Regular expressions in process algebra

Conference Paper *in* Proceedings - Symposium on Logic in Computer Science · July 2005

DOI: 10.1109/LICS.2005.43 · Source: IEEE Xplore

CITATIONS
12

READS
261

2 authors:

Jos Baeten
Centrum Wiskunde & Informatica
182 PUBLICATIONS **5,655** CITATIONS

SEE PROFILE

Flavio Corradini
University of Camerino
302 PUBLICATIONS **2,438** CITATIONS

SEE PROFILE

Regular Expressions in Process Algebra

J.C.M. Baeten
Division of Computer Science, Technische Universiteit Eindhoven
josb@win.tue.nl

F. Corradini
Department of Mathematics and Computer Science, Università di Camerino
flavio.corradini@unicam.it

Abstract

We tackle an open question of Milner ([10]). We define a set of so-called well-behaved finite automata that, modulo bisimulation equivalence, corresponds exactly to the set of regular expressions.

1 Introduction

In language theory, there is a well-known correspondence between the set of regular expressions and the set of finite (non-deterministic) automata: for each regular expression it is possible to find a finite automaton that admits the same language, and vice versa. But it is also well-known that this correspondence breaks down if we consider notions of equivalence, other than language equivalence. Of particular interest is bisimulation equivalence. Milner proves in [10] that not every finite automaton is bisimulation equivalent to a regular expression, that is, a closed term in the process algebra with atomic actions, successful and unsuccessful termination, choice, sequential composition and iteration. He poses the question how the set of finite behaviours that are bisimulation equivalent to a regular expression can be characterized. This paper tackles this open question. We define a set of so-called *well-behaved* finite behaviours, that correspond exactly to the set of regular expressions. For related work, see [9].

1.1 Acknowledgements

We thank Luca Aceto (Reykjavik University), Wan Fokkink (Vrije Universiteit, Amsterdam), Bas Luttik (Technische Universiteit Eindhoven) and the anonymous referees for their useful remarks and suggestions.

2 Process Algebra

We start out from the process algebra $\mathrm{BPA}^*_{\delta,\epsilon}$. Closed terms in this process algebra correspond exactly to the regular expressions of formal language theory. We use process algebra notation and not regular expression notation to emphasise the fact that we consider bisimulation equivalence as our notion of equivalence, and not language equivalence. $\mathrm{BPA}^*_{\delta,\epsilon}$ extends the basic process algebra BPA (see [5]) with constants δ and ϵ and iteration operator $*$. We assume we have a given set of actions A. This set, usually (but not necessarily) finite, is considered a parameter of the theory. The signature elements are:

- Binary operator $+$ denotes *alternative composition* or choice. Process $x + y$ executes either x or y, but not both. The choice is resolved upon execution of the first action. Notation $+$ is also used for regular expressions.
- Binary operator $\cdot$ denotes *sequential composition*. We choose to have sequential composition as a basic operator, different from CCS (see [11]). As a result, we have a difference between successful termination (ϵ) and unsuccessful termination (δ). As is done for regular expressions, this operator is sometimes not written.
- Constant δ denotes *inaction* (or deadlock), and is the neutral element of alternative composition. Process δ cannot execute any action, and cannot terminate. In language theory, this process corresponds to the constant 0.
- Constant ϵ denotes the *empty process* or *skip*. It is the neutral element of sequential composition. Process ϵ cannot execute any action, but terminates successfully. In language theory, this process corresponds to the constant 1.
- We have a constant a for each $a \in A$, a so-called *atomic action*. Process a executes action a and then

terminates successfully. This coincides with the notation in language theory. The set of actions A is considered a parameter of the theory.

- There is a unary operator $*$ called *iteration* or *Kleene star*. Process x^* can execute x any number of times, but can also terminate successfully. This coincides with the notation in language theory. In [4], a *binary* version of this operator is used. We can use the unary version, common in language theory, as we have a constant ϵ.

The process algebra $\text{BPA}^*_{\delta,\epsilon}$ is axiomatised by axioms A1-9 and KS1-3 in Table 1. Axioms A1-9 are standard. Compared to language theory, we do not have the law $x \cdot (y + z) = x \cdot y + x \cdot z$ (the 'wrong' distributivity, these terms differ in the moment of choice), and the law $x \cdot \delta = \delta$ (that would really characterize δ as 0; in $x \cdot \delta$, actions from x can be executed but no termination can take place). KS1 defines iteration in terms of a recursive equation. Taking $x = \delta$ in KS1 (and using A6, A7) yields $\delta^* = \epsilon$ (as in language theory). KS2 expresses that immediate termination can be omitted in iteration behaviour (in language theory, we say that we can assume that the iterated term does not have the empty word property); taking $x = \delta$ yields $\epsilon^* = \epsilon$. KS3 is the axiom of Troeger ([14]).

