Leibniz Universität Hannover
Institut für theoretische Informatik

Bachelor Thesis

Hierarchies of recursive functions

Maurice Chandoo

February 21, 2013

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Dedicated to my dear grandmother
Anna-Maria Kniejska

Contents

1 Preface

I would like to give a more or less exhaustive overview on different kinds of recursive functions and hierarchies which characterize these functions by their computational power. For instance, it will become apparent that primitve recursion is more than sufficiently powerful for expressing algorithms for the majority of known decidable "real world" problems. This comes along with the hunch that it is quite difficult to come up with a suitable problem of which the characteristic function is not primitive recursive therefore being a possible candidate for exploiting the more powerful multiple recursion. At the end the Turing complete μ-recursion is introduced. It completes the characterization of functions in terms of recursion.

If not mentioned differently, the proofs in this thesis were written from scratch using the basic ideas of the cited sources.

In the course of writing this thesis I have gained insights beyond the actual goal of this work. The intensive analysis of the structure of different types of recursion has revealed to me a new way of thinking in terms of such recursions. Although, this is something that can be hardly phrased into definitions and theorems, I still hope some of these insights are conveyed by reading this work.

2 Class of primitve recursive functions $\mathcal{PR}$

When one talks about a recursive function in a general sense, it is likely to mean a function which is defined by its predecessing function values. Apparently, this somewhat vague idea leaves room for different concepts, namely primitive, course-of-value and nested recursion as introduced by [Pet34] who has also shown their equivalence. We start out by defining the class of primitive recursive functions and gradually introduce the more sophisticated variants.

2.1 Primitive Recursion class $\mathcal{PR}$

The following defintion is based on [Wag85]. The set of primitive recursive functions embodies mappings from $\mathbb{N}^k$ to $\mathbb{N}$ for arbitrary k. The basic families of functions contained by $\mathcal{PR}$ are:

Let $n, n_1, \ldots, n_k \in \mathbb{N}$

- Constant 0-function: $0() = 0$
  $0 \in \mathcal{PR}$
- Projection function: $\pi_i^k(n_1, ..., n_k) = n_i$
  $\pi_i^k \in \mathcal{PR}$ for all $k \geq 1$ and $1 \leq i \leq k$
  Note, π without indices denotes the set of all projection functions
- Successor function: $\mathrm{suc}(n) = n + 1$

Using the concept of composition and primitive recursion we inductively define the complete set $\mathcal{PR}$. To denote that function f has arity k we write f^k just as done above.

- Composition: let $f^m, g_1^k, \ldots, g_m^k \in \mathcal{PR}$, then it holds that
  $f(g_1(n_1, \ldots, n_k), \ldots, g_m(n_1, \ldots, n_k)) \in \mathcal{PR}$ for arbitrary $k, m \geq 1$
- Primitive recursion: let $g^k, h^{k+2} \in \mathcal{PR}$ and f^{k+1} is defined by:
  $$f(n, n_1, \ldots, n_k) = \begin{cases} g(n_1, \ldots, n_k) & , n = 0 \\ h(n-1, n_1, \ldots, n_k, f(n-1, n_1, \ldots, n_k)) & , \text{else} \end{cases}$$
  $f \in \mathcal{PR}$
  We abbreviate this as $f = \mathrm{REC}[g, h]$. Notice, the parameters $n_1, \ldots, n_k$ are constant over the course of recursion.

For convenience we introduce some shortcuts which are covered by our definition in a more lengthy way.

- Fictive variables: let $\{x_1, \ldots, x_l\} \subset \{x_1, \ldots, x_k\}$ and $f^k, g^l \in \mathcal{PR}$:
  If a function $f(x_1, \ldots, x_k)$ is expected, we can as well pass $g(x_1, \ldots, x_l)$ since the following holds:
  $\pi_1^k(g(x_1, \ldots, x_l), 0, \ldots, 0) = g(x_1, \ldots, x_l)$
- Multiple application: let $f^1 \in \mathcal{PR}$:
  $f^{(i)}(x) = f(f^{(i-1)}(x))$ for all $i \geq 1$
  $f^{(0)}(x) = \pi_1^1(x)$
- Arbitrary Constant: we can pass any constant $c \in \mathbb{N}$ as parameter since $c = \text{suc}^{(c)}(0)$

For the sake of a better understanding of how to express functions in $\mathcal{PR}$ the following basic functions will be explained in detail.

Additon

Let $\text{add}(n, a) = n + a$, then its primitive recursvie definition is

$$\text{add}(n, a) = \begin{cases} \pi_1^1(a) & , n = 0 \\ \text{suc}(\text{add}(n-1, a)) & , \text{else} \end{cases}$$

or for short

$$\text{add}(n, a) = \text{REC}[\pi_1^1, \text{suc}(\pi_3^3(x, y, z))]$$

This simply expresses the fact that $(n - 1 + a) + 1 = n + a$ and $0 + a = a$.

Subtraction

Due to the fact that $\mathcal{PR}$ operates on $\mathbb{N}$, we define subtraction as

$$\text{sub}(n, a) = n - a = \max\{0, a - n\}$$

First, we need a predeccesor function $\text{pre}(n) = max\{0, n - 1\}$.

$$\text{pre}(n) = \text{REC}[0, \pi_1^2(x, y)]$$

Looking at the order of parameters in the definition of primitive recursion, we see that $x = n - 1$. Now our subtraction can be stated analogously to addition as

$$\text{sub}(n, a) = \text{REC}[\pi_1^1, \text{pre}(\pi_3^3(x, y, z))]$$

Translated to a more readable form:

$$\text{sub}(n, a) = \begin{cases} a & , n = 0 \\ \text{pre}(\text{sub}(n-1, a)) & , \text{else} \end{cases}$$

Multiplication

By using addition and the obvious identity $n * a = (n-1) * a + a$ we define multiplication:

$$\text{mult}(n, a) = n * a = \text{REC}[0, \text{add}(\pi_3^3(x, y, z)), \pi_2^3(x, y, z))]$$

Again, by looking at the values passed to the function h in the definition of primitive recursion we can deduce that $x = n-1, y = a, z = \text{mult}(n-1, a)$ and can rephrase the above function as:

$$\text{REC}[0, \text{add}(\text{mult}(n-1, a), a)]$$

To spare us from looking up the definition each time we will use the latter form from this point on.

Binaryzation

$$\text{sg}(n) = \text{REC}[0, 1](n) = \begin{cases} 0 & , n = 0 \\ 1 & , \text{else} \end{cases}$$

Inverted Binaryzation

$$\overline{\text{sg}}(n) = \text{REC}[1, 0](n) = \begin{cases} 1 & , n = 0 \\ 0 & , \text{else} \end{cases}$$

2.2 Course-of-value Recursion class $\mathcal{PR}_{cov}$

In the definition of $\mathcal{PR}$ we restricted ourselves to recursion that only used its immediate predeccesor function value $f(n-1)$. Instead, we would like to use any possible predecessors; for a call $f(n)$ these are $f(0), f(1), \ldots, f(n-1)$. To define the set of course-of-value recursions $\mathcal{PR}_{cov}$ we will use the same definition as for $\mathcal{PR}$ but substitute every occurence of $\mathcal{PR}$ with $\mathcal{PR}_{cov}$ and primitve recursion with

- Course-of-value recursion: let $g^k, \mu_1^{k+1}, \ldots, \mu_c^{k+1}, h^{k+1+c} \in \mathcal{PR}_{cov}$ and f^{k+1} is defined by:

$$f(n, n_1, \ldots, n_k) = \begin{cases} g(n_1, \ldots, n_k) & , n = 0 \\ h(n-1, n_1, \ldots, n_k, & \\ \quad f(\mu_1, n_1, \ldots, n_k), & \\ \qquad \vdots & , \text{else} \\ \quad f(\mu_c, n_1, \ldots, n_k)) & \end{cases}$$

$\mu_i = \mu_i(n-1, n_1, \ldots, n_k) \leq n-1$ for all $1 \leq i \leq c$
$f \in \mathcal{PR}_{cov}$ for all $c, k \in \mathbb{N}$
Notice, the parameters $n_1, \ldots, n_k$ are constant over the recursion.

So, with this kind of recursion we are now able to refer to a fixed number c of arbitrary predecessors.

Lemma 2.1

$\mathcal{PR} \subseteq \mathcal{PR}_{cov}$
Primitive recursion is a special case of course-of-value recursion with $c = 1$ and $\mu_1 = \pi_1^{k+1}$

□

2.3 Nested Recursion class $\mathcal{PR}_{nes}$

Yet another idea for extending $\mathcal{PR}$ is to allow a change of parameters over the course of recursion. There was a remark for both previous types of recursions which explicitly prohibited this. We define this set $\mathcal{PR}_{nes}$ in the same fashion as $\mathcal{PR}_{cov}$ with the help of $\mathcal{PR}$ and instead of primitive recursion allow

- Nested recursion: let $g^k, h^{k+2}, \lambda_i^{k+1} \in \mathcal{PR}_{nes}$ and f^{k+1} is defined by:
  $$f(n, n_1, \ldots, n_k) = \begin{cases} g(n_1, \ldots, n_k) & , n = 0 \\ h(n-1, n_1, \ldots, n_k, f(n-1, \lambda_1, \ldots, \lambda_k)) & , \text{else} \end{cases}$$
  $\lambda_i = \lambda_i(n-1, n_1, \ldots, n_k)$ for all $1 \leq i \leq k$
  $f \in \mathcal{PR}_{nes}$ for all $k \in \mathbb{N}$

An interesting case of nested recursion is obtained by using f as definition for some λ_i. Let us construct an example for this case:

$$f(n, a) = \begin{cases} g(a) & , n = 0 \\ h(n-1, a, f(n-1, f(n-1, a))) & , \text{else} \end{cases}$$

It is recognizable that such a function will always terminate as each recursive step will only call predecessors. How about defining a nested recursion such that some $\lambda_i(n, n_1, \ldots, n_k) = f(n+1, n_1, \ldots, n_k)$? The result is a function which will induce an infinite amount of function calls to itself thus never terminating. We will forbid any kind of nested function definition that leads to such a non-terminating tragedy.

