Progress for Memory Management of a Concurrent Data Object

Wim H. Hesselink, 29th September 1998
Dept. of Mathematics and Computing Science, University of Groningen
Postbox 800, 9700 AV Groningen, The Netherlands
Email: `wim@cs.rug.nl`. Fax +31.50.3633800. Web site:
`http://www.cs.rug.nl/~wim` .

1 Introduction

This paper complements the papers [Hes98, Hes99]. The reader is supposed to have these available. These papers present a method for formally proving progress for a distributed system with a weak form of fault tolerance, together with a nontrivial application of this method. Here we conclude the treatment of the application by showing that memory management makes enough progress.

Our specific application is a system where a number of client processes concurrently and repeatedly inspect and modify the value of a concurrent data object. These client actions are called invocations. The invocations are treated by four families of server processes: the families *Lin* of linearizers, *App* of appliers, *Coll* of collectors, and *Distr* of distributors.

The invocations of the clients are stored in a space of invocations. The invocation addresses are managed by the collectors. The linearizers form a queue of pending invocations, and the appliers treat the invocations by forming a new state of the data-object. These states are stored and retrieved in a second address space, said to consist of locations. The locations are managed by the distributors.

The processes in *Coll* (collectors) and *Distr* (distributors) are specified as follows. They have to supply the clients and the appliers with free addresses and locations, respectively, as formalized in the progress assertions

(1) $\{Coll\}: \ true \quad \circ\!\!\rightarrow \quad iloc.q \neq 0 \ , \ \text{ for all } q \in Cli \ ,$

(2) $\{Coll, Distr\}: \ true \quad \circ\!\!\rightarrow \quad staloc.q \neq 0 \ , \ \text{ for all } q \in App \ .$

Recall from [Hes99] that $L : P \circ\!\!\rightarrow Q$ (pronounced "P leads to Q") expresses, roughly speaking, that every execution that starts in a state where P holds contains a state where Q holds, provided every $U \in L$ is active enough, where U is a set of processes and an action of U is an action of any of its processes.

On the other hand, collectors and distributors must preserve all invariants for *Cli*, *Lin*, *App*, described in [Hes98]. In the mechanical proof of [Hes@], we have verified this for the programs for *Coll* and *Distr* of [Hes98].

It is easy to see that progress cannot be guaranteed if the sets Ad and Lo are too small in comparison with the sets of processes. Therefore, we have to assume that the sets Ad and Lo are large enough. Below we derive lower bounds for the sizes of Ad and Lo for which the implementations of collectors and distributors satisfy the requirements (1) and (2).

2 Sharing an address space

In this Section we extend the proof rules of [Hes99] with a trivial consequence of them and two nontrivial applications.

Corollary 2 (cancellation). Assume that $L : P \circ\!\!\rightarrow Q \vee R$ within k and $L : R \circ\!\!\rightarrow S$ within m. Then $L : P \circ\!\!\rightarrow Q \vee S$ within $k + m$.

Proof. The assumption $L : R \circ\!\!\rightarrow S$ within m implies $L : Q \vee R \circ\!\!\rightarrow Q \vee S$ within m by the Rules 0, 2, and 3. Then we use Rule 5. □

If one has fault tolerant processes that communicate by means of shared memory, the use of fixed channels easily leads to memory requirements quadratic in the number of processes. We aim at linear memory requirements and therefore let every address be potentially available to every process.

Let A be a finite set of addresses in the shared memory space. We use a function $next \in A \rightarrow A$ for a process to move from address to address. We say that function $next$ traverses A iff, for every pair x, $y \in A$, there is a number k with $next^k(x) = y$. It follows that, after $\#A$ applications of $next$, the process is back at the starting address.

If processes have to treat addresses under mutual exclusion, it is important to avoid congestion by letting them use *different* traversing functions, cf. [Hes96]. In the following we therefore parametrize the traversing functions by process names, and thus use traversing functions $next(q, ?)$ where q is a process name and "?" is a place holder for an argument in A that has to be supplied later.

Let $lock$ be a shared array of booleans $lock.k$ with k ranging over A. Assume that U is a subset of the set of processes, that the processes $q \in U$ have private variables y of type A, and that each process $q \in U$ has a program of the form

a	$y := next(self, y)$;
$a+1$	**if** $lock.y$ **then goto** a **else** $lock.y := true$ **fi** ;
...	*other instructions* ;
b	$lock.y := false$; **goto** a .

