第 2 章

レイアウト制約

図形による視覚的表現は直感的に理解が容易であり情報伝達の手段として有効であるが、複雑な構造や関係を持つ図形の場合は可読性や美しさを考慮してレイアウトしないと図形による視覚的表現のメリットを十分に活かすことができない。

レイアウト制約はユーザにより図形の文法にレイアウトの規則として定義され、図形がその文法により解釈された後、もともとの図形要素間の複雑な構造や関係を分かりやすくバランス良く表す位置関係を作り出す制約である。

2.1 軟かいレイアウト制約

我々は、レイアウト制約として軟かいレイアウト制約と硬いレイアウト制約の 2 種類を考える。

軟かいレイアウト制約は、図形の全体を自動描画アルゴリズムに従ってバランスよく分りやすくレイアウトする制約である。一方、硬いレイアウト制約は、図形の座標や図形間の距離などの制約を具体的に与えたい場合に用いる。

軟かいレイアウト制約と硬いレイアウト制約について述べる前に、図形言語の文法を定義するために用いられる Constraint Multiset Grammars （CMG）、グラフ描画アルゴリズムに関して説明する。

2.1.1 CMG

CMG[3] [7] は終端シンボルの集合、非終端シンボルの集合、開始シンボル、および、生成規則の集合から構成される。すべての終端シンボルと非終端シンボルは、

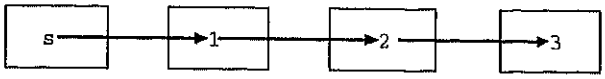

図 2.1: リスト構造

色、大きさ、位置などの属性を持っている。生成規則はトークン（終端シンボルもしくは非終端シンボルのインスタンス）のマルチセットとそれらの属性間の関係を保つ制約により構成される。ここで、マルチセットを用いる理由は、同じ種類のトークンでもそれぞれを別々のトークンとして扱うためである。本論文では右辺から左辺への書き換えを生成と呼ぶ。生成規則は次のように定義される。

$$
\begin{aligned}
T(\vec{x}) \;::=\; & T_1(\vec{x_1}),\cdots,T_n(\vec{x_n})\ where \\
& exists\ T'_1(\vec{x'_1}),\cdots,T'_m(\vec{x'_m}) \\
& where\ C(\vec{x_1},\cdots,\vec{x_n},\vec{x'_1},\cdots,\vec{x'_m})\ and \\
& \vec{x} \;=\; F(\vec{x_1},\cdots,\vec{x_n},\vec{x'_1},\cdots,\vec{x'_m})
\end{aligned}
$$

すなわち、トークンのマルチセット T_1, …, T_n （normal の構成要素）、T'_1, …, T'_m （exist の構成要素）の属性が制約 C を満たす場合、T_1, …, T_n が非終端シンボル $T(\vec{x})$ に書き換えられることを意味している。制約は、生成規則の構成要素になっているシンボルの属性間に成り立っている関係を表すものであり、生成規則が適用されると、構成要素の属性間に制約が課せられ、それらの関係を保存したまま図形の編集を可能にするのに用いられる。exist の構成要素は、定義したい $T(\vec{x})$ を認識するために、存在する必要がある構成要素である[1]。F は、構成要素の属性 $\vec{x_1},\cdots,\vec{x_n}$ と $\vec{x'_1},\cdots,\vec{x'_m}$ を引数とする関数であり、定義中の非終端シンボルの属性に値を与えることを定義している。

