Philippe Flajolet and Analytic Combinatorics:
Tree Rewriting Systems

by Jean-Marc Steyaert
INRIA-team AMIB, Laboratoire d'Informatique
Ecole Polytechnique

Principle:

– Trees are recursive structures
– Algorithms on trees are naturally recursive
– Translate recursive definitions into equations over o.g.f.
– Conclude by means of analytic combinatorics

Bibliography

[15] PF: Analyse en moyenne de la détection des arbres partiels.(1978)

[28] PF,JMS: On the analysis of tree-matching algorithms.(1980)

[31] PF,JMS: A complexity calculus for classses of recursive programs over tree structures. (1981)

[43] PF,JMS: Patterns and pattern-matching in trees. (1983)

[66] PF,H.Prodinger: Level number sequences for trees.(1987)

[67] PF,JMS: A complexity calculus for recursive tree algorithms.(1987)

[89] PF,P.Sipala,JMS: Analytic variations on the dcommon subexpression problem.(1990)

[177] PF,B.Chauvin,D.Gardy,B.Gittenberger: And/Or Trees Revisited.(2004)

Basics on trees (cf. B. Sedgewick's talk)

Simple families of trees (Meir & Moon): T= $\sum_{\omega\in\Omega}\omega(\mathsf{T},...,\mathsf{T}) = \Omega(\mathsf{T})$, where Ω is a finite set of operators ω of various arities $\delta(\omega)$.

Let $\Phi(u)$ be the enumerative polynomial of Ω: $[u^n]\Phi(u) = \text{card}\{\omega \in \Omega | \delta(\omega) = n\}$; then the o.g.f. $t(z)$ of $\mathcal{T}$satisfies: $t(z) = z\Phi(t(z))$.

Theorem(M&M): Under very general conditions the number of trees of size n satisfies, with $\rho = 1/\Phi(\tau)$ and $\tau\Phi'(\tau) = \Phi(\tau)$:

$$t_n = [z^n]t(z) = \sqrt{\frac{\Phi(\tau)}{2\pi\Phi''(\tau)}}\rho^{-n}n^{-3/2}(1 + O(1/n)).$$

Proof: Track the first singular point on the real positive axis, which is a branching point of order 2. Then apply the Darboux method.

N.B.: All this process can be automated (cf. B. Salvy's talk)

PL-trees: Programming on trees

Procedures, Functions
Conditionals: *if* cond *then* ... *else* ...
Conditional iterations: *for* i=range *while* cond *do* ... *od*

Primitive operations on trees:
root(X) -> label of the root of tree X
deg(X) -> arity of the root of tree X
for $1 \leq i \leq$ deg(X), $X[i]$ -> i-th root subtree of X:
$X =$ root$(X)(X[1], ..., X[$deg$(X)])$
tests on label, arity

```
Example:
function equal(X,Y:T)
if root(X)<>root(Y) then assign(false) else assign(true);
for i:=1 to deg(X) while assign(equal(X[i],Y[i])) do nil od fi
```

EXAMPLE

PATTERN-MATCHING IN TREES

- Binary trees

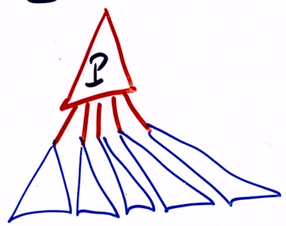
$$\mathcal{B} = \square + \underset{\mathcal{B}\ \ \mathcal{B}}{\overset{\circ}{\wedge}}$$

binary symbol (cons)

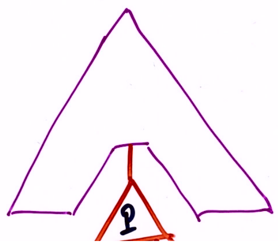
leaf (x)

- Pattern : A binary tree P 

- Root-occurrence : T is obtained from P by grafting binary trees to leaves of P

$T =$ 

- Occurrence

$T =$ 

- Algorithms

root? (P, T) =
  if P = x then true
  elseif T = x then false
  elseif root? (P_left, T_left)
    then root? (P_right, T_right) fi.

occ? (P, T) = root? (P, T) ;
  if P ≠ x then occ? (P, T_left)
    occ? (P, T_right) fi.

- Problem :

P being fixed, determine the average cost of running root? and occ? over all trees of size n.

