MODULE *Promises*

EXTENDS *Integers*, *Sequences*, *FiniteSets*

Objects, Heaps, and Methods

CONSTANT *Id*, *Value*

A value represents something like an int or a boolean. An *Id* is the heap address of an object. We can represent *null* as some particular *Id*. We assume that an *Id* is not a *Value*.

ASSUME $Id \cap Value = \{\}$

Some TLA+ Notation

A function f has a domain written DOMAIN f . The function assigns a value $f[x]$ for every element x in DOMAIN f. The domain of a function can be an infinite set.

$[S \rightarrow T]$ is the set of all functions f with domain S such that $f[x]$ is in the set T for all x in S.

A record r is a function whose domain is a non-empty finite set of strings, where we can write $r.fldName$ as an abbreviation for $r[\text{"}fldName\text{"}]$.

$[foo \mapsto 42, \text{bar} \mapsto v]$ is the record r whose domain is $\{\text{"}foo\text{"}, \text{"}bar\text{"}\}$ such that $r.foo = 42$ and $r.bar = v$.

$[foo : Nat, \text{bar} : V]$ is the set of all records $[foo \mapsto n, \text{bar} \mapsto v]$ such that n is in Nat and v is in V.

$Object \triangleq$

An *Object* is a record containing a *type* field that is a string. The fields represent the fields of the object. I don't bother representing usual types and classes. The classic class structure can be represented by an *Object* having a *class* field whose value is the *id* of an *Object* of type "class". The fields of the latter *Object* would represent the static fields of the class. The representation of methods is described below.

We assume that a *Value* is not an *Object*.

$$
\begin{array}{ll}
\text{LET} & Rcd(Labels) \triangleq \{R \in [Labels \rightarrow Value \cup Id] : R.type \in \text{STRING}\} \\
& LabelSets \triangleq \{S \in \text{SUBSET STRING} : IsFiniteSet(S) \land \text{"type"} \in S\} \\
\text{IN} & \text{UNION } \{Rcd(Labels) : Labels \in LabelSets\}
\end{array}
$$

ASSUME $Object \cap Value = \{\}$

$Heap \triangleq [Id \rightarrow Object \cup \{\langle\rangle\}]$

A *Heap* maps an *Dd* either to an *Object* or to $\langle\rangle$, the latter meaning that the *Id* is not the *Id* of any object.

$ReachableFrom(obj, H) \triangleq$

This is the set of *Ids* reachable from an *Object* *obj* in the heap H. If *obj* is not an *Object*, it is the empty set. Otherwise, it consists of all the *Ids* that are values of fields of *obj*, together with all the *Ids* reachable from objects in heap H with those *Ids*. It is defined in terms of two operators *IdsOf* and R.

LET $IdsOf(o) \triangleq$

The set of all values of fields of object *obj* that are *Ids*, or the empty set if *obj* is not an object.

IF $o \in Object$
THEN $\{i \in \{o[x] : x \in \text{DOMAIN } o\} : i \in Id\}$
ELSE $\{\}$

$R[n \in Nat] \triangleq$

Defines $R[n]$ to be the set of Ids reachable from id by a path of length at most n in the heap H.

IF $n = 0$ THEN $IdsOf(obj)$
ELSE $R[n-1] \cup \{IdsOf(H[i]) : i \in R[n-1]\}$

IN UNION $\{R[n] : n \in Nat\}$

CONSTANT $Method$, $Eval(\_, \_, \_, \_)$

We assume that there are only static methods. A method specifies the result of executing a method of some object. We are considering the execution of a method to be an atomic action. The result of executing a method M of an object obj with a list $args$ of arguments $args$ when the value of the heap is H is $Eval(M, obj, args, H)$, which is a record consisting of two fields:

- A *result* field that equals the value returned by the method.
- A *heap* field that is the heap after the execution.

For convenience, we assume that a method is a value.

