Tableaux and Resource Graphs for Separation Logic

Didier Galmiche and Daniel Méry

LORIA - Université Henri Poincaré
Campus Scientifique, BP 239
Vandœuvre-lès-Nancy, France

Abstract Separation Logic (SL) is often presented as an assertion language for reasoning about mutable data structures. As recent results about verification in SL have mainly been achieved from a model-checking point of view, our aim in this paper is to study Separation Logic from a complementary proof-theoretic perspective in order to provide results about proof-search in SL. We begin our study with a fragment of SL, denoted SLP, where first order quantifiers, variables and equality are removed. We first define specific structures, called *resource graphs*, that capture SLP models by considering heaps as resources via a labelling process. We then provide a tableau calculus that allows us to build such resource graphs from which either proofs, or countermodels can be generated. We finally prove soundess, completeness and termination of our tableau calculus before discussing extensions to various fragments of SL (including full SL) and the related decidability issues.

1 Introduction

Separation Logic (SL) and its variants are logics for reasoning about mutable data structures in which the pre- and postconditions are written using specific forms of conjunction or implication. In this context, Reynolds initially proposed an intuitionistic logic extended with a separation connective $*$ [25] and Ishtiaq and O'Hearn then investigated the same approach from the point of view of the logic of Bunched Implications (BI) [23], which allows a joint and modular treatment of intuitionistic connectives (additive implication $\rightarrow$ and conjunction $\wedge$) and linear connectives (multiplicative implication $-\!*$ and conjunction $*$). The resource interpretation of BI's connectives, where $*$ decomposes the current resource into pieces and $-\!*$ talks about new and fresh resource, is central.
Separation Logic, also denoted BI's pointer logic, provides an actual way of understanding the separating conjunction $*$ and implication $-\!*$ in the context of program verification apart from logical concerns [18]. From a semantic point of view, the models of SL are very specific models of Boolean BI (BBI) that validate the axioms for Hoare triples and in which the additive connectives are classical. Since Separation Logic is mainly used as an assertion language for specifying properties about programs that manipulate dynamically-allocated linked data structures, it is essential to provide methods and tools for verifying or proving SL specifications. As the existing works on automated assertion checking in SL are mainly based on a model-checking approach [2,26], we aim at developing an alternative approach based on theorem-proving in order to provide methods and tools for proof-search and countermodel generation in SL. Knowing that many other spatial logics [12,11,8], for which the model-checking approach is the only approach currently available, can also be

seen as extensions or specializations of SL, we expect to develop similar theorem-proving methods and tools for these logics too. Moreover, with our results on SL as a starting point, studying the combination of both approaches could be fruitful in order to improve the verification of properties expressed in various spatial logics.

The central contribution of this paper is a new characterization of validity in SL based on a labelled tableaux calculus that builds specific structures, called resource graphs, from which validity can be analyzed and countermodels can be generated. A few preliminary and incomplete results about this tableau calculus have already been presented in [15] but in the present paper we give the complete presentation, with refined concepts, more proofs and new results about countermodel generation, termination and decidability issues.

One challenge of this work is to investigate whether the general methodology applied for the standard version of BI [17], which admits intuitionistic additive connectives, can also be applied to SL. The results presented in this paper show that using a general methodology based on the design of an appropriate labelling algebra allows us to capture, in a non-trivial way, the essential properties of SL models inside specific structures called resource graphs from which validity, entailment and countermodel contruction in SL can be analyzed.

Starting from a propositional fragment of SL called SLP which contains no variables and no equality predicates, we define the central notion of resource graphs and provide a tableaux calculus that builds such resource graphs in parallel of the tableau construction process. This calculus represents a first theorem-proving alternative to the existing model-checking approaches w.r.t. the verification problem in SL. The main properties of the calculus, namely soundness, completeness and termination are then proved with an emphasis on how to generate countermodels from resource graphs when proof-search fails. Having in mind that full SL is not decidable, we study various extensions of SLP including first order quantifiers, variables and equality predicates and finally discuss the known decidability results and issues [9] using alternative proof-theoretic arguments.

In Section 2 we recall the main characteristics of Separation Logic and more particularly its language and semantics. With its local form of reasoning, SL is a logic for specifying and verifying properties of programs that manipulates dynamically-allocated linked data structures in a much simpler way than previous formalisms [5,3,13]. Knowing that existing works on automated assertion checking in SL are mainly based on a model-checking approach [2,26], our aim is to devise verification methods based on a theorem-proving approach. We view our work on SL as a first important step toward new proof-search methods for related spatial logics based on ambient or tree models [12,11,8] for which only model-checking techiques have been developed so far.

In Section 3 we focus on heaps, labels and resource graphs. Although a similar approach has already been successfully applied to BI, one challenge of this work is to study if the same general methodology also applies to SL. Two main obstacles arise:

- the first one is that the additive connectives of SL are classical, so that most of the crucial arguments relying on the intuitionistic nature of BI can no longer be applied;
- the second one lies in the fact that heap composition in SL interacts with the points-to predicate $\mapsto$ and admits subtle properties that require non-trivial adaptations to the initial notion of BI's resource graphs in order to capture entailment in SL appropriately.

For mixed resource logics such as BI or SL, the main principle of the methodology consists in reflecting the semantic relation of entailment at a syntactic level via a labelling process that results in an algebra of labels and label constraints obtained by means of a particular closure operator. The closure of a set of labels and label constraints then leads to the notion of resource graphs from which the validity of SL formulas can be analyzed. Thus, our first contribution is the definition of an appropriate notion of resource graphs for SL that captures the the properties of heap composition w.r.t. points-to predicates.

In Section 4 we present the main contribution which is a labelled tableaux calculus for SLP that builds resource graphs in parallel of the tableau construction process. This calculus represents a first theorem-proving alternative to the existing model-checking approaches, the choice of a tableau representation being justified by its known ability to ease counter-model construction. After an overview of the various kinds of expansion rules and their impact on resource graph construction, we introduce a specific notion of measure in order to determine how many cells are assumed to be actually present in the heap denoted by a given node in a resource graph. A key point here is that provability in SLP is characterized with a clear distinction between two distinct notions: structural and logical consistency. The former means that a resource graph actually represents a model and the latter means that a formula can be falsified in some model. We are quite confident that validity in some other resource logics that are closely related to SL can also be characterized through an appropriate notion of resource graphs using a similar approach. Therefore, we view our work on SLP as a first step toward providing proof-theoretic foundations and tableau-based calculi to a wide range of separation logics that deal with semi-structured data.

In Section 5 we prove the main properties of the tableaux calculus. We first prove soundness by introducing the size of a heap, which is the number of locations it contains. Then we proceed with the completeness proof, which relies on the effective construction of a countermodel from an open tableau branch. To complete these results we develop the main arguments that justify the termination of the tableau method for SLP from which we can deduce the well-known decidability and complexity results about the propositional fragment of SL [9] using alternative proof-theoretic arguments.

In Section 6 we study a few extensions of SLP including first order quantifiers, variables and the equality predicate. In order to deal with quantifiers, we use a standard technique that eliminates variables by instantiating parameters depending on the sign of a signed formula [14]. Equality is handled via (plain syntactic) term unification and requires modifying the condition for a closed tableau branch. Having in mind that SL is not decidable, we finally discuss decidability issues in a particular fragment of SL where memory cells are restricted to single values instead of pairs of values.

In Section 7 we summarize the results and develop some perspectives. We aim at applying the same proof-theoretic approach to the affine fragment of SL [18] which admits intuitionistic additives and allows one to write interesting properties about sharing. Moreover some spatial logics for trees or graphs [10] being strongly related to SL and Boolean BI we aim at studying these logics from our proof-theoretic perspective as an alternative and complementary basis to the model-checking approach. Moreover, comparisons with existing works on theorem proving and pointer programs [19,21,24] could lead to fruitful refinements of our results.

2 Separation Logic and Verification

Separation Logic (SL) is a logic for reasoning about mutable data structures and can be viewed as a specific extension of Hoare Logic in which pre- and post-conditions are written in the logic of Bunched Implications (BI) [23,25] [1]. BI allows one to write statements that mix the (additive) connectives of intuitionistic logic ($\wedge$, $\vee$, $\rightarrow$) with the (multiplicative) connectives of intuitionistic linear logic ($*$, $-\!\!*$). Accordingly, SL includes

- a spatial form of (separating) conjunction $*$ that splits a heap into distinct subheaps;
- a spatial form of (separating) implication $-\!\!*$ that describes fresh pieces of heaps;
- and a form of assertion that allows one to make statements about the content of a heap cell via the points-to predicate $\mapsto$.

Various problems about pointer management, including aliasing, are difficult to address with standard approaches and the correctness of a program that mutates data structures defined via pointers (such as linked-lists) usually depends on making restrictions on the sharing inside such data structures. In order to deal with such problems, Separation Logic admits simple axioms that capture the intuitive operational locality of assignment [22]. Separation logics usually comes in two flavours depending on the nature of the additive connectives $\wedge$ and $\rightarrow$ [25]:

- the first one uses intuitionistic additives and includes a monotonicity property which ensures that an assertion which holds for a portion of a storage still holds in any extension of this portion;
- the second one does not have the previous monotonicity property as it uses classical additives, but allows reasoning about explicit storage deallocation.

The underlying idea of separation used in SL can be generalized to describe the separation of other kinds of resources (*e.g.*, trees, graphs or processes [8,10,12]).

2.1 Separation Logic: Syntax and Semantics

In this section we summarize the main notions and results about SL. Let us begin with

- a countable set Loc of locations $l, k, m, l_1, k_1, m_1, \ldots$
- a countable set Cst of constants $nil, a, b, c, a_1, b_1, c_1, \ldots$
- a countable set Var of variables $x, y, z, x_1, y_1, z_1, \ldots$ and
- a countable set $Val = \mathbb{N} \cup Cst \cup Loc$ of values (integers, constants or locations).

We then define a *stack* s as a finite partial function $Var \rightharpoonup_{fin} Val$ that associates values to variables and a *heap* h as a finite partial function $Loc \rightharpoonup_{fin} Val \times Val$ that associates pairs of values. We respectively write $\mathcal{H}$ and $\mathcal{S}$ to denote the set of all heaps and the set of all stacks.
An expression E can be either a value or a variable or that can be interpreted as a value w.r.t. a stack s ($[\![E]\!]_s \in Val$). Variables that are bound by a stack are called *stack variables.*

[1] Separation Logic was indeed initially called BI's Pointer Logic [18].

Definition 2.1. *The language of* SL *consists of the points-to predicate* $\mapsto$*, the equality predicate* $=$*, the existential quantifier* $\exists$*, the propositional connectives of* BI *and the sets* Var*,* Val *and* $\mathcal{S}$ *of variables, values and stacks.*
SL *formulas are then inductively defined by the following grammar:*

- $At ::= (E \mapsto E_1, E_2) \mid E_1 = E_2$ *where* E*,* E_1 *and* E_2 *are expressions,*
- $\phi ::= At \mid \mathrm{I} \mid \phi * \phi \mid \phi \mathrel{-\!\!*} \phi \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid \phi \vee \phi \mid \exists x.\phi.$

Moreover, negation $\neg\phi$ is defined as a shorthand for $\phi \rightarrow \bot$. Since additive connectives in SL are classical, one could also take negation as primitive and define implication $\phi \rightarrow \psi$ as a shorthand for $(\neg\phi) \vee \psi$.
Given a stack s, the points-to predicate $(x \mapsto a, b)$ allows us to represent the state of the memory: there exists a location l in a heap h such that $[\![\, x \,]\!]_s = l$ and $h(l) = (a, b)$.
The use of pairs of values follows the original presentations of Separation Logic [22,25]. Recent works show that restricting to single values does not improve the decidability status of SL [7], which remains undecidable, but surely harms expressive power: for example, one can still describe a linked-list structure

$$(x_1 \mapsto x_2) * (x_2 \mapsto x_3) * \ldots * (x_{n-1} \mapsto x_n),$$

but can no longer state properties about the values contained in the nodes of the list as in

$$((x_1 \mapsto a_1, x_2) * (x_2 \mapsto a_2, x_3) * \ldots * (x_{n-1} \mapsto a_n, x_n)) \wedge (a_1 = a_n).$$

Since the actual use of SL mainly consists in writing formulas that describe specific properties of memory states and verifying that these properties hold in some model of the logic, we need to define precisely what a model of SL looks like.
We write e to denote the empty heap (the domain of which is empty) and we say that two heaps h_1 and h_2 are disjoint, written $h_1 \# h_2$, iff the domains of h_1 and h_2 are disjoint (more formally, iff $dom(h_1) \cap dom(h_2) = \emptyset$). We then define heap composition $(\cdot)$ as the function $\mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$ wich maps two disjoint heaps h_1 and h_2 to the heap $h_1 \cdot h_2$ given by the union of the two functions h_1 and h_2. A key point her is that composition of non-disjoint heaps is undefined so that heap composition is only a partial function.
In order to interpret SL formulas we define a forcing relation $(s, h) \models \phi$ which asserts that the formula ϕ is true in a stack $s \in \mathcal{S}$ and a heap $h \in \mathcal{H}$. It is required that the free variables of ϕ should be included in the domain of s.

Definition 2.2. *For all stacks* $s \in \mathcal{S}$ *and heaps* $h \in \mathcal{H}$*, the semantics of* SL *formulas is inductively defined as follows:*

- $(s, h) \models \top$ *always;*
- $(s, h) \models \bot$ *never;*
- $(s, h) \models E_1 = E_2$ *iff* $[\![\, E_1 \,]\!]_s = [\![\, E_2 \,]\!]_s$*;*
- $(s, h) \models (E \mapsto E_1, E_2)$ *iff* $dom(h) = \{\, [\![\, E \,]\!]_s \,\}$ *and* $h([\![\, E \,]\!]_s) = ([\![\, E_1 \,]\!]_s, [\![\, E_2 \,]\!]_s)$*;*
- $(s, h) \models \phi \wedge \psi$ *iff* $(s, h) \models \phi$ *and* $(s, h) \models \psi$*;*
- $(s, h) \models \phi \vee \psi$ *iff* $(s, h) \models \phi$ *or* $(s, h) \models \psi$*;*
- $(s, h) \models \phi \rightarrow \psi$ *iff* $(s, h) \models \phi$ *implies* $(s, h) \models \psi$*;*

- $(s, h) \models \mathrm{I}$ *iff* h *is the empty heap* e*;*
- $(s, h) \models \phi * \psi$ *iff* $\exists\, h_1, h_2 \in \mathcal{H}.\ h_1 \# h_2,\ h_1 \cdot h_2 = h,\ (s, h_1) \models \phi$ *and* $(s, h_2) \models \psi$*;*
- $(s, h) \models \phi \mathrel{-\!\!*} \psi$ *iff* $\forall\, h_1 \in \mathcal{H}.$ *if* $h_1 \# h$ *and* $(s, h_1) \models \phi$ *then* $(s, h_1 \cdot h) \models \psi$*;*
- $(s, h) \models \exists\, x.\phi$ *iff* $\exists\, v \in Val.\ ([\, s \mid x \mapsto v\,], h) \models \phi$.

Let us remark that the classical nature of the additive connectives in SL is captured via a pointwise interpretation of the additive implication $\to$. An intuitionistic version of $\phi \to \psi$ would require that $(s, h') \models \phi$ should imply $(s, h') \models \psi$ for all heaps h' extending h (heaps h' such that $h \subseteq h'$). Note also that $(s, h) \models (x \mapsto a, b)$ does not imply $(s, h') \models (x \mapsto a, b)$ for any heap h' that strictly extends h since the clause for $(s, h) \models (E \mapsto E_1, E_2)$ requires that the heap h should contain exactly one cell.
Given two formulas ψ and ϕ, entailment in SL is written $\psi \models \phi$ and defined as follows:

$$\psi \text{ entails } \phi \text{ iff } (s, h) \models \psi \text{ implies } (s, h) \models \psi \text{ for all stacks } s \text{ and heaps } h.$$

Validity in SL for a formula ϕ, written $\models \phi$, is then derived from entailment:

$$\phi \text{ is valid iff } (s, h) \models \phi \text{ for all stacks } s \text{ and heaps } h.$$

In SL models, the worlds are heaps (collections of cons cells in storage) and the separating conjunction $\phi * \psi$ is true just when the current heap can be split into two subheaps, one making ϕ true, the other one making ψ true. The separating implication $\phi \mathrel{-\!\!*} \psi$ generates new heaps such that whenever we get a fresh heap h_1 that makes ϕ true, combining it with the current heap h results in a new heap $h \cdot h_1$ that makes ψ true. All other connectives are interpreted pointwise.
For instance, the formula $(x \mapsto 3, 5) * ((x \mapsto 7, 5) \mathrel{-\!\!*} P)$ says that, in the current heap, x denotes a location that points to a cell containing the pair $(3, 5)$ and that if we update 3 to 7 then P will be true. Indeed, the semantics of $*$ splits a heap into two parts, one where $(x \mapsto 3, 5)$ holds and the other one where the location x is dangling. Then the semantics of $\mathrel{-\!\!*}$ and $\mapsto$ ensure that P must be true when the second heap is extended by binding x's location to the cell $(7, 5)$. The update therefore consists in first deleting the cell $(3, 5)$ from the current heap in order to bind the location denoted by x with the new cell $(7, 5)$.

2.2 Separation Logic and Proofs

Main concepts and results about the use of SL as an assertion language are given in [18,25]. As an interesting result, we can mention an operation that disposes of memory by creating dangling pointers through the command *dispose*(E), which deallocates a location by removing it from the heap.. From a semantic point of view, the dispose operation is defined by the following axiom (where a, b are not free in E):

$$\begin{array}{c}\{P * \exists\ a\ b.(E \mapsto a, b)\}\\ dispose(E)\\ \{P\}.\end{array}$$

Reasoning backwards from $\top$ we can find cases under which a program is safe to execute. With a double dispose we obtain $\bot$ for the precondition as expected, indicating that the program is not safe to execute for any start state:

$$\{\bot\}$$
$$\{\top * \exists\ a\ b.(x \mapsto a, b) * \exists\ c\ d.(x \mapsto c, d)\}$$
$$dispose(x)$$
$$\{\top * \exists\ a\ b.(x \mapsto a, b)\}$$
$$dispose(x)$$
$$\{\top\}$$

Separation Logic is a logic for specifying properties of dynamically-allocated linked data structures in a much simpler way than previous formalisms. Various works illustrate this point by showing that local reasoning allows a specification to focus only on the resources that are relevant to its soundness without bothering about the global context [3,5,13,25]. Therefore, an important challenge consists in providing methods and tools for verifying or proving such interesting specifications. Given that existing works on checking assertion written in Separation Logic automatically are mainly based on a model-checking approach [2,26], one of the main purpose of this paper is to study an alternative approach based on theorem-proving in order to provide methods and tools for proof-search and countermodel generation in Separation Logic. As a main result, we design a labelled tableau method which allows countermodel construction in case of non-provability. Further work will be devoted to comparisons between the model-checking approach and the theorem-proving approach and will lead to a deeper study of how both approaches can be combined in order to improve automated deduction.

Our starting point to develop proof-theoretic foundations for SL is our previous works on characterizing provability in the intuitionistic variant of BI through so-called resource graphs [17,16]. Two main obstacles arise:

- the first one is that SL is not really an extension of BI but is in fact an extension of Boolean BI (BBI) in which the additives are classical and thus it is not possible to extend our initial work to SL directly;
- the second one is that the $\mapsto$ predicate forces us to design resource graphs that take into account the specific properties of SL models.

Nevertheless, the general methodology that consists in reflecting the main properties of the models using labels, label constraints and resource graphs can be successfully adapted to Separation Logic. This can be seen as an important contribution as we expect to develop new theorem-proving methods for other spatial logics such as the Ambient Logic (a logic for mobile processes [12] which is also an extension of BBI), the labelled tree model [11] or even Context Logic [8] for which only model-checking techniques exist.

We begin our investigation of Separation Logic from a proof-search point of view with a fragment denoted SLP where there are no first order quantifiers, no variables and no equality predicates. In other words, SLP is the propositional fragment of SL where all atomic formulas are points-to predicates of the form $(l \mapsto a, b)$, where l, a and b are constants. We can then forget all stack variables in the semantic clauses of Definition 2.2. We shall conclude the paper with a discussion of how to extend the results obtained for SLP to more expressive fragments, including full SL.

3 Heaps, Labels and Resource Graphs

Purely syntactic proof methods (in sequent or natural deduction style) usually deal with a great amount of operational overhead (structural rules, permutabilities of inferences) which is mainly irrelevant w.r.t. the provability of a formula. On the other hand, purely semantic methods often abstract away too much of the operational aspects to be significantly helpful for countermodel construction. In the case of SL, we have a complete semantics based on partial monoids of heaps so that syntactic and semantic entailments (provability and validity) coincide [2]. Therefore, the main properties of SL models as well as entailment in SL can be reflected at a syntactic level using labels, label constraints and a specific closure operator [16] in order to define a resource driven proof method for SL.