Lemma 2.2

$\mathcal{PR} \subseteq \mathcal{PR}_{nes}$
Primitive recursion is a special case of nested recursion with $\lambda_i = \pi_{i+1}^{k+1}$ for all $1 \leq i \leq k$

□

2.4 Recursive Depth

We define the recursive depth $|f|_{rd}$ of a function $f \in \mathcal{PR}$ inductively as:

- if $f \in \{\text{suc}, 0\} \cup \pi$ then $|f|_{rd} = 0$
- if $f = g(h_1, \ldots, h_k)$ then $|f|_{rd} = \max\limits_{1 \leq i \leq k} \{|h_i|_{rd}\}$
  for $g, h_1, \ldots, h_k \in \mathcal{PR}$
- if $f = \text{REC}[g, h]$ then $|f|_{rd} = 1 + \max\{|g|_{rd}, |h|_{rd}\}$
  for $g, h \in \mathcal{PR}$

The recursive depth for a course-of-value recursion $f \in \mathcal{PR}_{cov}$ is defined as:

- $|f|_{rd} = 1 + max\{|g|_{rd}, |h|_{rd}, |\mu_1|_{rd}, \ldots, |\mu_c|_{rd}\}$
  if f is a recursion given by $g, h, \mu_1 \ldots, \mu_c \in \mathcal{PR}_{cov}$

The recursive depth for a nested recursion $f \in \mathcal{PR}_{nes}$ is defined as:

- $|f|_{rd} = 1 + \max\{|g|_{rd}, |h|_{rd}, |\lambda_1|_{rd}, \ldots, |\lambda_k|_{rd}\}$
  if f is a recursion given by $g, h, \lambda_1, \ldots, \lambda_k \in \mathcal{PR}_{nes}$

Beware of the same symbol name of recursive depth for different recursion types. This does not imply that two functions of different recursion types which represent the same mapping have equal recursive depth.

2.5 Recursive Relations

[Pet34] has used recursive relations to demonstrate with ease that certain number-theoretic functions, which we need to use later on, are primitive recursive. We extend this idea to express primitive recursive functions in a declarative manner.

- A relation R^k is a subset of $\mathbb{N}^k$

- The characteristic function r of R^k is given by:

  $$r(n_1, \ldots, n_k) = [\![e]\!] = \begin{cases} 1 & , (n_1, \ldots, n_k) \in R^k \\ 0 & , \text{else} \end{cases}$$

  with e being a boolean expression such that $R = \left\{(n_1, \ldots, n_k) \in \mathbb{N}^k \mid e\right\}$

- A relation R^k is recursive, iff its characteristic function $r \in \mathcal{PR}$

- A function b is a binary function, iff its target domain is $\{0, 1\}$

- All characteristic function of recursive relations are binary functions

- Logical conjunction $\wedge$ of two binary functions b_1^k, b_2^l:

  $$b_1 \wedge b_2 = \begin{cases} 1 & , b_1 + b_2 = 2 \\ 0 & , \text{else} \end{cases}$$

  with $b_1 := b_1(n_{1,1}, \ldots, n_{1,k})$ and $b_2 := b_2(n_{2,1}, \ldots, n_{2,l})$

- Logical negation $\neg$ of a binary function b^k:

  $$\neg b(n_1, \ldots, n_k) = \begin{cases} 1 & , b(n_1, \ldots, n_k) = 0 \\ 0 & , \text{else} \end{cases}$$

- For any binary function $b^k \in \mathcal{PR}$ it holds that $\neg b^k \in \mathcal{PR}$:
  $\neg b(n_1, \ldots, n_k) = \overline{\text{sg}}(b(n_1, \ldots, n_k))$

- Equality $n = m$ is a recursive relation since its characteristic function r is given by $r(n, m) = \overline{\text{sg}}((n - m) + (m - n))$

- For any recursive relation R^k its complement $\overline{R^k} = \mathbb{N}^k \setminus R^k$ is recursive as well with resp. characteristic functions $r, \overline{r}$:
  $\overline{r}(n_1, \ldots, n_k) = \neg r(n_1, \ldots, n_k)$

- For any two binary functions $b_1^k, b_2^l \in \mathcal{PR}$ it holds that $b_1 \wedge b_2 \in \mathcal{PR}$ since it can be stated as recursive relation $b_1 \wedge b_2 = 1 \Leftrightarrow b_1 + b_2 = 2$ with $b_1 := b_1(n_{1,1}, \ldots, n_{1,k})$ and $b_2 := b_2(n_{2,1}, \ldots, n_{2,l})$

- Binary functions in $\mathcal{PR}$ are closed under composition of logical conjunction and negation

- Greather than $n > m$ is a recursive relation since its characteristic function r is given by $r(n, m) = \text{sg}(n - m)$

- For any binary function $b^{k+1} \in \mathcal{PR}$ and $y^k \in \mathcal{PR}$ and the function $f(n_1, \ldots, n_k) = \max\{0 \leq x \leq y(n_1, \ldots, n_k) \mid b(x, n_1, \ldots, n_k) = 1 \vee x = 0\}$ it holds that $f \in \mathcal{PR}$. f can be expressed as follows:
  $f(n_1, \ldots, n_k) = f'(y(n_1, \ldots, n_k), n_1, \ldots, n_k)$
  $f'(n, n_1, \ldots, n_k) = \mathrm{REC}[0, g](n, n_1, \ldots, n_k)$
  $g(n-1, n_1, \ldots, n_k, f'_{pre}) = \mathrm{REC}[f'_{pre}, n](b(n, n_1, \ldots, n_k), n, f'_{pre})$

- For any binary function $b^{k+1} \in \mathcal{PR}$ and $y^k \in \mathcal{PR}$ and the function $f(n_1, \ldots, n_k) = \min\{0 \leq x \leq y \mid b(x, n_1, \ldots, n_k) = 1 \vee x = y\}$ with $y := y(n_1, \ldots, n_k)$ it holds that $f \in \mathcal{PR}$. f can be expressed as:
  $f(n_1, \ldots, n_k) = f'(y, n_1, \ldots, n_k, y)$
  $f'(n, n_1, \ldots, n_k, \overline{n}) = \mathrm{REC}[\overline{n}, g](n, n_1, \ldots, n_k, \overline{n})$
  $g(n-1, n_1, \ldots, n_k, \overline{n}, f'_{pre}) = \mathrm{REC}[f'_{pre}, \overline{n}-n](b(\overline{n}-n, n_1, \ldots, n_k), n, \overline{n}, f'_{pre})$

- With the above two items we have obtained an easy and compact way to define functions in $\mathcal{PR}$, for that we write:
  $f(n_1, \ldots, n_k) = [\![\max x \leq y(n_1, \ldots, n_k) : b(x, n_1, \ldots, n_k)]\!]$
  $f(n_1, \ldots, n_k) = [\![\min x \leq y(n_1, \ldots, n_k) : b(x, n_1, \ldots, n_k)]\!]$

For the next sections prime power decomposition is important, so we will show that the neccessary functions lie within $\mathcal{PR}$ with the help of recursive relations.

- $\mathrm{pow}(n, a) := a^n = \mathrm{REC}[1, \mathrm{mult}(a, \mathrm{pow}(n-1, a))]$

- $\mathrm{fac}(n) := n! = \mathrm{REC}[1, \mathrm{mult}(n, \mathrm{fac}(n-1))]$

- $\mathrm{div}(a, b) := \begin{cases} 0 & , b = 0 \\ \lfloor \frac{a}{b} \rfloor & , \text{else} \end{cases}$
  $\mathrm{div}(a, b) = [\![\max c \leq a : a \geq b * c]\!]$

- $\mathrm{dvsr}(a, b) := a|b = \begin{cases} 1 & , a \text{ is a divisor of } b \\ 0 & , \text{else} \end{cases}$
  $\mathrm{dvsr}(a, b) = \mathrm{REC}[1, \mathrm{dvsr}'(a, b)](b, a)$
  $\mathrm{dvsr}'(a, b) = \mathrm{sg}([\![\max c \leq b : b = a * c]\!])$

- $\mathrm{isprim}(n) := \begin{cases} 1 & , n \text{ is prime} \\ 0 & , \text{else} \end{cases}$
  $\mathrm{isprim}(n) = \overline{\mathrm{sg}}([\![\max c \leq n-1 : c|n \wedge c > 1]\!])$

- $\mathrm{p}(n) := p_n = \begin{cases} 2 & , n = 0 \\ \min\{c \in \mathbb{N} \mid p_{n-1} < c \wedge c \text{ is prime}\} & , \text{else} \end{cases}$

  $\mathrm{p}(n) = \mathrm{REC}[2, \mathrm{p}'(n-1, \mathrm{p}(n-1))]$

  $\mathrm{p}'(n, \mathrm{p}_{pre}) = [\![\min c \leq \mathrm{p}_{pre}! + 1 : \mathrm{isprim}(c) \wedge c > \mathrm{p}_{pre}]\!]$

- $\varpi(n, a) := \begin{cases} 0 & , a = 0 \\ \max\{c \in \mathbb{N} \mid p_n^c | a\} & , \text{else} \end{cases}$

  $\varpi(n, a) = [\![\max c \leq a : p_n^c | a]\!]$

To define the sequence of prime numbers p_n as primitve recursion we have used an upper boundary for the difference between two neighboring prime numbers, which we will proof now.

Lemma 2.3

$\forall$ prime p $\exists$ prime $p' : p < p' \leq p! + 1$
Assume that this is not true, which implies $p+1, \ldots, p!+1$ are not prime. Let y be a prime divisor of $p!+1 \Rightarrow y < p+1 \Rightarrow y|(p!+1) \wedge y|p!$ $\Rightarrow \forall a, b \in \mathbb{N} : y|(a(p!+1) + b * p!) \Rightarrow y|1 \; \lightning$

□

Definition by case

For $g, h_1, \ldots, h_c \in \mathcal{PR}$ and recursive relations $R_1^2, \ldots, R_c^2$ with resp. characteristic functions $r_1, \ldots, r_c$ we say f is a definition by case if

$$f(n, a) = \begin{cases} g(a) & , n = 0 \\ h_1(n-1, a, f(n-1, a)) & , r_1(n-1, a) = 1 \\ \vdots & \\ h_c(n-1, a, f(n-1, a)) & , r_c(n-1, a) = 1 \\ 0 & , \text{else} \end{cases}$$

and if more than one case condition is true for some (n, a) use the first case for which the condition is fulfilled, e.g. if $r_i(n, a) = r_j(n, a) = 1$ for $i < j$ then h_i is applied.

