現している。しかし、この研究では、本研究と異なり、階層間の整合性関係として言語包含関係を使用しており、さらに Assume-Guarantee 検証手法の部分手続きとなる、階層間の整合性の具体的検証手続きを述べていない。

本章では、環境との相互作用が表現できる時間 I/O オートマトンを定義して、要求仕様から設計仕様までを、時間 I/O オートマトンで統一的に仕様記述する。そして、時間模倣関係に基づく Assume-Guarantee 検証手法により、仕様間の整合性を自動検証できる枠組みを定義して階層的な設計手法を提案する。さらに、その手法を実装して、その有効性を計算機実験により示す。

以下の本章の構成は以下のとおりである。2. 2章では実時間システムの仕様記述手法を導入して、

2. 3章では時間模倣関係を基礎とした階層的な設計手法を提案する。2. 4章では設計支援システムと設計事例を示して、2. 5章ではまとめを述べる。

## 2.2 実時間システムの仕様記述手法と時間模倣関係

### 2.2.1 実時間システムの仕様記述手法

実時間システムでは多くのプロセスが並列動作している。実時間システムの要求仕様や設計仕様のプロセスは時間I/Oオートマトンで記述する。時間I/Oオートマトンは環境からの入力イベントに反応したり、環境に出力イベントを発生させながら状態遷移して動作する。また、状態において、時間経過して動作する。ここで、環境とは対象としているプロセス以外のプロセスである。まず、文献 [8] を基礎として、時間I/Oオートマトンを形式的に定義する。

#### Definition 1 (時間I/Oオートマトンの定義)

時間I/Oオートマトンは  $P = (S, S_0, EVENT, C, E)$  の5つ組で定義される。ここで、

$S$  は有限状態集合

$S_0 \subseteq S$  は初期状態集合

$EVENT$  はイベントの有限集合

$C$  はクロックの有限集合

$E \subseteq S \times S \times EVENT \times 2^C \times \Phi(C)$  は遷移関係

ただし、イベントの有限集合は  $EVENT = IN \cup OUT$  である。ここで、 $IN$  は入力イベントの有限集合、 $OUT$  は出力イベントの有限集合である。また、 $\Phi(C)$  はクロック集合  $C$  のタイミング制約式  $\delta$  であり、クロック集合  $C(x \in C)$  と非負整数の時刻定数  $d$  により以下のように帰納的に定義される：

1.  $x \leq d$  や  $d \leq x$  は  $\delta$  である。
2.  $\delta_1$  と  $\delta_2$  が  $\delta$  ならば、 $-\delta_1$  や  $\delta_1 \wedge \delta_2$  は  $\delta$  である。

状態遷移  $(s, s', a, \lambda, \delta)$  は入力イベント  $a$  または出力イベント  $a$  による状態  $s$  から状態  $s'$  までの遷移を表す。集合  $\lambda \subseteq C$  は、この状態遷移でリセットされるクロック集合を表す。本論文の以下では、

$$s \xrightarrow{a, \lambda, \delta} s'$$

は、状態遷移  $(s, s', a, \lambda, \delta) \in E$  を意味する。

時間I/Oオートマトンは時間付き言語  $(event, \tau)$  を受理する。ここで、 $event = event_1, event_2, \dots$ ,  $\tau = \tau_1, \tau_2, \dots$  である。ただし、 $event_i \in EVENT$ ,  $\tau_i \in \mathbf{R}$  ( $\mathbf{R}$  : 非負の実数) とする。時間I/Oオートマトンの走査列  $r$  は、以下のような無限列である：

$$\langle s_0, \nu_0 \rangle \xrightarrow{event_1, \tau_1} \langle s_1, \nu_1 \rangle \xrightarrow{event_2, \tau_2} \langle s_2, \nu_2 \rangle \xrightarrow{event_3, \tau_3} \dots$$

ここで、 $s_i \in S$ ,  $\nu_i \in [C \rightarrow \mathbf{R}]$  である。 ■

また、時間 I/O オートマトンの意味は、時間付き言語で定義できる。その詳細は文献 [8] と同様であり、本論文では紙面の都合上から省略する。

実時間システムは多くの時間 I/O オートマトンから構成されていると考えて、実時間システムの仕様は時間 I/O オートマトンのカルテジアン積（並列合成）で表現する。我々は、プロセス間の相互作用や環境との相互作用を単純にモデル化するために、文献 7) に従って、以下の考え方により、実時間システムの構成を定義する。

1. あるプロセスの入カイベントと他のプロセスの出カイベントが同じならば、その出カイベントによって入カイベントが生成されて、両方のプロセスは同期する。最終的には、このイベントは出カイベントとして認識する。
2. プロセスの出カイベント同士は同期しないと考える。すなわち、構成するプロセスの出カイベント間には、交わりがない。

以下では、時間 I/O オートマトンのカルテジアン積（並列合成）を定義する。

**Definition 2 (時間 I/O オートマトンのカルテジアン積)**

プロセス  $P_i = (S_i, S_{0i}, EVENT_i, C_i, E_i)$  ( $i = 1, \dots, n$ ) に対して、以下のように実時間システム  $P = (S, S_0, EVENT, C, E) = P_1 \parallel P_2 \parallel \dots \parallel P_n$  を構成する。なお、 $\parallel$  は並列合成を意味する。ただし、 $EVENT_i = IN_i \cup OUT_i$  とする。並列合成できるための条件として、以下がある：すべての  $i, j \in \{1, \dots, n\}, i \neq j$  に対して、 $OUT_i \cap OUT_j = \emptyset$  である。

1.  $S = S_1 \times \dots \times S_n$ . ただし、 $\times$  はカルテジアン積である。
2.  $S_0 = S_{01} \times \dots \times S_{0n}$ .
3.  $IN = \bigcup_{i=1, \dots, n} IN_i - OUT_{same}$ , ただし、 $OUT_{same}$  は入カイベントと等しい出カイベントの集合である。 $OUT_{same} = \emptyset$  の場合は、入カイベントが出カイベントとすべて異なるときである。一方、 $\bigcup_{i=1, \dots, n} IN_i = OUT_{same}$  の場合は、入カイベントが出カイベントとすべて等しいときである。
4.  $OUT = \bigcup_{i=1, \dots, n} OUT_i$ .
5.  $C = \bigcup_{i=1, \dots, n} C_i$ .
6.  $E$  は以下の規則で生成する：

以下では、

$$s_i \xrightarrow{a, \lambda_i, \delta_i} s_i'$$

かつ

$$s_j \xrightarrow{b, \lambda_j, \delta_j} s_j'$$

の場合に、生成規則を定義する。

(a) 入カイベントと出カイベントが同期するとき

i.  $a \in IN_i$  かつ  $b \in OUT_j$  かつ  $a = b$  のとき

$$s_i \times s_j \xrightarrow{b, \lambda_i \cup \lambda_j, \delta_i \wedge \delta_j} s_i' \times s_j'$$

ii.  $a \in OUT_i$  かつ  $b \in IN_j$  かつ  $a = b$  のとき

$$s_i \times s_j \xrightarrow{a, \lambda_i \cup \lambda_j, \delta_i \wedge \delta_j} s_i' \times s_j'$$

(b) その他のとき

$$s_i \times s_j \xrightarrow{a, \lambda_i, \delta_i} s_i' \times s_j$$

または

$$s_i \times s_j \xrightarrow{b, \lambda_j, \delta_j} s_i \times s_j'$$

■

### Example 1 (実時間システムの構成例)

実時間システムの構成例を図1に示す。まず、2つのプロセスが与えられたとして、並列合成して、実時間システムを構成する。図1 (1), (2)は2つのプロセスを示す。並列合成の場合、(1)の入力イベント  $a$  は (2) の出力イベント  $a$  であるので、並列合成 (3) では  $a$  は入力イベントではなく、出力イベントである。図1 (3) は、2つのプロセスから並列合成された実時間システムを示す。 ■

### 2.2.2 時間模倣関係の定義

次に、時間I/Oオートマトンの上で、時間模倣関係 [11] を定義する。なお、本論文の時間模倣関係は、引用文献中 [11] では安全性時間模倣関係である。以下に、時間模倣関係を形式的に定義する。

#### Definition 3 (時間模倣関係の定義)

上位レベルの仕様  $P^A = (S^A, S_0^A, EVENT^A, C^A, E^A)$  と下位レベルの仕様  $P^C = (S^C, S_0^C, EVENT^C, C^C, E^C)$  を任意の時間I/Oオートマトンとする。以下の条件を満たす  $R \subseteq S^A \times S^C$  を  $P^A$  から  $P^C$  への時間模倣関係と呼び、そのような関係  $R$  が存在するとき、 $P^A \preceq P^C$  と記述する。

#### 1. (時間模倣関係)

$(s_i^A, s_i^C) \in R$  ならば  $\forall e^A \in EVENT^A, \lambda^A, \delta^A$  について、

$$s_i^A \xrightarrow{e^A, \lambda^A, \delta^A} s_{i+1}^A$$

である  $s_{i+1}^A$  が存在するならば、

$$s_i^C \xrightarrow{e^A, \lambda^A, \delta^A} s_{i+1}^C$$

である  $s_{i+1}^C$  が存在して、 $(s_{i+1}^A, s_{i+1}^C) \in R$  である。なお、 $s_i^A, s_{i+1}^A \in S^A$  かつ  $s_i^C, s_{i+1}^C \in S^C$  である。

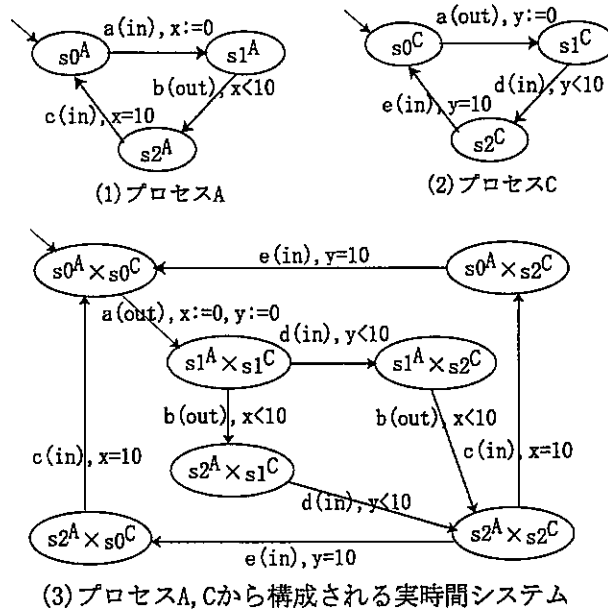


図 1: 実時間システムの構成例

## 2. (初期条件)

$\forall s_0^A \in S_0^A$  に対して,  $(s_0^A, s_0^C) \in R$  を満たす  $s_0^C \in S_0^C$  が存在する. ここで,  $(s_0^A, s_0^C) \in R$  とは, 上位レベルの初期状態から下位レベルの初期状態への時間模倣関係が存在することを意味する. すなわち, 時間模倣関係の中に, 初期状態が含まれていることを意味する.

■

一方, 時間モジュールの検証の論文 [13] では, 言語包含関係が使用されている. 言語包含関係は時間モジュール間のトレースの包含関係であり, 本論文では時間オートマトン間のイベント列の包含関係に対応する. なお, 時間モジュールの論文 [13] では, トレースは外部から観測可能な変数値の移り変わりである. 本論文のイベントは, 時間モジュールではイベントに対応する変数値への代入で表現できる. また, 時間模倣関係は言語包含関係よりも強い関係であることが知られている [11]. 言語包含検証は正則性や公平性を含む安全性や活性などが検証できる強力かつシンプルな検証手法であるが, 仕様を非決定性時間オートマトンで記述すれば, 言語包含検証は不可能である [8]. 以上より, 本論文では時間模倣関係を使用する. また, 言語包含関係は存在するが時間模倣関係は存在しない場合が存在する. これは, 言語包含関係が線形時間構造上の関係であり, 時間模倣関係が分岐時間構造上の関係であることに起因する. 分岐時間構造では, 時間の構造は各時点がその直後の時点を複数個持っており, 時間の構造は分岐した樹状の構造を持つ. すなわち, 分岐時間構造は非決定性の動作をうまく表現できる. ゆえに, 非決定性が存在する場合は多い実時間システムには時間模倣関係が適している

と考えられる。

## 2.3 時間模倣関係による階層的な設計手法

本論文では、言語包含関係 [13] よりも強い関係である時間模倣関係 [11] の Assume-Guarantee 検証を基礎として、時間 I/O オートマトンによる統一的な仕様記述による実時間システムの階層的な設計手法を提案する。最初に階層的な設計手法の概要を定義して、次に検証時に必要となるリージョングラフを定義して、最後に receptiveness の検証と Assume-Guarantee 形式の検証を定義する。なお、receptiveness はプロセスが物理的に実装可能な条件であり、プロセスが物理的に実装可能な条件とはプロセスが無限に動作する場合は経過時間が発散することを意味する。Assume-Guarantee 形式検証は対象プロセスと環境との並列合成により、システム全体を検証する手法である。また、Assume-Guarantee 形式検証では、対象プロセスと環境との並列合成で構成されるシステムが物理的に実装可能であることを保証する必要がある、このためにすべてのプロセスが receptive であることを保証する必要がある。

### 2.3.1 階層的な設計手法の概要

我々は、実時間システムの抽象度の高い上位レベルの仕様と抽象度の低い下位レベルの仕様を同一の時間 I/O オートマトンで仕様記述して、時間模倣関係の Assume-Guarantee 検証によりそれらの間の整合性を自動検証できる階層的な設計手法を提案する。

#### Definition 4 (階層的設計手法)

実時間システムの階層的設計手法は以下の手順から構成される。

1. まず、実時間システムの上位レベルのすべてのプロセスの仕様を時間 I/O オートマトンで記述する。
2. 次に、上位レベルの仕様を基礎として、それを実現する下位レベルのすべてのプロセスの仕様を時間 I/O オートマトンで設計する。
3. 次に、すべてのプロセスが *receptiveness* を満たすことを検証する。プロセスが *receptiveness* を満たすまで修正する。
4. 次に、上位レベルの仕様から下位レベルの仕様への時間模倣関係の Assume-Guarantee 検証により、時間模倣関係の存在性を確認する。時間模倣関係が存在するまで、下位レベルの仕様を修正する。
5. 以上のステップを繰り返して、最終的な実現仕様を設計する。

■

なお, receptiveness を満たさないプロセスは物理的に実装できないプロセスである. ゆえに, 計算機に実装可能な正しいプロセスはすべて receptiveness を満たす.

### 2.3.2 リージョングラフ

時間 I/O オートマトンのタイミング制約記述の時間領域は稠密なので, クロック変数にそのまま時間を割り当てると, 無限個の状態とクロック値のペアができてしまう. しかし, タイミング制約式の中の定数部分 (例えば  $x < d$  の  $d$  の部分) が非負整数なので, クロック値の整数部分と小数部分の順序関係が同じならば, 時間 I/O オートマトンの動作は区別されない. この考えから, 我々は, 動作が区別されないクロック割り当ての同値関係により, クロックリージョンと呼ばれる同値類を構成する [8]. そして, クロックリージョンにより, 時間 I/O オートマトンから, 有限な商構造であるリージョングラフを構成することができる [8].

まず, このクロック割り当ての同値関係 [8] を定義する.

#### Definition 5 (クロック割り当ての同値関係)

時間 I/O オートマトン  $P$  が与えられたとする. 任意のクロック変数  $x \in C$  に対して,  $P$  に現れる最大のクロック定数を  $c_x$  とする. また, 時間  $t \in \mathbf{R}$  に対して,

$\text{fract}(t)$  :  $t$  の小数部分

$\lfloor t \rfloor$  :  $t$  の整数部分

とする.  $P$  のクロック割り当ての集合を  $\Gamma(P)$  とし, その要素を  $\nu, \nu' \in [C \rightarrow \mathbf{R}]$  とする. 以下の条件を満たすときに限り,  $\nu \equiv \nu'$  と表わす.

1. 任意のクロック変数  $x \in C$  に対して,  $\lfloor \nu(x) \rfloor$  と  $\lfloor \nu'(x) \rfloor$  が同じ, または,  $\nu(x)$  と  $\nu'(x)$  の両方が  $c_x$  よりも大きい.
2.  $\nu(x) \leq c_x$  と  $\nu(y) \leq c_y$  であるすべての  $x, y \in C$  に対して,  $\text{fract}(\nu'(x)) \leq \text{fract}(\nu'(y))$  のときに限り,  $\text{fract}(\nu(x)) \leq \text{fract}(\nu(y))$  である.
3.  $\nu(x) \leq c_x$  であるすべての  $x \in C$  に対して,  $\text{fract}(\nu'(x)) = 0$  のときに限り,  $\text{fract}(\nu(x)) = 0$  である.

■

時間 I/O オートマトン  $P$  のクロックリージョンは同値関係  $\equiv$  によって導かれるクロック割り当ての同値類であり,  $[\nu]$  と記述する.

次に, クロックリージョンを基礎として, 時間 I/O オートマトンのリージョングラフ [8] を定義する.

#### Definition 6 (リージョングラフ)

時間 I/O オートマトン  $P = (S, S_0, \text{EVENT}, C, E)$  のリージョングラフは  $G = (Q, Q_0, \text{EVENT}, E')$

で定義される.

ここで,

1.  $Q : \langle s, [\nu] \rangle$  の有限集合
  - (a)  $s \in S$
  - (b)  $[\nu] = \{\nu' \mid \nu \equiv \nu'\}$ ,  $\nu \in \Gamma(P)$
2.  $Q_0 : \langle s_0, [\nu_0] \rangle$  の有限集合
  - (a)  $s_0 \in S_0$
  - (b)  $[\nu_0]$  : すべてのクロック変数が 0 の順序対の集合
3.  $EVENT$  : 有限イベント集合
4.  $E' \subseteq Q \times EVENT \times Q$  : 状態遷移関係

■

### 2.3.3 Receptiveness の自動検証

実時間システムのプロセスの動作の正当性を保証する概念として、並列演算閉包性が存在し、Nonzeno より弱い概念である *receptiveness* [14] を使用する [13]. プロセスは、以下のいずれかの場合に *receptive* であると言う.

1. プロセスの経過時間が発散する. つまり、プロセスの状態遷移が無限回起きると、プロセスの経過時間も無限に大きくなる.
2. プロセスの状態遷移が有限回起きる.

#### Definition 7 (Receptiveness の定義)

*receptiveness* はプロセスと環境 (対象としているプロセス以外のプロセス) との無限ゲームの下で、プロセスに対して定義される. その様子を図 2 に示す. ここで、プロセス及び環境は以下のように動作する: プロセスは出力イベントにより状態遷移するか、時間経過する. 環境は入力イベントを発生させるか、時間経過する. 無限ゲームの戦略により、以下のように、次状態の動作が決定される:

1. もし環境が入力イベントを提案するならば、環境の入力イベントが選択される.
2. もし環境が時間経過を提案してプロセスが出力イベントを提案するならば、プロセスの出力イベントが選択される.
3. もし環境とプロセスがともに時間経過を提案するならば、小さい時間経過が選択される.

上記戦略に従って動作するプロセスの経過時間が発散するか、またはプロセスの状態遷移が有限回ならば、プロセスは *receptive* である. ■

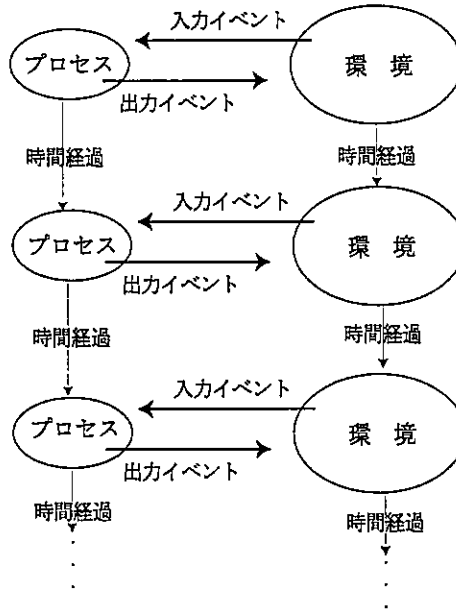


図 2: 無限ゲームの概念図

#### Definition 8 (Receptiveness の自動検証手法)

無限ゲームにおいて、プロセスの経過時間が発散するか、プロセスの状態遷移が有限回ならば、プロセスは *receptive* である。すなわち、プロセスが無限に動作する場合に、プロセスの経過時間が発散するかどうかを確かめればよい。その検証手順は以下のとおりである。

1. 経過時間を計測するために、リセットされないクロック変数 *now* を導入して、プロセスのリージョングラフ [8, 11] を構成する。
2. リージョングラフにおいて、初期状態集合から到達可能な強連結成分 [22] の集合を計算する。強連結成分を計算することは、無限に動作する状態列を計算することに対応する。
3. 強連結成分毎に、以下のすべての条件を満たす強連結成分が存在しなければ *receptive* である。
  - (a) 強連結成分の枝に出力イベントがある。
  - (b) 強連結成分の節点の中で、*now* 以外のクロック変数が 0 でない。
  - (c) 強連結成分の節点の中で、最大のクロック定数に関するクロック制約式が  $<$  または  $\leq$  である。  
この強連結成分はプロセスが有限時間内で無限に動作することを意味するので、この強連結成分が存在しなければプロセスは *receptive* である。

■

#### Example 2 (Receptiveness の事例)

*Receptiveness* の事例を図 3 に示す。図 3 (1) のプロセス A が *receptive* であるかどうかを検証する。ま

ず、経過時間を計測するために、リセットされないクロック変数  $now$  を導入して、プロセス  $A$  のリージョングラフを構成する。リージョングラフの強連結成分が無限な動作に対応している。この例の強連結成分では、プロセス  $A$  と環境が協調動作して、 $x = 0$  を含むので、経過時間は発散する。ゆえに、プロセス  $A$  は *receptive* である。

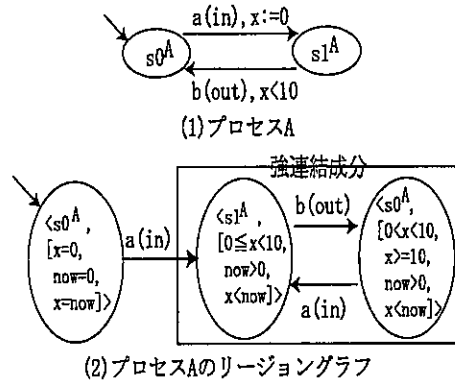


図 3: Receptiveness の事例

*Receptiveness* でない事例を図 4 に示す。まず、経過時間を計測するために、リセットされないクロック変数  $now$  を導入して、プロセス  $B$  のリージョングラフを構成する。この例の強連結成分では、プロセス  $B$  と環境が協調動作して、 $x < 1$  と  $x < 1$  でループしており、 $x = now$  なので、時間は発散しない。ゆえに、プロセス  $B$  は *receptive* でない。

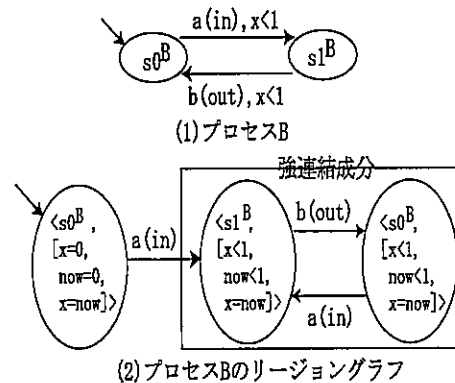


図 4: Receptiveness でない事例

### 2.3.4 Assume-Guarantee 形式による階層間の整合性検証

Assume-Guarantee 形式を基礎とする時間模倣関係による階層間の整合性の検証手法を定義する。プロセス 1 の抽象度の高い仕様  $P_1^A$  とプロセス 2 の抽象度の高い仕様  $P_2^A$  が並列動作する実時間システムがプロセス 1 の抽象度の低い仕様  $P_1^C$  とプロセス 2 の抽象度の低い仕様  $P_2^C$  が並列動作する実時間システムによって、正当に詳細化されていることを検証する場合を考える。この検証問題は、 $P_1^A \parallel P_2^A \preceq P_1^C \parallel P_2^C$  を示すことであるが、 $P_1^C \parallel P_2^C$  の状態数が多くなり、検証が困難である。そこで、以下のように、 $P_1^C \parallel P_2^C$  の構成を避けて、構成的に詳細化を検証する方法を提案する。ただし、 $P_1^A \parallel P_2^A$  は  $P_1^A$  と  $P_2^A$  が並列動作する実時間システムを意味する。

#### Theorem 1 (Assume-Guarantee 検証)

もし、 $P_1^A \parallel P_2^C \preceq P_1^C$  と  $P_1^C \parallel P_2^A \preceq P_2^C$  ならば、 $P_1^A \parallel P_2^A \preceq P_1^C \parallel P_2^C$  である。ただし、 $P_1^A$  と  $P_2^A$ ,  $P_1^C$ ,  $P_2^C$  は、すべて *receptiveness* を充足する。ここで、

- $P_1^A$  : プロセス 1 の抽象度の高い仕様
- $P_2^A$  : プロセス 2 の抽象度の高い仕様
- $P_1^C$  : プロセス 1 の抽象度の低い仕様
- $P_2^C$  : プロセス 2 の抽象度の低い仕様。

#### Proof 1

$P_1^A = (S_1^A, S_{01}^A, EVENT, C_1^A, E_1^A)$  及び  
 $P_2^A = (S_2^A, S_{02}^A, EVENT, C_2^A, E_2^A)$ ,  
 $P_1^C = (S_1^C, S_{01}^C, EVENT, C_1^C, E_1^C)$ ,  
 $P_2^C = (S_2^C, S_{02}^C, EVENT, C_2^C, E_2^C)$  とする。

証明すべき条件は以下である：

もし、 $(s_1^A \times s_2^C, s_1^C) \in R_1$  かつ  $(s_1^C \times s_2^A, s_2^C) \in R_2$  ならば、 $(s_1^A \times s_2^A, s_1^C \times s_2^C) \in R$  である。ただし、 $s_1^A \in S_1^A$ ,  $s_2^A \in S_2^A$ ,  $s_1^C \in S_1^C$ ,  $s_2^C \in S_2^C$  であり、 $R_1, R_2, R$  はそれぞれの時間模倣関係である。

1. すべてのプロセスが同期する場合を考える。

(1)  $(s_1^A \times s_2^C, s_1^C) \in R_1$  であり、すべてのプロセスは *receptive* なので、*receptive* の並列演算閉包性により、*receptive* なプロセス  $P_1^A \parallel P_2^C$  と  $P_1^C$  に関して、以下が成り立つ：

すべての  $e_1^A \in EVENT$ ,  $\lambda_1^A, \delta_1^A, \lambda_2^C, \delta_2^C$  について、

$$s_1^A \times s_2^C \xrightarrow{e_1^A, \lambda_1^A \cup \lambda_2^C, \delta_1^A \wedge \delta_2^C} s_1^A \times s_2^C /$$

である  $s_1^A \times s_2^C /$  が存在するならば、

$$s_1^C \xrightarrow{e_1^A, \lambda_1^C, \delta_1^C} s_1^C /$$

である  $s_1^{C1}$  が存在して,  $(s_1^{A1} \times s_2^{C1}, s_1^{C1}) \in R_1$  である.

(2)  $(s_1^C \times s_2^A, s_2^C) \in R_2$  であり, すべてのプロセスは *receptive* なので, *receptive* の並列演算閉包性により, *receptive* なプロセス  $P_1^C \parallel P_2^A$  と  $P_2^C$  に関して, 以下が成り立つ:

すべての  $e_2^A \in EVENT, \lambda_2^A, \delta_2^A, \lambda_1^C, \delta_1^C$  について,

$$s_1^C \times s_2^A \xrightarrow{e_2^A, \lambda_1^C \cup \lambda_2^A, \delta_1^C \wedge \delta_2^A} s_1^{C1} \times s_2^{A1}$$

である  $s_1^{C1} \times s_2^{A1}$  が存在するならば,

$$s_2^C \xrightarrow{e_2^A, \lambda_2^C, \delta_2^C} s_2^{C1}$$

である  $s_2^{C1}$  が存在して,  $(s_1^{C1} \times s_2^{A1}, s_2^{C1}) \in R_2$  である.

以上の (1) と (2), *receptiveness* の並列演算閉包性により, 以下が成り立つ:

すべての  $e^A \in EVENT, \lambda_1^A, \delta_1^A, \lambda_2^A, \delta_2^A$  について,

$$s_1^A \times s_2^A \xrightarrow{e^A, \lambda_1^A \cup \lambda_2^A, \delta_1^A \wedge \delta_2^A} s_1^{A1} \times s_2^{A1}$$

である  $s_1^{A1} \times s_2^{A1}$  が存在するならば,

$$s_1^C \times s_2^C \xrightarrow{e^A, \lambda_1^C \cup \lambda_2^C, \delta_1^C \wedge \delta_2^C} s_1^{C1} \times s_2^{C1}$$

である  $s_1^{C1} \times s_2^{C1}$  が存在して,  $(s_1^{A1} \times s_2^{A1}, s_1^{C1} \times s_2^{C1}) \in R$  である.

すなわち,  $(s_1^A \times s_2^A, s_1^C \times s_2^C) \in R$  が成り立つ.

2. プロセスが同期しない場合も同様に証明できる.

以上により, 定理の成り立つことが証明できた. ■

時間模倣関係を Assume-Guarantee 形式で自動検証するために, 時間模倣関係をリージョン模倣関係として検証する. なお, 時間模倣関係をリージョン模倣関係として検証する手法は文献 [11] で提案されている手法と同じである. このために, 時間 I/O オートマトン上の時間模倣関係の存在性問題はリージョングラフ上のリージョン模倣関係の存在性問題に帰着できることを示す [11].

まず, 上位レベルの仕様  $P^A = (S^A, S_0^A, EVENT, C^A, E^A)$  と下位レベルの仕様  $P^C = (S^C, S_0^C, EVENT, C^C, E^C)$  に対して, 以下を定義する.

1.  $RG_{P^A \parallel P^C} = (Q, Q_0, EVENT, E)$  は,  $P^A \parallel P^C$  のリージョングラフとする. ここで,
  - (a)  $Q : \langle s^A \times s^C, [\nu] \rangle$  の有限集合

- i.  $s^A \in S^A, s^C \in S^C$
  - ii.  $[\nu] = \{\nu' \mid \nu \equiv \nu'\}, \nu \in \Gamma(P^A \parallel P^C)$
  - (b)  $Q_0 : \langle s_0^A \times s_0^C, [\nu_0] \rangle$  の有限集合
    - i.  $s_0^A \in S_0^A, s_0^C \in S_0^C$
    - ii.  $[\nu_0]$ : 全てのクロック変数が 0 の順序対の集合
  - (c)  $EVENT$  : 有限イベント集合
  - (d)  $E \subseteq Q \times EVENT \times Q$  : 状態遷移関係
2.  $RG(s^A, s^C)$  は,  $\langle s^A \times s^C, [\nu] \rangle$  または  $\{\langle s^A, [\nu] \rangle, \langle s^C, [\nu] \rangle\}$  であり, 状態の対  $(s^A, s^C)$  が属するリージョングラフの同値類を表す.

以下に, リージョングラフ上のリージョン模倣関係を定義する.

**Definition 9 (リージョン模倣関係の定義)**

任意の  $RG(s_i^A, s_i^C) \in \chi$  に対して, 以下の条件が満たされるときに限り,  $\chi \subseteq RG_{P^A \parallel P^C}$  は  $P^A$  から  $P^C$  へのリージョン模倣関係であるという :

1. 任意の  $event \in EVENT$  に対して,

$$\langle s_i^A, [\nu_i] \rangle \xrightarrow{event} \langle s_{i+1}^A, [\nu_{i+1}] \rangle$$

ならば,  $RG(s_{i+1}^A, s_{i+1}^C) \in \chi$  である  $\langle s_{i+1}^C, [\nu_{i+1}] \rangle$  に対して,

$$\langle s_i^C, [\nu_i] \rangle \xrightarrow{event} \langle s_{i+1}^C, [\nu_{i+1}] \rangle$$

である.

2.  $\forall s_0^A \in S_0^A$  に対して,  $RG(s_0^A, s_0^C) \in \chi$  を満たす  $s_0^C \in S_0^C$  が存在する.

**Theorem 2 (時間模倣関係とリージョン模倣関係)**

$RG(s_i^A, s_i^C) \in \chi$  に対して,  $R_\chi = \{(s_i^A, s_i^C) \mid RG(s_i^A, s_i^C) \in \chi\}$  とする.  $\chi$  が  $P^A$  から  $P^C$  へのリージョン模倣関係である必要十分条件は,  $R_\chi$  が  $P^A$  から  $P^C$  への時間模倣関係  $P^A \preceq P^C$  である.

**Proof 2**

文献 [11] の Theorem 1 の証明と同様なので, 本論文では省略する.

以上より、時間模倣関係の存在性は、リージョン模倣関係の存在性に帰着して、検証する。ゆえに、時間I/Oオートマトンの間の時間模倣関係の存在性の検証アルゴリズムは以下の手続きから構成される [11]。まず、下位レベルの仕様と上位レベルの仕様の時間I/Oオートマトンのカルテジアン積を構成して、次に、カルテジアン積のリージョングラフを構成して、最後に、上位レベルの仕様から下位レベルの仕様へのリージョン模倣関係の存在性を検証する。

#### Definition 10 (時間模倣関係のアルゴリズム)

時間I/Oオートマトンの間の時間模倣関係に基づく検証アルゴリズムは以下の手続きから構成する。

1. 上位レベルの仕様と下位レベルの仕様の時間I/Oオートマトンのカルテジアン積を構成する。
2. カルテジアン積のリージョングラフを構成する。
  - (a) タイミング定数の組み合わせによって、クロックリージョンを構成する。
  - (b) 各状態をクロックリージョンに分割して、リージョングラフの状態集合を構成する。
  - (c) リージョングラフの状態集合をイベントで結ぶ。
3. リージョングラフ上のリージョン模倣関係を次のような手続きでチェックする。基本的考え方はリージョン模倣関係  $R$  を  $R(0), \dots, R(k), R(k+1)$  と帰納的に計算する。そして、 $R(k) = R(k+1)$  となる  $R(k)$  を  $R = R(k)$  とする。なお、上位レベルの仕様  $P^A = (S^A, S_0^A, EVENT, C^A, E^A)$  と下位レベルの仕様  $P^C = (S^C, S_0^C, EVENT, C^C, E^C)$  に対して、 $P^A \parallel P^C$  のリージョングラフを  $RG_{P^A \parallel P^C} = (Q, Q_0, EVENT, E)$  とする。ここで、

$Q : \langle s^A \times s^C, [\nu] \rangle$  の有限集合  
 $s^A \in S^A, s^C \in S^C$

$[\nu] = \{\nu \mid \nu \equiv \nu'\}, \nu \in \Gamma(P^A \parallel P^C)$

$Q_0 : \langle s_0^A \times s_0^C, [\nu_0] \rangle$  の有限集合  
 $s_0^A \in S_0^A, s_0^C \in S_0^C$

$[\nu_0] : \text{すべてのクロック変数が0の順序対の集合}$

$EVENT : \text{有限イベント集合}$

$E \subseteq Q \times EVENT \times Q : \text{状態遷移関係}$

$R(k)$  を計算するアルゴリズムは文献 [11] の Definition 12 と同様なので、本論文では省略する。 ■

## 2.4 実時間システムの階層的な設計の事例

### 2.4.1 設計支援システム

本手法を支援する設計支援システムは図5のように receptiveness 検証器と Assume-Guarantee 検証器から構成した。

1. receptiveness の検証は, receptiveness の並列演算閉包性により, プロセス毎に実現できる. receptiveness 検証器では, プログラミング言語形式で入力したプロセスから, 時間 I/O オートマトンのリージョングラフを生成して, receptiveness の条件をチェックする.
2. 時間模倣関係の Assume-Guarantee 検証は, リージョングラフ上の模倣関係として自動検証する. なお, リージョングラフは隣接リスト構造で表現する.

本設計支援システムにより, 上位レベルの仕様  $P_1^A, P_2^A$  から下位レベルの仕様  $P_1^C, P_2^C$  への時間模倣関係が存在するかどうかを Assume-Guarantee 形式で検証する. まず, プロセス毎に, receptiveness の充足性を検証して, receptiveness を充足するまでプロセスを修正する. 次に, Assume-Guarantee 形式により時間模倣関係の充足性を検証して, 充足するならば, さらに下位レベルの仕様を具体化して, 詳細な下位レベルの仕様を設計する. そうでなければ, 下位レベルの仕様を再設計して, 時間模倣関係の存在性を調べる. そして, 同様に, 下位レベルの仕様から詳細な下位レベルの仕様への時間模倣関係が存在するかどうかを検証する. 以上の設計作業を続けて, 最終的な実現仕様を設計する.

