Reflection: a powerfull and ubiquitous logical mechanism

Henk Barendregt

Faculty of Science
Radboud University
Nijmegen, The Netherlands

Huygens Lecture
Fall 2007

Contents

1. Overview

The phenomenon of *reflection* will be introduced and clarified by examples. Reflection plays in several ways a fundamental rôle for our existence. Among other places the phenomenon occurs in life, in language, in computing and in mathematical reasoning. A fifth place in which reflection occurs is our spiritual development. In all of these cases the effects of reflection are powerful, even downright dramatic. We should be aware of these effects and use them in a responsible way.

A prototype situation where reflection occurs is in the so called *lambda calculus*. This is a formal theory that is capable of describing algorithms, logical and mathematical proofs, but also itself. We will focus[1] on the mechanism that allows this self-reflection. This mechanism is quite simple, in the (intuitive) order of complexity of a computation like

$$(x^2 + xy + y^2)(x - y) = x^3 - y^3,$$

but—as argued above—it has quite dramatic effects.

Reflection: domain, coding and interaction

Reflection occurs in situations in which there is a *domain* of objects that all have *active meaning*, i.e. specific functions within the right context. Before turning to the definition itself, let us present the domains relevant for the four examples. The first domain is the class of proteins. These have indeed specific functions within a living organism, from bacterium to *homo sapiens*. The second domain consists of sentences in natural language. These are intended, among other things, to make statements, to ask questions, or to influence others. The third domain consists of (implemented) computable functions. These perform computations—sometimes stand alone, sometimes interactively with the user—so that an output results that usually serves us in one way or another. The fourth domain consists of mathematical theorems. These express valid phenomena about numbers, geometric figures or other abstract entities. When interpreted in the right way, these will enable us to make correct predictions.

Now let us turn to reflection itself. Besides having a domain of meaningful objects it needs *coding* and *interaction*. Coding means that for every object of the domain there is another object, the (not necessarily unique) *code*, from which the original object can be reconstructed exactly. This process of reconstruction is called *decoding*. A code C of an object O does not directly possess the active meaning of O itself. This happens only after decoding. Therefore the codes are outside the domain, and form the so-called *code set*. Finally, the interaction needed for reflection consists of the encounter of the objects and their codes. Hereby some objects may change the codes, after decoding giving

[1] A different focus on reflection is given in the Honours Program 'Reflection and its use, from Science to Meditation' (February 3 — March 30, 2004). The lecture notes for that course put the emphasis more on mathematics and meditation, see <ftp://ftp.cs.kun.nl/pub/CompMath/honours.pdf>.

rise to modified objects. This process of *global* feedback (in principle on the *whole* domain via the codes) is the essence of reflection[2].

Examples of reflection

Having given this definition, four examples of reflection will be presented.

1. **Proteins.** The first example has as domain the collection of proteins. A typical protein is shown in the following picture denoting in a stylized way human NGF, important for the development of the nervous system.

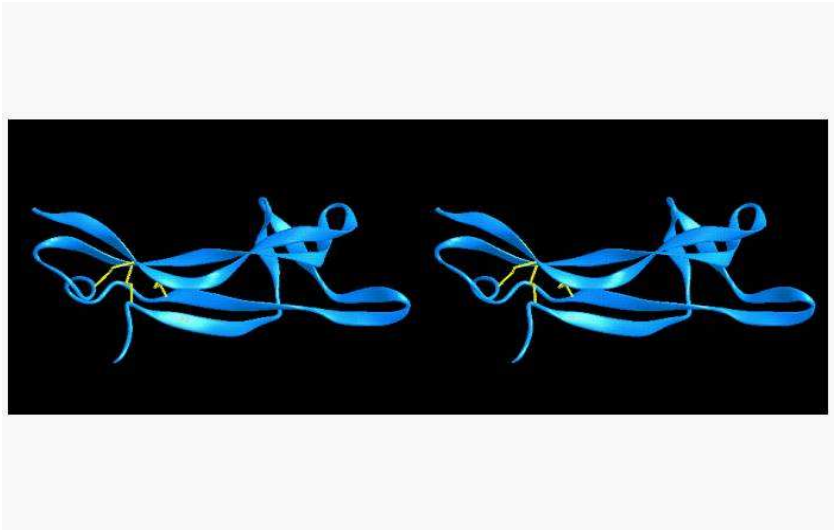


Figure 1: A schematic display of the protein NGF_Homo_Sapiens, a nerve growth factor. Courtesy of the Swiss Institute of Bioinformatics, see Peitsch at al. [1995]. ⟨`ftp://ftp.expasy.org/databases/swiss-3dimage/IMAGES/JPEG/S3D00467.jpg`⟩

The three dimensional structure of the protein can be perceived by looking at the picture with crossed eyes such that the left and right images overlap. Each protein is essentially a linear sequence of elements of a set of 20 amino acids. Because some of these amino acids attract one another, the protein assumes a three dimensional shape that provides its specific chemical meaning. The sequence of amino-acids for the NGF protein is shown in Fig.2.

[2]It should be emphasized that just the coding of elements of a domain is not sufficient for reflection. A music score may code for a symphony, but the two are on different levels: playing a symphony usually does not alter the written music. However, in aleatory music—the deliberate inclusion of chance elements as part of a composition—the performance depends on dice that the players throw. In most cases, the score (the grand plan of the composition) will not alter. But music in which it really does alter is a slight extension of this idea.

Protein: 241 amino acids; molecular weight 26987 Da.
www.ebi.ac.uk/cgi-bin/expasyfetch?X52599

```
MSMLFYTLIT  AFLIGIQAEP  HSESNVPAGH  TIPQVHWTKL  QHSLDTALRR  ARSAPAAAIA   60
ARVAGQTRNI  TVDPRLFKKR  RLRSPRVLFS  TQPPREAADT  QDLDFEVGGA  APFNRTHRSK  120
RSSSHPIFHR  GEFSVCDSVS  VWVGDKTTAT  DIKGKEVMVL  GEVNINNSVF  KQYFFETKCR  180
DPNPVDSGCR  GIDSKHWNSY  CTTTHTFVKA  LTMDGKQAAW  RFIRIDTACV  CVLSRKAVRR  240
A                                                                      241
```

Figure 2: Amino acid sequence of NGF Homo Sapiens.

To mention just two possibilities, some proteins may be building blocks for structures within or between cells, while other ones may be enzymes that enable life-sustaining reactions. The code-set of the proteins consists of pieces of DNA, a string of elements from a set of four 'chemical letters' (*nucleotides*). Three such letters uniquely determine a specific amino acid and hence a string of amino acids is uniquely determined by a sequence of nucleotides, see Alberts [1997]. A DNA string does not have the meaning that the protein counterparts have, for one thing because it has not the specific three dimensional folding.

ACGT-chain: length 1047 base pairs.
www.ebi.ac.uk/cgi-bin/expasyfetch?X52599

```
agagagcgct  gggagccgga  ggggagcgca  gcgagttttg  gccagtggtc  gtgcagtcca    60
aggggctgga  tggcatgctg  gacccaagct  cagctcagcg  tccggaccca  ataacagttt   120
taccaaggga  gcagctttct  atcctggcca  cactgaggtg  catagcgtaa  tgtccatgtt   180
gttctacact  ctgatcacag  cttttctgat  cggcatacag  gcggaaccac  actcagagag   240
caatgtccct  gcaggacaca  ccatccccca  agtccactgg  actaaacttc  agcattccct   300
tgacactgcc  cttcgcagag  cccgcagcgc  cccggcagcg  gcgatagctg  cacgcgtggc   360
ggggcagacc  cgcaacatta  ctgtggaccc  caggctgttt  aaaaagcggc  gactccgttc   420
accccgtgtg  ctgtttagca  cccagcctcc  ccgtgaagct  gcagacactc  aggatctgga   480
cttcgaggtc  ggtggtgctg  ccccccttcaa  caggactcac  aggagcaagc  ggtcatcatc   540
ccatcccatc  ttccacaggg  gcgaattctc  ggtgtgtgac  agtgtcagcg  tgtgggttgg   600
ggataagacc  accgccacag  acatcaaggg  caaggaggtg  atggtgttgg  gagaggtgaa   660
cattaacaac  agtgtattca  aacagtactt  ttttgagacc  aagtgccggg  acccaaatcc   720
cgttgacagc  gggtgccggg  gcattgactc  aaagcactgg  aactcatatt  gtaccacgac   780
tcacaccttt  gtcaaggcgc  tgaccatgga  tggcaagcag  gctgcctggc  ggtttatccg   840
gatagatacg  gcctgtgtgt  gtgtgctcag  caggaaggct  gtgagaagag  cctgacctgc   900
cgacacgctc  cctccccctg  ccccttctac  actctcctgg  gcccctccct  acctcaacct   960
gtaaattatt  ttaaattata  aggactgcat  ggtaatttat  agtttataca  gttttaaaga  1020
atcattattt  attaaatttt  tggaagc                                       1047
```

Figure 3: DNA code of NGF_Homo_Sapiens.

A simple calculation ($1047/3 \neq 241$) shows that not all the letters in the DNA sequence are used. In fact, some proteins (RNA splicing complex) make a selection as to what substring should be used in the decoding toward a new protein.

The first advantage of coding is that DNA is much easier to store and duplicate than the protein itself. The interaction in this example is caused by a modifying effect of the proteins upon the DNA.

The decoding of DNA takes place by making a selection of the code (splicing) and then replacing (for technical reasons) the t's by u's, e different nucleotide. In this way m-RNA is obtained. At the ribosomes the triplets from the alphabet $\{a, c, g, u\}$ in the RNA string are translated into aminoacids following the following table.

	U		C		A		G	
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys
	UUC	Phe	UCC	Ser	UAC	Tyr	UGC	Cys
	UUA	Leu	UCA	Ser	UAA	stop	UGA	stop
	UUG	Leu	UCG	Ser	UAG	stop	UGG	Trp
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg
	CUC	Leu	CCC	Pro	CAC	His	CGC	Arg
	CUA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
	CUG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser
	AUC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
	AUA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
	AUG	Met	ACG	Thr	AAG	Lys	AGG	Arg
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly
	GUC	Val	GCC	Ala	GAC	Asp	GGC	Gly
	GUA	Val	GCA	Ala	GAA	Glu	GGA	Gly
	GUG	Val	GCG	Ala	GAG	Glu	GGG	Gly

A	Ala
C	Cys
D	Asp
E	Glu
G	Gly
F	Phe
H	His
I	Ile
K	Lys
L	Leu
M	Met
N	Asn
P	Pro
Q	Gln
R	Arg
S	Ser
T	Thr
V	Val
W	Trp
Y	Tyr

Figure 4: The 'universal' genetic code and the naming convention for aminoacids. Three codons (UAA, UAG and UGA) code for the end of a protein ('stop').

2. **Language.** The domain of the English language is well-known. It consists of strings of elements of the Roman alphabet extended by the numerals and punctuation marks. This domain has a mechanism of coding, called *quoting* in this context, that is so simple that it may seem superfluous. A string in English, for example

Maria

has as code the quote of that string, i.e.

'Maria'.

In Tarski [1933/1995] it is explained that of the following sentences

1. Maria is a nice girl
2. Maria consists of five letters
3. 'Maria' is a nice girl
4. 'Maria' consists of five letters

the first and last one are meaningful and possibly valid, whereas the second and third are always incorrect, because a confusion of categories has been made (Maria consist of cells, not of letters; 'Maria' is not a girl, but a proper name).

We see the simple mechanism of coding, and its interaction with ordinary language. Again, we see that the codes of the words do not possess the meaning that the words themselves do.

3. **Computable functions.** A third example of reflection comes from computing. The first computers made during WW2 were *ad hoc* machines, each built for a specific use. Since hardware at that time was a huge investment, it was recycled by rewiring the parts after each completed job.

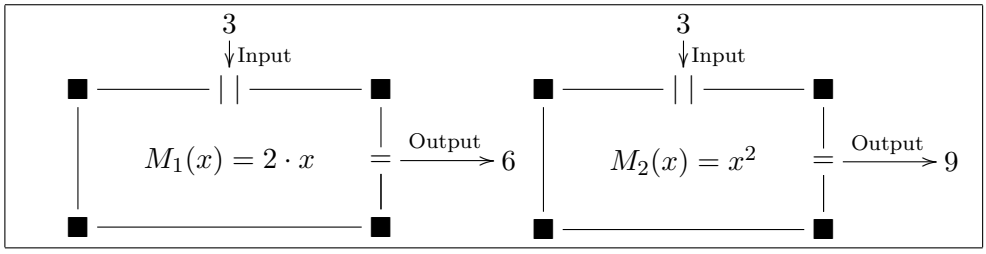


Figure 5: Two *ad hoc* machines: M_1 for doubling and M_2 for squaring.

