Composition of Motion Description Languages

Wenqi Zhang and Herbert G. Tanner

Mechanical Engineering Department,
University of New Mexico, Albuquerque NM 87131.

Abstract. We introduce a new formalism to define compositions of interacting heterogeneous systems, described by extended motion description languages (MDLes). The novelty of the formalism is in producing a composed system with a behavior that could be a superset of the union of the behaviors of its generators. We prove closedness of MDLes under this composition and we show that in the class of systems modeled using MDLes, language equivalence can be decidable. Our approach consists of representing MDLes as normed processes, recursively defined as a guarded system of recursion equations in restricted Greibach Normal Form over a basic process algebra. Basic processes have well defined semantics for composition, which we exploit to establish the properties of our composed MDLes.

1 Introduction

Motion Description Languages (MDLs) [1] translate collections of control algorithms into robust and reusable software [2]. MDLes (e standing for "extended") have been criticized for not capturing concurrency and interaction between systems. This paper is an attempt to address this issue, and set a framework in which MDLes can be composed, verified, and allow automated motion and task planning for collections of heterogeneous robotic systems.

We identify MDLes as recursive systems in some basic process algebra (BPA) written in Greibach Normal Form (Lemma 1). We propose a simple context-free grammar that generates MDLes and then we use the machinery available for BPAs to formally define a composition operation for MDLes at the level of grammars. The technical core of this paper indicates how appropriately defined MDLe grammars can be composed (Definition 8), and language equivalence (whether two such grammars generate the same finite traces), is decidable up to bisimulation. The main difference of our composition operation is the appearance in the composed system of events (transitions) not enabled in the generators: the composed system can behave in ways its generators cannot. In our approach, one still needs to identify beforehand these events that can be activated after the composition, but still the proposed definition partially captures the fact that the whole can be more than the sum of its parts.

A natural question that comes up is *why are basic process algebras a good formalism to map hybrid robotic systems to discrete models of computation?* This formalism is one of many possible; we are not sure if any single one is best. The justification for choosing BPAs comes first from our desire to model robotic systems using MDLes. Section 3 attempts a brief and incomplete introduction to MDLes and BPAs; the interested reader is refered for more information to [3] and [4].

Another modeling formulation is maneuver automata [5], which are finite automata that produce sequences of predetermined maneuvers for unmanned vehicles. Admissible motion is expressed as the set of traces the automaton accepts. Maneuver automata, however, generate regular languages, and MDLes are not [3]. Thus, maneuver automata appear to have less expressive power than MDLes.

Another popular framework in which concurrent systems are expressed is petri nets [6]. Petri nets generate context-sensitive languages. They are therefore more expressive than BPAs but this comes at a cost: bisimulation is undecidable for petri nets [7], which poses an obstacle for further analysis and abstraction. MDLes, on the other hand, are context-free [8]. The equality problem for context-free languages and push-down processes being undecidable notwithstanding [9], we show in this paper that the slightly finer semantics given to an MDLe expressed in a BPA framework allow decidability for language equivalence [10, 11]. The tools we use to arrive at this decidability result are the properties of BPAs introduced in [12–14], and refined in [15].

Although other modeling tools may be available, we feel that BPAs strike a reasonable balance between complexity and expressiveness when it comes to modeling systems expressed by, and controlled through, MDLes. Showing that under the extended notion of composition we introduce, the resulting system is an MDLe (Lemma 3), and that the decidability properties are preserved (Corollary 2), gives us hope that the resulting (big) system can be abstracted to the point that some of the available model checkers [16–18] can be used to construct admissible motion plans in the form of "counterexamples."

2 MDLe Preliminaries

MDLe is an extension of the early definitions of motion description languages [19]. It is a device-independent programming language for hybrid motion control, which allows one to compose complex, interrupt-driven control laws from a set of simple primitives, and a number of syntactic rules [2, 3]. Every MDLe string consists of a control part, an interruption part, and the special symbols ")", "(", and ",". Consider a robotic system, generically described in the form of the following dynamics

$$\dot{x} = f(x, u), \quad y = h(x); \quad x \in \mathbb{R}^n,\ u \in \mathbb{R}^m,\ y \in \mathbb{R}^p, \tag{1}$$

where x is the state of the system, u the control input, and y the measurable output. Let U be a finite set of feedback control laws (or quarks [8]) $u : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$, for (1), and B a finite set of boolean functions $\xi : \mathbb{R}^m \times \mathbb{R} \to \{0, 1\}$ of output y and time $t \leq T \in \mathbb{R}_+$ (the interrupt quarks [8]).

The basic element of an MDLe is an *atom*, denoted (ξ, u), where ξ is the *interrupt* selected from set B, and u is a control law selected from U. To *evaluate* or *run* an atom (ξ, u), means to apply the input u to (1) until the interrupt function ξ evaluates true ($\xi = 1$). An MDLe *plan* is composed of a sequence of atoms. For example, evaluating the plan $a = ((u_1, \xi_1), (u_2, \xi_2))$ means that the system state x, flows along $\dot{x} = f(x, u_1)$ until $\xi_1 = 1$, and then along $\dot{x} = f(x, u_2)$ until $\xi_2 = 1$. Plans can also be composed to generate higher order strings, as in $b = ((u_3, \xi_3), a, (u_4, \xi_4))$.

3 From MDLes to Basic Process Algebras

3.1 MDLes are context-free

The pumping lemma is utilized in [8] to show that an MDLe is not a regular language; rather, it is context-free. Context-free languages are generated by context-free grammars (CFGs), which can always be expressed in Chomsky normal form. A variation of the Chomsky normal form, is the Greibach normal form.

Definition 1 ([15]). *A context-free grammar in which every production rule is of the form $A \to a\alpha$, where A is a variable, a is a terminal, and α is a possibly empty string of variables, is said to be in* Greibach normal form *(*GNF*). If, moreover, the length of α (in symbols) does not exceed* 2*, we say that the context-free grammar is in restricted Greibach normal form.*

It is well known [20], that pushdown automata and context-free languages are equivalent in power.