Lemma 2.4

For a definition by case f given by $g, h_1, \ldots, h_c, r_1, \ldots, r_c \in \mathcal{PR}$ it holds that $f \in \mathcal{PR}$.

$$f(n, a) = \mathrm{REC}[g, h]$$

$$h(n, a, f_{pre}) = \sum_{i=1}^{c} \left[\!\!\left[\bigwedge_{1 \leq j < i} r_j(n, a) = 0 \wedge r_i(n, a) = 1 \right]\!\!\right] * h_i(n, a, f_{pre})$$

with the empty conjunction set to true.

□

2.6 Equivalence of $\mathcal{PR}$ and $\mathcal{PR}_{cov}$

Before we proof that every course-of-value recursion can be written as primitive recursive function, we will show that it can be restricted to the use of only one non-recursive parameter without loss of generality.

Lemma 2.5

For $k \in \mathbb{N}$, $f^{k+1}, f'^2 \in \mathcal{PR}_{cov}$:

$$\forall f^{k+1} \ \exists f'(n, a) : f(n, n_1, \ldots, n_k) = f'(n, p_1^{n_1} * \cdots * p_k^{n_k})$$

Define f' exactly the same as f and replace any occurence of n_i with $\varpi(i, a)$ for all $1 \leq i \leq k$.

□

Theorem 2.6

$\mathcal{PR}_{cov} \subseteq \mathcal{PR}$
This will be proven by showing that any course-of-value recursion can be rewritten as primitve recursion by induction over the recursive depth. Our inductive statement is

$$\forall f(n, a) \in \mathcal{PR}_{cov} \text{ with } |f|_{rd} \leq \ell \ \exists f'(n, a) \in \mathcal{PR} : f = f'$$

For the base case $\ell = 0$ no recursion can occur, so the inductive hypothesis obviously holds. For the inductive step $\ell + 1$ we have a function $f \in \mathcal{PR}_{cov}$ with $|f|_{rd} = \ell + 1$. This f is a course-of-value recursion[1] parameterized by the functions $g^1, h^{2+c}, \mu_1^2, \ldots, \mu_c^2$ which lie within $\mathcal{PR}$ as we can deduce from our inductive hypothesis. Now we construct a $f'(n, a)$ such that $f'(n, a) = f(n, a)$:

$$f'(n, a) = \varpi(n, f''(n, a))$$

$$f''(n, a) = \begin{cases} p_0^{g(a)} & , n = 0 \\ h'(n-1, a, f''(n-1, a)) & , \text{else} \end{cases}$$

[1] Actually, it could be a substitution given by some functions for which we would translate each one separately

$$h'(n-1,a,f''_{pre}) = f''_{pre} * \text{pow}(h(n-1,a,\varpi(\mu_1,f''_{pre}),\ldots,\varpi(\mu_c,f''_{pre})),p_n)$$

with $\mu_i = \mu_i(n-1,a)$ for all $1 \leq i \leq c$.

□

Proof of Correctness

Our claim is that $f = f'$ which, again, is proven by induction over n. Our base case $n = 0$:

$$f(0,a) = g(a) = \varpi(0,p_0^{g(a)}) = \varpi(0,f''(0,a)) = f'(0,a)$$

The inductive step $n+1$:

$$f(n+1,a) = h(n,a,f(\mu_1(n,a),a),\ldots,f(\mu_c(n,a),a))$$

$$f'(n+1,a) = \varpi(n+1,f''(n+1,a))$$

$$f''(n+1,a) = h'(n,a,f''(n,a))$$

$$f'(n+1,a) = h(n,a,\varpi(\mu_1(n,a),f''(n,a)),\ldots,\varpi(\mu_c(n,a),f''(n,a)))$$

This leads to the question
$h(n,a,f(\mu_1(n,a),a),\ldots,f(\mu_c(n,a),a)) \stackrel{?}{=}$
$h(n,a,\varpi(\mu_1(n,a),f''(n,a)),\ldots,\varpi(\mu_c(n,a),f''(n,a)))$

This holds true, iff

$$f(\mu_i(n,a),a) = \varpi(\mu_i(n,a),f''(n,a))\ \forall\ 1 \leq i \leq c$$

Due to our inductive hypothesis we can assume that

$$f(j,a) \stackrel{!}{=} f'(j,a) = \varpi(j,f''(j,a))\ \forall\ j < n+1$$

and the fact that f'' is preserving each predecessor due to h' meaning

$$\varpi(j,f''(j,a)) = \varpi(j,f''(l,a))\ \forall\ l \geq j$$

which conclusively proves our claim since

$$\mu_1(n,a),\ldots,\mu_c(n,a),j \in \{0,\ldots,n\}$$

□

2.7 Equivalence of $\mathcal{PR}$ and $\mathcal{PR}_{nes}$

As for $\mathcal{PR}_{cov}$, we will show that nested recursion can be restricted to the use of only one non-recursive parameter without loss of generality.

Lemma 2.7

For $k \in \mathbb{N}, f^{k+1}, f'^2 \in \mathcal{PR}_{nes}$:

$$\forall f^{k+1}\ \exists f'(n,a) : f(n, n_1, \ldots, n_k) = f'(n, p_1^{n_1} * \cdots * p_k^{n_k})$$

If $|f|_{rd} = 0$ define f' exactly the same as f and replace any occurence of n_i with $\varpi(i,a)$ for all $1 \leq i \leq k$. If f is a nested recursion parameterized by $g^k, h^{k+2}, \lambda_1^{k+1}, \ldots, \lambda_k^{k+1}$ we use an inductive argument over the recursive depth such that $g, h, \lambda_1, \ldots, \lambda_c$ can be assumed to be given. f' can be written as:

$$f'(n,a) = \begin{cases} g(\varpi(1,a), \ldots, \varpi(k,a)) & , n = 0 \\ h(n-1, \varpi(1,a), \ldots, \varpi(k,a), & \\ \quad f'(n-1, \prod_{i=1}^{k} p_i^{\lambda_i(n-1, \varpi(1,a), \ldots, \varpi(k,a))})) & , \text{else} \end{cases}$$

□

Theorem 2.8

$\mathcal{PR}_{nes} \subseteq \mathcal{PR}$
Again, we will use an inductive argument over the recursive depth. Our inductive statement is

$$\forall f(n,a) \in \mathcal{PR}_{nes} \text{ with } |f|_{rd} \leq \ell\ \exists f'(n,a) \in \mathcal{PR} : f = f'$$

For the base case $\ell = 0$ no recursion can occur, so the inductive hypothesis obviously holds. For the inductive step $\ell + 1$ we have a function $f \in \mathcal{PR}_{nes}$ with $|f|_{rd} = \ell + 1$. This f is a nested recursion[2] parameterized by the functions g^1, h^3, λ^2 which lie within $\mathcal{PR}$ as we can deduce from our inductive hypothesis. Now we construct a $f'(n,a)$ such that $f' = f$:

$$f'(n,a) = f''(n,a,n)$$

$$f''(n,a,\overline{n}) = \begin{cases} g(a_{\overline{n}}^{\overline{n}}) & , n = 0 \\ h(n-1, a_{\overline{n}}^{\overline{n}-n}, f''(n-1,a,\overline{n})) & , \text{else} \end{cases}$$

[2] same case as in the footnote of Theorem 2.6

$$w(n, a, \overline{n}) = a^n_{\overline{n}} = \begin{cases} a & , n = 0 \\ \lambda(\overline{n} - n, w(n-1, a, \overline{n})) & , \text{else} \end{cases}$$

□

Proof of Correctness

For a nested recursion f^2 parameterized by g^1, h^3, λ^2 we formalize the concept of calling its predecessor:

$$f(n, a) \to f(n-1, b) \Leftrightarrow \lambda(n-1, a) = b$$

The next step is to proof that the change of parameter during the nested recursion is given by the helper function w.

$$\forall\, 1 \leq m \leq n : f(m, a_n^{n-m}) \to f(m-1, a_n^{n-m+1})$$

$$f(m, a_n^{n-m}) = h(m-1, a_n^{n-m}, f(m-1, \lambda(m-1, a_n^{n-m}))) \Rightarrow$$
$$f(m, a_n^{n-m}) \to f(m-1, \lambda(m-1, a_n^{n-m}))$$
$$\lambda(m-1, a_n^{n-m}) = w(n-m+1, a, n) = a_n^{n-m+1}$$

Finally, we can begin to proof our claim $f(n, a) = f'(n, a)$ by induction. The base case $n = 0$:

$$f(0, a) = g(a) = g(a_0^0) = f''(0, a, 0) = f'(0, a)$$

For the case $n+1$ we have to carry out another induction within our current one:

$$\begin{array}{ccccc} f(n+1, a_{n+1}^0) & \to & \cdots & \to & f(0, a_{n+1}^{n+1}) \\ \| & & \| & & \| \\ f''(n+1, a, n+1) & \to & \cdots & \to & f''(0, a, n+1) \end{array}$$

In other words, we show the following

$$\forall\, 0 \leq m \leq n+1 : f(m, a_{n+1}^{n+1-m}) = f''(m, a, n+1)$$

For the case $m = 0$:

$$f(0, a_{n+1}^{n+1}) = g(a_{n+1}^{n+1}) = f''(0, a, n+1)$$

For the case $0 < m+1 \leq n+1$:

$$f(m+1, a_{n+1}^{n-m}) = h(m, a_{n+1}^{n-m}, f(m, \lambda(m, a_{n+1}^{n-m})))$$

$$f''(m+1,a,n+1) = h(m, a_{n+1}^{n-m}, f''(m,a,n+1))$$
$$f(m, \lambda(m, a_{n+1}^{n-m})) = f(m, a_{n+1}^{n+1-m}) \stackrel{*}{=} f''(m,a,n+1)$$

For (*) we used our induction hypothesis. We finalize our proof by connecting f' to f and f'':

$$f(n+1,a) = f'(n+1,a) = f''(n+1,a,n+1)$$

□

A nested recursion with $\lambda(n,a) = f(n,a)$ can be calculated by successively evaluating $f(0,a), f(0,f(0,a)), f(1,a), f(1,f(1,a)), \ldots, f(n,a), f(n,f(n,a))$. This can be expressed as definition by case using the parity of the recursive variable.

