Models and Separation Logics for Resource Trees

N. Biri and D. Galmiche

LORIA - Université Henri Poincaré
54506 Vandœuvre-lès-Nancy Cedex, France
biri{galmiche}@loria.fr

Abstract. In this paper, we propose a new data structure, called resource tree, that is a node-labelled tree in which nodes contain resources which belong to a partial monoid. We define the resource tree model and a new separation logic (BI-Loc) that extends the Bunched Implications logic (BI) with a modality for locations. In addition we consider quantifications on locations and paths and then we study decidability by model-checking in these models and logics. Moreover, we define a language to deal with resource trees and also an assertion logic derived from BI-Loc. Then soundness and completeness issues are studied and we show how the model and its associated language can be used to manage heap structures and also permission accounting.

1 Introduction

The notion of *resource* is a basic one in many fields, including computer science. The location, ownership, distribution, access to, and consumption of resources are central concerns in the design of systems (such as networks) and also of programs which access memory and manipulate data structures (such as pointers). Recently some spatial logics have been studied, in different contexts, to describe and reason about distributed and structured resources. These logics are equipped with a specific composition and separation operator that splits terms into two subterms in order to deal with them separately but they propose different interpretations of the notion of *separation*. In spatial logics, for trees or graphs [7,10] or ambients [12], names can be shared between separated subterms (separation is only used for location in space), but in separation logics [20,23,26], this operator forces resource names in separated components to be disjoint and the composition must be partially defined.

In this context, the logic of Bunched Implications (BI), is a resource-aware logic in which additive ($\wedge$,$\rightarrow$) and multiplicative ($*$,$-\!*$) connectives cohabit, that arises as a central logic of resource (de)composition and separation [24,25]. We have recently defined a resource semantics of BI, based on partially defined monoids, and proved that this semantics, closely related to semantics of BI's pointer logic and separation logics, is complete [19]. In this setting, we aim at studying the primitive notions of *location* and *distribution* from BI and then at defining a new separation/spatial logic that includes a modality for locations in order to reason on distributed resources. In addition, we aim at defining a particular tree model associated to this logic and then at reasoning and proving properties on structures based on such models. A key point is the proposal of models that are counterparts of structures and also of logics that are enough expressive to represent data properties but sufficiently restricted to give decidability of semantic satisfaction and syntactic entailment. Recent works on tree models (and their related logics) have some limits concerning information representation, particularly the representation of complex data inside tree nodes [7]. Moreover, the choice of a resource composition that is partial or not has to be clarified, knowing that partiality enables to ensure that substructures are disjoint [11].

In this paper, we define and study a new data model, called *a resource tree*, that is a labelled tree structure in which nodes contain resources that are elements of a partially defined monoid. We show that it is an appropriate model to deal with resource distribution. In fact such a tree structure allows to represent complex information, through resources inside nodes, and also to distinguish the structure from the information it contains. For instance, we can naturally update an existing structure and also characterize inconsistent trees (those not valid according to a specification). In order to define a logic for resource trees we consider a new logic BI-Loc, that extends BI with a modality for locations. It can be viewed as both a separation and spatial logic in which the BI's multiplicatives naturally introduce the resource separation and the location modality that gather resources in some locations and introduce a spatial distribution of resources. Definitions of a resource composition that is partial and of a modality for locations in a resource-aware logic are central points in this work. Moreover, we show that the model and logic can be extended with specific quantifications on locations and paths that appear crucial for some applications. From these new definitions and results, we propose a new language dedicated to the management and transformation of resource trees. We also define an assertion logic for this language that is derived from BI-Loc. In order to illustrate the power and the interest of the model and language based on resource trees, we study how they can be used to represent and work with semi-structured data like heap structures in pointer and permission models.

In Section 2, we define the notion of resource tree and illustrate some of its specific features by showing for instance how to represent and manipulate semi-structured data as resources trees. In Section 3 we propose a related separation logic, called BI-Loc, that is an extension of BI with a modality for locations. In Section 4, we study decidability problems on resource models. We first show that BI is decidable by model-checking for partial resource monoids that satisfy specific conditions. We prove some decidability results for the resource tree model and its separation logic [4]. In order to extend expressiveness, we propose, in Section 5, some extensions of the model with quantifications on locations and paths, and then prove some (un)decidability results for various extensions. In Section 6 we define a language dedicated to resource tree management and also an assertion logic and its related axioms. The proofs of soundness and completeness are developed in Section 7 with a focus on backward axioms and on weakest preconditions. In Section 8 we analyze the frame property in this particular context. In Section 9 we show how heaps and their management can be studied through resource trees. In order to emphasize this way to represent heaps, we also consider permission accounting and show how to represent trees with permissions [5] with our resource tree approach. Finally, we give some concluding remarks on these results and develop some different perspectives.

2 Resource Trees

We aim at defining an abstract model to reason about distributed resources. Such a model must be rich enough to represent complex information and to manage information distribution. Then, we divide the space into locations and we explicitly define the resource locations.

2.1 Definitions

In this context, it seems natural to divide a location into sub-locations and them to provide a hierarchical representation of the space. In this context, a *resource tree* is defined as a finite tree with labelled nodes which contain resources which belong to a partial monoid of resources. In a resource tree, a path (list of label names) always leads to at most one unique node. More formally, a resource tree can be seen as a pair (m, f) where m is the resource contained in a root node and f is a partial function which associates resource trees to labels. For instance, $(m, (l \mapsto (m', nil)))$ corresponds to a tree which contains the resource m and a child at location l which contains m'. Let us first formalize what a partial resource monoid is.

Definition 1 (Partial resource monoid). *A* partial resource monoid $\mathcal{M} = (M, e, \times, \sqsubseteq)$ *is a preordered commutative monoid in which the composition* $\times$ *is partially defined and satisfies the conditions:*

1. *for all* $m, n, k \in M$ $[(m \times n) \times k] \downarrow$ *iff* $[m \times (n \times k)] \downarrow$
2. *for all* $m, n, k \in M$ *if* $[k \times n] \downarrow$ *and* $n \sqsubseteq m$ *then* $[k \times m] \downarrow$ *and* $k \times n \sqsubseteq k \times m$

where $[m] \downarrow$ *means that m is defined in M.*

Formally, resource trees can be defined as follows:

Definition 2 (Resource trees). *Let Loc be an enumerable set of location names and M be a set of resources, the* set of resource trees *built from M and Loc (denoted $T_{M,Loc}$) is defined as $M \times [Loc \rightharpoonup_{fin} T_{M,Loc}]$.*

Here $\rightharpoonup_{fin}$ is for finite partial functions and locations are denoted $l, l', l_1, l_2, \ldots$. A *path* is a finite sequence of locations and paths are denoted $L, L', L_1, L_2, \ldots$. The resources are denoted $m, m', m_1, m_2, \ldots$, the resource trees are denoted $t, t', t_1, t_2, \ldots$ and the concatenation of two paths L and L' is denoted $L : L'$. A tree link is here a function, denoted $f, f', f_1, f_2 \ldots$, that associates each subtree to the location name that leads to it. Then the empty tree is (e, nil), a leaf node that contains a resource m is (m, nil) and a tree with a single child t at location l is $(e, l \mapsto t)$. Another example is given by the resource tree of Figure 1 that corresponds to $(e, (l_1 \mapsto (e, l_2 \mapsto (m_1, nil)), l_3 \mapsto (m_2, nil)))$.

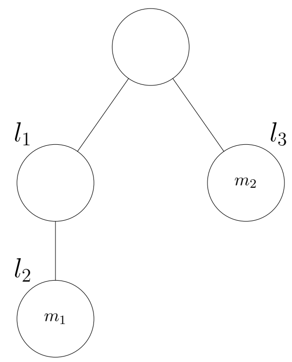


Fig. 1. A Resource Tree

Let $t = (m, f)$ be a resource tree, we use the notation $t(l)$ instead of $f(l)$ and then extend it to paths in the following way: if L is a path then $t(L)$ is the subtree located under the path L. Moreover, for a path $L = l_1, \ldots, l_n$, $(e, L \mapsto t)$ is the tree $(e, l_1 \mapsto (e, l_2 \mapsto (\ldots \mapsto (e, l_n \mapsto (t)) \ldots)))$.
Now we define particular partial monoids that are called partial resource tree monoids.

Definition 3 (Partial resource tree monoid). *Let Loc be an enumerable set of location names and $\mathcal{M} = (M, e, \times, \sqsubseteq)$ be a partial resource monoid, a* partial resource tree monoid $(T, (e, nil), |, \sqsubseteq_T)_{\mathcal{M},Loc}$ *satisfies the following conditions:*

1. *$T \subseteq T_{M,Loc}$.*
2. *$[(m, f)|(m', f')] \downarrow$ iff $[m \times m'] \downarrow$ in M and,*
   for all l, $[(f \cup f')(l)] \downarrow$ and $(m, f)|(m', f') = (m \times m', (f \cup f'))$
   where $(f \cup f')$ is defined by: for any $l \in Loc$, $(f \cup f')(l) = f(l)$ if $f'(l)$ is undefined, $= f'(l)$ if $f(l)$ is undefined and $= (f(l) \mid f'(l))$ otherwise.
3. *for all $(m, f), (m', f')$, we have $(m, f) \sqsubseteq_T (m', f')$ iff*
   i) $m \sqsubseteq m'$ and ii) for all l, $[f(l)] \downarrow$ iff $[f'(l)] \downarrow$ and if $[f(l)] \downarrow$ then $f(l) \sqsubseteq_T f'(l)$.
4. *$(T, (e, nil), |, \sqsubseteq_T)$ is a partial resource monoid.*

where $[m] \downarrow$ (resp. $[(m, f)] \downarrow$) means that m (resp. (m, f)) is defined in M (resp. $T_{M,Loc}$).

It defines the behavior of the tree composition operator $|$ that, from two trees $t = (m, f)$ and $t' = (m', f')$, composes resources at the same locations and then merges the trees to provide $(t \mid t')$.

The direct description of resource trees by a recursive domain equation is similar to the spirit to the representation of BI's heap models [20], as used in separation logics. In related work, a different style of model representation has been used, where a grammar of terms is given together with a syntactic structural equivalence notion [7]. We choose here the latter representation that provides a convenient language for writing down model elements.

Definition 4 (Resource Trees (2)). *Let Loc be an enumerable set of location names and $\mathcal{M} = (M, \times, e, \sqsubseteq)$ be a partial resource monoid, a* resource tree *over $\mathcal{M}$ (denoted T_M) is inductively defined as follows:*

$$t ::= m \mid (t \mid t) \mid [l](t) \text{ where } m \in M \text{ and } l \in Loc.$$

Moreover, for all resources $m, m' \in M$, $(m|m')$ is defined as $m \times m'$.

A node labelled with l which contains a subtree t is denoted $[l](t)$. The empty tree corresponds to the neutral element of the monoid (denoted e). For instance, $[l](m \mid [l']m')$ represents a tree with a child l containing both a resource m and a child l' that contains a resource m'. With this definition, the resource tree of Figure 1 is now denoted $([l_1][l_2]m_1 \mid [l_3]m_2)$.

We want that a unique location corresponds to a unique path and handle composition of trees with the same label on the root node, as $([l]t \mid [l]t')$, by merging nodes with the same labels. Compared to other tree models [7], a major improvement is that nodes are not only labels but can also contain information (resources). Moreover, we have a unique corresponding location for a given path and the composition is partial. There is a structural equivalence $\equiv$ between resource trees that is defined as follows:

$$\begin{array}{lll}
t \equiv t & \text{if } t \equiv t' \text{ then } t' \equiv t & \text{if } t \equiv t' \text{ and } t' \equiv t'' \text{ then } t \equiv t'' \\
(t \mid t') \equiv (t' \mid t) & (t \mid t') \mid t'' \equiv t \mid (t' \mid t'') & \text{if } t \equiv t' \text{ then } (t \mid t'') \equiv (t' \mid t'') \\
t \mid e \equiv t & ([l]t \mid [l]t') \equiv [l](t \mid t') & \text{if } t \equiv t'' \text{ then } [l]t \equiv [l]t'
\end{array}$$

The rule $([l]t \mid [l]t') \equiv [l](t \mid t')$ corresponds to the way we handle the composition of trees in case we have to compose two sibling nodes with the same label. In a nutshell, the tree composition operator $\mid$ merges nodes with the same label and composes others as the usual composition of trees. Then, the composition of two nodes with the same labels is equivalent to one node which contains resources and subtrees of these two nodes, as illustrated in Figure 2.

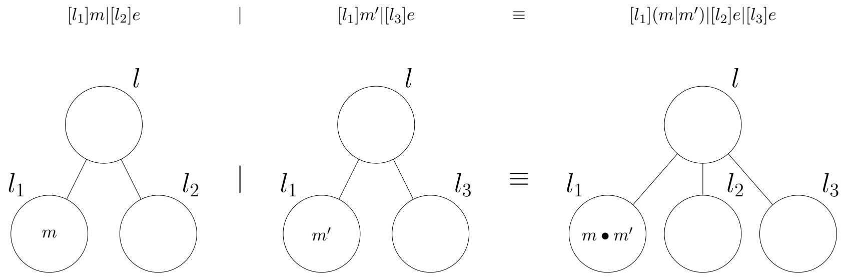


Fig. 2. Resource Tree Composition

Then we can define a partial ordering relation $\sqsubseteq_T$ on resource trees by extending the $\sqsubseteq$ relation of the resource monoid.

Definition 5. *Let t, t' be resource trees, the partial ordering relation $t \sqsubseteq_T t'$ is inductively defined by*
i) $m \sqsubseteq_T t'$ iff $t' = m'$ and $m \sqsubseteq m'$;
ii) $[l]t \sqsubseteq_T t'$ iff $t' = [l]t''$ and $t \sqsubseteq_T t''$;
iii) $(t_1 \mid t_2) \sqsubseteq_T t'$ iff $t' = (t'_1 \mid t'_2)$ with $t_1 \sqsubseteq_T t'_1$ and $t_2 \sqsubseteq_T t'_2$.

Let us note that a resource tree t defined with Definition 4 corresponds to a resource tree $\tilde{t}$ defined with Definition 2 as follows: $m \rightsquigarrow (m, nil)$, $(t \mid t') \rightsquigarrow (\tilde{t} \mid \tilde{t'})$ and $[l]t \rightsquigarrow (e, l \mapsto \tilde{t})$.
We can prove that, given two resource trees t and t' defined with Definition 4, we have $t = t'$ if and only if $\tilde{t} = \tilde{t'}$. From now we mainly consider the latter representation.

2.2 Resource Trees and Semi-structured Data

In order to illustrate how resource trees can be used to represent data and how specificities such as partiality or merging composition are important, we show the representation of semi-structured data, like XML data, as resource trees. Actually, the tree structure and the node contents are resources and the resource tree composition allows to alter tree structures in a fine way. We can compose, with the | operator, an existing tree with another one which adds a resource subtree into a specific node. However, this property entails a less intuitive representation of information. In fact, information which occurs through labels in [7] does not correspond to labels in resource trees but is represented by resources inside nodes. It provides an easier way to deal with complex information and then distinguishes the shape of data (the tree structure) from information it contains (tags and attributes). In our data, a location name only refers to a specific part of the tree structure.

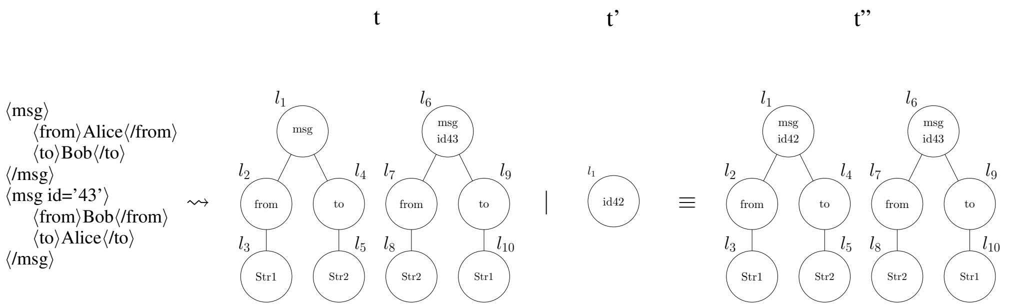


Fig. 3. Representing and Updating a XML data

In Figure 3 we show how a XML tree can be represented as a resource tree $t \equiv [l_1](msg \mid [l_2](from \mid [l_3]Str1) \mid [l_4](to \mid [l_5]Str2)) \mid [l_6](msg \mid id43 \mid [l_7](from \mid [l_8]Str1) \mid [l_9](to \mid [l_{10}]Str2))$.
Moreover we illustrate how to add an attribute (*id = '42'*) to a given node (corresponding to the first message) by composing t with a tree $t' \equiv [l_1]id42$. It results in the tree $t'' \equiv [l_1](msg \mid id42 \mid [l_2](from \mid [l_3]Str1) \mid [l_4](to \mid [l_5]Str2)) \mid [l_6](msg \mid id43 \ \mid [l_7](from \mid [l_8]Str1) \mid [l_9](to \mid [l_{10}]Str2))$.

Let us note that attributes and tags are both represented as resources. Attribute representation allows to describe pointer references (references to another part of the tree using the *id* attribute). Moreover, we can as well add a subtree at a given node instead of just one resource. Let us suppose that, instead of adding the attribute to the first message, we add it to the second one and that also, as it is the case in XML, the *id* attribute is unique, i.e., you cannot declare two *id* attributes in the same node. Consequently, the resulting

tree does not correspond to a valid XML data. The partial composition gives an easy way to treat this kind of invalid data. Actually, it is sufficient to declare that any composition of a resource representing an *id* data with another one representing another *id* data is undefined.
Compared to edge-labelled trees [7], resource trees allow representation of information inside nodes where the former can only represent tag names and tree structure. With our composition operator, we can alter an existing structure instead of just to add information next to the existing one, and then characterize inconsistent trees which are not valid according to a specification.

3 A Logic for Resource Trees

In this section, we explain why and how BI is our starting point for the design of a logic for resource trees. We extend it with a modality for locations which explicitly provides a spatial distribution of resources, following our previous approach for a modal linear logic dedicated to distribution and mobility [3,21]. We propose such a logic as associated to resource tree model, with a semantics defined by a satisfaction relation (a resource tree t satisfies the formula ϕ) and a validity relation (ϕ holds for all resource trees t).

