修 士 論 文

形式手法に基づくビジネスプロセスのリスクリカバリー手法

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

大井　聡史

2011 年 3 月

修 士 論 文

形式手法に基づくビジネスプロセスのリスクリカバリー手法

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　緒方 和博 准教授
審査委員　青木 利晃 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

0810008 大井　聡史

提出年月: 2011 年 2 月

概 要

ビジネスプロセスの形式化を行う。形式化したビジネスプロセスを元にリスクが発生したとしても、コンプライアンスの欠如が起こらないリスクリカバリーの手法を提案する。

目 次

第 1 章　序論 **1**
1.1　はじめに 1
1.2　研究目的 2
1.3　研究内容 2

第 2 章　内部統制 **3**
2.1　内部統制でのフローチャート 3
2.2　内部統制でのリスクとコントロール 6
2.3　リスクコントロールマトリックス 8

第 3 章　ビジネスプロセスの形式化 **11**
3.1　OTS/CafeOBJ 法 11
3.2　OTS/CafeOBJ 法を使ったビジネスプロセスの形式化 13
3.3　受注プロセスの形式化 17

第 4 章　リカバリー手法 **28**
4.1　具体例をもちいたリスクの分類 28
4.2　リスクパターンに対するコントロールパターン 30

第 5 章　実験 **36**

第 6 章　考察 **39**

第 7 章　結論 **40**

第1章　序論

1.1　はじめに

近年、内部統制[1] での不正や粉飾が問題になっている。その不正や粉飾の原因は、リスクと呼ばれるものにある。リスクとは、ビジネスプロセス内での非社会的なアクションを意味する。以下がリスクの例である。

リスク例 ： 注文書を紛失し、売上機会を喪失する。

このリスクは、注文書を紛失すると言うアクションによって注文書を元に品物を出荷するなどのビジネスプロセスでの作業が止まってしまい売上機会を喪失してしまう。これらのリスクが原因でいくつもの事件が起こっており、実際の事件例として、西武鉄道事件やライブドア証券取引法違反事件等がある。この様な内部統制での事件が多発したため、内部統制での不正や粉飾に対する対処が迫られた。そこで、米国の内部統制について定めたSOX法（Public Company Accounting Reform and Investor Protection Act of 2002）を参考に、日本でも金融商品取引法内の内部統制報告書の提出に関する記載を改正を行った。これをJ-SOX法と呼び、2008年（平成20年）4月1日以後に開始する事業年度から全ての企業に適用された [1]。J-SOX法の内容は、企業がリスクに対して信頼性のあるコントロール（社内チェック）を作成し、外部へ報告する事である。コントロールとは、リスクを起こらないように回避する、もしくはリスクが起こったとしてもリスクリカバリーするアクションを意味する。以下が上記のリスクの例に対するコントロールの例である。

コントロール例 ： 注文書の窓口を特定している（FAXや電話番号等）。

このコントロールは、紛失した注文書のクライアントに連絡をとり、再度同じ内容の注文書を再発行しているという事を行っている。以上の様に、リスクやコントロールは具体的なプロセスを書くことができるが、実際には例の通り具体的に書かれておらずリスクやコントロールに対する評価基準が曖昧である。そこで、本研究ではビジネスプロセスの形式化を行い曖昧性を無くし、形式化されたビジネスプロセスからリスクリカバリーの手法を提案する。

[1]内部統制とは、会社に存在する組織、体制、仕組み、手続きの総称である [1]。

1.2 研究目的

リスクリカバリーを考慮したビジネスプロセスの形式化を行う。

1.3 研究内容

ビジネスプロセスを OTS/CafeOBJ 法によって形式化を行う。形式化されたビジネスプロセスを元にリスクのパターンを見つけ出す。パターン化されたリスクから、対応するコントロールを検討する。検討したビジネスプロセスの形式化やリスクのパターン化の手法を用いて、他のビジネスプロセスで実験を行う。実験で得た結果から、リスクパターンに対応するコントロールが有効であるか考察する。

第2章　内部統制

2.1　内部統制でのフローチャート

本研究ではビジネスプロセスの形式化を行うが、形式化を説明する前に基礎知識として、実例を使用して内部統制が構築されるまでの過程を説明する。

内部統制を構築するにあたって、ビジネスプロセスの流れを把握している管理者がビジネスプロセスのフローチャートを作るところからはじまる。ビジネスプロセスのフローチャートの描き方は情報処理でのフローチャートとは異なり、直感的な図示であることが多い。ビジネスプロセスでのフローチャートは企業によって書き方は異なるが、代表的なものは産業能率大学で開発された「産能式」などがある。しかし、本研究では形式的ビジネスプロセスの例として書籍「内部統制の入門と実践」を用いているので、そこで使われているフローチャートを使用する。フローチャートの主な記号とその説明は次のとうりである。

図 2.1: 作業の記号

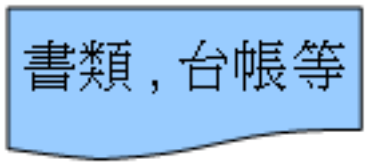

図 2.2: 書類、台帳等の記号

図 2.3: 照合、承認の記号

図 2.4: リスクの記号

図 2.1 はビジネスプロセスでの作業を表す記号である。図 2.2 は書類の作成や送付、台帳の内容の更新を表す記号である。図 2.3 は上長が行う書類の照合や承認を表す記号である。図 2.4 はどの作業でリスクが発生するかを表した記号であり、記号の中には発生するリスクの番号がはいる。

以上の記号を使って受注のビジネスプロセスをフローチャートで表したのが以下の図である。

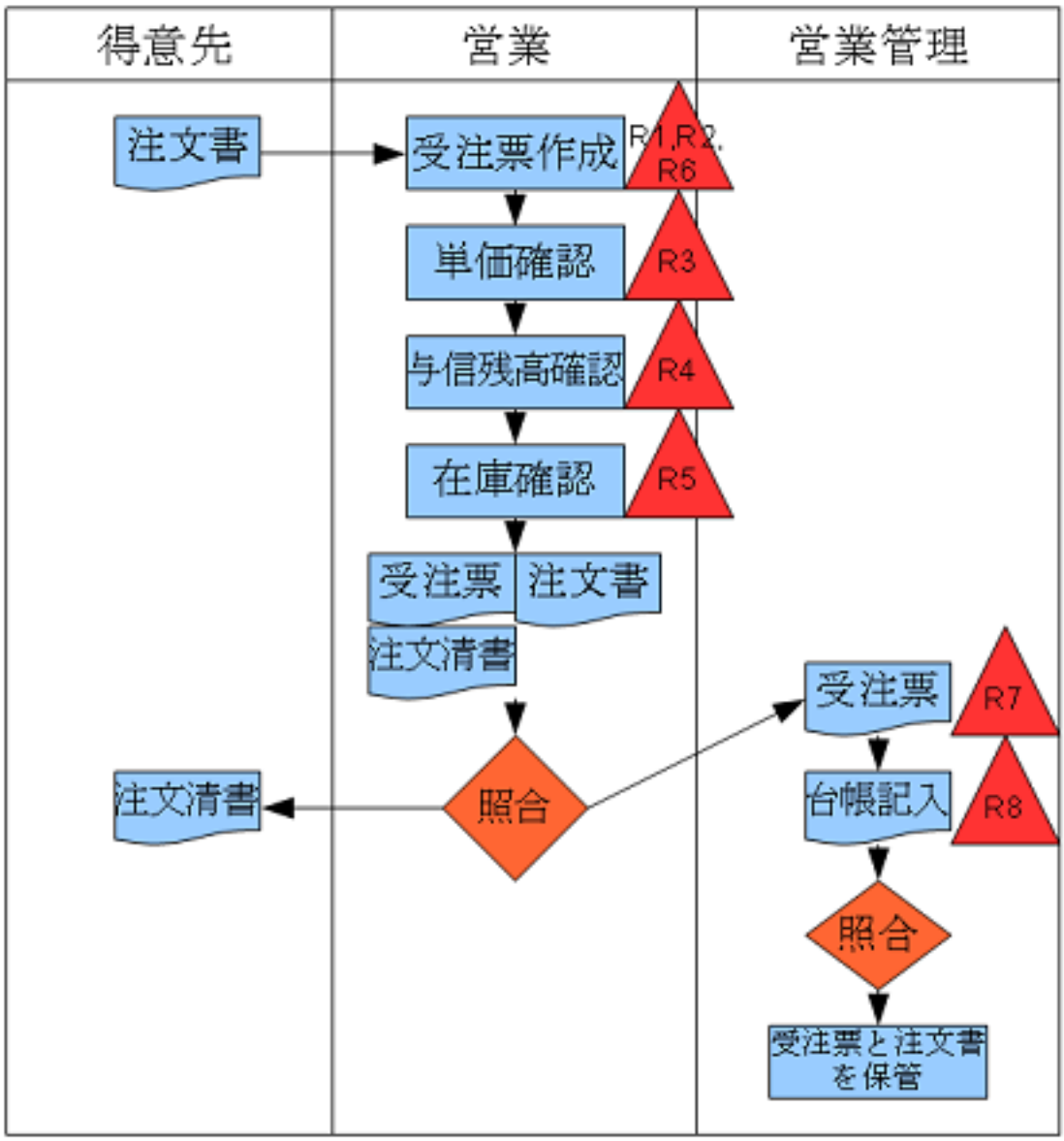

図 2.5: 受注プロセスのフローチャート

図 2.5 のフローチャートに記載されてる作業工程を説明する。はじめに得意先にある注文書を営業へ送付を行う。営業は受け取った注文書を元に受注票の作成を行う。作成された受注票の価格の記載に間違いがないか、単価表を元に単価確認をする。次に与信残高の記載が与信残高限度額を越えている注文を受けてないか顧客台帳、受注台帳、得意先元帳を元に与信残高確認を行う。そして、注文を受けた品物が期限までに得意先へ出荷することができか、商品有高帳を参照しながら在庫確認を行う。これらの確認作業が終わってから注文請書を作成する。作成された、受注票と注文請書、そして得意先から送付された注文書を営業の上長が再確認をおこない、確認した書類にサインを記載する。サインは上長が照合したことを示すものである。次に上長の照合をとおった注文請書は得意先に送付され、注文表と受注票は営業管理へ送付される。営業管理では受け取った注文書と受注票の内容を参照し、受注台帳に記載する。記載された受注台帳の内容を営業管理の上長が記載間違えが無いか照合をおこなう。照合が完了した後に受注票と注文書を営業管理で保管する。ここまでが受注のフローチャートの作業工程の流れである。

