Chapter 6

Recurrence relations and their closed-form solutions

In "divide and conquer" algorithms one usually ends up with a recurrence relation that "defines" the "timing function", $T(n)$. For example, it might look like

$$T(n) = \begin{cases} 1 & \textbf{if } n = 1 \\ T(n/2) + 1 & \textbf{otherwise} \end{cases}$$

In order to assess the "goodness" of the proposed algorithm by comparison to either our expectations or to another algorithm, we need to know $T(n)$ in "closed" form in terms of known functions, for example, n^r for $r > 0$, c^n for $c > 1$, $\log_b n$ for some integer $b > 1$.

Often, a preliminary analysis need only worry about the "asymptotic behaviour" of the algorithm, i.e., the behaviour for *large* inputs (n is the input size). "Big-O" notation is an excellent tool in this case, therefore the solution of recurrences is often sought in such notation. On occasion one requires an "exact" solution (this is much harder to achieve in general).

There is a big variety of recurrence relations and an equally big variety of solution techniques. Some restricted cases are handled well by packages such as *Mathematica* or *Maple V*. For the mathematical reasons that make the solutions tick the best reference is perhaps Knuth *et al.* "*Concrete Mathematics*" (Addison-Wesley).

In this chapter we restrict attention to simple classes of recurrences taken from both the "additive" and "multiplicative" cases. These characterizations in quotes refer to the manner of handling the argument of the recurrence. E.g., the recurrence above is multiplicative as the recursive call is to an argument obtained by *halving* the original argument n. On the other hand, the Fibonacci recurrence is additive.

6.1. Big-O, small-o, and the "other" $\sim$

This notation is due to the mathematician E. Landau and is in wide use in number theory, but also in computer science in the context of measuring (bounding above) computational complexity of algorithms for all "very large inputs".

6.1.1 Definition. Let f and g be two total functions of one variable, where $g(x) > 0$, for all x. Then

1. $f = O(g)$ —also written as $f(x) = O(g(x))$— read "f is big-oh g", means that there are positive constants C and K in $\mathbb{N}$ such that
$$x > K \text{ implies } |f(x)| \le Cg(x)$$
2. $f = o(g)$ —also written as $f(x) = o(g(x))$— read "f is small-oh g", means that
$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$
3. $f \sim g$ —also written as $f(x) \sim g(x)$— read "f is of the same order as g", means that
$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1$$

□

"$\sim$" between two sets A and B, as in $A \sim B$, means that there is a 1-1 correspondence $f : A \to B$. Obviously, the context will protect us from confusing this $\sim$ with the one introduced just now, in 6.1.1.

Both definitions 2. and 3. require some elementary understanding of differential calculus. Case 2. says, intuitively, that as x gets extremely large, then the fraction $f(x)/g(x)$ gets extremely small, infinitesimally close to 0. Case 3. says, intuitively, that as x gets extremely large, then the fraction $f(x)/g(x)$ gets infinitesimally close to 1; that is, the function outputs are infinitesimally close to each other.

6.1.2 Example.

1. $x = O(x)$ since $x \le 1 \cdot x$ for $x \ge 0$.
2. $x \sim x$, since $x/x = 1$, and stays 1 as x gets very large.
3. $x = o(x^2)$ since $x/x^2 = 1/x$ which trivially goes to 0 as x goes to infinity.
4. $2x^2 + 1000^{1000}x + 10^{350000} = O(x^2)$. Indeed
$$\frac{2x^2 + 1000^{1000}x + 10^{350000}}{3x^2} = 2/3 + 1000^{1000}/x + 10^{350000}/x^2 < 1$$

for $x > K$ for some well chosen K. Note that $1000^{1000}/x$ and $10^{350000}/x^2$ will each be $< 1/6$ for all sufficiently large x-values: we will have $2/3 + 1000^{1000}/x + 10^{350000}/x^2 < 2/3 + 1/6 + 1/6 = 1$ for all such x-values. Thus $2x^2 + 1000^{1000}x + 10^{350000} < 3x^2$ for $x > K$ as claimed.

In many words, in a polynomial, the order of magnitude is determined by the highest power term. $\square$

The last example motivates

6.1.3 Proposition. *Suppose that $f(x) \geq 0$ for all $x > L$, hence $|f(x)| = f(x)$ for all $x > L$. Now, if $f(x) \sim g(x)$, then $f(x) = O(g(x))$.*

Proof. The assumption says that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1$$

From "calculus 1" (1st year differential calculus) we learn that this implies that for some K, $x > K$ entails

$$\left|\frac{f(x)}{g(x)} - 1\right| < 1$$

hence

$$-1 < \frac{f(x)}{g(x)} - 1 < 1$$

therefore, $x > \max(K, L)$ implies $f(x) < 2g(x)$. $\square$

6.1.4 Proposition. *Suppose that $f(x) \geq 0$ for all $x > L$, hence $|f(x)| = f(x)$ for all $x > L$. Now, if $f(x) = o(g(x))$, then $f(x) = O(g(x))$.*

