Reflections on a Geometry of Processes

Clemens Grabmayer* Jan Willem Klop† Bas Luttik‡

June 10, 2005

Abstract

In this note we discuss some issues concerning a geometric approach to process algebra. We mainly raise questions and are not yet able to present significant answers.

1 Periodic Processes

Our point of departure is the axiom system BP in Table 1 together with guarded recursion.

$$\begin{array}{ll} x+y & = y+x \\ x+(y+z) & = (x+y)+z \\ x+x & = x \\ (x+y)\cdot z & = x\cdot z+y\cdot z \\ (x\cdot y)\cdot z & = x\cdot(y\cdot z) \end{array}$$

Table 1: BP (Basic Process Algebra)

We are in particular interested in *non-linear* recursion, where products of recursion variables are allowed, in contrast with linear recursion exemplified by $\langle X|X = aY + b,\ Y = cX + dY\rangle$ yielding only regular (finite-state) processes. Non-linear recursion also allows infinite-state processes, such as the counter $\langle C|C = uDC,\ D = uDD + d\rangle$ (with actions u, d for "up" and "down") or the process Stack that is definable by the infinite set of linear recursion equations over BP (cf. the left-hand side of Table 2), and more remarkably, by the finite set of non-linear recursion equations (cf. the right-hand side of Table 2).

This simple framework is already rich in structure. In [1] this framework was linked with context-free grammars (CFG's), in particular with those in (restricted) Greibach normal form.

*Vrije Universiteit Amsterdam. Postal address: Department of Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. E-mail: clemens@cs vu nl.

†Vrije Universiteit Amsterdam, Radboud Universiteit, and CWI. Postal address: Department of Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. E-mail: jwk@cs vu nl.

‡Eindhoven Technical University and CWI. Postal address: P.O. Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: s p luttik@tue nl.

$S_\lambda = 0\ S_0 + 1\ S_1$
$S_{d\sigma} = 0\ S_{0d\sigma} + 1\ S_{1d\sigma} + \underline{d}\ S_\sigma$
(for $d = 0$ or $d = 1$, and any string σ)

$S = T\ S$
$T = 0\ T_0 + 1\ T_1$
$T_0 = \underline{0} + T\ T_0$
$T_1 = \underline{1} + T\ T_1$

Table 2: Stack, an infinite linear and a finite non-linear BP -specification

There the fact was established that while the language equality problem for CFG's is unsolvable, the process equality problem for CFG's is solvable. A priori this is not implausible, because a process has much more inner 'structure' than a language (the set of its finite terminating traces). The decidability was demonstrated by Baeten, Bergstra, and Klop in [1] as a corollary of a result concerning the periodical geometry or topology of the corresponding process graph. In Figure 1 the periodicities of two examples are exhibited: of Stack on the left-hand side, and of the process $\langle X|X = bY + dZ, Y = b + bX + dYY, Z = d + dX + dZZ\rangle$ on the right-hand side (this graph repeats three finite graph fragments , β and γ as is also illustrated in Figure 2 below).

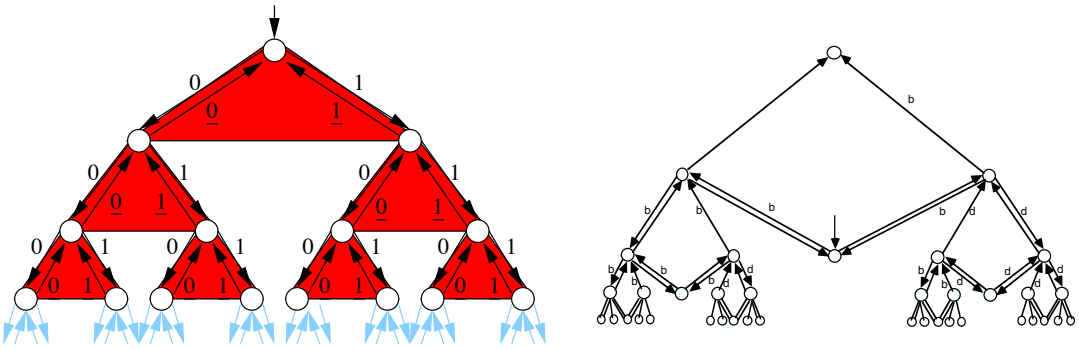

Figure 1: Tree-like periodic processes

The geometric proof in [1] is complicated. For the corollary of the decidability more stream-lined approaches have subsequently been found by using tableaux methods and other arguments (cf. Caucal in [7], Hüttel and Stirling in [11], and Groote in [10]). Also, the geometric aspects have been studied, for example by Caucal in [8] and by Burkart, Caucal, and Steffen in [5]. Actually, the related notion of *context-free graph* was introduced by Muller and Schupp [12] already in 1985.

