修 士 論 文

ObTS で記述した仕様のカラーペトリネットによるテストに関する研究

指導教官　片山卓也 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

810089 西田 雅彦

2000 年 8 月

要 旨

近年では, オブジェクト指向方法論を用いたソフトウエアの開発が注目されている. これは現在のソフトウェアが様々な機能を持ち, 複雑化 · 大規模化したものとなっているためである. 複雑化 · 大規模化したソフトウェアには, 含まれる誤りの数も増加するため, ソフトウェアにおけるテストは重要となる. さらにソフトウエアを開発するとき, 早くから誤りを見つけ出して改善を行ない, 開発を進めていくことは, 開発効率の向上や開発におけるコストの削減において有効である. このようなテスト方法の確立が望まれている.

そこで本研究では, オブジェクト指向を用いたソフトウェアの開発支援となるテスト方法を提案する. このために, まず形式的仕様記述モデル ObTS で並行性のあるモデルについて, その仕様を記述する. 次に, 記述したモデルの仕様を, カラーペトリネット (以下, CPN) へ変換する. そしてテストツール Design/CPN を用いて, 変換した CPN のテストを行ないモデルの問題点を発見する. このような, ObTS で記述したモデルを CPN へ変換して Design/CPN を利用したテスト方法を提案する. このテスト方法で用いる変換において, その変換規則の定義を行う.

目 次

図 目 次

第 1 章

はじめに

1.1 研究の背景

1.1.1 ソフトウエアのテスト

近年におけるハードウェアの劇的な高速化のもとで, ソフトウェア分野の開発では, 対象となるシステム自体に, より高度な処理を求められてきたため大規模化, 複雑化しており, オブジェクト指向を用いて効率よく開発を進めていくことが重要となっている. またソフトウェア開発の際に, ソフトウェアのテスト方法が確立されていることは, 開発における様々なコストに対して重要な役割を果たす. なぜならば, システムをモデル化し, プログラムとして実装していく際には, “誤りを発見し, 改善する” という作業をソフトウェア開発を進める上で, 繰り返し行なうこととなる. さらにソフトウェア開発における早い段階での誤りの改善は, コストに影響を与えることは少ないが, 誤りの発見が遅くなるほど, 改善にかかる時間や費用といったコストに対して非常に大きな影響を与えることとなる.

しかしシステム自体が複雑化, 大規模化しているソフトウェアの開発では, オブジェクト指向を用いても, システムが動的に決定する性質による問題があるため, テストを行なうことが困難となっており, 現在でもソフトウェアのテスト方法は確立されてはいない.

1.1.2 ObTS,ObCL,ObML

ObTS は, 伊藤 [1] が提案したオブジェクト指向における形式的仕様記述モデルである. 記述対象となるシステムにおいて, システムの構造はオブジェクトの階層構造を用いて表し, システムの動作はオブジェクトの状態遷移で表す. また関数による属性計算や属性付きイベントのブロードキャスト通信を用いることでモデル化を行なう.

ObTS で考えたモデルを, 具体的に記述するための仕様記述言語 ObCL がある. ObCL ではオブジェクト, イベント, フィールド, 属性の 4 つのクラスがあり, その集まりと system を一つ記述することで, システムのモデルをプログラムとして表現することができる.

ObML は, 関数型プログラミング言語 Standard ML 上に ObTS の本体と補助関数を組み込んだシミュレータである. ObCL で記述したシステムのコードは, ObCL コンバータを実行して変換することにより Standard ML 上で実行可能なコードの生成を行ない, そのコードを ObML へ読み込み, システムの動作についてシミュレーションを行なうことができる. このシミュレーションで, システムのモデルに対して動作確認による簡単なテストを行なうことができる. この簡単なテストは, テストデータをスクリプト言語により生成して実行するものである。このような簡単なテストでは, ソフトウェアが実行可能ということは言えるが場当たり的なテスト方法であるため, 誤りを見つけ出すような場合は困難である.

1.1.3 カラーペトリネット

1987 年に K. Jensen が提案したカラーペトリネット (CPN:Coloured Petri Nets) は, ペトリネットのトークンにカラーを属性として持たせることにより, 複雑なシステムの記述についても容易に表現すること可能にした概念である. カラーペトリネットは, 並行動作するシステムをモデル化することができ, また, そのソフトウエアの動作をテストすることができる. さらにカラーペトリネットには, 高機能なツールとして Design/CPN があり, このツールを使うことによって多様な問題に関してテストを行なうことができる.

1.2 研究の目的

本研究では, オブジェクト指向を用いたソフトウェアの開発支援となるテスト方法を提案する. この手法では, まず形式的仕様記述モデル ObTS で並行性のあるモデルの仕様を記述する. 次に, 記述したモデルの仕様を, カラーペトリネット (以下, CPN) へ変換する. そしてテストツール Design/CPN を用いて, 変換した CPN のテストを行ないモデルの問題点を発見する. またこのテスト方法において, ObTS で記述した仕様を CPN への変換する規則の定義を行なう.

1.3 論文の構成

本論文の構成は以下の通りである.

第 1 章 本章.

第 2 章 ObTS,ObCL,ObML に概要と特徴.

第 3 章 カラーペトリネットの説明と, その関連情報.

第 4 章 ObTS から CPN への変換規則, 方法や手順.

第 5 章 テストの方法や特徴.

第 6 章 記述例と解析結果.

第 7 章 考察, 評価および今後の課題.

第 8 章 まとめ. 今後の課題.

第 2 章

ObTS,ObCL,ObML

本章では, システムをモデルとして表現する概念の ObTS, そのモデル化したシステムを具体的な構文で記述する言語体系である ObCL, 記述したモデルを実際に動作確認することにより簡単なテストを行なうことができるシミュレーション環境である ObML について説明を行なう.

2.1 仕様記述モデル ObTS

ObTS は, オブジェクト指向による形式的仕様記述モデルである. 記述対象とするシステムのモデル化は, システムの構造をオブジェクトの階層構造として表し, システムの動作をオブジェクトの状態遷移で表す. また, 関数による属性計算や属性付きイベントのブロードキャストにより通信を行なう.

ObTS では, オブジェクトが属性と状態, またその状態遷移を持っており, オブジェクトは動作の委譲のために内部オブジェクトを階層的に定義することができる.

2.1.1 ObTS の特徴

ObTS では, 最初からすべてのオブジェクトを記述しておき, 実行時にオブジェクトの生成, 消滅はないこととする. 発生したイベントは, フィールドという領域の概念を用いて決められた範囲内でのオブジェクト間でイベントのやり取りを行なうことができる.

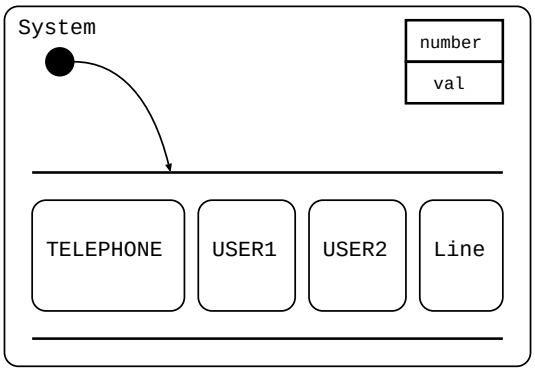

図 2.1: Tel. System

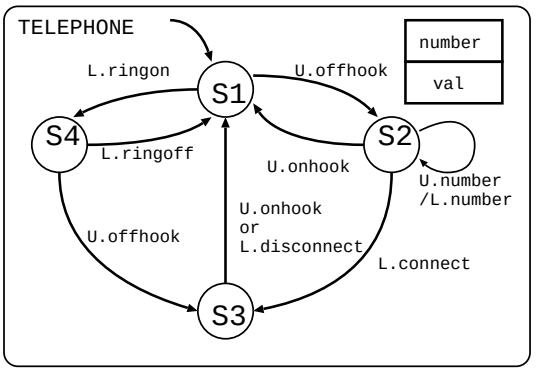

図 2.2: Telephone Object

図 2.1 では, オブジェクトとして System, その内部オブジェクトとして TELEPHONE, USER1,USER2,Line を角の丸い四角で表し, 内部オブジェクトの一つ TELEPHONE オブジェクトについて, 図 2.2 でさらに詳細に記述してある. 状態 S1,S2,S3,S4 は円で表し, 状態遷移は矢印, 初期遷移は黒丸からの矢印, 属性 val, number をオブジェクトの右上の四角として記述している. オブジェクトは, 入力イベントを受け取り, 属性を評価し, 出力イベントを出して状態遷移を行なう.

2.1.2 ObTS の定義

ObTS とオブジェクトの定義

定義 2.1 $ObTS\ M$ **は** $M = (O, E, root)$ **のような 3 つ組で定義される. このとき,**

O **はオブジェクトの集合**
E **はイベントの集合**
$root \in O$ **は最初に動作するオブジェクト**

である.

定義 2.2 オブジェクト $Ob \in O$ **は** $Ob = (Attr, States, Obs, Para, Rules, p_0, E_{io})$ **のように定義される. このとき,**

$Attr$	Ob の属性の集合
$States$	Ob の状態の集合
Obs	Ob の内部オブジェクトの集合
$Para$	並行動作する内部オブジェクトの集合類
	$Para \subset 2^{Obs}$ かつ $(x, y \in Para \wedge x \neq y \rightarrow x \cap y = 0)$
$Rules$	遷移規則の集合
p_0	Ob の初期遷移で, $p_0 \in Rules$
E_{io}	Ob が入出力に使用するイベントの名前の集合 $E_{io} \subseteq E$

である. ただし複数のオブジェクトに渡って議論するときには, $Attr, States, Obs, Para, Rules, E_{io}$ はそれぞれ $Attr(Ob), States(Ob), Obs(Ob), Para(Ob), Rules(Ob), E_{io}(Ob)$ のようにオブジェクト名を付加させる.

オブジェクト $Ob = (Attr, States, Obs, Para, Rules, p_0, E_{io})$ において, $Para$ に含まれない内部オブジェクトの集合を Obs^- と書く.

定義 2.3 Obs^-

$$Obs^- = Obs - \bigcup_{a \in Para} a$$

オブジェクト $Ob = (Attr, States, Obs, Para, Rules, p_0, E_{io})$ において, 状態遷移の遷移元, 遷移先になるものの集合を $\tilde{S}$ と書く. $\tilde{S}$ は状態, 単一の内部オブジェクトおよび並行動作する内部オブジェクトの集合からなっている.

定義 2.4 $\tilde{S}$

$$\tilde{S} = States \cup Obs^- \cup Para$$

状態遷移の対象 $x \in \tilde{S}$ が与えられたとき x 中のオブジェクトの集合を $OS(x)$ とする.

定義 2.5 $OS(x)$

$$OS(x) \iff \begin{cases} \{\} & \textbf{if} \quad x \in States \\ \{x\} & \textbf{if} \quad x \in Obs^- \\ x & \textbf{if} \quad x \in Para \end{cases}$$

オブジェクトの階層構造において, オブジェクト Ob の親オブジェクトは $Parent(Ob)$ とする.

定義 2.6 $Parent$

$$Parent(Ob) = \begin{cases} O_0 & \exists O_0 . Ob \in Obs(O_0) \\ \textbf{undefined} & \textbf{otherwise} \end{cases}$$

オブジェクト Ob の遷移規則において, 参照可能な属性の集合を $A_{ref}(Ob)$ とする.

定義 2.7 A_{ref}

$$A_{ref}(Ob) = \begin{cases} Attr(Ob) \cup A_{ref}(Parent(Ob)) & \textbf{if } Parent(Ob) \textbf{ is defined} \\ Attr(Ob) & \textbf{if } Parent(Ob) \textbf{ is undefined} \end{cases}$$

オブジェクト $Ob = (Attr, States, Obs, Para, Rules, p_0, E_{io})$ **において, 遷移** $p \in Rules$ **は次のように定義される.**

定義 2.8 状態遷移

$$p: \quad X_1 \to X_2 \ ; \ E_1[Cond]/E_2[Eval]$$

ただし,

$X_1, X_2 \in \tilde{S}$ **はそれぞれ遷移元, 遷移先**
$E_1, E_2 \subseteq E_{io}$ **はそれぞれ入力と出力のイベント集合**
$Cond$ **は遷移を実行するためのオブジェクト属性および入力イベント属性に関する条件式**
$Eval$ **は遷移を実行する際の属性評価式のならび**

である. **また** E_1 **にはイベント集合の代わりに, イベント名と** AND,OR,NOT **を用いたイベント論理式を書くことも可能である**. **また, 状態遷移図には図??のように書く**.

遷移の動作としては, X_1 **で入力** E_1 **を受け取ったとき条件** $Cond$ **が成り立っていたら**, X_2 **に遷移して** E_2 **を出力し**, $Eval$ **中の式を計算する**.

