第 4 章

「Rainbow」の適用例

4.1 スプリングモデル制約

4.1.1 スプリングモデル

スプリングモデル [20] は無向グラフを描画するためのアルゴリズムの中で最も基本となるアルゴリズムである。本アルゴリズムは、エッジの長さを一定にすることやノード間の対称性を顕示することを目指して無向グラフを描画する場合に多く応用される。

スプリングモデルとは、ノードを鉄のリングに、エッジを力学系を形成するバネに置き換え、リングの安定状態を見つけることで適切なレイアウトを求めるものである（図 4.1）。

このモデルは 2 種類のバネが使われる。すなわち隣接するノード間をつなぐバネと隣接しないノード間に斥力だけを与えるバネである。隣接するノード間に働く力 f_s は

$$f_s = c_1 \log(d/c_2)$$

により与えられる。ここで d はノード間の距離、c_1 と c_2 は定数とする。$d > c_2$ のとき f_s は引力、$d < c_2$ のとき斥力、$d = c_2$ のときノード間に力は働かない。また、隣接しないノード間に働く力 f_r は

$$f_r = c_3/d^2$$

により与えられる。ここで c_3 は定数とする。このような力 f_s と f_r をできるだけ緩

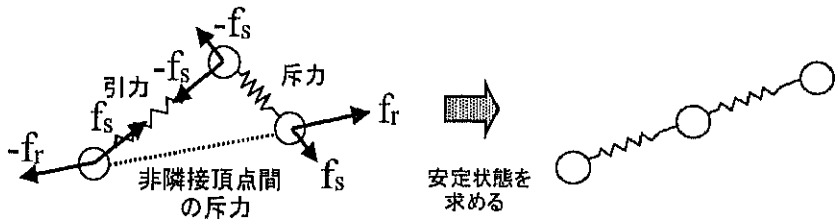

図 4.1: スプリングモデルによるグラフのレイアウト

和するように各ノードを (c_4 × そのノードに働く力) ずつ移動させることにより、すべての隣接するノード間の距離を c_2 の近傍に収束させる。c_4 は定数とする。

4.1.2 スプリングモデル制約の例

スプリングモデル制約の例として、データベース分野で実世界のデータ構造を記述するのに用いられる E-R ダイアグラム [24] を考える。E-R ダイアグラムの構成要素を次のように定義する。1) 四角の中に実体名が書いてある図形を実体ノード (entityNode) とする。2) 円の中に属性名が書いてある図形を属性ノード (attributeNode) とする。3) 実体ノード間の関連を表す直線を実体エッジ (entityEdge) とする。その直線の中心には関連型が書いてある。4) 実体ノードと属性ノード間の関係を表す直線を属性エッジ (attributeEdge) とする。これらの構成要素を解釈する生成規則を「Rainbow」を用いて定義する。例えば、実体ノードを解釈する生成規則の場合、ユーザは「Rainbow」の図形エディタに四角を描いてその中にテキストを書いて CMG 入力ウィンドウを開く。CMG 入力ウィンドウの非終端シンボルの名前の欄には、entityNode を書く。属性の欄には、実体ノードの属性 mid (中心) を四角の中心にするように書く。制約の欄には、四角の中心とテキストの中心を等しくする制約を選び、非終端シンボル entityNode の定義を完了する。さらに、非終端シンボル ERGraph を生成するように、E-R ダイアグラムを再帰的に定義する生成規則を定義する。

次に、解釈したい図 4.2の図形を選んで軟かいレイアウト CMG 入力ウィンドウを開くと、システムは、node と edge の構成要素の名前の欄に、四角、円、直線などの終端シンボルを除いて非終端シンボルの名前を自動的に書き出す。node と edge の構成要素の名前の欄には