Based on ideas of Turing this procedure was changed. In Turing [1936] the idea of machines that nowdays are called Turing machines were introduced. These are idealized computers in the sense that they have an unlimited memory, in order to make the theory technology independent. Among these there is the universal Turing machine that can simulate all the other machines via an extra argument, the program.

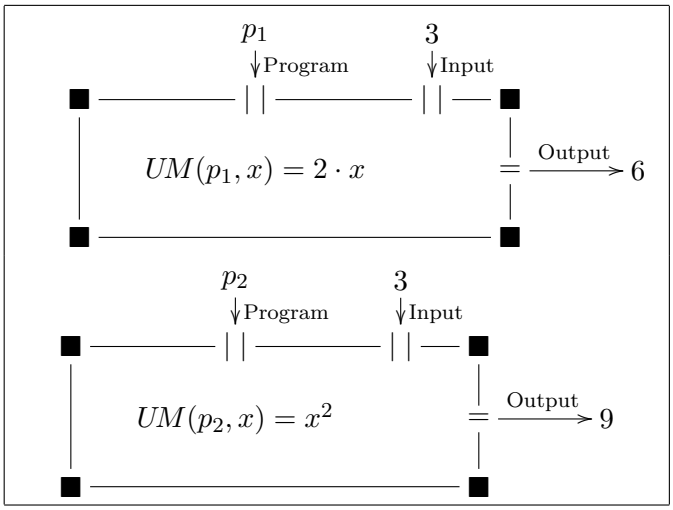


Figure 6: Universal machine UM with programs p_1, p_2 simulating M_1, M_2 respectively.

Under the influence of J. von Neumann, who knew Turings work, one particular computer was constructed, the *universal machine*, and for each particular computing job one had to provide two inputs: the instructions (the software) and the data that this recipe acts upon. This has become the standard for all subsequent computers. So p_1 is a code for M_1 and p_2 for M_2. Since we can

consider $M_1(p_2)$ and $M_2(p_2)$, there is interaction: agents acting on a code, in the second case even their own code.

The domain in this case consists of implemented computable functions, i.e. machines ready for a specific computing job to be performed. A code for an element of this domain consists of a program that simulates the job on a universal machine. The program of a computable function is not yet active, not yet executable in computer science terminology. Only after decoding does a program come into action. Besides coding, interaction is also present. In the universal machine the program and the data are usually kept strictly separate. But this is not obligatory. One can make the program and the input data overlap so that after running for a while on the universal computer, the initial program is modified.

4. **Mathematical theorems.** A final example in this section is concerned with mathematics. A mathematical theorem is usually about numbers or other abstract entities. Gödel introduced codes for mathematical statements and used as code-set the collection $\{0, 1, 2, 3, \ldots\}$ of natural numbers, that do not have any assertive power. As a consequence, one can formulate in mathematics not only statements about numbers, but via coding also about other such statements. There are even statements that speak about themselves. Again we see that both the coding and interaction aspects of reflection are present.

The power of reflection

The mentioned examples of reflection all have quite powerful consequences.

We know how dramatically life has transformed our planet. Life essentially depends on the DNA coding of proteins and the fact that these proteins can modify DNA. This modification is necessary in order to replicate DNA or to proof-read it preventing fatal errors.

One particular species, *homo sapiens*, possesses language. We know its dramatic effects. Reflection using quoting is an essential element in language acquisition. It enables a child to ask questions like: "Mother, what is the meaning of the word 'curious'?"

Reflection in computing has given us the universal machine. Just one design[3] with a range of possibilities through software. This has had a multi-trillion US$ impact on the present stage of the industrial revolution of which we cannot yet see all the consequences.

The effects of reflection in mathematics are less well-known. In this discipline there are statements of which one can see intuitively that they are true, but a formal proof is not immediate. Using reflection, however, proofs using intuition can be replaced by formal proofs[4], see Howe [1992] and Barendregt [1997], pp. 21-23. Formal provability is important for the emerging technology

[3]That there are several kinds of computers on the market is a minor detail: this has to do with speed and user-friendliness.

[4]Often an opposite claim is based on Gödel's incompleteness result. Given a mathematical theory $\mathcal{T}$ containing at least arithmetic that is consistent (expressed as $\mathrm{Con}(\mathcal{T})$), incompleteness states the following. There is a statement G (equivalent to 'G is not provable') within the language of $\mathcal{T}$ that is neither provable nor refutable in $\mathcal{T}$, but nevertheless valid, see Smullyan

of interactive (human-computer) theorem proving and proof verification. Such formal and machine-checked proofs are already changing the way hardware is being constructed[5] and in the future probably also on the way one will develop software. As to the art of mathematics itself, it will bring the technology of Computer Algebra (dealing exactly with equations between symbolic expressions involving elements like $\sqrt{2}$ and π) to the level of arbitrary mathematical statements (involving more complex relations than just equalities between arbitrary mathematical concepts).

The other side of reflection

Anything that is useful and powerful (like fire), can also have a different usage (such as arson). Similarly the power of reflection in the four given examples can be used in different ways.

Reflection in the chemistry of life has produced the species, but also it has as consequence the existence of viruses. Within natural language reflection gives rise to learning a language, but also to paradoxes[6]. The universal computer has as a consequence that there are unsolvable problems, notably the ones we are most interested in[7]. Reflection within mathematics has as a consequence that for almost all interesting consistent axiomatic theories, there are statements that cannot be settled (proved or refuted) within that theory (Gödel's incompleteness result mentioned above).

We see that reflection may be compared to the forbidden fruit: it is powerful, but at the same time, it entails dangers and limitations as well. A proper view of these limitations will make us more modest.

Reflection in spirituality

Insight (*vipassana*) meditation, which stems from classical Buddhism, concerns itself with our consciousness. When impressions come to us through our senses, we obtain a mental representation (e.g. an object in front of us). Now this mental image may be *recollected*: this means that we obtain the awareness of the awareness, also called *mindfulness*. In order to develop the right mindfulness it should be applied to all aspects of consciousness. Parts that usually are not seen as content, but as a coloring of consciousness, become just as important as the object of meditation. If a leg hurts during meditation, one should be mindful of it. Moreover, one learns not only to see the pain, but also the feelings and reactions in connection to that pain. This fine-grained mindfulness will have an 'intuitive analytic' effect: our mind becomes decomposed into its constituents

[1992]. It is easy to show that G is unprovable if $\mathcal{T}$ is consistent, hence by construction G is true. So we have informally proved that G follows from $\mathrm{Con}(\mathcal{T})$. Our (to some unconventional) view on Gödel's theorem is based on the following. By reflection one also can show formally that $\mathrm{Con}(\mathcal{T})\rightarrow G$. Hence it comes not as a surprise, that G is valid on the basis of the assumed consistency. This has nothing to do with the specialness of the human mind, in which we believe but on different grounds, see the section 'Reflection in spirituality'.

[5] Making it much more reliable.

[6] Like 'This sentence is false.'

[7] 'Is this computation going to halt or run forever?' See Yates [1998].

(input, feeling, cognition, conditioning and awareness). Seeing this, we become less subject to various possible vicious circles in our body-mind system that often push us into greed, hatred or compulsive thinking.

Because mindfulness brings the components of consciousness to the open in a disconnected, bare form, they are devoid of their usual meaning. The total information of ordinary mental states can be reconstructed from mindfulness. That is why it works like coding with the contents of our consciousness as domain.

The reflective rôle of mindfulness on our consciousness is quite similar to that of quoting in ordinary language. As proteins can purify part of our DNA, the insight into the constituents of consciousness can purify our mind. Mindfulness makes visible processes within consciousness, hitherto unseen. After that, mindfulness serves as a protection by not letting the components of consciousness exercise their usual meaning. Finally, the presence of mindfulness reorganizes consciousness, giving it a degree of freedom greater than before. Using mindfulness one may act, even if one does not dare; or, one may abstain from action, even if one is urged. Then wisdom will result: morality not based on duty but on virtue. This is the interaction of consciousness and mindfulness. Therefore, by our definition, one can speak of reflection.

This power of reflection via mindfulness also has another side to it. The splitting of our consciousness into components causes a vanishing of the usual view we hold of ourselves and the world. If these phenomena are not accompanied in a proper way, they may become disturbing. But during the intensive meditation retreats the teacher pays proper attention to this. With the right understanding and reorganization, the meditator obtains a new stable balance, as soon as one knows and has incorporated the phenomena.

Mental disorders related to stress can cause similar dissociations. Although the sufferers appear to function normally, to them the world or worse their person does not seem real. This may be viewed as an incomplete and unsystematic use of mindfulness. Perhaps this explains the enigma of why some of the sufferers become 'weller than well', as was observed in Menninger and Pruyser [1963]. These cured patients might very well have obtained the mental purification that is the objective of vipassana meditation.

Pure Consciousness

In Hofstadter [1979] the notion of 'strange loop' is introduced: 'Something that contains a part that becomes a copy of the total when zoomed out. 'Reflection' in this paper is inspired by that notion, but focuses on a special aspect: zooming out in reflection works via the mechanism of coding. The main thesis of Hofstadter is that 'strange loops' are at the basis of self-consciousness. I partly agree with this thesis and would like to add that mindfulness serves as the necessary zooming mechanism in the strange loop of self-consciousness. On the other hand, the thesis only explains the 'self' aspect, the consciousness part still remains obscure. I disagree with the title of Dennet [1993]: 'Consciousness explained'. No matter how many levels of cognition and feedback we place on top of sensory input in a model of the mind, it *a priori* seems not

able to account for experiences. We always could simulated these processes on an old-fashioned computer consisting of relays, or even play it as a social game with cards. It is not that I object to base our consciousness on outer agents like the card players (we depend on nature in a similar way). It is the claimed emergence of consciousness as a side effect of the card game that seems absurd. See Blackmore [2004] for a good survey of theories about consciousness.

Spiritual reflection introduces us to awareness beyond ordinary consciousness, which is without content, but nevertheless conscious. It is called *pure consciousness*. This phenomenon may be explained by comparing our personality to the images on a celluloid film, in which we are playing the title role of our life. Although everything that is familiar to us is depicted on the film, it is in the dark. We need light to see the film as a movie. It may be the case that this pure consciousness is the missing explanatory link between the purely neurophysiological activity of our brain and the conscious mind that we (at least think to) possess. This pure light is believed to transcends the person. The difference between you and me is in the matter (cf. the celluloid of the film). That what gives us awareness is said to come from a common source: the pure consciousness acting as the necessary 'light'.

To understand where this pure consciousness (our inner light) comes from we may have to look better into nature (through a new kind of physics, see e.g. Chalmers [1996] or Stapp [1996]) or better into ourselves (through insight meditation, see e.g. Goldstein [1983]). Probably we will need to do both.

Acknowledgement. Based on the chapter 'Reflection and its use, from Sicence to Meditation', in *Spiritual Information*, edited by Charles L. Harper, Jr., © 2004, Templeton Foundation Press.

2. Languages

Describing and generating languages is an important subject of study.

Formal languages are precisely defined via logical rules. See e.g. Kozen [1997] for a textbook. These languages are introduced for special purposes. For example the programming languages describe algorithms, i.e. calculation recipes, used to make computers do all kinds of (hopefully) useful things. Other formal languages can express properties of software, the so-called specification languages, or properties part of some mathematical theory. Finally, some formal languages are used in order to express proves of properties, for example the proof that a certain program does what you would like it to do[8].

Natural languages occur in many places on earth and are used by people to communicate. Part of linguistics uses ideas of formal languages in order to approach better and better the natural languages. The hope cherished by some is to be able to come up with a formal description of a large part, if not the total, of the natural languages. We will discuss mainly formal languages, giving only a hint how this study is useful for natural ones.

A *language* is a collection of *words*. A word is a string of symbols taken from a predefined *alphabet*. A typical questions are

- Does word w belong to language L?
- Are the languages L and L' equal?

Words over an alphabet

2.1. Definition. (i) An *alphabet* Σ is a set of symbols. Often this set is finite.

(ii) Given an alphabet Σ, a *word* over Σ is a finite sequence $w = s_1 \dots s_n$ of elements $s_i \in \Sigma$. It is allowed that $n = 0$ in which case $w = \epsilon$ the *empty* word.

(iii) We explain the notion of an *abstract syntax* by redefining the collection of words over Σ as follows:

$$\boxed{\texttt{word} := \epsilon \mid \texttt{word}\ s}\ ,$$

where $s \in \Sigma$.

(iv) Σ^* is the collection of all words over Σ.

2.2. Example. (i) Let $\Sigma_1 = \{0, 1\}$. Then

$$1101001 \in \Sigma_1^*.$$

(ii) Let $\Sigma_2 = \{a, b\}$. Then

$$\begin{aligned} abba &\in \Sigma_2^*. \\ abracadabra &\notin \Sigma_2^*. \end{aligned}$$

(iii) $abracadabra \in \Sigma_3^*$, with $\Sigma_3 = \{a, b, c, d, r\}$.