オブジェクト $Ob = (Attr, States, Obs, Para, Rules, p_0, E_{io})$ **において**, **初期遷移** $p_0 \in Rules$ **は次のように定義される**.

定義 2.9 初期遷移

$$p_0: \quad Ob \to X_2 \ ; \ /E_2[Eval]$$

ただし,

$X_2 \in \tilde{S}$ **は遷移先**
$E_2 \subseteq E_{io}$ **は出力イベント集合**
$Eval$ **は属性評価式のならび**

である. **初期遷移は各オブジェクトに** 1 **つだけ定義される**. **また**, **状態遷移図には図??のように書く**.

オブジェクト $Ob = (Attr, States, Obs, Para, Rules, p_0, E_{io})$ **の状態遷移**

$$p: \quad X_1 \to X_2 \ ; \ E_1[Cond]/E_2[Eval]$$

において,

定義 2.10 属性評価式は

$$a := exp$$

のように書く. **ただし**,

> a **はオブジェクト属性または出力イベントのイベント属性**
> exp **はその属性の値を計算する式で**, **式中で参照できる属性はオブジェクト属性および入力イベントのイベント属性に限られる**.

以下のような遷移規則 p **が**, **あったとき**,

$$p : X_1 \to X_2; E_1[Cond]/E_2[Eval]$$

それぞれ,

$$\begin{aligned} src(p) &= X_1 \\ dst(p) &= X_2 \\ input(p) &= E_1 \\ cond(p) &= Cond \\ output(p) &= E_2 \\ eval(p) &= Eval \end{aligned}$$

とする.

また, $Cond$ **および** $Eval$ **中に現れるすべての属性の集合を** $Attributes(p)$ **と書くこととする**. $Eval$ **中で特定の属性** a **の値を計算する式を** $ExpOf(a, eval(p))$ **と書くこととする**. **つまり、**

$$ExpOf(a, eval(p)) = \begin{cases} exp & \text{if } (a := exp) \in eval(p) \\ undefined & \text{otherwise} \end{cases}$$

とする.

2.2 仕様記述言語 ObCL

ObTS で記述した図 2.2 が表している TELEPHONE オブジェクトを, 仕様記述言語 ObCL を用いて以下に記述する.

```
– Event class with one attribute
event ATTRIBUTED
  inherit GENERIC_EVENT
  attribute val:Int
end
– Field for User–Phone
field User
  event offhook,onhook:GENERIC_EVENT
  event number:ATTRIBUTED
end
– Field for Line–Phone
field Line
  event connect,disconnect,ringon,ringoff:GENERIC_EVENT
  event number:ATTRIBUTED
end
–Field for Display
fieid Display
  event number:ATTRIBUTED
end
```

–行の先頭にハイフンの着いている行はコメントである.

–ここまではイベントクラスおよびフィールドクラスの宣言の記述

```
class TEL
  field U:User; L:Line; D:Display
  state s1,s2,s3,s4
  transition
  start is
    source init
    destination s1
  end
```

–TELEHONE のクラス TEL では, フィールド U,L,D を用いる. 状態に s1,s2,s3,s4 がある.
–source は遷移元の状態名を示し, destinetion は遷移後の状態を示す.

```
t1a is
  source s1
  input U.offhook
  destination s2
end
```

–状態遷移 t1a では, input は入力イベントを示し,
–入力イベント U.offhook により s1 から s2 への遷移を示している.

```
t2a is
  source s2
  input U.number
  do L.number.val:=U.number.val;
    D.number.val:=U.number.val;
  destination s2
  output L.number,D.number
end
```

–do 以下でイベントを用いて属性値の計算, 受渡しを行なっている.
–output は出力イベントを示す.

```
t2b is
  source s2
  input U.onhook
  destination s1
end
t2c is
  source s2
  input U.connect
  destination s3
end
t3 is
  source s3
  input U.onhook or L.disconnect
  destination s1
```

```
  end
  t1b is
    source s1
    input L.ringon
    destination s4
  end
  t4a is
    source s4
    input L.ringoff
    destination s1
  end
  t4b is
    source s4
    input U.offhook
    destination s3
  end
end
system EXAMPLE
  object tel:TEL
end
```

–system では, オブジェクトの自体と初期状態を記述している.

2.3 ObTS のシミュレーション環境 ObML

ObML は ObTS のシミュレーション環境で, 関数型プログラミング言語 Standard ML 上に構築してあり, ObTS 本体と必要となる補助関数を組み込んだシミュレータとなっている. シミュレータは, 各ステップごとのオブジェクトの状態と, イベントのスナップショットをシミュレーションした結果として返すことができる. また, 記述したシステムに対してテストを行なうことにより, システムの仕様を確かめることができる. テストでは, Standard ML の関数をテストスクリプトとして与えることができる.

ObTS, ObCL, ObML の環境では, システムをモデル化するときから実際に実行可能であり, シミュレーションを行なうことで仕様を確認し, 検討しながらの設計することがで

きる. このことにより, システムのモデル化という開発の早い段階から誤りを発見し, 改善することができる.

第 3 章

カラーペトリネット

この章では, カラーペトリネットについて説明を行なう. また, その説明に必要となる関連知識としてペトリネットについても説明を行なう.

3.1 ペトリネット

ペトリネット (Petri net) は, 多くのシステムに適用可能なグラフィックで数学的なモデル化ツールである. 並行的, 非同期的, 分散的, 並列的, 非決定的, 確率的な動作について記述することができ解析する能力にも優れている. 本節では, ペトリネットの特徴, 定義, 構成, 挙動について示す.

3.1.1 ペトリネットの特徴

ペトリネットの長所は, グラフィックツールとしてソフトウエアの構造を視覚的に把握することができる. またシミュレーションを行なうことにより, ソフトウエアの挙動を確認することができる. このことは一つのテストとすることができ, さらに数学的なツールとして, システムの挙動を示す状態方程式や代数方程式等の数学的モデルを立てることにより, ソフトウエアについて検証することができる.

一方短所として, 記述が複雑であるため設計者が意図した通りにペトリネットを記述することが難しいことが挙げられる.

3.1.2 ペトリネットの定義

ペトリネット PN **の定義を以下に示す。**

$$PN = \{P, T, F, W, M_0\}$$

$P = \{p_1, p_2, \cdots, p_m\}$	**プレースの有限集合**
$T = \{t_1, t_2, \cdots, t_n\}$	**トランジションの有限集合**
$F \subseteq (P \times T) \cup (T \times P)$	**アークの集合**
$W : F \to \{1, 2, 3, \cdots\}$	**重み付け関数**
$M_0 : P \to 0, 1, 2, 3, \cdots$	**初期マーキング**

$P \cap T = \phi$ **でかつ** $P \cup T = \phi$ **である**

特定の初期マーキングが決められていないペトリネット構造は N **で表される**. **初期マーキングが限定されているペトリネットは** (N, M_0) **で表される**.

3.1.3 ペトリネットの構成

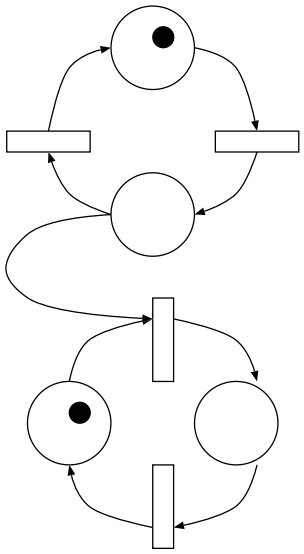

図 3.1: **ペトリネット**

ペトリネットを構成する要素を以下に示す.

- **プレース** (Place : ◯)

  状況を表す.

- **トランジション** (Transition : |)

  事象を表す.

- **アーク** (Arc : →)

  事象と状況の接続関係を表す. **アークの上に数値を付ける場合がある**. **これは多重度と呼ばれ**, **アークを通れるトークンの数を表している**.

- **トークン** (Token : •)

  状況が保持されていることを表すためにプレースに置かれる.

トランジションとプレースの典型的な解釈例を表 3.1 **に示す**.

入力プレース	**トランジション**	**出力プレース**
前提条件	**事象**	**後提条件**
入力データ	**計算過程**	**出力データ**
入力信号	**信号処理**	**出力信号**
要求される資源	**タスク・ジョブ**	**資源の開放**
条件	**論理節**	**結論**
バッファ	**プロセッサ**	**バッファ**

表 3.1: **トランジションとプレースの典型的な解釈例**

3.1.4 ペトリネットの挙動

ペトリネットでは, **現在の状態をプレースにトークンで印付けを行なうことにより表す**. **この印付けをマーキングと呼び**, **マーキングはトランジションの発火により変化する**. **すなわちマーキングはシステムの状態を表し**, **トランジションの発火はシステムの遷移を表す**. **トランジションの発火は**, **発火可能なトランジションのうちの一つに起きる可能性がある**. **発火可能な状態は**, **トランジションに入力している全てのプレースがトークンを持ち**, **その数が**, **アークの多重度を満たしている場合である**. **そして**, **発火すると**, **トランジションへ入力しているプレースからトークンを削除し**, **トランジションから出力しているプレースへトークンが置かれる**. **プレースから削除されるトークンの数と**, **プレースへ置かれるトークンの数はアークの多重度と同じである**.

トランジションの発火の様子を図 3.1.4図 3.1.4に示す. 図 3.1.4 (a) では, トランジションへ入力している全てのプレースがトークンを持ち, アークに添付されている多重度 (2) を満たしているため, (a) は発火可能である. 発火すると (b) のように, アークの多重度に従いトークンが削除され, トランジションから出力しているプレースへアークの多重度に従い, トークンが追加される.

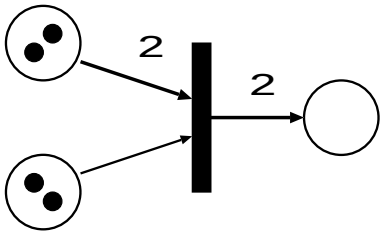

図 3.2: ペトリネットの挙動 (a) 発火前

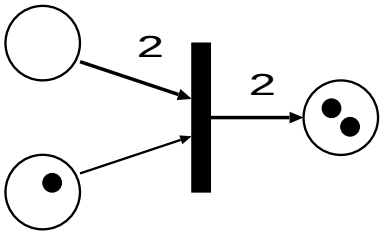

図 3.3: (b) 発火後

3.2 カラーペトリネット

カラーペトリネットは, 並行性を記述したペトリネットのトークンにカラーという属性の概念を取り入れた仕様記述言語である. カラーペトリネットは, ペトリネットでは記述が困難である複雑なシステムについてもモデル化を容易に行なうことができる. しかし, カラーペトリネットは機能の拡張を行なったことにより, 解析能力のがペトリネットと比較すると劣る.

本研究で, カラーペトリネットを用いる理由は 3 つ挙げられる.

- オブジェクト指向に基づくソフトウエアは, 並行な性質を持つことが多いため, 並行性を表現し解析することが容易にできる必要がある.
- オブジェクト指向に基づくソフトウエアは, 非常に複雑な構造を持つことが多いためペトリネットで表現することが困難である. それに対してカラーペトリネットは, カラーを用いることにより並行性を容易に扱うことができる. また, 階層化の概念もあり複雑なものを表現することができるためである.
- カラーペトリネットの解析能力は, ペトリネットによる解析と比べると劣るが, 逆に, より複雑なシステムについても記述, 解析することができるためである.

3.2.1 カラーペトリネットの特徴

上記でも述べたように, カラーペトリネットはペトリネットのトークンにカラーという属性を持ち, より複雑なシステムについても記述を可能にしている. ペトリネットには劣るものの, 解析する能力が高くソフトウエアのテストとして使うことができる.

3.2.2 カラーペトリネットの定義

カラーペトリネットを定義する.
カラーペトリネットは以下の組からなる.
$CPN = (\Sigma, P, T, A, N, C, G, E, I)$

1. Σ は以下のカラー集合である.
   型名 (カラー) の集合を $\mathcal{T}$ とする.$\mathcal{T}$ は, $int, real, char, \cdots$ などの基本型と, 基本型を組合わせたリストや集合型などの複合型の集合である.
   $t \in \mathcal{T}$ に対して, 型 t に属する値の集合を $\mathcal{D}_t$ で表す.
   $\mathcal{D}_t \neq \phi$ 全ての $t \in \mathcal{T}$ である時に, $\Sigma \subseteq \mathcal{T}$

2. P はプレースの有限集合.

3. T はトランジションの有限集合.
   $P \cap T = \phi$

4. A はアークの有限集合.
   $P \cap A = T \cap A = \phi$

5. N はノード関数. 定義は以下の通り.
   $A \to (P \times T) \cup (T \times P)$
   また $N(a) = (p, tr), \quad p \in P, \quad tr \in T$ のとき,

   $$In(a) = p$$
   $$Out(a) = tr$$

   とする.