Table 1. Axioms of $\text{BPA}^*_{\delta,\epsilon}$.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5
$x + \delta = x$	A6
$\delta \cdot x = \delta$	A7
$\epsilon \cdot x = x$	A8
$x \cdot \epsilon = x$	A9
$x^* = \epsilon + x \cdot x^*$	KS1
$(x + \epsilon)^* = x^*$	KS2
$x^* \cdot (y \cdot (x + y)^* + \epsilon) = (x + y)^*$	KS3

The regular expressions are the *closed* terms over this theory (i.e. the terms without variables). Many results in process algebra, like the following normal form proposition, only hold for the set of closed terms.

Definition 1 We define a set of normal forms inductively:

1. the constants δ, ϵ are normal forms;
2. if t, s are normal forms, and a is an atomic action, then also $a \cdot t$, $t^* \cdot s$ and $t + s$ are normal forms.

Proposition 2 Let t be a closed $\text{BPA}^*_{\delta,\epsilon}$-term. Then there is a normal form s such that $\text{BPA}^*_{\delta,\epsilon} \vdash t = s$.

Proof We can turn the axioms A4,5,7,8,9 of $\text{BPA}^*_{\delta,\epsilon}$ into rewrite rules, by orienting them from left to right. We obtain a confluent and terminating term rewrite system modulo A1-2. Then, reduce t to normal form. The result may still contain summands of the form a (only an atomic action) or a^* (only an iteration). These have to be replaced by $a \cdot \epsilon$ and $a^* \cdot \epsilon$, respectively. This proof is like several examples in [3] or [2]. □

As a consequence of this proposition, each closed term over $\text{BPA}^*_{\delta,\epsilon}$ can be written as δ, ϵ or in the form

$$a_1 \cdot t_1 + \ldots a_n \cdot t_n + u_1^* \cdot v_1 + \ldots + u_m^* \cdot v_m + \{\epsilon\},$$

for certain $n, m \in \mathbb{N}$ with $n + m > 0$, certain $a_i \in A$ and normal forms t_i, u_j, v_j. The ϵ summand may or may not occur.

We will have need to strengthen this result a little bit. For this, we use a result of Milner, proposition 6.2 of [10]:

Proposition 3 [10]: For each closed $\text{BPA}^*_{\delta,\epsilon}$-term t there is a closed $\text{BPA}^*_{\delta,\epsilon}$-term s with

$$\text{BPA}^*_{\delta,\epsilon} \vdash t = s$$

and s has no subterm of the form f^* with $f = f + \epsilon$.

Using this result, we can require in the normal form in addition that terms u_j do not satisfy $u_j = u_j + \epsilon$, i.e. these terms do not have the empty word property. This will ensure that the recursive specifications we define further down are guarded.

Next, we provide a model for $\text{BPA}^*_{\delta,\epsilon}$ on the basis of structured operational rules (so-called *SOS rules*) in the style of Plotkin (see [12]). The rules in Table 2 define the following relations on closed $\text{BPA}^*_{\delta,\epsilon}$-terms: binary relations $. \xrightarrow{a} .$ (for $a \in A$) and a unary relation $\downarrow$. Intuitively, they have the following meaning:

- $x \xrightarrow{a} x'$ means that x evolves into x' by executing atomic action a;
- $x \downarrow$ means that x has the option to terminate successfully (without executing an action)

Thus, the relations concern action execution and termination, respectively; we do not have need of a mixed relation $. \xrightarrow{a} \surd$ as in [3] or [2].

The rules provide a transition system for each closed term. Next, we define an equivalence relation on the resulting transition systems in the standard way.

Table 2. Deduction rules for $\mathrm{BPA}^*_{\delta,\epsilon}$ ($a \in A$).

$$\epsilon\downarrow \qquad x^*\downarrow \qquad a \xrightarrow{a} \epsilon$$

$$\frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$$

$$\frac{x\downarrow}{x+y\downarrow} \qquad \frac{y\downarrow}{x+y\downarrow}$$

$$\frac{x \xrightarrow{a} x'}{x\cdot y \xrightarrow{a} x'\cdot y} \qquad \frac{x\downarrow, y \xrightarrow{a} y'}{x\cdot y \xrightarrow{a} y'} \qquad \frac{x\downarrow, y\downarrow}{x\cdot y\downarrow}$$

$$\frac{x \xrightarrow{a} x'}{x^* \xrightarrow{a} x'\cdot x^*}$$

Definition 4 Let R be a binary *symmetric* relation on closed terms. We say R is a *bisimulation* if the following holds:

- whenever $R(x,y)$ and $x \xrightarrow{a} x'$ then there is a term y' such that $y \xrightarrow{a} y'$ and $R(x',y')$
- whenever $R(x,y)$ and $x\downarrow$ then $y\downarrow$

We say two closed terms t, s are *bisimulation equivalent* or *bisimilar*, notation $t \leftrightarroweq s$ if there is a bisimulation R with $R(s,t)$.

