# A Note on Recursive Definitions

Antonín Kučera

IA011, Semantics of Programming Languages (FI MU Brno)

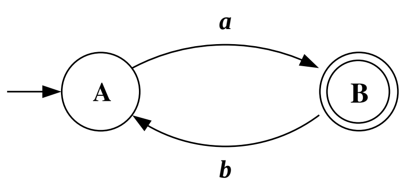
**Important:** The aim of these notes is to explain how domains, continuous functions, and the fixed-point theorem are used to define a precise meaning for recursively defined objects (i.e., objects which are 'defined in terms of themselves'). In fact, it is a crucial step towards understanding the denotational semantics of functional programs, and therefore these notes should be read right after the first part of the course, i.e., after understanding the denotational semantics of a **while**-loop.

The chosen level of presentation is semi-complete in the sense that we state definitions, formulate our claims, but just 'observe' why they hold (there are no formal proofs included in this text, though some of them are actually simple). In this case, the emphasis is on *building the right intuition*, not on constucting the proofs.

## Regular languages as fixed-points

At first glance, regular languages have nothing to do with formal semantics of programming languages. However, they allow to demonstrate the above discussed principle in a very simple way and on familiar objects.

**Example 1:** Consider the following finite-state automaton $\mathcal{A}$:




It can be equivalently represented by a regular grammar

$$\begin{aligned} A &\rightarrow aB \\ B &\rightarrow bA \mid \varepsilon \end{aligned}$$

where $A, B$ are non-terminal symbols. If we denote by $L_A$ and $L_B$ the languages

$$\begin{aligned} L_A &= \{w \in \{a,b\}^* \mid A \Rightarrow^* w\} \\ L_B &= \{w \in \{a,b\}^* \mid B \Rightarrow^* w\} \end{aligned}$$

(i.e., $L_A = \{a, aba, ababa, abababa, \ldots\}$, $L_B = \{\varepsilon, ba, baba, bababa, \ldots\}$), we can observe that the above given grammar in fact specifies a certain relationship between the languages $L_A$ and $L_B$, which is more explicitly expressed by equations

$$L_A = \{a\} \cdot L_B \tag{1}$$

$$L_B = \{b\} \cdot L_A \quad \cup \quad \{\varepsilon\} \tag{2}$$

Here, $L_1 \cdot L_2$ (where $L_1, L_2$ are languages) denotes the 'concatenation' of $L_1$ and $L_2$ defined by

$$L_1 \cdot L_2 = \{v_1 v_2 \mid v_1 \in L_1, v_2 \in L_2\}$$

If we are only interested in the language $L_A$ (which is the language accepted by the automaton $\mathcal{A}$), we can further simplify the equations by 'substituting' the occurence of $L_B$ in (1) by its corresponding right-hand side taken from (2). Thus we obtain

$$L_A = \{a\} \cdot (\ \{b\} \cdot L_A \ \cup \ \{\varepsilon\}\ ) \tag{3}$$

which can by further rewritten to

$$L_A = \{a\} \cdot \{b\} \cdot L_A \quad \cup \quad \{a\} \tag{4}$$

(here we use the distributivity of '$\cdot$' and the fact that $\{a\} \cdot \{\varepsilon\} = \{a\}$). The equation (4) cannot be directly seen as a definition of $L_A$ because it 'defines' $L_A$ in terms of itself. However, from the right-hand side of (4) we can (easily!) extract a kind of 'recipe' which allows to construct an infinite chain of finite approximations of $L_A$ such that the language $L_A$ is the 'limit' of this chain. Formally, we first define the domain

$$\mathcal{L} = (\mathcal{P}(\{a,b\}^*), \subseteq)$$

of all languages over the alphabet $\{a, b\}$. Note that $\emptyset$ is the bottom of $\mathcal{L}$. Now we define a function $\Gamma : \mathcal{L} \to \mathcal{L}$ as follows:

$$\Gamma(L) = \{a\} \cdot \{b\} \cdot L \quad \cup \quad \{a\} \tag{5}$$

The definition of $\Gamma$ 'mimics' the right-hand side of (4). Now one can prove that

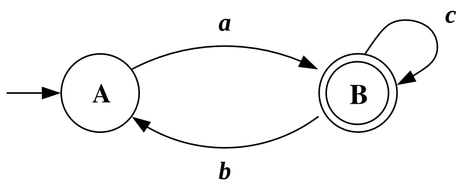
- $\Gamma$ is continuous, and
- $L_A = \mathbf{fix}\, \Gamma$.