Proof. The assumption says that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

From calculus 1 we learn that this implies that for some K, $x > K$ entails

$$\left|\frac{f(x)}{g(x)}\right| < 1$$

hence

$$-1 < \frac{f(x)}{g(x)} < 1$$

therefore, $x > \max(K, L)$ implies $f(x) < g(x)$. $\square$

These two propositions add to our toolbox:

6.1.5 Example.

1. $\ln x = o(x^r)$ for any positive real r. Here "ln" stands for $\log_e$ where e is the Euler constant

$$2.71828182845904523536028747135266249775724709 37\ldots$$

Seeing that both numerator and denominator

$$\lim_{x\to\infty} \frac{\ln x}{x^r}$$

go to ∞, we have here (if we do not do anything to mitigate) an impasse: We have a "limit" that is indeterminate:

$$\frac{\infty}{\infty}$$

So, we will use "l'Hôpital's rule" (the limit of the fraction is equal to the limit of the fraction of the derivatives):

$$\lim_{x\to\infty} \frac{\ln x}{x^r} = \lim_{x\to\infty} \frac{1/x}{rx^{r-1}} = \lim_{x\to\infty} \frac{1}{rx^r} = 0$$

2. $\ln x = O(\log_{10}(x))$. In fact, you can go from one log-base to the other:

$$\log_e(x) = \frac{\log_{10}(x)}{\log_{10}(e)}$$

The claim follows from 6.1.3 since trivially $\ln x \sim \log_{10}(x)/\log_{10}(e)$. For that reason —and since multiplicative constants are hidden in big-O notation— complexity- and algorithms-practitioners omit the base of the logarithm and write things like $O(\log n)$ and $O(n \log n)$. □

6.2. Solving recurrences; the additive case

The general case here is of the form[†]

$$\begin{aligned} T_0 &= k \\ s_n T_n &= v_n T_{n-1} + f(n) \textbf{ if } n > 0 \end{aligned}$$

a recurrence defining the *sequence* T_n, or equivalently, the *function* $T(n)$ (both jargons and notations spell out the same thing), in terms of the *known* functions (sequences) $s_n, v_n, f(n)$.

For the general case see Knuth cited above. Here we will restrict attention to the case $s_n = 1$ for all n and $v_n = a$ (a constant) for all n.

Subcase 1. $(a = 1)$ Solve

$$\begin{aligned} T_0 &= k \\ T_n &= T_{n-1} + f(n) \textbf{ if } n > 0 \end{aligned} \tag{1}$$

[†]Note the "additivity" in the relation between indices/arguments: n vs. $n-1$.

From (1), $T_n - T_{n-1} = f(n)$, thus

$$\sum_{i=1}^{n}(T_i - T_{i-1}) = \sum_{i=1}^{n} f(i)$$

the lower summation value dictated by the lowest valid value of $i-1$ according to (1).

6.2.1 Remark. The summation in the lhs above is called a "*telescoping (finite) series*" because the terms $T_1, T_2, \ldots, T_{n-1}$ appear both positively and negatively and pairwise cancel. Thus the series "contracts" into $T_n - T_0$ like a (hand held) telescope. □

Therefore

$$\begin{aligned} T_n &= T_0 + \textstyle\sum_{i=1}^{n} f(i) \\ &= k + \textstyle\sum_{i=1}^{n} f(i) \end{aligned} \tag{2}$$

If we know how to get the sum in (2) in closed form, then we solved the problem!

6.2.2 Example. Solve

$$p_n = \begin{cases} 2 & \textbf{if } n = 1 \\ p_{n-1} + n & \textbf{otherwise} \end{cases} \tag{3}$$

Here

$$\sum_{i=2}^{n}(p_i - p_{i-1}) = \sum_{i=2}^{n} i$$

Note the lower bound of the summation: It is here 2, to allow for the lowest $i-1$ value possible. That is 1 according to 3, hence $i = 2$.

Thus,

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

(Where did I get the $(n+2)(n-1)/2$ from?) The above answer is the same as (verify!)

$$p_n = 1 + \frac{(n+1)n}{2}$$

obtained by writing

$$2 + \sum_{i=2}^{n} i = 1 + \sum_{i=1}^{n} i$$

Subcase 2. ($a \neq 1$) Solve

$$\begin{aligned} &T_0 = k \\ &T_n = aT_{n-1} + f(n) \textbf{ if } n > 0 \end{aligned} \tag{4}$$

(4) is the same as

$$\frac{T_n}{a^n} = \frac{T_{n-1}}{a^{n-1}} + \frac{f(n)}{a^n}$$

To simplify notation, set

$$t_n \overset{\text{Def}}{=} \frac{T_n}{a^n}$$

thus the recurrence (4) becomes

$$\begin{aligned} t_0 &= k \\ t_n &= t_{n-1} + \frac{f(n)}{a^n} \textbf{ if } n > 0 \end{aligned} \tag{5}$$