3.1 Labels and Constraints

Our labelling language consists of a countable set $\mathcal{C}$ of constant symbols $c_1, c_2, \ldots, c_i, \ldots$

Definition 3.1 (label). *Let $\mathcal{I} = \{ i_1, \ldots, i_n \} (\subseteq \mathbb{N}^\star)$ be a finite set of indexes, a* label *is a (possibly empty) finite string $c_{i_1} \ldots c_{i_n} \in \mathcal{C}^*$ which satisfies the following condition :*

$$\textit{for all indexes } i_p, i_q \in \mathcal{I}, \textit{ if } p < q \textit{ then } i_p < i_q. \; \textit{(LL)}$$

A label x is a sublabel *of a label y, written $x \subseteq y$, iff all the constant symbols occurring in x also occur in y, more formally, iff $(\forall c_k \in \mathcal{C})(c_k \in x \Rightarrow c_k \in y)$.*
A label x is a strict sublabel *of y, written $x \subsetneq y$, iff $x \subseteq y$ and $x \neq y$.*

For example, the empty string ϵ is a label, the strings c_1, c_2, c_1c_2 and $c_2c_4c_5$ are labels The strings c_2c_1 and c_2c_2 are not labels because they do not satisfy the previous *(LL)* condition (label linearity) which requires that the indexes of the constant symbols should be sorted in a strict ascending order. Let us remark that *(LL)* implies that the same constant symbol cannot occur more than once in a label.
Following Definition 3.1, $\epsilon, c_2, c_4, c_5, c_2c_4, c_2c_5, c_4c_5$ and $c_2c_4c_5$ are all sublabels of $c_2c_4c_5$, $c_2c_4c_5$ being the only one of them which is not also strict. Let us remark that being a sublabel and being a substring are two distinct notions as illustrated in the previous example where c_2c_5 is a sublabel but not a substring of $c_2c_4c_5$.

Definition 3.2. *Let $\mathcal{I} = \{ i_1, \ldots, i_n \} (\subseteq \mathbb{N}^\star)$ be a finite set of indexes and x be the label $c_{i_1} \ldots c_{i_n}$, we define the* length *of the label x, written $|x|$, as the number of constant symbols occurring in x. Moreover, for all $k \in \mathbb{N}$ such that $1 \leq k \leq |x|$, x_k denotes the constant symbol which is at index k in x,* i.e., *$x_k = c_{i_k}$.*

Intuitively, constant symbols are intended to be a syntactic reflection of heaps. We now introduce the notion of label composition, which is intended to be a syntactic reflection of heap composition.

[2] This is not the case for Boolean BI, for which there is currently no known complete monoid-based semantics.

Definition 3.3 (label composition). *Let $\mathcal{L}$ be the smallest subset of $\mathcal{C}^*$ containing all the labels, the composition $x \circ y$ of two labels x and y is defined as the shortest label z containing the string xy as a sublabel.*

Since the composition $x \circ y$ of two labels x and y is required to be a label, the linearity condition *(LL)* of Definition 3.1 entails that $x \circ y$ is defined iff x and y do not share any constant symbol, thus making $\circ : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ a partial function.
It is routine to check that $\circ$ is associative, commutative and admits the empty string ϵ as its unit. Therefore, the structure $\mathcal{L}_A = (\mathcal{L}, \circ, \epsilon)$, called the *labelling algebra*, is in fact a partial commutative monoid of labels. For example, the following equalities hold:

$$c_1 \circ (((c_2 \circ \epsilon) \circ \epsilon) \circ (c_3 \circ c_4)) = (c_2 \circ c_1) \circ (\epsilon \circ (c_4 \circ c_3)) = c_1c_2c_3c_4.$$

Given that label composition $\circ$ is associative and commutative, we almost always omit parentheses. More often than not, we simplify our notation even further by omitting the composition symbol $\circ$ as well. This slight abuse of notation allows us to write labels as unordered sequences of symbols so that we can for example speak of "the label $c_2c_1\epsilon$" keeping in mind that what we actually mean is "the label c_1c_2 resulting from the evaluation of the composition $c_2 \circ c_1 \circ \epsilon$".
Let us now introduce some useful operations on labels.

Definition 3.4. *Let x and y be labels,*

- *$x \cap y$ is the longest label that is a sublabel of both x and y;*
- *$x \backslash y$ is the longest sublabel of x that does not share any constant symbol with y.*

In other words, $x \cap y$ is the label which contains all the constant symbols that are shared by x and y while $x \backslash y$ is the label obtained by discarding from x all the constant symbols that also occur in y. For example, we have

- $c_1c_2c_3 \backslash c_1c_3 = c_1c_2c_3 \backslash c_1c_3c_4 = c_2$ and $c_1c_2c_3 \backslash c_1c_2c_3 = c_1c_2c_3 \backslash c_1c_2c_3c_4 = \epsilon$;
- $c_1c_2c_3 \cap c_1c_3 = c_1c_2c_3 \cap c_1c_3c_4 = c_1c_3$, $c_1c_2c_3 \cap c_1c_2c_3 = c_1c_2c_3$ and $c_1c_2c_3 \cap c_4 = \epsilon$.

It is routine to check that $x \cap y = \epsilon$ iff x and y are disjoint and that $(x \backslash y) \circ (x \cap y) = x$ for all labels x and y.

Definition 3.5 (label constraint). *A* label constraint *is either an expression of the form* $\mathrm{T}x$*, where x is a label and* T *is a letter from the alphabet $\mathbb{T} = \{\mathrm{Z}, \mathrm{U}\}$, or a binary expression of the form $x \diamond y$ where x and y are labels.*
We write $\mathcal{K}$ to denote the set of all label constraints generated by $\mathcal{L}$.

Intuitively, the purpose of a label constraint $x \diamond y$ is to capture the fact that the labels x and y denote the same heap. Therefore, the symbol $\diamond$ is meant to be a syntactic reflection of equality between heaps. Accordingly, a label constraint of the form $(y \circ z) \diamond x$ means that if x, y, z are interpreted as heaps, then the heap denoted by x is the same as (or equal to) the one obtained by combining the heaps denoted by y and z.
On the other hand, label constraint of the form $\mathrm{Z}x$ (respectively $\mathrm{U}x$) means that x should

be interpreted as a heap containing no cell (respectively exactly one cell). For simplicity, we shall often write “constraint” instead of “label constraint”.
In order to capture the various resource interactions that occur in SL models we define an appropriate closure operator $(\cdot)^\dagger$ on sets of labels and label constraints.

Definition 3.6 **($(\cdot)^\dagger$-closure).** *Let X $(\subseteq \mathcal{L} \cup \mathcal{K})$ be a set of labels and label constraints, $X^\dagger$ is defined as the couple $(X_\mathcal{L}, X_\mathcal{K})$, where the sets $X_\mathcal{L}$ and $X_\mathcal{K}$ are defined by mutual induction according to the following rules:*

- *Base case:*
  - $X_\mathcal{L} = (X \cap \mathcal{L}) \cup \{ \epsilon \}$;
  - $X_\mathcal{K} = (X \cap \mathcal{K}) \cup \{ \mathrm{Z}\epsilon \}$.
- *Mutual induction:*
  - $x \in X_\mathcal{L}$ *and* $y \subsetneq x \Rightarrow y \in X_\mathcal{L}$ *(Saturation);*
  - $x \in X_\mathcal{L} \Rightarrow x \diamond x \in X_\mathcal{K}$ *($\diamond$-Reflexivity);*
  - $x \diamond y \in X_\mathcal{K} \Rightarrow x, y \in X_\mathcal{L}$ *($\diamond$-Completion);*
  - $x \diamond y \in X_\mathcal{K} \Rightarrow y \diamond x \in X_\mathcal{K}$ *($\diamond$-Symmetry);*
  - $x \diamond y \in X_\mathcal{K}$ *and* $y \diamond z \in X_\mathcal{K} \Rightarrow x \diamond z \in X_\mathcal{K}$ *($\diamond$-Transitivity);*
  - $x \diamond y \in X_\mathcal{K}$ *and* $y \circ z \in X_\mathcal{L} \Rightarrow (x \circ z) \diamond (y \circ z) \in X_\mathcal{K}$ *($\diamond$-Propagation);*
  - $\mathrm{T}x \in X_\mathcal{K} \Rightarrow x \in X_\mathcal{L}$ *(T-Completion, $\mathrm{T} \in \mathbb{T}$);*
  - $\mathrm{Z}x \in X_\mathcal{K}$ *and* $y \subsetneq x \Rightarrow \mathrm{Z}y \in X_\mathcal{K}$ *(Z-Decomposition);*
  - $\mathrm{U}x \in X_\mathcal{K}$ *and* $\mathrm{U}y \in X_\mathcal{K}$ *and* $y \subsetneq x \Rightarrow \mathrm{Z}(x \backslash y) \in X_\mathcal{K}$ *(U-Decomposition);*
  - $\mathrm{T}x \in X_\mathcal{K}$ *and* $x \diamond y \in X_\mathcal{K} \Rightarrow \mathrm{T}y \in X_\mathcal{K}$. *($\mathrm{T}$-Propagation, $\mathrm{T} \in \mathbb{T}$).*

The set $X_\mathcal{L}$ is called the domain of X *and the set $X_\mathcal{K}$ is called the* structure of X.
The set $X_\mathcal{K}^\diamond$ (respectively the set $X_\mathcal{K}^\mathrm{T}$, $\mathrm{T} \in \mathbb{T}$) is the restriction of $X_\mathcal{K}$ to the label constraints of the form $x \diamond y$ (respectively of the form $\mathrm{T}x$, $\mathrm{T} \in \mathbb{T}$).

The *(Saturation)* property ensures that $X_\mathcal{L}$ is closed under sublabels while *($\diamond$-Completion)* ensures that any label occurring in a label constraint of $X_\mathcal{K}$ also explicitly occurs as a label of $X_\mathcal{L}$. *($\diamond$-Reflexivity)*, *($\diamond$-Transitivity)*, *($\diamond$-Symmetry)* and *($\diamond$-Propagation)* entail that $\diamond$ is a congruence on $(X_\mathcal{L}, \circ)$ just as equality between heaps is a congruence on $(Heaps, \cdot)$.
(Z-Decomposition) reflects the fact that all subheaps of an empty heap are also empty. Now let h_x and h_y be the heaps denoted by x and y respectively, then, *(U-Decomposition)* reflects the fact that if h_y is a subheap of h_x and both h_x and h_y have exactly one location then all subheaps of h_x that are disjoint from h_y must be empty.
Finally, *(T-Propagation)* describes how the properties of being an empty heap, a heap with exactly one location, a heap with at least one location, or a heap with not only one location are propagated through heap composition. For example, if $\mathrm{T} = \mathrm{Z}$, $x = c_1$ and $y = c_2c_3$, then, *(T-Propagation)* means that if c_1 denotes an empty heap which may be obtained by composition of the two heaps denoted by c_2 and c_3 then c_2c_3 also denotes an empty heap, *i.e.*, if $\mathrm{Z}c_1 \in X_\mathcal{K}$ and $c_2c_3 \diamond c_1 \in X_\mathcal{K}$ then $\mathrm{Z}c_2c_3 \in X_\mathcal{K}$.

3.2 Resource Graphs

In this subsection, we explain how the $(\cdot)^{\dagger}$-closure of a set of labels and label constraints gives rise to a structure called a *resource graph*. This concept has already been introduced for BI [16] but needs specific adaptations for SL.

Definition 3.7. *Let $X\ (\subseteq \mathcal{L} \cup \mathcal{K})$ be a set of label and label constraints.*

1. *We define η as the function $X_{\mathcal{L}} \to \wp(\mathbb{T}) \times X_{\mathcal{L}}$ which maps each label x in the domain $X_{\mathcal{L}}$ of X to the couple (Γ, x) where $\Gamma\ (\subseteq \mathbb{T})$ is defined as follows:*
$$\forall\, \mathrm{T} \in \mathbb{T}.\ \mathrm{T} \in \Gamma \textit{ iff } \mathrm{T}x \in X_{\mathcal{K}}.$$
We write Γx to denote the couple (Γ, x) and we call Γ the tag of the label x.
2. *We define ν as the function $X_{\mathcal{K}}^{\diamond} \to \wp(\wp(\mathbb{T}) \times X_{\mathcal{L}})$ which maps each label constraint $x \diamond y$ in the structure $X_{\mathcal{K}}$ of X to the (unordered) pair $\{\, \eta(x), \eta(y) \,\}$.*

In the rest of the paper we shall use the letters Γ, Δ, Θ to range over tags. Using the functions η and ν we can proceed with the definition of a resource graph.

Definition 3.8 (resource graph). *The* resource graph $\mathcal{G}(X)[N, E]$ *associated to a set $X\ (\subseteq \mathcal{L} \cup \mathcal{K})$ of labels and label constraints is the undirected graph such that:*

- *the set of nodes N is the set $\eta(X_{\mathcal{L}}) = \{\, \eta(x) \mid x \in X_{\mathcal{L}} \,\}$;*
- *the set of edges E is the set $\nu(X_{\mathcal{K}}^{\diamond}) = \{\, \nu(k) \mid k \in X_{\mathcal{K}}^{\diamond} \,\}$.*

Let us remark that η is a one-to-one correspondence between nodes and labels. Strictly speaking, ν is not a one-to-one correspondence between edges and $\diamond$-constraints because a resource graph is an undirected graph ($\nu(x \diamond y) = \nu(y \diamond x) = \{\, \eta(x), \eta(y) \,\}$). However, ν can be thought of as a one-to-one correspondence between edges $\{\, \eta(x), \eta(y) \,\}$ and pairs $\{\, x \diamond y, y \diamond x \,\}$ of symmetric $\diamond$-constraints.
For a first example of resource graph, let us consider the set
$$X = \{\, \mathrm{Z}\epsilon, c_2c_3 \diamond c_1, \mathrm{U}c_2, \mathrm{U}c_2c_3 \,\}.$$

In order to obtain the resource graph $\mathcal{G}(X)$ associated to X we need to compute the closure $X^{\dagger}$ according to Definition 3.6. Therefore, we first add the label constraint $\mathrm{Z}c_3$ to satisfy *(*U*-Decomposition)* since we have $\mathrm{U}c_2$, $\mathrm{U}c_2c_3$ and $c_3 \subsetneq c_2c_3$. Then, because of $c_2c_3 \diamond c_1$ and $\mathrm{U}c_2c_3$, we add $\mathrm{U}c_1$ to satisfy *(*U*-Propagation)*. Applying *(*T*-Completion)* finally leads to the closure
$$X^{\dagger} = (X_{\mathcal{L}}, X_{\mathcal{K}}) = (\{\, \epsilon, c_1, c_2, c_3, c_2c_3 \,\}, \{\, \mathrm{Z}\epsilon, c_2c_3 \diamond c_1, \mathrm{U}c_2, \mathrm{U}c_2c_3, \mathrm{Z}c_3, \mathrm{U}c_1 \,\}).$$

Therefore, the resource graph $\mathcal{G}(X)$ looks as follows [3]:

$$\begin{tikzcd}[column sep=small] \mathcal{G}(X): & \boxed{\mathrm{Z}\epsilon} & \boxed{\mathrm{Z}c_3} & \boxed{\mathrm{U}c_1} \arrow[d, dash] & \boxed{\mathrm{U}c_2} \\ & & & \boxed{\mathrm{U}c_2c_3} & \end{tikzcd}$$

[3] For simplicity, we do not explicitly represent reflexive and transitive edges.

For a second example, let us consider the set

$$Y = \{\, \mathrm{Z}\epsilon, c_2c_3 \diamond c_1, \mathrm{U}c_2, \mathrm{U}c_3, \mathrm{Z}c_1 \,\}.$$

Successively applying *(*Z*-Propagation)* and *(*Z*-Decomposition)* to the label constraints $c_2c_3 \diamond c_1$ and $\mathrm{Z}c_1$, followed by an application of *(*T*-Completion)* leads to

$$Y^\dagger = (Y_\mathcal{L}, Y_\mathcal{K}) = (\{\, \epsilon, c_1, c_2, c_3, c_2c_3 \,\}, \{\, \mathrm{Z}\epsilon, c_2c_3 \diamond c_1, \mathrm{U}c_2, \mathrm{U}c_3, \mathrm{Z}c_1, \mathrm{Z}c_2c_3, \mathrm{Z}c_2, \mathrm{Z}c_3 \,\}).$$

Therefore, the resource graph $\mathcal{G}(Y)$ looks as follows [4]:

$$\mathcal{G}(Y): \quad \begin{tikzcd} \boxed{\mathrm{Z}\epsilon} & \boxed{\mathrm{ZU}c_3} & \boxed{\mathrm{Z}c_1} \arrow[d, dash] & \boxed{\mathrm{ZU}c_2} \\ & & \boxed{\mathrm{Z}c_2c_3} & \end{tikzcd}$$

3.3 Labels and Constraints vs Resource Graphs

The one-to-one correspondence induced by the functions η and ν given in Definition 3.7 allows us to translate any concept defined in terms of $(\cdot)^\dagger$-closure directly in terms of resource graphs and vice versa.
In particular, given a set $X\ (\subseteq \mathcal{L} \cup \mathcal{K})$ of label and constraints, any function $f : X_\mathcal{L} \to F$ defined on the domain $X_\mathcal{L}$ of X induces a function $g : N \to F$ defined on the nodes of $\mathcal{G}(X)[N, E]$ by setting:

$$\forall\, \Gamma x \in N.\ g(\Gamma x) = f(x).$$

Conversely, any function $g : N \to F$ defined on the nodes of $\mathcal{G}(X)[N, E]$ gives rise to a function $f : X_\mathcal{L} \to F$ on the domain $X_\mathcal{L}$ of X by setting:

$$\forall\, x \in X_\mathcal{L}.\ f(x) = g(\eta(x)).$$

More generally, any n-ary relation $R_\mathcal{L}\ (\subseteq X_\mathcal{L}^n)$ between the labels in the domain $X_\mathcal{L}$ of X induces a n-ary relation $R_N\ (\subseteq N^n)$ between the nodes N of $\mathcal{G}(X)[N, E]$ by setting:

$$\forall\, \Gamma_1 x_1, \ldots, \Gamma_\mathrm{n} x_n \in N.\ R_N(\Gamma_1 x_1, \ldots, \Gamma_\mathrm{n} x_n) \text{ iff } R_\mathcal{L}(x_1, \ldots, x_n).$$

The other way round, any n-ary relation $R_N\ (\subseteq N^n)$ between the nodes N of $\mathcal{G}(X)[N, E]$ induces a n-ary relation $R_\mathcal{L}\ (\subseteq X_\mathcal{L}^n)$ between the labels in the domain $X_\mathcal{L}$ of X by setting:

$$\forall\, x_1, \ldots, x_n \in X_\mathcal{L}.\ R_\mathcal{L}(x_1, \ldots, x_n) \text{ iff } R_N(\eta(x_1), \ldots, \eta(x_n)).$$

Definition 3.9. *Let $X\ (\subseteq \mathcal{L} \cup \mathcal{K})$ be a set of label and label constraints.*

1. *Given a label constraint k, we say that k holds in X, written $X \vdash k$, iff $k \in X_\mathcal{K}$.*
2. *Given a n-ary relation $R_\mathcal{L}\ (\subseteq \mathcal{L}^n)$ between labels, we write*

$$X \vdash R_\mathcal{L}(x_1, \ldots, x_n) \textit{ iff } R_\mathcal{L}(x_1, \ldots, x_n) \textit{ and } x_1, \ldots, x_n \in X_\mathcal{L}.$$

[4] We shall see later in the paper that the graph $\mathcal{G}(Y)$ cannot be realized by any heap structure since it contains nodes, namely c_2 and c_3, that have both the letter Z and the letter U in their tag.

3. *For all letters* $\mathrm{T} \in \mathbb{T}$*, a label* x *such that* $X \vdash \mathrm{T}x$ *is called a* T-label*, or a* T-constant *whenever* x *is a constant symbol.*

The previous definitions are lifted to the resource graph $\mathcal{G}(X)$ *associated to* X *as follows:*

1. *Given a label constraint* k*,* $\mathcal{G}(X) \vdash k$ *iff* $X \vdash k$*.*
2. *Given a n-ary relation* R_N $(\subseteq N^n)$*,*

$$\mathcal{G}(X) \vdash R_N(\Gamma_1 x_1, \ldots, \Gamma_n x_n) \textit{ iff } X \vdash R_{\mathcal{L}}(x_1, \ldots, x_n).$$

3. *For all letters* $\mathrm{T} \in \mathbb{T}$*, a node* Γx *such that* $\mathrm{T} \in \Gamma$ *is called a* T-node*.*

4 A Tableau Calculus for SLP

In this section, we introduce a tableau-based calculus for SLP which uses labels and constraints to build resource graphs. The choice of a tableau method is motivated by its well-known ability to provide countermodel extraction facilities [14], but our notions of labels, constraints and resource graphs can also be integrated into connection-based, or sequent-based calculi, in order to characterize provability in SLP.

4.1 Expansion Rules

In general, tableau methods are methods in which one tries to prove a formula ϕ by showing that there cannot exist any model $\mathcal{M}$ such that $\mathcal{M} \not\models \phi$. In classical logic, this amounts to showing that the formula $\neg\phi$ has no model. Tableau methods are therefore based on a refutation principle as the tableau construction process tries to build a model of $\neg\phi$ and concludes the validity of ϕ if it fails to do so.

In order to find a model of $\neg\phi$, the tableau construction process relies on the application of expansion rules (depending on the syntactic structure of ϕ) that should ideally allow the enumeration of all possible models of $\neg\phi$.

Since we deal with Separation Logic, which is a non classical logic, our tableau method uses signs F or T instead of the negation symbol $\neg$ to indicate whether a formula is to be proved or disproved. Such signs help avoiding confusion between negation in the object language (here, the formulas of SL) which may have various specific properties (like for instance $\neg\neg\phi = \phi$ or $(\neg\phi * \neg\psi) = \neg(\phi * \psi)$) and the meta-language used to describe the proof-search process.

