ソフトウェア論理の設計/検証における決定表の応用

Application of Decision Table to Software Logic with Designing and Testing

松尾谷　徹

Matsuodani Tohru

概　要

組込システムを含めた情報システムの信頼性，安全性を担保する上で，ソフトウェアが持つ複雑な論理を正確に表現する技法として決定表（decision table）がある．決定表は，古くからその応用局面において様々な工夫が行われながら広く使われていたが，近年，職場での伝承が途絶え衰退傾向にある．

ここでは，決定表の応用について，コードの自動生成を含む強い制約を持つ設計段階，網羅性を高めるテストケース設計に用いるテスト段階，さらに，テストケースの実行を不要とする新たなテスト技法である決定表による検証（decision table verification）について説明する．

1. はじめに

組込システムを含めた情報システムは，規模の増大と共に複雑性が増加の一途をたどっており，その結果，テスト段階において問題が顕在化している．テスト段階の問題とは，実装された複雑な論理を正確に網羅する合理的なテストケースを設計する方法が無く，信頼性を担保するのに必要なテストケース数が爆発し，十分なテストが出来ないことである．

ソフトウェアの論理を正確に表現する技法の一つに，決定表（decision table：判断表とも訳された）がある．決定表は，ソフトウェアの論理を，条件(condition)と，条件の組合せ規則(decision rule または単に rule)と，動作(action)の 3 つに分けて表現する技法である．その応用範囲は広く，設計段階だけでなくテスト段階においても優れた技法である．

近年, 開発段階において統一モデリング言語(UML: Unified Modeling Language)の普及と共に，UML に定義されていない技法が伝承されない大衆化傾向が強くなり，決定表も衰退しつつある．設計段階において複雑なソフトウェアを独立したオブジェクトに分解して詳細化すると, 一見, 複雑な論理は消失する．ところがテスト段階において，想定外の組合せで思わぬ不具合が顕在化する現象に悩まされる．これらの問題の背景に，UML には、複雑な条件間の組合せを論理として表現する技法が含まれてないことから，論理に対する技法や取り扱うスキルが不要であるかのような誤解がある．

ここでは，大衆化されたソフトウェア技法から取り残された，専門的な技法である決定表について説明する．2.決定表の歴史と表記規則では，その研究や応用の流れと共に決定表の種類や表記規則について説明し，3.設計における決定表の応用では，設計段階における応用と課題について，4.テストにおける決定表の応用では，テスト段階における応用と課題について説明する．5.決定表の新たな応用では，課題解決の一つのアプローチとして，決定表による検証（decision table verification）について解説する．決定表による検証とは，テストケースを実機で実行せずに検証を行う技法である．

2. 決定表の歴史と表記規則

ソフトウェア開発のライフサイクル，すなわち，設計・製造・テスト・保守のどの段階においても多くの技術者が参加し，人手による作業が行われている．どの段階においても，技術者はソフトウェアが持つ論理を正確に理解し，作業を行い，正確に記録する必然性がある．よって，自然言語やプログラムコード以外の表現によって，ソフトウェアが持つ論

理を正確に表現し，理解できる技法が必要なことは自明である．論理を表現する代表的な技法が決定表である．

2.1 決定表の歴史

ソフトウェアの設計に決定表が用いられたのは古く 1960 年代からである. 論理回路設計を行った経験を持つ電子技術者がスイッチング理論をソフトウェア論理に応用したことから使われた [1].

1970 年代になると，決定表からソースコードを自動生成するアルゴリズム [2]の研究と決定表の形式化 [3]が行われている．決定表からプログラムコードの自動生成を含む表記方法の統一のため，1976 年から ISO の標準化が始まり 1984 年に ISO 5806 : single-hit decision table が発行され，1985 年に JIS X0125 決定表が承認された [4]．この規格は，単適合決定表（single-hit decision table）と呼ばれる強い制約を持った特殊な決定表を取り扱っている．応用範囲の広い多重適合決定表（multiple-hit decision table）についての規格化は行われていない.

テストの分野では，設計以上に多くの組合せを取り扱う必要から，テストケース設計において決定表が応用されている [5]．単に組合せを表現するための表記法ではなく，ソフトウェアの構造から，起りえない組合せを除いた条件網羅を行うことを目的とする原因結果グラフ [6]や原因流れ図 [7]や論理テスト [10]の中で使われている．設計やテストとは別に，作られたプログラムの論理を見やすくするために決定表を用いる工夫も行われている [8].