3.2 The link between BPAs and push-down automata

A BPA is essentially an mathematical structure consisted of set of constants, $A = \{a, b, c, \ldots\}$, called *atomic actions*, a set Σ_{BPA} of two binary operators on these constants (the alternative composition $+$ and the sequential composition $\cdot$), and a set of axioms E_{BPA} that determines the properties of the operations on the atomic actions [15]. When the set of atomic actions A is assumed known, a basic process algebra is denoted simply as a couple in the form $\mathrm{BPA} = (\Sigma_{\mathrm{BPA}}, E_{\mathrm{BPA}})$. The set Σ_{BPA} is sometimes called *signature*, while set E_{BPA} *equation* set (hence the symbols). The theory associated to a BPA is considered to be parameterized by the set A, which is specified according to the particular application.

The symbol $\cdot$ denoting sequential composition is typically omitted, and we usually write xy instead of $x \cdot y$. We assume that $\cdot$ binds stronger than $+$, thus $(xy)+z = xy+z$ (brackets omitted). The set E_{BPA} consists of five axioms (or equations), appearing in Table 1. Composing atomic actions according to Table 1, yields more complex *processes*. Any such process, is an element of some algebra satisfying the axioms of BPA, and all processes produced in this way make up the set P. The axiom system of Table 1 is the core of a variety of more extensive process axiomatizations:

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5

Table 1. The axioms of a BPA.

- $x \cdot y$ is the process that first executes x, and upon completion of x, process y starts.
- $x + y$ is the process that either executes x, or executes y (but not both).

Just as in the case of finite state machines, processes are identified by the set of action sequences they admit. Some [21] prefer to include a set Atom of atomic processes or *atoms*. The set Proc of *processes* contains all terms in the free algebra over Atom generated by sequential composition and disjunction. Then a process algebra is defined by a finite set Π of productions of the form $X \xrightarrow{a} P$, where $X \in \mathrm{Atom}$, $a \in A$, and $P \in \mathrm{Proc}$. The semantics of the above production is as follows: atomic process X performs action a and evolves into process P. Let us identify a process with an automaton, in which a transition denotes the execution of an atomic action. The states of this automaton are all the processes derived through the set of production rules. Action relations are presented in Table 2, in which $x \xrightarrow{a} y$, with x and y being processes and a an atomic action, means that process x evolves into process y after the atomic action a is executed.

$a \xrightarrow{a} \surd$	R1
$a \xrightarrow{a} x' \Rightarrow x+y \xrightarrow{a} x'$ and $y+x \xrightarrow{a} x'$	R2
$x \xrightarrow{a} \surd \Rightarrow x+y \xrightarrow{a} \surd$ and $y+x \xrightarrow{a} \surd$	R3
$x \xrightarrow{a} x' \Rightarrow xy \xrightarrow{a} x'y$	R4
$x \xrightarrow{a} \surd \Rightarrow xy \xrightarrow{a} y$	R5

Table 2. The operational semantics of BPA.

The symbol $\surd$ stands for successful termination. It is said that a relation is true if and only if it can be derived from the relations of Table 2. Note the distinction between the relation operator ($\rightarrow$) and sequential composition ($\cdot$): the fact that $x \xrightarrow{a} y$ does not imply that $y = x \cdot a$, since a is an action executed as x runs, not after it is completed. The only thing that can be inferred about action a is that it is an action that process x can execute.

3.3 Recursive and guarded processes

Let us focus on a special type of BPAs with slightly finer semantics. The additional properties of this type of systems enable us to define composition more comfortably, and prove the decidability of language equivalence for the systems produced by means of composition.

Definition 2 ([10]). *A* recursive equation *over a* BPA *is an equation of the form $X = s(x)$, where X is a variable that can take values in P and $s(x)$ is a term over the* BPA *containing X, but no other variable.*

A set of recursive equations give rise to a specification:

Definition 3 ([10]). *A recursive specification E over a* BPA *is a set of recursion equations over the* BPA.

We thus have a set of variables $V = \{x_0, \cdots, x_n\}$, and equations of the form $X = s_x(V)$ with $x \in V$, where s_x is a term over the BPA containing variables in V. Set V contains one distinguished variable called the root variable x_0. A variable in V is called guarded in a given term, if it is preceded by an atomic action:

Definition 4 ([10]). *Let s be a term over a* BPA*, containing variable X.*

- *An occurrence of X in s is said to be* guarded*, if s has a sub term of the form $a \cdot t$, where a is an atomic action, and t a term containing this occurrence of X; otherwise this occurrence of X in s is said to be* unguarded*.*
- *A term s is* completely guarded *if all occurrences of all variables in s are guarded. A recursive specification E is completely guarded if all right hand sides of all equations of E are completely guarded terms.*

Just as production rules can be thought to be in Greibach normal form, so can equations over a BPA.

Definition 5 ([10]). *If a system E of recursion equations is guarded and without brackets, then each recursion equation is of the form $X_i = \sum_j a_j \cdot \alpha_j$, where α_j is a possibly empty product (sequential composition) of atoms and variables. Now if, in addition, α_j is exclusively a product of variables, E is said to be in* Greibach normal form*, analogous to the same definition for context-free grammars. If each α_j in E has length not exceeding 2, E is in* restricted *Greibach normal form.*

3.4 Composition of BPAs

BPAs can be equipped with a merge operator, $\|$. Process $x\|y$ is the process that executes process x and y in parallel. Notice that we do not assert that the first action has terminated when the second one starts; this can depend on the implementation of a process. The left merge operator, $\lfloor\!\lfloor$, describes two processes that occur in parallel, in a way similar to $\|$, but with the restriction that the first step must come from the process on the left of the expression. With the new operators, the BPA axioms are expanded as shown in Table 3, and the action relations are enriched as shown in Table 4.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x\|y = x\lfloor\!\lfloor y + y\lfloor\!\lfloor x$	M1
$a\lfloor\!\lfloor x = ax$	M2
$ax\lfloor\!\lfloor y = a(x\|y)$	M3
$(x + y)\lfloor\!\lfloor z = x\lfloor\!\lfloor z + y\lfloor\!\lfloor z$	M4

Table 3. The BPA axioms, expanded with the introduction of merge ($\|$) and left merge ($\lfloor\!\lfloor$) operators.