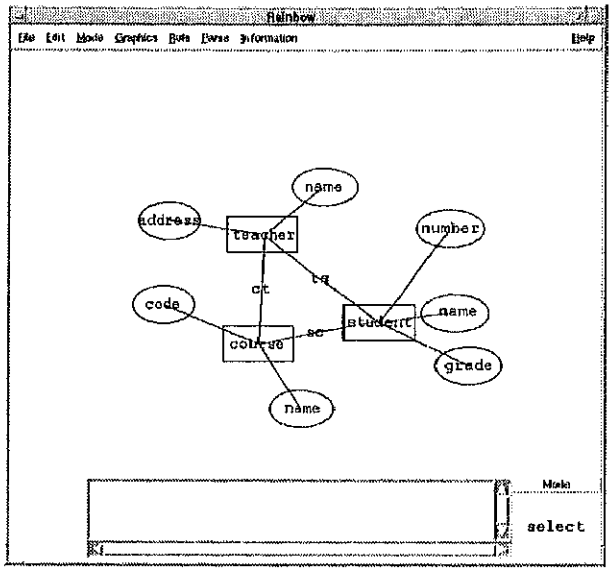

図 4.2: レイアウト前の E-R ダイアグラム

entityNode

attributeNode

entityEdge

attributeEdge

のように記述される。そうすると、ユーザは node の構成要素の名前の欄には

entityNode

attributeNode

edge の構成要素の名前の欄には

entityEdge

attributeEdge

を選ぶ。

次に、ユーザは「Layout Constant Input」メニューを選択してスプリングモデルの定数を決める。非終端シンボルの名前の欄には ERModel、軟らかいレイアウト制約の名前の欄には spring、再帰的に生成される非終端シンボルの名前の欄には

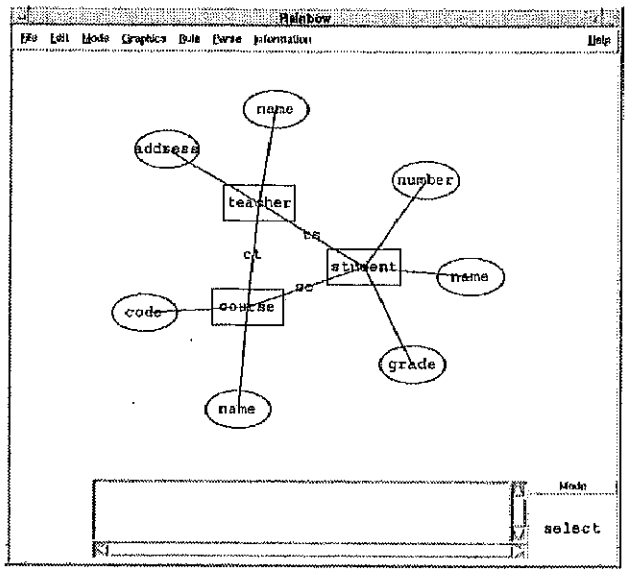

図 4.3: レイアウト後の E-R ダイアグラム

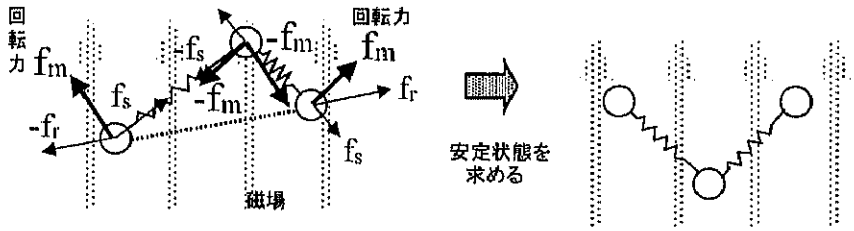

図 4.4: マグネティックスプリングモデルによるグラフのレイアウト

ERGraph を書いて軟かいレイアウトの生成規則を定義する。これらの生成規則を用いて、図 4.2の図形を解釈すると図 4.3のようにレイアウトされる。

4.2 マグネティックスプリングモデル制約

4.2.1 マグネティックスプリングモデル

マグネティックスプリングモデル [21] は、スプリングモデルに磁場の概念を入れたものである（図 4.4）。エッジを磁針と見なし、グラフの置かれた磁場から回転力を受ける。マグネティックスプリングモデルの磁場には、有向エッジに働くものと無向エッジに働くものの二種類がある。有向エッジの場合はエッジの終点が磁場

の北を向くように回転力を受ける。無向エッジの場合は磁場の向きは関係なくノードは南北の向きに近い方を向くように回転力が働く。回転力は次のように定義される。

$$f_m = c_5 b d^{\alpha} \mid t \mid^{\beta}$$

ここで、b は基準点（エッジの中心）における磁場の強さ、d は現在のエッジの長さである。t はエッジの基準点における磁場の北からの終点のずれの角度である。すなわち、有向エッジの場合は $0 < t \leq \pi$、無向エッジの場合は $0 < t \leq \pi/2$ となる。また、α は辺の長さの回転力への影響を制御する定数、β は t の回転力への影響を制御する定数である。また、隣接するノード間に働く力と隣接しないノード間に働く力は、f_s と f_r を用いる。