なお, SUN ULTRA(メインメモリ 24MB,143MHz) 上にインプリメントした設計支援システムの全体規模はC言語で7.3Kstepである.

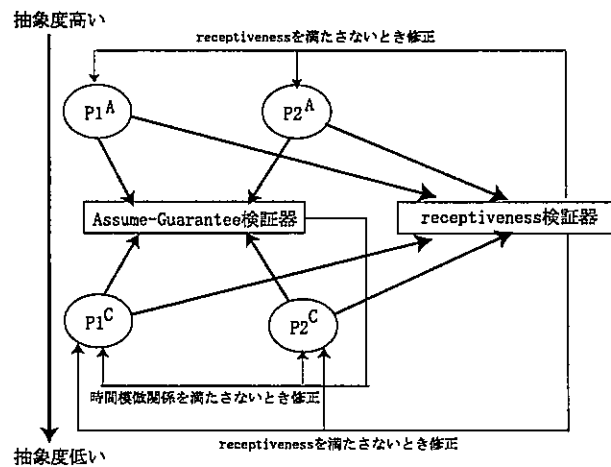


図 5: 設計支援システムの構成

## 2.4.2 階層的な設計の事例

本論文では、イーサネットのCSMA/CD [23] を基礎とした事例により、提案した階層的な設計手法の有効性を示す。

イーサネットのCSMA/CDはLANで広く使われており、送信局と受信局からなり、以下に送信局と受信局の各々を詳細に説明する。

1. 送信局はデータを送信する ( $send_i$ ) と、チャンネルの応答を感知する。もし、チャンネルがアイドルならば送信局はデータを送信する ( $begin_i$ )。しかし、チャンネルが busy であつたり ( $busy_i$ ) データが破壊されたら ( $cd_i$ )、10時刻未満待つて再送する ( $tau_i$ )。このイーサネットのCSMA/CDプロトコルの送信局の仕様は以下のように記述できる。

- まず、図6(1)のように、CSMA/CDプロトコルの*i*-送信局の上位レベルの仕様を設計して、時間I/Oオートマトンで仕様記述する。この仕様は、データを送信すると、チャンネルの busy やデータ破壊を繰り返して、正常にデータを送信することを意味する。
- 次に、図6(2)のように、CSMA/CDプロトコルの*i*-送信局の下位レベルの仕様を設計して、時間I/Oオートマトンで仕様記述する。下位レベルの仕様では、初期状態から非決定的に動作する。一方の動作はチャンネルの busy やデータ破壊を繰り返した後、正常に動作して、他方の動作はネットワーク上の何らかのトラブルにより、データの送信が失敗する可能性があることを意味する。

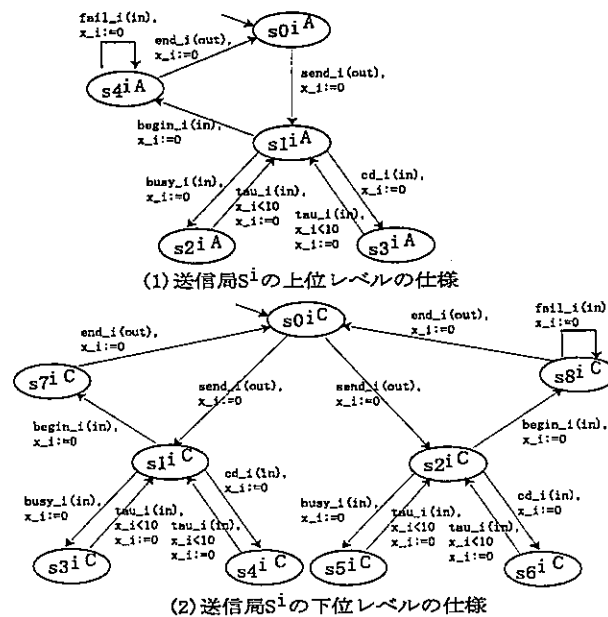


図 6: 送信局の仕様記述例

2. 受信局はデータ受信を開始し ( $send_i$ ), データが送信されると ( $begin_i$ ), 10時刻以内でデータを受信する ( $end_i$ ). もし, データ受信中にチャンネルが busy になったら ( $busy_i$ ), 10時刻以内のデータを受信待ちをして ( $fail_i$ ), その後データの受信を再開する. このイーサネットの CSMA/CD プロトコルの受信局の仕様は以下のように記述できる.

1. まず, 図7(1)のように, CSMA/CD プロトコルの  $i$ -受信局の上位レベルの仕様を設計して, 時間 I/O オートマトンで仕様記述する. この仕様は, データを受信すると, チャンネルの busy やデータ破壊を繰り返したり, 正常にデータが受信できることを意味する.
2. 次に, 図7(2)のように, CSMA/CD プロトコルの  $i$ -受信局の下位レベルの仕様を設計して, 時間 I/O オートマトンで仕様記述する. 下位レベルの仕様では, 初期状態から非決定的に動作する. 一方の動作はチャンネルの busy を繰り返した後, データ受信を失敗し, 他方の動作はネットワーク上のトラブルが全くなく, データの受信を正常に行なえたり, データ破壊によりデータ受信が失敗することを意味する.

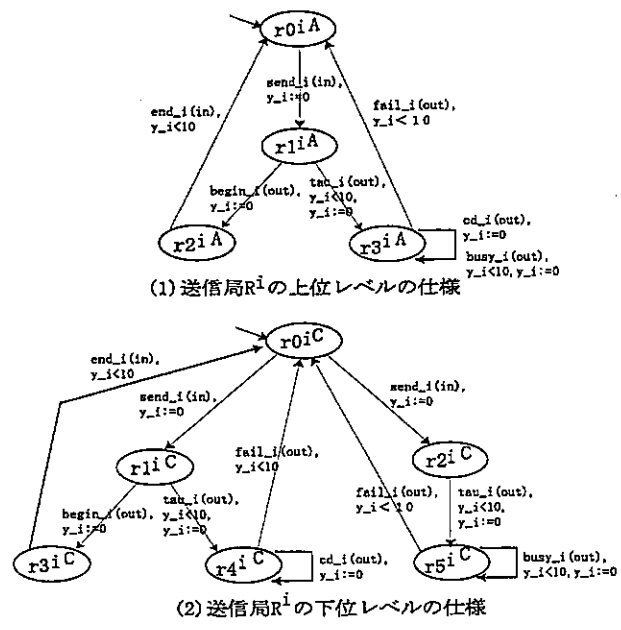


図 7: 受信局の仕様記述例

次に, Assume-Guarantee 形式の時間模倣関係により, 設計した上位レベルと下位レベルの仕様の整合性を検証する. 本論文では, 通信システムは複数の送信局と受信局の仕様から構成されるものと考えられる. すなわち, 通信システムは送信局と受信局の仕様のカルテジアン積である. 整合性検証問題では, これらの通信システムの下位レベルの仕様が上位レベルの仕様に対して, 時間模倣関係が存在するかどうかを判定する. すなわち,  $S^A \parallel R^C \leq S^C$  と  $S^C \parallel R^A \leq R^C$  により,  $S^A \parallel R^A \leq S^C \parallel R^C$  を

検証する。ただし、 $S^A$  と  $R^A$ ,  $S^C$ ,  $R^C$  は、すべて receptiveness を充足する。ここで、 $S^A$  と  $R^A$  は送信局及び受信局の上位レベルの仕様、 $S^C$  と  $R^C$  は送信局及び受信局の下位レベルの仕様である。

まず、プロセス毎に、receptiveness の充足性を検証した。図6と図7のすべてのプロセスは receptiveness を充足していることがわかった。次に、送信局と受信局の総数が2個、4個、6個、8個の場合の整合性の検証実験を行なって、Assume-Guarantee 検証手法と従来検証手法との比較評価を行った。UNIX の time コマンドで計測した検証の所要メモリと計算時間の比較結果を図8に示す。実験結果より、Assume-Guarantee 検証により、大幅に検証コストが削減できることがわかった。

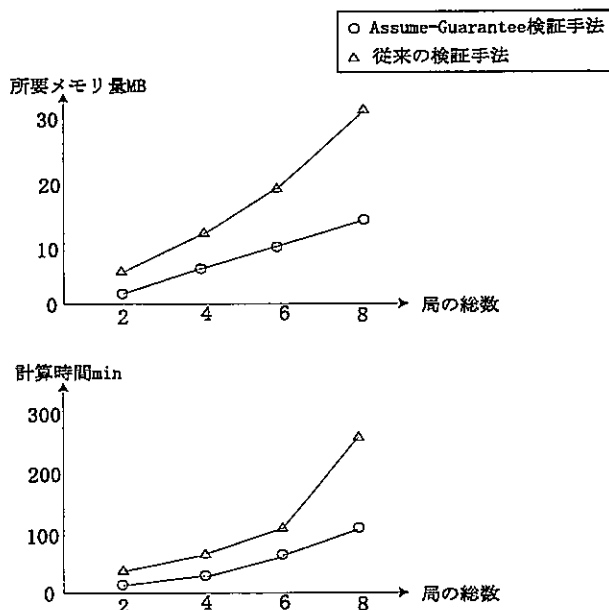


図 8: 時間模倣関係の検証コスト

## 2.5 むすび

本論文では、要求仕様から設計仕様を得るまでの各階層の仕様を同一のリアクティブな時間オートマトンの一種で記述し、階層間の整合性を時間模倣関係の Assume-Guarantee 形式で自動検証するための枠組みを提案した。そして、計算機実験により、その有効性を実証した。類似研究として、時間モジュールの Assume-Guarantee 形式による階層的設計支援 [13] が存在するが、本研究と異なり、階層間の整合性関係として言語包含関係を使用しており、さらに Assume-Guarantee 検証手法を述べていない。また、時間制約を考慮しないリアクティブシステムの階層的な設計を支援するシステムとしては、Concurrency workbench [24] や SMC [5] など多く存在する。一方、実時間システムの階層的な設計を支援するシステムとしては、Real-time Step [51] や KRONOS [26], Real-time COSPAN

[27], UPPAAL [28] などが存在する. Real-time Step は実時間時相論理の証明系であり, KRONOS と UPPAAL は実時間シンボリックモデルチェッカであり, 階層設計にはあまり有効ではない. また, Real-time COSPAN は抽象度の異なる仕様間に準同形写像を定義して, 階層的設計を支援しているが, 手作業で準同形写像を定義する必要があり, あまり実用的ではない. 以上により, 世界ではじめて, 実時間システムの Assume-Guarantee 形式の自動階層設計支援システムが構築できて, その有効性が確認できた.

今後の課題としては以下が考えられる.

1. BDDs(Binary Decision Diagrams) [29]などを基礎とするシンボリック検証技術による検証の所要メモリ量の削減
2. 抽象実行などによる所要メモリ量と計算時間の削減

## 3 実時間型開放分散システムの演繹的設計支援

### 3.1 まえがき

通信システムや制御システムの多くはハードリアルタイムシステムである [30]. さらに, これらは他のシステムと協調して動作するリアクティブシステムであることもしばしばである. したがって, これらのシステムの設計においては, 各システムにおける処理時間が他のシステムに及ぼす影響や, システム間の通信遅延といった実時間性をも考慮する必要がある. このことにより, (リアクティブ) ハードリアルタイムシステムの設計作業は複雑になりがちであり, また, システムの信頼性保証も一般的には困難である. したがって, ハードリアルタイムシステムの信頼性保証ができることは重要である. 本論文では, ハードリアルタイムシステムの信頼性保証のために, 形式的仕様記述言語と形式的検証手法を提案する. 具体的には, 以下のとおりである: まず, 時間状態チャート [31] を一般化して, 並行性やタイミング制約, 機能を統合的かつ形式的に仕様記述できる言語を提案する. 従来の形式的仕様記述言語では, 並行性やタイミング制約, 機能が統合的かつ形式的に仕様記述できない. さらに, 一般化した時間状態チャートが安全性や活性といった望ましい性質を充足するかどうかを形式的に演繹的検証する手法を提案する. そして, 一般化した時間状態チャートにより, ハードリアルタイムシステムを階層的に詳細化して設計できる手法, すなわち, 詳細化を形式的に演繹的検証する手法を提案する. 従来の形式的検証手法では, ハードリアルタイムシステムの形式的な詳細化演繹的検証手法は存在しない.

従来のハードリアルタイムシステムの形式的手法の主要な研究では, 以下のようなものがある:

1. Alur や Dill らは時間オートマトンを開発して, ハードリアルタイムシステムの形式的仕様記述言語とした [48]. さらに, 時間オートマトンの形式的検証手法を開発した [21]. しかし, 時間オートマトンは有限状態遷移システムであり, 複雑なシステムが仕様記述できない.
2. Lynch らは, 時間 I/O オートマトンを開発して, 無限な状態集合を有するシステムの仕様記述と詳細化の検証の公理系を開発した [10]. しかし, 時間 I/O オートマトンは抽象的であり, 動作モデル (いわゆる状態遷移モデル) や機能モデル (いわゆるデータ処理) などの区別が明確でないために, 実際のシステムの設計に使えない.
3. Kesten や Manna, Pnueli は, クロック遷移システム (clocked transition systems) を開発して, 無限な状態集合を有するシステムの仕様記述と安全性や活性の検証の公理系を開発した [51]. しかし, クロック遷移システムは並行性や機能などが陽に表現できないために, 仕様記述言語には適さない. さらに, クロック遷移システムには, 詳細化の検証の公理系が開発されていない.

本節では, まず, 時間状態チャート [31] を一般化して, 並行性やタイミング制約, 機能を統合的に仕様記述する. なぜならば, 既存の時間状態チャート [31] では, 状態遷移モデルをタイミング制約記述で拡張したのみであり, 値の変数への代入などの機能が表現できないので, 仕様記述言語としては表現能力が弱いからである.

次に、演繹的証明により、時間状態チャートの安全性や活性、詳細化を形式的に検証する。従来の状態チャートの研究では、動作モデルや機能モデルが独立しているために形式化が不十分であったり [34]、タイミング制約と機能を統合的に形式化していない [35]。また、ハードリアルタイムシステムの詳細化の検証の公理系が開発されていない [51]。

本節では、まず、時間状態チャートを定義して、その操作的意味をクロック遷移システム上で定義する。次に、安全性や活性を時相論理式で仕様記述して、クロック遷移システムが時相論理式を充足するかどうかを、演繹的証明で検証する手法を示す。次に、詳細化の検証の公理系により、時間状態チャートの詳細化を検証する。

本論文の構成は、次のとおりである。3. 2章では、時間状態チャートを定義する。3. 3章では、時間状態チャートの操作的意味をクロック遷移システム上で定義して、安全性や活性の検証手法を定義する。3. 4章では、時間状態チャートの操作的意味をクロック遷移システム上で定義して、詳細化の検証手法を定義する。最後に、3. 5章では、まとめと今後の課題を述べる。

## 3.2 時間状態チャートによる仕様記述

一般的には、ハードリアルタイムシステムは動作モデルと機能モデルから構成される。本節では、ハードリアルタイムシステムは動作モデルが大きく、機能モデルは小さいと考える。これにより、動作モデルのアクションの中に機能モデルを押し込む。そして、タイミング制約が記述できるように拡張して、時間状態チャート [31] が並行性と機能モデル、タイミング制約を統合的に表現できるようにする。

### 3.2.1 ステートチャートの概要

ステートチャート [36] はシステムが時間的に反応しているときに、動作がいつ、どのように、なぜ発生しているかを表現する。状態組み合わせ爆発を抑制するために、ステートチャートでは、AND形式(並列形式)とOR形式(階層形式)の状態遷移図で表現して、イベントがブロードキャスト通信される機構が備わっている。ブロードキャスト通信とは、ステートチャートを構成する、すべてのAND形式とOR形式の状態遷移図にイベントが通信されることを意味する。さらに、状態への遷移及び状態からの遷移がどの階層でも実現できるようになっている。図9に従って、ステートチャートの基本概念を簡単に説明する。図9 (1)(a)のOR形式の状態遷移図では、状態Dに存在することは状態Aまたは状態Cに存在することである。ゆえに、図9 (1)(a)は図1 (1)(b)と等しい。また、図9 (2)(a)のAND形式の状態遷移図では、状態Aと状態Dが並行動作して、イベント $\alpha$ とイベント $\beta$ は状態Aと状態Dにブロードキャスト通信される。ゆえに、図9 (2)(a)は図1 (2)(b)と等しい。以上のOR形式とAND形式を組み合わせると、図9 (3)のような一般的なステートチャートの基本形が構成できる。

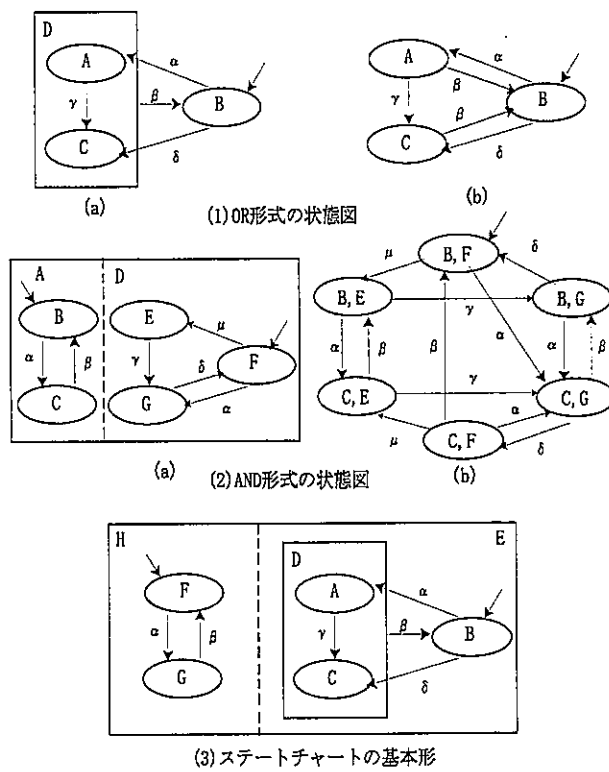


図 9: ステートチャートの例

### 3.2.2 時間ステートチャート

本節で対象とする時間ステートチャートは、ステートチャートを拡張したものであり、AND/OR形式の状態遷移図とブロードキャスト通信機構があり、状態遷移は  $\delta, \lambda, e/a$  または  $e/a$  でラベル付けされているとする。なお、 $e$  が外部からのイベントの場合は  $\delta, \lambda, e/a$  であり、 $e$  が内部イベントの場合は  $e/a$  である。ここで、 $\delta$  はタイミング制約式、 $\lambda$  はリセット式、 $e$  はイベント、 $a$  はアクションとする。

1. タイミング制約式  $\delta$  は、クロック集合  $C(x \in C)$  と非負整数の時刻定数  $d$  により以下のように帰納的に定義される：
  - (a)  $x \leq d$  や  $d \leq x$  は  $\delta$  である。
  - (b)  $\delta_1$  と  $\delta_2$  が  $\delta$  ならば、 $-\delta_1$  や  $\delta_1 \wedge \delta_2$  は  $\delta$  である。
2.  $\lambda$  はリセット式の集合であり、リセット式は  $x := 0(x \in C)$  で表現する。
3. イベントはある変数が特定の値に等しいといった条件文である。
4. アクションはある変数をある値に代入するといった実行文である。

さらに、以下の前提条件を仮定する。

#### 1. 同期性仮定

システムは環境よりも無限に速く動作して、環境からの刺激に対してシステムは瞬時に反応すると仮定する。例えば、 $t_0 : a/e_1, t_1 : e_1/e_2, \dots, t_n : e_n/b$  の状況において、 $a$  と  $b$  は環境との刺激・応答として、 $e_1, e_2, \dots, e_n$  は内部イベント又は内部アクションとすると、 $t_0, t_1, \dots, t_n$  は瞬時にチェイン状に発生する。ここで、各  $t_i (i \geq 0)$  はステップと呼び、 $\cup t_i$  はスーパーステップと呼ぶ。この仮定は、モデル化を単純にするための抽象化である。

#### 2. 因果関係保存

ステップに順番を付与して、順番が大きな遷移から小さな遷移が発生しないようにする。例えば、 $t_1 : a/b, t_2 : b/a$  の状況において、 $t_1, t_2, t_1, \dots$  は無限に発生する可能性があるが、因果関係の保存により、 $t_1, t_2$  で遷移が停止する。この仮定は、内部イベント/アクションの無限動作を回避するために、必要である。

#### 3. プライオリティの表現

同時に発生したイベントのどれが実行すべきかを指定できるようにする。例えば、 $t_1 : a/, t_2 : b/$  の状況において、 $a$  と  $b$  が同時に発生すると、非決定的に、どちらかが動作する。しかし、リアルタイムシステムでは、優先度を付けて実行させたいことがある。そこで、否定表現により、優先度を指定する。この例では、 $t_1 : a \cdot \bar{b}/, t_2 : b/$  のように記述することにより、 $a$  と  $b$  が同時に発生したときに、 $t_2 : b/$  が動作するように指定する。これにより、実行の優先度が指定できる。

次に、時間ステートチャートを形式的に定義する。

#### Definition 11 (時間ステートチャート)

時間ステートチャートは、 $TSC = (Event, Act, C, Q, Q_0, E, \rho, \phi)$  の8つ組で定義される。

1.  $Event$  はイベントの有限集合.
2.  $Act$  はアクションの有限集合.
3.  $C$  はクロック変数の有限集合.
4.  $Q$  は有限状態集合.
5.  $Q_0 \subseteq Q$  は初期状態集合.
6.  $E \subseteq 2^Q \times 2^Q \times 2^C \times \Phi(C) \times 2^{Event} \times 2^{Act}$  は状態遷移関係.  
ただし,  $\Phi(C)$  はクロック集合  $C$  のタイミング制約式  $\delta$  であり,  $2^C$  はリセットされるクロック集合の集合である.
7.  $\rho: Q \rightarrow 2^Q$  は各状態の下位階層の状態集合を決める階層関数.
8.  $\phi: Q \rightarrow \{BASIC, OR, AND\}$  は各状態の型を決める型関数.

■

状態遷移関係  $E$  は, 現在の状態と次状態, リセットされるクロック変数, イベント, アクションのベキ集合とタイミング制約式のカルテジアン積の部分集合である. イベントは, ある変数が特定の値であるといった条件式である. ここで, 変数を  $var_1, var_2 \in VAR$ , 変数の値を  $value_1, value_2 \in VALUE$  とすると, イベントは以下のように帰納的に定義される:

1.  $var_1 = value_1$  はイベント  $e$  である.
2.  $e_1$  と  $e_2$  がイベント  $e$  ならば,  $\neg e_1$  や  $e_1 \wedge e_2$  はイベントである.

アクションは, ある変数へのデータの代入である. アクションは以下のように帰納的に定義される:

1.  $var_1 := value_1$  はアクション  $a$  である.
2.  $a_1$  と  $a_2$  がアクション  $a$  ならば,  $a_1 \parallel a_2$  はアクションである. ただし,  $\parallel$  は並列動作することを意味する.

AND 形式により, 並列関係にある状態集合が表現できる. OR 形式により, 状態集合が階層構造を形成しており, 階層関数  $\rho$  によって下位階層の状態集合を定義する. また, 各状態は, 型関数  $\phi$  により, 階層の一番上のルート状態であるか, 又は AND/OR 形式に属するかが定義される.

### Example 3 (時間ステートチャートによる仕様記述例)

簡単なエアコン制御システムを考える. エアコン制御システムは, センサ感知プログラムより送信される, 温度計とドアに関するメッセージに反応して, アクチュエータ駆動プログラムにメッセージを送信してエアコンを作動する. すなわち, 部屋が *hot* または *vhot* の状態でドアが *close* の状態のときには, アクチュエータ駆動プログラムにメッセージを送信してエアコンを作動して ( $VS\_assert := on$  かつ  $ac := 1$ ), そうでないときにはエアコンを停止させる ( $VS\_assert := \perp$  かつ  $ac := \perp$ ). つまり, エアコン制御システムは, 温度とドアの状態により, エアコン駆動を制御する.

このエアコン制御システムの要求仕様を、時間状態チャートで仕様記述する。時間状態チャートの状態空間は AND 形式 (*TEMP* と *DOOR*) と OR 形式 (*hot* と *vhot*) から構成される。ここで、*message* は列挙型であり、*room\_is\_hot* や *room\_is\_cool*, *close\_door*, *open\_door* といった値を取る。また、*is\_room\_hot* や *ac*, *is\_door\_close* は 1 または 0 を含む自然数であり、*VS\_assert* はブール型であり、*c<sub>1</sub>*, *c<sub>2</sub>* はクロック変数である。図 10 にエアコン制御システムの時間状態チャートによる仕様記述例を示す。その動作の概要は以下のとおりである：まず、動作の開始時は状態 *cool* と状態 *open* にあり、動作の開始時から 5 時刻未満にメッセージ *message = room\_is\_hot* が送信されると、アクション *is\_room\_hot := 1* を実行して状態 *hot* と状態 *open* に遷移する。次に、状態 *hot* と状態 *open* において、状態 *hot* に存在してから 7 時刻未満にメッセージ *message = room\_is\_hot* が送信されると、アクション *is\_room\_hot := 2* を実行して状態 *vhot* と状態 *open* に遷移する。 ■

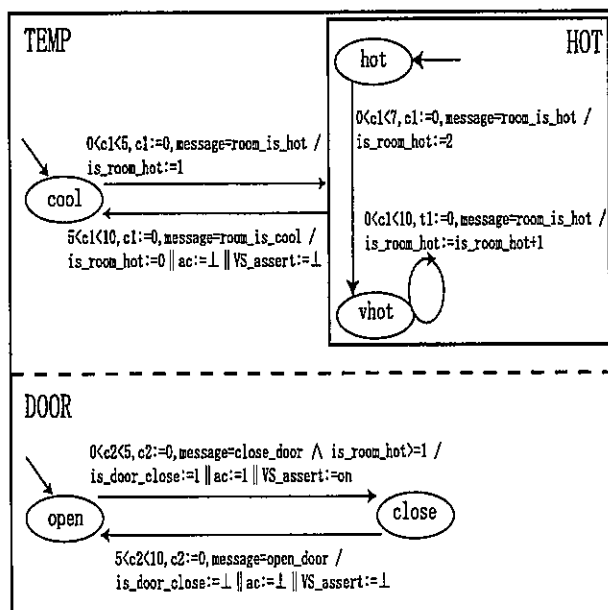


図 10: 時間状態チャートの仕様記述例

### 3.3 時間状態チャートの意味

本章では、クロック遷移システム上で時間状態チャートの操作的意味を表現する。最初にクロック遷移システムを定義して、次に時間状態チャートからクロック遷移システムへの変換方法を定義して、その操作的意味を定義する。

### 3.3.1 クロック遷移システム

ハードリアルタイムシステムの計算モデルとしてクロック遷移システム [51] を定義する。

まず、システム変数の有限集合を考える。システム変数は整数や実数などの型を持つ。我々はシステム変数にその型の値を割り付ける解釈として状態  $s$  を定義する。すべての状態の集合を  $\Sigma$  とする。

#### Definition 12 (クロック遷移システム)

クロック遷移システムは、 $CTS = (V, \Theta, T)$  の3つ組で定義される。ここで、

1.  $V$  : システム変数の有限集合。集合  $V = D \cup C$  は、離散変数の集合  $D = \{u_1, \dots, u_n\}$  とクロック変数の集合  $C = \{c_1, \dots, c_k\}$  に分類できる。離散変数は任意の型がありえるが、クロック変数は実数型である。さらに、リセットされないマスタクロック  $T \in C$  を導入する。
2.  $\Theta$  : 初期条件。これは、すべての初期状態を特徴付ける表明である。 $\Theta \rightarrow c_1 = \dots = c_k = T = 0$  が要求される。
3.  $T$  : 状態遷移の有限集合。各状態遷移  $\tau \in T$  は関数  $\tau : \Sigma \rightarrow 2^\Sigma$  であり、各状態  $s \in \Sigma$  に次状態  $\tau$ -successor  $\tau(s) \subseteq \Sigma$  を写像する。状態遷移  $\tau$  に関連する関数は表明  $\rho_\tau(V, V')$  により表現される。 $\rho_\tau(V, V')$  は状態遷移関係と呼び、状態  $s \in \Sigma$  を  $\tau$ -successor  $s' \in \tau(s)$  に関係付ける。ただし、状態  $s$  は  $V$  の型整合解釈であり、各変数  $v \in V$  に値  $s[v]$  を割り付ける。なお、システム変数の値は  $s$  の中の値や  $s'$  の中の値として参照する。任意の  $\tau \in T$  に対して、 $\rho_\tau \rightarrow T' = T$  が要求される。なお、 $\rho_\tau \rightarrow T' = T$  は状態遷移が瞬時に発生することを意味する。

■

### 3.3.2 時間状態チャートからクロック遷移システムへの変換

最初に時間状態チャートからフラットな時間状態チャートに変換して、次にフラットな時間状態チャートからクロック遷移システムに変換する。

まず、時間状態チャートからフラットな時間状態チャートへの変換を説明する。時間状態チャートの状態集合は、唯一のルート状態をもつ AND/OR ツリーを構成している。ゆえに、文献 [31] の手法により、ルート状態から、 $\phi(q)=\text{AND}$  ならば  $\rho(q)$  の並列合成を構成して、 $\phi(q)=\text{OR}$  ならば  $\rho(q)$  の階層を展開すれば、AND/OR 状態が存在しないフラットな時間状態チャート  $TSC' = (Event, Act, L, Q', Q_0', E')$  を構成できる。ただし、 $q \in Q$  とする。AND 形式の状態図を並列合成してフラットな状態図を構成すると、状態数が指数的に増大する。これは、積オートマトンの非空性判定問題が PSPACE 完全 [37] であることに裏付けられており、避けることができない問題である。

時間状態チャートから構成されるフラットな時間状態チャートは、以下のように定義される。

#### Definition 13 (フラットな時間状態チャート)

フラットな時間状態チャートは  $TSC^I = (Event, Act, C, Q^I, Q_0^I, E^I)$  の6つ組で定義される。

1.  $Event$  はイベントの有限集合.
2.  $Act$  はアクションの有限集合.
3.  $C$  はクロック変数の有限集合.
4.  $Q^I = Q_1 \times Q_2 \times \dots \times Q_n$  ( $n \in \text{自然数}$ ) は有限状態集合. ただし,  $n$  は並列動作する状態数である.
5.  $Q_0^I \subseteq Q^I$  は初期状態集合.
6.  $E^I \subseteq Q^I \times Q^I \times 2^C \times \Phi(C) \times 2^{Event} \times 2^{Act}$  は状態遷移関係.

#### Example 4 (時間状態チャートによる仕様記述例)

図10のエアコン制御システムの時間状態チャートから, フラットな時間状態チャートが構成できる. 図11に, エアコン制御システムの時間状態チャートから構成したフラットな時間状態チャートを示す.

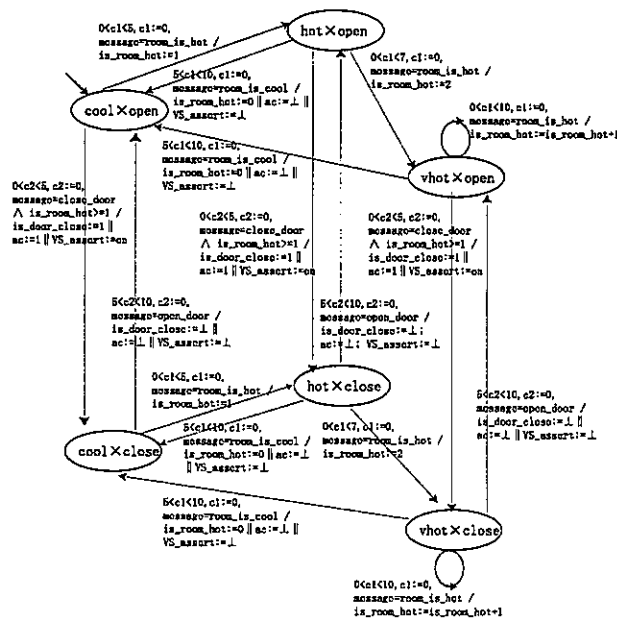


図 11: フラットな時間状態チャートの構成例

次に, フラットな時間状態チャートからクロック遷移システムへの変換を説明する. 一番目に, フラットな時間状態チャートからクロック遷移システムへの変換方法を定義する.

**Definition 14 (クロック遷移システムへの変換)**

フラットな時間状態チャート  $TSCl = (Event, Act, C, Ql, Q_0l, El)$  とクロック遷移システム  $CTS = (V, \Theta, T)$  が与えられたとき、以下のように、フラットな時間状態チャートからクロック遷移システムを構成する。

1.  $V = D \cup C$  は、離散変数の集合  $D = \{u_1, \dots, u_m\}$  とクロック変数の集合  $C = \{c_1, \dots, c_n, T\}$  からなる。 $D$  と  $C$  は以下のように構成される：
  - (a) 離散変数の集合  $D$  は並列動作する状態の集合  $Ql$  及びイベントとアクションに関わる変数の集合  $VAR$  から構成される。 $D = \{u_1, \dots, u_n, var_1, \dots, var_k\}$ , ただし,  $u_1 = q_1 \in Q_1, \dots, u_n = q_n \in Q_n, var_1, \dots, var_k \in VAR$ . なお,  $n$  は並列動作する状態の数であり,  $k$  は変数の数であり,  $m = n + k$  である。
  - (b)  $C$  は時間状態チャート上に現れるクロック変数  $c_1, \dots, c_n$  及びマスタクロック  $T$  から構成される。
2.  $\Theta$  は初期条件であり、以下のように構成される：
  - (a)  $D$  のすべての要素が初期値と等しくなるようにする。
  - (b)  $C$  に関しては  $c_1 = \dots = c_n = T = 0$  であるようにする。
3.  $T$  は状態遷移の有限集合である。状態遷移  $\tau \in T$  に関連する関数を表明  $\rho_\tau(V, V')$  により表現する。 $\rho_\tau(V, V')$  は現在の変数の値と次状態の変数の値との関係により定義する。

■

二番目に、フラットな時間状態チャートの動作がどのようにクロック遷移システムの動作に反映されるかを定義することによって、時間状態チャートの操作的意味を定義する。

**Definition 15 (時間状態チャートの操作的意味)**

フラットな時間状態チャートの動作がどのようにクロック遷移システムの動作に反映されるかを定義することによって、時間状態チャートの操作的意味を定義する。フラットな時間状態チャート  $TSCl = (Event, Act, C, Ql, Q_0l, El)$  とクロック遷移システム  $CTS = (V, \Theta, T)$  が与えられたとき、以下のように、操作的意味を定義する。

まず、フラットな時間状態チャート  $TSCl$  の動作を以下のように定義する。状態遷移は  $\delta, \lambda, e/a$  または  $e/a$  でラベル付けされているので、以下の2つの場合を考える。

1.  $\delta, \lambda, e/a$  のとき

$$q_i' \xrightarrow{\delta, \lambda, e/a} q_{i+1}'$$

ただし,  $q_i', q_{i+1}' \in Ql$ ,  $\delta$  はタイミング制約式,  $\lambda$  はリセット式,  $e$  はイベント,  $a$  はアクションである。

2.  $e/a$  のとき

$$q_i' \xrightarrow{e/a} q_{i+1}'$$

ただし,  $q_i', q_{i+1}' \in Q'$ ,  $e$  はイベント,  $a$  はアクションである.

次に, 上記に対応するクロック遷移システム  $CTS = (V, \Theta, T)$  の動作を以下のように定義する:

1.  $\delta, \lambda, e/a$  のとき

$(u_1 = q_1 \wedge \dots \wedge u_n = q_n \wedge var_1 = value_1 \wedge \dots \wedge var_k = value_k \wedge t_1 < c_1 < t_1' \wedge \dots \wedge t_n < c_n < t_n' \wedge t_T < T < t_T') \rightarrow (u_1 = q_1^{after} \wedge \dots \wedge u_n = q_n^{after} \wedge var_1 = value_1^{after} \wedge \dots \wedge var_k = value_k^{after} \wedge t_1^{after} < c_1 < t_1'^{after} \wedge \dots \wedge t_n^{after} < c_n < t_n'^{after} \wedge t_T^{after} < T < t_T'^{after})$ . なお,  $after$  は遷移後の状態名や変数値などを意味する記号である. ただし, 以下を満たす:

- (a)  $q_i' = q_1 \times \dots \times q_n$  及び  $q_{i+1}' = q_1^{after} \times \dots \times q_n^{after}$ .
- (b)  $\delta$  が表現する時間領域と  $t_1 < c_1 < t_1' \wedge \dots \wedge t_n < c_n < t_n'$  が表現する時間領域との交わり [38] が空集合でない.
- (c)  $\lambda$  によりリセットされるクロック変数のリセット値と  $t_1^{after} < c_1 < t_1'^{after} \wedge \dots \wedge t_n^{after} < c_n < t_n'^{after}$  が表現する時間領域との交わり [38] が空集合でない.
- (d) イベント  $e$  による, ある変数がある値であるといった条件  $var_i = value_i$  は  $var_1 = value_1 \wedge \dots \wedge var_i = value_i \wedge \dots \wedge var_k = value_k$  と矛盾しない.
- (e) アクション  $a$  による変数の更新  $var_i := value_i^{after}$  が  $var_1 = value_1^{after} \wedge \dots \wedge \dots \wedge var_i = value_i^{after} \wedge \dots \wedge var_k = value_k^{after}$  に反映されている.