Here, $self$ refers to the acting process and function $next(q, ?)$ traverses A for every process q. The array $lock$ consists of boolean consensus variables. It is not modified elsewhere or by processes other than those in U. We write $y.q$ for the value of y for process q and define

$$q \textbf{ at } k \quad \equiv \quad a + 1 < pc.q \leq b \quad \wedge \quad y.q = k \ .$$

Process q is said to *visit* address k if it performs an action that establishes q **at** k. We postulate the invariant

(Iq) $\qquad lock.k \quad \Rightarrow \quad (\exists\, q : q \in U : q \textbf{ at } k)$.

Now consider two predicates P and Q. We want to infer $\{U\} : P \circ\!\!\rightarrow Q$. For this purpose, we assume state functions vf and Hot to be given, where vf yields a natural number and Hot yields a set of addresses. The idea is that vf can only increase when Q is established, and that vf decreases when an address in Hot is visited by a process from U.

We define the set of the unlocked addresses $Free = \{k \mid \neg lock.k\}$.

Theorem A. Assume the invariant $\#U \leq \#Hot$, and assume for every $m \in \mathbb{N}$ and $k \in A$ the stepping relations

$$
\begin{array}{ll}
\text{(a)} & P \quad \rhd \quad Q \;\vee\; P \;; \\
\text{(b)} & P \;\wedge\; vf \leq m \quad \rhd \quad Q \;\vee\; vf \leq m \;; \\
\text{(c)} & P \;\wedge\; vf \leq m \;\wedge\; k \in Hot \cap Free \\
 & \quad \rhd \quad Q \;\vee\; vf < m \;\vee\; k \in Hot \cap Free \;.
\end{array}
$$

Assume that every process $q \in U$ satisfies, for every $m \in \mathbb{N}$,

$$\text{(d)} \qquad \{\{q\}\} : P \;\wedge\; vf \leq m \quad o\!\!\rightarrow \quad Q \;\vee\; vf < m \;\vee\; pc.q = a \;.$$

Then, for every $m \in \mathbb{N}$, we have $\{U\} : P \;\wedge\; vf \leq m \quad o\!\!\rightarrow \quad Q$.

Proof. We define predicates $P.m : P \wedge vf \leq m$ and $Q.m : Q \vee (P \wedge vf < m)$. By Rule 4 and Corollary 1, it suffices to prove $\{\{q\}\} : P.m \; o\!\!\rightarrow Q.m$ for every $q \in U$ and every natural number m. So we can fix q and m.

Conjoining (a) and (b), and (a) and (c), and weakening, we obtain

$$
\begin{array}{ll}
\text{(b')} & P.m \quad \rhd \quad Q.m \;\vee\; P.m \;; \\
\text{(c')} & P.m \;\wedge\; k \in Hot \cap Free \quad \rhd \quad Q.m \;\vee\; k \in Hot \cap Free \;.
\end{array}
$$

Using (d) and the PSP rule with $R := P$ and $S := Q$ and condition (a), and weakening, we get

$$\text{(d')} \qquad \{\{q\}\} : P.m \quad o\!\!\rightarrow \quad Q.m \;\vee\; pc.q = a \;.$$

Let $k \in A$ be an arbitrary address. We define predicate $G : \; pc.q = a + 1 \;\wedge\; y.q = k$. Our first aim is to prove

$$(*) \qquad \{\{q\}\} : P.m \quad o\!\!\rightarrow \quad G \;\vee\; Q.m \;.$$

To approach predicate G, we use that function $next(q, ?)$ traverses the finite set A. Indeed, this enables us to define a private ghost variable uf for process q with the invariant $0 \leq uf < \#A$, which only increases when G is established and which decreases when q executes instruction a without establishing G. So it satisfies

$$
\begin{array}{ll}
\text{(u0)} & uf \leq r \quad \rhd \quad G \;\vee\; uf \leq r \;; \\
\text{(u1)} & \{q\} : pc.q = a \;\wedge\; uf \leq r \quad \rhd \quad G \;\vee\; uf < r \;.
\end{array}
$$

Using (u0) and weakening, the PSP rule with $R := uf \leq r$ and $S := G$ sharpens (d') to

$$\{\{q\}\} : P.m \;\wedge\; uf \leq r \quad o\!\!\rightarrow \quad G \;\vee\; Q.m \;\vee\; (pc.q = a \;\wedge\; uf \leq r) \;.$$

On the other hand, since pc and uf are private variables of q, Rule 1 with (u1) yields

$$\{\{q\}\} : pc.q = a \;\wedge\; uf \leq r \quad o\!\!\rightarrow \quad G \;\vee\; uf < r \;.$$

