日本語諸方言の音調体系の定式化（3）

岡　田　英　俊

0　序

本稿は，岡田(1990,1991)に続いて，日本語諸方言の音調体系の定式化を行う。今回は，前2稿で取り上げた合計13の方言の音調体系を，プログラミング言語Scheme（Lispの一方言）を用いて，記述する。新たな方言は，取り上げない。使用したSchemeは，TUTScheme（湯浅太一 1991）である。

対象となる方言は，次のとおりである。

東京方言，京都方言，久見方言（島根県隠岐郡五箇村），
伊吹島方言（香川県観音寺市），村上方言（新潟県村上市）；
広島方言（香川県丸亀市），志々島方言（香川県三豊郡詫間町），
佐柳島方言（香川県仲多度郡多度津町），岩黒島方言（香川県坂出市），
真鍋島本浦方言（岡山県笠岡市），真鍋島岩坪方言（岡山県笠岡市），
金沢方言（石川県金沢市），高見島方言（香川県仲多度郡多度津町）．

岡田(1990)において，東京方言から村上方言までの5つの方言を扱い，岡田(1991)において，広島方言から高見島方言までの8つの方言を扱った。各方言の音調体系についての直観的な解説は，前2稿で示した。本稿では，繰り返さない。また，各方言の音調体系のデータ，及び，その基本的な捉え方は，すべて，先行する諸論考に負っている。依拠した論考は，前2稿で示した。

本稿において，プログラミング言語を使用する目的は，(a)定式化した結果を，ある程度広く通用している表現方法（すなわち，プログラミング言語）に基づいて，明示的に記述し，(b)計算機で実行することにより，定式化の誤りを検出する，ということである。したがって，大量のデータの統計的な処理を行うような場合と比べると，計算機の果たす役割は，全く異なる。

なお，筆者は，Schemeについて（また，プログラミングに関する事項全般について），十分な知識を有していない。本稿で示すプログラムには，不適切な点が，多数，存在するものと思われる。御教示を頂ければ，幸いである。

以下，まず，§1において，プログラムの概要を述べる。§2において，個々の関数の解説を行う。関数の定義そのものは，§3で示す。最後に，§4において，岡田(1990,1991)の訂正を行う。

1 概要

1.0 序

まず，プログラムの概要を述べる。

プログラムの作成に際しては，枠組みを，可能な限り，そのまま移し変えた。したがって，単なるプログラムとしては，非効率的な部分が多い。これは，本稿の目的からして，当然のことである。

処理の効率だけを考えるならば，そもそも，「縮約」の操作を設ける必要はない。縮約を必要とする音調パターンを，単なる「例外」として扱えばよい。しかし，そのような方針は，とっていない。

簡素化のため，プログラムにおけるエラー処理の部分の掲載は，省略した。岡田(1991：163)で定めた「適格性判定関数」も，組み込んでいない。

プログラムにおいては，tone と名付けた関数が，言わば，本体である。必要な情報を tone へ入力すると，出力として，音調パターンが得られる。

前稿では，入力たるアクセント素性構造，及び，出力たる音調パターンを，素性構造で表現した。プログラム化に際し，この入力・出力の表現を，若干，変更する。以下，これについて，及び，その他の修正事項などについて，述べる。

1.1 アクセント素性構造の表現

本稿のプログラムでは，前稿までのアクセント素性構造を，リストとして表現する。直観的に言えば，リストは，素性構造（すなわち，集合）とは異なり，要素の順序が定まっている。しかし，アクセント素性構造を表現するリストにおいて，要素の順序は，意味をもたない。任意の順序で並べればよい。

リストの要素たる素性指定自体も，リストで表現する。これは，実質的には，前稿までと同じである。すなわち，素性 x とその値 a から成る素性指定を，リスト（x a）によって，表現する。

Scheme には，真偽値として，それぞれ，#t,#f が備わっている。しかし，TUTScheme においては，偽と空リストが，同一視される。そのため，素性構造を扱うに際して，不都合の生ずる恐れがある。そこで，素性の値としての真偽値については，真を記号 t で表し，偽を記号 f で表すこととする。

プログラムにおいては，素性の名称を，次のように定める。矢印の左側が，前稿までの名称である。右側が，本稿のプログラムにおける名称である。プログラムにおいて，このように素性の名称を変更するのは，識別を容易にするために過ぎない。なお，素性 acc に

ついては，変更がない。素性 V については，次の段落で述べる。len は，“length”の略である。

T → tone
T′ → tone2
M → mark
N → len
P → particle
F → final
I → initial

岡田(1991：158)においては，素性 V を，素性構造を値とする素性として定めた。しかし，今のところ，素性構造を値とする素性は，この素性のみである。そこで，簡素化のため，素性 V に代わるものとして，var1，var2，var3を設ける。

V の値が，

$$\{(1, n_1), (2, n_2), (3, n_3)\}$$

であるとき，var1，var2，var3 の値が，それぞれ，n_1，n_2，n_3であるとして，表現する。

素性 var1，var2，var3 の値は，本来は，分節音に関する情報などから導かれる。あらかじめ与えておくものではない。しかし，本稿では，簡素化のため，これらの素性の値を導く過程を，扱わない。すでに値が定まっている時点を，出発点とする。

以上により，例えば，アクセント素性構造［+acc, 3 M, 4 N]/+I は，

((acc t) (mark 3) (initial t) (len 4))

として表現される(すでに述べたとおり，素性指定の配列順序に，意味はない。素性 len に関する指定を，最後に置くこととした)。

実際には，これに，方言名の情報を加える必要がある。本稿では，方言名を表現するための素性(岡田(1991：147)参照)は，設定しない。方言名を，アクセント素性構造とは別の情報として，関数 tone へ渡す。例は，§1.2 の末尾で挙げる。

1.2　音調パターンの表現

音調パターンは，前稿の表現では，素性構造である。本稿では，アクセント素性構造と同じく，リストで表現する。

音調パターンを表現するリストにおいて，音調変動を並べる順序は，当該音調変動のインデックスの順序関係(岡田(1990：168)で定義した)に従うこととする。もちろん，一旦，無作為に並べ，改めて，インデックスの順序関係に基づいて整列させることも，可能である。いずれにしても，整列が完了した時点を，出発点とする。

なお，インデックスの順序は，線型順序ではない。したがって，今述べたことは，正確

ではない。しかし，この種の些末な事項については，省略する。

素性の名称と同じく，識別を容易にするため（かつ，+，−などの演算記号の使用を避けるため），プログラムにおいて用いる音調変動セグメントの名称を，次のように定める。pos，neg は，それぞれ，“positive”，“negative” の略である。

$+ \rightarrow$ pos

$+' \rightarrow$ pos2

$- \rightarrow$ neg

$-' \rightarrow$ neg2

$e \rightarrow$ e

プログラムにおいて，時間軸成分は，実数ではなく，分数で表現する。ただし，Scheme には，データの型として，分数は備わっていない。そのため，ドット対を用いて，分数を表現する。例えば，分数 2/3 を，(2 . 3)によって表現する。

前稿までの枠組みにおいて，音調変動は，インデックス，時間軸成分，音調変動セグメント成分の 3 つの成分から成る対であった。これを，そのまま，リストとして表現すれば，十分である。しかし，本稿では，もう 1 つの情報を，音調変動に組み込む。組み込むのは，当該インデックスが集合 J_0，J_1，J_2 のいずれに属するか，という情報である。インデックス i が J_n に属することを，「インデックス i の類は n である」と称する。

J_0，J_1，J_2 は，岡田(1990：141)で定めた。直観的に言えば，J_0，J_1，J_2 の要素をインデックスとする音調変動は，それぞれ，アクセント単位の始点，アクセント標識の存在するモーラの終点，アクセント単位の終点，を基準点として，その時間軸成分が決まる。

インデックスの類は，音調変動の第 2 の要素として，インデックスの次に置く。例えば，前稿までの表示法で，

$(p, -1/2, +)$

と表現される音調変動は，$p \in J_2$ ならば，

(p 2 (−1 . 2) pos)

と表現される。

インデックスの類に関する情報は，本来は，音調変動に組み込む必要がない。しかし，組み込まないとすると，インデックスを受け取ってその類を返す関数を，作成しなければならない。かつ，その関数を，頻繁に呼び出さなければならなくなる。そのため，便宜上，音調変動に，この情報を組み込むこととした。

