グラフィクスとCAD　30－1
（１９８７．１１．６）

情報の図化インタフェース
－形式文法を用いた一手法－

鎌田富久　　川合　慧
（東京大学　理学部）

図は人間が一目で直観的に把握できる情報の表現手段であり、今やユーザインタフェースには欠かせない要素である。ユーザが内容を理解しやすい図は扱うデータや目的によって異なるので、様々なデータを様々な表現形式で視覚化することが必要である。しかしながら、データ形式毎にあるいは図の表現形式毎に図化モジュール（プログラム）を作っていたのでは効率が悪い。そこで、図化モジュールを生成するための一般的な手法が必要になる。

本稿では、データ形式を記述する形式言語（文脈自由文法）の生成規則に図化規則を対応させることによって図を生成する手法について述べる。各非終端記号に図形要素を対応させ、各生成規則に図形要素の幾何学的関係を対応させる。このような図化の記述を実現するために、我々はConstraint Grammarという形式システムを新たに考案した。図形要素の幾何学的関係は図形要素のパラメータの間のconstraintによって表現される。また、図化規則にあいまいさを導入し、図の配置を最適化する。さらに、本手法を拡張してネットワーク構造のデータの図化を試みる。

Interface for Visualizing Data
A Grammatical Approach

Tomihisa Kamada and Satoru Kawai

Department of Information Science
Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113 Japan

Pictorial representation of data is an effective way to present various information to users. How to visualize data depends on the form of data and the purpose. So, it is required to build a general method for producing visualization modules (programs) easily for a variety of data forms.

This paper presents a framework in which pictures are generated by the use of layout rules corresponding to the productions of formal language for a given data form. In this method, the non-terminals correspond to the graphical elements such as boxes and circles, and the productions correspond to the geometric relations among the graphical elements. In order to realize such a visualization mechanism, we have introduced the enhanced context-free grammar named Constraint Grammar in which the relations among variables of non-terminals are expressed in terms of constraints. Geometric relations are expressed by the use of such constraints. In addition, we also present an enhancement to our method in order to treat the data of network structure.

1．はじめに

図は、『河図洛書』という言葉があるように、書（文字による表現）と並ぶ情報の表現形式として、人間のコミュニケーションや認識に重要な役割を果たしてきた。このことはグラフィクスが人間とコンピュータのコミュニケーションに必要不可欠な要素であることを示している。ユーザインタフェースの側面から見れば、情報（データ）をユーザに様々な画像（図）で提示することが望まれる。

ユーザが内容を理解しやすい画像は、扱うデータや目的によって異なるので、様々なデータを様々な表現形式で視覚化することが必要である。データを図化する試みは、数値データなどを様々なグラフや表で表現するとか、あるいはプログラムをフローチャートで表現するといった個別的な研究がほとんどである。しかしながら、幅広い要求にこたえるユーザインタフェースを構築するためには、このようなデータ形式や図の表現形式に大きく依存したアプローチではなく、一般的にこの問題を捉えて効率よく図化モジュールを生成するための方法論を確立すべきである。

我々はこの問題を広い視野で捉え、データを言語表現から図表現に変換する一般的な枠組みを研究している。本稿では、図化インタフェースの１つの手法として、データ形式を記述する形式言語（文脈自由文法）の生成規則に図化規則を対応させる方法について述べる。

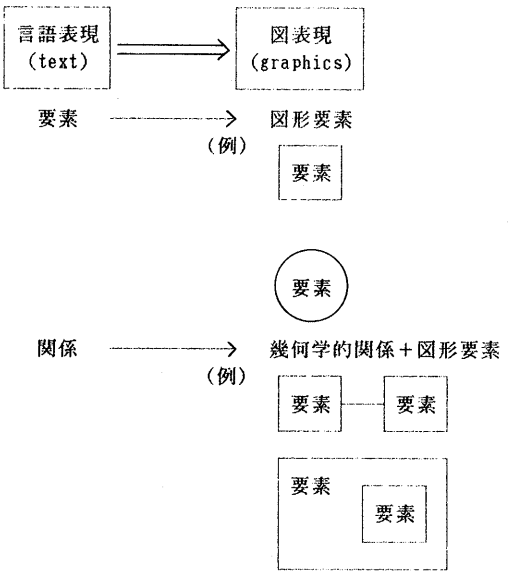
2．関係構造の図化

情報（データ）を図で表現すると内容を把握しやすいことがある。これは、言語表現（形式言語）が特定の規則に従った１次元の単語（記号）の配列であるのに対し、図表現は２次元の広がりを持っており、情報の持つ複数の関係を人間が自然に理解できる空間的な位置関係や図形として表現できるからである。

