Decision Procedures for Region Logic

Stan Rosenberg[⋆1], Anindya Banerjee[⋆⋆2], and David A. Naumann[⋆⋆⋆1]

[1] Stevens Institute of Technology, Hoboken NJ 07030, USA
[2] IMDEA Software Institute, Madrid, Spain

Abstract. Region logic is Hoare logic for object-based programs. It features local reasoning with frame conditions expressed in terms of sets of heap locations. This paper studies tableau-based decision procedures for RL, the quantifier-free fragment of the assertion language. This fragment combines sets and (functional) images with the theories of arrays and partial orders. The procedures are of practical interest because they can be integrated efficiently into the satisfiability modulo theories (SMT) framework. We provide a semi-decision procedure for RL and its implementation as a theory plugin inside the SMT solver Z3. We also provide a decision procedure for an expressive fragment of RL termed restricted-RL. We prove that deciding satisfiability of restricted-RL formulas is NP-complete. Both procedures are proven sound and complete. Preliminary performance results indicate that the semi-decision procedure has the potential toscale to large input formulas.

1 Introduction

Frame conditions are an important part of procedure specifications. For procedures acting on shared mutable objects, frame conditions must designate the set of existing heap locations that may be updated —the footprint, in the terminology of separation logic. Following the lead of Kassios [12], the authors have developed a variant of Hoare logic, dubbed *region logic*, to explore the use of ghost state to express frame conditions in terms of explicit location sets [2]. We seek perspicuous specifications and effective local reasoning in automated verification based on SMT provers, for programs at the Java level of abstraction where heap locations are not integer addresses but rather are designated like $p.f$ where p is an object reference and f a field name. In region logic, frame conditions are designated in terms of *image* expressions[3] like $G`f$ where $G$ is a set of references (a *region*) and $G`f$ is the set of f fields of objects in G. Verification conditions typically involve operations on sets and predicates like containment and disjointedness.

Object sets are ubiquitous in functional specifications for object based programs (e.g., [28,19]). We are particularly interested in images, owing to their use in frame conditions. For example, in a state where $G_1 \# G_2$ holds (i.e., the regions are disjoint),

⋆ Partially supported by US NSF award CRI-0708330.
⋆⋆ Partially supported by CM Project S2009TIC-1465 Prometidos, MICINN Project TIN2009-14599-C03-02 Desafios, EU NoE Project 256980 Nessos.
⋆⋆⋆ Partially supported by US NSF awards CRI-0708330, CCF-0915611.

[3] A more conventional notation for images is $f[G]$, but this collides with array notation.

a procedure that writes G_1`$f$ does not interfere with a formula that only depends on $f$ fields of objects in $G_2$. Images are also useful to express closure conditions: if $p \in G$ and $G$`$f \subseteq G$ then G contains the objects reachable from p via f. Using loop invariants that entail reachability properties but are expressed in pure first order logic, our prototype verifier performs well both for verifying data structure implementations and for local reasoning about data structure clients [22,25]. In addition to closure and disjointedness constraints, the assertion language of region logic features reference equality, points-to assertions, and type/subtype constraints (for Java's class types).

Previous experiments with automation of region logic [1,22] relied on axiomatization of region assertions, wherein regions are represented by $\mathsf{ref} \to \mathsf{bool}$ functions and the semantics is encoded by axioms using quantified formulas. However, automated reasoning about quantifiers is necessarily incomplete and typically ad-hoc. State-of-the-art SMT solvers perform E-matching [7] in order to limit the number of quantifier instantiations. Essentially, quantified formulas are annotated with syntactic patterns or triggers; typically the user supplies these annotations, otherwise default heuristics are used. Finding "good" patterns can drastically improve the performance of certain benchmarks. However, not all quantified formulas lend themselves to useful patterns. Furthermore, SMT solvers are typically not *refutationally-complete*; e.g., if a region assertion happens to be invalid, then its encoding using quantifiers may yield `UNKNOWN` which means that the solver did not find an unsatisfiable conjunction, although one must exist by Compactness of first-order logic.

Goal. Our ultimate goal in this work is to obtain an efficient decision procedure for the quantifier-free fragment of the region assertion language. Implicit in "efficient" is the requirement to integrate well within the SMT framework; i.e., decision procedures for region assertions must perform reasoning modulo theories such as partial order (for class types) and integers that arise in program verification conditions. In particular we want to decide verification conditions involving region assertions and heap updates. The latter requires reasoning modulo the theory of arrays.

Approach and contributions. There has been great progress in automated reasoning about sets and related theories, notably [26,24], but as we discuss in Sect. 6 prior work does not fully reach our goal.

Our approach is inspired by a tableau-based decision procedure for a simple language of sets of elements [27]. In that procedure, reasoning about sets is performed by tableau rules while reasoning about the elements of sets can be done entirely by an SMT solver.

We formalize a quantifier-free first-order language RL which is sufficiently expressive to accomodate the quantifier-free fragment of the region assertion language. We formalize a set of tableau-based rules collectively referred to as the RL-tableau calculus. Applying the rules has the effect of deriving a refutation proof in case the given formula is valid. If the formula is invalid, then a tableau obtained by saturating (i.e., exhaustively applying the rules) denotes a counterexample. RL-tableau rules contain only syntactic conditions, namely (subterm) occurence checks in their premises, so the rules are simple to implement. To check for saturation it suffices to ensure that every possible distinct rule instance has been applied.

We prove that the RL-tableau calculus is refutationally-complete. In general, for some invalid RL-formulas, the rules may be non-terminating. Thus, what we obtain is a semi-decision procedure for RL. We have implemented this procedure as a theory plugin inside Z3 [6]. Preliminary performance results are encouraging.

We conjecture that RL has a high complexity. If we consider only finite interpretations of RL, we can show that the resulting theory is NEXPTIME-hard (see [21]). This leads us to investigate a syntactic restriction of RL, called restricted-RL. The restriction amounts to disallowing assertions of the form $H \subseteq G\text{`}f$ but allowing assertions of the form $G\text{`}f \subseteq H$. We give a tableau calculus for restricted-RL and show that it provides a nondeterministic polynomial time decision procedure. We also prove that deciding conjunctions of restricted-RL literals is an NP-complete problem. Restricted-RL is sufficiently expressive to capture what we have found to be the most useful idioms of region logic, such as disjointedness and closure constraints.

Our main contributions can be summarized as follows.

- sound and complete tableau calculus for a theory RL which includes regions, reference subtyping, type-respecting functional images, arrays, etc.
- implementation of semi-decision procedure for RL as a theory plugin in Z3
- encouraging experimental results from synthetic benchmarks
- sound and complete calculus for an expressive fragment, restricted-RL
- NP-completeness of restricted-RL-tableau calculus

Full proofs and further details can be found in Rosenberg's dissertation [21].