このように有用な技術であるが，我が国では 1990 年代に入ると，現場において決定表の伝承が途絶えはじめた．原因の一つは，技術の大衆化が進み，企業の新人研修カリキュラムの充実と共に，先人達が築いた専門性の高い技術の伝承が衰退したことが考えられる.

2.2 決定表の要素

決定表の要素について JIS X 0125 決定表を基に表記法について図１で説明する．決定表の要素は３つあり，

1. 条件部：条件(condition)をリストアップした条件記述部と条件指定部から成る.
2. 動作部 : 動作(action)をリストアップした動作記述部と動作指定部から成る.
3. 規則(rule) : 条件指定部と動作指定部を貫く列の集まり.

説明のため，ある施設の入場料割引を例題にした決定表の例を表１に示す．この例題の仕様は「60 歳以上なら老齢割引，12 歳以下なら子供割引，女性で水曜日に限り特別割引，割引券持参ならクーポン割引をそれぞれ行う」とする.

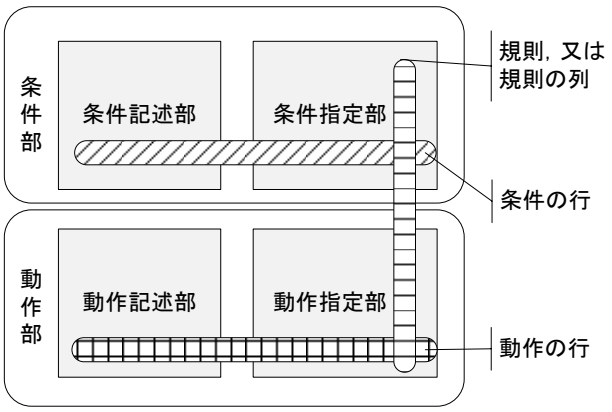

図１　JIS X 0125 決定表の要素とその名称

表1　入場料割引を示す決定表の例

規則		1	2	3	4
条件部	条件記述部				
	年齢	60歳以上	12歳以下		
	性別			女	
	曜日			水曜日	
	割引券				Y
動作部	動作記述部				
	老齢割引	X			
	子供割引		X		
	特別割引			X	
	クーポン割引				X

2.3 決定表の読み方

表１を例に説明を行う．この表の条件部には４つの条件（年齢，性別，曜日，割引券）があり，各条件は表の行を構成している．例えば，年齢の行に着目すると「60 歳以上」「12 歳以下」「空白」「空白」と示されている．このように条件指定部に「60 歳以上」と条件の詳細を示す指定方法を拡張指定と呼んでいる.

条件行の「割引券」は，割引券が「有る」か「無い」かの 2 値なので「Y」: 成立する，「N」: 成立しない，「空白」: 無関係のいずれかで表示する．このような条件行の指定を制限指定と呼んでいる.

JIS X 0125 では，条件指定部と動作指定部において無関係を表すのに「空白」では無く「－」を定義しており，記入漏れの「空白」と無関係を区別する目的である.

動作部には４つの動作（老齢割引，子供割引，特別割引，クーポン割引）があり．動作の行，たとえ

ば特別割引に注目すると「空白」「空白」「X」「空白」である.動作の行と規則の列がクロスする部分に「X」が記載されていれば，その規則の列にある条件の組合せが満たされた時，その動作を実行すると読む.

例えば，３番目の規則（規則の列）の条件部は，「性別」に「女」,「曜日」に「水曜日」が指定されている．この２つの条件が同時に成立する，つまり，「女」かつ「水曜日」（論理積）が成り立てば，特別割引が実行されることを示している．決定表の規則の列は，条件指定部において，論理積である．

2.4　規則間の関係

決定表の規則間の関係について考える．それぞれの規則間で排他関係とは限らないことから，何らかの評価の制約を考える必要が生ずる.

表１の例では，４つの規則がある．規則の評価方法として，１番から順に評価して行き，最初に成立した規則に対応する動作を実行し，後は打ち切るとするのが，単適合決定表である．打ち切らずに，最後まで規則を評価し複数の動作を許すのが，多重適合決定表である.