ASSUME $Method \subseteq Value$

Promises

$ResolvedPromise \triangleq$

$[type : \{\text{"promise"}\},$
$\ resolved : \{\text{TRUE}\},$
$\ value : Value \cup Id$
$]$

$UnresolvedPromise \triangleq$

A promise is resolved by executing a method to compute its value. There are two kinds of promises: *when* promises and the other kind. A *when* promise is returned by executing a *whenFulfilled* such as

$foo.whenFulfilled(arg \Rightarrow some\ code)$

The promise is specified by an object with these fields:

- A single-argument *Method*, which is dynamically created by executing the *whenFulfilled*. In the example, it is the *Method* that would be written in the class containing the expression as

  $newMethod(arg)\{some\ code\}$

- The *Id* of the object for which the *whenFulfilled* method was executed. In the example, references to *this* in *some code* refer to the fields of this object.
- A promise for method's the single argument. In the example, it is a promise for *foo*.

A non-*when* promise is one produced by executing something like

$foo.Bar(arg1, \ldots, argN)$

The promise is specified by an object with these fields:

- A *Method* that represents the Bar method of the appropriate class.

- A promise for an *Id* of the object *foo*.

- The argument list.

Because a promise is an *Object*, promises can appear just like any other object in the heap. They can also be used as arguments to methods.

$$
\begin{array}{l}
[\mathit{type} \quad : \{\text{“promise”}\}, \\
\ \mathit{resolved} \quad : \{\text{FALSE}\}, \\
\ \mathit{method} \quad : \mathit{Method}, \\
\ \mathit{isWhen} \quad : \{\text{TRUE}\}, \\
\ \mathit{objId} \quad : \mathit{Id}, \\
\ \mathit{argPromise} : \mathit{Id} \\
]
\end{array}
$$

$$\cup$$

$$
\begin{array}{l}
[\mathit{type} \quad : \{\text{“promise”}\}, \\
\ \mathit{resolved} \quad : \{\text{FALSE}\}, \\
\ \mathit{method} \quad : \mathit{Method}, \\
\ \mathit{isWhen} \quad : \{\text{FALSE}\}, \\
\ \mathit{objIdPromise} : \mathit{Id}, \\
\ \mathit{args} \quad : \mathit{Seq}(\mathit{Value} \cup \mathit{Id}) \\
]
\end{array}
$$

$$\mathit{Promise} \triangleq \mathit{ResolvedPromise} \cup \mathit{UnresolvedPromise}$$

$$\mathit{OKObject} \triangleq \{o \in \mathit{Object} : o.\mathit{type} = \text{“promise”} \Rightarrow o \in \mathit{Promise}\}$$

The set *Object* minus those objects of type “promise” that are not in the set *Promise*.

$$\mathit{OKHeap} \triangleq$$

The set of Heaps such that:

- There are no dangling pointers (fields of objects that are *Ids* that point to nothing).

- Every *Id* in that should be the *Id* of a promise is.

- There are no cycles of promises

$$
\begin{array}{l}
\{H \in \mathit{Heap} : \\
\quad \forall \mathit{id} \in \mathit{Id} : \\
\quad\quad \text{LET } \mathit{obj} \triangleq H[\mathit{id}] \\
\quad\quad \text{IN } \quad (\mathit{obj} \neq \langle\rangle) \Rightarrow \\
\quad\quad\quad\quad \land\ \mathit{obj} \in \mathit{OKObject} \\
\quad\quad\quad\quad \land\ \forall \mathit{fldName} \in \text{DOMAIN } \mathit{obj} : \\
\quad\quad\quad\quad\quad (\mathit{obj}[\mathit{fldName}] \in \mathit{Id}) \Rightarrow (H[\mathit{obj}[\mathit{fldName}]] \neq \langle\rangle) \\
\quad\quad\quad\quad \land\ (\mathit{obj} \in \mathit{Promise}) \land (\neg \mathit{obj}.\mathit{resolved}) \Rightarrow \\
\quad\quad\quad\quad\quad\quad \land\ \text{IF } \mathit{obj}.\mathit{isWhen} \text{ THEN } H[\mathit{obj}.\mathit{argPromise}] \in \mathit{Promise} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ELSE } H[\mathit{obj}.\mathit{objIdPromise}] \in \mathit{Promise} \\
\quad\quad\quad\quad\quad\quad \land\ \text{LET } \mathit{thePromise}(i) \triangleq
\end{array}
$$