2 Preliminaries

Throughout, we work with quantifier-free, many-sorted first-order logic with equality. We tacitly assume that each theory has the equality symbol always interpreted in the standard way. There are countably many variables of each of a theory's sorts.

Syntax. The language RL is boolean formulas over the signature Σ_{RL} defined by:

- sorts: $\mathsf{rgn}, \mathsf{ref}, \mathsf{arr}, \mathsf{fname}, \mathsf{cname}$
- constants:
  - **null**, of sort ref (*un-allocated reference*), **emp**, of sort rgn (*empty region*)
  - **alloc**, of sort rgn (*universal region*)
- function symbols:
  - $\cup, \cap, -$, of sort $\mathsf{rgn} \times \mathsf{rgn} \to \mathsf{rgn}$ (*union, intersection, difference*)
  - $\{\cdot\}$, of sort $\mathsf{ref} \to \mathsf{rgn}$ (*quasi singleton*)
  - img, of sort $\mathsf{arr} \times \mathsf{rgn} \times \mathsf{fname} \to \mathsf{rgn}$ (*image*)
  - read, of sort $\mathsf{arr} \times \mathsf{ref} \times \mathsf{fname} \to \mathsf{ref}$ (*field read*)
  - write, of sort $\mathsf{arr} \times \mathsf{ref} \times \mathsf{fname} \times \mathsf{ref} \to \mathsf{arr}$ (*field write*)
  - type, of sort $\mathsf{ref} \to \mathsf{cname}$ (*type of reference*)
  - dtype, of sort $\mathsf{fname} \to \mathsf{cname}$ (*enclosing type of field*)
- predicate symbols:
  - $\in$, of sort $\mathsf{ref} \times \mathsf{rgn}$ (*membership*), $\leq$, of sort $\mathsf{cname} \times \mathsf{cname}$ (*subtype*)

We use metavariables r, s, t for rgn-terms, u, v, w for ref-terms, h for arr-terms, f, g for fname-terms, K for cname-terms.

In region logic the heap is implicit: expressions like $x.f$ and $G`f$ are interpreted with respect to a program state. In that setting, an update to the heap, e.g., $x.f := y$, yields a new program state. We aim to decide verification conditions containing region assertions, so we represent heap updates explicitly. We use two-dimensional arrays to encode the heap. This representation is particularly useful for encoding frame conditions [4] and is used in `Verl` [25]. The heap is made explicit in *field read* and *image* expressions, $\mathsf{read}(h, u, f)$ for $u.f$ and $\mathsf{img}(h, r, f)$ for $r`f$. An update is encoded by a *field write* expression, e.g., $\mathsf{write}(h, u, f, v)$ for $u.f := v$. Note that function symbols are interpreted by total functions, yet field accesses are not defined everywhere.

To encode definedness of field accesses, we use type to encode the *runtime* class type of a reference, $\leq$ to encode subtyping (i.e., the subclass relation), and dtype to encode the class type enclosing a field (i.e., the class where a field is declared). For example, the RL formula $u \in \mathbf{alloc} \wedge \mathsf{type}(u) \leq \mathsf{dtype}(f)$ says that $u.f$ is defined. (Here, u denotes an allocated reference whose runtime type is a subclass of the class enclosing f's declaration; this is consistent with the semantics of field access in Java.) Because regions are untyped, i.e., may contain references of any class type, an image expression must account for references where a field access would be undefined. The semantics of RL reflects this constraint.

The syntax of RL is capable of expressing quantifier-free assertions of region logic (cf. [2,3]) with the exception of: field access expression $u.f$ in case f has type rgn, image expression $G`f$ in case f has type rgn, and type predicate $\mathsf{type}(K, x)$ where x has type rgn. See [21] for details on how to extend RL to handle these additional constructs.

Example. Following [2] we consider a finite binary tree with method $setLeftZero$ whose body has a single command $x.left.item := 0$. The $item$ field of parameter x's $left$ node is set to 0. (Each node has fields $item$, $left$ and $right$.) Specifications of the method, using region assertions, follow. The frame condition says that only the $item$ field of objects in r may be written.

requires $x \neq \mathbf{null} \wedge x.left \in r \wedge x.right \in s \wedge r \# s$
requires $r`left \subseteq r \wedge r`right \subseteq r \wedge s`left \subseteq s \wedge s`right \subseteq s$
requires $\forall o : Node \in s \cdot o.item > 0$
ensures $\forall o : Node \in s \cdot o.item > 0$
writable $r`item$

`Verl` works by translation to the intermediate form Boogie2 and uses the latter's VC generator, which is designed for performance rather than readability. To illustrate how the preceeding example is verified, we derived the following verification condition, for the **ensures** clause, by hand. It is a predicate on variables x and H, where H is a

two-dimensional array that represents the current heap.

$$
\begin{aligned}
&(1)\ (\forall h\colon \mathsf{arr}, o\colon \mathsf{ref}\cdot\mathsf{type}(o) \leq \mathit{Node} \wedge o \in \mathbf{alloc} \Rightarrow \mathsf{read}(h, o, \mathit{left}) = \mathbf{null} \\
&\qquad \vee (\mathsf{read}(h, o, \mathit{left}) \in \mathbf{alloc} \wedge \mathsf{type}(\mathsf{read}(h, o, \mathit{left})) \leq \mathit{Node})) \wedge \\
&(2)\ \ldots \text{ ditto, with } \mathit{right} \text{ for } \mathit{left}\ \ldots \wedge \\
&(3)\ \mathsf{type}(x) \leq \mathit{Node} \wedge (x = \mathbf{null} \vee x \in \mathbf{alloc}) \wedge \\
&(4)\ x \neq \mathbf{null} \wedge \mathsf{read}(H, x, \mathit{left}) \in r \wedge \mathsf{read}(H, x, \mathit{right}) \in s \wedge r \# s \wedge \\
&(5)\ \mathsf{img}(H, r, \mathit{left}) \subseteq r \wedge \mathsf{img}(H, r, \mathit{right}) \subseteq r \wedge \\
&(6)\ \mathsf{img}(H, s, \mathit{left}) \subseteq s \wedge \mathsf{img}(H, s, \mathit{right}) \subseteq s \wedge \\
&(7)\ (\forall o\colon \mathsf{ref}\cdot\mathsf{type}(o) \leq \mathit{Node} \wedge o \in s \Rightarrow \mathsf{read}(H, o, \mathit{item}) > 0) \\
\Rightarrow\ &(8)\ (\forall o\colon \mathsf{ref}\cdot\mathsf{type}(o) \leq \mathit{Node} \wedge o \in s \Rightarrow \mathsf{read}(H', o, \mathit{item}) > 0)
\end{aligned}
$$

where $H' \mathrel{\widehat{=}} \mathsf{write}(H, \mathsf{read}(H, x, \mathit{left}), \mathit{item}, 0)$. Line (8) thus encodes the weakest precondition for the assignment $x.\mathit{left}.\mathit{item} := 0$ to establish the specified postcondition. Conjuncts (1)–(7) are essentially the translation of the **requires** clauses including additional assumptions. (The additional assumptions are conjuncts (1)–(3) which stem from the semantics of the programming language, namely definedness of field dereferences, type of x, whether x is allocated.) To prove the above VC is valid it suffices to prove unsatisfiability of its negation. The VC can be automatically proven using Z3 to reason about the quantifiers, types and arrays in conjunction with our (semi)decision procedure for quantifier-free region assertions.