実際の音調パターンの算出例を，1 つだけ挙げる。東京方言において，第 3 モーラにアクセント標識の存在する 4 モーラ語（例えば，「青空」）の，句頭における音調（高低で表すと，低高高低）の算出を，例にとる。

アクセント素性構造は，次のとおりである。

((acc t) (mark 3) (initial t) (len 4))

関数 tone によって音調パターンを算出するには，方言名の情報を加えて，Scheme のシステムに対し，次のように入力する。

(tone 'tokyo '((acc t) (mark 3) (initial t) (len 4)))

すると，次の出力が得られる。

((e1 0 (0 . 1) e) (p1 0 (0 . 1) neg) (p2 0 (1 . 1) pos)
(k 1 (3 . 1) neg) (e2 2 (4 . 1) e))

直観的に言えば，この音調パターンは，長さが 4 モーラであり，低く始まって，第 1 モーラの終点で音調が上昇し，第 3 モーラの終点で音調が下降する。すなわち，期待された音調パターンが，得られたことになる。

1.3 前稿までの内容の拡張・修正

次に，前稿までの枠組み・分析について，拡張・修正を行う。ここで述べる事項は，当然，すべて，プログラムの中に表現されている。

最初に，合算・等間隔化の操作の適用時点に関して，修正を加える。従来は，合算・等間隔化の適用時点を，個々の縮約の操作（除去・圧縮等）の適用直後としていた。

まず，合算の適用時点については，不備がある。合算は，除去・圧縮等が全く適用されなかった場合にも，適用しなければならない。そこで，一連の縮約の操作（ただし，全体圧縮を除く）が終了した直後にも，1 回，合算を適用することとする。

また，変異規則・修正規則が適用された場合の，合算の適用時点については，岡田(1991)に，明示的な規定がなかった。これについては，一連の変異規則をすべて適用した直後，及び，一連の修正規則をすべて適用した直後に，合算を適用することとする。

一方，等間隔化については，適用時点を修正し，一連の縮約の操作（全体圧縮を含む）が終了した直後に適用するのみとする。これについては，従来のとおりでも，全く問題がない。しかし，今のところ，これ以外の機会に等間隔化を適用すべき例がないので，このように修正する。

合算・等間隔化の適用時点については，以上である。

次に，枠組みの拡張を行う。「表層規則」と名付ける新たな規則を導入する。

表層規則は，表層のレベル（または，それに近いレベル）において，音調変動の音調変動セグメント成分の変更を行う。直観的に言えば，ある条件を満たす「高」を「中」に変えるなどの，音声レベル（に近いレベル）の操作を行うことになる。

「表層」という語は，便宜上のものである。特別の意味はない。

表層規則において，音調変動のインデックスに言及することは，認めない。これにより，表層規則とそれ以外の規則の役割分担が，明確になる。なお，表層規則の適用直後は，合

算・等間隔化を適用しない。

岡田(1990,1991)で取り上げた方言には，表層規則の適用例がない。しかし，今後取り上げる方言のことを考慮して，早目に導入する。

次に，分析の修正を行う。修正を加えるのは，岡田(1990：155-160)の§4で扱った久見方言の分析である。

岡田(1990)においては，久見方言の語声調を，アクセント標識の有無とその位置という情報に，変換した。この分析を変更し，アクセント標識に関する情報への変換を，一切，行わないこととする。これに伴い，インデックスの名称や，縮約に関する情報の述べ方なども，変更を受ける。

なお，岡田(1990：158)において，久見方言の縮約に関し，語声調による場合分けが不要である旨，述べた。上記の修正により，このことは，成り立たなくなった。

1.4 新しい用語の導入

次に，前稿までは用いていなかった用語を，導入する。

岡田(1990：170)の(7.19)で定義した $b_1(I)$ を，「境界領域1」と称する。同じく，$b_2(I)$ を，「境界領域2」と称する。両者を総称して，「境界領域」と呼び，「境界領域1」「境界領域2」における1，2を，「境界領域の番号」と呼ぶ。

岡田(1990)において，$b_1(I)$，$b_2(I)$ は，いずれも，2つのインデックスから成る対であった。本稿では，これを，2つのインデックスから成るドット対によって，表現する。

直観的に言えば，境界領域1は，ある音調パターンにおいて，インデックスの類が0である音調変動（すなわち，アクセント単位の始点を基準点として時間軸成分の決まる音調変動）のうち最も「右側」にあるもののインデックスと，インデックスの類が1である音調変動（すなわち，アクセント標識の存在するモーラの終点を基準点として時間軸成分の決まる音調変動）のうち最も「左側」にあるもののインデックスから成る対である。

境界領域2は，インデックスの類が0または1である音調変動のうち最も「右側」にあるもののインデックスと，インデックスの類が2である音調変動（すなわち，アクセント単位の終点を基準点として時間軸成分の決まる音調変動）のうち最も「左側」にあるもののインデックスから成る対である。

これに応じて，縮約についても，新しい用語を導入する。

縮約の操作の適用は，2回に分かれている。直観的に言えば，1回目の一連の縮約の操作は，インデックスの類が0である音調変動の集合とインデックスの類が1である音調変動の集合の間の衝突・交差の解消を，目的としている。

2回目の一連の縮約の操作は，インデックスの類が0または1である音調変動の集合とインデックスの類が2である音調変動の集合の間の衝突・交差の解消を，目的としている。

この，1回目の一連の縮約の操作を，「第1次の縮約」と呼ぶ。2回目の一連の縮約の操作を，「第2次の縮約」と呼ぶ。

2　関数の解説

2.0　序

この§2では，各関数を，直観的に解説する。

関数の定義は，§3で示す。ただし，直観的な解説を与えれば十分であると判断した関数は，§3に掲載しない。その場合は，以下の解説において，関数とそのパラメーターの表示の直後に，「掲載せず」と記す。

以下，§2.1において，本体，及び，それと密接に関連した関数を扱う。§2.2，§2.3，§2.4において，それぞれ，素性についての関数，音調変動についての関数，分数についての関数を扱う。§2.5において，その他の関数を扱う。§2.6において，各方言の特性を定める関数を扱う。

なお，この，関数の分類は，便宜上のものに過ぎない。

2.1　本体，及び，それに関連した関数

(tone dialect fv-list)

この関数が，言わば，本体である。方言名 dialect とアクセント素性構造 fv-list を受け取って，音調パターンを返す。関数内部の処理としては，まず，関数 phase1 により，縮約の適用された段階の音調パターン（岡田(1991：151)で定めた P_1）を算出し，その音調パターンに，変異規則，修正規則，表層規則を，この順で，適用する。変異規則と修正規則の適用後には，それぞれ，合算（関数 sum-tc）を適用する。変異規則，修正規則，表層規則については，関数 proc へ，必要な情報とともに，項目名として，それぞれ，variation, modification, surface-rule を渡すことで，それぞれの規則の適用された結果の音調パターンが返ってくる。

(phase1 dialect fv-list)

岡田(1991：151)で定めた P_1 を算出する。内部処理としては，まず，全体圧縮が適用されない場合は，関数 phase1-aux によって，音調パターンを算出し，それに等間隔化を適用する。全体圧縮が適用される場合は，素性構造 fv-list における素性 len の値を2に変更した素性構造を用いて，関数 phase1-aux により，音調パターンを算出し，それに全体圧縮を適用し，さらに，等間隔化を適用する。したがって，言わば，関数 phase1 の実質的な部分は，関数 phase1-aux が担っている。

(phase1-aux dialect fv-list)

関数 phase1 の内部で用い，phase1 の実質的な部分を担う。関数 phase0 によって算出された音調パターンに対し，関数 contract によって縮約を加え，関数 sum-tc によって，合算を加える。縮約については，第1次の縮約と第2次の縮約を，順次，適用する。

(phase0 dialect fv-list)

方言名 dialect とアクセント素性構造 fv-list を受け取って，岡田(1991：151)で定めた P_0（縮約を適用する直前の段階の音調パターン）を返す。なお，岡田(1991：151)では，P_0，P_1 のほかに，P_2，P_3 も定めた。しかし，今のところ，この2つを使用する機会はない。よって，これに対応する段階は，関数として設定しなかった。

(select-tc dialect fv-list)