2.  $e/a$  のとき

$(u_1 = q_1 \wedge \dots \wedge u_n = q_n \wedge var_1 = value_1 \wedge \dots \wedge var_k = value_k \wedge t_1 < c_1 < t_1' \wedge \dots \wedge t_n < c_n < t_n' \wedge t_T < T < t_T') \rightarrow (u_1 = q_1^{after} \wedge \dots \wedge u_n = q_n^{after} \wedge var_1 = value_1^{after} \wedge \dots \wedge var_k = value_k^{after} \wedge t_1^{after} < c_1 < t_1'^{after} \wedge \dots \wedge t_n^{after} < c_n < t_n'^{after} \wedge t_T^{after} < T < t_T'^{after})$ . なお,  $after$  は遷移後の状態名や変数値などを意味する記号である. ただし, 以下を満たす:

- (a)  $q_i' = q_1 \times \dots \times q_n$  及び  $q_{i+1}' = q_1^{after} \times \dots \times q_n^{after}$ .
- (b) イベント  $e$  による, ある変数がある値であるといった条件  $var_i = value_i$  は  $var_1 = value_1 \wedge \dots \wedge var_i = value_i \wedge \dots \wedge var_k = value_k$  と矛盾しない.
- (c) アクション  $a$  による変数の更新  $var_i := value_i^{after}$  が  $var_1 = value_1^{after} \wedge \dots \wedge \dots \wedge var_i = value_i^{after} \wedge \dots \wedge var_k = value_k^{after}$  に反映されている.

■

**Example 5 (クロック遷移システムへの変換例)**

図 1 1 のフラットな時間状態チャートからクロック遷移システム  $CTS = (V, \Theta, T)$  を構成する. エアコン制御システムの時間状態チャートは, 以下のようにクロック遷移システムに変換できる:

- 1.  $V = D \cup C$  であり,  $D$  と  $C$  は以下のように定義できる.  $D = \{u_1, u_2, message, is\_room\_hot, is\_door\_close, ac, VS\_assert\}$ .  $C = \{c_1, c_2, T\}$ . ここで,  $u_1$  と  $u_2$  は  $TEMP$  と  $DOOR$  の状態を表す変数である. また,  $c_1$  と  $c_2$  はクロック変数であり,  $TEMP$  のタイミング制約は  $c_1$  で表現し

て, *DOOR* のタイミング制約は  $c_2$  で表現する.

2.  $\Theta$  は以下のように定義できる.

$$\Theta = (u_1 = cool \wedge u_2 = open \wedge message = \perp \wedge is\_room\_hot = \perp \wedge is\_door\_close = \perp \wedge ac = \perp \wedge VS\_assert = \perp \wedge c_1 = 0 \wedge c_2 = 0 \wedge T = 0).$$

ただし,  $\perp$  は未定義を意味する.

3. 以下のように状態遷移関係  $\rho_\tau$  が定義される:

(a)  $cool \times open$  から  $hot \times open$  に状態遷移するとき

$$(u_1 = cool \wedge u_2 = open \wedge message = \perp \wedge is\_room\_hot = \perp \wedge is\_door\_close = \perp \wedge ac = \perp \wedge VS\_assert = \perp \wedge 0 \leq c_1 \leq 5 \wedge 0 \leq c_2 \leq 5 \wedge 0 \leq T \leq 5) \wedge (u_1' = hot \wedge u_2' = open \wedge message' = room\_is\_hot \wedge is\_room\_hot' = 1 \wedge is\_door\_close' = \perp \wedge ac' = \perp \wedge VS\_assert' = \perp \wedge c_1' = 0 \wedge c_2' = 0 \wedge 0 \leq T' \leq 5).$$

(b)  $hot \times open$  から  $vhot \times open$  に状態遷移するとき

$$(u_1 = hot \wedge u_2 = open \wedge message = room\_is\_hot \wedge is\_room\_hot = 1 \wedge is\_door\_close = \perp \wedge ac = \perp \wedge VS\_assert = \perp \wedge 0 \leq c_1 \leq 7 \wedge 0 \leq c_2 \leq 7 \wedge 0 \leq T \leq 7) \wedge (u_1' = vhot \wedge u_2' = open \wedge message' = room\_is\_hot \wedge is\_room\_hot' = 2 \wedge is\_door\_close' = \perp \wedge ac' = \perp \wedge VS\_assert' = \perp \wedge c_1' = 0 \wedge c_2' = 0 \wedge T' \geq 0).$$

.....  
 .....

■

### 3.4 時間ステートチャートの安全性と活性の検証

時間ステートチャートが安全性や活性を充足することを検証するために, 時間ステートチャートから構成したクロック遷移システムで演繹的証明する. この場合, 時相論理の演繹的検証ルール [51] を使用する.

#### 3.4.1 時相論理

まず, 時相論理 [39] の部分集合であり, 公理系で証明する時相論理 [51] の構文を定義する.

##### Definition 16 (時相論理の構文)

時相論理式の集合は, 以下の規則で生成される論理式の集合のうち最小のものである:

1. 各原子命題  $P$  は論理式である. ただし, 原子命題はクロック遷移システムのシステム変数の集合への値の割り当てにより定義される表明式である.

2.  $p$  が論理式ならば,  $\Box p$  は論理式である.
3.  $p$  と  $q, r$  が論理式ならば,  $\Box (p \rightarrow (qWr))$  は論理式である.
4.  $p$  と  $r$  が論理式ならば,  $\Box (p \rightarrow \Diamond r)$  は論理式である.

■

時相論理式の直感的意味としては以下のとおりである:  $\Box p$  は常に  $p$  が成り立つことを意味する.  $\Box (p \rightarrow (qWr))$  は常に  $p$  ならば  $r$  が成り立つまで  $q$  が成り立つことを意味する.  $\Box (p \rightarrow \Diamond r)$  は常に  $p$  ならばいつかは  $r$  が成り立つことを意味する.

次に, 時相論理の意味を形式的に定義する.

### Definition 17 (時相論理の意味)

状態  $s$  と論理式  $p$  に対して,  $p$  が  $s$  上で成り立つことを  $s \models p$  と書く. 状態の無限列  $\sigma = s_0, s_1, s_2, \dots$  をモデルとする. モデル  $\sigma$  に対して, 位置  $j \geq 0$  で  $p$  が成り立つことを  $\sigma, j \models p$  と書く. モデル上における論理式の充足関係は, 以下のように定義できる:

1.  $\sigma, j \models p$  iff  $s_j \models p$ .
2.  $\sigma \models \Box p$  iff すべての  $j$  に対して  $s_j \models p$  である.
3.  $\sigma \models \Box (p \rightarrow (qWr))$  iff すべての  $i$  に対して  $s_i \models p$  ならば, すべての  $j \geq i$  に対して  $s_j \models r$ , またはある  $k \geq i$  に対して  $s_k \models r$  かつすべての  $j$  ( $i \leq j < k$ ) に対して  $s_j \models q$  である.
4.  $\sigma \models \Box (p \rightarrow \Diamond r)$  iff すべての  $i$  に対して  $s_i \models p$  ならば,  $j \geq i$  に対して  $s_j \models r$  である.

■

### 3.4.2 演繹的検証

安全性や活性の検証ルールは Kesten や Manna, Pnueli によって開発されている [51]. 文献 [51] では, 安全性として不変性と waiting-for 性質の検証ルールが提案されており, 本論文では紙面の都合上から, 不変性の検証例のみを以下に示す. また, 他の安全性や活性の検証についても不変性と同様に, 時間状態チャートをクロック遷移システムに変換して, クロック遷移システムから各検証ルールの表明式を作成して, 検証ルールの充足性を判定すれば検証できる. なお, 他の文献 [53] に, それらの具体例の一部を示す.

まず、不変性の検証ルールを定義して、次にその検証例を示す。

**Definition 18 (不変性の検証ルール)**

表明  $\varphi$  と  $p$  に対して、

1.  $\Theta \rightarrow \varphi$
2.  $\varphi \rightarrow p$
3.  $\rho_\tau \wedge \varphi \rightarrow \varphi' \ (\forall \tau \in \mathcal{T})$
4.  $\frac{\quad}{\square p}$
- 5.

これは不変性を証明するルールであり、ルールの各行は、以下を意味する。1行目は初期条件  $\Theta$  ならば表明  $\varphi$  であることを意味する。2行目は表明  $\varphi$  ならば表明  $p$  であることを意味する。3行目は、 $\forall \tau \in \mathcal{T}$  に対して、 $\varphi$  を保存することを意味する。4行目は前提と結論を分離するラインである。5行目は結論  $\square p$  であり、 $\square$  は常に成り立つことを意味する時相オペレータである。 ■

前例のエアコン制御システムにより、演繹的証明による不変性の検証の例を示す。

**Example 6 (演繹的証明による不変性の検証例)**

前例のエアコン制御システムにより、演繹的証明による不変性の検証の例を以下に示す。ここで、 $\Theta = (u_1 = cool \wedge u_2 = open \wedge message = \perp \wedge is\_room\_hot = \perp \wedge is\_door\_close = \perp \wedge ac = \perp \wedge VS\_assert = \perp \wedge t_1 = 0 \wedge t_2 = 0 \wedge T = 0)$  であり、 $\varphi = p = \neg(u_1 = hot \wedge u_2 = open \wedge message = open\_door \wedge is\_room\_hot \geq 1 \wedge is\_door\_close = 0 \wedge ac = 1 \wedge VS\_assert = on \wedge t_1 \geq 0 \wedge t_2 \geq 0 \wedge T \geq 0)$  とする。これは、部屋の温度が高くてドアが開いていると、常にエアコンが動作しないことを意味する。

まず、1行目は、 $(u_1 = cool \wedge u_2 = open \wedge message = \perp \wedge is\_room\_hot = \perp \wedge is\_door\_close = \perp \wedge ac = \perp \wedge VS\_assert = \perp \wedge t_1 = 0 \wedge t_2 = 0 \wedge T = 0) \rightarrow \neg(u_1 = hot \wedge u_2 = open \wedge message = open\_door \wedge is\_room\_hot \geq 1 \wedge is\_door\_close = 0 \wedge ac = 1 \wedge VS\_assert = on \wedge t_1 \geq 0 \wedge t_2 \geq 0 \wedge T \geq 0)$  が成り立つので、 $\Theta \rightarrow \varphi$  も成り立つ。

次に、2行目は、 $\varphi = p$  なので、 $\varphi \rightarrow p$  は成り立つ。

次に、3行目は、 $\forall \tau \in \mathcal{T}$  に対して、 $\rho_\tau \wedge \varphi \rightarrow \varphi'$  であり、以下に示す。

1.  $(u_1 = cool \wedge u_2 = open \wedge message = \perp \wedge is\_room\_hot = \perp \wedge is\_door\_close = \perp \wedge ac = \perp \wedge VS\_assert = \perp \wedge 0 \leq t_1 \leq 5 \wedge 0 \leq t_2 \leq 5 \wedge 0 \leq T \leq 5) \wedge (u_1' = hot \wedge u_2' = open \wedge message' = room\_is\_hot \wedge is\_room\_hot' = 1 \wedge is\_door\_close' = \perp \wedge ac' = \perp \wedge VS\_assert' = \perp \wedge t_1' = 0 \wedge t_2' = 0 \wedge 0 \leq T')$   $\wedge \neg(u_1 = hot \wedge u_2 = open \wedge message = open\_door \wedge is\_room\_hot \geq 1 \wedge is\_door\_close = 0 \wedge ac = 1 \wedge VS\_assert = on \wedge t_1 \geq 0 \wedge t_2 \geq 0 \wedge 0 \leq T) \rightarrow \neg(u_1' = hot \wedge u_2' = open \wedge message' = open\_door \wedge is\_room\_hot' \geq 1 \wedge is\_door\_close' = 0 \wedge ac' = 1 \wedge VS\_assert' = on \wedge t_1' \geq 0 \wedge t_2' \geq 0 \wedge T' \geq 0)$ .

これは,  $(u_1 = cool \wedge u_2 = open \wedge \dots) \wedge (u_1' = hot \wedge u_2' = open \wedge message' = room\_is\_hot \wedge \dots)$   
 $\wedge \neg(u_1 = hot \wedge u_2 = open \wedge \dots) \rightarrow \neg(u_1' = hot \wedge u_2' = open \wedge message' = open\_door \wedge \dots)$  な  
 ので成り立つ.

2.  $(u_1 = hot \wedge u_2 = open \wedge message = room\_is\_hot \wedge is\_room\_hot = 1 \wedge is\_door\_close = \perp \wedge ac = \perp$   
 $\wedge VS\_assert = \perp \wedge 0 \leq t_1 \leq 7 \wedge 0 \leq t_2 \leq 7 \wedge 0 \leq T \leq 7) \wedge (u_1' = vhot \wedge u_2' = open \wedge message' =$   
 $room\_is\_hot \wedge is\_room\_hot' = 2 \wedge is\_door\_close' = \perp \wedge ac' = \perp \wedge VS\_assert' = \perp \wedge t_1' =$   
 $0 \wedge t_2' = 0 \wedge T' \geq 0) \wedge \neg(u_1 = hot \wedge u_2 = open \wedge message = open\_door \wedge is\_room\_hot \geq$   
 $1 \wedge is\_door\_close = 0 \wedge ac = 1 \wedge VS\_assert = on \wedge t_1 \geq 0 \wedge t_2 \geq 0 \wedge T \geq 0) \rightarrow \neg(u_1' =$   
 $hot \wedge u_2' = open \wedge message' = open\_door \wedge is\_room\_hot' \geq 1 \wedge is\_door\_close' = 0 \wedge ac' =$   
 $1 \wedge VS\_assert' = on \wedge t_1' \geq 0 \wedge t_2' \geq 0 \wedge T' \geq 0).$

これは,  $(u_1 = hot \wedge u_2 = open \wedge message = room\_is\_hot \wedge \dots) \wedge (u_1' = vhot \wedge u_2' = open \wedge \dots)$   
 $\wedge \neg(u_1 = hot \wedge u_2 = open \wedge message = open\_door \wedge \dots) \rightarrow \neg(u_1' = hot \wedge u_2' = open \wedge \dots)$  な  
 ので成り立つ.

.....  
 .....

以下同様に,  $\forall \tau \in \mathcal{T}$  に対して,  $\rho_\tau \wedge \varphi \rightarrow \varphi'$  が成り立つことが示せる.

以上より,  $\Box p$  が成り立つ. ■

### 3.5 時間ステートチャートの詳細化検証

Kesten と Manna, Pnueli のクロック遷移システムに関する演繹的証明系 [51] では, 安全性や活性の公理系は存在するが, 詳細化の公理系は存在しない. 本論文では, 時間ステートチャートから構成したクロック遷移システムに関する詳細化検証の公理系を開発する.

### 3.6 詳細化検証の公理系

クロック遷移システムに関する詳細化検証の公理系では, 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されれば, 具体化仕様が抽象化仕様を詳細化していると考えられる. 機能の包含性は通常論理的な含意であり, タイミング制約の包含性はクロック変数が表現する時間領域が包含されていると考える. 例えば, 具体化仕様のクロック変数  $c^A$  のタイミング制約が  $l^A \leq c^A \leq u^A$  であり, 抽象化仕様のそれが  $l^C \leq c^C \leq u^C$  とすると,  $l^A \leq l^C \leq u^C \leq u^A$  であれば, タイミング制約が包含されるとする.

**Definition 19 (詳細化の検証ルール)**

$CTS^C = (V^C, \Theta^C, T^C)$  が  $CTS^A = (V^A, \Theta^A, T^A)$  を詳細化していることを検証するルールを以下に示す.

1.  $\Theta^C \rightarrow \Theta^A[\alpha]$
2.  $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha] \ (\forall \tau^C \in T^C \text{ に対して})$
3. -----
4.  $CTS^C \sqsubseteq CTS^A$

これは詳細化を証明するルールであり、ルールの各行は、以下を意味する。1行目は初期状態集合  $\Theta^C$  ならば  $\Theta^A[\alpha]$  であることを意味する。2行目は  $\rho_{\tau^C}$  の状態遷移が抽象的な状態遷移  $\rho_{\tau^A}[\alpha]$  により説明されることを意味する。3行目は前提と結論を分離するラインである。4行目は結論  $CTS^C \sqsubseteq CTS^A$  であり、具体化仕様が抽象化仕様を正しく詳細化していることを意味する。ここで、 $\alpha$  は  $V^A$  の変数を  $V^C$  の変数に置き換える写像または抽象化仕様の状態  $\Sigma^A$  を具体化仕様の状態  $\Sigma^C$  に置き換える写像を意味する。 ■

まず、具体化仕様の状態と抽象化仕様の状態が1対1の場合の詳細化検証の事例を説明して、次に、具体化仕様の状態と抽象化仕様の状態が多対1の場合の詳細化検証の事例を説明する。

**Example 7 (詳細化検証の例 (1対1の場合))**

図12のように、具体化仕様  $CTS^C = (V^C, \Theta^C, T^C)$  及び抽象化仕様  $CTS^A = (V^A, \Theta^A, T^A)$  が与えられたとする。ここで、 $\alpha$  を以下のように定義する：

$$x = \begin{cases} \text{if 具体化仕様の状態が } 3 \text{ then } y + 1 \\ \text{else } y - 1 \end{cases}$$

ただし、 $-$  は通常の  $-$  であるが、 $y = 0$  のときは  $y - 1 = 0 - 1 = 0$  とする。

以下では、具体化仕様が抽象化仕様を正しく詳細化していることを検証する。

まず、 $\Theta^C \rightarrow \Theta^A[\alpha]$  が成り立つことを示す。ここで、 $\Theta^C = (u_1 = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  及び  $\Theta^A[\alpha] = (u_1 = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  なので、 $\Theta^C \rightarrow \Theta^A[\alpha]$  は成り立つ。

次に、 $\forall \tau^C \in T^C$  に対して、 $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$  が成り立つことを示す。

1. 具体化仕様が状態1から状態2に遷移するとき

$(u_1 = 1 \wedge y = 0 \wedge 0 \leq c < 2 \wedge 0 \leq T < 2) \wedge (u_1' = 2 \wedge y' = y \wedge 0 \leq c' < 5 \wedge 0 \leq T' < 5) \rightarrow (u_1 = 1 \wedge y = 0 \wedge 0 \leq c < 5 \wedge 0 \leq T < 5) \wedge (u_1' = 2 \wedge y' = y \wedge 0 \leq c' < 10 \wedge 0 \leq T' < 10)$  である。具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されるので、上記論理式は成り立つ。

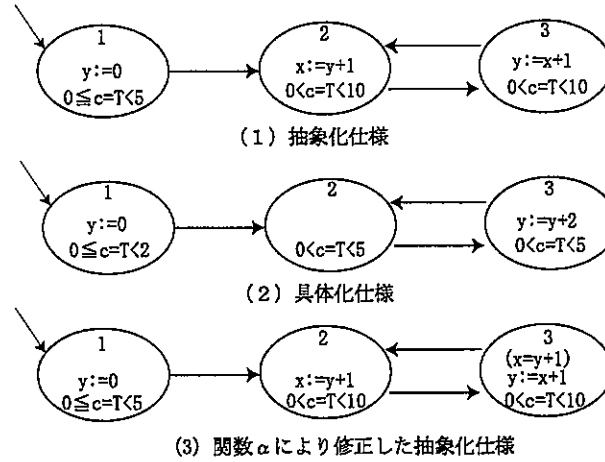


図 12: 詳細化検証の例 (1 対 1 の場合)

2. 具体化仕様が状態 2 から状態 3 に遷移するとき

$(u_1 = 2 \wedge y = y' \wedge 0 < c < 5 \wedge 0 < T < 5) \wedge (u_1' = 3 \wedge 0 < c' < 5 \wedge 0 < T' < 5) \rightarrow (u_1 = 2 \wedge y = y' \wedge 0 < c < 10 \wedge 0 < T < 10) \wedge (u_1' = 3 \wedge x = if \text{ 具体化仕様の状態が 3 then } y + 1 \text{ else } y - 1 = y' + 1 \wedge 0 < c' < 10 \wedge 0 < T' < 10)$  である。具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に含まれるので、上記論理式は成り立つ。

3. 具体化仕様が状態 3 から状態 2 に遷移するとき

$(u_1 = 3 \wedge 0 < c < 5 \wedge 0 < T < 5) \wedge (u_1' = 2 \wedge y' = y + 2 \wedge 0 < c' < 5 \wedge 0 < T' < 5) \rightarrow (u_1 = 3 \wedge 0 < c < 10 \wedge 0 < T < 10) \wedge (u_1' = 2 \wedge y' = x + 1 \wedge x = if \text{ 具体化仕様の状態が 3 then } y + 1 \text{ else } y - 1 = y + 1 \wedge 0 < c' < 10 \wedge 0 < T' < 10)$  である。具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に含まれるので、上記論理式は成り立つ。

上記 (2) と (3) を繰り返すので、 $\forall \tau^C \in T^C$  に対して、 $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$  が成り立つ。

以上より、具体化仕様が抽象化仕様を正しく詳細化していることが検証できた。

■

Example 8 (詳細化検証の例 (多対 1 の場合))

図 13 のように、具体化仕様  $CTS^C = (V^C, \Theta^C, T^C)$  及び抽象化仕様  $CTS^A = (V^A, \Theta^A, T^A)$  が与えられたとする。ここで、 $\alpha$  を以下のように定義する：

$$u_1 = \begin{cases} if \text{ 具体化仕様の状態} = 2 \text{ then } 3 \\ else \text{ 具体化仕様の状態} \end{cases}$$

ただし、 $u_1$  は抽象化仕様の状態である。 $\alpha$  は、具体化仕様の状態が 2 ならば抽象化仕様の状態が 3 であり、具体化仕様の状態が 2 でないならば抽象化仕様と具体化仕様の状態が等しいことを意味する。

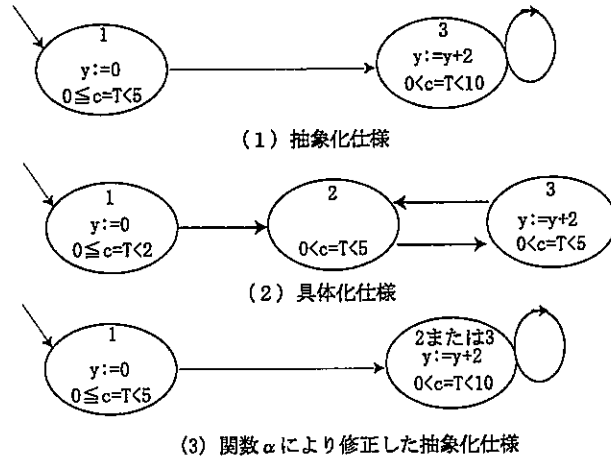


図 13: 詳細化検証の例 (多対 1 の場合)

以下では, 具体化仕様が抽象化仕様を正しく詳細化していることを検証する.

まず,  $\Theta^C \rightarrow \Theta^A[\alpha]$  が成り立つことを示す. ここで,  $\Theta^C = (u_1 = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  及び  $\Theta^A = (u_1 = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  である.  $\Theta^A$  の状態  $u_1 = 1$  は抽象化仕様の状態なので,  $\alpha$  により具体化仕様の状態に変換する. ところで,  $\alpha$  の定義により, 具体化仕様の状態が 2 でなければ抽象化仕様と具体化仕様の状態名は等しい. ゆえに,  $\Theta^A[\alpha] = (u_1 = \text{if 具体化仕様の状態} = 2 \text{ then } 3 \text{ else 具体化仕様の状態} = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  なので,  $\Theta^C \rightarrow \Theta^A[\alpha]$  は成り立つ.

次に,  $\forall \tau^C \in T^C$  に対して,  $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$  が成り立つことを示す.

1. 具体化仕様が状態 1 から状態 2 に遷移するとき

$(u_1 = 1 \wedge y = 0 \wedge 0 \leq c < 2 \wedge 0 \leq T < 2) \wedge (u_1' = 2 \wedge y = y' \wedge 0 \leq c' < 5 \wedge 0 \leq T' < 5) \rightarrow (u_1 = \text{if 具体化仕様の状態} = 2 \text{ then } 3 \text{ else 具体化仕様の状態} = 1 \wedge y = 0 \wedge 0 \leq c < 5 \wedge 0 \leq T < 5) \wedge (u_1' = \text{if 具体化仕様の状態} = 2 \text{ then } 3 \text{ else 具体化仕様の状態} = 3 \wedge y' = y \wedge 0 \leq c' < 10 \wedge 0 \leq T' < 10)$  である. 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されるので, 上記論理式は成り立つ.

2. 具体化仕様が状態 2 から状態 3 に遷移するとき

$(u_1 = 2 \wedge 0 < c < 5 \wedge 0 < T < 5) \wedge (u_1' = 3 \wedge y' = y \wedge 0 < c' < 5 \wedge 0 < T' < 5) \rightarrow (u_1 = \text{if 具体化仕様の状態} = 2 \text{ then } 3 \text{ else 具体化仕様の状態} = 3 \wedge 0 < c < 10 \wedge 0 < T < 10) \wedge (u_1' = \text{if 具体化仕様の状態} = 2 \text{ then } 3 \text{ else 具体化仕様の状態} = 3 \wedge y' = y \wedge 0 < c' < 10 \wedge 0 < T' < 10 \wedge u_1' = u_1)$  である. 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されるので, 上記論理式は成り立つ.

3. 具体化仕様が状態 3 から状態 2 に遷移するとき

$(u_1 = 3 \wedge y = 0 \wedge 0 < c < 5 \wedge 0 < T < 5) \wedge (u_1' = 2 \wedge y' = y + 2 \wedge 0 < c' < 5 \wedge 0 < T' < 5) \rightarrow$

$(u_1 = \text{if 具体化仕様の状態} = 2 \text{ then } 3 \text{ else 具体化仕様の状態} = 3 \wedge y = 0 \wedge 0 < c < 10 \wedge 0 < T < 10) \wedge (u_1' = \text{if 具体化仕様の状態} = 2 \text{ then } 3 \text{ else 具体化仕様の状態} = 3 \wedge y' = y + 2 \wedge 0 < c' < 10 \wedge 0 < T' < 10 \wedge u_1' = u_1)$  である. 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されるので, 上記論理式は成り立つ.

上記 (2) と (3) を繰り返すので,  $\forall T^C \in T^C$  に対して,  $\rho_{T^C} \rightarrow \bigvee_{T^A \in T^A} \rho_{T^A}[\alpha]$  が成り立つ. ゆえに, 具体化仕様が抽象化仕様を正しく詳細化していることが検証できた. ■

### 3.6.1 詳細化検証の事例

エアコン制御システムにより, 具体化仕様  $CTS^C = (V^C, \Theta^C, T^C)$  が抽象化仕様  $CTS^A = (V^A, \Theta^A, T^A)$  を詳細化していることの検証例を示す.

では, 以下に, エアコン制御システムの詳細化検証を示す. ここでは, 図 10 を具体化仕様, 図 14 を抽象化仕様とする. 図 10 と図 14 では, 状態図 *DOOR* は等しく, 状態図 *TEMP* のみが詳細化されたとする.

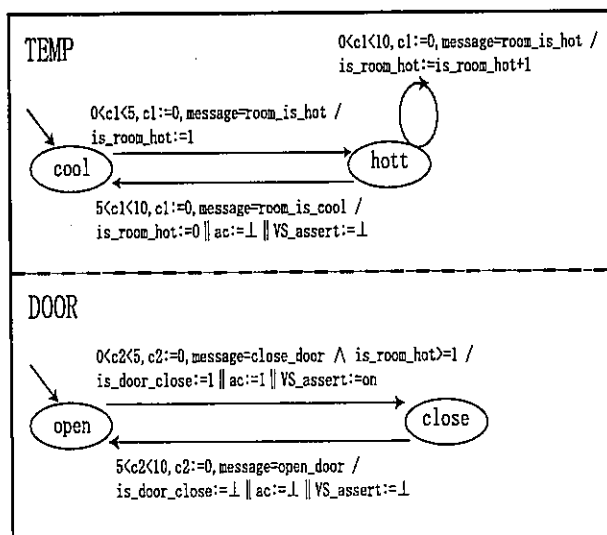


図 14: エアコン制御システムの抽象化仕様

以下では, 図 14 の状態図 *TEMP* が図 10 の状態図 *TEMP* によって正しく詳細化されていることのみを検証する.

ここで,  $\alpha$  を以下のように定義する:



$is\_room\_hot' = 3 \wedge is\_door\_close' = \perp \wedge ac' = \perp \wedge VS\_assert' = \perp \wedge c_1' = 0 \wedge 0 \leq T'$ )  $\rightarrow$   
 $(u_1 = if \text{ 具体化仕様の状態が } hot \text{ 又は } vhot \text{ then } hott \text{ else 具体化仕様の状態} = hott \wedge message =$   
 $room\_is\_hot \wedge is\_room\_hot = 2 \wedge is\_door\_close = \perp \wedge ac = \perp \wedge VS\_assert = \perp \wedge 0 \leq c_1 \leq$   
 $10 \wedge 0 \leq T \leq 10) \wedge (u_1' = if \text{ 具体化仕様の状態が } hot \text{ 又は } vhot \text{ then } hott \text{ else 具体化仕様の状態}$   
 $= hott \wedge u_2' = open \wedge message' = room\_is\_hot \wedge is\_room\_hot' = 3 \wedge is\_door\_close' = \perp$   
 $\wedge ac' = \perp \wedge VS\_assert' = \perp \wedge c_1' = 0 \wedge 0 \leq T')$  である. 具体化仕様の機能とタイミング制約が抽象化仕様のそれに包含されるので, 上記論理式は成り立つ.

.....  
 .....

同様な方法で, すべての状態遷移について, 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されることが示せる.

以上より,  $\forall \tau^C \in T^C$  に対して,  $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$  が成り立つ.

ゆえに, 具体化仕様が抽象化仕様を正しく詳細化していることが検証できた.

### 3.7 むすび

本節では, まず, 時間状態チャート [31] を一般化して, その操作的意味をクロック遷移システム上で定義した. これにより, ハードリアルタイムシステムの機能や並行性, タイミング制約が統合的に仕様記述できた. 次に, 安全性や活性を時相論理式で仕様記述して, クロック遷移システムが時相論理式を充足するかどうかを, 演繹的証明で検証する手法を示した. 次に, 詳細化の検証の公理系を開発して, 時間状態チャートの詳細化を検証した. 本論文のように, 機能モデルと動作モデルを統合的に仕様記述すると, データ領域や実行の無限性により, 仕様は無限な状態空間を有する時間付きの状態遷移システムになる. ゆえに, モデル検査 [41] は不可能であり, 検証作業は演繹的証明に頼らざるをえない. 演繹的証明は数学的訓練が必要な作業であり, 大規模システムへの適用は困難である. しかし, プロセスの相互排他制御の検証などのように検証対象を限定すれば, 小規模システムとして仕様記述できて, 演繹的検証が実現できて実用性は高いと考えられる. 以上より, 今後の研究課題としては, 以下が重要である.

1. 表明の自動生成や証明の再利用, GUIの向上などにより, 演繹的証明作業の効率的な支援を実現する.
2. 抽象実行などの抽象化技術を適用することにより, 時間状態チャートの自動検証を実現する.
3. 時間状態チャートのモジュールを定義して, その意味を形式的に定義することにより, モジュール検証を実現する.

## 4 実時間型開放分散システムの Assume-guarantee 方式による演繹的設計支援

### 4.1 まえがき

通信システムや制御システムの多くは実時間ソフトウェアである [30]. さらに, これらは他のシステムと協調して動作するリアクティブソフトウェアであることもしばしばである. したがって, これらのソフトウェアの設計においては, 各ソフトウェアにおける処理時間が他のソフトウェアに及ぼす影響や, ソフトウェア間の通信遅延といった実時間性をも考慮する必要がある. このことにより, (リアクティブ) 実時間ソフトウェアの設計作業は複雑になりがちであり, また, ソフトウェアの信頼性保証も一般的には困難である. したがって, 実時間ソフトウェアの信頼性保証ができることは重要である. 本節では, 実時間ソフトウェアの仕様やプログラムの一般的なモデルである, Pnueli や Manna らのクロック遷移モジュール上において, Assume-Guarantee 形式による演繹的詳細化検証手法を提案する. なお, 既に, 我々は実時間ソフトウェアの演繹的詳細化検証手法を開発しているが [42], この手法では, 検証時に状態空間組み合わせ爆発を発生させる. 本節では, 状態空間の組み合わせ爆発を抑制するために, Assume-Guarantee 形式による演繹的詳細化検証手法を提案する.

Assume-guarantee 形式 [12] は並列動作するシステムの効果的な検証方法論であり, 時相論理検証や詳細化検証に適用されている. 並列動作するシステムの検証では, 並列動作するプロセスの状態空間のカルテジアン積を検証の対象とする必要があり, 状態空間組み合わせ爆発を発生させる. Assume-Guarantee 形式の検証では, 環境の動作を仮定して構成的に検証するものであり, 状態の組み合わせ爆発が抑制できる有効な方法論である.