$a \xrightarrow{a} \surd$	R1
$a \xrightarrow{a} x' \Rightarrow x + y \xrightarrow{a} x'$ and $y + x \xrightarrow{a} x'$	R2
$x \xrightarrow{a} \surd \Rightarrow x + y \xrightarrow{a} \surd$ and $y + x \xrightarrow{a} \surd$	R3
$x \xrightarrow{a} x' \Rightarrow xy \xrightarrow{a} x'y$	R4
$x \xrightarrow{a} \surd \Rightarrow xy \xrightarrow{a} y$	R5
$x \xrightarrow{a} x' \Rightarrow x\|y \xrightarrow{a} x'\|y$ and $y\|x \xrightarrow{a} y\|x'$	R6
$x \xrightarrow{a} \surd \Rightarrow x\|y \xrightarrow{a} y$ and $y\|x \xrightarrow{a} y$	R7
$x \xrightarrow{a} x' \Rightarrow x\lfloor\!\lfloor y \xrightarrow{a} x'\|y$	R8
$x \xrightarrow{a} \surd \Rightarrow x\lfloor\!\lfloor y \xrightarrow{a} y$	R9

Table 4. The action relations of BPA, expanded using the composition operators.

Two BPA processes p_1 and p_2 are *bisimilar*, if whenever p_1 performs a certain action, p_2 can perform the same action, and *vise versa*. The following definition of bisimulation equivalence for processes is quoted from [11], and is chosen only because of its conceptual association to similar definitions of bisimulation for transition systems, that have appeared in the controls literature [22].

Definition 6 **([11]).** *A binary relation $\approx$ on the set of processes* Proc *is a bisimulation, if the following conditions are satisfied:*

- *for all p, q, and p' in* Proc*, and $a \in A$ such that $p \approx q$ and $p \xrightarrow{a} p'$, there exists $q' \in \mathrm{Proc}$ such that $q \xrightarrow{a} q'$ and $p' \approx q'$.*
- *for all p, q, and q' in* Proc*, and $a \in A$ such that $p \approx q$ and $q \xrightarrow{a} q'$, there exists $p' \in \mathrm{Proc}$ such that $p \xrightarrow{a} p'$ and $q' \approx p'$.*

4 Main Results

4.1 MDLes are a special class of BPAs

The representation of an MDLe as a BPA requires an intermediate step, which is the expression of the former as a context-free grammar. We define a context-free grammar $G = (V, \eta, R, S)$ which generates MDLe $L = \{(u, \zeta) : u \in U, \zeta \in B\}$ as follows:

- $V = \{u_1, u_2, u_3, \ldots, u_n\}$ is the finite set of *variables*, one for each controller $u_i \in U$;
- $\eta = \{\nu_1, \nu_2, \ldots \nu_n\}$ is the finite set of terminals, which are the atoms of L, $\nu = (u_i, \zeta_j)$ with $u_i \in V$, and $\zeta_j \in B$;
- S is the start symbol in V;
- R is the rules by which we create the strings of L:

$$S \to U \qquad U \to UU \qquad U \to \nu \qquad U \to \emptyset \tag{2}$$

where U can be any element of V, and ν an arbitrary element of η.

We define below the push-down automaton that is equivalent to the context-free grammar described above, according to [20]. Definition 7 allows us to conveniently switch between representations.

Definition 7. *Consider a context-free grammar G defined in (2). The push-down automaton $P = (V, \eta, \Sigma, \Gamma, \delta, S, Z_0)$ where*

- *$V = \{u_1, u_2, u_3, \ldots, u_n\}$ is the set of* states*, identified with the* variables *in G;*
- *$\eta = \{\nu_1, \nu_2, \ldots \nu_n\}$ is the set of* enabled events*, identified as the terminals in G and associated with possible transitions in P;*
- *$\Sigma = V \cup \eta$ is the stack alphabet;*
- *$\Gamma : V \to \Gamma(V)$ is the event activation function that determines which enabled events can generate transitions at each state;*
- *$\delta : V \times \eta \to V$ is the transition function such that $\delta(x, \nu) \mapsto R(x, \nu) = y \in V$;*
- *$S \in V$ is the start state in G;*
- *Z_0 is the start symbol in stack;*

The range of Γ defines all active events, the ones that correspond to transitions the automaton can autonomously take. Note the distinction between η and $\Gamma(V)$: this is what enables us to capture actions the system cannot execute autonomously, but potentially can in collaboration with another system. We allow $\Gamma(V) \not\subseteq \eta$, but the transitions which the automaton can autonomously take are in $\Gamma(V) \cap \eta$. Informally, we think of the transitions associated with events in η as ones that the system has the "potential" of taking (but may not know how), and the transitions associated with events in $\Gamma(V)$ as jumps that the system "knows" how to perform but may or may not have the capability of making. The next Lemma confines MDLes to set of languages generated by a special class of context-free grammars (CFGs).

Lemma 1. *An* MDL*e is produced by a* CFG *in Greibach normal form.*

Proof. We rewrite (2) in Chomsky normal form, an intermediate stage before we arriving at the Greibach normal form. Rewriting (2) in Chomsky normal form involves a sequence of steps, in which a transformation rule is applied to the set of rules written on the left to result in the rule set depicted on the right. Let us first combine rules (2) into a single one, using the disjunction operator $|$, for compactness. On the right we give the resulting set of rules after each transformation.

$$\begin{array}{ll} U \to UU & \\ U \to \nu & \qquad\qquad U \to UU|\nu|\emptyset. \\ U \to \emptyset & \end{array}$$

Step 1: Define a new start symbol S_0 to replace S.

$$\begin{array}{ll} & S_0 \to S \\ S \to U & S \to U \\ U \to UU|\nu|\emptyset & U \to UU|\nu|\emptyset \end{array}$$

Step 2: Remove $\emptyset$ from the rules that involve variable U.

$$\begin{array}{ll} S_0 \to S & S_0 \to S|\emptyset \\ S \to U|\emptyset & S \to U \\ U \to UU|\nu & U \to UU|\nu \end{array}$$

Step 3: Eliminate the original start variable S.

$$\begin{array}{ll} S_0 \to S|\emptyset & S_0 \to \emptyset \\ S \to U & S_0 \to UU|\nu \\ U \to UU|\nu & U \to UU|\nu \end{array} \tag{3}$$

Then we translate (3) into Greibach normal form, by first eliminating left-recursion.

