チューリング機械開発システムの作成 (2)

宮下英明　松沢正規*

目　次

9　ソース／オブジェクト／動作表ファイルの仕様

9.1 ソースファイル

ソースファイルは，拡張子が M のテクストファイル（M ファイル）である。

M プログラムは，プリプロセッサ行を定めるのに空白，タブ，改行コードが意味をもつということを除けば，フリーフォーマットである。即ち，これの読み込みにおいては，空白，タブ，改行コードはスキップされる。

モジュール（機械）名は，32文字まで許される。それより長い名は，32文字でカットする（エラーとはしない）。

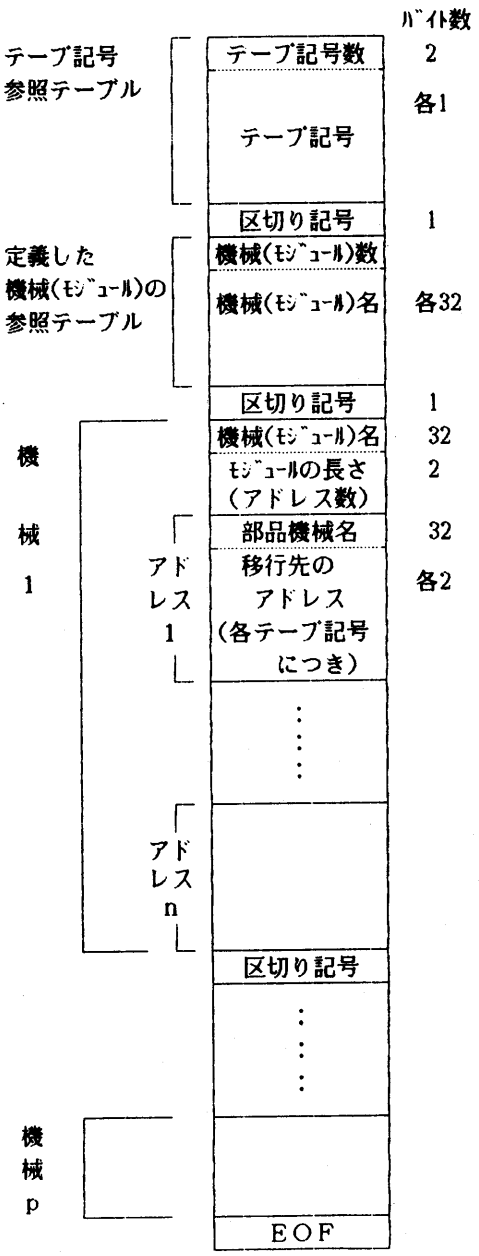
一つの M ソースプログラムには，複数のモジュール（機械）を記述できる。

9.2 オブジェクトファイル

オブジェクトファイルは，拡張子が OBJ のバイナリファイル（OBJ ファイル）である。その

平成 3 年 3 月29日受理
*金沢市立高尾台中学校

仕様を，つぎにように定める：



但し，exitに対応する＃を－2，モジュールの終了に対応する＊を－1，とそれぞれ定める(§7.3)。

9.3 動作表のファイル

9.3.1 TBL と BIN

動作表ファイル（チューリング機械定義ファイル）は，“動作表”のように見える必要はない。動作表ファイルについて肝心なことは，われわれにとってのそれの可読性ではなく，チューリング機械リアライザにとってのそれの可読性である。

われわれにとっての読みやすさと，リアライザにとっての読みやすさとは，同じではない。そこでわれわれは，動作表ファイルを二通りに定めることにする。

一つは，われわれにとっての可読性を優先した仕様であり，DSタイプの動作表をそのまま記述したテクストファイルである。そしてもう一つは，リアライザにとっての可読性（処理の効率性）とファイルのコンパクト性を優先した仕様のバイナリファイルである。この二種類の仕様のファイルは，それぞれ拡張子TBLとBINによって区別する。

チューリング機械リアライザTUは，この二種類のファイルを動作表ファイルとして受容できる。

BINファイルはMファイルのコンパイルによってつくられる。TBLファイルは，以下のいずれかの方法でつくられる：

(1) 直接エディタを用いてつくる；
(2) BINファイルをコンバータにかける；
(3) リンカの－tオプションで，BINファイルと同時に生成する．

TBL形式の動作表を受容できるようにしているのは，任意の動作表がコンパイル&リンクの方法で生成できるわけではなく，また，コンパイル&リンクの方法による動作表の作成が“最適化”の問題を孕んでいるからである。