単適合か多重適合かの区別は，設計段階で用いる場合であり，テストケース設計の場合は，常に多重適合として用いる．詳細については後述する.

3. 設計における決定表の応用

ソフトウェアが論理回路に比べて複雑な論理を持つ原因は,変数に対する条件判定と,条件判定によって変化する制御フローと，制御フロー上の動作が参照・更新するデータの流れ（データフロー）の組合せによって，プログラムに実装された論理が決定する自由度にある．例えば，２分岐の条件文をｎ個持つプログラムは，2^n 個の異なった論理を作ることが可能である.

大きな自由度を持つソースコードへ変換において,属人性を排除し一意に，意図する論理を設計し実装する技法として決定表が用いられる．強い制約を課し，ソースコードの自動生成をも可能にするのが単適合決定表である．多重適合決定表は，制約が少ないが，実装段階での自由度が残る.

3.1 単適合決定表

単適合とは,規則の一つだけしか実行されないことである．決定表に記述された複数の規則において，同時に成立する規則が存在した場合には，規則を順番に評価し，最初に成立した規則に対応する動作を行い，以降の規則を評価しない.

表１を単適合として動作を考えると，4 つの規則は，ただ一つしか適合させないことを意味する．例えば，60 歳以上の女性が水曜日に割引券を持参していても，１番目の規則が適合し，老齢割引のみを実行する．特別割引やクーポン割引は実行されない．規則 4 が適合するには，60 歳以上ではない，かつ，12 歳以下ではない，かつ，女性で水曜日では無い，かつ，割引券持参の場合である.

単適合決定表の場合，例題の意図が「各割引は併用できる」であれば，すべての組合せを表記する必要が生ずる．すべての組合せとは，条件「年齢」に対しては３値（60 歳以上，12 歳以下，13 以上 59 歳以下）と，条件「性別」と「曜日」に対して３値（女＆水曜日，男＆任意の曜日，女又は男＆水曜日以外）と，条件「割引券」の２値の組合せとなり，その数は 3×3×2=18 となる.表２にその一部だけを示す.

このような手間をかけて単適合決定表を作る必要性は，決定表からコード自動生成を可能にするため論理を一意に表現することにある．しかし，正確な論理表現のために，独立した動作についても組合せを記述する手間が必要となるデメリットがある.

表２　表１の単適合化（一部）

規則		1	2	3	4	5	6
条件部	条件記述部						
	年齢	60歳以上	12歳以下	13以上59歳以下	60歳以上	12歳以下	13以上59歳以下
	性別	女	女	女	男	男	男
	曜日	水曜日	水曜日	水曜日	－	－	－
	割引券	Y	Y	Y	Y	Y	Y
動作部	動作記述部						
	老齢割引	X	－	－	X	－	－
	子供割引	－	X	－	－	X	－
	特別割引	X	X	X	－	－	－
	クーポン割引	X	X	X	X	X	X
	割引なし	－	－	－	－	－	－

3.2 制限指定と拡張指定

決定表で取り扱う論理を正確にするもう一つの制約は，条件指定部の記述である制限指定と拡張指定である．2.3 決定表の読み方でも説明したように，制限指定は，条件の判定結果が 2 値（真か偽）であることを要求している.表１の割引券の例に該当し，「Y/N」「空白又はー」の３種類だけの記述に制限される.「－ 」は，当該規則に当該行の条件は関係しないことを表すが，表１は「空白」を代用している.

拡張指定の例は，表１の「年齢」に対する条件指定部は「60 歳以上」「12 歳以下」「空白」である．条件の判定結果は，多値論理となり，例えば「50 歳以

上」「60 歳以上」のように条件間で包含関係にある条件を記述することも表記上は許される．しかし動作条件を示す仕様において，曖昧になることから，単適合のルールを適用し，一意にするのが単適合決定表の制約である．

3.3 多重適合決定表

多重適合決定表に関する規格は存在しないが，実務における有用性は高い．多重適合決定表では，規則は全て評価され，成立した規則と対応する動作の実行は，ソフトウェアの実装時の仕様として決定表とは別に考える．

単適合決定表と比べ, 制約は弱く応用範囲は広い．例えば，例題とした入場料の割引問題で，各割引が併用出来る場合においても, 表 1 の各割引動作に「現在の定価から割引金額を差し引き，現在の定価とする」とすれば表 2 のような組合せを作る必要は無い．