マグネティックスプリングモデルは、複数種類のエッジを持つ有向グラフのレイアウトにおいてそれぞれのエッジを一定の方向に向かせることや、有向エッジと無向エッジが混在したグラフのレイアウトにおいて有向エッジだけを一定の方向に向かせ、また、無向エッジも特定の方向にそろえることの応用に用いられる。

4.2.2 マグネティックスプリングモデル制約の例

マグネティックスプリングモデル制約の例として、オブジェクト指向に基づくソフトウェア設計に用いられるオブジェクト図 [11] [25] の自動レイアウトについて考える。オブジェクト図の構成要素として、クラス（class）、関連（association）、汎化（generalization）、および、集約（aggregation）が存在する。ここで、クラスをノード、3 種類の関係をエッジとして見なすことができ、3 種類のエッジをそれぞれ違う方向に向けるようにすれば、オブジェクト図の自動レイアウトが可能である [26]。

それぞれのエッジへどの種類の磁場を与えるかは、オブジェクト図におけるそれぞれのエッジの特性や意味的な要素から以下のようにする。関連は、オブジェクト同士の対等な参照あるいは利用関係を表すもので、方向のない関係である。従って関連エッジは平行磁場とし、横向きに磁場を与える。汎化は、クラス間の概念的な包含関係、すなわちスーパークラスとサブクラス間の継承関係を表すものなので、上から下の有向関係である。従って汎化エッジは下向の磁場を与える。集約は、一

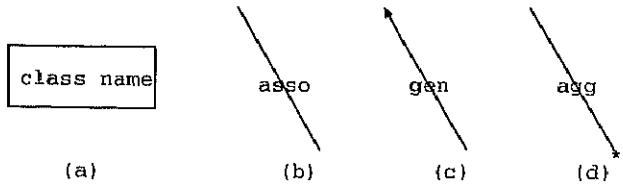

図 4.5: オブジェクト図の構成要素

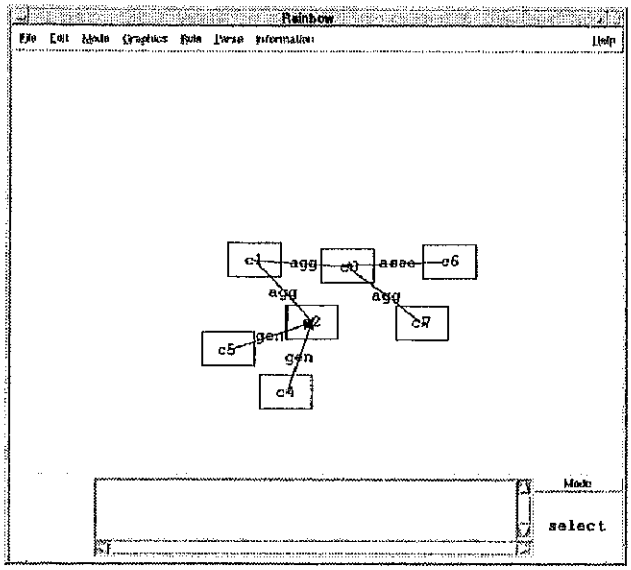

図 4.6: レイアウト前のオブジェクト図

方のオブジェクトが他方の部分となるような構造的な包含関係を表すものである。集約エッジは汎化エッジと区別するために斜め右下 45 度の磁場を与える。

我々は、オブジェクト図の構成要素を区別するため図 4.5のように定義する。(a) は四角の中にクラス名が書いてある図形をノードとして定義する。(b) は直線の中心に「asso」が書いてある図形を関連エッジとして定義する。(c) は直線の始点に矢印があって直線の中心に「gen」が書いてある図形を汎化エッジとして定義する。(d) は直線の終点に「*」があって直線の中心に「agg」が書いてある図形を集約エッジとして定義する。まず、これらのオブジェクト図の構成要素を解釈する生成規則を定義する必要がある。定義方法は、図 4.5のように図形を図形エディタに描いて開いた CMG 入力ウィンドウで行う。例えば、集約エッジの場合は、非