3.1 Partial Composition and Locations

BI is a resource-aware logic in which additives ($\wedge$,$\rightarrow$), that can be classical or intuitionistic, and multiplicatives ($*$,$-\!*$) cohabit [24]. Some separation logics, based on BI connectives, have been proposed for reasoning about mutable data structures and imperative programs [20,23,26]. Spatial logics, like Ambient logic [12] and spatial logic for trees (SLT), also deal with such connectives, mainly the multiplicatives ($*$ denoted $|$ and $-\!*$ denoted $\triangleright$), and spatial modalities for locations [7,11]. It is important to notice that $*$ is considered in separation logics, as a separation connective w.r.t. partial resource composition and in spatial logics as a spatial connective w.r.t. the tree structure of processes.
The strong similarities between models of such logics and new BI models for classical or intuitionistic additives [19] that capture interactions between $*$ and $-\!*$, encourage us to start the design of a logic for resource trees from some recent results on BI: decidability of propositional BI and a based-on partial monoid semantics of BI that is complete [19]. In these models the resource composition is partial and that ensures that decomposition leads to disjoint substructures. Therefore, we extend BI with a modality for locations in the spirit of recent works on linear logic for distribution and mobility that show how locations [21] and mobility aspects [3] can be introduced in such a resource-aware logic. Therefore we add a modality, denoted $[l]$, which indicates the location l where a formula holds. In this context, the formula $[l]\phi$ means that the formula ϕ holds at location l.

3.2 A New Separation Logic

We now define the logic for resource trees, called BI-Loc, and also a based-on resource tree model for it, according to the satisfaction relation $t \models \phi$. Moreover, we show the correspondence between validity and satisfaction relation.

Definition 6. *Let Loc be a set of location names and Σ be a set of propositional variables, BI-Loc formulae are inductively defined as follows:*

$$\phi ::= p \mid I \mid \phi * \phi \mid \phi -\!* \phi \mid \top \mid \bot \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid [l]\phi$$

with $l \in Loc$ and $p \in \Sigma$.

The modality [-] is the new spatial modality. For a finite sequence of locations $L = \{l_1, \ldots, l_n\}$ we write $[L]\phi$ instead of $[l_1] \ldots [l_n]\phi$. Let us now define the relationships between this logic and the partial tree monoids.

Definition 7 (Partial Resource Tree Model). *A* partial (resource) tree model *is a resource tree monoid* $(T,(e,nil),|,\sqsubseteq_T)_{\mathcal{M},Loc}$ *with a forcing relation* $\models$ *on* $M \times \Sigma$ *that satisfies the following condition:*

$$\text{for all } p \in \Sigma,\ t,t' \in T, \text{ if } t \models p \text{ and } t' \sqsubseteq_T t \text{ then } t' \models p.$$

and extended to BI-Loc formulae as follows:

- $t \models \phi * \phi'$ *iff there exist* t', t'' *s.t.* $[(t'|t'')]\downarrow$ *and* $t \sqsubseteq_T (t'|t'')$, *with* $t' \models \phi$ *and* $t'' \models \phi'$;
- $t \models I$ *iff* $t \sqsubseteq_T (e,nil)$;
- $t \models \phi \mathrel{-\!\!*} \phi'$ *iff for all* $t' \in M$, *if* $t' \models \phi$ *and* $[(t|t')]\downarrow$ *then* $(t|t') \models \phi'$;
- $t \models \phi \wedge \phi'$ *iff* $t \models \phi$ *and* $t \models \phi'$;
- $t \models \top$ *always;*
- $t \models \phi \vee \phi'$ *iff* $t \models \phi$ *or* $t \models \phi'$;
- $t \models \bot$ *never;*
- $t \models \phi \to \phi'$ *iff for all* t', *if* $t' \sqsubseteq_T t$ *and* $t' \models \phi$ *then* $t \models \phi'$;
- $t \models [l]\phi$ *iff* $t \sqsubseteq (m, l \mapsto t')$ *with* $t' \models \phi$.

This semantics relies on the partial monoid semantics of BI [19] but it deals with resource trees instead of resources. Consequently, we extend the definition of forcing relation in order to handle such trees and the [-] modality.

Let us clarify the meaning of some formulae, especially those involving units and the $\to$ connective. First of all, it appears that $[l]\top$ (resp. $[l]I$) is not equivalent to $\top$ (resp. I). The first one indicates that there exists a child node called $[l]$ (which must be empty in the case of $[l]I$) whereas the second one does not ensure such an existence. Secondly, $\bot$ does not behave like other units. Actually, $[l]\bot$ is equivalent to $\bot$ and both cannot be satisfied by a resource tree. Finally, $[l](\phi \to \phi)$ is not equivalent to $[l]\phi \to [l]\phi$: the first one is satisfied by any resource tree which has a child node l while the second one is always satisfied.

Let us note that, with our location modality, we can define formulae that check the (non-)existence of a path or a location in a tree. Thus, to express that a tree contains a path L (the path can be restricted to a location), we write the formula $exists(L)$ defined as $exists(L) \equiv \top * [L]\top$. It means that the tree can be decomposed in two parts, one that contains anything we want and the other that contains exactly the location L, with any subtree inside. We can also express that a resource tree t does not contain a path L using the formula $no(L)$ defined as $no(L) \equiv exists(L) \to \bot$. It is easy to prove that any tree that satisfies this formula does not contain a path L. For that, let us suppose that a resource tree which contains path L satisfies this formula. As it contains the path L, it satisfies $exists(L)$ and thus also $\bot$. By contradiction, such a tree does not exist and then we deduce the result.

Definition 8 (Satisfaction). *Let* $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T,\models)_{\mathcal{M},Loc}$ *be a partial resource tree model, t be a resource tree and* ϕ *be a BI-Loc formula, t* satisfies ϕ *if and only if* $t \models \phi$.

Definition 9 (Validity for a model). *Let* $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T,\models)_{\mathcal{M},Loc}$ *be a partial resource tree model, and* ϕ *a BI-Loc formula,* ϕ *is* valid *for* $\mathcal{T}$ *iff for any resource tree* $t \in \mathcal{T}$ *we have* $t \models \phi$.

The validity on a resource model could be expressed as a satisfaction relation.

Lemma 1. *Let* ϕ *be a BI-Loc formula, we have* $\models \phi$ *iff* $e \models \top \mathrel{-\!\!*} \phi$.

Proof. Direct consequence of the satisfaction and validity definitions. We have $e \models \top \mathrel{-\!\!*} \phi$ iff for any t such that $t \models \top$, we have $t \models \phi$. By definition, for all t $t \models \top$ and then we can conclude.

Definition 10. ϕ' *is a* logical consequence *of* ϕ ($\phi \models \phi'$) *if for any model* $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T,\models)_{\mathcal{M},Loc}$ *and any* $t \in \mathcal{T}$, *if* $t \models \phi$ *then* $t \models \phi'$.

Compared to the spatial logic SLT for edge-labelled trees [7] which handles only units, our logic handles propositional letters and a location name always refers to the same location (which is neither an advantage nor a disadvantage, but just another view of locations). Our logic does not include a placement operator @ but we can use propositions and resources in order to embed the behavior of SLT locations, by adding quantifications on locations in BI-Loc, as discussed in a next section.

4 Decidability on a Resource Model

Concerning the above-mentioned calculi (ambients, trees or pointers) and their spatial or separation logics, recent works have investigated the border between decidability and undecidability of model checking for the related logic [7,9,13]. Let us recall that the model-checking problem consists in deciding whether a given object satisfies (is a model of) a given formula. In these logics, we observe that decidability depends on interactions between the separation connectives ($*$, $-\!\!*$), the classical ones and the ones introducing spatial modalities. One key point is that the $-\!\!*$ connective introduces an infinite quantification on trees. In order to obtain the decidability by model-checking, we must be able to bound such a quantification and to master interactions of $-\!\!*$ with other connectives. These key points have been already identified, but in a different way, in the proof of the decidability of propositional BI [19].

4.1 Deciding Validity by Model Checking in BI

A main problem is the infinite quantification introduced by the $-\!\!*$ connective. It is not directly related to the tree structure and then we start by focusing on the resource monoid and formulae of BI logic. Let us recall that Σ is the set of propositional variables and $\models$ is the forcing relation defined in BI logic. First we define some sufficient conditions on monoids in order to decide the satisfaction by model checking. The first step consists in defining the notion of *boundable resource model*.

Definition 11 (Boundable Resource Model). *A partial resource model $(M, e, \times, \sqsubseteq)$ is said* boundable *if it satisfies the following conditions:*

1. *for any $m \in M$ there exist $m_1, \ldots, m_n \in M$ such that $m \sqsubseteq m_1 \times \ldots \times m_n$ and there is no $i \in [1..n]$ such that there exist $m', m'' \in M$ with $m_i \sqsubseteq m' \times m'', m' \neq e$ and $m'' \neq e$;*
2. *for any finite $\sigma \subset \Sigma$, and any integer n, there exists a congruence relation $\cong_{\sigma,n}$ such that $M_{/\cong_{\sigma,n}}$ is finite and for all m, m' we have $m \cong_{\sigma,n} m'$ iff $m = m'$ or*
   - *(a) for all $p \in \sigma$, $m \models p$ iff $m' \models p$, and*
   - *(b) for all $r, r' \in M$, if $(r \cong_{\sigma,n} r'$ and $[m \times r]\downarrow)$ then $([m' \times r']\downarrow$ and $m \times r \cong_{\sigma,n} m' \times r')$, and*
   - *(c) for all $m_1, \ldots, m_k$ if $m \sqsubseteq m_1 \times \ldots m_k$ $(k \leq n)$ then there exist $m'_1, \ldots, m'_k$ such that $m' \sqsubseteq m'_1 \times \ldots m'_k$ and for all $i \in [1..k]$ $m_i \cong_{\sigma,0} m'_i$, and*
   - *(d) if $m \sqsubseteq e$ then $m' \sqsubseteq e$.*

The above definition does not impose to have a finite monoid but only requires that the monoid has a finite quotient for each equivalence relation. We show that, for a given BI formula, there exists a finite set of equivalence classes to work on. Moreover we define the size of a BI formula.

Definition 12. *Let ϕ be a BI formula ϕ, the size of ϕ, denoted $s(\phi)$, is inductively defined by*

- $s(I) = 1$	- $s(\top) = 0$	- $s(\phi' \vee \phi'') = max(s(\phi'), s(\phi''))$
- $s(p) = 1$	- $s(\phi' -\!\!* \phi'') = s(\phi'')$	- $s(\phi' \wedge \phi'') = max(s(\phi'), s(\phi''))$
- $s(\perp) = 0$	- $s(\phi' * \phi'') = s(\phi') + s(\phi'')$	- $s(\phi' \to \phi'') = max(s(\phi'), s(\phi''))$

The size of a formula ϕ determines the number of resource decompositions that are necessary to decide if ϕ is satisfied or not by a resource. Our goal is to show that checking if a resource m satisfies $\phi -\!\!* \psi$ corresponds to checking if $m \times m'$ satisfies ψ for a finite subset of resources m'. In the next lemma we show that the congruence relation $\cong_{\sigma,n}$ is monotone with respect to the integer n.

Lemma 2. *Let $(M,e,\times,\sqsubseteq)$ be a boundable partial model, for all $m,m' \in M$, for all n and n' and for all $\sigma,\sigma' \subset \Sigma$, if ($m \cong_{\sigma,n} m'$, $\sigma' \subseteq \sigma$ and $n' \leq n$) then $m \cong_{\sigma',n'} m'$*

Proof. Let m,m',n,n',σ,σ' such that $m \cong_{\sigma,n} m'$, $\sigma' \subseteq \sigma$ and $n' \leq n$, if $m = m'$ then the result is trivial. Otherwise we show that the four conditions of item 2 in Definition 11 are verified for $\cong_{\sigma',n'}$.

(a) As $\sigma' \subseteq \sigma$ if $p \in \sigma'$ then $p \in \sigma$. Thus, by condition $2(a)$ of $\cong_{\sigma,n}$, for all $p \in \sigma'$, $m \models p$ iff $m' \models p$.
(b) Condition $2(b)$ is satisfied for any n and σ and then with $\cong_{\sigma',n'}$.
(c) Condition $2(c)$ is proved by induction on n. For $n = 0$, $n' = 0$ and then it is trivially verified. For $n \neq 0$, if there exist $m_1,\ldots,m_k$ such that $m \sqsubseteq m_1 \times \ldots \times m_k$ with $k \leq n'$, as $n' \leq n$, we have $k \leq n$ and there exists $m' \sqsubseteq m'_1 \times \ldots \times m'_k$ such that for all $i \in [1..k]$ $m_i \cong_{\sigma,0} m'_i$, and then $m_i \cong_{\sigma',0} m'_i$.
(d) If $m \sqsubseteq e$, as $m \cong_{\sigma,n} m'$, we have $m' \sqsubseteq e$ from condition $2(d)$.

Then we deduce that $m \cong_{\sigma',n'} m'$.

The next result relates the congruence relation $\cong_{\sigma,n}$ between resources with the resource decomposition.

Lemma 3. *Let $(M,e,\times,\sqsubseteq)$ be a boundable resource model and $m,m_1,m_2 \in M$ with $m_1 \times m_2 \cong_{\sigma,n} m$, there exist n_1,n_2, such that $n_1+n_2 = n$, and m'_1,m'_2 such that $m \sqsubseteq m'_1 \times m'_2$, $m_1 \cong_{\sigma,n_1} m'_1$ and $m_2 \cong_{\sigma,n_2} m'_2$.*

Proof. As we have a boundable resource model we consider the decomposition of m_1 into atomic resources $r_1 \times \ldots \times r_{n_1}$ and the decomposition of m_2 into atomic resources $r_{n_1+1} \times \ldots \times r_{n_1+n_2}$. Then $m_1 \times m_2$ can be decomposed into atomic resources $r_1 \times \ldots \times r_{n_1} \times r_{n_1+1} \times \ldots \times r_{n_1+n_2}$. As $m_1 \times m_2 \cong_{\sigma,n} m$, we have $n_1+n_2 = n$ and by condition $2(c)$, there exist $r'_1, r'_2, \ldots r'_{n_1}, r'_{n_1+1}, \ldots r'_{n_1+n_2}$ such that $m \sqsubseteq r'_1 \times \ldots r'_{n_1} \times r'_{n_1+1} \times \ldots r'_{n_1+n_2}$ and for all i, $r_i \cong_{\sigma,0} r'_i$. As $r_i \cong_{\sigma,0} r'_i$ and r_i is an atomic resource then r_i cannot be decomposed and we have $r_i \cong_{\sigma,n} r'_i$ for all n. Consequently, by condition $2(b)$, we have $r_1 \times \ldots \times r_{n_1} \cong_{\sigma,n_1} r'_1 \times \ldots \times r'_{n_1}$ and $r_{n_1+1} \times \ldots r_{n_1+n_2} \cong_{\sigma,n_1} r'_{n_1+1} \times \ldots r'_{n_1+n_2}$. We define m'_1 as $r'_1 \times \ldots \times r'_{n_1}$ and m'_2 as $r'_{n_1+1} \times \ldots r'_{n_1+n_2}$ and we can conclude.

We also show that the congruence relation $\cong_{\sigma,n}$ between resources preserves the satisfaction relation. Let us note that σ_ϕ represents the set of propositional variables of ϕ.

Lemma 4. *If $m \cong_{\sigma_\phi,s(\phi)} m'$ and $m \models \phi$ then $m' \models \phi$.*

Proof. By structural induction on ϕ.
- $\phi \equiv p$: by definition of $\cong_{\sigma_\phi,s(\phi)}$, we have $m \models p$ iff $m \models p$.
- $\phi \equiv \psi * \psi'$: let us suppose $m \models \psi * \psi'$, there exist m_1,m_2 such that $m_1 \models \psi$ and $m_2 \models \psi'$ and $m \sqsubseteq m_1 \times m_2$. We also have $m_1 \times m_2 \cong_{\sigma_\phi,s(\phi)} m'$. By Lemma 3, there exist m'_1,m'_2 such that $m' \sqsubseteq m'_1 \times m'_2$ with $m'_1 \cong_{\sigma_\phi,s(\psi)} m_1$ and $m'_2 \cong_{\sigma_\phi,s(\psi)} m_2$. By induction hypothesis we conclude.
- $\phi \equiv I$: by definition $m \models I$ then $m \sqsubseteq e$ and from condition 2d we have $m' \sqsubseteq e$.
- $\phi \equiv \psi \mathrel{-\!\!*} \psi'$: let m_1 such that $m_1 \models \psi$ and $m \times m_1 \models \psi'$. As $m \cong_{\sigma_\phi,s(\phi)} m'$, by definition of $\cong_{\sigma_\phi,s(\phi)}$, we have $m \times m_1 \cong_{\sigma_\phi,s(\phi)} m' \times m_1$. By induction hypothesis we conclude.
- $\phi \equiv \psi \wedge \psi'$: we have $m \models \psi \wedge \psi'$ then by definition $m \models \psi$ and $m \models \psi'$. As $m \cong_{\sigma_\phi,s(\phi)} m'$, by induction hypothesis, we have $m' \models \psi$ and $m' \models \psi'$ and then $m' \models \psi \wedge \psi'$.
- $\phi \equiv \top$: we have $m \models \top$ and $m' \models \top$.
- $\phi \equiv \psi \vee \psi'$: we have $m \models \psi \vee \psi'$ then, by definition, $m \models \psi$ or $m \models \psi'$. As $m \cong_{\sigma_\phi,s(\phi)} m'$, by induction hypothesis, we have $m' \models \psi$ or $m' \models \psi'$, then $m' \models \psi \vee \psi'$.
- $\phi \equiv \bot$: $m \models \bot$ and $m' \models \bot$ are never verified.
- $\phi \equiv \psi \to \psi'$: we have $m \models \psi \to \psi'$ and by definition if $m \models \psi$ then $m \models \psi'$. As we have $m \cong_{\sigma_\phi,s(\phi)} m'$, by induction hypothesis, if $m' \models \psi$ then $m' \models \psi'$ and thus $m' \models \psi \to \psi'$.

Corollary 1. *Let m be a resource and $\phi \multimap \phi'$ be a formula, we have $m \models \phi \multimap \phi'$ iff for any $m' \in M_{/\cong_{\sigma_\phi, s(\phi)}}$ such that $[m \times m'] \downarrow$, we have $m \times m' \models \phi'$.*

Proof. Immediate consequence of Lemma 4 and condition 2 of Definition 11.