従来の Assume-Guarantee 形式の検証の研究では, 以下のようなものがある:

1. 1981年に, 初めて, Misra らがネットワークのプロセスの不変性の検証 [43] に, Assume-Guarantee 形式の検証を適用した. Misra らの研究は CSP のトレース理論を基礎として Assume-Guarantee 形式の検証ルールを開発したものである.
2. 1983年に, Lamport [44] が並行プログラムの安全性や活性の検証に, Assume-Guarantee 形式の検証を適用した. Lamport は安全性や活性を時相論理式で表現して, Assume-Guarantee 形式の時相論理の検証ルールを開発した. この Lamport の研究が Assume-Guarantee 形式の時相論理検証の最初のものである. その後の 1985年に, Stark が同様な Assume-Guarantee 形式の時相論理検証手法 [45] を開発した. また, 1985年に, Pnueli が Assume-Guarantee 形式の時相論理検証手法のまとめとなる論文 [12] を発表して, 検証のスタンダードなテクニックを示した. その後, 1994年に, Chang らが Assume-Guarantee 形式の時相論理検証を実時間システムに拡張した [46].
3. 一方, Assume-Guarantee 形式の詳細化検証は以下のとおりである. 1996年に, 初めて, Alur と Henzinger が reactive module の Assume-Guarantee 形式の詳細化検証手法を提案した. Alur らの研究はリアクティブシステムの一般的なモデルの上で言語包含関係の詳細化検証を提案したも

のである。その後、1997年に、Alurらは、時間オートマトン [48] を一般化した timed module 上で言語包含関係の詳細化検証を提案した。その後、我々は時間オートマトン上で時間模関係 [11] の詳細化検証 [49] へと拡張した。最近、Henzingerらがハイブリッドシステムへ Assume-Guarantee 形式の自動詳細化検証手法を適用しており [50]、今後の発展が期待される有望な研究である。

本節では、上記の既存研究と異なり、実時間ソフトウェアの仕様やプログラムの一般的なモデルである、Pnueliらのクロック遷移モジュール (clocked transition modules) [51, 52] 上において、Assume-Guarantee 形式による演繹の詳細化検証手法を提案する。クロック遷移モジュール上では、安全性や活性などの時相論理検証の公理系は開発されている [51, 52] が、詳細化検証の公理系は開発されていない。

本節では、まず、実時間ソフトウェアの仕様やプログラムの一般的なモデルである、Pnueliらのクロック遷移モジュールを定義する。次に、クロック遷移モジュールの Assume-Guarantee 形式による演繹の詳細化検証手法を提案する。最後に、事例により、提案手法の有効性を示す。

本節の構成は、次のとおりである。4.2章では、クロック遷移モジュールを定義する。4.3章では、クロック遷移モジュールの Assume-Guarantee 形式による演繹の詳細化検証手法を提案する。4.4章では、Assume-Guarantee 形式による演繹の詳細化検証事例を述べる。最後に、まとめと今後の課題を述べる。

## 4.2 クロック遷移モジュールとその実時間ソフトウェア開発への適用

本章では、クロック遷移モジュールを定義して、クロック遷移モジュールが実時間ソフトウェアの形式的モデルであることを説明する。

### 4.2.1 クロック遷移モジュール

実時間ソフトウェアのモジュールの形式的計算モデルとしてクロック遷移モジュール [51, 52] を定義する。本節では、多数のモジュールが並列動作して、実時間ソフトウェアが構成されるとする。

まず、システム変数の有限集合を考える。システム変数は整数や実数などの型を持つ。我々はシステム変数にその型の値を割り付ける解釈として状態  $s$  を定義する。すべての状態の集合を  $\Sigma$  とする。

#### Definition 20 (クロック遷移モジュール)

クロック遷移モジュールは、 $\text{CTM} = (V, \Theta, T, \Pi)$  の4つ組で定義される。ここで、

1.  $V$  はシステム変数の有限集合である。集合  $V = D \cup C$  は、離散変数の集合  $D = \{u_1, \dots, u_n\}$  とクロック変数の集合  $C = \{t_1, \dots, t_k\}$  に分類できる。離散変数は任意の型がありえるが、クロック変数は実数型である。さらに、リセットされないマスタクロック  $T \in C$  を導入する。また、集合  $V$  は共有変数 ( $V^S$ ) とローカル変数 ( $V^P$ ) に分類できて、 $V = V^P \cup V^S$  である。ローカル変数

$(V^P)$ はモジュールのみが参照更新できる変数であり、共有変数  $(V^S)$ はモジュールの外部からも参照更新ができる変数である。ただし、マスタクロック  $T \in C$ は共有変数である。

並列モジュールが通信する手段として、共有変数を使うので、共有変数は必要である。つまり、CTMは共有変数を使用する並列計算モデルである。

2.  $\Theta$ は初期条件である。これは、すべての初期状態を特徴付ける表明である。 $\Theta \rightarrow t_1 = \dots = t_k = T = 0$ が要求される。
3.  $\Pi$ は時間前進条件である。これは時間前進の制限を表現するために使われる。
4.  $T$ は状態遷移の有限集合である。各状態遷移  $\tau \in T$ は関数  $\tau: \Sigma \rightarrow 2^\Sigma$ であり、各状態  $s \in \Sigma$ に次状態  $\tau$ -successor  $\tau(s) \subseteq \Sigma$ を写像する。状態遷移  $\tau$ に関連する関数は第一階述語論理式の表明  $\rho_\tau(V, V')$ により表現される。 $\rho_\tau(V, V')$ は状態遷移関係と呼び、状態  $s \in \Sigma$ を  $\tau$ -successor  $s' \in \tau(s)$ に関係付ける。ただし、状態  $s$ は  $V$ の型整合解釈であり、各変数  $v \in V$ に値  $s[v]$ を割り付ける。なお、システム変数の値は  $s$ の中の値や  $s'$ の中の値として参照する。任意の  $\tau \in T$ に対して、 $\rho_\tau \rightarrow T' = T$ が要求される。状態  $s \in \Sigma$ は変数  $V$ の型整合解釈であり、変数  $V$ の中には実数があるので、状態  $s \in \Sigma$ は非可算無限個存在する。下記で定義するような時間経過の状態遷移  $\rho_{tick}$ を導入すると、状態  $s \in \Sigma$ は可算無限個になる。

■

また、クロック遷移モジュール  $CTM = (V, \Theta, T, \Pi)$ の状態遷移  $T$ は、状態における時間の経過  $tick$ により、 $T_T = T \cup \{tick\}$ と拡張される。ここで、 $tick$ 状態遷移を表現する状態遷移関係  $\rho_{tick}$ は以下のとおりである：

$$\rho_{tick} : \exists \Delta. \left( \begin{array}{c} \Delta > 0 \wedge \forall t \in [0, \Delta]. \Pi(D, C + t) \\ \wedge \\ D' = D \wedge C' = C + \Delta \end{array} \right)$$

ここで、 $C' = C + \Delta$ は  $\bigwedge_{c \in C} (c' = c + \Delta)$ を意味する。 $tick$ は離散変数の値を保存して、 $\Pi$ を満足しながら  $\Delta$ により、すべてのクロック変数を同じスピードで増加させる。

ここで、拡張されたクロック遷移モジュール  $CTM = (V, \Theta, T \cup \{tick\}, \Pi)$ の動作列は、以下の条件を満たす無限の状態列  $\sigma: s_0, s_1, \dots$ である：

1.  $s_0 \models \Theta$ である。
2. 任意の  $j \geq 0$ に対して、 $s_{j+1} \in \tau(s_j)$ であるような  $\tau \in T \cup \{tick\}$ が存在する。

また、クロック遷移モジュール  $CTM = (V, \Theta, T, \Pi)$ は、モジュールの外部の環境の状態遷移  $\tau_E$ を追加することにより、閉じたシステム  $S = (V^*, \Theta, T \cup \{\tau_E\}, \Pi)$ と見なせる。ここで、 $V^*$ はモジュールと環境の変数である。また、 $\tau_E$ を表現する状態遷移関係  $\rho_{\tau_E}$ は以下のように定義される：

$$\rho_{\tau_E} : \left( \bigwedge_{u \in V^P} (u = u') \right) \wedge (T = T')$$

さらに、クロック遷移モジュールの並列合成は以下のように定義される。

**Definition 21 (クロック遷移モジュールの並列合成)**

2つのクロック遷移モジュール  $CTM_1 = (V_1, \Theta_1, \mathcal{T}_1, \Pi_1)$  と  $CTM_2 = (V_2, \Theta_2, \mathcal{T}_2, \Pi_2)$  が与えられたとき、並列合成  $CTM_1 \parallel CTM_2$  はクロック遷移モジュール  $CTM = (V, \Theta, \mathcal{T}, \Pi)$  として以下のように定義される。ここで、

1.  $V = V_1 \cup V_2, D = D_1 \cup D_2, C = C_1 \cup C_2, V^P = V_1^P \cup V_2^P, V^S = V_1^S \cup V_2^S$
2.  $\Theta = \Theta_1 \wedge \Theta_2$
3.  $\mathcal{T}$  の元は  $\mathcal{T}_1$  と  $\mathcal{T}_2$  の元から、以下のように構成されるもの全体である。
  - (a)  $\mathcal{T}_1$  中の遷移  $\tau$  に対して、その変数が  $\mathcal{T}_2$  に現れないとき

$$\rho_\tau \wedge \left( \bigwedge_{v \in V_2^P} (v = v) \right)$$

また、 $\mathcal{T}_2$  中の遷移  $\tau$  に対して、その変数が  $\mathcal{T}_1$  に現れないとき

$$\rho_\tau \wedge \left( \bigwedge_{v \in V_1^P} (v = v) \right)$$

- (b)  $\tau_1 \in \mathcal{T}_1$  と  $\tau_2 \in \mathcal{T}_2$  が同じ変数を持つとき

$$\rho_{\tau_1} \wedge \rho_{\tau_2}$$

4.  $\Pi = \Pi_1 \wedge \Pi_2$

■

**Example 9 (クロック遷移モジュールの例)**

図 15 のように、クロック遷移モジュール  $CTM = (V, \Theta, \mathcal{T}, \Pi)$  が与えられたとする。

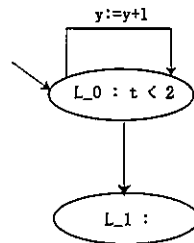


図 15: クロック遷移モジュールの例

ここで、各項は以下のとおりである：

1.  $V = \{\pi, y, t, T\}$  であり,  $D = \{\pi, y\}$  と  $C = \{t, T\}$  である. また,  $V^P = \{\pi, y, t\}$  と  $V^S = \{T\}$  である.
2.  $\Theta : \pi = L_0 \wedge y = t = T = 0$ .
3.  $T = \{\tau_0, \tau_1\}$ . ここで,  $\tau_0$  は  $L_0$  の自己ループであり,  $\tau_1$  は  $L_0$  から  $L_1$  への状態遷移である :
  - (a)  $\tau_0 : \pi = \pi' = L_0 \wedge y = y' + 1 \wedge t = t' \wedge T = T'$
  - (b)  $\tau_1 : \pi = L_0 \wedge \pi' = L_1 \wedge y = y' \wedge t = t' \wedge T = T'$
4.  $\Pi : \pi = L_0 \rightarrow t < 2$ .

ここで, *tick* 状態遷移を表現する状態遷移関係  $\rho_{tick}$  は以下のとおりである :

$$\rho_{tick} : \exists \Delta. \left( \begin{array}{c} \Delta > 0 \wedge \forall t \in [0, \Delta] \wedge (\pi = L_0 \rightarrow t + \Delta \leq 2) \\ \wedge \\ (\pi = \pi' \wedge y = y' \wedge t' = t + \Delta \wedge T' = T + \Delta) \end{array} \right)$$

クロック遷移モジュールの動作列は以下のとおりである :

$$\begin{aligned} & \langle \pi = L_0 \wedge y = 0 \wedge t = 0 \wedge T = 0 \rangle \xrightarrow{tick} \\ & \langle \pi = L_0 \wedge y = 0 \wedge t = 1 \wedge T = 1 \rangle \xrightarrow{\tau_0} \\ & \langle \pi = L_0 \wedge y = 1 \wedge t = 1 \wedge T = 1 \rangle \xrightarrow{\tau_0} \\ & \langle \pi = L_0 \wedge y = 2 \wedge t = 1 \wedge T = 1 \rangle \xrightarrow{tick} \\ & \langle \pi = L_0 \wedge y = 2 \wedge t = 2 \wedge T = 2 \rangle \xrightarrow{\tau_0} \\ & \langle \pi = L_0 \wedge y = 3 \wedge t = 2 \wedge T = 2 \rangle \xrightarrow{\tau_0} \\ & \langle \pi = L_0 \wedge y = 4 \wedge t = 2 \wedge T = 2 \rangle \xrightarrow{\tau_1} \\ & \langle \pi = L_1 \wedge y = 4 \wedge t = 2 \wedge T = 2 \rangle \xrightarrow{tick} \\ & \dots \\ & \dots \end{aligned}$$

■

#### 4.2.2 クロック遷移モジュールの実時間ソフトウェア開発への適用

実時間ソフトウェア開発のために作成される時間状態チャートやリアルタイム構造化仕様書, Z仕様書などの意味はクロック遷移モジュールに変換できる (例えば, 文献 [53] など). すなわち, 図 16 のように, クロック遷移モジュールは実時間ソフトウェアの一般的な形式的モデルと見なせる.

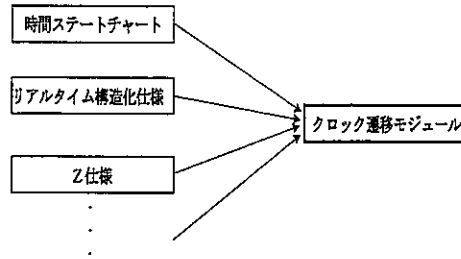


図 16: 一般的な計算モデルとしてのクロック遷移モジュール

### 4.3 Assume-Guarantee 形式による演繹的詳細化検証

本節では, Pnueli と Manna らのクロック遷移モジュール [51, 52] に関する Assume-Guarantee 形式による演繹的詳細化検証の公理系を開発する. 最初に receptiveness [52] の演繹的検証を定義して, 次に Assume-Guarantee 形式の演繹的詳細化検証の公理系を提案する. なお, receptiveness はモジュールが物理的に実装可能な条件である. Assume-Guarantee 形式検証は対象モジュールと環境との並列合成により, システム全体を検証する手法である. なお, 対象モジュールと環境との並列合成で構成されるシステムが物理的に実装可能であることを保証する必要がある. このために, すべてのモジュールが receptive であることを保証する必要がある.

#### 4.3.1 Receptiveness の演繹的検証

実時間ソフトウェアのモジュールの動作の正当性を保証する概念として, 並列演算閉包性が存在し, Nonzeno を保証する receptiveness [14] を使用する [52]. モジュールは, 以下のいずれかの場合に receptive であると言う.

1. モジュールの経過時間が発散する. つまり, モジュールの状態遷移が無限回起きると, モジュールの経過時間も無限に大きくなる.
2. モジュールの状態遷移が有限回起きる.

#### Definition 22 (Receptiveness の定義)

*receptiveness* はモジュールと環境 (対象としているモジュール以外のモジュール) との無限ゲームの下で, モジュールに対して定義される. ここで, モジュール及び環境は以下のように動作する: モジュールが状態遷移  $\tau_1$  を行うか, 時間経過  $\Delta_{mod}$  を行う. 環境が状態遷移  $\tau_E$  や *external* 遷移  $\tau_2$  を行うか, 時間経過  $\Delta_{env}$  を行う. ただし, モジュールの状態遷移  $\tau_1$  は状態遷移関係  $\rho_{\tau_1}$  と記述して, 時間経過  $\Delta_{mod}$  は状態遷移関係  $\rho_{tick}$  と記述する. また, 環境の状態遷移  $\tau_E$  は状態遷移関係  $\rho_{\tau_E}$  と記述して,

external遷移  $\tau_2$  は状態遷移関係  $\rho_{\tau_2}$  と記述して、時間経過  $\Delta_{env}$  は状態遷移関係  $\rho_{tick}$  と記述する。なお、external遷移  $\tau_2$  とは、環境により制御されるモジュール内の遷移である。ただし、モジュールの状態遷移  $\tau$  は、モジュールが提案する状態遷移  $\tau_1$  と external遷移  $\tau_2$  の和集合である (つまり、 $\tau = \tau_1 \cup \tau_2$ )。また、状態遷移  $\tau_1$  は *controlled* 遷移と呼ばれることがある。無限ゲームにおいては、以下のように、次の動作が決定される：

1. もし環境が状態遷移  $\tau_E$  や external遷移  $\tau_2$  を提案するならば、環境の状態遷移  $\tau_E$  や external遷移  $\tau_2$  が選択される。これは環境の動作である。
2. もし環境が時間経過  $\Delta_{env}$  を提案してモジュールが状態遷移  $\tau_1$  を提案するならば、モジュールの状態遷移  $\tau_1$  が選択される。これはモジュールの動作である。
3. もし環境とモジュールがともに時間経過  $\Delta_{mod}, \Delta_{env}$  を提案するならば、小さい時間経過  $\min\{\Delta_{mod}, \Delta_{env}\}$  が選択される。これは、 $\Delta_{mod} \leq \Delta_{env}$  ならばモジュールの動作であり、そうでなければ環境の動作である。

その様子を図 17 に示す。簡単に言えば、上記無限ゲームに従って動作するモジュールの経過時間が発散するか、またはモジュールの状態遷移  $\tau_1$  が有限回ならば、モジュールは勝つ。形式的には、任意の環境に対して、モジュールが勝つ戦略を持つならば、そのモジュールは *receptive* である。なお、モジュールの戦略とは、モジュールの任意の到達可能な状態に対して、*controlled* 遷移か  $\Delta_{mod}$  遷移を割り付ける関数である。 ■

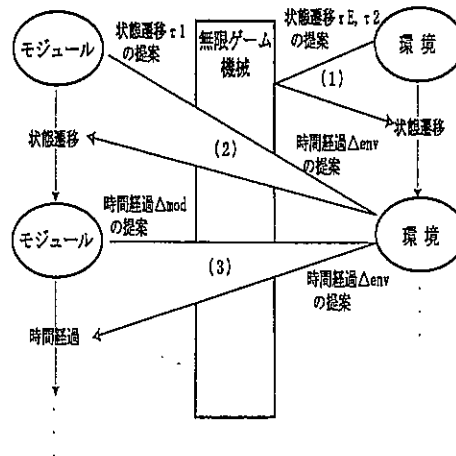


図 17: 無限ゲームの概念図

クロック遷移モジュールが *receptive* であることを示すために、我々は Manna らの Receptiveness ダイアグラム [52] を使用する。Receptiveness ダイアグラムは Receptiveness を検証するための証明図

であり, Receptiveness ダイアグラムにおいてモジュールが勝つ戦略を持つならば, そのクロック遷移モジュールは Receptiveness を満たすとする.

### Definition 23 (Receptiveness ダイアグラム)

クロック遷移モジュール CTM の Receptiveness ダイアグラムは有向グラフ  $D$  であり, 以下の条件 (1), (2), (3) を満たすものである. ここで, 任意のノード  $n$  は表明  $\varphi_n$  とランキング関数  $\delta_n$  によりラベル付けされている. なお, ランキング関数  $\delta_n$  はクロック遷移モジュール CTM のローカル変数から整礎順序集合への写像である. また,  $\{\varphi_n\} \text{ tick } \exists\{T \geq t_0 + \epsilon\}$  は,

$$\{\varphi_n(V)\} \rightarrow \exists V I . \rho_{\text{tick}}(V, V I) \wedge \{T I \geq t_0 + \epsilon\}$$

である. ただし,  $\varphi_n(V)$  はノード  $n$  の表明,  $\rho_{\text{tick}}(V, V I)$  は  $\text{tick}$  状態遷移を表す表明である. Receptiveness ダイアグラムは以下から構成される:

#### 1. 初期条件:

$(T = t_0) \rightarrow \bigvee_{n \in \text{Nodes}(D)} \varphi_n$ . ここで,  $t_0$  はクロック定数であり,  $\text{Nodes}(D)$  は有向グラフ  $D$  のノードの集合である.

#### 2. モジュールの動作:

有向グラフ  $D$  の任意のノードに対して, ( )  $\text{tick}$  遷移 (モジュールの時間経過) 又はランキング関数  $\delta_n$  を減少させる状態遷移  $\tau_1$  (モジュールが提案する状態遷移), または, ( )  $t_0 + \epsilon$  を超えて時間経過する  $\text{tick}$  遷移 (モジュールの時間経過) がラベル付けられている. 形式的には以下のように表現できる:

$$\begin{aligned} & \{\varphi_n\} \text{ tick } \exists\{T \geq t_0 + \epsilon\} \\ & \vee \\ & \bigvee_{e_m = \langle n, m \rangle \in \text{COout}(n)} \{\varphi_n \wedge \delta_n = u\} \tau(e_m) \exists\{\varphi_m \wedge \delta_m < u\} \end{aligned}$$

ここで,  $\epsilon$  は 0 より大きな定数であり,  $\text{COout}(n)$  は  $\text{tick}$  遷移 (モジュールの時間経過) か状態遷移  $\tau_1$  (モジュールが提案する状態遷移) によりラベル付けされる, ノード  $n$  から出るノードの集合である.

#### 3. 環境の動作:

すべてのノード  $n$  と  $\text{tick}$ , 環境か  $\text{external}$  遷移の  $\tau_I$  に対して,  $\tau_I$  により  $\varphi_n$  が保存される, または,  $\tau_I$  が  $\tau_I(n)$  の一つに遷移する. 形式的には以下のように表現できる:

$$\{\varphi_n \wedge \delta_n = u\} \tau_I \{ (\varphi_n \wedge \delta_n \leq u) \vee \left( \bigvee_{m \in N(\tau_I, n)} (\varphi_m \wedge \delta_m \leq u) \right) \}$$

ここで,  $N(\tau_I, n)$  は, 状態遷移  $\tau_I$  により  $n$  から  $m$  に遷移するようなノード  $m$  の集合である.

■

以下に, Receptiveness の証明例を示す.

**Example 10 (Receptiveness の証明例)**

Receptiveness の証明を簡単な例を用いて説明する.

図 18 は, 機能として電球を点滅させるだけのシステムの仕様である. 図 18 は文献 [52, 54] の記法に従う. なぜならば, クロック遷移モジュールは計算モデルであり, 仕様記述言語としては Manna らの表記法が適しているからである. その表記法は時間オートマトン [48, 41] と同様に, 状態及び状態遷移から構成されており, 状態には時間前進条件が付くことがあり, 状態遷移にはタイミング制約式, リセット式及びイベントが付くことがある. 図 18 の仕様をクロック遷移モジュールに対応づけながら, 以下に, 図 18 の仕様を説明する.

このシステムはモジュール *contoroller* とモジュール *lamp* の並列合成で構成される.

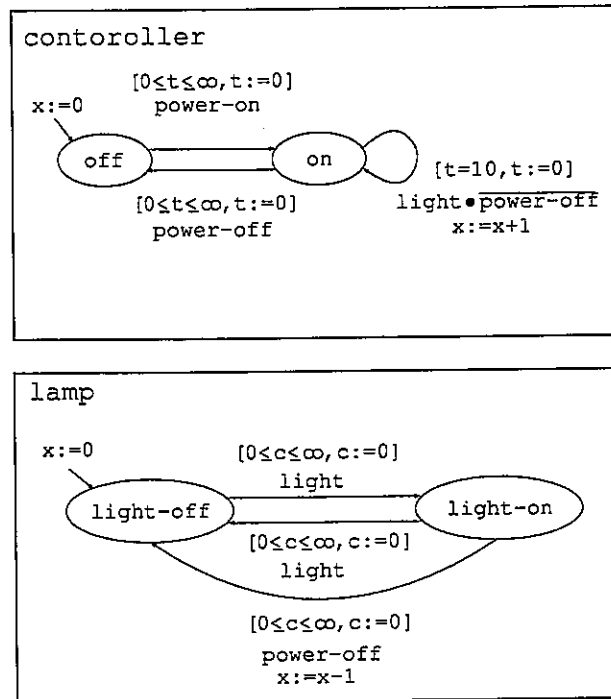


図 18: 電球システム

この仕様を以下に説明する.

1. 丸は状態を示し, 丸の中の文字列は状態名である.
2. 矢印は状態遷移を示し, `[ ]`で囲まれた部分はタイミング制約とクロック変数のリセットを示す. `[ ]`に囲まれていない文字列は遷移の引き金となるイベントであり, 状態遷移名となる. なお, `e.l`

は、イベント  $l$  が発生しない場合のみ、イベント  $e$  は発生できることを意味する。また、状態遷移に付いている  $x := x + 1$  は共有変数  $x$  を  $+1$  することを意味する。

3. システムの動作を制御するモジュール *controller* について説明する。

- (a) 状態名 *off* は電源が切れている状態名、状態名 *on* は電源が入っている状態名を示す。
- (b) イベント *power-on* は人間が電源を入れたことを示し、このモジュールの環境によって発生するイベントであるので、このイベントによる遷移は環境により制御されるモジュール内の遷移である *external* 遷移である。よって、この遷移は環境の動作となる。
- (c) イベント *power-off* は人間が電源を切ったことを示し、このイベントによる遷移は *power-on* による遷移と同様に *external* 遷移であり、この遷移は環境の動作となる。
- (d) イベント *light* は *controller* が電球に対して 10 秒ごとに光れや消えろといった信号を送ることを示し、このモジュール自身が発生させる内部イベントであるので、このイベントによる遷移はモジュール自身が制御している遷移である *controlled* 遷移である。この遷移はモジュールの動作となる。

4. 電球であるモジュール *lamp* について説明する。

- (a) 状態名 *light-off* は電球が消えている状態名、状態名 *light-on* は電球がついている状態名を示す。
- (b) このモジュール内の遷移は、同じイベントをもつ *controller* 内の遷移と同期する。また、すべての遷移は *contoroller* と人間によって制御され、自分自身で制御できない。よって、このモジュール内のすべての遷移は *external* 遷移であり、環境の動作となる。

この仕様をクロック遷移モジュールに対応づけて説明するために、モジュール *controller* をクロック遷移モジュールに変換する。ここで、このモジュールの状態名を示すローカル変数を  $u$  とする。

$$1. V = \{u, x, t, T\}.$$

$$(a) V^P = \{u, t\}.$$

$$(b) V^S = \{T, x\}.$$

$$2. \Theta : u = \text{off} \wedge x = 0 \wedge t = 0 \wedge T = 0.$$

$$3. \Pi : \text{なし}.$$

4. 各  $\tau \in T$  に対する状態遷移関係  $\rho_\tau$  は以下のように定義される:

$$(a) \rho_{\text{power-on}} : u = \text{off} \wedge u' = \text{on} \wedge x' = x \wedge 0 \leq t \wedge t' = 0 \wedge 0 \leq T \wedge T' = T.$$

$$(b) \rho_{\text{power-off}} : u = \text{on} \wedge u' = \text{off} \wedge x' = x \wedge 0 \leq t \wedge t' = 0 \wedge 0 \leq T \wedge T' = T.$$

$$(c) \rho_{\text{light-power-off}} : u = \text{on} \wedge u' = \text{on} \wedge x' = x + 1 \wedge t = 10 \wedge t' = 0 \wedge 0 \leq T \wedge T' = T'.$$

$$(d) \rho_{\text{tick}} : \exists \Delta > 0. (u = u' \wedge x' = x \wedge t' = t + \Delta \wedge T' = T + \Delta).$$

以上のシステムの *Receptiveness* を証明する。ここではモジュール *controller* の *Receptiveness* のみ証明する。また、モジュールは閉じたモジュールとして証明するので、環境の遷移  $\tau_E$  も考慮する。 $\tau_E$  は環境の動作である。以下に *Receptiveness* ダイアグラムを示す。

図 19 を以下に説明する。なお、入れ子のノードの外側のノードは内側の上位階層のノードであ

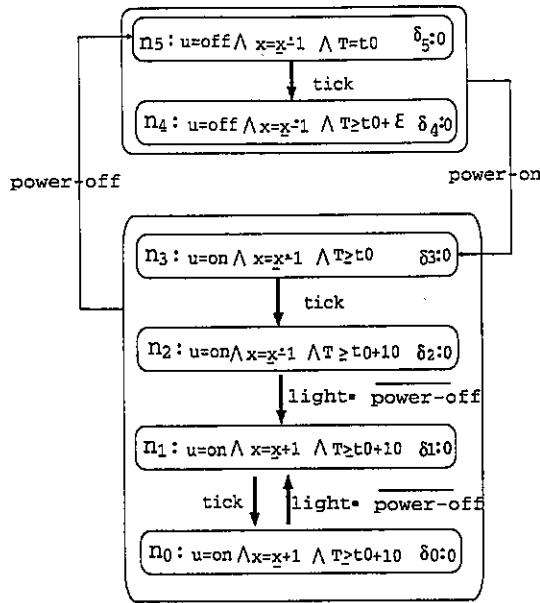


図 19: Receptiveness ダイアグラム

り, 外側のノードに存在することは, 内側のどれかのノードに存在することである. 例えば, 図 19 では,  $power-on$  により,  $n_5$  または  $n_4$  は  $n_3$  に遷移することを意味する. また,  $x = \underline{x} - 1 < 0$  のとき,  $x = \underline{x} - 1$  は  $x = 0$  であり, それ以外のときは,  $x = \underline{x} - 1$  は  $x = \underline{x} - 1$  である. ただし,  $\underline{x}$  は前の状態の  $x$  を意味する. なお, 図 19 では, ノード  $n_5$  より動作して, 最初の  $x$  の値は 0 であるので, 最初は  $\underline{x} = 0$  である.

$u$  は状態名を示すローカル変数である.  $t_0$  はシステムが動作を開始した瞬間のマスタークロック  $T$  の値である. モジュールの動作は太線の矢印で示し, 環境の動作は細線の矢印で示す.

1. ノード  $n_5$  とノード  $n_4$  により, 状態名  $off$  において, 環境の動作である  $\tau_E$  または  $external$  遷移であるイベント  $power-on$  による遷移が発生しなければ, モジュール内の時間が無限に発散することを表現できる. すなわち, この状態において, 環境に勝つモジュールの動作 (モジュールの  $tick$  遷移) があることがわかる.
2. 環境の動作であるイベント  $power-on$  による遷移によって状態名が  $on$  になった場合, 環境が動作を行わなければ, ノード  $n_3$  とノード  $n_2$  より, モジュール内の時間は必ず 10 秒進むことが表現でき, さらに  $controlled$  遷移であるイベント  $light$  による遷移が起きる. そして,  $tick$  遷移と  $light$  による遷移を繰り返して, 時間は無限に発散することを表現できる. すなわち, この状態において, 環境に勝つモジュールの動作 ( $n_1$  において  $tick$  遷移,  $n_0$  においてイベント  $light$  による遷移) があることがわかる.

以上より、モジュール *controller* は環境により何もされなければ、状態名 *off* において時間を無限に発散し続け、環境によって状態名 *on* にされてからも時間を無限に発散し続けることができる。よって、モジュール *controller* が *Receptiveness* を満たすことが証明できた。

■

### 4.3.2 Assume-Guarantee 形式による演繹的詳細化検証手法

まず、クロック遷移モジュールに関する演繹的詳細化検証手法 [42] を定義して、次に Assume-Guarantee 形式による演繹的詳細化検証手法を定義する。

クロック遷移モジュールに関する演繹的詳細化検証の公理系では、具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されれば、具体化仕様が抽象化仕様を詳細化していると考えられる。機能の包含性は通常の論理的な含意であり、タイミング制約の包含性はクロック変数が表現する時間領域が包含されていると考える。例えば、具体化仕様のクロック変数  $t^A$  のタイミング制約が  $l^A \leq t^A \leq u^A$  であり、抽象化仕様のそれが  $l^C \leq t^C \leq u^C$  とすると、 $l^A \leq l^C \leq u^C \leq u^A$  であれば、タイミング制約が包含されるとする。

#### Definition 24 (詳細化の検証ルール)

$\text{CTM}^C = (V^C, \Theta^C, T^C, \Pi^C)$  が  $\text{CTM}^A = (V^A, \Theta^A, T^A, \Pi^A)$  を詳細化していることを検証するルールを以下に示す。ただし、 $\text{CTM}^A$  のすべての共有変数は  $\text{CTM}^C$  の共有変数である。

1.  $\Theta^C \rightarrow \Theta^A[\alpha]$
2.  $\forall \tau^C \in T^C$  に対して以下が成り立つ：  
 $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$
3. -----
4.  $\text{CTM}^C \sqsubseteq \text{CTM}^A$

これは詳細化を証明するルールであり、ある  $\alpha$  が存在して、1と2を満たすとき、4のように  $\sqsubseteq$  が定義されることを意味する。なお、含意  $\rightarrow$  は機能とタイミング制約との両方の包含関係を意味しており、詳細化関係を表す。  $\text{CTM}^A$  のすべての共有変数は  $\text{CTM}^C$  の共有変数であるという条件は、外部の環境に与える影響（すなわち、共有変数に関する動作）が  $\text{CTM}^A$  と  $\text{CTM}^C$  で同じである必要があるからである。

ここで、ルールの各行は、以下を意味する。

1. 1行目は初期状態集合  $\Theta^C$  ならば  $\Theta^A[\alpha]$  であることを意味する。
2. 2行目は  $\rho_{\tau^C}$  の状態遷移が抽象的な状態遷移  $\rho_{\tau^A}[\alpha]$  により説明される事を意味する。
3. 3行目は前提と結論を分離するラインである。
4. 4行目は結論  $\text{CTM}^C \sqsubseteq \text{CTM}^A$  であり、具体化仕様が抽象化仕様を正しく詳細化していることを意味する。

ここで、 $\alpha$ は抽象化仕様のローカル変数  $V^{AP}$  を具体化仕様の変数  $V^C$  の算術式に置き換える写像又は  $V^{AP}$  の論理式を  $V^C$  の論理式に置き換える写像を意味する。ただし、算術式及び論理式は  $CTM^C$  または  $CTM^A$  に現れるものである（より正確には、算術式及び論理式は *Pnueli* らの文献 [51] 中の  $SPL_T$  言語で定義されるものである）。なお、 $\alpha$ は文献 [73] の *refinement mapping*（詳細化写像）に相当する。

■

まず、具体化仕様の状態と抽象化仕様の状態が1対1の場合の詳細化検証の事例を説明して、次に、具体化仕様の状態と抽象化仕様の状態が多対1の場合の詳細化検証の事例を説明する。

**Example 11 (詳細化検証の例 (1対1の場合))**

図20のように、具体化仕様  $CTM^C = (V^C, \Theta^C, T^C, \Pi^C)$  及び抽象化仕様  $CTM^A = (V^A, \Theta^A, T^A, \Pi^A)$  が与えられたとする。ここで、 $\alpha$ を以下のように定義する：

$$x = \begin{cases} \text{if 具体化仕様の状態が 3 then } y + 1 \\ \text{else } y - 1 \end{cases}$$

なお、 $y \neq 0$  のとき  $y - 1$  は  $y - 1$  である。ただし、 $y = 0$  のときは  $y - 1 = 0 - 1 = 0$  とする。

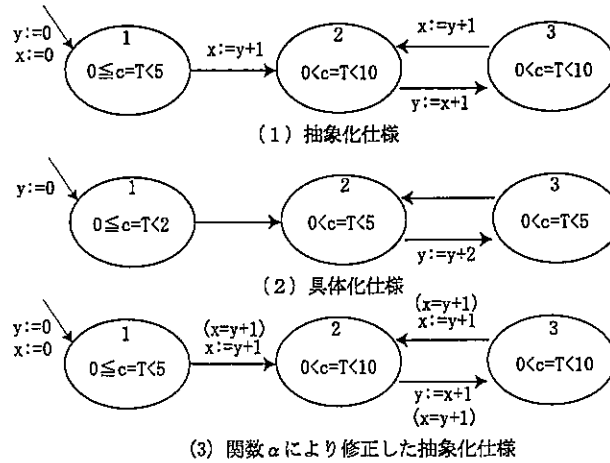


図 20: 詳細化検証の例 (1対1の場合)

図20の状態にタイミング制約式  $0 \leq c = T < 5$  が付けられているが、これは、時間前進条件  $u_1 = 1 \rightarrow 0 \leq c = T < 5$  を意味する。ただし、 $u_1 = 1$  は状態名のローカル変数  $u_1$  が1であることを意味する。

以下では、具体化仕様が抽象化仕様を正しく詳細化していることを検証する。

まず,  $\Theta^C \rightarrow \Theta^A[\alpha]$  が成り立つことを示す. ここで,  $\Theta^C = (u_1 = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  及び  $\Theta^A = (u_1 = 1 \wedge x = 0 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  である. また,  $\Theta^A[\alpha] = (u_1 = 1 \wedge (\text{if 具体化仕様の状態が 2 または 3 then } y + 1 \text{ else } y - 1) = 0 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  である. ゆえに,  $\Theta^C \rightarrow \Theta^A[\alpha]$  は成り立つ.

次に,  $\forall \tau^C \in \mathcal{T}^C$  に対して,  $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in \mathcal{T}^A} \rho_{\tau^A}[\alpha]$  が成り立つことを示す.

1. 具体化仕様が状態 1 から状態 2 に遷移するとき

$(u_1 = 1 \wedge y = 0 \wedge 0 \leq c = T < 2) \wedge (u_1' = 2 \wedge y' = y \wedge 0 \leq c' = T' < 5) \rightarrow (u_1 = 1 \wedge (\text{if 具体化仕様の状態が 2 または 3 then } y + 1 \text{ else } y - 1) = 0 \wedge y = 0 \wedge 0 \leq c = T < 5) \wedge (u_1' = 2 \wedge (\text{if 具体化仕様の状態が 2 または 3 then } y + 1 \text{ else } y - 1)' = y + 1 \wedge y' = y = 0 \wedge 0 \leq c' = T' < 10)$  である. 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されるので,  $\rho_{\tau^C} \rightarrow \rho_{\tau^A}[\alpha]$  は成り立つ.

2. 具体化仕様が状態 2 から状態 3 に遷移するとき

$(u_1 = 2 \wedge y = 0 \wedge 0 < c = T < 5) \wedge (u_1' = 3 \wedge y' = y + 2 \wedge 0 < c' = T' < 5) \rightarrow (u_1 = 2 \wedge (\text{if 具体化仕様の状態が 2 または 3 then } y + 1 \text{ else } y - 1) = y + 1 \wedge y = 0 \wedge 0 < c = T < 10) \wedge (u_1' = 3 \wedge y' = (\text{if 具体化仕様の状態が 2 または 3 then } y + 1 \text{ else } y - 1) + 1 \wedge 0 < c' = T' < 10)$  である. 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されるので,  $\rho_{\tau^C} \rightarrow \rho_{\tau^A}[\alpha]$  は成り立つ.

3. 具体化仕様が状態 3 から状態 2 に遷移するとき

$(u_1 = 3 \wedge 0 < c = T < 5) \wedge (u_1' = 2 \wedge y' = y \wedge 0 < c' = T' < 5) \rightarrow (u_1 = 3 \wedge 0 < c = T < 10) \wedge (u_1' = 2 \wedge (\text{if 具体化仕様の状態が 2 または 3 then } y + 1 \text{ else } y - 1)' = y + 1 \wedge y' = y \wedge 0 < c' = T' < 10)$  である. 具体化仕様の機能とタイミング制約が抽象化仕様の機能とタイミング制約に包含されるので,  $\rho_{\tau^C} \rightarrow \rho_{\tau^A}[\alpha]$  は成り立つ.

