ICOT Technical Report: TR-079

TR-079

逐次型推論マシン ϕの マイクロインタプリタ

山本　明，横田　実，西川　宏，
瀧　和男，内田俊一

September, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

１．はじめに

知識表現及びそれを効率良く利用するための研究が盛んになると共に、知識を基本的な推論機構でうまく表現できるプログラミング言語として論理型言語を用いる動きが活発になってきている。ＩＣＯＴにおいても第５世代コンピュータシステム開発の一環として論理型プログラミング言語であるPrologを用いて多くの研究が行なわれている。しかし、既存の計算機システムでは処理能力、メモリ空間が十分ではなく実用的なプログラムを記述、実行させることが困難であった。ＩＣＯＴではこのような状況を解決するためのソフトウェア開発用ツールとしてパーソナル逐次型推論マシン（φと呼ぶ）の開発を行なっている。φはソフトウェア開発用ツールとしてユーザに十分効率の良い実行環境を提供するため以下の特徴を備えている。

○論理型言語の効率の良い実行

論理型プログラミング言語としてPrologが良く知られている。φはProlog風のKL0 (kernel language version 0) と呼ばれる論理型言語を機械語とし、これを専用のハードウェアとマイクロプログラムで記述されたインタプリタ(マイクロインタプリタ)により直接効率良く解釈実行する。

○実用的な記憶空間と実行速度

実用的なプログラムを開発するには充分な記憶空間と実行速度が必要であるため、φは16M ワードの大きな記憶空間と、約30KLIPS(Logical Inference Per Second) の処理能力を備えている。

○パーソナルユース

充分な処理能力を得、かつ入出力装置も独占して使用することができるように、個人でマシンを専有して使用するパーソナルマシンである。

○豊富な入出力装置

ディスク、フロッピーディスク、高解像度のビットマップディスプレイ、ポインティングデバイス等の豊富な入出力装置を備え、かつ使い易いマンマシンインタフェースを提供している。

○ローカルネットワークのサポート

φはパーソナルマシンであるが、各φはローカルネットワーク（LAN)により接続されφ間の通信及び資源の共通利用を行なうことができる。

○マルチプロセシング機能のサポート

最大63個までのプロセスを同時にかつ独立に実行することの出来る実行環境をサポートしている。

○オペレーティング・システムのサポート

プロセススイッチ、割込み処理、メモリ管理等をハードウェア／ファームウェアでサポートすることにより処理速度の向上とOSプログラムの記述を容易にしている。

○評価データの収集、計測機能

ψは多くの実験的試みも取り入れたマシンであるため、個々の性能及びシステム全体の性能評価のためのデータ収集が出来るように設計されている。

本稿ではψの機械語である論理型言語KL0 及びオペレーティング・システムを効率良く実行するためのマイクロインタプリタの概略について報告する。

2. KL0 (Kernel Language Version 0)

KL0 はPrologに非常に良く似たプログラミング言語であるが、OSプログラムの記述易さのため、Prologにない入出力操作等のハードウェア操作機能及び柔軟な制御のための拡張された実行順序制御機能を備えている。

2.1 データタイプ

KL0 で定義されているデータのタイプには、基本データタイプとしてアトム、整数、実数、変数があり、構造体データタイプとしてベクタ、ストリングがある。

アトムはシンボリックな定数を表わすためのものであり32ビットで表現される。又、整数は32ビットで表現でされ、実数は指数部 8ビット仮数部24ビットで表現される。

ストリングはキャラクタ及びビット表現のために使用され、全ての構造体データは１次元配列のベクタの形式で表現される。その他にもインビジブルポインタ等のマイクロインタプリタのみが使用するいくつかのデータタイプがある。

○ 基本データタイプ
- ・アトム
- ・整数
- ・実数
- ・変数

○ 構造体データタイプ
- ・ベクタ
- ・ストリング

2.2 内部形式

KL0 で記述されたプログラムはコンパイラにより、ワード列からなるテーブル（内部形式）群に変換される。内部形式はヘッドに同一述語名と同数の引数をもつクローズの集合（プロシジャ）単位に生成され、そのテーブルはプロシジャヘッド部、クローズインデクシングテーブル部及びクローズの内部表現部の集合で構成される。（図－１）