Theorem 1 (Satisfaction Decidability). *Let $\mathcal{M} = (M, \times, e, \sqsubseteq)$ be a boundable partial resource monoid, for any BI formula ϕ and any resource $m \in M$, $m \models \phi$ is decidable.*

Proof. By structural induction on ϕ.
- $\phi \equiv p, I, \top, \bot$: the result is immediate.
- $\phi \equiv \psi * \psi'$: as we consider a boundable monoid, by Definition 11, there exist $m_1, \ldots, m_n$ that are atomic resources such that $m \sqsubseteq m_1 \times \ldots \times m_n$. Consequently, to check if $m \models \phi$, it is sufficient to check if there is a decomposition of $m_1 \times \ldots \times m_n$ in two resources such that they respectively satisfy ψ and ψ'. As there is a finite number of decompositions for $m_1 \times \ldots \times m_n$, by induction hypothesis, we can decide if $m \models \phi$.
- $\phi \equiv \psi \multimap \psi'$: according to Corollary 1, it is sufficient to check that, for all $m' \in M_{/\cong_{\sigma_\phi, s(\psi)}}$, if $m \models \psi$ then $m \times m' \models \psi'$. Consequently, we have a finite set of configurations to check and by induction hypothesis deduce the result.
- $\phi \equiv \psi \wedge \psi'$: we check that $m \models \psi$ and $m \models \psi'$ and conclude by induction hypothesis.
- $\phi \equiv \psi \vee \psi'$: we check that $m \models \psi$ or $m \models \psi'$ and conclude by induction hypothesis.
- $\phi \equiv \psi \rightarrow \psi'$: we check that $m \models \psi$ or $m \models \psi'$ holds and conclude by induction hypothesis.

Corollary 2 (Validity Decidability). *Let $\mathcal{M} = (M, \times, e, \sqsubseteq)$ be a boundable partial resource monoid, for any BI formula ϕ, $\models_{\mathcal{M}} \phi$ is decidable.*

Proof. Immediate consequence of Theorem 1 using Lemma 2.

What about intuitionistic additives?
The above results are obtained with additive connectives that are classical and it is interesting to analyze what happens if we consider additive connectives that are intuitionistic.
In fact we can show that Lemma 4 is also provable with intuitionistic additives from a modification of the above proof in the case $\psi \rightarrow \psi'$. In this context we have $m \models \psi \rightarrow \psi'$ meaning that, for all n such that $m \sqsubseteq n$, if $n \models \psi$ then $n \models \psi'$. As $m \cong_{\sigma_\phi, s(\phi)} m'$, by Definition 10 (2.c), then for all n' such that $m' \sqsubseteq n'$, there exists n such that $m \sqsubseteq n$ and $n \cong_{\sigma_\phi, 0} n'$. If $n' \models \psi$ we deduce, by induction hypothesis, that $n \models \psi$. As $m \models \psi \rightarrow \psi'$ we have $n' \models \psi'$ and then $m' \models \psi \rightarrow \psi'$. Therefore, in order to verify formulae $\psi \multimap \psi'$, we can deal with a finite set of resources and then control the number of cases induced by the $\multimap$ connective.
But Theorem 1 cannot be proved with intuitionistic additives, and mainly the intuitionistic implication, because we cannot show that for all n such that $m \sqsubseteq n$, $n \models \psi$ or $n \models \psi'$ for an infinite number of n. In order to prove this result in this case it would be necessary to restrict our initial definition of a boundable resource model by fixing that for all m there exists a finite number of n such that $m \sqsubseteq n$. With such a restriction we could prove that the theorem holds for intuitionistic additives.

4.2 Validity by Model Checking in BI-Loc

In order to extend the above results for BI to BI-Loc and resource trees we must deal with the tree structure by having the ability to bound resource trees when the $\multimap$ connective is considered. Therefore, we have two main problems: (i) to restrict location names, since an infinite number of locations leads to an infinite number of trees; (ii) to bound the height of trees. In order to restrict location names, it appears necessary that partiality only concerns the resource decomposition. In this context, we restrict the study to particular partial tree monoids that are defined as follows:

Definition 13. *A partial tree monoid $\mathcal{T} = (T, (e, nil), |, \sqsubseteq_T)_{\mathcal{M}, Loc}$ is* maximally defined *if for all $t, t' \in T$, $t|t'$ is undefined iff there exists a path L such that $t(L) = m$, $t'(L) = m'$ and $m \times m'$ is undefined.*

From now Loc_ϕ denotes the finite set of location names of a BI-Loc formula ϕ and Loc_t is the set of locations in a resource tree t. Moreover $fv(\phi)$ (resp. $fv(t)$) represents the set of free variables of the formula ϕ (resp. of the tree t).

Lemma 5. *Let ϕ be a formula, t be a resource tree based on a maximally defined tree monoid and $l, l' \notin Loc_\phi$ two location names, we have $t \models \phi$ and $[t\{^l/_{l'}\}]\downarrow$ if and only if $t\{^l/_{l'}\} \models \phi$.*

Proof. By structural induction on ϕ.

Consequently, for given tree t and formula ϕ, we can replace all location names not in Loc_ϕ by only one location name.

Corollary 3. *Let ϕ be a BI-Loc formula and t be a resource tree based on a maximally defined tree monoid, there exists a resource tree t' such that $Loc_{t'} \subseteq Loc_\phi \cup \{l_\pi\}$ (with $l_\pi \notin Loc_\phi$) and $t \models \phi$ if and only if $t' \models \phi$.*

Proof. Let us define $Loc' = \{l_1, \dots, l_n\} = Loc_t - Loc_\phi$. We replace in t each location of Loc' by l_π. By Lemma 5 we have $t \models \phi$ iff $t\{^{l_\pi}/_{l_1}\}\{\dots\}\{^{l_\pi}/_{l_n}\} \models \phi$. If $t' \equiv t\{^{l_\pi}/_{l_1}\}\{\dots\}\{^{l_\pi}/_{l_n}\}$ then we also have $Loc_{t'} = Loc_\phi \cup \{l_\pi\}$.

For a given resource tree t, we note $h(t)$ its height which is defined as usual for a tree. Moreover, we define the height of a BI-Loc formula.

Definition 14 (Formula height). *Let ϕ be a BI-Loc formula, the height of ϕ, $h(\phi)$, is inductively defined as:*

- $h(I) = 1$	- $h([l]\phi') = 1 + h(\phi')$	- $h(\phi' \odot \phi'') = max(h(\phi'), h(\phi''))$
- $h(p) = 1$	- $h(\phi' \mathrel{-\!\!*} \phi'') = h(\phi'')$	$if\ \odot \in \{\wedge, \vee, \rightarrow, *\}$
- $h(\perp) = 0$	- $h(\top) = 0$	

The height of a formula is the maximum height of the trees to deal with in order to check if a formula is valid or not. We introduce the notion of *restricted tree* at height h.

Definition 15. *Let t be a resource tree, the* restricted tree at height h*, is the tree t_h such that, for any path $l_1, \dots, l_n$, we have:*
- $t_h(l_1, \dots, l_n) = t(l_1, \dots, l_n)$ if $n < h$;
- $t_h(l_1, \dots, l_n) = (r, nil)$ if there exist $r \in M$ and $t' \in T_{M,Loc}$ such that $t(l_1, \dots, l_n) = (r, t')$ and $n = h$;
- $t_h(l_1, \dots, l_n)$ is undefined if $n > h$.

This definition ensures that the tree t_h has exactly the same contents than t for all paths of length less than h and does not contain a path of length greater than h. By construction this tree is unique. Now, we prove that if we aim at checking satisfaction of a formula ϕ for a given tree t, then we can only consider the tree t_h.

Lemma 6. *Let ϕ be a BI-Loc formula, for any resource tree t, $t \models \phi$ if and only if $t_{h(\phi)} \models \phi$.*

Proof. If $h(t) \leq h(\phi)$ then $t \equiv t_h$ and the proof is straightforward.
If $h(t) > h(\phi)$ then we show the result by structural induction on ϕ.
- $\phi \equiv p$: $t \models p$ iff $t = m$ and $m \models p$. As $h(p) = 1$ and $h(t) > h(\phi)$ we have $t \neq m$ and also $t_{h(p)} \neq m$. Therefore $t \not\models p$ and $t_{h(p)} \not\models p$.
- $\phi \equiv \psi * \psi'$: for all t', t'' such that $t = (t'|t'')$, we have $t_1 = t'_{h(\psi*\psi')}$, $t_2 = t''_{h(\psi*\psi')}$ and $(t_1|t_2)$ is defined. Moreover, we have $t'_{h(\psi)} = t_{1\,h(\psi)}$ and $t''_{h(\psi')} = t_{2\,h(\psi')}$. By induction hypothesis, we have $t' \models \psi$ iff $t'_{h(\psi)} \models \psi$ iff $t_1 \models \psi$ (the same for t'' and t_2). Then for all t', t'', $(t'|t'') \models \psi * \psi'$ iff $(t_1|t_2) \models \psi * \psi'$ and we can conclude.
- $\phi \equiv I$: $t \models p$ only if $t = m$ with $m \in M$ and $e \sqsubseteq m$. As $h(I) = 1$, $t_{h(I)} = m$ iff $t = m$. Thus, $t \models p$ iff $t_{h(p)} \models p$.
- $\phi \equiv \psi \mathrel{-\!\!*} \psi'$: for all t', by induction hypothesis, $(t'|t) \models \psi'$ iff $(t'|t)_{h(\psi')} \models \psi'$. By induction hypothesis we

also have $(t'|t)_{h(\psi')} \models \psi'$ iff $(t'|t_{h(\psi')}) \models \psi'$. Then $t \models \psi -\!\!* \psi'$ iff $t_{h(\psi')} \models \psi -\!\!* \psi'$ and then $h(\psi') = h(\psi -\!\!* \psi')$ and we can conclude.
- $\phi \equiv \psi \wedge \psi'$: $t \models \psi \wedge \psi'$ iff $t \models \psi$ and $t \models \psi'$. By induction hypothesis $t \models \psi$ (resp. $t \models \psi'$) iff $t_{h(\psi)} \models \psi$ and $t_{h(\psi')} \models \psi'$. By induction hypothesis, we also have $t_{max(h(\psi),h(\psi'))} \models \psi$ (resp. $t_{max(h(\psi),h(\psi'))} \models \psi'$) iff $t_{h(\psi)} \models \psi$ (resp. $t_{h(\psi')} \models \psi'$).
- $\phi \equiv \top$: we have $t \models \top$ and $t_{h(\top)} \models \top$.
- $\phi \equiv \psi \vee \psi'$: similar to case $\psi \wedge \psi'$.
- $\phi \equiv \bot$: we have $t \not\models \bot$ and $t_{h(\bot)} \not\models \bot$.
- $\phi \equiv \psi \rightarrow \psi'$: similar to case $\psi \wedge \psi'$.

We have seen above how to bound resources, location names, and consequently the width and height of the trees we deal with, taking into account the rule $([l]m \mid [l]m') \equiv [l](m \mid m')$.

Theorem 2 (Satisfaction Decidability for BI-Loc). *Let $\mathcal{T} = (T, (e, nil), |, \sqsubseteq_T, \models)_{\mathcal{M},Loc}$ be a partial tree model in which $(T, (e, nil), |, \sqsubseteq_T)$ is a partial tree monoid maximally defined and $\mathcal{M}$ is a boundable partial resource monoid. For any BI-Loc formula ϕ and any resource tree $t \in \mathcal{T}$, $t \models \phi$ is decidable.*

Proof. By structural induction on ϕ.
The key points concern the connectives $*$ and $-\!\!*$, the other cases being straightforward.
- $\phi \equiv \psi * \psi'$: in order to check if $t \models \psi * \psi'$ is satisfied, we have to check that among all decompositions of t into two subtrees t' and t'' there exists a decomposition such that $t' \models \psi$ and $t'' \models \psi'$. As the trees contain resources of a boundable resource monoid then there exists a finite number of decompositions for t.
- $\phi \equiv \psi -\!\!* \psi'$: in order to check that $t \models \psi -\!\!* \psi'$ is satisfied, we must check that, for all t' such that $t' \models \psi$, we have $(t|t') \models \psi'$. But the number of trees satisfying ψ could be infinite. From Lemma 6, we know that any tree such that $h(t) > h(\psi -\!\!* \psi')$ behaves like its restriction to height $h(\psi -\!\!* \psi')$. Therefore, it is sufficient to check that, if for all t' such that $h(t') \leq h(\psi -\!\!* \psi')$ and $t' \models \psi$ then $(t|t') \models \psi'$. Thus we can restrict the location names by Lemma 3. Finally, we can bound the number of resources at each node by Theorem 1.

Theorem 3 (Validity Decidability for BI-Loc). *Let $\mathcal{T} = (T, (e, nil), |, \sqsubseteq_T, \models)_{\mathcal{M},Loc}$ be a partial tree model in which $(T, (e, nil), |, \sqsubseteq_T)$ is a partial tree monoid maximally defined and $\mathcal{M}$ is a boundable partial resource monoid. For any BI-Loc formula ϕ, $\mathcal{T} \models \phi$ is decidable.*

Proof. It is a direct consequence of Theorem 2 and Lemma 1.

4.3 Decidability for All Resource Models

Reasoning about validity of a formula for all resource tree models cannot be done by model checking but by theorem proving. In order to consider decidability for all resource models, we aim at starting from our previous results on theorem proving in BI logic. Recently, we have proposed calculi [16,19] to check validity of propositional BI formulae and also related decision procedures that generate countermodels from semantic structures called resource graphs [16,18]. A similar approach has been recently proposed to characterize provability in BI's pointer logic [17]. It appears that we can obtain decision procedures and decidability results for BI-Loc by extension of such calculi with new rules that handle locations and with new provability conditions.
For instance, our tableau method fo propositional BI [16] can be extended to handle locations that do not introduce infinite loops. Actually, as in the decidability by model checking, the key point for decidability and finite model property is to bound infinite tableau branches introduced by the $-\!\!*$ connective. As the location modality does not introduce any infinite quantification on trees, we can build a finite tableau for BI-Loc. Moreover, we must study if we have criteria to decide if a given branch of a tableau is closed or not. Actually, we must ensure that a location exists for some subformula. Furthermore, as the $\top$ unit is local, we can obtain a partially closed branch (branch which is not closed for all locations). These points can be

systematically handled and thus could lead to complex criterias, but we can decide if a tableau is closed or not. We do not develop this study in this paper because we aim at first focusing on BI-Loc and its extensions as an assertion logic for a language dedicated to resource tree manipulations.

5 Extending BI-Loc with Quantifications

Coming back to Figure 3 we observe that it presents a resource tree representation of a XML data, in which location names are arbitrary and do not have a particular meaning. In order to make abstraction of location names, we provide quantifications on locations in BI-Loc and then ensure that something is true at a location without giving its name. For instance, we need quantifications to state that there exists a children location where a proposition is true ($\exists_{loc}x.\phi$) and that a proposition is true for all child locations ($\forall_{loc}x.\phi$). It seems also natural to extend such quantifications to paths (location sequences) in order to ensure that something is true somewhere ($\exists_{path}x.\phi$) or everywhere ($\forall_{path}x.\phi$).

5.1 A Resource Tree Model

Consequently, the formulae of BI-Loc with quantifications on locations and paths, denoted BI-Loc$_{\exists,\forall}$ are defined as follows:

$$\begin{array}{c}\phi ::= p \mid I \mid \phi * \phi \mid \phi \mathrel{-\!\!*} \phi \mid \top \mid \bot \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid [l]\phi \\ \mid \exists_{loc}x.\phi \mid \forall_{loc}x.\phi \mid \exists_{path}x.\phi \mid \forall_{path}x.\phi\end{array}$$

with l location or location variable and p propositional variable.

Therefore we have to extend the resource tree model given for BI-Loc.

Definition 16 (Resource Tree Model for BI-Loc$_{\exists,\forall}$). *A* partial (resource) tree model *for BI-Loc$_{\exists,\forall}$ is the extension of a* partial (resource) tree model *for BI-Loc that satisfies the $\models$ clauses of Definition 7 and in addition*

- $t \models \exists_{loc}x.\phi$ *iff there exists l such that* $t \models \phi\{^l/_x\}$.
- $t \models \forall_{loc}x.\phi$ *iff for all locations l,* $t \models \phi\{^l/_x\}$.
- $t \models \exists_{path}x.\phi$ *iff there exist $l_1 \dots l_n$ $(n \in \mathbb{N})$,* $t \models \phi)\{^{[l_1]\cdots[l_n]}/[x]\}$.
- $t \models \forall_{path}x.\phi$ *iff for all n and all locations $l_1 \dots l_n$,* $t \models \phi\{^{[l_1]\cdots[l_n]}/[x]\}$.

Our way to handle separation is very useful since it allows to easily express that we add resources at a given location. Furthermore, with quantifications we can as well express that we must decompose a tree in two disjoint subtrees (i.e., subtrees with no common path except the root). Let us note that subtrees are disjoint as soon as they do not share a location at the root level.

In order to express that t can be decomposed in two disjoint subtrees that respectively satisfy ϕ and ψ, we write that $t \models (\phi * \psi) \wedge (\exists_{path}x.\exists_{loc}y.((\phi \wedge exists(x:y)) * (\psi \wedge exists(x:y))) \rightarrow \bot)$ with x not free in ϕ, ψ, $x:y$ that represents the concatenation of paths x and y and $exists(L)$ is $\top * [L]\top$.

It means that there exists at least a decomposition of t into two subtrees t_1 and t_2 satisfying respectively ϕ and ψ and that each decomposition of this kind is different. The first subformula ensures that the decomposition exists and the subformula $\exists x.((\phi \wedge exists(x)) * (\psi \wedge exists(x))) \rightarrow \bot)$ means that for t_1 and t_2 such that $t_1 \models \phi$, $t_2 \models \psi$, and $(t_1|t_2) \equiv t$, t_1 and t_2 are disjoint. Let us prove it. We suppose that t_1 and t_2 are not disjoint. By definition, there exists a location l' shared by the two subtrees. Then we have $t_1 \models (\phi \wedge exists(l'))$ and $t_2 \models (\psi \wedge exists(l'))$ and consequently $(t_1|t_2) \models \bot$. We obtain a contradiction and then deduce the result.

5.2 Satisfaction and Validity Results

Now we have to analyze the consequences of such extensions of the model and logic w.r.t. our previous results about model-checking for satisfaction and validity for BI-Loc. We can show that satisfaction is decidable for BI-Loc$_{\exists,\forall}$ but without $-\!\!*$ subformulae (i.e., including the $-\!\!*$ connective). Then we define a procedure to check the satisfaction of a formula ϕ by a resource tree t.

