修 士 論 文

形式手法を用いた手順書の解析

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

穐山 周平

2013 年 3 月

修 士 論 文

形式手法を用いた手順書の解析

指導教官　二木 厚吉 教授

審査委員主査　二木 厚吉 教授
審査委員　青木 利晃 准教授
審査委員　緒方 和博 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1010002 穐山 周平

提出年月: 2013 年 2 月

概 要

医療現場では, 様々な事故が発生している。それらの事故はヒューマンエラーが起因することが多い。事故を防ぐには元となるヒューマンエラーをなくすことが考えられるが, 全てのヒューマンエラーをなくすことは現実的には難しいとされる。そのために業務を解析し、事故の起因となる重大なヒューマンエラーの把握することが必要となるが、解析には業務に関する知識や経験が求められる。そこで本研究では, 形式手法を用いた手順書の解析手法を提案する。業務の内容が明示的に記述された手順書から形式的なモデルを作成して推論を行うことで, 実際の業務が行われる前に知識や経験に基づかない解析を行う.

目 次

図 目 次

第1章　はじめに

1.1　背景

医療現場では, 患者に被害を与えるといった事故が発生している。それらの事故は業務を行う上で必要な作業の欠落, 不必要な作業をしてしまうといったヒューマンエラーが起因することが多い。事故を防ぐには元となるヒューマンエラーをなくすことが考えられるが, 全てのヒューマンエラーをなくすことは現実的には難しいとされる。そのため事故に結びつくような重大なヒューマンエラーに対して, 1つずつ対処していくことが求められる。つまり事故を防ぐにはヒューマンエラーが事故の起因となるか業務の解析し, 把握することが必要となる。実際の医療現場ではRCA (Root Cause Analysis) やFMEA (Failure Mode and Effects Analysis) が知られている。けれどもRCAは実際に起きた事故を元に解析を行うため事故前の解析が難しい。またFMEAは業務に関する知識や経験が求められる。

1.2　目的

本研究では, 形式手法を用いた手順書の解析手法を提案する。業務の内容が明示的に記述された手順書から形式的なモデルを作成して推論を行うことで, 実際の業務が行われる前に知識や経験に基づかない解析を行う。

1.3　本論文の構成

- 2章
  事故やヒューマンエラーの定義. ヒューマンエラーの対策で用いられる解析手法などの説明を行う.
- 3章
  本研究で用いるCafeOBJとモデル化手法であるOTS/CafeOBJ法の説明を行う.
- 4章
  検体検査の手順書を元に手順書モデルを作成する.

- 5 章
  手順書モデルが検体検査の目的を達成できるか検証を行う.
- 6 章
  手順書モデルにヒューマンエラー含ませたモデルを作成する.
- 7 章
  手順書モデルにヒューマンエラーを含ませたモデルが手順書の目的を達成できるか検証を行う.
- 8 章
  研究を行ってきた上でのまとめ, 今後の課題を記述する.

第2章　手順書とヒューマンエラー

2.1　手順書

手順書とは業務の目的を達成するために作業の要点や順番を記述したものである. 業務とは何らかの目的を達成するために必要な作業の順番である. 手順書の目的は作る側によって変わってくる. 事故が起きないよう安全に業務を行うためのものや, 効率的な作業を行うもの様々な目的がある. 本研究では手順書から人の行う作業を抜き出し, 手順書が業務の目的を達成できるものか, 人の行う作業が正しく行われなかった場合（ヒューマンエラー）, 何が起こるのかなどを考えていきたい.

2.2　作業

作業は複数の人の動作を組み合わせたアトミックな単位である. 全ての人の動作を正確に手順書に記述を行おうとすると量が膨大になってしまう. 手順書を見る人によって作業をどこまで記述するか変わってくる. 例えば医療現場での採血作業を見てみる. 患者に針を挿入する場所の消毒, 採血針を挿入, 抜針, 止血など様々な人の動作が採血作業に入ってくる.

2.3　事故

本研究では, 目的を達成できない状況を事故として定義する. 目的を達成てきない状況とは, 手順書を用いて作業を行っていく上で, 手順書の目的を満たすための望ましい状況がある. そのような状況から外れたときを目的を達成できない状況とする. 例えば, 採血手順書では, 採血作業を終えたときに検査したい患者本人から採血がされる. といったことが採血手順書で目的を満たすための望ましい状況の 1 つである. しかし誤った患者から採血を行ってしまった. 望ましい状況から外れたとき目的を達成できない状況となる. 事故は様々な定義がなされている. 例えば「事故は, そのシステムが現在あるいは将来生み出すはずの成果に対する信頼を失わせてシステムにダメージを与える事象」[3] さらに被害を被る対象によっては違った呼び方もされる. 医療分野における事故で患者に被害が及んだものを有害事象と呼ぶ. 有害事象とは「患者本来の基礎的条件によるものではなく, 医療的処置によって生じた傷害」[?] 基礎的条件とは患者の体調や疾患などを指し, それらが被害の

原因にならないということを意味する. 以上のように一般的に事故と言われるものは人や物に被害を与えてしまったような状況を事故と指すことが多い. また被害が及んでいないが, 事故に至る可能性がある状況はインシデントといった呼ばれ方がされる. インシデントや一般的な事故も手順書の目的を満たすための阻害となる状況であると考え, 二つを合わせたような状況を本研究では事故と定義した.

2.4 ヒューマンエラー

本研究ではヒューマンエラーを手順書に書かれた作業と異なる作業をすることをヒューマンエラーと定義する. このヒューマンエラーが事故が起きる原因の１つとしてあげられる. ヒューマンエラーは様々な定義がなされている.「すべきことが決まっているとき, すべきことをしないあるいはすべきでないことをする」[4] すべきこととは規則や法律などで明示されたものから常識で暗黙的に決まっているものから外れた行動を指す. またヒューマンエラーはヒューマンエラーを起こしたときの心理的な状態や作業の結果によって場合わけもされている.

2.4.1 心理的なヒューマンエラー

Norman によるヒューマンエラーの場合わけ

- ミステイク（意図した行動が間違っていたか）
- スリップ (行動が意図したようにいかなかったか)

2.4.2 作業の結果から見たヒューマンエラー

Swain によるヒューマンエラーの場合わけ [7]

- オミッションエラー（必要な作業を行わなかった）
- コミッションエラー（作業を行っているが違うことをした）
  - 作業を行っているが行うタイミングが早すぎるもしくは, 遅すぎる.
  - 本来やるべきではない作業を行う.
  - 作業遂行の順序が違う

2.5　事故を防ぐ

事故を防ぐためには原因であるヒューマンエラー自体を防ぐことが１つの解決方法となる. しかし全てのヒューマンエラーを防ぐのは現実的に厳しいとされる. またヒューマンエラーが必ず事故に繋がるわけではない. 例えば, 患者の本人確認という作業を行わなかっただけでは事故は起きない. 事故を防ぐためにはどのようなヒューマンエラーが事故に結びつくか関係性を見つける必要があると考える. 事故に結びつくヒューマンエラーを見つけることで事故を防ぐ手段を考えることができる.

2.6　事故を防ぐための解析手法

2.6.1　RCA(Root Cause Analysis)

業務を解析するために, 実際に起きた事故を元に事故の原因となったエラーや外的要因を調べる.

2.6.2　FMEA(Failure-mode and effects analysis)

「システムを構成するアイテムのひとつひとつについて, どのような故障がどのくらいの確率で起こる可能性があり, 故障が起こるとシステムにどんな悪影響が及ぶかをアイテムのほうから検討する手法 [2] 信頼性工学の手法の１つである. 医療分野でも応用されている.

2.6.3　従来の解析手法に求められるもの

RCA は実際に起きた事故を元に解析を行うため事故前の解析が難しい. また FMEA は業務に関する知識や経験が求められる.

2.7　形式手法を用いた手順書解析

本研究では形式手法を用いて形式的にモデルを作りその上で議論することで, 知識や経験に基づかず数学的に議論を行う方法を考える. 本研究では, 以下の方法をとる.

1. OTS/CafeOBJ を用いて状態遷移システムとして手順書のモデル（手順書モデル）を作成.
2. 手順書モデルが業務の目的を達成できるか検証.
3. 手順書モデルにヒューマンエラーを組み込みだモデルを作成.

4. ヒューマンエラーを含んだモデルが業務の目的を達成できる手順書か検証.

第3章　CafeOBJ

3.1　CafeOBJ とは

CafeOBJ は代数に基づく仕様記述言語である. 仕様を構成する等式を書換規則として実行することができる. 等式の実行は仕様の意味を規定する等式論理に忠実であるので CafeOBJ システムを用いた対話的検証が可能である.[8] [9] [5]

3.2　CafeOBJ の構文

実際の例を用いつつ基本的な構文の説明をする.

```
mod! HUM{
 [Man Woman < Hum ]
 op hikaku : Hum Hum -> Bool
 var H : Hum
 var M : Man
 var W : Woman
 eq hikaku(H, H)= true .
 eq hikaku(M, W) = false .
}
```

モジュール
CafeOBJ で記述された仕様はモジュール単位で記述される.

mod モジュール名 {
モジュール内
}

で構成されている.
ソート
ある集合を表す名前となる.
[···] でソートの定義を行うことができる.

```
 [Man Woman < Hum ]
```

例では Man， Woman，Humu のソートが宣言されている. また定義内における < はソート間の半順序関係を宣言している.

演算子

op function name : arity -> coarity .

op で演算子を宣言して関数名を記述する. 関数の引き値にとるソートの組をアリティ, 返し値となるソートをコアリティと呼ぶ. またソートの組（引数）-> ソート (返し値) の対をランクと呼ぶ. 例では演算子名 hikaku が ソート Hum の組 を取り Bool を返す関数を定義している.

変数

var variable name : sort .

var を用いて変数を宣言する. 変数名とソートを用いて定義する. 指定されたソートの項が変数に入る.

例では H,W,M の変数が定義されている.H はソート Hum の項, W は ソート Woman の項, M はソート Men の項が入る.

等式

eq LHS = RHS .

eq で等式の宣言を行い LHS と RHS には同ソートの項が入る.LHS と RHS の項の等価を定義する.

例では 2 引数とも同じ項, 変数 H を引き値としてとれば true を返す. ソート Man の項とソート Woman の項を取れば false を返すように宣言している.

シグネチャ

ソートの集合 S, ソートの順序関係 $\leq$,S^*×S で分類された演算子の集合 Σ の組 (S,$\leq$, Σ) をシグネチャと呼ぶ.

例では

S は {Hum,Woman,Man}

$\leq$ は {(Man,Man),(Man,Hum),(Woman,Hum),(Hum,Hum)}

Σ は $\Sigma_{HumHum,Hum} = \{\text{hikaku}\}$， それ以外の (w,s) $\in S^*$×S に対しては $\Sigma_{w,s} = \phi$ となる.

項

項は演算子と変数を合わせたものとなる. シグネチャと S ごとに分類された変数の集合 X に対して S ごとに分類された項の集合 T(Σ,X) は 3 つの条件から定義される.

- ソート s で定義された変数はソート s の項となる.
  $X_S \subseteq T_S$
- 演算子の返し値はコアリティのソートの項となる.
  $t_i \in T_{Si}$(i $\in$ 1,....,n),f $\in$ $\Sigma_{S1,S2,...,Sn,s}$ ならば f(t_1,t_2,...t_n) $\in T_S$
- s$\leq$s' ならば $T_s \subseteq T_{s'}$

3.3 OTS/CafeOBJ 法

CafeOBJ を用いて観測遷移機械 (OTS) を実現するための手法である. また数学モデルの一つである観測遷移機械は CafeOBJ 上で状態遷移システムを実現するための記述法である. 任意の状態を包含する状態空間 U を仮定する. 要素 u は状態となる. 観測遷移機械 S は（O,I,T）で定義される.

- **O**：**観測演算の集合**
  o ∈ O は状態空間 U から観測値である任意のデータ型 D を返す関数である.
  o : U → D
  外部から U を観測する.

- **I**：**初期状態の集合**
  I ⊆ U となる.

- **T**：**条件付遷移規則の集合**
  t ∈ T は状態から状態を返す関数となる.
  t により状態が変化する t(u) は u の t に関する事後状態という. 状態が変化するためには t に付随する効力条件による. 効力条件は状態から Bool 型を返す {true, false}
  c : U → B
  c が true を返す場合は遷移関数 t により状態が変化する.c が false を返す場合は状態は変化しない (u = t(u)) .

状態の等価性
u_1,u_2 ∈ U において
$(u_1 =_s u_2) = (\forall_o \in \mathrm{O}.\ \mathrm{o}(u_1) = \mathrm{o}(u_2))$ となる.
観測遷移機械 S の実行
観測遷移機械 S の実行は, 初期状態からはじまる. 遷移規則を非決定的に選択することで得られる状態の無限列 u_0,u_1,u_2... である. 状態の無限列は以下を満たす.

CafeOBJ 上での OTS 記述 OTS を CafeOBJ で記述するための構文の説明を行う.
状態空間 U

- **開始性**
  u_0 ∈ I となる.

- **連続性**
  i ∈ {0,1,2...} に対して, u_{i+1} $=_s$t(u_i) を満たす t ∈ T が存在する.

- **公平性**
  各 t ∈ T に対して, u_{i+1} $=_s$t(u_i) を満たす　i ∈ {0,1,2...} が無限に存在する.

S の実行であらわれる状態 u は S で到達可能であるという.

