GENERATING MARKED COMBSTRUCT GRAMMARS

An introduction and some applications

Marni Mishna July 1997

A combstruct grammar *G* can be used to generate objects, or can itself be manipulated to form related grammars. For example, one can associate a marked grammar *M(G)* with *G*. A marked grammar generates structures with a fixed number of tags affixed to a specified object. For example, one could tag cycles in a permutation, the substring of a regular expression or the components of a forest. *M(G)* could then generate the permutations with one cycle tagged, one occurence of the substring marked or two components in a forest tagged. The tags have the form of an Epsilon product. The relationship between the two grammars becomes evident when considering the generating functions. Marked grammars can be used to solve problems with numbers of components of a certain type (such as average number of cycles in a permutation), moments, or more complicated combinatorial parameters such as pathlength.

```
> with(combstruct);
```

[*allstructs*, *count*, *draw*, *finished*, *gfeqns*, *gfseries*, *gfsolve*, *iterstructs*, *mark*, *moment*, *nextstruct*, *prog_gfeqns*, *prog_gfseries*, *prog_gfsolve*, *variance*]

Marked Grammars

A marked structure from *M(G)* is equivalent to its counterpart in *G* with a certain number of substructures, or objects distinguished. The objects are distinguished with marks-- a product with some **Epsilon**. If for example, *G* describes binary trees, *G* could be marked in such a way that *M(G)* generates binary trees with a fixed number of atoms marked (nodes colored). Marked grammars are valid grammars consisting of productions made from **Union**, **Prod**, **Set**, **Sequence**, and **Cycle**. A particular non-terminal from the original grammar is chosen, to be the recipient of marks. The corresponding marked production may not be the same kind of production, though they do maintain a bijection. The marks are unordered, and if Z is the object we are marking we may only do so once for any given *Z*.

Combstruct generates marked grammars via the command combstruct[mark] . Given the marked grammar we can draw examples from the new grammar.
The production A_1 generates objects like *A* with one node marked. In general, the marked production A_k generates objects with *k* marks.

Example: Binary Trees

The grammar mbintree generates trees with two coloured nodes. It also contains the productions for trees with one and zero coloured nodes.

```
> bintree:= {A=Union(Z, T), T=Prod(Z, A, A)}:
  mbintree:=mark(bintree, unlabelled, [Z, tag], 2);
```

$mbintree := \{A_0 = \mathrm{Union}(Z_0, T_0),\ combstruct/T0_0 = \mathrm{Prod}(A_0, A_0),$

$T_0 = \text{Prod}(Z_0, \textit{combstruct/T0}_0)$, $tag = \text{E}$, $Z_1 = \text{Prod}(Z_0, tag)$, $Z_2 = \text{Prod}(Z_0, tag)$,

$A_1 = \text{Union}(Z_1, T_1)$, $A_2 = T_2$, $\textit{combstruct/T0}_1 = \text{Union}(\text{Prod}(A_1, A_0), \text{Prod}(A_0, A_1))$,

$\textit{combstruct/T0}_2 = \text{Union}(\text{Prod}(A_2, A_0), \text{Prod}(A_1, A_1), \text{Prod}(A_0, A_2))$,

$T_1 = \text{Union}(\text{Prod}(Z_1, \textit{combstruct/T0}_0), \text{Prod}(Z_0, \textit{combstruct/T0}_1))$,

$T_2 = \text{Union}(\text{Prod}(Z_1, \textit{combstruct/T0}_1), \text{Prod}(Z_0, \textit{combstruct/T0}_2))$, $Z_0 = Atom\}$

```
> draw([A[1], mbintree, unlabelled], size=5);
```

$$\text{Prod}(\text{Prod}(Z_0, tag), \text{Prod}(\text{Prod}(Z_0, \text{Prod}(Z_0, Z_0)), Z_0))$$

Thus A_2 has two nodes marked.

```
> draw([A[2], mbintree, unlabelled], size=5);
```

$$\text{Prod}(Z_0, \text{Prod}(\text{Prod}(Z_0, \text{Prod}(\text{Prod}(Z_0, tag), \text{Prod}(Z_0, tag))), Z_0))$$

