On Properties of Watson-Crick D0L Systems

Petr Sosík*

Institute of Computer Science, Silesian University, Bezručovo náměstí 13, 746 01 Opava, Czech Republic, E-mail:Petr.Sosik@fpf.slu.cz

Technical report No. 1999/1

Abstract

Watson-Crick D0L systems, introduced in 1998 by Arto Salomaa, arise from two major principles: the Lindenmayer rewriting and the Watson-Crick complementarity principle. Complementarity can be viewed as a purely language-theoretic operation. Majority of a certain type of symbols in a string (purines vs. pyrimidines) triggers a transition to the complementary string. This paper deals with an expressive power of deterministic interactionless Watson-Crick Lindenmayer systems. A surprising result is obtained: these systems, consisting of iterated morphism and a basic DNA operation, are alone able to express any Turing computable function.

1 Watson-Crick D0L systems

For elements of formal language theory we refer to [6, 7]. Here we only briefly fix some notation. Let $(\Sigma, \cdot)$ be a free monoid with the catenation operation and let the empty word be denoted λ. We denote $|w|_a$ the number of occurrences of a symbol a in a string w for $a \in \Sigma$, $w \in \Sigma^*$. Further we denote $|w|_\Gamma = \sum_{a\in\Gamma} |w|_a$ for an arbitrary $\Gamma \subseteq \Sigma$. For $w \in \Sigma^*$ we denote w^n the catenation of n copies of w for $n \geq 1$. An arbitrary permutation of a string w is denoted by $\phi(w)$.

A concept of Watson-Crick alphabet is a straightforward generalization of notions of "natural" DNA alphabet consisting of symbols $\{A, C, G, T\}$.

A *DNA-like alphabet* Σ is an alphabet with even cardinality $2n$, $n \geq 1$, where the letters are enumerated as follows:

$$\Sigma = \{a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n}\}.$$

We say that a_i and $\overline{a_i}$ are *complementary.* The letter to letter endomorphism h_W of Σ^* mapping each letter to the complementary letter is called the *Watson-Crick morphism.* Hence

$$h_W(a_i) = \overline{a_i},\ h_w(\overline{a_i}) = a_i, \quad 1 \leq i \leq n.$$

*Research supported by the Grant Agency of Czech Republic, grants No. 201/98/P047 and 201/99/1086.

In analogy with the DNA alphabet we call the non-barred letters *purines* and the barred letters *pyrimidines.* The subset of Σ^* consisting of all words, where the number of occurrences of pyrimidines is strictly greater than that of purines is denoted by PYR. The complement of PYR is denoted by PUR. Clearly, both PYR and PUR are context-free nonregular languages. We further denote the sets $\Sigma^{\mathrm{PUR}} = \{a_1, \ldots, a_n\}$ and $\Sigma^{\mathrm{PYR}} = \{\overline{a_1}, \ldots, \overline{a_n}\}$.

Definition 1.1 *A* Watson-Crick D0L scheme *is a construct* $G = (\Sigma, p)$, *where* $\Sigma = \{a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n}\}$, $p : \Sigma^* \longrightarrow \Sigma^*$ *is a morphism. Given a word* $w_0 \in \Sigma^*$, *the derivation sequence* $S(G, w_0)$ *defined by* G *consists of the words* $w_0, w_1, w_2, \ldots$, *where for* $i \geq 0$,

$$w_{i+1} = \begin{cases} p(w_i), & \text{if } p(w_i) \in PUR \\ h_W(p(w_i)), & \text{if } p(w_i) \in PYR. \end{cases}$$

The transition $w_i \Longrightarrow_G w_{i+1}$ is also called the derivation step of G. If $w_{i+1} = h_W(p(w_i))$, then we speak about complementation derivation step. We denote $\Longrightarrow_G^*$ the transitive and reflexive closure of $\Longrightarrow_G$ as usual.

For more details and motivation underlying the concept of Watson-Crick D0L systems see [4] (where a Watson-Crick D0L system differs from a Watson-Crick D0L scheme only by adding an axiom $w_0 \in PUR$).

2 Partial recursive functions

In this section we briefly summarize two equivalent characterizations of partial recursive functions used later. In what follows the notation $\mathcal{N}$ is used for the set of integers greater or equal to zero.

Definition 2.1 *A function* $f : \mathcal{N}^t \longrightarrow \mathcal{N}^s$ *is called* (partially) recursive *if there is a Turing machine* M *such that* $f(x_1, \ldots, x_t) = (y_1, \ldots, y_s)$, *if and only if*

$$M(1^{x_1+1}01^{x_2+1}0\ldots01^{x_t+1}) = 1^{y_1+1}0\ldots01^{y_s+1}.$$

In the previous definition the word "partially" will be omitted, if the domain of f is $\mathcal{N}^t$.

Definition 2.2 *The family of primitive recursive functions is the smallest family of integer-to-integer functions with the following properties:*

(i) It contains the following base functions:

0	*(nullary constant),*
$S(x) = x + 1$	*(successor function),*
$U_i^n(x_1, \ldots, x_n) = x_i$	*(projection functions), for* $1 \leq i \leq n$.

(ii) It is closed under the following operations:

- composition: *if* $h : \mathcal{N}^m \longrightarrow \mathcal{N}$, $g_1 : \mathcal{N}^n \longrightarrow \mathcal{N}$, ..., $g_m : \mathcal{N}^n \longrightarrow \mathcal{N}$ *are primitive recursive functions, then so is the function* $f : \mathcal{N}^n \longrightarrow \mathcal{N}$ *defined as follows:*

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

In examples we will also write $f(x_1, \ldots, x_n) = C(h, g_1, \ldots, g_m)(x_1, \ldots, x_n)$.

- primitive recursion: *if* $h : \mathcal{N}^n \longrightarrow \mathcal{N}$, $g : \mathcal{N}^{n+2} \longrightarrow \mathcal{N}$ *are primitive recursive functions, then so is the function* $f : \mathcal{N}^{n+1} \longrightarrow \mathcal{N}$ *defined as follows:*

$$\begin{aligned} f(0, x_1, \ldots, x_n) &= h(x_1, \ldots, x_n), \\ f(z+1, x_1, \ldots, x_n) &= g(z, f(z, x_1, \ldots, x_n), x_1, \ldots, x_n), \quad z \geq 0. \end{aligned}$$

In examples we will also write $f(z, x_1, \ldots, x_n) = R(h, g)(z, x_1, \ldots, x_n)$.

The following well known theorem establishes connection between inductively defined recursive functions and Turing machines used in Definition 2.1.

Theorem 2.3 *The family of partial recursive functions is the smallest family of integer-to-integer functions with the following properties:*

(i) It contains the nullary constant, the successor function and the projection functions.

(ii) It is closed under the operations composition, primitive recursion and minimalization*, defined as follows:*

If $h : \mathcal{N}^{n+1} \longrightarrow \mathcal{N}$ *is a partial recursive function, then so is the function* $f : \mathcal{N}^n \longrightarrow \mathcal{N}$*, where*

$$f(x_1, \ldots, x_n) = \min\{y \in \mathcal{N} \,|h(x_1, \ldots, x_n, y) = 0\},$$

and $h(x_1, \ldots, x_n, z)$ *is defined for all integers* z, $0 \leq z \leq y$. *Otherwise,* $f(x_1, \ldots, x_n)$ *is undefined.* f *is usually written in the form*

$$f(x_1, \ldots, x_n) = \mu y[h(x_1, \ldots, x_n, y) = 0].$$

In examples we will also write $f(x_1, \ldots, x_n) = M(h)(x_1, \ldots, x_n)$.

Moreover, it is known that any partial recursive function can be reduced to a function of one variable by using the *pairing* and *de-pairing* functions. For more details we refer to [8, 2]. Kleene's theorem shows the existence of an universal function.

Theorem 2.4 (Kleene) *There exist primitive recursive functions* g *and* h *such that for each partial recursive function* f *of one variable there is an integer* i_f *such that*

$$f(x) = g(\mu y[h(x, i_f, y) = 0]).$$

3 Main results

We show that any partial recursive function can be computed by a Watson-Crick D0L scheme and, moreover, that there exists an universal Watson-Crick D0L scheme computing all partial recursive functions.

Note first the following important fact: it is known that any partial recursive function can be reduced to one variable domain. To apply the operations in section 2 to such a functions, it is enough to have the composition for $m = 2$, the primitive recursion for $n = 1$ and the minimalization for $n = 1$. The same is necessary to construct the pairing and de-paring functions (see [8, 2] for details). Therefore we show the construction of these operations for the described cases.

Definition 3.1 *A (partial) function $f : \mathcal{N}^n \longrightarrow \mathcal{N}$ is said to be computed by a Watson-Crick D0L scheme $G = (\Sigma, p)$, if there are $\$, \#, A_{[1]}, \ldots, A_{[n]}, Z \in \Sigma$ such that $y = f(x_1, \ldots, x_n)$ if and only if*

$$\$A_{[1]}^{x_1} A_{[2]}^{x_2} \ldots A_{[n]}^{x_n} \Longrightarrow_G^* \#Z^y \Longrightarrow_G \#Z^y$$

and the symbols $Z, \#$ do not appear in the derivation sequence $S(G, \$A_{[1]}^{x_1} A_{[2]}^{x_2} \ldots A_{[n]}^{x_n})$ until the string $\#Z^y$ is produced.

Notice that this method of representing functions is different from D0L growth functions which are frequently used in the cited references. Our method is rather similar to the representation used with Turing machine. Notice also that it follows from the above definition that $p(\#) = \#$, $p(Z) = Z$.