第4章　手順書モデルの作成

本章では実際の検体検査手順書を用いて手順書モデルの作成を行っていく. 本研究では実際の事故の事例から, 作業で用いられるものの変化から事故や手順書の目的を達成の状況を判別できると考えた. そこで作業で用いられるものを状態, 作業で用いられるものの変化を作業として状態遷移システムを用いたモデル化を行う. 状態遷移システムを実現するために,CafeOBJ を用いて必要となるデータ型の定義,OTS の記述を行っていった.

4.1　実際に起こった事故例

まず実際にあった事例を元にどのような状況が事故至った状況なのかをみてみる.

> 部屋別に並べておいたスピッツ中から A 氏のスピッツを取り B 氏に採血ですというと直ぐにはいと言って手を出したためそのまま採血をした. 本来取るべき患者は隣のベッドの人であった. 同室の術後の患者 2 人が採血があり B 氏の採血を不思議に思わなかった.TEL がかかってくるまで全く気づかなかった.[1]

この事例では,A 氏のスピッツに A 氏の血が入っていることが望ましい状況なのに B 氏の血が入った A 氏のスピッツが作業によってできてしまう. このスピッツから事故が起きた状況と判別することができる. 事故の状態を招いた作業の原因としては, Aのための作業に患者 A のスピッツと患者 B を用いて作業を行ってしまった.

> 血糖とヘモグロビン A1c の検査中に再検チェックに引っかかっていた為データの確認をした所, 前回値とかなり違った値だったので, 再検をしようとしたら受付にてその患者の採血容器がなくなってしまったとの事であった. ラベルを再発行して, 当該患者より採血し検査した所, 前回値とよく似たデータであった. 前回値と大きく外れた検体は（既に検査済みで再検チェックとなった検体）は, 他の患者の採血容器に紛れて他の患者より採血されたものと思われた.[1]

血糖とヘモグロビン A1c の検査を受けた患者をA, 他の患者をBとする. この文章の見方を少し変えると患者 B のための作業（採血する）にも関わらず患者 A のチューブと患者 B を用いて作業を行ってしまった. そして, 患者 A のための作業（患者 A の健康状態を見る）で患者 B の血が入ったチューブを用いて検査を行った結果, 事故として表に出てきたと考えることができる. この例も誤ったものを用いて作業を行っていってしまったことが事故の状況を生み出してしまったと考えられる.

4.1.1　モデル化の方針

実際の事故例から事故や業務が目的通りに終えられたかを作業で用いられるものの変化から判別することができるのではないかと考えた. そこで作業で用いられるものの集合を状態、その作業要素を変化させる作業を遷移として状態遷移システムとして手順書モデル作成する. つまり CafeOBJ 上で状態遷移システムを実現するため OTS で定義された観測値を作業要素、遷移演算を作業として記述することとなる.

4.1.2　作業要素

ある患者のための作業を行う上で用いるものを作業要素とする. 例えば採血作業を行うためには, チューブや採血を行う患者が必要になる.

4.2　モデル化を行う作業と作業要素の抜き出し

4.2.1　検体検査手順書から作業の書き出し

本研究では検体検査の一部である検体を採取する部分の実際の手順書を用いて研究を行っていく. 検体検査とは患者から血液や尿などの検体を採取し検査を行うことである. 検査によって検体の中に含まれる成分などを分析して病気などの診断を行っていくことができる. 以下は実際の手順書を元に人が行う動作を書き出してきたものである.

1. 医者がオーダを出す.
2. 医者が患者を受付に移動させる.
3. 患者が受付に診察券を渡す.
4. 受付が診察券を読み込ませる.
5. 受付が診察券を返す.
6. 受付が受付票を患者に渡す.
7. 受付が受付票控えをナースに渡す.
8. ナースがラベルを取る.
9. ナースがチューブを取る.
10. ナースが受付票控えを読み込ませることでシステムが患者を処置室に移動させる.
11. ナースがチューブにラベルを貼る.

12. ナースが受付票控えを読み込みオーダーの内容を呼び出す.

13. ナースがオーダーの患者 ID とチューブの患者 ID を確認する.

14. ナースが患者から受付票を受け取る.

15. ナースがオーダーの患者 ID と受付票の患者 ID を確認する.

16. ナースが患者の名前とオーダの名前を確認する.

17. ナースが患者から採血をする.

この検体検査手順書から書き出した作業の流れは, 医者が患者の診察を行う. 診察を行った上で検体検査が必要だと判断すると, オーダリングシステム（医者の指示）を通して看護師に検体採取を行うことを伝える, 医者が患者に採血室の受付に促す, 患者は採血室の受付にいくと受付に診察券を渡し, 受付票を受付からもらい自身の順番が回ってくるまで待合室で待つ, 自身の順番が来ると採血室の中に案内され採血を行う, といったことが検体採取の大まかな流れとなる.

4.2.2 作業と作業要素の抜き出し

先で述べたとおり, ある患者のための作業要素の集合を状態と捉え, そのものを使って作業を行うことを遷移として捉えることで状態遷移のモデル化を行う. そのために手順書から書き出した作業の列からモデル作成を行うための作業と作業要素を抜き出していく. 抜き出しす方針としては、書き出した作業から、さらに作業要素が変化する作業を抜き出していく. 例えば,11. ナースがチューブにラベルを貼る, はモデル化する作業となる. これはチューブとラベルを用いることでラベルが貼られたチューブとなるので作業要素が変化した作業と捉えることができる. 逆に 2. 患者が受付に診察券を渡す作業は除外される.2 の作業では患者が診察券を用いて作業を行っているが, 診察券や他の作業要素に変化が起きていない. そして抜き出した以下の作業の列を元にモデル化を行う.

1. 医者がオーダーを出す.

2. ~~医者が患者を受付に移動させる.~~

3. ~~患者が受付にカードを渡す.~~

4. 受付がカードを読み込ませる.

5. ~~受付がカードを返す.~~

6. ~~受付が受付票を患者に渡す.~~

7. ~~受付が受付票控えをナースに渡す.~~
8. ~~ナースがラベルを取る.~~
9. ~~ナースがチューブを取る.~~
10. ~~ナースが受付票控えを読み込ませることでシステムが患者を処置室に移動させる.~~
11. ナースがチューブにラベルを貼る.
12. ~~ナースが受付票控えを読み込みオーダーの内容を呼び出す.~~(受付票控えの内容とオーダーの内容は一致するものとする.)
13. ナースがオーダーの患者 ID とチューブの患者 ID を確認する.
14. ~~ナースが患者から受付票を受け取る.~~
15. ナースがオーダーの患者 ID と受付票の患者 ID を確認する.
16. ~~ナースが患者の名前とオーダーの名前を確認する.~~（患者の名前と受付票の名前は一致するものとする.）
17. ナースが患者から採血をする.

4.3 型の定義

作業と作業要素の抜き出しを行ったので CafeOBJ 上でモデル作成に必要となるデータ型の定義を行っていく.

4.3.1 患者ＩＤ

患者を一意に表すためＩＤとなる. 受付票やラベルなどに記述され患者本人のものかなどを表現する.

```
mod* PATIENTID {
 [ PatientId ]
 op _=_ : PatientId PatientId -> Bool {comm}
 var PI : PatientId
 eq (PI = PI) = true .
}
```

`op _=_` : 患者ＩＤの等価性を判断するものである.`{comm}`は,`_=_`の交換法則を満たすことを表す.

4.3.2 検査内容

検査内容を表す. 医者がどのような検査を行うかオーダーに記述することで, その検査内容にあった採血管が用意される.

```
mod! EXAMINATION{
 [ Examination ]
 op _=_ : Examination Examination -> Bool {comm}
 var E : Examination
 eq (E = E) = true .
}
```

4.3.3 血液

患者から取った血液を表す.

```
mod! BLOOD {
  [ None_B Blood < Blood? ]
  pr(PATIENTID)
  ops no-blood : -> None_B
  op blood : PatientId -> Blood
  op get-patientid : Blood? -> PatientId
  op _=_ : Blood? Blood? -> Bool {comm}
  var B : Blood?
  var BE : Blood
  var N : None_B
  var PI : PatientId
  eq get-patientid(blood(PI)) = PI .
  eq (B = B) = true .
  eq (BE = N) = false .
}
```

op no-blood :まだ採血が行われてなく, 血液が存在しない.
op blood : 採血が行われ血液が存在する.
op get-patientid :どの患者の血液か取得する.

4.3.4 チェックマーク

作業要素が持つデータの比較結果を表す. 作業要素オーダーとチューブに張られた患者の名前を比較といった作業を表す.

```
mod! CHECK {
 [ Check ]
 ops no-check check-false check-true : -> Check
 op _=_ : Check Check -> Bool {comm}
 var C : Check
 eq (C = C) = true .
 eq (no-check = check-false) = false .
 eq (no-check = check-true) = false .
 eq (check-false = check-true) = false .
}
```

op no-check：チェックマークが付いてなく確認作業が行われていない.
op check-false：比較を行った結果一致しない.
op check-true：比較を行った結果一致する.

4.3.5 受付票

患者から受け取った診察券を受付が読み込ませると受付票が出力される. それを渡すことで, 待合室にいる採血を行いたい患者を採血室に呼び出すことができる. 今回は受付票を持っている人の名前と受付票に書かれている名前は必ず一致するものとする.

```
mod! RECEIPT {
 [ None_R Receipt < Receipt? ]
 pr(PATIENTID)
 op no-receipt : -> None_R
 op receipt : PatientId -> Receipt
 op get-patientid : Receipt? -> PatientId
 op _=_ : Receipt? Receipt? -> Bool {comm}
 var R : Receipt?
 var N : None_R
 var RE : Receipt
 var PI : PatientId
 eq get-patientid(receipt(PI)) = PI .
 eq (R = R) = true .
 eq (N = RE) = false .
}
```

`[ None_R Receipt < Receipt? ]`:`None_R`は受付票がまだ作れていないソートを表し,`Receipt`

は受付票が作られたソートを表す.Receipt?はNone_RとReceipt二つのソートを含んだソートとなる.
op no-receipt：受付票がまだ作られてなく存在していない.
op receipt：患者のＩＤが記述された受付票が存在する.
eq get-patientid：受付票に記述された患者ＩＤを見る.

4.3.6 オーダー

医者が看護婦に対してどのような検査を行うかオーダリングシステムを用いて指示を出す. このオーダーの情報を元にチューブや受付票などが作られる.

```
mod! ORDER {
 [ None_O Order < Order? ]
 pr(PATIENTID + EXAMINATION + CHECK)
 op no-order : -> None_O
 op order : Examination Check Check -> Order
 op get-examination : Order? -> Examination
 op get-receipt-check : Order? -> Check
 op get-tube-check : Order? -> Check
 op _=_ : Order? Order? -> Bool {comm}
 var O : Order?
 var OE : Order
 var N : None_O
 var PI : PatientId
 var EX : Examination
 vars C1 C2 : Check
 eq get-examination(order(EX, C1, C2)) = EX .
 eq get-receipt-check(order(EX, C1, C2)) = C1 .
 eq get-receipt-check(no-order) = no-check .
 eq get-tube-check(order(EX, C1, C2)) = C2 .
 eq get-tube-check(no-order) = no-check .
 eq (O = O) = true .
 eq (N = OE) = false .
}
```

[None_O Order < Order?]：None_Oは医者からのオーダーがまだ作成されてないソート,Orderは医者からのオーダーの作成が行われたソートを表す.Order?はNone_OとOrder二つを含んだソートとなる.

op no-order：オーダーがまだ作られてなく，存在していない．
op order ：患者の試験内容を表すExaminaion，受付票が患者本人のものか名前を比較した結果Chcek，チューブが患者本人のものか名前を比較した結果Checkの３つで構成される．
op get-examination：オーダーに記述された検査内容を見る．
op get-receipt-check ：受付票が患者のために用意されたものかオーダーの患者ＩＤと受付票の患者ＩＤを比較した結果を見る．比較した結果一致しない場合，違う患者が採血に来てしまったことを表す．
op get-tube-check ：チューブが患者のために用意されたものかオーダーの患者ＩＤと受付票の患者ＩＤを比較した結果を見る．比較した結果一致しない場合，違うチューブ紛れ込んでしまったことを表す．

4.3.7 ラベル

オーダーが作成され，受付票が読み込まれるとラベルが作成される．作成したラベルをチューブに貼ることによってどの患者に対して用意されたチューブなのか，何を検査するためのものか判別できるようになる．

```
mod! LABEL{
 [ None_L Label < Label? ]
 pr(PATIENTID + EXAMINATION)
 op no-label : -> None_L
 op label : PatientId Examination -> Label
 op get-patientid : Label? -> PatientId
 op get-examination : Label? -> Examination
 op _=_ : Label? Label? -> Bool {comm}
 var V : Label?
 var VE : Label
 var N : None_L
 var E : Examination
 var PI : PatientId
 eq get-patientid(label(PI, E)) = PI .
 eq get-examination(label(PI, E)) = E .
 eq (V = V) = true .
 eq (V = VE) = false .
}
```

[None_L Label < Label?]:None_Lはまだラベルが用意されてなく存在していないソー

ト.Label はラベルが用意されたソート.Label は None_L と Label を二つ含んだソートとなる.
op no-label :まだチューブに貼るためのラベルが用意されていないことを表す.
op label :チューブに張るためのラベルが用意されている. ラベルを貼られたチューブが患者本人のものか比較するための PatientId, ラベルに貼られたチューブが何を検査するためのものかみるための Examination の二つで構成される.
op get-patientid :ラベルに記述された患者ＩＤをみる.
op get-examination :ラベルに記述された試験内容をみる.