Proposition 5 Bisimulation equivalence is a congruence relation on closed $\mathrm{BPA}^*_{\delta,\epsilon}$-terms.

Proof This is a standard result following from the format of the deduction rules, see e.g. [2]. □

Theorem 6 *The theory $\mathrm{BPA}^*_{\delta,\epsilon}$ is sound for the model of transition systems modulo bisimulation, i.e. for all closed terms t, s we have*

$$\mathrm{BPA}^*_{\delta,\epsilon} \vdash t = s \quad \Longrightarrow \quad t \leftrightarroweq s$$

Proof This is also a standard result. □

Note that the reverse implication in the theorem above, indicating completeness of the axiom system, does not hold. In fact, a finite complete equational axiomatization is not possible, as shown by Sewell ([13]). This impossibility is due to the presence of the δ constant. In the absence of δ, more positive results can be found in [7] and [8] where a complete axiomatization of regular expressions up to bisimulation equivalence is given when the language satisfies the so-called hereditary non-empty word property (essentially requiring that the non-empty word property be satisfied at any depth within a star context).

The first axiom of iteration is a specific instance of a *recursive equation*. It is standard to use recursive equations to specify processes with possible infinite behaviour, see e.g. [3] or [2]. We proceed to define recursion in our setting.

Let V be a set of variables ranging over processes. A *recursive specification* $E = E(V)$ is a set of equations $E = \{X = t_X \mid X \in V\}$ where each t_X is a term over the signature in question (in our case, $\mathrm{BPA}^*_{\delta,\epsilon}$) and variables from V. A *solution* of a recursive specification $E(V)$ in our theory is a set of processes $\{\langle X|E\rangle | X \in V\}$ in some model of the theory such that the equations of $E(V)$ hold, if for all $X \in V$, X stands for $\langle X|E\rangle$. Mostly, we are interested in one particular variable $X \in V$, called the *initial* variable.

Let t be a term containing a variable X. We call an occurrence of X in t *guarded* if this occurrence of X is preceded by an atomic action (i.e., t has a subterm of the form $a \cdot s$, and this X occurs in s).

We call a recursive specification *guarded* if all occurrences of all its variables in the right-hand sides of all its equations are guarded or it can be rewritten to such a recursive specification using the axioms of the theory and the equations of the specification.

We can formulate the following principles:

- RDP (the *Recursive Definition Principle*): each recursive specification has at least one solution;
- RSP (the *Recursive Specification Principle*): each *guarded* recursive specification has at most one solution.

Different models of $\mathrm{BPA}^*_{\delta,\epsilon}$ will satisfy none, one or both of these principles. Let us look at the transition system models in particular.

Consider a recursive specification E. For each variable X of E, we can add a new constant $\langle X|E\rangle$ to our syntax. Table 3 provides deduction rules for these constants. They come down to looking upon $\langle X|E\rangle$ as the process $\langle t_X|E\rangle$, which is t_X with, for all $Y \in V$, all occurrences of Y in t_X replaced by $\langle Y|E\rangle$.

Table 3. Deduction rules for recursion ($a \in A$).

$$\frac{\langle t_X|E\rangle \xrightarrow{a} y}{\langle X|E\rangle \xrightarrow{a} y} \qquad \frac{\langle t_X|E\rangle\downarrow}{\langle X|E\rangle\downarrow}$$

Now if we add such constants $\langle X|E\rangle$ for all specifications E to our syntax, we obtain a model $\mathbb{G}^\infty$ for $\mathrm{BPA}^*_{\delta,\epsilon}$ that satisfies RDP and RSP (see e.g. [3]). If we add constants $\langle X|E\rangle$ only for guarded E, we obtain a model $\mathbb{G}$

satisfying RSP. The model $\mathbb{G}$ is really smaller than $\mathbb{G}^\infty$, since using unguarded recursion we can specify infinitely branching processes, whereas for guarded recursion we can always get a finitely branching solution. Thus, RDP doesn't hold any more on the second model. Note that, due to the presence of the constant ϵ, a difference between $\mathrm{BPA}^*_{\delta,\epsilon}$ and the standard theory BPA is that finite guarded recursion allows the specification of a process with unbounded branching (see [6]).