9.3.2 TBL ファイルの仕様

TBLファイルの仕様を，以下に述べる。なお，“行”は，改行コードで定義される“行”を意味するものとする。

リアライザは，行の先頭に書かれたーを，区切り記号として認識する。かつ，区切り記号のある行——以下，“区切り行”と呼ぶ——は，スキップして読み込まない。

区切り行は，三つ設けられる。したがって，TBL ファイルの内容は，四つの領域でなる。

リアライザは，一番目と四番目の領域をスキップして読み込まない。したがってこの領域を，表題／注釈の記述に使用できる。またこの二つの領域は，空でもよい。

第二および第三の領域の各行においては，スペースあるいはタブでなる列が，データの区切り記号になる。——即ち，リアライザは，スペース／タブ列をスキップして読み込まず，また，スペース／タブと異なる文字に出会うと，つぎのスペース／タブに出会うまでの記号列——以下これを単に“記号列”と呼ぶ——をデータとして読み込む。

第二領域は一行でなり，リアライザはこれを，〈テープ記号〉の列挙と認識する。テープ記号は１バイトの文字でなければならず，スペース／タブ列で区切って書く。

第三領域の各行は，〈内部状態〉，〈動作〉，そしてテープ記号に対応する〈状態推移〉の記述である。

最終行の〈内部状態〉を，〈終了（停止）状態〉として固定する。これに対する〈動作〉，〈状態推移〉は，すべて＊とする。

例：

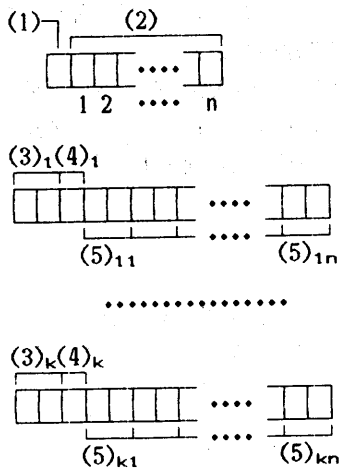
lb.tbl

空白（“無記号”）に出会うまで左移動

```
--------------------------------
                 _    1
--------------------------------
   q0    l      q1   q0
   q1    *      *    *
--------------------------------
```

9.3.3 BIN ファイルの仕様

BIN ファイルの仕様をつぎのように定める（１マスが１バイト）：



(1)　テープ記号の数：
　　１バイトの整数

(2)　テープ記号：
　　テープ記号（キャラクタコード）
　　（但し，ｒ，ｌ，ｅ以外）

$(3)_i$　ｉ番目の内部状態（アドレスに表現）：
　　２バイトの整数

$(4)_i$　ｉ番目の内部状態に対する動作：
　　右移動：文字ｒ（キャラクタコード）
　　左移動：文字ｌ（キャラクタコード）
　　消　去：文字ｅ（キャラクタコード）
　　書　込：テープ記号（キャラクタコード）

$(5)_{ij}$　ｉ番目の内部状態に対応する動作の後，ｊ番目のテープ記号に対応して推移する先の内部状態（アドレス）：
　　２バイトの整数

9.3.4 TB0 ファイル

SD タイプの動作表のファイル形式を，以下のように定める——ファイルの拡張子は，TB0 とする。

ファイル形式は，TBL ファイルの場合に準じ，第三領域の書き方のみ異なる。

即ち，第三領域の各行において，最初の記号列は〈内部状態〉であり，そしてこれの後に〈動作ー状態推移〉の列挙が続く。

最終行を除いては，各〈動作－状態推移〉は〈動作〉の1バイト文字と推移先の〈内部状態〉の記号列の連結である。

最終行の〈内部状態〉を，〈終了（停止）状態〉として固定する。これに対する〈動作－状態推移〉は，すべて記号列＊＊とする。

例：

lb.tb0

空白（“無記号”）に出会うまで左移動

	_	l
q0	lql	lql
q1	_q2	lql
q2	**	**

10　コンパイラ TC

われわれは，コンパイラとして

TC.EXE

を用意する。TCは，拡張子Mのソースファイル（Mファイル）からそれと同名で拡張子OBJのオブジェクトファイル（OBJファイル）を生成する。

10.1 TCの作業内容

TCは，つぎのことをする：

(1) Mファイルを読み込みモードで開き，つぎに同名のOBJファイルを書き込みモードで開く。

(2) Mプログラムの#symbol行を読み，〈記号参照テーブル〉の形でOBJファイルに書き込む。

(3) Mファイルに記述されている各モジュールfに対し，以下のことを繰り返す：

(3.1) fを読み込む。

(3.2) fのパース・トリーtを作成する。

(3.3) tをもとにfのリロケータブル・オブジェクトモジュールを作成し，OBJファイルに書き込む。

(3.4) fに対するこの作業において新たに確保したメモリを解放する。

(4) MファイルとOBJファイルを閉じる。

以下では，Mソースファイルに記述されたモジュールからリロケータブル・オブジェクトモジュールを生成する方法を，つぎのMプログラムを例にして，特に述べる：

```
#symbol 1  /*記号は_と1のみ*/
a
{
  if (_)
    b ;
  if(1) {
    c ;
    while {
      if (_)
        break ;
      while {
        if(1)
          exit ;
        d ;
        if(1)
          break ;
      }
      e ;
    }
  }
  else
    f ;
  g ;
}
```