By subcase 1, this yields

$$t_n = k + \sum_{i=1}^{n} \frac{f(i)}{a^i}$$

from which

$$T_n = ka^n + a^n \sum_{i=1}^{n} \frac{f(i)}{a^i} \tag{6}$$

6.2.3 Example. As an illustration solve the recurrence below.

$$T_n = \begin{cases} 1 & \textbf{if } n = 1 \\ 2T_{n-1} + 1 & \textbf{otherwise} \end{cases} \tag{7}$$

To avoid trouble, note that the lowest term here is T_1, hence its "translation" to follow the above methodology will be "$t_1 = T_1/2^1 = 1/2$". So, the right hand side of (6) applied here will have "ka^{n-1}" instead of "ka^n" (Why?) and the indexing in the summation will start at $i = 2$ (Why?)

Thus, by (6),

$$\begin{aligned} T_n &= 2^n(1/2) + 2^n \textstyle\sum_{i=2}^{n} \frac{1}{2^i} \\ &= 2^{n-1} + 2^n\left(\frac{(2^{-1})^{n+1} - 1}{2^{-1} - 1} - 1 - \frac{1}{2}\right) \\ &= 2^{n-1} + 2^n\left(2 - 2^{-n} - 1 - \frac{1}{2}\right) \\ &= 2^n - 1 \end{aligned}$$

In the end you will probably agree that it is easier to redo the work with (7) directly, first translating it to

$$t_n = \begin{cases} 1/2 & \textbf{if } n = 1 \\ t_{n-1} + 1/2^n & \textbf{if } n > 1 \end{cases} \tag{8}$$

rather than applying (6)!

We immediately get from (8)

$$T_n = 2^n t_n = 2^n \Big(1/2 + \sum_{i=2}^{n} 1/2^i\Big) = 2^n \Big(1/2 + \frac{(2^{-1})^{n+1} - 1}{2^{-1} - 1} - 1 - 1/2\Big)$$

etc.

The red terms are subtracted as they are missing from our $\sum$. The blue formula used is for

$$\sum_{i=0}^{n} 1/2^i \qquad \square$$

6.3. *Solving recurrences; the multiplicative case*

Subcase 1.

$$T(n) = \begin{cases} k & \textbf{if } n = 1 \\ aT(n/b) + c & \textbf{if } n > 1 \end{cases} \tag{1}$$

were a, b are positive integer constants ($b > 1$) and k, c any constants. Recurrences like (1) above arise in *divide and conquer* solutions to problems. For example, *binary search* has timing governed by the above recurrence with $b = 2, a = c = k = 1$.

Why does (1) with the above-mentioned parameters —$b = 2, a = c = k = 1$— capture the run time of binary search? First off, regarding "run time" let us be *specific*: we mean number of comparisons.

OK, to do such a search on a sorted (ascending order, say) array of length n, you first check the mid point (for a match with what you are searching for). If you found what you want, exit. If not, you know (due to the ordering) whether you should search the left half or the right half. So you call the procedure recursively on an arrow of length about $n/2$. This decision *and* call took $T(n/2) + 1$ comparisons. This equals $T(n)$. If the array has length 1, then you spend just one comparison, $T(1) = 1$.

We seek a general solution in big-O notation.

First convert to an "additive case" problem: To this end, seek a solution in the *restricted* set $\{n \in \mathbb{N} : n = b^m \text{ for some } m \in \mathbb{N}\}$. Next, set

$$t(m) = T(b^m) \tag{2}$$

so that the recurrence becomes

$$t(m) = \begin{cases} k & \textbf{if } m = 0 \\ at(m-1) + c & \textbf{if } m > 0 \end{cases} \tag{3}$$

hence, from the work in the previous section,

$$\sum_{i=1}^{m} \Big(\frac{t(i)}{a^i} - \frac{t(i-1)}{a^{i-1}}\Big) = c \sum_{i=1}^{m} a^{-i}$$

therefore

$$t(m) = a^m k + ca^m \begin{cases} m & \textbf{if } a = 1 \\ a^{-1}\dfrac{(a^{-1})^m - 1}{a^{-1} - 1} & \textbf{if } a \neq 1 \end{cases}$$

or, more simply,

$$t(m) = \begin{cases} k + cm & \textbf{if } a = 1 \\ a^m k + c\dfrac{a^m - 1}{a - 1} & \textbf{if } a \neq 1 \end{cases}$$

Using O-notation, and going back to T we get:

$$T(b^m) = \begin{cases} O(m) & \textbf{if } a = 1 \\ O(a^m) & \textbf{if } a \neq 1 \end{cases} \tag{4}$$

or, *provided we remember that this solution relies on the assumption that n* has the form b^m:

$$T(n) = \begin{cases} O(\log n) & \textbf{if } a = 1 \\ O(a^{\log_b n}) & \textbf{if } a \neq 1 \end{cases} = \begin{cases} O(\log n) & \textbf{if } a = 1 \\ O(n^{\log_b a}) & \textbf{if } a \neq 1 \end{cases} \tag{5}$$

If $a > b$ then we get slower than linear "run time" $O(n^{\log_b a})$. If on the other hand $b > a > 1$ then we get a sublinear run time, since then $\log_b a < 1$.