Step 1: Add a new rule $B \to V|VB$ to eliminate left-recursion $V \to VV$.

$$\begin{array}{ll} & S_0 \to \emptyset \\ S_0 \to \emptyset & S_0 \to UU|\nu \\ S_0 \to UU|\nu & B \to U|UB \\ U \to UU|\nu & U \to \nu|\nu B \end{array}$$

Step 2: The final step is to use the rule $V \to \nu|\nu B$ to make all the other rules start with a terminal.

$$\begin{array}{ll} S_0 \to \emptyset & \\ S_0 \to UU|\nu & S_0 \to \nu U|\nu BU|\nu|\emptyset \\ B \to U|UB & U \to \nu|\nu B \\ U \to \nu|\nu B & B \to \nu B|\nu BB|\nu \end{array} \tag{4}$$

The rule set (4) is now in Greibach normal form. □

The next Lemma states that an MDLe can be translated into a BPA in Greibach normal form [10].

Lemma 2. *The terms of an* MDLe *are a finite trace set of a normed process p, recursively defined by means of a guarded system of recursion equations in restricted Greibach normal form over a* BPA.

Proof. Lemma 1 allows us to express an MDLe as a CFG in Greibach normal form, which in addition satisfies the conditions of Notation 4.5 of [10]. We apply Notation 4.5 in conjunction with Proposition 5.2 of [10] to write the CFG of (4) as a BPA as follows. According to [10],

- If E is the system represented as a CFG in Greibach normal form, let E' denote the system represented in BPA by replacing $|$ by $+$, and $\rightarrow$ by $=$.
- Let E' be in restricted Greibach normal form over the BPA, with unique solution p. Then $\mathrm{ftr}(p)$ (the set of finite traces of p) is just the context-free language generated by E.

Applying the change of notation suggested,

$$\begin{array}{ll} S_0 \rightarrow \nu U|\nu BU|\nu|\emptyset & \begin{array}{l} S_0 = \emptyset \\ S_0 = \nu U + \nu BU + \nu \end{array} \\ U \rightarrow \nu|\nu B & U = \nu + \nu B \\ B \rightarrow \nu B|\nu BB|\nu & B = \nu B + \nu BB + \nu \end{array} \tag{5}$$

and thus we have a BPA in restricted Greibach normal form. Note that according to Definition 5, each variable string in the right hand side of (5) has length of at most two. By applying Proposition 5.2 of [10], to remove the parts of the system that do not contribute to the generation of the finite traces, we conclude that the BPA of (5) generates the strings of the original MDLe. □

4.2 Composition of MDLes

In the preceding section we distinguished between events associated to transitions a push-down automaton representing an MDLe can take autonomously, and events that cannot initiate transitions. Among the latter, there can be events that when *synchronized* with some of another push-down automaton (synchronization here implies a common interrupt function), become active and do initiate transitions. Given two push-down automata P_1 and P_2 defined according to Definition 7, we define the set $H \subseteq \eta_1 \cup \eta_2$ as the collection of events on which P_1 and P_2 should be synchronized. Set H includes those events that become active as a result of the composition of P_1 with P_2. Set H is composed of three components:

1. $(\Gamma_2 \cup \eta_1) \setminus (\Gamma_2 \cup \eta_2) \setminus (\Gamma_1 \cup \eta_1)$, (part I in Figure 1), which contains enabled events of P_1 that P_1 can now activate because of (with "knowledge" provided by) P_2;
2. $(\Gamma_1 \cup \eta_2) \setminus (\Gamma_2 \cup \eta_2) \setminus (\Gamma_1 \cup \eta_1)$, (part III in Figure1), which contains enabled events of P_2 that now become active because of (with "knowledge" provided by) P_1; and

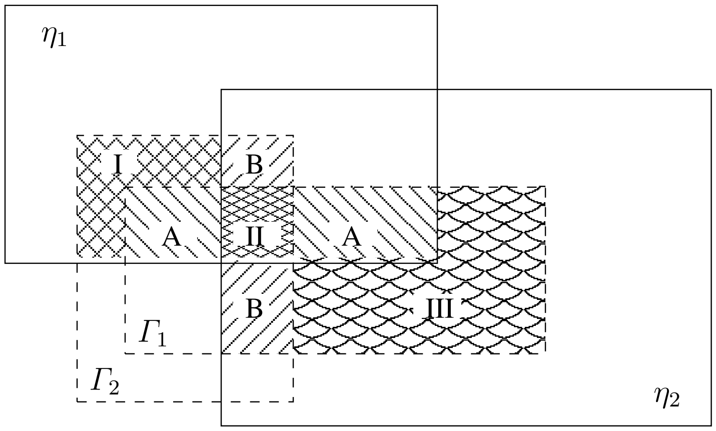

Fig. 1. Enabled, active and common events. Set A includes private active events of P_1; set B contains private active events of P_2; sets I, II, and III represent the common active events of the composed system, the ones that make up H.

3. $(\varGamma_1 \cup \eta_2) \cap (\varGamma_2 \cup \eta_1)$, (part II in Figure 1), which includes common active events in both systems.

Note that the components of H defined in 1 and 2 do not appear in the set of (active) events of the composed system under the conventional definition of composition [23]. Our definition of composition is stated as follows.