Semantics. The semantics of RL is given by Def. 2. Therein we make use of the theory of arrays T_{A}, the theory of partial orders $T_{\leq}$, and the theory of equality T_{E}; the definitions are standard and therefore omitted (see [21, Sect. 4.2.2]). It is convenient to refer to the union of these theories as the *background-theory*.

Definition 1 (Background Theory).

- Let $T_{\mathsf{A}} \cup T_{\mathsf{E}} \cup T_{\leq}$ be called the *background-theory* with respect to RL.
- Let any literal from $\Sigma_{\mathsf{A}} \cup \Sigma_{\mathsf{E}} \cup \Sigma_{\leq}$ be called a *background-literal*.
- Let Φ be any conjunction of RL-literals. We say Φ is *background-satisfiable* iff all background-literals of Φ are satisfiable modulo the background-theory.

Each of T_{A}, T_{E}, $T_{\leq}$ admits a decision procedure, each theory is infinitely stable,[4] and no two signatures share any function or predicate symbol except the $=$ symbol. Thus, a decision procedure for the background-theory can be obtained by combining the decision procedures for T_{A}, T_{E}, and $T_{\leq}$ à la Nelson-Oppen [20].

Definition 2 (RL-interpretation). An RL-*interpretation* is a Σ_{RL}-interpretation, $\mathscr{I}$, such that

- each sort $\tau \in \{\mathsf{ref}, \mathsf{rgn}, \mathsf{arr}, \mathsf{cname}, \mathsf{fname}\}$ is mapped to a non-empty set I_τ
- $I_{\mathsf{rgn}} = \mathscr{P}(\mathbf{alloc}^{\mathscr{I}})$, $\quad \mathbf{alloc}^{\mathscr{I}} \in \mathscr{P}(I_{\mathsf{ref}} \setminus \{\mathbf{null}^{\mathscr{I}}\})$, $\quad$ and $\mathbf{emp}^{\mathscr{I}} = \varnothing$
- symbols $\leq$ and $\mathsf{read}, \mathsf{write}$, are interpreted according to $T_{\leq}$, T_{A}, respectively
- symbols $\in, \cup, \cap, -$, are interpreted in the standard (set-theoretic) way

[4] A theory T is *infinitely stable* if every T-satisfiable formula has an infinite model.

– $\{a\}^{\mathscr{I}} = \{a\} \cap \mathbf{alloc}^{\mathscr{I}}$, for every $a \in I_{\mathsf{ref}}$
– For every $h \in I_{\mathsf{arr}}, r \in I_{\mathsf{rgn}}, f \in I_{\mathsf{fname}}$,

$$\mathsf{img}^{\mathscr{I}}(h,r,f) = \{\mathsf{read}^{\mathscr{I}}(h,a,f) \mid a \in r \wedge \mathsf{type}^{\mathscr{I}}(a) \leq^{\mathscr{I}} \mathsf{dtype}^{\mathscr{I}}(f)\} \cap \mathbf{alloc}^{\mathscr{I}}$$

The constant **alloc** is assigned any subset of the non-empty domain I_{ref} which excludes $\mathbf{null}^{\mathscr{I}}$. In region logic, **alloc** is implicitly updated following allocation so it contains the references to currently allocated objects, which serves to reason about freshness. The domain I_{rgn} is interpreted to be the set of all subsets[5] of $\mathbf{alloc}^{\mathscr{I}}$. Observe that I_{rgn} is non-empty since it contains at least the empty set. The quasi singleton is so named because $\{u\}^{\mathscr{I}}$ is empty if $u^{\mathscr{I}}$ is not in $\mathbf{alloc}^{\mathscr{I}}$. Note that **alloc** is not required to be finite; we return to this later.

The verification conditions generated by the `Verl` tool [25] impose additional constraints, in particular, heaps have no dangling references and object fields have values compatible with their types. For these constraints, quantifiers work well (e.g., see [17]), and they are not relevant in this paper.

The subset relation for regions can be expressed in more than one way; e.g., $r \subseteq s$ is equivalent to $r \cup s = s$ and to $r \cap s = r$. In the sequel, let $r \cup s = s$ be synonymous with $r \subseteq s$ unless otherwise noted. The membership predicate can encoded using other relations: $u \in r$ is equivalent to $\{u\} \subseteq r \wedge \{u\} \neq \mathbf{emp}$. We include $\in$ in the core syntax because membership is used directly by our decision procedure.

The image term $\mathsf{img}(h,r,f)$ is so called because its denotation is a region obtained by computing the (functional) *image* of r under f pointwise, for those points a where f is "defined", viz. the constraint $\mathsf{type}^{\mathscr{I}}(a) \leq \mathsf{dtype}^{\mathscr{I}}(f)$. To express it using the standard notion of images, $\mathsf{img}(h,r,f)$ is the image of r under $F\colon I_{\mathsf{ref}} \to I_{\mathsf{ref}}$, where F is the partial function obtained by restricting $\mathsf{read}^{\mathscr{I}}$ appropriately: $((h,a,f),b) \in F$ iff $a \in r$ and $\mathsf{read}^{\mathscr{I}}(h,a,f) = b$ and $\mathsf{type}^{\mathscr{I}}(a) \leq^{\mathscr{I}} \mathsf{dtype}^{\mathscr{I}}(f)$ and $b \in \mathbf{alloc}^{\mathscr{I}}$.

Definition 3 (RL**-model).** An RL-*model* of an RL formula Φ is an RL-interpretation that makes the formula **true**. We say $\mathscr{M} \models \Phi$ to denote that $\mathscr{M}$ is an RL-model for Φ.

Here and in the sequel, "satisfiable" typically denotes T-satisfiable. That is, Φ is satisfiable *modulo* theory T. We can say this concisely using the satisfaction relation: $\models_T \Phi$. When the theory is clear from the context, it can be elided (as in Def. 3).