The symbol means "can be omitted with loss of continuity".

Now an important observation. For functions $T(n)$ that are *increasing*,[†] i.e., $T(i) \leq T(j)$ if $i < j$ the restriction of n to have form b^m proves to be *irrelevant* in obtaining the solution. The solution is still given by (5) *for all n*. Here's why:

In the general case, n satisfies

$$b^{m-1} < n \leq b^m \text{ for some } m \geq 0 \tag{6}$$

Suppose now that $a = 1$ (upper case in (4)). We want to establish that $T(n) = O(\log n)$ for the general n (of (6)). By monotonicity of T and the second inequality of (6) we get

$$T(n) \overset{\text{by (6) right}}{\leq} T(b^m) \overset{\text{by (4)}}{=} O(m) \overset{\text{by (6) left}}{=} O(\log n)$$

The last invocation of (6) above used the first inequality therein.

The case where $a > 1$ is handled similarly. Here we found an answer $O(n^r)$ (where $r = \log_b a > 0$) *provided* $n = b^m$ (some m). Relax this proviso, and assume (6).

Now

$$T(n) \overset{\text{by (6) right}}{\leq} T(b^m) \overset{\text{by (4)}}{=} O(a^m) = O((b^m)^r) \overset{\text{Why?}}{=} O((b^{m-1})^r) \overset{\text{by (6) left}}{=} O(n^r)$$

where again the last invocation of (6) above used the first inequality therein.

[†]Such are the "complexity" or "timing" functions of algorithms.

Subcase 2.

$$T(n) = \begin{cases} k & \textbf{if } n = 1 \\ aT(n/b) + cn & \textbf{if } n > 1 \end{cases} \tag{1'}$$

were a, b are positive integer constants ($b > 1$) and k, c any constants. Recurrences like (1′) above also occur in divide and conquer solutions to problems. For example, *two-way merge sort* has timing governed by the above recurrence with $a = b = 2$ and $c = 1/2$. Quicksort has *average* run time governed, essentially, by the above with $a = b = 2$ and $c = 1$. Both lead to $O(n \log n)$ solutions. Also, *Karatsuba integer multiplication* has a run time recurrence as above with $a = 3, b = 2$.

These examples are named for easy look up, in case the trigger your interest or curiosity. It is not in the design of this course to expand on them. Merge Sort and Quicksort you might see in a course on data structures (e.g., EECS 2011) while Karatsuba's "fast multiplication" of natural numbers might appear in a course on algorithms like EECS 3101.

Setting at first (our famous *initial* restriction on n) $n = b^m$ for some $m \in \mathbb{N}$ and using (2) above we end up with a variation on (3):

$$t(m) = \begin{cases} k & \textbf{if } m = 0 \\ at(m-1) + cb^m & \textbf{if } m > 0 \end{cases} \tag{3'}$$

thus we need do

$$\sum_{i=1}^{m} \left(\frac{t(i)}{a^i} - \frac{t(i-1)}{a^{i-1}}\right) = c \sum_{i=1}^{m} (b/a)^i$$

therefore

$$t(m) = a^m k + ca^m \begin{cases} m & \textbf{if } a = b \\ (b/a)\dfrac{(b/a)^m - 1}{b/a - 1} & \textbf{if } a \neq b \end{cases}$$

Using O-notation, and using cases according as to $a < b$ or $a > b$ we get:

$$t(m) = \begin{cases} O(b^m m) & \textbf{if } a = b \\ a^m O(1) = O(a^m) & \textbf{if } b < a \qquad /^* \ (b/a)^m \to 0 \text{ as } m \to \infty \ ^*/ \\ O(b^m - a^m) = O(b^m) & \textbf{if } b > a \end{cases}$$

or, in terms of T and n, which is *restricted* to form b^m (using same calculational "tricks" as before):

$$T(n) = \begin{cases} O(n \log n) & \textbf{if } a = b \\ O(n^{\log_b a}) & \textbf{if } b < a \\ O(n) & \textbf{if } b > a \end{cases} \tag{4'}$$

The above solution is valid for *any* n without restriction, *provided* T is increasing. The proof is as before, so we will not redo it (you may wish to check the "new case" $O(n \log n)$ as an exercise).

In terms of complexity of algorithms, the above solution says that in a divide and conquer algorithm (governed by $(1')$) we have the following cases:

- The total size of all subproblems we solve (recursively) is *equal* to the original problem's size. Then we have a $O(n \log n)$ algorithm (e.g., merge sort).
- The total size of all subproblems we solve is *more* than the original problem's size. Then we go worse than linear ($\log_b a > 1$ in this case). An example is Karatsuba multiplication that runs in $O(n^{\log_2 3})$ time.
- The total size of all subproblems we solve is *less* than the original problem's size. Then we go in linear time (e.g., the problem of finding the k-th smallest in a *set* of n elements).