Our aim here is not to provide a formal proof of these claims (though it is in fact easy – try out!), but to realize (at an intuitive level) *why* $L_A = \mathbf{fix}\,\Gamma$, i.e., 'how it works'. The key is to (once again) 'unfold' the definition of $\mathbf{fix}\,\Gamma$:

$$\begin{array}{lclll}
\mathbf{fix}\,\Gamma & = & \bigcup_{n\in\omega}\Gamma^n(\emptyset) & = & \Gamma^0(\emptyset) = \emptyset \\
 & & & \cup & \Gamma^1(\emptyset) = \{a\}\cdot\{b\}\cdot\emptyset \quad \cup \quad \{a\} \quad = \quad \{a\} \\
 & & & \cup & \Gamma^2(\emptyset) = \{a\}\cdot\{b\}\cdot\{a\} \quad \cup \quad \{a\} \quad = \quad \{aba, a\} \\
 & & & \cup & \Gamma^3(\emptyset) = \{a\}\cdot\{b\}\cdot\{aba, a\} \quad \cup \quad \{a\} \quad = \quad \{ababa, aba, a\} \\
 & & & \vdots & 
\end{array}$$

Hence, $\Gamma^n(\emptyset)$ consists of all words of the form $a(ba)^i$, where $0 \leq i < n$ (it could be formally proved by induction on $n$). It means that *each* word $w \in L_A$ eventually appers in some $\Gamma^k(\emptyset)$; conversely, we see (and could prove by induction on j) that $\Gamma^j(\emptyset) \subseteq L_A$ for every $j \in \omega$. From this we get that $L_A = \mathbf{fix}\,\Gamma$.

**Example 2:** Now consider the finite-state automaton $\mathcal{B}$:




It can be equivalently represented by a regular grammar

$$\begin{array}{lcl}
A & \rightarrow & aB \\
B & \rightarrow & bA \mid cB \mid \varepsilon
\end{array}$$

which can be further rewritten to equations

$$L_A = \{a\}\cdot L_B \tag{6}$$
$$L_B = \{b\}\cdot L_A \quad \cup \quad \{c\}\cdot L_B \quad \cup \quad \{\varepsilon\} \tag{7}$$

We are again interested in the language $L_A$ (which is the language accepted by $\mathcal{B}$). As opposed to Example 1, now we cannot get rid of the equation for $L_B$ by substitution—indeed, if we substitute the occurence of $L_B$ within (6) by the right-hand side of (7), we introduce another 'copy' of $L_B$. The solution is to approximate both $L_A$ and $L_B$ at the same time, and compute a better approximation for both languages by applying the right-hand sides of (6) and (7) to the current approximations. Formally, define a domain

$$\mathcal{L} = (\mathcal{P}(\{a, b, c\}^*), \subseteq)$$

and another domain

$$\mathcal{L} \times \mathcal{L} \;=\; (\mathcal{P}(\{a,b,c\}^*) \times \mathcal{P}(\{a,b,c\}^*), \sqsubseteq)$$

where $(L_1, L_2) \sqsubseteq (L_1', L_2')$ iff $L_1 \subseteq L_1'$ and $L_2 \subseteq L_2'$. Hence, $\mathcal{L} \times \mathcal{L}$ is a domain of all *pairs* of languages over the alphabet $\{a,b,c\}$ with the componentwise ordering. Note that $(\emptyset, \emptyset)$ is the bottom of $\mathcal{L} \times \mathcal{L}$. Now we define a function $\Gamma : (\mathcal{L} \times \mathcal{L}) \to (\mathcal{L} \times \mathcal{L})$ as follows:

$$\Gamma((L_1, L_2)) \;=\; (\,\{a\} \cdot L_2, \quad \{b\} \cdot L_1 \cup \{c\} \cdot L_2 \cup \{\varepsilon\}\,) \tag{8}$$

Observe that the definition of $\Gamma$ again mimics the right-hand sides of (6) and (7); and again, we can prove that

- $\Gamma$ is continuous, and
- $(L_A, L_B) \;=\; \mathbf{fix}\,\Gamma$.