[8] If software is informally and formally specified, tested and proven correct, i.e. that it satisfies the specification it obtains five stars. The informal specification and tests serve to convince the reader that the requirements are correctly stated. There is very little five star software.

(iv) $\epsilon \in \Sigma^*$ for all Σ.

(v) Let $\Sigma_4 = \{\mathtt{A}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{V}, \mathtt{W}, \mathtt{Y}\}$. Then the following is a word in Σ_4^*.

```
MSMLFYTLITAFLIGIQAEPHSESNVPAGHTIPQVHWTKLQHSLDTALRRARSAPAAAIA
ARVAGQTRNITVDPRLFKKRRLRSPRVLFSTQPPREAADTQDLDFEVGGAAPFNRTHRSK
RSSSHPIFHRGEFSVCDSVSVWVGDKTTATDIKGKEVMVLGEVNINNSVFKQYFFETKCR
DPNPVDSGCRGIDSKHWNSYCTTTHTFVKALTMDGKQAAWRFIRIDTACVCVLSRKAVRRA
```

We have encountered it in Fig. 2 of section 1.

(vi) Let $\Sigma_5 = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$. Then an element of Σ_5^* is given in Fig. 3.

(vii) Let $\Sigma_6 = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{u}\}$. Then Σ_6 is "isomorphic to" Σ_5.

Operations on words

2.3. Definition. (i) Let $a \in \Sigma$, $w \in \Sigma^*$. Define $a.w$ 'by induction on w'.

$$\begin{aligned} a.\epsilon &= a \\ a.(us) &= (a.u)s \end{aligned}$$

(ii) If $w, v \in \Sigma^*$, then their *concatenation*

$$w{+}{+}v$$

in Σ^* is defined by induction on v:

$$\begin{aligned} w{+}{+}\epsilon &= w \\ w{+}{+}us &= (w{+}{+}u)s. \end{aligned}$$

We write $wv \equiv w{+}{+}v$ as abbreviation.

(iii) Let $w \in \Sigma^*$. Then $w^\vee$ is w "read backward" and is formally defined by

$$\begin{aligned} \epsilon^\vee &= \epsilon; \\ (wa)^\vee &= a(w^\vee). \end{aligned}$$

For example $(abba)^\vee = abba$, and $(abb)^\vee = bba$.

Languages

2.4. Definition. Let Σ be an alphabet. A *language* over Σ is just a subset $L \subseteq \Sigma^*$ (defined in one way or another).

2.5. Example. Let $\Sigma = \{a, b\}$. Define the following languages over Σ.

(i) $L_1 = \{w \mid w \text{ starts with } a \text{ and ends with } b\}$. Then

$$ab, abbab \in L_1, \text{ but } \epsilon, abba, bab \notin L_1.$$

(ii) $L_2 = \{w \mid abba \text{ is part of } w\}$. Then

$$abba, abbab \in L_2, \text{ but } \epsilon, ab, bab \notin L_2.$$

(iii) $L_3 = \{w \mid aa \text{ is not part of } w\}$. Then

$$aa, abbaab \notin L_3, \text{ but } \epsilon, a, ab, bab \in L_3.$$

(iv) $L_4 = \{\epsilon, ab, aabb, aaabbb, \ldots, a^n b^n, \ldots\}$
$= \{a^n b^n \mid n \geq 0\}$.

Then $\epsilon, aaaabbbb \in L_4$ but $aabbb, bbaa \notin L_4$.

(v) $L_5 = \{w \mid w \text{ is a palindrome, i.e. } w = w^\vee\}$. For example $abba \in L_5$, but $abb \notin L_5$.

Operations on languages

2.6. Definition. Let L, L_1, L_2 be languages over Σ. We define

$$\begin{aligned} L_1 L_2 &= \{w_1 w_2 \in \Sigma^* \mid w_1 \in L_1 \ \&\ w_2 \in L_2\}. \\ L_1 \cup L_2 &= \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}. \\ L^* &= \{w_1 w_2 \ldots w_n \mid n \geq 0 \ \&\ w_1, \ldots, w_n \in L\}. \\ L^+ &= \{w_1 w_2 \ldots w_n \mid n > 0 \ \&\ w_1, \ldots, w_n \in L\}. \end{aligned}$$

Some concrete languages

2.7. Definition. Let $\Sigma_1 = \{\mathsf{M}, \mathsf{I}, \mathsf{U}\}$. Define the language L_1 over Σ_1 by the following grammar, where $x, y \in \Sigma_1^*$.

axiom	MI		
rules	$x\mathsf{I}$	$\Rightarrow$	$x\mathsf{IU}$
	$\mathsf{M}x$	$\Rightarrow$	$\mathsf{M}xx$
	$x\mathsf{III}y$	$\Rightarrow$	$x\mathsf{U}y$
	$x\mathsf{UU}y$	$\Rightarrow$	xy

This means that by definition $\mathsf{MI} \in L_1$;
if $x\mathsf{I} \in L_1$, then also $x\mathsf{IU} \in L_1$,
if $\mathsf{M}x \in L_1$, then also $\mathsf{M}xx$,
if $x\mathsf{III}y \in L_1$, then also $x\mathsf{U}y$,
if $x\mathsf{UU}y \in L_1$, then also xy.

2.8. Example. (i) $\mathsf{MI}, \mathsf{MII}, \mathsf{MU}, \mathsf{MUU}, \mathsf{IMMIU}, \ldots \in \Sigma_1^*$
(ii) $\mathsf{MI}, \mathsf{MIU}, \mathsf{MIUIU}, \mathsf{MII}, \mathsf{MIIII}, \ldots \in L_1$.

2.9. Problem (Hofstadter's MU puzzle[9]). $\mathsf{MU} \in L_1$?
How would you solve this?

2.10. Definition. (i) $\Sigma_2 = \{\mathtt{p}, \mathtt{q}, -\}$
(ii) The language L_2 over Σ_2 is defined as follows ($x, y, z \in \Sigma_2^*$).

axioma's	$x\mathtt{pq}x$ if x consists only of $-$s
rule	$x\mathtt{p}y\mathtt{q}z \quad \Rightarrow \quad x\mathtt{p}y-\mathtt{q}z-$

[9] See Hofstadter [1979].

Languages over a single letter alphabet can be interesting.

2.11. DEFINITION. Laat $\Sigma_4 = \{a\}$.
(i) Define L_{41} as follows.

axiom	a
rule	$w \Rightarrow waa$

Then $L_{41} = \{a^n \mid n \text{ is an odd number}\}$. Here one has $a^0 = \lambda$ and $a^{n+1} = a^n a$. In other words $a^n = \underbrace{a \ldots a}_{n \text{ times}}$.
(ii) $L_{42} = \{a^p \mid p \text{ is a prime number}\}$.
(iii) Define L_{43} as follows.

axiom	a
rule	$w \Rightarrow ww$ $wwwa \Rightarrow w$

How can one decide whether Σ_4 is in L_{41}? The question '$w \in L_{42}$?' is more difficult. The difficulty is partly due to the specification of L_{42}. Language L_{43} has an easy grammar, but a difficult decision problem. For example it requires several steps to show that $aaa \in L_{43}$.
CHALLENGE. Do we have $L_{43} = \{a^n \mid n \geq 1\}$? The first person who sends via email the proof or refutation of this to <henk@cs.kun.nl> will obtain 100 €. Closing time 1.05.2004.

OPEN PROBLEM. (COLLATZ' CONJECTURE) Define L_{44} as follows.

axiom	a
rule	$w \Rightarrow ww$ $wwwaa \Rightarrow wwa$

Prove or refute Collatz' conjecture

$$L_{44} = \{a^n \mid n \geq 1\}.$$

The first correct solution by email before 1.05.2004 earns 150 €. Is there a relation bewteen L_{43} and L_{44}?

Regular languages

Some of the languages of Example2.5 have a convenient notation.

2.12. EXAMPLE. Let $\Sigma = \{a, b\}$. Then
(i) L_1 is denoted by $a(a \cup b)^* b$.
(ii) L_2 is denoted by $(a \cup b)^* abba(a \cup b)^*$.

2.13. DEFINITION. Let Σ be an alphabet.

(i) The *regular expressions* over Σ are defined by the following grammar

$$\mathtt{re} := \emptyset \mid \epsilon \mid \mathtt{s} \mid (\mathtt{re}.\mathtt{re}) \mid (\mathtt{re} \cup \mathtt{re}) \mid \mathtt{re}^*.$$

here s is an element of Σ. A more consice version of this grammar is said to be an *abstract syntax*:

$$\mathtt{re} := \emptyset \mid \epsilon \mid \mathtt{s} \mid \mathtt{re}.\mathtt{re} \mid \mathtt{re} \cup \mathtt{re} \mid \mathtt{re}^*.$$

(ii) Given a regular expression e we define a language $L(e)$ over Σ as follows.

$$\begin{aligned} L(\emptyset) &= \emptyset; \\ L(\epsilon) &= \{\epsilon\}; \\ L(s) &= \{s\}; \\ L(e_1 e_2) &= L(e_1)L(e_2); \\ L(e_1 \cup e_2) &= L(e_1) \cup L(e_2); \\ L(e^*) &= L(e)^* \end{aligned}$$

(iii) A language L over Σ is called *regular* if $L = L(e)$ for some regular expression e.

Note that $L^+ = L.L^*$ so that it one may make use of $^+$ in the formation of regular languages. Without a proof we state the following.

2.14. Proposition. (i) *L_3 of Example 2.5 is regular:*

$$L_3 = L((b \cup ab)^*(a \cup \epsilon)).$$

(ii) *$L_4 = \{a^n b^n \mid n \geq 0\}$ is not regular.* ■

There is a totally different definition of the class of regular languages, namely those that are "accepted by a finite automaton". The definition is not complicated, but beyond the scope of these lectures.

Context-free languages

There is another way to introduce languages. We start with an intuitive example. Consider the following *production system* (grammar) over the alphabet $\Sigma = \{a, b\}$.

$$\boxed{S \;\rightarrow\; \epsilon \mid aSb}$$

This is nothing more or less than the grammar

$$\boxed{\mathtt{exp} \;:=\; \epsilon \mid a\ \mathtt{exp}\ b}$$

The S stands for *start*. With this auxiliary symbol we start. Then we follow the arrow. There are two possibilities: ϵ and aSb. Since the first does not contain the auxiliary symbol any longer, we say that we have reached a terminal state

and therefore the word ϵ has been produced. The second possibility yields aSb, containing again the 'non-terminal' symbol S. Therefore this production has not yet terminated. Continuing we obtain

$$ab = a\epsilon b \text{ and } aaSbb.$$

And then

$$aabb \text{ and } aaaSbbb.$$

Etcetera. Therefore this grammar generates the language

$$L_5 = \{\epsilon, ab, aabb, aaabbb, a^4b^4, \ldots, a^nb^n, \ldots\},$$

also written as

$$L_5 = \{a^nb^n \mid n \geq 0\}.$$

The productions can be depicted as follows.

$$\begin{array}{l} S \to \epsilon; \\ S \to aSb \to ab; \\ S \to aSb \to aaSbb \to aabb; \\ S \to aSb \to aaSbb \to aaaSbbb \to aaabbb. \end{array}$$

L_5 as defined above is called a *contextfree language* and its grammar a *contextfree grammar*

A variant is

$$\boxed{S \quad \to \quad ab \mid aSb}$$

generating

$$L_5' = \{a^nb^n \mid n > 0\}.$$

2.15. Definition. A *contextfree grammar* consists of the following.

(i) An alphabet Σ.

(ii) A set V of *auxiliary*[10] *symbols*. Among them S, the *start symbol*.

(iii) A finite collection *production rules* of the form

$$X \to w$$

where X is an auxiliary symbol and w a word consisting of letters from the alphabet and the auxiliary symbols together; otherwise said $w \in (\Sigma \cup V)^*$, where $\cup$ denotes the union of two sets.

(iv) If there are two production rules with the same auxiliary symbol as its left hand side, for example $X \to w_1$ and $X \to w_2$, then we notate this in the grammar as

$$X \to w_1 \mid w_2.$$

For the auxiliary symbols we use upper case letters like S, A, B. For the elements of the alphabet we use lower case letters like a, b, c etcetera.

[10] These are also called *non-terminal* symbols.

2.16. Example. (i) L_5, L_5' above are contextfree languages. Indeed, the contextfree grammars are given.

(ii) $L_{41} = \{a^n \mid n \text{ odd}\}$ over $\Sigma = \{a\}$ is context-free. Take $V = \{S\}$ and as production-rules

$$\boxed{S \rightarrow aaS \mid a}$$

(iii) Define L_7 over $\Sigma = \{a, b\}$ using $V = \{S, A, B\}$ and the production-rules

$$\boxed{\begin{array}{lll} S & \rightarrow & AB \\ A & \rightarrow & Aa \mid \epsilon \\ B & \rightarrow & Bb \mid \epsilon \end{array}}$$

Then $L_7 = L(a^*b^*)$, i.e. all string a's followed by a string b's.

