On Model Checking Boolean BI

On Model Checking Boolean BI

Heng Guo, Hanpin Wang, Zhongyuan Xu, and Yongzhi Cao

Key Laboratory of High Confidence Software Technologies, Ministry of Education,
Institute of Software, School of Electronics Engineering and Computer Science,
Peking University, Beijing, China
{guoheng,whpxhy,xzy,caoyz}@pku.edu.cn

Abstract. The logic of bunched implications (BI), introduced by O'Hearn and Pym, is a substructural logic which freely combines additive and multiplicative implications. Boolean BI (BBI) denotes BI with classical interpretation of additives and its model is the commutative monoid. We show that when the monoid is finitely generated and propositions are recursively defined, or the monoid is infinitely generated and propositions are restricted to generator propositions, the model checking problem is undecidable. In the case of finitely related monoid and generator propositions, the model checking problem is EXPSPACE-complete.

1 Introduction

The logic of bunched implications (BI), introduced by O'Hearn and Pym [26], is a substructural logic which freely combines additive and multiplicative implications. The main purpose of BI is to reason about models which incorporate the notion of resource. Its best-known application in computer science is separation logic, which is a Hoare logic for reasoning about imperative program which can manipulate pointers [29]. Several other similar resource logics are developed, such as spatial logic [9, 10], which is introduced independently, and context logic [6, 7], which can be regarded as a generalization of BI and spatial logic.

Semantically, the models of BI vary from cartesian doubly closed categories, to partially ordered commutative monoids [27]. The interpretation of BI in the categorical models is necessarily intuitionistic. But the additive connectives can be interpreted classically in the monoid models, in which the partial order becomes an equivalence relation. This version of BI is called Boolean BI (BBI). Its expressivity is quite powerful [18], and has been shown to be a convenient way to characterize resource sensitive systems. Indeed, separation logic is interpreted on a partial monoid of heaps, which is a BBI model. Also the classical additives alongside multiplicative conjunction and implication could be found in spatial logic and context logic.

In this paper, we mainly discuss the model checking problem of BBI, *i.e.* given an element in a BBI model and a BI formula, decide whether the element satisfies the formula. It is shown that the provability of BI can be decided by Resource Tablueax (*cf.* [19]). However, this method cannot be applied to the model checking problem directly.

To give a decidable condition, a notion of boundable resource model is introduced in [2]. It is proved that the model checking problem in a boundable model is decidable.

However, this condition is given implicitly. To decide whether a model is boundable, one need to check whether the quotient of the monoid divided by some equivalence relations is finite. It is quite hard (and possibly undecidable) to verify this since there are infinitely many such relations. Compared with their work, in this paper, we show more explicit decidable conditions. To our best knowledge, there is not any other work concerning the model checking problem for BI.

In separation logic, all products are obtained from two heaps with disjoint domains (*cf.* [29]). But in general BBI model, the structure is less organized, since the generation relation can be defined arbitrarily. Hence, we expect that the model checking problem for BBI is quite complicated and would be decidable only under many restrictions.

First, we consider the propositions appeared in BI formulae. In a BBI model, a proposition variable is interpreted by the set composed of elements satisfying the formula. Obviously, when such a set is not recursive, we cannot check whether an element satisfies this proposition. However, even if propositions are recursively defined, we show that the model checking problem is undecidable for finitely generated free commutative monoid, somehow the simplest model, by reduction from *Hilbert 10th problem*.

Inspired by the fact that in separation logic atomic assertions are interpreted on one heap cell, we focus our attention on propositions that is satisfied by only one generator. But even with this restriction, the model checking problem is still undecidable in infinitely generated commutative monoid. The technique we use is to simulate *Minsky Machine* (*cf.* [25]), which is a classical and widely adopted method, like in the proof of the undecidability of full propositional linear logic [23] and the bisimulation relation between petri nets [20]. We should mention that although total monoid is a special case of partial monoid, the model checking problem for quantifier free assertion in separation logic, which is interpreted on an infinitely generated partial monoid, in contrast, is decidable [8].

To obtain decidable results, we put some additional restrictions on the model. We consider the case that the monoid is finitely generated. A special case of the model checking problem in this setting is equivalent to the word problem in monoids, which is shown to be EXPSPACE-complete in finitely generated commutative monoids (*cf.* [24]). This result sheds some light to the general problem and provides a complexity lower bound. With the help of some results from [21] and [16], we reduce the model checking problem to the problem of deciding whether two semi-linear sets overlap, which can be solved with the cost of at most exponential space. It follows that the in this case, the model checking problem is EXPSPACE-complete.

Furthermore, in the case of infinitely generated finitely related monoid, the model checking problem can be reduced to the finitely generated case. Indeed, every finitely generated monoid is finitely related (*cf.* [17]). Thus, for all finitely related monoid, the model checking problem is EXPSPACE-complete.

Model checking and validity problems for the spatial assertion language of separation logic are solved in [8]. Several decidable fragments are discovered and discussed [1, 3]. In comparison, the model we considered is more general and its structure can be more chaotic, and the formula is propositional. Thus, both our decidability and unde-

cidability results are essentially different from those of separation logic. It is easy to extend our complexity results to the case of partial monoid.

Interesting aspect of our undecidability proof is the explicit way of simulating Minsky Machine, which can be viewed as a sign of the strong expressivity of BBI and did not appear in the literature before. The technique used in our decidability proof reveals the relationship among BI formulae, rational sets in monoid and regular expressions. It suggests a way to extend BI and provide the possibility to apply classical algebraic results on the further analysis of BI or BBI.

In Section 2 we review some basic definitions and notations of BBI and semigroups. In Section 3 we show undecidability results, and Section 4 decidability and complexity results. Finally, in Section 5, some additional remarks are provided. For brevity, some of the proofs are omitted.

2 Preliminaries

We start with some basic definitions and denotations.

2.1 Boolean BI

In BI, there are additive connectives of classical propositional logic ($\neg$,$\vee$,$\wedge$,$\rightarrow$,$\top$,$\bot$) and multiplicative connectives ($*$,$-\!\!*$,$\top^*$).

