修 士 論 文

時間オートマトンから時間モジュールへの変換に関する考察

指導教官　落水 浩一郎 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

館 宣伸

2001 年 2 月 15 日

要 旨

通信システム、論理回路、オペレーティングシステムといった実時間システムの大規模化や多様化にともない、これらのシステムの設計段階における信頼性の検証が重要になってきている。これらの大規模なシステムでは、論理的動作の信頼性だけでなく、動作のタイミングの信頼性も要求されており、シミュレーションによるテストは困難である。そこで、設計されたシステムを数学的手法により信頼性を検証できることが望ましい。このような背景のもとで、実時間システムの仕様記述および検証のために時間オートマトン [1] および時間モジュール [2] が提案されている。Alur らは、文献 [2] において、時間モジュールが時間オートマトンの一般化であるということを示唆してはいるが、後者から前者への具体的な変換手続きは与えていない。

そこで、本論文では、時間オートマトンから時間モジュールへの変換について考察する。変換は、時間オートマトンから時間モジュールへの等価変換と、等価変換により得られた時間モジュールに対する最適化からなる。なお、本論文では、変換対象を時間 I/O オートマトンとする。これは、時間オートマトンのイベントに入力と出力の区別を付加し拡張したものである。対象を時間 I/O オートマトンとする理由は、本論文で記述の対象とするシステムやプロセスでは、イベントにおける入力と出力の区別が必要だからである。提案する変換は、時間 I/O オートマトンから時間モジュールへの等価変換と、等価変換より得られる時間モジュールに対して記述の最適化からなる。

この変換の対象として、まず、実時間システムの列車制御システムを例に挙げる。これにより、本論文で提案する変換が実時間システムへ適用可能であることを示す。また、変換の対象として、次に、ソフトウェアプロセスをとる。ソフトウェアプロセス記述の例題としては、Kellner らによる「ソフトウェアプロセスモデリングのための例題」[3] が有名であり、本論文もこれを例にとる。ソフトウェアプロセスを時間 I/O オートマトンで記述する試みは、本論文の特徴である。ソフトウェアプロセスを変換対象とすることにより、本論文で提案する変換が実時間システム以外にも、適用可能であることを示す。そして、列車制御システムとソフトウェアプロセスに対して、時間 I/O オートマトン記述と変換後 (最適化後) の時間モジュール記述の比較を行う。

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 背景

通信システム、論理回路、オペレーティングシステムといった実時間システムの大規模化や多様化にともない、これらのシステムの設計段階における信頼性の検証が重要になってきている。これらの大規模なシステムでは、論理的動作の信頼性だけでなく、動作のタイミングの信頼性も要求されており、シミュレーションによるテストは困難である。そこで、設計されたシステムを数学的手法により信頼性を検証できることが望ましい。

このような背景のもとで、実時間システムの仕様記述および検証のために時間オートマトン [1] および時間モジュール [2] が提案されている。前者は、仕様記述の分野でよく知られているオートマトンについて、その枝に時間制約や時間のリセットを付加し拡張したものである。後者は、ブール変数の値の遷移を式で表現するリアクティブモジュール [4] に、時間に関する遅延や時間制約等の概念を付加し拡張したものである。この拡張により、真偽値に関することと時間に関することを、変数値の遷移という同一の枠組で扱うことが可能となった。

ところで、Alur らは、文献 [2] において、時間モジュールが時間オートマトンの一般化であるということを示唆してはいるが、後者から前者への具体的な変換手続きは与えていない。

1.2 本研究の目的

そこで、本論文では、時間オートマトンから時間モジュールへの変換について考察する。変換は、時間オートマトンから時間モジュールへの等価変換と、等価変換により得られた時間モジュールに対する最適化からなる。

なお、本論文では、変換対象を時間 I/O オートマトンとする。これは、時間オートマトンのイベントに入力と出力の区別を付加し拡張したものである。対象を時間 I/O オートマトンとする理由は、本論文で記述の対象とするシステムやプロセスでは、イベントにおける入力と出力の区別が必要だからである。

変換について順に述べる。まず、時間 I/O オートマトンから時間モジュールへの等価変換を行う。このとき、時間モジュールのブール変数を用いて、時間 I/O オートマトンの状態、イベントを表現する。また、時間モジュールの更新式 (変数値の遷移を表現する式) を用いて、時間 I/O オートマトンの枝を表現する。

次に、この等価変換より得られる時間モジュールに対して記述の最適化を行う。時間 I/O オートマトンにおいては、複数の入力や出力を同時に行うときに、冗長な状態や枝が必要となることが多い。一方、時間モジュールでは、そのような場合に、更新式で記述することにより、変数の数や更新式の数を、等価変換後の時間モジュールより、減らすことができる。

この変換の対象として、まず、実時間システムである列車制御システムを例に挙げる。これにより、本論文で提案する変換が実時間システムへ適用可能であることを示す。

また、変換の対象として、次に、ソフトウェアプロセスをとる。ソフトウェアプロセスとは、ソフトウェアシステムの設計、コーディング、単体試験等からなる一連のソフトウェア作成の工程のことをいう。ソフトウェアプロセス記述の例題としては、Kellner らによる「ソフトウェアプロセスモデリングのための例題」[3] が有名であり、本論文もこれを例にとる。ソフトウェアプロセスを時間 I/O オートマトンで記述する試みは、本論文の特徴である [5][6][7][8][9][10]。ソフトウェアプロセスを変換対象とすることにより、本論文で提案する変換が実時間システム以外にも、適用可能であることを示す。

そして、列車制御システムとソフトウェアプロセスに対して、時間 I/O オートマトン記述と変換後 (最適化後) の時間モジュール記述の比較を行う。

1.3 本論文の構成

本論文の構成は次のとおりである。まず、第 2 章において、仕様記述モデルである時間オートマトンと時間モジュールの定義を述べる。次に第 3 章において、時間 I/O オートマトンから時間モジュールへの変換について述べる。第 4 章では、前章で述べた変換について例を用いて説明する。第 5 章では、これまで述べてきた 2 つのモデルについて考察する。最後に第 6 章にて、本論文のまとめと今後の課題について述べる。

第 2 章

仕様記述モデル

本章では、Alur らによって提案された時間オートマトン [1][11] と時間モジュール [2] を定義する。まず、先に提案された時間オートマトンについて定義する。そして、後で提案された時間モジュールについて定義する。

2.1 時間オートマトン

本節では、オートマトンの拡張である時間オートマトン [1] に関して述べる。

時間オートマトン間の通信は、同じイベントで時間オートマトンを同期遷移させることにより表現される。実時間システムの仕様記述や、ソフトウェアプロセスの記述において、特に、ソフトウェアプロセスの記述には、同じイベントによる通信でデータのやりとりを表現することがあるため、イベントに入力と出力の区別が必要となる。しかし、Alur らが提案した時間オートマトンでは、イベントにおける入力と出力の区別が存在しない。このため、本論文では、時間オートマトンを拡張し、入力と出力の区別 [12] を追加した時間 I/O オートマトンを用いて記述を行う。

2.2 時間 I/O オートマトン

本節では、時間オートマトンに入力と出力の区別を追加した時間 I/O オートマトンを定義する。

以下に、時間 I/O オートマトンを構文、動作の順に定義する。

2.2.1 時間 I/O オートマトンの構文

時間 I/O オートマトン A は、$A = \langle \Sigma, S, S_0, C, E \rangle$ の組で定義される。
ここで、

Σ はイベントの有限集合、
S は状態の有限集合、
$S_0 \subseteq S$ は初期状態の集合、
C はクロック変数 (非負の実数の変数) の有限集合、
$E \subseteq S \times S \times \Sigma \times 2^C \times \delta$ は枝の有限集合

を表している。
さらに、Σ は、

内部イベントの集合 Int、
出力イベントの集合 Out、
入力イベントの集合 In

の 3 つに分割される。すなわち、$\Sigma = Int \cup Out \cup In, Int \cap Out \cap In = \emptyset$ である。

また、枝 $\langle s, s', \sigma, \lambda, \delta \rangle \in E$ について、s, s' は状態、σ はイベント、λ は初期化されるクロック変数の集合、δ は時間制約の集合である。時間制約 δ は、クロック変数の集合 C(非負の実数の変数の集合) の時間制約式 δ の集合であり、δ は $x \in C$ と $c \in Q^+$(Q^+ は非負の有理数の定数の集合) により、

$$\delta := x \leq c \mid c \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2$$

と帰納的に定められるとする。

2.2.2 時間 I/O オートマトンの動作

時間 I/O オートマトンの一つの動作列 $\langle s_0, \nu_0 \rangle, \langle s_1, \nu_1 \rangle, \ldots$ は、時間語 $\langle \sigma, \tau \rangle$ が帰納的に定める。ここで, $\langle s_0, \nu_0 \rangle, \langle s_1, \nu_1 \rangle, \ldots$ はそれぞれ、状態の列 $s_0, s_1, \cdots$、クロック解釈 (clock interpretation, 各クロック変数への値の割り付け) の列 $\nu_0, \nu_1, \cdots$ である。また、σ, τ はそれぞれ、イベントの列 $\sigma_1, \sigma_2, \cdots$, 時刻列 $\tau_1, \tau_2, \cdots$ である.
ここで、τ は

単調性 (monotonicity) : $\tau_{i+1} > \tau_1, \forall i \geq 1$
前進性 (progress) : $\tau_i > t, \forall t, \exists i \geq 1$

を満たすとする。

まず, 時刻 $0 = \tau_0$ において、s_0 は初期状態の一つであり、すべてのクロック変数を初期化して、クロック解釈 ν_0 とする。次に, 時刻 τ_i において、状態が s_i にあり、クロック解釈が ν_i であるとする。いま, 枝 $\langle s_i, s_{i+1}, \sigma_i, \lambda_i, \delta_i \rangle$, 時刻 τ_i が存在して、クロック解釈 ν_i の各要素に経過時間 $\tau_{i+1} - \tau_i$ を加えたクロック解釈 $\nu^\star$ が時間制約 δ_{i+1} を満たすとする。このとき、イベント σ_i を発生し、状態 s_{i+1} に遷移する。また、時刻 τ_{i+1} におけるクロック解釈 ν_{i+1} は、$\nu^\star$ を、λ_{i+1} のすべての要素を初期化する変更で得られるクロック解釈とする。

時間 I/O オートマトン A_1、A_2 が共通のイベント σ を共有する枝 e_1、e_2 をそれぞれもつとする。ある時刻において、e_1、e_2 の時間制約がともに満たされるとき、A_1 と A_2 はイベント σ によって同時に遷移する。これを A_1、A_2 の同期遷移という。同期遷移は、3 つ以上の時間 I/O オートマトン間においても定義される。

2.3 時間モジュール

本節では、リアクティブモジュール [4] の拡張である時間モジュール [2] について述べる。時間モジュールで用いられる変数には、入力と出力に対応する分別がされている。よって、以下に時間モジュールを構文、動作の順に定義する。

2.3.1 時間モジュールの構文

時間モジュール P は、$P = \langle X_P, \mathcal{A}_P \rangle$ の組で定義される。

ここで、

X_P は型付きの変数の有限集合、
$\mathcal{A}_P$ はアトムの集合

を表している。

X_P は、型から次の 2 つに分けられる。

$discX_P$　離散変数
$clkX_P$　クロック (非負の実数) 変数

また、X_P は、各時間モジュール間の相互作用の観点から次の 2 つに分けられる。

$ctrX_P$　時間モジュール P 自身が更新する, 制御変数の集合
$extlX_P$　環境 (時間モジュール P 以外のモジュール) により更新される, 外部変数の集合

さらに、制御変数の集合 $ctrX_P$ は、次の 2 つに分けられる ($privX_P \cup intfX_P = ctrX_P$)。

$privX_P$　モジュール P のみ参照可能な内部変数の集合
$intfX_P$　環境から参照可能なインターフェイス変数の集合

ここで、外部変数の集合とインターフェイス変数の集合は、環境から参照可能な変数の集合である。この集合を $obsX_P = intfX_P \cup extlX_P$ と定義する。型付きの変数の集合に関する分割を図示すると以下のようになる [4]。

表 2.1: 時間モジュール変数

<table>
<tr><td>$privX_P$</td><td>$intfX_P$</td><td>$extlX_P$</td></tr>
<tr><td colspan="2">$ctrX_P$</td><td></td></tr>
<tr><td></td><td colspan="2">$obsX_P$</td></tr>
<tr><td colspan="3">X_P</td></tr>
</table>

時間モジュール P は、宣言部と本体の 2 つから成る。

宣言部 型付きの変数 X_P($privX_P$、$intfX_P$、$extlX_P$) を宣言

本体 アトムの集合 $\mathcal{A}_P$

ここで、アトムの構文について示す。

アトムの構文

時間モジュール P のアトム A は、$A = \langle X_P, Init_A, Update_A, Delay_A \rangle$ の組で定義される。
ここで、

X_P は型付きの変数の有限集合、
$Init_A$ は初期アクション、
$Update_A$ は更新アクション、
$Delay_A$ は時間アクション

を表している。

アトム A において、X_P は、次の 2 つに分けられる。

制御変数の集合　$ctrX_A \subseteq X_P$
待機変数の集合　$waitX_A \subseteq X_P - ctrX_A$

また、アトム A は、宣言部と本体の 2 つから成る。

宣言 型付きの変数 $X_P(ctrX_A、waitX_A)$ の宣言を行う。

本体 初期アクション $Init_A$、更新アクション $Update_A$、時間アクション $Delay_A$ からなる。

ここで、初期アクション、更新アクションは、条件付きの更新式であり、時間アクションは、条件付きの不等式である。これらの 3 つのアクションの条件 · 更新に現われる変数は、それぞれ

$Init_A$	条件：	$waitX_A$
	更新：	$ctrX_A$
$Update_A$	条件：	$X_P \cup waitX_A$
	更新：	$ctrX_A$
$Delay_A$	条件：	$ctrX_A \cup waitX_A$
	更新：	$ctrX_A \cup waitX_A$

の変数である。アトム A において、x が制御変数、y が待機変数であるとき、かつ、そのときに限り、x は y を待つといい、$x \succ_A y$ と表す。

更新アクションには、round-insensitivity という条件が必要である。この round-insensitivity とは、待機変数が変化しなければ、制御変数は変化しないという性質である。

ここで、時間モジュールの定義を補足する。時間モジュールの本体は、アトムの集合 $\mathcal{A}_P$ であり、$ctrX_A$ と $\succ$ に関する次の 3 つの条件を満たす。

1. $ctrX_P = \cup_{A \in \mathcal{A}_P} ctrX_A$
2. 任意の 2 つの X_P アトム A, B に対し、$ctrX_A \cap ctrX_B = \emptyset$
3. $\succ_P = \cup_{A \in \mathcal{A}_P} \succ_A$ が反対称的。

命題 2.3.1 $\succ_P = \cup_{A \in \mathcal{A}_P} \succ_A$ が反対称的であることにより、アトムを次のような適合順 $A_0, A_1, \cdots, A_n$ に並べることができる:
$0 \leq i < j \leq n$ なる任意の 2 つの整数 i, j に対し、A_i の待機変数が A_j の制御変数でない。
□

ここで、時間モジュールの並列合成について定義する。

定義 2.3.1 (合成) 時間モジュール P、Q における $P \parallel Q$ を以下に示す。

内部変数 $privX_{P\parallel Q} = privX_P \cup privX_Q$

インターフェイス変数 $intfX_{P\parallel Q} = intfX_P \cup intfX_Q$

外部変数 $extlX_{P\parallel Q} = (extlX_P \cup extlX_Q) \backslash intfX_{P\parallel Q}$

アトム $\mathcal{A}_{P\parallel Q} = \mathcal{A}_P \cup \mathcal{A}_Q$ □

2.3.2 時間モジュールの動作

時間モジュールの動作は、本体で行われるため、アトムの集合である。したがって、以下にアトムの動作を示す。

アトムの動作

アトムは、アクションの繰り返しからなる。このアクションには、以下の 3 つが存在する。

初期アクション　入力変数の初期値を設定する。
更新アクション　X_P の変数値を更新する。ただし、時間は経過しない。
時間アクション　X_P の時間が経過する。ただじ、変数値の値は更新しない。

適合順 $A_0, A_1, \cdots, A_n$ において、A_0 は初期アクションである。また、A_1 以降は、更新アクションと時間アクションが交互に行われる。

まず、初期アクションの記述に従い、アトムは待機変数の各変数の初期値を待ち、そして、制御変数の各変数を初期化する。

以降は、時間アクションの記述により、遅延可能時間を決定する。遅延可能時間が 0 ならば、更新アクションの記述により、待機変数の値の更新を待ち、制御変数の値を更新する。更新は 0 時間で行われる。

各アクションについて、条件が複数成立するときには、非決定的に一つの更新式または不等式を選ぶ。条件が一つも成立しないときには、どの制御変数の値も更新されず、遅延も行われない。ただし、ある状態から複数の更新式が存在する場合、その更新式のうち、0 でない有限の数の更新式を行う。

第 3 章

時間I/Oオートマトンから時間モジュールへの変換

本章では、時間 I/O オートマトンから時間モジュールへの変換について述べる。
　変換は次の 2 つの部分からなる。

1. 時間 I/O オートマトンから時間モジュールへの等価変換
2. 時間モジュールの最適化

以下に、上記の 2 つを順に定義する。

3.1　時間I/Oオートマトンから時間モジュールへの等価変換

　本節では、時間 I/O オートマトンから時間モジュールへの等価変換法を与え、その正当性を以下に示す。

3.1.1　等価変換法

　入力となる時間 I/O オートマトン $A = \langle \Sigma, S, S_0, C, E \rangle$ において、Σ は内部イベント Int, 出力イベント Out, 入力イベント In に分割される。

宣言部

A に対し、出力となる時間モジュール P を次のように構成する。時間モジュール P の型付きの変数の集合 X_P は、時間 I/O オートマトンに対して以下のようになる。

$discX_P = \Sigma \cup \{state\}$

$clkX_P = C$

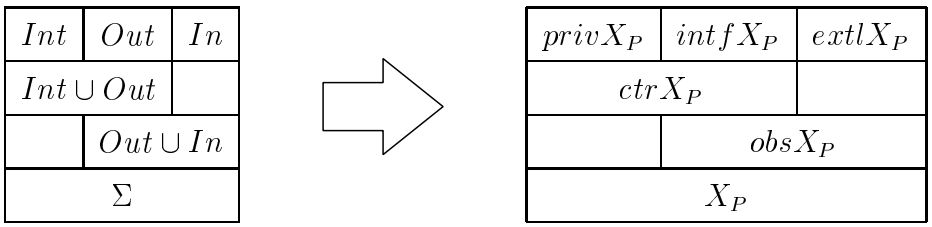
$privX_P = Int \cup \{state\} \cup C$

$intfX_P = Out$

$extlX_P = In$

時間モジュール変数についての対応表は、表 3.1 に示す。

表 3.1: 時間 I/O オートマトンのイベントと時間モジュールの変数



また、変数 $state \in privX_P$ のとる値の集合を、S とする。

本体 (初期アクション)

時間モジュールの本体の中の初期アクションでは、以下のことを行う。

- 内部変数とインターフェイス変数のそれぞれ σ について、$\sigma := false$ とする。

  $$\sigma_0 := false; \sigma_1 := false; \cdots \sigma_n := false$$

  これは、時間 I/O オートマトンにおいて、時刻 0 では、全てのイベントが発生していないということに対応する。

- $state := s_0$ とする。
  これは、時間 I/O オートマトンにおいて、初期状態を定義することに対応している。

- $\forall c \in C$ について $c := 0$ とする。
  これは、時間 I/O オートマトンにおいて、時刻 0 で各クロック変数を初期化するということに対応する。

本体 (更新アクション)

時間モジュールの本体の中の更新アクションは、時間 I/O オートマトンにおける枝の集合 E によって定める。時間 I/O オートマトンにおけるイベントによる動作は、時間モジュールの変数値の変化で表現する。各 $e \in E$ について、$e = \langle s, s', \sigma, \lambda, \delta \rangle$ のとき、σ が入力イベント In、出力イベント Out、内部イベント Int のどれかに属する。また、時間モジュールにおいて型付きの変数 X_P も同様に、外部変数 $extlX_P$、インターフェイス変数 $intfX_P$、内部変数 $privX_P$ のどれかに属する。ここで、時間 I/O オートマトンと時間モジュールの等価性の定義を与える。

定義 3.1.1 (等価性) 以下をもって、時間 I/O オートマトンと時間モジュールの等価性の定義とする。

時間 I/O オートマトン A と時間モジュール P は、以下を満たすとき、かつ、そのときに限り等価である。A において、状態 s でイベント σ を発生し、s' へ遷移するとき、時間モジュールにおいては、変数 $state = s$ で変数 σ の値を $true$ にし、変数 $state$ の値を s' にし、その後、変数 σ の値を $false$ に戻す。

