On Patterns and Graphs *by Brian Mayoh*

As graph grammars are used in many areas of computer science, it is not surprising that many different kinds of graph grammars have been introduced. As there are many practical and theoretical advantages of *context-freedom*, it is not surprising that most useful graph grammars fit the abstract characterisation of "contextfree" in [Co87]. However the practical and theoretical advantages of contextfreedom are not lost if one moves to the more expressive *pattern* version of a contextfree grammar. In this paper we have 3 aims:

- to explain what the pattern version of a contextfree grammar is and why the advantages of contextfreedom are preserved,
- to give the pattern version of several popular graph and hypergraph grammars,
- by giving striking examples to show that the extra expressive power of patterns is worth having.

#1 Patterns: what and why?

In formal language theory the idea of patterns seems to have appeared first in the contextual grammars of Marcus[Ma69] and Paun[Pa82,Pa94]. Then those interested in language learning started using patterns [An80], and now the formal linguists are reinvigorating the idea [DPS93]. In the string world one can define a pattern as a word W on terminal and nonterminal letters together with a partition of the occurrences of the nonterminals in W. If all occurrences of the same nonterminal are equivalent, then the pattern is *fractal;* if all occurrences of nonterminals are equivalent only to themselves, then the pattern is *discrete.* Thus a discrete fractal pattern has at most one occurrence of each nonterminal. When displaying patterns we display the partitition using the convention:

- λ is the empty pattern
- nonterminals with the same adornment are equivalent,
- unadorned nonterminals are only equivalent to themselves.

Thus a b S c b T a c S' b b T' is a pattern where the last two nonterminals are equivalent to each other and the first two are only equivalent to themselves. If W_1 (W_2 , W_3) are terminal words derivable from S (T, both S and T), then one can substitute in the pattern to get the terminal word:

a b W_1 c b W_2 a c W_3 b b W_3

Def.1. A string pattern multigrammar Γ consists of Σ a terminal alphabet, Δ a nonterminal alphabet, and for each S in Δ a set Π_S of patterns over Σ and Δ. Γ is fractal if all its patterns are fractal; Γ is discrete if all its patterns are discrete. The underlying context free grammars of Γ are given by choosing a nonterminal from Δ, then converting each pattern π in Π_S to the rule S -> p where word p is π without its adornments.

Def.2. A [discrete, fractal] string pattern language $L \subset \Sigma^*$ is generated by a [discrete, fractal] string pattern multigrammar Γ if L= L_S for some S in Δ where L_S is the S-component of the least fixed point of the language function Φ defined by

W in $\Phi_S[....L_\delta...]$ iff
W = $u_0\ v_1\ u_1\ v_2\ u_2.....v_n\ u_n$

for some pattern $u_0\ \delta_1\ u_1\ \delta_2\ u_2.....\delta_n\ u_n$ in Π_S and terminal words $u_0,u_1,u_2.....u_n,v_1,v_2....v_n$ such that v_i in $L_{\delta\ i}$ and δ_i eq δ_j implies $v_i = v_j$. Here and later $..L_\delta..$ represents a tuple of pattern sets L_δ one for each δ in Δ.

Clearly we could have formulated the definition using "rewriting steps and derivations". We have chosen the equivalent fixed point definition, both because it generalises more naturally to graph grammars and to emphasize that it is more general than the "pattern languages" of the formal languagists. Their definition corresponds to requiring "only one nonterminal" in ours (one gets the original Marcus contextual languages if also requires "only one nonterminal occurrence in a pattern"). Their pattern languages are incomparable with the contextfree string languages whereas we have

Theorem 1 String pattern languages strictly include the contextfree languages.
Proof: Contextfree languages are exactly the discrete string pattern languages (look at the conversion of patterns to contextfree rules in definition 1). The inclusion is strict because $\{a^{2^n}\}$ is not a contextfree language but it is generated by the following pattern multigrammar:
$\Sigma = \{a\}$ $\Delta = \{S\}$ $\Pi_S = \{a, S' S' \}$

Example 1 Leyton processes
How do we recognise shapes? Many experiments have shown that our eyes rapidly shift (the socalled sacchidic movements) between points of maximum

and minimum curvature. These points also play a critical role in the theory of memory developed by the psychologist Leyton [Le,HL]. In this theory the shape of physical objects reveals the history of the internal and external processes that caused them to be as they are. Computer evidence for this theory from shape analysis of the Hawaiian volcanic islands can be found in [La93]. Consider the figure

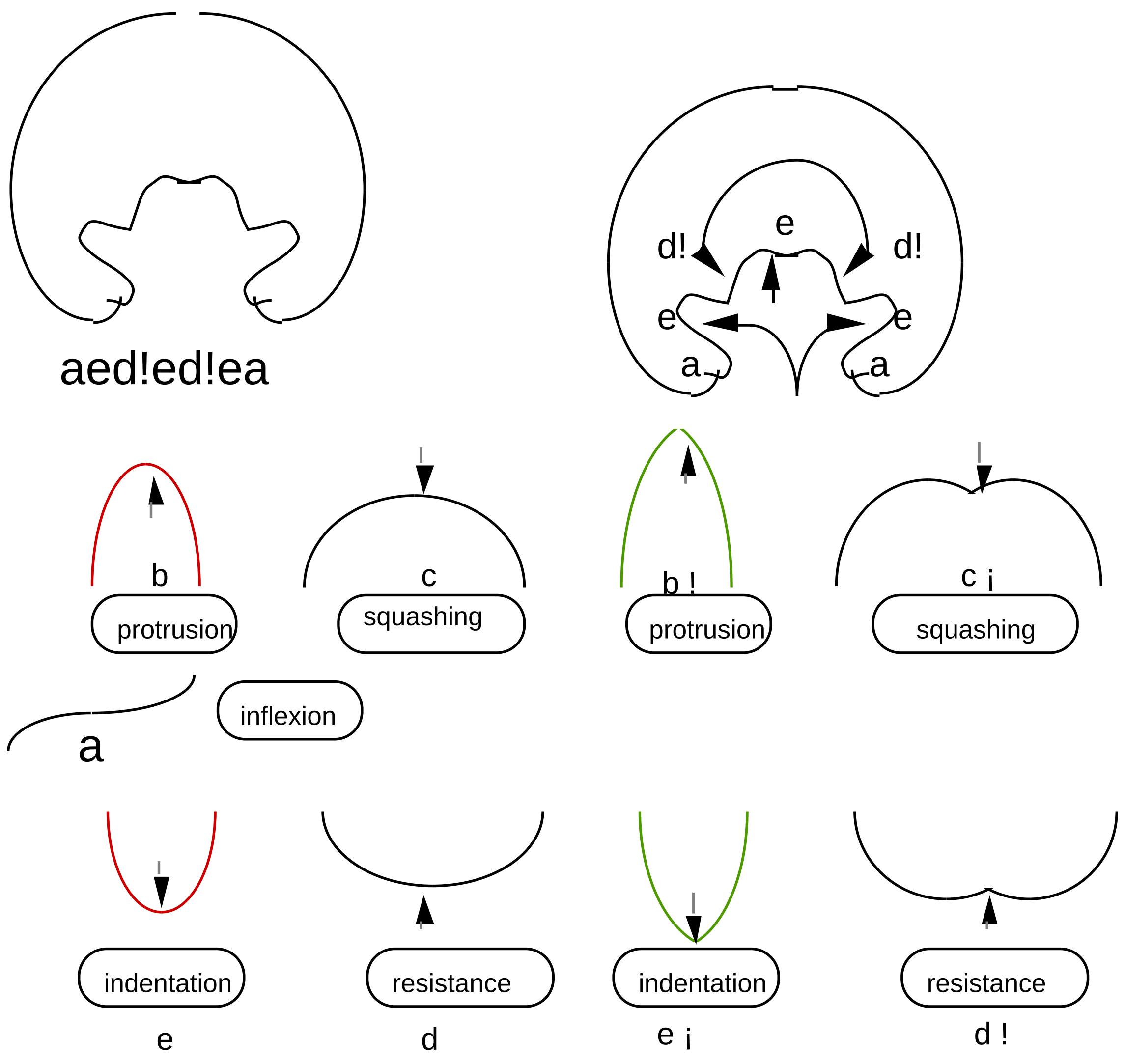


figure 1 Interpretation of the "island" multigrammar

The possible 2D shapes are given by the first pattern language of the "island" multigrammar

$\Sigma = \{a, b, c, d, e, !, ¡\}$ $\Delta = \{0,1,2,3,4\}$

$\Pi_0 = \{ \lambda, 1\ 2 \}$

$\Pi_1 = \{ b, b\ !, 1\ 2\ 1 \}$ = "protrusions"

Π_2 = { c, c ¡, 2 1 2, a 4 a }	= "squashings"
Π_3 = { d, d !, 3' 4 3', a 1 a}	= "resistances"
Π_4 = { e, e ¡, 4 3 4}	= "indentations"

The underlying CFG is the same as in [HL] but we have placed one equivalence to capture "negative curvature maxima are very local"

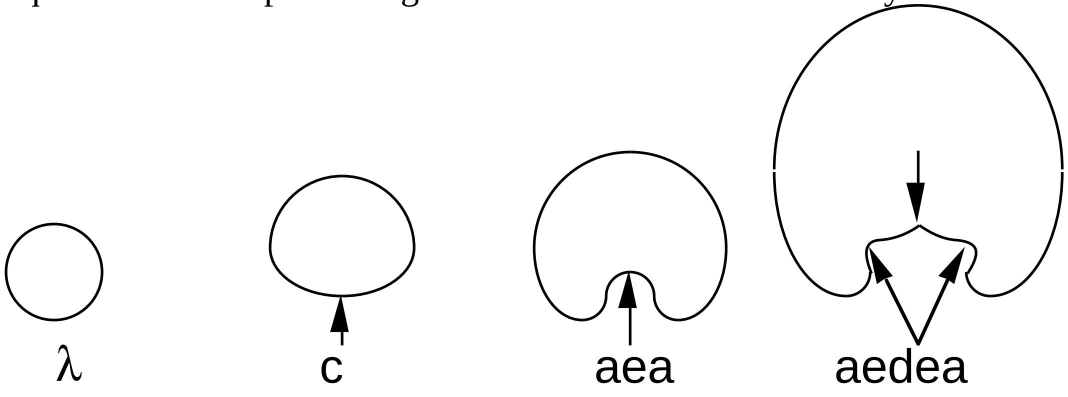


figure 2 Shapes generated by the "island" multigrammar

Example 2 Kolams and Lindenmayer languages

There is a close connection between fractal pattern languages and Lindenmayer languages even although the synchronisation mechanism is different. The OL grammars in [PKV89] for the three kolams in figure 3 correspond to the pattern multigrammars:

"Snake" Σ = {f,+,-} Δ = {S,X}
Π_S = {f + X' f + f + X' f}
Π_X = {l, X' f - f - f + X' f + f + X' f - f - f + X' }

"Anklets of Krishna" Σ = {f,+,-} Δ = {S,X}
Π_S = {- X' - - X'}
Π_X = {l, X' f X' - - X' f X' }

"Bag of Candles" Σ = {f,+,-} Δ = {S,X,Y,Z}
Π_S = {- X' - - X'}
Π_X = {Z f Y - - Z f Y , X' f X' - - X' f X' }
Π_Y = {f + + f f f f - - f - - f f f f + + f + + f f f f - - f}
Π_Z = {f - - f f f f + + f + + f f f f - - f - - f f f f + + f}

figure3 Three kolams

Before one can generalise pattern multigrammars to other worlds than strings, one must make a suitable abstraction. The abstraction of contextfreedom in [Co87] shows the way.

Def.3.A *substitution structure* Ξ consists of Σ a terminal alphabet, Δ a nonterminal alphabet, and a substitution space $[]\Pi$ of patterns over Σ and Δ. The elements of $[]\Pi$ are the patterns over Σ and Δ, the objects of $[]\Pi$ are the patterns over Σ, a Ξ- language is a set of objects of $[]\Pi$. The result of substituting any objects $e_1,e_2,....e_n$ for the n Δ- occurences in any element e is an object of $[]\Pi$, written as $e[\Delta \text{<-} <e_1,e_2,....e_n>]$. There must be a well ordering of the objects of $[]\Pi$ such that each e_i is before $e[\Delta \text{<-} <e_1,e_2,....e_n>]$.

Def.4. A substitution structure $\Xi = \{\Sigma, \Delta , []\Pi\}$ is *sorted* if there is a sort set $[]\Delta$, such that every pattern has a sort (its *skin*) and every occurrence of a nonterminal in a pattern has a sort (its *interface*).

Comment: Every substitution system can be trivially sorted (take any singleton for $[]\Delta$), but such a trivial sorting does not help in defining substitution.For a nontrivial sorting one need only define $e[\Delta \text{<-} <e_1,e_2,....e_n>]$ when each e_i has the same sort as the nonterminal it is replacing; if this condition is not met, the result of the substitution can be defined arbitrarily.