To see why is that, examine the 'unfolded' definition of **fix** $\Gamma$ which looks as follows:
$\mathbf{fix}\,\Gamma \;=\; \bigcup_{n \in \omega} \Gamma^n(\emptyset) =$

$$\begin{array}{lll}
 & \Gamma^0((\emptyset,\emptyset)) = (\emptyset,\emptyset) & \\
\cup & \Gamma^1((\emptyset,\emptyset)) = (\,\{a\}\cdot\emptyset, \quad \{b\}\cdot\emptyset \cup \{c\}\cdot\emptyset \cup \{\varepsilon\}\,) & = \; (\emptyset, \{\varepsilon\}) \\
\cup & \Gamma^2((\emptyset,\emptyset)) = (\,\{a\}\cdot\{\varepsilon\}, \quad \{b\}\cdot\emptyset \cup \{c\}\cdot\{\varepsilon\} \cup \{\varepsilon\}\,) & = \; (\{a\}, \{c,\varepsilon\}) \\
\cup & \Gamma^3((\emptyset,\emptyset)) = (\,\{a\}\cdot\{c,\varepsilon\}, \quad \{b\}\cdot\{a\} \cup \{c\}\cdot\{c,\varepsilon\} \cup \{\varepsilon\}\,) & = \; (\{ac, a\}, \{ba, cc, c, \varepsilon\}) \\
\vdots & & 
\end{array}$$

## Recursively defined functions

An important outcome of the second part of the course is a formal definition of the denotational semantics of functional programs. In fact, functional programs are (at a certain level of abstraction) just systems of recursively defined functions (i.e., each function is defined in terms of other functions in the system). We demonstrate the idea on two simple examples.

**Example 3:** Let $f : \mathbb{Z} \to \mathbb{Z}$ be a function 'defined' by

$$f(x) = \begin{cases} x * f(x-1) & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases} \tag{9}$$

The problem with this 'definition' is again that the $f$ is defined in terms of itself. However, we *do* have some intuitive 'operational' understanding of $f$ in the sense that we can 'compute' the value of $f$ for a given argument $n$ by applying the right-hand side of (9) repeatedly. Thus we get, for example,

$$f(4) = 4 * f(3) = 4 * 3 * f(2) = 4 * 3 * 2 * f(1) = 4 * 3 * 2 * 1 * f(0) = 4 * 3 * 2 * 1 * 1 = 24$$

Hence, we intuitively expect that the above equation actually denotes the function $F : \mathbb{Z} \to \mathbb{Z}$ defined by

$$F(x) = \begin{cases} x! & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases}$$

How can we formally establish the connection between the (recursive) equation (9) and its expected denotation $F$? Actually, in a very similar way as in Example 1 – we 'extract' a function $\Gamma : (\mathbb{Z} \rightharpoonup \mathbb{Z}) \to (\mathbb{Z} \rightharpoonup \mathbb{Z})$ from the right-hand side of the equation for $f$ as follows:

$$\Gamma(g)x = \begin{cases} x * g(x-1) & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases}$$

We can prove that

- $\Gamma$ is continuous,
- $F \ = \ \textbf{fix}\, \Gamma$.

Let us have a closer look at the approximations:

$$\begin{aligned}
\Gamma^0(\emptyset) &= \emptyset \\
\Gamma^1(\emptyset)x &= \begin{cases} x * \Gamma^0(x-1) & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases} \\
&= \begin{cases} \text{undefined} & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases} \\
\Gamma^2(\emptyset)x &= \begin{cases} x * \Gamma^1(x-1) & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases} \\
&= \begin{cases} \text{undefined} & \text{if } x \geq 2 \\ 1 & \text{if } x = 1 \\ 1 & \text{if } x < 1 \end{cases} \\
&= \begin{cases} \text{undefined} & \text{if } x \geq 2 \\ 1 & \text{if } x \leq 1 \end{cases}
\end{aligned}$$

In general, we have that

$$\Gamma^k(\emptyset)x \quad = \quad \begin{cases} \text{undefined} & \text{if } x \geq k \\ x! & \text{if } 1 \leq x < k \\ 1 & \text{if } x < 1 \end{cases}$$

which could be proved by induction on $k$. Hence, we see that indeed $F = \bigcup_{n\in\omega} \Gamma^n(\emptyset)$.

