

# Funcons-beta: Sets \*

The P<sub>L</sub>anCompS Project

Sets.cbs | PLAIN | PRETTY

---

## Sets

[ *Type*   **sets**  
*Funcon*   **set**  
*Funcon*   **set-elements**  
*Funcon*   **is-in-set**  
*Funcon*   **is-subset**  
*Funcon*   **set-insert**  
*Funcon*   **set-unite**  
*Funcon*   **set-intersect**  
*Funcon*   **set-difference**  
*Funcon*   **set-size**  
*Funcon*   **some-element**  
*Funcon*   **element-not-in** ]

*Meta-variables*    $GT <: \text{ground-values}$

*Built-in Type*   **sets**( $GT$ )

**sets**( $GT$ ) is the type of possibly-empty finite sets  $\{V_1, \dots, V_n\}$  where  $V_1 : GT, \dots, V_n : GT$ .

*Built-in Funcon*   **set**( $\_ : (GT)^*$ ) :  $\Rightarrow$  **sets**( $GT$ )

The notation  $\{V_1, \dots, V_n\}$  for **set**( $V_1, \dots, V_n$ ) is built-in.

*Assert*    $\{V^* : (GT)^*\} == \text{set}(V^*)$

Note that **set**( $\dots$ ) is not a constructor operation. The order and duplicates of argument values are ignored (e.g.,  $\{1, 2, 1\}$  denotes the same set as  $\{1, 2\}$  and  $\{2, 1\}$ ).

*Built-in Funcon*   **set-elements**( $\_ : \text{sets}(GT)$ ) :  $\Rightarrow (GT)^*$

For each set  $S$ , the sequence of values  $V^*$  returned by **set-elements**( $S$ ) contains each element of  $S$  just once. The order of the values in  $V^*$  is unspecified, and may vary between sets (e.g., **set-elements**  $\{1, 2\}$  could be  $(1, 2)$  and **set-elements**  $\{1, 2, 3\}$  could be  $(3, 2, 1)$ ).

---

\*Suggestions for improvement: [plancomps@gmail.com](mailto:plancomps@gmail.com).  
Reports of issues: <https://github.com/plancomps/CBS-beta/issues>.

*Assert* `set(set-elements( $S$ )) ==  $S$`

*Built-in Funcon* `is-in-set( $_$  :  $GT$ ,  $_$  : sets( $GT$ )) :  $\Rightarrow$  booleans`

`is-in-set( $GV$ ,  $S$ )` tests whether  $GV$  is in the set  $S$ .

*Assert* `is-in-set( $GV$  :  $GT$ , { }) == false`

*Assert* `is-in-set( $GV$  :  $GT$ , {  $GV$  } : sets( $GT$ )) == true`

*Built-in Funcon* `is-subset( $_$  : sets( $GT$ ),  $_$  : sets( $GT$ )) :  $\Rightarrow$  booleans`

`is-subset( $S_1$ ,  $S_2$ )` tests whether  $S_1$  is a subset of  $S_2$ .

*Assert* `is-subset({ },  $S$  : sets( $GT$ )) == true`

*Assert* `is-subset( $S$  : sets( $GT$ ),  $S$ ) == true`

*Built-in Funcon* `set-insert( $_$  :  $GT$ ,  $_$  : sets( $GT$ )) :  $\Rightarrow$  sets( $GT$ )`

`set-insert( $GV$ ,  $S$ )` returns the set union of  $\{GV\}$  and  $S$ .

*Assert* `is-in-set( $GV$  :  $GT$ , set-insert( $GV$  :  $GT$ ,  $S$  : sets( $GT$ ))) == true`

*Built-in Funcon* `set-unite( $_$  : (sets( $GT$ ))*) :  $\Rightarrow$  sets( $GT$ )`

`set-unite( $\dots$ )` unites a sequence of sets.

*Assert* `set-unite( $S$  : sets( $GT$ ),  $S$ ) ==  $S$`

*Assert* `set-unite( $S_1$  : sets( $GT$ ),  $S_2$  : sets( $GT$ )) == set-unite( $S_2$ ,  $S_1$ )`

*Assert* `set-unite( $S_1$  : sets( $GT$ ), set-unite( $S_2$  : sets( $GT$ ),  $S_3$  : sets( $GT$ )))`  
`== set-unite(set-unite( $S_1$ ,  $S_2$ ),  $S_3$ )`

*Assert* `set-unite( $S_1$  : sets( $GT$ ),  $S_2$  : sets( $GT$ ),  $S_3$  : sets( $GT$ ))`  
`== set-unite( $S_1$ , set-unite( $S_2$ ,  $S_3$ ))`

*Assert* `set-unite( $S$  : sets( $GT$ )) ==  $S$`

*Assert* `set-unite( ) == { }`

*Built-in Funcon* `set-intersect( $_$  : (sets( $GT$ ))+) :  $\Rightarrow$  sets( $GT$ )`

`set-intersect( $GT$ ,  $\dots$ )` intersects a non-empty sequence of sets.

*Assert* `set-intersect( $S$  : sets( $GT$ ),  $S$ ) ==  $S$`

*Assert* `set-intersect( $S_1$  : sets( $GT$ ),  $S_2$  : sets( $GT$ )) == set-intersect( $S_2$ ,  $S_1$ )`

*Assert* `set-intersect( $S_1$  : sets( $GT$ ), set-intersect( $S_2$  : sets( $GT$ ),  $S_3$  : sets( $GT$ )))`  
`== set-intersect(set-intersect( $S_1$ ,  $S_2$ ),  $S_3$ )`

*Assert* `set-intersect( $S_1$  : sets( $GT$ ),  $S_2$  : sets( $GT$ ),  $S_3$  : sets( $GT$ ))`  
`== set-intersect( $S_1$ , set-intersect( $S_2$ ,  $S_3$ ))`

*Assert* `set-intersect( $S$  : sets( $GT$ )) ==  $S$`

*Built-in Funcon* `set-difference`(`_` : `sets`(`GT`), `_` : `sets`(`GT`)) :  $\Rightarrow$  `sets`(`GT`)

`set-difference`( $S_1, S_2$ ) returns the set containing those elements of  $S_1$  that are not in  $S_2$ .

*Built-in Funcon* `set-size`(`_` : `sets`(`GT`)) :  $\Rightarrow$  `natural-numbers`

*Assert* `set-size`( $S$  : `sets`(`GT`)) == `length`(`set-elements`( $S$ ))

*Funcon* `some-element`(`_` : `sets`(`GT`)) :  $\Rightarrow$  `GT`?

*Assert* `some-element`( $S$  : `sets`(`GT`)) == `index`(1, `set-elements`( $S$ ))

*Assert* `some-element` { } == ( )

*Built-in Funcon* `element-not-in`(`GT` : `types`, `_` : `set`(`GT`)) :  $\Rightarrow$  `GT`?

`element-not-in`( $GT, S$ ) gives an element of the type  $GT$  not in the set  $S$ , or ( ) when  $S$  is empty. When the set of elements of  $GT$  is infinite, `element-not-in`( $GT, S$ ) never gives ( ).