Definition 1 (BI formula). *The set of BI formulae, denoted $\mathcal{BI}$ and ranged over by φ,φ_1,φ_2, is defined by:*

$$\varphi \;=\; p \mid \top \mid \bot \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \top^* \mid \varphi_1 * \varphi_2 \mid \varphi_1 -\!\!* \varphi_2$$

in which p ranges over P, the set of atomic propositions.

Definition 2 (BBI Model). *A BBI model $\mathcal{M}$ is a commutative monoid $\{M, \varepsilon, \circ\}$ (denoted M, for brevity), in which $\circ$ is the multiplication and ε its unit.*

Henceforth monoid will be used to denote commutative monoid, if not explicitly stated.

An environment mapping is needed to interpret proposition variables. The image of a proposition variable is the largest set in which every element satisfies it. By $\mathcal{P}(M)$ we denote the power set of M.

Definition 3 (Environment). *An environment ρ_M, or ρ for short, is a function $\rho : P \rightarrow \mathcal{P}(M)$.*

Definition 4 (Satisfaction Relation). *The satisfaction relation for BBI is defined inductively on the structure of the formulae as follows:*

$$\begin{aligned}
m &\models p \Leftrightarrow m \in \rho(p)\\
m &\models \top \Leftrightarrow \textit{always}\\
m &\models \neg\varphi \Leftrightarrow m \not\models \varphi\\
m &\models \varphi_1 \wedge \varphi_2 \Leftrightarrow m \models \varphi_1 \textit{ and } m \models \varphi_2\\
m &\models \top^* \Leftrightarrow m = \varepsilon\\
m &\models \varphi_1 * \varphi_2 \Leftrightarrow \exists m_1, m_2.\ m = m_1 \circ m_2 \ \textit{ s.t. } m_1 \models \varphi_1 \textit{ and } m_2 \models \varphi_2\\
m &\models \varphi_1 -\!\!* \varphi_2 \Leftrightarrow \forall m_1.\ m_1 \models \varphi_1.\ \textit{implies } m_1 \circ m \models \varphi_2
\end{aligned}$$

Since the additive connectives are interpreted classically, we treat $\bot$, $\vee$, and $\rightarrow$ as usual abbreviations. Define $\varphi_1 -\!\!*^{\exists} \varphi_2 = \neg(\varphi_1 -\!\!* \neg\varphi_2)$. Then $m \models \varphi_1 -\!\!*^{\exists} \varphi_2$ *iff* $\exists m_1.\ m_1 \models \varphi_1$ and $m_1 \circ m \models \varphi_2$. It is a existence analogue of multiplicative implication.

Sometimes we use $\models_M$ to emphasize the underlying model for the satisfaction relation.

We naturally extend the domain of environment function ρ from the set of propositions P to the set of BI formulae $\mathcal{BI}$, $\rho : \mathcal{BI} \rightarrow \mathcal{P}(M)$, $\rho(\varphi) = \{m \mid m \in M \wedge m \models \varphi\}$. Thus, the model checking problem of deciding whether $m \models \varphi$, is equivalent to the problem of deciding whether $m \in \rho(\varphi)$.

For $M_1, M_2 \subseteq M$, define:

$$M_1 \circ M_2 = \{m \mid \exists m_1, m_2.\ m = m_1 \circ m_2 \ \wedge\ m_1 \in M_1 \wedge m_2 \in M_2\}$$
$$M_1 : M_2 = \{m \mid \exists m_1.\ m_1 \in M_2 \wedge m \circ m_1 \models M_1\}$$

Thus, we get:

$$\rho(\varphi_1 * \varphi_2) = \rho(\varphi_1) \circ \rho(\varphi_2)$$
$$\rho(\varphi_1 -\!\!*^{\exists} \varphi_2) = \rho(\varphi_2) : \rho(\varphi_1)$$

Note that $\varphi_1 -\!\!* \varphi_2 = \neg(\varphi_1 -\!\!*^{\exists} \neg\varphi_2)$. In the following, we will use $-\!\!*^{\exists}$ when inducting on the structure of a formula.

Remarks In the general semantics of BI, partial monoid, rather than total monoid, is more widely adopted, since it is complete and reflects some intrinsic properties of resources. Indeed, the heap model of separation logic is a partial monoid. However, there is one way to transform a partial model into a total model. Define a special element π, which does not satisfy any formulae, and let any undefined product of two elements, or the product of π and any other element equal π. Then we get a total monoid and only need to pay some attention for such π when handling the model checking problem. Hence, for the simplicity of analysis, we adopt the notion of total monoid in this paper.

2.2 Semigroup Presentation

Since BBI models are monoids, we need the way to describe a monoid. We assume the reader is familiar with some basic notions and results.

Let X be a set of generators or so-called alphabet and X^* denotes the *free monoid* generated by X. A relation C on X^* is a *congruence* if it is an equivalence relation and whenever $(v, w) \in C$ then $(v + u, w + u) \in C$. For every relation R, it generates a congruence $\equiv_R$, which is the smallest congruence contains R.

If a semigroup $M \cong X^*/\equiv_R$, then the tuple $(X; R)$ is called a presentation of M. For a little abuse of language, we write $M = (X; R)$. M is *finitely generated* (f.g.) if there exists a presentation $(X; R)$ of M and X is finite, and is *finitely related* (f.r.) if finite R exists. By Redei's theorem (*cf.* [28, 14], also [17]), every f.g. commutative semigroup is f.r. . For a f.g. monoid $M = (X; R)$, every element m in M is a congruence class in X^*, denoted by $[m]_R$, or $[m]$ for short.

It is easy to see that a f.g. free commutative monoid is isomorphic to $\mathbb{N}^k$, assuming the cardinality of the generator set is k.

3 Undecidability Results

In this section, we show two undecidability results. For a proposition p, if $\rho(p)$ is a recursive set and the model is a f.g. free monoid, the satisfiability problem is not decidable. Hence so is the model checking problem. For a given BBI model $M = (X; R)$, if X and R is infinite, and for every $p \in P$, $\rho(p) = \{x\}$ for some $x \in X$, the model checking problem is also undecidable.