図 2.1のようなリスト構造を考えてみる。リスト構造の文法を定義するためには、次のように「再帰的に定義する生成規則」が必要になる。

生成規則 1 四角の中心にラベルとして s が書いてあるものをリストとする。

[1] その他に CMG は構成要素として not_exist、all などを持っている。詳しくは文献 [3] [5] [7] を参照。

生成規則 2 一つのリストが矢印によって四角につながれ、その四角の中心にラベルとして数字が書いてあるものをリストとする。

これらを CMG で記述すると次のようになる。

```
 1:list(point mid) ::= R:rectangle, T:text
 2:    where (
 3:       R.mid  == T.mid  &&
 4:       T.text == ''s''
 5:    ) {
 6:    mid = R.mid;
 7: }
 8:
 9:list(point mid) ::= R:rectangle, T:text,
10:            L:line, LL:list
11:    where (
12:       R.mid   == T.mid   &&
13:       R.mid   == L.end   &&
14:       LL.mid  == L.start
15:    ) {
16:    mid = R.mid;
17: }
```

生成規則 1 は図 2.1のラベル s の四角をリストとして解釈するための生成規則で、1 行目から 7 行目までが CMG の定義である。1 行目は四角 (R) とテキスト (T) で構成されている非終端シンボル「リスト (list) 」を定義している。リストは、属性として中心 (mid) を持つ。2、3、4、5 行目は、リストの構成要素間の制約を定義している。3 行目は、四角の中心 (R.mid) とテキストの中心 (T.mid) が等しいという制約を表す。4 行目は、テキストの文字列 (T.text) が「s」であることを表している。この二つの制約を満たすときに 6 行目が行われる。6 行目はリストの中心として四角の中心の値を代入することを表している。

生成規則2はラベルが付いている四角に一つのリストがつながっているものをリストとして解釈するための生成規則で、9行目から17行目までがCMGの定義である。13行目は四角の中心と直線の終点（L.end）が一致する制約、14行目はリストの中心（LL.mid）と直線の始点（L.start）が一致する制約を意味している。

2.1.2 グラフ描画アルゴリズム

図による視覚的な表現はインターフェースとして最も基本的なものの一つであり、使われる範囲も広く、簡便で親しみやすいという特徴を持っている。よって設計図、家系図、論理回路図など日常的にも多く利用されている。なかでも我々が普段、研究や仕事に扱う事の多い、フローチャート、組織図、システム構成図などは実体をノード、実体間の関係をエッジとしたグラフとして表現される。またこういったグラフはシステム工学、情報工学、ソフトウェア工学など様々な分野において、基礎的なモデルとして広く使われている。最近ではコンピュータの発達にともない、グラフを視覚的に表示したり、操作したりする事が強く求められるようになってきている。そのためこのような図の自動レイアウトを行う、グラフ描画アルゴリズムが数多く提案されている。

グラフ描画アルゴリズムが対象とするグラフは、その図的性質によって様々である。それゆえグラフの種類によって美的基準もまちまちである。そこで各クラスの特徴を考慮した描画法がグラフの種類ごとに開発されている。

グラフでいう美的基準とは、描画規約と描画規則のことである[1]。描画規約はノードとエッジに関する基本的約束であり、描画の際に必ず満たされる制約である。描画規則は“良い”描画の基準となるものである。ただどのようなグラフが“良い”かは個人により変化し、主観による部分が大きい。しかし、グラフの構成はノードとエッジというように、極度に単純化できるため、細かい違いはあるものの、グラフの性質からくる、共通で、基本的ないくつかの良い描画の基準を識別する事ができる。それらの描画規則の例をいくつか挙げる。

- エッジの折れ点数を最少とする
- エッジの交差数を最少にする
- 対称性 (がある場合) を顕示する

- ノードとエッジの配置や配線の密度を一様化する
- 子ノードを対称に配置する
- 階層構造を垂直あるいは水平に顕示する

グラフ描画アルゴリズムとは一般的に、対象とするグラフの特長から優先度の高い順に、これらの描画規則を取り出し、最適基準または制約条件とする。そして描画規約を最適化問題のゴールとして、複数のステップにおいて順次最適化問題を解くことにより、多くの描画規則を満たしていくものである。しかし、木などの交差の無いグラフを除くと、規則を満たせる効率的な方法が無いことが多く、最適解を得ることは困難なことが多い。したがって、ヒューリスティクスを用いた種々の発見的方法によるアルゴリズムも開発されてきている。