情報の持つ関係構造は、要素と要素間の関係という枠組みで考えることができる。従って、関係構造を図化するには要素と関係の図の世界への写像を与えればよい。そこで、要素は図形要素（長方形、円、・・・）に対応させ、関係は幾何学的な関係（並列・直列配置、包含、・・・）に対応させる。関係の写像には線で結ぶなどの付加的な要素も含める。また、要素の性質を表すような関係は図形要素の属性（大きさ、色、形、濃淡、・・・）に対応させることもある。

このような写像を目的に応じて変えることによって、関係構造を様々な図で表現することができる。また、どのような図が人間にとって分かりやすいかという問題には深入りすることはせず、図を生成するための道具を提供することを目的とする。

図形要素の幾何学的な関係は図形要素のパラメータの間の拘束条件(constraint)で表現される。このように図を記述するのに、図形の位置や大きさを直接指定するのではなく、図形間のconstraintで記述しようという試みはＩＤＥＡＬ［１］、ＰＩＣ［２］、Ｊｕｎｏ［３］などで見られる。また、オブジェクト指向環境で拘束条件をベースとしたグラフィクス・インタフェースを実現したものにThingLab［４，５］やＧＲＯＷ［６］がある。（拘束条件を図形の記述に取り入れた初期のシステムにSketchpad［７］やMETAFONT［８］がある。）ＶＬＳＩ設計の分野でも、シンボリック・レイアウトに拘束条件を用いて配置を決めるという考えが取り入れられている［９，10］。



我々は、constraintによる図形の記述とデータを記述する形式文法を結び付けるために、Constraint Grammarという形式システムを新たに考案し、図化に応用する。

3．Ｃｏｎｓｔｒａｉｎｔ　Ｇｒａｍｍａｒ

ある文法（文脈自由文法）で記述されたデータをうまく図化するためには、データの内部構造を図の配置に反映させる必要がある。そこで、文脈自由文法に機能を追加して図化の枠組み（主にレイアウト）を実現することを考える。具体的には、各非終端記号に図形要素を対応させ、各生成規則に図形要素の幾何学的関係を対応させる。

文脈自由文法に意味規則を追加した文法には属性文法(Attribute Grammar)［11,12］があるが、属性文法ではパラメータ（属性）の依存関係が１方向に固定されるため、複数の関係式で互いに依存し合う属性を記述するようなことはできない。図形の配置では、互いに依存するパラメータが重要で属性文法では記述しにくい。そこで、パラメータの依存関係の表現において、属性文法よりも

さらに一般的な文法Constraint Grammarを導入する。

Constraint Grammarでは、属性を計算する式（関数）を生成規則に対応させるのではなく、属性の関係式(constraint)を生成規則に対応させる。関係式は属性計算の順序を規定してはいない。従って、合成属性と相続属性のような区別はない。そこで、各非終端記号に付随するパラメータを属性と呼ぶのはやめて、変数と呼ぶことにする。また、グローバルな変数も導入する。

Constraint Grammarは次のような３つ組み＜Ｇ，Ｖ，Ｃ＞で定義される。

１．文脈自由文法　Ｇ＝＜Ｎ，Ｔ，Ｐ，Ｓ＞
　Ｎ：非終端記号の集合
　Ｔ：終端記号の集合
　Ｐ：生成規則の集合
　　Ｐ　ｐ：Ｘ０ → Ｘ１ Ｘ２ … Ｘｎ
　Ｓ：初期記号

２．Ｖ(Ｘ)：非終端記号Ｘ　Ｎに付随する変数の集合
　また、グローバル変数の集合をＶ０とする。

３．Ｃ(ｐ)：生成規則ｐ　Ｐに付随するconstraintの集合
　ｐ：Ｘ０ → Ｘ１ Ｘ２ … Ｘｎ のとき、Ｃ(ｐ)は変数の集合Ｖ(Ｘ０)∪Ｖ(Ｘ１)∪Ｖ(Ｘ２)∪…∪Ｖ(Ｘｎ)∪Ｖ０の間のconstraintの集合。