3.1 Recursively Defined Propositions

Obviously, for a proposition p, if $\rho(p)$ is not recursive, to check $m \models p$ is not computable. However, even if $\rho(p)$ is recursive, the model checking problem is still not computable. Indeed, there is a recursive set that we cannot decide whether it is empty. We will illustrate this using the result of *Hilbert 10th Problem* (H10).

Proposition 1 (Negative Solution of H10). *Given a polynomial of several variables $P(k_1, \ldots, k_m)$ with integer coefficients, it is undecidable whether there is a vector $(k_1, \ldots, k_m) \in \mathbb{N}^m$[1] that $P(k_1, \ldots, k_m) = 0$.*

Thus, for any given polynomial $P(k_1, \ldots, k_m)$, let $X^* \cong \mathbb{N}^m$ and $\rho(p) = \{x_1^{e_1} \ldots x_m^{e_m} | P(e_1, \ldots, e_m) = 0\}$. Clearly $\rho(p)$ is a recursive set since we can compute $P(e_1, \ldots, e_m)$ easily. But to model checking $\varepsilon \models \top \mathrel{-\!\!*}^{\exists} p$ is equivalent to decide whether the equation $P(k_1, \ldots, k_m) = 0$ has solutions. Hence the model checking problem is undecidable.

Redei's theorem [17] tells that every f.g. monoid is f.r. . But given a recursive relation R, we cannot compute a finite relation R' that $\equiv_R$ is the same as $\equiv_{R'}$. To see this, let $R = \{(x_1^{e_1} \ldots x_n^{e_m}, \varepsilon) | P(e_1, \ldots, e_m) = 0\}$ and again H10 reduces to it. Thus, we cannot decide the structure of a f.g. monoid if the finite generation relation is not given explicitly. In the following, when a monoid $G = (X; R)$ is finitely generated, we assume that R is finite.

3.2 Infinitely generated monoid

F.g. free monoid is somehow the simplest monoid. It can be easily embedded into infinitely generated monoid and has an empty generation relation set, which is the major obstacle to the model checking problem. Since in this model to model checking BBI with recursively defined propositions is undecidable, later discussion will be restricted on a certain kind of propositions.

In most of the settings, the resource model is discrete and properties of interest can be decomposed into several atomic assertions based on a single piece of the resource. For example, in separation logic, every formula is constructed from atomic assertions interpreted on just one heap cell, like "$x \mapsto -, -$". Hence, we focus our attention on the proposition which holds only for one generator element. Given a monoid $M = (X; R)$, we call a proposition p "generator proposition", if $\rho(p) = \{x\}, x \in X$. It is an analogue

[1] In the original problem, the vector is required to be in $\mathbb{Z}^m$. Here we slightly modify the requirement and it is easy to show these two problems are equivalent.

of the assertion "$x \mapsto -, -$" in separation logic. In the following, we will use p_x to denote the proposition which holds on x.

Many propositions can be constructed via generator propositions with BI connectives. Especially, for a proposition p, if $\rho(p)$ or $M \backslash \rho(p)$ is finite, then it can be expressed through these propositions. The proposition defined in Section 3.1 cannot be expressed via them, since we cannot construct a formula to compute polynomials.

However, in an infinitely generated monoid, even if only generator propositions appear in the formula, the model checking problem is still undecidable. We show this by the reduction from the halting problem of Minsky machine [25]. This technique of encoding Minsky Machine is widely used to prove undecidability results, like the undecidability of full propositional linear logic [23] and bisimulation relation between petri nets [20].

Definition 5 (Minsky Machine). *A Minsky machine C with nonnegative counters c_1, $\ldots, c_m$ is a program*

$$1 : COMM_1; \ldots; n : COMM_n$$

where $COMM_n$ is a HALT-command and $COMM_i$ ($i \in I_{n-1}$[2]) are commands of the following two types (assuming $k, k_1, k_2 \in I_n, j \in I_m$):

1. $c_j := c_j + 1$*; goto* k
2. *if* $c_j = 0$ *then goto* k_1 *else* ($c_j := c_j - 1$*; goto* k_2)

Note that type 2 command is indeed a branch command. We call "if $c_j = 0$ then goto k_1" the zero test part and "$c_j := c_j - 1$; goto k_2" the decrease part.

Minsky machine is a deterministic computation model. During computation, current value of relating counter determines to take which branch of type-2 command. Every Minsky machine generates a corresponding sequence of executed command number, or so called a run. If the machine halts, the sequence will be finite, ended with n. Otherwise it will be infinite.

The status of a Minsky machine during the computation can be presented by a tuple $\{k, c_1, \ldots, c_m\}$, in which k is the current command line, *i.e.* next command to be executed is $COMM_k$, and $\{c_1, \ldots, c_m\}$ expresses the status of counters. The initial state is $\{1, c_1, \ldots, c_m\}$ and $\{c_1, \ldots, c_m\}$ is considered as input. The halting problem of Minsky machine is to decide with empty input, whether the program halts with empty counters. It is known that even if a Minsky machine has only two counters, the halting problem is undecidable. In the following, to construct our reduction, we encode a two-counter Minsky machine in a countably infinitely generated monoid, and express the halting property by a satisfaction relation between an element in that monoid and a BBI formula.

Given Minsky machine C with two counters c_1, c_2 and commands $COMM_i$ ($i \in I_n$). We will construct an infinitely generated monoid $M_C = (X_C; R_C)$ to simulate the execution of C. Indeed, some congruence classes are corresponding to finite runs.

The generator set X_C is composed of four parts: set Q for the current command line, A_1 and A_2 for the current status of the two counters, S for the current position in a command sequence, and a special generator $halt$.

[2] I_n denote the set $\{1, 2, \ldots, n\}$.

We let $Q = \{q_i | i \in I_n\}$. Here q_i represents that the next command is $COMM_i$. Let $A_j = \{a_{j,i} | i \in \mathbb{N}\}$ ($j \in I_2$), $a_{j,i}$ represents that the current value of counter c_j is i. In our construction, the product $q_i \circ a_{1,n} \circ a_{2,m}$ corresponds to the state tuple (i, n, m).

