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１１年度 修士論文

DXL対応プログラム図言語に対する属性グラフ文法

指導教員　夜久竹夫教授

6198M13　宮崎征宏

２０００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(Diagram eXchange Language)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1996年に属性文脈自由グラフ文法により定式化された[12]. [12]では単純な描画に対応した描画条件とVignaの文脈自由グラフ文法にその描画条件を加えた属性グラフ文法が与えられた.

本論文では[12]の描画条件を変形して図としての正誤を表す描画条件を定める. また, 最近普遍的モデルになったNCEグラフ文法によりDXL対応Hichartプログラム図を定式化する属性NCEグラフ文法を構成し, その文法が定める図が新しい描画条件を満たすことを示す. 今後は本論文の結果がJIS X 0130関連文献により参照される.

KeyWords

DXL,Hichart, 文脈自由グラフ文法,NCEグラフ文法,
属性グラフ文法, プログラム図, 描画条件

目 次

1 序章

プログラムの図表示は,視覚的プログラミングツールの実現やソフトウェア事例データベースのプログラム情報の視覚化のために不可欠な機能である.

例えば,プログラム開発の際にプログラムの階層木構造がそのまま図表示に反映されていれば,全体構造が即座に把握できるだけでなく,部分的な修正,トレースも容易に行うことが可能となる.

また,ソフトウェア事例データベースからプログラムを参照あるいは再利用する際にも,プログラムを図表示することによってプログラムの視認性と解読性が向上し検索や利用を容易にする.

そこで近年,それぞれの特徴を持つ多様なプログラム図式の記述言語が報告されており,それらに基づいた多くの 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[3](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式言語とは異なった特徴を持っている.そして現在まで,幾つかの大学でプログラム言語の教育と研究のために Hichart 処理システムの開発が行われてきている.

一方,木構造図データ交換言語 DXL(Diagram eXchange Language for tree structured charts) は,多様な木構造図を用いた CASE ツール間で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.それぞれのプログラム図式言語に対して 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式言語に基づいた CASE ツール間でデータを相互に変換して利用できるようになる.

DXL に対応した Hichart 図の生成規則は既に基底文脈自由グラフ文法を Vigna の CFGG[2] によるものとする属性グラフ文法に基づいて簡易的な描画規則とともに定式化されており,また,この DXL 対応 Hichart 属性グラフ文法は,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.

しかし,基底自由文脈自由文法をより一般的な NCE グラフ文法に改めて新しい形式に定式化することにより,表やシミュレータ等の他の処理系と統一的に扱うことが可能になる.

本論文では DXL 対応 Hichart に対し,NCE 文法により厳密な文法を与え,定式化し,前述の目標を目指す.また,描画条件はプログラム図としての正誤を評価するように変更する.2 章では Hichart と DXL,[12] で用いられた文脈自由グラフ文法と今回導入する NCE 文法 [7] で用いられた描画条件を解説する.3 章では,今回新たに定めた描画条件を述べる.4 章では,今回用いる属性 NCE グラフ文法と DXL 対応グラフ文法ついて述べ,これが定めた描画条件を満たし,DXL のすべての構文に対応することを示す.

2 準備

2.1 Hichart[3]

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.その後改良と拡張 [9] 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.

Hichart はプログラム自身の階層木構造を直接反映している.したがって,Hichart 図として表示することによりプログラムの全体構造が即座に把握可能となり,可読性や視認性が飛躍的に向上する.また,Hichart 図では制御線と実際の制御の流れが一致しているので,プログラムの詳細な記述,トレース,修正が容易である.Hichart に用いられる記号を Hichart 記号または「セル」と呼ぶ.図 1 はプログラム「ハノイの塔」を Hichart を用いて表示したものである.

2.2 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,それらに基づいた多くの CASE ツールが開発されている.木構造図データ交換言語 DXL は,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

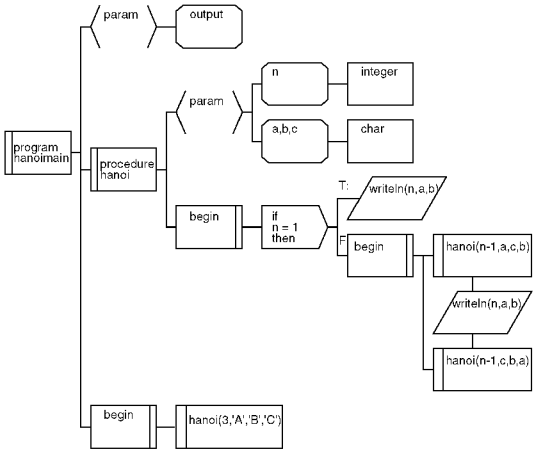


図 1: Hichart の例

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.

2.3 グラフ文法

ここでは,Hichart のグラフ文法 (Hichart グラフ文法) のために必要な概念について解説する. 従来の Hichart[6] は属性グラフ文法によって記述されている. 最初に 2.3.1 でグラフ言語に関する定義を述べ, 次に 2.3.2 で [12] で用いられた Vigna[2] の文脈自由グラフ文法,2.3.3 で属性グラフ文法, そして 2.3.4 で今回新たに導入する NCE グラフ文法について述べる.

2.3.1 Vigna の文脈自由グラフ文法 [2]

定義 [2] Σ(頂点 アルファベット) 上の グラフ とは,3 項組 $H = (V, E, \varphi)$ である. ただし,

1. V は頂点の空でない集合.
2. $E \subseteq V \times V$ は辺の集合.
3. $\varphi : V \to \Sigma$ は 頂点ラベル付け関数という. $^{\exists}v \in V$, $^{\exists}x \in \Sigma$ で $x = \varphi(v)$ となるとき,v は x とラベル付けされた といい,x を v の ラベル という.

$\mathcal{G} = \{D \mid D$ は Σ 上のグラフ $\}$ とする.$\mathcal{D} \subseteq \mathcal{G}$ を Σ 上の グラフ言語 という.

$G = (V_G, E_G, \varphi_G), H = (V_H, E_H, \varphi_H)$ を Σ 上のグラフとする. 次を満たす equivalence 関数

$$e : V_G \to V_H$$

1. $\varphi_G(n) = \varphi_H(e(n)),^{\forall} n \in V_G$,
2. $(n_1, n_2) \in E_G \Leftrightarrow (e(n_1)), e(n_2)) \in E_H$

が存在するとき,G は H と 同形($G \equiv H$) であるという.

2.3.2 文脈自由グラフ文法 [2]

定義 [2] Vigna[2] の 文脈自由グラフ文法(context-free graph grammar)(CFGG) とは 5 項組 $GG = (\Sigma_n, \Sigma_t, S, D_o, R)$ である. ただし,

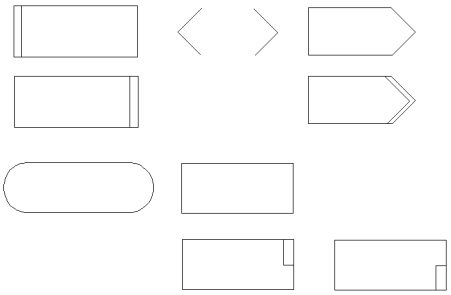
1. Σ_n は 非終端 頂点 アルファベット
2. Σ_t は 終端 頂点 アルファベット
3. $S \in \Sigma_n$ は開始ラベル
4. $D_0 = (V_0, E_0, \varphi_0)$ は開始グラフ. ただし, $V_0 = \{v_0\}, E_0 = \phi, \varphi_0(v_0) = S$
5. R は生成規則の集合.R の要素は 4 項組 $r =$ (A,D,I,O) で表される. ただし,
   (a) $A \in \Sigma_n$
   (b) $D = (V, E, \varphi)$ は $\Sigma = \Sigma_n \cup \Sigma_t$ 上の連結グラフ.Σ を total 頂点 アルファベットという.
   (c) $I \in V$ は入力 頂点
   (d) $O \in V$ は出力 頂点

図 2 に Hichart で用いられる終端・非終端頂点記号の例を示す.

生成規則 $r = (A, D, I, O)$ は $A \to D^{I,O}$ と書くことができる. また, 入力 頂点と出力 頂点が明らか

[module_packet] : プログラム入力前の状態
[profile_module_list] : プロファイル句の部分
[explanation] : 記述説明
[statement] : 文

非終端頂点記号



終端頂点記号

図 2: 終端・非終端頂点記号の一部

なときは, 単に $A \to D$ と書く.A を 生成規則 r の左辺,D を右辺という.

いくつかの生成規則によって,S とラベルされた１つの頂点のみからなるグラフ D_0(開始グラフ) から始めて,Σ_t によってラベルされたグラフが導出される. それぞれの導出では Σ_n でラベルされた頂点が,$A \to D^{I,O}$ という A を書き換える規則の右側で置き換えられる. もともと A に入っていた辺は I に入る辺となり,A から出ていく辺は O から出ていく辺となる.

GG によって定義される導出集合 $Y(GG)$ とは以下のように再帰的に定義されるグラフの集合である.

(1) $D_0 \in D(GG)$

(2) $D_1 = (V_1, E_1, \varphi_1) \in Y(GG)$, $\bar{v} \in V_1$, $\varphi_1(\bar{v}) = A \in \Sigma_n$, $r = (A, D_2, I, O) \in R$, $D_2 = (V_2, E_2, \varphi_2)$ とする.

ここで, D_2', I, O を次のように構成する.

(2.1) $D_2' = (V_2', E_2', \varphi_2') \equiv D_2$. ただし, equivalence 関数 $e : V_2 \to V_2', (V_1 - \{\bar{v}\}) \cap V_2' = \phi$

(2.2) $I' = e(I)$

(2.3) $O' = e(O)$

(3) さらに, $D_1' = (V_1', E_1', \varphi_1')$ を次のように構成する.

(3.1) $V_1' = (V_1 - \{\bar{v}\}) \cup \{v_I, v_O\}$, $v_I, v_O \notin V_1 \cup V_2'$;

(3.2)

- $\varphi_1'(v) = \varphi_1(v), {}^{\forall}v \in V_1 - \{\bar{v}\}$,
- $\varphi_1'(v_I) = \varphi_2(I)$,
- $\varphi_1'(v_O) = \varphi_2(O)$;

(3.3)

- $(v_1, v_2) \in E_1'$ ただし,$v_1, v_2 \in V_1 - \{\bar{v}\}, (v_1, v_2) \in E_1$;
- $(v_O, v_I) \in E_1'$ ただし,$(\bar{v}, \bar{v}) \in E_1$;
- $(v, v_I) \in E_1'$ ただし,${}^{\forall}v \in V_1 - \{\bar{v}\}, (v, \bar{v}) \in E_1$
- $(v_O, v) \in E_1'$ ただし,${}^{\forall}v \in V_1 - \{\bar{v}\}, (\bar{v}, v) \in E_1$
- E_1' にこれ以外の要素はない.

(4) $D = (V, E, \varphi) \in Y(GG)$. ただし, $D = D_2'$ $(I', O') \odot D_1'(v_I, v_O)$

D_1 と D の関係を $D_1 \Rightarrow D$ と書き,D_1 から D が 直接導出される という. また,r を 頂点 $\bar{v}$ に適用した生成規則 という.

$\Rightarrow$ の推移閉包を $\overset{*}{\Rightarrow}$ とすると,$Y(GG)$ は $Y(GG) = \{D \mid D_0 \overset{*}{\Rightarrow} D\}$ と書ける.$G \overset{*}{\Rightarrow} H$ のとき,G から H が 導出される という.$D_i \Rightarrow D_{i+1} \Rightarrow \cdots \Rightarrow D_{i+k}$ を長さ k の 導出 という.

導出の各ステップ $i > 0$ において, ステップ $i-1$ で導出された頂点を書き換える導出列を path という.

2.3.3 属性グラフ文法 [1]

定義 [1] 属性グラフ文法 は次の条件を満たす３項組 $G = \langle GG, A, F \rangle$ である.

1. $GG = (\Sigma_n, \Sigma_t, S, D_o, R)$ は,G の 基底文脈自由グラフ文法 と呼ばれる。また、$\Sigma = \Sigma_n \cup \Sigma_t$ 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の各元$X \in \Sigma$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$G$の属性集合という.ただし,$\mathcal{I}(S) = \phi$とする.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$r = X_0 \rightarrow D$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.　属性$a_0(X_{i_0})$を定義する意味規則は,$a_0(X_{i_0}) := f(a_1(X_{i_1}), \cdots, a_m(X_{i_m})), 0 \leq i_j \leq |Lab(D)|, X_{i_j} \in Lab(D), 0 \leq J \leq m$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$G$の意味規則集合という.

□

図3にHichartの属性グラフ文法で用いられる記号と関数を示す.