Using cancellation (Corollary 2), we thus get

$$\{\{q\}\} : P.m \;\wedge\; uf \le r \quad \mathrm{o}\!\!\rightarrow \quad G \;\vee\; Q.m \;\vee\; uf < r \; .$$

The PSP-rule with $R := P.m$ and $S := Q.m$ together with (b') now sharpens this to

$$\{\{q\}\} : P.m \;\wedge\; uf \le r \quad \mathrm{o}\!\!\rightarrow \quad G \;\vee\; Q.m \;\vee\; (P.m \wedge uf < r) \; .$$

Since uf is bounded, Corollary 1 applied with uf for vf implies formula (*).

Application of Rule 1 easily yields $\{\{q\}\} : G \;\mathrm{o}\!\!\rightarrow\; k \notin Free$ within 1. By Corollary 2, formula (*) therefore implies

$$\{\{q\}\} : P.m \quad \mathrm{o}\!\!\rightarrow \quad Q.m \;\vee\; k \notin Free \; .$$

This formula is sharpened by means of the PSP-rule with $R := P.m \wedge k \in Hot \cap Free$ and $S := Q.m$ and the conditions (b') and (c') to

$$\{\{q\}\} : P.m \;\wedge\; k \in Hot \cap Free \quad \mathrm{o}\!\!\rightarrow \quad Q.m \; .$$

Since the set A is finite, this implies by Rule 3 (disjunction) that

$$\{\{q\}\} : P.m \;\wedge\; Hot \cap Free \neq \emptyset \quad \mathrm{o}\!\!\rightarrow \quad Q.m \; .$$

On the other hand, $pc.q = a$ implies by invariant (Iq) that the number of locked addresses is less than $\#U$, and hence less than $\#Hot$. Therefore $pc.q = a$ implies that $Hot \cap Free$ is nonempty. By conditions (b') and (d'), the PSP-rule thus yields

$$\{\{q\}\} : P.m \quad \mathrm{o}\!\!\rightarrow \quad Q.m \;\vee\; (P.m \;\wedge\; Hot \cap Free \neq \emptyset) \; .$$

Finally, using Corollary 0, we get $\{\{q\}\} : P.m \;\mathrm{o}\!\!\rightarrow\; Q.m$, as required. □

Remark. In practice, one usually first invents a definition of Hot such that a visit of a process in U to an address in Hot is somehow productive for the goal Q. One then invents a variant function that is decremented **while** a process from U visits Hot. Finally, one modifies vf by "immediate depreciation" to satisfy condition (c). □

Surprisingly, we also need a heterogeneous version of Theorem A. So we consider two disjoint sets of addresses A_0 and A_1, two arrays *lock0* and *lock1*, and two disjoint subsets U_0 and U_1 of the set of processes. The processes from U_0 (and U_1) treat the addresses from A_0 and array *lock0* (and A_1 and *lock1*, respectively) in the same way as the processes from U treat A and *lock* above.

Instead of subsets Hot and $Free$ of A, we now use subsets H_0 and F_0 of A_0, and H_1 and F_1 of A_1. We use one variant function vf and claim

Theorem B. Assume the invariant $\#U_0 + \#U_1 \le \#H_0 + \#H_1$. Assume the stepping relations (a), (b) as in Theorem A, and for both $i = 0$ and $i = 1$:

(c$_i$)
$$P \;\wedge\; vf \le m \;\wedge\; k \in H_i \cap F_i$$
$$\rhd \quad Q \;\vee\; vf < m \;\vee\; k \in H_i \cap F_i \; .$$

Assume that all processes $q \in U_0 \cup U_1$ satisfy condition (d) of Theorem A.

Then we have $\{U_0, U_1\} : P \;\wedge\; vf \le m \quad \mathrm{o}\!\!\rightarrow \quad Q$.