Def.5. A *Ξ– pattern multigrammar* Γ consists of a substitution structure Ξ , and for each S in Δ a set Π_S of patterns over Σ and Δ. The underlying contextfree grammars of Γ are given by choosing a nonterminal from Δ, then converting each pattern π in Π_S to the rule S -> p where p is π without its adornments. A Ξ-pattern multigrammar Γ is *sorted* if Ξ is sorted and there is a function sort:$\Delta \rightarrow []\Delta$ such that

π in Π_S implies skin(π) = sort(S)

δ is occurrence of S in π implies interface(δ) = sort(S)

Def.6. A *Ξ– pattern language* L is generated by a pattern multigrammar Γ if L= L_S for some S in Δ where L_S is the S-component of the least fixed point of the language function Φ defined by

$$w \text{ in } \Phi_S[....L_\delta...] \text{ iff } w = \pi[\Delta \text{<-} <v_1,v_2....v_n>]$$

for some pattern π in Π_S and objects $v_1,v_2....v_n$ such that

$$v_i \text{ in } L_{\delta_i} \text{ and } \delta_i \text{ eq } \delta_j \text{ implies } v_i = v_j.$$

Example 3 Text Languages

Let us look at the pattern version of the very new contextfree text languages [EPR94]. A word on an alphabet S can be defined as a finite function f:V -> S and a linear order ρ of V; a *text* on an alphabet S can be defined as a finite function f:V -> S and two orders $<\rho_{vis}, \rho_{hid}>$ of V. One writes texts as a X[2,1] instead of:

$f(1) = a$, $f(2) = X$, $\rho_{vis} = 1 < 2$, $\rho_{hid} = 2 < 1$.

We define a substitution space $[]\Pi$ of patterns over Σ and Δ by taking texts on Σ and Δ as the elements of $[]\Pi$. The result of substituting any texts $e_1,e_2,...e_n$ for the n Δ.- occurences in any text e is the text $e[\Delta <\!-\!<e_1,e_2,....e_n>]$ given by :

$V' = V + V_1 + V_2 +....+ V_n$ - {v in V such that f(v) is nonterminal}

$\rho_{vis}' = \rho_{vis}$ except $V_1...V_n$ with their visible orderings inserted

$\rho_{hid}' = \rho_{hid}$ except $V_1...V_n$ with their hidden orderings inserted.

Now that we have a substitution system Ξ , definitions 5 and 6 give us text pattern multigrammars and text pattern languages. If only discrete text patterns are allowed, we get precisely the contextfree text grammars and languages of [EPR94]. Consider the interesting text multigrammar:

$\Sigma = \{a\}$ $\qquad\qquad \Delta = \{S,X,Y\}$

$\Pi_S = \{ X, Y \}$

$\Pi_X = \{ a, a\ X\ [2,1] \}$

$\Pi_Y = \{ a, Y'\ Y'[1,2] \}$

This generates the text pattern languages:

$L_S = L_X \cup L_Y$

$L_X = \{ a^n\ [n...1] \quad$ where $n > 0 \}$

$L_Y = \{ a^{2^n}\ [1...2^n] \quad$ where $n > 0 \}$

This is interesting because L_S is NOT a context free text language (shown by the argument in [EPR94] for a slightly different language), even although the underlying string language is regular.

Theorem 2 The Ξ- pattern languages are closed under union. The Ξ- pattern languages are closed under quasi-intersection, provided that Ξ allows patterns with 2 nonterminals.The Ξ- pattern languages have a decidable membership problem. The Ξ- pattern languages have an undecidable emptiness problem, provided that Ξ allows patterns with 2 nonterminals and the Ξ- contextfree languages have an undecidable intersection problem.

Proof: Suppose L_X and L_Y are pattern languages generated by disjoint multigrammars Γ_X and Γ_Y. Let $\Delta = \Delta_X \cup \Delta_Y \cup \{S\}$.

If one adjoins $\Pi_S = \{X,Y\}$ to Γ_X and Γ_Y, one gets a multigrammar that generates $L_S = L_X \cup L_Y$. If one adjoins $\Pi_S = \{X'\ Y'\}$ to Γ_X and Γ_Y; one gets a multigrammar that generates the quasi-intersection L'_S defined by:

$X Y[\Delta \text{ <- } \langle W,W \rangle]$ in L'_S iff W in both L_X and L_Y.
If one could decide whether L'_S were empty, one could decide whether L_X and L_Y had an empty intersection. However one can decide whether or not an object W is in a pattern language L. One has the pseudo-Ada algorithm:

```
function Member(W,nonterm):Boolean;
begin for each nonterm pattern P
    do if W fits P then
        if Member($W_i$,$\delta_i$) for all $\Delta$-occurrences in P
        then begin Print(P); return True; end;
    end;   return False;
end;
```

The well ordering requirement on substitution systems ensures that this algorithm always terminates.
Corollary: String pattern languages have an undecidable emptiness problem, because they are closed under quasi-intersection.
Those who dislike imperative programs may prefer their reformulation as logic programs as in the

Example 4 Meanders
In quantum physics a fascinating topological problem arose which was solved by the use of meanders [Ar88,LZ90]. A meander is a closed curve in the plane intersected by a line; there are four kinds of intersections of a closed curve and a line; two meanders are distinct if they have different kinds of intersections. The problem of counting the number of distinct meanders is still open, and a grammar that generated all and only meanders would probably solve the problem.

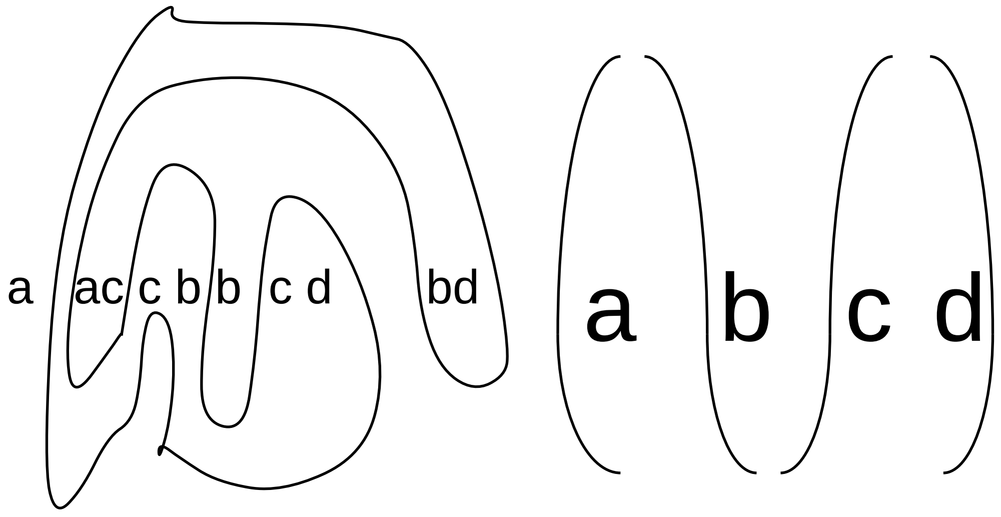


Fig.4 Interpretation of "meander" multigrammar

As an object in the pattern language L_S, generated by the following pattern multigrammar, may contain more than one meander, we have not quite solved the counting problem.

"meander" $\Sigma = \{a,b,c,d\}$ $\Delta = \{S,U,L\}$

$\Pi_S = \{ U' L' \}$

$\Pi_U = \{ \lambda, U U, a U c, a U d, b U c, b U d \}$

$\Pi_L = \{ \lambda, L L, a L b, a L d, c L b, c L d \}$

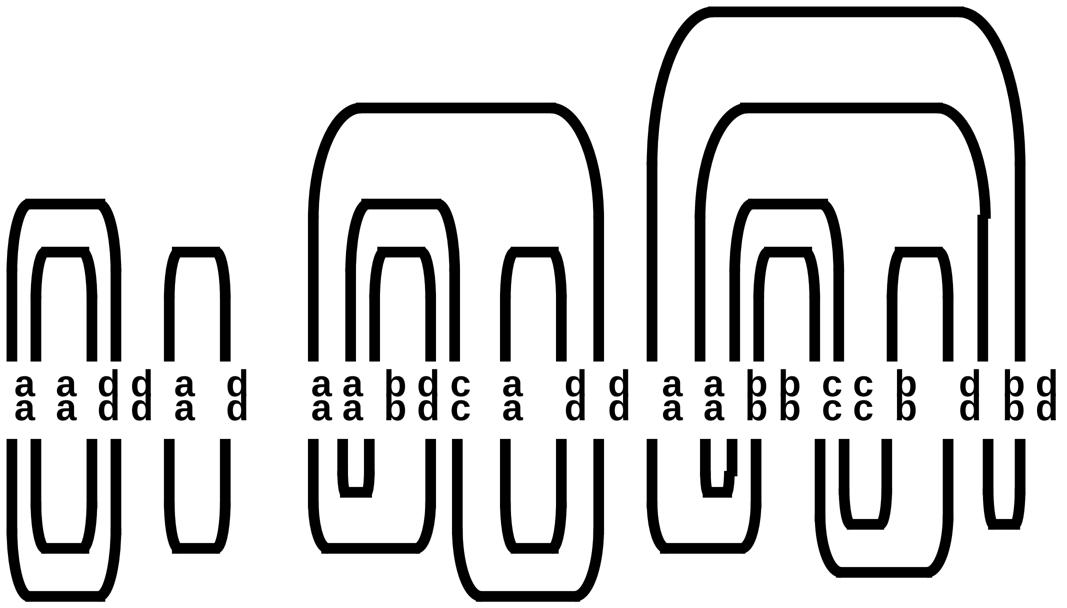

a a d d a d a a b d c a d d a a b b c c b d b d
a a d d a d a a b d c a d d a a b b c c b d b d

Fig.5 Some shapes generated by the "meander" multigrammar

The promised reformulation of the parsing algorithm as a logic program is:

```
S(<x,y>):- U(x), L(y), x = y.
U(<>) .                      L(<>).
U(<x,y>):- U(x), U(y).       L(<x,y>):- L(x), L(y).
U(<a,x,c>):- U(x).           L(<a,x,b>):- L(x).
U(<a,x,d>):- U(x).           L(<a,x,d>):- L(x).
U(<b,x,c>):- U(x).           L(<c,x,b>):- L(x).
U(<b,x,d>):- U(x).           L(<c,x,d>):- L(x).
```

Note 1: The expressive power of logic programming is not being used here: replacing the first "rule" by "S(x):- U(x), L(x)." would give true intersection, not quasi-intersection; Prolog's length predicate would allow many useful grammar constraints; moving to constraint logic programming [MTP94] would give still more expressive "context-free" grammars and languages.

Such natural generalisations of Ξ- pattern multigrammars will be studied in a later paper.
Note 2: Often logic programs have not only the usual "logical" and "procedural" readings but also a "graphical" reading as a set of structural rewriting rules. Of the many papers on this topic the reader might like to look at [MR92,Co87]

#2 Cellular Pattern Multilanguages

Imagine a partition of a 2D- (3D-) Euclidean subspace into rectangles (rectangular boxes), each labelled by a letter from Σ or Δ. If we define a pattern to be such a labelled partitioned subspace that is itself rectangular, then substitution can be "scaling,then replacement". Before giving a formal definition of this kind of pattern multilanguage, let us look at an example.

Example 5 Matrix Grammars
One of the many kinds of matrix grammar in the literature is defined by

- a string grammar G_0 to give the rows
- for each terminal letter t of G_0 a string grammar G_t to give the columns.

To show how such a matrix grammar can be reformulated as a pattern multigrammar, let us redo the meander example

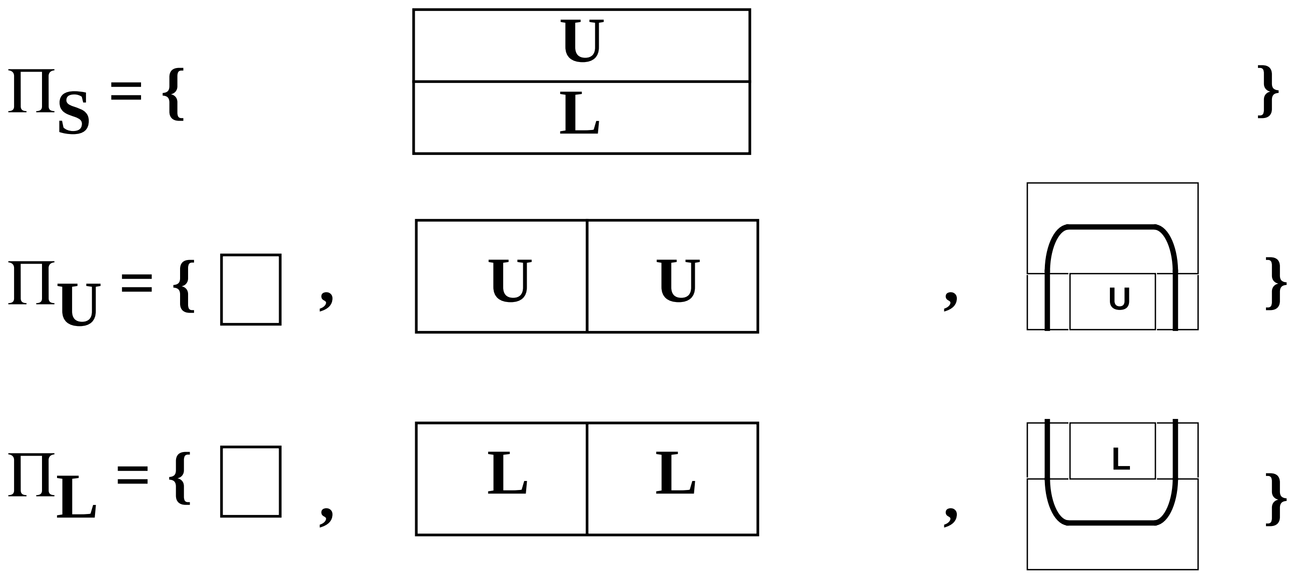


figure 6 matrix grammar and typical object

Notice that this pattern multigrammar can generate non-meanders, because it cannot capture " upper and lower rows have the same length".

