Bounded bags, stacks and queues

Bounded Stacks, Bags and Queues

J.C.M. Baeten[1] and J.A. Bergstra[2,3]

[1] Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands,
josb@win.tue.nl, http://www.win.tue.nl/cs/fm/josb/

[2] Programming Research Group, University of Amsterdam,
Kruislaan 403, NL-1098 SJ Amsterdam, The Netherlands

[3] Department of Philosophy, Utrecht University,
Heidelberglaan 8, NL-3584 CS Utrecht, The Netherlands,
janb@phil.ruu.nl, http://www.wins.uva.nl/research/prog/people/janb/

Abstract. We prove that a bounded stack can be specified in process algebra with just the operators alternative and sequential composition and iteration. The bounded bag cannot be specified with these operators, but can be specified if we add the parallel composition operator without communication (free merge). The bounded queue cannot even be specified in this signature; we need a form of variable binding such as given by general communication and encapsulation, the state operator, or abstraction.

1 Introduction

We investigate the definability of bounded bags, stacks and queues in process algebras with iteration. In particular, it is shown that a bounded stack can be specified in process algebra with just the operators alternative and sequential composition and iteration. The bounded bag cannot be specified with these operators, but can be specified if we add the parallel composition operator without communication (free merge). The bounded queue cannot even be specified in this signature; we need a form of variable binding such as given by general communication and encapsulation, the state operator, or abstraction.

This situation is remarkably similar to the situation we have with the specification of the unbounded versions with recursion instead of iteration. There, we have that the unbounded stack is not a regular process, but is finitely definable using alternative and sequential composition and recursion (see [4]). The unbounded bag cannot be finitely specified using alternative, sequential composition and recursion, but can be so specified if we use the free merge in addition (see [3]). The unbounded queue cannot be finitely specified using the signature including free merge (see [5]), but can be specified if we add an operator that features some form of variable binding, such as general communication with encapsulation, renaming or abstraction (see [1]).

From [2] we know that in process algebra with alternative, sequential and parallel composition (with communication) and iteration, not all regular processes can be defined. That paper also shows that we gain expressivity, each time we add one of these operators. The present paper shows that some well-known processes, namely bounded buffers, can serve to show the difference in expressivity.

2 Process Algebra with Iteration

The simplest process algebra is BPA, Basic Process Algebra. Its signature just contains a number of constants, called *atomic actions*, and two binary operators, $+$ which is *alternative composition* and $\cdot$ which is *sequential composition*. The axioms of BPA are the five axioms A1-5 in Table 1 below.

We extend BPA to BPA$_\delta$ by adding the constant δ denoting *inaction* with axioms A6,7 in Table 1. We extend BPA to PA by adding the parallel composition operator $\|$, *free merge* (merge without communication) with axioms M1-4. These axioms make use of the auxiliary operator $⫿\!\!\lfloor$, *left merge*. The combination of BPA$_\delta$ and PA is PA$_\delta$. A denotes the set of atomic actions, $A_\delta = A \cup \{\delta\}$.

$x + y = y + x$	A1	$x \parallel y = x ⌊\!⌊ y + y ⌊\!⌊ x$	M1
$(x + y) + z = x + (y + z)$	A2	$a ⌊\!⌊ x = a \cdot x$	M2
$x + x = x$	A3	$a \cdot x ⌊\!⌊ y = a \cdot (x \parallel y)$	M3
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$(x + y) ⌊\!⌊ z = x ⌊\!⌊ z + y ⌊\!⌊ z$	M4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5		
$x + \delta = x$	A6	$x^* y = x \cdot (x^* y) + y$	BKS
$\delta \cdot x = \delta$	A7		

Table 1. Axioms of PA$^*_\delta$ ($a \in A_\delta$).

Finally, we have in Table 1 the iteration operator * or *binary Kleene star* with defining equation BKS. In x^*y, we can iterate x and terminate by executing y. Much more about iteration can be found in [2]. Iteration gives a limited form of recursion, since p^*q is the solution of the recursive equation

$$X = p \cdot X + q.$$

BPA$^*_\delta$ is BPA$_\delta$ plus iteration.

Below we will occasionally use recursive equations. They will always be *linear*, i.e. of the form

$$X = \sum_{i=1}^{n} a_i \cdot X_i + \sum_{j=1}^{m} b_j$$

for variables X, X_i and $a_i, b_j \in A_\delta$. The key assumption we will need concerning linear recursive equations is that they have a unique solution.

We can use the axioms above and the assumption about unique solutions in order to prove that two process expressions denote the same process. Conversely, we will also need a way to tell when two process expressions cannot denote the same process. Certainly, two processes that are equal must be able to perform the same sequences of actions (must have the same *traces*. Even more, any state of one process must have a corresponding similar state in the other process. This equality is captured by the well-known notion of *bisimulation* (see [6]).