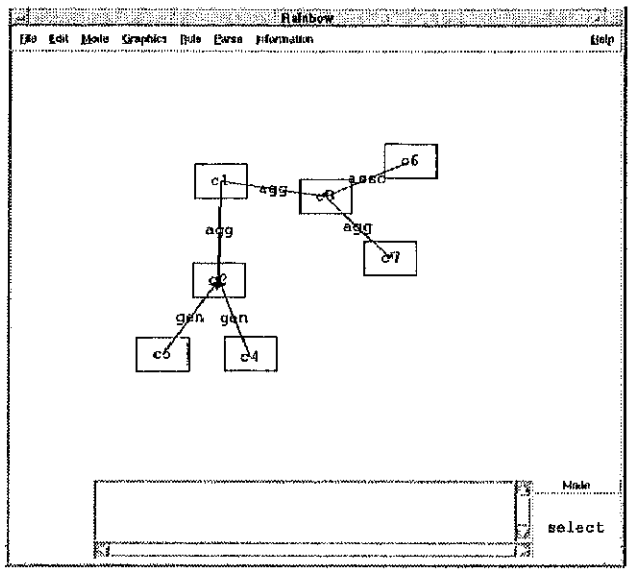

図 4.7: レイアウト後のオブジェクト図

終端シンボルの名前として CMG 入力ウィンドウの名前の欄に aggregation と書き、属性 start、 end は直線の始点、終点にするように書く。制約の欄には、直線の中心とテキストの中心、直線の終点と「*」の中心を等しくする制約を選び、非終端シンボル aggregation の定義を行う。さらに、非終端シンボル ObjectGraph を生成するように、オブジェクト図を再帰的に定義する生成規則を定義する。

また、解釈したい図 4.6の図形を選んで軟かいレイアウト CMG 入力ウィンドウを開く。ユーザは自動的に書き出された非終端シンボルの名前から node と edge の構成要素を選び、マグネティックスプリングモデルの定数を決める。名前として ObjectModel、軟らかいレイアウト制約の名前の欄には magnetic、再帰的に生成される非終端シンボルの名前の欄には ObjectGraph を書く。次に、それぞれの edge の構成要素にエッジの種類と磁場を与える。これらの与え方は、 edge の構成要素の名前の欄に選ばれた構成要素の名前の最初にエッジの種類と磁場の角度をユーザが付け加える。この例では次のように記述する。

```
{undirect(0) association}
{direct(90) generalization}
{direct(45) aggregation}
```

これらの生成規則を用いて図4.6の図形を解釈すると図4.7のような結果が得られる。

4.3 木構造制約

4.3.1 木構造

我々は、木の描画法としてWalkerの一般木描画アルゴリズム[22]を用いる。その理由は、木に関する描画法の研究として行われてきたいろいろなアルゴリズムの中で、現在最も優れていると考えられるからである[1]。

Walkerの一般木描画アルゴリズムでは、木のルートとなる親ノードは描画の最上の位置に配置され、親ノードの各子ノードは下方の階層に対応した平行線上に、あらかじめ与えられた順序に従い左から右に並べられる。ノードのy座標は連続する階層の間を一定の間隔を開けるから容易に求められる。従って、ノードのx座標を決定することが重要である。木構造のレイアウト規則モジュールでは、Walkerの一般木描画アルゴリズムを実現するための必要な情報を連想配列Wに格納している。連想配列Wのそれぞれの要素は図4.8のようになっている。本節では変数名など実際の文字列はタイプライタ体で，値が変るものはイタリックで表記する．ここで、*t-id*はトークンにユニークにつけられたidである。

ノードの最終的なx座標を求めるために、まず、後方順探索により、各ノードの仮x座標値と修正座標値を求める。これらの座標値の求め方について述べる。1) ノードBが葉であり、左側に兄弟ノードを持たない場合、prelim(B)とmodifier(B)は0である。2) ノードBが葉であり、左側に兄弟ノードAを持つ場合、prelim(B)は、

prelim(A) + Aの大きさの半分 + Bの大きさの半分 + 兄弟分離

であり、modifier(B)は0である。3) ノードBが葉ではなく、左側に兄弟ノードを持たない場合、prelim(B)は、

(prelim(Bの最も左側の子ノード) + prelim(Bの最も右側の子ノード)) / 2

であり、modifier(B)は0である。4) ノードBが葉ではなく、左側に兄弟ノードを持つ場合、prelim(B)は2)と同じ計算式で、modifier(B)は、

名前	値
W(root)	ルートノード
W(node)	node の構成要素のリスト
W(edge)	edge の構成要素のリスト
W(height)	木の高さ
W(yDistance)	階層間の一定の間隔
W(siblingDistance)	兄弟ノード間の最小距離（兄弟分離）
W(subtreeDistance)	部分木間の最小距離（部分木分離）
W(*t-id*.parent)	親ノード
W(*t-id*.child)	子ノードのリスト
W(*t-id*.leftChild)	最も左の子ノード
W(*t-id*.rightChild)	最も右の子ノード
W(*t-id*.leftSibling)	左側の兄弟ノード
W(*t-id*.rightSibling)	右側の兄弟ノード
W(*t-id*.leftNeihbor)	左側の部分木で同じ深さの最も右のノード
W(*t-id*.depth)	ノードの深さ
W(*t-id*.prelim)	仮 x 座標値
W(*t-id*.modifier)	修正座標値
W(*t-id*.final)	最終 x 座標値