The set S is a little more complex. Let $I'_n = \{1', 2', \ldots, n'\}$. We use $\lambda_k \in (I_n \cup I'_{n-1})^*$ to denote a command sequence of length k and $\lambda_k[i]$ to denote the ith element in λ_k. If $\lambda_k[i] \in I_{n-1}$, then it represents the ith command in this sequence is type 1 or the decrease part of type 2, and if $\lambda_k[i] \in I'_{n-1}$, it represents zero test part of type 2. $\lambda_k[i] = n$ means the ith command is the HALT-command. We denote the command sequence generated by the Minsky Machine C by λ_C. Every element in set S has two indices, i and λ_k, denoted by s_{i,λ_k}, which means the command sequence is λ_k and the current command is $\lambda_k[i]$. It is easy to see that the cardinality of X_C is countably infinite.

Then we define the generation relation R_C. Every equation in R_C corresponds to one step execution of C. For type-1 command $COMM_m$ "$c_j := c_j + 1$; goto r" ($j \in I_2$), if $\lambda_k[i] = m$ ($i \neq k$), then R_C contains the following equations:

$$s_{i,\lambda_k} \circ q_m \circ a_{j,t} = s_{i+1,\lambda_k} \circ q_r \circ a_{j,t+1} (t \in \mathbb{N})$$

For type-2 command $COMM_m$ "if $c_j = 0$ then goto r_1 else ($c_j := c_j - 1$; goto r_2)" ($j \in I_2$), if $\lambda_k[i] = m$ ($i \neq k$), then:

$$s_{i,\lambda_k} \circ q_m \circ a_{j,t} = s_{i+1,\lambda_k} \circ q_{r_2} \circ a_{j,t-1} (t \in \mathbb{N}^+)$$

or $\lambda_k[i] = m'$ ($i \neq k$), then:

$$s_{i,\lambda_k} \circ q_m \circ a_{j,0} = s_{i+1,\lambda_k} \circ q_{r_1} \circ a_{j,0}$$

Finally, if $\lambda_k[k] = n$, then:

$$s_{k,\lambda_k} \circ q_n \circ a_{1,0} \circ a_{2,0} = s_{k,\lambda_k} \circ halt$$

Example We have finished the construction of the infinitely generated monoid $M_C = (X_C; R_C)$ corresponding to a Minsky machine C. Before proceeding to the reduction, let's give a simple example to illustrate how our construction goes. Let a Minsky machine C as following:

1. $c_1 := c_1 + 1$; goto 3.
2. $c_2 := c_2 + 1$; goto 2.
3. If $c_1 = 0$ then goto 4 else ($c_1 := c_1 - 1$; goto 2).
4. HALT.

Note that C never halts, and λ_C is the infinite sequence $\{1, 3, 2, 2, 2, \ldots\}$.

In our construction, $Q = \{q_i | i \in I_4\}$. The generation relation R_C contains: ($i \neq k$)

$$\begin{aligned}
s_{i,\lambda_k} \circ q_1 \circ a_{1,t} &= s_{i+1,\lambda_k} \circ q_3 \circ a_{1,t+1} && (\text{if } \lambda_k[i] = 1, t \in \mathbb{N}) \\
s_{i,\lambda_k} \circ q_2 \circ a_{2,t} &= s_{i+1,\lambda_k} \circ q_2 \circ a_{2,t+1} && (\text{if } \lambda_k[i] = 2, t \in \mathbb{N}) \\
s_{i,\lambda_k} \circ q_3 \circ a_{1,t} &= s_{i+1,\lambda_k} \circ q_2 \circ a_{1,t-1} && (\text{if } \lambda_k[i] = 3, t \in \mathbb{N}^+) \\
s_{i,\lambda_k} \circ q_3 \circ a_{1,0} &= s_{i+1,\lambda_k} \circ q_4 \circ a_{1,0} && (\text{if } \lambda_k[i] = 3') \\
s_{k,\lambda_k} \circ q_4 \circ a_{1,0} \circ a_{2,0} &= s_{k,\lambda_k} \circ halt && (\text{if } \lambda_k[k] = 4)
\end{aligned}$$

For $\lambda_3 = \{1', 3, 5\}$, we can see that $[s_{1,\lambda_3} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$ contains only one element, since the first command is not type 2. For $\lambda_4 = \{1, 3', 2, 2\}$, $s_{1,\lambda_4} \circ q_1 \circ a_{1,0} \circ a_{2,0} = s_{2,\lambda_4} \circ q_3 \circ a_{1,1} \circ a_{2,0}$. The congruence class contains only these two elements, since the zero test in $COMM_3$ fails. Generally, the elements contained in the congruence class $[s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$ correspond to the longest common prefix of λ_k and λ_C.

Lemma 1. *For λ_k, assume the length of the longest common prefix of λ_k and λ_C is k'. Every element in congruence class $[s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$ has the form "$s_{t,\lambda_k} \circ q_i \circ a_{1,j_1} \circ a_{2,j_2}$", where $t \leq k' + 1$ and the tuple (i, j_1, j_2) is the state after executing first $t - 1$ elements in λ_C, or "$s_{k,\lambda_k} \circ halt$", when $\lambda_k = \lambda_C$.*

Proof. It can be verified straightforwardly by induction on k'. □

Thus, the Minsky machine C halts if and only if there exists some λ_k, such that $s_{k,\lambda_k} \circ halt \in [s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$. All we left to do is to express this property through some satisfaction relation of BBI.

First define $\phi_{as} = (\neg(\neg\top^* * \neg\top^*)) \wedge (\bigwedge_i \neg p_{q_i}) \wedge (\neg p_{halt})$. For $m \models \phi_{as}$, $m \models \neg(\neg\top^* * \neg\top^*)$. Thus m cannot be a product of any two non-unit elements in M_C, it follows that $m \in X$. But $m \models (\bigwedge_i \neg p_{q_i}) \wedge (\neg p_{halt})$, it implies that $m \in S$ or $m \in A_1 \cup A_2$. The reverse obviously holds. So $\rho(\varphi_{as}) = S \cup A_1 \cup A_2$.