2.2 内部統制でのリスクとコントロール

フローチャートの記述が終わると、フローチャートを元に何処でリスクが発生するか考察して書き出す。図 2.5 で起こりえるリスクは以下の 8 個である。

R1 架空の注文を受けるリスク

R2 注文書を紛失し、売り上げを喪失するリスク

R3 誤った単価で注文を受けるリスク

R4 与信限度額を超過した注文を受けるリスク

R5 注文のあった納品の期限に間に合わないリスク

R6 受注票の記入を誤り、異なる受注・出荷が行われるリスク

R7 受注台帳の記入の誤り、異なる受注・出荷が行われるリスク

R8 受注台帳が改ざんされるリスク

上記の各リスクについて説明する。リスク R1，R2，R6 は受注票作成時に発生するリスクである。R1 は架空の注文を受けているので、それを元に架空の内容が記載された受注票を作成してしまう。R2 は注文書を紛失しているので、注文書を元に受注票を作成することができずプロセスがとまってしまう。R6 は受注票を作る際に注文書とは異なる記載内容で作ってしまい、誤った数の品物を出荷してしまう。R3 は誤った単価での注文書を元に受注票を作成をしてしまい、間違った料金を請求してしまう。R4 は与信限度額を越えた注文書を元に受注票を作成してしまい、誤った金額を貸し付けてしまう。R5 は在庫確認する際に商品有高帳の在庫数より注文書の受注数の方が多く、注文書に記載されている受注数を期限内に出荷することができない。R7 は注文書と受注票の内容とは異なる記載を受注台帳にしてしまい、品物の出荷数を間違えて送ってしまう。R8 は受注台帳の内容が改ざんされてしまい、品物の出荷数を間違えて送ってしまう。以上が受注のフローチャートから起こりえるリスクのプロセスである。

リスクの書き出しが終わると、次はリスクに対するコントロールを考える。受注のリスクに対するコントロールは以下のようになる。

C1　注文書の受付窓口を特定している（特定のＦＡＸや電話番号）。

C2　必ず注文請書を発行し、注文内容を相手先に確認する。

C3　受注票や注文請書は連番管理されている。

C4　各種管理台帳（単価表、顧客台帳、注文台帳、得意先元帳など）はロックされており営業担当者が変更不可能となっている。

C5　上長は受注票、注文請書の内容（単価、与信限度、在庫確認など）をレビューし、承認する。

C6　単価を変更する場合や与信限度額を超過した場合には、別途申請、承認手続きが必要となる。

C7　受注台帳は、担当者以外の者が登録内容を照合する。

これらのコントロールはリスクに対して、単数もしくは複数のコントロールの組み合わせによってリスクに対応する。

例えば、R1 架空の受注を受けるリスクに対してのコントロールは以下になる

R1 に対するコントロール：C1 , C2 , C3 , C4 , C5 , C7

リスク R1 に対するコントロールの意味は、C1 の注文書を受けたクライアントの連絡先が特定できてることによって、C2 の注文請書を発行した際に相手が架空受注してないかを確認することができる。C3 の受注票や注文書が連番管理されていることによって過去の受注票や注文書などを確認しやすくなる。C4 の各種管理台帳をロックすることによって不当な改ざんによって、架空受注の注文内容を台帳に記載させない。C5 は作成した注文請書の記載内容に誤った箇所がないか上長がチェックを行い、商品を出荷する被害を防ぐ。C7 は第三者によって受注台帳と受注票の内容に違いが無いか照合をする。以上のコントロールによって、R1 のリスクを防いでいる。

2.3　リスクコントロールマトリックス

リスク１つに対してコントロールは単数、もしくは複数の組み合わせによって対応している。このリスクとコントロールの対応関係を表形式で記述したものをリスクコントロールマトリックスと呼ぶ。リスクとコントロールの書き出し作業が終わると、次はリスクコントロールマトリックスを作る。

次の表 2.1 は受注のリスクコントロールマトリックスである。

リスク	関連勘定	アサーション	コントロール
R1	売上、売掛金	発生、実在性、権利、義務	C1,C2,C3,C4,C5,C7
R2	売上、売掛金	該当なし	C1
R3	売上、売掛金	発生、実在性、権利、義務	C2,C4,C5,C6
R4	売上、売掛金	評価	C4,C5,C6
R5	売上、売掛金	該当なし	C2,C4,C5
R6	売上、売掛金	発生、実在性、権利、義務	C2,C5
R7	売上、売掛金	発生、実在性、権利、義務	C7
R8	売上、売掛金	発生、実在性、権利、義務	C4,C7

表 2.1: 受注のリスクコントロールマトリックス

リスクコントロールマトリックスにはリスクとコントロール以外に関連勘定とアサーションと言う項目がある。勘定鑑定とは、その項目のリスクが影響を与える決算書の科目のことである。アサーション（監査要点）とは、リスクそれぞれの要点のことである。この要点は企業会計審議会が公表している「監査基準」によると 6 つ存在する [3]。6 つの要点と内容は下記の表 2-2 に示す。

監査要点	要点内容
発生・実在性	資産、負債もしくは取引が発生・実在ていること
網羅性	資産、負債もしくは取引が網羅的にすべて計上されている。
権利と義務の帰属	資産に対する権利と負債に関する義務が会社に帰属していること
評価の妥当性	資産と負債が会計基準に従って適切に評価されていること
表示の妥当性	報告方式や表示科目が妥当であること
期間帰属の適正性	取引が適切な報告期間に計上されていること

表 2.2: アサーション表

発生・実在性のアサーションで考えられるリスクは、資産、負債もしくは取引が発生・実在していないリスクである。例として、架空取引など発生していない取引をしてしまうリスクなどである。　網羅性のアサーションで考えられるリスクは、資産、負債もしくは取引に漏れがあるリスクである。例として、簿外取引など記載内容に漏れがあるリスクなどである。　権利と義務の帰属のアサーションで考えれるリスクは、資産と負債が会社の法的な権利・義務を反映していないリスクである。例として資産の過大計上や負債の過小計上をしてしまうリスクなどである。　評価の妥当性のアサーションで考えられるリ

スクは、資産と負債が会計基準に従って評価されていないリスクである。例えば不適切な評価額を与えてしまうリスクなどがある。　表示の妥当性のアサーションで考えられるリスクは、誤った報告方式、勘定科目による情報開示がされるリスクである。　期間帰属の適正性のアサーションで考えられるリスクは、報告期間のズレ、タイミングの遅れが発生するリスクである。リスクはアサーションが単数もしくは複数ある場合と該当しない場合がある。以上がリスクコントロールマトリックスの項目の説明である。

　そして、フローチャートからリスクコントロールマトリックスを作るまでが内部統制を構築するための過程である。

第3章　ビジネスプロセスの形式化

3.1　OTS/CafeOBJ 法

本研究では、ビジネスプロセスのフローチャートとリスクコントロールマトリックスを元に OTS/CafeOBJ 法を用いてビジネスプロセスの形式化を行う。その為、OTS/CafeOBJ 法の説明を簡単な例を用いて説明する。
OTS/CafeOBJ 法は代数仕様言語である CafeOBJ 言語を用いて、状態と遷移によって数学的モデルを作成する手法である [2]。OTS/CafeOBJ 法では状態を観測することによって得られる情報をもとにモデルの振る舞いを記述する。
例えば、以下の状態遷移図のような状態遷移機械を考える。

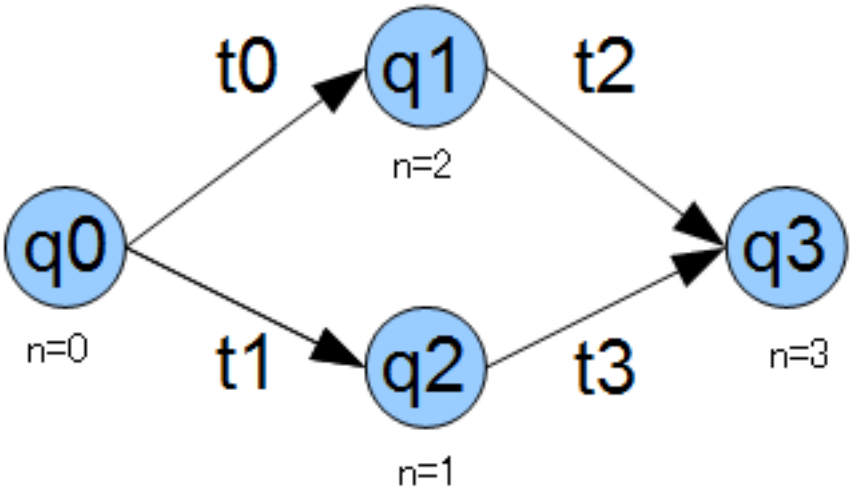

図 3.1: 状態遷移図

q0,q1,q2,q3 を状態として、t0,t1,t2,t3 を遷移とし、n は各状態の保有する自然数とする。そして、自然数を状態の観測値として遷移によって観測値が移り変わると考える。

この時に、各関数を以下の様に定義できる。

観測関数
op outnat : State -> Nat
状態を引数としてとり、状態の保持している自然数を返す関数

初期関数
op init : -> State
何も引数としてとらず、状態を返す初期関数

遷移関数
op t0 : State -> State
op t1 : State -> State
op t2 : State -> State
op t3 : State -> State
各遷移関数は状態を引数としてり、状態を返す関数

これらの関数を使って図 3-1 の振る舞いを以下のように記述できる

var S : State
var N : Nat

– 初期状態定義
eq outnat(init) = 0 .