First, we describe which actions a process expression can perform. We do this by defining an operational semantics for process expressions. This semantics is given by means of Plotkin style *action rules* (see [7]).

For each atomic action a we have two predicates on process expressions: a binary relation $\xrightarrow{a}$ and a unary relation $\xrightarrow{a} \surd$. Intuitively, they have the following meaning:

- $p \xrightarrow{a} q$ means that p can perform an a-step and evolve into q
- $p \xrightarrow{a} \surd$ means that p can perform an a-step and terminate successfully

The action rules defining these predicates by structural induction are given in Table 2 (x, y range over process expressions). In the following sections, we will use this operational semantics in a rather informal way: when we say that process expression p can do an a-step to process expression q, we mean $p \xrightarrow{a} q$.

$$a \xrightarrow{a} \surd$$

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

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

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y, y \parallel x \xrightarrow{a} y \parallel x', x \mathbin{\lfloor\!\lfloor} y \xrightarrow{a} x' \parallel y} \qquad \frac{x \xrightarrow{a} \surd}{x \parallel y \xrightarrow{a} y, y \parallel x \xrightarrow{a} y, x \mathbin{\lfloor\!\lfloor} y \xrightarrow{a} y}$$

$$\frac{x \xrightarrow{a} x'}{x^* y \xrightarrow{a} x' \cdot (x^* y)} \qquad \frac{x \xrightarrow{a} \surd}{x^* y \xrightarrow{a} x^* y}$$

$$\frac{y \xrightarrow{a} y'}{x^* y \xrightarrow{a} y'} \qquad \frac{y \xrightarrow{a} \surd}{x^* y \xrightarrow{a} \surd}$$

Table 2. Operational rules for PA^*_δ ($a \in A$).

On the basis of these action rules, we define the notion of bisimulation: we say a symmetric binary relation R on process expressions is a *bisimulation* iff the following holds for all process expressions p, p', q and all actions $a \in A$:

- if $p \xrightarrow{a} p'$ and $R(p, q)$, then there is a process expression q' such that $q \xrightarrow{a} q'$ and $R(p', q')$
- if $p \xrightarrow{a} \surd$ and $R(p, q)$, then $q \xrightarrow{a} \surd$

Then, we say that process expressions p and q are bisimilar, $p \leftrightarrows q$, iff there exists a bisimulation R with $R(p, q)$. From [2] we know that bisimulation is a congruence relation on process expressions, and that the set of process expressions modulo bisimulation constitutes a model for PA^*_δ.

3 Stack

We first give a system of linear equations for the stack. We have given a finite data type D. We use bounded sequences over D to parametrize the process variables. We use the following notations for such sequences:

- $[\,]$ denotes the empty sequence
- $[d]$ denotes the singleton sequence, for each $d \in D$
- $\frown$ denotes concatenation of sequences
- $|\sigma|$ denotes the length of sequence σ

The n-bounded stack ($n \geq 1$) has a specification with variables $S^n(\sigma)$, for each sequence σ with $|\sigma| \leq n$. The input of an element d is denoted by action $r(d)$ (*read* d), the output of d by $s(d)$ (*send* d). An alternative composition with one summand for each element of D is abbreviated by sum notation.

$$S^n([\,]) = \sum_{d \in D} r(d) \cdot S^n([d])$$

$$S^n([d]^\frown\sigma) = s(d) \cdot S^n(\sigma) + \sum_{e \in D} r(e) \cdot S^n([e]^\frown[d]^\frown\sigma)$$

(for each $d \in D$ and sequence σ, if $|\sigma| < n - 1$)

$$S^n([d]^\frown\sigma) = s(d) \cdot S^n(\sigma)$$

(for each $d \in D$ and sequence σ, if $|\sigma| = n - 1$).

It can be noted that this case distinction can be avoided, if we specify operators $head, tail$ on sequences, and use a conditional operator that allows one or both summands depending on the length of the sequence. We refrain from doing this here, since we do not want to deal with the relationship between error handling in data type specifications and process algebra.

Theorem 1 *The bounded stack can be specified in* BPA^*_δ.

Proof Consider the following specification.

$$Stack(n) =$$

$$\left(\sum_{d_1 \in D} r(d_1) \cdot \left(\sum_{d_2 \in D} r(d_2) \cdot \left(\ldots \left(\sum_{d_n \in D} r(d_n) \cdot s(d_n)\right)^* \ldots\right)^* s(d_2)\right)^* s(d_1)\right)^* \delta$$

Inductively, we can define these processes as follows.

$$Elt(1) = \sum_{d \in D} r(d) \cdot s(d)$$

$$Elt(n+1) = \sum_{e \in D} r(e) \cdot Elt(n)^* s(e)$$

$$Stack(n) = Elt(n)^* \delta.$$

Now we have to prove that $Stack(n) = S^n([\,])$ holds for each n. In order to do this, we have to provide an expression for each state of the stack in terms of the variables $Stack(n), Elt(n)$. We define these expressions $T^n(\sigma)$ inductively:

$$T^n([\,]) = Stack(n)$$

$$T^n([d]^\frown\sigma) = (Elt(n-k-1)^* s(d)) \cdot T^n(\sigma)$$

(for each $d \in D$ and sequence σ with $|\sigma| = k < n-1$)

$$T^n([d]^\frown\sigma) = s(d) \cdot T^n(\sigma)$$

(for each $d \in D$ and sequence σ with $|\sigma| = n-1$).

