日本大学大学院修士課程

総合基礎科学研究科地球情報数理科学専攻

平成１２年度 修士論文

属性グラフ文法による
プログラム図の構文解析

指導教員　夜久竹夫教授

6199M13　類瀬健二

２００１年２月

異なるCASEツール間でのプログラム図交換を目的としてDXL(a Diagram eXchange Language for tree-structured charts)が考案され，1995年にJIS X 0130[6]として日本工業標準規格として制定された．Hichart(HIerarchical flowCHART description language)は1978年に考案されたプログラム図言語でJIS X 0130の対象図の一つとして引用された．DXLに対応したHichartは2000年に属性edNCEグラフ文法により定式化された[8].

本論文では，[8]に対して等価な順位グラフ文法が存在することを確認し，Hichartに対応した属性edNCE順位グラフ文法を構成する．また，順位関係を用いた構文解析アルゴリズムの構築および計算量の評価を行う．

KeyWords
DXL,Hichart, 属性順位グラフ文法, プログラム図

目 次

1 はじめに

プログラムの図表示は，視覚的プログラミングツールやソフトウェア事例データベースにおけるプログラム情報の視覚化には不可欠な機能である．

近年，様々なプログラム図の記述言語が報告されており，それぞれに特徴を持っており，それらの言語に基づいた多くの CASE ツールが開発されている．なかでも，1978 年に夜久らにより提案された Hichart(HIerarchical flowCHART description language) は木フローチャートを基礎図式とする図形型言語であり，プログラムの階層構造，制御の流れおよびデータ構造を同時に表示できるという他のプログラム図言語とは異なる特徴も持っている．また現在まで，Hichart 処理システムの開発が行われてきている [2],[3],[4],[5].

一方，木構造図データ交換言語 DXL(Diagram eXchange Language for tree structured charts) は，多種多様木構造図に対する CASE ツール間でのデータを交換するための標準データ交換形式として設計され，JIS X 0130 規格に制定された．各プログラム図言語に対して DXL との間の変換ツールを作成することにより，異なるプログラム図言語の CASE ツール間でデータを相互に変換し，利用ができるようになる．

DXL に対応した Hichart プログラム図の生成規則の基底グラフ文法を Rozenberg の edNCE グラフ文法 [7] によるものとした属性グラフ文法 [2] に基づいて初歩的な描画規則とともに定式化されている [8]．この DXL 対応 Hichart 属性グラフ文法については，67 個のプロダクションとそれらに付随する意味規則からなる文法として報告されている．

基底グラフ文法をより普遍的な edNCE グラフ文法に定式化したことにより，表やシミュレータ等の他の処理系と統一的に扱える可能性がある [9].

そこで，本論文では HCGG に対して等価な順位グラフ文法があることを確認し，Hichart 対応順位グラフ文法を構成する．また，順位関係を用いた構文解析アルゴリズムの構築および計算量の評価を行う．

2 章では，edNCE グラフ文法，属性 edNCE グラフ文法，順位グラフ文法，Hichart，DXL，HCGG について解説する．3 章では，HCGG に対して等価な順位グラフ文法が存在することを示し，Hichart 対応順位グラフ文法 (HCPGG) を構成する．4 章では，順位関係を用いて HCPGG の構文解析を行なうためのアルゴリズムを構築し，そのアルゴリズムの計算量を評価する．また HCPGG の構文解析の例を示す．5 章では，実際に開発した HCPGG Parser について説明する．6 章では，現在までの結果及び今後の課題を記述する．

2 準備

この章では，Hichart のグラフ文法のために必要な概念について解説する．最初に 2.1 節ではグラフ文法に関する定義を述べる．2.2 節では属性 edNCE グラフ文法に関する定義を述べる．2.3 節では今回新たに導入する順位グラフ文法に関する定義を述べる．2.4,2.5 節では今回研究の対象とする Hichart および DXL について述べる．最後に 2.6 節では宮崎により提案された DXL 対応 Hichart のための属性グラフ文法について解説する．

2.1 edNCE グラフ文法 [7]

定義 2.1[7] Σ を 頂点のアルファベット，Γ を 辺のアルファベット とする．Σ と Γ 上のグラフとは 3 項組 $H = (V, E, \varphi)$ である．ただし，

1. V :頂点の有限集合
2. E :$\{(v, \gamma, w) \mid v, w \in V, v \neq w, \gamma \in \Gamma\}$ の部分集合で，辺のラベルのアルファベット
3. φ :頂点をラベル付けする関数

□

また H の構成要素もまた，ぞれぞれ V_H,E_H,φ_H で表す．

もし， $E_K = \{(f(v), \gamma, f(w)) | (v, \gamma, w) \in E_H\}$ とすべての $v \in V_H$ に対して，$\varphi_K(f(v)) = \varphi_H(v)$ であるような全単射 $f : V_H \to V_K$ が存在するとき，グラフ H と K は 同形(isomorphic) であるという．

Σ と Γ 上の全ての (具体的な) グラフの集合を $GR_{\Sigma,\Gamma}$ で表す．そして，全ての抽象的なグラフの集合を $[GRE_{\Sigma,\Gamma}]$ で表す．$[GR_{\Sigma,\Gamma}]$ の部分集合を グラフ言語(graph language) と呼ぶ．

edNCE 文法の生成規則は，以下のように表す．

定義 2.2[7] $X_0 \to$(D,C) は edNCE 文法の生成規則である．

1. X_0:非終端な頂点アルファベット
2. D:グラフ
3. $C \subset \Sigma \times \Gamma \times \Gamma \times V_D \times$ {in,out}:
   接続関係(connection instructions)

□

生成規則によって行なわれる置き換えステップは，X_0 とラベル付けされた頂点 (”母頂点”(the ”mother node”)) を与えられた”主 (host)”グラフ H から取り去り，その場所に D(”娘グラフ”(the ”daughter graph”)) を代入し，そして C 中の接続命令により指定された方法で H の残りと D をつなぐ事で成り立つ．ペア (D, C) はオブジェクトの新しいタイプとしてみなすことが出来る．そして，置き換えステップは，主グラフ内の母頂点に対するこのオブジェクト (D, C) の置換としてみなすことが出来る．

形式的に，Σ と Γ 上の (近傍を制御した)埋め込みを持つグラフ(a graph with (neighbourhood controlled) embedding) とは，ペア (H, C) であり，$H \in GR_{\Sigma,\Gamma}$, そして $C \subset \Sigma \times \Gamma \times \Gamma \times V_H \times \{in, out\}$．$C$ は (H, C) の 接続関係(connection relation) であり，$\delta \in \Sigma$, $\beta, \gamma \in \Gamma$, $x \in V_H$, $d \in \{in, out\}$ である．C のそれぞれの要素 $(\delta, \beta, \gamma, x, d)$ は (H, C) の 接続命令(connection intstruction) であり，接続命令を $(\delta, \beta/\gamma, x, d)$ と書く．

$C_K = \{(\delta, \beta/\gamma, f(x), d) | (\delta, \beta/\gamma, x, d) \in C_H\}$ であるような，H から K への同形写像が存在するならば，埋め込みを持つ 2 つのグラフ (H, C_H) と (K, C_K) は同形である．

Σ と Γ 上の全てのグラフの集合は，$GR_{\Sigma,\Gamma}$ で表す．あらゆる通常のグラフもまた空の埋め込みを持つグラフとしてみなせる (すなわち，$C \neq 0$)．したがって，$GR_{\Sigma,\Gamma} \subset GR_{\Sigma,\Gamma}$ である．’埋め込みを持つグラフ’の

代わりに’グラフ’という．直感的に，埋め込みを持つグラフ (D, C) に対して，C の接続命令 $(\delta, \beta/\gamma, x, out)$ は次の事を意味する．もし，(D, C) を置きかえるための母頂点 v から，ラベル δ をもつ頂点 w への β とラベル付けされた辺が存在するならば，その時は埋め込みプロセスは，x から w への γ とラベル付けされた辺で確立する．

また edNCE グラフ文法は次のように構成する．

定義 2.3[7] edNCE グラフ文法 は次を満たす 6 項組 $G = (\Sigma, \Delta, \Gamma, \Omega, R, S)$ である．

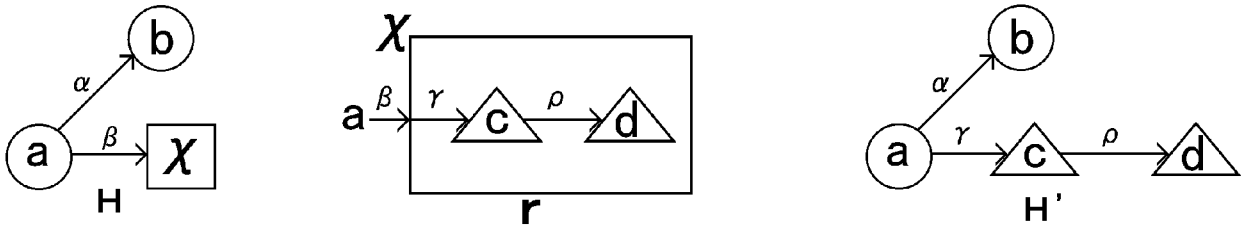
1. Σ : 頂点アルファベット
2. Δ : 終端頂点アルファベット
3. Γ : 辺のアルファベット
4. $\Omega \subseteq \Delta$: 終端辺アルファベット
5. R : 生成規則の有限集合
6. $S \in \Sigma - \Delta$: 初期状態

また $X_0 \in \Sigma$ - Δ，(D,C) $\in \mathrm{GRE}_{\Sigma,\Gamma}$ において生成規則 R を $X_0 \to$ (D,C) と記する．

□

グラフ H に生成規則 r を適応して H’ が得られるとき，H $\Rightarrow$ H’ と記述する．また$\Rightarrow$の推移的閉包を $\overset{*}{\Rightarrow}$ と記述する．

図 1 に生成規則と導出の例を示す．ホストグラフ (a) に生成規則 (b) を適応して得られた結果グラフが (c) である．



(a) ホストグラフ　　(b) 生成規則　　(c) 結果グラフ

図 1: 生成規則と導出の例

2.2 属性 edNCE グラフ文法 [8]

属性 edNCE グラフ文法(attribute edNCE graph grammar) は属性を各節頂点に付加し，属性の値を評価するための意味規則を文脈自由な edNCE グラフ文法の各生成規則に付加することで属性グラフ文法がつくられる．

定義 2.4[8] 属性 edNCE グラフ文法 は次の条件を満たす 3 項組 $G_N = \langle GG, A, F \rangle$ である．

1. $GG = (\Sigma, \Delta, \Gamma, \Omega, R, S)$ は edNCE グラフ文法で，G_N の 基底 edNCE グラフ文法 と呼ばれる．また，Σ, Δ 上のグラフ $D = (V, E, \varphi)$ について, $Lab(D) = \{\varphi(n) \mid n \in V\}$ とする．
2. GG の $\Sigma \cup \Delta$ の各元 X に対して，互いに素な 2 つの有限集合，継承属性 の集合 $\mathcal{I}(X)$ と 合成属性 の集合 $\mathcal{S}$(X) が付随している．X の属性全体の集合を $A(X) = \mathcal{I}(X) \cup \mathcal{S}(X)$ で表す．$A = \bigcup_{X \in \Sigma} \mathrm{A(X)}$ を G_N の 属性集合 という．また，X の属性 a を $a(X)$ で表し，a がとりうる値全体の集合を $\mathcal{V}(a)$ で表す．
3. R の各生成規則 $r = X_0 \to (D, C)$ に対し，$\mathcal{S}(X_0) \cup \bigcup_{x \in Lab(D)} \mathcal{I}(X)$ の属性のみをすべて定義する 意味規則 の集合 F_r が付随している．$0 \leq i_j \leq |Lab(D)|, X_{i_j} \in Lab(D), 0 \leq j \leq m$ という形をしている．ただし，f は $\mathcal{V}(a_1(X_{i_1})) \times \cdots \times \mathcal{V}(a_m(X_{i_m}))$ から $\mathcal{V}(a_0(X_{i_0}))$ の中への写像である．このとき，$a_0(X_{i_0})$ は r において $a_j(X_{i_j})(1 \leq j \leq m)$ に依存するという．集合 $F = \bigcup_{r \in R} F_r$ を G_N の 意味規則集合 という．

□

2.3 順位グラフ文法 [10]

2.3.1 順位グラフ文法

順位グラフ文法(precedence graph grammar) とは，グラフ文法が合流，相称的，唯一可逆，再起の非終端がなく，かつ順位対立を持たないことである．

また順位グラフ文法は主な性質として，

1) 順位グラフ文法は近隣の保存と曖昧でない事

2) 曖昧でないことは，属性グラフ文法に対してとりわけ重要であり導出木の属性の意味評価が容易な事をもつ

定義 2.5[10] 二つの頂点 (v,w ∈ V) の ラベル組 とは lab_G(v,w) $=^{def}$ (φ(v),(v,γ,w),(w,γ,v),φ(w)) である．

□

定義 2.6[10] 導出仕様書(derivation specification) とは，s=(P,$\tilde{X}_0$,$\tilde{D}$,$\tilde{b}$) であり，$\tilde{X}_0 \cong X_0$,$\tilde{D} \cong$D,$\tilde{b}$:$V_{\tilde{D}} \to V_D$ である．また 導出列(derivation sequence) とは d=($G_{i-1} \to_{S_i} G_i$|1≤i≤n) である．

□

定義 2.7[10] $S_i \leq_D S_j$ でも $S_j \leq_D S_i$ でもない場合 S_i，S_j は 比較できない といい，$*$ と書く．ただし $\leq_D$ は推移的閉方，半順序で D の導出順序と呼ばれる．

□

定義 2.8[10] v∈$V_{\tilde{R}_i}$ ならば S_D(v) $\stackrel{\text{def}}{=}$ S_i である．

□

定義 2.9[10] v⋈w⇔S_D(v)⋈$_D$$S_D$(w) である．ただし v,w∈$V_{G_n}$,⋈∈{≒,<,>} である．

□

定義 2.10[10] ⋈ ∈{≒,<,>,$*$} は 2 頂点間の順位関係 である．その ラベル間の順位関係 は $R_{\bowtie}$ であり，⋈ はすべての lab_G(v,w) の集合である．

□

定義 2.11[10] 拡張辺ラベルアルファベット(extended edge label alphabet) とは，$\Gamma_{\bowtie}$=^{def} $\Gamma \times$ {≒,<,>,×,·} であり，⋈ は，次の通りに Γ 上でグラフ $G_{\bowtie}$ に $\Gamma_{\bowtie}$ 上のすべてのグラフ G をつくる．

また，$V_{\bowtie}$=V,$\varphi_{\bowtie}$=φ,$E_{\bowtie}$={(v,(x,υ),w)|v,w∈V,x∈Γ,(v,x,w)∈E,υ∈{≒,<,>,$*$}} である．

□

定義 2.12[10] 順位グラフ文法は次を満たす 6 項組 $GG_{\bowtie} = (\Sigma, \Delta, \Gamma_{\bowtie}, \Omega_{\bowtie}, R, S)$ である．

1. Σ : 頂点アルファベット
2. Δ : 終端頂点アルファベット
3. $\Gamma_{\bowtie}$: 拡張辺ラベルアルファベット
4. $\Omega_{\bowtie}$: 拡張辺ラベルアルファベット (終端辺ラベルアルファベット)
5. R : 生成規則の有限集合
6. $S \in \Sigma - \Delta$: 初期状態

□

定義 2.13[10] Instantaneous Description(ID) とは
(G,K,Ψ) の 3 項組である.

1. G:インスタンスグラフ
2. K:G 内の頂点の順序リスト
3. Ψ:そこまでの 導出仕様書 の構成の集合

□

頂点間の順位関係は主に 3 通りあげられる．ひとつ目に頂点間の順位が等しい順位関係，ふたつ目に頂点間の順位が自分よりも相手の頂点のほうが高い場合，そしてみっつ目に頂点間の順位が比較できない場合がある．この 3 通りの関係について図 2 によって表わすことができる．

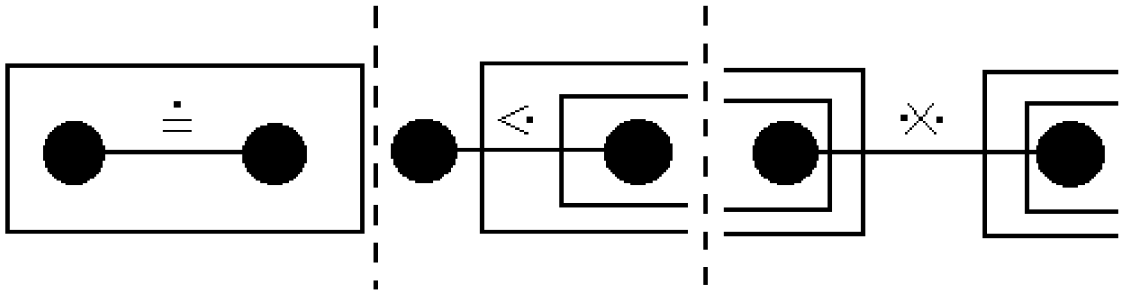


図 2: 順位関係

n $\in \aleph$, d' = ($G_{i-1} \rightarrow_{s_i} G_i$ — 1 $\leq$ i $\leq$ n) は導出列の長さ n で， $s_i = (p_i, \widetilde{X}_i, \widetilde{D}_i, \widetilde{b}_i)$, 1 $\leq$ i $\leq n$ を示す．もし，ある $\widetilde{D}_i$ が d' ならば，完全部分グラフ H$\subset G_n$ は ハンドル(handle) である．もちろん，その時は構文解析 G_n は上昇型であり，ハンドルは最初に還元しなければならない．一方 (反対に)，ある導出可能なグラフ G を示す．もし，すべての隣接する v,w $\in V_H$ に対して lab_G(v,w) $\in$ $R_{\doteq}$ であり，かつ， R_G(H) からなる v' に隣接するすべての頂点 lab_G(v',w') $\notin$ $R_{\doteq} \cup R_{>}$, w' $\in$ V_H である時，完全部分グラフ H$\subset$G は G 内の 順位ハンドル(precedence handle) と呼ばれる．また，H$\subset$G は，H は G の 完全部分グラフ(full subgraph) であることを表わし，G|V は，H$\subset$G の制限 (V_H=V) を表わす．また $R_G(H)$ は，G$\subset H$ の完全部分グラフ H' を表わす.

完全部分グラフ H$\subset$G は以下の場合，順位ハンドル(precedence handle) をとる.
lab'$_G$(v,w) において

1. lab_G(v,w) $\in$ $R_{\doteq}$ (v,w $\in$ $V_{H'}$)
2. lab_G(v',w') $\notin$ $R_{\doteq} \cup R_{>}$ (v' $\in$ R_G(H'),w' $\in$ $V_{H'}$)

である．ただし，$V_{H'}$ とは，H' に存在する頂点を指す.

順位グラフ文法の構文解析は次の通りに行われる．

1. 任意の頂点でスタートする．
2. できる限り等しい，または上っている順位に沿って進んでいく．シフトはスタック内の頂点を移動または新たに頂点を追加する．
3. それ以上，上がることができないなら，順位ハンドルが決定する．また順位ハンドルは生成規則の rhs と同相である．グラフ内の順位ハンドルに対して生成規則を逆適応して lhs に還元させ，そして 2. にすすむ．

順位グラフ文法の構文解析は，上の手順を踏んで ID を書き換えて行なっていく．

最初の ID(G_0,K_0,ϕ) について G_0 は入力グラフ，K_0 は G_0 のある頂点である．(G,K,Ψ) はある ID で，K=< v_1,$\cdots$,v_k >,k$\geq$1 で示す．j は最小の指数 $1 \leq$ j $\leq$ k であり，等しい順位に沿った v_j から v_k までからなる G 内のいくつかのパスである．その時，TOP(G,K) を以下のように定義する．また TOP は シフト(shift)，還元(reduce) と呼ばれる二つの型の 動作(moves) からなる．

定義 2.14[10] TOP(G,K) とは

$$TOP(G,K) \underset{def}{=} G|\{v_j,\cdots,v_k\}(1 \leq j \leq k, v_j\text{から } v_k\text{までは等しい順位})$$

で定義される．また TOP(G,K) は

シフト:(G,K,Ψ) $\longmapsto_S$ (G,Kw,Ψ) は以下の場合である．

1. w $\in$ V が K 内でまだ発生しないならば
2. TOP(G,K) 内のいくつかの v に対して lab_G(v,w) $\in R_{\doteq} \cup R_{\lessdot}$

還元:(G,K_1K_2,Ψ) $\longmapsto_R$ (G,K_1w,$\Psi \cup$ { s }) は以下の場合である．

1. w が G または Ψ 内でまだ使われていない頂点ならば
2. d=(p,$\widetilde{X}$,$\widetilde{D}$,$\widetilde{b}$),G'$\rightarrow_d$G, d$\notin\Psi$，K_2=$V_{\widetilde{D}}$，G|K_2=TOP(G,K)(順位ハンドル),

G'|{w}=$\widetilde{X}$，K_1K_2 は K_1 と K_2 両方の列の結びつきを示す．

と呼ぶ 2 通りの型の動作からなる．

□

またシフトと還元の例を以下に示す.