10.2 整　形

コンパイラは，先ず，ソースプログラムに対して整形の処理をする。即ち，“何もしない”チューリング機械nullを，つぎの規則に従って挿入する：

(1) if, while, break, return, exitの前に挿入する；

(2) 二つの連続した｝の間に挿入する；
(3) モジュールを閉じる｝の前に挿入する；
(4) 略記号；(これは null; の略記号と見なされる) を null; に換える。

null は，コンパイル・リンクの処理ルーチンを簡単にするために導入されるダミーのチューリング機械である。

例の M プログラムは，つぎのようになる：

```
a
｛
    null ;                          0
    if (_) ｛
        b ;                         1
    ｝
    null ;                          2
    if(1) ｛
        c ;                         3
        while ｛
            null ;                  4
            if (_) ｛
                null ;              5
                break ;
            ｝
            null ;                  6
            while ｛
                null ;              7
                if(1) ｛
                    null ;          8
                    exit ;
                ｝
                d ;                 9
                null ;              10
                if(1) ｛
                    null ;          11
                    break ;
                ｝
                null ;              12
            ｝
            e ;                     13
        ｝
        null ;                      14
    ｝
    else ｛
        f ;                         15
    ｝
    g ;                             16
    null ;                          17
｝
```

なお，コンパイラは整形と併行して，
(1) 括弧の対応，
(2) 位置的に予約語であるべき語の綴り，
(3) elseif, else がこれに対応する if をもっているかどうか

のチェックも行なう。

10.3 パース

ソースプログラムを整形すると，コンパイラはつぎにソースプログラムの文法解析(パース)に入る。これは，ソースプログラムの構造木の作成で終わる。

パースは，つぎの二つの作業を併行して行なう形で進められる。

トークンへの分解
構造木の作成

この作業は，ソースプログラムの頭から順になされる。

10.3.1 トークンへの分解

トークンの画定は，つぎの規則による：

《｛,｝, ；のうしろで区切る》

例では，トークンへの分解はつぎのようになる：

a ｛	1	｝	21
null ;	2	d ;	22
if (_) ｛	3	null ;	23
b ;	4	if(1) ｛	24
｝	5	null ;	25
null ;	6	break ;	26
if(1) ｛	7	｝	27
c ;	8	null ;	28

```
while {          9  |  }              29
null ;           10 |  e ;            30
if (_) {         11 |  }              31
null ;           12 |  null ;         32
break ;          13 |  }              33
}                14 |  else {       · 34
null ;           15 |  f ;            35
while {          16 |  }              36
null ;           17 |  g ;            37
if(1) {          18 |  null ;         38
null ;           19 |  }              39
exit ;           20 |
```

10.3.2 構造木の作成

構造木は，トークンを一つ画定する毎につぎの処理をしていくことで，最終的に得られる：

(1) トークン構造体をメモリに一つ確保する
(2) この構造体に，トークンの情報を書き込む
(3) ここまでに作成されている構造木に，この構造体を規則にしたがってリンクする。

10.3.3 トークン構造体への書き込み

トークン構造体は，つぎのように定義される：

```
id ( char )
name    ⟶ char 型配列
left    ⟶ トークン構造体
right   ⟶ トークン構造体
```

id には，トークンのアイデンティティ：
if，elseif，else，while，break，exit，return，MAIN，MACHINE，NIL
を書き込む。ここで，

(1) MAIN は，本モジュール（以下，“メインモジュール”と呼ぶ）のアイデンティティ
(2) MACHINE は基底動作および本モジュールが呼び出しているモジュール（以下，単に“モジュール”と呼ぶ）のアイデンティティ，
(3) NIL は } のアイデンティティ。

name には，

(1) トークンが基底動作あるいはモジュールのときは，それの名
(2) if，elseif ならば，その引き数

を書き込む。

10.3.4 トークン構造体のリンク

トークン構造体のリンク処理は，再帰的になされる。この処理では，新しいトークン構造体 S に対し，S をリンクするターゲットのトークン構造体 T が，その都度指定されている。

リンクの規則は，つぎのようになる：

(1) T のトークンが if，elseif，else，while で，S のトークンがこの文の中にあるときは，T の left に S をつなげる。
(2) (1)以外の場合は，T の right に S をつなげる。

ターゲット更新の規則は，つぎのようになる：

(1) S のトークンが if，elseif，else，while のときは，いまのネストでターゲットを S に更新するとともに，再帰のネストを一つ進めた処理を初期ターゲット S で始める。
(2) S のトークンが NIL のときは，再帰のネストを一つ戻す。
(3) (1)，(2)以外の場合は，ターゲットを S に更新する。