Now it is straightforward to show that the set of variables $T^n(\sigma)$ form a solution for the linear equations $S^n(\sigma)$. By uniqueness of solutions, we obtain $T^n(\sigma) = S^n(\sigma)$, so in particular $Stack(n) = S^n([\,])$. □

4 Bag

We proceed to give a system of linear equations for the bag. We now use bounded bags or multi-sets over data set D to parametrize the process variables. We use the following notations for such bags:

- $\emptyset$ denotes the empty bag
- $\{d\}$ denotes the singleton bag, for each $d \in D$
- $\cup$ denotes bag union, $-$ bag difference
- $|\beta|$ denotes the size of bag β

The n-bounded bag ($n \geq 1$) has a specification with variables $B^n(\beta)$, for each multi-set β with $|\beta| \leq n$. As before, the input of d is denoted $r(d)$, the output of d by $s(d)$.

$$B^n(\emptyset) = \sum_{d \in D} r(d) \cdot B^n(\{d\})$$

$$B^n(\beta) = \sum_{d \in \beta} s(d) \cdot B^n(\beta - \{d\}) + \sum_{e \in D} r(e) \cdot B^n(\{e\} \cup \beta)$$

(for each multi-set β with $0 < |\beta| < n$)

$$B^n(\beta) = \sum_{d \in \beta} s(d) \cdot B^n(\beta - \{d\})$$

(for each multi-set β with $|\beta| = n$)

Theorem 2 *The bounded bag can be specified in PA^*_δ.*

Proof Consider the following specification.

$$Bag(n) = \mathop{\Big\|}_{i=1}^{n} \left(\sum_{d \in D} r(d) \cdot s(d) \right)^* \delta$$

Or, defined by induction:

$$Bag(1) = Elt(1)^* \delta$$

$$Bag(n+1) = Bag(n) \parallel Bag(1)$$

Now we have to prove that $Bag(n) = B^n(\emptyset)$ holds for each n. In order to do this, we will provide an expression for each state of the bag in terms of the variables $Bag(n)$. We define these expressions $C^n(\beta)$ as follows:

$$C^n(\emptyset) = Bag(n)$$

$$C^n(\beta) = Bag(n-k) \parallel \mathop{\Big\|}_{d \in \beta} (s(d) \cdot Bag(1))$$

(for each multi-set β with $0 < |\beta| = k < n$)

$$C^n(\beta) = \mathop{\Big\|}_{d \in \beta} (s(d) \cdot Bag(1))$$

(for each multi-set β with $|\beta| = n$).

Now it is straightforward to show that the set of variables $C^n(\beta)$ form a solution for the linear equations $B^n(\beta)$. By uniqueness of solutions, we obtain $C^n(\beta) = B^n(\beta)$, so in particular $Bag(n) = B^n(\emptyset)$. □

Next, we want to prove the following theorem.

Theorem 3 *$Bag(n)$ cannot be specified in BPA^*_δ, if $n \geq 2$ and $|D| \geq 2$.*

The proof of this theorem is quite involved. We first make a number of definitions and prove some propositions.

We consider the set of finite transition systems over the set of atomic actions A that contain no deadlock nodes. Let G be such a transition system, then $|G|$ denotes the set of states of G. G has a root $r \in |G|$ and a transition relation $\rightarrow \subseteq |G| \times A \times |G|$. The domain of $\rightarrow$ is $dom(\rightarrow) = \{s \in |G| | \exists t \in |G| \exists a \in A s \xrightarrow{a} t\}$, the codomain is $codom(\rightarrow) = \{t \in |G| \mid \exists s \in |G| \; \exists a \in A \; s \xrightarrow{a} t\}$.

For $p \in dom(\rightarrow)$, $G(p)$ is the process (modulo bisimulation) represented by the graph G with p serving as the root. We notice that states in G are not labeled.

Example 4 Let $|G| = \{0, 1, 2, 3, 4\}, A = \{a, b, c, d\}, \rightarrow = \{0 \xrightarrow{a} 1, 1 \xrightarrow{b} 2, 2 \xrightarrow{c} 1, 1 \xrightarrow{d} 3, 3 \xrightarrow{a} 4\}$ with root 0, then $G(0) = a \cdot ((b \cdot c)^*(d \cdot a)), G(1) = (b \cdot c)^*(d \cdot a), G(2) = c \cdot G(1), G(3) = a$. Note $4 \notin dom(\rightarrow)$.