ここで、更新式における型付きの変数への対応を以下のようにする。

外部変数 (入力イベント) その時間モジュール自身で更新不可能を行えないため、更新式の条件となる。
ここでは、時間 I/O オートマトンのイベントが生じていることを条件とするため、時間モジュールの外部変数が true であることを更新式の条件とする。

制御変数 (出力イベントと内部イベント) その時間モジュール自身で更新可能であるため、更新式の更新となる。
ここでは、時間 I/O オートマトンのイベントが生じていることを表すために、時間モジュールの制御変数の値を false から true へ更新する。

時間制約 時間モジュールにおいても時間制約として更新式の条件になる。

クロック変数の初期化 時間モジュールにおいても値の更新であるため、更新式の更新になる。

3 種類の型付き変数によって、それぞれ更新アクションが異なる。このように、内部変数 $privX_P$(時間 I/O オートマトンにおいて同期遷移しないイベント) と参照可能な変数 $obsX_P$(時間 I/O オートマトンにおいて同期遷移するイベント) に分ける。この 3 つの変数における更新式を表現すると次のようになる。

- σ が In であるとき、$\langle s, s', \sigma(in), \lambda, \delta \rangle$ は、
  $state = s \wedge \sigma(extl) = true \wedge \delta \longrightarrow \lambda; state := s'$ となる。
- σ が Out であるとき、$\langle s, s', \sigma(out), \lambda, \delta \rangle$ は、
  $state = s \wedge \delta \longrightarrow \sigma(intf) := true; \lambda; state := s'$ となる。
- σ が Int であるとき、$\langle s, s', \sigma(int), \lambda, \delta \rangle$ は、
  $state = s \wedge \delta \longrightarrow \sigma(priv) := true; \lambda; state := s'$ となる。

ここで、変数における更新アクションの補足をする。時間 I/O オートマトンにおけるイベントは、一度発生したら消滅する。しかし、時間モジュールにおける制御変数は、上記の式だけでは、時間 I/O オートマトンのイベントが発生したまま消滅しない状況になる。このため、上記の更新式の次の変数 $state = s'$ に対して、次の更新式の変更を行う。この更新式の変更は、内部変数とインターフェイス変数で異なる。これは、インターフェイス変数と他の時間モジュールの外部変数が相互作用をするからである。内部変数、インターフェイス変数と外部変数の順に以下に示す。

まず、内部変数では、更新式の変更を以下のように行う。

$$state = s' \wedge \sigma = true \longrightarrow \sigma := false$$

さらに、この状態 s' を更新式の条件にもつ式が複数存在する場合、その全ての更新式に対して次の下線部の条件を加える。

$$state = s' \wedge \underline{\sigma = false} \longrightarrow state := s''$$

ここで、時間 I/O オートマトンにおける入力イベント、出力イベントは、同期遷移するため、同名のイベントが全て遷移可能になるまで遷移を待っている。これに対し、時間モジュールにおける外部変数、インターフェイス変数について述べる。外部変数は、条件で

あるため、インターフェイス変数の値の変化を待つ。一方、インターフェイス変数は、更新であるため、外部変数における同名の変数が遷移可能でなくとも、変数の更新を行ってしまう。

このことから、インターフェイス変数と外部変数に対して、以下のような追加式の付加を行う。ここで、あるインターフェイス変数 σ を含む時間モジュールを $\mathcal{P}$ とし、その変数 σ により、$state = s$ から $state = s'$ へ遷移する。また、外部変数 σ を含む時間モジュールを $\mathcal{P}_i$ とする。ただし、$1 \leq i \leq n$ の自然数である。この変数 σ により、$state = s_i$ から $state = s_i'$ へ遷移する。

ここで、時間モジュール $\mathcal{P}$ に対して、$\mathcal{P}_i$ が σ による遷移が可能であることを伝えるために、$\mathcal{P}_i$ のインターフェイス変数として σ_i を用いる。まず、$\mathcal{P}_i$ が $state = s_i$ において変数 σ_i を true する更新式を以下のように追加する。

$$state = s_i \ \longrightarrow \ \sigma_i := true$$

また、$\mathcal{P}$ において、σ を true にする更新式に、全ての σ_i が true であるという条件を以下のように追加する。

$$state = s \wedge \underline{\sigma_1 = true \wedge \cdots \wedge \sigma_i = true \wedge \cdots \wedge \sigma_n = true} \ \longrightarrow \ \sigma := true; state := s'$$

そして、$\mathcal{P}_i$ において、σ の値が true による更新式に、σ_i の値を false にする更新を以下のように追加する。

$$state = s_i \wedge \sigma = true \ \longrightarrow \ \underline{\sigma_i := false;} state := s_i'$$

このあと、$\mathcal{P}$ は、σ の値を false にする更新式を変更するのだが、これは σ_i の値が全て false になった後に行う。この更新式の変更を以下に示す。

$$state = s' \wedge \sigma = true \wedge \underline{\sigma_1 = false \wedge \cdots \wedge \sigma_i = false \wedge \cdots \wedge \sigma_n = false} \ \longrightarrow \ \sigma := false$$

また、この付加した更新式の変更による更新式の更新がされない可能性がある。この更新式を必ず行うようにするために、$state = s'$ からの更新式の条件に σ の値が false であるという条件を以下のように追加する。

$$state = s' \wedge \underline{\sigma = false} \ \longrightarrow \ state := s"$$

本体 (時間アクション)

時間 I/O オートマトンには、状態に関する時間制約は存在しない。このことにより、時間モジュールの本体の中の時間アクションは、次のように変数 state を含む更新式から導出する。ある変数 state の値をもつ更新式の全ての時間制約の共通部分とする。ただし、変数 state の値を更新にもち、時間のリセットが存在する場合は、その変数 state の値における時間制約の下限を 0 とする。

ここで、2 つの更新式を次のように考える。

$$state = s \wedge \delta \ \longrightarrow \ \sigma := true; \lambda; state := s'$$

$$state = s' \wedge \sigma = true \wedge \delta' \ \longrightarrow \ \lambda'; state := s''$$

変数 state の値 s' に対して、時間制約 δ と δ' の共通部分を δ'' とするとき、時間アクションを導出すると次の式になる。

$$state = s' \wedge \delta'' \ \longrightarrow \ \delta''$$

変数 state の値 s' を含む更新式に時間制約が存在しない場合は、次の式になる。

$$state = s' \ \longrightarrow \ true$$

3.1.2 等価変換法の正当性

大域時間を T とする。時間 I/O オートマトンにおいて、初期状態の開始時は、$T = 0$ である。一方、時間モジュールでは、初期アクションの開始前に、T の値を 0 にしている。このため開始時は、同じく $T = 0$ である。

内部変数

ここから、時間モジュール自身で変更可能な内部変数に関して正当性の証明を行う。これは、時間 I/O オートマトンでは、内部イベントに対応する。

ここで、初期状態 (変数 state) を S_0 とし、S_0 から S_1 への遷移に関するイベント (変数) を σ_1 とする。時間 I/O オートマトンにおいては、初期状態から開始する前には、何も行っていない。一方、時間モジュールにおいては、更新アクションを行う前に、初期アクションにおいて、以下のような変数の値の更新を行っている。

$$state := S_0; \sigma_1 := false; \sigma_2 := false; \cdots; \sigma_{n-1} := false; \sigma_n := false;$$

$n \geq 1$ の自然数に対して、S_{n-1} から S_n への遷移を示す。状態 S_{n-1} において、$\sigma_{n-1} = false$ と仮定する。時間 I/O オートマトンにおいては、次の枝になる。

$$\langle S_{n-1}, S_n, \sigma_n, \emptyset, T = t_n \rangle$$

一方、時間モジュールにおいては、次の式になる。

$$state = S_{n-1} \wedge \sigma_{n-1} = false \wedge T = t_n \ \longrightarrow \ \sigma_n := true; state := S_n$$

上式の $\sigma_{n-1} = false$ は、帰納法の仮定より成立する。この後、時間 I/O オートマトンは、σ_n の発生後、S_n への遷移が終了すれば、σ_n は消滅する。時間モジュールは、変数 $state$ の値を S_n にした後、次の式を行う。

$$state = S_n \wedge \sigma_n = true \ \longrightarrow \ \sigma_n := false$$

ここで、変数 $state$ の値が S_n において $\sigma_n = false$ が成立する。

上記より、時間 I/O オートマトンの枝と時間モジュールの更新式の対応関係をここに示す。時間 I/O オートマトンの遷移 $\langle S_{n-1}, S_n, \sigma_n, \emptyset, T = t_n \rangle$ に対応する時間モジュールの式は、

$$state = S_{n-1} \wedge \sigma_{n-1} = false \wedge T = t_n \ \longrightarrow \ \sigma_n := true; state := S_n$$

$$state = S_n \wedge \sigma_n = true \ \longrightarrow \ \sigma_n := false$$

である。

ここで、$n = 1$ のとき、時間 I/O オートマトンにおいては、

$$\langle S_0, S_1, \sigma_1, \emptyset, T = t_1 \rangle$$

であり、初期状態 S_0 から状態 S_1 へ $T = t_1$ でイベント σ_1 が発生して遷移している。また、時間モジュールにおいては、

$$state = S_0 \wedge \sigma_0 = false \wedge T = t_1 \ \longrightarrow \ \sigma_1 := true; state := S_1$$

であり、更新アクション開始時は、初期アクション直後であるから、変数 $state$ の値は S_0 であり、σ_0 は存在しないものの、任意の制御変数が false であるため、条件を満たし、σ_1 の値を true にし、変数 $state$ の値を S_1 にしている。その後、

$$state = S_1 \wedge \sigma_1 = true \ \longrightarrow \ \sigma_1 := false$$

により、時間 I/O オートマトンにおけるイベントを消滅させている。

以上から、任意の $n \geq 1$ において、時間モジュールの 2 式が成立し、時間 I/O オートマトンの動作と等価であることがわかる。

外部変数とインターフェイス変数

ここからは、時間モジュールにおける外部変数とインターフェイス変数に対して行う。これは、時間 I/O オートマトンにおける入力イベントと出力イベントに対応する。

ここで、入力イベントが存在する時間 I/O オートマトンを A_i、出力イベントが存在する時間 I/O オートマトンを A とする。また、外部変数をもつ時間モジュールを P_i、インターフェース変数をもつ時間モジュールを P とする。このとき、$A(P)$ において、初期状態 (変数 state) を S_0 とし、S_0 から S_1 への遷移に関するイベント (変数) を σ_1 とする。また、A_i において、初期状態を S_{0_i} とし、S_{0_i} から S_{1_i} への遷移に関するイベントを σ_1 とし、P_i において、初期の変数 state を S_{0_i} とし、S_{0_i} から S_{1_i} への遷移する際に用いる変数を σ_{1_i} とする。時間 I/O オートマトンにおいては、初期状態から開始する前には、何も行っていない。一方、時間モジュールにおいては、更新アクションを行う前に、初期アクションにおいて、以下のような更新を行っている。P、P_i の順に示す。

$$state := S_0; \sigma_1(intf) := false; \sigma_2(intf) := false; \cdots; \sigma_{n-1}(intf) := false; \sigma_n(intf) := false;$$

$$state := S_{0_i}; \sigma_{1_i}(extl) := false; \sigma_{2_i}(extl) := false; \cdots; \sigma_{n-1_i}(extl) := false; \sigma_{n_i}(extl) := false;$$

$n \geq 1$ の自然数に対して、$A(P)$ における S_{n-1} から S_n への遷移と、$A_i(P_i)$ における S_{n-1_i} から S_{n_i} への遷移を示す。状態 S_{n-1} において、$\sigma_{n-1}(intf) = false$、また、状態 S_{n-1_i} において、$\sigma_{n-1}(extl) = false$、と仮定する。時間 I/O オートマトンにおいては、次のようになる。

$$\langle S_{n-1}, S_n, \sigma_n(out), \emptyset, T = t_n \rangle$$

$$\langle S_{n-1_i}, S_{n_i}, \sigma_n(in), \emptyset, T = t_n \rangle$$

一方、時間モジュールにおいては、次の式になる。

$$state = S_{n-1} \wedge \sigma_{n-1}(intf) = false \wedge \sigma_{n_i} \wedge T = t_n \ \longrightarrow \ \sigma_n(intf) := true; state := S_n$$

$$state = S_{n-1_i} \wedge \sigma_n(extl) = true \wedge T = t_n \ \longrightarrow \ \sigma_{n_i} := false; state := S_n$$

上式の $\sigma_{n-1}(intf) = false$ は、帰納法の仮定より成立する。この後、時間 I/O オートマトンは、σ_n の発生後、σ_n は消滅する。また、σ_n は、入力イベント、出力イベントの同名のイベントが同時に遷移する。それに関して、時間モジュールでは、以下の式を行う。

$$state = S_{n-1_i} \ \longrightarrow \ \sigma_{n_i}(intf) := true$$

$$state = S_n \wedge \sigma_n(intf) = true \wedge \sigma n_i(extl) = false \ \longrightarrow \ \sigma_n(intf) := false$$

ここで、変数 $state$ の値が S_n において $\sigma_n = false$ が成立する。

上記より、時間 I/O オートマトンの枝と時間モジュールの更新式の対応関係をここに示す。時間 I/O オートマトンの遷移

$$\langle S_{n-1}, S_n, \sigma_n(out), \emptyset, T = t_n \rangle$$

$$\langle S_{n-1_i}, S_{n_i}, \sigma_n(in), \emptyset, T = t_n \rangle$$

に対応する時間モジュールの式は、

$$state = S_{n-1} \wedge \sigma_{n-1}(intf) = false \wedge \sigma_{n_i} \wedge T = t_n \ \longrightarrow \ \sigma_n(intf) := true; state := S_n$$

$$state = S_n \wedge \sigma_n(intf) = true \wedge \sigma_{n_i}(extl) = false \ \longrightarrow \ \sigma_n(intf) := false$$

$$state = S_{n-1_i} \ \longrightarrow \ \sigma_{n_i}(intf) := true$$

$$state = S_{n-1_i} \wedge \sigma_n(extl) = true \wedge T = t_n \ \longrightarrow \ \sigma_{n_i} := false; state := S_n$$

である。

ここで、$n = 1$ のとき、時間 I/O オートマトンにおいては、

$$\langle S_0, S_1, \sigma_1(out), \emptyset, T = t_1 \rangle$$

$$\langle S_{0_i}, S_{1_i}, \sigma_1(in), \emptyset, T = t_1 \rangle$$

であり、初期状態 S_0 から状態 S_1 へ $T = t_1$ でイベント $\sigma_1(out)$ が発生し、かつ、同時に初期状態 S_{0_i} から状態 S_{1_i} へ $T = t_1$ でイベント $\sigma_1(in)$ が発生して遷移している。また、時間モジュールにおいては、

$$state = S_{0_i} \ \longrightarrow \ \sigma_{1_i}(intf) := true$$

$$state = S_0 \wedge \sigma_0(intf) = false \wedge \sigma_{1_i}(extl) = true \wedge T = t_1 \ \longrightarrow \ \sigma_1(intf) := true; state := S_1$$

$$state = S_{0_i} \wedge \sigma_1(extl) = true \wedge T = t_1 \ \longrightarrow \ \sigma_{1_i} := false; state := S_1$$

$$state = S_1 \wedge \sigma_1(intf) = true \wedge \sigma_{1_i}(extl) = false \ \longrightarrow \ \sigma_1(intf) := false$$

であり、更新アクション開始時は、初期アクション直後であるから、変数 $state$ の値は S_0、S_{0_i} であり、σ_0 の値を true にする前に、σ_{0_i} を true にすることにより、入力可能であることを示している。また、σ_0 は存在しないものの、初期アクションにより任意の制御変数が false であるため、条件を満たし、σ_1 の値を true にし、変数 $state$ の値を S_1、S_{1_i} にしている。インターフェイス変数の値は、全ての入力が終わったあとに、false に戻すことにより、時間 I/O オートマトンにおけるイベントを消滅させ、かつ、同期イベントの動作も表現している。

以上から、任意の $n \geq 1$ において、時間モジュールの 4 式が成立し、時間 I/O オートマトンの動作と等価であることがわかる。

これまでのことから、時間 I/O オートマトンから時間モジュールへの動作の等価性を証明した。以下に、逆に、時間モジュールから時間 I/O オートマトンへの動作の等価性を証明する。

時間モジュールでは、初期アクションの開始前に、T の値を 0 にしている。一方、時間 I/O オートマトンにおいて、初期状態の開始時は、$T = 0$ である。このため開始時は、同じく $T = 0$ である。どの変数に関しても、この等価変換では、時間 I/O オートマトンの動作と等価な時間モジュールを構成している。前述の証明により、時間 I/O オートマトンから時間モジュールへの動作の等価性が証明されたため、等価変換法から時間モジュールと時間 I/O オートマトンへの動作と等価であることがわかる。

これにより、時間 I/O オートマトンから時間モジュールへの等価変換の動作の双方向の等価性の証明を終了する。

時間 I/O オートマトンの動作例を下に示す。

$$s_0[x = 0] \xrightarrow{a(int),x<3} s_1[x = 2] \xrightarrow{b(int),x<5} s_2[x = 3]$$

ここで、上記のイベントに付加されている (int)、(out) は、それぞれ内部イベント、出力イベントを表現している。

時間 I/O オートマトンの枝は、

$$\langle s_0, s_1, a(int), \emptyset, x < 3\rangle,$$
$$\langle s_1, s_2, b(int), \emptyset, x < 5\rangle$$

である。

これに対し、時間モジュールの更新アクションは、

$$state = s_0 \wedge x < 3 \ \longrightarrow \ state := s_1; a(priv) := true,$$
$$a(priv) = true \ \longrightarrow \ a(priv) := false,$$

$$state = s_1 \wedge x < 5 \longrightarrow state := s_2; b(priv) := true,$$
$$b(priv) = true \longrightarrow b(priv) := false$$

である。ここで、上記の変数に付加されている (priv) は、それぞれ外部変数を表現している。

これを動作列 (状態とクロック解釈の対) で表現すると、

$$\langle s_0, x = 0 \rangle,$$
$$\langle s_1, x = 2 \rangle,$$
$$\langle s_2, x = 3 \rangle$$

となる。

時間モジュールの表現形式の式では、可視性に乏しいため、本論文では、下図の時間モジュールの時間オートマトン的表現を提案する。ここで、時間モジュールの動作を、以下

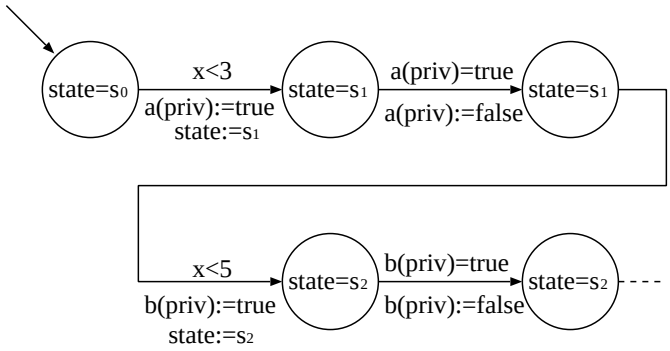


図 3.1: 時間モジュールの時間オートマトン的表現

のような変数値の集合の変化で表現する。

$$\{state = s_0, x = 0, a(priv) = false, b(priv) = false\}$$
$$\rightarrow \{state = s_1, x = 2, a(priv) = true, b(priv) = false\}$$
$$\rightarrow \{state = s_1, x = 2, a(priv) = false, b(priv) = false\}$$
$$\rightarrow \{state = s_2, x = 3, a(priv) = false, b(priv) = true\}$$
$$\rightarrow \{state = s_2, x = 3, a(priv) = false, b(priv) = false\}$$

このように、変数 $state$ と x をみると、時間 I/O オートマトンの動作は、変換した時間モジュールの動作と等しいことが分かる。

3.2 時間モジュールの最適化

本節では、2. の最適化について、最適化法を示し、その正当性を示す。

3.2.1 最適化法

入力となるのは、時間 I/O オートマトンから等価変換して得られた時間モジュールである。

この時間モジュールにおいて、一つの更新式には、イベントに対応する変数は高々一つしか存在しない。これは、時間 I/O オートマトンの一つの枝には、一つのイベントしかもつことができないためである。

最初の最適化を行なう前に、等価変換の時に付加した更新式の変更は、一時的に全て取り除き、最後に最適化による更新式の変更の付加を行う。ここで、本論文では、以下の順に最適化を行う。