Note that the auxiliary symbols can be determined from the production- rules.

The name 'Context-free grammars' refers to the fact that the left-hand side of the production rules consist of single auxiliary symbols. (For example the rule $Sa \rightarrow Sab$ is not allowed.) One never needs to look at the context in which the auxiliary symbol is standing.

An important restriction on the context-free grammars consists of the *right-linear* grammars.

2.17. Definition. A *right-linear grammar* is a context-free grammar such that in every production rule

$$X \rightarrow w$$

one has that w is of one of the following forms

(i) $w = \epsilon$

(ii) $w = vY$ with $v \in \Sigma^*$ and Y an auxiliary symbol.

That is to say, in a right-linear grammar auxiliary symbols on the right of a rule only stand at the end and only as a single occurrence.

2.18. Example. (i) In Example 2.16 only L_{41} is a right-linear grammar.

(ii) Sometimes it is possible to transform a context-free grammar in an equivalent right-linear one. The following right-linear grammar (over $\Sigma = \{a, b\}$) also produces L_7 example 2.16 (iii).

$$\boxed{\begin{array}{lll} S & \rightarrow & aS \mid B \\ B & \rightarrow & bB \mid \epsilon \end{array}}$$

Without a proof we state the following.

2.19. Theorem. *Let L be a language over Σ. Then*

$$L \text{ is regular} \quad \Leftrightarrow \quad L \text{ has a right-linear grammar.}$$

Hence every regular language is context-free.

Now we give a grammar for a small part of the English language.

2.20. Definition. Let L_{English} be defined by the following grammar.

$$
\begin{array}{rcl}
S = \langle sentence\rangle & \to & \langle noun-phrase\rangle\langle verb-phrase\rangle. \\
\langle sentence\rangle & \to & \langle noun-phrase\rangle\langle verb-phrase\rangle\langle object-phrase\rangle. \\
\langle noun-phrase\rangle & \to & \langle name\rangle \mid \langle article\rangle\langle noun\rangle \\
\langle name\rangle & \to & John \mid Jill \\
\langle noun\rangle & \to & bicycle \mid mango \\
\langle article\rangle & \to & a \mid the \\
\langle verb-phrase\rangle & \to & \langle verb\rangle \mid \langle adverb\rangle\langle verb\rangle \\
\langle verb\rangle & \to & eats \mid rides \\
\langle adverb\rangle & \to & slowly \mid frequently \\
\langle adjective-list\rangle & \to & \langle adjective\rangle\langle adjective-list\rangle \mid \epsilon \\
\langle adjective\rangle & \to & big \mid juicy \mid yellow \\
\langle object-phrase\rangle & \to & \langle adjective-list\rangle\langle name\rangle \\
\langle object-phrase\rangle & \to & \langle article\rangle\langle adjective-list\rangle\langle noun\rangle
\end{array}
$$

In this language we have for example as element

John slowly rides the yellow bike.

Do exercise 2.10 to produce more sentences and to contemplate what is the alphabet of this language.

Other classes of languages: the Chomsky hierarchy

2.21. Definition. Let Σ be an alphabet.

(i) A *context-sensitive* language over Σ is introduced like a context-free language by production rules of the form

$$uXv \to uwv,$$

where $u, v, w \in \Sigma^*$ and $w \neq \epsilon$. Here X is an auxiliary symbol. The difference between these languages and the context-free ones is that now the production of

$$X \to w$$

only is allowed within the context

$$u \ldots v.$$

(ii) The *enumerable languages* over Σ are also introduced by similar grammars, but now the production rules are of the form

$$uXv \to uwv,$$

where $w = \epsilon$ is allowed.

(iii) A language L over Σ is called *computable* if and only if both L and $\overline{L}$ are enumerable. Here $\overline{L}$ is the complement of L:

$$\overline{L} = \{w \in \Sigma^* \mid w \notin L\}.$$

A typical context-sensitive language is

$$\{a^n b^n c^n \mid n \geq 0\}.$$

A typical computable language is

$$a^p \mid p \text{ is a prime number.}$$

A typical enumerable language is L_{44}.

The following families of languages are strictly increasing:

1. The regular languages;
2. The context-free languages;
3. The context-sensitive languages;
4. The computable languages;
5. The enumerable languages.

Let us abbreviate these classes of languages as RL,CFL, CSL, CL, EL, respectively. Then we have the proper inclusions can be depicted in the following diagram.

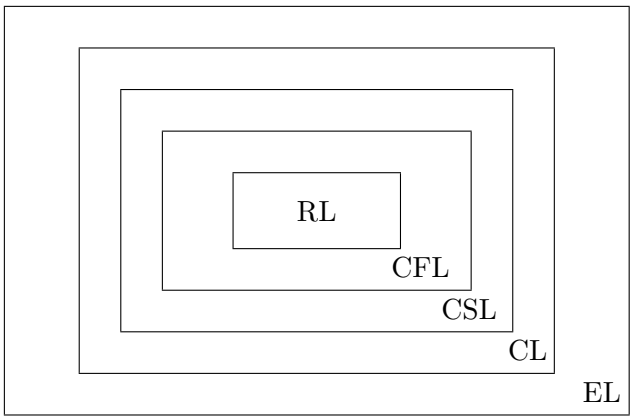


Figure 7: The Chomsky hierarchy

In Chomsky [1956] the power of these definition mechanisms is discussed for the generation of natural languages. He argues that a natural language is too complex to be described by a context-free grammar. Moreover, Chomsky argues that the computable and enumerable languages are too complex to be able to learn by three year old children. The open problem of linguistics is whether a natural language can be described as a context-sensitive language.

Reflection over the classes of languages

There is a uniform way to describe the regular languages. By definition a language L is regular if and only if there is a regular expression e such that $L = L(e)$. This set of regular expressions is itself not a regular language. Reflection over the regular languages pushes us outside this class.

2.22. DEFINITION. (i) A *universal notation system* for the regular languages over an alphabet Σ consists of a language L_u over an alphabet Σ_u and a decoding $d : L_u \to \{L \mid L \text{ is regular}\}$, such that for every regular language L there is at least one code c such that $d(c) = L$.

(ii) Such a universal coding system is called *regular* if the language

$$\{cv \mid v \in d(c)\}$$

over the alphabet $\Sigma \cup \Sigma_u$ is regular.

2.23. PROPOSITION. *(V. Capretta) There is no regular universal notation system for the regular languages.*

We will not give a proof, as it requires some knowledge about the regular languages.

A similar negative result is probably also valid for the context-free and context-sensitive languages. We know that this negative result is the case for the computable languages. But for the enumerable languages there does exists a notation system that itself is enumerable.

2.24. DEFINITION. (i) A *universal notation system* for the enumerable languages over an alphabet Σ consists of a language L_u over an alphabet Σ_u and a decoding $d : L_u \to \{L \mid L \text{ is enumerable}\}$, such that for every enumerable language L there is at least one code c such that $d(c) = L$.

(ii) Such a universal coding system is called *enumerable* if the language

$$\{cv \mid v \in d(c)\}$$

over the alphabet $\Sigma \cup \Sigma_u$ is enumerable.

2.25. PROPOSITION. *There is an enumerable universal notation system for the enumerable languages.*

PROOF. (Sketch) The reason is that the enumerable languages are those languages that are accepted by a Turing machine. Turing machines take as input a string w and start a computation, that can halts or not. Now L is enumerable if and only if there is a Turing machine M_L such that

$$w \in L \;\Leftrightarrow\; M_L(w) \text{ halts.}$$

There is an universal Turing machine M_u, see section 1. This means that for every Turing machine M there is a code c_M such that

$$M(w) = M_u(c_M w).$$

Define $f(c) = \{w \mid M(cw) \text{ halts}\}$. Then given an enumerable language L one has

$$\begin{aligned} w{\in}L &\Leftrightarrow M_L(w) \text{ halts} \\ &\Leftrightarrow M(c_{M_L} w) \text{ halts} \\ &\Leftrightarrow w{\in}d(c_{M_L}), \end{aligned}$$

hence

$$L = d(c_L).$$

Therefore $L_u = \{c_M \mid M \text{ a Turing machine}\}$ with decoding d is a universal notation mechanism for the enumerable languages. Moreover, the notation system is itself enumerable:

$$\{cw \mid w{\in}d(c)\} = \{cw \mid M_u(cw) \text{ halts}\},$$

which is the languages accepted by L_{M_u} and hence enumerable. ■

We end this section by observing that the reflection of the enumerable languages is a different from the one that is present in the natural language like English, see section 1. The first one has as domain the collection of enumerable languages; the second one has as domain the collection of strings within a language. We will encounter this second form of reflection quite precisely in section 5.

Exercises

2.1. (i) Show that $\mathsf{MUI}{\in}L_1$
(ii) Show that $\mathsf{IMUI} \notin L_1$

2.2. Which words belong to L_2? Motivate your answers.

1. $--\mathtt{p}--\mathtt{p}--\mathtt{q}----$
2. $--\mathtt{p}--\mathtt{q}--\mathtt{q}----$
3. $--\mathtt{p}--\mathtt{q}----$
4. $--\mathtt{p}--\mathtt{q}------$

2.3. Let $\Sigma_3 = \{a, b, c\}$. Define L_3 by

axiom	ab		
rule	xyb	$\Rightarrow$	$yybx$

The following should be answered by 'yes' or 'no', plus a complete motivation why this is the right answer.
(i) Do we have $ba{\in}L_3$?

(ii) Do we have $bb \in L_3$?

2.4. (i) Show that $\epsilon \notin L_{44}$.
(ii) Show that $aaa \in L_{44}$.

2.5. Given is the grammar

$$\begin{array}{lcl} S & \rightarrow & aSb \mid A \mid \epsilon \\ A & \rightarrow & aAbb \mid abb \end{array}$$

This time $V = \{S, A\}$ and $\Sigma = \{a, b\}$.
Can one produce abb and aab?
What is the collection of words in the generated language?

2.6. Produce the language of 2.5 with axioms and rules as in 2.7.

2.7. (i) Consider the context-free grammar over $\{a, b, c\}$

$$\begin{array}{lcl} S & \rightarrow & A \mid B \\ A & \rightarrow & abS \mid \epsilon \\ B & \rightarrow & bcS \mid \epsilon \end{array}$$

Which of the following words belong to the corresponding language L_8?

$$abab, bcabbc, abba.$$

(ii) Show that L_8 is regular by giving the right regular expression.
(iii) Show that L_8 has a right-linear grammar.

2.8. Let $\Sigma = \{a, b\}$.
(i) Show that $L_{41} = \{a^n \mid n \text{ is odd}\}$, see 2.16, is regular. Do this both by providing a regular expression e such that $L_{41} = L(e)$ and by providing a right-linear grammar for L_{41}.
(ii) Describe the regular language $L(a(ab^*)^*)$ by a context-free grammar.
(iii) Let L_9 consists of words of the form

$$aba \ldots aba$$

(i.e. a's b's alternating, starting with an a and ending with one; a single a is also allowed). Show in two ways that L_9 is regular.

2.9. Let $\Sigma = \{a, b, c\}$. Show that

$$L = \{w \in \Sigma^* \mid w \text{ is a palindrome}\}$$

is context-free.

2.10. (i) Show how to produce in L_{English} the sentence

Jill frequently eats a big juicy yellow mango.

(ii) Is the generated language context-free?
(iii) Is this grammar right-linear?
(iv) Produce some sentences of your own.
(v) What is the alphabet Σ?
(vi) What are the auxiliary symbols?
(vii) Extend the alphabet with a symbol for 'space' and adjust the grammar accordingly.

3. Combinatory Logic

In this section we introduce the theory of combinators, also called *Combinatory Logic* or just **CL**. It was introduced by Schönfinkel [1924] and Curry [1930].

3.1. Definition. We define the following finite alphabet

$$\Sigma_{\mathbf{CL}} = \{\mathbf{I}, \mathbf{K}, \mathbf{S}, x, ',), (\}$$

3.2. Definition. We introduce two simple regular grammars over $\Sigma_{\mathbf{CL}}$.

(i) $\boxed{\texttt{constant} := \mathbf{I} \mid \mathbf{K} \mid \mathbf{S}}$

(ii) $\boxed{\texttt{variable} := x \mid \texttt{variable}'}$

Therefore we have the following collection of variables: $x, x', x'', x''', \ldots$.

3.3. Notation. We use

$$x, y, z, \ldots, x', y', z', \ldots, x_0, y_0, z_0, \ldots, x_1, y_1, z_1, \ldots$$

to denote arbitrary variables.