We will use the following definitions, where X, Y range over processes and p, q range over $dom(\rightarrow)$.

- $X \sqsubseteq Y$ iff $X + Y = Y$
- $I(X) \subseteq A$ is the set of initial actions of X
- $X \in K_G(p, q)$ iff $X \cdot G(q) \sqsubseteq G(p)$
- $X \in K^+_G(p, q)$ iff $X \cdot G(q) = G(p)$
- $X \in K^\infty_G(p, q)$ iff $X \in K_G(p, q)$ and X has an infinite trace
- $X \in K^f_G(p, q)$ iff $X \in K_G(p, q)$ and X has only finite traces
- $K^{+,f}_G(p, q) = K^+_G(p, q) \cap K^f_G(p, q)$, and similarly for other double superscripts

We notice that $\delta \in K_G(p, q)$ but no $X \in K_G(p, q)$ has a proper state equal to δ and $\delta \notin K^+_G(p, q)$.

Further, we call G *deterministic* if $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ imply $q = q'$, and we call G *invertible* if $p \xrightarrow{a} q$ and $p' \xrightarrow{a} q$ imply $p = p'$. Next, we call G *fully abstract* if $G(p) = G(q)$ implies that $p = q$. We call G *non-stuttering* if for no state p and action a we have $p \xrightarrow{a} p$.

We extend $\rightarrow$ to $\rightarrow^* \subseteq |G| \times A^* \times |G|$ in the usual way. If G is deterministic and $s \xrightarrow{\sigma}{}^* t$ we will also denote t with $\sigma(s)$. Similarly, if G is invertible we will denote s with $\sigma^{-1}(t)$. We write $\sigma(p) \downarrow$ if $\sigma(p)$ is defined, $\sigma(p) \uparrow$ if $\sigma(p)$ is not defined.

Lemma 5 (Representation Lemma) Let G be deterministic, then for $p \in dom(\rightarrow)$ we have

$$G(p) = \sum_{a \in A, a(p)\downarrow, a(p) \in dom(\rightarrow)} a \cdot G(a(p)) + \sum_{a \in A, a(p)\downarrow, a(p) \notin dom(\rightarrow)} a$$

Proof The proof follows straightforwardly from the definitions. $\square$

Next, we prove a series of propositions about the sets $K_G(p,q)$.

Proposition 6 Let G be deterministic and fully abstract. If $X \in K_G(p,q)$ and $X \xrightarrow{\sigma}^* \surd$ then $\sigma(p) = q$.

Proof $X \cdot G(q) \xrightarrow{\sigma}^* G(q)$ and $G(p) \xrightarrow{\sigma}^* G(\sigma(p))$, so by determinism $G(q) = G(\sigma(p))$ whence using full abstraction $q = \sigma(p)$. $\square$

Proposition 7 If $X + Y \in K_G^\infty(p,q)$ then $X, Y \in K_G(p,q)$ and $X \in K_G^\infty(p,q)$ or $Y \in K_G^\infty(p,q)$.

Proof If $X + Y \in K_G^\infty(p,q)$ then $(X+Y)\cdot G(q) = X \cdot G(q) + Y \cdot G(q) \sqsubseteq G(p)$. Thus $X \cdot G(q) \sqsubseteq G(p)$ and $Y \cdot G(q) \sqsubseteq G(p)$, so $X, Y \in K_G(p,q)$. Further, if $X + Y$ has an infinite trace then either X or Y has an infinite trace. $\square$

Proposition 8 Let G be deterministic. Let $X \in K_G(p,q), |\sigma| > 0$. If $X \xrightarrow{\sigma}^* Y$ then $Y \in K_G^+(\sigma(p),q)$.

Proof We use induction on the length of σ. If $|\sigma| = 1$, let $\sigma = a$. Using the representation lemma for p and $X \cdot G(q) \sqsubseteq G(p)$, we find $a \cdot Y \cdot G(q) = a \cdot G(a(p))$ and $Y \cdot G(q) = G(a(p))$ which yield $Y \in K_G^+(a(p),q)$.

If $|\sigma| = n+1$, put $\sigma = a \cdot \tau$. We have $X \xrightarrow{a} Y \xrightarrow{\tau}^* Z$. Using the same argument as above we find $Y \in K_G^+(a(p),q)$, so certainly $Y \in K_G(a(p),q)$. Then using the induction hypothesis: $Z \in K_G(\sigma(a(p)),q) = K_G(a\sigma(p),q)$. $\square$

Proposition 9 Let $X \cdot Y \in K_G^\infty(p,q)$. If X does not terminate then $X \in K_G^\infty(p,q)$.