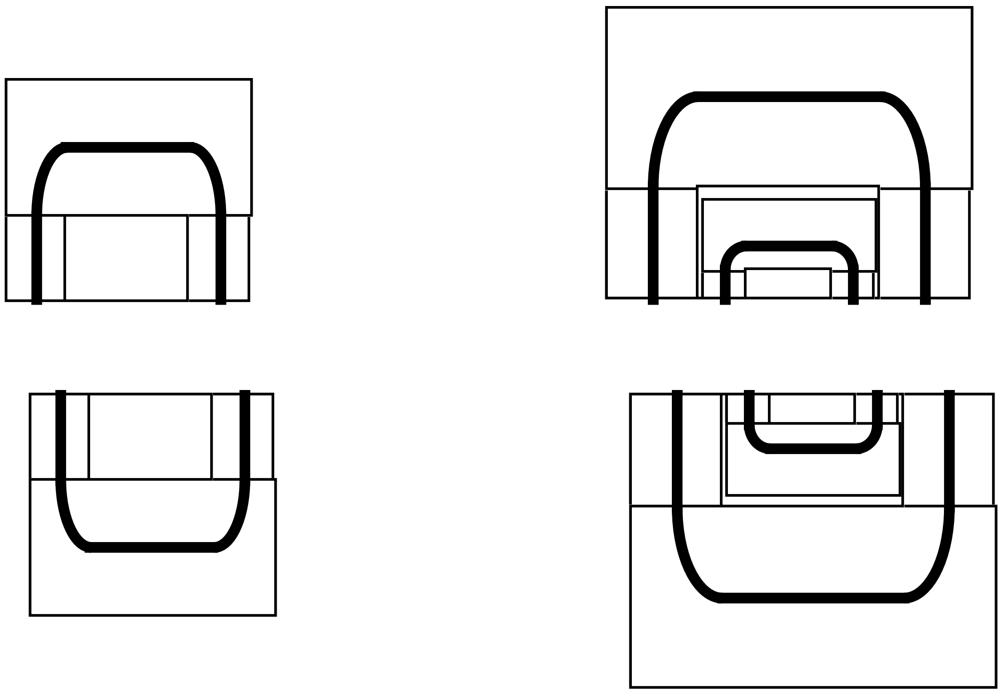
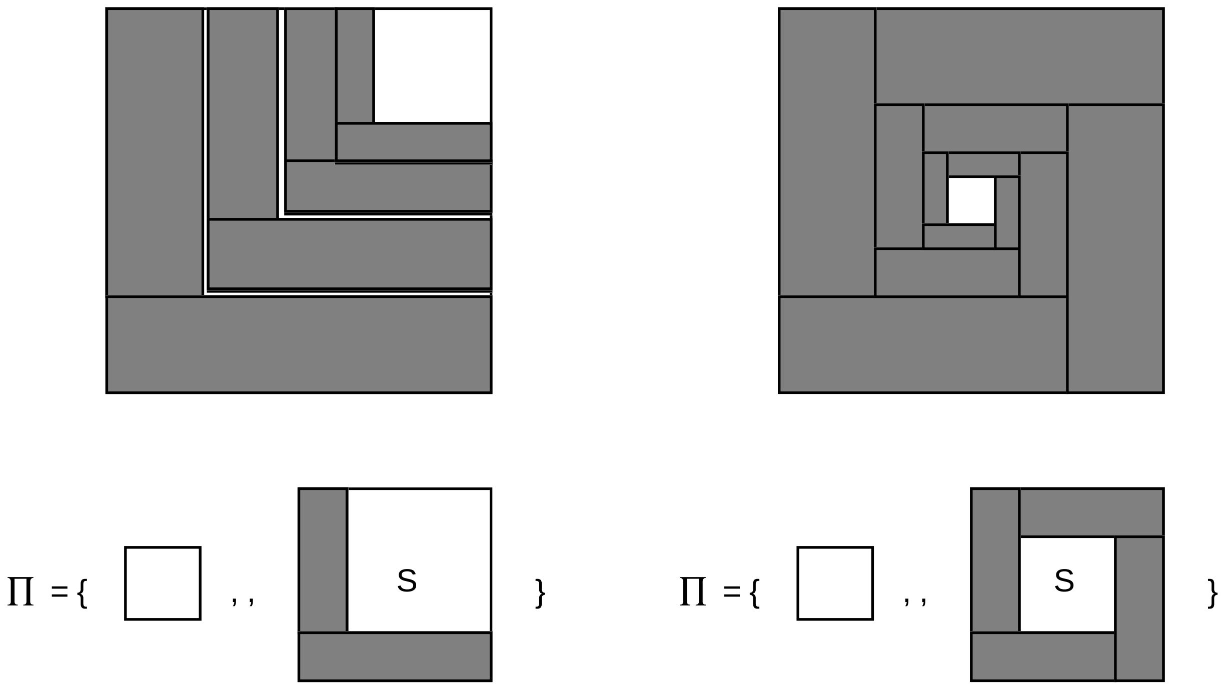


figure 7 typical lower and upper halves of meanders

Example 6 Floor and Building layouts

Figure 8 shows two panelled floors, rectangular partitions of rectangles. Such panellings have been studied by mathematicians [SSR92] who have been interested in such properties as *sliceabilty* , whether the partition can be decomposed by cuts which go from one edge of a part to another. Figure 8 also shows two pattern multigrammars, one generating sliceable pannelled floors, the other not.



Σ = Shaded Rectangles

figure 8 Floor panelling pattern multigrammmars

Figure 9 shows a layout for a building estate, another kind of rectangular partition of rectangles. The pattern multigrammar for this and similar layouts has nonterminals for two building contractors; it uses pattern equivalences to ensure that the building contractors are treated equally.

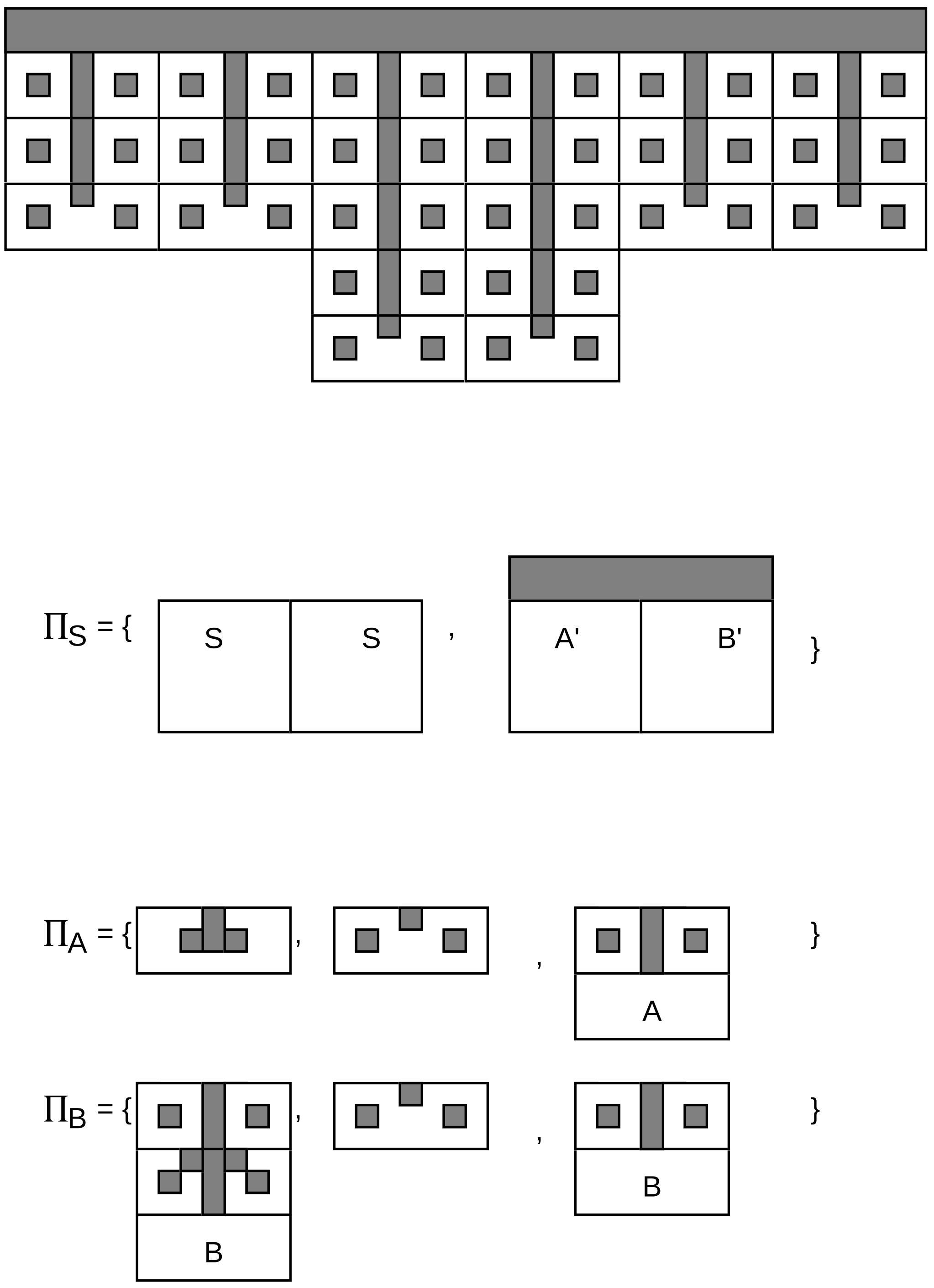


figure 9 Estate layout and an estate pattern multigrammmars

Example 7 3D Fractals

Figure 10 shows a pattern multigrammar for a typical 3D fractal

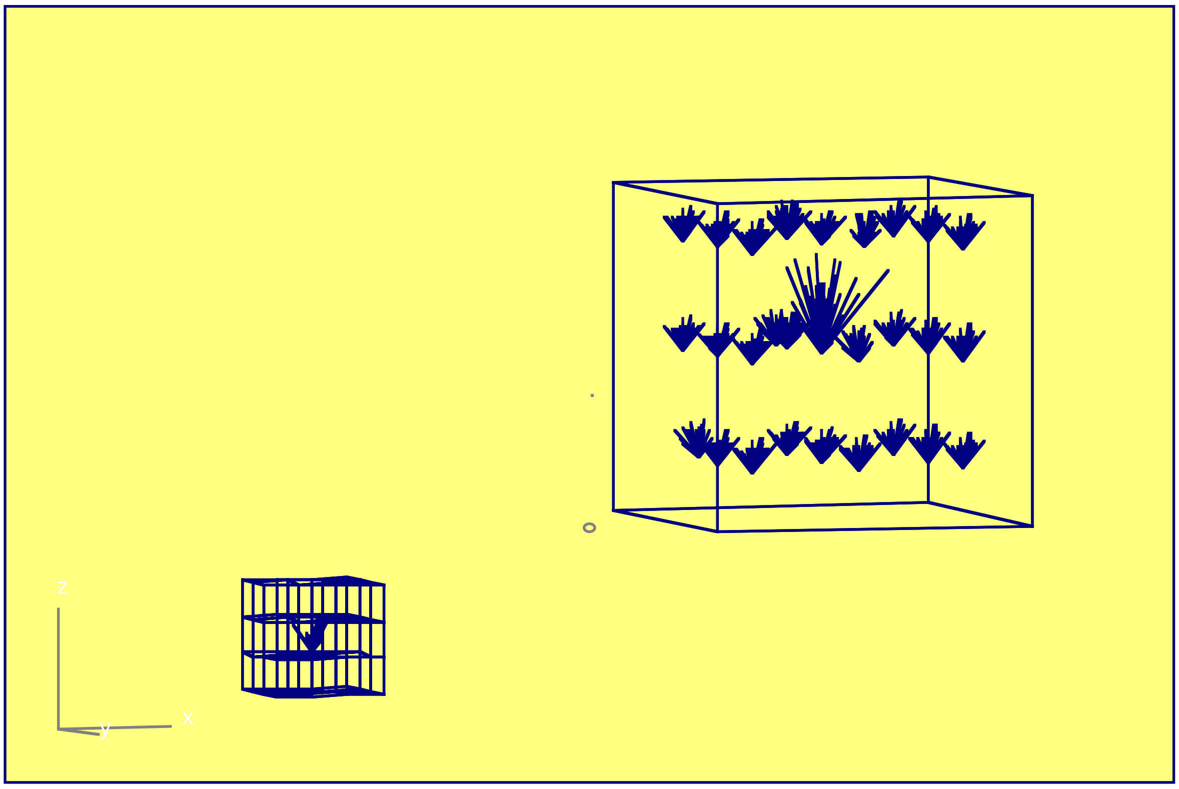


figure 10 3D fractal pattern multigrammar

For the formal definition of cellular pattern multilanguages we must formulate our labelled partitions of Euclidean spaces as a substitution structure Ξ. First we define a cellular pattern over Σ and Δ as a rectangular Euclidean subspace, partitioned into rectangular parts, each labelled by a letter from Σ or Δ, such that: each coordinate of each part is a rational number. This requirement ensures that each pattern has rational Skin Parameters (span of pattern in each dimension and coordinates of "centre") and each part of each pattern has rational Interface Parameters.