関数 phase0 の内部で用いる。方言名 dialect とアクセント素性構造 fv-list を受け取って，当該方言における音調変動のリストの初期設定の中から，dialect と fv-list に応じた音調変動を選択し，リストにして返す。各音調変動の時間軸成分は，相対表示のままである。

(remove-class1 tc-list)

関数 select-tc の内部で用いる。音調変動のリスト tc-list の中から，インデックスの類が1である音調変動を取り除き，得られたリストを返す。

(assign-time val-mark val-len tc-list)

関数 phase0 の内部で用いる。素性 mark，len の値と，音調変動のリスト tc-list を受け取って，tc-list の各音調変動の時間軸成分の相対表示を，絶対表示に変換し，得られたリストを返す。

(contract border-number dialect fv-list tc-list)

関数 phase1-aux の内部で用いる。境界領域の番号 border-number，方言名 dialect，素性構造 fv-list，音調パターン tc-list を受け取って，tc-list に対し，border-number に応じて，第1次または第2次の縮約を加え，得られた音調パターンを返す。内部処理としては，まず，関数 get-info によって，縮約の適用に必要な情報を取得する。その情報を使用して，関数 count により，縮約の操作（除去・圧縮等）を適用すべき回数(number)を，算出する。それらの情報に基づいて，縮約を実行する。その際，関数 separate を使用する。separate は，言わば，境界領域を number だけ広げる操作を行う（直観的に言えば，境界領域を n モーラ広げてから，除去・圧縮等を合計 n 回適用する，というのが，縮約の操作である)。なお，縮約の各操作の適用直後には，合算(関数 sum-tc）を適用する。

(get-info border-number dialect fv-list tc-list)

関数 contract の内部で用いる。縮約の適用に必要な情報を返す。具体的には，関数

proc へ，項目名 contraction1 または contraction2（それぞれ，第 1 次，第 2 次の縮約に対応する），方言名 dialect，素性構造 fv-list，関数 get-border によって得られた境界領域，を送る。

(get-border border-number tc-list)

関数 get-info，及び，関数 border-displacement（後出）の内部で用いる。境界領域の番号 border-number と音調パターン tc-list を受け取って，当該音調パターンにおける，番号 border-number の境界領域を返す。

(count border-number condition tc-list)

関数 contract の内部で用いる。境界領域の番号 border-number，条件 condition，音調パターン tc-list を受け取って，縮約の操作（除去・圧縮等）を適用すべき回数を返す。

(border-displacement border-number tc-list)

関数 count の内部で用いる。境界領域の番号 border-number と音調パターン tc-list を受け取って，直観的に言えば，当該境界領域の「幅」（正負を考える）を返す。当該境界領域が存在しないときは，0 を返す。

(separate border-number addend tc-list)

関数 contract の内部で用いる。境界領域の番号 border-number，分数addend，音調パターン tc-list を受け取って，直観的に言えば，当該境界領域の「幅」を addend だけ広げ，得られた音調パターンを返す。

(delete index tc-list)

インデックス index，音調パターン tc-list を受け取って，index をパラメーターとする除去の操作を tc-list に適用し，得られた音調パターンを返す。この関数は，関数 proc において，縮約に関する情報を記述する際に用いる。これは，次の compress についても，同様である。

(compress index tc-list)

インデックス index，音調パターン tc-list を受け取って，index をパラメーターとする圧縮の操作を tc-list に適用し，得られた音調パターンを返す。

(compress-total tc-list)

音調パターン tc-list を受け取って，全体圧縮の操作を適用し，得られた音調パターンを返す。

(sum-tc dialect tc-list)

方言名 dialect，音調パターン tc-list を受け取って，tc-list（において隣接する 2 つの音調変動から成る各対）に合算の操作を適用し，得られた音調パターンを返す。2 つの異なる音調変動 tc1，tc2 に合算が適用されるのは，その時間軸成分が等しいときに

限られる。そして，(a)それぞれの音調変動セグメント成分をドット対にしたものがsummation-list1の要素として存在し，かつ，合算の禁止に該当しない，または，(b)それぞれの音調変動セグメント成分をドット対にしたものがsummation-list2の要素として存在する，のいずれかが満たされていることが，適用条件である。(a)の場合，2つの音調変動は，相殺される。(b)の場合，2つの音調変動のうちの前者が消え，後者が残る。

(prohibited? index1 index2 dialect)

関数sum-tcの内部で用いる。インデックスindex1，index2，方言名dialectを受け取って，index1，index2をそれぞれインデックスとする2つの音調変動の合算の禁止の有無を判定する。

summation-list1

summation-list2

いずれも，関数sum-tcの内部で用いるリストである(この2つのみ，関数ではない)。リストの各要素は，2つの音調変動セグメントから成るドット対である。この2つのリストの機能については，関数sum-tcを解説した際に述べた。

(moved? index dialect fv-list tc-list)

岡田(1991：152)で定めた関数movedと同じ機能を果たす。インデックスindex，方言名dialect，素性構造fv-list，音調パターンtc-listを受け取って，直観的に言えば，dialectとfv-listから算出された音調パターンtc-listにおいて，indexをインデックスとする音調変動が何らかの規則によって移動させられているかどうかを，判定する。

(moved-pl? index dialect fv-list)

関数moved?におけるtc-listを，(phase1 dialect fv-list)で置き換えたものである。すなわち，直観的に言えば，dialectとfv-listから算出されたphase1の段階の音調パターンにおいて，indexをインデックスとする音調変動が何らかの規則によって移動させられているかどうかを，判定する。関数内部で，関数moved?を使用する。

(equalize tc-list)(掲載せず)

関数phase1の内部で用いる。音調パターンtc-listに等間隔化の操作を加え，得られた音調パターンを返す。等間隔化の操作は，岡田(1990：148-149)で定義した。この操作の内容は，直観的には，明解である。しかし，厳密に定義を書くと，煩雑なものになる。そのため，それほど重要な操作ではないということもあって，掲載を省略する。

2.2 素性についての関数

(val feature fv-list)

素性構造fv-listにおける素性featureの値を返す。素性featureの値が指定されてい

なければ，空リストを返す。

(eqval? feature value fv-list)

素性構造 fv-list における素性 feature の値が value に等しいかどうかを判定する。

(specified? feature fv-list)

素性構造 fv-list において，素性 feature の値が指定されているかどうかを判定する。

(unspecified? feature fv-list)

specified? の否定。

(subst-val feature value fv-list)

素性構造 fv-list において，素性 feature の値が指定されているならば，素性 feature の値を，value に変更し，得られた素性構造を返す。素性 feature の値が指定されていないならば，fv-list をそのまま返す。

(remove-pair feature fv-list)

素性構造 fv-list から，素性 feature とその値から成るリスト（すなわち，素性 feature に関する素性指定）を取り除き，得られた素性構造を返す。

(select-pair feature-list fv-list)

素性構造 fv-list から，素性のリスト feature-list の中に含まれている素性に関する素性指定のみを取り出して，リストにして返す。取り出す順序は，fv-list における順序を保持する。

2.3 音調変動についての関数

(index-comp tc)（掲載せず）

音調変動 tc のインデックスを返す。

(class-comp tc)（掲載せず）

音調変動 tc のインデックスの類を返す。

(time-comp tc)（掲載せず）

音調変動 tc の時間軸成分を返す。

(segment-comp tc)（掲載せず）

音調変動 tc の音調変動セグメント成分を返す。

(get-tc index tc-list)（掲載せず）

音調パターン tc-list の中から，インデックスを index とする音調変動を取り出して，返す。

(add-time addend tc)（掲載せず）

音調変動 tc の時間軸成分に分数 addend を加えた音調変動を返す。

(subtract-time subtrahend tc)（掲載せず）

音調変動 tc の時間軸成分から分数 subtrahend を減じた音調変動を返す。

(subst-time time tc)（掲載せず）

音調変動 tc の時間軸成分を time に置き換えた音調変動を返す。

(subst-segment segment tc)（掲載せず）

音調変動 tc の音調変動セグメント成分を segment に置き換えた音調変動を返す。

(displacement index-pair tc-list)

index-pair は，2つのインデックスから成るドット対である。仮に，これを，(index1 . index2) とおく。音調パターン tc-list において，index1，index2 をインデックスとする音調変動を，仮に，それぞれ，tc1，tc2 とおく。このとき，関数 displacement は，tc2 の時間軸成分から tc1 の時間軸成分を減じた値を返す。tc1，tc2 のいずれかが存在しなければ，空リストを返す。