グラフ描画アルゴリズムが対象とするグラフは、木、有向グラフ、無向グラフ、複合グラフの各クラスに分類される。

木は、もともと平面グラフなのでエッジの交差なしにレイアウトされる。また、木は組織図のような階層構造を表すのによく用いられる。有向グラフは、各エッジの方向を一定方向の流れとして階層的に、または、各エッジを区別してレイアウトされることが多い。無向グラフは、交差するエッジがないように平面に描くことができる。また、無向グラフはグラフ理論、グラフアルゴリズム、さらにグラフ描画法において最もよく研究されている基礎的かつ重要なクラスである。木、有向グラフ、無向グラフは、ノード間の隣接関係を表すグラフに対して、複合グラフは隣接関係と包含関係を表すグラフである。複合グラフは、様々な分野において人間の思考を助長するための道具として用いられる。

2.1.3 軟かいレイアウト制約とは

軟かいレイアウト制約とは、図形の全体を自動描画アルゴリズムに従ってバランスよくレイアウトする制約である。

ユーザにより図形の特定な部分がレイアウトされても図形の全体を考えるとバランスよく、または、見やすくレイアウトされてない場合が多い。レイアウトでは、図形の特定な部分をレイアウトすることより、軟かいレイアウト制約のように図形の全体を把握しやすく、また、見やすくレイアウトすることが大事である。

2.1.4 軟かいレイアウト制約の種類

グラフ描画アルゴリズムでは、木やグラフの描画を扱う。無向グラフを描画するのに最も基本となるアルゴリズムとしてスプリングモデル [20] がある。また、このスプリングモデルをもとにエッジの方向を考えることにより、有向グラフの描画を行うマグネティックスプリングモデル [21] がある。さらに、木を描画するのに現在最も優れていると考えられる Walker の一般木描画アルゴリズム [22] が存在する [1]。

我々は、これらの木やグラフをインタラクティブにレイアウトするための軟かいレイアウト制約として、スプリングモデル制約、マグネティックスプリングモデル制約、木構造制約を考える。

軟らかいレイアウト制約の名前として、スプリングモデル制約は `spring`、マグネティックスプリングモデル制約は `magnetic`、木構造制約は `treeStructure` を用いる。

2.1.5 軟かいレイアウト制約の扱い方

軟かいレイアウト制約を CMG に基づいて考えると、**CMG の制約として扱う方法と CMG の生成規則として扱う方法が考えられる。**

軟かいレイアウト制約を **CMG の制約として扱う方法**では、生成規則に以下のように軟かいレイアウト制約を指定する。この生成規則には、CMG の生成規則に $softConstraintName$ が追加されている。

$$
\begin{aligned}
T(\vec{x}) \ ::= \ & T_1(\vec{x_1}), \cdots, T_n(\vec{x_n}) \ where \\
& exists \ T'_1(\vec{x'_1}), \cdots, T'_m(\vec{x'_m}) \\
& where \ C(\vec{x_1}, \cdots, \vec{x_n}, \vec{x'_1}, \cdots, \vec{x'_m}) \ and \\
& softConstraintName \ and \\
& \vec{x} \ = \ F(\vec{x_1}, \cdots, \vec{x_n}, \vec{x'_1}, \cdots, \vec{x'_m})
\end{aligned}
$$

この生成規則の意味は、`normal` の構成要素や `exist` の構成要素のマルチセットの属性が制約 C を満たす場合、`normal` の構成要素が非終端シンボル $T(\vec{x})$ に書き

換えられることを表す。さらに、構成要素に指定された軟らかいレイアウト制約の名前（*softConstraintName*）に従ってレイアウトが与えられることを表している。