Proof. We use the predicates $P.m$ and $Q.m$ from the proof of Theorem A. It now suffices to prove that $\{U_0, U_1\} : P.m \ \mathrm{o}\!\!\rightarrow Q.m$. Just as in the proof of Theorem A, we get that, for $i \in \{0, 1\}$ and $q \in U_i$,

$$\{\{q\}\} : P.m \quad \wedge \quad H_i \cap F_i \neq \emptyset \quad \mathrm{o}\!\!\rightarrow \quad Q.m \ .$$

By delegation, monotony, and disjunction, this implies

$$\{U_0, U_1\} : P.m \quad \wedge \quad (H_0 \cap F_0) \cup (H_1 \cap F_1) \neq \emptyset \quad \mathrm{o}\!\!\rightarrow \quad Q.m \ .$$

Just as in the proof of Theorem A, we have for every process $q \in U_0$ that $pc.q = a$ implies that the number of locked addresses in A_0 is less than $\#U_0$, so that the number of locked addresses in $A_0 \cup A_1$ is less than $\#U_0 + \#U_1$, and hence less than $\#H_0 + \#H_1$. Therefore $pc.q = a$ implies that $(H_0 \cap F_0) \cup (H_1 \cap F_1)$ is nonempty. Using the PSP-rule we obtain

$$\{U_0\} : P.m \quad \mathrm{o}\!\!\rightarrow \quad Q.m \quad \vee \quad (P.m \ \wedge \ (H_0 \cap F_0) \cup (H_1 \cap F_1) \neq \emptyset) \ .$$

We finally use monotony and transitivity to get $\{U_0, U_1\} : P.m \ \mathrm{o}\!\!\rightarrow Q.m$. □

Remark. The generalization to an arbitrary number of triples A, U, *lock*, is straightforward. □

3 Collector processes

We now describe the collector processes to establish formula (1). Each collector q uses the function $nexti(q, ?)$ to traverse the set *Cli* of the clients and attempts to provide client $sg.q$ with the new address $yg.q$ in command 77. The collector inspects the addresses by means of a function $nexta(q, ?)$, which traverses Ad.

We introduce the new shared variables

$$\begin{array}{l} lock0, open, isnx : \textbf{array } \ Ad0 \ \textbf{ of } \ Bool \ ; \\ ispo : \textbf{array } \ Lo \ \textbf{ of } \ Bool \ . \end{array}$$

For each separate address the collectors work under mutual exclusion by means of consensus variables *lock0*, as analysed in Section 2. The flag *open* is used to mark addresses ready for recycling and not yet used. This flag is raised at 78 and lowered at 75 just before recycling is attempted. If address *yg* is not open, it is tested whether it is in use by the object or by linearizers or appliers. If not, its *nx* pointer can be reset to zero, and the address is tested whether it is in use by a client or whether it has an address pointing to it. If all these tests have been passed, its *post* pointer is reset to zero and the address can be used for recycling. The flag $isnx.z$ (or $ispo.u$) is used to indicate the potential existence of an address k with $nx.k = z$ (or $post.k = u$). We thus arrive at the following form of garbage collection

Collector

```
59      yg := nexta(self, yg) ;
60      if lock0.yg then goto 59 else lock0.yg := true fi ;
```

61 **if** $open.yg$ **then goto** 75 **fi** ;

62 **if** $usob.yg$ **then goto** 79 **fi** ;

63 **if** $cnt.yg \neq 0$ **then goto** 79 **fi** ;

64 $zg := nx.yg$; { decouple zg }

65 $nx.yg := 0$;

66 $isnx.zg := false$;

67 **if** $isil.yg$ **then goto** 79 **fi** ;

68 **if** $isnx.yg$ **then goto** 79 **fi** ;

69 $usob.yg := true$;

70 $ug := post.yg$; { decouple ug }

71 $post.yg := 0$;

72 $ispo.ug := false$;

73 $isil.yg := true$;

74 $isnx.yg := true$;

75 $open.yg := false$;

76 $sg := nexti(self, sg)$;

77 **if** $iloc.sg = 0$ **then** { recycle yg }
  $iloc.sg := yg$; $own.yg := sg$; **goto** 79 **fi** ;

78 $open.yg := true$;

79 $lock0.yg := false$; **goto** 59 .

Memory leakage means that addresses are no longer needed for the execution and yet cannot be reclaimed by the garbage collectors. The absence of memory leakage therefore means that the flags *usob*, *isil*, and *isnx* are never unnecessarily high. This is expressed as follows. For every $k \in Ad$, we claim the invariants

(Mq0) $\neg usob.k \;\vee\; (\exists\, q \in Coll :: 69 < pc.q \leq 78 \;\wedge\; yg.q = k)$
$\vee\; open.k \;\vee\; (own.k \neq 0 \;\wedge\; post.k = 0)$
$\vee\; k \in \{nx.staHead, staHead\}$
$\vee\; (\exists\, q \in App :: 53 < pc.q \leq 55 \;\wedge\; y.q = k)$;

(Mq1) $\neg isil.k \;\vee\; post.k = 0 \;\vee\; (\exists\, q \in Cli :: 23 < pc.q \leq 27 \;\wedge\; i.q = k)$;

(Mq3) $\neg isnx.k \;\vee\; post.k = 0 \;\vee\; (\exists\, m \in Ad :: nx.m = k)$
$\vee\; (\exists\, q \in Coll :: pc.q = 66 \;\wedge\; zg.q = k)$.