Corollary 2.9

$\mathcal{PR}$= $\mathcal{PR}_{cov}$= $\mathcal{PR}_{nes}$

2.8 Loop programs

Loop programs are a well known concept in computer science which have the nice property of always terminating. We will show that they are exactly as powerful as primitive recursion and furthermore can be seen as nested recursion in disguise[3]. We define Loop programs as in [Vol11].

Syntactical components

- Variables: $x_1, x_2, x_3, \ldots$
- Constants: $0, 1, 2, \ldots$
- Keywords: LOOP, DO, END
- Other symbols: '+', '−', ':=', ';'

[3]This qualifies as a reason as to why a formal analysis of Loop programs is stressful in general refering to the previous proof of correctness

Syntax

- If x_i, x_j are variables and c is a constant, then "$x_i := x_j + c$;" and "$x_i := x_j - c$;" are Loop programs
- If P_1, P_2 are Loop programs so is "P_1 P_2"
- If P is a Loop program and x_i is a variable, then "LOOP x_i DO P END;" is a Loop program

Semantics

A Loop program P calculates a function $f_P : \mathbb{N}^k \mapsto \mathbb{N}$ with the input $n_1, \ldots, n_k$. It does that by setting the values of the k-first variables to its respective input $x_i = n_i$ at the beginning of the program. All other variables used in P are set to 0 initially.

- After execution of "$x_i := x_j + c$;" the value contained by $x_i = x_j + c$
- After execution of "$x_i := x_j - c$;" the value contained by $x_i = \max\{0, x_j - c\}$
- "$P1$ $P2$" indicates to execute $P1$ and then execute $P2$ with all the values of variables being the same as at the end of $P1$ when executing $P2$.
- "LOOP x_i DO P END;" means to execute P x_i times. Possible changes of the value of x_i during execution of the loop body P do not affect the number of iterations

At the end of the execution of a Loop program the value of the variable x_1 is returned. We call the set of all Loop programs LOOP.

Loop nesting depth

Given an arbitrary Loop program P, we define its Loop nesting depth $|P|_{ld}$ as follows:

- if $P =$ "$x_i := x_j + c$;" then $|P|_{ld} = 0$
- if $P =$ "$x_i := x_j - c$;" then $|P|_{ld} = 0$
- if $P =$ "P_1 P_2" then $|P|_{ld} = max\{|P_1|_{ld}, |P_2|_{ld}\}$
- if $P =$ "LOOP x_i DO P_l END;" then $|P|_{ld} = |P_l|_{ld} + 1$

Theorem 2.10

LOOP $\subseteq \mathcal{PR}$
A Loop program P which uses k input variables and m variables in total, can be described as a collection of functions $f_1, \dots f_m$ which calculate the value of their respective variable after the execution of the program. The function f_P calculated by P is given by

$$f_P(n_1, \dots, n_k) = f_1(n_1, \dots, n_k, 0, \dots, 0)$$

To show this we prove:

$$\forall\, P \in \text{LOOP and } |P|_{ld} \leq \ell : f_P \in \mathcal{PR}$$

with f_P being the function calculated by the Loop program P.
For the base case $\ell = 0$ we argue that a Loop program with nesting depth 0 can only consist of a finite sequence of arithmetic operations of either the form "$x_i := x_j + c$" or "$x_i := x_j - c$". The values of these assignments can be simply calculated by

$$x_j + c = \text{suc}^{(c)}(x_j) \quad \text{and} \quad x_j - c = \text{pre}^{(c)}(x_j)$$

So, such an operation is determined by i, j, c and the operator '+' or '−'. Let A be the set which contains all possible arithmetic operations for a program which uses m variables:

$$A = \left\{(i, j, \circ) \mid 1 \leq i, j \leq m \text{ and } \circ \in \left\{f^{(c)} \mid f \in \{\text{pre}, \text{suc}\} \text{ and } c \in \mathbb{N}\right\}\right\}$$

For any Loop program given by a non-empty sequence $((i_1, j_1, \circ_1), \dots, (i_l, j_l, \circ_l))$ of length l we can determine $f_1, \dots, f_m$ as follows:

for $1 \leq p \leq m$, $1 \leq q \leq l$:

$$f_{p,0} = \pi^n_p$$

$$f_{p,q} = \begin{cases} \circ_q(f_{j_q,q-1}) & , i_q = p \\ f_{p,q-1} & , i_q \neq p \end{cases}$$

$$f_p(x_1, \dots, x_m) = f_{p,l}(x_1, \dots, x_m)$$

To illustrate the idea behind this we will compute the function table for a concrete example.

Line i	Program	$f_{1,i}$	$f_{2,i}$	$f_{3,i}$
0		x_1	x_2	x_3
1	$x_1 = x_2 + 3;$	$\text{suc}^{(3)}(x_2)$	x_2	x_3
2	$x_2 = x_1 - 2;$	$\text{suc}^{(3)}(x_2)$	$\text{pre}^{(2)}(\text{suc}^{(3)}(x_2))$	x_3
3	$x_3 = x_2 + 4;$	$\text{suc}^{(3)}(x_2)$	$\text{pre}^{(2)}(\text{suc}^{(3)}(x_2))$	$\text{suc}^{(4)}(\text{pre}^{(2)}(\text{suc}^{(3)}(x_2)))$
4	$x_1 = x_1 - 1$	$\text{pre}(\text{suc}^{(3)}(x_2))$	$\text{pre}^{(2)}(\text{suc}^{(3)}(x_2))$	$\text{suc}^{(4)}(\text{pre}^{(2)}(\text{suc}^{(3)}(x_2)))$

Note, for better readability the actual variables were used instead of the corresponding projection functions

For the inductive step $\ell + 1$:

$$P = \text{"LOOP } x_i \text{ DO } P_l \text{ END;"}$$

we show

$$f_P \in \mathcal{PR} \text{ for } |P_l|_{ld} \leq \ell$$

Due to the inductive hypothesis there is a collection of primitive recursive functions which calculate the value of each of the m variables after the execution of P_l. Let these functions be $f_{l,1}, \ldots, f_{l,m}$ then we obtain f_P as nested recursion:

$$f_P(x_1, \ldots, x_m) = h_1(x_i, x_1, \ldots, x_m)$$

$$h_p(n, x_1, \ldots, x_m) = \begin{cases} x_p & , n = 0 \\ h(n-1, f_{l,1}(x_1, \ldots, x_m), \ldots, f_{l,m}(x_1, \ldots, x_m)) & , \text{else} \end{cases}$$

It is noteworthy that a nested recursion is semantically equivalent to a LOOP instruction in the following sense:

$$x_j^k := \text{value of } x_j \text{ after execution of "LOOP } k \text{ DO } P_l \text{ END;"}$$

$$x_j^k = h_j(k, x_1, \ldots, x_m) \; \forall \, 1 \leq j \leq m, k \geq 0$$

Additionally, $P' = \text{"}P_1; P_2\text{"}$ is in $\mathcal{PR}$. P_1, P_2 can be Loop programs as in the inductive step or programs with nesting depth smaller than $n + 1$. By reason of this restriction there are primitive recursive functions for P_1 and P_2. Let us call them $f_{1,1}, f_{2,1} \ldots, f_{1,m}, f_{2,m}$ with m being the maximum count of variables used in either P_1 or P_2. The function $f_{P'}$ calculating P' is given by:

$$f_{P'}(x_1, \ldots x_m) = f_{2,1}(f_{1,1}(x_1, \ldots x_m), \ldots, f_{1,m}(x_1, \ldots x_m))$$

□

Theorem 2.11

$\mathcal{PR} \subseteq$ LOOP
First, we introduce an extension of Loop programs:
If $x_i, x_{a_1}, \ldots, x_{a_n}$ are variables and $P \in$ LOOP with f_P being the corresponding function then "$x_i := f_P(x_{a_1}, \ldots, x_{a_n})$;" means that after the execution of this line $x_i = f_P(x_{a_1}, \ldots, x_{a_n})$. Furthermore the evaluation of f_P has no meaningful side effects to the calling program. This can be accomplished by appropriately renaming the variables occuring in P.

$$\forall f \in \mathcal{PR} \text{ with } |f|_{rd} \leq \ell \; \exists P \in \text{LOOP s.t. } f_P = f$$

For any set of functions $h_0^k, h_1^n, \ldots, h_k^n \in \mathcal{PR}$ which can be calculated by Loop programs $P_{h_0}, \ldots, P_{h_k}$ the composition $f^n = h_0(h_1, \ldots, h_k)$ can also be expressed as Loop program. P_f is given by:

1 $x_{h,1} := h_1(x_1, \ldots x_n)$;
2 $\vdots$
3 $x_{h,k} := h_k(x_1, \ldots x_n)$;
4 $x_1 := h_0(x_{h,1}, \ldots, x_{h,k})$;

The base case $\ell = 0$ is trivial as it just requires to express constant 0-function, successor and projection function as Loop programs. For the inductive step $\ell + 1$ we show:

$$f^{n+1} = \text{REC}[g^n, h^{n+2}] \text{ with } |g|_{rd}, |h|_{rd} \leq \ell \text{ is calculated by } P_f$$

For that, the induction hypothesis grants us that g, h can be calculated by some Loop programs. P_f is given by:

1 $x_f := g(x_2, \ldots, x_n)$;
2 **LOOP** x_1 **DO**
3 $\quad x_f := h(x_{cnt}, x_2, \ldots, x_n, x_f)$;
4 $\quad x_{cnt} := x_{cnt} + 1$;
5 **END**;
6 $x_1 = x_f$;