上記 (2) と (3) を繰り返すので,  $\forall \tau^C \in \mathcal{T}^C$  に対して,  $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in \mathcal{T}^A} \rho_{\tau^A}[\alpha]$  が成り立つ.

以上より, 具体化仕様が抽象化仕様を正しく詳細化していることが検証できた. ■

**Example 12 (詳細化検証の例 (多対 1 の場合))**

図 2 1 のように, 具体化仕様  $\text{CTM}^C = (V^C, \Theta^C, \mathcal{T}^C, \Pi^C)$  及び抽象化仕様  $\text{CTM}^A = (V^A, \Theta^A, \mathcal{T}^A, \Pi^A)$  が与えられたとする. ここで,  $\alpha$  を以下のように定義する:

$$u_1^A = \begin{cases} \text{if } u_1^C = 2 \text{ then } 3 \\ \text{else } u_1^C \end{cases}$$

ただし,  $u_1^A$  は抽象化仕様の状態名を示すローカル変数であり,  $u_1^C$  は具体化仕様の状態名を示すローカル変数である.  $\alpha$  は,  $u_1^C = 2$  ならば  $u_1^A = 3$  であり,  $u_1^C = 2$  でないならば  $u_1^A = u_1^C$  を意味する. つまり,  $\alpha$  は以下を意味する:

1.  $u_1^A = 1$  を  $u_1^C = 1$  に置き換える.
2.  $u_1^A = 3$  を  $u_1^C = 2 \vee u_1^C = 3$  に置き換える

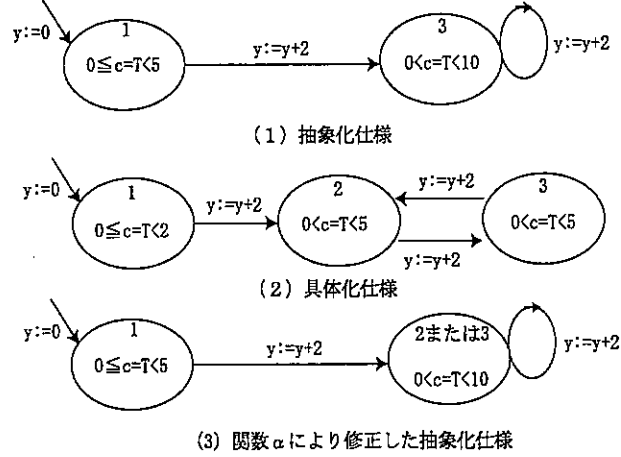


図 21: 詳細化検証の例 (多対 1 の場合)

以下では, 具体化仕様が抽象化仕様を正しく詳細化していることを検証する.

まず,  $\Theta^C \rightarrow \Theta^A[\alpha]$  が成り立つことを示す. ここで,  $\Theta^C = (u_1^C = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  及び  $\Theta^A = (u_1^A = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  である.  $\Theta^A$  の状態  $u_1^A = 1$  は,  $\alpha$  により,  $u_1^C = 1$  に置き換える. ゆえに,  $\Theta^A[\alpha] = (u_1^C = 1 \wedge y = 0 \wedge c = 0 \wedge T = 0)$  なので,  $\Theta^C \rightarrow \Theta^A[\alpha]$  は成り立つ.

次に,  $\forall \tau^C \in T^C$  に対して,  $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$  が成り立つことを示す.

1. 具体化仕様が状態 1 から状態 2 に遷移するとき

$(u_1^C = 1 \wedge y = 0 \wedge 0 \leq c = T < 2) \wedge (u_1^{C'} = 2 \wedge y' = y + 2 \wedge 0 \leq c' = T' < 5) \rightarrow (u_1^A = 1 \wedge y = 0 \wedge 0 \leq c = T < 5) \wedge (u_1^{A'} = 3 \wedge y' = y \wedge 0 \leq c' = T' < 10)$  であり,  $\alpha$  により, 上記は成り立つ.

2. 具体化仕様が状態 2 から状態 3 に遷移するとき

$(u_1^C = 2 \wedge 0 < c = T < 5) \wedge (u_1^{C'} = 3 \wedge y' = y + 2 \wedge 0 < c' = T' < 5) \rightarrow (u_1^A = 3 \wedge 0 < c = T < 10) \wedge (u_1^{A'} = 3 \wedge y' = y + 2 \wedge 0 < c' = T' < 10)$  であり,  $\alpha$  により, 上記は成り立つ.

3. 具体化仕様が状態 3 から状態 2 に遷移するとき

$(u_1^C = 3 \wedge y = 0 \wedge 0 < c = T < 5) \wedge (u_1^{C'} = 2 \wedge y' = y + 2 \wedge 0 < c' = T' < 5) \rightarrow (u_1^A = 3 \wedge y = 0 \wedge 0 < c = T < 10) \wedge (u_1^{A'} = 3 \wedge y' = y + 2 \wedge 0 < c' = T' < 10)$  であり,  $\alpha$  により, 上記は成り立つ.

上記 (2) と (3) を繰り返すので,  $\forall \tau^C \in T^C$  に対して,  $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$  が成り立つ.

ゆえに、具体化仕様が抽象化仕様を正しく詳細化していることが検証できた。

■

次に、Assume-Guarantee 形式を基礎とする演繹的詳細化検証手法を定義する。モジュール 1 の抽象度の高い仕様  $CTM_1^A$  とモジュール 2 の抽象度の高い仕様  $CTM_2^A$  から構成される並列動作する実時間ソフトウェアがモジュール 1 の抽象度の低い仕様  $CTM_1^C$  とモジュール 2 の抽象度の低い仕様  $CTM_2^C$  から構成される並列動作する実時間ソフトウェアによって、正当に詳細化されていることを検証する場合を考える。この検証問題は、 $CTM_1^C \parallel CTM_2^C \sqsubseteq CTM_1^A \parallel CTM_2^A$  を示すことであるが、 $CTM_1^C \parallel CTM_2^C$  の状態数が多くなり、検証が困難である。そこで、以下のように、 $CTM_1^C \parallel CTM_2^C$  の構成を避けて、構成的に詳細化を検証する方法を提案する。ただし、 $CTM_1^A \parallel CTM_2^A$  は  $CTM_1^A$  と  $CTM_2^A$  が並列動作する実時間ソフトウェアを意味する。

**Theorem 3 (Assume-Guarantee 形式演繹的詳細化検証)** もし、 $CTM_1^C \parallel CTM_2^A \sqsubseteq CTM_1^A$  と  $CTM_2^C \parallel CTM_1^A \sqsubseteq CTM_2^A$  ならば、 $CTM_1^C \parallel CTM_2^C \sqsubseteq CTM_1^A \parallel CTM_2^A$  である。ただし、 $CTM_1^A$  と  $CTM_2^A$ ,  $CTM_1^C$ ,  $CTM_2^C$  は、すべて *receptiveness* を充足する。また、 $CTM_1^A \parallel CTM_2^A$  のすべての共有変数は  $CTM_1^C \parallel CTM_2^C$  の共有変数である。ここで、 $CTM_1^A$  : モジュール 1 の抽象度の高い仕様、 $CTM_2^A$  : モジュール 2 の抽象度の高い仕様、 $CTM_1^C$  : モジュール 1 の抽象度の低い仕様、 $CTM_2^C$  : モジュール 2 の抽象度の低い仕様である。

### Proof 3

(記号の説明)

以下の証明では、次のような記号を用いる：ここで、 $\alpha_i$  は  $CTM_i^A$  のローカル変数  $V_i^{AP}$  を  $CTM_i^C$  の変数  $V_i^C$  の算術式に置換すること又は  $CTM_i^A$  のローカル変数  $V_i^{AP}$  の論理式を  $CTM_i^C$  の変数  $V_i^C$  の論理式に置換することを意味する。

また、 $\rho_{\tau_j}^{iC}$  は  $CTM_i^C$  の  $j$  番目の状態遷移関係であり、 $\rho_{\tau_j}^{iA}[\alpha_i]$  は  $CTM_i^A$  の  $j$  番目の状態遷移関係とする。

(記号の説明終)

まず、 $CTM_1^C$  は *receptiveness* を満たすので、 $CTM_1^C$  の初期条件  $\Theta^{1C}$  及び状態遷移関係の列  $\rho_{\tau_1}^{1C}$ ,  $\rho_{\tau_2}^{1C}$ ,  $\rho_{\tau_3}^{1C}$ , ... が存在する。次に、 $CTM_2^C$  は *receptiveness* を満たすので、 $CTM_2^C$  の初期条件  $\Theta^{2C}$  及び状態遷移関係の列  $\rho_{\tau_1}^{2C}$ ,  $\rho_{\tau_2}^{2C}$ ,  $\rho_{\tau_3}^{2C}$ , ... が存在する。

また、 $CTM_1^A$  は *receptiveness* を満たすので、 $CTM_1^A$  の初期条件  $\Theta^{1A}[\alpha_1]$  及び状態遷移関係の列  $\rho_{\tau_1}^{1A}[\alpha_1]$ ,  $\rho_{\tau_2}^{1A}[\alpha_1]$ ,  $\rho_{\tau_3}^{1A}[\alpha_1]$ , ... が存在する。次に、 $CTM_2^A$  は *receptiveness* を満たすので、 $CTM_2^A$  の初期条件  $\Theta^{2A}[\alpha_2]$  及び状態遷移関係の列  $\rho_{\tau_1}^{2A}[\alpha_2]$ ,  $\rho_{\tau_2}^{2A}[\alpha_2]$ ,  $\rho_{\tau_3}^{2A}[\alpha_2]$ , ... が存在する。

この定理の証明では、 $CTM_1^C \parallel CTM_2^A \sqsubseteq CTM_1^A$  と  $CTM_2^C \parallel CTM_1^A \sqsubseteq CTM_2^A$  を満たす初期条件及び状態遷移関係の列から、 $CTM_1^C \parallel CTM_2^C \sqsubseteq CTM_1^A \parallel CTM_2^A$  を満たす初期条件及び状態遷移関係の列を構成する。

以下では、一般性を失うことなく、証明を簡単にするために、モジュールの並列合成はすべて変数が等しい場合を考える。つまり、モジュールの並列合成は状態遷移関係の論理積と考える。他の場合も同様に証明できるので、紙面の都合上から、上記場合を証明する。

$CTM_1^C \parallel CTM_2^A \sqsubseteq CTM_1^A$  と  $CTM_2^C \parallel CTM_1^A \sqsubseteq CTM_2^A$  を満たす初期条件及び状態遷移関係の列から、以下のような初期条件及び状態遷移関係の列を構成する：

1. 初期条件の構成：

- (a) 仮定より、 $\Theta^{1C} \wedge \Theta^{2A}[\alpha_2] \rightarrow \Theta^{1A}[\alpha_1]$  なので、 $\Theta^{1C} \wedge \Theta^{2A}[\alpha_2] \rightarrow \Theta^{1A}[\alpha_1] \wedge \Theta^{2A}[\alpha_2]$  である。ゆえに、 $\Theta^{1C} \rightarrow \Theta^{1A}[\alpha_1]$  である。
  - (b) 仮定より、 $\Theta^{2C} \wedge \Theta^{1A}[\alpha_1] \rightarrow \Theta^{2A}[\alpha_2]$  なので、 $\Theta^{2C} \wedge \Theta^{1A}[\alpha_1] \rightarrow \Theta^{1A}[\alpha_1] \wedge \Theta^{2A}[\alpha_2]$  である。ゆえに、 $\Theta^{2C} \rightarrow \Theta^{2A}[\alpha_2]$  である。
- 以上の (a) と (b) より、 $\Theta^{1C} \wedge \Theta^{2C} \rightarrow \Theta^{1A}[\alpha_1] \wedge \Theta^{2A}[\alpha_2]$  である。

2. 状態遷移関係の構成：

仮定より、 $\rho_{\tau_1}^{1C} \wedge \rho_{\tau_1}^{2A}[\alpha_2] \rightarrow \rho_{\tau_1}^{1A}[\alpha_1]$  かつ  $\rho_{\tau_1}^{2C} \wedge \rho_{\tau_1}^{1A}[\alpha_1] \rightarrow \rho_{\tau_1}^{2A}[\alpha_2]$  である。すなわち、 $\rho_{\tau_1}^{1C} \wedge \rho_{\tau_1}^{2A}[\alpha_2]$  が  $\rho_{\tau_1}^{1A}[\alpha_1]$  を詳細化する関係が存在して、かつ、 $\rho_{\tau_1}^{2C} \wedge \rho_{\tau_1}^{1A}[\alpha_1]$  が  $\rho_{\tau_1}^{2A}[\alpha_2]$  を詳細化する関係が存在する。ゆえに、 $\rho_{\tau_1}^{1C} \wedge \rho_{\tau_1}^{2C}$  が  $\rho_{\tau_1}^{1A}[\alpha_1] \wedge \rho_{\tau_1}^{2A}[\alpha_2]$  を詳細化する関係が構成できる。以上より、 $\rho_{\tau_1}^{1C} \wedge \rho_{\tau_1}^{2C} \rightarrow \rho_{\tau_1}^{1A}[\alpha_1] \wedge \rho_{\tau_1}^{2A}[\alpha_2]$  が構成できる。

以下では、 $\rho_{\tau_2}^{1C} \wedge \rho_{\tau_2}^{2C} \rightarrow \rho_{\tau_2}^{1A}[\alpha_1] \wedge \rho_{\tau_2}^{2A}[\alpha_2]$  を構成する：

- (a) すべてのモジュールは *receptiveness* を満たすので、 $\rho_{\tau_1}^{2A}[\alpha_2]$ ,  $\rho_{\tau_2}^{2A}[\alpha_2]$  といった状態遷移列が存在する。ゆえに、 $\rho_{\tau_1}^{1C} \wedge \rho_{\tau_1}^{2A}[\alpha_2]$ ,  $\rho_{\tau_2}^{1C} \wedge \rho_{\tau_2}^{2A}[\alpha_2]$  といった状態遷移列が存在する。
- (b) 仮定より、 $\rho_{\tau_1}^{1C} \wedge \rho_{\tau_1}^{2A}[\alpha_2] \rightarrow \rho_{\tau_1}^{1A}[\alpha_1]$  であり、 $\rho_{\tau_1}^{1A}[\alpha_1]$ ,  $\rho_{\tau_2}^{1A}[\alpha_1]$  といった状態遷移列が存在する。ゆえに、 $\rho_{\tau_2}^{1C} \wedge \rho_{\tau_2}^{2A}[\alpha_2] \rightarrow \rho_{\tau_2}^{1A}[\alpha_1]$  が構成できる。
- (c) すべてのモジュールは *receptiveness* を満たすので、 $\rho_{\tau_1}^{1A}[\alpha_1]$ ,  $\rho_{\tau_2}^{1A}[\alpha_1]$  といった状態遷移列が存在する。ゆえに、 $\rho_{\tau_1}^{2C} \wedge \rho_{\tau_1}^{1A}[\alpha_1]$ ,  $\rho_{\tau_2}^{2C} \wedge \rho_{\tau_2}^{1A}[\alpha_1]$  といった状態遷移列が存在する。
- (d) 仮定より、 $\rho_{\tau_1}^{2C} \wedge \rho_{\tau_1}^{1A}[\alpha_1] \rightarrow \rho_{\tau_1}^{2A}[\alpha_2]$  であり、 $\rho_{\tau_1}^{2A}[\alpha_2]$ ,  $\rho_{\tau_2}^{2A}[\alpha_2]$  といった状態遷移列が存在する。ゆえに、 $\rho_{\tau_2}^{2C} \wedge \rho_{\tau_2}^{1A}[\alpha_1] \rightarrow \rho_{\tau_2}^{2A}[\alpha_2]$  が構成できる。

以上の (a), (b), (c), (d) より、

$$\rho_{\tau_2}^{1C} \wedge \rho_{\tau_2}^{2C} \rightarrow \rho_{\tau_2}^{1A}[\alpha_1] \wedge \rho_{\tau_2}^{2A}[\alpha_2]$$

が構成できる。

以上を繰り返すと、以下が構成できる：

$$\rho_{\tau_3}^{1C} \wedge \rho_{\tau_3}^{2C} \rightarrow \rho_{\tau_3}^{1A}[\alpha_1] \wedge \rho_{\tau_3}^{2A}[\alpha_2],$$

$$\rho_{\tau_4}^{1C} \wedge \rho_{\tau_4}^{2C} \rightarrow \rho_{\tau_4}^{1A}[\alpha_1] \wedge \rho_{\tau_4}^{2A}[\alpha_2],$$

.....  
 .....

ゆえに,  $CTM_1^C \parallel CTM_2^C \sqsubseteq CTM_1^A \parallel CTM_2^A$  が証明できた。

■

#### 4.4 Assume-Guarantee 形式による演繹的詳細化検証の事例

本章では, 簡単な踏み切りの事例により, 提案した手法の有効性を示す。

##### 4.4.1 踏み切りの事例

図 22 は簡単な踏み切りの具体化仕様である。Train<sup>C</sup> は列車を示すモジュールである。Gate<sup>C</sup> は踏み切りを示すモジュールである。この踏み切りは列車が近づくと門を下げ, 列車が踏み切りから出て少し離れると門を上げるだけの踏み切りである。

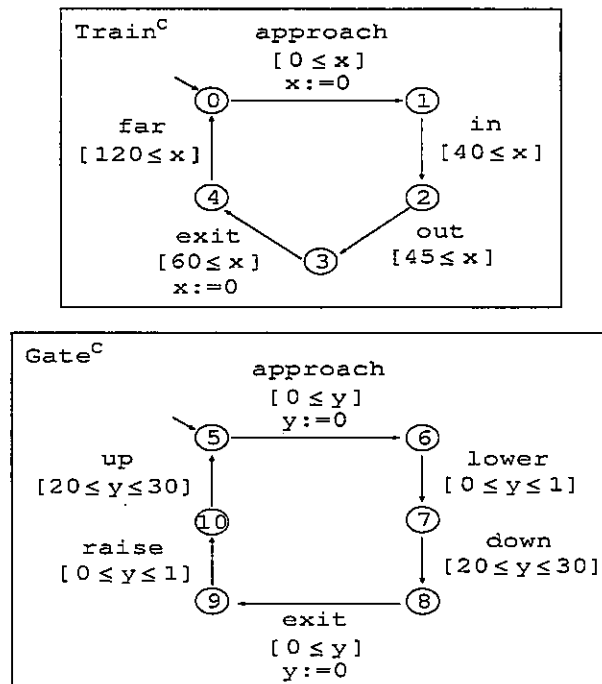


図 22: 踏み切りの具体化仕様

以下に図 22 を説明する。ここで, 丸は状態を示し, 中の整数は状態名である。また, 矢印は状態遷移を示して, 遷移に付く文字列は上から遷移名, 中間の [ ] で囲まれた文字列は遷移の可能な条件, 下の文字列は変数への代入を示す。

1. モジュール  $Train^C$  を以下に説明する.

- (a)  $x$  は実数型のクロック変数である. 単位は秒である.
- (b) 遷移 approach は, 列車が踏み切りに近づいたことを示す.
- (c) 遷移 in は, 列車が踏み切りに入ったことを示し, どの列車も踏み切りに近づいてから 40 秒間より小さい時間間隔で踏み切りに入らないとする.
- (d) 遷移 out は, 列車が踏み切りから出たことを示し, どの列車も踏み切りに近づいてから踏み切りを出るまで 45 秒間以上は必要であるとする.
- (e) 遷移 exit は, 列車が踏み切りを出て少し離れたことを示し, どの列車も踏み切りに近づいてから踏み切りを出て少し離れるまで 60 秒間以上は必要であるとする.
- (f) 遷移 far は, 列車が踏み切りを出て遠く離れたことを示し, どの列車も踏み切りを出て少し離れてから 120 秒間以上で遠く離れるとする. これより後の状態遷移は, 次に踏み切りに近づいてきた列車を示すとする.

2. モジュール  $Gate^C$  を以下に説明する.

- (a)  $y$  は実数型のクロック変数である. 単位は秒である.
- (b) 遷移 approach は, 列車が踏み切りに近づいたことにより送られてくる信号を受信したことを示す.
- (c) 遷移 lower は, 受信した信号を処理し, 門を下げ始めたことを示す. 0 秒以上 1 秒間以下で処理を終え門を下げ始めるとする.
- (d) 遷移 down は, 門を下げ終えたことを示す. 信号を受信してから 20 秒間以上 30 秒間以下で下げ終えたとする.
- (e) 遷移 exit は, 列車が踏み切りを出て少し離れたことにより送られてくる信号を受信したことを示す.
- (f) 遷移 raise は, 受信した信号を処理し, 門を上げ始めたことを示す. 0 秒以上 1 秒間以下で処理を終え門を上げ始めるとする.
- (g) 遷移 up は, 門を上げ終えたことを示す. 信号を受信してから 20 秒間以上 30 秒間以下で上げ終えたとする.

3. 遷移名の一致する遷移は同期する.

図 2 2 の各モジュールが Receptiveness を満たすことを以下に証明する. ここで, 紙面の都合上,  $Gate^C$  の証明のみ示すが,  $Train^C$  も Receptiveness を満たす.

はじめに, Receptiveness の証明には遷移の区別が必要であるので,  $Gate^C$  の各遷移をモジュールが制御する遷移と環境が制御する遷移に区別する. このモジュールにおいて, 遷移 approach と遷移 exit は環境である  $Train^C$  によって制御される external 遷移であり, その他の遷移はこのモジュール自身が制御する controlled 遷移である.

次に, 遷移の区別を考慮し, 環境の状態遷移である  $\tau_E$  を加えた閉じたシステムとして, Receptiveness ダイアグラムを作る. 図 2 3 は,  $Gate^C$  の Receptiveness ダイアグラムである. ここで,  $n_i$  はノード名

であり,  $u_2$  は状態名を示すローカル変数である.  $\delta$  はランキング関数であり,  $\delta_i : i$  は  $\delta(n_i) = i$  である.  $\epsilon$  は  $\epsilon = 20$  である. また, 太矢印はモジュールが制御する controlled 遷移または tick 遷移であり, 細矢印は環境が制御する external 遷移または  $\tau_E$  である.

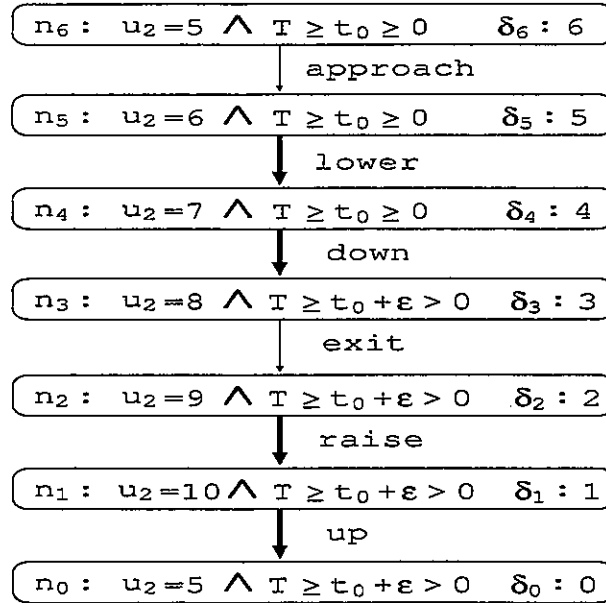


図 23:  $Gate^C$  の Receptiveness ダイアグラム

図 23 の Receptiveness ダイアグラムは Receptiveness の条件を満たしている. よって,  $Gate^C$  が Receptiveness を満たすことを証明できたことになる.

図 23 の Receptiveness ダイアグラムを説明する. このダイアグラムの始まりのノードは  $n_6$  または  $n_5$  または  $n_4$  である, 始まりの時刻を  $t_0$  とすると, ノード  $n_3$  に到達すれば実時間は必ず  $t_0 + \epsilon$  以上進んでいることを示す. これは遷移 approach と遷移 lower が  $T = t_0$  で発生しても, 遷移 down は必ず  $\epsilon$  以上の時間が経過した後で発生するというにより示される. すなわち, ノード  $n_3$  の示す状態に到達するたびに  $\epsilon$  以上は確実に実時間は進むことを示す. また, 細矢印で示される環境に制御される遷移はいつでも発生することが可能であるが, 発生しない可能性も考えられる. しかし, 発生しない場合はその他の遷移 ( $\tau_E$  または tick) を発生させているか, モジュールが tick 遷移を発生させているかのどちらかであり, 環境は時間の前進を妨げないように動作しているはずであるので, 細矢印で示される遷移が発生しなければ時間は前進し続けることになる. また, このダイアグラムはノード  $n_0$  に到達し終わるが, ノード  $n_0$  の示す状態に到達した時点の実時間の値を  $t_0$  とすることで, 同じダイアグラムで無限の計算における Receptiveness を証明することができる. よって, 各ノードで示される状態を各ノードから出ている矢印の示す遷移に写像する戦略が "モジュールが勝つ戦略" である. この "モジュールが勝つ戦略" による状態遷移を無限に続けていくとノード  $n_3$  の示す状態に無限に到達す

るので少なくとも  $\epsilon$  ずつ実時間は進んでいき、発散する。

#### 4.4.2 Assume-Guarantee 形式による演繹的詳細化検証実験

本章では、Receptiveness の証明で用いた踏み切りの仕様である図 2 2 とその抽象化仕様である図 2 4 を用いて、Assume-Guarantee 形式による演繹的詳細化検証の有効性を示す。なお、状態や遷移の意味、表記法は図 2 2 と同様である。

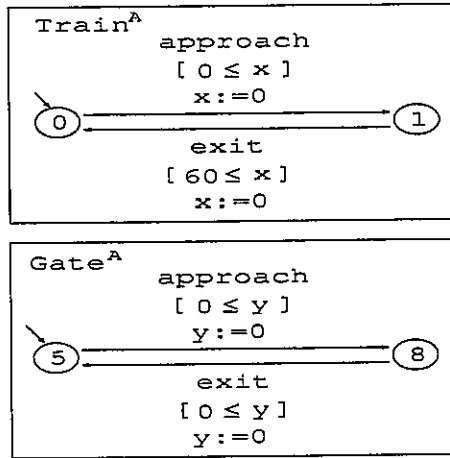


図 24: 踏み切りの抽象化仕様

図 2 2 と図 2 4 に対する Assume-Guarantee 形式による演繹的詳細化検証は、以下の二つが成り立つことを証明することで、 $Train^C \parallel Gate^C \sqsubseteq Train^A \parallel Gate^A$  が成り立つことを証明することである。

1.  $Train^C \parallel Gate^A \sqsubseteq Train^A$ .
2.  $Train^A \parallel Gate^C \sqsubseteq Gate^A$ .

ここで、Receptiveness の証明は、紙面の都合上、省略するが、すべてのモジュールは Receptiveness を満たしている。

$Train^A \parallel Gate^C \sqsubseteq Gate^A$  の証明を以下に示す。

はじめに、二つのモジュールの並列合成  $Train^A \parallel Gate^C$  のクロック遷移モジュールを以下に示す。ここで、モジュール  $Train^A$  の状態名を示すローカル変数を  $U_1$ 、モジュール  $Gate^C$  の状態名を示すローカル変数を  $u_2$  とする。

1.  $V = \{U_1, x, T\} \cup \{u_2, y, T\}$ .
  - (a)  $V^P = \{U_1, x\} \cup \{u_2, y\}$ .
  - (b)  $V^S = \{T\}$ .

2.  $\Theta : (U_1 = 0 \wedge x = 0 \wedge T = 0) \wedge (u_2 = 5 \wedge y = 0 \wedge T = 0)$ .
3.  $\Pi : (u_2 = 6 \rightarrow y \leq 1) \wedge (u_2 = 7 \rightarrow y \leq 30) \wedge (u_2 = 9 \rightarrow y \leq 1) \wedge (u_2 = 10 \rightarrow y \leq 30)$ .
4. 各  $\tau \in T$  に対する状態遷移関係  $\rho_\tau$  は以下のように定義される:
  - (a)  $\rho_{\text{approach}} : (U_1 = 0 \wedge U_1' = 1 \wedge 0 \leq x \wedge x' = 0 \wedge 0 \leq T \wedge T = T') \wedge (u_2 = 5 \wedge u_2' = 6 \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T')$ .
  - (b)  $\rho_{\text{lower}} : (U_1 = U_1' \wedge x = x') \wedge (u_2 = 6 \wedge u_2' = 7 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T')$ .
  - (c)  $\rho_{\text{down}} : (U_1 = U_1' \wedge x = x') \wedge (u_2 = 7 \wedge u_2' = 8 \wedge 20 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T')$ .
  - (d)  $\rho_{\text{exit}} : (U_1 = 1 \wedge U_1' = 0 \wedge 60 \leq x \wedge x' = 0 \wedge 0 \leq T \wedge T = T') \wedge (u_2 = 8 \wedge u_2' = 9 \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T')$ .
  - (e)  $\rho_{\text{raise}} : (U_1 = U_1' \wedge x = x') \wedge (u_2 = 9 \wedge u_2' = 10 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T')$ .
  - (f)  $\rho_{\text{up}} : (U_1 = U_1' \wedge x = x') \wedge (u_2 = 10 \wedge u_2' = 5 \wedge 20 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T')$ .
  - (g)  $\rho_{\text{tick}} : \exists \Delta > 0. (U_1 = U_1' \wedge u_2 = u_2' \wedge x' = x + \Delta \wedge y' = y + \Delta \wedge T' = T + \Delta \wedge (u_2 = 6 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 7 \rightarrow y + \Delta \leq 30) \wedge (u_2 = 9 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 10 \rightarrow y + \Delta \leq 30))$ .

次に、モジュール  $Gate^A$  のクロック遷移モジュールを以下に示す。ここで、このモジュールの状態名を示すローカル変数を  $U_2$  とする。

1.  $V = \{U_2, y, T\}$ .
  - (a)  $V^P = \{U_2, y\}$ .
  - (b)  $V^S = \{T\}$ .
2.  $\Theta : U_2 = 5 \wedge y = 0 \wedge T = 0$ .
3.  $\Pi$  : なし.
4. 各  $\tau \in T$  に対する状態遷移関係  $\rho_\tau$  は以下のように定義される:
  - (a)  $\rho_{\text{approach}} : U_2 = 5 \wedge U_2' = 8 \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T'$ .
  - (b)  $\rho_{\text{exit}} : U_2 = 8 \wedge U_2' = 5 \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T'$ .
  - (c)  $\rho_{\text{tick}} : \exists \Delta > 0. (U_2 = U_2' \wedge y' = y + \Delta \wedge T' = T + \Delta)$ .

次に、 $Train^A \parallel Gate^C$  の各遷移の対応する  $Gate^A$  の遷移を以下に示す：

	$Train^A \parallel Gate^C$	$Gate^A$
対応する遷移	approach	approach
	lower	idling
	down	idling
	exit	exit
	raise	idling
	up	idling
	tick	tick

ここで,  $Gate^A$  の遷移 idling の遷移関係は以下である:  $\rho_{idling} : U_2 = U_2' \wedge y = y' \wedge T = T'$ .

次に, 詳細化写像  $\alpha$  を以下に定義する:

$$U_2 = \begin{cases} \text{if } u_2 = 6 \text{ then } 8 \\ \text{elseif } u_2 = 7 \text{ then } 8 \\ \text{elseif } u_2 = 9 \text{ then } 5 \\ \text{elseif } u_2 = 10 \text{ then } 5 \\ \text{else } u_2 \end{cases}$$

つまり,  $\alpha$  は以下を意味する:

1.  $u_2 = i (i \neq 6, 7, 9, 10)$  のとき,  $U_2 = i$  を  $u_2 = i$  に置き換える.
2.  $u_2 = 6$  のとき,  $U_2 = 8$  を  $u_2 = 6$  に置き換える.
3.  $u_2 = 7$  のとき,  $U_2 = 8$  を  $u_2 = 7$  に置き換える.
4.  $u_2 = 9$  のとき,  $U_2 = 5$  を  $u_2 = 9$  に置き換える.
5.  $u_2 = 10$  のとき,  $U_2 = 5$  を  $u_2 = 10$  に置き換える.

以下では, 上記置き換えを  $U_2[\alpha]$  と記述する.

次に, 詳細化の検証ルールを用いて,  $Train^A \parallel Gate^C \sqsubseteq Gate^A$  を以下に証明する:

1. 初期条件のルールを証明する.

$$(U_1 = 0 \wedge x = 0 \wedge T = 0) \wedge (u_2 = 5 \wedge y = 0 \wedge T = 0) \rightarrow \underbrace{U_2[\alpha] = 5}_{u_2=5} \wedge y = 0 \wedge T = 0.$$

この論理式は成り立つので, 初期条件のルールを満たす.

2. 状態遷移関係のルールを証明する.

(a)  $\rho_{approach}$  について.

$$(U_1 = 0 \wedge U_1' = 1 \wedge 0 \leq x \wedge x' = 0 \wedge 0 \leq T \wedge T = T') \wedge (u_2 = 5 \wedge u_2' = 6 \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{U_2[\alpha] = 5}_{u_2=5} \wedge \underbrace{U_2[\alpha]' = 8}_{u_2'=6} \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T'.$$

この論理式は成り立つ.

(b)  $\rho_{lower}$  について.

$$(U_1 = U_1' \wedge x = x') \wedge (u_2 = 6 \wedge u_2' = 7 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{U_2[\alpha] = 8}_{u_2=6} = \underbrace{U_2[\alpha]' = 8}_{u_2'=7} \wedge y = y' \wedge T = T'.$$

この論理式は成り立つ.

