# Encoding Separation Logic in SMT-LIB v2.5


Radu Iosif[2], Cristina Serban[2], Andrew Reynolds[1], and Mihaela Sighireanu[3]

[1] The University of Iowa
[2] Verimag/CNRS/Université de Grenoble Alpes
[3] IRIF/Université Paris Diderot



**Abstract.** We propose an encoding of Separation Logic using SMT-LIB v2.5. This format is currently supported by SMT solvers (CVC4) and inductive proof-theoretic solvers (SLIDE and SPEN). Moreover, we provide a library of benchmarks written using this format, which stems from the set of benchmarks used in SL-COMP'14 [7].


## 1 Preliminaries

We consider formulae in multi-sorted first-order logic. A *signature* $\Sigma$ consists of a set $\Sigma^{\mathrm{s}}$ of sort symbols and a set $\Sigma^{\mathrm{f}}$ of *function symbols* $f^{\sigma_1 \cdots \sigma_n \sigma}$, where $n \geq 0$ and $\sigma_1, \ldots, \sigma_n, \sigma \in \Sigma^{\mathrm{s}}$. If $n = 0$, we call $f^{\sigma}$ a *constant symbol*. We make the following assumptions:

1. all signatures $\Sigma$ contain the Boolean sort $\mathsf{B}$, where $\top$ and $\bot$ denote the Boolean constants *true* and *false*.
2. $\Sigma^{\mathrm{f}}$ contains a boolean equality function $\approx^{\sigma\sigma\mathsf{B}}$ for each sort symbol $\sigma \in \Sigma^{\mathrm{s}}$.

Let $\mathsf{Vars}$ be a countable set of first-order variables, each $x^{\sigma} \in \mathsf{Vars}$ having an associated sort $\sigma$. First-order terms and formulae over the signature $\Sigma$ (called $\Sigma$-terms and $\Sigma$-formulae) are defined as usual. A first-order variable is *free* if it does not occur within the scope of a quantifier, and we write $\varphi(\mathbf{x})$ to denote that the free variables of the formula $\varphi$ belong to the set $\mathbf{x}$.

A $\Sigma$-*interpretation* $\mathcal{I}$ maps:

- each sort symbol $\sigma \in \Sigma$ to a non-empty set $\sigma^{\mathcal{I}}$,
- each function symbol $f^{\sigma_1,\ldots,\sigma_n,\sigma} \in \Sigma$ to a total function $f^{\mathcal{I}} : \sigma_1^{\mathcal{I}} \times \ldots \times \sigma_n^{\mathcal{I}} \to \sigma^{\mathcal{I}}$ where $n > 0$, and to an element of $\sigma^{\mathcal{I}}$ when $n = 0$, and
- each variable $x^{\sigma} \in \mathsf{Vars}$ to an element of $\sigma^{\mathcal{I}}$.

For an interpretation $\mathcal{I}$ a sort symbol $\sigma$ and a variable $x$, we denote by $\mathcal{I}[\sigma \leftarrow S]$ and, respectively $\mathcal{I}[x \leftarrow v]$, the interpretation associating the set $S$ to $\sigma$, respectively the value $v$ to $x$, and which behaves like $\mathcal{I}$ in all other cases. By writing $\mathcal{I}[\sigma \leftarrow S]$ we ensure that all variables of sort $\sigma$ are mapped by $\mathcal{I}$ to elements of $S$. For a $\Sigma$-term $t$, we write $t^{\mathcal{I}}$ to denote the interpretation of $t$ in $\mathcal{I}$, defined inductively, as usual. A satisfiability relation between $\Sigma$-interpretations and $\Sigma$-formulas, written $\mathcal{I} \models \varphi$, is also defined inductively, as usual. In this case, we say that $\mathcal{I}$ is a *model* of $\varphi$.

A (multi-sorted first-order) *theory* is a pair $T = (\Sigma, \mathbf{I})$ where $\Sigma$ is a signature and $\mathbf{I}$ is a non-empty set of $\Sigma$-interpretations, the *models* of $T$. A $\Sigma$-formula $\varphi$ is $T$-*satisfiable* if it is satisfied by some interpretation in $\mathbf{I}$.