4.3.8 チューブ

患者の採血を行うためのチューブを表す. 本来は患者が何を検査するかによって使うチューブ形が変わってくる. しかし今回はどのような検査を行おうとチューブの形は変わらないとする.

```
mod! TUBE {
 [ None_T Tube < Tube? ]
 pr(PATIENTID + EXAMINATION + BLOOD + LABEL)
 op no-tube : -> None_T
 op tube : Label? Blood? -> Tube
 op get-blood : Tube? -> Blood?
 op get-label : Tube? -> Label
 op _=_ : Tube? Tube? -> Bool {comm}
 var T : Tube?
 var N : None_T
 var TE : Tube
 var L : Label?
 var EX : Examination
 var B : Blood?
 eq get-blood(tube(L, B)) = B .
 eq get-blood(no-tube) = no-blood .
 eq get-label(tube(L, B)) = L .
 eq (T = T) = true .
 eq (N = TE) = false .
}
```

[None_T Tube < Tube?] : None_T はまだ採血を行う患者のためのチューブが用意されていないソート,Tube は患者のために用意されたチューブが存在することを表すソートとなる.Tube?は None_T と Tube の二つを含んだソートとなる.

`op no-tube`：まだ採血を行う患者のためのチューブが用意されていないことを表す.
`op tube` :チューブが誰のものか, そのチューブを用いて何を検査するか確認するための `Label`, チューブに血液が入っているか, どの患者の血液かみるための `Blood` の二つで構成される.
`op get-label` :チューブに張られたラベルをみる.
`op getblood`：チューブに血が入っているか, 入っていないか見る.
`op get-patientid`：チューブに記述された患者ＩＤを見る.

4.4 ヒューマンエラーを組み込んだ手順書のモデル

4.4.1 観測関数

- オーダー
  患者 P の作業要素となるオーダーを観測する.
  bop p-order : Sys PatientId -> Order

- 受付票
  患者 P の作業要素となる受付票を観測する.
  bop p-receipt : Sys PatientId -> Reciept

- チューブ
  患者 P の作業要素となるチューブを観測する.
  bop p-tube : Sys PatientId -> Tube

- ラベル
  患者 P の作業要素となるラベルを観測する.
  bop p-label : Sys PatientId -> Label?

4.4.2 初期状態

観測遷移システムの初期状態を決める. 採血業務の手順書モデルに置いての初期状態は, 医者が患者に対してのオーダーを出す前の作業要素の状態を指す.

- オーダー
  医者がオーダー作成しておらず, オーダーが存在しない.
  eq p-order(init, PI) = no-order .

- 受付票
  医者がまだオーダーを作成していないので, 患者ＰＩの作業要素受付票も作成されていない.
  eq p-receipt(init, PI) = no-receipt .

- チューブ
  医者がまだオーダーを作成していないので, 患者ＰＩの作業要素チューブも作成されていない.
  eq p-tube(init, PI) = no-tube .

- ラベル
  医者がまだオーダーを作成していないので, 患者ＰＩの作業要素ラベルも作成されていない.
  eq p-label(init, PI) = no-label .

4.4.3 遷移演算：1. 医者がオーダーを出す.

これから採血を行う患者のためにオーダーが作成される.

遷移条件 c-order-create

オーダー作成されていないことが遷移するための条件となる.

```
op c-order-create : Sys PatientId -> Bool .
eq c-order-create(S, PI) = (p-order(S, PI) = no-order)  .
```

遷移 order-create

観測関数 p-order から返される観測値が変化する.

p-order :

患者 PI の作業要素オーダーがまだ存在しない状態である (p-order(S, PI) = no-order) から 患者 PI のためのオーダー作成作業 order-create(S, PI) が行われる. それによりオーダー order(EX, no-check, no-check) が作成される. 任意の試験内容ＥＸが与えられ, チューブと受付票が患者 PI のために用意された作業要素か確認がまだ行われていない no-check となっている.

```
 bop order-create : Sys Examination PatientId -> Sys
ceq p-order(order-create(S, EX, PI1), PI2)
                      = (if PI1 =  PI2
                      then order(EX, no-check, no-check) else p-order(S, PI2) fi)
                      if c-order-create(S, PI1) .

ceq p-receipt(order-create(S, EX, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-order-create(S, PI1) .

ceq p-label(order-create(S, EX, PI1), PI2)
                      = p-label(S, PI2)
```

```
                        if c-order-create(S, PI1) .

ceq p-tube(order-create(S, EX, PI1), PI2)
                        = p-tube(S, PI2)
                        if c-order-create(S, PI1) .

ceq order-create(S, EX, PI) = S
                        if not c-order-create(S, PI) .
```

4.4.4 遷移演算：４．受付がカードを読み込ませる.

オーダーの内容から, チューブ, 受付票, ラベルが作成される.

遷移条件 c-card-read

オーダーが作成されている. 受付票, チューブ, ラベルがまだ作成されていないことが遷移するための条件となる.

```
op c-card-read : Sys PatientId -> Bool .
eq c-card-read(S, PI) = (not (p-order(S, PI) = no-order)) and
                        (p-receipt(S, PI) = no-receipt) and
                        (p-tube(S, PI) = no-tube) and
                        (p-label(S, PI) = no-label) .
```

遷移 card-read

観測関数 p-receipt,p-label,p-tube から得られる観測値が変化する.

p-receipt:

患者 PI の作業要素受付票がまだ存在しない状態 (p-receipt(S, PI) = no-receipt) から患者 PI のためのカード読み込み作業 card-read(S, PI) が行われる. それにより患者 PI の ID が書かれた receipt(PI) が作成される.

p-label :

患者 PI の作業要素ラベルがまだ存在しない状態である (p-label(S, PI) = no-label) から患者 PI のためのカード読み込み作業 card-read(S, PI) が行われる. それにより患者 PI の作業要素ラベル label(PI, get-examination(p-order(S, PI))) が作成される. 患者 ID の PI と患者 PI のために用意されたオーダー p-order(S,PI) の試験内容 get-examination(p-order(S, PI)) が書かれたラベルが用意された状態を表す.

p-tube :

患者 PI の作業要素チューブがまだ存在しない状態である (p-tube(S, PI) = no-tube) から患者 PI のためのカード読み込み作業 card-read(S, PI) が行われる. それにより tube(no-label, no-blood) が作成される. ラベルが貼られてなく no-label と血液が入ってない no-blood のチューブが用意された状態を表す.

```
 bop card-read : Sys PatientId -> Sys
ceq p-order(card-read(S, PI1), PI2)
                      = p-order(S, PI2)
                      if c-card-read(S, PI1) .

ceq p-receipt(card-read(S, PI1), PI2)
                      = (if PI1 =  PI2
                      then receipt(PI2) else p-receipt(S, PI2) fi)
                      if c-card-read(S, PI1) .

ceq p-label(card-read(S, PI1), PI2)
                      = (if PI1 =  PI2
                      then label(PI2, get-examination(p-order(S, PI2)))
                      else p-label(S, PI2) fi)
                      if c-card-read(S, PI1) .

ceq p-tube(card-read(S, PI1), PI2)
                      = (if PI1 =  PI2
                      then tube(no-label, no-blood) else p-tube(S, PI2) fi)
                      if c-card-read(S, PI1) .

ceq card-read(S, PI) = S
                      if not c-card-read(S, PI) .
```

4.4.5 遷移：１１．ナースがチューブにラベルを貼る.

チューブとラベルを用いて、チューブにラベルを貼り付ける.

遷移条件 c-put-label

チューブ, チューブに貼るラベルが作成されている. チューブにラベルが貼られていないことが遷移するための条件となる.

```
op c-put-label : Sys PatientId -> Bool .
eq c-put-label(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                         (get-label(p-tube(S, PI)) = no-label) and
                         (not (p-label(S, PI) = no-label))) .
```

遷移関数 put-label

観測遷移 p-tube,p-label から返される観測値が変化する.

p-tube :

患者 PI の作業要素チューブが存在 (not (p-tube(S, PI) = no-tube)) して, そのチューブにラベルが貼られていない状態 (get-label(p-tube(S, PI)) = no-label) から患者 PI のためのラベルを貼る作業 put-label(S, PI) が行われる. それによりラベルが貼られたチューブ tube(p-label(S, PI), B) に遷移する. 患者 PI の作業要素ラベル p-label(S, PI) と患者 PI の作業要素チューブ p-tube(S, PI) の二つの作業要素を用いてチューブにラベルが張られたことを状態として表す.

p-label :
患者 PI の作業要素ラベルが存在している状態 (not (p-label(S, PI) = no-label)) から患者 PI のためのラベルを貼る作業 put-label(S, PI) が行われる. それにより no-label に遷移する. ラベルがチューブに貼られたので貼るラベルが存在しなくなる状態を表す.

```
 bop put-label : Sys PatientId -> Sys
ceq p-order(put-label(S, PI1), PI2)
                      = p-order(S, PI2)
                      if c-put-label(S, PI1) .

ceq p-receipt(put-label(S, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-put-label(S, PI1) .

ceq p-tube(put-label(S, PI1), PI2)
                      =  (if PI1 =  PI2
                          then tube(p-label(S, PI2), get-blood(p-tube(S, PI2)))
                          else p-tube(S, PI2) fi)
                        if c-put-label(S, PI1) .

ceq p-label(put-label(S, PI1), PI2)
                      = no-label
                      if c-put-label(S, PI1) .

ceq put-label(S, PI) = S
                      if not c-put-label(S, PI) .
```

4.4.6 遷移：１３．ナースがオーダーの患者 ID とチューブに貼られたラベルの患者 ID を確認する.

チューブを用いて、採血する患者のために用意されたチューブか確認を行う.

遷移条件 c-tube-check

チューブが患者本人のものかの確認が行われていない, 確認するためのチューブが存在することが遷移するための条件となる.

```
op c-tube-check : Sys PatientId -> Bool .
eq c-tube-check(S, PI) = (get-tube-check(p-order(S, PI)) = no-check) and
                         (not (p-tube(S, PI) = no-tube ))  .
```

遷移関数 tube-check

観測関数 p-order の観測値が変化する.

p-order :

患者 PI の作業要素チューブが患者 PI のものか確認されていない状態 (get-tube-check(p-order(S, PI)) = no-check) から患者 PI のためのチューブが患者 PI のものか確認する作業 tube-check(S, PI) が行われる. 確認作業の中で患者 PI の作業要素チューブにラベルが貼られている (not (get-label(p-tube(S, PI)) = no-label)) ならば確認の結果がオーダーから返される. 確認の結果は, 患者 PI のための作業で用いられるチューブは患者 PI か (PI1 = get-patientid(get-label(p-tube(S, PI1)))) の条件によって結果が変わる. 条件を満たせば患者 PI の作業要素チューブは患者 ID の PI が記載されたラベルが貼られたチューブなので患者 PI のチューブと確認された order(EX, RE, check-true) 状態となる. 条件を満たせなければ患者 PI の作業要素チューブは患者 ID の PI とは一致しない患者 ID が記載されたラベルを持ったチューブなので患者 PI のチューブではないと確認された order(EX, RE, check-false) 状態となる.

```
bop tube-check : Sys PatientId -> Sys
ceq p-order(tube-check(S, PI1), PI2)
 =  (if PI1 =  PI2
      then if (not (get-label(p-tube(S, PI1)) = no-label))
           then if PI1 = get-patientid(get-label(p-tube(S, PI1)))
              then  order(get-examination(p-order(S, PI2)),
                          get-receipt-check(p-order(S, PI2)), check-true)
              else  order(get-examination(p-order(S, PI2)),
                           get-receipt-check(p-order(S, PI2)), check-false)  fi
           else p-order(S, PI2) fi
       else p-order(S, PI2) fi)
 if c-tube-check(S, PI1) .
```

```
ceq p-receipt(tube-check(S, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-tube-check(S, PI1) .

ceq p-label(tube-check(S, PI1), PI2)
                      = p-label(S, PI2)
                      if c-tube-check(S, PI1) .

ceq p-tube(tube-check(S, PI1), PI2)
                      = p-tube(S, PI2)
                      if c-tube-check(S, PI1) .

ceq tube-check(S, PI) = S
                      if not c-tube-check(S, PI) .
```

4.4.7 遷移：１５．オーダーに記述された患者 ID と受付票に記述された患者 ID を比較する.

受付票に記載された患者 ID を用いて, 採血する患者ため本人か確認を行う.