(c)  $\rho_{down}$  について.

$$(U_1 = U_1' \wedge x = x') \wedge (u_2 = 7 \wedge u_2' = 8 \wedge 20 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{U_2[\alpha] = 8}_{u_2=7} =$$

$$\underbrace{U_2[\alpha]' = 8 \wedge y = y' \wedge T = T'}_{u_2'=8}$$

この論理式は成り立つ。

(d)  $\rho_{exit}$  について。

$$(U_1 = 1 \wedge U_1' = 0 \wedge 60 \leq x \wedge x' = 0 \wedge 0 \leq T \wedge T = T') \wedge (u_2 = 8 \wedge u_2' = 9 \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{U_2[\alpha] = 8}_{u_2=8} \wedge \underbrace{U_2[\alpha]' = 5}_{u_2'=9} \wedge 0 \leq y \wedge y' = 0 \wedge 0 \leq T \wedge T = T'.$$

この論理式は成り立つ。

(e)  $\rho_{raise}$  について。

$$(U_1 = U_1' \wedge x = x') \wedge (u_2 = 9 \wedge u_2' = 10 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{U_2[\alpha] = 5}_{u_2=9} =$$

$$\underbrace{U_2[\alpha]' = 5 \wedge y = y' \wedge T = T'}_{u_2'=10}$$

この論理式は成り立つ。

(f)  $\rho_{up}$  について。

$$(U_1 = U_1' \wedge x = x') \wedge (u_2 = 10 \wedge u_2' = 5 \wedge 20 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{U_2[\alpha] = 5}_{u_2=10} =$$

$$\underbrace{U_2[\alpha]' = 5 \wedge y = y' \wedge T = T'}_{u_2'=5}$$

この論理式は成り立つ。

(g)  $\rho_{tick}$  について。

i.  $u_2 = 6$  の場合。

$$\exists \Delta > 0. (U_1 = U_1' \wedge u_2 = u_2' \wedge x' = x + \Delta \wedge y' = y + \Delta \wedge T' = T + \Delta \wedge (u_2 = 6 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 7 \rightarrow y + \Delta \leq 30) \wedge (u_2 = 9 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 10 \rightarrow y + \Delta \leq 30)) \rightarrow \exists \Delta > 0. (\underbrace{U_2[\alpha] = 8}_{u_2=6 \vee u_2=7} = \underbrace{U_2[\alpha]' = 8 \wedge y' = y + \Delta \wedge T' = T + \Delta}_{u_2'=6 \vee u_2'=7}).$$

この論理式は成り立つ。

ii.  $u_2 = 7$  の場合。

$$\exists \Delta > 0. (U_1 = U_1' \wedge u_2 = u_2' \wedge x' = x + \Delta \wedge y' = y + \Delta \wedge T' = T + \Delta \wedge (u_2 = 6 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 7 \rightarrow y + \Delta \leq 30) \wedge (u_2 = 9 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 10 \rightarrow y + \Delta \leq 30)) \rightarrow \exists \Delta > 0. (\underbrace{U_2[\alpha] = 8}_{u_2=6 \vee u_2=7} = \underbrace{U_2[\alpha]' = 8 \wedge y' = y + \Delta \wedge T' = T + \Delta}_{u_2'=6 \vee u_2'=7}).$$

この論理式は成り立つ。

iii.  $u_2 = 9$  の場合。

$$\exists \Delta > 0. (U_1 = U_1' \wedge u_2 = u_2' \wedge x' = x + \Delta \wedge y' = y + \Delta \wedge T' = T + \Delta \wedge (u_2 = 6 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 7 \rightarrow y + \Delta \leq 30) \wedge (u_2 = 9 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 10 \rightarrow y + \Delta \leq 30)) \rightarrow \exists \Delta >$$

$$0. \underbrace{(U_2[\alpha] = 5)}_{u_2=9} = \underbrace{U_2[\alpha]'}_{u_2'=9} = 5 \wedge y' = y + \Delta \wedge T' = T + \Delta.$$

この論理式は成り立つ。

iv.  $u_2 = 10$  の場合.

$$\exists \Delta > 0. (U_1 = U_1' \wedge u_2 = u_2' \wedge x' = x + \Delta \wedge y' = y + \Delta \wedge T' = T + \Delta \wedge (u_2 = 6 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 7 \rightarrow y + \Delta \leq 30) \wedge (u_2 = 9 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 10 \rightarrow y + \Delta \leq 30)) \rightarrow \exists \Delta >$$

$$0. \underbrace{(U_2[\alpha] = 5)}_{u_2=9} = \underbrace{U_2[\alpha]'}_{u_2'=9} = 5 \wedge y' = y + \Delta \wedge T' = T + \Delta.$$

この論理式は成り立つ。

v. その他の場合.

$$\exists \Delta > 0. (U_1 = U_1' \wedge u_2 = u_2' \wedge x' = x + \Delta \wedge y' = y + \Delta \wedge T' = T + \Delta \wedge (u_2 = 6 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 7 \rightarrow y + \Delta \leq 30) \wedge (u_2 = 9 \rightarrow y + \Delta \leq 1) \wedge (u_2 = 10 \rightarrow y + \Delta \leq 30)) \rightarrow \exists \Delta >$$

$$0. \underbrace{(U_2[\alpha] = U_2[\alpha]')}_{u_2} \wedge y' = y + \Delta \wedge T' = T + \Delta.$$

この論理式は成り立つ。

以上より,  $Train^A \parallel Gate^C$  のすべての遷移は状態遷移関係のルールを満たす。

以上より, 検証ルールの前提をすべて満たすので, 結論である以下は成り立つ:

$$Train^A \parallel Gate^C \sqsubseteq Gate^A.$$

以上の  $Train^A \parallel Gate^C \sqsubseteq Gate^A$  と同様に証明することで,  $Train^C \parallel Gate^A \sqsubseteq Train^A$  も成り立つことを証明できる。

以上より,  $Train^C \parallel Gate^A \sqsubseteq Train^A$  と  $Train^A \parallel Gate^C \sqsubseteq Gate^A$  の二つが成り立つことを証明することができたので, 以下が成り立つ。

$$Train^C \parallel Gate^C \sqsubseteq Train^A \parallel Gate^A.$$

よって, 図 2 2 は図 2 4 を詳細化している。

#### 4.4.3 Assume-Guarantee 形式によらない演繹的詳細化検証実験

Assume-Guarantee 形式を用いない場合の演繹的詳細化検証の証明, すなわち,  $Train^C \parallel Gate^C \sqsubseteq Train^A \parallel Gate^A$  を以下に示す。なお, 詳細化写像は Assume-Guarantee 形式を用いる場合の二つの詳細化写像と同様である。

1. 初期条件のルールを証明する。

$$(u_1 = 0 \wedge x = 0 \wedge T = 0) \wedge (u_2 = 5 \wedge y = 0 \wedge T = 0) \rightarrow \underbrace{U_1[\alpha] = 0}_{u_1=0} \wedge x = 0 \wedge T = 0 \wedge \underbrace{U_2[\alpha] = 5}_{u_2=5} \wedge y =$$

$$0 \wedge T = 0.$$

この論理式は成り立つので, 初期条件のルールを満たす。

2. 状態遷移関係のルールを証明する.

(a)  $\rho_{lower}$  について.

i.  $u_1 = 2$  の場合.

$$(u_1 = u_1' \wedge x = x') \wedge (u_2 = 6 \wedge u_2' = 7 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{(U_1[\alpha] = U_1[\alpha]')}_{u_1} \wedge x = x' \wedge T = T' \wedge \underbrace{(U_2[\alpha] = 8)}_{u_2=6} = \underbrace{(U_2[\alpha]' = 8)}_{u_2'=7} \wedge y = y' \wedge T = T'.$$

この論理式は成り立つ.

ii.  $u_1 = 3$  の場合.

$$(u_1 = u_1' \wedge x = x') \wedge (u_2 = 6 \wedge u_2' = 7 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{(U_1[\alpha] = U_1[\alpha]')}_{u_1} \wedge x = x' \wedge T = T' \wedge \underbrace{(U_2[\alpha] = 8)}_{u_2=6} = \underbrace{(U_2[\alpha]' = 8)}_{u_2'=7} \wedge y = y' \wedge T = T'.$$

この論理式は成り立つ.

iii.  $u_1 = 4$  の場合.

$$(u_1 = u_1' \wedge x = x') \wedge (u_2 = 6 \wedge u_2' = 7 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{(U_1[\alpha] = U_1[\alpha]')}_{u_1} \wedge x = x' \wedge T = T' \wedge \underbrace{(U_2[\alpha] = 8)}_{u_2=6} = \underbrace{(U_2[\alpha]' = 8)}_{u_2'=7} \wedge y = y' \wedge T = T'.$$

この論理式は成り立つ.

iv. その他の場合.

$$(u_1 = u_1' \wedge x = x') \wedge (u_2 = 6 \wedge u_2' = 7 \wedge 0 \leq y \wedge y = y' \wedge 0 \leq T \wedge T = T') \rightarrow \underbrace{(U_1[\alpha] = U_1[\alpha]')}_{u_1} \wedge x = x' \wedge T = T' \wedge \underbrace{(U_2[\alpha] = 8)}_{u_2=6} = \underbrace{(U_2[\alpha]' = 8)}_{u_2'=7} \wedge y = y' \wedge T = T'.$$

この論理式は成り立つ.

.....  
.....

以上より,  $Train^C \parallel Gate^C \sqsubseteq Train^A \parallel Gate^A$  が証明できた.

#### 4.4.4 検証実験の評価

抽象化仕様である図 2 4 と具体化仕様である図 2 2 に対する Assume-Guarantee 形式による演繹的詳細化検証と Assume-Guarantee 形式を用いない演繹的詳細化検証を比較すると以下の結果が得られた.

1. Assume-Guarantee 形式を用いる場合.

	$Train^C \parallel Gate^A$ $\sqsubseteq Train^A$	$Train^A \parallel Gate^C$ $\sqsubseteq Gate^A$
証明に必要な 論理式の数	10	12
証明に必要な 論理式の数の総計	22	

## 2. Assume-Guarantee 形式を用いない場合.

	$Train^C \parallel Gate^C$ $\sqsubseteq Train^A \parallel Gate^A$
証明に必要な 論理式の数	54
証明に必要な 論理式の数の総計	54

以上の結果より, Assume-Guarantee 形式を用いると, この例の証明に必要な論理式の数は約 59%減少する. Assume-Guarantee 形式による演繹的詳細化検証は, 並列に動作するモジュール数が多いほど有効であり, また, 詳細化写像が複雑であるほど有効であると考えられる. ゆえに, Assume-Guarantee 形式による演繹的詳細化検証は, 実際のシステムを対象とする場合にさらに有効である.

## 4.5 むすび

本論文では, 実時間ソフトウェアの仕様やプログラムの一般的なモデルである, Pnueli らのクロック遷移モジュール上において, 演繹的詳細化検証手法を提案した. なお, 提案した演繹的詳細化検証手法では, 状態空間の組み合わせ爆発を抑制するために, Assume-Guarantee 形式による検証を実現した. 一般的に, 実時間ソフトウェアでは, データ領域や実行の無限性により, その動作は無限な状態空間を有する時間付きの状態遷移システムになる. ゆえに, モデル検査 [41] は不可能であり, 検証作業は演繹的証明 [54] に頼らざるをえない. 演繹的証明は数学的訓練が必要な作業であり, 大規模システムへの適用は困難である. しかし, 本論文のように, Assume-Guarantee 形式により, 演繹的検証問題を小さくすれば, 実用性は高いと考えられる. 以上より, 今後の研究課題としては, 以下が重要である.

1. 表明の自動生成や証明の再利用, GUI の向上などにより, 演繹的証明作業の効率的な支援 [54] を実現する.

2. 抽象実行 [56] などの抽象化技術を適用することにより, クロック遷移モジュールの自動検証 [57] を実現する.

## 5 ハイブリッドモデルに基づく開放分散システムの設計支援

### 5.1 まえがき

組込み型システムのほとんどは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であり、アナログ動作とデジタル動作が混在している [58, 59]. また、自動車や飛行機などの多くの組込み型システムは safety-critical な状況で動作するので、形式的な開発方法論による信頼性保証が重要であり、従来より多数の信頼性保証のための研究がなされている. 本研究では、ハイブリッドモデルの観点から、組込み型システムをモデル化して信頼性保証できる開発方法論を提案する.

まず、本研究では、以下の図 25 のように、組込み型システムの開発プロセス [60] を考える：

1. 最上流工程：制御系の開発
2. 上流工程：ハイブリッドモデルの開発
3. 中流工程：ソフトウェアモデルの開発（マルチタスクとスケジューリング）
4. 下流工程：プログラムの開発

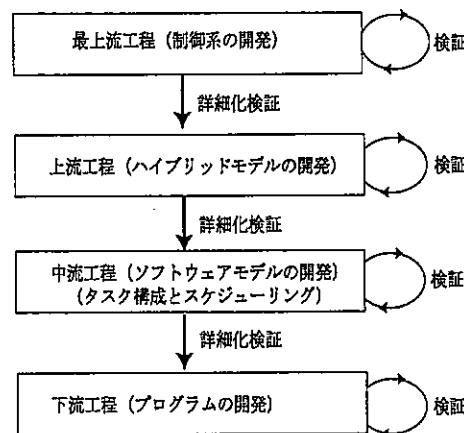


図 25: 組込み型システムの開発プロセス

図 25 の開発プロセスは以下のとおりである：まず、最上流工程では、 $Matrix_x$  とハイブリッドオートマトンにより、組込み型システムの制御法則を仕様記述して、その安全性や活性などを検証する. 次に、上流工程では、上記の制御仕様から、ハイブリッドオートマトンの並列合成を構成して、その安全性や活性などを検証する. 次に、中流工程では、マルチタスクとリアルタイムスケジューラにより、ソフトウェアモデルを構成して、その安全性や活性などを検証する. 最後に、下流工程では、ソフトウェアモデルに従って、プログラム開発して、その安全性や活性などを検証する. さらに、各上位の工程が下位の工程により正しく詳細化されていることを検証する.

最上流工程と上流工程に関しては、既に、我々は、*Matrix<sub>x</sub>* とハイブリッドオートマトンから、ハイブリッドオートマトンの並列合成として上流工程の仕様を構成する手法を提案している [60]。また、各工程の検証に関しても、多数研究されている [61, 62, 49]。

本論文では、上流工程の仕様が並列なハイブリッドオートマトンで構成された後に、ハイブリッドオートマトンで記述された中流工程のソフトウェアモデルを構成して、中流工程の仕様が上流工程の仕様を正しく詳細化していることを検証する手法について述べる。

上流工程で構成された並列ハイブリッドオートマタは、対象とする組込み型システムの特性により、中流工程のソフトウェアモデルに詳細化される。以下に、その主要な特性を列挙する：

1. ハードウェアの特性：
  - (a) CPU が単一かマルチプロセッサか
2. タスクの特性：
  - (a) タスクが周期的か非周期的か散発的か
  - (b) タスクが相互に従属か独立か
3. スケジューリングの特性：
  - (a) スケジューリングがプリエンプティブかノンプリエンプティブか
  - (b) スケジューリングがオンラインかオフラインか
  - (c) スケジューリングが動的か静的か

上記の組込み型システムの特性は、以前より、リアルタイムシステム分野では広く知られている [63]。性能向上を追及して、上記の特性が複雑に絡み合うならば、上流工程から中流工程への詳細化は非常に複雑である。しかし、最近では、マイクロプロセッサの性能が飛躍的に向上して、性能向上を目指したシステム構築よりも設計や保守などの容易性に着目したシステム構築が主流である。さらに、組込み型システムの大規模化がこのような傾向に拍車をかけている。その最たる例としては、C 言語による組込み型システムの開発などであり、最近では、オブジェクト指向手法 [64] などのソフトウェア工学が取り入れられつつある。本論文では、以上の状況から、基本的には、上流工程の仕様の個々のハイブリッドオートマトンが中流工程の個々のタスクのハイブリッドオートマトンに 1 対 1 対応して、それらのタスクがスケジューリングされると仮定した場合の詳細化を考える。さらに、複数のタスクが単一の CPU で制御されており、これらのタスクは非周期的に起動されて相互に独立しており、スケジューリングはオフライン型の静的なプリエンプティブとする。この場合に、中流工程が上流工程を正しく詳細化しているかどうかを検証する手法を提案する。

従来のハイブリッドモデルに基づく組込み型システムの詳細化検証や段階的詳細化開発に関する主要な研究には以下がある：

1. 1993 年に、Henzinger らがフェーズ遷移システム [65] を基礎とした詳細化仕様の構成方法を提案した [66]。これは、フェーズ遷移システムの状態に付いているハイブリッド時相論理式を詳細化しながら、詳細化仕様を構成するものである。詳細化仕様の正当性を保証する手法としては、仕

様の動作の模倣性の検証である。

2. 1995年に、HenzingerがAlurらのハイブリッドオートマトン [67] の双模倣関係の検証アルゴリズムを提案した [68]. これは、ハイブリッドオートマトンの自動詳細化検証の基礎となる双模倣関係を計算するアルゴリズムを開発したものである。
3. 1996年に、LynchらがハイブリッドI/Oオートマトンの模倣関係の公理系を提案した [69]. これは、抽象的なモデルであるハイブリッドI/Oオートマトンの詳細化検証を可能とする基礎理論である。
4. 2000年に、AlurやHenzingerらがCharonやMasaccioといったコンポーネントベースのハイブリッドシステムの詳細化検証を基礎とした開発方法論を提案している [70, 71]. AlurやHenzingerらの方法論は現実のハイブリッドシステムの構造や動作を言語包含性の観点から、詳細化検証するものである。
5. 2001年に、HenzingerらはGiottoと呼ばれる抽象的な組込み型プログラマモデルを開発しており、組込み型プログラムを周期タスクなどから、どのように構成すべきかを定義している [72]. しかし、上流工程からどのようにGiottoを構成すべきかを論じていない。さらに、Giottoが上流工程を正しく詳細化しているかどうかを検証する手法も提案していない。

上記の既存研究の1. から4. は、演繹的または自動的に、模倣関係や言語包含関係による詳細化を保証するものである。しかし、Abadiらが指摘しているように、単なる模倣関係や言語包含関係では、詳細化関係が存在しない場合が多い [73]. なぜならば、抽象化仕様には具体化仕様に含まれない内部動作が存在する場合が多いからである。本節では、この問題を解決するために、抽象化仕様の内部動作を具体化仕様の動作で置換する詳細化写像を導入することによって、詳細化関係を検証する。なお、本論文では、上流工程の抽象化仕様から中流工程の具体化仕様への詳細化開発は上流工程の抽象化仕様の内部動作を詳細化することを前提とする。上記の詳細化写像により、この前提は形式化されて、詳細化検証を実現するものである。

本節では、あるスケジューリングポリシーを基礎として、組込み型システムの上流工程の仕様を中流工程のソフトウェアモデルに実装した場合に、詳細化写像を導入した詳細化検証手法を提案して、その詳細化の正しさを検証する。なお、上流工程の仕様と中流工程のソフトウェアモデルはハイブリッドオートマトンで仕様記述する。

以降の本節の構成は以下のとおりである。5. 2章では、組込み型システムの上流工程から中流工程への段階的詳細化開発を述べる。5. 3章では、中流工程が上流工程を正しく詳細化していることを検証する手法を提案する。最後に、5. 4章では、まとめと今後の課題を述べる。

## 5.2 上流工程から中流工程への段階的詳細化開発

本節では、本論文で事例とする列車問題の説明及びその上流工程と中流工程の仕様の説明を行う。なお、本論文では、複数の線路のゲートの開閉が単一のCPUで制御されており、ゲートの開閉のタス

クは非周期的に起動されて相互に独立しており、スケジューリングはオフライン型の静的なプリエンブタイプとする。

### 5.2.1 組込み型システムの事例

本論文では、複数の線路のゲートの開閉が単一の CPU 上のコントローラで制御される問題を対象とする。

まず、以下の図 26 に、その列車問題を図示する。図 26 (1) は、2つの線路上のゲートの開閉が互いに独立な2つのコントローラで制御される様子を示す。図 26 (2) は、制御系の観点から、図 26 (1)を図示する。図 26 (2) では、2つの列車をセンシングするセンサが2つあり、2つのゲートをアクチュエートするアクチュエータが2つあることを示す。

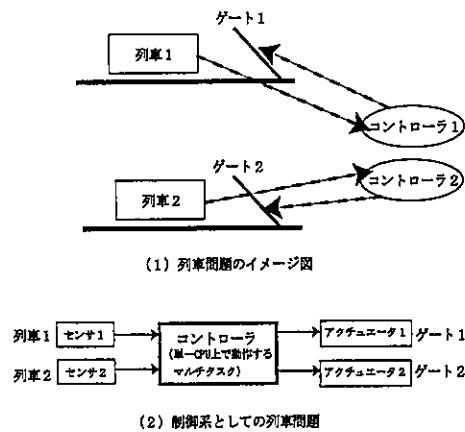


図 26: 列車問題

### 5.2.2 上流工程の仕様記述

次に、図 26 の列車問題の上流工程の仕様をハイブリッドオートマトンで仕様記述する。

まず、Alur らの線形ハイブリッドオートマトン [62] を拡張した線形ハイブリッドオートマトンを定義する。この線形ハイブリッドオートマトンは観測可能な変数と観測不能な変数を区別するものである。

#### Definition 25 (線形ハイブリッドオートマトン)

線形ハイブリッドオートマトンは、 $A = (\vec{x}, V, local, inv, dif, E, act, Label, syn, E_{\theta})$  の 10 組で定義される。ここで、

## 1. データ変数

有限なベクタ  $\vec{x} = (x_1, x_2, \dots, x_n)$  は実数値のデータ変数である。  $\vec{x}$  のサイズ  $n$  は  $A$  の次元と呼ばれる。

データの状態は  $n$ -次元の実数空間  $R^n$  の点  $\vec{s} = (s_1, \dots, s_n)$  であり、各データ変数  $x_i$  に実数値  $s_i \in R$  を割り付ける関数である。凸状のデータの領域は  $R^n$  の凸状の多面体である。データの領域は凸状のデータの領域の有限な和集合である。凸状のデータの述語は  $\vec{x}$  上の凸状の線形な論理式である。凸状のデータの述語  $p$  は凸状のデータの領域  $\ll p \gg \subseteq R^n$  を定義する。ここで、  $p[\vec{x} := \vec{s}]$  が *true* のときに限り  $\vec{s} \in \ll p \gg$  である。

任意のデータ変数  $x_i$  に対して、  $x_i$  の第一階導関数を  $\dot{x}_i$  と表現する。第一階導関数の包含は  $R^n$  中の凸状の多面体である。速度の述語は  $\vec{x}$  上の凸状の線形な論理式である。速度の述語  $r$  は第一階導関数の包含  $\ll r \gg \subseteq R^n$  を定義する。ここで、  $r[\vec{x} := \vec{s}]$  が *true* のときに限り  $\vec{s} \in \ll r \gg$  である。

任意のデータ変数  $x_i$  に対して、状態遷移後の  $x_i$  の新しい値を  $x_i'$  と表記する。アクションの述語  $q = (\vec{y}, q')$  は、更新された変数の集合  $\vec{y} \subseteq \vec{x}$  とその更新された変数のデータの述語である凸状の線形な論理式  $q'$  から構成される。アクションの述語  $q$  の閉包  $\{q\}$  は凸状の線形な論理式  $q' \wedge \bigwedge_{x \in \vec{x} \setminus \vec{y}} (x' = x)$  である。すなわち、更新されないデータ変数は変化しないままである。アクションの述語  $q$  は状態変数から凸状のデータの領域への関数  $\ll q \gg$  である：すべてのデータの状態  $\vec{s}, \vec{s}' \in R^n$  に対して、  $\{q\}[\vec{x}, \vec{x}' := \vec{s}, \vec{s}']$  が *true* のときに限り  $\vec{s} \in \ll q \gg(\vec{s}')$  である。もしデータの領域  $\ll q \gg(\vec{s})$  が非空ならば、アクションの述語  $q$  はデータの状態の中で実行可能である。

## 2. 制御ロケーション

ノードの有限集合  $V$  は制御ロケーションである。

線形ハイブリッドオートマトン  $A$  の状態  $(v, \vec{s})$  は制御ロケーション  $v \in V$  とデータの状態  $\vec{s} \in R^n$  から構成される。領域  $R = \bigcup_{v \in V} (v, S_v)$  は各制御ロケーションのデータの領域  $S_v \subseteq R^n$  を集めたものである。  $A$  の状態の述語  $\phi = \bigcup_{v \in V} (v, p_v)$  は各制御ロケーションのデータの述語  $p_v$  を集めたものである。  $A$  の状態の述語  $\phi$  は領域  $\ll \phi \gg = \bigcup_{v \in V} (v, \ll p_v \gg)$  を定義する。  $R$  を定義する状態の述語が存在するならば、領域  $R$  は線形である。

領域  $(v, S) \cup \bigcup_{v' \neq v} (v', \emptyset)$  を  $(v, S)$  と書き、  $(v, p) \cup \bigcup_{v' \neq v} (v', false)$  を  $(v, p)$  と書く。状態の述語を書くとき、制御ロケーションの集合  $V$  上の範囲をとるロケーションカウンタ  $l$  を使う。ロケーション制約  $l = v$  は状態の述語  $(v, true)$  を示す。状態の述語としてデータの述語を使うとき、データの述語  $p$  は  $\bigcup_{v \in V} (v, p)$  を示す。2つの状態の述語  $\phi = \bigcup_{v \in V} (v, p_v)$  と  $\phi' = \bigcup_{v \in V} (v, p_v')$  に対して、  $\neg \phi = \bigcup_{v \in V} (v, \neg p_v)$ ,  $(\phi \vee \phi') = \bigcup_{v \in V} (v, p_v \vee p_v')$ ,  $(\phi \wedge \phi') = \bigcup_{v \in V} (v, p_v \wedge p_v')$  である。

## 3. ローカル変数

*local* はローカル変数の有限集合であり、制御ロケーション  $V$  と有限なベクタ  $\vec{x}$  の中で観測でき

ない要素から構成される。ローカル変数は他のプロセスから観測できない変数である。

#### 4. ロケーション不変性

各制御ロケーション  $v \in V$  に凸状のデータの述語  $inv(v)$ , つまり  $v$  の不変式を割り付ける関数である。オートマトン  $A$  の制御は不変式  $inv(v)$  が  $true$  である限り  $v$  に留まるので, 不変式  $inv(v)$  は一つのロケーションから別のロケーションへのシステムの前進を強制するために使われる。  $\vec{s} \in \ll inv(v) \gg$  ならば状態  $(v, \vec{s})$  は存在しえる。  $A$  の存在しえる状態を  $\Sigma_A$  として, その状態の述語を  $\bigcup_{v \in V} (v, inv(v))$  とする。

#### 5. 連続的アクティビティ

各制御ロケーション  $v \in V$  に速度の述語  $dif(v)$ , つまり  $v$  のアクティビティを割り付ける関数は  $dif$  である。アクティビティはデータ変数の値が変化する速度を制約する。つまり, オートマトンの制御がロケーション  $v$  に存在するとき, すべてのデータ変数の第一導関数は導関数の包含  $\ll dif(v) \gg$  内に存在する。

#### 6. 遷移

エッジの有限多重集合  $E$  は遷移と呼ばれる。各遷移  $(v, v')$  は遷移元ロケーション  $v \in V$  と遷移先ロケーション  $v' \in V$  を示す。各ロケーション  $v \in V$  に対して, 遷移  $e_v = (v, v)$  が存在する。

#### 7. 離散的アクション

各遷移  $e \in E$  にアクションの述語  $act(e)$  を割り付けるラベリング関数  $act$  は  $e$  のアクションである。アクション  $act(e)$  が実行可能のときにのみ, オートマトンの制御は  $e = (v, v')$  により, ロケーション  $v$  からロケーション  $v'$  へ進む。もし  $act(e)$  がデータの状態  $\vec{s}$  において実行可能ならば, すべてのデータ変数の値は非決定的に  $\vec{s}$  からデータの領域  $\ll act(e) \gg (\vec{s})$  のある点に変化する。

#### 8. 同期ラベルとラベリング関数

同期ラベルの有限集合は  $Label$  であり, 各遷移  $e \in E$  に同期ラベルの部分集合を割り付けるラベリング関数は  $syn$  である。集合  $Label$  は  $A$  のアルファベットと呼ばれる。同期ラベルは2つのオートマトンの並列合成を定義するために使われる。つまり, 2つのオートマトンが同期ラベル  $a$  を共有するならば, 一方のオートマトンの  $a$ -遷移が他方のオートマトンの  $a$ -遷移によって遂行されなければならない。

#### 9. エントリーエッジ

エントリーエッジ  $E_\Theta$  は遷移元のロケーションは存在しないが, エントリーロケーション  $v_i \in V$  がある。  $E_\Theta$  は  $x_1 = c_1 \wedge x_2 = c_2 \wedge \dots \wedge x_n = c_n$  によりラベル付けされている。ただし,  $c_1, c_2, \dots, c_n$  は実数値である。

■

次に, ハイブリッドオートマトンの並列合成を定義する。

**Definition 26 (線形ハイブリッドオートマトンの並列合成)**

2つの線形ハイブリッドオートマトン  $A_1 = (\bar{x}_1, V_1, local_1, inv_1, dif_1, E_1, act_1, Label_1, syn_1, E_{\Theta_1})$  と  $A_2 = (\bar{x}_2, V_2, local_2, inv_2, dif_2, E_2, act_2, Label_2, syn_2, E_{\Theta_2})$  が与えられたとする。ここで、 $A_1$  と  $A_2$  の次元は各々  $n_1$  と  $n_2$  である。 $A_1$  と  $A_2$  の並列合成は、 $A = (\bar{x}_1 \cup \bar{x}_2, V_1 \times V_2, local_1 \cup local_2, inv, dif, E, act, Label_1 \cup Label_2, syn, E_{\Theta})$  である。

1.  $V_1 \times V_2$  の各ロケーション  $(v, v')$  は不変式  $inv(v, v') = inv_1(v) \wedge inv_2(v')$  とアクティビティ  $dif(v, v') = dif_1(v) \wedge dif_2(v')$  を持つ。
2.  $E_{\Theta} = E_{\Theta_1} \wedge E_{\Theta_2}$  である。すなわち、エントリーロケーション  $(v_{1i}, v_{2i})$  である。ここで、 $v_{1i} \in V_1$  は  $A_1$  のエントリーロケーションであり、 $v_{2i} \in V_2$  は  $A_2$  のエントリーロケーションである。また、 $E_{\Theta}$  は  $(x_{11} = c_{11} \wedge \dots \wedge x_{1n_1} = c_{1n_1}) \wedge (x_{21} = c_{21} \wedge \dots \wedge x_{2n_2} = c_{2n_2})$  によりラベル付けされている。ただし、 $\bar{x}_1 = (x_{11}, x_{12}, \dots, x_{1n_1})$ ,  $\bar{x}_2 = (x_{21}, x_{22}, \dots, x_{2n_2})$ ,  $c_{11}, \dots, c_{1n_1}, c_{21}, \dots, c_{2n_2}$  は実数値である。
3. 以下のいずれかのときに限り、 $E$  は  $e = ((v_1, v_1'), (v_2, v_2'))$  を含む：
  - (a)  $v_1 = v_1'$ , かつ、 $Label_1 \cap syn_2(e_2) = \emptyset$  を有する  $e_2 = (v_2, v_2') \in E_2$  が存在する。  
このとき、 $act(e) = act_2(e_2)$  かつ  $syn(e) = syn_2(e_2)$  である。
  - (b)  $Label_1 \cap syn_2(e_2) = \emptyset$  を持つ  $e_2 = (v_2, v_2') \in E_2$  が存在して、かつ、 $v_2 = v_2'$  である。  
このとき、 $act(e) = act_1(e_1)$  かつ  $syn(e) = syn_1(e_1)$  である。
  - (c)  $syn_1(e_1) \cap Label_2 = syn_2(e_2) \cap Label_1$  を持つ  $e_1 = (v_1, v_1') \in E_1$  と  $e_2 = (v_2, v_2') \in E_2$  が存在する。  
このとき、 $act_1(e_1) = (\bar{y}_1, q_1')$  かつ  $act_2(e_2) = (\bar{y}_2, q_2')$  ならば、 $act(e) = (\bar{y}_1 \cup \bar{y}_2, q_1' \wedge q_2')$  かつ  $syn(e) = syn_1(e_1) \cup syn_2(e_2)$  である。

■

次に、列車問題の上流工程の仕様をハイブリッドオートマトンで仕様記述する。図 27 では、2つの線路のゲートが互いに独立な2つのコントローラで制御されている。なお、図 27 はハイブリッドオートマトンの視覚的表現であり、形式的には Definition 1 で表現できる。

以下に、図 27 の列車 1, ゲート 1, コントローラ 1 の動作を簡単に説明する。変数  $x_1$  はゲート 1 から列車 1 までの距離を表す。 $\dot{x}_1$  は  $x_1$  の一階導関数であり、列車 1 の速度を表す。最初、列車 1 はゲート 1 から離れており、48メートル/秒と 52メートル/秒の間の速度で動作する。次に、列車 1 がゲート 1 に接近するとき、踏み切りから距離 1000メートルに置かれているセンサが列車 1 を検知して、コントローラ 1 に信号  $app_1$  を送信する。それから、列車 1 は  $near_1$  になり、40 と 52 の間の速度で移動する。コントローラ 1 が  $idle_1$  で信号  $app_1$  を受信するならば、5時刻以内にゲート 1 に信号  $lower_1$  を送信する。ここで、コントローラ 1 の遅延は変数  $z_1$  で計測する。もしゲート 1 が開くならば、速度 20度により、90度から 0度へ下がる。ここで、ゲート 1 の位置は変数  $y_1$  で表現する。踏み切りから 100 過

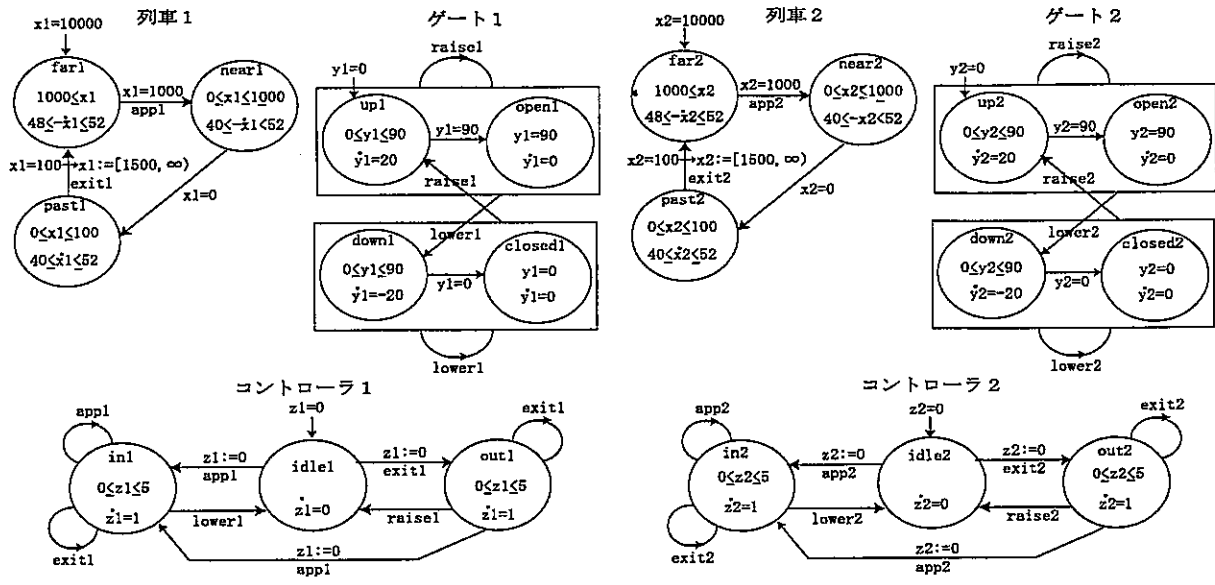


図 27: 列車問題のハイブリッドオートマトン

きたところにある第2のセンサはコントローラ1に信号  $exit_1$  を送信して、5時刻以内にゲート1に信号  $raise_1$  を送信する。我々は、列車間の距離は1500と仮定する。

列車2、ゲート2、コントローラ2の動作も同様である。

以下に、図27の視覚的表現のコントローラ1とコントローラ2のハイブリッドオートマトンを形式的に定義する。

まず、図27の視覚的表現のコントローラ1のハイブリッドオートマトンを

$$A_1 = (\bar{x}_1, V_1, local_1, inv_1, dif_1, E_1, act_1, Label_1, syn_1, E_{\Theta 1})$$

で形式的に定義する。

1.  $\bar{x}_1 = (z_1)$ .
2.  $V_1 = \{in_1, idle_1, out_1\}$ .
3.  $local_1 = \{\pi_1\}$ . ただし、 $\pi_1$  はロケーションを表す変数である。
4.  $inv_1(in_1) \equiv 0 \leq z_1 \leq 5$ ,  $inv_1(idle_1) \equiv true$ ,  $inv_1(out_1) \equiv 0 \leq z_1 \leq 5$
5.  $dif_1(in_1) \equiv z_1 = 1$ ,  $dif_1(idle_1) \equiv z_1 = 0$ ,  $dif_1(out_1) \equiv z_1 = 1$
6.  $E_1 = \{(in_1, in_1), (in_1, idle_1), (idle_1, in_1), (idle_1, out_1), (out_1, idle_1), (out_1, out_1), (out_1, in_1)\}$
7.  $act_1((idle_1, in_1)) \equiv z_1' = 0$ ,  $act_1((idle_1, out_1)) \equiv z_1' = 0$ ,  $act_1((out_1, in_1)) \equiv z_1' = 0$ . ただし、 $z_1'$  は状態遷移後の  $z_1$  の値を意味する。

8.  $Label_1 = \{app_1, lower_1, exit_1, raise_1\}$ .
9.  $syn_1((in_1, in_1)) = app_1, syn_1((in_1, in_1)) = exit_1, syn_1((in_1, idle_1)) = lower_1, syn_1((idle_1, in_1)) = app_1, syn_1((idle_1, out_1)) = exit_1, syn_1((out_1, idle_1)) = raise_1, syn_1((out_1, out_1)) = exit_1, syn_1((out_1, in_1)) = app_1$
10.  $E_{\Theta_1}$  はエントリーロケーション  $idle_1$  とラベル  $z_1 = 0$  で定義される。

次に、図 2 7 の視覚的表現のコントローラ 2 のハイブリッドオートマトンを

$$A_2 = (\vec{x}_2, V_2, local_2, inv_2, dif_2, E_2, act_2, Label_2, syn_2, E_{\Theta_2})$$