The third possibility is adding still fewer constants, namely adding only $\langle X|E\rangle$ for so-called *regular* recursive specifications. We call an equation *regular* if it is in one of the two forms

1. $X = \delta + a_1 \cdot X_1 + \cdots + a_n \cdot X_n$, or
2. $X = \epsilon + a_1 \cdot X_1 + \cdots + a_n \cdot X_n$,

for certain $n \in \mathbb{N}, a_i \in A, X_i \in V$. In the first equation, the δ summand is only needed in case $n = 0$. For regular specifications, each variable corresponds directly to a state in the generated transition system. We usually present a regular equation as follows:

$$X = \sum_{1 \leq i \leq n} a_i \cdot X_i + \{\epsilon\},$$

where an empty sum stands for δ and the ϵ summand is optional.

The model $\mathbb{R}$ of $\mathrm{BPA}^*_{\delta,\epsilon}$ is obtained if we add constants $\langle X|E\rangle$ only for finite regular E. $\mathbb{R}$ is the model of *regular* processes, it is equivalent to the model of finite transition systems modulo bisimulation, see e.g. [5]. Again we can establish that $\mathbb{R}$ is really smaller than $\mathbb{G}$, a process in the difference is the counter C defined by the following specification (p standing for plus, m for minus):

$$C = T \cdot C \qquad T = p \cdot S \qquad S = m + T \cdot S.$$

Finally, the term model $\mathbb{P}$ of $\mathrm{BPA}^*_{\delta,\epsilon}$ is even smaller than $\mathbb{R}$. In [4] it is shown that there are regular processes that cannot be defined just using iteration. In the model $\mathbb{P}$, the principle RSP boils down to the following conditional axiom:

$$x = y \cdot x + z \text{ guarded} \quad \Longrightarrow \quad x = y^* \cdot z \qquad \text{RSP}^*.$$

The guardedness of this equation would be expressed in language theory as follows: y does not have the empty word property (that is, y does not have ϵ as a summand).

3 Well-Behaved Specifications

We define a class of recursive specifications over $\mathrm{BPA}^*_{\delta,\epsilon}$ that we will call *well-behaved*. The idea is that the class of well-behaved specifications corresponds exactly to the class of closed $\mathrm{BPA}^*_{\delta,\epsilon}$-terms, modulo bisimilarity.

Consider sequences of natural numbers ranged over by σ, ρ (sometimes with a prime or index) ($\sigma, \rho \in \mathbb{N}^*$). Call a subset S of $\mathbb{N}^*$ *downwards closed* if the empty sequence $\lambda \in S$, and, whenever $\sigma n \in S$, also $\sigma \in S$ and $\sigma k \in S$ for all $k < n$.

Definition 7 A recursive specification E over $\mathrm{BPA}^*_{\delta,\epsilon}$ is *in suitable form* if

1. it is finite and guarded;
2. the set of variables X_σ is indexed by a downwards closed subset of $\mathbb{N}^*$;
3. each equation has the following form:

$$X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + \\ + e_\sigma \cdot X_\sigma + e_\rho \cdot X_\rho + c,$$

   where $m \geq 0$, $e_{\sigma i}, e_\sigma, e_\rho$ are closed $\mathrm{BPA}^*_{\delta,\epsilon}$-terms, $c \in A \cup \{\delta, \epsilon\}$, and ρ is a proper prefix of σ. Note that, as we can take δ for a term e_σ, e_ρ, c, the last three terms may or may not be present. If there is a summand of the form $e_\rho \cdot X_\rho$ present with e_ρ terminating (i.e. the term can terminate successfully after a finite number of steps), we call the variable X_ρ a *cycling* variable;
4. when X_ρ is a cycling variable (it occurs on the right-hand side of an equation of a variable with longer index) then its equation is of the form

$$X_\rho = \epsilon \cdot X_{\rho 0} + \epsilon \cdot X_{\rho 1},$$

   i.e. $m = 2$, $e_{\rho 0} = e_{\rho 1} = \epsilon$ and none of the optional summands is present. The variable $X_{\rho 0}$ represents the cycling part of X_ρ (the part that eventually leads back to X_ρ) and the variable $X_{\rho 1}$ represents the exit part of X_ρ (the part that can lead to termination, or to some cycling variable higher up than X_ρ). Below, we will define well-behavedness of a specification in such a way that a split can be made in this way.

A recursive specification is *in regular suitable form* if it is in suitable form and all the occurring e_σ in equations of non-cycling variables are in A.

Definition 8 Let E be a recursive specification in suitable form over a set of variables $\{X_\sigma : \sigma \in S \subset \mathbb{N}^*\}$. As S is finite, we can define a notion with induction on the depth of the variable tree below X_σ (so, we define this first for the *maximal* sequences $\sigma \in S$).