Proof As X does not terminate, $X = X \cdot Y$. $\square$

Proposition 10 Let G be deterministic and let $X \cdot Y \in K_G(p,q)$. If $X \xrightarrow{\sigma}^* \surd$ then $X \in K_G(p,\sigma(p))$ and $Y \in K_G(\sigma(p),q)$. If moreover $X \cdot Y$ has an infinite trace, then at least one of X, Y has an infinite trace.

Proof If $X \xrightarrow{\sigma}^* \surd$ then $X \cdot Y \xrightarrow{\sigma}^* Y$. Using Proposition 8, we find $Y \in K_G^+(\sigma(p),q)$, so certainly $Y \in K_G(\sigma(p),q)$. Moreover, it follows that $Y \cdot G(q) = G(\sigma(p))$. Since $X \cdot Y \cdot G(q) \sqsubseteq G(p)$, we get $X \cdot G(\sigma(p)) \sqsubseteq G(p)$. This means $X \in K_G(p,\sigma(p))$. The last remark is immediate. $\square$

Proposition 11 Let G be deterministic, invertible and fully abstract. If $X^*Y \in K_G(p,q)$, then $X \in K_G(p,p)$ and $Y \in K_G(p,q)$.

Proof $Y \cdot G(q) \sqsubseteq (X \cdot (X^*Y) + Y) \cdot G(q) = (X^*Y) \cdot G(q) \sqsubseteq G(p)$, so $Y \in K_G(p,q)$.

For X, we distinguish two cases. If X does not terminate, then $X \cdot G(p) = X = X \cdot G(q) \sqsubseteq (X+Y) \cdot G(q) = (X^*Y) \cdot G(q) \sqsubseteq G(p)$, so $X \in K_G(p,p)$.

Otherwise, there is a trace σ with $X \xrightarrow{\sigma}{}^* \surd$. This implies $X^*Y \xrightarrow{\sigma} X^*Y$. It follows by Proposition 8 that $X^*Y \in K_G^+(\sigma(p),q)$. Applying the same argument once more we obtain $X^*Y \in K_G^+(\sigma(\sigma(p)),q)$. This means $G(\sigma(p)) = (X^*Y) \cdot G(q) = G(\sigma(\sigma(p)))$. Using full abstraction $\sigma(p) = \sigma(\sigma(p))$. Using invertibility $p = \sigma(p)$ and so $X^*Y \in K_G^+(p,q)$. This means $(X^*Y) \cdot G(q) = G(p)$.

Now $X \cdot G(p) = X \cdot (X^*Y) \cdot G(q) \sqsubseteq (X^*Y) \cdot G(q) \sqsubseteq G(p)$, which means $X \in K_G(p,p)$. $\square$

Proposition 12 Let G be deterministic and fully abstract. If there is an infinite path in G from p that avoids q then $K_G^{+,f}(p,q) = \emptyset$.

Proof Suppose $X \in K_G^+(p,q)$. Let $p \xrightarrow{a_0} p_1 \xrightarrow{a_1} p_2 \dots$ be an infinite path avoiding q. Thus $\sigma = a_0a_1a_2\dots$ is a trace of $G(p) = X \cdot G(q)$. Since $G(p)$ does not deadlock, X does not deadlock either. Let τ be a finite initial segment of σ.

Suppose that $X \xrightarrow{\tau} \surd$, then by Proposition 6 $q = \tau(p)$ which contradicts the assumption on σ. We see that after no finite initial trace of σ X terminates, whence it has an infinite trace. Hence $K_G^{+,f}(p,q) = \emptyset$. $\square$

Proposition 13 Let G be deterministic, non-stuttering and fully abstract. Let $X \in K_G^f(p,p)$. If $a \in I(X)$, then it is not the case that $X \xrightarrow{a} \surd$. Whenever $X \xrightarrow{\sigma}{}^* Y$, we have $Y \in K_G^{+,f}(\sigma(p),p)$.

Proof Suppose $X \xrightarrow{a} \surd$. Using $X \cdot G(p) \sqsubseteq G(p)$ and determinism, we find $G(p) = G(a(p))$. Full abstraction yields $p = a(p)$ which contradicts the non-stuttering property.

Next, let $X \xrightarrow{\sigma}{}^* Y$ then $Y \in K_G^+(\sigma(p),p)$ (by Proposition 8). As an infinite trace for Y implies an infinite trace for X, we obtain finiteness as well. $\square$

Now we need one more definition for the last, highly technical proposition. Let $W_G(p,a,b)$ be the property of graph G, state p and actions a,b that holds if for each state q there is an infinite path in G from p which avoids q and that either starts with a step a or with a step b.

Proposition 14 Let $X \in K_G(p,q)$, suppose $a,b \in I(X)$ such that $W_G(p,a,b)$ holds. Then $X \in K_G^\infty(p,q)$.