$$
\begin{array}{l}
\quad\quad\quad\quad \text{IF } H[i].resolved \\
\quad\quad\quad\quad\quad \text{THEN } i \\
\quad\quad\quad\quad\quad \text{ELSE IF } H[i].isWhen \text{ THEN } H[i].argPromise \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ELSE } H[i].objIdPromise \\
\quad\quad\quad R[n \in Nat] \triangleq \\
\quad\quad\quad\quad \text{IF } n = 0 \text{ THEN } id \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ELSE } thePromise(R[n-1]) \\
\quad\quad \text{IN } \quad \forall n \in Nat \setminus \{0\} : R[n] \neq id \quad\quad\quad\quad \}
\end{array}
$$

$ReachableWithoutPromises(obj, H) \triangleq$

Like *ReachableFrom*, except that it does not follow links inside objects of type "promise", which as described above represent promises.

$$
\begin{array}{l}
\text{LET } IdsOf(o) \triangleq
\end{array}
$$

The set of all values of fields of object *obj* that are *Ids*, or the empty set if *obj* is not an object.

$$
\begin{array}{l}
\quad\quad \text{IF } (o \in Object) \land (o.type \neq \text{"promise"}) \\
\quad\quad\quad \text{THEN } \{i \in \{o[x] : x \in \text{DOMAIN } o\} : i \in Id\} \\
\quad\quad\quad \text{ELSE } \{\} \\
\quad R[n \in Nat] \triangleq
\end{array}
$$

Defines $R[n]$ to be the set of *Ids* reachable from *id* by a path of length at most n in the heap H.

$$
\begin{array}{l}
\quad\quad \text{IF } n = 0 \text{ THEN } IdsOf(obj) \\
\quad\quad\quad\quad\quad\quad \text{ELSE } R[n-1] \cup \{IdsOf(H[i]) : i \in R[n-1]\} \\
\text{IN } \quad \text{UNION } \{R[n] : n \in Nat\}
\end{array}
$$

$$
\begin{array}{l}
\text{ASSUME } \forall M \in Method, \\
\quad\quad\quad obj \in OKObject, \\
\quad\quad\quad args \in Seq(Value \cup Id) : \\
\quad\quad\quad \text{LET } E(H) \triangleq Eval(M, obj, args, H) \\
\quad\quad\quad\quad\quad UseableId(H) \triangleq ReachableWithoutPromises(obj, H) \cup \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{UNION } \{ReachableWithoutPromises(args[i], H) : \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad i \in 1 \,..\, Len(args)\} \\
\quad\quad\quad \text{IN } \quad \land \forall H \in OKHeap : \\
\quad\quad\quad\quad\quad\quad\quad \land E(H) \in [result : Value \cup Id, heap : OKHeap]
\end{array}
$$

For convenience, we assume that evaluating method M produces some result and new heap for arbitrary *OKObject obj*, argument list *args*, and heap H even when they are meaningless–for example, if the result of executing M depends on the values of fields of *obj* that do not exist.

$$
\begin{array}{l}
\quad\quad\quad\quad\quad\quad\quad \land \forall id \in Id : \\
\quad\quad\quad\quad\quad\quad\quad\quad (H[id] \neq E(H).heap[id]) \Rightarrow \lor id \in UseableId(H) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \lor H[id] = \langle\rangle
\end{array}
$$

Evaluating M can modify the heap only by modifying objects whose *Id* is reachable from *obj* or an argument and by creating new objects.

$$
\begin{array}{l}
\quad\quad\quad\quad\quad\quad \land \forall H1, H2 \in OKHeap : \\
\quad\quad\quad\quad\quad\quad\quad \land UseableId(H1) = UseableId(H2)
\end{array}
$$

$$
\begin{array}{l}
\land \forall id \in \mathit{UseableId}(H1) : H1[id] = H2[id] \\
\Rightarrow \quad \land E(H1).\mathit{result} = E(H2).\mathit{result} \\
\qquad \land \forall id \in \mathit{Id} : (H1[id] \neq E(H1).\mathit{heap}[id]) \Rightarrow \\
\qquad\qquad\qquad (E(H1).\mathit{heap}[id] = E(H2).\mathit{heap}[id])
\end{array}
$$

If two heaps are the same on the *Ids* reachable from *obj* and the arguments, then evaluating M in the two heaps produces the same result, and it produces the same changes to the two heaps.