図 4.8: 木構造のレイアウト規則モジュールの内部表現

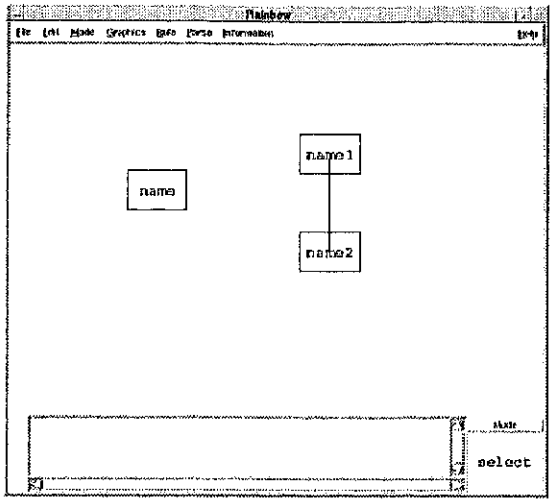

図 4.9: 「Rainbow」の図形エディタ

```
prelim(B) - (prelim(Bの最も左側の子ノード) +
prelim(Bの最も右側の子ノード)) / 2
```

である。仮 x 座標値と修正座標値を求めながら隣接する部分木間の分離を行う。部分木間の分離は、隣接するノードの根を兄弟分離の値だけ引き離し、一つ下がった階層で部分木分離が達成されるまで引き離す。この過程を短い方の部分木の最下層に着くまで順次繰り返す。

次に、先行順探索により、各ノードの仮 x 座標値にそのノードの祖先の修正座標値をすべて足し会わせることにより、各ノードの最終 x 座標値を計算する。最終 y 座標値は、各ノードの深さに階層間の一定の間隔をかけて求める。最終 x と y 座標値に従って図形を再描画する。

4.3.2 木構造制約の例

会社などの仕組みを表すのに用いられる組織図を解釈しながらレイアウトすることを考える。組織図は、部署を表すノードと部署間の包含関係を表すエッジで木構造に構成される。

「Rainbow」を用いて、組織図を解釈しながらインタラクティブに木構造にレイアウトするためには、組織図の構成要素を解釈する通常の生成規則の他に木構造のレイアウトの生成規則を定義する必要がある。「Rainbow」では、図形を用いて生

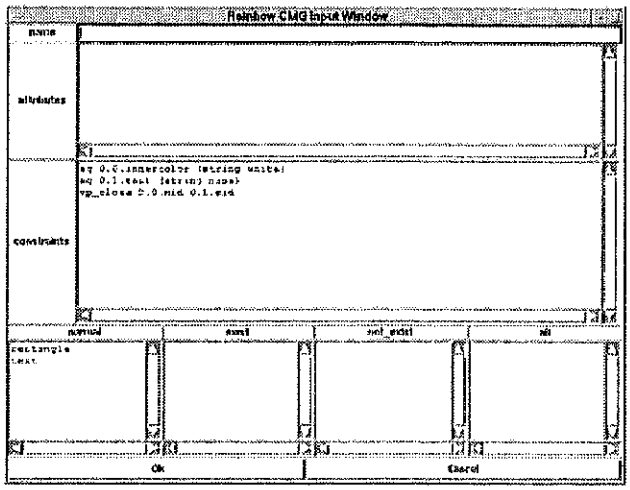

図 4.10: ノードの CMG 入力ウィンドウ (1)

成規則の定義を行っている。まず、ユーザは非終端シンボルノード（tNode）を生成する生成規則を定義するため、図形エディタに四角（rectangle）を描き、その中心に適当なテキスト（text）を書く（図 4.9の左の図形）。これらをまとめて選択し、図形エディタの「Rule」メニューから「Make New Production Rule」をクリックすると「CMG 入力ウィンドウ（図 4.10）」が開く。そのとき「Rainbow」によって、図形から構成要素とそれらの属性間に成り立っている制約は CMG 入力ウィンドウに書き出される。normal の構成要素の欄には、rectangle と text が書かれる。また、制約の欄（constraints）には以下のような制約が書かれる。

```
eq 0.0.innercolor {string white}
eq 0.1.text {string name}
vp_close 0.0.mid 0.1.mid
```