[X]	非終端頂点アルファベットX
<X>	文脈自由文法における非終端アルファベットX
"X"	文脈自由文法における終端アルファベットX
"•"	辺の形を整えるダミー頂点
GapX,GapY	セルとセルの最小幅、最小高
RootX,TopY	セルのX座標、Y座標の最小値
MinW,MinH	セルの最小の大きさ
Max(X,Y)	XとYの最大値を求める関数
get_width(X)	入力文字列のリストXからセルの幅を求める関数
get_height(X)	入力文字列のリストXからセルの高さを求める関数
get_str(X)	入力文字列のリストXからセル内部の文字列のリストを求める関数
get_line(X,Y)	始点セルXと終点セルYを結ぶ線が通る頂点の座標のリストを求める関数

図3: 属性グラフ文法で定義される記号、関数

□

2.3.4　edNCEグラフ文法[8]

定義　[8]　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は頂点の有限集合
2. Eは$\{(v, \gamma, w) \mid v, w \in V, v \neq w, \gamma \in \Gamma\}$の部分集合で,辺のラベルのアルファベット
3. φは頂点をラベル付けする関数

Hの構成要素もまた,ぞれぞれV_H, E_H, φ_Hで表す.

□

定義 [8]　$E_K = \{(f(v), \gamma, f(w)) | (v, \gamma, w) \in E_H\}$,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$f : V_H \rightarrow V_K$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$[GR_{\Sigma,\Gamma}]$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,$\mathcal{X} \rightarrow (D, C)$という形の生成規則であり,$\mathcal{X}$は非終端な頂点のラベル,$D$はグラフ,そして$C$は接続命令(connection instructions)の集合である.

そのような集合による書き換えステップは,$\mathcal{X}$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見ることが出来る.そして,書き換えステップは,主グラフ内の母頂点に対するこのオブジェクト(D, C)の置換として見ることが出来る.

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,$GRE_{\Sigma,\Gamma}$ 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. $(C \neq 0)$; したがって,$GR_{\Sigma,\Gamma} \subset GRE_{\Sigma,\Gamma}$ である. そして,' 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.

定義 [8] 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 は頂点アルファベット
2. Δ は終端頂点アルファベット
3. Γ は辺のアルファベット
4. $\Omega \subseteq \Delta$ は終端辺ラベル
5. R は生成規則の有限集合
6. $S \in \Sigma - \Delta$ は初期状態

$(D, C) \in GRE_{\Sigma,\Gamma}$ である生成規則 $r = \mathcal{X} \in \Sigma - \Delta$ を $\mathcal{X} \to (D, C)$ と記する.

□

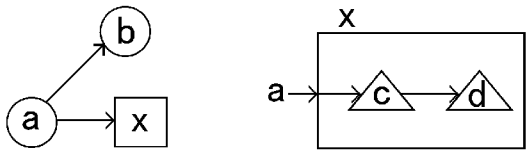
グラフ H に生成規則 r を適用して H' が得られるとき,H から H' が H $\Rightarrow$ H' と書く. また $\Rightarrow$ の推移閉包を $\stackrel{*}{\Rightarrow}$ と書く.

図 4 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) を適用して得られた結果グラフが (c) である.

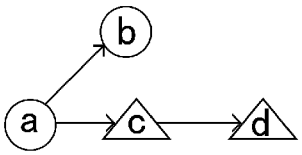
開始グラフ G_s から導出される終端グラフの集合

$$\{[H] \in GR_{\Delta,\Gamma}|G_s \stackrel{*}{\Rightarrow} H\ {}^{\forall}G_s\}$$

を G によって生成されるグラフ言語という.



ホストグラフ (a)　　生成規則 (b)



結果グラフ (c)

図 4: 生成規則・導出の例

Hichart グラフ文法は, これらの概念を元に定式化される. 従来は Vigna の CFGG[2] を用いていたのに対して, 今回は NCE グラフ文法を導入した. これは NCE 文法のほうが一般的であるという判断によるものである.

2.4 プログラム図の属性評価 [5, 7]

本研究ではプログラム図の描画の際の問題を解決するために文法に属性評価式を加える. 属性の種類は次の図 5 に示す.

Hichart 図の各セル (頂点) は 2 次元平面上に配置され, 各々x,y 座標や幅・高さといった情報を属性として持つ. プログラム図の導出の際, 各セルの属性は, 適用した生成規則の意味規則により定められる. ここでは, セルの属性評価の条件について述べる.

2.4.1 属性評価の条件 [5]

Hichart は 2 次元平面上に描画される図形であり, その形状を明確に表現するために属性評価式は以下

継承属性	
top	:部分木の最上部の X 座標
x	:セル x の座標
MinW,MinH	:セルの最小サイズ
GapX,GapY	:セルとセルの間隔
id	:セルの識別番号
合成属性	
bottom	:部分木の最下部の y 座標
y	:セルの y 座標
w,h	:セルの幅、高さ
cell	:セルの種類
string	:セルの内部に書かれる文字列
lines	:セル同士を結ぶ線
ll	:セルラベル
cl	:条件ラベル
nc	:部分木に存在するセルの数

図 5: 継承属性, 合成属性

の条件を満たすように定める.

- セル同士は重ならない
- 辺同士は交わらない

ここでは, この 2 つの条件を形式的に定義する.

定義 1 [5] 木構造 とは $T = (V, E, r, depth, width)$ の 5 項組である. ここで,(V, E) は 木 で,V はセルの集合,E を辺の集合とする. $r \in V$ は 根 セルである. $depth$ は V から Z への写像で $depth(p)$ はセル p の上下方向の長さを表す. $width$ は V から Z への写像で $width(p)$ はセル p の左右方向の長さを表す.

□

定義 2 [5] 2 項組 (T, π) を 木構造図式 という. ここで T は木構造で,$\pi : T$ のセルの集合 $\rightarrow Z \times Z$ は T の 配置 といわれる. $\pi(p) = (x, y)$ をセル p の 座標 という. このとき, その x 座標,y 座標は,$\pi_x(p)$, $\pi_y(p)$ でそれぞれ表される. また, セル p の座標はセル p の左上隅を指す. プログラム図式への適用を考慮し, 座標はディスプレイ座標系とする.

□

木構造図式は各セルに属性 $width(p)$, $depth(p)$, $\pi_x(p)$, $\pi_y(p)$ が割り当てられた根付き木である. ここでは, 子供セルに 上から下へ と順序がついている 順序 木構造図式を対象とする. 木構造図式は, 長方形が木構造上に配置された図式を意味する.

木構造図式 $D = (T, \pi)$ の 深さ $depth(T, \pi)$ を次のように定める:

$depth(T, \pi) \equiv$

$max(\{\pi_y(P) + depth(p) - \pi_y(q) - 1 \mid$ ただし,
p と q は T 中のセルで, $\pi_y(p) > \pi_y(q)\})$

セル p の レベル$level(p)$ は,p と根セルとの間の辺の個数で定義される.

これらの条件の元で, セルの配置に関する前述の条件を, 以下のように定義する.

条件 C0 [5] 木構造図式 (T, π) において, セル p と q のレベルが等しく,$\pi_y(p) < \pi_y(q)$ であるならば,$\pi_y(q$ の長男$) > \pi_y(p)$ の末っ子+$depth(p$ の末っ子), すなわち, 線は交差しない.

□

条件 C1 [5] セル p に対して,

$area(p, \pi) \equiv$

$\{(x, y) \mid \pi_x(p) \leq x \leq \pi_x(p) + width(p) - 1,$
$\pi_y(p) \leq y \leq \pi_y(p) + depth(p) - 1\}$

としたとき, すべての p,$q(p \neq q)$ について

$$area(p, \pi) \cap area(q, \pi) = \phi$$

すなわち, セル同士は重ならない.

□

2.4.2 セル配置における条件 [7],[10],[11]

セル配置の際, 前章の条件を満たした上でかつ, アルゴリズムが単純で全体の処理の流れが見やすいという理由で, 以下の 4 つの条件を元にセルの座標を計算し, セルを配置する.

なお, ここに出てくるセル P の レベル とは, 根セルとセル P との間の接続線の中で、一方のセルの右辺ともう一方のセルの左辺を結ぶ線の本数である. 従って根セルのレベルは 0 である. また,P_i,Q_j をそれぞれ, レベル i のセル P_i, レベル j のセル Q_j とし, これらを根とする部分木を $T(P_i)$,$T(Q_j)$ とする.

条件 1 [7] 同じレベルのセルの X 座標はすべて等しい (図 6 左). よって図 6 右のようになることはない.

$$x(P_i) = x(Q_i)$$

□

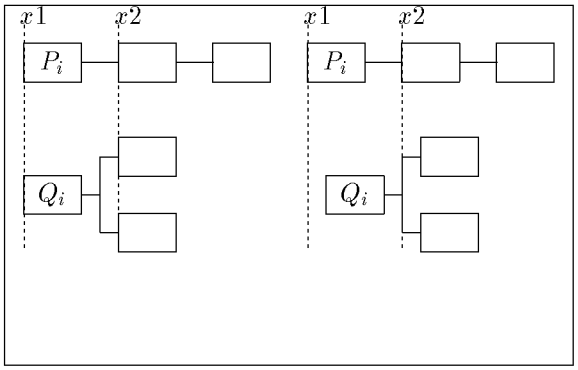


図 6: 条件 1:同一レベルのセルの配置

条件 2 [7] 1 本の接続線で繋がれている隣り合うセルは X 軸に関して $GapX$(定数) だけ離れて配置される. セル P_i の幅を $width(P_i)$ とすると

$$x(P_i + 1) - x(P_i) = width(P_i) + GapX$$

従って上記の 2 つの条件から,「葉」でないセルの幅はすべて同じである. この値を $MinW$ とすると,

$$width(P_i) = MinW$$

(ただし P_i は「葉」ではない任意のセル)

□

条件 3 [7] 部分木 $T(P_i)$ と部分木 $T(Q_i)$ $(y(P_i) < y(Q_i))$ は Y 軸に関して $GapY$(定数) だけ離れて配置される. 従って, セルが大きくなっても下のセルと重なることはない.

$$min\{y(q) - (y(p) + height(p))\} = GapY$$

ここで, セル p は $T(P_i)$ 内の任意のセル, セル q は $T(Q_i)$ 内の任意のセル, $height(p)$ はセル p の高さとする.

□

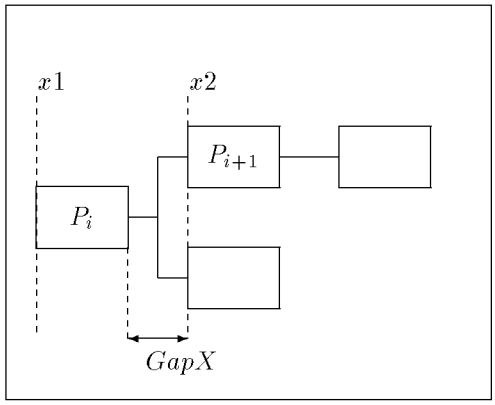


図 7: 条件 2:隣り合うセルの配置

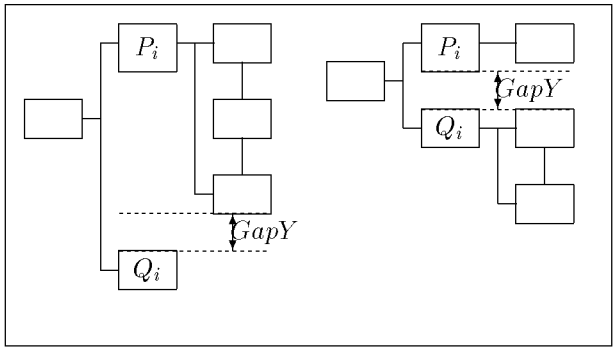


図 8: 条件 3: 2 つの部分木の配置

条件 4 [7] セル P_i が 2 つ子セル $P_{i,1}$,$P_{i,2}$ を持つ $(y(P_{i,1}) < y(P_{i,2}))$ とき, セル P_i は Y 座標に関して 2 つの子セルの中心に配置される.

$$y(P_i) = (y(P_{i,1}) + y(P_{i,2}))/2$$

一般に, セル P_i が k 個の子セル $P_{i,1}$,$P_{i,2}$,...,$P_{i,k-1}$,$P_{i,k}$ を持つ $(y(P_{i,j}) < y(P_{i,j+1}), 1 \leq j \leq k-1)$ とき, P_i の Y 座標は以下のようになる.

$y(P_i) =$
$(y(P_{i,1}) + (y(P_{i,2}) + \cdots + (y(P_{i,k-1}) + y(P_{i,k}))$
$/2 \cdots /2)/2)/2$

ただし, 子セル $P_{i,j}$ の下辺と子セル $P_{i,j+1}$ の上辺が接続線で繋がれているとき,P_i の Y 座標は $P_{i,1}$ の Y 座標に等しくなる.

$$y(P_i) = y(P_{i,1})$$

□

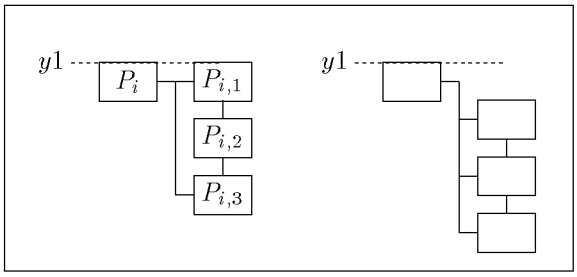


図 9: 条件 4 の例外 (左)

これらの条件を満すことにより, プログラム図として問題なく描画することができる. しかし, これらを厳密に守らなくともプログラム図として成立する描画は可能である.

3 プログラム図として成立するための描画条件

前章で述べた描画規則は, セルの配置等を意味規則で厳密に規定している. しかし, これらすべてに従わずともある程度の条件を満たせばプログラム図として成立する.