3 RL-Tableau Calculus

Our aim is to decide whether a given *conjunction* of RL-literals is satisfiable. (An arbitrary formula is converted to CNF whence it suffices to guess a satisfiable conjunction of literals.) Our solution is based on the (refutational) proof method of *analytic tableaux* [23,8]. The tableau calculus we describe was inspired by the work of Zarba [27].

Given an arbitrary RL-conjunction Φ, RL-tableau calculus lets us infer all membership literals (literals of the form $u \in r$, $u \notin r$) as well as background-literals entailed

[5] This reflects the semantics in [2] that regions contain only allocated references, also implemented in Verl [25].

by Φ. We apply inference rules until either inconsistency is detected or every (applicable) rule has been applied. Some inference rules are *conjunctive*, meaning that they infer a conjunction of literals, while others are *disjunctive*, meaning that they infer a disjunction of (conjunctions of) literals —a case split.

A *tableau* is a rooted, finitely branching tree with literals as nodes. Following standard terminology, a *branch* is a path from the root that is maximal —i.e., it includes a leaf or is infinite. A tableau rule is applied to a branch. If applicable, a conjunctive rule adds one or more literals, extending the branch linearly. A disjunctive rule creates a fork, so the branch becomes several branches.

In a nutshell, the proof method works as follows. For the formula Φ to be decided, construct an initial tableau comprising a single branch whose nodes are the conjuncts of Φ. Repeatedly, non-deterministically choose an inference rule which when applied adds new nodes, possibly splitting the branch. The goal is to try to close each branch by determining that some of its nodes are contradictory. If we succeed in closing every branch then Φ is unsatisfiable. On the other hand, if there exists an *open* (i.e., not closed) branch and every rule instance has been applied, then Φ is satisfiable.

RL-tableau calculus comprises the rules in Figure 1. The premise of each rule is composed of a set of literals and possibly some subterm occurrence checks denoted by occurs predicate; $\mathsf{occurs}(t)$ holds whenever term t occurs as a (sub)term in any of the literals of a given branch. The conclusion of a conjunctive rule (such as the first of the $\cap$-rules and the second of the img-rules) is composed of a set of literals; for a disjunctive rule (such as the third of the $=$-rules, the second of the $\{\cdot\}$-rules, the $\in$-rules, and the first of the img-rules) each disjunct is associated with some set of literals.

Rules are applied within a branch. Thus for a given branch B, in order to apply a rule, we must find an applicable rule instance for B. A rule instance assigns terms to the free variables occurring in the corresponding rule. Subsequently, to check if the rule instance is applicable we must verify that the instantiated premise holds. E.g., let σ denote a rule instance in B for the first $\cap$-rule in Fig. 1 such that $\sigma(u) = u$, $\sigma(r) = r$, $\sigma(s) = r \cup s$. Then, σ is applicable iff $u \in r \cap (r \cup s)$ occurs in B. As a result of applying σ, we would add $u \in r$ and $u \in r \cap (r \cup s)$ to B.

The rules marked with $(*)$ create fresh variables denoted by w; freshness is enforced by negative occurrence checks. (While other rules may yield fresh terms, e.g., $\mathsf{read}(h, u, f)$ in the first img-rule in Fig. 1, these terms do not occur in antecedents; hence, only a bounded number of such terms can be created.) We preclude "dumb" rule applications—those rule applications which repeatedly apply the same rule instance—by tracking rule instances which already have been applied (see [21, Sect. 4.3]). E.g., for a given literal $r \neq s$ we can apply the third $=$-rule in Fig. 1 exactly once; the instantiation of w is irrelevant.

Definition 4 (Closed Branch). A branch B of an RL-tableau is *closed* iff it contains any of the following contradictory sets of literals

- any conjunction of background-literals *unsatisfiable* modulo the background-theory
- any two complementary $\in$-literals; i.e., literals of the form $u \in r$ and $u \notin r$
- any literal of the form $u \in \mathbf{emp}$

A tableau is *closed* iff all of its branches are closed. A branch/tableau which is not closed is *open*.

$=$-rules

$$\frac{r = s \qquad u \in r}{u \in s} \qquad \frac{r = s \qquad u \in s}{u \in r} \qquad \frac{r \neq s \quad \neg\mathsf{occurs}(w)}{\begin{array}{l|l} w \in r & w \in s \\ w \notin s & w \notin r \end{array}}\ (*)$$

$\cap$-rules

$$\frac{u \in r \cap s}{u \in r \qquad u \in s} \qquad \frac{u \in r \qquad u \in s \qquad \mathsf{occurs}(r \cap s)}{u \in r \cap s}$$

$\{\cdot\}$-rules

$$\frac{u \in \{v\}}{u = v} \qquad \frac{\mathsf{occurs}(\{u\})}{u \notin \textbf{alloc} \mid u \in \{u\}}$$

$\in$-rules

$$\frac{u \in r \qquad v \in s \qquad \mathsf{occurs}(r \cap s)}{u \in s \mid u \notin s} \qquad \frac{u \in r \qquad \mathsf{occurs}(r - s)}{u \in s \mid u \notin s}$$

$$\frac{u \in r}{u \in \textbf{alloc}} \qquad \frac{u \in r}{u \neq \textbf{null}} \qquad \frac{u \in r \qquad v \notin r}{u \neq v}$$

img-rules

$$\frac{u \in r \qquad \mathsf{occurs}(\mathsf{img}(h,r,f))}{\mathsf{type}(u) \not\leq \mathsf{dtype}(f) \mid \mathsf{read}(h,u,f) \notin \textbf{alloc} \mid \mathsf{read}(h,u,f) \in \mathsf{img}(h,r,f)}$$

$$\frac{u \in \mathsf{img}(h,r,f) \qquad \neg\mathsf{occurs}(w)}{w \in r \qquad \mathsf{type}(w) \leq \mathsf{dtype}(f) \qquad \mathsf{read}(h,w,f) = u}\ (*)$$

Fig. 1. Selected RL-tableau rules. (Omitting set union and difference, see [21, Fig. 4.1]).

The first condition in Def. 4 uses a decision procedure for the background-theory (see Sect. 2). The remaining conditions are purely syntactic. Intuitively, we need the first condition because the following rules propagate background-literals: the first $\{\cdot\}$-rule, fourth and fifth $\in$-rules and both img-rules.

A branch B of an RL-tableau is *satisfiable* iff there exists an RL-model for the conjunction of all literals in B. A tableau is *satisfiable* iff at least one of its branches is satisfiable.

If a branch is satisfiable, then by semantics (Def. 2) none of the conditions in Def. 4 can hold. Therfore, a satisfiable branch is open. The other direction—an open branch is satisfiable, may not hold. E.g., the branch corresponding to $u \in r \cap s$, $u \notin r$, is open by Def. 4. Yet, it is easily seen that an application of the first $\cap$-rule will yield a new branch, with the literals $u \in r$ and $u \in s$; this branch is closed owing to the second condition in Def. 4. Thus, a priori we cannot determine whether an open branch is satisfiable unless all rules in the branch have been exhaustively applied.