遷移条件 c-receipt-check

受付票が患者本人のものかの確認が行われていない, 確認するための受付票が存在する. チューブか患者本人のものと確認されていることが遷移するための条件となる.

```
op c-receipt-check : Sys PatientId -> Bool .
eq c-receipt-check(S, PI) = (get-receipt-check(p-order(S, PI)) = no-check) and
                            (not (p-receipt(S, PI) = no-receipt ) and
                            (get-tube-check(p-order(S, PI)) = check-true) ) .
```

遷移関数 receipt-check

観測関数 p-order から返される観測値が変化する.

p-order :

患者 PI の作業要素受付票が患者 PI のものか確認されていない状態 (get-receipt-check(p-order(S, PI)) = no-check) から患者 PI の作業要素受付票が患者 PI のものか確認する作業 receipt-check(S, PI) が行われる. 確認作業の中で患者 PI の作業要素受付票は患者 PI が記載された受付票か (PI1 = get-patientid(p-receipt(S, PI1))) の条件によって遷移が変わる. 条件を満たせば患者 PI の作業要素受付票は患者 ID の PI が記載された受付票なので患者 PI の受付票と確認された order(EX, check-true, true) 状態となる. 条件を満たせなければ患者 PI の作業で用いられる受付票は患者 ID の PI とは一致しない患者 ID が記載された

受付票なので患者 PI の受付票ではないと確認された order(EX, check-false, fakse) 状態となる.

```
bop receipt-check : Sys PatientId -> Sys
ceq p-order(receipt-check(S, PI1), PI2)
  =  (if PI1 =  PI2
     then if PI1 = get-patientid(p-receipt(S, PI1))
        then  order(get-examination(p-order(S, PI2)),
                    check-true, get-tube-check(p-order(S, PI2)))
        else  order(get-examination(p-order(S, PI2)),
                    check-false, get-tube-check(p-order(S, PI2)))  fi
     else p-order(S, PI2) fi)
  if c-receipt-check(S, PI1) .

ceq p-receipt(receipt-check(S, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-receipt-check(S, PI1) .

ceq p-label(receipt-check(S, PI1), PI2)
                      = p-label(S, PI2)
                      if c-receipt-check(S, PI1) .

ceq p-tube(receipt-check(S, PI1), PI2)
                      = p-tube(S, PI2)
                      if c-receipt-check(S, PI1) .

ceq receipt-check(S, PI) = S
                      if not c-receipt-check(S, PI) .
```

4.4.8 遷移：１７．ナースが患者から採血をする

チューブ、受付票を用いて患者から血液を採取を行う.

遷移条件 c-blood-gather

チューブ, 受付票が存在する. チューブに血液が入っていない. チューブと受付票が患者本人のものであると確認が行われていることが遷移するための条件となる.

```
op c-blood-gather : Sys PatientId -> Bool .
eq c-blood-gather(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                           (get-blood(p-tube(S, PI)) = no-blood) and
```

```
                    (not (p-receipt(S, PI)) = no-receipt) and
                    (get-receipt-check(p-order(S, PI)) = check-true)  and
                    (get-tube-check(p-order(S, PI)) = check-true) )  .
```

遷移関数 blood-gather
観測関数 p-tube から返される観測値が変化する.
p-tube :
患者 PI の作業要素チューブが存在 (not (p-tube(S, PI) = no-tube)) し, そのチューブに血液が入っていない (get-blood(p-tube(S, PI)) = no-blood) チューブに対して患者 PI のための採血作業 blood-gather(S, PI) が行われる. 患者 PI の作業要素受付票 p-receipt(S, PI) を用いてチューブの中に受付票に記載された患者 ID の血液 blood(get-patientid(p-receipt(S, PI)) が入れられる tube(L, blood(get-patientid(p-receipt(S, PI))). 今回, 患者 PI が記載された受付票を持っているは必ず患者 PI とする. つまり患者 PI が記載されたを持った受付票を持った患者 PI から取った血液 blood(get-patientid(p-receipt(S, PI)) をチューブに入れることを表す.

```
bop blood-gather : Sys PatientId -> Sys
ceq p-order(blood-gather(S, PI1), PI2)
                    = p-order(S, PI2)
                    if c-blood-gather(S, PI1) .

ceq p-receipt(blood-gather(S, PI1), PI2)
                    = p-receipt(S, PI2)
                    if c-blood-gather(S, PI1) .

ceq p-label(blood-gather(S, PI1), PI2)
                    = p-label(S, PI2)
                    if c-blood-gather(S, PI1) .

ceq p-tube(blood-gather(S, PI1), PI2)
                    =  (if PI1 =  PI2
                        then tube(get-label(p-tube(S, PI2)),
                        blood(get-patientid(p-receipt(S, PI2))))
                        else p-tube(S, PI2) fi)
                      if c-blood-gather(S, PI1) .

ceq blood-gather(S, PI) = S
                    if not c-blood-gather(S, PI) .
```

第5章　手順書モデルの検証

本章では作成した手順書モデルを用いた検証を行う. 検証を行うため手順書モデルの満たすべき性質を業務の目的として設定した. 手順書モデルが性質を満たせば, その手順書モデルは業務の目的を達成できると確認できる. 逆に手順書モデルが性質を満たすことが確認できなければ, 事故が起きる可能性を表す.

5.1　手順書の目的

患者 PI への採血業務が行われたとき, 患者 PI のために用意されたチューブを用いて, 患者 PI 本人から採血が行われるかを検証する.CafeOBJ 上で記述すると

```
eq inv1(S, P) =
(not (get-blood(p-tube(S, P)) = no-blood))
implies
(P = get-patientid(get-blood(p-tube(S, P))) and
 P = get-patientid(get-label(p-tube(S, P)))) .
```

患者 P の作業要素チューブに血が入っているなら,
患者 P の作業要素チューブの血は患者 P のものである.
患者 P の作業要素チューブに貼られたラベルに記載される名前は患者 P である.

5.2　検証

基底段階における証明のため以下の証明節を作成し CafeOBJ 上で実行すると true が返された.

```
-- BaseCase
open INV1 .
op p : -> PatientId .
red inv1(init, p) .
close
```

次に帰納段階における証明を行った. 遷移関数ごとの証明節を作成し, 証明を行っている.

遷移関数 order-create

遷移条件 c-order-create が成り立つ場合, 成り立たない場合で証明節を作成し,true が返された.

遷移関数 card-read

遷移条件 c-card-read が成り立つ場合, 成り立たない場合で証明節を作成し, true が返された.

遷移関数 put-label

遷移条件 c-put-label が成り立つ場合

さらに p1 = get-patientid(no-label) が成り立つ場合, 成り立たない場合で証明節を作成することで true が返された.

遷移条件 c-put-label 成り立たない場合

true が返された.

遷移関数 receipt-check

遷移条件 c-receipt-check が成り立つ場合, 成り立たない場合で証明節を作成し, true が返された.

遷移関数 tube-check

遷移条件 c-tube-check が成り立つ場合, 成り立たない場合で証明節を作成し, true が返された.

遷移関数 blood-gather

遷移条件 c-blood-gather が成り立つ場合

証明節を実行すると以下の結果が返される.

```
((p1 = get-patientid(p-receipt(s,p1))) and
 (p1 = get-patientid(get-label(p-tube(s,p1))))):Bool
```

遷移条件 c-blood-gather が成り立たない場合

true が返される.

遷移条件 c-blood-gather が成り立つ場合, 条件分けではこれ以上有効な証明節を作成することができなかったので二つの補題を用いた.

補題 1

これから作業をする患者 ID とレシートに記述された患者 ID を比較した結果が一致 check-true する. ならば患者 P の作業要素受付表 p-receipt(S,P) に記載された患者 ID get-patientid(p-receipt(S,P)) は P である.

```
eq inv2(S, P) = (get-receipt-check(p-order(S, P)) = check-true)
              implies
               (P = get-patientid(p-receipt(S,P)))  .
```

補題 2
これから作業をする患者 ID とチューブに貼られたラベルの患者 ID を比較した結果が一致 check-true する. ならば患者 P の作業要素チューブ p-tube(S, P) に貼られたラベルに記載された患者 ID get-patientid(get-label(p-tube(S, P))) は P である.

```
eq inv3(S, P) = (get-tube-check(p-order(S, P)) = check-true)
                implies
                  (P = get-patientid(get-label(p-tube(S, P))))
```

補題 1 と 2 を用いることで遷移条件 c-blood-gather が成り立つ場合の証明節が true を返し,inv1 の証明を行うことができた.
補題 1 の証明
inv1 と同じように遷移演算ごとの証明節を作成し, 条件分けをしていくことで証明することができた.
補題 2 の証明
遷移条件 c-put-label 以外の遷移演算は証明節を作成し, 条件分けをしていくことで証明することができた.
遷移条件 c-put-label が成り立ち,
(((get-tube-check(p-order(s,p1)) = no-check) = false) の場合
証明節を実行すると以下の結果が返される.

```
((((p1 = get-patientid(p-label(s,p1))) and
(p1 = getpatientid(no-label))) and
(get-tube-check(p-order(s,p1)) = check-true)) xor
(((get-tube-check(p-order(s,p1)) = check-true) and
(p1 = get-patientid(no-label))) xor true)):Bool
```

条件分けではこれ以上有効な証明節を作成することができなかったので一つの補題を用いた.
補題 3
患者 P の作業要素チューブにラベルが貼られていない (get-label(p-tube(S, P)) = no-label) . ならばこれから作業をする患者 ID とチューブに貼られたラベルの患者 ID を比較する作業がまだ行われていない no-check .

```
eq inv4(S, P) = (get-label(p-tube(S, P)) = no-label)
                implies
                  (get-tube-check(p-order(S, P)) = no-check)  .
```

補題 3 を用いることで遷移条件 c-put-label が成り立つ場合の証明節が true を返し, 補題 2 の証明を行うことができた.
補題 3 の証明

inv1 と同じように遷移演算ごとの証明節を作成し, 条件分けをしていくことで証明することができた.

全ての命題に対して証明譜を作成することができ, 手順書の目的を達成できる手順書のモデルであると検証することができた.

第6章　手順書モデルにヒューマンエラーを含ませたモデルの作成

ヒューマンエラーは先に述べた通りに, 手順書に書かれた作業と異なる作業をすることである. そこで遷移条件を緩める, 新たな遷移関数を加えることでヒューマンエラーを表現した. 検体検査手順書モデルではヒューマンエラーを分類した中でオミッションエラーとコミッションエラーの作業遂行の順序, 本来やるべきでない作業（取り違え）を組み込む.

6.1　オミッションエラー, 作業遂行の順序

ヒューマンエラーの記述を行う. 手順書は目的を達成するために作業の順番を明示的に記述したものとなる. そのためモデルを作成するとき遷移の順番が決まるように遷移条件を決める. そこで遷移するための条件を緩めることで複数の遷移を作り, 作業を抜いたり入れ替えるといったヒューマンエラーを表現する.
例：採血作業を行うときの遷移条件を緩くする.

```
eq c-blood-gather(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                            (get-blood(p-tube(S, PI)) = no-blood) and
                            (not (p-receipt(S, PI)) = no-receipt) and
                            (get-receipt-check(p-order(S, PI)) = check-true)  and
                            (get-tube-check(p-order(S, PI)) = check-true) )  .
```

⇒

```
eq c-blood-gather(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                            (get-blood(p-tube(S, PI)) = no-blood) and
                            (not (p-receipt(S, PI)) = no-receipt))  .
```

手順書通りの順番に従うと, 確認作業を行って採血を行う. つまり確認作業を行った結果 (get-receipt-check(p-order(S, PI)) = check-true) と (get-tube-check(p-order(S, PI)) = check-true) が採血作業の遷移を行うための条件となる. ヒューマンエラーを組み込むためにヒューマンエラーを組み込んだモデルでは, その条件を抜いている. 条件を抜くことで確認作業を行ってから採血作業を行う遷移, 確認をせずに採血作業を行う遷移と手順書の順番が変わった遷移を作ることができる.

6.2 取り違え

手順を抜くと入れ替えるといったヒューマンエラーと違い遷移条件を加えるだけでは取り違えを表すことができない. 新たに遷移関数を加えることで誤った遷移を行うヒューマンエラーを組み込む. 今回は受付票の取り違えをモデルの中に組み込んだ. ある患者のための作業で用いられていた受付表を違う患者の作業に紛れ込ませる. そうすることで誤ってその受付票を用いてしまう状況を作り, 取り違えを表した.

- 受付票の取り違い
  bop receipt-pass-other : Sys PatientId PatientId -> Sys

遷移条件 c-receipt-pass-othe
患者 PI1 のための作業で用いられる受付表が存在 (not (p-receipt(S, PI1) = no-receipt)) している. 患者 PI1 のための作業で用いられる受付票が患者 PI1 のものか確認されていない (get-receipt-check(p-order(S, PI1)) = no-check) 状態である. また患者 PI2 のための作業で用いられる受付表が存在 (not (p-receipt(S, PI2) = no-receipt)) している. 患者 PI2 のための作業で用いられる受付票が患者 PI2 のものか確認されていない (get-receipt-check(p-order(S, PI2)) = no-check) 状態であれば受付票の取り違いが起きる. つまり受付票 2 つ存在して, 確認作業が行われていない場合, 取り違えが起きる可能性を表している.

```
op c-receipt-pass-other : Sys PatientId PatientId -> Bool .
eq c-receipt-pass-other(S, PI1, PI2) =
                    (get-receipt-check(p-order(S, PI1)) = no-check) and
                    (get-receipt-check(p-order(S, PI2)) = no-check) and
                    (not (p-receipt(S, PI1) = no-receipt)) and
                    (not (p-receipt(S, PI2) = no-receipt))  .
```

遷移関数 receipt-pass-othe
観測関数 p-receipt から返される観測値が変化する.
p-receipt :
患者 PI1 と患者 PI2 のそれぞれの作業要素に受付票が存在するとき receipt-pass-other(S, PI1, PI2) が行われる. 患者 PI2 の作業要素受付票が患者 PI1 の作業要素受付表 receipt(get-patientid(p-receipt(S, PI1)) へ移る.

```
ceq p-order(receipt-pass-other(S, PI1, PI2), PI3)
                        = p-order(S, PI3)
                        if c-receipt-pass-other(S, PI1, PI2) .


ceq p-receipt(receipt-pass-other(S, PI1, PI2), PI3)
                        = (if PI2 =  PI3
```

```
                        then receipt(get-patientid(p-receipt(S, PI1)))
                        else p-receipt(S, PI3) fi)
                    if c-receipt-pass-other(S, PI1, PI2) .


ceq p-label(receipt-pass-other(S, PI1, PI2), PI3)
                    = p-label(S, PI3)
                    if c-receipt-pass-other(S, PI1, PI2) .

ceq p-tube(receipt-pass-other(S, PI1, PI2), PI3)
                    = p-tube(S, PI3)
                    if c-receipt-pass-other(S, PI1, PI2) .


ceq receipt-pass-other(S, PI1, PI2) = S
                    if not c-receipt-pass-other(S, PI1, PI2) .
```

6.2.1 取り違えの状態

実際の取り違えが起きた状況を CafeOBJ 上で実行したものを図 6.1 で示す. 患者 I D の p1,p2 のための作業は両方ともオーダーが作成, カード読み込み作業が行われ受付票が用意されている. その後遷移関数 receipt-pass-other によって二つの受付票が取り違えられたときの状態を表す. 書き換えの結果を見てみると p-receipt(Sys, p1) が receipt(p2) となっている. 患者 p1 のための作業を行うときに患者 p2 の受付票がまぎれこんでしまい誤って作業に用いてしまう可能性あることを表す.