最適化 (複数の入力) 複数の入力のための連続する更新式を一つにする。

最適化 (複数の出力) 複数の出力のための連続する更新式を一つにする。

最適化 (複数の入力順) 入力される順序がわからないためにされた場合分けを一つの更新式にする。

最適化 (入力と出力) 変数値による条件と更新の両方を含んだ一つの更新式を構成する。

最適化 (判定結果) 判定結果による場合分けのための複数の変数を一つの変数の異なる値の違いにより表現する。

最適化 (更新式の変更) 最初に取り除いた更新式の変更を再び付加する。

これらの最適化について、以下に述べる。

最適化 (複数の入力)

まず、時間 I/O オートマトンの入力イベントである、時間モジュールにおける外部変数をもつ更新式について最適化を行う。

更新アクションに記述されている更新式の順に以下のことを行う。ここで、ある変数 state の値を S_{n-1} から S_n へ更新する更新式を A とする。さらに、S_n から S_{n+1} へ更新する更新式を B とする。

これらの更新式 A と B の例を順に示す。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \delta_n \longrightarrow state := S_n$$

$$state = S_n \wedge \sigma_{n+1} = true \wedge \delta_n \longrightarrow state := S_{n+1}$$

次に述べる 4 つの条件を満たすような上記の更新式 A と B が見つかるまで、順に確認し続ける。ただし、この確認は、更新アクションの全ての更新式の確認後、終了する。ここで、更新式 A と B は、以下の条件を満たしたものである。

- A と B の条件に外部変数が含まれている。
- A と B の時間制約が同じ。
- A の更新に時間のリセットが存在しない (B は時間のリセットが存在してもよい)。
- 変数 state の値を S_n へ更新する式が A のみである。

ただし、この 4 つ目の条件に関しては、一度全ての更新式の確認をしなければならない。この条件を満たしている場合、下式のような一つの新しい更新式 C を構成する。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \sigma_{n+1} = true \wedge \delta_n \longrightarrow state := S_{n+1}$$

再び初めの条件の確認を継続するのだが、C が A になることがあるため、C から継続する。

最適化 (複数の出力)

二つ目として、時間 I/O オートマトンの出力イベントであった時間モジュールのインターフェイス変数をもつ更新式について最適化を行う。

この最適化は、**複数の入力**とほぼ同じことを行なう。ただし、『入力』と『出力』の違いから、『外部変数』であるところが、『インターフェイス変数』になり、それにともない、『条件』であるところが、『更新』になる。

$$state = S_{n-1} \wedge \delta_n \longrightarrow \sigma_n := true; state := S_n$$

$$state = S_n \wedge \delta_n \longrightarrow \sigma_{n+1} := true; state := S_{n+1}$$

上式の更新式 A と B が、4 つの条件を全て満たしている場合、下式のような一つの新しい更新式 C を構成する。

$$state = S_{n-1} \wedge \delta_n \longrightarrow \sigma_n := true; \sigma_{n+1} := true; state := S_{n+1}$$

最適化 (複数の入力順)

入力は他の時間モジュールからのものであり、その入力が複数個存在する場合、それらの入力は他の一つの時間モジュールからの入力とは限らない。複数個の入力は、他の時間モジュールが、どの順序で入力してくるか分からない。そのために、時間 I/O オートマトンでは、先にどの入力が行われても良いように、その状態から入力の数の枝の数を増やす必要があった。それにともない、状態数も増加してしまっている。**複数の入力**から考えると、A と B の変数が逆の入力順も考えて、A' と B' も加えた 4 式を順に下に示す。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \delta_n \longrightarrow state := S_n$$

$$state = S_n \wedge \sigma_{n+1} = true \wedge \delta_n \longrightarrow state := S_{n+1}$$

$$state = S_{n-1} \wedge \sigma_{n+1} = true \wedge \delta_n \longrightarrow state := S_n$$

$$state = S_n \wedge \sigma_n = true \wedge \delta_n \longrightarrow state := S_{n+1}$$

上の 4 式は、**複数の入力**により、下の 2 式になる。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \sigma_{n+1} = true \wedge \delta_n \longrightarrow state := S_{n+1}$$

$$state = S_{n-1} \wedge \sigma_{n+1} = true \wedge \sigma_n = true \wedge \delta_n \longrightarrow state := S_{n+1}$$

この 2 つの式は、記述の順序が異なるだけで、同じ式である。このため、これらの式をどちらかの式で表現する。

最適化 (入力と出力)

ここまでの最適化の手法によって、複数個の更新式から一つの更新式を構成した。ここで、入力に関する更新式 A と、制御変数による更新式 B を順に示す。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \delta_n \longrightarrow state := S_n$$

$$state = S_n \wedge \delta_n \longrightarrow \sigma_{n+1} := true; state := S_{n+1}$$

この更新式 A と B は、以下の条件を満たしている。

- A の更新に時間のリセットが存在しない。
- A と B の時間制約が同じ。
- B が複数個存在しない。

ただし、この 3 つ目の条件に関しては、一度全ての更新式の確認をしなければならない。この条件を満たしている場合、下式のような一つの新しい更新式 C を構成する。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \delta_n \ \longrightarrow \ \sigma_{n+1} := true; state := S_{n+1}$$

最適化 (判定結果)

時間 I/O オートマトンにおいて、判定結果による場合分けはそれぞれ別々のイベントが必要となる。これは、等価変換後の時間モジュールにも反映されている。反映された更新式 A と B と C の例を下に示す。ただし、これらの更新式の時間制約は全て同じである。

$$state = S_n \wedge \sigma_{n+1} = true \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \sigma'_{n+1} = true \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \sigma''_{n+1} = true \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

このような場合、変数 state と同様の変数 σ^*_{n+1} に、σ_{n+1} と σ'_{n+1} と σ''_{n+1} の値をもたせる。この変数を用いることにより、下の下線部のように一つの変数で表現する。

$$state = S_n \wedge \underline{\sigma^*_{n+1} = \sigma_{n+1}} \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \underline{\sigma^*_{n+1} = \sigma'_{n+1}} \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \underline{\sigma^*_{n+1} = \sigma''_{n+1}} \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

最適化 (更新式の変更)

ここで、最初に一時的に取り除いた更新式の変更を付加する。ここで行う付加も、等価変換で行なった付加と同じように行う。

3.2.2 最適化法の正当性

大域時間を T とする。最適化前の時間モジュールにおいて、初期アクション開始前に、T の値を 0 にしている。同様に、最適化後の時間モジュールでも、初期アクションの開始前に、T の値を 0 にしている。このため初期アクション開始時は、同様に $T = 0$ である。

最適化 (複数の入力)

最適化を行う際の時間制約は、条件より同じであるため、最適化前の時間モジュールは下式のようになる。

$$state = S_{n-1} \wedge \sigma_n = true \wedge T = t_n \longrightarrow state := S_n$$

$$state = S_n \wedge \sigma_{n+1} = true \wedge T = t_n \longrightarrow state := S_{n+1}$$

また、最適化後の式は、下のようになる。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \sigma_{n+1} = true \wedge T = t_n \longrightarrow state := S_{n+1}$$

したがって、同時刻に 2 つの更新式を別々に行うことを、一度に行っているだけにすぎないため、動作の等価性は保たれている。

最適化 (複数の出力)

最適化としては、**複数の入力**と同じことを行っているため、動作の等価性は保たれている。

最適化 (複数の入力順)

最適化を行う際の時間制約は、条件より同じであるため、最適化前の時間モジュールは下式のようになる。

$$state = S_{n-1} \wedge \sigma_n = true \wedge \sigma_{n+1} = true \wedge T = t_n \longrightarrow state := S_{n+1}$$

$$state = S_{n-1} \wedge \sigma_{n+1} = true \wedge \sigma_n = true \wedge T = t_n \longrightarrow state := S_{n+1}$$

このように、同時刻に同じ動作をする複数の式は、最適化後にその中の一つの更新式だけを残す。同じ動作の式は、一つでも動作の等価性は保たれている。

最適化 (入力と出力)

最適化を行う際の時間制約は、条件より同じであるため、最適化前の時間モジュールは下式のようになる。

$$state = S_{n-1} \wedge \sigma_n = true \wedge T = t_n \ \longrightarrow \ state := S_n$$

$$state = S_n \wedge T = n \ \longrightarrow \ \sigma_{n+1} := true; state := S_{n+1}$$

この 2 つの式の最適化後の更新式は、下のようになる。

$$state = S_{n-1} \wedge \sigma_n = true \wedge T = t_n \ \longrightarrow \ \sigma_{n+1} := true; state := S_{n+1}$$

したがって、同時刻に 2 つの更新式を別々に行うことを、一度に行っているだけにすぎないため、動作の等価性は保たれている。

最適化 (判定結果)

最適化を行う際の時間制約は、条件より同じであるため、最適化前の時間モジュールは下式のようになる。

$$state = S_n \wedge \sigma_{n+1} = true \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \sigma'_{n+1} = true \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \sigma''_{n+1} = true \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

また、最適化後の更新式は、下のようになる。

$$state = S_n \wedge \underline{\sigma^*_{n+1} = \sigma_{n+1}} \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \underline{\sigma^*_{n+1} = \sigma'_{n+1}} \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

$$state = S_n \wedge \underline{\sigma^*_{n+1} = \sigma''_{n+1}} \wedge \delta_{n+1} \ \longrightarrow \ state := S_{n+1}$$

このように、変数の値の変化がある更新式のみで行っていた動作が、最適化後は、ある一つの変数の値によって行う更新式が決まっている。したがって、最適化による動作の等価性は保たれている。

第 4 章

変換の適用例

本章では、列車制御システムとソフトウェアプロセスに対して、本論文で提案している変換を適用する。

4.1 列車制御システム

この列車制御システムは、実時間システムの仕様の一例として挙げる。そして、このシステムに本論文における変換を適用する [13]。

4.1.1 仕様

本節では、列車制御システムの仕様について述べる。このシステムは、列車とゲートの 2 つの部分から成る。この 2 つはそれぞれ相互作用があり、その相互作用により成り立つ。以降に、列車とゲートについての仕様を述べる。

列車

列車は、ゲートに接近するまで前進し続ける。そして、ゲートに接近したら、接近していることをゲートに通知する。次に、ゲートが下りてからゲートに入る。ゲートが下りるまでにゲートに入りそうになった場合は、入る前に停車して待機する。ただし、ゲートに接近してから遅くとも 5 分以内にゲートを出る。ゲートを通過したら、再びゲートに接近するまで前進し続ける。

ゲート

列車が接近するまで待機する。列車が接近したら、ゲートを下ろす。このとき、列車が接近してからゲートを下ろすまでの時間は 1 分〜2 分とする。列車が通過したら、ゲートを上げる。ゲートを上げ終えたら、再び列車が接近するまで待機する。

4.1.2 時間 I/O オートマトンによる記述

本節で、列車制御システムの仕様を時間 I/O オートマトンで記述する。

まず、時間 I/O オートマトン **列車** $= \langle \Sigma_{\textbf{列車}}, S_{\textbf{列車}}, S_{0_{\textbf{列車}}}, C_{\textbf{列車}}, E_{\textbf{列車}} \rangle$ に対して、

$$
\begin{aligned}
&\Sigma_{\textbf{列車}} = \{\, \textbf{接近}\ (out), \textbf{下げる}\ (in), \textbf{入る}\ (int), \textbf{出る}\ (out)\} \\
&S_{\textbf{列車}} = \{0, 1, 2, 3\} \\
&S_{0_{\textbf{列車}}} = \{0\} \\
&C_{\textbf{列車}} = \{x\} \\
&E_{\textbf{列車}} = \left\{ \begin{array}{l}
\langle 0, 1, \textbf{接近}\ (out), \{x\}, \emptyset \rangle, \\
\langle 1, 2, \textbf{下げる}\ (in), \emptyset, \{x \leq 5\} \rangle, \\
\langle 2, 3, \textbf{入る}\ (int), \emptyset, \{x \leq 5\} \rangle, \\
\langle 3, 0, \textbf{出る}\ (out), \emptyset, \{x \leq 5\} \rangle
\end{array} \right\}
\end{aligned}
$$

とする。

また、時間 I/O オートマトン **ゲート** $= \langle \Sigma_{\textbf{ゲート}}, S_{\textbf{ゲート}}, S_{0_{\textbf{ゲート}}}, C_{\textbf{ゲート}}, E_{\textbf{ゲート}} \rangle$ に対して、

$$
\begin{aligned}
&\Sigma_{\textbf{ゲート}} = \{\, \textbf{接近}\ (in), \textbf{下げる}\ (out), \textbf{出る}\ (in), \textbf{上げる}\ (int)\} \\
&S_{\textbf{ゲート}} = \{0, 1, 2, 3\} \\
&S_{0_{\textbf{ゲート}}} = \{0\} \\
&C_{\textbf{ゲート}} = \{y\} \\
&E_{\textbf{ゲート}} = \left\{ \begin{array}{l}
\langle 0, 1, \textbf{接近}\ (in), \{y\}, \emptyset \rangle, \\
\langle 1, 2, \textbf{下げる}\ (out), \emptyset, \{1 \leq y \leq 2\} \rangle, \\
\langle 2, 3, \textbf{出る}\ (in), \emptyset, \emptyset \rangle, \\
\langle 3, 0, \textbf{上げる}\ (int), \emptyset, \emptyset \rangle
\end{array} \right\}
\end{aligned}
$$

とする。

上記の記述の図的表現を図 4.1 に示す。図中の記述で、イベントに付加された (in)、(out)、(int) は、それぞれ入力イベント、出力イベント、内部イベントを示している。

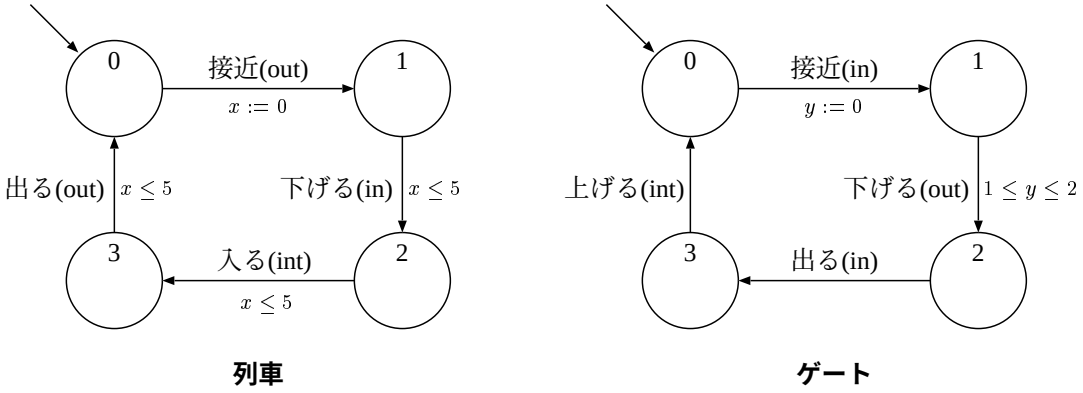


図 4.1: 時間 I/O オートマトンの図的表現 (列車制御システム)

4.1.3 時間モジュールへの等価変換

本節で、列車制御システムを時間 I/O オートマトンで記述したものを等価変換する。等価変換は、3.1 節の手法に従って行う。以下に変換後の時間モジュールを記述する。

module 列車 (列車制御システム)

external 下げる, 接近 1, 出る 1;
interface 接近, 出る, 下げる 1;
private state={0, 1, 2, 3}, 入る, x;

init

state:=0; 接近:=false; 出る:=false; 下げる 1:=false; 入る:=false;
x:=0;

update

| state=0 ∧ 出る 1=false ∧ 出る=true → 出る:=false
| state=0 ∧ 接近 1=true ∧ 出る=false → 接近:=true; x:=0;
state:=1
| state=1 ∧ 接近=true ∧ 接近 1=false ∧ $x \leq 5$ → 接近:=false
| state=1 ∧ $x \leq 5$ → 下げる 1:=true
| state=1 ∧ 下げる=true ∧ 接近=false ∧ $x \leq 5$ → 下げる 1:=false; state:=2

| state=2 $\wedge$ $x \leq 5$ $\rightarrow$ 入る:=true; state:=3

| state=3 $\wedge$ 入る=true $\wedge$ $x \leq 5$ $\rightarrow$ 入る:=false

| state=3 $\wedge$ 出る 1=true $\wedge$ 入る=false $\wedge$ $x \leq 5$ $\rightarrow$ 出る:=true; state:=0

delay

| state=0 $\rightarrow$ true

| state=1 $\wedge$ $x \leq 5$ $\rightarrow$ $x \leq 5$

| state=2 $\wedge$ $x \leq 5$ $\rightarrow$ $x \leq 5$

| state=3 $\wedge$ $x \leq 5$ $\rightarrow$ $x \leq 5$

図 4.2: 等価変換後の時間モジュール (列車制御システム : **列車**)

module ゲート (列車制御システム)

external 接近, 出る, 下げる 1;
interface 下げる, 接近 1, 出る 1;
private state={0, 1, 2, 3}, 上げる, y;

init

state:=0; 下げる:=false; 接近 1:=false; 出る 1:=false; 上げる:=false; y:=0;

update

| state=0 $\rightarrow$ 接近 1:=true

| state=0 $\wedge$ 上げる=true $\rightarrow$ 上げる:=false

| state=0 $\wedge$ 上げる=false $\wedge$ 接近=true $\rightarrow$ 接近 1:=false; y:=0; state:=1

| state=1 $\wedge$ 下げる 1=true $\wedge$ $1 \leq y \leq 2$ $\rightarrow$ 下げる:=true; state:=2

| state=2 $\wedge$ 下げる 1=false $\wedge$ 下げる=true $\rightarrow$ 下げる:=false

| state=2 $\rightarrow$ 出る:=false

| state=2 $\wedge$ 下げる=false $\wedge$ 出る=true $\rightarrow$ 出る 1:=false; state:=3

| state=3 $\rightarrow$ 上げる:=true; state:=0

delay

| state=0 → true

| state=1 ∧ $1 \leq y \leq 2$ → $1 \leq y \leq 2$

| state=2 → true

| state=3 → true

図 4.3: 等価変換後の時間モジュール (列車制御システム : **ゲート**)