We claim that the Minsky machine C halts if and only if $q_1 \circ a_{1,0} \circ a_{2,0} \models \varphi$, where $\varphi = \phi_{as} \mathrel{-\!\!*}^{\exists} (p_{halt} * \phi_{as})$. Clearly, φ is constructed via generator propositions.

If $q_1 \circ a_{1,0} \circ a_{2,0} \models \varphi$ holds, then there exists some $s_1, s_2 \in S \cup A_1 \cup A_2$, that $s_2 \circ halt \in [s_1 \circ q_1 \circ a_{1,0} \circ a_{2,0}]$. If $s_1 \in A_1 \cup A_2$, then the congruence class $s_1 \circ q_1 \circ a_{1,0} \circ a_{2,0}]$ contains only one element since no generation relation can be applied, contradiction. Then $s_1 \in S$, without loss of generality we can assume that $s_1 = s_{1,\lambda_k}$ for some λ_k, since we can discard the elements before s_1 to get a new sequence. By Lemma 1, $s_2 = s_{k,\lambda_k}$, and hence C halts. On the other hand, if C halts and $|\lambda_C| = k$, then by Lemma 1, $s_{k,\lambda_C} \circ halt \in [s_{1,\lambda_C} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$. Hence $q_1 \circ a_{1,0} \circ a_{2,0} \models \varphi$ holds.

Theorem 1. *The model checking problem for countable infinitely generated monoid against BI formulae in which only generator propositions appear is not decidable.*

Remarks Since total monoid is a special case of partial monoid, our undecidability result is instantly generalized to the case of partial monoid. In separation logic, the underlying model is a partially defined countably infinitely generated monoid. However, unlike our result, the model checking problem for quantifier free assertions of separation logic is decidable. This difference results from the organized structure of the model of separation logic, whereas an arbitrary monoid could be far more chaotic.

4 Decidability and Complexity Results

In this section we show that the model checking problem for finite related monoid against BI formulae, under the restriction of generator propositions, is decidable and EXPSPACE-complete.

First we claim that for a infinitely generated finitely related monoid, the problem can be reduced to the f.g. case. In fact, there are finitely many generation relations in such a monoid. Thus only finitely many generators will appear in one congruence class. Then there will be only finitely many generators involved in a model checking problem under our assumptions. Every other generator can be treated as the same irrelevant generator.

Formally, given a monoid $M = (X; R)$, an element m, and a formula φ, where X is infinite, R is finite, and $m \in M$. Without loss of generality, we can assume that generators appeared in m and R and generators whose corresponding propositions appeared in φ are first k generators. Let δ be the homomorphism that maps every other generator to the $k+1$th generator. Then by the induction on the structure of φ, the following lemma holds:

Lemma 2. *$m \models_M \varphi$ iff $m \models_{\delta(M)} \varphi$.*

In the following, we will deal with the f.g. monoid. We prove that in this case, to check $m \models \varphi$ is equivalent to check whether $[m]_R$ and some set related to φ in X^* overlap. Indeed, $[m]_R$ or every such set is computable semi-linear set and we can decide whether their intersection is empty. To show this, first we cite some notations and results about rational sets and semi-linear sets in [16].

Definition 6 (Rational Sets). *Let M be a monoid (not necessarily be commutative). The class of rational subsets of M is the least class $\mathscr{E}$ of subsets of M satisfying the following conditions:*

1. *The empty set is in $\mathscr{E}$;*
2. *Each single element set is in $\mathscr{E}$;*
3. *If $X, Y \in \mathscr{E}$ then $X \cup Y \in \mathscr{E}$;*
4. *If $X, Y \in \mathscr{E}$ then $X \circ Y \in \mathscr{E}$;*
5. *If $X \in \mathscr{E}$ then $X^* \in \mathscr{E}$.*

Here X^* denotes the submonoid of M generated by X. Note that a f.g. monoid M itself is a rational set.

Definition 7 (Semi-linear Sets). *A subset $X = \{a\} \circ B^*$ with $a \in M$, $B \subseteq M$, and B finite, is called linear. A finite union of linear sets is called semi-linear.*

Clearly every semi-linear set is completely determined by the series of a_i and B_i. Let $A = \{a_1, \ldots, a_n\}$, $B = \{B_1, \ldots, B_n\}$. We call $(A; B)$ the closed representation of a semi-linear set.

Then we tabulate several useful results from [16].

Proposition 2. *For a f.g. commutative monoid M, A subset $X \subseteq M$ is rational iff it is semi-linear.*

Proposition 3 (Th III, Cor III.1 Cor III.4 in [16]). *If X and Y are rational subsets of a commutative monoid M, then their intersection $X \cap Y$, difference $Y \backslash X$ (hence $\overline{X} = M \backslash X$) and $Y : X$ are rational.*

Recall that $\rho(\varphi_1 * \varphi_2) = \rho(\varphi_1) \circ \rho(\varphi_2)$, $\rho(\varphi_1 -\!\!*^{\exists} \varphi_2) = \rho(\varphi_2) : \rho(\varphi_1)$. By induction, it follows that for all formulae φ, $\rho(\varphi)$ is semi-linear set.

Thus, if we can generate a closed representation of $\rho(\varphi)$ and decide whether m belongs to it, we can check $m \models \varphi$. However, we are not aware of a constructive way to generate the closed representation of $X \cap Y$, $\overline{X}$, $X \circ Y$, and $Y : X$ in the literature. We transform this problem to a corresponding problem in X^*, in order to avoid the complicated structure in M, which is caused by R.

Consider the canonical surjective morphism $\alpha : X^* \mapsto M$. It is easy to see that $\alpha^{-1}(m) = [m]_R$ and $m \in \rho(\varphi)$ is equivalent to $[m]_R \subseteq \alpha^{-1}(\rho(\varphi))$. If $\exists x, x \in [m]_R \cap \alpha^{-1}(\rho(\varphi))$, then $\alpha(x) \in \rho(\varphi)$. Thus, $[m]_R = [x]_R \subseteq \alpha^{-1}(\rho(\varphi))$. It implies that $m \in \rho(\varphi)$ is equivalent to $[m]_R \cap \alpha^{-1}(\rho(\varphi)) \neq \emptyset$.