Definition 8. *Consider two* MDLe*s, expressed as context-free grammars* $G_1 = (V_1, \eta_1, R, S_{01})$ *and* $G_2 = (V_2, \eta_2, R, S_{02})$*, both with rule sets* R *of the form* (2)*. Let* E_1 *and* E_2 *be their corresponding representations as a system of guarded recursive equations, in restricted Greibach normal form over a* BPA*. The composition of* G_1 *and* G_2 *is defined as the context-free grammar* $G = (V, \eta, R, S_0)$*, where (with reference to Figure 1)*

- $V := V_1 \times V_2$*;*
- $\eta := \eta_1 \cup \eta_2$*, is the set of enabled events (also denoted* $\eta_{(1\|2)}$*);*
- $S_0 := S_{01} \times S_{02}$*;*
- $R(V \times \eta) = R((V_1, V_2) \times \eta) := \begin{cases} (R(V_1, \eta), R(V_2, \eta)) & \textit{if } \eta \in H, \\ (R(V_1, \eta), V_2) & \textit{if } \eta \in A, \\ (V_1, R(V_2, \eta)) & \textit{if } \eta \in B \\ \textit{undefined}, & \textit{otherwise.} \end{cases}$

The transitions of the composed system still respect the grammar rules (2), however, the composition restricts the domain of R. The push-down automaton representing the composed system can be defined as follows:

Definition 9. *The automaton resulting from the composition of push-down automata* P_1 *and* P_2 *that accept the strings of two different* MDLe*s is the automaton* $P_1 \| P_2 = (V, \eta_{(1\|2)}, \Sigma, \varGamma_{(1\|2)}, \delta, V_0, Z_0)$*, where (with reference to Figure 1)*

- *$V = V_1 \times V_2$ is the set of states;*
- *$\eta = \eta_{(1\|2)} = \eta_1 \cup \eta_2$ is the input alphabet;*
- *$\Sigma = (V_1 \times V_2) \cup \eta_{(1\|2)}$ is the stack alphabet;*
- *$\Gamma_{(1\|2)}((u_1, u_2)) = \Gamma_1(u_1) \cup \Gamma_2(u_2)$ is the set of inputs that may generate transitions at state (u_1, u_2),*
- $\delta(V \times \eta) = R((V_1, V_2) \times \eta) := \begin{cases} (R(V_1,\eta), R(V_2,\eta)) & \textit{if } \eta \in H, \\ (R(V_1,\eta), V_2) & \textit{if } \eta \in A, \\ (V_1, R(V_2,\eta)) & \textit{if } \eta \in B \\ \textit{undefined}, & \textit{otherwise} \end{cases}$
- *$V_0 = (V_{01} \times V_{02})$ is the set of initial (start) states;*
- *$Z_0 = (Z_{01} \times Z_{02})$ is the start symbol of the stack.*

For the composition of Definition 8 to be well defined, we need to make sure that when we compose variables and terminals of two systems in (guarded) Greibach normal form over a BPA, the result is a term that conforms to the same rules. This is the goal of the next section.

4.3 MDLes are closed under composition

The next result establishes that operation $\|\!\lfloor$ is closed.

Lemma 3. *An MDLe written as a system of guarded recursive equations in restricted Greibach normal form is closed under the left merge $\|\!\lfloor$ operator.*

Proof. Assume that G is written as a system of guarded recursive equations in restricted Greibach form, according to (4). We prove the claim by taking all merge combinations of variables in this representation, and showing that the result is a system of equations that are also guarded in restricted Greibach normal form. According to Table 3,

$$U\|\!\lfloor B = (\nu + \nu B)\|\!\lfloor B \overset{M4,M2}{=} \nu B + (\nu B)\|\!\lfloor B \overset{M3}{=} \nu B + \nu(B\|B) \overset{A3,M1}{=} \nu B + \nu(B\|\!\lfloor B)$$

$$\begin{aligned} U\|\!\lfloor S_0 &= (\nu + \nu B)\|\!\lfloor S_0 \overset{M4,M2}{=} \nu S_0 + (\nu B)\|\!\lfloor S_0 \overset{M3}{=} \nu S_0 + \nu(B\|S_0) \\ &\overset{M1}{=} \nu S_0 + \nu(B\|\!\lfloor S_0 + S_0\|\!\lfloor B) \\ B\|\!\lfloor S_0 &= (\nu + \nu BB + \nu B)\|\!\lfloor S_0 \overset{M4,M2}{=} \nu S_0 + (\nu B)\|\!\lfloor S_0 + (\nu BB)\|\!\lfloor S_0 \\ &\overset{M3,A5}{=} \nu S_0 + \nu(B\|\!\lfloor S_0 + S_0\|\!\lfloor B) + \nu(BB)\|\!\lfloor S_0 \\ &\overset{M3}{=} \nu S_0 + \nu(B\|\!\lfloor S_0 + S_0\|\!\lfloor B) + \nu(BB\|\!\lfloor S_0 + S_0\|\!\lfloor BB) \end{aligned}$$

Note that reversing the order of variables in the above merge operations yields the same type of expressions encountered above:

$$\begin{aligned} B\|\!\lfloor U &= (\nu B + \nu BB + \nu)\|\!\lfloor U = \nu U + \nu(BB\|\!\lfloor U + U\|\!\lfloor BB) + \nu(B\|\!\lfloor U + U\|\!\lfloor B) \\ S_0\|\!\lfloor U &= (\nu U + \nu BU + \nu)\|\!\lfloor U = \nu U + \nu(U\|\!\lfloor U) + \nu(BU\|\!\lfloor U + U\|\!\lfloor BU) \\ S_0\|\!\lfloor B &= (\nu U + \nu BU + \nu)\|\!\lfloor B = \nu B + \nu(U\|\!\lfloor B + B\|\!\lfloor U) + \nu(BU\|\!\lfloor B + B\|\!\lfloor BU) \end{aligned}$$

All expressions above are guarded recursive equations in restricted Greibach normal form. All variable occurrences are guarded by ν and variable products do not exceed three in length. Since the left-merge operation $\lfloor\!\lfloor$ is closed, it follows from M1 in Table 3 that $\|$ is closed too. $\square$

4.4 MDLe equivalence is decidable

Systems of guarded recursive equations enjoy nice properties in the sense that verifying the bisimulation equivalence is decidable [10].