Constraint Grammarの文の解析は、文脈自由文法Ｇに従って構文解析をし、解析木の各ノードの変数にconstraintを張り巡らすことによって行われる。そして、Constraint Solver によって変数の値を求める。Constraint Solverはどのようなconstraintを許すかによっていろいろなもの（単純に連立一次方程式解くものから、高次の方程式や不等式を解くものまで）が考えられる。

Constraint Grammarは計算の効率を別にすれば、完全に属性文法をシミュレートできる。

４．Constraint Grammarを用いた図化システム

図を構成する基本的な図形要素Point、Box、Circleなどを変数の集合として定義しておいて、Constraint Grammarの非終端記号に付随させる。図形のレイアウトでは、解析木の親子関係や兄弟関係での図形の関係づけはもちろん重要だが、どうしても解析木上では離れているノード間の関係づけも必要になる。これを実現するために、１つの非終端記号に対応づけられた図形要素（変数の集合）が次々に非終端記号に受け渡されて行く機構を入れた。

簡単な例として、科学の分類の図化について説明する。データが次のような形式で表現されているとする。

```
Science { Mathematics { Algebra Analytics
Geometry } Physics { Dynamics { Hydrodynamics
} Electronics Optics } Chemistry { Organic
Inorganic } Biology }
```

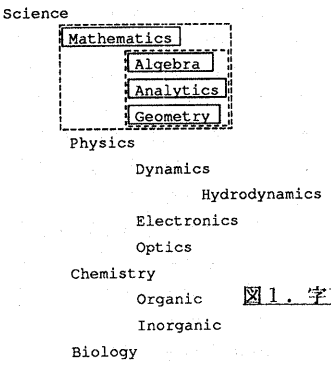


図１．字下げによる階層表現

このデータを図１のように表現するとしよう。図１が示すように、長方形を階層的に配置することで図が構成されている。そこで、すべての非終端記号に図形要素Boxを対応させる。

S(Box), TREE(Box), SUBTREE(Box), ITEM(Box)

図１を生成するには次のようなConstraint Grammarを与えればよい。

```
(1) S(%1) → TREE(%1)
      { %1.xpos = %1.ypos = 0 }
(2) TREE(%1) → ITEM(%2) '{' SUBTREE(%3) '}'
      { Comp_Over(%1,%2,%3,left_indent) }
(3) TREE(%1) → ITEM(%1)
(4) SUBTREE(%1) → TREE(%2) SUBTREE(%3)
      { Comp_Over(%1,%2,%3,left_align) }
(5) SUBTREE(%1) → TREE(%1)
(6) ITEM(%1) → <name>
      { Box(%1,enclose,<name>) }
```

Box は４つの変数xpos、ypos、xsiz、ysizを持つ。生成規則で%?が変数の集合である図形要素を表し、１つの規則内で同じ%?は同一視する。図形要素の関係は変数の間のconstraintで表され、例えば、Comp_Over は次のようなconstraintの集合のマクロである。

```
Comp_Over(%1,%2,%3,mode) {
      %1.ypos = %3.ypos
      %3.ypos + %3.ysiz + vs = %2.ypos
      %1.ysiz = %2.ysiz + %3.ysiz + vs
  case(mode) {
    left_indent:
      %1.xpos = %2.xpos
      %3.xpos = %2.xpos + hi
      %1.xsiz = max(%2.xsiz, %3.xsiz + hi)
    left_align:
      %1.xpos = %2.xpos
      %2.xpos = %3.xpos
      %1.xsiz = max(%2.xsiz, %3.xsiz)
    ..............................
  }
}
```

Comp_Over(%1,%2,%3,mode)はBox %3の上にBox %2を置き（modeがleft_indent のときは左側を字下げし、modeがleft_alignのときは左側を揃える）、両者を囲む大きな長方形がBox %1になるように関係づける。また、Box(%1,enclose,<name>)はBox %1が文字列<name>を囲むように関係づける。従って、初期記号S に付随するBox は図全体を囲む長方形になるように関係づけられる。

このように、図形と図形をある規則に従って配置し全体を１つの図形とするという操作を再帰的に行うことで図を生成する考え方はＥＱＮ [13] やＶＬＳＩのシンボリック・レイアウト [10] にも見られる。