さらに、等価変換後の時間モジュールの図的表現を図 4.4 に示す。

4.1.4 時間モジュールの最適化

本節では、等価変換を行うことによって得られた時間モジュールの最適化を行う。最適化は、3.2 節の手法に従って行う。

module 列車 (列車制御システム)

external 下げる, 接近 1, 出る 1;
interface 接近, 出る, 下げる 1;
private state={0, 1, 2}, 入る, x;

init

state:=0; 接近:=false; 出る:=false; 下げる 1:=false; 入る:=false; x:=0;

update

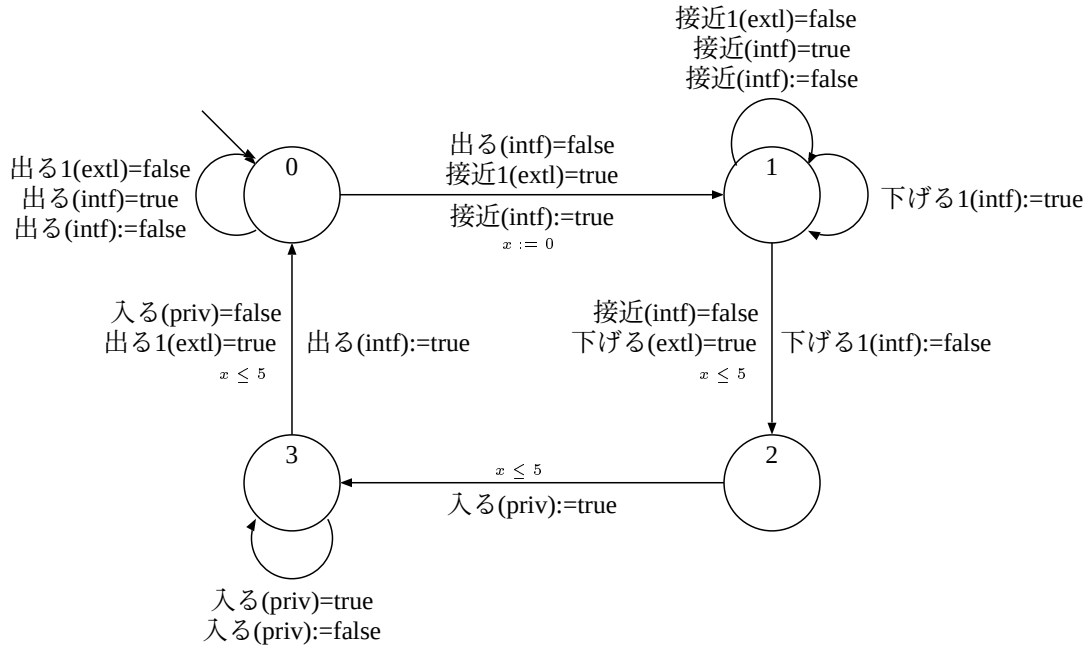
| state=0 ∧ 出る 1=false ∧ 出る=true → 出る:=false

| state=0 ∧ 接近 1=true ∧ 出る=false → 接近:=true; x:=0; state:=1

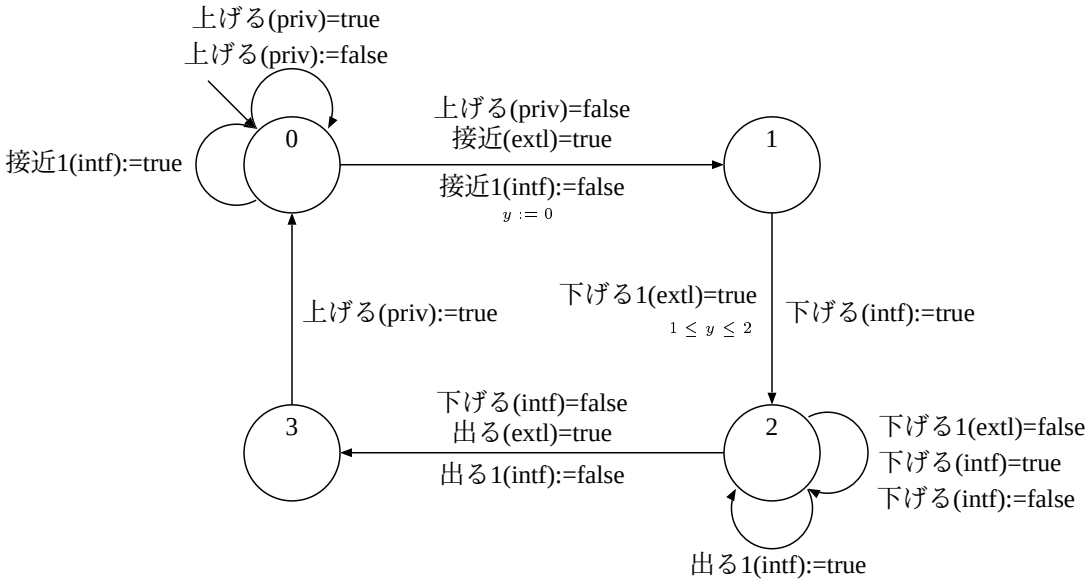
| state=1 ∧ 接近=true ∧ 接近 1=false ∧ $x \leq 5$ → 接近:=false

| state=1 ∧ $x \leq 5$ → 下げる 1:=true

| state=1 ∧ 下げる=true ∧ 接近=false ∧ $x \leq 5$ → 下げる 1:=false; 入る:=true; state:=2



列車



ゲート

図 4.4: 等価変換後の時間モジュールの図的表現 (列車制御システム)

| state=2 ∧ 入る=true ∧ $x \leq 5$ → 入る:=false

| state=2 ∧ 出る 1=true ∧ 入る=false ∧ $x \leq 5$ → 出る:=true; state:=0

delay

| state=0 → true

| state=1 ∧ $x \leq 5$ → $x \leq 5$

| state=2 ∧ $x \leq 5$ → $x \leq 5$

図 4.5: 最適化後の時間モジュール (列車制御システム : **列車**)

module ゲート (列車制御システム)

external 接近, 出る, 下げる 1;
interface 下げる, 接近 1, 出る 1;
private state={0, 1, 2}, 上げる, y;

init

state:=0; 下げる:=false; 接近 1:=false; 出る 1:=false; 上げる:=false; y:=0;

update

| state=0 → 接近 1:=true

| state=0 ∧ 上げる=true → 上げる:=false

| state=0 ∧ 上げる=false ∧ 接近=true → 接近 1:=false; y:=0; state:=1

| state=1 ∧ 下げる 1=true ∧ $1 \leq y \leq 2$ → 下げる:=true; state:=2

| state=2 ∧ 下げる 1=false ∧ 下げる=true → 下げる:=false

| state=2 → 出る:=false

| state=2 ∧ 下げる=false ∧ 出る=true → 出る 1:=false; 上げる:=true; state:=0

delay

$|$ state=0 $\rightarrow$ true

$|$ state=1 $\wedge$ $1 \leq y \leq 2 \rightarrow 1 \leq y \leq 2$

$|$ state=2 $\rightarrow$ true

図 4.6: 最適化後の時間モジュール (列車制御システム : **ゲート**)

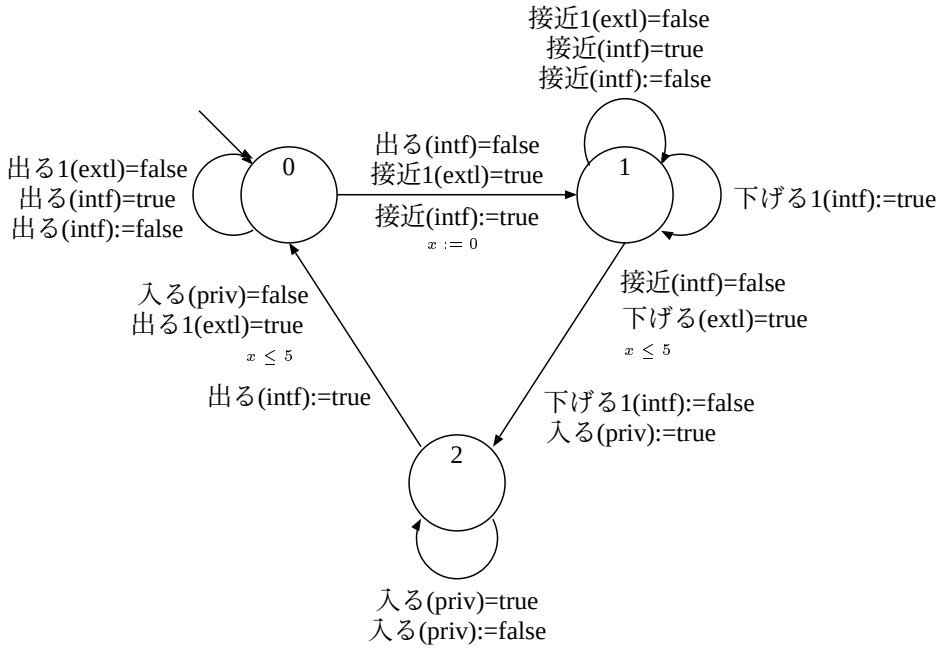
最適化後の時間モジュールの図的表現を図 4.7 に示す。

4.2 ソフトウェアプロセス

本節では、ソフトウェアプロセスモデリングのための例題をとりあげる。この例題は、ソフトウェアプロセスの比較的限定された部分に注目していて、ソフトウェアシステムの設計、コーディング、単体試験、比較的局所的な変更の管理に焦点を当てている。本論文では、このソフトウェアプロセスモデリングのための例題の核問題と、一部の拡張問題について記述する。この例題プロセスの詳細が隠蔽された最も抽象的なステップとして、**変更の実行および試験プロセス**がある。このステップから部分ステップへ分解したものを以下に示す。

- タスクのスケジューリングおよび割当プロセス (Schedule and Assign Tasks, SAT)
- デザイン変更プロセス (Modify Design, MD)
- デザインレビュープロセス (Review Design, RD)
- コード変更プロセス (Modify Code, MC)
- 試験計画の変更プロセス (Modify Test Plan, MTP)
- 単体試験パッケージの変更プロセス (Modify Unit Test Package, $MUTP$)
- 単体試験プロセス (Test Unit, TU)
- 進捗状況管理プロセス (Monitor Progress, MP)

本論文では、上記の 8 つの部分ステップについて、3 章の変換に基づいて行う。ここで、上記の 8 つの部分ステップについて説明する [14]。



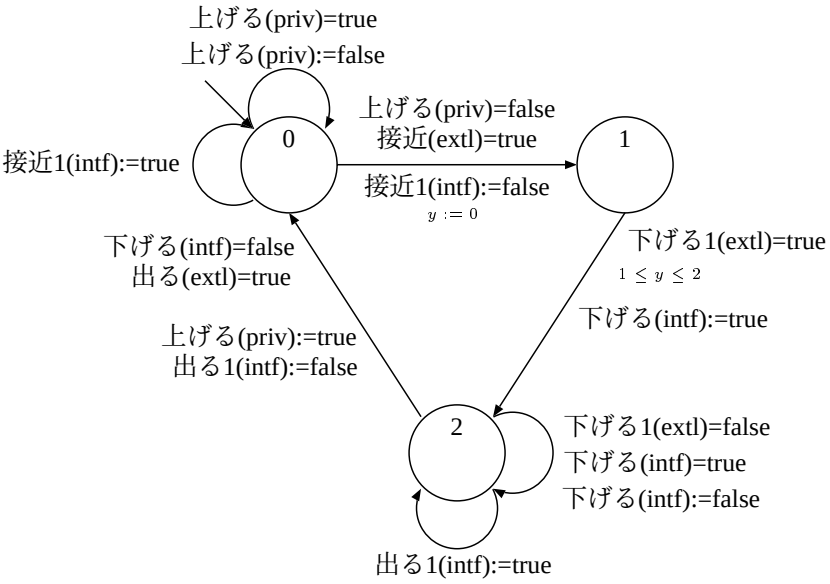


図 4.7: 最適化後の時間モジュールの図的表現 (列車制御システム)

タスクのスケジューリングおよび割当プロセス　この部分ステップは、プロジェクトを管理する機能であり、この例題プロセスで最初に実行されるものである。また、このソフトウェアの変更に関する作業のスケジュールを作成し、個々のタスクを指定されたスタッフのメンバに割り当てる作業を含む。

デザイン変更プロセス　この部分ステップは、要求変更により影響を受ける単体コードユニットのデザインの修正を行う。修正されたデザインはレビューされ、最後にコードという形で実現される。デザインレビューのステップからのフィードバックに基づいたデザインの修正も行う。

デザインレビュープロセス　この部分ステップは、修正されたデザインの正式なレビューを行うものであり、デザインの修正を行ったチームによって行われる。レビューの結果は、無条件合格、小変更要請、大変更要請、のうちのどれかである。レビューが終了すると、プロジェクトマネージャに結果が通知される。ここでは、発見された誤りの数と、レビューの準備し、実行しているレビューチームの全体の作業がプロジェクトマネージャに報告される。

コード変更プロセス　この部分ステップは、デザインの変更をコードに実現し、修正されたソースコードをオブジェクトコードにコンパイルする。ソースコードの追加修正の要求を示す試験結果のフィードバックに基づいて行われる場合もある。

試験計画の変更プロセス　この部分ステップは、ソフトウェアの修正のきっかけとなる要求変更に関連した機能の試験を含む試験計画及び目的の修正を行う。これらの試験計画はソフトウェアデザインに似ている。試験されるべき機能と能力、試験の方法を規定する。一方で、実際の試験データや手続きなどは、次の MUTP で扱われる。

単体試験パッケージの変更プロセス　この部分ステップは、試験計画と目的のために修正に従って、影響を受けたコードユニットのための実際の試験パッケージの修正を行う。試験パッケージは自動化された構成管理の下におかれ、単体試験パッケージの新しいバージョンはこのステップが終了した時に作成される。修正されたデザイン及びこのユニットのためのソースコードはこのステップへの入力としても使われることもある。このステップの次の繰り返しは、単体試験パッケージの追加修正が要求されたことを示す試験からのフィードバックに基づいて行われる。

単体試験プロセス　この部分ステップは、修正されたコード上の単体試験パッケージの実行と、その結果の解析を行う。試験パッケージ全体の実行が解析やその先の活動に先だって行われる。単体試験は有用であるが、単体コードに対する妥当なカバレージの達成を試すために自動化されたカバレージ解析器が用いられており、カバレージが 90%に達すると受け入れられる。全ての試験が完了し、カバレージが 90%に達すると単体試験は終了する。さもなければ、試験の結果を解析し、適切な活動を決定する。ソースコードの修正、単体試験の修正のどちらか一方あるいは両方が必要であるということを解析の結果とする。指定された修正が行われると、ユニットは再試験される。

もしユニットが試験にパスし、許容し得る達成率を満たせば、この例題プロセスは完了する。プロジェクトマネージャには、ユニットコードの最新バージョンがインテグレーション試験に利用できることが通知される。

進捗状況管理プロセス　この部分ステップは、プロジェクトマネージャによるタスクの進行状態のモニタリングを行う。これは、各ステップの終了通知や、その他の非公式な情報に基づいて行われる。タスクが計画に沿って実行されている間は、このステップでは何も起こらず何も出力されない。しかし計画からの逸脱が生じると、タスクの再スケジューリングが行われる。逸脱が深刻な場合には、プロジェクトマネージャは全ての修正作業を中断し、構成管理委員会 (以下、CCB と略す) にこの変更作業を中止するように勧告する。CCB は中止に同意するか、中断した箇所からの再開を指示する。このような場合、関係者は彼らの作業について、その再開または中止をプロジェクトマネージャに通知される。

これらのステップのフロー図を図 1 に示す。

4.2.1　概要

本節では、前で述べた 8 つの部分プロセスの中の単体試験 (TU) プロセスを用いる。

ここで、以下に示される状況を簡単に説明する。まず、単体試験プロセスは、オブジェクトコードと単体試験パッケージが共に完成し次第、開始する。そして、単体試験を実行し、試験結果を試験履歴ファイルに保存する。試験結果が成功の時には、成功した試験の通知を進捗状況管理プロセスに送信し、単体試験プロセス終了を進捗状況管理プロセスに通知し、このプロセスは終了する。一方、試験結果が失敗の時は、コードのフィードバックをコード変更プロセスへ、単体試験パッケージのフィードバックを単体試験パッケージ変更プロセスへ送信し、単体試験プロセス終了を進捗状況管理プロセスに通知しこのプ

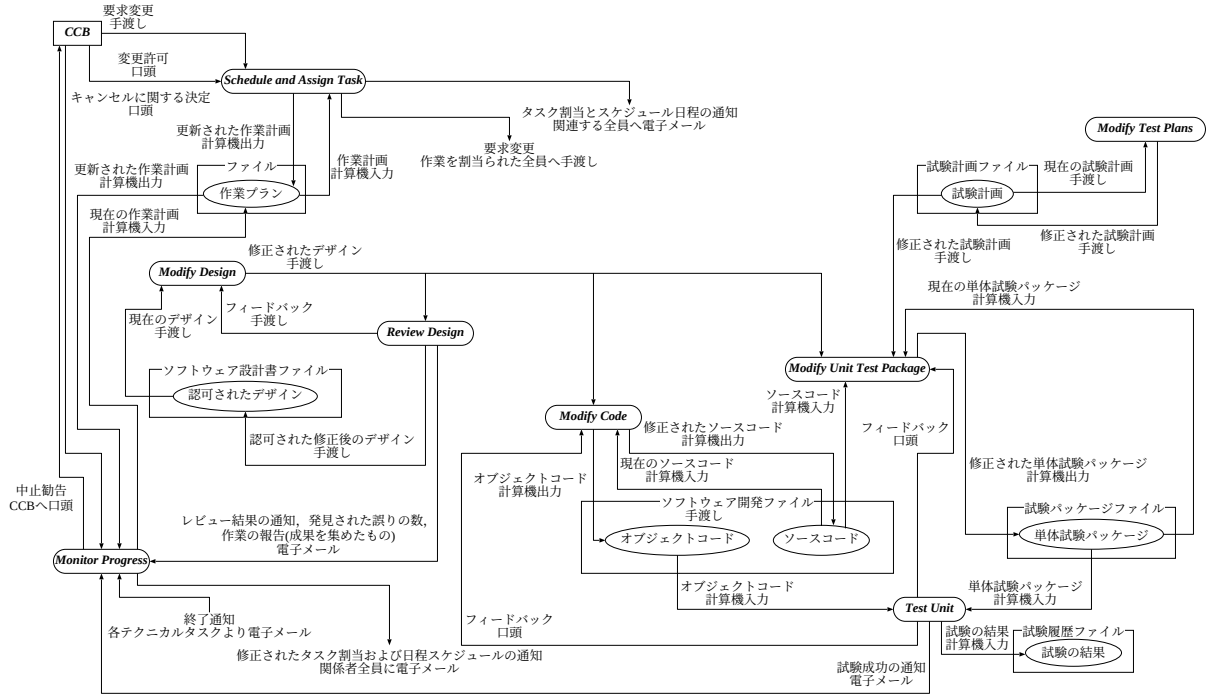


図 4.8: ソフトウェアプロセスモデリングの例題のフロー図

ロセスは終了する。このプロセスでは、完成したオブジェクトコードと完成した単体試験パッケージが入力イベントである。また、試験結果と成功した試験の通知とコードのフィードバックと単体試験パッケージのフィードバックと単体試験プロセス終了が出力イベントである。そして、単体試験プロセス開始と成功と失敗が内部イベントである。この部分プロセスにおいて、時間 I/O オートマトンで記述し、本論文における変換を適用する [15]。

4.2.2 時間 I/O オートマトンによる記述

本節で、単体試験プロセスを時間 I/O オートマトンで記述する。

まず、時間 I/O オートマトン 単体試験プロセス $= \langle \Sigma_{\text{単体試験プロセス}}, S_{\text{単体試験プロセス}}, S_{0_{\text{単体試験プロセス}}}, C_{\text{単体試験プロセス}}, E_{\text{単体試験プロセス}} \rangle$ に対して、

$\Sigma_{\text{単体試験プロセス}} = \{$ 完成したオブジェクトコード (in), 完成した単体試験パッケージ (in), 単体試験プロセス開始 (int), 試験結果 (out), 成功 (in), 失敗 (in), 成功した試験の通知 (out), コードのフィードバック (out), 単体試験パッケージの

フィードバック (out), 単体試験プロセス終了 (out)}

$S_{\text{単体試験プロセス}} = \{TU0, TU1, TU2, TU3, TU4, TU5, TU6, TU7, TU8, TU9\}$

$S_{0_{\text{単体試験プロセス}}} = \{TU0\}$

$C_{\text{単体試験プロセス}} = \emptyset,$

$$E_{\text{単体試験プロセス}} = \left\{ \begin{array}{l} \langle TU0, TU1, \text{完成したオブジェクトコード } (in), \emptyset, \emptyset \rangle, \\ \langle TU0, TU2, \text{完成した単体試験パッケージ } (in), \emptyset, \emptyset \rangle, \\ \langle TU1, TU3, \text{完成した単体試験パッケージ } (in), \emptyset, \emptyset \rangle, \\ \langle TU2, TU3, \text{完成したオブジェクトコード } (in), \emptyset, \emptyset \rangle, \\ \langle TU3, TU4, \text{単体試験プロセス開始 } (int), \emptyset, \emptyset \rangle, \\ \langle TU4, TU5, \text{試験結果 } (out), \emptyset, \emptyset \rangle, \\ \langle TU5, TU6, \text{成功 } (in), \emptyset, \emptyset \rangle, \\ \langle TU5, TU7, \text{失敗 } (in), \emptyset, \emptyset \rangle, \\ \langle TU6, TU9, \text{成功した試験の通知 } (out), \emptyset, \emptyset \rangle, \\ \langle TU7, TU8, \text{コードのフィードバック } (out), \emptyset, \emptyset \rangle, \\ \langle TU8, TU9, \text{単体試験パッケージのフィードバック } (out), \emptyset, \emptyset \rangle, \\ \langle TU9, TU0, \text{単体試験プロセス終了 } (out), \emptyset, \emptyset \rangle \end{array} \right\}$$

とする。

上記の記述の図的表現を図 4.9 に示す。図中の記述で、イベントに付加された (in)、(out)、(int) は、それぞれ入力イベント、出力イベント、内部イベントを示している。

4.2.3 時間モジュールへの等価変換

時間 I/O オートマトンから時間モジュールへ等価変換後を以下に示す。このとき、等価変換に関しては 3 章に基づいている。

module 単体試験プロセス *(Test Unit)*

external 完成したオブジェクトコード, 完成した単体試験パッケージ, 成功, 失敗, 試験結果 1, 成功した試験の通知 1, コードのフィードバック 1, 単体試験パッケージのフィードバック 1, 単体試験プロセス終了 1;

interface 試験結果, 成功した試験の通知, コードのフィードバック, 単体試験パッケージのフィードバック, 単体試験プロセス終了, 完成したオブジェクト