## 2 Ground Separation Logic

Let $T = (\Sigma, \mathbf{I})$ be a theory and let $\mathsf{Loc}$ and $\mathsf{Data}$ be two sorts from $\Sigma$, with no restriction other than the fact that $\mathsf{Loc}$ is always interpreted as a countable set. Also, we consider that $\Sigma$ has a designated constant symbol $\mathsf{nil}^{\mathsf{Loc}}$. We define the *Ground Separation Logic* $\mathsf{SL}(T)^{g}_{\mathsf{Loc},\mathsf{Data}}$ to be the set of formulae generated by the following syntax:

$$\varphi := \phi \mid \mathsf{emp} \mid \mathsf{t} \mapsto \mathsf{u} \mid \varphi_1 * \varphi_2 \mid \varphi_1 \mathrel{-\!\!*} \varphi_2 \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \exists x^{\sigma} \,.\, \varphi_1(x)$$

where $\phi$ is a $\Sigma$-formula, and $\mathsf{t}$, $\mathsf{u}$ are $\Sigma$-terms of sorts $\mathsf{Loc}$ and $\mathsf{Data}$, respectively. As usual, we write $\forall x^{\sigma} \,.\, \varphi(x)$ for $\neg\exists x^{\sigma} \,.\, \neg\varphi(x)$. We omit specifying the sorts of variables and functions when they are clear from the context.

Given an interpretation $\mathcal{I}$, a *heap* is a finite partial mapping $h : \mathsf{Loc}^{\mathcal{I}} \rightharpoonup_{\mathrm{fin}} \mathsf{Data}^{\mathcal{I}}$. For a heap $h$, we denote by $\mathrm{dom}(h)$ its domain. For two heaps $h_1$ and $h_2$, we write $h_1 \# h_2$ for $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$ and $h = h_1 \uplus h_2$ for $h_1 \# h_2$ and $h = h_1 \cup h_2$. We define the *satisfaction relation* $\mathcal{I}, h \models_{\mathsf{SL}} \phi$ inductively, as follows:

$$
\begin{array}{ll}
\mathcal{I}, h \models_{\mathsf{SL}} \phi & \iff \mathcal{I} \models \phi \text{ if } \phi \text{ is a } \Sigma\text{-formula} \\
\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{emp} & \iff h = \emptyset \\
\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{t} \mapsto \mathsf{u} & \iff h = \{(\mathsf{t}^{\mathcal{I}}, \mathsf{u}^{\mathcal{I}})\} \text{ and } \mathsf{t}^{\mathcal{I}} \not\approx \mathsf{nil}^{\mathcal{I}} \\
\mathcal{I}, h \models_{\mathsf{SL}} \phi_1 * \phi_2 & \iff \text{there exist heaps } h_1, h_2 \text{ s.t. } h = h_1 \uplus h_2 \text{ and } \mathcal{I}, h_i \models_{\mathsf{SL}} \phi_i, i = 1, 2 \\
\mathcal{I}, h \models_{\mathsf{SL}} \phi_1 \mathrel{-\!\!*} \phi_2 & \iff \text{for all heaps } h' \text{ if } h' \# h \text{ and } \mathcal{I}, h' \models_{\mathsf{SL}} \phi_1 \text{ then } \mathcal{I}, h' \uplus h \models_{\mathsf{SL}} \phi_2 \\
\mathcal{I}, h \models_{\mathsf{SL}} \exists x^{S} . \varphi(x) & \iff \mathcal{I}[x \leftarrow s], h \models_{\mathsf{SL}} \varphi(x), \text{ for some } s \in S^{\mathcal{I}}
\end{array}
$$

The satisfaction relation for $\Sigma$-formulae, Boolean connectives $\wedge$, $\neg$, and linear arithmetic atoms, are the classical ones from first-order logic. Notice that the range of a quantified variable $x^{S}$ is the interpretation of its associated sort $S^{\mathcal{I}}$. A formula $\varphi$ is said to be *satisfiable* if there exists an interpretation $\mathcal{I}$ and a heap $h$ such that $\mathcal{I}, h \models_{\mathsf{SL}} \varphi$. We say that $\varphi$ *entails* $\psi$, written $\varphi \models_{\mathsf{SL}} \psi$, when every pair $(\mathcal{I}, h)$ which satisfies $\varphi$, also satisfies $\psi$.