ここでは,Hichart プログラム図の描画を考慮して, 新に描画条件を定める.

また, 前提として, 前章の条件 C0 および C1 は満たすものとする.

条件 A1 (cf. 条件 1) 同じ親セルを持つ子セルの X 座標はすべて等しい.

$$x(P_{i,j}) = x(P_{i,k}) \quad (j \leq k)$$

□

条件 A2 (cf. 条件 2) 1 本の接続線で繋がれている隣り合うセルは X 軸に関して重ならずに配置される. セル P_i の幅を $width(P_i)$ とすると

$$x(P_i + 1) > x(P_i) + width(P_i)$$

□

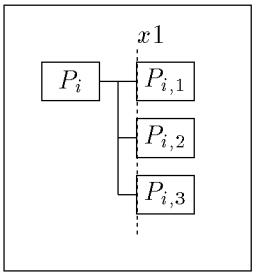


図 10: 条件 A1:同一の親を持つセルの配置

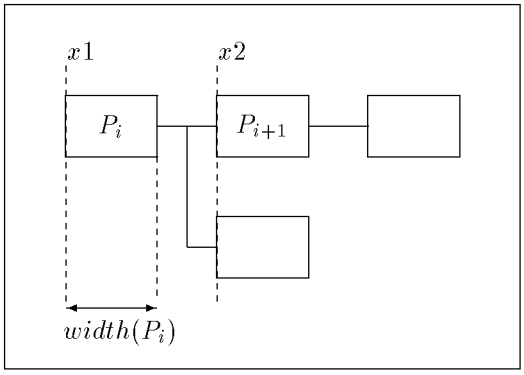


図 11: 条件 A2:隣り合うセルの配置

条件 A3 (cf. 条件 3) 部分木 $T(P_i)$ と部分木 $T(Q_i)$ $(y(P_i) < y(Q_i))$ は Y 軸に関して重ならずに配置される. 従って, セルが大きくなっても下のセルと重なることはない.

$$min\{y(q) - (y(p) + height(p))\} > 0$$

ここで, セル p は $T(P_i)$ 内の任意のセル, セル q は $T(Q_i)$ 内の任意のセル, $height(p)$ はセル p の高さとする.

□

条件 A4 (cf. 条件 4)
セル P_i が子セル $P_{i,1}, P_{i,2}, \ldots P_{i,n}$ を持つ $(y(P_{i,j}) < y(p_{i,j+1}))$ とき, セル P_i は Y 座標に関してその長男と末っ子の間に配置される.

$$y(P_{i,1}) < y(P_i) < y(P_{i,n})$$

ただし, 子セル $P_{i,j}$ の下辺と子セル $P_{i,j+1}$ の上辺が接続線で繋がれているとき,P_i の Y 座標は $P_{i,1}$ の

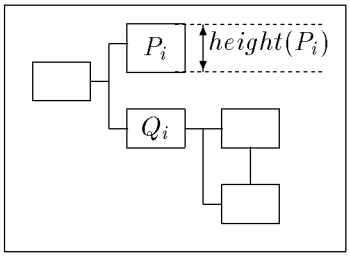


図 12: 条件 A3: 2 つの部分木の配置

Y 座標に等しくなる.

$$y(P_i) = y(P_{i,1})$$

□

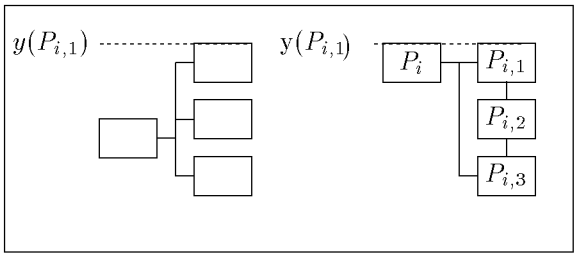


図 13: 条件 A4 の例外 (右側)

これらの条件により, 従来の制約よりも柔軟に属性評価に対応でき, また, 図としての正誤判定も可能になる.

4 Hichart の NCE グラフ文法

ここでは, 今回用いる属性 NCE 文法と, これに基づいて定義される DXL 対応 Hichart のグラフ文法 (DHGG) について述べる.

4.1 属性 NCE グラフ文法

定義 属性 NCE 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,G_N の 基底 NCE 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.$A = \bigcup_{X \in \Sigma}$ 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 $r = X_0 \rightarrow (D, C)$ 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. 属性 $a_0(X_{i_0})$ を定義する意味規則は,$a_0(X_{i_0}) := f(a_1(X_{i_1}), \cdots, a_m(X_{i_m}))$ または $a_0(X_{i_0}) > f(a_1(X_{i_1}), \cdots, a_m(X_{i_m}))$, $0 \leq i_j \leq |Lab(D)|, X_{i_j} \in Lab(D), 0 \leq j \leq m$ 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.

□

4.2 DXL 対応 Hichart のグラフ文法

DXL 対応 Hichart グラフ文法 (DHGG) は属性 NCE 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る. この文法は DXL の構文全てに対応している (図 14). 生成規則の例を図 16 に示す. すべての生成規則は付録に収録する.

定義 DHGG に基づき導出される階層型の疑似木構造グラフを Hichart プログラム図という.

□

図 15 にプログラム図の導出の例を示す. この例では, 非終端セル「[module_list]」に生成規則「module_list」が適用され, 非終端セル「[module]」が生成されている.

定理 DHGG が生成するプログラム図は, 描画条件 A1∧A2∧A3∧A4 を満足する.

DXL 構文	対応する生成規則
モジュール・パケット	module_packet
プロファイル句	profile
モジュール識別句	module
モジュール論理句	explanation_ module_algorithm
文	
文の列	statement_list
文	statement
基本文	
基本文	fundamental_statement
ブロック化文	blocked_statement
	abstract_statement
	abstract
	block_specification
複合文	compound_statement
順次文	sequential_statement
並列文	parallel_statement
繰り返し文	iterative_statement
前判定繰り返し文	pre-tested_statement
後判定繰り返し文	post-tested_statement
継続繰り返し文	continued_statement
選択文	selective_statement
単純選択文	if_then_statement
多岐選択文	exclusive_statement
if 型選択文	exclusive_if_statement
case 型選択文	exclusive_case_statement
多重岐選択文	when_statement
打ち切り文	terminate_statement

図 14: DXL 構文と生成規則の対応

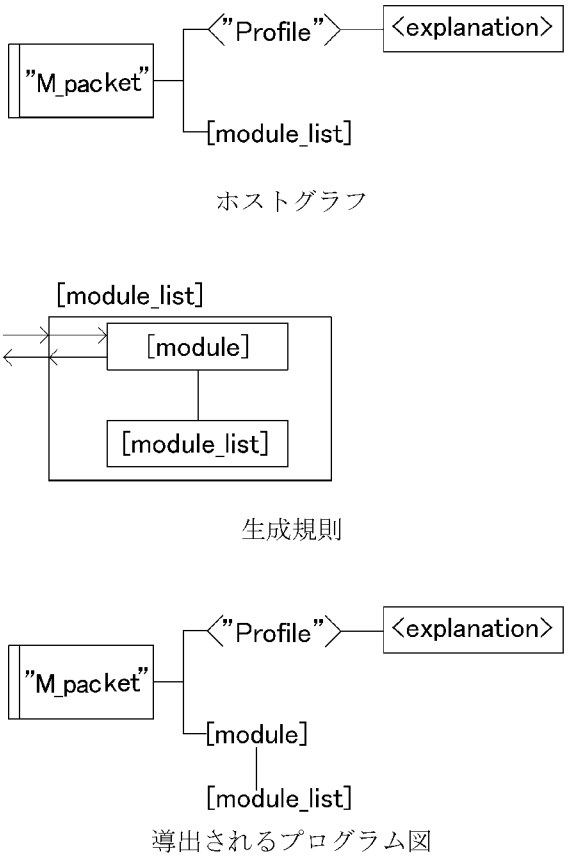


図 15: プログラム図の導出の例

□

証明

A1) 複数の子セルを生成する生成規則の意味規則は, 生成する子セルすべてに等しい X 座標を与える. また, 兄弟セルを生成する生成規則の意味規則は, 生成する兄弟セルに書き換える元のセルと等しい X 座標を与える. したがって, 同じ親を持つ子セルの X 座標は等しい.

A2) 子セルを生成する生成規則の意味規則は, 子セルの X 座標に, 親セルの X 座標 + 親セルの幅より大きな X 座標を与える. したがって,1 本の接続線でつながれているセルは,X 座標に関して重ならずに配置される.

A3) 部分木 $T(P)$ の親セルの兄弟セルを生成する生成規則の意味規則は, 部分木 $T(P)$ の最下部より大きな Y 座標を与える. また,Y 座標は合成属性であり, 生成された兄弟セルの親セルを根までさかのぼって同様に評価される. したがって, 部分木 $T(P)$ とその兄弟木 $T(Q)$ は重ならない.

A4) 複数の子セルを生成する生成規則の意味規則は, 子セルの Y 座標の間の Y 座標を親セルに与える. また,Y 座標は合成属性であり, 生成された兄弟セルの親セルを根までさかのぼって同様に評価される. したがって, 親セルは Y 座標に関して子セルの間に配置される.

□

5 まとめ

従来の DXL の生成規則は CFGG[2] を基底文脈自由グラフ文法としており, 概念的にはやや古いものとなっていた.

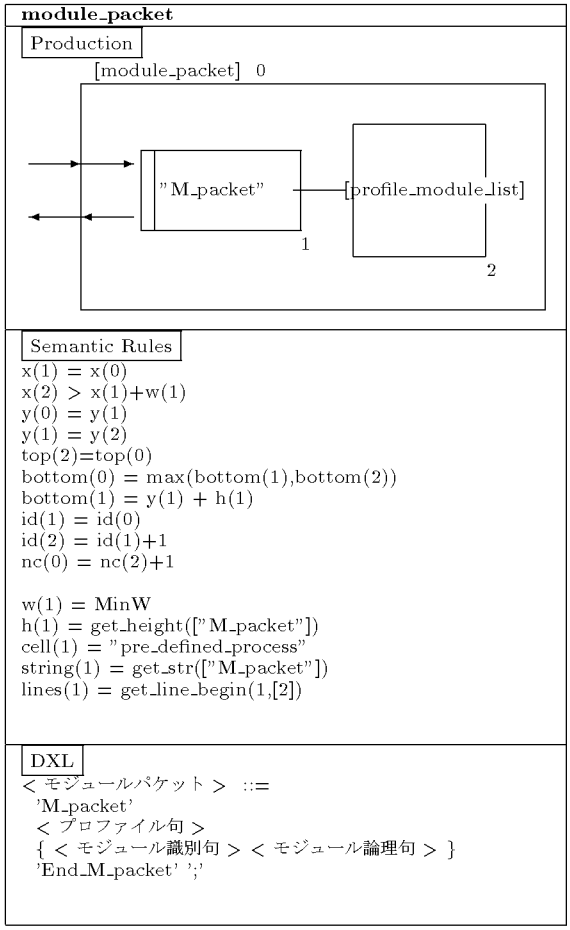


図 16: 生成規則の例 (NCE 文法)

しかし,NCE グラフ文法を基底とすることにより, 表やシミュレータの一般的な処理系と統一的に扱うことができる.

本論文では, 属性 NCE 文法を用いて Hichart の構文規則とセルの配置条件を定義した. この属性 NCE 文法は Hichart の構文規則と図表示のための情報を導出する属性評価式から成る.