Definition 17. *Let t be a bounded resource tree and ϕ a BI-Loc$_{\exists,\forall}$ formula without $-\!\!*$ subformulae. The procedure $Check(t,\phi)$ is inductively defined as follows:*

- $Check(t,p) = true$ *if* $t=(m,nil)$ *and* $m \models p$, $false$ *otherwise;*
- $Check(t,\phi * \phi') = \bigvee_{(t_1|t_2)\sqsubseteq t}(Check(t_1,\phi) \wedge Check(t_2,\phi'))$;
- $Check(t,I) = true$ *if* $t=(m,nil)$ *and* $e \sqsubseteq m$, $false$ *otherwise;*
- $Check(t,\phi \wedge \phi') = Check(t,\phi) \wedge Check(t,\phi')$;
- $Check(t,\top) = true$;
- $Check(t,\phi \vee \phi') = Check(t,\phi) \vee Check(t,\phi')$;
- $Check(t,\bot) = false$;
- $Check(t,\phi \to \phi') = \neg Check(t,\phi) \vee Check(t,\phi')$;
- $Check(t,[l]\phi) = true$ *if* $t=(m,l\mapsto t')$ *and* $Check(t',\phi)$, $= false$ *otherwise;*
- $Check(t,\exists_{loc}x.\phi) = \bigvee_{l\in\{l_0\}\cup Loc_\phi \cup Loc_t} Check(t,\phi\{^l/_x\})$ *with* $l_0 \in Loc \setminus (Loc_\phi \cup Loc_t)$;
- $Check(t,\forall_{loc}x.\phi) = \bigwedge_{l\in\{l_0\}\cup Loc_\phi \cup Loc_t} Check(t,\phi\{^l/_x\})$ *with* $l_0 \in Loc \setminus (Loc_\phi \cup Loc_t)$;
- $Check(t,\exists_{path}X.\phi) = \bigvee_{L\in Reachable(t,\phi)} Check(t,\phi\{^L/_X\})^*$;
- $Check(t,\forall_{path}X.\phi) = \bigwedge_{L\in Reachable(t,\phi)} Check(t,\phi\{^L/_X\})^*$.

* *with $Reachable(t,\phi)$ that defines the set of paths $l_1,\ldots l_n$ such that $n<h(\phi)$ and $l_i \in \{l_0\}\cup Loc_\phi \cup Loc_t$ with $l_0 \in Loc\setminus(fv(\phi)\cup fv(t))$. Moreover $\bigvee$ (resp. $\bigwedge$) represents the disjunction (resp. conjunction) on sets of locations or paths.*

Lemma 7. *Let $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T)_{\mathcal{M},Loc}$ be a partial tree model in which $(T,(e,nil),|,\sqsubseteq_T)$ is a tree monoid maximally defined and $\mathcal{M}$ is a partial boundable monoid. Let $t\in T$ be a resource tree and ϕ be a BI-Loc$_{\exists,\forall}$ formula, if ϕ does not contain $-\!\!*$ subformulae then $Check(t,\phi)$ terminates.*

Proof. By structural induction on ϕ.

Theorem 4 (Satisfaction Decidability). *Let $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T,\models)_{\mathcal{M},Loc}$ be a partial tree model in which $(T,(e,nil),|,\sqsubseteq_T)$ is a tree monoid maximally defined and $\mathcal{M}$ is a partial boundable monoid. Let $t\in T$ be a resource tree and ϕ be a BI-Loc$_{\exists,\forall}$ formula, if ϕ does not contain $-\!\!*$ subformulae, $t\models\phi$ is decidable and $t\models\phi$ if and only if $Check(t,\phi)$.*

Proof. By Lemma 7 the *Check* procedure terminates. Then we show, by structural induction on ϕ, that $t\models\phi$ iff $Check(t,\phi)$.

Then we show that validity and satisfaction are undecidable for BI-Loc$_{\exists,\forall}$ that includes $-\!\!*$ subformulae. Moreover, we have the same even if we only consider quantifications on locations. In order to prove these results, we show that the validity for BI-Loc$_{\exists,\forall}$, even without $-\!\!*$, is undecidable. It is closely related to results for BI's pointer logic [9] and Ambient logic [13]. In both proofs, the result relies on the following undecidability result (Trakhtenbrot [27]): even if a signature consists only of one binary relation, we cannot decide if a closed first order formula ϕ admits a finite model.

The proof consists in reducing the above validity problem to our validity problem. To do such a reduction, we provide a relation $[\![\,.\,]\!]_{FO}$ between a first order logic formula with a single binary relation R and a BI-Loc formula, that is defined as follows:
$[\![\phi\vee\phi']\!]_{FO} = [\![\phi]\!]_{FO} \vee [\![\phi']\!]_{FO}$, $[\![\phi\wedge\phi']\!]_{FO} = [\![\phi]\!]_{FO} \wedge [\![\phi']\!]_{FO}$, $[\![\phi\to\phi']\!]_{FO} = [\![\phi]\!]_{FO} \to [\![\phi']\!]_{FO}$,

$[\![\neg\phi]\!]_{FO} = [\![\phi]\!]_{FO} \to \bot$, $[\![\exists x.\phi]\!]_{FO} = \exists_{loc} x.(([d][x]I * \top) \wedge [\![\phi]\!]_{FO})$, $[\![R(x_1,x_2)]\!]_{FO} = [r][x_1][x_2]I * \top$.

The underlying idea is to represent the domain $\mathcal{D}$ by a location name below the location d and to represent the relation $R(x_1,x_2) \in \mathcal{S}$ by a location $[r][x_1][x_2]$ where the location name r stands for the name of the relation R (we consider same names as variables and elements of $\mathcal{D}$). Then we can prove the following lemma:

Lemma 8. *Let ϕ be a first-order formula, we have $\not\models [\![\phi]\!]_{FO} \to \bot$ iff there exists a finite model of ϕ.*

Proof. We define a relation between a structure $\mathcal{S}$ over a domain $\mathcal{D}$ ($\mathcal{S}$ a set of objects of the form $R(a_1,a_2)$ where a_1,a_2 belong to $\mathcal{D}$) and a resource tree t.
We have $\mathcal{S} \rightsquigarrow_{tree} t$ iff (i) if $a \in \mathcal{D}$ then there exists t' such that $t = ([d][a]e|t')$ and (ii) if $a_1,a_2 \in \mathcal{D}$ and $R(a_1,a_2) \in \mathcal{S}$, then there exists t' such that $t = ([r][a_1][a_2]e|t')$.
Then, we can build the reverse relation $\rightsquigarrow_{struct}$ from a subset of resource trees into the set of structures. Considering a closed first-order formula ϕ, we show that (i) $\mathcal{S} \models \phi$ iff for t such that $\mathcal{S} \rightsquigarrow_{tree} t$, we have $t \models [\![\phi]\!]_{FO}$ and (ii) $t \models [\![\phi]\!]_{FO}$ if and only if for $\mathcal{S}$ such that $t \rightsquigarrow_{struct} \mathcal{S}$ we have $\mathcal{S} \models \phi$. The proof is done by structural induction on ϕ.

By Lemma 8 and Trakhtenbrot's theorem we deduce the following result:

Theorem 5 (Validity Undecidability). *Let $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T,\models)_{\mathcal{M},Loc}$ be a resource tree model where $(T,(e,nil),|,\sqsubseteq_T)$ is a maximally defined tree monoid based on a boundable partial monoid $\mathcal{M}$. For any formula ϕ of BI-Loc$_{\exists,\forall}$ without $-\!\!*$ subformulae, $\mathcal{T} \models \phi$ is undecidable.*

Then, by Lemma 1 which shows that we can express validity as a satisfaction relation, we prove the following results for BI-Loc$_{\exists,\forall}$ including $-\!\!*$ subformulae.

Theorem 6 (Satisfaction Undecidability). *Let $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T,\models)_{\mathcal{M},Loc}$ be a resource tree model where $(T,(e,nil),|,\sqsubseteq_T)$ is a maximally defined tree monoid based on a boundable partial monoid $\mathcal{M}$. For any formula ϕ of BI-Loc$_{\exists,\forall}$ and for any resource tree $t \in \mathcal{T}$, $t \models \phi$ is undecidable.*

Proof. Direct consequence of Theorem 5 and Lemma 1.

Theorem 7 (Validity Undecidability). *Let $\mathcal{T} = (T,(e,nil),|,\sqsubseteq_T,\models)_{\mathcal{M},Loc}$ be a resource tree model where $(T,(e,nil),|,\sqsubseteq_T)$ is a maximally defined tree monoid based on a boundable partial monoid $\mathcal{M}$. For any formula ϕ of BI-Loc$_{\exists,\forall}$, $\mathcal{T} \models \phi$ is undecidable.*

Proof. As the subset without $-\!\!*$ formulae is undecidable, BI-Loc with quantifications is undecidable.

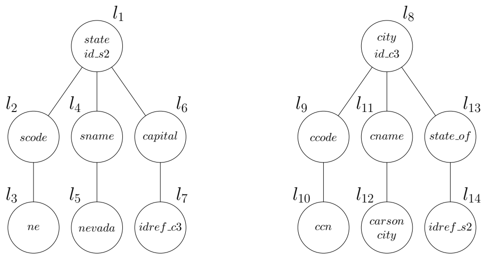
Such quantifications, mixed with the expressivity of resource trees, allow to treat useful data structures knowing that they may lead to undecidability if their use is not enough restricted. In this context, it could be interesting to analyze possible relationships between our results and works about hybrid logics from both model-checking and complexity perspectives [2,15].

5.3 Semi-structured Data

Coming back to our example about representation of XML trees as resources trees, let us illustrate the interest of our model and logic for such a representation of semi-structured data.
In order to represent XML data, we must represent the different data (entities, attributes) and to ensure some specificities of XML data such as the unicity of an attribute for a given node and some children restriction (some elements cannot have children). The structure of our resource tree is given by the tree structure of *entities* and *data*. It means that both *data* and *entities* correspond to a node. However, we cannot have two sibling nodes with the same label. Then we arbitrary choose the node labels that are not significant here.

The only way to differentiate all these nodes consists in representing elements and data as resources inside nodes. We also define *attributes* as resources. To ensure that nodes corresponding to data are childless, we decide that their corresponding resources lead to an undefined resource tree when they are composed with any resource tree. Similarly, a resource corresponding to an attribute cannot be composed with another resource corresponding to the same attribute.

⟨state id='s2'⟩
  ⟨scode⟩NE⟨/scode⟩
  ⟨sname⟩Nevada⟨/sname⟩
  ⟨capital⟩idref='c3'⟨/capital⟩
⟨/state⟩
⟨city id='c3'⟩
  ⟨ccode⟩CCN⟨/ccode⟩
  ⟨cname⟩Carson City⟨/cname⟩
  ⟨state_of⟩idref='s2'⟨/state_of⟩
⟨/city⟩



The relationship between a semi-structured data and a resource tree is mainly syntactic: the resulting resource tree does not contain any semantical information. Actually, such informations of the initial semi-structured data are expressed thanks to propositions of BI-Loc and their interpretation of the resource tree model.
Let us illustrate this point from a DTD (Document Type Definition) that is a document which defines how a XML document must be designed. It fixes which entities and attributes are valid, which attributes can be defined for which elements and so on. We show here how some DTD's rules can be defined as BI-Loc formulae. We first recall that $no(L)$ is defined as $exists(L) \rightarrow \bot$ that means that in a given tree there is no node corresponding to the path L. For instance, to ensure that t has exactly one child, which could be either a son element or a $daughter$ element we write $t \models \forall_{path} x.(([x](elem \wedge parent) * \top) \rightarrow ([x](\forall y.([y]\top \rightarrow [y](elem \wedge (son \vee daughter) * \top))) * no(x)))$. Moreover, to ensure that a *parent* element *can* have an attribute which indicates its sex we write $t \models \forall_{path} x.(([x](elem \wedge parent) * \top) \rightarrow ([x]((elem \wedge parent) * (I \vee (attrib \wedge sex))) * \top))$. The unicity of the *id* attribute of value *id_val* corresponds to $t \models (\top * \exists_{path} x.[x](id \wedge id\_val) * \exists_{path} x.[x](id \wedge id\_val)) \rightarrow \bot$. We could also develop the way to represent pointers in trees and show that BI-Loc$_{\exists,\forall}$ is well adapted to check some specific constraints. Before to illustrate these results and their consequences on the management of heap structures, a key point consists in defining a language (commands and models) in order to deal with resource trees.

6 A Language for Resource Tree Transformations

The above models allow to represent static tree data structures and we can use BI-Loc for describing a static configuration of resource trees. We aim at defining a language dedicated to resource tree management and then at expressing pre- and postconditions in a related assertion logic. The approach we develop is closed to O'Hearn and Reynolds approaches that are dedicated to mutable data structures [20,26]. Our core language allows some basic modifications on a resource tree: to add or dispose a location or a subtree, to add or update resources at a given location and to lookup to the content of locations.
We give here a formal definition of this language and discuss about its axiomatization through Hoare triples and finally about locality inside this language.

6.1 Commands and models

This language is related to the imperative language for heap manipulation of [20,23]. The idea is to keep the conditional and loop commands and then to adapt the assignment and model manipulations to deal with resource trees.

C ::=	if (E) then C else C'	If ... then ... else ...
	while (E) do C	Loop
	$C\ ;\ C$	Sequentiality
	$x := new_{loc}(E)$	Create new location
	$dispose_{loc}(E)$	Delete a location
	$x := E$	Variable assignment
	$x := res(E)$	Resource content lookup
	$x := tree(E)$	Tree content lookup
	$update_{res}(E_1, E_2)$	Update Resource Content
	$update_{tree}(E_1, E_2)$	Update Tree Content
	$add(E_1, E_2)$	Add content

Here, E, E_i are simple expressions (they do not contain commands). The semantic domain of these expressions depends on the command they are involved in. The three first commands (*if-then-else, while, sequentiality*) are standard control commands. The two following commands ($new_{loc}(E)$ and $dispose_{loc}(E)$) modify the tree structure by allowing to create or to dispose nodes. Consequently, in these two commands, the expression E must be interpreted as a path. In the first one, E refers to the path of the parent node of the location we want to create. We stock the path corresponding to this location in a given variable. For the second one E contains the path which must be deleted from the tree. The next command ($x := E$) makes a direct variable assignment, the expression being interpreted as any kind of value. Then, we have two commands ($x := res(E)$ and $x := tree(E)$) that assign a variable with the resource or the tree contained in a given location. Here, the expression E must also be interpreted as the path we have to look at in order to find the resource (resp. the tree). The next two commands ($update_{res}(E_1, E_2)$ and $update_{tree}(E_1, E_2)$) update the resource or the tree contained in a location. The expression E_1 is the path of the location we want to update and E_2 must be interpreted as respectively a resource and a tree. These commands replace the resource (resp. the tree) in E_1 by the one in E_2. The last command ($add(E_1, E_2)$) adds contents, either a tree or a resource, to a location. By adding contents we mean that E_2 must be interpreted as a resource or a resource tree and will be composed with the content of the location E_1.
Formally, an expression is either a variable, a resource, a resource tree, a list of locations (namely a path) or any other expression which can be interpreted as a member of these sets. We use the notation E_T to denote an expression interpreted as a resource tree. As we need program variables we extend our resource tree to handle them. The content of a variable is either a path or a resource or a whole resource tree and they are maintained according to a stack discipline. Such an approach allows to manipulate as well subtrees than variables in the program. A stack is a finite function which associates variables and their values. Consequently, we now manipulate a resource tree and a stack of variables, this pair being called a *state*. Its formal definition is given below:

Definition 18. *Let Loc be a set of locations, $\mathcal{M} = (M, \times, e, \sqsubseteq)$ be a resource monoid, $\mathcal{V} = \{x, y, z, \ldots\}$ be a set of variables and $T_{\mathcal{M}}$ be the set of resource trees. A* state *is a pair (s,t) composed by a store $s : \mathcal{V} \rightharpoonup_{fin} M \cup \mathcal{L}^* \cup T_{\mathcal{M}}$ and a resource tree t.*

Consequently, given a expression E, $[\![E]\!]_s = s(E)$ if E is a variable, E otherwise. The commands are interpreted using a relation $\rightsquigarrow$ on configurations. A configuration is either a triple C, s, t (where C is a command, s a stack of variables and t a resource tree) or a final configuration s, t (without command). The semantics of commands are given through the definition of this relation $\rightsquigarrow$. Let us start with the standard semantics of control commands

$$\frac{i = 1 \text{ if } [\![E]\!]s = true, i = 2 \text{ if } [\![E]\!]s = false}{\text{if } (E) \text{ then } C_1 \text{ else } C_2, s, t \rightsquigarrow C_i, s, t} \qquad \frac{C, s, t \rightsquigarrow s', t' \qquad C', s', t' \rightsquigarrow s'', t''}{(C; C'), s, t \rightsquigarrow s'', t''}$$

$$\frac{[\![E]\!]s = true}{\text{while } (E) \text{ do } C \text{ end}, s, t \rightsquigarrow C; \text{while } (E) \text{ do } C \text{ end}, s, t} \qquad \frac{[\![E]\!]s = false}{\text{while } (E) \text{ do } C \text{ end}, s, t \rightsquigarrow s, t}$$

We consider now the semantic rules that are specific to resource trees. We recall that $t(L)$, defined in Section 2, represents the subtree of t at path L.

$$\frac{[\![E]\!]_s = L \in Loc^* \text{ },l \in Loc \text{ s.t. } [t(L)] \downarrow \text{ and } t(L:l) \text{ undefined}}{x := new_{loc}(E), s, t \rightsquigarrow [s|x \mapsto L:l], t|[L][l]e}$$

$$\frac{[\![E]\!]_s = L \in Loc^* \text{ s.t. } t \equiv t'|[L]t'' \text{ and } t'(L) \text{ undefined}}{x := dispose_{loc}(E), s, t \rightsquigarrow s, t'}$$

In the case of the $dispose_{loc}$ command, we ensure that t' does not contain a subtree at location L.

$$\frac{[\![E]\!]_s = v \in M \cup \mathcal{L}^*}{x := E, s, t \rightsquigarrow [s|x \mapsto v], t}$$

$$\frac{[\![E]\!]_s = L \in Loc^* \text{ },m \in M \text{ s.t. } [t(L)] \downarrow \text{ and } t(L) = (m,f)}{x := res(E), s, t \rightsquigarrow [s|x \mapsto m], t}$$

The difficulty with the $res(E)$ command is to ensure that we do not have no other resources than m at location L.

$$\frac{[\![E]\!]_s = L \in Loc^* \text{ },m \in M \text{ s.t. } [t(L)] \downarrow \text{ and } t(L) = (m,f)}{x := tree(E), s, t \rightsquigarrow [s|x \mapsto (m,f)], t}$$

$$\frac{[\![E_2]\!]_s = m \in M, [\![E_1]\!]_s = L \in Loc^* \text{ s.t. } t \equiv t'|[L]m' \text{ and } t'(L) = (e,f)}{update_{res}(E_1,E_2), s, t \rightsquigarrow s, t'|[L]m''}$$

$$\frac{[\![E_2]\!]_s = t', [\![E_1]\!]_s = L \in Loc^* \text{ s.t. } t \equiv t''|[L]t''' \text{ and } t''(L) \text{ undefined}}{update_{tree}(E_1,E_2), s, t \rightsquigarrow s, t''|[L]t'}$$

$$\frac{[\![E_2]\!]_s = t', [\![E_1]\!]_s = L \in Loc^* \text{ s.t. } [t(L)] \downarrow}{add(E_1,E_2), s, t \rightsquigarrow s, t|[L]t'}$$

Definition 19. *Let C be a command, s be a store and t be a resource tree,*
- the configuration C,s,t is stuck *iff there is no configuration K such that $C,s,t \rightsquigarrow K$.*
- the configuration C,s,t is safe *iff for any configuration K such that $C,s,t \rightsquigarrow^* K$ then K is a terminal or non-stuck configuration, $\rightsquigarrow^*$ being the transitive closure of $\rightsquigarrow$.*

By stuck, we mean that the command cannot be executed. For example, a command which will try to create a new location below a resource instead of below a given path will be stuck.