なお、テーブル中の各ワードにはマイクロインタプリタがクローズ処理を効率よく実行できるようにデータタイプを表わすタグが付加されている。

○プロシジャヘッド部

内部形式のテーブルサイズやクローズヘッド中に含まれる引数の数等のそのプロシジャ全体に関する情報で構成される。

oクローズインデクシングテーブル部

ハッシュ対象となる引数の位置等インデクシングのための情報及びハッシュした結果により実行すべきクローズの置かれた場所を示すテーブルからなる。インデクシングを行わない場合にはこのテーブル部は生成されない。

oクローズ部

KLO で記述されたクローズに一対一に対応して生成され、クローズヘッド部、ヘッド引数部及びボディゴール部の集合から構成される。クローズヘッド部はクローズのタイプ、引数のタイプ、ユニフィケーションに失敗した際に次に実行すべきクローズへのポインタ等マイクロインタプリタが実行制御のために使用する情報で、ヘッド引数部はヘッドに出現する引数で構成されている。又、ボディ部の内部形式にはさらにユーザ定義述語呼び出し、組込述語呼び出し、間接呼び出しがある。

［ユーザ定義述語呼び出し形式］

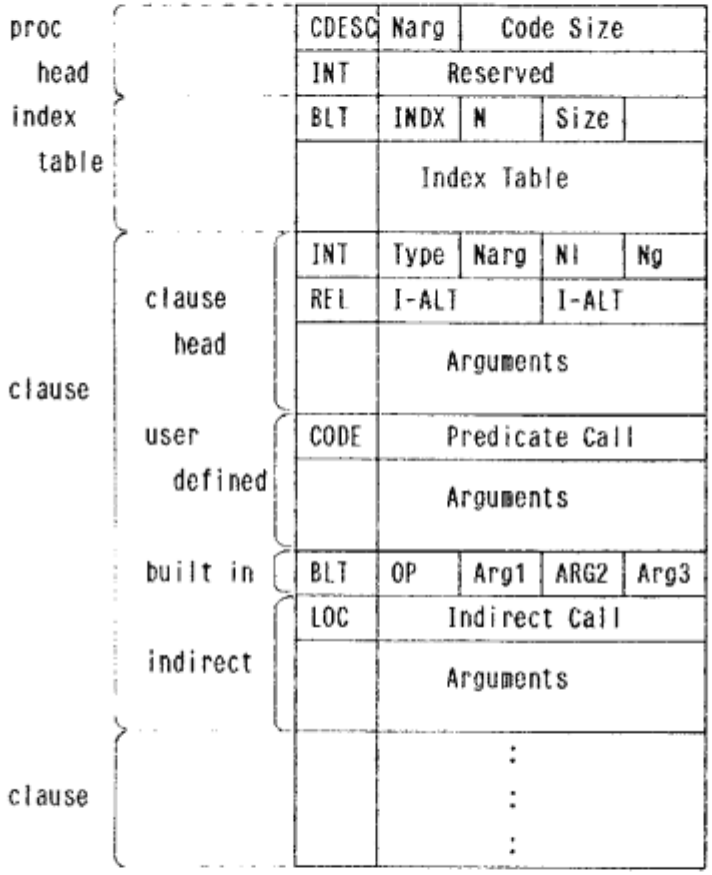
ユーザが定義した述語の呼び出しに対応して生成され、呼び出すべき述語へのポインタとそのゴールの引数列で構成される。

［組込述語呼び出し形式］

システム側で用意した述語（組込述語）に対応して生成され、オペレーションコード部とオペランド部から構成される。効率の良い実行を行なうため、ほとんどの組込み述語のオペレションコードとオペランドは１ワードにパックされた形の内部形式に変換される。

［間接呼び出し形式］

プログラム実行中にダイナミックにユーザ定義述語を呼び出すための間接呼び出しに対応して生成され、ポインタ部と引数列で構成される。ポインタ部の値はオペレーティングシステムによって実行時にセットされる。


proc
head
index
table
clause
clause
head
user
defined
built in
indirect
clause
CDESC
Narg
Code Size
INT
Reserved
BLT
INDX
N
Size
Index Table
INT
Type
Narg
Nl
Ng
REL
I-ALT
I-ALT
Arguments
CODE
Predicate Call
Arguments
BLT
OP
Arg1
ARG2
Arg3
LOC
Indirect Call
Arguments

図－1　KL0 内部形式

3．マイクロインタプリタ

ψは機械語である論理型言語KL0 をマイクロインタプリタで効率良く実行するための各種の方式を採用していると共に、オペレーティングシステムのハードウェアに依存したいくつかの機能をファームウェアでサポートすることによりシステム全体の処理能力の向上を図っている。

マイクロインタプリタは次の機能を持つ。

実行順序制御処理
組込述語処理
OSサポート処理

3.1 実行順序制御処理

実行制御処理は実行制御とユニフィケーションとから成り、これらはいくつかの方式及び専用のハードウェアにより高速かつ効率良く実行される。

3.1.1 実行制御

実行制御はPrologと同様に呼び出し元ゴールによる処理対象クローズの呼び出し、クローズについての全ての処理が完了した時の呼び出し元クローズへのリターン、処理に失敗した際の別の解を求めるためのバックトラックからなる。

a)スタック

ψでは実行順序及び実行環境を管理するための方式としてスタックを使用している。スタックは４本あり、それぞれグローバルスタック、ローカルスタック、コントロールスタック及びトレイルスタックと呼ばれる。