現在の ID を (G_1,n_3,ϕ) とする．これは，現在のグラフが G_1，現在の順位ハンドルが n_3 である事を示している．図 3 はシフトの例を表わしており，左の図に対する ID は現在のものより (G_1,n_3,ϕ) である．図 3 の右の図において順位ハンドルは n_3 であるが，順位ハンドルを調べる操作をすることにより，n_3 と n_2 の順位が等しいことがわかるので順位ハンドルが図 3 の右の図のように変化する．よって K が <n_2,n_3> となり，ID は (G_1,<n_2,n_3>,ϕ) となる．また還元は，図 4 の左の図に対する ID は (G_1,<n_2,n_3>,ϕ) であるが，

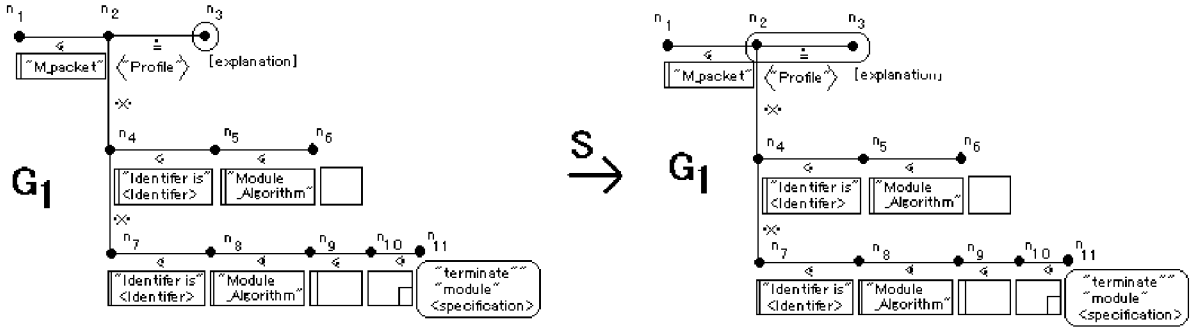


図 3: parsing(handle)

還元を行なうことにより，G_1 は，生成規則 P_4 の逆適応により図 4 の右の図 G_2 に置き換えられ，これに対する ID は (G_2,<n_2'>,P_4) となる.

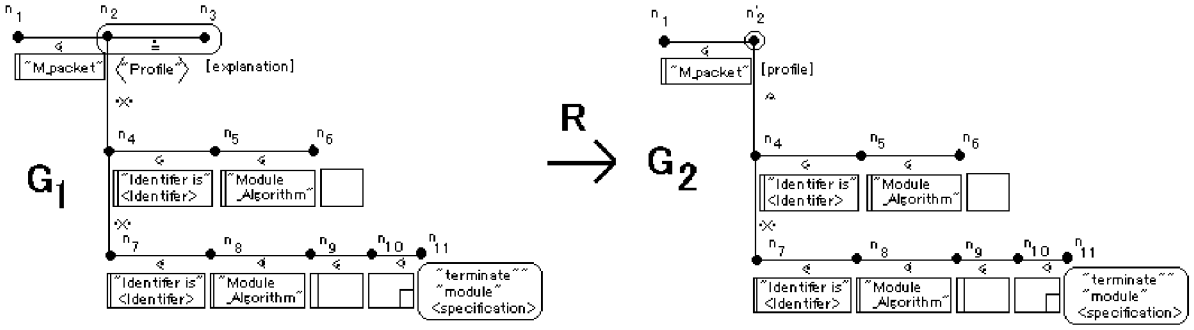


図 4: parsing(reduce)

2.4 Hichart[1]

プログラム図の起源は Goldstein と von Neumann(1947) により導入された流れ図にある．以来，この Goldstein − Neumann 型流れ図の利点を残しながら，数多くの構造型プログラム図が提唱され，構造化の方向に発展してきている．

Hichart は 1978 年に夜久，二木により発表された階層型流れ図言語である．Hichart は繰り返し記号 を初めて導入したプログラム図式言語で，プログラムを作成している要素間の制御の流れと階層的なつながりを疑似木構造グラフとして表現する．Hichart はプログラム自身の階層木構造を直接反映しているので，Hichart によってプログラムの全体構造が即座に把握可能となり，また可読性や視認性が飛躍的に向上する．その後改良と拡張 [3] が続けられ，現在ではプログラムの仕様を形作る 4 つの基本要素，(1) アルゴリズムの流れ，(2) データの流れ，(3) データ構造，(4) プログラム構造をすべて統一的に表示することができる．また Hichart に用いられる記号を Hichart 記号または「セル」と呼ぶ．

図 5 はプログラム「ハノイの塔」を Pascal 対応 Hichart を用いて表示したものである．

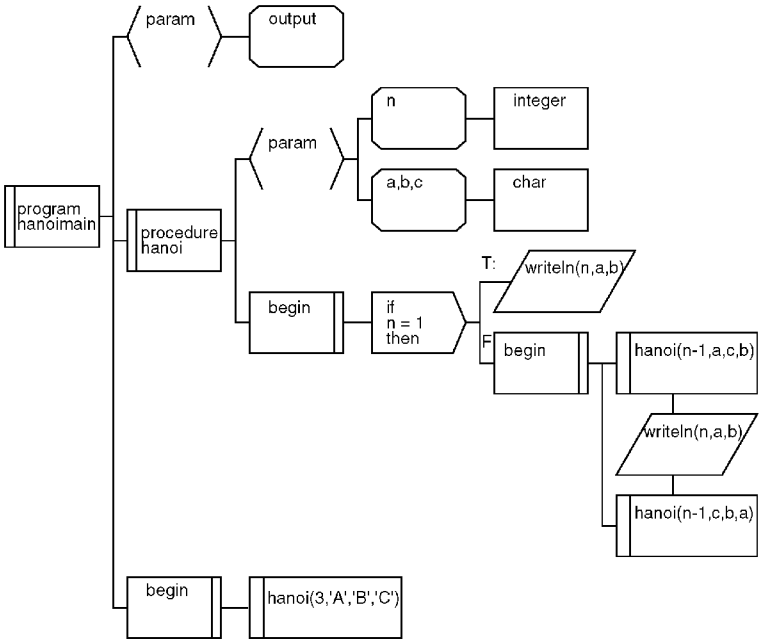


図 5: Hichart (Tower of hanoi)

2.5 DXL[6]

現在，ソフトウェアの仕様化や設計においては多様な図が用いられており，近年はそれらに基づいた多くの CASE ツールが開発されている．しかし，図の表記法の十分な標準化がなされておらず，これが CASE ツールで作成した図のデータ形式を標準化し，それを交換することによって，多様な図の表記法を知らなくても，使用及び設計情報の再利用や流通ができるようにしようという動きが活発化してきた．

国内ではプログラムの論理を表現する図としてさまざまな木構造図が普及し，広くソフトウェア開発に使われている．プログラム構成要素は，JIS X 0128(プログラム構成要素及びその表記法) で規格化されているものの，この要素を図で表現する木構造図自体の規格化はなされていない．このため，複数社による大規模ソフトウェアの開発などでは，各社が異なる木構造図を使用していることにより，(1) 納入するドキュメントの表記法を統一するため，ある一社の木構造図を開発に用いる．これによって，他社の開発者はその木構造図を学ぶことが必要になったり，自社の CASE ツールが使えないなどという問題が起こる．(2) ドキュメントの表記法と統一せずに，各社ごとに別々の表記法で記述する．これによって，開発途中での相互レビューや開発完了後の保守時に開発者や保守者が複数の木構造図を学ぶことが必要になる．(3) 最近の木構造図は，CASE ツールを用いて作成されることが多いが，現状では異なる表記法の木構造図間の相互変換が困難であり，表記法が違うと再利用ができない．という弊害が起きている．

しかし，すでにこれらの木構造図の作成や編集を支援する CASE ツール (以下，木構造図ツールと呼ぶ) が普及し，すでに作成されたデータが大量に蓄積されているため，特定の木構造図を規格とすることは非現実的である．そこで，データの再利用や流通を促進するために，木構造図用のデータ交換言語 DXL(Diagram eXchange Language for tree-structured charts) を定め，CASE ツールで作成した図のデータ形式を標準化してデータの再利用や流通を促進するために設計され，JIS X 0130 として制定された．

この目的のために，DXL の設計方針として以下の事柄が考慮されている．

(1) 木構造図ツールで作成したデータから DXL への変換ツールの作成を容易にするため，構文は計算機処理が容易なように設計し，かつ，人間が読んだり書いたりすることも配慮した．

(2) 木構造図との対応が容易なように，手続き的な記述をする構文とした．

(3) DXL の構文要素に対応する木構造図の構成要素の選定にあたっては，モジュール論理の設計品質の向上に有効な構成要素を重視した．

(4) 将来の木構造図の改善や木構造図ツール特有の個別情報 (データ設計情報，ソースコード等) の付加などが可能なように，拡張性のある構文とした．

(5) 関連する国際的な標準 (P1175，CDIF 等) との共存を考慮した．

2.6 HCGG[8]

DXL 対応 Hichart グラフ文法は属性 edNCE グラフ文法に基づき,67 個の生成規則と各生成規則に付随する意味規則により定義されている．この文法は DXL の構文すべてに対応している．生成規則の例を図 6 に示す.

定義 2.11[8] DXL 対応 Hichart に対する属性 edNCE グラフ文法を HCGG と呼ぶ.

□

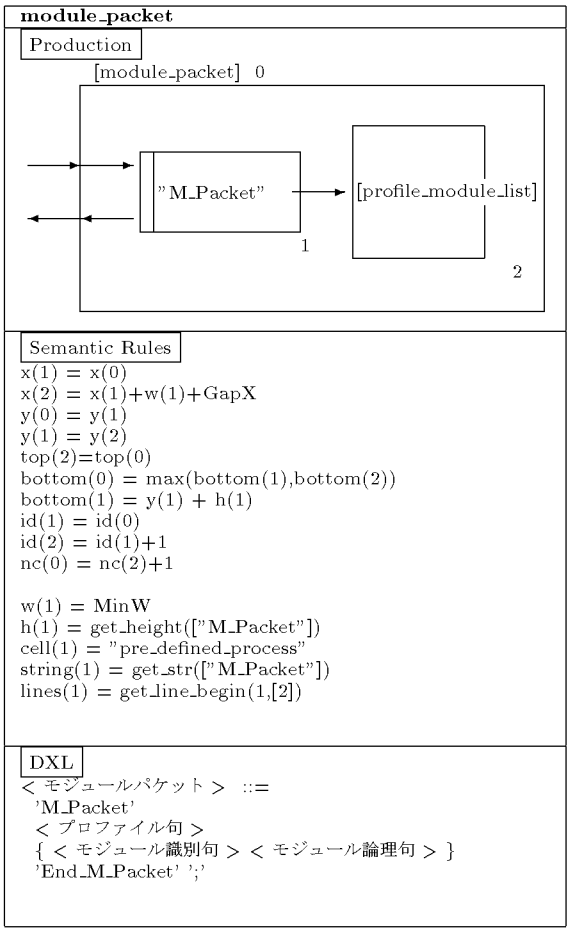


図 6: 生成規則の例 (edNCE 文法)

3 HCGG に対する順位グラフ文法

この章では，DXL 対応 Hichart のための順位グラフ文法を定式化する．

3.1 HCPGG

我々は HCGG に対して，順位関係に矛盾が生じないように生成規則を修正し，Hichart 対応順位グラフ文法(HiChart Precedence Graph Grammar) を構成し，この文法が HCGG によって作られるプログラム図と等価であることを確認する．

定義 3.1 HCPGG は 3 項組 $\langle GG_{\bowtie}, A, F\rangle$ によって構成され，生成規則 69 個，意味規則 728 個からなり，スタートグラフは”[module_packet]” とする．

□

定理 3.2 L(HCGG) を生成する順位グラフ文法が存在する．

証明 HCPGG を考えると HCPGG の頂点アルファベット間において，順位関係を矛盾なく定義できる．その方法及び結果を以下によって示す．

手順 1

HCPGG のすべての生成規則と定義 2.8 を用いてハッセ図を作成する．まず初めに HCPGG の開始記号である”[module_packet]”に対して適用可能な生成規則を探すと生成規則 1 が発見できる．この生成規則を適応すると，終端頂点アルファベット”M_Packet” と非終端頂点アルファベット”[profile_module_list]”を新たに生成する．そしてこの新たに生成された頂点アルファベットに対して，生成規則を何回適応した時に生成された頂点かを示す番号を頂点アルファベットに割り当てていく．次に”[profile_module_list]”に対して上と同じ手順を踏んでいき，すべての計算を行なっていく．そしてこの計算結果を用いて HCPGG のための木構造のハッセ図を作成する．図 7 は HCPGG に対して上の計算を行ない作成したハッセ図の一部であり，この図の各節はグラフを表わしている．

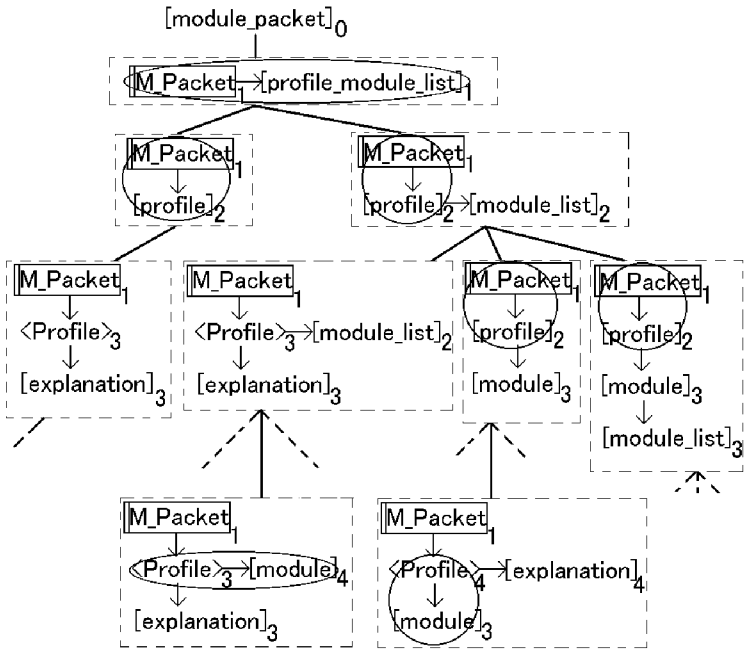


図 7: Hasse-diagram(HCPGG 用)

手順 2

次に上のハッセ図，定義 2.5,2.7,2.9,2.10 を用いて，頂点アルファベット間の順位関係を求め，その順位関係表を作成する．lab_G(v,w)=(”M_Packet”,♯,ϕ,”[profile_module_list]”) の順位関係を求めるときは，ハッセ図内の各節において，そのような関係になっているグラフを発見する．この 2 頂点間の関係は手順 1 で割り振った番号を見るとすべて同じ数値である事がわかるので，(”M_Packet”,♯,ϕ,”[profile_module_list]”)$\in R_{\doteq}$ となる．また (”Profile”,♯,ϕ,”[module]”) の順位関係は図 7 の一番下の二つの楕円に囲まれた所に注目すると，”Profile”と”[module]”のどちらの方が順位が高いのかを求める事ができないので，定義 2.7 より比較できないということになり，(”Profile”,♯,ϕ,”[module]”)$\in R_{\times}$ となる．上のようにハッセ図内のすべての頂点アルファベット間の順位関係を求め，矛盾がない事を確認した．また HCPGG のすべての生成規則と順位関係表を付録として載せている．

よって L(HCGG) となる順位グラフ文法 HCPGG が存在することがわかる．

□

定義 3.3 表 1 は，HCPGG の順位関係表の一部であり，606 個の順位関係から構成され，2 頂点アルファベット間はグラフにおいて直接連結される．

□

	[module_list]	”Profile”	[explanation]	[module]	”Identifier is”
[module_profile]					
”M_Packet”		⋖			
[profile_module_list]					
[profile]	≐			⋖	⋖
[module_list]					
”Profile”	⋗		≐	⋇	⋇
[explanation]					
[module]	≐			⋖	⋖
”Identifier_is”	⋗		⋖	⋇	⋇
[explanation_module_algorithm]					
[module_algorithm]					
<explanation>					
”Module_Algorithm”					

表 1 HCPGG の順位関係表

4 HCPGGに対する構文解析

4.1 HCPGGに対する構文解析アルゴリズム

HCPGGの構文解析アルゴリズムを，構文解析を行なう"HCPGG_analysis"と属性評価を行なう"attribute_evaluation"の2部から構成した．

4.1.1 HCGPP_analysis

"HCPGG_analysis"では入力グラフを与え，構文木を作成する．

Algorithm 1

```
HCPGG_analysis(input graph){
  while(G != start graph ){
    precedence_analysis(graph);
  }
  if(入力グラフが構文解析可能)
  parse tree = generate_parse_tree(derivation sequence);
}
```

アルゴリズム"HCPGG_analysis"では入力したグラフがスタートグラフになるまで"precedence_analysis"を繰り返し，"generate_parse_tree"で構文木を作成する．また，"generate_parse_tree"は492行で構成する．

アルゴリズム"precedence_analysis"では，シフトと還元を行ない，入力したグラフに対して，このグラフのハンドルに対する生成規則の左辺に置きかえる．

Algorithm 2

```
Graph precedence_analysis(graph){
  K = shift_order_list(graph)
  production = find_production(K);
  graph = replace_graph(graph,K,production);
  K = rewriting_order_list(K,production);
  return(graph)
}
```

アルゴリズム"precedence_analysis"は4ステップからなる．最初のステップは，"shift_order_list"(315行)で，入力したグラフに対して順序ハンドルの探索を行ない，頂点の順序リストを作成する．第2のステップでは，その順序リストから，順位ハンドルを右辺とする生成規則を"find_production"(218行)によって発見する．第3のステップでは，グラフの置き換え(生成規則の逆適応)の作業を"replace_graph"(2774行)で行なう．第4のステップでは，順序リストKから，生成規則の右辺に相当する頂点を取り除き，左辺に相当する頂点を付け加える作業を"rewriting_order_list"(250行)が行なう．

アルゴリズムを記述するために，Hichart のセル，順序リストそして生成規則に関して以下のような写像や用語を用意する．

まず v∈V の時，label(v):V→Σ を頂点 v の頂点アルファベットへの写像とする．$^{\exists}$function∈{parent,child1,child2,previous,next}:V → V とし，それぞれ頂点 v の親の頂点，はじめの子供の頂点，2 番目の子供の頂点，前の頂点，次の頂点を求める関数である．$function_1$∈{child1,child2,previous,next}，$^{\exists}function_2$∈{parent,child2,previous,next}，$^{\exists}function_3$∈{parent,previous,next}，$^{\exists}function_4$∈{parent,child1,previous,next}，$^{\exists}function_5$∈{parent,child1,child2,previous}，$^{\exists}function_6$∈{parent,child1,child2,next}，$^{\exists}function_7$∈{child1,child2,previous,next} とし，これらは function の部分集合からなる関数である．

次に，順序リストを K[j](j:整数) と表わす事にする．順序リストの最後尾を示す番号を Z とし K[Z] で表わす．ただし K[Z]≠ϕ，K[Z+1]=ϕ である．node(K[Z]) を V→V への写像とし K[Z] の頂点，number(K[Z]) を V→N(整数) への写像とし順位ハンドル内の頂点の個数を求める関数とし，また H を順位ハンドルを表わすこととする．また K に頂点を加える条件は K 内に対して加える頂点が存在しない時とし，K の最後尾に頂点を追加するものとする．

最後に生成規則に関して，i を生成規則の rhs 内の頂点の個数を表わす整数とする．p∈P を生成規則，P^i を生成規則の rhs の頂点の数が i 個である生成規則の集合 (P^1∪P^2∪P^3=P) とする．rhs_number(p) を P→N への写像とし，生成規則の rhs の頂点の個数を求める関数，rhs_nonterminal(p) を V→N への写像とし，生成規則の rhs 内の非終端頂点アルファベットの数を求める関数，rhs_id(Σ) を Σ→N への写像とし，生成規則の rhs 内の頂点アルファベット Σ の右下の番号を求める関数，rhs(p) を P→G への写像とし，生成規則の rhs(グラフ) を求める関数，lhs(p) を P→Σ への写像とし，生成規則の lhs の頂点アルファベットを求める関数とする．

これらの関数や用語を用いて，shift_order_list,find_production,replace_graph,rewriting_order_list,generate_parse_tree に関するアルゴリズムを以下のように設計する．