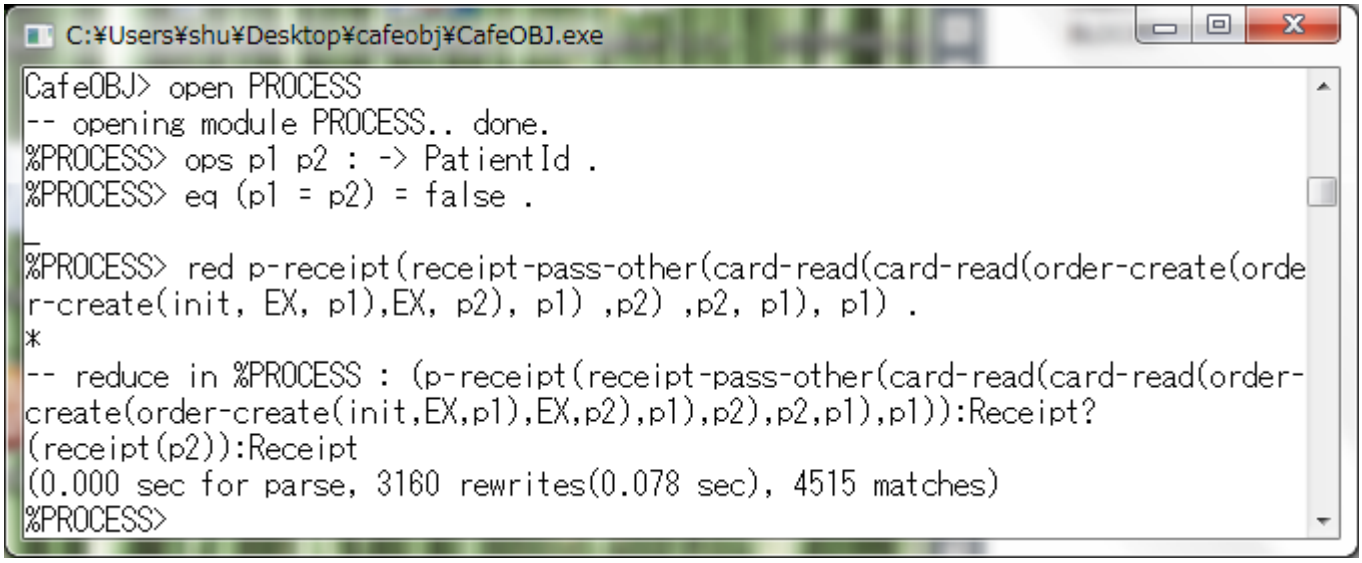
C:¥Users¥shu¥Desktop¥cafeobj¥CafeOBJ.exe
CafeOBJ> open PROCESS
-- opening module PROCESS.. done.
%PROCESS> ops p1 p2 : -> PatientId .
%PROCESS> eq (p1 = p2) = false .
%PROCESS> red p-receipt(receipt-pass-other(card-read(card-read(order-create(orde
r-create(init, EX, p1),EX, p2), p1) ,p2) ,p2, p1), p1) .
*
-- reduce in %PROCESS : (p-receipt(receipt-pass-other(card-read(card-read(order-
create(order-create(init,EX,p1),EX,p2),p1),p2),p2,p1),p1)):Receipt?
(receipt(p2)):Receipt
(0.000 sec for parse, 3160 rewrites(0.078 sec), 4515 matches)
%PROCESS>

図 6.1: 取り違え実行例

第7章　ヒューマンエラーを含んだ手順書モデルの検証

ヒューマンエラーを加えたモデルを用いた手順書モデルの解析を行う. 検体検査手順書モデルでは,7 章で提案したオミッションエラーや取り違えを含んだモデル 3 つを作成し、それぞれが手順書の目的を達成できるか検証を行った. ヒューマンエラーを加えたモデルが業務の目的を達成できると検証された場合, そのヒューマンエラーが起きても事故に至らない手順書モデルであることがいえる. 逆に, 手順書モデルが業務の目的を達成できると検証したにも関わらず、ヒューマンエラーを加えたモデルが業務の目的を達成できると検証できない場合、そのヒューマンエラーが事故を起こす可能性を表す.

7.1　手順書モデルにオミッションエラー, 作業遂行の順序を含ませたモデルの検証

5 章で記述した手順書の目的を達成する inv1 を用いて, ヒューマンエラーを含んだモデルでも手順書の目的を達成できる手順書か検証を行う.

```
eq inv1(S, P) =
(not (get-blood(p-tube(S, P)) = no-blood))
implies
(P = get-patientid(get-blood(p-tube(S, P))) and
 P = get-patientid(get-label(p-tube(S, P)))) .
```

患者 P の作業要素で血が入ったチューブがあるならば,
患者 P の作業要素チューブに入っている血はその患者 P のものである.
かつ患者 P 作業要素チューブに張られたたラベルの名前はその患者 P と一致する.

7.1.1　手順書モデルの変更

6 章で記述したとおりオミッションエラー, 作業遂行の順序のヒューマンエラーを組み込むためには遷移するための条件を緩める. そこで手順書モデルに記述される遷移関数を

以下のように変更した.

遷移条件 c-blood-gather

6 章の例でも説明を行ったが, 手順書では確認作業を行ったあと遷移関数 blood-gather が行われる. そこで確認作業を行わなくても blood-gather ができるように遷移条件を抜いた.

```
eq c-blood-gather(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                            (get-blood(p-tube(S, PI)) = no-blood) and
                            (not (p-receipt(S, PI)) = no-receipt) and
                            (get-receipt-check(p-order(S, PI)) = check-true)  and
                            (get-tube-check(p-order(S, PI)) = check-true) )  .
```

⇒

```
eq c-blood-gather(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                            (get-blood(p-tube(S, PI)) = no-blood) and
                            (not (p-receipt(S, PI)) = no-receipt))  .
```

遷移条件 c-receipt-check

手順書どおりに従うと遷移関数 tube-check を行ってから, 遷移関数 receipt-check 行う. そのため手順書のモデルでは (get-tube-check(p-order(S, PI)) = check-true) の条件があることで確認作業の順番が決まっていた. そこでその条件を抜くことで確認作業の順序が入れ替わるようになった.

```
eq c-receipt-check(S, PI) = (get-receipt-check(p-order(S, PI)) = no-check) and
                             (not (p-receipt(S, PI) = no-receipt ) and
                             (get-tube-check(p-order(S, PI)) = check-true) ) .
```

⇒

```
eq c-receipt-check(S, PI) = (get-receipt-check(p-order(S, PI)) = no-check) and
                             (not (p-receipt(S, PI) = no-receipt ) and  .
```

7.1.2 検証

第 5 章で記述した inv1 を用いて検証を行った.

inv1

遷移条件 c-blood-gather 以外の遷移演算は証明節を作成し, 条件分けをしていくことで証明することができた.

遷移条件 c-blood-gather が成り立つ場合

証明節を実行すると以下の結果が返される.

```
((p1 = get-patientid(p-receipt(s,p1))) and
(p1 = get-patientid(get-label(p-tube(s,p1))))):Bool
```

遷移条件 c-blood-gather が成り立つ場合, 条件分けではこれ以上有効な証明節を作成することができなかった. 採血が行われる前に患者 p1　の作業要素は p1 のものであることがいえる必要があると考え二つの補題を用いた.

補題 4

患者 P の作業で用いられる受付票が作成されている (not (p-receipt(S, P) = no-receipt)) . ならば受付票には患者 P が記載されている (P = get-patientid(p-receipt(S,P))) .

```
eq inv2(S, P) = (not (p-receipt(S, P) = no-receipt))
                implies
                (P = get-patientid(p-receipt(S,P)))  .
```

補題 5

患者 P の作業要素チューブが作成されている (not (get-label(p-tube(S, P)) = no-label)) . ならばチューブには患者 P が記載されている (P = get-patientid(get-label(p-tube(S, P)))) .

```
eq inv3(S, P) = (not (get-label(p-tube(S, P)) = no-label))
                implies
                (P = get-patientid(get-label(p-tube(S, P))))  .
```

補題 4 では受付票, 補題 5 ではチューブそれぞれの作業要素が存在すれば必ず患者本人のものであることを保障し, 表したものである. このモデルでは取り違えが起きないモデルなので補題 4 と 5 を用いれることが可能と考えた. 実際に補題 4 と 5 を用いたところ遷移条件 c-blood-gather が成り立つ場合の証明節が true を返し, inv1 の証明を行うことができた.

補題 4 の証明

inv1 と同じように遷移関数ごとの証明節を作成し, 条件分けをしていくことで証明することができた.

補題 5 の証明

遷移条件 c-put-label 以外の遷移関数は証明節を作成し, 条件分けをしていくことで証明することができた.

遷移条件 c-put-label が成り立つ場合

証明節を実行すると以下の結果が返される.

```
(p1 = get-patientid(p-label(s,p1))):Bool
```

条件分けではこれ以上有効な証明節を作成することができなかったので一つの補題を用いた.

```
eq inv4(S, P) = (not (p-label(S, P) = no-label))
              implies
                (P = get-patientid(p-label(S, P)))  .
```

補題 6 では作業要素であるラベルが存在すれば必ず患者本人のものと保障することを表したものである. 補題 6 を用いることで遷移条件 c-put-label が成り立つ場合の証明節が true を返し, 補題 5 の証明を行うことができた.

補題 6 の証明

inv1 と同じように遷移関数ごとの証明節を作成し, 条件分けをしていくことで証明することができた.

全ての命題に対して証明譜を作成することができ, 手順書モデルにオミッションエラー, 作業遂行の順序を含ませたモデルでも手順書の目的を達成できると検証することができた.

7.2 手順書モデルに取り違えを含ませたモデルの検証

7.2.1 手順書モデルの変更

6 章で記述した新たな遷移関数 receipt-pass-other を手順書のモデルに組み込むことで取り違えを含んだモデルを作成した.

7.2.2 検証

第 5 章で記述した inv1 を用いて取り違えを含んだモデルが手順書の目的を達成できるか検証を行った.

inv1

遷移条件 c-blood-gather 以外の遷移関数は証明節を作成し, 条件分けをしていくことで証明することができた.

遷移条件 c-blood-gather が成り立つ場合

証明節を実行すると以下の結果が返される.

```
((p1 = get-patientid(p-receipt(s,p1))) and (p1 = get-patientid(get-lab
el(p-tube(s,p1))))):Bool
```

遷移条件 c-blood-gather が成り立つ場合, 条件分けではこれ以上有効な証明節を作成することができなかったので二つの補題を用いた.

第 5 章で用いた補題 1 と 2 になる. 補題 1 と 2 を用いることで遷移条件 c-blood-gather が成り立つ場合の証明節が true を返し, inv1 の証明を行うことができた.

補題 1 の証明

inv1 と同じように遷移関数ごとの証明節を作成し, 条件分けをしていくことで証明することができた.

補題 2 の証明

inv1 と同じように遷移関数ごとの証明節を作成し, 条件分けをしていくことで証明するこ

とができた.
遷移条件 c-put-label 以外の遷移関数は証明節を作成し, 条件分けをしていくことで証明することができた.
遷移条件 c-put-label が成り立ち,
(((get-tube-check(p-order(s,p1)) = no-check) = false) の場合
証明節を実行すると以下の結果が返される.

```
((((p1 = get-patientid(p-label(s,p1))) and
(p1 = getpatientid(no-label))) and
(get-tube-check(p-order(s,p1)) = check-true)) xor
(((get-tube-check(p-order(s,p1)) = check-true) and
(p1 = get-patientid(no-label))) xor true)):Bool
```

条件分けではこれ以上有効な証明節を作成することができなかったので一つの補題を用いた. 5 章で用いた補題 3 となる. 補題 3 を用いることで遷移条件 c-put-label が成り立ち,(((get-tube-check(p-order(s,p1)) = no-check) = false) の場合の証明節は true を返し補題 2 を証明することができた.
補題 3 の証明
inv1 と同じように遷移関数ごとの証明節を作成し, 条件分けをしていくことで証明することができた.

全ての命題に対して証明譜を作成することができ, 手順書モデルに受付票の取り違えを含ませたモデルでも手順書の目的を達成できると検証することができた.