例では，図1に示した構造木が生成される。

10.4 オブジェクトモジュールの作成

ソースプログラムをパースしてこれの構造木を作成すると，つぎはオブジェクトモジュール作成の段階になる。作業は，

(1) 構造木の先頭のトークンをターゲットとすることから出発し，
(2) ターゲットのトークン構造体を参照して，規則の定める処理を行なう

という形で進められる。結果として，所期のオブジェクトモジュールが得られる。

10.4.1 MT 構造体

オブジェクトモジュールの作成にあたり，先ず，動作表の“一行”（モジュールの“一項”）：

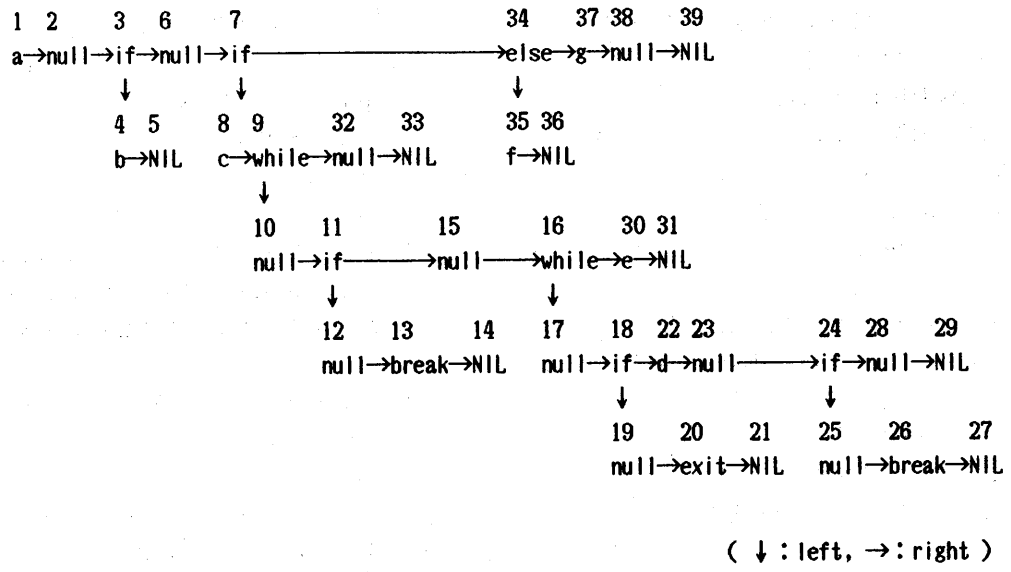
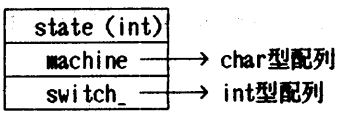


図　1

〈内部状態〉
〈機　械〉
〈テープ記号 s_1に対し推移する内部状態〉
……
〈テープ記号 s_kに対し推移する内部状態〉
（s_1, ……, s_kは，テープ記号の全体）
を書き込む MT (module-term)構造体を定義する：



即ち，
(1) state には，〈内部状態〉の記号（アドレス）としての整数が書かれる。
(2) machine は，
〈state に書かれている内部状態に対応する機械名〉
を書いた配列をポイントする。
(3) switch_は，
〈state に書かれている内部状態において machine に書かれている動作をした後，各テープ記号に対して推移する内部状態（整数値）〉
を書いた配列をポイントする。

10.4.2 道　具
この作業のために，スタック：
(1) ネスト情報保存スタック
(2) ペンディングアドレス保存スタック
(3) ループ開始アドレス保存スタック
(4) ループブレークアドレス保存スタック
(5) IF 引数スタック
およびフラグ：
(1) ループブレークフラグ
(2) IF ブレークフラグ
そして，新しく確保した MT 構造体の state に書き込むべき整数を保存する変数：
new_state（初期値 0 ）
を，道具として用意する。

10.4.3 処理規則
処理規則はつぎのようになる。即ち，ターゲットのトークンの id が
(1) MAIN ならば，
(1.1)ネスト情報保存スタックに MAIN をプッシュする。

