Ⅰ.「日本発・世界初のプログラム革命」であるシナリオ関数とは

講演者：根来文生

1．プログラムの観察結果
2．プログラムの在り方（プログラムが具備する条件）
3．シナリオ関数の構造
4．シナリオ関数の動性
5．シナリオ関数の特徴
6．プログラム革命の重要性
7．終わりに

■質疑応答

根来　文生（Negoro Fumio）とは

１９４０年　三重県　柏崎村で生まれる。
１９７１年　日本大学　理工学研究所（院）、数学専攻修士卒、日本大学講師
１９７８年　ISD研究所設立
２００３年　岩手県立大学　ソフトウエア情報学博士課程卒、博士（学位論文：ソフトウエア構築の摂理に関する統一理論）

１．プログラム開発技法　VOCJUの創立論考
２．LYEE理論の創立論考
３．システム開発の指導：多数
４．LYEE理論の国際学術研究（３年間）、IOS出版（NEU TRENDS IN SOFTWARE METHODOLOGIES TOOLS AND TECHNIQUES（Vol.84,98,111,129）
５．シナリオ関数の創立論考
６．LYEE理論の招待講演（ドイツ、フランス、イギリス、イタリア、デンマーク、スエーデン、イタリア、アメリカ）：１７大学、２研究所
７．その他論文あり
８．趣味
①囲碁、日本棋院神奈川県支部連合会元会長、
②母に読ませる為の小説、１７年前から執筆中（出版社は決定しているが未完）：「春の夢」

１．プログラムの観察結果

１９７３年に始められた私の研究では２００８年迄、継続して行われたことはプログラムのあるべき姿を捉える為に行われたプログラムの解析である。この解析法は今日では「主語系譜」と呼ばれている。プログラムに内在する全名詞（領域名）の主語（領域に成立する様相）系譜を取り出す方法で、プログラムのあり様を観察することが出来る。従来プログラムでは５００本以上、私が誘導した方法論のプログラムでは３００本ぐらいである。

Ａ．観察結果

①従来のプログラムは例外なく論理結合型である。
②論理結合型とはソースの作り方並びにその実行形態のあり様が共に非同期型となる様相のことである。

Ｂ．われわれの認識法、思考法、行動様式が論理結合に帰着する理由

①われわれは非同期の世界で成立し存在する：時制
②故に、われわれの思考、認識を成立させる「脳作」は無意識的に非同期の様相を捉える仕組みなっている。
③結果、われわれの行動様式もまた無意識的に論理結合に帰着する。
④そして、プログラムの解析から論理結合の成立には限界があることがわかった。：主語数

C．論理結合型プログラムの課題（1/2）

論理結合型プログラムの解析作業で明らかになったことを以下に示す。

①電算機システムを完成させるツールは応用プログラムである。ＯＳではない。
②論理結合型プログラムのプログラマは、プログラムは作れるが、内在する問題点を解法することが出来ない。だれも教えていないからである。
これはウイルス作成者がウイルスの本質性がわかっていないのウイルスを作る振る舞いに似ている。
③ＳＥの役割は本来的にはプログラマの補佐役である。
④量子コンピュータの高速性はＯＳを排除することに貢献できる点にある：消費電力
⑤電算機システムは利便性が得やすいので、関係者は介在する課題を論考する哲学的観点を見失っている：現状
⑥われわれには論理結合型プログラムの存在証明が出来ない：
２００８年には、われわれが出来る論理結合型プログラムの正統性判定は全体の７割ぐらいであることの証明が出来た。
⑦この分野の殆どの人々は、プログラムは動けばよいとだけ思っているのは論理結合型思考法がその原因である。

C．論理結合型プログラムの課題(2/2)