Definition 4.1 (signed formula). *A* signed formula *is an expression* $\mathrm{S}\,\phi : x$*, where* $\mathrm{S} \in \{\mathrm{F}, \mathrm{T}\}$ *is a sign,* ϕ *is a* SLP *formula and* x *is a label.*

Intuitively, a signed formula $\mathrm{S}\,\phi : x$ means that whenever the label x is interpreted as a heap h_x in some stack s we have $(s, h_x) \models \phi$ if $\mathrm{S} = \mathrm{T}$ and $(s, h_x) \not\models \phi$ if $\mathrm{S} = \mathrm{F}$.

Let us now recall that entailment in Separation Logic is defined as follows:

$$\psi \text{ entails } \phi \ (\psi \models \phi) \ \text{ iff } \ \text{for all stacks } s \text{ and heaps } h,\ (s,h) \models \psi \text{ implies } (s,h) \models \phi \quad \textit{(E)}.$$

Validity is then defined so that

$$\phi \text{ is valid } (\models \phi) \text{ iff for all stacks } s \text{ and heaps } h,\ (s,h) \models \phi \quad \textit{(V)},$$

which is in turn equivalent to the following statement that clearly shows the link between entailment and validity:

$$\phi \text{ is valid } (\models \phi) \text{ iff } \top \text{ entails } \phi\ (\top \models \phi) \quad \textit{(VE)}.$$

Since we do not deal with variables in the SLP fragment, we can forget the stacks and rewrite entailment and validity so that

$$\psi \text{ entails } \phi\ (\psi \models \phi) \text{ iff for all heaps } h,\ h \models \psi \text{ implies } h \models \phi \quad \textit{(E')};$$

$$\phi \text{ is valid } (\models \phi) \text{ iff for all heaps } h,\ h \models \phi \quad \textit{(V')}.$$

It is then clear from the definition of $-\!\!*$ that in SLP

$$\psi \models \phi \text{ iff } e \models \psi -\!\!* \phi \quad \textit{(EE)},$$

which, using the link *(VE)* between validity and entailment, implies that

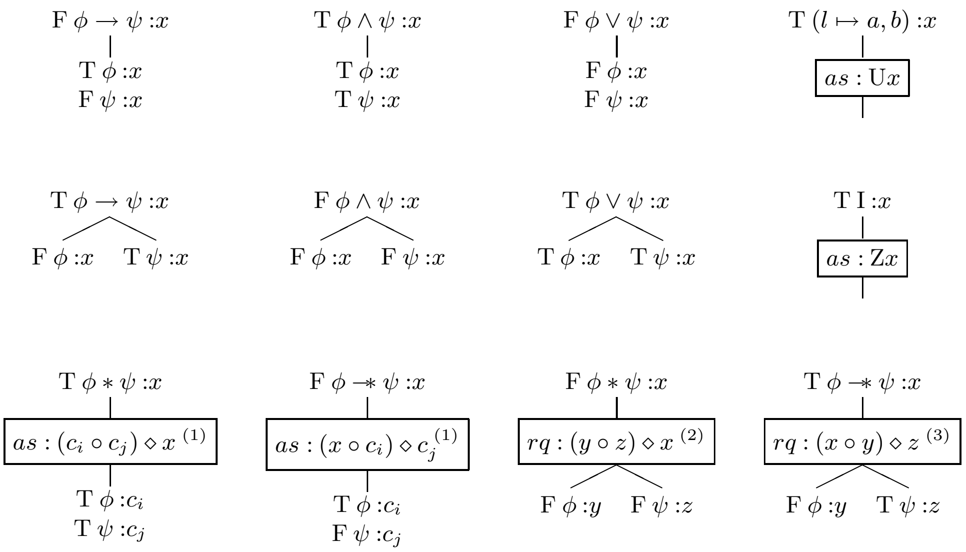
$$\models \phi \text{ iff } e \models \top -\!\!* \phi \quad \textit{(VEE)}.$$

The *(EE)* condition means that in order to prove an entailment $\psi \models \phi$, we only need to show that the empty heap e satisfies the formula $\psi -\!\!* \phi$. Let us remark that the previous statement is not true if $-\!\!*$ is replaced by $\rightarrow$ because $e \models \psi \rightarrow \phi$ does not imply $h \models \psi \rightarrow \phi$ for all heaps h and all formulas ψ and ϕ.
In terms of tableaux, where one tries to refute the existence of a model, what we need to show is that trying to disprove the formula $\psi -\!\!* \phi$ w.r.t. the syntactic reflection ϵ of the empty heap e eventually fails (leads to contradictions), which justifies the fact that a tableau proof should start with the signed formula $\mathrm{F}\ \psi -\!\!* \phi : \epsilon$ as formalized in the following definition.

Definition 4.2 (tableau). *Let ϕ be a formula in* SLP*.*

1. *A* tableau for $\psi \models \phi$ *is a binary tree $\mathcal{T}$ built according to the rules given in Figure 1 and such that:*
   - *all nodes are labelled either with a signed formula, or with a label constraint,*
   - *the root node is labelled with the signed formula $\mathrm{F}\ \psi -\!\!* \phi : \epsilon$.*
2. *A tableau* tableau for $\models \phi$ *(more shortly, a* tableau for ϕ*) is a tableau for $\top \models \phi$.*

We divide and present the tableau rules of Figure 1 into four groups.
The first group of rules consists of the *conservative* rules $\{\, \mathrm{S}\wedge, \mathrm{S}\vee, \mathrm{S}\rightarrow \mid \mathrm{S} \in \{\mathrm{F}, \mathrm{T}\}\,\}$ associated to the additive connectives. Conservative rules standing on the first line of Figure 1 are given the type α, while those standing on the second line are given the type β. This terminology is justified by the fact that if one forgets about the labels, the conservative rules simply amount to the standard α and β tableau rules for classical logic. Let us note that applying a rule of type α rule in a tableau branch results in a simple extension of that branch, while applying a rule of type β splits one tableau branch in



$^{(1)}$ c_i, c_j are new constants and $i < j$ $\quad$ $^{(2)}$ $\mathcal{B} \vdash (y \circ z) \diamond x$ $\quad$ $^{(3)}$ $\mathcal{B} \vdash (x \circ y) \diamond z$

Figure1. TSLP Expansion Rules.

two. Moreover, conservative rules expand tableau branches without modifying the labels (propagation from formulas to subformulas). For example the expansion of a signed formula $\mathrm{F}\ \phi \wedge \psi : x$ in a tableau branch $\mathcal{B}$ splits it in two branches $\mathcal{B}_1$ and $\mathcal{B}_2$, the first one containing $\mathrm{F}\ \phi : x$ and the second one containing $\mathrm{F}\ \psi : x$, which semantically means that one should either falsify ϕ, or falsify ψ in order to falsify $\phi \wedge \psi$.
The second group of rules consists of the *assertive* rules $\mathrm{T}*$ and $\mathrm{F}{-\!\!*}$, which are given the type $\pi\alpha$. Assertive rules introduce two new constants and a new label constraint, which is called an *assertion* because it is established as a fact. For example, the expansion of a signed formula $\mathrm{T}\ \phi * \psi : x$ leads to the introduction of the assertion $as : (c_i \circ c_j) \diamond x$ which means that whenever the labels x, c_i and c_j are respectively interpreted as heaps h_x, h_{c_i} and h_{c_j}, it must be the case that $h_x = h_{c_i} \cdot h_{c_j}$, *i.e.*, it must be the case that the heap denoted by x can be obtained by the composition of the two heaps denoted by c_i and c_j. Let us note that this necessarily implies that the heaps h_{c_i} and h_{c_j} denoted by c_i and c_j should be disjoint. Moreover, the proviso requiring $i < j$ ensures that $c_i \circ c_j = c_i c_j$.

Definition 4.3 (resource graph $\mathcal{G}(\mathcal{B})$). *Given a tableau branch $\mathcal{B}$,*

- *the set of all the assertions occurring in $\mathcal{B}$ is denoted $\mathcal{B}^{\mathcal{A}}$;*
- *the notations $\mathcal{B}_{\mathcal{L}}$ and $\mathcal{B}_{\mathcal{K}}$ are shorthands for the sets $\mathcal{B}^{\mathcal{A}}_{\mathcal{L}}$ and $\mathcal{B}^{\mathcal{A}}_{\mathcal{K}}$ respectively;*
- *for all label constraints k, k holds in $\mathcal{B}$ (written $\mathcal{B} \vdash k$) iff $k \in \mathcal{B}_{\mathcal{K}}$;*
- *for all n-ary relation $R\,(\subseteq \mathcal{L}^n)$ between labels,*

$$\mathcal{B} \vdash R(x_1, \ldots, x_n) \textit{ iff } R(x_1, \ldots, x_n) \textit{ and } x_1, \ldots, x_n \in \mathcal{B}_{\mathcal{L}};$$

The resource graph associated to $\mathcal{B}$, *denoted* $\mathcal{G}(\mathcal{B})$, *is the resource graph* $\mathcal{G}(\mathcal{B}^{\mathcal{A}})$ *generated by the assertions occurring in* $\mathcal{B}$.
Using the correspondence discussed in Subsection 3.3, we obtain:

- *for all label constraints* k, $\mathcal{G}(\mathcal{B}) \vdash k$ *iff* $\mathcal{B} \vdash k$;
- *for all n-ary relation* $R\,(\subseteq \mathcal{L}^n)$ *between labels,*

$$\mathcal{G}(\mathcal{B}) \vdash R(\eta(x_1), \ldots, \eta(x_n)) \text{ iff } \mathcal{B} \vdash R(x_1, \ldots, x_n).$$

The assertions occurring in a tableau branch $\mathcal{B}$ are used to distinguish a particular class of SLP models and such a class is captured by the resource graph $\mathcal{G}(\mathcal{B})$, which is a graphical representation of the closure $(\mathcal{B}_{\mathcal{L}}, \mathcal{B}_{\mathcal{K}})$. Therefore, a key point of our tableau system is that it builds a resource graph for each tableau branch in parallel of the construction of the tableau itself.
The third group of rules consists of the *generative* rules $\mathrm{F}{*}$ and $\mathrm{T}{-\!\!*}$, which are given the type $\pi\beta$. Generative rules do not introduce new labels in a tableau branch $\mathcal{B}$ but rather reuse the ones that already exist in the resource graph $\mathcal{G}(\mathcal{B})$ (in $\mathcal{B}_{\mathcal{L}}$) by the time the expansion needs to be performed. Moreover, the label constraint introduced by a generative rule is called a *requirement* because it is intended to behave as a goal that should be achieved from the set of assertions (which behave as facts). For example, the expansion of a signed formula $\mathrm{F}\,\phi * \psi : x$ in a tableau branch $\mathcal{B}$ requires us to find two labels $y, z \in \mathcal{B}_{\mathcal{L}}$ such that $(y \circ z) \diamond x \in \mathcal{B}_{\mathcal{K}}$, *i.e.*, such that $\mathcal{B} \vdash (y \circ z) \diamond x$. Unlike all other rules, generative rules can be expanded several times since there might be several labels $y, z \in \mathcal{B}_{\mathcal{L}}$ such that $\mathcal{B} \vdash (y \circ z) \diamond x$ and each suitable choice for y and z might be tried. From a semantic point of view, the proviso $\mathcal{B} \vdash (y \circ z) \diamond x$ means that it should always be the case that whenever a SLP model belongs to the class of models captured by the resource graph $\mathcal{G}(\mathcal{B})$ (by the closure $(\mathcal{B}_{\mathcal{L}}, \mathcal{B}_{\mathcal{K}})$), this model should be such that the heap h_x denoted by x can be obtained by the composition of the two heaps h_y and h_z denoted by y and z, *i.e.*, $h_x = h_y \cdot h_z$.
The fourth and last group of rules consists of the *special* rules $\{\mathrm{TI}, \mathrm{T}{\mapsto}\}$, which deal with the multiplicative unit I and points-to predicates by introducing a new assertion without introducing new labels. For example, the assertion $as : \mathrm{U}x$ introduced by the expansion of a signed formula $\mathrm{T}\,(l \mapsto a, b) : x$ reflects the fact that if the heap denoted by x is to satisfy $(l \mapsto a, b)$ then it should contain exactly one cell. Similarly, the assertion $as : \mathrm{Z}x$ introduced by the expansion of the signed formula $\mathrm{T}\,\mathrm{I} : x$ reflects the fact that x should denote the empty heap if it is to satisfy I.
Let us remark that we do not give any expansion rules for $\mathrm{F}\,\mathrm{I} : x$ and $\mathrm{F}\,(l \mapsto a, b) : x$ (although such rules could be defined) since such signed formulas shall be dealt with in the definition of logical consistency (Definition 4.15) as we shall later.
The $\mathrm{F}{-\!\!*}$ rule can be safely reformulated so that it generates only one new constant c_i instead of two by setting $c_j = x \circ c_i$. Doing so immediately turns the initial assertion $as : (x \circ c_i) \diamond c_j$ into $as : (x \circ c_i) \diamond (x \circ c_i)$ which only imposes as a fact that x and c_i should denote disjoint heaps. Similarly for the $\mathrm{T}{-\!\!*}$ rule, setting $z = x \circ y$ turns the initial requirement $rq : (x \circ y) \diamond z$ into $rq : (x \circ y) \diamond (x \circ y)$ which trivially holds in a tableau branch $\mathcal{B}$ by the *($\diamond$-Reflexivity)* condition of Definition 3.6 as soon as the label $x \circ y$ occurs in $\mathcal{B}_{\mathcal{L}}$.

More formally, if $x \circ y \in \mathcal{B}_{\mathcal{L}}$ then $\mathcal{B} \vdash (x \circ y) \diamond (x \circ y)$. The previous arguments then lead to the alternative versions of the tableau rules for $-\!\!*$ depicted in Figure 2.

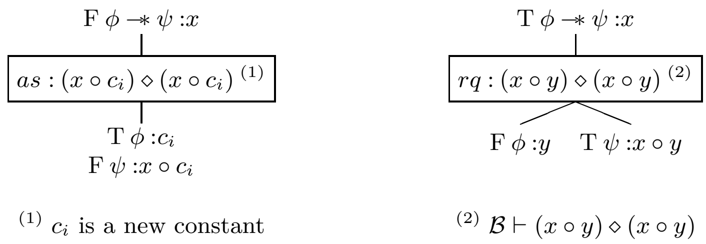


Figure2. Alternative Rules F$'-\!\!*$ and T$'-\!\!*$.

In order to keep the notation of our tableaux reasonably compact, we almost always

- use the alternate versions F$'-\!\!*$ and T$'-\!\!*$ of the tableau rules for $-\!\!*$;
- omit the trivial assertion $as : (x \circ c_i) \diamond (x \circ c_i)$ introduced by F$'-\!\!*$;
- omit the trivial requirement $rq : (x \circ y) \diamond (x \circ y)$ introduced by of T$'-\!\!*$;
- omit the assertions introduced by the special rules since we keep track of them in the resource graphs associated to the tableau branches.

Figure 3 presents a step by step illustration of how the expansion rules build a tableau together with a resource graph for each branch.
At the beginning, the tableau has only one branch containing the signed formula

$$\mathrm{F}\ ((l \mapsto a, b) * ((l \mapsto a, b) -\!\!* (k \mapsto c, d))) -\!\!* (m \mapsto e, f) : \epsilon$$

so that the associated resource graph is the resource graph named $\mathcal{G}_0$ in Figure 3.
In Step 1 (marked with a check sign $\surd_1$), the initial signed formula is expanded using the rule F$'-\!\!*$, which requires the introduction of a new constant c_1 and a new label $\epsilon \circ c_1$, which is in fact equal to the label c_1. Since F$'-\!\!*$ is a $\pi\alpha$ rule, it merely extends the only existing branch the resource graph of which becomes $\mathcal{G}_1$.
In Step 2 we apply the rule T$*$ on the signed formula

$$\mathrm{T}\ (l \mapsto a, b) * ((l \mapsto a, b) -\!\!* (k \mapsto c, d)) : c_1$$

which introduces two new constants c_2 and c_3 and a new assertion $as_1 : c_2c_3 \diamond c_1$ that imposes as a fact that c_1 can be decomposed into two parts c_2 and c_3.
Let us also note that Step 2 results in the introduction of the two signed formulas

$$\mathrm{T}\ (l \mapsto a, b) : c_2 \text{ and } \mathrm{T}\ (l \mapsto a, b) -\!\!* (k \mapsto c, d) : c_3$$

which respectively lead to Step 3 and Step 4.
In Step 3, we apply the rule T$\mapsto$ on the signed formula T $(l \mapsto a, b) : c_2$, which inserts the new assertion $as_2 : \mathrm{U}c_2$ between the two signed formulas introduced in Step 2. Such an assertion imposes as a fact that the label c_2 should denote a heap with exactly one cell.
In Step 4, applying the rule T$'-\!\!*$ on the signed formula

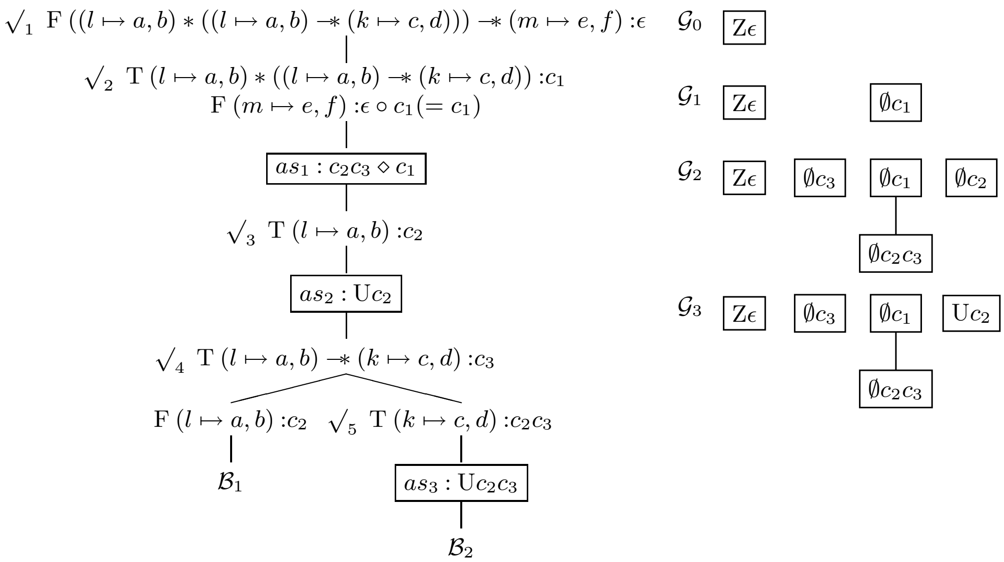


Figure3. Tableau $\mathcal{T}$ for $(l \mapsto a,b) * ((l \mapsto a,b) \mathrel{-\!\!*} (k \mapsto c,d)) \models (m \mapsto e,f)$.

$$\mathrm{T}\ (l \mapsto a,b) \mathrel{-\!\!*} (k \mapsto c,d) : c_3$$

requires us to find a label y such that $(y \circ c_3) \diamond (y \circ c_3)$ holds in the current resource graph, which is the resource graph $\mathcal{G}_3$ generated during the previous expansion step. Choosing c_2 for y is admissible because the labels c_2, c_3, c_2c_3 occur in $\mathcal{G}_3$ so that $(c_2 \circ c_3) \diamond (c_2 \circ c_3)$ trivially holds in $\mathcal{G}_3$. However, several other choices may be admissible, for example, one can also choose ϵ for y since $c_2 \circ \epsilon = c_2$ and $c_2 \diamond c_2$ holds in $\mathcal{G}_3$. Anyway, all available choices may be considered one after the other as $\mathrm{T}'\mathrel{-\!\!*}$ is a generative rule and can therefore be expanded as many times as needed.
After Step 4 the tableau splits into two branches $\mathcal{B}_1$ and $\mathcal{B}_2$, the first one being extended with the signed formula $\mathrm{F}\ (l \mapsto a,b) : c_2$, the second one with $\mathrm{T}\ (k \mapsto c,d) : c_2c_3$.
Since a generative rule does not modify the resource graph of the tableau branch it splits, after Step 4 is completed we obtain

$$\mathcal{G}(\mathcal{B}_1) = \mathcal{G}(\mathcal{B}_2) = \mathcal{G}_3.$$

The former equality does not hold any longer after Step 5 since the second tableau branch is extended with the assertion $as_3 : \mathrm{U}c_2c_3$ while the first one remains unchanged. We then finally obtain the resource graphs $\mathcal{G}(\mathcal{B}_1)$ and $\mathcal{G}(\mathcal{B}_2)$ depicted in Figure 4.

4.2 Measuring and Normalizing Resource Graphs

In this subsection, we introduce the notion of *measure* on a resource graph. The purpose of this notion is to unambiguously determine how many cells are assumed to be actually present in the heap h_x denoted by a label x since the information conveyed in a resource graph is generally an incomplete abstraction. For example in the resource graph $\mathcal{G}(\mathcal{B}_1)$ of

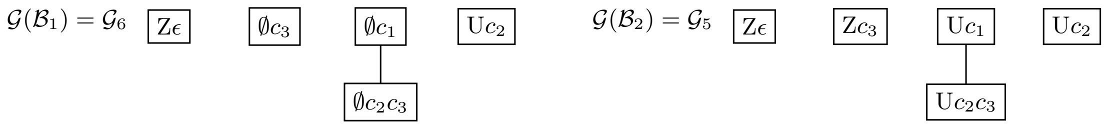


Figure4. Resource Graphs for $\mathcal{T}$.