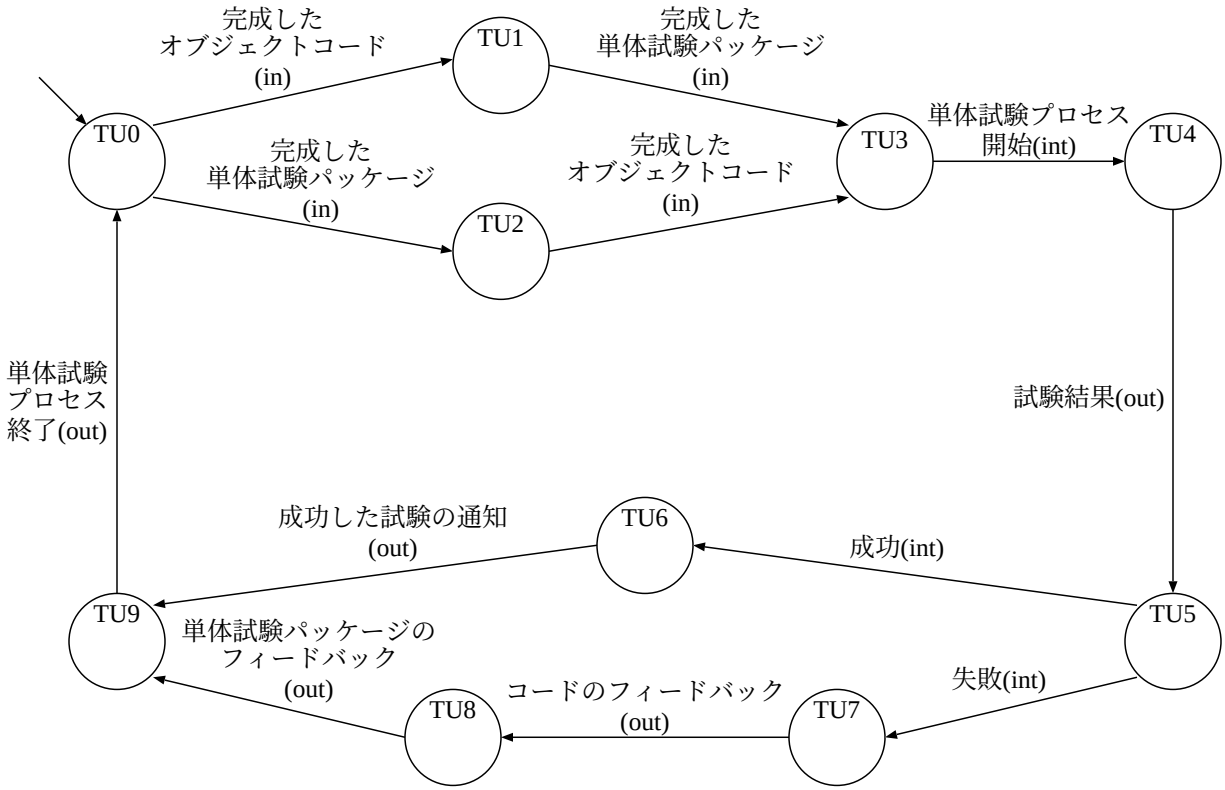


図 4.9: 時間 I/O オートマトンの図的表現 (単体試験プロセス)

コード 1, 完成した単体試験パッケージ 1, 成功 1, 失敗 1;
private state={TU0, TU1, TU2, TU3, TU4, TU5, TU6, TU7, TU8, TU9},
単体試験プロセス開始;

init

state:=TU0; 単体試験プロセス開始:=false; 試験結果:=false;
成功した試験の通知:=false; コードのフィードバック:=false;
単体試験パッケージのフィードバック:=false; 単体試験プロセス終了:=false; 完成したオブジェクトコード 1:=false;
完成した単体試験パッケージ 1:=false; 成功 1:=false; 失敗 1:=false;

update

| state=TU0 ∧ 単体試験プロセス終了 1=false ∧ 単体試験プロセス終了=true → 単体試験プロセス終了:=false
| state=TU0 → 完成したオブジェクトコード 1:=true

| state=TU0 → 完成した単体試験パッケージ 1:=true
| state=TU0 ∧ 単体試験プロセス終了=false ∧ 完成したオブジェクトコード=true → 完成したオブジェクトコード 1:=false; state:=TU1
| state=TU0 ∧ 単体試験プロセス終了=false ∧ 完成した単体試験パッケージ=true → 完成した単体試験パッケージ 1:=false; state:=TU2
| state=TU1 → 完成した単体試験パッケージ 1:=true
| state=TU1 ∧ 完成した単体試験パッケージ=true → 完成した単体試験パッケージ 1:=false; state:=TU3
| state=TU2 → 完成したオブジェクトコード 1:=true
| state=TU2 ∧ 完成したオブジェクトコード=true → 完成したオブジェクトコード 1:=false; state:=TU3
| state=TU3 → 単体試験プロセス開始:=true; state:=TU4
| state=TU4 ∧ 単体試験プロセス開始=true → 単体試験プロセス開始:=false
| state=TU4 ∧ 試験結果 1=true ∧ 単体試験プロセス開始=false → 試験結果:=true; state:=TU5
| state=TU5 ∧ 試験結果 1=false ∧ 試験結果=true → 試験結果:=false
| state=TU5 → 成功 1:=true; state:=TU6
| state=TU5 → 失敗 1:=true; state:=TU7
| state=TU5 ∧ 試験結果=false ∧ 成功=true → 成功 1:=false; state:=TU6
| state=TU5 ∧ 試験結果=false ∧ 失敗=true → 失敗 1:=false; state:=TU7
| state=TU6 ∧ 成功した試験の通知 1=true → 成功した試験の通知:=true; state:=TU9
| state=TU7 ∧ コードのフィードバック 1=true → コードのフィードバック:=true; state:=TU8
| state=TU8 ∧ コードのフィードバック=true ∧ コードのフィードバック 1=false → コードのフィードバック:=false

| state=TU8 ∧ コードのフィードバック=false ∧ 単体試験パッケージのフィードバック 1=true → 単体試験パッケージのフィードバック:=true; state:=TU9

| state=TU9 ∧ 成功した試験の通知=true ∧ 成功した試験の通知 1=false → 成功した試験の通知:=false

| state=TU9 ∧ 単体試験パッケージのフィードバック=true ∧ 単体試験パッケージのフィードバック 1=false → 単体試験パッケージのフィードバック:=false

| state=TU9 ∧ 成功した試験の通知=false ∧ 単体試験パッケージのフィードバック=false ∧ 単体試験プロセス終了 1=true → 単体試験プロセス終了:=true; state:=TU0

delay

| state=TU0 → true

| state=TU1 → true

| state=TU2 → true

| state=TU3 → true

| state=TU4 → true

| state=TU5 → true

| state=TU6 → true

| state=TU7 → true

| state=TU8 → true

| state=TU9 → true

図 4.10: 等価変換後の時間モジュール (単体試験プロセス)

等価変換後の時間モジュールの図的表現を図 4.11 に示す。

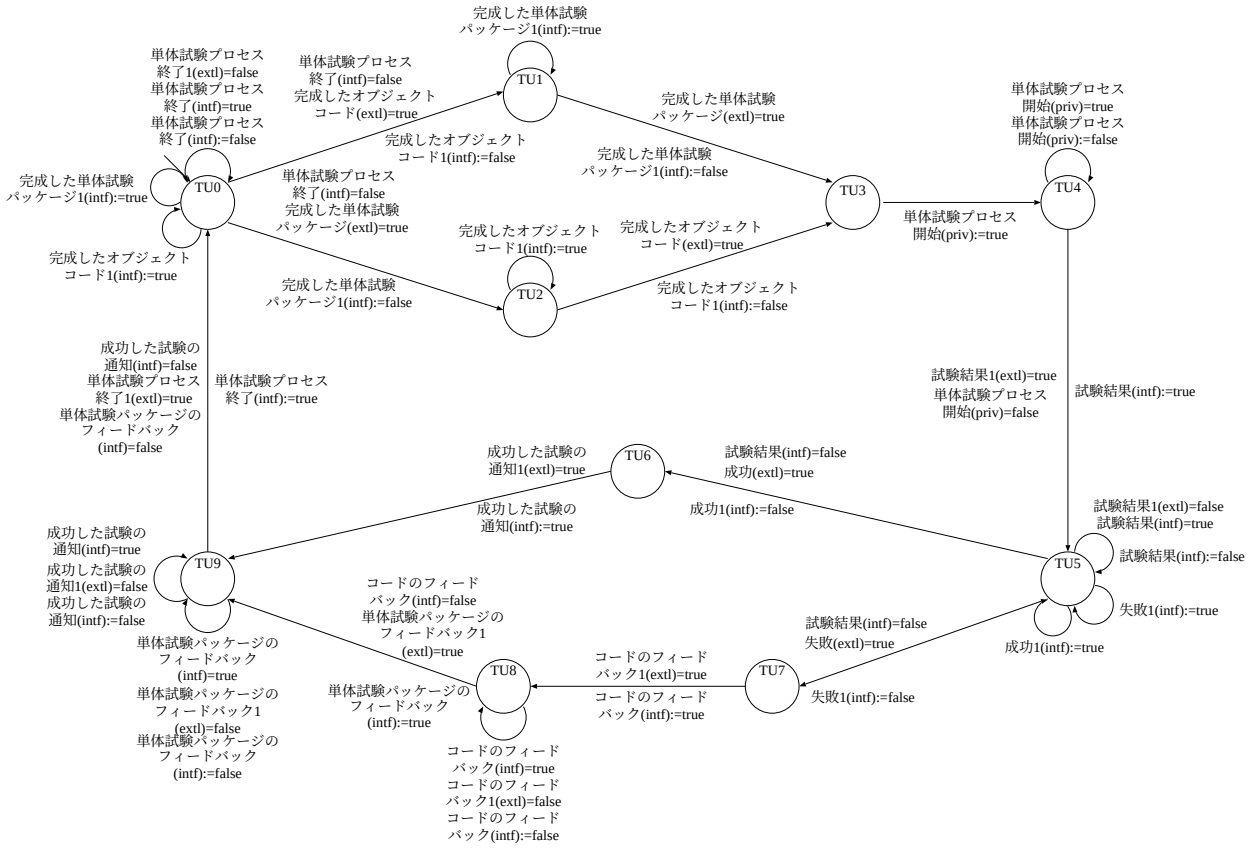


図 4.11: 等価変換後の時間モジュールの図的表現 (単体試験プロセス)

4.2.4 時間モジュールの最適化

以下に、等価変換して得られた時間モジュールの最適化を行う。ここで、『判定』は、単体試験の成功および失敗をブール変数の値で表現する入力変数である。

module 単体試験プロセス *(Test Unit)*

external 完成したオブジェクトコード, 完成した単体試験パッケージ, 成功, 失敗, 試験結果 1, 成功した試験の通知 1, コードのフィードバック 1, 単体試験パッケージのフィードバック 1, 単体試験プロセス終了 1;

interface 試験結果, 成功した試験の通知, コードのフィードバック, 単体試験パッケージのフィードバック, 単体試験プロセス終了, 完成したオブジェクトコード 1, 完成した単体試験パッケージ 1, 成功 1, 失敗 1;

private state={TU0, TU1, TU2, TU3}, 単体試験プロセス開始;

init

state:=TU0; 単体試験プロセス開始:=false; 試験結果:=false; 成功した試験の通知:=false; コードのフィードバック:=false; 単体試験パッケージのフィードバック:=false; 単体試験プロセス終了:=false; 完成したオブジェクトコード 1:=false; 完成した単体試験パッケージ 1:=false; 成功 1:=false; 失敗 1:=false;

update

| state=TU0 ∧ 単体試験プロセス終了 1=false ∧ 単体試験プロセス終了=true → 単体試験プロセス終了:=false

| state=TU0 → 完成したオブジェクトコード 1:=true

| state=TU0 → 完成した単体試験パッケージ 1:=true

| state=TU0 ∧ 単体試験プロセス終了=false ∧ 完成したオブジェクトコード=true ∧ 完成した単体試験パッケージ=true → 完成したオブジェクトコード 1:=false; 完成した単体試験パッケージ 1:=false; 単体試験プロセス開始:=true; state:=TU1

| state=TU1 ∧ 単体試験プロセス開始=true → 単体試験プロセス開始:=false

| state=TU1 ∧ 試験結果 1=true ∧ 単体試験プロセス開始=false → 試験結果:=true; state:=TU2

| state=TU2 ∧ 試験結果 1=false ∧ 試験結果=true → 試験結果:=false

| state=TU2 ∧ 試験結果=false ∧ 判定=成功 ∧ 成功した試験の通知 1=true → 成功した試験の通知:=true; state:=TU3

| state=TU2 ∧ 試験結果=false ∧ 判定=失敗 ∧ コードのフィードバック 1=true ∧ 単体試験パッケージのフィードバック 1=true → コードのフィードバック:=true; 単体試験パッケージのフィードバック:=true; state:=TU3

| state=TU3 ∧ コードのフィードバック=true ∧ コードのフィードバック 1=false → コードのフィードバック:=false

| state=TU3 ∧ 成功した試験の通知=true ∧ 成功した試験の通知 1=false → 成功した試験の通知:=false

| state=TU3 ∧ 単体試験パッケージのフィードバック=true ∧ 単体試験パッケージのフィードバック 1=false → 単体試験パッケージのフィードバック:=false

| state=TU3 ∧ 成功した試験の通知=false ∧ 単体試験パッケージのフィードバック=false ∧ コードのフィードバック=false ∧ 単体試験プロセス終了 1=true → 単体試験プロセス終了:=true; state:=TU0

delay

| state=TU0 → true

| state=TU1 → true

| state=TU2 → true

| state=TU3 → true

図 4.12: 最適化後の時間モジュール (単体試験プロセス)

最適化後の時間モジュールの図的表現を図 4.13 に示す。

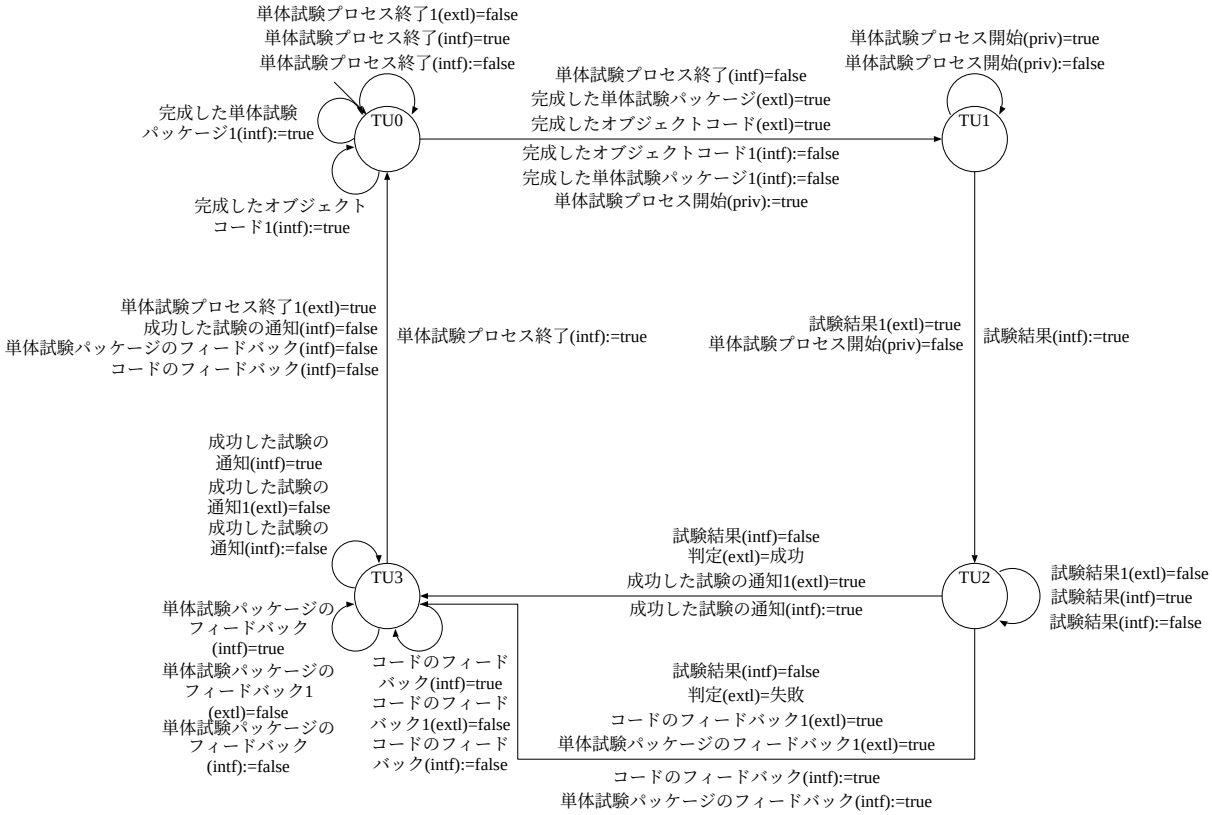


図 4.13: 最適化後の時間モジュールの図的表現 (単体試験プロセス)

第 5 章

考察

本章では、時間 I/O オートマトンから時間モジュールへの変換についての考察として、この変換の入力となる時間 I/O オートマトンと出力となる最適化後の時間モジュールの比較を行う。

まず、時間 I/O オートマトンによる記述においては、一つの枝に一つのイベントしかもつことができないため、**複数の入力**、**複数の出力**、**複数の入力順**、**入力と出力**、**判定結果**、**強制的な遷移**のそれぞれについて、以下のような工夫をして表現する必要がある。

複数の入力 入力を複数個同時に行うときには、以下のようにして記述する。複数の入力が他のただ一つの時間 I/O オートマトンからのものである場合、それらを総称するような一つのイベントを設け、そのイベントにより同期通信を行う。しかし、複数の入力が他の複数の時間 I/O オートマトンからのものである場合、複数の入力イベントを必要とし、それぞれ入力の数だけ入力イベントが必要となる。このため、入力の数だけ枝も必要となり、さらに、それにともない状態も必要となる。

複数の出力 これは、**複数の入力**と対になるものであり、記述に関して必要になるものも同じである。

複数の入力順 これは、**複数の入力**に関連する。他の時間 I/O オートマトンからのイベントの入力される順序が決まっていることは、あまりない。このとき、時間 I/O オートマトンにおいては、どのイベントがどの順序で入力されても良いように、その入力の数だけ場合分けを行う。このため、入力の数の階乗個の状態と枝が必要になる。

入力と出力 入力イベントが生じたら瞬時に、内部イベントが生じるような場合でも、時間 I/O オートマトンでは、2 つの枝で表現する必要がある。

判定結果 時間制約以外の条件を記述できないために、判定結果の数だけイベントを必要とする。また、そのイベントの数だけ状態と枝が必要となる。これは、時間制約による場合分けならば生じない。

強制的な遷移 他のシステムやプロセスからのイベントにより、その時点でどの状態であっても初期状態に遷移しなければならないとき、時間 I/O オートマトンにおいては、全ての状態からそのイベントによる枝が必要である。つまり、状態の数だけ枝の数が必要である。

これらのことは、等価変換後の時間モジュールでは、まだ時間 I/O オートマトンの記述を反映している。しかし、等価変換後の時間モジュールに以下のような、記述の最適化を行うことができる。

最適化 (複数の入力) 時間モジュールでは、時間 I/O オートマトンと異なり、イベントでなく、ブール変数の値の更新により遷移が生じる。さらに、ブール変数の値の変化による条件をもつことができる。条件を複数個もつことができるため、複数の更新式を一つの更新式にすることができる。このとき、複数の更新式を一つの更新式にするには、3.2.1 のいくつかの条件を満たさなければならない。

最適化 (複数の出力) 複数の更新を一つの更新式にすることが可能であるため、複数の出力に関する更新式を一つの更新式にすることができる。このとき、複数の更新式を一つの更新式にするには、3.2.1 のいくつかの条件を満たさなければならない。

最適化 (複数の入力順) これは、**複数の入力**を解決することにより、入力の順序に関しても解決する。入力の順序がわからないために生じた場合分けは、**複数の入力**により、それらの更新式が全て同じ式になる。したがって、これらの同じ更新式は複数個必要ないため、一つの更新式にする。

最適化 (入力と出力) 時間モジュールでは、時間 I/O オートマトンの入力イベントは外部変数として条件になる。このため、外部変数の更新式による遷移後、瞬時に遷移をする場合、そのあとの更新式を外部変数のみの更新式と一つにすることができる。

最適化 (判定結果) 時間モジュールで記述可能な、変数 state のような変数を用いる。これにより、どの変数が true かによって場合分けをするのではなく、変数の値が何であるかによって場合分けする。すると、変数、枝、状態の数が減少する。