ここで、1行目は rectangle（0.0）の塗り潰す色 innercolor が白であるという制約で、2行目は text（0.1）の文字列か name であるという制約である。3行目は、rectangle の中心（mid）と text の中心がほぼ一致していることを表す制約である。

我々は、この CMG 入力ウィンドウの非終端シンボルの名前の欄に tNode と書き、非終端シンボルの属性の欄には tNode の属性 mid を rectangle の中心にするよう

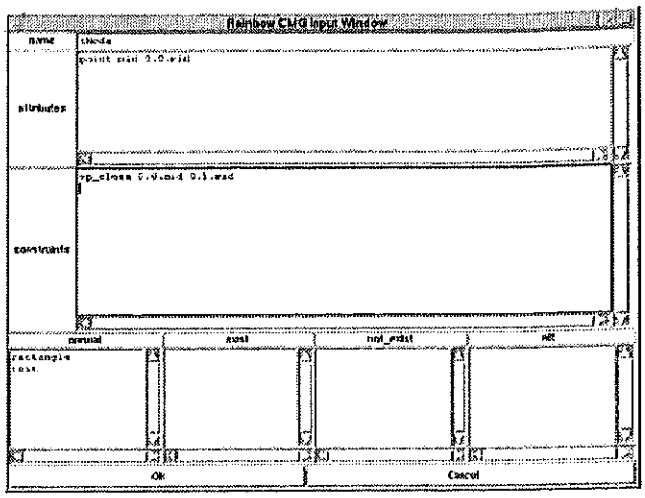

図 4.11: ノードの CMG 入力ウィンドウ (2)

に書く。また、制約の欄には、rectangle の中心と text の中心を等しくする 3 行目の制約だけを選ぶ（図 4.11）。「OK」ボタンをクリックすることにより、tNode の定義を完了する。同様にして非終端シンボルエッジ（tEdge）の生成規則を定義する。図 4.9の右の図形のように二つのノードとそのノード間を結ぶ直線（line）を描いて選択して開かれた CMG 入力ウィンドウに書かれた情報を修正した CMG 入力ウィンドウが図 4.12である。

さらに，非終端シンボル organizationGraph を生成するするように，組織図を再帰的に定義する生成規則を記述する。図 4.9の左の図形を用いて tNode を organizationGraph として認識する生成規則，図 4.9の右の図形を用いて二つの organizationGraph とそれらの間を結ぶ tEdge を organizationGraph として認識する生成規則を記述する。

次に、ユーザはレイアウトの生成規則を定義する。ユーザは図 4.13のような図形を図形エディタに描き、一部または全体を選択して図形エディタの「Rule」メニューから「Make New Layout Production Rule」をクリックする。そうすると、「軟かいレイアウト CMG 入力ウィンドウ（図 4.14）」が開く。「Rainbow」は、図 4.13の図形から四角、円、直線などの終端シンボルを除いて非終端シンボルの名前，すなわち tNode，tEdge，organizationGraph，を自動的に node と edge の構成要素の欄に書き出すので，ユーザは，それぞれの構成要素にしたい非終端シンボルの名

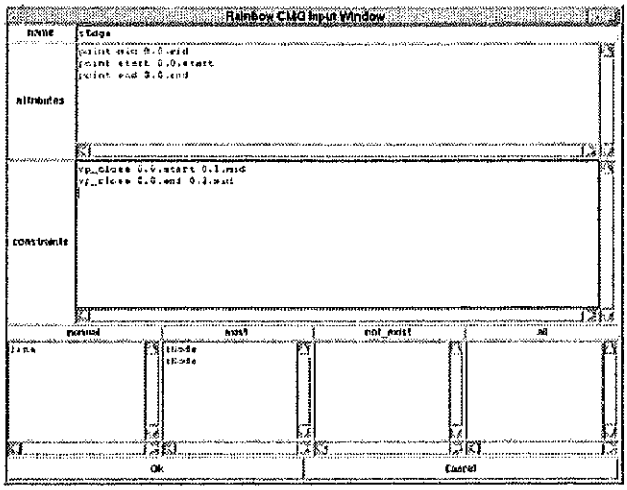

図 4.12: エッジの CMG 入力ウィンドウ

前を選ぶ。次に、ユーザは「Layout Constant Input」メニューを選択して階層間の一定の間隔、兄弟ノード間の最初距離などの定数を決める。非終端シンボルの名前の欄には treeDiagrams、軟らかいレイアウト制約の名前 SC の欄には tree, 再帰的に生成される非終端シンボルの名前 GS の欄には organizationGraph を書き、「OK」ボタンをクリックすることにより、木構造のレイアウトの生成規則の定義を完了する（図 4.15）。