□

Corollary 2.12

$\mathcal{PR} =$ LOOP

2.9 Grzegorczyk Hierarchy

This hierarchy classifies primitive recursive functions by their growth. For its definition we follow [Wag85] with a small adjustment. We start by recursively defining a series of boundary functions:

$$B_0(n,a) = n+1\ ,\ B_1(n,a) = n+a\ ,\ B_2(n,a) = n*a$$

$$\forall n \geq 2 : B_{n+1}(n,a) = \mathrm{REC}[1, B_n(a, B_{n+1}(n-1,a))]$$

All boundary functions are primitive recursive and $|B_i|_{rd} = i$ holds for all $i \geq 0$. We define bounded primitive recursion for $g^1, h^3, k^2 \in \mathcal{PR}$ [4] as:

$$f^2 = \mathrm{BPR}[g,h,k] \overset{def}{\Leftrightarrow} f = \mathrm{REC}[g,h] \wedge f \leq k$$

Now, the Grzegorczyk Hierarchy $\mathcal{E}^i$ for all $i \geq 0$ is defined as:

- $B_i \in \mathcal{E}^i$
- if $|f|_{rd} = 0$ then $f \in \mathcal{E}^i$
- if $f(n,a) = g(h(n,a), k(n,a))$ then $f \in \mathcal{E}^i$ for $g,h,k \in \mathcal{E}^i$
- if $f = \mathrm{BPR}[g,h,k]$ then $f \in \mathcal{E}^i$ for $k \in \mathcal{E}^i$

Lemma 2.13

$\forall\, i \geq 0 : \mathcal{E}^i \subseteq \mathcal{E}^{i+1}$
To prove this we show

$$\forall\, i \geq 0 : B_i \in \mathcal{E}^{i+1} \Leftrightarrow\ \exists\, m \in \mathcal{E}^{i+1} : B_i \leq m$$

With that we can conclude that anything used to construct a function which resides in $\mathcal{E}^i$ can also be utilized in $\mathcal{E}^{i+1}$ thus justifying our statement. The two special cases:

- $i = 0$: $B_0(n,a) \leq B_1(n+1,a) \Leftrightarrow n+1 \leq n+1+a$
- $i = 1$: $B_1(n,a) \leq B_2(n{+}1, a{+}1) \Leftrightarrow n{+}a \leq (n{+}1)(a{+}1) = na{+}n{+}a{+}1$

[4] [Wag85] has defined BPR for $g,h,k \in \mu\mathcal{R}$ which we have restricted to $\mathcal{PR}$ for the sake of simplicity. This is justified by the result in [Grz53] that shows that the Grzegorczyk Hierarchy coincides with $\mathcal{PR}$

The common case using induction:
$(\star)\ \forall i \geq 3 : B_{i-1}(n,a) \leq B_i(n,a)$
$i = 3 : B_2 \leq B_3 \Leftrightarrow a * n \leq a^n \ \forall\, a, n \geq 0$
$i + 1 > 3 : B_i \leq B_{i+1}$
$(\diamond)\ \forall n \geq 0 : B_i(n,a) \leq B_{i+1}(n,a)$
$n = 0 : B_i(0,a) \leq B_{i+1}(0,a) \Leftrightarrow 1 \leq 1$
$n + 1 : B_i(n+1,a) \leq B_{i+1}(n+1,a)$
$\Leftrightarrow B_{i-1}(a, B_i(n,a)) \leq B_i(a, B_{i+1}(n,a))$
$\overset{(\diamond)}{\Leftrightarrow} B_{i-1}(a,x) \leq B_i(a, x+y) \wedge x = B_i(n,a),$
$\wedge\, x + y = B_{i+1}(n,a)$
$\overset{(\star)}{\Leftrightarrow} B_{i-1}(a,x) \leq B_i(a,x) \leq B_i(a,x+y)$

$B_i(a,x) \leq B_i(a,x+y)$ holds because all boundary functions are obviously montonously increasing w.r.t each variable.

□

Theorem 2.14

$\forall\, i \geq 0 : B_{i+1} \notin \mathcal{E}^i$
This is proven by the fact that there exists no function $m \in \mathcal{E}^i$ s.t. $B_{i+1} \leq m$. It is obvious that without the use of substitution B_i attains the highest functions values in $\mathcal{E}^i$. Using substitution only once, it can be verified that $B_i(B_i(n,a), B_i(n,a))$ attains the highest function values. We generalize this for k substitutions:

$$B_i^{(0)} = B_i(n,a)\ ,\ B_i^{(k+1)} = B_i(B_i^{(k)}(n,a), B_i^{(k)}(n,a))$$

We conclude that being able to argue that for any $k \geq 0$ there exists a tuple (n,a) such that $B_i^{(k)}(n,a) < B_{i+1}(n,a)$ implies that $B_{i+1} \notin \mathcal{E}^i$. We start with our special cases B_0 and B_1.

$$B_0^{(k)}(n,a) = n + 1 + k\ ,\ B_1^{(k)}(n,a) = 2^k(n+a)$$

These identities can be easily proved by induction. Now, we can show

$$\forall\, k \geq 0\, \exists\, n, a \geq 0 : B_0^{(k)}(n,a) < B_1(n,a) \Leftrightarrow n + 1 + k < n + a$$

$$\text{Possible solution: } n = 0\ ,\ a = k + 2$$

$$\forall\, k \geq 0\, \exists\, n, a \geq 0 : B_1^{(k)}(n,a) < B_2(n,a) \Leftrightarrow 2^k(n+a) < n * a$$

$$z := n = a : 2^{k+1}z < z^2 \Leftrightarrow 2^{k+1} < z$$

$$\text{Possible solution: } n = a = 2^{k+2}$$

For the general case it has to be shown that

$$(\diamond)\ \forall\, i \geq 3\, \exists\, n, a \geq 0 : B_{i-1}^{(k)}(n,a) < B_i(n,a)$$

holds for all $k \geq 0$. For $k = 0$:

$$(\star)\ \exists\, n \geq 0\, \exists\, a_0 \geq 0\, \forall\, a \geq a_0 : B_{i-1}(n,a) < B_i(n,a)$$

We proof the above statement by induction over i and start with $i = 3$:

$$\exists\, a_0 \geq 0\, \forall\, a \geq a_0 : B_2(n,a) < B_3(n,a) \Leftrightarrow n * a < a^n = a * a^{n-1}$$

$$\Leftrightarrow n < a^{n-1} \Leftrightarrow\ \sqrt[n-1]{n} < a \Rightarrow a_0 := \left\lceil \sqrt[n-1]{n} \right\rceil + 1$$

The case $i + 1 > 3$:

$$\exists\, a_0 \geq 0\, \forall\, a \geq a_0 : B_i(n,a) < B_{i+1}(n,a)$$

$$\Leftrightarrow B_{i-1}(a, B_i(n-1,a)) < B_i(a, B_{i+1}(n-1,a))$$

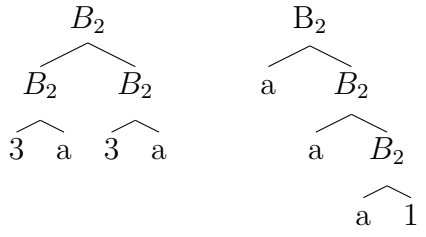
$$\text{due to } (\star)\ \exists\, x_0 \geq 0\, \forall\, x \geq x_0 : B_{i-1}(n,x) < B_i(n,x)$$

From our previous Lemma we know that $B_i \leq B_{i+1}$ holds for all $i \geq 2$ and furthermore that all boundary functions are strictly montonously increasing w.r.t. to each variable for sufficiently large values.

$$x := B_i(n-1,a)\ ,\ y := B_{i+1}(n-1,a)\ ,\ x \leq y$$

$$\forall\, x \geq x_0 : B_{i-1}(a,x) < B_i(a,x) \leq B_i(a,y)$$

Therefore the statement $(\diamond)$ holds for $k = 0$. For $k > 0$ we will provide no more than an informal, intuitional argument. For a formal proof of this last part look at [Grz53]. Let us consider the example $B_2^{(1)}(3,a)$ (left tree) compared to $B_3(3,a)$ (right tree):



Since $B_2(n,a) = n * a$ the left tree yields $3^2 * a^2$ and the right tree yields a^3. In general, $B_i^{(k)}(n,a)$ can be described as a balanced binary tree parameterized by the constant k with a depth of $k+1$ and $2^{k+1}-1$ nodes B_i. The structure of $B_{i+1}(n,a)$ is determined by the variable n with a depth of n and n nodes

B_i. The intuition why B_{i+1} will exceed the function value of $B_i^{(k)}$ at some point, is that choosing a sufficiently large n will lead to a function which has a higher growth rate than $B_i^{(k)}$ with respect to a as seen in the example $9a^2$ vs. a^3 for $n = 3$.
An interesting observation is that for $i \in \{1, 2\}$ $B_i^{(k)} = B_{i+1}(2^k, B_i(n, a))$ which can be explained by the commutative and associative property of these operators.

□

Corollary 2.15

$\forall\, i \geq 0 : \mathcal{E}^i \subsetneq \mathcal{E}^{i+1}$

Lemma 2.16

$\bigcup_{i\in\mathbb{N}} \mathcal{E}^i \subseteq \mathcal{PR}$
Per definition $\mathcal{E}^i \subseteq \mathcal{PR}$ for all $i \geq 0$. It follows that the union of subsets of $\mathcal{PR}$ cannot be larger than $\mathcal{PR}$.

□

Due to our adjusted definition of bounded primitive recursion this statement is trivial, which indeed is not the case [5].