(1.2) rightがポイントするトークンをターゲットトークンとする。
(2) MACHINEならば,
(2.1) MT構造体をメモリに1つ確保する。以下, new_state = nであるとして,
(2.2) stateに, nを書く。
(2.3) nameに, ターゲットトークンのnameをコピーする。
(2.4) switch_配列を, RETURNを表わす値(ここでは−1)で埋める(初期化)。(以下, この値が入っている箇所を“未記入”と言い表わす)。
(2.5)ループブレイクフラグまたはIFブレイクフラグが立っているとき,
(2.5.1) ペンディングアドレス保存スタックにプッシュされているすべてのアドレスをポップする;
(2.5.2) ポップされた各アドレスmに対し, mをstate値とするMT構造体のswitch_配列の未記入箇所すべてに, nを書込む。
(2.5)そうでないとき(これは, 直前のトークンがMACHINEかifの場合である)は, ペンディングアドレス保存スタックのラストインをポップし, このアドレスをstateとするMT構造体に対して, 以下の処置をする:
(2.5.1) IF引数スタックにテープ記号(番号)が入っているときは, これらをすべてポップし, 各々のテープ記号について, これと対応するswitch_配列の箇所にnを書き込む。
(2.5.2) IF引数スタックが空のときは, switch_配列をnで埋める。
(2.6) ペンディングアドレス保存スタックにnをプッシュする。
(2.7) new_stateをn + 1にする。
(2.8) rightがポイントするトークンをターゲットトークンとする。
(3) breakならば,
(3.1) ペンディングアドレス保存スタックからアドレスを一つポップし, ループブレークアドレス保存スタックにこれをプッシュする。
(3.2) rightがポイントするトークンをターゲットトークンとする。
(4) exitならば,
(4.1) ペンディングアドレス保存スタックからアドレスを一つポップし, これがstate値になっているMT構造体に対し, switch_配列の未記入箇所すべてにEXITを表わす値(ここでは−2)を書き込む。
(4.2) rightがポイントするトークンをターゲットトークンとする。
(5) returnならば,
(5.1) ペンディングアドレス保存スタックからアドレスを一つポップする。
(switch_配列は, 初期化においてRETURNを表わす値で埋めたので, いまポップしたアドレスをstate値とするMT構造体に対する処理は既に済んでいることになる。)
(5.2) rightがポイントするトークンをターゲットトークンとする。
(6) ifならば,
(6.1)ネスト情報保存スタックにIFをプッシュする。
(6.2) ターゲットトークンのnameに書き込まれているIF引数を, 全てIF引数スタックにプッシュする。
(6.3) ペンディングアドレス保存スタックのラストインを一つの自動変数if_start_addressに保存し, ネストを一つ進めて, leftがポイントするトークンをターゲットトークンとする再帰処理に入る。
(6.4) 再帰処理から戻ったら,
(6.4.1) IFブレイクフラグを立てる。
(6.4.2) if_start_addressをペンディングアドレス保存スタックにプッシュする。
(6.4.3)ターゲットトークンを, rightがポイントするトークンにする。
(7) elseifならば,
(7.1)ネスト情報保存スタックにIFをプッシュ

する。

(7.2) ペンディングアドレス保存スタックのラストインを参照し，if_start_address に保存する。(この elseif に対応する if を right でポイントするトークンを x とするとき，x に対して確保された MT 構造体の state 値が，いまのアドレスに当たる。)

(7.3)このとき IF ブレイクフラグが立っていることになるが ((6.4.1) 参照)，これを下ろす。

(7.4) ターゲットトークンの name に書き込まれている ELSEIF の引数を，全て IF 引数スタックにプッシュする。

(7.5) ペンディングアドレス保存スタックのラストインを自動変数 if_start_address に保存し，ネストを一つ進めて，left がポイントするトークンをターゲットトークンとする再帰処理に入る。

(7.6) 再帰処理から戻ったら，

(7.6.1) IF ブレイクフラグを立てる。

(7.6.2)) if_start_address をペンディングアドレス保存スタックにプッシュする。

(7.6.3)ターゲットトークンを，right がポイントするトークンにする。

(8) else ならば，

(8.1)ネスト情報保存スタックに IF をプッシュする。

(8.2)このとき IF ブレイクフラグが立っていることになるが，これを下ろす。

(8.3) ネストを一つ進めて，left がポイントするトークンをターゲットトークンとする再帰処理に入る。

(8.4) 再帰処理から戻ったら，IF ブレイクフラグを立てる。

(8.5)ターゲットトークンを，right がポイントするトークンにする。

(9) while ならば，

(9.1)ネスト情報保存スタックに while をプッシュする。

(9.2) ループ開始アドレス保存スタックに，ペンディングアドレス保存スタックのラストインをコピーする。

(9.3) ネストを一つ進めて，left がポイントするトークンをターゲットトークンとする再帰処理に入る。

(9.4) 再帰処理から戻ったら，ループブレイクフラグを立てる。

(9.5)ターゲットトークンを，right がポイントするトークンにする。

(10) NIL ならば，

ネスト情報保存スタックから内容をひとつポップする。それが

(10.1) MAIN なら，

作業終了。

(10.2) if ならば，

一つ前のネストに戻る。

(10.3) while ならば，

(10.3.1)ループ開始アドレス保存スタックからアドレスを一つポップする。これを q とする。

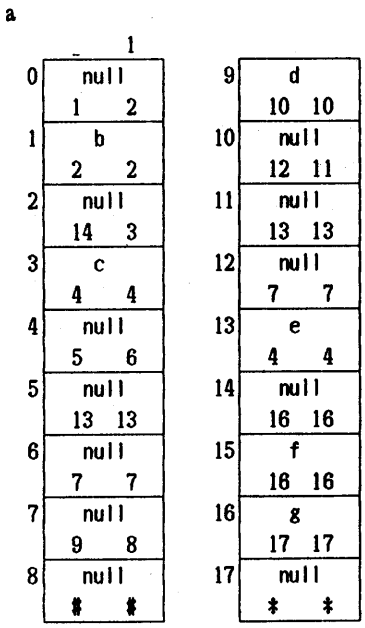
(10.3.2)ペンディングアドレス保存スタックに入っている全てのアドレスをポップし，各アドレスについて，これを state 値とする MT 構造体の switch_配列の未記入部分全てに q を書き込む。