⑧電算機システムを高度化させる上でプログラムは、電子部品の数を少なくし、
電子部品の精度を補完し、応用操作を簡易化させる為の最終的な役割を担う存在である。
しかし、そのプログラムに欠陥があれば、この役割は担えない。プログラム欠陥とは
われわれの論理思考で導かれるプログラムでは解法できない問題の事である。
⑨論理結合型プログラムのバグはゼロにはならない。
⑩論理結合型プログラムの間にはデッドロック問題がある。
⑪ウイルスは論理結合型の阻止技術では防げない。
⑫論理結合型プログラムの開発維持費用は年々高騰化する。
⑬ＩＴ分野の歴史は問題の解法ができないことで潤ってきた。
⑭プログラム課題の解法を見失ったこの分野の行く末には暗澹たるものがある。
⑮プログラム問題は論理結合型プログラムの欠陥により生じる。
⑯論理結合型プログラムでビッグデータ、人工知能、ロボット、自動運転などのアイデアが
横行しているのは恥ずべき現実である。
⑰われわれは原子力と同様に電算機を手にはしたが、原子力には深刻な事故が起きれば、
われわれの手には負えない様に、ウイルスで、現状のＩＴ文明の崩壊はたやすいことを
肝に銘じなければならない。

２．プログラムの在り方（プログラムが具備する条件）

①開発時は非同期性、故に、命令の存在証明が不可欠、

②実行時は同期性

③プログラムの３題問題（ウイルスの無力化（免疫性）、バグレス、プログラム間の自動調和性）の自動解法
④テストレス

⑤プログラムの自動生成法の成立：自動生成ツール

3．上記２．を充足させるシナリオ関数の構造

①シナリオ関数の定義式

②ベクトル
③座標関数
④同期関数
⑤LYEE空間定義テーブル
⑥シナリオ関数の解

$$S = \Phi 0(\Phi 4[\{L4\},\{W4\},T4,E41,E42] + \Phi 2[\{R2\},\{L2\},T2] + \Phi 3[\{L3\},T31,T32,T33])$$

S：プログラムの解　Φ０：同期関数　Φ4・Φ2・Φ3：座標関数

T4･E41･E42･T2･T31･T32･T33：制御ベクトル

主語ベクトル(主語系譜図で表現)

［L4：代入文，W4：出力文，R2：入力文，L2：定値・定置文，L3：条件文】

３．上記２．を充足させるシナリオ関数の構造

①シナリオ関数の定義式
②ベクトル
③座標関数
④同期関数
⑤LYEE空間定義テーブル
⑥シナリオ関数の解

参照図：同期関数と座標関数＆ベクトル

同期関数（SF）の起動点（この起動点は座標関数の帰り点でもある）

（LYEE空間で見るシナリオ関数で用いられる全領域は初期値化状態で定義されて、
同期関数にデータDIVとしてまとめて編集されておかれる。）

同期関数の処理が終了したかを問う
問い方はE41の第４領域がオン？である。

正常終了：終了手続きをとる
プログラム（SEP）を
起動させて終了する。

＊１同期周期とは同期関数の起動点から始まり、帰り点に戻ることを
３度繰り返す状態の事である。
帰り点に戻ることを３度繰り返す状態の事である。
＊シナリオ関数の本例は最大５同期周期１５度で終了する。

同期関数の処理が終了していなければ、同期関数は３種の座標関数のどれを
起動させるかを決めて、その座標関数を起動させる。これが同期関数の仕事の
全てである。起動させる座標関数の決め方は重要である。即ち、以下の手順

T31の第４領域はオン？ YS → T31の第４領域をリセットし同じシナリオ関数の座標関数４を起動する。
NO

T4の第４領域はオン？ YS → T4の第４領域をリセットし同じシナリオ関数の座標関数４を起動する。
NO

T2の第４領域はオン？ YS → T2の第４領域をリセットし同じシナリオ関数の座標関数４を起動する。
NO

T32の第４領域はオン？ YS → T32の第４領域をリセットして、ランク構造の下位の指定されたシナリオ関数の座標関数４を起動する。
NO

T33の第４領域はオン？ YS → T33の第４領域をリセットして、ランク構造上位のシナリオ関数の座標関数３を起動する。
NO

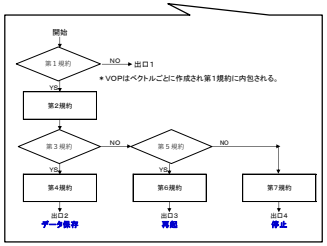
同じシナリオ関数の座標関数４を
起動して、この同期関数を出る。
（初期起動の時）

