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

[31] PF, JMS: A complexity calculus for classes of recursive programs over tree structures. (1981)
Basics on trees (cf. B. Sedgewick’s talk)

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

Let \( \Phi(u) \) be the enumerative polynomial of \( \Omega \): \([u^n]\Phi(u) = \text{card}\{\omega \in \Omega|\delta(\omega) = n\} \); then the o.g.f. \( t(z) \) of \( T \) 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 ≤ i ≤ 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
  \[ B = \cdot + \text{binary symbol} \]
  \( \uparrow \)
  \text{leaf} 
  \( \times \)

- Pattern: A binary tree \( P \)

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

- Occurrence
  \[ T = \]
**Algorithm**

```
root? (I, T) =
    if I = x then true
    elseif T = x then false
    elseif root? (I_left, I_left)
        then root? (I_right, I_right)
    else

occ? (I, T) = root? (I, T); if I ≠ x then occ? (I_left, I_left)
    else occ? (I_right, I_right)
```
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)

\( \tau A(X_1, X_2, \ldots, X_m) \): cost of running \( A \) on \( X_1, X_2, \ldots, X_m \) of sizes \( n_1, n_2, \ldots, n_m \)
consider \( \tau A(z_1, z_2, \ldots, z_m) = \sum_{|X_1|=n_1, \ldots, |X_m|=n_m} \tau A(X_1, \ldots, X_m) z_1^{n_1} \ldots z_m^{n_m} \)
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.
\( \chi A(z_1, z_2, \ldots, z_m) = \sum_{|X_1|=n_1, \ldots, |X_m|=n_m} A_{\text{true}}(X_1, \ldots, X_m) z_1^{n_1} \ldots z_m^{n_m} \)

Challenge: translate the constructors and basic operations
\( A = C(B_1, B_2, \ldots, B_k) \) \( \Rightarrow \)
\( \tau A = \Gamma(\tau B_1, \tau B_2, \ldots, \tau B_k; \chi B_1, \chi B_2, \ldots, \chi B_k) \)
\[ A = B_1; B_2 \rightarrow \tau A(z) = \tau B_1(z) + \tau B_2(z) \]

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

\[ A = \text{if } Q \text{ then } B \text{ else } C \rightarrow \tau A(z) = \tau Q(z) + \tau B(z|Q) + \tau C(z|\neg Q) \]

\[ A = \text{for } i \text{ do } B(X[i]) \text{ od } \rightarrow \tau A(z) = z\tau B(z)\Phi'(t(z)) \]

copying a tree \(\rightarrow \tau\text{copy}(z) = zt'(z)\)

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)

\[ \tau_{\text{root}}(z) = B(z) = \sum_{t \in B} 1.2^t \]

\[ \tau_{\text{root}}(\frac{z}{z^2}) = B(z) + z \cdot \tau_{\text{root}}(\frac{z}{z^2}) \cdot B(z) \]

\[ + z \cdot B(z) \cdot \tau_{\text{root}}(\frac{1}{z}) \]

\[ = 3B(z) - 2 \]

\[ \tau_{\text{root}}(\frac{z}{z^2}) = z \cdot B(z) = B(z) - 1 \]

\[ \tau_{\text{root}}(\frac{z}{z^2}) = B(z) + 2 \cdot \tau_{\text{root}}(\frac{z}{z^2}) \cdot B(z) \]

\[ + z \cdot \tau_{\text{root}}(\frac{z}{z^2}) \cdot \tau_{\text{root}}(\frac{1}{z}) \]

\[ = (5 - 3z)B - 4 \quad 4.74 \]

\[ \tau_{\text{root}}(\frac{z}{z^2}) = (5 - 2z)B - 4 \quad 4.12 \]

\[ \tau_{\text{root}}(\frac{z}{z^2}) = (7 - 7z - z^2)B - 6 + 2z \quad 5.316 \]
Complexity of iteration over subtrees

\( A(X) = B(X); \) for \( i = 1..\text{deg}(X) \) 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 \( A \)
\( \bar{\tau}_{an} \sim c.\theta \frac{\Gamma(\alpha-1/2)}{\Gamma(\alpha)} n^{\alpha+1/2} \)

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**

Formal derivation (generalized)

\[ S = \text{set of symbols} \quad \text{constants, operators, variables...} \]

\[ D = \text{set of derivation rules} \]

\[ T = \begin{array}{c}
T_1
\vdots
T_k
\end{array} \]

\[ D(T) = \begin{array}{c}
\text{header} \\
\text{# copies} = \alpha(\#) \\
\text{# derived subtrees} = \beta(\#)
\end{array} \]