Let ρ be a prefix of σ. We say X_σ *cycles back to* X_ρ if:

- in case X_σ is cycling (so its equation is of the form $X_\sigma = \epsilon \cdot X_{\sigma 0} + \epsilon \cdot X_{\sigma 1}$), then X_σ cycles back to X_ρ iff $X_{\sigma 0}$ cycles back to X_σ (the cycling part) and $X_{\sigma 1}$ cycles back to X_ρ (the exit part);
- in case X_σ is not cycling, we require that its equation is of the form

$$X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + \\ + e_\sigma \cdot X_\sigma + e_\rho \cdot X_\rho,$$

  so there is no constant term present, and all $X_{\sigma i}$ cycle back to X_ρ.

Next, we define when a variable X_σ is well-behaved, again with induction on the depth of the variable tree below X_σ. We say X_σ is *well-behaved* if:

- in case X_σ is cycling, we say X_σ is well-behaved iff $X_{\sigma 0}$ cycles back to X_σ and $X_{\sigma 1}$ is well-behaved;
- in case X_σ is not cycling, we require that its equation is of the form

$$X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + e_\sigma \cdot X_\sigma + c,$$

  so there is no cycling variable present, and all $X_{\sigma i}$ are well-behaved.

Finally, we call a recursive specification E in suitable form *well-behaved* iff its initial variable X_λ is well-behaved.

Theorem 9 *Every well-behaved recursive specification E has a closed term in $BPA^*_{\delta,\epsilon}$ as a solution (up to bisimulation equivalence).*

In order to prove this theorem, we prove two lemmas. The theorem will follow immediately from the second lemma.

Lemma 10 Let E be a recursive specification in suitable form, and suppose X_σ cycles back to X_ρ. Then there is a closed term e over $BPA^*_{\delta,\epsilon}$ such that $X_\sigma = e \cdot X_\rho$ (here, = denotes derivability in $BPA^*_{\delta,\epsilon}$ + RSP*).

Proof By induction on the depth of the variable tree below X_σ.

In the base case, there are no variables below X_σ, so the equation of X_σ must be either $X_\sigma = e_\rho \cdot X_\rho$ or $X_\sigma = e_\sigma \cdot X_\sigma + e_\rho \cdot X_\rho$ or $X_\sigma = e_\sigma \cdot X_\sigma$. As the specification is guarded, term e_σ does not have the empty word property, i.e. $e_\sigma = e_\sigma + \epsilon$ does not hold. In the first case, we are done immediately, in the second case, it follows from RSP* that $X_\sigma = e^*_\sigma \cdot e_\rho \cdot X_\rho$ and in the third case, we have $X_\sigma = e^*_\sigma \cdot \delta \cdot X_\rho$. RSP* can be applied as the condition on e_σ holds.

In the induction case, there are two subcases.

- if X_σ is cycling, we have $X_\sigma = \epsilon \cdot X_{\sigma 0} + \epsilon \cdot X_{\sigma 1}$ and $X_{\sigma 0}$ cycles back to X_σ and $X_{\sigma 1}$ cycles back to X_ρ. We can apply the induction hypothesis to $X_{\sigma 0}$ and $X_{\sigma 1}$, so there are closed terms f_0, f_1 such that $X_{\sigma 0} = f_0 \cdot X_\sigma$ and $X_{\sigma 1} = f_1 \cdot X_\rho$. Putting this together, we obtain $X_\sigma = \epsilon \cdot X_{\sigma 0} + \epsilon \cdot X_{\sigma 1} = X_{\sigma 0} + X_{\sigma 1} = f_0 \cdot X_\sigma + f_1 \cdot X_\rho = f_0^* \cdot f_1 \cdot X_\rho$. RSP* can be applied since f_0 does not have the empty word property, as a guard must be passed on the path from X_σ back to X_σ.
- if X_σ is not cycling, we have $X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + e_\sigma \cdot X_\sigma + e_\rho \cdot X_\rho$ and all $X_{\sigma i}$ cycle back to X_ρ. By guardedness e_σ does not have the empty word property. By induction hypothesis there are closed terms f_i such that $X_{\sigma i} = f_i \cdot X_\rho$. But then $X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + e_\sigma \cdot X_\sigma + e_\rho \cdot X_\rho = e_{\sigma 0} \cdot f_0 \cdot X_\rho + \ldots + e_{\sigma(m-1)} \cdot f_{m-1} \cdot X_\rho + e_\sigma \cdot X_\sigma + e_\rho \cdot X_\rho = e_\sigma \cdot X_\sigma + (e_{\sigma 0} \cdot f_0 + \ldots e_{\sigma(m-1)} \cdot f_{m-1} + e_\rho) \cdot X_\rho = e^*_\sigma \cdot (e_{\sigma 0} \cdot f_0 + \ldots e_{\sigma(m-1)} \cdot f_{m-1} + e_\rho) \cdot X_\rho$.