There is a mapping from the marked objects to the grammar. Hence, it is possible that the structures produced do not directly resemble objects from the original grammar. This is even the case for the 0-marked objects.

```
> draw([A[0], mbintree, unlabelled], size=5);
> draw([A, bintree, unlabelled], size=5);
```

$$\text{Prod}(Z_0, \text{Prod}(\text{Prod}(Z_0, \text{Prod}(Z_0, Z_0)), Z_0))$$

$$\text{Prod}(Z, \text{Prod}(Z, Z, Z), Z)$$

The number of different trees of size five with one node marked is the number of trees of size five times five, the number of ways to mark the nodes of a tree of five nodes.

```
> count([A, bintree, unlabelled], size =5);
> count([A[1], mbintree, unlabelled], size=5);
```

$$2$$

$$10$$

We can choose type labelled as well. The labels are put on after the marks.

```
> mbintree:=mark(bintree, labelled, [Z, tag], 2):
> draw([A[1], mbintree, labelled], size=5);
```

$$\text{Prod}({Z_0}_4, \text{Prod}({Z_0}_3, \text{Prod}({Z_0}_5, \text{Prod}({Z_0}_2, \text{Prod}({Z_0}_1, tag)))))$$

We can just as easily mark only the leaves in the binary tree. We must redescribe the grammar as to make it clear what the leaves are.

```
> BTree:= {T=Union(Leaf, T2), T2=Prod(Node, T, T),
  Node=Atom, Leaf=Epsilon}:
> L_Btree:=mark(BTree, unlabelled, [Leaf, L], 1):
> count([T[1], L_Btree, unlabelled], size=5)/count([T[0],
  L_Btree, unlabelled], size=5);
```

$$6$$

The average number of leaves in a binary tree with five nodes is six. Although in the case of trees this number is constant, this computation shows how marking can be applied to the computation of averages of combinatorial parameters.

Applications

Generating Moments

Generating functions are very useful tools for finding means, standard deviations and other moments of distribution. Given a structure *A* and the substructure of interest marked with a *u* , consider the generating function *A(x,u)*. The generating function for the mean is

$$\frac{[x^n]\left(\frac{\partial}{\partial u}\mathrm{A}(x,u)\right)_{u=1}}{[x^n]\,\mathrm{A}(x,1)} = \sum a_{n,k}\,k\,x^n$$ where $a_{n,k}$ is the number of objects of size *n* with *k* occurrences of the subobject. It is possible to extract this quantity from the univariate generating function for A_1 . The generating function $A_2(x)$ yields information about the variance and in fact the k^{th} moment can be determined from the generating function of A_k . Determining moments in this way is useful when the bivariate generating function cannot be found in its explicit form.

Set Partitions

We can compare the result using marks with the result obtained by working with the original generating function.
Consider the following grammar defining Set partitions.

```
> set_part:={S=Set(Prod(part,Set(Z,card>=1))),part=Epsilo
  n}, labelled:
  m_set_part:=mark(set_part,[part,u],2):
> set_part_gf:=gfsolve(set_part, z, [[u,part]]):
> m_set_part_gf:=gfsolve(m_set_part,labelled,z):
> s[0]:= subs(set_part_gf,S(z,u));
  s[1]:=subs(m_set_part_gf,S[1](z));
  s[2]:= subs(m_set_part_gf,S[2](z));
```

$$s_0 := \mathbf{e}^{(u\,(\mathbf{e}^z-1))}$$

$$s_1 := \mathbf{e}^{(\mathbf{e}^z-1)}\,\mathbf{e}^z - \mathbf{e}^{(\mathbf{e}^z-1)}$$

$$s_2 := \frac{1}{2}\,\mathbf{e}^{(\mathbf{e}^z-1)}\,(\mathbf{e}^z)^2 - \mathbf{e}^{(\mathbf{e}^z-1)}\,\mathbf{e}^z + \frac{1}{2}\,\mathbf{e}^{(\mathbf{e}^z-1)}$$

The first moment is the derivative of *S(u,z)* with repect to *u*, evaluated at *u*=1. This is the same as our $S_1(z)$.

```
> expand(subs(u=1,diff(s[0],u))-s[1]);
```

$$0$$

```
> expand(subs(u=1, 1/2*diff(s[0], u, u))-s[2]);
```

$$0$$

The mean number of external nodes / leaves in a binary tree

We return to binary trees to show how we can use marking to determine information about the number of substructures in a structure. The first example uses a marked grammar to illustrate the plain fact that the number of external nodes in a binary tree of size *2n+1* is *n+1*. This is followed by applying the same technique to determine the average number of internal nodes in a binary tree of size *n* with both children external. (Such nodes are often called *leaves*)

