ＥＳＰプログラムのＣＥＳＰシステムへの移植事例

2Q－10

田島　守彦　（電子技術総合研究所）

１．はじめに

このたびESPで開発中の学習システムＲＬＳ(Recursive Learning System)[1]を SPARC station2上のCESPシステム[2](3.2版)に移植した。文法的にはほとんど違いがないが、環境が大きく異なることに起因する困難があった。変換の経験者も多くなく、また変換記録も残されていなかったので、本稿で問題点および解決策についてまとめておく事は有用と思われる。関連して両者の動作の比較も行う。

２．ＲＬＳと動作環境

RLSは現在12MW(40bit/W)のﾒﾓﾘをもつMELCOM PSI-II(200ns版)およびPSI専用のOSであるSIMPOS上にESPで開発されている。RLSに加え、RLS上で動くｹﾞｰﾑﾌﾟﾚｲおよびｹﾞｰﾑ学習を行うﾌﾟﾛｸﾞﾗﾑであるRLS-Othelloも同時に移植した。ESPおよびCESPはｵﾌﾞｼﾞｪｸﾄ指向型の言語なので、継承機能やｲﾝｽﾀﾝｽの生成機能が特徴である。RLSおよびRLS-Othelloは合わせて12個のﾌｧｲﾙに分割されており、ｸﾗｽ数は107、ｿｰｽﾌﾟﾛｸﾞﾗﾑは152K byteである。RLSは実行中に膨大な数のｲﾝｽﾀﾝｽを生成するという特徴がある。

移植先はSUN Microsystems の SPARCstation 2 でOSは4.1.1、主記憶は48Mﾊﾞｲﾄ、ｽﾜｯﾌﾟ領域は92Mﾊﾞｲﾄである。CESPには機械語生成版とｴﾐｭﾚｰﾀ版があり、我々は機械語生成版を使用した。またCESPのｳｨﾝﾄﾞｳ機能はX-Windowを使用する。

３．言語および環境の相違

CESPの言語仕様はESPのそれの拡張になっており、移植に当たっては言語上の大きな問題は生じない。組み込み述語についてもほぼ同じものが用意されている。問題になるのは主に環境である。ESPはOS(SIMPOS)のもとで働いており、ｳｨﾝﾄﾞｳやﾌｧｲﾙなどの資源を初め、Prologでは言語に含まれている:assertや:retract、:clauseなどのメタ述語などもSIMPOSが提供している。CESPではこれらに相当する述語をCESP自身が持っているが、ｸﾗｽやﾒｿﾄﾞのかなりのものがESPのものと異なっている。特にｳｨﾝﾄﾞｳ回りの環境の細部が異なっているので移植作業が必要とされる。

ｳｨﾝﾄﾞｳ回りの移植にあたっては、通常全く書き直すことが行われている。が、我々は次の２方法で変換を行った。

(1)SIMPOSクラスやメソドの定義

大抵の場合にはCESPに用意されているｸﾗｽを多重継承することで必要なｸﾗｽやﾒｿﾄﾞを定義出来た。この目的のためにespclassと称する専用ﾌｧｲﾙを用意し、そのような12ｸﾗｽを一括して定義した。

その際、被継承ｸﾗｽで定義されているﾒｿﾄﾞ自身を、そのﾒｿﾄﾞを継承する側で利用し同名のﾒｿﾄﾞを再帰的に定義することも出来る。この場合にはｽｰﾊﾟｰｸﾗｽのﾒｿﾄﾞのｶｯﾄによる制御が必要である。

(2)直接的な書換え

画素数のようなﾊｰﾄﾞｳｪｱ･ﾚﾍﾞﾙの違いの解決、あるいは将来ESP側の変更がないｸﾗｽの移植では、直接ﾌﾟﾛｸﾞﾗﾑの中味を書き換えた。

効率という面では(2)が良いが、ESPプログラムとの同一性維持、変更部分の明示の目的には(1)が良い。我々は可能な限り(1)の方法を用いた。

４．個々の相違と変換

変換例を種類毎に示す。ここに挙げたのは我々が必要としたものの一部のみであり、全てを尽くしているわけではない。

４．１　被変換クラス

(1)CESPｸﾗｽの(多重)継承で直接定義

standard_scrolling_menu
standard_io_window

A Case of Transplantation of ESP Programs to the CESP System
TAJIMA, Morihiko
Electrotechnical Laboratory

window_without_label
as_mouse_input
labeled_sash
as_scroll
monogyny_hash_index
library_utility