で形式的に定義する。

1.  $\vec{x}_2 = (z_2)$ .
2.  $V_2 = \{in_2, idle_2, out_2\}$ .
3.  $local_2 = \{\pi_2\}$ . ただし、 $\pi_2$  はロケーションを表す変数である。
4.  $inv_2(in_2) \equiv 0 \leq z_2 \leq 5, inv_2(idle_2) \equiv true, inv_2(out_2) \equiv 0 \leq z_2 \leq 5$ .
5.  $dif_2(in_2) \equiv \dot{z}_2 = 1, dif_2(idle_2) \equiv \dot{z}_2 = 0, dif_2(out_2) \equiv \dot{z}_2 = 1$
6.  $E_2 = \{(in_2, in_2), (in_2, in_2), (in_2, idle_2), (idle_2, in_2), (idle_2, out_2), (out_2, idle_2), (out_2, out_2), (out_2, in_2)\}$
7.  $act_2((idle_2, in_2)) \equiv z_2' = 0, act_2((idle_2, out_2)) \equiv z_2' = 0, act_2((out_2, in_2)) \equiv z_2' = 0$ .
8.  $Label_2 = \{app_2, lower_2, exit_2, raise_2\}$ .
9.  $syn_2((in_2, in_2)) = app_2, syn_2((in_2, in_2)) = exit_2, syn_2((in_2, idle_2)) = lower_2, syn_2((idle_2, in_2)) = app_2, syn_2((idle_2, out_2)) = exit_2, syn_2((out_2, idle_2)) = raise_2, syn_2((out_2, out_2)) = exit_2, syn_2((out_2, in_2)) = app_2$
10.  $E_{\Theta_2}$  はエントリーロケーション  $idle_2$  とラベル  $z_2 = 0$  で定義される。

本論文では、コントローラを設計の対象とする。上流工程のコントローラのハイブリッドオートマトンの仕様はコントローラのハイブリッドオートマトンの並列合成であり、図 2 8 に視覚的表現を示す。

次に、図 2 8 の視覚的表現のコントローラ 1 とコントローラ 2 の並列ハイブリッドオートマトンを  $A = (\vec{x}, V, local, inv, dif, E, act, Label, syn, E_{\Theta})$  で形式的に定義する。2つの線形ハイブリッドオートマトン  $A_1 = (\vec{x}_1, V_1, local_1, inv_1, dif_1, E_1, act_1, Label_1, syn_1, E_{\Theta_1})$  と  $A_2 = (\vec{x}_2, V_2, local_2, inv_2, dif_2, E_2, act_2, Label_2, syn_2, E_{\Theta_2})$  が与えられている。  $A_1$  と  $A_2$  の並列合成は、  $A = (\vec{x}_1 \cup \vec{x}_2, V_1 \times V_2, local_1 \cup local_2, inv, dif, E, act, Label_1 \cup Label_2, syn, E_{\Theta})$  である。

ただし、各項は以下のとおりである：

1.  $\vec{x}_1 \cup \vec{x}_2 = (z_1, z_2)$ .
2.  $V_1 \times V_2 = \{(in_1, in_2), (in_1, idle_2), (in_1, out_2), (idle_1, in_2), (idle_1, idle_2), (idle_1, out_2)\}$ ,

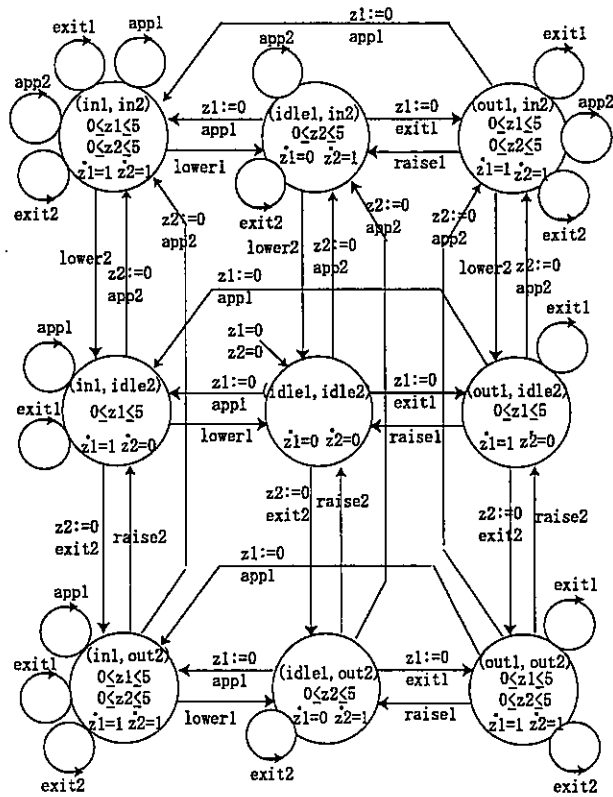


図 28: 上流工程の検証対象の仕様

- $(out_1, in_2), (out_1, idle_2), (out_1, out_2)\}$ .
3.  $local_1 \cup local_2 = \{\pi_1, \pi_2\}$ .
  4.  $inv((in_1, in_2)) \equiv 0 \leq z_1 \leq 5 \wedge 0 \leq z_2 \leq 5, inv((in_1, idle_2)) \equiv 0 \leq z_1 \leq 5, inv((in_1, out_2)) \equiv 0 \leq z_1 \leq 5 \wedge 0 \leq z_2 \leq 5, inv((idle_1, in_2)) \equiv 0 \leq z_2 \leq 5, inv((idle_1, idle_2)) \equiv z_1 = 0 \wedge z_2 = 0, inv((idle_1, out_2)) \equiv 0 \leq z_2 \leq 5, inv((out_1, in_2)) \equiv 0 \leq z_1 \leq 5 \wedge 0 \leq z_2 \leq 5, inv((out_1, idle_2)) \equiv 0 \leq z_1 \leq 5, inv((out_1, out_2)) \equiv 0 \leq z_1 \leq 5 \wedge 0 \leq z_2 \leq 5.$
  5.  $dif((in_1, in_2)) \equiv \dot{z}_1 = 1 \wedge \dot{z}_2 = 1, dif((in_1, idle_2)) \equiv \dot{z}_1 = 1 \wedge \dot{z}_2 = 0, dif((in_1, out_2)) \equiv \dot{z}_1 = 1 \wedge \dot{z}_2 = 1, dif((idle_1, in_2)) \equiv \dot{z}_1 = 0 \wedge \dot{z}_2 = 1, dif((idle_1, idle_2)) \equiv \dot{z}_1 = 0 \wedge \dot{z}_2 = 0, dif((idle_1, out_2)) \equiv \dot{z}_1 = 0 \wedge \dot{z}_2 = 1, dif((out_1, in_2)) \equiv \dot{z}_1 = 1 \wedge \dot{z}_2 = 1, dif((out_1, idle_2)) \equiv \dot{z}_1 = 1 \wedge \dot{z}_2 = 0, dif((out_1, out_2)) \equiv \dot{z}_1 = 1 \wedge \dot{z}_2 = 1.$
  6.  $E = \{((idle_1, idle_2), (in_1, idle_2)), ((in_1, idle_2), (idle_1, idle_2)), ((in_1, idle_2), (in_1, idle_2)), ((in_1, idle_2), (in_1, idle_2)), ((idle_1, idle_2), (idle_1, in_2)), ((idle_1, in_2), (idle_1, idle_2)), \dots\}$ .
    - (a)  $act(((idle_1, idle_2), (in_1, idle_2))) \equiv z_1' = 0, syn(((idle_1, idle_2), (in_1, idle_2))) = app_1$
    - (b)  $syn(((in_1, idle_2), (idle_1, idle_2))) = lower_1$
    - (c)  $syn(((in_1, idle_2), (in_1, idle_2))) = app_1$
    - (d)  $syn(((in_1, idle_2), (in_1, idle_2))) = exit_1$
    - (e)  $act(((idle_1, idle_2), (idle_1, in_2))) \equiv z_2' = 0, syn(((idle_1, idle_2), (idle_1, in_2))) = app_2$
    - (f)  $syn(((idle_1, in_2), (idle_1, idle_2))) = lower_2$
  7.  $Label_1 \cup Label_2 = \{app_1, lower_1, exit_1, raise_1, app_2, lower_2, exit_2, raise_2\}$ .
  8.  $E_\Theta$  はエンタリーロケーション  $(idle_1, idle_2)$  とラベル  $z_1 = 0 \wedge z_2 = 0$  で定義される。

### 5.2.3 中流工程への段階的詳細化開発

次に、図 2 7 の複数の線路のゲートが単一の CPU 上のコントローラで制御される場合のソフトウェアモデルを設計する。コントローラのタスクは非周期的に起動されて相互に独立しており、スケジューリングはオフライン型の静的なプリエンプティブとする。図 2 8 の上流工程の仕様を詳細化して、図 2 9 の中流工程のソフトウェアモデルを設計する。図 2 9 のソフトウェアモデルでは、図 2 7 のコントローラ 1 とコントローラ 2 の仕様が各々コントローラ 1 とコントローラ 2 のタスクになり、スケジューラでスケジューリングされると考える。なお、3 つのタスクはすべてハイブリッドオートマトンで表現する。つまり、本節における詳細化とは、中流工程において、新たにスケジューラというタスクを追加して仕様をスケジュールするものである。図 2 9 では、ゲート 1 に関する処理が優先度高く

て、ゲート2の処理中にプリエンプトされるスケジューリングを考えている。また、列車とゲートはセンサとアクチュエータに対応しており、コントローラのみがソフトウェア作成の対象なので、列車とゲートは上流工程と中流工程においては同じであるとする。

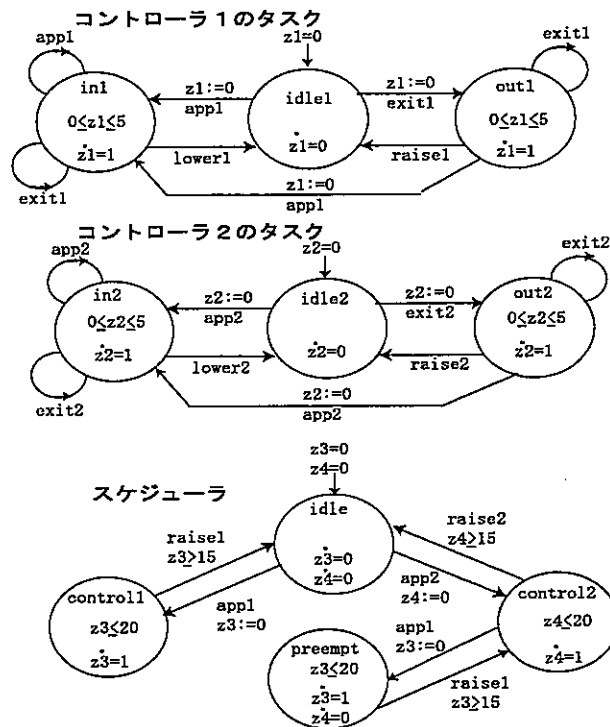


図 29: 中流工程のソフトウェアモデルのタスクとスケジューラ

図 29 の中流工程のソフトウェアモデルのタスクとスケジューラは並列に動作するので、並列なハイブリッドオートマトンを考える必要がある。以下の図 30 に、ソフトウェアモデルの並列ハイブリッドオートマトンを示す。ただし、紙面の都合上から、状態名と状態遷移のみを示す。

次に、図 29 の視覚的表現のコントローラ 1、コントローラ 2 とスケジューラのタスクの並列ハイブリッドオートマトンを  $A = (\bar{x}, V, local, inv, dif, E, act, Label, syn, E_{\Theta})$  で形式的に定義する。3 つの線形ハイブリッドオートマトン  $A_1 = (\bar{x}_1, V_1, local_1, inv_1, dif_1, E_1, act_1, Label_1, syn_1, E_{\Theta_1})$ ,  $A_2 = (\bar{x}_2, V_2, local_2, inv_2, dif_2, E_2, act_2, Label_2, syn_2, E_{\Theta_2})$ ,  $A_3 = (\bar{x}_3, V_3, local_3, inv_3, dif_3, E_3, act_3, Label_3, syn_3, E_{\Theta_3})$  が与えられている。ただし、 $A_1$  はコントローラ 1 のタスク、 $A_2$  はコントローラ 2 のタスク、 $A_3$  はスケジューラのタスクである。 $A_1$ ,  $A_2$  と  $A_3$  の並列合成は、 $A = (\bar{x}_1 \cup \bar{x}_2 \cup \bar{x}_3, V_1 \times V_2 \times V_3, local_1 \cup local_2 \cup local_3, inv, dif, E, act, Label_1 \cup Label_2 \cup Label_3, syn, E_{\Theta})$  である。

ただし、各項は以下のとおりである：

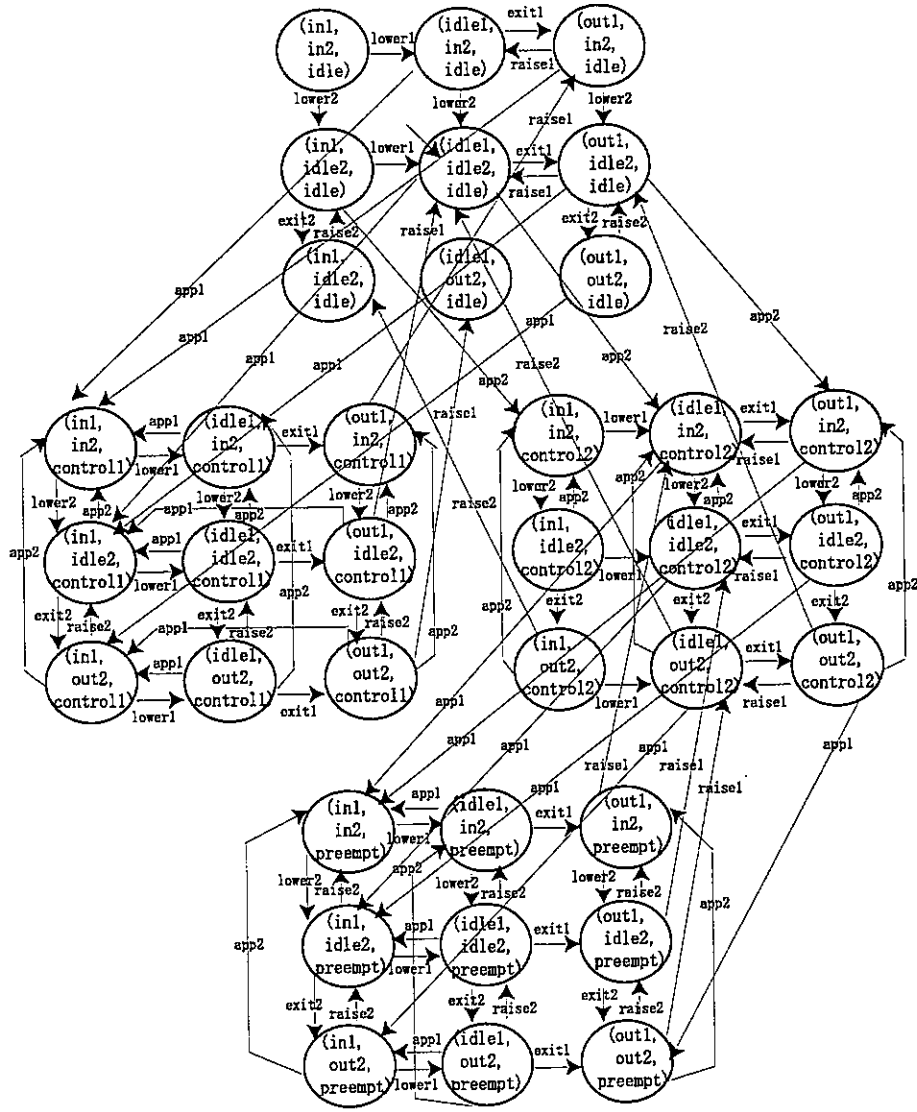


図 30: 中流工程の並列ハイブリッドオートマトン

1.  $\bar{x}_1 \cup \bar{x}_2 \cup \bar{x}_3 = (z_1, z_2, z_3, z_4)$ .
2.  $V_1 \times V_2 \times V_3 = \{(idle_1, idle_2, idle),$   
 $(idle_1, idle_2, control_1), (idle_1, idle_2, control_2),$   
 $(idle_1, idle_2, preempt), (idle_1, in_2, idle),$   
 $\dots\}$ .
3.  $local_1 \cup local_2 \cup local_3 = \{\pi_1, \pi_2, \pi_3, z_3, z_4\}$ .
4.  $inv((idle_1, idle_2, idle)) \equiv 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4,$   
 $inv((idle_1, idle_2, control_1)) \equiv 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4,$   
 $inv((idle_1, idle_2, control_2)) \equiv 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4 \leq 20,$   
 $inv((idle_1, idle_2, preempt)) \equiv 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4,$   
 $inv((idle_1, in_2, idle)) \equiv 0 \leq z_1 \wedge 0 \leq z_2 \leq 5 \wedge 0 \leq z_3 \wedge 0 \leq z_4,$   
 $\dots,$   
 $\dots$
5.  $dif((idle_1, idle_2, idle)) \equiv z_1 = 0 \wedge z_2 = 0 \wedge z_3 = 0 \wedge z_4 = 0,$   
 $dif((idle_1, idle_2, control_1)) \equiv z_1 = 0 \wedge z_2 = 0 \wedge z_3 = 1 \wedge z_4 = 0,$   
 $dif((idle_1, idle_2, control_2)) \equiv z_1 = 0 \wedge z_2 = 0 \wedge z_3 = 0 \wedge z_4 = 1,$   
 $dif((idle_1, idle_2, preempt)) \equiv z_1 = 0 \wedge z_2 = 0 \wedge z_3 = 1 \wedge z_4 = 0,$   
 $dif((idle_1, in_2, idle)) \equiv z_1 = 0 \wedge z_2 = 1 \wedge z_3 = 0 \wedge z_4 = 0,$   
 $\dots,$   
 $\dots$
6.  $E = \{((idle_1, idle_2, idle), (idle_1, idle_2, control_1)),$   
 $((idle_1, idle_2, control_1), (idle_1, idle_2, idle)),$   
 $((idle_1, idle_2, idle), (idle_1, idle_2, control_2)),$   
 $((idle_1, idle_2, control_2), (idle_1, idle_2, idle)),$   
 $((idle_1, idle_2, control_2), (idle_1, idle_2, preempt)),$   
 $((idle_1, idle_2, preempt), (idle_1, idle_2, control_2)),$   
 $\dots,$   
 $\dots\}$ .  
  - (a)  $act(((idle_1, idle_2, idle), (idle_1, idle_2, control_1)))$   
 $\equiv z_3' = 0,$   
 $syn(((idle_1, idle_2, idle), (idle_1, idle_2, control_1)))$   
 $= app_1$
  - (b)  $act(((idle_1, idle_2, control_1), (idle_1, idle_2, idle)))$   
 $\equiv z_3' \geq 15,$   
 $syn(((idle_1, idle_2, control_1), (idle_1, idle_2, idle)))$

- $= raise_1$
- (c)  $act(((idle_1, idle_2, idle), (idle_1, idle_2, control_2)))$   
 $\equiv z_4' = 0,$   
 $syn(((idle_1, idle_2, idle), (idle_1, idle_2, control_2)))$   
 $= app_2$
- (d)  $act(((idle_1, idle_2, control_2), (idle_1, idle_2, idle)))$   
 $\equiv z_4' \geq 15,$   
 $syn(((idle_1, idle_2, control_2), (idle_1, idle_2, idle)))$   
 $= raise_2$
- (e)  $act(((idle_1, idle_2, control_2),$   
 $(idle_1, idle_2, preempt))) \equiv z_3' = 0,$   
 $syn(((idle_1, idle_2, control_2),$   
 $(idle_1, idle_2, preempt))) = app_1$
- (f)  $act(((idle_1, idle_2, preempt),$   
 $(idle_1, idle_2, control_2))) \equiv z_3' \geq 15,$   
 $syn(((idle_1, idle_2, preempt),$   
 $(idle_1, idle_2, control_2))) = raise_1$   
 $\dots\dots\dots,$   
 $\dots\dots\dots$

7.  $Label_1 \cup Label_2 \cup Label_3 = \{app_1, lower_1,$   
 $exit_1, raise_1, app_2, lower_2, exit_2, raise_2\}.$

8.  $E_{\Theta}$  はエン트리ロケーション  $(idle_1, idle_2, idle)$  とラベル  $z_1 = 0 \wedge z_2 = 0 \wedge z_3 = 0 \wedge z_4 = 0$  で定義される.

### 5.3 上流工程から中流工程への詳細化検証

本章では、詳細化検証問題を定義して、詳細化検証手法を提案する.

本節では、図 28 の上流工程のコントローラのハイブリッドオートマトンの並列合成が図 30 の中流工程のコントローラのソフトウェアモデルのハイブリッドオートマトンによって正しく詳細化されているかどうかを検証する.

#### 5.3.1 詳細化検証手法

本論文における詳細化検証では、ハイブリッドオートマトンをフェーズ遷移システムに変換して、詳細化検証の公理系により行う.

まず, システム変数の有限集合  $Var$  を考える. システム変数は整数や実数などの型を持つ. ここで,  $R^n$  を実数の集合とする. 状態は  $\sigma : Var \rightarrow R^n$  であり,  $Var$  の中の変数の型整合解釈である. 状態の集合を  $\Sigma_V$  とする. なお, 変数の型としては整数やブール値などがあるが, いずれも実数の部分集合なので, 状態を  $\sigma : Var \rightarrow R^n$  とする.

時間は非負実数  $R^+$  によってモデル化される. 区間  $I = [a, b)$  は  $a \leq t < b$  であるような点  $t \in R^+$  の集合である. ここで,  $a, b \in R^+$  かつ  $a < b$  である.  $I = [a, b)$  を区間とする. 関数  $f : I \rightarrow R^n$  は, 以下の条件を満たすならば,  $I$  上で各部分がスムーズである:

1.  $a$  において,  $f$  の右極限とすべての右導関数が存在する.
2. すべての点  $t \in I$  において,  $f$  の右極限と左極限, すべての右導関数と左導関数が存在して,  $f$  は右または左から連続である.
3.  $b$  において,  $f$  の左極限とすべての左導関数が存在する.

2つの関数  $f, g : I \rightarrow R^n$  は有限の点以外のすべてにおいて一致するならば,  $I$  上において区別不能である. もし, 各部分がスムーズな2つの関数が开区間  $I$  上で区別不能ならば, 2つの関数は  $I$  における, すべての極限と導関数において一致する.

変数  $V$  上のフェーズ  $P = \langle I, f \rangle$  は以下の順序対から構成される:

1. 区間  $I = [a, b)$  である.
2. 関数  $f_x : I \rightarrow R^n$  の型整合族は  $f = \{f_x | x \in Var\}$  である. なお, 関数  $f_x : I \rightarrow R^n$  は区間  $I = [a, b)$  上で各部分がスムーズであり, 各点  $t \in I$  に変数  $x \in Var$  の値を割り付ける.

フェーズ  $P$  はすべての実数値の時間  $t \in I$  に状態  $f(t) \in \Sigma_V$  を割り付ける.

以下では, フェーズ  $P$  の左終端の極限状態  $\vec{P} \in \Sigma_V$  を

$$\vec{P} = \lim_{t \rightarrow a} \{f(t) | a < t < b\} \quad (1)$$

と書き, フェーズ  $P$  の右終端の極限状態  $\overleftarrow{P} \in \Sigma_V$  を

$$\overleftarrow{P} = \lim_{t \rightarrow b} \{f(t) | a < t < b\} \quad (2)$$

と書く.

次に, Pnueli らのフェーズ遷移システム [65] を拡張した, フェーズ遷移システムを定義する. このフェーズ遷移システムは観測可能な変数と観測不能な変数を区別するものである.

### Definition 27 (フェーズ遷移システム)

フェーズ遷移システムは,  $M = (Var, L, \Phi, \Theta, T)$  の5つ組で定義される. ここで,

1.  $Var$  はシステム変数の有限集合である.
2.  $L \subseteq Var$  はローカル変数であり, システムの外部からは観測できない変数である.

3.  $\Phi$  は  $Var$  上のフェーズの不変式の有限集合である。各フェーズ不変式  $\phi \in \Phi$  は表明式  $\rho_\phi(Var, \dot{Var})$  によって表現される。ここで、 $\dot{Var}$  はシステム変数  $Var$  の導関数である。
4.  $\Theta$  は初期条件である。これは、すべての初期状態を特徴付ける表明である。
5.  $T$  は状態遷移の有限集合である。各状態遷移  $\tau \in T$  は表明式  $\rho_\tau(Var, Var')$  によって表現される。

■

$\bar{P} = P_0, P_1, P_2, \dots$  は無限なフェーズ列である。ここで、すべての  $i \geq 0$  に対して、 $P_i = \langle [a_i, a_{i+1}), f_i \rangle$  である。

フェーズ列が以下の条件を満たすフェーズ列  $\bar{P} = P_0, P_1, P_2, \dots$  と等しければ、このフェーズ列はフェーズ遷移システム  $M$  の計算である：

1. 初期条件：もし  $P_0 = [a, b)$  ならば  $\Theta$  は  $a$  で成り立つ。
2. 連続動作：すべての  $0 \leq i < |\bar{P}|$  に対して、 $P_i$  が  $\phi$ -フェーズであるようなフェーズ不変式  $\rho_\phi \in \Phi$  が存在する。
3. 離散的状態遷移：すべての  $0 \leq i < |\bar{P}| - 1$  に対して、 $\rho_\tau(\overrightarrow{P}_i[Var], \overrightarrow{P}_{i+1}[Var])$  が成り立つような状態遷移  $\tau \in T$  が存在する。
4. 発散性： $\bar{P}$  は発散する。

次に、ハイブリッドオートマトンからフェーズ遷移システムへの変換方法を定義する。

### Definition 28 (ハイブリッドオートマトンの変換方法)

線形ハイブリッドオートマトン  $A = (\bar{x}, V, local, inv, dif, E, act, Label, syn, E_\Theta)$  は、以下のように、フェーズ遷移システム  $M = (Var, L, \Phi, \Theta, T)$  に変換される：

1.  $Var = \pi \cup \bar{x}$   
ただし、 $\pi$  は制御ロケーション  $V$  を表現するロケーション変数である。つまり、システム変数の有限集合  $Var$  はロケーション変数  $\pi$  とデータ変数  $\bar{x}$  である。
2.  $L = local$   
つまり、ローカル変数  $L$  はハイブリッドオートマトンのローカル変数  $local$  であり、 $local \subseteq Var$  である。
3.  $\Phi = \{\phi_{l_i} | l_i \in V\}$   
ここで、任意の  $l_i \in V$  に対して、  
$$\rho_{\phi_{l_i}} : inv(l_i) \wedge (\pi = l_i) \wedge dif(l_i)$$
  
である。
4.  $\Theta = (x_1 = c_1 \wedge \dots \wedge x_n = c_n) \wedge (\pi = l_i) \wedge inv(l_i)$   
ここで、 $l_i$  はエン트리ロケーションであり、 $(x_1 = c_1 \wedge \dots \wedge x_n = c_n)$  はエントリーエッジのラベルである。

$$5. T = \{\tau_{(l_i, l_j)} \mid (l_i, l_j) \in E\}$$

ここで,  $e = (l_i, l_j) \in E$  に対して,

$$\rho_{\tau_{(l_i, l_j)}} : act(e) \wedge \pi = l_i \wedge \pi' = l_j$$

である.

なお, 同期ラベルの集合  $Label$  とラベリング関数  $syn$  は状態遷移の名前  $label \in Label$  を定義するものであり,  $syn(e) = label$  である.

次に, 詳細化検証の基本概念を定義する. 図 31 の (1) と (2) のように, 抽象化仕様及び具体化仕様が与えられたとする. ハイブリッドシステムの詳細化検証では, フェーズ  $\rho_\phi(Var, \dot{Var})$  と状態遷移  $\rho_\tau(Var, Var')$  の詳細化を考慮する必要がある. 本論文では, 具体化仕様が抽象化仕様を詳細化するには, 抽象化仕様のフェーズが具体化仕様のフェーズを包含して, すべての具体化仕様の状態遷移に対して, ある抽象化仕様の状態遷移が存在することを意味する. この詳細化検証の基本概念のイメージを図 31 (3) に示す. ただし,  $T$  はマスタークロックである. また,  $x$  は現在の  $x$  を意味して,  $x'$  は次の時点の  $x$  を意味する. なお, インデックス A は抽象化仕様を意味して, インデックス C は具体化仕様を意味する.

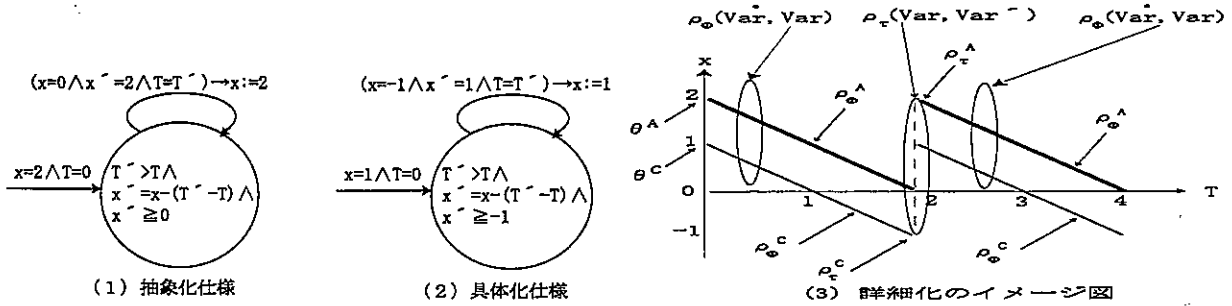


図 31: 詳細化検証の基本概念図

ハイブリッドシステムは,  $M = (Var, L, \Phi, \Theta, T)$  の 5 つ組で定義される.  $L \subseteq Var$  はローカル変数であり, システムの外部からは観測できない変数である. 本論文では, 上流工程の抽象化仕様から中流工程の具体化仕様への詳細化開発は上流工程の抽象化仕様の内部動作を詳細化することを前提としている. そこで, システムの外部から観測可能な変数と観測不可能な変数との区別は重要である. ハイブリッドシステムは分散並行システムの一つであり, プロセス代数 [74] と同様に, 観測可能な動作における模倣性の概念の拡張として, 本論文では詳細化関係を考える.

抽象化仕様  $M^A$  の観測可能なフェーズが具体化仕様  $M^C$  の観測可能なフェーズを包含して, すべての具体化仕様  $M^C$  の状態遷移に対して, ある抽象化仕様  $M^A$  の状態遷移が存在するならば,  $M^C$  は  $M^A$  を詳細化すると呼び,  $M^C \sqsubseteq M^A$  と表記する. 以下に, 詳細化検証ルールを定義する:

### Definition 29 (詳細化検証ルール)

$M^C = (Var^C, L^C, \Phi^C, \Theta^C, T^C)$  が  $M^A = (Var^A, L^A, \Phi^A, \Theta^A, T^A)$  を観測可能な動作において詳細化していることを検証するルールを以下に示す：

なお、 $M^A$  の観測可能な変数 (つまり、 $Var^A - L^A$ ) は  $M^C$  の観測可能な変数 (つまり、 $Var^C - L^C$ ) である。

1.  $\Theta^C \rightarrow \Theta^A[\alpha]$
2.  $\forall \phi^C \in \Phi^C$  に対して以下が成り立つ：  
 $\rho_{\phi^C} \rightarrow \bigvee_{\phi^A \in \Phi^A} \rho_{\phi^A}[\alpha]$
3.  $\forall \tau^C \in T^C$  に対して以下が成り立つ：  
 $\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in T^A} \rho_{\tau^A}[\alpha]$
4. -----
5.  $M^C \sqsubseteq M^A$

これは詳細化を証明するルールであり、ある  $\alpha$  が存在して、1.、2. と 3. を満たすとき、5. のように  $\sqsubseteq$  が定義されることを意味する。ここで、ルールの各行は、以下を意味する：1行目は初期条件  $\Theta^C$  ならば初期条件  $\Theta^A[\alpha]$  であることを意味する。2行目は  $\rho_{\phi^C}$  のフェーズ不変式が  $\rho_{\phi^A}[\alpha]$  のフェーズ不変式に包含されることを意味する。3行目は  $\rho_{\tau^C}$  の状態遷移が抽象的な状態遷移  $\rho_{\tau^A}[\alpha]$  により説明されることを意味する。4行目は前提と結論を分離するラインである。5行目は結論  $M^C \sqsubseteq M^A$  であり、具体化仕様が抽象化仕様を正しく詳細化していることを意味する。

ここで、 $\alpha$  は  $L^A$  の式を  $Var^C$  の式に置き換える写像を意味する。なお、 $X_\alpha(Var^C)$  により、 $Var^C$  の式を変数  $X \in L^A$  に割り当てることを表現する。

以下のように、 $\alpha$  はフェーズ上の写像を与える：

$P^C$  は  $M^C$  の計算の中に現れるフェーズとする。この  $P^C$  を具体化フェーズと呼ぶ。具体化フェーズ  $P^C$  に対応する抽象化フェーズが  $P^A = m_\alpha(P^C)$  であることは、 $P^A$  の中の抽象化仕様の任意のローカル変数  $X \in L^A$  の値が式  $X_\alpha(Var^C)$  の値であることを意味する。ここで、 $m_\alpha$  は  $M^C$  から  $M^A$  への詳細化写像と呼ぶ。この詳細化写像は Abadi らの詳細化写像 [73] である。

■

### 5.3.2 詳細化検証の実験 (その1)

ここでは、図28の上流工程のコントローラのハイブリッドオートマトンの並列合成が図30の中流工程のコントローラのソフトウェアモデルのハイブリッドオートマトンによって正しく詳細化されているかどうかを検証する。すなわち、中流工程のコントローラのフェーズ遷移システム  $M^C = (Var^C, L^C, \Phi^C, \Theta^C, T^C)$  が上流工程のコントローラのフェーズ遷移システム  $M^A = (Var^A, L^A, \Phi^A, \Theta^A, T^A)$  を観測可能な動作において詳細化していることを検証する。

まず、図30の中流工程のコントローラのソフトウェアモデルのハイブリッドオートマトンから、中流工程のコントローラのフェーズ遷移システム  $M^C = (Var^C, L^C, \Phi^C, \Theta^C, T^C)$  を構成する。

$$1. Var^C = \{z_1, z_2, z_3, z_4, \pi_1^C, \pi_2^C, \pi_3^C\}.$$

$$2. L^C = \{z_3, z_4, \pi_1^C, \pi_2^C, \pi_3^C\}.$$

$$3. \Phi^C = \{\phi(idle_1, idle_2, idle), \phi(idle_1, idle_2, control_1), \\ \phi(idle_1, idle_2, control_2), \phi(idle_1, idle_2, preempt), \\ \phi(idle_1, in_2, idle), \dots\}.$$

$$(a) \rho_{\phi(idle_1, idle_2, idle)}^C : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = idle \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4$$

$$(b) \rho_{\phi(idle_1, idle_2, control_1)}^C : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = control_1 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 1 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4$$

$$(c) \rho_{\phi(idle_1, idle_2, control_2)}^C : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = control_2 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 1 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4 \leq 20$$

$$(d) \rho_{\phi(idle_1, idle_2, preempt)}^C : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = preempt \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 1 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4$$

$$(e) \rho_{\phi(idle_1, in_2, idle)}^C : \pi_1^C = idle_1 \wedge \pi_2^C = in_2 \wedge \pi_3^C = idle \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 1 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \leq 5 \wedge 0 \leq z_3 \wedge 0 \leq z_4$$

.....

$$4. \Theta^C : z_1 = 0 \wedge z_2 = 0 \wedge z_3 = 0 \wedge z_4 = 0 \wedge \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = idle \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0$$

$$5. T^C = \{\tau((idle_1, idle_2, idle), (in_1, idle_2, control_1))^C,$$

$$\tau((out_1, idle_2, control_1), (idle_1, idle_2, idle))^C,$$

$$\tau((idle_1, idle_2, idle), (idle_1, in_2, control_2))^C,$$

$$\tau((idle_1, out_2, control_2), (idle_1, idle_2, idle))^C,$$

$$\tau((idle_1, in_2, control_2), (in_1, in_2, preempt))^C,$$

$$\tau((out_1, in_2, preempt), (idle_1, in_2, control_2))^C,$$

.....\}.

$$(a) \rho_{\tau((idle_1, idle_2, idle), (in_1, idle_2, control_1))^C} : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = idle \wedge \pi_1^{C'} = in_1 \wedge \pi_2^{C'} = idle_2 \wedge \pi_3^{C'} = control_1 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1' \leq 5 \wedge 0 \leq z_2' \wedge 0 \leq z_3 \leq 20' \wedge 0 \leq z_4 \wedge \dot{z}_1' = 1 \wedge \dot{z}_2' = 0 \wedge \dot{z}_3' = 1 \wedge \dot{z}_4' = 0$$

$$(b) \rho_{\tau((out_1, idle_2, control_1), (idle_1, idle_2, idle))^C} : \pi_1^C = out_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = control_1 \wedge \pi_1^{C'} = idle_1 \wedge \pi_2^{C'} = idle_2 \wedge \pi_3^{C'} = idle \wedge 0 \leq z_1 \leq 5 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4 \wedge \dot{z}_1 = 1 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \wedge 0 \leq z_3' \wedge 0 \leq z_4' \wedge \dot{z}_1' = 0 \wedge \dot{z}_2' = 0 \wedge \dot{z}_3' = 0 \wedge \dot{z}_4' = 0$$

$$(c) \rho_{\tau((idle_1, idle_2, idle), (idle_1, in_2, control_2))^C} : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = idle \wedge \pi_1^{C'} = idle_1 \wedge \pi_2^{C'} = in_2 \wedge \pi_3^{C'} = control_2 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \leq 5 \wedge 0 \leq z_3' \wedge 0 \leq z_4' \leq 20 \wedge \dot{z}_1' = 0 \wedge \dot{z}_2' = 1 \wedge \dot{z}_3' = 0 \wedge \dot{z}_4' = 1$$

- (d)  $\rho_{\tau((idle_1, out_2, control_2), (idle_1, idle_2, idle))}^C : \pi_1^C = idle_1 \wedge \pi_2^C = out_2 \wedge \pi_3^C = control_2 \wedge \pi_1^C \wedge \pi_2^C \wedge \pi_3^C = idle_1 \wedge idle_2 \wedge \pi_3^C = idle \wedge 0 \leq z_1 \wedge 0 \leq z_2 \leq 5 \wedge 0 \leq z_3 \wedge 0 \leq z_4 \leq 20 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 1 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 1 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \wedge 0 \leq z_3' \wedge 0 \leq z_4' \wedge \dot{z}_1' = 0 \wedge \dot{z}_2' = 0 \wedge \dot{z}_3' = 0 \wedge \dot{z}_4' = 0$
- (e)  $\rho_{\tau((idle_1, in_2, control_2), (in_1, in_2, preempt))}^C : \pi_1^C = idle_1 \wedge \pi_2^C = in_2 \wedge \pi_3^C = control_2 \wedge \pi_1^C \wedge \pi_2^C \wedge \pi_3^C = in_1 \wedge \pi_2^C \wedge \pi_3^C = preempt \wedge 0 \leq z_1 \wedge 0 \leq z_2 \leq 5 \wedge 0 \leq z_3 \wedge 0 \leq z_4 \leq 20 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 1 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 1 \wedge 0 \leq z_1' \leq 5 \wedge 0 \leq z_2' \leq 5 \wedge 0 \leq z_3' \leq 20 \wedge 0 \leq z_4' \wedge \dot{z}_1' = 1 \wedge \dot{z}_2' = 1 \wedge \dot{z}_3' = 1 \wedge \dot{z}_4' = 0$
- (f)  $\rho_{\tau((idle_1, idle_2, preempt), (idle_1, idle_2, control_2))}^C : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = preempt \wedge \pi_1^C \wedge \pi_2^C \wedge \pi_3^C = idle_1 \wedge \pi_2^C \wedge \pi_3^C = control_2 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 1 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \wedge 0 \leq z_3' \wedge 0 \leq z_4' \leq 20 \wedge \dot{z}_1' = 0 \wedge \dot{z}_2' = 0 \wedge \dot{z}_3' = 0 \wedge \dot{z}_4' = 1$
- .....
- .....