### 2.1 SMT-LIB Encoding

We write ground SL formulae in SMT-LIB using the following functions:

```
(emp Bool)
(sep Bool Bool Bool :left-assoc)
(wand Bool Bool Bool :right-assoc)
(par (Loc Data) (pto Loc Data Bool))
(par (Loc) (nil Loc))
```

Observe that `pto` and `nil` are polymorphic functions, with sort parameters `Loc` and `Data`. There is no restriction on the choice of `Loc` and `Data`, as shown below.

Is `nil` always of sort `Loc` ? Here `Loc` is just a parameter with no special meaning.

Assume that Loc is an uninterpreted sort U, Data is the integer sort Int, and that $x^{\mathsf{U}}, y^{\mathsf{U}}, a^{\mathsf{Int}}$ and $b^{\mathsf{Int}}$ are constants:

```
(declare-sort U 0)

(declare-const x U)
(declare-const y U)
(declare-const a Int)
(declare-const b Int)
```

We write the SL formula $\mathsf{emp} \wedge ((x \mapsto a * y \mapsto b) \mathrel{-\!\!*} (x \mapsto \mathsf{nil} * \top))$ in SMT-LIB as follows:

```
(and emp (wand (sep (pto x a) (pto y b)) (sep (pto x (as nil Int)) true)))
```

In addition to the classical SMT-LIB typing constraints, the SL theories require that the heap models are well-typed. For instance, a separation constraint of the form:

```
(sep (pto x y) (pto a b))
```

with the above constant declarations results in a typing error, because `(pto x y)` requires the heap to be of type $\mathsf{U} \rightharpoonup \mathsf{U}$, whereas `(pto a b)` requires the heap to be of type $\mathsf{Int} \rightharpoonup \mathsf{Int}$, and combining heaps of different types is not allowed.

1. It is currently unclear what the type of the heap introduced by emp should be. The solution currently adopted in CVC4 is to give emp parameters used only to infer the type, as in `(emp x a)`. Another solution, used in SLIDE, is to infer the type of emp based on the context. For instance the type of the emp heap in `(sep (pto x a) emp)` is $\mathsf{U} \rightharpoonup \mathsf{Int}$. We leave this point for discussion.
2. It is usually expected that `nil` be of type Loc, however currently one can force other type that Loc, like in the previous example. Shall we impose a stricter type checking on `nil` ?

This heap typing restriction is not a limitation of the expressive power of the SMT-LIB encoding and can be easily overcome by using datatypes (available in SMT-LIB v2.5). Suppose, for instance that we would like to specify a heap consisting of cells containing both integer and boolean data. The idea is to declare a union type:

```
(declare-datatype BoolInt ((cons_bool (d Bool)) (cons_int (d Int))))
```

and use it to describe a mixed data heap, as in:

```
(sep (pto x (cons_bool false)) (pto y (cons_int 0)))
```

The same workaround can be used to specify heaps with mixed addresses, although this is a much less common situation in practice.

### 2.2 Separation Logic with Inductive Definitions

Let $\mathsf{Pred}$ be a set of second-order variables, each $P^{\sigma_1 \ldots \sigma_n} \in \mathsf{Pred}$ having an associated tuple of parameter sorts $\sigma_1, \ldots, \sigma_n \in \Sigma^{\mathrm{s}}$. In addition to the first-order terms built using variables from $\mathsf{Vars}$ and function symbols from $\Sigma^{\mathrm{f}}$, we enrich the language of $\mathsf{SL}$ with the boolean terms $P^{\sigma_1 \ldots \sigma_n}(t_1, \ldots, t_n)$, where each $t_i$ is a first-order term of sort $\sigma_i$, for $i = 1, \ldots, n$. Each second-order variable $P^{\sigma_1 \ldots \sigma_n} \in \mathsf{Pred}$ is provided with an inductive definition $P(x_1, \ldots, x_n) \leftarrow \phi_P(x_1, \ldots, x_n)$, where $\phi_P$ is a formula in the extended language, possibly containing occurrences of $P$. The satisfaction relation is then extended as follows:

$$\mathcal{I}, h \models_{\mathsf{SL}} P^{\sigma_1 \ldots \sigma_n}(t_1, \ldots, t_n) \iff \mathcal{I}, h \models_{\mathsf{SL}} \phi_P(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}})$$

where $\phi_P$ is the inductive definition of $P^{\sigma_1 \ldots \sigma_n}$. Observe that, given a set of inductive definitions, the set of possible models for each second-order variable is the least fixed point of a monotonic and continuous function mapping tuples of sets of models to a set of models.