Proof Let σ be an infinite path starting from p avoiding q. Assume, without loss of generality, that σ starts with a. Arguing as in Proposition 12, we find $X \in K_G^\infty(p,q)$. However, instead of $X \in K_G^+(p,q)$, we now use $a \in I(X)$ to see that X allows the initial a step. $\square$

Now we have collected all ingredients necessary to start the proof for the bag.

Theorem 3 *$Bag(n)$ cannot be specified in BPA^*_δ, if $n \geq 2$ and $|D| \geq 2$.*

Proof Take $n = 2$ and $|D| = 2$, say $D = \{0, 1\}$. The argument for larger n, D is not more complicated. Let G be the graph of $Bag(2)$. The states are multi-sets of data elements of size 0,1 or 2, denoting the contents of the bag, so $\emptyset, 0, 1, 00, 01$ and 11. The graph of $Bag(2)$ is shown in Fig. 1.

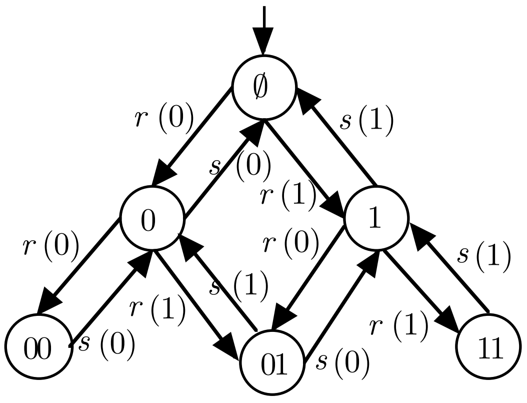

Fig. 1. The graph of $Bag(2)$.

Notice that G is deterministic, invertible and fully abstract. We see $B^2(\emptyset) \in K^\infty_G(\emptyset, \emptyset)$.

We will prove that for all states p, q, no element of $K^\infty_G(p, q)$ can be defined by an expression over BPA^*_δ. Since $B^2(\emptyset) \in K^\infty_G(\emptyset, \emptyset)$, this suffices to prove the theorem. Suppose, for a contradiction, that P is a minimal expression over BPA^*_δ defining some element of a $K^\infty_G(p, q)$ for certain states p, q. We use a case distinction on the form of P.

- P is an atomic action, then P has no infinite trace. This is a contradiction.
- P is a sum, so $P = Q + R$. Since $P \in K^\infty(p, q)$, at least one of Q and R is in $K^\infty(p, q)$ by Proposition 7. This contradicts the minimality of P.
- P is a product, so $P = Q \cdot R$. Using Proposition 9, either $X \in K^\infty_G(p, q)$ of for some r we have $R \in K^\infty_G(r, q)$, again contradicting the minimality of P.
- Otherwise, P is an iteration, so $P = Q^*R$. By Proposition 11, $Q \in K_G(p, p)$, $R \in K_G(p, q)$. From minimality of P we conclude $Q \in K^f_G(p, p)$.
  We claim now that either $p = 0$ and $Q = r(0) \cdot s(0)$, or $p = 1$ and $Q = r(1) \cdot s(1)$. In order to prove this claim, we first make the following observation.

The only states m, n of the bag for which $K_G^{+,f}(m,n) \neq \emptyset$ are either $m = 00, n = 0$ (e.g., process $s(0) \in K_G^{+,f}(m,n)$) or $m = 11, n = 1$ (e.g., process $s(1) \in K_G^{+,f}(m,n)$). For, for all other pairs m, n there is, by inspection of the graph, an infinite path from m avoiding n, and by Proposition 12 this gives $K_G^{+,f}(m,n) = \emptyset$.

Now, to prove the claim, an initial action a of $Q \in K_G^f(p,p)$ cannot lead to termination, by Proposition 13. So action a leads to $Y \in K_G^{+,f}(a(p),p)$. By the previous observation, we must have either $a(p) = 00, p = 0$ and $a = r(0)$, or $a(p) = 11, p = 1$ and $a = r(1)$. By inspection of the graph, we see that in the one case Y must start with $s(0)$ and in the other case with $s(1)$.

Consider the first case and let $Y \in K_G^f(0,0)$. Due to Proposition 8, if σ is a non-terminating trace of Y (say $Y \xrightarrow{\sigma}{}^* Z$), we have that $Z \in K_G^{+,f}(\sigma(0),0)$, and hence by the observation above $\sigma(0) = 00$. Now look at a trace τ of Y. For a trace of length 1 this is fine. The first action is $r(0)$ and it leads to state 00. As 00 differs from 0, $r(0)$ is not a terminating trace for Y. Now consider a larger trace. Inspection of the graph G shows that the second action in a trace of Y must be $s(0)$. After this second step, termination is necessary. Otherwise (with $\sigma = r(0)s(0)$), $\sigma(0) = 00$ must hold which is false. So $r(0)s(0)$ is the one and only completed trace of the finite process Y. It follows that $Y = r(0) \cdot s(0)$.