次に、図 2 8 の上流工程のコントローラの仕様のハイブリッドオートマトンから、上流工程のコントローラのフェーズ遷移システム  $M^A = (Var^A, L^A, \Phi^A, \Theta^A, T^A)$  を構成する。

1.  $Var^A = \{z_1, z_2, \pi_1^A, \pi_2^A\}$ .
2.  $L^A = \{\pi_1^A, \pi_2^A\}$ .
3.  $\Phi^A = \{\phi(idle_1, idle_2), \phi(idle_1, in_2), \phi(out_1, in_2), \phi(out_1, idle_2), \phi(in_1, in_2), \phi(in_1, idle_2), \phi(idle_1, out_2), \phi(in_1, out_2), \phi(out_1, out_2)\}$ .
- (a)  $\rho_{\phi(idle_1, idle_2)}^A : \pi_1^A = idle_1 \wedge \pi_2^A = idle_2 \wedge \dot{z}_1 = 1 \wedge \dot{z}_2 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2$
- (b)  $\rho_{\phi(idle_1, in_2)}^A : \pi_1^A = idle_1 \wedge \pi_2^A = in_2 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 1 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \leq 5$

.....

.....

4.  $\Theta^A : z_1 = 0 \wedge z_2 = 0 \wedge \pi_1^A = idle_1 \wedge \pi_2^A = idle_2 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0$
5.  $T^A = \{\tau(idle_1, idle_2)(idle_1, in_2)^A, \tau(idle_1, in_2)(idle_1, in_2)^A, \tau(idle_1, in_2)(idle_1, idle_2)^A, \tau(idle_1, idle_2)(in_1, idle_2)^A, \tau(in_1, idle_2)(idle_1, idle_2)^A, \tau(in_1, idle_2)(in_1, idle_2)^A, \tau(idle_1, idle_2)(out_1, idle_2)^A, \tau(out_1, idle_2)(idle_1, idle_2)^A, \tau(out_1, idle_2)(out_1, idle_2)^A, \tau(idle_1, idle_2)(idle_1, out_2)^A, \tau(idle_1, out_2)(idle_1, idle_2)^A, \tau(idle_1, out_2)(idle_1, out_2)^A, \tau(idle_1, out_2)(idle_1, in_2)^A, \tau(out_1, idle_2)(in_1, idle_2)^A, \tau(idle_1, in_2)(out_1, in_2)^A, \tau(out_1, in_2)(idle_1, in_2)^A, \tau(out_1, in_2)(out_1, in_2)^A, \tau(idle_1, in_2)(in_1, in_2)^A, \tau(in_1, in_2)(idle_1, in_2)^A, \tau(in_1, in_2)(in_1, in_2)^A, \tau(out_1, in_2)(in_1, in_2)^A, \tau(idle_1, out_2)(in_1, out_2)^A,$

$$\begin{aligned} & \mathcal{T}(in_1, out_2)(idle_1, out_2)^A, \mathcal{T}(in_1, out_2)(in_1, out_2)^A, \\ & \mathcal{T}(idle_1, out_2)(out_1, out_2)^A, \mathcal{T}(out_1, out_2)(idle_1, out_2)^A, \\ & \mathcal{T}(out_1, out_2)(out_1, out_2)^A, \mathcal{T}(out_1, out_2)(in_1, out_2)^A, \\ & \mathcal{T}(in_1, out_2)(in_1, in_2)^A, \mathcal{T}(in_1, out_2)(in_1, idle_2)^A, \\ & \mathcal{T}(in_1, idle_2)(in_1, out_2)^A, \mathcal{T}(in_1, idle_2)(in_1, in_2)^A, \\ & \mathcal{T}(in_1, in_2)(in_1, idle_2)^A, \mathcal{T}(out_1, idle_2)(out_1, in_2)^A, \\ & \mathcal{T}(out_1, in_2)(out_1, idle_2)^A, \mathcal{T}(out_1, out_2)(out_1, in_2)^A \}. \end{aligned}$$

$$(a) \rho_{\mathcal{T}(idle_1, idle_2)(idle_1, in_2)}^A : \pi_1^A = idle_1 \wedge \pi_2^A = idle_2 \wedge \pi_1^{A'} = idle_1 \wedge \pi_2^{A'} = in_2 \wedge z_1 \geq 0 \wedge z_2 \geq 0 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \leq 5 \wedge \dot{z}_1' = 0 \wedge \dot{z}_2' = 1$$

$$(b) \rho_{\mathcal{T}(idle_1, in_2)(idle_1, idle_2)}^A : \pi_1^A = idle_1 \wedge \pi_2^A = in_2 \wedge \pi_1^{A'} = idle_1 \wedge \pi_2^{A'} = idle_2 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \leq 5 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 1 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \wedge \dot{z}_1' = 1 \wedge \dot{z}_2' = 0$$

.....  
 .....

次に,  $M^C = (Var^C, L^C, \Phi^C, \Theta^C, T^C)$  が  $M^A = (Var^A, L^A, \Phi^A, \Theta^A, T^A)$  を観測可能な動作において詳細化していることを検証する。

ここで,  $\alpha$  を以下のように定義する :

$$\alpha : (\pi_1^A, \pi_2^A) \rightarrow \left\{ \begin{array}{ll} \text{if} & (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, idle_2, state) \\ \text{then} & (idle_1, idle_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, in_2, state) \\ \text{then} & (idle_1, in_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, out_2, state) \\ \text{then} & (idle_1, out_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (in_1, idle_2, state) \\ \text{then} & (in_1, idle_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (out_1, idle_2, state) \\ \text{then} & (out_1, idle_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (in_1, in_2, state) \\ \text{then} & (in_1, in_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (out_1, in_2, state) \\ \text{then} & (out_1, in_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (in_1, out_2, state) \\ \text{then} & (in_1, out_2) \\ \text{elseif} & (\pi_1^C, \pi_2^C, \pi_3^C) = (out_1, out_2, state) \\ \text{then} & (out_1, out_2) \end{array} \right.$$

ただし,  $state$  は  $idle, control_1, control_2, preempt$  のいずれかである.

1.  $\Theta^C \rightarrow \Theta^A[\alpha]$  の検証

$\Theta^C : z_1 = 0 \wedge z_2 = 0 \wedge z_3 = 0 \wedge z_4 = 0 \wedge \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = idle \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0$  及び  $\Theta^A : z_1 = 0 \wedge z_2 = 0 \wedge \pi_1^A = idle_1 \wedge \pi_2^A = idle_2 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0$  である.

ここで, 以下の  $\alpha$  を使う :

$\alpha : (\pi_1^A, \pi_2^A) \rightarrow$   
 $\left\{ \begin{array}{l} \text{if } (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, idle_2, idle) \\ \text{then } (idle_1, idle_2) \end{array} \right.$

また, 変数  $z_3, z_4$  は観測不能なローカル変数である.

以上より, 観測可能な動作において,

$\Theta^C \rightarrow \Theta^A[\alpha]$

は成り立つ.

2.  $\forall \phi^C \in \Phi^C$  に対して以下が成り立つことの検証 :

$\rho_{\phi^C} \rightarrow \bigvee_{\phi^A \in \Phi^A} \rho_{\phi^A}[\alpha]$

(a)  $\rho_{\phi_{(idle_1, idle_2, idle)}^C} : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = idle \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4$  及び  $\rho_{\phi_{(idle_1, idle_2)}^A} : \pi_1^A = idle_1 \wedge \pi_2^A = idle_2 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2$  である.

ここで, 以下の  $\alpha$  を使う :

$\alpha : (\pi_1^A, \pi_2^A) \rightarrow$   
 $\left\{ \begin{array}{l} \text{if } (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, idle_2, idle) \\ \text{then } (idle_1, idle_2) \end{array} \right.$

また, 変数  $z_3, z_4$  は観測不能なローカル変数である.

以上より, 観測可能な動作において,

$\rho_{\phi_{(idle_1, idle_2, idle)}^C} \rightarrow \rho_{\phi_{(idle_1, idle_2)}^A}[\alpha]$

は成り立つ.

(b)  $\rho_{\phi_{(idle_1, idle_2, control_1)}^C} : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = control_1 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 1 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4$  及び  $\rho_{\phi_{(idle_1, idle_2)}^A} : \pi_1^A = idle_1 \wedge \pi_2^A = idle_2 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge 0 \leq z_1 \wedge 0 \leq z_2$  である.

ここで, 以下の  $\alpha$  を使う :

$\alpha : (\pi_1^A, \pi_2^A) \rightarrow$   
 $\left\{ \begin{array}{l} \text{if } (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, idle_2, control_1) \\ \text{then } (idle_1, idle_2) \end{array} \right.$

また, 変数  $z_3, z_4$  は観測不能なローカル変数である.

以上より, 観測可能な動作において,

$$\rho_{\phi_{(idle_1, idle_2, control_1)}}^C \rightarrow \rho_{\phi_{(in_1, idle_2)}}^A[\alpha]$$

は成り立つ.

.....  
 .....

3.  $\forall \tau^C \in \mathcal{T}^C$  に対して以下が成り立つことの検証:

$$\rho_{\tau^C} \rightarrow \bigvee_{\tau^A \in \mathcal{T}^A} \rho_{\tau^A}[\alpha]$$

(a)  $\rho_{\tau_{((idle_1, idle_2, idle), (in_1, idle_2, control_1))}}^C : \pi_1^C = idle_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = idle \wedge \pi_1^{C'} = in_1 \wedge \pi_2^{C'} = idle_2 \wedge \pi_3^{C'} = control_1 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \wedge 0 \leq z_4 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 0 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1' \leq 5 \wedge 0 \leq z_2' \wedge 0 \leq z_3 \leq 20' \wedge 0 \leq z_4 \wedge \dot{z}_1' = 1 \wedge \dot{z}_2' = 0 \wedge \dot{z}_3' = 1 \wedge \dot{z}_4' = 0$  及び  $\rho_{\tau_{(idle_1, idle_2)(in_1, idle_2)}}^A : \pi_1^A = idle_1 \wedge \pi_2^A = idle_2 \wedge \pi_1^{A'} = in_1 \wedge \pi_2^{A'} = idle_2 \wedge 0 \leq z_1 \wedge 0 \leq z_2 \wedge \dot{z}_1 = 0 \wedge \dot{z}_2 = 0 \wedge 0 \leq z_1' \leq 5 \wedge 0 \leq z_2' \wedge \dot{z}_1' = 1 \wedge \dot{z}_2' = 0$  である.

ここで, 以下の  $\alpha$  を使う:

$$\alpha : (\pi_1^A, \pi_2^A) \rightarrow \begin{cases} \text{if } (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, idle_2, idle) \\ \text{then } (idle_1, idle_2) \\ \text{elseif } (\pi_1^C, \pi_2^C, \pi_3^C) = (in_1, idle_2, control_1) \\ \text{then } (in_1, idle_2) \end{cases}$$

また, 変数  $z_3, z_4$  は観測不能なローカル変数である.

以上より, 観測可能な動作において,

$$\rho_{\tau_{((idle_1, idle_2, idle), (in_1, idle_2, control_1))}}^C \rightarrow \rho_{\tau_{(idle_1, idle_2)(in_1, idle_2)}}^A[\alpha]$$

は成り立つ.

(b)  $\rho_{\tau_{((out_1, idle_2, control_1), (idle_1, idle_2, idle))}}^C : \pi_1^C = out_1 \wedge \pi_2^C = idle_2 \wedge \pi_3^C = control_1 \wedge \pi_1^{C'} = idle_1 \wedge \pi_2^{C'} = idle_2 \wedge \pi_3^{C'} = idle \wedge 0 \leq z_1 \leq 5 \wedge 0 \leq z_2 \wedge 0 \leq z_3 \leq 20 \wedge 0 \leq z_4 \wedge \dot{z}_1 = 1 \wedge \dot{z}_2 = 0 \wedge \dot{z}_3 = 1 \wedge \dot{z}_4 = 0 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \wedge 0 \leq z_3' \wedge 0 \leq z_4' \wedge \dot{z}_1' = 0 \wedge \dot{z}_2' = 0 \wedge \dot{z}_3' = 0 \wedge \dot{z}_4' = 0$  及び  $\rho_{\tau_{(out_1, idle_2)(idle_1, idle_2)}}^A : \pi_1^A = out_1 \wedge \pi_2^A = idle_2 \wedge \pi_1^{A'} = idle_1 \wedge \pi_2^{A'} = idle_2 \wedge 0 \leq z_1 \leq 5 \wedge 0 \leq z_2 \wedge \dot{z}_1 = 1 \wedge \dot{z}_2 = 0 \wedge 0 \leq z_1' \wedge 0 \leq z_2' \wedge \dot{z}_1' = 0 \wedge \dot{z}_2' = 0$  である.

ここで, 以下の  $\alpha$  を使う:

$$\alpha : (\pi_1^A, \pi_2^A) \rightarrow \begin{cases} \text{if } (\pi_1^C, \pi_2^C, \pi_3^C) = (out_1, idle_2, control_1) \\ \text{then } (out_1, idle_2) \\ \text{elseif } (\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, idle_2, idle) \\ \text{then } (idle_1, idle_2) \end{cases}$$

また, 変数  $z_3, z_4$  は観測不能なローカル変数である.

以上より, 観測可能な動作において,

$$\begin{aligned} & \rho_{\tau((out_1, idle_2, control_1), (idle_1, idle_2, idle))}^C \\ & \rightarrow \rho_{\tau(out_1, idle_2)(idle_1, idle_2)}^A[\alpha] \end{aligned}$$

は成り立つ.

.....  
.....

以上の 1. , 2. , 3. より,  $M^C \sqsubseteq M^A$  が成り立つことが検証できた.

### 5.3.3 詳細化検証の実験 (その2)

次に, 図 2 8 の上流工程のコントローラのハイブリッドオートマトンの並列合成が以下の図 3 2 の中流工程のコントローラのソフトウェアモデルのハイブリッドオートマトンによって正しく詳細化されているかどうかを検証する. すなわち, 中流工程のコントローラのフェーズ遷移システム  $M^C = (Var^C, L^C, \Phi^C, \Theta^C, T^C)$  が上流工程のコントローラのフェーズ遷移システム  $M^A = (Var^A, L^A, \Phi^A, \Theta^A, T^A)$  を観測可能な動作において詳細化していることを検証する. 図 3 2 では, コントローラ 1 のタスクに異常処理を追加したものである.

ここで,  $\alpha$  を以下のように定義する:

$$\alpha : (\pi_1^A, \pi_2^A) \rightarrow$$

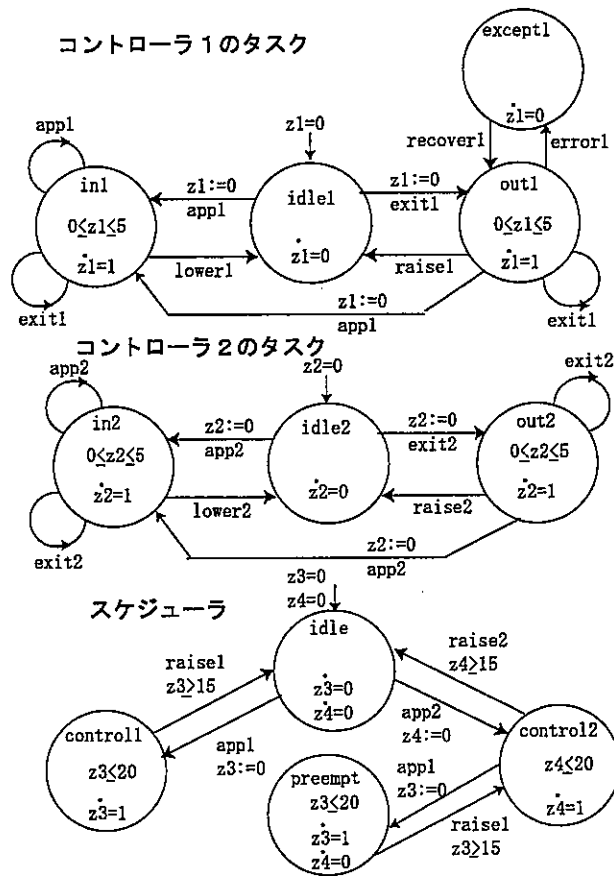


図 32: もう一つの中流工程のソフトウェアモデル

```

if       $(\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, idle_2, state_3)$ 
then     $(idle_1, idle_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, in_2, state_3)$ 
then     $(idle_1, in_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (idle_1, out_2, state_3)$ 
then     $(idle_1, out_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (in_1, idle_2, state_3)$ 
then     $(in_1, idle_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (out_1, idle_2, state_3)$ 
then     $(out_1, idle_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (in_1, in_2, state_3)$ 
then     $(in_1, in_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (out_1, in_2, state_3)$ 
then     $(out_1, in_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (in_1, out_2, state_3)$ 
then     $(in_1, out_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (out_1, out_2, state_3)$ 
then     $(out_1, out_2)$ 
elseif  $(\pi_1^C, \pi_2^C, \pi_3^C) = (except_1, state_2, state_3)$ 
then     $(out_1, state_2)$ 

```

ただし、 $state_2$  は  $idle_2, in_2, out_2$  のいずれかであり、 $state_3$  は  $idle, control_1, control_2, preempt$  のいずれかである。

この  $\alpha$  を使って、前述と同様に詳細化検証を行うと、中流工程のコントローラのフェーズ遷移システム  $M^C = (Var^C, L^C, \Phi^C, \Theta^C, T^C)$  が上流工程のコントローラのフェーズ遷移システム  $M^A = (Var^A, L^A, \Phi^A, \Theta^A, T^A)$  を観測可能な動作において詳細化していることが検証できた。

詳しい詳細化検証の手続きは前述と同様なので、紙面の都合上から、省略する。

### 5.3.4 議論

一般的には、ソフトウェアの上流工程の仕様と中流工程の仕様との間には大きなセマンティックギャップがあり、単なる模倣関係や言語包含関係では、詳細化の検証はできない。そこで、本論文では、上流工程の仕様の内部動作を中流工程の仕様の動作で置換する詳細化写像を導入することによって、そのセマンティックギャップを克服して詳細化関係を検証した。これは、上記の実験が示すように、手作業で詳細化写像を定義して、すべてのフェーズ遷移と状態遷移を洗い出さなければならないといった困難な作業を伴う。もちろん、これにより、すべての上流工程の仕様と中流工程の仕様との詳細化

関係が検証できるわけではない。本論文では、上流工程の仕様から中流工程の仕様への詳細化開発は上流工程の仕様の内部動作を詳細化することを仮定している。この仮定により、すべての中流工程の仕様の観測可能な動作に対して、ある上流工程の仕様の観測可能な動作が存在するといった詳細化の正しさの基準を定めることができた。すなわち、詳細化検証の問題は詳細化開発のスタイルにすべて依存しており、詳細化開発のスタイルの形式的な定義が重要となるであろう。とりわけ、組込み型システムの開発スタイルにとっては、スケジューリングポリシーやハードウェア構成なども重要な影響因子となり、複雑である。

一方、本論文で提案した詳細化検証手法は健全であるが、上記のとおり完全ではない。しかし、Abadiらの手法 [73] に従って、提案した検証手法を拡張すれば完全性は得られるものと考えられる。また、この詳細化検証を効率的に行うには、定理証明器による計算機支援が不可欠である。明らかに、この詳細化検証を行うときに構成する論理式は一般的には決定不能な論理式であるので、これらへの対処も重要となる。その対処方法は、ハードウェアやソフトウェアといった問題領域に対応して、SVCなどの論理式の真偽判定器が提案されており [75, 76, 77]、検証技術の実用化には非常に重要である。また、HOL [78] や PVS [79] といった定理証明器が開発されており、上記の論理式の真偽判定器と組み合わせると大きな効果を上げると考えられる。

## 5.4 むすび

マイクロプロセッサのほとんどは組込み型システムに使われており、組込み型システム開発の工業化は重要である。また、組込み型システムはマイクロプロセッサとソフトウェアによって、何らかの制御法則に支配された制御系をコントロールするものであり、ハイブリッドモデルの観点は必要不可欠である。さらに、組込み型システムのソフトウェアモデルはリアルタイムオペレーティングシステム上で動作しており、プリエンプティブなどのスケジューリングポリシーはハイブリッドモデルを使うと簡潔に記述できる。本論文では、以上の考え方より、ハイブリッドモデルによる組込み型システム開発の上流工程から中流工程への詳細化検証を提案して実験した。具体的には、あるスケジューリングポリシーを基礎として、ハイブリッドオートマトンで仕様記述された、組込み型システムの上流工程の仕様を中流工程のソフトウェアモデルに実装して、詳細化写像を導入した詳細化検証手法を提案して、その詳細化の正しさを検証した。

今後の研究課題としては以下が考えられる：

1. 定理証明器上に、提案した詳細化検証手法を実装する。
2. 他のスケジューリングポリシーなどのより多くの事例を試行して、上流工程から中流工程への段階的詳細化開発手法を形式化する。
3. 中流工程から下流工程への段階的詳細化開発手法とその詳細化検証理論を開発する。

## 6 まとめと今後の展望

本研究では、従来個別に研究されていた形式的手法を統合化して、新しい分散システムの設計支援を実現した。そのポイントは、開放型システムとして分散システムをとらえて、実時間モデル及びハイブリッドモデルの観点からモデル化して、自動検証及び演繹的検証により設計の正当性を効率的に保証するものである。以上の観点より、我々は、以下の4つのモデルから、設計支援を実現した。

1. 実時間型開放分散システムの Assume-guarantee 方式による自動的設計支援
2. 実時間型開放分散システムの演繹的設計支援
3. 実時間型開放分散システムの Assume-guarantee 方式による演繹的設計支援
4. ハイブリッドモデルに基づく開放分散システムの設計支援

提案手法により、従来不可能であった分散システムの自律性や発展性が仕様記述できて、効率的な設計支援が実現できた。

本研究では、ある意味では、我々は提案した設計支援手法をプロトタイプ的に実装して小規模システムに適用及び評価した。今後は、本格的に提案手法を計算機上に実装して、実問題に適用して評価する予定である。

## 参考文献

- [1] S. Mullender. Distributed Systems, Second Edition. *Addison-Wesley*, P.601, 1993.
- [2] J.M. Wing, J. Woodcock, J. Davies. FM'99-Formal Methods. *Lecture Notes in Computer Science 1708,1709*, 1999.
- [3] R. Milner. Elements of interaction (A. M. Turing Award Lecture). *Communications of the ACM*, Vol.36, No.1, pp.78-89, 1991.
- [4] A. Pnueli. Verification Engineering: A Future Profession. (A. M. Turing Award Lecture). *16th ACM Symp. on Principles of Distributed Computing*, 1997.
- [5] J.R. Burch, E.M. Clarke, K.L. McMillan, D. Dill, L.J. Hwang. Symbolic Model Checking:  $10^{20}$  States and Beyond. *Proc. 5th LICS*, pp.428-439, 1990.
- [6] R. Cleaveland, J. Parrow, B. Steffen. The concurrency workbench. *Lecture Notes in Computer Science 407*, pp.24-37, 1989.
- [7] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, Y. Wang. UPPAAL-a tool suite for automatic verification of real-time systems. *Lecture Notes in Computer Science 1066*, pp.232-243, 1996.
- [8] R. Alur, D. Dill. The theory of Timed automata. *Lecture Notes in Computer Science 600*, pp.45-73, 1992.
- [9] R. Alur, C. Courcoubetis, D. Dill. Model checking for real-time systems. *Proc. of 5th LICS*, pp. 414-425, 1992.
- [10] N.A. Lynch, H. Attiya. Using mapping to prove timing properties. *Distributed Computing*, No.6, pp.121-139, 1992.
- [11] 山根智. 時間模倣関係による実時間システムの階層的設計手法. 情報処理学会論文誌, Vol.40, No.7, pp.3001-3015, 1999. (preliminary versions appeared in FM-Trends 98, LNCS 1641, pp.151-167, 1998).
- [12] A. Pnueli. *In Transition From Global to Modular Temporal Reasoning about Programs*. Logics and Models of Concurrent Systems, NATO ASI Series, pp.123-144, 1985.
- [13] R. Alur and T.A. Henzinger. *Modularity for timed and hybrid systems*. LNCS 1243, pp. 74-88, 1997.

- [14] R. Gawlick, R. Segala, J.F. Sogaard-Andersen, N. Lynch. *Liveness in Timed and Untimed Systems*. Information and Computation, Vol.141, pp.119-171, 1998.
- [15] T.H. Cormen, C.E. Leiserson, R.L. Rivest. Introduction to algorithms, MIT Press, 1990.
- [16] IEEE Computer Society. IEEE ANSI/IEEE 802.3, ISO/DIS 8802/3. IEEE Computer Society Press, 1985.
- [17] Y. Kesten, Z. Manna, A. Pnueli. Verifying clocked transition systems. In *Lecture Notes in Computer Science 1066*, pp.13-40, 1996.
- [18] C. Daws, A. Olivero, S. Tripakis, S. Yovine. The tool KRONOS. In *Lecture Notes in Computer Science 1066*, pp.208-219, 1996.
- [19] R. Alur, R. Kurshan. Timing analysis in COSPAN. In *Lecture Notes in Computer Science 1066*, pp.220-231, 1996.
- [20] R. E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*.Vol.C-35, No.8, pp. 677-691, IEEE Computer Society, 1986.
- [21] R. Alur, C. Courcoubetis, D. Dill. Model checking for real-time systems. *Proc. of 5th LICS*, pp. 414-425, 1990.
- [22] T.H. Cormen, C.E. Leiserson, R.L. Rivest. Introduction to algorithms, MIT Press, 1990.
- [23] IEEE Computer Society. IEEE ANSI/IEEE 802.3, ISO/DIS 8802/3. IEEE Computer Society Press, 1985.
- [24] R. Cleaveland, J. Parrow, B. Steffen. The concurrency workbench. In *Lecture Notes in Computer Science 407*, pp.24-37, 1989.
- [25] Y. Kesten, Z. Manna, A. Pnueli. Verifying clocked transition systems. In *Lecture Notes in Computer Science 1066*, pp.13-40, 1996.
- [26] C. Daws, A. Olivero, S. Tripakis, S. Yovine. The tool KRONOS. In *Lecture Notes in Computer Science 1066*, pp.208-219, 1996.
- [27] R. Alur, R. Kurshan. Timing analysis in COSPAN. In *Lecture Notes in Computer Science 1066*, pp.220-231, 1996.
- [28] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, Y. Wang. UPPAAL-a tool suite for automatic verification of real-time systems. In *Lecture Notes in Computer Science 1066*, pp.232-243, 1996.

- [29] R. E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*. Vol.C-35, No.8, pp. 677-691, IEEE Computer Society, 1986.
- [30] 情報処理学会. 特集：リアルタイムシステム. 情報処理学会誌, VOL.35, NO.1, pp.11-54, 1994.
- [31] 山根 智. 時間ステートチャートに基づくリアルタイムシステム検証方式. 情報処理学会論文誌, Vol.35, No.12 ,pp.2640-2650, 1994.
- [32] R. Alur, D. Dill. Automata for Modeling Real-Time Systems. *Lecture Notes in Computer Science 443*, pp.322-335, 1990.
- [33] Y. Kesten, Z. Manna, A. Pnueli. Verifying clocked transition systems. *Lecture Notes in Computer Science 1066*, pp.13-40, 1996.
- [34] D. Harel, H. Lachover, A. Naamad, A. Pnueli, et-al. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Trans. on SE*, VOL.16, NO.4, pp.403-414, 1990.
- [35] A. Peron, A. Maggiolo. Transitions as Interrupts:A New Semantics for Timed Statecharts. *Lecture Notes in Computer Science 789*, pp.806-821, 1994.
- [36] H. Harel, M. Politi. Modeling Reactive Systems with Statecharts. *McGraw-Hill*, P.258, 1998.
- [37] M.R. Garey, D.S. Johnson. Computers and Intractability. *W.H. FREEMAN AND COMPANY*, P.340, 1979.
- [38] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. *Proceedings of IEEE RTSS*, pp. 157-166, 1992.
- [39] Z. Manna, A. Pnueli. *Temporal Verification of Reactive Systems*. Springer-Verlag, P. 512, 1995.
- [40] 山ノ口 , 山根. 実時間システムの演繹的検証. 電子情報通信学会研究報告, SS2000-6, pp.1-8, 2000.
- [41] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proc. of 7th LICS*, pp. 394-406, 1992.
- [42] 山根. ハードリアルタイムシステムの形式的仕様記述と形式的検証. 情報処理学会論文誌, Vol.42, No.6, pp.1623-1635, 2001

- [43] J. Misra, K.M. Chandy. Proofs of Networks of Processes. *IEEE Trans. on SE*, Vol.7, No.4, pp.417-426, 1981.
- [44] L. Lamport. Specifying Concurrent Program Modules. *ACM TOPLAS*, Vol.5, No.2, pp.190-222, 1983.
- [45] E. W. Stark. A Proof Technique for Rely/Guarantee Properties. *LNCS 206*, pp. 369-391, 1985.
- [46] E. Chang, Z. Manna, A. Pnueli. Compositional Verification of Real-time Systems. *LICS*, pp. 458-465, 1994
- [47] R. Alur, T.A. Henzinger. Reactive modules. *LICS*, pp.207-218, 1996.
- [48] R. Alur, D. Dill. Automata for Modeling Real-Time Systems. *LNCS 443*, pp.322-335, 1990.
- [49] 山根. Assume-Guarantee 検証による実時間システムの階層的設計支援手法. 情報処理学会論文誌, Vol.41, No.12, pp.3352-3362, 2000
- [50] T. A. Henzinger, M. Minea, V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. *LNCS 2034*, pp. 275-290, 2001.
- [51] Y. Kesten, Z. Manna, A. Pnueli. Verifying clocked transition systems. *LNCS 1066*, pp.13-40, 1996.
- [52] N. Bjorner, Z. Manna, H. Sipma, T. Uribe. Deductive Verification of Real-time Systems using STeP. *Theoretical Computer Science*, Vol.253, No.1, pp.27-60, 2001.
- [53] 山ノ口, 山根. 実時間システムの演繹的検証. 電子情報通信学会研究報告 *SS2000-6*, pp.1-8, 2000
- [54] Z. Manna and the STeP group. STeP: The Stanford Temporal Prover (Educational Release), User's Manual. Technical report STAN-CS-TR-95-1562. *Computer Science Department, Stanford University*, P.138, 1995.
- [55] M. Abadi, L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253-284, 1991.
- [56] P. Cousot, R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th POPL*, pp. 238-252, 1977.
- [57] 山根. 統一的セマンティックモデルに基づくリアクティブシステムの形式的開発方法論. 情報処理学会論文誌, Vol.41, No.3, pp.767-782, 2000

- [58] F.W. Vaandrager. Hybrid systems. *Images of SMC Research 1996*, pp.305-316, Stichting Mathematisch Centrum, 1996.
- [59] 山根智. ハイブリッドシステムのモジュール演繹的検証手法. 電子情報通信学会研究報告, CST2001-4, pp.21-28, 2001
- [60] 山根智. ハイブリッドシステムの高信頼性設計方法論. 電子情報通信学会研究報告, FTS2001-71, pp.17-24, 2001
- [61] J.C. Corbett. Timing Analysis of Ada Tasking Programs. *IEEE Transactions on Software Engineering*, 22(7), pp.461-483, 1996.
- [62] R. Alur, T.A. Henzinger, P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22(3), pp.181-201, 1996.
- [63] A.M. Tilborg, G.M. Koob. Foundations of Real-time Computing: Formal Specifications and Methods. *Kluwer Academic Pub.*, P.316, 1991.
- [64] B. Selic, G. Gullekson, P.T. Ward. Real-Time Object-Oriented Modeling. *Wiley*, P.525, 1994.
- [65] O. Maler, Z. Manna, A. Pnueli. From timed to hybrid systems. *LNCS 600*, pp.447-484, 1992.
- [66] T.A. Henzinger, Z. Manna, A. Pnueli. Towards refining temporal specifications into hybrid systems. *LNCS 736*, pp. 60-76, 1993.
- [67] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho. Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems. *LNCS 736*, pp. 209-229, 1993.
- [68] T.A. Henzinger. Hybrid automata with finite bisimulations. *LNCS 944*, pp. 324-335, 1995.
- [69] N. Lynch, R. Segala, F. Vaandrager, H.B. Weinberg. Hybrid I/O Automata. *LNCS 1066*, pp. 496-510, 1996.
- [70] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifications of hybrid systems in CHARON. *LNCS 1790*, pp. 6-19, 2000
- [71] T. A. Henzinger. Masaccio: a formal model for embedded components. *LNCS 1872*, pp. 549-563, 2000.
- [72] T. A. Henzinger, B. Horowitz, C. M. Kirsch. Giotto: a time-triggered language for embedded programming. *LNCS 2211*, pp. 166-184, 2001.

- [73] M. Abadi, L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253-284, 1991.
- [74] R. Milner. Communication and Concurrency. *Prentice Hall*, P.260, 1989.
- [75] N. Bjorner, M. Stickel, T. E. Uribe. A practical integration of first-order reasoning and decision procedures. *LNCS 1249*, pp. 101-115, 1997.
- [76] A. Pnueli, Y. Rodeh, O. Shtrichman, M. Siegel. Deciding equality formulas by small domains instantiations. *LNCS 1633*, pp. 455-469, 1999.
- [77] C.W. Barrett, D.L. Dill, A. Stump. A Framework for Cooperating Decision Procedures. *LNCS 1831*, pp. 79-97, 2000.
- [78] M.J.C. Gordon, T.F. Melham. Introduction to HOL. *Cambridge University Press*, P.471, 1991.
- [79] S. Owre, J.M. Rushby, N. Shankar. PVS: A prototype verification system *LNAI 607*, pp. 748-752, 1992