(move index-d-list tc-list)

音調パターン tc-list において，index-d-list に定められた情報に基づいて，移動を実行し，得られた音調パターンを返す。index-d-list は，例えば，

((i1 . (−1 . 1)) (i2 . (−1 . 2)))

のような形のリストである。リストの各要素は，(index . ratio) という形をとる。index は，移動すべき音調変動のインデックスである。ratio は，移動すべき量（正負を考える）である。上記の例は，インデックスを i1 とする音調変動を −1 移動し，インデックスを i2 とする音調変動を −1/2 移動することを示す。なお，例えば，2つのリスト

(i2 . (−1 . 2)), (i2 −1 . 2)

は，同じものである。かつ，後者の表現の方が，簡潔である。しかし，これらは，データの型として分数を有する Lisp を使用する場合には，

(i2 . −1/2)

と表現することになるはずのものである。よって，前者の形のままとする。

2.4 分数についての関数

この§2.4 で解説する関数の定義は，いずれも，§3 に掲載しない。「掲載せず」の表示は，省略する。

(numer rat)

分数 ratio の分子を返す。

(denom rat)

分数 ratio の分母を返す。

(make-rat int1 int2)

分子を整数 int1 とし分母を整数 int2 とする分数を返す。

(integer->rat int)

整数 int を，分子が int で分母が 1 の分数に変換する。

(rat-quotient ratio)

分数 ratio の分子を分母で除し，小数点以下を切り捨てた値を返す。

(reduce-rat ratio)

分数 ratio を約分する。分母は正数とする。

(rat+ ratio1 ratio2)

分数 ratio1 に，分数 ratio2 を加えて，約分する。約分は，関数 reduce-rat による(以下同じ)。

(rat1+ ratio)

分数 ratio に，1 を加えて，約分する。

(rat- ratio1 ratio2)

分数 ratio1 から，分数 ratio2 を減じて，約分する。

(rat1- ratio)

分数 ratio から，1 を減じて，約分する。

(rat* ratio1 ratio2)

分数 ratio1 に，分数 ratio2 を乗じて，約分する。

(rat2/ratio)

分数 ratio を，2 で除して，約分する。

(rat=? ratio1 ratio2)

分数 ratio1 と分数 ratio2 が等しいかどうかを判定する。

(rat>? ratio1 ratio2)

分数 ratio1 が分数 ratio2 より大きいかどうかを判定する。

(rat>=? ratio1 ratio2)

分数 ratio1 が分数 ratio2 以上かどうかを判定する。

(rat<? ratio1 ratio2)

分数 ratio1 が分数 ratio2 より小さいかどうかを判定する。

(rat<=? ratio1 ratio2)

分数 ratio1 が分数 ratio2 以下かどうかを判定する。

(rat-integer? ratio)

分数 ratio の分母が 1 に等しいかどうか(すなわち，事実上整数であるかどうか)を判定する。

(rat-not-integer? ratio)

rat-integer? の否定。

2.5 その他の関数

(not-touch? ratio)

分数 ratio が 0 より大きいかどうかを判定する。したがって，広く使える。ただし，プログラムでは，音調変動の「衝突」(これを，“touch” とした) がないことを判定するためにしか用いない。そのため，実態を反映させる関数名とした。

(not-cross? ratio)

分数 ratio が 0 以上かどうかを判定する。not-touch と同様の理由で，関数名を not-cross とした (“cross” は，「交差」)。

(tautology x)

x が何であっても，#t (すなわち，真) を返す。

2.6 各方言の特性を定める関数

各方言の特性は，関数 proc のみによって，定める。これ以外の関数は，直接には，方言名に依存する処理を行わない。この措置により，一般的な枠組みと，方言固有の事項とが，明確に分離される。

これまで述べてきた関数と同じ形式で，proc をパラメーターとともに示すと，次のようになる (proc は，“procedure” の略である。ただし，ここで言う “procedure” に，特別の意味はない)。

(proc item dialect . rest)

proc は，項目名 item，方言名 dialect，及び，その他の情報(項目名によって必要な数が異なる) を受け取って，何らかの情報 (項目名によって性質が異なる) を返す。「その他の情報」は，残余パラメーター rest にバインドされる。

proc においては，方言名によって，処理の分岐を行う。そして，各方言ごとに，9 つの項目を設ける。以下，各項目について，直観的に解説する。項目名の直後の括弧の中に，方言名のほかに必要な情報 (すなわち，残余パラメーターにバインドされるべき情報) を付記する。

tc-list-init

当該方言における初期設定たる音調変動のリストを返す。

make-index-list (素性構造 fv-list)

与えられた方言名と素性構造に応じて，インデックスのリストを返す。返すリストは，岡田(1990：142)で定めた I_t, I_p, I_k の合併集合をリストに置き換えたものに相当する。ただし，I_t, I_p, I_k に相当する区別は，設けなかった。§2.1 で述べた関数 select-tc は，前項 tc-list-init の結果と，本項 make-index-list の結果を用いて，音調変動の選択を行

う。

total-compression

当該方言における全体圧縮の適用の有無を，真偽値で返す。

prohibition

合算の禁止されている音調変動のインデックスのドット対から成るリストを返す。合算の禁止がなければ，空リストを返す。

contraction1（素性構造 fv-list，境界領域 border）

第1次の縮約に関する情報を返す。返されるリストは，例えば，

(not-touch? compress p2 delete p1)

のような形をしている。これは，「衝突を解消すべく(not-touch?)，p2 をパラメーターとする圧縮(compress)，p1 をパラメーターとする除去(delete)を，この順に，適用する」という内容である。すなわち，リストの最初の要素は，縮約が適用されないための条件である。第2要素以降は，縮約が順次適用される場合の，縮約の操作の種類とそのパラメーターが，順次，交互に並ぶ。いかなる条件下でも縮約が適用されない場合に返されるリストは，

(tautology)

である。この場合，第2要素以降は，当然，不要である。なお，除去・圧縮以外の操作が必要な場合は，その操作を関数の形にして，操作の種類の位置に置けばよい。例えば，佐柳島方言において，

```
(lambda (index tc-list)
  (move '((t22 . (-1 . 2))
          (k   . (-1 . 2))
          (e2  . (-1 . 1)))
        tc-list))
```

という操作の例がある。この場合，操作のパラメーターたるインデックスは，事実上，不要である。しかし，形式を整える必要がある。そこで，このような場合は，パラメーターとして，便宜上，常に，e2 を設定する。

contraction2（素性構造 fv-list，境界領域 border）

第2次の縮約に関する情報を返す。その他については，contraction1 と同様である。

variation（素性構造 fv-list，音調パターン tc-list）

tc-list に変異規則を適用し，得られた音調パターンを返す。したがって，当該方言に変異規則が存在しないときは，tc-list をそのまま返す(これについては，次の modification, surface-rule においても，同様である)。