In the following, we again concentrate on the first case. We have $(r(0) \cdot s(0))^* R \in K_G^\infty(0,q)$. As $(r(0) \cdot s(0))^* R \xrightarrow{r(0)s(0)}{}^* (r(0) \cdot s(0))^* R$, by Proposition 8 in fact $(r(0) \cdot s(0))^* R \in K_G^{+,\infty}(0,q)$. This means $r(1), s(0) \in I(R)$ since both $r(1)$ and $s(0)$ are in $I(G(0))$. Inspection of G (slightly cumbersome) yields that $W_G(0, r(1), s(0))$ holds. Thus by Proposition 14 $R \in K_G^\infty(0,q)$ which contradicts the minimality of P.

□

5 Queue

We give a system of linear equations for the (first in first out) queue. We use the notation for sequences we also used in the case of the stack.

The n-bounded queue ($n \geq 1$) has a specification with variables $Q^n(\sigma)$, for each sequence σ with $|\sigma| \leq n$.

$$Q^n([\,]) = \sum_{d \in D} r(d) \cdot Q^n([d])$$

$$Q^n(\sigma^\frown[d]) = s(d) \cdot Q^n(\sigma) + \sum_{e \in D} r(e) \cdot Q^n([e]^\frown\sigma^\frown[d])$$

(for each $d \in D$ and sequence σ, if $|\sigma| < n - 1$)

$$Q^n(\sigma^\frown[d]) = s(d) \cdot Q^n(\sigma)$$

(for each $d \in D$ and sequence σ, if $|\sigma| = n - 1$).

Theorem 15 *$Q^n(\emptyset)$ cannot be specified in PA^*_δ, if $n \geq 2$ and $|D| \geq 2$.*

Proof The proof is similar to the proof for the bag. Again take $n = 2$ and $D = \{0, 1\}$. Let G be the graph of $Queue(2)$. The states are sequences of data elements of size 0,1 or 2, denoting the contents of the queue. See Fig. 2.

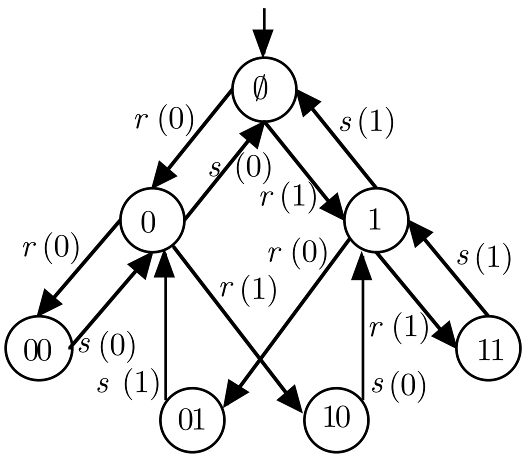

Fig. 2. The graph of $Queue(2)$.

Notice that G is deterministic, invertible and fully abstract. We use the same induction as in the case of the bag. The cases for atomic action, sum and product go the same as for the bag. We have two cases left.

- Let $P \in K^\infty_G(p, q)$ be of the form $Q \cdot R$. We find that $Q \in K_G(p, p), R \in K_G(p, q)$. We may assume $Q \in K^+_G(p, p)$. As in the case of the bag, we can show that either $p = 0$ and $Q = r(0) \cdot s(0)$ or $p = 1$ and $Q = r(1) \cdot s(1)$. The proof of the claim involves more case distinction than in the case of the bag, but is straightforward. The proof is finished in a similar way.
- The remaining case is $P = Q \parallel R$. Since P has an infinite trace, at least one of Q, R has an infinite trace. Without loss of generality, assume this is Q. Suppose R has a terminating trace, $R \overset{\sigma}{\to}^* \surd$. Then $Q \parallel R \overset{\sigma}{\to}^* Q$ and so $Q \in K^+_G(\sigma(p), q)$. Since Q has an infinite trace, this contradicts the minimality of P. It follows that R has no finite terminating traces. If R deadlocks, say $R \overset{\sigma}{\to}^* \delta$, then $Q \parallel R \overset{\sigma}{\to}^* Q \parallel \delta = Q \cdot \delta$. Then $Q \cdot \delta \in K^+_G(\sigma(p), q)$,