– t0 関数
op c-t0 : State -> Bool
eq c-t0(S) = (outnat(S) = 0) .
ceq outnat(t0(S)) = 2 if c-t0 .

– t1 関数
op c-t1 : State -> Bool
eq c-t1(S) = (outnat(S) = 0) .
ceq outnat(t1(S)) = 1 if c-t1 .

– t2 関数
op c-t2 : State -> Bool
eq c-t2(S) = (N = 2) .
ceq outnat(t2(S)) = 3 if c-t2 .

– t3 関数
op c-t3 : State -> Bool
eq c-t3(S) = (N = 1) .
ceq outnat(t1(S)) = 3 if c-t3 .

S は状態の変数、N は自然数の変数として定義している。初期状態は n = 0 なので、自然数の観測関数である outnat 関数は init を引数として取る時は 0 を返す。t0 関数で定義されている c-t0 とは、状態が t0 関数で遷移するための条件であり、この場合 n = 0 の時に遷移することができる。また、この条件を効力条件と呼ぶ。t0 関数の効力条件を満たす場合、自然数の観測値を n = 2 に変化させる。以下の遷移関数も同様に、効力条件を満たせば観測値が変化する。以上の様に、OTS/CafeOBJ 法は初期関数、観測関数、遷移関数を定義し、定義された各関数によってモデルの振る舞いを記述する事ができる。
本研究で OTS/CafeOBJ 法を用いたのは、CafeOBJ 言語は等式による証明をすることができる為である。今回、形式化したビジネスプロセスの形式的検証は行っていないが、今後、形式化したビジネスプロセスの形式的検証を行うことによって信頼性を高める為に OTS/CafeOBJ 法を用いる。

3.2 OTS/CafeOBJ 法を使ったビジネスプロセスの形式化

OTS/CafeOBJ 法でのビジネスプロセスを形式化かするにあたって、各型や関数の定義と振る舞いを記述する上での規則や前提条件を説明する。

以下は状態と遷移の定義である。

状態 : ドキュメント（書類、台帳）の場所、存在の有無、内容
遷移 : ドキュメントに対する作業

状態をドキュメントの置いてある場所や、存在の有無、記述内容によって定義し、遷移をドキュメントに対する作業とする。この状態と遷移定義の理由は、ビジネスプロセスの作業とリスクの大部分はドキュメントに関するものである。よって、ドキュメントに着目することによってビジネスプロセスの大部分の振る舞いが記述でき、リスクの考察も容易にする為である。

リスクの前提条件を下記にしめす。

・リスクはドキュメントに着目している為、ドキュメントに対するリスクであること

・上長はリスクが発生するようなアクションはとらない

ビジネスプロセスの形式化で用いる型は以下に定義する。

State：State は状態を表す型である

Document：　Document は書類や台帳などのドキュメントを表す型である

Place：Place はビジネスプロセス内での場所を表す型である

Risk：Risk はリスクをあらわす型である

NAT：NAT は自然数を表した型である

BOOL：BOOL はブール値をあらわした型である

初期関数は以下に定義する。

op init：-> State

状態の観測関数は以下の様に定義する。

– 書類の位置
op place：State Document -> Place

place 関数は状態とドキュメントを引数として取り、引数でとったドキュメントがどの場所にあるかという Place 型を返す。

– 書類の内容
op contents：State Document -> Nat

contents 関数は状態とドキュメントを引数として取り、ドキュメントの記載内容をあらわしている NAT 型を返す。

– 書類の社内に存在
op exist : State Document -> Bool

exist 関数は状態とドキュメントを引数として取り、ドキュメントが存在する場合は true を返し、ドキュメントが存在しない場合は false を返す関数である。

– データの比較などのチェック
op check : State Document Document -> Bool

check 関数は状態とドキュメントとドキュメントを引数として取り、ドキュメントとドキュメントの contents の観測値が同じ場合に true を返し、異なる場合は false を返す。

– その課の上長に許可を得たドキュメント
op check-chief : State Document Place -> Bool

check-chief 関数は状態とドキュメント、場所を引数として取り、ドキュメントが引数でとった場所の上長に許可や照合などの印をもらっているならば true を返し、もらっていないならば false を返す関数である。

実験用の関数として以下の 2 つの関数を定義する。

– 最終状態であるか
op final : State -> Bool

final 関数は状態を引数としてとり、その状態が最終状態であれば true を返し、最終状態でなければ false を返す関数である。

– リスクの有無
op risk : State Risk -> Bool

risk 関数は状態とリスクを引数として取り、引数としてとったリスクが起こってる場合に true を返し、リスクが起こってない場合は false を返す関数である。

遷移関数の定義は、ビジネスプロセスによって多種あるので、次節で具体例をだして説明する。

以上が型や各関数の定義である。以下では定義した関数を使う上での規則である。

初期化の規則は以下になる。

形式化するビジネスプロセスのドキュメントが初期状態に存在する場合
exist(init,D) = true $\wedge$ not(place(init,D) = none) $\wedge$ contents(S,D) = 1

形式化するビジネスプロセスのドキュメントが初期状態に存在しない場合
exist(init,D) = false $\wedge$ place(init,D) = none $\wedge$ contents(S,D) =0

check-chief(init,D,P) = false .

check(init,D) = false

観測関数の規則は以下になる

place(S,D) = none の時、遷移関数によって place(S,D)= none ではなくなった場合、place(S,D) はどの様な遷移関数を使用しても place(S,D) = none とならないとする。

check(S,D) = true の時、どの様な遷移関数を使用しても check(S,D) = false とならないとする。

check-chief(S,D) = true の時、どの様な遷移関数を使用しても check-chief(S,D) = false とならないとする。

以上の定義、前提条件、規則を用いてビジネスプロセスの形式化を行う。

3.3　受注プロセスの形式化

前節ではビジネスプロセスの定義、前提条件、規則を説明したが、それらを用いて具体的なビジネスプロセスの形式化を説明する。具体的な例として 2 章で例として取り上げた受注プロセスを形式化する。以下は受注プロセスのフローチャートを元にドキュメントの作業に着目し、状態遷移図で表したものである。

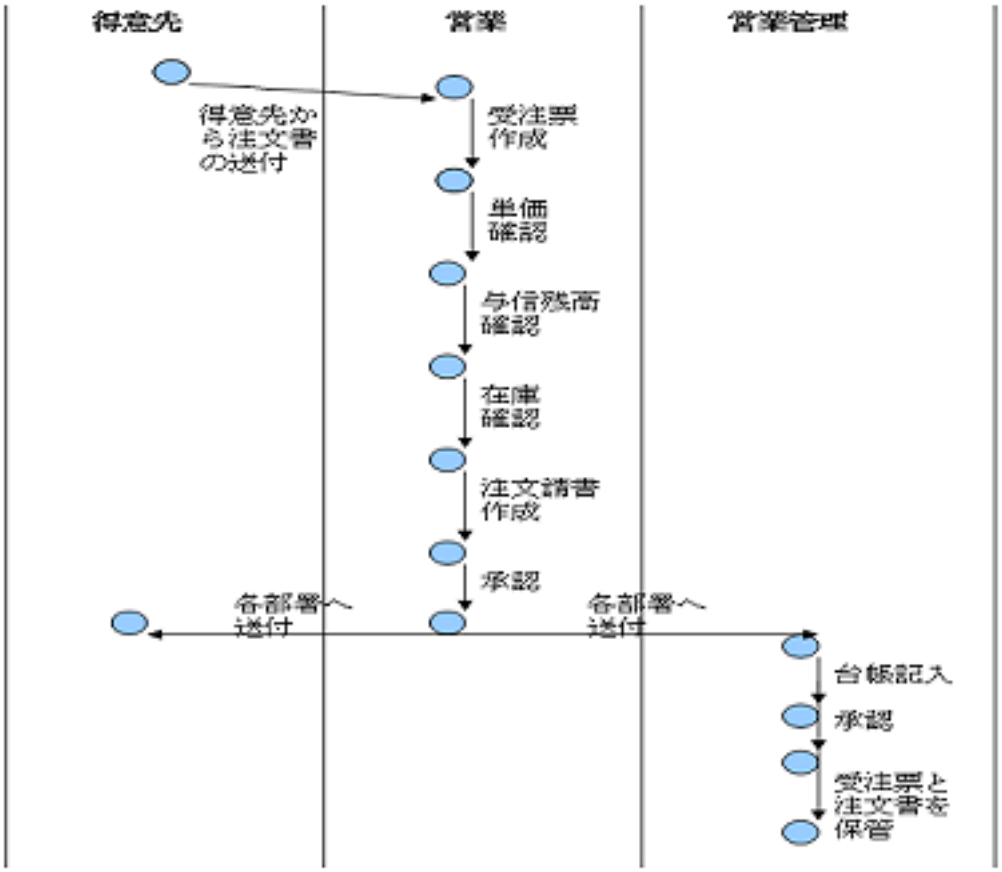

図 3.2: 受注プロセスの状態遷移図

受注プロセスの状態遷移図の振る舞いを形式化していく。

受注プロセスのドキュメントの移動する場所は、得意先、営業、営業管理となる。よって Place の種類は以下のように定義する。

ops client sales salsemanagement archive none : -> Place .

client は得意先、sales は営業、 salsemanagement は営業管理、archive は書類を保管するための倉庫、none はどの場所でもない事を示している。

そして、受注プロセスで扱うドキュメントは注文書、受注票、単価表、顧客台帳、受注台帳、得意先元帳、商品有高合帳、注文請書である。よって Document の種類は下記のように定義する。

ops order ordervote price clientbook orderbook customerbook productbook ack chiefbook: -> Document .

order は注文書、 ordervote は受注票、price は単価表、clientbook は顧客台帳、orderbook は受注台帳、 customerbook は得意先元帳、productbook は商品有高合帳、ack は注文請書、chiefbook はドキュメントを比較する際に上長が独自の知識で確認してる場合に比較するためのドキュメントを示す。