［グローバルスタック］

構造体中に現われる変数に対応するセル、構造体中には現われないがローカルスタックに格納された場合処理の途中にそのセルが消えてしまう可能性のある変数（アンセイフ変数）及び特殊な制御用の情報が格納される。又、構造体のデータ表現の形式としてストラクチャシェアリング方式を採用しているため、構造体データは構造体へのポインタであるスケルトンアドレスのセルと、構造体中の変数の値を規定するためのフレームアドレスのセルで表現される。これらもグローバルスタック中に格納され、変数セルからポインタによって指される。グローバルスタック中のセル及び制御情報はユニフィケーション時に生成され、ユニフィケーション失敗によるバックトラック又はガベージコレクタにより解放される以外には解放されることがない。

［ローカルスタック］

クローズヘッドの引数及びボディゴールに出現し、構造体中の変数には現れずかつアンセーフ変数ではない変数に対応したセルの格納領域である。ローカルスタック中のセルは

ユニフィケーション時に生成され、対応するクローズに他の選択肢がなく処理が決定的に終了した場合に解放される。

［トレイルスタック］

ユニフィケーションに失敗した場合、他の選択肢の残っている最近の述語までバックトラックして処理をやり直す必要があり、バックトラックする選択肢以降バックトラック時点までに実行されたユニフィケーションの結果はすべて無効となる。したがってこの間にユニフィケーションによって変数にバインドされた値をリセット（アンドゥ）する必要があり、そのためのアドレス情報がユニフィケーション処理中にこのスタック内に格納される。

［コントロールスタック］

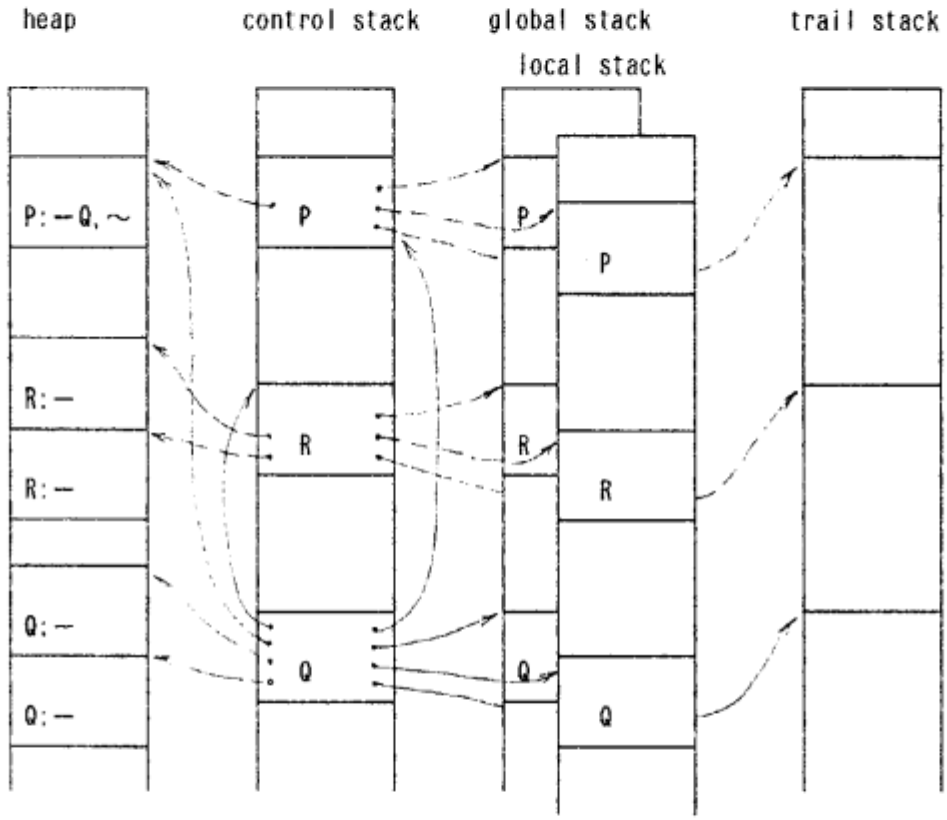
クローズの呼び出し、呼び出し元クローズへのリターン、バックトラック等の実行順序の制御に必要な情報はコントロールフレームとしてコントロールスタック中に格納される。全ての実行順序の制御はこのコントロールフレームを使用して行なわれる。コントロールフレームは10ワードで構成されており、各クローズの実行に対応して生成消滅される。フレームとして格納される情報には下表で示されるように自クローズの実行環境等に関する情報（i 、ii、iii 、v ）、自クローズの呼び出し元へリターンするための情報（vi、vii 、viii）、ユニフィケーションに失敗した際にバックトラックするための情報(ix 、x)、拡張された実行制御のための情報（iv）がある。（図－２）