解釈したい図形を実際に実行するには、レイアウトの生成規則を定義するときに用いられた図形またはその一部を選択し、図形エディタの「Parse」メニューから「Parse Selected Items」をクリックする。それにより、定義された生成規則が適用されると tNode, tEdge, organizationGraph が生成される。レイアウトの生成規則が適用されると、再帰的に生成される非終端シンボル GS の構成要素が GS を持っ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求められ、図形を木構造にレイアウトする。図 4.13の図形を解釈すると、図 4.16のようなレイアウト結果が得られる。

4.3.3 木構造制約と硬いレイアウト制約を混ぜた例

木構造制約と硬いレイアウト制約を混ぜた例として、親族の関係を表すのに用いられる家系図を挙げる。家系図とは、人を表すノード（node）、夫婦関係を表す

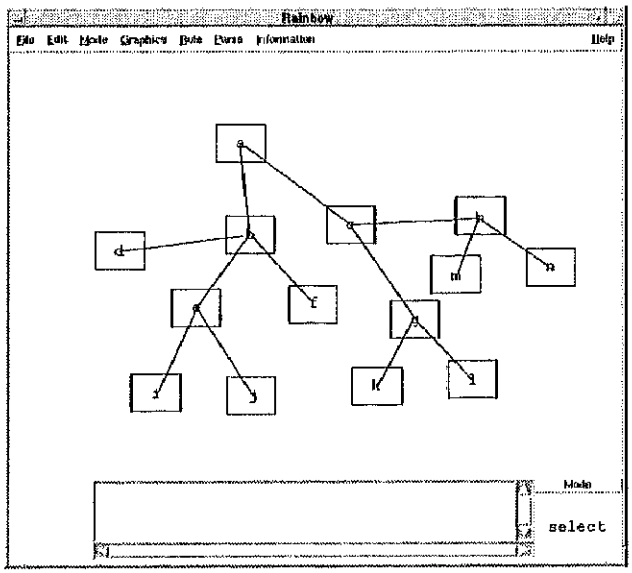

図 4.13: レイアウト前の組織図

親エッジ（pEdge）、親エッジとそのエッジに結ばれている二つのノードを表す親ノード（pNode）、親ノードとノードに親子関係を表す子エッジ（cEdge）で構成されている。これらの構成要素を解釈する生成規則を「Rainbow」を用いて定義する。そのとき、夫婦関係を分りやすく表すために、親ノードの構成要素の二つのノードを同じ平行線上に配置し、ノード間の距離を一定にすることが望まれる。そこで、我々はある図形を親ノードとして認識する生成規則の制約に layout_eq 制約と layout_dist 制約を用いる。さらに、非終端シンボル kinshipGraph を生成するように、家系図を再帰的に定義する生成規則を定義する。

また、子エッジと結ばれている親ノードとノードとの親子関係を分りやすくするため、木構造に表すことが望まれる。そこで、我々はレイアウトの生成規則の軟かいレイアウトの種類として tree を用いる。非終端シンボルの名前の欄には kinshipDiagrams、再帰的に生成される非終端シンボルの名前の欄には kinshipGraph、node の構成要素の名前の欄には node と pNode、 edge の構成要素の名前の欄には cEdge を用いる。定義された生成規則が適用されていくと図 4.17が図 4.18のように木構造の描画が行われる。

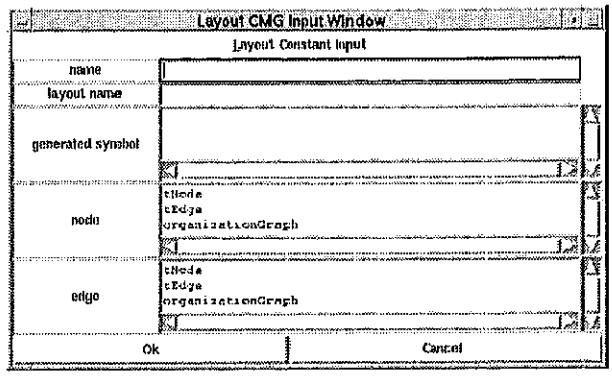

図 4.14: 組織図の軟かいレイアウト CMG 入力ウィンドウ (1)