From the program to its complexity

Idea: Associate to instructions, blocs, constructs, o.g.f. to capture the running time complexity!
Extension of combinatorial constructs (cf. B.Sedgewick)

$\tau A(X_1, X_2, ..., X_m)$: cost of running A on $X_1, X_2, ..., X_m$ of sizes $n_1, n_2, ..., n_m$
consider $\tau A(z_1, z_2, ..., z_m) = \sum_{|X_1|=n_1,...,|X_m|=n_m} \tau A(X_1, ..., X_m) z_1^{n_1} ... z_m^{n_m}$
which represents the cumulative costs of running A over all possible inputs, indexed by their sizes

In the same spirit for programs (functions) with boolean result, consider characteristic o.g.f.
$\chi A(z_1, z_2, ..., z_m) = \sum_{|X_1|=n_1,...,|X_m|=n_m} A_{true}(X_1, ..., X_m) z_1^{n_1} ... z_m^{n_m}$

Challenge: translate the constructors and basic operations
$A = \mathcal{C}(B_1, B_2, ..., B_k)$ −>
$\tau A = \Gamma(\tau B_1, \tau B_2, ..., \tau B_k, \chi B_1, \chi B_2, ..., \chi B_k)$

$A = B_1; B_2$ −> $\tau A(z) = \tau B_1(z) + \tau B_2(z)$

$A = B(X[i])$ −> $\tau A(z) = z\tau B(z)\Psi(t(z))$, with $\Psi(u) = \Phi(u)/u$ and variants in case of equality between subtrees...

$A = if\ Q\ then\ B\ else\ C$ −> $\tau A(z) = \tau Q(z) + \tau B(z|Q) + \tau C(z|\neg Q)$

$A = for\ i\ do\ B(X[i])\ od$ −> $\tau A(z) = z\tau B(z)\Phi'(t(z))$

copying a tree −> $\tau\text{copy}(z) = zt'(z)$

and also : conditional iteration, characteristic functions, etc.

- Generating series for costs can be computed recursively on the pattern structure (e.g. counting cost 1 for a generalized test)

$$\tau root[x](z) = B(z) = \sum_{t \in \mathcal{B}} 1 . z^{|t|}$$

$$\tau root[\circ(x,x)](z) = B(z) + z\, \tau root[x](z) . B(z) + z . B(z) . \tau root[x](z)$$

$$= 3B(z) - 2$$

$$\chi root[\circ(x,x)](z) = z B^2(z) = B(z) - 1$$

$$\tau root[\circ(\circ,\cdot)](z) = B(z) + z\, \tau root[\circ](z) . B(z) + z\, \chi root[\circ](z) . \tau root[x](z)$$

$$= (5-3z) B - 4 \longrightarrow 4\tfrac{1}{4}$$

$$\tau root[\circ(\cdot,\circ)](z) = (5-2z) B - 4 \longrightarrow 4\tfrac{1}{2}$$

$$\tau root[\circ(\circ(\circ,\cdot),\cdot)](z) = (7-7z-z^2) B - 6 + 2z \longrightarrow 5\tfrac{3}{16}$$

Average # of comparisons

Complexity of iteration over subtrees

$A(X) = B(X);\ for\ i = 1..deg(X)\ do\ A(X[i])\ od$ (the mapboth of Lisp)

Theorem:
Under very large analytic conditions, if the coefficients of the complexity descriptor of B,
$b_n \sim c.n^{\alpha}.\rho^{-n}$ for some constants c and $\alpha \leq 1/2$, then the average complexity of A
$\overline{\tau a_n} \sim c.\theta \frac{\Gamma(\alpha - 1/2)}{\Gamma(\alpha)} n^{\alpha + 1/2}$

Proof: from the true nature of the singularity and the basic asymptotics of the o.g.f. $t(z)$ for trees; translating the program, we get the equation:
$\tau a(z) = \tau b(z) + z\tau a(z)\Phi'(t(z))$ from which
$\tau a(z) = \tau b(z).zt'(z)/t(z)$

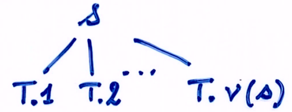
TOP-DOWN RECURSION

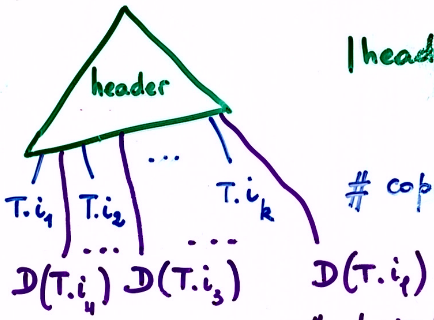
(P. Flajolet , JMS)

Formal derivation (generalized)

S = set of symbols   constants operators variables...