(i)　自クローズを含むプロシージャのベースアドレス
(ii)　ローカルスタック中の自クローズが使用している変数領域のベースアドレス
(iii) グローバルスタック中の自クローズが使用している変数領域のベースアドレス
(iv)　自クローズの実行レベル番号
(v)　トレイルスタックの格納開始アドレス
(vi)　呼び出し元クローズのコントロフレームベースアドレス
(vii) リターン先コントロールフレームベースアドレス
(viii)リターン後に実行すべきリターン先クローズのボディゴールアドレス
(ix)　バックトラック時に実行すべきクローズアドレス
(x)　バックトラック先のコントロールフレームベースアドレス

b)　クローズ呼び出しとバックトラック

ボディゴールとのユニフィケーションの対象となるクローズは内部形式中のポインタを使用して呼び出され、ユニフィケーションは呼び出し元ボディゴールと呼び出された側のクローズヘッドとの間で行なわれる。

呼び出しの際には、ユニフィケーションのための変数領域が生成されると共に、クローズ


heap
control stack
global stack
local stack
trail stack
P:－Q,～
R:－
R:－
Q:－
Q:－
P
R
Q
P
P
R
Q

図－２　スタック構成

処理が終了した場合の呼び出し元クローズへのリターンのための情報、ユニフィケーションが失敗した場合のバックトラックのための情報が生成される。 通常、ユニフィケーションが成功した場合、呼び出された側のクローズを新たに呼び出し元クローズとして、そのボディゴールによる呼び出しが行なわれるが、ユニフィケーションに成功したクローズがユニットクローズであれば、呼び出し時に生成されたリターン情報に従って呼び出し元へリターンし、以後の処理はリターン先クローズの未実行ボディゴールから実行される。

ユニフィケーションに失敗した場合（フェイル）、呼び出し時に生成されたバックトラック情報に従って他の選択肢のある最近実行されたクローズ（バックトラックポイント）まで戻って、ユニフィケーションがやり直される。この際バックトラックポイント以降今までに生成されたローカルスタック、グローバルスタック中の変数領域の解放及び、バックトラックポイント以前に生成された変数領域であってもバックトラックポイント以後に行なわれたユニフィケーションでバインドされた変数の値はトレイルスタックに積まれている情報によってリセットされる。