modification（素性構造 fv-list，音調パターン tc-list）

tc-list に修正規則を適用し，得られた音調パターンを返す。

surface-rule（素性構造 fv-list，音調パターン tc-list）

tc-list に表層規則を適用し，得られた音調パターンを返す。

3 関数の定義

3.0 序

以下，関数の定義を示す。便宜上，4つの節に分けて掲載する。次のとおりである。§3.3は，§2.3,§2.4,§2.5に相当する。§3.4は，§2.6に相当する。

§3.1 本体，及び，それに関連した関数

§3.2 素性についての関数

§3.3 音調変動についての関数，及び，その他の関数

§3.4 各方言の特性を定める関数

§2.0で述べたとおり，あえて定義を示す必要がないと判断した関数は，掲載しない。そのような関数については，それぞれの節の冒頭に，名称のみを記す。

3.1 本体，及び，それに関連した関数

掲載しない関数：equalize

```
(define (tone dialect fv-list)
  (let ((tc-list (phase1 dialect fv-list)))
    (set! tc-list
      (sum-tc
        dialect
        (proc 'variation dialect fv-list tc-list)))
    (set! tc-list
      (sum-tc
        dialect
        (proc 'modification dialect fv-list tc-list)))
    (set! tc-list
      (proc 'surface-rule dialect fv-list tc-list))
    tc-list))

(define (phase1 dialect fv-list)
  (equalize
```

```
    (if (and (proc 'total-compression dialect)
             (eqval? 'len 1 fv-list))
        (compress-total
          (phase1-aux dialect (subst-val 'len 2 fv-list)))
        (phase1-aux dialect fv-list))))

(define (phase1-aux dialect fv-list)
  (let ((tc-list (phase0 dialect fv-list)))
    (set! tc-list (contract 1 dialect fv-list tc-list))
    (set! tc-list (contract 2 dialect fv-list tc-list))
    (sum-tc dialect tc-list)))

(define (phase0 dialect fv-list)
  (assign-time (val 'mark fv-list)
               (val 'len  fv-list)
               (select-tc dialect fv-list)))

(define (select-tc dialect fv-list)
  (let ((index-list '( ))
        (tc-list '( )))
    (set! index-list
      (append '(e1 e2)
              (proc 'make-index-list dialect fv-list)))
    (set! tc-list
      (select-pair index-list
                   (proc 'tc-list-init dialect)))
    (if (eqval? 'acc 'f fv-list)
        (remove-class1 tc-list)
        tc-list)))

(define (remove-class1 tc-list)
  (let ((tc (car tc-list)))
    (case (class-comp tc)
      ((0) (cons tc (remove-class1 (cdr tc-list))))
      ((1) (remove-class1 (cdr tc-list)))
      ((2) tc-list))))
```

```
(define (assign-time val-mark val-len tc-list)
  (map (lambda (tc)
         (case (class-comp tc)
           ((0) tc)
           ((1) (add-time (integer->rat val-mark) tc))
           ((2) (add-time (integer->rat val-len) tc))))
       tc-list))

(define (contract border-number dialect fv-list tc-list)
  (let* ((info (get-info border-number dialect fv-list tc-list))
         (condition (car info))
         (number (count border-number condition tc-list)))
    (if (zero? number)
        tc-list
        (let ((tc-list2 (separate border-number number tc-list)))
          (do ((func-index-list
                  (cdr info) (cddr func-index-list))
               (counter number (1- counter)))
              ((zero? counter) tc-list2)
            (let ((func (car func-index-list))
                  (index (cadr func-index-list)))
              (set! tc-list2 (func index tc-list2))
              (set! tc-list2
                (sum-tc dialect tc-list2))))))))

(define (get-info border-number dialect fv-list tc-list)
  (case border-number
    ((1) (proc 'contraction1
               dialect
               fv-list
               (get-border 1 tc-list)))
    ((2) (proc 'contraction2
               dialect
               fv-list
               (get-border 2 tc-list)))))
```

```
(define (get-border border-number tc-list)
  (let* ((tc (cadr tc-list))
         (class (class-comp tc)))
    (cond ((< class border-number)
           (get-border border-number (cdr tc-list)))
          ((= class border-number)
           (cons (index-comp (car tc-list))
                 (index-comp tc)))
           (else '( )))))

(define (count border-number condition tc-list)
  (if (eq? condition tautology)
      0
      (do ((counter 0 (1+ counter))
           (d (border-displacement border-number tc-list)
              (rat1+ d)))
          ((condition d) counter))))

(define (border-displacement border-number tc-list)
  (let ((border (get-border border-number tc-list)))
    (if (null? border)
        (integer->rat 0)
        (displacement border tc-list))))

(define (separate border-number addend tc-list)
  (map (lambda (tc)
         (if (>= (class-comp tc) border-number)
             (add-time (integer->rat addend) tc)
             tc))
       tc-list))

(define (delete index tc-list)
  (let* ((tc-ref (get-tc index tc-list))
         (time-ref (time-comp tc-ref)))
    (map (lambda (tc)
           (cond ((rat<=? (time-comp tc) (rat1- time-ref))
```

```
                    tc)
                   ((rat>=? (time-comp tc) time-ref)
                    (subtract-time (integer->rat 1) tc))
                   (else
                     (subst-time (rat1- time-ref) tc))))
          tc-list)))

(define (compress index tc-list)
   (let* ((tc-ref (get-tc index tc-list))
          (time-ref (time-comp tc-ref)))
    (map (lambda (tc)
           (cond ((rat<=? (time-comp tc) (rat1- time-ref))
                   tc)
                  ((rat>=? (time-comp tc) (rat1+ time-ref))
                   (subtract-time (integer->rat 1) tc))
                  (else
                    (subst-time (rat2/ (rat+ (rat1- time-ref)
                                             (time-comp tc)))
                                tc))))
          tc-list)))

(define (compress-total tc-list)
  (map (lambda (tc)
         (subst-time (rat2/ (time-comp tc)) tc))
       tc-list))

(define (sum-tc dialect tc-list)
  (if (null? (cdr tc-list))
     tc-list
     (let ((tc1 (car  tc-list))
           (tc2 (cadr tc-list)))
        (if (equal? (time-comp tc1) (time-comp tc2))
           (cond ((and
                     (member (cons (segment-comp tc1)
                                   (segment-comp tc2))
```

```
                    summation-list1)
                (not (prohibited? (index-comp tc1)
                                  (index-comp tc2)
                                  dialect)))
            (sum-tc dialect (cddr tc-list)))
           ((member (cons (segment-comp tc1)
                          (segment-comp tc2))
                    summation-list2)

            (sum-tc dialect (cdr tc-list)))
           (else (cons tc1
                       (sum-tc dialect (cdr tc-list)))))
      (cons tc1
            (sum-tc dialect (cdr tc-list)))))))

(define (prohibited? index1 index2 dialect)
  (member (cons index1 index2)
          (proc 'prohibition dialect)))

(define summation-list1
  '((neg . pos) (pos . neg) (neg2 . pos2) (pos2 . neg2)))

(define summation-list2
  '((neg2 . neg)))

(define (moved? index dialect fv-list tc-list)
  (let ((tc-list0 (phase0 dialect fv-list)))
    (cond ((unspecified? index tc-list0) #f)
          ((unspecified? index tc-list) #t)
          ((not (rat=? (time-comp (get-tc index tc-list))
                       (time-comp (get-tc index tc-list0))))
           #t)
          (else #f))))

(define (moved-p1? index dialect fv-list)
  (moved? index dialect fv-list (phase1 dialect fv-list)))
```

3.2 素性についての関数

```
(define (val feature fv-list)
  (let ((fv (assv feature fv-list)))
    (if (null? fv)
        '( )
        (cadr fv))))

(define (eqval? feature value fv-list)
  (equal? (val feature fv-list) value))

(define (specified? feature fv-list)
  (not (null? (assv feature fv-list))))

(define (unspecified? feature fv-list)
  (null? (assv feature fv-list)))

(define (subst-val feature value fv-list)
  (cond ((null? fv-list) '( ))
        ((eqv? (caar fv-list) feature)
         (cons (list feature value) (cdr fv-list)))
        (else
          (cons (car fv-list)
                (subst-val feature value (cdr fv-list))))))

(define (remove-pair feature fv-list)
  (cond ((null? fv-list)
         '( ))
        ((eqv? (caar fv-list) feature)
         (remove-pair feature (cdr fv-list)))
        (else (cons (car fv-list)
                    (remove-pair feature (cdr fv-list))))))

(define (select-pair feature-list fv-list)
  (cond ((null? fv-list)
         '( ))
        ((memv (caar fv-list) feature-list)
```

```
        (cons (car fv-list)
              (select-pair feature-list (cdr fv-list))))
      (else (select-pair feature-list (cdr fv-list)))))
```

3.3 音調変動についての関数，及び，その他の関数

掲載しない関数：

index-comp, class-comp, time-comp, segment-comp, get-tc, add-time, subtract-time, subst-time, subst-segment, numer, denom, make-rat, integer->rat, rat-quotient, reduce-rat, rat+, rat1+, rat−, rat1−, rat*, rat2/, rat=?, rat>?, rat>=?, rat<?, rat<=?, rat-integer?, rat-not-integer?

```
(define (displacement index-pair tc-list)
  (let  ((tc1 (get-tc (car index-pair) tc-list))
         (tc2 (get-tc (cdr index-pair) tc-list)))
   (if  (or (null? tc1) (null? tc2))
        '( )
        (rat− (time-comp tc2) (time-comp tc1)))))

(define (move index-d-list tc-list)
  (if (null? tc-list)
      '( )
      (map (lambda (tc)
             (let ((index-d-pair
                    (assq (index-comp tc) index-d-list)))
               (if (null? index-d-pair)
                   tc
                   (add-time (cdr index-d-pair) tc))))
           tc-list)))

(define (not-touch? ratio)
  (rat>? ratio (integer->rat 0)))

(define (not-cross? ratio)
  (rat>=? ratio (integer->rat 0)))
```

```
(define (tautology x) #t)
```