Now we describe informally the construction of a Watson-Crick D0L scheme $G = (\Sigma, p)$ computing a given partial recursive function f composed from the base functions and the operations defined above. The following problem arises: in the operations in Section 2 a function f_j can be used as a part of another function f_i definition (we will also write that f_j is *called* by f_i). In a corresponding Watson-Crick D0L scheme computing f_i by generating a string w, this string can simultaneously contain distinct parts w_i and w_j describing a current state of computation of f_i and f_j, respectively. Operations over w_j performed during computation of f_j must not influence w_i until f_j is finished. But when a complementation step occurs during computation of f_j, it affects the whole string w and hence also the part w_i. To solve this situation, we must adopt the following rules:

- Each occurrence of a base function or a result of an operation over functions within our composed function is considered as a separate function f_i with an unique index i.
- Each such a function f_i will be computed by a certain Watson-Crick sub-scheme (Σ_i, p_i) for some $i \geq 1$. If a function f_i calls a function f_j, then $\Sigma_j \subset \Sigma_i$ and $(p_j - \{(\#_j, \#_j), (Z_j, Z_j)\}) \subset p_i$ (remind that $p_j(\#_j) = \#_j$,

$p_j(Z_j) = Z_j$). For further use we fix the following notation: If w is a string over $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_n$, then for any i, $1 \leq i \leq n$ we denote

$$w_i = g_i(w), \qquad \text{where} \qquad g_i(a) = \begin{cases} a, & \text{for } a \in \Sigma_i - \bigcup_{j \neq i} \Sigma_j, \\ \lambda & \text{otherwise.} \end{cases}$$

- The called function must always have higher index than the calling function. Precisely:
  - if $f_i(x_1, \ldots, x_n) = f_j(f_k(x_1, \ldots, x_n), f_\ell(x_1, \ldots, x_n))$, then $i < j$, $i < k$ and $i < \ell$;
  - if f_i is a result of primitive recursion over f_j and f_k, then $i < j$ and $i < k$;
  - if $f_i(x_1, \ldots, x_n) = \mu y[f_j(x_1, \ldots, x_n, y) = 0]$, then $i < j$.
- Whenever a function $f_i : \mathcal{N}^n \longrightarrow \mathcal{N}$ wants to call a function $f_j : \mathcal{N}^m \longrightarrow \mathcal{N}$ with the arguments $x_1, \ldots, x_m$, then p_i generates the string $\$_j A_{j[1]}^{x_1} \ldots A_{j[m]}^{x_m}$. Simultaneously p_i rewrites all other symbols of w_i such that $|w_i|_{\Sigma_i^{\mathrm{PUR}}} = |w_i|_{\Sigma_i^{\mathrm{PYR}}}$ and $p_i(w_i) = w_i$ ("sleeping" state). In each moment only *one* such function f_j is called by f_i. It follows due to transitivity that when f_j is being computed (and it is not calling some f_k), then $w_j \in PUR$ iff $w \in PUR$ and the same holds for PYR. (Remind that w_i, w_j are scattered substrings of w.)
- We define $p_i(\#_j Z_j^x) \in PYR$ for any $x \geq 0$ (see below proof for details). Hence if p_j finishes computation of f_j and produces a result string $\#_j Z_j^{f_j(x_1,\ldots,x_m)}$, then complementation step occurs. Then w_i is rewritten to $h(w_i)$ and this is a signal to leave "sleeping" state and to continue the computation of f_i.
- A complementation step can occur also during computation of f_j before f_j is finished, and in such a situation also w_i is rewritten to $h(w_i)$. Now p_i must be able to distinguish whether f_j has already finished or not. Hence p_i must perform a test T_i, in which *another* complementation occurs if and only if $\#_j$ was previously in w. If not, then it is necessary to restore the previous sleeping state of w_i.
- Now consider, that f_i itself is called from some f_k computed by (Σ_k, p_k) for $k < i$, and in the described situation p_k tries to perform a similar test T_k since the complementation in f_j affects both w_i and w_k. But these tests cannot be done simultaneously, because the conditions $w_i \in PUR$ iff $w \in PUR$ and $w_k \in PUR$ iff $w \in PUR$ may not be simultaneously satisfied for an arbitrary w_i and w_k.

  The possible solution is that each p_ℓ (for $\ell = i, j, k, \ldots$) must perform i non-complementation steps after any complementation. During these i steps $w_i \in PUR_i$ and p_k can perform test T_k, since $k < i$. The situation is described at figure 1.

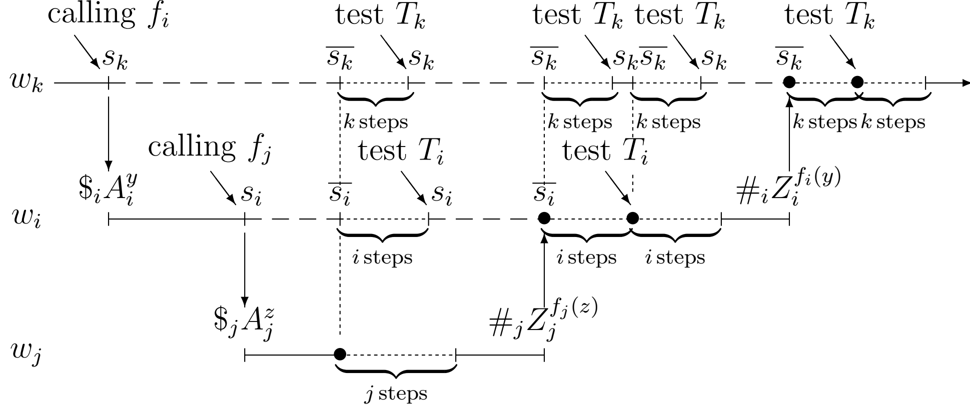


Figure 1: Calling of $f_i(y)$ by $f_k(x)$ and $f_j(z)$ by $f_i(y)$. s_k and s_i are sleeping states of w_k and w_i, respectively. We denote $\overline{s_k} = h(s_k)$ and $\overline{s_i} = h(s_i)$. T_k and T_i are tests of presence of $\#_i$ and $\#_j$, respectively, in w. $\bullet$ is a complementation step. Solid lines denote computing a function, dashed lines denote a sleeping state, dotted horizontal lines denote waiting steps after complementation.

- Due to transitivity this solution keeps valid also if a function f_{i_1} calls f_{i_2} which calls f_{i_3} ...which calls f_{i_k}, where $i_1 < i_2 < i_3 < \ldots < i_k$. If the previous conditions is satisfied, there is no interference between schemes computing these functions.

Lemma 3.2 *There are Watson-Crick D0L schemes computing (i) the nullary constant 0, (ii) the successor function $S(x)$, (iii) the projection function $U_i^n(x_1, \ldots, x_n)$.*

Proof.

(i) Consider the scheme $G = (\{\$, A_{[1]}, \#, \overline{\$}, \overline{A_{[1]}}, \overline{\#}\}, p)$ such that $p(\$) = \#$, $p(A_{[1]}) = \lambda$ and $p(X) = X$ for all other $X \in \Sigma$.

(ii) Consider the scheme $G = (\{\$, \#\, A_{[1]}, Z, \overline{\$}, \overline{\#}, \overline{A_{[1]}}, \overline{Z}\}, p)$ such that $p(\$) = \#Z$, $p(A_{[1]}) = Z$ and $p(X) = X$ for all other $X \in \Sigma$.

(iii) Consider the scheme $G = (\{\$, \#\, A_{[1]}, \ldots, A_{[n]}, Z, \overline{\$}, \overline{\#}, \overline{A_{[1]}}, \ldots, \overline{A_{[n]}}, \overline{Z}\}, p, \$)$ such that $p(\$) = \#$, $p(A_{[i]}) = Z$, $p(A_{[j]}) = \lambda$ for $j \neq i$ and $p(X) = X$ for all other $X \in \Sigma$.

□

The following proposition shows an instructive exception from the above rules when the construction of the composition operation is much simpler than in the general case proven later.

Proposition 3.3 *Consider a composition $f_i(x_1, \ldots, x_n) = f_j(f_{k_1}(x_1, \ldots, x_n), \ldots, f_{k_m}(x_1, \ldots, x_n))$, where functions f_j, $f_{k_1}, \ldots, f_{k_m}$ are computed by the Watson-Crick D0L schemes (Σ_j, p_j), $(\Sigma_{k_1}, p_{k_1}), \ldots, (\Sigma_{k_1}, p_{k_1})$. Let at least one of the following conditions is satisfied:*

(i) $m = 1$;

(ii) all the functions $f_{k_1}, \ldots, f_{k_m}$ *are computed in the same number of steps without using complementation.*

Then there exists a scheme (Σ_i, p_i) *computing the function* f_i.

Proof. Due to the condition (i) or (ii) we can omit waiting cycles and tests in the informal description following definition 3.1. Also all the functions $f_{k_1}, \ldots, f_{k_m}$ are called simultaneously unlike the above rules state. Let

$$\Sigma_i = \Sigma \cup \Sigma_j \cup \Sigma_{k_1} \cup \ldots \cup \Sigma_{k_m}, \qquad p_i = p \cup p'_j \cup p'_{k_1} \cup \ldots \cup p'_{k_m},$$

where $p'_\ell = p_\ell - \{(\#_\ell, \#_\ell), (Z_\ell, Z_\ell)\}$ for $\ell \in \{j, k_1, \ldots, k_m\}$. Let $\Sigma^{\mathrm{PUR}} = \{\$_i, \#_i, A_{i[1]}, \ldots, A_{i[n]}, Z_i\}$ and let Σ^{PYR} contain their complement counterparts. The partial morphism p over Σ_i is defined by

$$\begin{array}{llll} p(\$_i) = \$_{k_1} \ldots \$_{k_m} & p(\#_{k_1}) = \$_j & p(Z_{k_1}) = A_{j[1]} & p(\#_j) = \#_i \\ p(A_{i[1]}) = A_{k_1[1]} \ldots A_{k_m[1]} & p(\#_{k_2}) = \lambda & p(Z_{k_2}) = A_{j[2]} & p(Z_j) = Z_i \\ \vdots & \vdots & \vdots & \\ p(A_{i[n]}) = A_{k_1[n]} \ldots A_{k_m[n]} & p(\#_{k_m}) = \lambda & p(Z_{k_m}) = A_{j[m]} & \end{array}$$

and $p(X) = X$ for all other $X \in \Sigma$. p is undefined for all $X \in \Sigma_i - \Sigma$ that are not contained in the above list. Hence for all $X \in \Sigma$ we have an unique value of $p_i(X)$ and p_i is a morphism. Function of the system (Σ_i, p_i) is obvious. □

Example 3.4 *The following scheme* (Σ, p) *computes the function* $f(x) = x + 2 = I(I(x))$, *or more explicitly* $f(x) = C(I, I)(x)$. *Assigning an unique index to each occurrence of a function or an operation due to the above rules, we have* $f(x) = C_1(I_2, I_3)(x)$. *An input adopts the form* $\$_1 A^x_{1[1]}$, *an output is* $\#_1 Z_1^{x+2}$. *Let* $\Sigma^{\mathrm{PUR}} = \{\$_1, \$_2, \$_3, \#_1, \#_2, \#_3, A_{1[1]}, A_{2[1]}, A_{3[1]}, Z_1, Z_2, Z_3\}$ *and let*

$$\begin{array}{llll} p(\$_1) = \$_3 & p(\$_3) = \#_3 Z_3 & p(A_{3[1]}) = Z_3 & p(Z_3) = A_{2[1]} \\ p(\#_3) = \$_2 & p(\$_2) = \#_2 Z_2 & p(A_{2[1]}) = Z_2 & p(Z_2) = Z_1 \\ p(\#_2) = \#_1 & & & \end{array}$$

plus completing rules.