3.1.2 ユニフィケーション

ユニフィケーションは呼び出し元のボディゴールの引数セルの値と呼び出されたクローズヘッドの引数セルの値とのパターンマッチングである。

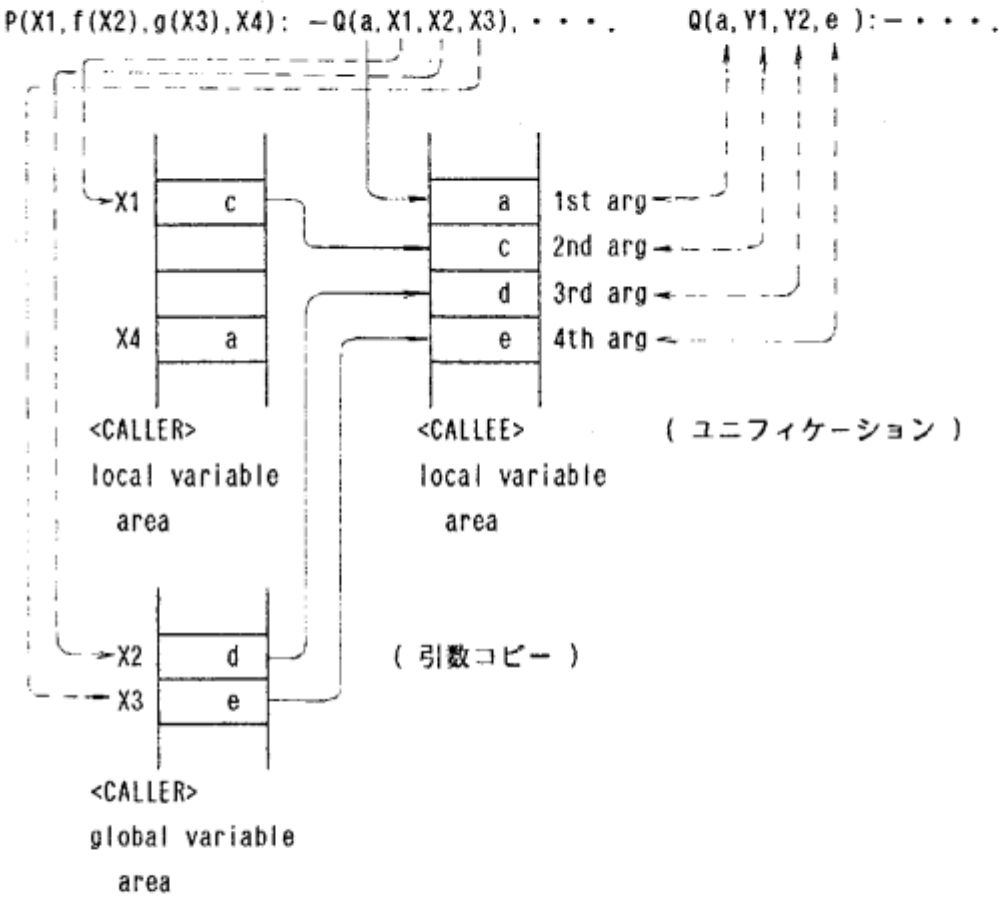
a)引数コピー

ユニフィケーションの手続きは 2フェーズに分離して処理することが可能である。

ψではユニフィケーションに先立って、呼び出し元の引数セルの値を呼び出される側の変数領域にまとめてコピーする（引数コピー）。ユニフィケーションはこのコピーされた変数セルの値と呼び出されたクローズのヘッド引数の値とを順次比較することによって行なわれる。このように従来のユニフィケーションを２フェーズに分離することにより若干のオーバヘッドはあるが、マイクロインタプリタはコピー時には呼び出し元の環境のみを、ユニフィケーションの際には呼び出されたクローズの環境のみを管理するだけで良くなる。又、ユニフィケーションの失敗によりバックトラックし、別のユニフィケーションの候補であるクローズ処理を行なう際には新たに引数コピーする必要がなく以前の引数コピーによって生成された値を再利用することが出来るという利点がある。（図－３）

b)ユニフィケーションルール

ユニフィケーションは以下のルールによって行なわれる。すなわち呼び出し元引数セルと呼ばれた側の引数セルがどちらも未定義変数セルでなければ、データタイプ及びデータの値が一致した場合にユニフィケーションに成功し、どちらか一方でも不一致の場合失敗する。少なくとも一方が未定義変数ならばユニフィケーションは常に成功し、一方が未定義変数で他方のセルにはある値がバインドされている場合、その値が未定義変数のセルにセットされる。又、両方とも未定義変数である場合、一方のセルから他方のセルへポインタが張られ、両方の未定義変数の値が論理的に同一であることを示される。



図－３　引数コピーとユニフィケーション

このルールのほかにベクタ同士、ストリング同士のユニフィケーションでは以下のような特別なユニフィケーションルールが適用される。

［ヒープベクタ］

本体がヒープ中に生成されるベクタであるヒープベクタ同士では呼び出し元、呼び出された側両方のデータタイプ、要素数及び本体アドレスが全て一致した場合にユニフィケーションに成功する。上記のどれか一つでも不一致の場合に失敗する。

［スタックベクタ］

本体がグローバルスタック中に生成されるベクタであるスタックベクタ同士では、呼び出し元、呼び出された側両方のデータタイプ、ベクタの要素数が一致し、かつ個々の要素同志のユニフィケーションに成功した場合に成功する。要素同志のユニフィケーションには通常のルールが適用される。

［ストリング］

呼び出し元、呼び出された側両方のデータタイプ、ストリングの要素数、要素のデータタイプ、オフセット及び本体アドレスが一致した場合にユニフィケーションに成功する。上記のどれか一つでも不一致の場合に失敗する。

3.1.3 テイルリカージョンオプティミゼーション

実行すべきゴールがまだ残っている場合、又はバックトラックによって実行されるべきクローズが残っている場合そのクローズの変数領域は保存する必要があるが、クローズの処理が決定的に終了し（バックトラックによって実行されるべきクローズが残っていない）かつ他に実行すべきゴールが無い場合（クローズの最後のゴールの実行）、もはやそのクローズの変数領域は不要である。引数コピー方式では、コピーした時点でユニフィケーション処理における呼び出し元クローズ処理は完了してしまう。したがって、マイクロインタプリタは引数コピーにより呼び出し元の変数セルの値を呼び出されたクローズの変数領域にコピーした時点で、もし対応する呼び出しがそのクローズの最終ゴールによるものであり、かつそのクローズの処理が決定的に終了する場合、呼び出し元の変数領域を解放することが出来る。これをテイルリカージョンオプティミゼーションと呼ぶ。引数コピー方式と一体となったこのようなテイルリカージョンオプティミゼーションは不要な変数セル領域によるスタックの消費を極力押えることが出来る。

3.1.4 バッファ

φでは処理の高速化のためにフレームバッファと呼ぶ呼び出されたクローズのユニフィケーションのための作業領域と、トレイルバッファと呼ぶトレイルスタックの先頭31ワード分のバッファをCPU 内部のワークファイルに持っている。