7.3 手順書モデルにオミッションエラー, 作業遂行の順序と取り違えとを含ませたモデルの検証

第 5 章で記述した inv1 を用いててヒューマンエラーを含んだモデルが手順書の目的を達成できるか検証を行った.
inv1
遷移条件 c-blood-gather 以外の遷移関数は証明節を作成し, 条件分けをしていくことで証明することができた.
遷移条件 c-blood-gather が成り立つ場合
証明節を実行すると以下の結果が返される.

```
((p1 = get-patientid(p-receipt(s,p1))) and
(p1 = get-patientid(get-label(p-tube(s,p1))))):Bool
```

このモデルに関しては証明を行うことができなかった. 先の二つで利用した補題を利用することを考えたが, 取り違えを含んだモデルの検証時に用いた補題 1 と 2 はオミッション

エラーによって用いることができなくなり、補題4と5は取り違えによって用いることができなくなってしまった. 複合的にヒューマンエラーが関わることで手順書の目的を達成できなくするような状況になってしまっている可能性がある.

第8章　結論

8.1　まとめ

本研究では以下のことを行った.

- 手順書から人の動作を作業として抜き出し, 手順書を解析するための作業の列を作成した.
- 抜き出した作業から作業のときに用いられるもの, そのものを変化させるための作業に注目した手順書モデルの作成を行った.
- 検体検査の目的を命題として設定した.
- 手順書モデルを用いて検体検査の目的を達成できる手順書か検証を行った.
- 事故の原因となるヒューマンエラーを定義, 分類を行った.
- 検体検査の目的を達成できる手順書モデルにヒューマンエラーを組み込む手法を考え手順書モデルにヒューマンエラーを含ませたモデルを作成した.
- 手順書モデルにヒューマンエラーを含ませたモデルを用いてどのようなヒューマンエラーが事故に繋がるか検証を行った.

8.2　関連研究

関連研究としては, [10] 永藤によって同じように検体検査の手順書をプロセス代数を用いてモデル化を行っている. さらに人的誤りをモデルに組込みモデル検査によって手順書の頑健性の検証を行っている. [6] Osterweil らはプロセスを定義するための言語である Little-JIL を用いて医療業務の正確な仕様作成を行っている.

8.3　今後

- 手順書モデルの詳細化と新たなヒューマンエラーの追加
  今回手順書モデルを作成するにおいていくつか単純化した部分の詳細化と他のヒューマンエラーを加えることで様々な業務上の状況を考えることができるのではないか.

- 他業務手順書の解析
  今回は検体検査の医療業務のみを取り扱ったが他業務でもこの提案手法を用いて他業務でも提案手法が応用できるか.

- 他業務並行した状況のモデル化
  医療現場では検体検査だけではなく, 様々な異なる業務が並行して行われている. 今回のモデルでは同じ業務が並行して行われているモデルは作成することができたが, 他業務が並行して動くモデルといった現実に近いモデルを考える.

謝辞

本研究を進めるにあたって長い間ご指導してくださった二木厚吉教授厚く御礼を申し上げます. また副指導教員として貴重なご意見をいただいた青木利晃准教授に深く御礼申し上げます. 研究だけではなく, 輪講でも熱心にご指導をしてくださった千葉勇輝助教に深く御礼を申し上げます. 何度も研究の相談に乗ってくださった永藤直行さん, 有本泰仁さんに深く御礼を申し上げます. そして二木研究室, 青木研究室のみなさんに感謝いたします.

参考文献

[1] 安全門. http://anzenmon.jp/.

[2] 信頼性工学のはなし. 日科技連出版社, 1988.

[3] 人は誰でも間違える: より安全な医療システムを目指して. 日本評論社, 2000.

[4] ヒューマンエラー. 丸善, 2003.

[5] Kazuhiro Ogata, Kokichi Futatsugi, and Nec Software Hokuriku. Proof scores in the ots/cafeobj method. In In Proc. of The 6th IFIP WG6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2003), volume 2884 of LNCS, pp. 170–184. Springer, 2003.

[6] Leon J. Osterweil, George S. Avrunin, Bin Chen, Lori A. Clarke, Rachel Cobleigh, Elizabeth A. Henneman, and Philip L. Henneman. Engineering medical processes to improve their safety: An experience report.

[7] A.D. Swain, H.E. Guttmann, U.S. Nuclear Regulatory Commission, United States. Office of Nuclear Regulatory Research, and Sandia Laboratories. Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications: Final Report. Sandia National Laboratories, 1983.

[8] 二木厚吉, 緒方和博, 中村正樹. Cafeobj 入門 (1) : 形式手法と cafeobj. コンピュータソフトウェア, Vol. 25, No. 2, pp. 1–13, apr 2008.

[9] 中村正樹, 二木厚吉, 緒方和博. Cafeobj 入門 (2) : 構文と意味. コンピュータソフトウェア, Vol. 25, No. 2, pp. 14–27, apr 2008.

[10] 永藤直行, 二木厚吉, 緒方和博. 状況に依存しない人的誤りの埋込み手法, 2011.

付録 手順書モデル

```
full-reset
-- 患者の ID 現状では患者の名前＝患者の ID
mod* PATIENTID {
 [ PatientId ]
 op _=_ : PatientId PatientId -> Bool {comm}
 var PI : PatientId
 eq (PI = PI) = true .

}

-- 試験内容
mod! EXAMINATION{
 [ Examination ]

 op _=_ : Examination Examination -> Bool {comm}

 var E : Examination

 eq (E = E) = true .
}

mod! BLOOD {
  [ None_B Blood < Blood? ]

  pr(PATIENTID)

  ops no-blood : -> None_B
  op blood : PatientId -> Blood
  op get-patientid : Blood? -> PatientId

  op _=_ : Blood? Blood? -> Bool {comm}
```

```
  var B : Blood?
  var BE : Blood
  var N : None_B
  var PI : PatientId

  eq get-patientid(blood(PI)) = PI .

  eq (B = B) = true .
  eq (BE = N) = false .
}

-- 確認
mod! CHECK {
 [ Check ]
 ops no-check check-false check-true : -> Check
 op _=_ : Check Check -> Bool {comm}

 var C : Check

 eq (C = C) = true .
 eq (no-check = check-false) = false .
 eq (no-check = check-true) = false .
 eq (check-false = check-true) = false .
}

mod! LABEL{
 [ None_L Label < Label? ]

 pr(PATIENTID + EXAMINATION)
 op no-label : -> None_L
 op label : PatientId Examination -> Label
 op get-patientid : Label? -> PatientId
 op get-examination : Label? -> Examination
 op _=_ : Label? Label? -> Bool {comm}

 var V : Label?
 var VE : Label
```

```
 var N : None_L
 var E : Examination
 var PI : PatientId

 eq get-patientid(label(PI, E)) = PI .

 eq get-examination(label(PI, E)) = E .

 eq (V = V) = true .
 eq (V = VE) = false .

}

-- オーダー
mod! ORDER {
 [ None_O Order < Order? ]

 pr(PATIENTID + EXAMINATION + CHECK)
 -- オーダー（患者の ID）
 op no-order : -> None_O
 op order : Examination Check Check -> Order
 -- オーダーの試験内容.
 op get-examination : Order? -> Examination
 op get-receipt-check : Order? -> Check
 op get-tube-check : Order? -> Check

 op _=_ : Order? Order? -> Bool {comm}

 var O : Order?
 var OE : Order
 var N : None_O
 var PI : PatientId
 var EX : Examination
 vars C1 C2 : Check

 eq get-examination(order(EX, C1, C2)) = EX .
 eq get-receipt-check(order(EX, C1, C2)) = C1 .
 eq get-receipt-check(no-order) = no-check .
```

```
 eq get-tube-check(order(EX, C1, C2)) = C2 .
 eq get-tube-check(no-order) = no-check .

 eq (O = O) = true .
 eq (N = OE) = false .

}
-- 受付票
mod! RECEIPT {
 [ None_R Receipt < Receipt? ]

 pr(PATIENTID)

 op no-receipt : -> None_R
 op receipt : PatientId -> Receipt
 op get-patientid : Receipt? -> PatientId
 op _=_ : Receipt? Receipt? -> Bool {comm}

 var R : Receipt?
 var N : None_R
 var RE : Receipt
 var PI : PatientId

 eq get-patientid(receipt(PI)) = PI .

 eq (R = R) = true .
 eq (N = RE) = false .

}

mod! TUBE {
 [ None_T Tube < Tube? ]

 pr(PATIENTID + EXAMINATION + BLOOD + LABEL)

 op no-tube : -> None_T
 op tube : Label? Blood? -> Tube
 op get-blood : Tube? -> Blood?
```

```
 op get-label : Tube? -> Label
 op _=_ : Tube? Tube? -> Bool {comm}

 var T : Tube?
 var N : None_T
 var TE : Tube
 var L : Label?
 var EX : Examination
 var B : Blood?

 eq get-blood(tube(L, B)) = B .
 eq get-blood(no-tube) = no-blood .

 eq get-label(tube(L, B)) = L .
 eq get-label(no-tube) = no-label .

 -- eq get-examination(tube(L, EX, B)) = EX .

 eq (T = T) = true .
 eq (N = TE) = false .

}

mod* PROCESS {

 *[ Sys ]*

 pr(ORDER + RECEIPT + CHECK + TUBE)

 op init : -> Sys

 -- 観測演算
 bop p-order : Sys PatientId -> Order?
 bop p-receipt : Sys PatientId -> Receipt?
 bop p-tube : Sys PatientId -> Tube?
 bop p-label : Sys PatientId -> Label?

 -- 遷移関数
```

```
-- オーダー作成
bop order-create : Sys Examination PatientId -> Sys
-- 受付票読み込み（レシート, チューブ, ラベル発行）
bop card-read : Sys PatientId -> Sys
bop put-label : Sys PatientId -> Sys
-- 名前比較（オーダー（カルテのシステム）と受付票の内容）
bop receipt-check : Sys PatientId -> Sys
 -- 名前比較（オーダー（カルテのシステム）とチューブの内容）
bop tube-check : Sys PatientId -> Sys
-- 採血
bop blood-gather : Sys PatientId -> Sys

var S : Sys
var EX : Examination
vars PI PI1 PI2 PI3 : PatientId
var B : Blood

eq p-order(init, PI) = no-order .
eq p-receipt(init, PI) = no-receipt .
eq p-tube(init, PI) = no-tube .
eq p-label(init, PI) = no-label .

-- オーダー発行 (オーダーが出ていない)
op c-order-create : Sys PatientId -> Bool .
eq c-order-create(S, PI) = (p-order(S, PI) = no-order)  .
ceq p-order(order-create(S, EX, PI1), PI2)
                      = (if PI1 =  PI2 then
                       order(EX, no-check, no-check) else p-order(S, PI2) fi)
                      if c-order-create(S, PI1) .

ceq p-receipt(order-create(S, EX, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-order-create(S, PI1) .

ceq p-label(order-create(S, EX, PI1), PI2)
                      = p-label(S, PI2)
                      if c-order-create(S, PI1) .
```

```
ceq p-tube(order-create(S, EX, PI1), PI2)
                      = p-tube(S, PI2)
                      if c-order-create(S, PI1) .

ceq order-create(S, EX, PI) = S
                      if not c-order-create(S, PI) .

op c-card-read : Sys PatientId -> Bool .
eq c-card-read(S, PI) = (not (p-order(S, PI) = no-order)) and
                        (p-receipt(S, PI) = no-receipt) and
                        (p-tube(S, PI) = no-tube) and
                        (p-label(S, PI) = no-label)   .

ceq p-order(card-read(S, PI1), PI2)
                      = p-order(S, PI2)
                      if c-card-read(S, PI1) .

ceq p-receipt(card-read(S, PI1), PI2)
                      = (if PI1 =  PI2 then receipt(PI2)
                      else p-receipt(S, PI2) fi)
                      if c-card-read(S, PI1) .


ceq p-label(card-read(S, PI1), PI2)
                    = (if PI1 =  PI2
                    then label(PI2, get-examination(p-order(S, PI2)))
                    else p-label(S, PI2) fi)
                    if c-card-read(S, PI1) .

ceq p-tube(card-read(S, PI1), PI2)
                      = (if PI1 =  PI2
                      then tube(no-label, no-blood)
                      else p-tube(S, PI2) fi)
                      if c-card-read(S, PI1) .

ceq card-read(S, PI) = S
                      if not c-card-read(S, PI) .
```

```
-- ラベルを貼る
 op c-put-label : Sys PatientId -> Bool .
 eq c-put-label(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                          (get-label(p-tube(S, PI)) = no-label) and
                          (not (p-label(S, PI) = no-label))) .

 ceq p-order(put-label(S, PI1), PI2)
                       = p-order(S, PI2)
                       if c-put-label(S, PI1) .

 ceq p-receipt(put-label(S, PI1), PI2)
                       = p-receipt(S, PI2)
                       if c-put-label(S, PI1) .


 ceq p-tube(put-label(S, PI1), PI2)
                       =  (if PI1 =  PI2
                           then tube(p-label(S, PI2), get-blood(p-tube(S, PI2)))
                           else p-tube(S, PI2) fi)
                         if c-put-label(S, PI1) .

 ceq p-label(put-label(S, PI1), PI2)
                       = no-label
                       if c-put-label(S, PI1) .

 ceq put-label(S, PI) = S
                       if not c-put-label(S, PI) .

 -- 名前比較 (名前比較がまだ行われていない and レシートが発行されている)
 op c-receipt-check : Sys PatientId -> Bool .
 eq c-receipt-check(S, PI) = (get-receipt-check(p-order(S, PI)) = no-check) and
                             (not (p-receipt(S, PI) = no-receipt ) and
                             (get-tube-check(p-order(S, PI)) = check-true) ) .


 ceq p-order(receipt-check(S, PI1), PI2)
      =  (if PI1 =  PI2
     then if PI1 = get-patientid(p-receipt(S, PI1))
```

```
    then  order(get-examination(p-order(S, PI2)), check-true,
     get-tube-check(p-order(S, PI2)))
    else  order(get-examination(p-order(S, PI2)), check-false,
    get-tube-check(p-order(S, PI2)))  fi
    else p-order(S, PI2) fi)
    if c-receipt-check(S, PI1) .

ceq p-receipt(receipt-check(S, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-receipt-check(S, PI1) .

ceq p-label(receipt-check(S, PI1), PI2)
                      = p-label(S, PI2)
                      if c-receipt-check(S, PI1) .

ceq p-tube(receipt-check(S, PI1), PI2)
                      = p-tube(S, PI2)
                      if c-receipt-check(S, PI1) .

ceq receipt-check(S, PI) = S
                      if not c-receipt-check(S, PI) .

-- 名前比較 (名前比較がまだ行われていない and チューブが)
op c-tube-check : Sys PatientId -> Bool .
eq c-tube-check(S, PI) = (get-tube-check(p-order(S, PI)) = no-check)
                         and (not (p-tube(S, PI) = no-tube ))  .


ceq p-order(tube-check(S, PI1), PI2)
   =  (if PI1 =  PI2
   then if (not (get-label(p-tube(S, PI1)) = no-label))
   then if PI1 = get-patientid(get-label(p-tube(S, PI1)))
   then  order(get-examination(p-order(S, PI2)),
   get-receipt-check(p-order(S, PI2)), check-true)
   else  order(get-examination(p-order(S, PI2)),
   get-receipt-check(p-order(S, PI2)), check-false)  fi
   else p-order(S, PI2) fi
   else p-order(S, PI2) fi)
```

```
   if c-tube-check(S, PI1) .

ceq p-receipt(tube-check(S, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-tube-check(S, PI1) .

ceq p-label(tube-check(S, PI1), PI2)
                      = p-label(S, PI2)
                      if c-tube-check(S, PI1) .

ceq p-tube(tube-check(S, PI1), PI2)
                      = p-tube(S, PI2)
                      if c-tube-check(S, PI1) .

ceq tube-check(S, PI) = S
                      if not c-tube-check(S, PI) .


-- 採血 (チューブがある and 血が採られていない)
op c-blood-gather : Sys PatientId -> Bool .
eq c-blood-gather(S, PI) = ((not (p-tube(S, PI) = no-tube)) and
                           (get-blood(p-tube(S, PI)) = no-blood) and
                           (not (p-receipt(S, PI)) = no-receipt) and
                           (get-receipt-check(p-order(S, PI)) = check-true)  and
                           (get-tube-check(p-order(S, PI)) = check-true) )  .

ceq p-order(blood-gather(S, PI1), PI2)
                      = p-order(S, PI2)
                      if c-blood-gather(S, PI1) .

ceq p-receipt(blood-gather(S, PI1), PI2)
                      = p-receipt(S, PI2)
                      if c-blood-gather(S, PI1) .

ceq p-label(blood-gather(S, PI1), PI2)
                      = p-label(S, PI2)
                      if c-blood-gather(S, PI1) .
```

```
 ceq p-tube(blood-gather(S, PI1), PI2)
                       =  (if PI1 =  PI2
            then tube(get-label(p-tube(S, PI2)),
            blood(get-patientid(p-receipt(S, PI2))))
                           else p-tube(S, PI2) fi)
                         if c-blood-gather(S, PI1) .

 ceq blood-gather(S, PI) = S
                       if not c-blood-gather(S, PI) .

}
```