Lemma 3.5 *Let* $f_j(z, x, y)$, $f_k(y)$ *be partial recursive functions computed by the Watson-Crick D0L schemes* (Σ_j, p_j) *and* (Σ_k, p_k), *respectively, for some* $j, k > 1$. *Then there exist a Watson-Crick D0L scheme* (Σ_i, p_i), *where* $i < j$ *and* $i < k$, *computing the function* $f_i(x, y) = R(f_k, f_j)(x, y)$.

Proof. As a basic principle, we replace recurrent calling of f_i by a cycle, hence $f_i(x, y)$ is computed as follows:

1. Assign $z := 0$.
2. Compute $f_i(z, y) := f_k(y)$.

3. While $z < x$ do:
   compute $f_i(z+1, y) := f_j(z, f_i(z, y), y)$ and
   assign $z := z + 1$.

Complementation is used to control the loop, i.e. to test whether $z < x$. Other complementations are used to test whether functions f_k, f_j have finished their computation or not, as explained above.

Let $\Sigma_i = \Sigma \cup \Sigma_j \cup \Sigma_k$ and let $p_i = p \cup p'_j \cup p'_k$, where $p'_j = p_j - \{(\#_j, \#_j), (Z_j, Z_j)\}$ and $p'_k = p_k - \{(\#_k, \#_k), (Z_k, Z_k)\}$. Let

$$\Sigma^{\mathrm{PUR}} = \{\$_i, \#_i, A_{i[1]}, A_{i[2]}, Z_i, S, A, B, C, D, E, F, G, H, I, J, K, L, S_{(0)}, \ldots, S_{(2i+1)}, B_{(0)}, \ldots, B_{(2i+1)}, F_{(1)}, \ldots, F_{(2i+1)}, H_{(1)}, \ldots, H_{(2i+1)}, J_{(1)}, \ldots, J_{(2i+1)}, Z_{(0)}, \ldots, Z_{(i)}, \#_{(0)}, \ldots, \#_{(i)}\}$$

and let Σ^{PYR} contains their complement counterparts. We consider $\Sigma \cap \Sigma_j = \Sigma \cap \Sigma_k = \emptyset$.

The description of the partial morphism p over Σ_i follows. We use plain lower indices $_{j}, _{k}$ to denote symbols of Σ_j and Σ_k, respectively. Lower indices in braces $_{(0)}, \ldots, _{(i)}$ denote symbol for counting cycles and lower indices in square braces $_{[1]}, _{[2]}$ denote order of function parameters.

$\phi(w)$	Productions of p	Comments
$\$_i A^x_{i[1]} A^y_{i[2]}$		We start with an arguments x, y.
$\Downarrow$	$p(\$_i) = \$_k$ $p(A_{i[2]}) = I\overline{J}A_{k[1]}$ $p(A_{i[1]}) = G\overline{H}$	We generate the starting string of f_k. Notice that $\lvert w_i\rvert_{\Sigma_i^{\mathrm{PUR}}} = \lvert w_i\rvert_{\Sigma_i^{\mathrm{PYR}}}$ holds
$\$_k A^y_{k[1]} (G\overline{H})^x (I\overline{J})^y$		
$\Downarrow$ $\vdots$ $\Downarrow$	$p(G) = G$ $p(\overline{H}) = \overline{H}$ $p(I) = I$ $p(\overline{J}) = \overline{J}$	Computation of $f_k(y)$ via p_k is performed with w_i in a sleeping state.
$\#_k Z_k^{f_k(y)} (G\overline{H})^x (I\overline{J})^y$		
$\Downarrow$	$p(\#_k) = \overline{S_{(0)}}$ $p(Z_k) = \overline{B_{(0)}}$	We consider $z = 0$ and assign $f_i(z, y) := f_k(y)$. We run a complementation step to break the sleeping state of f_i.
① $(\overline{E}F)^z S_{(0)} B_{(0)}^{f_i(z,y)} (\overline{G}H)^x (\overline{I}J)^y$		
$\Downarrow$	$p(F) = F_{(1)}$, $p(\overline{E}) = \lambda$ $p(S_{(0)}) = S_{(1)}$, $p(B_{(0)}) = B_{(1)}$ $p(H) = H_{(1)}$, $p(\overline{G}) = \lambda$ $p(J) = J_{(1)}$, $p(\overline{I}) = \lambda$	
$F^z_{(1)} S_{(1)} B_{(1)}^{f_i(z,y)} H^x_{(1)} J^y_{(1)}$		

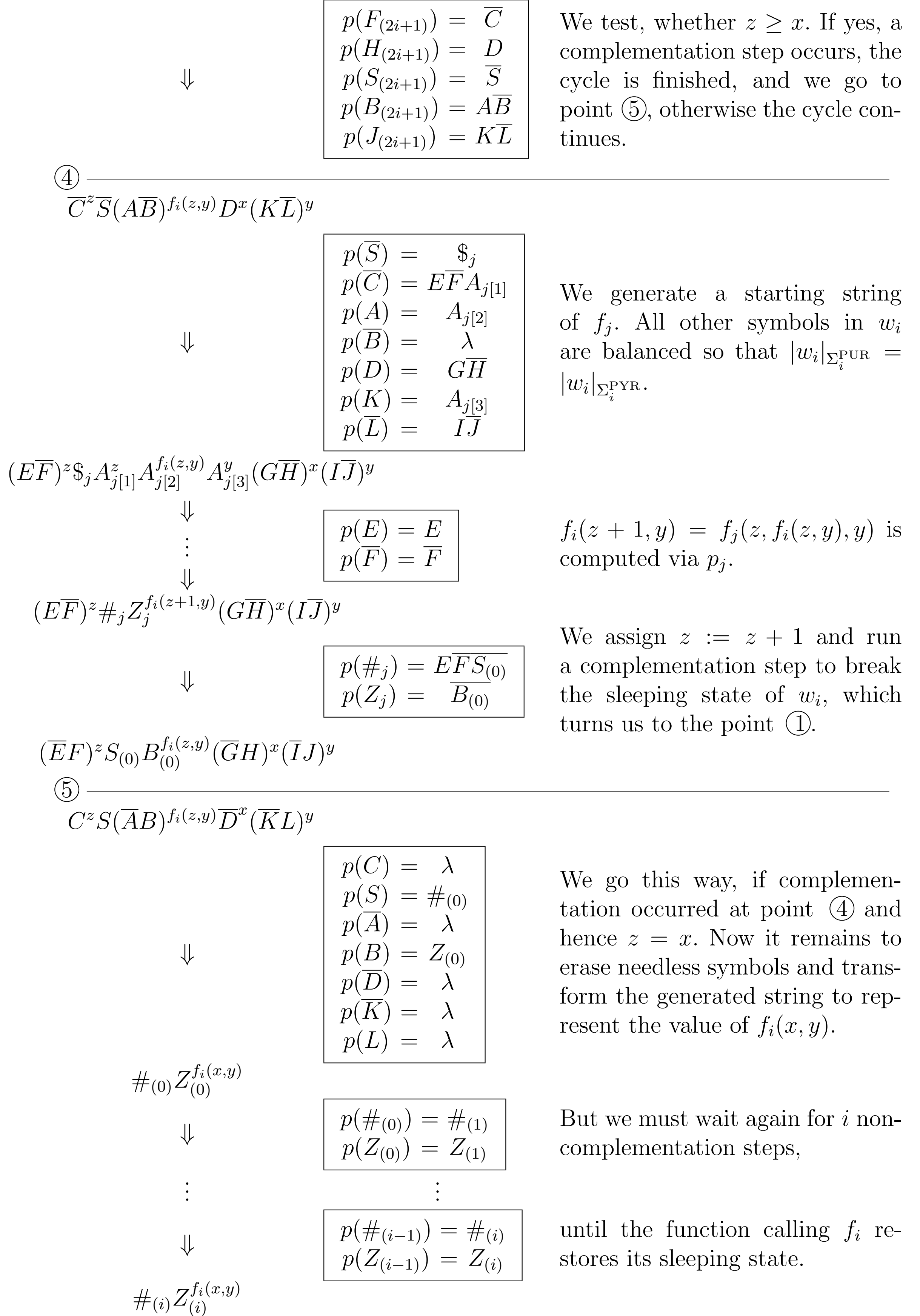
$\Downarrow$	$p(F_{(1)}) = F_{(2)}$ $p(H_{(1)}) = H_{(2)}$ $p(S_{(1)}) = S_{(2)}$ $p(B_{(1)}) = B_{(2)}$ $p(J_{(1)}) = J_{(2)}$	Now after complementation we must wait for i steps, during which a complementation must not occur.
$\vdots$	$\vdots$	
$\Downarrow$	$p(F_{(i-1)}) = F_{(i)}$ $p(H_{(i-1)}) = H_{(i)}$ $p(S_{(i-1)}) = S_{(i)}$ $p(B_{(i-1)}) = B_{(i)}$ $p(J_{(i-1)}) = J_{(i)}$	During these i steps the functions calling f_i must return their substrings to their sleeping states.
$F^z_{(i)} S_{(i)} B^{f_i(z,y)}_{(i)} H^x_{(i)} J^y_{(i)}$		
$\Downarrow$	$p(F_{(i)}) = E\overline{F_{(i+1)}}$ $p(H_{(i)}) = G\overline{H_{(i+1)}}$ $p(S_{(i)}) = \overline{S_{(i+1)}}$ $p(B_{(i)}) = \overline{B_{(i+1)}}$ $p(J_{(i)}) = I\overline{J_{(i+1)}}$	Now we test, whether the previous complementation step occurred due to finishing of the function f_j (f_k) or not. If yes, we go to point ③.