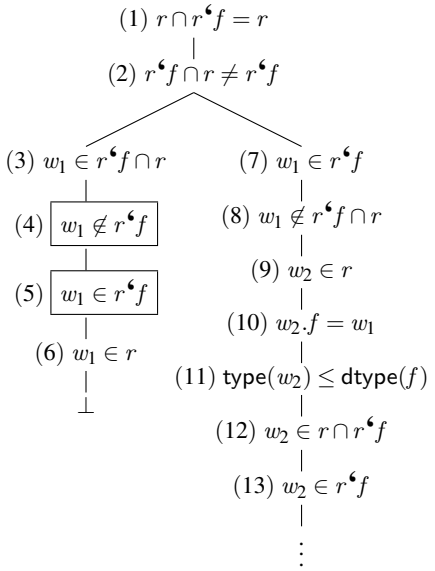
A branch B of a RL-tableau is *saturated* iff whenever a rule instance applies, and is not "dumb", the literals in its conclusion already occur in B. A tableau is *saturated* iff all of its branches are saturated.

Intuitively, a saturated branch is closed under all possible inferences. The notion of saturation plays a key role in establishing completeness (of the proof method). That is, if an RL-tableau for Φ yields an open and saturated branch, then Φ is satisfiable. Conversely, if the tableau is closed, then Φ is unsatisfiable.

A branch B of a RL-tableau is *completed* iff it is either saturated or closed. A tableau is *completed* iff all of its branches are completed. A completed tableau can be obtained by applying the rules until every branch either becomes closed or saturated.

Theorem 1 (Soundness). *Let Φ be a conjunction of* RL*-literals. If there exists a closed* RL*-tableau for Φ, then Φ is unsatisfiable.*

Theorem 2 (Completeness). *Let Φ be a conjunction of* RL*-literals. If Φ is unsatisfiable, then every completed* RL*-tableau for Φ is closed.*



- (3), (4) *and* (7), (8) from (2) by third of $=$-**rules**
- (5), (6) from (3) by first of $\cap$-**rules**
- (9), (10), (11) from (7) by second of img-**rules**
- (12) from (1), (9) by second of $=$-**rules**
- (13) from (12) by first of $\cap$-**rules**

Boxed nodes are contradictory; right branch is infinite.

Fig. 2. RL-tableau for $r \subsetneq r\text{`}f$, that is, $r \cap r\text{`}f = r \wedge r\text{`}f \cap r \neq r\text{`}f$.

Example. Several illustrative examples of RL-tableau are given in [21, Sect. 4.4]. Here, we describe one for which every completed RL-tableau must be infinite. Fig. 2 illustrates an RL-tableau for the conjunction $r \cap r\text{`}f = r \wedge r\text{`}f \cap r \neq r\text{`}f$ which denotes: $r$ is a *proper* subset of $r\text{`}f$. (For brevity we use region logic notation $r\text{`}f$ to stand for $\mathsf{img}(h, r, f)$ since the example only involves a single heap.) Observe that if the right branch is to be completed, then the branch must be infinite; the second img-rule will have been applied infinitely often. The tableau in Fig. 2 is not unique; e.g., we could swap the right and the left branches to obtain another RL-tableau. However, it is not

difficult to see that every completed RL-tableau for the given conjunction is infinite. Intuitively, $r \subseteq r\text{`}f$ expresses that there exists a function $f\colon\ r \to r \cup s$, for some $s$, such that $f$ is surjective onto $r$, whereas $r\text{`}f \not\subseteq r$ expresses that the range of f extends beyond r. No such function exists for any finite r.

4 Implementation of Semi-Decision Procedure

Using the RL-tableau calculus we can construct a completed tableau $\mathscr{T}$ for any given RL-conjunction Φ. If $\mathscr{T}$ is closed, then owing to Theorem 1 Φ is unsatisfiable. If $\mathscr{T}$ is open, then owing to Theorem 2 Φ is satisfiable; indeed $\mathscr{T}$ determines a model. As witnessed by the previous example, some RL-conjunctions may yield completed RL-tableaux which are infinite. However, we can obtain a semi-decision procedure by ensuring that rules are applied in a systematic fashion.[6] Essentially, img-rules are applied in a lock-step fashion; i.e., apply exhaustively non-img-rules, apply img-rules for occurring terms, repeat.

The semi-decision procedure for RL has been implemented as a theory plugin in Z3; theory plugins are based on the DPLL(T) architecture [9]. In a nutshell, Z3 guesses a conjunction of literals, say L, which must hold. The plugin is notified with each asserted RL-literal $l \in L$ at which point it asserts new theory lemmas into Z3's context. (Z3's context represents the current tableau branch.)

The theory lemmas are nothing more than instances of the RL-tableau rules. Each rule is potentially applied to ensure saturation.[7] Reasoning modulo the background-theory is peformed entirely within Z3. (New background-literals are propagated by insantiating rules, e.g., first $\{\cdot\}$-rule.) Thus, to check if a branch is closed, we merely check if the literals $u \in r$ and $u \notin r$ have been asserted or the literal $u \in \mathbf{emp}$ has been asserted; both checks are purely syntactic. E.g., if $u \in \mathbf{emp}$ is asserted, then we simply assert $u \in \mathbf{emp} \Rightarrow \mathsf{false}$.

Our preliminary evaluation used synthetic benchmarks [21, Figs. 4.13, 4.14] to compare the performance of the semi-decision procedure versus an axiomatization of RL which relies on Z3's quantifier reasoning. The looping phenomenon (cf. Fig. 2) is observed in some cases, when the image rules are implemented directly as presented in Fig. 1. Our implementation includes an option to switch on certain heuristics described in [21, Sect. 4.4.1]. With that option all our benchmarks terminate, but there is no guarantee in general. The benchmarks include basic properties of boolean algebra and function images, as well as formulas, both valid and invalid, involving the full signature of RL except array writes. These capture typical verification conditions except for excluding literals involving integers or array manipulation, our goal being to focus evaluation on performance of the tableau procedure itself.

The procedure terminates fast (under 50ms) for valid formulas (i.e., `UNSAT` for their negations). For invalid formulas, it terminates fast with `UNKNOWN`, due to (potential) incomplete[8] quantifier reasoning in the theory of partial orders; partial order is typically

[6] The same idea is used in Smullyan's tableaux for first-order logic. For details see [21, Fig. 4.7].

[7] We describe how to do this efficiently in [21, Sect. 4.11].