遷移関数は以下の様に定義する。

– 注文書を送付
op send-order : State Document -> State

– 受注票作成
op create-ordervote : State Document -> State

– 単価確認
op price-confirmation : State Document -> State

– 与信残高確認
op credit-confirmation : State Document -> State

– 在庫確認
op availability-confirmation : State Document -> State

– 注文請書作成
op create-ack : State Document -> State

– 注文請書のチェック
op check-ack : State Document -> State

– 受注票、注文書、注文請書の冗長の承認
op check-document : State Document Document Document -> State

– 各場所に、注文書、受注票、注文請書を送付
op send-ack-order-vote : State Document Document Document -> State

– 受注台帳の更新
op ledger-update : State Document -> State

– 台帳の確認
op check-book : State Document -> State

– 台帳のチェック
op checkchief-book : State Document -> State

– 受注票と注文書の保管
op storage : State Document Document -> State

定義した、遷移関数の振る舞いは以下になる。

- 注文書を送付する遷移関数
op c-send-order : State Document -> Bool
eq c-send-order(S,D) = (D = order) and (place(S,D) = client) and exist(S,D) .
ceq send-order(S,D) = S if not c-send-order(S,D) .
ceq place(send-order(S,D),order) = sales if c-send-order(S,D) .
ceq place(send-order(S,D),D1) = place(S,D1) if c-send-order(S,D) .
ceq contents(send-order(S,D),D1) = contents(S,D1) if c-send-order(S,D) .
ceq exist(send-order(S,D),D1) = exist(S,D1) if c-send-order(S,D) .
ceq check(send-order(S,D),D1,D2) = check(S,D1,D2) if c-send-order(S,D) .
ceq check-chief(send-order(S,D),D1,P) = check-chief(S,D1,P) if c-send-order(S,D) .

受注プロセスは始めに注文書を営業に送付する。そこで、効力条件として送付されるドキュメントは order であり、order の送付される前の場所は client である。この条件を満たしている状態ならば、send-order 関数で遷移することができる。send-order 関数は order は client から slaes へ送付するので、観測値として place が client から sales へ変化する。

– 受注票作成する遷移関数 op c-create-ordervote : State Document -> Bool
eq c-create-ordervote(S,D) = (D = ordervote) and not(exist(S,D)) and (place(S,D) = none) and exist(S,order) and (place(S,order) = sales) .
ceq create-ordervote(S,D) = S if not c-create-ordervote(S,D) .
ceq place(create-ordervote(S,D),ordervote) = sales if c-create-ordervote(S,D) .
ceq place(create-ordervote(S,D),D1) = place(S,D1) if c-create-ordervote(S,D) .
ceq contents(create-ordervote(S,D),ordervote) = contents(S,order) if c-create-ordervote(S,D) .
ceq contents(create-ordervote(S,D),D1) = contents(S,D1) if c-create-ordervote(S,D) .
ceq exist(create-ordervote(S,D),ordervote) = true if c-create-ordervote(S,D) .
ceq exist(create-ordervote(S,D),D1) = exist(S,D1) if c-create-ordervote(S,D) .
ceq check(create-ordervote(S,D),D1,D2) = check(S,D1,D2) if c-create-ordervote(S,D) .
ceq check-chief(create-ordervote(S,D),D1,P) = check-chief(S,D1,P) if c-create-ordervote(S,D) .

create-ordervote 関数は送付された注文書を元に受注表を作成する。効力条件として、作られるドキュメントは ordervote であり、ordervote はまだ作られていないので exist(S,ordervote) は false であり、place(S,ordervote) = none となる、そして order は exist(S,order) が true であり、送付された後なので place(S,order) は sales となる。条件を満たす場合、create-ordervote 関数は受注表を作成するので、contents,exist,place の観測値が変化する。

– 単価確認する遷移関数

op c-price-confirmation : State Document -> Bool
eq c-price-confirmation(S,D) = (D = ordervote) and (check(S,D,price) = false) and (contents(S,D) = contents(S,price)) and exist(S,D) and exist(S,order) .
ceq price-confirmation(S,D) = S if not c-price-confirmation(S,D) .
ceq place(price-confirmation(S,D),D1) = place(S,D1) if c-price-confirmation(S,D) .
ceq contents(price-confirmation(S,D),D1) = contents(S,D1) if c-price-confirmation(S,D) .
ceq exist(price-confirmation(S,D),D1) = exist(S,D1) if c-price-confirmation(S,D) .
ceq check(price-confirmation(S,D),ordervote,price) = true if c-price-confirmation(S,D) .
ceq check(price-confirmation(S,D),D1,D2) = check(S,D1,D2) if c-price-confirmation(S,D)
.
ceq check-chief(price-confirmation(S,D),D1,P) = check-chief(S,D1,P) if c-price-confirmation(S,D)
.

price-confirmation 関数は単価確認をする遷移である。効力条件は確認するドキュメントは ordervote であること、ordervote の確認はまだされてないので check(S,ordervote,price) = false、ordervote と price の記載内容が一緒であること、そして比較した各ドキュメントが存在することである。この条件を満たす場合、単価確認がされたので check(S,ordervote,price) = true となる。

– 与信残高確認する遷移関数

op c-credit-confirmation : State Document -> Bool
eq c-credit-confirmation(S,D) = (D = ordervote) and exist(S,D) and exist(S,order) and not(check(S,D,clientbook)) and not(check(S,D,orderbook)) and not(check(S,D,customerbook)) and (contents(S,D) = contents(S,clientbook)) and (contents(S,D) = contents(S,orderbook)) and (contents(S,D) = contents(S,customerbook)) .
ceq credit-confirmation(S,D) = S if not c-credit-confirmation(S,D) .
ceq place(credit-confirmation(S,D),D1) = place(S,D1) if c-credit-confirmation(S,D) .
ceq contents(credit-confirmation(S,D),D1) = contents(S,D1) if c-credit-confirmation(S,D)
.
ceq exist(credit-confirmation(S,D),D1) = exist(S,D1) if c-credit-confirmation(S,D) .
ceq check(credit-confirmation(S,D),ordervote,clientbook) = true if c-credit-confirmation(S,D)
.
ceq check(credit-confirmation(S,D),ordervote,orderbook) = true if c-credit-confirmation(S,D)
.

ceq check(credit-confirmation(S,D),ordervote,customerbook) = true if c-credit-confirmation(S,D)
.
ceq check(credit-confirmation(S,D),D1,D2) = check(S,D1,D2) if c-credit-confirmation(S,D)
.
ceq check-chief(credit-confirmation(S,D),D1,P) = check-chief(S,D1,P) if c-credit-confirmation(S,D)
.

credit-confirmation 関数は与信残高を確認する遷移である。効力条件は確認するドキュメントは ordervote であること、check(S,D,orderbook)) と check(S,D,customerbook）はまだ与信残高確認がされていないので false であること、ordervote と orderbook、customerbook の記載内容が一緒であること、そして各ドキュメントが存在する。この条件を満たす場合、check(S,D,orderbook)) と check(S,D,customerbook）を true にする。

– 在庫確認する遷移関数

op c-availability-confirmation : State Document -> Bool
eq c-availability-confirmation(S,D) = (D = ordervote) and exist(S,D) and exist(S,order) and not(check(S,D,productbook)) and (contents(S,D) = contents(S,productbook)) .
ceq availability-confirmation(S,D) = S if not c-availability-confirmation(S,D) .
ceq place(availability-confirmation(S,D),D1) = place(S,D1) if c-availability-confirmation(S,D)
.
ceq contents(availability-confirmation(S,D),D1) = contents(S,D1) if c-availability-confirmation(S,D)
.
ceq exist(availability-confirmation(S,D),D1) = exist(S,D1) if c-availability-confirmation(S,D)
.
ceq check(availability-confirmation(S,D),ordervote,productbook) = true if c-availability-confirmation(S,D) .
ceq check(availability-confirmation(S,D),D1,D2) = check(S,D1,D2) if c-availability-confirmation(S,D)
.
ceq check-chief(availability-confirmation(S,D),D1,P) = check-chief(S,D1,P) if c-availability-confirmation(S,D) .

availability-confirmation 関数は在庫確認をする遷移である。効力条件はドキュメントは ordervote であること、まだ在庫確認がされていないので check(S,D,productbook) が false であること、ordervote と productbook の記載内容が一緒であること、そして各種ドキュメントが存在すること。この条件を満たす場合、check(S,D,productbook) が true となる。

– 注文請書作成する遷移関数

op c-create-ack : State Document -> Bool
eq c-create-ack(S,D) = (D = ack) and not(exist(S,D)) and exist(S,order) and exist(S,ordervote) and (place(S,D) = none).
ceq create-ack(S,D) = S if not c-create-ack(S,D) .
ceq place(create-ack(S,D),ack) = sales if c-create-ack(S,D) .
ceq place(create-ack(S,D),D1) = place(S,D1) if c-create-ack(S,D) .
ceq contents(create-ack(S,D),ack) = 1 if c-create-ack(S,D) .
ceq contents(create-ack(S,D),D1) = contents(S,D1) if c-create-ack(S,D) .
ceq exist(create-ack(S,D),ack) = true if c-create-ack(S,D) .
ceq exist(create-ack(S,D),D1) = exist(S,D1) if c-create-ack(S,D) .
ceq check(create-ack(S,D),D1,D2) = check(S,D1,D2) if c-create-ack(S,D) .
ceq check-chief(create-ack(S,D),D1,P) = check-chief(S,D1,P) if c-create-ack(S,D) .

create-ack 関数は注文請書を作成する遷移である。効力条件はドキュメントが ack であり、ack はまだ作られていないこと、そして order と ordervote が存在している。条件を満たす場合、ack の exist,place,contents の観測値が変化する。

– 注文請書の確認する遷移関数

op c-check-ack : State Document -> Bool
eq c-check-ack(S,D) = (D = ack) and exist(S,D) and not(check-chief(S,D,sales)) and not(check(S,D,ordervote)) .
ceq check-ack(S,D) = S if not c-check-ack(S,D) .
ceq place(check-ack(S,D),D1) = place(S,D1) if c-check-ack(S,D) .
ceq contents(check-ack(S,D),D1) = contents(S,D1) if c-check-ack(S,D) .
ceq exist(check-ack(S,D),D1) = exist(S,D1) if c-check-ack(S,D) .
ceq check(check-ack(S,D),ack,ordervote) = true if c-check-ack(S,D) .
ceq check(check-ack(S,D),D1,D2) = check(S,D1,D2) if c-check-ack(S,D) .
ceq check-chief(check-ack(S,D),D1,P) = check-chief(S,D1,P) if c-check-ack(S,D) .