so $Q \cdot \delta = G(\sigma(p))$. As $G(\sigma(p))$ cannot deadlock, Q cannot terminate, so $Q \cdot \delta = Q$ and $Q \in K^{+}_{G}(\sigma(p), q)$. This contradicts the minimality of P so R cannot deadlock either.
Thus R has no deadlocks and cannot terminate. Symmetrically, the same holds for Q. Now take a step from Q, say $Q \xrightarrow{a} Q'$. By Proposition 8 $Q' \parallel R \in K^{+,\infty}_{G}(a(p), q)$, in fact due to nontermination $Q' \parallel R = G(a(p))$. Observe that both actions $s(0), s(1)$ must occur in the alphabet of $Q' \parallel R$.
Suppose Q' has a trace containing $s(0)$, say $Q' \xrightarrow{\sigma}{}^{*} Q'' \xrightarrow{s(0)} Q'''$, and R has a trace containing $s(1)$, say $R \xrightarrow{\rho}{}^{*} R' \xrightarrow{s(1)} R''$. Then $Q' \parallel R \xrightarrow{\sigma\rho}{}^{*} Q''' \parallel R''$ and both $s(0), s(1)$ are possible from this state. This is impossible, since a queue can only output one action at a time. It follows that R has no traces containing $s(1)$. As we have a bounded queue, its traces must have some output, so this must be $s(0)$. The same argument shows that Q' must show output $s(0)$ only, but then $Q' \parallel R$ never allows $s(1)$, which is a contradiction.

□

If we go beyond the signature of PA^{*}_{δ}, then finite specifications for a bounded queue without recursion (but with iteration) can be given. A well-known one is that an n-bounded queue can be given as a parallel composition of n coupled one-place buffers. In order to specify this, we need parallel composition with communication, encapsulation and abstraction. In terms of the chaining operator of Vaandrager (see [8]), we can give a definition as follows:

$$Queue(1) = Elt(1)^{*}\delta$$

$$Queue(n+1) = Queue(n) \gg Queue(1)$$

.
For more details, we refer to [8].
Here, we give a different finite specification for the queue, in the signature obtained by adding the state operator of [1] to $\mathrm{BPA}^{*}_{\delta}$. The state operator is indexed by a finite data type S, and comes with two functions:

- $action : A \times S \to A_{\delta}$, giving the action that is executed when an action is tried in a certain state (the result is δ if the intended action is blocked in this state)
- $effect : A \times S \to S$, giving the resulting state when the action is executed in a certain state

Then, we have the equations for the state operator given in Table 3.
Now, to define the n-bounded queue, we use as state space the set of sequences of data elements of length $\leq n$. We use actions $r(d), s(d)$ as before and the extra action *out*. The action and effect functions are trivial (i.e. $action(a, s) = a$, $effect(a, s) = s$) except in the following cases:

$\lambda_s(\delta) = \delta$
$\lambda_s(a) = action(a, s)$
$\lambda_s(a \cdot x) = a(s) \cdot \lambda_{effect(a,s)}(x)$
$\lambda_s(x + y) = \lambda_s(x) + \lambda_s(y)$

Table 3. Axioms for the State Operator ($a \in A$).

- $effect(out, \sigma^\frown[d]) = \sigma$ (for $d \in D$, $|\sigma| < n$)
- $effect(r(d), \sigma) = [d]^\frown\sigma$ (for $d \in D$, $|\sigma| < n$)
- $action(out, [\,]) = \delta$
- $action(out, \sigma^\frown[d]) = s(d)$ (for $d \in D$, $|\sigma| < n$)
- $action(r(d), \sigma) = \delta$ (for $d \in D$, $|\sigma| = n$)

Now, the definition of the n-bounded queue is as follows:

$$Queue(n) = \lambda_{[\,]}((out + \sum_{d \in D} r(d))^* \delta)$$

Then it is not difficult to show that $Queue(n) = Q^n([\,])$ holds for each n, by showing that $Q^n(\sigma) = \lambda_\sigma((out + \sum_{d\in D} r(d))^* \delta)$, for each sequence σ of length $\leq n$.

6 Conclusion

We find a remarkable similarity between the definability issues for bounded and unbounded bags, stacks and queues, when using iteration in the bounded cases and recursion in the unbounded cases.

References

1. J.C.M. Baeten and J.A. Bergstra. Global renaming operators in concrete process algebra. *Information and Computation*, 78:205–245, 1988.
2. J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.
3. J.A. Bergstra and J.W. 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.
4. J.A. Bergstra and J.W. Klop. Algebra of communicating processes. In J.W. de Bakker, M. Hazewinkel, and J.K. Lenstra, editors, *Mathematics and Computer Science I*, volume 1 of *CWI Monograph*, pages 89–138. North-Holland, Amsterdam, 1986.
5. J.A. Bergstra and J. Tiuryn. Process algebra semantics for queues. *Fundamenta Informaticae*, X:213–224, 1987.

6. D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference*, number 104 in LNCS, pages 167–183. Springer Verlag, 1981.
7. G.D. Plotkin. An operational semantics for CSP. In D. Bjørner, editor, *Proceedings Conference on Formal Description of Programming Concepts II*, pages 199–225. North-Holland, Amsterdam, 1983.
8. F.W. Vaandrager. Process algebra semantics of POOL. In J.C.M. Baeten, editor, *Applications of Process Algebra*, number 17 in Cambridge Tracts in Theoretical Computer Science, pages 173–236. Cambridge University Press, 1990.