② ———

$(E\overline{F_{(i+1)}})^z w_j (G\overline{H_{(i+1)}})^x (I\overline{J_{(i+1)}})^y$		
$\Downarrow$	$p(\overline{F_{(i+1)}}) = \overline{F}$ $p(\overline{H_{(i+1)}}) = \overline{H}$ $p(\overline{J_{(i+1)}}) = \overline{J}$	If not, there are no symbols $\overline{S_{(i+1)}}$ and $\overline{B_{(i+1)}}$ in w_i and we restore the sleeping state of w_i it had before the point ①.
$(E\overline{F})^z w_j (G\overline{H})^x (I\overline{J})^y$		

③ ———

$(\overline{E}F_{(i+1)})^z S_{(i+1)} B^{f_i(z,y)}_{(i+1)} (\overline{G}H_{(i+1)})^x (\overline{I}J_{(i+1)})^y$		
$\Downarrow$	$p(F_{(i+1)}) = F_{(i+2)}$ $p(H_{(i+1)}) = H_{(i+2)}$ $p(S_{(i+1)}) = S_{(i+2)}$ $p(B_{(i+1)}) = B_{(i+2)}$ $p(J_{(i+1)}) = J_{(i+2)}$	If at point ① f_j was finished, there was the symbol $\overline{S_{(i+1)}}$ in w_i and another complementation occurred at ②.
$\vdots$	$\vdots$	
$\Downarrow$	$p(F_{(2i)}) = F_{(2i+1)}$ $p(H_{(2i)}) = H_{(2i+1)}$ $p(S_{(2i)}) = S_{(2i+1)}$ $p(B_{(2i)}) = B_{(2i+1)}$ $p(J_{(2i)}) = J_{(2i+1)}$	Again we must wait after complementation for i steps, during which a complementation must not occur.
$F^z_{(2i+1)} S_{(2i+1)} B^{f_i(z,y)}_{(2i+1)} H^x_{(2i+1)} J^y_{(2i+1)}$		



$p(F_{(2i+1)}) = \overline{C}$
$p(H_{(2i+1)}) = D$
$p(S_{(2i+1)}) = \overline{S}$
$p(B_{(2i+1)}) = A\overline{B}$
$p(J_{(2i+1)}) = K\overline{L}$
We test, whether $z \geq x$. If yes, a complementation step occurs, the cycle is finished, and we go to point ⑤, otherwise the cycle continues.
④
$\overline{C}^z\overline{S}(A\overline{B})^{f_i(z,y)}D^x(K\overline{L})^y$
$p(\overline{S}) = \$_j$
$p(\overline{C}) = E\overline{F}A_{j[1]}$
$p(A) = A_{j[2]}$
$p(\overline{B}) = \lambda$
$p(D) = G\overline{H}$
$p(K) = A_{j[3]}$
$p(\overline{L}) = I\overline{J}$
We generate a starting string of f_j. All other symbols in w_i are balanced so that $|w_i|_{\Sigma_i^{\mathrm{PUR}}} = |w_i|_{\Sigma_i^{\mathrm{PYR}}}$.
$(E\overline{F})^z\$_jA^z_{j[1]}A^{f_i(z,y)}_{j[2]}A^y_{j[3]}(G\overline{H})^x(I\overline{J})^y$
$p(E) = E$
$p(\overline{F}) = \overline{F}$
$f_i(z+1,y) = f_j(z, f_i(z,y), y)$ is computed via p_j.
$(E\overline{F})^z\#_jZ_j^{f_i(z+1,y)}(G\overline{H})^x(I\overline{J})^y$
$p(\#_j) = E\overline{F}\overline{S_{(0)}}$
$p(Z_j) = \overline{B_{(0)}}$
We assign $z := z+1$ and run a complementation step to break the sleeping state of w_i, which turns us to the point ①.
$(\overline{E}F)^zS_{(0)}B_{(0)}^{f_i(z,y)}(\overline{G}H)^x(\overline{I}J)^y$
⑤
$C^zS(\overline{A}B)^{f_i(z,y)}\overline{D}^x(\overline{K}L)^y$
$p(C) = \lambda$
$p(S) = \#_{(0)}$
$p(\overline{A}) = \lambda$
$p(B) = Z_{(0)}$
$p(\overline{D}) = \lambda$
$p(\overline{K}) = \lambda$
$p(L) = \lambda$
We go this way, if complementation occurred at point ④ and hence $z = x$. Now it remains to erase needless symbols and transform the generated string to represent the value of $f_i(x,y)$.
$\#_{(0)}Z_{(0)}^{f_i(x,y)}$
$p(\#_{(0)}) = \#_{(1)}$
$p(Z_{(0)}) = Z_{(1)}$
But we must wait again for i non-complementation steps,
$p(\#_{(i-1)}) = \#_{(i)}$
$p(Z_{(i-1)}) = Z_{(i)}$
until the function calling f_i restores its sleeping state.
$\#_{(i)}Z_{(i)}^{f_i(x,y)}$

$$\Downarrow \qquad \boxed{\begin{array}{rcl} p(\#_{(i)}) &=& \#_i \\ p(Z_{(i)}) &=& Z_i \end{array}} \qquad \text{We produce an output and finish.}$$

$$\#_i Z_i^{f_i(x,y)}$$

Moreover $p(X) = X$ for all other $X \in \Sigma$. $p(X)$ is undefined for all $X \in (\Sigma_j \cup \Sigma_k - \{\#_j, Z_j, \#_k, Z_k\})$.

It follows from the previous description that the described scheme (Σ_i, p_i) performs the cyclic computation of f_i described at the beginning of the proof. Notice that this scheme follows the rules given at the beginning of this section, which prevents problems with mutual interference of called and calling functions. □

We considered $\Sigma \cap \Sigma_k = \Sigma \cap \Sigma_j = \emptyset$ in the above proof. To guarantee this in the following examples, we add to all symbols of Σ the lower index i (the symbols $\$_i, \#_i, A_{i[m]}, Z_i$ have it assigned already in the proofs).

Example 3.6 *Predecessor function can be constructed as*

$$\begin{array}{rcl} P(0) &=& 0 \\ P(z+1) &=& U_1^2(z, P(z)) \end{array}$$

i.e. $P(x) = R_1(0_2, U^2_{1(3)})(x)$. *Here* $U^2_{1(3)}$ *is the projection of a first variable from two, which has assigned unique index 3. The alphabet of the scheme* (Σ, p) *computing predecessor is*

$$\Sigma^{\mathrm{PUR}} = \{\$_1, S_1, A_1, B_1, C_1, D_1, E_1, F_1, G_1, H_1, I_1, J_1, K_1, L_1, A_{1[1]}, A_{1[2]}, S_{1[0]}, S_{1[1]}, S_{1[2]}, S_{1[3]}, B_{1[0]}, B_{1[1]}, B_{1[2]}, B_{1[3]}, F_{1[1]}, F_{1[2]}, F_{1[3]}, H_{1[1]}, H_{1[2]}, H_{1[3]}, J_{1[1]}, J_{1[2]}, J_{1[3]}, Z_{1[0]}, Z_{1[1]}, \#_{1[0]}, \#_{1[1]}, Z_1, \#_1 \$_2, A_{2[1]}, \$_3, A_{3[1]}, A_{3[2]}, A_{3[3]}, \#_3, Z_3, \#_2, Z_2\},$$

the morphism p *is defined by*

$$\begin{array}{llll}
p(\$_1) = \$_2 & p(A_{1[2]}) = I_1\overline{J_1}A_{2[1]} & p(A_{1[1]}) = G_1\overline{H_1} & p(Z_2) = \overline{B_{1[0]}} \\
p(\#_2) = \overline{S_{1[0]}} & p(F_1) = F_{1[1]} & p(H_1) = H_{1[1]} & p(J_1) = J_{1[1]} \\
p(\overline{E_1}) = \lambda & p(\overline{G_1}) = \lambda & p(\overline{I_1}) = \lambda & p(S_{1[0]}) = S_{1[1]} \\
p(B_{1[0]}) = B_{1[1]} & p(F_{1[1]}) = E_1\overline{F_{1[2]}} & p(H_{1[1]}) = G_1\overline{H_{1[2]}} & p(S_{1[1]}) = \overline{S_{1[2]}} \\
p(B_{1[1]}) = \overline{B_{1[2]}} & p(J_{1[1]}) = I_1\overline{J_{1[2]}} & p(\overline{F_{1[2]}}) = \overline{F_1} & p(\overline{H_{1[2]}}) = \overline{H_1} \\
p(\overline{J_{1[2]}}) = \overline{J_1} & p(S_{1[2]}) = S_{1[3]} & p(B_{1[2]}) = B_{1[3]} & p(F_{1[2]}) = F_{1[3]} \\
p(H_{1[2]}) = H_{1[3]} & p(J_{1[2]}) = J_{1[3]} & p(F_{1[3]}) = \overline{C_1} & p(H_{1[3]}) = D_1 \\
p(S_{1[3]}) = \overline{S_1} & p(B_{1[3]}) = A_1\overline{B_1} & p(J_{1[3]}) = K_1\overline{L_1} & p(\overline{S_1}) = \$_3 \\
p(\overline{C_1}) = E_1\overline{F_1}A_{3[1]} & p(A_1) = A_{3[2]} & p(\overline{B_1}) = \lambda & p(D_1) = G_1\overline{H_1} \\
p(K_1) = A_{3[3]} & p(\overline{L_1}) = I_1\overline{J_1} & p(\#_3) = E_1\overline{F_1 S_{1[0]}} & p(Z_3) = \overline{B_{1[0]}} \\
p(C_1) = \lambda & p(S_1) = \#_{1[0]} & p(\overline{A_1}) = \lambda & p(B_1) = Z_{1[0]} \\
p(\overline{D_1}) = \lambda & p(\overline{K_1}) = \lambda & p(L_1) = \lambda & p(\#_{1[0]}) = \#_{1[1]} \\
p(Z_{1[0]}) = Z_{1[1]} & p(\#_{1[1]}) = \#_1 & p(Z_{1[1]}) = Z_1 & p(\$_2) = \#_2 \\
p(A_{2[1]}) = \lambda & p(\$_3) = \#_3 & p(A_{3[1]}) = Z_3 & p(A_{3[2]}) = \lambda \\
p(A_{3[3]}) = \lambda & & &
\end{array}$$

and $p(X) = X$ *for all other* $X \in \Sigma$.