[8] As of release 2.17, Z3 supports complete instantiation though it is not yet fully integrated with theory plugins.

axiomatized in Z3. For these benchmarks, however, it is sound to treat read, type, dtype, and $\leq$ as uninterpreted; then all the invalid benchmarks terminate under 50ms. Our results suggest that the semi-decision procedure has the potential to scale (see [21, Sect. 4.12]). Even on small benchmarks, its performance is much more predictable than the one that is reliant on quantifier instantiation. For example, out of 44 `UNSAT` benchmarks, `Z3` timed out on 32 of them when using axioms for the region theory, while the theory plugin returned `UNSAT` in under 50ms for each of the 44 benchmarks. (Timeout was set to 100 seconds.) For the 8 `UNSAT` benchmarks on which `Z3` terminated, its performance exhibited high variance (due to quantifiers).

5 Restricted-$\mathsf{RL_E}$-tableau Calculus

Unrestricted RL appears to be of high complexity. In [21, Prop. 4.33] we show that it becomes NEXPTIME-hard if we change Def. 2 to require **alloc** to be interpreted as a *finite* set. (The proof builds on ideas of [10,26].) By imposing simple syntactic restrictions we obtain a theory for which satisfiability is NP-complete, yet which is expressive enough to encompass the verification conditions that arise from the specifications we have used in case studies [1,22]. We retain Def. 2 unchanged.

Definition 5 (restricted-$\mathsf{RL_E}$, restricted-RL). A restricted-$\mathsf{RL_E}$ literal is one such that

- there is no write symbol
- img symbol occurs only in the forms $\mathsf{img}(h,r,f) \subseteq s$ and $u \notin \mathsf{img}(h,r,f)$ where r, s are any img-free rgn-terms and h, f, u are any terms of the appropriate sort

A restricted-RL literal is one that satisfies the second of these restrictions.

The first restriction is only superficial since the theory of arrays can be reduced to the theory of equality by eliminating write terms. (See [21, Sect. 4.6] or [11].) The second restriction is key in establishing NP upper-bound; it essentially disallows literals of the form $s \subseteq \mathsf{img}(h,r,f)$ while allowing literals of the form $\mathsf{img}(h,r,f) \subseteq s$.

Our complexity result is the same for both restricted-RL and restricted-$\mathsf{RL_E}$. However, for technical [9] reasons our tableaux are formulated for restricted-$\mathsf{RL_E}$.

New rules. The restricted-$\mathsf{RL_E}$ tableau calculus has the img-rules in Fig. 3 which replace the img-rules in Fig. 1. The other rules are the same as in Fig. 1. The background-theory, T_{BG}, for restricted-$\mathsf{RL_E}$ is the union of the theory of equality, T_{E}, and the theory of partial orders, $T_{\leq}$. The rules in Fig. 3 provide complete reasoning about img literals for the restricted-$\mathsf{RL_E}$ theory. (R1) is nearly the same as the first img-rule in Fig. 1. The only difference is the addition of $\mathsf{type}(u) \leq \mathsf{dtype}(f)$ to the right-most disjunct. (R2), however, is an entirely new rule. (R2) deals with the case when $\mathsf{read}(h,u,f)$ does not occur in a branch (and hence not in the input conjunction). Intuitively, (R2) says that if some $\mathsf{read}(h',u',f')$ occurs in a branch, then we must guess if $\mathsf{read}(h',u',f') \in \mathsf{img}(h,r,f)$; the entailment conditions express that $\mathsf{read}(h',u',f')$ has the same denotation as $\mathsf{read}(h,u,f)$. That is, the equalities $h' = h$, $u' = u$, $f' = f$ are implied by the current branch, modulo the background-theory.

[9] The background theory of restricted-$\mathsf{RL_E}$ is convex; not so for restricted-RL, due to arrays.

(R1)

$$\frac{u \in r \qquad \mathsf{occurs}(\mathsf{img}(h,r,f)) \qquad \mathsf{occurs}(\mathsf{read}(h,u,f))}{\mathsf{type}(u) \not\leq \mathsf{dtype}(f) \mid \mathsf{read}(h,u,f) \notin \mathbf{alloc} \mid \begin{array}{l}\mathsf{type}(u) \leq \mathsf{dtype}(f)\\ \mathsf{read}(h,u,f) \in \mathsf{img}(h,r,f)\end{array}}$$

(R2)

$$\frac{\begin{array}{c}u \in r \qquad \mathsf{occurs}(\mathsf{img}(h,r,f)) \qquad \neg\mathsf{occurs}(\mathsf{read}(h,u,f)) \qquad \mathsf{occurs}(\mathsf{read}(h',u',f'))\\ \models_{T_{\mathsf{BG}}} h' = h \qquad \models_{T_{\mathsf{BG}}} u' = u \qquad \models_{T_{\mathsf{BG}}} f' = f\end{array}}{\mathsf{type}(u') \not\leq \mathsf{dtype}(f') \mid \mathsf{read}(h',u',f') \notin \mathbf{alloc} \mid \begin{array}{l}\mathsf{type}(u') \leq \mathsf{dtype}(f')\\ \mathsf{read}(h',u',f') \in \mathsf{img}(h,r,f)\end{array}}$$

Fig. 3. img rules for restricted-RL_E tableau.

Checking implied equalities. One way to fulfill the semantic checks in (R2) is to compute all the implied equalities on all the background-terms in $\pi(\mathsf{B})$. To check if an equality $u = v$ is implied by the current branch modulo the background-theory, it suffices to check unsatisfiability of the conjunction $\pi(\mathsf{B}) \wedge u \neq v$ modulo T_{BG}. (Here $\pi(\mathsf{B})$ is the conjunction of all background-literals in the branch.)

Computing all implied equalities every time (R2) is considered would be inefficient. We describe one possible optimization. The optimization relies on the observation that fresh equality literals are added by the first $\{\cdot\}$-rule in Fig. 1; remaining rules can add background-literals of the form $u \neq v$, $u \neq \mathbf{null}$ and $\mathsf{type}(u) \not\leq \mathsf{dtype}(f)$, none of which could imply new equalities. Consequently, all implied equalities can be precomputed and updated whenever the first $\{\cdot\}$-rule is applied in the branch.

In practice, the implementation of (R2) may not need to query implied equalities. SMT solvers typically implement Nelson-Oppen combination method [18]. Z3 uses model-based theory combination [5]. In both frameworks, all implied equalities on mutually shared variables are eventually propagated to all other theories.