次に、図２のようなトリー状の表示をするには非終端記号SUBTREE をSUBTREE(Box,Box)にし、次のような文法で記述する。

```
(1') S(%1) → TREE(%1)
       { %1.xpos = %1.ypos = 0 }
(2') TREE(%1) → ITEM(%2) '{' SUBTREE(%2,%3) '}'
       { Comp_Over(%1,%2,%3,center) }
(3') TREE(%1) → ITEM(%1)
(4') SUBTREE(%1,%2) → TREE(%3) SUBTREE(%1,%4)
       { Comp_Left(%2,%3,%4,top_align)
         Line_Connect(%1,%3,bottom_to_top) }
(5') SUBTREE(%1,%2) → TREE(%2)
       { Line_Connect(%1,%2,bottom_to_top) }
(6') ITEM(%1) → <name>
       { Box(%1,enclose,<name>) }
```

生成規則(2')でComp_Over(%1,%2,%3,center)はBox %3の上の中央にBox %2を置き、両者を囲む長方形がBox %1になるように関係づける。また、上に置かれるBox %2を線で結ぶために非終端記号SUBTREE に受け継がせる。生成規則(4')でComp_Left(%2,%3,%4,top_align) はBox %3をBox %4の左に上部を揃えて置き、両者を囲む長方形全体がBox %1になるように関係づける。また、Line_Connect(%1,%3,bottom_to_top) はBox %1の下部とBox %3の上部を線で結ぶように関係づける。

図２(a)と(b)のようなトリー階層表示の違いは、単に図形を結ぶ線の引き方が違うと考える。つまり、Line_Connectの違いと考える。

このように図化規則（constraintによる図形要素の関係づけ）を目的に応じて変えることによって、簡単に望むべき図表現が得られる。また、図形要素や図化規則はユーザが新しく追加でき、目的別に図化ライブラリーを構築することができる。

５．あいまいさの導入と最適化

分かりやすい図を生成するためには、ローカルな関係づけだけではなく、全体的なバランスを考慮する必要がある。そこで、図形要素間のconstraintにＯＲ条件を許し、ある評価式が最大あるいは最小になるにconstraintを選択する。

例として、ＬＩＳＰのリスト構造を図化することを考える。アトムを長方形のセル（CAR部とCDR部を持つ）に対応させ、CDR部のつながりを右矢印で、CAR部のつながりを下矢印で表すことにする。そこで、非終端記号には次のように図形要素を付随させる。

S(Box), LIST(Box), SERIES(Box), ATOM(Cell)

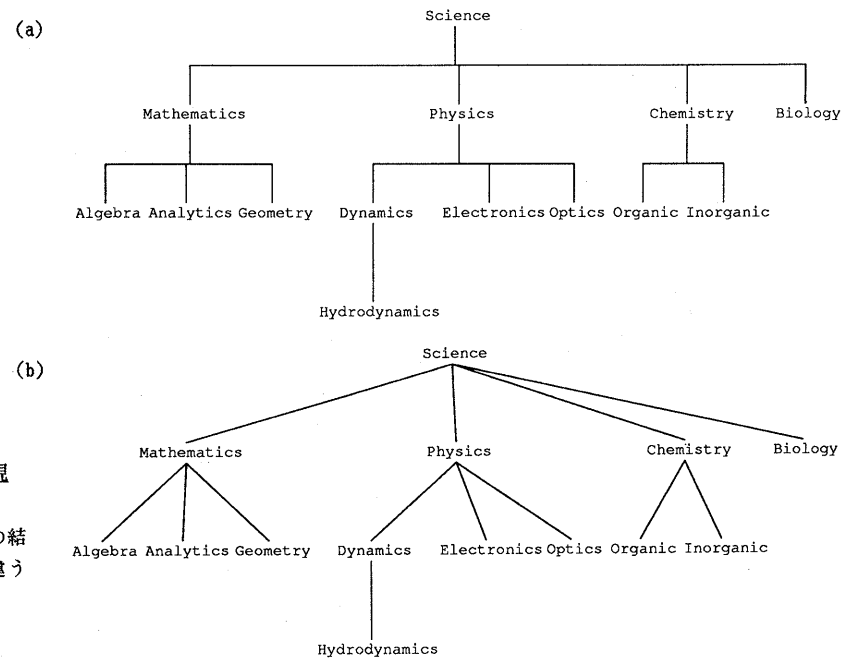


図２．トリー状の階層表現

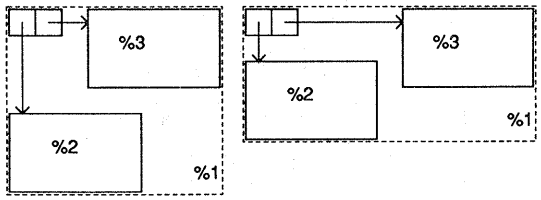
(a)と(b)の違いは単に線の結び方（Line_Connect）が違うと考える。