6. C はカラー関数. 定義は以下の通り.
   $P \to \Sigma$

7. G **はガード関数. 定義は以下の通り.**
   $T \to expr_{\mathcal{B}}$
   $expr_{\mathcal{B}}$ **は, 論理式. すなわち** true **と** false **を値として取る式.**

8. E **はアーク式関数. 定義は以下の通り.**
   型 t **の多重式の集合を** MS **とすると,**

$$MS = \{\sum_{e \in S} n_e `e | s \subseteq t, n_e \in expr_{int}, e \in expr_t\}$$

   t **をこの多重式の型と呼び,** n_e **は多重式の係数である.**

$$A \to MS_\Sigma$$

$$MS_\Sigma = \bigcup_{t \in \Sigma} MS_t$$

   $E(a)$ **の型と** $In(a)$ **の型, または** $E(a)$ **の型と** $Out(a)$ **の型が同じでなければならない.**

   $In(a) \in P$ **のとき,**
   $Type(E(a)) \in Type(In(a))$
   $Out(a) \in P$ **のとき,**
   $Type(E(a)) = Type(Out(a))$

9. I **は初期化関数. 定義は以下の通り.**
   MS_Σ **中に, 変数を含んでいないものを** MS'_Σ **とすると,**
   $P \to MS'_\Sigma$
   $I(p) = MS_t$ **のとき,** C(p)=t **となっていなければならない.**

3.2.3 カラーペトリネットの構成

カラーペトリネットは以下の要素からなる.

- **トークン (データ):**

  カラーペトリネットは, データタイプ, データオブジェクト, そしてデータの値を持つ変数を使用する. カラーペトリネットのデータは, CPN ML **と呼ばれるコンピュータ言語で定義する. また, データの型をカラー** (color) **によって宣言する.**

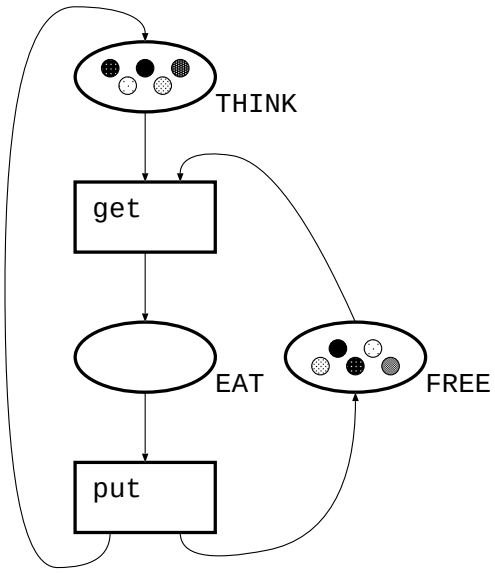

図 3.4: **カラーペトリネットの記述例**

- **カラー (または属性とも呼ぶ):**

  データの型をカラー (color) **によって宣言する**.

- **プレース:**

  データを所有する場所.

- **トランジション:**

  データを変換する発火する場所.

- **アーク:**

  プレースとトランジションを連結し, **データの流れを指定する**.

- **入力アーク式:**

  トークンの発火を起こすのに必要なデータを式で指定する.

- **ガード:**

  トークンの発火を起こすのに、真でなければならない条件を定義する.

- **出力アーク式:**

  トークンの発火が起きた場合に生成されるデータを式で指定する.

初期状態は, プレースに初期マーキングを行なうことによって表す. 初期マーキングは, データの値を文字列で羅列することにより行なう.

3.3 階層的カラーペトリネット

プログラムを記述するとき, 関数や手続きによって階層化して表現するのが一般的である. 仕様を記述する場合でも, システムの多くは 1 枚の図では記述できないほど複雑であり, 階層化して記述することが多い.

今まで述べてきたペトリネットとカラーペトリネットには, このような階層化の概念はない. 階層化の概念を取り入れたモデルが階層的カラーペトリネット (HCPN:Hierarchical Coloured Petri Nets) である. 階層的カラーペトリネットを用いることにより, 関数呼び出しとして記述することができる. また, 階層的カラーペトリネットは一つのカラーペトリネットへまとめることが可能なため, カラーペトリネットと同様の解析能力を持つ.

3.3.1 階層的カラーペトリネットの構成

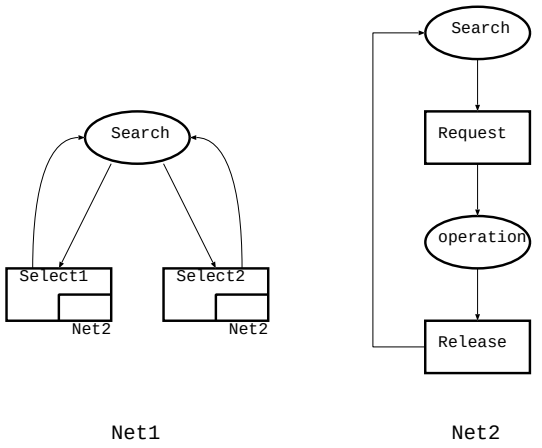

図 3.5: HCPN

カラーペトリネットに階層的なネットの記述を導入したものとして階層的カラーペトリネット (Hierarchical Colored Petri Net) があるこれは以下の要素から構成される.

- 非階層的カラーペトリネットの集合 (非階層的カラーペトリネットはページと呼ばれる)：S

- **代入節点 (トランジションの部分集合)：**SN
- **ページ割当て：**SA:SN $\rightarrow$ S
- **ポート節点 (プレースの部分集合)：**PN
- **ポート型：**PT:PN $\rightarrow$ in,out,i/o,general
- **ポート割当て：**PA:SN **の各節点** t $\hookrightarrow$ (t **に隣接する節点**, SA(t) **のポート節点**) **の対の集合**
- **融合集合：プレースの部分集合の集合** FS
- **融合型：**FT: FS $\rightarrow$ global,page,instance
- **プライムページ：**PP $\subset S_{MS}$

  階層的カラーペトリネットはトランジションを別の一つのカラーペトリネットに対応させることにより定義する.

3.3.2 融合プレース (fusion place)

プログラムを階層的に記述するとき, **大域変数を用いることがある**. **また**, **複数のプロセスを表現するような場合**, **各々のプロセスは別のネットで記述した方が便利である**. **そのような場合**, **共有データを表現できる必要がある**. **主に**, **大域変数や共有データを表現するのに用いられるのが**, **融合プレースである**.

図 3.7**に融合プレースの例を示す**. SubnetA **と** SubnetB **はプレース** Share **を共有している**. **従って**, subnetA **の** gotoShare **トランジションが発火し**, **一つのトークンが** Share **プレースに入ると同時に**, subnetB **の** Share **プレースにも同じトークンが入る**.

融合プレースには, FG **が印付けされており**, FG **の上に記されている名前 この場合** *Share* **が同じものを一つのプレースとして考えることができる**.

3.3.3 階層的カラーペトリネットをカラーペトリネットへまとめる

複数のカラーペトリネットによって書かれたシステムは, **一つのカラーペトリネットにまとめることができる**. **そして複数のカラーペトリネットは**, **一つのカラーペトリネットとして**, **解析を行なうことができる**.

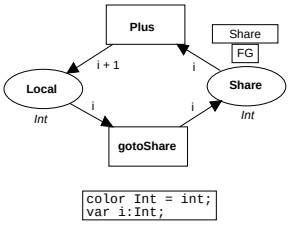

図 3.6: (a)subnetA

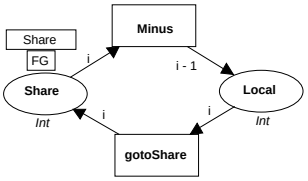

図 3.7: (b)subnetB

図 3.8: **融合プレースの例**

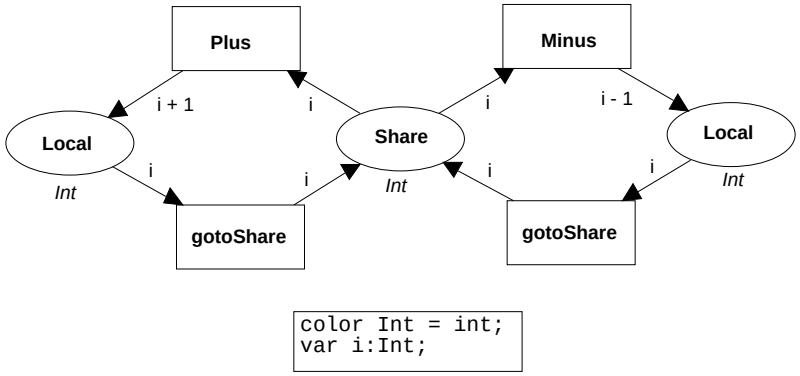

図 3.9: **融合プレースを用いてカラーペトリネットをまとめる**

融合プレースは一つのプレースとしてみなされるため, **図** 3.7**は図** 3.9**のカラーペトリネットとして解釈される**.

以上から分かるように, **カラーペトリネットでの階層化の挙動は**, **実際にはマクロ展開のようなものであり**, **同じネットの呼び出しが多いとき非常に大きなネットになるという問題点がある**.

3.4 Design/CPN

Design/CPN **は**, Meta Software **社が販売している階層化カラーペトリネットのツールである**. **次のような特徴が挙げられる**.

- **階層化カラーペトリネットの定義がグラフィックエディタによりすべて画面上で行なえる**.
- **定義したカラーペトリネットは**, **型付の関数型言語である** Standard ML **を拡張した** CPN ML **に翻訳された後に実行される**. **実行過程におけるトークンの移動は画面上で確認できる**. CPN ML **が備えている抽象データ型の機能を利用したカラー集合の定義が可能である**.
- **トランジションに** CPN ML **のコードを埋め込むことができる**. **これはそのトランジションが発火したときに実行する**.
- **トランジションが発火してから出力プレースにトークンが出現するまでの時間を定義できる**. **この時間は実時間ではなく**, **トークンに付随する数値として与えられる**.

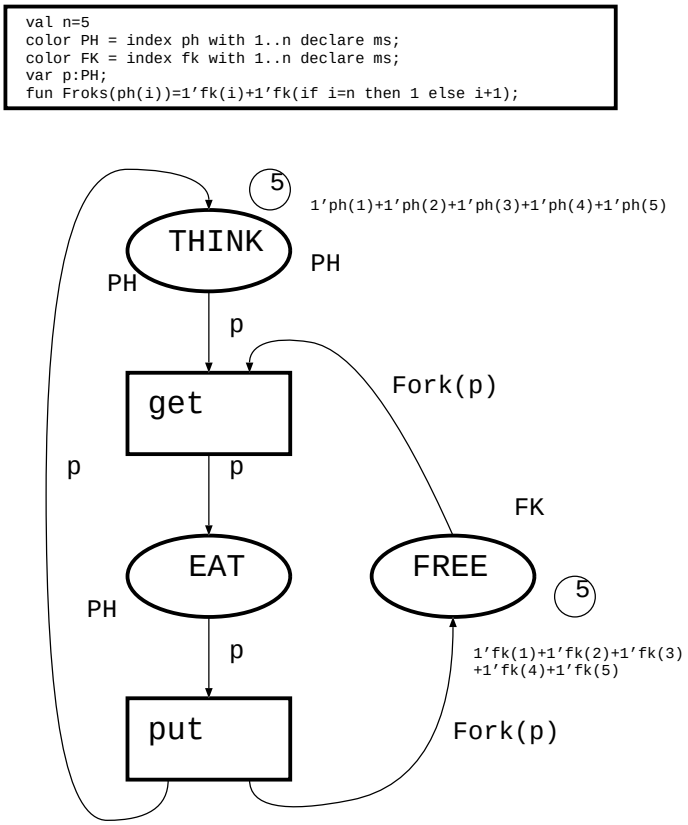

図 3.10: Design/CPN **の記述例**

第 4 章

ObTS から CPN への変換

4.1 概要

まず最初に本章で述べる変換規則に関する前提条件と, 変換規則に必要な補助定義を述べる. 次に ObTS で書かれた記述を階層的 CPN に変換する規則を述べる. 変換後の CPN が ObTS のセマンティクスを考慮せずに構造のみを変換する規則を述べ, その後で ObTS セマンティクスに基づいて動作させるための拡張された変換規則を述べる.

4.2 準備

4.2.1 前提

本章で述べる変換規則においては, ObTS の記述中に現れる属性の計算式そのものや属性に関する条件式そのものの変換については扱わないものとする. 実際には, ObTS(ObCL, ObML) で主に扱われる整数, 文字列, 論理型の値に関する演算は, Design/CPN でそのまま扱うことができる.
また, ここでは説明を簡単にするために, 扱うイベントは整数属性を一つだけ持つイベントのみとすることが, この変換規則を ObTS で扱うことのできる一般のイベントに対応させることは可能である.