(10.3.3)ループブレイク保存スタックに入っているアドレスを，＞q である限りポップし，全てペンディングアドレス保存スタックにプッシュする。

(10.3.4) 1 つ前のネストに戻る。

10.4.4 結　果

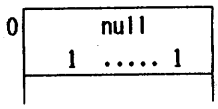
例の構造木からは，つぎのリロケータブル・オブジェクトモジュールが生成される—ここで＃は，exit に対応して移行先アドレスが無い状態であり，＊はこのモジュールの終了ということで，移行先アドレスが無い状態である：



10.5 圧　縮

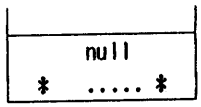
オブジェクトモジュールが得られると，つぎにこれの圧縮の処理に入る。即ち，これからnullのセルをつぎの規則によって削除する：

(1) アドレス0が

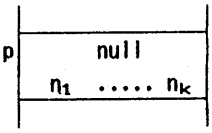


であるときは，これを消去する。

(2) 末尾の

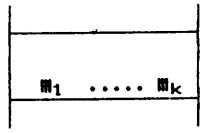


を除く全てのnullに対し，つぎの処理を行なう。即ち，



に対し，

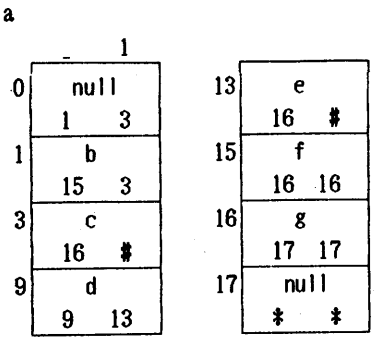
(2-1) アドレスpを含む全ての枠



の $m_i = p$ を n_i に書き換える。

(2-2) このnullの枠を消去する。

この処理の結果，先のリロケータブル・オブジェクトモジュールは，つぎのように圧縮される：



なおこの例では，d，eが呼び出されることはない。これらの圧縮は，“最適化”の主題になる。

10.6 オブジェクトファイルの出力

オブジェクトモジュールを，オブジェクトファイルに出力する。このとき，オブジェクトファイルの仕様を満たすよう，適当に情報を付加する。

11　リンカ TL

リンカとして

TL.EXE

を用意する。

TLは，OBJファイルから拡張子BINの動作表ファイル（BINファイル）を生成する。

11.1 TL起動の書式

TL起動の書式は，

TL　[－t]　〈ファイル名〉＋…
　　＋〈ファイル名〉,〈動作表ファイル名〉

である。ここで，

(1) －tは，TBLファイルを同時に生成することを指定するオプション。

(2) 〈ファイル名〉は，リンクするOBJファイルの名で，拡張子を省略したもの。
(3) 〈動作表ファイル名〉は，生成するBINファイルの名で，拡張子を省略したもの。

11.2 TLの作業内容

11.2.1 初期作業

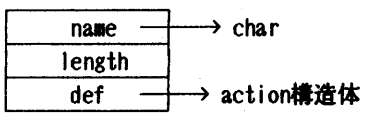
(1) コマンドラインで指定されたすべてのOBJファイルを読み込みモードで開き，また，同じくコマンドラインで指定された動作表ファイルを，書き込みモードで開く。
(2) 先頭に指定されたOBJファイルからテープ記号参照テーブルを読み込み，残りのOBJファイルのテープ記号参照テーブルを，これと照合する。違いがあれば，エラーとし，開いていたファイルを閉じて作業を終了する。
以下，テープ記号がつぎのk個であるとする：

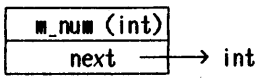
$$S_0 = \_,\ S_1,\ \cdots\cdots,\ S_{k-1}$$

11.2.2 〈作業テーブル〉の作成——ステップ1

各OBJファイルにある〈定義機械参照テーブル〉をもとに，第1階〈作業テーブル〉を以下のように作成する：

(1) machine構造体とaction構造体を





のように定義する。
(2) 基底機械の数と，null登録のための1，そして各OBJファイルで定義されている機械(モジュール)の数の合計sを求め，machine構造体のポインタs個分のメモリを，配列の形で確保する。
(3) 機械の登録として行なうことは，
(3.1) 各機械に対して，その都度machine構造体のメモリを確保する；
(3.2) nameにファイル名をコピーする；
(3.3) lengthに機械の(モジュールとしての)長さを書き込む。
このとき，メモリ確保に先立って，既に同じ機械名が読み込まれていないかを，ここまでにつくられている〈作業テーブル〉に対してチェックする。そして同じ機械名があれば，エラーとし，開いているファイルを閉じて，作業を終了する。
(4) 機械の登録は，基底機械とnullの登録から始める。つぎに，各OBJファイルに対し，〈定義機械参照テーブル〉から機械名を一つずつ読み込み登録する。