3.4. Definition. We now introduce the main (conextfree-)language over $\Sigma_{\mathbf{CL}}$ consisting of **CL**-terms (that we may call simply "terms" if there is little danger of confusion).

$$\boxed{\texttt{term} := \texttt{constant} \mid \texttt{variable} \mid (\texttt{term}\,\texttt{term})}$$

3.5. Definition. A *combinator* is a term without variables.

The intended interpretation of (PQ), where P, Q are terms is: P considered as function applied to Q considered as argumnent. These functions are allowed to act on any argument, including themselves!

3.6. Proposition. (i) *Binary functions can be simulated by unary functions.*
(ii) *Functions with n-arguments can be simulated by unary functions.*

Proof. (i) Consider $f(x, y)$, if you like $f(x, y) = x^2 + y$. Define

$$\begin{aligned} F_x(y) &= f(x, y); \\ F(x) &= F_x. \end{aligned}$$

Then

$$\begin{aligned} F(x)(y) &= F_x(y) \\ &= f(x, y). \end{aligned}$$

(ii) Similarly. ■

In order to economize on parentheses we will write $F(x)(y)$ as Fxy giving it the intended interpretation $(Fx)y$ (association to the left).

3.7. Example. Examples of **CL**-terms.

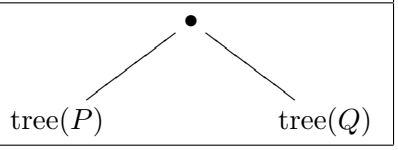
(i) $\mathbf{I}$;
(ii) x;
(iii) $(\mathbf{I}x)$;
(iv) $(\mathbf{IK})$;
(v) $(\mathbf{KI})$;
(vi) $((\mathbf{KI})\mathbf{S})$;
(vii) $(\mathbf{K}(\mathbf{IS}))$;
(viii) $(\mathbf{K}(\mathbf{I}(\mathbf{SS})))$;
(ix) $(((\mathbf{KI})\mathbf{S})\mathbf{S})$;
(x) $(((\mathbf{S}x)y)z)$;
(xi) $(x(\mathbf{S}(y\mathbf{K})))$.

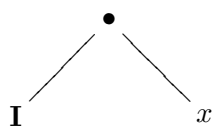
3.8. Notation. We use

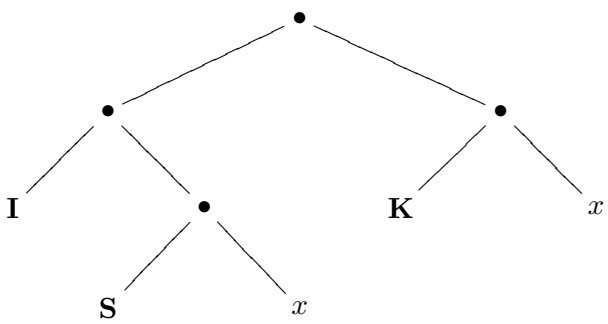
$$M, N, L, \ldots, P, Q, R, \ldots, X, Y, Z, \ldots$$

to denote arbitrary terms. Also c to denote a constant $\mathbf{I}, \mathbf{K}$ or $\mathbf{S}$.

3.9. Definition. For a term M we define a "tree" denoted by tree(M).

M	tree(M)
x	x
c	c
(PQ)	

3.10. Example. (i) tree($\mathbf{I}x$) = 

(ii) tree($(\mathbf{I}(\mathbf{S}x))(\mathbf{K}x)$) = 

3.11. NOTATION. In order to save parentheses we make the following convention ("association to the left"):

$$PQ_1 \dots Q_n \text{ stands for } (..((PQ_1)Q_2)\dots Q_n).$$

For example $\mathbf{K}PQ$ stands for $((\mathbf{K}P)Q)$ and $\mathbf{S}PQR$ for $(((\mathbf{S}P)Q)R)$. We also write this as

$$\begin{aligned} PQ_1 \dots Q_n &\equiv (..((PQ_1)Q_2)\dots Q_n); \\ \mathbf{K}PQ &\equiv ((\mathbf{K}P)Q); \\ \mathbf{S}PQR &\equiv (((\mathbf{S}P)Q)R). \end{aligned}$$

The **CL**-terms are actors, acting on other terms.

3.12. DEFINITION. We postulate the following (here P, Q, R are arbitrary)

$\mathbf{I}P$	$=$	P	$(\mathbf{I})$
$\mathbf{K}PQ$	$=$	P	$(\mathbf{K})$
$\mathbf{S}PQR$	$=$	$PR(QR)$	$(\mathbf{S})$

Figure 8: Axioms of **CL**.

If $P = Q$ can be derived from these axioms we write $P =_{\mathbf{CL}} Q$. (If there is little danger of confusion we will write simply $P = Q$.)

We will show some consequences of the axioms.

3.13. PROPOSITION. (i) $\mathbf{SKK}x =_{\mathbf{CL}} x$.
(ii) $\mathbf{SKK}P =_{\mathbf{CL}} P$.
(iii) $\mathbf{KI}xy =_{\mathbf{CL}} y$.
(iv) $\mathbf{SK}xy =_{\mathbf{CL}} y$.

PROOF. (i) $\mathbf{SKK}x = \mathbf{K}x(\mathbf{K}x)$, by axiom $(\mathbf{S})$
$= x$, by axiom $(\mathbf{K})$.
(ii) $\mathbf{SKK}P = \mathbf{K}P(\mathbf{K}P)$, by axiom $(\mathbf{S})$
$= P$, by axiom $(\mathbf{K})$.
(iii) $\mathbf{KI}xy = \mathbf{I}y$, by axiom $(\mathbf{K})$
$= y$, by axiom $(\mathbf{I})$.
(iv) $\mathbf{SK}xy = \mathbf{K}y(xy)$, by axiom $(\mathbf{S})$
$= y$, by axiom $(\mathbf{K})$. ■

The second result shows that we did not need to introduce **I** as a primitive constant, but could have defined it as **SKK**. Results (iii) and (iv) show that different terms can have the same effect. Comparing result (i) and (ii) one sees that variables serve as generic terms. If an equation holds for a variable, it holds for all terms. This gives the following principle of substitution.

3.14. DEFINITION. Let M, L be terms and let x be a variable. The result of *substitution* of L for x in M, notation

$$M[x := L]$$

is defined by recursion on M.

M	$M[x{:=}L]$
x	L
y	y, provided $x \not\equiv y$
c	c
PQ	$(P[x{:=}L])(Q[x{:=}L])$

3.15. Proposition. *If* $P_1 =_{\mathbf{CL}} P_2$, *then* $P_1[x{:=}Q] =_{\mathbf{CL}} P_2[x{:=}Q]$.

3.16. Example.

$$\begin{aligned} x\mathbf{I}[x{:=}\mathbf{S}] &\equiv \mathbf{SI}. \\ \mathbf{KI}xx[x{:=}\mathbf{S}] &\equiv \mathbf{KISS}. \\ \mathbf{KI}yx[x{:=}\mathbf{S}] &\equiv \mathbf{KI}y\mathbf{S}. \end{aligned}$$

Now we will perform some magic.

3.17. Proposition. (i) *Let* $\mathbf{D} \equiv \mathbf{SII}$. *Then (doubling)*

$$\mathbf{D}x =_{\mathbf{CL}} xx.$$

(ii) *Let* $\mathbf{B} \equiv \mathbf{S}(\mathbf{KS})\mathbf{K}$. *Then (composition)*

$$\mathbf{B}fgx =_{\mathbf{CL}} f(gx).$$

(iii) *Let* $\mathbf{L} \equiv \mathbf{D}(\mathbf{BDD})$. *Then (self-doubling, life!)*

$$\mathbf{L} =_{\mathbf{CL}} \mathbf{LL}.$$

Proof. (i)
$$\begin{aligned} \mathbf{D}x &\equiv \mathbf{SII}x \\ &= \mathbf{I}x(\mathbf{I}x) \\ &= xx. \end{aligned}$$

(ii)
$$\begin{aligned} \mathbf{B}fgx &\equiv \mathbf{S}(\mathbf{KS})\mathbf{K}fgx \\ &= \mathbf{KS}f(\mathbf{K}f)gx \\ &= \mathbf{S}(\mathbf{K}f)gx \\ &= \mathbf{K}fx(gx) \\ &= f(gx). \end{aligned}$$

(iii)
$$\begin{aligned} \mathbf{L} &\equiv \mathbf{D}(\mathbf{BDD}) \\ &= \mathbf{BDD}(\mathbf{BDD}) \\ &= \mathbf{D}(\mathbf{D}(\mathbf{BDD})) \\ &\equiv \mathbf{DL} \\ &= \mathbf{LL}. \ \blacksquare \end{aligned}$$

We want to understand and preferably also to control this!

Exercises

3.1. Rewrite to a shorter form **KISS**.

3.2. Rewrite **SIDI**.

3.3. Draw tree(**L**).

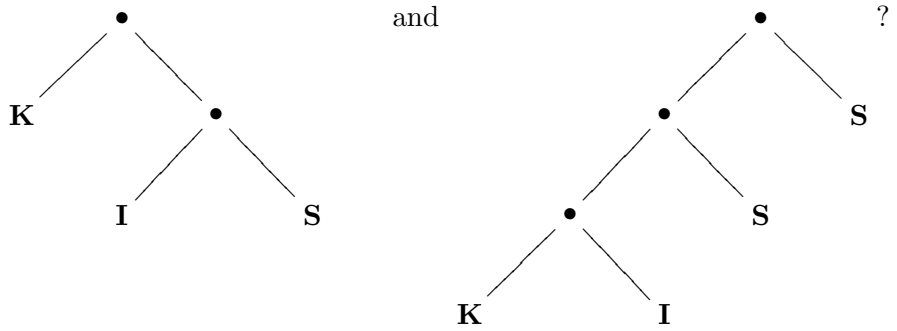
3.4. Given the terms **I**, $x\mathbf{I}$, $y\mathbf{S}$ and $xy\mathbf{K}$.
(i) Choose from these terms P, Q, R such that

$$P[x:=Q][y:=R] \equiv P[x:=R][y:=Q] \not\equiv P.$$

(ii) Now choose other terms P, Q, R such that

$$P[x:=Q][y:=R] \not\equiv P[x:=R][y:=Q].$$

3.5. Which of the terms in Example 3.7 have as trees respectively



3.6. Construct a combinator F such that (F says: "You do it with **S**")

$$Fx =_{\mathbf{CL}} x\mathbf{S}.$$

3.7.$^{+}$ Construct a combinator C such that ("You do it with the next two in reverse order")

$$\mathsf{C}xyz =_{\mathbf{CL}} xzy.$$

3.8.* Construct a combinator G such that (G says: "You do it with me")

$$Gx =_{\mathbf{CL}} xG.$$

3.9.* Construct a combinator O (the "Ogre": 'I eat everything!') such that

$$Ox =_{\mathbf{CL}} O.$$

3.10.* Construct a combinator P (the "Producer") such that

$$P =_{\mathbf{CL}} P\mathbf{I}.$$

$^{+}$ More difficult.
$*$ Needs theory from later sections.

4. Lambda calculus

The intended meaning of an expression like

$$\lambda x.3x$$

is the function

$$x \longmapsto 3x$$

that assigns to x (that we think for this explanation to be a number) the value $3x$ (3 times x). So according to this intended meaning we have

$$(\lambda x.3x)(6) = 18.$$

The parentheses around the 6 are usually not written: $(\lambda x.3x)6 = 18$. But this example requires that we have numbers and multiplication. The lambda calculus, to be denoted by λ, does not require any pre-defined functions at all. The system was introduced in Church [1932] and in an improved way in Church [1936]. For a modern overview, see Barendregt [1984].

4.1. Definition. We define the following finite alphabet

$$\Sigma_\lambda = \{x, ', \lambda,), (\}$$

4.2. Definition. We introduce the following grammar over Σ_λ.

$$\texttt{variable} := x \mid \texttt{variable}'$$

Therefore we have the following collection of variables: $x, x', x'', x''', \ldots$.

4.3. Definition. We now introduce the main language over Σ_λ consisting of λ-terms (that we may call simply "terms" if there is little danger of confusion).

$$\boxed{\texttt{term} := \texttt{variable} \mid (\lambda\texttt{variable}\ \texttt{term}) \mid (\texttt{term}\ \texttt{term})}$$

4.4. Notation. We use for λ the same convention for naming arbitrary variables and terms as for **CL**: $x, y, z, \ldots$ and $M, N, L, \ldots$ denote arbitrary variables and terms respectively. $M \equiv N$ means that M, N are literally the same lambda term.