2.10 Recursive depth Hierarchy

Using the recursive depth for primitive recursion we define the hierarchy:

$$\mathcal{PR}^i = \{f \in \mathcal{PR} \mid |f|_{rd} \leq i\}$$

Lemma 2.17

$\bigcup_{i\in\mathbb{N}} \mathcal{PR}^i = \mathcal{PR}$
Per definition $\mathcal{PR}^i \subset \mathcal{PR}$ holds for all $i \geq 0$. Considering the definition of recursive depth for every $f \in \mathcal{PR}$ there is a $c \in \mathbb{N}$ s.t. $|f|_{rd} = c$. Hence, $\mathcal{PR} \subseteq \bigcup_{i\in\mathbb{N}} \mathcal{PR}^i$

□

Lemma 2.18

$\forall\, i \geq 0 : \mathcal{PR}^i \subseteq \mathcal{E}^{i+1}$
Every function $f^2 \in \mathcal{PR}^i$ can be built by using bounded primitive recursion

[5] See footnote 4

avaiable to $\mathcal{E}^{i+1}$ since there must be some $c \geq 0$ such that $f \leq B_{i+1}^{(c)}$. To show that it can be argued that $B_i^{(c)}$ is the function which attains the highest function values in $\mathcal{PR}^i$ for some c and conclude that $B_i^{(c)} \leq B_{i+1}^{(c)}$.

□

Lemma 2.19

$\bigcup_{i\in\mathbb{N}} \mathcal{E}^i = \mathcal{PR}$
The previous Lemma implies that $\bigcup_{i\in\mathbb{N}} \mathcal{PR}^i \subseteq \bigcup_{i\in\mathbb{N}} \mathcal{E}^i$ which further implies $\mathcal{PR} \subseteq \bigcup_{i\in\mathbb{N}} \mathcal{E}^i$.

□

For our next Theorem we need to distinguish the recursive depth of a nested recursion. Let us denote it with $|f|_{nd}$ for $f \in \mathcal{PR}_{nes}$ from now on. We introduce

$$\mathcal{PR}_{nes}^i = \{f \in \mathcal{PR}_{nes} \mid |f|_{nd} \leq i\}$$

and

$$\text{LOOP}^i = \{P \in \text{LOOP} \mid |P|_{ld} \leq i\}$$

Lemma 2.20

$\forall\, i \geq 0 : \text{LOOP}^i \subseteq \mathcal{PR}_{nes}^i$
For $i = 0$ this obviously holds as shown in Theorem 2.10 and 2.11. For $i+1$ we look at the program $P' =$ "LOOP x_k DO P END" with $|P|_{ld} = i$, m variables and $1 \leq k \leq m$. Due to the inductive hypothesis there are $\lambda_1, \ldots, \lambda_m$ which each model the change of the respective variable $x_1, \ldots x_m$ after one iteration of the loop. Further it holds that $|\lambda_j|_{nd} = i$ for all $1 \leq j \leq m$. Using a nested recursion as stated in Theorem 2.10 we obtain a function f s.t. $f = f_{P'}$ and $|f|_{nd} = i + 1$. For the case "P_1; P_2" with $|P_1|_{ld}, |P_2|_{ld} \leq i + 1$ substitution can be used which does not affect the recursive depth.

□

Theorem 2.21

$\forall\, i \geq 0 : \mathcal{PR}_{nes}^i \subseteq \mathcal{PR}^{2i}$
For $i = 0$ the statement $\mathcal{PR}_{nes}^0 \subseteq \mathcal{PR}^0$ obiously holds. For the case $i + 1$:

$$\mathcal{PR}_{nes}^{i+1} \subseteq \mathcal{PR}^{2(i+1)}$$

We show for

$$f^2 \in \mathcal{PR}_{nes}^{i+1} \text{ with } |f|_{nd} = i + 1$$

that there exists a $f'(n,a) \in \mathcal{PR}$ with $|f'|_{rd} \leq 2(i+1)$ s.t. $f' = f$.

$$f(n,a) = \begin{cases} g(a) & , n = 0 \\ h(n-1, a, f(n-1, \lambda(n-1,a))) & , \text{else} \end{cases}$$

$$|f|_{nd} = 1 + \max\{|g|_{nd}, |h|_{nd}, |\lambda|_{nd}\} = i+1$$
$$\Leftrightarrow i = \max\{|g|_{nd}, |h|_{nd}, |\lambda|_{nd}\}$$

Our inductive hypothesis ensures the existence of $g', h', \lambda' \in \mathcal{PR}$ with $|g'|_{rd}, |h'|_{rd}, |\lambda'|_{rd} \leq 2i$ such that $g = g', h = h', \lambda = \lambda'$. Using the translation of Theorem 2.8 to convert our nested recursion f into a primitve recursion f' with the help of g', h', λ' we can conclude

$$|f'|_{rd} = 1 + \max\{|g'|_{rd}, |h'|_{rd}, |w|_{rd}\}$$

$$|w|_{rd} = 1 + |\lambda'|_{rd}$$
$$\Rightarrow |f'|_{rd} \leq 1 + 1 + 2i = 2(i+1)$$

□

Corollary 2.22

$\text{LOOP}^i \subseteq \mathcal{PR}^{2i}$

Stated differently, this means that any Loop program with nesting depth i can be written as a primitive recursive function with a recursive depth of at most $2i$.

2.11 Turing Machine Simulation

We define a Turing Machine with one-sided infinite tape which always halts based on the more general version given in [Hop79] as:

$$M = (Q, \Gamma, b, \Sigma, \delta, q_0, q_+, q_-)$$

- Q is the finite, non-empty set of states
- Γ is the finite, non-empty set of the tape alphabet
- $b \in \Gamma$ is the blank symbol
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of the input alphabet

- $q_0 \in Q$ is the initial state
- $q_+, q_- \in Q$ where q_+ is the accepting final state and q_- is the rejecting final state
- $\delta : Q \setminus \{q_+, q_-\} \times \Gamma \mapsto Q \times \Gamma \times \{0, 1, 2\}$ is a computable, total transition function with 0 being a left shift, 2 being a right shift and 1 being no shift on the tape

The characteristic function of a Turing Machine M for some input x is defined as:

$$c(x) = \begin{cases} 0 & , M(x) \text{ stops with final state } q_- \\ 1 & , M(x) \text{ stops with final state } q_+ \end{cases}$$

A function $f(n)$ is said to be a runtime function for a Turing Machine M, iff for all inputs x of length n $M(x)$ induces at most $f(n)$ calls of the transition function until it reaches a final state.

Theorem 2.23

Given a Turing machine $M = (Q, \Gamma, b, \Sigma, \delta, q_0, q_+, q_-)$ with a runtime function $f \in \mathcal{PR}$, its characteristic function c is primitive recursive.

First, we argue that the elements of the sets Q and Γ can be enumerated by counting from 0 and we can replace the symbols with the respective number. Let us assume that $b = 0$ and $q_0 = 0$. Since the transition function δ is computable and only has a finite definition domain the result for each input can be computed and expressed as definition by case. This concludes that each of the following functions is primitive recursive.

$$\delta_Q(q, \gamma) = q' \overset{def}{\Leftrightarrow} \delta(q, \gamma) = (q', x, y)$$

$$\delta_\Gamma(q, \gamma) = \gamma' \overset{def}{\Leftrightarrow} \delta(q, \gamma) = (x, \gamma', y)$$

$$\delta_D(q, \gamma) = d \overset{def}{\Leftrightarrow} \delta(q, \gamma) = (x, y, d)$$

Our simulator can be defined as nested recursion with $\mathcal{P}_n$ being the prime number sequence for this argumentation:

$$h(r, q, t, p) = \begin{cases} 0 & , r = 0 \\ h'(q, h(r-1, \lambda_Q(q,t,p), \lambda_T(q,t,p), \lambda_D(q,t,p))) & , \text{else} \end{cases}$$

$$h'(q, h_{pre}) = \begin{cases} 0 & , q = q_- \\ 1 & , q = q_+ \\ h_{pre} & , \text{else} \end{cases}$$

$$\lambda_Q(q,t,p) = \delta_Q(q, \varpi(p,t)) \ , \ \lambda_D(q,t,p) = p + \delta_D(q, \varpi(p,t)) - 1$$

$$\lambda_T(q,t,p) = \left\lfloor \frac{t}{\mathcal{P}_p^{\varpi(p,t)}} \right\rfloor * \mathcal{P}_p^{\delta_\Gamma(q,\varpi(p,t))}$$

For all inputs $x = x_0 \dots x_{n-1}$ with $x_i \in \Sigma$ for $0 \leq i < n$ it holds that

$$c(x) = h(f(n) + 1, 0, \prod_{i=0}^{n-1} \mathcal{P}_i^{x_i}, 0)$$

The nested recursion saves the current state in q, the tape position in p and the tape in t by utilizing prime power decomposition. λ_Q simulates the state transition, λ_D does that for the tape position and λ_T modifies the current tape cell accordingly. $\varpi(p,t)$ returns the current cell content. h calls $f(n)+1$ to ensure that h' is always called at least once. Due to the requirement that M has to terminate after doing at most $f(n)$ steps, $h(0, \dots)$ is never called. □

Corollary 2.24

$\forall\, L \in$ EXPSPACE with characteristic function c it holds that $c \in \mathcal{PR}$.

Proof sketch: utilize that DSPACE$(s(n)) \subseteq$ DTIME$(2^{O(s(n))})$ as known from introductory course to computational complexity and show that $O(2^{2^{n^{O(1)}}}) \subseteq \mathcal{PR}$.

3 Multiple and μ-recursion

3.1 Multiple Recursion class $\mathcal{MR}$

The next probable 'natural' extension to recursive functions would be instead of only recursing over one variable to do this over r recursive variables. The following definition is more powerful than the one stated in [Pet36] since it inherently allows for nested and course-of-value recursion. [Pet35] has shown that this definition does not exceed multiple recursion by the same means we have used to show that nested and course-of-value recursion do not extend $\mathcal{PR}$. Therefore we will spare us from showing this again. This class of recursive functions will be called $\mathcal{MR}$ and as extension self-evidently contains all functions of $\mathcal{PR}$.