We feel that there is still much to be explained about the geometric aspects of process graphs. We present a question concerning the fact that periodic graphs in BP come in two kinds: 'linear' graphs as on the left-hand side, and 'branching' graphs as on the right-hand side in Figure 2.

Question 1 *Is it decidable whether a system E of equations (in Greibach normal form) yields a linear (type I) or a branching (type II) graph?*

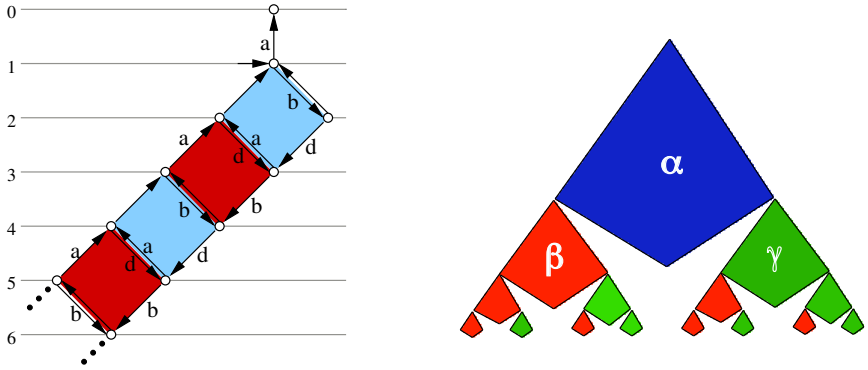

Figure 2: 'Linear' periodic graphs (type I, left), 'branching' periodic graphs (type II, right)

Another graph of type II is the 'butterfly' process graph in Figure 3 of the recursive BP -specification $\langle X|X = a + bY + fXY,\ Y = cX + dZ,\ Z = gX + eXZ\rangle$. The relevance of the

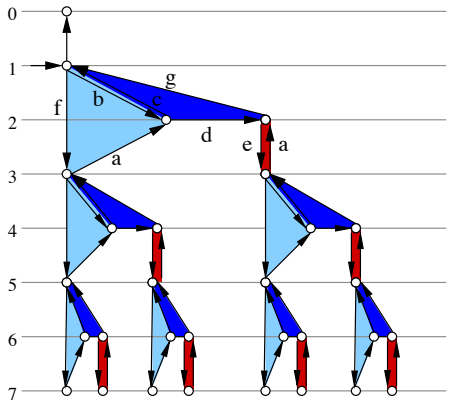

Figure 3: A 'butterfly' process graph.

distinction between type I and type II graphs is made clear below, in order to show that certain graphs are *not* of type I or type II.

In the study of BP -definable graphs an important property is that of being "normed". A graph is *normed* if from every node in it there is a path to a terminating node. (In term rewriting terminology this is called the weak normalization property WN.) The norm of a node is then the minimum number of steps to termination. Originally, the decidability of context-free processes (BP -definable processes) was established in [1] only for the normed case. Subsequently this was generalized by Christensen, Hüttel, and Stirling in [9] to all BP -definable processes.

Note that the norm of a node in a process graph is preserved under bisimulation: if norms are pictorially represented by drawing the process graph with horizontal 'level' lines, arranging points with the same norm on the same level (see the graph left in Figure 2 and the graph

in Figure 3), then bisimulations relate only points on horizontal lines. Collapsing a normed graph to its canonical form is a compression in horizontal direction.

An important question is whether BP -definable processes are closed under minimization (i.e. under compressing a graph such that it is minimal under bisimulation; the resulting graph is also called the "canonical" graph). The question whether such a statement does in fact hold was left open in [1]. Making a graph canonical can alter its geometry considerably. For instance, consider the counter C mentioned above. The process graph g of C is a linear sequence of nodes $C, DC, DDC,$ connected by u-steps to the right and d-steps to the left. The merge $C \parallel C$ in the process algebra P has a grid-like graph similar to that of the process Bag on the left side in Figure 6 below. But if we collapse this graph g for $C \parallel C$ to its canonical form by identifying the bisimilar nodes on diagonal lines, we obtain again the graph g for C. So a grid may collapse to a linear graph.