Cellは長方形のセルで予め図形として定義してあるものとする。従って、リストの形式（単純化してある）を記述する次のような文法に図化規則を付け加えればよい。

```
(1)  S(%1) → LIST(%1)
       { %1.xpos = %1.ypos = 0 }
(2)  LIST(%1) → '(' SERIES(%1) ')'
(3)  SERIES(%1) → LIST(%2) SERIES(%3)
       { Cell(%4)
         Comp_Left(%5,%4,%3,top_align)
         Comp_Over(%1,%5,%2,left_align)
         Arrow_Connect(%4,%3,h_arrow)
         Arrow_Connect(%4,%2,v_arrow) }
(4)  SERIES(%1) → ATOM(%2) SERIES(%3)
       { Comp_Left(%1,%2,%3,top_align)
         Arrow_Connect(%2,%3,h_arrow) }
(5)  SERIES(%1) → LIST(%2)
       { Cell(%3) Nil_Bar(%3)
         Comp_Over(%1,%3,%2,left_align)
         Arrow_Connect(%3,%2,v_arrow) }
(6)  SERIES(%1) → ATOM(%2)
       { Nil_Bar(%2) Box_Contain(%1,%2) }
(7)  ATOM(%1) → <name>
       { Cell(%1,put,<name>) }
```

生成規則(3)は図３(a)のように、まずBox %3の左に新しいCell %4 を置き、両者を囲む長方形がBox %5になるように関係づける。そして、Box %2の上にBox %5を置き、両者を囲む長方形がBox %1になるように関係づける。つまり、CDR部を最初に結合し次にCAR部を結合する。また、Arrow_Connect はCellからBoxに矢印（h_arrowは右矢印でv_arrowは下矢印）を引くことを表す。Nil_Barはセルの CDR部に斜め線を引くことを表す。その結果、図４(a)のようなリストの構造図が得られる。

図４(a) を見ると、入れ子の深いリストでは縦長な図になってしまいそうである。そこで、生成規則(3) の図化規則のComp_LeftとComp_Overを、図３(b)のようにCAR部から結合するように変えると、



(a) CDR部から結合する　(b) CAR部化ら結合する

図３．CAR部とCDR部の結合の図化

```
Comp_Over(%5,%4,%2,left_align)
Comp_Left(%1,%5,%3,top_align)
```

図４(b) のような図が得られる。しかし、今度は入れ子の浅いリストでは横長な図になってしまう。

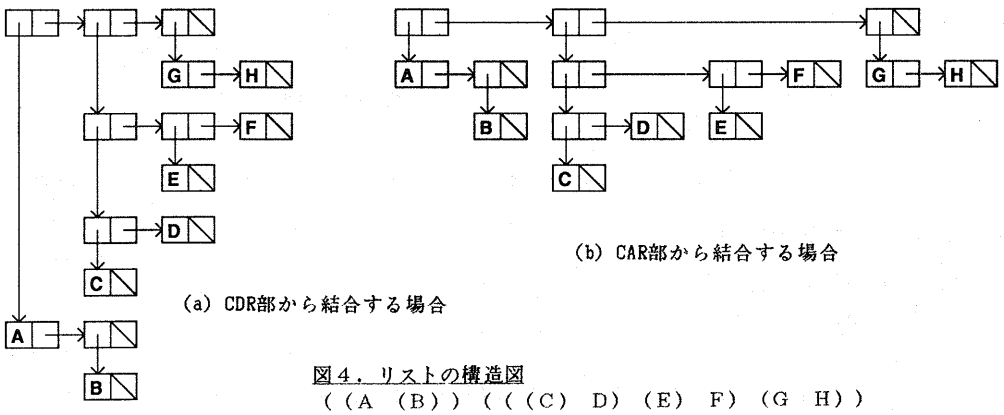
このような場合、入力データに依存して図化規則の一部を選択するようにしたい。この要求に答えるために図化規則にあいまいさを導入した。つまり、上の２種類の図化規則をＯＲで結んで記述することができる。

```
{ Comp_Left(%5,%4,%3,top_align)
  Comp_Over(%1,%5,%2,left_align) }
OR
{ Comp_Over(%5,%4,%2,left_align)
  Comp_Left(%1,%5,%3,top_align) }
```