4.2.2 階層的カラーペトリネットの合成

変換規則を定義する前に, ネットの合成に関する演算子を以下のように定義する.

$$H = H_1 \oplus H_2$$

$H_1 = (S_1, SN_1, SA_1, PN_1, PT_1, PA_1, FS_1, FT_1, PP_1)$ で,
$H_2 = (S_2, SN_2, SA_2, PN_2, PT_2, PA_2, FS_2, FT_2, PP_2)$ のとき,
$H = (S, SN, SA, PN, PT, PA, FS, FT, PP)$ は次のように求められる.

$$\begin{aligned}
S &= S_1 \cup S_2 \\
SN &= SN_1 \cup SN_2 \\
SA &= SA_1 \cup SA_2 \\
PN &= PN_1 \cup PN_2 \\
PT &= PT_1 \cup PT_2 \\
PA &= PA_1 \cup PA_2 \\
FS &= FS_1 \cup FS_2 \\
FT &= FT_1 \cup FT_2 \\
PP &= PP_1 \cup PP_2
\end{aligned}$$

ただし, H_1 と H_2 に含まれる個々のサブネット中の要素について, すべてのトランジションの名前, および, 融合プレースやポート節点に含まれないプレースの名前は重複の無いように改名されるものとする.

4.3 基本変換規則

本節では ObTS から階層的 CPN への変換規則を定義する. ここで述べる変換規則は, ObTS 図や ObCL コードに現れる構造を, そのまま CPN に変換するものであり, (図やコードに現れない)ObTS の動作セマンティクスを考慮していない. ObTS の動作セマンティクスに基づいて CPN が動作するためには, 本節の変換規則の他に, 次節で述べる拡張変換規則を用いる必要がある.

4.3.1 ObTS 全体の変換

ある ObTS $M = (O, E, root)$ を階層的カラーペトリネット
$H = (S, SN, SA, PN, PT, PA, FS, FT, PP)$ に変換することを

$$\mathcal{C}_M(O, E, root) = (S, SN, SA, PN, PT, PA, FS, FT, PP)$$

とする.

実際には ObTS M のオブジェクト集合 O のうち, $root$ を除くすべてのオブジェクトは $root$ オブジェクトの階層下にあり, 階層構造は木構造になっている. また, イベントの集合 E はに登場するすべてのオブジェクトの入出力イベント集合の和集合と厳密に等しい (つまり $E = \bigcap_{o \in \mathcal{O}} E_{io}(O)$) とすれば, 階層構造をたどれるようにオブジェクトに対する変換関数 $\mathcal{C}_O$ を再帰的に定義すれば, オブジェクトを変換して得られる階層的カラーペトリネットが, 目的の階層的カラーペトリネット $\mathcal{H}$ となる. つまり,

$$\mathcal{C}_M(O, E, root) = \mathcal{C}_O(root)$$

となる.

4.3.2 オブジェクトの変換

ObTS でモデル化した際の, それぞれのオブジェクトについて以下で定義する規則を用いて CPN へ変換する. 具体的には ObTS オブジェクトの状態遷移図に現れる状態と遷移をそれぞれ CPN のプレースとトランジションにする. 状態遷移の詳細は後に述べる遷移規則ネットで表し, また内部オブジェクトは本節の変換規則を再帰的に用いて変換し, それらを 4.2.2で定義したように合成することで, ObTS オブジェクトに対する CPN が得られる.

$$\mathcal{C}_O(obj) = (S, SN, SA, PN, PT, PA, FS, FT, PP) \oplus \left(\bigoplus_{p \in Rules} \mathcal{C}_{trans}(p) \right) \oplus \left(\bigoplus_{o \in Obs} \mathcal{C}_O(o) \right)$$

obj が $(Attr, States, Obs, Para, Rules, p_0, E_{io})$ で示される ObTS オブジェクトであるとき,

$$\begin{aligned} S &= \{(\Sigma, P, T, A, N, C, G, E, I)\} \\ \Sigma &= \{INT, STRING, NAME, STEP, ST\} \end{aligned}$$

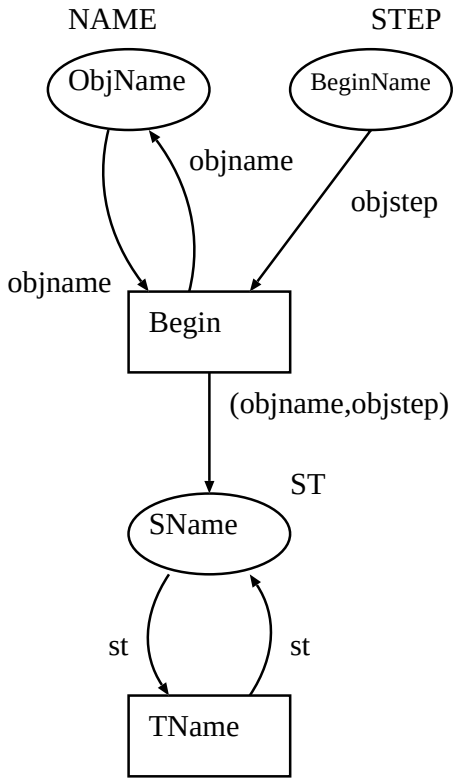

図 4.1: **オブジェクトネット**

$$
\begin{aligned}
P &= \{ObjName, BeginName\} \cup States \\
T &= \{Begin\} \cup \{Name(p) | p \in Rules\} \\
&\quad \textbf{ただし}, Name(p) \textbf{は遷移規則} \ p \ \textbf{を一意に識別する名前を返す関数} \\
A &= \{ObjName \to Begin, Begin \to ObjName, BeginName \to Begin\} \\
&\cup \{s \to t | s = src(p) \wedge t = Name(p) \wedge p \in Rules\} \\
&\cup \{t \to s | t = Name(p) \wedge s = dst(p) \wedge p \in Rules\} \\
N &= \{(x \to y, (x, y)) | x \to y \in A\} \\
C &= \{(ObjName, NAME), (BeginName, STEP)\} \cup \{(s, ST) | s \in States\} \\
G &= \{(Begin, true)\} \cup \{(Name(p), cond(p)) | p \in Rules\} \\
E &= \{(ObjName \to Begin, objname), (Begin \to ObjName, objname), \\
&\quad (BeginName \to Begin, objstep)\} \\
&\cup \{(s \to t, st) | s = src(p) \wedge t = Name(p) \wedge p \in Rules\}
\end{aligned}
$$

$$
\begin{aligned}
& \cup \quad \{(t \to s, st) | t = Name(p) \wedge s = dst(p) \wedge p \in Rules\} \\
I &= (ObjName, \text{""}) \\
SN &= \{Name(p) | p \in Rules\} \\
SA &= \{(Name(p), \mathcal{C}_{trans}(p)) | p \in Rules\} \\
PN &= \{src(p) | p \in Rules\} \cup \{dst(p) | p \in Rules\} \\
PT &= \{(src(p), general) | p \in Rules\} \cup \{(dst(p), general) | p \in Rules\} \\
PA &= \{(t, (src(p), SA(t) \textbf{の} TrIn)) | t = Name(p) \wedge p \in Rules\} \\
& \cup \quad \{(t, (dst(p), SA(t) \textbf{の} TrOut)) | t = Name(p) \wedge p \in Rules\} \\
FS &= \{\} \\
FT &= \{(x, global) | x \in FS\} \\
PP &= \{(p, 1) | p \in S\}
\end{aligned}
$$

4.3.3 状態遷移規則の変換

ObTS **における個々の遷移に対応する** CPN **として以下に示すように表現し**, **定義とする**. **ただし**, ObTS **側での遷移が自己遷移であるときは** TrIn, TrOut **の二つのプレースを** TrIO **とまとめる**.

$$\mathcal{C}_{trans}(p) = (S, SN, SA, PN, PT, PA, FS, FT, PP)$$

1. $src(p) \neq dst(p)$ の場合:

$$
\begin{aligned}
S &= \{(\Sigma, P, T, A, N, C, G, E, I)\} \\
\Sigma &= \{INT, ST, STT, EV\} \\
P &= \{TrIn, St, Val, TranID, TrOut\} \\
& \cup \quad input(p) \cup output(p) \cup Attributes(eval(p)) \\
T &= \{TrBegin, Cal, TrEnd\} \\
A &= \{TrIn \to TrBegin, TrBegin \to St, TrBegin \to Val, St \to Cal, Val \to Cal\} \\
& \cup \quad \{Cal \to TranID, TranID \to TrEnd, TrEnd \to TrOut\} \\
& \cup \quad \{e \to TrBegin | e \in input(p)\} \\
& \cup \quad \{TrEnd \to e | e \in output(p)\}
\end{aligned}
$$

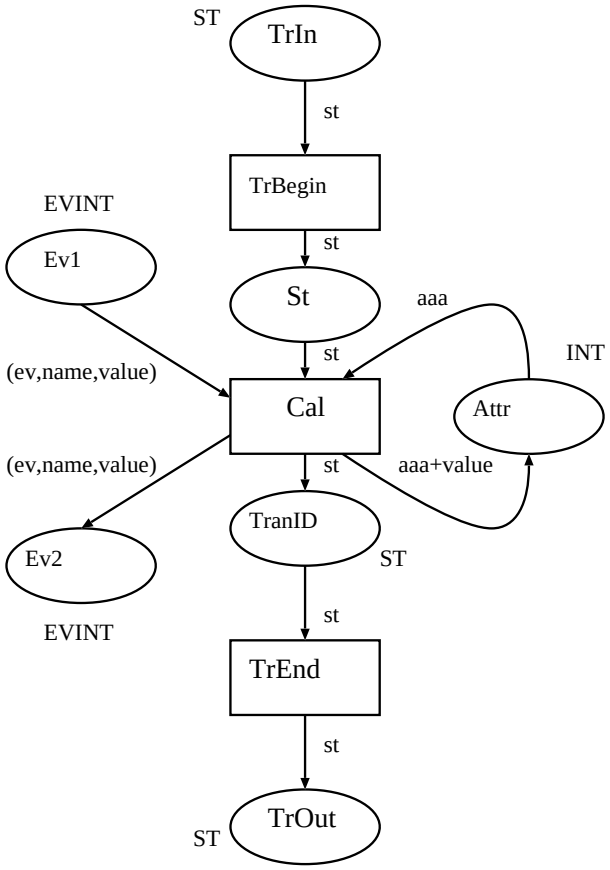

図 4.2: **遷移規則ネット**

$$
\begin{aligned}
& \cup \quad \{a \to Cal | a \in Attributes(eval(p))\} \\
& \cup \quad \{Cal \to a | a \in Attributes(eval(p))\} \\
N \quad &= \quad \{(x \to y, (x, y)) | x \to y \in A\} \\
C \quad &= \quad \{(TrIn, ST), (Val, INT), (St, ST), (TranID, ST), (TrOut, ST)\} \\
& \cup \quad \{(e, EV) | e \in input(p) \cup output(p)\} \\
& \cup \quad \{(a, INT) | a \in Attributes(eval(p))\} \\
G \quad &= \quad \{(t, true) | t \in T\} \\
E \quad &= \quad \{(TrIn \to TrBegin, st), (TrBegin \to St, st), (TrBegin \to Val, value), (Val \to Cal, value), \\
& \qquad (St \to Cal, st)\} \\
& \cup \quad \{(Cal \to TranIn, st), (Cal \to TranID, st), (TranID \to TrEnd, st)\}
\end{aligned}
$$

$$
\begin{aligned}
&\cup \{(TrEnd \to TrOut, st)\} \\
&\cup \{(e \to TrBegin, (ev, name, value))|e \in input(p)\} \\
&\cup \{(TrEnd \to e, (ev, name, value))|e \in output(p)\} \\
&\cup \{(a \to Cal, a)|a \in Attributes(eval(p))\} \\
&\cup \{(Cal \to a, exp)|a \in Attributes(eval(p)) \wedge exp = ExpOf(a, eval(p))\} \\
I &= \{\} \\
SN &= \{\} \\
SA &= \{\} \\
PN &= \{TrIn, TrOut\} \\
PT &= \{(TrIn, in), (TrOut, out)\} \\
PA &= \{\} \\
FS &= \{\{e\}|e \in input(p) \cup output(p) \cup Attributes(eval(p))\} \\
FT &= \{(x, global)|x \in FS\} \\
PP &= \{(p, 1)|p \in S\}
\end{aligned}
$$