Theorem 3 (Soundness). *Let Φ be any conjunction of restricted-RL_E-literals. If there exists a closed restricted-RL_E-tableau for Φ, then Φ is unsatisfiable.*

Theorem 4 (Completeness). *Let Φ be any conjunction of restricted-RL_E literals. If Φ is unsatisfiable, then every completed restricted-RL_E tableau for Φ is closed.*

Complexity. Observe that the img-rules in Fig. 3 can create fresh terms only of the form $\mathsf{type}(u)$, $\mathsf{dtype}(f)$, both of sort cname, where neither u nor f is fresh; the number of such fresh terms is $\mathcal{O}(n^2)$ where n is the size of the input. This is in stark contrast to the img-rules in Fig. 1 which can create an unbounded number of fresh terms of sort ref (e.g., Fig. 2). We can derive the following branch bound for restricted-RL_E tableaux.

Lemma 1 (branch bound). *Let Φ be any restricted-RL_E conjunction. Let $\mathsf{size}(\Phi) = n$. Let B be any branch of a restricted-RL_E tableau for Φ. Then, $\mathsf{size}(\mathsf{B})$ is $\mathcal{O}(n^3)$.*

Conjunctions of resricted-RL_E literals suffice to encode arbitrary boolean clauses. Intuitively, non-determinism due to disjunctions can be encoded by singleton sets. Thus we have the following result ([21, Lemma 4.57]).

Lemma 2. *Deciding the satisfiability of a conjunction of restricted-*$\mathsf{RL_E}$ *literals is* NP*-hard.*

Owing to Lemma 1 and the fact that the premises in (R1) and (R2) of Fig. 3 can be checked in polynomial time,[10] we can formulate a non-deterministic decision procedure which runs in polynomial time. Given an arbitrary restricted-$\mathsf{RL_E}$ formula, Φ, we first transform it into CNF using Tseitsin's encoding, e.g., [13]. (The equisatisfiable CNF formula is linear in the size of Φ.) Next, we guess a conjunction of disjuncts and use it as the input to the tableau procedure; the tableau procedure merely applies all possible tableau rules in a non-deterministic fashion until each branch is complete.

Lemma 3. *Deciding the satisfiability of a restricted-*$\mathsf{RL_E}$ *formula is in* NP.

Theorem 5. *The satisfiability problem for restricted-*$\mathsf{RL_E}$ *is* NP*-complete.*

Recall that restricted-$\mathsf{RL_E}$ is obtained from restricted-RL by eliminating all array literals, i.e., literals containing write-terms. The reduction introduces only polynomially many fresh literals (see [21, Lemma 4.62]). Consequently, we can apply the reduction and appeal to Theorem 5 to obtain

Theorem 6. *The satisfiability problem for restricted-*RL *is* NP*-complete.*

6 Related Work

Our decision procedure extends the tableau-based decision procedure for the quantifier-free language **2LST**—*two-level syllogistic modulo* T [27]. **2LST** is an extension of **2LS**—a two-sorted language of sets of elements where the element sort is uninterpreted. Additionally, **2LST** has any number of constant, function and predicate symbols over the elem sort in some theory T, provided as a parameter. The function and predicate symbols are of the form: $F\colon \mathsf{elem} \times \cdots \times \mathsf{elem} \to \mathsf{elem}$, $P\colon \mathsf{elem} \times \cdots \times \mathsf{elem}$, of any arity. The interpretation of these symbols is dictated according to T. The decision problem is NP-complete assuming the decision problem for T is in NP. A tableau-based decision procedure for **2LST** was presented in [27]. It corresponds essentially to our Fig. 1, excluding the second $\{\cdot\}$, and the third and fourth $\in$-rules; in our setting, the background-theory plays the role of T. Note, if we keep the third $\in$-rule, we essentially obtain a decision procedure for **2LST** with a universal set, denoted by **alloc**. Consequently, RL can be seen as an extension of **2LST** with a universal set and images.

The tableau-based decision procedure is a combination method different from that of Nelson-Oppen. Notably, it does not perform the equality propagation between T and **2LS** in the sense of Nelson-Oppen. (Although, equalities amongst T-terms are propagated by tableau rules.) Furthermore, T need not be stably infinite. Intuitively, a decision procedure for T serves as a *black box*. After a rule is applied, a tableau simply asks the black box to determine if the new branch(es) remains open, i.e., whether a conjunction of elem-literals is T-satisfiable.

[10] This stems from the fact that T_{E}, $T_{\leq}$ are convex and can be decided in polynomial time.

Kuncak, et al. give a decision procedure for a quantified language of sets of uninterpreted elements with cardinality constraints [14]. The language is known as **BAPA**—boolean algebra with Presburger arithmetic. It permits quantification over sets and integers; quantification over elements is expressible in terms of set quantification. The decision procedure for **BAPA** admits quantifier-elimination. The restriction to quantifier-free formulas is called **QFBAPA**. This language has no separate syntax for element terms; elements are encoded by fresh set-variables whose cardinality is constrained to be 1. Remarkably, **QFBAPA**'s decision problem was shown to be NP-complete by Kuncak and Rinard [16].

Yessenov, et al. introduce a decidable language **QFBAPA**-Rel with image expressions under unary function symbols and predicate symbols of any arity [26]. As in **QFBAPA**, the element sort is un-interpreted. This language is very expressive and suited to verification of object based programs. It comes close to subsuming RL, indeed one of their examples is based on our specification for $setLeftZero$. Their functions are total whereas our images are not: region r in $r`f$ may contain some objects that lack field f. Perhaps this can be patched by introducing types.

For a function $f : A \to A$ and set $X \subseteq A$, the decision procedure of [26] eliminates terms of the form $f[X]$ by first rewriting X as a union of (disjoint) Venn regions, i.e., $X = \bigcup v_i$ and $f[X] = \bigcup f[v_i]$. Subsequently, each $f[v_i]$ is replaced by a fresh variable t_i with the cardinality constraints: $|t_i| \leq |v_i|$ and $|t_i| = 0 \Leftrightarrow |v_i| = 0$. The resulting formula is in **QFBAPA** which is NP-complete, thus the decision problem is in NEXPTIME. However, because the translation (to Venn regions) yields a **QFBAPA** formula of exponential size, the decision procedure does not seem practical.

Recently Suter et al. [24] have shown that it is possible to obtain an SMT-based decision procedure for **QFBAPA**. Their decision procedure has been implemented as a plugin in Z3. To reason about interpreted elements of sets, axiomatized predicate symbols singleton and element are added to **QFBAPA** subject to axioms $|\mathsf{singleton}(\mathsf{e})| = 1$ (for singleton set) and $\mathsf{element}(\mathsf{singleton}(e)) = e$. The crux of the decision procedure is an algorithm that decomposes a formula such that the number of considered Venn regions is significantly reduced. Several experiments show that in practice the decomposition algorithm can handle formulas with a large number of set variables, despite exponential worst-case complexity. Suter et al. conjecture that their approach can be extended to **QFBAPA**-Rel.