6.4. Generating Functions

We saw some simple cases of recurrence relations with additive and multiplicative index structure (we reduced the latter to the former). Now we turn to a wider class of additive index structure problems where our previous technique of utilizing a "telescoping sum"

$$\sum_{i=1}^{n}(t(i) - t(i-1))$$

does not apply because the right hand side still refers to $t(i)$ for some $i < n$. Such is the case of the well known Fibonacci sequence F_n given by

$$F_n = \begin{cases} 0 & \textbf{if } n = 0 \\ 1 & \textbf{if } n = 1 \\ F_{n-1} + F_{n-1} & \textbf{if } n > 1 \end{cases}$$

The method of *generating functions* that solves this harder problem also solves the previous problems we saw.

Here's the method in *outline*. We will then embark on a number of fully worked out examples.

Given a recurrence relation

$$t_n = \ldots t_{n-1} \ldots t_{n-2} \ldots t_{n-3} \ldots \tag{1}$$

with the appropriate "starting" (initial) conditions. We want t_n in "closed form" in terms of known functions. Here are the steps:

1. Define a *generating function* of the *sequence* $t_0, t_1, \ldots, t_n, \ldots$

$$\begin{aligned} G(z) &= \sum_{i=0}^{\infty} t_i z^i \\ &= t_0 + t_1 z + t_2 z^2 + \cdots + t_n z^n + \cdots \end{aligned} \tag{2}$$

   (2) is a *formal power series*, where *formal* means that we only are interested in the *form* of the "infinite sum" and *not* in any issues of convergence[†] (therefore "meaning") of the sum. It is stressed that our disinterest in convergence matters is *not* a simplifying convenience but it is due to the fact that convergence issues are *irrelevant* to the problem at hand.

   In particular, we will *never* have to consider values of z or make substitutions into z.

2. Using the recurrence (1), find a *closed form* of $G(z)$ as a function of z (this *can* be done *prior* to knowing the t_n in closed form!)

3. Expand the closed form $G(z)$ back into a power series

$$\begin{aligned} G(z) &= \sum_{i=0}^{\infty} a_i z^i \\ &= a_0 + a_1 z + a_2 z^2 + \cdots + a_n z^n + \cdots \end{aligned} \tag{3}$$

   But now we *do have* the a_n's in terms of known functions, because we know $G(z)$ in closed form! We only need to compare (2) and (3) and proclaim

$$t_n = a_n \quad \text{for } n = 0, 1, \ldots$$

   The problem has been solved.

Steps 2. and 3. embody all the real work. We will illustrate by examples how this is done in practice, but first we need some "tools":

From here on we will put our in use to advise the reader of what can be omitted.

The derivation of these formulas is trivial, but really long, so let us concentrate on 2 or 3 "boxed" results —forgetting the arithmetic!— that we will be employing!
These will be boxed and provided as aids in, e.g., an exam situation.

The Binomial Expansion. For our purposes we will be content with just one tool, the "binomial expansion theorem" of calculus:

For any *real* m,
$$\begin{aligned} (1+z)^m &= \sum_{r=0}^{\infty} \binom{m}{r} z^r \\ &= \cdots + \binom{m}{r} z^r + \cdots \end{aligned} \tag{4}$$

[†]In Calculus one learns that power series converge in an interval like $|z| < r$ for some real $r \geq 0$. The $r = 0$ case means the series diverges for *all* z.

where for any $r \in \mathbb{N}$ and $m \in \mathbb{R}$

$$\binom{m}{r} \overset{\text{def}}{=} \begin{cases} 1 & \textbf{if } r = 0 \\ \dfrac{m(m-1)\cdots(m-[r-1])}{r!} & \textbf{otherwise} \end{cases} \tag{5}$$

The expansion (4) terminates with last term

$$\binom{m}{m} z^m \overset{\text{by (5)}}{=} z^m$$

as the "binomial theorem of *Algebra* says, iff m is a *positive integer*. In *all* other cases (4) is non-terminating (infinitely many terms). As we remarked before, we will not be concerned with when (4) converges.

Note that (5) gives the familiar

$$\begin{aligned}\binom{m}{r} &= \frac{m(m-1)\cdots(m-[r-1])}{r!} \\ &= \frac{m(m-1)\cdots(m-[r-1])(m-r)\cdots 2\cdot 1}{r!(m-r)!} \\ &= \frac{m!}{r!(m-r)!}\end{aligned}$$

when $m \in \mathbb{N}$. *In all other cases we use* (5) for if $m \notin \mathbb{N}$, then "$m!$" is meaningless.