2. src(p)=dst(p) の場合:

$$
\begin{aligned}
S &= \{(\Sigma, P, T, A, N, C, G, E, I)\} \\
\Sigma &= \{INT, ST, STT, EV\} \\
P &= \{St, Val, TranID, TrIO\} \\
&\cup input(p) \cup output(p) \cup Attributes(eval(p)) \\
T &= \{TrBegin, Cal, AddID, TrEnd\} \\
A &= \{TrIO \to TrBegin, TrBegin \to St, TrBegin \to Val, St \to Cal, Val \to Cal\} \\
&\cup \{Cal \to TranID, TranID \to TrIO\} \\
&\cup \{TrEnd \to TrIO\} \\
&\cup \{e \to TrBegin|e \in input(p)\} \\
&\cup \{TrEnd \to e|e \in output(p)\} \\
&\cup \{a \to Cal|a \in Attributes(eval(p))\} \\
&\cup \{Cal \to a|a \in Attributes(eval(p))\} \\
N &= \{(x \to y, (x, y))|x \to y \in A\}
\end{aligned}
$$

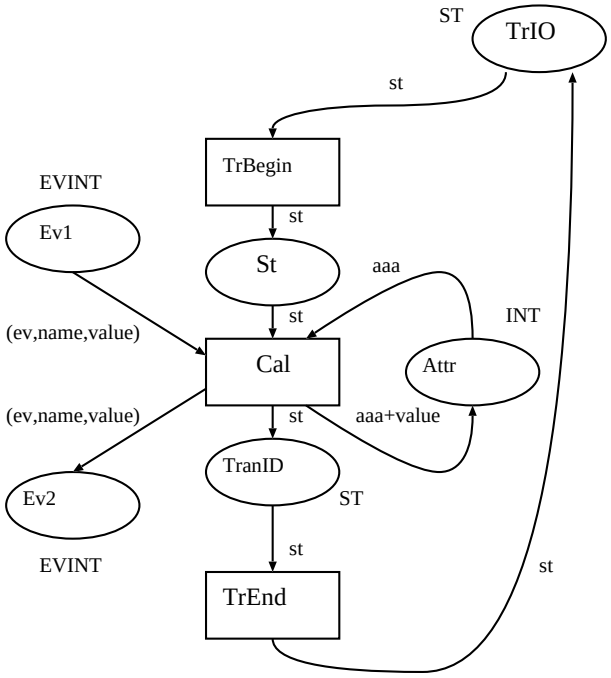

図 4.3: **入出力イベントでの遷移規則ネット**

$$
\begin{aligned}
C &= \{(TrIO, ST), (Val, INT), (St, ST), (TranID, ST)\} \\
&\cup \{(e, EV) | e \in input(p) \cup output(p)\} \\
&\cup \{(a, INT) | a \in Attributes(eval(p))\} \\
G &= \{(t, true) | t \in T\} \\
E &= \{(TrIO \rightarrow TrBegin, st), (TrBegin \rightarrow St, st), (TrBegin \rightarrow Val, value), (Val \rightarrow Cal, value), \\
&\quad (St \rightarrow Cal, st)\} \\
&\cup \{((Cal \rightarrow TranID, st), (TranID \rightarrow TrEnd, st))\} \\
&\cup \{(TrEnd \rightarrow TrIO, st)\} \\
&\cup \{(e \rightarrow TrBegin, (ev, name, value)) | e \in input(p)\} \\
&\cup \{(TrEnd \rightarrow e, (ev, name, value)) | e \in output(p)\} \\
&\cup \{(a \rightarrow Cal, a) | a \in Attributes(eval(p))\} \\
&\cup \{(Cal \rightarrow a, exp) | a \in Attributes(eval(p)) \wedge exp = ExpOf(a, eval(p))\} \\
I &= \{\} \\
SN &= \{\}
\end{aligned}
$$

$$
\begin{aligned}
SA &= \{\} \\
PN &= \{TrIO\} \\
PT &= \{(TrIO, i/o)\} \\
PA &= \{\} \\
FS &= \{\{e\} | e \in input(p) \cup output(p) \cup Attributes(eval(p))\} \\
FT &= \{(x, global) | x \in FS\} \\
PP &= \{(p, 1) | p \in S\}
\end{aligned}
$$

4.4 拡張変換規則

前節で述べた基本変換規則では, ObTS **の動作セマンティクスを考慮していないため**, **変換前の** ObTS **の記述と変換後の** CPN **の記述の動作が異なってしまう場合がある**. **そこで**, **本節では変換後の** CPN **が** ObTS **の動作セマンティクスに基づいて動作するように拡張した変換規則を定義する**.

基本変換規則に対して, **オブジェクトネットの動作を制御するためのスケジューラネット**, **および**, **イベントのブロードキャストを実現するためのイベント管理ネットを追加するとともに**, **それに対応してオブジェクトネットと遷移規則ネットの変換規則も拡張する**.

4.4.1 オブジェクトの変換

後に述べるスケジューラネットの導入に当たり, ObTS **セマンティクスの個々の『ステップ』において**, **個々のオブジェクトが実際には状態遷移を起こさない場合にも**, **『状態遷移を起こさなかった』ことをスケジューラネットに通知する必要がある**. **そこで**, **オブジェクトの変換自体は先に述べたものと同様でよいが**, **個々のオブジェクトの状態遷移図に対して**, **ダミーの遷移規則を追加しながら変換する必要がある**.

4.4.2 状態遷移規則の変換

ObTS における個々の遷移に対応する CPN として以下に示すように表現し, 定義とする. ただし, ObTS 側での遷移が自己遷移であるときは TrIn, TrOut の二つのプレースを TrIO とまとめる.

$$\mathcal{C}_{trans}(p) = (S, SN, SA, PN, PT, PA, FS, FT, PP)$$

1. $src(p) \neq dst(p)$ の場合:

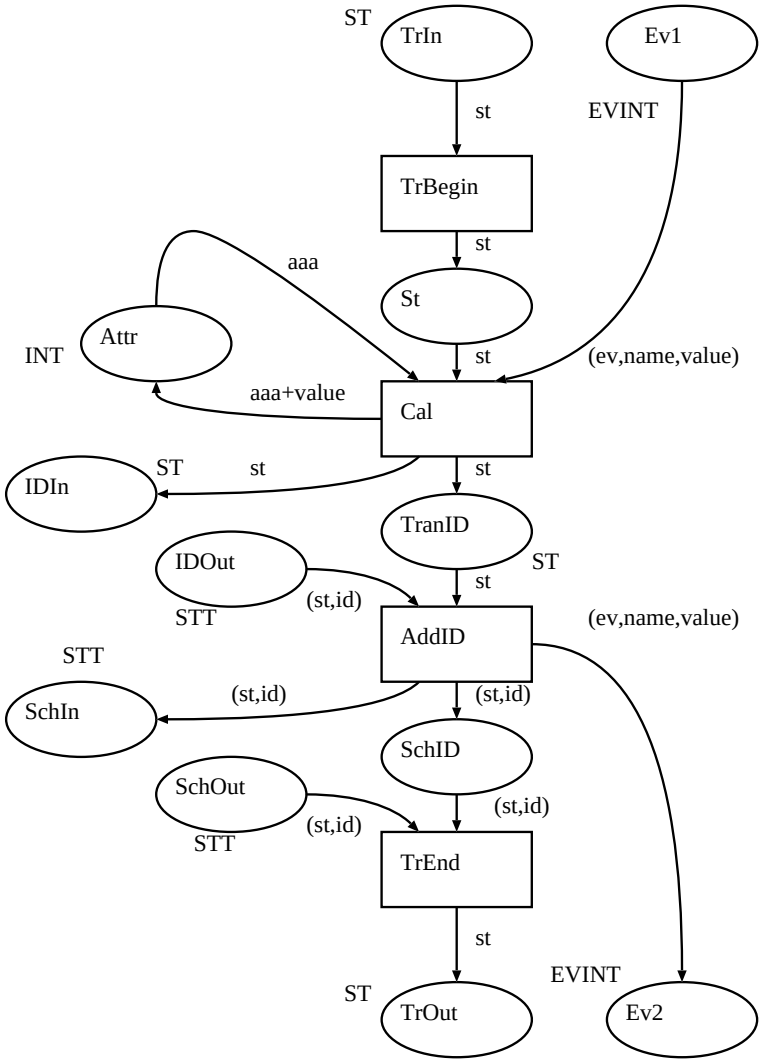

図 4.4: 動的セマンティクスを考慮した遷移規則ネット

$$
\begin{aligned}
S &= \{(\Sigma, P, T, A, N, C, G, E, I)\} \\
\Sigma &= \{INT, ST, STT, EV\} \\
P &= \{TrIn, St, Val, IDIn, TranID, IDOut, SchIn, SchID, SchOut, TrOut\} \\
&\cup\ input(p) \cup output(p) \cup Attributes(eval(p)) \\
T &= \{TrBegin, Cal, AddID, TrEnd\} \\
A &= \{TrIn \to TrBegin, TrBegin \to St, TrBegin \to Val, St \to Cal, Val \to Cal\} \\
&\cup\ \{Cal \to IDIn, Cal \to TranID, IDOut \to AddID, TranID \to AddID\} \\
&\cup\ \{AddID \to SchIn, AddID \to SchID, SchOut \to TrEnd, SchID \to TrEnd, \\
&\quad TrEnd \to TrOut\} \\
&\cup\ \{e \to TrBegin | e \in input(p)\} \\
&\cup\ \{TrEnd \to e | e \in output(p)\} \\
&\cup\ \{a \to Cal | a \in Attributes(eval(p))\} \\
&\cup\ \{Cal \to a | a \in Attributes(eval(p))\} \\
N &= \{(x \to y, (x, y)) | x \to y \in A\} \\
C &= \{(TrIn, ST), (Val, INT), (St, ST), (IDIn, ST), (TranID, ST), (IDOut, STT)\} \\
&\cup\ \{(SchIn, STT), (SchID, STT), (SchOut, STT), (TrOut, ST)\} \\
&\cup\ \{(e, EV) | e \in input(p) \cup output(p)\} \\
&\cup\ \{(a, INT) | a \in Attributes(eval(p))\} \\
G &= \{(t, true) | t \in T\} \\
E &= \{(TrIn \to TrBegin, st), (TrBegin \to St, st), (TrBegin \to Val, value), (Val \to Cal, value), \\
&\quad (St \to Cal, st)\} \\
&\cup\ \{(Cal \to IDIn, st), (Cal \to TranID, st), (TranID \to AddID, st), \\
&\quad (IDOut \to AddID, (st, id))\} \\
&\cup\ \{(AddID \to SchIn, (st, id)), (AddID \to SchID, (st, id)), (SchOut \to TrEnd, (st, id)), \\
&\quad (SchID \to TrEnd, (st, id)\} \\
&\cup\ \{(TrEnd \to TrOut, st)\} \\
&\cup\ \{(e \to TrBegin, (ev, name, value)) | e \in input(p)\} \\
&\cup\ \{(TrEnd \to e, (ev, name, value)) | e \in output(p)\}
\end{aligned}
$$

$$
\begin{aligned}
&\cup \ \{(a \to Cal, a) | a \in Attributes(eval(p))\} \\
&\cup \ \{(Cal \to a, exp) | a \in Attributes(eval(p)) \wedge ExpOf(a, eval(p))\} \\
I &= \{\} \\
SN &= \{\} \\
SA &= \{\} \\
PN &= \{TrIn, TrOut\} \\
PT &= \{(TrIn, in), (TrOut, out)\} \\
PA &= \{\} \\
FS &= \{\{IDIn\}, \{IDOut\}, \{SchIn\}, \{SchOut\}\} \\
&\cup \ \{\{e\} | e \in input(p) \cup output(p) \cup Attributes(eval(p))\} \\
FT &= \{(x, global) | x \in FS\} \\
PP &= \{(p, 1) | p \in S\}
\end{aligned}
$$

2. src(p)=dst(p) の場合:

$$
\begin{aligned}
S &= \{(\Sigma, P, T, A, N, C, G, E, I)\} \\
\Sigma &= \{INT, ST, STT, EV\} \\
P &= \{TrIn, St, Val, IDIn, TranID, IDOut, SchIn, SchID, SchOut, TrOut\} \\
&\cup \ input(p) \cup output(p) \cup Attributes(eval(p)) \\
T &= \{TrBegin, Cal, AddID, TrEnd\} \\
A &= \{TrIn \to TrBegin, TrBegin \to St, TrBegin \to Val, St \to Cal, Val \to Cal\} \\
&\cup \ \{Cal \to IDIn, Cal \to TranID, IDOut \to AddID, TranID \to AddID\} \\
&\cup \ \{AddID \to SchIn, AddID \to SchID, SchOut \to TrEnd, SchID \to TrEnd, \\
&\quad TrEnd \to TrOut\} \\
&\cup \ \{e, TrBegin \to | e \in input(p)\} \\
&\cup \ \{TrEnd \to e | e \in output(p)\} \\
&\cup \ \{a \to Cal | a \in Attributes(eval(p))\} \\
&\cup \ \{Cal \to a | a \in Attributes(eval(p))\} \\
N &= \{(x \to y, (x, y)) | x \to y \in A\} \\
C &= \{(TrIn, ST), (Val, INT), (St, ST), (IDIn, ST), (TranID, ST), (IDOut, STT)\}
\end{aligned}
$$

