Funcons-beta: Maps *

The PLanCompS Project

`Maps.cbs` | PLAIN | PRETTY

Maps

[*Type* maps
Funcon map
Funcon map-elements
Funcon map-lookup
Alias lookup
Funcon map-domain
Alias dom
Funcon map-override
Funcon map-unite
Funcon map-delete]

Meta-variables GT <: ground-values
$T^?$ <: values?

Built-in Type maps(GT, $T^?$)

maps(GT, $T^?$) is the type of possibly-empty finite maps from values of type GT to optional values of type $T^?$.

Built-in Funcon map(_ : (tuples(GT, $T^?$))*) : ⇒(maps(GT, $T^?$))?

map(tuple(K_1, ${V_1}^?$), ⋯ , tuple(K_n, ${V_n}^?$)) constructs a map from K_1 to ${V_1}^?$, . . . , K_n to ${V_n}^?$, provided that K_1, . . . , K_n are distinct, otherwise the result is ( ).

Note that map(⋯) is not a constructor operation.

The built-in notation $\{K_1 \mapsto {V_1}^?, \cdots, K_n \mapsto {V_n}^?\}$ is equivalent to map(tuple(K_1, ${V_1}^?$), ⋯ , tuple(K_n, ${V_n}^?$)). Note however that in general, maps cannot be identified with sets of tuples, since the values ${V_i}^?$ are not restricted to ground-values.

When $T^?$ <: types, maps(GT, $T^?$) <: types. The type MT : maps(GT, $T^?$) represents the set of value-maps MV : maps(GT, values?) such that dom(MV) is a subset of dom(MT) and for all K in dom(MV), map-lookup(MV, K) : map-lookup(MT, K).

Built-in Funcon map-elements(_ : maps(GT, $T^?$)) : ⇒(tuples(GT, $T^?$))*

*Suggestions for improvement: `plancomps@gmail.com`.
Reports of issues: `https://github.com/plancomps/CBS-beta/issues`.

The sequence of tuples $(\text{tuple}(K_1, V_1{}^?), \cdots, \text{tuple}(K_n, V_n{}^?))$ given by map-elements(M) contains each mapped value K_i just once. The order of the elements is unspecified, and may vary between maps.

Assert map(map-elements(M)) == M

Built-in Funcon map-lookup($\_ : \text{maps}(GT, T^?), K : GT) : \Rightarrow (T^?)^?$

Alias lookup = map-lookup

map-lookup(M, K) gives the optional value to which K is mapped by M, if any, and otherwise ( ).

Built-in Funcon map-domain($\_ : \text{maps}(GT, T^?)) : \Rightarrow \text{sets}(GT)$

Alias dom = map-domain

map-domain(M) gives the set of values mapped by M.

map-lookup(M, K) is always ( ) when K is not in map-domain(M).

Built-in Funcon map-override($\_ : (\text{maps}(GT, T^?))^*) : \Rightarrow \text{maps}(GT, T^?)$

map-override$(\cdots)$ takes a sequence of maps. It returns the map whose domain is the union of their domains, and which maps each of those values to the same optional value as the first map in the sequence in whose domain it occurs . When the domains of the M^* are disjoint, map-override(M^*) is equivalent to map-unite(M^*).

Built-in Funcon map-unite($\_ : (\text{maps}(GT, T^?))^*) : \Rightarrow (\text{maps}(GT, T^?))^?$

map-unite$(\cdots)$ takes a sequence of maps. It returns the map whose domain is the union of their domains, and which maps each of those values to the same optional value as the map in the sequence in whose domain it occurs, provided that those domains are disjoint - otherwise the result is ( ).

Built-in Funcon map-delete($\_ : \text{maps}(GT, T^?), \_ : \text{sets}(GT)) : \Rightarrow \text{maps}(GT, T^?)$

map-delete(M, S) takes a map M and a set of values S, and returns the map obtained from M by removing S from its domain.

Assert map-domain(map-delete(M, S)) == set-difference(map-domain(M), S)