欠点としては，論理的な矛盾を含んだ決定表を書くことも可能である．特に，条件部に拡張指定を用いると, 排他ではない多値をも記述できることから，実装において多様性が生じる．

3.4 単適合と多重適合の実装上の差

単適合と多重適合の差は，実装される制御構造の違いでもある．表 1 の例題に対する単適合の制御構造と多重適合の制御構造を図 2 に示す．

単適合に対応する制御構造は, else-if の連結によって判定を行うこと，動作が指定された後は決定表の処理を終えることが特徴である．多重適合に対応する制御構造は，各判定は真偽に関わらず，次の判定を行う制御構造である．

図 2 に示した制御構造は，どちらも良く見かける制御構造で有り特別なものではない．問題なのは，プログラミングを行っているエンジニアが，else-if の結合と if の結合の違いが，ソフトウェア論理に与える影響について正しく理解しているか否かにある．しかも現実のソフトウェアでは，単適合型と多重適合型のコードが混在しており，コードから論理を正しく読みとるのは非常に難解である．

3.5 設計における決定表の応用と課題

決定表は，歴史的に手続き型プログラミングから生まれてきた技法であり，論理を制御構造によって表現する．そのためデータ指向やオブジェクト指向における応用例は少ない．

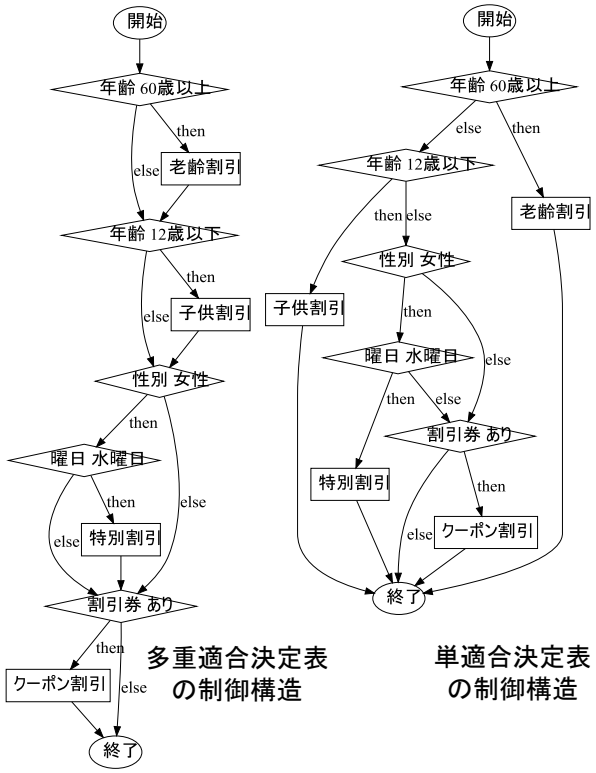

図 2　多重適合と単適合の制御構造

手続き型プログラミングにおいて，複雑な論理を表現する方法は，形式手法を除けば，実用されているのは決定表しか無い．それにも関わらず，普及しない原因は，適切な教科書や指導者が不足していること，JIS X0125 単適合決定表が全てであるかのような誤解，などにあると思われる．

データ指向やオブジェクト指向においても，複雑な論理が無い訳ではなく， UML に論理を表現する技法が定義されていないことから，論理を表現しなくて良いと考える誤解がある．こちらも応用事例など，普及のための資料や指導者が不足している．

決定表は，データフローによるソフトウェア論理を明示的に表現していない課題がある．手続き型プログラミングにおいても，ファイルから大量の入力を読みループ処理を行う場合や状態遷移に応用するには複数の決定表を結合するなどの工夫が必要である．データ指向やオブジェクト指向においては，タスク間でオブジェクトを参照することによって変化する論理の表現に対して工夫が必要である．

4. テストにおける決定表の応用

テストにおける課題は，テストの網羅度を保ちながら，少ないテストケース数に抑えること，及び,

テスト担当者の属人性による網羅度の低下を防ぐことである．少ないテストケース数で網羅を保つためには，不要な組合せを削減する技術が必要である．

先に述べたように，ｎ件の条件が存在すると，起り得る組合せの数は，2^n個となり全ての組合せを網羅するテストケース数の爆発が生ずる．この課題に対し，多くのテスト技法は，完全な網羅を捨て妥協できる網羅基準を使うことによって成立している．