（連鎖）

座標関数４

パレット４：［｛L4，｝，｛W4，｝，E41，E42，T4］に
並ぶベクトルの順位１のベクトルを指定する。

パレット４に並ぶ指定されたの順位のベクトル起動する。 → ベクトルの叙述構造



起動を終えたベクトルの正統性を判定 異常 → SLPの起動

正統 → E42の第４領域がオン、又はOS障害が発症していれば、ここでシステムを停止させる為にSLPを起動させます。SLPはその為のプログラムです。

ベクトルの順位を更新

更新したベクトルの順位をオーバー？
NO → 更新されたパレット４ → パレット４に並ぶ指定されたの順位のベクトル起動する。

パレット4の全ベクトルの成立を問わず前ベクトルを

パレット4の全主語ベクトルは臨界か？

臨界とは全主語ベクトルの第６フラグが全てリセットされている状態の事です。

パレット4の全ベクトルの第７フラグをリセット

戻りカウンタ（RC）をプラス１する

スタック（NS）の更新プログラムを起動して戻る

同期間数の先頭に戻る

補記：シナリオ関数の起動規則

（１）シナリオ関数を構成する同期関数、３種の座標関数、
スタック更新プログラム、ベクトル、SEP、SLP、はすべて
STARTとENDで単位化されｔいる
（２）同期関数は開始起動し、座標関数は同期関数で起動される。
（３）ベクトルは座標関数で１個づつ起動される。
（４）スタック更新プログラムは座標関数で起動される。
（５）座標関数は起動を終えると同期関数に戻る。
（６）ベクトルは起動を終える都度その座標関数に戻る。
（７）SLP、SEPは起動されるとシナリオ関数にはリターンをしない。
（８）SLP、SEPはクライアント作成問題と置き、本願では関与しない。

＊ＲＣ値がある値を超えているのに
Ｅ４１の第４領域がオンにならないのは
①仕様書に誤記があるか
②ウィルスで実行命令が置換されたかの
何れかが原因である。

主語の変数主語を最大５個とする場合の
本例に於いては、
ＲＣの限界数（シナリオ関数を継続する値）
は１５度に設定される。

＊戻りカウンタ（ＲＣ）はシナリオ関数に
起きている論理矛盾（仕様書の誤記、
ウィルスによる命令置換が原因）を捉える為に
ベクトルＥ４２で使用される。

３．上記２．を充足させるシナリオ関数の構造

①シナリオ関数の定義式
②ベクトル
③座標関数
④同期関数
⑤LYEE空間定義テーブル
⑥シナリオ関数の解

No	項目	備考
1	主語名	主語を格納する名詞の領域名
2	ベクトル記号	L2、R2、L4、W4、L3のいずれか
3	ベクトル名	ベクトル記号＋「_」+主語名で自動生成。ルールでは「,」で区切ることになっているが、 プログラミング言語仕様の問題で「_」で区切る。
4	実行命令	プログラミング言語上の実行命令
5	変数主語	変数主語を「,」で区切って入力
6	第4領域名	主語+「_f4」で自動生成
7	コピー領域名	主語+「_f4copy」で自動生成
8	型	主語、第4領域、コピー領域のプログラミング言語上の型
9	配列	主語、第4領域、コピー領域が配列の場合のプログラミング言語上の定義
10	初期値化コード	主語、第4領域、コピー領域を初期値化する際のプログラミング言語上のコード
11	オン判定コード	文脈チェックのためのプログラミング言語上の判定用コード
12	R2ベクトル専用属性チェック	R2ベクトル専用の属性チェック用コード *こちらは時間的な都合で補助金の締め切りまでに自動生成ツールに実装される可能性は低い。