D = set of derivation rules

with T = 

$D(T)$ = 

$|\text{header}| = \eta(s)$

copies $= \alpha(s)$

derived subtrees $= \beta(s)$

Equations for the family of trees

$$\mathcal{F} = \sum_{s \in S} s\,(\underbrace{\mathcal{F}, \mathcal{F}, \dots \mathcal{F}}_{\nu(s)})$$

$$\Downarrow$$

$$f(z) = \sum z \cdot (f(z))^{\nu(s)}$$

Complexity of Differentiation algorithms

Algebraic expressions with $+, -, x, \sqrt{,}/$

Thm : The average running time of standard differentiation is $O(n^{3/2}$ compared to a $O(n^2)$ worst case behaviour

If sharing of subexpressions is allowed then the average running keeps linear

The complexity calculus applies more generally to generalized tree transducers, specified along the same lines;
4 types of average behaviours are possible:
- linear
- sesquilinear
- quadratic
- exponential

Size of the derived trees ≈ cost of derivation

$$\mathrm{trans}(s(T.1,\dots,T.m)) = \eta(s) + \sum{}' \alpha(s)_i\, |T.i| + \sum \beta(s)_i\, \mathrm{trans}(T.i)$$

$$\mathrm{trans}(z) = \sum_{T \in \mathcal{F}} \mathrm{trans}(T)\, z^{|T|}$$

Using "complexity calculus"* rules

$$\mathrm{trans}(z \mid T.\mathrm{root} = s) = \eta(s) \cdot z \cdot (f(z))^{\nu(s)} + \alpha(s) \cdot z^2 \cdot f'(z) \cdot (f(z))^{\nu(s)-1} + \beta(s) \cdot z \cdot \mathrm{trans}(z) \cdot (f(z))^{\nu(s)-1}$$

$$\mathrm{trans}(z) = \frac{z \cdot E(f(z)) + z \cdot f'(z) \cdot A(f(z))}{1 - z \cdot B(f(z))}$$

$$E(u) = \sum \eta(s)\, u^{\nu(s)}$$

$$A(u) = \sum \alpha(s)\, u^{\nu(s)-1}$$

$$B(u) = \sum \beta(s)\, u^{\nu(s)-1}$$

Analytic study

f has its dominant singularity in $z = \rho \in \mathbb{R}^+$ and locally

$$f(z) = \tau + \gamma\left(1 - \frac{z}{\rho}\right)^{1/2} + \text{s.o.t.} \qquad \tau = f(\rho)$$

$$f'(z) = \tau' + \gamma'\left(1 - \frac{z}{\rho}\right)^{-1/2} + \text{s.o.t.}$$

$$\text{numerator} = \rho^2 \alpha \gamma' \left(1 - \frac{z}{\rho}\right)^{-1/2} + \text{s.o.t.}$$

denominator and trans $\Big\}$ depends on $B(\tau)$?? $\Phi'(\tau)$

- $B \neq$ constant
  - $B(\tau) < \Phi'(\tau)$
    $$\text{trans}(z) = \frac{\rho^2 \alpha \gamma'}{1 - \rho B(\tau)} \left(1 - \frac{z}{\rho}\right)^{-1/2} + \text{s.o.t.}$$
  - $B(\tau) = \Phi'(\tau)$
    $$\text{trans}(z) = \frac{\rho^2 \alpha \gamma'}{-\rho \gamma B'(\tau)} \left(1 - \frac{z}{\rho}\right)^{-1} + \text{s.o.t.}$$
  - $B(\tau) > \Phi'(\tau)$
    singularity (pole) in $z = \omega < \rho$
- $B =$ constant
  - $<$ and $>$ similar
  - $=$
    $$\text{trans}(z) = \rho^2 \alpha \gamma' \left(1 - \frac{z}{\rho}\right)^{-3/2} + \text{s.o.t.}$$

More algorithms

Tree compatibility: returns the greatest common root part of X and Y; average cost is $O(1)$ (compare to naive string matching)

Tree matching: searching a motif in a tree, at all positions; average cost is $O(n)$ the constant depending on the motif's shape

Tree simplification: applying $t - t = 0$ syntactically; average cost is linear in the input size

Further developments (by the AofA community): recursive tree simplification, boolean expressions, unification, higher order differentiation, rewriting systems, etc.

Limiting laws can also be derived in a systematic way

Tree compaction

How much space can be saved by sharing systematically the identical subtrees of a given binary tree?