check-ack 関数は注文請書に記載間違えがないか確認する遷移である。効力条件はドキュメントは ack であり、check-chief(S,D,sales) と check(S,D,ordervote) は確認がおわっていないので false、そして各種ドキュメントが存在してる。条件を満たす場合、check-chief(S,D,sales) と check(S,D,ordervote) は true になる。

– D 注文書、D1 受注票、D2 注文請書を冗長がチェックする遷移関数

```
op c-check-document : State Document Document Document -> Bool
eq c-check-document(S,D,D1,D2) = (D = order) and (D1 = ordervote) and (D2 = ack) and
exist(S,D) and exist(S,D1) and exist(S,D2) and not(check-chief(S,D,sales)) and not(check-
chief(S,D1,sales)) and not(check-chief(S,D2,sales))
and check(S,ordervote,price) and check(S,ordervote,clientbook) and check(S,ordervote,orderbook)
and check(S,ordervote,customerbook) and check(S,ordervote,productbook) and check(S,D2,ordervote)
.
ceq check-document(S,D,D1,D2) = S if not c-check-document(S,D,D1,D2) .
ceq place(check-document(S,D,D1,D2),D3) = place(S,D3) if c-check-document(S,D,D1,D2)
.
ceq contents(check-document(S,D,D1,D2),D3) = contents(S,D3) if c-check-document(S,D,D1,D2)
.
ceq exist(check-document(S,D,D1,D2),D3) = exist(S,D3) if c-check-document(S,D,D1,D2)
.
ceq check(check-document(S,D,D1,D2),D3,D4) = check(S,D3,D4) if c-check-document(S,D,D1,D2)
.
ceq check-chief(check-document(S,D,D1,D2),order,sales) = true if c-check-document(S,D,D1,D2)
.
ceq check-chief(check-document(S,D,D1,D2),ordervote,sales) = true if c-check-document(S,D,D1,D2)
.
ceq check-chief(check-document(S,D,D1,D2),ack,sales) = true if c-check-document(S,D,D1,D2)
.
ceq check-chief(check-document(S,D,D1,D2),D3,P) = check-chief(S,D3,P)
if c-check-document(S,D,D1,D2) .
```

check-document 関数は注文書、受注票、注文請書をチェックする遷移である。効力条件はドキュメントは order,ordervote,ack であり、各種ドキュメントは sales の上長のチェックをうけていないこと、各種ドキュメントは台帳と照合をしていない、そして、各種ドキュメントは存在する。条件を満たす場合、各種ドキュメントは sales の上長にチェックを受けたので check-chief(S,D,sales) の観測値が true になる。

– 各場所に、D 注文書、D1 受注票、D2 注文請書を送付する遷移関数

op c-send-ack-order-vote : State Document Document Document -> Bool

eq c-send-ack-order-vote(S,D,D1,D2) = (D = order) and (D1 = ordervote) and (D2 = ack) and (place(S,D) = sales) and (place(S,D1) = sales) and (place(S,D2) = sales) and exist(S,D) and exist(S,D1) and exist(S,D2) and check-chief(S,D,sales) and check-chief(S,D1,sales) and check-chief(S,D2,sales) .

ceq send-ack-order-vote(S,D,D1,D2) = S if not c-send-ack-order-vote(S,D,D1,D2) .

ceq place(send-ack-order-vote(S,D,D1,D2),order) = salsemanagement if c-send-ack-order-vote(S,D,D1,D2) .

ceq place(send-ack-order-vote(S,D,D1,D2),ordervote) = salsemanagement if c-send-ack-order-vote(S,D,D1,D2) .

ceq place(send-ack-order-vote(S,D,D1,D2),ack) = client if c-send-ack-order-vote(S,D,D1,D2) .

ceq place(send-ack-order-vote(S,D,D1,D2),D3) = place(S,D3) if c-send-ack-order-vote(S,D,D1,D2) .

ceq contents(send-ack-order-vote(S,D,D1,D2),D3) = contents(S,D3) if c-send-ack-order-vote(S,D,D1,D2) .

ceq exist(send-ack-order-vote(S,D,D1,D2),D3) = exist(S,D3) if c-send-ack-order-vote(S,D,D1,D2) .

ceq check(send-ack-order-vote(S,D,D1,D2),D3,D4) = check(S,D3,D4) if c-send-ack-order-vote(S,D,D1,D2) .

ceq check-chief(send-ack-order-vote(S,D,D1,D2),D3,P) = check-chief(S,D3,P) if c-send-ack-order-vote(S,D,D1,D2) .

send-ack-order-vote 関数は注文書、受注票を営業管理へ、注文請書を得意先へ送付する遷移である。効力条件はドキュメントは order,ordervote,ack であること、そして sales の上長のチェックを受けており、各種ドキュメントは sales にあり、各種ドキュメントは存在する。条件を満たす場合、order,ordervote の場所は salsemanagement になり、ack は client に変化する。

– 受注台帳の更新をする遷移関数

op c-ledger-update : State Document -> Bool

eq c-ledger-update(S,D) = (D = orderbook) and check-chief(S,order,sales) and check-chief(S,ordervote,sales) and (place(S,ack) = client) and (place(S,order) = salsemanagement) and (place(S,ordervote) = salsemanagement) and exist(S,order) and exist(S,ordervote) .

ceq ledger-update(S,D) = S if not c-ledger-update(S,D) .

ceq place(ledger-update(S,D),D1) = place(S,D1) if c-ledger-update(S,D) .
ceq contents(ledger-update(S,D),orderbook) = contents(S,orderbook) + contents(S,ordervote) if c-ledger-update(S,D) .
ceq contents(ledger-update(S,D),D1) = contents(S,D1) if c-ledger-update(S,D) .
ceq exist(ledger-update(S,D),D1) = exist(S,D1) if c-ledger-update(S,D) .
ceq check(ledger-update(S,D),D1,D2) = check(S,D1,D2) if c-ledger-update(S,D) .
ceq check-chief(ledger-update(S,D),D1,P) = check-chief(S,D1,P) if c-ledger-update(S,D) .

ledger-update 関数は受注票と注文書を元に受注台帳を更新する遷移である。効力条件はドキュメントは orderbook である、各種ドキュメントは sales の上長にチェックを受けており、order,ordervote の場所は salsemanagement にあり、各種ドキュメントは存在する。条件を満たす場合、orderbook の内容は更新される。

– 台帳の確認をする遷移関数
op c-check-book : State Document -> Bool
eq c-check-book(S,D) = (D = orderbook) and exist(S,order) and exist(S,ordervote) and not(check(S,D,chiefbook)) and not(check-chief(S,D,salsemanagement)) and (contents(S,D) = 2) .
ceq check-book(S,D) = S if not c-check-book(S,D) .
ceq place(check-book(S,D),D1) = place(S,D1) if c-check-book(S,D) .
ceq contents(check-book(S,D),D1) = contents(S,D1) if c-check-book(S,D) .
ceq exist(check-book(S,D),D1) = exist(S,D1) if c-check-book(S,D) .
ceq check(check-book(S,D),orderbook,chiefbook) = true if c-check-book(S,D) .
ceq check(check-book(S,D),D1,D2) = check(S,D1,D2) if c-check-book(S,D) .
ceq check-chief(check-book(S,D),D1,P) = check-chief(S,D1,P) if c-check-book(S,D) .

check-book 関数は更新した受注台帳の内容を確認する遷移である。効力条件はドキュメントは orderbook であり、 salsemanagement の上長の確認とチェックをうけていない、orderbook は order,ordervote の内容を更新した記載内容であり、各種台帳は存在する。条件を満たす場合、orderbook は salsemanagement の上長の確認をうける。

– 上長が台帳を承認する遷移関数
op c-checkchief-book : State Document -> Bool
eq c-checkchief-book(S,D) = (D = orderbook) and exist(S,order) and exist(S,ordervote) and check(S,D,chiefbook) and not(check-chief(S,D,salsemanagement)) .
ceq checkchief-book(S,D) = S if not c-checkchief-book(S,D) .

ceq place(checkchief-book(S,D),D1) = place(S,D1) if c-checkchief-book(S,D) .
ceq contents(checkchief-book(S,D),D1) = contents(S,D1) if c-checkchief-book(S,D) .
ceq exist(checkchief-book(S,D),D1) = exist(S,D1) if c-checkchief-book(S,D) .
ceq check(checkchief-book(S,D),D1,D2) = check(S,D1,D2) if c-checkchief-book(S,D) .
ceq check-chief(checkchief-book(S,D),orderbook,salsemanagement) = true if c-checkchief-book(S,D) .
ceq check-chief(checkchief-book(S,D),D1,P) = check-chief(S,D1,P) if c-checkchief-book(S,D) .

checkchief-book 関数は更新した受注台帳の内容をチェックして、確認の印をつける遷移である。効力条件はドキュメントは orderbook であり、 salsemanagement の上長のチェックはうけてないない、orderbook は order,ordervote の内容を更新した記載内容であり、各種台帳は存在する。条件を満たす場合、orderbook は salsemanagement の上長のサインをうける。

– 受注票と注文書を保管する遷移関数
op c-storage : State Document Document -> Bool
eq c-storage(S,D,D1) = (D = order) and (D1 = ordervote) and
check-chief(S,orderbook,salsemanagement) .
ceq storage(S,D,D1) = S if not c-storage(S,D,D1) .
ceq place(storage(S,D,D1),order) = archive if c-storage(S,D,D1) .
ceq place(storage(S,D,D1),ordervote) = archive if c-storage(S,D,D1) .
ceq place(storage(S,D,D1),D2) = place(S,D2) if c-storage(S,D,D1) .
ceq contents(storage(S,D,D1),D2)= contents(S,D2) if c-storage(S,D,D1) .
ceq exist(storage(S,D,D1),D2) = exist(S,D2) if c-storage(S,D,D1) .
ceq check(storage(S,D,D1),D2,D3) = check(S,D2,D3) if c-storage(S,D,D1) .
ceq check-chief(storage(S,D,D1),D2,P) = check-chief(S,D2,P) if c-storage(S,D,D1) .

storag 関数はチェックをうけた受注票と注文書を保管する関数である。効力条件は保管するドキュメントは order,ordervote であり、保管されるドキュメントは上長のチェックをうけている。条件を満たす場合、order,ordervote の場所は archive になり、最終状態となる。

この事から、受注プロセスの最終状態は order,ordervote を保管された状態になるので、下記のように final 関数を定義できる。

eq final(S) = (place(S,order) = archive) and (place(S,ordervote) = archive) .