(2)CESPｸﾗｽを継承したｸﾗｽを別に定義し利用

CESPとESPで同名で機能の異なるﾒｿｯﾄﾞがある場合には間接的にCESPｸﾗｽを利用するのがよい。

choice_window
popup_choice_window

(3)全く新しく定義

as_markers（markerを別に定義して利用する）

４．２　メソド・マクロ等の変換

ﾒｿｯﾄﾞの名称、引数の形式など不必要と思われる変更点が多い。左がESP、右がCESPである。

:kill(Window)	:close(Window)
:retract_all/4	:abolish/4
:write_lines/2	:write_line/2
:write(W,key#cr),:write(W,key#lf)	前半のみ
:get_mouse_position/3	unitで割って使用
:draw_line/8	:draw_abs_line/5
:draw_filled_circle/5	同名ﾒｿｯﾄﾞ/7
#font_13	"a14" または "a24"
control#"c" etc.	key#etx etc.
mouse#r etc.	'mouse#r' etc.
key#enter, keypad#'0' etc.	なし
dynamic((p1/a1, ..))	dynamic p1/a1, ..

４．３　その他

移植時には、項目とｸﾘｯｸの種別(1,m,r)を同時に返すﾏｳｽ読み込みのためのﾒｿｯﾄﾞ :read_with_click/3がなかった。CESPで同一機能を実現するのは困難であるため要望したところ、最新版3.3で取り入れられた。文字列の基点の位置が異なる(ESPでは左上、CESPでは左下)。これなど不必要な変更である。

ｶﾀﾛｸﾞ(ｺﾝﾊﾟｲﾙ)ではﾏﾆｭｱﾙにはない制限があった。CESPｼｽﾃﾑでは、ｽﾛｯﾄ初期化述語で使用する被参照ｸﾗｽは別ﾌｧｲﾙにして、参照ｸﾗｽのｶﾀﾛｸﾞの事前にｶﾀﾛｸﾞする必要がある。不必要な制限である。

５．速度比較

両者でのｶﾀﾛｸﾞ時間を計測した。ただしCESPプログラムにはespclassのﾌｧｲﾙが含まれる。

ESP on PSI-II	CESP on SS2
3′30″	11′15″

ESPのｶﾀﾛｸﾞが3倍以上速い。

単純なﾌﾟﾛｸﾞﾗﾑ例での速度比較を行った。ackerman/3 はAckermann関数である。reverse/2 はﾘｽﾄの反転を行う。要素内も再帰的に反転する。入力例は各ﾚﾍﾞﾙに10要素、10ﾚﾍﾞﾙのﾘｽﾄである。

	ESP on PSI-II	CESP on SS2
ackerman(3,8,R)	33″	51″
reverse(L,R)	6″	10″

すなわちESPがCESPの約1.5倍速い。

ｳｨﾝﾄﾞｳへの表示はCESPｼｽﾃﾑが幾分高速であった。ﾊｰﾄﾞ的な表示能力の違いであろう。

６．結び

CESPへの書換はｳｨﾝﾄﾞｳ回りを除けばそれほど煩雑ではない。しかしｳｨﾝﾄﾞｳ回りでは、ﾒｿｯﾄﾞ名や引数、引数形式の不必要と思われる変更が行われており、ﾕｰｻﾞを混乱させる。

CESPのような言語の開発者は移植のためのﾕｰﾃｨﾘﾃｨをﾕｰｻﾞｰに提供する事が望ましい。その望ましい形としては、①ESP→CESP変換のための変換ﾌﾟﾛｸﾞﾗﾑ、②我々のespclassのようなﾓｼﾞｭｰﾙの提供、③変換事例集や対応表　のような順序になろう。

CESPｼｽﾃﾑのｶﾀﾛｸﾞ時間はESPに比較して非常に遅い。改善が必要であろう。またｶﾀﾛｸﾞ順にﾏﾆｭｱﾙにない制限があった。

PSI-III(MELCOM PSI/UX) のﾌﾟﾛｸﾞﾗﾑ実行速度はPSI-II(200ns)の約4倍だという。一方CESPｼｽﾃﾑにおいては、機械語版のﾌﾟﾛｸﾞﾗﾑ実行速度はｴﾐｭﾚｰﾀ版の約2倍である。AI言語研究所の計画では、近い将来CESPはｴﾐｭﾚｰﾀ版のみになる。しかしPSI-III･ESP/SS2･CESP(ｴﾐｭﾚｰﾀ版)のﾌﾟﾛｸﾞﾗﾑ実行速度比は約12になる。この速度比の大きさを考慮すると、その計画は時期早尚かも知れない。

謝辞

本研究は㈱ＡＩ言語研究所との共同研究です。移植にあたってはAI言語研究所の田中吉廣、松浦聡の両氏にご指導頂きました。

文献

1)Tajima, M. and Sanechika, N.: Utilization and Learning of Knowledge in Game Programs, Proc. of the Game Playing System Workshop, pp.40-41, Tokyo (1991).
2)中澤修, 近山隆: ｵﾌﾞｼﾞｪｸﾄ指向論理型言語CommonESP, 情報処理学会誌, 32, 6, pp.694-703 (1991).