Let us record the very useful special case when m is a *negative integer*, $-n$ ($n > 0$).

$$\begin{aligned}(1+z)^{-n} &= \cdots + \tfrac{-n(-n-1)\cdots(-n-[r-1])}{r!} z^r + \cdots \\ &= \cdots + (-1)^r \tfrac{n(n+1)\cdots(n+[r-1])}{r!} z^r + \cdots \\ &= \cdots + (-1)^r \tfrac{(n+[r-1])\cdots(n+1)n}{r!} z^r + \cdots \\ &= \cdots + (-1)^r \tbinom{n+r-1}{r} z^r + \cdots \end{aligned} \tag{6}$$

$$(1-z)^{-n} = \cdots + \binom{n+r-1}{r} z^r + \cdots \tag{7}$$

Finally, let us record in "boxes" some important special cases of (6) and (7)

$$\boxed{\begin{aligned}(1-z)^{-1} = \frac{1}{1-z} &= \cdots + \tbinom{r}{r} z^r + \cdots \\ &= \cdots + z^r + \cdots\end{aligned}} \tag{8}$$

The above is the familiar "converging geometric progression" (converging for $|z| < 1$, that is, <u>but this is the last time I'll raise **irrelevant** convergence issues</u>). Two more special cases of (6) will be helpful:

$$\boxed{\begin{aligned}(1-z)^{-2} = \tfrac{1}{(1-z)^2} &= \cdots + \tbinom{r+1}{r} z^r + \cdots \\ &= 1 + 2z + \cdots + (r+1)z^r + \cdots\end{aligned}} \tag{9}$$

and

$$\boxed{\begin{aligned}(1-z)^{-3} = \tfrac{1}{(1-z)^3} &= \cdots + \tbinom{r+2}{r} z^r + \cdots \\ &= 1 + 3z + \cdots + \tfrac{(r+2)(r+1)}{2} z^r + \cdots\end{aligned}} \tag{10}$$

6.4.1 Example. Solve the recurrence

$$\begin{aligned} a_0 &= 1 \\ a_n &= 2a_{n-1} + 1 \qquad \textbf{if } n > 0 \end{aligned} \tag{i}$$

Write (i) as

$$a_n - 2a_{n-1} = 1 \tag{ii}$$

Next, form the generating function for a_n, and a "shifted" copy of it (multiplied by $2z$; z does the shifting) underneath it (this was "inspired" by (ii)):

$$\begin{aligned} G(z) &= a_0 + a_1 z + a_2 z^2 + \cdots + a_n z^n + \cdots \\ 2zG(z) &= \phantom{a_0 +{}} 2a_0 z + 2a_1 z^2 + \cdots + 2a_{n-1} z^n + \cdots \end{aligned}$$

Subtract the above term-by-term to get

$$\begin{aligned} G(z)(1-2z) &= 1 + z + z^2 + z^3 + \cdots \\ &= \frac{1}{1-z} \end{aligned}$$

Hence

$$G(z) = \frac{1}{(1-2z)(1-z)} \tag{iii}$$

(iii) is $G(z)$ in closed form. To expand it back to a (known) power series we first use the "*partial fractions*" method (familiar to students of calculus) to write $G(z)$ as the sum of *two* fractions with *linear* denominators. I.e., find constants A and B such that (iv) below is true for all z:

$$\frac{1}{(1-2z)(1-z)} = \frac{A}{(1-2z)} + \frac{B}{(1-z)}$$

or

$$1 = A(1-z) + B(1-2z)$$

Setting in turn $z \leftarrow 1$ and $z \leftarrow 1/2$ we find $B = -1$ and $A = 2$, hence

$$\begin{aligned} G(z) &= \frac{2}{1-2z} - \frac{1}{1-z} \\ &= 2\big(\cdots (2z)^n \cdots\big) - \big(\cdots z^n \cdots\big) \\ &= \cdots (2^{n+1} - 1) z^n \cdots \end{aligned}$$

Comparing this known expansion with the original power series above, we conclude that

$$a_n = 2^{n+1} - 1$$

Of course, we solved this problem much more easily in Section 6.2. However due to its simplicity it was worked out here again to illustrate this new method. Normally, you apply the method of generating functions when there is *no other simpler way to do it.*

6.4.2 Example. Solve

$$\begin{aligned} &p_1 = 2 \\ &p_n = p_{n-1} + n \qquad \textbf{if } n > 1 \end{aligned} \tag{i}$$

Write (i) as

$$p_n - p_{n-1} = n \tag{ii}$$

Next, form the generating function for p_n, and a "shifted" copy of it underneath it (this was "inspired" by (ii)).
Note how this sequence starts with p_1 (rather than p_0). Correspondingly, the constant term of the generating function is p_1.

$$\begin{aligned} G(z) &= p_1 + p_2 z + p_3 z^2 + \cdots + p_{n+1} z^n + \cdots \\ zG(z) &= \quad p_1 z \;+ p_2 z^2 + \cdots + p_n z^n \quad + \cdots \end{aligned}$$