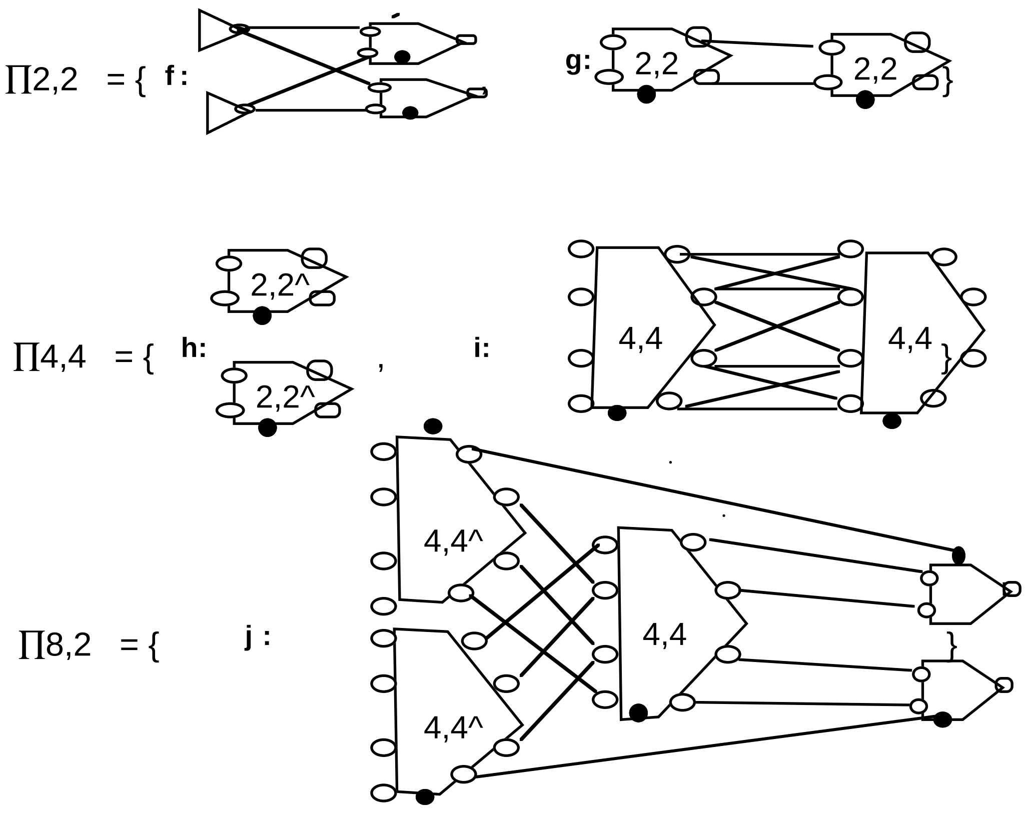
Now we have a substitution space []Π of patterns over Σ and Δ and we must define the result of a substitution in a pattern: e[Δ <- <e_1,e_2,....e_n>]. For each e_i in the substitution we have rational skin parameters for e_i and a rectangular part in e with rational interface parameters, so these parameters give a linear embedding of e_i in the appropriate part of e. Note that none of our cellular pattern multigrammars are sorted, even although the underlying cellular substitution structure is sorted and the sorts are used in defining the results of substitutions.

#3 Hypergraph Pattern Multilanguages

Even if one is only interested in generating graphs, it is usually more convenient to exploit the greater expressive freedom of hypergraph grammars. Suppose each hyperedge of a hypergraph is labelled by a letter from Σ and Δ. For each of the many ways of substituting a hypergraph for a hyperedge in another hypergraph one can give a formal definition of a particular kind of hypergraph pattern multilanguage.

Example 8 Neural nets

Most forms of neural nets can be specified as hypergraphs- a vertex for each neuron and a labelled hyperedge for each "hidden" neuron. A sorted substitution space for neural nets is given by: the sort of a net is the number of inputs and outputs it has, the sort of a labelled neuron is the number of inputs and outputs it has, and the substitution of a net for a nonterminal labelled neuron of the same sort is given by identifying inputs and outputs. Figure 11 shows a sorted neural net pattern multigrammar and some of the neural nets it generates. These neural nets are "emotional" in that they mimic the chemical activity of the brain. A later paper will explain this new kind of net; for now it suffices to say that the usually implicit "bias input" of each neuron is made explicit (the black dots in the figure)



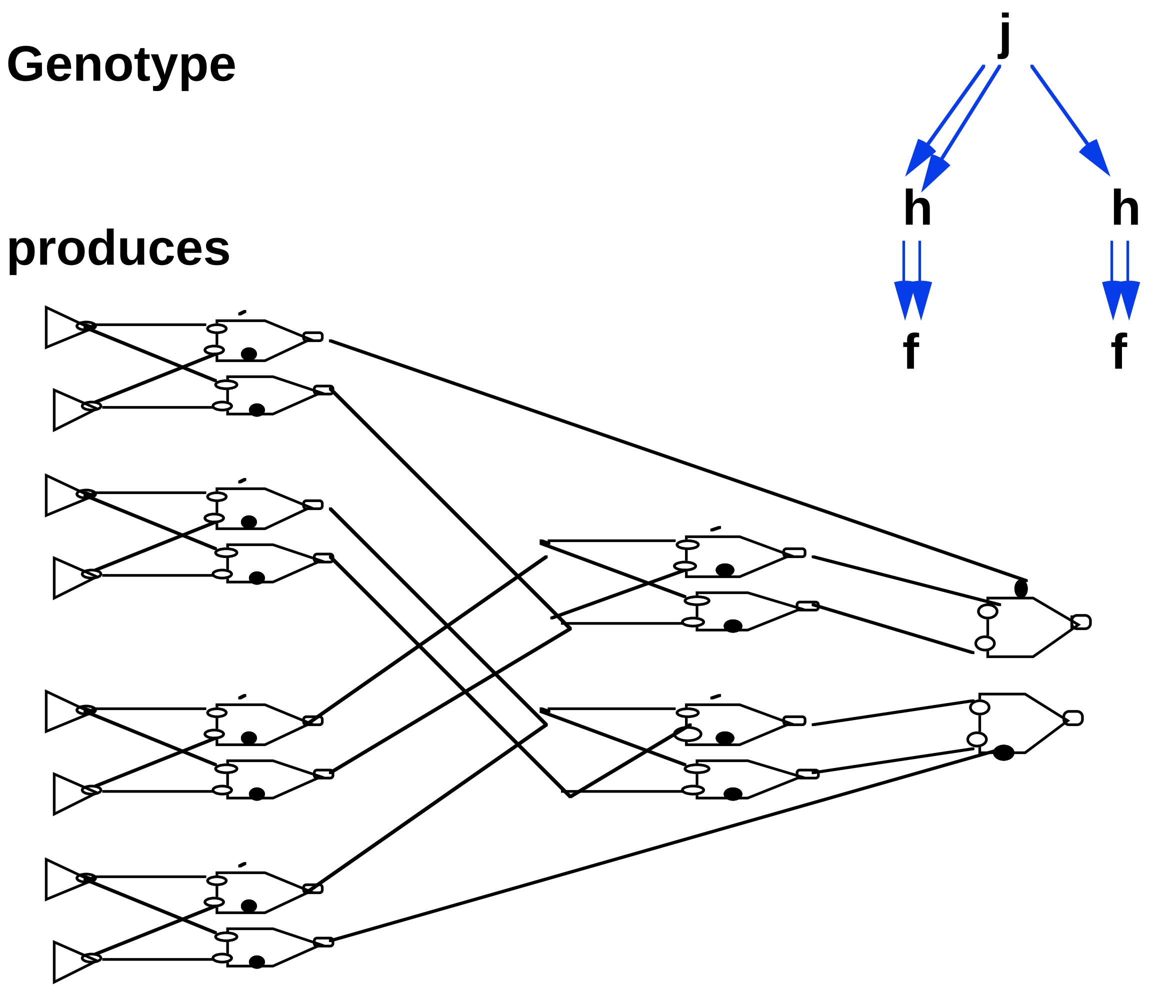


Fig 11 Neural net pattern multigrammar

Large neural nets are notoriously hard to understand as they behave in obscure ways. Using pattern equivalences to make such nets structured and modular is a natural way of alleviating this problem. A pattern multigrammar may also be useful in the proliferating art [FF94] of using genetic algorithms to generate neural nets (the genotypes are derivations in the multigrammar). Kitano, Mjolness and others have used matrix grammars (remember example 5) and genetic algorithms to generate neural nets. Conversely genetic algorithms may be useful in the proliferating art [CO94] of inducing grammars from examples, even inducing pattern multigrammars.

For the formal definition of hypergraph pattern multilanguages we must formulate some notion of labelled hypergraphs as a substitution structure Ξ .

The simplest notion is Δ-ranked directed hypergraphs- hyperedges are sequences of vertices, []Δ is the positive integers, each letter in Δ has a rank, and each hyperedge labelled by δ in Δ is a sequence of length rank(d). First we define a hypergraph pattern over Σ and Δ as a Δ-ranked directed hypergraph with a *skin* (not only is each hyperedge labelled by a letter from Σ or Δ, but there is an extra skin hyperedge). Just as the sort of the directed hypergraph is the length of its skin hyperedge, so the "interface" sort of each of its labelled directed hyperedges is given by their lengths.

Now we have a sorted substitution space []Π of patterns over Σ and Δ and we must define the result of a substitution in a pattern: e[Δ <-<e_1,e_2,....e_n>]. For each e_i in the substitution we have a skin hyperedge for e_i and an interface hyperedge in e with the same length, so these parameters define the result of replacing the nonterminal hyperedge by e_i: identify the vertices of the nonterminal hyperedge with the corresponding skin hyperedge vertices, add the other e_i-vertices and all e_i-hyperedges except its skin. Note that hypergraph pattern multigrammars are usually sorted, because one only wants to define substitution when skins and interfaces have the same sort (have the same number of inputs and outputs for example 8, are vertex sequences of the same length for Δ-ranked directed hypergraphs).

#4 Combinatorial Pattern Multilanguages

Graph grammars have proved useful for describing the developement of biological organisms (partitions of 3D space into labelled cells) and geographical analyses (partitions of 2D space into labelled regions). Many of the graph grammars used have been somewhat ad hoc because there is no obvious way of defining spatial substitution. It seems cleanest to follow the combinatorial topologists [So91] and redefine the notion of hypergraph; from now on a labelled hypergraph over Σ and Δ contains a set D of *darts,* a map from D to Σ and Δ, and two partitions of D called Vertex and Edge. A labelled hypergraph may contain more information than this; putting an order on D gives the labelled directed hypergraphs of #3, putting two orders on D gives the text languages of example 3; further examples will suggest other useful extra information.

Example 9 planar graphs and maps

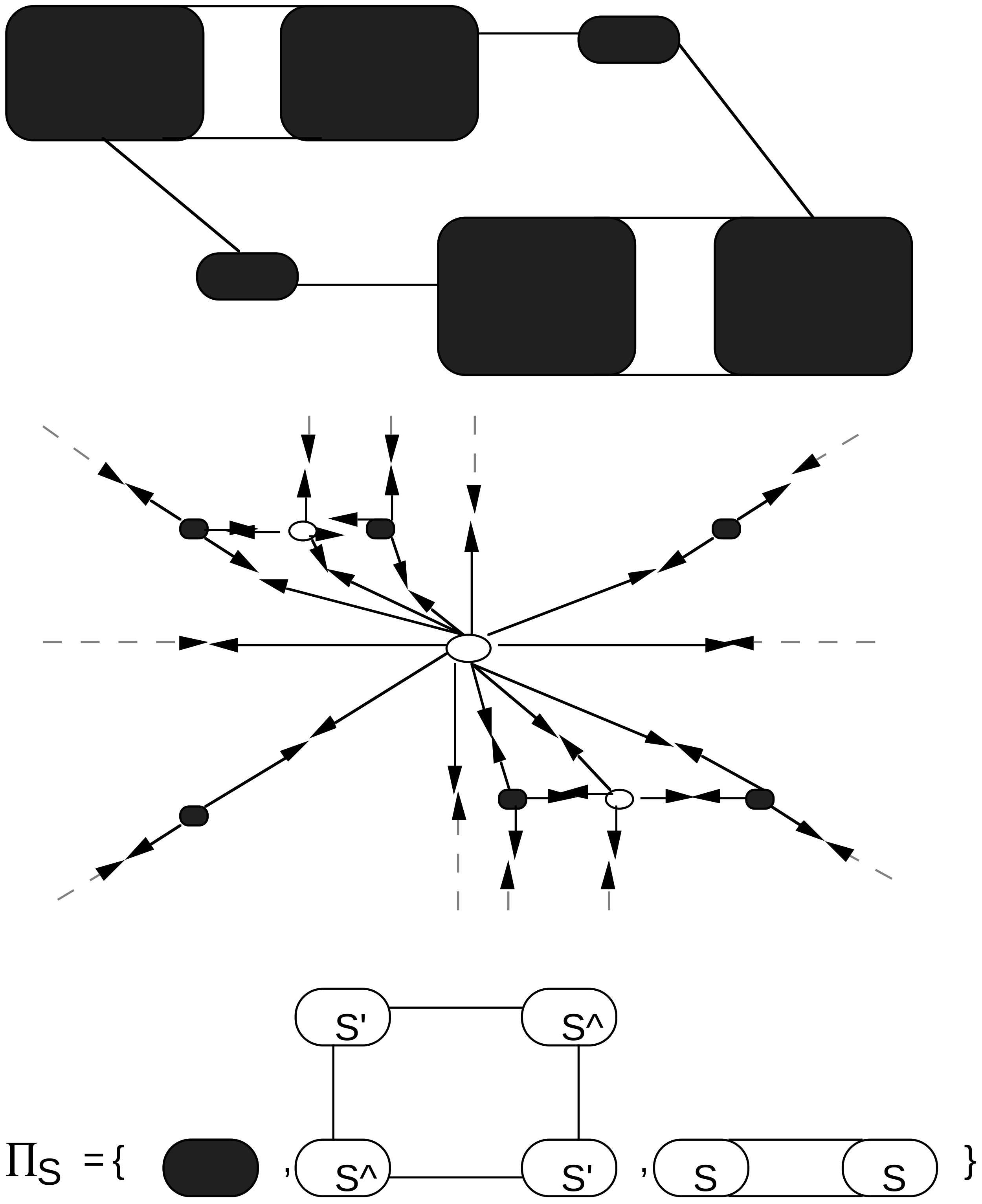


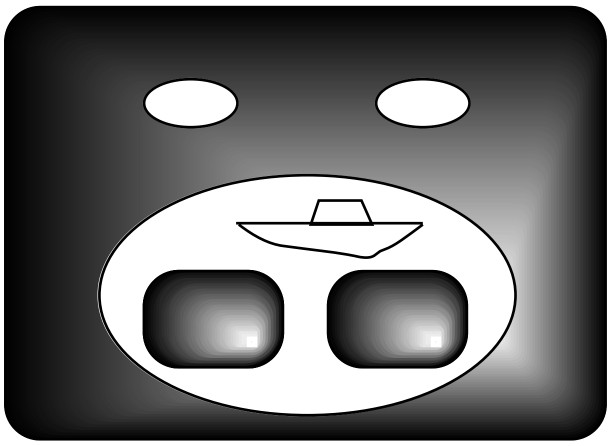
figure 12 Planar maps and a map pattern multigrammar