To count the number of external nodes we generate the grammar that has two external nodes marked. With the generating functions we can compute the moments and determine the average number of external nodes (which should be almost a half) and the variance (which should be zero).

```
> bintree:={T=Union(XN, ST), ST=Prod(N, T, T), XN=Atom,
  N=Atom}:
  mbintree:=mark(bintree, unlabelled, [XN, tree], 2):
  gf_bintree:=gfsolve(mbintree,unlabelled,z):
```

$T_1(z)$ gives us the first moment with respect to that which we have marked, the leaves. Thus to find the average number of leaves in an object of size n, we divide the coefficient of z^n in $T_1(z)$ by the corresponding coefficient in $T_0(z)$.

```
> T0:=subs(gf_bintree,T[0](z)); T1:=
  subs(gf_bintree,T[1](z));
  T2:=factor(subs(gf_bintree,T[2](z)));
```

$$T0 := \frac{1}{2}\,\frac{1-\sqrt{1-4\,z^2}}{z}$$

$$T1 := \frac{z}{\sqrt{1-4\,z^2}}$$

$$T2 := -\frac{z^3}{\sqrt{-(2\,z-1)\,(2\,z+1)}\;(2\,z-1)\,(2\,z+1)}$$

```
> coeff(series(T1, z, 130), z^121)/coeff(series(T0, z,
  130), z^121)/121.;
```

$$.5041322314$$

```
> 61/121.;
```

$$.5041322314$$

Since it is the same for each tree the variance should be 0. We can compute it using the twice marked production.

```
> t0seq:=series(T0, z, 20):
  t1seq:=series(T1, z, 20):
  E2seq:=series(2*T2+T1, z, 20):
  seq(coeff(E2seq, z^i)/coeff(t0seq, z^i)-(coeff(t1seq,
  z^i)/coeff(t0seq, z^i))^2, i=[1,3,5,7,9,11,13,15,17]);
```

Our results are consistent with common knowledge.

$$0, 0, 0, 0, 0, 0, 0, 0, 0$$

Now we try the same technique to determine the average number of internal nodes such that both nodes are external. We will consider the trees with external nodes removed. The leaves on the pared trees correspond to the internal nodes with both nodes external nodes.

```
> pared_bintree:= {T=Union(L, ST), ST=Union(Prod(N, T,
  T), Prod(N, Epsilon, T), Prod(N, T, Epsilon)),L=Atom,
  N=Atom}:
  mpbintree:=mark(pared_bintree, unlabelled, [L, tree],
  2):
```

```
gf_pbintree:=gfsolve(mpbintree,unlabelled,z):
```

$T_1(z)$ gives us the first moment with respect to that which we have marked, the leaves. Thus to find the average number of leaves in an object of size n, we divide the coefficient of z^n in $T_1(z)$ by the corresponding coefficient in $T_0(z)$, as before

```
> T[0]:=subs(gf_pbintree,T[0](z));
  T[1]:=factor(subs(gf_pbintree,T[1](z)));
  T[2]:=factor(subs(gf_pbintree,T[2](z)));
```

$$T_0 := \frac{1}{2}\frac{-2\,z+1-\sqrt{1-4\,z}}{z}$$

$$T_1 := \frac{z}{\sqrt{1-4\,z}}$$

$$T_2 := -\frac{z^3}{\sqrt{1-4\,z}\,(4\,z-1)}$$

We can now determine closed forms for the average and the variance.

```
> with(gfun):
> TT[0]:=rsolve(diffeqtorec(holexprtodiffeq(T[0], y(z)),
  y(z), u(n))minus {u(0)=0}, u(n));
  TT[1]:=rsolve(diffeqtorec(holexprtodiffeq(T[1], y(z)),
  y(z), u(n)) minus {u(1)=0, u(0)=0}, u(n));
  TT[2]:=rsolve(diffeqtorec(holexprtodiffeq(T[2], y(z)),
  y(z), u(n)) minus {u(2)=0, u(1) = 0, u(0) = 0}, u(n));
```