Normedness plays a part when graphs are compressed to their canonical form. In [5] Burkart, Caucal, and Steffen give the following example of a BP -graph that after compression to canonical form no longer is a BP -graph: For the process with recursive definition $\langle Z|Z = aAZ + cD,\ A = aAA + cD + b,\ D = dD\rangle$ in BP , the graph on the left in Figure 4 is its associated BP -process graph, while the graph on the right is the respective minimization,

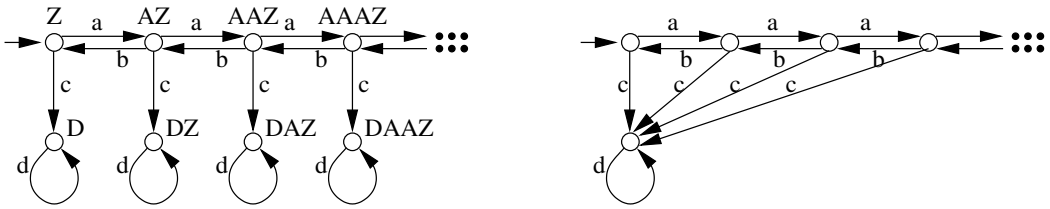

Figure 4: Counterexample against the preservation of BP -graphs under minimization.

which does not have the periodical structure of a BP -graph. Note that neither of these graphs is normed.

Question 2 *How can those BPA-graphs be characterized whose canonical graphs are again* BP *-graphs?*

We note that Question 2 has already received quite some attention in Caucal's work. Contrasting with the counterexample for the unnormed case given above, in [7] he has shown the following theorem.

Theorem 1 (Caucal, 1990) The class of normed BP -graphs is closed under minimization.

The (obvious) link between CFG's and BP -definable processes was first mentioned in [1]. An example is the graph on the right in Figure 1 and in Figure 2 above: it determines as context-free language (CFL) the language of words having equal numbers of letter b and d. An intriguing question is the following.

Question 3 *How does the classical pumping lemma for CFL's relate to the periodicity present in* BP *-definable processes?*

Another interesting observation, due to H.P. Barendregt, is the following. It is well-known that the language $L = \{a^n b^n c^n | n \geq 0\}$ is not a CFL. This language can be obtained as the set of finite traces of the triangular, infinite, minimal graph in Figure 5. Intuitively it is obvious that this graph is not tree-like periodic. This leads to the next question.

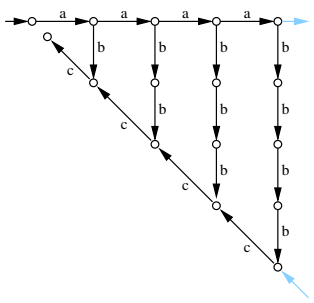

Figure 5: The language L.

Question 4 *Can the fact that the graph in Figure 5 is not a* BP *-graph (when established rigorously) be used to conclude that L is not a CFL, applying the correspondence between CFL's and definability in* BP *as well as the ensuing tree-like periodicity?*

2 Non-definability of Bag in BP

The expressiveness of the operations defined by the axioms of BP is limited; basically only sequential processes can be defined. The axiom system P is an extension of BP with axioms for the merge $\|$ (interleaving) and the auxiliary operator $\lfloor\!\lfloor$ (left merge). In P we

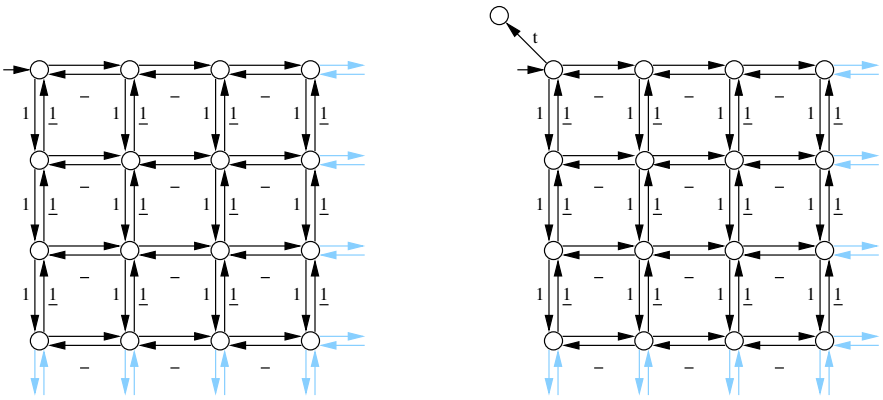

Figure 6: The minimal process graphs of the process Bag (on the left-hand side), and of a terminating variant Bag_t of Bag (on the right-hand side).

have a succinct recursive definition for the process Bag (over data $\{0,1\}$) as follows:

$$\mathrm{B} = 0\ \underline{0}\,\|\,\mathrm{B}) + 1\ \underline{1}\,\|\,\mathrm{B})$$

It has been proved by Bergstra and Klop in [3] that the process Bag cannot be defined by means of a finite recursive specification over BP . Considering the minimal process graph for it in Figure 6, this does not come as a surprise: it is not tree-like, but "grid-like". Below we give an alternative proof of this fact.