Let $n_1, \ldots, n_r, a \in \mathbb{N}$

- Predecessor: $\text{pre}(a) \in \mathcal{MR}$
- Composition: let $f^m, g_1^r, \ldots, g_m^r \in \mathcal{MR}$, then it holds that $f(g_1(n_1, \ldots, n_r), \ldots, g_m(n_1, \ldots, n_r)) \in \mathcal{MR}$ for $m \in \mathbb{N}$
- Multiple recursion: let $g^r, h^{r+1+c}, \mu_i^{r+1}, \lambda_{i,j}^{r+1} \in \mathcal{MR}$ for $1 \leq i \leq c$ $1 \leq j \leq r$ and f is defined by:

$$f(n_1, \ldots, n_r, a) = \begin{cases} g(n_2, \ldots, n_r, a) & , n_1 = 0 \\ h(n_1, \ldots, n_r, a, f_1, \ldots, f_c) & , \text{else} \end{cases}$$

$f_i = f_i(\mu_i, \lambda_{i,1}, \ldots, \lambda_{i,r})$, $\mu_i = \mu_i(n_1, \ldots, n_r, a)$ with $\mu_i < n_1$ and $\lambda_{i,j} = \lambda_{i,j}(n_1, \ldots, n_r, a)$
$f \in \mathcal{MR}$ for all $r, c \geq 0$.

The same as for nested recursion applies for the λ_i functions. A λ_i is only allowed to be defined in terms of f such that all calls of f induce only a finite number of function calls, or in other words f remains total. Even though any of the recursive variables but the first can grow during the recursion all functions in $\mathcal{MR}$ still remain total since the first recursive variable is strictly decreasing for each recursive step.
Our next step is to show that the class of multiple recursion is a strict superset of primitive recursion. For that we use exactly the same argumentation as in [Pet35].
As we have seen, primitive recursive functions can be restricted to the use of only one parameter without loss of generality. Now we will show that a primitive recursion with one parameter can be rewritten into one without any parameters.

Lemma 3.1

$\forall\, f(n,a) = \mathrm{REC}[g^1, h^3] \in \mathcal{PR}\ \exists\, f'(n) \in \mathcal{PR} : f(n,a) = f'(2^n * 3^a)$

If $x = 2^n * 3^a$ is divisible by 2, this implies $n \neq 0$ and therefore h has to be called else g. This can be written as course-of-value recursion:

$$f'(x) = \begin{cases} 0 & ,x = 0 \\ g(\varpi(1,x)) & ,\neg(2|x) \\ h(\varpi(0,x) - 1, \varpi(1,x), f'(\left\lfloor \frac{x}{2} \right\rfloor)) & ,2|x \end{cases}$$

Note, that the function value of $f'(0)$ is not bound by f, so $f(0)$ can be an arbitrary value.

□

With this we state an equivalent definition of $\mathcal{PR}$ which we call $\mathcal{PR}_2$.

Let $n, n_1, n_2 \in \mathbb{N}$

- $\{\pi_1^2, \pi_2^2, \mathrm{pow}, \varpi\} \subset \mathcal{PR}_2$
- $\mathrm{p}(n,a) := p_n \in \mathcal{PR}_2$.
  Note, that the parameter a is a fictive variable
- Composition: let $f^2, g_1^2, g_2^2 \in \mathcal{PR}_2$, then
  $f(g_1(n_1,n_2), g_2(n_1,n_2)) \in \mathcal{PR}_2$
- Primitive recursion: let $h^2 \in \mathcal{PR}_2$ and f is defined by:
  $f(n) = \begin{cases} 0 & ,n = 0 \\ h(n-1, f(n-1)) & ,\text{else} \end{cases}$
  $f^2 \in \mathcal{PR}_2$; note that the second variable is fictive
  for short $f = \mathrm{REC}_2[h]$

Since $\mathcal{PR}_2$ is obviously a subset of $\mathcal{PR}$ it just has to be shown that the constant-0 function and the successor function can be expressed in $\mathcal{PR}_2$ to show that both definitions are equivalent.

- $0 = \varpi(n,n)$
- $n * a = \varpi(0, (p_0^n)^a)$
- $n + 1 = \varpi(0, p_0^n * p_0)$

Now we can enumerate all functions in $\mathcal{PR}_2$ as a sequence

$$\varphi_0(n,a), \varphi_1(n,a), \ldots, \varphi_m(n,a)$$

such that every function in $\mathcal{PR}$ occurs at least once in this series. In other words φ is a valid enumeration. Examing the function $\phi(m,n,a) := \varphi_m(n,a)$ leads us to a diagonalization argument.

Theorem 3.2

$\phi(m,n,a) \notin \mathcal{PR}$
Assume the opposite: $\phi(m,n,a) \in \mathcal{PR} \Rightarrow \phi(n,n,a) = \varphi_n(n,a) \in \mathcal{PR} \Rightarrow \phi(n,n,a)+1 \in \mathcal{PR} \Rightarrow \exists m : \varphi_m(n,a) = \phi(n,n,a)+1 \Rightarrow$ for the function value with $n = m$: $\varphi_m(m,a) = \phi(m,m,a) + 1 \Rightarrow \phi(m,m,a) = \phi(m,m,a) + 1$ $\lightning$
□

To argue that $\phi(m,n,a) \in \mathcal{MR}$ we need to state a concrete enumeration. The first five functions are defined as:

- $\varphi_0(n,a) = \pi_1^2(n,a)$
- $\varphi_1(n,a) = \pi_2^2(n,a)$
- $\varphi_2(n,a) = \mathrm{p}(n,a)$
- $\varphi_3(n,a) = \mathrm{pow}(n,a)$
- $\varphi_4(n,a) = \varpi(n,a)$

For $k > 4$ and $k \equiv_2 0$ primitive recursion is used:

- $\varphi_k(n,a) = \begin{cases} 0 & , n = 0 \\ \varphi_{\varpi(0,k)}(n-1, \varphi_k(n-1,a)) & , \text{else} \end{cases}$

For $k > 4$ and $k \equiv_2 1$ substitution is used:

- $\varphi_k(n,a) = \varphi_{\varpi(1,k)}(\varphi_{\varpi(2,k)}(n,a), \varphi_{\varpi(3,k)}(n,a))$

Lemma 3.3

φ is a valid enumeration
To show that this is a valid enumeration which covers all functions in $\mathcal{PR}$ at least once, for any given k there needs to be an r such that $\varphi_r = \mathrm{REC}_2[\varphi_k]$. A possible solution is $r = 2^k * 5$. Obviously, we cannot find an r for $k = 0$ since r needs to be even and bigger than 4. However, this isn't bad as

$\mathrm{REC}_2[\varphi_0] = \mathrm{REC}_2[\pi_1^2] = n$ thus not foiling our argumentation.
This needs to hold for substitution as well; explicitly stated: given k_1, k_2, k_3 there has to exist a s s.t. $\varphi_s = \varphi_{k_1}(\varphi_{k_2}, \varphi_{k_3})$. A valid relation would be $s = 3^{k_1} * 5^{k_2} * 7^{k_3} * 11$.

□

Finally, we separate $\mathcal{MR}$ from $\mathcal{PR}$ with the help of the aforementioned function ϕ.

$$\phi(m,n,a) = \begin{cases} n & ,m = 0 \\ a & ,m = 1 \\ \mathrm{p}(n) & ,m = 2 \\ \mathrm{pow}(n,a) & ,m = 3 \\ \varpi(n,a) & ,m = 4 \\ 0 & ,m > 4 \wedge m \equiv_2 0 \wedge n = 0 \\ \phi(\varpi_0(m), n-1, \phi(m,n-1,a)) & ,m > 4 \wedge m \equiv_2 0 \wedge n > 0 \\ \phi(\varpi_1(m), \phi(\varpi_2(m), n, a), & \\ \qquad \phi(\varpi_3(m), n, a)) & ,m > 4 \wedge m \equiv_2 1 \end{cases}$$

Theorem 3.4

$\phi(m,n,a) \in \mathcal{MR}$

$$\phi(m,n,a) = \phi'(m,n,p_1^a)$$

$$\phi'(m,n,b) = \begin{cases} \phi'(\varpi_0(m+1), n, \frac{b}{2}) & ,M = 1 \\ \phi'(\varpi_1(m+1), n, \frac{b}{4}) & ,M = 2 \\ n & ,m = 0 \\ a' & ,m = 1 \\ \mathrm{p}(n) & ,m = 2 \\ \mathrm{pow}(n,a') & ,m = 3 \\ \varpi(n,a') & ,m = 4 \\ 0 & ,m > 4 \wedge m \equiv_2 0 \wedge n = 0 \\ \phi'(m-1, n-1, p_0^1 * p_1^{\phi'(m,n-1,b)}) & ,m > 4 \wedge m \equiv_2 0 \wedge n > 0 \\ \phi'(m-1, \phi'(\varpi_2(m), n, b), & \\ \quad p_0^2 * p_1^{\phi'(\varpi_3(m),n,b)}) & ,m > 4, m \equiv_2 1 \end{cases}$$

$$M = \varpi(0,b)\ ,\ a' = \varpi(1,b)$$

Why is this a valid multiple recursion? For the case $M = 1$ and $M = 2$ we use course-of-value recursion to obtain the predeccesor $\varpi_0(m+1)$ resp. $\varpi_1(m+1)$ and a nested recursion to modify the non-recursive parameter b. Note, that $\varpi_i(m+1) < m$ for all $m > 4$ and $i \in \{0, 1\}$ thus not contradicting the necessity of a strictly decreasing first recursive variable. The case of primitive recursion($m > 4 \wedge m, m \equiv_2 0, n > 0$) uses a nested recursion which is validly defined in terms of ϕ'. For the substitution case($m > 4 \wedge m, m \equiv_2 1$) two self-refering nested recursions are used as well.