Next we will show how to decide whether $[m]_R \cap \alpha^{-1}(\rho(\varphi)) \neq \emptyset$. Indeed, $[m]_R$ is also a semi-linear set. In [21], an algorithm has been developed to compute the closed representation of a congruence class with the cost of at most exponential space.

Proposition 4 (Th.10 in [21]). *Let f.g. monoid $M = (X; R)$, $m \in M$. There is an algorithm which generates a closed representation of $[m]_R$ using at most space $2^{c \cdot size(m,R)}$, where $c > 0$ is some constant independent of m and R.*

Thus, we can generate the representation of $[m]_R$ for every $m \in M$. In order to compute the closed representation of $\alpha^{-1}(\rho(\varphi))$, we need to make induction on the structure of φ. It is easy to verify the following lemma.

Lemma 3. *For a f.g. monoid $M = (X; R)$ and BI formulae φ, φ_1, and φ_2, the following holds:*

- $\alpha^{-1}(\rho(p_x)) = [x]_R$
- $\alpha^{-1}(\rho(\top)) = X^*$
- $\alpha^{-1}(\rho(\neg\varphi)) = \overline{\alpha^{-1}(\rho(\varphi))}$
- $\alpha^{-1}(\rho(\varphi_1 \wedge \varphi_2)) = \alpha^{-1}(\rho(\varphi_1)) \cap \alpha^{-1}(\rho(\varphi_2))$
- $\alpha^{-1}(\rho(\top^*)) = [\varepsilon]_R$
- $\alpha^{-1}(\rho(\varphi_1 * \varphi_2)) = \alpha^{-1}(\rho(\varphi_1)) \circ \alpha^{-1}(\rho(\varphi_2))$
- $\alpha^{-1}(\rho(\varphi_1 -\!\!*^{\exists} \varphi_2)) = \alpha^{-1}(\rho(\varphi_2)) : \alpha^{-1}(\rho(\varphi_1))$

Since $\alpha^{-1}(\rho(p_x)) = [x]_R$, Proposition 4 also builds up the basis of our induction. To compute the closed representations of $\overline{X}$, $X \cap Y$, $X \circ Y$, and $X : Y$, we consider the case of $\mathbb{N}^k$, since for a generator set X, $|X| = k$, X^* and $\mathbb{N}^k$ are isomorphic.

The case of generating the closed representation of $\overline{X}$ is the most complicated. We need to compute the representation of a set in which every element is not larger than any element in a given set.

Formally, define a partial order $\leq$ on $\mathbb{N}^k$. For two vectors $v = \{v_1, \ldots, v_k\}, v' = \{v'_1, \ldots, v'_k\}$ in $\mathbb{N}^k$, $v \leq v'$ iff $\forall i, v_i \leq v'_i$, and $v < v'$ iff $v \leq v'$ and $\exists i, v_i \neq v'_i$.

Lemma 4. *For a set of vectors $B = \{b_1, \ldots, b_n\}$ in $\mathbb{N}^k$, the set $m(B) = \{a | a \in \mathbb{N}^k, \forall b_i \in B, a < b_i$ or a and b_i are not comparable. $\}$ is a semi-linear set, and there is an algorithm which generates a closed representation of it.*

Proof (sketch). Consider the kth component of one element in $m(B)$. It must be sandwiched between the counterparts of two divisions of the vectors. Then the first $k-1$ components of it should be smaller than or not comparable with the counterparts of the smaller division. These elements compose an instance of lower dimension. Thus, the closed representation can be generated inductively on the dimension k. □

Lemma 5. *For two semi-linear sets $X, Y \subseteq \mathbb{N}^k$, given their closed representation $(A_X; B_X)$, $(A_Y; B_Y)$, there is an algorithm which generates a closed representation of $\overline{X}$, $X \cap Y$, $X + Y$, and $Y - X$.* [3]

Proof. For two semi-linear sets $X = \bigcup_i (a_i + B_i^*)$ and $Y = \bigcup_j (a_j + B_j^*)$, it is easy to see that

$$\begin{aligned} X + Y &= \textstyle\bigcup_{i,j}((a_i + B_i^*) + (a_j + B_j^*)) \\ X \cap Y &= \textstyle\bigcup_{i,j}((a_i + B_i^*) \cap (a_j + B_j^*)) \\ Y - X &= \textstyle\bigcup_{i,j}((a_j + B_j^*) - (a_i + B_i^*)) \\ \overline{X} &= \textstyle\bigcap_i (\overline{a_i + B_i^*}) \end{aligned}$$

Then we only have to deal with linear sets.

"$X + Y$": For two linear sets $a + B^*$ and $a' + B'^*$, their summation is:

$$(a + a') + (B \cup B')^*$$

"$X \cap Y$": For two linear sets $a + B^*, a' + B'^* \subseteq \mathbb{N}^k$. Assume $B = \{b_1, \ldots, b_n\}$ and $B' = \{b'_1, \ldots, b'_{n'}\}$, then every element in $X \cap Y$ corresponds to two vectors x_i, $x'_i \in \mathbb{N}^k$, which satisfies the following system of linear diophantine equations:

$$\sum_{i=1}^{n} b_i x_i - \sum_{j=1}^{n'} b'_j x'_j = a' - a$$

Indeed, the solution of this system forms a semi-linear set, and there are many algorithms devoted to this problem, e.g. [15, 22].

"$Y - X$": For two linear sets $X = a + B^*$ and $Y = a' + B'^*$, assume $B = \{b_1, \ldots, b_n\}$ and $B' = \{b'_1, \ldots, b'_{n'}\}$. We can see that

$$Y - X = \{(a' - a) + \sum_{i=1}^{n'} (t'_i b'_i) - \sum_{j=1}^{n} (t_j b_j) | t'_i, t_j \in \mathbb{N}\} \cap \mathbb{N}^k$$

It is similar to the "$X \cap Y$" case. We can get the representation after solving the system of linear diophantine equations:

$$(a' - a) + \sum_{i=1}^{n'} (t'_i b'_i) - \sum_{j=1}^{n} (t_j b_j) = \sum_{i=1}^{k} x_i e_i$$

[3] Note that the addition in $\mathbb{N}^k$ corresponds to the multiplication in free monoid. Thus, $X + Y$ and $Y - X$ correspond $X \circ Y$ and $Y : X$, respectively.

in which t'_i, t_i, x_i are variables.

