A Formalization of Separation Logic in SMT-LIB v2.5
Syntax, Types and Semantics

Cristina Serban

Verimag

February 27, 2015
Eliminating the Space sort

- The Space sort is replaced by the Bool sort in the typing
- The recursive definitions become boolean predicates
- The casting operators tobool and tospace are eliminated

```
(define-fun btree ((?root Btree_t)) Space
 (tospace (or (and (= ?root nil) (tobool emp))
              (exists ((?X Btree_t) (?Y Btree_t))
                (and (distinct ?root nil)
                     (tobool (ssep
                          (pto ?root (sref (ref lson ?Y) (ref rson ?Z)))
                          (btree ?X) (btree ?Y))
                     )
                )
             )
          )
       )
)
```
Eliminating the Space sort

- The Space sort is replaced by the Bool sort in the typing
- The recursive definitions become boolean predicates
- The casting operators tobool and tospace are eliminated

```
(define-fun-rec btree ((?root Btree_t)) Bool
    (or (and (= ?root nil) emp)
        (exists ((?X Btree_t) (?Y Btree_t))
            (and (distinct ?root nil)
                (ssep
                    (pto ?root (sref (ref lson ?Y) (ref rson ?Z)))
                    (btree ?X) (btree ?Y)))
                )
            )
        )
    )
)
```
Introducing the \texttt{Ref} sort

- Consider that user declared sorts correspond to record types
- Define a parametrized sort \texttt{Ref} for a reference to a record
  
  \begin{verbatim}
  (declare-sort Ref 1)
  \end{verbatim}

- \texttt{Void} denotes a reference to any user record

\begin{verbatim}
(declare-sort Btree_t 0)
(declare-fun root () Btree_t)
\end{verbatim}
Introducing the Ref sort

- Consider that user declared sorts correspond to record types
- Define a parametrized sort Ref for a reference to a record
  
  \[(\text{declare-sort} \ \text{Ref} \ 1)\]

- Void denotes a reference to any user record
  
  \[(\text{declare-sort} \ \text{Btree}_t \ 0)\]
  \[(\text{declare-fun} \ \text{root} () (\text{Ref} \ \text{Btree}_t))\]
Typing of heap nodes

- Using selector functions

(declare-sort Btree_t 0)
(define-sort Btree_ref () (Ref Btree_t))

(declare-fun data () (Field Btree_ref Int))
(declare-fun lson () (Field Btree_ref Btree_ref))
(declare-fun rson () (Field Btree_ref Btree_ref))

(define-fun-rec btree((?root Btree_ref)) Bool
  (or (and (= ?root nil) emp)
      (exists ((?d Int) (?left Btree_ref) (?right Btree_ref))
          (and (distinct ?root nil)
               (ssep (pto ?root
                   (sref (ref data ?d)
                       (ref lson ?left)
                       (ref rson ?right)))
               (btree ?left) (btree ?right))))))
Typing of heap nodes

- Using constructor functions

```lisp
(declare-sort Btree_t 0)
(define-sort Btree_ref () (Ref Btree_t))

(declare-fun Btree (Int Btree_ref Btree_ref) Btree_ref)

(define-fun-rec btree((?root Btree_ref)) Bool
  (or (and (= ?root nil) emp)
      (exists ((?d Int) (?left Btree_ref) (?right Btree_ref))
        (and (distinct ?root nil)
             (ssep (pto ?root (Btree ?d ?left ?right))
                   (btree ?left) (btree ?right))))))
```

**Problem:** When you call a constructor twice with the same parameters, you should get a different result - contrary to the definition of a function ($\forall x. f(x) = f(x)$).
Typing of heap nodes

- Using datatypes, in a similar manner as Z3 or CVC4

**Syntax**

(declare-datatypes (T1 ... Tj)
  ((D1 (C1 (S1 [TYPE_1]) ... (Sn [TYPE_n]))) ... (Cn ...) ))
  ...
  (Dm (....) ... (....)))

  T1 ... Tj  parameters
  D1 ... Dm  datatypes
  C1 ... Cn  constructors for datatype D1
  S1 ... Sn  selectors for constructor C1
  TYPE_1 ... TYPE_n  previously declared types or one of D1 ... Dm
Typing of heap nodes

- Using datatypes, in a similar manner as Z3 or CVC4

```
(declare-datatypes (T) ((Lst nil (cons (hd T) (tl Lst)))))
(declare-const l1 (Lst Int))
(declare-const l2 (Lst Int))
(declare-const l3 (Lst Int))
(declare-const x Int)

(assert (= l3 (cons 0 (cons 1 nil))))
(assert (= l2 (cons x l3)))
(assert (= l1 l2))
(check-sat) ; sat: x = 2, l3 = (0,1), l2 = (2,0,1), l1 = (2,0,1)

(assert (= 0 (hd (tl l1))))
(check-sat) ; sat

(assert (not (= x (hd l1))))
(check-sat) ; unsat
```
Typing of heap nodes

- Using datatypes, in a similar manner as Z3 or CVC4

```
(declare-datatypes ()
  ((Btree_t nil
    (cons (data Int)
      (lson (Ref Btree_t))
      (rson (Ref Btree_t)))))))

(define-sort Btree_ref (Ref Btree_t))

(define-fun-rec btree((?root Btree_ref)) Bool
  (or (and (= ?root nil) emp)
    (exists ((?d Int) (?left Btree_ref) (?right Btree_ref)
      (and (distinct ?root nil)
        (ssep (pto ?root (cons ?d ?left ?right))
          (btree ?left) (btree ?right))))))))
```

- Whenever a new datatype is declared, a pto function is created for that datatype, in a manner similar to = for new sorts
References