Theorem 1 ([10]). *Let E_1, E_2 be normed systems of guarded recursion equations (over basic process algebras) in restricted Greibach normal form. Then the bisimulation relation $\approx$, that is whether $E_1 \approx E_1$, is decidable.*

Theorem 1 allows us to conclude that

Corollary 1. *If* MDLe*s are written in the form of a system of guarded recursive equations in Greibach normal form over a* BPA*, the bisimulation relation is decidable.*

Proof. Using Lemma 1, each MDLe is written as a context-free language in Greibach Normal Form. Lemma 2 translates this representation into a system of guarded recursive equations in restricted Greibach normal form over a BPA. By Theorem 1 of [10], language equivalence for systems in (guarded) restricted Greibach normal form such as the MDLes translated using Lemma 2, is decidable up to bisimilarity. $\square$

A natural question that arises next, is whether the composition operator preserves bisimilarity: do we loose this property when we expand the basic process algebra system by including the operators $\|$ and $\lfloor\!\lfloor$ to arrive at the system the semantics of which are described in Tables 3 and 4? The following section ensures us that we do not.

4.5 MDLe composition preserves bisimilarity

Proposition 1. *The composition operator $\|$ preserves bisimilarity. That is, if $P \approx Q$, then $P\|R \approx Q\|R$.*

Proof. Consider a relation $\mathcal{R}$ over the set of processes, such that $P\|R$ and $Q\|R$ belong to $\mathcal{R}$ whenever $P \approx Q$. We show that $\mathcal{R}$ is a bisimulation.

Case 1. Process P (or Q) executes action a. If $P \approx Q$, then $(P\|R, Q\|R) \in \mathcal{R}$. Assume that $P \xrightarrow{a} P'$. Then by action relation R6 in Table 4, we have $P \xrightarrow{a} P' \Rightarrow P\|R \xrightarrow{a} P'\|R$. Since $P \approx Q$, there exists Q' such that $Q \xrightarrow{a} Q'$, and $P' \approx Q'$. By definition, $(P'\|R, Q'\|R) \in \mathcal{R}$. Similarly, it can be shown that if $Q \xrightarrow{a} Q'$, then there exists a P', with $P' \approx Q'$ and $(P'\|R, Q'\|R) \in \mathcal{R}$.

Case 2. Process R executes action a. Since bisimulation is reflexive, this case reduces to the previous one, and $(P\|R, P\|R) \in \mathcal{R}$.

Case 3. Process P terminates after executing action a ($P \xrightarrow{a} \surd$). Relation R7 of Table 4 implies that $P \xrightarrow{a} \surd \Rightarrow P\|R \xrightarrow{a} R$. Since $P \approx Q$, we need to have $Q \xrightarrow{a} \surd$. Thus, by R7 of Table 4, $Q\|R \xrightarrow{a} R$. By definition, $R \approx R$ and thus the processes derived with the a-transition belong $\mathcal{R}$. The case where Q terminates after executing a is identical.

Case 4. Process R terminates after executing a ($R \xrightarrow{a} \surd$). By R7 of Table 4, $R \xrightarrow{a} \surd \Rightarrow P\|R \xrightarrow{a} P$. Similarly, $R \xrightarrow{a} \surd \Rightarrow Q\|R \xrightarrow{a} Q$. Given that $P \approx Q$, the processes derived from $P\|R$ and $Q\|R$ when R executes a, belong to $\mathcal{R}$.

Case 5. Processes P and R are synchronously execute action a. In this case, we resort to axiom M1 of Table 3, and treat the transitions of P and R separately according to cases 1 and 2 above. The case where Q executes a synchronously with R is identical.

Case 6. Processes P and R terminate synchronously by executing action a. Axiom M1 of Table 3 allows us to treat the synchronous transition to termination as an asynchronous one. In this case, we proceed according to cases 3 and 4.

Thus, for all combinations of possible transitions for $P\|R$ and $Q\|R$, we have that $P\|R \approx Q\|R$ if $P \approx Q$. The conditions of Definition 6 are satisfied and therefore $\mathcal{R}$ is a bisimulation relation. □

From Proposition 1 it follows that

Corollary 2. *The composition of* MDL*es is decidable up to bisimulation equivalence.*

Proof. The operation $'\|'$ is closed (Lemma 3) and also preserve bisimilarity (Lemma 1), which means the composition of MDLes can also be written as a system of guarded recursive equations in restricted Greibach normal form over a BPA. By Theorem 1, this composition is decidable. □

5 A Case Study: the Sliding Block Puzzle

Representing an instance of the sliding block puzzle as a multi-robot hybrid system serves as a reality check, to ensure that our formulation captures the possible interaction between heterogeneous robot systems. In a general sliding puzzle puzzle, the challenge is to slide blocks on a flat surface with the purpose of achieving a desired configuration. No block can be removed from the board. Quoting Gardiner [24]

> These puzzles are very much in what of a theory. Short of trial and error, no one knows how to determine if a given state is to obtainable from another given state, and if it is obtainable, no one knows how to find the minimum chain of moves for achieving the desired state.

It has been shown that in general, sliding-block puzzles are PSPACE-complete [25, 26]. However, under certain simplifying assumptions and for cases of such puzzles like the one we consider here (Figure 2), a polynomial algorithm can be constructed to move a single block from any initial position to any final position [26].

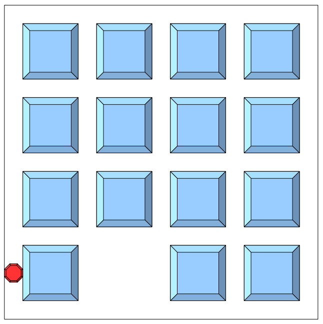

Fig. 2. Realization of a sliding block puzzle. Square blocks (tiles) cover all but one cell of a 4×4 grid. A robot (round object) is moving along the rows and columns of the grid reconfiguring the blocks. Blocks and robot are modeled as agents moving according to their own MDLe.

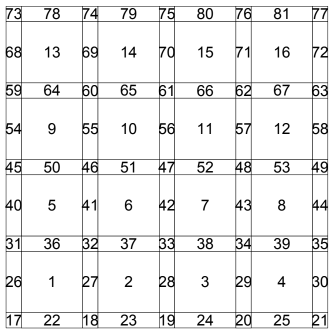

Fig. 3. Enumeration of agent positions for the agents in the sliding block puzzle. Positions 1 through 16 can be occupied by blocks. (In Figure 2, position 2 is not occupied.) Positions 17 through 81 represent possible positions for the robot agent.