また、実行順序制御を高速に行なうための制御レジスタも持っている。

a)フレームバッファ

引数コピーにおける呼び出し元ボディゴールの引数の値はフレームバッファにコピーされ、ユニフィケーションはこのフレームバッファ内の値と呼び出されたクローズのヘッド引数との間で行われる。フレームバッファ内のデータ（ローカルフレーム）はユニフィケーション終了の後、必要に応じてローカルスタックの先頭にセーブされる。又、フレームバッファは2組用意されており引数コピーの際、もし一方のフレームバッファに呼び出し元クローズのローカルフレームが残っている場合、その値を使用して他方のフレームバッファに呼び出し元クローズのボディゴールの引数の値がコピーされる。

この2組のフレームバッファは、不要なメモリアクセスを極力押え以下のように引数コピー、テイルリカージョンオプティミゼーションの処理の高速化に貢献している。

［ユニフィケーション処理］

フレームバッファ内の値を使用してユニフィケーションが行なわれるため、メモリアクセスを極力少なくすることが出来る。

［ユニットクローズ処理］

ヘッド部しかないクローズ（ユニットクローズ）では、そのクローズに他の選択肢（バックトラックにより実行されるべきクローズ）が残っていない場合ユニフィケーション終了後はもはやそのローカルフレーム領域は保存する必要がないため、バッファ内のデータはローカルスタック内にセーブされることなく捨てることが出来る。

［ユニフィケーションの失敗時］

ユニフィケーションに失敗した時、そのクローズの他の選択肢にバックトラック（シャローバックトラック）する場合、フレームバッファ内の値はそのまま再利用することが出来る。又、現在実行中のクローズよりずっと以前にさかのぼってユニフィケーションをやり直す（ディープバックトラック）場合には、ユニフィケーションに失敗したフレームバッファ内のデータは不要となるためローカルスタックへのセーブを行なうことなく捨てることが出来る。

［組込述語処理］

組込述語は、クローズの呼びだしや呼び出したクローズとのユニフィケーションの必要がなくその組込述語を含むクローズの変数領域のみを使用して処理が行なわれる。したがって、クローズヘッドに直接続く組込述語の処理ではフレームバッファ内のデータのセーブはボディ部にユーザ定義述語が出現するまで延期され、その間はフレームバッファ内のデータがそのまま使用でき、メモリアクセスを少なくすることが出来る。

［トランジティブクローズ処理］

クローズのボディ部にユーザ定義述語が1つしかなく、かつその述語がクローズの最後のボディゴールである場合、そのクローズをトランジティブクローズと呼ぶ。トランジティブクローズの処理ではフレームバッファが2組あることにより、そのクローズに他の選択肢がない限りフレームバッファ内データは全くスタックにセーブされることなく処理が行なわれる。

なお、他の選択肢がなく決定的に処理が終了する場合の呼び出し元クローズのローカルフレームは、テイルリカージョンオプティミゼーションにより呼び出し元ボディゴールの引数を呼び出されたクローズ側の変数領域にコピーした後に解放される。フレームバッファは呼び出されたクローズの一時的な変数領域として使用され、ユニフィケーション処理終了後フレームバッファ中のデータはローカルスタックに格納されるため、上記のように呼び出し元クローズの変数領域が不要となった場合、呼び出されたクローズ側の変数領域をそこに上書き出来る。

b)トレイルバッファ

トレイルスタックに格納されるべきユニフィケーション失敗時のアンドゥのための情報は一旦トレイルバッファに積まれ、トレイルバッファが一杯になった時点でトレイルバッファからトレイルスタックに移される。最新のアンドゥのための情報を一旦トレイルスタックに積んだ後トレイルスタックに格納することは無駄な場合もあるが以下の点で大きな効果がある。

［ユニフィケーション失敗時］

ユニフィケーションに失敗した場合のアンドゥ処理はトレイルバッファに情報が残っている場合、その情報を取り出すためのメモリアクセスを必要としない。又、この情報はアンドゥ処理終了時には不要となるためメモリに格納することなく捨てることができる。

［カット処理時］

カットは、プログラム実行時に自クローズの他の選択肢を刈り取ることにより、不要な枝の探索を行なわないようにする機能である。トレイルスタック内の情報はバックトラックのアンドゥの際使用されるため、カットされた枝の情報はアンドゥ時にはもはや使用されることは無く、カット処理でその情報はトレイルスタックより取り除くことが出来る。この場合、トレイルバッファ中にあるデータはそれを取り除くだけでよい。

c)制御用レジスタ

マイクロインタプリタがKL0 プログラムの実行制御を効率良く行なうことが出来るように、ゆには29種類の実行制御用内部レジスタが用意されている。これらのレジスタには以下のものがある。