以上が受注プロセスの形式化である。

第4章　リカバリー手法

4.1　具体例をもちいたリスクの分類

前節では受注プロセスの形式化をした。その形式化された受注プロセスを例に、リスクが起こったとしてもリスクをリカバリーし、正常なビジネスプロセスの状態に戻すリスクリカバリーの手法を説明する。

はじめに、リスクがリカバリーされているとは下記の式が成り立つことをである。

final(S) imples not(risk(S,R))　.

この式は、最終状態であるならばリスクは発生していない事を意味する。よってリスクをリカバリーするには、上記の性質を満たすようなコントロールを考えなくてはならない。そこで、リスクをパターン化することによってパターン化されたリスクからコントロールの性質を考察できるのではないかと考えた。以下は受注プロセスをつかったリスクパターン化の説明である。

受注プロセスで考えられるドキュメントに対するリスクの性質は下記になる。

– Risk1 あやまった単価で注文をうける
eq risk(S,r1) = (place(init,order) = client) and not(contents(S,order) = contents(S,price)) .

– Risk2 注文書紛失
eq risk(S,r2) = not(place(S,order) = none) and not(exist(S,order)) .

– Risk3 受注表紛失
eq risk(S,r3) = not(place(S,ordervote) = none) and not(exist(S,ordervote)) .

– Risk4 注文請書紛失
eq risk(S,r4) = not(place(S,ack) = none) and not(exist(S,ack)) .

– Risk5 誤った受注票作成
eq risk(S,r5) = exist(S,ordervote) and (contents(S,ordervote) = contents(S,order))
.

– Risk6 誤った与信限度額で注文をうける
eq risk(S,r6) = (place(init,order) = client) and not(contents(S,order) = contents(S,clientbook)) and not(contents(S,order) = contents(S,orderbook)) and not(contents(S,order) = contents(S,customerbook)) .

– Risk7 期限に間に合わない在庫数の注文をうける
eq risk(S,r7) = (place(init,order) = client) and not(contents(S,order) = contents(S,productbook)) .

– Risk8 誤った注文請書の作成
eq risk(S,r8) = exist(S,ordervote) and not(contents(S,ack) = contents(S,ordervote))
.

この性質を以下の様に分類できる。

ドキュメントを紛失する。
not(place(S,D) = none) and not(exist(S,D))

D はドキュメントを表しており、D が none でないと言うことは、一度は D は作成されてたことを表している。つまり、D が none でなく、D が存在しない場合、D は紛失されたことになる。受注プロセスでのリスクで該当するのは Risk2 注文書紛失,Risk3 受注表紛失,Risk4 注文請書紛失である。

ドキュメントの作成の間違い。
exist(S,D) and not(contents(S,D) = contents(S,D1))

D は記載内容が比較されるドキュメントをあらわし、D1 は比較すべきドキュメントを表している。比較されるドキュメント D が存在するとき、比較すべきドキュメントの記載内容と間違っている場合、D は間違った記載内容でドキュメントを作ったことになる。受注プロセスでのリスクで該当するのは Risk5 誤った受注票作成,Risk8 誤った注文請書の作成である。

内容が間違っているドキュメントが外部から送付される place(init,D) = client and not(contents(S,D) = contents(S,D1))

D は記載内容が比較されるドキュメントをあらわし、D1 は比較すべきドキュメントを表している。比較されるドキュメント D が初期状態でクライアントにあり、比較すべきドキュメントの記載内容と間違っている場合、D は間違った記載内容で注文を受けたことになる。受注プロセスでのリスクで該当するのは Risk1 あやまった単価で注文をうけるである。

受注プロセスでは 3 つのリスクパターンに分類することができたが、他の複数のビジネスプロセスでも形式化を行ったところ、上記の 3 つのリスクパターンに分類することができた。

4.2 リスクパターンに対するコントロールパターン

この 3 つのリスクパターンから考察したコントロールのパターンを以下である。

– ドキュメントを紛失のリスクに対するコントロール

– control1 紛失したドキュメントを再発行する
op re-create-document : State Document -> State
op c-re-create-document : State Document -> Bool
eq c-re-create-document(S,D) = not(exist(S,D)) and not(place(S,D) = none)
.
ceq re-create-document(S,D) = S if not c-re-create-document(S,D) .
ceq place(re-create-document(S,D),D1) = place(S,D1) if c-re-create-document(S,D)
.
ceq contents(re-create-document(S,D),D1) = contents(S,D1) if c-re-create-document(S,D)
.
ceq exist(re-create-document(S,D),D1) = (if D1 = D then true else exist(S,D1)
fi) if c-re-create-document(S,D) .
ceq check(re-create-document(S,D),D1,D2) = check(S,D1,D2) if c-re-create-document(S,D) .
ceq check-chief(re-create-document(S,D),D1,P) = check-chief(S,D1,P) if c-re-create-document(S,D) .

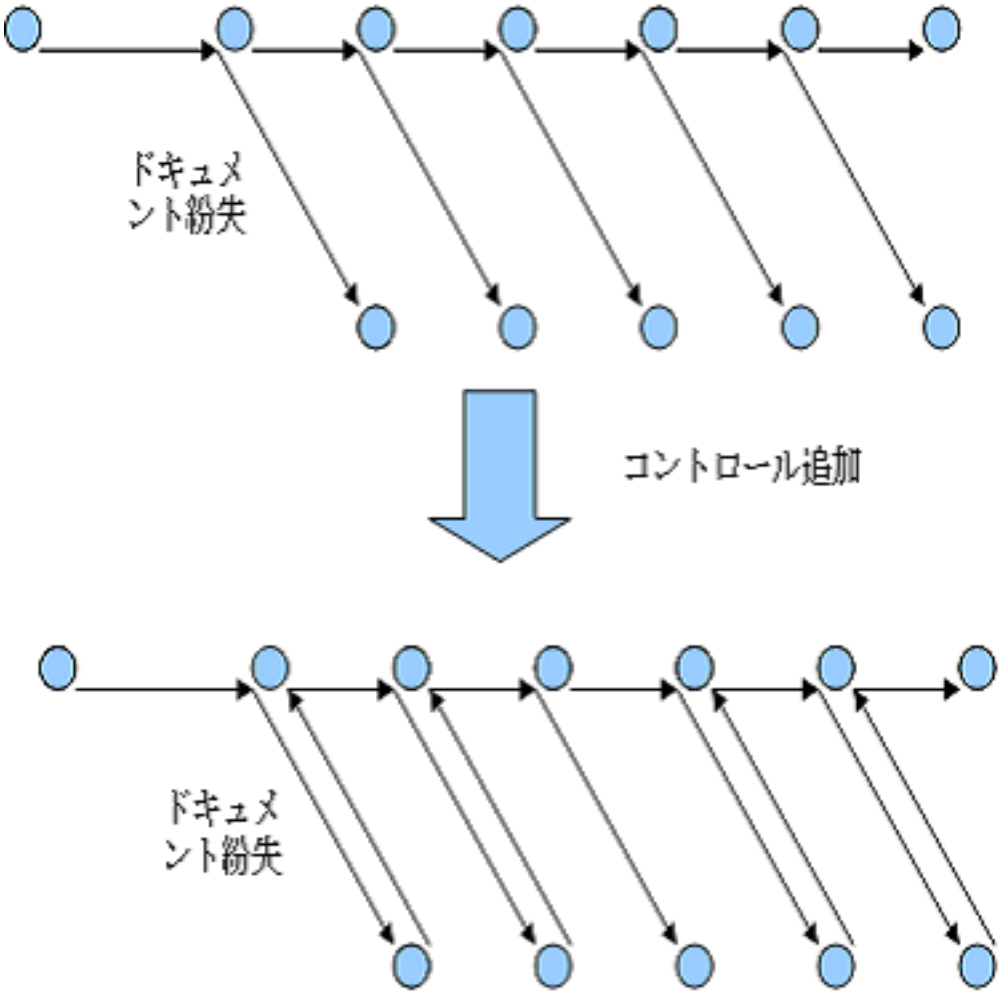

図 4.1: ドキュメントを紛失に対するコントロールの状態遷移図

上記はドキュメントを紛失する遷移が発生したときに、どのようにコントロールを追加しているか示している。ドキュメントが紛失された際に、作業としては特定してる窓口に問い合わせを行いドキュメントを再発行するコントロールを追加することによってリスクをリカバリーする。

–ドキュメントの作成の間違いのリスクに対するコントロール

– control2 誤ったドキュメント内容の書き直し
op re-write-document : State Document Document -> State
op c-re-write-document : State Document Document -> Bool
eq c-re-write-document(S,D,D1) = not(check-chief(S,D,place(S,D))) and not(check(S,D,D1)) and not(contents(S,D) = contents(S,D1)) and not(place(S,D) = none) .
ceq re-write-document(S,D,D1) = S if not c-re-write-document(S,D,D1) .
ceq place(re-write-document(S,D,D1),D2) = place(S,D2) if c-re-write-document(S,D,D1) .
ceq contents(re-write-document(S,D,D1),D2) = (if D2 = D then contents(S,D1) else contents(S,D2) fi) if c-re-write-document(S,D,D1) .
ceq exist(re-write-document(S,D,D1),D2) = exist(S,D2) if c-re-write-document(S,D,D1) .
ceq check(re-write-document(S,D,D1),D2,D3) = check(S,D2,D3) if c-re-write-document(S,D,D1) .
ceq check-chief(re-write-document(S,D,D1),D2,P) = check-chief(S,D2,P) if c-re-write-document(S,D,D1) .