Theorem 2 (Bergstra, Klop, 1984) Bag is not BP -definable.

Proof (Sketch). Suppose that the process Bag is BP -definable. Then there exists a recursive specification E in BP such that Bag is bisimilar with a tree-like periodic graph $g\ E)$ as defined by Baeten, Bergstra, and Klop in [1]. Then $g\ E)$ is a "BPA-graph" according to the terminology used in [5].[1]

In [5] Burkart, Caucal, and Steffen have shown that, for *every* BP -graph G, the canonical graph of G is a "pattern graph", which means that it can be generated from a finite (hyper)graph by a reduction sequence of length ω according to a deterministic (hypergraph) grammar.[2] Since Bag is itself a canonical graph and since therefore Bag is the canonical graph of the BP -graph $g\ E)$, it follows that Bag is a pattern graph.

A theorem due to Caucal in [8] states that all (rooted) pattern graphs of finite degree are "context-free" according to the definition of Muller and Schupp in [12].[3] It follows that Bag is context-free. However, it is not difficult to verify that Bag is actually *not* a context-free graph according to the definition in [12].

In this way we have arrived at a contradiction with our assumption that Bag is definable in BP .

By using Caucal's theorem, Theorem 1, it is also possible to establish quickly the non-definability in BP of many normed graphs. For example, for the terminating version Bag_t of Bag (where Bag_t is normed) with the process graph on the right in Figure 6, it can be reasoned as follows. This graph is canonical, so if it were BP -definable, then it would be a graph of type I or type II. However, for a type I graph it holds that the number of nodes in a sphere $B\ s,\rho)$, where s is the center and ρ is the radius, depends linearly on ρ; for a type II graph this dependance is of exponential form. But for the graph under consideration the number of nodes in a ball $B\ s,\rho)$ only depends quadratically on ρ. Hence this graph is not BP -definable.

Where do we need the preservation of BP -definability under minimization? The process graph of Bag_t is clearly not one obtainable by a BP -definition, as it is not of type I or type II. But equality of processes is considered here modulo bisimulation—so it is not inconceivable that there is a BPA-definition E of Bag_t such that $g\ E)$ *after compression* to canonical form can $g\ E))$ were just the process graph graph $\mathrm{Bag}_t)$ for Bag_t on the right in Figure 6. So can $g\ E)) =$ graph $\mathrm{Bag}_t)$ holds. But with the preservation property, Theorem 1, we have can $g\ E)) = g\ E')$ for some BP -specification E', hence $g\ E')$, and therefore graph $\mathrm{Bag}_t)$, are of type I or type II, quod non.

[1] In earlier papers of Caucal (e.g. in [6] and [8]) BPA-graphs were known under the name "alphabetic graphs".

[2] "Pattern graphs" according to this definition used by Caucal and Montfort in [6] are called "regular graphs" in the later paper [5] by Burkart, Caucal, and Steffen. Because the use of the attribute "regular" for process graphs could lead to wrong associations, we avoid this terminology from (hyper)graph rewriting here.

[3] Note that the class of "context-free" graphs in Muller and Schupp's definition does not coincide with the graphs associated with "context-free" processes (the class of BP -graphs), but that it forms a strictly richer class of graphs corresponding to the class of transition graphs of push-down automata.

3 The strange geometry of Queue

After the paradigm processes Stack and Bag, we now turn to the third paradigm process Queue (the first-in-first-out version with unbounded capacity). Table 3 gives the infinite BP -specification.

$$\begin{array}{l} Q = Q_\lambda = \sum_{d\in D} r_1\ d)\ Q_d \\ Q_{\sigma d} = s_2\ d)\ Q_\sigma + \sum_{e\in D} r_1\ e)\ Q_{e\sigma d} \\ (\text{for } d \in D, \text{and } \sigma \in D^*) \end{array}$$

Table 3: Queue, infinite BP -specification

As before, the endeavour is to specify Queue in a finite way. It was proved by Bergstra and Tiuryn [4] that the system BP is not sufficient for that; in fact, they showed that Queue cannot even be defined in CP *with handshaking communication* (see [2] for a complete treatment of the axiom system CP). But Queue has a finite recursive specification in CP with *renaming* operators (see Table 4, the specification is originally due to Hoare using the 'chaining'-operation).

$$\begin{array}{l} Q = \sum_{d\in D} r_1\ d)\ \rho_{c_3\to s_2} \circ \partial_H)\ \rho_{s_2\to s_3}\ Q) \,\|\, s_2\ d)\ Z) \\ Z = \sum_{d\in D} r_3\ d)\ Z \end{array}$$

Table 4: Queue, finite CP-specification with renaming

An ambitious question is the following.