LYEE空間定義テーブル：サンプル例

主語名	ベクトル記号	ベクトル名	実行命令	変数主語	第4領域名	コピー領域名	型	配列	初期値化コード	オン判定コード	最終ベクトル

fp1	L4	L4_fp1	if (fp1 == 0)
			{
			<tab>fopen_s(&fp1, "G:¥¥Projects¥¥LYEE¥¥ScenarioTools¥¥TestProject14¥¥test.txt", "rt");
			}
			
					fp1_f4	fp1_f4copy	FILE*		fp1 = 0;	fp1 != 0	FALSE

fp2	L4	L4_fp2	if(fp2==0)
			{
			<tab>fopen_s(&fp2,"G:¥¥Projects¥¥LYEE¥¥ScenarioTools¥¥TestProject14¥¥test2.txt", "wt");
			}
			
					fp2_f4	fp2_f4copy	FILE*		fp2 = 0;	fp2 != 0	FALSE

inBuf	R2	R2_inBuf	if (inBuf_3_f4 != 0)
			{
			<tab>memset(inBuf, 0, 80);
			<tab>fgets(inBuf, 80, fp1);
			<tab>eof_Initialize();
			}
			
				fp1,inBuf_3	inBuf_f4	inBuf_f4copy	char	[80]	memset(&inBuf, 0, sizeof(inBuf));	inBuf[0] != 0	FALSE

inBuf_3	L3	L3_inBuf_3	if (fp1)	fp1	inBuf_3_f4	inBuf_3_f4copy	int		inBuf_3 = 0;	inBuf_3 != 0	FALSE
outBuf	L4	L4_outBuf	memcpy_s(outBuf, 80, inBuf, 80);	inBuf,outBuf_3	outBuf_f4	outBuf_f4copy	char	[80]	memset(outBuf, 0, sizeof(outBuf));	outBuf[0] != 0	FALSE
outBuf_3	L3	L3_outBuf_3	if (inBuf[0] !=0 && eof == 1)	inBuf,eof	outBuf_3_f4	outBuf_3_f4copy	int		outBuf_3 = 0;	outBuf_3 != 0	FALSE

bWrite	W4	W4_bWrite	fputs(outBuf,fp2);
			inBuf_Initialize();
			inBuf_3_Initialize();
			bWrite_Initialize();
			bWrite_3_Initialize();
			outBuf_Initialize();
			outBuf_3_Initialize();
				fp2,outBuf,bWrite_3	bWrite_f4	bWrite_f4copy	int		bWrite = 1;	bWrite != 0	FALSE

bWrite_3	L3	L3_bWrite_3	if (fp2 && outBuf[0] != 0)	fp2,outBuf	bWrite_3_f4	bWrite_3_f4copy	int		bWrite_3 = 0;	bWrite_3 != 0	FALSE

close1	L4	L4_close1	if (fp1 && close1_3 == 1)
			{
			fclose(fp1);
			}
				fp1,close1_3	close1_f4	close1_f4copy	int		close1 = 0;	close1 != 0	FALSE

close1_3	L3	L3_close1_3	if (fp1 != 0 && eof == 2)	fp1,eof	close1_3_f4	close1_3_f4copy	int		close1_3 = 0;	close1_3 != 0	FALSE