参考文献

[1] D.E.Knuth; Semantic of Context–Free Languages; *Math. Syst. Theory2*,pp.127–145 (1968)

[2] Pierluigi Della Vigna, Carlo Ghezzi; Context–Free Graph Grammars; *Information and Control 37*,pp.207–233(1978)

[3] T.Yaku, K.Futatsugi, A.Adachi and E.Moriya; HICHART–A Hierachical Flowchart Description Language–; *Proc. IEEE COMPSAC 11*, pp157–163(1987)

[4] 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**, pp.81–92 (1988)

[5] 海野浩, 安斎公士, 小倉耕一, 西野哲郎, 中西美智子, 夜久竹夫; 木構造図式の描画問題; **情報処理学会論文誌 Vol.33, No.7**, pp.879–886 (1992)

[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, pp.1–36 (1995)

[7] 大井裕一; Hichart を用いたプログラム開発支援システム; **東洋大学大学院工学研究科修士論文**, pp1–54(1995)

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

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

[10] Y.Adachi,T.Imaki,K.Tsuchida and T.Yaku; Program Visualization Using Attribute Graph Grammars; *Proc. IFIP WCC*(1998)

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

[12] 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**, pp149–164(1998)

研究業績

論文

1. DXL 対応プログラム図言語に対する属性グラフ文法, 東洋大学工学部研究報告 (投稿中)

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土田賢省教授, 宮寺教授, 安達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, 冨山君, 類瀬君, ならびに同研究室の皆様に心よりお礼を申し上げます。

付録

1. 非終端アルファベット・終端アルファベット一覧

非終端セルアルファベット:		
[module_packet]	[profile_module_list]	[profile]
[module_list]	[module]	[explanation_module_algorithm]
[explanation]	[module_algorithm]	[statement_list]
[label_statement]	[statement]	[fundamental_statement]
[blocked_statement]	[abstract_compound]	[abstract]
[block_specification]	[compound_statement]	[sequential_statement]
[parallel_statement]	[parallel_statement_list]	[iterative_statement]
[pre_tested_statement]	[post_tested_statement]	[continued_statement]
[selective_statement]	[if_then_statement]	[branch_statement_list]
[exclusive_select_statement]	[exclusive_if_statement]	[else_if_statement]
[expensive_case_statement]	[inclusive_case_statement]	[when_statement]
[terminate_statement]		

Table1: 非終端セルアルファベット一覧

非終端アルファベット:		
<< module_packet >>	<< profile_module_list >>	<< profile >>
<< module_list >>	<< module >>	<< explanation_module_algorithm >>
<< explanation >>	<< module_algorithm >>	<< statement_list >>
<< label_statement >>	<< statement >>	<< fundamental_statement >>
<< blocked_statement >>	<< abstract_compound >>	<< abstract >>
<< block_specification >>	<< compound_statement >>	<< sequential_statement >>
<< parallel_statement >>	<< prallel_statement_list >>	<< iterative_statement >>
<< pre_tested_statement >>	<< post_tested_statement >>	<< continued_statement_ >>
<< selective_statement >>	<< if_then_statement >>	<< branch_statement_list >>
<< exclusive_select_statement >>	<< exclusive_if_statement >>	<< else_if_statement >>
<< expensive_case_statement >>	<< inclusive_case_statement >>	<< when_statement >>
<< terminal_statement >>		

Table2:非終端アルファベット一覧

終端アルファベット:				
”M_packet”	”Profile”	”Identifer is”	”Module_Algorithm”	”impertive”
”null”	”call”	”goto”	”abstract”	”begin”
”parallel”	”condition”	”for”	”until”	”while”
”loop”	”if”	”then”	”else_if”	”case”
”terminate”	”system”	”module”	”block”	

Table3:終端アルファベット一覧

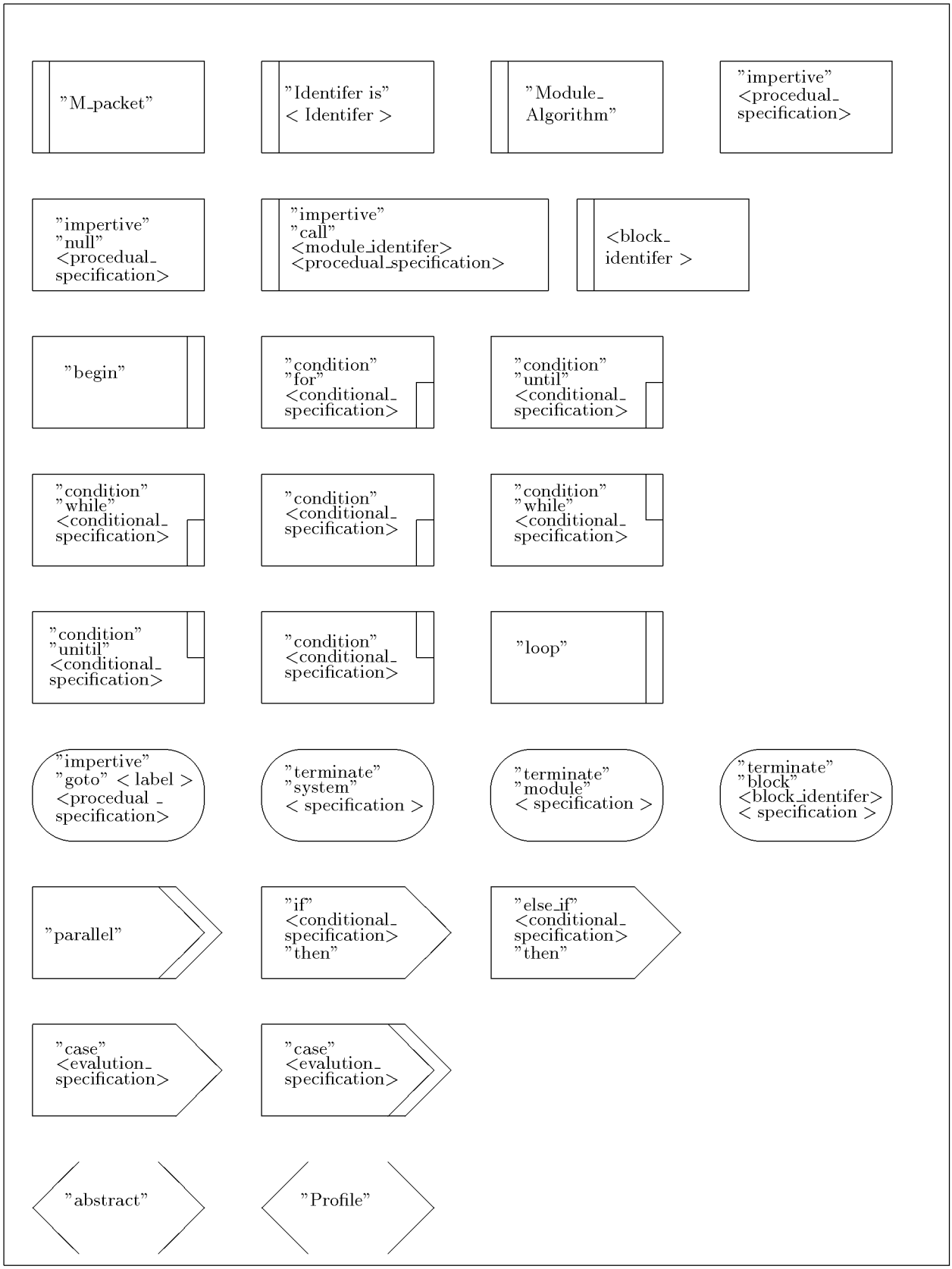


Table4:終端セルアルファベット

継承属性：	
top	:部分木の最上部の x 座標
x	:セルの x 座標
RootX	:ルートセルの x 座標
TopY	:最上位セルの y 座標
MinW, MinH	:セルの最小サイズ
id	:セルの識別番号
合成属性：	
bottom	:部分木の最下部の y 座標
y	:セルの y 座標
w, h	:セルの幅、高さ
cell	:セルの種類
string	:セルの内部にかかれる文字列
lines	:セル同士を結ぶ線
ll	:セルラベル
cl	:条件ラベル
nc	:部分木内に存在するセルの数

Table5:継承属性, 合成属性

[X]	非終端ノードアルファベット X
” < X > ”	文脈自由文法における非終端アルファベット X
”X”	文脈自由文法における終端アルファベット X
” • ”	辺の形を整えるためのダミーノード
RootX,TopY	セルの X 座標、Y 座標の最小値
MinW,MinH	セルの最の大きさ
max(X,Y)	X と Y の最大値を求める関数
get_width(X)	入力文字列のリスト X からセルの幅を求める関数
get_height(X)	入力文字列のリスト X からセルの高さを求める関数
get_str(X)	入力文字列のリスト X からセル内部の文字列のリストを求める関数
get_line(X,Y)	始点セル X と終点セル Y を結ぶ線が通る頂点の座標のリストを求める関数

Table6:属性グラフ文法で定義される記号、関数

<< X >>	非終端アルファベット X
” < X > ”	終端アルファベット X(文脈自由文法における非終端アルファベット X)
”X”	終端アルファベット (文脈自由文法における終端アルファベット X)
RootX, TopY	セルの X 座標、Y 座標の最小値
MinW, MinH	セルの最小の大きさ
max(X, Y)	X と Y の最大値を求める関数
get_width(X)	入力文字列のリスト X からセルの幅を求める関数
get_height(X)	入力文字列のリスト X からセルの高さを求める関数
get_str(X)	入力文字列のリスト X からセル内部の文字列のリストを求める関数
get_line(X, Y)	始点セル X と終点セル Y を結ぶ線が通る頂点の座標のリストを求める関数

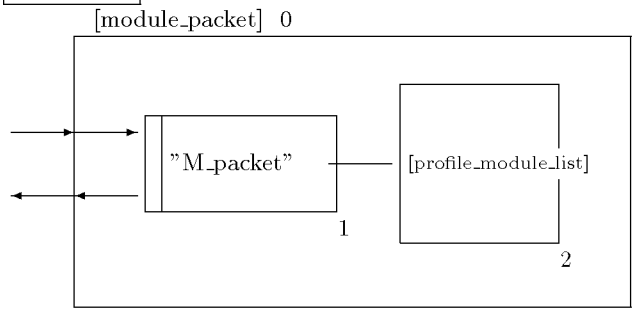
Table7:属性文法で定義される記号、関数

2.DXL 対応 Hichart 生成規則一覧

DXL – Production Rule – 01

module_packet

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1)+w(1)
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])

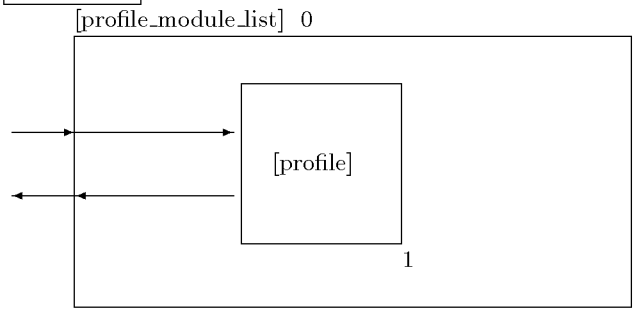
DXL

< 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

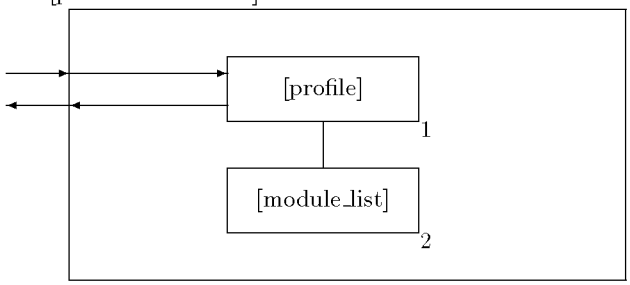
< モジュールパケット > ::=
'M_packet'
< プロファイル句 >
{ < モジュール識別句 > < モジュール論理句 > }
'End_M_packet' ';'

DXL – Production Rule – 03

profile_module_list(2)

Production

[profile_module_list] 0



Semantic Rules

x(1) = x(0)
x(2) = x(1)
y(0) = y(1)
top(1) = top(0)
top(2) > bottom(1)
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

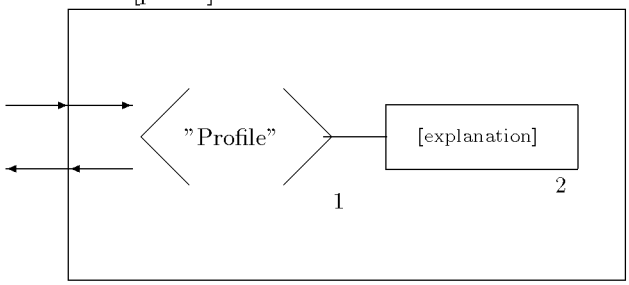
< モジュールパケット > ::=
'M_packet'
< プロファイル句 >
{ < モジュール識別句 > < モジュール論理句 > }
'End_M_packet' ';'

DXL – Production Rule – 04

profile

Production

[profile] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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])

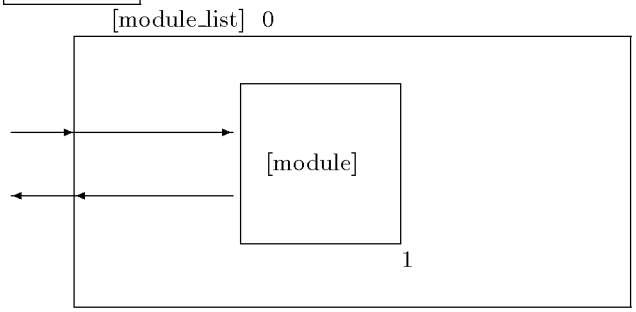
DXL

< 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)

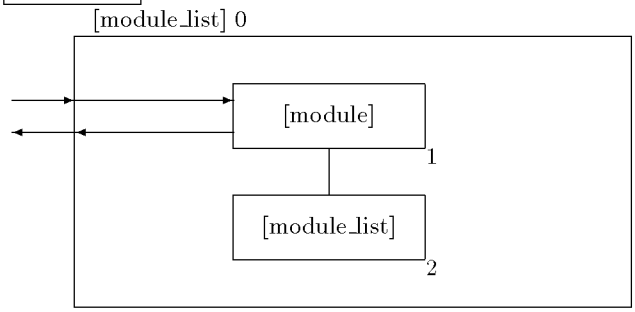
DXL

< 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)
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])

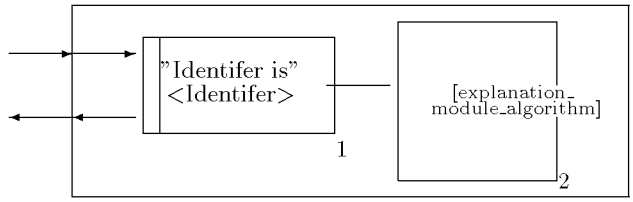
DXL

< 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 [module] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1)+w(1)
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(["Identifer is", <identifer>"])
cell(1) = "pre_defined_process"
string(1) = get_str(["Identifer is", <identifer>"])
lines(1) = get_line_begin(1,[2])