6.2 BI-Loc as Assertion Language

We express assertions of our language in a logic derived from BI-Loc. We have slightly modified it in order to add tests on expressions, and some quantifications that are useful to reason about the contents of a resource tree.

Definition 20. *Let Loc be a set of location names the* assertions *(preconditions and postconditions) for commands are defined as follows:*

$$\begin{array}{rl}\phi ::= & \alpha \mid I \mid \phi * \phi \mid \phi -\!\!* \phi \mid \top \mid \bot \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid \quad \mid [l]\phi \\ & \mid \exists_{loc} x.\phi \mid \exists_{path} x.\phi \mid \exists_{res} x.\phi \mid \exists_{tree} x.\phi \mid \forall_{loc} x.\phi \mid \forall_{path} x.\phi \mid \forall_{res} x.\phi \mid \forall_{tree} x.\phi\end{array}$$

We have added quantifications on resources and paths in order to directly reason about contents of the resource tree. We have also replaced the set of propositions by the set of atomic formulae α which is defined as $\alpha ::= E \mid E = E$, where E is an expression.

The semantics of assertions is given by a forcing relation of the form $s,t \models \phi$ which asserts that ϕ is true for a given stack s and a resource tree t. We observe that the semantic clauses are closed to the resource tree model.

Definition 21 (Semantic clauses). *Let ϕ be an assertion of Definition 20, s be a stack and t be a resource tree, the forcing relation $s,t \models \phi$ is defined as follows:*

- $s,t \models [l]\phi$ *iff there exists t' such that $[l]t' \sqsubseteq_T t$ and $s,t' \models \phi$*
- $s,t \models \exists_{loc} x.\phi$ *iff there exists l such that $[s|x \mapsto l],t \models \phi$.*
- $s,t \models \forall_{loc} x.\phi$ *iff for any location l, $[s|x \mapsto l],t \models \phi$.*
- $s,t \models \exists_{path} x.\phi$ *iff there exist $l_1 \dots l_n$ $(n \in \mathbb{N})$, $[s|x \mapsto (l_1,\dots l_n)],t \models \phi$.*
- $s,t \models \forall_{path} x.\phi$ *iff for any n and any $l_1 \dots l_n$, $[s|x \mapsto (l_1,\dots l_n)],t \models \phi$.*
- $s,t \models \exists_{res} x.\phi$ *iff there exists $m \in M$, $[s|x \mapsto m],t \models \phi$.*
- $s,t \models \forall_{res} x.\phi$ *iff for any $m \in M$, $[s|x \mapsto m],t \models \phi$.*
- $s,t \models \exists_{tree} x.\phi$ *iff there exists $t' \in T_M$, $[s|x \mapsto t'],t \models \phi$.*
- $s,t \models \forall_{tree} x.\phi$ *iff for any $t' \in T_M$, $[s|x \mapsto t'],t \models \phi$.*
- $s,t \models \top$ *always.*
- $s,t \models \bot$ *never.*
- $s,t \models E = E'$ *iff $[\![E]\!]_s = [\![E']\!]_s$.*
- $s,t \models E$ *iff $t = [\![E]\!]_s$ if $[\![E]\!]_s$ is a resource tree, $[\![E]\!]_s \sqsubseteq_T t$ otherwise.*
- $s,t \models \phi \vee \phi'$ *iff $s,t \models \phi$ or $s,t \models \phi'$.*
- $s,t \models \phi \wedge \phi'$ *iff $s,t \models \phi$ and $s,t \models \phi'$.*
- $s,t \models \phi \rightarrow \phi'$ *iff if $s,t \models \phi$ then $s,t \models \phi'$.*
- $s,t \models I$ *iff $e \sqsubseteq_T t$*
- $s,t \models \phi * \phi'$ *iff there exist t',t'' such that $[(t'|t'')]\downarrow$, $(t'|t'') \sqsubseteq_T t$, $s,t' \models \phi$ and $s,t'' \models \phi'$.*
- $s,t \models \phi -\!\!* \phi'$ *iff for any t' such that $s,t' \models \phi$ and $[(t|t')]\downarrow$, we have $s,(t|t') \models \phi'$.*

Even if we work on resource trees while O'Hearn et al. work directly with heaps [20], both models have a lot of similarities. The behavior of the connective is identical. We do not have a *points-to* operator to indicate that a value corresponds to a location, but we consider a location modality which expresses that a resource is at a given location. Furthermore, we have introduced different kinds of quantifications to reason about different semantical domains of our model.
If we compare with semantic clauses of the generic model for resource trees presented before, the new quantifications introduce non-trivial properties. First, we want to emphasize that for all s,t, we have $s,t \models \exists_{tree} x.x$ since $[s|x \rightarrow t],t \models x$. Quantifications on resources and trees allow to precisely define the contents of the structure we consider. It leads to some unusual results with respect to other separation logics. For example, the formula $\exists_{tree} x,y.(((x*y) \wedge p) \rightarrow x*(x -\!\!* p))$ is always true. Actually, the left part ensures that there exist subtrees x,y such that $t \equiv (x|y)$ and $s,t \models p$. Consequently $s,(x|y) \models p$ and thus $s,y \models x -\!\!* p$. Contrary to $((\phi * \phi') \wedge \phi'') \rightarrow (\phi * (\phi' -\!\!* \phi''))$ which is not always true, a resource tree may satisfy ϕ' but not ϕ'' if it is composed with a resource tree satisfying ϕ.

Definition 22. *Let ϕ,ϕ' be assertions of Definition 20, such that $fv(\phi) \cup fv(\phi') \subseteq dom(s)$. The semantic consequence relation $\models$ is defined by $\phi \models \phi'$ iff for all s,t, if $s,t \models \phi$ then $s,t \models \phi'$.*

The usual rules of BI Logic are sound for $-\!\!*$. We also have the following rules

$$\frac{\phi * \psi \models \psi'}{\phi \models \psi -\!\!* \psi'} \quad \frac{\phi \wedge \psi \models \psi'}{\phi \models \psi \rightarrow \psi'} \quad \frac{\phi \models \phi' \quad \psi \models \psi'}{\phi * \psi \models \phi' * \psi'} \quad \frac{[L']\phi \models [L']\phi'}{[L]\phi \models [L]\phi'}$$

The last one, on paths, defines a key behavior of location modality: if a semantic consequence holds at a given location, we can change the location without loosing this property. Furthermore, we can check expressions without looking at the resource tree. Such expressions are called *pure* expressions. A pure expression is an expression where the atomic formulae are always equality and without the unit I.

Lemma 9. *A pure expression only depends on the stack of variables, i.e., if ϕ is a pure assertion then $s,t \models \phi$ iff for any resource tree t' we have $s,t' \models \phi$.*

We can show that if ϕ and ψ are pure then $\psi * \phi$ (resp. $\psi -\!\!* \phi$) is equivalent to $\phi \wedge \psi$ (resp. $\phi \rightarrow \psi$).

6.3 Hoare triples

Hoare triples are of the form $\{\phi\}C\{\psi\}$ where ϕ and ψ are assertions and C is a command. We consider an interpretation such that a command we apply is not stuck.

Definition 23 (Sound triples). *A triple $\{\phi\}C\{\psi\}$ is* sound *iff for any configuration C,s,t such that $s,t \models \phi$ and $fv(\phi) \cup fv(\psi) \subseteq dom(s)$ then C,s,t is safe and if $C,s,t \rightsquigarrow^* s',t'$ then $s',t' \models \psi$.*

6.4 Basic axioms

Now we have to fix the command axioms and to deal with the fact that the separation connective $*$ does not ensure separation of locations. Contrary to the usual approach [20,23], freshness is not ensured by the separation connective. Actually, in pointer logic $\phi * \psi$ requires that we can split a heap in two disjoint subheaps in order to satisfy a formula but it is not the case in BI-Loc. We need more complex preconditions for ensuring that a subtree does not contain resources (or subtrees) at a given path.
Let us start with the usual Hoare rules for the control commands.

$$\frac{\{\phi \wedge E\}C\{\psi\} \quad \{\phi \wedge (E \rightarrow \bot)\}C'\{\psi\}}{\{\phi\}\text{if } (E) \text{ then } C \text{ else } C'\{\psi\}} \quad \frac{\{\phi \wedge E\}C\{\psi\} \quad (\phi \wedge (E \rightarrow \bot)) \rightarrow \psi}{\{\phi\}\text{while } (E) \text{ do } C \text{ end}\{\psi\}}$$

We go on with the usual Hoare rules for sequencing, consequence and simple assignment:

Sequencing: *Consequence:*

$$\frac{\{\phi\}C\{\phi'\} \quad \{\phi'\}C\{\psi\}}{\{\phi\}C;C'\{\psi\}} \qquad \frac{\phi \models \phi' \quad \{\phi'\}C\{\psi'\} \quad \psi' \models \psi}{\{\phi\}C\{\psi\}}$$

In the rule above, $\models$ is the semantic consequence relation.

- *Simple assignment*: $\{\phi\{^{E}/_{x}\}\}x := E\{\phi\}$

- *Resource observation*:

$$\{\exists_{res}x_1.(\phi\{^{x_1}/_{x}\} \wedge (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot) * [E]x_1))\}x := res(E)\{\phi\}$$

Here the precondition ensures that the command leads to the postcondition ϕ and that we consider the right value to put in x. Then we must decompose the resource tree into two subtrees: one that contains exactly the path E and its resources, another one that contains the rest. But we have to be sure that the second subtree does not contains resources of E. Then, we verify that we cannot find a resource (different from e) at the path E: $\forall_{res} y.((y = e) \vee (([E]y * \top) \rightarrow \bot)$.

- *Tree observation*:

$$\{\exists_{tree} x_1.(\phi\{^{x_1}/_x\} \wedge (no(E) * [E]x_1))\}x := tree(E)\{\phi\}.$$

We have a quite similar triple for tree lookup. This one is simpler since we already have defined a formula ($no(L)$) which checks that a path does not belong to a tree.

- *Location creation*:

$$\{\phi \wedge no(E:l)\}x := new_{loc}(E)\{\phi * [x]e \wedge (x = E:l)\}$$

The precondition ensures that ϕ is true and there is no path $[E][l]$ in the initial tree. It is important because the postcondition does not ensure that the created location is fresh, and then we have to fix it in the precondition. The main issue with this axiom is that it requires to give explicitly the created location contrary to the idea of arbitrary choice of locations. This problem is solved with the backward axiom presented below.

- *Location dispose*:

$$\{\phi \wedge no(E) * [E]\top\}x := dispose_{loc}(E)\{\phi\}$$

We only have to ensure that the postcondition does not involve the location we dispose.

- *Content update and addition*:

$$\{(\exists_{res} x.x = E_2) \wedge \exists_{res} y.\phi \ \wedge \ \forall_{res} x.((x = e) \ \vee \ ((\top * [E_1]x) \rightarrow \bot)) * [E_1]y\} \ x := update_{res}(E_1, E_2) \ \{\phi_{1,2}\}$$

$$\{(\exists_{tree} x.x = E_2) \wedge (\phi \wedge no(E_1) * [E_1]m)\}x := update_{tree}(E_1, E_2)\{\phi_{1,2}\}$$

$$\{(\exists_{res} x.x = E_2) \wedge \phi \wedge exists(E_1)\}x := add(E_1, E_2)\{\phi_{1,2}\}$$

with $\phi_{1,2} \equiv \phi * [E_1]E_2$.

The issue with *update* is the same as the one with location look-up: to isolate all resources inside a location. Thus, we decompose the tree into one subtree with these resources and another subtree with the rest and we then put the new contents into the locations. The *add* command is easier since we just put the resources next to the existing one.

6.5 Backward axioms

Some of the below axioms require that both conditions are written according to a special shape. We aim at finding a way to express axioms which is less restrictive. To do so, we propose backward axioms with generic postconditions.
A backward axiom is an axiom in which we describe in the precondition how we must extend actual structures to obtain a postcondition ϕ. It allows to reason backward on a program. For location creation, ϕ will

stand if we extend current tree with any new location. Thus, we have the following axiom:

- *Back location creation*:

$$\{\forall_{loc}x'.(no(E:x') \rightarrow ([E][x']e \mathrel{-\!\!*} \phi\{^{E:x'}/_x\}))\}x := new_{loc}(E)\{\phi\}$$

$\forall_{loc}$ ensures that we can use any location and $no(E:x')$ ensures that this location does not exist yet. Contrary to the axiom presented above this one handles the arbitrary choice of the new location name. And the $\mathrel{-\!\!*}$ connective indicates what we must add to our resource tree to obtain the postcondition.

- *Back content addition*:

$$\{(\exists_{res}x.x = E_2) \wedge (exists(E_1) \wedge ([E_1]E_2 \mathrel{-\!\!*} \phi))\}x := add(E_1, E_2)\{\phi\}$$

If the location intended to receive the content already exists, we add the right content at the right location, and we obtain ϕ.

- *Back content modification*:

$$\{(\exists_{res}x.x = E_2) \wedge (\exists_{res}y.[E_1]y * \forall_{res}x.((x = e) \vee (([E_1]x * \top) \rightarrow \bot)) \wedge ([E_1]E_2 \mathrel{-\!\!*} \phi))\}update_{res}(E_1, E_2)\{\phi\}$$

$$\{(\exists_{tree}x.x = E_2) \wedge (\exists_{tree}y.[E_1]y * (no(E_1) \wedge ([E_1]E_2 \mathrel{-\!\!*} \phi)))\}update_{tree}(E_1, E_2)\{\phi\}$$

To satisfy ϕ after an update command, we must ensure that the tree without the resources (resp. without the resource tree) inside location E_1 will satisfies ϕ if we add to E_1 the resource (resp. the resource tree) E_2.

7 Weakest pre-conditions

We first define weakest pre-conditions that corresponds to define, from a postcondition and a given instruction, the set of configurations that satisfy the postcondition when the command is applied without being stuck or failed.

Definition 24 (Weakest pre-condition). *Let C be a command and ϕ a formula, the* weakest pre-condition *$wp(C, \phi)$ is the set of configurations such that $s,t \in wp(C, \phi)$ iff if C, s, t is safe and if $C, s, t \rightsquigarrow s', t'$ then $s', t' \models \phi$.*

Moreover, a Hoare axiom $\{\phi\}C\{\psi\}$ is said *sound* if all configurations s,t that satisfy $s,t \models \phi$ also satisfy $s,t \in wp(C, \psi)$. It is said *complete* if and only if all configurations $s,t \in wp(C, \psi)$ verify $s,t \models \phi$.