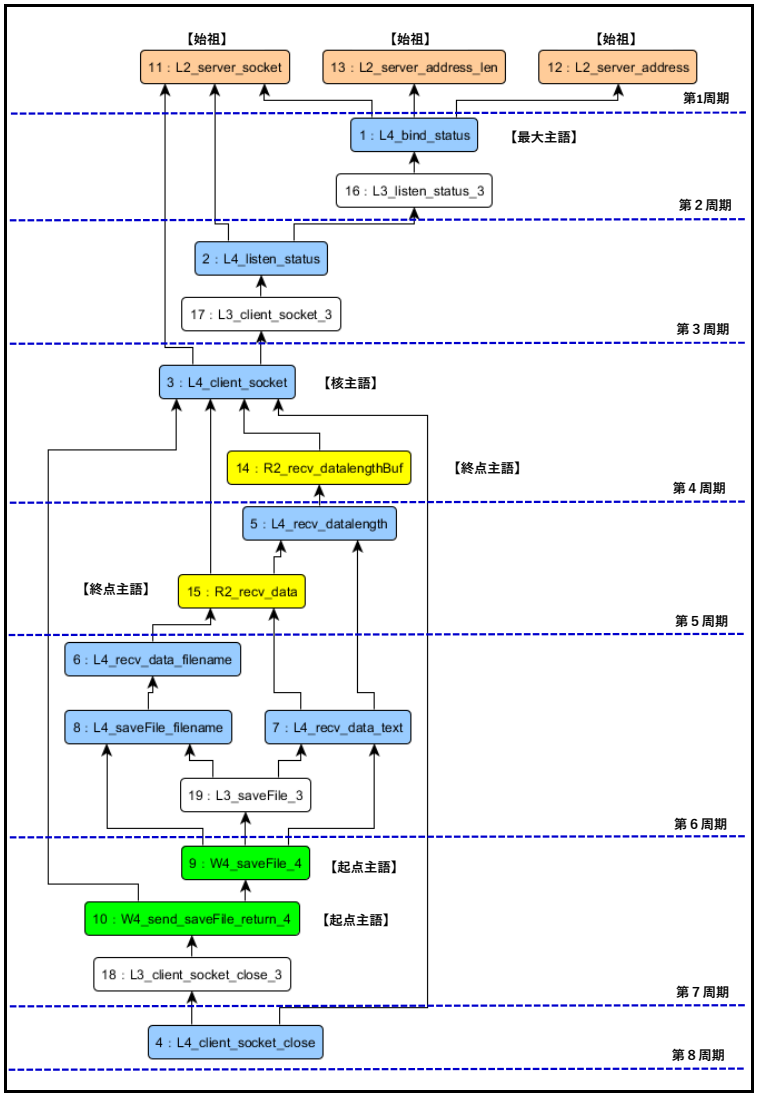
close2	L4	L4_close2	if (close2_3 == 1)
			{
			fclose(fp2);
			}
				fp2,close2_3	close2_f4	close2_f4copy	int		close2 = 0;	close2 != 0	FALSE

close2_3	L3	L3_close2_3	if (fp2 != 0 && bWrite == 1 && eof == 2)	fp2,bWrite,eof	close2_3_f4	close2_3_f4copy	int		close2_3 = 0;	close2_3 != 0	FALSE
closeAll_3	L3	L3_closeAll_3		close1,close2	closeAll_3_f4	closeAll_3_f4copy	int		closeAll_3 = 0;	closeAll_3 != 0	TRUE
eof	L4	L4_eof	eof = feof(fp1)? 2 : 1;	fp1	eof_f4	eof_f4copy	int		eof = 0;	eof != 0	FALSE