DXL

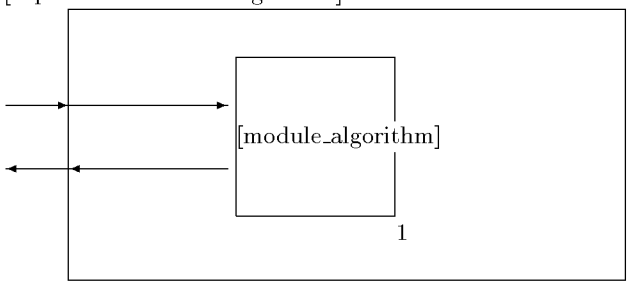
< モジュールパケット > ::=
'M_packet' < プロファイル句 >
{ < モジュール識別句 > < モジュール論理句 > }
'End_M_packet' ';'
< モジュール識別句 > ::=
'Identifation'
'Identifer' 'is' < 識別子 >
[< 説明記述 >]
'End_Identifation' ';'
< モジュール論理句 > ::=
'Module_Algorithm'
< 次の列 >
'End_Module_Algorithm' ';'

DXL – Production Rule – 08

explanation_module_algorithm(1)

Production

[explanation_module_algorithm] 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

< 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' ';'

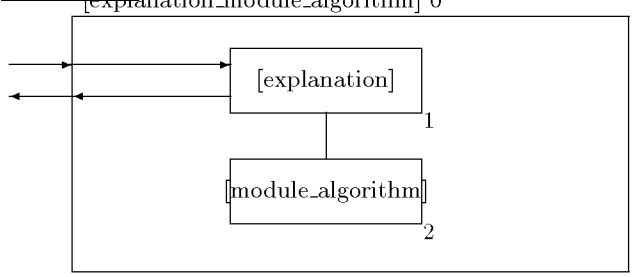
< モジュール識別句 > ::=
'Identifation'
'Identifer' 'is' < 識別子 >
[< 説明記述 >]
'End_Identifation' ';'

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

DXL – Production Rule – 09

explanation_module_algorithm(2)

Production



Semantic Rules

x(1) = x(0)
x(2) = x(1)
y(0) = y(1)
top(1) = top(0)
top(2) > bottom(1)
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' ';'

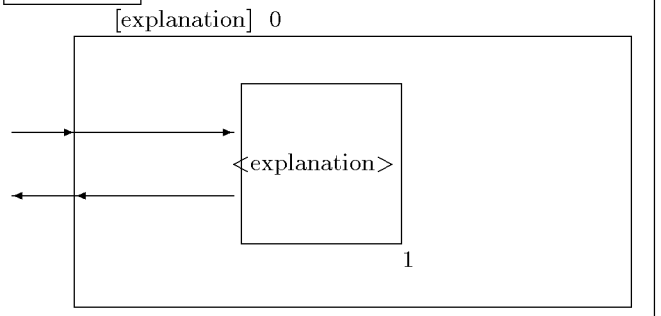
< モジュール識別句 > ::=
'Identifation'
'Identifer' 'is' < 識別子 >
[< 説明記述 >]
'End_Identifation' ';'

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

DXL – Production Rule – 10

explanation

Production



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>])

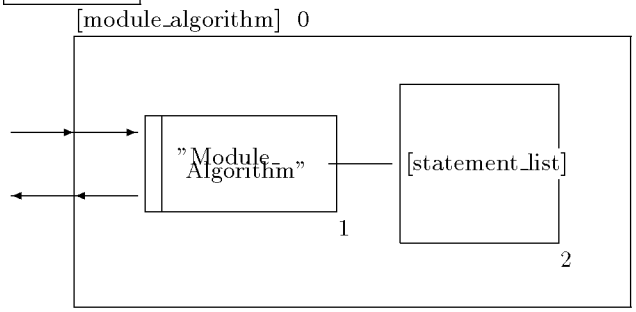
DXL

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

DXL – Production Rule – 11

module_algorithm

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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])

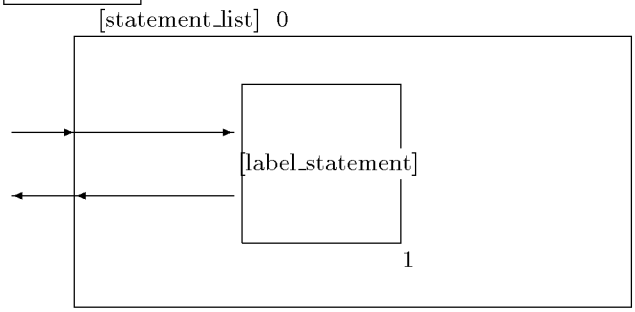
DXL

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

DXL – Production Rule – 12

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)

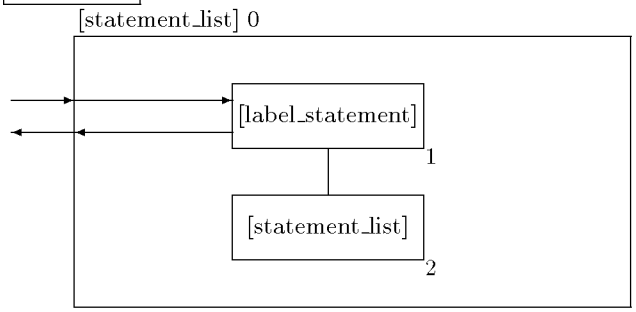
DXL

< 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)
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])

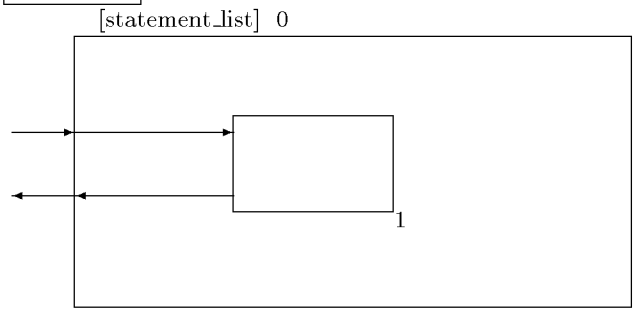
DXL

< 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)
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([""])

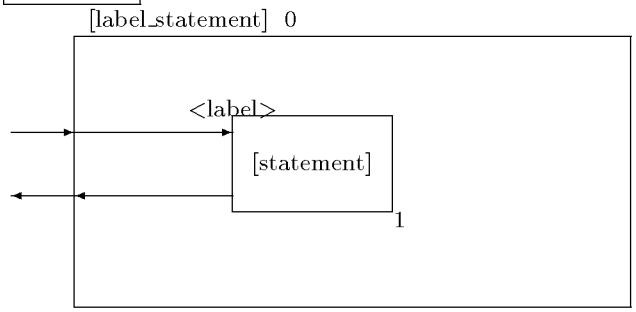
DXL

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

DXL – Production Rule – 15

label_statement(1)

Production



Semantic Rules

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

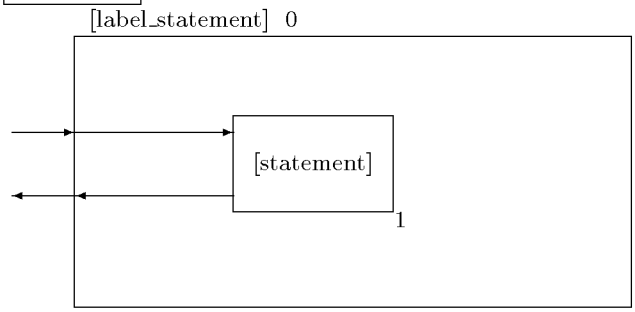
DXL

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

DXL – Production Rule – 16

label_statement(2)

Production



Semantic Rules

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

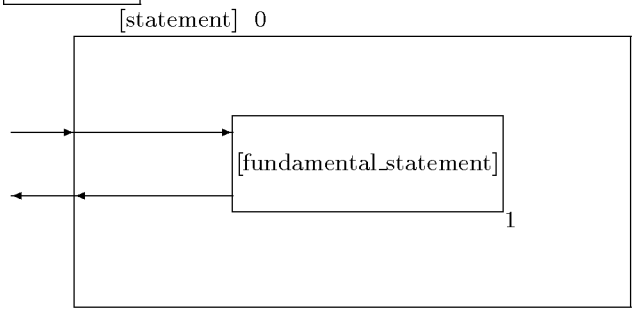
DXL

< 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)

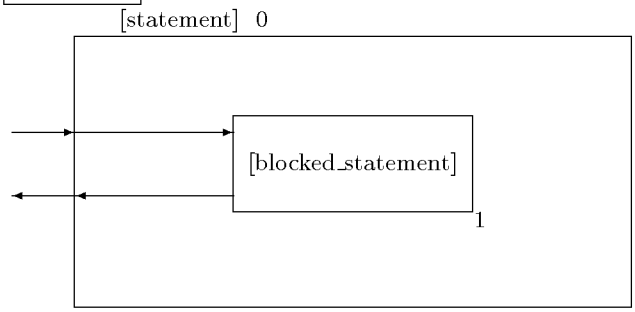
DXL

< 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

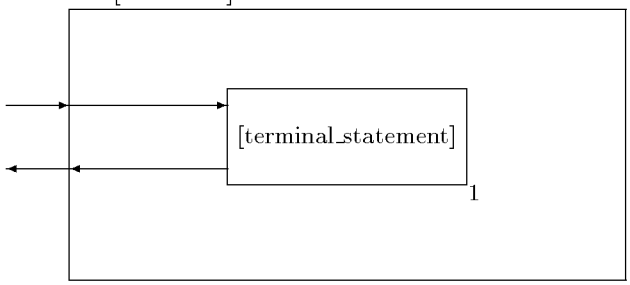
< 文 > ::=
< 基本文 > | < ブロック文 > | < 打切り文 >

DXL – Production Rule – 19

statement(3)

Production

[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

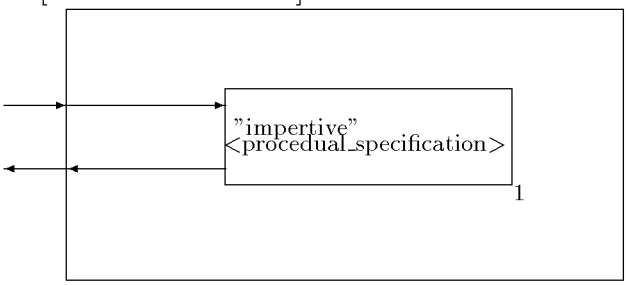
< 文 > ::=
< 基本文 > | < ブロック文 > | < 打切り文 >

DXL – Production Rule – 20

fundamental_statement(1)

Production

[fundamental_statement] 0



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

DXL

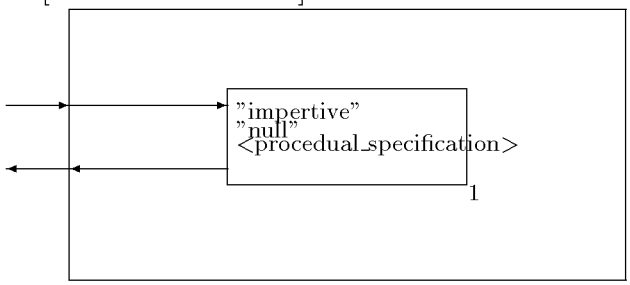
< 文 > ::=
’imperative’
[<< 基本文型 >>] < 手続き仕様記述 > ’;’

DXL – Production Rule – 21

fundamental_statement(2)

Production

[fundamental_statement] 0



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

DXL

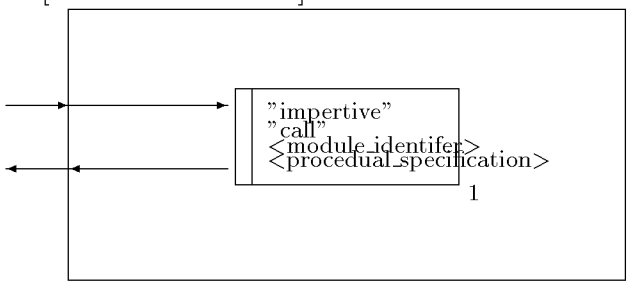
< 文 > ::=
  'imperative'
  [<< 基本文型 >>] < 手続き仕様記述 > ';'
<< 基本文型 >> ::=
<<null>> | << モジュール呼び出し >> | <<goto>>
<<null>> ::= "null"

DXL – Production Rule – 22

fundamental_statement(3)

Production

[fundamental_statement] 0



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", "call",
    <identifer>, <procedual_specification>])
h(1) =
  get_height(["imperative", "call",
    <identifer>, <procedual_specification>])
cell(1) = "pre_defined_process"
string(1) =
  get_str(["imperative", "call",
    <identifer>, <procedual_specification>])