強制的な遷移 他のシステムやプロセスからの変数値が true になることにより、その時点で変数 state の値が何であっても初期アクション後の変数 state の値に遷移させる。そのためには、その変数値の条件と初期アクションの更新から構成する、新たな更新式を一つ追加するだけでよい。したがって、どの状況であっても対応しなければならない命令などに対して、時間モジュールは適しているといえる。

第 4 章における適用例からも、時間 I/O オートマトンの状態の数より、時間モジュールの変数 $state$ の数の方が少なく記述できることが確認できた。さらに、時間モジュールでは、等価変換で更新式の変更が付加されるが、その後の最適化によって時間 I/O オートマトンの枝の数より時間モジュールの更新式の数の方が少なく記述できることも確認できた。

ところで、時間モジュールは、更新式で記述しているため、時間 I/O オートマトンと比較すると可視性に乏しい。そこで、本論文では、時間モジュールの図的表現を提案した。これにより、時間モジュールの直観的理解を支援している。

第 6 章

おわりに

6.1 まとめ

本論文では、時間I/O オートマトンから時間モジュールへの変換について考察した。この変換は以下の 2 つの部分からなる。

- 時間I/O オートマトンから時間モジュールへの等価変換
- 等価変換によって得られた時間モジュールの最適化

これらの 2 つの手法をそれぞれ提案し、さらに、その正当性の証明を行った。

この変換を、本論文では、以下の 2 つの例について適用した。

- 実時間システムの例である、列車制御システムの仕様、
- 実時間システム以外の例である、ソフトウェアプロセス記述

考察の結果、時間モジュールの方が、等価変換後の時間モジュールより変数 state の数、更新式の数が少なく記述できることがわかった。

6.2 今後の課題

ソフトウェアプロセス記述に関する最適化として、次のことを検討する予定である。本論文では、ソフトウェアプロセスへの適用において、各部分プロセス毎に記述を行った。しかし、実際には、一人の開発者がこれらの部分プロセスを複数兼任していることがある。このため、部分プロセス間の通信で不要になる箇所も生じてくる。部分プロセスの合

成方法は、その開発者がどの部分プロセスとどの部分プロセスの作業を兼任しているかによって異なってくる。このように、どのように兼任していても部分プロセスの合成が行える最適化法の開発を一つ目の課題とする。

本論文では、列車制御システムの仕様と Kellner らの例題仕様のソフトウェアプロセスの記述に変換を適用した。他の実時間システムの仕様や、他のソフトウェアプロセスを、時間 I/O オートマトンで記述し、本論文の等価変換と、最適化が適用可能であるかどうかは、まだ確認していない。二つ目の課題として、他の実時間システムの仕様や、他のソフトウェアプロセスへの適用を挙げる。

また、上記の適用が可能でない場合、どのような最適化が可能となるのか、また、その最適化は、今まで適用可能であったものに対しても可能かどうかの確認も必要である。ここで、三つ目の課題として、本論文の最適化が適用不可能な例を見つけ出し、それに対する最適化法の開発、本論文の適用例への新しい最適化法の適用の確認、を挙げる。

謝辞

なによりもまず、本研究を進めるにあたり、終始懇切なる御指導を賜わりました、落水浩一郎教授に深く感謝いたします。

本論文をまとめるにあたって適切な御指導を頂きました、片山卓也教授、篠田陽一助教授に深く感謝いたします。

本研究を進めるにあたり適切な御意見を頂きました、日比野靖教授、二木厚吉教授、ならびに、現在、東京工業大学大学院情報理工学研究科の渡部卓雄助教授に深く感謝いたします。

本研究の開始時から、細部にわたる議論にまで応じて頂き、本論文をまとめるにあたり御指導を頂きました服部哲助手に深く感謝いたします。また、本研究に関して多くの有意義な助言を頂きました、村越広享助手、藤枝和宏助手に深く感謝いたします。

本研究について適切な御意見を頂きました、電子情報通信学会ソフトウェアサイエンス研究会の皆様に深く感謝いたします。

最後に、本研究に関して有意義な助言を頂きました落水研究室、篠田研究室の皆様に心より感謝いたします。

参考文献

[1] R. Alur and D. L. Dill. A theory of timed automata. Theoretical Computer Science, Vol.126, pp. 183-235, 1994.

[2] R. Alur and T. A. Henzinger. Modularity for timed and hybrid systems. CONCUR97, LNCS Vol.1243, pp. 74-88, 1998.

[3] M. I. Kellner, P. H. Feiler, A. Finkelstein, T. Katayama, L. J. Osterweil, M. H. Penedo, H. D. Rombach. Software Process Modeling Example Problem. 6th ISPW, pp. 19-29, 1990.

[4] R. Alur and T. A. Henzinger. Reactive Modules. 11th Annual IEEE Symposium on Logic in Computer Science, pp.207-218, 1996.

[5] Jin Sa, Brian Warboys. Modelling Processes Using a Stepwise Refinement. EWSPT 1994, pp.40-58, 1994

[6] F. Shams Aliee and B.C. Warboys. Roles Represent Patterns. Proceedings of the Workshop on Pattern Languages of Object-Oriented Programs at ECOOP'95, 1995.

[7] Ian Robertson. An Implementation of the ISPW-6 Process Example. EWSPT 1994, pp.187-206, 1994

[8] Maarten Steenhuis. Reflections on functional modelling in SOCCA. Master's thesis, Department of Computer Science, Leiden University, August 1995

[9] R.F. Bruynooghe, R.M. Greenwood, I. Robertson, J. Sa, R.A. Snowdon and B.C. Warboys. PADM: Towards a Total Process Modelling System. Software Process Modelling and Technology. Research Studies Press, pp. 293-334, 1994.

[10] J. Sa and B.C. Warboys. A Formal Description of the ISPW-6 Software Process Example. Technical Report UMCS-93-6-1, Department of Computer Science, University of Manchester, 1993.

[11] R. Alur and D. L. Dill. Automata For Modeling Real-Time Systems. LNCS 443, pp.322-335, 1990

[12] R. Gawlick, R. Segala, J. Sogaard-Andersen, N. Lynch. Liveness in Timed and Untimed Systems. ICALP'94, LNCS Vol. 820, pp.166-177, 1994.

[13] 館宣伸, 服部哲, 落水浩一郎. 時間オートマトンから時間モジュールへの変換について. 電気関係学会北陸支部連合大会, pp.266, 2000.

[14] 落水浩一郎. ソフトウェアプロセスに関する研究の概要. 情報処理 Vol.36 No.5, pp.379-391, 1995.

[15] 館宣伸, 服部哲, 落水浩一郎. ソフトウェアプロセスの時間オートマトンおよび時間モジュールによる記述法の比較. 信学技報 SS2000-46, pp.33-40, 2001.

付 録 A

ソフトウェアプロセスの例

ソフトウェアプロセスについて、まず、構成管理委員会、ソフトウェアプロセスの部分プロセスについて、タスクのスケジューリングおよび割当プロセス、デザイン変更プロセス、デザインレビュープロセス、コード変更プロセス、試験計画変更プロセス、単体試験パッケージ変更プロセス、単体試験プロセス、進捗状況管理プロセス、そして、ソフトウェアプロセスにおけるファイルについて、ファイル (作業計画)、ソフトウェア設計書ファイル (デザイン)、ソフトウェア開発ファイル (ソースコード)、ソフトウェア開発ファイル (オブジェクトコード)、試験計画ファイル (試験計画)、試験パッケージファイルファイル (単体試験パッケージ)、試験履歴ファイル (試験結果) の順にそれぞれ、時間 I/O オートマトン記述し、図的表現を以下に示す。

構成管理委員会 $= \langle \Sigma_{\text{構成管理委員会}}, S_{\text{構成管理委員会}}, S_{0_{\text{構成管理委員会}}}, C_{\text{構成管理委員会}}, E_{\text{構成管理委員会}} \rangle$ に対して、

$$\Sigma_{\text{構成管理委員会}} = \{ \text{変更許可}\,(out), \text{要求変更}\,(out), \text{中止決定}\,(out), \text{中止勧告}\,(in) \}$$
$$S_{\text{構成管理委員会}} = \{CCB0, CCB1\}$$
$$S_{0_{\text{構成管理委員会}}} = \{CCB0\}$$
$$C_{\text{構成管理委員会}} = \emptyset,$$
$$E_{\text{構成管理委員会}} = \left\{ \begin{array}{l} \langle CCB0, CCB0, \text{中止決定}\,(out), \emptyset, \emptyset \rangle, \\ \langle CCB0, CCB0, \text{中止勧告}\,(in), \emptyset, \emptyset \rangle, \\ \langle CCB0, CCB1, \text{要求変更}\,(out), \emptyset, \emptyset \rangle, \\ \langle CCB1, CCB0, \text{作業計画}\,(out), \emptyset, \emptyset \rangle \end{array} \right\}$$

タスクのスケジューリングおよび割当プロセス $= \langle \Sigma_{\text{タスクのスケジューリングおよび割当プロセス}},$

図 A.1: 時間 I/O オートマトンによる記述 (構成管理委員会)

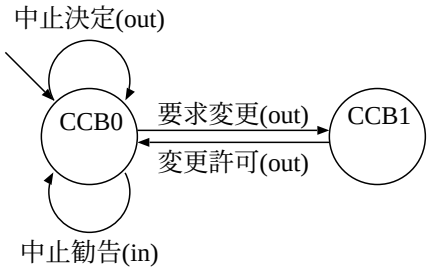


図 A.2: 時間 I/O オートマトンの図的表現 (構成管理委員会)

$S_{\text{タスクのスケジューリングおよび割当プロセス}}$, $S_{0_{\text{タスクのスケジューリングおよび割当プロセス}}}$, $C_{\text{タスクのスケジューリングおよび割当プロセス}}$, $E_{\text{タスクのスケジューリングおよび割当プロセス}}\rangle$ に対して、

$\Sigma_{\text{タスクのスケジューリングおよび割当プロセス}} = \{$ 変更許可 (in), タスクのスケジューリングおよび割当プロセス開始 (int), 要求変更 (in), 作業計画 (in), 修正後の作業計画 (out), タスク割当とスケジュール日程の通知 (out), 要求変更の通知 (out), タスクのスケジューリングおよび割当プロセス終了 $(out)\}$

$S_{\text{タスクのスケジューリングおよび割当プロセス}} = \{SAT0, SAT1, SAT2, SAT3, SAT4, SAT5, SAT6, SAT7\}$

$S_{0_{\text{タスクのスケジューリングおよび割当プロセス}}} = \{SAT0\}$

$C_{\text{タスクのスケジューリングおよび割当プロセス}} = \emptyset,$

$E_{\text{タスクのスケジューリングおよび割当プロセス}} =$

$$\left\{\begin{array}{l}
\langle SAT0, SAT1, \text{変更許可}\ (in), \emptyset, \emptyset\rangle, \\
\langle SAT1, SAT2, \text{タスクのスケジューリングおよび割当プロセス開始}\ (int), \emptyset, \emptyset\rangle, \\
\langle SAT2, SAT3, \text{要求変更}\ (in), \emptyset, \emptyset\rangle, \\
\langle SAT3, SAT4, \text{作業計画}\ (in), \emptyset, \emptyset\rangle, \\
\langle SAT4, SAT5, \text{修正後の作業計画}\ (out), \emptyset, \emptyset\rangle, \\
\langle SAT5, SAT6, \text{タスク割当とスケジュール日程の通知}\ (out), \emptyset, \emptyset\rangle, \\
\langle SAT6, SAT7, \text{要求変更の通知}\ (out), \emptyset, \emptyset\rangle, \\
\langle SAT7, SAT0, \text{タスクのスケジューリングおよび割当プロセス終了}\ (out), \emptyset, \emptyset\rangle
\end{array}\right\}$$

図 A.3: 時間 I/O オートマトンによる記述 (タスクのスケジューリングおよび割当プロセス)

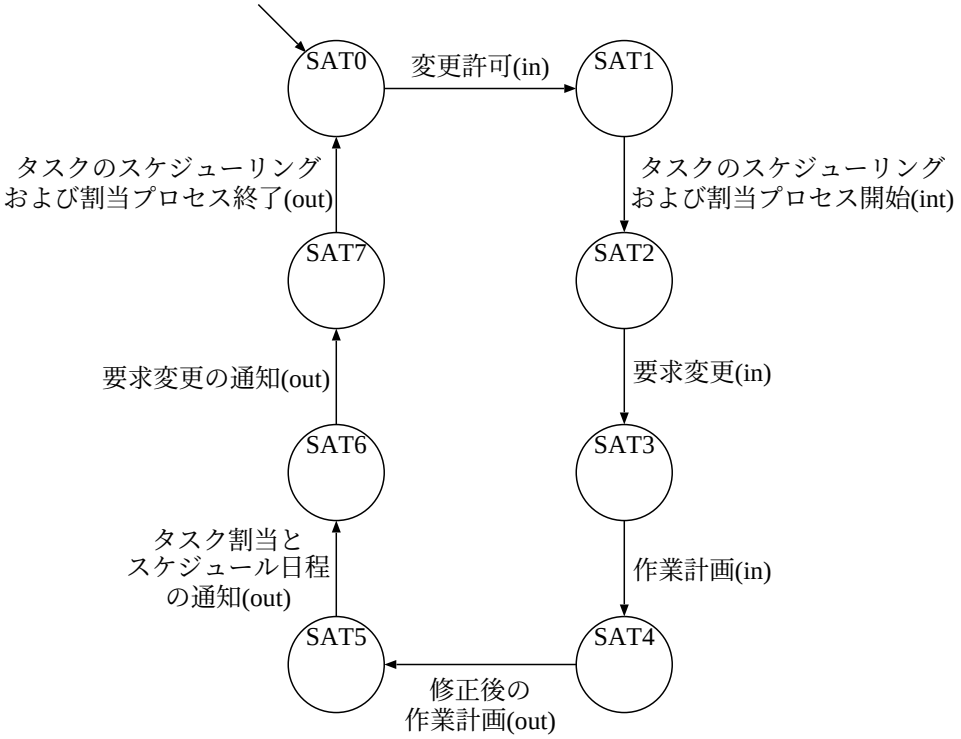


図 A.4: 時間 I/O オートマトンの図的表現 (タスクのスケジューリングおよび割当プロセス)

デザイン変更プロセス $= \langle \Sigma_{\text{デザイン変更プロセス}}, S_{\text{デザイン変更プロセス}}, S_{0_{\text{デザイン変更プロセス}}}, C_{\text{デザイン変更プロセス}}, E_{\text{デザイン変更プロセス}} \rangle$ に対して、

$\Sigma_{\text{デザイン変更プロセス}} = \{$ タスク割当とスケジュール日程の通知 (in), デザインのフィードバック (in), デザイン変更プロセス開始 (int), デザイン (in), 修正後のデザイン (out), デザイン変更プロセス終了 $(out)\}$

$S_{\text{デザイン変更プロセス}} = \{MD0, MD1, MD2, MD3, MD4\}$

$S_{0_{\text{デザイン変更プロセス}}} = \{MD0\}$

$C_{\text{デザイン変更プロセス}} = \emptyset,$

$$E_{\text{デザイン変更プロセス}} = \left\{ \begin{array}{l} \langle MD0, MD1, \text{タスク割当とスケジュール日程の通知 } (in), \emptyset, \emptyset \rangle, \\ \langle MD0, MD1, \text{デザインのフィードバック } (in), \emptyset, \emptyset \rangle, \\ \langle MD1, MD2, \text{デザイン変更プロセス開始 } (int), \emptyset, \emptyset \rangle, \\ \langle MD2, MD3, \text{デザイン } (in), \emptyset, \emptyset, \rangle, \\ \langle MD3, MD4, \text{修正後のデザイン } (out), \emptyset, \emptyset \rangle, \\ \langle MD4, MD0, \text{デザイン変更プロセス終了 } (out), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.5: 時間 I/O オートマトンによる記述 (デザイン変更プロセス)

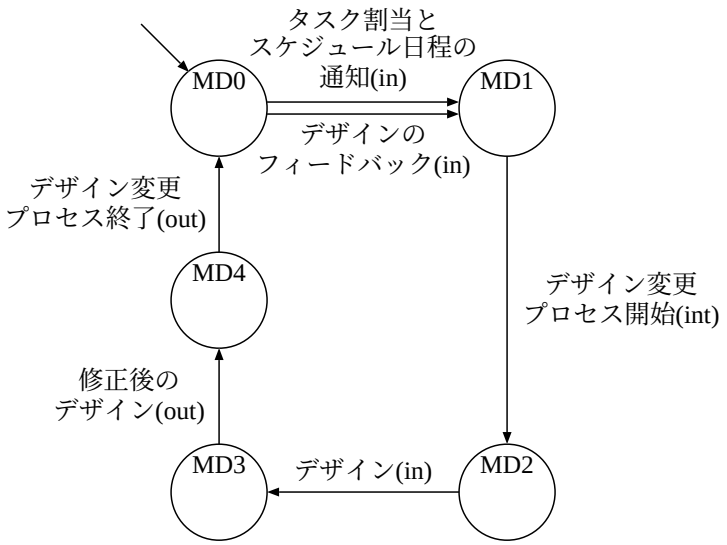


図 A.6: 時間 I/O オートマトンの図的表現 (デザイン変更プロセス)

デザインレビュープロセス $= \langle \Sigma_{\text{デザインレビュープロセス}}, S_{\text{デザインレビュープロセス}}, S_{0_{\text{デザインレビュープロセス}}}, C_{\text{デザインレビュープロセス}}, E_{\text{デザインレビュープロセス}} \rangle$ に対して、

$\Sigma_{\text{デザインレビュープロセス}} =$ { タスク割当とスケジュール日程の通知 (in), 修正後のデザイン (in), デザインレビュープロセス開始 (int), デザイン認可 (in), 認可されたデザイン (out), デザイン不可 (in), デザインのフィードバック (out), レビュー結果の通知と発見された誤りの数と作業報告 (out), デザインレビュープロセス終了 (out)}

$S_{\text{デザインレビュープロセス}} = \{RD0, RD1, RD2, RD3, RD4, RD5, RD6, RD7\}$

$S_{0_{\text{デザインレビュープロセス}}} = \{RD0\}$

$C_{\text{デザインレビュープロセス}} = \emptyset,$

$E_{\text{デザインレビュープロセス}} =$

$$
\left\{
\begin{array}{l}
\langle RD0, RD1, \text{タスク割当とスケジュール日程の通知}\ (in), \emptyset, \emptyset\rangle, \\
\langle RD1, RD2, \text{修正後のデザイン}\ (in), \emptyset, \emptyset\rangle, \\
\langle RD2, RD3, \text{デザインレビュープロセス開始}\ (int), \emptyset, \emptyset\rangle, \\
\langle RD3, RD4, \text{デザイン認可}\ (in), \emptyset, \emptyset\rangle, \\
\langle RD4, RD6, \text{認可されたデザイン}\ (out), \emptyset, \emptyset\rangle, \\
\langle RD3, RD5, \text{デザイン不可}\ (in), \emptyset, \emptyset\rangle, \\
\langle RD5, RD6, \text{デザインのフィードバック}\ (out), \emptyset, \emptyset\rangle, \\
\langle RD6, RD7, \text{レビュー結果の通知と発見された誤りの数と作業報告}\ (out), \emptyset, \emptyset\rangle, \\
\langle RD7, RD0, \text{デザインレビュープロセス終了}\ (out), \emptyset, \emptyset\rangle
\end{array}
\right\}
$$

図 A.7: 時間 I/O オートマトンによる記述 (デザインレビュープロセス)

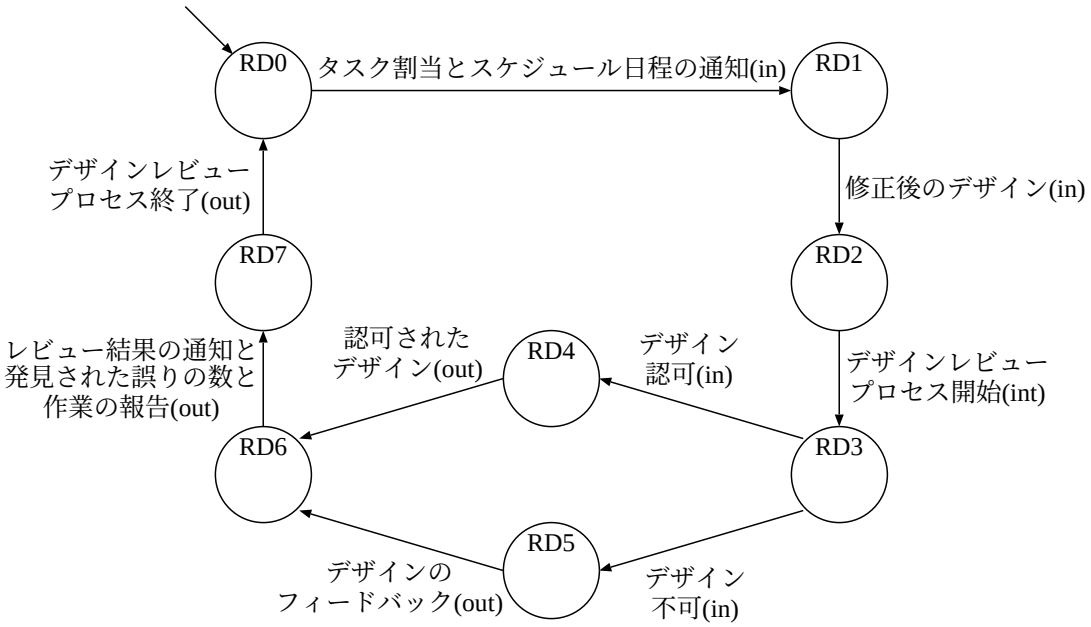


図 A.8: 時間 I/O オートマトンの図的表現 (デザインレビュー)