4 The progress for invocation addresses

We turn to the proof of claim (1) that *true* leads to $iloc.q0 \neq 0$ under the actions of collectors. Claim (1) can only be proved if the set Ad is large enough in comparison with the sets of processes. In order to express such bounds, we define nA and nL to be the number of elements of the sets Ad and Lo. We define pI, pL, pA, pC, pD to be the number of processes in Cli, Lin, App, $Coll$, and $Distr$, respectively.

We use Theorem A of section 2. For this purpose, we provide the collectors with private ghost variables gs that, roughly speaking, stand for the number of steps of sg needed to reach $q0$. Ghost variable gs is only modified at command 77 according to

$$\textbf{if}\ \ gs = 0\ \ \textbf{then}\ \ gs := pI - 1\ \ \textbf{else}\ \ gs := gs - 1\ \ \textbf{fi}\ .$$

It is initialized in such a way that we have the invariants $0 \leq gs.q < pI$ and

$$gs.q = 0 \quad \wedge \quad pc.q = 77 \quad \Rightarrow \quad sg.q = q0\ .$$

The main property of gs is that every execution of command 77 verifies the postcondition $iloc.q0 \neq 0$ or decrements the variant function

$$vf0 \quad = \quad (\sum q \in Coll :: gs.q)\ .$$

This is the starting point for the construction of a variant function vf.

To approach 77, a collector q needs to find a suitable address. In order to express suitability of an address, we define the state functions $using.k$ and $psopen.k$ (pseudo-open) for addresss k, given by

$$\begin{array}{l} using.k \quad \equiv \quad (\exists\, q :: 69 < pc.q \leq 78 \quad \wedge \quad yg.q = k)\ ; \\ psopen.k \quad \equiv \quad \neg usob.k \quad \wedge \quad \neg exla.k \quad \wedge \quad \neg isil.k \quad \wedge \quad \neg isnx.k\ , \\ \text{where}\ \ exla.k \equiv (\exists\, q : 29 < pc.q \leq 38\ \vee\ 44 < pc.q \leq 57\ :\ y.q = k)\ . \end{array}$$

We would like to use Theorem A with the set $Hot0$ given by

$$Hot0 \quad = \quad \{k \in Ad \mid open.k\ \vee\ psopen.k\ \vee\ using.k\,\}\ .$$

Using old invariants one can prove that $k \in Hot0$ is only invalidated when some collector q executes step 77 with $yg.q = k$ and $iloc.(sg.q) = 0$.

Unfortunately, we have no way to guarantee that $Hot0$ contains enough elements. The point is that, although we can find upper bounds for the number of addresses k with $usob.k$ or $exla.k$ or $isil.k$, we cannot bound the number of addresses k with $isnx.k$. We therefore extend Hot with addresses that are used to make $isnx$ false. So we define Hot as the union of $Hot0$ and $Hot1$ where $Hot1$ is the subset of Ad given by

$$\begin{array}{rl} k \in Hot1 \quad \equiv & (nx.k \neq 0 \quad \wedge \quad \neg usob.k \quad \wedge \quad \neg exla.k) \\ & \vee \quad (\exists\, q :: pc.q = 66 \quad \wedge \quad yg.q = k \quad \wedge \quad zg.q \neq 0)\ . \end{array}$$

Using old invariants one can prove that $k \in Hot1$ is only invalidated when some collector q executes 66 with $yg.q = k$ and $zg.q \neq 0$, and thus decrements the variant function

$$vf1 \quad = \quad \#\{k \mid isnx.k\}\ .$$

Before defining the main variant function, we treat the invariant $\#Coll \leq \#Hot$, which is needed to apply Theorem A.