Figure 4, the tag of $\emptyset c_3$ being empty, it does not give any useful information about the number of locations occuring in the heap denoted by c_3.
A similar approach has been applied in [6,20] in the context of model-checking to encode SL into first order logic and extend it with temporal modalities. Here, the notion of measure is used from a proof-search point of view in order to guide the tableau construction process.

Definition 4.4. *A* measure *on a set X ($\subseteq \mathcal{L} \cup \mathcal{K}$) of label and label constraints is a total function $\mu : X_{\mathcal{L}} \to \mathbb{N}$ which satisfies the following conditions:*

- $\forall\, x \in X_{\mathcal{L}}.$ *if* $\mathrm{Z}x \in X_{\mathcal{K}}$ *then* $\mu(x) = 0$*;*
- $\forall\, x \in X_{\mathcal{L}}.$ *if* $\mathrm{U}x \in X_{\mathcal{K}}$ *then* $\mu(x) = 1$*;*
- $\forall\, x, y \in X_{\mathcal{L}}.$ *if* $x \circ y \in X_{\mathcal{L}}$ *then* $\mu(x \circ y) = \mu(x) + \mu(y)$*;*
- $\forall\, x, y \in X_{\mathcal{L}}.$ *if* $x \diamond y \in X_{\mathcal{K}}$ *then* $\mu(x) = \mu(y)$*.*

A set X ($\subseteq \mathcal{L} \cup \mathcal{K}$) is measurable *iff there exists a measure on it.*
A measure μ on a set $X(\subseteq \mathcal{L} \cup \mathcal{K})$ induces a measure $\mu : N \to \mathbb{N}$ on the corresponding resource graph $\mathcal{G}(X)[N, E]$ defined as follows:

$$\forall\, \Gamma x \in N.\ \mu(\Gamma x) = \mu(x).$$

A resource graph $\mathcal{G}(X)$ is measurable iff X is measurable.

The previous definition gives rise to the notion of *normal resource graph* which is devised to take into account the following semantical facts:

- there is only one empty heap *(Fact 1)*,
- the empty heap is the unit of heap composition *(Fact 2)*.

In general, a resource graph (set of labels and constraints) does not satisfy *(Fact 1)* since there can be several distinct Z-nodes (Z-labels). We remedy the former situation by defining the following notion of Z-equivalence.

Definition 4.5 (Z-equivalence). *Let X ($\subseteq \mathcal{L} \cup \mathcal{K}$) be a set of label and label constraints,*

$$\forall\, x, y \in X_{\mathcal{L}}.\ X \vdash x \simeq y \ \textit{ iff } \ \forall\, c \in \mathcal{C}.\ \mathrm{Z}c \notin X_{\mathcal{K}} \Rightarrow (c \in x \Leftrightarrow c \in y) \quad \textit{(ZEL)}.$$

Let $\mathcal{G}(X)[N, E]$ be the resource graph associated to X,

$$\forall\, \Gamma x, \Delta y \in N.\ \mathcal{G}(X) \vdash \Gamma x \simeq \Delta y \ \textit{ iff } \ X \vdash x \simeq y \quad \textit{(ZEG)}.$$

Intuitively, two labels x and y are equivalent upto $\simeq$ (Z-equivalent) if all of their constant symbols that are not Z-constants are the same.
However, not all nodes in a resource graph are known for sure to denote empty heaps since the information may only be partial. Nevertheless, given a particular measure on a resource graph one can unambiguously determine all the nodes that should denote empty heaps. This additional information given by a measure gives rise to the notion of μ-equivalence.

Definition 4.6 **(μ-equivalence).** *Let X ($\subseteq \mathcal{L} \cup \mathcal{K}$) be a set of label and label constraints,*

$$\forall\, x, y \in X_{\mathcal{L}}.\ X \vdash x \approx y \textit{ iff } \forall\, c \in \mathcal{C}.\ \mu(c) \neq 0 \Rightarrow (c \in x \Leftrightarrow c \in y) \quad (\mu EL);$$

Let $\mathcal{G}(X)[N, E]$ be the resource graph associated to X,

$$\forall\, \Gamma x, \Delta y \in N.\ \mathcal{G}(X) \vdash \Gamma x \approx \Delta y \textit{ iff } X \vdash x \approx y \quad (\mu EG).$$

The following lemma shows that the notion of Z-equivalence is in fact contained in the notion of μ-equivalence.

Lemma 4.1. *Let $X(\subseteq \mathcal{L} \cup \mathcal{K})$ be a set of label and label constraints and μ be a measure on $\mathcal{B}$, then*

1. $\forall\, x, y \in X_{\mathcal{L}}.\ (\mathrm{Z}x \in X_{\mathcal{K}}$ *and* $\mathrm{Z}y \in X_{\mathcal{K}}) \Rightarrow X \vdash x \simeq y$;
2. $\forall\, x, y \in X_{\mathcal{L}}.\ X \vdash x \simeq y \Rightarrow X \vdash x \approx y$.

Similarly, for the resource graph $\mathcal{G}(X)[N, E]$ associated to X:

1. $\forall\, \Gamma x, \Delta y \in N.\ (\mathrm{Z} \in \Gamma$ *and* $\mathrm{Z} \in \Delta) \Rightarrow \mathcal{G}(X) \vdash \Gamma x \simeq \Delta y$;
2. $\forall\, \Gamma x, \Delta y \in N.\ \mathcal{G}(X) \vdash \Gamma x \simeq \Delta y \Rightarrow \mathcal{G}(X) \vdash \Gamma x \approx \Delta y$.

Proof. The first implication is a consequence of the *(Z-Decomposition)* condition of Definition 3.6. Since $\mathrm{Z}z \in X_{\mathcal{K}}$ for all labels z such that $z \subseteq x$ or $z \subseteq y$, in particular, $\mathrm{Z}c \in X_{\mathcal{K}}$ for all constant symbols $c \in \mathcal{C}$ such that $c \in x$ or $c \in y$. Therefore, for all constant symbols $c \in \mathcal{C}$, either $\mathrm{Z}c \in X_{\mathcal{K}}$ and then the condition $\mathrm{Z}c \notin X_{\mathcal{K}} \Rightarrow (c \in x \Leftrightarrow c \in y)$ is trivially satisfied, or $\mathrm{Z}c \notin X_{\mathcal{K}}$ and then $(c \in x \Leftrightarrow c \in y)$ holds because $c \notin x$ and $c \notin y$.
For the second implication, we observe that $\approx$ contains $\simeq$ because for all constant symbols $c \in X_{\mathcal{L}}$, $\mu(c) \neq 0$ implies $\mathrm{Z}c \notin X_{\mathcal{K}}$ by Definition 4.4.

Definition 4.7 **(μ-compatibility).** *Let X ($\subseteq \mathcal{L} \cup \mathcal{K}$) be a set of label and label constraints. An equivalence relation $R_{\mathcal{L}}$ ($\subseteq X_{\mathcal{L}} \times X_{\mathcal{L}}$) between the labels of $X_{\mathcal{L}}$ is* μ-compatible *with a measure μ on X iff*

$$\forall\, x, y \in X_{\mathcal{L}}.\ X \vdash R_{\mathcal{L}}(x, y) \Rightarrow \mu(x) = \mu(y) \quad (\mu CL).$$

Accordingly, an equivalence relation R_N ($\subseteq N \times N$) between the nodes of the resource graph $\mathcal{G}(X)[N, E]$ is μ-compatible *with a measure μ on $\mathcal{G}(X)[N, E]$ iff*

$$\forall\, \Gamma x, \Delta y \in N.\ \mathcal{G}(X) \vdash R_N(\Gamma x, \Delta y) \Rightarrow \mu(\Gamma x) = \mu(\Delta y) \quad (\mu CG).$$

Lemma 4.2. *The μ-equivalence $\approx$ induced by a measure μ on a set X ($\subseteq \mathcal{L} \cup \mathcal{K}$) of label and label constraints (or its associated resource graph $\mathcal{G}(X)$) is μ-compatible with μ.*

Proof. Suppose that $X \vdash x \approx y$ for all labels $x, y \in X_{\mathcal{L}}$. Now let $z = x \cap y$, $x' = x \backslash z$ and $y' = y \backslash z$, *i.e.*, $x = x' \circ z$, $y = y' \circ z$, where z is the label containing the constant symbols shared by x and y. Since $x' \cap y' = \epsilon$ (x and y are disjoint), for all constant symbols c occuring in x', c does not occur in y'. Therefore, *(μEL)* implies that $\mu(c) = 0$ for all constant symbols occurring in x', so that $\mu(x') = 0$. Similarly, we get $\mu(y') = 0$. Consequently, we have $\mu(x) = \mu(x') + \mu(z) = \mu(z) = \mu(y') + \mu(z) = \mu(y)$.

4.3 Quotients of Resource Graphs

In this subsection we formally define what we call the *quotient of a resource graph* by an equivalence relation.

Definition 4.8. *Let $X\,(\subseteq \mathcal{L} \cup \mathcal{K})$ be a set of label and label constraints and $R\,(\subseteq X_{\mathcal{L}} \times X_{\mathcal{L}})$ be an equivalence relation between labels of $X_{\mathcal{L}}$. The quotient of X by R is written $X_{/R}$ and is such that:*

- $X_{/R} = (X_{\mathcal{L}/R}, X_{\mathcal{K}/R})$*;*
- $X_{\mathcal{L}/R} = \{\, \overline{x} \mid x \in X_{\mathcal{L}} \,\}$ *with* $\overline{x} = \{\, y \in X_{\mathcal{L}} \mid R(x, y) \,\}$*;*
- $X_{\mathcal{K}/R} = X^{\mathrm{T}}_{\mathcal{K}/R} \cup X^{\diamond}_{\mathcal{K}/R}$ *with*
    - $X^{\mathrm{T}}_{\mathcal{K}/R} = \{\, \mathrm{T}\overline{x} \mid \mathrm{T} \in \mathbb{T}$ *and* $\overline{x} \in X_{\mathcal{L}/R}$ *and* $\exists\, u \in \overline{x}.\, \mathrm{T}u \in X_{\mathcal{K}} \,\}$ *and*
    - $X^{\diamond}_{\mathcal{K}/R} = \{\, \overline{x} \diamond \overline{y} \mid \overline{x}, \overline{y} \in X_{\mathcal{L}/R}$ *and* $\exists\, u \in \overline{x}.\, \exists\, v \in \overline{y}.\, u \diamond v \in X_{\mathcal{K}} \,\}$*.*

In other words, the elements of $X_{\mathcal{L}/R}$ are equivalence classes of labels modulo R and the elements of $X_{\mathcal{K}/R}$ are constraints between equivalence classes of labels which we call *"label class constraints"* although we more often simply speak of "constraints" when the context is clear. A label class constraint holds between two equivalence classes of labels whenever the corresponding type of label constraint holds between some members u and v of each class. Adapting Definition 3.8 to equivalence classes of labels we get the following definition for the quotient of a resource graph.

Definition 4.9. *Let $\mathcal{G}(X)[N, E]$ be a resource graph and $R\,(\subseteq X_{\mathcal{L}} \times X_{\mathcal{L}})$ be an equivalence relation between labels of $X_{\mathcal{L}}$. The quotient of $\mathcal{G}(X)$ by R is written $\mathcal{G}(X_{/R})[N_{/R}, E_{/R}]$ and is such that:*

- $N_{/R} = \{\, \Gamma\overline{x} \mid x \in X_{\mathcal{L}} \,\}$ *with* $\overline{x} = \{\, y \in X_{\mathcal{L}} \mid R(x, y) \,\}$, $\Gamma = \{\, \mathrm{T} \in \mathbb{T} \mid \exists\, u \in \overline{x}.\, \mathrm{T}u \in X_{\mathcal{K}} \,\}$*;*
- $E_{/R} = \{\, \{\, \Gamma\overline{x}, \Delta\overline{y} \,\} \mid \Gamma\overline{x}, \Delta\overline{y} \in N_{/R}$ *and* $\exists\, u \in \overline{x}.\, \exists\, v \in \overline{y}.\, u \diamond v \in X_{\mathcal{K}} \,\}$*.*

In other words, the first step to compute the quotient $\mathcal{G}(X_{/R})[N_{/R}, E_{/R}]$ of a resource graph $\mathcal{G}(X)[N, E]$ by the equivalence relation R consists in gathering all the labels that are equivalent upto R, thus obtaining a set of label classes $\overline{x} = \{\, y \in X_{\mathcal{L}} \mid R(x, y) \,\}$.
The second step consists in deriving the nodes of the quotient by giving each label class $\overline{x}$ a tag Γ obtained by merging (using set-union) the tags of all the labels populating $\overline{x}$.
Finally, the last step requires putting an edge between two nodes $\Gamma\overline{x}$ and $\Delta\overline{y}$ of the quotient whenever the label classes $\overline{x}$ and $\overline{y}$ respectively contain a label u and a label v for which the corresponding nodes $\eta(u)$ and $\eta(v)$ in the initial graph $\mathcal{G}(X)$ are related by an edge.

Definition 4.10. *Let $\mathcal{G}(X)$ be a resource graph, μ be a measure on $\mathcal{G}(X)$ and $\approx$ be the equivalence induced on $\mathcal{G}(X)$ by μ. The quotient $\mathcal{G}(X_{/\approx})$ is called the* normal resource graph *associated to $(\mathcal{G}(X), \mu)$.*

The resource graph $\mathcal{G}_5$ depicted in Figure 4 contains only Z-nodes and U-nodes so that the first condition required in Definition 4.4 implies that there exists only one measure μ on $\mathcal{G}_5$. Moreover, this measure satisfies the following equations:

- $\mu(\mathrm{Z}\epsilon) = \mu(\mathrm{Z}c_3) = 0$;
- $\mu(\mathrm{U}c_1) = \mu(\mathrm{U}c_2c_3) = \mu(\mathrm{U}c_2) + \mu(\mathrm{Z}c_3) = \mu(\mathrm{U}c_2) = 1$.

Since $\mu(\mathrm{Z}c_3) = 0$ implies $\mathcal{G}_5 \vdash \mathrm{U}c_2 \approx \mathrm{U}c_2c_3$, we obtain the following normal resource graph:

$$\mathcal{G}(\mathcal{B}_{2/\approx}) = \mathcal{G}_{5/\approx} : \quad \boxed{\mathrm{Z}\overline{\epsilon}} \qquad \begin{array}{c} \boxed{\mathrm{U}\overline{c_1}} \\ | \\ \boxed{\mathrm{U}\overline{c_2}} \end{array} \qquad \begin{array}{l} \overline{\epsilon} = \overline{c_3} = \{\, \epsilon, c_3 \,\} \\ \overline{c_1} = \{\, c_1 \,\} \\ \overline{c_2} = \overline{c_2c_3} = \{\, c_2, c_2c_3 \,\} \end{array}$$

4.4 Structural Consistency

In this subsection we introduce the notion of *structural consistency* for resource graphs. Such a notion allows us to determine whether a resource graph can actually represent a model of Separation Logic in the sense of Definition 2.2.

Definition 4.11. *Let $\mathcal{B}$ be a tableau branch, $\Pi(\mathcal{B})$ is the set of points-to predicates*

$$\Pi(\mathcal{B}) = \{\, (l \mapsto a, b) \mid \exists\, x \in \mathcal{L}.\, \mathrm{T}\ (l \mapsto a, b) : x \in \mathcal{B} \,\}.$$

A π-valuation on $\mathcal{B}$ *is a function $\pi : \mathcal{B}_{\mathcal{L}} \to \wp(\Pi(\mathcal{B}))$ from the labels of $\mathcal{B}_{\mathcal{L}}$ to the subsets of $\Pi(\mathcal{B})$ such that:*

$$\forall\, x \in \mathcal{B}_{\mathcal{L}}. \begin{cases} \pi(x) = \{\, (l \mapsto a, b) \mid \mathrm{T}\ (l \mapsto a, b) : x \in \mathcal{B} \,\} \ \text{if } \mathrm{U}x \in \mathcal{B}_{\mathcal{K}} \\ \pi(x) = \emptyset \ \text{otherwise} \end{cases}$$

We then define the set $\Lambda(\mathcal{B})$ as the following extension of $\Pi(\mathcal{B})$:

$$\Lambda(\mathcal{B}) = \Pi(\mathcal{B}) \cup \{\, (L_i \mapsto A_i, B_i) \mid \forall\, i \in \mathbb{N}. \forall\, x, y \in Val.\ (L_i \mapsto x, y) \notin \Pi(\mathcal{B}) \,\}.$$

Finally, given a π-valuation π on $\mathcal{B}$, a λ-interpretation for $(\mathcal{B}, \pi)$ *(more shortly for $\mathcal{B}$) is a function $\lambda : \mathcal{B}_{\mathcal{L}} \to \wp^m(\Lambda(\mathcal{B}))$ extending π which maps each label in $\mathcal{B}_{\mathcal{L}}$ to a* finite multiset over $\Lambda(\mathcal{B})$ *and which satisfies the following conditions:*

- $\forall\, x \in \mathcal{B}_{\mathcal{L}}.\ \lambda(x) \supseteq \pi(x)$ *(π-extension);*
- $\forall\, x, y \in \mathcal{B}_{\mathcal{L}}.$ *if* $x \circ y \in \mathcal{B}_{\mathcal{L}}$ *then* $\lambda(x \circ y) \supseteq \lambda(x) \uplus \lambda(y)$ [5];
- $\forall\, x, y \in \mathcal{B}_{\mathcal{L}}.$ *if* $x \diamond y \in \mathcal{B}_{\mathcal{K}}$ *then* $\lambda(x) = \lambda(y)$.

In terms of the resource graph $\mathcal{G}(\mathcal{B})[N, E]$ associated to the tableau branch $\mathcal{B}$

- *a π-valuation on $\mathcal{G}(\mathcal{B})$ is a function such that $\forall\, \Gamma x \in N.\ \pi(\Gamma x) = \pi(x)$;*

[5] We write multisets using double brackets $\{\!\{ \dots \}\!\}$ and use $\uplus$ to denote multiset-union.

– *a λ-interpretation for $\mathcal{G}(\mathcal{B})$ is a function such that $\forall\, \Gamma x \in N.\ \lambda(\Gamma x) = \lambda(x)$.*

If we apply the previous definition to the second branch of the tableau given in Figure 3 and its associated resource graph $\mathcal{G}_5$, we have to set

$$\pi(\mathrm{U}c_2) = \{\, (l \mapsto a, b)\,\} \text{ and } \pi(\mathrm{U}c_2c_3) = \{\, (k \mapsto c, d)\,\},$$

which leads to the following decorated resource graph:

$$\mathcal{G}(\mathcal{B}_2) = \mathcal{G}_5 : \quad \begin{array}{ccccc} \boxed{\mathrm{Z}\epsilon} & \boxed{\mathrm{Z}c_3} & \boxed{\mathrm{U}c_1} & \boxed{\mathrm{U}c_2}\ \ l \mapsto a, b & \\ & & \Big| & & \\ & & \boxed{\mathrm{U}c_2c_3}\ \ k \mapsto c, d & & \end{array}$$

We now try to extend the previous π-valuation into a λ-interpretation.
Let us first remark that a given π-valuation can in general give rise to several distinct λ-interpretations. For example, a first λ-interpretation λ_1 can be obtained for $\mathcal{G}_5$ if we set

$$\lambda_1(\mathrm{Z}\epsilon) = \lambda_1(\mathrm{Z}c_3) = \emptyset.$$

Trying to satisfy the conditions of Definition 4.11 then leads us to the following equations:

1. $\lambda_1(\mathrm{Z}\epsilon) = \lambda_1(\mathrm{Z}c_3) = \emptyset$
2. $\lambda_1(\mathrm{U}c_2) = \{\!\{\, (l \mapsto a, b)\,\}\!\}$
3. $\lambda_1(\mathrm{U}c_2c_3) = \lambda_1(\mathrm{U}c_2) \uplus \lambda_1(\mathrm{Z}c_3) = \{\!\{\, (l \mapsto a, b), (k \mapsto c, d)\,\}\!\}$
4. $\lambda_1(\mathrm{U}c_1) = \lambda_1(\mathrm{U}c_2c_3) = \{\!\{\, (l \mapsto a, b), (k \mapsto c, d)\,\}\!\}$

Setting $\lambda_2(\mathrm{Z}\epsilon) = \emptyset$ and $\lambda_2(\mathrm{Z}c_3) = \{\!\{\, (l \mapsto a, b)\,\}\!\}$ leads to another λ-interpretation λ_2:

1. $\lambda_2(\mathrm{Z}\epsilon) = \emptyset$
2. $\lambda_2(\mathrm{U}c_2) = \lambda_2(\mathrm{Z}c_3) = \{\!\{\, (l \mapsto a, b)\,\}\!\}$
3. $\lambda_2(\mathrm{U}c_2c_3) = \lambda_2(\mathrm{U}c_2) \uplus \lambda_2(\mathrm{Z}c_3) = \{\!\{\, (l \mapsto a, b), (l \mapsto a, b), (k \mapsto c, d)\,\}\!\}$
4. $\lambda_2(\mathrm{U}c_1) = \lambda_2(\mathrm{U}c_2c_3) = \{\!\{\, (l \mapsto a, b), (l \mapsto a, b), (k \mapsto c, d)\,\}\!\}$

Let us note that since a λ-interpretation maps a node to a multiset, the points-to predicate $(l \mapsto a, b)$ occurs twice in $\lambda_2(\mathrm{U}c_2c_3)$ (and thus also in $\lambda_2(\mathrm{U}c_1)$), one occurrence coming from $\lambda_2(\mathrm{U}c_2)$, the other one coming from $\lambda_2(\mathrm{Z}c_3)$.