$$TT_0 := \frac{4^n\,\Gamma\left(n+\frac{1}{2}\right)}{\Gamma(2+n)\sqrt{\pi}}$$

$$TT_1 := \frac{4^{(n-1)}\,\Gamma\left(n-\frac{1}{2}\right)}{\Gamma(n)\sqrt{\pi}}$$

$$TT_2 := \frac{1}{32}\frac{4^n\,\Gamma\left(n-\frac{3}{2}\right)}{\Gamma(-2+n)\sqrt{\pi}}$$

```
> simplify(TT[1]/TT[0], power, GAMMA);
```

$$\frac{1}{2}\frac{n\,(n+1)}{2\,n-1}$$

```
> asympt("/n, n);
```

A quarter of the tree is leaves, as expected.

$$\frac{1}{4}+\frac{3}{8}\frac{1}{n}+\frac{3}{16}\frac{1}{n^2}+\frac{3}{32}\frac{1}{n^3}+\frac{3}{64}\frac{1}{n^4}+\frac{3}{128}\frac{1}{n^5}+\mathrm{O}\left(\frac{1}{n^6}\right)$$

```
> simplify((2*TT[2]+TT[1])/TT[0]-(TT[1]/TT[0])^2,
```

```
symbolic);
```

$$\frac{1}{2}\,\frac{n\,(n+1)\,(n^2-3\,n+2)}{(2\,n-1)^2\,(2\,n-3)}$$

```
> asympt("/n, n);
```

The normalized variance is tending to a constant. This is expected.

$$\frac{1}{16}+\frac{1}{32}\,\frac{1}{n}-\frac{3}{32}\,\frac{1}{n^2}-\frac{9}{64}\,\frac{1}{n^3}+\mathrm{O}\left(\frac{1}{n^4}\right)$$

The mean number of cycles in a random permutation

We can use the same method to determine information about substructures other than atoms. For example, we could count the average number of cycles in a random permutation. Just as easily we can also compute the variance.

```
> Perm:= {P=Set(C), C=Cycle(Z, card>0)}:
> M_Perm:=mark(Perm, labelled, [C, m], 2):
  M_Perm_gf:= gfsolve(M_Perm, labelled, z):
> p0:=subs(M_Perm_gf, P[0](z));p1:=subs(M_Perm_gf,
  P[1](z));p2:=subs(M_Perm_gf, P[2](z));
```

$$p0 := -\frac{1}{-1+z}$$

$$p1 := -\frac{\ln\left(-\frac{1}{-1+z}\right)}{-1+z}$$

$$p2 := -\frac{1}{2}\,\frac{\ln\left(-\frac{1}{-1+z}\right)^2}{-1+z}$$

Next, we can try to solve for the coefficients in the generating function. We can see from the closed form that the coefficients of p0 are all 1.

```
> PP0:=rsolve(diffeqtorec(holexprtodiffeq(p0, y(z)),
  y(z), u(n)), u(n));
> PP1:=rsolve(diffeqtorec(holexprtodiffeq(p1, y(z)),
  y(z), u(n)), u(n));
```

$$PP0 := \{\,\mathrm{u}(n)=1,\ \mathrm{u}(0)=1\,\}$$

$$PP1 := \Psi(n+1)+\gamma$$

Thus, a permutation of n has on average $\Psi(n+1)+\gamma$ cycles. We can now try to determine the variance.

```
> PP2:=rsolve(diffeqtorec(holexprtodiffeq(p2, y(z)),
  y(z), u(n)) minus{u(1)=0,u(0)=0}, u(n));
```

$$PP2 := \mathrm{rsolve}\left(\left\{\,\mathrm{u}(2)=\frac{1}{2},\right.\right.$$
$$(-1-2\,n-n^2)\,\mathrm{u}(n)+(5\,n+3+2\,n^2)\,\mathrm{u}(n+1)+(-3\,n-n^2-2)\,\mathrm{u}(2+n)+1\,\},$$
$$\left.\mathrm{u}(n)\right)$$