Algorithm2_1*order list* shift_order_list(*graph, order list K*){
  K'=ϕ;
  while(K'=K){
    K'=K;
    if(K_2 has 3 nodes, v, w_1, w_2){
      if(v,w_1=child1(v),w_2=child2(v),lab_G(v,w_1)$\in R_{\doteq}$ and lab_G(v,w_2)$\in R_{\doteq}$){
        if($lab_G(w_1,function_1(w_1)\in R_<$)) K に $^{\exists}function_1(w_1)$ を加える;
        else if($lab_G(w_2,function_1(w_2)\in R_<$)) K に $^{\exists}function_1(w_2)$ を加える;
        else if($lab_G(w_1,function_1(w_1)\in R_{\doteq}$)) Error;
        else if($lab_G(w_2,function_1(w_2)\in R_{\doteq}$)) Error;
      }
      else Error;
    }
    else if(K_2 has 2 nodes(v,w),lab_G(v,w)$\in R_{\doteq}$){
      if(w=parent(v) and v=child1(w)){
        if(lab_G(w,$^{\exists}function_2$(w))$\in R_<$) K に $^{\exists}function_2$(w) を加える;
        else if(lab_G(w,child2(w))$\in R_{\doteq}$) K に child2(w) を加える;
        else if(lab_G(w,$^{\exists}function_3$(w))$\in R_{\doteq}$) Error;
      }
      else if(w=parent(v) and v=child2(w)){
        if(lab_G(w,$^{\exists}function_4$(w))$\in R_<$) K に $^{\exists}function_4$(w) を加える;
        else if(lab_G(w,child2(w))$\in R_{\doteq}$) K に child2(w) を加える;
        else if(lab_G(w,$^{\exists}function_3$(w))$\in R_{\doteq}$) Error;
      }
      else if(w=previous(v)){
        if(lab_G(w,$^{\exists}function_5$(w)$\in R_<$)) K に $^{\exists}function_5$(w) を加える;
        else if(lab_G(w,$^{\exists}function_5$(w))$\in R_{\doteq}$) Error;
      }
      else if(w=next(v)){
        if(lab_G(w,$^{\exists}function_6$(w))$\in R_<$)) K に $^{\exists}function_6$(w) を加える;
        else if(lab_G(w,$^{\exists}function_6$(w))$\in R_{\doteq}$) Error;
      }
      else if(w=child1(v)){
        if(lab_G(w,$^{\exists}function_7$(w))$\in R_<$)) K に $^{\exists}function_7$(w) を加える;
    else if(w=next(v),lab_G(w,$^{\exists}function_7$(w))$\in R_{\doteq}$)) Error;
      }
    }
    else if(K_2 have a node){
      if(lab_G(v,$^{\exists}$function(v))$\in R_<$) K に $^{\exists}$function(v) を加える;
      else if(lab_G(v,$^{\exists}$function(v))$\in R_{\doteq}$) K に $^{\exists}$function(v) を加える;
    }
    else Error;
  }
}

Algorithm2_2*production* find_production(*order list* K){
if(K_2=1]) *production* = { $^{\exists}$p | p $\in$ P^1, p=H });
else if(K_2=2]) *production* = { $^{\exists}$p | p $\in$ P^2, p=H });
else if(number[K_2=3]) *production* = { $^{\exists}$p | p $\in$ P^3, p=H });
else *production* = Error;
}

Algorithm2_3*graph* replace_graph(*graph*, *production*){
for(j=1;j<i | P^i,++j) $^{\exists}$function = ϕ;
label(node[K[Z-j+1]))=lhs(p);
v = node(K[Z-j+1]);
if(v,w $\in$ V_H) $^{\exists}$function = ϕ;
/* w は v に隣接する頂点 */
else $^{\exists}$function = w;
}

Algorithm2_4*order list* rewriting_order_list(*order list* K,*production*){
for(j=1;j¡number(K[Z];++j)) K[Z-j]=ϕ;
number[Z-j+1] = 1;
}

Algorithm2_5*parse tree* generate_parse_tree(*production sequence* p_i){
while($p_i \neq \phi$){
if(rhs_nonterminal(p) = 2){
parse tree = parse tree and p_i;
if(lhs(p_{i+1}) = rhs_id($^{\exists}sigma$)) generate_parse_tree(production sequence p_{i+1});
/* $^{\exists}sigma$ は生成規則の rhs 内の頂点アルファベットを示す */
else if(rhs_nonterminal(p) = 1){
parse tree = parse tree and p_i;
generate_parse_tree(production sequence p_{i+1});
}
else parse tree = parse tree and p_i;
return(production sequence p_{i+1});
}
}

4.1.2 attribute_evaluation

”attribute_evaluation”では，構文木に対して属性評価を行なう．

Algorithm 3

attribute_evaluation(parse tree){
S = attribute_parse_tree(parse tree);
attribute_calcurate(S);
}

アルゴリズム”attribute_evaluation”では，”attribute_parse_tree”(115 行) において属性計算を行なう順番 (S) を求める．次に”attrbibute_calcurate”(1267 行) で任意の生成規則の継承または合成属性の計算を行なう．

Algorithm3_1 *attribute sequence* attribute_parse_tree(parse tree ,attribute sequence S){
if(child1($p_{i_1,i_2,\cdots,i_k}$)=ϕ){
S[j] = 継承 and $p_{i_1,i_2,\cdots,i_k}$;
attribute_parse_tree_I(parse_tree,S[j+1]);
}
S[j] = 合成 and $p_{i_1,i_2,\cdots,i_k}$;
attribute_parse_tree_S(S[j+1]);
}

Algorithm3_2

attribute sequence attribute_parse_tree_I(parse tree, attribute sequence S){
if(child1($p_{i_1,\cdots,i_k}$) = ϕ){
S[j] = 継承 and $p_{i_1,\cdots,i_k}$;
attribute_parse_tree_I(parse_tree,S[j+1]);
}
S[++j] = 継承 and $p_{i_1,\cdots,i_k}$;
S[++j] = 合成 and $p_{i_1,\cdots,i_k}$;
attribute_parse_tree_S($p_{i_1,\cdots,i_{k-2},i_{k-1}}$, S[j]);
}

Algorithm3_3

attribute sequence attribute_parse_tree_I(parse tree, attribute sequence S){
if(lhs($p_{i_1,\cdots,i_k}$) = ”module_packet”) S[++j] = 合成 and $p_{i_1,\cdots,i_k}$;
else if(parent($p_{i_1,\cdots,i_k}$) $\neq \phi$, p $\neq$ child2(parent($p_{i_1,\cdots,i_k}$))){
S[++j] = 合成 and $p_{i_1,\cdots,i_k}$;
S[++j] = 合成 and $p_{i_1,\cdots,i_{k-1}}$;
S[++j] = 継承 and $p_{i_1,\cdots,i_{k-1},i_{k+1}}$;
attribute_parse_tree_I($p_{i_1,\cdots,i_{k-1},i_{k+1},i_{k+2}}$, S[j]);
else if(next($p_{i_1,\cdots,i_k}$)) child1($p_{i_1,\cdots,i_k}$)=ϕ){
S[j] = 継承 and $p_{i_1,\cdots,i_k}$;
attribute_parse_tree_I(parse_tree,S[j+1]);
}
S[j] = 合成 and $p_{i_1,i_2,\cdots,i_k}$;
attribute_parse_tree_S(S[j+1]);
}

4.1.3 例

今まで述べてきたアルゴリズムの一例を以下に示す．例えば図 8 を入力グラフとし Algorithm1 を実行する．入力グラフは”頂点”，”頂点アルファベット”，”頂点間の関係”，”セルの形”，”セルの幅と高さ”の情報からなる．この入力グラフを入力とし Algorithm2 を行なうと入力グラフに対して任意の生成規則を逆適応していくことによりスタートグラフに置き換える事ができる．したがって図 8 は構文解析可能な事がわかり構文木を作成する．図 8 に対する構文木を図 9 となり，構文木の各節は生成規則からなる．次にこの構文木を入力として，Algorithm3 を実行した場合，各生成規則の意味規則に基づいて属性の計算を行ない図 10 を出力する．図 10 は Hichart 図のレイアウト情報を属性計算によって各セルの座標に割り当てられている．また頂点アルファベットの下にある”[”,”]”内の数値は属性の一部であり，左から順に x,y 座標，部分木の最下部の y 座標，部分木の最上部の x 座標，セルの識別番号，部分木内に存在するセルの数を表している．また，四角内の数値はセルの幅，高さを表している．

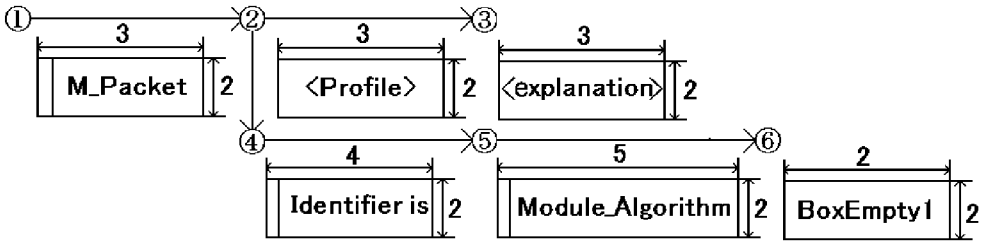


図 8: 入力グラフ

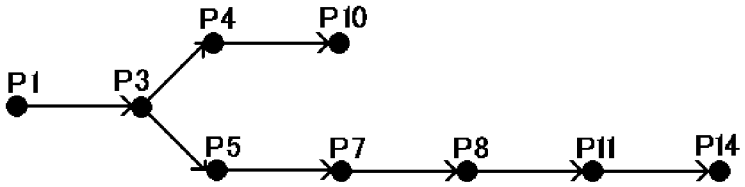


図 9: 構文木

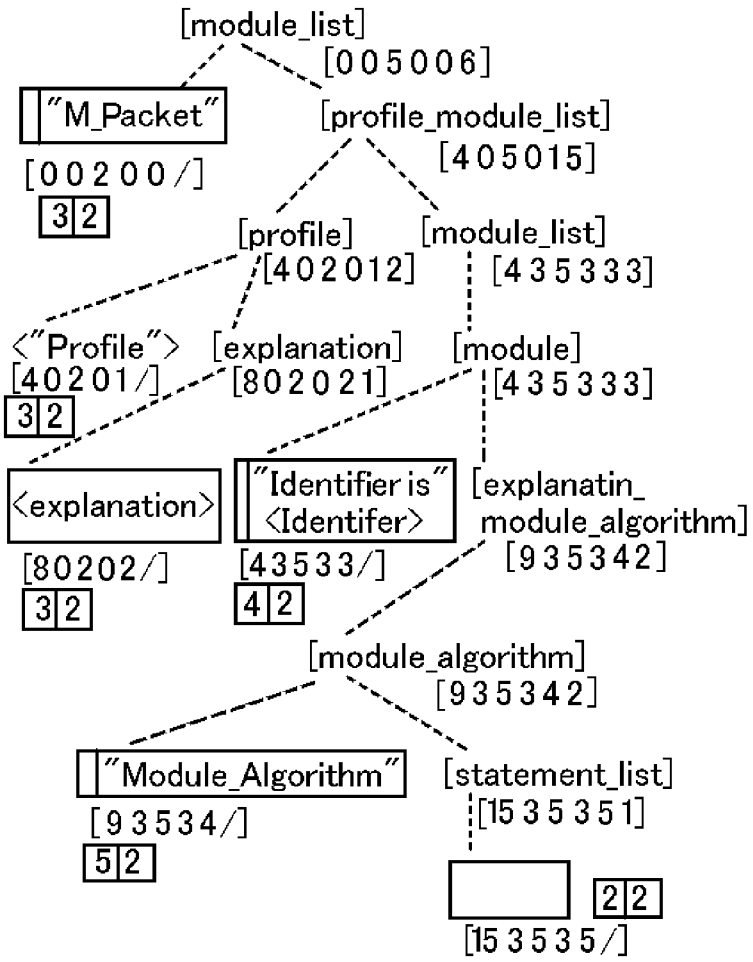


図 10: 属性評価後の構文木

4.2 HCPGGによる構文解析アルゴリズムの計算時間

定理 4.1 "HCPGG_analysis"は線形時間 O(m) で実行可能である.

証明 HCPGGにおいて，グラフ内の辺の数がmの場合，そのグラフに対する導出列の長さは高々mの定数倍となる．なぜなら1) 入力されたグラフに対して還元を行なうと高々11回でグラフ内の辺（頂点) が1つ減少する．2) 入力したグラフが文法的に正しければ高々11m+6回でスタートグラフに置き換えられる．の2つより上のことがいえる.

"precedence_analysis"は，導出列の長さの回数だけ行なわれる．従って高々11m+6回行なわれる.

"shift_order_list"は"precedence_analysis"が行なわれている間に，順位ハンドルとなる頂点の移動を辺の数であるm回移動する．等しい順位ハンドルの発見を高々定数 (5*4*3) 回行なっている.

"find_production"はまずif文で順位ハンドル内の頂点の個数に応じて3つに分岐する．次に，"順位ハンドル＝生成規則のrhs"となる生成規則を探索する．この時，すべての生成規則のrhsにおいて，頂点の個数は最大でも3個より，3つに分岐する．また，生成規則の個数は定数個である．よってO("find_production") = O(1) である.

"replace_graph"は"find_production"で発見した生成規則を用いてグラフ内の順位ハンドルを生成規則に置き換える．この時，HCPGGにおいて1つの頂点から接続される頂点の数は高々5個 (親のセル，子供のセル (2つ)，前のセル，次のセル) より，1つの情報を書き換える時間は定数で表わされる．また，"replace_graph"では最大でも3つの頂点の書き換えしか行なわない．よってO("replace_graph") = O(1) である.

"rewriting_order_list"では，HCPGGにおいて，Kの書き換えは高々Kの最後尾から3つの頂点までしか行なわない．よってO("rewriting_order_list") = O(1) である.

以上より precedence_analysis が 1 回行われるときの"find_production"，"replace_graph"，"rewriting_order_list"の計算時間はO(1) である.

また，"generate_parse_tree"の計算時間は導出列の長さの定数倍よりO(m) である.

以上の結果よりHCPGG_analysisは辺の数に対して線形時間で行なうことができる.

□

定理 4.2 "attribute_evaluation"は線形時間 (O(m)) で実行可能である.

証明 "attribute_evaluation"において，HCPGG_analysisの結果を受けて入力される構文木の頂点 (生成規則) の個数は高々(11m+6) である．また属性評価を行なうときに高々4回しか構文木の各頂点を通過しない．従って線形時間で実行可能である.

□

5 HCPGG parser

この章では，HCPGG の parser の実行例を示す．

まず，入力されるグラフを図 8 とする．この図の情報をプログラムに対して入力した例が図 11 である．

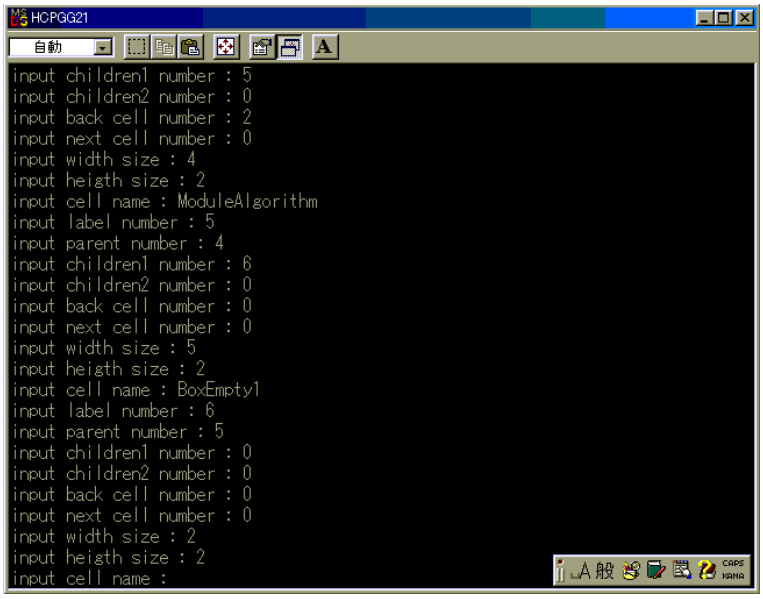


図 11: 入力画面

入力されたグラフは構文解析を行なわれ，その結果として得られたリストは以下のようになる．

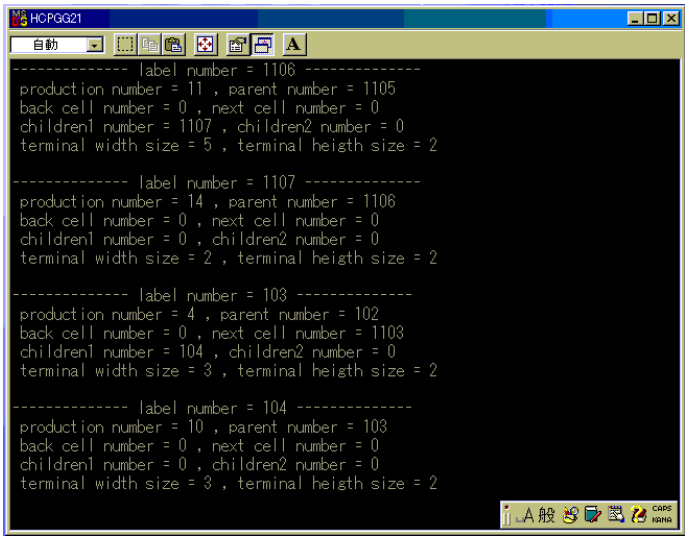


図 12: 出力画面 (構文木)

また，このリストは図 9 の木構造を表わしている．

図 13 は入力したグラフ内における終端頂点アルファベットの X 座標 (X)，セルの識別番号 (id)，Y 座標 (Y)，部分木の最下部の Y 座標 (bottom) を出力したものである．

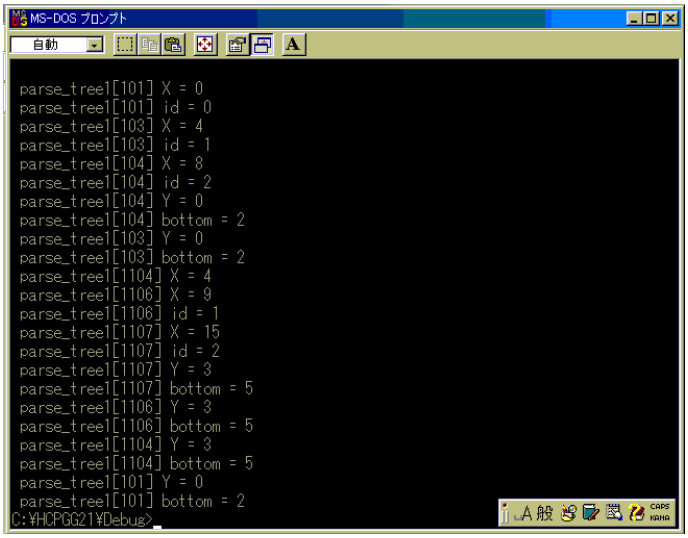


図 13: 出力画面 (属性評価)

また，このリストは図 10 内の終端頂点アルファベットの座標などの情報を表わしている．

6　終わりに

本論文では，HCGG と等価な順位グラフ文法の存在を確認し HCPGG を構成した．また HCPGG に対する構文解析アルゴリズムの構築および計算量の評価，また Parser の開発を約 7000 行で行なった．

今後は Hicahrt 処理システムのエディタ等の開発を行っていく必要がある．

参考文献

[1] T.Yaku, K.Futatsugi, A.Adachi and E.Moriya; HICHART–A Hierachical Flowchart Description Language–; *Proc. IEEE COMPSAC 11*, 157-163(1987)

[2] 西野哲郎; 属性グラフ文法とその Hichart 型プログラム図式に対するエディタへの応用; コンピュータソフトウェア, Vol,5, No.2, 81-92 (1988)

[3] Y.Adachi, K.Anzai, K.Tsuchida and T.Yaku; Hierachical Program Diagram Editor Based on Attribute Graph Grammar; *Proc. IEEE COMPSAC 20*, 205-213(1996)

[4] Y.Adachi, Y.Miyadera, K.Sugita, K.Tsuchida and T.Yaku; A Visual Programming Environment Based on Graph Grammars and Tidy Drawing; *Proc. ICSE 20*-Ⅱ,74-79(1998)

[5] 安達由洋, 大井裕一, 大澤優, 二木厚吉, 夜久竹夫; DXL 対応 Hichart プログラム図に対する属性グラフ文法; **日本大学文理学部自然科学研究所研究紀要第 33 号**, 149-164(1998)

[6] 木構造図用データ交換言語 DXL; JIS X 0130-1995; 日本工業標準審議会, 1-36 (1995)

[7] Grzegoz Rozenberg; Handbook of Graph Grammars and Computing by Graph Transformation; World Scientific(1996)

[8] 宮崎征宏, 類瀬健二, 土田賢省, 夜久竹夫; プログラム図に対する描画を考慮した NCE 属性グラフ文法; 電子情報通信学会技術研究報告 Vol.100 No.52,1-8(2000)

[9] T.Arita, K.Tomiyama, K. Tsuchida, Y. Yaku et al; Syntactic Processing of Diagrams by Graph Grammars; *Proc. IFIP WCC ICS2000* ,145-151(2000).