3.4 各方言の特性を定める関数

```
(define (proc item dialect . rest)
  (let ((fv-list '( ))
        (tc-list '( ))
        (border '( )))
    (case item
      ((contraction1 contraction2)
       (set! fv-list (car rest))
       (set! border (cadr rest)))
      ((variation modification surface-rule)
       (set! fv-list (car rest))
       (set! tc-list (cadr rest)))
      ((make-index-list)
       (set! fv-list (car rest)))
      ((tc-list-init total-compression prohibition)))
    (case dialect
      ;; 東京方言
      ((tokyo)
       (case item
         ((tc-list-init)
          '((e1 0 (0 . 1) e) (p1 0 (0 . 1) neg) (p2 0 (1 . 1) pos)
            (k  1 (0 . 1) neg)
            (e2 2 (0 . 1) e)))
         ((make-index-list)
          (append
            '(k)
            (if (eqval? 'initial 't fv-list)
                '(p1 p2))))
         ((total-compression) #f)
         ((prohibition) '( ))
         ((contraction1)
```

```
      (cond ((equal? border '(p2 . k))
             (list not-touch?
                   delete 'p2))
            (else (list tautology))))
    ((contraction2)
      (cond ((equal? border '(p2 . e2))
             (list not-touch?
                   delete 'p2))
            (else (list tautology))))
    ((variation) tc-list)
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 京都方言
((kyoto)
 (case item
    ((tc-list-init)
      '((e1 0   (0 . 1) e)   (t1 0 (0 . 1) neg)
        (t2 1 (-1 . 1) pos) (k  1 (0 . 1) neg)
        (p  2 (-1 . 1) pos) (e2 2 (0 . 1) e)))
    ((make-index-list)
     (append
       (if (eqval? 'tone 2 fv-list)
           '(t1 t2))
       '(k)
       (if (and (eqval? 'tone  2 fv-list)
                (eqval? 'acc   'f fv-list)
                (eqval? 'final  't fv-list))
           '(p))))
    ((total-compression) #t)
    ((prohibition) '( ))
    ((contraction1) (list tautology))
    ((contraction2)
```

```
      (cond ((equal? border '(k . e2))
             (list not-touch?
                   compress 'k))
             (else (list tautology))))
    ((variation) tc-list)
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 久見方言
((kumi)
 (case item
    ((tc-list-init)
      '((e1  0    (0 . 1) e)    (t21 0 (0 . 1) neg)
        (t1  0    (1 . 1) neg) (t22 0 (1 . 1) pos)
        (t31 0    (1 . 1) neg) (t23 0 (2 . 1) neg)
        (t32 0    (3 . 1) pos) (t33 0 (4 . 1) neg)
        (p   2 (-1 . 2) pos) (e2  2 (0 . 1) e)))
    ((make-index-list)
     (append
       (case (val 'tone fv-list)
         ((1) '(t1))
         ((2) '(t21 t22 t23))
         ((3) '(t31 t32 t33)))
       (if (and (eqval? 'tone  1 fv-list)
                (eqval? 'final  't fv-list))
           '(p))))
    ((total-compression) #t)
    ((prohibition) '( ))
    ((contraction1) (list tautology))
    ((contraction2)
     (cond ((equal? border '(t23 . e2))
            (list not-touch?
                  delete 't22))
```

```
            ((equal? border '(t33 . e2))
             (list not-touch?
                   delete 't32
                   delete 'e2
                   delete 't31))
            (else (list tautology))))
    ((variation) tc-list)
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 伊吹島方言
((ibukijima)
 (case item
    ((tc-list-init)
      '((e1 0   (0 . 1) e)     (t1 0  (0 . 1) neg)
        (t3 0   (2 . 1) neg2)
        (t2 1 (-1 . 1) pos)  (k  1  (0 . 1) neg)
        (p  2 (-1 . 1) pos)  (e2 2  (0 . 1) e)))
    ((make-index-list)
     (append
        (case (val 'tone fv-list)
          ((2) '(t1 t2))
          ((3) '(t3)))
        '(k)
        (if (and (eqval? 'tone  2 fv-list)
                 (eqval? 'acc   'f fv-list)
                 (eqval? 'final  't fv-list))
            '(p))))
    ((total-compression) #t)
    ((prohibition) '( ))
    ((contraction1) (list tautology))
    ((contraction2)
     (cond ((equal? border '(k . e2))
            (list not-touch?
                  compress 'k))
```

```
            ((equal? border '(t3 . e2))
             (list not-touch?
                   delete 't3))
            (else (list tautology))))
    ((variation) tc-list)
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 村上方言
((murakami)
 (case item
    ((tc-list-init)
      '((e1 0    (0 . 1) e)    (k1 0 (0 . 1)  neg)
        (p1 0    (1 . 1) pos) (p2 0 (2 . 1)  neg)
        (k2 1 (-1 . 1) pos) (k3 1 (1 . 1)  neg)
        (e2 2    (0 . 1) e)))
    ((make-index-list)
     (append
        '(k1 k2 k3)
        (if (eqval? 'initial 't fv-list)
            '(p1 p2))))
    ((total-compression) #f)
    ((prohibition) '((p2 . k2)))
    ((contraction1)
     (cond ((equal? border '(p2 . k2))
            (list not-cross?
                  delete 'p2
                  delete 'k2))
            (else (list tautology))))
    ((contraction2)
     (cond ((equal? border '(k3 . e2))
            (list not-touch?
                  delete 'k3
                  delete 'e2))
```

```
            ((equal? border '(p2 . e2))
             (list not-touch?
                   compress 'p2
                   delete 'p1))
            (else (list tautology))))
    ((variation) tc-list)
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 広島方言
((hiroshima)
 (case item
    ((tc-list-init)
     '((e1  0    (0 . 1) e)      (t21 0    (0 . 1) neg)
       (t22 1 (-1 . 1) pos)  (k   1    (0 . 1) neg)
       (p1  2 (-1 . 1) neg2) (p2  2 (-1 . 1) pos2)
       (e2  2    (0 . 1) e)))
    ((make-index-list)
     (append
       (if (eqval? 'tone 2 fv-list)
           '(t21 t22))
       '(k)
       (if (and (eqval? 'tone      1 fv-list)
                (eqval? 'acc       'f fv-list)
                (eqval? 'particle  't fv-list))
           '(p1))
       (if (and (eqval? 'tone  2 fv-list)
                (eqval? 'acc   'f fv-list)
                (eqval? 'final  't fv-list))
           '(p2))))
    ((total-compression) #t)
    ((prohibition) '( ))
    ((contraction1) (list tautology))
    ((contraction2) (list tautology))
    ((variation) tc-list)
```

```
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 志々島方言
((shishijima)
 (case item
    ((tc-list-init)
      '((e1  0    (0 . 1) e)     (p11 0 (0 . 1) neg2)
        (t21 0    (0 . 1) neg)  (p12 0 (1 . 1) pos2)
        (t22 1 (-1 . 1) pos)  (k    1 (0 . 1) neg)
        (p2  2 (-1 . 1) pos2) (e2   2 (0 . 1) e)))
    ((make-index-list)
     (append
        (if (eqval? 'tone 2 fv-list)
            '(t21 t22))
        '(k)
        (if (and (eqval? 'tone    1  fv-list)
                 (eqval? 'initial  't fv-list))
            '(p11 p12))
        (if (and (eqval? 'tone  2  fv-list)
                 (eqval? 'acc    'f fv-list)
                 (eqval? 'final  't fv-list))
            '(p2))))
    ((total-compression) #t)
    ((prohibition) '( ))
    ((contraction1)
     (cond ((equal? border '(p12 . k))
            (list not-touch?
                  delete 'p12))
            (else (list tautology))))
    ((contraction2) (list tautology))
    ((variation) tc-list)
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 佐柳島方言
```

```
((sanagijima)
 (case item
   ((tc-list-init)
    '((e1  0   (0 . 1) e)     (t11  0 (0 . 1) neg2)
      (p11 0   (0 . 1) neg2) (t21  0 (0 . 1) neg)
      (t12 0   (1 . 1) pos2) (p12 0 (2 . 1) pos2)
      (p13 0   (3 . 1) neg)
      (t22 1   (0 . 1) pos)  (k    1 (1 . 2) neg)
      (p2  2 (-1 . 2) pos)  (e2   2 (0 . 1) e)))
   ((make-index-list)
    (append
      (if (and (eqval? 'tone 1 fv-list)
               (eqval? 'acc  't fv-list))
          '(t11 t12))
      (if (eqval? 'tone 2 fv-list)
          '(t21 t22))
      '(k)
      (if (and (eqval? 'tone    1 fv-list)
               (eqval? 'acc     'f fv-list)
               (eqval? 'initial 't fv-list))
          '(p11 p12))
      (if (and (eqval? 'tone  1 fv-list)
               (eqval? 'acc   'f fv-list)
               (eqval? 'final 't fv-list))
          '(p13))
      (if (and (eqval? 'tone  2 fv-list)
               (eqval? 'acc   'f fv-list)
               (eqval? 'final 't fv-list))
          '(p2))))
   ((total-compression) #f)
   ((prohibition) '( ))
   ((contraction1) (list tautology))
   ((contraction2)
```

```
(cond ((equal? border '(p12 . e2))
       (list not-touch?
             delete 'p12
             compress 'p12))
      ((equal? border '(p13 . e2))
       (if (eqval? 'initial 't fv-list)
           (list not-cross?
                 delete 'p12
                 compress 'p12)
           (list not-cross?
                 delete 'p13
                 delete 'p13)))
      ((equal? border '(k . e2))
       (if (eqval? 'tone 1 fv-list)
           (list not-touch?
                 delete 't12)
           (list not-touch?
                 (lambda (index tc-list)
                     (move '((t22 . (-1 . 2))
                             (k   . (-1 . 2))
                             (e2  . (-1 . 1)))
                           tc-list))
                 'e2)))
      (else (list tautology))))
((variation)
 (if (eqval? 'var1 1 fv-list)
     (set! tc-list
         (move '((p12 . (-1 . 1))
                 (p13 . (-1 . 1)))
               tc-list)))
 (if (eqval? 'var2 1 fv-list)
     (set! tc-list
```

```
            (move '((t12 . (-1 . 1))
                    (t22 . (-1 . 2))
                    (k   . (-1 . 2)))
                  tc-list)))
      tc-list)
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 岩黒島方言
((iwagurojima)
 (case item
   ((tc-list-init)
    '((e1  0   (0 . 1) e)     (t11 0 (0 . 1) neg2)
      (t21 0   (0 . 1) neg)   (t12 0 (1 . 1) pos)
      (p1  0   (4 . 1) neg)
      (t22 1   (0 . 1) pos)   (k   1 (1 . 2) neg)
      (p2  2 (-1 . 1) pos2) (e2  2 (0 . 1) e)))
   ((make-index-list)
    (append
      (if (and (eqval? 'tone    1 fv-list)
               (eqval? 'acc     't fv-list)
               (eqval? 'initial  't fv-list))
          '(t11 t12))
      (if (eqval? 'tone 2 fv-list)