DXL

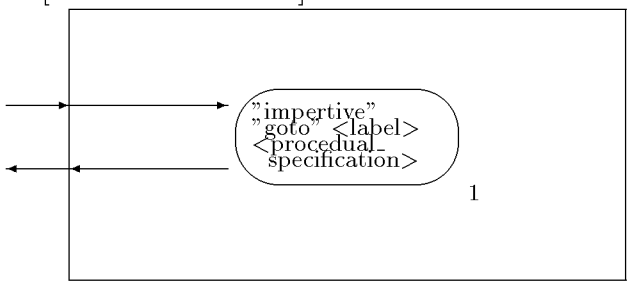
< 文 > ::=
  'imperative'
  [<< 基本文型 >>] < 手続き仕様記述 > ';'
<< 基本文型 >> ::=
<<null>> | << モジュール呼び出し >> | <<goto>>
<< モジュール呼び出し >> ::= "call" << モジュール識別子 >>

DXL – Production Rule – 23

fundamental_statement(4)

Production

[fundamental_statement] 0



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

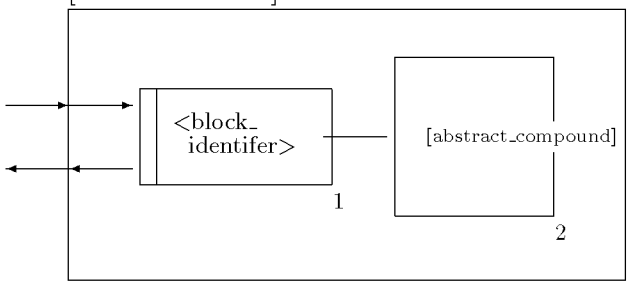
< 文 > ::=
  'imperative'
  [<< 基本文型 >>] < 手続き仕様記述 > ';'
<< 基本文型 >> ::=
<<null>> | << モジュール呼び出し >> | <<goto>>
<<goto>> ::= "goto" << 名札 >>

DXL – Production Rule – 24

blocked_statement(1)

Production

[blocked_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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_identifer>])
cell(1) = "pre_defined_process"
string(1) = get_str([<block_identifer>])
lines(1) = get_line_begin(1,[2])

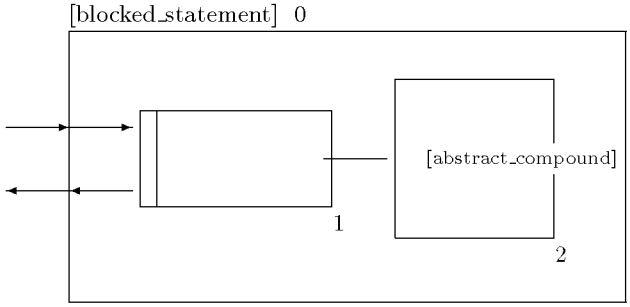
DXL

< 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)
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])

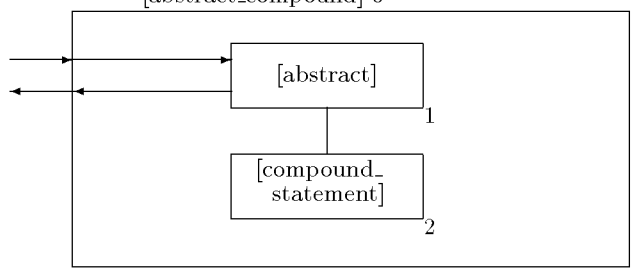
DXL

< 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)
bottom(0) = bottom(2)
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)

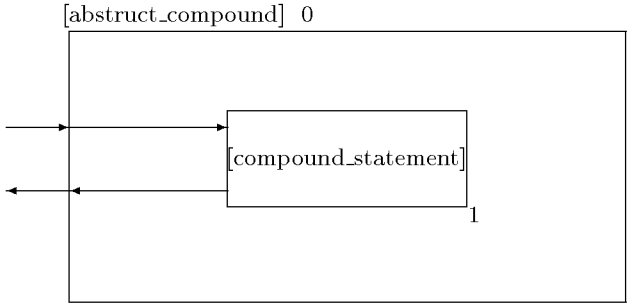
lines(1) = get_line(1, [2])

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 >

abstruct_compound(2)

Production



Semantic Rules

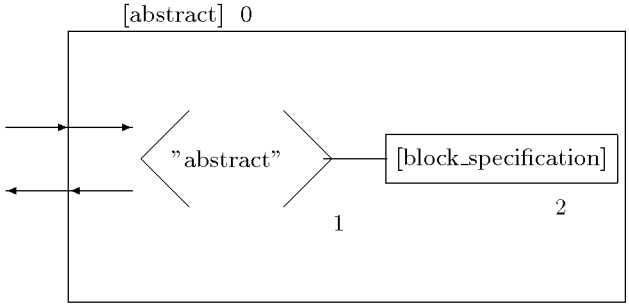
x(1) = x(0)
y(0) = y(1)
top(1) = top(0)
bottom(0) = bottom(1)
id(1) = id(0)
nc(0) = nc(1)

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 >

abstract

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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([”abstract”])
cell(1) = ”caption”
string(1) = get_str([”abstract”])
lines(1) = get_line_begin(1,[2])

DXL

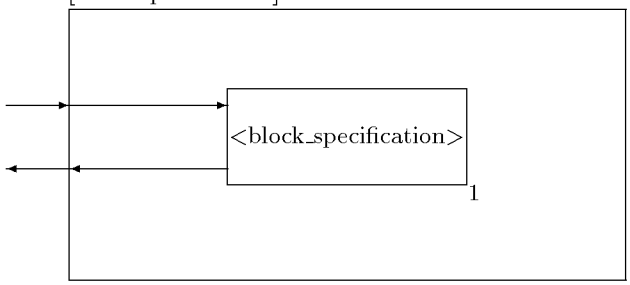
< ブロック化文 > ::=
[< ブロック識別子 >]
[”abstract” << ブロック仕様記述 >>]
< 複合文 >

DXL – Production Rule – 29

block_specification

Production

[block_specification] 0



Semantic Rules

x(1) = x(0)
y(0) = y(1)
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

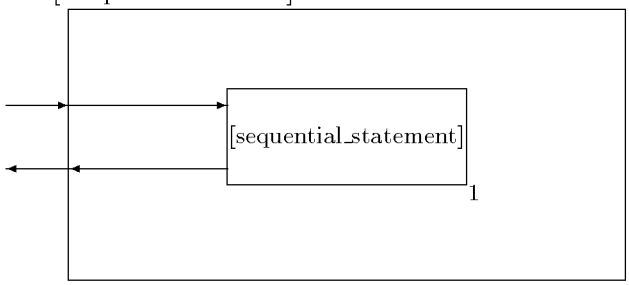
< ブロック化文 > ::=
[< ブロック識別子 >]
["abstruct" << ブロック仕様記述 >>]
< 複合文 >

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

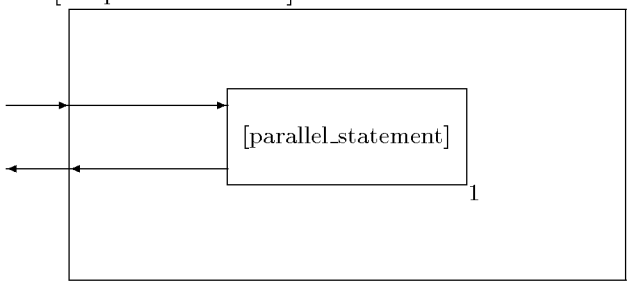
< 複合文 > ::=
< 順次文 > | < 並列文 > | < 繰り返し文 > | < 選択文 >

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

DXL

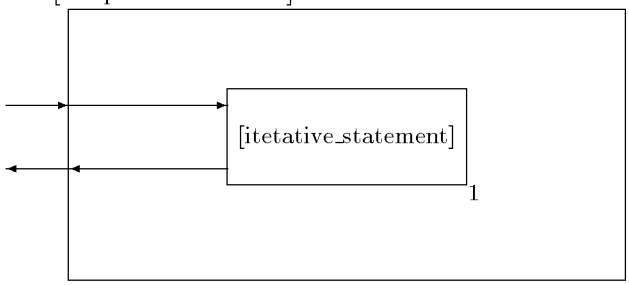
< 複合文 > ::=
< 順次文 > | < 並列文 > | < 繰り返し文 > | < 選択文 >

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

DXL

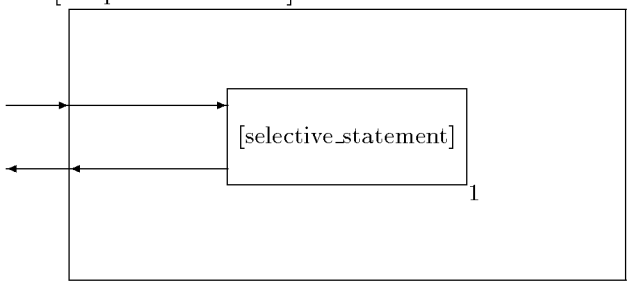
< 複合文 > ::=
< 順次文 > | < 並列文 > | < 繰り返し文 > | < 選択文 >

DXL – Production Rule – 33

compound_statement(4)

Production

[compound_statement] 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) = nc(1)

DXL

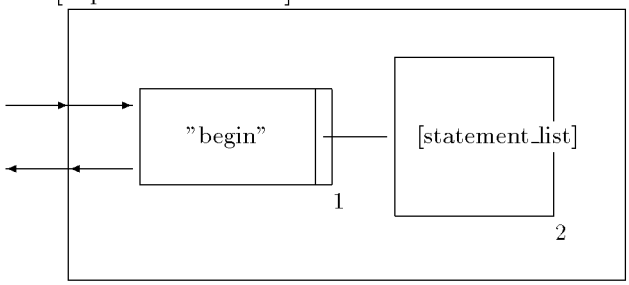
< 複合文 > ::=
< 順次文 > | < 並列文 > | < 繰り返し文 > | < 選択文 >

DXL – Production Rule – 34

sequential_statement

Production

[sequential_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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) = "conditions_iteration"
string(1) = get_str(["begin"])
lines(1) = get_line_begin(1,[2])

DXL

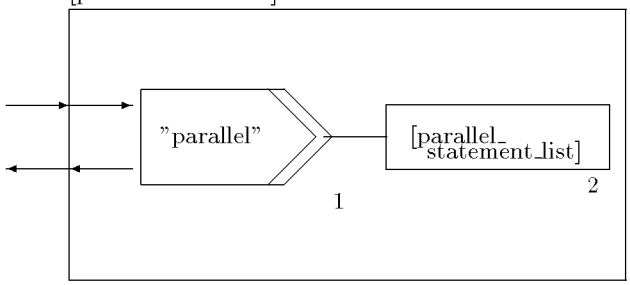
< 順次文 > ::=
'begin'
< 文の列 >
'end' ';'

DXL – Production Rule – 35

parallel_statement

Production

[parallel_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

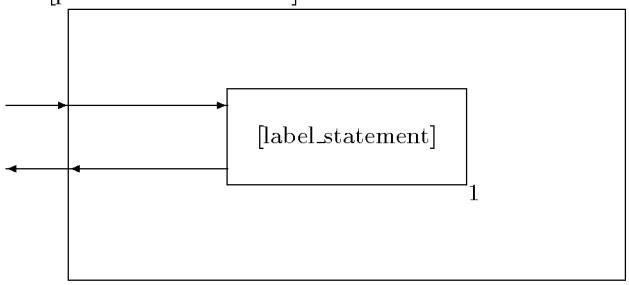
< 並列文 > ::=
'parallel'
< 文の列 >
'end_parallel' ';'

DXL – Production Rule – 36

parallel_statement_list(1)

Production

[parallel_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

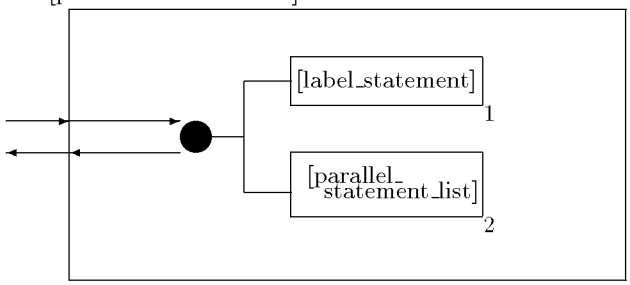
< 並列文 > ::=
'parallel'
< 文の列 >
'end_parallel' ';'

DXL – Production Rule – 37

parallel_statement_list(2)

Production

[parallel_statement_list] 0



Semantic Rules

x(1) = x(0)
x(2) = x(1)
y(1) < y(0) < y(2)
top(1) = top(0)
top(2) > bottom(1)
bottom(0) = bottom(2)
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)

DXL

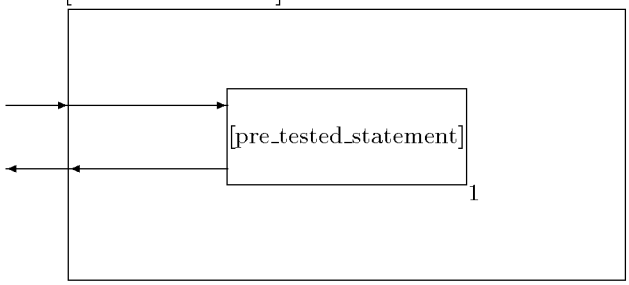
< 並列文 > ::=
'parallel'
< 文の列 >
'end_parallel' ';'

DXL – Production Rule – 38

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)