In order to estimate $\#Hot$, we investigate the complement of Hot. For any address $k \in Ad$, we prove using (a.o.) the invariants claimed in Section 3 that

$$(*) \qquad \begin{array}{rl} k \notin Hot \quad \Rightarrow \quad exla.k \quad \vee \quad k \in \{staHead, nx.staHead\} & \\ \vee \quad (\exists\, q \in Cli :: iloc.q = k) \quad \vee \quad k \in X \cup Y\ , & \end{array}$$

where X and Y are subsets of Ad defined by

$$k \in X \equiv nx.k = 0 \ \wedge \ (\exists\, m :: nx.m = k) \ ,$$
$$k \in Y \equiv nx.k = 0 \ \wedge \ ((\exists\, q \in \mathit{Cli} :: i.q = k) \vee (\exists\, q \in \mathit{Coll} :: zg.q = k)).$$

In order to find a bound for $\#(X \cup Y)$, we write $Z = \{k \mid nx.k \neq 0\}$ and observe $\#(X \cup Y) + \#Z \leq nA$ and $\#X \leq \#Z$. We also have $\#Y \leq pI + pC$. This implies

$$2 \cdot \#(X \cup Y) \leq \#(X \cup Y) + \#Z + \#Y \leq nA + pI + pC \ .$$

Since the number of addresses k that satisfy one of the first three disjuncts of the consequent of (*) is bounded by $pL + pA + 2 + pI$, we get

$$\#(Ad \setminus \mathit{Hot}) \ \leq \ pL + pA + 2 + pI + \tfrac{1}{2} \cdot (nA + pI + pC) \ .$$

Since $\#(Ad \setminus \mathit{Hot}) = nA - \#\mathit{Hot}$, this implies

$$nA \ \leq \ 4 + 3 \cdot pI + 2 \cdot pL + 2 \cdot pA + pC + 2 \cdot \#\mathit{Hot} \ .$$

Therefore, in order to guarantee the required invariant $\#\mathit{Coll} \leq \#\mathit{Hot}$, it suffices to postulate

$$nA \ \geq \ 4 + 3 \cdot pI + 2 \cdot pL + 2 \cdot pA + 3 \cdot pC \ .$$

We turn to the construction of the variant function. We use the constituents *vf0* and *vf1* defined above. Function *vf0* is only incremented when the postcondition Q holds or is established. Otherwise it is decremented at instruction 77. Function *vf1* is decremented at 66 if $zg.q \neq 0$. It is only incremented at 74. We therefore define the variant function *vf* by

$$\mathit{vf} \quad = \quad 2 \cdot \mathit{vf0} + \mathit{vf1} + \#(\mathit{Coll} \setminus \mathit{Prom}) + \#(\mathit{Coll} \setminus \mathit{Fin})$$

where *Prom* and *Fin* are the subsets of *Coll* given by

$$\begin{array}{lcl} q \in \mathit{Prom} & \equiv & (60 < pc.q \leq 77 \ \wedge \ yg.q \in \mathit{Hot0}) \\ & \vee & (60 < pc.q \leq 66 \ \wedge \ yg.q \in \mathit{Hot1}) \ ; \\ q \in \mathit{Fin} & \equiv & 74 < pc.q \leq 77 \ . \end{array}$$

We now apply Theorem A with $P := \mathit{true}$ and $Q := \mathit{iloc}.q0 \neq 0$ and $U := \mathit{Coll}$. We have to verify several conditions. Firstly, it is clear that the theory of Section 2 is applicable and that *vf* is always a natural number. Furthermore, condition (a) of Theorem A holds trivially.

The property $q \in \mathit{Prom}$ is only invalidated when q executes 77, or when $yg.q \notin \mathit{Hot0}$ and q executes 66. The property $q \in \mathit{Fin}$ is only invalidated when q executes 77. In this way one proves that condition (b) of Theorem A holds. When a collector visits an address in *Hot*, it enters *Prom* and decrements *vf* in that way. Since addresses are only removed from *Hot* by visiting collectors, this shows that condition (c) holds. So Theorem A is applicable. Finally, since *vf* is bounded from above, we conclude $\{\mathit{Coll}\} : \ \mathit{true} \circ\!\!\rightarrow \mathit{iloc}.q0 \neq 0$, which is formula (1).

5 Progress for state locations

We finally introduce processes to establish formula (2), in words: to distribute locations or to give *staloc* adequate nonzero values. The procedure is similar as for invocation addresses, but simpler since some work has been done in Collector. The proof will be based on Theorem B of Section 2, so we start by introducing a shared array of boolean consensus variables:

$$lock1 : \textbf{array}\ \ Lo\ \ \textbf{of}\ \ Bool\ .$$

We use the flags *ispo*, which were introduced in the previous section to indicate that a location may contain a state value. In this way, we arrive at

Distributor
90 $\quad ug := nextu(self, ug)$;
91 $\quad$ **if** $lock1.ug$ **then goto** 90 **else** $lock1.ug := true$ **fi** ;
92 $\quad$ **if** $ispo.ug$ **then goto** 96 **fi** ;
93 $\quad ispo.ug := true$;
94 $\quad sg := nexta(self, sg)$;
95 $\quad$ **if** $staloc.sg = 0$ **then** $staloc.sg := ug$ **else goto** 94 **fi** ;
96 $\quad lock1.ug := false$; **goto** 90 .