Question 5 *Is there a geometric (topological) property of processes definable by handshaking communication?*

Finally, we turn to geometric properties of the process Queue. Surprisingly, it is unexpectedly problematic to draw the process graph of Queue in a 'neat' way (cf. also Figure 7), similar to Stack and Bag. We would like to uncover the 'deep' reason for this difficulty.

Question 6 *Is it possible to fit g* Queue) *in the binary tree space?*

References

[1] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for process generating context-free languages. *Journal of the ACM*, 40(3):653–682, 1993.

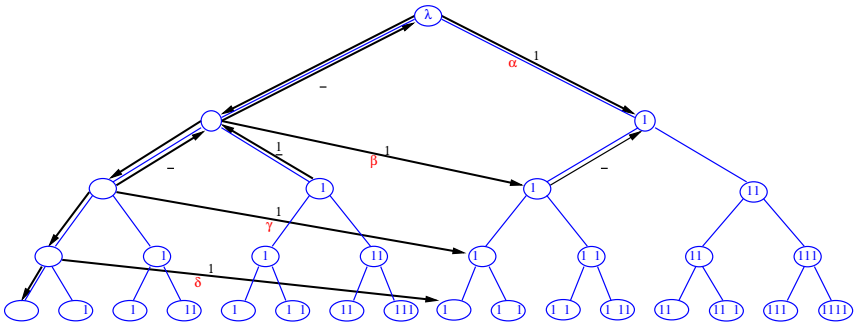

Figure 7: Attempt at drawing Queue in 'tree space'.

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

[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 of ICALP'84*, volume 172 of *LNCS*, pages 82–95. Springer, 1984.

[4] J. A. Bergstra and J. Tiuryn. Process algebra semantics for queues. *Fundamenta Informaticae*, X:213–224, 1987.

[5] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *Proceedings of CONCUR'96*, 1996.

[6] D. Caucal and R. Montfort. On the transition graphs of automata and grammars. In *Proceedings of WG 90*, volume 484 of *LNCS*, pages 61–86. Springer, 1990.

[7] D. Caucal. Graphes canoniques de graphes algébriques. *Theoret. Inform. and Appl.*, 24(4):339–352, 1990.

[8] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 1992.

[9] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121:143–148, 1995.

[10] J. F. Groote. A short proof of the decidability of bisimulation for normed bpa-processes. *Information Processing Letters*, 42:167–171, 1992.

[11] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of LICS'91*, pages 376–386, 1991.

[12] D.E. Muller and P.E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 1985.

From CRL to mCRL2

Motivation and outline

Jan Friso Groote, Aad Mathijssen, Muck van Weerdenburg, Yaroslav Usenko
Department of Mathematics and Computer Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
J.F.Groote, A.H.J.Mathijssen, M.J.van.Weerdenburg, Y.S.Usenko}@tue.nl

Abstract

We sketch the language *mCRL2*, the successor of μCRL, which is a process algebra with data, devised in 1990 to model and study the behaviour of interacting programs and systems. The language is improved in several respects guided by the experience obtained from numerous applications where realistic systems have been modelled and analysed. Just as with μCRL, the leading principle is to provide a minimal set of primitives that allow effective specifications, that conform to standard mathematics and that allow standard mathematical manipulations and proof methodologies. In the first place the equational abstract datatypes have been enhanced with higher-order constructs and standard data types, ranging from booleans, numbers and lists to sets, bags and higher-order function types. In the second place multi-actions have been introduced to allow a seamless integration with Petri nets. In the last place communication is made local to enable compositionality.

1 The history of CRL

In an attempt to construct a language to which all existing specification languages could be translated, a common representation language (CRL) was constructed in an EC funded project called SPECS. This language became a monstrum for which is was impossible to device a coherent semantics, let alone to be used as a basis for further theory or tool building.

Upon these findings, in 1990 a minimal language called μCRL (*micro Common Representation Language*) came into being as the simplest conceivable language to model realistic systems. The language is a process algebra with data. The data is specified using first-order equational logic which was the norm at the time. Earlier developed languages such as LOTOS [2] and PSF [8] also contained equational datatypes. However, μCRL was much simpler than these languages.