□

Lemma 11 Let E be a recursive specification in suitable form, and suppose variable X_σ is well-behaved. Then there is a closed term e over $BPA^*_{\delta,\epsilon}$ such that $X_\sigma = e$ (= is again derivability in $BPA^*_{\delta,\epsilon}$ + RSP*).

Proof By induction on the depth of the variable tree below X_σ.

In the base case, there are no variables below X_σ, so the equation of X_σ must be either $X_\sigma = c$ or $X_\sigma = e_\sigma \cdot X_\sigma + c$ or $X_\sigma = e_\sigma \cdot X_\sigma$. By guardedness, e_σ does not have the empty word property. In the first case, we are done immediately, in the second case, it follows from RSP* that $X_\sigma = e^*_\sigma \cdot c$, and in the third case, $X_\sigma = e^*_\sigma \cdot \delta$.

In the induction case, there are two subcases.

- if X_σ is cycling, then $X_\sigma = \epsilon \cdot X_{\sigma 0} + \epsilon \cdot X_{\sigma 1}$ and $X_{\sigma 0}$ cycles back to X_σ and $X_{\sigma 1}$ is well-behaved. By the definition of cycling back there is a closed term f such that $X_{\sigma 0} = f \cdot X_\sigma$. By induction hypothesis there is a closed term f' such that $X_{\sigma 1} = f'$. Thus $X_\sigma = X_{\sigma 0} + X_{\sigma 1} = f \cdot X_\sigma + f' = f^* \cdot f'$. As f gives the path from X_σ back to X_σ, it follows from guardedness that it cannot have the empty word property.
- if X_σ is not cycling, we have $X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + e_\sigma \cdot X_\sigma + c$ and all $X_{\sigma i}$ are well-behaved. By guardedness e_σ does not have the empty word property. By induction hypothesis this implies that there are closed terms f_i such that $X_{\sigma i} = f_i$. Thus $X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + e_\sigma \cdot X_\sigma + c = e_{\sigma 0} \cdot f_0 + \ldots + e_{\sigma(m-1)} \cdot f_{m-1} + e_\sigma \cdot X_\sigma + c = e^*_\sigma \cdot (e_{\sigma 0} \cdot f_0 + \ldots + e_{\sigma(m-1)} \cdot f_{m-1} + c)$.

□

Note 12 Our notion of cycling was inspired by (but is different from) the notion of ruling in [9]; our notion of well-behaved was inspired by their notion of hierarchical. It should be noted that they work in a different setting, as they use the law $x \cdot \delta = \delta$, which is invalid in the present setting. (Another difference is that the law $x + x = x$ is not valid in their setting, but this is not the crucial difference for the present results.)

On the other hand, the present work also can be applied in their setting. Adding the law $x \cdot \delta = \delta$ amounts to considering the constant δ as *predictable failure* in the words of [1]. In [1], it is proven that using this law, every closed term over BPA_δ is either equal to δ or can be written without δ. This can be extended to $\mathrm{BPA}^*_{\delta,\epsilon}$ (crucial point: $\delta^* = \epsilon$), and using a normal form without δ in the sequel will make all results go through.

4 Regular Expressions

Now we consider the reverse direction, how to transform a given regular expression into a *regular* well-behaved recursive specification. Recall from Section 2 that each closed term over $\mathrm{BPA}^*_{\delta,\epsilon}$ can be written as δ, ϵ or in the form

$$a_1 \cdot t_1 + \ldots a_n \cdot t_n + u_1^* \cdot v_1 + \ldots + u_m^* \cdot v_m + \{\epsilon\},$$

for certain $n, m \in \mathbb{N}$ with $n + m > 0$, certain $a_i \in A$ and normal forms t_i, u_j, v_j. The ϵ summand may or may not occur.

Starting from such a term e in normal form, we describe an algorithm to arrive at a recursive specification. Consider an example, taken from [9]. Take $e = a(a^*b + c) + (c^* + a^*b)^*c^* + a$. This term is in normal form, but the star term $c^* + a^*b$ has the empty word property, as $c^* = c^* + \epsilon$. Therefore, we rewrite this term to $e' = a(a^*b + c) + (cc^* + a^*b)^*c^* + a$. Associate X_λ to e'.