4.5. Notation. Again we use association to the left to save parentheses:

$$PQ_1 \ldots Q_n \equiv (..((PQ_1)Q_2) \ldots Q_n).$$

Another saving of parentheses is by "associating to the right":

$$\lambda x_1 \ldots x_n.M \equiv (\lambda x_1(\lambda x_2(..(\lambda x_n(M))..)))).$$

Outer parentheses are omitted (or added) for higher readability. For example

$$(\lambda x.x)y \equiv ((\lambda x x)y).$$

4.6. Convention. Consider the term $x.x$. This will have the following effect

$$(\lambda x.x)P = P.$$

But also one has

$$(\lambda y.y)P = P.$$

It will be useful to identify $\lambda x.x$ and $\lambda y.y$, writing

$$\lambda x.x \equiv \lambda y.y.$$

We say that these two terms are equal up to naming of *bound variables*. Note

$$x(\lambda x.x) \not\equiv y(\lambda y.y),$$

as the first x is not a bound but a *free* occurrence of this variable. We will take care to use different names for free and bound variables, hence will not use $x(\lambda x.x)$ but rather $x(\lambda y.y)$.

4.7. Definition. We postulate for λ-terms

$$(\lambda x.M)N = M[x{:=}N] \qquad (\beta\text{-rule})$$

Here we only substitute for the free (occurences of the) variable x. If we can derive from this axiom that $M = N$, we write $M =_\lambda N$ (or simply $M = N$).

4.8. Proposition. *Define* $\mathsf{I} \equiv \lambda p.p$; $\mathsf{K} \equiv \lambda pq.p$; $\mathsf{S} \equiv \lambda pqr.pr(qr)$. *Then* $\mathsf{I}P =_\lambda P$; $\mathsf{K}PQ =_\lambda P$; $\mathsf{S}PQR =_\lambda PR(QR)$.

Proof.

$$\begin{array}{rcl}
\mathsf{I}P & \equiv & (\lambda p.p)P \\
& =_\lambda & p[p{:=}P] \\
& \equiv & P; \\
\mathsf{K}PQ & \equiv & (\lambda p(\lambda q.p))PQ \\
& =_\lambda & (\lambda q.p)[p{:=}P]Q \\
& =_\lambda & (\lambda q.P)Q \\
& =_\lambda & P[q{:=}Q] \\
& \equiv & P. \\
\mathsf{S}PQR & \equiv & (\lambda pqr.pr(qr))PQR \\
& = & (\lambda qr.pr(qr))[p{:=}P]QR \\
& \equiv & (\lambda qr.Pr(qr))QR \\
& = & (\lambda r.Pr(qr))[q{:=}Q]R \\
& \equiv & (\lambda r.Pr(Qr))R \\
& = & (Pr(Qr))[r{:=}R] \\
& \equiv & PR(QR). \ \blacksquare
\end{array}$$

4.9. Definition. Let M be a term, $\vec{x} = x_1, \ldots, x_n$ a list of variables and $\vec{N} = N_1, \ldots, N_n$ a coresponding list of terms. Then the result of *simultaneously*

substituting $\vec{N}$ *for* $\vec{x}$ *in* M, notation $M[\vec{x}:=\vec{N}]$, is defined as follows.

M	$M[\vec{x}:=\vec{N}]$
x_i	N_i
y	y, provided y is not one of the x_i's
PQ	$(P[\vec{x}:=\vec{N}])(Q[\vec{x}:=\vec{N}])$
$\lambda y.P$	$\lambda y.(P[\vec{x}:=\vec{N}])$

4.10. Proposition. (i) $M[x:=x]\equiv M$.

(ii) *Suppose the variable* y *is not in* $\vec{N}$. *Write* $P^*\equiv P[\vec{x}:=\vec{N}]$. *Then*

$$(P[y:=Q])^*\equiv P^*[y:=Q^*].$$

(iii) $M_1=_\lambda M_2 \Rightarrow M_1[\vec{x}:=\vec{N}]=_\lambda M_2[\vec{x}:=\vec{N}]$.

(iv) $(\lambda\vec{x}.M)\vec{x}=_\lambda M$.

(v) $(\lambda\vec{x}.M)\vec{N}=_\lambda M[\vec{x}:=\vec{N}]$.

Proof. (i) Obvious. Or by induction

M	$M[x:=x]$
x	$x[x:=x]\equiv x$
y	$y[x:=x]\equiv y$
PQ	$(P[x:=x])(Q[x:=x])\equiv PQ$
$\lambda x.P$	$(\lambda z.P)[x:=x]\equiv\lambda z.(P[x:=x])\equiv\lambda z.P$

(ii) By induction on P.

P	$P[y:=Q]$	$(P[y:=Q])^*$	$P^*[y:=Q^*]$
y	Q	Q^*	Q^*
x_i	$x_i[y:=Q]\equiv x_i$	$x_i^*\equiv N_i$	$N_i[y:=Q]\equiv N_i$
z	$z\ (\neq y,x_i)$	z	z
P_1P_2	$P_1[y:=Q](P_2[y:=Q])$	$(P_1[y:=Q])^*(P_2[y:=Q])^*\equiv$ $(P_1^*[y:=Q^*])(P_2^*[y:=Q^*])$	$(P_1^*P_2^*)[y:=Q^*]$
$\lambda z.P_1$	$\lambda z.(P_1[y:=Q])$	$\lambda z.(P_1^*[y:=Q^*])$	$(\lambda z.P_1)^*[y:=Q^*]\equiv$ $(\lambda z.P_1^*)[y:=Q^*]$ $\lambda z.(P_1^*[y:=Q^*])$

(iii) Given a proof of $M_1=_\lambda M_2$ we can perform in all equations the simultaneous substitution $[\vec{x}:=\vec{N}]$. For an axiom (the β-rule) we use (ii):

$$\begin{aligned}((\lambda x.P)Q) &\equiv (\lambda x.P^*)Q^*\\ &= P^*[y:=Q^*]\\ &\equiv (P[y:=Q])^*, \qquad \text{by (ii).}\end{aligned}$$

(iv) We do this for $\vec{x}=x_1,x_2$.

$$\begin{aligned}(\lambda\vec{x}.M)\vec{x} &\equiv (\lambda x_1(\lambda x_2.M))x_1x_2\\ &\equiv (\lambda x_1x_2.M)x_1x_2\\ &\equiv ((\lambda x_1(\lambda x_2.M))x_1)x_2\\ &= (\lambda x_2.M)x_2\\ &= M.\end{aligned}$$

(v) By (iv) $(\lambda\vec{x}.M)\vec{x} = M$ and hence $(\lambda\vec{x}.M)\vec{N} = M[x := \vec{N}]$, by (iii). ■

4.11. THEOREM (Fixedpoint theorem). *Let F be a lambda term. Define $X \equiv WW$ with $W \equiv \lambda x.F(xx)$. Then*

$$FX = X.$$

Hence every term F has a fixed-point.

PROOF. We have

$$\begin{aligned} X &\equiv WW \\ &\equiv (\lambda x.F(xx))W \\ &= F(WW) \\ &\equiv FX. \blacksquare \end{aligned}$$

4.12. APPLICATION. There exists a self-doubling lambda term D:

$$D = DD.$$

PROOF. Let $F \equiv \lambda x.xx$. Now apply the fixedpoint theorem: there exists a term X suxh that $FX = XX$. Taking $D \equiv X$ one has

$$\begin{aligned} D &\equiv X \\ &= FX \\ &= XX \\ &\equiv DD. \blacksquare \end{aligned}$$

4.13. APPLICATION. There exists an 'ogre' O, a lambda term monster that eats everything (and does not even get any fatter):

$$OM = O,$$

for all terms M.

PROOF. Define O to be the fixedpoint of K. Then $O = \mathsf{K}O$, hence

$$\begin{aligned} OM &= \mathsf{K}OM \\ &= O. \blacksquare \end{aligned}$$

How do we do this systematically? We would like a solution for X in an expression like

$$Xabc = bX(c(Xa)).$$

At the left hand side X is the 'boss'. At the right hand side things that happen depend on the arguments of X. We solve this as follows.

$$\begin{aligned} Xabc = bX(c(Xa)) &\Leftarrow X = \lambda abc.bX(c(Xa)) \\ &\Leftarrow X = (\lambda xabc.bx(c(xa)))X \\ &\Leftarrow X \text{ is fixedpoint of } \lambda xabc.bx(c(xa)). \end{aligned}$$

The consequence is that for this X one has for all A, B, C

$$XABC = BX(C(XA)).$$

In order to formulate a general theorem, we like to abbreviate an expression like $bX(c(Xa))$ as $C[a, b, c, X]$. Such an expression $C[\ldots]$ is called a *context*. An expression like $C[y_1, \ldots, y_n, x]$ will be abbreviated as $C[\vec{y}, x]$.

4.14. THEOREM. *Given a context $C[\vec{y}, x]$. Then there exists a term X such that*

$$X\vec{y} = C[\vec{y}, X].$$

Hence for all terms $\vec{P} = P_1, \ldots, P_n$

$$X\vec{P} = C[\vec{P}, X].$$

PROOF. Define $F \equiv \lambda x\vec{y}.C[\vec{y}, x]$ and let $FX = X$. Then

$$\begin{array}{rcll} X\vec{y} & = & FX\vec{y} & \\ & \equiv & (\lambda x\vec{y}.C[\vec{y}, x])X\vec{y}, & \text{by definition of } F, \\ & = & C[\vec{y}, X]. \blacksquare & \end{array}$$

4.15. APPLICATION. There exists a 'slave' term P such that for all its arguments A

$$PA = AP.$$

PROOF. Take $C[a, p] = ap$. Then

$$PA = C[A, P] = AP. \blacksquare$$

4.16. DEFINITION. (i) A lambda term of the form $(\lambda x.M)N$ is called a *redex*.
(ii) A lambda term without a redex as part is said to be *in normal form* (nf).
(iii) A lambda term M *has a nf* if $M =_\lambda N$ and N is in nf.

It is the case that every lambda term has at most one nf. We omit the details of the proof. See e.g. Barendregt [1984] for a proof.

4.17. FACT. Let M be a lambda term. Then M may or may not have a nf. If it does have one, then that nf is unique.

SK has as nf $\lambda xy.y$. The term $\Omega \equiv \omega\omega$ with $\omega \equiv \lambda x.xx$ does not have a nf.

4.18. REMARK. In writing out one should restore brackets:

$$\begin{array}{rcl} \mathsf{K}\Omega & = & \mathsf{K}(\omega\omega) \\ & = & \mathsf{K}((\lambda x.xx)(\lambda x.xx)) \\ & = & \lambda y.\Omega \end{array}$$

The following is wrong: $\mathsf{K}\Omega \equiv \mathsf{K}\omega\omega = \omega$ (**do not memorize**!).

Simulating lambda terms with combinators

4.19. Definition. Let P be a CL-term. Define the CL-term $\lambda^* x.P$ as follows.

P	$\lambda^* x.P$
x	$\mathbf{I}$
y	$\mathbf{K}y$
c	$\mathbf{K}c$
QR	$\mathsf{S}(\lambda^* x.Q)(\lambda^* x.R)$

4.20. Theorem. *For all CL-terms P one has*

$$(\lambda^* x.P)x =_{CL} P.$$

Hence

$$(\lambda^* x.P)Q =_{CL} P[x := Q].$$

Proof.

P	$\lambda^* x.P$	$(\lambda^* x.P)x$
x	$\mathbf{I}$	$\mathbf{I}x = x$
y	$\mathbf{K}y$	$\mathbf{K}yx = y$
c	$\mathbf{K}c$	$\mathbf{K}cx = c$
QR	$\mathsf{S}(\lambda^* x.Q)(\lambda^* x.R)$	$\mathsf{S}(\lambda^* x.Q)(\lambda^* x.R)x =$ $(\lambda^* x.Q)x((\lambda^* x.R)x) =$ QR

The second result now follows from Porposition 3.15. ■

Now the attentive reader can solve exercises 3.7-3.10

4.21. Definition. (i) Now we can translate a lambda term M into a CL-term $M_{\mathbf{CL}}$ as follows.

M	$M_{\mathbf{CL}}$
x	x
PQ	$P_{\mathbf{CL}}Q_{\mathbf{CL}}$
$\lambda x.P$	$\lambda^* x.P_{\mathbf{CL}}$

(ii) A translation in the other direction, from a **CL**-term P to a lambda term P_λ, is as follows.

P	P_λ
x	x
$\mathbf{I}$	I
$\mathbf{K}$	K
$\mathbf{S}$	S
PQ	$P_{\mathbf{CL}}Q_{\mathbf{CL}}$

This makes it possible to translate results from **CL**-terms to lambda terms and vice versa. The self-reproducing **CL**-term $\mathbf{L}$ with $\mathbf{L} = \mathbf{LL}$ can be found by first constructing it in lambda calculus.

Exercises

4.1. Write $\langle \vec{a} \rangle \equiv \lambda z.z\vec{a}$. For example, $\langle a \rangle \equiv \lambda z.za$, $\langle a, b \rangle \equiv \lambda z.zab$. Simplify
(i) $\langle a \rangle \langle b \rangle$;
(ii) $\langle a \rangle \langle b, c \rangle$;
(iii) $\langle a, b \rangle \langle c \rangle$;
(iv) $\langle a, b \rangle \langle c, d \rangle$;
(v) $\langle \langle a, b, c \rangle \rangle \langle f \rangle$.