In the simple instance of the sliding block puzzle depicted in Figure 2, the goal is for the robot (initially at position 26) to move the block at position 1 to location 6. Robot and blocks are thought to be autonomous agents, each with its own MDLe. A block can do nothing by itself; any transitions within the block's MDLe may only be activated after composition with the robot agent, which can *push* a block to a different location. However, these potential transitions in the block's configuration need to be encoded in its enabled event set η.

For a block to be able to make a transition (which is synchronized with a corresponding on in the robot's event set), the destination location must be unoccupied; thus blocks need to keep track of whether nearby locations are occupied. We therefore model the state of the block as a triplet, consisting of the state of motion (the analogous of the controller in a robotic system), its position, and the availability of an empty location in the immediate neighborhood. The block automaton is $B = (V_b, \eta_b, V_b \cup \eta_b, \varGamma_b, \delta_b, V_{0b}, Z_{0b})$, where

1. $V_b := \{V_{b1}, V_{b2}, V_{b3}\}$ is the set of states, where
   - $V_{b1} \in \{u_1, \dots, u_5\}$ is a motion state: u_1 (be pushed east), u_2 (be pushed west), u_3 (be pushed north), u_4 (be pushed south), u_5 (stay at location);
   - $V_{b2} \in \{1, \dots, 16\}$ is the position of the block; and
   - $V_{b3} \in \{b_1, \dots, b_5\}$ are possible empty nearby locations: b_1 (east), b_2 (west), b_3 (north), b_4 (south), b_5 (none);
2. $\eta_b = \{\nu_b \mid \nu_b = ((u_i, j, b_k), \xi)\}$, with i and k in $\{1, \dots 5\}$, and j in $\{1, \dots, 16\}$, includes all events (MDLe atoms; ξ_b is the block's interrupt function) ;
3. $\varGamma_b : V_b \to 2^{\eta_b}$ is the event activation function (initially mapping to $\emptyset$);

4. $\delta_b : V_b \times \eta_b \rightarrow V_b$ is the transition function, also mapping to $\emptyset$ since the range of Γ_b is empty, suggesting that the block automaton can make no transitions on its own (except for the case of u_5).

Symbols V_{0b} and Z_{0b} correspond to the initial state and stack symbol, respectively.

For the robot, an atom consists of the state of motion (controller running) and its position. The robot can move along the rows and columns of the grid, and push against a block in order to move it. The automaton for the robot is a tuple $R = (V_r, \eta_r, V_r \cup \eta_r, \Gamma_r, \delta_r, V_{0r}, Z_{0r})$, where

1. $V_r = \{(V_{r1}, V_{r2})\}$ is the set of states, where
   - $V_{r1} \in \{w_1, \ldots, w_9\}$ are the available controllers for the robot: w_1 (push east) w_2 (push west), w_3 (push north), w_4 (push south), w_5 (stay at location), w_6 (move east), w_7 (move west), w_8 (move north), w_9 (move south); and
   - $V_{r2} \in \{17, \ldots, 81\}$ are the possible positions for the robot;
2. $\eta_r = \{\nu_r \mid \nu_r = ((w_i, j), \xi_r)\}$, where i is in $\{1, \ldots, 9\}$, j in $\{17, \ldots, 81\}$, and ξ_r is the robot's interrupt function, includes all the events associated with possible robot transitions;
3. $\Gamma_r : V_r \rightarrow 2^{\eta_r}$ is the activation function determining which events are active at each robot state; and
4. $\delta_r : V_r \times \eta_r \rightarrow V_r$ is the transition function.

Similarly, V_{0r} and Z_{0r} are the initial state and the start stack symbol for the robot automaton, respectively.

The system expressing all possible transitions in the sliding block puzzle is generated by composing the robot with the fifteen blocks. Note that traditional notions of (parallel) composition [23] produce a system where nothing can happen (the puzzle configuration cannot change). However, by identifying "pushing" events in both systems as common: $u_i = w_i$, for $i = 1, \ldots, 5$ and including them in $H = \{u_1, \ldots, u_5\}$, the composed system can take synchronised transitions on these events. Therefore, to move a block from position 1 to position 6, starting from the configuration shown in Figure 2 we give the composed system the following input string:

((5,27,1,1,1,1,26),2)((8,27,5,2,2,5,27),1)((8,32,2,2,2,8,27),1)((8,41,2,2,2,8,32),1)((8,46,2,2,2,8,41),1)
((5,46,2,2,2,8,46),1)((7,46,2,2,2,5,46),1)((7,50,2,2,2,7,46),1)((5,50,2,2,2,7,50),1)((8,50,2,2,2,5,50),1)
((4,50,4,5,4,5,50),2)((5,36,4,5,4,4,50),1)((6,36,5,1,3,5,36),1)((6,32,3,1,3,6,36),1)((6,37,3,1,3,6,32),1)
((6,33,3,1,3,6,37),1)((5,33,3,1,3,6,33),1)((8,33,3,1,3,5,33),1)((8,42,3,1,3,8,33),1)((5,42,3,1,3,8,42),2)
((7,42,3,1,3,5,42),1)((2,42,2,6,2,5,42),1)((5,41,2,6,2,2,42),1)((9,41,5,5,1,5,41),2)((9,32,1,5,1,9,41),1)
((9,27,1,5,1,9,32),1)((9,18,1,5,1,9,27),1)((5,18,1,5,1,9,18),1)((6,18,1,5,1,5,18),1)((6,23,1,5,1,6,18),1)
((5,23,1,5,1,6,23),1)((8,23,1,5,1,6,23),1)((3,23,3,2,3,5,23),2)((5,37,3,2,3,3,23),1)((8,37,5,6,4,5,37),1)

where due to space restrictions, we have abbreviated the composed atoms, and included only the components corresponding to the block (distinguished by its position) that physically interacts with the robot. The physical outcome of this plan is shown in Figure 4.