[10] Manfred Kaul; Practical Applications of Precedence Graph Grammars,Graph Grammars and Their Application to Computer Science, LNCS 291,326-342(December-1986),Virginia:Springer-Verlag

謝辞

本研究の機会を与えて下さり、数多くの示唆、実り多い議論、親切な御指導を下さった夜久竹夫教授に心より感謝の意を表します。

修士論文を丁寧に査読して下さり、数多くの助言を下さった東洋大学の土田賢省教授に心より感謝の意を表します。

ご指導、ご支援を頂きました応用数学科の諸先生方、有田友和氏、冨山聖宣氏、仲川俊一氏、ならびに同研究室の四年生の皆様に心よりお礼を申し上げます。

付録

1 HCPGG の各頂点アルファベット間の順位関係表

2 hichart 対応順位グラフ文法の生成規則 69 個

付録 1 HCPGG の各頂点アルファベット間の順位関係表

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
[module_packet]							
"M_Packet"			R_3	R_4		R_4	
[profile_module_list]							
[profile]					R_2		
[module_list]							
"Profile"					R_5		R_2
[explanation]							
[module]					R_8		
"Identifier_is"					R_5		R_4
[explanation_module_algorithm]							
[module_algorithm]							
<explanation>							
"Module_Algorithm"							

	["imperative" "call" <procedual_specification>]	["imperative" "goto" <procedual_specification>]	<block_identifier>	[abstract_compound]	[ ]	[abstract]	[compound_statement]
[module_packet]							
"M_Packet"							
[profile_module_list]							
[profile]							
[module_list]							
"Profile"							
[explanation]							
[module] "Identifier_is"							
[explanation_module_algorithm]							
[module_algorithm]							
<explanation>							
"Module_Algorithm"	R_4	R_4	R_4		R_4		

	[module]	"Identifier_is"	[explanation_module_algorithm]	[module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
[module_packet]								
"M_Packet"								
[profile_module_list]								
[profile]	R_4	R_4						
[module_list]								
"Profile"	R_{10}	R_{10}			R_4			
[explanation]				R_3		R_8		
[module]	R_8	R_8		R_8				
"Identifier_is"	R_{10}	R_{10}	R_2	R_4	R_6	R_6		
[explanation_module_algorithm]								
[module_algorithm]								
<explanation>			R_5	R_5		R_{10}		
"Module_Algorithm"							R_1	R_4

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[itetative_statement]
[module_packet]						
"M_Packet"						
[profile_module_list]						
[profile]						
[module_list]						
"Profile"						
[explanation]						
"Identifier_is"						
[explanation_module_algorithm]						
[module_algorithm]						
<explanation>						
"Module_Algorithm"						

	[ ]	[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	["imperative" <procedual_specification>]	["imperative" "null" <procedual_specification>]
[module_packet]							
"M_Packet"							
[profile_module_list]							
[profile]							
[module_list]							
"Profile"							
[explanation]							
"Identifier_is"							
[explanation_module_algorithm]							
[module_algorithm]							
<explanation>							
"Module_Algorithm"	R_4	R_4	R_4	R_4	R_4	R_4	R_4

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
[module_packet]								
"M_Packet"								
[profile_module_list]								
[profile]								
[module_list]								
"Profile"								
[explanation]								
"Identifier_is"								
[explanation_module_algorithm]								
[module_algorithm]						R_4		
<explanation>								
"Module_Algorithm"								

	[continued_statement]	"condition" "for" <conditional_specification>	"condition" "until" <conditional_specification>	"condition" "while" <conditional_specification>	"condition" <conditional_specification>
[module_packet]					
"M_Packet"					
[profile_module_list]					
[profile]					
[module_list]					
"Profile"					
[explanation]					
"Identifier_is"					
[explanation_module_algorithm]					
[module_algorithm]					
<explanation>					
"Module_Algorithm"					

	[exclusive_case_statement]	[else_if_statement]	"else_if" <conditional_specification> "then"	"case" <evalution_specification>	"case" <evalution_specification>	[when_statement]
[module_packet]						
"M_Packet"						
[profile_module_list]						
[profile]						
[module_list]						
"Profile"						
[explanation]						
"Identifier_is"						
[explanation_module_algorithm]						
[module_algorithm]						
<explanation>						
"Module_Algorithm"						

	"condition" "while" <conditional_specification>	"condition" "until" <conditional_specification>	"condition" <conditional_specification>	"Loop"	[if_then_statement]	[exclusive_select_statement]
[module_packet]						
"M_Packet"						
[profile_module_list]						
[profile]						
[module_list]						
"Profile"						
[explanation]						
"Identifier_is"						
[explanation_module_algorithm]						
[module_algorithm]						
<explanation>						
"Module_Algorithm"						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
[module_packet]			
"M_Packet"			
[profile_module_list]			
[profile]			
[module_list]			
"Profile"			
[explanation]			
"Identifier_is"			
[explanation_module_algorithm]			
[module_algorithm]			
<explanation>			
"Module_Algorithm"	R_6	R_6	R_6

	[include_select_statement]	"if" <conditional_specification> "then"	[branch_statement_list_cushion]	[branch_statement_list]		[exclusive_if_statement]
[module_packet]						
"M_Packet"						
[profile_module_list]						
[profile]						
[module_list]						
"Profile"						
[explanation]						
"Identifier_is"						
[explanation_module_algorithm]						
[module_algorithm]						
<explanation>						
"Module_Algorithm"						

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
[statement_list]							
[label_statement]							
[statement]							
[fundamental_statement]							
[blocked_staatement]							
[terminal_statement]							
"imperative" <procedual_specification>							
"imperative" "null" <procedual_specification>							

	"imperative" "call" <procedual_specification>	"imperative" "goto" <procedual_specification>	<block_identifier>	[abstract_compound]		[abstract]	[compound_statement]
[statement_list]							
[label_statement]	R_c	R_c	R_c		R_c		
[statement]	R_{st}	R_{st}	R_{st}		R_{st}		
[fundamental_statement]	R_{st}	R_{st}	R_{st}		R_{st}		
[blocked_staatement]	R_{st}	R_{st}	R_{st}		R_{st}		
[terminal_statement]	R_{st}	R_{st}	R_{st}		R_{st}		
"imperative" <procedual_specification>	R_{st}	R_{st}	R_{st}		R_{st}		
"imperative" "null" <procedual_specification>	R_{st}	R_{st}	R_{st}		R_{st}		

	"Identifier_is"	[explanation_module_algorithm]	[module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
[statement_list]							
[label_statement]						R_c	R_c
[statement]						R_{st}	R_{st}
[fundamental_statement]						R_{st}	R_{st}
[blocked_staatement]						R_{st}	R_{st}
[terminal_statement]						R_{st}	R_{st}
"imperative" <procedual_specification>						R_{st}	R_{st}
"imperative" "null" <procedual_specification>						R_{st}	R_{st}

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[itetative_statement]
[statement_list]						
[label_statement]						
[statement]						
[fundamental_statement]						
[blocked_staatement]						
[terminal_statement]						
"imperative" <procedual_specification>						
"imperative" "null" <procedual_specification>						

		[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	"imperative" <procedual_specification>	"imperative" "null" <procedual_specification>
[statement_list]							
[label_statement]	R_c	R_c	R_c	R_c	R_c	R_c	R_c
[statement]	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}
[fundamental_statement]	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}
[blocked_staatement]	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}
[terminal_statement]	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}
"imperative" <procedual_specification>	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}
"imperative" "null" <procedual_specification>	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}	R_{st}

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
[statement_list]								
[label_statement]								
[statement]								
[fundamental_statement]								
[blocked_staatement]								
[terminal_statement]								
"imperative" <procedual_specification>								
"imperative" "null" <procedual_specification>								

	[continued _statement]	"condition" "for" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" "while" <conditional _specification>	"condition" <conditional _specification>
[statement _list]					
[label _statement]					
[statement]					
[fundamental _statement]					
[blocked _staatement]					
[terminal _statement]					
"imperative" <procedual _specification>					
"imperative" "null" <procedual _specification>					

	[exclusive _case _statement]	[else_if _statement]	"else_if" <conditional _specification> "then"	"case" <evalution _specification>	"case" <evalution _specification>	[when _statement]
[statement _list]						
[label _statement]						
[statement]						
[fundamental _statement]						
[blocked _staatement]						
[terminal _statement]						
"imperative" <procedual _specification>						
"imperative" "null" <procedual _specification>						

	"condition" "while" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" <conditional _specification>	"Loop"	[if_then _statement]	[exclusive _select _statement]
[statement _list]						
[label _statement]						
[statement]						
[fundamental _statement]						
[blocked _staatement]						
[terminal _statement]						
"imperative" <procedual _specification>						
"imperative" "null" <procedual _specification>						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
[statement _list]			
[label _statement]	R_{ϵ}	R_{ϵ}	R_{ϵ}
[statement]	R_{*}	R_{*}	R_{*}
[fundamental _statement]	R_{*}	R_{*}	R_{*}
[blocked _staatement]	R_{*}	R_{*}	R_{*}
[terminal _statement]	R_{*}	R_{*}	R_{*}
"imperative" <procedual _specification>	R_{*}	R_{*}	R_{*}
"imperative" "null" <procedual _specification>	R_{*}	R_{*}	R_{*}

	[include _select _statement]	"if" <conditional _specification> "then"	[branch _statement _list_cushion]	[branch _statement_list]		[exclusive _if_statement]
[statement _list]						
[label _statement]						
[statement]						
[fundamental _statement]						
[blocked _staatement]						
[terminal _statement]						
"imperative" <procedual _specification>						
"imperative" "null" <procedual _specification>						

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
"imperative" "call" <procedual_specification>							
"imperative" "goto" <procedual_specification>							
<block_identifier>							
[abstract_compound]							
[ ]							
[abstract]							
[compound_statement]							
"abstract"							
[block_specification]							

	"imperative" "call" <procedual_specification>	"imperative" "goto" <procedual_specification>	<block_identifier>	[abstract_compound]	[ ]	[abstract]	[compound_statement]
"imperative" "call" <procedual_specification>	R_{36}	R_{36}	R_{36}		R_{36}		
"imperative" "goto" <procedual_specification>	R_{36}	R_{36}	R_{36}		R_{36}		
<block_identifier>	R_{36}	R_{36}	R_{36}	R_{35}	R_{36}	R_{45}	R_{45}
[abstract_compound]							
[ ]	R_{36}	R_{36}	R_{36}	R_{35}	R_{36}	R_{45}	R_{45}
[abstract]							R_{45}
[compound_statement]							
"abstract"							R_{59}
[block_specification]							

	"Identifier_is"	[explanation_module_algorithm]	[module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
"imperative" "call" <procedual_specification>						R_{59}	R_{36}
"imperative" "goto" <procedual_specification>						R_{59}	R_{36}
<block_identifier>						R_{59}	R_{36}
[abstract_compound]							
[ ]						R_{59}	R_{36}
[abstract]							
[compound_statement]							
"abstract"							
[block_specification]							

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[itetative_statement]
"imperative" "call" <procedual_specification>						
"imperative" "goto" <procedual_specification>						
<block_identifier>	R_{49}			R_{49}	R_{49}	R_{49}
[abstract_compound]						
[ ]	R_{45}			R_{49}	R_{49}	R_{49}
[abstract]				R_{45}	R_{45}	R_{45}
[compound_statement]						
"abstract"		R_{59}	R_{49}	R_{36}	R_{36}	R_{36}
[block_specification]						

	[ ]	[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	"imperative" <procedual_specification>	"imperative" "null" <procedual_specification>
"imperative" "call" <procedual_specification>	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}
"imperative" "goto" <procedual_specification>	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}
<block_identifier>	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}
[abstract_compound]							
[ ]	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}	R_{36}
[abstract]							
[compound_statement]							
"abstract"							
[block_specification]							

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
"imperative" "call" <procedual_specification>								
"imperative" "goto" <procedual_specification>								
<block_identifier>	R_{49}	R_{49}	R_{49}				R_{49}	R_{49}
[abstract_compound]								
[ ]	R_{49}	R_{49}	R_{49}				R_{49}	R_{49}
[abstract]	R_{45}	R_{49}	R_{45}				R_{45}	R_{45}
[compound_statement]								
"abstract"	R_{36}	R_{36}	R_{36}				R_{36}	R_{36}
[block_specification]								

	[continued _statement]	"condition" "for" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" "while" <conditional _specification>	"condition" <conditional _specification>
"imperative" "call" <procedual _specification>					
"imperative" "goto" <procedual _specification>					
<block _identifier>	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}
[abstract _compound]					
	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}
[abstract]	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}
[compound _statement]					
"abstract"	R_{∞}	R_{∞}	R_{∞}	R_{∞}	R_{∞}
[block _specification]					

	[exclusive _case _statement]	[else_if _statement]	"else_if" <conditional _specification> "then"	"case" <evalution _specification>	"case" <evalution _specification>	[when _statement]
"imperative" "call" <procedual _specification>						
"imperative" "goto" <procedual _specification>						
<block _identifier>	R_{ϕ}			R_{ϕ}	R_{ϕ}	
[abstract _compound]						
	R_{ϕ}			R_{ϕ}	R_{ϕ}	
[abstract]	R_{ϕ}			R_{ϕ}	R_{ϕ}	
[compound _statement]						
"abstract"	R_{∞}			R_{∞}	R_{∞}	
[block _specification]						

	"condition" "while" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" <conditional _specification>	"Loop"	[if_then _statement]	[exclusive _select _statement]
"imperative" "call" <procedual _specification>						
"imperative" "goto" <procedual _specification>						
<block _identifier>	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}
[abstract _compound]						
	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}
[abstract]	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}	R_{ϕ}
[compound _statement]						
"abstract"	R_{∞}	R_{∞}	R_{∞}	R_{∞}	R_{∞}	R_{∞}
[block _specification]						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
"imperative" "call" <procedual _specification>	R_{∞}	R_{∞}	R_{∞}
"imperative" "goto" <procedual _specification>	R_{∞}	R_{∞}	R_{∞}
<block _identifier>	R_{∞}	R_{∞}	R_{∞}
[abstract _compound]			
	R_{∞}	R_{∞}	R_{∞}
[abstract]			
[compound _statement]			
"abstract"			
[block _specification]			

	[include _select _statement]	"if" <conditional _specification> "then"	[branch _statement _list_cushion]	[branch _statement_list]		[exclusive _if_statement]
"imperative" "call" <procedual _specification>						
"imperative" "goto" <procedual _specification>						
<block _identifier>	R_{ϕ}	R_{ϕ}				R_{ϕ}
[abstract _compound]						
	R_{ϕ}	R_{ϕ}				R_{ϕ}
[abstract]	R_{ϕ}	R_{ϕ}				R_{ϕ}
[compound _statement]						
"abstract"	R_{∞}	R_{∞}				R_{∞}
[block _specification]						

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
"block_specification"							
[sequential_statement]							
[parallel_statement]							
[itetative_statement]							
[selective_statement]							
"begin"							
"parallel"							
[parallel_statement_list]							
●							
[label_statement_cushion]							

	"impertive" "call" <procedual_specification>	"impertive" "goto" <procedual_specification>	<block_identifier>	[abstract_compound]	[ ]	[abstract]	[compound_statement]
"block_specification"							
[sequential_statement]							
[parallel_statement]							
[itetative_statement]							
[selective_statement]							
"begin"	R_6	R_4	R_4		R_6		
"parallel"	R_6	R_4	R_4		R_6		
[parallel_statement_list]							
●	R_6	R_4	R_4		R_6		
[label_statement_cushion]							

	"Identifier_is"	[explanation_module_algorithm]	[module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
"block_specification"							
[sequential_statement]							
[parallel_statement]							
[itetative_statement]							
[selective_statement]							
"begin"						R_5	R_6
"parallel"							R_6
[parallel_statement_list]							
●							R_6
[label_statement_cushion]							

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[itetative_statement]
"block_specification"						
[sequential_statement]						
[parallel_statement]						
[itetative_statement]						
[selective_statement]						
"begin"						
"parallel"						
[parallel_statement_list]						
●						
[label_statement_cushion]						

	[ ]	[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	"imperative" <procedual_specification>	"imperative" "null" <procedual_specification>
"block_specification"							
[sequential_statement]							
[parallel_statement]							
[itetative_statement]							
[selective_statement]							
"begin"	R_6	R_6	R_6	R_6	R_6	R_6	R_6
"parallel"		R_6	R_6	R_6	R_6	R_6	R_6
[parallel_statement_list]							
●		R_6	R_6	R_6	R_6	R_6	R_6
[label_statement_cushion]							

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
"block_specification"								
[sequential_statement]								
[parallel_statement]								
[itetative_statement]								
[selective_statement]								
"begin"								
"parallel"				R_5	R_6			
[parallel_statement_list]								
●				R_5	R_6	R_5		
[label_statement_cushion]								

	[continued _statement]	"condition" "for" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" "while" <conditional _specification>	"condition" <conditional specification>
"block _specification"					
[sequential _statement]					
[parallel _statement]					
[itetative _statement]					
[selective _statement]					
"begin"					
"parallel"					
[parallel _statement _list]					
●					
[label _statement _cushion]					

	[exclusive _case _statement]	[else_if _statement]	"else_if" <conditional _specification> "then"	"case" <evalution _specification>	"case" <evalution _specification>	[when _statement]
"block _specification"						
[sequential _statement]						
[parallel _statement]						
[itetative _statement]						
[selective _statement]						
"begin"						
"parallel"						
[parallel _statement _list]						
●						R_5
[label _statement _cushion]						

	"condition" "while" <conditional specification>	"condition" "until" <conditional specification>	"condition" <conditional _specification>	"Loop"	[if_then _statement]	[exclusive _select _statement]
"block _specification"						
[sequential _statement]						
[parallel _statement]						
[itetative _statement]						
[selective _statement]						
"begin"						
"parallel"						
[parallel _statement _list]						
●						
[label _statement _cushion]						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
"block _specification"			
[sequential _statement]			
[parallel _statement]			
[itetative _statement]			
[selective _statement]			
"begin"	R_6	R_6	R_6
"parallel"	R_6	R_6	R_6
[parallel _statement _list]			
●	R_6	R_6	R_6
[label _statement _cushion]			

	[include _select _statement]	"if" <conditional _specification> "then"	[branch _statement _list_cushion]	[branch _statement_list]		[exclusive _if_statement]
"block _specification"						
[sequential _statement]						
[parallel _statement]						
[itetative _statement]						
[selective _statement]						
"begin"						
"parallel"						
[parallel _statement _list]						
●			R_5	R_4		
[label _statement _cushion]						

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
[pre_tested_statement]							
[post_tested_statement]							
[continued_statement]							
"condition" "for" <conditional_specification>							
"condition" "until" <conditional_specification>							
"condition" "while" <conditional_specification>							
"condition" <conditional_specification>							

	"imperative" "call" <procedual_specification>	"imperative" "goto" <procedual_specification>	<block_identifier>	[abstract_compound]	[ ]	[abstract]	[compound_statement]
[pre_tested_statement]							
[post_tested_statement]							
[continued_statement]							
"condition" "for" <conditional_specification>	R_6	R_6	R_6		R_6		
"condition" "until" <conditional_specification>	R_6	R_6	R_6		R_6		
"condition" "while" <conditional_specification>	R_6	R_6	R_6		R_6		
"condition" <conditional_specification>	R_6	R_6	R_6		R_6		

	"Identifier_is"	[explanation_module_algorithm]	[module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
[pre_tested_statement]							
[post_tested_statement]							
[continued_statement]							
"condition" "for" <conditional_specification>					R_6	R_5	R_6
"condition" "until" <conditional_specification>					R_6	R_5	R_6
"condition" "while" <conditional_specification>					R_6	R_5	R_6
"condition" <conditional_specification>					R_6	R_5	R_6

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[iterative_statement]
[pre_tested_statement]						
[post_tested_statement]						
[continued_statement]						
"condition" "for" <conditional_specification>						
"condition" "until" <conditional_specification>						
"condition" "while" <conditional_specification>						
"condition" <conditional_specification>						

	[ ]	[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	"imperative" <procedual_specification>	"imperative" "null" <procedual_specification>
[pre_tested_statement]							
[post_tested_statement]							
[continued_statement]							
"condition" "for" <conditional_specification>	R_6	R_6	R_6	R_6	R_6	R_6	R_6
"condition" "until" <conditional_specification>	R_6	R_6	R_6	R_6	R_6	R_6	R_6
"condition" "while" <conditional_specification>	R_6	R_6	R_6	R_6	R_6	R_6	R_6
"condition" <conditional_specification>	R_6	R_6	R_6	R_6	R_6	R_6	R_6

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
[pre_tested_statement]								
[post_tested_statement]								
[continued_statement]								
"condition" "for" <conditional_specification>								
"condition" "until" <conditional_specification>								
"condition" "while" <conditional_specification>								
"condition" <conditional_specification>								

	[continued _statement]	"condition" "for" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" "while" <conditional _specification>	"condition" <conditional _specification>
[pre_tested _statement]					
[post_tested _statement]					
[continued _statement]					
"condition" "for" <conditional _specification>					
"condition" "until" <conditional _specification>					
"condition" "while" <conditional _specification>					
"condition" <conditional _specification>					

	[exclusive _case _statement]	[else_if _statement]	"else_if" <conditional _specification> "then"	"case" <evalution _specification>	"case" <evalution _specification>	[when _statement]
[pre_tested _statement]						
[post_tested _statement]						
[continued _statement]						
"condition" "for" <conditional _specification>						
"condition" "until" <conditional _specification>						
"condition" "while" <conditional _specification>						
"condition" <conditional _specification>						