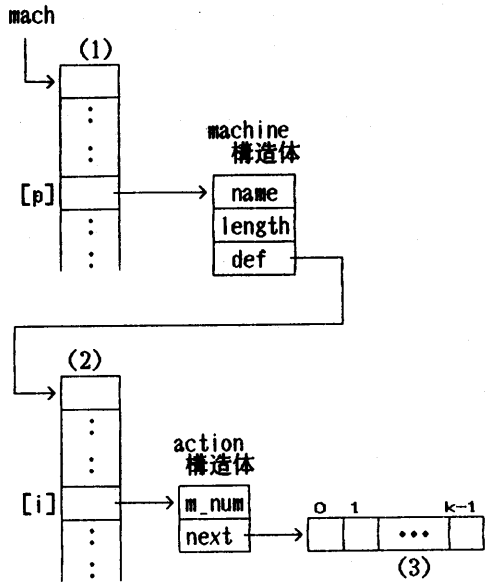
11.2.3 〈作業テーブル〉の作成——ステップ2

引き続き，各OBJファイルから機械(モジュール)の内容を読み込みながら，〈作業用テーブル〉を以下のように作成する：

(1) 読み込んだ〈機械(モジュール)名〉とmach[p]−>nameがポイントする文字列が一致するようなpを特定する。
(2) 〈モジュールの長さ〉nを読み込み，action構造体のポインタn個分のメモリを，配列の形で確保する。そしてmach[p]−>defがこれの先頭のアドレスを指すようにする。
(3) (mach[p]−>def)[i−1]までの処理が終わっていて，〈部品機械名〉がさらに読み込めるとき，action構造体一つ分のメモリを確保して，(mach[p]−>def)[i]がこれの先頭のアドレスを指すようにする。さらに，整数型の配列をテープ記号数の長さだけ確保し，((mach[p]−>def)[i])−>nextがこれの先頭のアドレスを指すようにする。
(4) 読み込んだ〈部品機械名〉とmach[q]−>nameがポイントする文字列が一致するようなqを特定する。このqを((mach[p]−>def)[i])−>m_numに書き込む。
(5) 引き続き〈移行先アドレス〉を読み込み，(_を含めた) k個のテープ記号に対応する移行

先アドレスを,((mach [p]−> def)[i])−> next が指す配列に書き込む。

以上の処理によって,〈作業用テーブル〉が以下の形で実現される：



ここで,

(1)の配列の番号 p は，mach [p]−> name に保存されている機械（モジュール）名——Xとする——の登録番号と解釈される。

(2)の配列の番号 i は，(mach [p]−> def)[i] がポイントする動作の，モジュールXにおけるアドレスと一致する。

(3)のマスに入っている数は，モジュールXのアドレス i に示されている〈移行先アドレス〉。

last = mach [p]−> length − 1 とするとき，

(((mach[p]−>def)[last])−>next)[j]=−1

(j = 0，……，k − 1)

11.2.4 リンク＝動作表作成

リンク＝動作表作成は,つぎのように行なう。即ち，mach [m]−> name が“ main ”を指すとき，

entry = m ,

adr_off = 0 （アドレス・オフセット），

return_adr = return

として，（再帰的）処理

write_tbl (entry , adr_off , return_adr) に入る。ここで，write_tbl (entry , adr_off , return_adr) は， i が 0 から last = mach [entry]−> length − 1 まで動くところの,つぎの処理：

(1) entry1 = ((mach [entry]−> def) [i])−> m_num が基底機械あるいは null の登録番号であるとき，BIN ファイルに

i + adr_off ,
machine [0],
next [0]
.....
machine [k − 1],
next [k − 1]

を順に書き込む——ここで，

(1.1) machine[j] (j= 0 , … , k − 1) は，

(1.1.1) entry1 が基底機械の登録番号のときは mach [entry 1]−> name で，

(1.1.2) null の登録番号のときは,記号 sj を書き込む基底機械の名(j = 0 のときは e)

(1.2) next [j] (j= 0 , ……, k − 1) は，

(1.2.1) i ≠ last のとき，

next [j] =
(((mach [entry]−> def)[i])−>
next)[j]+ adr_off

(1.2.2) i = last のとき，

next [j] = return_adr [j]

である。

(2) entry1 が基底機械および null でないとき，処理：

write_tbl (entry 1, adr_off + last + 1 ,
((mach [entry]−> def)[i])−> next)