最も網羅性が高いと思われるホワイトボックステストの分岐網羅は，動作間でデータフローの変化が無いと仮定している.ブラックボックステストでは，経験則が多く，直交表などを用いる組合せテストでは，２から３因子以上の組合せの影響は少ないと仮定し削減する．業務テストでは，実業務の流れによる組合せに限定している．

決定表は，論理テストと呼ばれる条件の複雑なテストのための技法の中で，論理の組合せを表現し，論理に関する網羅を高めるために使われている．

4.1 テストと設計の違い

設計とテストの大きな違いは，設計においては，意図した条件で意図した動作が機能する論理のみを取り扱うが，テストでは，意図しない条件においてもソフトウェアが無害であることを確認する必要がある．意図しない不具合は，ソフトウェア論理の不具合として混入する可能性があり，仕様書で定義された条件とその組合せについてテストするだけでは不具合を見落とす.

設計との違いを示す概念に,有則と無則がある[9)]．有則とは，仕様で定義された条件とそれに対応する動作の集合のこと．無則とは，仕様では定義されていないが入力可能な条件とその条件における動作の集合のこと．起り得る全ての条件空間の中で有則の補集合に相当するのが無則である．

この概念によれば，テストは，有則が正しく動作することを確認すると有則の網羅と共に，無則において無害であることを確認する無則の網羅が必要である．安全性の分野では，設計段階において，無則の集合を極力小さくする禁則を積極的に導入している．高信頼性システムや安全関連システムにおけるテストの網羅性は，３つの概念で整理することが出来る．

1. 有則網羅：与えられた仕様上の条件や動作が正しく機能することの網羅.
2. 禁則網羅：禁則動作を起動する条件で正しく禁則が機能することの網羅.
3. 無則網羅：有則と禁則の捕集合において，無害であることの網羅.

設計段階において禁則の概念を持たないソフトウェア設計も多く存在する．しかし，実装されたソフトウェアには，制御フローが存在するので起りえない組合せも多数存在する．明示的に禁則の仕様を持たない場合でも，無則における要素数が2^n個になることは無い．つまり，暗黙の禁則が存在する．

4.2 テストケース用の決定表と要素

テストケース設計に用いる決定表は，設計における決定表と異なった定義が必要である．規格などで合意されたテスト用決定表の定義は未だ無いが，設計における決定表の定義に合わせて，次のように定義することが出来る．

条件に対応するテストの概念は原因(cause)である．動作に対応する概念は結果(effect)である．規則に対応する概念がテストケースである．

テストにおける原因とは，テストのために準備する入力データや環境に当たる．結果とは，テストのために与えた原因（テストデータ）によって被テスト対象からの出力である．結果には２つの側面があり，テストケース設計時に求めた結果と，実際にテストを行って得られるテスト実行の結果である．テストケース設計時の結果をテストケースの正解値と呼びテスト実行時の結果と区別する．テストの合否判定は，テストケース設計で得られた正解値としての結果と，実際のテスト実行によって得られたテスト結果を比較して行う．

図3にテストケース設計用の決定表の構造を示す.図3の原因部には,テスト入力となる変数やパラメータをリストアップする．テストにおいても，設計における制限指定と拡張指定が可能であるが，無則網羅を考慮すると，条件の捕集合を取り扱う必要性から，制限指定が必要となる．テストにおける制限指定は，テスト技法として古くから用いられている同値分割である．同値分割とは，条件の取り得る域値を同値クラスに分割する技法である．分割された同値クラスは,２値論理として取り扱うことが出来る.

結果部は，設計における動作と変わらないが，テストの場合には，何らかの出力値として確認できることが必要である．例えば，ソフトウェア内部の状態を遷移させる動作なら，状態遷移の結果として出力される値や，テスト用のデバッガーで直接状態変

数の変化を観測する必要がある.