7 Discussion

We conjecture that it is possible to devise a terminating tableau procedure to decide full RL, with respect to interpretations where **alloc** is finite; but this is unlikely to be of practical value because of its NEXPTIME-time complexity. (Our procedure is incomplete for finite **alloc**, as illustrated by the example in Fig. 2.) We conjecture that NEXPTIME-time complexity also holds for RL without the restriction to finite **alloc**. For typical verification condition generators, there is no need to explicitly require the heap domain to be finite. However, if **alloc** is fixed to be finite, then one can express cardinality constraints of the form $|r| \leq |s|$, for any rgn-terms r, s. Observe that under a finite interpretation of regions, $r \subseteq \mathsf{img}(h, s, f)$ implies the cardinality constraint

$|r| \leq |s|$. (By cardinality, $r \subseteq \mathsf{img}(h,s,f)$ implies $|r| \leq |\mathsf{img}(h,s,f)|$; by image semantics and finiteness of regions, $|\mathsf{img}(h,s,f)| \leq |s|$, whence $|r| \leq |s|$ by transitivity.) The case of restricted-RL is simpler since every satisfiable formula has a finite model owing to Lemma 1 and Theorem 4.

In previous work we used a translation to the **BSR** fragment[11] inspired by [15]. For example a restricted-RL literal $r`f \subseteq s$ where r, s are variables, can be roughly encoded by a **BSR** formula $\forall u, v \cdot v \neq \textbf{null} \Rightarrow (r(u) \wedge f(u,v) \Rightarrow s(v))$, where r, s, f are predicate symbols. While the translation is possible for a language akin to restricted-RL, it breaks down when we encounter RL literals of the form $s \subseteq r`f$; such a literal would result in a formula with the quantifier prefix $\forall\exists$ which does not belong to **BSR**.

Although the procedures presented here seem promising, much more thorough performance evaluation is needed. The semi-decision procedure is not currently integrated with the `Verl` tool, but we have already instrumented `Verl` with the necessary hooks. We plan to complete that integration and also to implement the decision procedure for restricted RL. This will enable comparison between performance of the (semi)-decision procedures and the axiomatic implementation of RL already present in `Verl`. (And with **QFBAPA**-Rel, if an SMT-based implementation becomes available.) Ordinary use of `Verl` will assess performance on full VCs for a range of correct and incorrect programs, i.e., involving integers and other theories besides RL. From those VCs we may also extract benchmark formulas in the RL and restricted RL fragments, for more direct comparative evaluation of the procedures.

As presented here, RL reflects the semantics in the original paper on region logic [2]. Subsequently we streamlined the assertion language by allowing regions to contain null [3]; this validates a slightly different set of formulas (e.g., $x \in \{x\}$ becomes valid, whereas only $x \neq \textbf{null} \Rightarrow x \in \{x\}$ is valid according to Def. 2). It should be straightforward to adapt the tableau procedures to this semantics.

Acknowledgements. Thanks to Nikolaj Bjørner and Leonardo de Moura for help with Z3 integration and feedback on the implementation. Thanks to Clark Barrett for studying proofs of soundness and completeness of tableaux. Thanks to the anonymous referees for their comments.

References

1. A. Banerjee, M. Barnett, and D. A. Naumann. Boogie meets regions: a verification experience report. In *Verified Software: Theories, Tools, Experiments*, pages 177–191, 2008.
2. A. Banerjee, D. A. Naumann, and S. Rosenberg. Regional logic for local reasoning about global invariants. In *ECOOP*, pages 387–411, 2008.
3. A. Banerjee, D. A. Naumann, and S. Rosenberg. Local reasoning for global invariants, part I: Region logic. Extended version of [2], available at [25]., July 2011.
4. M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *FMCO*, pages 364–387, 2005.
5. L. de Moura and N. Bjørner. Model-based theory combination. *Electr. Notes Theor. Comput. Sci.*, 198(2):37–49, 2008.

[11] Bernays-Schönfinkel-Ramsey fragment.

6. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.
7. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.
8. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Graduate texts in Computer Science. Springer, 1996.
9. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CAV*, pages 175–188, 2004.
10. R. Givan, D. A. McAllester, C. Witty, and D. Kozen. Tarskian set constraints. *Inf. Comput.*, 174(2):105–131, 2002.
11. D. Kapur and C. G. Zarba. A reduction approach to decision procedures. Technical report, University of New Mexico, 2005.
12. I. T. Kassios. Dynamic frames: Support for framing, dependencies and sharing without restrictions. In *FM*, pages 268–283, 2006.
13. D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. EATCS. Springer, 2008.
14. V. Kuncak, H. H. Nguyen, and M. C. Rinard. An algorithm for deciding BAPA: Boolean algebra with Presburger arithmetic. In *CADE*, pages 260–277, 2005.
15. V. Kuncak and M. C. Rinard. Decision procedures for set-valued fields. *Electr. Notes Theor. Comput. Sci.*, 131:51–62, 2005.
16. V. Kuncak and M. C. Rinard. Towards efficient satisfiability checking for Boolean algebra with Presburger arithmetic. In *CADE*, pages 215–230, 2007.
17. K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR (Dakar)*, pages 348–370, 2010.
18. Z. Manna and C. G. Zarba. Combining decision procedures. In *10th Anniversary Colloquium of UNU/IIST*, volume 2757 of *LNCS*, pages 381–422. Springer, 2002.
19. M. Marron, M. Méndez-Lojo, M. V. Hermenegildo, D. Stefanovic, and D. Kapur. Sharing analysis of arrays, collections, and recursive structures. In *PASTE*, pages 43–49, 2008.
20. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.
21. S. Rosenberg. *Region Logic: Local Reasoning for Java Programs and its Automation*. PhD thesis, Stevens Institute of Technology, June 2011. Available at [25].
22. S. Rosenberg, A. Banerjee, and D. A. Naumann. Local reasoning and dynamic framing for the composite pattern and its clients. In *VSTTE*, pages 183–198, 2010.
23. R. M. Smullyan. *First-Order Logic*. Springer, 1968.
24. P. Suter, R. Steiger, and V. Kuncak. Sets with cardinality constraints in satisfiability modulo theories. In *VMCAI*, pages 403–418, 2011.
25. Verl: VErifier for Region Logic. Software distribution, at `http://www.cs.stevens.edu/~naumann/pub/VERL/`.
26. K. Yessenov, R. Piskac, and V. Kuncak. Collections, cardinalities, and relations. In *VMCAI*, pages 380–395, 2010.
27. C. G. Zarba. Combining sets with elements. In *Verification: Theory and Practice*, pages 762–782, 2003.
28. K. Zee, V. Kuncak, and M. C. Rinard. Full functional verification of linked data structures. In *PLDI*, pages 349–361, 2008.