3．上記２．を充足させるシナリオ関数の構造

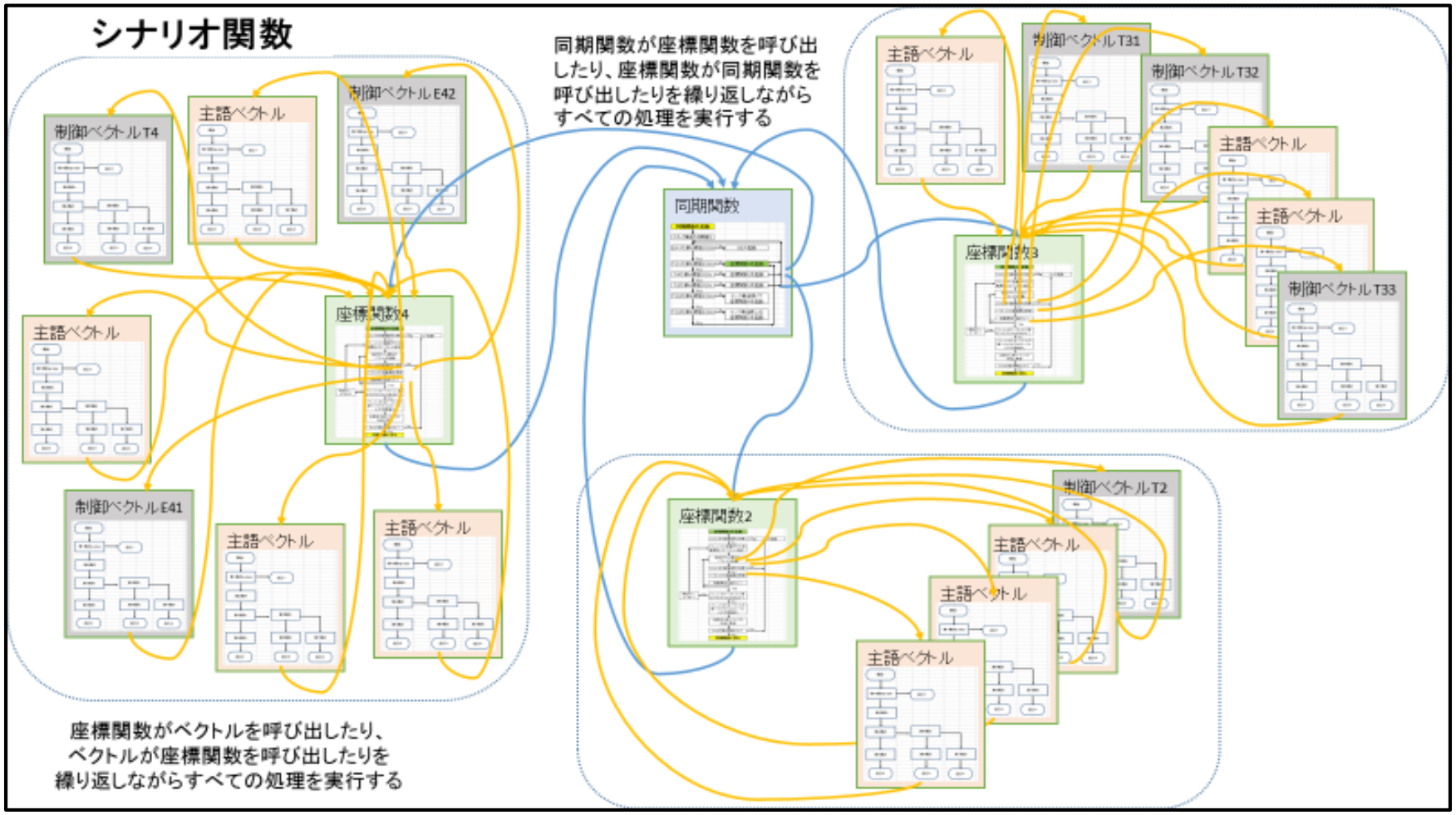
①シナリオ関数の定義式
②ベクトル
③座標関数
④同期関数
⑤LYEE空間定義テーブル
⑥シナリオ関数の解