– control3 訂正されたドキュメントの上長の承認
op re-check-document : State Document Document -> State
op c-re-check-document : State Document Document -> Bool
eq c-re-check-document(S,D,D1) = not(check-chief(S,D,place(S,D))) and not(check(S,D,D1)) and (contents(S,D) = contents(S,D1)) and not(place(S,D) = none) .
ceq re-check-document(S,D,D1) = S if not c-re-check-document(S,D,D1) .
ceq place(re-check-document(S,D,D1),D2) = place(S,D2) if c-re-check-document(S,D,D1) .
ceq contents(re-check-document(S,D,D1),D2) = contents(S,D2) if c-re-check-document(S,D,D1) .
ceq exist(re-check-document(S,D,D1),D2) = exist(S,D2) if c-re-check-document(S,D,D1) .
ceq check(re-check-document(S,D,D1),D2,D3) = (if (D = D2) and (D1 = D3) then true else check(S,D2,D3) fi) if c-re-check-document(S,D,D1) .
ceq check-chief(re-check-document(S,D,D1),D2,P) = (if (D = D2) and (P =

place(S,D)) then true else check-chief(S,D1,P) fi) if c-re-check-document(S,D,D1)
.

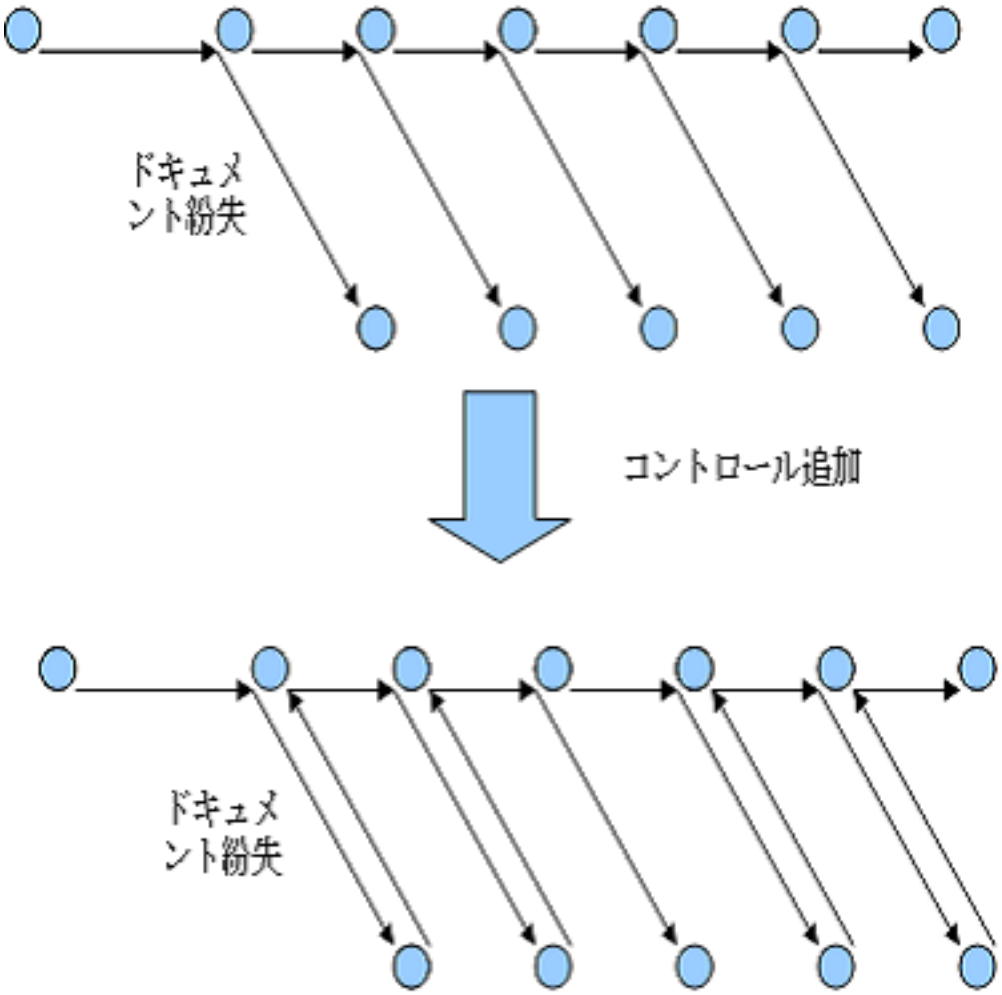

図 4.2: ドキュメントを紛失に対するコントロールの状態遷移図

上記はドキュメントの作成を間違える遷移が発生したときに、どのようにコントロールを追加しているか示している。間違えたドキュメントを作成した場合、コントロールは書き直しを行い、書き直したドキュメントが正しいものか上長がチェックを行う事によってリスクリカバリーしている。

–内容が間違っているドキュメントが外部から送付されるリスクに対するコントロール

– 窓口に問い合わせて、order を作り直す。
op re-order : State Document -¿ State
op c-re-order : State Document -¿ Bool
eq c-re-order(S,D) = (place(init,D) = client) and not(contents(S,order) = 1) .
ceq re-order(S,D) = S if not c-re-order(S,D) .
ceq place(re-order(S,D),D1) = place(init,D1) if c-re-order(S,D) .
ceq contents(re-order(S,D),D1) = contents(init,D1) if c-re-order(S,D) .
ceq exist(re-order(S,D),D1) = exist(init,D1) if c-re-order(S,D) . ceq check(re-order(S,D),D1,D2) = check(init,D1,D2) if c-re-order(S,D) .
ceq check-chief(re-order(S,D),D1,P) = check-chief(init,D1,P) if c-re-order(S,D) .

上記は記載内容が間違ったドキュメントをうけとった遷移が発生したときに、どのようにコントロールを追加しているか示している。間違ったドキュメントが送付された場合、そのドキュメントを確認する遷移でリスクは発見される。発見されたリスクは内容を確認するために、窓口に問い合わせ、再注文させるコントロールを追加することによって、リスクリカバリーしている。

以上がリスクパターンに対するコントロールパターンである。このコントロールによって、形式化した複数のビジネスプロセスはリスクリカバリーすることができた。

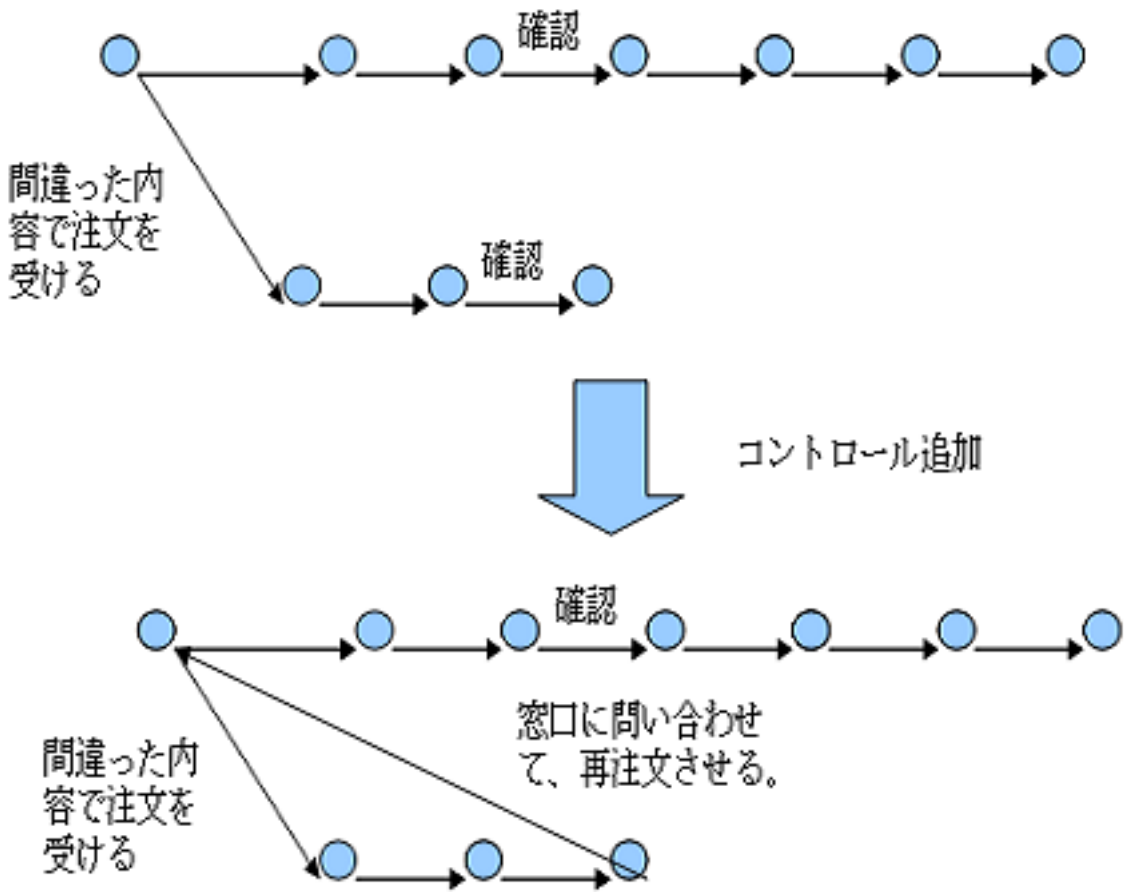

図 4.3: ドキュメントを紛失に対するコントロールの状態遷移図

第5章　実験

これまでのビジネスプロセスの形式化の手法や規則、リスク、コントロールパターンを用いて他のビジネスプロセスでもどのような結果が得られるか実験を行う。実験方法は形式化したビジネスプロセスを用いてプロセスの経路を用意し、どのような結果が得られるか見る。実験では売上計上プロセスを用いて実験を行った。下記は売上計上プロセスの振る舞いを表した状態遷移図である。