	"condition" "while" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" <conditional _specification>	"Loop"	[if_then _statement]	[exclusive _select _statement]
[pre_tested _statement]						
[post_tested _statement]						
[continued _statement]						
"condition" "for" <conditional _specification>						
"condition" "until" <conditional _specification>						
"condition" "while" <conditional _specification>						
"condition" <conditional _specification>						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
[pre_tested _statement]			
[post_tested _statement]			
[continued _statement]			
"condition" "for" <conditional _specification>	R4	R4	R4
"condition" "until" <conditional _specification>	R4	R4	R4
"condition" "while" <conditional _specification>	R4	R4	R4
"condition" <conditional _specification>	R4	R4	R4

	[include _select _statement]	"if" <conditional _specification> "then"	[branch _statement _list_cushion]	[branch _statement_list]		[exclusive _if_statement]
[pre_tested _statement]						
[post_tested _statement]						
[continued _statement]						
"condition" "for" <conditional _specification>						
"condition" "until" <conditional _specification>						
"condition" "while" <conditional _specification>						
"condition" <conditional _specification>						

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
"condition" "while" <conditional_specification>							
"condition" "until" <conditional_specification>							
"condition" <conditional_specification>							
"Loop"							
[if_then_statement]							
[exclusive_select_statement]							
[include_select_statement]							

	"imperative" "call" <procedual_specification>	"imperative" "goto" <procedual_specification>	<block_identifier>	[abstract_compound]		[abstract]	[compound_statement]
"condition" "while" <conditional_specification>	R_6	R_6	R_6		R_6		
"condition" "until" <conditional_specification>	R_6	R_6	R_6		R_6		
"condition" <conditional_specification>	R_6	R_6	R_6		R_6		
"Loop"	R_6	R_6	R_6		R_6		
[if_then_statement]							
[exclusive_select_statement]							
[include_select_statement]							

	"Identifier_is"	[explanation_module_algorithm]	[module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
"condition" "while" <conditional_specification>					R_6	R_6	R_6
"condition" "until" <conditional_specification>					R_6	R_6	R_6
"condition" <conditional_specification>					R_6	R_6	R_6
"Loop"					R_6	R_6	R_6
[if_then_statement]							
[exclusive_select_statement]							
[include_select_statement]							

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[itetative_statement]
"condition" "while" <conditional_specification>						
"condition" "until" <conditional_specification>						
"condition" <conditional_specification>						
"Loop"						
[if_then_statement]						
[exclusive_select_statement]						
[include_select_statement]						

		[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	"imperative" <procedual_specification>	"imperative" "null" <procedual_specification>
"condition" "while" <conditional_specification>	R_6	R_6	R_6	R_6	R_6	R_6	R_6
"condition" "until" <conditional_specification>	R_6	R_6	R_6	R_6	R_6	R_6	R_6
"condition" <conditional_specification>	R_6	R_6	R_6	R_6	R_6	R_6	R_6
"Loop"	R_6	R_6	R_6	R_6	R_6	R_6	R_6
[if_then_statement]							
[exclusive_select_statement]							
[include_select_statement]							

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
"condition" "while" <conditional_specification>								
"condition" "until" <conditional_specification>								
"condition" <conditional_specification>								
"Loop"								
[if_then_statement]								
[exclusive_select_statement]								
[include_select_statement]								

	[continued_statement]	"condition" "for" <conditional_specification>	"condition" "until" <conditional_specification>	"condition" "while" <conditional_specification>	"condition" <conditional_specification>
"condition" "while" <conditional_specification>					
"condition" "until" <conditional_specification>					
"condition" <conditional_specification>					
"Loop"					
[if_then_statement]					
[exclusive_select_statement]					
[include_select_statement]					

	[exclusive_case_statement]	[else_if_statement]	"else_if" <conditional_specification> "then"	"case" <evalution_specification>	"case" <evalution_specification>	[when_statement]
"condition" "while" <conditional_specification>						
"condition" "until" <conditional_specification>						
"condition" <conditional_specification>						
"Loop"						
[if_then_statement]						
[exclusive_select_statement]						
[include_select_statement]						

	"condition" "while" <conditional_specification>	"condition" "until" <conditional_specification>	"condition" <conditional_specification>	"Loop"	[if_then_statement]	[exclusive_select_statement]
"condition" "while" <conditional_specification>						
"condition" "until" <conditional_specification>						
"condition" <conditional_specification>						
"Loop"						
[if_then_statement]						
[exclusive_select_statement]						
[include_select_statement]						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
"condition" "while" <conditional_specification>	R_{ϕ}	R_{ϕ}	R_{ϕ}
"condition" "until" <conditional_specification>	R_{ϕ}	R_{ϕ}	R_{ϕ}
"condition" <conditional_specification>	R_{ϕ}	R_{ϕ}	R_{ϕ}
"Loop"	R_{ϕ}	R_{ϕ}	R_{ϕ}
[if_then_statement]			
[exclusive_select_statement]			
[include_select_statement]			

	[include_select_statement]	"if" <conditional_specification> "then"	[branch_statement_list_cushion]	[branch_statement_list]		[exclusive_if_statement]
"condition" "while" <conditional_specification>						
"condition" "until" <conditional_specification>						
"condition" <conditional_specification>						
"Loop"						
[if_then_statement]						
[exclusive_select_statement]						
[include_select_statement]						

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
"if" <conditional_specification> "then"							
[branch_statement_list_cushion]							
[branch_statement_list]							
[exclusive_if_statement]							
[exclusive_case_statement]							
[else_if_statement]							
"else_if" <conditional_specification> "then"							

	"imperative" "call" <procedual_specification>	"imperative" "goto" <procedual_specification>	<block_identifier>	[abstract_compound]		[abstract]	[compound_statement]
"if" <conditional_specification> "then"							
[branch_statement_list_cushion]							
[branch_statement_list]							
	R_6	R_6	R_4		R_4		
[exclusive_if_statement]							
[exclusive_case_statement]							
[else_if_statement]							
"else_if" <conditional_specification> "then"							

	"Identifier_is"	[explanation_module_algorithm]	[module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
"if" <conditional_specification> "then"							
[branch_statement_list_cushion]							
[branch_statement_list]							
					R_6	R_5	R_6
[exclusive_if_statement]							
[exclusive_case_statement]							
[else_if_statement]							
"else_if" <conditional_specification> "then"							

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[itetative_statement]
"if" <conditional_specification> "then"						
[branch_statement_list_cushion]						
[branch_statement_list]						
[exclusive_if_statement]						
[exclusive_case_statement]						
[else_if_statement]						
"else_if" <conditional_specification> "then"						

		[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	"imperative" <procedual_specification>	"imperative" "null" <procedual_specification>
"if" <conditional_specification> "then"							
[branch_statement_list_cushion]							
[branch_statement_list]							
	R_6	R_6	R_6	R_6	R_6	R_6	R_6
[exclusive_if_statement]							
[exclusive_case_statement]							
[else_if_statement]							
"else_if" <conditional_specification> "then"							

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
"if" <conditional_specification> "then"								
[branch_statement_list_cushion]								
[branch_statement_list]								
[exclusive_if_statement]								
[exclusive_case_statement]								
[else_if_statement]								
"else_if" <conditional_specification> "then"								

	[continued _statement]	"condition" "for" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" "while" <conditional _specification>	"condition" <conditional _specification>
"if" <conditional _specification> "then"					
[branch _statement _list_cushion]					
[branch _statement_list]					
[exclusive _if_statement]					
[exclusive _case _statement]					
[else_if _statement]					
"else_if" <conditional _specification> "then"					

	[exclusive _case _statement]	[else_if _statement]	"else_if" <conditional _specification> "then"	"case" <evalution _specification>	"case" <evalution _specification>	[when _statement]
"if" <conditional _specification> "then"		R_3	R_4			
[branch _statement _list_cushion]						
[branch _statement_list]						
[exclusive _if_statement]						
[exclusive _case _statement]						
[else_if _statement]						
"else_if" <conditional _specification> "then"		R_3	R_4			

	"condition" "while" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" <conditional _specification>	"Loop"	[if_then _statement]	[exclusive _select _statement]
"if" <conditional _specification> "then"						
[branch _statement _list_cushion]						
[branch _statement_list]						
[exclusive _if_statement]						
[exclusive _case _statement]						
[else_if _statement]						
"else_if" <conditional _specification> "then"						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
"if" <conditional _specification> "then"			
[branch _statement _list_cushion]			
[branch _statement_list]			
	R_3	R_5	R_6
[exclusive _if_statement]			
[exclusive _case _statement]			
[else_if _statement]			
"else_if" <conditional _specification> "then"			

	[include _select _statement]	"if" <conditional _specification> "then"	[branch _statement _list_cushion]	[branch _statement_list]		[exclusive _if_statement]
"if" <conditional _specification> "then"			R_2	R_5	R_6	R_8
[branch _statement _list_cushion]						
[branch _statement_list]						
[exclusive _if_statement]						
[exclusive _case _statement]						
[else_if _statement]						
"else_if" <conditional _specification> "then"			R_2	R_5	R_6	

	[module_packet]	"M_Packet"	[profile_module_list]	[profile]	[module_list]	"Profile"	[explanation]
"case" <evalution_specification>							
"case" <evalution_specification>							
[when_statement]							
"terminate" "system" <specification>							
"terminate" "module" <specification>							
"terminate" "block" <block_identifier> <specification>							

	"imperative" "call" <procedual_specification>	"imperative" "goto" <procedual_specification>	<block_identifier>	[abstract_compound]	[ ]	[abstract]	[compound_statement]
"case" <evalution_specification>							
"case" <evalution_specification>							
[when_statement]							
"terminate" "system" <specification>	R_∞	R_∞	R_∞		R_∞		
"terminate" "module" <specification>	R_∞	R_∞	R_∞		R_∞		
"terminate" "block" <block_identifier> <specification>	R_∞	R_∞	R_∞		R_∞		

	"Identifier_is"	[explanation_module_algorithm]	<explanation>	"Module_Algorithm"	[statement_list]	[label_statement]
"case" <evalution_specification>						
"case" <evalution_specification>						
[when_statement]						
"terminate" "system" <specification>				R_∞	R_0	R_∞
"terminate" "module" <specification>				R_∞	R_0	R_∞
"terminate" "block" <block_identifier> <specification>				R_∞	R_0	R_∞

	"abstract"	[block_specification]	"block_specification"	[sequential_statement]	[parallel_statement]	[itotative_statement]
"case" <evalution_specification>						
"case" <evalution_specification>						
[when_statement]						
"terminate" "system" <specification>						
"terminate" "module" <specification>						
"terminate" "block" <block_identifier> <specification>						

	[ ]	[statement]	[fundamental_statement]	[blocked_statement]	[terminal_statement]	"imperative" <procedual_specification>	"imperative" "null" <procedual_specification>
"case" <evalution_specification>							
"case" <evalution_specification>							
[when_statement]							
"terminate" "system" <specification>	R_∞	R_∞	R_∞	R_∞	R_∞	R_∞	R_∞
"terminate" "module" <specification>	R_∞	R_∞	R_∞	R_∞	R_∞	R_∞	R_∞
"terminate" "block" <block_identifier> <specification>	R_∞	R_∞	R_∞	R_∞	R_∞	R_∞	R_∞

	[selective_statement]	"begin"	"parallel"	[parallel_statement_list]	●	[label_statement_cushion]	[pre_tested_statement]	[post_tested_statement]
"case" <evalution_specification>					R_0			
"case" <evalution_specification>					R_0			
[when_statement]								
"terminate" "system" <specification>								
"terminate" "module" <specification>								
"terminate" "block" <block_identifier> <specification>								

	[continued _statement]	"condition" "for" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" "while" <conditional _specification>	"condition" <conditional _specification>
"case" <evalution _specification>					
"case" <evalution _specification>					
[when _statement]					
"terminate" "system" <specification>					
"terminate" "module" <specification>					
"terminate" "block" <block_identifier> <specification>					

	[exclusive _case _statement]	[else_if _statement]	"else_if" <conditional _specification> "then"	"case" <evalution specification>	"case" <evalution specification>	[when _statement]
"case" <evalution _specification>						R_{**}
"case" <evalution _specification>						R_{**}
[when _statement]						
"terminate" "system" <specification>						
"terminate" "module" <specification>						
"terminate" "block" <block_identifier> <specification>						

	"condition" "while" <conditional _specification>	"condition" "until" <conditional _specification>	"condition" <conditional _specification>	"Loop"	[if_then _statement]	[exclusive _select _statement]
"case" <evalution _specification>						
"case" <evalution _specification>						
[when _statement]						
"terminate" "system" <specification>						
"terminate" "module" <specification>						
"terminate" "block" <block_identifier> <specification>						

	"terminate" "system" <specification>	"terminate" "module" <specification>	"terminate" "block" <block_identifier> <specification>
"case" <evalution _specification>			
"case" <evalution _specification>			
[when _statement]			
"terminate" "system" <specification>	R_{**}	R_{**}	R_{**}
"terminate" "module" <specification>	R_{**}	R_{**}	R_{**}
"terminate" "block" <block_identifier> <specification>	R_{**}	R_{**}	R_{**}

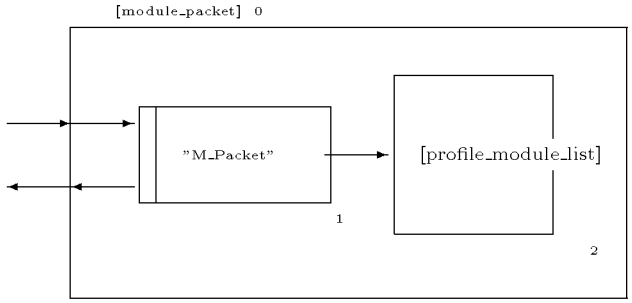
	[include _select _statement]	"if" <conditional specification> "then"	[branch _statement _list_cushion]	[branch _statement_list]		[exclusive _if_statement]
"case" <evalution _specification>				R_{**}		
"case" <evalution _specification>				R_{**}		
[when _statement]						
"terminate" "system" <specification>						
"terminate" "module" <specification>						
"terminate" "block" <block_identifier> <specification>						

付録 2 hichart 対応順位グラフ文法の生成規則 69 個

DXL – Production Rule – 01

module_packet

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1)+w(1)+GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max(bottom(1),bottom(2))
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1)+1
nc(0) = nc(2)+1

w(1) = MinW
h(1) = get_height(["M_Packet"])
cell(1) = "pre_defined_process"
string(1) = get_str(["M_Packet"])
lines(1) = get_line_begin(1,[2])
```

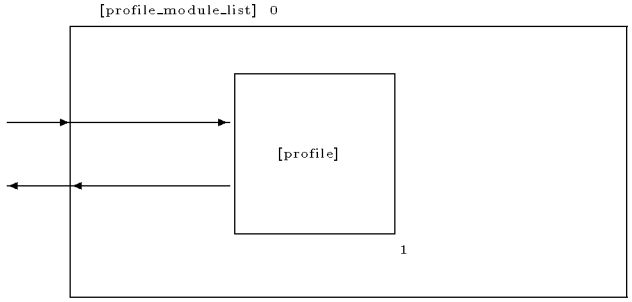
DXL

```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'
```

DXL – Production Rule – 02

profile_module_list(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1)=top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

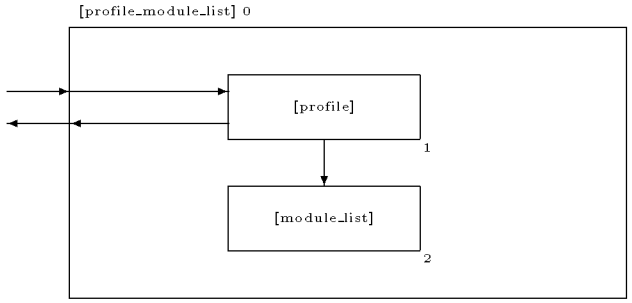
DXL

```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'
```

DXL – Production Rule – 03

profile_module_list(2)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1)
y(0) = y(1)
top(1) = top(0)
top(2) = bottom(1)+GapY
bottom(0) = bottom(2)
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)

line(1) = get_line(1,[2])
```

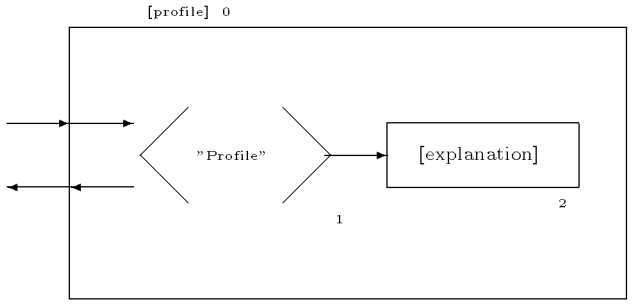
DXL

```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'
```

DXL – Production Rule – 04

profile

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2) = top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
bottom(2) = y(2) + h(2)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( ["Profile" ] )
cell(1) = "caption"
string(1) = get_str( ["Profile"] )
line(1) = get_line(1,[2])
```

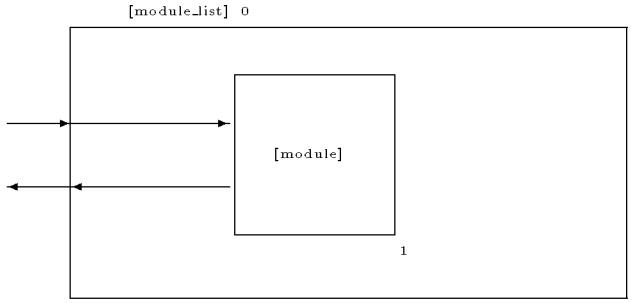
DXL

```
< プロファイル句 > ::=
  'Profile'
  [ 説明記述 ]
  'End_Profile' ';'
```

DXL – Production Rule – 05

module_list(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

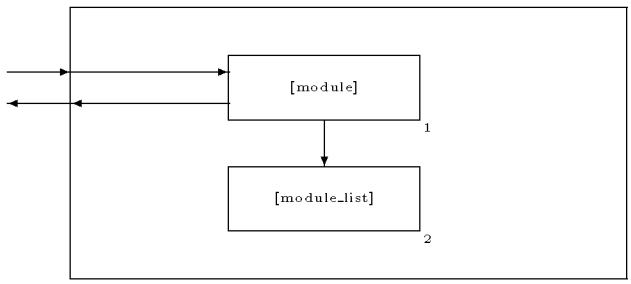
DXL

```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'
```

DXL – Production Rule – 06

module_list(2)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1)
y(0) = y(1)
top(1) = top(0)
top(2) = bottom(1) + GapY
bottom(0) = bottom(2)
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)

line(1) = get_line(1,[2])
```

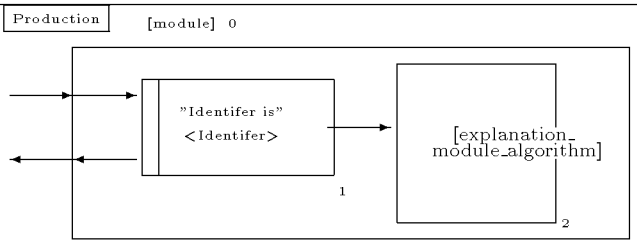
DXL

```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'
```

DXL – Production Rule – 07

module

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1)+w(1)+GapX
y(0) = y(1)
y(1) = y(2)
top(2) = top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( ["Identifier is", <identifier>"] )
cell(1) = "pre_defined_process"
string(1) = get_str( ["Identifier is", <identifier>"] )
lines(1) = get_line_begin(1,[2])
```

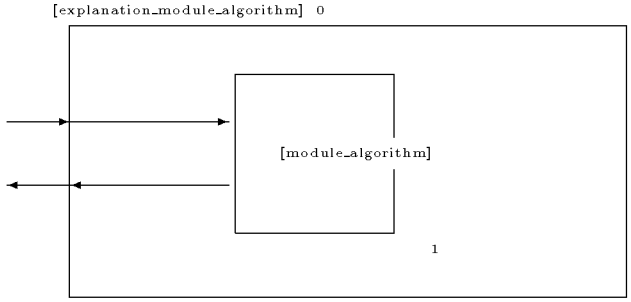
DXL

```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'
< モジュール識別句 > ::=
  'Identification'
  'Identifier' 'is' < 識別子 >
  [< 説明記述 >]
  'End_Identification' ';'
< モジュール論理句 > ::=
  'Module_Algorithm'
  < 文の列 >
  'End_Module_Algorithm' ';'
```

DXL – Production Rule – 08

explanation_module_algorithm(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

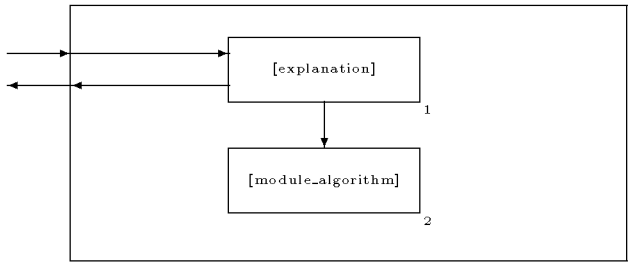
```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'

< モジュール識別句 > ::=
  'Identification'
  'Identifier' 'is' < 識別子 >
  [< 説明記述 >]
  'End_Identification' ';'

< モジュール論理句 > ::=
  'Module_Algorithm'
  < 文の列 >
  'End_Module_Algorithm' ';'
```

DXL – Production Rule – 09

explanation_module_algorithm(2)

Production

[explanation_module_algorithm] 0



Semantic Rules