主語系譜図



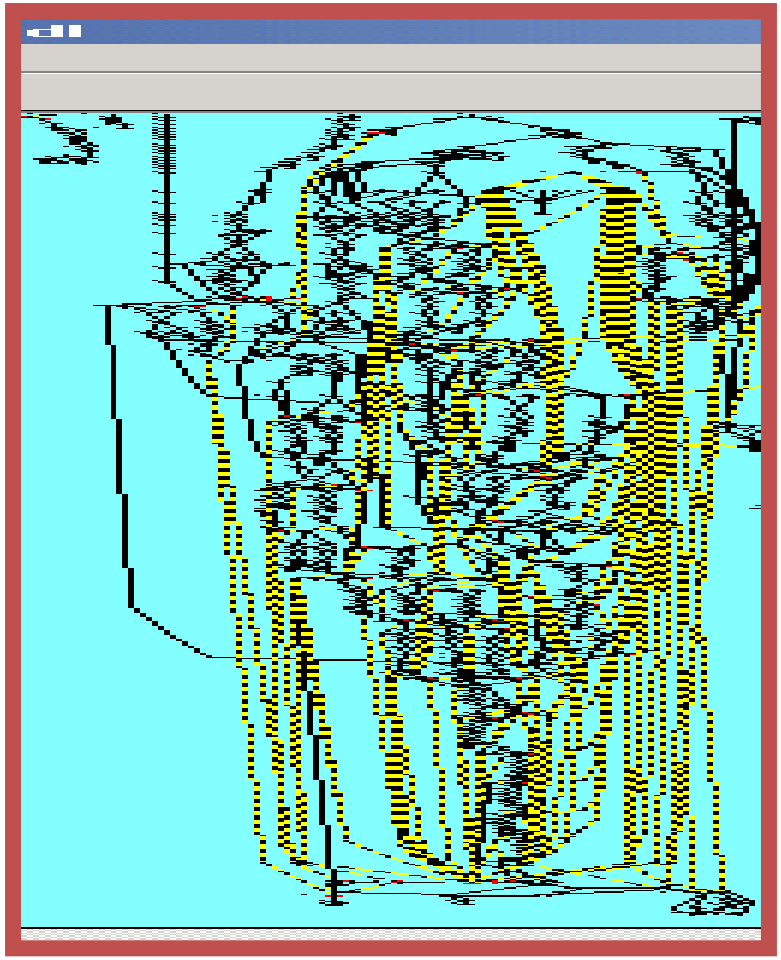
4．シナリオ関数の動性

シナリオ関数の構造（実行様相）

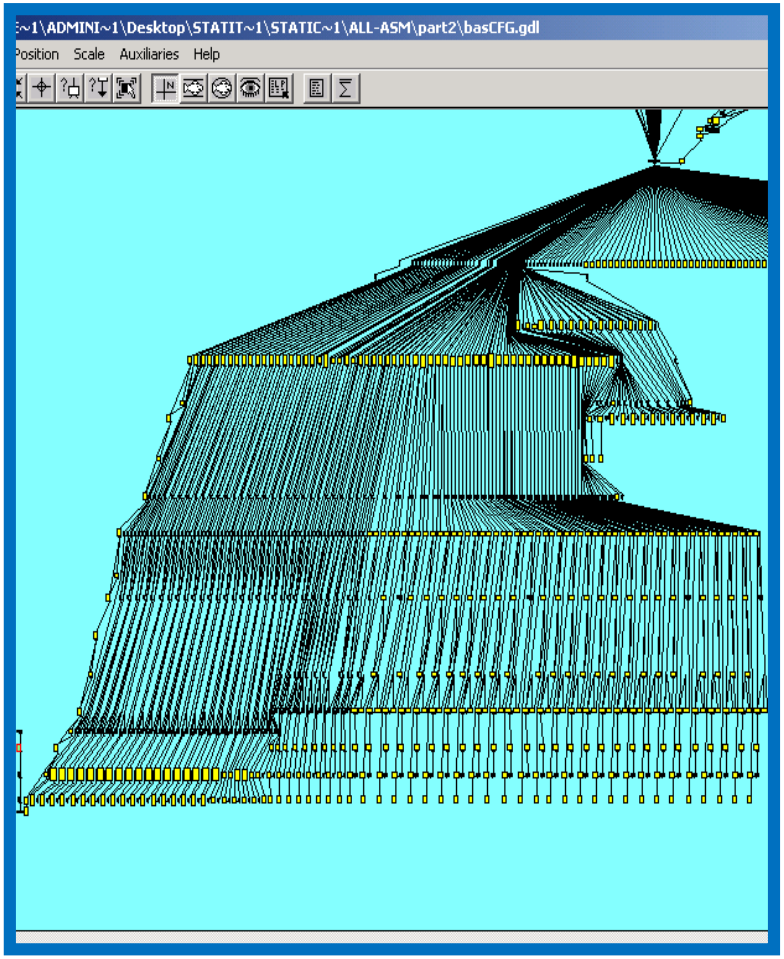


SW(従来プログラム）の問題構造と抜本的な方策：ラバール大学プログラム構造解析

論理結合型プログラム



データ結合型プログラム



５．シナリオ関数の特徴（1/2）

本研究では５００本以上の論理結合型プログラムの分析が行われている
そして、わかったことはわれわれの知性には宿命的な欠陥（名詞数の臨界問題）があり、
それをこえると論理結合の正統性は成立させられないという事である。

①同期アルゴリズムとは全名詞の主語を「同時」に成立させる仕組である。

②同期アルゴリズムで成立する全名詞の主語の様相は主語系譜と名付けられている。

③同期アルゴリズムを成立させる仕組みがシナリオ関数である。

④シナリオ関数は形而上学的に主語系譜を成立させるプログラムに帰着する。

⑤上記①の「同時」はシナリオ関数の全ベクトル達の第３規約で実現される。

⑥シナリオ関数のソースプログラムの構造は主語系譜を成立させるアルゴリズムから
導かれている。

⑦主語系譜はシナリオ関数の動性の解として位置づけられる。

⑧同期アルゴリズムはシナリオ関数の動性の完結様相（予定調和）として成立する。

⑨シナリオ関数はわれわれの思考法の欠陥を調和させるプログラムである。

⑩調和はプログラムに属す全名詞の全主語を同時に成立させる仕組で模型化されている。

⑪シナリオ関数では通常のデータ処理は主語系譜を成立させる過程で充足される。

５．シナリオ関数の特徴（2/2）

⑫プログラム問題はわれわれの思考法の欠陥により生じる現象である。

⑬プログラム問題は主語系譜の成立を阻害する様相である。

⑭非同期アルゴリズムで発症する問題は非同期アルゴリズムでは解法できない。