Complete binary tree of height h has size $n = 2^{h+1} - 1$ and after compaction has size $h + 1$
Right comb of height h has size $n = h + 1$ and is not changed by compaction

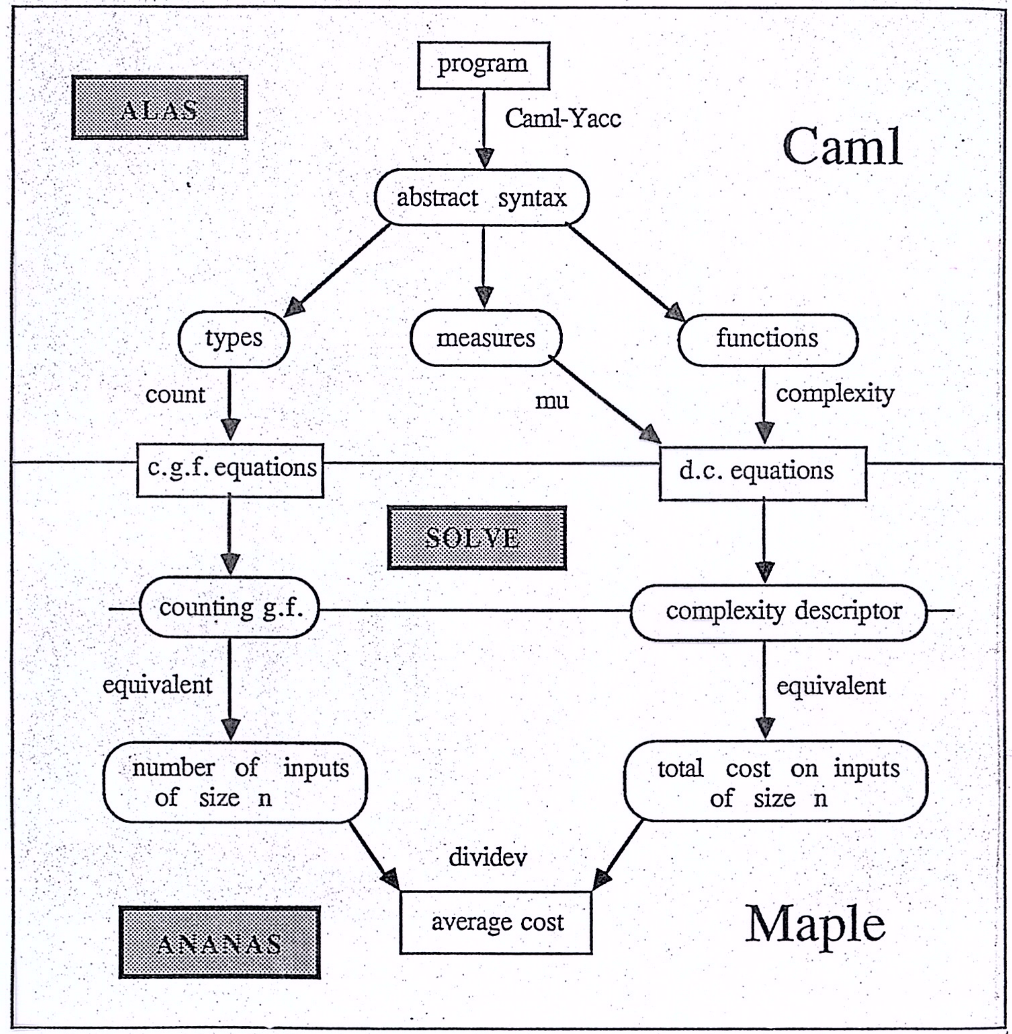
Theorem: The average size of a tree of size n after compaction is $O(n/\sqrt{\log\log n})$

Comment: such a result cannot be obtained directly by the complexity calculus! a weaker version along these lines gives $o(n)$

Automating the process

by P. Flajolet, B. Salvy, P. Zimmermann

ΛΥΩ



ALAS
program
Caml-Yacc
Caml
abstract syntax
types
measures
functions
count
mu
complexity
c.g.f. equations
d.c. equations
SOLVE
counting g.f.
complexity descriptor
equivalent
equivalent
number of inputs of size n
total cost on inputs of size n
dividev
average cost
ANANAS
Maple

Dérivation formelle

```
type  expression  =  zero | one | x
                    | plus(expression,expression)
                    | times(expression,expression)
                    | expo(expression);
      plus,times,expo,zero,one,x  =  atom(1);

function  diff(e:expression):expression;
case  e  of
   plus(e1,e2)          :  plus(diff(e1),diff(e2));
   times(e1,e2)         :  plus(times(diff(e1),copy(e2)),
                                times(copy(e1),diff(e2)));
   expo(e1)             :  times(diff(e1),copy(e));
   zero                 :  zero;
   one                  :  zero;
   x                    :  one
end;

function  copy  (e:expression):expression;

measure  plus,times,expo,zero,one,x  :  1;
```

```
#analyze"diff";;
Average cost for diff on random inputs of size n is:

                    1/2     1/2  3/2
           (126 + 6    ) Pi     n
1/2 ------------------------------ + O(n)
                  1/2 3/2  3/4    1/2
      (- 1 + 2 6     )    6     23
```

Work in progress...