Example 3.7 *Subtraction* $\mathrm{Sub}(x,y) = \max\{y-x,0\}$ *can be constructed as*

$$\begin{aligned}\mathrm{Sub}(0,x) &= U_1^1(x)\\ \mathrm{Sub}(z+1,x) &= P(U_2^3(z,\mathrm{Sub}(z,x),x))\end{aligned}$$

i.e. $\mathrm{Sub}(x,y) = R_1(U^1_{1(2)}, C_3(R_4(0_5, U^2_{1(6)}), U^3_{2(7)}))(x,y)$*. Addition can be constructed similarly using successor instead of predecessor, resulting in* $A(x,y) = R_1(U^1_{1(2)}, C_3(S_4, U^3_{2(5)}))(x,y)$*. We do not give a detailed description of corresponding schemes* (Σ, p) *since the number of rules of* p *exceeds 100. They can be constructed following exactly the above proofs.*

Example 3.8 *The function* 2^{2^n}*, which cannot be generated as a D0L length sequence, can be constructed as a composition of two* 2^n *functions, where*

$$\begin{aligned}2^0 &= S(0)\\ 2^{n+1} &= A(U_2^2(2^n,n), U_2^2(2^n,n)).\end{aligned}$$

Lemma 3.9 *Let* $f_j(x,y)$*,* $f_k(x_1,\ldots,x_n)$*,* $f_\ell(x_1,\ldots,x_n)$*, be a partial recursive functions computed by the Watson-Crick D0L schemes* $G_j = (\Sigma_j, p_j)$*,* $G_k = (\Sigma_k, p_k)$*, and* $G_\ell = (\Sigma_\ell, p_\ell)$*, respectively, for some* $j,k,\ell > 1$*. Then there exists Watson-Crick D0L scheme* (Σ_i, p_i) *for some* i *such that* $i < j$*,* $i < k$ *and* $i < \ell$ *computing the composition* $f_i(x_1,\ldots,x_n) = f_j(f_k(x_1,\ldots,x_n), f_\ell(x_1,\ldots,x_n))$*.*

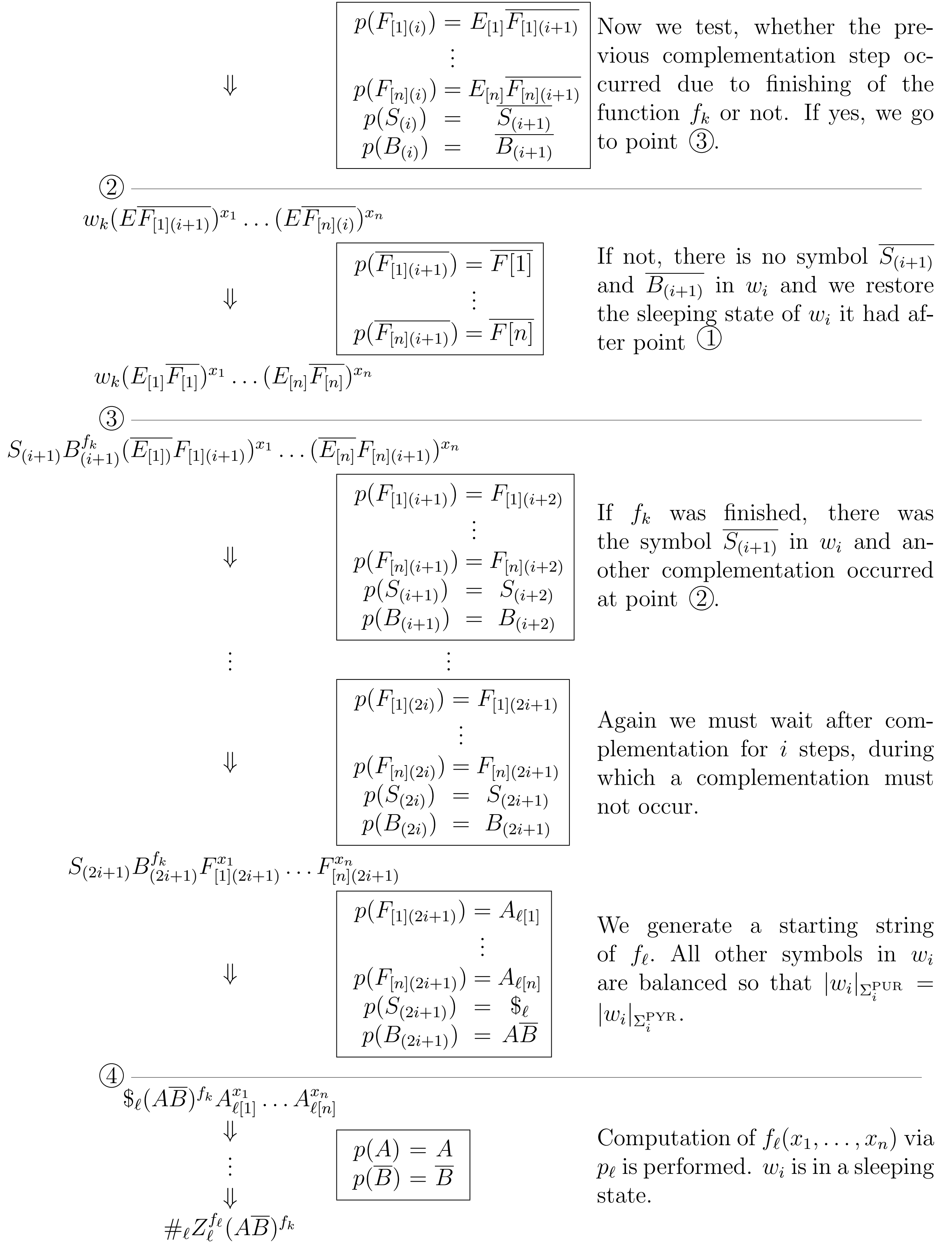
Proof. In the case $m = 2$ we meet the following complication: the functions f_k and f_ℓ cannot be computed in parallel, since complementation may occur during their computation and they could interfere. Hence we call first $f_k(x_1,\ldots,x_n)$, having saved the values $x_1,\ldots,x_n$. After finishing f_k we run complementation to restore the saved parameters and we call $f_\ell(x_1,\ldots,x_n)$, having saved the value returned by f_k. After finishing f_ℓ we run complementation again to restore the saved value and then we call f_j with the results of f_k and f_ℓ. Moreover, we must always check, whether the last complementation was the f_i's one or whether it was performed during computation of f_k or f_j. This needs two more complementations. And between all these complementations we must have waiting cycles, as explained before.

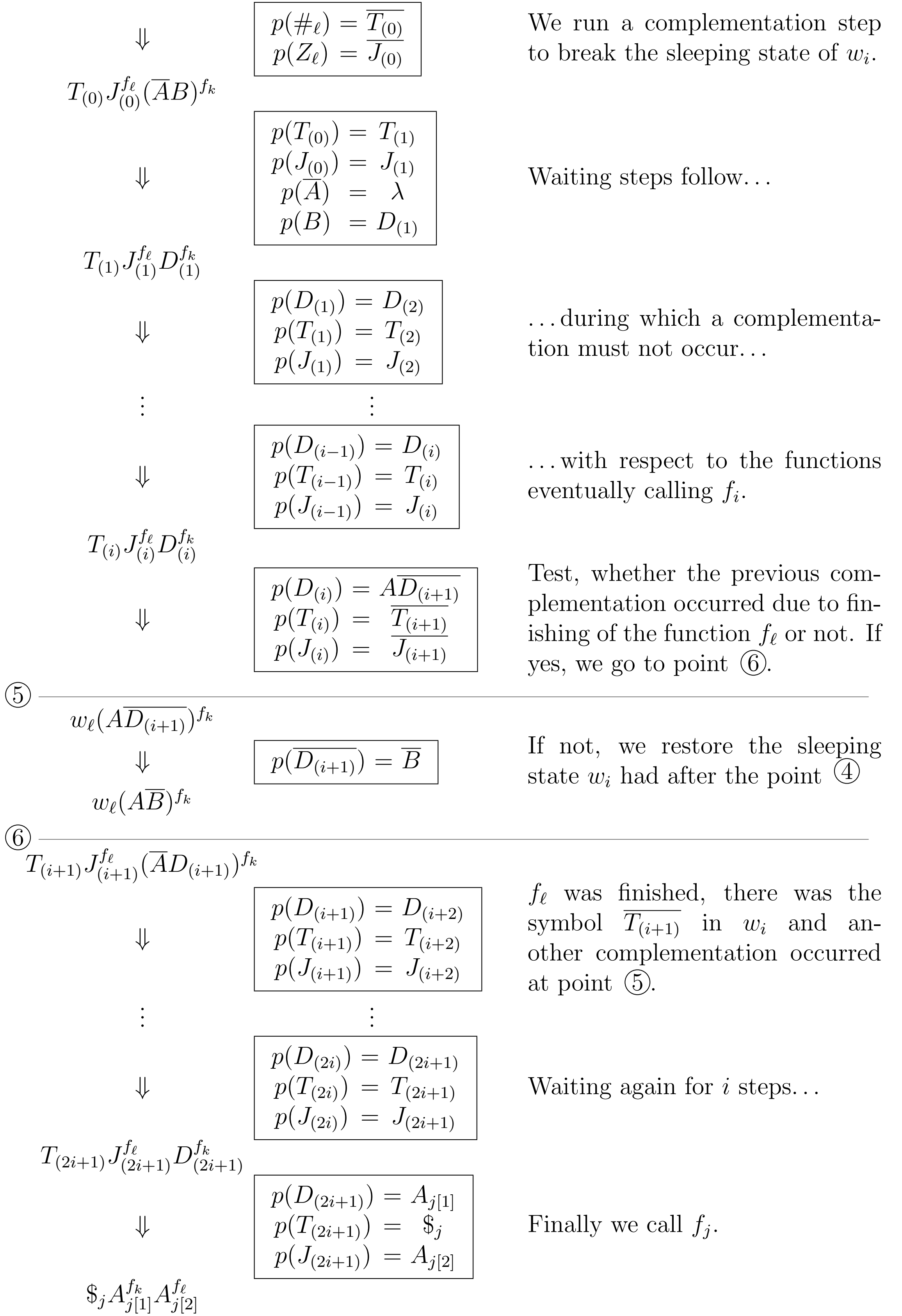
Let $\Sigma_i = \Sigma \cup \Sigma_j \cup \Sigma_k \cup \Sigma_\ell$ and let $p_i = p \cup p'_j \cup p'_k \cup p'_\ell$, where $p'_j = p_j - \{(\#_j,\#_j),(Z_j,Z_j)\}$, $p'_k = p_k - \{(\#_k,\#_k),(Z_k,Z_k)\}$ and $p'_\ell = p_\ell - \{(\#_\ell,\#_\ell),(Z_\ell,Z_\ell)\}$. Let