しかしながら、よく考えてみると軟かいレイアウト制約を CMG の制約として扱う方法では、図形の全体のレイアウトはできないことが分かる。

その理由は次の通りである。グラフ構造のような図形の文法は、例としてあげられたリスト構造のように「再帰的に定義する生成規則」により定義される。図形にこの生成規則が再帰的に適用されると、まず基本的な図形（すなわち円、四角、直線、テキストなど）を組み合わせて非終端シンボルを作る。それから、その非終端シンボルと他のシンボルを組み合わせて非終端シンボルを作ることを繰り返す。我々は、このような非終端シンボルを「再帰的に生成される非終端シンボル」と呼ぶ。また、我々は「再帰的に生成される非終端シンボル」を繰り返し作ることを低い階層を持つ非終端シンボルから高い階層を持つ非終端シンボルを生成して行くことであると考える。例えば、リスト構造を例にすると、生成規則 1 により、四角とテキストを組み合わせて非終端シンボル `list` を生成する。生成規則 2 により、`list`、四角、テキスト、直線を組み合わせて `list` を生成する。生成規則 2 が繰り返し適用されることにより、図形の全体を `list` として認識する。軟かいレイアウト制約を CMG の制約として扱うと、「再帰的に生成される非終端シンボル」が生成されるたびに軟かいレイアウト制約をその非終端シンボルの構成要素に与える。そうすると、低い階層を持つ非終端シンボルから軟かいレイアウト制約が与えられて行くので、図形の一部のレイアウトは可能だが、図形の全体をバランスよくレイアウトすることはできない。

そこで次に、軟かいレイアウト制約を CMG の生成規則として扱う方法について考える。軟かいレイアウト制約を CMG の生成規則（軟らかいレイアウトの生成規則と呼ぶ）として扱うと、生成規則は図形の全体に適用されるので、図形の全体をバランス良くレイアウトすることが可能であると思われる。

軟かいレイアウト制約とは、図形の全体をグラフ描画アルゴリズムに従ってレイアウトする制約であるので、図形の全体を `node` と `edge` で構成するグラフ構造に対応させて制約を与える。

また、`node` や `edge` の構成要素を扱うためには、図形の中に `node` の構成要素や

edgeの構成要素にしたい非終端シンボルを生成する生成規則が必要になる。その他に、図形のパーシングを行うため、nodeの構成要素やedgeの構成要素を用いて、図形を再帰的に定義する生成規則が必要になる。

まず、リスト構造[2]を例にしてnodeやedgeの構成要素を定義するための生成規則を定義する。

生成規則1 四角の中心にラベルとしてテキストが書いてあるものをノードとする。

```
1: lNode(point mid, string text) ::=
2:              R:rectangle, T:text
3:   where (
4:      R.mid == T.mid
5:   ) {
6:   mid  = R.mid;
7:   text = T.text;
8: }
```

生成規則2 ノード間を結ぶ直線をエッジとする。

```
1: lEdge(point start, point end) ::= L:line
2:   where ( exist N1:lNode, N2:lNode
3:      where (
4:         L.start == N1.mid  &&
5:         L.end   == N2.mid
6:   ) {
7:   start = L.start;
8:   end   = L.end;
9: }
```

次は、lNodeやlEdgeを用いてリスト構造を「再帰的に定義する生成規則」を定義する。

[2]リスト構造は木構造や他のグラフ構造より単純な構造を持っているので、軟かいレイアウト制約として扱う必要がないが、説明を分かりやすくするために論文の中でリスト構造の例題を扱う。

生成規則３ リスト構造の生成規則１と同じであるが、構成要素がノードである。

```
1:list(point mid, string text) ::= N:lNode
2:    where (
3:       N.text == ''s''
4:    ) {
5:    mid  = N.mid;
6:    text = N.text;
7: }
```

生成規則４ リスト構造の生成規則２と同じであるが、構成要素がリスト、エッジ、ノードである。

```
1:list(point mid, string text) ::=
2:            L:list, E:lEdge, N:lNode
3:    where (
4:       E.start == L.mid  &&
5:       E.end   == N.mid
6:    ) {
7:    mid  = N.mid;
8:    text = N.text;
9: }
```