□

Proof of Correctness

We show

$$(\diamond)\ \forall\, i \geq 0 : \phi(i, n, a) = \phi'(i, n, b) \text{ with } b = p_1^a$$

For $0 \leq i \leq 4$ this obviously holds. For the case of recursion, namely $i + 1 > 4 \wedge i + 1 \equiv_2 0$ it holds for the special case $n = 0$. So per induction we can assume

$$(\star)\ \phi(i + 1, n, a) = \phi'(i + 1, n, b)$$

for proving

$$\phi(i + 1, n + 1, a) \stackrel{?}{=} \phi'(i + 1, n + 1, b)$$

$$\phi(i + 1, n + 1, a) = \phi(\varpi_0(i + 1), n, \phi(i + 1, n, a))$$

$$\phi'(i + 1, n + 1, b) = \phi'(i, n, p_0^1 * p_1^{\phi'(i+1,n,b)}) = \phi'(\varpi_0(i + 1), n, p_1^{\phi'(i+1,n,b)})$$

Using $(\diamond)$ this is true, iff

$$\phi'(i + 1, n, b) = \phi(i + 1, n, a)$$

which holds due to $(\star)$. For substitution we show this for all $i + 1 > 4$ and $i + 1 \equiv_2 1$.

$$\phi(i + 1, n, a) = \phi(\varpi_1(i + 1), \phi(\varpi_2(i + 1), n, a), \phi(\varpi_3(i + 1), n, a))$$

$$\phi'(i + 1, n, b) = \phi'(i, \phi'(\varpi_2(i + 1), n, b), p_0^2 * p_1^{\phi'(\varpi_3(i+1),n,b)})$$

$$= \phi'(\varpi_1(i + 1), \phi'(\varpi_2(i + 1), n, b), p_1^{\phi'(\varpi_3(i+1),n,b)})$$

Due to the fact that $\varpi_j(i + 1) < i + 1$ for $i > 3, j \geq 1$ we can use $(\diamond)$. Therefore

$$z_j := \phi'(\varpi_j(i + 1), n, b) = \phi(\varpi_j(i + 1), n, a) \text{ for } j \in \{2, 3\}$$

$$\phi'(\varpi_1(i + 1), z_2, p_1^{z_3}) = \phi(\varpi_1(i + 1), z_2, z_3)$$

□

Remarks

The set of multiple recursion $\mathcal{MR}$ can be divided into subsets $\mathcal{MR}^i$ for all $i > 0$. Such a subset contains only functions with at most i recursive variables. The set $\mathcal{MR}^1 = \mathcal{PR}$ and we have just shown that $\mathcal{MR}^1 \subsetneq \mathcal{MR}^2$. [Pet36] has generalized this diagonalization argument to prove that indeed $\mathcal{MR}^i \subsetneq \mathcal{MR}^{i+1}$ for all $i > 0$ holds. In [Pet50] transfinite recursion is introduced and shown to be a strict superset of $\mathcal{MR}$, again by diagonalization. For that, the number of recursive variables for the diagonalization function of $\mathcal{MR}$ is determined by a variable of the function itself. For an eventual reference we denote the set of transfinite recursive functions with $\mathcal{TR}$.

3.2 μ-Recursion class $\mu\mathcal{R}$

The class of $\mu\mathcal{R}$ as shown in [Wag85] is defined as follows:

- $f \in \mathcal{PR} \Rightarrow f \in \mu\mathcal{R}$
- μ-recursion: for total $f^{k+1} \in \mu\mathcal{R}$ let
  $\mu[f](n_1, \ldots, n_k) = \min\{x \geq 0 \mid f(x, n_1, \ldots, n_k) = 0\}$
  $\mu[f] \in \mu\mathcal{R}$ for all $k \geq 0$

To prove that μ-Recursion is Turing complete we will use While programs for which this property is known.

Definition of While programs

For the definition of While programs we refer to section 2.8 Loop programs. A While program is semantically and syntactically defined in the same way as a Loop program with the following exception:

- If P is a While program and x_i is a variable, then "LOOP x_i DO P END;" is <u>not</u> a While program
- If P is a While program and x_i is a variable, then "WHILE $x_i \neq 0$ DO P END;" is a While program. This statement means to succesively execute P as long as x_i doesn't contain the value 0.

We call the set of all While programs WHILE. A While program can be interpreted as a set of functions for each variable just as for a Loop program. If a While program does not terminate for a certain input the corresponding function value is undefined for this case.
The While nesting depth $|P|_{wd}$ is determined in the same way as the Loop nesting depth with the addition:

- if $P =$ "WHILE $x_i \neq 0$ DO P_l END;", then $|P|_{wd} = |P_l|_{wd} + 1$

Lemma 3.5

WHILE $\subseteq \mu\mathcal{R}$
Our inductive hypothesis:

$$\forall\, P \in \text{WHILE and } |P|_{wd} \leq \ell : f_P \in \mu\mathcal{R}$$

with f_P being the function calculated by the While program P.
The base case $\ell = 0$ is the same as in Theorem 2.10 because the set of While programs equals the set of Loop programs if both are restricted to a nesting depth of 0 and $\mathcal{PR}$ is a subset of $\mu\mathcal{R}$ per definition.
For $\ell + 1$:

$$\text{"WHILE } x_i \neq 0 \text{ DO } P_l \text{ END;"}$$

The inductive hypothesis grants us as set of functions $f_1, \ldots, f_m$ which calculate the value of their respective variable $x_1, \ldots, x_m$ after the execution of P_l. For $1 \leq j \leq m$:

$$h_j(n, x_1, \ldots, x_m) = \begin{cases} x_j & , n = 0 \\ h_j(n-1, f_1(x_1, \ldots, x_m), \ldots, f_m(x_1, \ldots, x_m)) & , \text{else} \end{cases}$$

With that f_P can be expressed as

$$f_P = h_1(z, x_1, \ldots, x_m)$$

$$z := \mu[h_i](x_1, \ldots, x_m)$$

z is the smallest number such that executing P_l z times leads to $x_i = 0$. This is exactly the number of iterations induced by the "WHILE" instruction. If z is undefined for a particular input, so is f_P. For the While program $P =$ "P_1 P_2" the same argumentation as in Theorem 2.10 can be applied.

□

Lemma 3.6

$\mu\mathcal{R} \subseteq$ WHILE
For this we simply proof that

$$\forall\, f \in \mu\mathcal{R}\, \exists\, P \in \text{WHILE} : \mu[f](x_1, \ldots, x_m) = f_P$$

with f_P being the function calculated by the While program P. Essentially, this is yet another inductive argument over the recursive depth such that f can be assumed to be calculated by some While program. The wanted While program P is:

```
1 x_cnt := 0;  x_f := f(0, x_1, ..., x_m);
2 WHILE x_f ≠ 0 DO
3     x_cnt := x_cnt + 1;
4     x_f := f(x_cnt, x_1, ..., x_m);
5 END;
6 x_1 = x_cnt;
```

□

Corollary 3.7

$\mu\mathcal{R}$ is Turing complete.

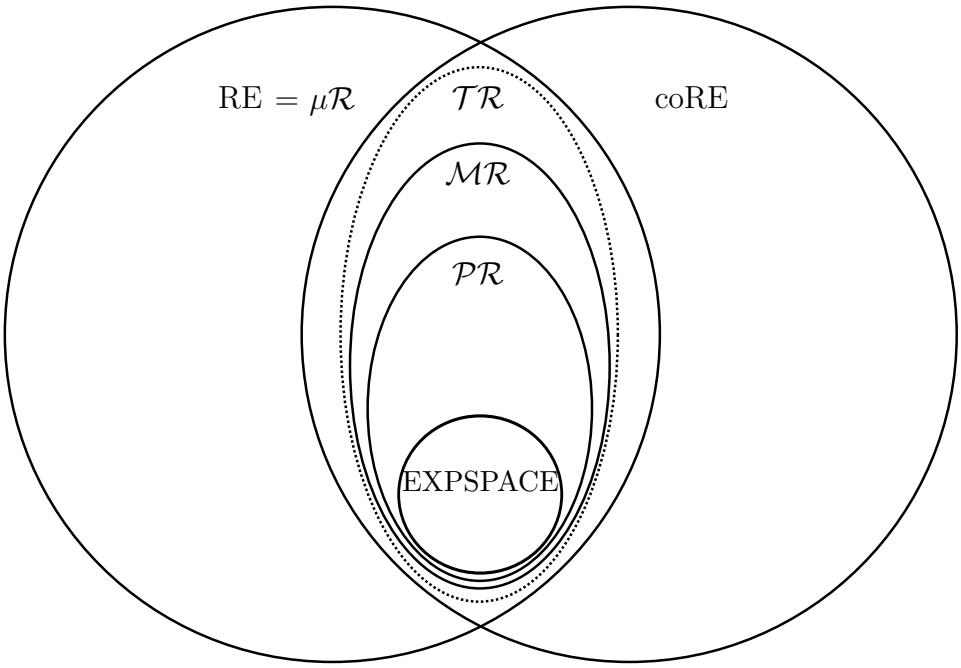
3.3 Synopsis

To adequately classify the different types of recursion we have encountered in terms of complexity classes we only look at binary functions and introduce following known complexity classes.

$\mathrm{RE} = \{L \subseteq \mathbb{N} \mid \exists \text{ Turing Machine } M \text{ which halts on some input } n, \text{ iff } n \in L\}$

$$\mathrm{coRE} = \{L \subseteq \mathbb{N} \mid \mathbb{N} \setminus L \in \mathrm{RE}\}$$

Now, the results can be subsumed in the following figure:



List of literature

[Grz53] A. Grzegorczyk (1953). Some classes of recursive functions. Rozprawy Matematyczne. Pages 32–33, 39–40

[Hop79] J. Hopcroft and J. Ullman (1979). Introduction to Automata Theory, Languages and Computation. Addison-Wesley. Page 148

[Pet34] R. Peter (1934). Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion. Mathematische Annalen 110. Pages 613–623

[Pet35] R. Peter (1935). Konstruktion nichtrekursiver Funktionen. Mathematische Annalen 111. Pages 45–53

[Pet36] R. Peter (1936). Über die mehrfache Rekursion. Mathematische Annalen 113. Pages 490, 495

[Pet50] R. Peter (1950). Zusammenhang der mehrfachen und transfiniten Rekursion. The Journal of Symbolic Logic, Vol. 15. Pages 248–249

[Vol11] H. Vollmer (2011). Skript zur Vorlesung Grundlagen der Theoretischen Informatik, Wintersemester 2011/2012. Leibniz Universität Hannover. Pages 33–34

[Wag85] K. Wagner and G. Wechsung (1985). Computational Complexity. D. Reidel Publishing Company. Pages 39–46