**Example 4:** Let $g : (\mathbb{Z} \times \mathbb{Z}) \to \mathbb{Z}$ and $f : \mathbb{Z} \to \mathbb{Z}$ be functions 'defined' as follows:

$$g(x, y) \quad = \quad \begin{cases} f(x + y) & \text{if } x \leq 0 \text{ or } y \leq 0 \\ g(x - 1, y - 1) + 3 & \text{if } x > 0 \text{ and } y > 0 \end{cases} \tag{10}$$

$$f(x) \quad = \quad \begin{cases} x * f(x - 1) & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases} \tag{11}$$

We see that the function $g$ is 'defined' in terms of $f$ and itself (the function $f$ is the same function as in Example 3). Again, we have some intuitive understanding of $f$ and $g$ which allows to evaluate those functions on given arguments. For example,

$$g(2, 6) = g(1, 5) + 3 = g(0, 4) + 3 + 3 = f(4) + 3 + 3 = 24 + 3 + 3 = 30$$

Hence, we expect that (11) specifies the function $F$ of Example 3, and a closer analysis of (10) reveals that (10) actually specifies the function $G$ defined as follows:

$$G(x, y) \quad = \quad \begin{cases} (x + y)! & \text{if } (x \leq 0 \text{ or } y \leq 0) \text{ and } (x{+}y) \geq 1 \\ 1 & \text{if } (x \leq 0 \text{ or } y \leq 0) \text{ and } x < 1 \\ 3 * \min\{x, y\} \,+\, |x - y|! & \text{if } x > 0 \text{ and } y > 0 \end{cases}$$

Again, this observation is based just on our intuitive 'operational' understanding of $f$ and $g$. However, we can formalize the intuition in a similar way as in Example 3. The (only) difference is that now we have to assign the meaning to *two* functions at the same time. Actually, it is the same kind of distinction as between Example 1 and Example 2, so we *know* how to do it: Define a function

$$\Gamma : (((\mathbb{Z} \times \mathbb{Z}) \rightharpoonup \mathbb{Z}) \times (\mathbb{Z} \rightharpoonup \mathbb{Z})) \to (((\mathbb{Z} \times \mathbb{Z}) \rightharpoonup \mathbb{Z}) \times (\mathbb{Z} \rightharpoonup \mathbb{Z}))$$

which for a given *pair* of functions $(h, l)$ returns another pair of functions $(h', l')$, i.e., $\Gamma(\ (h, l)\ ) = (h', l')$ where

$$h'(x, y) \quad = \quad \begin{cases} h(x + y) & \text{if } x \leq 0 \text{ or } y \leq 0 \\ l(x - 1, y - 1) + 3 & \text{if } x > 0 \text{ and } y > 0 \end{cases} \tag{12}$$

$$l'(x) \quad = \quad \begin{cases} x * l(x - 1) & \text{if } x \geq 1 \\ 1 & \text{if } x < 1 \end{cases} \tag{13}$$

The definition of $\mathtt{h}'$ matches the right-hand side of (10), and the definition of $\mathtt{l}'$ matches the right-hand side of (11). Now we could prove that

- $\Gamma$ is continuous, and
- $(\mathsf{G}, \mathsf{F}) = \mathbf{fix}\,\Gamma$.

General systems of recursively defined functions (which will be called *declarations*) can contain an abritrary finite number of equations. However, the meaning (denotation) of such a declaration is defined in the very same way as in Example 4 – instead of a *pair* of functions, we refine the whole *tuple* of functions by a suitable $\Gamma$ whose definition is based on the right-hand sides of all equations in the given declaration. Thus, $\Gamma$ can become a very complex function but the intuition behind its functionality is still the same as in Example 4. However, it becomes increasingly more difficult to prove that $\Gamma$ is indeed continuous. This problem is elegantly solved by developing a piece of *domain theory* – we observe that domains as well as continuous functions can be constructed in a systematic and structural way. In the end we shall design a sort of 'language' of continuous function, and we prove that every function expressible in that language is continuous. The proof is basically done by induction on the structure of continuous functions in the language, and therefore it is (relatively) easily manageable.