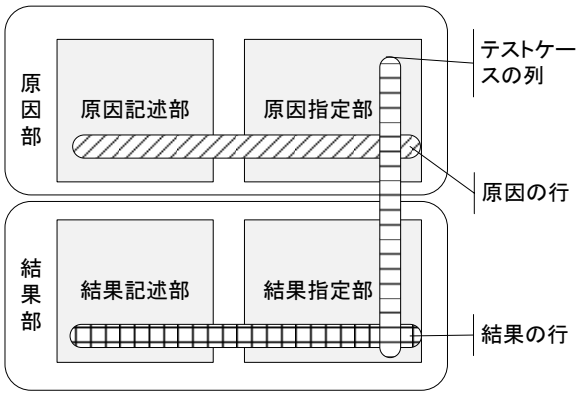

図 3　テストケース設計用の決定表

4.3 原因と同値分割

テストケース設計用の決定表の特徴は，原因記述部にある．設計における条件は，意図する動作を意図する条件で実行することであるが，テストは意図しない条件でも確認する必要がある.

表 1 で示した例題の条件「性別」を例に考える．外部仕様から定義すると条件「性別」は「男」「女」の 2 値である. ソフトウェアの実装段階において「性別」は変数を使って表現される．変数の型が論理値（ブール値）で定義されていれば，2 値以外は取りえない．もし，変数の型が整数で「男」=1,「女」=0 と定義されていると，1 や 0 以外の値が入力される自由度がある．その結果「女」の捕集合が「男」とは限らない. 同値分割では「女」「男」「それ以外」と分割し「それ以外」の具体的なテストデータ値は，同値の境界値から「2」や「-1」を使ってテストする.

同様に，条件「年齢」について考えると「年齢」も変数であり，様々な値を取り得る．そこで，値をある程度の大きさの領域:同値クラスとして考える．テストでは，同値クラス（以下，同値と省略する）に分解することを同値分割と呼んでいる.「0 歳から 12 歳」「60 歳以上」の 2 つの部分集合が仕様で定義された同値で有り，これらを有効同値と呼ぶ.「13 歳から 59 歳」「マイナスの値」「数字以外」「Null」などが無効同値である．無効同値の値そのものは，多様であることから，決定表では値そのものではなく，部分集合である同値として扱う.

同値は, ある変数に対してとり得る値を同値類(重複しない）として部分集合に分解したものである．論理の厳密さを求める場合は, 完全同値分割と呼ぶ,「母集合＝Σ分割された同値」を満たす重複と漏れの無い分割が必要になる.

図 4 は，漏れの無い同値分割の例を示す．同値間の制約を線で示す方法は, G J.Myers の原因結果グラフ [6]で用いられている．ベン図を使って完全同値分割を行うのは原因流れ図 [9,11]である.

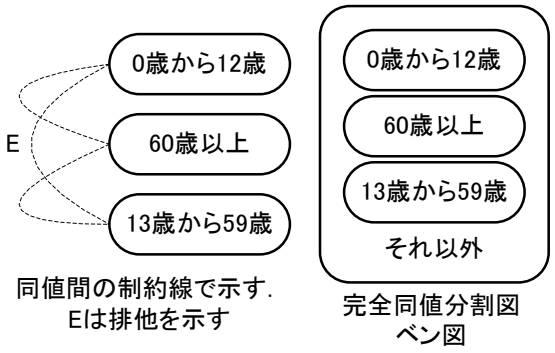

図 4　同値分割の例

表 3　テストケース設計の例

テストケース			1	2	3	4	5	6	7	8	9	10
原因部	年齢	0歳から12歳	Y									
		60歳以上		Y								
		13歳から59歳			Y							
		それ以外				Y						
	性別	男						Y				
		女					Y		Y			
		それ以外								Y		
	曜日	水曜日					Y	Y		－		
		それ以外							Y	－		
	割券	あり									Y	
		なし										Y
結果部	老齢割引		X									
	子供割引			X								
	特別割引						X					
	クーポン割引										X	
	エラー					X				X		
	各割引なし				X			X	X			X

4.4 テストケース（原因の組合せ）

表 3 にテストケース設計の例を示す．原因部は，原因名（年齢，性別,）をさらに同値に分解して表示する．個々の同値は 2 値（Y/N）をとり，同じ原因の同値は排他であり，何れか一つの同値しか真にならない.「それ以外」は else に相当する.