          '(t21 t22))
      '(k)
      (if (and (eqval? 'tone  1 fv-list)
               (eqval? 'acc   'f fv-list)
               (eqval? 'final  't fv-list))
          '(p1))
      (if (and (eqval? 'tone  2 fv-list)
               (eqval? 'acc   'f fv-list)
               (eqval? 'final  't fv-list))
          '(p2))))
   ((total-compression) #f)
```

```
((prohibition) '( ))
((contraction1) (list tautology))
((contraction2)
 (cond ((equal? border '(p1 . e2))
        (list not-touch?
              delete 'p1
              delete 'e2
              delete 'p1
              delete 'p1))
       ((equal? border '(t21 . p2))
        (list not-touch?
              compress 'p2))
       ((equal? border '(k . e2))
        (if (eqval? 'tone 1 fv-list)
            (if (eqval? 'initial 't fv-list)
                (list not-touch?
                      delete 't12)
                (list not-touch?
                      delete 'k))
            (list not-touch?
                  (lambda (index tc-list)
                    (move '((t22 . (-1 . 1))
                            (k   . (-1 . 2))
                            (e2  . (-1 . 1)))
                          tc-list))
                  'e2)))
       (else (list tautology))))
((variation)
 (if (eqval? 'var2 1 fv-list)
     (move '((t12 . (-1 . 1))
             (t22 . (-1 . 1))
             (k   . (-1 . 2)))
             tc-list)
     tc-list))
```

```
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 真鍋島本浦方言
((manabeshima-honura)
 (case item
    ((tc-list-init)
     '((e1  0    (0 . 1) e)    (t11 0 (0 . 1) neg2)
       (t21 0    (0 . 1) neg) (t12 0 (1 . 1) pos2)
       (p1  0    (4 . 1) neg)
       (t22 1    (0 . 1) pos) (k   1 (1 . 2) neg)
       (p2  2 (-1 . 2) pos) (e2  2 (0 . 1) e)))
    ((make-index-list)
     (append
       (if (and (eqval? 'tone    1 fv-list)
                (eqval? 'acc     't fv-list)
                (eqval? 'initial  't fv-list))
           '(t11 t12))
       (if (eqval? 'tone 2 fv-list)
           '(t21 t22))
       '(k)
       (if (and (eqval? 'tone 1 fv-list)
                (eqval? 'acc  'f fv-list))
           '(p1))
       (if (and (eqval? 'tone  2 fv-list)
                (eqval? 'acc    'f fv-list)
                (eqval? 'final  't fv-list))
           '(p2))))
    ((total-compression) #f)
    ((prohibition) '( ))
    ((contraction1) (list tautology))
    ((contraction2)
     (cond ((equal? border '(p1 . e2))
```

```
            (list not-cross?
                delete 'p1
                delete 'p1
                delete 'p1))
          ((equal? border '(k . e2))
           (if (eqval? 'tone 1 fv-list)
               (if (eqval? 'initial 't fv-list)
                   (list not-touch?
                         delete 't12)
                   (list not-touch?
                         delete 'k))
               (list not-touch?
                     (lambda (index tc-list)
                       (move '((t22 . (-1 . 2))
                               (k   . (-1 . 2))
                               (e2  . (-1 . 1)))
                             tc-list))
                     'e2)))
          (else (list tautology))))
  ((variation) tc-list)
  ((modification)
   (if (and (specified? 'k tc-list)
            (rat=? (time-comp (get-tc 'k tc-list))
                   '(5 . 2))
            (rat>=? (displacement '(k . e2) tc-list)
                    '(1 . 1)))
       (set! tc-list
             (move '((t22 . (1 . 1))
                     (k   . (1 . 1)))
                   tc-list)))
   (if (and (specified? 't22 tc-list)
            (rat=? (displacement '(t22 . e2) tc-list)
                   '(1 . 2))
            (eqval? 'final 'f fv-list))
```

```
        (set! tc-list
            (move '((t22 . (-1 . 2)))
                    tc-list)))
    tc-list)
  ((surface-rule) tc-list)))
;; 真鍋島岩坪方言
((manabeshima-iwatsubo)
 (case item
   ((tc-list-init)
    '((e1  0   (0 . 1) e)     (t21  0 (0 . 1) neg)
      (p11 0   (0 . 1) neg2) (p12 0 (1 . 1) pos2)
      (t11 0   (4 . 1) neg2) (t12  0 (5 . 1) neg)
      (t22 1   (0 . 1) pos)  (t23  1 (1 . 2) pos)
      (k1  1   (1 . 2) neg)  (k2   1 (1 . 1) neg)
      (p2  2 (-1 . 2) pos)  (e2   2 (0 . 1) e)))
   ((make-index-list)
    (append
      (if (and (eqval? 'tone 1  fv-list)
               (eqval? 'acc  'f fv-list))
          '(t11 t12))
      (if (eqval? 'tone 2 fv-list)
          (if (eqval? 'final 'f fv-list)
              '(t21 t22)
              '(t21 t23)))
      (if (eqval? 'tone 1 fv-list)
          '(k1))
      (if (eqval? 'tone 2 fv-list)
          '(k2))
      (if (and (eqval? 'tone    1 fv-list)
               (eqval? 'acc     't fv-list)
               (eqval? 'initial  't fv-list))
          '(p11 p12))
```

```
      (if (and (eqval? 'tone 2 fv-list)
               (eqval? 'acc  'f fv-list)
               (eqval? 'final 't fv-list))
          '(p2))))
  ((total-compression) #f)
  ((prohibition) '( ))
  ((contraction1) (list tautology))
  ((contraction2)
   (cond ( (equal? border '(t12 . e2))
           (if (eqval? 'final 'f fv-list)
               (list not-cross?
                     delete 't12
                     delete 't12
                     delete 't12
                     delete 't12)
               (list not-touch?
                     delete 't12
                     delete 't12
                     delete 'e2
                     delete 't12
                     delete 't12)))
         ((equal? border '(k1 . e2))
          (if (eqval? 'initial 'f fv-list)
              (list not-touch?
                    delete 'k1)
              (list not-touch?
                    delete 'p12)))
         ((and (equal? border '(k2 . e2))
               (eqval? 'particle 'f fv-list))
          (if (eqval? 'final 'f fv-list)
              (list not-cross?
                    delete 't22)
              (list not-cross?
                    delete 't23)))
```

```
        (else (list tautology))))
((variation)
 (if (eqval? 'var1 1 fv-list)
     (set! tc-list
         (move '((t12 . (-1 . 1)))
               tc-list)))
 (if (eqval? 'var2 1 fv-list)
     (set! tc-list
         (move '((p12 . (-1 . 1))
                 (k1  . (-1 . 2)))
               tc-list)))
 (case (val 'var3 fv-list)
   ((1) (set! tc-list
           (move '((t23 . (-1 . 1))
                   (k2  . (-1 . 1)))
                 tc-list)))
   ((2) (set! tc-list
           (move '((t22 . (1 . 1))
                   (t23 . (1 . 1))
                   (k2  . (1 . 1)))
                 tc-list))))
 tc-list)
((modification)
 (if (and (eqval? 'var3 0 fv-list)
          (specified? 'k2 tc-list)
          (rat=? (time-comp (get-tc 'k2 tc-list))
                 '(3 . 1)))
     (if  (or
           (and (eqval? 'final 't fv-list)
                (rat>=? (displacement '(k2 . e2) tc-list)
                        '(2 . 1)))
           (and (eqval? 'final 'f fv-list)
                (rat>=? (displacement '(k2 . e2) tc-list)
                        '(1 . 1))))
```

```
                (set! tc-list
                  (move '((t22 . (1 . 1))
                          (t23 . (1 . 1))
                          (k2  . (1 . 1)))
                        tc-list))))
      (if (and (specified? 't23 tc-list)
               (rat=? (displacement '(t23 . e2) tc-list)
                      '(1 . 2))
               (not (moved-pl? 't23 dialect fv-list)))
          (set! tc-list
            (move '((t23 . (-1 . 2))
                    (k2  . (-1 . 2)))
                  tc-list)))
      tc-list)
    ((surface-rule) tc-list)))
;; 金沢方言
((kanazawa)
 (case item
   ((tc-list-init)
    '((e1 0 (0 . 1) e) (p1 0 (0 . 1) neg) (p2 0 (1 . 1) pos)
      (k  1 (0 . 1) neg)
      (e2 2 (0 . 1) e)))
   ((make-index-list)
    (append
      '(k)
      (if (eqval? 'initial 't fv-list)
          '(p1 p2))))
   ((total-compression) #f)
   ((prohibition) '( ))
   ((contraction1)
    (cond ((equal? border '(p2 . k))
           (list not-touch?
                 delete 'p2))
           (else (list tautology))))
```

```
    ((contraction2)
     (cond ((equal? border '(p2 . e2))
            (list not-touch?
                  delete 'p2))
           ((equal? border '(k . e2))
            (list not-touch?
                  compress 'k))
           (else (list tautology))))
    ((variation)
     (if (eqval? 'tone2 2 fv-list)
         (move '((p2 . (-1 . 1)))
               tc-list)
         tc-list))
    ((modification) tc-list)
    ((surface-rule) tc-list)))
;; 高見島方言
((takamishima)
 (case item
    ((tc-list-init)
     '((e1  0   (0 . 1) e)   (t1  0 (0 . 1) neg)
       (p11 0   (0 . 1) neg) (p12 0 (1 . 1) pos)
       (t2  1 (-1 . 1) pos) (k   1 (0 . 1) neg)
       (p2  2 (-1 . 1) pos) (e2  2 (0 . 1) e)))

    ((make-index-list)
     (append
       (if (eqval? 'tone 2 fv-list)
           '(t1 t2))
       '(k)
       (if (and (eqval? 'tone    1 fv-list)
                (eqval? 'initial  't fv-list))
           '(p11 p12))
```

```
      (if (and (eqval? 'tone  2 fv-list)
               (eqval? 'acc   'f fv-list)
               (eqval? 'final  't fv-list))
          '(p2))))
  ((total-compression) #f)
  ((prohibition) '( ))
  ((contraction1)
   (cond ((equal? border '(p12 . k))
          (list not-touch?
                delete 'p12))
          (else (list tautology))))
  ((contraction2)
   (cond ((equal? border '(p12 . e2))
          (list not-touch?
                delete 'p12))
          ((equal? border '(t1 . p2))
           (list not-touch?
                 compress 'p2))
          ((equal? border '(k . e2))
           (list not-touch?
                 compress 'k))
          (else (list tautology))))
  ((variation)
   (if (eqval? 'tone2 2 fv-list)
       (move '((p12 . (-1 . 1)))
             tc-list)
       tc-list))
  ((modification) tc-list)
  ((surface-rule) tc-list))))))
```