そして、複数の図化規則はある評価式が最大あるいは最小になるように選択される。リストの構造図では、例えば図全体の長方形（Sに付随するBox）の縦と横の長さの差が最小になるように指定すると図５が得られる。このとき、生成規則(1) で次のような指定をする。

```
minimize { abs( %1.xsiz - %1.ysiz ) }
```

このように図化規則にあいまいさを導入し、図全体のバランスを考えた最適化を行うことによって、さらに柔軟なデータの図化が実現される。



(a) CDR部から結合する場合

(b) CAR部から結合する場合

図４．リストの構造図

（（A （B））（（（C） D） （E） F） （G H））

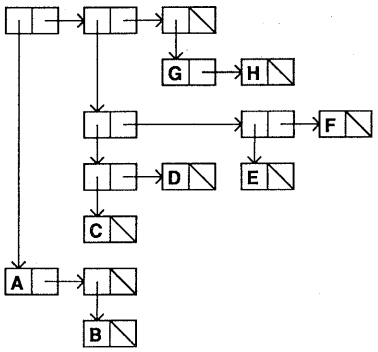


図５．最適化したリストの構造図
（データは図４と同じ）

６．ネットワーク型データへの拡張

データがある文脈自由文法（ＢＮＦ形式）で記述されている場合には上述の手法は有効であるが、データが１つの文脈自由文法で記述されていない（あるいは記述できない）場合には直接この手法を使うことはできない。データがある文法で記述されているならば、そのデータの内部構造は木構造になる。しかし、一般的にはデータの構造は必ずしも木構造ではなく、ネットワーク（グラフ）構造である。そこで、このようなネットワーク構造のデータを扱うために、上述の手法を拡張することを考える。

例えば、上の科学の分類のデータが次のように５つに分けて記述されている場合を考えよう。

```
Science { Mathematics Physics Chemistry Biology}
Mathematics { Algebra Analytics Geometry }
Physics { Dynamics Electronics Optics }
Chemistry { Organic Inorganic }
Dynamics { Hydrodynamics }
```

この場合、同じ名前の終端記号は１つの図形要素で図化したい。通常の文脈自由文法では終端記号はすべて別のものとして扱われるので、名前の一致する終端記号を同一視し、それに対応する図形要素を１つの図形要素として扱う機構を付け加える。

上の５つのデータを１つ１つ解析する文法を考えて、解析の結果得られる５つの解析木を終端記号を同一視することで結び付ける。図２と同じ図を生成するには次のような文法を考えればよい。

```
S(Box), NODE(Box,Box), ITEMS(Box,Box), ITEM(Box)
(1) S(%1) → NODE(%1,%2) '{' ITEMS(%2,%3) '}'
      { Comp_Over(%1,%2,%3,center) }
(2) NODE(%1,%2) → <name>
      { Box(%2,enclose,<name>) }
(3) ITEMS(%1,%2) → ITEM(%3) ITEMS(%1,%4)
      { Comp_Left(%2,%3,%4,top_align)
        Line_Connect(%1,%3,bottom_to_top) }
(4) ITEMS(%1,%2) → ITEM(%2)
      { Line_Connect(%1,%2,bottom_to_top) }
(5) ITEM(%1) → <name>
      { Box(%1,enclose,<name>) }
```

この文法で分類のデータの最初の２行を解析した解析木は図６のようになる。このとき、解析木(a)と解析木(b)の終端記号Mathematics を同一視して、解析木(a) の非終端記号ITEMに付随するBox %1を解析木(b) の非終端記号NODEに付随するBox %1と同一視したい。そうすれば、NODEに付随するBox %1は解析木(b) で全体を囲む長方形に関係づけられるので、解析木(a)ではMathematicsの子供の広がりを考慮した横の配置が可能になる。