Here $nextu(q, ?)$ and $nexta(q, ?)$ traverse the sets Lo and App, respectively, see Section 2. Indeed, every distributor must serve every applier, since it may be that there remains only one living distributor and only one living applier, and then the remaining distributor must serve the remaining applier.

The absence of memory leakage for locations is expressed by the invariant

$$\begin{array}{ll}(\text{Mq4}) & \neg ispo.u \ \ \vee\ \ (\exists\, q \in Distr :: \ 93 < pc.q \leq 95 \ \wedge\ ug.q = u) \\ & \vee\ \ (\exists\, q \in App :: \ staloc.q = u) \ \ \vee\ \ (\exists\, k \in Ad :: \ post.k = u) \\ & \vee\ \ (\exists\, q \in Coll :: \ pc.q = 72 \ \wedge\ ug.q = u)\ .\end{array}$$

We turn to the verification of claim (2). Here we need Theorem B of Section 2. We introduce private ghost variables gs for the distributors which are only modified at 95, according to

$$\textbf{if}\ \ gs = 0\ \ \textbf{then}\ \ gs := pA - 1\ \ \textbf{else}\ \ gs := gs - 1\ \ \textbf{fi}\ .$$

They are initialized in such a way that we have the invariants $0 \leq gs.q < pA$ and

$$gs.q = 0 \ \ \wedge\ \ pc.q = 95 \ \ \Rightarrow\ \ sg.q = q0\ .$$

The main property of gs is that every execution of command 95 verifies the postcondition $staloc.q0 \neq 0$ or decrements the variant function

$$wf0 \ \ = \ \ (\textstyle\sum q \in Coll :: gs.q)\ .$$

We introduce a set $HotL$ of locations given by

$$u \in HotL \ \ \equiv\ \ \neg ispo.u \ \ \vee\ \ (\exists\, q :: 92 < pc.q \leq 95 \ \wedge\ ug.q = u)\ .$$

A location can only leave HotL when it becomes the new location staloc of some applier. In order to determine $\#\mathit{HotL}$, we investigate the complement of HotL.

Using (Mq4), it is easy to prove that

$$nL \;\leq\; \#\mathit{HotL} + pA + \#\mathit{Loc1} + pC \;,$$

where $\mathit{Loc1}$ is the subset of Lo defined by

$$u \in \mathit{Loc1} \quad\equiv\quad (\exists\, k \in \mathit{Ad} :: \mathit{post}.k = u) \;.$$

We now observe that $\#\mathit{Loc1} \leq \#\mathit{Ad1}$ where $\mathit{Ad1} = \{k \mid \mathit{post}.k \neq 0\}$. A location $\mathit{post}.k \neq 0$ can be freed by the collectors if $k \in \mathit{HotA}$ where $\mathit{HotA} \subseteq \mathit{Ad}$ is given by

$$k \in \mathit{HotA} \equiv \mathit{psopen}.k \;\vee\; (\exists\, q \in \mathit{Coll} :: 69 < pc.q \leq 74 \;\wedge\; yg.q = k) \;.$$

We also need the set $\mathit{Hot1}$ defined in Section 4, and define $\mathit{HotC} = \mathit{HotA} \cup \mathit{Hot1}$. Using old invariants, one proves that, for every address k, we have

$$\begin{array}{ll} & k \in \mathit{Ad1} \quad\wedge\quad k \notin \mathit{HotC} \\ \Rightarrow & k \in \{\mathit{staHead}, \mathit{nx}.\mathit{staHead}\} \quad\vee\quad \mathit{exla}.k \quad\vee\quad k \in X \cup Y \;. \end{array}$$

The above implication yields $\#\mathit{Ad1} \;\leq\; \#\mathit{HotC} + 2 + pL + pA + \frac{1}{2}\cdot(nA + pI + pC)$. Combining this with the previous bounds of nL and $\#\mathit{Loc1}$, we get

$$nL \;\leq\; \#\mathit{HotL} + \#\mathit{HotC} + \tfrac{1}{2}\cdot(nA + pI + pC) + 2 + pL + 2\cdot pA + pC.$$

We therefore postulate $nL \;\geq\; \frac{1}{2}\cdot(nA + pI) + 2 + pL + 2\cdot pA + 2\frac{1}{2}\cdot pC + pD$. This guarantees the invariant $pC + pD \leq \#\mathit{HotL} + \#\mathit{HotC}$, which will enable us to apply Theorem B to the present situation.