Lemma 10 (Soundness). *The basic axioms are sound w.r.t. the semantic clauses.*

Proof. By case analysis on each axiom.
- $x := E$: we assume that $s,t \models \phi\{^E/_x\}$. Thus, due to $\rightsquigarrow$ semantics, $x := E, s, t \rightsquigarrow [s, x \mapsto E], t$ and we have $[s, x \mapsto E], t \models \phi$.
- $x := res(E)$: we assume that $s,t \models \exists_{res}x_1.(\phi\{^{x_1}/_x\} \wedge (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot) * [E]x_1))$. So there exists a resource m such that $s,t \models \phi\{^m/_x\} \wedge (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot) * [E]m)$. As we have $s,t \models (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot) * [E]m)$, $[\![E_s]\!]$ exists and consequently, $x := res(E)$ is not stuck. This formula ensures that the content of t at location $[\![E]\!]_s$ is exactly m. Otherwise it would exist $t'', m' \neq e$ such that $t = [E]m|[E]m'|t'$ and $(t'|[E]m') \models ([E]m' * \top)$ and then $(t'|[E]m') \not\models (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot)$ and $s,t \not\models (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot) * [E]m)$. Consequently, $x := res(E), s, t \rightsquigarrow [s, x \mapsto m], t$ and we

have $[s,x \mapsto m],t \models \phi$.
- $x := tree(E)$: we assume that $s,t \models \exists_{tree} x_1.(\phi\{^{x_1}/_x\} \wedge (no(E) * [E]x_1))$. Then there exists t' and $L = [\![E]\!]_s$ such that t can be decomposed into two subtrees $[L]t'$ and t'' such that $t''(L)$ is undefined and thus $s,t'' \models no(E)$. Consequently, for $x_1 = t''$, by definition, $x := tree(E),s,t \rightsquigarrow [s,x \mapsto t'],t$ and then $[s,x \mapsto t'],t \models \phi$.
- $x := new_{loc}(E)$: we consider $x = new_{loc}(E),s,t$ such that, by definition, $new_{loc}(E),s,t \rightsquigarrow [s|x \mapsto L : l],t|[L][l]\mathbf{e}$, with $[\![E]\!]_s = L$ and $[L][l]$ is not a path of t. We now suppose that $s,t \models \phi \wedge no(E : l)$. Then, by definition $[\![E]\!]_s$ exists and the configuration is not stuck. Moreover, $[s|x \mapsto L : l],(t|[L][l]e) \models x = L : l$ and $[s|x \mapsto l],(t|[L][l]e) \models \phi * [L][l]e$. Thus $[s|x \mapsto l],(t|[L][l]e) \models \phi * [L][x]e \wedge (x = E : l)$.
- $dispose_{loc}(E)$: we assume that $s,t \models \phi \wedge no(E) * [E]\top$. Then $[\![E]\!]_s$ is a path (denoted L) and the configuration $dispose_{loc}(E),s,t$ is not stuck. Furthermore, by definition of the semantic clauses, there exist t',t'' such that $t = (t'|[L]t'')$ with $s,t' \models \phi \wedge no(E)$ and $s,[L]t'' \models [E]\top$. By definition $t'(L)$ is not defined and t'' is only defined in l. Thus $dispose_{loc}(E),s,t \rightsquigarrow s,t'$ and then $s,t' \models \phi$.
- $add(E_1,E_2)$: we assume that $s,t \models (\exists_{res} x.x = E_2) \wedge \phi \wedge exists(E_1)$. Consequently, $[\![E_1]\!]_s = L$ is a path and $[\![E_2]\!]_s = m$ is a resource. Thus $add(E_1,E_2),s,t \rightsquigarrow s,(t|[L]m)$ and we can deduce $s,(t|[L]m) \models \phi * [E_1]E_2$.
- $update_{tree}(E_1,E_2)$: we assume that $s,t \models (\exists_{tree} x.x = E_2) \wedge (\phi \wedge no(E_1) * [E_1]m)$. Thus $[\![E_1]\!]_s = L$ is a path and $[\![E_2]\!]_s = t'$ is a resource tree. Then there exist two subtrees u,u' such that $s,u \models \phi \wedge no(E_1)$ and $s,u' \models [E_1]\top$. Then u contains nothing in E_1 and u' only contains a subtree in E_1. Thus we have $update_{tree}(E_1,E_2),s,t \rightsquigarrow s,(u|[L]t')$ and by definition $(u|[L]t') \models \phi * [E_1]E_2$.
- $update_{res}(E_1,E_2)$: we assume that $s,t \models (\exists_{res} x.x = E_2) \wedge \exists_{res} y.\phi \wedge \forall_{res} x.((x = e) \vee ((\top * [E_1]x) \rightarrow \bot)) * [E_1]y$. Then $[\![E_1]\!]_s = L$ is a path and $[\![E_2]\!]_s = m$ is a resource. We can decompose t in two subtrees t',t'' such that $s,t' \models \phi \wedge \forall_{res} x.((x = e) \vee ((\top * [E_1]x) \rightarrow \bot)$ and $s,t'' \models \exists_{res} y.[E_1]y$. Then there exist a resource m and a tree t_1 such that $t'' = ([L]m|t_1)$. Then t' does not contain resources in L and therefore $update(E_1,E_2),s,t \rightsquigarrow (t'|[L]m)$ and $(t'|[L]m) \models \phi * [E_1]E_2$.

We must also prove that the backward axioms are also sound.

Lemma 11 (Backward axioms soundness). *The backward axioms are sound w.r.t. the semantic clauses.*

Proof. - $x := new_{loc}(E)$: we assume that $s,t \models \forall_{loc} x'.(no(E : x') \rightarrow ([E][x']e -\!\!* \phi\{^{E:x'}/_x\})$. We know that $[\![E]\!]_s$ is a path denoted L and consider a location l such that $t(L : l)$ is undefined. We have $x := new_{loc}(E),s,t \rightsquigarrow [s,x \mapsto L : l],(t|[L][l]e)$ and $s,t \models ([E][l]e -\!\!* P\{^{E:l}/_x\})$ and $[s,x \mapsto L : l],(t|[L][l]e) \models \phi$.
- $x := add(E_1,E_2)$: we assume that $s,t \models (\exists_{res} x.x = E_2) \wedge (exists(E_1) \wedge ([E_1]E_2 -\!\!* \phi))$. By definition, $[\![E_1]\!]_s$ is a path in t denoted L ($t(L)$ defined) and $[\![E_2]\!]_s$ is a resource tree t'. Thus the command is not stuck and we have $add(E_1,E_2),s,t \rightsquigarrow s,(t|[L]t')$. As $s,t \models [E_1]E_2 -\!\!* t$ we deduce $s,(t|[L]t') \models \phi$.
- $update_{res}(E_1,E_2)$: we assume that $s,t \models (\exists_{res} x.x = E_2) \wedge (\exists_{res} y.[E_1]y * \forall_{res} x.((x = e \rightarrow \bot) \rightarrow (([E_1]x * \top) \rightarrow \bot)) \wedge ([E_1]E_2 -\!\!* \phi))$. By definition $[\![E_1]\!]_s$ is a path (denoted L) of t ($t(L)$ is defined) and $[\![E_2]\!]_s$ is a resource (denoted m). We can decompose t in two subtrees t',t'' such that $s,t' \models \exists_{res} y.[E_1]y$ and $s,t'' \models \forall_{res} x.((x = e) \vee (([E_1]x * \top) \rightarrow \bot)) \wedge ([E_1]E_2 -\!\!* \phi)$. Then $s,t'' \models \forall_{res} x.(x = e) \vee (([E_1]x * \top) \rightarrow \bot)$ that means that the subtree t'' does not contain resources in L and then the command is not stuck. We deduce that $update_{res}(E_1,E_2),s,t \rightsquigarrow s,(t''|[L]m)$. As $s,t'' \models [E_1]E_2 -\!\!* \phi$ we have $s,(t''|[L]m) \models \phi$.
- $update_{tree}(E_1,E_2)$: we assume that $s,t \models (\exists_{res} x.x = E_2) \wedge (\exists_{tree} y.[E_1]y * (no(E_1) \wedge ([E_1]E_2 -\!\!* \phi)))$. By definition $[\![E_1]\!]_s$ is a path (denoted L) and $t(L)$ is defined and $[\![E_2]\!]_s$ is a tree (denoted u). We can decompose t into two subtrees t',t'' such that $t' \models \exists_{tree} y.[E_1]y$ and $s,t'' \models (no(E_1) \wedge ([E_1]E_2 -\!\!* \phi))$. Then t does not contain a subtree in L and as the configuration $update_{tree}(E_1,E_2),s,t$ is not stuck we have $update_{tree}(E_1,E_2),s,t \rightsquigarrow s,(t''|[L]u)$. From $s,t'' \models [E_1]E_2 -\!\!* \phi$ and $s,[L]u \models [E_1]E_2$, we deduce $s,(t''|[L]u) \models \phi$.

We now prove that the backward axioms are complete.

Lemma 12 (Backward axioms completeness). *The backward axioms are complete w.r.t. semantic clauses.*

Proof. We prove that for each configuration s',t' of the weakest precondition set $wp(C,\phi)$ and for each configuration C,s,t such that $C,s,t \rightsquigarrow s',t'$, if $\{\psi\}C\{\phi\}$ then $s,t \models \psi$.
- $s,t \in wp(x := E,\phi)$: by definition, $[s,x \mapsto E],t \models \phi$ and then $s,t \models \phi\{^E/_x\}$.
- $s,t \in wp(x := res(E),\phi)$: by definition, $[s,x \mapsto m],t \models \phi$, with $t(L) = (m,t')$, where $t = (t_1|[L]m)$ with $L = [\![E]\!]_s$. Thus $s,t \models \phi\{m/x\}$ and $t_1(L) = e$ and then $s,t_1 \models (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot))$. Thus $s,[L]m \models [E]m$ and $s,t \models \exists_{res}x_1.(\phi\{^{x_1}/_x\} \wedge (\forall_{res}y.(y = e) \vee (([E]y * \top) \rightarrow \bot) * [E]x_1))$.
- $s,t \in wp(x := tree(E),\phi)$: by definition, $[s,x \mapsto t'],t \models \phi$, and there exist t,L such that $L = [\![E]\!]_s$ and $t(L) = t'$. Then there exists t'' such that $t = (t''|[L]t')$. Thus $s,[L]t' \models \exists_{tree}x_1.[E]x_1$ and $s,t'' \models no(E)$. Consequently, $s,t \models \exists_{tree}x_1.(\phi\{^{x_1}/_x\} \wedge (no(E) * [E]x_1))$.
- $s,t \in wp(x := new_{loc}(E),\phi)$: by definition, $[s|x \mapsto L:l],(t|[L][l]e) \models \phi$, with $L = [\![E]\!]_s$ and $t(L:l)$ undefined. Thus, by definition of $-\!\!*$, $[s|x \mapsto L:l],t \models no(L:l) \rightarrow ([L][l]I -\!\!* \phi)$. As it is true for any location l, we finally deduce $s,t \models \forall_{loc}x'.(no(E:x') \rightarrow ([E][x']e -\!\!* \phi\{^{E:x'}/_x\}))$.
- $s,t \in wp(x := add(E_1,E_2),\phi)$: by definition, $s,(t|[L]m) \models \phi$ with $m = [\![E_2]\!]_s$, $L = [\![E_1]\!]_s$ and $[t(L)]\downarrow$. Then $s,t \models [E_1]E_2 -\!\!*\phi$ and as $t(L)$ is defined we have $s,t \models exists(E_2)$ and finally $s,t \models exists(E_1) \wedge ([E_1]E_2 -\!\!*\phi)$.
- $s,t \in wp(x := update_{res}(E_1,E_2),\phi)$: by definition $s,(t'|[L]m'') \models \phi$ with $[\![E_1]\!]_s = L$, $[\![E_2]\!]_s = m$ and t' such that $t'(L) = (e,f)$ and there exists m' such that $t = (t'|[L]m')$. As $t'(L) = (e,f)$ we have $s,t' \models \forall_{res}x.((x = e) \vee (([E_1]x * \top) \rightarrow \bot))$ and as $s,t'|[L]m \models \phi$ we deduce $s,t' \models \forall_{res}x.(x = e) \vee (([E_1]x * \top) \rightarrow \bot) \wedge ([E_1]E_2 -\!\!* P)$. Finally, as $t = (t'|[L]m')$ we have $s,t \models \exists_{res}y.[E_1]y * (\forall_{res}x.((x = e) \vee (([E_1]x * \top) \rightarrow \bot)) \wedge ([E_1]E_2 -\!\!*\phi))$.
- $s,t \in wp(x := update_{res}(E_1,E_2),\phi)$: by definition we have $s,(t_1|[L]t') \models \phi$ with $[\![E_2]\!]_s = t'$, $[\![E_1]\!]_s = L$ and t_1 such that $t_1(L)$ is undefined and there exists t_2 such that $t = (t_1|[L]t_2)$. As $t_1(L)$ is undefined we have $s,t_1 \models no(L)$ and as $s,(t_1|[L]t') \models \phi$ we deduce $s,t_1 \models no(L) \wedge ([E_1]E_2 -\!\!*\phi)$. Finally, as $t = (t_1|[L]t_2)$, we obtain $s,t \models \exists_{tree}y.[E_1]y * (no(E_1) \wedge ([E_1]E_2 -\!\!*\phi))$.

The previous results of soundness and completeness given for backward axioms allow to show that the pre-conditions of the axioms are minimal.

Theorem 8. *Let C be a command, ψ a postcondition, and $\{\phi\}C\{\psi\}$ be a Hoare triple built from axioms. For a stack s and a resource tree t we have $s,t \in wp(C,\psi)$ if and only if $s,t \models \phi$.*

Proof. Direct consequence of soundness results of Lemma 10 and Lemma 11 with the completeness result of Lemma 12.

8 The Frame property

Given a program and its preconditions and postconditions, we want to know if we can extend these conditions (frame property). Besides the usual problem with variables pointed out in [23], the introduction of locations and the way we compose locations introduce new issues to study. We now detail the related problems and define a frame property for a restricted set of commands.
First, we cannot extend the context using variables that are modified through commands. This is an usual restriction. Let us suppose that a command C modifies a variable z and that we have $\{\phi\}C\{\phi'\}$ (with no z in ϕ or ϕ'), then we do not have $\{\phi * [z]p\}C\{\phi' * [z]p\}$ since the value of z can be modified and we cannot ensure $[z]p$ after C.
Another problem is that the resource tree composition can lead to modifications of subtrees and resources below a given location. Then composition can alter the contents of a location and modify the behavior of our program. With the above language, we do not know which subparts of a resource tree can be modified or not. Roughly, we must ensure that the composition does not alter a location we update or read. Let us suppose for example that we have $\{\phi\}dispose_{loc}E\{\phi'\}$. To check $\{\psi * \phi\}dispose_{loc}E\{\psi * \phi'\}$, we must verify that there is no tree t such that $t \models \psi$ and $t \models exists(E)$. Otherwise, we cannot ensure that ψ will be true after the command execution. We have a similar problem with the *update* and *lookup* commands.
It can be easily solved with a single command specifying that the location we deal with is not involved in

the formula ψ. But it is difficult to generalize to a sequence of commands. We cannot know in advance to which locations correspond an expression involved in commands. Finally, the last problem comes from the location freshness in a new_{loc} command. Such a command requires to build a new, fresh location name. If we add another resource tree, we can add new location names and we are not sure anymore that the location names we create are fresh for the whole tree satisfying $\psi * \phi$.
These two last issues considerably restrict the frame rule possibility in the general case. However, if we do not take in account the new_{loc} command, it is sufficient to assume that we do not modify the values of free variables of ψ and that ϕ and ψ are disjoint. For the first part, ψ cannot contain free variables which are modified by C and these variables are those appearing at the left of a command (in $x := |E|$ and $x := E$). The set of variables modified by a command C is denoted $\mathit{Modified}(C)$. For the second part, we use the formula which represents the separation of the tree defined in section 5. Actually, if the tree satisfies $(\phi * \psi) \wedge (\exists_{loc} x.((\phi \wedge exists(x)) * (\psi \wedge exists(x))) \rightarrow \bot)$ then we ensure that ψ will is still true after execution of command C.

Theorem 9 (Frame property). *If we consider the set of commands above without the new_{loc} command, the rule below is sound if we do not have resources at the root of resource trees*

$$\frac{\{(\phi * \psi) \wedge (\exists_{loc} x.((\phi \wedge exists(x)) * (\psi \wedge exists(x))) \rightarrow \bot)\} C \{\phi' * \psi\}}{\{\phi\} C \{\phi'\}} \; *$$

$*$: $fv(\psi) \cap \mathit{Modified}(C) = \emptyset$

Proof. We suppose that $\{\phi\}C\{\phi'\}$ and show by induction on C that $\{(\phi * \psi) \wedge (\exists_{loc} x.((\phi \wedge exists(x)) * (\psi \wedge exists(x))) \rightarrow \bot)\} C \{(\phi' * \psi) \wedge (\exists_{loc} x.((\phi' \wedge exists(x)) * (\psi \wedge exists(x))))\}$.
- $dispose_{loc}(E)$: we assume that $s,t \models (\phi * \psi) \wedge (\exists_{loc} x.((\phi \wedge exists(x)) * (\psi \wedge exists(x))) \rightarrow \bot)$. By definition, there exist t_1, t_2 such that $s, t_1 \models \phi$ and $s, t_2 \models \psi$. As $s, t_1 \models \phi$, we have $C, s, t_1 \rightsquigarrow s', t_1'$ with $s, t_1' \models \psi$. Moreover, by definition of new_{loc}, $t_1([\![E]\!]s)$ is defined. Then, as $s, (t_1|t_2) \models \exists_{loc} x.((\phi \wedge exists(x)) * (\phi' \wedge exists(x)) \rightarrow \bot$, we deduce that t_1 and t_2 are disjoint and $t_2([\![E]\!]s)$ is undefined. As new_{loc} has only effect on this location then t_2 is not modified by the command and $s', (t_1'|t_2) \models \psi * \phi'$.
- $x := E$: as the command only alters the stack and $fv(\psi) \cap Modified(C) = \emptyset$ we can conclude.
- $x := res(E)$: we consider s,t such that $s,t \models (\phi * \phi') \wedge (\exists_{loc} x.((\phi \wedge exists(x)) * (\phi' \wedge exists(x))) \rightarrow \bot)$. By definition, there exist t_1, t_2 such that $s, t_1 \models \phi$ and $s, t_2 \models \phi'$. By hypothesis, as $s, t_1 \models \phi$ we have $s, t_1 \models \psi$ and then $C, s, t_1 \rightsquigarrow s', t_1'$ with $s, t_1' \models \psi$. Moreover, by definition of $x := res(E)$, $t_1([\![E]\!]s)$ is defined. As we have $s, (t_1|t_2) \models \exists_{loc} x.((\phi \wedge exists(x)) * (\phi' \wedge exists(x)) \rightarrow \bot$, we deduce that t_1 and t_2 are disjoint and thus $t_2([\![E]\!]s)$ is undefined. The result of the command is not modified by t_2 et then $s', (t_1'|t_2) \models \psi * \phi'$.
- $x := tree(E)$: similar to $x := res(E)$.
- $update_{res}(E_1, E_2)$: similar to $dispose_{loc}(E)$.
- $update_{tree}(E_1, E_2)$: similar to $dispose_{loc}(E)$.
- $add(E_1, E_2)$: similar to $dispose_{loc}(E)$.
- $C'; C''$: we consider s,t such that $s,t \models (\phi * \phi') \wedge (\exists_{loc} x.((\phi \wedge exists(x)) * (\phi' \wedge exists(x))) \rightarrow \bot)$. By definition of $*$, there exist t_1, t_2 such that $s, t_1 \models \phi$ and $s, t_2 \models \phi'$. By induction hypothesis, t_2 does not modify the result of C and C' commands and is also not modified by them. Thus, if $C'; C'', s, t_1 \rightsquigarrow s', t_1'$ then $s, (t_1'|t_2) \models \phi' * \psi$.

9 Resource Trees and Heap Manipulation

The model and the language proposed for resource trees are generic and we aim at studying two main instantiations with two kinds of heap structures: the model of pointers [23] and the model of permissions [5]. Thus we start from our model and the related commands in order to represent heaps as resource trees and to reason about pointer programs from a specialized Hoare triple semantics of our resource tree language. Within this approach we aim also at analyzing the permission accounting model [5] which can be seen as a particular extension of the heap model. In this context, we show how the related assertion logic allows to solve an open problem for this model and more generally how such logics and models can be useful for reasoning on heap structures.

9.1 Resource Trees and Heaps

We briefly recall on heaps structures and explain how to represent them as resource trees. Then, we relate the satisfaction problem in the pointer logic and in BI-Loc. Finally, we explain how commands on heaps can be represented as commands for resource tree manipulation.