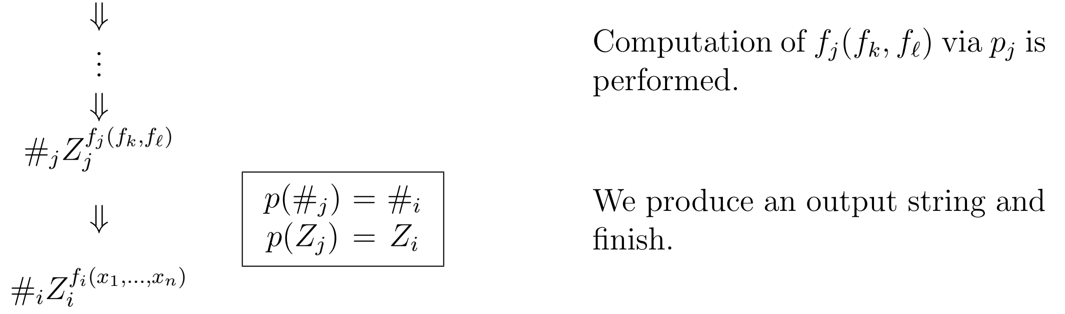
$$\begin{aligned}\Sigma^{\mathrm{PUR}} = & \{\$_i, \#_i, A_{i[1]},\ldots,A_{i[n]}, Z_i, S, A, B, E_{[1]},\ldots,E_{[n]}, F_{[1]},\ldots,F_{[n]},\\ & S_{(0)},\ldots,S_{(2i+1)}, B_{(0)},\ldots,B_{(2i+1)}, D_{(1)},\ldots,D_{(2i+1)}, T_{(0)},\ldots,T_{(2i+1)},\\ & J_{(0)},\ldots,J_{(2i+1)}\}\\ \cup & \{F_{[x](y)} | 1 \le x \le n, 1 \le y \le 2i+1\}\end{aligned}$$

and let Σ^{PYR} contains their complement counterparts. We consider $\Sigma \cap (\Sigma_j \cup \Sigma_k \cup \Sigma_\ell) = \emptyset$. The description of the partial morphism p over Σ_i follows.

$\phi(w)$	Productions of p	Comments
①		
$\$_i A_{i[1]}^{x_1} \dots A_{i[n]}^{x_n}$		We start with arguments $x_1, \dots, x_n$.
$\Downarrow$	$p(\$_i) = \$_k$ $p(A_{i[1]}) = E_{[1]}\overline{F_{[1]}}A_{k[1]}$ $\vdots$ $p(A_{i[n]}) = E_{[n]}\overline{F_{[n]}}A_{k[n]}$	We generate the starting string of f_k. Symbols in w_i are balanced so that $\lvert w_i\rvert_{\Sigma_i^{\mathrm{PUR}}} = \lvert w_i\rvert_{\Sigma_i^{\mathrm{PYR}}}$.
$\$_k A_{k[1]}^{x_1} \dots A_{k[n]}^{x_n}(E_{[1]}\overline{F_{[1]}})^{x_1} \dots (E_{[n]}\overline{F_{[n]}})^{x_n}$		
$\Downarrow$ $\vdots$ $\Downarrow$	$p(E_{[1]}) = E_{[1]}$ $p(\overline{F_{[1]}}) = \overline{F_{[1]}}$ $\vdots$ $p(E_{[n]}) = E_{[n]}$ $p(\overline{F_{[n]}}) = \overline{F_{[n]}}$	Computation of $f_k(x_1, \dots, x_n)$ via p_k is performed. w_i is in a sleeping state.
$\#_k Z_k^{f_k}(E_{[1]}\overline{F_{[1]}})^{x_1} \dots (E_{[n]}\overline{F_{[n]}})^{x_n}$		
$\Downarrow$	$p(\#_k) = \overline{S_{(0)}}$ $p(Z_k) = \overline{B_{(0)}}$	We run a complementation step to break the sleeping state.
$S_{(0)}B_{(0)}^{f_k}(\overline{E_{[1]}}F_{[1]})^{x_1} \dots (\overline{E_{[n]}}F_{[n]})^{x_n}$		
$\Downarrow$	$p(F_{[1]}) = F_{[1](1)}$ $\quad p(\overline{E_{[1]}}) = \lambda$ $\vdots$ $p(F_{[n]}) = F_{[n](1)}$ $\quad p(\overline{E_{[n]}}) = \lambda$ $p(S_{(0)}) = S_{(1)}$ $\quad p(B_{(0)}) = B_{(1)}$	
$S_{(1)}B_{(1)}^{f_k}F_{[1](1)}^{x_1} \dots F_{[n](1)}^{x_n}$		
$\Downarrow$	$p(F_{[1](1)}) = F_{[1](2)}$ $\vdots$ $p(F_{[n](1)}) = F_{[n](2)}$ $p(S_{(1)}) = S_{(2)}$ $p(B_{(1)}) = B_{(2)}$	Now after complementation we must wait for i steps, during which a complementation must not occur.
$\vdots$	$\vdots$	
$\Downarrow$	$p(F_{[1](i-1)}) = F_{[1](i)}$ $\vdots$ $p(F_{[n](i-1)}) = F_{[n](i)}$ $p(S_{(i-1)}) = S_{(i)}$ $p(B_{(i-1)}) = B_{(i)}$	During these i steps the functions calling f_i must restore their sleeping states.
$S_{(i)}B_{(i)}^{f_k}F_{[1](i)}^{x_1} \dots F_{[n](i)}^{x_n}$		


$p(F_{[1](i)}) = E_{[1]}\overline{F_{[1](i+1)}}$
$\vdots$
$p(F_{[n](i)}) = E_{[n]}\overline{F_{[n](i+1)}}$
$p(S_{(i)}) = \overline{S_{(i+1)}}$
$p(B_{(i)}) = \overline{B_{(i+1)}}$
Now we test, whether the previous complementation step occurred due to finishing of the function f_k or not. If yes, we go to point ③.
②
$w_k(E\overline{F_{[1](i+1)}})^{x_1} \dots (E\overline{F_{[n](i)}})^{x_n}$
$p(\overline{F_{[1](i+1)}}) = \overline{F[1]}$
$\vdots$
$p(\overline{F_{[n](i+1)}}) = \overline{F[n]}$
If not, there is no symbol $\overline{S_{(i+1)}}$ and $\overline{B_{(i+1)}}$ in w_i and we restore the sleeping state of w_i it had after point ①
$w_k(E_{[1]}\overline{F_{[1]}})^{x_1} \dots (E_{[n]}\overline{F_{[n]}})^{x_n}$
③
$S_{(i+1)}B_{(i+1)}^{f_k}(\overline{E_{[1]}}F_{[1](i+1)})^{x_1} \dots (\overline{E_{[n]}}F_{[n](i+1)})^{x_n}$
$p(F_{[1](i+1)}) = F_{[1](i+2)}$
$\vdots$
$p(F_{[n](i+1)}) = F_{[n](i+2)}$
$p(S_{(i+1)}) = S_{(i+2)}$
$p(B_{(i+1)}) = B_{(i+2)}$
If f_k was finished, there was the symbol $\overline{S_{(i+1)}}$ in w_i and another complementation occurred at point ②.
$p(F_{[1](2i)}) = F_{[1](2i+1)}$
$\vdots$
$p(F_{[n](2i)}) = F_{[n](2i+1)}$
$p(S_{(2i)}) = S_{(2i+1)}$
$p(B_{(2i)}) = B_{(2i+1)}$
Again we must wait after complementation for i steps, during which a complementation must not occur.
$S_{(2i+1)}B_{(2i+1)}^{f_k}F_{[1](2i+1)}^{x_1} \dots F_{[n](2i+1)}^{x_n}$
$p(F_{[1](2i+1)}) = A_{\ell[1]}$
$\vdots$
$p(F_{[n](2i+1)}) = A_{\ell[n]}$
$p(S_{(2i+1)}) = \$_\ell$
$p(B_{(2i+1)}) = A\overline{B}$
We generate a starting string of f_ℓ. All other symbols in w_i are balanced so that $|w_i|_{\Sigma_i^{\mathrm{PUR}}} = |w_i|_{\Sigma_i^{\mathrm{PYR}}}$.
④
$\$_\ell(A\overline{B})^{f_k}A_{\ell[1]}^{x_1} \dots A_{\ell[n]}^{x_n}$
$p(A) = A$
$p(\overline{B}) = \overline{B}$
Computation of $f_\ell(x_1, \dots, x_n)$ via p_ℓ is performed. w_i is in a sleeping state.
$\#_\ell Z_\ell^{f_\ell}(A\overline{B})^{f_k}$