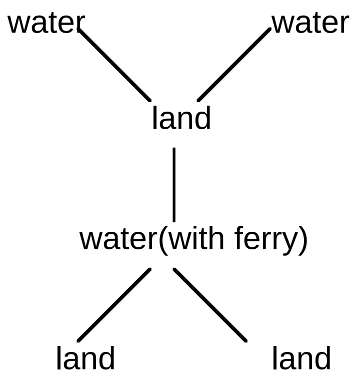
Since the formulation of the four colour conjecture, many mathematicians have been interested in graph colourings. Figure 12 shows a typical coloured map, its formulation as a coloured planar graph, and a map pattern multigrammar that generates it. Imagine a *dart* on either side of each "boundary"; the boundaries give the Edge permutation of darts and circular traversal of the boundaries of each "region" gives the Vertex permutation of darts. This dart representation of a planar map is completed by defining its skin as the Vertex cycle for the outside region of the map. The other Vertex cycles are the interfaces for the inner regions. Substituting a map e for a region v is done by dropping the skin darts of e and identifying their Edge companions with the interface darts of v. As the reader can invent many different ways of "identifying two sets of darts", we need not give a precise definition here.

Example 10 pavings and biological organisms
One can represent 3D structures by introducing a third Paving permutation of darts. Informally this permutation gives the ordering of components around vertices, but we do not have the space here to give a precise, formal definition of the paving substitution system. The reader can probably reconstruct it from the biological example [LL92] in figure 13.

figure 13 Biological organisms and a paving pattern multigrammar

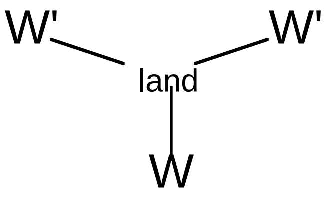
Example 11 LH graphics and other node labellings





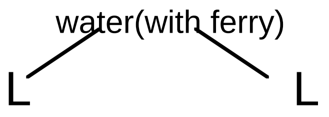
$\Delta = \{ L, W \}$ $\Sigma = \{$ land , water $\}$ attributes = { ?ferry? }

$\Pi_L = \{$ land ,



$\}$

$\Pi_W = \{$ water ,



$\}$

figure 14 LHgraphics and a LH pattern multigrammar

In [HM90] a way of representing "knowledge" is described; as one can see in figure14, knowledge is represented by an undirected graph with labels and attributes for each vertex in the graph. In the dart approach the set of vertex labels can be taken as the set Σ of terminal dart labels (all terminal darts at a vertex get its labels). The set Δ of nonterminal dart labels is used both for designated "skin" hyperedges and for occasional hyperedges in patterns. This reformulation gives the Δ - sorted hypergraphs for which we describe a substitution structure at the end of this section.

Example 12 Petri Nets

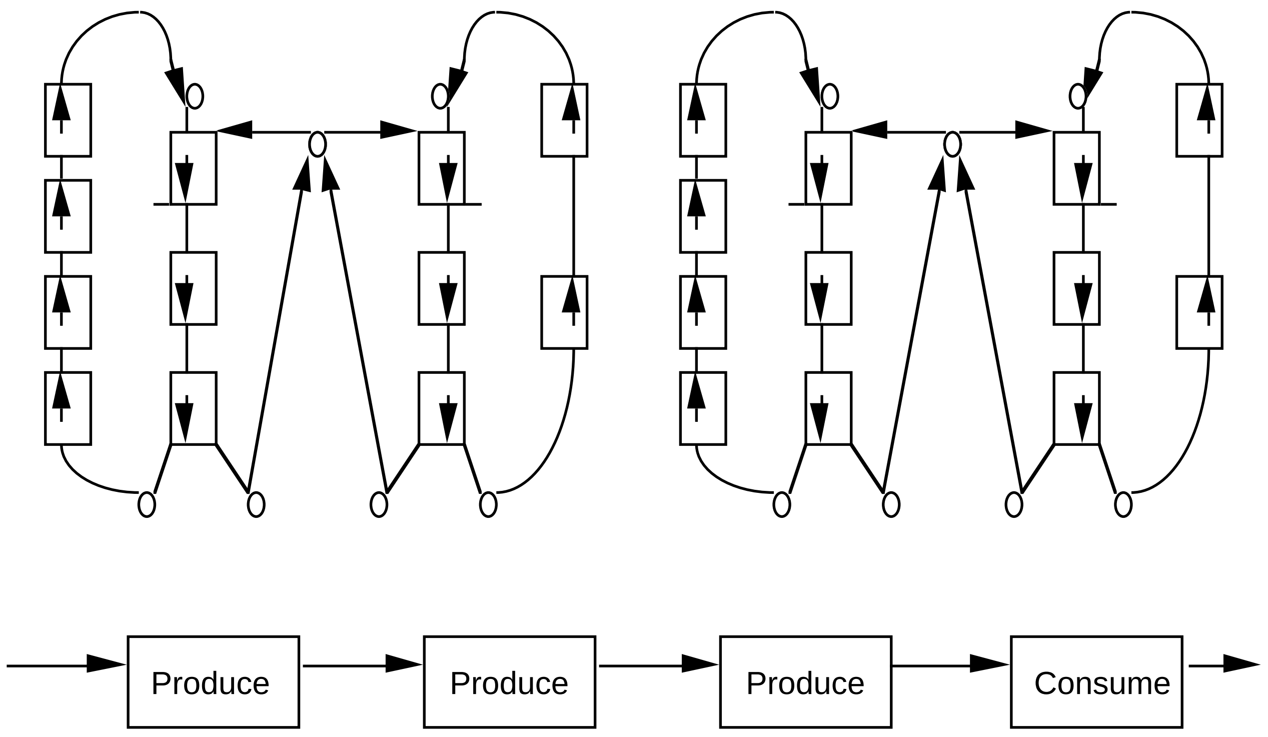


figure 15 typical Petri Net and action net

Although Petri nets are usually drawn as in figure 15, one can define them as hypergraphs, with "places" as vertices and "transitions" as hyperedges. The dart labelling must distinguish between the inputs and outputs of places and transitions. One way of doing this is to let dart labels be pairs < n, tl > , where n is a non-zero integer and tl is a transition label from Σ or Δ. Note that Petri nets have natural skins- the input (output) places, the vertices whose dart labels are all negative (all positive). In defining $e[\Delta \leftarrow <e_1,e_2,....e_n>]$ the skin darts of e_i have to be assigned to the vertices of e. Assuming that the darts of a nonterminal S have different labels, one can decree: skin darts labelled < n, tl> are assigned to the same vertex as the nonterminal dart labelled < n, S> (if no such dart exists, then the skin vertex is retained and keeps its dart labelled < n,tl>). A glance at figure 16 will make this more clear

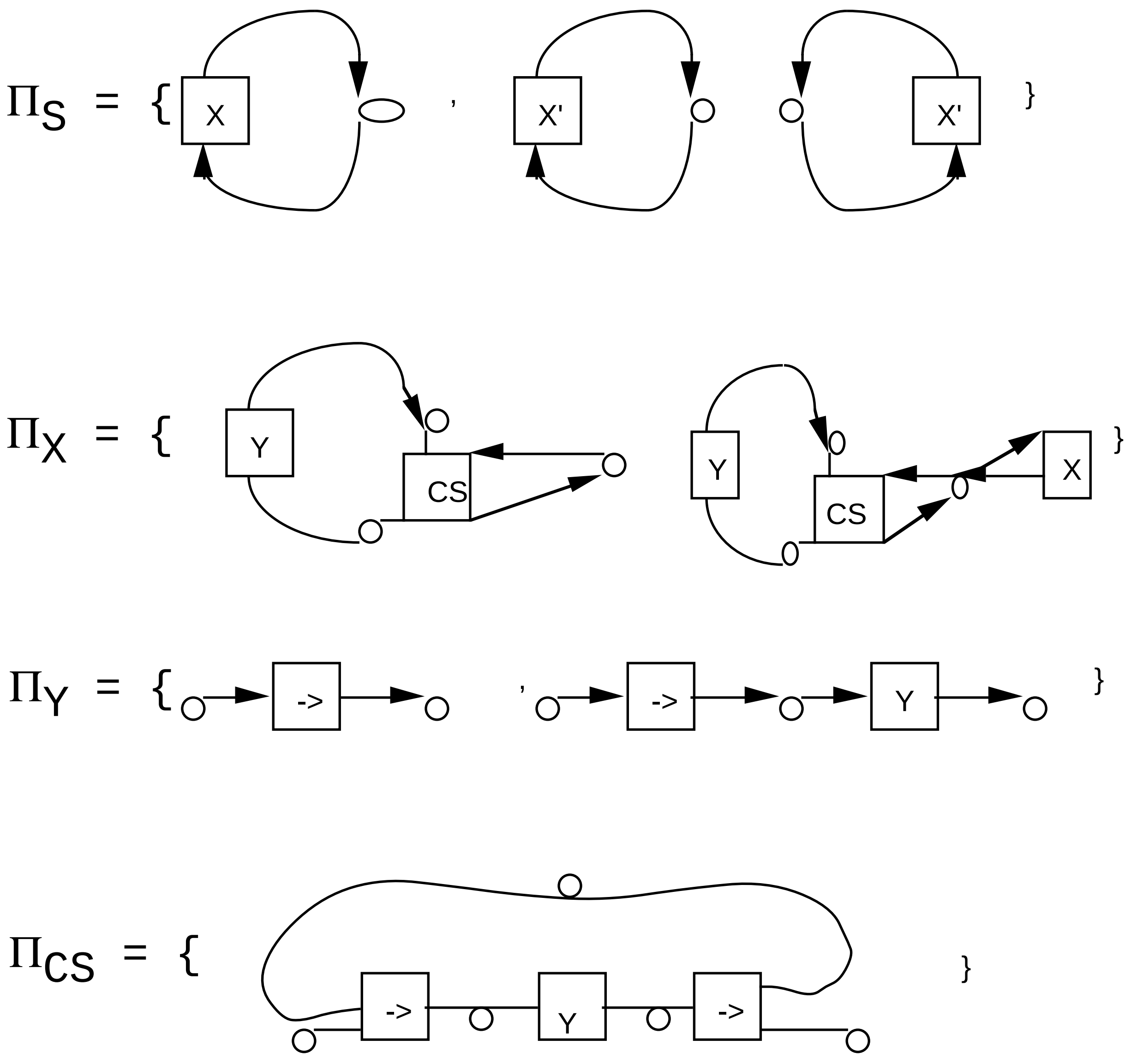


figure 16 A Petri net pattern multigrammar

Large Petri nets are hard to understand, but using pattern equivalences to make such nets structured and modular is a natural way of alleviating this problem.

Example 13 Action Nets and Planning
In practice and in theory one often represents plans as action nets like that in figure 15. If one defines "states" as maps from "features" to "values", one can define action nets as hypergraphs with "features" as vertices and "actions" as hyperedges. The dart labelling must distinguish between the preconditions and postconditions of actions. One way of doing this is to let dart labels be triples < fe, int, al > where fe is a positive postcondition feature or negative

precondition feature, int is an interval of values and al is an action label from Σ or Δ. Then one can describe the interface sort of an action occurrence as the features mentioned in its dart labels, one can describe the skin sort of an action net as the features mentioned in its input or output dart labels, and one can describe substitution as " matching features and ignoring value-intervals of nonterminals".

Plans in the form of large action nets are hard to understand, but using pattern equivalences to make such nets structured and modular may alleviate this problem.