### 2.3 SMT-LIB Encoding

An inductive definition $P(x_1, \ldots, x_n) \leftarrow \phi_P(x_1, \ldots, x_n)$ is written in SMT-LIB using a recursive function definition. For instance, the inductive definition of a doubly-linked list segment:

$$\begin{aligned} \mathsf{dllseg}(h, p, t, n) \leftarrow & (\mathsf{emp} \wedge h \approx n \wedge p \approx t) \vee \\ & (\exists x^{\mathsf{Loc}} \,.\, h \mapsto (x, p) * \mathsf{dllseg}(x, h, t, n)) \end{aligned}$$

is written into SMT-LIB as follows:

```
(declare-datatype Node ((node (next Loc) (prev Loc))))

(define-fun-rec dllseg ((h Loc) (p Loc) (t Loc) (n Loc)) Bool
    (or (and emp (= h n) (= p t))
        (exists ((x Loc)) (sep (pto h (node x p)) (dllseg x h t n)))
    )
)
```

### 2.4 A Detailed Example

Let us go through an example step by step. First of all, the preamble of and SMT-LIB file describing a $\mathsf{SL}$ satisfiability query must contain (at least):

```
(set-logic SEPLOG)
```

Since $\mathsf{SL}$ is used in combination with other theories, it is customary to start with:

```
(set-logic ALL_SUPPORTED)
```

We consider the slightly modified version of the dllseg definition above, which describes a doubly-linked list segment with ordered integer data:

$$\mathsf{dllseg}_{ord}(h, p, t, n, min) \leftarrow (\mathsf{emp} \wedge h \approx n \wedge p \approx t) \vee$$
$$(\exists x^{\mathsf{Loc}} \exists d^{\mathsf{Int}} . \ h \mapsto (d, x, min) * \mathsf{dllseg}_{ord}(x, h, t, n, d)) \wedge min \leq d$$

Since we do not perform any pointer arithmetic reasoning, we can declare Loc to be an uninterpreted sort:

```
(declare-sort U 0)
```

We encode the definition of $\mathsf{dllseg}_{ord}$ as:

```
(declare-datatype Node ((node (data Int) (next Loc) (prev Loc))))

(define-fun-rec dllseg_ord ((h Loc) (p Loc) (t Loc) (n Loc) (min Int)) Bool
    (or (and emp (= h n) (= p t))
        (exists ((x Loc) (d Int)) (and
                                    (sep (pto h (node x p)) (dllseg_ord x h t n))
                                    (<= min d)
                                   )
        )
    )
)
```

Let us consider the problem of proving that a $\mathsf{dllseg}_{ord}$ to which a node is appended is again a $\mathsf{dllseg}_{ord}$, provided that the data of the new node it smaller than the minimal element of the first $\mathsf{dllseg}_{ord}$:

$$x \mapsto (m, u, v) * \mathsf{dllseg}_{ord}(u, x, z, t, n) \wedge m \leq n \models_{\mathsf{SL}} \mathsf{dllseg}_{ord}(x, y, z, t, m)$$

We encode this entailment problem as an assertion asking whether the negated problem is satisfiable:

```
(declare-const x U)
(declare-const y U)
(declare-const z U)
(declare-const u U)
(declare-const v U)
(declare-const t U)
(declare-const m Int)
(declare-const n Int)

(assert (not (implies
              (and (sep (pto x (node m u v)) (dllseg_ord u x z t n)) (<= m n))
              (dllseg_ord x y z t m)
             )
        )
)
```

The entailment holds when the assertion is unsatisfiable, which can be checked in the standard way, using `(check-sat)`. However, the dual problem:

```
(assert (not (implies
             (dllseg_ord x y z t m)
             (and (sep (pto x (node m u v)) (dllseg_ord u x z t n)) (<= m n))
            )
       )
)
```

is satisfiable, and the counter-model can be obtained in the standard way, using `(get-model)`. Observe that the model of a satisfiable SL query consists of an interpretation of the constants and a specification of the heap.