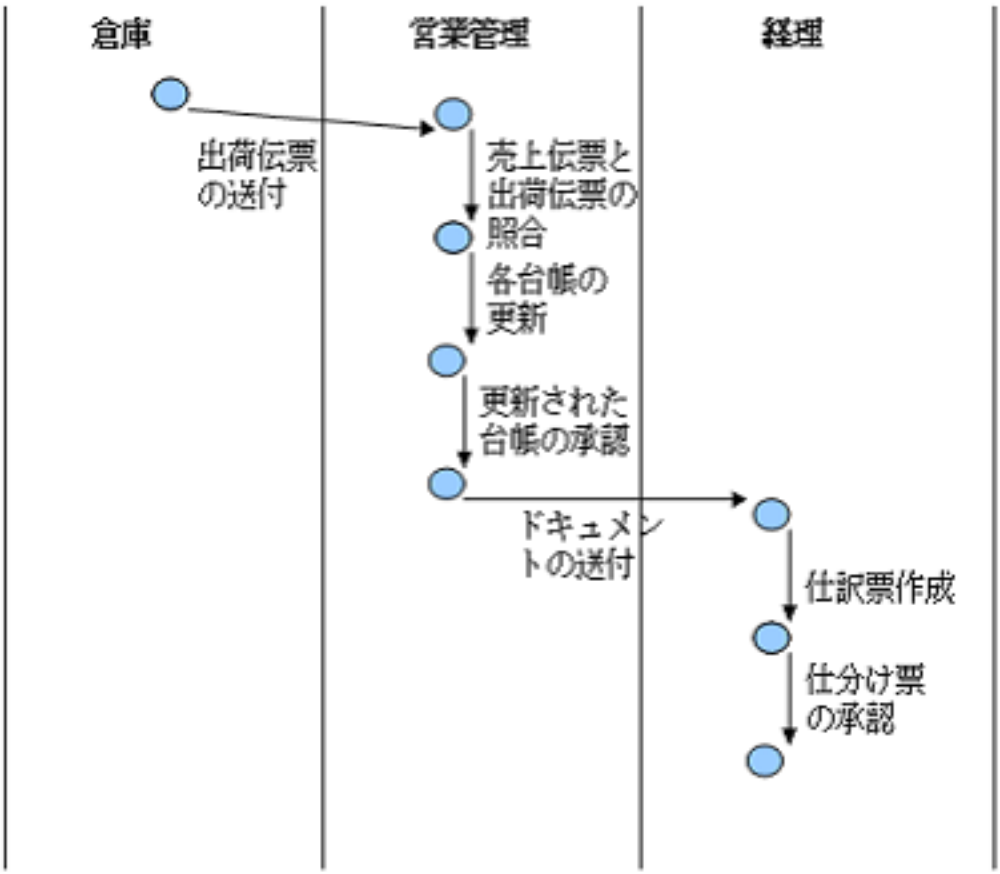

図 5.1: 売上計上プロセスの状態遷移図

この売上計上プロセスから考えられるリスクは以下である。

– risk1 仕訳票の紛失
eq risk(S,r1) = not(place(S,sortvote) = none) and-also not(exist(S,sortvote))
.
– risk2 仕訳票の作成間違い
eq risk(S,r2) = exist(S,sortvote) and-also not(contents(S,sortvote) = contents(S,shipmentslip))
.
– risk3 出荷伝票の紛失
eq risk(S,r3) = not(place(S,shipmentslip) = none) and-also not(exist(S,shipmentslip))
.
– risk4 売上伝票の紛失
eq risk(S,r4) = not(place(S,salesslip) = none) and-also not(exist(S,salesslip))
.

これらのリスクの性質を満たすような観測値に変える遷移を追加して、最終状態までたどり着くか実験を行った。以下がその結果の一部である。

```
-- reduce in %addedsales : (risk(check-sortvote(bad-check-sortvo
ote(d-checkaccountant(send-shipmentslip-salesslip(check-ledger(l
heck(send-shipment(init,shipmentslip),shipmentslip,salesslip),or
book,salesnotebook),orderbook,customerbook,salesnotebook),shipme
),shipmentslip,salesslip),sortvote),sortvote),sortvote),r1)):Boo
(true):Bool
(0.000 sec for parse, 1282 rewrites(0.000 sec), 4714 matches, 76
-- reduce in %addedsales : (final(check-sortvote(bad-check-sortv
vote(d-checkaccountant(send-shipmentslip-salesslip(check-ledger(
check(send-shipment(init,shipmentslip),shipmentslip,salesslip),c
rbook,salesnotebook),orderbook,customerbook,salesnotebook),shipm
p),shipmentslip,salesslip),sortvote),sortvote),sortvote))):Bool
(false):Bool
(0.000 sec for parse, 10 rewrites(0.000 sec), 54 matches, 0 mem
```

図 5.2: リスク遷移のみの実験

この結果は考えられる全ての経路で risk 関数が true の時、final 関数は false となった。つまり、リスクが発生した場合、ビジネスプロセスは活動を停止してしまい最終状態までたどり着くことができなかった。

次にリスクに対してパターン化したコントロールを追加を追加して実験を行った。以下がその結果の一部である。

```
-- reduce in %addedsales : (risk(check-sortvote(re-create-docume
tvote1(create-sortvote(d-checkaccountant(send-shipmentslip-sales
r(ledger-update(d-check(send-shipment(init,shipmentslip),shipmen
,orderbook,customerbook,salesnotebook),orderbook,customerbook,sa
pmentslip,salesslip),shipmentslip,salesslip),sortvote),sortvote)
ote),r1)):Bool
(false):Bool
(0.000 sec for parse, 1486 rewrites(0.015 sec), 5711 matches, 86
-- reduce in %addedsales : (final(check-sortvote(re-create-docum
rtvote1(create-sortvote(d-checkaccountant(send-shipmentslip-sale
er(ledger-update(d-check(send-shipment(init,shipmentslip),shipme
),orderbook,customerbook,salesnotebook),orderbook,customerbook,s
ipmentslip,salesslip),shipmentslip,salesslip),sortvote),sortvote
vote))):Bool
(true):Bool
(0.000 sec for parse, 10 rewrites(0.000 sec), 52 matches, 7 memo
```

図 5.3: コントロール追加の実験

この結果はリスクを追加したことにより risk 関数が false になり、final 関数は true となった。

第6章　考察

前章の結果では、コントロールを追加していないでリスクだけ発生した経路では risk 関数が true であるときは final 関数は false という結果を返した。
この結果からビジネスプロセス内でリスクが発生し場合にコントロールがなければ、最終状態までたどりつくことができずにプロセスの何処かで停止することがわかる。
そして、コントロールを追加した経路では final 関数が true のときに risk 関数は false になるという結果が得られた。
つまり、一度リスクが発生し risk 関数が true になるが追加されたコントロールがリスク発生を検知して遷移が起動するため、risk が false になりただし状態へ戻ることによって最終状態にたどりつけていることが考察できる。
以上の事から、本研究で考案した形式方法を用いてビジネスプロセスを形式化すれば、リスクを回避できることがわかった。この事から、コントロールを発動するための条件式が適切であったと考える。しかし、本研究ではいくつかのビジネスプロセスでしか実験を行っていたい為、統計的には十分であるとはいえず、より多くのビジネスプロセスで実験を行う必要があると考える。そして、形式化をする上でビジネスプロセスによっては遷移や状態を多く作る場合がある。その場合、本研究の形式化を用いると結果のレスポンス時間が多大にかかる。その為、作った遷移関数に対して memo 関数を使うことによって同じ計算をやらないですむため、レスポンス時間が早くなることがわかった。

第7章　結論

本研究では、研究目的であったリスクリカバリーを考慮した形式化をドキュメントに着目することによってビジネスプロセスの振る舞いと、そのプロセスで起こりえるリスクを形式化することができた。リスクリカバリーの手法では、ドキュメントに対するリスクに着目することによってリスクをパターン化することができ、そのリスクパターンに対するコントロールもパターン化することができた。これらの手法をもとに、その他のビジネスプロセスで実験を行いリスクパターンに対するコントロールパターンが有効であることがわかった。しかし、今回の実験では有効な結果が得られたが、今後はより多くのビジネスプロセスに対して実験を行いリスクリカバリーの手法の精度を上げていかなくてはならない。また、今回のリスクリカバリー手法の実験では形式的検証を用いて実験を行っていない。その為、今後は形式的検証を行い信頼性を保障する必要がある。そして、今回提案したリスクリカバリー手法を用いて、リスク遷移がはいってる状態遷移機械を入力として与え、リスクリカバリーをする遷移を追加した状態遷移機械を出力するような自動遷移追加システムを作ることによって、内部統制を構築する際に人為的ミスによって不正や粉飾などが起こることを未然防ぐことができ、社会に役立つようなシステムができると考えている。

参考文献

[1] 著者　佐々野　未知
内部統制の入門と実践
　出版社　中央経済社

[2] 著者　二木　厚吉　　中村　正樹　　緒方　和博
CafeOBJ 入門 (1)
出版社　Japan Society for Software Science and Technology (JSSST)

[3] 著者　二木　厚吉　　中村　正樹　　緒方　和博
CafeOBJ 入門 (2)
出版社　Japan Society for Software Science and Technology (JSSST)

[4] 著者　二木　厚吉　　中村　正樹　　緒方　和博
CafeOBJ 入門 (3)
出版社　Japan Society for Software Science and Technology (JSSST)

[5] 金融庁「企業会計審議会監査部会の公開草案の公表」
URL : http://www.fsa.go.jp/news/newsj/17/singi/f-20050720-2.html

[6] 産業能率大学　総合研究所
URL : http://www.hj.sanno.ac.jp/cgi-bin/WebObjects/107c2074456.woa/wa/read/11a05626225/

[7] (Y. Arimoto, S. Iida, K. Futatsugi, Formalization of Risks and Control Activities in Business Processes.
In Proceedings of 2nd World Congress on Computer Science and Information Engineering (CSIE2011), IEEE, 2011)