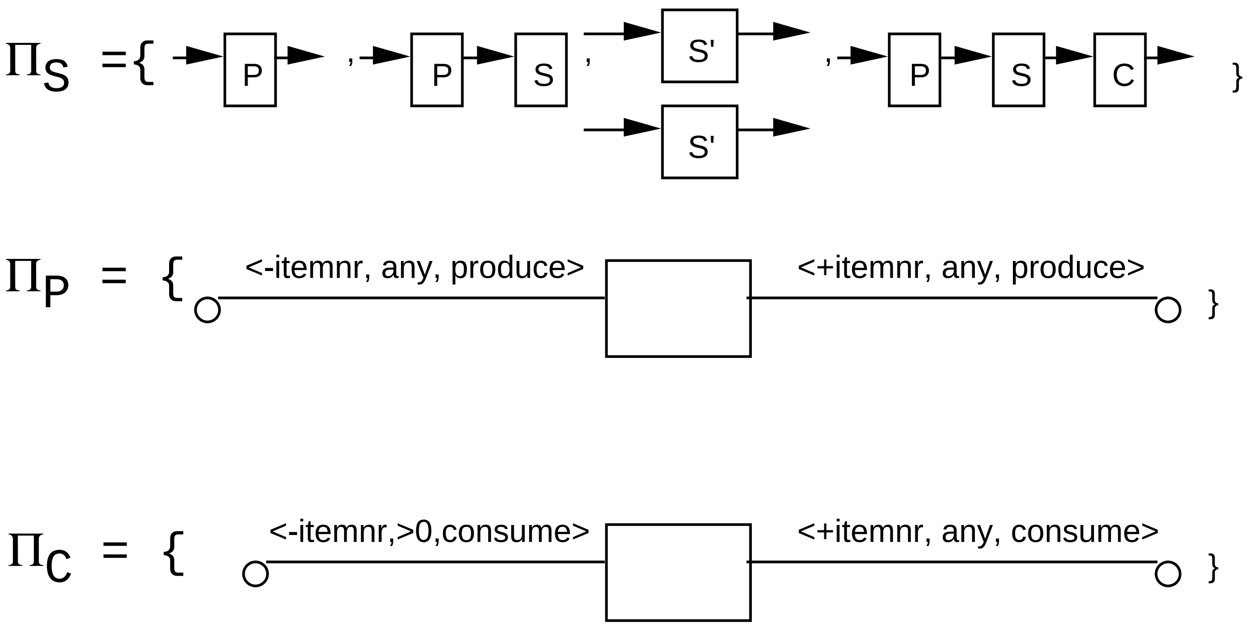


figure 17 An action net pattern multigrammar

For the formal definition of combinatorial pattern multilanguages we must formulate our new notions of labelled hypergraphs as substitution structures. The simplest notion is Δ-sorted hypergraphs- Δ is partitioned, and each hyperedge having a dart with nonterminal label δ has all its darts labelled by nonterminals from the same part of Δ as δ. First we define a hypergraph pattern over Σ and Δ as a Δ- sorted hypergraph with a designated skin-hyperedge, whose darts are labelled by nonterminals from the same part of Δ.

Now we have a substitution space []Π of patterns over Σ and Δ and we must define the result of a substitution in a pattern: $e[\Delta <- <e_1,e_2,....e_n>]$. For each e_i in the substitution we have:

- a skin hyperedge for e_i all of whose darts are labelled by nonterminals from the same part of Δ

- a hyperedge in e whose darts are labelled by the same nonterminals so these parameters define the result of replacing the nonterminal edge by e_i:
- to identify these hyperedge darts with the corresponding skin hyperedge darts, and add all the other e_i-darts.

Note that combinatorial pattern multigrammars are usually sorted, because one only wants to define substitution when skins and interfaces use dart labels from the same part of Δ.

Summary

In this paper we have explained what the pattern version of a contextfree grammar is and why the advantages of contextfreedom are preserved, we have given the pattern version of several popular graph and hypergraph grammars, and we have given striking examples to show that the extra expressive power of patterns is worth having. We have indicated how simple computer programs can generate and analyse pattern languages, but we have not described how they can learn pattern multigrammars from a sample of actual patterns.

References

[An80] D.Angluin "Finding patterns common to a set of strings" J.Comp.Sys.Sci.21(1980)46-62

[Ar88]V.I.Arnol'd ."A branched covering of CP^2 -> S^4, hyperbolicity and projective topology", Sib.Math.J. 29 (1988)717-726

[Co87] B.Courcelle "An axiomatic definition of context-free rewriting and its application to NLC graph grammars" Th.Comp.Sci. 55 (1987) 141-181

[CO94] R.C.Carrasco, J.Oncina (eds.) "Grammatical inference and applications" Springer LNAI 862(1994)

[DPS93] J.Dassow, G.H.Paun, A.Salomaa "Grammars based on patterns" Int.J.Found.Comp.Sci.4 (1993)1-14

[EH94] J.Engelfriet & L.Heyker "Hypergraph Languages of bounded degree" J.Comp.Sys.Sci.48 (1994)

[EPR94] A.Ehrenfeucht & P.ten Pas & G.Rozenberg "Context-free Text Grammars", Acta Informatica 31 (1994)161-206

[FF94] D.S.Fogel, L.J.Fogel "Evolutionary computation" Guest Editorial for special issue IEEE Trans. Neural Networks 5(1994)3-14.

[HL91] P.J.Hayes & M.Leyton "Processes at discontinuities" Proc.Int:Conf.Art.Int.,pp.1267-1272 Detroit1991

[HM90] L.Hess & B.Mayoh "The Four Musicians: analogies and expert systems- a graphic approach" pp 430-445 Springer LNCS532
[La93] T.W.Larsen "Proces grammatik og proces historie for 2D objekter" DAIMI IR-115, Aarhus Univ.Report 1993.
[Le92] M.Leyton "Symmetry, Causality, Mind" MIT press 1992
[LL92] J.Lück, H,B.Lück "Cellworks: an application to plant morphogenesis" in [RS92]
[LZ90] S.K.Lando,A.K.Zvonkin "Meanders", Institut des Hautes Etudes Scientifiques report IHES/M/90/91
[Ma69] S.Marcus "Contextual grammars" Rev.Roum.Math.Pures Appl. 14(1969)1525-1534
[MTP94] B.Mayoh, E.Tyugu, J.Penjam (eds:) "Constraint programming" NATO ASI Series F, Vol. 131, Springer 1994
[MR92] U.Montanari,F.Rossi "Graph grammars as contextdependent rewriting systems: a partial ordering semantics"
Springer LNCS581(1992)232-247
[Na89] R.Narasimhan(ed.) "A perspective in theoretical computer science: Commemorative volume for Gift Simoney" World Scientific 1989
[Pa82] Gh.Paun "Gramatici contextuale" Ed.Acad.Soc.Romania 1994
[Pa94] Gh.Paun "Marcus contextual grammars after 25 years" Bull.EATCS 52(1994)263-273
[PKV89] P.Prusinkiewicz & K.Krithivasan & M.G.Vijayanarayana "Applications of L-systems to algorithmic generation of south indian folk art patterns and karnatic music" pp229-247 in [Na89]
[RS92] G.Rozenberg, A.Salomaa(Eds.) "Lindenmayer Systems" Springer1992
[So91] E.Sopena "Hypermap rewriting: a combinatorial approach" Th.Comp.Sci. 85 (1991) 253-281
[SSR92] R.Simoney & K.G.Subramanian & T.Robinson "Map L-systems with multiple markers" in [RS92]

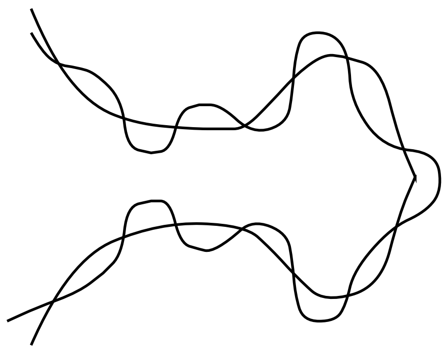
"DNA Pattern multigrammars" by Brian Mayoh

How can one go from the sequence of nucleotides in a DNA molecule to a description of its secondary structure? A popular answer [Se93] is:use one or more grammars to parse the DNA string and the resulting derivation trees give descriptions of the secondary structure of the DNA string. As grammars usually give languages with more than one word, they can be used to parse more than one DNA string, so they can handle genetic variation. The language L generated by a grammar can cover not only the possible alleles of a gene but also the possible mutations that are viable. Biologists have devised several elegant grammars for DNAstrings, but they also hope that machine learning techniques can extract suitable grammars from a sample of "related" DNA strings [Se93].

In this paper we introduce DNA pattern multigrammars and argue that they are particularly appropriate for both machine learning and the description of DNA secondary structure.

#1 Grammars that describe DNA strings

Proteins are long sequences of aminoacids, so their primary structures can be represented as words on 20 symbol alphabet. In a later paper we will describe how grammars can give the secondary structure of proteins. An RNA molecule is a long sequence of ribonucleotides - Adenine, Cytocine, Guanine or Uracil - so its primary structure can be represented as a word on the alphabet { a, c, g, u}. A DNA molecule is one or two long sequences of deoxyribonucleotides - Adenine, Cytocine, Guanine or Thymine - so its primary structure can be represented as one or two words on the alphabet { a, c, g, t}. As shown in figure 1, one of the two words for double stranded DNA can be ignored, because it is uniquely determined by the other word and the pairing rules: $\underline{a}$ = t, $\underline{c}$ = g, $\underline{g}$ = c, $\underline{t}$ = a .

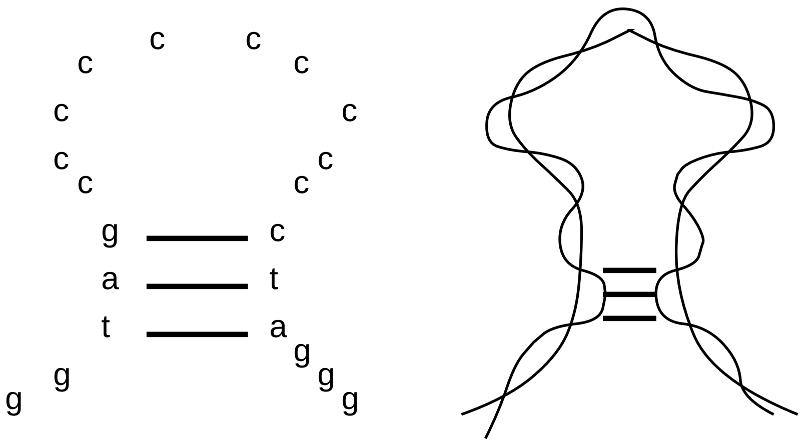


g g t a g c c c c c c c c c c c t a g g g

c c a t c g g g g g g g g g g g a t c c c

Fig.1 Twin Strand DNA molecule and its twin words

The secondary structure of RNA and DNA molecules arises from the fact that these pairing rules also apply to letters in their individual "primary structure words". As shown in figure 2, the rules give *crossdependencies* that indicate how the molecules fold in space.



c t a = $\underline{\text{g a t}}$

g g t a g c c c c c c c c c c c $\underline{\text{g a t}}$ g g g

g g t a g c c c c c c c c c c c c t a g g g

Fig.2 Crossdependencies and word duality

For spatial reasons "reversal" is involved in the definition of matching dual words. For both DNA and RNA the dual $\underline{\lambda}$ of the empty word λ is just the empty word λ. For DNA the dual word $\underline{W}$ is defined by:

$$\underline{a\,W} = \underline{W}\,t, \quad \underline{cW} = \underline{W}\,g, \quad \underline{g\,W} = \underline{W}\,c, \quad \underline{t\,W} = \underline{W}\,a$$

For RNA the dual word $\underline{W}$ is defined by:

$$\underline{a\,W} = \underline{W}\,u, \quad \underline{cW} = \underline{W}\,g, \quad \underline{g\,W} = \underline{W}\,c, \quad \underline{u\,W} = \underline{W}\,a$$

These dualities should be incorporated in any grammar for describing and analysing families of DNA and RNA molecules. Among such grammars we should mention stochastic context-free grammars [SBHMSUH94] and string grammars [Se93]. In the next paragraph we introduce our rival "pattern grammars".

One can define a pattern as a word W on terminal and nonterminal letters together with a partition of the occurrences of the nonterminals in W. When displaying patterns we display the partitition using the convention:

- λ is the empty pattern
- nonterminals with the same adornment are equivalent,
- unadorned nonterminals are only equivalent to themselves.

Thus a b S c b T a c S' b b T' is a pattern where the last two nonterminals are equivalent to each other and the first two are only equivalent to themselves. If W_1 (W_2 , W_3) are terminal words derivable from S (T, both S and T), then one can substitute in the pattern to get the terminal word: a b W_1 c b W_2 a c W_3 b b W_3

To get an RNA or DNA pattern we need only add: nonterminals may or may not be underlined; dual words are substituted for underlined patterns.

Def.1. A *DNA (RNA) pattern multigrammar* Γ consists of Δ, a nonterminal alphabet, and for each S in Δ a set Π_S of DNA(RNA) patterns over Δ and {a, c, g, t} ({a, c, g, u }).

Def.2. A *DNA (RNA) pattern language* L is generated by a DNA (RNA) pattern multigrammar Γ if L= L_S for some S in Δ where L_S is the S-component of the least fixed point of the language function Φ defined by

W in $\Phi_S[....L_\delta...]$ iff

$W = u_0\ v_1\ u_1\ v_2\ u_2 .....v_n\ u_n$

for some pattern $u_0\ \delta_1\ u_1\ \delta_2\ u_2 .....\delta_n\ u_n$ in Π_S and terminal words $u_0, u_1, u_2 ..... u_n, v_1, v_2 .... v_n$ such that v_i in $L_{\delta\ i}$ and

- δ_i eq δ_j implies $v_i = v_j$
- $\underline{\delta_i}$ eq δ_j implies $\underline{v_i} = v_j$
- $\underline{\delta_i}$ eq $\underline{\delta_j}$ implies $\underline{v_i} = \underline{v_j}$.