First let us recall the semantic domain concerned by a heap structure (such as presented in [23]):
$Ints = \{\ldots,-1,0,1,\ldots\}$; $Loc \subseteq Ints$ with usually $Loc = \{1,\ldots,n\}$.
$Heap = Loc \rightharpoonup_{fin} Ints$; $Store = Var \rightharpoonup_{fin} Ints$; $State = Store \times Heap$;

In a heap structure the set of locations is a subset of relative numbers. As the set of locations of resource trees is enumerable, we use the same set without loss of generality. Furthermore, we consider paths with length equal to one.
We have to define a partial monoid of resources $\mathcal{M} = (M,\times,e,\sqsubseteq)$ such that resources included in locations of a resource tree correspond to values that are associated to a location in a heap.
Thus, we consider $Ints$ as the set of resources M and the composition $\times$ has to be totally undefined (to ensure that no location can contain two values). The issue is that with such a choice we do not have any neutral element. Therefore the partial monoid of resources we define is the following :
$\mathcal{M} = (Ints \cup \{e\},e,\times,\sqsubseteq)$ such that for all $m,m' \in Ints$, $m\times m'$ is undefined and for all $m \in Ints \cup \{e\}$ we have $m\times e = e\times m = m$. Moreover $\sqsubseteq$ is defined such that $m \sqsubseteq n$ if and only if $m = n$ and undefined elsewhere.

The correspondence between a heap h and the resource tree $[\![h]\!]_{RT}$ is defined as follows:
$[\![nil]\!]_{RT} = e$ and $[\![\,[h|l \mapsto v]\,]\!]_{RT} = ([l]v \mid [\![h]\!]_{RT})$.

A resource tree t is said *pure* if there exist $l_1,\ldots l_n \in Loc$ and $m_1,\ldots m_n \in Ints$ such that $t \equiv [l_1]m_1|\ldots|[l_n]m_n$. It means that all locations of t contain an integer and none contains only the resource e.

Lemma 13. *Let t be a resource tree, t is* pure *iff there exists a heap h such that $[\![h]\!]_{RT} \equiv t$.*

Proof. By definition of $[\![\cdot]\!]_{RT}$, we deduce that if $[\![h]\!]_{RT} \equiv t$ then t is pure. Now let us assume that t is a pure tree. Then there exist locations $l_1,\ldots,l_n$ and resources $m_1,\ldots,m_n$ such that $t \equiv [l_1]m_1|\ldots|[l_n]m_n$. If we consider the heap $h = [l_1 \mapsto m_1|l_2 \mapsto m_2|\ldots|l_n \mapsto m_n]$, we have $[\![h]\!]_{RT} \equiv t$.

Definition 25. *The* restriction of a tree t to a pure tree t_p *is defined such that, for any location l, if there exist $m \in Ints$ and t' such that $t = ([l]m|t')$, then there exists t'' such that $l \notin Loc_{t''}$ and $t_p = ([l]m|t'')$, else there is no t',t'' such that $t_p = ([l]t'|t'')$ (the location l does not exist in t_p).*

A separation logic for heaps. We show how the separation logic of [20], called PL, dedicated to pointers and heap manipulations, can be mapped into our assertion logic.
Let us first remind the syntax of PL: $\phi ::= \alpha \mid E \mapsto F \mid false \mid \phi \to \phi \mid \exists x.\phi \mid emp \mid \phi * \phi \mid \phi -\!\!* \phi$.
We have $\alpha ::= E = F \mid E < F$ where E,F are either a variable, an integer, or an expression $E+F$, $E-F$, $E\times F$. Moreover $false$ and emp respectively correspond to $\perp$ and I in our logic while $E \mapsto F$ ensures that the location E contains the value F.

The semantics is defined according to a satisfaction judgement $s,h \models \phi$ which means that an assertion holds for a given store s and heap h.

- $s,h \models \alpha$ iff $[\![\alpha]\!]_s = true$
- $s,h \models E \mapsto F$ iff $[\![E]\!]_s \in dom(h)$ and $h([\![E]\!]_s) = [\![F]\!]_s$
- $s,h \models false$ never
- $s,h \models \phi \rightarrow \phi'$ iff if $s,h \models \phi$ then $s,h \models \phi'$
- $s,h \models \exists\, x.\phi$ iff $\exists\, v \in Ints\; s.t.\; [s|x \mapsto v],h \models \phi$.
- $s,h \models emp$ iff $h = []$ is the empty heap
- $s,h \models \phi * \phi'$ if there exist h_0, h_1 (that are disjoint) such that $h_0 \times h_1 = h$ with $s,h_0 \models \phi$ and $s,h_1 \models \phi'$
- $s,h \models \phi -\!\!* \phi'$ if for all h', (where h' and h are disjoint) such that $s,h' \models \phi$ we have $s,h' \times h \models \phi'$.

where $h \times h$ denotes the union of disjoint heaps (union of functions with disjoint domains).
Now, given a formula ϕ of BI's pointer logic, its corresponding formula $[\![\phi]\!]_{BI-Loc}$ is defined as follows:

- $[\![E \mapsto F]\!]_{BI-Loc} = [E]F$;
- $[\![\phi \odot \phi']\!]_{BI-Loc} = [\![\phi]\!]_{BI-Loc} \odot [\![\phi']\!]_{BI-Loc}$ for $\odot \in \{\rightarrow, -\!\!*, *\}$;
- $[\![emp]\!]_{BI-Loc} = I$;
- $[\![false]\!]_{BI-Loc} = \bot$;
- $[\![\alpha]\!]_{BI-Loc} = \alpha$;
- $[\![\exists x.\phi]\!]_{BI-Loc} = \exists_{res} x.\phi$.

We then show that if a tree t satisfies an assertion ϕ on pointers, its pure restriction t_p also satisfies this assertion. For that, we study the cases which can lead to a *non-pure* tree.

Lemma 14. *Let t,t' be two trees, l be a location, and ϕ a PL formula such that $t = (t'|[l]e)$ and $l \notin Loc_t$, if $s,t \models [\![\phi]\!]_{BI-Loc}$ then $s,t' \models [\![\phi]\!]_{BI-Loc}$.*

Proof. By structural induction on ϕ. Let us show the main case $\phi \equiv \psi -\!\!* \psi'$:
We assume that $s,t \models [\![\psi -\!\!* \psi']\!]_{BI-Loc}$. Let t'' be a tree such that $s,t'' \models [\![\psi]\!]_{BI-Loc}$. By definition, we obtain $s,t''|(t'|[l]e) \models [\![\psi']\!]_{BI-Loc}$, and by induction hypothesis, $s,(t''|t') \models [\![\psi']\!]_{BI-Loc}$ and consequently $s,t' \models [\![\psi -\!\!* \psi']\!]_{BI-Loc}$.

Lemma 15. *Let t, t', t'' be resource trees, l_1, l_2 be locations such that $t = (t'|[l_1][l_2]t'')$ and ϕ a PL formula. If $s,t \models [\![\phi]\!]_{BI-Loc}$ then $s,t' \models [\![\phi]\!]_{BI-Loc}$.*

Proof. By structural induction on ϕ.

Lemma 16. *Let t,t' be resource trees and m be a resource such that $t = (m \mid t')$ and ϕ be a PL formula. If $s,t \models [\![\phi]\!]_{BI-Loc}$ then $s,t' \models [\![\phi]\!]_{BI-Loc}$.*

Proof. By structural induction on ϕ.

We can establish an important property about the relationships between BI's pointer logic (PL) and BI-Loc. The correspondence between formulae is such that if a tree satisfies $[\![\phi]\!]_{BI-Loc}$, we can find a corresponding pure tree which satisfies this formula:

Corollary 4. *Let t,t' be two trees, s be a stack, l be a location and ϕ be a PL formula, we suppose that $t = ([l]e|t')$ and $l \notin Loc_{t'}$. If $s,t \models [\![\phi]\!]_{BI-Loc}$ then $s,t' \models [\![\phi]\!]_{BI-Loc}$.*

Proof. Direct consequence of Lemmas 14, 15 and 16.

Then the satisfaction holds in the heap model if it holds in the associated resource tree model.

Lemma 17. *Let h be a heap, s be a stack and ϕ be a PL formula without $-\!\!*$ subformulae, we have $s,[\![h]\!]_{RT} \models [\![\phi]\!]_{BI-Loc}$ if and only if $s,h \models \phi$.*

Proof. By structural induction on ϕ.
- $\phi \equiv emp$: we assume that $s,[\![h]\!]_{RT} \models [\![emp]\!]_{BI-Loc}$. then $s,[\![h]\!]_{RT} \models I$. Thus $[\![h]\!]_{RT} = e$ and by definition h is the empty heap. The reverse implication is proved following the same schema.
- $\phi \equiv false$: the formula is never satisfied in both cases.
- $\phi \equiv \alpha$: $[\![\alpha]\!]_{BI-Loc} = \alpha$ and then the satisfaction of α only depends on s. Consequently, $s,h \models \alpha$ iff $s,[\![h]\!]_{RT} \models [\![\alpha]\!]_{BI-Loc}$.
- $\phi \equiv E \mapsto F$: we have $[\![E]\!]_s = l$ and $[\![F]\!]_s = m$. We assume $s,[\![h]\!]_{RT} \models [\![\phi]\!]_{BI-Loc}$, then $s,[\![h]\!]_{RT} \models [E]F$. Consequently, $[\![h]\!]_{RT} = [l]m$ and by definition the only heap which corresponds to $[l]m$ is $[l \mapsto m]$. As $s,[l \mapsto m] \models \phi$, we can conclude. The reverse implication is proved following the same schema.
- $\phi \equiv \psi \to \psi'$: by induction hypothesis, we have $s,h \models \psi$ iff $s,[\![h]\!]_{RT} \models [\![\psi]\!]_{BI-Loc}$ and $s,h \models \psi'$ iff $s,[\![h]\!]_{RT} \models [\![\psi']\!]_{BI-Loc}$. Consequently, $s,h \models \phi$ iff $s,[\![h]\!]_{RT} \models [\![\phi]\!]_{BI-Loc}$.
- $\phi \equiv \psi * \psi'$: we assume $s,[\![h]\!]_{RT} \models [\![\phi]\!]_{BI-Loc}$, so $s,[\![h]\!]_{RT} \models [\![\psi]\!]_{BI-Loc} * [\![\psi']\!]_{BI-Loc}$. As $[\![\psi]\!]_{BI-Loc}$ and $[\![\psi']\!]_{BI-Loc}$ correspond to PL formulae, they do not contain subformulae of the form $[l]I$. Consequently, by Corollary 4, there exist t,t' such that $[\![h]\!]_{RT} = (t|t')$ and $Loc_t \cap Loc_{t'} = \emptyset$. By Lemma 13 there exist h_0,h_1 that are disjoint and such that $[\![h_0]\!]_{RT} = t$ and $[\![h_1]\!]_{RT} = t'$. Consequently $s,h \models \phi$. The reverse implication is proved following the same schema.

Corollary 5. *Let h be a heap, s be a stack and ϕ be a PL formula without $\psi \to \psi'$ subformulae where ψ contains a $-\!\!*$ subformula. If $s,[\![h]\!]_{RT} \models [\![\phi]\!]_{BI-Loc}$ then $s,h \models \phi$.*

Proof. By Lemma 17, it is sufficient to prove the result for formulae of the form $\psi -\!\!* \psi'$. Let us assume that $s,[\![h]\!]_{RT} \models [\![\psi -\!\!* \psi']\!]_{BI-Loc}$. Given a heap h' such that $s,h' \models \phi$, by induction hypothesis we have $s,[\![h']\!]_{RT} \models [\![\psi]\!]_{BI-Loc}$. We deduce that $s,([\![h]\!]_{RT}|[\![h']\!]_{RT}) \models [\![\psi']\!]_{BI-Loc}$ and by induction hypothesis that $s,h \times h' \models \psi'$.

We cannot prove the reverse implication of Corollary 5 because of the $-\!\!*$ connective. Actually, the set of resource trees which satisfy $[\![\phi]\!]_{BI-Loc}$ can contain trees which are not pure. Therefore a resource tree t satisfies $[\![\phi -\!\!* \phi']\!]_{BI-Loc}$ if we only consider pure trees. In the general case, the formula $\phi -\!\!* \phi'$ stands in the pointer logic and not in BI-Loc.

Heap manipulation. We have defined in Section 6 a language dedicated to resource tree manipulations. We compare this language with the core system of heap manipulation. We also define each modification of a heap as a modification of the corresponding resource tree. The system we consider consists of the following commands and semantics [23]:

Content assignment

$$\frac{[\![E]\!]_s = l \in Loc, [\![F]\!]_s = m \in M}{[E] := F, s, h \rightsquigarrow s, [h|l \mapsto m]}$$

Dispose

$$\frac{[\![E]\!]_s = l \in Loc \cap dom(h)}{dispose(E), s, h \rightsquigarrow s, (h \setminus l)}$$

where $(h \setminus l)$ represents the heap h without the cell l.

Variable assignment

$$\frac{[\![E]\!]_s = m \in M}{x := E, s, h \rightsquigarrow [s|x \mapsto m], h}$$

Content Lookup

$$\frac{[\![E]\!]_s = l \in Loc, h(l) = m \in M}{x := [E], s, h \rightsquigarrow [s|x \mapsto m], h}$$

Cons

$$\frac{[\![E_i]\!]_s = m_i \in M, \forall i \in [\![0..n-1]\!]\ (l+i) \in Loc \cap dom(h)}{cons(E_1,\ldots,E_n), s, h \rightsquigarrow s, [h|l \mapsto m_1|\ldots|l+n-1 \mapsto m_n]}$$

The backward axioms that correspond to these commands expressed in the assertion language are:

$$\begin{array}{l}
\{\exists x.(E \mapsto x) * ((E \mapsto F) -\!\!* \phi)\}[E] := F\{\phi\}\\
\{\exists x.(E \mapsto x) * \phi\}dispose(E)\{\phi\}\\
\{\forall x'.((x' \mapsto E_1) * \ldots * ((x'+n-1) \mapsto E_n)) -\!\!* \phi\{^{x'}/_x\}\}x := cons(E_1,\ldots E_n)\{\phi\}\\
\{\phi\{^E/_x\}\}x := E\{\phi\}\\
\{\exists x'.\phi\{^{x'}/_x\} \wedge ((E \mapsto x') * \top)\}x := [E]\{\phi\}
\end{array}$$

where $E \mapsto ..$ indicates that the location E is in the heap.

The commands are quite similar to the ones that alter resource trees. Actually, for all those commands except *cons*, the correspondence $[\![\,.\,]\!]_{PT}$ is defined as follows:

$$\begin{array}{l}
[\![\,[E] := F\,]\!]_{PT} = update_{res}(E,F),\\
[\![\,dispose(E)\,]\!]_{PT} = dispose_{loc}(E),\\
[\![x := E\,]\!]_{PT} = x := E,\\
[\![x := [E]\,]\!]_{PT} = x := res(E).
\end{array}$$

Dealing with the *cons* command is almost impossible with the original manipulation language. Actually, in a resource tree, we can only create one location at a time. Furthermore, the choice of location names is arbitrary. Then we cannot ensure to build n adjacent cells with our actual language. Thus, we replace the *new* command by a command which builds n adjacent locations. Then we have the following axiom for resource trees:

$$\{\phi \wedge no(E:l) \wedge no(E:l+1) \wedge \ldots \wedge no(E:l+n-1)\}new_{loc}(n,E)\{\phi * [E][l]e * \ldots * [E][l+n-1]e\}$$

The correspondence is given by:

$$[\![x := cons(E_1,\ldots,E_n)\,]\!]_{PT} = x := new_{loc}(n,nil); update(x,E_1);\ldots; update(x+n-1,E_n);$$

It points out the major difference between the two languages. To create n adjacent heap entries with values $E_1,...,E_n$, we must first create these locations below the root of the tree with the *new* axiom and then put the right values inside them with the *update* command. The semantics of the command is such that we can ensure that if we execute a program on a heap and if we execute the corresponding program on the corresponding resource tree, we have the following result:

Lemma 18. *Let C,s,h be a configuration that is not stuck and s',h' be a configuration such that $C,s,h \rightsquigarrow s',h'$, the configuration $[\![C]\!]_{PT}, s, [\![h]\!]_{RT}$ is not stuck and if $[\![C]\!]_{PT}, s, [\![h]\!]_{RT} \rightsquigarrow s'', t$ then we have $s' = s''$ and $[\![h']\!]_{RT} = t$.*

Proof. Direct consequence of the command semantics.

We have a correspondence for both heap model and pointer logic with resource tree model and BI-Loc. Thus, BI-Loc and resource trees can be used to check properties on heaps. More formally, we have the following result:

$$
\begin{array}{ll}
\{\phi_0\} & \\
x := cons(a,a); & \\
\{\phi_1\} & \phi_0 \equiv \forall x'((x' \mapsto a) * ((x'+1) \mapsto a)) -\!\!* \phi_1\{^{x'}/_x\} \\
y := cons(b,b); & \phi_1 \equiv \forall y'((x' \mapsto b) * ((y'+1) \mapsto b)) -\!\!* \phi_2\{^{y'}/_y\} \\
\{\phi_2\} & \phi_2 \equiv \exists z((x+1) \mapsto z) * (((x+1) \mapsto t_x) -\!\!* \phi_3) \\
[x+1] := t-x; & \phi_3 \equiv \exists z'.((y+1) \mapsto z') * (((y+1) \mapsto x-t) -\!\!* \phi_f) \\
\{\phi_3\} & \phi_f \equiv (x \mapsto a) * ((x+1) \mapsto t-x) * (y \mapsto b) * ((y+1) \mapsto x-t) \\
[y+1] := x-t; & \\
\{\phi_f\} &
\end{array}
$$

Fig. 4. An Example

Theorem 10. *Let ϕ and ϕ' be two assertions and C be a program with heaps and pointers,*
(i) if ϕ and ϕ' do not contain $-\!\!*$ subformulae then $\{[\![\phi]\!]_{BI-Loc}\}[\![C]\!]_{PT}\{[\![\phi']\!]_{BI-Loc}\}$ if and only if $\{\phi\}C\{\phi'\}$.
(ii) if ϕ and ϕ' do not contain subformulae $\phi_1 \rightarrow \phi_2$ in which ϕ_1 contains $-\!\!*$ subformulae and if $\{[\![\phi]\!]_{BI-Loc}\}[\![C]\!]_{PT}\{[\![\phi']\!]_{BI-Loc}\}$ then we have $\{\phi\}C\{\phi'\}$.