We cannot solve PP2 explicitly, so we solve for particular values of n, say 1..10.

```
> p0seq:=series(p0, z, 20):p1seq:=series(p1, z,
  20):E2seq:=series(2*p2+p1, z, 20):
  seq(eval(coeff(E2seq, z^i)/coeff(p0seq,
  z^i)-(coeff(p1seq, z^i)/coeff(p0seq, z^i))^2),
  i=1..10);
```

$$0, \frac{1}{4}, \frac{17}{36}, \frac{95}{144}, \frac{2951}{3600}, \frac{3451}{3600}, \frac{190699}{176400}, \frac{839971}{705600}, \frac{8186939}{6350400}, \frac{350339}{254016}$$

Alternately we can use the variance command to get these values :

```
> seq(combstruct[variance]([P, Perm, labelled], size =i,
  C), i=1..10);
```

$$0, \frac{1}{4}, \frac{17}{36}, \frac{95}{144}, \frac{2951}{3600}, \frac{3451}{3600}, \frac{190699}{176400}, \frac{839971}{705600}, \frac{8186939}{6350400}, \frac{350339}{254016}$$

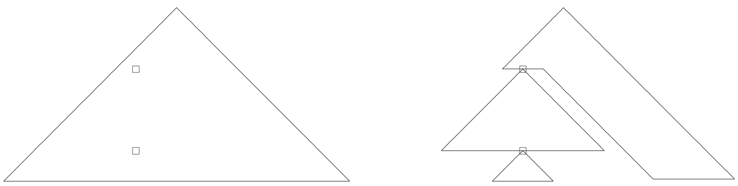
Path Length

Creating the bijection

The internal pathlength of a tree is the sum of the distances from each node to the root node. When the nodes of the tree are used to store data, the average pathlength of that type of tree is clearly related to the average complexity of data retrieval. We can use a marked grammar of a tree to determine the average pathlength of a family of trees. The distance from a node to the root is the number of predecessors of the node. Thus, we can count the total pathlength by counting the number of ways we can place two marks on the tree, one (call it *adult*) an ancestor of the other (call it *kid*).

We can describe such trees as the composition of three smaller trees
Tree1: The subtree rooted at *kid*
Tree2: The subtree rooted at *adult* with Tree1 removed
Tree3: The entire tree with Tree1 and Tree2 removed

Mapping of Tree with Two Colored nodes



Our leaves will be epsilons and thus of no size so we can superimpose the nodes corresponding to *kid* and *adult* over the marked leaves.
With marking we can easily generate all trees with one leaf marked

Binary Trees

```
> T:='T':
```

```
> BTree:= {T=Union(L, ST), ST=Prod(N, T, T), N=Atom,
  L=Epsilon}:
  BTree_mleaf:=mark(BTree, unlabelled, [L, m1], 1):
```

Next, we add our bijection. We must make one of the two trees with a marked leaf a tree with at least one node, else duplication occurs. Thus, our bijection is a tree of size at least one with one leaf marked, a tree with one leaf marked followed by a tree.

```
> BTree_pl:= BTree_mleaf union {PT= Prod(ST[1], T[1],
  T[0])}:
```

To count the total pathlengths of all binary trees with five nodes we count this new object.

```
> count([PT, BTree_pl, unlabelled], size = 3);
```

$$58$$

Since all of the generating functions are algebraic, using gfun we can determine a linear recurrence which is solvable with rsolve,

```
> BTree_gf:=gfsolve(BTree_pl, labelled, z):
> Bcount:=rsolve(diffeqtorec(holexprtodiffeq(subs(BTree_g
  f, PT(z)), y(z)), y(z), u(n)), u(n));
```

$$Bcount := 2\,4^n - 4\,\frac{4^n \left(\frac{1}{2}\right)^n 2^n\,\Gamma\left(\frac{3}{2}+n\right)}{\Gamma(2+n)\sqrt{\pi}}$$

```
> TreeCount:=rsolve(diffeqtorec(holexprtodiffeq(subs(BTre
  e_gf,T[0](z)), y(z)), y(z), u(n)), u(n));
```

$$TreeCount := \frac{4^n\,\Gamma\left(n+\frac{1}{2}\right)}{\Gamma(2+n)\sqrt{\pi}}$$