我々は、軟らかいレイアウトの生成規則を以下のように定義する。

$$
\begin{aligned}
S(\vec{x}) \;::=\; & \textit{nodes } S_1, \cdots, S_n \\
& \textit{edges } S'_1, \cdots, S'_m \\
& \textit{where } SC \textit{ and } GS(\vec{x_1}, \cdots, \vec{x_n}) \textit{ and} \\
& \vec{x} \;=\; F(\vec{x_1}, \cdots, \vec{x_n})
\end{aligned}
$$

ここで、S は非終端シンボルの名前、S_1, …, S_n は $node$ の構成要素の名前、S'_1, …, S'_m は $edge$ の構成要素の名前、SC は軟らかいレイアウト制約の名前、

GS は再帰的に生成される非終端シンボルの名前を示している。ここで、軟らかいレイアウト制約の名前として、spring、magnetic、treeStructure が用いられる。

軟らかいレイアウトの生成規則は、通常の生成規則と異なり、図形の全体に存在する node の構成要素 S_1, …, S_n のマルチセットと edge の構成要素 S'_1, …, S'_m のマルチセットを GS の構成要素から求めて非終端シンボル S として認識し、指定された軟らかいレイアウト制約の名前 SC に従ってレイアウトを行うことを意味する。

各非終端シンボルは、自分の構成要素の情報を持っているので、軟かいレイアウトの生成規則が適用される時に、GS の構成要素が GS を持たなくなるまで繰り返し検査し、node や edge の構成要素のマルチセットを動的に求めて、SC に従ってレイアウトを行うことができる。このため、図形エディタに node や edge の構成要素になっている非終端シンボルが追加・削除されても、きちんとレイアウトを行うことができる。

また、複数の図形があるときに、それぞれの図形に軟らかいレイアウトの生成規則を与え、きちんとレイアウトを行うことができる。複数の図形がパーシングされると、複数の図形と同じ数の GS が生成される。それぞれの GS の構成要素から、それぞれの図形に対する node の構成要素や edge の構成要素のマルチセットを求め、軟らかいレイアウトの生成規則を適用すれば可能である。

リスト構造の軟らかいレイアウトの生成規則は、以下のように定義される。

```
1: layoutList(point mid) ::= nodes:lNode,
2:                           edges:lEdge
3:   where (
4:      listStructure  &&
5:      list
6:   ) {
7:   mid = list.mid;
8: }
```

ここで、4行目の listStructure は軟かいレイアウト制約の種類の名前である

リスト構造制約（SC）、5行目のlistは「再帰的に生成される非終端シンボル」の名前（GS）を示している。

2.2 硬いレイアウト制約

硬いレイアウト制約とは、図形の一部をローカルにレイアウトする制約である。特に、図形の座標や図形間の距離などの制約を具体的に与えたい場合に用いる。

2.2.1 硬いレイアウト制約の種類

図形の座標を一致させて図形を描画する制約は具体的に次のように記述する。

```
layout_eq 変数1 変数2
```

ここで、変数1と変数2は終端シンボルもしくは非終端シンボルの属性を示している。変数1と変数2の型としてはinteger、pointを用いることができる。変数2の値として変数1の値を代入して、変数2を属性として持つ図形を変った位置に描画する。例えば、図形の構成要素としてノードN1とN2があるとする。layout_eq N1.mid_y N2.mid_yは、N1とN2の構成要素が解釈された後、N2の構成要素の属性mid_y（中心のy座標）の値をN1の構成要素の属性mid_yの値と等しくしてN2の構成要素を描画する。