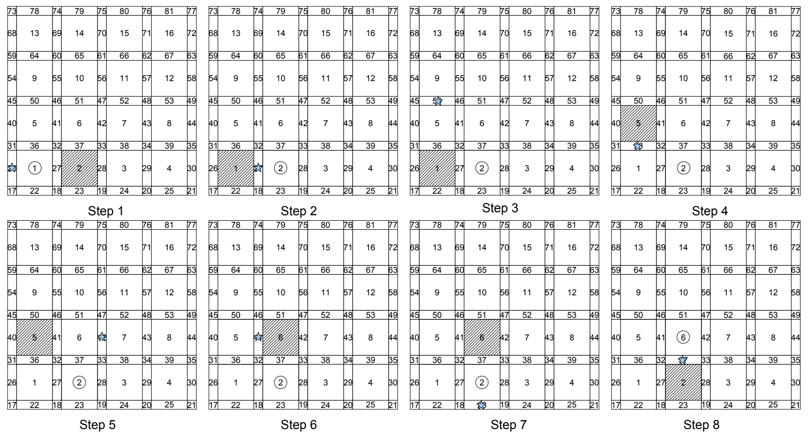

Fig. 4. Successive snapshots of the solution to the sliding puzzle problem. The robot ($\star$) moves a block ($\bigcirc$) from position 1 to position 6.

6 Conclusion

Our approach to composition of MDLes and cooperative behavior between heterogeneous systems is based on allowing systems to have additional cooperative transitions, that become active only when the systems are composed with appropriate others. We engineered the mechanics of this interaction by identifying these related, or interdependent, transitions between systems and placing them in a set H that affects how the transitions of the composed system are synchronized. By mapping MDLes to a specific type of basic process algebras we obtained well defined semantics to such compositions, and established computability properties (at least when it comes to language equivalence) for these processes and their compositions. Further steps include the construction of a bisimulation algorithm to allow us to abstract the discrete (but big) systems resulting from such compositions, and the subsequent use of available model checkers for motion and task planning by negating reachability predicates and using counterexamples.

Acknowledgement This work is supported by NSF IIS grant # 0447898.

References

1. Brockett, R.: Formal languages for motion description and map making. In Bailleul, J., Brockett, R., Donald, B., eds.: Robotics. Volume 41. ACM (1990) 181–193
2. Manikonda, V., Krishnaprasad, P., Hendler, J.: Languages, behaviors, hybrid architectures and motion control. In Baillieul, J., Willems, J.C., eds.: Mathematical Control Theory. Springer-Verlag (1998) 200–226
3. Hristu, D., Krishnaprasad, P., Anderson, S., Zhang, F., L.D'Anna, Sodre, P.: The MDLe engine: A software tool for hybrid motion control. Technical Report 2000-54, Institute for Systems Research, University of Maryland (2000)

4. Baeten, J.: A brief history of process algebra. Technical Report CSR 04-02, Vakgroep Informatica, Technische Universiteit Eindhoven (2004)
5. Frazzoli, E., Dahleh, M.A., , Feron, E.: Maneuver-based motion planning for nonlinear systems with symmetries. IEEE Trans. on Robotics **21** (December 2005) 1077–1091
6. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77** (1989) 541–580
7. Jancar, P.: Undecidability of bisimilarity for petri nets and some related problems. Theoretical Computer Science **148** (1995) 281–301
8. Hristu-Varsakelis, D., Egerstedt, M., Krishnaparsad, P.: On the structural complexity of the motion description language MDLe. In: Proceedings of the 42nd IEEE Conference on Descision and Control. (2003) 3360–3365
9. Burkart, O., Steffen, B.: Composition, decomposition and model checking of pushdown processes. Nordic Journal of Computing **158** (1995) 89–125
10. Baeten, J., Bergstra, J., J.W.Klop: Decidability of bisimulation equivalence for process generating context-free languages. Journal of the ACM **40** (1993) 653–683
11. Hirshfeld, Y., Jerrum, M., Moller, F.: A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel process. Theoretical Computer Science **158** (1996) 143–159
12. Milner, R.: A Calculus of Communicating Systems. Volume 42 of Lecture Notes in Computer Sciences. Springer-Verlag (1980)
13. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
14. Hoare, C.: Communicating Sequential Processes. Lecture Notes in Computer Sciences. Prentice-Hall (1985)
15. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Volume 18 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (1991)
16. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: UPPAAL - a tool suite for automatic verification of real-time systems. In: DIMACS Workshop on Verification and Control of Hybrid Systems. Springer Verlag (1995)
17. Henzinger, T., Ho, P.H., Wong-Toi, H.: A user guide to HYTECH. In Brinksma, E., Cleaveland, W., Larsen, K., Margaria, T., Steffen, B., eds.: TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems. Volume 1019 of Lecture Notes in Computer Science 1019. Springer-Verlag (1995) 41–71
18. Daws, C., Olivero, A., Tripakis, S., Yovine, S.: The tool KRONOS. In: Hybrid Systems III. Volume 1066 of Lecture Notes in Computer Science. Springer-Verlag (1996) 208–219
19. Brockett, R.W.: On the computer control of movement. In: Proceedings of the IEEE International Conference on Robotics and Automation. (1988) 534–540
20. Sipser, M.: Introduction to the Theory of Computation. PWS Publishing Company (1997)
21. Hirshfeld, Y., Jerrum, M.: Bisimulation equivalence is decidable for normed process algebra. Technical Report ECS-LFCS-98-386, School of Informatics at the University of Edinburgh (1998)
22. Pappas, G.J.: Bisimilar linear systems. Automatica **39** (2003) 2035–2047
23. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Kluwer Academic (2001)
24. Gardner, M.: The hypnotic fascination of sliding-block puzzles. Scientific American **210** (1964) 122–130
25. Hopcroft, J., Schwarz, J., Sharir., M.: On the complexity of motion planning for multiple independent objects:pspace-hardness of the 'warehouseman's problem. International Journal of Robotics Tesearch **3** (1984) 76–88
26. Robert A. Hearn, E.D.D.: Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theoretical Computer Science **343** (October 2005) 72–96