Here ....L_δ.. ...represents a tuple of pattern sets L_δ one for each δ in Δ.

Conventions: Any word can be substituted for an underlined space in a pattern. The ornament & indicates that any word may be substituted but the same word must be substituted at other occurrences of & and the dual of the word must be substituted at other occurrences of <u>&</u> . Both these conventions are used in the one pattern DNAparsing multigrammar in figure 3.

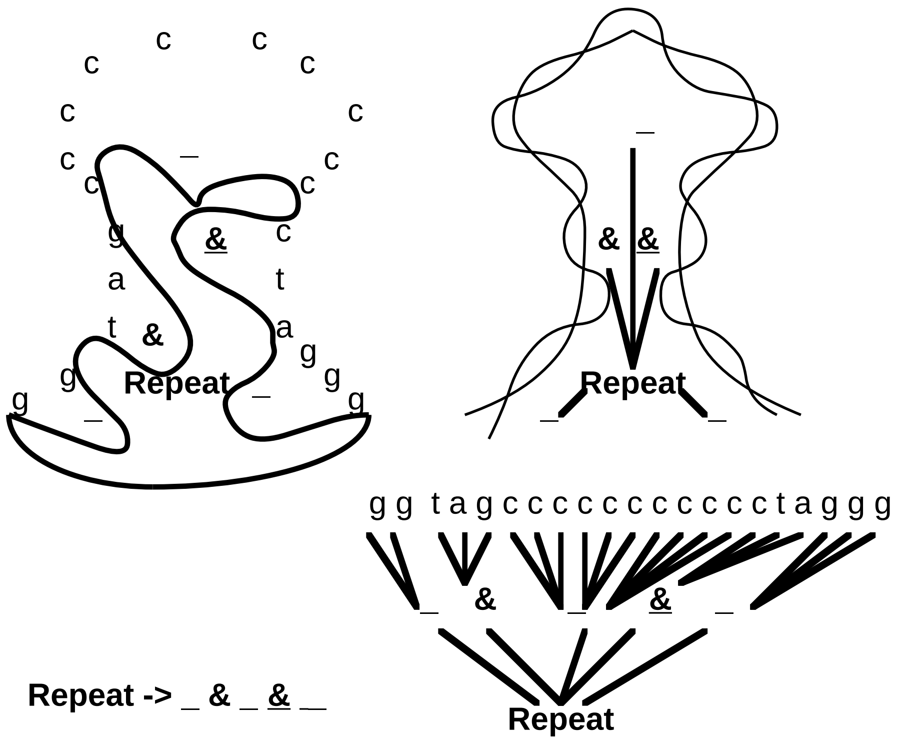


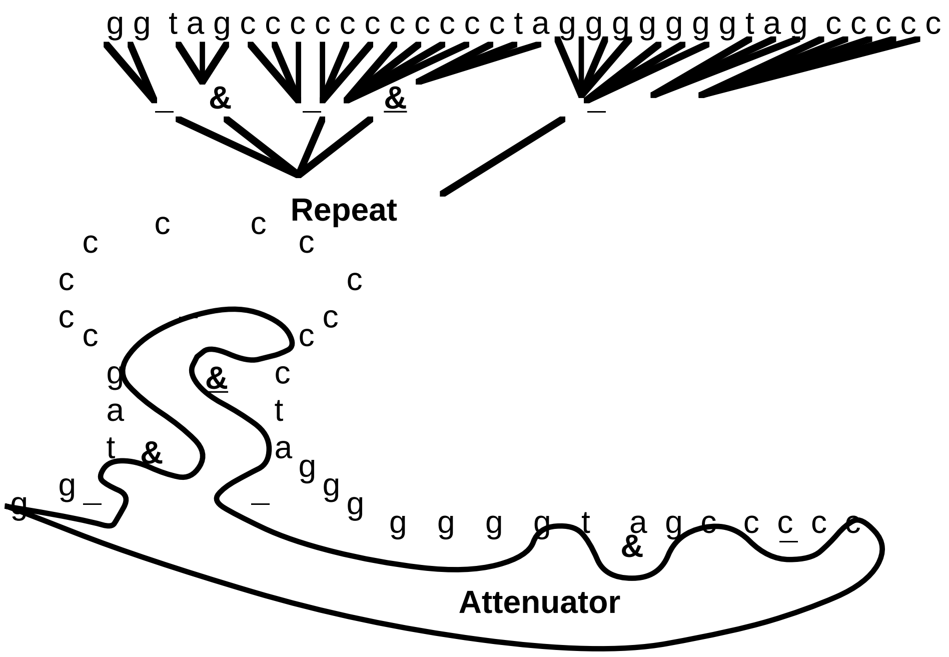
Fig.3 DNA Parsing with a one pattern grammar

#2 Grammars describing DNA secondary structure.

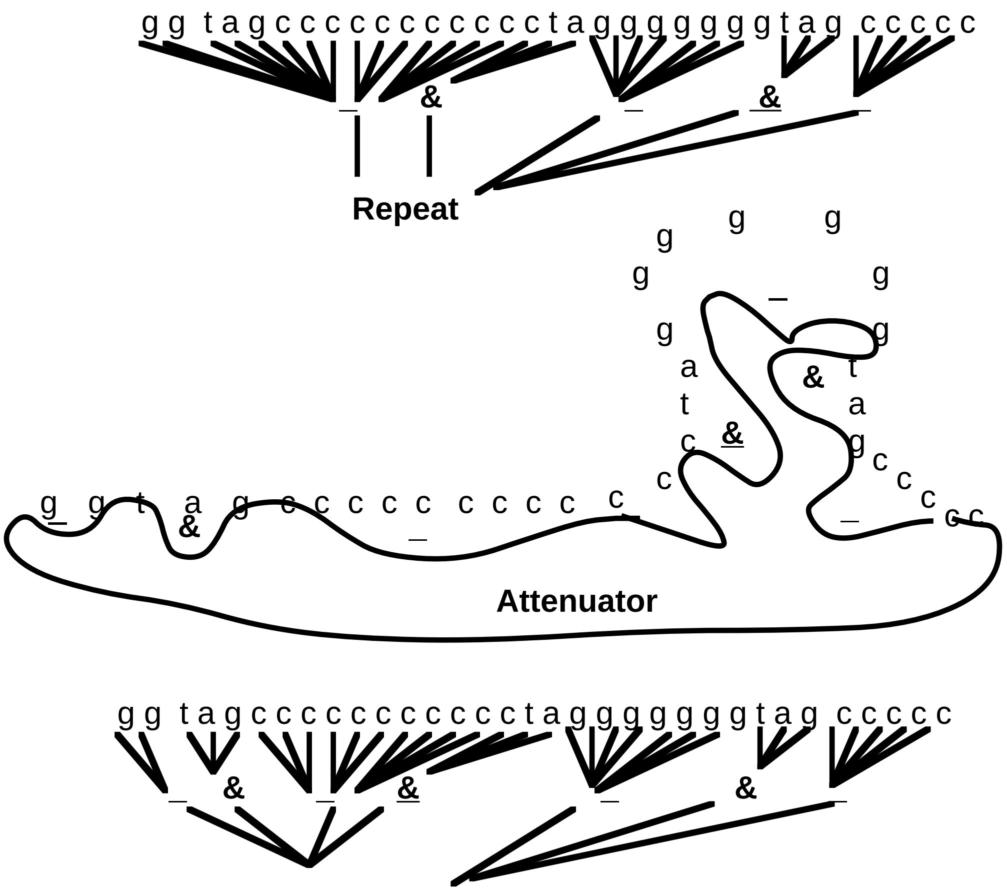
Every cell in a multicellular organism like us has identical DNA molecules, but most cells are very different; hair is very unlike skin, muscle and brain cells. The reason for these differences is that in each cell some of the *genes* encoded in the DNA are active and some are passive. Sometimes this genetic switching between active and passive genes corresponds to different secondary structures of the DNA. As grammars can be ambiguous, allow for alternative derivations of the same word, they can neatly capture alternative secondary structures of DNA and RNA molecules. A simple example of this is given by the two pattern grammar:

InvertedRepeat	-> _ & _ <u>&</u> _
Attenuator	-> _ & _ <u>&</u> _ & _

Clearly any word that can be derived from the attenuator pattern can also be derived from the repeat pattern in two different ways. Figure 4 gives an example of this simple example of possible genetic switching.



g g t a g c c c c c c c c c c c t a g g g g g g g t a g c c c c c
_ & _ & _
Repeat
Attenuator



g g t a g c c c c c c c c c c c c t a g g g g g g g t a g c c c c c
&
&
Repeat
g g t a g c c c c c c c c c c c c t a g g g g g g g t a g c c c c c
&
&
Attenuator
g g t a g c c c c c c c c c c c c t a g g g g g g g t a g c c c c c
&
&
&

Fig.4 Genetic Switching

Another common secondary structure is the PseudoKnot; it is prominent in the RNA pattern multigrammar for tobacco mosaic virus RNA:

PseudoKnot -> & _ ¨& _ & _ ¨&

tmv_3prime -> PseudoKnot PseudoKnot PseudoKnot _ & _ ¨& &'

_ InvertedRepeat &' ¨& _ & InvertedRepeat PseudoKnot _

InvertedRepeat -> & _ &

Notice that one has the alternative derivation of
" _ & _ ¨& &' _ InvertedRepeat &' ¨& _ & " from the Pseudoknot pattern. Figure 5 shows the secondary structure of a simple PseudoKnot and the more intricate tmvRNA.

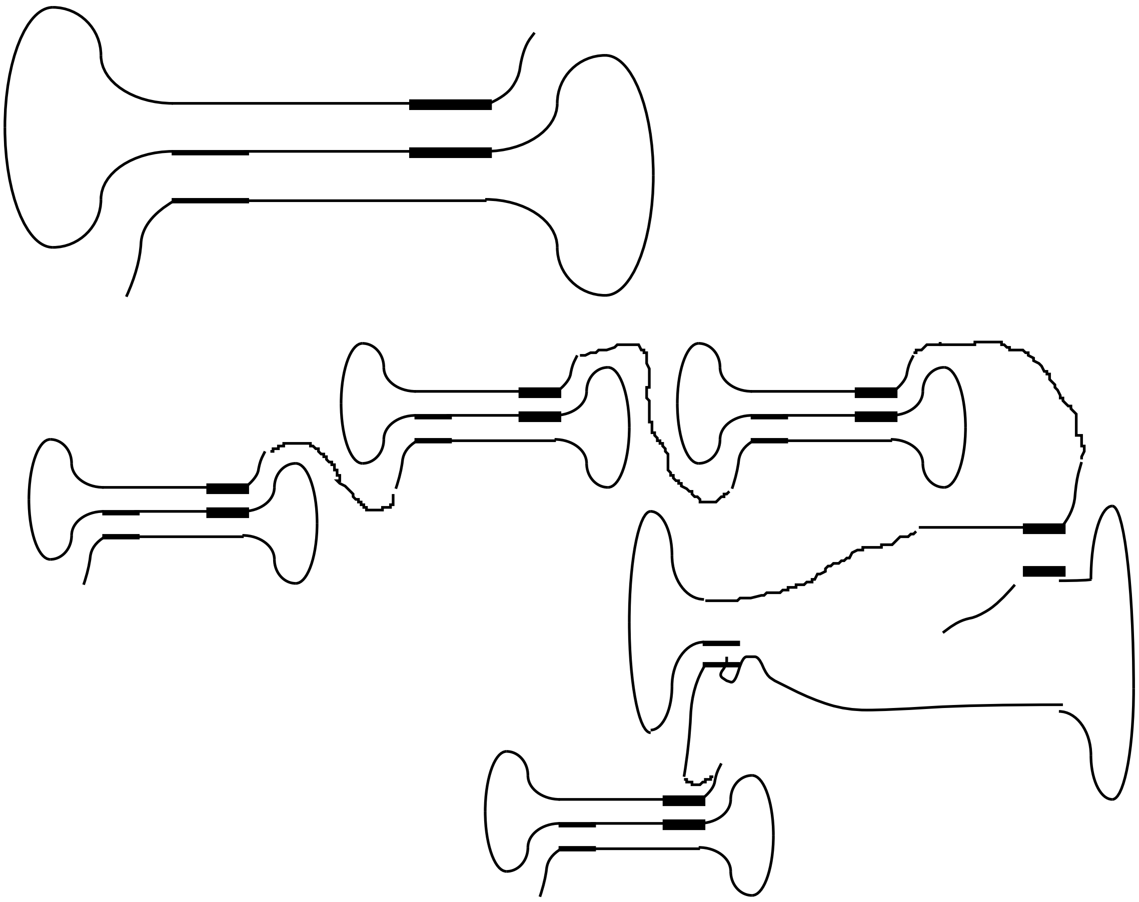


Fig.5 Pseudoknots and Tobacco Mosaic Virus RNA (177 nucleotides)

The string grammar for transfer RNA [Se 93p92] begins with:
tRNA(Codon) ---> AcceptorArm, _, DArm, _, ~DArm, _, AnticodonArm, _ ,~AnticodonArm, _, TpsiCArm, _, ~TpsiCarm, _, ~AcceptorArm, _.