1. $X_\lambda = aX_0 + \epsilon X_1 + a$. Thus, X_0 is associated to $a^*b + c$, X_1 to $(cc^* + a^*b)^*c^*$.
2. $X_0 = \epsilon X_{00} + c$. Thus, X_{00} is associated to a^*b.
3. $X_{00} = X_{000} + X_{001}$. Each star-term is split into two parts: the cycling part, where the loop is executed at least once, and the exit part, where the exit is chosen. Such a term will turn into a cycling variable. Here, X_{000} corresponds to $a \cdot X_{00}$, and X_{001} corresponds to b.
4. $X_{000} = aX_{00}$. Variable X_{000} cycles back to X_{00}.
5. $X_{001} = b$.
6. $X_1 = X_{10} + X_{11}$. Again, a star-term is split into two parts. Here, X_{10} corresponds to $(cc^* + a^*b) \cdot X_1$, and X_{11} corresponds to c^*.
7. $X_{10} = cX_{100} + X_{101}$. Here, X_{100} corresponds to $c^* \cdot X_1$, and X_{101} corresponds to $a^*b \cdot X_1$.
8. $X_{100} = X_{1000} + X_{1001}$. Again, a star term.
9. $X_{1000} = cX_{100}$. Variable X_{1000} cycles back to X_{100}.
10. $X_{1001} = X_1$. Variable X_{1001} cycles back to X_1.
11. $X_{101} = X_{1010} + X_{1011}$. Split of star.
12. $X_{1010} = aX_{101}$. Variable X_{1010} cycles back to X_{101}.
13. $X_{1011} = bX_1$. Variable X_{1011} cycles back to X_1.
14. $X_{11} = X_{110} + X_{111}$. Split of star.
15. $X_{110} = cX_{11}$.
16. $X_{111} = \epsilon$.

Note that the resulting recursive specification is guarded. Moreover, the resulting specification is much more restricted than the general format of well-behaved specifications, in the following way: an equation is required to be in the form

$$X_\sigma = e_{\sigma 0} \cdot X_{\sigma 0} + \ldots + e_{\sigma(m-1)} \cdot X_{\sigma(m-1)} + \\ + e_\sigma \cdot X_\sigma + e_\rho \cdot X_\rho + c.$$

Here, we have that all expressions $e_{\sigma i}, e_\rho$ are constants, and that term $e_\sigma \cdot X_\sigma$ does not occur (stated differently, we can always take $e_\sigma = \delta$).

In general, we define a regular well-behaved recursive specification with solution a given $\mathrm{BPA}^*_{\delta,\epsilon}$-term e in normal form by structural induction on e.

In the base case, if $e \in A \cup \{\delta, \epsilon\}$, we simply define $X_\lambda = e$.

In the induction step, we can write $e = a_0 \cdot t_0 + \ldots a_{n-1} \cdot t_{n-1} + u_0^* \cdot v_0 + \ldots + u_{m-1}^* \cdot v_{m-1} + \{\epsilon\}$ for certain $n, m \in \mathbb{N}$ with $n + m > 0$, certain $a_i \in A$ and simpler terms t_i, u_j, v_j. Thus, we can assume there are regular well-behaved recursive specifications E_i, F_j, G_j with these terms as solutions. We proceed to define a recursive specification for e as follows.

1. $X_\lambda = a_0 \cdot X_0 + \ldots a_{n-1} \cdot X_{n-1} + \epsilon \cdot X_n + \cdots + \epsilon \cdot X_{n+m-1} + \{\epsilon\}$
2. $X_{i\sigma} = t_\sigma^i$ for each equation $X_\sigma = t_\sigma$ in E_i, where t_σ^i is obtained from t_σ by replacing each occurring variable X_ρ by $X_{i\rho}$

3. $X_{n+j} = \epsilon \cdot X_{(n+j)0} + \epsilon \cdot X_{(n+j)1}$ for each $j < m$
4. $X_{(n+j)0\sigma} = t^j_\sigma$ for each equation $X_\sigma = t_\sigma$ in F_j, where t^j_σ is obtained from t_σ by replacing each occurring variable X_ρ by $X_{(n+j)0\rho}$ and replacing each constant term c by $c \cdot X_{n+j}$
5. $X_{(n+j)1\sigma} = t^j_\sigma$ for each equation $X_\sigma = t_\sigma$ in G_j, where t^j_σ is obtained from t_σ by replacing each occurring variable X_ρ by $X_{(n+j)0\rho}$

It is obvious that the resulting recursive specification is regular.

Proposition 13 Let e be a closed $\mathrm{BPA}^*_{\delta,\epsilon}$-term. The regular recursive specification as defined above has solution e and is well-behaved.

Proof In the base case, we have only one variable X_λ, so the results are immediate (the variable is not cycling).

In the induction step, we have $e = a_0 \cdot t_0 + \ldots a_{n-1} \cdot t_{n-1} + u^*_0 \cdot v_0 + \ldots + u^*_{m-1} \cdot v_{m-1} + \{\epsilon\}$ for certain $n, m \in \mathbb{N}$ with $n+m > 0$, certain $a_i \in A$ and simpler terms t_i, u_j, v_j. By induction hypothesis, we can assume there are regular well-behaved recursive specifications E_i, F_j, G_j with these terms as solutions.