図形間の距離を具体的に与えて図形を描画する制約は、

```
layout_dist 変数1 変数2 distance
```

のように記述することができる。ここで、変数1と変数2は終端シンボルもしくは非終端シンボルの属性、distanceは図形間の距離を表す定数である。変数1と変数2の型としてはinteger、pointを用いることができる。変数1の値とdistanceほど離れている座標を求めて、変数2の値に代入したあと変数2を属性として持つ図形を変った位置に描画する。例えば、図形の構成要素としてノードN1とN2があるとする。layout_dist N1.mid_x N2.mid_x 100は、N1とN2の構成要素が解釈された後、N1の構成要素の属性mid_x（中心のx座標）とN2の構成要素の属性mid_xとの距離を100ドットにしてN2の構成要素を描画する。

2.2.2 硬いレイアウト制約の扱い方

硬いレイアウト制約は、CMG に基づいた制約の拡張として考えることができる。その理由は、硬いレイアウト制約は通常の制約のように生成規則が適用される特定の図形要素間に与えられる制約だからである。

すなわち、生成規則の定義、

$$
\begin{aligned}
T(\vec{x}) \;::=\; & T_1(\vec{x_1}),\cdots,T_n(\vec{x_n})\ where \\
& exists\ T'_1(\vec{x'_1}),\cdots,T'_m(\vec{x'_m}) \\
& where\ C(\vec{x_1},\cdots,\vec{x_n},\vec{x'_1},\cdots,\vec{x'_m})\ and \\
& HC(\vec{x_1},\cdots,\vec{x_n})\ and \\
& \vec{x} \;=\; F(\vec{x_1},\cdots,\vec{x_n},\vec{x'_1},\cdots,\vec{x'_m})
\end{aligned}
$$

の中で硬いレイアウト制約（HC）は、通常の制約（C）のところに記述する。これは、硬いレイアウト制約が CMG に基づいた制約と同様の性質を持つものだからであり、通常の制約の延長として扱うことができる。

例えば、図 2.2（a）のように円（Node）とそれらを結ぶ直線（Edge）から構成されるグラフ構造があるとする。また、図 2.2の各ノードのとなりに書いてある数字は図形の「構成要素の種類. 構成要素の順番」だとする。図 2.2（b）のようにエッジでつながっているノード間の距離を 200 して各ノードを四角にレイアウトしたい場合、次のように生成規則を定義する。

```
1:graph() ::= N1:Node, N2:Node, N3:Node, N4:Node,
2:            E1:Edge, E2:Edge, E3:Edge
3:   where (
4:      E1.start == N1.mid  &&
5:      E1.end   == N2.mid  &&
6:      E2.start == N2.mid  &&
7:      E2.end   == N3.mid  &&
8:      E3.start == N3.mid  &&
9:      E3.end   == N4.mid  &&
```

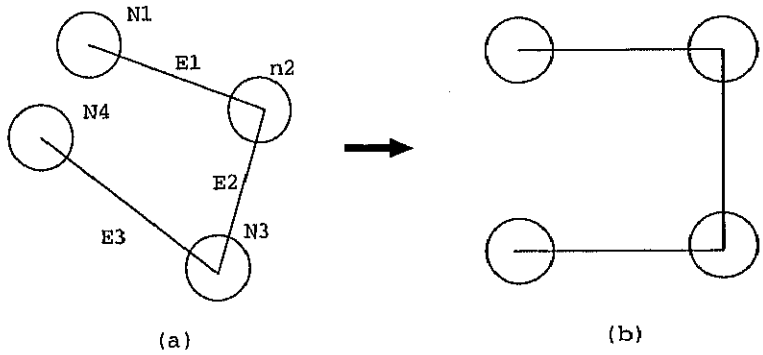

図 2.2: 硬いレイアウト制約の例

```
10:       layout_eq    N1.mid_y  N2.mid_y &&
11:       layout_dist  N1.mid_x  N2.mid_x  200 &&
12:       layout_eq    N2.mid_x  N3.mid_x &&
13:       layout_dist  N2.mid_y  N3.mid_y  200 &&
14:       layout_eq    N3.mid_y  N4.mid_y &&
15:       layout_dist  N3.mid_x  N4.mid_x  -200
16:    )
```