"$\overline{X}$": Assume $X = a + B^*$. For $x \in \mathbb{N}^k$, if $\exists b_i \leq x$, then we can get the vector $x - b_i$ still in $\mathbb{N}^k$. Repeat this operation and finally we will get some $x' \in m(B)$. By Lemma 4, a closed representation of $m(B)$ is computable, denoted by $\bigcup_j (a_j + B_j^*)$. Thus, $\mathbb{N}^k$ is decomposed as:

$$\mathbb{N}^k = \bigcup_j (a_j + B_j^* + B^*)$$

Assume we get $a' \in m(B)$ from a, $a = a' + \sum_{i=1}^n r_i b_i$, and $a' \in a_t + B_t^*$. Such a' might not be unique but computable and the quantity is finite. Since $X \subseteq (a_t + B_t^* + B^*)$, $\overline{X}$ can be expressed as:

$$\overline{X} = \bigcup_{a \in a_t + B_t^* + B^*} ((a_t + B_t^* + B^*) \backslash X) \cup \bigcup_{j \neq t} (a_j + B_j^* + B^*)$$

$(a_t + B_t^*) \cap B^* = \emptyset$ follows that $(a_t + B_t^* + B^*) \backslash X$ contains two part: elements that do not belong to $a' + B^*$, and elements that belong to $a' + B^*$ but are smaller than a. The former can be expressed as $((a_t + B_t^*) \backslash \{a'\}) + B^*$, and the latter is finite, all semi-linear sets. It is easy to see that if there are more than one such a' belong to the same $a_t + B_t^*$, we only need to consider one of them. This concludes our argument. □

Mention that to decide whether two semi-linear sets overlap, we only need to compute their intersection, which is already solved. Thus, we have provided a way to decide whether $[m]_R \subseteq \alpha^{-1}(\rho(\varphi))$, which is equivalent to $m \models \varphi$. It follows that the model checking problem in this case is decidable.

As stated in Proposition 4, generating the closed representation of a congruence class or the set defining a proposition costs exponential space. Solving the system of linear diophantine system and other operations do not exceed the exponential space upper bound. Thus the overall space cost is at most exponential w.r.t. the length of φ and the sizes of m, R, and all propositions appeared in φ.

To get the complexity lower bound, we need to introduce the word problem of monoid. For a monoid, the word problem is to decide whether two words are in the same congruence class. In a f.g. commutative monoid $M = (X; R)$, for two words u and v, if $u = \prod_{i=1}^n x_i^{r_i}$ and $v = \prod_{i=1}^n x_i^{s_i}$ $(r_i, s_i \in \mathbb{N})$, then the word problem is equal to check whether $v \models \prod_{i=1}^n p_{x_i}^{r_i}$ or $\varepsilon \models \prod_{i=1}^n p_{x_i}^{r_i} \mathrel{-\!\!*} \prod_{i=1}^n p_{x_i}^{s_i}$. It is known that the word problem in a commutative monoid is EXPSPACE-complete (*cf.* [24]). Thus, the model checking problem is EXPSPACE-hard.

In summary, the model checking problem for f.g. monoids under our restriction of propositions is EXPSPACE-complete. Together with Lemma 2 and Redei's theorem [17], we conclude that:

Theorem 2. *The model checking problem for finitely related monoid against BI formulae in which only generator propositions appear is EXPSPACE-complete.*

Remarks If we want to do model checking in a partial monoid, first define a special element π as stated before (Section 2). Then generate $\rho(\varphi)$ normally, except that after

computing every representation of the set specified by the subformula of φ, eliminate the component corresponding to π. Compute the congruence class $[m]$ normally and it is easy to see it does not contain π. Thus the partial monoid can be model checked like total monoid.

5 Additional Remarks

In this section we provide additional remarks, along with some discussion of related works and future work.

5.1 Fragments and Complexity

It is natural to ask whether the complexity lower bound of EXPSPACE could be reduced if we only concern about some fragment of BI or for some special monoid. From our reduction, as long as the multiplicative conjunction or implication is considered, the complexity cannot be lower.

For f.g. free monoid under our restriction of propositions, the model checking problem is PSPACE-complete. Indeed, the PSPACE-hardness follows from the result in [8], since their proof of PSPACE-hardness does not employ any essential property of the predicate or the partiality. For example, we can treat $x \mapsto \mathtt{nil}, \mathtt{nil}$ as generator proposition p_x. The upper bound results from the fact that the cost of exponential space is caused by computing the congruence class, which is a singleton set in free monoid.

5.2 Automata Theory

Recall that Kleene theorem asserts that in a free commutative monoid, the class of rational set is equal to the class of set which can be recognized by finite automata. It is shown that in the case of finitely generated commutative monoid, Kleene Theorem holds in a monoid iff it is rational (*cf.* [30]). For a BI formular φ, $\rho(\varphi)$ is a rational set in f.g. monoid. Thus $\rho(\varphi)$ is recognizable by finite automata in a rational monoid. If we extend BI with a new connective to characterize the set X^* in the monoid, then a set S is rational iff there is a φ that $S = \rho(\varphi)$. Hence the language generated by this kind of BI formula cannot be recognizable by finite automata in non-rational monoid. As a comparison, it is shown that all languages generated by context logic formulae are recognizable by finite automata (*cf.* [5]).

5.3 Model Checking Mobile Ambient

In [12, 13, 11], it is shown that if the calculus contains repetition or the logic contains guarantee, the model checking problem is not decidable. The model they treated is a free monoid if we omit the nesting of ambients. The guarantee connective is a counterpart of multiplicative implication in BBI. The repetition is just a counterpart of the connective discussed in Section 5.2 above. Introducing such a connective in BBI does not affect the decidability of model checking problem for rational monoids, and hence free monoids. Thus, the undecidability of their problem resulted from the nesting of ambients.