［実行環境用レジスタ］

実行中のクローズの使用している各スッタクのベースアドレス、実行プログラムのアドレスを保持する

［バックトラック用レジス］

実行中のクローズのユニフィケーションに失敗した際にバックトラック先クローズが使用するフレームのベースアドレス、及びそ際のプログラム実行アドレスを保持する。［リターン用レジスタ］

ユニフィケーションに成功して呼び出し元クローズへリターンするためのコントロールフレームのベースアドレス、及びそ際のプログラム実行アドレスを保持する。

〔拡張制御用レジスタ〕
遠隔のカット、バインドフック等の拡張された実行制御を実現するための情報を保持する。
〔トップレジスタ〕
各スタックの先頭及びプログラムの格納領域であるヒープの先頭アドレスを保持する。
〔ユニフィケーション用レジスタ〕
ユニフィケーションの際、構造体のスケルトンアドレス、フレームベースアドレスを保持する。

3.2 組込述語

頻繁に実行される基本的な処理をシステムの一部として述語の形式（組込述語）でユーザに提供することは実用的な処理系として有用である。ψではオペレーティングシステムのプログラムをはじめユーザプログラムの記述能力と処理能力を高めるため約 160種類の組込述語を用意している。

3.2.1 組込述語の種類

組込述語には以下の種類のものがある。
○データ操作用組込述語
変数にバインドされている値の属性チェック、データ構造の生成及び構造体データの要素又は部分要素のアクセスを行なうための組込述語。
○基本演算用組込述語
整数又は実数の算術演算、整数又はストリングの論理演算及び比較を行なうための組込述語。
○ハードウェア操作用組込述語
入出力装置へのアクセス、内部レジスタ等のハードウェアの内部資源へのアクセスを行なうための組込述語。
○OSインタフェース用組込述語
新しく生成するプロセスの実行環境の設定、ソフトウェアトラップの発生、メモリ管理を行なうための組込述語。○実行順序制御用組込述語
不要となった枝の検索をおこなわないようにするためのカット、特定の変数に値がバインドされた際に、指定された処理を起動させるように指示する等の拡張された実行制御用の組込述語。

3.2.2 エクセプション

組込述語の中には、算術演算のように演算結果のオーバフローや、不正な入力データによる例外事象を発生させるものがある。このような例外事象が発生した場合、無条件にプ

ログラムの実行が停止されたり、処理が失敗（フェイル）するよりも例外事象が発生した場合に行なう処理をユーザが指定できることはプログラムの柔軟性を増し、デバッグも容易になる。したがって、φでは組込述語の処理に際して引数の入力条件、出力条件のチェックを行ない、例外事象を検出するとその例外事象に対応したハンドラを起動する機能を備えている。基本的なハンドラはシステムが提供しているが、ユーザが自由にハンドラを登録することもできる。

例外事象が検出された場合のハンドラの起動は例外事象を発生させた組込述語が呼び出し元クローズとなって通常のクローズ呼び出しと同じ機構でハンドラを呼び出したかのように行なわれ、エクセプションハンドラの処理が完了すると例外事象を発生させた組込述語の次の述語から処理が続行される。例外事象の情報（OPコード、例外原因情報等）は、ハンドラのヘッド部に引数の値として引き渡たされる。

3.3 OSサポート

OSの機能のうちハードウェアに近い機能として割込み処理、プロセススイッチ、実メモリの割付け等がある。割込み処理及びプロセススイッチは頻繁に発生するためこれらの処理速度はシステム全体の性能に大きく影響する。したがってφでは割込み処理及びプロセススイッチの大部分をファームウェアによってサポートすることにより処理の高速化を計っている。又限られた実記憶を有効に使用することは重要なことでありφでは実行中のプログラムに対する実メモリの割当て及び不要な実メモリの回収をソフトウェアとファームウェアとが協力して行なっている。

3.3.1 プロセス管理

φでは最大63個までのプロセスを同時にかつ独立に実行することができる。各プロセスにはプログラムの実行順序制御管理のため、論理的に独立な 4本のスタック（ グローバル、ローカル、コントロール、トレイル）が割当てられる。現在実行中のプロセスの処理を中断し別のプロセスの処理を行なうためには中断されるプロセスの処理が後で続行出来るようにそのプロセスの実行制御情報を格納する必要がある。実行制御情報としてφにはプロセスコントロールブロック（PCB)、拡張プロセスコントロールブロック（EPCB）及びエリアトップポインタ（ATP)がある。PCB にはプロセスの実行プライオリティ、プログラムの置かれているエリア情報等のソフトウェアとファームウェアとのインタフェース情報が含まれている。EPCBには内部レジスタの値、使用スタックの管理情報等のそのプロセスのプログラムを続行するためにマイクロインタプリタが必要とする情報が含まれる。ATP には各スタックのスタックトップ位置の情報が含まれている。プロセスの生成ではプログラムの指示に従ってマイクロプログラムによりPCB 、EPCB、ATP の初期化が行なわれる。また実行中のプロセスから別のプロセスへのプロセススイッチでは、実行中のプロセスのPCB 、