We can determine the general expression for pathlength of a tree of size n:

```
> simplify(Bcount, power, symbolic);
```

$$\frac{2\,4^n\,\Gamma(2+n)\sqrt{\pi} - 4^{(n+1)}\,\Gamma\left(\frac{3}{2}+n\right)}{\Gamma(2+n)\sqrt{\pi}}$$

We can determine the expression of average pathlength

```
> simplify(Bcount/TreeCount, power, symbolic);
```

$$2\,\frac{\Gamma(n)\,n^2\sqrt{\pi} + \Gamma(n)\,n\sqrt{\pi} - 2\,\Gamma\left(n+\frac{1}{2}\right)n - \Gamma\left(n+\frac{1}{2}\right)}{\Gamma\left(n+\frac{1}{2}\right)}$$

Asympt gives us information about the size of this asymptotically:

```
> map(combine, asympt(Bcount/TreeCount, n),exp);
```

$$2\,\frac{\sqrt{\pi}}{\left(\frac{1}{n}\right)^{3/2}} - 4\,n + \frac{9}{4}\,\frac{\sqrt{\pi}}{\sqrt{\frac{1}{n}}} - 2 + \frac{17}{64}\sqrt{\pi}\,\sqrt{\frac{1}{n}} + \frac{3}{512}\sqrt{\pi}\left(\frac{1}{n}\right)^{3/2} + O\left(\left(\frac{1}{n}\right)^{5/2}\right)$$

Ternary Trees

We can apply the same approach to other types of trees . One way to generalize binary trees is to consider trees with more than two children per node. Ternary trees have three children per node. We use the exact same procedure:

```
> TTree:= {T=Union(L, ST), ST=Prod(N, T, T, T),N=Atom,
  L=Epsilon }:
> TTree_mleaf:=mark(TTree, unlabelled, [L, ll], 1):
> TTree_pl:= TTree_mleaf union {PT= Prod(ST[1], T[1],
  T[0])}:
```

We can now find the average pathlength of ternary trees:

```
> count([PT, TTree_pl, unlabelled], size =2);
```

$$27$$

To make a comparison with binary trees we will consider two trees not of the same number of nodes, but with the same number of sum of nodes and leaves. A binary tree with n_1 nodes has n_1+1 leaves, for a total sum of $2n_1$+1. A ternary tree with n_2 nodes has $2n_2$+1 leaves, for a total sum of $3n_2$+1 . If we set $n_1 = 60$ and $n_2 = 40$ these sums are equal.

```
> evalf(count([PT, TTree_pl, unlabelled], size
  =40)/count([T[0], TTree_pl, unlabelled], size =40));
```

$$1041.333317$$

```
> evalf(count([PT, BTree_pl, unlabelled], size
  =60)/count([T[0], BTree_pl, unlabelled], size =60));
```

$$1436.476101$$

As one would expect the average path length for ternary trees is much shorter than for binary trees. Next we try to solve for a closed form for the average pathlength of ternary trees:

```
> TTree_gf:=gfsolve(TTree_pl, unlabelled, z):
> rsolve(diffeqtorec(holexprtodiffeq(subs(TTree_gf,
  PT(z)), y(z)), y(z), u(n)), u(n));
```

$$\begin{aligned}&\mathrm{rsolve}(\{\,\mathrm{u}(2)=3,\ \mathrm{u}(3)=33,\ \mathrm{u}(4)=282,\ \mathrm{u}(5)=2199,\ \mathrm{u}(6)=16392,\\&\mathrm{u}(7)=119061,\ \mathrm{u}(0)=0,\ \mathrm{u}(1)=0,\ (13122+59049\,n+59049\,n^2)\,\mathrm{u}(n)\\&\ +(-10044-36450\,n-18954\,n^2)\,\mathrm{u}(n+1)+(-5928+189\,n+999\,n^2)\,\mathrm{u}(n+2)\\&\ +(3000+1498\,n+184\,n^2)\,\mathrm{u}(n+3)+(-136\,n-288-16\,n^2)\,\mathrm{u}(n+4)\,\},\ \mathrm{u}(n))\end{aligned}$$

Unfortunately, the solver is unable to solve the expression.

```
>
```

References

Moments:

1. Sedgewick, R, Flajolet, P. *An Introduction to the Analysis of Algorithms,* Addison Wellsley 1996
2. Wilf, H. G*eneratingfunctionology,* Academic Press 1990