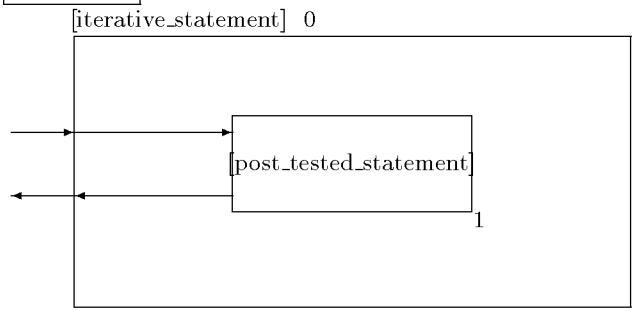
DXL

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

DXL – Production Rule – 39

iterative_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)

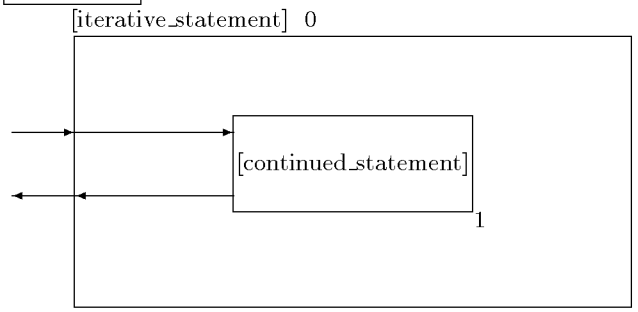
DXL

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

DXL – Production Rule – 40

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)

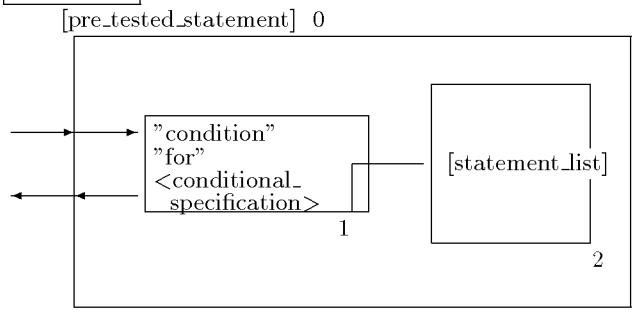
DXL

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

DXL – Production Rule – 41

pre_tested_statement(1)

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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])

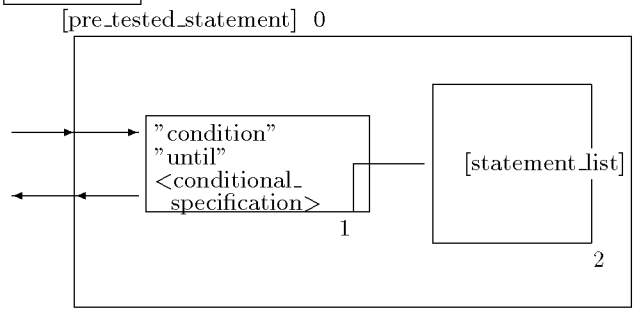
DXL

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

DXL – Production Rule – 42

pre_tested_statement(2)

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

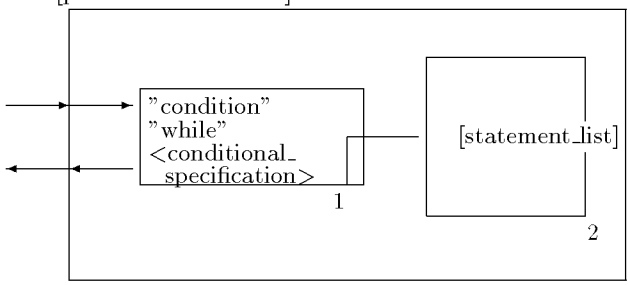
< 前判定繰り返し文 > ::=
'condition'
[<< 前判定繰り返し型 >>]
< 条件仕様記述 >
'loop'
< 文の列 >
'end_loop' ';'
<< 前判定繰り返し型 >> ::= "while" | "until" | "for"

DXL – Production Rule – 43

pre_tested_statement(3)

Production

[pre_tested_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

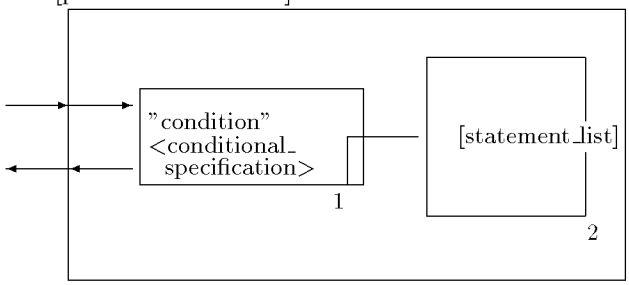
< 前判定繰り返し文 > ::=
'condition'
[<< 前判定繰り返し型 >>]
< 条件仕様記述 >
'loop'
< 文の列 >
'end_loop' ';'
<< 前判定繰り返し型 >> ::= "while" | "until" | "for"

DXL – Production Rule – 44

pre_tested_statement(4)

Production

[pre_tested_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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])

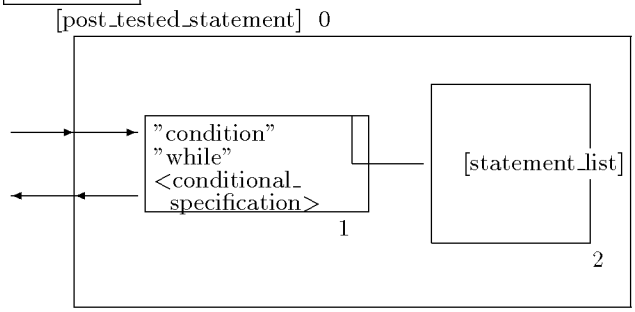
DXL

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

DXL – Production Rule – 45

post_tested_statement(1)

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

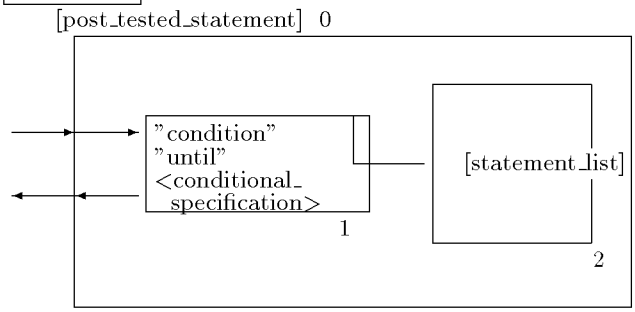
< 後判定繰り返し文 > ::=
'loop' < 文の列 > 'end_loop'
'condition'
[<< 後判定繰り返し型 >>]
< 条件仕様記述 > ';'

<< 後判定繰り返し型 >> ::= "while" | "until"

DXL – Production Rule – 46

post_tested_statement(2)

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

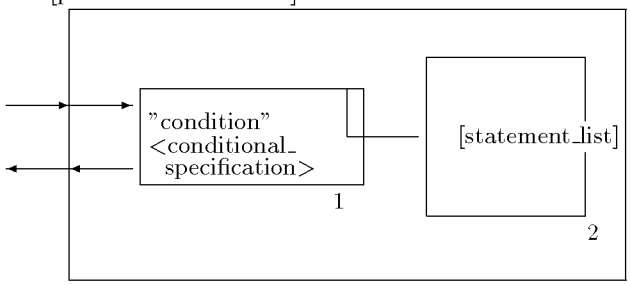
< 後判定繰り返し文 > ::=
'loop' < 文の列 > 'end_loop'
'condition'
[<< 後判定繰り返し型 >>]
< 条件仕様記述 > ';'

<< 後判定繰り返し型 >> ::= "while" | "until"

post_tested_statement(3)

Production

[post_tested_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

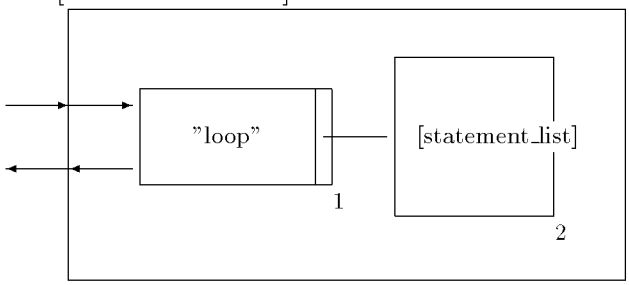
< 後判定繰り返し文 > ::=
'loop' < 文の列 > 'end_loop'
'condition'
[<< 後判定繰り返し型 >>]
< 条件仕様記述 > ';'

<< 後判定繰り返し型 >> ::= "while" | "until"

continued_statement

Production

[continued_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

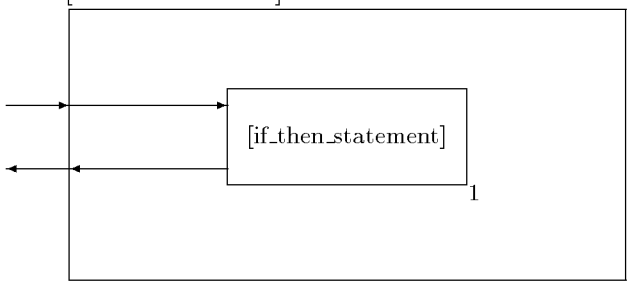
< 継続繰り返し文 > ::=
'loop'
< 文の列 >
'end_loop' ';'

DXL – Production Rule – 49

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

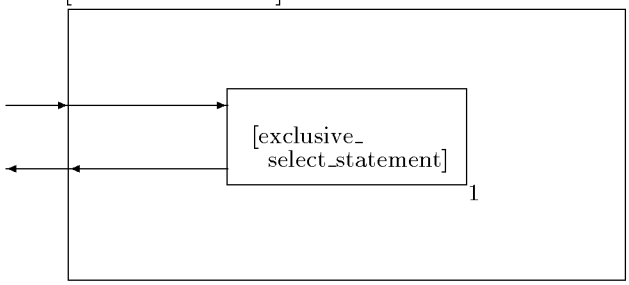
< 選択文 > ::=
< 単純選択文 > | < 多岐選択文 > | < 多重岐選択文 >

DXL – Production Rule – 50

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)

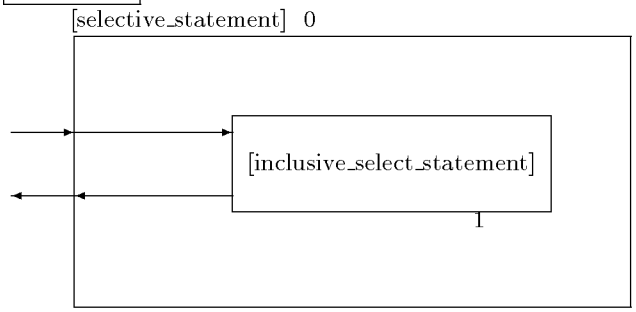
DXL

< 選択文 > ::=
< 単純選択文 > | < 多岐選択文 > | < 多重岐選択文 >

DXL – Production Rule – 51

selective_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)

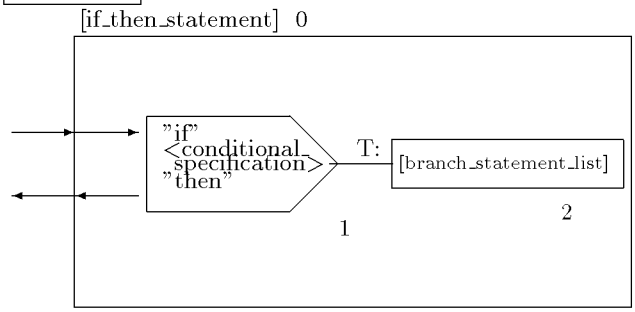
DXL

< 選択文 > ::=
< 単純選択文 > | < 多岐選択文 > | < 多重岐選択文 >

DXL – Production Rule – 52

if_then_statement

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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)
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

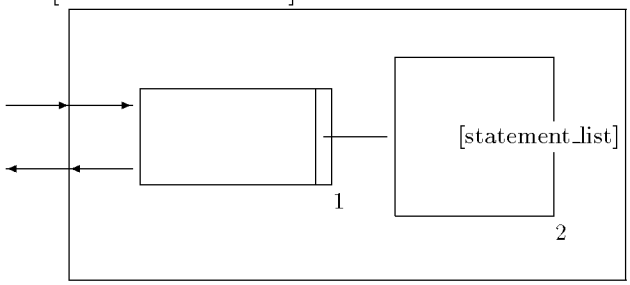
< 単純選択文 > ::= 'if' < 条件仕様記述 > 'then'
< 文の列 >
'end_if'

DXL – Production Rule – 53

branch_statement_list

Production

[branch_statement_list] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

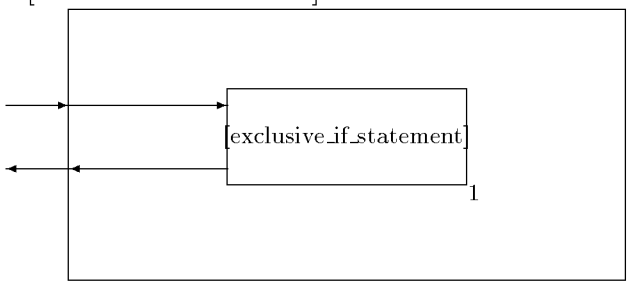
< 文の列 > ::=
[[<< 名札 >> ";"] < 文 >]