付録 証明スコア（手順書モデル）

```
mod INV1 {
 pr(PROCESS)
 op inv1 :  Sys PatientId -> Bool .

 var S : Sys .
 var P : PatientId .

 eq inv1(S, P) = (not (get-blood(p-tube(S, P)) = no-blood))
               implies
              (P = get-patientid(get-blood(p-tube(S, P))) and
               P = get-patientid(get-label(p-tube(S, P)))) .
}

mod ISTEP1{
 pr(INV1)
 ops s s' : -> Sys
 op istep1 : PatientId -> Bool
 var P : PatientId
 eq istep1(P)
          = inv1(s, P) implies inv1(s', P) .
}


mod INV2 {
 pr(ISTEP1)
 op inv2 :  Sys PatientId -> Bool .

 var S : Sys .
 var P : PatientId .

 eq inv2(S, P) = (get-receipt-check(p-order(S, P)) = check-true)
```

```
               implies
                (P = get-patientid(p-receipt(S,P)))  .
}

mod ISTEP2{
 pr(INV2)

 op istep2 : PatientId -> Bool
 var P : PatientId
 eq istep2(P)
          = inv2(s, P) implies inv2(s', P) .
}

mod INV3 {
 pr(ISTEP2)
 op inv3 :  Sys PatientId -> Bool .

 var S : Sys .
 var P : PatientId .

 eq inv3(S, P) = (get-tube-check(p-order(S, P)) = check-true)
               implies
                 (P = get-patientid(get-label(p-tube(S, P))))  .
}

mod ISTEP3{
 pr(INV3)

 op istep3 : PatientId -> Bool
 var P : PatientId
 eq istep3(P)
          = inv3(s, P) implies inv3(s', P) .
}

mod INV4 {
 pr(ISTEP3)
 op inv4 :  Sys PatientId -> Bool .
```

```
 var S : Sys .
 var P : PatientId .

 eq inv4(S, P) = (get-label(p-tube(S, P)) = no-label)
                implies
                  (get-tube-check(p-order(S, P)) = no-check)  .
}

mod ISTEP4{
 pr(INV4)

 op istep4 : PatientId -> Bool
 var P : PatientId
 eq istep4(P)
          = inv4(s, P) implies inv4(s', P) .
}


-- BaseCase
open INV1 .
op p : -> PatientId .
red inv1(init, p) .
close

-- InductionCase

--> Case1 :: order-create
--> Case1-1 :: c-order-create = true

open ISTEP1 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
-- Case1
eq s' = order-create(s, e, p1) .
-- Case1-1
eq p-order(s, p1) = no-order .
red istep1(p2) .
close
```

```
--> Case1-1*******end

--> Case1-2 :: c-order-create =false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
eq s' = order-create(s, e, p1) .
-- Case1-2
eq c-order-create(s, p1) = false .

red istep1(p2) .
close
--> Case1-2*******end
--> Case1*******end

--> Case2 :: card-read
--> Case2-1 :: c-card-read = true
--> Case2-1-1 :: p1 = p2
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case2
eq s' = card-read(s, p1) .
-- Case2-1
eq (p-order(s, p1) = no-order) = false .
eq p-receipt(s, p1) = no-receipt .
eq p-tube(s, p1) = no-tube .
eq p-label(s, p1) = no-label .
-- Case2-1-1
eq p2 = p1 .
red istep1(p2) .
close
--> Case2-1-1*******end


--> Case2-1-2 :: p1 =/= p2
open ISTEP1 .
ops p1 p2 : ->  PatientId .
eq s' = card-read(s, p1) .
```

```
eq (p-order(s, p1) = no-order) = false .
eq p-receipt(s, p1) = no-receipt .
eq p-tube(s, p1) = no-tube .
eq p-label(s, p1) = no-label .
-- Case2-1-2
eq (p2 = p1) = false .
red istep1(p2) .
close
--> Case2-1-2*******end
--> Case2-1*******end

--> Case2-2 :: c-card-read = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
eq s' = card-read(s, p1) .
-- Case2-2
eq c-card-read(s, p1) = false .
red istep1(p2) .
close
--> Case2-2*******end
--> Case2*******end

--> Case3 :: put-label
--> Case3-1 :: c-put-label = true
--> Case3-1-1 :: p1 = p2
--> Case3-1-1-1 :: p1 = get-patientid(no-label)
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
-- Case3-1-1
eq p2 = p1 .
-- Case3-1-1-1
eq p1 = get-patientid(no-label) .
```

```
red istep1(p2) .
close
--> Case3-1-1-1*******end

--> Case3-1-1-2 :: (p1 = get-patientid(no-label)) = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
-- Case3-1-1
eq p2 = p1 .
-- Case3-1-1-2
eq (p1 = get-patientid(no-label)) = false .
red istep1(p2) .
close
--> Case3-1-1-2*******end
--> Case3-1-1*******end

--> Case3-1-2 :: (p1 = p2) = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
-- Case3-1-2
eq (p2 = p1) = false .
red istep1(p2) .
close
--> Case3-1-2*******end
--> Case3-1*******end
```

```
--> Case3-2 :: c-put-label = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-2
eq c-put-label(s, p1) = false .
red istep1(p2) .
close
--> Case3-2*******end
--> Case3*******end

--> Case4 :: receipt-check
--> Case4-1 :: c-receipt-check = true
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-1
eq get-receipt-check(p-order(s, p1)) = no-check .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-tube-check(p-order(s, p1)) = check-true .
red istep1(p2) .
close
--> Case4-1*******end

--> Case4 :: receipt-check
--> Case4-2 :: c-receipt-check = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-2
eq c-receipt-check(s, p1) = false .
red istep1(p2) .
close
--> Case4-2*******end
--> Case4*******end
```

```
--> Case5 :: tube-check
--> Case5-1 :: c-tube-check = true
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
red istep1(p2) .
close
--> Case5-1*******end

--> Case5 :: tube-check
--> Case5-2 :: c-tube-check = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-2
eq c-tube-check(s, p1) = false .
red istep1(p2) .
close
--> Case5-2*******end
--> Case5*******end

--> Case6 :: blood-gather
--> Case6-1 :: c-blood-gather = true
--> Case6-1-1 :: p1 = p2
open INV3 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-blood(p-tube(s, p1)) = no-blood .
eq (p-receipt(s, p1) = no-receipt) = false .
```

```
eq get-receipt-check(p-order(s, p1)) = check-true .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case6-1-1
eq p2 = p1 .
red (inv2(s, p2) and inv3(s, p2)) implies  istep1(p2)  .
close

--> Case6-1-2 :: (p1 = p2) = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-blood(p-tube(s, p1)) = no-blood .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-receipt-check(p-order(s, p1)) = check-true .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case6-1-2
eq (p2 = p1) = false .
red istep1(p2) .
close
--> Case6-1-2*******end
--> Case6-1*******end

--> Case6-2 :: c-blood-gather = false
open ISTEP1 .
ops p1 p2 : ->  PatientId .
-- Case7
eq s' = blood-gather(s, p1) .
-- Case6-2
eq c-blood-gather(s, p1) = false .

red istep1(p2) .
close
--> Case6-2*******end
--> Case6*******end
```

```
eof

-- BaseCase
open INV2 .
op p : -> PatientId .
red inv2(init, p) .
close

-- InductionCase

--> Case1 :: order-create
--> Case1-1 :: c-order-create = true
--> Case1-1-1 :: p1 = p2

open ISTEP2 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
-- Case1
eq s' = order-create(s, e, p1) .
-- Case1-1
eq p-order(s, p1) = no-order .
-- Case1-1-1
eq p2 = p1 .
red istep2(p2) .
close
--> Case1-1-1*******end

--> Case1-1-2 :: p1 =/= p2

open ISTEP2 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
-- Case1
eq s' = order-create(s, e, p1) .
-- Case1-1
eq p-order(s, p1) = no-order .
-- Case1-1-1
eq (p2 = p1) = false .
```

```
red istep2(p2) .
close
--> Case1-1-2*******end
--> Case1-1*******end

--> Case1-2 :: c-order-create =false
open ISTEP2 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
eq s' = order-create(s, e, p1) .
-- Case1-2
eq c-order-create(s, p1) = false .

red istep2(p2) .
close
--> Case1-2*******end
--> Case1*******end

--> Case2 :: card-read
--> Case2-1 :: c-card-read = true
--> Case2-1-1 :: p1 = p2
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case2
eq s' = card-read(s, p1) .
-- Case2-1
eq (p-order(s, p1) = no-order) = false .
eq p-receipt(s, p1) = no-receipt .
eq p-tube(s, p1) = no-tube .
eq p-label(s, p1) = no-label .
-- Case2-1-1
eq p2 = p1 .
red istep2(p2) .
close
--> Case2-1-1*******end


--> Case2-1-2 :: p1 =/= p2
```

```
open ISTEP2 .
ops p1 p2 : ->  PatientId .
eq s' = card-read(s, p1) .
eq (p-order(s, p1) = no-order) = false .
eq p-receipt(s, p1) = no-receipt .
eq p-tube(s, p1) = no-tube .
eq p-label(s, p1) = no-label .
-- Case2-1-2
eq (p2 = p1) = false .
red istep2(p2) .
close
--> Case2-1-2*******end
--> Case2-1*******end

--> Case2-2 :: c-card-read = false
open ISTEP2 .
ops p1 p2 : ->  PatientId .
eq s' = card-read(s, p1) .
-- Case2-2
eq c-card-read(s, p1) = false .
red istep2(p2) .
close
--> Case2-2*******end
--> Case2*******end


--> Case3 :: put-label
--> Case3-1 :: c-put-label = true
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
-- Case3-1-1
red istep2(p2) .
```

```
close
--> Case3-1*******end

--> Case3-2 :: c-put-label = false
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-2
eq c-put-label(s, p1) = false .
red istep2(p2) .
close
--> Case3-2*******end
--> Case3*******end

--> Case4 :: receipt-check
--> Case4-1 :: c-receipt-check = true
--> Case4-1-1 :: p1 = p2
--> Case4-1-1-1 :: get-patientid(p-receipt(s,p1)) = p1
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-1
eq get-receipt-check(p-order(s, p1)) = no-check .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case4-1-1
eq p2 = p1 .
-- Case4-1-1-1
eq get-patientid(p-receipt(s,p1)) = p1 .

red istep2(p2) .
close
--> Case4-1-1-1*******end

--> Case4-1-1-2 :: (get-patientid(p-receipt(s,p1)) = p1) = false
open ISTEP2 .
```

```
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-1
eq get-receipt-check(p-order(s, p1)) = no-check .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case4-1-1
eq p2 = p1 .
-- Case4-1-1-2
eq (get-patientid(p-receipt(s,p1)) = p1) = false .

red istep2(p2) .
close
--> Case4-1-1-2*******end
--> Case4-1-1*******end

--> Case4-1-2 :: p1 =/= p2
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-1
eq get-receipt-check(p-order(s, p1)) = no-check .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case4-1-2
eq (p2 = p1) = false .

red istep2(p2) .
close
--> Case4-1-2*******end

--> Case4 :: receipt-check
--> Case4-2 :: c-receipt-check = false
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case4
```

```
eq s' = receipt-check(s, p1) .
-- Case4-2
eq c-receipt-check(s, p1) = false .
red istep2(p2) .
close
--> Case4-2*******end
--> Case4*******end

--> Case5 :: tube-check
--> Case5-1 :: c-tube-check = true
--> Case5-1-1 :: p1 = p2
--> Case5-1-1-1 :: get-patientid(get-label(p-tube(s,p1))) = p1
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
-- Case5-1-1
eq p2 = p1 .
-- Case5-1-1-1
eq get-patientid(get-label(p-tube(s,p1))) = p1  .

red istep2(p2) .
close
--> Case5-1-1-1*******end

--> Case5-1-1-2 :: (get-patientid(get-label(p-tube(s,p1))) = p1) = false
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
-- Case5-1-1
eq p2 = p1 .
```

```