The RNA pattern multigrammar for transfer RNA begins with:

tRNA -> & CloverLeaf <u>&</u>

CloverLeaf -> _ Darm _ AntiCodonArm _ TψCarm _

Figure 6 gives the full multigrammar and the derivation for tRNA; figure 7 shows that biological reality is more complex than the simple examples in this paper.

tRNA -> & CloverLeaf <u>&</u>

CloverLeaf -> _ Darm _ AntiCodonArm _ TψCarm _

Darm -> & _ <u>&</u>

AntiCodonArm -> & _ Codon _ <u>&</u>

TψCarm -> & _ <u>&</u>

Codon -> Nucleotide Nucleotide Nucleotide

Nucleotide -> g | c | a | u

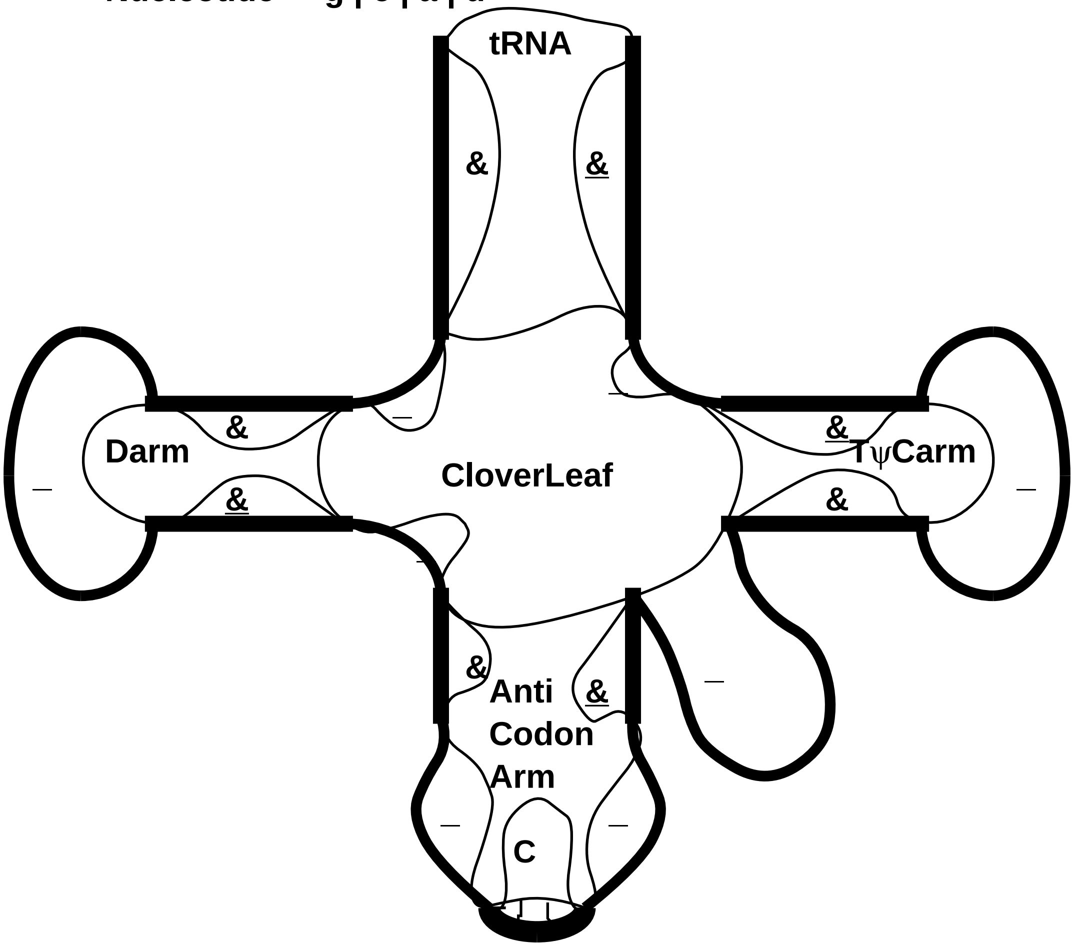


Fig. 6 grammatical derivation of transfer RNA

Fig. 7 Real transfer RNA
a) spatially realistic secondary structure
b)corrected cloverleaf secondary structure
c) tertiary structure

#3 Genetic variation: alleles and viri.

One can consider DNA or RNA strings of length N as points in a *sequence space* of dimension N. As there are only 4^N such points in this space, its topology is very different from our familiar Euclidean space; in particular the largest possible distance (= number of mutations) between two points is N, and there are many "shortest" paths between points, not just one "straight line". Any DNA or RNA pattern language corresponds to a set of points in this strange sequence space; the DNA or RNA strings in any population correspond to a multiset of points in this strange sequence space, as the same string may occur many times in the population. Consider the population of people now living in Denmark. The corresponding (multi)set of sequence space points is included in the (multi)set for the people who have ever lived in Denmark, which is included in the (multi)set for the people who will ever live in Denmark. This is because there is genetic variation in people, genes have alleles. Traditionally one assumes that the genetic variation of a species is given by a sequence space point for the socalled "wild type" and a cluster of nearby points, but this will only be true if "nearby" does not mean "few mutations away from". There is much evidence that the traditional view is totally inadequate for influenza, HIV and other viri, because they have vast populations and high mutation rates [Ei92]. Eigen has introduced the term *quasispecies* for populations of viri and other organisms with very dynamic genetic variation. In sequence space quasispecies will give cloudy, fractal multisets; species will give clean, compact multisets; nevertheless both should be amenable to description by DNA and RNA pattern multigrammars.

Fig.8 Population dynamics

a) *perfect replication of wild type*
b) *highly imperfect replication*
c) *perfect replication with alleles*
d) *quasispecies*

Viri seem to be a promising area for grammatical approaches to DNA and RNA analysis. The chemical and biological details of genetic switching, we described earlier, were first worked out for the bacteriophage λ (a virus that preys on bacteria) [Pt86]. The phage λ switches between two forms: a virulent form, in which bursts the bacteria, and a dormant form where it reproduces when the bacteria cell divides. The two forms differ in which genes are expressed and which are repressed. It is not difficult to devise a pattern multigrammar for phage λ in which the two forms correspond to alternative derivations of its RNA sequence:

phage -> early1	cloff	early2	headon tailon lysison
phage -> early1	clon	early2	headoff tailoff lysisoff
early1 -> recombination		C111	N
early2 -> cro	C11	replication	Q

and patterns for cro and the other genes - cloff (headon, tailon, lysison) has the same pattern as clon (headoff, tailoff, lysisoff)

phage -> early1 cloff early2
headon tailon lysison
phage -> early1 clon early2
headoff tailoff lysisoff

Fig.9 Genetic Switching
in the phage λ

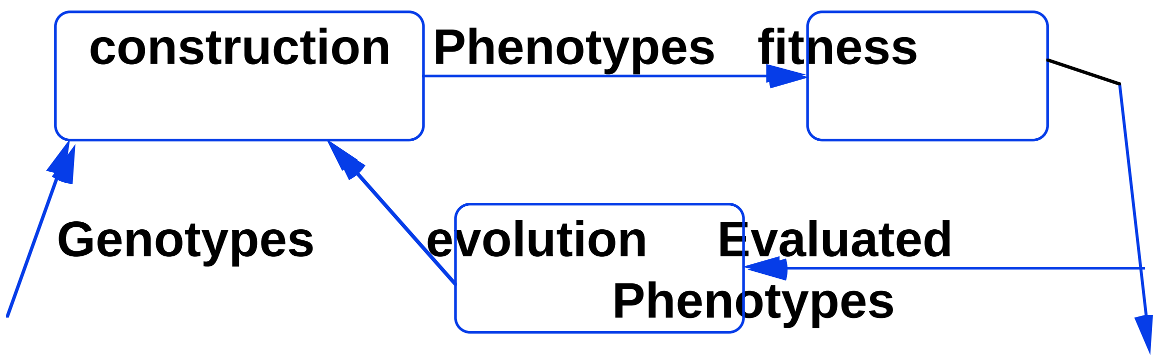
#4 Machine learning of DNA grammars.

Where do grammars come from? There are many machine methods for inventing grammars that fit positive and negative examples of the "language" to be described by the grammar [OP91,CO94]. Some of these methods allow expert knowledge to preadapt the grammar search space. Among the methods with this advantage, the one that seems most appropriate to the hunt for RNA and DNA grammars seems to be *genetic algorithms* , even although it is not mentioned in the latest survey of machine learning applied to gene recognition [CS94]. Genetic algorithms can be used if one can measure the *fitness* of a candidate RNA/DNA grammar. The discussion of sequence spaces and quasispecies in the last section suggests a suitable fitness measure: how well does the language generated by the grammar include the multiset of positive examples and exclude the multiset of negative examples.

The basic idea of genetic algorithms is shown in figure 10: a population of phenotypes is constructed from a population of genotypes, the fitness of the phenotypes is evaluated, their fitness influences the evolution of a new genotype population and a new "life-cycle" starts. Figure 10 also shows a version of the usual evolutionary operators, crossover and mutation, that is appropriate if we make the reasonable assumption:

Genotype = Phenotype = DNA/RNA pattern multigrammar

so the construction phase is trivial.



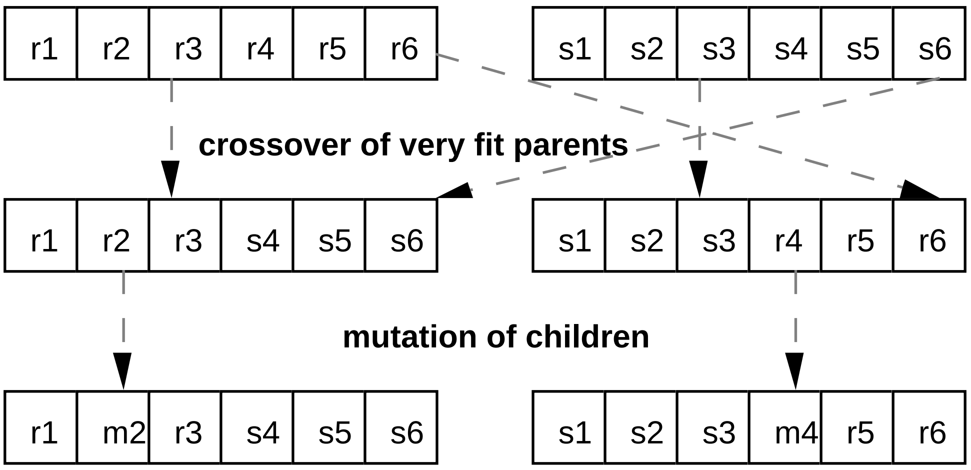


Fig 10 Genetic algorithms & grammar evolutionary operators

Any attempt to learn DNA/RNA pattern multigrammars by genetic programming will only be successful if wise decisions have been made on various crucial questions.

- *genotype representation* Do we want PRE,SUF,LIG operations on grammars [Se93], or do sorted lists of patterns suffice?
- *crossovers* How do we ensure that useful structural building blocks are not destroyed? Should we insist that all grammars contain such common building blocks as: internal repeats, Pseudoknots etc.?
- *mutations* Do we need more than changing the underlining and ornamentation (equivalence relation) of patterns?
- *fitness* Clearly recognition of positive (negative) examples should be rewarded (punished), but should precise and detailed grammars be rewarded (remember the alternative derivation in the tobacco virus grammar) ?
- *examples* Wise choice of negative examples is essential, but should

all positive examples be used? Coevolution as in [Hi90] but using sampling and spatial rotations to get new positive and negative examples.

#5 Conclusion.

DNA and RNA pattern multigrammars are expressive and learnable; they may be useful in analysing DNA and RNA molecules.

References

[CO94] R.C.Carrasco, J.Oncina (eds.) "Grammatical inference and applications" Springer LNAI 862(1994)

[CS94] M.W.Craven, J.W.Shavlik "Machine learning approaches to gene recognition" IEEE Expert???(1994)2-10

[Ei92] M.Eigen "Steps toward life", Oxford U.P.1992

[Ei93] M.Eigen "Viral quasispecies", Sci.Amer.1993 july 32-40

[Hi90] W.D.Hillis "Co-evolving parasites improve simulated evolution as an optimisation procedure" in S.Forrest(Ed.) "Emergent computation" pp228-234, North Holland 1990.

[KSQMSWSR74] S.H.Kim, F.L.Suddath, F.L.Quigley, A.McPherson, J.L.Sussman, A.H.J.Wang, N.C.Seegman,A.Rich "Three-dimensional tertiary structure of Yeast phenylalanine transfer RNA", Science 185(1974) 435-439

[OP91] G.C.Overton, J.A.Pastor "A platform for applying multiple machine learning strategies to the task of understanding gene structure" IEEE???(1991) 450 -457

[Pt86] M.Ptashne "A Genetic Switch: gene control and phage λ"
Cell Press and Blackwell Scientific Publications, 1986.

[SBHMSUH94] Y.Sakakibara, M.Brown, R.Hughey, I.S.Mian, K.Sjolander, R.C.Underwood, D.Haussler "Recent methods for RNA modelling using stochastic context-free grammars" Springer LNCS 807(1994) 289-306

[Se93] D.B.Searls "The computational linguistics of biological sequences" in *Artificial Intelligence and Molecular Biology* chapter 2, pages 47-120. AAAI Press 1993.

[Wa77] J.D.Watson "Molecular biology of the gene", W.A.Benjamin, Inc.1977