Definition 4.12. *Let $\mathcal{B}$ be tableau branch.*
The restriction of a π-valuation π (on $\mathcal{B}$) to its locations, denoted π^L, is such that:

$$\forall\, x \in \mathcal{B}_{\mathcal{L}}.\ \pi^L(x) = \{\, l \mid \exists\, a, b \in Val.\, (l \mapsto a, b) \in \pi(x)\,\}.$$

Similarly, the restriction of a λ-interpretation λ to its locations, denoted λ^L, is such that:

$$\forall\, x \in \mathcal{B}_{\mathcal{L}}.\ l^n \in \lambda^L(x) \textit{ iff } l \textit{ occurs } n \textit{ times in } \lambda(x) \textit{ on the lhs of a points-to predicate.}$$

Let us write $|\alpha|$ for the size of a multiset α, i.e., *the number of elements $\sum_{z^n \in \alpha} n$ it contains.*
A λ-interpretation λ for $\mathcal{B}$ is well-formed *iff*

$$\forall\, x \in \mathcal{B}_{\mathcal{L}}.\ \forall\, l^n \in \lambda^L(x).\, n \leq 1 \qquad \textit{(SC1)}.$$

A λ-interpretation λ for $\mathcal{B}$ is compatible *with a measure μ on $\mathcal{B}$ iff*

$$\begin{cases} \forall\, x \in \mathcal{B}_{\mathcal{L}}.\ |\lambda^L(x)| \leq \mu(x) \quad \textit{(SC2)}; \\ \forall\, x, y \in \mathcal{B}_{\mathcal{L}}.\ \textit{if } \mathcal{B} \vdash x \approx y \textit{ then } \lambda(x) = \lambda(y) \quad \textit{(SC3)}. \end{cases}$$

A λ-interpretation λ for $\mathcal{B}$ is maximally compatible *with a measure μ on $\mathcal{B}$ iff*

$$\lambda \textit{ is compatible with } \mu \textit{ and } \forall\, x \in \mathcal{B}_{\mathcal{L}}.\ |\lambda^L(x)| = \mu(x) \qquad \textit{(SC4)}.$$

Adapting the previous definition in terms of the resource graph $\mathcal{G}_5$ given in Figure 4

$$\pi^L(\mathrm{U}c_2c_3) = \{\, k \,\} \text{ and } \lambda_2^L(\mathrm{U}c_2c_3) = \{\!\{\, l^2, k \,\}\!\}.$$

Therefore, λ_2 is not well-formed since the location l occurs twice in $\lambda_2^L(\mathrm{U}c_2c_3)$. Moreover, λ_2 is not compatible with the measure μ defined on $\mathcal{G}_5$ after Definition 4.10 since it does not satisfy the two conditions *(SC2)* and *(SC3)* of Definition 4.12: *(SC2)* does not hold because $|\lambda_2^L(\mathrm{U}c_2c_3)| = 3 \not\leq 1 = \mu(\mathrm{U}c_2c_3)$ and *(SC3)* does not hold because $\mathcal{G}_5 \vdash \mathrm{U}c_2 \approx \mathrm{U}c_2c_3$ and

$$\lambda_2(\mathrm{U}c_2) = \{\!\{\, (l \mapsto a, b) \,\}\!\} \neq \{\!\{\, (l \mapsto a, b), (l \mapsto a, b), (k \mapsto c, d) \,\}\!\} = \lambda_2(\mathrm{U}c_2c_3).$$

Although λ_1 is well-formed, it is no more compatible with μ than λ_2 because

$$\lambda_1(\mathrm{U}c_2c_3) = \{\!\{\, (l \mapsto a, b), (k \mapsto c, d) \,\}\!\} \text{ implies } |\lambda_1^L(\mathrm{U}c_2c_3)| = 2,$$

which contradicts *(SC2)* since $\mu(\mathrm{U}c_2c_3) = 1$ and obviously $2 \not\leq 1$.

Lemma 4.3. *Let $\mathcal{B}$ for a tableau branch, if λ and λ' are two λ-interpretations for $\mathcal{B}$, then. the function $\lambda \cap \lambda' : \mathcal{B}_{\mathcal{L}} \to \wp^m(\Lambda(\mathcal{B}))$ given by $\forall\, x \in \mathcal{B}_{\mathcal{L}}.\ \lambda \cap \lambda'(x) = \lambda(x) \cap \lambda'(x)$ is a λ-interpretation for $\mathcal{B}$.*

Proof. We show that $\lambda \cap \lambda'$ satisfies all the conditions required by Definition 4.11 for a λ-interpretation:

1. for all label $x \in \mathcal{B}_{\mathcal{L}}$, we have $\lambda \cap \lambda'(x) \supseteq \pi(x)$ since by definition of a λ-interpretation $\lambda(x) \supseteq \pi(x)$ and $\lambda'(x) \supseteq \pi(x)$, which implies $\lambda(x) \cap \lambda'(x) \supseteq \pi(x)$;
2. given that for all labels $x, y \in \mathcal{B}_{\mathcal{L}}$ such that $x \circ y \in \mathcal{B}_{\mathcal{L}}$

$$\begin{array}{ll} \text{we have both} & \lambda(x \circ y) \supseteq \lambda(x) \uplus \lambda(y) \text{ and } \lambda'(x \circ y) \supseteq \lambda'(x) \uplus \lambda'(y), \\ \text{we get} & \lambda(x \circ y) \cap \lambda'(x \circ y) \supseteq (\lambda(x) \uplus \lambda(y)) \cap (\lambda'(x) \uplus \lambda'(y)), \\ \text{which implies} & \lambda(x \circ y) \cap \lambda'(x \circ y) \supseteq (\lambda(x) \cap \lambda'(x)) \uplus (\lambda(y) \cap \lambda'(y)), \\ \text{and finally} & \lambda \cap \lambda'(x \circ y) \supseteq \lambda \cap \lambda'(x) \uplus \lambda \cap \lambda'(y); \end{array}$$

3. given that for all labels $x, y \in \mathcal{B}_{\mathcal{L}}$ such that $x \diamond y \in \mathcal{B}_{\mathcal{K}}$ we have $\lambda(x) = \lambda(y)$ and $\lambda'(x) = \lambda'(y)$, we get $\lambda(x) \cap \lambda'(x) = (\lambda(y) \cap \lambda'(y))$, which implies $\lambda \cap \lambda'(x) = \lambda \cap \lambda'(y)$.

Corollary 4.1. *Let $\mathcal{B}$ for a tableau branch, the function $\lambda_m : \mathcal{B}_{\mathcal{L}} \to \wp^m(\Lambda(\mathcal{B}))$ given by*

$$\forall\, x \in \mathcal{B}_{\mathcal{L}}.\ \lambda_m(x) = \cap \{\, \lambda(x) \mid \lambda \textit{ is a } \lambda\textit{-interpretation for } \mathcal{B} \,\}$$

is the smallest possible λ-interpretation for $\mathcal{B}$.

Proof. It is an easy consequence of Lemma 4.3.

Let us finally remark that there exists no compatible λ-interpretation for the resource graph $\mathcal{G}_5$ since λ_1 is not compatible with $\mathcal{G}_5$ and λ_1 is in fact λ_m, *i.e.*, the smallest possible λ-interpretation for $\mathcal{G}_5$.

Definition 4.13 (structural consistency). *Let $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) be a tableau branch (resource graph), μ be a measure on $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) and λ be a λ-interpretation for $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$). We say that $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) is structurally consistent with (μ, λ) iff λ is well-formed and maximally compatible with μ.*
Accordingly, $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) is structurally consistent with μ iff there exists a λ-interpretation such that $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) is structurally consistent with (μ, λ) and $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) is structurally consistent iff there exists a measure for which it is structurally consistent with.

The notion of structural consistency means that a resource graph actually represents a "real" model of SLP. For example, the resource graph $\mathcal{G}_5$ is not structurally consistent since it cannot be compatible with any measure on $\mathcal{G}_5$ as explained previously.

4.5 Logical Consistency

Before we proceed with the notion of *logical consistency*, which intuitively means that a formula of SLP can be falsified, we need to introduce a new equivalence relation.

Definition 4.14 ($\lambda\mu$-equivalence). *Let $\mathcal{B}$ be a tableau branch. Given a measure μ on $\mathcal{B}$ and a λ-interpretation λ for $\mathcal{B}$, the relation $\sim$ ($\subseteq \mathcal{B}_\mathcal{L} \times \mathcal{B}_\mathcal{L}$) is the smallest equivalence induced by μ and λ such that:*

$$\forall\, x, y \in \mathcal{B}_\mathcal{L}.\ \mathcal{B} \vdash x \sim y \ \text{ iff } \ \mu(x) = \mu(y) \text{ and } \lambda(x) = \lambda(y).$$

For the resource graph $\mathcal{G}(\mathcal{B})[N, E]$ associated to $\mathcal{B}$

$$\forall\, \Gamma x, \Delta y \in N.\ \mathcal{G}(\mathcal{B}) \vdash \Gamma x \sim \Delta y \ \text{ iff } \ \mathcal{B} \vdash x \sim y.$$

Lemma 4.4. *If μ is a measure on a tableau branch $\mathcal{B}$ and λ is a λ-interpretation for $\mathcal{B}$ that is compatible with μ then*

$$\forall\, x, y \in \mathcal{B}_\mathcal{L}.\ \text{if } \mathcal{B} \vdash x \approx y \text{ then } \mathcal{B} \vdash x \sim y.$$

Proof. Let $x, y \in \mathcal{B}_\mathcal{L}$ such that $\mathcal{B} \vdash x \approx y$. Since λ is compatible with μ, condition *(SC3)* of Definition 4.12 implies $\lambda(x) = \lambda(y) \quad (a)$.
Now let $z = x \cap y$, $x' = x \backslash z$ and $y' = y \backslash z$, *i.e.*, $x = x' \circ z$ and $y' = y \circ z$, where z is the label containing all the constant symbols shared by x and y. The definition of $\mathcal{B} \vdash x \approx y$ then implies that $\forall\, c \in \mathcal{C}$. if $c \in (x' \circ y')$ then $\mu(c) = 0$. Therefore, $\mu(x') = \mu(y') = 0$ and

$$\mu(x) = \mu(x') + \mu(z) = \mu(z) = \mu(y') + \mu(z) = \mu(y) \quad (b).$$

Finally, *(a)* and *(b)* implies that $\mathcal{B} \vdash x \sim y$.

Definition 4.15 (logical consistency). *Let $\mathcal{B}$ be a tableau branch and μ be a measure on $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$), $\mathcal{B}$ is* logically consistent with *μ if none of the following conditions holds:*

– $\exists\, x, y \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{T}\,(l \mapsto a, b) : x \in \mathcal{B},\ \mathrm{F}\,(l \mapsto a, b) : y \in \mathcal{B}$ *and* $\mathcal{B} \vdash x \sim y$ *(LC1);*
– $\exists\, x \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{F}\,\mathrm{I} : x \in \mathcal{B}$ *and* $\mu(x) = 0$ *(LC2);*
– $\exists\, x \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{F}\,\top : x \in \mathcal{B}$ *(LC3);*
– $\exists\, x \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{T}\,\bot : x \in \mathcal{B}$ *(LC4).*

A tableau branch $\mathcal{B}$ is logically consistent *if it is logically consistent with some measure on $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$).*

Looking back at the tableau of Figure 3 we can see that the second branch $\mathcal{B}_2$ is logically consistent for the only measure μ definable on $\mathcal{G}(\mathcal{B}_2) = \mathcal{G}_5$ because, on one hand, the fact that the initial formula does not contain any occurrence of I, $\top$ or $\bot$ implies that none of the conditions *(LC2)* – *(LC4)* holds and, on the other hand, the fact that $\mathcal{B}_2$ does not contain any points-to predicate with sign F implies that condition *(LC1)* does not hold either. On the contrary, the first branch $\mathcal{B}_1$ is not logically consistent because

$$\mathrm{T}\,(l \mapsto a, b) : c_2 \in \mathcal{B}_1,\ \mathrm{F}\,(l \mapsto a, b) : c_2 \in \mathcal{B}_1$$

and for all measures on $\mathcal{G}(\mathcal{B}_1)$, $c_2 = c_2$ implies that $c_2 \sim c_2$ holds in $\mathcal{B}_1$ ($\mathcal{B}_1 \vdash c_2 \sim c_2$).

Definition 4.16. *A tableau branch $\mathcal{B}$ is* open *iff there exists a measure μ on $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) such that $\mathcal{B}$ ($\mathcal{G}(\mathcal{B})$) is structurally consistent with μ and $\mathcal{B}$ is logically consistent with μ. A tableau branch that is not open is said to be* closed*. A tableau is open if it contains at least one open branch, it is closed otherwise.*

Definition 4.17 (provability). *A tableau $\mathcal{T}$ is a* TSLP*-proof of $\psi \models \phi$ iff there is a finite sequence of tableaux $(\mathcal{T}_i)_{1 \leq i \leq n}$ such that:*

1. *$\mathcal{T}_1$ is the one-node tableau the root of which is labelled with $\mathrm{F}\,\psi \mathrel{-\!\!*} \phi : \epsilon$;*
2. *$\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ by a rule of Figure 1;*
3. *$\mathcal{T}_n = \mathcal{T}$ and $\mathcal{T}$ is closed.*

An entailment $\psi \models \phi$ is provable in TSLP *if there exists a* TSLP*-proof of $\psi \models \phi$.*

Let us summarize all the information about the tableau $\mathcal{T}$ of Figure 3.
On one hand, we know that for all the measures definable on the first branch $\mathcal{B}_1$, $\mathcal{B}_1$ is logically inconsistent. On the other hand, we know that the only measure definable on the second branch $\mathcal{B}_2$ makes the resource graph $\mathcal{G}(\mathcal{B}_2)$ structurally inconsistent.
Therefore, according to Definition 4.16, $\mathcal{B}_1$ and $\mathcal{B}_2$ are closed, which implies that $\mathcal{T}$ is closed and is a TSLP-proof of $(l \mapsto a, b) * ((l \mapsto a, b) \mathrel{-\!\!*} (k \mapsto c, d)) \models (m \mapsto e, f)$.

4.6 Theorem-proving vs. Model-checking

Before proving the main properties of the tableau calculus let us emphasize the interest of the theorem-proving approach. Compared to model-checking, which is the main approach considered for Separation Logic so far [2], theorem-proving aims at discovering all classes of models for which a formula is satisfied. As seen in this section, a key feature of our tableau method is that provability is captured through two distinct notions: structural and logical consistency. The first one ensures that a resource graph actually denotes a

model in the class of Separation Logic models while the second one ensures that a formula can be falsified in some model.
Such an approach allows us to distinguish whether a formula is valid for intrinsic logical reasons (for example $\phi \rightarrow \phi$ should be valid in any "reasonable" resource logic) or because the conditions required for a given structure to be a model in the class of Separation Logic models are too restrictive to allow the existence of a model. This key point also makes the method more modular as one can change the conditions of structural consistency in order to match other classes of resource models. For example, one could consider resource models for which the composition is not an aggregation, *i.e.*, models for which the condition $\mu(x \circ y) = \mu(x) + \mu(y)$ does not hold in general.

4.7 Another Example

We complete this section with another example that illustrates some key points of the tableau construction and the related characterization of validity.

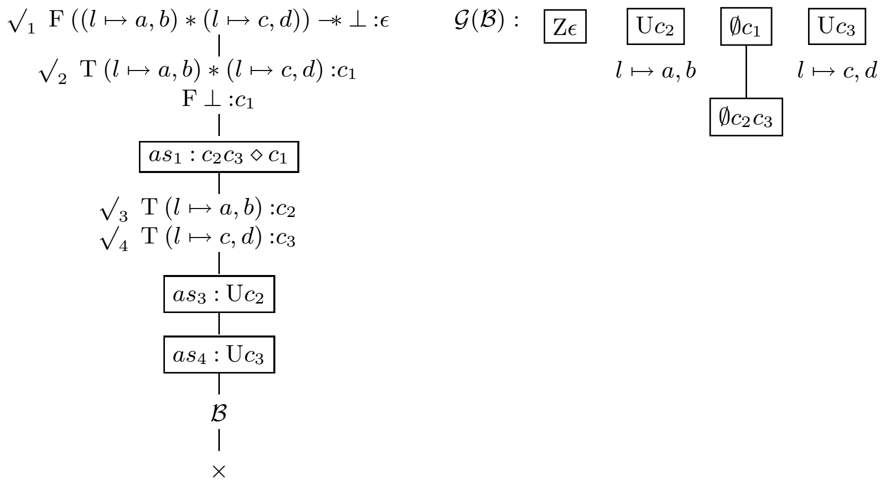


Figure5. Tableau for $(l \mapsto a, b) * (l \mapsto c, d) \models \perp$.

Figure 5 gives a tableau for $(l \mapsto a, b) * (l \mapsto c, d) \models \perp$ which means that a heap cannot have two distinct cells at the same location. The tableau has only one branch $\mathcal{B}$ and its resource graph $\mathcal{G}(\mathcal{B})$ (after Step 4) admits only one measure μ which is such that:

$$\mu(\mathrm{Z}\epsilon) = 0,\ \mu(\mathrm{U}c2) = \mu(\mathrm{U}c3) = 1,\ \mu(\emptyset c_1) = \mu(\emptyset c_2c_3) = \mu(\mathrm{U}c_2) + \mu(\mathrm{U}c_3) = 2.$$

Moreover, the tableau branch $\mathcal{B}$ induces the following π-valuation on $\mathcal{G}(\mathcal{B})$:

$$\pi(\mathrm{U}c_2) = \{\, (l \mapsto a, b) \,\},\ \pi(\mathrm{U}c_3) = \{\, (l \mapsto c, d) \,\}.$$

According to Definition 4.11, any λ-interpretation should then verify the equation

$$\lambda(\emptyset c_2c_3) \supseteq \lambda(\emptyset c_2) \uplus \lambda(\emptyset c_3) = \{\!\{\, (l \mapsto a, b), (l \mapsto c, d) \,\}\!\}.$$

Since the previous equation implies that the location l should occur at least twice in any λ-interpretation, it is clear that no λ-interpretation can satisfy the condition *(SC1)* of Definition 4.12 and thus no λ-interpretation can be well-formed.
Consequently, Definition 4.13 implies that, for all measures μ and λ-interpretations λ, the resource graph $\mathcal{G}(\mathcal{B})$ is not structurally consistent with (μ, λ) which, by Definition 4.16, finally implies that the tableau branch $\mathcal{B}$ is closed.
Therefore, the tableau of Figure 5 is a TSLP-proof of $(l \mapsto a, b) * (l \mapsto c, d) \models \bot$.

5 Properties of the Tableau Calculus

In this section we study and prove the main properties of the tableau calculus, namely soundness, completeness and termination.

5.1 Soundness

Before we proceed with the soundness proof, let us first introduce the notation $Size(h)$ to denote the size of a heap h, *i.e.*, the number of locations it contains.

Definition 5.1 (realization). *Let $\mathcal{B}$ be a tableau branch. A* realization of $\mathcal{B}$ *is a function $\|-\| : \mathcal{B}_{\mathcal{L}} \to \mathcal{H}$ satisfying the following conditions:*

- $\forall x, y \in \mathcal{B}_{\mathcal{L}}.\ x \circ y \in \mathcal{B}_{\mathcal{L}} \Rightarrow \|x \circ y\| = \|x\| \cdot \|y\|$;
- $\forall x \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{Z}x \in \mathcal{B}_{\mathcal{K}} \Rightarrow Size(\|x\|) = 0$;
- $\forall x \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{U}x \in \mathcal{B}_{\mathcal{K}} \Rightarrow Size(\|x\|) = 1$;
- $\forall x, y \in \mathcal{B}_{\mathcal{L}}.\ x \diamond y \in \mathcal{B}_{\mathcal{K}} \Rightarrow \|x\| = \|y\|$;
- $\forall x \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{T}\,\phi : x \in \mathcal{B} \Rightarrow \|x\| \models \phi$;
- $\forall x \in \mathcal{B}_{\mathcal{L}}.\ \mathrm{F}\,\phi : x \in \mathcal{B} \Rightarrow \|x\| \not\models \phi$.

A tableau branch $\mathcal{B}$ is realizable *if there exists a realization of $\mathcal{B}$.*
A tableau $\mathcal{T}$ is realizable *if it contains a realizable branch.*

Let us remark that the first condition, namely $\|x \circ y\| = \|x\| \cdot \|y\|$, implies that a realization $\|-\|$ is completely determined by its values on the constants occurring in $\mathcal{B}_{\mathcal{L}}$. Moreover, it is not difficult to see that $\mathcal{B} \vdash x \approx y \Rightarrow \|x\| = \|y\|$.