Subtract the above term-by-term to get

$$\begin{aligned} G(z)(1-z) &= 2 + 2z + 3z^2 + 4z^3 + \cdots + (n+1)z^n + \cdots \\ &= 1 + \frac{1}{(1-z)^2} \qquad \text{by (9)} \end{aligned}$$

Hence

$$\begin{aligned} G(z) &= \tfrac{1}{1-z} + \tfrac{1}{(1-z)^3} \\ &= \big(\cdots z^n \cdots\big) + \big(\cdots \tfrac{(n+2)(n+1)}{2} z^n \cdots\big) \qquad \text{by (10)} \\ &= \cdots \left(1 + \frac{(n+2)(n+1)}{2}\right) z^n \cdots \end{aligned}$$

Comparing this known expansion with the original power series above, we conclude that

$$p_{n+1} = 1 + \frac{(n+2)(n+1)}{2}, \text{ the coefficient of } z^n$$

or

$$p_n = 1 + \frac{(n+1)n}{2}$$

6.4.3 Example. Here is one that cannot be handled by the techniques of Section 6.2.

$$\begin{aligned} &s_0 = 1 \\ &s_1 = 1 \\ &s_n = 4s_{n-1} - 4s_{n-2} \qquad \textbf{if } n > 1 \end{aligned} \tag{i}$$

Write (i) as

$$s_n - 4s_{n-1} + 4s_{n-2} = 0 \tag{ii}$$

to "inspire"

$$\begin{aligned}
G(z) &= s_0 + s_1 z + s_2 z^2 + \cdots + s_n z^n + \cdots \\
4zG(z) &= \quad 4s_0 z + 4s_1 z^2 + \cdots + 4s_{n-1} z^n + \cdots \\
4z^2 G(z) &= \qquad 4s_0 z^2 + \cdots + 4s_{n-2} z^n + \cdots
\end{aligned}$$

By (ii),

$$\begin{aligned}
G(z)(1-4z+4z^2) &= 1 + (1-4)z \\
&= 1 - 3z
\end{aligned}$$

Since $1 - 4z + 4z^2 = (1-2z)^2$ we get

$$\begin{aligned}
G(z) &= \tfrac{1}{(1-2z)^2} - 3z\tfrac{1}{(1-2z)^2} \\
&= \big(\cdots (n+1)(2z)^n \cdots\big) - 3z\big(\cdots (n+1)(2z)^n \cdots\big) \\
&= \big(\cdots \big[(n+1)2^n - 3n2^{n-1}\big] z^n \cdots\big)
\end{aligned}$$

Thus,

$$\begin{aligned}
s_n &= (n+1)2^n - 3n2^{n-1} \\
&= 2^{n-1}(2n+2-3n) \\
&= 2^n(1-n/2)
\end{aligned}$$

6.4.4 Example. Here is another one that cannot be handled by the techniques of Section 6.2.

$$\begin{aligned}
s_0 &= 0 \\
s_1 &= 8 \\
s_n &= 2s_{n-1} + 3s_{n-2} \quad \textbf{if } n > 1 \qquad (i)
\end{aligned}$$

Write (i) as

$$s_n - 2s_{n-1} - 3s_{n-2} = 0 \qquad (ii)$$

Next,

$$\begin{aligned}
G(z) &= s_0 + s_1 z + s_2 z^2 + \cdots + s_n z^n + \cdots \\
2zG(z) &= \quad 2s_0 z + 2s_1 z^2 + \cdots + 2s_{n-1} z^n + \cdots \\
3z^2 G(z) &= \qquad 3s_0 z^2 + \cdots + 3s_{n-2} z^n + \cdots
\end{aligned}$$

By (ii),

$$G(z)(1-2z-3z^2) = 8z$$

The roots of $1 - 2z - 3z^2 = 0$ are

$$z = \frac{-2 \pm \sqrt{4+12}}{6} = \frac{-2 \pm 4}{6} = \begin{cases} -1 \\ 1/3 \end{cases}$$

hence $1 - 2z - 3z^2 = -3(z+1)(z-1/3) = (1-3z)(1+z)$, therefore

$$G(z) = \frac{8z}{(1-3z)(1+z)} = \frac{A}{1-3z} + \frac{B}{1+z} \text{ splitting into partial fractions}$$

By a calculation as in the previous example, $A = 2$ and $B = -2$, so

$$\begin{aligned} G(z) &= \tfrac{2}{1-3z} - \tfrac{2}{1+z} \\ &= 2\big(\cdots (3z)^n \cdots\big) - 2\big(\cdots (-z)^n \cdots\big) \\ &= (\cdots [2 \cdot 3^n - 2(-1)^n] z^n \cdots) \end{aligned}$$

hence $s_n = 2 \cdot 3^n - 2(-1)^n$

6.4.5 Example. The Fibonacci recurrence.

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \quad \textbf{if } n > 1 \end{aligned} \tag{i}$$