2.2.3 通常の制約と硬いレイアウト制約の違い

図形の解釈が成功するためには、文法に書かれた通常の制約を満す必要がある。また、通常の制約は図形を解釈することにより、図形間の関係をそのまま維持する制約である。それに対して、レイアウト制約は図形が解釈された後、もともとの図形要素間になかった位置関係を作り出す制約である。

通常の制約の中で eq 制約がある。eq 制約は、図形を解釈するときに予め二つの図形の属性の値が等しい (equal) ことを意味する。この制約が一度成り立つと、一つの属性の値が変っても常にこの制約が成り立つように他の属性の値が書き換えられる。それに対して、layout_eq は、図形を解釈するときに異なっていた二つの図形の属性の値を解釈が成功した後等しくする。

硬いレイアウト制約を導入した理由は、生成規則により生成される非終端シンボルの一部をローカルにレイアウトするためである。また、硬いレイアウト制約と軟

かいレイアウト制約を混ぜて用いることにより、空間パーサの応用が広がると考えられる。

2.2.4 空間パーサとレイアウト制約との関係

空間パーサとレイアウト制約との関係を明確にするため、「空間パーサとレイアウト機能を別々にしたシステム」について考察し、「空間パーサにレイアウト制約を追加したシステム（我々が提案したシステム）」と比較を行うことにより、「我々が提案したシステム」の特色を明らかにすることを試みる。

「空間パーサとレイアウト機能を別々にしたシステム」には、様々な作り方が可能であると思われるが、最も自然な作り方は、まず空間パーサにより、図形からノードやエッジに相当する非終端シンボルを認識し、次に、ユーザが定義したレイアウト規則により、認識したノードやエッジをレイアウトするシステムであろう。

この場合、空間パーサは、図形からノードやエッジに相当する非終端シンボルを認識した時点で、情報をレイアウトシステムに渡してしまうので図形のレイアウトは可能であるが、正確には与えられた図形がグラフ構造であるかどうかのパーシングは行っていない。通常、グラフ構造のパーシングを行うためには再帰的な生成規則により、文法を定義する必要がある。しかしながら、グラフ構造を再帰的に最後までパーシングすると一つの非終端シンボルしか存在しなくなる。この場合、パーシングしながらレイアウトすることが困難である。

一方、「我々が提案したシステム」では、軟らかいレイアウトの生成規則を用いて図形のパーシングとレイアウトを同時に行うことができる。

また、再帰的に定義された生成規則が図形に適用された場合、例としてあげたリスト構造のように、低い階層を持つ非終端シンボルから高い階層を持つ非終端シンボルを生成して行く。「我々が提案したシステム」では、空間パーサにレイアウト制約を追加することにより、まず軟らかいレイアウト制約や硬いレイアウト制約が与えられた図形を非終端シンボルとして認識することができる。これを低い階層を持つ非終端シンボルのレイアウトだと考える。また、その非終端シンボルにさらに軟らかいレイアウト制約や硬いレイアウト制約を与えることができる。すなわち、ノードやエッジとして扱おうとする図形に硬いレイアウト制約を与え、ローカルなレイアウトを行い、非終端シンボルとして認識する。次に、ノードやエッジの全体

に軟らかいレイアウトの生成規則を適用し、グラフ描画アルゴリズムに従ってバランスよくレイアウトすることが可能である。従って、「我々が提案したシステム」では、階層的にレイアウトすることが容易である。

しかしながら、「空間パーサとレイアウト機能を別々にしたシステム」では、階層的にレイアウトすることが困難である。あえて階層的にレイアウトするためには、まず低い階層を持つ非終端シンボルのパーシングが終ってからレイアウトを行い、次に、より高い階層を持つ非終端シンボルのパーシングが終ってからレイアウトを行うというように図形のパーシングとレイアウトを繰り返し行う必要があるが、これらを実装することは「空間パーサとレイアウト機能を別々にしたシステム」では難しい。