Proof. By case analysis on C.
The proof of (i) is a direct consequence of Lemma 17 and Lemma 18.
The proof of (ii) is a direct consequence of Corollary 5 and Lemma 18.

An example We give here a simple example of pointer program from [23], presented in Figure 4 that illustrates the previous points.

The assertions are defined from backward axioms for this program and the code and Hoare triples are given. From these data we built the equivalent program and Hoare triples in the model of partial resource trees. Then we use the correspondence previously defined.
We now present, in the following table, the commands on pointers on the left-hand side and commands on resource trees on the right-hand side.

Pointer manipulation	**Resource tree manipulation**
$x := cons(a,a);$	$x := new_{loc}(2,nil); update(x,a); update(x+1,a);$
$y := cons(b,b);$	$y := new_{loc}(2,nil); update(y,b); update(y+1,b);$
$[x+1] := t-x;$	$update_{res}(x+1,t-x);$
$[y+1] := x-t;$	$update_{res}(y+1,x-t);$

Moreover we define assertions for resource trees. We start with the BI-Loc formula that is equivalent to ϕ_f, namely, $[\![\phi_f]\!]_{BI-Loc} = [x]a * [x+1](t-x) * [y]b * [y+1](x-t)$. From this formula we consider backward axioms previously defined and, command by command, we create the following assertions:

$$
\begin{array}{l}
\{\phi'_0\}\\
x := new_{loc}(2, nil);\\
\{\phi'_1\}\\
update_{res}(x, a);\\
\{\phi'_2\}\\
update_{res}(x+1, a);\\
\{\phi'_3\}\\
y := new_{loc}(2, nil);\\
\{\phi'_4\}\\
update_{res}(y, b);\\
\{\phi'_5\}\\
update_{res}(y+1, b);\\
\{\phi'_6\}\\
update_{res}(x+1, t-x);\\
\{\phi'_7\}\\
update_{res}(y+1, x-t);\\
\{[\![\phi_f]\!]s\}
\end{array}
$$

$$
\begin{array}{lcl}
\phi'_0 & \equiv & \forall x'.((no(x') \wedge no(x'+1)) \rightarrow (([x']I * [x'+1]I) \mathrel{-\!\!*} \phi'_1\{^{x'}/_x\}))\\
\phi'_1 & \equiv & \exists_{res} z.[x]z * (\forall_{res} z'.((z' = e) \vee (([x]z' * \top) \rightarrow \bot)) \wedge ([x]a \mathrel{-\!\!*} \phi'_2))\\
\phi'_2 & \equiv & \exists_{res} z.[x+1]z * (\forall_{res} z'.((z' = e) \vee (([y+1]z' * \top) \rightarrow \bot))\\
 & & \qquad \wedge ([x+1]a \mathrel{-\!\!*} \phi'_3))\\
\phi'_3 & \equiv & \forall y'.((no(y') \wedge no(y'+1)) \rightarrow (([y']I * [y'+1]I) \mathrel{-\!\!*} \phi'_4\{^{y'}/_y\}))\\
\phi'_4 & \equiv & \exists_{res} z.[y]z * (\forall_{res} z'.((z' = e) \vee (([y]z' * \top) \rightarrow \bot)) \wedge ([y]b \mathrel{-\!\!*} \phi'_5))\\
\phi'_5 & \equiv & \exists_{res} z.[y+1]z * (\forall_{res} z'.((z' = e) \vee (([y+1]z' * \top) \rightarrow \bot))\\
 & & \qquad \wedge ([y+1]b \mathrel{-\!\!*} \phi'_6))\\
\phi'_6 & \equiv & \exists_{res} z.[x+1]z * (\forall_{res} z'.((z' = e) \vee (([x+1]z' * \top) \rightarrow \bot))\\
 & & \qquad \wedge ([x+1](t-x) \mathrel{-\!\!*} \phi'_7))\\
\phi'_7 & \equiv & \exists_{res} z.[y+1]z * (\forall_{res} z'.((z' = e) \vee (([y+1]z' * \top) \rightarrow \bot))\\
 & & \qquad \wedge ([y+1](x-t) \mathrel{-\!\!*} [\![\phi_f]\!]s))\\
[\![\phi_f]\!]s & \equiv & [x]a * [x+1](t-x) * [y]b * [y+1](x-t)
\end{array}
$$

If we compare axioms of Figure 4 with these ones we observe that axioms for update of resources are more complex in the latter. That is due to the fact that, in the general definition of resource trees, the separation into two subtrees does not ensure that they are disjoint. For pointers, this point is not central because a location can only contain one non-decomposable resource, but we will see in the next section that such a property can be helpful. The example was mentioned in [23] in a different form and does not use backward axioms but constructive axioms in which postconditions are given from preconditions. One main interest of resource tree models is that, in order to verify logical consequences we could consider proof-search methods extended from the ones recently proposed for separation logics [16,17].

Frame property. We can express Hoare triples of heaps commands through BI-Loc and resource trees. Then we can wonder why the frame rule problem with the new_{loc} command discussed in previous sections do not occur here. The initial problem is that, in the general case, the *new* command will not ensure freshness of the created location if we extend the context. The differences between both formalisms are mainly due to restriction on the creation commands.

First, the command which creates new heap entries builds an entry with a resource inside and this resource cannot be e. Secondly, we can prove that there is no formula ψ in pointer logic such that $[\![\psi]\!]_{BI-Loc}$ requires an empty location. It means that, if $s, (t|[L]e) \models [\![\psi]\!]_{BI-Loc}$ then $s, t \models [\![\psi]\!]_{BI-Loc}$.

Consequently, if we use the command *cons* in a larger context, by adding a formula ψ in the preconditions, we have three possibilities:

1. The created cell does not exist in a tree that satisfies $[\![\psi]\!]_{BI-Loc}$. Then, we cannot alter $[\![\psi]\!]_{BI-Loc}$ from the application of this new command.
2. The created cell exists in a tree that satisfies $[\![\psi]\!]_{BI-Loc}$ and contains resources different of e. Then, the resource tree is undefined and the postcondition is always satisfied.
3. The created cell exists in a tree that satisfies $[\![\psi]\!]_{BI-Loc}$ and only contains e. Then, due to the above remark, the same resource tree without this location still ensures $[\![\psi]\!]_{BI-Loc}$ and then this tree is not modified by other commands.

9.2 Resource Trees and Permission Accounting

Recently, Bornat et al. have studied the problem of permission accounting in the context of separation logics [5]. They extend the initial heap model in order to handle permission access to variables. A weight is

associated to each $\mapsto$ in order to fix the right associated to this relation. For example, $l \overset{1}{\mapsto} m$ indicates that we can read and write into variable l and the content of this variable is m, while $l \overset{x}{\mapsto} m$ with $0 < x < 1$ means that we can only read the value of l.
Our objective here is to show how we can naturally express this model from resource trees. Moreover, we show that such a representation and the use of BI-Loc allows to naturally solve an open problem mentioned in [5]: how to represent trees with permissions in separation logics ?

The general model of permission accounting is defined as follows: $Heaps = L \rightharpoonup_{fin} (V \times M)$. Thus we associate to a location l a pair (v,m) where v is the value attached to l and m is the access permission to l. Then L is a set of locations, V a set of values and M a partial commutative semi-group.
We define the composition on $V \times M$ by extension of the composition on M. Then we have $(v,m) \times (v',m') = (v, m \times m')$ if $v = v'$ and $m \times m'$ is defined.
In addition, the composition of heaps $h \times h'$ with $h, h' \in Heaps$ is defined as follows:
$i) dom(h \times h') = dom(h) \cup dom(h')$ and
$$\begin{aligned} ii) \forall l \in dom(h \times h')(h \times h')(l) &= h(l)\ if\ h'(l)\ undefined, \\ &= h'(l)\ if\ h(l)\ undefined, \\ &= h(l) \times h'(l)\ otherwise. \end{aligned}$$

Let us observe that we now authorize the composition $(l \mapsto v,m) \times (l \mapsto v,m') = (l \mapsto v, m \times m')$ of two heaps which do not have disjoint domains.

Our first goal is to show how to represent such a structure with resource trees. Then we define e such that for all $(v,m) \in V \times M$ we have $(v,m) \times e = e \times (v,m) = (v,m)$. Then $V \times M \cup \{e\}$ is a partial monoid and we can use it to build resource trees. A key point is that the behavior of the resource tree composition is exactly the composition of heaps with permissions. We can express a heap entry $l \mapsto (v,m)$, also denoted $l \overset{v}{\mapsto} m$ as the resource tree $[l](v,m)$, while e corresponds to the empty heap. More generally, given a heap with permissions h, we denote its corresponding resource tree $[\![h]\!]$. We can wonder how $[l]e$ can be interpreted in the permission context. As we have $[l](v,m) = [l](v,m) \times [l]e$, we consider that $[l]e$ ensures that l is defined but that we cannot access or write anything on it.
Having this representation, we now consider an unsolved problem mentioned in [5]: how to represent trees with permissions. A first attempt consists of the following separation logic formulae initially for specifying a tree with permission z:

$$\begin{aligned} &ztree\ z\ nil\ Empty &&= emp \\ &ztree\ z\ t\ (Tip\ \alpha) &&= t \overset{z}{\mapsto} (0,\alpha,0) \\ &ztree\ z\ t\ (Node\ \gamma\rho) &&= \exists\, l.\ \exists\, r.\ \ t \overset{z}{\mapsto} (1,l,r) * ztree\ z\ l\ \gamma * ztree\ z\ r\ \rho \end{aligned}$$

In this representation, the empty tree ($ztree\ z\ nil\ Empty$) corresponds to the empty heap. The leaf with the value α ($ztree\ z\ t\ (Tip\ \alpha)$) corresponds to the heap entry $t \overset{z}{\mapsto} (0,\alpha,0)$. At the node t, we find the triple $(0,\alpha,0)$ with the permission z. The two values 0 in the triple indicate that this node is a leaf, α is the value at the leaf. The node with children γ, ρ ($ztree\ z\ t\ (Node\ \gamma\rho)$) corresponds to a heap entry $t \overset{z}{\mapsto} (1,l,r)$ which indicates that the current node t has children (the value 1 in the triple) that are in two sub-heaps.
We do have $ztree\ z + z'\ t\ \tau \equiv ztree\ z\ t\ \tau * ztree\ z'\ t\ \tau$ but a key point is that $*$ does not guarantees that the subparts of the formula are disjoint. Such a problem is not new in the perspective of resource trees. The solution we propose is based on the formula $no(l)$, defined in Section 3 as $(\top * [l]\top) \rightarrow \bot$, which expresses that a location l is not in a location or a subtree.
Here we use this formula to ensure that the left child and the right child of a given node do not share a given location. With this definition, we can propose a new simple way to express trees with permissions by using resource trees:

$$
\begin{array}{ll}
ztree'\ z\ nil\ Empty & = e \\
ztree'\ z\ t\ (Tip\ \alpha) & = [t]((0,\alpha,0),z) \\
ztree'\ z\ t\ (Node\ \gamma\rho) & = \exists_{loc}\ l.\ \exists_{loc}\ r.\ \ ([t]((1,l,r),z) \wedge no(l) \wedge no(r)) * \\
 & \quad disjoint((ztree\ z\ l\ \gamma, ztree\ z\ r\ \rho))
\end{array}
$$

where, for two assertions ϕ and ϕ', $disjoint(\phi,\phi') \equiv (\phi * \phi') \wedge \forall_{loc} l.((\phi * [l]e) \wedge (\phi' * [e]) \to \perp)$.
The formula $no(l)$ is used in fact to avoid cyclic links in the tree: a node cannot be its own child. Then the formula $disjoint(\phi,\phi')$ is satisfied if there exist two subtrees that are disjoint, one satisfying ϕ and the other satisfying ϕ'. It is used to ensure that a given node of one subtree is not a child of the other subtree: the subtrees do not share location names.

10 Conclusion and perspectives

This work emphasizes that BI logic and its based-on relations and monoids resource semantics [19] is a logical kernel from which some spatial and separation logics are defined, in order to reason about various data structures or resource distribution and interactions. In this context, we define a particular data model, called resource tree, in which nodes contain resources that are elements of a partially defined monoid and also a new separation logic, denoted BI-Loc, with a modality for locations. From these new model and logic and their properties, we propose a language dedicated to the management of resource trees and their transformations and also define an assertion logic based on BI-Loc and its related axioms. In order to illustrate these proposals and results, we show how heaps and pointer models can be studied and analyzed from based-on resource trees representation and how we can solve some related problems.
At the present state, data representation is static because the resources, distributed in the tree, cannot move from their initial location. A first attempt could be to extend our logic with movement modalities as it has been done for modeling distributed and mobile systems within linear logic [3]. Moreover, we have in mind to study validity and reasoning in some separation and spatial logics starting from our approach and also comparisons with related works in hybrid logics [2,15]. In this context we could study hierarchical storage in a resource tree perspective. Recently, Ahmed et al. have presented a new substructural logic that encode invariants necessary for reasoning about hierarchical storage [1]. They showed how this logic can be used to described the layout of bits in a memory word, the layout of memory words in a region, and the layout of regions in an address space. Their results are illustrated with a simplified version of the abstract machine used in the ML Kit with regions. In fact, their model and logic are strongly related to the resource tree model and BI-Loc even if we cannot directly handle with infinite trees and we do not have an operator for adjacency. We could study how to reason about the abstract machine from resource trees and also then complete the results of [1] by providing a Hoare triple semantics for the abstract machine.
By studying relationships with other recent works like context tree logic [8], with a focus on representation of complex data inside nodes, we could provide a model which uniformly handles all attributes in semi-structured data, while other models seem restricted to identifiers and particular attributes. Another recent work proposes bigraphs as a model for structures in global computing that make clear the difference between structural separation and name separation [22]. Then comparisons of our work with this model and its underlying spatial logics [14] could be fruitful. A last important point to develop is the proposal, from our results in BI [16,19] or BI's pointer logic [17], of proof theories and theorem-proving methods in separation logics, with a particular focus on countermodel generation [18]. It is essential in the perspective to decide the validity of properties concerning distribution and mobility aspects in our models but also in spatial logics for concurrency, graphs or trees [6,8,10].

References

1. A. Ahmed, L. Jia, and D. Walker. Reasoning about hierarchical storage. In *18th Symposium on Logic in Computer Science*, Ottawa, Canada, 2003.

2. C. Areces, P. Blackburn, and M. Marx. Hybrid logics: characterization, interpolation and complexity. *Journal of Symbolic Logic*, 66(3):977–1010, 2001.
3. N. Biri and D. Galmiche. A modal linear logic for distribution and mobility. In *FLOC'02 Workshop on Linear Logic*, July 2002. Copenhagen, Danemark.
4. N. Biri and D. Galmiche. A Separation Logic for Resource Distribution. In *23rd Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'03, LNCS 2914*, pages 23–37, December 2003. Mumbai, India.
5. R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *The 32nd Annual Symposium on Principles of Programming Languages, POPL'05*, Long Beach, California, January 2005.
6. L. Caires and L. Cardelli. A spatial logic for concurrency (part II). In *Int. Conference on Concurrency Theory, CONCUR 2002, LNCS 2421*, pages 209–225, 2002.
7. C. Calcagno, L. Cardelli, and A. Gordon. Deciding validity in a spatial logic for trees. In *ACM Sigplan Workshop on Types in Language Design and Implementation, TLDI'03*, New Orleans, USA, 2003.
8. C. Calcagno, Ph. Gardner, and U. Zarfaty. Context logic and tree update. In *The 32nd Annual Symposium on Principles of Programming Languages, POPL'05*, Long Beach, California, 2005.
9. C. Calcagno, H. Yang, and P. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *21st Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'01, LNCS 2245*, pages 108–119, Bangalore, India, 2001.
10. L. Cardelli, Ph. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *Int. Conference on Automata, Langages and Programming, ICALP'02, LNCS 2380*, pages 597–610, 2002.
11. L. Cardelli and G. Ghelli. A query language for semistructured data based on the ambient logic. *Math. Struct. in Comp. Science*, 14(3):285–327, 2004.
12. L. Cardelli and A.D. Gordon. Anytime, anywhere - modal logics for mobile ambients. In *27th ACM Symposium on Principles of Programming Languages, POPL 2000*, pages 1–13, Boston, USA, 2000.
13. W. Charatonik and J.M. Talbot. The decidability of model checking mobile ambients. In *15th Int. Workshop on Computer Science Logic, CSL 2001, LNCS 2142*, pages 339–354, Paris, France, 2001.
14. G. Conforti, D. Macedonio, and V. Sassone. Spatial logics for bigraphs. In *Int. Colloquium on Automata, Languages and Programming, ICALP'05, LNCS 3580*, pages 766–778, Lisboa, Portugal, 2005.
15. M. Franceschet and M. de Rijke. Model checking for hybrid logics (with an application to semi-structured data). *Journal of Applied Logic*, 4(3):279–304, 2006.
16. D. Galmiche and D. Méry. Semantic labelled tableaux for propositional BI without bottom. *Journal of Logic and Computation*, 13(5):707–753, 2003.
17. D. Galmiche and D. Méry. Characterizing provability in BI's pointer logic through resource graphs. In *Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2005, LNAI 3835*, pages 459–473, Montego Bay, Jamaica, December 2005.
18. D. Galmiche and D. Méry. Resource graphs and countermodels in resource logics. *Electronic Notes in Theoretical Computer Science*, 125(3):117–135, 2005.
19. D. Galmiche, D. Méry, and D.Pym. The semantics of BI and Resource Tableaux. *Math. Struct. in Comp. Science*, 15(6):1033–1088, 2005.
20. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *28th ACM Symposium on Principles of Programming Languages, POPL 2001*, pages 14–26, London, UK, 2001.
21. N. Kobayashi, T. Shimizu, and A. Yonezawa. Distributed concurrent linear logic programming. *Theoretical Computer Science*, 227(1-2):185–220, 1999.
22. R. Milner. Bigraphical reactive systems. In *Int. Conference on Concurrency Theory, CONCUR 2001, LNCS 2154*, pages 16–35, 2001.
23. P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *15th Int. Workshop on Computer Science Logic, CSL 2001, LNCS 2142*, pages 1–19, Paris, France, 2001.
24. P.W. O'Hearn and D. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.
25. D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002.
26. J. Reynolds. Separation logic: A logic for shared mutable data structures. In *IEEE Symposium on Logic in Computer Science*, pages 55–74, Copenhagen, Danemark, July 2002.
27. B. A. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR*, 70:569–572, 1950.