```
x(1) = x(0)
x(2) = x(1)
y(0) = y(1)
top(1) = top(0)
top(2) = bottom(1)+GapY
bottom(0) = bottom(2)
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)
```

DXL

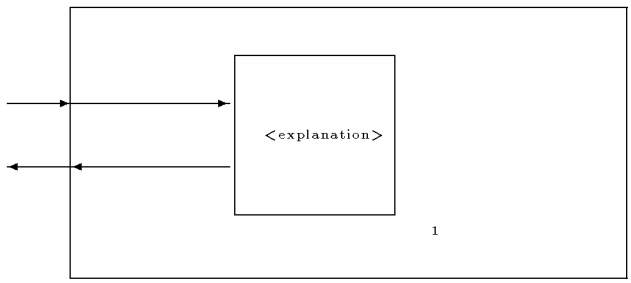
```
< モジュールパケット > ::=
  'M_Packet'
  < プロファイル句 >
  { < モジュール識別句 > < モジュール論理句 > }
  'End_M_Packet' ';'

< モジュール識別句 > ::=
  'Identification'
  'Identifier' 'is' < 識別子 >
  [< 説明記述 >]
  'End_Identification' ';'

< モジュール論理句 > ::=
  'Module_Algorithm'
  < 文の列 >
  'End_Module_Algorithm' ';'
```

DXL – Production Rule – 10

explanation

Production

[explanation] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1)=top(0)
bottom(0) = bottom(1)
bottom(1) = y(1) + h(1)
id(1) = id(0)
nc(0) = 1

w(1) = get_width( [<explanation>] )
h(1) = get_height( [<explanation>] )
cell(1) = "process"
string(1) = get_str( [<explanation>] )
```

DXL

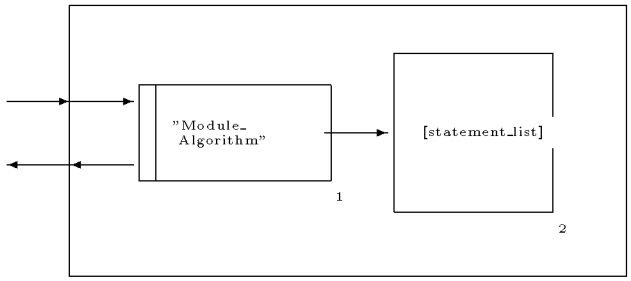
```
< プロファイル句 > ::=
  'Profile'
  [< 説明記述 >]
  'End_Profile' ';'
```

DXL – Production Rule – 11

module_algorithm

Production

[module_algorithm] 0



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height(["Module_Algorithm"])
cell(1) = "pre_defined_process"
string(1) = get_str(["Module_Algorithm"])
lines(1) = get_line_begin(1,[2])
```

DXL

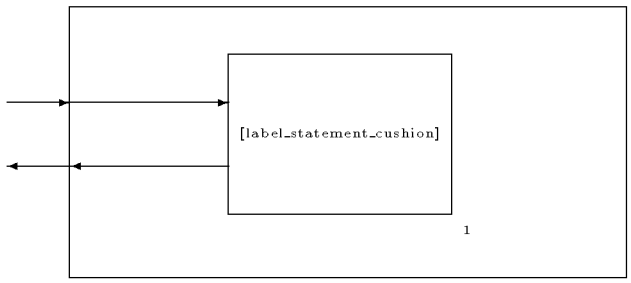
```
< モジュール論理句 > ::=
  'Module_Algorithm'
  < 文の列 >
  'End_Module_Algorithm' ';'
```

DXL – Production Rule – 12

statement_list(1)

Production

[statement_list] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

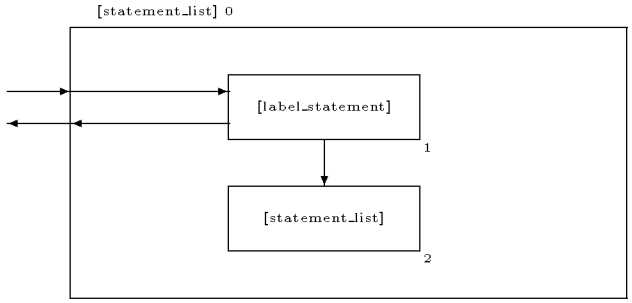
DXL

```
< 文の列 > ::=
  { {<< 名札 >> ":" } < 文 > }
```

DXL – Production Rule – 13

statement_list(2)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1)
y(0) = y(1)
top(1) = top(0)
top(2) = bottom(1) + GapY
bottom(0) = bottom(2)
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)

lines(1) = get_line(1,[2])
```

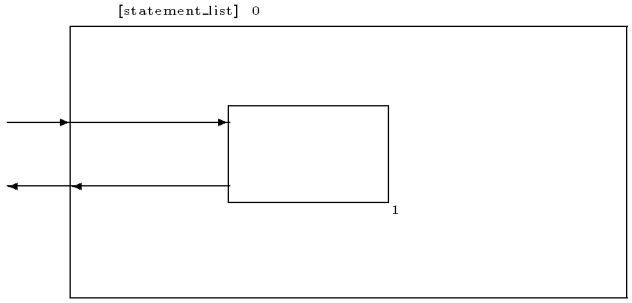
DXL

```
< 文の列 > ::=
  { { << 名札 >> ":" } < 文 > }
```

DXL – Production Rule – 14

statement_list(3)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
y(1) = top(0)
top(1) = top(0)
bottom(0) = bottom(1)
bottom(1) = y(1) + h(1)
id(1) = id(0)
nc(0) = 1

w(1) = MinW
h(1) = MinH
cell = "process"
string(1) = get_str( [""] )
```

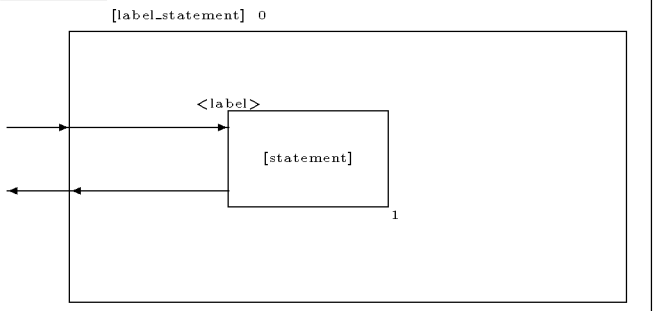
DXL

```
< 文の列 > ::=
  { { << 名札 >> ":" } < 文 > }
```

DXL – Production Rule – 15

label_statement(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
ll(1) = <label>
id(1) = id(0)
nc(0) = nc(1)
```

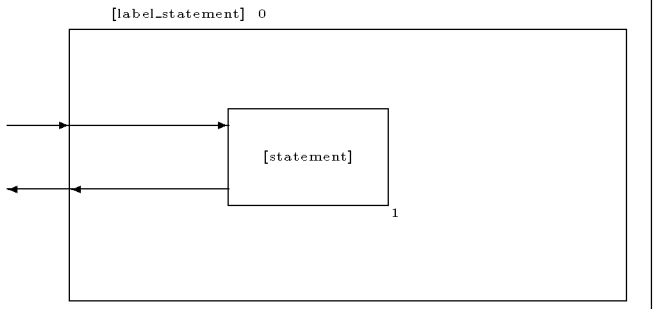
DXL

```
< 文の列 > ::=
  { { << 名札 >> ":" } < 文 > }
```

DXL – Production Rule – 16

label_statement(2)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

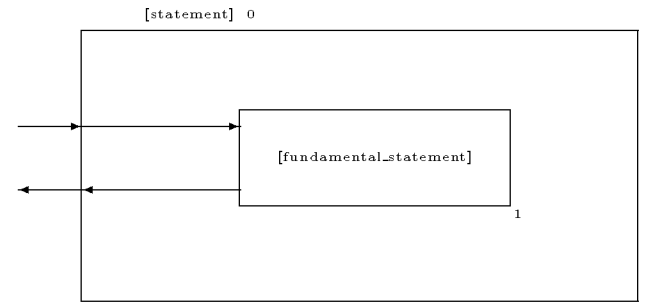
DXL

```
< 文の列 > ::=
  { { << 名札 >> ":" } < 文 > }
```

DXL – Production Rule – 17

statement(1)

Production



Semantic Rules

x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

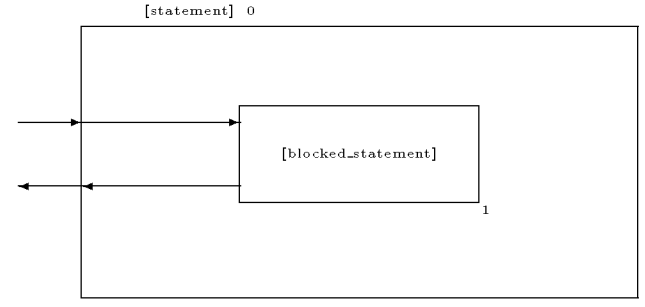
DXL

< 文 > ::=
< 基本文 > | < ブロック化文 > | < 打切り文 >

DXL – Production Rule – 18

statement(2)

Production



Semantic Rules

x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

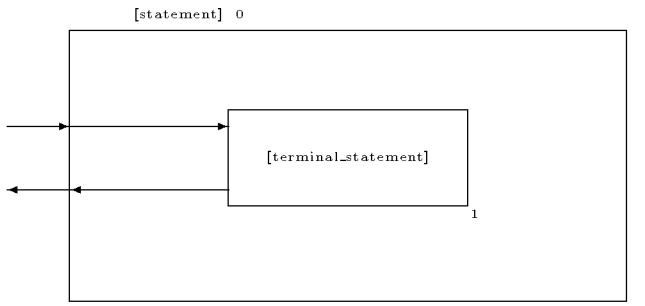
DXL

< 文 > ::=
< 基本文 > | < ブロック化文 > | < 打切り文 >

DXL – Production Rule – 19

statement(3)

Production



Semantic Rules

x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

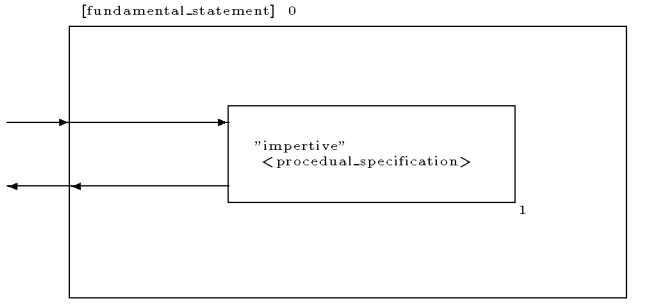
DXL

< 文 > ::=
< 基本文 > | < ブロック化文 > | < 打切り文 >

DXL – Production Rule – 20

fundamental_statement(1)

Production



Semantic Rules

x(1) = x(0)
y(0) = y(1)
y(1) = top(1)
y(0) = top(0)
bottom(0) = bottom(1)
bottom(1) = y(1) + h(1)
id(1) = id(0)
nc(0) = 1

w(1) =
get_width(["imperative", <procedual_specification>])
h(1) =
get_height(["imperative", <procedual_specification>])
cell(1) = "process"
string(1) =
get_str(["imperative", <procedual_specification>])

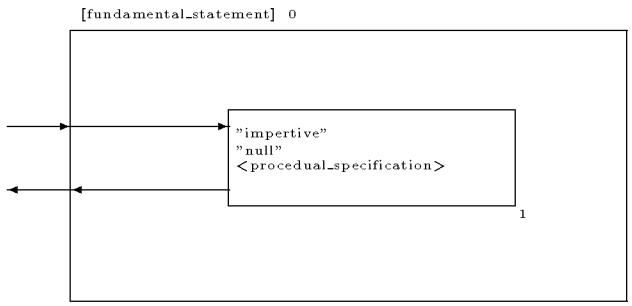
DXL

< 基本文 > ::=
'imperative'
[<< 基本文型 >>] < 手続き仕様記述 > ';'

DXL – Production Rule – 21

fundamental_statement(2)

Production



Semantic Rules

```
x(1) = x(0),y(0) = y(1)
y(1) = top(1),top(1) = top(0)
bottom(0) = bottom(1)
bottom(1) = y(1) + h(1)
id(1) = id(0),nc(0) = 1
w(1) = get_width( [ "imperative", "null",
    <procedual_specification> ] )
h(1) = get_height( [ "imperative", "null",
    <procedual_specification> ] )
cell(1) = "process"
string(1) = get_str( [ "imperative", "null",
    <procedual_specification> ] )
```

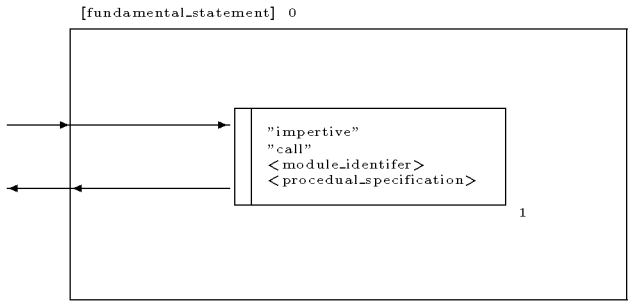
DXL

```
< 基本文 > ::=
  'imperative'
  [ << 基本文型 >> ] < 手続き仕様記述 > ';'
<< 基本文型 >> ::=
<<null>> | << モジュール呼び出し >> | <<goto>>
<<null>> ::= "null"
```

DXL – Production Rule – 22

fundamental_statement(3)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
y(1) = top(1)
top(1) = top(0)
bottom(0) = bottom(1)
bottom(1) = y(1) + h(1)
id(1) = id(0)
nc(0) = 1

w(1) =
  get_width( [ "imperative", "call",
    <identifier>, <procedual_specification> ] )
h(1) =
  get_height( [ "imperative", "call",
    <identifier>, <procedual_specification> ] )
cell(1) = "pre_defined_process"
string(1) =
  get_str( [ "imperative", "call",
    <identifier>, <procedual_specification> ] )
```

DXL

```
< 基本文 > ::=
  'imperative'
  [ << 基本文型 >> ] < 手続き仕様記述 > ';'
<< 基本文型 >> ::=
<<null>> | << モジュール呼び出し >> | <<goto>>
<< モジュール呼び出し >> ::= "call" << モジュール識別子 >>
```

DXL – Production Rule – 23

fundamental_statement(4)

Production

Semantic Rules

```
x(1) = x(0),y(0) = y(1)
top(1) = top(1)
bottom(0) = bottom(1)
bottom(1) = y(1) + h(1)
id(1) = id(0),nc(0) = 1
w(1) = get_width( [ "imperative", "goto", <label>,
    <procedual_specification> ] )
h(1) = get_height( [ "imperative", "goto", <label>,
    <procedual_specification> ] )
cell(1) = "pre_defined_process"
string(1) = get_str( [ "imperative", "goto", <label>,
    <procedual_specification> ] )
```

DXL

```
< 基本文 > ::=
  'imperative'
  [ << 基本文型 >> ] < 手続き仕様記述 > ';'
<< 基本文型 >> ::=
<<null>> | << モジュール呼び出し >> | <<goto>>
<<goto>> ::= "goto" << 名札 >>
```

DXL – Production Rule – 24

blocked_statement(1)

Production

Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([<block_identifier>])
cell(1) = "pre_defined_process"
string(1) = get_str([<block_identifier>])
lines(1) = get_line_begin(1,[2])
```

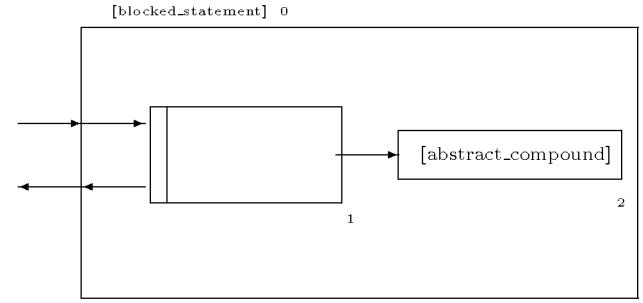
DXL

```
< ブロック化文 > ::=
  [< ブロック識別子 >]
  ["abstract" << ブロック仕様記述 >>]
  < 複合文 >
```

DXL – Production Rule – 25

blocked_statement(2)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "" ])
cell(1) = "pre_defined_process"
string(1) = get_str([ "" ])
lines(1) = get_line_begin(1,[2])
```

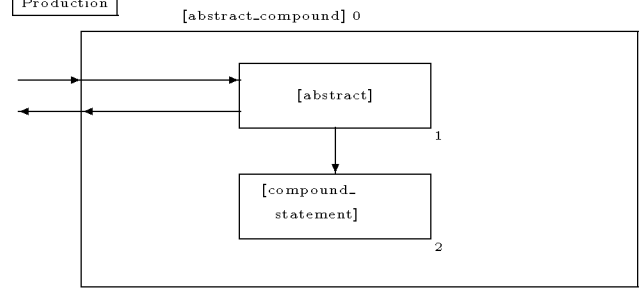
DXL

```
< ブロック化文 > ::=
  [< ブロック識別子 >]
  ["abstract" << ブロック仕様記述 >>]
  < 複合文 >
```

DXL – Production Rule – 26

abstruct_compound(1)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1)
y(0) = y(1)
top(1) = top(0)
top(2) + bottom(1) + GapY
bottom(0) = bottom(2)
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)

lines(1) = get_line(1, [2] )
```

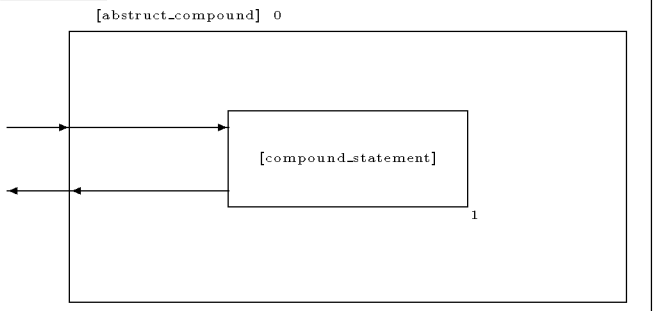
DXL

```
< ブロック化文 > ::=
  [< ブロック識別子 >]
  [ "abstruct" << ブロック仕様記述 >> ]
  < 複合文 >
```

DXL – Production Rule – 27

abstruct_compound(2)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

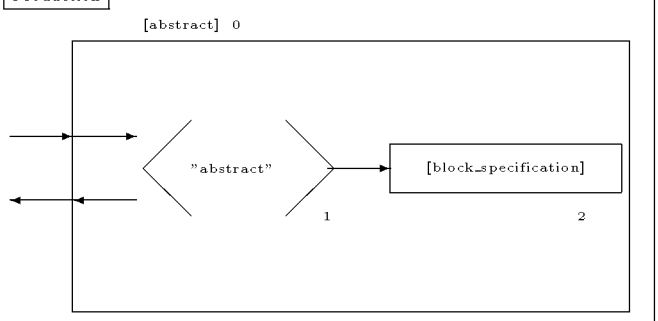
DXL

```
< ブロック化文 > ::=
  [< ブロック識別子 >]
  [ "abstruct" << ブロック仕様記述 >> ]
  < 複合文 >
```

DXL – Production Rule – 28

abstract

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
y(2) = top(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "abstract" ])
cell(1) = "caption"
string(1) = get_str([ "abstract" ])
lines(1) = get_line_begin(1,[2])
```

DXL

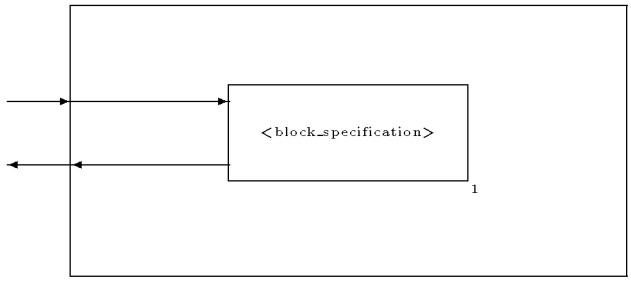
```
< ブロック化文 > ::=
  [< ブロック識別子 >]
  ["abstract" << ブロック仕様記述 >>]
  < 複合文 >
```

DXL – Production Rule – 29

block_specification

Production

[block_specification] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
y(1) = top(1)
top(1) = top(0)
bottom(0) = bottom(1)
bottom(1) = y(1) + h(1)
id(1) = id(0)
nc(0) = nc(1)

w(1) = get_width( [<block_specification>] )
h(1) = get_height( [<block_specification>] )
cell(1) = "process"
string(1) = get_str( [<block_specification>] )
```

DXL

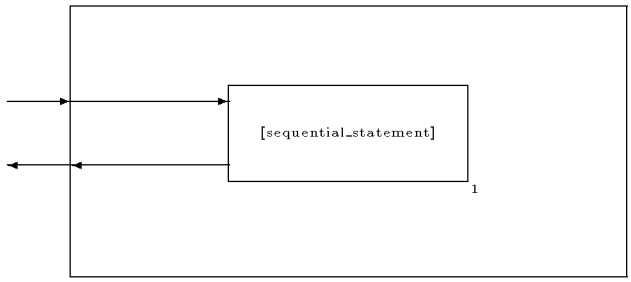
```
< ブロック化文 > ::=
  [< ブロック識別子 >]
  [ "abstruct" << ブロック仕様記述 >> ]
  < 複合文 >
```