コード変更プロセス $= \langle \Sigma_{\text{コード変更プロセス}}, S_{\text{コード変更プロセス}}, S_{0_{\text{コード変更プロセス}}}, C_{\text{コード変更プロセス}}, E_{\text{コード変更プロセス}} \rangle$ に対して、

$\Sigma_{\text{コード変更プロセス}} = \{$ タスク割当とスケジュール日程の通知 (in), コードのフィードバック (in), コード変更プロセス開始 (int), デザイン認可 (in), 修正後のデザ

イン (in), ソースコード (in), コンパイル (int), ソースコードの完成 (out), オブジェクトコード完成 (out), コード変更プロセス終了 (out)}

$S_{\text{コード変更プロセス}} = \{MC0, MC1, MC2, MC3, MC4, MC5, MC6, MC7, MC8, MC9, MC10\}$

$S_{0_{\text{コード変更プロセス}}} = \{MC0\}$

$C_{\text{コード変更プロセス}} = \{mc\}$,

$$E_{\text{コード変更プロセス}} = \left\{ \begin{array}{l} \langle MC0, MC1, \text{タスク割当とスケジュール日程の通知}\ (in), mc := 0, \emptyset \rangle, \\ \langle MC0, MC1, \text{コードのフィードバック}\ (in), mc := 0, \emptyset \rangle, \\ \langle MC1, MC2, \text{コード変更プロセス開始}\ (int), \emptyset, mc < 5 \rangle, \\ \langle MC2, MC3, \text{デザイン認可}\ (in), \emptyset, mc < 5 \rangle, \\ \langle MC3, MC4, \text{修正後のデザイン}\ (in), \emptyset, mc < 5 \rangle, \\ \langle MC4, MC5, \text{ソースコード}\ (in), \emptyset, mc < 5 \rangle, \\ \langle MC5, MC5, \text{コンパイル}\ (int), \emptyset, mc < 5 \rangle, \\ \langle MC5, MC9, \text{ソースコード完成}\ (out), \emptyset, mc < 5 \rangle, \\ \langle MC2, MC6, \text{修正後のデザイン}\ (in), \emptyset, mc < 5 \rangle, \\ \langle MC6, MC7, \text{ソースコード}\ (in), \emptyset, mc < 5 \rangle, \\ \langle MC7, MC7, \text{コンパイル}\ (in), \emptyset, mc < 5 \rangle, \\ \langle MC7, MC8, \text{デザイン認可}\ (in), \emptyset, \emptyset \rangle, \\ \langle MC8, MC9, \text{ソースコード完成}\ (out), \emptyset, mc < 5 \rangle, \\ \langle MC9, MC10, \text{オブジェクトコード完成}\ (out), \emptyset, mc < 5 \rangle, \\ \langle MC10, MC11, \text{コード変更プロセス終了}\ (out), \emptyset, mc < 5 \rangle \end{array} \right\}$$

図 A.9: 時間 I/O オートマトンによる記述 (コード変更プロセス)

試験計画変更プロセス $= \langle \Sigma_{\text{試験計画変更プロセス}}, S_{\text{試験計画変更プロセス}}, S_{0_{\text{試験計画変更プロセス}}}, C_{\text{試験計画変更プロセス}}, E_{\text{試験計画変更プロセス}} \rangle$ に対して、

$\Sigma_{\text{試験計画変更プロセス}} =$ { タスク割当とスケジュール日程の通知 (in), 試験計画 (in), 試験計画変更プロセス開始 (int), 試験計画完成 (out), 試験計画変更プロセス終了 (out)}

$S_{\text{試験計画変更プロセス}} = \{MTP0, MTP1, MTP2, MTP3, MTP4\}$

$S_{0_{\text{試験計画変更プロセス}}} = \{MTP0\}$

$C_{\text{試験計画変更プロセス}} = \emptyset$,

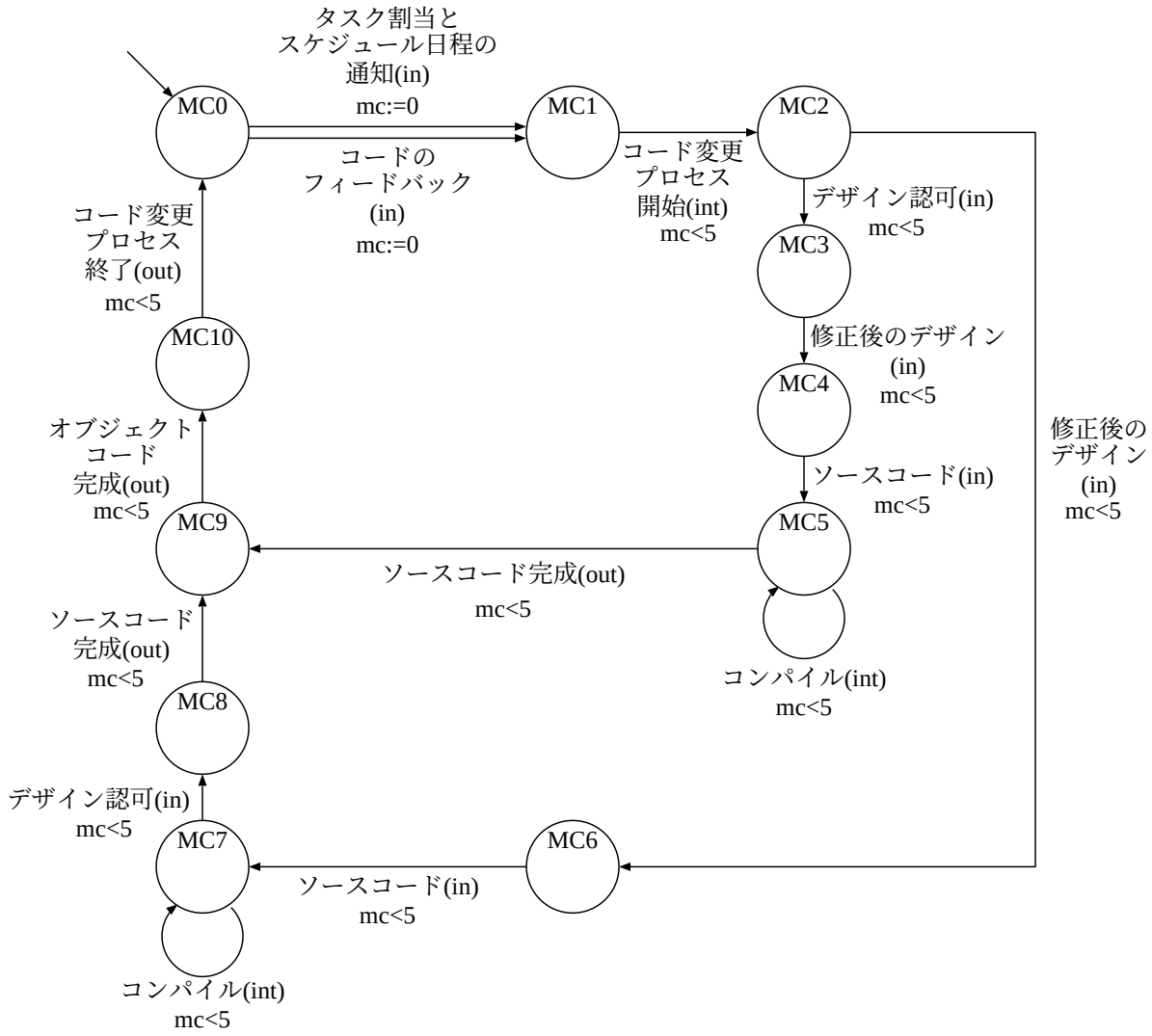


図 A.10: 時間 I/O オートマトンの図的表現 (コード変更プロセス)

$$E_{\text{試験計画変更プロセス}} = \left\{ \begin{array}{l} \langle MTP0, MTP1, \text{タスク割当とスケジュール日程の通知}\ (in), \emptyset, \emptyset \rangle, \\ \langle MTP1, MTP2, \text{試験計画}\ (in), \emptyset, \emptyset \rangle, \\ \langle MTP2, MTP3, \text{試験計画変更プロセス開始}\ (int), \emptyset, \emptyset \rangle, \\ \langle MTP3, MTP4, \text{試験計画完成}\ (out), \emptyset, \emptyset \rangle, \\ \langle MTP3, MTP4, \text{修正後のデザイン}\ (in), \emptyset, \emptyset \rangle, \\ \langle MTP4, MTP0, \text{試験計画変更プロセス終了}\ (out), \emptyset, \emptyset \rangle \end{array} \right\}$$

単体試験パッケージ変更プロセス $= \langle \Sigma_{\text{単体試験パッケージ変更プロセス}}, S_{\text{単体試験パッケージ変更プロセス}}, S_{0_{\text{単体試験パッケージ変更プロセス}}}, C_{\text{単体試験パッケージ変更プロセス}}, E_{\text{単体試験パッケージ変更プロセス}} \rangle$ に対して、

図 A.11: 時間 I/O オートマトンによる記述 (試験計画変更プロセス)

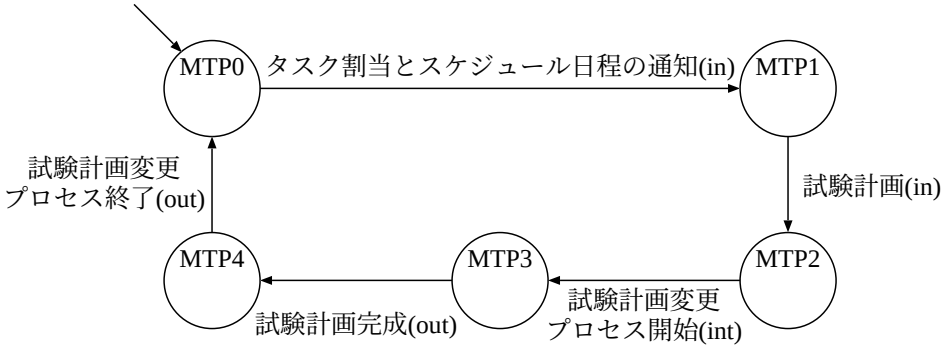


図 A.12: 時間 I/O オートマトンの図的表現 (試験計画変更プロセス)

$\Sigma_{\text{単体試験パッケージ変更プロセス}} = \{$ 試験計画変更プロセス終了 (in), 単体試験パッケージのフィードバック (in), 単体試験パッケージ変更プロセス開始 (int), 試験計画 (in), 完成したソースコード (in), 単体試験パッケージ (in), 修正後のデザイン (in), 単体試験パッケージ完成 (out), 単体試験パッケージ変更プロセス終了 $(out)\}$

$S_{\text{単体試験パッケージ変更プロセス}} = \{MUTP0, MUTP1, MUTP2, MUTP3, MUTP4, MUTP5, MUTP6, M$

$S_{0_{\text{単体試験パッケージ変更プロセス}}} = \{MUTP0\}$

$C_{\text{単体試験パッケージ変更プロセス}} = \emptyset,$

$E_{\text{単体試験パッケージ変更プロセス}} =$

$$\left\{\begin{array}{l}
\langle MUTP0, MUTP1, \text{試験計画変更プロセス終了}\ (in), \emptyset, \emptyset\rangle, \\
\langle MUTP0, MUTP1, \text{単体試験パッケージのフィードバック}\ (in), \emptyset, \emptyset\rangle, \\
\langle MUTP1, MUTP2, \text{単体試験パッケージ変更プロセス開始}\ (int), \emptyset, \emptyset\rangle, \\
\langle MUTP2, MUTP3, \text{試験計画}\ (in), \emptyset, \emptyset\rangle, \\
\langle MUTP3, MUTP4, \text{完成したソースコード}\ (in), \emptyset, \emptyset\rangle, \\
\langle MUTP4, MUTP5, \text{単体試験パッケージ}\ (in), \emptyset, \emptyset\rangle, \\
\langle MUTP5, MUTP6, \text{修正後のデザイン}\ (in), \emptyset, \emptyset\rangle, \\
\langle MUTP6, MUTP7, \text{単体試験パッケージ完成}\ (out), \emptyset, \emptyset\rangle, \\
\langle MUTP7, MUTP0, \text{単体試験パッケージ変更プロセス終了}\ (out), \emptyset, \emptyset\rangle
\end{array}\right\}$$

図 A.13: 時間 I/O オートマトンによる記述 (単体試験パッケージ変更プロセス)

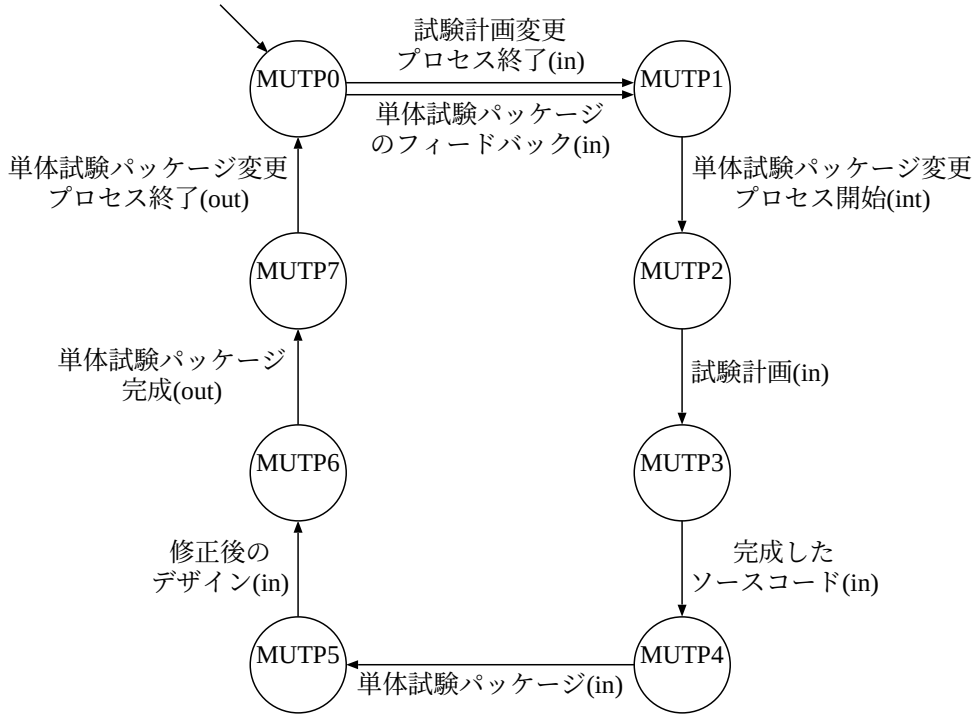


図 A.14: 時間 I/O オートマトンの図的表現 (単体試験パッケージ変更プロセス)

単体試験プロセス $= \langle \Sigma_{\text{単体試験プロセス}}, S_{\text{単体試験プロセス}}, S_{0_{\text{単体試験プロセス}}}, C_{\text{単体試験プロセス}}, E_{\text{単体試験プロセス}} \rangle$ に対して、

$\Sigma_{\text{単体試験プロセス}} = \{$ 完成したオブジェクトコード (in), 完成した単体試験パッケージ (in), 単体試験プロセス開始 (int), 試験結果 (out), 成功 (in), 失敗 (in), 成功した試験の通知 (out), コードのフィードバック (out), 単体試験パッケージのフィードバック (out), 単体試験プロセス終了 $(out)\}$

$S_{\text{単体試験プロセス}} = \{TU0, TU1, TU2, TU3, TU4, TU5, TU6, TU7, TU8, TU9\}$

$S_{0_{\text{単体試験プロセス}}} = \{TU0\}$

$C_{\text{単体試験プロセス}} = \emptyset,$

$$E_{\text{単体試験プロセス}} = \left\{ \begin{array}{l} \langle TU0, TU1, \text{完成したオブジェクトコード}\ (in), \emptyset, \emptyset \rangle, \\ \langle TU0, TU2, \text{完成した単体試験パッケージ}\ (in), \emptyset, \emptyset \rangle, \\ \langle TU1, TU3, \text{完成した単体試験パッケージ}\ (in), \emptyset, \emptyset \rangle, \\ \langle TU2, TU3, \text{完成したオブジェクトコード}\ (in), \emptyset, \emptyset \rangle, \\ \langle TU3, TU4, \text{単体試験プロセス開始}\ (int), \emptyset, \emptyset \rangle, \\ \langle TU4, TU5, \text{試験結果}\ (out), \emptyset, \emptyset \rangle, \\ \langle TU5, TU6, \text{成功}\ (in), \emptyset, \emptyset \rangle, \\ \langle TU5, TU7, \text{失敗}\ (in), \emptyset, \emptyset \rangle, \\ \langle TU6, TU9, \text{成功した試験の通知}\ (out), \emptyset, \emptyset \rangle, \\ \langle TU7, TU8, \text{コードのフィードバック}\ (out), \emptyset, \emptyset \rangle, \\ \langle TU8, TU9, \text{単体試験パッケージのフィードバック}\ (out), \emptyset, \emptyset \rangle, \\ \langle TU9, TU0, \text{単体試験プロセス終了}\ (out), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.15: 時間 I/O オートマトンによる記述 (単体試験プロセス)

進捗状況管理プロセス $= \langle \Sigma_{\text{進捗状況管理プロセス}}, S_{\text{進捗状況管理プロセス}}, S_{0_{\text{進捗状況管理プロセス}}}, C_{\text{進捗状況管理プロセス}}, E_{\text{進捗状況管理プロセス}} \rangle$ に対して、

$\Sigma_{\text{進捗状況管理プロセス}} = \{$ タスクのスケジューリングおよび割当プロセス終了 (in), 進捗状況管理プロセス開始 (int), レビュー結果の通知と発見された誤りの数と作業報告 (in), 現在の作業計画 (in), 修正後の作業計画 (out), タスク割当とスケジュール日程の通知 (out), 中止決定 (in), 中止勧告 (out), 成功した試験の通知 (in), 進捗状況管理プロセス終了 $(out)\}$