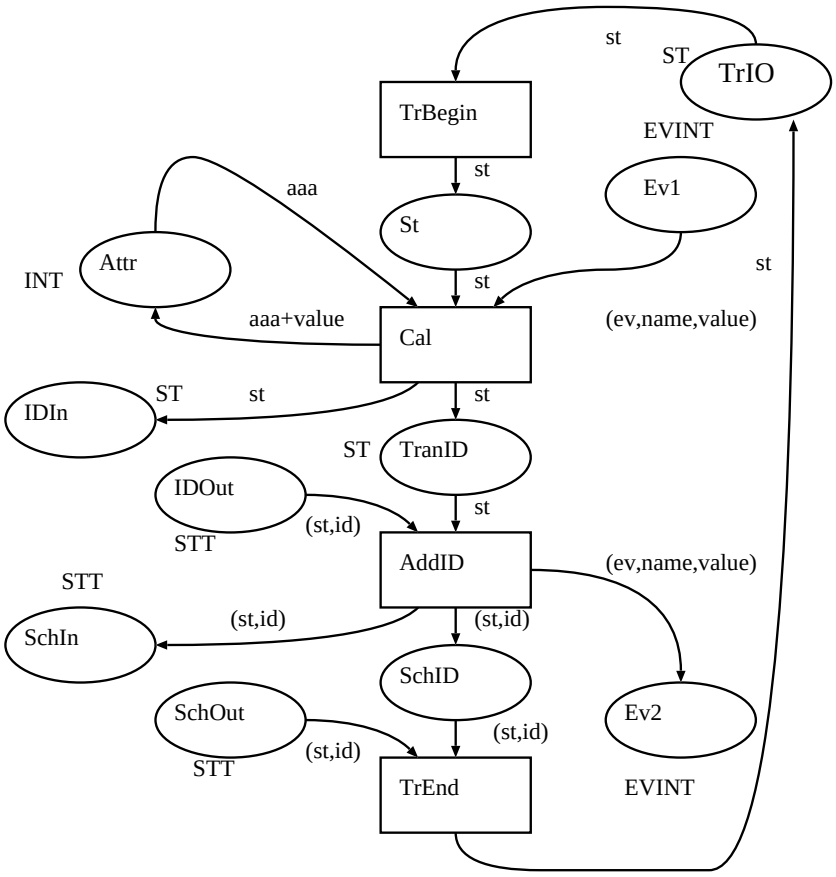

図 4.5: **入出力イベントでのセマンティクスを考慮した遷移規則ネット**

$$
\begin{aligned}
&\cup \quad \{(SchIn, STT), (SchID, STT), (SchOut, STT), (TrOut, ST)\} \\
&\cup \quad \{(e, EV) | e \in input(p) \cup output(p)\} \\
&\cup \quad \{(a, INT) | a \in Attributes(eval(p))\} \\
G \quad &= \quad \{(t, true) | t \in T\} \\
E \quad &= \quad \{(TrIn \to TrBegin, st), (TrBegin \to St, st), (TrBegin \to Val, value), (Val \to Cal, value), \\
&\quad (St \to Cal, st)\} \\
&\cup \quad \{(Cal \to IDIn, st), (Cal \to TranID, st), (TranID \to AddID, st), \\
&\quad (IDOut \to AddID, (st, id))\} \\
&\cup \quad \{(AddID \to SchIn, (st, id)), (AddID \to SchID, (st, id)), (SchOut \to TrEnd, (st, id)), \\
&\quad (SchID \to TrEnd, (st, id)\}
\end{aligned}
$$

$$
\begin{aligned}
& \cup \{(TrEnd \rightarrow TrOut, st)\} \\
& \cup \{(e \rightarrow TrBegin, (ev, name, value))|e \in input(p)\} \\
& \cup \{(TrEnd \rightarrow e, (ev, name, value))|e \in output(p)\} \\
& \cup \{(a \rightarrow Cal, a)|a \in Attributes(eval(p))\} \\
& \cup \{(Cal \rightarrow a, exp)|a \in Attributes(eval(p)) \wedge ExpOf(a, eval(p))\} \\
I &= \{\} \\
SN &= \{\} \\
SA &= \{\} \\
PN &= \{TrIn, TrOut\} \\
PT &= \{(TrIn, in), (TrOut, out)\} \\
PA &= \{\} \\
FS &= \{\{IDIn\}, \{IDOut\}, \{SchIn\}, \{SchOut\}\} \\
& \cup \{\{e\}|e \in input(p) \cup output(p) \cup Attributes(eval(p))\} \\
FT &= \{(x, global)|x \in FS\} \\
PP &= \{(p, 1)|p \in S\}
\end{aligned}
$$

4.4.3 スケジューラネット

このスケジューラネットは ObTS **での, ステップを表しており, 状態遷移における同期を取っている. スケジューラネットは全体で一つ用意する.**

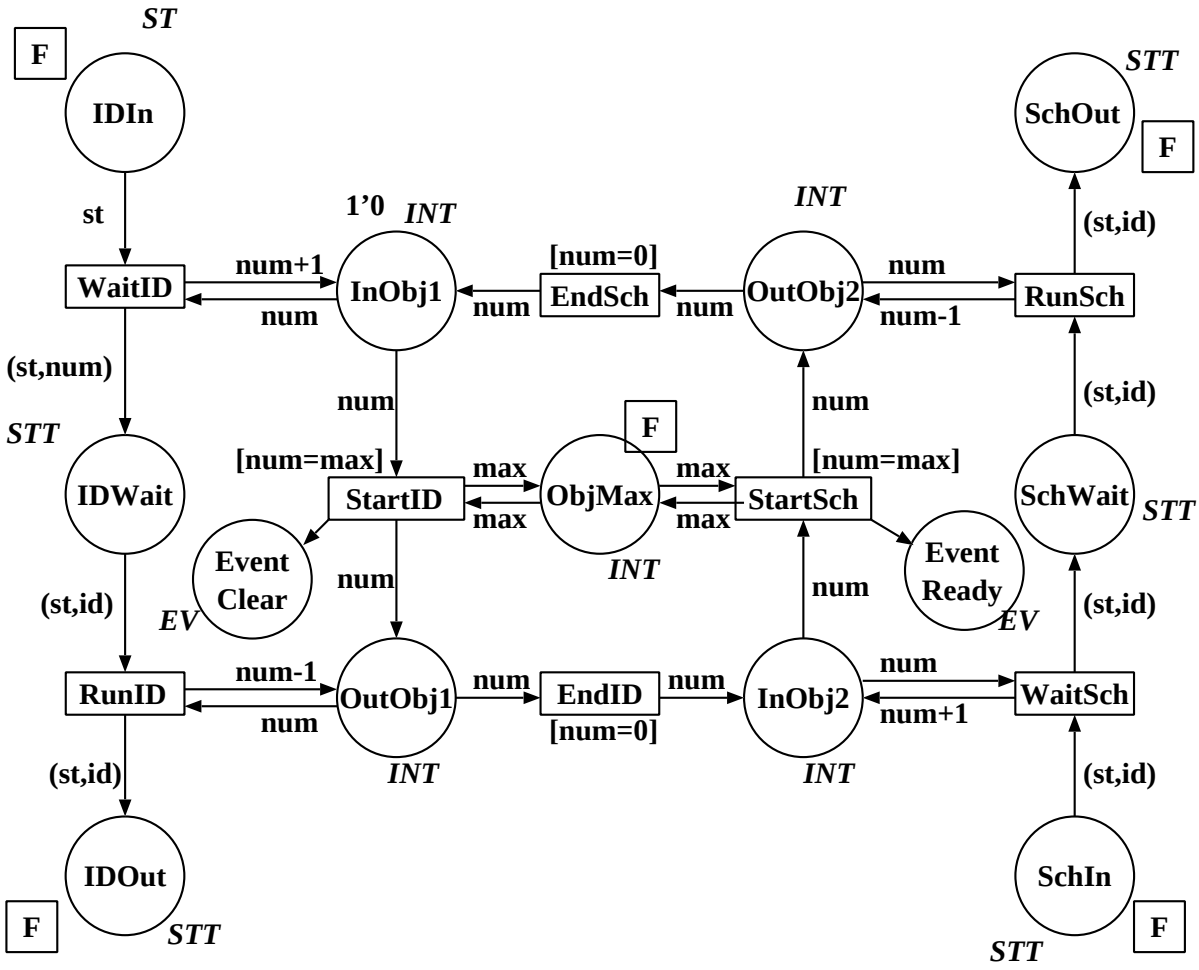

図 4.6: **スケジューラネット**

$$H_{sch} = (S, SN, SA, PN, PT, PA, FS, FT, PP)$$

$$\begin{aligned} S &= \{(\Sigma, P, T, A, N, C, G, E, I)\} \\ \Sigma &= \{ST, STT, INT, EV\} \\ P &= \{IDIn, IDWait, IDOut, SchIn, SchWait, SchOut, ObjMax\} \end{aligned}$$

$$
\begin{aligned}
&\cup \{InObj1, OutObj1, InObj2, OutObj2, EventClear, EventReady\} \\
T &= \{WaitID, RunID, StartID, EndID\} \\
&\cup \{WaitSch, RunSch, StartSch, EndSch\} \\
A &= \{IDIn \to WaitID, WaitID \to IDWait, IDWait \to RunID, RunID \to IDOut\} \\
&\cup \{SchIn \to WaitSch, WaitSch \to SchWait, SchWait \to RunSch, RunSch \to SchOut\} \\
N &= \{(x \to y, (x, y)) | x \to y \in A\} \\
C &= \{(IDIn, ST), (IDWait, STT)\} \\
G &= \{(t, true) | t \in T\} \\
E &= \{(IDIn \to WaitID, st), (WaitID \to IDWait, (st, num))\} \\
I &= \{\} \\
SN &= \{\} \\
SA &= \{\} \\
PN &= \{\} \\
PT &= \{\} \\
PA &= \{\} \\
FS &= \{\{IDIn\}, \{IDOut\}, \{SchIn\}, \{SchOut\}, \{ObjMax\}, \{EventClear\}, \{EventReady\}\} \\
FT &= \{(p, global) | p \in FS\} \\
PP &= \{(s, 1) | s \in S\}
\end{aligned}
$$

4.4.4 イベント管理ネット

イベント管理ネットはイベントを管理するもので, 図 4.7 に示すネットはイベント毎に用意する.

これは同一のフィールドのオブジェクトで一つの用意する.

$$\mathcal{C}_E(Events) = (S, SN, SA, PN, PT, PA, FS, FT, PP)$$

$$
\begin{aligned}
S &= \{(\Sigma, P, T, A, N, C, G, E, I)\} \\
\Sigma &= \{Ev\} \\
P &= \{EventClear, EventReady\}
\end{aligned}
$$

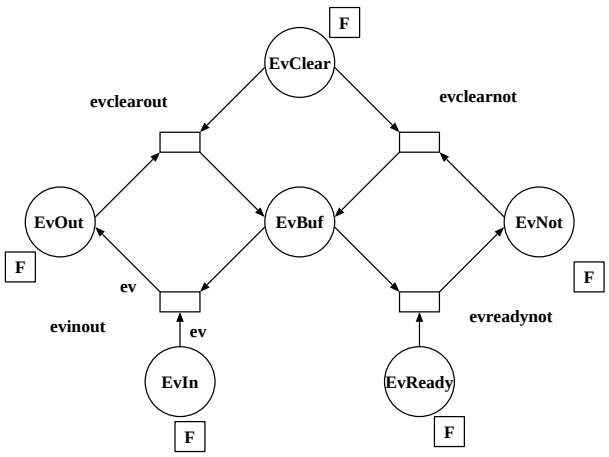

図 4.7: **イベント管理ネット**

$$
\begin{aligned}
& \cup \quad \bigcup_{e \in Events} \{eClear, eReady, eOut, eBuf, eNot, eIn\} \\
T \;=\; & \{allclear, allready\} \\
& \cup \quad \bigcup_{e \in Events} \{einout, ereadynot, eclearout, eclearnot\} \\
A \;=\; & \{EventClear \rightarrow allclear, EventReady \rightarrow allready\} \\
& \cup \quad \{allclear \rightarrow eClear | e \in Events\} \\
& \cup \quad \{allready \rightarrow eReady | e \in Events\} \\
& \cup \quad \{eClear \rightarrow eclearout | e \in Events\} \\
& \cup \quad \{eClear \rightarrow eclearnot | e \in Events\} \\
& \cup \quad \{eReady \rightarrow ereadynot | e \in Events\} \\
& \cup \quad \{eBuf \rightarrow ereadynot | e \in Events\} \\
& \cup \quad \{eIn \rightarrow einoutt | e \in Events\} \\
& \cup \quad \{eBuf \rightarrow einoutt | e \in Events\} \\
& \cup \quad \{einout \rightarrow eOut | e \in Events\} \\
& \cup \quad \{ereadynot \rightarrow eNot | e \in Events\} \\
& \cup \quad \{eOut \rightarrow eclearout | e \in Events\} \\
& \cup \quad \{eNot \rightarrow eclearnot | e \in Events\} \\
& \cup \quad \{eclearout \rightarrow eBuf | e \in Events\}
\end{aligned}
$$