DXL – Production Rule – 30

compound_statement(1)

Production

[compound_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

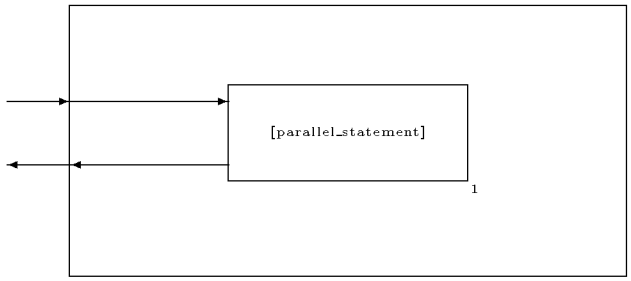
```
< 複合文 > ::=
  < 順次文 > | < 並列文 > | < 繰返し文 > | < 選択文 >
```

DXL – Production Rule – 31

compound_statement(2)

Production

[compound_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

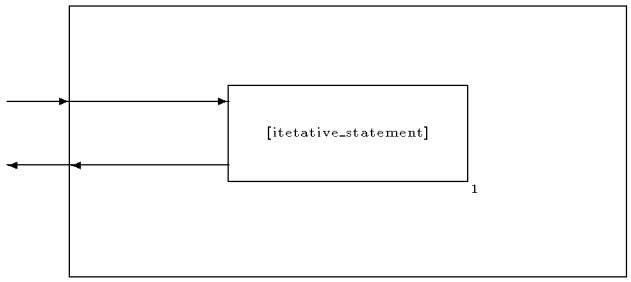
```
< 複合文 > ::=
  < 順次文 > | < 並列文 > | < 繰返し文 > | < 選択文 >
```

DXL – Production Rule – 32

compound_statement(3)

Production

[compound_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

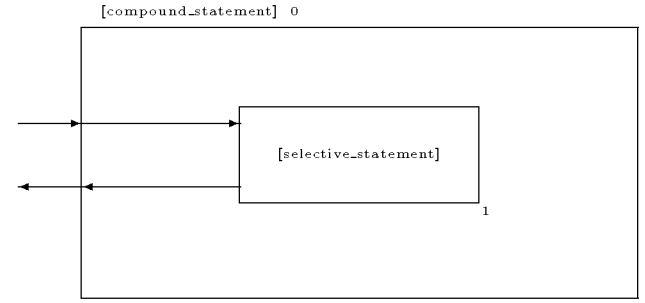
DXL

```
< 複合文 > ::=
  < 順次文 > | < 並列文 > | < 繰返し文 > | < 選択文 >
```

DXL – Production Rule – 33

compound_statement(4)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

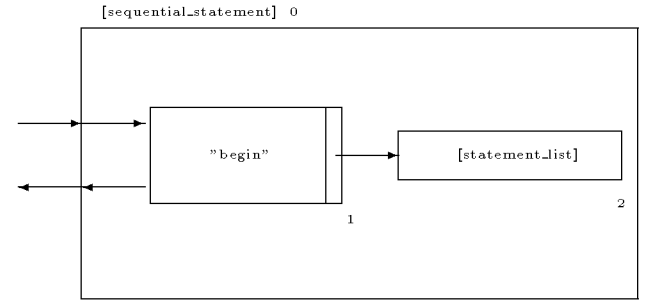
DXL

```
< 複合文 > ::=
  < 順次文 > | < 並列文 > | < 繰返し文 > | < 選択文 >
```

DXL – Production Rule – 34

sequential_statement

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "begin" ])
cell(1) = "continuous_iteration"
string(1) = get_str([ "begin" ])
lines(1) = get_line_begin(1,[2])
```

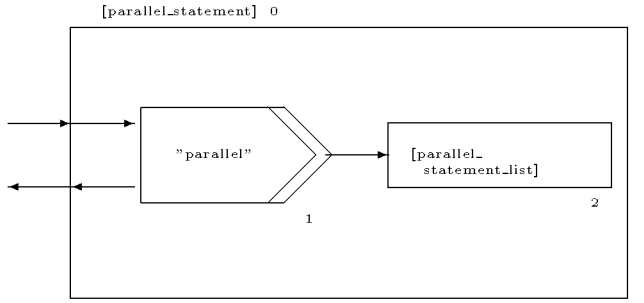
DXL

```
< 順次文 > ::=
  'begin'
  < 文の列 >
  'end' ';'
```

DXL – Production Rule – 35

parallel_statement

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "parallel" ])
cell(1) = "parallel_asynch"
string(1) = get_str([ "parallel" ])
lines(1) = get_line_begin(1,[2])
```

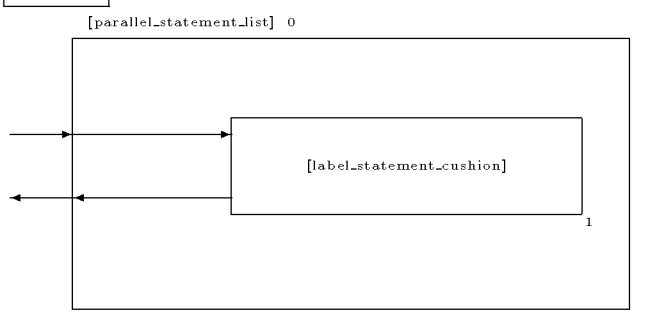
DXL

```
< 並列文 > ::=
  'parallel'
  < 文の列 >
  'end_parallel' ';'
```

DXL – Production Rule – 36

parallel_statement_list(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

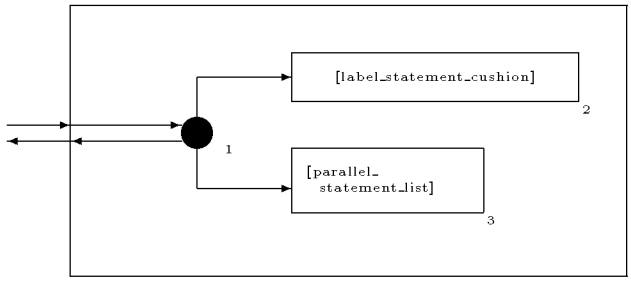
```
< 並列文 > ::=
  'parallel'
  < 文の列 >
  'end_parallel' ';'
```

DXL – Production Rule – 37

parallel_statement_list(2)

Production

[parallel_statement_list] 0



Semantic Rules

x(2) = x(0)
x(3) = x(2)
y(0) = (y(2) + y(3)) / 2
top(2) = top(0)
top(3) = bottom(2) + GapY
bottom(0) = bottom(3)
id(2) = id(0)
id(3) = id(2) + nc(2)
nc(0) = nc(2) + nc(3)

DXL

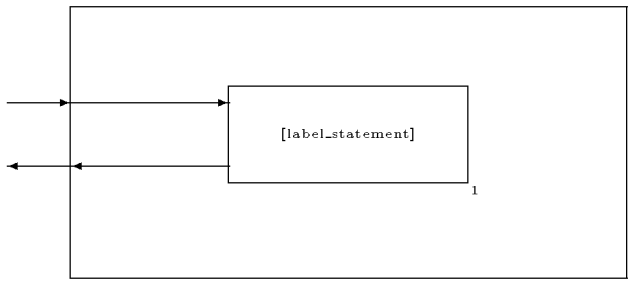
< 並列文 > ::=
'parallel'
< 文の列 >
'end_parallel' ';'

DXL – Production Rule – 38

label_statement_cushion(1)

Production

[label_statement_cushion] 0



Semantic Rules

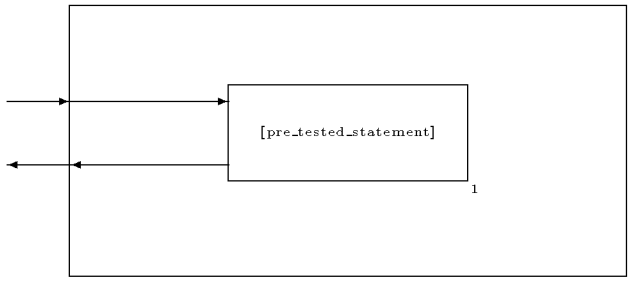
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

DXL – Production Rule – 39

iterative_statement(1)

Production

[iterative_statement] 0



Semantic Rules

x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

DXL

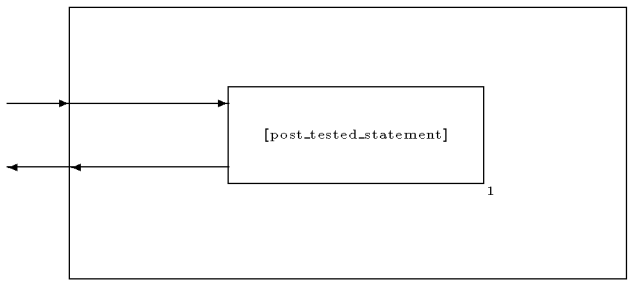
< 繰返し文 > ::=
< 前判定繰返し文 > | < 後判定繰返し文 > |
< 継続繰返し文 >

DXL – Production Rule – 40

iterative_statement(2)

Production

[iterative_statement] 0



Semantic Rules

x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

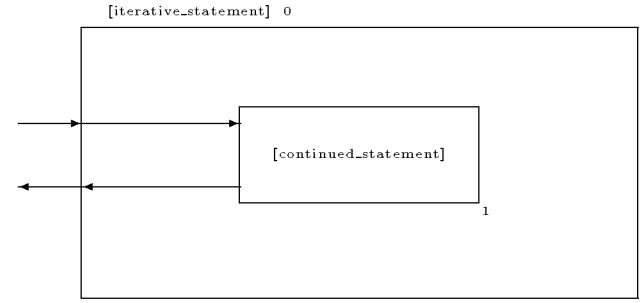
DXL

< 繰返し文 > ::=
< 前判定繰返し文 > | < 後判定繰返し文 > |
< 継続繰返し文 >

DXL – Production Rule – 41

iterative_statement(3)

Production



Semantic Rules

```
x(1) = x(0),y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0),nc(0) = nc(1)
```

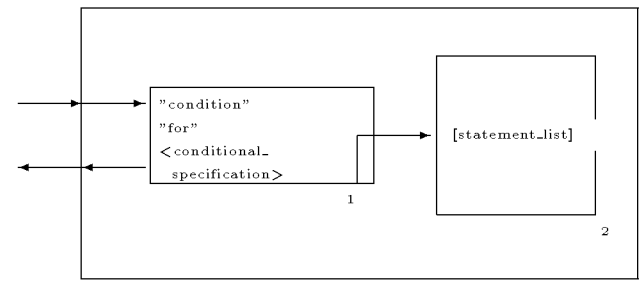
DXL

```
< 繰返し文 > ::=
  < 前判定繰返し文 > | < 後判定繰返し文 > |
    < 継続繰返し文 >
```

DXL – Production Rule – 42

pre_tested_statement(1)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( [ "condition", "for",
    <conditional_specification> ] )
cell(1) = "pre-tested_iteration"
string(1) =
 get_str( [ "condition", "for", <conditional_specification> ] )
lines(1) = get_line_begin(1,[2])
```

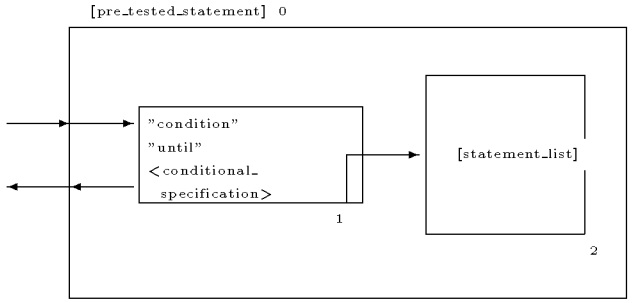
DXL

```
< 前判定繰返し文 > ::=
  'condition'
  [<< 前判定繰返し型 >>]
  < 条件仕様記述 >
  'loop'
  < 文の列 >
  'end_loop' ';'
<< 前判定繰返し型 >> ::= "while" | "until" | "for"
```

DXL – Production Rule – 43

pre_tested_statement(2)

Production



Semantic Rules

```
x(1) = x(0),x(2) = x(1) + w(1) + GapX
y(0) = y(1),y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0),id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( [ "condition", "until",
  <conditional_specification> ] )
cell(1) = "pre-tested_iteration"
string(1) =
    get_str( [ "condition", "until", <conditional_specification> ] )
lines(1) = get_line_begin(1,[2])
```

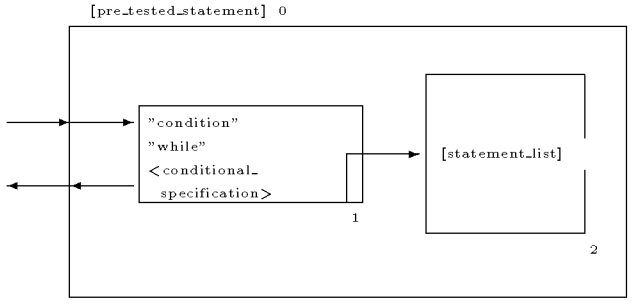
DXL

```
< 前判定繰返し文 > ::=
  'condition'
  [<< 前判定繰返し型 >>]
  < 条件仕様記述 >
  'loop'
  < 文の列 >
  'end_loop' ';'
<< 前判定繰返し型 >> ::= "while" | "until" | "for"
```

DXL – Production Rule – 44

pre_tested_statement(3)

Production



Semantic Rules

```
x(1) = x(0),x(2) = x(1) + w(1) + GapX
y(0) = y(1),y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0),id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( [ "condition", "while",
    <conditional_specification> ] )
cell(1) = "pre-tested_iteration"
string(1) = get_str( [ "condition", "while",
    <conditional_specification> ] )
lines(1) = get_line_begin(1,[2])
```

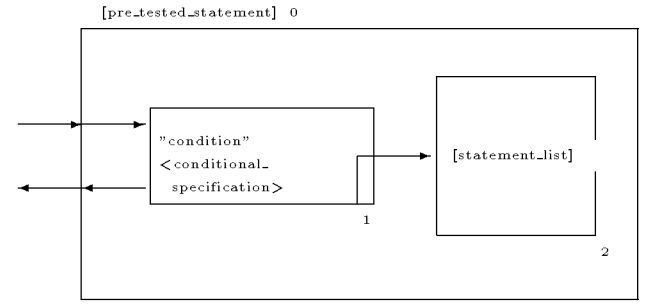
DXL

```
< 前判定繰返し文 > ::=
  'condition'
  [<< 前判定繰返し型 >>]
  < 条件仕様記述 >
  'loop'
  < 文の列 >
  'end_loop' ';'
<< 前判定繰返し型 >> ::= "while" | "until" | "for"
```

DXL – Production Rule – 45

pre_tested_statement(4)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( [ "condition", <conditional_specification> ] )
cell(1) = "pre-tested_iteration"
string(1) = get_str( [ "condition", <conditional_specification> ] )
lines(1) = get_line_begin(1,[2])
```

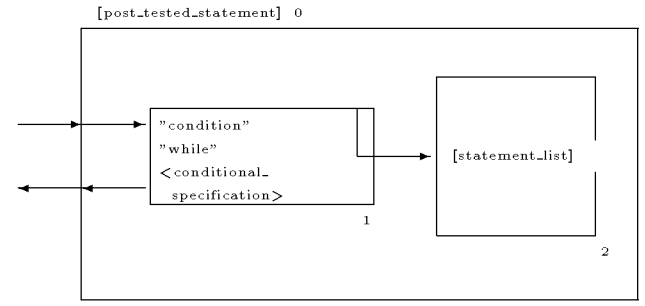
DXL

```
< 前判定繰返し文 > ::=
  'condition'
  [<< 前判定繰返し型 >>]
  < 条件仕様記述 >
  'loop'
  < 文の列 >
  'end_loop' ';'
<< 前判定繰返し型 >> ::= "while" | "until" | "for"
```

DXL – Production Rule – 46

post_tested_statement(1)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( [ "condition", "while",
    <conditional_specification> ] )
cell(1) = "post-tested_iteration"
string(1) = get_str( [ "condition", "while",
    <conditional_specification> ] )
lines(1) = get_line_begin(1,[2])
```

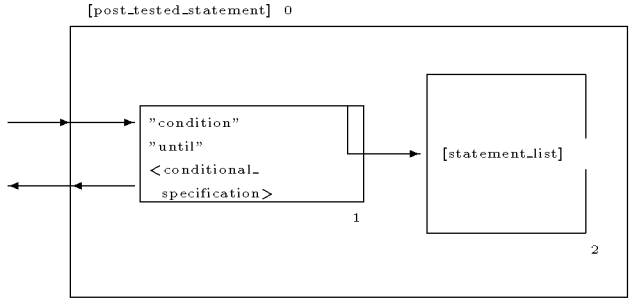
DXL

```
< 後判定繰返し文 > ::=
  'loop' < 文の列 > 'end_loop'
  'condition'
  [<< 後判定繰返し型 >>]
  < 条件仕様記述 > ';'

<< 後判定繰返し型 >> ::= "while" | "until"
```

DXL – Production Rule – 47

post_tested_statement(2)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height( [ "condition", "until",
    <conditional_specification> ] )
cell(1) = "post-tested_iteration"
string(1) = get_str( [ "condition", "until",
    <conditional_specification> ] )
lines(1) = get_line_begin(1,[2])
```

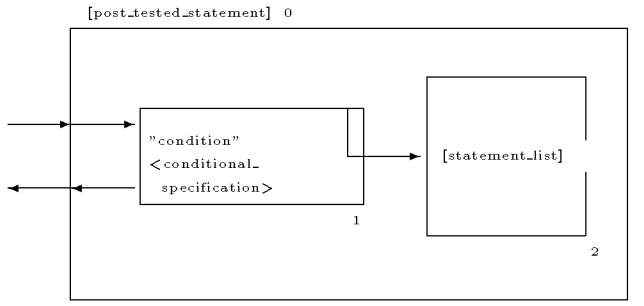
DXL

```
< 後判定繰返し文 > ::=
  'loop' < 文の列 > 'end_loop'
  'condition'
  [<< 後判定繰返し型 >>]
  < 条件仕様記述 > ';'

<< 後判定繰返し型 >> ::= "while" | "until"
```

DXL – Production Rule – 48

post_tested_statement(3)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) =
  get_height( [ "condition", <conditional_specification> ] )
cell(1) = "post-tested_iteration"
string(1) =
  get_str( [ "condition", <conditional_specification> ] )
lines(1) = get_line_begin(1,[2])
```

DXL

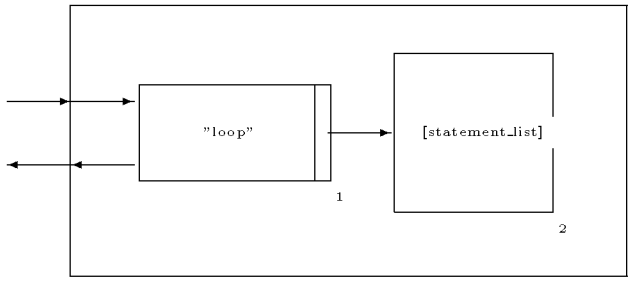
```
< 後判定繰返し文 > ::=
  'loop' < 文の列 > 'end_loop'
  'condition'
  [<< 後判定繰返し型 >>]
  < 条件仕様記述 > ';'

<< 後判定繰返し型 >> ::= "while" | "until"
```

DXL – Production Rule – 49

continued_statement

Production

[continued_statement] 0



Semantic Rules

```
x(1) = x(0),x(2) = x(1) + w(1) + GapX
y(0) = y(1),y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0),id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "loop" ])
cell(1) = "continuous_iteration"
string(1) = get_str([ "loop" ])
lines(1) = get_line_begin(1,[2])
```

DXL

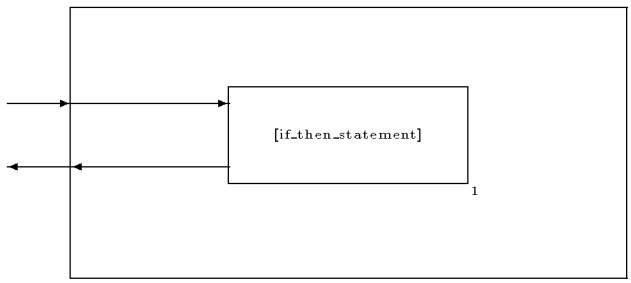
```
< 継続繰返し文 > ::=
  'loop'
  < 文の列 >
  'end_loop' ';'
```

DXL – Production Rule – 50

selective_statement(1)

Production

[selective_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

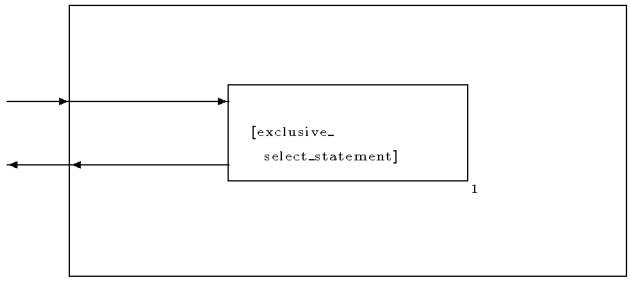
```
< 選択文 > ::=
  < 単岐選択文 > | < 多岐選択文 > | < 多重岐選択文 >
```

DXL – Production Rule – 51

selective_statement(2)

Production

[selective_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

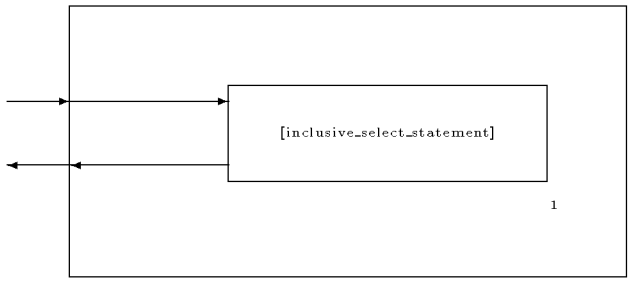
```
< 選択文 > ::=
  < 単岐選択文 > | < 多岐選択文 > | < 多重岐選択文 >
```

DXL – Production Rule – 52

selective_statement(3)

Production

[selective_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

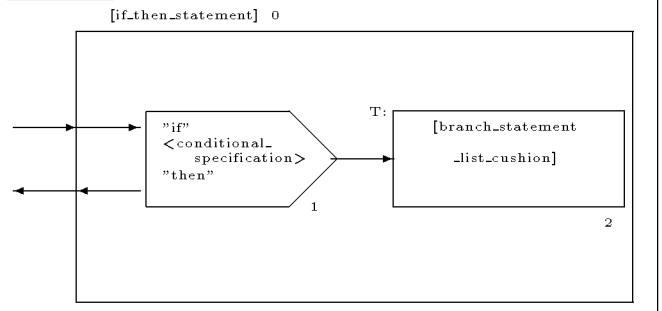
DXL