Lemma 5.1. *If a tableau branch $\mathcal{B}$ is closed then it is not realizable.*

Proof. We show that if $\mathcal{B}$ is realizable for some realization $\|-\|$ then $\mathcal{B}$ is open.

1. Let us first show that the resource graph $\mathcal{G}(\mathcal{B})[N, E]$ associated to the branch $\mathcal{B}$ is structurally consistent. For that we need to show that there exists some measure μ on $\mathcal{G}(\mathcal{B})$ and a λ-interpretation λ for $\mathcal{G}(\mathcal{B})$ which is well-formed and maximally compatible with μ. It is not difficult (although quite lengthy) to show that such a pair (μ, λ) can be obtained by setting:

$$\forall \Gamma x \in N.\ \begin{cases} \mu(\Gamma x) = Size(\|x\|) \\ (l \mapsto a, b) \in \lambda(\Gamma x) \text{ iff } \|x\|(l) = (a, b) \end{cases}$$

The four conditions *(SC1)–(SC4)* are enforced by the previous definition since a heap $\|x\|$ cannot associate more than one cell to a location (condition *(SC1)*), contains exactly as many cells as locations in its domain (conditions *(SC2)* and *(SC4)*) and finally, if $\mathcal{B} \vdash x \approx y$ then $\|x\| = \|y\|$, which obviously implies that $\|x\|$ and $\|y\|$ have the same cells associated to the same locations (condition *(SC3)*).
2. We now show that if $\mathcal{B}$ is realizable then it is logically consistent. Suppose otherwise, then at least one of the conditions *(LC1)–(LC4)* of Definition 4.15 holds. Let us assume that *(LC1)* holds (other cases are similar), then, there is a points-to predicate $(l \mapsto a, b)$ such that $\mathrm{T}\,(l \mapsto a, b) :x \in \mathcal{B}$, $\mathrm{F}\,(l \mapsto a, b) :y \in \mathcal{B}$ and $\mathcal{B} \vdash x \sim y$ for some labels x and y. Since $\|{-}\|$ is a realization of $\mathcal{B}$, we deduce $\|x\| \models (l \mapsto a, b)$, $\|y\| \not\models (l \mapsto a, b)$ and $\|x\| = \|y\|$, which is a contradiction.

From (1) and (2) we can conclude that $\mathcal{B}$ is open according to Definition 4.16.

Lemma 5.2. *The rules of* TSLP *preserve realizability.*

Proof. We show that, for all realizable tableaux $\mathcal{T}$, if $\mathcal{T}'$ is a tableau obtained from $\mathcal{T}$ by application of a tableau rule, then $\mathcal{T}'$ is realizable. Since $\mathcal{T}$ is realizable, it contains a branch $\mathcal{B}$ which is realizable for some realization $\|{-}\|$. If the signed formula $\mathrm{S}\,\phi :x$ that has been expanded to obtain $\mathcal{T}'$ does not belong to $\mathcal{B}$, then $\mathcal{T}'$ is realizable since it still contains $\mathcal{B}$. Otherwise, we show by case analysis on $\mathrm{S}\,\phi :x$ that the corresponding expansion rule preserves realizability.

- case $\mathrm{T}\,\phi * \psi :x$
  $\mathcal{B}$ is extended into the branch $\mathcal{B}'$ such that $\mathcal{B}' = \mathcal{B} + \mathrm{T}\,\phi :c_i + \mathrm{T}\,\psi :c_j + as : c_i c_j \diamond x$, c_i and c_j being new constants. Since $\|{-}\|$ realizes $\mathcal{B}$, we have $\|x\| \models \phi * \psi$. Therefore, there exist two heaps h_1 and h_2 such that $h_1 \# h_2$, $h_1 \cdot h_2 = \|x\|$, $h_1 \models \phi and h_2 \models \psi$. We then only need to extend $\|{-}\|$ to c_i and c_j by setting $\|c_i\| = h_1$ and $\|c_j\| = h_2$ in order to obtain $\|c_i\| \models \phi$ and $\|c_j\| \models \psi$. Since $\|c_i\| \cdot \|c_j\| = \|x\|$, $\mathcal{B}'$ is realizable and, consequently, $\mathcal{T}'$ is realizable.
- case $\mathrm{F}\,\phi * \psi :x$
  $\mathcal{B}$ splits into two branches $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{B}_1 = \mathcal{B} + \mathrm{F}\,\phi :y + rq : (y \circ z) \diamond x$ and $\mathcal{B}_2 = \mathcal{B} + \mathrm{F}\,\psi :z + rq : (y \circ z) \diamond x$. An admissible application of the F$*$ rule requires that $\mathcal{B} \vdash (y \circ z) \diamond x$, which implies $\|y\| \cdot \|z\| = \|x\|$. Since $\|{-}\|$ realizes $\mathcal{B}$, we have $\|x\| \not\models \phi * \psi$. Therefore, for all heaps h_1 and h_2 such that $h_1 \# h_2$ and $h_1 \cdot h_2 = \|x\|$, either $h_1 \not\models \phi$, or $h_2 \not\models \psi$, which implies that either $\|y\| \not\models \phi$, or $\|z\| \not\models \psi$. Then, either $\mathcal{B}_1$, or $\mathcal{B}_2$ is realizable and, consequently, $\mathcal{T}'$ is realizable.
- other cases are similar.

Theorem 5.1 (soundness). *If there exists a* TSLP*-proof of* $\psi \models \phi$ *then the entailment* $\psi \models \phi$ *holds in* SLP*.*

Proof. Let $(\mathcal{T}_i)_{1 \le i \le n}$ be a TSL-proof of $\psi \models \phi$. Suppose that $\psi \models \phi$ does not hold in SLP, then $e \not\models \psi \mathrel{-\!\!*} \phi$. Consequently, $\|\epsilon\| = e$ is a trivial realization of $\mathcal{T}_0$. Lemma 5.2 then entails that all tableaux in $(\mathcal{T}_i)_{1 \le i \le n}$ are realizable. This is a contradiction because $\mathcal{T}_n$ is closed by definition of a TSL-proof, which implies that $\mathcal{T}_n$ is not realizable by Lemma 5.1.

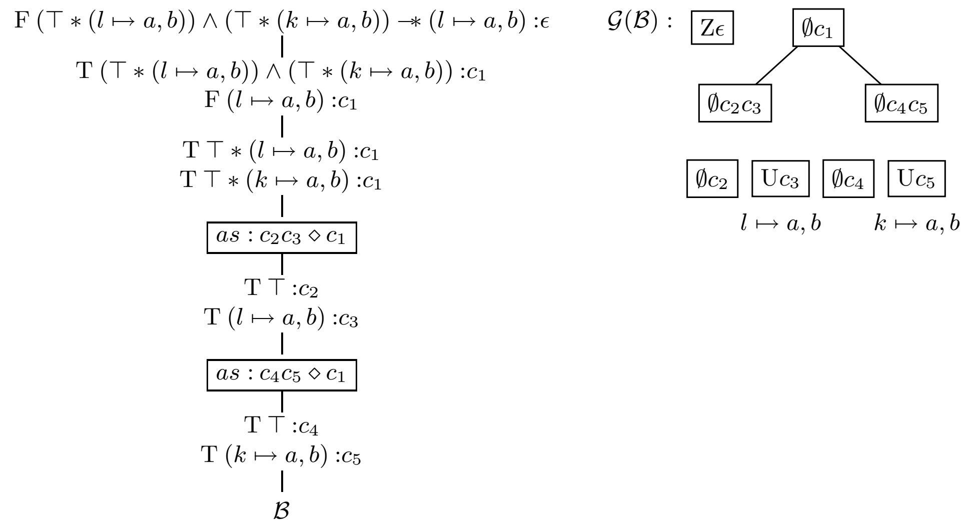


Figure6. Open Tableau for $(\top * (l \mapsto a, b)) \wedge (\top * (k \mapsto a, b)) \models (l \mapsto a, b)$

5.2 Completeness and Countermodel Generation

We give here a proof of completeness of the calculus based on countermodel construction.

Definition 5.2. *A signed formula* $\mathrm{S}\,\phi :x$ *is* analyzed *in a branch* $\mathcal{B}$*, denoted* $\mathcal{B} \succ \mathrm{S}\,\phi :x$*, iff* $\mathrm{S}\,\phi :y \in \mathcal{B}$ *for some label* $y \in \mathcal{B}_{\mathcal{L}}$ *such that* $\mathcal{B} \vdash x \approx y$*.*

Definition 5.3. *A signed formula* $\mathrm{S}\,\phi :x$ *is* completely analyzed *or* fulfilled *in a branch* $\mathcal{B}$*, denoted* $\mathcal{B} \Vdash \mathrm{S}\,\phi :x$*, if it matches one of the following cases:*

- $\mathcal{B} \Vdash \mathrm{T}\,\mathrm{I} :x$ *iff* $\mathcal{B} \succ \mathrm{T}\,\mathrm{I} :x$ *and* $\mathcal{B} \vdash \mathrm{Z}x$*;*
- $\mathcal{B} \Vdash \mathrm{T}\,(l \mapsto a, b) :x$ *iff* $\mathcal{B} \vdash \mathrm{U}x$*;*
- $\mathcal{B} \Vdash \mathrm{T}\,\psi \wedge \chi :x$ *iff* $\mathcal{B} \succ \mathrm{T}\,\psi :x$ *and* $\mathcal{B} \succ \mathrm{T}\,\chi :x$*;*
- $\mathcal{B} \Vdash \mathrm{F}\,\psi \vee \chi :x$ *iff* $\mathcal{B} \succ \mathrm{F}\,\psi :x$ *and* $\mathcal{B} \succ \mathrm{F}\,\chi :x$*;*
- $\mathcal{B} \Vdash \mathrm{T}\,\psi \vee \chi :x$ *iff* $\mathcal{B} \succ \mathrm{T}\,\psi :x$ *or* $\mathcal{B} \succ \mathrm{T}\,\chi :x$*;*
- $\mathcal{B} \Vdash \mathrm{F}\,\psi \rightarrow \chi :x$ *iff* $\mathcal{B} \succ \mathrm{T}\,\psi :x$ *and* $\mathcal{B} \succ \mathrm{F}\,\chi :x$*;*
- $\mathcal{B} \Vdash \mathrm{T}\,\psi \rightarrow \chi :x$ *iff* $\mathcal{B} \succ \mathrm{F}\,\psi :x$ *or* $\mathcal{B} \succ \mathrm{T}\,\chi :x$*;*
- $\mathcal{B} \Vdash \mathrm{F}\,\psi * \chi :x$ *iff* $(\forall\, y, z \in \mathcal{B}_{\mathcal{L}})$ $(\mathcal{B} \vdash (y \circ z) \approx x \Rightarrow (\mathcal{B} \succ \mathrm{F}\,\psi :y$ *or* $\mathcal{B} \succ \mathrm{F}\,\chi :z))$*;*
- $\mathcal{B} \Vdash \mathrm{T}\,\psi * \chi :x$ *iff* $(\exists\, y, z \in \mathcal{B}_{\mathcal{L}})$ $(\mathcal{B} \vdash (y \circ z) \approx x$ *and* $\mathcal{B} \succ \mathrm{T}\,\psi :y$ *and* $\mathcal{B} \succ \mathrm{T}\,\chi :z)$*;*
- $\mathcal{B} \Vdash \mathrm{F}\,\psi \mathrel{-\!\!*} \chi :x$ *iff* $(\exists\, y \in \mathcal{B}_{\mathcal{L}})$ $(x \circ y \in \mathcal{B}_{\mathcal{L}}$ *and* $\mathcal{B} \succ \mathrm{T}\,\psi :y$ *and* $\mathcal{B} \succ \mathrm{F}\,\chi :x \circ y)$*;*
- $\mathcal{B} \Vdash \mathrm{T}\,\psi \mathrel{-\!\!*} \chi :x$ *iff* $(\forall\, y \in \mathcal{B}_{\mathcal{L}})$ $(x \circ y \in \mathcal{B}_{\mathcal{L}} \Rightarrow (\mathcal{B} \succ \mathrm{F}\,\psi :y$ *or* $\mathcal{B} \succ \mathrm{T}\,\chi :x \circ y))$*;*
- *for all other cases,* $\mathcal{B} \Vdash \mathrm{S}\,\phi :x$ *iff* $\mathcal{B} \succ \mathrm{S}\,\phi :x$*.*

Definition 5.4. *A branch* $\mathcal{B}$ *is* complete *iff it is open and all signed formulas in* $\mathcal{B}$ *are fulfilled. A tableau* $\mathcal{T}$ *is* complete *iff it contains a complete branch.*

It is standard to define a tableau construction procedure that builds either a closed tableau or a complete tableau [14].
Let us now explain how to construct a countermodel from a complete and open branch using the tableau depicted in Figure 6. The first thing we need to check is whether the branch $\mathcal{B}$ is open, *i.e.*, if it is both structurally and logically consistent for some measure μ on $\mathcal{G}(\mathcal{B})$ and some λ-interpretation λ for $\mathcal{G}(\mathcal{B})$ such that λ is well-formed and maximally compatible with μ.
Let us first prove that $\mathcal{G}(\mathcal{B})$ is measurable. In this example, the resource graph $\mathcal{G}(\mathcal{B})$ implies that all measures μ on $\mathcal{G}(\mathcal{B})$ should satisfy the following equations:

$$\begin{cases} \mu(\mathrm{Z}\epsilon) = 0 \\ \mu(\mathrm{U}c_3) = \mu(\mathrm{U}c_5) = 1 \\ \mu(\emptyset c_1) = \mu(\emptyset c_2 c_3) = \mu(\emptyset c_2) + \mu(\mathrm{U}c_3) \\ \mu(\emptyset c_1) = \mu(\emptyset c_4 c_5) = \mu(\emptyset c_4) + \mu(\mathrm{U}c_5) \end{cases}$$

The previous system of μ-equations admits infinitely many solutions of the following form: for all $X \in \mathbb{N}$, μ is a measure on $\mathcal{G}(\mathcal{B})$ iff:

$$\begin{cases} \mu(\mathrm{Z}\epsilon) = 0 \\ \mu(\mathrm{U}c_3) = \mu(\mathrm{U}c_5) = 1 \\ \mu(\emptyset c_2) = \mu(\emptyset c_4) = X \\ \mu(\emptyset c_1) = \mu(\emptyset c_2 c_3) = \mu(\emptyset c_4 c_5) = X + 1 \end{cases}$$

Then we take into account the conditions required by Definition 4.11. We know that all λ-interpretations λ for $\mathcal{G}(\mathcal{B})$ should satisfy the following equations:

$$\begin{cases} \lambda(\emptyset c_1) = \lambda(\emptyset c_2 c_3) \supseteq \lambda(\emptyset c_2) \uplus \lambda(\mathrm{U}c_3) \supseteq \{\!\{\, (l \mapsto a, b), (k \mapsto a, b) \,\}\!\} \\ \lambda(\emptyset c_1) = \lambda(\emptyset c_4 c_5) \supseteq \lambda(\emptyset c_4) \uplus \lambda(\mathrm{U}c_5) \supseteq \{\!\{\, (l \mapsto a, b), (k \mapsto a, b) \,\}\!\} \\ |\lambda^L(\emptyset c_1)| = |\lambda^L(\emptyset c_2 c_3)| = |\lambda^L(\emptyset c_4 c_5)| \geq 2 \end{cases}$$

Now if $\mathcal{B}$ is to be open, we can make use of Definition 4.12 to help us narrow down the set of admissible measures on $\mathcal{G}(\mathcal{B})$. Indeed, in order to satisfy condition *(SC4)*, any given measure μ must be such that:

$$\begin{cases} |\lambda^L(\mathrm{Z}\epsilon)| = 0 \\ |\lambda^L(\mathrm{U}c_3)| = |\lambda^L(\mathrm{U}c_5)| = \mu(\mathrm{U}c_5) = \mu(\mathrm{U}c_3) = 1 \\ |\lambda^L(\emptyset c_2)| = |\lambda^L(\emptyset c_4)| = \mu(\emptyset c_2) = \mu(\emptyset c_4) = X \\ 2 \leq |\lambda^L(\emptyset c_4 c_5)| = |\lambda^L(\emptyset c_2 c_3)| = |\lambda^L(\emptyset c_1)| = \mu(\emptyset c_1) = \mu(\emptyset c_2 c_3) = \mu(\emptyset c_4 c_5) = X + 1 \end{cases}$$

We can therefore deduce that $X \geq 1$.
Let us try the measure μ induced by setting $X = 1$. In this case, we obtain:

$$\begin{cases} \mu(\mathrm{Z}\epsilon) = 0 \\ \mu(\mathrm{U}c_3) = \mu(\mathrm{U}c_5) = 1 \\ \mu(\emptyset c_2) = \mu(\emptyset c_4) = 1 \\ \mu(\emptyset c_1) = \mu(\emptyset c_2 c_3) = \mu(\emptyset c_4 c_5) = 2 \end{cases} \qquad \begin{cases} |\lambda^L(\mathrm{Z}\epsilon)| = \mu(\mathrm{Z}\epsilon) = 0 \\ |\lambda^L(\mathrm{U}c_3)| = |\lambda^L(\mathrm{U}c_5)| = 1 \\ |\lambda^L(\emptyset c_2)| = |\lambda^L(\emptyset c_4)| = 1 \\ |\lambda^L(\emptyset c_1)| = |\lambda^L(\emptyset c_2 c_3)| = |\lambda^L(\emptyset c_4 c_5)| = 2 \end{cases}$$

Since $|\lambda^L(\emptyset c_2)| = |\lambda^L(\emptyset c_4)| = 1$, we must complete $\lambda(\emptyset c_2)$ and $\lambda(\emptyset c_4)$ so that they contain exactly one points-to predicate. Since we know nothing about these points-to predicates for the moment we simply set

$$\lambda(\emptyset c_2) = \{\!\{\, (L_1 \mapsto A_1, B_1) \,\}\!\} \text{ and } \lambda(\emptyset c_4) = \{\!\{\, (L_2 \mapsto A_2, B_2) \,\}\!\}.$$

We can now consider L_i, A_i, B_i $(1 \leq i \leq 2)$ as variables in a process of multiset unification in order to solve the following system of λ-equations:

$$\begin{cases} \lambda(\mathrm{Z}\epsilon) = \emptyset \\ \lambda(\mathrm{U}c_3) = \{\!\{\, (l \mapsto a, b) \,\}\!\} \\ \lambda(\mathrm{U}c_5) = \{\!\{\, (k \mapsto a, b) \,\}\!\} \\ \lambda(\emptyset c_2) = \{\!\{\, (L_1 \mapsto A_1, B_1) \,\}\!\} \\ \lambda(\emptyset c_4) = \{\!\{\, (L_2 \mapsto A_2, B_2) \,\}\!\} \\ \lambda(\emptyset c_1) = \lambda(\emptyset c_2 c_3) = \lambda(\emptyset c_4 c_5) = \{\!\{\, (l \mapsto a, b), (k \mapsto a, b) \,\}\!\} \end{cases}$$

Having $|\lambda^L(\emptyset c_1)| = |\lambda^L(\emptyset c_2 c_3)| = 2$ on one hand and $|\lambda^L(\emptyset c_2)| = |\lambda^L(\mathrm{U}c_3)| = 1$ on the other hand implies that $\lambda(\emptyset c_2 c_3) = \lambda(\emptyset c_2) \uplus \lambda(\mathrm{U}c_3)$. A similar argument then also yields $\lambda(\emptyset c_4 c_5) = \lambda(\emptyset c_4) \uplus \lambda(\mathrm{U}c_5)$, so that we obtain the two equations:

$$\begin{cases} \{\!\{\, (l \mapsto a, b), (k \mapsto a, b) \,\}\!\} = \{\!\{\, (L_1 \mapsto A_1, B_1), (l \mapsto a, b) \,\}\!\} \\ \{\!\{\, (l \mapsto a, b), (k \mapsto a, b) \,\}\!\} = \{\!\{\, (L_2 \mapsto A_2, B_2), (k \mapsto a, b) \,\}\!\} \end{cases}$$

for which a solution is given by

$$(L_1 \mapsto A_1, B_1) = (k \mapsto a, b) \text{ and } (L_2 \mapsto A_2, B_2) = (l \mapsto a, b).$$

Finally, we have found a measure μ on $\mathcal{G}(\mathcal{B})$ and a λ-interpretation λ for $\mathcal{G}(\mathcal{B})$ which is well-formed and maximally compatible with μ and such that:

$$\begin{cases} \mu(\mathrm{Z}\epsilon) = 0 \\ \mu(\mathrm{U}c_3) = \mu(\mathrm{U}c_5) = 1 \\ \mu(\emptyset c_2) = \mu(\emptyset c_4) = 1 \\ \mu(\emptyset c_1) = \mu(\emptyset c_2 c_3) = \mu(\emptyset c_4 c_5) \\ \mu(\emptyset c_1) = 2 \end{cases} \qquad \begin{cases} \lambda(\mathrm{Z}\epsilon) = \emptyset \\ \lambda(\mathrm{U}c_3) = \lambda(\emptyset c_4) = \{\!\{\, (l \mapsto a, b) \,\}\!\} \\ \lambda(\mathrm{U}c_5) = \lambda(\emptyset c_2) = \{\!\{\, (k \mapsto a, b) \,\}\!\} \\ \lambda(\emptyset c_1) = \lambda(\emptyset c_2 c_3) = \lambda(\emptyset c_4 c_5) \\ \lambda(\emptyset c_1) = \{\!\{\, (l \mapsto a, b), (k \mapsto a, b) \,\}\!\} \end{cases}$$

Therefore, the resource graph $\mathcal{G}(\mathcal{B})$ is structurally consistent with (μ, λ) and since $\mathcal{B}$ is also logically consistent with (μ, λ), we have shown that $\mathcal{B}$ is an open branch.
The last step of the countermodel construction process consists in deriving a partial monoid of heaps from the nodes in the resource graph $\mathcal{G}(\mathcal{B})[N, E]$. We proceed as follows: for all nodes Γx, we define a heap $h_x : Loc \to Val \times Val$ such that:

$$h_x(l) = (a, b) \text{ iff } (l \mapsto a, b) \in \lambda(\Gamma x).$$

Then, we define $\mathcal{M} = (H, \cdot, h_\epsilon)$ as the structure such that $H = \{\, h_x \mid \Gamma x \in N \,\}$, knowing that heap composition is given by the union of disjoint partial functions.