Actions

VARIABLE *heap* — The variable whose value is the current *Heap*.

$$
\begin{array}{l}
\mathit{ResolveWhenPromise}(id) \triangleq
\end{array}
$$

The action that modifies the heap by resolving a *when* promise when the promise it is waiting for has been resolved.

$$
\begin{array}{l}
\text{LET } \mathit{promise} \triangleq \mathit{heap}[id] \\
\text{IN} \quad \land \mathit{promise} \in \mathit{Promise} \\
\qquad \land \neg \mathit{promise.resolved} \\
\qquad \land \mathit{promise.isWhen} \\
\qquad \land \mathit{heap}[\mathit{promise.argPromise}].\mathit{resolved} \\
\qquad \land \mathit{heap}' = \\
\qquad\quad \text{LET } E \triangleq \mathit{Eval}(\mathit{promise.method}, \\
\qquad\qquad\qquad \mathit{heap}[\mathit{promise.objId}], \\
\qquad\qquad\qquad \langle \mathit{heap}[\mathit{promise.argPromise}].\mathit{value} \rangle, \\
\qquad\qquad\qquad \mathit{heap}) \\
\qquad\quad \text{IN} \quad [E.\mathit{heap} \text{ EXCEPT } ![id] = [\mathit{type} \mapsto \text{“promise”}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{resolved} \mapsto \text{TRUE}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{value} \mapsto E.\mathit{result}]]
\end{array}
$$

$$
\mathit{ResolveNonWhenPromise}(id) \triangleq
$$

The action that modifies the heap by resolving a non-*when* promise when the promise it is waiting for has been resolved.

$$
\begin{array}{l}
\text{LET } \mathit{promise} \triangleq \mathit{heap}[id] \\
\text{IN} \quad \land \mathit{promise} \in \mathit{Promise} \\
\qquad \land \neg \mathit{promise.resolved} \\
\qquad \land \neg \mathit{promise.isWhen} \\
\qquad \land \mathit{heap}[\mathit{promise.objIdPromise}].\mathit{resolved} \\
\qquad \land \mathit{heap}' = \\
\qquad\quad \text{LET } E \triangleq \mathit{Eval}(\mathit{promise.method}, \\
\qquad\qquad\qquad \mathit{heap}[\mathit{heap}[\mathit{promise.objIdPromise}].\mathit{value}], \\
\qquad\qquad\qquad \mathit{promise.args}, \\
\qquad\qquad\qquad \mathit{heap}) \\
\qquad\quad \text{IN} \quad [E.\mathit{heap} \text{ EXCEPT } ![id] = [\mathit{type} \mapsto \text{“promise”}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{resolved} \mapsto \text{TRUE}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{value} \mapsto E.\mathit{result}]]
\end{array}
$$

$$
\mathit{GarbageCollectResolvedPromise}(id) \triangleq
$$

The action that garbage collects a promise that has been resolved and whose value is no longer usable.

$$
\begin{aligned}
&\land\ heap[id] \in Promise \\
&\land\ heap[id].resolved \\
&\land\ \forall\, i \in Id \setminus \{id\} : \\
&\qquad (heap[i] \neq \langle\rangle) \Rightarrow \forall\, fldName \in \text{DOMAIN } i : \\
&\qquad\qquad\qquad\qquad\qquad (heap[i][fldName] \in Id) \Rightarrow \\
&\qquad\qquad\qquad\qquad\qquad\quad (heap[heap[i][fldName]] \neq id) \\
&\land\ heap' = [heap \text{ EXCEPT } ![id] = \langle\rangle]
\end{aligned}
$$

* Modification History

* Last modified Sun *Mar* 06 11:28:30 *PST* 2011 by *lamport*

* Created Sat *Mar* 05 13:24:42 *PST* 2011 by *lamport*