$p(\#_\ell) = \overline{T_{(0)}}$
$p(Z_\ell) = \overline{J_{(0)}}$
We run a complementation step to break the sleeping state of w_i.
$T_{(0)} J_{(0)}^{f_\ell} (\overline{A}B)^{f_k}$
$p(T_{(0)}) = T_{(1)}$
$p(J_{(0)}) = J_{(1)}$
$p(\overline{A}) = \lambda$
$p(B) = D_{(1)}$
Waiting steps follow...
$T_{(1)} J_{(1)}^{f_\ell} D_{(1)}^{f_k}$
$p(D_{(1)}) = D_{(2)}$
$p(T_{(1)}) = T_{(2)}$
$p(J_{(1)}) = J_{(2)}$
...during which a complementation must not occur...
$p(D_{(i-1)}) = D_{(i)}$
$p(T_{(i-1)}) = T_{(i)}$
$p(J_{(i-1)}) = J_{(i)}$
...with respect to the functions eventually calling f_i.
$T_{(i)} J_{(i)}^{f_\ell} D_{(i)}^{f_k}$
$p(D_{(i)}) = A\overline{D_{(i+1)}}$
$p(T_{(i)}) = \overline{T_{(i+1)}}$
$p(J_{(i)}) = \overline{J_{(i+1)}}$
Test, whether the previous complementation occurred due to finishing of the function f_ℓ or not. If yes, we go to point ⑥.
⑤
$w_\ell(A\overline{D_{(i+1)}})^{f_k}$
$p(\overline{D_{(i+1)}}) = \overline{B}$
$w_\ell(A\overline{B})^{f_k}$
If not, we restore the sleeping state w_i had after the point ④
⑥
$T_{(i+1)} J_{(i+1)}^{f_\ell} (\overline{A} D_{(i+1)})^{f_k}$
$p(D_{(i+1)}) = D_{(i+2)}$
$p(T_{(i+1)}) = T_{(i+2)}$
$p(J_{(i+1)}) = J_{(i+2)}$
f_ℓ was finished, there was the symbol $\overline{T_{(i+1)}}$ in w_i and another complementation occurred at point ⑤.
$p(D_{(2i)}) = D_{(2i+1)}$
$p(T_{(2i)}) = T_{(2i+1)}$
$p(J_{(2i)}) = J_{(2i+1)}$
Waiting again for i steps...
$T_{(2i+1)} J_{(2i+1)}^{f_\ell} D_{(2i+1)}^{f_k}$
$p(D_{(2i+1)}) = A_{j[1]}$
$p(T_{(2i+1)}) = \$_j$
$p(J_{(2i+1)}) = A_{j[2]}$
Finally we call f_j.
$\$_j A_{j[1]}^{f_k} A_{j[2]}^{f_\ell}$



$\phi(w)$	Productions of p	Comments
$\Downarrow$ $\vdots$ $\Downarrow$		Computation of $f_j(f_k, f_\ell)$ via p_j is performed.
$\#_j Z_j^{f_j(f_k,f_\ell)}$		
$\Downarrow$	$p(\#_j) = \#_i$ $p(Z_j) = Z_i$	We produce an output string and finish.
$\#_i Z_i^{f_i(x_1,\ldots,x_n)}$		

Moreover $p(X) = X$ for all other $X \in \Sigma$. $p(X)$ is undefined for $X \in (\Sigma_j \cup \Sigma_k \cup \Sigma_\ell - \{\#_j, Z_j, \#_k, Z_k, \#_\ell, Z_\ell\})$. The scheme (Σ_i, p_i) follows the rules given at the beginning of this section, which prevents mutual interference of called and calling functions. □

See Example 3.11 for an application of the above composition.

Lemma 3.10 *Let $f_j(z, x)$ be a partial recursive function computed by the Watson-Crick D0L scheme $G_j = (\Sigma_j, p_j)$ for some $j > 1$. Then there exists a Watson-Crick D0L scheme (Σ_i, p_i) for some $i < j$ computing the function $f_i(x) = \mu y[f_j(x, y) = 0]$.*

Proof. According to [8] there is only one effective method for computing f_i :

1. Assign $y := 0$.
2. While $f_j(x, y) > 0$ do
   assign $y := y + 1$.
3. Return $f_i(x) := y$.

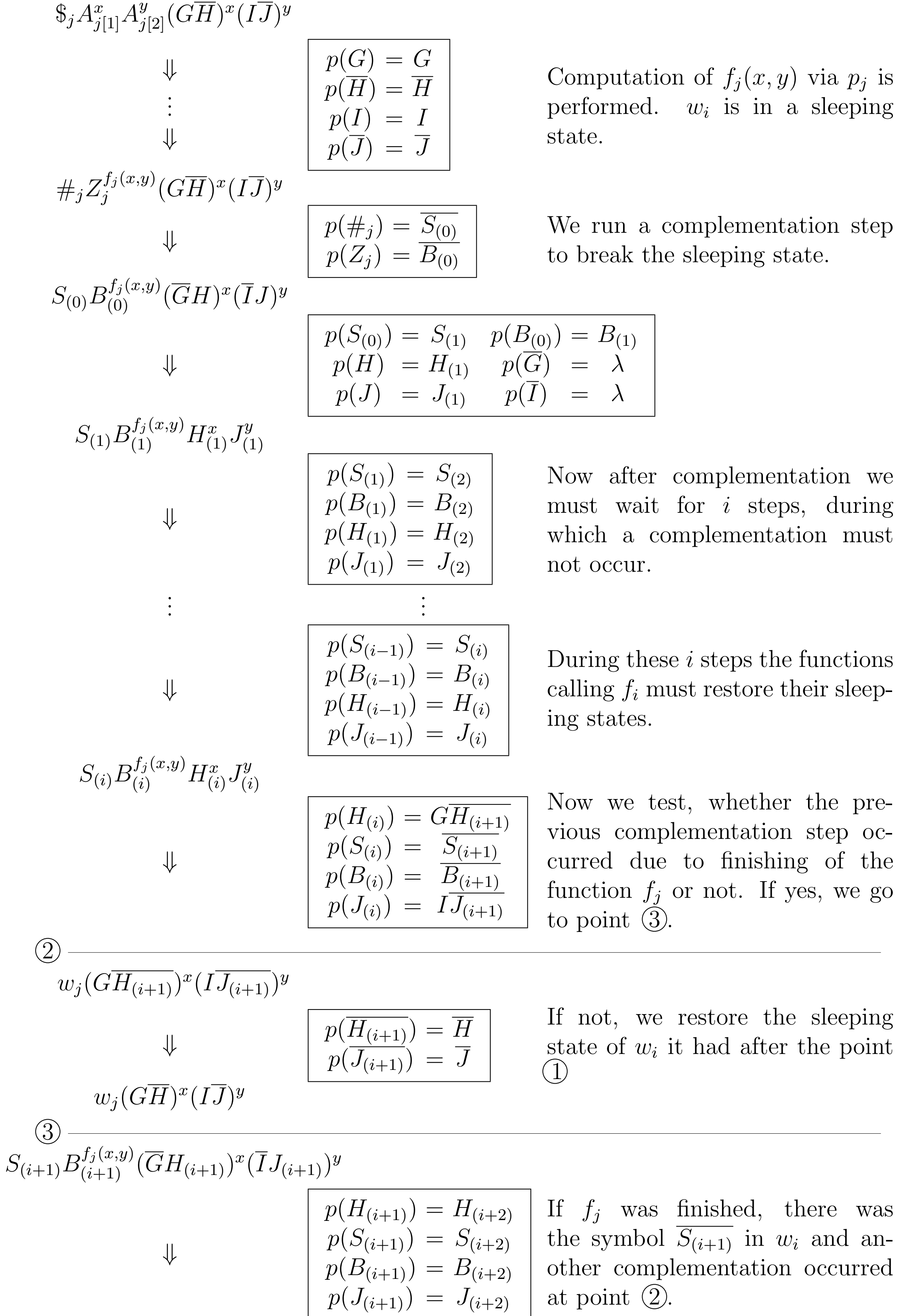
Complementation is used to control the cycle, i.e to test whether $f_j(x, y) > 0$. Other complementations are used to test whether function f_j has finished its computation or not. Let $\Sigma_i = \Sigma \cup \Sigma_j$ and let $p_i = p \cup p'_j$, where $p'_j = p_j - \{(\#_j, \#_j), (Z_j, Z_j)\}$. Let

$$\Sigma^{\mathrm{PUR}} = \{\$_i, \#_i, A_i, Z_i, S, A, B, D, G, H, I, J, K, L, S_{(0)}, \ldots, S_{(2i+1)}, B_{(0)}, \ldots, B_{(2i+1)}, H_{(1)}, \ldots, H_{(2i+1)}, J_{(1)}, \ldots, J_{(2i+1)}, Z_{(0)}, \ldots, Z_{(i)}, \#_{(0)}, \ldots, \#_{(i)}\}$$