In the first research phase proof methodologies were developed to give mathematical proofs of distributed algorithms and protocols. A number of proof techniques have been uncovered such as *cones and foci*, *τ-confluence* and *coordinate transformations* (see [6] for an overview). Many systems have been verified using these techniques, but particularly noteworthy is the most complex sliding window protocol in [9] (see [3]). Verification of this protocol led to the detection of an unknown deadlock in the protocol, it showed that the external behaviour of the original protocol was prohibitively complex and catalysed the development of many proof methodologies.

In the second research phase a toolset for μCRL was developed [1]. The primary motivation for this was that industrial specifications quickly became far too large to be handled manually. Large specifications, like ordinary programs, turned out to contain flaws such as deadlocks and tools were

required to ensure the absence of anomalies. For plain verifications, the tool can handle systems with more than 10^9 states. By using confluence, abstract interpretation and symbolic reasoning much larger systems, containing hundreds of components have been verified. For half a decade the tool plays an essential role in teaching the design of dependable systems at various universities.

2 Why must CRL be changed?

It turns out to be impossible to design a complete specification language that is immediately right. In [4] time was added to the language. Furthermore, constructors were added to the specification of functions in the datatypes of μCRL to make the available induction principles explicit. And finally, the possibility to specify an initial state of a process had to be added. As time passed it became more and more obvious that the language would benefit from some more changes.

First of all changes were required in the abstract data types, although their expressive power was more than sufficient. A relatively minor problem was that in μCRL all basic datatypes, such as the naturals and the booleans had to be explicitly encoded. Much more serious was the negative effect on interhuman communication of specifications. Different persons could give widely different specifications of for instance the naturals. This meant that before getting to the gist of a specification, first the specification of the naturals had to be understood. Furthermore, because all functions in μCRL are prefix functions, standard notation, such as an infix + for addition on natural numbers could not be used. This is not a problem for small specifications but seriously decreases the readability of large ones.

In practice first-order abstract datatypes also discourage the use of higher-order objects, such as functions, sets, relations and quantifiers. For instance sets are often modelled as finite lists. This tends to make specifications more complex than necessary.

A strong argument against the use of bare abstract data types came from manually proving the correctness of specifications. Given a specification, many elementary facts about the data are not self evident and proving them draws away energy from the main task, namely finding the core correctness argument for the protocol or distributed system under study. For an abstractly specified sort Nat, it is not self evident that it indeed represents natural numbers in a true way. Hence, the truth of simple identities had to be established using axioms and induction principles. For instance commutativity of addition must be established separately for each specification of natural numbers. For tools, properties like $x > y \wedge y > z \rightarrow x > z$ turned out a hurdle that was hard to overcome. By having standard data types, dedicated integer linear programming techniques can be employed with which we can prove or disprove the validity of inequality based formulas that are many orders of magnitude larger than the one above. Actually, the μCRL toolset already made a number of silent assumptions about certain data types (esp. the booleans) and certain functions (esp. it assumed that a function *eq* represented equality). This enabled the development of a very effective equality BDD prover [5] but actually violates the philosophy of abstract data types.

Despite these disadvantages, equational abstract data types were more than sufficiently expressive for any data type that needed to be specified. As the structure of data is very simple, we could device optimal algorithms to handle data with little effort. Repeated comparative experiments show that the μCRL tool set contains the most efficient state exploration tools in terms of the number of states that it can store in main memory. Comparing to for instance SPIN [7], the μCRL toolset is approximately a factor 4 slower in dealing with abstractly specified bits and bytes, which are built-in data types in SPIN.

Another issue that we ran into with μCRL is the relationship between different process specifi-

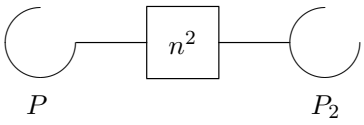

Figure 1: A simple coloured Petri net

cation formalisms. We see three main streams. There are assertional specification formalisms, Petri nets and process algebras. We would all benefit if these formalisms would be integrated. In the past we did not find any difficulties relating assertional methods and μCRL. However with Petri nets we ran into a problem. Consider the coloured Petri net in figure 1. There are two places P and P_2 and a transition labeled with n^2 in the middle. The tokens in this coloured Petri net contain natural numbers and the transition squares the number in each token that it processes. The standard semantics of this system is that a token atomically leaves P , its value is squared and it is put in P_2.

The natural structure preserving translation of this Petri net into process algebra is the parallel process $P \parallel T \parallel P_2$. Using a standard synchronous communication a token can be read from P into T, and in a subsequent step be forwarded from T to P_2. But now we have translated what was a single atomic step into two atomic steps. This is bad for at least the two following reasons. In the first place this innocent looking doubling of states increases the number of states worsening the severity of the state explosion problem, which is one of the core problems we try to avoid. In the second place nice properties about Petri nets, such as state invariants do not easily carry over when introducing such intermediate states.