Equations for the family of trees

\[ T_r = \sum_{\Delta \in S^*} \alpha \left( \frac{T_r}{T} \right) \]

\[ f(3) = \sum_{\Delta \in S^*} \beta \left( \sum_{\Delta \in S^*} \right) \]
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

\[ \text{trans} \left( \alpha(T.1, \ldots, T.m) \right) = \eta \left( \alpha \right) \]
\[ + \sum \alpha \left( \alpha \right) i \left( T.i \right) \]
\[ + \sum \beta \left( \alpha \right) i \text{trans} \left( T.i \right) \]

\[ \text{trans} \left( \beta \right) = \sum_{T \in \mathcal{T}} \text{trans} \left( T \right) \beta \left( T \right) \]

Using "complexity calculus" rules

\[ \text{trans} \left( \beta \mid T.\text{root} = 2 \right) = \]
\[ \eta \left( \beta \right) \cdot \beta \cdot \left( f \left( \beta \right) \right)^{\nu \left( \beta \right)} \]
\[ + \alpha \left( \beta \right) \cdot \beta^2 \cdot f' \left( \beta \right) \cdot \left( f \left( \beta \right) \right)^{\nu \left( \beta \right) - 1} \]
\[ + \beta \left( \beta \right) \cdot \beta \cdot \text{trans} \left( \beta \right) \cdot \left( f \left( \beta \right) \right)^{\nu \left( \beta \right) - 1} \]

\[ \text{trans} \left( \beta \right) = \beta \cdot E \left( f \left( \beta \right) \right) + \beta \cdot f' \left( \beta \right) \cdot A \left( f \left( \beta \right) \right) \]
\[ 1 - \beta \cdot B \left( f \left( \beta \right) \right) \]

\[ E \left( \omega \right) = \sum \eta \left( \omega \right) \omega^{\nu \left( \omega \right)} \]
\[ A \left( \omega \right) = \sum \alpha \left( \omega \right) \omega^{\nu \left( \omega \right) - 1} \]
\[ B \left( \omega \right) = \sum \beta \left( \omega \right) \omega^{\nu \left( \omega \right) - 1} \]
Analytic study

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

\[
\hat{f}(z) = \tau + \mathcal{O} \left( 1 - \frac{\rho^2}{z} \right)^{\frac{1}{2}} + \text{s.o.t.} \quad \tau = \hat{f}(\rho)
\]

\[
\hat{f}'(z) = \tau' + \mathcal{O} \left( 1 - \frac{\rho^2}{z} \right)^{-\frac{1}{2}} + \text{s.o.t.}
\]

numerator \( = \rho^2 \alpha^{\prime} \left( 1 - \frac{\rho^2}{\rho^2} \right)^{-\frac{1}{2}} + \text{s.o.t.} \)

denominator \( \} \) depends on \( B(\tau) \) !? \( \hat{\phi}'(\tau) \)

\( B \neq \text{constant} \)
- \( \text{trans (z)} = \frac{\rho^2 \alpha \gamma'}{1 - \rho B(\tau) \rho} \left( 1 - \frac{\rho^2}{\rho^2} \right)^{-\frac{1}{2}} + \text{s.o.t.} \)
- \( \text{trans (z)} = \frac{\rho^2 \alpha \gamma'}{-\rho B(\tau) \rho} \left( 1 - \frac{\rho^2}{\rho^2} \right)^{-\frac{1}{2}} + \text{s.o.t.} \)
- \( \text{trans (z)} = \frac{\rho^2 \alpha \gamma'}{-\rho B(\tau) \rho} \left( 1 - \frac{\rho^2}{\rho^2} \right)^{-\frac{1}{2}} + \text{s.o.t.} \)

\( \text{singularly (pole) in } z = \omega < \rho \)

\( B = \text{constant} \)
- \( \text{trans (z)} = \rho^2 \alpha \gamma' \left( 1 - \frac{\rho^2}{\rho^2} \right)^{-\frac{3}{2}} + \text{s.o.t.} \)
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
Dérivation formelle

```plaintext
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:

\[
\begin{align*}
  & \frac{1}{2} \quad \frac{1}{2} \quad \frac{3}{2} \\
  & (\frac{126}{1} + 6 \pi) \quad n \\
  & \frac{1}{2} \quad \frac{1}{3} \quad \frac{3}{4} \quad \frac{3}{2} \\
  \text{with} \quad (-1 + 2 \pi) \quad \frac{6}{1} \quad 23
\end{align*}
\]

\[19\]
Work in progress...