5.4 Model Checking BI and CBI

Our discussion is restricted in the domain of BBI. In the general monoid model of BI, the interpretation of a formula is intuitionistic, according to a partial order, which makes the structure of the model and the model checking problem more complicated. The result showed here is just a special case. Maybe some requirement like ascending chain condition is needed to obtain a decidability result.

Another line of future work is to solve the model checking problem for Classical BI (CBI) (*cf.* [4]), which is an extension of BBI. In CBI, the model is more organized, similar as an inverse monoid or a cancellative monoid. Thus, the decidable condition might be loosened and the complexity might be reduced. Furthermore, the model adopted in CBI is relational monoid. It is a generalization of partial and total monoid and was introduced in [18]. The model checking problem in this semantic setting needs further analysis.

Acknowledgment

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2009CB320701, the National Natural Science Foundation of China under Grant No.60873061, and the National 863 Plans Projects of China under Grant No.2006AA01Z160.

References

1. J. Berdine, C. Calcagno, and P. W. O'Hearn. A decidable fragment of separation logic. In *Proc. 24th Found. of Software Tech. and Theor. Comp. Sci. (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004.
2. N. Biri and D. Galmiche. A separation logic for resource distribution. In *Proc. 23th Found. of Software Tech. and Theor. Comp. Sci. (FSTTCS'03)*, volume 2914 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 2003.
3. M. Bozga, R. Iosif, and S. Perarnau. Quantitative separation logic and programs with lists. In *Proc. 4th Int. Joint Conf. Auto. Reasoning (IJCAR'08)*, volume 5195 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2008.
4. J. Brotherston and C. Calcagno. Classical bi: a logic for reasoning about dualising resources. In *Proc. 36th ACM Symp. Principles of Prog. Lang. (POPL'09)*, pages 328–339. ACM, 2009.
5. C. Calcagno, T. Dinsdale-Young, and P. Gardner. Decidability of context logic. unpublished, available at `http://www.doc.ic.ac.uk/~ccris/ftp/decidCL.pdf`, 2008.
6. C. Calcagno, P. Gardner, and U. Zarfaty. Context logic and tree update. In *Proc. 32th ACM Symp. Principles of Prog. Lang. (POPL'05)*, pages 271–282. ACM, 2005.
7. C. Calcagno, P. Gardner, and U. Zarfaty. Context logic as modal logic: completeness and parametric inexpressivity. In *Proc. 34th ACM Symp. Principles of Prog. Lang. (POPL'07)*, pages 123–134. ACM, 2007.
8. C. Calcagno, H. Yang, and P. W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *Proc. 21th Found. of Software Tech. and Theor. Comp. Sci. (FSTTCS'01)*, volume 2245 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2001.

9. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. 27th ACM Symp. Principles of Prog. Lang. (POPL'00)*, pages 365–377, 2000.
10. L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.
11. W. Charatonik, S. Dal-Zilio, A. D. Gordon, S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. In *Proc. 4th Found. of Software Sci. and Comp. Struct. (FOSSACS'01)*, volume 2030 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2001.
12. W. Charatonik, S. Dal-Zilio, A. D. Gordon, S. Mukhopadhyay, and J.-M. Talbot. Model checking mobile ambients. *Theor. Comput. Sci.*, 308(1-3):277–331, 2003.
13. W. Charatonik and J.-M. Talbot. The decidability of model checking mobile ambients. In *Proc. 15th Comp. Sci. Logic (CSL'01)*, volume 2142 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2001.
14. A. H. Clifford and G. B. Preston. *The Algebraic Theory of Semigroups. Volume 2*. The American Mathematical Society, 1967.
15. E. Domenjoud and I. Lorraine. Solving systems of linear diophantine equations: An algebraic approach. In *Proc. 16th Math. Found. of Comp. Sci. (MFCS'91)*, volume 520 of *Lecture Notes in Computer Science*, pages 141–150. Springer, 1991.
16. S. Eilenberg and M.-P. Schutzenberger. Rational sets in commutative monoids. *J. Algebra*, 13(2):173–191, October 1969.
17. P. Freyd. Redei's finiteness theorem for commutative semigroups. *Proc. of the AMS*, 19:1003, 1968.
18. D. Galmiche and D. Larchey-Wendling. Expressivity properties of boolean bi through relational models. In *Proc. 26th Found. of Software Tech. and Theor. Comp. Sci. (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 357–368. Springer, 2006.
19. D. Galmiche, D. Méry, and D. J. Pym. Resource tableaux. In *Proc. 16th Comp. Sci. Logic (CSL'02)*, volume 2471 of *Lecture Notes in Computer Science*, pages 183–199. Springer, 2002.
20. P. Jančar. Undecidability of bisimilarity for petri nets and some related problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995.
21. U. Koppenhagen and E. W. Mayr. The complexity of the coverability, the containment, and the equivalence problems for commutative semigroups. In *Proc. 11th Symp. Fund. of Comp. Theory (FCT'97)*, volume 1279 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 1997.
22. D. Lankford. Non-negative integer basis algorithms for linear equations with integer coefficients. *J. Automated Reasoning*, 5(1):25–35, March 1989.
23. P. Lincoln, J. C. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. In *Proc. 31st Symp. Found. of Comp. Sci. (FOCS '90)*, volume II, pages 662–671. IEEE, 1990.
24. E. Mayr and A. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.*, 46(3):305–329, 12 1982.
25. M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.
26. P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.
27. D. J. Pym, P. W. O'Hearn, and H. Yang. Possible worlds and resources: the semantics of bi. *Theor. Comput. Sci.*, 315(1):257–305, 2004.
28. L. Redei. *The Theory of Finitely Generated Commutative Semigroups*. Oxford University Press, NY, 1965.
29. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. 17th IEEE Symp. Logic in Comp. Sci. (LICS'02)*, pages 55–74. IEEE Computer Society, 2002.
30. C. P. Rupert. On commutative kleene monoids. *Semigroup Forum*, 43(1):163–177, December 1991.