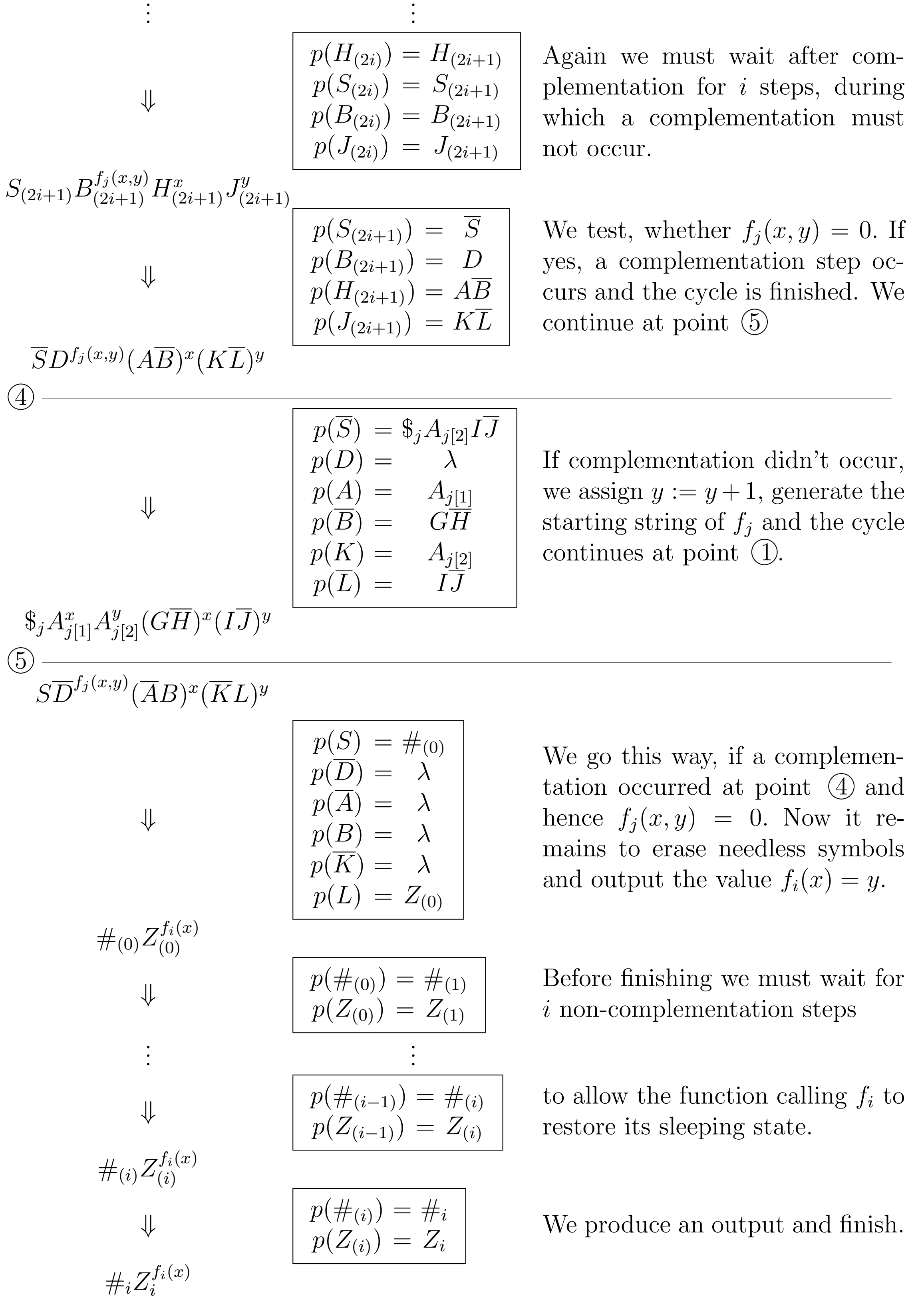
and let Σ^{PYR} contains their complement counterparts. We consider $\Sigma \cap \Sigma_j = \emptyset$. In the following description of the partial morphism p over Σ_i we use the same notation as in the previous proofs.

$\phi(w)$	Productions of p	Comments
$\$_i A_i^x$		We assume $y = 0$.
$\Downarrow$	$p(\$_i) = \$_j$ $p(A_i) = G\overline{H}A_{j[1]}$	We call f_j and rewrite w_i so that $\lvert w_i\rvert_{\Sigma_i^{\mathrm{PUR}}} = \lvert w_i\rvert_{\Sigma_i^{\mathrm{PYR}}}$.

①



$\$_j A^x_{j[1]} A^y_{j[2]} (G\overline{H})^x (I\overline{J})^y$
$p(G) = G$
$p(\overline{H}) = \overline{H}$
$p(I) = I$
$p(\overline{J}) = \overline{J}$
Computation of $f_j(x,y)$ via p_j is performed. w_i is in a sleeping state.
$\#_j Z_j^{f_j(x,y)} (G\overline{H})^x (I\overline{J})^y$
$p(\#_j) = \overline{S_{(0)}}$
$p(Z_j) = \overline{B_{(0)}}$
We run a complementation step to break the sleeping state.
$S_{(0)} B_{(0)}^{f_j(x,y)} (\overline{G}H)^x (\overline{I}J)^y$
$p(S_{(0)}) = S_{(1)}$
$p(B_{(0)}) = B_{(1)}$
$p(H) = H_{(1)}$
$p(\overline{G}) = \lambda$
$p(J) = J_{(1)}$
$p(\overline{I}) = \lambda$
$S_{(1)} B_{(1)}^{f_j(x,y)} H^x_{(1)} J^y_{(1)}$
$p(S_{(1)}) = S_{(2)}$
$p(B_{(1)}) = B_{(2)}$
$p(H_{(1)}) = H_{(2)}$
$p(J_{(1)}) = J_{(2)}$
Now after complementation we must wait for i steps, during which a complementation must not occur.
$p(S_{(i-1)}) = S_{(i)}$
$p(B_{(i-1)}) = B_{(i)}$
$p(H_{(i-1)}) = H_{(i)}$
$p(J_{(i-1)}) = J_{(i)}$
During these i steps the functions calling f_i must restore their sleeping states.
$S_{(i)} B_{(i)}^{f_j(x,y)} H^x_{(i)} J^y_{(i)}$
$p(H_{(i)}) = G\overline{H_{(i+1)}}$
$p(S_{(i)}) = \overline{S_{(i+1)}}$
$p(B_{(i)}) = \overline{B_{(i+1)}}$
$p(J_{(i)}) = I\overline{J_{(i+1)}}$
Now we test, whether the previous complementation step occurred due to finishing of the function f_j or not. If yes, we go to point ③.
②
$w_j (G\overline{H_{(i+1)}})^x (I\overline{J_{(i+1)}})^y$
$p(\overline{H_{(i+1)}}) = \overline{H}$
$p(\overline{J_{(i+1)}}) = \overline{J}$
If not, we restore the sleeping state of w_i it had after the point ①
$w_j (G\overline{H})^x (I\overline{J})^y$
③
$S_{(i+1)} B_{(i+1)}^{f_j(x,y)} (\overline{G}H_{(i+1)})^x (\overline{I}J_{(i+1)})^y$
$p(H_{(i+1)}) = H_{(i+2)}$
$p(S_{(i+1)}) = S_{(i+2)}$
$p(B_{(i+1)}) = B_{(i+2)}$
$p(J_{(i+1)}) = J_{(i+2)}$
If f_j was finished, there was the symbol $\overline{S_{(i+1)}}$ in w_i and another complementation occurred at point ②.



$p(X) = X$ for all other $X \in \Sigma$. $p(X)$ is undefined for $X \in (\Sigma_j - \{\#_j, Z_j\})$.

It follows from the previous description that the given construction performs the cyclic computation of f_i described at the beginning of the proof. Again this scheme follows the rules given at the beginning of this section, which prevents mutual interference of called and calling functions. □

Example 3.11 *Function* $(x) = \lceil \log_2 x \rceil$ *can be computed as* $f(x) = \mu y[\mathrm{Sub}(2^y, x) = 0]$, *which gives more explicitly* $f(x) = M(C(\mathrm{Sub}, C(2^y, U_2^2), U_1^2))(x)$. *To obtain the final scheme* (Σ, p), *the functions* Sub *and* 2^n *must be expanded according to their description in examples 3.7 and 3.8.*

Theorem 3.12 *Let* $f : \mathcal{N}^n \longrightarrow \mathcal{N}$ *be a partial recursive function. Then there exists a Watson-Crick D0L scheme* G *computing* f.

Proof. Follows from Theorem 2.3 and from Lemma 3.2, 3.5, 3.9 and 3.10 due to the fact, that all constructions in these lemmata follow the rules given at the beginning of this section. Hence we can arbitrarily combine base functions and operations to compute any partial recursive function. Notice also that for any scheme (Σ_i, p_i) constructed according to the above proofs and computing a function $f_i(x)$ the number of symbols in Σ_i depends only on i and not on x. Since i depends on description of f_i in terms of base functions and operations from Section 2, Σ_i is always finite for any given f_i. □

Corollary 3.13 *There exist an universal Watson-Crick D0L scheme* G_U *such that for each partial recursive function* $f : \mathcal{N} \longrightarrow \mathcal{N}$ *there is an integer* i_f *such that the scheme* G_U *computes the function* $g(x, i_f) = f(x)$.

Proof. Follows directly from Theorem 2.4 and 3.12. □

4 Conclusions

We studied the properties of Watson-Crick complementation if the framework of deterministic Lindenmayer systems. We obtained an universal computational power only by enhancing the D0L iterated morphism by a complementation mechanism, which is one of the basic DNA operations. This unexpected result may be inspiring for both biological and computational research in molecular genetics and molecular computing. All examples given in this paper (and also many others) were generated and tested by the Lisp software package created at Silesian University in Opava. The package consists of two parts: a Watson-Crick D0L scheme emulator and a tool for easy construction of partial recursive functions by Watson-Crick D0L systems following exactly the given proofs.

5 Acknowledgements

I am grateful to Arto Salomaa for inspiration, useful remarks and discussions, to Erzsebet Csuhaj-Varjú for her invitation to Sztaki, where I found the primary idea and to Alica Kelemenová and Jozef Kelemen for continuous support.

References

[1] J. Dassow, G. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, 1989.

[2] J. Gruska: *Foundations of Computing.* International Thomson Computer Press, London, 1997.

[3] S. Kleene: General recursive functions on natural numbers. *Mathematische Annalen* **112**, 727–742 (1936).

[4] V. Mihalache, A. Salomaa: *Language-Theoretic Aspects of DNA Complementarity.* TUCS (Turku Centre for Computer Science) report 202/1998.

[5] G. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms.* Springer-Verlag, 1998.

[6] G. Rozenberg, A. Salomaa: *The Mathematical Theory of L systems.* Academic Press, New York, 1980.

[7] A. Salomaa, G. Rozenberg (eds.): *Handbook of Formal Languages.* Springer-Verlag, 1997.

[8] C. H. Smith: *A Recursive Introduction to the Theory of Computation.* Springer-Verlag, 1994.