```
< 選択文 > ::=
  < 単岐選択文 > | < 多岐選択文 > | < 多重岐選択文 >
```

DXL – Production Rule – 53

if_then_statement

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(1) = top(0)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
cl(2) = "T:" id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "if",
  <conditional_specification>, "then" ])
cell(1) = "exclusive_selection"
string(1) = get_str([ "if",
  <conditional_specification>, "then" ])
lines(1) = get_line_begin(1,[2])
```

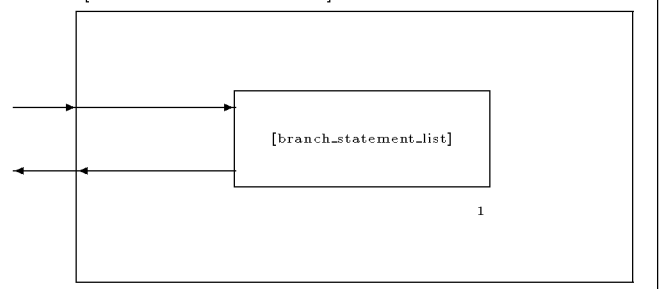
DXL

```
< 単岐選択文 > ::= 'if' < 条件仕様記述 > 'then'
  < 文の列 >
  'end_if' ';'
```

DXL – Production Rule – 54

branch_statement_list_cushion

Production



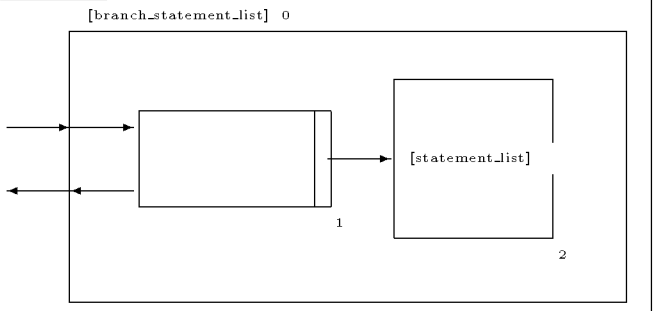
Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL – Production Rule – 55

branch_statement_list

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = MinH
cell(1) = "continuous_iteration"
string(1) = get_str([ "" ])
lines(1) = get_line_begin(1,[2])
```

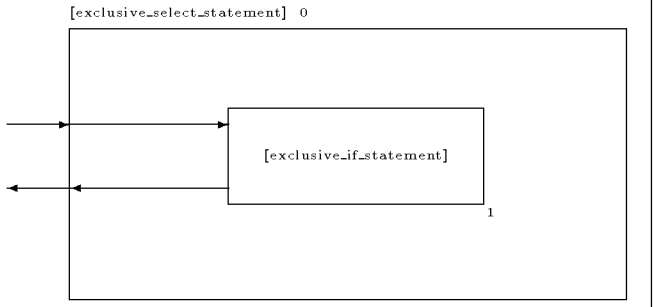
DXL

```
< 文の列 > ::=
  { { << 名札 >> ";" } < 文 > }
```

DXL – Production Rule – 56

exclusive_select_statement(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

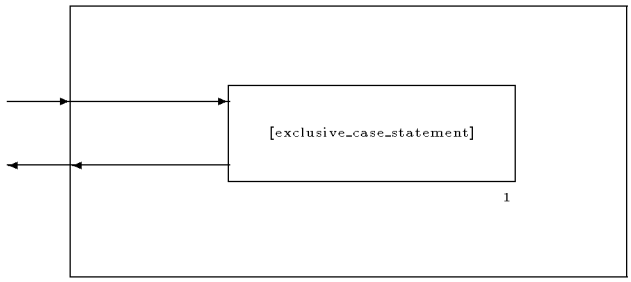
```
< 多岐選択文 > ::=
  <if 型選択文 > | <case 型選択文 >
```

DXL – Production Rule – 57

exclusive_select_statement(2)

Production

[exclusive_select_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

DXL

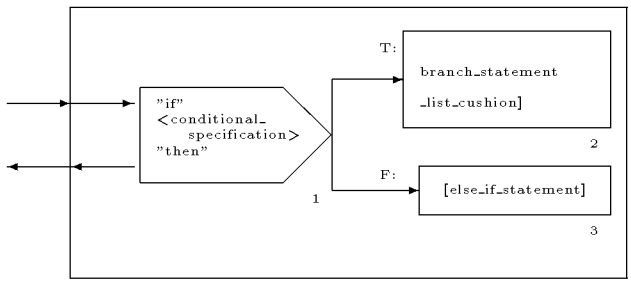
```
< 多岐選択文 > ::=
  <if 型選択文 > | <case 型選択文 >
```

DXL – Production Rule – 58

exclusive_if_statement(1)

Production

[exclusive_if_statement] 0



Semantic Rules

```
x(1) = x(0)
x(2) = x(0) + w(1) + GapX
x(3) = x(0) + w(1) + GapX
y(0) = (y(2) + y(3)) / 2
top(2) = top(0)
top(3) = bottom(2) + GapY
bottom(0) = max( bottom(1), bottom(3) )
cl(2) = "T:"
cl(3) = "F:"
id(1) = id(0)
id(2) = id(0) + 1
id(3) = id(0) + nc(2) +1
nc(0) = nc(2) + nc(3) + 1

w(1) = MinW
h(1) = get_height([ "if", <conditional_specification>,
          "then" ])
cell(1) = "exclusive_selection"
string(1) = get_str([ "if", <conditional_specification>,
          "then" ])
lines(1) = get_line_begin(1,[2,3])
```

DXL

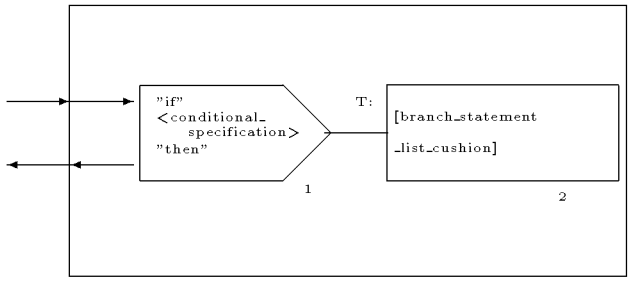
```
<if 型選択文 > ::= 'exclusive_select'
  'if' < 条件仕様記述 > 'then' < 文の列 >
  { 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
  [ 'else' < 文の列 > ]
  'end_exclusive_select' ';'
```

DXL – Production Rule – 59

exclusive_if_statement(2)

Production

[exclusive_if_statement] 0



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(0) = y(2)
top(1)=top(0)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
cl(2) = "T:"
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) =
  get_height([ "if", <conditional_specification>, "then" ])
cell(1) = "exclusive_selection"
string(1) =
  get_str([ "if", <conditional_specification>, "then" ])
lines(1) = get_line_begin(1,[2])
```

DXL

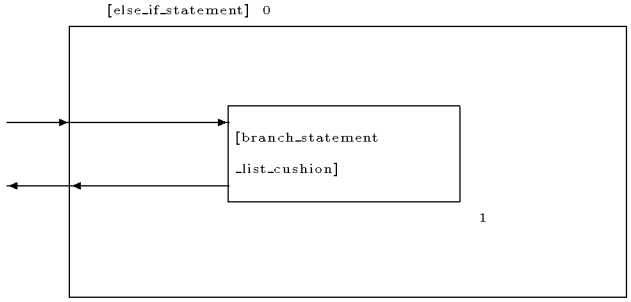
```
<if 型選択文 > ::= 'exclusive_select'
  'if' < 条件仕様記述 > 'then' < 文の列 >
  { 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
  [ 'else' < 文の列 > ]
  'end_exclusive_select' ';'
```

DXL – Production Rule – 60

else_if_statement(1)

Production

[else_if_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)
```

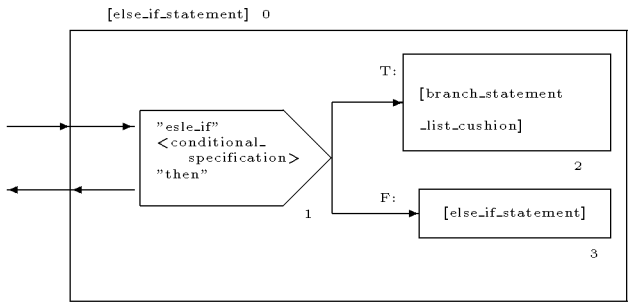
DXL

```
<if 型選択文 > ::= 'exclusive_select'
  'if' < 条件仕様記述 > 'then' < 文の列 >
  { 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
  [ 'else' < 文の列 > ]
  'end_exclusive_select' ';'
```

DXL – Production Rule – 61

else_if_statement(2)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(0) + w(1) + GapX
x(3) = x(0) + w(1) + GapX
y(0) = (y(2) + y(3)) / 2
top(2) = top(0)
top(3) = bottom(2) + GapY
bottom(0) = max( bottom(1), bottom(3) )
cl(2) = "T:"
cl(3) = "F:"
id(1) = id(0)
id(2) = id(0) + 1
id(3) = id(0) + nc(2) +1
nc(0) = nc(2) + nc(3) + 1

w(1) = MinW
h(1) =
  get_height([ "if", <conditional_specification>, "then" ])
cell(1) = "exclusive_selection"
string(1) =
  get_str([ "if", <conditional_specification>, "then" ])
lines(1) = get_line_begin(1,[2,3])
```

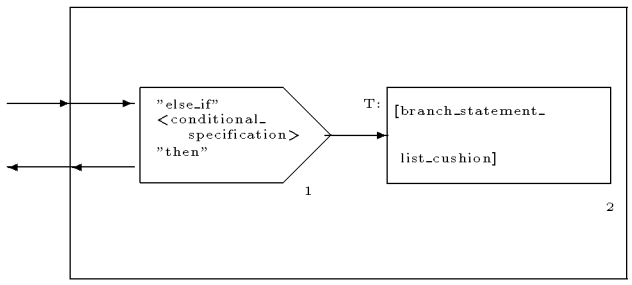
DXL

```
<if 型選択文 >  ::= 'exclusive_select'
  'if' < 条件仕様記述 > 'then' < 文の列 >
  { 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
  [ 'else' < 文の列 > ]
  'end_exclusive_select' ';'
```

DXL – Production Rule – 62

else_if_statement(3)

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(0) + w(1) + GapX
y(0) = y(1)
y(1) = y(2)
top(1)=top(0)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
bottom(1) = y(1) + h(1)
cl(2) = "T:"
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) =
  get_height([ "else_if", <conditional_specification>, "then" ])
cell(1) = "exclusive_selection"
string(1) =
  get_str([ "else_if", <conditional_specification>, "then" ])
lines(1) = get_line_begin(1,[2])
```

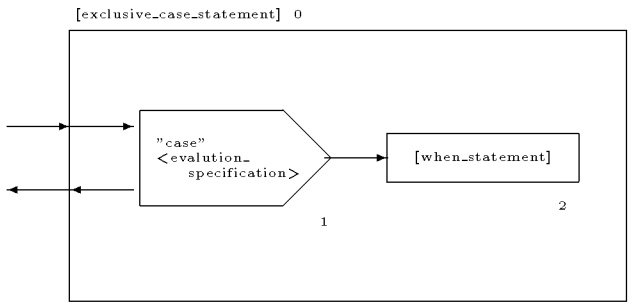
DXL

```
<if 型選択文 >  ::= 'exclusive_select'
  'if' < 条件仕様記述 > 'then' < 文の列 >
  { 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
  [ 'else' < 文の列 > ]
  'end_exclusive_select' ';'
```

DXL – Production Rule – 63

exclusive_case_statement

Production



Semantic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(0) = y(2)
top(1)=top(0)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
id(1) = id(0)
id(2) = id(1) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "case", <evalution_specification> ])
cell(1) = "exclusive_selection"
string(1) = get_str([ "case", <evalution_specification> ])
lines(1) = get_line_begin(1,[2])
```

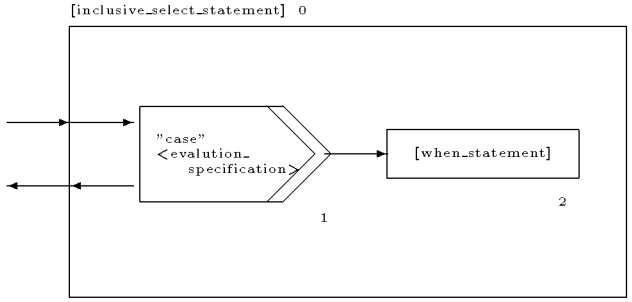
DXL

```
<case 型選択文 >  ::= 'exclusive_select'
  'case' < 評価式仕様記述 >
  { 'when' < 選択値仕様記述 > ':' < 文の列 > }
  'end_exclusive_select' ';'
```

DXL – Production Rule – 64

inclusive_select_statement

Production



Sentic Rules

```
x(1) = x(0)
x(2) = x(1) + w(1) + GapX
y(0) = y(1)
y(0) = y(2)
top(1)=top(0)
top(2)=top(0)
bottom(0) = max( bottom(1), bottom(2) )
id(1) = id(0)
id(2) = id(0) + 1
nc(0) = nc(2) + 1

w(1) = MinW
h(1) = get_height([ "case", <evalution_specification> ])
cell(1) = "parallel_asynch"
string(1) = get_str([ "case", <evalution_specification> ])
lines(1) = get_line_begin(1,[2])
```

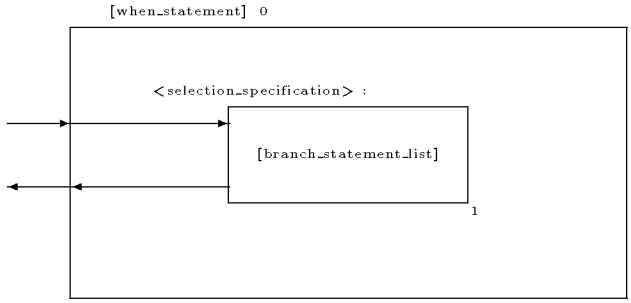
DXL

```
< 多重吸選択文 >  ::= 'inclusive_select'
  'case' < 評価式仕様記述 >
  { 'when' < 選択値仕様記述 > ':' < 文の列 > }
  'end_inclusive_select' ';'
```

DXL – Production Rule – 65

when_statement(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
cl(1) = <selection_specification> ":"
id(1) = id(0)
nc(0) = nc(1)
```

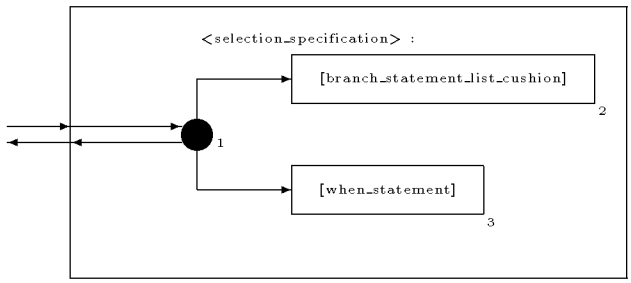
DXL

```
<case 型選択文 > ::= 'exclusive_select'
  'case' < 評価式仕様記述 >
  { 'when' < 選択値仕様記述 > ':' < 文の列 > }
  'end_exclusive_select' ';'

< 多重岐選択文 > ::= 'inclusive_select'
  'case' < 評価式仕様記述 >
  { 'when' < 選択値仕様記述 > ':' < 文の列 > }
  'end_inclusive_select' ';'
```

DXL – Production Rule – 66

when_statement(2)

Production



Semantic Rules

```
x(2) = x(0)
x(3) = x(0)
y(0) = (y(2) + y(3)) / 2
top(2) = top(0)
top(3) = bottom(2) + GapY
bottom(0) = bottom(3)
cl(2) = <selection_specification> ":"
id(2) = id(0)
id(3) = id(2) + nc(2)
nc(0) = nc(2) + nc(3)
```

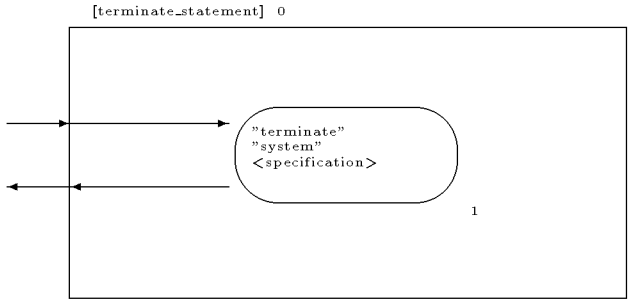
DXL

```
<case 型選択文 > ::= 'exclusive_select'
  'case' < 評価式仕様記述 >
  { 'when' < 選択値仕様記述 > ':' < 文の列 > }
  'end_exclusive_select' ';'

< 多重岐選択文 > ::= 'inclusive_select'
  'case' < 評価式仕様記述 >
  { 'when' < 選択値仕様記述 > ':' < 文の列 > }
  'end_inclusive_select' ';'
```

DXL – Production Rule – 67

terminate_statement(1)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

w(1) =
  get_width( [ "terminate", "system", <specification> ] )
h(1) =
  get_height( [ "terminate", "system", <specification> ] )
cell(1) = "terminal"
string(1) =
  get_str( [ "terminate", "system", <specification> ] )
```

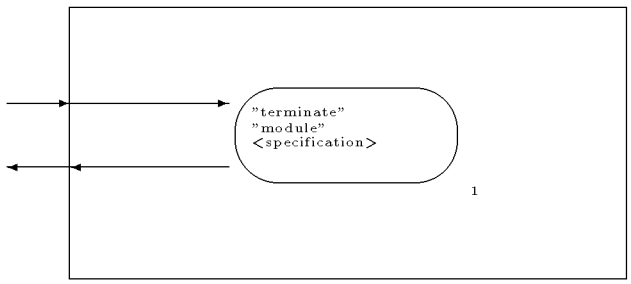
DXL

```
< 打切り文 > ::=
  'terminate' < 打切り対象 > < 仕様記述 > ';'
< 打切り対象 > ::=
  'system' | 'module' | 'block'< ブロック識別子 >
```

DXL – Production Rule – 68

terminate_statement(2)

Production



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

w(1) =
  get_width( [ "terminate", "module", <specification> ] )
h(1) =
  get_height( [ "terminate", ",module", <specification> ] )
cell(1) = "terminal"
string(1) =
  get_str( [ "terminate", "module", <specification> ] )
```

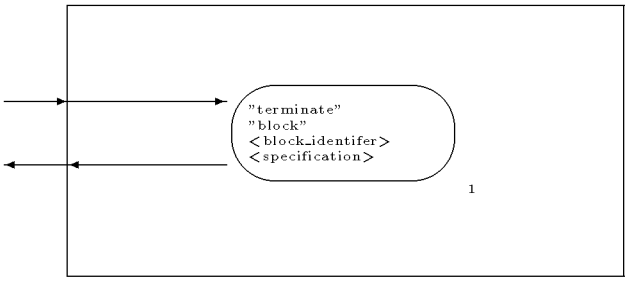
DXL

```
< 打切り文 > ::=
  'terminate' < 打切り対象 > < 仕様記述 > ';'
< 打切り対象 > ::=
  'system' | 'module' | 'block'< ブロック識別子 >
```

terminate_statement(3)

Production

[terminate_statement] 0



Semantic Rules

```
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

w(1) = get_width( [ "terminate", "block",
  <block_identifier>, <specification> ] )
h(1) = get_height( [ "terminate", "block",
  <block_identifier>, <specification> ] )
cell(1) = "terminal"
string(1) = get_str( [ "terminate", "block",
  <block_identifier>, <specification> ] )
```

DXL

```
< 打切り文 > ::= 'terminate'
< 打切り対象 > < 仕様記述 > ';'
< 打切り対象 > ::= 'system'
| 'module' | 'block'< ブロック識別子 >
```