In order to avoid the introduction of such an intermediate state and still allow for direct structural translations, we felt forced to introduce multi-actions. In a multi-action zero or more actions can occur simultaneously. The typical notation is $a|b|c$ for a multi-action in which actions a, b and c happen at the same time. Now we can describe the transition in figure 1 by a process that reads a token with value n and in the same multi-action delivers the token with a value n^2. There is no straightforward way to do this in μCRL.

Another problem occurs when describing complex systems with non-uniform communication. In μCRL there is a global communication operator that is not compositional. To make the new language compositional, we need to define it locally.

3 The mCRL2 language

The mCRL2 language is a movement back from the bare minimum concept of μCRL towards a slightly richer language. Therefore, we propose to call it a *milli Common Representation Language*, or *mCRL*. Experience has taught that though we have designed the language with utmost care, we may still have made mistakes in its design and fundamentally new extensions such as stochastic or hybrid behaviour may be added in the future. Hence, we added a version number to the name paving the way for *mCRL3*, *mCRL4*, etc. to come. By the way, the name μCRL is not really suited for internet because of the initial Greek letter.

3.1 Data language

The mCRL2 data language uses *higher-order* abstract data types as a core theory. To this theory, standard data types are added. We list these data types without further ado as they are commonly known. All the common operators on these are made available in normal mathematical notation. In order to get a quick idea, an expression using this datatype is provided.

- The sort with constants *true*, *false* and all standard operators. It is also possible to use the quantifiers $\forall$ and $\exists$ ranging over any datatype. E.g. $b \wedge \mathit{false} \Rightarrow \forall n{:}\mathbb{N}.n < 3$.
- Unbounded positive, natural and integer numbers. Typical examples of expressions using numbers are $1 \quad 464748473698768976$ **div** $\mathit{exp}(3,n), \mathit{succ}(m) \leq n \quad 1$ or $x == x * x \quad 1$.
- Function types. For two given sorts *A* and *B* the sort $A{\to}B$ contains all functions from domain A to B. Function application and lambda abstraction are part of the language. E.g. let $f = \lambda x{:}\mathbb{N}, b{:}\ \ .\mathit{if}(b, x, 2 * x)$. Then $f(3, \mathit{false})$ is equal to 6.
- Following functional languages, it is possible to declare structured types. These are especially useful for enumerated data types and complex data structures such as for instance trees. A sort *MS* of machine states can be declared by

  $$\textbf{sort}\ \mathit{MS}\ =\ \textbf{stru}\ \ \textbf{t}\ o\quad \mid \mathit{standby} \mid \mathit{starting} \mid \mathit{running} \mid \mathit{broken};$$

  The sort of binary trees with numbers as their leaves looks like

  $$\textbf{sort}\ T\ =\ \textbf{stru}\ \ \textbf{t}\ \mathit{leaf}(\mathbb{N}) \mid \mathit{node}(T,T);$$

  It is possible to specify projection and recognition functions simultaneously, e.g.:

  $$\textbf{sort}\ T\ =\ \textbf{stru}\ \ \textbf{t}\ \mathit{leaf}(\mathit{getnumber}{:}\mathbb{N})?\mathit{isLeaf} \mid \mathit{node}(\mathit{left}{:}T, \mathit{right}{:}T)?\mathit{isNode};$$

- Because lists are very commonly used datatypes, there is a built-in type of lists with standard operations. The list of natural numbers is $\mathit{List}(\mathbb{N})$. The following list expressions are all equivalent: $[3,4,5]$, $3 \triangleright [4,5]$, $[3,4] \triangleleft 5$ and $[] +\!\!+ [3,4] +\!\!+ [5]$.
- Sets are very commonly used in mathematical specification, and as bags are a basic concept in Petri nets, both have been included in the language. Sets are denoted in the normal mathematical way. Typically, $\{1,2,4\}$, $\{1,2\} \cup \{1,4\}$ are sets. The set of primes is

  $$\{n{:}\mathbb{N} \mid \forall m{:}\mathbb{N}.(1 < m \wedge m < n\ \Rightarrow\ n\ \textbf{mod}\ m > 0)\ \}.$$

- Bags are sets where the multiplicity of elements is recalled. For enumerations this count is appended to each element, e.g. $\{0{:}0, 1{:}1, 2{:}4\}$. For comprehensions the boolean condition is replaced by a natural number, e.g. $\{m{:}\mathbb{N} \mid m^2\}$ is the bag in which each number m occurs m^2 times.