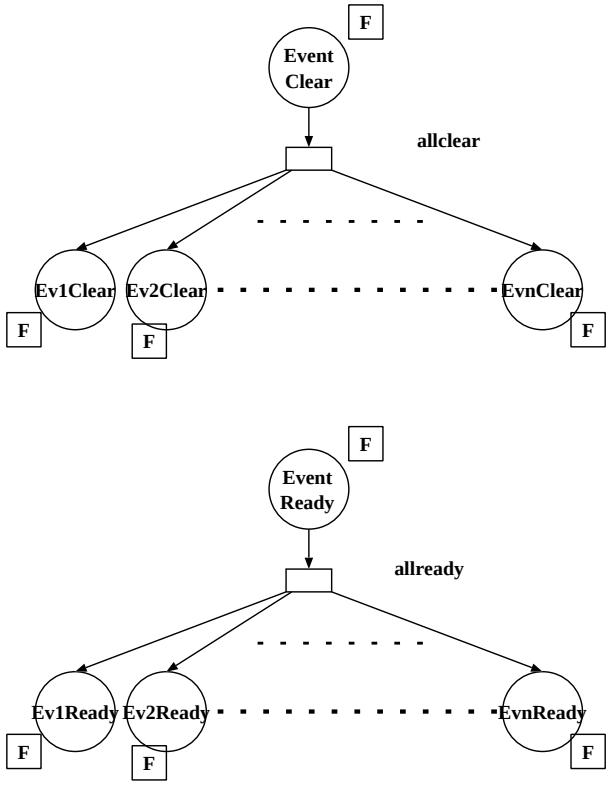

図 4.8: **イベントを複数のオブジェクトへ渡す**

$$
\begin{aligned}
& \cup \quad \{eclearnot \rightarrow eBuf | e \in Events\} \\
N &= \{(x \rightarrow y, (x, y)) | x \rightarrow y \in A\} \\
C &= \{(e, Ev) | e \in P\} \\
G &= \{(t, true) | t \in T\} \\
E &= \{(a, ev) | a \in A\} \\
I &= \{\} \\
SN &= \{\} \\
SA &= \{\} \\
PN &= \{\} \\
PT &= \{\} \\
PA &= \{\} \\
FS &= \{\{p\} | p \in (P - \{eBuf | e \in Events\})\}
\end{aligned}
$$

$$FT = \{(p, global)|p \in FS\}$$
$$PP = \{(s, 1)|s \in S\}$$

4.4.5 拡張したオブジェクトネット (内部オブジェクトの定義)

内部オブジェクトについて考える．単一の内部オブジェクトと並行動作する内部オブジェクトがある．

単一の内部オブジェクトは次に示すように変換できる．

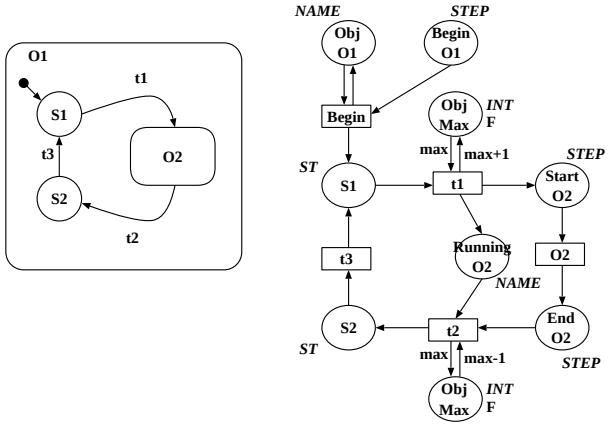

図 4.9: **単一の内部オブジェクト**

並行動作する内部オブジェクトのときは次のように変換できる．

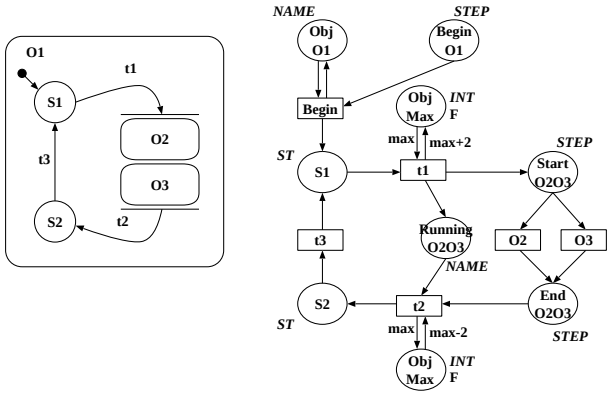

図 4.10: **並行動作のある内部オブジェクト**

第 5 章

テスト

本章では, 提案した手法により変換したカラーペトリネットを利用して, どのような解析が行なえるかについて述べる.

5.1 可達グラフについて

可達グラフとは, カラーペトリネットにおいて連続してトランジションを発火させた場合のマーキングの変化を, 一マーキングを一ノードとして有向グラフにしたものを言う.

例えば, あるマーキング M1 から, 何らかのトランジションが発火して別のマーキング M2 へ変化する, という場合は M1 のノードから M2 のノードへの有向のアークが伸びることになる.

ペトリネットにおいては, 複数のトランジションが発火可能であるマーキングも存在する. そのような場合には, その各々のトランジションが発火した場合についてのマーキングのノードをつくり, それぞれにアークが伸びる.

さらにカラーペトリネットにおいては, トークンがカラーの性質を持つため, 同じトランジションの発火でもそのトランジションを発火させるトークンが違うことがある (アーク式に記述されたカラー変数へのバインドが異なることがある). それによって発火後のマーキングも当然違うので, ノードも別となり, アークも別々に伸びる.

図に簡単なカラーペトリネットとその可達グラフを示す. この可達グラフが算出できれば, 初期マーキングから到達可能なすべてのマーキングが得られる.

5.1.1 デッドロック

可達グラフにおける「活性」の判定により, デッドノード, つまりそのマーキングにおいて, それ以上トランジションが発火不能であるというノードを抽出することが可能であり, カラーペトリネットのシステムがどのようなマーキングにおいて停止するかをしることができる.

それらのマーキングがすべて正常に実行が終了したことを示すマーキングであれば問題は無いが, そうでない場合はシステムがデッドロックを起こしていると考えられる.

また, 可達グラフにおいては, そのデッドノードに至る道筋も示されるため, どのようになるとデッドロックが起こるのかも, たどることができる.

5.1.2 状態への可達性

可達グラフによって判定できる性質には「有界性」がある. これは, あるプレースにおかれるトークンの数の最小値, および最大値を求めることというものである.

これを利用して, 状態遷移図における状態の入り口を示すプレースにおかれるトークンの数の「最大値」を求めることにより, その状態への可達性を調べることができる. 最大値が 0 であれば, その状態には到達することは無いし, 1 であれば実行条件によっては到達するということが言える. 例えば, 状態遷移図上に, 動作が正常であれば到達するはずの無い状態を用意することにより, その状態に到達可能でないことで正当性を確かめることもできる.

第 6 章

記述例

6.1 Telephone の記述例

6.1.1 システムの仕様 ObTS モデル

並行性を含まない記述例として ObTS で状態が 4 つで初期遷移、自己遷移、相互遷移を含むシステムとしてモデルを記述する。ObTS の記述

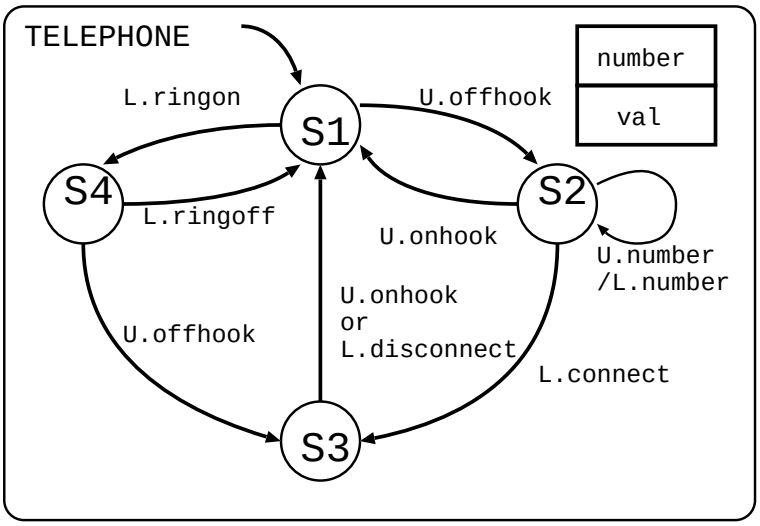

図 6.1: ObTS での記述

6.1.2 ObCL コード

```
event ATTRIBUTED
  inherit GENERIC_EVENT
  attribute val:Int
end
field User
  event offhook,onhook:GENERIC_EVENT
  event number:ATTRIBUTED
end
field Line
  event connect,disconnect,ringon,ringoff:GENERIC_EVENT
  event number:ATTRIBUTED
end
fieid Display
  event number:ATTRIBUTED
end

class TEL
  field U:User; L:Line; D:Display
  state s1,s2,s3,s4
  transition
  start is
    source init
    destination s1
  end
  t1a is
    source s1
    input U.offhook
    destination s2
  end
  t2a is
    source s2
```

```
   input U.number
   do L.number.val:=U.number.val;
      D.number.val:=U.number.val;
   destination s2
   output L.number,D.number
end
t2b is
   source s2
   input U.onhook
   destination s1
end
t2c is
   source s2
   input U.connect
   destination s3
end
t3 is
   source s3
   input U.onhook or L.disconnect
   destination s1
end
t1b is
   source s1
   input L.ringon
   destination s4
end
t4a is
   source s4
   input L.ringoff
   destination s1
end
t4b is
   source s4
```

```
    input U.offhook
    destination s3
  end
end

system EXAMPLE
  object tel:TEL
end
```