結果の部は，仕様から定義された 4 つの割引以外に,「エラー」と「割引なし」を追加している.「エラー」は, 仕様上は定義されていないが, 誤ったデータが入力された時，実装上の都合で追加された動作（実行時仕様）である．テストは，仕様に表れない実装上の動作，例えば「バッファ不足」や「ファイル読込エラー」などについても原因と結果を洗い出

しテストを行う必要がある．「各割引なし」は，それぞれの割引が動作しない結果を表す．正確には「老齢割引も子供割引もなし」「特別割引なし」「クーポン割引なし」と分割するが，紙面の関係で省略した．別の記述方法として，結果に「N」を記入し，動作しないことを示す表記もある．

テストケースの設計は，先ず，原因を共有する単機能について行う．表３の例では，原因「年齢」を共有する「老齢割引」「子供割引」と，原因「性別」「曜日」と「特別割引」，原因「割引券」と「クーポン割引」である．原因指定部の「空白」は，無関係を意味する．「－」は原因内の同値間では排他的論理和を意味し，原因間で「－」が存在すれば倫理和を示す．表３のテストケース８の「－」は，水曜日でもそれ以外でも良いと読む．

4.5 無則網羅のテストケース

表３は，有則網羅と，原因を共有する組合せにおける無則網羅を達成している．原因を共有するには，{1-4}，{5-8}，{9-10}の３群である．

組合せとしての無則について考える．表３では，{1-4}，{5-8}，{9-10}の３群で組合せが生ずる．禁則を考慮せずに組合せると 4＊4＊2=32 のテストケースになる．禁則として，エラーが生じた場合は，処理を打ち切るとすれば，テストケース 4，8 は単適合となり，以降の組合せができない．

残る {1-3}，{5-7}，{9-10}は，動作間でデータフロー上の結合が無ければ，それぞれのテストケースを１回は選択する基準で３個の連結したテストケースとなる．この連結は，図２の多重適合決定表の制御構造で考えるなら，ホワイトボックステストの分岐網羅に相当する．

原因流れ図技法 [9,11]は，モジュールや関数や画面の流れ図から，テスト用の決定表を作成する．データフローについては，状態遷移や画面遷移などを考慮する工夫が行われている．

4.6 テストにおける決定表の応用と課題

決定表は，テストにおいも論理を表現する技法として有効である．テスト固有の特性から JIS X0125 の規定は使えず，テスト用の定義が必要である．

多重適合型であれば，設計とテストでの差は，条件の部分に実装時仕様から生ずる原因を追加するなどすれば，再利用することが出来る．複雑な論理を維持管理するためには，設計－テスト－保守を通した決定表の運用が必要である．自然言語と難解なソースコードによる設計－テスト－保守は，決して合理的な技術ではない．

決定表の問題ではないが，テストにおける組合せの無則網羅は，大きな課題である．この課題については次に述べる．

5. 決定表の新たな応用

近年，テストに関連する新たな技術が出現している．上流工程から連続したアプローチで，形式仕様を用いて設計し，形式仕様から有効系のテストケースを自動的に抽出する技法が開発されている [12,13]．

それとは逆に，リバース技術を使った Concolic testing と呼ばれるもので，プログラムコードを入力とし，シンボリック実行によって実在する制御フローを自動探索する [14,15,16]．ヒューリスティックな手法であり，複雑なポインター操作など特殊なプログラミングによっては探索が漏れることもあるが，実在する 150K ステップのソフトウェア製品に対し，実用上の分岐網羅を達成している．Concolic testing を実現するツールも公開されている [15,16]．なお Concolic は concrete と symbolic の造語である．

Concolic testing は，実装されたソフトウェアの実在する制御フロー上で分岐網羅を達成することにより，テストを達成できると考えている．従来のテストとは異なり，無則網羅が飛躍的に進む一方，有則網羅について問題がある．問題は，実在する制御フローは，仕様で求める論理と一致しているとは限らない有則網羅のことである．