4.2. Simplify
(i) $(\lambda a.aa)(\lambda b.(\lambda cd.d)(bb))$;
(ii) $(\lambda abc.bca)cab$.

4.3. Construct a λ-term A such that for all variables b one has

$$Ab = AbbA.$$

4.4. Find two λ-terms B such that for all variables a one has

$$Ba = BaaB.$$

[Hint. One can use 4.14 to find one solution. Finding other solutions is more easy by a direct method.]

4.5. Define for two terms F, A and a natural number n the term $F^n A$ as follows.

$$\begin{aligned} F^0 A &= A; \\ F^{n+1} A &= F(F^n A). \end{aligned}$$

Now we can represent a natural number n as a lambda term c_n, its so called *Church numeral*.

$$\mathsf{c}_n \equiv \lambda fa.f^n a.$$

Define

$$\begin{aligned} A_+ &\equiv \lambda nmfa.nf(mfa); \\ A_\times &\equiv \lambda nmfa.n(mf)a; \\ A_{\exp} &\equiv \lambda nmfa.nmfa. \end{aligned}$$

Show that

$$\begin{aligned} &\text{(i)} && A_+ \mathsf{c}_n \mathsf{c}_m &&= c_{n+m}; \\ &\text{(ii)} && A_\times \mathsf{c}_n \mathsf{c}_m &&= c_{n \times m}; \\ &\text{(iii)} && A_{\exp} \mathsf{c}_n \mathsf{c}_m &&= c_{m^n}. \end{aligned}$$

Moral: one can do computations via lambda calculus.

4.6. Another well known and useful term is $\mathsf{C} \equiv \lambda xyz.xzy$. Express C in terms of I, K and S, only using application of these terms.

5. Self-reflection

We present the following fact with a proof depending on another fact.

5.1. Fact. Let x, y be to distinct variables (e.g. x, x' or x'', x'''). Then

$$x \neq_\lambda y.$$

Proof. Use Fact 4.17. If $x =_l y$, then x has two normal forms: itself and y. ■

5.2. Application. $\mathsf{K} \neq_\lambda \mathsf{I}$.

Proof. Suppose $\mathsf{K} = \mathsf{I}$. Then

$$\begin{aligned} x &= \mathsf{K}xy \\ &= \mathsf{I}\mathsf{K}xy \\ &= \mathsf{K}\mathsf{I}xy \\ &= \mathsf{I}y \\ &= y, \end{aligned}$$

a contradiction. ■

5.3. Application. There is no term P_1 such that $P_1(xy) =_\lambda x$.

Proof. If P_1 would exist, then as $\mathsf{K}xy = x = \mathsf{I}x$ one has

$$\mathsf{K}x = P_1((\mathsf{K}x)y) = P_1(\mathsf{I}x) = \mathsf{I}.$$

Therefore we obtain the contradiction

$$x = \mathsf{K}xy = \mathsf{I}y = y. \blacksquare$$

In a sense this is a pity. The agents, that lambda terms are, cannot separate two of them that are together. When we go over to codes of lambda terms the situation changes. The situation is similar for proteins that cannot always cut into parts another protein, but are able to have this effect on the code of the proteins, the DNA.

Data types

Before we enter the topic of coding of lambda terms, it is good to have a look at some datatypes.

Context-free languages can be considered as algebraic data types. Consider for example

$$\boxed{\begin{array}{lcl} S & \to & 0 \\ S & \to & S^+ \end{array}}$$

This generates the language

$$\text{Nat} = \{0, 0^+, 0^{++}, 0^{+++}, \ldots\}$$

that is a good way to represent the natural numbers

$$0, 1, 2, 3, \ldots$$

Another example generates binary trees.

$$\begin{array}{lcl} S & \to & \spadesuit \\ S & \to & \bullet SS \end{array}$$

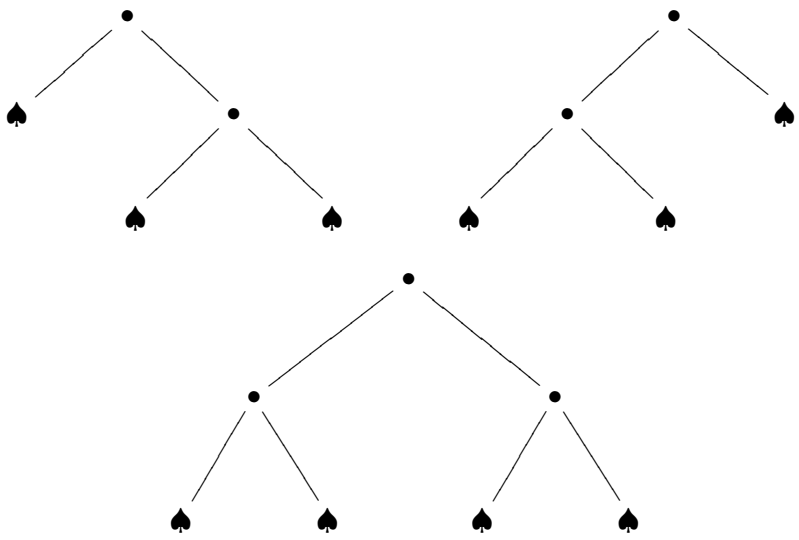
generating the language that we call *Tree*. It is not necessary to use parentheses. For example the words

$$\bullet\spadesuit\bullet\spadesuit\spadesuit$$
$$\bullet\bullet\spadesuit\spadesuit\spadesuit$$
$$\bullet\bullet\spadesuit\spadesuit\bullet\spadesuit\spadesuit$$

are in Tree. With parentrheses and commas these expressions become more readable for humans, but these auxiliary signs are not necessary:

$$\bullet(\spadesuit, \bullet(\spadesuit, \spadesuit))$$
$$\bullet(\bullet(\spadesuit, \spadesuit), \spadesuit)$$
$$\bullet(\bullet(\spadesuit, \spadesuit), \bullet(\spadesuit, \spadesuit))$$

These expressions have as 'parse trees' respectively the following:



One way to represent as lambda terms such data types is as follows.

5.4. Definition. (Böhm and Berarducci)
(i) An element of Nat, like 0^{++} will be represented first like

$$s(sz)$$

and then like

$$\lambda sz.s(sz).$$

If n is in Nat we write $\ulcorner n \urcorner$ for this lambda term. So $\ulcorner 2 \urcorner \equiv \lambda sz.s(sz)$, $\ulcorner 3 \urcorner \equiv \lambda sz.s(s(sz))$. Note that $\ulcorner n \urcorner \equiv \lambda sz.s^n z \equiv \mathsf{C}_n$.

(ii) An element of Tree, like $\bullet\spadesuit\bullet\spadesuit\spadesuit$ will be represented first by

$$bs(bss)$$

and then by

$$\lambda bs.bs(bss).$$

This lambda term is denoted by $\ulcorner\bullet\spadesuit\bullet\spadesuit\spadesuit\urcorner$, in this case. In general a tree t in Tree will be represented as lambda term $\ulcorner t\urcorner$.

Now it becomes possible to compute with trees. For example making the mirror image is performed by the term

$$F_{\text{mirror}} \equiv \lambda tbs.t(\lambda pq.bqp)s.$$

The operation of enting one tree at the endpoints of another tree is performed by

$$F_{\text{enting}} \equiv \lambda t_1 t_2 bs.t_1 b(t_2 bs).$$

The attentive reader is advised to make exercises 5.4 and 5.5.

Tuples and projections

For the efficient coding of lambda terms as lambda terms a different representation of datatypes is needed. First we find a way to connect terms together in such a way, that the components can be retrieved easily.

5.5. Definition. Let $\vec{M} \equiv M_1, \ldots, M_n$ be a sequence of lambda terms. Define

$$\langle M_1, \ldots, M_n\rangle \equiv \lambda z.zM_1, \ldots, M_n.$$

Here the variable z should not be in any of the M's. Define

$$U_i^n \equiv \lambda x_1, \ldots, x_n.x_i.$$

5.6. Proposition. *For all natural numbers i, n with $1 \leq i \leq n$, one has*

$$\langle M_1, \ldots, M_n\rangle U_i^n = M_i.$$

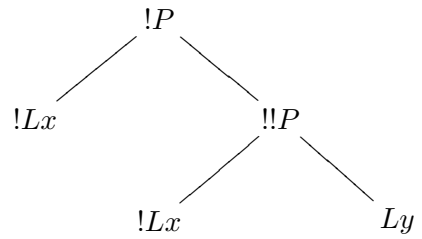
5.7. Corollary. *Define $P_i^n \equiv \lambda z.zU_i^n$. Then $P_i^n\langle M_1, \ldots, M_n\rangle = M_i$.*

Now we introduce a new kind of binary trees. At the endpoints there is not a symbol $\spadesuit$ but a variable x made into a leaf Lx. Moreover, any such tree may get ornamented with a !. Such binary trees we call labelled trees or simply ltrees.

5.8. Definition. The data type of ltrees is defined by the following context-free grammar. The start symbol is `ltree`.

`ltree`	$\rightarrow$	L `var`
`ltree`	$\rightarrow$	P `ltree ltree`
`ltree`	$\rightarrow$	! `ltree`
`var`	$\rightarrow$	x
`var`	$\rightarrow$	`var`$'$

A typical ltree is $!P!Lx!!P!LxLy$ or more readably $!P(!Lx, !!P(!Lx, Ly))$. It can be representated as a tree as follows.



We say that for ltree there are three *constructors*. A binary constructor P that puts two trees together, and two unary constructors L and !. L makes from a variable an ltree and ! makes from an ltree another one.

Now we are going to represent these expressions as lambda terms.

5.9. Definition. (Böhm, Piperno and Guerrini)
(i) We define three lambda terms $F_L, F_P, F_!$ to be used for the representation of ltree.

$$\begin{aligned} F_L &\equiv \lambda xe.eU_1^3xe; \\ F_P &\equiv \lambda xye.eU_2^3xye; \\ F_! &\equiv \lambda xe.eU_3^3xe. \end{aligned}$$

These definitions are a bit more easy to understand if written according to their intended use (do exercise 5.7).

$$\begin{aligned} F_Lx &= \lambda e.eU_1^3xe; \\ F_Pxy &= \lambda e.eU_2^3xye; \\ F_!x &= \lambda e.eU_3^3xe. \end{aligned}$$

(ii) For an element t of ltree we define the representing lambda term $\ulcorner t \urcorner$.

$$\begin{aligned} \ulcorner Lx \urcorner &= F_Lx; \\ \ulcorner Pt_1t_2 \urcorner &= F_P\ulcorner t_1 \urcorner\ulcorner t_2 \urcorner; \\ \ulcorner !t \urcorner &= F_!\ulcorner t \urcorner. \end{aligned}$$

Actually this is just a mnemonic. We want that the representations are normal forms, do not compute any longer.

$$\begin{aligned} \ulcorner Lx \urcorner &\equiv \lambda e.eU_1^3xe; \\ \ulcorner Pt_1t_2 \urcorner &\equiv \lambda e.eU_2^3\ulcorner t_1 \urcorner\ulcorner t_2 \urcorner e; \\ \ulcorner !t \urcorner &\equiv \lambda e.eU_3^3\ulcorner t \urcorner e. \end{aligned}$$

The representation of the data was chosen in such a way that computable function on them can be easily represented. The following result states that there exist functions on the represented labelled trees such that their action on a composed tree depend on the components and that function in a given way.

5.10. PROPOSITION. *Let A_1, A_2, A_3 be given lambda terms. Then there exists a lambda term H such that*[11].

$$\begin{aligned} H(F_L x) &= A_1 x H \\ H(F_P x y) &= A_2 x y H \\ H(F_! x) &= A_3 x H \end{aligned}$$

PROOF. We try $H \equiv \langle\langle B_1, B_2, B_3\rangle\rangle$ where the $\vec{B}$ are to be determined.

$$\begin{aligned} H(F_L x) &= \langle\langle B_1, B_2, B_3\rangle\rangle(F_L x) \\ &= F_L x\langle B_1, B_2, B_3\rangle \\ &= \langle B_1, B_2, B_3\rangle U_1^3 x\langle B_1, B_2, B_3\rangle \\ &= U_1^3 B_1, B_2, B_3 x\langle B_1, B_2, B_3\rangle \\ &= B_1 x\langle B_1, B_2, B_3\rangle \\ &= A_1 x H, \end{aligned}$$

provided that $B_1 \equiv \lambda x b.A_1 x\langle b\rangle$. Similarly

$$\begin{aligned} H(F_P x y) &= B_2 x y\langle B_1, B_2, B_3\rangle \\ &= A_2 x y H, \end{aligned}$$

provided that $B_2 \equiv \lambda x y b.A_2 x y\langle b\rangle$. Finally,

$$\begin{aligned} H(F_! x) &= B_3 x\langle B_1, B_2, B_3\rangle \\ &= A_3 x H, \end{aligned}$$

provided that $B_3 \equiv \lambda x b.A_3 x\langle b\rangle$. ■

Stil we have as goal to represent lambda terms as lambda terms in nf, such that decoding is possible by a fixed lambda term. Moreover, finding the code of the components of a term M should be possible from the code of M, again using a lambda term. To this end the (represented) constructors of ltree, $F_L, F_P, F_!$, will be used.