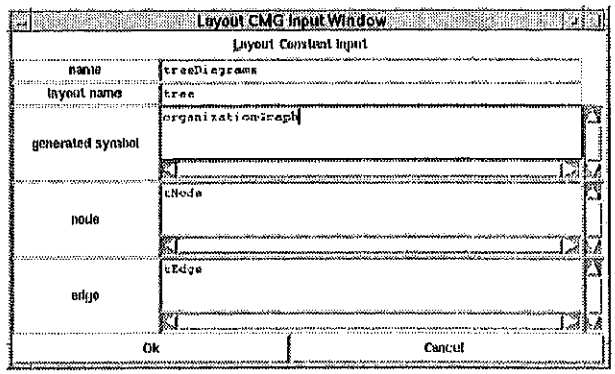

図 4.15: 組織図の軟かいレイアウト CMG 入力ウィンドウ (2)

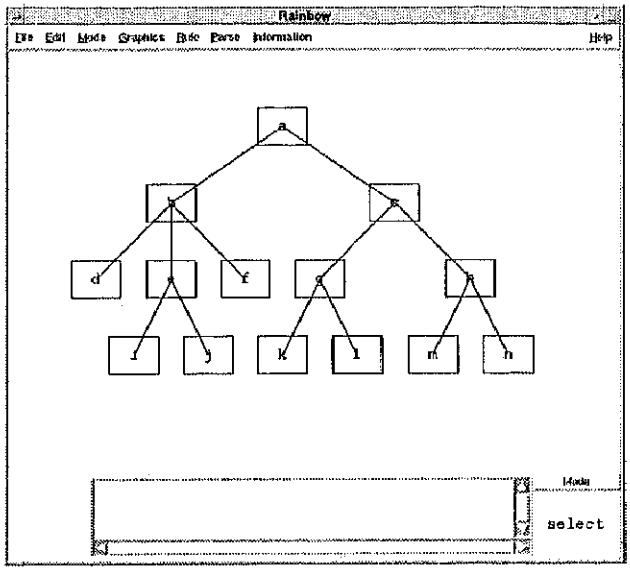

図 4.16: レイアウト後の組織図

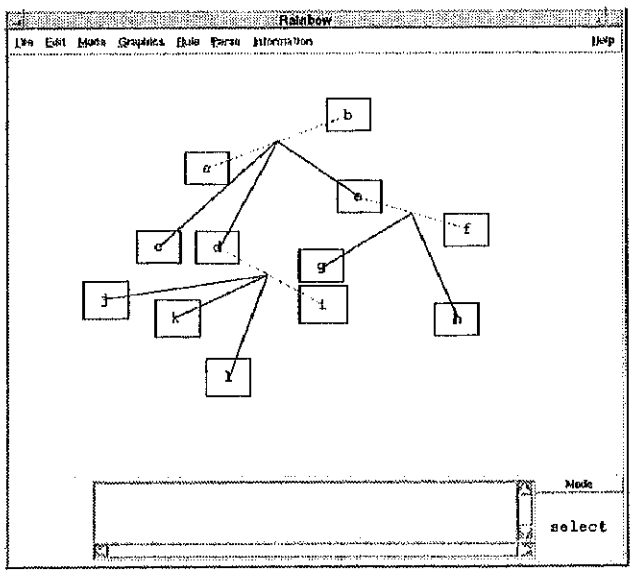

図 4.17: レイアウト前の家系図

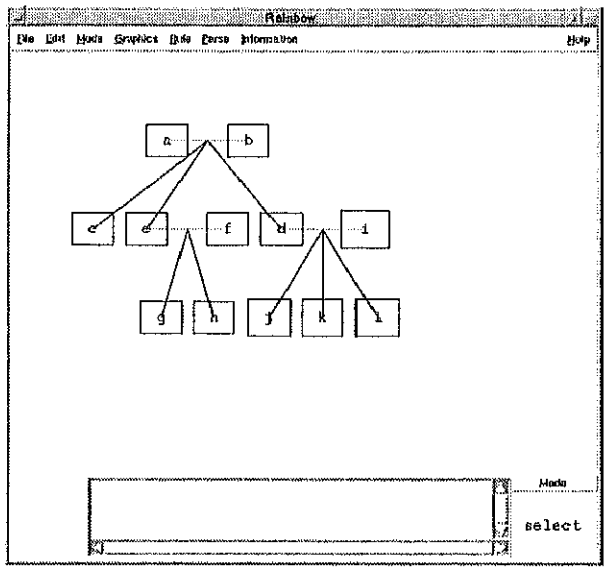

図 4.18: レイアウト後の家系図