つまり、同一の終端記号が見つかると、次のような解析木をまたがる関係づけが行われる。

```
ITEM(%1) → NODE(%1,%2)
```

このような終端記号の同一視と解析木をまたがる関係づ

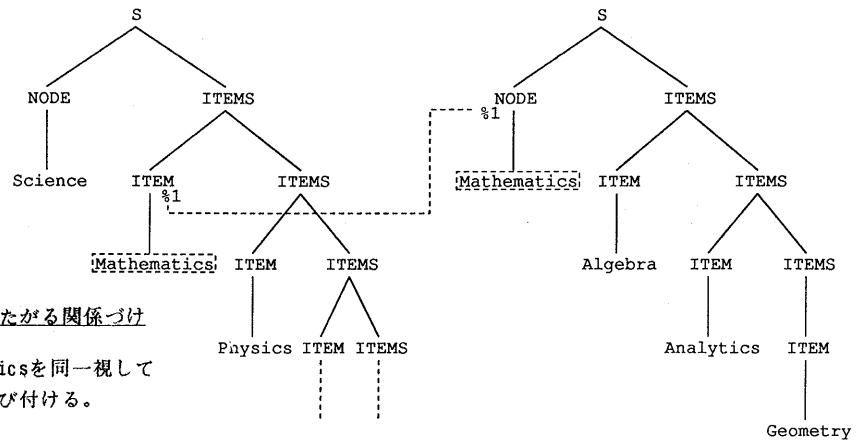


図６．解析木をまたがる関係づけ

終端記号Mathematicsを同一視して
２つの解析木を結び付ける。

(a) Science { Mathematics ... } の解析木　(b) Mathematics { Algebra ... } の解析木

けは複数の文法の間でも同様に行うことができ、ネットワーク状の内部構造にconstraintを張り巡らすことができる。

7. おわりに

文脈自由文法で記述できるデータについて、生成規則に図化規則を対応させてデータを図化する手法について述べた。そして、データの内部構造がネットワーク状のものを扱うための拡張について述べた。

図形要素間の幾何学的関係を記述するために我々が新たに考案したConstraint Grammarは、非終端記号に付随する変数の間にconstraintを張り巡らす汎用的な形式システムなので、レイアウト以外にも適用できる。

また、constraintの集合から変数の値を求めるConstraint Solver は独立した問題として捉え、不等式などを含む複雑なconstraintを解くことが今後の課題である。

参考文献

1. C.J.Van Wyk,"A High-Level Language for Specifying Pictures," ACM Trans. on Graphics, vol.1, no.2, pp.163-182, Apr. 1982.
2. B.W.Kernighan,"PIC - A Language for Typesetting Graphics," Software-Practice & Experience, vol.12, no.1, pp.1-21, Jan. 1982.
3. G.Nelson,"Juno, a constraint-based graphics system," SIGGRAPH, vol.19, no.3, pp.235-243, Jul. 1985.
4. A.Borning,"The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory," ACM Trans. on Prog. and Systems, vol.3, no.4, pp.353-387, Oct. 1981.
5. A.Borning and R.Duisberg,"Constraint-Based Tools for Building User Interfaces," ACM Tran. on Graphics, vol.5, no.4, pp.345-374,Oct.1986.
6. P.S.Barth,"An Object-Oriented Approach to Graphical Interfaces," ACM Trans. on Graphics, vol.5, no.2, pp.142-172, Apr. 1986.
7. I.E.Sutherland,"Sketchpad: A Man-Machine Graphical Communication System," Proc. SJCC, pp.329-346, 1963.
8. D.E.Knuth,"TEX and METAFONT," Digital Press, 1979.
9. J.A.Roach,"The Rectangle Placement Language," Proc. ACM IEEE 21st Design Automation Conf., pp.405-411, Jun. 1984.
10. J.M.Mata,"ALLENDE: A Procedural Language for the Hierarchical Specification of VLSI Layouts " Proc. ACM IEEE 22nd Design Automation Conf., pp.183-189, 1985.
11. D.E.Knuth,"Semantics of Context-Free Languages ," Math. Sys. Theory, vol.2, no.2, pp.127-145, 1968.
12. A.V.Aho, R.Sethi and J.D.Ullman,"Compilers - Priciples, Techniques, and Tools," Addison-Wesley, 1986.
13. B.W.Kernighan and L.L.Cherry,"A System for Typesetting Mathematics," CACM, vol.18, no.3, pp.151-157, Mar. 1975.
14. ジャック・ベルタン（森田喬　訳）：図の記号学、地図情報センター、1982.
15. 出原栄一：コンピュータグラフィック論（吉川弘之編）第３章「人間と図形言語」、日科技連、1977.
16. 出原、吉田、渥美：図の体系－図的思考とその表現 日科技連、1986.