4 前2稿の誤りの訂正

岡田(1990,1991)に存在する誤りを，以下のとおり，訂正する。誤りの中には，今回，計算機で検証することによって，初めて発見できたものがある。計算機によって検証を行う

ことの実務上の利点が，ここに存する。

［岡田(1990)の訂正］

p.164 の(6.1)の「柄」の音調

（誤）柄⌉

（正）⌈柄⌉

［岡田(1991)の訂正］

p.169 の(3.9)の 3 行目の最初の式

（誤）I_p([1 T, −acc]/+I)= $\{p_{11}, p_{12}\}$

（正）I_p([1 T]/+I)= $\{p_{11}, p_{12}\}$

p.177 の(5.8)の 3 行目

（誤）$(p_2, -1, +)$

（正）$(p_2, -1, +')$

p.177 の(5.11)の 4 行目

（誤）$b_2=(p_2, e_2)$

（正）$b_2=(t_{21}, p_2)$

p.180 の(6.11)の 2 行目

（誤）$b_2=(k, e_2)$, [1 T]

（正）$b_2=(k, e_2)$, [1 T]/+I

同じく(6.11)の 2 行目と 3 行目の間に，下記を挿入する。

$b_2=(k, e_2)$, [1 T]/−I のとき，$(d_2>0)$, $\alpha_1[k]$,

p.184 の(7.3)の下から 8 行目

（誤）[2 T, +acc]/−P/−F/(3, 1)V

（正）[2 T, +acc]/−P/−F/(3, 2)V

p.185 の11行目

（誤）[2 T, +acc]/+P/−F/(3, 1)V

（正）[2 T, +acc]/+P/−F/(3, 2)V

p.188 の 3 行目

（誤）and not $moved(t_{23}, R, P_1))$ {

（正）and not $moved(t_{23}, R, P_1(R)))$ {

p.185 の下から10行目が，

[2 T, +acc]/+P/+F/(3, 1)V

となっているが，これは誤りである。この指定に該当するのは，この行に続く 3 行にわたっ

て示した6つの音調パターンのうちの，最初の2つのみである。それ以外の4つは，

[2 T, +acc]/+P/+F/(3, 2)V

である。

p.187の(7.16)に示した修正規則Aは，変異形に適用されてはならない。しかし，このことが，適用条件の中に組み込まれていない。そこで，修正規則Aの1行目を，次のように改める。

if ($val((R, \mathrm{V}), 3)=0$ and $spec(P, k_2)$ and $valt(P, k_2)=3$) {

参考文献

岡田英俊(1990)「日本語諸方言の音調体系の定式化」『東京大学言語学論集'89』137-176.
岡田英俊(1991)「日本語諸方言の音調体系の定式化(2)」『東京大学言語学論集』11.143-202.
湯浅太一(1991)『超高速インタープリタScheme』岩波書店.

[付記]
当研究は，平成3年度科学研究費補助金（研究課題「日本語諸方言の音調体系の定式化」）によるものである。