1. The equations of specifications E_i are well-behaved. They are copied with i prefixed to each variable index. As a consequence, the variables X_i are well-behaved.
2. Variables X_{n+j} are cycling. Their equations are in the required form.
3. The equations of specifications G_j are well-behaved. They are copied with $(n+j)1$ prefixed to each variable index. As a consequence, the variables $X_{(n+j)1}$are well-behaved.
4. Now we take a look at F_j. As each constant term in F_j in the construction is multiplied by X_{n+j}, there are no constant terms in the resulting specification, and, whenever a variable has a constant term in F_j, the corresponding variable will cycle back to X_{n+j}. Now there may be inner loops in the specification below $X_{(n+j)0}$. In this case we must use the first clause in the definition of cycling back. We can conclude that $X_{(n+j)0}$ cycles back to X_{n+j}.
5. We conclude that variables X_{n+j} are well-behaved.
6. Finally, X_λ is well-behaved, and so the entire specification is well-behaved.

By construction, e is a solution for X_λ. □

Thus, for each closed $\mathrm{BPA}^*_{\delta,\epsilon}$-term we can find a regular well-behaved recursive specification that has this term as a solution. Note that the resulting specifications are more restricted than the general class of well-behaved specifications, for instance we can require regularity.

Looking back, what is still missing is a procedure, that given a finite behaviour, determines whether or not it has a well-behaved specification. Notice that it is enough if we find a bound on the set of well-behaved specifications we need to consider, as bisimulation is decidable on finite behaviours.

In specific cases, it is easy enough to turn a given finite behaviour into well-behaved form. To give an example, consider the guarded recursive specification $\{X = aY, Y = bX + aZ, Z = cX + aY\}$. In this form, it is not a well-behaved recursive specification. It turns into one, by replacing X by aY everywhere on the right-hand side. We get the following well-behaved specification:

$$\begin{aligned}
X_\lambda &= aX_0 \\
X_0 &= X_{00} + X_{01} \\
X_{00} &= bX_{000} + aX_{001} \\
X_{000} &= aX_0 \\
X_{001} &= cX_{0010} + aX_0 \\
X_{0010} &= aX_0 \\
X_{01} &= \delta
\end{aligned}$$

5 Conclusion

We have defined a set of well-behaved recursive specifications that corresponds exactly to the set of regular expressions, using bisimulation as the notion of equivalence. The same result holds if we restrict to the set of well-behaved regular recursive specifications, that has a direct interpretation as a set of finite transition systems. Thus, we can say that we have defined a structural property, that characterizes the set of finite automata that are expressible by a regular expression (modulo bisimulation). This means we have solved the open question of Milner ([10]).

On the other hand, it can be argued that we have not solved this open question, as, given a finite transition system, we have not presented an algorithm to determine whether or not this transition system is well-behaved. Note that Bosscher does give such an algorithm, as long as the constant δ is not present (see [6]).

Our results can be adapted to the setting of [9], where the constant δ really acts as the zero process.

References

[1] J. Baeten and J. Bergstra. Process algebra with a zero object. In J. Baeten and J. Klop, editors, *Proceedings CONCUR'90*, number 458 in Lecture Notes in Computer Science, pages 83–98. Springer Verlag, 1990.

[2] J. Baeten and C. Verhoef. Concrete process algebra. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 149–269. Oxford University Press, 1995.

[3] J. Baeten and W. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[4] J. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[5] J. Bergstra and J. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings 11th ICALP*, number 172 in LNCS, pages 82–95. Springer Verlag, 1984.

[6] D. Bosscher. *Grammars Modulo Bisimulation*. PhD thesis, University of Amsterdam, 1997.

[7] F. Corradini. A step forward towards equational axiomatizations of Milner bisimulation in Kleene star. In *Proceedings FICS 2000*, 2000.

[8] F. Corradini, R. De Nicola, and A. Labella. An equational axiomatization of bisimulation over regular expressions. *Journal of Logic and Computation*, 12:301–320, 2002.

[9] R. De Nicola and A. Labella. Nondeterministic regular expressions as solutions of equational systems. *Theoretical Computer Science*, 203:179–189, 2003.

[10] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Comput. System Sci.*, 28(3):439–466, 1984.

[11] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[12] G. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60:17–139, 2004. Reprint from 1981 in Special Issue on Structural Operational Semantics.

[13] P. Sewell. Nonaxiomatisability of equivalences over finite state processes. *Annals of Pure and Applied Logic*, 90:163–191, 1997.

[14] D. Troeger. Step bisimulation is pomset equivalence on a parallel language without explicit internal choice. *Math. Struct. Comput. Sci.*, 3(1):25–62, 1993.