## 3 Abduction and Frame Inference

Abduction and frame inference (or bi-abduction for both) are problems that occur in the context of program verification. In this case, the solver is not only required to give a yes/no answer to a satisfiability query, but to infer SL formulae that ensure the validity of a given entailment. Given SL formulae $\varphi(\mathbf{x})$ and $\phi(\mathbf{y})$, and second-order variables $X(\mathbf{x},\mathbf{y})$ and $Y(\mathbf{x},\mathbf{y})$, we consider the following synthesis problems:

1. The *abduction problem* asks for a satisfiable definition of a $X$ such that $\varphi(\mathbf{x}) * X(\mathbf{x},\mathbf{y}) \models_{\mathsf{SL}} \psi(\mathbf{y})$. Sometimes $X$ is called an *anti-frame*. Observe that $X \leftarrow \bot$ is always a solution, but not a very interesting one.
2. The *frame inference problem* asks for a definition of $Y$ such that $\varphi(\mathbf{x}) \models_{\mathsf{SL}} \exists \mathbf{z} . \psi(\mathbf{y}) * Y(\mathbf{x},\mathbf{y})$, where $\mathbf{z} = \mathbf{y} \setminus \mathbf{x}$.
3. The *bi-abduction problem* asks for both a satisfiable definition of $X$ and a definition of $Y$ such that $\varphi(\mathbf{x}) * X(\mathbf{x},\mathbf{y}) \models_{\mathsf{SL}} \psi(\mathbf{y}) * Y(\mathbf{x},\mathbf{y})$.

The capability of solving the above problems is key to using a given SL solver for practical program verification purposes. For this reason, we aim at finding a standard way of specifying these problems in SMT-LIB.

## 4 Additional Resources

The quest for a suitable format for SL solvers started with SL-COMP'14 [7], which adopted the `QF_S` format, described in [6]. The current proposal is inspired by `QF_S`, and relies on the datatypes introduced SMT-LIB v2.5 for an elegant treatment of union and record types. The tools supporting SMT-LIB as a native language are:

- CVC4 [3] – a description of the SL format of CVC4 is provided in [2] (a slightly modified version of the current proposal)
- SLIDE (under construction) – uses the encoding from the current proposal.
- SPEN [1] – a description of the SL format of SPEN (`QF_S`) is available in [6].

Other tools that participated to SL-COMP'14 have been adapted to `QF_S` by means of a specialized front-end [5]. It is our goal to convince the developpers of SL solvers to adopt SMT-LIB as the native input language of their tools, rather than use a translator from SMT-LIB. For this purpose, we provide a C++ front-end [4] that can be used to parse and type check SL inputs encoded in SMT-LIB using the current specification.

## References


1. Constantin Enea, Mihaela Sighireanu, and Zhilin Wu. On automated lemma generation for separation logic with inductive definitions. In *ATVA 2015, Proceedings*, pages 80–96, 2015.
2. Andrew Reynolds. Cvc4 separation logic format, 2016.
URL: `http://church.cims.nyu.edu/wiki/Separation_Logic`.
3. Andrew Reynolds, Radu Iosif, Cristina Serban, and Tim King. A decision procedure for separation logic in SMT. In *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*, pages 244–261, 2016.
4. Cristina Serban. Smt-lib front end, 2016.
URL: `https://github.com/cristina-serban/parse-smtlib`.
5. Mihaela Sighireanu. `QF_S` front end, 2014.
URL: `https://github.com/mihasighi/smtcomp14-sl/tree/master/smtlib2parser-1.4`.
6. Mihaela Sighireanu. The `qf_s` logic, 2014.
URL: `https://github.com/mihasighi/smtcomp14-sl/wiki`.
7. Mihaela Sighireanu and David Cok. Report on sl-comp 2014. *Journal on Satisfiability, Boolean Modeling and Computation*, 1, 2014.