5.11. DEFINITION. (Mogensen) Define for a lambda term M its code $\ulcorner M\urcorner$ as follows.

$$\begin{aligned} \ulcorner x\urcorner &\equiv \lambda e.eU_1^3 xe &&= F_L x; \\ \ulcorner MN\urcorner &\equiv \lambda e.eU_2^3 \ulcorner M\urcorner\ulcorner N\urcorner e &&= F_P \ulcorner M\urcorner\ulcorner N\urcorner; \\ \ulcorner \lambda x.M\urcorner &\equiv \lambda e.eU_3^3(\lambda x.\ulcorner M\urcorner)e &&= F_! (\lambda x.\ulcorner M\urcorner). \end{aligned}$$

[11] A weaker requirement is the following, where H of a composed ltree depends on H of the components in a given way:

$$\begin{aligned} H(F_L x) &= A_1 x(Hx) \\ H(F_P x y) &= A_2 x y(Hx)(Hy) \\ H(F_! x) &= A_3 x(Hx) \end{aligned}$$

This is called primitive recursion, whereas the proposition provides (general) recursion.

The trick here is to code the lambda with lambda itself, one may speak of an inner model of the lambda calculus in itself. Putting the ideas of Mogensen [1992] and Böhm et al. [1994] together, as done by Berarducci and Böhm [1993], one obtains a very smooth way to create the mechanism of reflection the lambda calculus. The result was already proved in Kleene [1936][12].

5.12. THEOREM. *There is a lambda term* E *(evaluator or self-interpreter) such that*

$$\begin{aligned} \mathsf{E}\ulcorner x\urcorner &= x; \\ \mathsf{E}\ulcorner MN\urcorner &= \mathsf{E}\ulcorner M\urcorner(\mathsf{E}\ulcorner N\urcorner); \\ \mathsf{E}\ulcorner \lambda x.M\urcorner &= \lambda x.(\mathsf{E}\ulcorner M\urcorner). \end{aligned}$$

It follows that for all lambda terms M *one has*

$$\mathsf{E}\ulcorner M\urcorner = M.$$

PROOF. By Proposition 5.10 for arbitary $A_1, \ldots, A_3$ there exists an E such that

$$\begin{aligned} \mathsf{E}(F_L x) &= A_1 x \mathsf{E}; \\ \mathsf{E}(F_P mn) &= A_2 mn\mathsf{E}; \\ \mathsf{E}(F_! p) &= A_3 p\mathsf{E}. \end{aligned}$$

If we take $A_1 \equiv \mathsf{K}$, $A_2 \equiv \lambda abc.ca(cb)$ and $A_3 \equiv \lambda abc.b(ac)$, then this becomes

$$\begin{aligned} \mathsf{E}(F_L x) &= x; \\ \mathsf{E}(F_P mn) &= \mathsf{E}m(\mathsf{E}n); \\ \mathsf{E}(F_! p) &= \lambda x.(\mathsf{E}(px)). \end{aligned}$$

But then (do exercise 5.9)

$$\begin{aligned} \mathsf{E}(\ulcorner x\urcorner) &= x; \\ \mathsf{E}(\ulcorner MN\urcorner) &= \mathsf{E}\ulcorner M\urcorner(\mathsf{E}\ulcorner N\urcorner); \\ \mathsf{E}(\ulcorner \lambda x.M\urcorner) &= \lambda x.(\mathsf{E}(\ulcorner M\urcorner)). \end{aligned}$$

That E is a self-interpreter, i.e. $\mathsf{E}\ulcorner M\urcorner = M$, now follows by induction on M. ■

5.13. COROLLARY. *The term* $\langle\langle \mathsf{K}, \mathsf{S}, \mathsf{C}\rangle\rangle$ *is a self-interpreter for the lambda calculus with the coding defined in Definition 5.11.*

PROOF. $\mathsf{E} \equiv \langle\langle B_1, B_2, B_3\rangle\rangle$ with the $\vec{B}$ coming from the $A_1 \equiv \mathsf{K}$, $A_2 \equiv \lambda abc.ca(cb)$ and $A_3 \equiv \lambda abc.b(ac)$. Looking at the proof of 5.10 one sees

$$\begin{aligned} B_1 &= \lambda xz.A_1 x\langle z\rangle \\ &= \lambda xz.x \\ &= \mathsf{K}; \end{aligned}$$

[12] But only valid for lambda terms M without free variables.

$$
\begin{array}{rcll}
B_2 & = & \lambda xyz.A_2xy\langle z\rangle & \\
 & = & \lambda xyz.\langle z\rangle x(\langle z\rangle y) & \\
 & = & \lambda xyz.xz(yz) & \\
 & = & \mathsf{S}; & \\
B_3 & = & \lambda xz.A_3x\langle z\rangle & \\
 & = & \lambda xz.(\lambda abc.b(ac))x\langle z\rangle & \\
 & = & \lambda xz.(\lambda c.xcz) & \\
 & \equiv & \lambda xzc.xcz & \\
 & \equiv & \lambda xyz.xzy & \\
 & \equiv & \mathsf{C}, & \text{see exercise 4.6.}
\end{array}
$$

Hence $\mathsf{E} = \langle\langle \mathsf{K}, \mathsf{S}, \mathsf{C}\rangle\rangle$. ■

This term

$$\mathsf{E} = \langle\langle \mathsf{K}, \mathsf{S}, \mathsf{C}\rangle\rangle$$

is truly a tribute to

Kleene, Stephen Cole
(1909-1994)

(using the family-name-first convention familiar from scholastic institutions) who invented in 1936 the first self-interpreter for the lambda calculus[13].

The idea of a language that can talk about itself has been heavily used with higher programming languages. The way to translate ('compile') these into machine languages is optimized by writing the compiler in the language itself (and run it the first time by an older *ad hoc* compiler). This possibility of efficiently executed higher programming languages was first put into doubt, but was realized by mentioned reflection since the early 1950-s and other optimalizations. The box of Pandora of the world of IT was opened.

The fact that such extremely simple (compared to a protein like titin with slightly less than 27000 aminoacids) self interpreter is possible gives hope to understand the full mechanism of cell biology and evolution. In Buss and Fontana [1994] evolution is modelled using lambda terms.

Exercises

5.1. Show that

$$
\begin{array}{rcl}
\mathsf{K} & \neq_\lambda & \mathsf{S}; \\
\mathsf{I} & \neq_\lambda & \mathsf{S}.
\end{array}
$$

.

[13] Kleene's construction was much more involved. In order to deal with the 'binding effect' of λx lambda terms where first translated into **CL**before the final code was obtained. This causes some technicalities that make the original E more complex.

5.2. Show that there is no term P_2 such that $P_2(xy) =_\lambda y$.

5.3. Construct all elements of Tree with exactly four ♠s in them.

5.4. Show that

$$\begin{aligned} F_{\text{mirror}}{}^{\ulcorner}\bullet\spadesuit\bullet\spadesuit\spadesuit{}^{\urcorner} &= {}^{\ulcorner}\bullet\bullet\spadesuit\spadesuit\spadesuit{}^{\urcorner}; \\ F_{\text{mirror}}{}^{\ulcorner}\bullet\bullet\spadesuit\spadesuit\spadesuit{}^{\urcorner} &= {}^{\ulcorner}\bullet\spadesuit\bullet\spadesuit\spadesuit{}^{\urcorner}; \\ F_{\text{mirror}}{}^{\ulcorner}\bullet\bullet\spadesuit\spadesuit\bullet\spadesuit\spadesuit{}^{\urcorner} &= {}^{\ulcorner}\bullet\bullet\spadesuit\spadesuit\bullet\spadesuit\spadesuit{}^{\urcorner}. \end{aligned}$$

5.5. Compute $F_{\text{enting}}{}^{\ulcorner}\bullet\spadesuit\bullet\spadesuit\spadesuit{}^{\urcorner}{}^{\ulcorner}\bullet\bullet\spadesuit\spadesuit\spadesuit{}^{\urcorner}$.

5.6. Define terms P_i^n such that for $1 \leq i \leq n$ one has

$$P_i^n\langle M_1, \ldots, M_n\rangle = M_i.$$

5.7. Show that the second set of three equations in definition 5.9 follows from the first set.

5.8. Show that given terms A_1, A_2, A_3 there exists a term H such that the scheme of primitive recurion, see footnote 5.10 is valid.

5.9. Show the last three equations in the proof of Theorem 5.12.

5.10. Construct lambda terms P_1 and P_2 such that for all terms M, N

$$P_1{}^{\ulcorner}MN{}^{\urcorner} = M \;\&\; P_2{}^{\ulcorner}MN{}^{\urcorner} = N.$$

References

Alberts, B. et al. [1997]. *The Cell*, Garland.

Barendregt, H. P. [1984]. *The Lambda Calculus, its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics 103, revised edition, North-Holland Publishing Co., Amsterdam.

Barendregt, H. P. [1997]. The impact of the lambda calculus in logic and computer science, *Bull. Symbolic Logic* **3**(2), pp. 181–215.

Berarducci, Alessandro and Corrado Böhm [1993]. A self-interpreter of lambda calculus having a normal form, *Computer science logic (San Miniato, 1992)*, Lecture Notes in Comput. Sci. 702, Springer, Berlin, pp. 85–99.

Blackmore, S. [2004]. *Consciousness, an Introduction*, Oxford University Press, Oxford.

Böhm, Corrado, Adolfo Piperno and Stefano Guerrini [1994]. λ-definition of function(al)s by normal forms, *Programming languages and systems—ESOP '94 (Edinburgh, 1994)*, Lecture Notes in Comput. Sci. 788, Springer, Berlin, pp. 135–149.

Buss, L.W. and W. Fontana [1994]. 'the arrival of the fittest': Toward a theory of biological organization, *Bulletin of Mathematical Biology* **56**(1), pp. 1–64.

Chalmers, D. [1996]. *The Conscious Mind, Towards a Fundamental Theory*, Oxford University Press, Oxford.

Chomsky, N. [1956]. Three models of the description of language, *IRE Transactions on Information Theory* **2**(3), pp. 113–124.

Church, A. [1932]. A set of postulates for the foundation of logic, *Annals of Mathematics, second series* **33**, pp. 346–366.

Church, A. [1936]. An unsolvable problem of elementary number theory, *American Journal of Mathematics* **58**, pp. 345–363.

Curry, H. B. [1930]. Grundlagen der kombinatorischen Logic,, *American Journal of Mathematics* **52**, pp. 509–536, 789–834.

Dennet, D. [1993]. *Consciousness Explained*, Penguin Books.

Goldstein, J. [1983]. *The Experience of Insight*, Shambhala.

Hofstadter, D. [1979]. *Gödel Escher Bach, An Eternal Golden Braid*, Harvester Press.

Howe, D. [1992]. Reflecting the semantics of reflected proof, *Proof Theory, ed. P. Aczel*, Cambridge University Press, pp. 229–250.

Kleene, S. C. [1936]. Lambda-definability and recursiveness, *Duke Mathematical Journal* **2**, pp. 340–353.

Kozen, Dexter C. [1997]. *Automata and computability*, Undergraduate Texts in Computer Science, Springer-Verlag, New York.

Menninger, K., M. Mayman and P. Pruyser [1963]. *The Vital Balance. The Life Process in Mental Health and Illness*, Viking.

Mogensen, Torben Æ. [1992]. Efficient self-interpretation in lambda calculus, *J. Funct. Programming* **2**(3), pp. 345–363.

Peitsch, M.C., D.R. Stampf, T.N.C. Wells and J.L. Sussman [1995]. The swiss-3dimage collection and pdb-browser on the world-wide web, *Trends in Biochemical Sciences* **20**, pp. 82–84. URL: `<www.expasy.org>`.

Schönfinkel, M. [1924]. Über die Bausteine der mathematischen Logik, *Mathematische Annalen* **92**, pp. 305–316.

Smullyan, R. [1992]. *Gödel's Incompleteness Theorems*, Oxford University Press.

Stapp, H. [1996]. The hard problem: A quantum approach, *Journal of Consciousness Studies* **3**(3), pp. 194–210.

Tarski, A. [1933/1995]. *Introduction to Logic*, Dover.

Turing, A.M. [1936]. On Computable Numbers, with an Application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society, Series 2* **42**, pp. 230–265.

Yates, M. [1998]. What computers can't do, *Plus*. ⟨`plus.maths.org/issue5/index.html`⟩.