Lemma 5.3. *If $\mathcal{B}$ is a complete branch then $\mathcal{M} = (H, \cdot, h_\epsilon)$ is a* SLP*-model such that:*

a) if $\mathcal{B} \Vdash \mathrm{T}\,\phi : x$, then $\exists\, z \in \mathcal{B}_{\mathcal{L}}.\, \mathcal{B} \vdash z \sim x$ and $h_z \models \phi$;
b) if $\mathcal{B} \Vdash \mathrm{F}\,\phi : x$, then $\exists\, z \in \mathcal{B}_{\mathcal{L}}.\, \mathcal{B} \vdash z \sim x$ and $h_z \not\models \phi$.

Proof. Firstly, let us note that condition *(SC4)* of Definition 4.13 implies that h_x is a function for all nodes Γx and that h_ϵ is the empty heap e. Moreover condition *(SC3)* implies that if there exists z such that $\mathcal{B} \vdash z \sim xy$ then $h_x \cdot h_y = h_z$. The two properties a) and b) then follow by a lengthy induction on ϕ w.r.t. conditions of Definition 5.3.

Theorem 5.2 (completeness). *If the entailment $\psi \models \phi$ holds in* SLP*, then there exists a* TSLP*-proof of $\psi \models \phi$.*

Proof. Suppose that $\psi \models \phi$ has no TSLP-proof, then, there exists no sequence of tableaux $(\mathcal{T}_i)_{1 \leq i \leq n}$ such that $\mathcal{T}_n$ is closed. Therefore, any (fair) tableau construction procedure results in a tableau containing a complete branch $\mathcal{B}$ from which we can build the structure $\mathcal{M} = (H, \cdot, h_\epsilon)$. Since $\mathcal{B} \succ \mathrm{F}\ \psi \mathrel{-\!\!*} \phi : \epsilon$, Lemma 5.3 entails that $\mathcal{M}$ is a SLP-model such that $h_\epsilon \not\models \psi \mathrel{-\!\!*} \phi$, which contradicts the fact that $\psi \models \phi$ holds in SLP.

5.3 Termination

In this section we give arguments about the termination of the tableau method for SLP. Here we cannot use the termination arguments given for the BI logic [17]. For this logic the termination of the tableau method relies on the possibility of reusing the atomic labels introduced by the $\mathrel{-\!\!*}_R$ and $*_L$ rules, so that only a finite number of atomic labels can be introduced during the expansion of a tableau branch.

The fact that atomic labels may be reused in the tableau method is justified by the fact that BI is complete w.r.t. Beth resource semantics and that this completeness property is preserved when the semantics is restricted to *regular valuations*. A valuation is regular if for all formulas ϕ whenever there exists a world m such that $m \models \phi$ there also exists the smallest world c_ϕ (w.r.t. the underlying preordering on worlds) such that $c_\phi \models \phi$. Let us remark that such an argument cannot be used in the case of SL since there is no such thing as the smallest heap that satisfies a SL formula.

Figure 7 shows a tableau for which an infinite branch $\mathcal{B}$ can be constructed if one only relies on the expansion rules because $\mathcal{B}$ contains $\pi\alpha$ formulas in the scope of a $\pi\beta$ formula. In the beginning, when Step 2 is performed, the only admissible label for the expansion of the $\pi\beta$ formula $\mathrm{T}\ \top \mathrel{-\!\!*} (((k \mapsto c, d) \mathrel{-\!\!*} (l \mapsto a, b)) \to (l \mapsto a, b)) : c_1$ is ϵ since $c_1 \circ \epsilon = c_1$ and c_1 is obviously defined in the resource graph of the branch. However, when we reach Step 4, the new labels c_2 and c_1c_2 are introduced in the branch $\mathcal{B}$ by the $\pi\alpha$ formula $\mathrm{F}\ (k \mapsto c, d) \mathrel{-\!\!*} (l \mapsto a, b) : c_1$, which immediately enables a new possibility for the expansion of the previous $\pi\beta$ formula. Therefore, in Step 5, we make a new expansion of the $\pi\beta$ formula using the label c_2 since c_1c_2 is now defined in the resource graph of the branch. This leads in Step 6 to the introduction of a new instance of the $\pi\alpha$ formula of Step 4 which is now labelled with c_1c_2 instead of c_1. Step 7 then generates new labels c_3 and $c_1c_2c_3$, which by the *(Saturation)* condition of 3.6 also generates c_1c_3 and c_2c_3. Again, those new labels can be reused by the $\pi\beta$ formula and we enter an infinite expansion loop for the branch $\mathcal{B}$.

Let us now take a closer look at the infinite branch $\mathcal{B}$ using the specific notions of measures and λ-interpretations.

After Step 4, the introduction of the signed formula $\mathrm{T}\ (k \mapsto c, d) : c_2$ comes together with

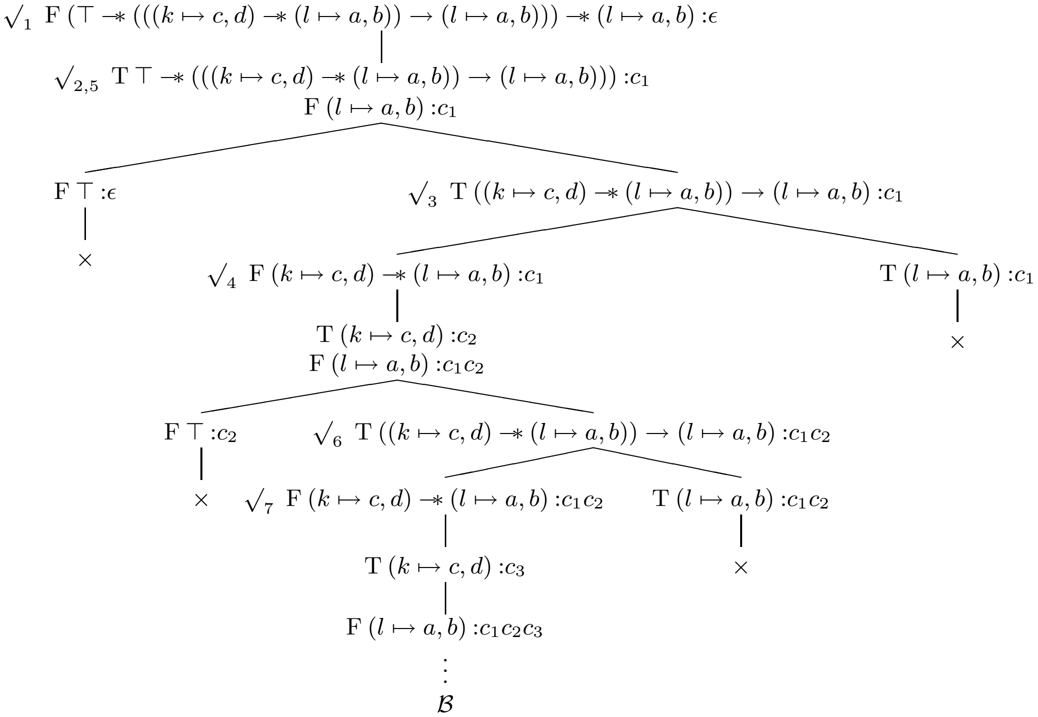


Figure7. Infinite Tableau for $(k \mapsto c, d) \multimap (l \mapsto a, b)) \to (l \mapsto a, b)) \models (l \mapsto a, b)$.

the assertion Uc_2 (not explicitly depicted in Figure 7) which implies that all measures μ on $\mathcal{G}(\mathcal{B})$ should be such that $\mu(c_2) = 1$. Moreover, we can also deduce that for all λ-interpretations λ, $\lambda(c_2) \supseteq \{\!\{\, (k \mapsto c, d) \,\}\!\}$. Similarly, after T $(k \mapsto c, d)$:c_3 and the assertion Uc_2 are introduced in Step 7, we deduce that for all measures μ, $\mu(c_3) = 1$ and for all λ-interpretations λ, $\lambda(c_3) \supseteq \{\!\{\, (k \mapsto c, d) \,\}\!\}$.
After Step 7, we know that for all measures μ and all λ-interpretations the following equations hold:

$$\mu(c_2) = \mu(c_3) = 1 \text{ and } \lambda(c_2c_3) \supseteq \lambda(c_2) \uplus \lambda(c_3) \supseteq \{\!\{\, (k \mapsto c, d), (k \mapsto c, d) \,\}\!\},$$

which contradicts both conditions *(SC1)* and *(SC2)* of Definition 4.12. Therefore, for all measures μ, there exists no λ-interpretation λ for $\mathcal{G}(\mathcal{B})$ which is well-formed and compatible with μ. Consequently, $\mathcal{B}$ cannot be structurally consistent and according to Definition 4.16, the tableau branch $\mathcal{B}$ is closed after Step 7.
Taking into account the information induced by the measures and λ-interpretation, we can prove that our tableau method always terminate. Let us remark that in order to keep a tableau branch $\mathcal{B}$ open, we need to have a measure μ on $\mathcal{G}(\mathcal{B})$ and a λ-interpretation λ for $\mathcal{G}(\mathcal{B})$ such that λ is well-formed and maximally compatible with μ. In the example of Figure 7, this would require

$$\mu(c_2) = \mu(c_3) = 1 \text{ and } \lambda(c_2) = \lambda(c_3) = \{\!\{\, (k \mapsto c, d) \,\}\!\}$$

by conditions *(SC1)* and *(SC2)* of Definition 4.12, which in turn implies that $c_2 \sim c_3$ holds in $\mathcal{B}$. Therefore, upto the equivalence $\sim$ induced by (μ, λ), c_2 and c_3 should semantically denote the same heap.

Theorem 5.3 (termination). TSLP *terminates for all* SLP *formulas.*

Proof. Let $\mathcal{T}$ be a tableau for a formula ϕ which contains an infinite open branch $\mathcal{B}$. Since $\mathcal{B}$ is open, we know from Definition 4.16 that there exists a measure μ and a λ-interpretation λ which is well-formed and maximally compatible with μ.
Let us consider the quotient $\mathcal{G}(\mathcal{B}_{/\sim})$ of the resource graph $\mathcal{G}(\mathcal{B})$ by the equivalence $\sim$ induced by (μ, λ). It is not difficult to see that there can only be finitely many classes in $N_{/\sim}$ because

1. the initial formula ϕ admits only a finite number of subformulas of the form $(l \mapsto a, b)$ with sign T and
2. if $\mathrm{T}\ (l \mapsto a, b) : x$ and $\mathrm{T}\ (l \mapsto a, b) : y$ both occur in $\mathcal{B}$ then $x \sim y$ holds in $\mathcal{B}$.

Given that a λ-interpretation only depends on signed formulas of the form $\mathrm{T}\ (l \mapsto a, b) : x$, that λ is well-formed and that we start with a finite number of signed subformulas $\mathrm{T}\ (l \mapsto a, b) : x$, the tableau calculus TSLP cannot introduce an infinite number of disjoint label compositions without contradicting condition *(SC1)* of Definition 4.12.

6 Extensions to Full SL

In this section we explain how our tableau calculus for SLP can be extended in order to take into account the introduction of variables, first order quantifiers and equality. Here, the points-to predicate $\mapsto$ associates a cell to a stack variable (as in $(x \mapsto a, b)$ where x points to the cell (a, b)) and such variables are in the scope of the first order quantifiers so that all the formulas we consider are closed. Moreover, we assume without loss of generality that all stack variables are bound to a unique quantifier. For example, in the formula

$$\phi = (\exists\, x \forall\, y \forall\, z.(x \mapsto y, z)) \to (\forall\, x \exists\, y.(x \mapsto y, y)),$$

the stack variables x and y are bound by two quantifiers (one on both side of the additive implication $\to$) and should be renamed appropriately before we can proceed with the notions presented in this section. Such a renaming results in the formula

$$\phi' = (\exists\, x_1 \forall\, y_1 \forall\, z.(x_1 \mapsto y_1, z)) \to (\forall\, x_2 \exists\, y_2.(x_2 \mapsto y_2, y_2)).$$

Since we use the letters x, y and z to denote stack variables in SL formulas we now use the letters u, v and w for arbitrary labels.

6.1 Eliminating First Order Quantifiers

Let us first note that we only have to deal with the existential quantifier since $\forall\, x.\varphi(x)$ is nothing but syntactic sugar for $\neg \exists\, x.\neg\varphi(x)$.
Given an entailment $\psi \models \phi$ between two formulas ψ and ϕ,

$$\begin{array}{cc}
\mathrm{T}\,\exists\, x.\varphi(x) :(s,u) & \mathrm{F}\,\exists\, x.\varphi(x) :(s,u) \\
| & | \\
\mathrm{T}\,\varphi(X) :([\, s \mid x \mapsto X\,], u)\ ^{(*)} & \mathrm{F}\,\varphi(t) :([\, s \mid x \mapsto t\,], u) \\
\\
\mathrm{F}\,\forall\, x.\varphi(x) :(s,u) & \mathrm{T}\,\forall\, x.\varphi(x) :(s,u) \\
| & | \\
\mathrm{F}\,\varphi(X) :([\, s \mid x \mapsto X\,], u)\ ^{(*)} & \mathrm{T}\,\varphi(t) :([\, s \mid x \mapsto t\,], u)
\end{array}$$

$^{(*)}$: X is a new parameter.

Figure8. Tableau Rules for the Quantifiers.

- $Val(\psi \models \phi)$ is the set $Val(\psi) \cup Val(\phi)$ of all the values occurring in ψ or ϕ;
- $Par(\psi \models \phi)$ is a countable set of symbols, called parameters, that do not occur in $Val(\psi \models \phi)$;
- $VP(\psi \models \phi) = Val(\psi \models \phi) \cup Par(\psi \models \phi)$.

In order to eliminate quantifiers, we use a standard technique that instantiates variables with values or parameters depending on the sign S of a signed formula $\mathrm{S}\,\exists\, x.\varphi(x) :u$ [14]. The corresponding expansion rules then looks as depicted in Figure 6.1.
Given a tableau branch $\mathcal{B}$ in a tableau for $\psi \models \phi$, $Par(\mathcal{B})\,(\subseteq Par(\psi \models \phi))$ denotes the set of the parameters occurring in $\mathcal{B}$ and we set

$$Val(\mathcal{B}) = Val(\psi \models \phi) \text{ and } VP(\mathcal{B}) = Val(\mathcal{B}) \cup Par(\mathcal{B}).$$

Applying a T$\exists$ rule in a tableau branch $\mathcal{B}$ then results in the introduction of a parameter $X \in Par(\psi \models \phi)$ that does not already occur in $Par(\mathcal{B})$. Moreover, the variable x is bound to this new parameter X, so that the current stack s is extended in a new stack $s' = [\, s \mid x \mapsto X\,]$. A key point is therefore that compared to the propositional case, our tableau calculus now incrementally builds a stack (via the T$\exists$ rule) for each tableau branch, initially starting with the empty stack.
Unlike F$\exists$, the T$\exists$ rule reuses either a parameter that already occurs in $Par(\mathcal{B})$, or a value already that already occurs in $Val(\mathcal{B})$. If neither a parameter, nor a value can be chosen because $VP(\mathcal{B})$ is empty, the F$\exists$ rule behaves exactly as the T$\exists$ rule w.r.t. parameter introduction (it generates a new parameter X). Following the standard terminology of first-order logic, T$\exists$ and F$\forall$ are called δ-rules, while F$\exists$ and T$\forall$ are called γ-rules. As usual, a γ-rule may be applied to a signed formula as many times as needed to try all the values and all the parameters occurring in a tableau branch
Let us note that the tableau rules for quantifier elimination only operate on the stack component s of a generalized label (s, u) and have therefore no impact on the resource graph of a tableau branch. Indeed, resource graphs capture the properties of the heap structure underlying a particular SL model so that heaps are considered as the actual resources (labels). Consequently, provided that we always take care of discarding the stack component s of a generalized label (s, u) (thus obtaining a proper label u of SLP), all the notions defined in Section 4 (measure, λ-interpretation, structural and logical consistency)

$$
\begin{array}{c}
\surd_1\ \mathrm{F}\,(\exists x \forall y.(x \mapsto y, b)) \mathrel{-\!\!*} (\exists z.(z \mapsto a, b)) :([\,], \epsilon) \\
| \\
\surd_2\ \mathrm{T}\,\exists x \forall y.(x \mapsto y, b) :([\,], c_1) \\
\surd_3\ \mathrm{F}\,\exists z.(z \mapsto a, b)) :([\,], c_1) \\
| \\
\surd_{4,5}\ \mathrm{T}\,\forall y.(X \mapsto y, b) :([\,x \mapsto X\,], c_1) \\
| \\
\mathrm{F}\,(X \mapsto a, b) :([\,z \mapsto X\,], c_1) \\
| \\
\mathrm{T}\,(X \mapsto b, b) :([\,x \mapsto X, y \mapsto b\,], c_1) \\
| \\
\mathrm{T}\,(X \mapsto a, b) :([\,x \mapsto X, y \mapsto a\,], c_1) \\
| \\
\mathcal{B} \\
| \\
\times
\end{array}
$$

Figure9. Tableau for $\exists x \forall y.(x \mapsto y, b) \models \exists z.(z \mapsto a, b)$.

straightforwardly lift from SLP to its extension with first order quantifiers without any further modification.

Figure 6.1 illustrates how quantifiers are eliminated in the tableau construction process. Note that we use the capital letters X, Y and Z to denote the parameters introduced by the elimination of the variables x, y and z. The interesting steps are Step 4 and Step 5. In Step 4, we need to apply the $\mathrm{T}\forall$ on the signed formula

$$\mathrm{T}\,\forall y.(X \mapsto y, b) :([\,x \mapsto X\,], c_1),$$

which requires us to choose a value for y. Since $VP(\mathcal{B}) = \{\, a, b, X \,\}$, we have three possibilities and we choose to first try to instantiate y with b, which results in the introduction of the signed formula

$$\mathrm{T}\,(X \mapsto b, b) :([\,x \mapsto X, y \mapsto b\,], c_1).$$

Obviously, this does not help us closing the tableau branch because the only points-to predicate with sign F occurs in the signed formula $\mathrm{F}\,(X \mapsto a, b) :([\,z \mapsto X\,], c_1)$ introduced in Step 3. However, as $\mathrm{T}\forall$ is a γ-rule, we can perform a second expansion of the signed formula of Step 4 in order to try another choice, thus leading to Step 5 where y is instantiated with a. Such a choice then allows us to close the tableau branch $\mathcal{B}$ by condition *(LC1)* of Definition 4.15 since, discarding the stack component of the generalized labels, we have $\mathrm{F}\,(X \mapsto a, b) :c_1 \in \mathcal{B}$, $\mathrm{T}\,(X \mapsto a, b) :c_1 \in \mathcal{B}$ and $\mathcal{B} \vdash c_1 \sim c_1$.