We turn to the variant function. For this purpose we first define

$$\mathit{wf1} \quad=\quad \mathit{vf1} + \#(\mathit{Coll} \setminus \mathit{ProL}) + \#\mathit{FinL} \;,$$

where ProL and FinL are the subsets of Coll given by

$$\begin{array}{lll} q \in \mathit{ProL} & \equiv & (60 < pc.q \leq 72 \;\wedge\; yg.q \in \mathit{HotA}) \\ & \vee & (60 < pc.q \leq 66 \;\wedge\; yg.q \in \mathit{Hot1}) \;; \\ q \in \mathit{FinL} & \equiv & 72 < pc.q \leq 74 \;. \end{array}$$

When a collector q visits HotC, it decrements $\mathit{wf1}$ since it enters ProL. If q exits from ProL at 66, it decrements $\mathit{vf1}$, so that $\mathit{wf1}$ remains constant. If q exits ProL at 72, it enters FinL. So then $\mathit{wf1}$ increases with 2. If q increments $\mathit{vf1}$ at 74, it exits from FinL. So then $\mathit{wf1}$ remains constant.

Therefore, we only have to compensate for the increment with 2 at 72. We now define $\mathit{wf2} = \#\{u \mid \mathit{ispo}.u\}$, and indeed it can be proved that $\mathit{wf2}$ decreases with 1 at 72. It clearly can only increase (with 1) at 93. We define

$$\begin{array}{l} \mathit{wf3} \;=\; \#(\mathit{Distr} \setminus \mathit{ProD}) \quad\wedge\quad \mathit{wf4} \;=\; \#(\mathit{Distr} \setminus \mathit{ProE}) \text{ , where} \\ \quad q \in \mathit{ProD} \;\equiv\; 91 < pc.q \leq 93 \quad\wedge\quad \neg\mathit{ispo}.(ug.q) \;; \\ \quad q \in \mathit{ProE} \;\equiv\; 93 < pc.q \leq 95 \;. \end{array}$$

Function *wf3* decreases when a distributor visits a location in *HotL*. It can increase with 1 at 93. Function *wf4* decreases at 93. It increases with 1 at 95, but only if $staloc.(sg.q) = 0$. Finally, we define

$$wf \quad = \quad wf1 + 2 \cdot wf2 + wf3 + 3 \cdot wf4 + 3 \cdot wf0 \ .$$

This function only increases when the postcondition $staloc.q0 \neq 0$ is verified. It decreases whenever a collector visits *HotC* or a distributor visits *HotL*. It also decreases when a distributor jumps backward from 95 to 94. Now Theorem B can be applied to prove formula (2).

6 Conclusion

We thus have proved that the progress assertions (1) and (2) hold if

$$\begin{array}{l} nA \ \geq \ 4 + 3 \cdot pI + 2 \cdot pL + 2 \cdot pA + 3 \cdot pC \ \text{ and} \\ nL \ \geq \ \frac{1}{2} \cdot (nA + pI) + 2 + pL + 2 \cdot pA + 2\frac{1}{2} \cdot pC + pD \ . \end{array}$$

By the results of [Hes99], it follows that $L : pc.q0 = 21 \ \mathrm{o}\!\!\rightarrow pc.q0 = 20$ holds for every client $q0$. The main point of these inequalities is that the lower bounds of $nA = \#Ad$ and $nL = \#Lo$ are explicit, and that they are linear in the number of processes.

References

[Hes96] W.H. Hesselink: Bounded Delay for a Free Address. Acta Informatica **33** (1996) 233–254.

[Hes98] W.H. Hesselink: The design of a linearization of a concurrent data object. In: D. Gries, W.-P. de Roever (eds.): Programming Concepts and Methods, Proceedings Procomet '98, Chapman & Hall, IFIP 1998, pp. 205–224.

[Hes99] W.H. Hesselink: Progress under bounded fairness. Submitted.

[Hes@] W.H. Hesselink: Web site: `http://www.cs.rug.nl/~wim/linproc`