に入る。

TBL ファイルの作成では,アドレスを ASCII 文字列で書き込むことになるが,このときには，null と return の対に対応するところの最後の書

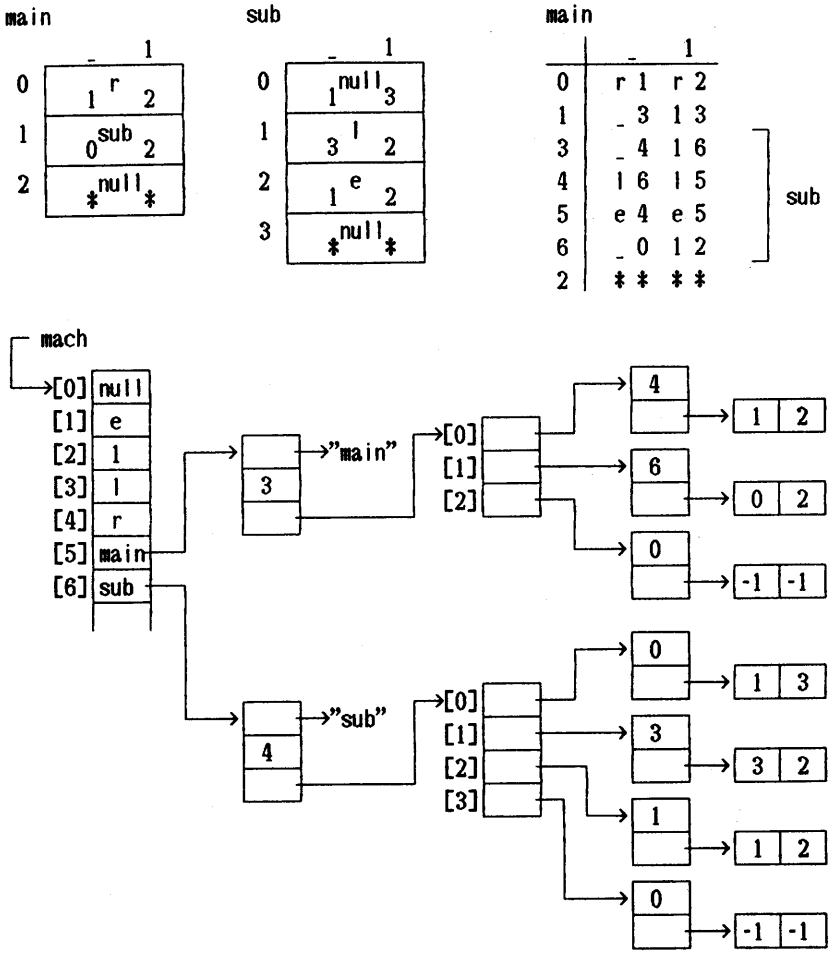


図　2

き込みを，k個の＊＊の書き込みにする。

なお，リンクの図式を図2に示す。

11.3 MAKE ファイルの作成

TC，TLの使用では，MAKEユーティリティを利用できる。このときのMAKEファイルは，つぎのような具合になる—但し，a，x，yのどれかにmainモジュールがただ一つ存在し，a，x，yから呼び出されるモジュールは全てa，x，yのどれかに含まれるものとする：

```
#-------------------------
#   a.mk （ makefile ）
#-------------------------

.m.obj:
    tc $*.m

a.obj:  a.m

x.obj:  x.m

y.obj:  y.m

a.bin:  a.obj x.obj y.obj
    tl -t a + x + y, a
```

12　チューリング機械リアライザ TU

われわれは，チューリング機械リアライザ——兼デバッガ（チューリング機械定義プログラムの）——として，実行プログラム

TU.EXE

を用意する。

デバッガとしての用途のために，トレース機能も付加されている。

13　ファイルコンバータ

われわれはユーティリティとして，BIN, TBL, TB0 三種類のファイルの間のコンバータ

CONV.EXE

を用意する。

コンバートの方向は，つぎのオプションによって指定する：

－ xy

ここで，x，yは，

BIN を表わす b
TBL を表わす t
TB0 を表わす 0

のいずれかで，オプションの意味は“x から y へのコンバート”である。

BIN と TBL の間のコンバートは，データ型の変換に過ぎない。

TBL から TB0 へのコンバートは，つぎのようにする。即ち，

		s_1 .... s_k
⋮	⋮	••••
q	a	q_1 .... q_k
⋮	⋮	••••

の各行に対し

		s_1 .... s_k
⋮	⋮	••••
q	a	r .... r
⋮	⋮	••••
r	null	q_1 .... q_k

の処置をする。そして

q	a	r .... r

の形の行を

q	ar .... ar

に書き換え，

q	null	q_1 .... q_k

の形の行を

q	s_1q_1 .... s_kq_k

に書き換える。

逆に，TB0 から TBL へのコンバートは，つぎのようにする。即ち，

	s_1 .... s_k
⋮	••••
q	a_1q_1 .... a_kq_k
⋮	••••

の各行に対し，

	s_1 .... s_k
⋮	••••
q	s_1r_1 .... s_kr_k
⋮	••••
r_1	a_1q_1 .... a_1q_1
⋮	••••
r_k	a_kq_k .... a_kq_k

の処置をする。そして

q	s_1q_1 .... s_kq_k

の形の行を

q	null	q_1 .... q_k

に書き換え，

q	ar .... ar

の形の行を

r	a	q .... q

に書き換える。