6.1.3 変換した CPN:オブジェクトネット

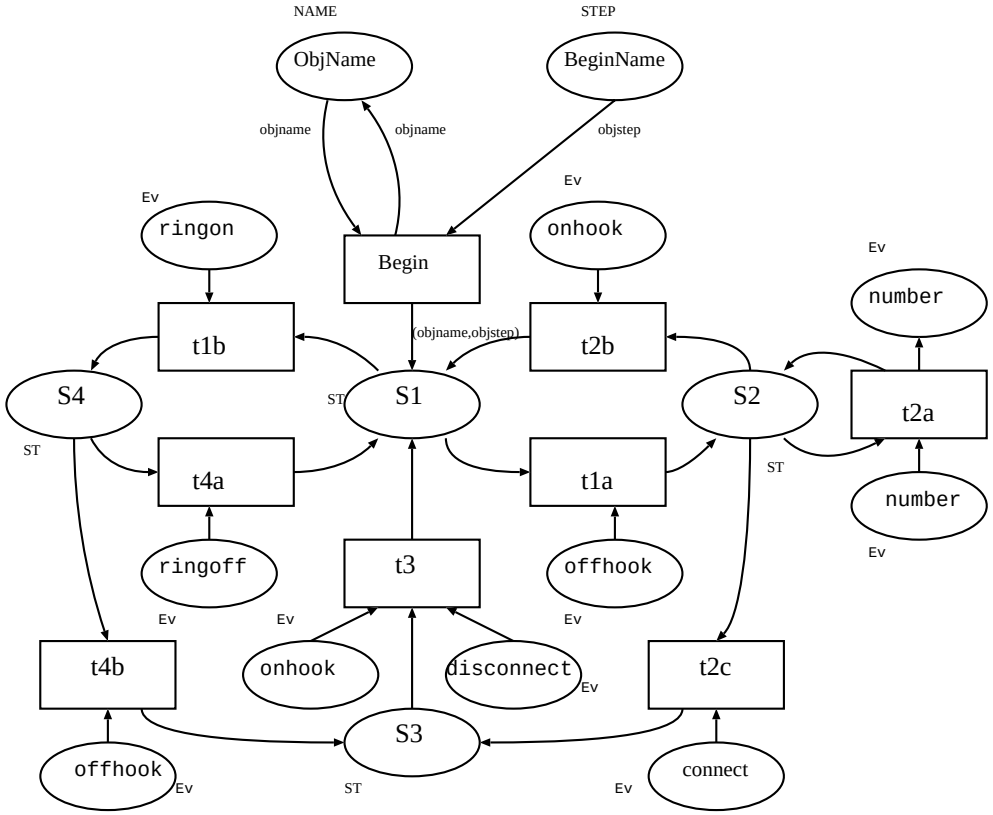

図 6.2: **オブジェクトネット**

6.1.4 遷移規則ネット

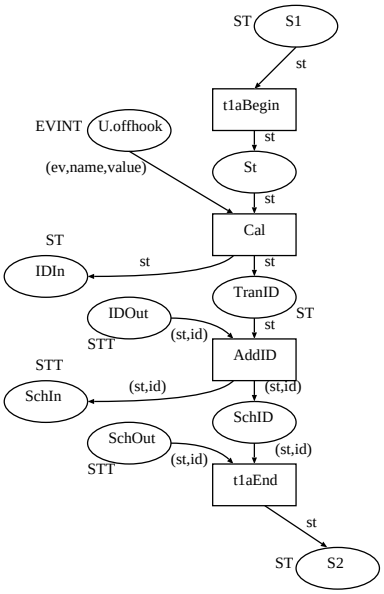

図 6.3: **遷移規則ネット** t1a

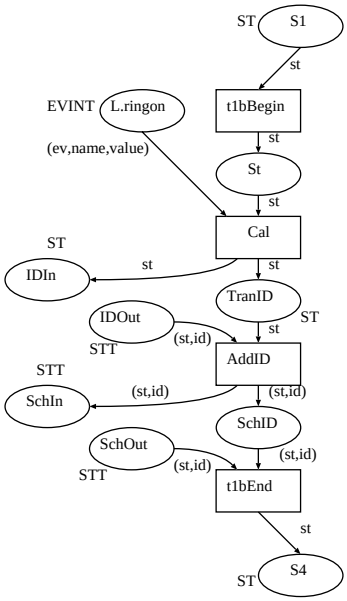

図 6.4: **遷移規則ネット** t1b

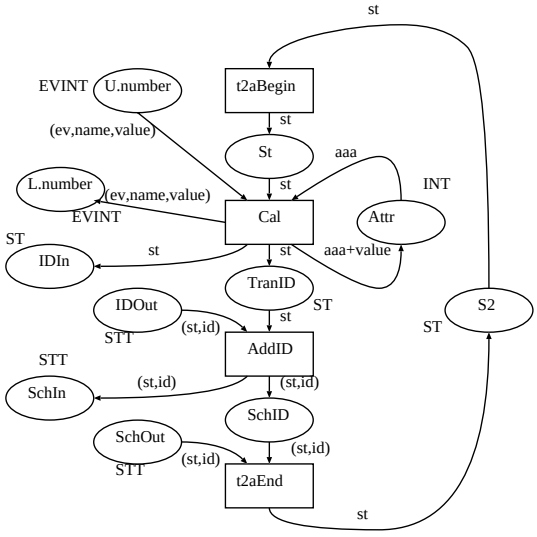

図 6.5: **遷移規則ネット** t2a

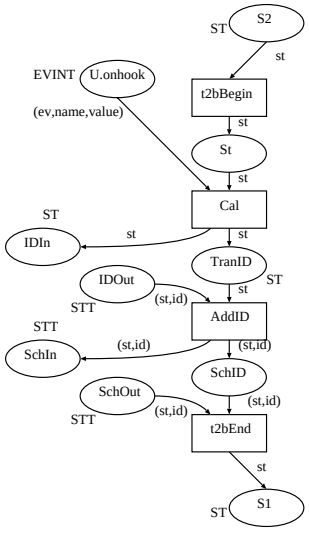

図 6.6: **遷移規則ネット** t2b

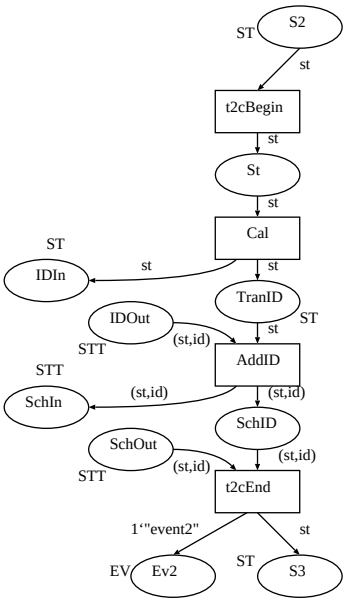

図 6.7: **遷移規則ネット** t2c

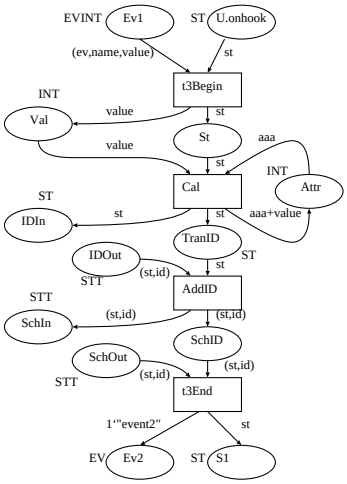

図 6.8: **遷移規則ネット** t3or1

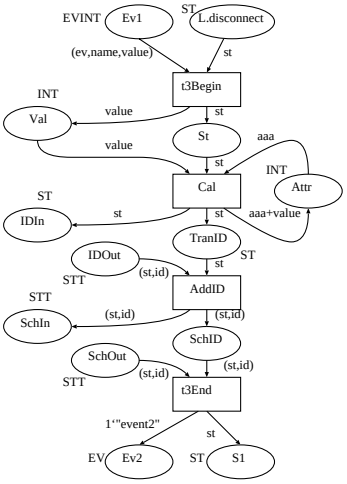

図 6.9: **遷移規則ネット** t3or2

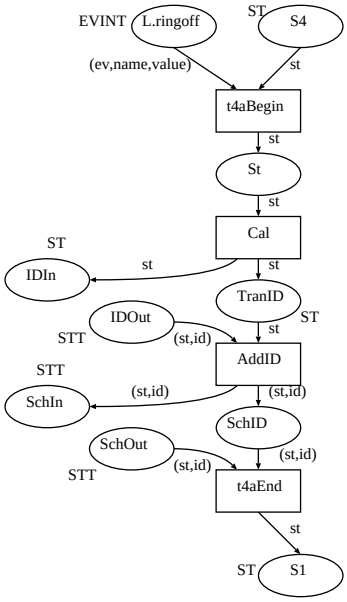

図 6.10: **遷移規則ネット** t4a

6.1.5　イベントネット

イベントネットについては, 各イベントに対して図 4.7 で示したイベント管理ネットを作成する. さらに, 図 4.8 で示したイベントの複製と配信を行なうネットを作成すればよい.

第 7 章

考察、評価

7.1 考察

ObTS で記述した仕様を, 今回定義した基本変換規則と拡張変換規則によりオブジェクトネット, 遷移規則ネット, スケジューラネット, イベント管理ネットの各ネットを生成する. 基本変換規則を用いて, ObTS の一つのオブジェクトを CPN の一つのオブジェクトネットへ変換し, 状態の遷移に関しては, 各遷移ごとに遷移規則ネットへと変換する. このとき ObTS の状態遷移図の状態の数と, 変換したオブジェクトネットの CPN で状態を示すプレースの数は, 同じであり, ここまでの変換では図は同じ形となっている. これは階層化カラーペトリネットの概念, 融合プレースであるフュージョンやサブネットであるページを使うことにより, 実現している. CPN では, 更に初期遷移を表す部分とトランジション, イベントを表すプレースを付け加えるため, 図は複雑になっていく. しかし, それでもペトリネットのように階層化しない方法であれば, 遷移規則ネット同じネットに記述しなければならないため, 図の見た目は非常に複雑となってしまう.

本研究で定義した変換規則で, 非常に難しかったことは ObTS での動的セマンティクスを満たすための拡張変換規則である. ObTS で状態が遷移する際に, 一オブジェクト一遷移することを示すステップをスケジューラネットで実現している. また, ObTS のイベントが同じフィールドのオブジェクトに対してブロードキャスト通信を行なうことを実現するために, イベント管理ネットにおいてオブジェクトと見立てているサブネットの数だけトークンを複製して配信している. ObTS から CPN への変換は非常に単純に行なうことができると思っていたが, 実際に動作セマンティクスを満たすようなネットの構造や機構を考えることは困難であった.

この変換によって気づいた点として, ObTS では拡張変換規則で実現している事柄、例

えばイベントの遷移がステップ単位で実行することや, イベントがフィールドという領域にあるオブジェクトすべてにブロードキャストすることで通信を行なうこと, こういったObTS の特徴となっている事柄は ObTS の仕様であり, ObTS を使う際には気が付かない点でもある. しかし, システムをモデル化する際に, ObTS には ObTS 固有の特徴があることを強く意識して記述を行なわなければ, システムの動作を正確にモデル化していくことができない、と感じました。
またこの気づいた点に関して、仮に ObTS を拡張することを考えたとき, イベントを用いた通信や, イベントの遷移方法に制約を与えることでより複雑なシステムの記述が可能となるかも知れない.

7.2 評価

ObTS から CPN への変換は, 並行動作を含むモデルについても可能であるが, テスト方法としては, ObTS の動作を忠実に CPN で行なうような変換規則を定義しているため CPN でも, ObML でのシミュレーションと同様で, 動作確認による実行可能という程度しか, 現在は確認できていない. しかし, 変換規則について計算機支援環境を実現することで, 適用例を多く試すことができるようになり, 様々な性質についても調べることが可能になると考えられる.

現段階での評価は優れているとは言えないが, 今後の課題にもあげる変換の正当性の証明や, 大規模な適用例を行なうことで非常に優れたツールとなる.

7.3 今後の課題

ObTS へのフィードバック

ここで ObTS からカラーペトリネットへの変換を定義したが, 誤りなど改善すべきことが分かった際に, ObTS のどこを直せば良いか示す方法や, さらに, CPN から ObTS への変換の実現することでツールとして充実が望まれる.

例題への適用

今回の例題は記述例が小さいため, さらに大きなシステムに対する適用を行なう.

記述の正当性の証明

今回の方法では, ObTS **側から** CPN **側への変換における正当性について**, **何も言っていない**. **ゆえに変換が正当であることを証明する必要がある**.

第 8 章

おわりに

本研究では, 仕様記述モデル ObTS を用いてシステムをモデル化した仕様を, カラーペトリネットへと変換する規則を定義した. また、例題に定義した変換規則を適用することでカラーペトリネットへの変換を行ない, 動作確認のテストをした. 例題を用いた試行錯誤により, ObTS 側での動作を忠実に再現できるように変換規則を定義し, 動作セマンティクスを満すカラーペトリネットへの変換を行なった.

このテスト方法でカラーペトリネットに変換でき, そのカラーペトリネットによって動作確認等のいくつかのテストができたことにより, 提案した手法の有効性を示した.

謝辞

本研究を進めるにあたり熱心に御指導いただいた片山卓也教授, 伊藤恵助手, 渡辺晴美リサーチアソシエイトに深く感謝します. また, 本論文をまとめるにあたって様々な面で御協力いただいたソフトウエア基礎講座の皆様に心より感謝します.

参考文献

[1] 伊藤 恵, 片山 卓也: オブジェクト指向方法論のための動的モデル ObTS, コンピュータソフトウェア vol.14 No.2, pp.22-37, March 1997.

[2] 久保秋 真, 伊藤 恵, 片山 卓也: ObTS モデルに基づくオブジェクト指向仕様記述言語と支援環境, 日本ソフトウエア科学会第 14 大会論文集, D11-4, pp.589-592 (1997).

[3] 渡辺 晴美, 徳岡 宏樹, Wu Wenxin, 佐伯 元司, "カラーペトリネットによるオブジェクト指向ソフトウエアのテストと解析方法", 電子情報通信学会和文誌 D-I, Vol.J82-D-I, No.3, 1999, 3

[4] K. Jensen, "COLOURED PETRI NETS – Basic Concepts, Analysis Methods and Practical Use – Volume 1-3", Springer-Verlag, 1992, 1994, 1997.

[5] Boris Beizer, "Software Testing Techniques (Second Edition)", Van Nostrand Reinhold, 1990

[6] 小野 間彰, 山浦 恒央 訳, "ソフトウエアテスト技法", 日経 BP 出版センター, 1994

[7] 青山 幹雄, 内平 直志, 平石 邦彦, "ペトリネットの理論と実践", 朝倉書店 , 1995

[8] J. ランボー, M. ブラハ, W. プレメラニ, F. エディ, W. ローレンセン, 羽生田栄一: オブジェクト指向方法論 OMT モデル化と設計, トッパン (1992)

[9] 渡辺 晴美, 伊藤 恵, 西田 雅彦, 片山 卓也, "オブジェクト指向開発法における分析・設計仕様間のテスト方法",oo1999