EPCB、ATP がCPU の内部メモリ及び各プロセスのローカルスタックの底に格納され、新たに実行されるプロセスに対応したPCB 、EPCB、ATP がCPU の内部メモリ及びローカルスタックの底からCPU 内に取り出される。各プロセスに対応したCPU 内部メモリ及びローカルスタックの底の格納エリアとしてそれぞれ32ワードが確保されている。

3.3.2 メモリ管理

φでは仮想記憶は採用していないが、物理メモリを効率良く使用するため論理空間と物理空間とを分離している。物理メモリはページと呼ばれる単位で管理されており、論理アドレスから物理アドレスへの変換はアドレス変換テーブルを介して行なわれる。この変換テーブルの管理はソフトウェアとファームウェアとが共同して行なう。未使用の物理メモリ量はファームウェアで管理されており、各プロセスへの物理メモリの割当てはプログラムで指定された量を越えず、割当てるべき物理メモリがある限りマイクロインタプリタが必要に応じて自動的に各プロセスに割当てる。プロセスに割当てるべき物理メモリが無くなった場合にはトラップによりガベージコレクタが呼び出される。

3.3.4 割込み処理

φにおける割込み機構はプロセススイッチで実現している。割込みが検出されると割込み原因は割込み原因テーブルにセットされ、割込み原因に対応したハンドラのプロセス番号が割り込みインデックステーブルから取り出される。そのプロセス番号に従って、割込まれたプロセスはハンドラプロセスにプロセススイッチされる。通常の割込み検出はユーザ定義述語ではユニフィケーションの成功または失敗の直後、組込み述語では処理の成功又は失敗の直後に行なわれる。（図－４） しかし、処理時間の長いユニフィケーション処理又は組込み述語処理では高速入出力装置等の緊急度の高い割込みが発生した場合、通常の割込み検出タイミングまで処理を延期することなく処理の途中でも割込みを受けつける。割込み原因テーブルにセットされる割込み原因は原因コード、パラメータ及び割込まれたプロセスのプロセス番号から構成される。

φには図－５のように10種類の割込み（トラップ）原因がある。なお、プログラムに非同期に発生する原因を割込み、プログラムに同期して発生するものをトラップと呼ぶ。

```
P：－Q，ｂ１，ｂ２，Ｒ．
  ↑          ↑    ↑
Q：－Ｓ．
  ↑
Ｓ．        Ｒ．
  ↑          ↑
```

Ｐ，Ｑ，Ｒ，Ｓ：ユーザ定義述語
ｂ１，ｂ２：　組込述語
↑：割込みチェクタイミング

図－４

・ハードウェアエラートラップ
・プログラムエラートラップ
・ガベージコレクタ呼び出しトラップ
・エリアリミットチエックトラップ
・トレーストラップ
・オペレータ割込み
・パワーフェイル割込み
・I/O 割込み
・タイマ割込み
・メモリエラーコレクト割込み

図－５

4．おわりに

マイクロインタプリタの開発はほぼ完了し、現在ψの上でOSの開発が行なわれている。又これと平行してマイクロインタプリタの評価、ガベージコレクタの開発が行なわれている。性能は当初の目標である30KLIPS を達成出来る見込みである。今後はシステム全体としての評価も行ない、さらに高速化するためにOSの一部ファームウェア化等も考えて行く予定である。

最後に、日頃有益な助言をいただくＩＣＯＴメンバ及び協力いただいたメーカの方々に深く感謝する。

参考文献

[1] Chikayama,T., et.al.,Fifth Generation Kernel Language,Proc. of Logic Programming Conference '84,Japan,1983.

[2] Uchida, S., Yokota, M., Yamamoto, A., Taki, K., and Nishikawa, H., Outline of the Personal Sequential Inference Machine: PSI, New Generation Computing, Vol. 1, No.1, 1983.

[3] Warren, D.H.D., Implementing Prolog-Compiling Predicate Logic Program Vol. 1-2, D.A.I.Research Report, No.39-40, Department of Artificial Intelligence, Univ. of Edinburgh, 1977.

[4] Warren, D.H.D., An Impoved Prolog Implementation which Optimizes Tail Recursion, D.A.I.Research Report, No.156, Department of Artificial Intelligence, Univ. of Edingburgh, 1980.

[5] Yokota, M., Yamamoto, A., Taki, K., Nishikawa, H., and Uchida, S., The Design and Implementation of a Personal Sequential Inference Machine: PSI, New Generation Computing Vol.1, No.2, 1983.