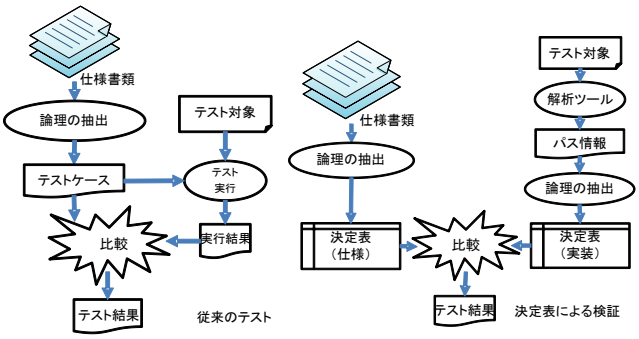

図５ 従来のテストと決定表による検証

Concolic testing を使って，論理のテストにおける有則網羅と無則網羅を達成する方法として，決定表による検証（decision table verification）が研究されている [17]． 従来のテストの流れは，図５に示すように，テストケースからテストデータを人手で作成し，テストを実行し，正解値と比較してテスト結果

を得ていた．一方，決定表による検証は，ツールを使って実装された制御フローを解析し，その解析結果から決定表（実装）を作成する．一方，仕様からは 3. で示した方法で決定表（仕様）を得る．両者を比較することにより，有則網羅と無則網羅を達成する方法である．

6 おわりに

長い歴史のある決定表について，設計とテストにおける応用と，新たな応用について紹介した．

システムの信頼性や安全性を担保するには，ソフトウェアが持つ複雑性に対し,必要な技術を選択し，適切な運用を行うと言う大原則から，決定表の応用と普及が必要である．特にテスト，あるいは，テストを通したライフサイクルとしての応用に期待する．

参考文献

1) Pollack et al.: Decision Tables: Theory and Practice, John Wiley & Son (1971)
2) Robert L. Ashenhurst: The synthetic approach to decision table conversion, Communications of the ACM, Vol. 19, Issue 6, 343-351 (1976)
3) 守屋慎次：デシジョンテーブルの形式化，情報処理, Vol. 19, No. 519, 398-405(1978)
4) 守屋慎次,菅忠義 : 決定表 ，情報処理 28(9), 1136-1140, (1987-09-15)
5) 古川善吾,野木兼六,徳永健司：AGENT : 機能テストのためのテスト項目作成の一手法，情報処理論文誌 25(5) , 281-288 (1986.03.15)
6) Grenford J. Myers, 長尾真監訳：ソフトウェア・テストの技法 2 版，近代科学社(2006).
7) 赤沢隆夫，松尾谷徹：原因流れ図によるテストケース抽出技法，情報処理全国大会，第 38 回，1225-1226（1989.03.15）
8) 守屋慎次,平松啓二：デシジョンテーブルによるプログラムの自動ドキュメンテーション，情報処理 17(9), 820-827(1976.09.15）
9) 松尾谷徹：テスト/デバッグ技法の効果と効率(<特集>ソフトウェアテストの最新動向)，情報処理 49(2), 168-173（2008.02.15）
10) ボーリス・バイザー著 ；小野間彰，山浦恒央訳：ソフトウェアテスト技法，日経 BP 出版センター, 267-276(1994.2)
11) 松本正雄，小山田正史，松尾谷徹共著 : ソフトウェア開発・検証技法，電子情報通信学会(1997.1)
12) 栗田 太郎：携帯電話組込み用モバイル FeliCa IC チップ開発における形式仕様記述手法の適用，情報処理 49(5), 506-513 (2008.05.15)
13) 栗田太郎,荒木啓二郎：モデル規範型形式手法 VDM と仕様記述言語 VDM++，信頼性 31(6), 394-403(2009)
14) Rupak Majumdar, Koushik Sen, : Hybrid Concolic Testing, icse, 416-426, 29th ICSE'07 (2007)
15) K. Sen, D. Marinov, and G. Agha :‘CUTE: A concolic unit testing engine for C’, 5th joint meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE’05), ACM(2005)
16) P. Godefroid, N. Klarlund, and K. Sen : ‘DART: Directed automated random testing’, Proc. of the ACM SIGPLAN 2005 Conference on PLDI(2005)
17) Keiji Uetsuki, Tohru Matsuodani, Kazuhiko Tsuda: Software logical structure verification method by modeling implemented specification, KES'11 Proceedings of the 15th international conference on Knowledge-based and intelligent information and engineering systems - Volume Part III,336-345,(2011)

1972 年 3 月関西大学大学院修士課程修了．1972 年 4 月日本電気入社．周辺装置の設計を経て，汎用 OS 開発他システム開発に従事．2002 年 8 月早期退職後，現職．

2005 年 3 月筑波大博士課程修了．博士（システムズ・マネジメント）