Another example is given in Figure 10 where we end up with a tableau that contains two branches. The second one is closed because for all measures, $\mathrm{T} \perp :c_1 \in \mathcal{B}_2$ implies that $\mathcal{B}_2$ is not logically consistent. The first branch is also closed because its resource graph $\mathcal{G}(\mathcal{B}_1)$ is not measurable (otherwise $\mathrm{ZU}c_2$ would imply $\mu(c_2) = 0$ and $\mu(c_2) = 1$) and thus cannot be structurally consistent.

$$
\begin{array}{c}
\surd_1\ \mathrm{F}\,((\exists x.\mathrm{I} \rightarrow \bot) \rightarrow \bot) \mathrel{-\!\!*} ((\exists y.(y \mapsto a, b) * \top) \rightarrow \bot) :([\,], \epsilon) \\
| \\
\surd_5\ \mathrm{T}\,(\exists x.\mathrm{I} \rightarrow \bot) \rightarrow \bot :([\,], c_1) \\
\surd_2\ \mathrm{F}\,(\exists y.(y \mapsto a, b) * \top) \rightarrow \bot :([\,], c_1) \\
| \\
\surd_3\ \mathrm{T}\,\exists y.(y \mapsto a, b) * \top :([\,], c_1) \\
\mathrm{F}\,\bot :([\,], c_1) \\
| \\
\surd_4\ \mathrm{T}\,(Y \mapsto a, b) * \top :([\,y \mapsto Y\,], c_1) \\
| \\
\boxed{as_1 : c_2c_3 \diamond c_1} \\
| \\
\mathrm{T}\,(Y \mapsto a, b) :([\,y \mapsto Y\,], c_2) \\
\mathrm{T}\,\top :([\,y \mapsto Y\,], c_3) \\
\begin{array}{cc}
\surd_6\ \mathrm{F}\,\exists x.\mathrm{I} \rightarrow \bot :([\,], c_1) & \mathrm{T}\,\bot :([\,], c_1) \\
| & | \\
\surd_7\ \mathrm{F}\,\mathrm{I} \rightarrow \bot :([\,x \mapsto a\,], c_1) & \mathcal{B}_2 \\
| & | \\
\mathrm{T}\,\mathrm{I} :([\,x \mapsto a\,], c_1) & \times \\
\mathrm{F}\,\bot :([\,x \mapsto a\,], c_1) & \\
| & \\
\mathcal{B}_1 & \\
| & \\
\times &
\end{array}
\end{array}
$$

$$
\mathcal{G}(\mathcal{B}_1) : \quad \boxed{\mathrm{Z}c_3} \quad \boxed{\mathrm{Z}\epsilon} \quad \boxed{\mathrm{Z}c_1} - \boxed{\mathrm{Z}c_2c_3} \quad \boxed{\mathrm{ZU}c_2}
$$

Figure10. Tableau for $((\exists x.\mathrm{I} \rightarrow \bot) \rightarrow \bot) \models ((\exists y.(y \mapsto a, b) * \top) \rightarrow \bot)$.

$$
\begin{array}{c}
\surd_1\ \mathrm{F}\,(\exists x_1y_1z_1.((x_1 \mapsto a, y_1) * (z_1 \mapsto c, d)) \wedge (y_1 = z_1)) \mathrel{-\!\!*} (\exists x_2y_2.(x_2 \mapsto a, y_2) * (y_2 \mapsto c, d)) :([\,], \epsilon) \\
| \\
\surd_{2,3,4}\ \mathrm{T}\,\exists x_1y_1z_1.((x_1 \mapsto a, y_1) * (z_1 \mapsto c, d)) \wedge (y_1 = z_1) :([\,], c_1) \\
\surd_{7,8}\ \mathrm{F}\,\exists x_2y_2.(x_2 \mapsto a, y_2) * (y_2 \mapsto c, d) :([\,], c_1) \\
| \\
\surd_5\ \mathrm{T}\,((X_1 \mapsto a, Y_1) * (Z_1 \mapsto c, d)) \wedge (Y_1 = Z_1) :([\,x_1 \mapsto X_1, y_1 \mapsto Y_1, z_1 \mapsto Z_1\,], c_1) \\
| \\
\surd_6\ \mathrm{T}\,(X_1 \mapsto a, Y_1) * (Z_1 \mapsto c, d) :([\,x_1 \mapsto X_1, y_1 \mapsto Y_1, z_1 \mapsto Z_1\,], c_1) \\
\mathrm{T}\,(Y_1 = Z_1) :([\,x_1 \mapsto X_1, y_1 \mapsto Y_1, z_1 \mapsto Z_1\,], c_1) \\
| \\
\boxed{as_1 : c_2c_3 \diamond c_1} \\
| \\
\mathrm{T}\,(X_1 \mapsto a, Y_1) :([\,x_1 \mapsto X_1, y_1 \mapsto Y_1, z_1 \mapsto Z_1\,], c_2) \\
\mathrm{T}\,(Z_1 \mapsto c, d) :([\,x_1 \mapsto X_1, y_1 \mapsto Y_1, z_1 \mapsto Z_1\,], c_3) \\
| \\
\surd_7\ \mathrm{F}\,(X_1 \mapsto a, Y_1) * (Y_1 \mapsto c, d) :([\,x_2 \mapsto X_1, y_2 \mapsto Y_1\,], c_1) \\
| \\
\boxed{rq_1 : c_2c_3 \diamond c_1} \\
\begin{array}{cc}
\mathrm{F}\,(X_1 \mapsto a, Y_1) :([\,x_2 \mapsto X_1, y_2 \mapsto Y_1\,], c_2) & \mathrm{F}\,(Y_1 \mapsto c, d) :([\,x_2 \mapsto X_1, y_2 \mapsto Y_1\,], c_3) \\
| & | \\
\mathcal{B}_1 & \mathcal{B}_2 \\
| & | \\
\times & \times
\end{array}
\end{array}
$$

Figure11. Tableau for $\exists x_1y_1z_1.((x_1 \mapsto a, y_1) * (z_1 \mapsto c, d)) \wedge (y_1 = z_1) \models \exists x_2y_2.(x_2 \mapsto a, y_2) * (y_2 \mapsto c, d)$.

6.2 Dealing with Equality

Rather than adding new expansion rules to the tableau calculus, we deal with the equality predicate $=$ through an extension of the notion of logical consistency. More precisely, given a tableau branch $\mathcal{B}$, the set $\mathcal{B}^{=}$ of all the signed formulas of the form $\mathrm{T}\, t_1 = t_2 :(s,u)$ occurring in $\mathcal{B}$ induces a renaming ρ such that:

- if t_1, t_2 are parameters then $t_1/t_2 \in \rho$ (t_1 is renamed as t_2);
- if t_1 is a parameter and t_2 is a constant then $t_1/t_2 \in \rho$;
- if t_1 is a constant and t_2 is a parameter then $t_2/t_1 \in \rho$;

The tableau branch $\mathcal{B}$ is now considered logically consistent iff

1. $\mathcal{B}\rho$ is logically consistent according to Definition 4.15 and,
2. for all signed formulas $\mathrm{T}\, t_1 = t_2 :(s,u) \in \mathcal{B}^{=}$, considering t_1 and t_2 as terms in a (purely syntactic) term-unification process,
   - either $\mathrm{S} = \mathrm{T}$ and $t_1\rho$ is syntactically equal to (unifies with) $t_2\rho$ *(LC5)*,
   - or $\mathrm{S} = \mathrm{F}$ and $t_1\rho$ is not syntactically equal to (does not unify with) $t_2\rho$ *(LC6)*.

Figure 11 shows how the equality predicate is handled by the tableau calculus. We eventually get two tableau branches $\mathcal{B}_1$ and $\mathcal{B}_2$ sharing the following resource graph:

$$\mathcal{G}(\mathcal{B}_1) = \mathcal{G}(\mathcal{B}_2): \quad \begin{array}{cccc} & & & \boxed{\mathrm{U}c_3} \\ \boxed{\mathrm{Z}\epsilon} & \boxed{\emptyset c_1} & \text{———} & \boxed{\emptyset c_2 c_3} \\ & & & \boxed{\mathrm{U}c_2} \end{array}$$

Moreover, since $\mathcal{B}_1$ and $\mathcal{B}_2$ contain the same signed formula

$$\mathrm{T}\,(Y_1 = Z_1) :([\, x_1 \mapsto X_1, y_1 \mapsto Y_1, z_1 \mapsto Z_1\,], c_1)$$

and since Y_1, Z_1 are parameters, the renamings ρ_1 and ρ_2 respectively induced by $\mathcal{B}_1$ and $\mathcal{B}_2$ are the same and such that:

$$\rho_1 = \rho_2 = \{\, Y_1/Z_1 \,\}.$$

Therefore, $\mathcal{B}_1\rho$ is not logically consistent by condition *(LC1)* of Definition 4.15 since for $\mathcal{B}_1$, once the stack component of the generalized labels are discarded, we have

$$\mathrm{T}\,(Y_1 \mapsto c, d) :c_2 \in \mathcal{B}_1,\ \mathrm{F}\,(Y_1 \mapsto c, d) :c_2 \in \mathcal{B}_1 \text{ and } \mathcal{B}_1 \vdash c_2 \sim c_2.$$

Similarly for $\mathcal{B}_2\rho$, once Y_1 is renamed as X_1, we have.

$$\mathrm{T}\,(Z_1 \mapsto c, d) :c_3 \in \mathcal{B}_2,\ \mathrm{F}\,(Z_1 \mapsto c, d) :c_3 \in \mathcal{B}_2 \text{ and } \mathcal{B}_2 \vdash c_3 \sim c_3.$$

It then follows from Definition 4.16 that $\mathcal{B}_1$ and $\mathcal{B}_2$ are closed.

6.3 Decidability Issues

It is known that full SL is not decidable [9]. This undecidability result is obtained through Trakhtenbrot's theorem by showing that any first order formula ϕ with a single binary relation $R(x, y)$ can be faithfully encoded in SL using points-to predicates. The encoding $rd(\phi)$ proceeds as follows (with $(x \hookrightarrow a, b)$ being syntactic sugar for $(x \mapsto a, b) * \top$):

$$\begin{array}{ll} rd(\phi) & = (\exists x.\,(x \hookrightarrow nil, nil)) \to prd(\phi) \\ prd(R(x,y)) & = (\exists z.\,(z \hookrightarrow x, y)) \wedge (x \hookrightarrow nil, nil) \wedge (y \hookrightarrow nil, nil) \\ prd(\phi \to \psi) & = prd(\phi) \to prd(\psi) \\ prd(\bot) & = \bot \\ prd(x = y) & = (x = y) \wedge (x \hookrightarrow nil, nil) \\ prd(\exists x.\,\phi) & = \exists x.\,((x \hookrightarrow nil, nil) \wedge prd(\phi)) \end{array}$$

Intuitively, the translation encodes the relation and its universe by heap cells and the guard $\exists x.\,(x \mapsto nil, nil)$ in the definition of rd ensures that the universe of a finite structure is not empty (see [9] for further details). Even if SL is undecidable, some recent works aim at finding particular fragments for which decidability can be obtained. For example, one such fragment is given in [1].

A fragment for which undecidability has been established very recently [7] is the fragment where $\mapsto$ predicates are restricted so that the cells contain only one location. More precisely, let us call SL_1 this variant of SL in which the points-to are of the form $x \mapsto y$, where y can only be bound to a location (nil being considered as special location).

Let us remark that the undecidability of the SL_1 fragment cannot be deduced from the proof given in [9]. Indeed, the encoding $rd(\phi)$ no longer works with the restricted form of the points-to predicate since $prd(R(x, y))$ requires that the heap cells should be pairs of values. However, this fragment seems very interesting in the context of shape analysis where one only cares about the shape of a structure and not about the data it contains. For example, a property such as the circularity of a list does not depend on the data contained in the nodes of the list.

Let us analyze this decidability problem from the proof-search point of view with our new calculus. It is clear that the undecidability of full SL entails that our tableau system TSL does not always terminate for all formulas. Let us illustrate this point with the formula $\phi = (\forall x \exists y.\,(x \mapsto y)) \mathrel{-\!\!*} (\exists u \forall v.\,(u \mapsto v))$ for which a tableau is given on the left-hand-side of Figure 12. On the right-hand-side of Figure 12 we give a tableau for the first-order formula $(\forall x \exists y.\,P(x, y)) \to (\exists u \forall v.\,P(u, v))$, where points-to predicates are replaced by an uninterpreted predicate P in order to show the difference between TSL and the standard tableau system for first-order logic. It is clear that if one relies only on the expansion rules, then both tableaux grow infinitely because there are signed formulas of type δ in the scope of signed formulas of type γ. Since δ signed formulas endlessly generate fresh locations (parameters for the second tableau) to be used with the γ signed formulas, the tableau method cannot terminate.

However, after Step 4, the tableau branch $\mathcal{B}_1$ contains a signed formula $\mathrm{T}\,(k \mapsto l_1) : c_1$ and thus also an assertion $\mathrm{U}c_1$ (not explicitly shown), which necessarily implies $\mu(\mathrm{U}c_1) = 1$ for all measures μ on the corresponding resource graph $\mathcal{G}(\mathcal{B}_1)$. Therefore, as soon as Step 8 is

$$\begin{array}{c}
\surd_1\ \mathrm{F}\,(\forall x\exists y.\,(x\mapsto y)) \mathrel{-\!\!*} (\exists u\forall v.\,(u\mapsto v)) :\epsilon\\
|\\
\surd_{2,6}\ \mathrm{T}\,\forall x\exists y.\,(x\mapsto y) :c_1\\
\surd_{3,7}\ \mathrm{F}\,\exists u\forall v.\,(u\mapsto v) :c_1\\
|\\
\surd_4\ \mathrm{T}\,\exists y.\,(k\mapsto y) :c_1\\
|\\
\surd_5\ \mathrm{F}\,\forall v.\,(k\mapsto v) :c_1\\
|\\
\mathrm{T}\,(k\mapsto l_1) :c_1\\
|\\
\mathrm{F}\,(k\mapsto m_1) :c_1\\
|\\
\surd_8\ \mathrm{T}\,\exists y.\,(l_1\mapsto y) :c_1\\
\mathrm{T}\,\exists y.\,(m_1\mapsto y) :c_1\\
|\\
\surd_9\ \mathrm{F}\,\forall v.\,(l_1\mapsto v) :c_1\\
\mathrm{F}\,\forall v.\,(m_1\mapsto v) :c_1\\
|\\
\mathrm{T}\,(l_1\mapsto l_2) :c_1\\
|\\
\mathrm{F}\,(l_1\mapsto m_2) :c_1\\
|\\
\vdots\\
|\\
\mathcal{B}_1
\end{array}
\qquad\qquad
\begin{array}{c}
\surd_1\ \mathrm{F}(\forall x\exists y.\,P(x,y)) \to (\exists u\forall v.\,P(u,v))\\
|\\
\surd_{2,6}\ \mathrm{T}\forall x\exists y.\,P(x,y)\\
\surd_{3,7}\ \mathrm{F}\exists u\forall v.\,P(u,v)\\
|\\
\surd_4\ \mathrm{T}\exists y.\,P(k,y)\\
|\\
\surd_5\ \mathrm{F}\forall v.\,P(k,v)\\
|\\
\mathrm{T}P(k,l_1)\\
|\\
\mathrm{F}P(k,m_1)\\
|\\
\surd_8\ \mathrm{T}\exists y.\,P(l_1,y)\\
\mathrm{T}\exists y.\,P(m_1,y)\\
|\\
\surd_9\ \mathrm{F}\forall v.\,P(l_1,v)\\
\mathrm{F}\forall v.\,P(m_1,v)\\
|\\
\mathrm{T}P(l_1,l_2)\\
|\\
\mathrm{F}P(l_1,m_2)\\
|\\
\vdots\\
|\\
\mathcal{B}_2
\end{array}$$

Figure12. Tableau for $(\forall x\exists y.\,(x\mapsto y)) \to (\exists u\forall v.\,(u\mapsto v))$

performed, the branch $\mathcal{B}_1$ becomes structurally inconsistent since, on one hand, we have $\mu(\mathrm{U}c_1) = 1$ and, on the other hand, the introduction of $\mathrm{T}\,(l_1\mapsto l_2) :c_1$ requires that all λ-interpretations λ for $\mathcal{G}(\mathcal{B}_1)$ should be such that $\{\!\{\, k, l1 \,\}\!\} \subseteq \lambda^L(\mathrm{U}c_1)$, which obviously contradicts condition *(SC2)* of Definition 4.12 since $|\lambda^L(\mathrm{U}c_1)| \not\leq \mu(\mathrm{U}c_1)$.

The previous example clearly shows that, in the case of TSL, one can sometimes make use of the additional information conveyed by the notions of measures and λ-interpretations to devise termination criteria for some specific fragments. For example, the analysis of the previous example leads to the following theorem.

Theorem 6.1. *Let ϕ be a formula of SL_1. If for all variables X occurring in ϕ on the left-hand-side of a $\mapsto$ predicate ϕ is such that X is not bound to a quantifier which is the principal connective of a subformula (of ϕ) of type δ, then the validity of ϕ is decidable.*

Proof. First, let us remark that a TSL tableau can only admit a finite number of label-classes modulo the equivalence $\sim$ generated by a given measure μ. Moreover, the tableau rules for the quantifiers do not introduce new labels but only introduce new locations. Consequently, for a tableau branch $\mathcal{B}$ to grow infinitely without being closed, there must be some label x for which an infinite number of locations is created. But then, since locations can only be reused through formulas of the form $\mathrm{T}\,\forall X.\,(X\mapsto Y) :x$ the resource graph $\mathcal{G}(\mathcal{B})$ associated to $\mathcal{B}$ cannot be structurally consistent because for all measures μ,

$\mathrm{T}\,(X \mapsto Y) : x$ implies $\mu(\Gamma x) = 1$, which contradicts condition *(SC2)* of Definition 4.12 and the fact that x admits infinitely many locations.

In further work, we aim at studying the (un)decidability of other fragments of SL from our tableau-based proof-search approach.

7 Conclusions and Perspectives

Separation Logic provides some verification formalisms for pointer programs and allows to express properties about data structures with shared mutable state [18,25]. Here we have studied proof-theoretic foundations for this logic and then proposed a new characterization of validity in SL that is based on a theorem-proving approach from a particular tableau calculus, with labels and constraints, that builds resource graphs from which countermodels can be extracted.

Until now the methods and tools to deal with mechanized program verification in SL were based on a model-checking approach [2,26]. Further work will be devoted to the implementation of our calculus and then its extensions, for instance with inductive definitions, in order to provide a new tool for proving assertions in SL. Moreover a deeper study of the possibilities to mix the existing model-checking approach with our based-on tableaux approach could be fruitful in order to improve automatic assertion verification. As we can generate countermodels in case of non-validity we could expect to be able to analyze a failure case in order to refine and correct the initial specification to deal with. In addition, a key feature of this work is that provability is captured through two distinct notions: the structural consistency and the logical consistency. The first one ensures that a resource graph actually denotes a model in the class of Separation Logic models while the second one ensures that a formula can be falsified in some model. Then we can distinguish whether a formula is valid for intrinsic logical reasons or because the conditions required for a given structure to be a model in the class of SL models are too restrictive to allow the existence of a model. This central point also makes the verification method more modular as one could change the conditions of structural consistency in order to match other classes of resource models. For example, one could consider resource models for which the composition is not an aggregation.

Therefore, further works will also concern the development of the same methodology, based on the capture of the semantics by defining particular resource graphs from which validity is studied, to other logics related to SL. We can first mention the affine variant of SL [18] with intuitionistic additives that allows to also prove interesting properties about sharing, but also some spatial logics that can be seen as extensions of BBI like Ambient Logic [12], labelled tree model [11] or Context Logic [8]. As only model-checking methods have been developed for these logics, our present contributions could be seen as a first step towards the design of theorem-proving methods and tools in the context of program verification. Moreover some comparisons with our recent work on a separation logic for resource trees [4] from both specification and theorem-proving view points could be helpful to refine our results. Moreover, comparisons with existing works on theorem-proving dedicated to pointer programs [19,21,24], even they dont deal with the same logic could be also fruitful for some refinements.

References

1. J. Berdine, C. Calcagno, and P. O'Hearn. A Decidable Fragment of Separation Logic. In *24th Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2004, LNCS 3328*, pages 97–109, December 2004. Chennai, India.
2. J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO 2005, LNCS 4111*, pages 115–137, 2005.
3. J. Berdine, C. Calcagno, and P. O'Hearn. Symbolic execution with separation logic. In *APLAS 2005, LNCS 3780*, pages 52–68, 2005.
4. N. Biri and D. Galmiche. Models and separation logics for resource trees. *Journal of Logic and Computation*, 17(4):687–726, 2007.
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. Rémi Brochenin, Stéphane Demri, and Étienne Lozes. Reasoning about sequences of memory states. In Sergei N. Artemov and Anil Nerode, editors, *Proceedings of the Symposium on Logical Foundations of Computer Science (LFCS'07)*, volume 4514 of *Lecture Notes in Computer Science*, pages 100–114, New-York, NY, USA, June 2007. Springer.
7. Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. In Michael Kaminski and Simone Martini, editors, *Proceedings of the 16th Annual EACSL Conference on Computer Science Logic (CSL'08)*, Lecture Notes in Computer Science, Bertinoro, Italy, September 2008. Springer. To appear.
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. TQL: a query language for semi-structured data based on the ambient logic. *Mathematical Structures in Computer 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. D. Distefano, P. O'Hearn, and H. Yang. A local shape analysis based on separation logic. In *TACAS 2006, LNCS 3920*, pages 287–302, 2006.
14. M. Fitting. *First-Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer Verlag, 1990.
15. 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.
16. D. Galmiche and D. Méry. Resource graphs and countermodels in resource logics. *Electronic Notes in Theoretical Computer Science*, 125(3):117–135, 2005.
17. D. Galmiche, D. Méry, and D. Pym. The semantics of BI and Resource Tableaux. *Math. Struct. in Comp. Science*, 15(6):1033–1088, 2005.
18. 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.
19. J. Jenson, M. Jorgensen, N. Klarkund, and M. Schwartzback. Automatic verification of pointer programs using monadic second-order logic. In *Conf. on Programming Language Design and Implementation, PLDI'97*, pages 225–236, 1997.
20. Étienne Lozes. Elimination of spatial connectives in static spatial logics. *Theoretical Computer Science*, 330(3):475–499, February 2005.
21. F. Mehta and T. Nipkow. Proving pointer programs in higher-order logic. In *Int. Conference on Automated Deduction, CADE-19, LNCS 2741*, pages 121–135, Miami, USA, July 2003.

22. 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.
23. P.W. O'Hearn and D. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.
24. S. Ranise and D. Deharbe. Applying light-weight theorem proving to debugging and verifying pointer programs. *Electronic Notes in Theoretical Computer Science*, 86(1), 2003.
25. 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.
26. T. Weber. Towards mechanized program verification with separation logic. In *Int. Workshop on Computer Science Logic, CSL'04, LNCS 3210*, pages 250–264, Wroclaw, Poland, September 2004.