$S_{\text{進捗状況管理プロセス}} = \{MP0, MP1, MP2, MP3, MP4, MP5, MP6, MP7\}$

$S_{0_{\text{進捗状況管理プロセス}}} = \{MP0\}$

$C_{\text{進捗状況管理プロセス}} = \emptyset,$

$E_{\text{進捗状況管理プロセス}} =$

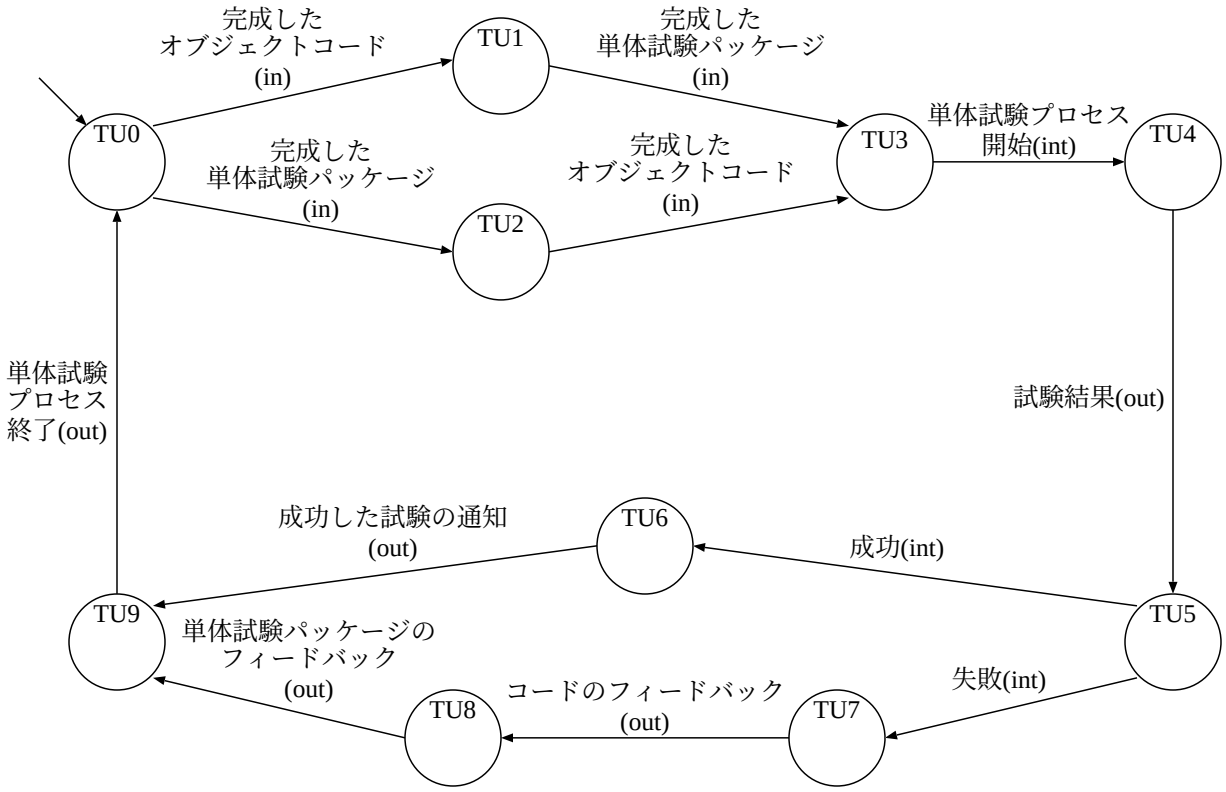


図 A.16: 時間 I/O オートマトンの図的表現 (単体試験プロセス)

$$
\left\{
\begin{array}{l}
\langle MP0, MP1, \text{タスクのスケジューリングおよび割当プロセス終了}\ (in), \emptyset, \emptyset\rangle, \\
\langle MP1, MP2, \text{進捗状況管理プロセス開始}\ (int), \emptyset, \emptyset\rangle, \\
\langle MP2, MP3, \text{レビュー結果の通知と発見された誤りの数と作業報告}\ (in), \emptyset, \emptyset\rangle, \\
\langle MP3, MP4, \text{現在の作業計画}\ (in), \emptyset, \emptyset\rangle, \\
\langle MP4, MP5, \text{修正後の作業計画}\ (out), \emptyset, \emptyset\rangle, \\
\langle MP5, MP7, \text{タスク割当とスケジュール日程の通知}\ (out), \emptyset, \emptyset\rangle, \\
\langle MP2, MP6, \text{中止決定}\ (in), \emptyset, \emptyset\rangle, \\
\langle MP6, MP7, \text{中止勧告}\ (out), \emptyset, \emptyset\rangle, \\
\langle MP2, MP7, \text{成功した試験の通知}\ (in), \emptyset, \emptyset\rangle, \\
\langle MP7, MP0, \text{進捗状況管理プロセス終了}\ (out), \emptyset, \emptyset\rangle
\end{array}
\right\}
$$

図 A.17: 時間 I/O オートマトンによる記述 (進捗状況管理プロセス)

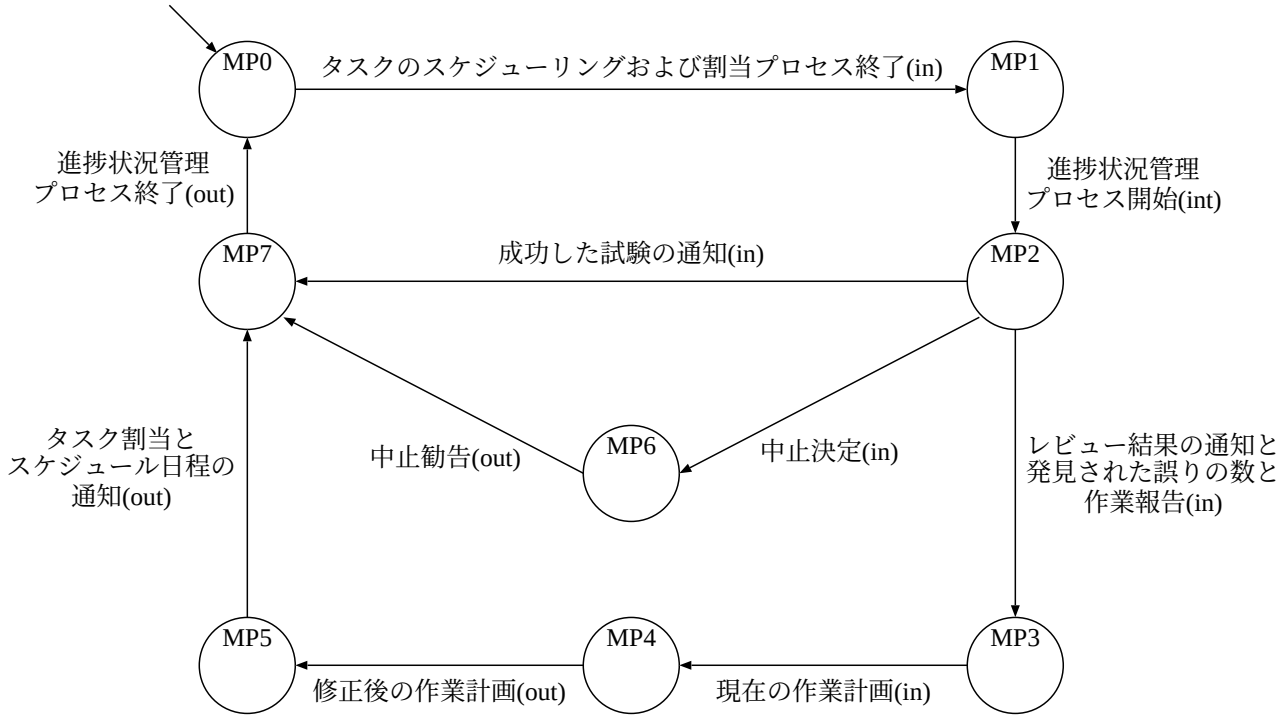


図 A.18: 時間 I/O オートマトンの図的表現 (進捗状況管理プロセス)

ファイル(作業計画) $= \langle \Sigma_{\text{ファイル (作業計画)}}, S_{\text{ファイル (作業計画)}}, S_{0_{\text{ファイル (作業計画)}}}, C_{\text{ファイル (作業計画)}}, E_{\text{ファイル (作業計画)}} \rangle$ に対して、

$$\Sigma_{\text{ファイル (作業計画)}} = \{ \text{作業計画 } (out), \text{現在の作業計画 } (out), \text{修正後の作業計画 } (in), \text{作業計画更新 } (int) \}$$

$$S_{\text{ファイル (作業計画)}} = \{F0, F1\}$$

$$S_{0_{\text{ファイル (作業計画)}}} = \{F0\}$$

$$C_{\text{ファイル (作業計画)}} = \emptyset,$$

$$E_{\text{ファイル (作業計画)}} = \left\{ \begin{array}{l} \langle F0, F0, \text{作業計画 } (out), \emptyset, \emptyset \rangle, \\ \langle F0, F0, \text{現在の作業計画 } (out), \emptyset, \emptyset \rangle, \\ \langle F0, F1, \text{修正後の作業計画 } (in), \emptyset, \emptyset \rangle, \\ \langle F1, F0, \text{作業計画更新 } (int), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.19: 時間 I/O オートマトンによる記述 (ファイル (作業計画))

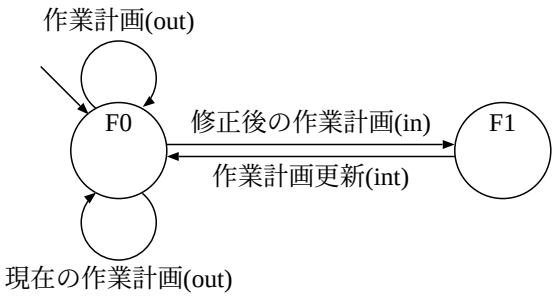


図 A.20: 時間 I/O オートマトンの図的表現 (ファイル (作業計画))

ソフトウェア設計書ファイル (デザイン) $= \langle \Sigma_{\text{ソフトウェア設計書ファイル (デザイン)}}$, $S_{\text{ソフトウェア設計書ファイル (デザイン)}}$, $S_{0_{\text{ソフトウェア設計書ファイル (デザイン)}}}$, $C_{\text{ソフトウェア設計書ファイル (デザイン)}}$, $E_{\text{ソフトウェア設計書ファイル (デザイン)}} \rangle$ に対して、

$\Sigma_{\text{ソフトウェア設計書ファイル (デザイン)}} = \{$ デザイン (out), 認可されたデザイン (in), デザイン更新 $(int)\}$

$S_{\text{ソフトウェア設計書ファイル (デザイン)}} = \{D0, D1\}$

$S_{0_{\text{ソフトウェア設計書ファイル (デザイン)}}} = \{D0\}$

$C_{\text{ソフトウェア設計書ファイル (デザイン)}} = \emptyset,$

$$E_{\text{ソフトウェア設計書ファイル (デザイン)}} = \left\{ \begin{array}{l} \langle D0, D0, \text{デザイン } (out), \emptyset, \emptyset \rangle, \\ \langle D0, D1, \text{認可されたデザイン } (in), \emptyset, \emptyset \rangle, \\ \langle D1, D0, \text{デザイン更新 } (int), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.21: 時間 I/O オートマトンによる記述 (ソフトウェア設計書ファイル (デザイン))

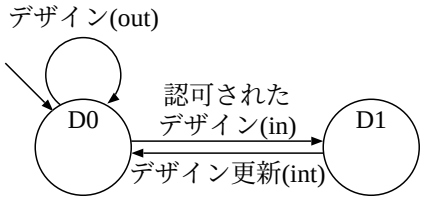


図 A.22: 時間 I/O オートマトンの図的表現 (ソフトウェア設計書ファイル (デザイン))

ソフトウェア開発ファイル (ソースコード) $= \langle \Sigma_{\text{ソフトウェア開発ファイル (ソースコード)}}$, $S_{\text{ソフトウェア開発ファイル (ソースコード)}}$, $S_{0_{\text{ソフトウェア開発ファイル (ソースコード)}}}$, $C_{\text{ソフトウェア開発ファイル (ソースコード)}}$, $E_{\text{ソフトウェア開発ファイル (ソースコード)}} \rangle$ に対して、

$\Sigma_{\text{ソフトウェア開発ファイル (ソースコード)}} = \{$ ソースコード (out), 完成したソースコード (out), ソースコード完成 (in), ソースコード更新 $(int)\}$

$S_{\text{ソフトウェア開発ファイル (ソースコード)}} = \{S0, S1\}$

$S_{0_{\text{ソフトウェア開発ファイル (ソースコード)}}} = \{S0\}$

$C_{\text{ソフトウェア開発ファイル (ソースコード)}} = \emptyset,$

$$E_{\text{ソフトウェア開発ファイル (ソースコード)}} = \left\{ \begin{array}{l} \langle S0, S0, \text{ソースコード } (out), \emptyset, \emptyset \rangle, \\ \langle S0, S0, \text{完成したソースコード } (out), \emptyset, \emptyset \rangle, \\ \langle S0, S1, \text{ソースコード完成 } (in), \emptyset, \emptyset \rangle, \\ \langle S1, S0, \text{ソースコード更新 } (int), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.23: 時間 I/O オートマトンによる記述 (ソフトウェア開発ファイル (ソースコード))

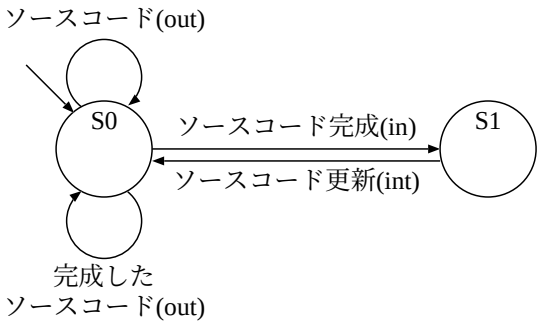


図 A.24: 時間 I/O オートマトンの図的表現 (ソフトウェア開発ファイル (ソースコード))

ソフトウェア開発ファイル (オブジェクトコード) $= \langle \Sigma_{\text{ソフトウェア開発ファイル (オブジェクトコード)}}$, $S_{\text{ソフトウェア開発ファイル (オブジェクトコード)}}$, $S_{0_{\text{ソフトウェア開発ファイル (オブジェクトコード)}}}$, $C_{\text{ソフトウェア開発ファイル (オブジェクトコード)}}$, $E_{\text{ソフトウェア開発ファイル (オブジェクトコード)}} \rangle$ に対して、

$\Sigma_{\text{ソフトウェア開発ファイル (オブジェクトコード)}} = \{$ 完成したオブジェクトコード (out), オブジェクトコード完成 (in), オブジェクトコード更新 $(int)\}$

$S_{\text{ソフトウェア開発ファイル (オブジェクトコード)}} = \{O0, O1\}$

$S_{0_{\text{ソフトウェア開発ファイル (オブジェクトコード)}}} = \{O0\}$

$C_{\text{ソフトウェア開発ファイル (オブジェクトコード)}} = \emptyset,$

$$E_{\text{ソフトウェア開発ファイル (オブジェクトコード)}} = \left\{ \begin{array}{l} \langle O0, O0, \text{完成したオブジェクトコード } (out), \emptyset, \emptyset \rangle, \\ \langle O0, O1, \text{オブジェクトコード完成 } (in), \emptyset, \emptyset \rangle, \\ \langle O1, O0, \text{オブジェクトコード更新 } (int), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.25: 時間I/Oオートマトンによる記述(ソフトウェア開発ファイル(オブジェクトコード))

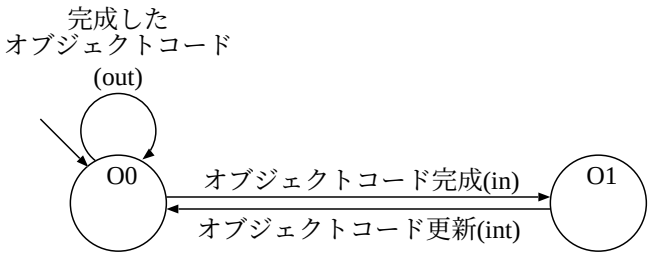


図 A.26: 時間I/Oオートマトンの図的表現(ソフトウェア開発ファイル(オブジェクトコード))

試験計画ファイル(試験計画) $= \langle \Sigma_{\text{試験計画ファイル (試験計画)}}, S_{\text{試験計画ファイル (試験計画)}}, S_{0_{\text{試験計画ファイル (試験計画)}}}, C_{\text{試験計画ファイル (試験計画)}}, E_{\text{試験計画ファイル (試験計画)}} \rangle$ に対して、

$\Sigma_{\text{試験計画ファイル (試験計画)}} =$ { 試験計画 (out), 完成した試験計画 (out), 試験計画完成 (in), 試験計画更新 (int)}

$S_{\text{試験計画ファイル (試験計画)}} = \{T0, T1\}$

$S_{0_{\text{試験計画ファイル (試験計画)}}} = \{T0\}$

$C_{\text{試験計画ファイル (試験計画)}} = \emptyset,$

$$E_{\text{試験計画ファイル (試験計画)}} = \left\{ \begin{array}{l} \langle T0, T0, \text{試験計画 } (out), \emptyset, \emptyset \rangle, \\ \langle T0, T0, \text{完成した試験計画 } (out), \emptyset, \emptyset \rangle, \\ \langle T0, T1, \text{試験計画完成 } (in), \emptyset, \emptyset \rangle, \\ \langle T1, T0, \text{試験計画更新 } (int), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.27: 時間I/Oオートマトンによる記述(試験計画ファイル(試験計画))

試験パッケージファイル(単体試験パッケージ) $= \langle \Sigma_{\text{試験パッケージファイル (単体試験パッケージ)}}, S_{\text{試験パッケージファイル (単体試験パッケージ)}}, S_{0_{\text{試験パッケージファイル (単体試験パッケージ)}}}, C_{\text{試験パッケージファイルファイル (単体試験パッケージ)}}, E_{\text{試験パッケージファイル (単体試験パッケージ)}} \rangle$ に対して、

$\Sigma_{\text{試験パッケージファイル (単体試験パッケージ)}} =$ { 単体試験パッケージ (out), 完成した単体試験パッケージ (out), 単体試験パッケージ完成 (in), 単体試験パッケージ更新 (int)}

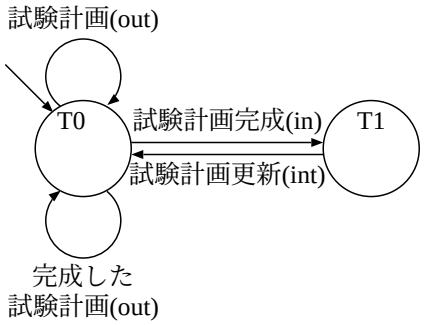


図 A.28: 時間 I/O オートマトンの図的表現 (試験計画ファイル (試験計画))

$$S_{\text{試験パッケージファイル (単体試験パッケージ)}} = \{P0, P1\}$$
$$S_{0_{\text{試験パッケージファイル (単体試験パッケージ)}}} = \{P0\}$$
$$C_{\text{試験パッケージファイル (単体試験パッケージ)}} = \emptyset,$$
$$E_{\text{試験パッケージファイル (単体試験パッケージ)}} = \left\{ \begin{array}{l} \langle P0, P0, \text{単体試験パッケージ}\,(out), \emptyset, \emptyset\rangle, \\ \langle P0, P0, \text{完成した単体試験パッケージ}\,(out), \emptyset, \emptyset\rangle, \\ \langle P0, P1, \text{単体試験パッケージ完成}\,(in), \emptyset, \emptyset\rangle, \\ \langle P1, P0, \text{単体試験パッケージ更新}\,(int), \emptyset, \emptyset\rangle \end{array} \right\}$$

図 A.29: 時間 I/O オートマトンによる記述 (試験パッケージファイル (単体試験パッケージ))

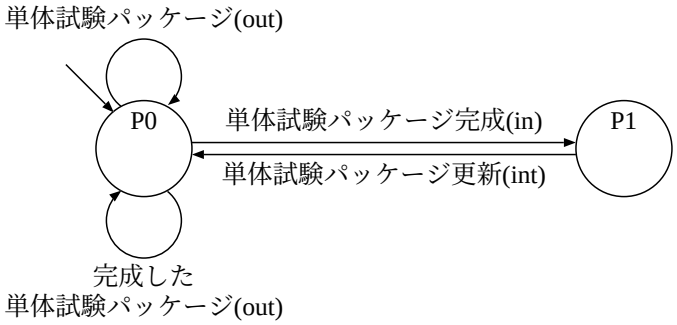


図 A.30: 時間 I/O オートマトンの図的表現 (試験パッケージファイル (単体試験パッケージ))

試験履歴ファイル (試験結果) $= \langle \Sigma_{\text{試験履歴ファイル (試験結果)}}, S_{\text{試験履歴ファイル (試験結果)}}, S_{0_{\text{試験履歴ファイル (試験結果)}}}, C_{\text{試験履歴ファイル (試験結果)}}, E_{\text{試験履歴ファイル (試験結果)}} \rangle$ に対して、

$$\Sigma_{\text{試験履歴ファイル (試験結果)}} = \{\text{ 試験結果 } (in), \text{試験結果の追加 } (int)\}$$

$$S_{\text{試験履歴ファイル (試験結果)}} = \{R0, R1\}$$

$$S_{0_{\text{試験履歴ファイル (試験結果)}}} = \{R0\}$$

$$C_{\text{試験履歴ファイル (試験結果)}} = \emptyset,$$

$$E_{\text{試験履歴ファイル (試験結果)}} = \left\{ \begin{array}{l} \langle R0, R1, \text{試験結果 } (in), \emptyset, \emptyset \rangle, \\ \langle R1, R0, \text{試験結果の追加 } (int), \emptyset, \emptyset \rangle \end{array} \right\}$$

図 A.31: 時間 I/O オートマトンによる記述 (試験履歴ファイル (試験結果))

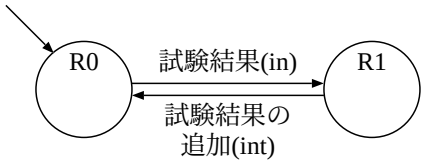


図 A.32: 時間 I/O オートマトンの図的表現 (試験履歴ファイル (試験結果))