-- Case5-1-1-2
eq (get-patientid(get-label(p-tube(s,p1))) = p1) = false  .

red istep2(p2) .
close
--> Case5-1-1-2*******end
--> Case5-1-1*******end

--> Case5-1-2 :: p1 =/= p2
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
-- Case5-1-2
eq (p2 = p1) = false .

red istep2(p2) .
close
--> Case5-1-2*******end
--> Case5-1*******end

--> Case5 :: tube-check
--> Case5-2 :: c-tube-check = false
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-2
eq c-tube-check(s, p1) = false .
red istep2(p2) .
close
--> Case5-2*******end
--> Case5*******end

--> Case6 :: blood-gather
```

```
--> Case6-1 :: c-blood-gather = true
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-blood(p-tube(s, p1)) = no-blood .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-receipt-check(p-order(s, p1)) = check-true .
eq get-tube-check(p-order(s, p1)) = check-true .
red istep2(p2)  .
close

--> Case6-2 :: c-blood-gather = false
open ISTEP2 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-2
eq c-blood-gather(s, p1) = false .

red istep2(p2) .
close
--> Case6-2*******end
--> Case6*******end


-- BaseCase
open INV3 .
op p : -> PatientId .
red inv3(init, p) .
close

-- InductionCase

--> Case1 :: order-create
--> Case1-1 :: c-order-create = true
```

```
--> Case1-1-1 :: p1 = p2

open ISTEP3 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
-- Case1
eq s' = order-create(s, e, p1) .
-- Case1-1
eq p-order(s, p1) = no-order .
-- Case1-1-1
eq p2 = p1 .
red istep3(p2) .
close
--> Case1-1-1*******end

--> Case1-1-2 :: p1 =/= p2

open ISTEP3 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
-- Case1
eq s' = order-create(s, e, p1) .
-- Case1-1
eq p-order(s, p1) = no-order .
-- Case1-1-1
eq (p2 = p1) = false .
red istep3(p2) .
close
--> Case1-1-2*******end
--> Case1-1*******end

--> Case1-2 :: c-order-create =false
open ISTEP3 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
eq s' = order-create(s, e, p1) .
-- Case1-2
eq c-order-create(s, p1) = false .
```

```
red istep3(p2) .
close
--> Case1-2*******end
--> Case1*******end


--> Case2 :: card-read
--> Case2-1 :: c-card-read = true
--> Case2-1-1 :: p1 = p2
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case2
eq s' = card-read(s, p1) .
-- Case2-1
eq (p-order(s, p1) = no-order) = false .
eq p-receipt(s, p1) = no-receipt .
eq p-tube(s, p1) = no-tube .
eq p-label(s, p1) = no-label .
-- Case2-1-1
eq p2 = p1 .
red istep3(p2) .
close
--> Case2-1-1*******end


--> Case2-1-2 :: p1 =/= p2
open ISTEP3 .
ops p1 p2 : ->  PatientId .
eq s' = card-read(s, p1) .
eq (p-order(s, p1) = no-order) = false .
eq p-receipt(s, p1) = no-receipt .
eq p-tube(s, p1) = no-tube .
eq p-label(s, p1) = no-label .
-- Case2-1-2
eq (p2 = p1) = false .
red istep3(p2) .
close
```

```
--> Case2-1-2*******end
--> Case2-1*******end

--> Case2-2 :: c-card-read = false
open ISTEP3 .
ops p1 p2 : ->  PatientId .
eq s' = card-read(s, p1) .
-- Case2-2
eq c-card-read(s, p1) = false .
red istep3(p2) .
close
--> Case2-2*******end
--> Case2*******end


--> Case3 :: put-label
--> Case3-1 :: c-put-label = true
--> Case3-1-1 :: p1 = p2
--> Case3-1-1-1 :: (get-tube-check(p-order(s,p1)) = no-check
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
-- Case3-1-1
eq p2 = p1 .
-- Case3-1-1-1
eq get-tube-check(p-order(s,p1)) = no-check .
red istep3(p2) .
close
--> Case3-1-1-1*******end

--> Case3-1-1-2 :: ((get-tube-check(p-order(s,p1)) = no-check) = false
open INV4 .
ops p1 p2 : ->  PatientId .
```

```
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
-- Case3-1-1
eq p2 = p1 .
-- Case3-1-1-2
eq (get-tube-check(p-order(s,p1)) = no-check) = false .

red inv4(s, p2) implies istep3(p2) .
close
--> Case3-1-1-2*******end
--> Case3-1-1*******end

eof

--> Case3-1-2 :: p1 =/= p2
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
-- Case3-1-2
eq (p2 = p1) = false .

red istep3(p2) .
close
--> Case3-1-2*******end

--> Case3-2 :: c-put-label = false
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case3
```

```
eq s' = put-label(s, p1) .
-- Case3-2
eq c-put-label(s, p1) = false .
red istep3(p2) .
close
--> Case3-2*******end
--> Case3*******end

--> Case4 :: receipt-check
--> Case4-1 :: c-receipt-check = true
--> Case4-1-1 :: p1 = p2
--> Case4-1-1-1 :: get-patientid(p-receipt(s,p1)) = p1
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-1
eq get-receipt-check(p-order(s, p1)) = no-check .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case4-1-1
eq p2 = p1 .

red istep3(p2) .
close
--> Case4-1-1*******end

--> Case4-1-2 :: p1 =/= p2
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-1
eq get-receipt-check(p-order(s, p1)) = no-check .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case4-1-2
eq (p2 = p1) = false .
```

```
red istep3(p2) .
close
--> Case4-1-2*******end

--> Case4 :: receipt-check
--> Case4-2 :: c-receipt-check = false
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-2
eq c-receipt-check(s, p1) = false .
red istep3(p2) .
close
--> Case4-2*******end
--> Case4*******end


--> Case5 :: tube-check
--> Case5-1 :: c-tube-check = true
--> Case5-1-1 :: p1 =/= p2
--> Case5-1-1-1 :: get-patientid(get-label(p-tube(s,p1))) = p1
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
eq p2 = p1 .
eq get-patientid(get-label(p-tube(s,p1))) = p1 .
red istep3(p2) .
close
--> Case5-1-1-1*******end

--> Case5-1-1-2 :: (get-patientid(get-label(p-tube(s,p1))) = p1) = false
open ISTEP3 .
```

```
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
eq p2 = p1 .
eq (get-patientid(get-label(p-tube(s,p1))) = p1) = false .
red istep3(p2) .
close
--> Case5-1-1-2*******end
--> Case5-1-1*******end

--> Case5-1-2 :: p1 =/= p2
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
eq (p2 = p1) = false .
red istep3(p2) .
close
--> Case5-1-2*******end
--> Case5-1*******end

--> Case5 :: tube-check
--> Case5-2 :: c-tube-check = false
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-2
eq c-tube-check(s, p1) = false .
red istep3(p2) .
close
--> Case5-2*******end
```

```
--> Case5*******end

--> Case6 :: blood-gather
--> Case6-1 :: c-blood-gather = true
--> Case6-1-1 :: p2 = p1
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-blood(p-tube(s, p1)) = no-blood .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-receipt-check(p-order(s, p1)) = check-true .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case6-1-1
eq p2 = p1 .
red istep3(p2)  .
close

--> Case6-1-2 :: p2 =/= p1
open ISTEP3 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-blood(p-tube(s, p1)) = no-blood .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-receipt-check(p-order(s, p1)) = check-true .
eq get-tube-check(p-order(s, p1)) = check-true .
-- Case6-1-2
eq (p2 = p1) = false .
red istep3(p2)  .
close

--> Case6-2 :: c-blood-gather = false
open ISTEP3 .
```

```
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-2
eq c-blood-gather(s, p1) = false .

red istep3(p2) .
close
--> Case6-2*******end
--> Case6*******end


-- BaseCase
open INV4 .
op p : -> PatientId .
red inv1(init, p) .
close

-- InductionCase

--> Case1 :: order-create
--> Case1-1 :: c-order-create = true
open ISTEP4 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
-- Case1
eq s' = order-create(s, e, p1) .
-- Case1-1
eq p-order(s, p1) = no-order .

red istep4(p2) .
close
--> Case1-1*******end

--> Case1-2 :: c-order-create =false
open ISTEP4 .
ops p1 p2 : ->  PatientId .
op e : -> Examination .
```

```
eq s' = order-create(s, e, p1) .
-- Case1-2
eq c-order-create(s, p1) = false .

red istep4(p2) .
close
--> Case1-2*******end
--> Case1*******end


--> Case2 :: card-read
--> Case2-1 :: c-card-read = true
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case2
eq s' = card-read(s, p1) .
-- Case2-1
eq (p-order(s, p1) = no-order) = false .
eq p-receipt(s, p1) = no-receipt .
eq p-tube(s, p1) = no-tube .
eq p-label(s, p1) = no-label .

red istep4(p2) .
close
--> Case2-1*******end

--> Case2-2 :: c-card-read = false
open ISTEP4 .
ops p1 p2 : ->  PatientId .
eq s' = card-read(s, p1) .
-- Case2-2
eq c-card-read(s, p1) = false .
red istep4(p2) .
close
--> Case2-2*******end
--> Case2*******end

--> Case3 :: put-label
```

```
--> Case3-1 :: c-put-label = true
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-label(p-tube(s, p1)) = no-label .
eq (p-label(s, p1) = no-label) = false .
red istep4(p2) .
close
--> Case3-1*******end

--> Case3-2 :: c-put-label = false
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case3
eq s' = put-label(s, p1) .
-- Case3-2
eq c-put-label(s, p1) = false .
red istep4(p2) .
close
--> Case3-2*******end
--> Case3*******end


--> Case4 :: receipt-check
--> Case4-1 :: c-receipt-check = true
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-1
eq get-receipt-check(p-order(s, p1)) = no-check .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-tube-check(p-order(s, p1)) = check-true .
red istep4(p2) .
close
```

```
--> Case4-1*******end

--> Case4 :: receipt-check
--> Case4-2 :: c-receipt-check = false
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case4
eq s' = receipt-check(s, p1) .
-- Case4-2
eq c-receipt-check(s, p1) = false .
red istep4(p2) .
close
--> Case4-2*******end
--> Case4*******end


--> Case5 :: tube-check
--> Case5-1 :: c-tube-check = true
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-1
eq get-tube-check(p-order(s, p1)) = no-check .
eq (p-tube(s, p1) = no-tube) = false .
red istep4(p2) .
close
--> Case5-1*******end

--> Case5 :: tube-check
--> Case5-2 :: c-tube-check = false
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case5
eq s' = tube-check(s, p1) .
-- Case5-2
eq c-tube-check(s, p1) = false .
red istep4(p2) .
```

```
close
--> Case5-2*******end
--> Case5*******end

--> Case6 :: blood-gather
--> Case6-1 :: c-blood-gather = true
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-1
eq (p-tube(s, p1) = no-tube) = false .
eq get-blood(p-tube(s, p1)) = no-blood .
eq (p-receipt(s, p1) = no-receipt) = false .
eq get-receipt-check(p-order(s, p1)) = check-true .
eq get-tube-check(p-order(s, p1)) = check-true .
red istep4(p2)  .
close

--> Case6-2 :: c-blood-gather = false
open ISTEP4 .
ops p1 p2 : ->  PatientId .
-- Case6
eq s' = blood-gather(s, p1) .
-- Case6-2
eq c-blood-gather(s, p1) = false .

red istep4(p2) .
close
--> Case6-2*******end
--> Case6*******end
```