Write (i) as

$$F_n - F_{n-1} - F_{n-2} = 0 \tag{ii}$$

Next,

$$\begin{aligned} G(z) &= F_0 + F_1 z + F_2 z^2 + \cdots + F_n z^n + \cdots \\ zG(z) &= \quad F_0 z + F_1 z^2 + \cdots + F_{n-1} z^n + \cdots \\ z^2 G(z) &= \quad\quad F_0 z^2 + \cdots + F_{n-2} z^n + \cdots \end{aligned}$$

By (ii),

$$G(z)(1 - z - z^2) = z$$

The roots of $1 - z - z^2 = 0$ are

$$z = \frac{-1 \pm \sqrt{1+4}}{2} = \begin{cases} \dfrac{-1+\sqrt{5}}{2} \\ \dfrac{-1-\sqrt{5}}{2} \end{cases}$$

For convenience of notation, set

$$\phi_1 = \frac{-1+\sqrt{5}}{2}, \qquad \phi_2 = \frac{-1-\sqrt{5}}{2} \tag{iii}$$

Hence

$$\begin{aligned} 1 - z - z^2 &= -(z - \phi_1)(z - \phi_2) \\ &= -(\phi_1 - z)(\phi_2 - z) \end{aligned} \tag{iv}$$

therefore

$$G(z) = \frac{z}{1 - z - z^2} = \frac{A}{\phi_1 - z} + \frac{B}{\phi_2 - z} \text{ splitting into partial fractions}$$

from which (after some arithmetic that I will not show),

$$A = \frac{\phi_1}{\phi_1 - \phi_2}, \qquad B = \frac{\phi_2}{\phi_2 - \phi_1}$$

so

$$\begin{aligned}G(z) &= \tfrac{1}{\phi_1-\phi_2}\left[\tfrac{\phi_1}{\phi_1-z} - \tfrac{\phi_2}{\phi_2-z}\right]\\ &= \tfrac{1}{\phi_1-\phi_2}\left[\tfrac{1}{1-z/\phi_1} - \tfrac{1}{1-z/\phi_2}\right]\\ &= \tfrac{1}{\phi_1-\phi_2}\left(\left(\cdots [\tfrac{z}{\phi_1}]^n \cdots\right) - \left(\cdots [\tfrac{z}{\phi_2}]^n \cdots\right)\right)\end{aligned}$$

therefore

$$F_n = \frac{1}{\phi_1-\phi_2}\left(\frac{1}{\phi_1^n} - \frac{1}{\phi_2^n}\right) \qquad (v)$$

Let's simplify (v):

First, by brute force calculation, or by using the "known" relations between the roots of a 2nd degree equation, we find

$$\phi_1\phi_2 = -1, \qquad \phi_1 - \phi_2 = \sqrt{5}$$

so that (v) gives

$$\begin{aligned}F_n &= \tfrac{1}{\sqrt{5}}\left(\tfrac{\phi_2^n}{(\phi_1\phi_2)^n} - \tfrac{\phi_1^n}{(\phi_1\phi_2)^n}\right)\\ &= \tfrac{1}{\sqrt{5}}\left((-1)^n \tfrac{\left((1+\sqrt{5})/2\right)^n}{(-1)^n} - (-1)^n \tfrac{\left((1-\sqrt{5})/2\right)^n}{(-1)^n}\right)\\ &= \tfrac{1}{\sqrt{5}}\left(\left[\tfrac{1+\sqrt{5}}{2}\right]^n - \left[\tfrac{1-\sqrt{5}}{2}\right]^n\right)\end{aligned}$$

In particular, we find that

$$F_n = O\left(\left[\frac{1+\sqrt{5}}{2}\right]^n\right)$$

since

$$\left[\frac{1-\sqrt{5}}{2}\right]^n \to 0 \text{ as } n \to \infty$$

since $(1-\sqrt{5})/2$ is about -0.62.

That is, F_n grows exponentially with n, since $|\phi_2| > 1$.

Bibliography

[Dav65] M. Davis, *The undecidable*, Raven Press, Hewlett, NY, 1965.

[Hin78] P. G. Hinman, *Recursion-theoretic hierarchies*, Springer-Verlag, New York, 1978.

[Kle43] S.C. Kleene, *Recursive predicates and quantifiers*, Transactions of the Amer. Math. Soc. **53** (1943), 41–73, [Also in [Dav65], 255–287].

[Knu73] Donald E. Knuth, *The Art of Computer Programming; Fundamental Algorithms*, 2nd ed., vol. 1, Addison-Wesley, 1973.

[Kur63] A.G. Kurosh, *Lectures on General Algebra*, Chelsea Publishing Company, New York, 1963.

[Tou03a] G. Tourlakis, *Lectures in Logic and Set Theory, Volume 1: Mathematical Logic*, Cambridge University Press, Cambridge, 2003.

[Tou03b] ______, *Lectures in Logic and Set Theory, Volume 2: Set Theory*, Cambridge University Press, Cambridge, 2003.

[Tou08] ______, *Mathematical Logic*, John Wiley & Sons, Hoboken, NJ, 2008.