Currently, there are discussions about the inclusion of real numbers. As functions are available, it is possible to represent real numbers. Moreover, this opens the way towards stochastic and hybrid systems where functions from reals to reals play an important role. Another interesting concept is the selector functions ε. The expression $\varepsilon x{:}S.c(x)$ equals a unique value x that satisfies condition $c(x)$. It satisfies the axiom $\exists x{:}S.c(x) \Rightarrow c(\varepsilon x{:}S.c(x))$. These extensions may show up in mCRL3.

3.2 Multi-actions and local communication

In order to facilitate the connection with Petri nets, multi-actions are introduced. A multi-action is a collection of ordinary actions that happen at the same time. A few examples of multi-actions are $a,\ a|b,\ b|a,\ a|b|c,\ a|b|a$ and $a(t)|b(u)|a(v)$.

In mCRL2 parallel composition does not communicate. Instead, it introduces multi-actions, e.g. the composition $a \parallel b$ of actions a, b is equal to $a \cdot b + b \cdot a + a|b$. As a result the number of multi-actions can increase exponentially in the size of the number of parallel compositions. Hence, we also need operators to restrict this behaviour. First of all we have the *blocking* operator ∂_H (which was called encapsulation in μCRL) that blocks all multi-actions of which a part occurs in the action set H, e.g. $\partial_{\{a\}}(a + b \cdot (a|c)) = b \cdot$. On the other hand, we have the visibility operator ∇_V called *allow* that specifies precisely which multi-actions are allowed, namely the ones in V. For instance $\nabla_{\{a,b\}}(a \parallel b) = a \cdot b + b \cdot a$, $\nabla_{\{a|b\}}(a \parallel b) = a|b$, and $\nabla_{\{a,b|c\}}(a \parallel b \parallel c) = a \cdot (b|c) + (b|c) \cdot a$.

Communication of actions is defined using the concept of multi-actions. The *local* communication operator $_C$ realises communication of multi-actions with equal data arguments. Unlike μCRL, communication does not block. For instance, if $t = u$ and $t \neq v$, then $_{\{a|b,c\}}(a(t)|b(u)) = c(t)$, $_{\{a|b,c\}}(a(t)|b(v)) = a(t)|b(v)$ and $_{\{a|b|c,d\}}(a|b|c|d) = d|d$, but also $\sum_{d:D} {}_{\{a|a,a\}}(a(d)|a(t)) = \sum_{d:D} d = t \rightarrow a(t), a(d)|a(t)$, i.e. if $d = t$ then $a(t)$ and if $d \neq t$ then $a(d)|a(t)$ for a certain d.

4 Epilogue

The language mCRL2 is an attempt to make μCRL more applicable in practise and to facilitate hierarchical Petri nets. The language is extended with higher-order datatypes, standard datatypes, multi-actions and local communication. Because the new language has essentially the same structure as its predecessor, all current μCRL specifications can be easily expressed in the new language and all proof methodologies, theorems and tools carry over with only minor modifications.

References

[1] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A Toolset for Analysing Algebraic Specifications. In proceedings CAV'01. LNCS 2102, pages 250–254, 2001.

[2] P.H.J. van Eijk, C.A. Vissers and M. Diaz, editors. *The formal description technique LOTOS*. Elsevier Science Publishers B.V., 1989.

[3] W. Fokkink, J.F. Groote, J. Pang, B. Badban and J.C. van de Pol. Verifying a sliding window protocol in μCRL. In C. Rattray, S. Maharaj and C. Shankland (eds), proceedings of the 10th International Conference on Algebraic Methodology and Software Technology, Stirling, Scotland, LNCS 3116, Springer-Verlag pp. 148-163, 2004.

[4] J.F. Groote. The syntax and semantics of timed μCRL. Technical report SEN-R9709, CWI, Amsterdam, 1997.

[5] J.F. Groote and J.C. van de Pol. Equational Binary Decision Diagrams. In M. Parigot and A. Voronkov, *Logic for Programming and Reasoning, LPAR2000*, Lecture Notes in Artificial Intelligence, volume 1955, Springer Verlag, pages 161-178, 2000.

[6] J.F. Groote and M. Reniers. Algebraic process verification. In J.A. Bergstra, A. Ponse and S.A. Smolka. Handbook of Process Algebra, pages 1151-1208, Elsevier, Amsterdam, 2001.

[7] G.J. Holzmann. The spin model checker: Primer and reference manual. Addison-Wesley, 2003.

[8] S. Mauw and G.J. Veltink. A process specification formalism. *Fundamenta Informaticae*, XIII:85–139, 1990.

[9] A.S. Tanenbaum. *Computer Networks*. Prentice Hall, 1981.