⑮非同期アルゴリズムで発症する問題は同期アルゴリズムでのみ解法される。

⑯シナリオ関数の同期アルゴリズムは閉じているので、プログラム問題は生じない。

⑰結果的に、シナリオ関数は主語系譜の成立を阻害する原因を自律的に捉える。

⑱結果的に、シナリオ関数は主語系譜の成立を阻害する原因を自律的に解法する。

⑲シナリオ関数は自律的にプログラム問題を解法するが、その為の仕組みが特別に
具備されているわけではない。結果的にその様になるという事である。

⑳シナリオ関数は単純な自動生成ツールで作成できる。
そして、従来プログラムのバグ原因は上記（２０）の自動生成ツールで機械的に発見できる。

6．プログラム革命の重要性

プログラム問題は部分解決ではなく、完全解決（解法）を図るために
プログラム問題を不可避的に内在させる論理結合型プログラムをシナリオ関数型
プログラムに置き換えることの提案

７．終わりに

　シナリオ関数は２０種規模の特許体系となり、日本、米国、英国、独、伊、露、中国での特許化が図られる予定です。日本では２０１６年、２０１７年に２件の基本特許権が査定されました。日本ではシナリオ関数の自動生成ツールの開発を「平成２８年度補正革新的ものづくりの支援補助金」に申し込み、認可されています。本年１１月には完成します。皆様にもご覧になって頂きたいです。今後は「囲碁の不敗（無敗ではない）のアルゴリズム」「ウイルスの作成法」の特許化を進めます。前者は、ＡＩとはシナリオ関数の思考法でなければシステム化が出来ないことを、後者はシナリオ関数でなければシステムの崩壊は回避できないことをおおくのひとに告げる為です。本研究は独りでとぼとぼと長い道のりを歩んできたのですが、古くからの友人、知人に助けられての成果です。関様はそのおひとりです。この世に貢献できれば、生きてきた証しとしてうれしいです。今はシナリオ関数で世界革命の潮時をじっと待つという心境です。本日は皆様にご清聴をいただきありがとうございました。

以上