DXL – Production Rule – 54

exclusive_select_statement(1)

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

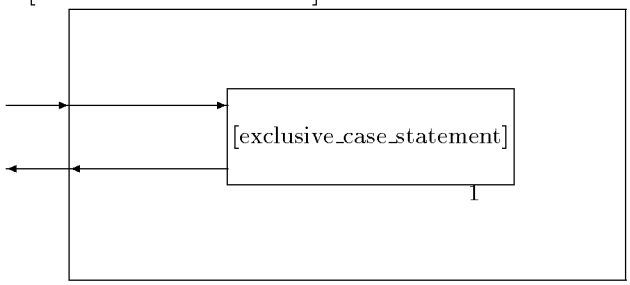
< 多岐選択文 > ::=
<if 型選択文 > | <case 型選択文 >

DXL – Production Rule – 55

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

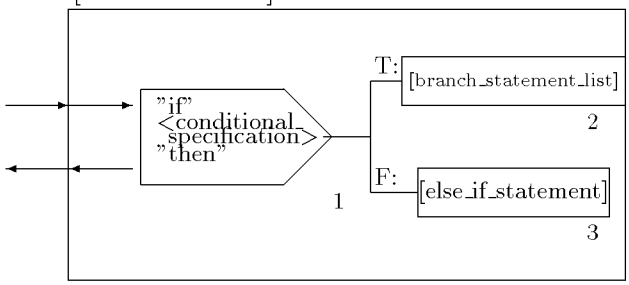
< 多岐選択文 > ::=
<if 型選択文 > | <case 型選択文 >

DXL – Production Rule – 56

exclusive_if_statement(1)

Production

[if_then_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(0) + w(1)
x(3) > x(0) + w(1)
y(2) < y(1) < y(3)
top(2) = top(0)
top(3) > bottom(2)
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

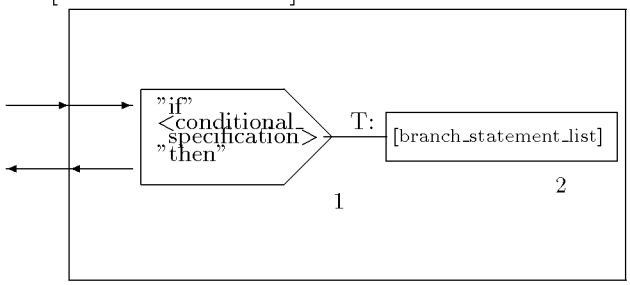
<if 型選択文 > ::= 'exclusive_select'
'if' < 条件仕様記述 > 'then' < 文の列 >
{ 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
['else' < 文の列 >]
'end_exclusive_select' ';'

DXL – Production Rule – 57

exclusive_if_statement(2)

Production

[exclusive_if_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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(["if", <conditional_specification>, "then"])
cell(1) = "exclusive_selection"
string(1) =
get_str(["if", <conditional_specification>, "then"])
lines(1) = get_line_begin(1,[2])

DXL

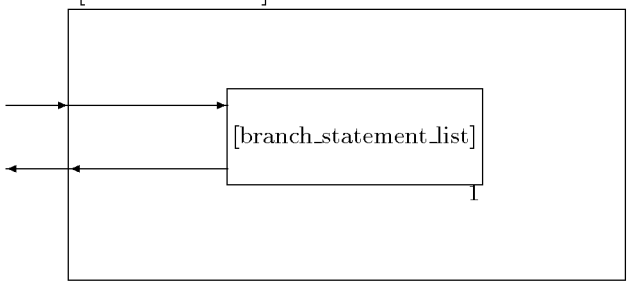
<if 型選択文 > ::= 'exclusive_select'
'if' < 条件仕様記述 > 'then' < 文の列 >
{ 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
['else' < 文の列 >]
'end_exclusive_select' ';'

DXL – Production Rule – 58

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)

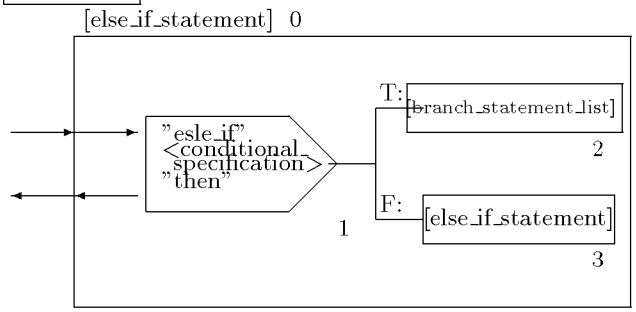
DXL

<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 – 59

else_if_statement(2)

Production



Semantic Rules

x(1) = x(0)
x(2) > x(0) + w(1)
x(3) > x(0) + w(1)
y(2) < y(1) < y(3)
top(2) = top(0)
top(3) > bottom(2)
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])

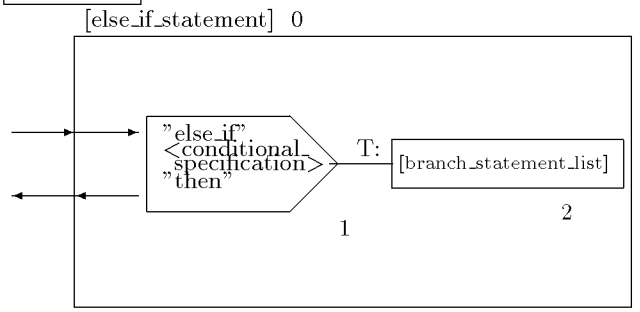
DXL

<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 – 60

else_if_statement

Production



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
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

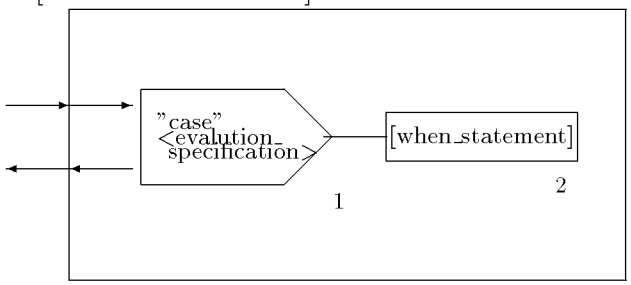
<if 型選択文 > ::= 'exclusive_select'
  'if' < 条件仕様記述 > 'then' < 文の列 >
  { 'else_if' < 条件仕様記述 > 'then' < 文の列 > }
  ['else' < 文の列 >]
  'end_exclusive_select' ';'

DXL – Production Rule – 61

exclusive_case_statement

Production

[exclusive_case_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
y(0) = y(1)
y(1) = 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

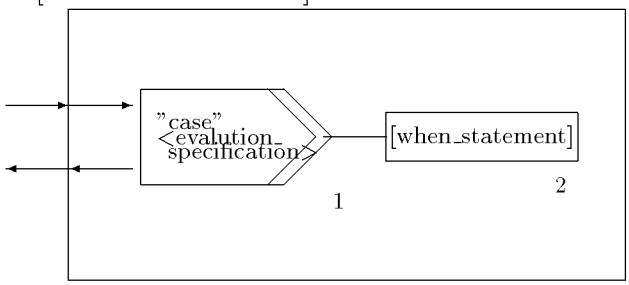
<case 型選択文 > ::= 'exclusive_select'
'case' < 評価式仕様記述 >
{ 'when' < 選択仕様記述 > ':' < 文の列 > }
'end_exclusive_select' ';'

DXL – Production Rule – 62

inclusive_case_statement

Production

[inclusive_case_statement] 0



Semantic Rules

x(1) = x(0)
x(2) > x(1) + w(1)
y(0) = y(1)
y(1) = 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) = "parallel_asynch"
string(1) = get_str(["case", <evalution_specification>])
lines(1) = get_line_begin(1,[2])

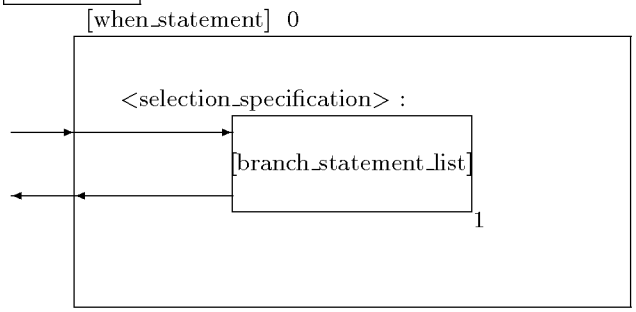
DXL

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

DXL – Production Rule – 63

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

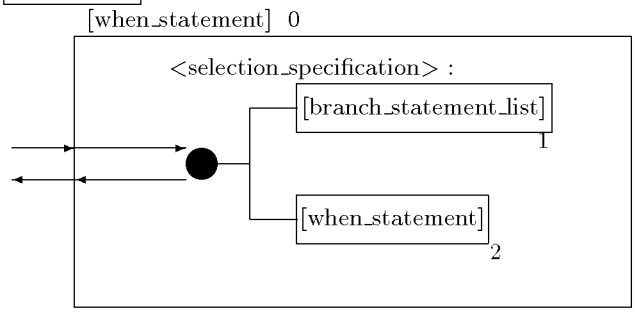
<case 型選択文 > ::= 'exclusive_select'
'case' < 評価式仕様記述 >
{ 'when' < 選択仕様記述 > ':' < 文の列 > }
'end_exclusive_select' ';'

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

DXL – Production Rule – 64

when_statement(2)

Production



Semantic Rules

x(1) = x(0)
x(2) = x(0)
y(1) < y(0) < y(2)
top(1) = top(0)
top(2) = bottom(1) + GapY
bottom(0) = bottom(2)
cl(1) = <selection_specification> ":"
id(1) = id(0)
id(2) = id(1) + nc(1)
nc(0) = nc(1) + nc(2)

DXL

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

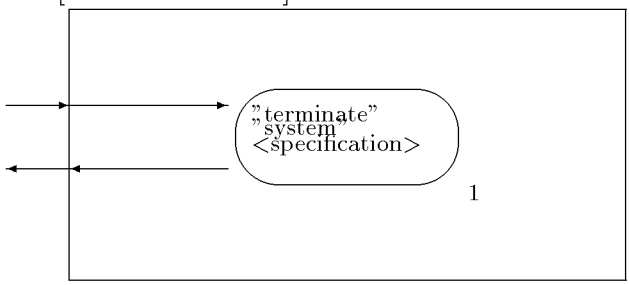
< 多重岐選択文 > ::= 'inclusive_select'
'case' < 評価式仕様記述 >
{ 'when' < 選択仕様記述 > ':' < 文の列 > }
'end_inclusive_select' ';'

DXL – Production Rule – 65

terminate_statement(1)

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”, ”system”, <specification>])
h(1) =
  get_height([”terminate”, ”system”, <specification>])
cell(1) = ”terminal”
string(1) =
  get_str([”terminate”, ”system”, <specification>])

DXL

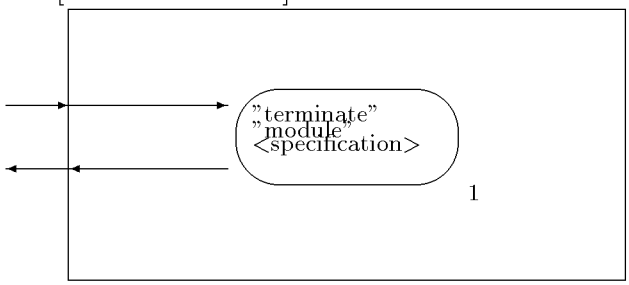
< 打ち切り文 > ::=
  ’terminate’ < 打ち切り対象 > < 仕様記述 > ’;’
< 打ち切り対象 > ::=
  ’system’ | ’module’ | ’block’< ブロック識別子 >

DXL – Production Rule – 66

terminate_statement(2)

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”, ”module”, <specification>])
h(1) =
  get_height([”terminate”, ”,module”, <specification>])
cell(1) = ”terminal”
string(1) =
  get_str([”terminate”, ”module”, <specification>])

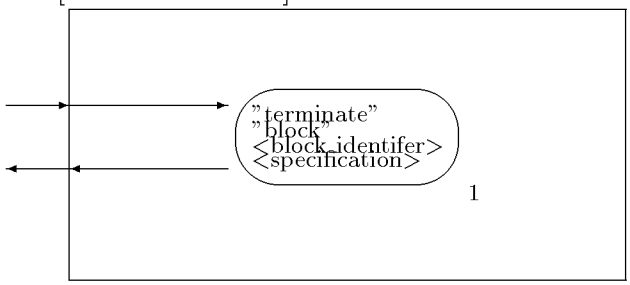
DXL

< 打ち切り文 > ::=
  ’terminate’ < 打ち切り対象 > < 仕様記述 > ’;’
< 打ち切り対象 > ::=
  ’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_identifer>, <specification>])
h(1) = get_height([”terminate”, ”block”, <block_identifer>, <specification>])
cell(1) = ”terminal”
string(1) = get_str([”terminate”, ”block”, <block_identifer>, <specification>])

DXL

< 打ち切り文 > ::= ’terminate’ < 打ち切り対象 > < 仕様記述 > ’;’
< 打ち切り対象 > ::= ’system’ | ’module’ | ’block’< ブロック識別子 >